آنالیز الگوریتم‌ها

مرتضی علیمی

دانشگاه صنعتی شریف، بهار ۱۳۹۹

فهرست مطالب

فهرست ۶

پیش‌گفتار ۸

اطلاعات درس ۸

اول جلسات اصلی ۱۵

۱ مقدمه ۱۷

۲ مسئله پیدا کردن نزدیکترین زوج نقطه ۲۵

۳ جستجوی اول سطح ۳۱

۴ جستجوی اول عمق ۳۸

۵ الگوریتم‌های حریصانه ۱ ۴۹

۶ الگوریتم‌های حریصانه ۲ ۵۴

۷ برنامه‌ریزی پویا ۱ ۵۹

۸ برنامه‌ریزی پویا ۲ ۶۶

۹ برنامه‌ریزی پویا ۳ ۷۸

۱۰ کوتاهترین مسیر ۱ ۸۴

۱۱ کوتاهترین مسیر ۲ ۹۱

۱۲ دور با کمترین میانگین وزن و درخت کم‌عمق–سبک ۹۶

۵.۱۲ محاسبه کوتاهترین مسیر در عمل (اختیاری) ۱۰۴

۱۳ جریان بیشینه ۱. مبانی و الگوریتم‌ها ۱۱۵

۱۴ جریان بیشینه ۲. کاربردها، نتایج ترکیبیاتی، فرمول‌بندی‌های متفاوت ۱۳۰

۱۵ جریان بیشینه ۳. کاربردهای بیشتر ۱۴۰

۱۶ تطابق وزن‌دار در گراف دوبخشی، جریان با کمترین هزینه، الگوریتم‌های کارآتر برای جریان بیشینه (اختیاری) ۱۴۸

۱۷ پیچیدگی محاسباتی ۱۵۷

۱۸ مسائل ان‌پی–تمام ۱ ۱۶۸

۱۹ مسائل ان‌پی–تمام ۲ ۱۷۷

۲۰ گشت و گذاری در دنیای پیچیدگی (اختیاری) ۱۸۵

۲۱ جستجوی هوشمندانه ۱۹۷

۲۲ جستجوی محلی ۲۱۳

۲۳ برنامه‌ریزی خطی ۱ ۲۳۸

۲۴ برنامه‌ریزی خطی ۲ ۲۵۰

۲۵ برنامه‌ریزی خطی ۳ ۲۶۰

۲۶ برنامه‌ریزی خطی ۴ ۲۷۰

دوم جلسات اضافه ۲۸۳

۱ الگوریتم‌های برخط ۲۸۵

۲ مسئله خبرگان و روش به‌روزرسانی ضربی وزن‌ها ۲۹۵

۳ الگوریتم‌های تصادفی ۳۰۱

۴ نامساوی‌های مارکوف و چبیشف ۳۱۳

۵ نامساوی چرنوف ۳۲۳

۶ استفاده از چندجمله‌ای‌ها در طراحی الگوریتم ۳۳۲

نمونه‌گیری، مسئله بستار متعدی ۳۴۰

شمارش و نمونه‌گیری تصادفی ۳۴۵

شکستن تقارن، جواب‌های یکتا، لم ایزوله‌سازی ۳۴۹

الگوریتم‌های سریع‌تر برای مسئله زیردرخت فراگیر کمینه ۳۵۸

الگوریتم‌های پارامتر_ثابت ۳۷۸

تجزیه درختی ۳۸۶

الگوریتم‌های تقریبی ۱: روش حریصانه ۴۱۴

الگوریتم‌های تقریبی ۲: روش گرد کردن داده و برنامه‌ریزی پویا ۴۲۵

الگوریتم‌های تقریبی ۳: روش‌های مبتنی بر برنامه‌ریزی خطی ۴۳۳

الگوریتم‌های جویباری ۴۴۰

فشرده‌سازی گراف ۴۵۰

هندسه محاسباتی ۴۵۴

سوم جلسات حل تمرین ۴۶۳

۱ روش تقسیم و حل و جستجوی همه حالت‌ها ۴۶۵

۲ مؤلفه‌های قویاً همبند و رأس‌های برسی ۴۶۹

۳ پیاده‌سازی الگوریتم‌های زیردرخت فراگیر کمینه ۴۷۸

۴ برنامه‌ریزی پویا ۴۸۲

۵ کاربردهایی از مسئله جریان بیشینه و تطابق دوبخشی ۴۸۶

۶ الگوریتم‌های گراف ۴۹۰

۷ تحویل چندجمله‌ای و جریان بیشینه ۴۹۳

۸ مسائل انپی_تمام ۴۹۸

۹ مثال‌های بیشتری از انپی_تمامیت ۵۰۹

۱۰ جستجوی هوشمندانه و جستجوی محلی ۵۱۵

۱۱ الگوریتم‌های برخط ۵۱۷

۱۲ الگوریتم‌های تصادفی ۵۲۳

۱۳ کران‌های مارکوف، چبیشف و چرنوف ۵۲۶

۱۴ مدل‌سازی با برنامه‌های خطی ۵۲۹

۱۵ دوگانی ۵۳۴

۱۶ استفاده از قضیه دوگانی برای اثبات قضایای ترکیباتی ۵۴۰
۱۷ مدل‌سازی مسائل ترکیباتی با برنامه‌های صحیح ۵۴۴

چهارم تمرین‌های هفتگی ۵۵۱

۱ مقدمات ۵۵۳
۲ جستجوی اول سطح ۵۵۴
۳ جستجوی اول عمق ۵۵۵
۴ الگوریتم‌های حریصانه ۵۵۶
۵ برنامه‌ریزی پویا ۵۵۷
۶ کوتاهترین مسیر در گراف‌ها ۵۵۸
۷ جریان بیشینه ۱ ۵۶۰
۸ جریان بیشینه ۲ ۵۶۲
۹ مسائل ان‌پی تمام ۱ ۵۶۳
۱۰ مسائل ان‌پی تمام ۲ ۵۶۴
۱۱ پس‌گرد، جستجوی محلی ۵۶۵
۱۲ برنامه‌ریزی خطی ۵۷۱
۱۳ دوگانی ۵۷۳
۱۴ مدل‌سازی ۵۷۴

پنجم تمرین‌های اضافه ۵۷۵

۱ الگوریتم‌های برخط و مسئله خبرگان ۵۷۷
۲ الگوریتم‌های تصادفی و نامساوی‌های مارکوف و چبیشف ۵۷۸
۳ نامساوی چرنوف و چندجمله‌ای‌ها ۵۷۹

ششم پاسخ تمرین‌های هفتگی ۵۸۱

پاسخ تمرین سری ۱ ۵۸۳
پاسخ تمرین سری ۲ ۵۸۵
پاسخ تمرین سری ۳ ۵۸۸
پاسخ تمرین سری ۴ ۵۹۰
پاسخ تمرین سری ۵ ۵۹۳
پاسخ تمرین سری ۶ ۵۹۶
پاسخ تمرین سری ۷ ۶۰۱
پاسخ تمرین سری ۸ ۶۰۶
پاسخ تمرین سری ۹ ۶۰۸
پاسخ تمرین سری ۱۰ ۶۱۱
پاسخ تمرین سری ۱۲ ۶۱۳
پاسخ تمرین سری ۱۴ ۶۱۷

هفتم پاسخ تمرین‌های اضافه ۶۲۱

پاسخ تمرین اضافه سری ۱ ۶۲۳

پاسخ تمرین اضافه سری ۲ ۶۲۵

پاسخ تمرین اضافه سری ۳ ۶۲۶

هشتم تمرین‌های عملی ۶۲۷

۱ تقسیم و حل، الگوریتم‌های مقدماتی گراف ۶۲۹

۲ برنامه‌ریزی پویا ۶۴۵

۳ کوتاهترین مسیر و جریان بیشینه ۶۶۰

نهم آزمون‌ها ۶۷۵

آزمون میان‌ترم ۶۷۷

پاسخ آزمون میان‌ترم ۶۷۸

آزمون پایان‌ترم ۶۸۰

پاسخ آزمون پایان‌ترم ۶۸۲

آزمونک اختیاری ۶۸۶

پاسخ آزمونک اختیاری ۶۸۷

پیش‌گفتار

این درسنامه شامل تمامی مستندات مربوط به درس آنالیز الگوریتم‌ها به انضمام مطالب اضافه‌ای است که در نیم‌سال دوم سال تحصیلی ۹۹–۱۳۹۸ در دانشکده ریاضی دانشگاه صنعتی شریف ارائه شده است.

مطالبی که در این درسنامه آمده است بیشتر از یک درس معمولی الگوریتم است؛ در واقع تلاش من بر این بوده است که در حد توانم محتوای جامعی فراهم بشود که دانستنش برای یک دانشجوی علوم کامپیوتر مفید و حتی در برخی موارد ضروری است.

اکثریت مطالب این درسنامه (حتی پاسخ‌های تمرین‌ها) از لحاظ صحت علمی بررسی نشده‌اند و بنابراین احتمالاً حاوی ایرادهای فراوان هستند. اما امیدوارم حتی با صورت فعلی بتواند مورد استفاده دانشجویان علاقمند قرار گیرد.

مکتوب شدن مطالب درس حاصل همکاری جمعی تیم ارائه کننده درس و دانشجویان درس است. از همگی عزیزان به خاطر زحماتشان صمیمانه تشکر می‌کنم. به‌طور خاص علاقمندم از دوست عزیزم آقای هانی احمدزاده که جلسات مربوط به برنامه‌ریزی خطی را ارائه کردند، دستیار ارشد درس آقای علیرضا توفیقی، و مسئول جزوه‌ها خانم الهه قاسمی سپاسگزاری کنم. تمرین‌های نظری و عملی درس مرهون تلاش تیم دستیاران درس است. از همگی این عزیزان متشکرم. همچنین از دانشجویانی که با دقت نظر و صرف زمان در بالابردن کیفیت و غنای درسنامه‌ها نقش داشتند ــ به‌خصوص چند نفر از دانشجویان که در این زمینه نقش پررنگ‌تری داشتند (که تاثیر کارشان در درسنامه مشخص است) قدردانی می‌کنم.

در حال حاضر زحمت مدیریت و نگهداری این درسنامه به عهده آقای علیرضا دادگرنیا است. برای مطرح کردن بازخوردهایتان یا ایرادهایی که می‌بینید می‌توانید از طریق آدرس alirezadadgarnia1378@gmail.com با ایشان مکاتبه کنید.

ویدیوهای درس و آخرین نسخه از درسنامه را می‌توانید در اینجا بیابید.

مرتضی علیمی

مرداد ۱۳۹۹

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

اطلاعات درس

۱ هدف درس

هدف از درس آنالیز الگوریتم‌ها آشنایی با روش‌ها و مفاهیم بنیادین مرتبط با طراحی الگوریتم‌ها، و برخی الگوریتم‌های کلاسیک و پرکاربرد است. انتظار می‌رود دانشجو در پایان درس بتواند از تکنیک‌های گفته شده برای طراحی الگوریتم‌های جدید استفاده کند.

۲ سرفصل تقریبی

- مباحث کلاسیک
  - آشنایی با مدل‌های محاسباتی، روش‌های تحلیل الگوریتم‌ها، ملاک‌های کارآیی الگوریتم‌ها.
  - روش تقسیم و حل.
  - روش جستجوی همه حالت‌ها.
  - روش حریصانه.
  - برنامه‌ریزی پویا.
  - الگوریتم‌های کلاسیک گراف‌ها: پیمایش گراف‌ها، پیدا کردن مولفه‌های همبندی، الگوریتم‌های کوتاه‌ترین مسیر، شبکه جریان.
  - آشنایی با روش‌های نشان دادن سختی مسائل و مفهوم انپی‌ـ تمامیت. روش‌های مهم برای حل مسائل سخت.
  - برنامه‌ریزی خطی.
- مباحث اضافه
  - مباحثی در الگوریتم‌های تصادفی.
  - الگوریتم‌های جویباری.
  - الگوریتم‌های برخط.
  - روش به‌روزرسانی ضربی وزن‌ها.

۳ بارم‌بندی و تقویم درس

- میان‌ترم: ۶ نمره.
  پنجشنبه ۴ اردیبهشت صبح یا بعدازظهر.

- پایان‌ترم: ۸ نمره.
  سه‌شنبه ۲۷ خرداد ساعت ۹.

- تمرین‌های نظری: ۳ نمره.
  تمرین‌ها به صورت هفتگی خواهند بود؛ هر سری تمرین سه‌شنبه شب آپلود می‌شود و موعد تحویل آن ۱۲ ظهر سه‌شنبه هفته بعد است. برای هر دانشجو صرفاً ۱۰ سری تمرینی که بیشترین نمره را در آن‌ها کسب کرده لحاظ خواهد شد.

- تمرین‌های عملی: ۳ نمره.

  ۱. سری اول: از ۲ اسفند تا ۲۳ اسفند.

  ۲. سری دوم: از ۲۳ اسفند تا ۲۲ فروردین.

  ۳. سری سوم: از ۲۲ فروردین تا ۱۲ اردیبهشت.

  هر دانشجو می‌تواند تمرین‌های عملی خود را در مجموع با ۳ روز تأخیر تحویل دهد.

- جزوه‌نویسی: ۱ نمره اضافه.
  جزوه‌ای که می‌نویسید را باید در ۲ مرحله بفرستید ـ نسخه اولیه: حداکثر ۳۳ ساعت بعد از کلاس، نسخه نهایی: یک هفته بعد از مهلت فرستادن نسخه اولیه. برای جزوه نوشتن از قالب منتشر شده در سایت درس استفاده کنید. نمره بالا به جزوه‌هایی تعلق می‌گیرد که از لحاظ جامعیت و پوشش مطالب گفته شده در کلاس، درستی فنی، نحوه بیان، و اصول صفحه‌بندی کیفیت بالایی داشته باشند. برای نوشتن جزوه یک جلسه، لازم است آن جلسه را در کلاس شرکت کنید. همچنین ننوشتن جزوه جلسه‌ای که آن را رزرو کرده‌اید ناپسند است و بی‌احترامی به سایر دانشجویان و تیم ارائه دهنده درس محسوب می‌شود.

  تلاش می‌کنیم جزوه‌ها را در اسرع وقت روی سایت درس آپلود کنیم، اما توجه کنید که جزوه‌های آپلود شده توسط مدرس درس بررسی نشده‌اند.

  برای جزوه‌نویسی با خانم قاسمی هماهنگ کنید.

- کوئیز از مباحث اضافه: ۱ نمره اضافه.
  چهارشنبه ۱۱ تیر ساعت ۱۱.

توجه کنید که:

- شرط لازم برای گذراندن درس کسب ۷ نمره از میان‌ترم و پایان‌ترم است.

- در صورتی که نمره پایان‌ترمتان بیشتر از نمره میان‌ترمتان باشد، بارم‌های این دو امتحان به ۴ و ۱۰ تغییر خواهد کرد.

- میان ترم و پایان‌ترم صرفاً از مباحث کلاسیک خواهد بود (تقریباً تا آخر جلسه ۲۵).

- در روز پنجشنبه ۱۵ اسفند ساعت ۹ تا ۱۲:۳۰ دو جلسه جبرانی برگزار خواهد شد. در مورد کلاس جبرانی جلسه ۲۷ اسفند بعداً اطلاع‌رسانی خواهد شد.
- زمان کلاس تمرین دوشنبه‌ها ساعت ۵ تا ۷ خواهد بود.

۴ نقش کلاس و منبع درس

نقش کلاس درس ارائه تصویر کلی از هر مبحث درسی و گشودن گره‌های مربوط به مباحث پیچیده‌تر است. هدف برگزاری کلاس این است که دانشجو بعد از کلاس جهت‌گیری ذهنی خوبی نسبت به مبحث مورد نظر داشته باشد، به طوری که بتواند با مطالعه مبحث مربوطه از منبع درسی، جزئیات را به طور کامل متوجه شود. هدف دیگر، مطرح کردن برخی مطالب است که به صورت سرراست در منابع وجود ندارد. به طور خاص توجه کنید که برخی مباحث استاندارد ممکن است به گونه‌ای متفاوت با چیزی که انتظار دارید در کلاس ارائه شوند. دانشجویان مسئول دانستن مطالب گفته شده در کلاس درس، کلاس حل‌تمرین، و منابع هر جلسه می‌باشند.

اکیداً توصیه می‌شود دانشجو در جلسات درس شرکت کند. کلاس درس حدود ۱۳:۳۲ شروع می‌شود و حدود ۱۴:۵۲ پایان می‌یابد. همچنین توصیه می‌شود تا ۴۸ ساعت بعد از هر جلسه، حدود ۲ برابر زمان کلاس را صرف مرور و بازنویسی مطالب گفته شده در کلاس کنید.

۵ نقش کلاس حل تمرین

مشارکت در کلاس درسی و به طور خاص مطرح کردن و پرسیدن نکات مبهم، بسیار پسندیده است و توصیه می‌شود. اما ماهیت کلاس و سرفصل استانداردی که لازم است پوشش داده شود حجم تعامل را محدود می‌کند. نقش جلسات حل تمرین دادن فرصت بیشتر به دانشجویان برای اندیشیدن و تامل کردن در مورد ایده‌ها و مباحث مطرح شده در کلاس، بررسی روش‌ها و ایده‌های بدیل و احیاناً نادرست برای رسیدن به نتایج یکسان، و دیدن مثال‌ها و تمرین‌های مرتبط است. بنابراین دانشجویان در این جلسات نقش پررنگ‌تر و فعال‌تری نسبت به کلاس درسی دارند.

۶ استانداردهای اخلاقی و آکادمیک

تلاش ما در این درس این است که استانداردهای اخلاقی و آکادمیک را تا جای ممکن رعایت کنیم. بنابراین بسیار مهم هست که دانشجویان در همه موارد ادب لازم ــ به معنای اعم کلمه ــ را نسبت به درس، سایر دانشجویان درس، و تیم ارائه کننده درس رعایت کنند. از مصادیق این امر شرکت در جلسات درس، عدم صحبت کردن سر کلاس با دانشجویان دیگر، عدم استفاده از وسایل الکترونیکی جز برای دسترسی به منابعی که مستقیماً به محتوای فعلی در حال ارائه مرتبط است (یا موارد مطلقاً ضروری)، ورود و خروج به موقع به کلاس، نشستن در جای مناسب و رعایت نظم کلاس است. متقابلاً مدرس و دستیاران درس حداکثر تلاششان را برای رعایت ادب نسبت به درس و دانشجویان درس انجام خواهند داد.

انتظار می‌رود که تمرین‌های تحویل داده شده توسط هر دانشجو کار خود دانشجو باشد. در مورد تمرین‌های عملی، بحث کردن در مورد ایده‌های حل مسئله با سایر دانشجویان درس مجاز است (و حتی توصیه می‌شود)، اما دیدن یا آگاهی پیدا کردن از جزئیات راه‌حل سایر دانشجویان مجاز نیست. بنابراین قبل از موعد تحویل تمرین نباید حتی به بخشی از کد همکلاسیتان نگاه کنید (چه به صورت تایپ شده چه احیاناً به صورت شبه‌کد روی کاغذ). طبعاً کمک به اشکال‌زدایی برنامه همکلاسی هم مجاز نیست. همچنین نباید جزئیات پیاده‌سازی خود را برای همکلاسیتان بازگو کنید یا امکان دیده شدن کدتان توسط همکلاسی را فراهم کنید. ابزار اصلی ما برای تشخیص تخلف‌ها در تمرین‌های عملی، امکان سایت کوئرا برای پیدا کردن مشابهت کد است. همچنین ممکن است بعد از پایان موعد تمرین‌های عملی، یک آزمون عملی از تمرین‌های داده شده گرفته شود تا تشخیص دهیم آیا تمرین‌ها کار خود دانشجو است یا خیر.

در تمرین‌های نظری نیز می‌توانید در مورد ایده‌های حل سوال با همکلاسی خود مشورت کنید، اما مجاز به نگاه کردن به راه‌حل هم‌کلاسی و نشان دادن راه‌حل خود به هم‌کلاسی نیستید. همچنین لازم است نام هر کدام از دوستانتان را که برای حل تمرین با آنها مشورت کرده‌اید در ابتدای پاسخ تمرینتان بنویسید. این قاعده برای محافظت از شماست و شانس اینکه کار شما تخطی از استانداردهای آکادمیک تشخیص داده شود را کاهش می‌دهد. ممکن است بعد از موعد تحویل هر سری تمرین، یک کوئیز از سوال‌های تمرین سر کلاس گرفته شود. این کوئیزها نمره‌ای ندارد، اما اگر جواب درست را برای تمرینی تحویل داده باشید و نتوانید در کوئیز، جواب درست را برای همان تمرین بنویسید، این موضوع به عنوان شاهدی بر تخلف در نگارش تمرین‌ها تلقی خواهد شد.

همچنین استفاده از راه‌حل‌های آماده، مثلاً در اینترنت، مجاز نیست. در صورتی که از منبعی استفاده می‌کنید، لازم است آن را در ابتدای پاسخ تمرینتان ذکر کنید تا از خودتان در برابر تشخیص داده شدن به عنوان متخلف محافظت کنید.

با موارد نقض قوانین گفته شده در مورد تمرین‌ها به طور جدی برخورد خواهد شد. مجازات نقض این قوانین، می‌تواند شامل صفر گرفتن در کل سری تمرین مورد نظر (و نه فقط سوال شامل تخلف)، صفر گرفتن در کل تمرین‌های نظری و عملی، و از دست دادن هرگونه نمره اضافه (چه ۲ نمره اضافه اعلام شده و چه هر نمره احتمالی اعلام نشده دیگر) باشد.

در صورتی که تردید داشتید کار شما تخلف از قواعد درس محسوب می‌شود یا خیر، لازم است در این مورد با تیم ارائه دهنده درس مشورت کنید. همچنین در صورتی که تخلفی کردید، بهترین کار این است که داوطلبانه این موضوع را به ما اعلام کنید؛ در این صورت نسبت به حالتی که ما تخلف شما را کشف کنیم برخورد بسیار نرم‌تری با شما خواهد شد.

توجه کنید که بررسی تخلف‌ها احتمالاً به آخر ترم موکول خواهد شد. بنابراین در صورت تخلف ممکن است نمره اولیه‌ای که دریافت می‌کنید نمره کامل باشد و حتی در اواخر ترم هم نمره کاملی برای شما توسط دستیاران درس اعلام شود و ناگهان تخلف شما اعلام شود و کل نمره تمرین‌ها و نمره‌های اضافه را از دست بدهید.

۷ در مورد آزمون‌ها

هدف کلاسیک آزمون‌ها سنجش میزان توانایی فکری دانشجو در مورد مبحث درس است. این موضوع شامل دانستن مطالب درس و توانایی به کاربستن آنها برای حل مسائل جدید است.

اما آزمون‌ها می‌توانند علاوه بر ارزشیابی، ادامه فرایند یادگیری مطالب درس هم باشند. به همین منظور تلاش خواهیم کرد که بعد از میان‌ترم و پایان‌ترم پاسخ یا حداقل راهنمایی برای سوال‌ها را منتشر کنیم.

همچنین تلاش می‌کنیم که تصحیح برگه‌ها تا حد امکان دقیق باشد. برای راه‌حل‌های نوشته شده، علاوه بر دانستن ایده درست، نحوه نوشتن راه‌حل هم نقش مهمی در میزان نمره‌ای که می‌گیرید دارد. اگر ایده درستی برای راه‌حل داشته باشید، اما به طور ناقص یا گنگ بیانش کنید، نمره کامل نخواهید گرفت.

این موضوع هم به خاطر آگاه کردن دانشجویان از اشتباهات و نواقص احتمالی راه‌حل‌هایشان و کمک به پرورش ذهن دقیق‌تر و بیان شفاف‌تر است و هم به خاطر رعایت عدالت و افتراق با دانشجویانی که راه‌حلشان را دقیق‌تر و کامل‌تر نوشته‌اند.

در همین راستا راه‌حل‌هایی که ایده‌شان از اساس نادرست باشد و به راه‌حل درست سوال منجر نشود نمره‌ای نخواهند گرفت. به عبارت دیگر نمره‌دهی بر اساس میزان درستی راه‌حل و درجه وضوح و دقت بیان آن انجام می‌شود، و نه حجم راه‌حل.

۸ نکاتی برای موفقیت در درس

کلیدی‌ترین نکته برای موفقیت در درس، وقت گذاشتن و تلاش مداوم در طول ترم است. به درست آوردن توانایی طراحی الگوریتم‌ها و پرورش یافتن تفکر الگوریتمی پروسه‌ای زمان‌بر است، و بنابراین استراتژی وقت گذاشتن زیاد مقطعی (مثلاً در بازه نزدیک به امتحان‌ها) نتایج درخشانی تولید نخواهد کرد.

نکته مهم دیگر جدی گرفتن تأثیر «نوشتن» است. نوشتن مطالبی که یاد می‌گیرید به تحکیم و تعمیق مطالب کمک شایانی می‌کند. همچنین زمانی که تمرین حل می‌کنید، تلاش به نوشتن راه‌حلتان به دقیق‌تر کردن و فهمیدن ایرادهای احتمالی آن کمک می‌کند.

گرفتن بازخورد از دیگران نیز برای یادگیری مطالب بسیار مفید است. به همین دلیل توصیه می‌شود که حداقل بخشی از درس خواندنتان را به همراه دوستانتان انجام دهید. همچنین از زمان‌های تعیین شده برای حل تمرین و رفع اشکال استفاده کنید.

هنگام توجه به هرگونه راهنما و توصیه برای نحوه یادگیری درس (یا هر کار دیگر)، در نظر گرفتن تفاوت‌های فردی اهمیت بسیار دارد. مثلاً بعضی از دانشجویان هستند که در حضور دیگران و در یک محیط تعاملی و گروهی یادگیری بهتری دارند و برخی دیگر در تنهایی بهتر یاد می‌گیرند. به طور خاص بعضی، سر کلاس بهتر مطالب را یاد می‌گیرند و بعضی ارتباط کمتری برقرار می‌کنند و نیاز بیشتری دارند که مطالب را خودشان بخوانند (بخشی از این مسئله به خاطر شخصیت متفاوت دانشجویان، و بخشی به خاطر شخصیت متفاوت اساتید است). فعال‌تر بودن در جمع هرچند مثبت است، لزوماً به معنی باهوش‌تر بودن یا دانشمندتر بودن نیست.

سعی کنید روش‌هایی که برای شخص شما بهتر کار می‌کنند را بشناسید و از آن‌ها بیشتر استفاده کنید و در عین حال از سایر روش‌ها هم غافل نشوید. برای مثال اگر از زمره دانشجویانی هستید که از کلاس استفاده کمتری می‌برند، تمرکز بیشتری روی مطالعه شخصی خودتان بگذارید، ولی حضور در کلاس را به هیچ عنوان ترک نکنید. طبعاً این موضوع جدای از دادن بازخورد به تیم ارائه دهنده درس برای اصلاح نواقصشان است.

شاید بتوان گفت بهترین معیار برای سنجش میزان پیشرفت خودتان در درس، انرژی ذهنی‌ای است که مصرف می‌کنید. به این دلیل فکر کردن روی مسائل مختلف (چه نکات پوشش داده شده در کلاس و چه مسائل جدید)، لازمه یادگیری است. مطالعه کتاب درسی و توجه به حرف‌های مدرس لازم است و جهت‌دهی ذهنی لازم برای استفاده مفید از زمانی که برای فکر کردن می‌گذارید را به شما می‌دهد، اما مواجهه انفعالی با مطالب صرفاً می‌تواند پیش‌زمینه یادگیری مطالب باشد و یادگیری عمیق نیازمند درگیر شدن با مباحث و صرف انرژی فکری است.

آخرین نکته که اهمیتش از بقیه نکات کمتر نیست: سعی کنید از درس لذت ببرید! ما در حد توانمان تلاش خواهیم کرد که درس برای افراد علاقمند جذاب باشد و فرصت خوبی برای یادگیری مطالب جالب فراهم شود، اما محدودیت‌ها و ضعف‌های جدی‌ای که لاجرم در دانش، تجربه، شخصیت و وقت مدرس درس وجود دارد باعث می‌شود که بخش مهمی از وظیفه ایجاد تجربه لذت‌بخش آموزش به عهده خود دانشجویان باشد.

برخی منابع درس

[CLRS] Cormen, Liserson, Rivest, Stein “Introduction to Algorithms”, 3rd edition, 2009.

[KT] Kleinberg, Tardos, “Algorithm Design”, 2005.

[DPV] Dasgupta, Papadimitriou, Vazirani, “Algorithms”, 2006.

[Erickson] Jeff Erickson, “Algorithms”, 2019.

[Creative] Udi Manber, “Introduction to Algorithms, A Creative Approach”, 1989.

[Berkeley] https://cs170.org/

[MIT1] https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-a

[MIT2] https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analy

[CMU] http://www.cs.cmu.edu/afs/cs/academic/class/15451-f19/www/index.html

بخش اول

جلسات اصلی

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۱: مقدماتی از الگوریتم‌ها

نگارنده: مهدی مستانی و علیرضا دادگرنیا

در این جلسه،مقدماتی از الگوریتم‌ها و چگونگی تحلیل آن‌ها بیان می‌کنیم. پیش‌نیاز این درس ، ریاضیات گسسته و ساختمان‌داده است(هر چند نیازی به جزئیات ساختمان داده در این درس نیازی نداریم) اما لازم است که با چگونگی کارکرد ساختمان‌داده‌هایی همچون درهم‌سازی[1] آشنایی داشته باشید.

۱ تطابق پایدار

مسئله تطابق(ازدواج) پایدار[2] بدین شرح است که گراف دوبخشی داریم که هر بخش آن n رأس دارد(که رئوس یک بخش آن را مرد و بخش دیگر آن را با زن مدل می‌کنند). برای هر رأس لیستی از ترجیحات از n رأس طرف مقابل وجود دارد(هر لیست کامل است و در واقع شامل n عنصر است که به ترتیب ترجیحات مرتب شده است).

حال سؤالی که وجود دارد این است که آیا یک تطابق (ازدواج)پایدار وجود دارد یا نه و اگر وجود دارد، چگونه آن را به طور کارآ بیابیم.منظور از پایدار این است که مردی و زنی نباشد که بخواهند یکدیگر را به زوج‌های فعلی خود ترجیح دهند. مثلاً اگر رأس x با رأس a و رأس y با رأس b منطبق[3] شده باشند و رأس y ، رأس a را به b (که زوج فعلی‌اش است) ترجیح دهد و رأس a نیز ، رأس y را به x (که زوج فعلی‌اش است) ترجیح دهد ،آنگاه (y,a) بیانگر این هستند که این مثال، ازداج پایداری نیست. این ازدواج اگر لیست ترجیحات افراد به صورت زیر باشد، ناپایدار است: $x : a,b$ و $y : a,b$ و $a : y,x$ و $b : x,y$. ازدواج پایدار این ترجیحات، $(x,b),(y,a)$ است.

این مسئله ، کاربردهای زیادی دارد؛مثلاً در برخی کشورها، منطبق کردن پزشکان و بیمارستان‌ها از این روش استفاده می‌شود. حال الگوریتمی ارائه می‌دهیم که به هر دو این سؤال، پاسخی دهد.

الگوریتم این سؤال،الگوریتم خواستگاری است.الگوریتم بدین صورت است که در این اجرای الگوریتم منطبق شدن های موقت داریم.مثلاً ممکن است در طول اجرای الگوریتم،به طور موقت بعضی مردها و زن‌ها با یکدیگر منطبق شوند(می‌توان آن را به دوران نامزدی تشبیه کرد!).

الگوریتم بدین صورت است که در ابتدا همه مردها و زن‌ها تنها هستند و با کسی منطبق نشده‌اند.الگوریتم بدین صورت است که تا زمانی که یک مرد تنها وجود دارد. یکی از مرد های تنها را به دلخواه انتخاب می‌کنیم(این فرد را m بنامید) و این مرد به اولین زنی که در لیست ترجیحاتش هنوز پیشنهاد نداده است (هنوز خواستگاری نکرده است)،پیشنهاد می‌دهد.(این زن را w بنامید). اگر w تنها است آنگاه (m,w) به طور موقت به هم اختصاص پیدا می‌کند و در غیر اینصورت اگر (m',w) یک زوج موقت است،اگر w ، m' را به m ترجیح می دهد ، m را رد می کند و در غیر این صورت (m,w) یک زوج موقت می شود و m' تنها می شود.

[1] hasing
[2] stable marriage
[3] match

حال باید نشان دهیم که الگوریتم تمام می‌شود و هنگامی که تمام شد، n زوج تشکیل شده توسط الگوریتم، تشکیل یک ازدواج پایدار می‌دهند.

مشاهده ۱ : وقتی که زنی به مردی نسبت پیدا کرد،نسبت داده‌شده باقی می‌ماند؛زیرا همانطور که از روند الگوریتم مشخص است، وقتی زنی به مردی اختصاص پیدا کرد،این اختصاص قرار نیست از بین برود و تنها زمانی تغییر پیدا می‌کند که مردی به این زن پیشنهاد دهد که از زوج فعلی او بهتر باشد و پس این زن قرار نیست دیگر هیچ وقت تنها شود.(اما ممکن است که برای مردها این اتفاق رخ دهد).همینطور وضعیت زن‌ها فقط می‌تواند که بهتر شود.(با فرض این‌که ازدواج با هر مردی از تنها بودن این زن بهتر است!!)

مشاهده ۲: اگر مردی تنها باشد، آن وقت حتماً یک زن وجود دارد که به او پیشنهاد نداده است(مطابق با الگوریتم). بدین دلیل که اگر مرد تنهای m به تمام زن‌ها پیشنهاد داده باشد و لیست تمام شده باشد، بدین معنی است که تمامی زن‌ها او را رد کرده اند (یا بلافاصله یا بعداً که با پیشنهاد بهتری مواجه شده‌اند).بنابر این همه زن‌ها اکنون اختصاص یافته‌اند و چون تعداد مرد و زن‌ها برابر است، پس نمی‌تواند این مرد تنها باشد، پس تناقض است.

این دو مشاهده بیان می‌کند که این الگوریتم پایان می‌پذیرد.(بعد از حداکثر n^2 پیشنهاد، پایان می‌یابد و دیگر هیچ مردی تنها نیست).

خروجی الگوریتم ، یک ازدواج پایدار است.فرض کنید که زوج (x,b) زوج ناراحت(ناپایدار) باشند. فرض کنید که (x,a) و (y,b) زوج‌های تخصیص یافته توسط الگوریتم باشد. منظور از ناراحت بودن زوج (x,b) این است که x ، b را به a که زوج فعلی‌اش است، ترجیح می‌دهد و b نیز x را به y که زوج فعلی‌اش است، ترجیح می‌دهد.این‌که x ، b را به a که زوج فعلی‌اش است، ترجیح می‌دهد،یعنی این‌که x قبل از a به b پیشنهاد داده است(زیرا پیشنهاد مردها به ترتیب لیست ترجیحات است) و رد شده است وگرنه به سراغ اولویت پایین‌تر نمی‌رفت.این اتفاق زمانی رخ می‌دهد که مورد بهتری برای b پیدا شده است و این فرد برایش مطلوب‌تر از x است(چون وضع زن‌ها طبق مشاهده ۱ بدتر نمی‌شود) که با ناراحت بودن زوج (x,b) تناقض است.

این الگوریتم برای مردان بهینه و برای زنان غیر بهینه() است.

حال می‌خواهیم کمی راجع‌به پیاده سازی الگوریتم صحبت کنیم.اگر نشان‌دهیم که می‌توان حلقه $while$ الگوریتم را در زمان ثابتی اجرا کرد، آنگاه چون حداکثر n^2 بار این حلقه اجرا می‌شود،پس زمان اجرای این الگوریتم از زمان $O(n^2)$ است.برای این‌کار با بتوانیم در $O(1)$ یک مرد تنها را پیدا کنیم. برای این‌کار می‌توانیم یک لیست پیوندی[۴]داشته باشیم که در ابتدا همه مردان در این لیست حضور دارند و هر دفعه از ابتدای لیست یک نفر را انتخاب می‌کنیم و این کار در زمان $O(1)$ است و همینطور مردانی که در حین اجرای الگوریتم تنها می‌شوند را به انتهای لیست اضافه کنیم. برای بررسی لیست ترجیحات در زمان $O(1)$ نیز می‌توانیم برای هر مرد یک آرایه که ترتیب اولویت‌ها در آن وجود دارد را نگه داری کنیم و یک شمارنده[۵]که نشان‌دهنده آخرین زنی است که این مرد به آن پیشنهاد داده است کدام است. برای قسمتی که زن w را باید بررسی کنیم که تنها است یا اگر تنها نیست، با چه‌کسی زوج موقت است، می‌توانیم یک آرایه به طول n داشته باشیم که برای هر زن مشخص‌کنیم که زوج موقتش چه‌کسی است(اگر تنها است، آن خانه ــ۱ است).برای قسمت آخر نیز باید که اگر w با مرد دیگری زوج موقت است(مانند m')، باید تشخیص‌دهیم که آیا w ، m' را به m ترجیح می دهد یا خیر.برای این‌کار می‌توانیم برای هر زن، آرایه‌ای درست کنیم که اگر آرایه ترجیحات (آن را $pref$ بنامید) این‌گونه باشد که خانه i ام آرایه بیان‌کننده این است که اولویت i ام این زن چه‌کسی است،آنگاه آرایه جدید(آن را $rank$ بنامید) این‌گونه است که خانه i ام آن بیان می‌کند که مرد شماره i ام ، اولویت چندم این زن است که این آرایه را از آرایه ترجیحات برای هر زن می‌توان در $O(n)$ ساخت. در واقع شبه کد ساختن آرایه $rank$ به صورت زیر است:

```
1  for i = 1 to n
2        rank[pref[i]] = i
```

[۴] linked list
[۵] counter

بنابراین برای مقایسه این که آیا w ، m' را به m ترجیح می دهد یا خیر، باید مقایسه $w.rank[m] > w.rank[m']$ را انجام‌دهیم تا این قسمت را انجام‌دهیم.

پس این قسمت نیز در زمان ثابت انجام می‌شود . پس هر بار اجرای حلقه $while$ در زمان ثابت قابل انجام است.

۲ تعاریف

منظور از مسئله در این درس این است که به ما داده‌هایی را به عنوان **ورودی** داده‌اند و خواسته سوال را به عنوان **خروجی** مسئله از ما می‌خواهند. در واقع منظور از ورودی،خانواده‌ای از نامتناهی ورودی است و مشخص شده است که برای هر ورودی از این خانواده ، چه خروجی‌ای مد نظر است(در واقع تابعی داریم که مشخص می‌کند که متناظر با هر ورودی، چه خروجی‌ای قرار دارد).حال بیانی از طراحی الگوریتم این است که ما بتوانیم این تابع را بیابیم.در مورد مسائلی که تعداد ورودی‌های متناهی است،مطلوبیتی از لحاظ محاسباتی ندارد.

یک مثال برای تعداد متناهی ورودی ، مسئله زیر است:

ورودی: یک موقعیت از صفحه شطرنج

خروجی: آیا بازیکن با مهره سفید،استراتژی برد دارد یا خیر.

تعداد ورودی‌های این مسئله، متناهی است(ولی تعداد آن خیلی زیاد است) و در دسته‌بندی مسائل محاسباتی که آن‌ها را بررسی می‌کنیم نیست. اما اگر این مسئله را تعمیم دهیم و برای مثال صفحه شطرنج را $n \times n$ در نظر بگیریم، آنگاه تعداد ورودی‌ها متناهی نیست.

منظور از الگوریتم،پروسه دقیق و متناهی است و کاملاً واضح باشد که در هر مرحله از الگوریتم چه پروسه‌ای باید انجام شود و تعداد گام‌های الگوریتم متناهی باشد و تمام شود. هر گام الگوریتم نیز باید قابل اجرا باشد، برای مثال اگر در گامی از الگوریتمی داشته باشیم: "اگر ۴ بزرگترین عدد صحیحی است که معادله $w^n + x^n + y^n = z^n$ جواب صحیح دارد آنگاه ...در غیر اینصورت ..." این گام از لحاظ دقیق بودن دچار مشکل نیست ولی فهمیدن درست بودن آن دشوار است.

بیان دقیق‌تر الگوریتم این است که باید یک مدل ریاضی برای کامپیوتر ارائه دهیم که یک مدل ، مدل ماشین تورینگ است ولی برای بیان مدل ریاضی در این درس از این مدل استفاده نمی‌شود و از مدل $WordRam$ است. فرض ما در این مدل این است که کامپیوتر حافظه‌ای دارد که تعدادی خانه دارد و در هر خانه آن می‌توان عددی نگه‌داشت که w بیت است و اگر ورودی ما از سایز n است،آنگاه $w \geq \lg n$ است.در این درس خیلی نیازی نیست که به طور دقیق مدل محاسباتی را بررسی کنیم اما می‌توان همه این موارد را دقیق بیان کرد.در این مدل می‌توان برخی اعمال را در زمان ثابت انجام داد مانند خواندن و نوشتن در یک خانه حافظه ، یا اعمال مقدماتی مثل جمع و اعمال بیتی .

در این درس بیان الگوریتم‌ها ممکن است به صورت توصیفی باشد یا آن‌ها را به صورت شبه کدی بیان کنیم و باید استدلال کنیم که هر گام(خط) از الگوریتم چقدر طول می‌کشد و با این کار می‌توانیم زمان اجرای الگوریتم را بهتر تحلیل کنیم.

۳ تحلیل زمان اجرای الگوریتم

ممکن است این سوال مطرح شود که دقیقاً منظور ما از زمان اجرا چیست. ابتدا باید بگوییم که زمان اجرا را برحسب طول ورودی می‌سنجیم. همان‌طور که گفتیم در مسائل محاسباتی، یک خانواده از ورودی‌ها داریم و می‌خواهیم الگوریتم ما برای هر کدام از آن‌ها خروجی مورد نظر را بدهد. هدف این است که زمان اجرای الگوریتم را برای هر طول ورودی حساب کنیم. حال فرض کنید ورودی الگوریتم n عدد متفاوت باشد. سوالی که مطرح می‌شود این است که زمانی که می‌خواهیم الگوریتم را تحلیل کنیم، بین همهٔ حالت‌های ممکن ورودی‌ها کدام را

در نظر می‌گیریم؟ همان‌طور که احتمالاً قبلاً دیده‌اید، بدترین حالت را تحلیل می‌کنیم. یعنی $T(n)$، زمان اجرای الگوریتم، را تعریف کنیم بیش‌ترین زمان اجرای الگوریتم روی ورودی‌های به‌طول n. حُسنی که این کار دارد این است که از نظر ریاضی تحلیل آن آسان است (در اکثر مسائل) و تضمین می‌کند که زمان اجرای ما در بدترین حالت هم خوب است. همچنین، حداقل برای مسائل کلاسیک و ساده‌تر، همبستگی خوبی با الگوریتم در عمل دارد. اما همیشه هم این‌گونه نیست. الگوریتم‌هایی وجود دارند که در عمل خیلی خوب کار می‌کنند اما زمان اجرای بدترین حالت آن‌ها بد است. در واقع ورودی‌های خیلی خاصی وجود دارند که الگوریتم به شدت روی آن‌ها بد عمل می‌کند اما انگار در عمل آن ورودی‌های خیلی خاص هیچ‌وقت اتفاق نمی‌افتند یا احتمال اتفاق افتادن آن‌ها بسیار کم است. پس نیاز است که یک‌سری روش‌های تحلیل مدرن‌تر نیز وجود داشته باشند که الگوریتم‌هایی که تحت ملاک‌هایی می‌خواهیم خوب به نظر برسند را تحلیل کنند. این روش‌های تحلیل نیز وجود دارند و زیاد روی آن‌ها کار شده است. به خصوص در بیست سال اخیر که گاهی به آن تحلیل ورای بدترین حالت[۶] گفته می‌شود. در این درس به آن نمی‌پردازیم اما چند جایگزین ساده‌تر نیز وجود دارد مانند تحلیل حالت متوسط. در واقع ساده‌ترین چیزی که به ذهن می‌رسد این است که به جای تحلیل بدترین حالت، حالت متوسط را تحلیل کنیم. به نظر گزینهٔ جذابی می‌آید اما یک ایرادهایی دارد. مهم‌ترین ایراد آن این است که متوسط را چه چیزی تعریف کنیم؟ اگر بخواهیم با امید ریاضی، متوسط را تعریف کنیم نیاز داریم که بدانیم ورودی‌های ما از چه توزیعی پیروی می‌کنند. هر نوع توزیعی روی ورودی‌ها بذاریم ممکن است مصنوعی باشد یعنی به عمل ربطی نداشته باشد. در واقع ممکن است تحلیل ریاضی خوبی داشته باشد اما عملاً ما را گمراه کند و به نظر بیاید که تحت یک توزیع زمان اجرای الگوریتم خوب است اما تضمینی برای آن وجود ندارد و ممکن است در عمل بسیار بد عمل کند. برخلاف تحلیل بدترین حالت که اگر خوب بود به ما تضمین می‌داد که الگوریتم همیشه خوب کار می‌کند. بنابراین تحلیل حالت متوسط چیزی نیست که خیلی مهم باشد هر چند روی آن زیاد کار شده است و نتایج جالبی نیز در مورد آن وجود دارد.

دقت کنید که گفتیم زمان اجرا را برحسب طول ورودی محاسبه می‌کنیم و ممکن است این تصور به وجود بیاید که منظور یک تابع است اما، همان‌طور که احتمالاً در درس ساختمان داده‌ها و مثالی که در همین جلسه به آن اشاره کردیم، دیده‌اید، زمان اجرا را به صورت مرتبه‌ای محاسبه می‌کنیم و ضرایب ثابت را اکثر اوقات نادیده می‌گیریم. سوال دیگر این است که طول ورودی به چه معناست؟ تعریف دقیق طول ورودی، تعداد بیت‌هایی که در ورودی وجود دارد، است. مثلاً اگر ورودی مسئله یک گراف است و راس‌ها و یال‌های آن وزن‌هایی دارند، کل بیت‌هایی که برای ذخیرهٔ این اطلاعات لازم است را طول ورودی می‌گوییم. اما بسیاری از اوقات در تحلیل الگوریتم‌ها، طول ورودی را تعداد عناصر ورودی در نظر می‌گیریم. برای مثال اگر بخواهیم تعدادی عدد را ورودی بگیریم، در صورتی‌که بخواهیم تعداد بیت‌ها را در نظر بگیریم طول این اعداد مهم می‌شود و محاسبهٔ آن سخت می‌شود. به همین دلیل ما فرضی را روی مدل محاسباتی قرار می‌دهیم که هر عمل مقدماتی روی دو عدد را در $O(1)$ انجام می‌دهد. با این فرض می‌توانیم طول ورودی را همان تعداد عناصر ورودی در نظر بگیریم و دیگر به اندازهٔ اعداد ورودی دقت نکنیم. بسیاری از الگوریتم‌ها و تحلیل‌هایی که ما انجام می‌دهیم به این شکل هستند اما گاهی لازم می‌شود که به سراغ تعریف دقیق برویم. در واقع ممکن است اعداد ورودی خیلی بزرگ باشند یا فرضی که اضافه کردیم معقول نباشد یا تعداد گام‌های الگوریتم به اندازهٔ اعداد ورودی ربط پیدا کند. در این شرایط لازم است که تعداد بیت‌ها را به عنوان طول ورودی در نظر بگیریم. به‌طور خاص در جلسات برنامه‌ریزی پویا[۷]، مسئلهٔ جریان بیشینه[۸] و بحث‌هایی که در قسمت پیچیدگی محاسباتی داریم، گاهی اندازهٔ اعداد در زمان اجرا تاثیر دارند و لازم است که به آن‌ها دقت کنیم.

تا این‌جا توضیح دادیم که یک مسئله‌ای داریم به نام مسئلهٔ محاسباتی، به دنبال الگوریتم برای حل آن هستیم و دربارهٔ خود الگوریتم و زمان اجرای آن بحث کردیم. حال توجه کنید که ما به دنبال الگوریتمی هستیم که زمان اجرای آن خوب باشد. پس نیاز داریم تعریفی از کارآیی[۹] (خوب بودن) الگوریتم نیز ارائه دهیم. بسیاری از مسائل محاسباتی که ما با آن‌ها سر و کار داریم به این شکل هستند که یک الگوریتم بد یا الگوریتمی که خیلی لازم نیست فکری در مورد آن بکنیم، را به راحتی می‌توان پیدا کرد اما طبعاً زمان اجرای آن خیلی خوب نیست. برای مثال مسئلهٔ مجموعهٔ مستقل بیشینه را در نظر بگیرید. در این مسئله ورودی یک گراف و خروجی زیرمجموعه‌ای مستقل از رئوس با اندازهٔ بیشینه، است. زیرمجموعهٔ مستقل یعنی تعدادی رأس که بین هیچ دوتایی از آن‌ها یال وجود نداشته باشد. این مسئله یکی از

[6] Beyond worst case analysis
[7] Dynamic Programming
[8] Maximum flow
[9] Efficiency

مسائل کلاسیک الگوریتم‌های گراف است. به راحتی می‌توانیم الگوریتمی بدهیم که مسئله را حل کند. تنها کافی است همهٔ زیرمجموعه‌های رئوس را بررسی کنیم. یعنی 2^n زیرمجموعه را باید بررسی کنیم و یک زمانی را صرف کنیم که بفهمیم بین هر دوتایی یال وجود دارد یا وجود ندارد. مثلاً می‌توانیم در این زیرمجموعه هر زوج از نقاط را در نظر بگیریم و وجود یال بین آن‌ها را بررسی کنیم که هزینهٔ $O(n^2)$ نیز برای این بررسی باید بدهیم. پس به هر حال با بررسی همهٔ حالات می‌توانیم مسئله را حل کنیم. در واقع خیلی از مسائل الگوریتمِ جستجوی همهٔ حالت‌ها[۱۰] دارند چون تعداد جواب‌های آن‌ها متناهی است و می‌توانیم با بررسی کردن همهٔ حالت‌ها خروجی مدنظر را پیدا کنیم. اما نکته این است که معمولاً الگوریتم‌های به این شکل، زمان اجرای بدی دارند و برحسب ورودی نمایی هستند. در همین سوال زمان اجرای الگوریتمی که بدون فکر پیدا کردیم $O(2^n n^2)$ است. در بعضی از سوالات هم ممکن است به مرتبهٔ $n!$ برسیم. مثلاً در مسئلهٔ تطابق پایدار $n!$ حالت مختلف وجود دارد. البته این هم سوال خوبی برای فکر کردن است که برنامه‌ای بنویسید که همهٔ زیرمجموعه‌های یک مجموعهٔ n عضوی یا همهٔ جایشگت‌های n عدد را تولید کند. اگر این کار را انجام دهید خیلی از مسائل را می‌توانید با جستجوی همهٔ حالات حل کنید. طبعاً این زمان اجراها خوب نیستند زیرا مثلاً اگر n (تعداد عناصر ورودی) یکی زیاد شود آنگاه زمان اجرا دو برابر می‌شود اما ما دوست داریم که اگر تعداد عناصر ورودی در یک عددی ضرب شد زمان اجرا نیز در یک عدد ضرب شود. اگر بخواهیم چنین زمان اجرایی داشته باشیم به زمان اجرای چندجمله‌ای می‌رسیم. پس یک تعریف استاندارد و قابل دفاع برای کارآیی یک الگوریتم، داشتن زمان اجرای چندجمله‌ای است. برای مثال اگر زمان اجرای الگوریتم $O(n^k)$ باشد و تعداد عناصر ورودی دو برابر شود آنگاه زمان اجرا در یک عدد ثابت (2^k) ضرب می‌شود که در مقایسه با زمان اجرای نمایی بهبود خیلی بزرگی محسوب می‌شود. همچنین چندجمله‌ای‌ها خواص خیلی خوبی نیز دارند. مثلاً جمع، ضرب و ترکیبِ دو چندجمله‌ای باز هم یک چندجمله‌ای است. پس اگر زمان اجرای چندجمله‌ای را تعریف الگوریتم خوب در نظر بگیریم آنگاه اگر یک الگوریتم خوب، یک الگوریتم خوب دیگر را به عنوان زیربرنامه داشته باشد و تعدادی دفعه از آن استفاده کند، باز هم زمان اجرا چندجمله‌ای است. در واقع اگر تعدادی الگوریتم با زمان اجرای چندجمله‌ای را به روش‌های مختلف با هم تلفیق کنیم همچنان یک الگوریتم کارآمد داریم. یک نکتهٔ دیگر نیز این است که اگر یک الگوریتم را در مدل‌های محاسباتی یا ریاضیاتی مختلف که برای کامپیوتر می‌توان ارائه داد، مانند Word RAM یا ماشین تورینگ، اجرا کنیم معمولاً زمان اجرای آن‌ها به‌طور چندجمله‌ای با هم فرق دارند. این موارد باعث می‌شوند که این تعریف، تعریف خوش‌دستی باشد. یعنی اگر زمان اجرای یک الگوریتم در مدل Word RAM چندجمله‌ای باشد در مدل ماشین تورینگ نیز چندجمله‌ای است. تنها ممکن است این چندجمله‌ای‌ها با هم فرق داشته باشند مثلاً در یک مدل $O(n^2)$ و در مدل دیگر $O(n^3)$ باشد. البته اگر با ماشین تورینگ آشنایی ندارید، جای نگرانی وجود ندارد و قرار نیست از این مفاهیم استفاده‌ای بکنیم. در عین حال این تعریف کاستی‌هایی نیز دارد. مثلاً اگر زمان اجرای یک الگوریتم $O(n^{100})$ باشد به وضوح کارآ نیست. این نکتهٔ معقولی است اما در عمل کم پیش می‌آید و مسائلی که الگوریتم چندجمله‌ای دارند، زمان اجرای آن‌ها معمولاً حداکثر $O(n^3)$ است و درجهٔ بالایی ندارند. مشابه این اعتراض به نمادگذاری O نیز وارد است یعنی ممکن است زمان اجرای یک الگوریتم $O(n)$ باشد اما آن ثابتی که در O مخفی شده، بسیار بزرگ باشد و الگوریتم کارآمدی نباشد. حال اگر بتوانیم نشان دهیم که یک مسئله الگوریتمی با زمان اجرای چندجمله‌ای ندارد آنگاه بهتر قانع می‌شویم که واقعاً الگوریتم کارآ ندارد و یک مسئلهٔ سخت است. در این رابطه، در بخش پیچیدگی محاسباتی توضیحات خیلی بیش‌تری ارائه خواهیم داد. درست است که الگوریتم کارآ را الگوریتمی با زمان اجرای چندجمله‌ای تعریف کردیم اما در این درس هدف ما این نیست که برای مسائل الگوریتم با زمان اجرای $O(n^{100})$ ارائه دهیم و هدف ما این است که الگوریتم با زمان اجرای $O(n)$، $O(n \log n)$ و $O(n^2)$ ارائه دهیم یعنی یک چندجمله‌ای که درجهٔ آن پایین باشد. در بعضی مسائل پیدا کردن یک الگوریتم با زمان اجرای $O(n^2)$ کار سختی نیست و هوشمندی این است که الگوریتم با زمان اجرای $O(n \log n)$ پیدا کنیم که خیلی بهتر از $O(n^2)$ است اگر n بزرگ شود.

این نکته را باید اضافه کنیم که در بین مسائلی که با آن‌ها روبه‌رو می‌شویم چند نوع از آن‌ها هستند که خیلی زیاد ظاهر می‌شوند. مثلاً مسئلهٔ مجموعهٔ مستقل بیشینه، یک مسئلهٔ بهینه‌سازی است. زیرمجموعه‌های مختلفی از رئوس داریم که مستقل هستند. مثلاً هر تک رأس یک مجموعهٔ مستقل است اما مسئله، زیرمجموعه‌ای را می‌خواهد که سایز آن بیشینه است. بنابراین می‌توان گفت این مسئله، یک مسئلهٔ بهینه‌سازی است. نوع دیگر مسائل، مسائل جستجو هستند یعنی سوال این است که یک چیزی وجود دارد یا ندارد و اگر وجود دارد یکی از آن‌ها را خروجی دهیم. برای مثال در مسئلهٔ تطابق پایدار، هدف این است که یک تطابق پایدار را پیدا کنیم. البته گاهی هم فقط جواب مسئله به شکل *بله* یا خیر است. مثلاً گرافی داده شده و سوال این است که آیا گراف دور همیلتونی (دوری که از همهٔ رئوس بگذرد) دارد؟

[10] Brute force

پس چند نوع مسئلهٔ مختلف داریم و نکته این است که در بسیاری از مواقع می‌توان این مسائل را به هم تبدیل کرد. برای مثال در مسئلهٔ مجموعهٔ مستقل بیشینه، می‌توانیم یک عدد k به ورودی اضافه کنیم و خروجی این باشد که آیا گراف زیرمجموعه‌ای مستقل با اندازهٔ حداقل k دارد؟ با این کار این مسئلهٔ بهینه‌سازی به یک مسئلهٔ جستجو تبدیل می‌شود.

حال بد نیست که یک‌سری از زمان اجراهای رایج را به همراه مثال ببینیم. فرض ما بر این است که مدل محاسباتی ما یک‌سری اعمال مثال خواندن یک خانه از حافظه یا اعمال ریاضی و منطقی مقدماتی روی دو خانه از حافظه یا مقایسهٔ دو خانه از حافظه و در کل گام‌های مقدماتی الگوریتم ما را در $O(1)$ انجام می‌دهد. مثال‌هایی هم از الگوریتم‌هایی با زمان اجرای $O(\log n)$ دیده‌ایم. به عنوان مثال در صورتی‌که خواندن ورودی را جزء زمان اجرای الگوریتم حساب نکنیم، پیدا کردن یک عدد در یک آرایهٔ مرتب‌شده با استفاده از جستجوی دودویی[11]، $O(\log n)$ زمان می‌برد. زمان اجرای دیگری که خیلی جاها ظاهر می‌شود و اصولاً بهترین زمان اجرا قلمداد می‌شود زمان اجرای خطی یا $O(n)$ است یعنی زمان اجرا برحسب طول ورودی خطی باشد چون معمولاً خواندن ورودی را نیز جزء زمان اجرا حساب می‌کنیم این بهترین زمان اجرایی است که می‌توانیم به آن برسیم. به عنوان مثال اگر بخواهیم در آرایه‌ای کمینه یا بیشینه را پیدا کنیم یا دو آرایهٔ مرتب‌شده را ادغام کنیم. اگر الگوریتم مرتب‌سازی ادغامی[12] را به یاد آورید، ادغام دو آرایهٔ مرتب‌شده را می‌توانستیم در $O(n)$ انجام دهیم. مثال دیگر این است که در لیستی از n عدد مرتب‌شده، دو عدد پیدا کنیم که جمع آن‌ها k شود. این مسئله نیز در $O(n)$ قابل حل است. در واقع دو اشاره‌گر، یکی در ابتدای آرایه و یکی در انتهای آرایه، قرار می‌دهیم. سپس عدد متناظر این دو خانه را با هم جمع می‌زنیم. اگر حاصل k شد که حل مسئله تمام است. اگر کم‌تر از k شد یعنی خانهٔ اول آرایه به در نمی‌خورد چون حتی اگر آن را با بزرگ‌ترین عدد هم جمع کنیم باز به k نمی‌رسد بنابراین می‌توانیم اشاره‌گر خانهٔ اول را یکی جلو ببریم و همین‌طور اگر جمع آن‌ها بیش‌تر از k شد یعنی خانهٔ آخر آرایه به درد نمی‌خورد و می‌توانیم اشاره‌گر خانهٔ آخر را یکی عقب ببریم. به همین ترتیب در هر گام می‌توانیم یکی از دو اشاره‌گر به سمت دیگری حرکت دهیم و بنابراین بعد از n گام بالأخره معلوم می‌شود که دو عدد با مجموع k وجود دارند یا ندارند. الگوریتم بدون فکر این مسئله نیز به این شکل است که همهٔ $\binom{n}{2}$ حالت ممکن را بررسی کنیم که زمان اجرای آن $O(n^2)$ است. حال سوال سخت‌تر این است که در لیستی از n عدد (نه لزوماً مرتب‌شده)، دو پیدا کنید که جمع آن‌ها k شود. الگوریتم بدیهی این است که ابتدا آرایه را مرتب کنیم و بعد از آن مانند مسئلهٔ قبل عمل کنیم که زمان اجرای آن $O(n \log n)$ است زیرا این زمان صرف مرتب‌سازی می‌شود (با استفاده از الگوریتم‌هایی نظیر مرتب‌سازی ادغامی) و سپس در $O(n)$ دو عدد را پیدا می‌کنیم. نکتهٔ جالب این است که در $O(n)$ نیز می‌توانیم مسئله را حل کنیم. در واقع باید از درهم‌سازی[13] استفاده کنیم. یعنی همهٔ n عدد را در یک جدول درهم‌سازی ذخیره کنیم سپس روی آرایه حرکت کنیم و هر عدد a که دیدیم را بررسی کنیم که $k-a$ در جدول درهم‌سازی وجود دارد یا ندارد. این کار به‌طور متوسط هزینهٔ $O(1)$ را دارد پس در کل به‌طور متوسط هزینهٔ $O(n)$ را باید بدهیم. به زمان اجرای $O(n \log n)$ می‌رسیم. مهم‌ترین مسئله‌ای که در این زمان حل می‌شود، مسئلهٔ مرتب‌سازی است که به عنوان زیربرنامه در خیلی از مسائل دیگر نیز ظاهر می‌شود و خیلی از مسائل دیگری که مثلاً با روش تقسیم و حل، حل می‌شوند چنین زمان اجرایی دارند. یک مسئلهٔ جالب در این زمینه، که به‌طور مفصل در جلسهٔ بعد راجع‌به آن صحبت خواهد شد، این است که یک‌سری نقطه در صفحه به ما داده شده و باید نزدیک‌ترین زوج نقطه را پیدا کنیم. این مسئله هم الگوریتم بدیهی $O(n^2)$ دارد و در جلسهٔ بعد الگوریتم‌های بهتری برای آن خواهیم داد. مسئلهٔ دیگری که الگوریتم بدیهی آن $O(n^3)$ است، مسئلهٔ 3SUM است. ورودی مسئله اعداد $a_1, a_2, \ldots, a_n$ هستند و سوال این است که آیا سه عدد مانند a_i، a_j و a_k وجود دارند که جمع آن‌ها صفر شود؟ برای این مسئله نیز با استفاده از جدول درهم‌سازی می‌توان الگوریتم $O(n^2)$ داد. در واقع همهٔ اعداد را در یک جدول درهم‌سازی ذخیره می‌کنیم و برای هر زوج از اعداد بررسی می‌کنیم که آیا قرینهٔ مجموع آن‌ها در جدول وجود دارد یا ندارد. سوالی که مطرح می‌شود این است که آیا الگوریتم قطعی $O(n^2)$ نیز داریم؟ الگوریتم قطعی $O(n^2 \log n)$ داریم زیرا می‌توانیم ابتدا همهٔ اعداد را مرتب کنیم و برای هر زوج عدد a, b، با استفاده از جستجوی دودویی بررسی کنیم که آیا $-(a+b)$ در بین اعداد وجود دارد یا ندارد. الگوریتم قطعی $O(n^2)$ نیز داریم. باز هم آرایه را مرتب می‌کنیم و با استفاده از مسئلهٔ قبل، برای هر عدد a بررسی می‌کنیم که آیا دو عدد وجود دارند که مجموع آن‌ها برابر با $-a$ شود.

[11] Binary search
[12] Merge sort
[13] Hashing

۴ کلیات درس

در این بخش قرار است راجع‌به کلیت درس توضیحاتی ارائه دهیم. نکتهٔ اول این است که در جزوه‌ها مسائلی وجود دارند که شاید بهتر باشد قبل از خواندن راه حل آن‌ها مقداری خودتان روی آن سوال فکر کنید و بعد به سراغ راه حل سوال بروید. بعضی مواقع نیز شاید نیاز باشد که بعد از خواندن یک مطلب برای درک بهتر راجع‌به آن فکر کنید. در واقع هر چه بیش‌تر ذهن خود را درگیر مطالب کنید طبعاً بهتر می‌توانید یاد بگیرید. نکتهٔ دوم این است که مطالب متنوع هستند. بعضی از آن‌ها ممکن است مقداری سخت‌تر باشند و بعضی آسان‌تر. به‌طور خاص در انتهای بعضی از جلسات ممکن است مطالبی گفته شود که مقداری سخت‌تر یا پیشرفته‌تر باشند. می‌توانید این مطالب را دنبال نکنید یا زمان دیگری به سراغ آن‌ها بروید. مثلاً جلسهٔ دوم به همین شکل است و می‌توانید مسئله‌ای که در انتهای جلسه بررسی می‌شود را دنبال نکنید یا زمان دیگری به سراغ آن بروید. جلساتی نیز وجود دارند که از حالت عادی طولانی‌تر هستند مانند جلسات ۲۱ و ۲۲ که در انتهای آن‌ها مطالب اختیاری گفته می‌شود و در بودجهٔ امتحان نیز قرار ندارند. در این درس قرار است که هم راجع‌به تکنیک‌های متنوع طراحی الگوریتم، هم مسائل کلاسیکی که خیلی از اوقات در عمل با آن‌ها روبه‌رو می‌شویم، به‌طور مستقیم یا زیربرنامه در مسائل جدید، بحث کنیم. همین‌طور تا حدی راجع‌به مدل‌های محاسباتی دیگر. اکثر مسائلی که بررسی خواهیم کرد، الگوی کلاسیک دارند یعنی یک ورودی (به صورت یک‌جا) داریم و قرار است که یک خروجی بدهیم. گاهی ممکن است مدل مسئله به این شکل باشد که ورودی را به صورت یک‌جا نداریم و ورودی تکه‌تکه به ما داده شود. الگوریتم ما باید در هر لحظه تصمیمی بگیرد و نمی‌تواند دیگر تصمیمش را برگرداند. در واقع یک مدل برخط[۱۴] است که در جلسات اضافه راجع‌به آن‌ها صحبت می‌شود. ممکن است حافظهٔ کافی نداشته باشیم که کل ورودی را ذخیره کنیم و تنها بتوانیم یک بار کل ورودی را نگاه کنیم و اطلاعاتی از آن را نگه داریم و یک‌سری اطلاعات راجع‌به ورودی به‌دست بیاوریم. گاهی نیز ممکن است نخواهیم مسئله را دقیق حل کنیم و بخواهیم تقریبی حل کنیم یا ممکن است بخواهیم الگوریتم تصادفی ارائه دهیم که به این الگوریتم‌ها در درس بیش‌تر می‌پردازیم. به‌طور خاص بخش زیادی از جلسات اضافه دربارهٔ تکنیک‌های تصادفی است، در بخش مقدماتی و کلاسیک درس نیز گاهی گریزی به الگوریتم‌های تصادفی می‌زنیم. به عنوان مثال در جلسهٔ بعد یک الگوریتم تصادفی را برای مسئلهٔ پیدا کردن نزدیک‌ترین زوج نقطه خواهیم دید. اصولاً در درس‌های مقدماتی الگوریتم، از جمله درس ما و حتی در قسمت‌هایی که مقداری پیشرفته‌تر می‌شود، همچنان به اصطلاح الگوریتم‌های ما، الگوریتم‌های ترکیبیاتی هستند. یک‌سری از روش‌هایی که روش‌های مدرن‌تر در طراحی الگوریتم هستند و در یکی دو دههٔ اخیر رواج بیش‌تری پیدا کرده‌اند نسبت به قبل، طراحی الگوریتم با استفاده از روش‌های بهینه‌سازی پیوسته هستند. البته روش بهینه‌سازی خطی را در این درس بررسی خواهیم کرد. اما روش‌های بهینه‌سازی پیوسته یا روش‌های مبتنی بر جبر خطی، روش‌های مدرن طراحی الگوریتم هستند که در این درس به آن‌ها پرداخته نمی‌شود و بیش‌تر با الگوریتم‌های ترکیبیاتی سر و کار داریم.

اگر راجع‌به خود جلسات بخواهیم توضیح دهیم، جلسات ۱ تا ۱۱، مباحث بنیادی و کلاسیک هستند یعنی مباحثی هستند که باید به اندازهٔ کافی روی آن‌ها وقت بگذارید و سعی کنید که هر چه‌قدر لازم است تأمل کنید، مسئله حل کنید و در کل مباحث این جلسات را خوب یاد بگیرید. روش‌های کلاسیک طراحی الگوریتم مانند الگوریتم‌های مقدماتی گراف، الگوریتم‌های پیمایش گراف، کوتاه‌ترین مسیر و برنامه‌ریزی پویا قسمت‌های مقدماتی درس را تشکیل می‌دهند که کامل باید دنبال کنید. از آن‌جا به بعد می‌توان گفت که درس چند تکه می‌شود که بعضاً می‌توان آن‌ها را مستقل از هم نیز دنبال کرد. مثلاً جلسات ۱۳ تا ۱۶، راجع‌به جریان بیشینه است. در جلسات ۱۷ تا ۲۰، راجع‌به اثبات سختی مسائل بحث خواهد شد و جلسات ۲۱ و ۲۲، در رابطه با حل مسائل سخت هستند که می‌توان این پنج جلسه را نیز یک تکه در نظر گرفت. در جلسات ۲۳ تا ۲۶، برنامه‌ریزی خطی بررسی می‌شود و این جلسات قسمت کلاسیک درس را تشکیل می‌دهند. این تکه‌ها را لازم نیست که به‌ترتیب دنبال کنید فقط خوب است که مقدماتی از مسئلهٔ جریان بیشینه را بدانید حتی اگر نخواهید آن را کامل دنبال کنید، مثلاً خوب است که جلسهٔ ۱۳ و نیمی از جلسهٔ ۱۴ را ببینید اما بعد از آن می‌توانید به قسمت‌های دیگر مانند مسائل سخت یا برنامه‌ریزی خطی سر بزنید. می‌توان گفت دو تکهٔ مسائل سخت و برنامه‌ریزی خطی از یکدیگر مستقل هستند.
بعد از این جلسات، جلسات اضافهٔ درس هستند که آن‌ها نیز وابستگی زیادی به جلسات ۱۳ تا ۲۶ ندارند به جز حالت‌های خاص. مثلاً ممکن است اسمی از ان‌پی‌ـ تمامیت[۱۵] آورده شود که تعریف آن در جلسات مسائل سخت گفته می‌شود. خود جلسات اضافه نیز تکه‌هایی

[14] Online
[15] **NP**-Completeness

دارند و مهم‌ترین موضوع آن‌ها را الگوریتم‌های تصادفی تشکیل می‌دهند. جلسات ۱ و ۲ اضافه در رابطه با الگوریتم‌های برخط هستند. دو جلسهٔ الگوریتم‌های پارامتر ثابت و تجزیهٔ درختی یک تکهٔ دیگر هستند که ربط خاصی به جلسات قبل از خود ندارند، شاید بتوان گفت که تنها پیش‌نیاز آن جلسهٔ ۲۱ است. الگوریتم‌های تقریبی را نیز می‌توان به‌طور مستقل دنبال کرد به جز جلسه‌ای که راجع‌به روش‌های مبتنی بر برنامه‌ریزی خطی است برای طراحی الگوریتم‌های تقریبی و طبعاً پیش‌نیاز آن برنامه‌ریزی خطی است. بیش‌تر جلسات اضافه که در رابطه با الگوریتم تصادفی است، خوب است که به ترتیب دنبال شود.

نکتهٔ نهایی، که احتمالاً خودتان می‌دانید، این است که هر چه‌قدر ذهن خود را بیش‌تر درگیر کنید، بهتر می‌توانید مباحث را یاد بگیرید یعنی هر چه‌قدر بیش‌تر فکر کنید، مسئله حل کنید، مسئله‌هایی که سر کلاس یا کلاس حل تمرین حل شده‌اند و حل آن‌ها در جزوه موجود است، را قبل از دیدن راه حل خودتان حل کنید، تاثیر زیادی در یادگیری شما دارد. در کل میزان یادگیری شما تناسب دارد با میزانی که انرژی ذهنی صرف می‌کنید و سعی می‌کنید یک چیزهایی را بفهمید یا مسئلهٔ جدید حل کنید، حتی اگر حل نشود.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۲: تقسیم و حل[1] نگارنده: علیرضا دادگرنیا

در جلسهٔ قبل با مقدمات و کلیات درس آشنا شدیم. در این جلسه قرار است مروری بر روش تقسیم و حل داشته باشیم و یک مثال کلاسیک را با استفاده از این روش و یک روش تصادفی که زمان اجرای بهتری به ما می‌دهد حل کنیم. در انتهای جلسه نیز یک مثال دیگر را با همان روش تصادفی حل می‌کنیم.

۱ روش تقسیم و حل

روش تقسیم و حل از سه مرحله تشکیل شده است. در این روش ابتدا مسئله را به چند زیرمسئله با سایز کوچک‌تر **تقسیم** می‌کنیم سپس هر کدام از این زیرمسئله‌ها را **حل** می‌کنیم و در نهایت زیرمسئله‌ها را با هم **تلفیق** (ادغام) می‌کنیم تا به جواب مسئلهٔ اصلی برسیم. برای مثال الگوریتم مرتب‌سازی ادغامی[2] ابتدا آرایهٔ nتایی را به دو زیرآرایهٔ $\frac{n}{2}$تایی تقسیم می‌کرد، هر کدام از این زیرآرایه‌ها به طور بازگشتی مرتب می‌کرد و جواب این دو زیرآرایه را با هم ادغام می‌کرد تا به جواب آرایهٔ اصلی برسیم. در این الگوریتم دو مرحلهٔ تقسیم و حل به راحتی انجام می‌شوند اما برای قسمت تلفیق نیاز داریم که یک الگوریتم بنویسیم که در $O(n)$ تلفیق را انجام دهد. گاهی ممکن است، برعکس الگوریتم مرتب‌سازی ادغامی، بخش تقسیم نیاز به عمل هوشمندانه‌ای داشته باشد و بخش تلفیق ساده باشد و گاهی ممکن است هر دو بخش نیاز به الگوریتم هوشمندانه‌ای داشته باشند و هیچ‌کدام به سادگی قابل انجام نباشند. برای مثال در مرتب‌سازی سریع[3] بخش تلفیق بعد از حل زیرمسئله‌ها خودبه‌خود انجام می‌شود و چالش اصلی بخش تقسیم مسئله است.

۲ مسئلهٔ پیدا کردن نزدیک‌ترین زوج نقطه در صفحه

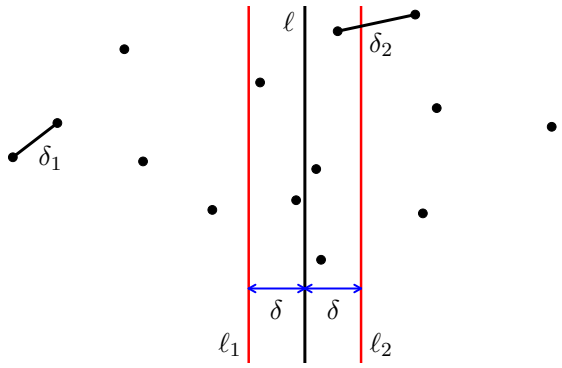
مسئله این است که n نقطهٔ $p_1, p_2, \dots, p_n$ در صفحه به ما داده شده است و ما باید اندیس‌های i و j را پیدا کنیم طوری‌که $d(p_i, p_j)$ کمینه باشد (منظور از $d(p_i, p_j)$ فاصلهٔ دو نقطهٔ p_i و p_j است). یک راه حل بدیهی برای این مسئله با پیچیدگی زمانی $O(n^2)$ وجود دارد به این صورت که کافی است برای هر دو نقطهٔ p_i و p_j، که $\binom{n}{2}$ حالت دارد، $d(p_i, p_j)$ را حساب کنیم و بین همهٔ مقادیر به‌دست آمده کمینه را پیدا کنیم. اما مانند اکثر سوالات تقسیم و حل ما به دنبال راه حلی با پیچیدگی زمانی $O(n \log n)$ هستیم.

ابتدا حالت یک بعدی را بررسی می‌کنیم. کافی است نقاط را مرتب کنیم سپس فاصلهٔ هر دو نقطهٔ متوالی را حساب کنیم و بین آن‌ها کمینه را پیدا کنیم. از آن‌جا که مرتب کردن نقاط به زمان $O(n \log n)$ و پیدا کردن کمینه به زمان $O(n)$ نیاز دارد، پیچیدگی زمانی کل الگوریتم $O(n \log n)$ است.

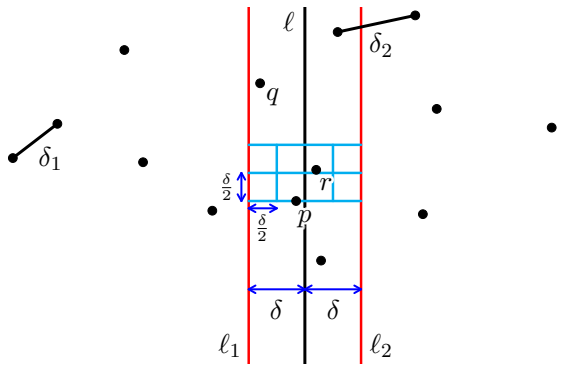
حال به مسئلهٔ اصلی باز می‌گردیم. شاید این ایده به ذهن برسد که نقاط را برحسب مشخصهٔ x (y) آن‌ها مرتب کنیم و امیدوار باشیم آن دو نقطه‌ای که کم‌ترین فاصله را دارند، مشخصهٔ x (y) آن‌ها نیز به هم نزدیک باشد. اما حالات بسیاری وجود دارد که این اتفاق نمی‌افتد.

[1] Divide and Conquer (and Combine)
[2] Merge Sort
[3] Quick Sort

نقاط را برحسب مشخصهٔ x مرتب می‌کنیم، $\frac{n}{2}$ تای اول را در یک دسته قرار می‌دهیم و بقیهٔ نقاط را در دستهٔ دوم (تقسیم) که دو دسته به وسیلهٔ یک خط از یک‌دیگر جدا شده‌اند سپس مسئله را به طور بازگشتی برای هر دو دسته حل می‌کنیم (حل). اما در این مسئله قسمت تلفیق چندان بدیهی نیست. فرض کنید کمینهٔ فاصله در دستهٔ اول δ_1 و در دستهٔ دوم δ_2 باشد. دقت کنید که تنها باید فاصلهٔ زوج نقاطی که یکی از آن‌ها در دستهٔ اول و دیگری در دستهٔ دوم قرار دارد را بررسی کنیم زیرا بقیهٔ زوج نقاط در قسمت بازگشتی بررسی شده‌اند. همچنین تعداد این زوج نقاط تقریباً $\frac{n^2}{4}$ است پس ما نمی‌توانیم همهٔ آن‌ها را بررسی کنیم. قرار می‌دهیم $\delta = \min\{\delta_1, \delta_2\}$.



مانند شکل بالا دو خط به موازاتِ خط تقسیم‌کنندهٔ دو دسته و به فاصلهٔ δ از آن رسم می‌کنیم. هر نقطه‌ای که خارج از نوارِ به وجود آمده بین دو خط ℓ_1 و ℓ_2 قرار دارد فاصله‌ای بزرگ‌تر از δ از نقاط طرف دیگر خط ℓ دارد پس تنها کافی است نقاط داخل نوار را بررسی کنیم. توجه کنید که ممکن است همهٔ نقاط، داخل نوار قرار بگیرند یعنی لزوماً وجود این نوار تعداد زوج نقاطی که باید بررسی کنیم را کاهش نمی‌دهد. حال ایده‌ای مشابه با حالت یک بعدی را استفاده می‌کنیم. در حالت یک بعدی هر نقطه را با نقطهٔ بعدی در نظر می‌گرفتیم، نشان می‌دهیم در این‌جا اگر نقاط بین نوار را برحسب مشخصهٔ y مرتب کنیم، کافی است هر نقطه را با ۷ نقطهٔ بعدی در نظر بگیریم. یک نقطهٔ دلخواه مانند p را در نظر می‌گیریم و نوار را مانند شکل زیر تقسیم می‌کنیم:



در واقع خطوط به شکلی رسم شده‌اند که نقطهٔ p روی خط پایین قرار داشته باشد. دقت کنید که در هیچ مربعی نمی‌تواند دو نقطه وجود داشته باشد زیرا در این‌صورت فاصلهٔ آن‌ها از هم حداکثر به اندازهٔ قطر مربع یعنی $\frac{\sqrt{2}}{2}\delta$ است که از δ کوچک‌تر است اما می‌دانیم فاصلهٔ زوج نقاطی که یک سمت خط ℓ قرار دارند حداقل δ است و این تناقض است. هم‌چنین فاصلهٔ نقاطی که بالاتر از مستطیل رسم شده قرار دارند (مانند q)، از p حداقل برابر با δ است. پس تنها کافی است فاصلهٔ نقاطی که درون یکی از مربع‌ها قرار دارند(مانند r)، از p را حساب کنیم و بین همهٔ مقادیر به‌دست آمده کمینه را پیدا کنیم. جواب نهایی مسئله، کمینهٔ این عدد و δ است.

دقت کنید که برای نوشتن الگوریتم این سوال ما نیازی به کشیدن این خطوط نداریم. در واقع برای مرحلهٔ تلفیق همهٔ نقاط داخل نوار را پیدا می‌کنیم، بر حسب مشخصهٔ y مرتب می‌کنیم و از پایین به بالا فاصلهٔ هر نقطه با ۷ نقطهٔ بعد از آن را حساب می‌کنیم. توضیحات بالا به ما کمک می‌کند تا بفهمیم چرا این الگوریتم درست کار می‌کند.

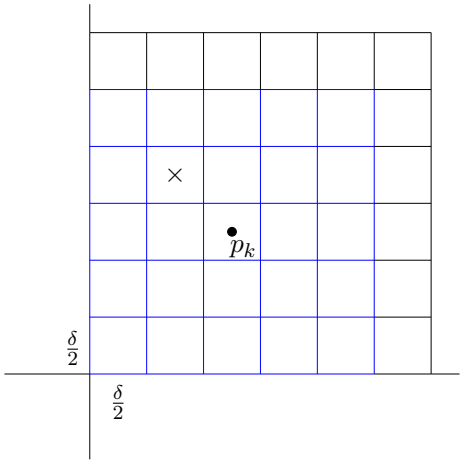
حال توجه کنید که نیاز داریم مراحل تقسیم و تلفیق پیچیدگی زمانی $O(n)$ داشته باشند اما اگر بخواهیم هر بار نقاط را مرتب کنیم به زمان $O(n \log n)$ نیاز داریم. برای حل این مشکل ابتدا یک بار نقاط را برحسب مشخصهٔ y و یک بار برحسب مشخصهٔ x مرتب می‌کنیم. در مرحلهٔ تقسیم یک بار روی آرایهٔ مرتب‌شده برحسب x حرکت می‌کنیم و مرتب‌شدهٔ نقاطی که در این مرحله داریم را پیدا می‌کنیم (کافی اسن اندیس نقاط را ذخیره کنیم). به‌طور مشابه برای مرحلهٔ تلفیق روی آرایهٔ مرتب‌شده برحسب y حرکت می‌کنیم و نقاطی که مشخصهٔ x آن‌ها بین دو خط ℓ_1 و ℓ_2 قرار می‌گیرد را جدا می‌کنیم. پس در هر دو مرحله تنها نیاز داریم هزینهٔ $O(n)$ بدهیم. پیچیدگی زمانی الگوریتم بازگشتی را با $T(n)$ نشان می‌دهیم. داریم

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

که طبق قضیهٔ اساسی نتیجه می‌شود $T(n) = O(n \log n)$ و با توجه به این که دو هزینهٔ $O(n \log n)$ نیز در ابتدای الگوریتم داده‌ایم، پیچیدگی زمانی کل الگوریتم $O(n \log n)$ می‌شود.

در ادامه یک الگوریتم تصادفی با زمان متوسط $O(n)$ برای این مسئله ارائه می‌دهیم. ایدهٔ کلی الگوریتم این است که یک ترتیب تصادفی از نقاط مانند $p_1, p_2, \ldots, p_n$ را در نظر می‌گیریم و به همین ترتیب نقاط را بررسی کنیم، به این صورت که اگر به نقطهٔ p_k رسیدیم مسئله را برای نقاط $p_1, p_2, \ldots, p_k$ حل کنیم و در هر مرحله به‌طور متوسط هزینهٔ $O(1)$ بدهیم. اگر بتوانیم این کار را انجام دهیم واضح است که پیچیدگی زمانی کل الگوریتم $O(n)$ است. حال برای این مسئله هدف این است که به هر نقطهٔ p_k که رسیدیم تنها نقاط نزدیک آن (مثلاً یک دایره به شعاع r) را بررسی کنیم و چالشی که داریم این است که چگونه این الگوریتم را پیاده‌سازی کنیم.

فرض کنید کل نقاط ما در یک مربع 1×1 قرار دارند که گوشهٔ پایین چپ آن مبدا مختصات است. این کار با تغییر مقیاس‌بندی و مکان محورهای مختصات به‌سادگی قابل انجام است. فرض کنید کوتاه‌ترین فاصله‌ای که تا الان پیدا کردیم δ است. مربع 1×1 را با مربع‌های $\frac{\delta}{2} \times \frac{\delta}{2}$ شبکه‌بندی می‌کنیم.



به هر مربع شماره‌ای نسبت می‌دهیم که نشان‌دهندهٔ سطر و ستونی است که در آن قرار دارد و برای مثال در شکل بالا به مربعی که در آن $\times$ وجود دارد شمارهٔ $(2, 4)$ را نسبت می‌دهیم. مانند الگوریتم تقسیم و حل، در این‌جا نیز در هر مربع حداکثر یک نقطه قرار دارد (به غیر از خود p_k) و هر نقطه‌ای خارج از ۲۵ مربع اطراف p_k (مربع‌های کشیده شده با رنگ آبی در شکل بالا) قرار داشته باشد فاصله‌ای بیش‌تر از δ از p_k دارد. دقت کنید که این مربع‌ها را در زمان $O(1)$ می‌توانیم پیدا کنیم زیرا تنها کافی است مربعی که p_k در آن قرار دارد را پیدا کنیم. برای این، دو مشخصهٔ p_k را بر $\frac{\delta}{2}$ تقسیم می‌کنیم و شمارهٔ مربعی که p_k در آن قرار دارد به‌دست می‌آید. پس اگر بتوانیم در زمان $O(1)$ تشخیص دهیم که در هر کدام از ۲۵ مربع مشخص‌شده نقطه‌ای قرار دارد یا نه، فاصلهٔ p_k از حداکثر ۲۵ نقطه را حساب می‌کنیم و کمینهٔ آن‌ها را با δ مقایسه می‌کنیم. همهٔ این مراحل به هزینهٔ $O(1)$ نیاز دارد. برای ذخیره کردن این‌که در هر مربع نقطه‌ای وجود دارد یا نه از جدول درهم‌سازی[۴] استفاده می‌کنیم. در واقع شمارهٔ مربع‌هایی که در آن‌ها نقطه قرار دارد را در جدول درهم‌سازی ذخیره می‌کنیم. توجه کنید که

[4] Hash Table

ما همواره یک عدد را در جدول درهم‌سازی ذخیره می‌کردیم اما این‌جا یک زوج مرتب داریم. برای حل این مشکل می‌توانیم به‌سادگی هر زوج مرتب را به یک عدد نظیر کنیم. به عنوان مثال اگر مربع ما m سطر دارد به زوج مرتب (a, b) عدد $ma + b$ را نظیر می‌کنیم. همچنین برای هر مربع یک اشاره‌گر[۵] به نقطه‌ای که درون آن قرار دارد نیز تعریف می‌کنیم. در این‌صورت به‌طور متوسط در زمان $O(1)$ می‌توانیم همهٔ نقاطی که در ۲۵ مربع اطراف p_k قرار دارند را پیدا کنیم.

به این نکته اشاره می‌کنیم که ساختن شبکه‌بندی و جدول درهم‌سازی با استفاده از این فرض است که مقدار δ یک عدد ثابت است و در صورتی که مقدار δ تغییر کند دیگر هیچ‌کدام استفاده‌ای ندارند و باید شبکه‌بندی و جدول درهم‌سازی را از نو بسازیم که به‌طور متوسط هزینهٔ $O(k)$ را باید برای این کار بدهیم. نشان می‌دهیم این اتفاق به احتمال کمی رخ می‌دهد. دقت کنید که زمان اجرای الگوریتم از دو قسمت تشکیل شده است. قسمت اول کل زمان بدون در نظر گرفتن ساختن دوباره جدول درهم‌سازی است که پیچیدگی زمانی $O(n)$ دارد زیرا ما برای هر p_k هزینهٔ $O(1)$ می‌دهیم و قسمت دوم کل زمان ساختن جدول‌های درهم‌سازی جدید است که آن را T می‌نامیم. اگر نشان دهیم این قسمت نیز به‌طور متوسط در زمان $O(n)$ انجام می‌شود اثبات تمام است. توجه کنید که ما در ابتدای الگوریتم ترتیب نقاط را به شکل تصادفی در نظر گرفتیم. واضح است که زمانی δ تغییر می‌کند که بین k نقطهٔ اول، p_k یکی از دو سر پاره‌خطی باشد که کوتاه‌ترین طول را بین همهٔ زوج نقاط دارد. از آن‌جا که ترتیب نقاط تصادفی است، احتمال این اتفاق $\frac{2}{k}$ است زیرا احتمال این‌که هر کدام از دو سر کوتاه‌ترین پاره‌خط، عضو آخر دنباله باشند $\frac{1}{k}$ است. اگر فرض کنیم حداکثر هزینهٔ متوسط ck را برای ساختن جدول درهم‌سازی می‌دهیم، نتیجه می‌شود

$$\mathbf{E}[T] \leq \sum_{k=1}^{n} \frac{2}{k} \times ck = O(n)$$

پس حکمی که به دنبال آن بودیم اثبات می‌شود. دقت کنید که نقاط حتماً باید ترتیب تصادفی داشته باشند زیرا اگر ترتیب نقاط به‌طور دشمنانه انتخاب شده باشد ممکن است برای هر p_k مقدار δ تغییر کند و مجبور شویم nبار جدول درهم‌سازی را بسازیم که پیچیدگی زمانی آن $O(n^2)$ می‌شود.

به روشی که در این مسئله استفاده کردیم Randomized Incremental گفته می‌شود. در ادامه مسئلهٔ کوچک‌ترین دایرهٔ دربرگیرنده[۶] را با این روش حل خواهیم کرد.

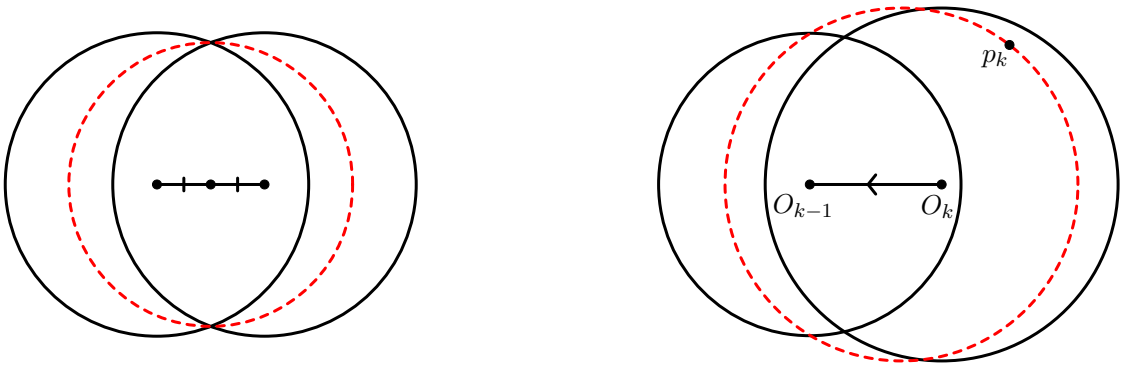
۳ مسئلهٔ کوچک‌ترین دایرهٔ دربرگیرنده

مسئله این است که تعدادی نقطه در صفحه به ما داده شده است و می‌خواهیم کوچک‌ترین دایره‌ای را پیدا کنیم که همهٔ نقاط داخل یا روی آن دایره قرار بگیرند. اولین نکته‌ای که در رابطه با این مسئله وجود دارد این است که همواره این دایره از حداقل ۲ نقطه می‌گذرد زیرا اگر از یک نقطه بگذرد می‌توانیم دایره‌ای کوچک‌تر از آن رسم کنیم که از دو نقطه می‌گذرد و همهٔ نقاط را شامل می‌شود. در واقع دایره را کوچک می‌کنیم تا به یک نقطهٔ دیگر گیر کند. نکتهٔ دوم این است که از هر سه نقطه دقیقاً یک دایره می‌گذرد، پس اگر بدانیم ۳ نقطه روی کوچک‌ترین دایره وجود دارند آن دایره به صورت یکتا تعیین می‌شود. الگوریتم بدیهی که برای این مسئله وجود دارد به این صورت است که همهٔ دوتایی‌ها و سه‌تایی‌های ممکن از نقاط را در نظر بگیریم، برای دوتایی‌ها دایره به قطر آن دو نقطه و برای سه‌تایی‌ها دایرهٔ یکتایی که از آن سه نقطه می‌گذرد را بررسی کنیم تا کوچک‌ترین دایره که همهٔ نقاط را شامل می‌شود پیدا کنیم. پیچیدگی زمانی این الگوریتم $O(n^4)$ است زیرا در کل $O(n^3)$ دایره را باید در نظر بگیریم و در $O(n)$ بررسی کنیم که آیا همهٔ نقاط داخل آن دایره قرار دارند یا نه.

حال می‌خواهیم با استفاده از روش Randomized Incremental الگوریتمی تصادفی با متوسط زمان اجرای $O(n)$ ارائه دهیم. فرض کنید $p_1, p_2, \ldots, p_k, \ldots, p_n$ یک ترتیب تصادفی از نقاط باشد. کوچک‌ترین دایره‌ای که نقاط $p_1, p_2, \ldots, p_m$ را شامل می‌شود، را D_m می‌نامیم. فرض می‌کنیم D_{k-1} را داریم و می‌خواهیم p_k را به نقاط $p_1, p_2, \ldots, p_{k-1}$ اضافه کنیم. اگر p_k داخل دایرهٔ D_{k-1}

[۵] Pointer
[۶] Smallest Enclosing Circle

قرار داشته باشد که لازم نیست کار خاصی انجام دهیم و قرار می‌دهیم $D_k = D_{k-1}$. پس فرض می‌کنیم p_k خارج از D_{k-1} قرار دارد و باید دایرهٔ جدیدی بسازیم. فرض کنید که D_k از p_k نگذرد. به وضوح D_k از D_{k-1} بزرگ‌تر است. مرکز D_k را O_k و مرکز D_{k-1} را O_{k-1} می‌نامیم. دقت کنید که بقیهٔ نقاط به جز p_k در اشتراک دو دایره قرار دارند. حال اگر O_k را روی خط O_kO_{k-1} به سمت O_{k-1} حرکت دهیم، اشتراک دو دایره همچنان نقاط قبلی را شامل می‌شود و بزرگ‌تر نیز می‌شود. می‌توانیم این کار را تا جایی ادامه دهیم که p_k روی مرز D_k قرار بگیرد. پس می‌توانستیم از همان اول فرض کنیم که D_k از p_k می‌گذرد. بد نیست به این نکته اشاره کنیم که کوچک‌ترین دایره‌ای که همهٔ نقاط را شامل می‌شود همواره یکتا است و اگر دو دایره وجود داشته باشد می‌توانیم دایرهٔ کوچک‌تری بسازیم که همهٔ نقاط را شامل شود.



پس حال باید همان مسئلهٔ قبلی را با این فرض اضافه که p_k روی محیط دایره قرار دارد، حل کنیم. مانند قبل فرض می‌کنیم برای نقاط $p_1, p_2, \ldots, p_{i-1}$ $(i < k)$ مسئله حل شده است و می‌خواهیم نقطهٔ p_i را اضافه کنیم. اگر نقطهٔ p_i داخل دایره باشد که دایره تغییری نمی‌کند اما اگر p_i خارج از دایره باشد باید کوچک‌ترین دایره‌ای را پیدا کنیم که از p_i و p_k می‌گذرد و همهٔ نقاط را شامل می‌شود. پس به مسئله‌ای جدید می‌رسیم با این فرض اضافه که p_i و p_k روی محیط دایره قرار دارند. فرض می‌کنیم برای نقاط $p_1, p_2, \ldots, p_{j-1}$ $(j < i)$ مسئله حل شده است و می‌خواهیم نقطهٔ p_j را اضافه کنیم. اگر نقطهٔ p_j داخل دایره باشد که دایره تغییری نمی‌کند اما اگر p_j خارج از دایره باشد باید کوچک‌ترین دایره‌ای را پیدا کنیم که از p_j، p_i و p_k می‌گذرد و همهٔ نقاط را شامل می‌شود که این دایره یکتا تعیین می‌شود و الگوریتم متوقف می‌شود. برای درک بهتر از این الگوریتم شبه کد آن را می‌نویسیم:

SEC $(p = p_1, p_2, \ldots, p_n)$

1 Let $p_1, p_2, \ldots, p_n$ be a random permutation of p_i's
2 $D_2 =$ smallest circle containing p_1, p_2
3 **for** $i = 3$ **to** n
4 **if** $p_i \in D_{i-1}$
5 $D_i = D_{i-1}$
6 **else**
7 $D_i = \text{SEC1}(i-1,\ p_i)$
8 **return** D_n

```
SEC1 (k, q)
1  D_1 = smallest circle containing p_1, q
2  for i = 2 to k
3      if p_i ∈ D_{i-1}
4          D_i = D_{i-1}
5      else
6          D_i = SEC2(i − 1, p_i, q)
7  return D_k
```

```
SEC2 (k, q_1, q_2)
1  D_0 = smallest circle containing q_1, q_2
2  for i = 1 to k
3      if p_i ∈ D_{i-1}
4          D_i = D_{i-1}
5      else
6          D_i = Circle having p_i, q_1, q_2 on its boundary
7  return D_k
```

توجه کنید که در تابع SEC2 دیگر بازگشتی نداریم زیرا دایره‌ای که از نقاط p_i، q_1 و q_2 می‌گذرد به‌طور یکتا تعیین می‌شود. حال زمان اجرای الگوریتم را محاسبه می‌کنیم. واضح است که زمان اجرای تابع SEC2، $O(k)$ است. برای تابع SEC1، اگر زمان‌هایی که در else صرف می‌شود را در نظر نگیریم زمان اجرای آن $O(k)$ است. مانند تحلیلی که در مسئلهٔ پیدا کردن نزدیک‌ترین زوج نقطه انجام دادیم، احتمال این‌که p_i خارج از دایره بیفتد $O\left(\frac{1}{i}\right)$ است پس به‌طور متوسط هزینه‌ای که برای else می‌دهیم نیز $O(1)$ است پس کل زمان اجرای تابع SEC1 نیز به‌طور متوسط $O(k)$ است و به‌طور مشابه ثابت می‌شود زمان اجرای تابع SEC، به‌طور متوسط $O(n)$ است که همان زمان اجرایی است که به دنبال آن بودیم.

با استفاده از روش Randomized Incremental می‌توان مسئلهٔ برنامه‌ریزی خطی[۷] که بعداً با آن آشنا خواهیم شد، را در ابعاد پایین حل کرد.

[7] Linear Programming

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۳: الگوریتم های پیمایش گراف ها

نگارنده: مهدی مستانی

در جلسه قبل،با مفهوم تقسیم و حل آشنا شدیم. در این جلسه و جلسه بعد قصد داریم راجع به الگوریتم های پیمایش در گراف [۱] ها صحبت کنیم.

۱ نمونه‌ای از مسائل

مسئله بدین صورت است که گرافی (جهت دار یا بدون جهت)داریم و میخواهیم گراف را **پیمایش**[۲] کنیم؛

منظور از پیمایش در حالت کلی به معنی گذشتن و دیدن تمام رأس های[۳]گراف است و معمولا پیمایش از طریق یال‌های موجود در گراف انجام می‌شود

مسائلی که در این مبحث با آنها برخورد داریم، معمولا الگوریتم های مشابه‌ای دارند.

به طور کلی مسائل بدین صورت هستند:

۱) **ورودی** : گراف $G = (V, E)$ و یک رأس از گراف مانند $s \in V$ به عنوان ورودی داده می‌شود.

خروجی : از رأس s به چه رأس هایی می‌توان رسید؟

۲) **ورودی** : گراف $G = (V, E)$ و دو رأس از گراف مانند $s, t \in V$ به عنوان ورودی داده می‌شود.

خروجی : آیا از رأس s به رأس t مسیری[۴] وجود دارد یا خیر؟

به این مسئله دسترس پذیری[۵] گویند.

و مسائل دیگری از این نوع که در ادامه تعدادی از آن‌ها را مورد بررسی قرار می‌دهیم.

۲ کاربرد

این نوع مسائل در دنیای امروز کاربرد زیادی دارند،زیرا که گراف امروزه کاربرد های زیادی دارد.برخی از کاربرد های این نوع مسائل بدین صورت است:

[1] graph
[2] Graph traversal
[3] node
[4] path
[5] reachability

۱)کوتاه ترین مسیر بین دو نقطه از شهر، چیست؟

۲)در مبحث Web crawling کاربرد دارد؛مثلاً از صفحه ای شروع کرده و سعی کنیم کل صفحات وب را به طور هوشمندی پیمایش کرده و صفحات تکراری را بررسی نکنیم و مانند آن

۳) زباله روب[۶] در برخی زبان های برنامه نویسی ، از این روش برای حذف کردن فضای هایی است که برنامه فعلی دیگر به این حافظه ها دسترسی ندارد.

۴) مسئله ای داشته باشیم که بخواهیم تعداد زیادی حالت را برای سوالی بررسی کنیم؛ مثلا بخواهیم مکعب روبیک را حل کنیم.بدین صورت که یک حالت خاص از مکعب روبیک را به ما داده اند و ما بخواهیم عملیات هایی را بیابیم که بتوانیم به مکعب روبیک درست شده برسیم. در اینجا می‌توانیم به هر حالت از مکعب روبیک، یک رأس نسبت دهیم و اگر بتوانیم از حالتی از مکعب روبیک با یک تغییر وضعیت در مکعب ، به حالت دیگر از مکعب روبیک برسیم، بین این دو رأس یال رسم می‌کنیم و هدف این است که به رأسی برسیم که متناظر با حالتی است که مکعب ، کامل است.

و مسائل دیگری از این دست.

۳ نمایش و ذخیره گراف در کامپیوتر

برای نمایش و ذخیره سازی گراف در کامپیوتر روش های مختلفی داریم که برخی از آن ها را معرفی می‌کنیم:

۱) **ماتریس مجاورت**[۷] : اگر گراف n رأسی داشته باشیم ، ماتریسی $n \times n$ را نگه داری کنیم بدین صورت که درایه (i, j) در ماتریس مجاورت ۱ است اگر بین رأس i و j یالی باشد در غیر اینصورت ۰ است.

مزایا: بررسی وصل بودن دو رأس به یکدیگر(وجود یال بین دو رأس) سریع است(در زمان $O(1)$)

معایب: حافظه زیادی برای ذخیره کردن ، مصرف می‌کند($\Omega(n^2)$) در صورتی که مثلا برای نمایش یک درخت،می‌توان این اطلاعات را در حافظه بسیار کمتری نگه داری کرد. همینطور لیست کردن همسایه های هر رأس $\Omega(n)$ زمان می‌برد.

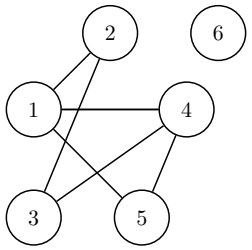
۲) **لیست مجاورت**[۸] : فرض کنید که آرایه ای داریم (با $\mathrm{Adj}[n]$ نمایش می‌دهیم)که هر خانه از این آرایه متناظر با یک رأس است.(فرض کنید رأس ها را با شماره های $1, 2, \ldots, n$ شماره دهی کرده ایم.) حال درایه $\mathrm{Adj}[i]$ ، یک لیست پیوندی است که همسایه های رأس i ام را در خود نگه‌داری می‌کند.

مزایا: حافظه مصرفی $O(|V| + |E|)$ است که در آن V مجموعه رئوس و E مجموعه یال های گراف است.

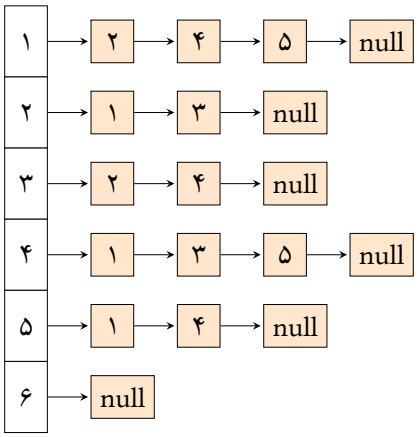
لیست کردن همسایه های یک رأس سریع است.مثلا برای رأس i ام ، برابر $O(deg(i))$ است که در آن $deg(i)$ درجه[۹] رأس i ام است.

معایب: بررسی اینکه دو رأس به یکدیگر یال دارند یا خیر، ممکن است که در زمان سریع ($O(1)$) عملی نباشد و بیشتر از آن زمان ببرد.

[6] garbage collector
[7] Adjacency matrix
[8] adjacency list
[9] degree



شکل ۱: گراف ساده



شکل ۲: لیست مجاورت گراف ساده ۱

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

شکل ۳: ماتریس مجاورت گراف ساده ۱

۳) نمایش ضمنی[10]

این نوع نمایش بدین گونه نیست که کل گراف را در حافظه کامپیوتر ذخیره کرده باشیم.مثلاً تابعی داریم که این تابع به عنوان ورودی یک رأس از گراف مانند v را به عنوان ورودی می‌گیرد و همسایه های آن رأس در گراف را بر می‌گرداند.این حالت برای مواقعی است که گراف ما بزرگ باشد.از مزیت‌های این روش می‌توان به حافظه بسیار کمی می‌گیرد ، اشاره کرد ؛ زیرا که فقط یک تابع داریم و باید تنها اطلاعات آن را ذخیره سازی کنیم.

به عنوان یک مثال، مسئله درست کردن مکعب روبیک را در نظر بگیرید. تعداد حالت هایی که این مکعب روبیک می‌تواند داشته باشد بسیار زیاد است(و ذخیره سازی همه آن ها تقریبا غیر عملی است).اما مثلاً اگر یک حالت خاص از مکعب روبیک را به ما بدهند،می‌توانیم تابعی بنویسیم که همه چرخش های ممکن را بررسی کنیم و آن را به عنوان خروجی بدهیم (در اینجا حالت های مکعب روبیک رأس ها هستند و رسیدن از یک چرخش به حالت دیگر، یال ها هستند).

۴ الگوریتم جستجوی سطح اول[11]

حال می‌خواهیم بررسی کنیم که پیمایش روی گراف را چگونه انجام دهیم . وقتی که می‌خواهیم بررسی کنیم از یک رأس دلخواه مانند s به چه رأس‌هایی می‌توانیم برسیم،یک راهکار این است که قدم به قدم جلو برویم؛یعنی اینکه از رأس s شروع می‌کنیم و بررسی می‌کنیم که از رأس s به چه رأس های دیگری می‌توانیم با یک گام برسیم.این رأس‌ها را ،رأس‌های مرحله‌ی[12]یک می‌نامیم.حال سراغ رأس های مرحله یک رفته بررسی می‌کنیم که از این رأس ها به چه رأس هایی می‌توانیم برسیم(ممکن است شامل اشتراکاتی باشند که در اینجا مهم نیست)؛ در این مرحله به تعدادی رأس جدید می‌رسیم. این رأس‌های مرحله جدید را ، رأس‌های مرحله ۲ می‌نامیم و همین کار را برای رأس های مرحله ۲ انجام می‌دهیم و همینطور الی آخر .

ابتدا برای هر رأس، مرحله آن را برابر با مقدار (۱ــ) قرار می‌دهیم ؛ یعنی :

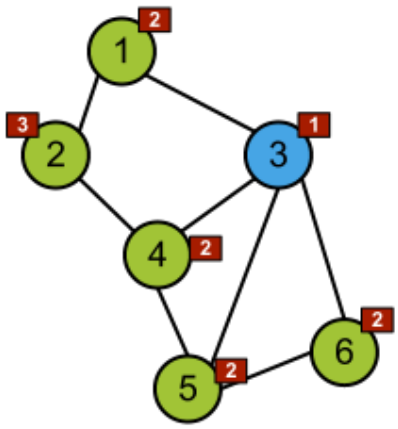
for each $u \in V$: u.level= -1

حال شبه کد زیر را برای الگوریتم بالا پیاده سازی می‌کنیم:

[10] implicit
[11] Breadth First Search
[12] level

```
BFS(node v)
1   v.level=0
2   i = 0
3   current={v}
4   while current not empty
5       next={}
6       i = i + 1
7       for each v in current
8           for each u in Adj[v]
9               if (u.level= -1)
10                  u.level=i
11                  u.parent=v
12                  next.add(u)
13      current=next
```

نمونه ای از عملیات BFS که روی رأس شماره ۳ اجرا شده است را مشاهده می‌فرمایید:



توضیح این الگوریتم بدین صورت است که در ابتدا ، مرحلهٔ رأسی که از آن شروع می‌کنیم را برابر با صفر و آن را در $current$ قرار می‌دهیم. تابع مذکور ، ابتدا یک رأس ریشه مانند v را به عنوان شروع می‌گیرد. سپس رأس‌ها را سطح بندی می‌کند. سطح بندی به این صورت است که تمام رأس‌های مجاور v را در سطح اول قرار میدهیم، حال برای سطح $i+1$ ، رأس u که مجاور یکی از رئوس سطح i است را در این سطح قرار میدهیم به شرطی که u در هیچ سطح دیگری نیامده باشد. به عبارت دیگر رأس هارا بر اساس کوتاه‌ترین فاصله از v سطح بندی میکنیم. حال برای پیمایش به ترتیب سطح، و در هر سطح به ترتیب دلخواه وارد رأس‌ها میشویم. پس اولین زمانی که به هر رأس می‌رسیم، با کمترین فاصله ممکن نسبت به v رسیده ایم.

این الگوریتم هم برای گراف های جهت دار کار می‌کند و هم گراف های بدون جهت.در بعضی الگوریتم ها که برای گراف های جهت دار است،می‌توانیم یال های بدون جهت را،به صورت دو یال جهت دار در دو جهت متفاوت در نظر گرفت.

نکاتی پیرامون الگوریتم :

۱ـ زمان اجرای این الگوریتم برابر با $O(|V|+|E|)$ است ؛ زیرا که هر رأسی یک بار بررسی می‌شود و هیچ یالی بیش از دو بار بررسی نمی‌شود.همینطور حافظه مصرفی اضافه در بدترین حالت $\Theta(|V|)$ است که صرف نگه داری مقادیر در $current$ و $next$ می‌شود.

۲ـ در پایان الگوریتم ، مشخصهٔ $u.level$ برای هر رأس u ، برابر با طول کوتاهترین مسیر از u به v است.

ادعا: در پایان اجرای k ام از حلقه‌ی while ، رأس های با فاصله حداکثر k از s به درستی برچسب[۱۳]گذاری شده اند و $current$ دقیقاً شامل رأس‌هایی است که فاصله s تا آن‌ها k است.

اثبات : این ادعا با استفاده از استقرا به راحتی ثابت می‌شود.پایه استقرا واضح است.فرض کنید که فاصله s تا رأسی $k+1$ باشد،اگر مسیری از این رأس تا رأس s که به طول $k+1$ است را تا یال یکی به مانده به آخر طی کنیم،به رأسی می‌رسیم که فاصله s تا آن رأس برابر k است و طبق فرض استقرا این رأس ، در مرحله k ام اجرای حلقه $while$ در $current$ است و در مرحله $k+1$ ام این رأس جدید در لیست فعلیِ $current$ اضافه می‌شود و هر رأسی که به این لیست اضافه می‌شود،این ویژگی ها را دارد.پس ادعا ثابت می‌شود.

۳ـ کاربرد مشخصهٔ $parent$ برای این است که بخواهیم خود مسیر با کوتاهترین فاصله را چاپ کنیم.یعنی اگر برای گراف ، یال هایی را بگیریم که در مجموعهٔ $F=\{(u,u.parent)|\ \forall u\in V, u.parnet\neq null\}$ باشند، گراف حاصل درخت است. (اگر s را رأسی در نظر بگیریم که در ابتدای پیمایش از آن شروع به حرکت کردن کرده ایم، آنگاه توجه داریم که رئوسی که $u.parnet=null$ است، شامل رئوسی است که از رأس s به آنها مسیری وجود ندارد و در واقع در مولفه همبندی شامل s قرار ندارند.)

حال اگر فرض کنیم که $V_\pi=\{u.parent\neq null|\forall u\in V_G\}\cup\{s\}$ و $E_\pi=\{(u,u.parent)|\ \forall u\in V_\pi-\{s\}\}$ آنگاه گراف $G_\pi=(V_\pi,E_\pi)$ درختی است که در آن اگر مسیر s به v برای هر $v\in V$ را در نظر بگیریم،این مسیر کوتاه ترین از s به v است؛ این درخت را ،(درخت کوتاه ترین مسیر)[۱۴]می‌نامیم.این ادعا نیز به طور استقرایی ثابت می‌شود؛یعنی فرض می‌کنیم که هنگامی که حلقه $while$ را k بار اجرا کرده ایم،برای رئوسی که فاصله s تا آن رئوس حداکثر k است،چنین درختی داریم(که مسیر s به v کوتاه ترین مسیر از s به v است.) حال هنگامی که حلقه $while$ برای $k+1$ امین بار اجرا می‌شود،هر رأس که $u.level=k+1$ است،یک یال به رئوسی دارد که $t.level=k$ هستند و بنابر این در آن درخت،مسیری به طول k از s به t وجود داشت،حال یک یال هم از t به u داریم ، پس مسیر به طول $k+1$ از s به u داریم.پس به طور استقرایی می‌توانیم بیان کنیم که این درخت برای همه رئوس این ویژگی را دارد. بنابراین می‌توانیم تابعی بازگشتی بنویسیم که با داشتن $parent$ برای هر رأس، کوتاهترین مسیر از s به هر رأس را چاپ کند.

دیگر کاربرد نگه داری $parent$ این است که اگر گراف ، همبند نباشد، می‌توانیم مولفه های‌همبندی اش را پیدا کنیم؛ بدین صورت که در ابتدا $\forall u\in V: u.level=-1$ را اجرا می‌کنیم. سپس از یک رأس دلخواه مانند v ، یک بار الگوریتم جستجوی سطح‌اول(BFS) را بر روی راس v اجرا می‌کنیم ؛ این کار مولفهٔ همبندی v را می‌دهد.سپس به سراغ رأس دیگری می‌رویم که در مولفه همبندی v نیامده و همین کار را تکرار می‌کنیم تا دیگر $level$ برای هیچ رأسی برابر با (۱ـ) نباشد. شبه کد این کار به صورت زیر است:

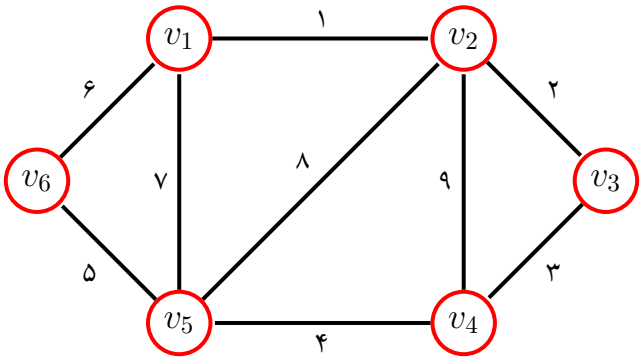
```
1  numOfComponent=0
2  for each u ∈ V
3        u.level= -1
4  for each v ∈ V
5       if (v.level= -1)
6             BFS(v)
7             numOfComponent++
```

[13] label
[14] shortest path tree

نکته اضافی : می‌توانیم مسئله دسترس پذیری در گراف بدون جهت را با $O(1)$ متغیر سراسری حل کرد که توسط Omer Reingold حل شده است. ،ولی برای گراف جهت دار حدس زده می‌شود که چنین الگوریتمی وجود نداشته باشد.

۵ کوتاه ترین مسیر در گراف وزن دار

فرض کنید در گرافی ، هر یال وزنی دارد(روی هر یال آن ، عددی قرار دارد).نمونه ای از گراف وزن دار را در شکل زیر مشاهده می فرمایید:



می‌خواهیم مسیر(در صورت وجود) با کمترین وزن بین دو رأس را بیابیم. می توانیم که مسئله را اینگونه شبیه سازی کنیم:رأس‌ها را تعدادی مهره در نظر می‌گیریم و یال ها نیز نخ هایی هستند که طول هر نخ برابر با وزن یال متناظرش می‌باشد.حال اگر بخواهیم کوتاه ترین فاصله بین دو مهره را پیدا کنیم ،دو مهره از یکدیگر دور می‌کنیم تا دیگر نتوانیم بیشتر از این مهره ها را از یکدیگر دور کنیم (در صورت بیشتر دور کردن، نخ ها پاره می‌شوند). ادعا می‌کنیم که کوتاه ترین مسیر بین دو مهره، حداکثر میزان دور کردن دو مهره از یکدیگر است. البته واضح است که کوتاه ترین مسیر بین دو مهره از این حداکثر میزان دور کردن دو مهره از یکدیگر بیشتر نیست.

یک شهود برای حل مسئله این است که فرض می‌کنیم همه مهره ها را در یک نقطه قرار داده ایم.سپس رأس(مهره) s را از مجموعه جدا کرده و آن رأس(مهره) را آن قدر دور می‌کنیم تا دیگر بیشتر از این نتوانیم دور کنیم ؛ در غیر این صورت ،نخی پاره می‌شود.رأس(مهره) غیر از s را که به این نخ متصل است را t می‌نامیم.حال مجموعه st را با یکدیگر دور می‌کنیم تا مجددا دیگر نتوانیم بیشتر از این دور کنیم و همینطور این کار را ادامه می‌دهیم تا به رأس(مهره) مورد نظر برسیم.

الگوریتم مورد نظر برای حل این سوال الگوریتم دایکسترا است.

مراجع

[1] Cormen, Thomas H., et al. *Introduction to Algorithms*. 3rd ed., MIT Press, 2009, pp. 528-539.

[2] www.geeksforgeeks.org

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۴: ادامه الگوریتم‌های پیمایش گراف‌ها (جست‌وجوی اول عمق و تعمیم آن) نگارنده: سنا نادعلی

در جلسه گذشته برای حل صورت‌های مختلف مسئله پیمایش گراف و پیداکردن فاصله دو راس، الگوریتم جست‌وجوی اول سطح را ارائه‌دادیم. در این جلسه می‌خواهیم ابتدا پیاده‌سازی دیگری از الگوریتم جست‌وجوی اول سطح را بررسی کنیم و سپس از ایده آن برای طراحی الگوریتم جست‌وجوی اول عمق استفاده کنیم. بعد از آن که تعدادی از کاربردهای این الگوریتم را دریافتیم، به تعمیم این ایده می‌پردازیم.

۱ الگوریتم جست‌وجوی اول سطح ـ پیاده‌سازی مجدد

گراف G را در نظر بگیرید. رأس s از آن داده‌شده‌است. می‌خواهیم با پیمایش گراف، فاصله هر رأس تا رأس s را مشخص کنیم.

به شبه‌کد زیر توجه کنید:

```
BFS(s)
 1  for each v ∈ V(G)
 2      v.level = −1
 3  s.level = 0
 4  Q = new Queue
 5  Q.enque(s)
 6  while Q is not empty
 7      v = Q.deque
 8      for each (v, w) ∈ E(G)
 9          if w.level == −1
10              w.level = v.level + 1
11              w.parent = v
12              Q.enque(w)
```

در خطوط ۱ تا ۴ تابع فوق، ابتدا مقدار اولیه سطح[1] هر رأس به جز رأس شروع را برابر ۱ـ قرار می‌دهیم (مقدار ۱ـ به این معناست که سطح این رأس هنوز مشخص نشده‌است). سپس برای رأس شروع مقدار اولیه را برابر ۰ قرارداده و آن را وارد صف می‌کنیم. در حلقه while هر بار یک رأس را از صف خارج کرده و برای هر همسایه‌ی آن که تا به حال سطح آن مشخص نشده، سطح و پدرش را مشخص کرده آن را وارد صف می‌کنیم.

[1] level

۲ الگوریتم جست‌وجوی اول عمق

نکته قابل توجه در الگوریتم فوق این است که داده‌ساختار صف می‌تواند با هر داده‌ساختار دیگری جایگزین شود. از اولین انتخاب‌هایی که ممکن است به ذهن برسد داده‌ساختار **پشته** [۲] است. الگوریتمی که از این جایگزینی حاصل می‌شود ـ پس از کمی تغییر ـ یکی از الگوریتم‌های مهم در پیمایش گراف‌ها را می‌سازد که الگوریتم جست‌وجوی اول عمق[۳] نامیده می‌شود.

۱.۲ پیاده‌سازی الگوریتم

پس از جایگزینی، الگوریتم به صورت زیر در می‌آید. دلیل نام‌گذاری این تابع در ادامه مشخص می‌شود.

```
DFS-Visit(s)
 1  for each v ∈ V
 2        v.mark = false
 3  s.mark = true
 4  S = new Stack
 5  S.push(s)
 6  while S is not empty
 7        v = S.pop
 8        for each (v, w) ∈ E
 9              if w.mark == false
10                    w.mark = true
11                    w.parent = v
12                    S.push(w)
```

همچنین این الگوریتم را می‌توان به صورت بازگشتی پیاده‌سازی کرد:

```
DFS-Visit(v)
1  v.mark = true
2  time = time + 1
3  v.d = time
4  for each (v, w) ∈ E(G)
5        if w.mark == false
6              w.parent = v
7              DFA-Visit(w)
8  time = time + 1
9  v.f = time
```

[2] Stack
[3] Depth-first Search

فرض کنید $time$ یک متغیر جهانی‌ست که مقدار اولیه آن **صفر** است. در این صورت $v.d$ زمان کشف[۴] رأس v و $v.f$ زمان پایان[۵] بررسی رأس v و نوادگانش است.

ادعا ۱. *الگوریتم فوق برای یک راس داده‌شده v، تمام رئوسی را که از آن قابل دسترسی هستند نشان‌دار[۶] می‌کند.*

به عنوان تمرین می‌توانید این ادعا را اثبات کنید.

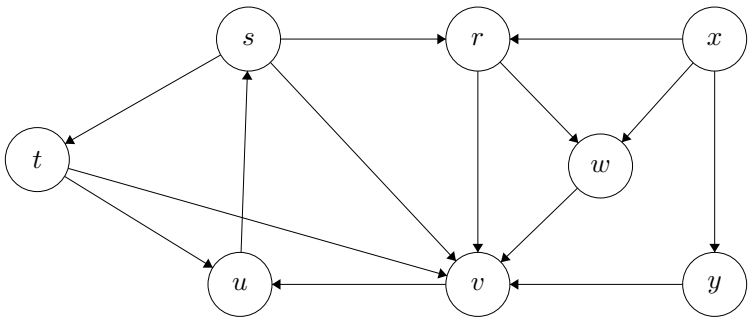
الگوریتمی که در ادامه می‌بینیم همه رئوس گراف را بررسی می‌کند. به همین دلیل معمولاً این الگوریتم را الگوریتم جست‌وجوی اول عمق می‌نامیم، نه الگوریتم فوق را.

DFS(G)

1 **for** *each* $v \in V(G)$
2 $\quad v.mark = false$
3 $time = 0$
4 **for** *each* $v \in G.V$
5 $\quad$ **if** *not* $v.mark$
6 $\quad\quad$ DFS-Visit(v)

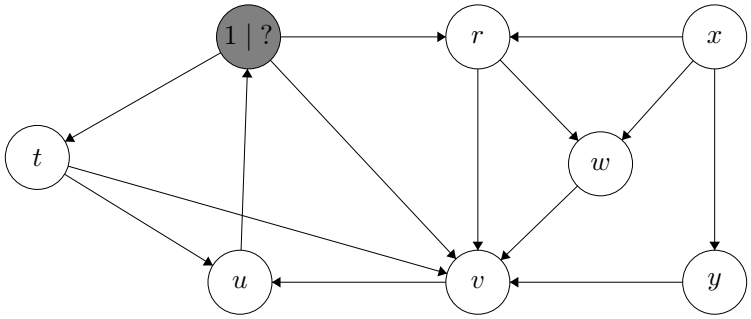
مثال: الگوریتم جست‌وجوی اول عمق را روی گراف زیر اجرا می‌کنیم:

فرض کنید s اولین رأسی باشد که تابع DFS-Visit برای آن صدا زده‌می‌شود (در هر شکل رئوس سفید رئوسی هستند که تابع DFS-Visit هنوز برای آن‌ها صدا زده‌نشده‌است. رنگ خاکستری نشان‌دهنده رئوسی است که پیش از تابع DFS-Visit برای آن‌ها صدا زده‌شده‌است اما هنوز پایان نیافته‌است. و رنگ مشکی رئوسی را مشخص می‌کند که اجرای این تابع برایشان به پایان رسیده‌است).

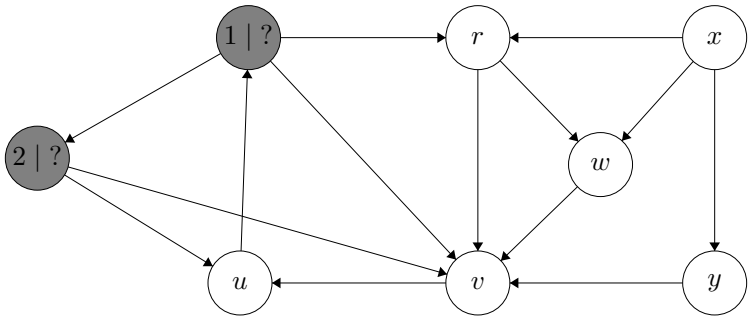


ابتدا این رأس نشانه‌گذاری می‌شود. مقدار $time$ یک واحد افزایش می‌یابد. پس $s.d = 1$ خواهد بود. حال همه یال‌های خروجی از s بررسی می‌شوند تا اگر تا به حال نشان‌دار نشده‌اند تابع DFS-Visit برایشان صدا زده‌شود. فرض کنید یال (s,t) اولین یال باشد.

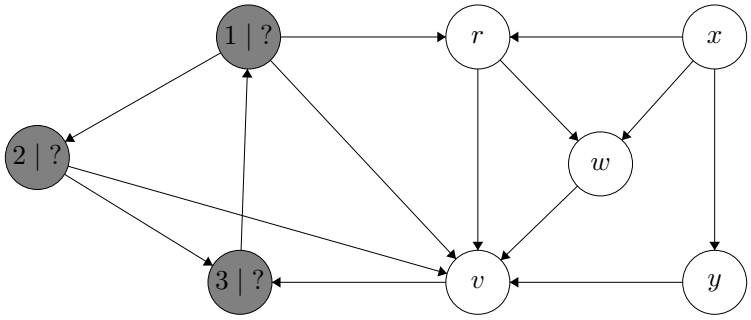
[4] discovery time
[5] finishing time
[6] *mark*



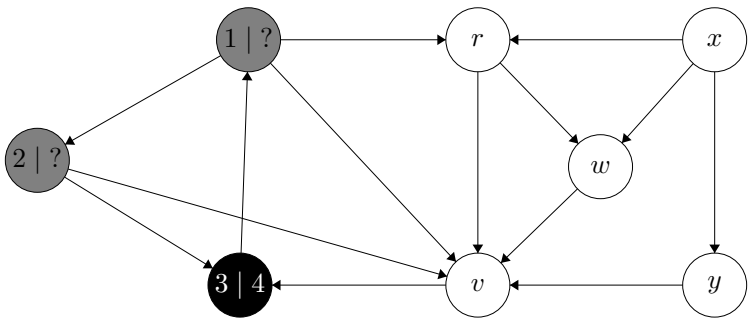
چون t هنوز نشانه‌گذاری نشده‌است تابع DFS-Visit(t) صدا زده‌می‌شود. مقدار $time$ یک واحد دیگر افزایش می‌یابد پس $t.d = 2$ و $t.parent = s$ خواهدبود. سپس همه یال‌های خروجی از t بررسی خواهندشد. فرض می‌کنیم اولین یال (t, u) باشد.



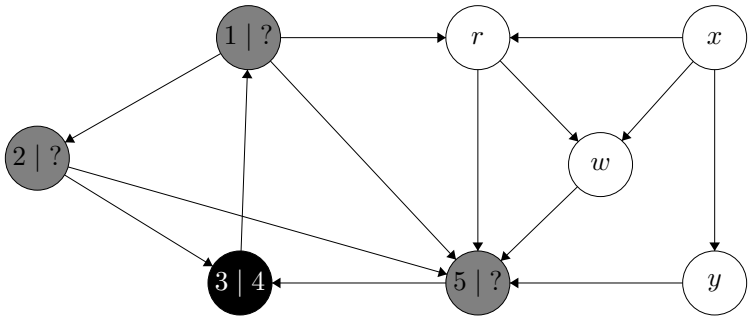
چون u هنوز نشانه‌گذاری نشده‌است تابع DFS-Visit(u) صدا زده‌می‌شود. مقدار $time$ یک واحد دیگر افزایش می‌یابد پس $u.d = 3$ و $u.parent = t$ خواهدبود. سپس تنها یال خروجی این رأس یعنی (u, s) بررسی می‌شود.



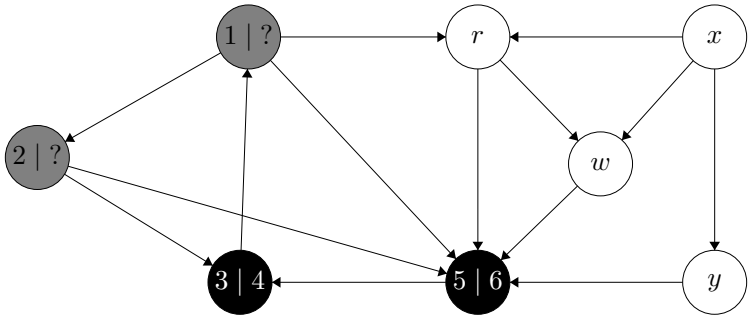
چون s پیش از این نشانه‌گذاری شده‌است بدون آن که شرط if برقرار شود حلقه for پایان می‌یابد. متغیر $time$ یک واحد دیگر افزایش میابد. پس $u.f = 4$ قرار داده‌می‌شود. اجرای DFS-Visit(u) به پایان می‌رسد. این تابع درون تابع DFS-Visit(t) صدا زده‌شده‌بود. حال با پایان اجرای این تابع دیگر یال‌های خروجی از t بررسی می‌شوند. تنها یال باقی‌مانده (t, v) می‌باشد.



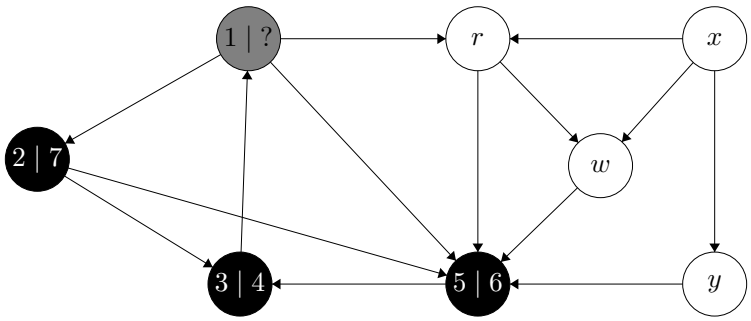
چون v هنوز نشانه‌گذاری نشده‌است تابع DFS-Visit(v) صدا زده‌می‌شود. مقدار $time$ یک واحد دیگر افزایش می‌یابد پس $v.d = 5$ و $v.parent = t$ خواهدبود. سپس تنها یال خروجی این رأس یعنی (v, u) بررسی می‌شود.



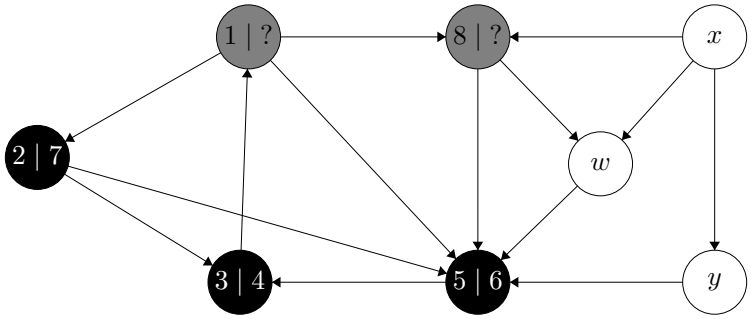
چون u پیش از این نشانه‌گذاری شده‌است بدون آن که شرط if برقرار شود حلقه for پایان می‌یابد. متغیر $time$ یک واحد دیگر افزایش می‌یابد. پس $v.f = 6$ قرار داده‌می‌شود. اجرای DFS-Visit(v) به پایان می‌رسد.



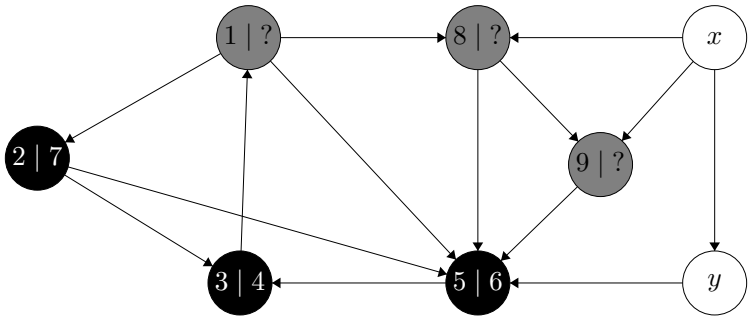
تابع DFS-Visit(v) درون تابع DFS-Visit(t) صدا زده‌شده‌بود. چون t نیز یال خروجی دیگری ندارد حلقه for پایان می‌یابد. متغیر $time$ یک واحد دیگر افزایش می‌یابد. پس $t.f = 7$ قرار داده‌می‌شود. اجرای DFS-Visit(t) نیز به پایان می‌رسد. تابع DFS-Visit(t) درون تابع DFS-Visit(s) صدا زده‌شده‌بود. بنابراین بعد از پایان آن یال‌های دیگر خروجی از s بررسی می‌شوند. s دو یال خروجی دارد. یال (s, u) و یال (s, r).



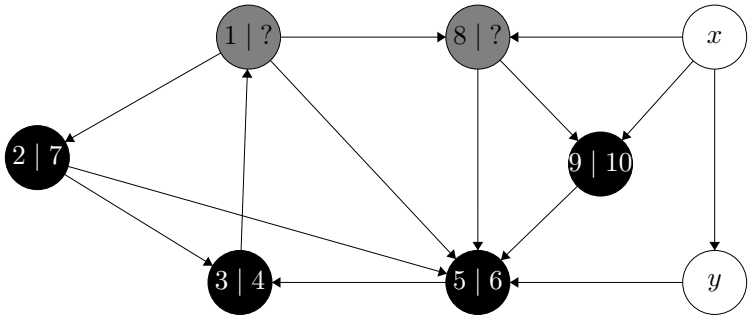
راس v پیش از این نشان‌دار شده‌است اما رأس r نه. پس تابع DFS-Visit(r) صدا زده‌می‌شود. مقدار $time$ یک واحد دیگر افزایش می‌یابد پس $r.d = 8$ و $r.parent = s$ خواهدبود.



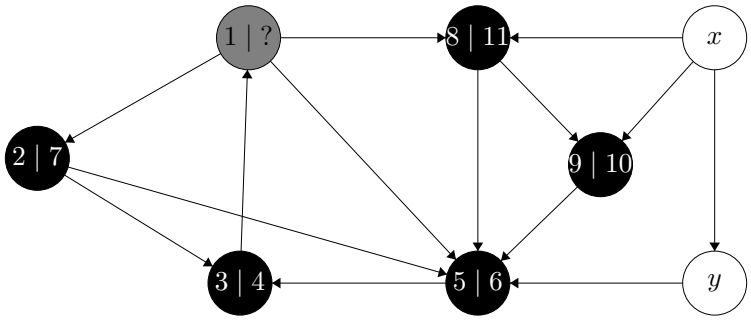
رأس r دو یال خروجی دارد. (r, v) و (r, w). فقط w نشان‌دار نشده‌است. پس تابع DFS-Visit(w) فراخوانی می‌شود. مقدار $time$ یک واحد دیگر افزایش می‌یابد پس $w.d = 9$ و $w.parent = r$ خواهدبود. سپس تنها یال خروجی این رأس یعنی (w, v) بررسی می‌شود.



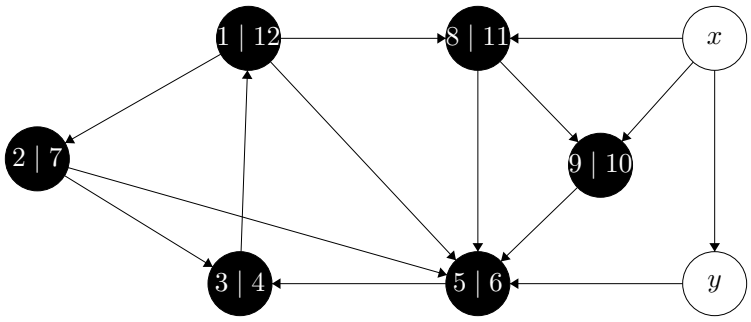
چون v پیش از این نشانه‌گذاری شده‌است بدون آن که شرط if برقرار شود حلقه for پایان می‌یابد. متغیر $time$ یک واحد دیگر افزایش می‌یابد. پس $w.f = 10$ قرار داده‌می‌شود. اجرای DFS-Visit(w) به پایان می‌رسد.



تابع DFS-Visit(w) درون تابع DFS-Visit(r) صدا زده‌شده‌بود. چون r نیز یال خروجی دیگری ندارد حلقه for پایان می‌یابد. متغیر $time$ یک واحد دیگر افزایش می‌یابد. پس $r.f = 11$ قرار داده‌می‌شود. اجرای DFS-Visit(r) نیز به پایان می‌رسد.



تابع DFS-Visit(r) درون تابع DFS-Visit(s) صدا زده‌شده‌بود. چون همه یال‌های خروجی s تا به حال بررسی شده‌اند پس حلقه for پایان می‌یابد. متغیر $time$ یک واحد دیگر افزایش می‌یابد. پس $s.f = 12$ قرار داده‌می‌شود. اجرای DFS-Visit(s) به پایان می‌رسد.



بقیه مراحل را خودتان می‌توانید به طریق مشابه دنبال کنید. دقت کنید حلقه for خط ۳ تابع DFS(G) تا به حال فقط برای یک رأس اجرا شده‌است.

⊠

۲.۲ خاصیت پرانتزی

ادعا ۲. *زمان کشف و پایان رئوس دارای خاصیت پرانتزی هستند. یعنی اگر به زمان کشف هر رأس یک پرانتز راست و به زمان خاتمه آن یک پرانتز چپ نسبت دهیم و در نهایت با توجه به تغییرات زمان، پرانتزها را کنار هم قرار دهیم، یک پرانتزگذاری صحیح حاصل می‌شود.*

به عنوان تمرین می‌توانید درستی ادعای فوق را ثابت کنید.

۳.۲ برچسب‌گذاری یال‌ها

در نظر بگیرید که الگوریتم جست‌وجوی اول عمق بر روی گراف G اجرا شده است. در این صورت یال‌های G را بعد از اجرای این الگوریتم می‌توان به ۴ دسته تقسیم کرد:

۱ـ یال‌های درختی[7] : یال‌هایی هستند که به فرم (v.parent,v) باشند. در واقع یال (u, v) یک یال درختی است اگر رأس v اولین بار هنگام بررسی یال (u, v) کشف شود. درختی که از اجتماع یال‌های درختی حاصل می‌شود، ***درخت اول عمق*** نامیده می‌شود.

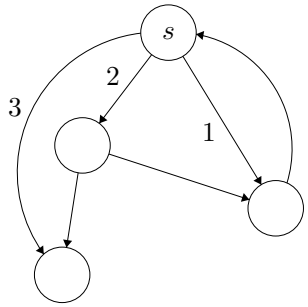
۲ـ یال‌های برگشتی[8] : یال (u, v) را برگشتی گوییم اگر رأس v از اجداد راس u باشد. توجه داریم که یک رأس جد خودش محسوب می‌شود. بنابراین طوقه‌ها همواره یال برگشتی هستند.

۳ـ یال‌های روبه‌جلو[9] : یال غیر درختی (u, v) را روبه‌جلو گویند اگر رأس v از نوادگان رأس u باشد.

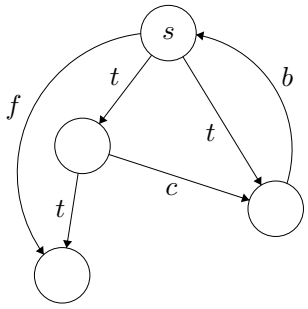
۴ـ یال‌های تقاطعی[10] : یال‌هایی هستند که جزء هیچ یک از دسته‌های بالا قرار نگیرند. به عبارت دیگر یال بین دو راسی است که هیچ یک نوه یا فرزند دیگری نباشد.

مثال: الگوریتم جست‌وجوی اول عمق را روی گراف زیر اجرا می‌کنیم (رأس شروع را s در نظر بگیرید. همچنین ترتیب یال‌هایی که از این رأس بررسی می‌شوند در شکل مشخص شده‌است.).

[7] tree edges
[8] back edges
[9] forward edges
[10] cross edges

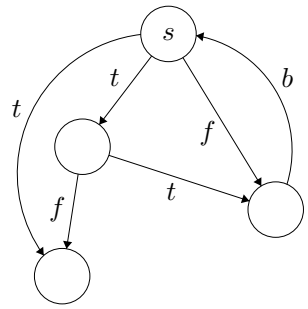


در این صورت دسته‌بندی یال‌های گراف به صورت زیر خواهدبود:



⊠

نکته: دقت داریم که ترتیب بررسی یال‌ها در شکل درخت و نوع یال‌ها تاثیر دارد. برای مثال اگر یال‌های گراف فوق را به ترتیب برعکس بررسی می‌کردیم درخت نهایی و نوع یال‌ها به صورت زیر می‌بود:



نکته: می‌توان نشان داد بعد از اجرای الگوریتم جست‌وجوی اول عمق روی یک گراف بی‌جهت، فقط یال‌های درختی و برگشتی ایجاد خواهدشد.

حال می‌خواهیم بررسی کنیم که چگونه می‌توان نوع هر یال را هنگام اجرای االگوریتم جست‌وجوی اول عمق مشخص کرد. رنگ‌بندی زیر را برای رئوس در نظر بگیرید:

- به هر راس قبل از اجرای الگوریتم رنگ سفید نسبت می‌دهیم.
- در حین اجرای تابع DFS-Visit روی راس v، رنگ این راس خاکستری خواهد بود.
- پس از پایان اجرای تابع DFS-Visit روی راس v، رنگ این راس مشکی می‌شود.

حال با حالت‌بندی روی رنگ راس w می‌توان نوع یال را مشخص کرد:

- اگر w سفید باشد یال vw درختی است. زیرا سفید بودن یک رأس نشان می‌دهد این رأس تا به حال نشان‌دار نشده‌است.

ـ اگر w خاکستری باشد یال vw برگشتی است. زیرا با توجه به ادعا۱ v از w قابل دسترسی است پس w از اجداد v است.
ـ اگر w مشکی باشد یال vw تقاطعی یا روبه‌جلو است. در این حالت می‌توان از خاصیت پرانتزی زمان های شروع و پایان برای تشخیص نوع یال استفاده کرد. چون رأس w مشکی است یعنی بررسی آن به پایان رسیده‌است. اما بررسی رأس v همچنان ادامه دارد. پس $w.f < v.f$.
حال دو حالت ممکن است رخ دهد.
یا رأس v پس از پایان بررسی w بررسی شده‌است ($v.s > w.f$) که در این صورت از ادعا ۱ نتیجه می‌شود v از نوادگان w نیست. پس این یال تقاطعی است.
یا رأس v هنگام بررسی w بررسی شده‌است ($v.s < w.f$) که در این صورت w از نوادگان v و یال vw روبه‌جلو است.

۴.۲ چند کاربرد

تشخیص وجود دور:

ورودی: گراف G. خروجی: آیا گراف G دور دارد؟

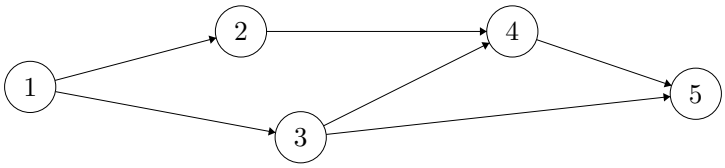
ادعا ۳. *گراف G دور دارد، اگر و فقط اگر $DFS(G)$ حداقل یک یال برگشتی داشته‌باشد.*

اثبات. ابتدا فرض کنید گراف G یال برگشتی داشته باشد. این یال را $e_0 = (u, v)$ بنامید. با توجه به تعریف یال برگشتی، u از نوادگان v است. پس مسیری مانند $W = ve_1v_1...e_ku$ از v به u موجود است. از اجتماع یال برگشتی و مسیر W یک دور به وجود می‌آید.

این بار فرض کنید گراف G دوری مانند $C = v_0e_1v_1...e_kv_ke_0v_0$ دارد. بدون کاستن از کلیت مسئله می‌توان فرض کرد v_0 اولین راس دور است که تابع DFS-Visit برای آن فراخوانی می‌شود. با توجه به *ادعا ۱* چون بین v_0 و v_k مسیر $C = v_0e_1v_1...e_kv_k$ موجود است و v_k هنوز نشان‌دار نشده‌است پس v_k از نوادگان v_0 خواهد بود پس با توجه به تعریف، یال v_k, v_0 برگشتی خواهدبود. □

مرتب‌سازی توپولوژیک[11]:

ورودی: *گراف جهت‌دار و بدون دور*[12] G. خروجی: ترتیب خطی از راس‌های گراف G که جهت همه یال‌ها از چپ به راست باشد.



تعبیری از مسئله فوق در ادامه آمده‌است. فرض کنید رأس‌های گراف تعدادی فعالیت هستند(مراحل مختلف آماده‌شدن برای رفتن به دانشگاه) که باید با رعایت روابط پیشنیازی انجام شوند. یال‌ها مشخص‌کننده روابط پیشنیازی هستند(جوراب پوشیدن پیشنیاز کفش پوشیدن است). می‌خواهیم ترتیب انجام کارها را به گونه‌ای مشخص کنیم که روابط پیشنیازی رعایت شوند.

لم ۱. *مسئله مرتب‌سازی توپولوژیک برای گراف‌های جهت‌دار بدون دور جواب دارد.*

[11] **Topological Sorting**
[12] Directed Acyclic Graph

اثبات. هر گراف جهت‌دار بدون دور راس چاه[۱۳] دارد (راس چاه راسی است که فقط یال ورودی داشته‌باشد). فرض کنید هیچ رأسی چاه نباشد ،یعنی همه رئوس یال خروجی دارند. در این صورت می‌توان از یک راس شروع کرد و هر بار با یکی از یال‌های خروجی‌اش به یکی از راس‌های مجاور رفت. چون تعداد رأس‌های گراف محدود است بالاخره وارد رأس تکراری می‌شویم. پس این گراف گشت بسته و در نتیجه دور دارد که تناقض است.

حال با استقرا روی تعداد رئوس حکم را ثابت می‌کنیم. اگر $|V(G)| = 1$ باشد به وضوح حکم برقرار است. فرض می‌کنیم برای $|V(G)| = n - 1$ حکم برقرار باشد. گراف جهت‌دار بدون دور G با n راس را در نظر بگیرید. این گراف حداقل یک راس چاه مانند v دارد. این راس را از گراف حذف کنید. گراف باقی‌مانده به وضوح بی‌دور است پس طبق فرض استقرا مسئله مرتب‌سازی توپولوژیک برای $G\backslash\{v\}$ جواب دارد. یعنی ترتیب خطی روی رئوس این گراف وجود دارد که جهت همه یال‌ها از چپ به راست است. حال راس v را به انتهای این ترتیب خطی اضافه کنید. چون v رأس چاه است پس همه یال‌های v یال ورودی هستند یعنی جهتشان از چپ به راست است. و حکم اثبات می‌شود. □

اثبات فوق الگوریتمی برای حل مسئله مرتب‌سازی توپولوژیک ارائه می‌دهد.

علاوه بر الگوریتم فوق می‌توان از الگوریتم جست‌وجوی اول عمق برای حل این مسئله استفاده‌کرد:
ـ الگوریتم جست‌وجوی اول عمق را روی گراف G اجرا کن.
ـ رأس‌ها را به ترتیب عکس زمان خاتمه در خروجی چاپ کن (واضح است که می‌توان این مرحله را در پیاده سازی تابع DFS وارد کرد).

اثبات. با برهان خلف و حالت‌بندی روی راس انتهای یال برگشتی درستی الگوریتم فوق ثابت می‌شود. □

۳ تعمیم

همانطور که دیدیم با تغییر صف به پشته در الگوریتم جست‌وجوی اول سطح توانستیم به الگوریتم کاربردی دیگری دست‌یابیم. اما به این دو داده‌ساختار محدود نیستیم. می‌توان الگوریتم فوق را برای سایر داده‌ساختارها تعمیم داد.

```
WFS(s)
 1  for each v ∈ V
 2      v.mark = false
 3  B = new Bag
 4  B.insert((∅, s))
 5  while B is not empty
 6      (v, w) = B.extract
 7      if w.mark == false
 8          w.mark = true
 9          w.parent = v
10          for each (w, t) ∈ E
11              B.insert(w, t)
```

[13] sink

این پیاده‌سازی با پیاده‌سازی‌های قبلی چند تفاوت دارد:
اولاً یال‌ها به جای رئوس وارد داده‌ساختار می‌شوند.
ثانیاً همه همسایه‌های w وارد داده‌ساختار می‌شوند و بررسی نشان‌دار بودن یا نبودن آن‌ها به زمان خروجشان موکول شده‌است.

ویژگی اول در پیاده‌سازی الگوریتم روی گراف‌های وزن‌دار به کارمی‌آید. برای مثال اگر به جای B از هرم کمینه استفاده‌کنیم به الگوریتم ارائه‌شده توسط یارنیک_پریم[۱۴] برای مسئله درخت فراگیر کمینه[۱۵] می‌رسیم. در این حالت الگوریتم را جست‌وجوی اول بهترین[۱۶] می‌نامیم.

همچنین اگر در این الگوریتم به هر یال (یعنی هر رأس در هرم) یک کلید[۱۷] نسبت دهیم، با توجه به این که این کلید را چگونه محاسبه کرده‌باشیم به الگوریتم دایجسترا[۱۸] یا الگوریتم محاسبه جریان در شبکه می‌رسیم.

مراجع

[1] Cormen, Thomas H., et al. *Introduction to Algorithms*. 3rd ed., MIT Press, 2009, pp. 594-610.

[2] Erickson, Jeff. *Algorithms*. 1st ed., 2019, pp. 199-204.

[14] Jarnik-Prim
[15] Minimum Spaning Tree
[16] Best-first Search
[17] key
[18] dijkstra

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۵: الگوریتم‌های حریصانه

نگارنده: محمد حسین مروجی

در این جلسه به معرفی الگوریتم های حریصانه [1] پرداخته می‌شود که بیشتر سعی بر این است که با استفاده حل مسئله موضوع توضیح داده شود. در میانه این جلسه مفهومی به نام ماترویید معرفی می‌شود و قضیه‌ای جالب در مورد آن گفته می‌شود.

۱ الگوریتم‌های حریصانه

این الگوریتم ها، الگوریتم های ساده ولی پرکابردی هستند که سعی می کنیم با ارائه چند مثال آن‌ها را توضیح دهیم و با کاربردهایشان آشنا شویم.

۱.۱ مسئله بازه‌ها

سوال: تعدادی بازه به ما داده شده است. الگوریتمی ارائه دهید که بیشترین تعداد بازه را انتخاب کند به طوری که هیچ دو تا بازه با هم اشتراک نداشته باشند.

برای حل این سوال می توانیم چند رویکرد داشته باشیم. رویکرد اول این است که بازه ها را برحسب زمان شروعشان به ترتیب صعودی مرتب کنیم و سپس از بازه اول شروع کنیم و در هر مرحله بازه ای را که اشتراکی با بازه های قبلی انتخاب شده ندارد انتخاب کنیم. رویکرد دوم می تواند این باشد که بازه هارا بر حسب زمان پایانشان به ترتیب صعودی مرتب کنیم و همان الگوریتم رویکرد اول را تکرار کنیم. رویکرد سوم می تواند این باشد که بازه ها را برحسب طولشان و به ترتیب صعودی مرتب کنیم و الگوریتم رویکرد اول را روی آن اجرا کنیم.

اما رویکرد مناسب برای این سوال همان رویکرد دوم است این موضوع را می توانیم با استفاده از آزمون و خطا روی یک یا چند مثال ببینیم که این رویکرد درست کار می کند. به شبه کد این الگوریتم توجه کنید.

```
SelectIntervals(intervals)
1  sort intervals by finishing times so that f_1 ≤ f_2 ≤ ... ≤ f_n
2  S ← ∅
3  for j = 1 to n
4      if interval[j] is compatible with S
5          S ← S ∪ {interval[j]}
6  return S
```

[1] Greedy algorithms

در خط ۴ شبه کد می توانستیم ابتدای بازه j ام را با انتهای بازه ای که آخرین بار انتخاب شده مقایسه کنیم.

اثبات بهینه بودن الگوریتم: فرض کنید در الگوریتم بالا بازه های $i_1, i_2, ..., i_k$ انتخاب شده باشند و در جواب بهینه بازه های $j_1, j_2, ..., j_m$ انتخاب شده باشند.با این فرض که $i_1 = j_1, ..., i_r = j_r$ که در آن r بزرگترین اندیس با این خاصیت است. حال اگر چند مجموعه جواب بهینه برای بازه ها داشتیم آن مجموعه‌ای را در نظر می گیریم که بیشترین r ممکن را داشته باشد.ترتیبی که بازه ها در بالا در نظر گرفته شده‌اند به صورت صعودی از نظر انتهای بازه است. حال به این نکته دقت می کنیم که می‌توانیم بازه j_{r+1} را از مجموعه جواب بهینه حذف کنیم و به جای آن بازه i_{r+1} را جایگزین کنیم به طوری که جواب همچنان بهینه بماند که این ما را به تناقض می رساند زیرا فرض کرده بودیم r بیشینه را انتخاب کرده ایم که به تناقض می رسیم.

حالا سوالی را مشابه سوال قبل مطرح می کنیم و حل آن را برای تمرین بیشتر به خواننده واگذار می‌کنیم.

سوال: تعدادی بازه که هر کدام وزن خاصی دارند داده شده است. الگوریتمی ارائه کنید که که تعدادی بازه انتخاب کند به طوری که هیچ دوتایی با هم اشتراک نداشته باشند و جمع مقدار وزن آن ها بیشترین ممکن باشد.

نکته ای که درباره این سوال وجود دارد این است که الگوریتم سوال قبل برای این سوال کار نمی کند زیرا می توانیم بازه‌ها را جوری وزن دهی کنیم که انتخابی که الگوریتم حریصانه در مثال قبل انجام می دهد خوب نباشد پس باید سعی کنیم جور دیگری از الگوریتم حریصانه استفاده کنیم.

۲.۱ شمای کلی از الگوریتم حریصانه

می‌توان الگوریتم حریصانه را در ۳ مرحله توصیف کرد:

۱. مسئله بهینه سازی رابه مسئله ای که یک انتخاب در آن انجام می‌دهیم تبدیل می‌کنیم و مسئله به یک زیرمسئله کوچکتر تبدیل می‌شود.

۲. ثابت می‌کنیم همیشه یک جواب بهینه برای مسئله وجود دارد که شامل انتخاب حریصانه است.

۳. نشان می دهیم خاصیت زیرمسئله بهینه[۲] برقرار است یعنی به این ترتیب که نشان می دهیم بعد از انتخاب حریصانه، زیرمسئله‌ای که باقی می‌ماند این خاصیت را دارد که اگر یک جواب بهینه برای زیرمسئله با انتخاب حریصانه تلفیق کنیم، یک جواب بهینه برای مسئله اصلی بدست می‌آید.

۱.۲.۱ کلیت مسئله هایی که روش حریصانه برای آن‌ها کار می‌کند

این مسئله ها معمولا به این صورت هستند که ورودی یک مجموعه S و یک مجموعه $A \subseteq 2^S$ و تابع وزن $w : S \to Q^+$ است و خروجی مورد نظر نیز یک مجموعه مانند $B \subseteq S$ که $B \in A$ به طوری که B بهینه باشد. منظور از بهینه بودن این است که مثلا اندازه مجموعه بزرگترین یا کوچکترین باشد و یا $w(B) = \sum_{x \in B} w(x)$ بیشترین یا کمترین باشد. به مثال زیر توجه کنید.

فرض کنید S مجموعه‌ای از بردارها باشد و A زیرمجموعه‌های مستقل خطی از S باشد. حال می خواهیم الگوریتمی ارائه دهیم تا مجموعه‌ای را از A انتخاب کند که بیشترین بردار مستقل خطی را داشته باشد.

الگوریتم حریصانه برای این سوال اینگونه است که ابتدا مسئله را به مسئله انتخاب چند چیز تبدیل می کنیم اینگونه که می گوییم هر مرحله برداری را انتخاب می کنیم که با بردارهای انتخاب شده قبلی وابسته خطی نباشد. طبق خواصی که در جبرخطی برای بردار های

[2] Optimal substructure

مستقل خطی و فضای برداری وجود دارد می دانیم الگوریتم ما درست کار می‌کند چون می دانیم اگر یک مجوعه بردار مستقل خطی داشته باشیم می توانیم این مجموعه را گسترش دهیم به پایه ای برای فضای برداری تبدیل کنیم.

حالتی که در مثال قبل بررسی شد حالتی بود که تابع وزن تابع ثابت یک بود حال فرض کنید اینگونه نباشد و حال این حالت را بررسی می کنیم. سوال این است که آیا الگوریتم زیر برای این مسئله کار می‌کند؟ الگوریتم این است که s_i ها را بر حسب وزنشان به ترتیب نزولی مرتب می کنیم یعنی بدین صورت:

$$S = \{s_1, s_2, ..., s_n\}, w(s_1) \geq w(s_2) \geq ... \geq w(s_n)$$

حال از s_1 شروع می کنیم و آن را انتخاب می کنیم.حال در مرحله i ام بردار s_i را انتخاب می‌کنیم اگر و تنها اگر s_i با بردار های قبلی انتخاب شده تشکیل یک مجموعه مستقل خطی را بدهند.

در ادامه بحث با مفهوم ماتروید آشنا می‌شویم. فرض کنید S مجموعه ای متناهی باشد و $I \subseteq 2^S$.

(S, I) را یک ماتروید می‌گوییم اگر سه خاصیت زیر برقرار باشد:

۱. $\emptyset \in I$

۲. اگر $B \subseteq A, A \in I$ آنگاه $B \subseteq I$

۳. اگر $|B| > |A|, A, B \in I$ آنگاه وجود داشته باشد $x \in B \backslash A$ به طوری که $(A \cup \{x\})A + x \in I$

نکته جالبی در مورد ماتروید ها وجود دارد که تحت قضیه زیر عنوان می کنیم.

قضیه) اگر $w : S \to Q^+$ یک تابع وزن دلخواه باشد آنگاه الگوریتم حریصانه با شبه کد زیر، زیرمجموعه مستقل از S با بیشترین وزن را پیدا می کند.

نکته جالبی که درباره این قضیه وجود دارد این است که برعکس قضیه بالا نیز درست است یعنی اگر الگوریتم حریصانه برای (S, I) و قضیه بالا کارا باشد و I شرط اول و دوم ماتروید بودن را داشته باشد در این صورت می‌توانیم نتیجه بگیریم شرط سوم ماتروید بودن را نیز خواهد داشت.

```
SELECTOPT(S)
1  suppose w(s_1) ≥ w(s_2) ≥ ... ≥ w(s_n)
2  A ← ∅
3  for i = 1 to n
4        if A + s_i ∈ I
5              A = A + s_i
6  return S
```

برهان: فرض کنید خروجی الگوریتم حریصانه $A = \{a_1, ..., a_m\}$ باشد و فرض کنید k اولین اندیس باشد که الگوریتم اشتباه کرده است یعنی جواب بهینه شامل $\{a_1, ..., a_{k-1}\}$ وجود دارد ولی جواب بهینه ای که شامل $\{a_1, ..., a_k\}$ باشد وجود ندارد.

فرض کنید $B = \{a_1, ..., a_{k-1}, b_1, ..., b_l\}$ یک جواب بهینه باشد و $C = \{a_1, ..., a_k\}$ حال الگوریتم مقابل را اجرا می‌کنیم

1 **while** $|B| > |C|$
2 choose $x \in B \backslash C$ such that $C + x \in I$
3 $C = C + x$

این حلقه پس از $l - 1$ مرحله به اتمام می رسد. در نهایت چون مجموعه نهایی یکی از b_i ها را ندارد پس به شکل زیر خواهد بود

$$C_{\text{final}} = \{a_1, ..., a_k, b_1, ..., b_{i-1}, b_{i+1}, ..., b_l\}$$

حال یک ادعا را سعی می‌کنیم بیان و اثبات کنیم.

ادعا: $w(a_k) \geq w(b_i)$.

برهان: فرض کنید $w(a_k) < w(b_i)$ در این صورت b_i توسط الگوریتم حریصانه زودتر از a_k بررسی شده است ولی انتخاب نشده است، یعنی مجموعه $\{a_1, ..., a_{k-1}, b_i\}$ مستقل نیست پس B نیز مستقل نیست که با فرض اولیه در تناقض است. پس ادعا ثابت می‌شود.

پس C_{final} خود یک جواب بهینه است که شامل a_k است که تناقض است پس در کل حکم قضیه اثبات می‌شود.

۲.۲.۱ زیرجنگل های یک گراف

در این قسمت می‌خواهیم حکم جالبی در مورد گراف ها را بررسی کنیم.

سوال: ثابت کنید زیرجنگل‌های یک گراف تشکیل یک ماتروید می‌دهند یا به بیانی دیگر اگر $G = (V, E), I = \{f \subseteq E | \text{f haven't cycle}\}$ ثابت کنید (E, I) یک ماتروید است.

برهان: خاصیت یک و دو از خواص ماتروید برای (E, I) واضح است. حال سعی می کنیم خاصیت سوم را اثبات کنیم یعنی اگر F, F' زیرجنگل‌هایی از G باشند و $|F'| > |F|$ آنگاه یال مانند $x \in F' \backslash F$ وجود دارد که $F + x$ همچنان یک جنگل باشد به طور معادل می‌توان گفت یال $x \in F'$ وجود دارد که دو سرش در دو مولفه همبندی متفاوت از F است. حال فرض کنید چنین یالی وجود ندارد حال هر مولفه همبندی مانند C از F را در نظر بگیریم یک درخت است و دارای $|C| - 1$ یال است. طبق فرض خلف نتیجه می‌شود مولفه های F' در مولفه‌های F جای می‌گیرند. پس مجموعه راس های C در F' نیز حداکثر دارای $|C| - 1$ یال است. پس در کل نتیجه می‌شود $|F'| \leq |F|$ که تناقض است پس فرض خلف باطل و حکم برقرار است.

از قضیه بالا نتیجه می‌گیریم مسئله بزرگترین جنگل فراگیر[3] را می‌توانیم با الگوریتم حریصانه حل کنیم و به عنوان تمرین بیشتر فکر کنید که چرا و چگونه می توانیم با الگوریتم حریصانه مسئله کوچکترین درخت فراگیر[4] را حل کنیم.

۳.۲.۱ زمان بندی کارهای واحد

برای تمرین بیشتر مسئله جالبی را مطرح می‌کنیم و سعی کنید که برای تسلط بیشتر به الگوریتم حریصانه آن را حل کنید.

سوال: تعداد n کار را باید انجام بدهیم که زمانی که هرکار طول می کشد یک است. کار i ام باید تا زمان مشخصی مانند $1 \leq d_i \leq n$ تمام شود که این زمان ها در ورودی داده شده است. اگر کار i ام تا زمان موعدش تمام نشود باید جریمه w_i را پرداخت کنیم که این اعداد نیز در ورودی داده شده است. حال الگوریتمی ارائه دهید یک زمان بندی مناسب از کارها بدهد به طوری کمترین میزان ممکن جریمه را پرداخت کنیم.

[3] Maximum spanning forest
[4] Minimum spanning tree

راهنمایی: مجموعه S را مجوعه تمام کارها بگیرید و مجموعه I را مجموعه همه‌ی زیرمجموعه‌هایی از S بگیرید به طوری که کارهای آن زیرمجموعه را بتوان جوری برنامه ریزی کرد که همگی کارها قبل از موعدشان انجام شوند.

مراجع

[1] Cormen, Thomas H., et al. *Introduction to Algorithms*. 3rd ed., MIT Press, 2009, pp. 414-450.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۶: ادامه الگوریتم حریصانه

نگارنده: نیما بهرنگ

در جلسه قبل کلیاتی از مسائلی که الگوریتم حریصانه برایشان کار می‌کند را مطالعه کردیم و مدلی برای این دسته مسائل ارائه کردیم. در این جلسه به ادامه الگوریتم حریصانه می‌پردازیم.

۱ زیردرخت فراگیر کمینه

مسئله کمینه زیردرخت فراگیر [1] با یک گراف وزن دار $G = (V, E)$ و تابع وزن آن $C : E \to Q_+$ تعریف می‌شود و در آن به دنبال یک زیردرخت فراگیر هستیم که جمع یالهایش کمینه باشد.
در ماتروئیدها دیدیم که مسئله زیرجنگل با بیشینه وزن وجود داشت که ارتباط زیادی با مسئله ما دارد.

یک رویکرد حریصانه برای حل این مسئله، *رویکرد افزایشی* است.

۱.۱ رویکرد افزایشی

در ابتدا گراف خالی $H = (V, \emptyset)$ را در نظر می‌گیریم و گام به گام یال مناسبی اضافه می‌کنیم. اولین شرط برای افزودن یال، جلوگیری از تشکیل دور در گراف است.

- **برش**[2]: منظور از برش، افراز (S, V-S) از V است که مجموعه رأس های ما را به دو دسته تقسیم می‌کند و هر رأس در دقیقا یکی از این دسته ها می‌باشد.
  یال های برش $\delta(S)$ شامل تمام یال هایی از G است که دقیقا یک سر آنها در S می‌باشد.
- **نکته**: هر زیرگراف همبند از G باید شامل حداقل یک یال از $\delta(S)$ باشد.

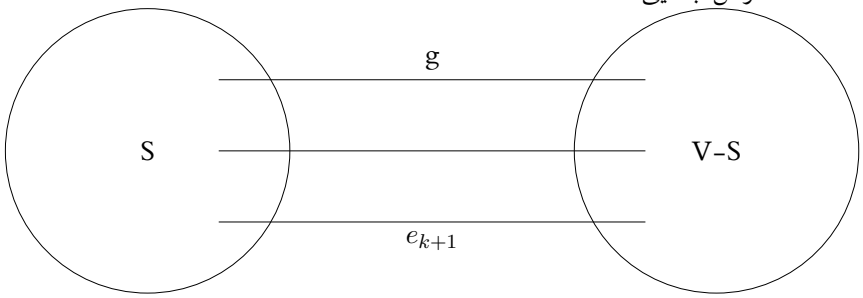
رویکرد افزایشی برای افزودن یال این است که یک برش (S, V-S) در نظر بگیریم که $E(H) \cap \delta(S) = \emptyset$ و کم وزن ترین یال از $\delta(S)$ را به H اضافه کنیم.

لم ۱. *اجرای رویکرد افزایشی به یک درخت فراگیر کمینه منجر می‌شود.*

اثبات. یال های بدست آمده توسط الگوریتم را با $\{e_1, e_2, ..., e_{n-1}\}$ نشان می‌دهیم.
درخت فراگیر کمینه ای را در نظر می‌گیریم که بیشترین تعداد یال‌های مشترک را با یال‌های الگوریتم دارد. بدون تغییر در کلیت مسئله، فرض

[1] MST: minimum spanning tree
[2] Cut

می‌کنیم یال‌های $\{e_1, ..., e_k\}$ مشترک باشند. پس هیچ درخت فراگیر کمینه‌ای با بیشتر از k یال مشترک نداریم.
یال های درخت کمینه مفروض T^* را با $\{e_1, ..., e_k, f_1, ...f_{n-1-k}\}$ نشان می‌دهیم.
حال یال e_{k+1} را به T^* اضافه می‌کنیم پس یک دور در درخت ساخته می‌شود. این یال، کم وزن ترین یال در یال‌های مربوط به یک برش (S, V-S) بوده‌است، پس با افزودن این یال و حذف یالی در دور ساخته شده که در مجموعه یال‌های این برش وجود دارد g ، همچنان یک درخت فراگیر داریم و مجموع وزن درخت حداکثر مساوی درخت اولیه می‌ماند و یا کمتر می‌شود که در هر دو صورت به تناقض می‌رسیم. زیرا اگر وزن کم شود، پس درخت مفروض، کمینه نبوده‌است و اگر وزن ثابت بماند، یک درخت فراگیر کمینه با k+1 یال مشترک یافته‌ایم که خلاف فرض ابتدایی است.



□

پس این روش همواره به یک درخت پوشای کمینه منجر می‌شود.
روش های متفاوتی برای پیدا کردن برش‌ها وجود دارد.

- **پرایم_جارنیک** [۳]

در ابتدا یک مجموعه‌ی خالی به نام H فرض می‌کنیم. یکی از رأس‌ها (v_1) را انتخاب می‌کنیم و به H اضافه می‌کنیم و H را به عنوان برش در نظر می‌گیریم $(H, V - H)$ حال کمینه یال این برش را انتخاب می‌کنیم و رأسی را که در انتهایی از یال که در H نیست (v_2) را به H اضافه می‌کنیم. مجدد H را به عنوان برش در نظر می‌گیریم و همین کار را تکرار می‌کنیم.

- **کروسکال** [۴]:

یال‌ها را بر اساس وزنشان از کوچک به بزرگ مرتب می‌کنیم و از کوچکترین یال شروع می‌کنیم و یکی یکی یال‌ها را در صورتی که با یال‌های انتخاب شده‌ی پیش از خود تشکیل دور ندهند، آن یال را به مجموعه اضافه می‌کنیم.

- **برووکا** [۵]:

در هر گام کمینه یال از هر زیردرخت به باقی زیردرخت‌ها را به مجموعه اضافه می‌کنیم.
در ابتدا تمام تک رأس‌ها هرکدام یک زیردرخت هستند، حال برای هر رأس کمترین یال را انتخاب می‌کنیم و اضافه می‌کنیم. حال تعدادی زیردرخت بزرگتر داریم. به ازای هرکدام، کمینه یال از آن زیردرخت به باقی زیردرخت‌ها را به مجموعه اضافه می‌کنیم تا درخت‌های بزرگتری ساخته‌شود و در انتها با تکرار این گام‌ها به یک درخت فراگیر کمینه برسیم.

الگوریتم برووکا به دلیل اینکه به صورت پردازش موازی قابل انجام است، از باقی الگوریتم‌ها در برخی مواقع گزینه بهتری‌است.
شرط متمایز بودن وزن یال‌ها برای ایجاد نشدن دور در الگوریتم لازم است و اگر تمام وزن‌ها متمایز باشد، دور ایجاد نمی‌شود.

- **تمرین**: اثبات کنید که اگر وزن تمام یال‌ها متمایز باشد، درخت فراگیر کمینه، یکتا است.

---
[۳] Prime-Jarnik
[۴] Kruskal
[۵] Boruvka

۲.۱ رویکرد کاهشی

در هر گام به دنبال حذف یک یال هستیم و یک رویکرد حذف یک سنگین ترین یال در یک دور است.

- **تمرین:** اثبات کنید که این رویکرد به یک درخت فراگیر کمینه منجر می‌شود.(راهنمایی: اولین جایی که یالی که حذف کردیم گراف باقیمانده را به گرافی تبدیل می‌کند که هیچ درخت فراگیر کمینه‌ای، زیرمجموعه یال‌های باقی مانده نباشد را در نظر بگیرید)

۳.۱ رویکرد جستجوی موضعی

یک درخت پوشا در نظر می‌گیریم و تا زمانی که یالی وجود دارد در داخل درخت و یک یال با وزن کمتر در خارج درخت که با افزودن یال خارجی و حذف یال داخلی، همچنان یک درخت خواهیم‌داشت، به تعویض یال‌ها ادامه می‌دهیم.

$$\exists e \in T, e' \in E - T : T - e + e' \; is \; tree, \; w(e) > w(e') \rightarrow T = T + e' - e$$

- **سوال:** آیا زمان اجرای الگوریتم چندجمله‌ای است؟
- **سوال:** چرا هنگام توقف، یک زیردرخت فراگیر کمینه بدست می‌آید؟

علت نام‌گذاری *جستجوی موضعی* این است که در هر گام شروع می‌کنیم جواب فعلی خود را کمی بهینه کنیم و فرض اساسی این است.

۴.۱ رویکرد ترکیبی/قاعده قرمز و آبی [۶]

در این رویکرد که در کتاب [۷] موجود است، با رویکرد کاهشی، دوری که هیچ یال با رنگ قرمز ندارد را انتخاب و یال بدون رنگ با بیشترین وزن را به رنگ قرمز در می‌آوریم. با رویکرد افزایشی، یک برش در نظر می‌گیریم که هیچ یال آبی ندارد و کمترین یال رنگ نشده از برش را به رنگ آبی در می‌آوریم.
این دو قاعده را به ترتیبی دلخواه یا با نظمی مشخص می‌توان انجام داد که در آخر یال های آبی نشان دهنده درخت فراگیر کمینه است.

- **نکته:** برای الگوریتم های پیدا کردن درخت فراگیر کمینه، زمان خطی بر حسب n برای الگوریتمی تصادفی وجود دارد و برای الگوریتم‌های قطعی، الگوریتمی بهینه وجود دارد که اوردر زمانی آن مشخص نیست.

[۶] Blue-rule Red-rule
[۷] algorithms Network and structures Data

۲ حافظه نهان/صفحه بندی [۸]

در دنیای کامپیوتر مسئله‌ای که وجود دارد این است که انواع مختلفی از حافظه وجود دارد که برخی از آنها سریع تر هستند ولی اندازه آنها کمتر است. برای مثال حافظه رم [۹] نسبت به هارد [۱۰] یا حافظه نهان [۱۱] نسبت به رم و کاربرد وجود حافظه سریعتر این است که مقدار محدودی اطلاعات روی آن نگهداری شود تا نیاز به استفاده از حافظه کندتر، کمتر شود.
در حافظه نهان ما می‌توانیم k مورد [۱۲] ذخیره کنیم و دنباله m تایی $d_1, d_2, ..., d_m$ که مواردی است که ما به مرور زمان نیاز به خواندن آن مورد داریم. در هربار که درخواست خواندن یک مورد انجام می‌شود، اگر در حافظه‌ی نهان باشد، از حافظه نهان خوانده می‌شود [۱۳] و اگر در حافظه نهان نباشد، مجبور می‌شویم یک مورد را از حافظه نهان پاک کنیم و مورد جدید را به داخل حافظه نهان اضافه کنیم [۱۴] .
هدف در این مسئله این است که تعداد مواردی که یک مورد در حافظه نهان وجود ندارد را کمینه کنیم. مسئله مطرح در اینجا این است که هنگام بیرون انداختن یک مورد، کدام مورد باید بیرون انداخته شود. روش های متفاوتی که به روش حریصانه هستند وجود دارند. -LIFO/ FIFO, LRU, LFU که این الگوریتم ها الگوریتم های برخط [۱۵] هستند و به اطلاعات گذشته تا کنون دسترسی دارند.اما الگوریتم‌های غیر برخط [۱۶] هم وجود دارد که فرض آنها دانستن دنباله مواردی که در آینده خوانده می‌شوند نیز هست و با فرض دانستن تمام این اطلاعات، تصمیم به حذف موارد از حافظه نهان می‌گیرد. یک روش این است که موردی که در آینده‌ی دورتری به آن نیاز داریم را از حافظه نهان حذف کنیم. اینکه چنین روشی خوب است، نیاز به اثبات دارد.

- **برنامه ریزی کاهش یافته**[۱۷]: اگر در هر گام تنها موردی که در آن گام مورد نیاز بوده و در حافظه نهان نبوده را به حافظه نهان اضافه کنیم.
  افزودن یک مورد حافظه نهان، عملیاتی زمان بر است و اثبات می‌شود که برنامه ریزی کاهش یافته بهترین روش برای افزودن موارد به حافظه نهان است. برای اینکار یک روش غیر کاهش یافته را به یک روش کاهش یافته تبدیل می‌کنیم که به تعداد الگوریتم اول، مورد از حافظه حذف می‌کند.

- **دورترین در آینده**[۱۸] همانطور که قبلا اشاره شد، یک روش برای حذف، حذف کردن موردی است که در آینده‌ی دورتری به آن نیاز خواهدشد. حال به اثبات بهینه بودن این روش می‌پردازیم.
  ابتدا به این می‌پردازیم که یک روش برنامه‌ریزی کاهش یافته برای هر روش دورترین در آینده وجود دارد که هزینه ای برابر یکدیگر در j گام اول دارند که با استقرا ثابت می‌کنیم به ازای هر j برقرار می‌ماند.
  پایه استقرا j=۰ است. برای j گام اول الگوریتم دورترین در آینده، یک الگوریتم کاهش یافته با تعداد برابر هزینه وجود دارد. در گام j+1 چند حالت ممکن است پیش بیاید.
  فرض کنید موردی که در گام j+۱ خوانده‌شده‌است، d باشد و S حافظه نهان آن الگوریتم برنامه ریزی کاهش یافته باشد که تا گام j با حافظه نهان الگوریتم دورترین در آینده (S_{FF}) یکسان است. حال ما یک حافظه نهان کاهش یافته S' از روی S می‌سازیم که در j+1 گام با الگوریتم دورترین در آینده یکسان باشد و هزینه ای معادل با حافظه نهان کاهش یافته اولیه داشته باشد.

---
[۸] Caching/Paging
[۹] RAM
[۱۰] HDD
[۱۱] Cache
[۱۲] item
[۱۳] hit Cache
[۱۴] miss Cache
[۱۵] online
[۱۶] offline
[۱۷] schedule reduced
[۱۸] future in farthest

۱. d در حال حاضر در حافظه نهان موجود باشد
پس همچنان دو حافظه نهان در گام j+۱ نیز یکسان خواهند بود: $S = S'$

۲. d در حافظه نهان نباشد و هر دو الگوریتم عضو مشابهی را از حافظه نهان خود حذف کنند
باز همچنان مشابه خواهند بود: $S = S'$

۳. d در حافظه نهان موجود نباشد و الگوریتم دورترین در آینده عضو e را حذف کند و الگوریتم برنامه‌ریزی کاهش یافته عضو f را حذف کند.
حال باید نشان دهیم که در آینده S' حداکثر به اندازه S هزینه بر است.
j' را اولین جایی فرض می‌کنیم که بعد از j+1 ، دو حافظه نهان S, S' متفاوت عمل می‌کنند و مورد g در گام j' قرار است خوانده‌شود.

(آ) g = e

چنین حالتی ممکن نیست زیرا طبق فرض دورترین، e دیرتر از f باید مورد نیاز باشد وگرنه الگوریتم دورترین می‌بایست e را حذف می‌کرد.

(ب) g = f

چون f در S نیست، فرض می‌کنیم که در این گام S e' را حذف کند تا f را به حافظه نهان اضافه کند.
اگر $e = e'$ باشد، پس S برابر با S' شده است و در S' هزینه کمتری انجام شده‌است.
اگر $e \neq e'$ باشد، در S' عضو e' را حذف می‌کنیم و e را اضافه می‌کنیم پس با هزینه ای برابر $S = S'$ شد.
و در ادامه S' همانند S رفتار کند.

(ج) $g \neq f, e$ و S عضو e را حذف می‌کند.
S' عضو f را حذف کند.
حال دوباره با هزینه ای برابر، $S = S'$ شده است و در ادامه S و S' یکسان عمل کنند.

پس دیدیم که می‌توان یک الگوریتم کاهش یافته ساخت که در j+1 گام مشابه الگوریتم دورترین در آینده باشد پس به استقرا به ازای هر تعداد گام نیز شرط صحیح است.

آنچه تا کنون دیدیم مربوط به الگوریتم های غیر برخط بود اما در مورد الگوریتم های بر خط نیز تقریب هایی وجود دارد. مثلا

LRU is k-competitive: for any sequence of requests σ, $LRU(\sigma) \leq k.FF(\sigma) + k$
Randomized marking is $O(log(k)) - competitive$

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۷: برنامه‌ریزی پویا[1] نگارنده: آرمیتا کاظمی‌نجف‌آبادی

طی دو جلسه پیش با الگوریتم حریصانه، نحوه به‌کارگیری آن برای حل مسائل و مدل کلی اثبات درستی‌اش آشنا شدیم.
این جلسه قصد داریم با برنامه‌ریزی پویا که تکنیک مهمی‌در طراحی الگوریتم است آشنا شویم.

۱ طرح یک مسأله

ورودی: عدد n
خروجی: به چند طریق می‌توان عدد n را به صورت مجموع تعدادی ۱ و ۳ و ۴ نوشت طوری که ترتیب استفاده نیز مهم باشد.
(مثلا هر دو مورد ۴+۳ و ۳+۴ در شمارش جواب برای $n = ۷$ حساب شوند.)

به راحتی می‌توانیم با یک الگوریتم بازگشتی جوابی برای مسأله بیابیم.

```
SUM(n)
1  if n ≤ −1
2        return 0
3  elseif n = 0
4        return 1
5  else
6        return SUM(n − 4) + SUM(n − 3) + SUM(n − 1)
```

اثبات درستی: برای همه مجموعه‌های مرتب از ۱ و ۳ و ۴ ها که مجموعی برابر n دارند می‌توان سه حالت مجزا متصور شد؛ اینکه آخرین عددِ ظاهرشده در مجموع، ۱ یا ۳ یا ۴ باشد. بنابراین پاسخ مسأله، مجموع تعداد اعضای هر یک از این سه حالتِ مجزا است.
اگر پاسخ این مسأله را به فرم $\text{SUM}(n)$ نمایش دهیم، بسته به اینکه آخرین عدد ۱ یا ۳ یا ۴ باشد، به ترتیب $\text{SUM}(n-1)$ ، $\text{SUM}(n-3)$ و $\text{SUM}(n-4)$ حالت مجزا خواهیم داشت.
یعنی: $\text{SUM}(n-4) + \text{SUM}(n-3) + \text{SUM}(n-1) = \text{SUM}(n)$

پایه: اگر n کمتر از صفر باشد، نمی‌توان عدد را به فرم مطلوب نوشت. اگر n برابر صفر باشد به یک طریق می‌توان این کار را کرد! (اینکه هیچ یک از ۱و ۳و ۴ را استفاده نکنیم.)
هر حالتِ دیگر نهایتا به حالات پایه‌ی مذکور ختم شده و به درستی جواب آن محاسبه می‌شود. (منظور از پایه، حالاتی است که غیربازگشتی محاسبه می‌شوند.)

[1] Dynamic Programming

تحلیل زمانی: با در نظر گرفتن درخت بازگشتیِ مسأله، به راحتی می‌توان پی برد که الگوریتم، زمان اجرای زیادی (از مرتبه[۲] ی نمایی) دارد. چون تا طبقه‌ی $\lfloor \frac{n}{4} \rfloor$ تعداد رئوس هر طبقه نسبت به طبقه قبل سه برابر می‌شود. (با توجه به درخت بازگشتی، شاخه‌ای که در طبقات کمتری حضور دارد، شاخه‌ای است که پدر هر کس ۴ واحد از آن بیشتر است و با شروع از n و هر بار ۴ واحد کم شدن، طول آن $\lfloor \frac{n}{4} \rfloor$ خواهد بود.) بنا به گزاره اخیر و اینکه رئوس آن طبقات سه فرزند دارند، برگ‌های درخت از $\Omega(3^{\lfloor \frac{n}{4} \rfloor})$ است. که نشان‌دهنده زمان اجرای نمایی الگوریتم است.
توجه کنیم که در شبه کدِ الگوریتم ارائه شده، خط آخر است که باعث می‌شود تابع به شکل بازگشتی صدا زده شود و مقصر اصلیِ بالارفتنِ هزینه زمانی است.

بهینه‌سازی: با کمی‌توجه می‌توان این روند را بهینه کرد. نکته اینجاست که ممکن است $\textsc{Sum}(i)$ بارها توسط این تابع به طور بازگشتی محاسبه شود؛ در صورتی که اگر بار اول آن را در جایی ذخیره کنیم، لازم نخواهد بود هربار مقدارش را بازگشتی محاسبه کنیم.
ایده‌ی طرح‌شده را می‌توان به کمک یک آرایه به طول n که در ابتدا مقدار همه خانه‌هایش برابر -1 است، پیاده کرد. به این ترتیب که هرگاه به‌ازای یک i به مقدارِ $\textsc{Sum}(i)$ نیاز داشتیم، ابتدا به خانه i در آرایه نگاه کنیم. اگر مقدار آن منفی یک بود، می‌بایست $\textsc{Sum}(i)$ را محاسبه کرده و در آن خانه قرار دهیم. در غیر این صورت، $\textsc{Sum}(i)$ قبلا محاسبه شده و ما در $O(1)$ به آن دسترسی داریم. فرض کنید A آرایه‌ی مذکور باشد که همه خانه‌هایش را منفی یک کرده‌ایم. تابع زیر را برای محاسبه جواب صدا می‌زنیم.

```
SUM(n, A)
1  if n ≤ −1
2      return 0
3  elseif n = 0
4      return 1
5  elseif A[n] ≠ −1
6      return A[n]
7  else
8      A[n] = SUM(n − 4, A) + SUM(n − 3, A) + SUM(n − 1, A)
9  return A[n]
```

نهایتاً زمان اجرای کل الگوریتم، به تعداد بارهایی که سراغ آرایه می‌رویم بستگی دارد. مقدار هر خانه از آرایه حداکثر یک بار تغییر داده می‌شود و زمانی که مقدار خانه n ام مشخص شود، الگوریتم به پایان رسیده است.
اگر بخواهیم دقیق‌تر بگوییم، باید به این هم توجه کنیم که هر خانه مثل i حداکثر چندبار به طور بازگشتی در $O(1)$ مقدارش دیده می‌شود. پاسخ این است که خانه i ، حداکثر سه بار توسط الگوریتم بازگشتی که خانه‌ی $i+1$ یا خانه $i+3$ یا $i+4$ را منفی یک یافته‌است، دیده می‌شود. بعد از اینکه در هر یک از این سه خانه عددی جز منفی یک قرار گرفت، از آن پس اجرای تابعِ بازگشتی قبل از رسیدن به i و دیدن آن متوقف خواهد شد.
پس زمان اجرای الگوریتم، خطی یا همان $O(3n) = O(n)$ است.

با همین ایده‌ی کمک‌گرفتن از آرایه می‌توان مسأله را به کمک یک حلقه ساده ی for، حل کرد.
به این صورت که به ترتیب خانه‌های آرایه را از 0 تا $n-1$ به کمک خانه‌هایی که قبلا پر کرده‌ایم مقداردهی کنیم. این کار امکان‌پذیر است چون مقدارِ $\textsc{Sum}(i)$ به مقادیر $\textsc{Sum}(i-1)$ ، $\textsc{Sum}(i-3)$ و $\textsc{Sum}(i-4)$ بستگی دارد که هر سه قبل از $\textsc{Sum}(i)$ مقداردهی شده‌اند.
در این پیاده‌سازی خطی‌بودن زمان اجرا روشن‌تر است.

[۲] Order

```
SUM(n)
1  A[0] = 1
2  A[1]= 1
3  A[2]= 1
4  A[3]= 2
5  for i = 4 to n
6        A[i] = A[i − 1] + A[i − 3] + A[i − 4]
7  return A[i]
```

۲ روش کلی حل مسائل به کمک برنامه‌ریزی پویا

به کمک مثال قبل، با ایده‌ی کلی برنامه‌ریزی پویا آشنا شده و می‌توانیم یک صورت‌بندی کلی از این روش ارائه دهیم.

ـ مسأله را به زیرمسأله‌هایی تبدیل می‌کنیم که تعداد این زیرمسأله ها اکثرا از مرتبه ی چندجمله‌ای است. (زیرمساله‌ها معمولا شبیه مساله اصلی‌اند.)

ـ جواب بهینه‌ی یک مسأله را می‌توان برحسب جواب بهینه‌ی زیرمسأله‌هایش بدست آورد.

برای پیاده‌سازی این ایده، دو روش کلی داریم:

ـ **روش پیمایشی**[۳]: خانه‌های آرایه‌ای که قرار است جواب زیرمسأله‌ها را در خود جای دهد؛ به ترتیبی خاص پر کنیم. هر خانه جواب یک زیرمسأله را نگه می‌دارد و آرایه لزوما یک بعدی نیست. معمولا بُعد آن به تعداد مولفه‌های ورودی تابع بازگشتی که جواب را محاسبه می‌کند بستگی دارد. (مثال های بعدی، این مفهوم را روشن تر می‌کنند.)

ـ **روش بالا به پایین**[۴]: همین کار (به دست آوردن جواب زیرمسأله ها و ذخیره جواب در یک آرایه) را بازگشتی انجام دهیم و دقت کنیم محاسبه اضافه‌ای انجام نشود. حالات پایه‌ای را به درستی مشخص کنیم.

۳ مسأله کوله پشتی[۵]

ورودی: عدد s که اندازه یا ظرفیت کوله‌پشتی را نشان می‌دهد و مجموعه‌ای از n عنصر که عنصر i اندازه‌ی s_i و ارزش v_i دارد.
خروجی: زیرمجموعه‌ای از عناصر که جمع اندازه‌هایشان از s بیشتر نبوده و جمع ارزش‌هایشان بیشینه باشد.
این مسأله دو صورت متفاوت دارد:

ـ **صورت کسری:** ممکن است کسری از یک عنصر در کوله‌پشتی باشد.
می‌توان صورت بهتری برای این حالت ارائه داد. فرض کنید s زمانی معین باشد که می‌بایست طی آن n سوال را حل کنیم که حل کردن سوال i زمان s_i می‌گیرد و نمره‌اش v_i است. به علاوه اگر کسری از یک سوال را حل کنیم به طور طبیعی کسری از نمره را خواهیم گرفت. در این حالت می‌توان مسأله را به کمک الگوریتم حریصانه حل کرد. می‌بایست عناصر را به ترتیب نزولی با توجه به نسبت ارزش به اندازه‌شان (چگالی) انتخاب کرد.

[3] iterative
[4] memoization
[5] knapsack

ـ **صورت صفر و یکی:** هر عنصر یا به طور کامل در کوله‌پشتی هست یا نیست.

برای این حالت مثالی بیاورید که الگوریتم حریصانه برایش کار نکند! (یا به طور ساده تر مثالی ارائه دهید که الگوریتم حریصانه مربوط به سوال قبل، به ازای آن جواب بهینه ندهد.)

سعی می‌کنیم زیرمسأله‌های مناسبی که منجر به حل این مساله می‌شوند پیدا کنیم:

یک تلاش ناموفق این است که بخواهیم زیرمساله‌ی $V[i]$ را حل کنیم که $V[i]$ بیشینه‌ی ارزش یک زیرمجموعه از عناصر 1، 2، ...، i با حداکثر اندازه s می‌باشد.

با کمی‌تغییر در این ایده می‌توانیم زیرمساله‌ی بهتری معرفی کنیم که رابطه‌ای بازگشتی برای مساله اصلی‌مان ایجاد کند:

زیرمسأله $V(i, B)$ را در نظر می‌گیریم که $V(i, B)$ بیشینه ارزش یک زیرمجموعه از عناصر 1، 2، ...، i با حداکثر اندازه‌ی B است.

بنابراین جواب مسأله اصلی‌مان $V(n, s)$ خواهد بود. با کمی‌دقت می‌توان رابطه بازگشتی مناسبی برای $V(i, B)$ بدست آورد. (با حالت‌بندی روی اینکه در جواب بهینه، عنصر i در کیف قرار می‌گیرد یا خیر.)

$$V(i,B) = \begin{cases} 0 & i = 0 \\ V(i-1,B) & s_i > B \\ \max\{v_i + V(i-1, B-s_i), V(i-1,B)\} & otherwise \end{cases}$$

با توجه به $V(i, B)$ که دو مولفه ورودی دارد، می‌بینیم آرایه‌ی موردنظر باید دوبعدی باشد. اندازه‌ی آن نیز $n \times s$ است. (فرض کنیم مقدار s صحیح است.) با توجه به رابطه بازگشتی بدست آمده اگر i صفر نباشد، جواب مسأله به زیرمسأله‌ای در سطر پایین‌تر آن بستگی دارد. بنابراین با پیمایش خانه‌های گراف، از پایین به بالا به صورت سطری (همه‌ی خانه‌های یک سطر طی‌شده سپس به سطر بالاتر می‌رویم.) می‌توانیم جواب مسأله اصلی را بدست آوریم.

می‌توان مسأله را از روش بالا به پایین نیز حل کرد که در این صورت نیاز نیست درگیر ترتیب پیمایش خانه‌ها شویم. تنها باید حالات پایه‌ای را درست تشخیص دهیم. در اینجا $i = 0$ حالت پایه‌ای ست.

۴ درخت وابستگی

تا به حال سعیمان بر این بوده که جواب مسأله اصلی را بر اساس جواب زیرمسأله‌هایی از آن بدست آوریم. بر این اساس گرافی جهت‌دار می‌سازیم. به ازای هر یک از زیرمسأله‌ها رأسی قرار داده و در صورتی که جواب مسأله‌ی هدف به زیرمسأله‌های خاصی بستگی داشته‌باشد، یالی جهت‌دار از رأس متناظر با هر یک از زیرمسأله‌ها به رأس متناظر با مسأله هدف ایجاد می‌کنیم. (مسأله هدف خود ممکن است یک زیرمسأله باشد)

حل‌پذیر بودن مسأله به این بستگی خواهد داشت که گراف بدست‌آمده دور نداشته‌باشد.

به عبارت دیگر کافی‌ست گرافی جهت‌دار و بدون دور داشته‌باشیم که بتوانیم مرتب‌سازی توپولوژیک روی آن انجام‌دهیم. و نهایتا به روش پیمایشی روی رئوسِ مرتب‌شده، جواب هر زیرمسأله را تا رسیدن به مسأله نهایی محاسبه کنیم. چرا که اگر پیمایش‌مان به ترتیب توپولوژیک انجام شود، مطمئن هستیم جواب هر زیرمسأله‌ای که برای حل یک مسأله لازم است از پیش محاسبه شده است.

۵ مسأله هم‌ترازی[۶]

ورودی: رشته‌ی s به طول m و رشته ی t به طول n ؛ عدد d (هزینه‌ی فاصله) و اعداد a_{xy} به طوری که x کاراکتری از رشته s و y کارکتری از رشته t باشد. (هزینه تطابق x و y)

[6] alignment or edit distance

خروجی: دو رشته چقدر تطابق دارند یا به بیانی دیگر، کمینه هزینه لازم برای هم‌تراز کردن دو رشته چقدر است.
کاربرد ها: بیوانفورماتیک (تعیین شباهت پروتئین ها)، ژنتیک، تصحیح املایی[۷]، ترجمه

ابتدا می‌بایست تطابق را تعریف کنیم. تطابق، یک مجموعه از زوج‌های مرتب (x_i, y_j) است به طوری که x_i کاراکتر i ام از رشته s و y_j کاراکتر j ام از رشته t بوده و هر کاراکتر حداکثر در یک زوج‌مرتب آمده است. به علاوه هیچ دو زوجی تقاطع ندارند. (دو زوجِ (x_i, y_j) و (x_l, y_k) را متقاطع گوییم اگر $i < l, j \geq k$ یا $i \leq l, j > k$ برقرار باشد.) هر زوج مرتبِ (x_i, y_j) که در تطابق ظاهر می‌شود، هزینه‌ی a_{xy} و هر عنصری که در زوج مرتبی نیامده، هزینه‌ی d به ما (هزینه کل) تحمیل می‌کند.

می‌توان به شکلی دیگر نیز مسأله را توصیف کرد. طوری بین حروف یا ابتدا و انتهای هر رشته، کاراکتر فاصله (با هزینه d) اضافه کنیم (یا نکنیم) که اولا طول دو رشته یکی شود و ثانیا اگر دو رشته را حرف به حرف زیر هم بنویسیم، مجموع هزینه‌ی تفاوت دو به دوی حرف‌هایی که روبه‌روی هم قرار گرفته‌اند، کمینه شود.
زیرمسأله‌ای که در اینجا به ما کمک می‌کند $OPT(i, j)$ می‌نامیم که کمترین هزینه لازم برای تطابق دادن i کاراکتر ابتدایی رشته اول با j کاراکتر ابتدایی رشته‌ی دوم است.
نهایتا رابطه بازگشتی زیر بدست می‌آید :

$$OPT(i,j) = \begin{cases} j \times d & i = 0 \\ i \times d & j = 0 \\ \min\{a_{x_i y_j} + OPT(i-1, j-1), d + OPT(i-1, j), d + OPT(i, j-1)\} & ij \neq 0 \end{cases}$$

درستی رابطه را بررسی می‌کنیم. اگر $i = 0$ باشد، باید در واقع یک زیررشته به طول صفر را با زیررشته‌ای به طول j تطابق دهیم. این کار به یک طریق ممکن است (پس همین یک روش کم‌هزینه‌ترین روش است.) از آنجا که مجموعه زوج‌مرتب‌هایی که در تطابق می‌آیند تهی ست؛ هزینه کل برابر $j \times d$ خواهد بود چون هیچ یک از j عنصر در زوج‌مرتبی ظاهر نشده‌اند.
مشابها اگر $j = 0$ باشد می‌توان گفت کمینه‌ی هزینه کل، $i \times d$ است.
حال فرض کنید هیچ یک از i و j صفر نیستند. در حالت اول x_i و y_j ، هردویشان در زوج‌مرتب‌های تطابق بهینه حضور دارند. از آنجا که هریک آخرین کاراکتر در رشته‌ی خود هستند، اگر در تطابق‌های متفاوتی ظاهرشده باشند تقاطع پیش می‌آید. بنابراین در این حالت می‌بایست خودشان یکی از زوج‌های تطابق باشند. هزینه‌ی بهینه هم در این حالت برابر $a_{x_i y_j} + OPT(i-1, j-1)$ خواهد بود.
در حالت دوم مثلا x_i در تطابق بهینه حاضر نیست پس می‌بایست هزینه‌ی d را پرداخت کند و بقیه هزینه‌ها هم در بهینه‌ترین حالت برابر $OPT(i-1, j)$ خواهدبود که هزینه نهایی‌اش $d + OPT(i-1, j)$ است.
مشابها حالت سوم این است که y_j در تطابق حاضر نباشد. در این حالت مانند حالت قبل، کمترین هزینه نهایی $d + OPT(i, j-1)$ است. بنابراین جواب $OPT(i, j)$ ، کمینه‌ی این سه حالت خواهد بود. (هر تطابق قابل تصور در حداقل یکی از این سه حالت می‌گنجد.)

اکنون برای استفاده از برنامه‌ریزی پویا، آرایه‌ای دو بعدی به اندازه $m \times n$ در نظر می‌گیریم. برای حل مسأله به روش پیمایشی، به راحتی می‌توان از خانه $(0, 0)$ آرایه شروع‌کرد، به‌ترتیب خانه‌های یک سطر را پیمود و سپس به سطر بالا رفته، همین کار را از چپ به راست تکرار کرد. این ترتیبی توپولوژیک از رئوس است. زیرا برای بدست آوردن جواب یک مسأله که متناظر با یک خانه است، احتمالا به همسایه چپ، همسایه پایین و همسایه‌ی مشترک این دو همسایه نیاز خواهیم داشت که همگی قبل از خانه موردنظر مقداردهی شده‌اند.
اگر هم بخواهیم مسأله را از روش بالا به پایین یا بازگشتی حل کنیم، کافی است پایه‌ی مناسبی انتخاب کرده و مقادیر آن را غیربازگشتی وارد آرایه کنیم. در اینجا $i = 0$ و $j = 0$ می‌بایست غیربازگشتی در آرایه مقداردهی شوند. هدف محاسبه‌ی $OPT(m, n)$ است. الگوریتم زیر یک پیاده‌سازی به روش پیمایشی است:

[7] spell checking

SEQUENCE-ALIGNMENT(m, n, A)

```
1  for i = 0 to m
2      A[i, 0]= i × d
3  for j = 0 to n
4      A[0, j]= j × d
5  for i = 1 to m
6      for j = 1 to n
7          A[i, j]= min(a_{x_i y_j} + A[i − 1, j − 1], d + A[i − 1, j], d + A[i, j − 1])
8  return A[m, n]
```

۶ مسأله بلندترین زیر دنباله مشترک[۸]

ورودی: رشته‌های s و t

خروجی: بلندترین زیردنباله مشترک (زیردنباله‌ای از اعضای نه لزوما متوالی رشته اصلی که به همان ترتیب آمده‌اند.)

تمرین یک: به طور شهودی انگار دنبال یک تطابق برای دو رشته هستیم. سعی کنید به کمک مساله هم ترازی، راه حل برای این سوال بیابید. (راهنمایی: هزینه تطابق دو حرف را اگر مشابه بودند، صفر و اگر متفاوت بودند، برابر دو در نظر گیرید. هزینه فاصله یا d را برابر یک در نظر گیرید. ادعا: تطابقی که کمترین هزینه را دارد، اعضای بلندترین زیر دنباله مشترک دو رشته را به ترتیب به هم تطابق داده.)

تمرین دو: مسأله را مستقیما به کمک برنامه‌ریزی پویا حل کنید.

(راهنمایی: $LCS(i, j)$ را بلندترین زیردنباله‌ی مشترکِ i کاراکتر ابتدایی رشته اول با j کاراکتر ابتدایی رشته‌ی دوم در نظر بگیرید و مشابه مسأله هم‌ترازی، رابطه‌ای بازگشتی برای آن بیابید.)

۷ مسأله بلندترین زیردنباله متقارن

ورودی: رشته ی s به طول n

خروجی: طول بزرگترین زیردنباله‌ی متقارن s

تعریف زیردنباله‌ی یک رشته در سوال قبل ذکر شده‌است.

یک تلاش ناموفق برای تعریف زیرمساله مطلوب: $L(i)$ را طول بزرگ‌ترین زیردنباله متقارن از i کاراکتر اول رشته در نظر گیریم. هدف محاسبه‌ی $L(n)$ است. مشکل اینجاست که نمی‌توان رابطه‌ای بازگشتی برای آن بدست آورد.

تلاش موفق: $L(i, j)$ را طول بزرگترین زیردنباله متقارن از کارکتر i ام تا کارکتر j ام رشته‌ی s در نظر بگیریم. با کمی‌دقت می‌توان رابطه بازگشتی زیر را برای محاسبه‌ی $L(1, n)$ به کار برد:

$$L(i,j) = \begin{cases} 0 & i > j \\ 1 & i = j \\ 2 + L(i+1, j-1) & s_i = s_j \\ \max\{L(i+1,j), L(i,j-1)\} & s_i \neq s_j \end{cases}$$

[8] LCS or Longest Common Subsequence

همین که رابطه بازگشتی مشخص شد می‌توانیم با دو روش پیمایشی یا بالا به پایین آن را به سرانجام رسانیم. آرایه‌ای که اطلاعات زیرمساله‌ها در آن نگه داشته می‌شود دوبعدی و اندازه‌اش $n \times n$ است. برای اجرای روش پیمایشی می‌بایست نشان‌دهیم گراف وابستگی‌اش فاقد دور است. اما برای روش بالا به پایین کافی است پایه‌ای مناسب انتخاب کنیم. که در اینجا $i \geq j$ پایه است. (بدون دور بودنِ گراف وابستگی‌ش به این خاطر است که محاسبه‌ی یک خانه، وابسته به محاسبه مقدار خانه‌هایی است که اختلاف شماره سطر و ستون‌شان از اختلاف شماره سطر و شماره ستون آن خانه کمتر است. بنابراین در روش پیمایشی، پیمایش خانه‌هایی که اختلاف سطر و ستون‌شان کمتر است، اولویت دارد. پس پیمایش از قطر اصلی شروع می‌شود که می‌دانیم جواب مسأله در این حالت پایه‌ای برابر 1 است.)

تمرین: سعی کنید این مسأله را به کمک مسأله قبل (بلندترین زیردنباله مشترک) حل کنید.
(راهنمایی اول: با توجه به اینکه مسأله LCS، بر خلاف این مسأله دو رشته به عنوان ورودی می‌گیرد؛ سعی کنید از رشته‌ی s به دو رشته دست پیدا کنید که حل مسأله LCS برای آنان منجر به حل این مسأله شود.)
(راهنمایی دوم: رشته‌ی t را با پیمودن رشته s از انتها به ابتدا بسازید در اینصورت رشته t را معکوس s می‌نامیم. ادعا می‌کنیم جواب مسأله LCS روی این دو رشته همان طول بزرگترین زیردنباله متقارن s است. روی اینکه طول بدست آمده از حل مسأله LCS بزرگتر است یا طول بزرگترین زیردنباله متقارن، حالت‌بندی (فرض خلف) کنید و نشان دهید اگر جواب LCS بزرگتر باشد، زیردنباله‌ای متقارن می‌توان به کمک آن ساخت که طولش حداقل به اندازه جواب LCS باشد.)

۸ تمرین

خواسته‌ی اکثر مسائلی که این جلسه مطرح شد، بر یافتن **اندازه** ی مجموعه‌ای که در شروطی مطلوب صدق کند متمرکز بود.
برای مثال **طول** بزرگترین زیر رشته متقارن.
برای هر یک از سوالات این چنینی، سعی کنید با نوشتن شبه کدی مناسب، علاوه بر یافتن **اندازه** ی مجموعه جواب، خود مجموعه را نیز به دست آورید. (راهنمایی: برای ایده گرفتن، به مسأله چاپ کردن سنگین‌ترین زیرمجموعه مستقل درخت در جلسه ۹ رجوع کنید.)

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۸: برنامه‌ریزی پویا (۲) نگارنده: امیرحسین افشارراد

جلسه‌ی گذشته مفهوم برنامه‌ریزی پویا[1] معرّفی شد و مثال‌هایی نظیر مسأله‌ی جمع‌زدن اعداد، مسأله‌ی کوله‌پشتی[2]، و مسأله‌ی هم‌ترازی[3] (یا مسأله‌ی فاصله‌ی ویرایش[4]) مورد بررسی قرار گرفتند. در این جلسه نیز ابتدا چند مفهوم و نکته‌ی کلّی در ارتباط با برنامه‌ریزی پویا بیان می‌شوند و در ادامه سه مثال مهم نیز مورد بررسی قرار خواهند گرفت.

۱ نکات مقدّماتی

۱. انتخاب نام dynamic programming برای خانواده‌ای از الگوریتم‌ها که در این جلسات مورد بررسی قرار می‌گیرند، هم ریشه‌ی محتوایی و هم ریشه‌ی تاریخی دارد. در واقع لفظ programming نه به معنای «برنامه‌نویسی» بلکه به معنای «برنامه‌ریزی» به کار رفته است و ریشه‌ی انتخاب آن را نیز می‌توان در همان ساختار جدولی که در حل این گونه مسائل به کار می‌روند جست‌وجو کرد. (منظور همان جدولی است که برای حلّ این گونه مسائل شروع به پر کردن آن می‌کنیم و وقتی به انتهای آن می‌رسیم، جواب نهایی مسأله به دست می‌آید.) همچنین به نظر می‌رسد انتخاب لفظ dynamic نیز ــ به نقل از بنیان‌گذار این الگوریتم‌ها ــ بیشتر ریشه‌ی تاریخی دارد و ارتباط مستقیمی با ساختار این الگوریتم‌ها ندارد.

۲. جلسه‌ی قبل به طور مفصّل در مورد دو رویکرد اساسی در برنامه‌ریزی پویا توضیح داده شد: رویکرد بالا به پایین[5] (memoization) و رویکرد پایین به بالا[6]. توجّه کنید که در حلّ یک مسأله به کمک برنامه‌ریزی پویا تفاوت چندانی بین استفاده از این دو روش وجود ندارد و پیشنهاد می‌شود در ابتدا هر روشی که از نظر ذهنی ساده‌تر و طبیعی‌تر به نظر می‌رسد مورد استفاده قرار گیرد و تمرکز اصلی نه بر روی انتخاب از میان این دو رویکرد، بلکه بر روی تعریف صحیح زیرمسأله‌ها و روابط بازگشتی بین آن‌ها باشد. با این حال خوب است به تفاوت‌های جزئی زیر در مورد این دو رویکرد نیز توجّه کنید:

(آ) در استفاده از رویکرد بالا به پایین، فراخوانی توابع بازگشتی تو در تو باعث تحمیل یک سربار[7] به مسأله می‌شوند؛ امّا در رویکرد پایین به بالا چون از فراخوانی توابع بازگشتی استفاده نمی‌شود چنین سرباری وجود ندارد و این یک مزیت برای رویکرد دوم به حساب می‌آید که می‌تواند در عمل زمان اجرای بهتری را برای آن فراهم سازد.

(ب) در استفاده از رویکرد پایین به بالا، در حالت کلّی همه‌ی زیرمسأله‌های ممکن حل می‌شوند و در نهایت حلّ مسأله‌ی اصلی نیز حاصل می‌شود. با این حال ممکن است بسته به ساختار رابطه‌های بازگشتی بین زیرمسأله‌ها، برای رسیدن به جواب اصلی نیازی نباشد تمامی زیر مسأله‌ها حل شوند؛ و حلّ تنها بخشی از آن‌ها برای رسیدن به جواب نهایی کافی باشد. در این صورت استفاده از رویکرد بالا به پایین می‌تواند بهتر باشد، چرا که این روش تنها زیرمسأله‌های مورد نیاز را حل می‌کند؛ حال آن که استفاده از رویکرد پایین به بالا در حالت عادی، همه‌ی زیرمسأله‌های موجود را حل کرده و از این نظر بهینه نمی‌باشد.

[1] Dynamic Programming
[2] Knapsack problem
[3] Alignment problem
[4] Edit distance
[5] top-down
[6] bottom-up
[7] overhead

۳. همان طور که پیش‌تر اشاره شده است، یک بار دیگر تأکید می‌شود که برنامه‌ریزی پویا را به چشم «پر کردن یک جدول» یا «نوشتن یک حلقه‌ی for» نگاه نکنید (اگرچه برخی منابع از برنامه‌ریزی پویا چنین یاد می‌کنند). اصولاً برای طرّاحی یک الگوریتم به سبک برنامه‌ریزی پویا همیشه دو گام اساسی زیر را در نظر بگیرید و به دقّت سعی بر انجام آن‌ها داشته باشید:

- تعریف دقیق زیرمسأله‌ها
- یافتن دقیق رابطه‌ی بین زیرمسأله‌ها

توجّه کنید که صورت‌بندی و تعریف دقیق زیرمسأله‌ها بسیار مهم است؛ بنابراین در این جا قرارداد می‌کنیم که همواره تعریف زیرمسأله‌ها را به صورت دقیق بنویسید؛ و در امتحان‌های درس نیز ذکر نکردن تعریف دقیق زیرمسأله‌ها ــ حتّی در صورت درستی حلّ مسأله و صحّت روابط بین زیرمسأله‌ها ــ می‌تواند منجر به کسر نمره شود.

۴. در نظر داشته باشید که برنامه‌ریزی پویا شاید مهم‌ترین روشی باشد که برای طرّاحی الگوریتم‌ها در این درس مشاهده می‌کنید. یک توصیه‌ی کلّی این است که اگر ایده‌ای برای حلّ یک مسأله نداشتید پیش‌فرض خود را استفاده از برنامه‌ریزی پویا در نظر بگیرید. به طور خاص، برنامه‌ریزی پویا بسیار قدرتمندتر و مهم‌تر از روش حریصانه است؛ و در مسائلی که الگوریتم حریصانه کار می‌کند نیز اگر رویکرد برنامه‌ریزی پویا را پیش بگیرید، احتمالاً در جریان پیاده‌سازی مراحل آن و یافتن رابطه‌ی بین زیرمسأله‌ها خود به خود مشاهده می‌کنید که این رابطه به صورتی ساده و با یک انتخاب حریصانه برقرار می‌شود؛ و به این نتیجه می‌رسید که این مسأله با روش حریصانه نیز قابل حل است.

۵. **(زمان اجرا در مسأله‌های برنامه‌ریزی پویا)** به وضوح در یک مسأله‌ی برنامه‌ریزی پویا، زمان اجرای الگوریتم از رابطه‌ی ۱ به دست می‌آید:

$$\text{زمان اجرا} = \text{تعداد زیرمسأله‌ها} \times \text{زمانی که برای حلّ هر زیرمسأله صرف می‌شود} \quad (۱)$$

در مسأله‌ی کوله‌پشتی که در جلسه‌ی قبل مورد بررسی قرار گرفت، مطابق با توضیحات همان جلسه و نیز مطابق با رابطه‌ی ۱، زمان اجرا $O(nS)$ بود که در آن، n تعداد عناصر موجود و S ظرفیت کوله‌پشتی بود. با توجّه به این که n و S هر دو پارامترهای ورودی مسأله هستند، نمی‌توان بدون اعمال محدودیت بر روی S ادّعا کرد که زمان اجرای این الگوریتم چندجمله‌ای است. در واقع زمان اجرای این الگوریتم *شبه‌چندجمله‌ای* است که تعریف آن (به نقل از مرجع [۲]) به صورت زیر است:

تعریف ۱. *یک الگوریتم عددی در زمان **شبه‌چندجمله‌ای**[8] اجرا می‌شود، هر گاه زمان اجرای آن بر حسب **مقدار عددی** ورودی‌های آن (بزرگ‌ترین ورودی موجود) چندجمله‌ای باشد.*

در واقع الگوریتم‌هایی که زمان اجرای آن‌ها چندجمله‌ای است، تنها تابع *تعداد* ورودی‌ها هستند و این که این ورودی‌ها از نظر مقدار عددی چه وضعیتی دارند اهمیت چندانی برای آن ندارد. به عنوان مثال، بسیاری از الگوریتم‌های مقایسه‌ای مرتّب‌سازی چنین هستند و تنها چیزی که بر زمان اجرا اثر دارد تعداد عناصر موجود برای مرتّب‌سازی است؛ و الگوریتم از نظر زمان اجرا تابع بزرگی یا کوچکی مقادیر عددی این عناصر نیست. امّا الگوریتم‌هایی که زمان اجرای آن‌ها تابع مقدار عددی ورودی‌هاست (و بر حسب آن مقدار، زمان چندجمله‌ای دارد)، طبق تعریف در حالت کلّی زمان اجرای شبه‌چندجمله‌ای خواهند داشت. به عنوان مثال در مسأله‌ی کوله‌پشتی، دیدیم که زمان اجرا به صورت $O(nS)$ درمی‌آمد که در آن، S پارامتری از مسأله است که مقدار عددی آن در زمان اجرای الگوریتم اثرگذار است. در واقع S بر حسب تعداد بیت‌های مورد نیاز برای نمایش آن ($\lg S$) نمایی است ($S = 2^{\lg S}$)؛ امّا اگر S بیش از اندازه بزرگ نباشد (مثلاً از مرتبه‌ی n یا توانی از آن باشد)، زمان اجرای کلّ الگوریتم نیز بر حسب n چندجمله‌ای خواهد بود. با این حال چون در حالت کلّی تضمینی بر روی اندازه‌ی S وجود ندارد، زمان اجرای این الگوریتم طبق تعریف ۱ شبه‌چندجمله‌ای خواهد بود. به صورت شهودی چنین زمانی به خوبی زمان چندجمله‌ای نیست؛ امّا از زمان نمایی بهتر است.

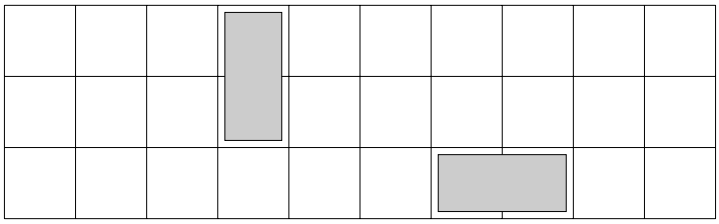
[8] Pseudo-polynomial

۲ مسأله‌ی فرش‌کردن صفحه‌ی مُشبّک

ورودی. عدد طبیعی n

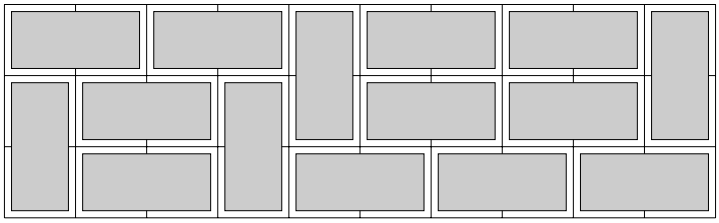
خروجی. تعداد راه‌های فرش‌کردن[۹] یک صفحه‌ی $3 \times n$ با دومینوها (با سایز 2×1)

به عنوان مثال شکل ۱ یک صفحه‌ی مشبّک $3 \times n$ را به ازای $n = 10$ به همراه دو دومینو که یکی به صورت افقی و دیگری به صورت عمودی بر روی صفحه قرار گرفته‌اند، نشان می‌دهد.



شکل ۱: صفحه‌ی مشبّک 3×10

همچنین در شکل ۲ نیز یک نمونه از فرش‌کردن کامل صفحه‌ی مشبّک به ازای $n = 10$ مشاهده می‌شود.



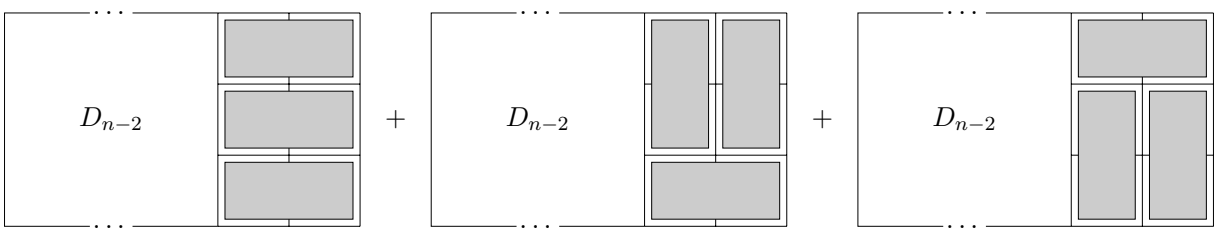
شکل ۲: صفحه‌ی مشبّک 3×10 فرش‌شده توسّط دومینوها

برای حلّ این مسأله با استفاده از برنامه‌ریزی پویا، باید زیرمسأله‌های مناسب تعریف کنیم. اوّلین چیزی که به ذهن می‌رسد، تعریف زیرمسأله‌ی D_n مطابق با رابطه‌ی ۲ است:

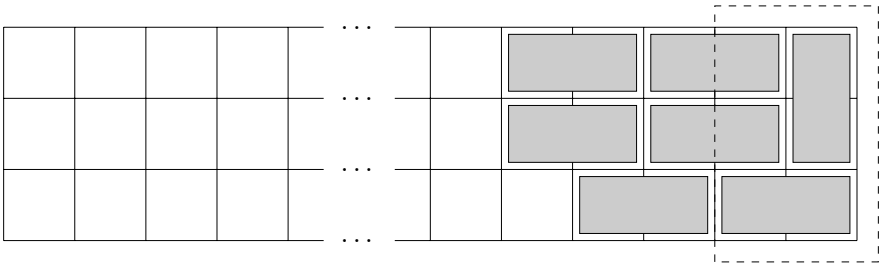
$$D_n = \text{تعداد راه‌های فرش کردن یک مستطیل } 3 \times n \text{ با دومینوها} \qquad (۲)$$

در این حالت چیزی که در ابتدا به ذهن می‌رسد، تقسیم مسأله‌ی اصلی به سه زیرمسأله از جنس D_n مطابق با شکل ۳ است. با این حال باید دقّت کنید که آن چه در شکل ۳ مشاهده می‌شود همه‌ی حالات برای محاسبه‌ی D_n را پوشش نمی‌دهد. به عنوان مثال، شکل ۴ یک نمونه از پاسخ‌های معتبر مسأله است که اگر حالت‌بندی خود را به شکل ۳ محدود کنیم، قادر به بازسازی آن حالت نخواهیم بود. می‌توان نمونه‌ی شکل ۴ را با همان الگویی که روی این شکل وجود دارد ادامه داد، و این یعنی چنین پاسخی به هیچ وجه با زیرمسأله‌هایی از جنس D_j قابل مدل‌سازی نخواهد بود و به نظر می‌رسد که ایده‌ی فعلی برای برنامه‌ریزی پویا شکست خورده است. با این حال، این مشکل به کمک تعریف انواع دیگری از زیرمسأله‌ها قابل حل است. علاوه بر زیرمسأله‌ی D_n که در رابطه‌ی ۲ تعریف شده بود، دو زیرمسأله‌ی A_n و B_n را نیز به ترتیب در روابط ۳ تعریف می‌کنیم.

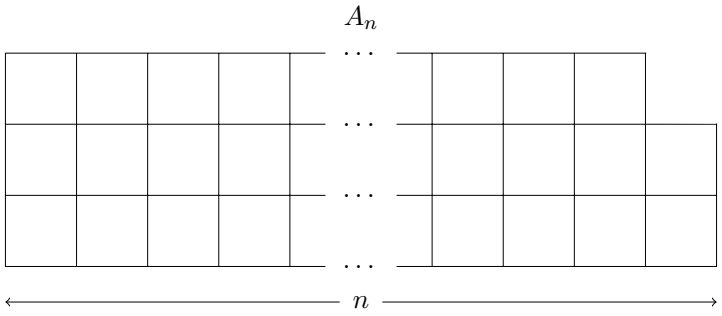
$$\begin{aligned} A_n &= \text{تعداد راه‌های فرش کردن صفحه‌ی مشبّک شکل ۵ با دومینوها} \\ B_n &= \text{تعداد راه‌های فرش کردن صفحه‌ی مشبّک شکل ۶ با دومینوها} \end{aligned} \qquad (۳)$$



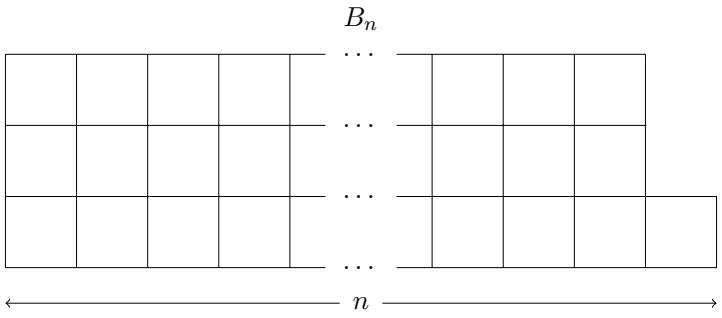
شکل ۳: تقسیم مسأله‌ی اصلی به سه زیرمسأله از جنس D_n



شکل ۴: نمونه‌ای از پاسخ‌های معتبر مسأله که با استفاده از حالت‌بندی‌های شکل ۳ قابل دستیابی نمی‌باشد.



شکل ۵: فرم صفحه‌ی مشبّک زیرمسأله‌ی A_n



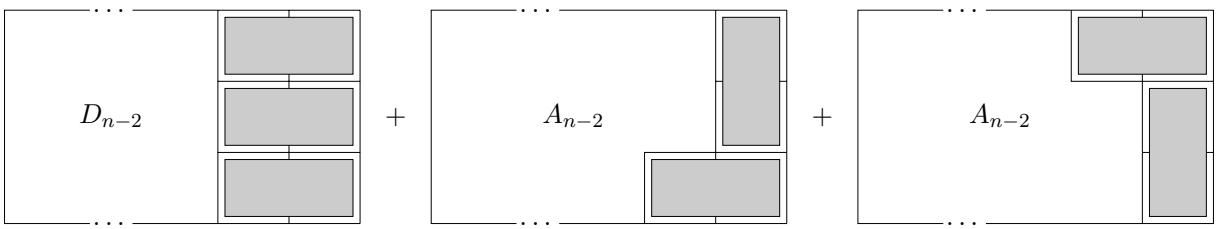
شکل ۶: فرم صفحه‌ی مشبّک زیرمسأله‌ی B_n

حال می‌توانیم مسأله‌ی اصلی (یعنی D_n) را بر حسب زیرمسأله‌های تعریف‌شده بیان کنیم. برای این کار کافی است به چگونگی پر شدن ستون آخر جدول توجّه کنیم. اگر سه خانه‌ی ستون آخر توسّط سه دومینوی مختلف پر شده باشند، یک مسأله از جنس D_{n-2} خواهیم داشت. اگر سه خانه‌ی ستون آخر توسّط دو دومینوی مختلف پر شده باشند، بسته به چگونگی قرارگیری این دومینوها، دو مسأله از جنس

[9] tiling

A_{n-1} خواهیم داشت. شکل ۷ این سه حالت را نمایش می‌دهد. به این ترتیب، رابطه‌ی ۴ بین زیرمسأله‌ها برقرار می‌شود.

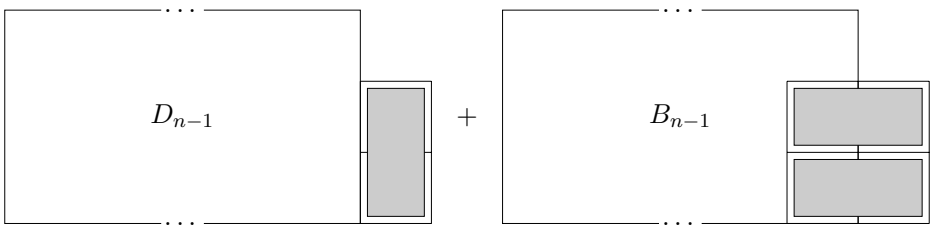
$$D_n = D_{n-2} + 2A_{n-1} \tag{۴}$$



شکل ۷: تقسیم مسأله‌ی D_n به سه زیرمسأله از جنس‌های D_n و A_n

با این حال، رابطه‌ی ۴ برای حلّ مسأله‌ی اصلی کافی نخواهد بود، چرا که دو زیرمسأله‌ی مختلف از نوع A و D در آن درگیر هستند و در حال حاضر رابطه‌ای برای حلّ زیرمسأله‌ی A_n نداریم؛ بنابراین به یک رابطه‌ی دیگر نیز نیاز داریم. برای این کار، زیرمسأله‌ی A_n را به زیرمسأله‌های کوچک‌تر تبدیل می‌کنیم تا معادله‌ی ۵ حاصل شود. شکل ۸ با حالت‌بندی بر روی وضعیت پر شدن ستون آخر، درستی معادله‌ی ۵ را توجیه می‌کند.

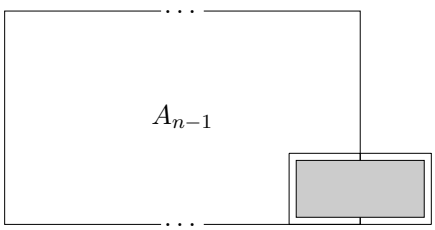
$$A_n = D_{n-1} + B_{n-1} \tag{۵}$$



شکل ۸: تقسیم مسأله‌ی A_n به دو زیرمسأله از جنس‌های B_n و D_n

مجدّداً مشاهده می‌شود که با نوشتن معادله‌ی ۵، یک نوع جدید از زیرمسأله‌ها، یعنی زیرمسأله‌ی نوع B به مجموعه‌ی معادلات اضافه شد؛ بنابراین معادلات ۴ و ۵ برای حلّ مسأله کافی نخواهند بود. این بار لازم است رابطه‌ای بازگشتی برای B_n بیابیم. مطابق با شکل ۹ و با حالت بندی بر روی آخرین ستون (که تنها یک حالت برای پر کردن آن وجود دارد)، زیرمسأله‌ی B_n به سادگی تبدیل به زیرمسأله‌ی A_{n-1} می‌شود (معادله‌ی ۶).

$$B_n = A_{n-1} \tag{۶}$$



شکل ۹: تبدیل زیرمسأله از جنس B_n به زیرمسأله‌ی کوچک‌تر از جنس A_n

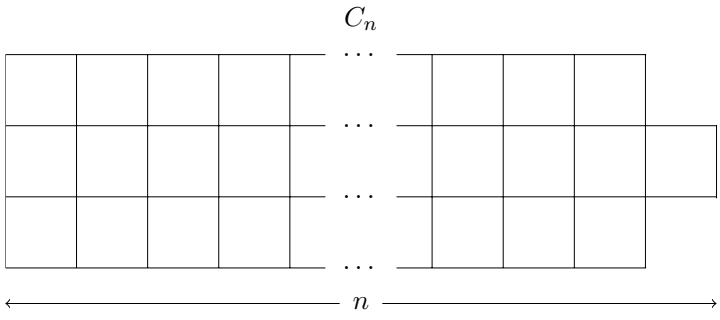
به این ترتیب با کنار هم قرار دادن سه معادله‌ی ۴، ۵، و ۶ مجموعه معادلات ۷ به دست می‌آید.

$$\begin{cases} D_n = D_{n-2} + 2A_{n-1} \\ A_n = D_{n-1} + B_{n-1} \\ B_n = A_{n-1} \end{cases} \quad (۷)$$

به این ترتیب به کمک سه معادله‌ی بازگشتی ۷، سه زیرمسأله‌ی A_n و B_n و D_n قابل حل خواهند بود و پاسخ نهایی مسأله نیز به دست می‌آید. با در نظر گرفتن رویکرد پایین به بالای برنامه‌ریزی پویا، این کار معادل با پر کردن هم‌زمان سه جدول (یا پر کردن جدولی که در هر خانه‌ی آن سه مؤلّفه موجود است) می‌باشد. ضمناً توجّه کنید که جدول‌ها در این مسأله در واقع آرایه‌هایی به طول n هستند.

مشاهده کردیم که در این مسأله برای به دست آوردن D_n مجبور به تعریف زیرمسأله‌های دیگری شدیم که به صورت صریح حلّ آن‌ها هدف ما نبود. این اتّفاق شبیه به مواردی از کاربرد استقرای ریاضی است که برای اثبات حکمی، اگر همان حکم را مستقیماً در نظر بگیریم اثبات به کمک استقرا امکان‌پذیر نخواهد بود، امّا اگر حکم قوی‌تری را در نظر بگیریم (به دلیل آن که فرض استقرا نیز قوی‌تر می‌شود) اثبات حکم میسّر می‌شود. (نمونه‌های از چنین کاربردی از استقرا را در حلّ روابط بازگشتی در درس ساختمان داده مشاهده کرده‌اید.)

به این ترتیب حلّ این مسأله تمام شده است. همچنین شکل ۱۰ نیز نشان‌دهنده‌ی زیرمسأله‌ی دیگری برای این مسأله (C_n) است که در کلاس درس بیان شد، امّا در حلّ مسأله نیازی به استفاده از آن نشد و به همین دلیل، آن را صرفاً برای تکمیل مبحث در پایان ذکر کرده‌ایم.



شکل ۱۰: فرم صفحه‌ی مشبّک زیرمسأله‌ی C_n

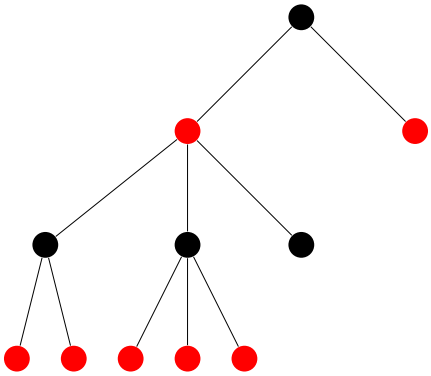
۳ (برنامه‌ریزی پویا روی درخت‌ها) مسأله‌ی سنگین‌ترین زیرمجموعه‌ی مستقل در درخت

ورودی. یک درخت $T = (V, E)$ و یک تابع وزن $w : V \to \mathbb{Q}_+$.
خروجی. زیرمجموعه‌ی مستقلی از درخت که بیشترین وزن را داشته باشد.

که منظور از یک زیرمجموعه‌ی مستقل، تعریف ۲ است:

تعریف ۲. *اگر $G = (V, E)$ یک گراف باشد، $U \subseteq V$ را یک زیرمجموعه‌ی مستقل از رئوس گراف G گوییم، هرگاه بین هیچ دو رأس از U یالی از گراف G وجود نداشته باشد.*

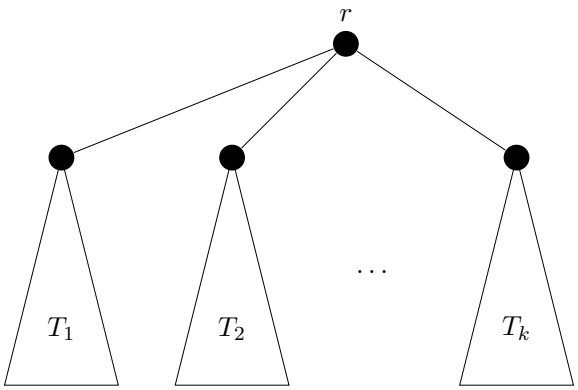
به عنوان مثال، شکل ۱۱ درختی را نشان می‌دهد که یک زیرمجموعه‌ی مستقل آن (رئوس قرمزرنگ) مشخّص شده است. در واقع در این شکل ۷ رأس به رنگ قرمز مشخّص شده‌اند که بین هیچ دوتایی از آن‌ها یالی وجود ندارد.



شکل ۱۱: یک زیرمجموعه‌ی مستقل از یک درخت

در ادامه می‌خواهیم به کمک برنامه‌ریزی پویا، الگوریتمی برای حلّ این مسأله ارائه دهیم. دقّت کنید که این مسأله در حالتی که وزن همه‌ی رئوس برابر در نظر گرفته شود (مثلاً برابر با ۱)، تبدیل به مسأله‌ی پیدا کردن بزرگ‌ترین زیرمجموعه‌ی مستقل از درخت می‌شود؛ با این حال الگوریتم حل برای حالتی که رئوس وزن‌های متفاوت داشته باشند (دقّت کنید که تابع وزن بر روی رئوس تعریف شده است و نه یال‌ها) نیز کاملاً مشابه است و از نظر الگوریتم، تفاوتی میان این دو حالت وجود ندارد.

برای حلّ مسأله، اوّلین و مهم‌ترین گام انتخاب درست زیرمسأله‌هاست. با توجّه به این که مسأله‌ی اصلی بر روی یک درخت تعریف شده است، به وضوح زیرمسأله‌ها نیز باید از جنس درخت بوده، یا به عبارتی زیردرخت‌هایی از درخت اصلی باشند. همچنین برای آن که بتوانیم ارتباط ساختارمندی بین زیرمسأله‌ها برقرار کنیم، درخت مسأله را از یک رأس دلخواه ریشه‌دار می‌کنیم (یعنی یک رأس دلخواه را به عنوان ریشه‌ی درخت انتخاب می‌کنیم). در این صورت اگر ریشه‌ی درخت را r بنامیم، می‌توان فرزندان r را در نظر گرفت و زیردرخت‌هایی که هر یک از فرزندان r ریشه‌های آن هستند را به عنوان زیرمسأله‌های مطلوب در نظر گرفت. شکل ۱۲ صورتی از تقسیم مسأله به این زیردرخت‌ها را نشان می‌دهد.



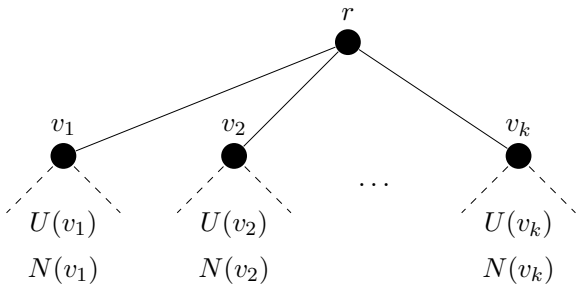
شکل ۱۲: فرم کلّی درخت ریشه‌دار و زیردرخت‌های فرزندان ریشه‌ی درخت

حال واضح است که اگر r در جواب مسأله نباشد، آنگاه پاسخ مسأله اجتماع پاسخ زیرمسأله‌های تعریف‌شده بر روی T_iها ($1 \leq i \leq k$) خواهد بود. علّت این امر آن است که هیچ یالی بین T_i و T_j ($i \neq j$) وجود ندارد، بنابراین برای یافتن بزرگ‌ترین زیرمجموعه‌ی مستقل کلّی، کافی است بزرگ‌ترین زیرمجموعه‌ی مستقل را در هر یک از T_iها بیابیم و حاصل را اجتماع بگیریم. امّا اگر خود r در جواب اصلی مسأله حضور داشته باشد، آنگاه هیچ یک ازفرزندان r مجاز به حضور در پاسخ نهایی نیستند؛ و به عبارتی در آن حالت باید مسأله‌ی یافتن بزرگ‌ترین زیرمجموعه‌ی مستقل را در هر یک از T_iها با این قید که *ریشه‌ی درخت در جواب نباشد*، پیدا کنیم. به این ترتیب به نظر می‌رسد

که لازم است دو نوع زیرمسأله تعریف کنیم. صورت‌بندی این زیرمسأله‌ها در رابطه‌ی ۸ بیان شده است.

$$\begin{aligned} U(r) &= \text{وزن سنگین‌ترین زیرمجموعه‌ی مستقل شامل } r \text{ در درخت به ریشه‌ی } r \\ N(r) &= \text{وزن سنگین‌ترین زیرمجموعه‌ی مستقل در درخت به ریشه‌ی } r \text{ که شامل } r \text{ نباشد} \end{aligned} \tag{۸}$$

حال با تعریف این زیرمسأله‌ها و با دیدگاه بازگشتی، مطابق با شکل ۱۳ فرض می‌کنیم که همه‌ی زیرمسأله‌ها برای فرندان r حل شده است و می‌خواهیم به کمک نتایج آن، دو زیرمسأله‌ی U و N را برای خود r نیز حل کنیم.



شکل ۱۳: ساختار کلّی درخت و زیرمسأله‌ها

یافتن رابطه‌ی بازگشتی برای $U(r)$ و $N(r)$ ساده است. برای محاسبه‌ی $U(r)$، طبق تعریف زیرمسأله‌ی U از رابطه‌ی ۸ فرض می‌شود که خود r جزء جواب است، بنابراین هیچ یک از فرزندان r مجاز به حضور در جواب نیستند؛ بنابراین باید برای همه‌ی فرزندان r مسأله‌ی N را حل کنیم. به این ترتیب رابطه‌ی بازگشتی ۹ حاصل می‌شود که در آن منظور از $\mathcal{C}(r)$ مجموعه‌ی فرزندان r است.

$$U(r) = w(r) + \sum_{v \in \mathcal{C}(r)} N(v) \tag{۹}$$

همچنین با استدلال مشابه برای محاسبه‌ی $N(r)$، چون فرض می‌کنیم خود r در زیرمجموعه‌ی مستقل مورد جست‌وجو حضور ندارد، محدودیتی برای حضور یا عدم حضور فرزندان r در پاسخ وجود ندارد. بنابراین برای یافتن بزرگ‌ترین جواب، باید برای هر یک از فرزندان r مثل v، بیشینه‌ی $U(v)$ و $N(v)$ را به عنوان وزن سنگین‌ترین زیرمجموعه‌ی مستقل زیردرخت مربوطه در نظر بگیریم. به این ترتیب رابطه‌ی بازگشتی ۱۰ حاصل می‌شود.

$$N(r) = \sum_{v \in \mathcal{C}(r)} \max\{U(v), N(v)\} \tag{۱۰}$$

به این ترتیب با کنار هم قرار دادن دو رابطه‌ی بازگشتی از معادلات ۹ و ۱۰، می‌توان جواب تمامی زیرمسأله‌ها را به صورت بازگشتی محاسبه کرد و به پاسخ اصلی مسأله دست یافت. همچنین به وضوح می‌توان با استفاده از یک تابع بازگشتی، شبه‌کدی برای محاسبه‌ی $U(r)$ و $N(r)$ نوشت. این شبه کد در ادامه قابل مشاهده است:

```
MaxInd(r)
1  N(r) = 0
2  U(r) = w(r)
3  for each v in C(r)
4      MaxInd(v)
5      U(r) = U(r) + N(v)
6      N(r) = N(r) + max{U(v), N(v)}
```

دقّت کنید که شبه‌کد فوق نمونه‌ای از پیاده‌سازی رویکرد بالا به پایین (memoization) در استفاده از برنامه‌ریزی پویاست؛ با این حال در آن تمهیدی برای جلوگیری از فراخوانی چندباره‌ی تابع بازگشتی بر روی ورودی یکسان اندیشیده نشده است. (از جلسه‌ی قبل به خاطر دارید که در حالت کلّی هنگام پیاده‌سازی این رویکرد لازم است توجّه کنیم که تابع بازگشتی برای هر ورودی متمایز، حدّاکثر یک بار اجرا شود و خروجی آن ذخیره گردد تا در دفعات بعدی، از خروجی‌ای که قبلاً محاسبه شده استفاده گردد. همچنین در همان جلسه بررسی شد که عدم انجام چنین کاری می‌تواند مرتبه‌ی زمانی اجرای الگوریتم را به شدّت نابهینه کند.) علّت این امر در این مسأله آن است که ماهیت مسأله به گونه‌ای است که قطعاً تابع بازگشتی بر روی یک ورودی متمایز بیش از یک بار اجرا نمی‌شود. برای آن که درک بهتری از علّت این موضوع به دست آید، می‌توان شبه کد فوق را به یک روش «پیمایش درخت» تشبیه کرد. به عبارتی، آن چه در اصل انجام می‌شود مشابه با پیاده‌سازی DFS است، با این تفاوت که هرگاه اجرای تابع بازگشتی مربوط به DFS روی یکی از رئوس به پایان می‌رسد، دو مقدار عددی نیز (U و N مربوط به آن رأس) محاسبه می‌شوند و سپس فرآیند به پایان می‌رسد. با در نظر گرفتن این دیدگاه، تابع بازگشتی بر روی هر رأس، یک و فقط یک بار فراخوانی می‌شود.

همچنین (همان طور که در جلسه‌ی قبل اشاره شده است) در مسائل برنامه‌ریزی پویا، به صورت ابتدایی خروجی الگوریتم یک *مقدار بهینه* خواهد بود. با این حال این سؤال در هر مسأله‌ای قابل طرح و پاسخ دادن است که «کدام انتخاب، منجر به حصول این مقدار بهینه می‌گردد؟». به طور خاص در این مسأله، الگوریتم فعلی وزن سنگین‌ترین زیرمجموعه‌ی مستقل از درخت را به دست می‌آورد؛ امّا این سؤال مطرح است که این وزن حاصل از انتخاب کدام مجموعه از رئوس بوده است. در ادامه شبه‌کدی برای چاپ کردن رئوس مربوط به سنگین‌ترین زیرمجموعه‌ی مستقل از رئوس ارائه می‌شود. دقّت کنید که در این شبه‌کد، پارامتر t مشخّص می‌کند که پاسخ مسأله از نوع U محاسبه شود یا از نوع N.

```
PrintMaxInd(r, t)
1  if t == U
2      print(r)
3      for each x in C(r)
4          PrintMaxInd(x, N)
5  else
6      for each x in C(r)
7          if N(x) > U(x)
8              PrintMaxInd(x, N)
9          else
10             PrintMaxInd(x, U)
```

ضمناً جالب است بدانید که مسأله‌ی یافتن سنگین‌ترین زیرمجموعه‌ی مستقل برای یک گراف دلخواه مسأله‌ی سختی است و الگوریتم چندجمله‌ای برای حلّ آن پیدا نشده است. با این حال مشاهده می‌شود که این مسأله برای درخت‌ها (یا گروهی بزرگ‌تر ــ گراف‌هایی با عرض درختی ثابت ــ) قابل حل است.

۴ درخت دودویی جست‌وجوی بهینه

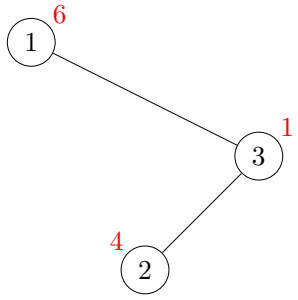
ورودی. n عنصر با فرکانس دسترسی به هر کدام، f_i. (که منظور از f_i می‌تواند درصد دسترسی یا تعداد دفعات دسترسی به عنصر iاُم باشد و $1 \leq i \leq n$)

خروجی. یک درخت جست‌وجوی دودویی متشکّل از این عناصر که مجموع زمان‌های دسترسی به عناصر (با فرکانس‌های f_i) کمینه باشد.

دقّت کنید که چون در ساختار درخت جست‌وجوی دودویی تنهای مقایسه‌ی عناصر با یک‌دیگر مهم است (و مقادیر مطلق آن‌ها اهمیّت ندارد) می‌توان در ورودی تنها عدد n را دریافت کرد و فرض کرد که عناصر موجود در درخت اعداد 1 تا n هستند. به این ترتیب، f_i نیز تعداد (یا درصد) دفعات دسترسی به عنصر i خواهد بود. همچنین با توجّه به ساختار درخت جست‌وجوی دودویی، زمان دسترسی به عنصر i، یکی بیشتر از عمق آن خواهد بود (عمق ریشه را صفر در نظر بگیرید). به این ترتیب، هدف این مسأله آن است که عناصر را به گونه‌ای در درخت قرار دهد که عبارت ۱۱ کمینه شود (کلّ هزینه‌ی دسترسی به یک عنصر، حاصل‌ضرب هزینه‌ی یک بار دسترسی در تعداد دفعات دسترسی به آن عنصر می‌باشد):

$$\text{Cost} = \sum_{i=1}^{n} f_i(\text{depth}(i) + 1) \tag{۱۱}$$

به عنوان مثال فرض کنید $n = 3$ و مقادیر f_iها به صورت $f_1 = 6, f_2 = 4, f_3 = 1$ داده شده باشند. در این صورت شکل ۱۴ نمونه‌ای از درخت ساخته‌شده برای این عناصر را نشان می‌دهد. همچنین تعداد دفعات دسترسی به هر رأس از درخت نیز با رنگ قرمز در کنار آن نوشته شده است.

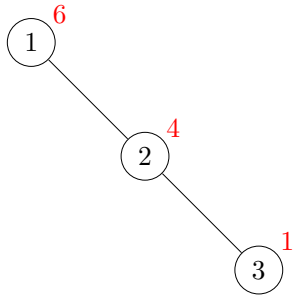


شکل ۱۴: یک نمونه درخت برای $n = 3$

برای درخت شکل ۱۴ مقدار کلّ هزینه‌ی دسترسی مطابق با رابطه‌ی ۱۱ قابل محاسبه است:

$$\text{Cost} = \sum_{i=1}^{n} f_i(\text{depth}(i) + 1) = 6 \times 1 + 1 \times 2 + 4 \times 3 = 20 \tag{۱۲}$$

واضح است که هزینه‌ی محاسبه‌شده در رابطه‌ی ۱۲ بهینه نیست، و می‌توان با تغییر ساختار درخت آن را کاهش داد. به عنوان مثال، شکل ۱۵ چینش دیگری از همین عناصر در یک درخت جست‌وجوی دودویی است؛ به گونه‌ای که هزینه‌ی کلّ آن (محاسبه‌شده در معادله‌ی ۱۳) کم‌تر شده است.



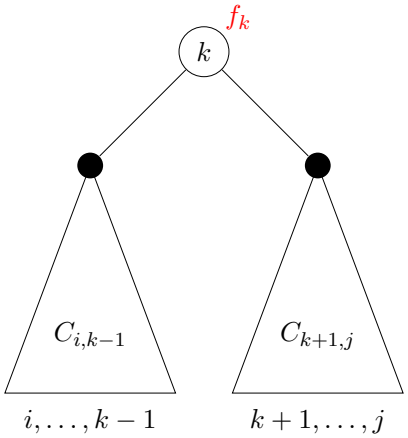
شکل ۱۵: یک نمونه درخت برای $n = 3$

$$\text{Cost} = \sum_{i=1}^{n} f_i(\text{depth}(i) + 1) = 6 \times 1 + 4 \times 2 + 1 \times 3 = 17 \tag{۱۳}$$

در ادامه به سراغ حلّ مسأله در حالت کلّی می‌رویم. باید زیرمسأله‌ی مناسب را تعریف کنیم و رابطه‌ی بازگشتی بین زیرمسأله‌ها را برقرار سازیم. با اندکی دقّت واضح است که یک انتخاب مناسب برای تعریف زیرمسأله‌ی مناسب، رابطه‌ی ۱۴ است:

$$C_{ij} = \text{هزینه‌ی کلّ دسترسی‌ها در درخت جست‌وجوی دودویی بهینه روی کلیدهای } i \text{ تا } j \tag{۱۴}$$

حال برای برقراری یک رابطه‌ی بازگشتی بین زیرمسأله‌های مختلف، کافی است با توجّه به ساختار درخت جست‌وجوی دودویی توجّه کنیم که در عمل برای هر زیرمسأله تنها کافی است عنصری که در ریشه‌ی درخت قرار می‌گیرد را انتخاب کنیم. انتخاب این عنصر، سایر عناصر را دقیقاً به دو زیردرخت راست و چپ تقسیم می‌کند، که برای هر کدام از آن‌ها زیرمسأله‌ی مربوطه به صورت بازگشتی قابل حل است. شکل ۱۶ نموداری از این تقسیم‌بندی را نشان می‌دهد.



شکل ۱۶: انتخاب ریشه‌ی درخت جست‌وجوی دودویی و تبدیل مسأله به زیرمسأله‌های کوچک‌تر

حال می‌توانیم رابطه‌ی ریاضی بین C_{ij}ها را به دست آوریم. اگر $i > j$ باشد که درخت تهی خواهد بود و هزینه‌ی آن نیز برابر با صفر است. اگر $i = j$ باشد، تنها یک عنصر داریم که هزینه‌ی f_i تحمیل خواهد کرد. برای حالت $i < j$، باید همه‌ی حالات مختلف برای انتخاب ریشه‌ی درخت (k) را بررسی کنیم و کمینه‌ی هزینه‌ی این حالات را به عنوان پاسخ در نظر بگیریم. برای این کار، به ازای هر k، می‌توانیم از هزینه‌های دو زیردرخت ایجادشده (مطابق با شکل ۱۶) استفاده کنیم. تنها نکته‌ای که لازم است به آن توجّه کنیم این است که هزینه‌ی این دو زیردرخت در این حالت دقیقاً برابر با $C_{i,k-1}$ و $C_{k+1,j}$ نخواهد بود. علّت این امر آن است که این هزینه‌ها با فرض آن که ریشه‌ی درخت در عمق صفر باشد محاسبه شده است؛ و این در حالی است که این دو زیردرخت در مسأله‌ی اصلی ما از عمق ۱ شروع شده‌اند، بنابراین برای هر رأس مثل l، به اندازه‌ی f_l نیز هزینه‌ی اضافی تحمیل می‌شود. در نتیجه برای همه‌ی عناصر دو زیردرخت، یعنی همه‌ی اعداد i تا j *به استثنای* k باید هزینه‌ی بیشتری به اندازه‌ی f_l بپردازیم. از طرفی برای خود ریشه‌ی درخت نیز (طبق رابطه‌ی ۱۱) لازم است هزینه‌ی f_k داده شود. به این ترتیب، باید برای تمامی عناصر از i تا j هزینه‌ی اضافی f_l (مازاد بر هزینه‌های $C_{i,k-1}$ و $C_{k+1,j}$) پرداخته شود؛ و این رابطه‌ی بازگشتی مورد نظر را به صورت معادله‌ی ۱۵ در می‌آورد:

$$C_{ij} = \begin{cases} 0 & i < j \\ f_i & i = j \\ \min_{i \le k \le j} \left\{ C_{i,k-1} + C_{k+1,j} + \sum_{l=i}^{j} f_l \right\} & i < j \end{cases} \tag{۱۵}$$

با توجّه به رابطه‌ی بازگشتی ۱۵، واضح است که برای محاسبه‌ی C_{ij} لازم است که در گام‌های قبلی، مسأله‌هایی که طول بازه‌ی آن‌ها (منظور از طول بازه برای مسأله‌ی C_{ij} عدد $|j - i|$ است) کوچک‌تر بوده است، حل شده باشند.

همچنین می‌توان زمان اجرای این الگوریتم را نیز به سادگی تحلیل کرد. تعداد زیرمسأله‌هایی که حل می‌شوند از $O(n^2)$ است (به ازای همه‌ی حالات انتخاب i و j) و برای هر زیرمسأله نیز هزینه‌ی $O(n)$ صرف می‌شود (زیرا باید همه‌ی حالات k را بررسی کرد که در حالت کلّی تعداد حالات مختلف برای k نیز $O(n)$ است). به این ترتیب زمان اجرای این الگوریتم $O(n^3)$ خواهد بود. در تمرینی از مرجع [۱]، روشی برای حلّ این مسأله در زمان $O(n^2)$ ارائه شده است.

۵ تمارین اختیاری

۱. به چند طریق می‌توان عدد n داده‌شده را به صورت مجموعی از اعداد ۱ و ۳ و ۴ نوشت؛ به گونه‌ای ترتیب در جمع مهم نباشد. (یعنی مثلاً $3+1$ و $1+3$ یک حالت شمرده می‌شوند. دقّت کنید در جلسه‌ی قبلی درس، این مسأله در حالتی که ترتیب در جمع مهم باشد حل شده است.)

۲. مسأله‌ی یافتن سنگین‌ترین زیرمجموعه‌ی مستقل درخت (بخش ۳) را در نظر بگیرید. فرض کنید علاوه بر درخت، عدد k هم در ورودی داده شده است. می‌خواهیم بیشینه‌ی وزن یک زیرمجموعه‌ی مستقل با اندازه‌ی k را پیدا کنیم. الگوریتمی برای این مسأله ارائه دهید.

۳. جلسه‌ی قبل در ارتباط با مسأله‌ی هم‌ترازی و مسأله‌ی طولانی‌ترین زیررشته‌ی مشترک[۱۰] (LCS) بحث شد. همچنین چگونگی حلّ مسأله‌ی طولانی ترین زیررشته‌ی متقارن[۱۱] (LPS) به کمک مسأله‌ی LCS نیز به عنوان تمرین مطرح شد. علاوه بر این‌ها، یک مسأله‌ی جالب دیگر، مسأله‌ی طولانی‌ترین زیردنباله‌ی صعودی (غیرنزولی)[۱۲] (LIS) است. به عنوان تمرین با استفاده از مسأله‌ی LCS، مسأله‌ی LIS را حل کنید؛ یعنی با فرض این که الگوریتمی برای یافتن طولانی‌ترین زیررشته‌ی مشترک دو رشته‌ی داده‌شده داشته باشید، الگوریتمی طرّاحی کنید که طولانی‌ترین زیردنباله‌ی صعودی (غیرنزولی) از یک دنباله‌ی داده‌شده را محاسبه کند.

مراجع

[1] Cormen, Thomas H., et al. *Introduction to Algorithms*. 3rd ed., MIT Press, 2009, pp. 397-403.

[2] *"Pseudo-polynomial time"* (2019) Wikipedia. Available at: https://en.wikipedia.org/wiki/Pseudo-polynomial_time (Accessed: 6 March 2020).

[10] Longest Common Substring
[11] Longest Palindromic Substring
[12] Longest Increasing Subsequence

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۹: برنامه‌ریزی پویا ۳ نگارنده: مهدی مستانی

در جلسه‌ی قبل تعدادی مسئله که با کمک برنامه‌ریزی پویا قابل‌حل هستند (مانند مسئله‌ی سنگین‌ترین زیرمجموعه‌ی مستقل درخت) را بررسی کردیم. در این جلسه نیز در ادامه‌ی این موضوع به بررسی مثال‌های دیگری در این رابطه می‌پردازیم.

۱ بازی سکه نوبتی

بازی سکه نوبتی[1] بدین صورت است که n سکه با ارزش‌های $v_1, v_2, v_3, ..., v_n$ در یک ردیف داریم (که در آن n عددی زوج است).دو بازیکن به نوبت هرکدام یک سکه از یک سر ردیف برمی‌دارند. برنده این بازی کسی است که در انتها مجموع ارزش سکه‌هایش بیشتر باشد.

سوال اولی که قصد داریم به آن پاسخ دهیم این است که آیا نفر اول استراتژی برد دارد؟

پاسخ این سوال مثبت است. کافی است نفر اول دو مجموع $(v_1 + v_3 + v_5 + \dots)$ و $(v_2 + v_4 + v_6 + v_8 + \dots)$ را محاسبه کند(جمع سکه‌های زوج و فرد).اگر مجموع اندیس‌های فرد بزرگتر بود در مرحله‌ی اول v_1 را برداشته و بدین صورت نفر دوم مجبور است یکی از دو سکه‌ی v_2 یا v_n را بردارد و در مرحله‌ی بعد نیز نفر اول دوباره سکه با اندیس فرد موجود را برمی‌دارد و این روند ادامه پیدا می‌کند. در انتها تمامی سکه‌های فرد در دست نفر اول و تمامی سکه‌های زوج در دست نفر دوم خواهد بود و نفر اول برنده است. برای حالتی که مجموع اندیس‌های زوج بیشتر شود هم به صورت مشابه عمل می‌کند.

حال سوال اصلی‌ای که می‌خواهیم آن را بررسی کنیم این است که بیشترین مقدار پولی که نفر اول می‌تواند بدست آورد چقدر است؟(می‌دانیم که هر دو نفر بهترین استراتژی را برای بدست آوردن حداکثر مقدار ممکن استفاده می‌کنند. این مسئله را می‌توان به این صورت هم بررسی کرد که نفر اول سعی می‌کند تا جای ممکن ارزش پول خود را بیشتر کند و نفر دوم سعی می‌کند تا جای ممکن مقدار پولی که نفر اول می‌تواند بدست آورد را کم کند(این کار معادل این است که خودش تا جای ممکن مقدار بیشتری بدست آورد).

قصد داریم که این مسئله با را با برنامه‌ریزی پویا حل‌کنیم.برای این‌کار باید زیرمسئله‌ها را به‌خوبی تعریف کنیم و برای حل مسئله اصلی، انتخاب‌ها را به طرز صحیحی انجام دهیم.انتخابی که نفر اول دارد، دو حالت دارد(یا سکه ابتدا را بر دارد یا سکه انتها را).در واقع هر بازیکن در نوبت خود(به جز آخرین نوبت بازیکن دوم)،دو انتخاب دارند و یا باید از بین سکه‌های باقیمانده ، سکه ابتدایی را بردارد یا سکه انتهایی و هر یک از این سکه‌ها را که انتخاب کند، این دنباله سکه‌ها تبدیل به یک دنباله جدید با طول کمتر می‌شود.

پس اگر در لحظه‌ای از بازی، بازی بر روی آرایه‌ی $\{v_i, v_{i+1}, \dots, v_j\}$ از سکه‌ها انجام شود، آنگاه برای حل مسئله ، مقدار $C(i,j)$ را تعریف می‌کنیم:

$C(i,j)$: (بیشترین ارزشی که نفر اول می‌تواند کسب کند (با فرض بازی بر روی دنباله سکه‌های $\{v_i, v_{i+1}, \dots, v_j\}$

[1] Alternating Coin Game

که البته چون در ابتدا فرض کرده بودیم که تعداد سکه‌ها در ابتدای بازی زوج است، پس باید طول این دنباله عددی زوج باشد(در واقع باید داشته باشیم که عدد $j-i+1$ ، مقداری زوج است).

حال در دنباله سکه‌های $\{v_i, v_{i+1}, \ldots, v_j\}$ نفر اول دو انتخاب دارد؛ یا v_i را انتخاب می‌کند یا v_j.

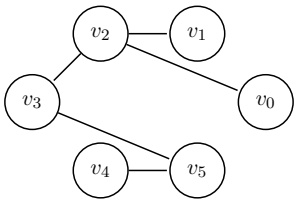
حال باید بین این دو انتخاب، آن انتخابی را انجام دهد که با توجه به انتخاب‌های ممکن باقی‌مانده برای نفر دوم،ارزش نهایی خودش بیشتر شود. پس برای مثال اگر نفر اول v_i را بردار، نفر دوم سعی دارد که در دنباله سکه‌های $\{v_{i+1}, v_{i+2}, \ldots, v_j\}$ سود خودش را بیشینه کند که همانطور که بیان شد،این کار معادل این است که سود نفر اول را کمینه کند. پس نفر انتخابی را انجام می‌دهد که در دنباله‌ی پس از انتخابش، نفر اول سود کمتری را بتواند کسب کند.پس اگر نفر دوم v_{i+2} را انتخاب کند، نفر اول می‌تواند در ادامه بازی ، سود $C(i+2,j)$ را به‌دست آورد و اگر نفر دوم v_j را انتخاب کند، نفر اول در ادامه بازی می‌تواند سود $C(i+1,j-1)$ را حاصل کند که چون نفر دوم قصد دارد که سود نفر دوم را کمینه کند،پس باید بین این دو حالت آن حالتی را انتخاب کند که مقدار C آن کمتر است. اگر مشابه همین استدلال را برای انتخاب سکه v_j توسط نفر اول انجام‌دهیم،آنگاه به این نتیجه می‌رسیم که مقدار $C(i,j)$ به‌صورت زیر قابل تعریف است:

$$C(i,j) = \max \begin{cases} v_i + \min\{C(i+2,j), C(i+1,j-1)\} \\ \\ v_j + \min\{C(i+1,j-1), C(i,j-2)\} \end{cases}$$

حال جواب نهایی مسئله، مقدار $C(1,n)$ است.همینطور چون هر مسئله فقط به زیرمسئله‌های کوچک‌تر از خود ارتباط دارد پس گراف وابستگی این مسئله دور ندارد(DAG است) و این الگوریتم به درستی کار می‌کند.

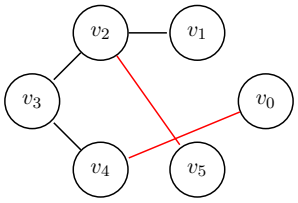
۲ شمارش زیردرخت‌های فراگیر مسطح

در این مسئله گرافی داریم که رأس‌های آن را بر روی یک دایره(یا یک چندصلعی‌منتظم) قرار دارند. حال می‌خواهیم بررسی کنیم که چند زیر درخت فراگیر از این گراف می‌توانیم انتخاب کنیم که مسطح باشد.منظور از مسطح این است که یال‌های آن یکدیگر را قطع نکنند؛ البته با این فرض که یال‌ها را به صورت پاره‌خط رسم کنیم . برای مثال گراف زیر یک گراف مسطح است:



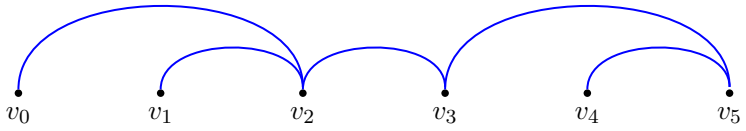
شکل ۱: یک گراف مسطح

اما گراف زیر غیرمسطح است:



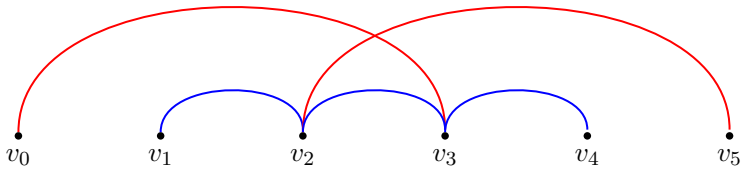
شکل ۲: یک گراف غیر مسطح(برخورد در یال‌های قرمز رنگ)

اما می‌توانیم برای حل مسئله شیوه‌ی کشیدن گراف را تغییر دهیم.بدین صورت که همه‌ی رأس‌ها را بر روی یک خط(مثلاً بر محور x ها قرار دهیم و یال‌ها به صورت نیم‌دایره‌هایی از بالای محور باشند و یکدیگر را قطع نکنند. (می‌توان استدلال کرد دو تعریف فوق برای گراف مسطح معادل هستند اما در اینجا بررسی نمی‌شود) به طور مثال شکل متناظر با گراف مسطح ۱ ، شکل زیر است:



شکل ۳: شکل متناظر با گراف ۱

ولی مثلا شکل زیر،نشان می‌دهد که شکل متناظر با گراف غیر مسطح ۲ ، خاصیت گفته شده را ندارد:



شکل ۴: شکل متناظر با گراف ۲ که در کمان‌های قرمز رنگ تقاطع دارند

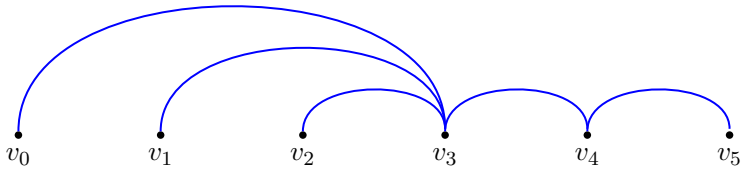
حال تلاش می‌کنیم که مسئله معادل جدید را حل کنیم.برای این مسئله نیز می‌خواهیم که از برنامه‌ریزی پویا کمک بگیریم. برای این‌کار مشابه قبل، باید سعی کنیم که زیرمسئله‌ها را بیابیم و انتخاب‌ها برای مسئله اصلی را به طرز صحیحی، انجام دهیم.

یک زیرمسئله به نظر منطقی این است که $U(i,j)$ را به صورت زیر تعریف کنیم:

$U(i,j) =$ تعداد زیردرخت‌های فراگیر مسطح راس‌های i تا j

برای حل این مسئله در ابتدا باید توجه داشته‌باشیم که همه زیردرخت‌ها را بشماریم و همینطور هیچ زیردرختی را دوبار(یا چندبار) نشماریم.(حالت‌های تکراری را حذف کنیم).

اگر به ۳ توجه کنیم،این‌گونه به نظر می‌رسد که در این شکل،رأس‌های با اندیس ۰ تا ۲ ، از یک منظر از سایر رئوس ایزوله شده اند. یا مثلاً در شکل زیر داریم که رأس‌های بین ۰ تا ۳،حالت ایزوله‌ای نسبت به سایر رئوس دارند و مثلاً نمی‌توان رئوس ۱ یا ۲ را به رئوس بعد از ۳ وصل کرد:



شکل ۵: شکل متناظر با گراف یک گراف مسطح

تلاش می‌کنیم با حالت‌بندی را بر روی این‌که هر زیرمسئله را به زیرمسئله‌های دیگر تقسیم کنیم(مشابه ایده ایزوله بودن تعدادی از رئوس) ،به حل این مسئله بپردازیم. یک ایده این است دورترین رأسی که به رأس اول (رأس با شماره صفر)وصل است را درنظربگیریم و روی این

رأس حالت بندی انجام دهیم(زیرا که رأس‌های بین آن‌ها یک زیردرخت ایزوله هستند). این تقسیم‌بندی گراف را به دو زیردرخت ایزوله‌ی کوچکتر تقسیم‌بندی می‌کند پس می‌تواند انتخاب خوبی برای حل زیرمسئله‌ها باشد.

با استفاده از همین ایده سعی می‌کنیم تا مقدار $U(i,j)$ را محاسبه کنیم. دورترین رأسی که رأس i به آن یال دارد را به عنوان متغیر در نظر می‌گیریم. این رأس میتواند رأس $i+1$ باشد یا رأس $i+2$ یا ...یا j. (واضح است که این حالت بندی، شامل تمامی زیردرخت‌ها می‌شود و حالت‌ها با یکدیگر هم‌پوشانی ندارند) . اگر دورترین رأسی که رأس i به آن یال دارد رأس k ام باشد (یعنی رأس i به هیچ رأس بعد از k یال ندارد.)تعداد حالت‌های چنین درختی برابر است با تعداد حالت‌هایی که رأس k تا j تشکیل یک درخت فراگیر مسطح بدهند (که طبق تعریف همان $U(k,j)$ است) ضرب‌در تعداد حالت‌هایی که رأس‌های i تا k تشکیل چنین درختی بدهند. اما این تعداد برابر با $U(i,k)$ نیست ؛ زیرا طبق تعریف، مقدار $U(i,k)$ ، برابر با تعداد کل زیردرخت‌هایی است که می‌توان با استفاده از رئوس i تا k ساخت و در این زیردرخت‌ها الزاما یال ik وجود ندارد ولی در این حالت ما باید حتما یال i به k باید وجود داشته‌باشد. اگر تعداد چنین زیردرخت‌هایی (زیر درخت‌هایی از رئوس i تا k که حتما یال(کمان) ik در آن وجود دارد)را $C(i,k)$ بنامیم آنگاه داریم :

$$U(i,j) = \sum_{k=i+1}^{j} C(i,k) \times U(k,j)\delta(i,k)$$

(در اینجا متغیر $\delta(i,k)$ نیز اضافه شده‌است که در ادامه به توضیح آن می‌پردازیم)

برای محاسبه این مقدار، باید بتوانیم به طور بازگشتی $C(i,j)$ را نیز به‌دست آوریم.برای مقدار $C(i,j)$ داریم که:

$C(i,j)$ = تعداد زیردرخت‌های فراگیرمسطح رأس‌های i تا j که در آن‌ها رأس i به رأس j متصل باشند.

اگر یال i به j را در نظر نگیریم ، طبق تعریف درخت در گراف،می‌دانیم که باید دو مؤلفه تشکیل شود و به علاوه i و j در دو مؤلفه‌ی جداگانه قرار بگیرند. می‌دانیم تعدادی از رأس‌ها در مولفه‌ی همبندی رأس i و بقیه‌ی آن‌ها در مولفه‌ی همبندی j قرار دارند. حال ادعا می‌کنیم که اگر k آخرین رأسی باشد که در مولفه‌ی همبندی i قرار دارد تمامی رأس‌های قبل از آن نیز در همین مولفه هستند و تمام رأس‌های بعد از آن در مولفه‌ی j هستند. می‌توان این ادعا را با مسطح بودن گراف به سادگی نشان داد؛زیرا اگر مسیر از i به k در این درخت را در نظر بگیرید،هر رأسی که در این مسیر وجود دارد که در مؤلفه‌ی i است و هر رأسی که نیست، زیر یکی از کمان های بین دو رأس افتاده است و بنابراین این رئوس باید در مؤلفه‌ی مربوط به j باشد که با مسطح بودن گراف در تناقض است(زیرا به هیچ طریقی نمی‌تواند به بیرون از آن کمانی که در آن قرار دارد، یال داشته باشد). پس هر رأس بین i و k ، باید در مؤلفه‌ی مربوط به رأس i باشد.

بنابراین با برداشتن یال i و j گراف تبدیل به دو مولفه‌ی مسطح جدا شده‌است. که هرکدام از این مولفه‌ها نیز یک درخت فراگیر مسطح هستند و تعداد حالت‌های آن‌ها از $U(k,j)$ پیروی می‌کند. بنابرین داریم :

$$C(i,j) = \sum_{k=i}^{j-1} U(i,k) \times U(k+1,j)$$

نکته ای که وجود دارد این است که در رابطه ۲ باید یال i به k وجود داشته باشد تا بتوانیم این محاسبات را انجام دهیم وگرنه باید برای آن k خاص،این مقدار را برابر با صفر در نظر بگیریم. بدین منظور مقدار $\delta(i,k)$ را در رابطه اضافه می‌کنیم و بدین صورت تعریف می‌شود هنگامی که رأس i به k یال دارد برابر با یک و در غیر این صورت صفر است.در واقع :

$$\delta(i,k) = \begin{cases} 1 & i \text{ به } k \text{ یال داشته باشد} \\ 0 & \text{در غیر این‌صورت} \end{cases}$$

واضح است که زیرمسئله‌های تعریف شده همیشه کوچک‌تر از مسئله‌ی اصلی هستند و گراف وابستگی آن دور ندارد. اما بهتر است که حالت‌های مرزی را نیز بررسی کنیم.

۳ مسئله فروشنده دوره‌گرد

مسئله فروشنده‌ی دوره‌گرد[۲] بدین صورت است که یک فروشنده‌ی دوره‌گرد قصد دارد از تمام شهرهای کشور خود عبور کند و به مبدأ خود بازگردد به شرط آن که از هر شهر دقیقا یک‌بار عبورکند و کمترین مسافت ممکن را طی‌کند.

ورودی : گراف وزن‌دار $G = (V, E)$ با تابع وزن $C : E \to \mathbb{Q}^+$

خروجی : دوری با طول n با وزن کمینه(دور همیلتونی با وزن کمینه)

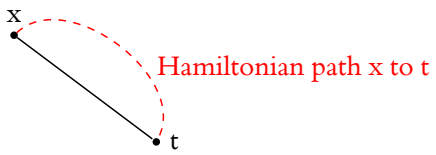
پاسخ : این مسئله راه‌حل چند جمله‌ای ندارد و از جمله مسائل NP-Hard است. یک الگوریتم بدیهی این است که می‌توان تمام دورهای موجود را چک کرده و کمینه وزن را انتخاب کنیم. این الگوریتم از مرتبه زمانی $O(n!)$ است. اما ما به دنبال یک الگوریتم نمایی که مرتبه زمانی آن بهتر باشد هستیم. در واقع هدف پیدا کردن الگوریتمی از مرتبه زمانی $O(n^2 \times 2^n)$ است.برای این کار از برنامه‌ریزی پویا استفاده خواهیم‌کرد . پس باید رابطه‌ی زیرمسئله‌های آن را پیدا کنیم. یک ایده برای حل این سؤال این است که یک رأس خاص(مثلا رأس با اندیس ۱ که آن را x می‌نامیم) را فیکس کنیم و بر روی این کار کنیم که در این دور همیلتونی بهینه، قبل و بعد از آن رئوسی هستند و تلاش کنیم که این دو رأس را بیابیم.حال مثلاً اگر فرض کنیم که می‌دانیم رئوس قبل و بعد از آن چیست(مثلاً u, v) اکنون زیر مسئله ما چیست! بررسی زیرمسئله برای دورها کار راحتی نیست اما می‌توانیم مسئله را طوری تغییر دهیم که به‌جای دور همیلتونی مسیر همیلتونی داشته‌باشیم تا تبدیل آن به مسئله‌های کوچک‌تر ساده‌تر باشد. برای دور همیلتونی اگر رأس x با رأسی مانند t همسایه باشد آنگاه دور ما دارای یک یال از x به t و علاوه بر آن دارای یک مسیر همیلتونی از x به t نیز هست که مجموع آن‌ها کمینه شده‌است. حال اگر بتوانیم در این مسیر همیلتونی رأس بعد از x را بیابیم، آنگاه می‌توانیم که زیر مسئله‌ها را نیز تعریف کنیم و آن‌ها را حل کنیم.

پس یک سر مسیر را فیکس می‌کنیم(رأس x).حال مقدار $C(S, v)$ را به صورت زیر تعریف می‌کنیم: ($S \subseteq V$)

$$C(S, v) = S \text{ هزینه‌ی کوتاه‌ترین مسیر از } x \text{ به } v \text{ در رئوس}$$

حال اگر به مسئله اصلی توجه کنیم، در دور همیلتونی کمینه ، رأس مجاورِ رأس x رأسی مانند t است . در واقع این دور شامل دو مسیر است که یکی از آن‌ها یال xt است و دیگری یک مسیر همیلتونی از x به t است که جمع وزن این دو مجموع کمینه شده است. پس با توجه به تعاریف و توضیحات، پاسخ مسئله‌اصلی ما به صورت زیر خواهد بود:(که در آن V مجموعه همه رئوس گراف است و $w(xt)$ وزن یال xt است).

$$\min_{t \neq x} \{C(V, t) + w(xt)\}$$



[۲] Traveling Salesman Problem

در رابطه بالا اگر یال xt در گراف وجود نداشت، $w(xt) = \infty$ قرار می‌دهیم. حال اگر زیر مسئله‌های آن را بتوانیم حل کنیم جواب مسئله‌ی اصلی را نیز می‌توانیم پیدا کنیم. پس برای این مسئله‌ی جدید باید به دنبال زیرمسئله‌های مناسب بگردیم. حال اگر رأس یکی مانده به آخر مسئله(مسیر همیلتونی کمینه از مجموعه S از x به t)را v بنامیم هزینه‌ی این مسیر برابر است با هزینه‌ی مسیر x به v به علاوه‌ی وزن یال vt . پس داریم :

$$C(S,t) = \min_{v}\{C(S-t,v) + w(vt)\}$$

حال کافی است بررسی کنیم که زمان اجرای این الگوریتم از مرتبه زمانی مورد نظر ما است : می‌دانیم تعداد زیرمسئله‌ها از $O(n \times 2^n)$ است و زمان مصرف‌شده برای هر زیرمسئله $O(n)$ است. پس زمان کلی اجرای الگورتیم از $O(n^2 \times 2^n)$ خواهدبود.

نکته‌ای که باید به آن توجه کرد، این است که زیرمسئله‌ها را چگونه نمایش دهیم تا بهینه باشد.برای این کار می‌توانیم از بردار مشخصه استفاده کنیم. یعنی مثلا اگر $S = \{1,3,4\}$ آنگاه می‌توانیم به این مجموعه بردار بیتی 1101 متناظر کنیم . حال می‌توانیم با عملگر های منطقی مانند xor یک عضو از این مجموعه را حذف کنیم(بردار متناظر با $S = \{1,3,4\}$ که متناظر با 1101 را با 1000 xor کنیم تا $S - \{4\} = \{1,3\}$ با بردار مشخصه 0101 حاصل شود). با این روش، حافظه مصرفی و زمان اجرای الگوریتم نیز به طرز کارآیی کاهش می‌یابد.

۴ فاصله‌ی ویرایش

مسئله‌ی فاصله‌ی ویرایش[۳]به دلیل کمبود زمان کامل بررسی نخواهد شد توضیحات اجمالی در رابطه با آن داده می‌شود. در جلسات قبل مسئله‌ی هم‌ترازی دنباله‌ها[۴]را بررسی کرده‌ایم. در برخی مسائل حافظه‌ی مصرفی برای ما بسیار مهم خواهد بود. پس می‌خواهیم سعی کنیم که با ثابت نگه داشتن زمان ، حافظه‌ی اصلی را کم کنیم. ایده‌ی کلی آن به این صورت است که چون محاسبه‌ی هر ستون جدول فقط به ستون قبلی آن و چند خانه‌ی همسایه‌ی هر خانه بستگی دارد برای بدست آوردن مقدار مورد نظر دو ستون current و previous کافی است و می‌توان مقدار حافظه را به $O(max\{m,n\})$ کاهش داد. برای حل خود مسئله با حافظه‌ی کمتر هم در اسلایدهای قرار گرفته در سایت درس ، الگوریتم هیرشبرگ[۵] به کمک تقسیم و حل با حافظه‌ی $O(m+n)$ مسئله را حل می‌کند.

مراجع

[1] Cormen, Thomas H., et al. *Introduction to Algorithms*. 3rd ed., MIT Press, 2009.

[2] CMU-DP

[3] MIT-DP

[3]Edit Distance
[4]Sequence Alignment
[5]Hirschberg

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۱۰: الگوریتم پیدا کردن کوتاه‌ترین مسیر

نگارنده: آیسان نیشابوری

در این جلسه مسئله‌ی کوتاه‌ترین مسیر و راه‌های حل آن را بررسی می‌کنیم.

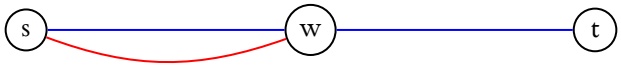
۱ مسئله‌ی کوتاه‌ترین مسیر:

داده‌های این الگوریتم عبارتند از:

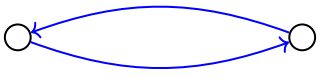
ـ **ورودی** : گراف جهتدار $D = (V, E)$ و تابع وزن $w : E \to \mathbb{Q}$ [می‌توانیم رأس $s \in V$ و $t \in V$ را به عنوان ابتدا و انتهای مسیر بدهیم].

ـ **خروجی** : [طول] کوتاه‌ترین مسیر از s به t یا [طول] کوتاه‌ترین مسیر از s به همه‌ی رأس‌ها یا [طول] کوتاه‌ترین مسیر بین هر $u, v \in V$

نکته ۱: پیدا کردن کوتاه‌ترین مسیر از s به t از چند زیر مسئله‌ی بهینه تشکیل شده‌است. مثلا برای این که مسیری کوتاه‌ترین مسیر از s به t باشد باید شامل کوتاه‌ترین مسیر از s به w هم باشد وگرنه کوتاه‌ترین مسیر از s به w را جایگزین می‌کنیم و مسیرمان کوتاه‌تر می‌شود.



نکته ۲: برای هر گراف بدون جهت هم یک گراف جهتدار تعریف می‌کنیم که به ازای هر یال در گراف جهتدارمان یک یال رفت و یک یال برگشت می‌گذاریم (البته جاهایی وجود دارد که این روش کار نمی‌کند).



نکته ۳: می‌توانیم به جای تابع وزن، برای یال‌ها تابع طول یا هزینه تعریف کنیم.

تعریف: گشت[1] ، یک دنباله از رأس‌ها مثل $s = v_0v_1v_2..v_k = t$ است که $v_iv_{i+1} \in E$ باشد.

تعریف: اگر گشتی رأس تکراری نداشته باشد به آن مسیر می‌گوییم.

ما معمولا در الگوریتم‌هایمان گشت پیدا میکنیم. پس سؤالی که پیش میاید این است که در چه صورتی کوتاه‌ترین گشت، مسیر است؟

[1] Walk

اگر گشتمان دور داشته باشد، حتما جمع وزن یال‌های درون دور منفی است زیرا اگر مثبت باشد می‌توانیم آن را حذف کنیم و وزن کم می‌شود و اگر صفر باشد، می‌توانیم بدون کم شدن وزن آن را حذف کنیم.

پس در صورتی که در گراف دور منفی نداشته باشیم کوتاه‌ترین گشت، کوتاه ترین مسیر هم هست. اگر در گرافی دور منفی داشته باشیم اندازه‌ی کوتاه‌ترین مسیر آن $-\infty$ می‌شود زیرا می‌توانیم بی‌نهایت بار دور منفی را طی کنیم و وزن کم‌تر شود.

در حالت خاص‌تر می‌توانیم مسئله را برای گراف‌هایی با تابع وزن مثبت تعریف کنیم.

در حالت کلی مسئله‌ی کوتاه‌ترین مسیر، یک مسئله‌ی NP-Hard می‌باشد.

تعریف: برش[۲] ، افرازی از مجموعه‌ی رئوس گرافمان به دو زیر مجموعه است.

تعریف: st برش را برشی تعریف می‌کنیم که s در یک مجموعه و t در مجموعه‌ی دیگر باشد.

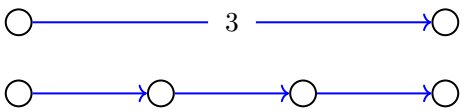
قضیه ۱. *یک قضیه.* [*برای گراف‌های بدون وزن*] *طول کوتاه‌ترین مسیر از s به t، برابر بیشینه‌ی تعداد st برش‌های مجزاست (منظورمان از مجزا برای مجموعه‌ی یال‌های بین دو مجموعه‌ی برش‌هاست).*

اثبات. (تمرین). □

تمرین: قضیه‌ی بالا را برای گراف‌های وزندار تعمیم دهید.

۲ مسئله‌ی کوتاه‌ترین مسیر برای گراف‌هایی با وزن مثبت و صحیح:

راه حل ۱: هر یال با وزن w را با w تا یال با وزن یک جایگذاری می‌کنیم و روی گراف جدید BFS می‌زنیم.



ـ **زمان اجرا:** زمان اجرای این الگوریتم $O(V'+E')$ می‌باشد که V' و E' تعداد رأس‌ها و یال‌های گراف جدیدمان هستند. هر یال با وزن w باعث اضافه شدن $w-1$ رأس و $w-1$ یال می‌شود. پس اگر L را بیش‌ترین وزن، بین وزن‌های یال‌هایمان بگیریم، داریم

$$O(V'+E') = O(V + \sum_{e\in E} w(e)) = O(V+EL)$$

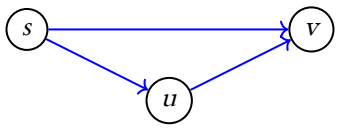
که V و E تعداد رأس‌ها و یال‌های گراف قبلی‌مان هستند. زمان $O(V+EL)$، چندجمله‌ای نیست اما شبه چندجمله‌ای است.

راه حل ۲ (الگوریتم دایکسترا [۳])

قضیه ۲. *نامساوی مثلث: اگر کوتاه‌ترین مسیر بین هر دو رأس را با δ نشان دهیم داریم*

$$\delta(s,v) \le \delta(s,u) + \delta(u,v)$$

[2] Cut
[3] Dijkstra



اثبات. یکی از مسیرهای s به v این است که از s به u برویم و سپس به v. پس این مسیر بزرگ‌تر مساوی کوتاه‌ترین مسیر است. □

پس اگر $w(u,v)$ را وزن بین رأس u و v تعریف کنیم (که اگر یالی وجود نداشته باشد برابر $+\infty$ است) داریم

$$\delta(s,v) \leq \delta(s,u) + w(u,v)$$

در این الگوریتم همواره برای هر رأس v، $d(v)$ را نگه می‌داریم که یک کران بالا برای $\delta(s,v)$ می‌باشد.

تابع $relax$ را به صورت زیر تعریف می‌کنیم.

```
RELAX(e = uv)
1  if d(v) > d(u) + w(uv)
2        d(v) = d(u) + w(uv)
```

الگوریتم کلی:

```
1  d(s) = 0
2  ∀v ∈ V(v ≠ s) : d(v) = ∞
3  while true
4        Choose e ∈ E
5        relax(e)
```

$relax$ کردن هیچ وقت ضرر ندارد یعنی همیشه $d(v) \geq \delta(s,v)$ باقی می‌ماند.

اثبات. اولین جایی را در نظر بگیرید که این قانون نقض می‌شود یعنی یال $e = uv$ را $relax$ می‌کنیم و قانون نقض می‌شود. پس داریم

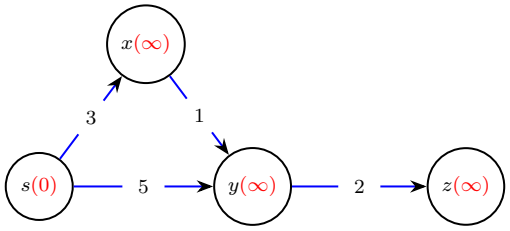
$$d(v) = d(u) + w(u,v) \geq \delta(s,u) + w(u,v) \geq \delta(s,v)$$

که این با فرض ما در تناقض است. پس امکان ندارد $d(v)$ کم تر از $\delta(s,v)$ شده باشد. □

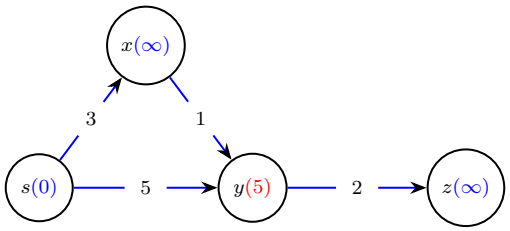
تمرین: اگر به حالتی برسیم که با $relax$ کردن هیچ کدام از یال‌های گراف تغییری در مقادیر d ایجاد نشود. ثابت کنید برای هر رأس v، $d(v) = \delta(s,v)$.

مثال: در این مثال d ابتدایی هر رأس جلوی آن و در پرانتز نوشته شده.

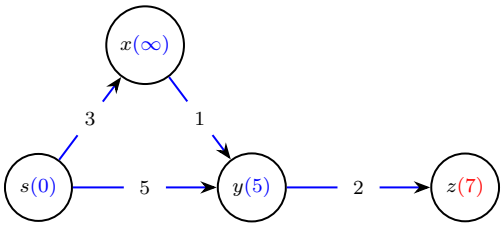
ابتدا این گراف را داریم



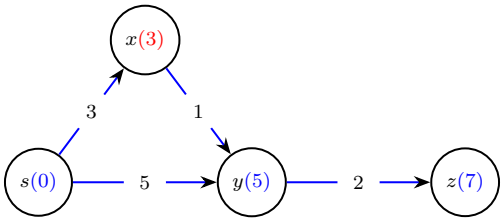
یال sy را ریلکس می‌کنیم ←



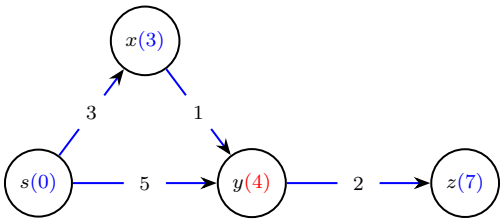
یال yz را ریلکس می‌کنیم ←



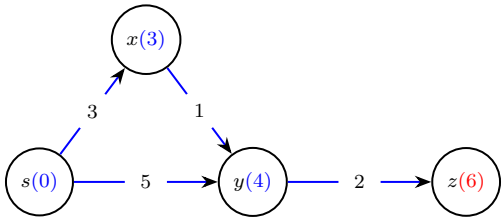
یال sx را ریلکس می‌کنیم ←



یال xy را ریلکس می‌کنیم ←



یال zy را ریلکس می‌کنیم ←



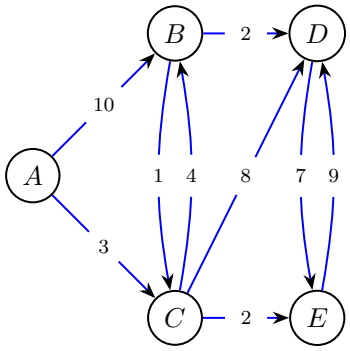
⊠

برای الگوریتم دایکسترا یک مجموعه از رأس‌های V که $\delta(s,v)$ برای آن‌ها مشخص شده نگه می‌داریم و آن را S می‌نامیم.

در هر مرحله رأسی که درون S نیست و تخمین آن کمینه است را به S اضافه می‌کنیم و همه‌ی یال‌های خروجی آن را $relax$ می‌کنیم.

```
1  d(s) = 0
2  ∀v ≠ s : d(v) = ∞
3  S = ∅
4  while S ≠ V
5        Choose v ∈ V \ S minimizing d(v)
6        S = S ∪ {v}
7        for each e = vw ∈ Adj[v]
8              relax(vw)
```

مثال: گراف زیر را داریم و الگوریتم دایکسترا را روی آن اجرا می‌کنیم. مقدار d برای هر کدام از رأس‌هایی که درون S نیستند در هر مرحله مشخص شده است و در هر مرحله هم رأسی که انتخاب می‌شود با قرمز مشخص شده.



A	B	C	D	E
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

⊠

اثبات درستی: با استقرا نشان می‌دهیم در همه‌ی مراحل اجرای الگوریتم $\forall v \in S : d(v) = \delta(s, v)$. فرض کنید این گزاره تا قبل از مرحله‌ی فعلی برقرار باشد. رأس انتخاب شده در مرحله‌ی فعلی را v می‌نامیم. فرض کنید $d(v) > \delta(s, v)$ باشد.

یک کوتاه ترین مسیر از s به v در نظر می‌گیریم به صورت $s = v_1 v_2 ... v_k = v$. فرض کنید در این مسیر i کوچک‌ترین اندیسی باشد که $v_i \notin S$ داریم

$$d(v_i) \leq d(v_{i-1}) + w(v_{i-1}, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i)$$

از طرفی هم چون این مسیر زیر مسیری در کوتاه‌ترین مسیر از s به v است. پس مسیر از s به v_i هم کوتاه‌ترین مسیر است پس

$$\delta(s, v_{i-1}) + w(v_{i-1}, v_i) = \delta(s, v_i) \leq \delta(s, v) < d(v) \Rightarrow d(v_i) < d(v)$$

و این تناقض است زیرا v را رأسی گرفتیم که تخمین کمینه را دارد.

تمرین: گرافی پیدا کنید که الگوریتم دایکسترا برای آن نادرست باشد.

حال به نکاتی برای پیاده سازی این الگوریتم اشاره می‌کنیم.

ـ پیاده‌سازی معمولی: $d(v)$ همه‌ی رأس‌ها را در یک آرایه نگه می‌داریم و در هر مرحله همه ی $d(v)$ ها را چک می‌کنیم و مینیمم را پیدا می‌کنیم. پس خط ۵ الگوریتممان $O(V)$ طول می‌کشد و V بار هم اجرا می‌شود پس کل اجرا برای این قسمت $O(V^2)$ وقت می‌گیرد. برای $relax$ کردن هم هر یال حداکثر یک بار از $O(1)$ $relax$ می‌شود، پس این قسمت الگوریتم از $O(E)$ است و کل الگوریتم از $O(V^2 + E)$ می‌باشد که همان $O(V^2)$ است زیرا حداکثر $\binom{n}{2}$ یال داریم که از $O(V^2)$ می‌باشد.

ـ پیاده‌سازی با هرم: اگر d ها را با هرم ذخیره کنیم خط ۵ در هر مرحله $O(\log V)$ طول می‌کشد چون حذف کردن عدد کمینه از هرم در زمان $O(\log V)$ انجام می‌شود و این کار V مرحله انجام می‌شود، پس در کل خط ۵ $O(V \log V)$ زمان می‌گیرد. همچنین وقتی که یالی را $relax$ می‌کنیم باید $\log V$ زمان بپردازیم تا هرممان را مرتب کنیم و هر یال حداکثر یک بار $relax$ می‌شود و پس در کل برای $relax$ کردن‌ها هزینه‌ی $O(E \log V)$ می‌پردازیم. در نهایت هزینه‌ی کل این پیاده‌سازی $O(V \log V + E \log V)$ می‌شود.

ـ پیاده‌سازی با هرم فیبوناچی: اگر d هایمان را در این هرم نگه داریم زمان اجرای الگوریتممان در زمان $O(V \log V + E)$ انجام می‌شود.

پیاده سازی با الگوریتم Dial : فرض کنید حداکثر وزن یال‌ها L باشد. پس d هیچ کدام از رأس‌ها از nL بیش‌تر نمی‌شود که n برابر تعداد رأس‌هاست. یک آرایه $nL + 2$ تایی می‌گیریم که خانه‌ی اول و آخر به ترتیب نماینده‌ی $+\infty$ و $-\infty$ هستند و بقیه‌ی خانه‌ها به ترتیب نماینده‌ی اعدادی یک تا nL. هر خانه ی این آرایه لیست پیوندی[۴]ای از رأس‌هاست. برای مثال خانه‌ی i ام لیستی از رأس‌هایی است که d شان برابر i است.

در هر مرحله یک اشاره‌گر $current$ نگه می‌داریم که از آنجا رأس بعدی S را انتخاب می‌کنیم. این اشاره‌گر هیچ وقت به سمت اعداد کمتر حرکت نمی‌کند زیرا هیچ وقت رأس‌هایی که تا الان دیده‌ایم d شان از d مینیمم فعلی‌مان کم تر نمی‌شود.

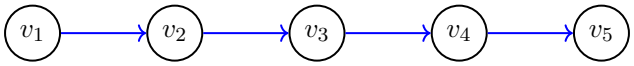
پس اشاره‌گرمان فقط به سمت راست (سمت اعداد بزرگ‌تر) حرکت دارد و خط ۵ از $O(1)$ انجام می‌شود. همچنین اگر مکان رأس‌هایمان را بدانیم $relax$ هم از $O(1)$ انجام می‌شو دو اشاره‌گرمان هم nL بار به سمت راست حرکت می‌کند پس هزینه‌ی کل الگوریتم‌مان $O(VL + E)$ می‌باشد.

تمرین: زمان الگوریتم Dial را به $O(E \log(L))$ برسانید.

[4] Linked list

۳ مسئله‌ی کوتاه‌ترین مسیر در حالت کلی:

اگر وزن‌های غیر محدود داشته باشیم و به ترتیب خوبی یال‌ها را $relax$ کنیم باز می‌توانیم به کوتاه‌ترین مسیر برسیم. مثلا در گراف زیر اگر به ترتیب یال‌های $v_1v_1v_2, ..., v_4v_5$ را ریلکس کنیم باز به کوتاه‌ترین مسیر می‌رسیم.



اما برای گرف های خاص میتوان مسئله را راحت‌تر حل کرد. برای مثال برای گراف‌های جهتدار بدون دور [۵] می‌توان در زمان $O(E)$ کوتاه‌ترین مسیر را پیدا کرد به این صورت که رأس‌ها را به صورت توپولوژیکی مرتب می‌کنیم و فرض می‌کنیم به صورت $v_1, v_2, ..., v_n$ مرتب می‌شوند به طوری که هر رأس تنها بتواند به رأس‌های سمت راست خود یال داشته باشد. آنگاه الگوریتم زیر را برای پیدا کردن کوتاه ترین مسیر اجرا می‌کنیم.

```
1  for i = 1 to V − 1
2      for each e = v_i w
3          relax(e)
```

همه‌ی مسیرها از چپ به راست حرکت می‌کنند پس یه ترتیب خوبی $relax$ می‌شوند.

الگوریتم بلمن فورد [۶] :

برای گراف‌های کلی‌تر که دور منفی نداشته باشند می‌توان الگوریتم زیر را اجرا کرد.

```
1  for i = 1 to V − 1
2      for each e ∈ E
3          relax(e)
```

اگر کل یال‌ها را $relax$ کنیم برای هر کوتاه‌ترین مسیر از s به t اولین یال آن مسیر هر $relax$ میشود. حال اگر این پروسه را به تعداد $V-1$ بار انجام دهیم در مرحله‌ی i ام، یال i ام کوتاه‌ترین مسیر به طور درستی $relax$ می‌شود و طول مسیر حداکثر $V-1$ یال است پس اگر این کار را $V-1$ بار انجام دهیم کل مسیر به طریقه‌ی خوبی $relax$ می‌شود،

زمان اجرا: زمان اجرای این الگوریتم $O(VE)$ است زیرا در هر مرحله همه‌ی یال‌ها $relax$ می‌شوند و V مرحله داریم.

مراجع

[۱] فیلم جلسه‌ی دهام آنالیز الگوریتم.

[2] Cormen, Thomas H., et al. *Introduction to Algorithms*. 3rd ed., MIT Press, 2009, pp. 643-678.

[5] Directed Acyclic Graph (DAG)
[6] Bellman–Ford

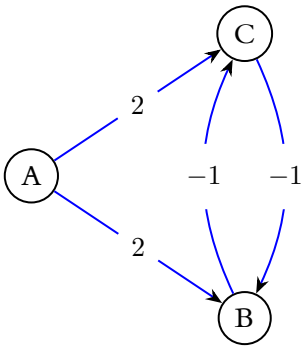
آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۱۱: مسئله‌ی کوتاهترین مسیر ۲ نگارنده: بردیا آریان فرد

جلسه‌ی قبل مبحث کوتاهترین مسیر را شروع کردیم. در این جلسه قصد داریم به این مبحث از دیدگاه برنامه‌ریزی پویا نگاه کنیم.
در جلسه قبل دیدیم که در مسئله‌ی کوتاهترین گشت نامساوی مثلثی برقرار است، امّا این قضیه درباره‌ی کوتاهترین مسیر برقرار نیست (این مسئله و مسئله‌ی بلندترین مسیر هیچکدام به علت ذات زیرمسئله‌ای برنامه‌ریزی پویا، با این روش حل نمی‌شوند!). مانند مثال زیر:



۱ روش بلمن_فورد با دیدگاه برنامه‌ریزی پویا

فرض کنید d_t برابر است با طول کوتاهترین مسیر از s به t . رابطه‌ی زیر در میان زیرمسئله‌ها برقرار است:

$$d_t = \min_x \{d_x + w(x,t)\}$$

گراف وابستگی‌ها باید به نحوی تعریف شده‌باشد که سمت راست به معنایی واقعاً کوچکتر از سمت چپ باشد. پس d_t^k را تعریف می‌کنیم «طول کوتاهترین مسیر از s به t که حداکثر k یال دارد.»
که برای این زیرمسائل نیز می‌توان گفت:

$$d_t^k = \min\left\{d_t^{k-1}, \min_x\{d_x^{k-1} + w(x,t)\}\right\}$$

که باید برای تمام رئوس مجاور، این مینیمم را حساب کنیم (یال‌های خروجی را می‌توان با زمان خطی به یال‌های ورودی تبدیل کرد.) امّا چون باید این کار را برای تمامی یال‌های تمامی رئوس تکرار کرد، می‌توان به جای این‌کار یک بار این عمل را برای تمامی یال‌ها انجام داد:

```
1  d_s^0 = 0, d_t^0 = ∞ ∀t ≠ s
2  for k = 1 to n − 1
3      d_v^k = d_v^{k−1}
4      for (u, v) ∈ E
5          if d_u^{k−1} + w(u, v) < d_v^k
6              d_v^k = d_u^{k−1} + w(u, v)
```

در واقع، با این کار یال‌ها را ریلکس می‌کنیم و از آنجا که ریلکس کردن اضافه ضرر ندارد، می‌توان اندیس‌های بالا را حذف کرد. که این، در واقع همان الگوریتم بلمن_فورد [1] است.
دیدیم که در صورت عدم وجود دور منفی، کوتاهترین گشت، برابر کوتاهترین مسیر است، پس بلمن_فورد به درستی کار می‌کند ولی در غیر این صورت نه. حال می‌خواهیم بفهمیم که آیا گراف دور منفی دارد یا نه.

قضیه ۱. *گراف دور منفی ندارد، اگر و تنها اگر* $d^n = d^{n-1}$
نکته: وقتی برای d *اندیس نمی‌گذاریم، منظور بردار تمامی* d_i*ها است.*

اثبات. اگر دور منفی وجود نداشته‌باشد، نباید بار $n-$ ام اتفاقی بیفتد چرا که تمامی مسیرهای کمینه طول حداکثر $n-1$ دارند و تا قبل از این مرحله ذخیره شده‌اند. برای اثبات طرف دیگر، از برهان خلف استفاده می‌کنیم. فرض کنید یک دور منفی با رئوس $v_1, v_2, ..., v_k$ داریم ولی در مرحله‌ی n فاصله‌ی رأسی تغییر نمی‌کند. نتیجه می‌گیریم:

$$\left.\begin{array}{l} d_{v_2} \le d_{v_1} + w(v_1, v_2) \\ d_{v_3} \le d_{v_2} + w(v_2, v_3) \\ \cdots \\ d_{v_1} \le d_{v_k} + w(v_k, v_1) \end{array}\right\} \implies \sum_{i=1}^{k} d_{v_i} \le \sum_{i=1}^{k} d_{v_i} + \sum_{i=1}^{k} w(v_i v_{i-1}) \implies 0 \le \sum_{i=1}^{k} w(v_i v_{i-1})$$

که از تناقض فوق، حکم نتیجه می‌شود. □

نکته: تمامی این‌ها با این فرض هستند که دور منفی از s قابل دسترسی باشند.
تمرین: روشی برای چاپ کردن رئوس یک دور منفی(در صورت وجود) ارائه دهید.

۲ کوتاهترین مسیر بین تمام زوج رأس‌ها

در مواقعی پیدا کردن کوتاهترین مسیر بین تمام زوج رأس‌ها بسیار کاربردی است، مثلاً اگر در یک نقشه تعداد زیادی درخواست کوتاهترین مسیر بین دو راس داشته‌باشیم ممکن است بهینه نباشد که هر بار کوتاهترین مسیر را پیدا کنیم و بهتر باشد که یک بار کوتاهترین مسیر همه را پیدا کرده و سپس فقط جواب از روی آن‌ها برگردانیم.

راه اوّلیه ۱: برای گراف‌های بدون وزن
برای حل این حالت می‌توانیم از هر یک از رئوس، الگوریتم جستجوی سطح_اوّل [2] را اجرا کنیم که این الگوریتم به وضوح از پیچیدگی زمانی $O(VE)$ است.

راه اوّلیه ۲: برای گراف‌هایی با وزن مثبت
برای حل این حالت نیز می‌توان از تمامی رئوس، الگوریتم دایکسترا [3] را اجرا کرد. اگر دایکسترا را با هرم فیبوناچی [4] پیاده‌سازی کنیم، پیچیدگی زمانی این الگوریتم $O(V^2 logV + VE)$ است.

راه اوّلیه ۳: برای حالت کلّی
برای این حالت، از هر یک از رئوس الگوریتم بلمن_فورد را اجرا می‌کنیم. این الگوریتم از پیچیدگی زمانی $O(V^2E)$ است. این الگوریتم، از دو راه قبلی زمان بدتری دارد، امّا در سه راه بعدی، آن را ارتقا می‌دهیم.
نکته: گاهی بهتر است فرض کنیم گراف را به شکل ماتریس مجاورت داریم.

[1] Bellman-Ford
[2] BFS
[3] Dijkstra
[4] Fibonacci Heap

راه ۱: $O(V^3 logV)$

فرض کنید $d^k_{u,v}$ برابر است با کوتاهترین مسیر بین u و v با حداکثر k یال. می‌توان گفت:

$$d^k_{u,v} = \min\left\{d^{k-1}_{u,v}, \min_{x\to v}\{d^{k-1}_{u,x} + w(x,v)\}\right\}$$

محاسبه کوتاهترین مسیر از روی این تعریف در واقع همان n بار اجرای بلمن_فورد است، چرا که هر رأس یک بار به عنوان مبدأ و یک بار به عنوان مقصد انتخاب می‌شود. این‌جا هم می‌توان k را حذف کرد:

```
1  d_{u,u} = 0(∀u ∈ V), d_{u,v} = ∞ (∀u ≠ v)
2  for k = 1 to n
3      for v ∈ V
4          for u ∈ V
5              for x ∈ V
6                  if d_{u,x} + w(x,v) < d_{u,v}
7                      d_{u,v} = d_{u,x} + w(x,v)
```

محاسباتی که در بالا انجام دادیم، شبیه ضرب ماتریس بود با این تفاوت که در رابطه‌ی

$$C_{i,j} = \sum A_{i,k}B_{k,j}$$

به جای جمع، کمینه حساب می‌کنیم و به جای ضرب، جمع می‌کنیم. اگر این نماد ضرب را o نام‌گذاری کنیم، می‌توان گفت:

$$D^k = D^{k-1} \text{ o } W$$

که ماتریس D ، حاوی فاصله‌ی دوبه‌دوی رئوس است.

پس آن را می‌توان مانند الگوریتم سریعی که برای به توان رساندن (از روی D^{2^i} ها) داشتیم، با $O(logV)$ بار عمل ضرب انجام‌داد. که این پیچیدگی زمانی کل الگوریتم را به $O(V^3 logV)$ می‌رساند.

این طرز فکر، چند کاربرد دیگر هم دارد، مثلاً، فرض کنید می‌خواهیم بفهمیم که آیا بین دو رأس مسیر هست یا نه. برای این کار کافی‌است مثل همین روش عمل کرده و در انتها، به جای بی‌نهایت‌ها صفر و به جای باقی درایه‌ها یک بگذاریم. یا برای مثال فرض کنید یک گراف داریم که رئوس آن، رشته‌های به طول n از ۰ و ۱ هستند و تابعی به نام $e(v,u)$ داریم که با گرفتن دو رأس v و u به ما می‌گوید که آیا بین این دو رأس یال وجود دارد یا نه. حال فرض کنید می‌خواهیم بفهمیم که آیا بین دو رأس مسیر هست یا نه. فرض کنید در این‌جا فقط مشکل حافظه داریم و زمان برای ما مطرح نیست. اگر بخواهیم از روش‌هایی مانند جستجوی عمق_اول [۵] استفاده کنیم و یک آرایه‌ی $mark$ داشته‌باشیم که در آن ذخیره کنیم که آیا یک رأس را دیده‌ایم یا نه، حافظه‌ی زیادی مصرف می‌شود.

تمرین: با این روش، این مسئله را با حافظه‌ی چندجمله‌ای حل کنید.

تمرین: با این روش، مسئله‌ی فروشنده‌ی دوره‌گرد [۶] را با حافظه‌ی بهتر حل کنید.

راه ۲: فلوید_وارشال[۷] $O(V^3)$

تعریف $d^k_{u,v}$ را تغییر می‌دهیم. فرض کنید $d^k_{u,v}$ برابر است با کوتاهترین مسیر از u به v که فقط از رئوس با برچسب $1, 2, 3, ..., k$ در مسیرش استفاده می‌کند. برای محاسبه‌ی آن، می‌توان گفت:

$$d^k_{u,v} = \min\{d^{k-1}_{u,v}, d^{k-1}_{u,k} + d^{k-1}_{k,v}\}$$

[5] DFS

[6] Travelling Salesman Problem

[7] Floyd-Warshall

که در این تعریف، بر خلاف تعریف قبلی، هر یک از $O(n^3)$ زیرمسئله در زمان $O(1)$ محاسبه می‌شوند. پس کل الگوریتم در زمان $O(n^3)$ اجرا می‌شود. در این‌جا هم می‌توان k را حذف کرد، چرا که به‌روزرسانی بیش از حد برای ما ضرری ندارد:

```
1  d_{u,u} = 0(∀u ∈ V), d_{u,v} = ∞ (∀u ≠ v)
2  d_{u,v} = w(u,w)(∀(u,v) ∈ E)
3  for k = 1 to n
4      for u ∈ V
5          for v ∈ V
6              if d_{u,k} + d_{k,v} < d_{u,v}
7                  d_{u,v} = d_{u,k} + d_{k,v}
```

الگوریتم فوق، فلوید_وارشال نام دارد که پیچیدگی زمانی آن، همانطور که مطرح شد، $O(V^3)$ است.

تمرین: با استفاده از این روش نیز روشی برای پیدا کردن رئوس یک دور منفی در گراف(در صورت وجود) بیابید.(بررسی این که آیا دور منفی وجود دارد یا نه، با بررسی این که آیا فاصله‌ی رأسی از خودش منفی می‌شود یا نه، انجام می‌شود.)

راه ۳: جانسون $O(V^2 logV + VE)$

تا به اینجای کار بهترین الگوریتم اجرای دایکسترا از تمامی رئوس بود که برای گراف‌های غیر چگال($E = o(V^2)$) بهترین عملکرد را داشت، امّا در گراف‌هایی با یال منفی درست کار نمی‌کرد.

تعریف: تابع $p : V \to Q$ را یک «پتانسیل شدنی [۸] » می‌نامیم، اگر روی رأس‌های گراف وزن‌دار جهت‌دار $D = (V, A)$ دارای شرط زیر باشد:

$$\forall a = (u, v) \in A, w(a) \geq p(v) - p(u) \implies w(a) - p(v) + p(u) \geq 0$$

و $w'(a)$ را برای هر یال a در گراف، مطابق زیر در نظر بگیرید:

$$w'(a) = w(a) - p(v) + p(u)$$

سیستم وزن‌گذاری جدید، وزن تمامی یال‌ها را نامنفی می‌کند. حال بررسی می‌کنیم که آیا یال‌های کوتاهترین مسیر، در این سیستم وزن‌گذاری تغییر می‌کند یا نه. یک مسیر مانند $v_0, v_1, v_2, ..., v_s$ در نظر بگیرید.

$$\left.\begin{array}{l} w'(v_0, v_1) = w(v_0, v_1) - p(v_1) + p(v_0) \\ w'(v_1, v_2) = w(v_1, v_2) - p(v_2) + p(v_1) \\ \dots \\ w'(v_{s-1}, v_s) = w(v_{s-1}, v_s) - p(v_s) + p(v_{s-1}) \end{array}\right\} \implies \sum_{i=0}^{s-1} w'(v_i, v_{i+1}) = \sum_{i=0}^{s-1} w(v_i, v_{i+1}) + p(v_0) - p(v_s)$$

از آن‌جا که به تمامی مسیرهای بین v_0 و v_s یک عدد ثابت اضافه می‌شود، کوتاهترین مسیر تغییر نمی‌کند.

سؤال: پتانسیل را چگونه پیدا کنیم؟

با اجرای الگوریتم بلمن_فورد از یک رأس، مانند s ، شرط مطلوب به دست می‌آید. چون اگر به ازای یک یال مانند u, v شرط مطلوب برقرار نباشد، یعنی:

$$d(v) > d(u) + w(u, v)$$

[8] Feasible Potential

هنوز هم می‌توان این یال را ریلکس کرد. که این به معنای دور منفی داشتن گراف است و گراف کلاً تابع پتانسیل ندارد، اگر و تنها اگر دور منفی داشته‌باشد. چرا که اگر دور منفی نداشت، حاصل بلمن_فورد، خود یک تابع پتانسیل شدنی است و اگر هم دور منفی داشته‌باشد، می‌توان گفت:

$$0 > w(c) = \sum w(v_i, v_{i+1}) \geq \sum p(v_{i+1}) - p(v_i) = 0$$

از تناقض فوق می‌توان نتیجه گرفت که هیچ تابع پتانسیل شدنی‌ای برای گراف با دور منفی وجود ندارد.

الگوریتم جانسون [۹] به نحو زیر عمل می‌کند:

۱ـ یک بار الگوریتم بلمن_فورد را اجرا می‌کنیم و تابع پتانسیل را به دست می‌آوریم.

۲ـ وزن‌دهی جدید را از روی وزن‌دهی اصلی محاسبه می‌کنیم.

۳ـ V بار دایکسترا را اجرا می‌کنیم (از هر رأس، یک بار).

۴ـ طول تمامی کوتاهترین مسیرهای گراف اصلی را از روی طول کوتاهترین مسیر در گراف جدید و مقادیر تابع پتانسیل در ابتدا و انتهای مسیر، محاسبه می‌کنیم.

این الگوریتم از نظر پیچیدگی زمانی برابر است با V بار اجرای دایکسترا و یک بار اجرای بلمن_فورد، که این برابر است با $O(V^2 logV + VE)$.

[9] Johnson

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

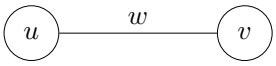
جلسه ۱۲: دور با کمترین میانگین وزن و درخت کم‌عمق_سبک

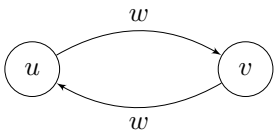
نگارنده: جواد فرخ‌نژاد

در دو جلسه قبل مسئله‌ی کوتاهترین مسیر در گراف‌ها را بررسی کردیم و الگوریتم‌های دایکسترا[۱]، بلمن_فورد[۲]، فلوید_وارشال[۳] و جانسون[۴] را در این باره دیدیم. در این جلسه ابتدا یک مسئله‌ی کلاسیک با نام دور با کمترین میانگین وزن[۵] را بررسی کرده و سپس قضیه‌ای در رابطه با درخت کم‌عمق_سبک[۶] بیان و اثبات می‌کنیم.

۱ دور با کمترین میانگین وزن

در جلسه‌ی ۱۰ دیدیم که خیلی از الگوریتم‌ها برای گراف‌های جهتدار را می‌توان برای گراف‌های بدون‌جهت هم به‌کار برد به این‌صورت که از روی گراف بدون‌جهت یک گراف جهتدار می‌سازیم و الگوریتم را روی آن به‌کار می‌گیریم. این گراف اینطوری ساخته می‌شد که برای هر یال بین دو رأس u و v از گراف اصلی دو یال جهتدار یکی از u به v و یکی از v به u درنظر می‌گرفتیم. اما این روش همواره جواب نمی‌دهد مثلا الگوریتم بلمن_فورد برای گراف‌های بدون‌جهت که حداقل یک یال منفی دارند جواب نمی‌دهد. زیرا اگر با این روش گراف جدید را بسازیم در این گراف یک دور منفی به طول ۲ ساخته می‌شود و الگوریتم بلمن_فورد برای گراف‌هایی که دور منفی دارند جواب نمی‌دهد.





این مشکل را به سادگی نمی‌توان حل کرد. به‌طورکلی برای حل مسئله‌ی کوتاهترین مسیر در گراف‌های بدون‌جهت با وزن منفی هم الگوریتم کلی وجود دارد که از مسئله‌ی تطابق[۷] و مسئله‌ی تی_جوین[۸] استفاده می‌کند. در درس بهینه‌سازی ترکیبیاتی به این موضوع پرداخته می‌شود.

حال برای اینکه بتوانیم الگوریتم بلمن_فورد را روی گراف‌های با دور منفی هم به کار بگیریم می‌خواهیم به وزن هر یال یک عدد $-\epsilon$ اضافه کنیم به‌طوریکه گراف جدید دور منفی نداشته باشد، درضمن به دنبال کوچکترین عدد $-\epsilon$ موردنظر هستیم ($\epsilon > 0$). این عدد دقیقا همان قدرمطلق میانگین وزن یال‌ها در دور با کمترین میانگین وزن است. یعنی اگر برای هر دور C از گراف تعریف کنیم:

$$\mu(C) = \frac{\sum_{e \in C} w(e)}{|C|}$$

[۱] Dijkstra
[۲] Bellman-Ford
[۳] Floyd-Warshall
[۴] Johnson
[۵] Minimum Mean Weight Cycle
[۶] Shallow-Light Tree
[۷] Matching
[۸] T-join

آنگاه کمترین مقداری که μ به ازای دورهای مختلف C از گراف می‌تواند اتخاذ کند ϵ است که به دنبالش هستیم. درواقع اگر عدد $-\epsilon$ را به تمام یال‌های گراف اضافه کنیم آنگاه به میانگین وزن یال‌ها در هر دور دقیقا $-\epsilon$ اضافه می‌شود و بنابراین از این به بعد دور منفی نخواهیم داشت.

برای سادگی فرض می‌کنیم رأسی در گراف مانند s وجود دارد که به تمام رأس‌های گراف مسیر جهتدار دارد، اگر چنین رأسی موجود نبود خودمان یک رأس جدید s به گراف اضافه می‌کنیم و از s به تمام رئوس گراف یال جهتدار با وزن صفر اضافه می‌کنیم و از این به‌بعد با گراف جدید کار می‌کنیم. دقت کنید که این تغییر لطمه‌ای به روند کار وارد نمی‌کند چون با این تغییر، در گراف دور جدیدی تشکیل نمی‌شود و دورهای قبلی هم میانگین وزنشان تغییری نمی‌کند. حال تعریف می‌کنیم:

$$d_k(v) = \text{وزنِ کم‌وزن‌ترین مسیر از } s \text{ به } v \text{ که دقیقا } k \text{ یال دارد}$$

$$A = \min_{v \in V} \left\{ \max_{0 \le k < n} \left\{ \frac{d_n(v) - d_k(v)}{n-k} \right\} \right\}$$

که در آن V مجموعه‌ی رئوس گراف و n تعداد رأس‌های گراف است.

قضیه ۱. $\epsilon = A$

قبل از بیان اثبات قضیه به این نکته دقت کنید که محاسبه‌ی A در زمان $O(|E||V|)$ قابل انجام است (E مجموعه‌ی یال‌های گراف است). زیرا رابطه‌ی بازگشتی زیر را داریم:

$$\forall v \in V, 0 \le k < n: \quad d_{k+1}(v) = \min_{u\,:\,uv \in E} \{ d_k(v) + w(uv) \}$$

بنابراین با مطالبی که از برنامه‌ریزی پویا آموختیم می‌توانیم d_k را برای تمام رئوس گراف در زمان $O(|E|)$ بیابیم چون برای k ثابت هر یال حداکثر یکبار در رابطه‌ی بازگشتی بالا ظاهر می‌شود. تعداد k های مختلف هم $n = |V|$ است. بنابراین در زمان $O(|E||V|)$ می‌توان تمام مقادیر $d_k(v)$ را یافت. حال برای یک رأس v دلخواه $\max_{0 \le k < n} \{ \frac{d_n(v) - d_k(v)}{n-k} \}$ در زمان $O(|V|)$ قابل محاسبه است و مینیمم گرفتن روی این مقادیر برای v های مختلف هم زمان $O(|V|)$ می‌گیرد. نهایتا زمان محاسبه‌ی A برابر خواهد بود با: $O(|V|^2 + |V||E|) = O(|V||E|)$. حال به اثبات قضیه می‌پردازیم.

اثبات. بدون کاسته شدن از کلیت مسئله می‌توان فرض کرد $\epsilon = 0$ (دقت کنید که منظور از ϵ کمترین عدد بین میانگین وزن دورهای گراف است.) چون اگر به وزن همه‌ی یال‌های گراف عدد $-\epsilon$ را اضافه کنیم آنگاه ϵ مربوط به گراف جدید برابر با صفر است و عدد A برای گراف جدید دقیقا به اندازه‌ی $-\epsilon$ زیاد شده‌است. چراکه با این تغییرات $d_k(v)$ که طول k دارد به اندازه‌ی $-k\epsilon$ زیاد می‌شود و مشابها $d_n(v)$ نیز به اندازه‌ی $-n\epsilon$ زیاد می‌شود و بنابراین هر کدام از $\frac{d_n(v) - d_k(v)}{n-k}$ ها دقیقا به اندازه‌ی $-\epsilon$ زیاد می‌شوند پس درکل A نیز به اندازه‌ی $-\epsilon$ زیاد می‌شود. پس اگر حکم در گراف جدید اثبات شود برای گراف قبلی نیز صادق خواهد بود.

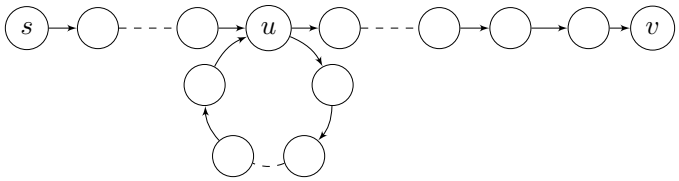
حال با فرض اینکه $\epsilon = 0$ ثابت می‌کنیم $A = 0$، برای این منظور ثابت می‌کنیم $A \le 0$ و $A \ge 0$.

اثبات $A \ge 0$:

فرض کنید v یک رأس دلخواه باشد اگر ثابت کنیم $\max_{0 \le k < n} \{ \frac{d_n(v) - d_k(v)}{n-k} \} \ge 0$ آنگاه A که یکی از این مقادیر است نیز نامنفی خواهد بود.

مسیری از s به v با n رأس را در نظر می‌گیریم که وزنش $d_n(v)$ است. اگر چنین مسیری به طول n از s به v موجود نباشد مقدار $\max_{0 \le k < n} \{ \frac{d_n(v) - d_k(v)}{n-k} \}$ بی‌نهایت است زیرا فرض کرده بودیم از s به v حداقل یک مسیر وجود دارد پس به ازای حداقل یک مقدار $0 \le k < n$، $d_k(v)$ متناهی است و بنابراین کسر $\frac{d_n(v) - d_k(v)}{n-k}$ بی‌نهایت است. پس درکل $\max_{0 \le k < n} \{ \frac{d_n(v) - d_k(v)}{n-k} \}$ نیز بی‌نهایت است. اگر چنین مسیری موجود باشد آنگاه این مسیر دارای رأس تکراری است چرا که طول مسیر n است و تعداد رئوس گراف نیز n است.

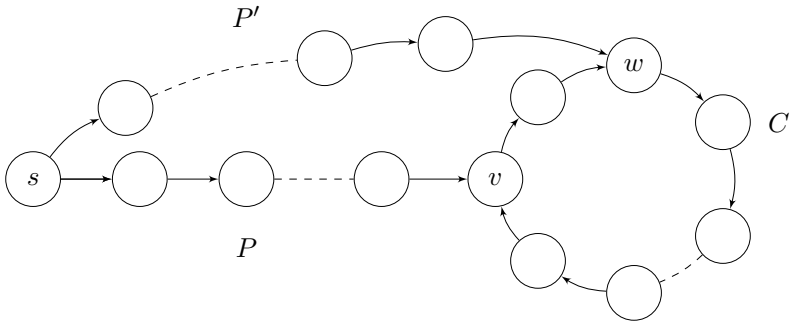
فرض کنید رأس u در این مسیر تکرار شده باشد، بنابراین این مسیر از s به v تشکیل شده‌است از یک مسیر از s به u سپس یک دور از u به u سپس یک مسیر از u به v.



وزن دور از u به u نامنفی است بنابراین اگر این دور را از مسیر s به v حذف کنیم به یک مسیر جدید از s به v می‌رسیم که تعداد یال‌هایش کمتر است و وزن آن نیز کمترمساوی $d_n(v)$ است. مثلا t یال دارد $(t < n)$ و وزنش w است $(w \leq d_n(v))$. حال بنابر تعریف $d_t(v)$ داریم $w \geq d_t(v)$ بنابراین به ازای این t که پیدا کردیم مقدار کسر $\frac{d_n(v)-d_t(v)}{n-t}$ نامنفی است پس $\max_{0 \leq k < n} \left\{\frac{d_n(v)-d_k(v)}{n-k}\right\}$ نیز نامنفی است.

اثبات $A \leq 0$:

فرض کنید C دوری از گراف باشد که میانگین وزن یال‌هایش صفر شده‌است، پس جمع وزن یال‌هایش نیز صفر است.
P را مسیری درنظر می‌گیریم که از s به یکی از رأس‌های C می‌رود و در بین تمام چنین مسیرهایی کمترین وزن را دارد. ادعا می‌کنیم که برای هر رأس از C مثل w اگر مسیری را درنظر بگیریم که از s شروع می‌شود و مطابق با P به v می‌رود سپس از v روی C حرکت می‌کند تا به w برسد، آنگاه وزن این مسیر برابر است با $\delta(s,w)$ که منظور از نماد $\delta(s,w)$ مشابه جلسات پیش وزنِ کم‌وزن‌ترین مسیر از s به w در کل گراف اصلی است.



برای اثبات ادعا با برهان‌خلف فرض کنید مسیر بیان شده کم‌وزن‌ترین مسیر از s به w نباشد. کم‌وزن‌ترین مسیر از s به w را P' می‌نامیم.
اگر مسیری را درنظر بگیریم که از s شروع می‌شود و مطابق با P' به w می‌رود سپس از w روی C حرکت می‌کند تا به v برسد، و جمع وزن یال‌هایی که از w تا v روی C پیموده شده‌اند را l بگیریم، در نهایت به یک مسیر از s به v با جمع وزن یال‌های $w(P') + l$ می‌رسیم، پس بنابر تعریف مسیر P داریم $w(P) \leq w(P') + l$. حال این رابطه را به این‌صورت بازنویسی می‌کنیم $w(P) - l \leq w(P')$.
سمت کوچکتر نامساوی بالا درواقع وزن یک مسیر از s به w است، به این‌صورت که ابتدا از s مطابق با P به v می‌رویم که جمع وزن یال‌هایش $w(P)$ است سپس از v روی C به w می‌رویم که جمع وزن یال‌هایش $-l$ است (چون جمع وزن یال‌ها در دور C صفر است و از w تا v برابر l است).
پس مسیری از s به w یافتیم که جمع وزن یال‌هایش از $w(P')$ بیشتر نیست. پس بنابر تعریف P' این مسیر یافته‌شده نیز باید کم‌وزن‌ترین مسیر از s به w باشد که با فرض اولیه در تناقض است. پس ادعا اثبات شد.
حال با شروع از s مطابق با P حرکت می‌کنیم تا به v برسیم. تعداد یال‌های پیموده‌شده حداکثر $n-1$ است. سپس روی دور C آنقدر می‌چرخیم تا تعداد یال‌های پیموده‌شده به n برسد. فرض کنید نهایتا در رأس w متوقف شویم. پس مسیری از s به w با طول دقیقا n یافتیم

که جمع وزن یال‌هایش برابر با $\delta(s,w)$ است چرا که دور زدن در C جمع وزن یال‌ها را زیاد نمی‌کند و با دقت کردن به ادعای اثبات‌شده در کل این رابطه را داریم: وزن مسیری که از s به w با طول n یافتیم $= d_n(w) = \delta(s,w)$.
چون $\delta(s,w)$ کم‌وزن‌ترین مسیر در کل از s به w است پس برای هر k داریم $\delta(s,w) \le d_k(w)$ با جایگذاری $d_n(w)$ به‌جای $\delta(s,w)$ می‌بینیم که:

$$\forall \circ \le k < n: \quad d_n(w) \le d_k(w) \Longrightarrow \frac{d_n(w) - d_k(w)}{n-k} \le \circ \Longrightarrow \max_{\circ \le k < n} \left\{ \frac{d_n(w) - d_k(w)}{n-k} \right\} \le \circ$$

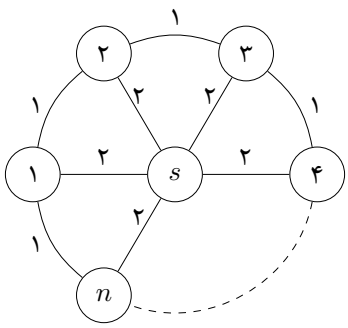
بنابراین یک v یافتیم که به ازای آن $\max_{\circ \le k < n} \left\{ \frac{d_n(v) - d_k(v)}{n-k} \right\}$ نامثبت است، پس A که مینیمم این مقادیر است نیز نامثبت است. □

۲ درخت کم‌عمق‌ـ سبک

در جلسات پیش با درخت فراگیر کمینه[۹] و درخت کوتاهترین مسیر[۱۰] آشنا شدیم. البته تعریف درخت کوتاهترین مسیر در جلسات پیش گفته نشد که اینجا تعریف را بیان می‌کنیم.

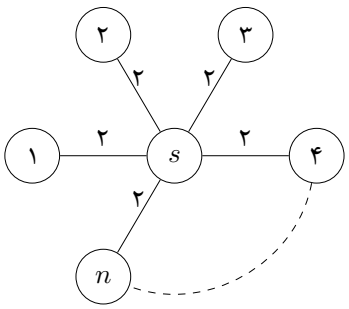
در الگوریتم‌های BFS و DFS وقتی برای اولین بار به رأس v می‌رسیدیم، پدر آن را رأسی قرار می‌دادیم که از آن وارد v شده‌ایم.
مشابها اگر مثلا الگوریتم بلمن‌ـ فورد را با شروع از رأس s روی یک گراف اجرا کنیم، برای یک رأس v پدر آن را رأس u قرار می‌دهیم به‌طوریکه $d(v)$ برای آخرین بار توسط ریلکس[۱۱] کردن یال uv از گراف تغییر کرده‌باشد و به مقدار $\delta(s,v)$ رسیده‌باشد. بنابراین بسته به اینکه کدام الگوریتم را به کار می‌گیریم می‌توان از روی آن یک درخت شامل تمام رئوس گراف ساخت که ریشه‌ی آن، رأس شروع الگوریتم باشد.

به‌طور کلی انتظار نداریم که در یک گراف دلخواه بین یال‌های درخت فراگیر کمینه و درخت کوتاهترین مسیر رابطه‌ی ساده‌ای برقرار باشد. مثلا گراف زیر را درنظر بگیرید:

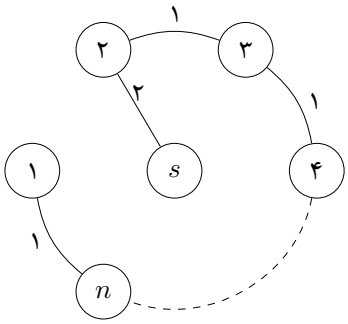


در این گراف اگر درخت فراگیر کمینه و درخت کوتاهترین مسیر (با شروع از رأس s) را در نظر بگیریم، این دو درخت فقط در یک یال مشترک‌اند و وزن درخت فراگیر کمینه برابر با $n+۱$ و وزن درخت کوتاهترین مسیر $۲n$ است.

[۹] Minimum Spanning Tree
[۱۰] Shortest Path Tree
[۱۱] Relax



درخت کوتاهترین مسیر



درخت فراگیر کمینه

حال قضیه‌ی زیر بیان می‌کند که در هر گراف دلخواه می‌توان یک درخت فراگیر انتخاب کرد که به‌نوعی هم خواص درخت فراگیر کمینه را داشته باشد و هم خواص درخت کوتاهترین مسیر را.

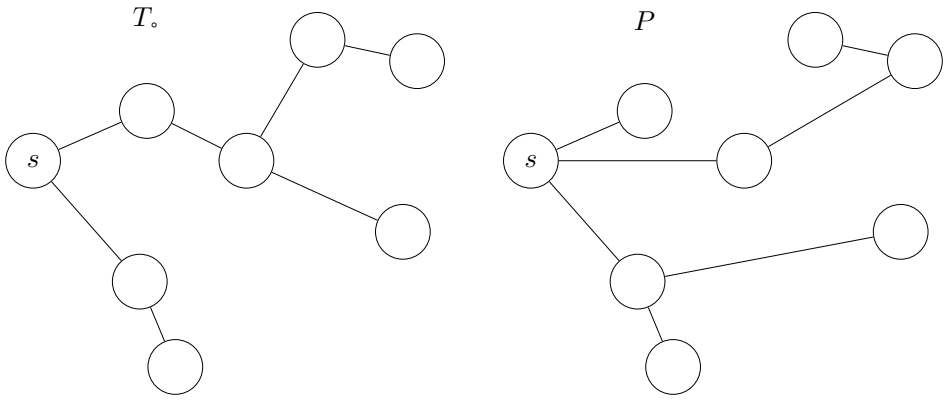
قضیه ۲. *فرض کنید $G = (V, E)$ گراف بدون‌جهت و $c : E \to \mathbb{Q}^+$ تابع وزن باشد بعلاوه $s \in V$ و $\epsilon > 0$ داده شده‌اند. آنگاه درخت فراگیر T از G وجود دارد به‌طوریکه:*

(۱) $\forall v \in V : \quad \delta_T(s,v) \le (1+\epsilon)\delta_G(s,v)$

(۲) $\sum_{e\in T} c(e) \le (1+\frac{2}{\epsilon})w(MST)$

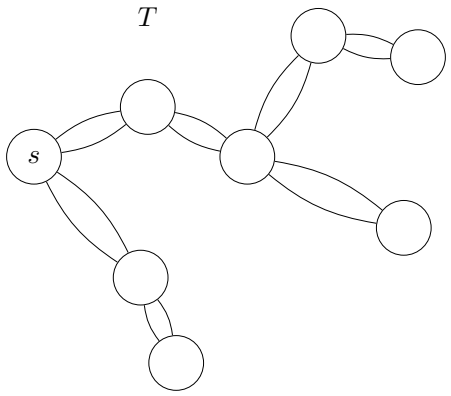
(۳) *این درخت را می‌توان در زمان خوبی یعنی $O\big(|E| + |V|\lg|V|\big)$ پیدا کرد*

اثبات. فرض کنید T_0 یک درخت فراگیر کمینه و P یک درخت کوتاهترین مسیر با شروع از s باشد. هر دوی این درخت‌ها را می‌توان در زمان $O\big(|E| + |V|\lg|V|\big)$ بدست آورد. حال قرار می‌دهیم $T = T_0$. این T قرار است تبدیل به درخت نهایی شود که آن را ارائه می‌دهیم.



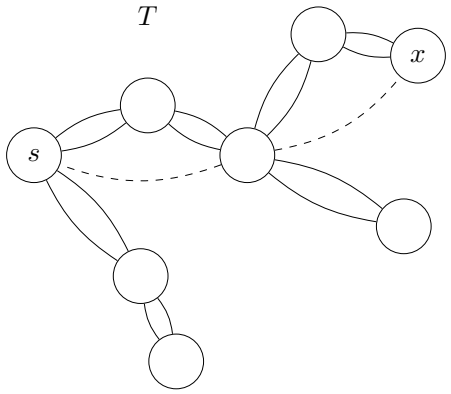
یال‌های T را دوبرابر می‌کنیم به این‌صورت که برای هر یال uv بین دو رأس u و v یک یال دیگر نیز بین u و v با همان وزن قرار می‌دهیم. چون درجه‌ی تمام رئوس گراف جدید زوج است بنابراین دارای تور اویلری[۱۲] (گشتی که از هر یک از یال‌های گراف دقیقا یکبار بگذرد و نهایتا به رأس اولیه برگردد) است.

[۱۲] Eulerian Tour



حال از رأس s شروع به پیمایش این تور اویلری در T می‌کنیم. در مسیر خود هرجا برای اولین بار به رأسی مانند x رسیدیم رابطه‌ی $\delta_T(s,x) \leq (1+\epsilon)\delta_G(s,x)$ را بررسی می‌کنیم. اگر برقرار بود کاری انجام نمی‌دهیم و مسیر را روی تور اویلری ادامه می‌دهیم. اما اگر $\delta_T(s,x) > (1+\epsilon)\delta_G(s,x)$ بود آنگاه یال‌های مسیری که از s به x در P وجود دارد (یعنی همان مسیر با وزن کمینه از s به x در G) را به گراف T اضافه می‌کنیم.

دقت کنید که وقتی می‌خواهیم یال‌های مسیر از s به x در P را به گراف T اضافه کنیم ممکن است گراف T شامل تعدادی از یال‌های این مسیر باشد در این‌صورت یال‌های تکراری را دوباره اضافه **نمی‌کنیم**.



نهایتا پس از اتمام تور اویلری ما یک گراف T داریم که در شرط (۱) قضیه صدق می‌کند. چرا که برای هر رأس x از گراف وقتی در تور اویلری به رأس x می‌رسیم، یا شرط $\delta_T(s,x) \leq (1+\epsilon)\delta_G(s,x)$ برقرار است و از این به بعد هم برقرار می‌ماند، یا اینکه ما مسیری که بین s و x با وزن کمینه‌ی $\delta_G(s,x)$ در G وجود دارد را به T اضافه کردیم یعنی از این به بعد $\delta_T(s,x) = \delta_G(s,x)$، پس از این به بعد درگراف T جدید شرط $\delta_T(s,x) \leq (1+\epsilon)\delta_G(s,x)$ برقرار است و برقرار می‌ماند.

در انتهای کار گراف T که داریم ممکن است درخت نباشد، در این‌صورت یک الگوریتم دایکسترا روی آن با شروع از رأس s اجرا می‌کنیم و یک زیردرخت فراگیر کمینه داخل T بدست می‌آوریم. کافی است ثابت کنیم این زیردرخت بدست آمده شرط (۲) را ارضاء می‌کند. چرا که در این فرآیند ممکن است تعدادی از یال‌های گراف T نهایی کم شود بنابراین جمع وزن یال‌های این زیردرخت حداکثر به اندازه‌ی جمع وزن یال‌های T نهایی است. بنابراین کافی‌است برای T نهایی بدست آمده که ممکن است درخت نباشد شرط (۲) قضیه را اثبات کنیم.

فرض می‌کنیم $v_1, v_2, ..., v_k$ به ترتیب رئوسی باشند که در فرآیند بالا ما یال‌های کم‌وزن‌ترین مسیر از s به آنها در P را به گراف T اضافه کردیم (تعریف می‌کنیم $v_0 = s$). $c_w(v_{i-1}, v_i)$ را وزن زیرگشتِ w از اولین باری که به v_{i-1} می‌رسیم تا اولین باری که به v_i می‌رسیم درنظر می‌گیریم.

بنابر تعریف v_i داریم $\delta_T(s,v_i) > (1+\epsilon)\delta_G(s,v_i)$ و از نامساوی مثلث در T داریم $\delta_G(s,v_{i-1}) + c_w(v_{i-1},v_i) \geq \delta_T(s,v_i)$.

چرا که مسیر با وزن $\delta_G(s,v_{i-1})$ از s به v_{i-1} (همان مسیر از s به v_{i-1} در P) در مرحله قبلی به T اضافه شده‌است و در T فعلی مسیری

از s به v_i با وزن $\delta_G(s,v_{i-1})+c_w(v_{i-1},v_i)$ وجود دارد. از دو نامساوی اخیر برای هر $1\le i\le k$ داریم:

$$\delta_G(s,v_{i-1})+c_w(v_{i-1},v_i)>(1+\epsilon)\delta_G(s,v_i)$$

برای اینکه رابطه‌ی بالا به ازای $i=1$ نیز درست باشد تعریف کردیم $v_0=s$. حال با جمع زدن طرفین این روابط برای $1\le i\le k$ داریم:

$$\begin{aligned}
&\sum_{i=1}^{k}\delta_G(s,v_{i-1})+\sum_{i=1}^{k}c_w(v_{i-1},v_i)>\sum_{i=1}^{k}(1+\epsilon)\delta_G(s,v_i)\\
\Rightarrow\quad &\delta_G(s,v_0)+\sum_{i=2}^{k}\delta_G(s,v_{i-1})+\sum_{i=1}^{k}c_w(v_{i-1},v_i)>\sum_{i=1}^{k-1}\delta_G(s,v_i)+\delta_G(s,v_k)+\epsilon\sum_{i=1}^{k}\delta_G(s,v_i)\\
\Rightarrow\quad &\sum_{i=1}^{k}c_w(v_{i-1},v_i)>\delta_G(s,v_k)+\epsilon\sum_{i=1}^{k}\delta_G(s,v_i)\Rightarrow\sum_{i=1}^{k}c_w(v_{i-1},v_i)>\epsilon\sum_{i=1}^{k}\delta_G(s,v_i)
\end{aligned}$$

اگر به $\sum_{i=1}^{k}c_w(v_{i-1},v_i)$ دقت کنیم می‌بینیم که این حاصل‌جمع درواقع چیزی جز جمع وزن تعدادی از یال‌های مجزای تور اویلری نیست. بنابراین این حاصل‌جمع حداکثر به اندازه‌ی جمع وزن کل یال‌های تور اویلری است. بنابراین $\sum_{i=1}^{k}c_w(v_{i-1},v_i)\le 2\sum_{e\in T_0}c(e)$.
پس به‌طور کلی داریم:

$$\begin{aligned}
&2\sum_{e\in T_0}c(e)\ge\epsilon\sum_{i=1}^{k}\delta_G(s,v_i)\Rightarrow\frac{2}{\epsilon}\sum_{e\in T_0}c(e)\ge\sum_{i=1}^{k}\delta_G(s,v_i)\\
\Rightarrow\quad &\big(1+\frac{2}{\epsilon}\big)\sum_{e\in T_0}c(e)\ge\sum_{e\in T_0}c(e)+\sum_{i=1}^{k}\delta_G(s,v_i)
\end{aligned}$$

سمت راست نامساوی بالا جمع وزن یال‌های گراف T نهایی است. چون هریال از گراف T نهایی را درنظر بگیریم یا از ابتدا در T موجود بوده یعنی در T_0 بوده که در این صورت وزن این یال در $\sum_{e\in T_0}c(e)$ محاسبه می‌شود، یا اینکه این یال بعدها به T اضافه شده که در این‌صورت هم در $\sum_{i=1}^{k}\delta_G(s,v_i)$ محاسبه می‌شود. بنابراین شرط (۲) قضیه، هم برای T نهایی و هم برای زیردرختی که از روی T توسط الگوریتم دایکسترا بدست آمد برقرار است.

حال به اثبات قسمت (۳) قضیه می‌پردازیم. اگر به روند اثبات دقت کنید می‌بینید که برای اثبات این قسمت کافیست دو چیز را اثبات کنیم، یکی اینکه بررسی کردن رابطه‌ی $\delta_T(s,x)>(1+\epsilon)\delta_G(s,x)$ برای رئوس مختلف x درکل زمان زیادی نمی‌گیرد، دیگری اینکه اضافه کردن یال‌های گراف P به T نیز در کل زمان زیادی نمی‌گیرد.

برای مورد اول چون در مراحل مختلف T تغییر می‌کند بنابراین برای یک x ثابت ممکن است $\delta_T(s,x)$ به ازای T های مختلف تغییر کند برای همین محاسبه‌ی $\delta_T(s,x)$ ممکن است سخت باشد. برای حل این مشکل فرآیند را اینطوری تغییر می‌دهیم که به‌جای بررسی کردن رابطه‌ی $\delta_T(s,x)>(1+\epsilon)\delta_G(s,x)$ رابطه‌ی $\delta_G(s,y)+c_w(y,x)>(1+\epsilon)\delta_G(s,x)$ را بررسی می‌کنیم. که در آن y آخرین رأس از تور اویلری قبل از x است که مسیر از s به y در P را به T اضافه کردیم.
می‌توانیم درحین حرکت در تور اویلری یک متغیر M نگه‌داری کنیم و همواره وزن یالی که از آن گذر می‌کنیم را با M جمع کنیم و وقتی به رأسی رسیدیم که مجبور شدیم مسیرش در P را به T اضافه کنیم آنگاه M را صفر می‌کنیم. در این‌صورت وقتی به یک رأس مثل x می‌رسیم بررسی رابطه‌ی $\delta_G(s,y)+c_w(y,x)>(1+\epsilon)\delta_G(s,x)$ در زمان $O(1)$ قابل انجام است. چون مقادیر $\delta_G(s,y)$ و $\delta_G(s,x)$ را از ابتدای راه‌حل ذخیره داریم و مقدار $c_w(y,x)$ هم همین M فعلی است که ذخیره کردیم. اگر به روند اثبات نگاه کنیم می‌بینیم که اگر الگوریتم را این‌طوری که بیان شد تغییر دهیم باز هم T نهایی که به آن می‌رسیم شرایط قضیه را ارضاء می‌کند.

برای مورد دوم به ازای هر رأس از درخت P یک متغیر boolean ذخیره می‌کنیم که آیا یالی که از این رأس به پدرش در P وجود دارد تا حالا به گراف T اضافه شده‌است یا نه. در این‌صورت وقتی بخواهیم مسیر از s به x در P را به T اضافه کنیم باید در درخت P از رأس

x به بالا برویم تا به ریشه یعنی s برسیم و هر یالی که اضافه نشده بود را اضافه کنیم.
اگر در این بالا رفتن به رأسی مثل w رسیدیم که یالی که بین w و پدرش در P است قبلا به T اضافه شده باشد آنگاه تمام یال‌های از w به بالا نیز قبلا به T اضافه شده‌اند (چرا؟). پس با این کار درواقع هریال از درخت P حداکثر یکبار بررسی می‌شود.
به‌طور کلی با این نحوه‌ی پیاده‌سازی می‌توان دید که وقتی در طول تور اویلری شروع به حرکت می‌کنیم حداکثر زمان $O(|E| + |V|)$ صرف می‌شود و درکل الگوریتم هم دوبار از الگوریتم دایکسترا استفاده کردیم (یکبار برای یافتن P در ابتدای راه‌حل و یکبار هم برای یافتن زیردرخت کوتاهترین مسیر از T نهایی که داشتیم) و یکبار هم در ابتدای راه‌حل برای یافتن $T_\circ$ از یکی از الگوریتم‌های یافتن درخت فراگیر کمینه استفاده کردیم. پس در کل زمان یافتن درخت موردنظر $O\big(|E| + |V| \lg |V|\big)$ است.

□

مراجع

[1] Wikipedia.org

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۱۲.۵: کوتاه‌ترین مسیر در عمل

نگارنده: سید پوریا فاطمی

در این جلسه مسئله‌ی کوتاه‌ترین مسیر[1] در عمل را بررسی خواهیم کرد و عمده‌ی جلسه مربوط به پیدا سازی سریع الگوریتم دایکسترا[2] برای پیدا کردن کوتاه‌ترین مسیر بین دو نقطه است. در عمل برنامه‌هایی مثل نقشه داریم که انتظار داریم کوتاه‌ترین مسیر بین یک مبدا تا یک مقصد خاص را بدست بیاورد. می‌دانیم که در بدترین حالت هیچ الگوریتمی نداریم که این مسئله را بتواند سریع‌تر از پیدا کردن کوتاه‌ترین مسیر از یک نقطه تا همه‌ی نقطه‌های حل کند ولی در عمل می‌توان کارهایی کرد. در آخر نیز به مسئله پیدا کردن کوتاه‌ترین مسیر در سیستم‌های توزیع‌شده می‌پردازیم. در مواقع مسئله‌ای که می‌خواهیم در مورد آن حرف بزنیم این است که کوتاه‌ترین مسیر از s به t در گراف G را پیدا کنیم.

۱ بهتر کردن عمل‌کرد الگوریتم دایکسترا

کار بدیهی برای بهتر کردن زمان الگوریتم دایکسترا در عمل این است که وقتی به مقصد دل‌خواه رسیدیم الگوریتم را تمام کنیم و اجرا الگوریتم برای بقیه گراف را متوقف کنیم.

یک کار کمی هوشمندانه‌تر این است که برای پیدا کردن کوتاه‌ترین مسیر از s به t نیاز نیست الگوریتم دایکسترا از s ران شود و می‌توان الگوریتم مربوطه را با برعکس کردن یال‌های گراف از t اجرا کرد. این کار زمانی که در سمتی از راس s که ربطی به راس t ندارد تعداد زیادی یال و راس باشد کارا است و باعث می‌شود الگوریتم دایکسترا بیهوده آن‌ها را پیمایش نکند. درواقع الگوریتم دایکسترا شبیه آن است که یک گوی را به مرکز مبدا آن دائما بزرگ‌تر کنیم تا به راس مقصد برسیم پس وجود راس‌ها و یال‌های زائد در یک طرف راس مبدا باعث کند شدن الگوریتم می‌شود.

ایده‌ی هوشمندانه‌تر دایکسترای دوطرفه است.

[1] Shortest path
[2] Dijkstra

۲ الگوریتم دایکسترای دوطرفه

به طور هم‌زمان یک نسخه از الگوریتم دایکسترا را از راس s اجرا می‌کنیم و یک نسخه از الگوریتم دایکسترا را از t در گراف G^T اجرا می‌کنیم. چون مدل محاسباتی ما مدل پردازش موازی نیست منظور از به طور هم‌زمان این است که یک یا چند گام از الگوریتم دایکسترا را در یکی از نسخه‌های اجرا می‌کنیم و سپس در نسخه‌ی دیگر یک یا چند گام را انجام می‌دهیم و همین کار را تکرار می‌کنیم. در الگوریتم دایکسترا یک صف الویت[3] داشتیم که با انواع هرم[4] قابل پیدا سازی است. هرمی که متناظر با نسخه اجرا با شروع از s می باشد را Q_f و هرمی که متناظر با نسخه اجرا از t می‌باشد را Q_b می‌نامیم و برای هر راس v دو حافظه $d_f(v), d_b(v)$ داریم. هم‌چنین مجموعه S و T که نشان‌دهنده‌ی راس‌های دیده‌شده تا مرحله مورد نظر از راس s و t هستند را تعریف می‌کنیم. مقداردهی‌های اولیه به شرح زیر است:

$$d_f(s) = 0,\ d_f(v) = \infty\ \ \forall\ v \neq s$$

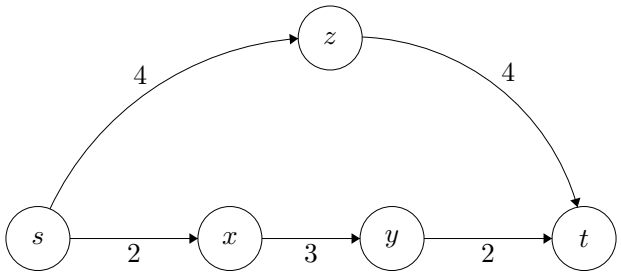
$$d_b(t) = 0,\ d_b(v) = \infty\ \ \forall\ v \neq t$$

$$Q_f = V,\ Q_b = V$$

$$S = \varnothing,\ T = \varnothing$$

الگورتم دایسکترا را یک گام برای نسخه‌ی اول و یک گام برای نسخه‌ی دوم اجرا می‌کنیم و این روند را ادامه می‌دهیم. حال نکته این است که بعد از این‌کار چگونه کوتاه‌ترین مسیر بین s و t پیدا کنیم. آیا صبر می‌کنیم یکی از دو الگوریتم به مبدا یا مقصد برسد؟ این حرف معادل این است که صبر کنیم t به مجموعه S اضافه شود و یا s به مجموعه T اضافه شود. ولی به نظر این روش زیاده کاری است

ایده‌ی هوشمندانه‌تر این است که هر موقع که دو مجموعه با هم تلاقی پیدا کردند روند اجرای الگوریتم را متوقف کنیم و در واقع حالت مطلوب و با زمان اجرای بهتر این است که شرط خاتمه را این شرط بگذاریم نه این‌که یکی از مجموعه‌ها به راس دیگر برسد. پس به محض این‌که اشتراک S و T ناتهی شد، الگوریتم را متوقف می‌کنیم. به نظر می‌رسد در این حالت کوتاه‌ترین مسیر s به t مسیری‌ست که از راس اشتراک دو مجموعه می‌گذرد. یعنی به کمک مسیر بهینه‌ی بدست‌آمده در مجموعه S از s به راس مربوطه رفته و از راس مربوطه به کمک مسیر بهینه بدست‌آمده در مجموعه T به راس t برویم. ولی این حدس درستی نیست و مشکل دارد. مثال زیر را نگاه کنید:



$$S = \{s, x, z\}\ ,\ T = \{t, y, z\}$$

همان‌طور که در مثال بالا می‌بینید راس z اشتراک دو مجموعه S و T است ولی کوتاه‌ترین مسیر s, x, y, t است.

قضیه ۱. *برای پیدا کردن کوتاه‌ترین مسیر باید راس v که عضو مجموعه T باشد و $d_f(v) + d_b(v)$ برای آن کمینه باشد را پیدا کنیم و ادعا این است که:*

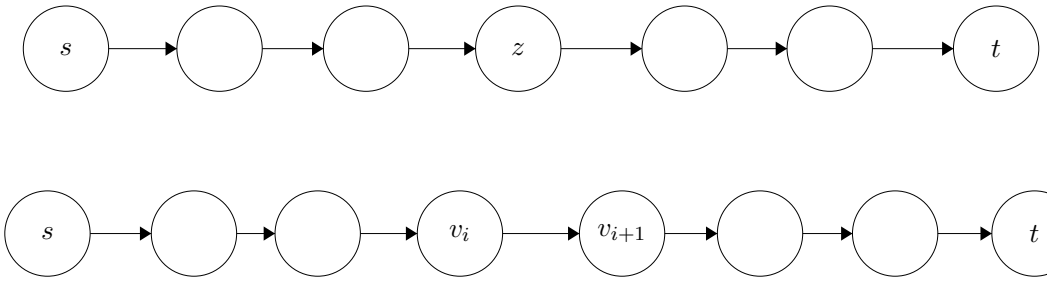
$$\delta(s, t) = d_f(v) + d_b(v)$$

در واقع همین راس v راسی است که روی کوتاه‌ترین مسیر از s به t وجود دارد.

[3] Priority queue
[4] Heap

اثبات. یک کوتاه‌ترین مسیر از s به t را در نظر بگیرید. آخرین راسی که به مجموعه S اضافه شده است را v_i بنامید. اگر بتوانیم اثبات کنیم v_{i+1} به مجموعه T اضافه شده است اثبات تمام است زیرا در الگوریتم دایکسترا وقتی راسی را به یک مجموعه اضافه می‌کنیم کوتاه‌ترین فاصله تا آن راس پیدا شده است. پس می‌دانیم طول کوتاه‌ترین مسیر v_{i+1} تا t برابر $d_b(v_{i+1})$ است و طول کوتاه‌ترین مسیر از s تا v_i برابر $d_f(v_i)$ است. حال نکته این است که وقتی v_i را به مجموعه‌ی S اضافه کرده‌ایم همه یال‌های خروجی از v_i را ریلکس[۵] کرده‌ایم. پس مقدار $d_f(v_{i+1})$ نیز درست است و برابر کوتاه‌ترین مسیر s تا v_{i+1} است. پس اگر راس v_{i+1} در مجموعه T وجود داشته باشد اثبات تمام است و کافی‌ست از راس s به راس v_{i+1} برویم و از v_{i+1} به t برویم.

حال استدلال می‌کنیم که راس v_{i+1} باید به مجموعه T اضافه شده باشد. فرض کنید در لحظه اتمام الگوریتم $S \cap T = \{z\}$ باشد. پس کوتاه‌ترین مسیر از s به z و کوتاه‌ترین مسیر از z به t را می‌دانیم. حال فرض می‌کنیم v_{i+1} به T اضافه نشده باشد. پس نتیجه می‌گیریم طول کوتاه‌ترین مسیر از v_{i+1} به t بزرگ‌تر از طول کوتاه‌ترین مسیر از z به t است. چون z به مجموعه T اضافه‌شده ولی v_{i+1} اضافه نشده است. از طرف دیگر v_{i+1} به مجموعه S اضافه نشده ولی z شده است پس طول کوتاه‌ترین مسیر s به v_{i+1} نیز از طول کوتاه‌ترین مسیر s به z بزرگ‌تر است. پس نتیجه می‌شود طول کوتاه‌ترین مسیر از s به t از مسیر گذرنده از z بیشتر است که این تناقض است. پس راس v_{i+1} باید به مجموعه T اضافه شده باشد و قضیه گفته شده اثبات شد.



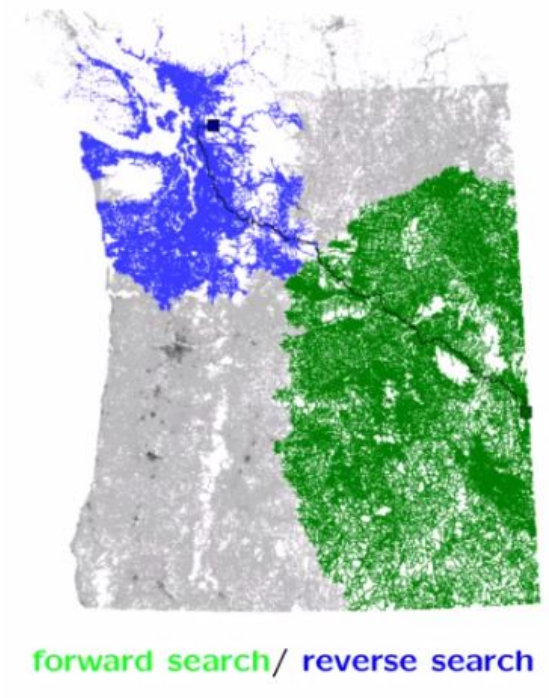
□

در واقع وقتی راسی در مجموعه T را پیدا می‌کنیم که مقدار $d_f(v) + d_b(v)$ برایش کمینه است راسی مانند v_{i+1} را پیدا کرده‌ایم. در بدترین حالت می‌توان دید که زمان اجرای ما عوض نمی‌شود ولی در عمل خیلی از اوقات محاسبات کمتری انجام میشود.

[5] Relax

مثال: در شکل زیر دایکسترا عادی از راس سبز رنگ برای رسیدن به راس آبی رنگ اجرا شده است.

و در شکل زیر دایکسترای دوطرفه برای این دو راس اجرا شده است.



همان‌طور که می‌بینید تعداد زیادی از راس‌ها در روش دوطرفه بررسی نشده‌اند. ☒

برای اطلاعات بیشتر در محاسبه کوتاه‌ترین مسیر در عمل می‌توانید اسم $Andrew\ v.\ Glodberg$ را در اینترنت جستجو کنید.

حتی شرط‌های خاتمه بهتر از شرط گفته‌شده در بالا نیز می‌توان گذاشت! دو مقدار $m_f = min_{v \notin S} d_f(v)$ و $m_b = min_{v \notin T} d_b(v)$ را تعریف کنید. همچنین متغیر μ را برابر طول کوتاه‌ترین مسیر از s به t که تا به حال پیداشده در نظر بگیرید که طول کوتاه‌ترین مسیر از s به t را برای هر راس می‌توان به کمک $d_f(v) + d_b(v)$ پیدا کرد و سپس بین همه‌ی آن‌ها کمینه گرفت.

پس داریم $\mu = min_{v \in V}\{d_f(v) + d_b(v)\}$ حال هرگاه $m_f + m_b$ بیش از μ شد الگوریتم را خاتمه می‌دهیم. می‌توان دید این شرط از شرط قبلی بهتر مساوی است به این معنی که تعداد راس‌هایی که در الگوریتم دایکسترا به سراغ آن‌ها می‌رویم با این شرط خاتمه از شرط خاتمه قبلی بیشتر نیست.

۳ روش دیگری برای بهتر کردن عمل‌کرد الگوریتم دایکسترا!

همان‌طور که در جلسات قبلی دیدیم به یک پتانسیل، پتانسیل شدنی یا معتبر می‌گوییم اگر: $w(uv) \geqslant p(v) - p(u)$

به کمک پتانسیل تعریف شده می‌توانیم یک وزن جدید تعریف کنیم به صورت $w'(uv) = w(uv) - p(v) + p(u)$ و اگه نامساوی گفته شده برای همه‌ی یال‌ها برقرار باشد w'ها برای همه یال‌ها مثبت است و می‌توان از الگوریتم دایکسترا استفاده کرد. در بحث ما مفیدتر است که تعریف معتبر بودن را به $w(uv) \geqslant p(u) - p(v)$ تغییر دهیم و در نتیجه $w'(uv) = w(uv) - p(u) + p(v)$ تعریف می‌شود. این‌جا فرض این است که وزن‌ها مثبت است و نمی‌خواهیم از پتانسیل برای مثبت کردن یال‌ها استفاده کنیم.

در واقع با تغییر وزن‌ها ممکن است تعداد یال‌هایی که در عمل بررسی می‌کنیم در الگوریتم دایکسترا کمتر شود. با تغییر وزن‌ها ممکن است ترتیب بررسی شدن راس‌ها عوض شود و زودتر به راس مقصد برسیم.

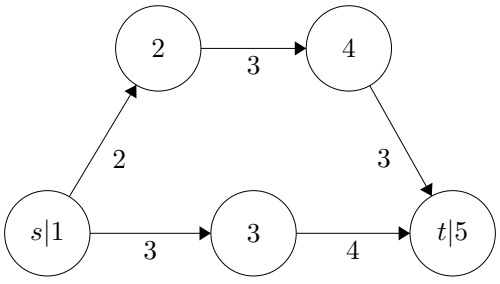
قضیه ۲. *می‌توان وزن یال‌ها گراف را تغییر نداد و برای اعمال پتانسیل شدنی الگوریتم در هر مرحله راسی مانند v که $d(v) + p(v)$ کمینه است را انتخاب کند. این روش مقداری پیاده‌سازی را تمیزتر می‌کند.*

اثبات. دلیل معادل بودن این است که همان‌طور که در جلسات قبلی نشان داده شد اگر مسیری از s به t در نظر بگیرید، داریم:

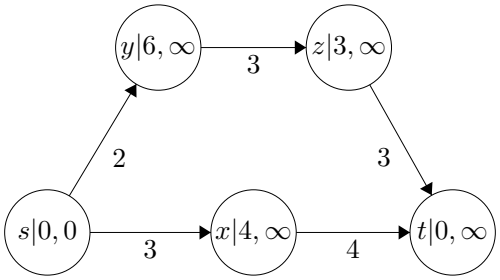
$$\delta_{w'}(s,t) = \delta_w + p(t) - p(s)$$

که علت هم واضح است زیرا برای هر راس میانی در مجموع وزن یال‌ها یک‌بار پتانسیل مثبت می‌آید و یک بار منفی و فقط پتاسیل سر و ته باقی می‌ماند.با توجه به رابطه‌ی گفته‌شده و با توجه به این که $p(s)$ در رابطه ثابت است در کمینه گرفتن تاثیری ندارد و می‌توان در نظر نگرفت و با توجه به این‌که $d(v)$ معادل با کوتاه‌ترین فاصله راس v از راس s تا این لحظه اجرا الگوریتم است پس مقدار انتخاب راسی که مقدار $d(v) + p(v)$ برای آن کمینه است با انتخاب راس به کمک پتانسیل معتبر تعریف‌شده معادل است. □

مثال: دایکسترا عادی را روی گراف زیر اجرا شده و روی هر راس شماره‌ای می‌گذاریم که نشان‌دهنده ترتیب اضافه شدن راس‌ها به مجموعه مربوط به الگوریتم دایکسترا می‌باشد. همان‌طور که مشخص است قبل از رسیدن به راس t نیاز است همه‌ی راس‌ها را به مجموعه رئوس اضافه کنیم.

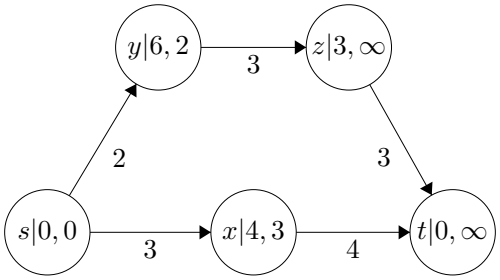


حال فرض کنید به راس‌ها پتانسیلی می‌دهیم که روی هر راس در کنار اسم راس نوشته شده است. حال با اجرای مرحله به مرحله الگوریتم گفته‌شده و نوشتن $d(v)$ برای راس‌های مربوطه (بعد از پتانسیل راس و جدا شده با کاما) داریم: (در مرحله اول مجموعه ما فقط راس مبدا است)



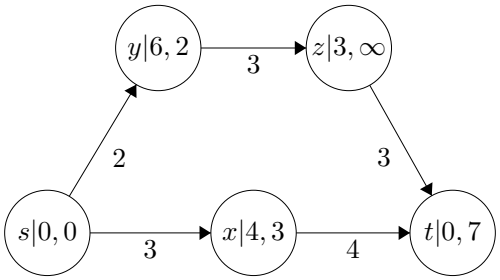
$$S = \{s\}$$

با توجه به این‌که $2 + 6 > 3 + 4$ پس راس x به مجموعه اضافه می‌شود:



$$S = \{s, x\}$$

با توجه به اینکه $2 + 6 > 7 + 0$ پس راس t به مجموعه اضافه می‌شود:



$$S = \{s, x, t\}$$

همان‌طور که دیدیم با پتانسیل‌گذاری گفته‌شده هیچ راسی اضافی‌ای وارد مجموعه الگوریتم دایکسترا نشد و فقط راس‌های روی کوتاه‌ترین مسیر را به مجموعه اضافه کردیم. ☒

به طور کلی اگر پتانسیل هر راس را برابر کوتاه‌ترین فاصله آن راس تا راس t بگذاریم یک پتانسیل معتبر داریم و فقط راس‌های روی کوتاه‌ترین مسیر بررسی می‌شوند. در واقعیت فاصله هر راس تا مقصد را نمی‌دانیم(اگر می‌دانستیم نیاز به اجرای الگوریتم نبود) پس سعی می‌کنم یک کران پایین[۶] از آن را به عنوان پتانسیل قرار دهیم. پس باید فاصله تا راس مقصد را تخمین بزنیم. مثلا اگر گراف حالت هندسی داشته باشد می‌توان گفت کوتاه‌ترین فاصله یک راس تا مقصد حداقل به اندازه فاصله اقلیدسی آن راس تا مقصد است پس می‌توان فاصله

[6] Lower bound

اقلیدسی را به عنوان یک کران پایین برای پتانسیل آن راس در نظر گرفت. در حالت اقلیدسی می‌توان به کمک نامساوی مثلثی نشان داد که پتانسیل‌هایی که به نحو گفته‌شده تعریف می‌شوند معتبر اند. در واقع در این حالت اگر مکان راس v را با $L(v)$ نشان دهیم پتانسیل‌ها به طریق روبه‌رو تعریف می‌شوند: $p(v) = ||L(v) - L(t)||$

در حالتی که گراف حالت اقلیدسی ندارد و نامساوی مثلث در آن صدق نمی‌کند نیز می‌توان پتانسیل را به نحو خوبی تعریف کرد! اگر عدد $m = max_{uv \in E} \frac{||L(v)-L(u)||}{w(uv)}$ را در نظر بگریم و پتانسیل‌ها را به صورت $p(v) = \frac{||L(v)-L(t)||}{m}$ تعریف کنیم باز هم پتانسیل‌ها معتبر خواهند بود.

کار دیگری که قابل تصور است که پتانسیل‌گذاری را با دایکسترای دوطرفه تلفیق کنیم. ولی پتانسیل‌گذاری دیگر به سادگی گفته‌شده نیست و نیاز است شرط‌های دیگری نیز برای پتانسیل‌ها در نظر بگیریم.برای این کار نیاز است برای هر دوطرفی که داریم الگوریتم دایجسترا را اجرا می‌کنیم یک پتانسیل هم در نظر بگیریم. یکی از شرط‌هایی که به کمک آن می‌توان این دو مسئله را تلفیق کرد شرط $p_f + p_b = const$ است و شرط‌های دیگری هم وجود دارد.

سوال دیگر این است که چگونه می‌توان پتانسیل‌های خوبی پیدا کرد؟ در واقع با توجه به این که حالت پتانسیل مساوی با فاصله تا راس t ایده‌آل است می‌خواهیم تا جای ممکن $p(v) = \delta(v, t)$ باشد. در واقع دنبال کران‌های پایین خوب برای $\delta(v, t)$ هستیم. یکی از ایده‌ها استفاده از $Land\ mark$ است. $Land\ mark$ به این معناست که در گراف به طریقی تعدادی از راس‌ها را انتخاب کنیم و بعد برای هر راس فاصله را تا همه آن راس‌ها پیدا کنیم. اگر یکی از راس‌های $Land\ mark$ را راس L در نظر بگیریم تعریف می‌کنیم: $p(v) = \delta(v, L) - \delta(p, L)$ باز می‌توان دید که پتانسیل تعریف‌شده پتانسیل معتبر است و یک کران پایین برای $\delta(v, t)$ است.

البته می‌توان پتانسیل تعریف‌شده را به صورت $p(v) = max_L\{\delta(v, L) - \delta(p, L)\}$ تعریف کرد و به صورت شهودی به این دلیل که داریم کران پایین تعریف می‌کنیم هر چقدر مقدار این کران بیشتر شود بهتر است و به حالت ایده‌آل $\delta(v, t)$ نزدیک‌تر می‌شویم. نکته‌ای که وجود دارد تعداد راس‌های موجود در $Land\ mark$ تعداد راس‌های ثابتی هستند و تعداد آن‌ها زیاد نیست.

برای پیدا کردن $Land\ mark$ نیز روش‌هایی وجود دارد مثلا ساده‌ترین کار انتخاب به صورت تصادفی است یا روش دیگر این است که ابتدا راس دلخواهی را در گراف انتخاب کنیم سپس دورترین راس از راس مربوطه را پیدا کنیم و آن را v_1 بنامیم. بعد از آن دورترین راس از v_1 را پیدا می‌کنیم و آن را v_2 می‌نامیم. سپس دورترین راس از مجموعه $\{v_1, v_2\}$ را v_3 می‌نامیم. این روند را آن‌قدر ادامه می‌دیم تا تعداد راس مورد نظر خود در $Land\ mark$ را بدست بیاوریم. در واقع به صورت شهودی انگار می‌خواهیم راس‌هایی را پیدا کنیم که در محیط گراف پخش‌اند و تا آن‌جایی که می‌شود هر دو راس انتخابی از هم دیگر دور هستند.

یک روش دیگر که مقداری با روش‌های گفته‌شده متفاوت است و جدیدتر از روش‌های گفته‌شده می‌باشد این است که یک سلسله مراتب در گراف در نظر بگیریم و در بین راس‌هایی که مهم‌تر هستند کوتاه‌ترین فاصله را پیدا کنیم. منظور از راس مهم راسی است که در پیدا کردن کوتاه‌ترین مسیر تا تعداد زیادی از راس‌های دیگر دخیل است. کوتاه‌ترین مسیرهای بدست‌آمده بین این رئوس را به عنوان یال جدید یا یال‌های سطح بالاتر به گراف اضافه کنیم و در محاسبه الگوریتم دایکسترا از آن‌ها استفاده کنیم. در واقع گراف را به کمک این روش چندسطحی می‌کنیم. به این روش‌ها، روش‌های سلسله‌مراتبی[۷] می‌گویند. برای دانایی بیشتر از این روش‌ها جستجو کردن را به شما پیشنهاد می‌کنیم.

[7] Hierarchical methods

۴ کوتاه‌ترین مسیر در سیستم های توزیع‌شده

صورت مسئله این است که تعدادی روتر[۸] داریم و بین روترها یک گراف وجود دارد و آن‌ها می‌خواهند کوتاه‌ترین فاصله خود را تا یک مقصدی پیدا کنند. اگر ممکن بود که هر کدام از روترها کل گراف را داشته باشند می‌شد از الگوریتم دایکسترا برای پیدا کردن کوتاه‌ترین مسیر استفاده کرد. ولی مشکل الگوریتم دایکسترا این است که نیاز به اطلاعات $global$ دارد یعنی نیاز است شما کل گراف مربوطه را بدانید ولی در سیستم‌های توزیع‌شده مطلوب نیست که هر راسی(روتری) کل اطلاعات شبکه را داشته باشد. به همین دلیل در سیستم‌های توزیع‌شده شاید بهتر باشد الگوریتمی استفاده کنیم اطلاعات محلی[۹] نیاز دارد پس اینجا ممکن است الگوریتم بلمن‌فورد[۱۰] برای پیاده‌سازی مناسب‌تر باشد زیرا نیاز به اطلاعات محلی دارد و برای ریلکس کردن هر یال فقط نیار به اطلاعات سر و ته یال داریم. در واقع هر روتر کافی است از راس‌هایی که به اون یال دارند d را بگیرد و وزن یال‌های اتصالی با آن‌ها را هم داشته باشد تا بتواند d خود را بروز کند پس اطلاعات محلی نیاز است.

در پیاده‌سازی‌ای که از الگوریتم بلمن‌فورد قبلا گفته‌شده در $n-1$ مرحله همه یال‌ها را ریلکس می‌کردیم ولی شاید بهینه‌تر باشد که ریلکس کردن‌ها را در هر مرحله فقط برای یال‌هایی انجام دهیم که واقعا تغییر کرده‌اند. اگر هر راس از همه‌ی راس‌هایی که به او یال دارند در هر مرحله d آن‌ها را بپرسد تا d مربوط به خود را بروز کند (روش $Pull\ Base$) ممکن است کار اضافه‌ای انجام دهد و از راسی که در مرحله قبل dاش تغییر نکرده، در مورد d سوال پرسیده شود. ولی اگر از این روش استفاده کنیم که هر راسی وقتی dاش آپدیت شد به همه‌ی همسایه‌های خود d جدید را خبر دهد و راس‌ها به کمک پیام‌های دریافتی اگر لازم باشد d خود را بروز کنند (روش $Push\ Base$) این کار اضافه انجام نمی‌شود. شبیه کد آن به شکل زیر میشود:

```
1  for i = 1 to n − 1
2      for each v ∈ V
3          if d(v) has changed in the previous round
4              for each vw ∈ E
5                  relax(vw)
```

در واقع در کد بالا که برای روش $Push\ Base$ است ریلکس کردن در هر مرحله را راس w انجام می‌دهد. البته در کد بالا فرض شده است که الگوریتم به صورت هم‌زمان[۱۱] اجرا می‌شود. به این دلیل که یک تعداد راند داریم و همه‌ی راس‌های شبکه با هم هماهنگ هستند ولی در عمل این‌طوری نیست. فرض کنید می‌خواهیم کوتاه‌ترین فاصله از s به بقیه راس‌ها را بدست آوریم و مدل غیرهم‌زمان[۱۲] باشد. شبه کد زیر داریم:

```
1  make s active
2  while there is an active node
3      choose an active node v
4          for each vw ∈ E
5              relax(vw)
6              if d(w) has changed
7                  make w active
```

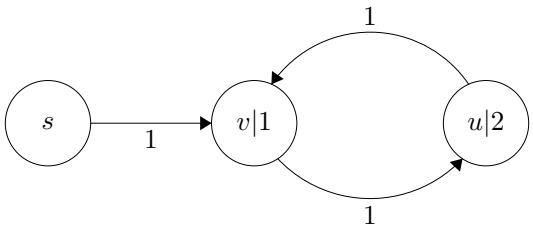
[8] Router
[9] Local
[10] Bellman ford
[11] Synchronous
[12] Asynchronous

منظور از راس فعال در شبه کد بالا راسی است که d آن تغییر کرده است و لازم است یال‌های خروجی آن ریلکس شوند. می‌توان نشان داد که این پیاده‌سازی از الگوریتم هم همگرا می‌شود ولی نکته این است که زمان اجرا ممکن است به شدت زیاد شود. ممکن است حتی زمان اجرا نمایی شود.

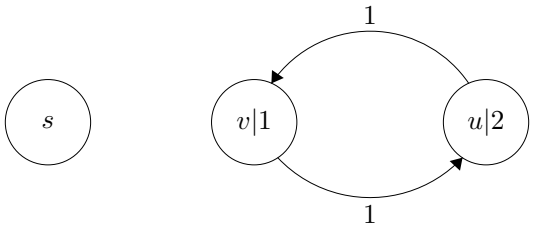
در عمل ممکن است کوتاه‌ترین فاصله هر راس تا تعداد زیادی راس را بخواهیم محاسبه کنیم که کافی است همین الگوریتم بلمن‌فورد گفته‌شده را به صورت موازی برای هر راس اجرا کنیم و d به جای یک عدد یک آرایه خواهید بود که به این روش $distance\ vector\ protocol$ می‌گویند.

در الگوریتم بلمن‌فورد فرض ما بر این بود که وزن یال‌های ثابت می‌مانند یعنی چیزی به عنوان ورودی به ما داده شده است و می‌خواهیم کوتاه‌ترین فاصله‌ها را در گراف مربوطه پیدا کنیم. ولی در سیستم‌های توزیع‌شده ممکن است این اتفاق نیفتد. مثلا اگر وزن هر یال متناظر با میزان تاخیر فرستادن اطلاعات از روی آن لینک باشد ممکن است وزن یال‌ها با زمان تغییر کنند یا ممکن است یک روتر و یا یک لینک کلا قطع شود. در واقع اگر یال قطع‌شده در وسط یک کوتاه‌ترین مسیر نباشد تاثیری ندارد ولی اگر یال قطع شده در مسیر مربوط به کوتاه‌ترین مسیر باشد مشکل به وجود می آید.

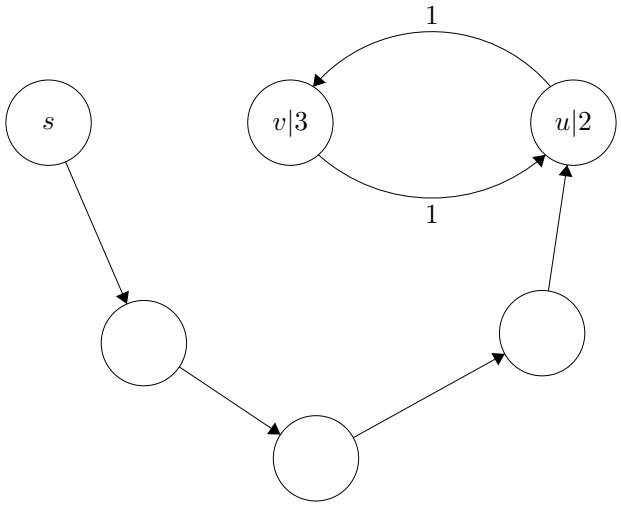
مثال: گراف زیر را در نظر بگیرید و وزن همه یال‌ها را یک در نظر بگیرید. ابتدا فرض کنید همه فاصله‌ها درست حساب شده و فاصله از راس s را روی راس‌ها می‌نویسیم.



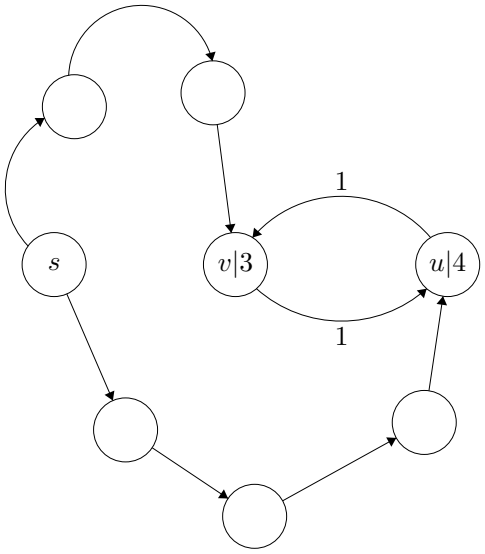
حال فرض کنید یال s به v قطع شود.



حالا راس v چک می‌کند که در کوتاه‌ترین مسیرش تا s یال مربوطه قطع شده است و راس s پدر او بوده است. به همین دلیل سعی می‌کند فاصله خود تا راس s را بروز کند. برای این‌کار به u نگاه می‌کند و می‌بیند که فاصله آن با s برابر ۲ واحد است و فکر می‌کند راس u از مسیری غیر از مسیری که قطع شده است به راس s وصل است به همین دلیل فاصله خود تا راس s را به ۳ تغییر می‌دهد.



بعد دوباره راس u می‌بیند که پدرش در کوتاه‌ترین مسیر مربوطه v بوده است و یال بین آن‌ها خراب نشده است ولی مقدار $d(v)$ به عدد ۳ تغییر کرده است به همین دلیل فکر می‌کند که مسیری از s به v به طول ۳ وجود دارد و به همین دلیل مقدار خود d خود را به ۴ تغییر میدهد.



⊠

همان‌طور که واضح است در گراف مثال گفته‌شده هیچ مسیری بین راس s و راس‌های دیگر وجود ندارد ولی مقدار فاصله‌های آن ها به همین نحو تا بینهایت زیاد میشود. به این مشکل شمردن تا بینهایت[۱۳] می‌گویند. برای دوری از همین مشکلات در عمل به این سمت رفته‌اند که به جای استفاده از $distance\ vector\ protocol$ از $path\ vector\ protocol$ استفاده کنند و الگوریتم‌هایی شبیه دایکسترا استفاده می‌شود. در واقع برای هر راس به جای این‌که فقط کوتاه‌ترین فاصله فعلی تا راس‌های مقصد را نگاه دارند کل مسیر یا حداقل نمایشی از آن مسیر را نگه می‌دارند به طوری که اجازه دهد اگر مشکلی در شبکه پیش آید آپدیت‌ها بهتر انجام شود. مثلا پروتکل BGP که در اینترنت استفاده می‌شود از روش $path\ vector\ protocol$ استفاده می‌کند.

[13] Counting to infinity

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۱۳: جریان در شبکه‌ها (۱) نگارنده: احسان شریفیان

از این جلسه به بعد می‌خواهیم چند جلسه در مورد مبحث مهمی به نام جریان در شبکه‌ها[1] صحبت کنیم که مبحث مهم و پرکاربردی است که در آن می‌توان تکنیک‌های مهم طراحی الگوریتم را مشاهده کرد و هم‌چنین به خودی خود مساله‌ی مهمی است که در بسیاری از کاربردهای تئوری و عملی ظاهر می‌شود و مساله‌های مختلفی را می‌توان به کمک آن حل نمود.

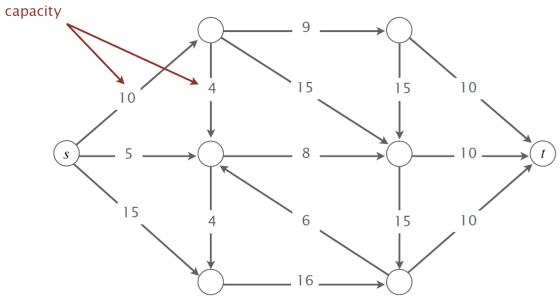
۱ تعاریف

۱.۱ شبکه‌ی جریان

تعریف. *شبکه‌ی جریان*[2] *یک چندتایی* $G = (V, E, s, t, c)$ *است که*

- (V, E) *یک گراف جهت‌دار با راس منبع*[3] s *و راس چاهک*[4] t *است.(به طور ضمنی فرض می‌کنیم که همه‌ی رئوس از راس* s *دسترس‌پذیر هستند)*
- *برای هر یال* $e \in E$ *یک ظرفیت*[5] *نامنفی* $c(e) \geq 0$ *تعریف شده است.*

می‌توان به این گراف به عنوان یک شبکه‌ای از حمل جریان مایعات مثل نفت نگاه کرد که راس نماد یک مرکز توزیع و هر یال نماد یک لوله‌ی انتقال مایع با ظرفیت مشخصی است. یا می‌توان آن را به چشم مدل ترافیک در یک شبکه دید. هم‌چنین فرض ما این است که جریان از یک راس مانند s که راس منبع است تولید شده و به راس t که راس چاهک می‌باشد خاتمه می‌یابد و در بقیه‌ی راس‌ها جریان نه تولید می‌شود و نه از بین می‌رود.



[1] Network Flow
[2] Flow network
[3] source
[4] sink
[5] capacity

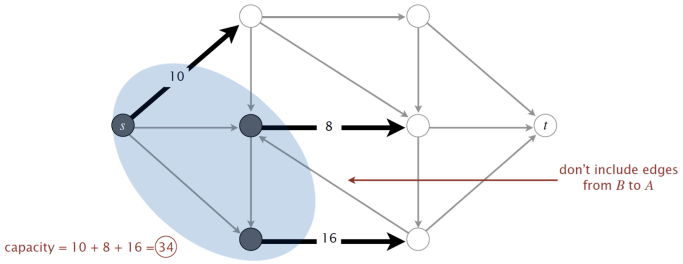
۲.۱ برش و ظرفیت آن

تعریف. *یک st–برش[۶] ، یک افراز از راس‌ها به دو مجموعه‌ی A و B است به صورتی که $s \in A$ و $t \in B$ باشد.*

تعریف. *ظرفیت این برش را برابر با مجموع ظرفیت یال‌های از A به B در نظر می‌گیریم. این تعریف شامل یال‌های از B به A نمی‌شود.*

$$\text{cap}(A, B) = \sum_{\text{e out of A}} c(e) \tag{۱}$$

به عنوان مثال ظرفیت برش نشان داده شده در شکل زیر محاسبه شده است.



مساله‌ی برش کمینه. یافتن برشی که کم‌ترین ظرفیت را داشته باشد.

۳.۱ مساله‌ی جریان بیشینه

ابتدا به صورت دقیق جریان یا st–جریان[۷] در یک شبکه‌ی جریان را تعریف می‌کنیم و سپس مساله‌ی جریان بیشینه را توصیف می‌کنیم.

تعریف. *یک جریان یا st–جریان تابعی مانند f بر روی یال‌هاست که ویژگی‌های زیر را داشته باشد*

- *جریان عبوری از هر یال نامنفی باشد و از ظرفیت آن یال تجاوز نکند. به تعبیری، برای هر یال $e \in E$ داشته باشیم:*

$$0 \leq f(e) \leq c(e) \tag{۲}$$

- *مجموع جریان ورودی و خروجی برای راس‌های غیر منبع و چاهک برابر باشد. به تعبیری، برای هر راس $v \in V - \{s, t\}$ داشته باشیم:*

$$\sum_{\text{e in to v}} f(e) = \sum_{\text{e out of v}} f(e) \tag{۳}$$

تعریف. *مقدار این جریان بر روی شبکه را به صورت مجموع جریان خالص خروجی از راس منبع تعریف می‌کنیم یعنی:*

$$val(f) = \sum_{\text{e out of s}} f(e) - \sum_{\text{e in to s}} f(e) \tag{۴}$$

در گراف‌هایی که در ادامه رسم شده‌اند یک زوج عدد بر روی یال‌ها نوشته شده که عدد سمت چپ بیانگر جریان در آن یال و عدد سمت راست ظرفیت آن یال است.

مساله‌ی جریان بیشینه. یافتن جریانی که بیشترین مقدار را داشته باشد.

[۶] st-cut
[۷] st-flow

۲ الگوریتم فورد–فالکرسون

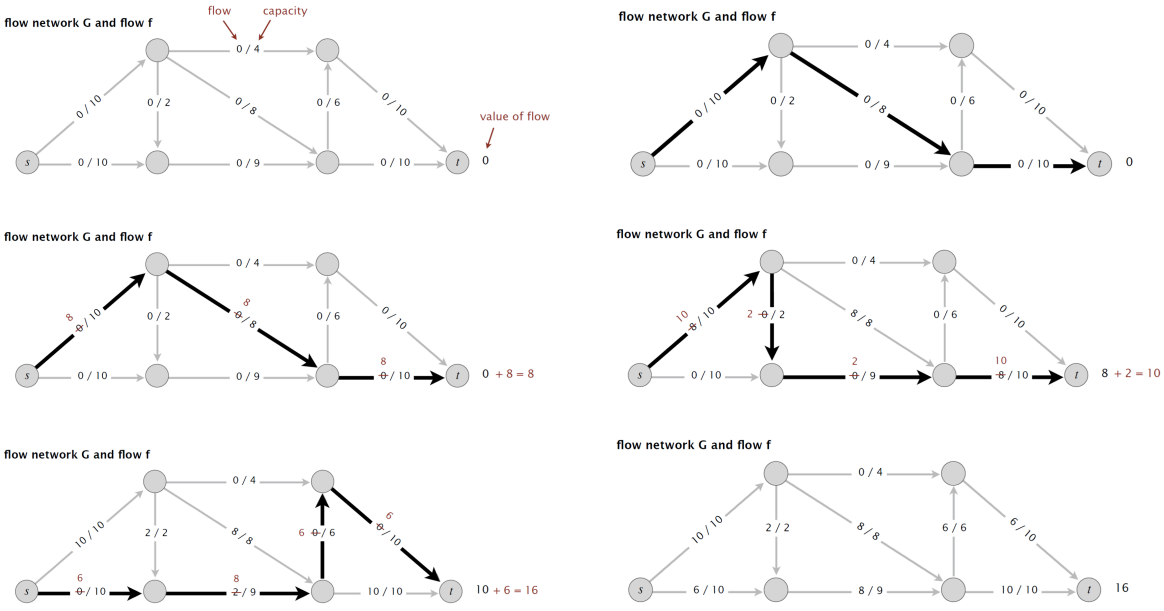
در این قسمت می‌خواهیم روشی برای یافتن پاسخ به مساله‌ی جریان بیشینه را بیابیم که در نهایت منجر به الگوریتم فورد–فالکرسون[۸] می‌شود. در ابتدا روشی که پیش می‌گیریم الگوریتم حریصانه است و سپس با برطرف کردن ایرادات آن سعی می‌کنیم به پاسخی درست برای مساله‌ی جریان بیشینه برسیم که در نهایت ما را به الگوریتم اصلی می‌رساند.

۱.۲ الگوریتم حریصانه

در ابتدا ایده‌ای که به ذهن می‌رسد این است که

- در شروع الگوریتم جریان همه‌ی یال‌های شبکه را صفر بگذاریم.
- یک مسیر از راس s به t مانند P بیابیم که برای هر یال در آن مسیر داشته باشیم $f(e) < c(e)$ (یعنی مسیری را بیابیم که می‌توان جریان همه‌ی یال‌ها را به مقداری بالاتر افزایش داد).
- جریان را در مسیر P تا جای ممکن افزایش دهیم.
- این فرآیند را آنقدر تکرار کنیم که دیگر مسیری با ویژگی گفته شده از s به t وجود نداشته باشد.

به عنوان مثال این فرآیند را رو گرافی که در ادامه تصاویر آن آمده است، پیاده کرده‌ایم. در ابتدا جریان همه‌ی یال‌ها را صفر قرار داده و در هر گام یک مسیر از s به t می‌یابیم که جریان یال‌های آن قابل افزایش باشند. این کار را آنقدر تکرار می‌کنیم تا دیگر نتوانیم چنین مسیری را بیابیم. در نهایت مقدار جریان پیدا شده برابر با $val(f) = 16$ می‌شود اما می‌توان مشاهده کرد که مقدار جریان بیشینه برای این شبکه‌ی جریان، 19 می‌باشد.



[8] Ford-Fulkerson algorithm

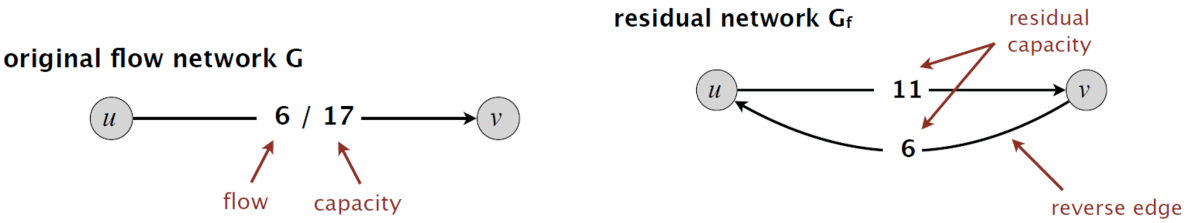
سوال. *چرا روش الگوریتم حریصانه به جواب نمی‌رسد؟*

پاسخ. *چون در طی الگوریتم اگر جریان یک یال افزایش یابد، در ادامه هیچ‌گاه جریان آن کاهش پیدا نمی‌کند. در حالی‌که ممکن است ما انتخاب‌های نامناسبی در طی الگوریتم داشته باشیم و نیاز باشد که دوباره جریان یک یال را کم کنیم تا بتوانیم در کل مقدار جریان بیشتری را داشته باشیم. برای حل این مشکل یک گراف جدید از روی گراف اصلی می‌سازیم و آن را گراف باقی‌مانده می‌نامیم. در ادامه به بررسی ویژگی‌ها و خواص این گراف می‌پردازیم.*

۲.۲ شبکه‌ی باقی‌مانده

فرض کنید شبکه‌ی جریان اصلی G شامل یک یال مانند $e = (u, v) \in E$ با جریان $f(e)$ و ظرفیت $c(e)$ باشد.
در این صورت می‌توان یال معکوس[۹] را به صورت $e^{reverse} = (v, u)$ تعریف کرد که هدف آن این است که جریان را به صورت معکوس بفرستد.
حال می‌توان مفهومی به نام ظرفیت باقی‌مانده[۱۰] برای یال‌ها تعریف کرد که اگر یک یال در گراف اصلی وجود داشت ظرفیت باقی‌مانده‌ی آن به اندازه‌ی مقدار جریانی باشد که می‌توان به این یال افزود تا به ظرفیت اصلی‌اش برسد و اگر یک معکوس یک یال در گراف اصلی وجود داشت ظرفیت باقی‌مانده‌ی آن به اندازه‌ی جریان آن یال اصلی در گراف اصلی باشد، که بیانگر این امر است که می‌توان جریان یال اصلی را به این اندازه کم کرد. به بیان دیگر

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^{reverse} \in E \end{cases}$$



تعریف. *شبکه‌ی باقی‌مانده[۱۱] یک شبکه‌ی جریان $G_f = (V, E_f, s, t, c_f)$ است که در آن E_f مجموعه‌ی همه‌ی یال‌های با ظرفیت باقی‌مانده‌ی مثبت است.*

تعریف. *یک مسیر افزایشی[۱۲]، یک مسیر ساده از s به t در شبکه‌ی باقیمانده‌ی G_f است.*

تعریف. *ظرفیت bottleneck یک مسیر افزایشی مانند P، کمینه "ظرفیت باقی‌مانده"ی یال‌های این مسیر می‌باشد.*

به وسیله‌ی این تعاریف می توان روشی ارائه کرد که یک جریان f بر روی شبکه را به یک جریان مانند f' با مقدار بیشتر تبدیل کند. این کار را به کمک میسرهای افزایشی در شبکه‌ی باقی‌مانده انجام می‌دهیم. تابع AUGMENT یک جریان روی شبکه (f)و ظرفیت یال‌ها (c) و یک مسیر افزایشی P در شبکه‌ی باقی‌مانده G_f را به عنوان ورودی می‌گیرد و جریان یال‌های شبکه را به گونه‌ای تغییر می‌دهد که مقدار جریان در حالت جدید به اندازه‌ی ظرفیت bottleneck مسیر افزایشی بیشتر شود. یعنی

$$f' \leftarrow \text{AUGMENT}(f, c, P)$$

به صورتی که $val(f') = val(f) + \text{bottleneck}(G_f, P)$.

[9] reverse edge
[10] residual capacity
[11] residual network
[12] augmenting path

AUGMENT(f, c, P)

```
1  δ ← bottleneck capacity of augmenting path P
2  for each edge e ∈ P
3      if e ∈ E
4          f(e) ← f(e) + δ
5      else
6          f(e^reverse) ← f(e^reverse) − δ
7  return f
```

۳.۲ الگوریتم فورد–فالکرسون

به وسیله‌ی شبکه باقیمانده می‌توان مسیرهای افزایشی را یافت و با استفاده از تابع AUGMENT می‌توان یک جریان روی شبکه را به جریان با مقدارهای بیشتری تبدیل کرد. پس تا زمانی که بتوان در شبکه باقی‌مانده یک مسیر افزایشی پیدا کرد می‌توان مقدار جریان روی شبکه‌ی اصلی را نیز به اندازه‌ی ظرفیت bottleneck آن مسیر افزایشی بهبود بخشید. الگوریتم فورد–فالکرسون از همین ایده استفاده می‌کند.
این الگوریتم در ابتدا جریان همه‌ی یال‌ها را صفر گذاشته و تا زمانی که بتوان یک مسیر افزایشی در شبکه‌ی باقی‌مانده‌ی متناظر با شبکه‌ی اصلی یافت، جریان در شبکه را اصلاح می‌کند.

FORD-FULKERSON(G)

```
1  for each edge e ∈ E :
2      f(e) = 0
3  G_f ← residual network of G with respect to flow f
4  while (There exists an s → t path P in G_f)
5      f ← AUGMENT(f, c, P)
6      Update G_f
7  return f
```

۳ قضیه‌ی جریان بیشینه – برش کمینه

می‌خواهیم در انتهای این قسمت این قضیه‌ی مهم را ثابت کنیم که بیشینه مقدار ممکن برای یک جریان روی یک شبکه همان کمینه مقدار برای برش‌های آن است. برای اثبات این قضیه چند گام میانی را برمی‌داریم که البته هر کدام به خودی خود نتایج مهمی را بیان می‌کنند.

۱.۳ لم مقدار جریان

لم ۱. فرض کنید f یک جریان دلخواه و (A, B) یک برش دلخواه از شبکه باشند. در این صورت مقدار جریان f که در رابطه‌ی (۴) تعریف شده و برابر مقدار جریان خالص خروجی از راس منبع است، برابر مقدار جریان خالص خروجی از مجموعه رئوس A به مجموعه رئوس B است. یعنی

$$val(f) = \sum_{\text{e out of A}} f(e) - \sum_{\text{e in to A}} f(e) \qquad (۵)$$

به صورت شهودی مشخص است که راس‌هایی میانی(راس‌های غیر منبع) در A با توجه به خاصیت بیان شده در رابطه‌ی (۳) هیچ جریانی تولید نمی‌کنند یا جریان در آن‌ها از بین نمی‌رود فلذا جریان خالص خروجی از مجموعه رئوس A تفاوتی با جریان خالص خروجی از راس منبع نمی‌کند. همین مطلب به بیان دقیق به صورت زیر قابل نوشتن است.

برهان.

$$\begin{aligned} val(f) &\overset{\text{رابطه‌ی (۴)}}{=} \sum_{\text{e out of s}} f(e) - \sum_{\text{e in to s}} f(e) \\ &\overset{\text{توضیح ۱}}{=} \sum_{v \in A} \left(\sum_{\text{e out of v}} f(e) - \sum_{\text{e in to v}} f(e) \right) \\ &\overset{\text{توضیح ۲}}{=} \sum_{\text{e out of A}} f(e) - \sum_{\text{e in to A}} f(e) \qquad \square \end{aligned}$$

توضیح ۱. می‌توان علاوه بر s مجموع فوق را برای راس‌های دیگر در A اضافه کرد چرا که طبق رابطه‌ی (۳) این مجموع برای آن‌ها صفر است.

توضیح ۲. برای هر یال مانند $e = (u, v)$ که $u, v \in A$ جریان عبوری از u به v یک بار با علامت مثبت در ترم $\sum_{\text{e out of v}} f(e)$ ظاهر شده و یک بار با علامت منفی در ترم $\sum_{\text{e in to v}} f(e)$ ظاهر می‌شود(چرا که هر دو راس در مجموعه‌ی A قرار دارند) و از میان این‌ها تنها جریان‌هایی باقی می‌مانند که یک سرشان در A است و یک سرشان خارج A. یال‌های خارج شونده از A با علامت مثبت ظاهر می‌شوند و یال‌های داخل شونده به A با علامت منفی ظاهر می‌شوند.

۲.۳ دوگانی ضعیف

لم۲. دوگانی ضعیف [۱۳] فرض کنید f یک جریان دلخواه و (A, B) یک برش دلخواه از شبکه باشند. در این صورت خواهیم داشت

$$val(f) \leq \text{cap}(A, B) \tag{۶}$$

برهان.

$$\begin{aligned} val(f) &\overset{\text{لم۱}}{=} \sum_{\text{e out of A}} f(e) - \sum_{\text{e in to A}} f(e) \\ &\overset{\text{رابطه‌ی (۲)}}{\leq} \sum_{\text{e out of A}} f(e) \\ &\overset{\text{رابطه‌ی (۲)}}{\leq} \sum_{\text{e out of A}} c(e) \\ &\overset{\text{رابطه‌ی (۱)}}{=} \text{cap}(A, B) \qquad \square \end{aligned}$$

به صورت شهودی نیز می‌توان گفت که اگر جریانی روی شبکه باشد، مقدار آن جریان نمی‌تواند از ظرفیت یک برش بیشتر باشد.

نتیجه ۱. فرض کنید f یک جریان دلخواه و (A, B) یک برش دلخواه از شبکه باشند. اگر $val(f) = \text{cap}(A, B)$ باشد، آنگاه f همان جریان بیشینه و (A, B) همان برش کمینه است.

[13] weak duality

برهان.

- برای هر جریان f' داریم $\text{cap}(A,B) = val(f) \overset{\text{لم۲ (دوگانی ضعیف)}}{\leq} val(f')$ پس f جریان بیشینه است.
- برای هر برش (A',B') داریم $val(f) = \text{cap}(A,B) \overset{\text{لم۲ (دوگانی ضعیف)}}{\geq} \text{cap}(A',B')$ پس (A,B) برش کمینه است.

بنابراین اگر بتوان جریانی را روی شبکه یافت که مقدار آن با ظرفیت یکی از برش‌های شبکه برابر باشد، می‌توان ادعا کرد آن جریان، جریان بیشینه است (و به طور ضمنی ثابت می‌شود که آن برش یافته شده، برش کمینه است).
تا الان ما ثابت کردیم اگر یک جریان روی شبکه داشته باشیم، حتما مقدار آن از ظرفیت هر برش دلخواهی کمتر است. در قضیه‌ی اساسی بعد ثابت می‌کنیم اگر روی شبکه جریان بیشینه داشته باشیم، حتما برشی وجود دارد که ظرفیت آن برابر با مقدار جریان باشد و طبق نتیجه‌ی (۱) این برش همان برش کمینه خواهد بود.

۳.۳ قضیه‌ی جریان بیشینه – برش کمینه

این قضیه از قضایای مهم جریان در شبکه‌ها و بهینه‌سازی ترکیبیاتی می‌باشد.
قضیه ۱.

(۷) | مقدار جریان بیشینه = ظرفیت برش کمینه |

برهان. برای اثبات قضیه از یک گام میانی استفاده می‌کنیم و در اصل نشان می‌دهیم سه گزاره‌ی زیر در مورد هر جریان f روی شبکه با هم معادل‌اند.

(i) یک برش (A,B) وجود دارد که به صورتی که $\text{cap}(A,B) = val(f)$.

(ii) f جریان بیشینه است.

(iii) در شبکه‌ی باقی‌مانده‌ی متناظر با شبکه‌ی اصلی با جریان f هیچ مسیر افزایشی وجود ندارد.

(i) $\implies$ (ii): این همان نتیجه‌ی (۱) است.
(ii) $\implies$ (iii): می‌توان از نقیض گزاره‌ی (iii) نقیض گزاره‌ی (ii) را نتیجه گرفت. یعنی اگر یک مسیر افزایشی متناظر با جریان f در گراف باقی‌مانده وجود داشته باشد، می‌توان جریان f را به وسیله‌ی این مسیر افزایشی و تابع $\text{AUGMENT}(f,c,P)$ طوری بهبود داد که مقدار آن درنهایت افزایش یابد که در این صورت دیگر f جریان بیشینه نبوده است.
(iii) $\implies$ (i): گام اصلی و اساسی در اثبات قضیه همین بخش است. طبق گزاره‌ی (iii) فرض می‌کنیم در گراف باقی‌مانده‌ی متناظر با f هیچ مسیر افزوده‌ای وجود ندارد.

- مجموعه رئوس A را برابر با راس‌های قابل دسترس از s در شبکه باقی‌مانده‌ی G_f در نظر بگیرید.
- طبق تعریف مجموعه‌ی A، داریم $s \in A$.
- طبق خاصیت جریان f که هیچ مسیر افزایشی در شبکه‌ی باقی‌مانده ندارد، داریم $t \notin A$.

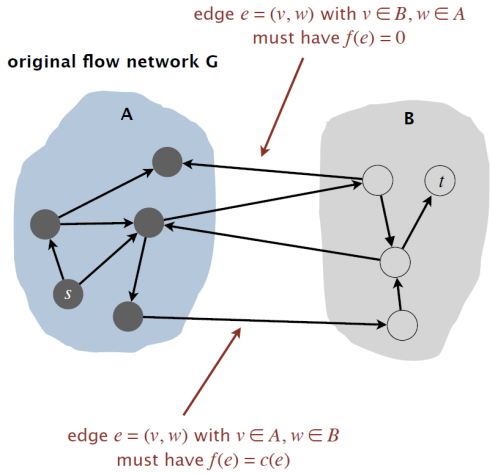
پس A و مجموعه‌ی رئوس مکمل آن B را می‌توان یک برش برای شبکه در نظر گرفت. نشان خواهیم داد (A, B) همان برش کمینه است.

$$val(f) \overset{\text{لم ۱}}{=} \sum_{\text{e out of A}} f(e) - \sum_{\text{e in to A}} f(e)$$
$$\overset{\text{توضیح ۱}}{=} \sum_{\text{e out of A}} c(e) - 0$$
$$\overset{\text{رابطه‌ی (۱)}}{=} \text{cap}(A, B)$$

طبق نتیجه‌ی (۱)، چون ظرفیت برش (A, B) برابر با مقدار جریان f شده است، (A, B) برش کمینه است. □

توضیح ۱. در حقیقت باید نشان دهیم برای هر یال خارج‌شونده از A جریان برابر با ظرفیت آن یال و برای هر یال داخل‌شونده به A جریان برابر صفر است. هر کدام از این دو گزاره نیز به راحتی با برهان خلف ثابت می‌شوند.

فرض کنید یک یال خارج شونده از A مانند $e = (u, v) \in E$ که $u \in A$ و $v \in B$ وجود دارد که در جریان آن برابر با ظرفیت آن نیست. در این صورت در شبکه‌ی باقی‌مانده یال e با ظرفیت باقی‌مانده‌ی مثبت $c(e) - f(e)$ وجود دارد و چون راس u از راس منبع دسترس‌پذیر بود و یال $e = (u, v)$ نیز در شبکه‌ی باقی‌مانده وجود دارد پس، راس v نیز از راس منبع قابل دسترس است و این طبق تعریف A بدین معناست که $v \in A$ ولی این اتفاق با فرض $v \in B$ در تناقض است. به صورت مشابه می‌توان صفر بودن جریان یال‌های واردشونده به A را نیز نشان داد و اثبات قضیه کامل می‌گردد.



قضیه ۲. اگر یک جریان بیشینه‌ی f داده شده باشد، می‌توان یک برش کمینه (A, B) را در $O(m)$ یافت(m تعداد یال‌هاست).

برهان. کافی است مجموعه رئوس A را مجموعه رئوس دسترس‌پذیر از s در شبکه‌ی باقی‌مانده‌ی G_f قرار دهیم. این کار نیز با یک BFS یا DFS در گراف G_f به سادگی قابل انجام است.

۴ آنالیز الگوریتم فورد–فالکرسون

فرض اولیه. فرض می‌کنیم ظرفیت همه‌ی یال‌ها یک عدد صحیح بین 1 تا C باشد.

اصل ۱. در طی الگوریتم فورد–فالکرسون برای هر یال e جریان $f(e)$ و ظرفیت باقی‌مانده $c_f(e)$ عدد صحیح باقی می‌ماند.

برهان. با استقرا روی شماره گام الگوریتم به راحتی قابل مشاهده است. در ابتدای الگوریتم جریان همه‌ی یال‌ها صفر است که عددی صحیح می‌باشد. در گام‌های میانی نیز جریان‌های برخی یال‌ها به اندازه‌ی ظرفیت bottleneck مسیر افزایشی تغییر می‌کنند که طبق فرض استقرا هم جریان‌ها و هم همه‌ی ظرفیت‌های باقی‌مانده از جمله ظرفیت bottleneck مسیر افزایشی صحیح هستند، پس بعد از اعمال تغییرات نیز جریان‌ها و ظرفیت‌های باقی‌مانده صحیح می‌مانند.

برای ادامه فرض کنید تعداد راس‌های شبکه‌ی اصلی برابر n و تعداد یال‌های آن برابر m باشد.

قضیه ۳. الگوریتم فورد–فالکرسون حداکثر پس از $val(f^*)$ گام یافتن مسیرهای افزایشی تمام می‌شود، که در آن f^* یک جریان بیشینه است. هم‌چنین به وضوح داریم $val(f^*) \leq nC$.
برهان. هر بار اجرای حلقه‌ی افزایشی الگوریتم، مقدار جریان را حداقل یک واحد افزایش می‌دهد.

نتیجه ۲. زمان اجرای الگوریتم فورد–فالکرسون $O(mnC)$ است.
برهان. هر گام الگوریتم به اندازه‌ی ساختن شبکه‌ی باقی‌مانده و یک BFS یا DFS در آن برای یافتن مسیر افزایشی طول می‌کشد که از $O(m)$ است. طبق قضیه(۳) چون تعداد کل گام‌های الگوریتم نیز از $O(nC)$ است، پس کل الگوریتم از $O(mnC)$ است که طبق مباحث گذشته زمانی شبه‌چندجمله‌ای می‌باشد.

بدین معنا ممکن است که با انتخاب مسیرهای افزایشی با ظرفیت bottleneck کم، تعداد گام‌های الگوریتم زیاد شود که این اتفاق ما را به این سمت می‌برد که انتخاب مسیرهای افزایشی خود را به نحوی هوشمندانه‌تر انجام دهیم.

قضیه ۴. یک جریان بیشینه با مقدار صحیح حتما وجود دارد.
برهان. این قضیه از قضیه‌ی(۳) و اصل(۱) به راحتی نتیجه می‌شود. الگوریتم فورد–فالکرسون طبق قضیه‌ی(۳) تمام می‌شود و طبق اصل(۱) در طی مراحل آن همه‌ی جریان یال‌ها صحیح می‌ماند، پس در نهایت مقدار جریان بیشینه‌ی یافته شده نهایی نیز صحیح است.

نکته. تا بدین‌جا فرض ما بر صحیح بودن ظرفیت یال‌ها بود. می‌توان نشان داد که در صورت گویا بودن ظرفیت یال‌ها نیز تمام گزاره‌ها و نتایج به دست آمده صحیح می‌باشند. چرا که در روند استدلالی هیچ کدام تغییری به وجود نمی‌آید یا می‌توان این‌گونه تعبیر کرد که با یک تغییر مقیاس می‌توان همه‌ی ظرفیت‌های گویای مساله را در عددی ضرب کرد تا صحیح شوند، سپس الگوریتم فورد–فالکرسون را اجرا کرد و در نهایت همه‌ی مقادیر به دست آمده را بر آن عدد تقسیم کرد.
اما در حالت کلی، اگر ظرفیت یال‌ها، حقیقی باشند و در بین آن‌ها ظرفیت گنگ نیز یافت شود، ممکن است الگوریتم فورد–فالکرسون تمام نشود و حتی به مقدار جریان بیشینه نیز همگرا نشود.

سوال. آیا راه حلی برای اصلاح زمان شبه‌چندجمله‌ای الگوریتم فورد–فالکرسون وجود دارد؟ به تعبیر دیگر آیا می‌توان با روش‌های هوشمندانه‌تری مسیرهای افزایشی را انتخاب کرد تا تعداد اجرای حلقه‌های الگوریتم بهبود یابد و یا روش یافتن مسیرهای افزایشی بهتر شوند؟

پاسخ. سه دیدگاه به حل این سوال برای یافتن مسیرهای افزایشی خوب وجود دارد:

- یافتن مسیر افزایشی با ماکزیمم ظرفیت bottleneck (یعنی می‌خواهیم مسیری از s به t در شبکه‌ی باقی‌مانده بیابیم که وزن مینیمم یال آن ماکزیمم باشد. این کار نیز با الگوریتمی شبیه الگوریتم دایکسترا قابل انجام است اما ما به آن نمی‌پردازیم)
- یافتن مسیر افزایشی با ظرفیت bottleneck به قدر کافی بزرگ(از نصف ماکزیمم ظرفیت ممکن بیشتر باشد)
- یافتن مسیر افزایشی با کم‌ترین تعداد یال‌ها

در ادامه ما روش‌های دوم و سوم را بررسی می‌کنیم.

۵ الگوریتم capacity-scaling

۱.۵ دیدگاه کلی

در این الگوریتم قصد داریم مسیرهای افزایشی با ظرفیت bottleneck به قدر کافی بزرگ را انتخاب کنیم.

- برای این‌کار یک پارامتر مقیاس‌سنج[۱۴] به نام Δ را نگه می‌داریم که قرار است یال‌های با ظرفیت کم‌تر از آن کنار گذاشته شوند.
- شبکه‌ی $G_f(\Delta)$ را بخشی از شبکه‌ی باقی‌مانده در نظر می‌گیریم که فقط شامل یال‌های با ظرفیت باقی‌مانده‌ی بزرگتر از Δ است.
- ظرفیت bottleneck هر مسیر افزایشی در $G_f(\Delta)$ حداقل به اندازه‌ی Δ است.
- الگوریتم فورد–فالکرسون را با مسیرهای افزایشی در $G_f(\Delta)$ اجرا می‌کنیم تا دیگر مسیر افزایشی‌ای در آن یافت نشود. سپس مقدار Δ را نصف می‌کنیم و الگوریتم را تکرار می‌کنیم. این کار را تا آنجا ادامه می‌دهیم که مقدار Δ برابر با یک شود.

۲.۵ شبه‌کد الگوریتم

MATHRMCAPACITY-SCALING(G)

```
1  for each edge e ∈ E :
2      f(e) = 0
3  Δ ← (largest power of 2) ≤ C
4  while (Δ ≥ 1)
5      G_f(Δ) ← Δ-residual network of G with respect to flow f
6      while (There exists an s → t path P in G_f(Δ))
7          f ← AUGMENT(f, c, P)
8          Update G_f(Δ)
9      Δ ← Δ/2
10 return f
```

۳.۵ اثبات درستی الگوریتم

فرض اولیه. در این الگوریتم نیز فرض می‌کنیم که ظرفیت همه‌ی یال‌ها اعدادی صحیح بین 1 و C هستند.

اصل ۲. پارامتر مقیاس‌سنج Δ همواره توانی از ۲ است.

برهان. در ابتدا ما این پارامتر را توانی از ۲ انتخاب کردیم و در طی فازهای مختلف الگوریتم، این پارامتر نصف می‌شود پس همواره توانی از ۲ باقی می‌ماند.

اصل ۳. در طی این الگوریتم برای هر یال e جریان $f(e)$ و ظرفیت باقی‌مانده $c_f(e)$ عدد صحیح باقی می‌ماند.

برهان. مشابه الگوریتم فورد–فالکرسون.

[14] scaling

قضیه ۵. این الگوریتم تمام می‌شود و هنگامی که تمام شود، f به دست آمده یک جریان بیشینه است. (در اینجا فرض می‌کنیم الگوریتم تمام شود.)

برهان. هنگامی که $\Delta = 1$ شود، $G_f(\Delta)$ همان G_f خواهد بود و طبق شبه‌کد، وقتی الگوریتم پایان می‌یابد که مسیر افزایشی در G_f وجود نداشته باشد و طبق گزاره‌های معادل قضیه‌ی(۱)، جریان f یافت شده، جریان بیشینه است.

۴.۵ بررسی زمان اجرای الگوریتم

فرض کنید شبکه‌ی اصلی شامل n راس و شامل m یال است.

لم۳. تعداد دفعات اجرای بیرونی‌ترین حلقه‌ی الگوریتم یا تعداد دفعات تغییر فازهای مقیاس‌سنجی و تغییر پارامتر Δ برابر $1 + \lceil \log_2 C \rceil$ یا $O(\log C)$ است.

برهان. در حالت اولیه $C/2 \leq \Delta \leq C$ است و در هر گام Δ نصف می‌شود.

لم۴. فرض کنید f جریان شبکه در پایان فاز مقیاس‌سنجی با پارامتر Δ باشد. در این صورت $\text{(مقدار جریان بیشینه)} \leq val(f) + m\Delta$

برهان. برای اثبات این لم، ما نشان می‌دهیم برش (A, B) وجود دارد که برای آن رابطه‌ی $\text{cap}(A, B) \leq val(f) + m\Delta$ برقرار است و طبق لم۲ (دوگانی ضعیف) این لم به سادگی نتیجه می‌شود. اثبات ادعای فوق نیز بسیار شبیه به اثبات قضیه‌ی جریان بیشینه – برش کمینه یا همان قضیه‌ی(۱) است.

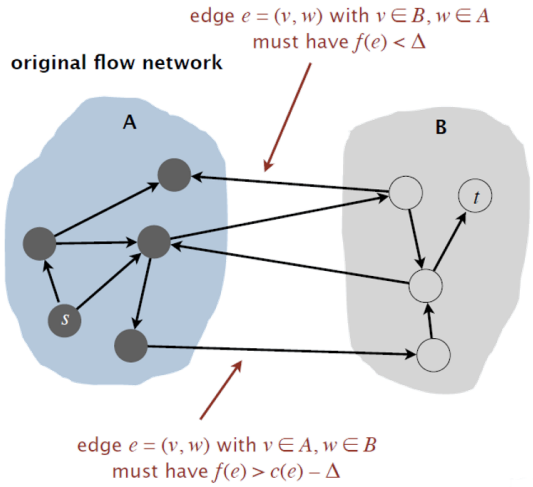
- مجموعه رئوس A را برابر با راس‌های قابل دسترس از s در شبکه باقی‌مانده‌ی $G_f(\Delta)$ در نظر بگیرید.
- طبق تعریف مجموعه‌ی A، داریم $s \in A$.
- طبق خاصیت جریان f که هیچ مسیر افزایشی در شبکه‌ی باقی‌مانده $G_f(\Delta)$ ندارد، داریم $t \notin A$.

پس A و مجموعه‌ی رئوس مکمل آن B را می‌توان یک برش برای شبکه در نظر گرفت. نشان خواهیم داد (A, B) همان برش مد نظر است که برای آن ویژگی $\text{cap}(A, B) \leq val(f) + m\Delta$ را داریم.

$$\begin{aligned} val(f) &\overset{\text{لم۱}}{=} \sum_{\text{e out of A}} f(e) - \sum_{\text{e in to A}} f(e) \\ &\overset{\text{توضیح ۱}}{\geq} \sum_{\text{e out of A}} (c(e) - \Delta) - \sum_{\text{e in to A}} \Delta \\ &= \sum_{\text{e out of A}} c(e) - \sum_{\text{e out of A}} \Delta - \sum_{\text{e in to A}} \Delta \\ &\overset{\text{توضیح ۲}}{\geq} \text{cap}(A, B) - m\Delta \qquad \square \end{aligned}$$

توضیح ۱. برای نتیجه گرفتن این نامساوی، ما از این حقیقت استفاده کردیم که در انتهای فاز یافتن مسیرهای افزایشی در گراف $G_f(\Delta)$ برای هر یال خارج شونده از A باید رابطه‌ی $f(e) > c(e) - \Delta$ و برای هر یال وارد شونده به A باید رابطه‌ی $f(e) < \Delta$ برقرار باشد. اثبات این ادعا نیز به سادگی با برهان خلف امکان‌پذیر است. فرض کنید یک یال خارج‌شونده از A مانند $e = (u, v)$ وجود دارد که $u \in A$ و $v \in B$ و برای آن رابطه‌ی $f(e) > c(e) - \Delta$ برقرار نیست. در این صورت در گراف باقیمانده یالی از u به v وجود خواهد داشت و در این صورت v نیز از s قابل دسترسی است و این با فرض $v \in B$ در تناقض است.

توضیح ۲. حداکثر تعداد یال‌های ورودی به A به علاوه‌ی تعداد یال‌های خروجی از A به اندازه‌ی تعداد کل یال‌ها (m) است.



لم۵. در هر فاز مقیاس‌سنجی (که Δ در آن ثابت است) حداکثر $2m$ بار افزایش مقدار جریان رخ می‌دهد.

برهان.

- فرض کنید f جریان شبکه در شروع فاز با پارامتر Δ باشد (و بنابراین f همان جریان شبکه در پایان فاز با پارامتر 2Δ است).
- طبق لم۴، خواهیم داشت $val(f) + m(2\Delta) \geq$ (مقدار جریان بیشینه).
- هر بار افزایش در فاز با پارامتر Δ مقدار $val(f)$ را به اندازه‌ی Δ زیاد می‌کند.
- طبق دو بند اخیر در این فاز مقدار جریان f حداکثر $2m$ بار می‌تواند زیاد شود تا به مقدار جریان بیشینه برسد.

قضیه۶. زمان اجرای الگوریتم capacity-scaling، از مرتبه‌ی $O(m^2 \log C)$ است.

برهان.

- لم(۳) + لم(۵) $\Leftarrow$ تعداد افزایش‌ها از مرتبه‌ی $O(m \log C)$ است.
- یافتن هر مسیر افزایشی و اعمال آن از مرتبه‌ی $O(m)$ است.

سوال. آیا می‌توان الگوریتمی ارائه داد که زمان اجرای آن برای یافتن جریان بیشینه کاملا مستقل از C باشد؟

پاسخ. بله، چنین الگوریتمی وجود دارد و الگوریتم کوتاه‌ترین مسیر افزایشی که در قسمت بعد بررسی می‌کنیم دارای زمان $O(m^2 n)$ است.

۶ الگوریتم کوتاه‌ترین مسیر افزایشی

در قسمت قبل، الگوریتم فورد–فالکرسون را به وسیله‌ی نگه‌داشتن پارامتر Δ و استفاده از مسیرهای افزایشی با ظرفیت bottleneck بزرگ اصلاح کردیم. هدف در این قسمت اصلاح الگوریتم فورد–فالکرسون با انتخاب مسیرهای با طول کمینه با استفاده از BFS در شبکه‌ی باقی‌مانده است، که منجر به الگوریتم کوتاه‌ترین مسیر افزایشی[15] یا الگوریتم *ادموندز–کارپ* می‌شود.

۱.۶ شبه‌کد الگوریتم

SHORTEST-AUGMENTING-PATH(G)

```
1  for each edge e ∈ E :
2      f(e) = 0
3  G_f ← residual network of G with respect to flow f
4  while (There exists an s → t path P in G_f)
5      P ← BREADTH-FIRST-SEARCH(G_f, s)
6      f ← AUGMENT(f, c, P)
7      Update G_f
8  return f
```

۲.۶ آنالیز الگوریتم

فرض کنید شبکه‌ی جریان دارای n راس و m یال باشد. برای این الگوریتم، ما دو لم را بیان و ثابت می‌کنیم که به کمک آن‌ها می‌توان مرتبه‌ی زمانی کل الگوریتم را محاسبه کرد.

لم۶. در طی این الگوریتم هیچ‌گاه طول کوتاه‌ترین مسیر افزایشی، کاهش نمی‌یابد.

لم۷. بعد از حداکثر m بار افزایش به‌وسیله‌ی کوتاه‌ترین مسیر، طول کوتاه‌ترین مسیر افزایشی اکیداً افزایش می‌یابد.

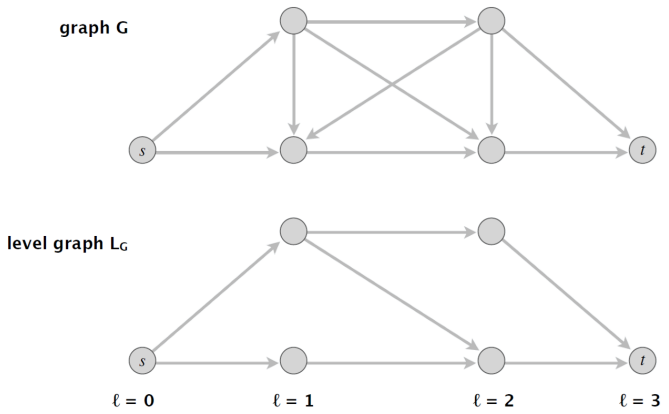
برای اثبات این دو لم یک گراف جدید به نام گراف سطح‌بندی[16] را تعریف می‌کنیم و با توجه به ویژگی‌های آن گراف دو لم اخیر را ثابت می‌کنیم.

تعریف. *فرض کنید گراف جهت‌دار $G = (V, E)$ با راس منبع s داده شده است. گراف سطح‌بندی $L_G = (V, E_G)$ از روی این گراف به صورت زیر تعریف می‌شود.*

- $l(v) =$ *طول کوتاه‌ترین مسیر از s به v است (که همان سطح راس v نسبت به s است هنگامی که از s، BFS را اجرا کنیم).*
- $L_G = (V, E_G)$ *یک زیرگراف از گراف G است که فقط شامل یال‌هایی مانند $(v, w) \in E$ است که $l(w) = l(v) + 1$.*

به عنوان نمونه در شکل زیر گراف سطح‌بندی یک گراف دلخواه ترسیم شده است.

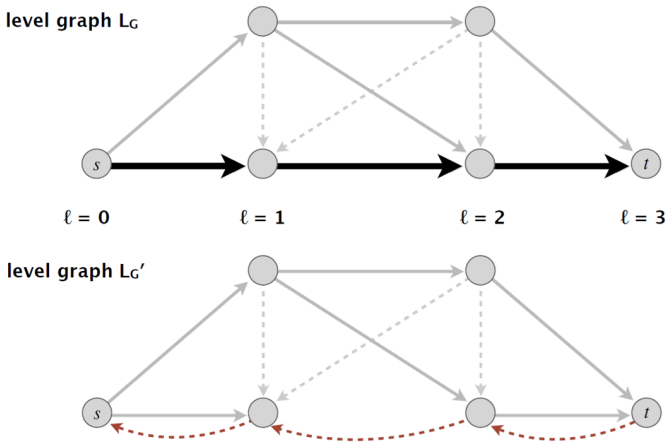
[15] Shortest augmenting path algorithm
[16] level graph



طبق تعریف به وضوح ویژگی کلیدی این گراف آن است که در گراف اصلی (یعنی همان گراف G) P یک کوتاه‌ترین مسیر از s به یک راس دلخواه مانند v است، اگر و تنها اگر P در L_G یک مسیر از s به v باشد.

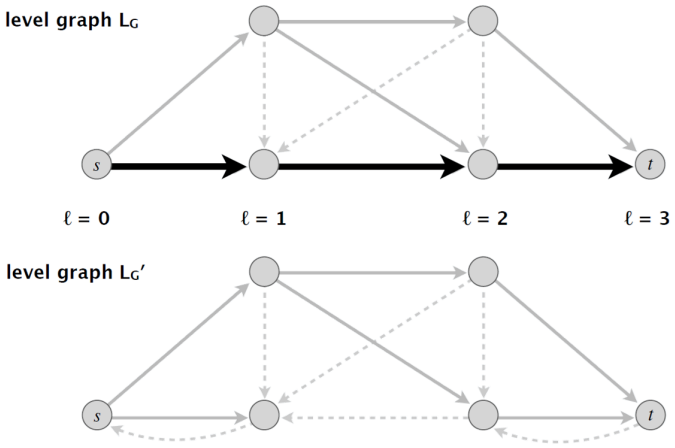
برهان لم ۶.

- فرض کنید f و f' جریان پیش و پس از یک افزایش با کوتاه‌ترین مسیر باشند.
- L_G و $L_{G'}$ را گراف سطح‌بندی متناظر با گراف‌های باقیمانده‌ی G_f و $G_{f'}$ در نظر بگیرید.
- تنها یال‌های از یک سطح به یک سطح پایین‌تر – یال‌های بازگشتی – به گراف $G_{f'}$ اضافه می‌شوند.
- این یال‌ها نمی‌توانند طول کوتاه‌ترین مسیر از s به t را کاهش دهند. در حقیقت اگر بخواهد چنین اتفاقی بیفتد، باید در اثر این افزایش یک یال از سطح A به سطح B که حداقل دو سطح بالاتر از A است، اضافه شود که یال‌های بازگشتی نه تنها این خاصیت را ندارند بلکه یک سطح را به سطح قبلی خود متصل می‌کنند. در نتیجه طول مسیرهای از s به t که از یک یال بازگشتی استفاده می‌کنند قطعا از طول کوتاه‌ترین مسیر قبلی بیشتر است.



برهان لم ۷.

- در اثر یک افزایش، حداقل یک یال از گراف L_G (که توضیح آن در برهان لم ۶ آورده شده است) حذف می‌شود که همان یال‌های با ظرفیت bottleneck کوتاه‌ترین مسیر افزایشی یافت شده است.
- بنابراین یال‌های گراف L_G به با هر افزایش حذف می‌شوند تا جایی که یالی برای L_G باقی نماند و طول کوتاه‌ترین مسیر بین s و t یک واحد افزایش یافته و گراف‌سطح بندی باید دوباره به روزرسانی شود.



قضیه ۷. زمان اجرای الگوریتم کوتاه‌ترین مسیر افزایشی، از مرتبه‌ی $O(m^2n)$ است که یک زمان اجرای چندجمله‌ای است.

برهان.

- لم(۱) + لم(۲) $\Leftarrow$ حداکثر m مسیر افزایشی با طول k وجود دارد.
- حداکثر $n-1$ مسیر با طول‌های مختلف وجود دارد.(چون مسیرهای افزایشی همگی مسیرهای ساده و بدون راس تکراری هستند)
- در نتیجه می‌توان گفت در کل حداکثر mn بار افزایش مقدار جریان رخ می‌دهد.
- هر بار افزایش از طریق کوتاه‌ترین مسیر افزایشی با ابزار BFS قابل انجام است که زمان اجرای آن از $O(m)$ است.
- از دو بند اخیر می‌توان صورت قضیه‌ی (۷) را نتیجه گرفت.

مراجع

[۱] ویدیوی سیزدهمین جلسه‌ی درس که قابل دسترسی از اینجا است.

[2] Network Flow 1 and 2, *Lecture slides by Kevin Wayne* Available at:
http://www.cs.princeton.edu/~wayne/kleinberg-tardos.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۱۴: جریان در شبکه ۲ نگارنده: امین کشیری

جلسه‌ی قبل مبحث جریان در شبکه‌ها [۱] را شروع کردیم و مسائلی مقدماتی مانند فرمول‌بندی مسئله، ارتباط جریان بیشینه [۲] با برش کمینه [۳] و الگوریتم‌های موجود در این زمینه را بررسی کردیم. در این جلسه به بعضی از کاربردهای این مسئله و فرمول‌بندی‌های متفاوتی که با مسئله‌ی اصلی جریان بیشینه معادل هستند می‌پردازیم.

۱ مقدمه

کاربرد اولیه‌ی مسئله‌ی جریان بیشینه و برش کمینه برمی‌گردد به زمانی که آمریکایی‌ها می‌خواستند شبکه‌ی ریلی شوروی که آن را به کشورهای اروپای شرقی وصل می‌کرد را با کمترین تخریب شبکه‌ی ریلی کاملاً از کار بیندازند.

امروزه این مسئله در حیطه‌های بسیار زیادی کاربرد دارد که با تعدادی از آن‌ها امروز آشنا می‌شویم.بعضی از کاربردهای این مسئله در زیر آمده اند:

- Data mining
- Open-pit mining
- Bipartite matching
- Network reliability
- Baseball elimination
- Image segmention
- Network connectivity
- Markov random fields
- Distributed computing
- Security of statistical data
- Egalitarian stable matching

[1] network flow
[2] maximum flow
[3] minimum cut

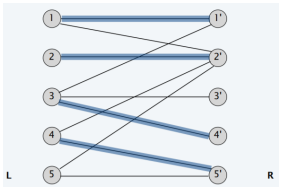
- Network instruction detection
- Multi-camera reconstruction
- Sensor placement for homeland security
- Many, many, more...

۲ تطابق بیشینه در گراف دوبخشی (Bipartite Matching)

تطابق[۴] یعنی پیدا کردن تعدادی از یال‌های یک گراف بدون جهت به طوری که اشتراکی با هم نداشته باشند(یعنی هیچ دو یالی در یک راس اشتراک نداشته باشند). حال مسئله‌ی تطابق بیشینه به دنبال بیشترین تعداد این یال‌هاست. این مسئله نسخه‌ی وزن دار هم دارد یعنی به دنبال بیشترین مجموع وزن یال‌ها باشیم. ما در این درس به دنبال حل مسئله‌ی تطابق در حالت کلی نیستیم و حالت ساده‌تری از این مسئله یعنی تطابق در گراف دو بخشی را بررسی می‌کنیم.

تعریف: **گراف دوبخشی**[۵] به گرافی گفته می‌شود که بتوانیم مجوعه راس‌هابش را به دو مجموعه‌ی L و R افراز کنیم به شرطی که یک سر هر یال در مجموعه‌ی R باشد و سر دیگر آن در مجموعه‌ی L. در واقع در همچین گرافی بین راس های یک مجموعه هیچ یالی وجود ندارد.

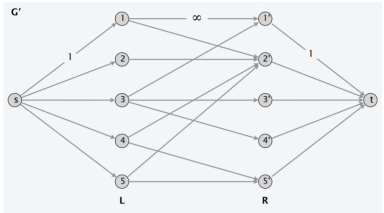
مثال: شکل پایین نشان دهنده‌ی یک تطابق در یک گراف دو بخشی است.



⊠

این مسئله راه‌حل های مستقیمی هم دارد که الگوریتم های پیچیده‌ای هم نیستند، اما ما می‌خواهیم با استفاده از مسئله‌ی جریان بیشینه آن را حل کنیم. پس باید به گونه‌ای گرافی بسازیم که جریان بیشینه در آن یک تطابق با سایز بیشنیه را در گراف اصلی به ما بدهد.

راه‌حل: گراف G' را از روی گراف اولیه (G) به این صورت می‌سازیم که دو راس s و t را به گراف اضافه می‌کنیم. سپس از راس s به تک تک راس‌های L و مشابهاً از تک تک راس‌های R به t یک یال جهت دار با ظرفیت ۱ وصل می‌کنیم. تمام یال‌های گراف قبلی را نیز به یک یال جهت دار از L به R با ضرفیت بی‌نهایت تبدیل می‌کنیم. حال G' یک شبکه است.

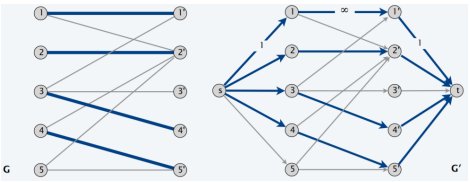


ادعا: جریان بیشینه در این شبکه به ما یک تطابق با اندازه بیشینه را می‌دهد.

[4] matching
[5] bipartite graph

قضیه ۱. *بین تطابق‌های با اندازه k در گراف G و جریان‌های صحیح (به ازای هر یال) با اندازه‌ی k در G' یک تناظر یک به یک وجود دارد.*

اثبات. فرض کنید تطابقی با اندازه k در گراف G داریم. حال جریانی را در نظر بگیرید که از هر کدام از یال‌های متناظر تطابق در گراف G' یک واحد جریان را از s به t می‌فرستد.



برعکس، فرض کنید f یک جریان صحیح در شبکه‌ی G' با اندازه‌ی k باشد. حال مجموعه‌ی M را قرار دهید یال‌هایی که بین L و R هستند و جریان عبوری از آن‌ها ۱ است. حال چون ورودی و خروجی هرکدام از راس‌ها (به جز s و t) در شبکه‌ی G' حداکثر یک است، پس هر راس حداکثر می‌تواند سر یا ته یکی از یال‌های M باشد. از طرفی تعداد این یال‌ها دقیقا k است (با استفاده از لم جریان [۶] در جلسه‌ی قبل و برش $(L \cup \{s\}, R \cup \{t\})$ و با توجه با این نکته که از هر یال حداکثر جریان ۱ می‌گذرد به راحتی دیده می‌شود). پس تناظر یک به یک اثبات می‌شود. □

در مورد زمان اجرای این الگوریتم چه می‌توانیم بگوییم (با فرض $|L| = |R| = n$)؟ می‌توان دید که هربار یک مسیر افزایشی پیدا میکنیم جریان ۱ واحد افزایش می‌یابد، از طرفی جریان نیز حداکثر می‌تواند n باشد، بنابراین زمان اجرای این الگوریتم $O(mn)$ است (دقیقا مشابه استدلال جلسه‌ی قبل).

حال می‌خواهیم به کمک قضیه‌ی جریان بیشینه ـ برش کمینه [۷] یک قضیه ترکیبیاتی را نیز ثابت کنیم.

۳ تطابق کامل در گراف دوبخشی (Perferct Matching in Bipartite Graphs)

یک **تطابق کامل**[۸] به تطابقی گفته می‌شود که هر راس گراف متناظر دقیقا در یک یال از مجوعه‌ی M (مجموعه یال‌های تطابق) آمده باشد. در گراف دوبخشی این به این معناست که تعداد یال‌ها دقیقا برابر $|L|$ یا $|R|$ (دقت کنید که در این حالت $|L| = |R|$) باشد. حال می‌خواهیم بررسی کنیم در این حالت در چه حالتی تطابق کامل داریم.

نمادگذاری: اگر S زیرمجموعه‌ای از راس ها باشد، آنگاه مجموعه‌ی تمام همسایه‌های آن را با $N(S)$ نمایش می‌دهیم. همسایه‌های S نیز یعنی تمام راس هایی در گراف که حداقل به یکی از راس‌های S یال داشته باشد.

شرط لازم: اگر در یک گراف دوبخشی تطابق کامل داشه باشیم آنگاه حتماً به ازای هر زیر مجموعه $S \subseteq L$ داریم: $|N(S)| \geq |S|$

اثبات. هر راس S باید بتواند به همسایه‌ای متفاوت از بقیه راس‌های S متصل شده باشد. □

اما جالب است بدانید این شرط کافی نیز هست و این نکته معروف به قضیه هال است:

[6] flow lemma
[7] max-flow min-cut
[8] perfect matching

قضیه ۲. ***Hall's marriage theorem [Frobenius 1917, Hall 1935]:*** *فرض کنید G یک گراف دو بخشی است که در آن $|L| = |R|$.گراف G یک تطابق کامل دارد اگر و تنها اگر به ازای هر $S \subseteq L$ داشته باشیم: $|N(S)| \geq |S|$*

اثبات. یک طرف قضیه قبلا ثابت شد. برای طرف دیگر فرض کنید $|N(S)| \geq |S|$.

- با برهان خلف، فرض کنید G تطابق کامل نداشته باشد.
- فرض کنید (A, B) یک برش کمینه در G' باشد.
- چون با توجه به تناظر یک به یک بررسی شده اندازه جریان بیشینه باید حتماً از $|L| = n$ کمتر باشد (چون در غیر این صورت تطابق کامل داشتیم) با استفاده از قضیه جریان بیشینه_برش کمینه نتیجه می‌گیریم: $cap(A, B) < |L|$
- تعریف کنید: $L_A = L \cap A$ ، $L_B = L \cap B$ ، $R_A = R \cap A$ ، $R_B = R \cap B$. دقت کنید در این حالت $A = \{s\} \cup L_A \cup R_A$ و B نیز شامل بقیه راس‌ها می‌شود.
- دقت کنید تمام همسایه‌های یک راس از L_A در R_A قرار دارند. زیرا ظرفیت تمام یال‌های بین L و R بی‌نهایت است و اگر یکی از راس‌های L_A به R_B یال داشته باشد آنگاه ظرفیت برش (A, B) بی‌نهایت می‌شود (که خلاف فرض کمینه بودن آن است). در نتیجه: $N(L_A) \subseteq R_A$.
- بنابراین ظرفیت برش (A, B) برابر است با مجموع ظرفیت یال‌های خروجی از s به L_B به‌علاوه‌ی ظرفیت یال های خروجی از R_A به t . در نتیجه داریم: $cap(A, B) = |L_B| + |R_A|$ (دقت کنید ظرفیت تمام این یال‌ها ۱ است).
- از طرفی داریم: $|L_B| + |R_A| = cap(A, B) < |L| = |L_A| + |L_B| \iff |R_A| < |L_A|$. حال با استفاده از قسمت‌های قبل داریم: $|N(L_A)| \leq |R_A| < |L_A|$ که این تناقض است (در صورت قضیه قرار دهید $S = L_A$).

□

در جدول زیر می‌توانید خلاصه‌ای از الگوریتم‌هایی که تا امروز برای حل مسئله‌ی تطابق بیشینه در گراف دوبخشی داده شده است را ببینید (البته بهترین زمان اجرا هنوز مشخص نیست):

year	worst case	technique	discovered by
1955	$O(mn)$	augmenting path	Ford-Fulkerson
1973	$O(mn^{\frac{1}{2}})$	blocking flow	Hopcroft-carp, Karzanov
2004	$O(mn^{2.378})$	fast matrix multiplication	Mucha-Sankowski
2013	$\tilde{O}(m^{\frac{10}{7}})$	electrical flow	Madry

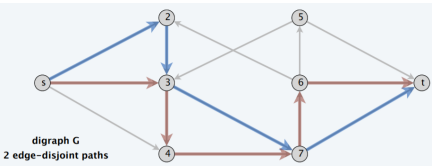
برای پیدا کردن تطابق در حالت کلی هم الگوریتم وجود دارد و همچنین ساختار آن‌ها نیز به خوبی بررسی شده است. اولین الگوریتم در این زمینه را Jack Edmonds ارائه داده است که معروف به الگوریتم غنچه [۹] است. Edmonds از اولین کسانی بود که به اهمیت ارائه الگوریتم‌های چند جمله‌ای پی برد (که آن را در مقاله‌ای در سال 1965 ارائه داد). اما بهترین الگوریتمی که تا امروز برای این مسئله ارائه شده است از $O(mn^{\frac{1}{2}})$ است.

[9] blossom algorithm

۴ مسیرهای یال‌مجزا (Edge Disjoint paths)

دو **مسیر یال‌مجزا**[10] در یک گراف یعنی دو مسیری که هیچ یال مشترکی نداشته باشند (می‌توانند راس مشترک داشته باشند). فرض کنید مانند قسمت‌های قبل دو راس مبدأ [11] و مقصد [12] نیز داشته باشیم (فعلا فرض کنید یال‌ها جهت دار هستند) و می‌خواهیم بیشترین تعداد مسیرهای یال‌مجزا بین مبدأ و مقصد را پیدا کنیم.

مثال: در شکل زیر تعداد مسیرهای یال‌مجزا از s به t برابر با ۲ است.



⊠

حال چگونه این مسئله را به کمک مسئله‌ی جریان بیشینه حل کنیم؟ به هرکدام از یال‌های گراف ظرفیت ۱ نسبت دهید و گراف تبدیل به شبکه می‌شود. حال جریان بیشینه را در این شبکه پیدا کنید. ادعا می‌کنیم این جریان بیشینه متناظر با بیشترین تعداد مسیرهای یال‌مجزا است.

قضیه ۳. *بین k مسیر یال‌مجزا از s به t در گراف G و جریان‌های صحیح با اندازه k در G' تناظری یک به یک وجود دارد.*

اثبات. ($\Rightarrow$)

- فرض کنید P_1 تا P_k مسیرهایی یال‌مجزا باشند.
- به ازای هرکدام از این مسیرها جریان عبوری از تمام یال هایی که عضو این مسیر هستند را ۱ کنید.
- اضافه کردن هر مسیر باعث افزایش جریان به اندازه‌ی ۱ واحد می‌شود و چون مسیرها با هم یال مشترکی ندارند، در نهایت جریان به اندازه‌ی k واحد افزایش می‌یابد (از هیچ یالی نیز جریان بیش از ۱ عبور نخواهد کرد) و جریانی با اندازه‌ی k خواهیم داشت.

($\Leftarrow$)

- حال فرض کنید f یک جریان صحیح با اندازه‌ی k در شبکه‌ی G' باشد. روی اندازه‌ي جریان استقرا میزنیم.
- اگر $k = 0$ آنگاه هیچ مسیر یال‌مجزایی از s به t نمی‌توانیم بیابیم.
- اگر $0 < k$ باشد، حتما یالی خروجی با جریان ۱ از راس s وجود دارد. سر دیگر این یال را u بنامید. با توجه به حفظ جریان (مقدار جریان ورودی و خروجی برای تمام راس‌های درونی برابر است) حتما یالی با جریان ۱ از u نیز خارج میشود. با همین استدلال آن‌قدر ادامه می‌دهیم تا به راس t برسیم (دقت کنید تنها راسی که جریان خروجی آن صفر نیست t است). مسیر طی شده نشان دهنده‌ی یک مسیر از s به t در گراف اصلی است. حال جریان عبوری از تمام یال‌های این مسیر را صفر می‌کنیم (تنها نکته‌ای که وجود دارد این است که اگر حین این مراحل به راسی تکراری رسیدیم، با همان استدلال‌های قبلی میتوانیم آنقدر ادامه دهیم تا به t برسیم. درواقع

[10] edge disjoint
[11] source
[12] sink

اگر همچین دوری وجود داشته باشد، می‌توانیم جریان تمام یال‌های این دور را صفر کنیم و هیچ تغییری در جریان کل عبوری به t رخ نمی‌دهد).

- اکنون جریان کل عبوری از s به t برابر $k-1$ است و طبق فرض استقرا، حتما $k-1$ مسیر یال‌مجزای دیگر می‌توانیم پیدا کنیم (دقت کنید چون جریان عبوری از تمام یال‌های مسیر انتخاب شده را صفر کردیم، در ادامه‌ی مراحل دیگر هیچ‌کدام از این یال‌ها نمی‌توانند استفاده شوند و حتماً بقیه مسیرها یال مشترکی با مسیر ما نخواهند داشت) و این مسیرها به‌علاوه‌ی مسیری که ما پیدا کردیم در مجموع k مسیر یال‌مجزا در گراف G تشکیل می‌دهند.

□

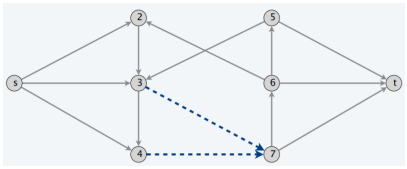
نتیجه: می‌توانیم مسئله‌ی پیدا کردن بیشترین تعداد مسیرهای یال‌مجزا از s به t را به کمک مسئله‌ی جریان بیشینه حل کنیم.

اثبات. با استفاده از قضیه وجود جریان بیشینه‌ی صحیح در جلسه‌ی قبل و قضیه ۳ ، با پیدا کردن یک جریان بیشینه صحیح می‌توانیم بیشترین تعداد مسیرهای یال‌مجزا را بیابیم. □

تنها نکته‌ای که باید به آن دقت کنید این است که شاید عبارت تناظر یک به یک برای قضایای بالا عبارت مناسبی نباشد. بهتر است بگوییم اگر k مسیر یال‌مجزا داشته باشیم حتما یک جریان بیشینه با اندازه‌ی k داریم و برعکس. زیرا به دلیل وجود دورها (که در اثبات نیز به آن اشاره شد) ممکن است متناظر با دو جریان مختلف از s به t در شبکه‌ی G' تنها یک مسیر یال‌مجزا در گراف G یافت شود (در صورت وجود دور، با صفر کردن جریان یال‌های دور، هنوز هم مسیر متناظر با آن در گراف G ثابت می‌ماند).

۵ مسئله‌ی Network Connectivity

یک شبکه به ما داده شده است. می‌خواهیم حداقل تعداد یال‌هایی را حذف کنیم (یال‌های برشی) که دیگر مسیری از s به t وجود نداشته باشد.



برای مثال در شکل بالا یال‌هایی که با خط‌چین نشان داده شده‌اند این خاصیت را دارند. در این زمینه یک قضیه کلاسیک وجود دارد که به صورت زیر است:

قضیه ۴. ***[Menger 1927] Menger's theorem:*** *بیشترین تعداد مسیرهای یال‌مجزا از s به t برابر است با کمترین تعداد یال‌هایی که با حذف آن‌ها s از t جدا [۱۳] می‌شود (یعنی مسیری بین s و t وجود نخواهد داشت).*

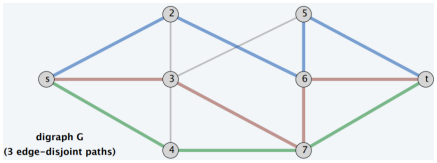
اثبات. • فرض کنید حذف مجموعه‌ی $F \subseteq E$ باعث جدا شدن s از t شود و $|F| = k$. حداقل یکی از یال‌های هر مسیری از s به t عضو F است (زیرا با حذف F تمام مسیرها قطع می‌شوند). پس تعداد کل مسیرهای یال‌مجزا کمترمساوی k است.

[13] disconnected

- حال فرض کنید بیشترین تعداد مسیرهای یال‌مجزا برابر k است. پس جریان بیشینه برابر با k است (طبق ۳). طبق قضیه جریان بیشینه–برش کمینه، حتما برش کمینه‌ای مانند (A, B) با ظرفیت k در گراف وجود دارد. حال F را مجموعه‌ی تمام یال‌های خروجی از A به B در نظر بگیرید. واضح است که با حذف این مجموعه، s از t جدا می‌شود(چون دیگر یالی وجود ندارد که مارا از مجموعه‌ی A به B ببرد). درواقع در قسمت قبل ثابت کردیم تعداد مسیرهای یال‌مجزا کمترمساوی k است و در این قسمت نیز نشان دادیم که تنها با حذف k یال s و t از هم جدا می‌شوند (و همواره تنها کافی است به اندازه‌ی تعداد مسیرهای یال‌مجزا یال از گراف حذف کنیم تا s و t از هم جدا شوند). در نتیجه همواره بیشترین تعداد مسیرهای یال‌مجزا از s به t برابر است با کمترین تعداد یال‌هایی که با حذف آن‌ها s از t جدا می‌شود.

□

اما اگر همین مسئله را در گراف بدون جهت حل کنیم چه؟ به سادگی می‌توان دید که جواب مسئله در این حالت با حالتی که گراف جهت دار است متفاوت است. برای مثال شکل زیر (که حالت جهت دار آن را در قسمت قبل دیده بودیم) ۳ مسیر یال‌مجزا دارد درصورتی که در حالت قبلی ۲ مسیر یال‌مجزا داشت:



برای حل این مسئله، گراف را با تبدیل استانداردی که قبلاً هم دیده‌ایم به یک گراف جهت دار تبدیل می‌کنیم. به این صورت که به ازای هریال در گراف اصلی دو یال در گراف جدید می‌گذاریم. ادعا می‌کنیم در این حالت هم اگر جریانی با اندازه‌ی k از s به t داشته باشم می‌توانیم k مسیر یال‌مجزا در گراف اصلی پیدا کنیم. تنها مشکلی که ممکن است به وجود بیاید این است که از هردو یال متناظر یک یال از گراف بدون جهت، جریانی با اندازه‌ی ناصفر عبور کند (در این جا ۱). در این حالت، بعد از این که جریان بیشینه را در این گراف پیدا کردم، می‌توانم جریان هردو یال را صفر کنم و هیچ تغییری در جریان بیشینه رخ نمی‌دهد و باز هم به سادگی می‌توانیم ببینیم در این حالت از هر دو یال متناظر با یک یال در گراف اصلی (بدون جهت) تنها یکی از آن دو انتخاب شده است و می‌توانیم به اندازه‌ی جریان بیشینه، مسیر یال‌مجزا پیدا کنیم.

قضیه‌ی Menger ورژن‌های دیگری نیز دارد. مثلاً می‌توانیم در مورد مسیرهای راس‌مجزا نیز صحبت کنیم که به طور مشابه ثابت می‌شود (به عنوان تمرین می‌توانید به آن‌ها فکر کنید):

قضیه ۵. *بیشترین تعداد مسیرهای راس‌مجزا از s به t برابر است با کمترین تعداد راس‌هایی که با حذف آن‌ها s از t جدا می‌شود.*

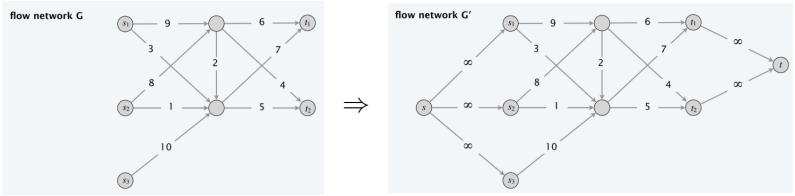
۶ تعمیم‌های مسئله‌ی جریان بیشینه

در این بخش به مسائلی خواهیم پرداخت که شباهت‌های بسیار زیادی به مسئله‌ی جریان بیشینه دارند و درواقع می‌توانند تعمیم‌هایی از آن به شمار روند.

مسئله‌ی وجود بیش از یک مبدأ و مقصد

اولین مسئله‌ای که بررسی می‌کنیم، حالت‌هایی است که بیشتر از یک مبدأ و مقصد داریم و می‌خواهیم بیشترین جریان ممکن که از راس‌های مبدأ شروع می‌شود و به راس‌های مقصد می‌رسد را پیدا کنیم. برای حل این گونه مسائل، دو راس دیگر به گراف اضافه می‌کنیم(اصطلاحاً super source و super sink) و آن‌ها را s و t می‌نامیم. سپس s را به تمام مبدأهای دیگر و تمام مقصدهای دیگر

را به t با ظرفیت بی‌نهایت وصل می‌کنیم. به سادگی دیده می‌شود که یک جریان از s به t در شبکه‌ی جدید متناظر با یک جریان از تمامی مبدأها به مقصدها در شبکه‌ی قبلی است.



مسئله‌ی گردش به همراه عرضه و تقاضا[۱۴]

در این حالت علاوه بر ورودی‌های قبلی، برای هر راس تفاضل جریان ورودی و خروجی نیز داده شده است (که آن را برای راس v با $d(v)$ نشان می‌دهیم). از تفاوت لغوی بین عرضه و تقاضا می‌گذریم و این مقدار را برای تمام راس‌ها تقاضا می‌نامیم (با این نامگذاری تقاضا می‌تواند منفی باشد). در این حالت دیگر مبدأ و مقصدی وجود نداری و همچنین دیگر جریان بیشینه نیز معنی ندارد بلکه تنها به دنبال جریانی هستیم که این شرط‌ها را براورده کند. اولین نکته‌ای که وجود دارد این است که برای این که چنین جریانی وجود داشته باشه، باید مجموع تقاضای تمام راس‌ها صفر شود. زیرا جریان هر یال یک بار با عدد منفی و یک بار با عدد مثبت در این جمع ظاهر می‌شود. معادلاً می‌توانیم بگوییم: $\sum_{v:d(v)>0} d(v) = -\sum_{v:d(v)<0} d(v)$ (اگر عبارت راست را به سمت چپ معادله ببرید به سادگی دیده می‌شود که باید برابر صفر شود زیرا برابر با مجموع تمام تقاضاها می‌شود که طبق استدلال‌های بالا باید برابر صفر شود). در این حالت به همچین جریانی، دیگر جریان نمی‌گوییم و اصطلاحی که متداول است **گردش**[15] است (این مدل مانند شبیه‌سازی تعدادی تولید کننده و مصرف کننده است البته باید به این دقت کنید که همه‌ی تولید کننده‌ها کالای یکسانی را تولید می‌کنند. اگر کالاهای متفاوتی تولید کنند آنگاه مسئله‌ی پیچیده‌تری [۱۶] می‌شود که ما دیگر به آن نمی‌پردازیم). این مسئله را می‌توان به کمک مسائل قبلی حل کرد. باز هم دو راس s و t را به گراف اضافه می‌کنیم، سپس s را به تمام راس‌هایی که تقاضای آن‌ها منفی است با یالی با همان ظرفیت و تمام راس‌هایی که تقاضای آن‌ها مثبت است را با یالی با همان ظرفیت به t وصل می‌کنیم (دقت کنید ظرفیت یال‌ها را مثبت می‌گذاریم). حال به دنبال یک جریان بیشینه از s به t می‌گردیم. اگر مقدار جریان بیشینه‌ی پیدا شده برابر با مجموع ظرفیت یال‌هایی که از s خارج می‌شوند باشد، یعنی توانسته‌ایم در شبکه‌ی اولیه گردشی پیدا کنیم که تمام تقاضاها را ارضا می‌کند. زیرا تنها کافی است s و t را از شبکه حذف کنیم و این دقیقا گردشی خواهد بود که تقاضای تمام راس‌ها را ارضا می‌کند. مشابهاً اگر گردشی وجود داشته باشد که تمام تقاضاها را ارضا کند، می‌توانیم با اضافه کردن s و t جریان بیشینه‌ای در شبکه پیدا کنیم که مقدار آن برابر با مجموع ظرفیت تمام یال‌های خروجی از s است (که خود مساوی با مجموع ظرفیت یال‌های ورودی به t است).

قضیه ۶. ***Integrality theorem****: اگر ظرفیت تمام یال‌ها و تقاضای تمام راس‌ها عدد صحیح باشد و گردشی با این شرایط وجود داشته باشد، آنگاه حتماً یک گردش با مقدار صحیح نیز وجود خواهد داشت.*

اثبات. مستقیماً از تعریف مسئله‌ی جریان بیشینه و قضیه‌ی وجود جریان بیشینه‌ی صحیح برای جریان بیشینه [۱۷] نتیجه می‌شود. □

قضیه ۷. *شرط لازم و کافی برای وجود یک گردش معتبر این است که به ازای هر برش مثل (A, B) داشته باشیم:*

$\sum_{v\in B} d(v) \le cap(A, B)$ *(دقت کنید در این جا برش‌ها دیگر یک $s-t$ برش نیستند و می‌توانند هر راسی را شامل شوند).*

[14] circulation with supplies and demands
[15] circulation
[16] multi commodity flow
[17] integrality theorem for max flow

اثبات. یک طرف قضیه تقریباً واضح است. مثلا فرض کنید مجموع تقاضای مجموعه‌ی B مثبت شود. این تقاضا باید از جایی تأمین شود، و تنها یال‌های ورودی به این مجموعه از A وارد می‌شوند. پس حتماً باید مجموع ظرفیت این یال‌ها از تقاضای B بیشتر باشد. اثبات کافی بودن نیز بسیار شبیه قضایای قبل انجام می‌شود. فرض کنید این گردش وجود ندارد، و در شبکه‌ای که با اضافه کردن s و t ساخته بودیم، یک برش کمینه را در نظر بگیرید، و نشان دهید برشی وجود دارد که شرط بالا را نقض کرده باشد. □

پیچیدگی‌های دیگری را نیز می‌توانیم به این مسئله اضافه کنیم. مثلاً علاوه بر ظرفیت برای یال‌ها (منظور از ظرفیت حداکثر جریان عبوری است)، مقدار کمینه‌ای نیز برای جریان عبوری از آن‌ها نیز قرار دهیم. می‌خواهیم این مسئله را با ترفندی به مسئله‌ی قبلی تبدیل کنیم. برای مثال، فرض کنید یالی داریم که ظرفیت آن بین ۲ و ۹ است. فرض می‌کنیم دو واحد جریان به صورت پیش‌فرض از این یال گذشته است، و به جای این یال یک یال جدید با ظرفیت ۷ می‌گذاریم. سپس با این مقدار تقاضای سر و ته یال را تغییر می‌دهیم. در واقع تقاضای راس سر یال را ۲ واحد افزایش و تقاضای راس ته آن را ۲ واحد کاهش می‌دهیم. حال اگر با این شرایط بتوانیم در شبکه‌ی جدید یک گردش پیدا کنیم که تقاضای شبکه را ارضا کند، می‌توانیم حد پایین ظرفیت یال‌ها را به آن‌ها اضافه کنیم و این کار گردشی به ما می‌دهد که در شبکه‌ی اصلی تمام شرایط را رعایت کرده است.

مسئله‌ی Survey design

حال می‌خواهیم کاربردی از صورت بندی‌های جدیدی که برای مسئله‌ی جریان بیشینه بررسی کردیم را مشاهده کنیم. فرض کنید می‌خواهیم تعدادی نظر سنجی برای محصولاتی از یک شرکت طراحی کنیم.

- n_1 مشتری و n_2 محصول داریم.
- سوالات نظرسنجی افراد متفاوت با هم تفاوت دارد.
- از مشتری i تنها در صورتی می‌توانیم سوالی در مورد محصول j بپرسیم که آن را خریده باشد.
- می‌خواهیم از مشتری i بین c_i تا c'_i سوال بپرسیم.
- همچنین می‌خواهیم از p_j تا p'_j نفر در مورد محصول j بپرسیم.

هدف: آیا می‌توانیم یک نظرسنجی طراحی کنیم که شرایط بالا را ارضا کند؟

حالت خاص مسئله‌ی بالا وقتی $c_i = c'_i = p_j = p'_j = 1$ است همان مسئله‌ی تطابق کامل در گراف دوبخشی است. در حالت کلی، به ازای هر مشتری و محصول یک راس می‌گذاریم. بازهم مانند قبل، دو راس s و t را به گراف اضافه کنیم. از راس s به هر مشتری یک یال با ظرفیت مربوط به تعداد سوالات پرسیده شده از آن فرد وصل می‌کنیم (برای مشتری i م ظرفیت $[c_i, c'_i]$) و به طور مشابه از هر محصول نیز یالی با ظرفیت مربوط به تعداد افرادی که باید از آن‌ها در مورد این محصول می‌پرسیدیم (برای محصول j ظرفیت $[p_j, p'_j]$) به t وصل می‌کنیم. بین هر مشتری و هر محصولی که خریده است نیز یالی با ظرفیت ۱ می‌گذاریم (بین ۰ و ۱). این مسئله خیلی شبیه به مسئله‌ی جریان بیشینه شد اما بهتر است از t به s نیز یالی با ظرفیت بی‌نهایت بگذاریم و تقاضای تمام راس‌ها را نیز صفر بگذاریم. حال اگر در شبکه‌ی حاصل یک گردش معتبر داشته باشیم، معادل است با این که می‌توانیم این نظرسنجی را با شرایط خواسته شده طراحی کنیم.

قضیه ۸. ***قضیه‌ی گردش هافمن:*** *فرض کنید G یک شبکه باشد. شرط لازم و کافی برای وجود یک گردش معتبر این است که به ازای هر زیر مجموعه‌ای از راس‌ها مثل $U \subseteq V$ داشته باشیم: $l(\delta^{in}(U)) \leq c(\delta^{out}(U))$.*

$\delta^{in}(U)$ به معنای تمام یال‌های ورودی به راس‌های زیرمجموعه‌ی U است و برعکس. $l(X)$ نیز به ازای هر زیرمجوعه‌ای مانند X از یال‌ها یعنی مجموع حد پایین تمام یال‌های عضو X (به طور مشابه c یعنی مجموع حد بالای یال‌ها). درواقع یعنی حداقل جریانی که یه یک مجموعه وارد می‌شود (به دلیل حداقل جریان یال‌های ورودی به آن) باید کمتر مساوی حداکثر جریانی باشد که می‌تواند از آن خارج شود. در اینجا این قضیه را اثبات نمی‌کنیم. اما به عنوان راهنمایی برای اثبات آن، جریانی روی یال‌ها پیدا کنید که تا جای ممکن به یک گردش نزدیک باشد. یعنی حداقل و حداکثر جریان یال‌ها را رعایت کرده باشد و | جریان خالص ورودی به v | $\sum_v$ تا جای ممکن کمینه باشد. اگر نشان دهیم جمع بالا برابر صفر است یعنی گردش مورد نظر وجود دارد (دقت کنید تقاضا ها صفر است). حال S و T را به ترتیب تعریف کنید مجموعه راس‌هایی که جریان خالص ورودی به آن‌ها مثبت و منفی است. حال اگر گراف باقی‌مانده [۱۸] را در نظر بگیریم، هیچ مسیری در آن نباید از S به T وجود داشته باشد و از وجود نداشتن این مسیر می‌توانیم یک مجموعه‌ی U پیدا کنیم که این شرط را نقض کرده باشد.

[18] residual network

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۱۵: جریان در شبکه ۳

نگارنده: مریم محمدی یکتا

در جلسات قبل، قضیه‌ی برش کمینه_جریان بیشینه را بیان و ثابت کردیم. در این جلسه چند کاربرد از این قضیه را می‌بینیم.

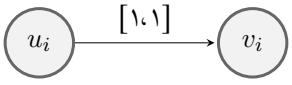
مسأله‌ی اول: زمان‌بندی خطوط پرواز [1]

فرض کنید شما صاحب یک خط هواپیمایی باشید و اطلاعات تعدادی پرواز در اختیارتان باشد. می‌خواهید تعدادی خدمه پرواز انتخاب کنید که به این پروازها سرویس دهند و از طرفی سود شما بیشینه شود(به عبارتی تعداد خدمه‌ی پروازتان کمینه باشد).
اطلاعاتی که از هر پرواز در اختیار دارید شامل مبدا، مقصد، زمان پرواز و زمان فرود است. مثلا پرواز تهران به شیراز که ساعت ۱۰:۰۰ صبح پرواز می‌کند و ساعت ۱۱:۰۰ صبح به مقصد می‌رسد. به علاوه می‌دانید اگر خدمه‌ی پرواز بخواهند در دو پرواز پشت سر هم شرکت کنند، زمان مشخصی برای استراحت نیاز دارند.

در چه صورتی یک تیم پرواز می‌تواند در دو پرواز شرکت کند؟ مثلا اگر یک تیم در پرواز A شرکت کرده باشند و بخواهند در پرواز B شرکت کنند، مقصد پرواز A باید مبدأ پرواز B باشد و زمان کافی برای استراحت تیم پرواز بین پرواز A و B وجود داشته باشد. بنابراین با داشتن اطلاعات مربوط به هر دوی این پروازها به سادگی می‌توان مشخص کرد که آیا این دو پرواز با هم سازگار هستند یا خیر(به این معنی که آیا یک تیم پرواز می‌تواند این پروازها را پشت سر هم انجام دهد یا خیر).

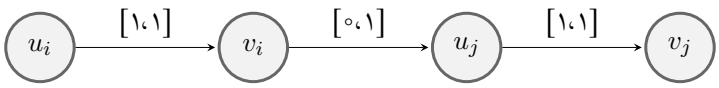
یک شبکه در نظر بگیرید. در این شبکه برای پرواز i که از مبدأ a به مقصد b انجام می‌شود، یک رأس u_i به نمایندگی مبدأ و یک رأس v_i به نمایندگی مقصد در نظر بگیرید و از u_i به v_i یک یال رسم کنید. برای جریان گذرنده از این یال، ظرفیت ۱ و کران پایین [2] ۱ در نظر بگیرید. یعنی:

$$1 = l(e) \leq f(e = u_i v_i) \leq c(e) = 1$$



در این صورت مطمئن خواهیم بود $f(e) = 1$. یعنی جریان حتما از رأس u_i به رأس v_i می‌رود یا به تعبیری یک تیم پرواز از u_i به v_i پرواز می‌کند.

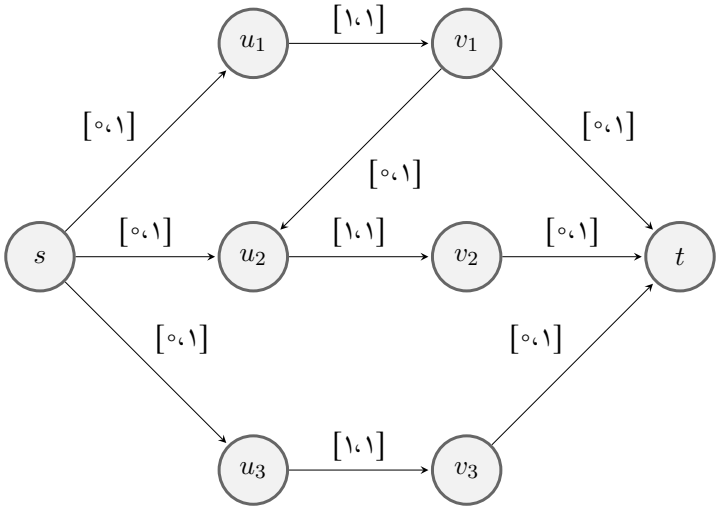
به علاوه در صورتی که دو پرواز i و j با یکدیگر سازگار بودند، از v_i به u_j یک یال با ظرفیت ۱ و کران پایین ۰ رسم کنید.



[1] Airline scheduling
[2] lower bound

اگر جریان گذرنده از یال $v_i \to u_j$ برابر ۱ باشد، به این معنی است که تیم پروازی که پرواز i را سرویس داده، پرواز j را نیز سرویس می‌دهد.

علاوه بر رئوسی که تاکنون در شبکه داشتیم، یک رأس مبدأ (s) و یک رأس مقصد (t) به شبکه اضافه کنید. از s به رأس اول همه‌ی پروازها(یعنی همه‌ی u_i ها) یک یال با ظرفیت ۱ و کران پایین ۰ رسم کنید و از رأس دوم هر پروازی (یعنی از هر v_i ی) به رأس t یالی با ظرفیت ۱ و کران پایین صفر در نظر بگیرید :



فرض کنید هر تیم پرواز در ابتدای روز از رأس s پروازهای خود را شروع می‌کند و می‌خواهد در انتهای روز در رأس t پروازهای خود را تمام کند. بنابراین هر جریانی که از رأس s به رأس t برسد نماینده ی یک تیم پرواز است.

فرض کنید از ابتدا تعداد کل تیم‌های پرواز مشخص شده باشد و برابر c باشد. اگر تعداد کل پروازها k باشد و $c \geq k$، این c خدمه‌ی پرواز برای سرویس دادن به همه‌ی پرواز ها کافی هستند(هر خدمه‌ی پرواز به یکی از پروازها سرویس می‌دهد).

با فرض ثابت بودن c برای هر راس، تقاضا [۳] تعریف می‌کنیم:

$$d(v) = \begin{cases} c & \text{if } v = t \\ -c & \text{if } v = s \\ 0 & o.w. \end{cases}$$

در این شبکه، به دنبال گردشی هستیم که شرایط خواسته شده را داشته باشد. اگر یک گردش معتبر در این شبکه بیابیم، می‌توانیم c مسیر یال مجزا از s به t بیابیم [۱] که هرمسیر نماینده‌ی پروازهایی است که یک خدمه‌ی پرواز سرویس می‌دهد: از چه شهری شروع به پرواز کرده و در کدام شهر به مقصد رسیده و سپس به کدام شهر رفته و ... پس در این صورت می‌توان همه‌ی k پرواز را با c تیم سرویس داد. برعکس اگر c تیم پرواز داشته باشیم که بتوانند همه‌ی این k پرواز را سرویس دهند، مسیر حرکت این c تیم در شبکه‌ی معرفی شده، ایجاد c مسیر یال مجزا می‌کند بنابراین می‌توان یک گردش معتبر در شبکه یافت. پس با مشخص بودن مقدار c می‌توان گفت آیا سرویس دادن به k پرواز ممکن است یا خیر. اما هدف ما پیدا کردن کمترین تعداد تیم‌های پرواز است یعنی مقدار c نامعین است.

می‌دانیم حتما با k تیم پرواز می‌توانیم این‌کار را انجام دهیم. پس $0 < c \leq k$ و می‌توان با جستجوی دودویی [۴] روی مقادیر ممکن

[3] demand
[4] binary search

برای c، حداقل مقدار c را بیابیم. یعنی در ابتدا قرار دهید $c = \frac{k}{2}$ و بررسی کنید آیا با c تیم پرواز می‌توان k پرواز را سرویس داد یا خیر. اگر ممکن بود، بررسی کنیم آیا با $\frac{k}{4}$ تیم این کار ممکن است یا خیر و ...

پس لازم است $O(lgk)$ بار مسأله‌ی گردش معتبر را حل کنیم. شبکه ی ما $O(k)$ راس و $O(k^2)$ یال دارد. پس در $O(k^3)$ می‌توان مشخص کرد آیا گردش معتبری در شبکه وجود دارد یا خیر. بنابراین هر بار حل کردن مسأله‌ی گردش معتبر $O(k^3)$ زمان می‌برد و در کل می‌توان در $O(k^3lgk)$ مقدار کمینه‌ی c را یافت.

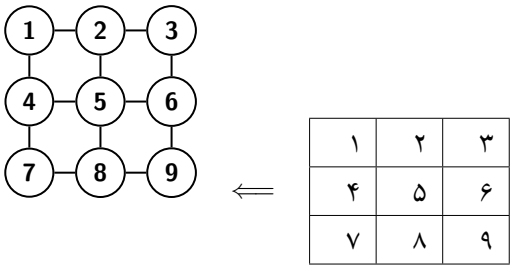
البته مساله‌ی زمان‌بندی خطوط در دنیای واقعی پیچیده‌تر است و فاکتورهای بیشتری در مسأله دخیل هستند.

مسأله‌ی دوم: بخش‌بندی تصویر [5]

تصویری به ما داده شده و می‌خواهیم اشیا متفاوت در تصویر را از هم جدا کنیم.

در این مساله هدف ما صرفا جدا کردن پس‌زمینه [6] از پیش‌زمینه [7] است.

فرض کنید عکسی که داده شده، مجموعه‌ای از پیکسل‌ها است. گرافی در نظر بگیرید که رأس‌های آن همان پیکسل‌های عکس است و یال‌های آن مجموعه‌ی پیکسل‌هایی ست که با هم همسایه هستند. به طور معمول فرض می‌کنیم عکس ما به شکل مستطیلی شامل $m \times n$ پیکسل است و در نتیجه گراف حاصل مستطیل است(البته مساله را برای گراف‌ها در حالت کلی می‌توان حل کرد.)



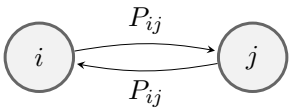
فرض کنید برای پیکسل i دو عدد a_i و b_i داده شده باشد که $a_i \geq 0$ احتمال این است که پیکسل i در پیش زمینه باشد و $b_i \geq 0$ احتمال این است که این پیکسل در پس‌زمینه باشد (به این a_i به چشم جایزه‌ای نگاه کنید که اگر پیکسل i را در پس‌زمینه قرار دهیم، دریافت می‌کنیم و b_i جایزه‌ای است که اگر پیکسل i را در پیش‌زمینه قرار دهیم دریافت می‌کنیم). اگر i, j دو پیکسل همسایه‌ی هم باشند، P_{ij} هزینه‌ای است که اگر یکی از i و j را در پس‌زمینه و دیگری را در پیش‌زمینه قرار دهیم، می‌پردازیم.

هدف پیدا کردن افرازی از رئوس(پیکسل‌ها) به دو مجموعه‌ی A و B (پیکسل‌های پیش‌زمینه و پس‌زمینه) است به طوری که عبارت زیر بیشینه شود:

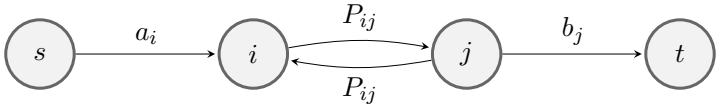
$$S = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}|=1}} P_{ij}$$

گرافی که معرفی کردیم را در نظر بگیرید. اگر $e = ij \in E$ یالی از گراف باشد، مانند شکل این یال را با دو یال جهت‌دار عوض کنید و به هر کدام از یال‌ها، وزن p_{ij} نسبت دهید:

[5] image segmentation
[6] foreground
[7] background



یک رأس مبدأ (s) و یک رأس مقصد (t) نیز به شبکه‌ی حاصل اضافه کنید. از رأس s به هر رأس مانند i یالی با ظرفیت a_i رسم کنید و از هر رأس مانند j به رأس t یالی با ظرفیت b_j اضافه کنید:



بیشینه کردن مقدار S معادل کمینه کردن مقدار زیر است:

$$\sum_{i\in V} a_i + \sum_{j\in V} b_j - S$$
$$= \sum_{i\in V} a_i + \sum_{j\in V} b_j - \sum_{i\in A} a_i - \sum_{j\in B} b_j + \sum_{\substack{(i,j)\in E \\ |A\cap\{i,j\}|=1}} P_{ij}$$
$$= \sum_{i\in B} a_i + \sum_{j\in A} b_j + \sum_{\substack{(i,j)\in E \\ |A\cap\{i,j\}|=1}} P_{ij}$$

اگر در شبکه‌ای که ساختیم، یک $s-t$ برش مانند (A_1, B_1) در نظر بگیریم ، داریم:

$$cap(A_1, B_1) = \sum_{i\in B_1} a_i + \sum_{j\in A_1} b_j + \sum_{\substack{(i,j)\in E \\ |A_1\cap\{i,j\}|=1}} P_{ij}$$

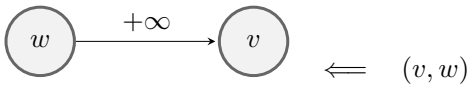
پس به دنبال کمینه کردن مقدار $cap(A_1, B_1)$ هستیم و مسأله معادل پیدا کردن یک برش کمینه در این شبکه است و می‌توانیم با استفاده از قضیه‌ی برش کمینه_جریان بیشینه آن را بیابیم.

مسأله‌ی سوم: انتخاب پروژه [8]

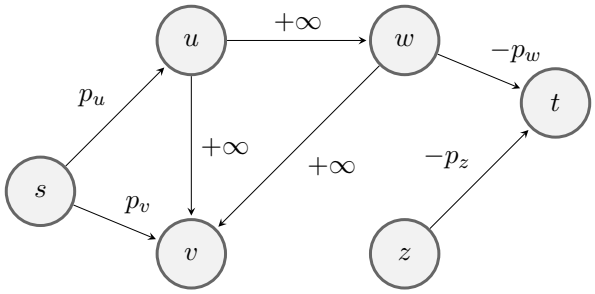
مجموعه‌ی P از پروژه‌ها داده شده است و هر پروژه جایزه [9] ای دارد(احتمالا منفی) و تعدادی رابطه‌ی پیش‌نیازی به صورت زوج پروژه به ما داده شده. مثلا (u, v) یک رابطه‌ی پیش‌نیازی است که نشان می‌دهد پروژه‌ی u پیش‌نیاز پروژه‌ی v است. جایزه‌ی پروژه‌ی v را با p_v نمایش می‌دهیم.
هدف انتخاب تعدادی پروژه است به طوری که جمع جایزه‌ی آن‌ها بیشینه شود.

یک گراف پیش‌نیازی از روی روابط پیش‌نیازی بسازید. یعنی اگر v پیش‌نیاز w باشد، از w به v یک یال رسم کنید و ظرفیت این یال را $+\infty$ قرار دهید:

[8] project selection
[9] revenue



در نهایت یک رأس مبدأ (s) و یک رأس مقصد (t) به گراف اضافه کنید و برای هر پروژه‌ی v که $p_v > 0$ یال $s \to v$ را با ظرفیت p_v رسم کنید و برای هر پروژه‌ی u که $p_u < 0$ یال $u \to t$ را با ظرفیت $-p_u$ به گراف اضافه کنید.



فرض کنید (A, B) یک $s - t$ برش کمینه باشد، فقط پروژه‌های مجموعه‌ی A را انجام دهید. با توجه به بحث جلسه‌ی قبل، چون (A, B) یک برش کمینه است پس هیچ یالی با ظرفیت $+\infty$ از A به B وجود ندارد. پس هر یالی که از A خارج می‌شود دو حالت دارد:

- یالی از s به v که $v \in B$ و $p_v > 0$، یعنی پروژه‌ای با جایزه‌ی مثبت که آن را انجام نداده‌ایم.
- یالی از w به t که $w \in A$ و $p_w < 0$ ، یعنی پروژه‌ای با جایزه‌ی منفی که آن را انجام داده‌ایم.

بنابراین داریم:

$$cap(A, B) = \sum_{\substack{v \in B \\ p_v > 0}} p_v + \sum_{\substack{v \in A \\ p_v < 0}} (-p_v)$$
$$= \sum_{\substack{v \in V \\ p_v > 0}} p_v - \sum_{v \in A} p_v$$

چون $cap(A, B)$ کمینه است و $\sum_{\substack{v \in V \\ p_v > 0}} p_v$ ثابت است بنابراین $\sum_{v \in A} p_v$ بیشینه است و در نتیجه جایزه‌ی پروژه‌های انتخاب شده بیشینه است.
پس مسأله‌ی پروژه‌ها نیز معادل پیدا کردن یک برش کمینه در گراف است.

مسأله‌ی حفر معدن [۱۰]

می‌خواهیم معدنی حفر کنیم و می‌دانیم هر بلوک مانند v از زمین که از معدن استخراج می‌شود ، با توجه به ترکیبی که دارد و عمقی که در آن قرار دارد، سود p_v دارد(ارزش آن بلوک منهای هزینه ی استخراج آن). برای حفاری هر بلوک لازم است ابتدا بلوک‌های بالای آن را استخراج کنیم یعنی استخراج هر بلوک پیش‌نیاز هایی دارد. می‌خواهیم تعدادی بلوک را استخراج کنیم که سودمان بیشینه شود. این مسأله به سادگی قابل تبدیل به مساله‌ی قبل است.

[10] open pit mining

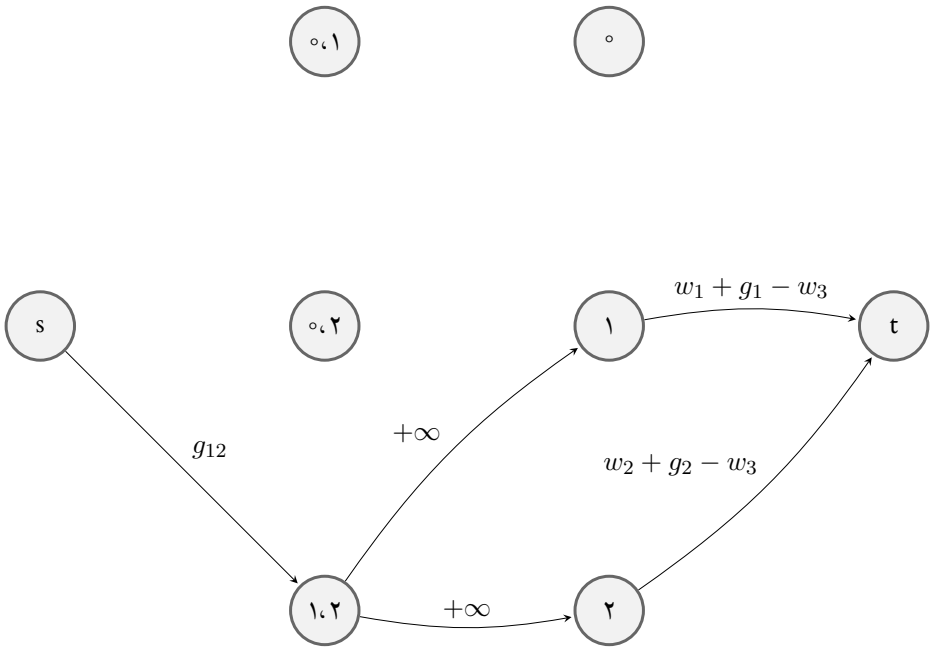
مسأله‌ی چهارم: **baseball elimination**

ورزشی در نظر بگیرید که فقط برد و باخت دارد و تساوی در آن مطرح نیست. فرض کنید یک لیگ برگزار شده و جدول رده‌بندی و تعداد بردهای هر تیم تا این لحظه، تعداد بازی‌های باقی‌مانده‌ی هر تیم و تیم‌هایی که باید با آن‌ها مسابقه بدهد را داریم.
سوال این است که آیا یک تیم خاص امکان قهرمانی دارد یا خیر. یعنی اگر این تیم همه‌ی بازی های پیش روی خود را ببرد، آیا ممکن است برنده‌ی لیگ نیز بشود؟ در صورتی یک تیم برنده‌ی لیگ می‌شود که در انتهای لیگ، تعداد بردهای این تیم بزرگ‌تر یا مساوی تعداد بردهای هر تیم دیگری باشد.

پس مجموعه‌ای از تیم ها مانند S داده شده و تیم $z \in S$ مشخص شده و تیم x تاکنون w_x بازی را برده و می‌دانیم تیم‌های x, y قرار است g_{xy} بار دیگر با یکدیگر مسابقه دهند و تیم x در کل g_x مسابقه‌ی دیگر پیش رو دارد. آیا می‌توان حالتی از برد و باخت در بازی‌های پیش رو در نظر گرفت به طوری که تیم z برنده‌ی لیگ شود؟

یک شبکه با رأس مبدأ s و رأس مقصد t بسازید. برای هر تیم x که $x \neq z$ رأس x را به شبکه اضافه کنید و از این رأس به t یالی با ظرفیت $w_z + g_z - w_x$ رسم کنید (اگر $w_z + g_z - w_x < 0$ ، حتی اگر تیم z همه‌ی بازی‌های پیش روی خود را ببرد باز هم امتیازش از w_x کمتر است پس امکان ندارد بتواند برنده‌ی لیگ شود. پس فرض کنید $w_z + g_z - w_x \geq 0$). به‌علاوه برای هر دو تیم i, j که $g_{ij} > 0$ رأس (i, j) را به شبکه اضافه کنید و از رأس s یالی با ظرفیت g_{ij} به (i, j) رسم کنید و از رأس (i, j) به رئوس i و j یال‌هایی با ظرفیت $+\infty$ اضافه کنید.

مثلا فرض کنید ۴ تیم داریم که با اعداد ۰ تا ۳ شماره‌گذاری شده باشند و می‌خواهیم امکان برنده شدن تیم شماره‌ی ۳ را بررسی کنیم. یال هایی که بین رئوس مربوط به تیم های ۱ و ۲ رسم می‌شوند مانند شکل زیر است:



جریان بیشینه در این شبکه‌ای که برای این n تیم ساخته‌ایم را بیابد. اگر جریان بیشینه f باشد و داشته باشیم $val(f) = \sum_{i,j \in S} g_{ij}$ آنگاه رأس z امکان برنده شدن دارد. با توجه به نحوه‌ی ساختن شبکه، g_{ij} بازی بین تیم های i و j انجام می‌شود و میزان جریانی که از رأس (i, j) به رأس i می‌رود نشان‌دهنده‌ی تعداد بازی‌هایی است که i از j می‌برد و این تیم حداکثر می‌تواند $w_z + g_z - w_i$ بازی را ببرد تا z شانس قهرمانی داشته باشد. بنابراین اگر جریانی پیدا کنیم که همه‌ی یال‌های خروجی از s را اشباع کند، توانسته‌ایم دنباله ای از برد و باخت‌ها در همه‌ی بازی‌های پیش رو مشخص کنیم به طوری که در نهایت z برنده‌ی لیگ شود.

برای هر $T \subseteq S$ تعریف کنید:

$$w(T) := \sum_{i \in T} w_i, \quad g(T) := \sum_{\{x,y\} \subseteq T} g_{xy}$$

قضیه ۱ [11] تیم z هیچ‌گاه نمی‌تواند برنده‌ی لیگ شود اگر و تنها اگر $T^* \subseteq S$ وجود داشته باشد به طوری که

$$w_z + g_z < \frac{w(T^*) + g(T^*)}{|T^*|}$$

$\Longleftarrow$

اگر تنها بازی‌های بین تیم‌های T^* را در نظر بگیریم، در نهایت جمع تعداد بردهای این تیم‌ها از بازی‌های باقی‌مانده بین خودشان برابر $g(T^*)$ است بنابراین جمع بردهای تیم‌های T^* از کل بازی‌ها حداقل برابر $w(T^*) + g(T^*)$ است و یکی از تیم‌های T^* وجود دارد که حداقل

$$\frac{w(T^*) + g(T^*)}{|T^*|}$$

بازی را برده. پس اگر داشته باشیم

$$w_z + g_z < \frac{w(T^*) + g(T^*)}{|T^*|}$$

تعداد بردهای z از تعداد بردهای این تیم اکیدا کمتر است و نمی‌تواند برنده‌ی لیگ باشد.

$\Longrightarrow$

از شبکه‌ای که ساختیم استفاده می‌کنیم. فرض کنید z نمی‌تواند قهرمان لیگ شود. پس در شبکه‌ای که ساختیم، جریان بیشینه، همه‌ی یال‌های خروجی از s را اشباع نمی‌کند. یک $s-t$ برش کمینه مانند (A, B) در نظر بگیرید و فرض کنید:

$$T^* = \{v : v \in A\}$$

یعنی v تیمی باشد که رأس متناظر این تیم در T^* آمده باشد.
دقت کنید رأس (x, y) در A قرار دارد اگر و تنها اگر x و y هر دو در T^* آمده باشند: از رأس (x, y) به رأس‌های x و y یال‌هایی با ظرفیت $+\infty$ داریم، بنابراین طبق آن‌چه در جلسه‌ی قبل بیان کردیم، اگر (x, y) در A قرار داشته باشد، x و y مجبورند در A باشند. اگر x و y در A قرار داشته باشند اما رأس (x, y) در B باشد، می‌توان (x, y) را به A منتقل کرد و $cap(A, B)$ به اندازه‌ی g_{xy} کاهش می‌یابد. پس رأس (x, y) در A قرار دارد اگر و تنها اگر x و y هر دو در A و T^* باشند.
چون $cap(A, B) = val(f)$ و فرض کردیم f همه‌ی یال‌های خروجی از s را اشباع نمی‌کند، پس

$$cap(A, B) = val(f) < g(S - \{z\})$$

یال‌هایی که از A به B می‌رود را می‌توان دو دسته کرد:

- یال هایی که از s به راس هایی که نماینده‌ی دو تیم هستند و در B قرار دارند می‌رود: $e = s \to (u, v), \quad (u, v) \in B$ وزن این یال برابر g_{uv} است. چون (u, v) در B قرار دارد، پس حداقل یکی از u و v در A قرار ندارند، پس جمع ظرفیت این‌گونه یال ها برابر $g(S - \{z\}) - g(T^*)$ است.
- یال‌هایی که از أسری مانند u در A به راس t در B می‌رود. ظرفیت هر یال به این شکل، $w_z + g_z - w_u$ است، پس جمع ظرفیت این‌گونه یال‌ها برابر $\sum_{u \in A}(w_z + g_z - w_u)$ یا همان $\sum_{u \in T^*}(w_z + g_z - w_u)$ است.

[11] Hoffman-Rivlin theorem, 1967

پس داریم:

$$\begin{aligned} cap(A,B) &= g(S-\{z\}) - g(T^*) - \sum_{u\in T^*}(w_z + g_z - w_u) \\ &= g(S-\{z\}) - r(T^*) - w(T^*) + |T^*|(w_z+g_z) \end{aligned}$$

و گفتیم $cap(A,B) < g(S-\{z\})$ بنابراین:

$$g(S-\{z\}) - g(T^*) - w(T^*) + |T^*|(w_z+g_z) < g(S-\{z\})$$

$$\longrightarrow w_z + g_z < \frac{w(T^*)+g(T^*)}{|T^*|}$$

مراجع

[۱] جلسه‌ی چهاردهم درس

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۱۶: تطابق وزن‌دار در گراف دوبخشی، جریان با کمترین هزینه و الگوریتم‌های کارآتر جریان بیشینه

نگارنده: محمد مهدی استاد شریف معمار

در جلسات پیش مسئله‌ی بیشترین تطابق در گراف دوبخشی بدون وزن را دیدیم. در این جلسه می‌خواهیم این مسئله را برای گراف‌های دوبخشی وزن‌دار بررسی کنیم، سپس مسئله‌ی *جریان با کمترین هزینه* را حل کرده و در آخر تعدادی الگوریتم کارآتر برای مسئله‌ی جریان بیشینه ارائه دهیم.

۱ تطابق وزن‌دار در گراف دوبخشی[۱]

نسخه‌های مختلفی از این مسئله وجود دارد. مسئله‌ای که می‌خواهیم به طور مستقیم حل کنیم مسئله‌ی *تطابق کامل با وزن کمینه*[۲] است که در این مسئله فرض را بر این می‌گذاریم که گراف ما دوبخشی‌ست، یال‌های آن وزن دارند، تطابق کامل دارد و به دنبال تطابق کاملی هستیم که کمترین وزن را دارد. توجه کنید بسیاری از مسائل دیگری را که می‌توان مطرح کرد به این مسئله قابل تبدیل هستند. برای مثال مسئله‌ی *تطابق کامل با وزن بیشینه*[۳]. در این مسئله مانند مسئله‌ی قبل فرض را بر این می‌گذاریم که گراف تطابق کامل دارد و به دنبال تطابقی با بیشترین وزن هستیم. برای حل این مسئله می‌توانیم وزن هر یال را قرینه کنیم و مسئله‌ی قبلی را حل کنیم. همچنین مسئله‌ی کلی‌تری مانند *تطابق با وزن بیشینه*[۴]. در این مسئله دیگر فرض وجود تطابق کامل را نداریم. برای حل این مسئله توجه کنید هر یالی که وزن منفی دارد را می‌توان از گراف حذف کرد، زیرا انتخاب آن یال به عنوان یکی از یال‌های تطابق، جمع وزن یال‌های تطابق را کاهش می‌دهد. حال در گراف حاصل می‌توان به ازای هر دو راس u, v که به هم یال ندارند و در یک بخش نیستند (گراف دوبخشی‌ست)، یالی با وزن ۰ بین u, v بکشیم. مسئله‌ی اخیر معادل با مسئله‌ی قبلی‌ست زیرا گراف حاصل تطابق کامل دارد و مسئله‌ی قبل نیز با مسئله‌ی اول قابل حل بود. بنابراین کافی‌ست همان مسئله‌ی *تطابق کامل با وزن کمینه* را حل کنیم.

۱.۱ تطابق کامل با وزن کمینه

فرض کنید گراف دوبخشی وزن‌دار $G = (X, Y)$ با تابع وزن w داده شده است و می‌خواهیم تطابقی کامل مانند M را در گراف پیدا کنیم که جمع وزن یال‌های M مینیمم شود.

ایده‌ی کلی برای حل این مسئله همان الگوریتم فورد_فالکرسون است با کمی تغییر. ابتدا گراف را جهت‌دار می‌کنیم. به جای هر یال $e = uv$ که $u \in X$ و $v \in Y$ یال جهت‌داری از u به v می‌کشیم و ظرفیت آن را ۱ می‌گذاریم . حال دو راس s, t را به گراف اضافه می‌کنیم تا یک شبکه تشکیل شود. از s به هرکدام از رئوس X یال جهت‌داری می‌کشیم و ظرفیت آن را ۱ می‌گذاریم، همچنین از هرکدام از رئوس Y به t یال جهت‌داری می‌کشیم و ظرفیت آن را ۱ می‌گذاریم.

[۱] weighted bipartite matching
[۲] minimum weight perfect matching
[۳] maximum weight perfect matching
[۴] maximum weight matching

در جلسه‌ی ۱۴ دیدیم که با این روش می‌توان مسئله‌ی تطابق بیشینه در گراف بدون وزن را حل کنیم به این‌صورت که در قضیه‌ی ۳ جلسه‌ی ۱۴ ثابت شد که پیدا کردن k مسیر یال مجزا از s به t، متناظر با جریانی با اندازه‌ی k است. پس در آن جلسه برای پیدا کردن بزرگترین تطابق، بیشترین جریان عبوری در گراف را پیدا کردیم، اما اینجا دیگر به دنبال بزرگترین تطابق نیستیم. بلکه به دنبال تطابقی کامل با کمترین وزن در گراف هستیم. برای اینکار از نسخه‌ای از الگوریتم فورد_فالکرسون که در آن هرمرحله به دنبال کوتاه‌ترین مسیر افزایشی از s به t بودیم استفاده می‌کنیم (*الگوریتم ادموندز کارپ*[1]). برای پیدا کردن کوتاه‌ترین مسیر در گراف باقیمانده، طولی برای هر یال تعریف میکنیم. طول تعریف شده برای هر یال به این صورت تعیین می‌شود که اگر یال مورد نظر در گراف اصلی بود، طول آن را برابر وزن یال در گراف اصلی درنظر می‌گیریم و اگر برعکس آن یال در گراف اصلی وجود داشت طول آن یال را قرینه‌ی وزن برعکس آن یال در گراف اصلی تعریف میکنیم. در حقیقت داریم:

$$\forall a \in E(G_f), l(a) = \begin{cases} w(|a|) & a \in E(G) \\ -w(|a|) & \bar{a} \in E(G) \end{cases}$$

منظور از $|a|$، متناظرِ یال a در گراف بدون جهت است. تابع l در بالا برای این تعریف شد که کوتاه‌ترین مسیر از s به t را براساس این تابع پیدا کنیم. همچنین توجه کنید که برای یال های متصل به s و یال‌های متصل به t طول ۰ را درنظر می‌گیریم و تابع بالا فقط برای یال‌های بین X, Y است.

حال به شرح الگوریتم می‌پردازیم:

- در هر مرحله کوتاه‌ترین مسیر افزایشی از s به t را براساس تابع l پیدا می‌کنیم و جریان را در راستای آن افزایش می‌دهیم. این روند را تا وقتی که به یک جریان با مقدار $|X| = n$ نرسیده‌ایم تکرار می‌کنیم.

توجه کنید که چون طول بعضی از یال‌ها می‌تواند منفی باشد برای پیدا کردن کوتاه‌ترین مسیر از الگوریتم بلمن_فورد استفاده می‌کنیم. ادعا این است که تطابق کامل متناظر با جریان بیشینه‌ای که به این روش تعیین می‌شود کمینه است. برای اثبات این موضوع ابتدا چند نکته را ذکر می‌کنیم که به درک موضوع کمک بیشتری می‌کند:

نکته ۱. در هر لحظه، ظرفیت باقیمانده از هر یال در گراف G یا برابر ۱ است یا برابر ۰ است، زیرا ظرفیت تمام یال‌ها در شبکه ۱ است و هر مسیر افزایشی که از s به t پیدا کنیم یال bottlebeck در آن ظرفیت ۱ دارد. پس در کل ظرفیت باقیمانده از هر یال یا برابر ۱ است یا برابر ۰ است.

نکته ۲. در هر لحظه در شبکه‌ی باقیمانده هر یالی که از Y به X است متناظر با یک یال در تطابق انتخاب شده تا آن لحظه است. دلیل این امر در جلسه‌ی ۱۴ توضیح داده شده است. به طور خلاصه می‌توان گفت هر یالی که از Y به X است به معنی این است که جریانی در گذشته از برعکس آن یال رد شده است و بنابر نکته‌ی ۱ ظرفیت آن یال ۰ شده و به جای آن در شبکه‌ی باقیمانده این یال با ظرفیت ۱ اضافه شده است. توجه کنید که پس از هربار برقراری جریان در شبکه، یکی از یال‌های خروجی از s ظرفیتش تکمیل می‌شود (بنابر نکته‌ی ۱) و بنابراین باید در گراف باقیمانده از X یالی به s رسم کنیم اما اینکار را نمی‌کنیم و یال‌هایی مانند این را از گراف باقیمانده حذف می‌کنیم. این کار را برای یال‌هایی که در گراف باقیمانده از t خارج می‌شوند نیز انجام می‌دهیم. دلیل اینکه این یال‌ها را در گراف باقیمانده رسم نمی‌کنیم این است که قضیه‌ی ۱ که در پایین گفته شده است برقرار باشد.

نکته ۳. اگر در لحظه‌ای در گراف باقیمانده یالی از u به v مانند e وجود داشته باشد (u و v دلخواه هستند) و e در مسیر افزایشی که در مرحله‌ی بعد انتخاب می‌شود وجود داشته باشد، پس از دادن جریان به این مسیر و افزایش ۱ واحد به مقدار شار کلی، جهت یال e برعکس می‌شود. دلیل این نکته بنابر نکات ۱ و ۲ واضح است و فقط برای شهود بیشتر گفته شد. درحقیقت این نکته می‌گوید که اگر یالی اکنون از Y به X وجود دارد بنابر نکته‌ی ۲ میدانیم این یال یکی از کاندیدها برای تطابق نهایی‌ست و بنابر نکته‌ی اخیر پس از اجرای یک مرحله از الگوریتم اگر این یال در کوتاه‌ترین مسیر افزایشی باشد جهتش عکس می‌شود، یعنی دیگر یک کاندید برای تطابق نهایی نیست.

[1] Edmonds–Karp algorithm

حال قضیه‌ی ۱ را بیان می‌کنیم:

قضیه ۱. *یک تطابق کامل، کمینه است اگر و فقط اگر گراف باقیمانده‌ی متناظر، دور منفی نداشته باشد. منظور از دور منفی دوری است که جمع طول (همان تابع l) یال‌های آن منفی شود.*

اثبات. فرض کنید که تطابق کامل کمینه مانند M داریم اما گراف باقیمانده‌ی متناظر دور منفی دارد. یال‌های این دور تماما بین رئوس X, Y هستند و بنابراین یکی در میان یال‌ها از Y به X هستند و درنتیجه بنابر نکته‌ی ۲، متناظر این یال‌ها در گراف اصلی، در M هستند. نام این یال‌ها را $a_1, a_2, ..., a_t$ بنامید و بقیه‌ی یال‌ها را $b_1, b_2, ..., b_t$ بنامید. توجه کنید از آنجا که گراف دوبخشی است، دور فرد ندارد. چون این دور یک دور منفی است داریم:

$$l(a_1) + l(a_2) + ... + l(a_t) + l(b_1) + l(b_2) + ... + l(b_t) < 0$$

$$\implies -w(|a_1|) - w(|a_2|) - ... - w(|a_t|) + w(|b_1|) + w(|b_2|) + ... + w(|b_t|) < 0$$

$$\implies -w(|a_1|) - w(|a_2|) - ... - w(|a_t|) < w(|b_1|) + w(|b_2|) + ... + w(|b_t|)$$

پس اگر به جای آنکه یال‌های $|a_1|, |a_2|, ..., |a_t|$ را در تطابق M انتخاب کنیم یال‌های $|b_1|, |b_2|, ..., |b_t|$ را انتخاب کنیم جمع وزن یال‌های تطابق جدید کمتر می‌شود و این یعنی M کمینه نیست که در تناقض با فرض اولیه است.

حال فرض کنید گراف باقیمانده دور منفی ندارد ولی کمینه نیست. نام این تطابق را M بنامید و نام تطابق کامل کمینه‌ی گراف را M' بنامید. حال تفاضل متقارن[1] M و M' را در نظر بگیرید. از آنجا که هر دو تطابق کامل هستند به ازای هر راس مانند $v \in X \cup Y$، یا این راس هیچ یالی ندارد که در $M\Delta M'$ باشد یا دقیقا دو یال در $M\Delta M'$ دارد که یکی در M و دیگری در M' است. بنابراین در $M\Delta M'$ درجه‌ی هر راس ۲ است و درنتیجه $M\Delta M'$ اجتماع تعدادی دور است. حال توجه کنید چون گراف دوبخشی است طول هرکدام از این دورها زوج است. از آنجا که وزن M' کمتر از M است، دوری مانند C در $M\Delta M'$ وجود دارد که جمع وزن یالهایی از C که متعلق به M' هستند کمتر از جمع وزن یال‌هایی از C است که در M هستند. فرض کنید یال‌هایی از C که متعلق به M هستند به ترتیب $a_1, a_2, ..., a_t$ باشند و یال‌هایی که متعلق به M' هستند به ترتیب $b_1, b_2, ..., b_t$ باشند. از آنجا که M تطابقی بود که متناظر با گراف باقیمانده‌ی نهایی بود و همچنین بنابر نکته‌ی ۲، تمام یال‌های $a_1, a_2, ..., a_t$ از Y به X هستند و بنابراین طول آنها قرینه‌ی وزن یال متناظرشان در گراف اصلی است. حال داریم:

$$l(a_1) + l(a_2) + ... + l(a_t) + l(b_1) + l(b_2) + ... + l(b_t)$$

$$= -w(|a_1|) - w(|a_2|) - ... - w(|a_t|) + w(|b_1|) + w(|b_2|) + ... + w(|b_t|)$$

عبارت بالا یک عبارت منفی است زیرا جمع وزن یال‌های M' در C کمتر از جمع وزن یال‌های M در C است. بنابراین این دور یک دور منفی‌ست که در تناقض با فرض اولیه است. پس قضیه اثبات شد.

□

بنابر قضیه‌ی بالا برای اینکه ثابت کنیم الگوریتم درست کار می‌کند کافی‌ست ثابت کنیم در گراف باقیمانده‌ی نهایی دور منفی نداریم. برای این منظور به صورت استقرایی ثابت می‌کنیم هیچ‌گاه در حین الگوریتم دور منفی پیدا نمی‌کنیم و درنتیجه در پایان نیز دور منفی نداریم پس الگوریتم درست کار می‌کند.

طبق قضیه‌ای در جلسات پیشین می‌دانیم یک گراف دور منفی ندارد اگر و فقط اگر یک پتانسیل معتبر بتوان برای راس‌های آن گراف تعریف کرد. پس قضیه‌ی زیر را ثابت می‌کنیم.

[1] symmetric difference

قضیه ۲. *همیشه یک پتانسیل معتبر برای راس‌های گراف وجود دارد. منظور از تابع پتانسیل معتبر، تابعی مانند p روی رئوس گراف است که برای هر یال $e = uv$ داریم*

$$l(e) \geq p(v) - p(u) \tag{۱}$$

اثبات. برای هر راس v، تابع پتانسیل p را برابر کوتاه‌ترین فاصله‌ی s تا v در نظر بگیرید. یعنی داریم $p(v) = \delta(s, v)$. میخواهیم ثابت کنیم این تابع پتانسیل معتبر است. در ابتدای الگوریتم بنابر نامساوی مثلث، رابطه‌ی (۱) درست است. حال فرض کنید تا مرحله‌ای از اجرای الگوریتم رابطه‌ی ۱ برقرار است و اکنون قرار است جریان را از مسیر افزایشی $P = s, a_1, a_2, ..., a_{2k}, t$ بگذرانیم. یعنی P کوتاه‌ترین مسیر افزایشی در گراف باقیمانده است. توجه کنید بنابر نکته‌ی ۱ هرکدام از یال‌های $e_i = a_i a_{i+1}$ $1 \leq i \leq 2k-1$ یا ظرفیت باقیمانده‌شان ۰ است یا ۱. پس از اجرای الگوریتم، ظرفیت باقی‌مانده‌ی هر کدام ۱ واحد در مبنای ۲ تغییر می‌کند. از آنجا که P کوتاه‌ترین مسیر است پس برای هر $1 \leq i \leq 2k-1$ داریم:

$$\delta(s, a_{i+1}) = \delta(s, a_i) + w(e_i) \Longrightarrow p(a_{i+1}) = p(a_i) + l(|e_i|) \tag{۲}$$

یعنی نامساوی رابطه‌ی (۱) دیگر تساوی‌ست. حال بعد از اجرای الگوریتم بنابر نکته‌ی ۳ جهت یال e_i عوض می‌شود و طبق تعریف تابع l، طول آن قرینه می‌شود. حال داریم:

$$(۲) \Longrightarrow -p(a_{i+1}) = -p(a_i) - l(|e_i|) \Longrightarrow -l(|e_i|) = p(a_i) - p(a_{i+1})$$

ولی طبق تعریف l می‌دانیم $-l(|e_i|) = l(|a_{i+1}a_i|)$، پس همچنان نیز تساوی برای یال باقیمانده برقرار است. بنابراین p یک پتانسیل معتبر است.

□

بنابر قضیه‌ی ۲ نتیجه می‌گیریم هیچگاه گراف باقیمانده دور منفی پیدا نمی‌کند و درنتیجه بنابر قضیه‌ی ۱ تطابق کامل متناظر کمینه است و این یعنی الگوریتم درست کار می‌کند.

حال به تحلیل زمانی این الگوریتم می‌پردازیم. توجه کنید بنابر قضیه‌ی ۲ می‌توان یک پتانسیل معتبر برای گراف تعریف کرد پس با توجه به الگوریتم جانسون می‌توان از این تابع پتانسیل برای مثبت کردن طول یال‌ها استفاده کرد و سپس از الگوریتم دایکسترا برای پیدا کردن کوتاه‌ترین مسیر استفاده کرد. چون می‌دانیم گراف تطابق کامل دارد، پس پیدا کردن کوتاه‌ترین مسیر افزایشی دقیقا n بار اتفاق می‌افتد. در نتیجه پیچیدگی زمانی الگوریتم $O(n(m + n\log n))$ است. راجع به پیاده‌سازی این تابع پتانسیل و نحوه‌ی آپدیت آن در حین اجرای الگوریتم حرفی زده نشد اما گفته شد پیاده‌سازی خوبی برای آن وجود دارد. دقت کنید بنابر الگوریتمی که برای پیدا کردن کوتاه‌ترین مسیر استفاده می‌کنیم، زمان اجرای الگوریتم تغییر می‌کند؛ مثلا اگر از الگوریتم بلمن‌ـفورد برای پیدا کردن کوتاه‌ترین مسیر استفاده کنیم، زمان اجرای الگوریتم از $O(n(mn)) = O(mn^2)$ می‌شود.

۲ جریان با کمترین هزینه[1]

در جلسه‌ی ۱۳ مسئله‌ی جریان بیشینه را حل کردیم. حال فرض کنید به ازای عبور جریان از هر یالی ما مقداری هزینه بدهیم و می‌خواهیم کمترین هزینه را بدهیم. به طور دقیق فرض کنید در شبکه‌ی $G = (V, E, s, t, c)$ علاوه بر تابع ظرفیت c، تابع هزینه‌ی $k : E \longrightarrow \mathbb{Q}$

[1] minimum-cost flow

هم داده شده است و از بین تمام جریان‌های بیشینه مانند f ما به دنبال جریانی هستیم که هزینه‌ی f کمینه شود. هزینه‌ی جریان f را به صورت زیر تعریف می‌کنیم:

$$cost(f) = \sum_{e \in E(G)} f(e).k(e)$$

این مسئله بسیار شبیه به مسئله‌ی *تطابق کامل با وزن کمینه* است، به این صورت که تابع وزن در آن مسئله (w) همان تابع هزینه در این مسئله (k) است و دیگر مانند آن مسئله ظرفیت هرکدام از یال‌ها ۱ نیست بلکه ممکن است عددی مخالف ۱ باشد. در حقیقت این مسئله یک تعمیم از مسئله‌ی پیشین است با این تفاوت که گراف ما دیگر لزوما دوبخشی نیست و همچنین ظرفیت یال‌ها نیز لزوما ۱ نیست. ذکر یک نکته ضروری‌ست که بنابر قضیه‌ی ۴ در جلسه‌ی ۱۳ می‌دانیم اگر ظرفیت یال‌ها صحیح باشد حتما یک جریان بیشینه با مقدار صحیح وجود دارد (به طور دقیق‌تر از هر یالی مقدار صحیحی جریان می‌گذرد)، اما در این مسئله دیگر لزوما این اتفاق نمی‌افتد و ممکن است جریان بیشینه‌ای که هزینه‌ی کمینه دارد مقدار اعشاری داشته باشد. ما این حالت کلی را در نظر نمی‌گیریم و فرض می‌کنیم ظرفیت یال‌ها عددی صحیح است و از هرکدام از یال‌ها نیز جریانی با مقدار صحیح می‌گذرد.

نکته‌ی جالب این است که الگوریتمی که برای مسئله‌ی *تطابق کامل با وزن کمینه* به کار بردیم در اینجا نیز کار می‌کند. یعنی الگوریتمی که در اینجا استفاده می‌کنیم به صورت زیر است:

- در هر مرحله کوتاه‌ترین مسیر افزایشی از s به t را براساس تابع l پیدا می‌کنیم و جریان را در راستای آن افزایش می‌دهیم. این روند را تا زمانی که دیگر مسیری افزایشی از s به t وجود نداشته باشد ادامه می‌دهیم.

تابع l مشابه قبل تعریف می‌شود و وزن هر یال نیز برابر هزینه‌ی آن یال در نظر گرفته می‌شود. قضیه‌ی ۱ نیز با صورتی دیگر در اینجا درست است ولی اثباتی که در آنجا ارائه شد دیگر معتبر نیست. صورت جدید قضیه‌ی ۱ به صورت زیر است:

قضیه ۳. *یک جریان بیشینه، کمینه است (هزینه‌اش کمینه است) اگر و فقط اگر گراف باقیمانده‌ی متناظر، دور منفی نداشته باشد.*

همچنین قضیه‌ی ۲ نیز برای این مسئله صادق است و این یعنی همواره پتانسیل معتبر برای این گراف وجود دارد و در نتیجه همواره دور منفی نداریم پس الگوریتم برای این مسئله نیز کار می‌کند.

برای تحلیل زمان اجرای الگوریتم به این نکته دقت کنید که در مسئله‌ی قبل ما می‌دانستیم مقدار جریان بیشینه چقدر است اما در این مسئله دیگر مقدار جریان بیشینه معلوم نیست. اگر این مقدار را C بنامیم پیچیدگی زمانی الگوریتم از $O(C(m + n \log n))$ می‌شود. البته توجه کنید الگوریتم‌هایی نیز برای این مسئله وجود دارند که زمان اجرای آنها چندجمله‌ای است.

۳ الگوریتم‌های کارآتر برای مسئله‌ی جریان بیشینه

در جلسه‌ی ۱۳ بهترین الگوریتمی که برای مسئله‌ی جریان بیشینه بیان کردیم الگوریتم ادموندز کارپ بود که در زمان $O(m^۲n)$ اجرا می‌شد. حال می‌خواهیم تعدادی الگوریتم بهینه‌تر برای این مسئله ارائه دهیم.

۱.۳ الگوریتم دینیک[۱]

در الگوریتم ادموندز کارپ دیدیم که پس از اجرای هر m مرحله از الگوریتم، طول کوتاهترین مسیر افزایشی از s به t یکی زیاد می‌شود. حال به دنبال الگوریتمی هستیم که پس از ۱ مرحله از اجرای الگوریتم، طول کوتاه‌ترین مسیر از s به t یکی زیاد شود. ایده‌ای که برای حل این

[۱] Dinic's algorithm

مسئله استفاده می‌شود توسط شخصی به نام دینیک ارائه شد که در آن به جای اینکه در هر مرحله به دنبال کوتاه‌ترین مسیر از s به t باشیم، به دنبال پیدا کردن یک جریان از s به t در گراف باقیمانده باشیم یا اینکه به دنبال تعدادی مسیر یال مجزا از s به t در گراف باقیمانده باشیم (توجه کنید تعدادی مسیر یال مجزا حالت خاصی از یک جریان از s به t است) و این جریان را یکجا به جریان در گراف اصلی اضافه کنیم. البته باید پیدا کردن این جریان در گراف باقیمانده در زمان کمی انجام شود. همچنین توجه کنید ما به دنبال جریان بیشینه در گراف باقیمانده نیستیم و تنها به دنبال جریانی هستیم که با اضافه کردن آن به جریان در گراف اصلی، طول کوتاه‌ترین مسیر افزایشی از s به t حداقل یکی زیاد شود. به چنین جریانی در گراف باقیمانده blocking flow گفته می‌شود. پس ایده‌ی دینیک به این صورت است که به جای اینکه به دنبال کوتاه‌ترین مسیر افزایشی در گراف باقیمانده باشیم به دنبال یک blocking flow در گراف باقیمانده هستیم. پیاده‌سازی‌های مختلفی طبق این ایده برای مسئله‌ی جریان بیشینه انجام شده است. مثلا یک پیاده‌سازی که توسط اسلیتور و تارجان[۱] انجام شده در زمان $O(mn\log m)$ کار می‌کند و پیاده‌سازی دیگری که توسط کارزانوف[۲] انجام شده در زمان $O(n^۳)$ کار می‌کند.

۲.۳ الگوریتم‌های push-relabel یا preflow-push

در الگوریتم‌هایی مثل فورد_فالکرسون یا دینیک ما یک جریان در نظر می‌گرفتیم و تا زمانی که دیگر یک مسیر افزایشی بین s و t وجود نداشت آن جریان را افزایش می‌دادیم. در الگوریتم‌های push-relabel ما یک جریان بیشینه در نظر می‌گیریم که لزوما خواص یک جریان را ندارد، مثلا جمع جریان‌های ورودی به بعضی راس‌ها ممکن است با جمع جریان‌های خروجی از آن‌ها یکی نباشد، و سعی می‌کنیم آن جریان را به گونه‌ای تغییر دهیم که خواص جریان را پیدا کند و به اصطلاح شدنی[۳] شود. حال به توضیح یکی از این الگوریتم‌ها می‌پردازیم:

پیش جریان[۴]. به تابعی مانند $f : E \longrightarrow \mathbb{Q}$ که به ازای هر یال مانند e داشته باشیم $f(e) \le c(e)$ و جمع جریان‌های ورودی به هر راس به جز s بزرگتر مساوی جمع جریان‌های خروجی از آن راس باشد یعنی شرط

$$\forall v \in V, v \neq s, \sum_{e\, out\, of\, v} f(e) \le \sum_{e\, in\, to\, v} f(e)$$

برقرار باشد، یک پیش‌جریان گوییم.

راس فعال[۵]. به راسی که جمع جریان‌های ورودی به آن بزرگتر از جمع جریان‌های خروجی از آن باشد راس فعال می‌گوییم و اگر این دو مقدار با هم برابر بودند به آن راس غیرفعال گوییم.

الگوریتم، دو تابع f و $p : V \longrightarrow (Q)$ را نگه می‌دارد که f یک پیش جریان است و p را یک label برای رئوس می‌نامیم. ویژگی p این است که برای هر یال uv در گراف باقیمانده باید داشته باشیم

$$p(v) \ge p(u) - ۱ \qquad (۳)$$

در ابتدای الگوریتم قرار می‌دهیم $p(s) = n, \forall v \neq s : p(v) = ۰$ که n مرتبه‌ی گراف است. همچنین برای هر یال خروجی از s مانند e قرار می‌دهیم $f(e) = c(e)$ و گراف باقیمانده را آپدیت می‌کنیم. برای بقیه‌ی یال‌ها مقدار جریان عبوری از آنها را ۰ می‌گذاریم. یعنی در ابتدا ما این فرض را می‌کنیم که در جریان بیشینه از همه‌ی یال‌های خروجی از s به اندازه‌ی ظرفیت آن‌ها جریان رد شده است. الگوریتم به این صورت عمل می‌کند:

- تا زمانی‌که راس فعالی در گراف باقیمانده وجود دارد، یک راس فعال مانند u که $p(u)$ بیشینه است و $u \neq s, u \neq t$ را انتخاب می‌کنیم و تا غیر فعال شدن آن مرحله‌ی زیر را انجام می‌دهیم:

[۱] Sleator and Tarjan
[۲] Karzanov
[۳] feasible
[۴] preflow
[۵] active node

- یک یال خروجی از u مانند uv را که $p(v) = p(u) - ۱$ است در نظر می‌گیریم و یال uv را push می‌کنیم. در صورتی‌که چنین یالی وجود نداشت راس u را relabel می‌کنیم. سپس گراف باقیمانده را آپدیت می‌کنیم.

دو اصطلاح push و relabel را تعریف می‌کنیم. relabel کردن یک راس مانند u یعنی مقدار $p(u)$ را ۱ واحد زیاد کنیم. همچنین push کردن یال uv در صورتی تعریف می‌شود که راس u فعال باشد و به این صورت تعریف می‌شود: اختلاف جمع جریان‌های ورودی به u و جمع جریان‌های خروجی از u را S بنامید ($S = \sum_{e\ into\ v} f(e) - \sum_{e\ out\ of\ v} f(e)$). تابع push، جریان زیر را از uv در گراف باقیمانده عبور می‌دهد:

$$f(uv) = \begin{cases} S & S \le c(uv) \\ c(uv) & else \end{cases}$$

در حقیقت اگر ظرفیت باقیمانده‌ی uv بزرگ‌تر مساوی مقدار S بود، از یال uv جریان S را می‌گذرانیم تا راس u غیر فعال شود. و اگر ظرفیت uv کمتر از S بود، به اندازه‌ی ظرفیت uv از آن جریان می‌گذرانیم تا اختلاف جریان‌های ورودی و خروجی u کمتر شود. برای اینکه ثابت کنیم الگوریتم بالا در زمان خوبی انجام می‌شود، ۳ لم زیر را ثابت می‌کنیم:

لم ۱. *تعداد relabel کردن‌ها حداکثر $۲n^۲$ است.*

اثبات. فرض کنید u یک راس فعال باشد. این یعنی مسیری از s به u در گراف اصلی وجود دارد که مقدار غیر صفری جریان از هر راس در آن مسیر گذشته است، درنتیجه در گراف باقیمانده مسیری از u به s وجود دارد. اما بنابر رابطه‌ی (۳) که برای label ها داریم و اینکه label راس s برابر n است و همچنین اینکه طول هر مسیر در گراف حداکثر $n - ۱$ است نتیجه می‌گیریم label راس u حداکثر برابر $۲n - ۱$ می‌شود. درنتیجه داریم: $p(u) \le ۲n - ۱$ و این یعنی تعداد relabel کردن‌های یک راس حداکثر $۲n - ۱$ است. در نتیجه تعداد کل relabel کردن ها حداکثر $n(۲n - ۱) = ۲n^۲ - n$ است.

□

لم ۲. *تعداد pushهای اشباع‌کننده[۱] حداکثر از $O(nm)$ است. به یک push، اشباع‌کننده می‌گوییم اگر هنگامی که یالی را push کردیم، جریان در آن یال به اندازه‌ی ظرفیت آن یال شود.*

اثبات. یک یال دلخواه مانند uv را درنظر بگیرید و فرض کنید یک push اشباع‌کننده روی آن اتفاق بیافتد یعنی ظرفیت یال uv تکمیل شود، در نتیجه برعکس این یال یعنی یال vu در گراف باقیمانده وجود دارد. برای اینکه دوباره بتوانیم یال uv را push کنیم باید ابتدا جریانی در یال vu push کنیم تا دوباره یال uv در گراف باقیمانده ظاهر شود. اما می‌دانیم یال uv وقتی push می‌شود که $p(u) = p(v) + ۱$ باشد و یال vu وقتی push می‌شود که $p(v) = p(u) + ۱$ باشد و این یعنی در این حین باید label راس v دو واحد زیاد شود، یعنی ۲ بار relabel شود. ولی بنابر اثبات لم ۱ می‌دانیم که راس v حداکثر $۲n - ۱$ بار relabel می‌شود و این یعنی حداکثر n بار می‌توانیم روی یال uv یک push اشباع‌کننده انجام دهیم. پس ثابت شد هر یال $O(n)$ بار push اشباع‌کننده می‌شود و این یعنی در کل تعداد push های اشباع‌کننده از $O(nm)$ است.

□

لم ۳. *تعداد pushهای غیر اشباع‌کننده[۲] حداکثر از $O(n^۳)$ است. به یک push، غیر اشباع‌کننده می‌گوییم اگر هنگامی که یالی را push کردیم، جریان در آن یال همچنان کمتر از ظرفیت آن یال باشد. در حقیقت یک push غیر اشباع‌کننده که روی یک یال مثل uv انجام شود، راس u غیرفعال می‌شود.*

[۱] saturating push
[۲] non-saturating push

اثبات. بنابر لم ۱ میدانیم تعداد تغییرات تابع p (تعداد دفعاتی که label راسی افزایش یافته است) از $O(n^۲)$ است، حال ثابت می‌کنیم بین هر دو تغییر تابع p، حداکثر $O(n)$ بار push غیر اشباع‌کننده انجام می‌شود که از آن نتیجه بشود تعداد کل push های غیر اشباع‌کننده از $O(n^۳)$ است. فرض کنید روی یال uv یک push غیر اشباع‌کننده انجام شود. پس از اینکار راس u غیر فعال می‌شود و در نتیجه طبق الگوریتم باید به دنبال راس فعال دیگری در گراف بگردیم. توجه کنید طبق الگوریتم، ما همیشه راس فعالی را انتخاب می‌کنیم که label آن بیشینه است، پس اگر راس u غیر فعال شد دیگر فعال نمی‌شود زیرا لازمه‌ی فعال شدن u وجود راسی مانند t است که از t به u در گراف باقیمانده یال وجود داشته باشد که این یعنی باید $p(u) = p(t) - ۱$ باشد که در تناقض با بیشینه بودن $p(u)$ است. همچنین توجه کنید بعد از push کردن یال uv ممکن است راس v فعال شود اما شرط اینکه push انجام دهیم این است که $p(v) = p(u) - ۱$ باشد و این یعنی حتی label راس v نیز از u کمتر است. پس اگر راسی غیر فعال شد، از آنجا که بین دو تغییر p هستیم و تابع p تغییر نمی‌کند و همچنین اینکه راس انتخابی ما بیشترین label را دارد، بنابراین دیگر فعال نمی‌شود. در نتیجه به ازای هر push غیر اشباع‌کننده، یک راس غیرفعال می‌شود و آن راس نیز دیگر فعال نمی‌شود. بنابراین حداکثر n بار push غیر اشباع‌کننده بین هر دو تغییر p انجام می‌شود. در نتیجه تعداد کل push های غیر اشباع‌کننده از $O(n^۳)$ است.

□

بنابر لم‌های بالا می‌توان نتیجه گرفت که الگوریتم پایان می‌پذیرد و همچنین بسته به نوع پیاده‌سازی الگوریتم، می‌تواند پیچیدگی زمانی کمی نیز داشته باشد. همچنین توجه کنید از آنجا که $p(s) = n, p(t) = ۰$، پس در انتها مسیر افزایشی از s به t وجود ندارد (به دلیل رابطه‌ی (۳) که برای p داریم و همچنین اینکه طول یک مسیر حداکثر $n - ۱$ است نتیجه می‌شود $p(s) - p(t) \leq n - ۱$ ، پس هیچ مسیر افزایشی از s به t وجود ندارد). همچنین در پایان الگوریتم، f از یک *پیش جریان* به یک *جریان* تبدیل می‌شود زیرا تمام رئوس غیرفعال می‌شوند. در نتیجه بنابر قضیه‌ای در جلسه‌ی ۱۳ چون از s به t هیچ مسیر افزایشی نیست، پس f یک جریان بیشینه است. بنابراین الگوریتم درست کار می‌کند.

این الگوریتم را می‌توان به گونه‌ای پیاده‌سازی کرد که پیچیدگی زمانی آن $O(n^۳)$ باشد. ذکر این نکته نیز مفید است که در عمل برای پیدا کردن بیشینه جریان اغلب از الگوریتم‌های push-relabel استفاده می‌شود.

۴ چند کتاب مفید برای مبحث جریان

این جلسه آخرین جلسه از مبحث جریان بود، برای همین به معرفی دو کتاب مفید در این زمینه می‌پردازیم:

- *نام کتاب:* **Network flows: theory, algorithms, and applications**

  نویسنده: **Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin**

  این کتاب یک کتاب کلاسیک در این زمینه است و الگوریتم‌ها و کاربردهای مختلف از این مسئله را بیان می‌کند.

- *نام کتاب:* **Network flow algorithms**

  نویسنده: **David P. Williamson**

  این کتاب یک کتاب بسیار جدید در این زمینه است (چاپ کتاب برای تابستان سال ۲۰۱۹ است). بیشتر الگوریتمی است و الگوریتم‌هایی که در ۳۰ سال اخیر در این زمینه به دست آمده را پوشش داده است.

مراجع

[۱] جزوه‌ی جلسه‌های ۱۳ و ۱۴ درس.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۱۷: پیچیدگی محاسباتی

نگارنده: محبوبه عنایتی

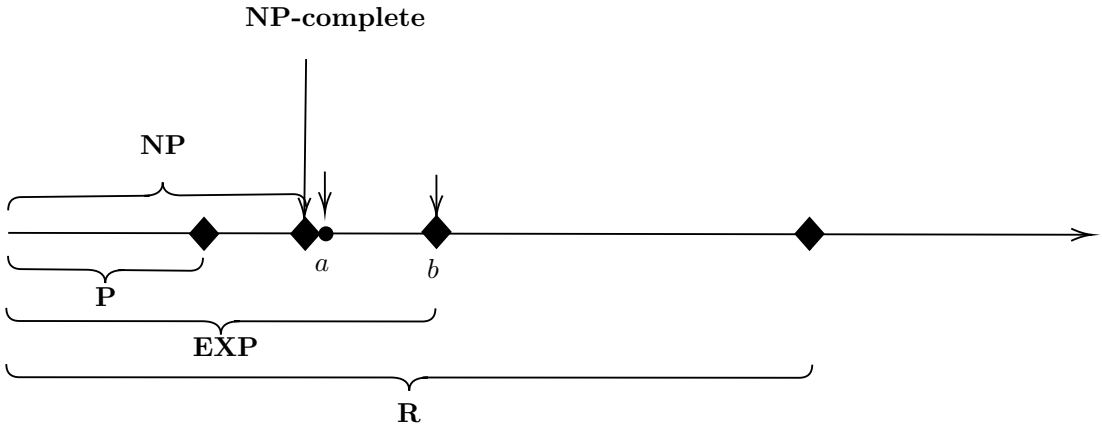
تاکنون سعی بر این بوده است که برای یک سری از مسائل محاسباتی، الگوریتمی کارآ و بهینه با زمان اجرای چندجمله‌ای درجه پایین بر حسب طول ورودی ارائه دهیم. در این حین با چندین تکنیک (مانند برنامه‌ریزی پویا[1]) و مسائل پرکاربرد و مهم آشنا شدیم که از این پس می‌توانیم بسیاری از مسائل دیگر را با آن‌ها مدل کنیم (مانند جریان بیشینه[2]). اما مسائل زیاد دیگری نیز هستند که با چنین الگوریتم‌هایی قابل حل نبوده و یا الگوریتم کارآیی برایشان یافت نشده است. هدف مبحث پیچیدگی محاسباتی[3] بررسی سختی این قبیل مسائل، از نظر وجود و نوع الگوریتم مخصوص به هر یک، و داشتن تخمین خوبی از آن‌ها است.

۱ طبقه‌بندی مسائل

در مبحث پیچیدگی، هنگامی که درباره‌ی مسئله‌ای صحبت می‌کنیم منظور مسئله‌ای از دسته مسائل تصمیم‌گیری[4] و یا به بیان دیگر، مسائلی است که خروجی آن‌ها تنها یکی از موارد «بله» (Yes) یا «خیر» (No) می‌باشد. کلاس‌های پیچیدگی نیز بر همین اساس تعریف می‌شوند.

تعریف ۱. به مجموعه‌هایی از مسائل، که هر مسئله با توجه به میزان سختی‌اش در حداقل یکی از آن‌ها قرار می‌گیرد، کلاس‌های پیچیدگی گویند.

در ادامه‌ی این بخش، به تعریف و بررسی کلاس‌های پیچیدگی می‌پردازیم. دقت کنید که هیچ یک از تعاریف پیشِ رو، به نوع مدل محاسباتی وابسته نمی‌باشند.



شکل ۱: نمودار رابطه‌ی کلاس‌ها

[1] dynamic programming
[2] MaximumFlow
[3] Computational Complexity
[4] decision problems

۱.۱ کلاس پی[5]

تعریف ۲. به مجموعه‌ای از مسائل که در زمان اجرای چندجمله‌ای قابل حل هستند، کلاس پی گفته می‌شود. به بیانی دیگر :

$$\mathbf{P} = \left\{\text{مسائل قابل حل در زمان چندجمله‌ای یا معادلاً در } O(n^c)\right\}$$

که n سایز ورودی و c عددی ثابت است.

تعریف کلاس پی استوار[6] است؛ چرا که در تمام مدل‌های محاسباتی یکسان بوده و همان طور که در ادامه توضیح خواهیم داد، اگر مسئله‌ای با الگوریتمی حاصل از تلفیق الگوریتم‌های دو مسئله‌ی دیگر که عضوی از کلاس پی هستند حل شود، خود به کلاس پی تعلق دارد.

دلایل تعریف کلاس پی:

- تمام مسائلی که الگوریتم خوبی برای آن‌ها وجود دارد، متعلق به این کلاس هستند.
- اگر یک سری الگوریتم (که تعداد آن‌ها از مرتبه‌ی چندجمله‌ای باشد) را، که هر یک زمان اجرای چندجمله‌ای دارند، به روش‌های گوناگون با هم تلفیق کنیم، باز به الگوریتمی با زمان اجرای چندجمله‌ای دست خواهیم یافت. منظور از تلفیق، هر یک از عمل‌های ترکیب، جمع، ضرب و ... است.

به عنوان مثالی از این دست مسائل می‌توان به مسئله‌ی کوتاه‌ترین مسیر، جریان بیشینه، درخت فراگیر کمینه، پیدا کردن مؤلفه‌های همبندی گراف و بسیاری از مسائلی که با برنامه‌ریزی پویا حل می‌شوند مانند یافتن مجموعه‌ی مستقل رأس‌ها در درخت، اشاره نمود.

در بخش برنامه‌ریزی پویا[7] با مسئله‌ی فروشنده‌ی دوره‌گرد نیز آشنا شدیم و برای آن الگوریتم بهتری نسبت به بررسی‌کردن همه‌ی جایگشت‌ها در $O(n!)$ ارائه دادیم که مرتبه‌ی زمان اجرای آن بیشتر از 2^n بود؛ با این حال چندجمله‌ای نبوده و نمی‌توان آن را متعلق به کلاس پی دانست.

اگر بخواهیم برای هر یک از اعضای کلاس پی نیز تفاوت قائل شویم، به این معنا که تفاوت بین $O(n)$ با $O(n^2)$ یا $O(n\log n)$ را در نظر بگیریم، باید به نکات ظریف دیگری از جمله مدل محاسباتی آن مانند ماشین تورینگ و یا word RAM[8] توجه کنیم.

۲.۱ کلاس نمایی[9]

تعریف ۳. به مجموعه مسائلی که در زمان نمایی قابل حل هستند، کلاس نمایی گفته می‌شود. به عبارتی دیگر:

$$\mathbf{EXP} = \left\{\text{مسائل قابل حل در زمان نمایی یا معادلاً در } O(2^{n^c})\right\}$$

که n سایز ورودی و c عددی ثابت است.

[5] **P**olynomial
[6] robust
[7] جلسه‌ی نهم آنالیز الگوریتم‌ها (بهار ۹۹)
[8] word random access machine
[9] **Exp**onential

واضح است که $\mathbf{P} \subseteq \mathbf{EXP}$ و ثابت شده که کلاس نمایی بزرگتر از کلاس پی است.

برای بسیاری از مسائل، پیدا کردن جواب در زمان نمایی چندان دشوار نیست چرا که تنها کافیست با زمان اجرای تا حد امکان خوبی همه‌ی جواب‌ها را بررسی کرده و جواب‌های مورد نظر را از آن‌ها استخراج کنیم. با این وجود، یافتن الگوریتمی با زمان اجرای چندجمله‌ای، حتی اگر از $O(n^{100})$ هم باشد، پیشرفت بزرگی محسوب می‌شود.

مثال: موقعیت خاصی از شطرنج تعمیم‌یافته (و یا بازی صفحه‌ای دیگری مانند GO یا چکرز[10]) داده شده است. با فرض این که بازیکن سفید در ادامه به بهترین نحو ممکن بازی کند، آیا استراتژی برد دارد؟ یا به بیانی دیگر، آیا قادر است با n حرکت، بازیکن سیاه را مات کند؟

توجه کنید که سوال در مورد نسخه‌ای تعمیم‌یافته از این گونه مسائل صحبت می‌کند. یعنی ابعاد صفحه نیز به عنوان ورودی مسئله داده شده است. زیرا به عنوان مثال اگر صفحه‌ی 8×8 به عنوان پیش‌فرض قرار گیرد (یعنی طول ورودی مسئله عددی ثابت باشد)، تعداد کل موقعیت‌های آن از $O(1)$ خواهد بود؛ هر چه قدر هم که این تعداد بزرگ باشد!

اگر شروط محدودکننده‌ای برای شطرنج $n \times n$ در نظر بگیریم که تعداد حرکات حداکثر مقدار یک چندجمله‌ای بر حسب n باشد، پیچیدگی آن نسبت به حالت عدم وجود هر گونه قیدی، کاهش می‌یابد. در این صورت می‌توان مکان آن را در شکل ۱ در حدود نقطه a در نظر گرفت. در صورتی که چنین شروطی در نظر گرفته نشوند، می‌توان اثبات کرد که مسئله، از دشوارترین مسائل کلاس نمایی می‌باشد؛ جایی حدود نقطه b در شکل ۱. ☒

مثال: آیا برنامه‌ی داده شده پس از k قدم اجرا متوقف خواهد شد؟
ورودی. برنامه A و عدد k (برنامه‌ی A می‌تواند یک برنامه‌ی جاوایی، ماشین تورینگ و غیره باشد).
خروجی. «بله» یا «خیر».

همان گونه که ممکن است در ابتدا به ذهن آید، برای حل این مسئله باید فرایند k گام اجرای برنامه‌ی ورودی را شبیه‌سازی نمود و تاکنون راه دیگری برای آن یافت نشده است. بنابراین زمان اجرای آن نمایی است؛ زیرا همان طور که در جلسات پیش نیز اشاره شد، اندازه یا مقدار یک ورودی نسبت به طول آن، نمایی است و چون تعداد دفعات اجرای برنامه برابر است با مقدار k، زمان اجرا نمایی محسوب می‌شود. این مسئله نیز، مطابق با اثباتی، از دشوارترین مسائل نمایی بوده و در شکل ۱ جایی در حدود نقطه **b** قرار دارد. ☒

خوب است بدانید می‌توان مسائلی ساخت که به زمان نمایی مضاعف[11] نیاز داشته باشند و در زمان $O(2^{2^{n^c}})$ حل شوند. به مانند مثال اخیر، هنگامی که تعداد گام‌های اجرای مسئله 2^k باشد.

۳.۱ کلاس $\mathbf{R}$

تعریف ۴. مجموعه‌ی مسائلی را که در زمانی متناهی قابل حل باشند، کلاس $\boldsymbol{R}$ نامند.

همه‌ی مسائل در کلاس $\mathbf{R}$ قابل گنجایش نیستند. از جمله مسئله‌ی توقف[12].

مثال: مسئله‌ی توقف . آیا اگر برنامه‌ی داده‌شده را اجرا کنیم، هرگز پایان می‌یابد؟
ورودی. برنامه‌ی A (ورودی A می‌تواند جزئی از ورودی مسئله و یا تهی باشد).
خروجی. «بله» یا «خیر». ☒

[10] Checkers
[11] double exponential
[12] Halting problem

همان طور که در ابتدا اشاره شد، در این مبحث نسخه[۱۳] تصمیم‌گیری مسئله‌ها در نظر گرفته می‌شود. هر مسئله را می‌توان به صورت تابعی از رشته‌های دودویی (صفر و یک) در نظر گرفت که خروجی آن یکی از عبارات «بله» یا «خیر» است.

$$f : \{\text{رشته‌های دودویی}\} \to \{\text{«بله»}, \text{«خیر»}\}$$

برای مثال، مسئله‌ی وجود مسیر را در نظر بگیرید:
ورودی. گراف $G = (V, E)$ و $s, t \in V$.
خروجی. «بله» یا «خیر» (آیا در گراف G از رأس s مسیری به رأس t وجود دارد؟).
می‌توانیم هر گراف ورودی مسئله را رشته‌هایی از صفر و یک‌ها در نظر بگیریم و خروجی الگوریتم برای ورودی‌هایی غیر متناظر با هر نوع گرافی را «خیر» تعریف کنیم. به رشته‌هایی که معرف هیچ یک از ورودی‌های مناسب مسئله نیستند، اصطلاحاً رشته‌های خراب گفته می‌شود.

قضیه ۱. برای بسیاری از مسائل الگوریتمی وجود ندارد.

اثبات. مجموعه‌ی رشته‌های دودویی با طول متناهی، شماراست و کاردینال آن برابر کاردینال اعداد طبیعی می‌باشد. هر الگوریتم را می‌توان متناظر با رشته‌ای دودویی و متناهی دانست. بنابراین تعداد کل الگوریتم‌ها متناسب با تعداد اعداد طبیعی است. از طرفی تعداد مسائل ناشماراست؛ زیرا هر تابع f به عنوان نماینده‌ی یک مسئله، زیرمجموعه‌ای از مجموعه‌ی رشته‌های دودویی را به عنوان ورودی می‌گیرد و می‌دانیم که مجموعه‌ی توانی اعداد طبیعی (و این‌جا معادلاً رشته‌های دودویی) هم‌توان با مجموعه‌ی اعداد حقیقی می‌باشد. بنابراین، کاردینال مجموعه توابع f، یا مجموعه‌ی کل مسئله‌ها، برابر با کاردینال اعداد حقیقی است.
در نتیجه، تعداد کل مسائل از تعداد کل الگوریتم‌ها بیشتر است و چون هر الگوریتم دقیقاً یک مسئله را حل می‌کند، برای بسیاری از مسائل الگوریتمی وجود ندارد. □

۴.۱ کلاس انپی

پیش از ارائه‌ی هر گونه تعریف رسمی برای کلاس انپی ، ابتدا به بررسی چند مسئله طبق تعریف غیر رسمی آن می‌پردازیم.

$$\mathbf{NP} = \{\text{مسائلی که در زمان چندجمله‌ای به وسیله‌ی یک الگوریتم «خوش‌شانس» قابل حل هستند}\}$$

- مسئله‌ی بلندترین مسیر[۱۴] :
  ورودی. گراف $G = (V, E)$، $s, t \in V$، $w : E \to \mathbb{R}^+$ و $k \in \mathbb{R}^+$.
  خروجی. «بله» یا «خیر» (آیا مسیری از رأس s به رأس t با طول بزرگ‌تر یا مساوی k وجود دارد؟).

  اگر در صورت مسئله، تابع وزنی نباشد (یا وزن تمامی یال‌ها یکسان باشد) و مقدار k برابر $|V| - 1$ باشد، مسئله تبدیل به مسئله‌ی یافتن مسیر هامیلتونی[۱۵] می‌گردد.

  فرض کنید مسیری به طول k وجود داشته باشد. اگر از s شروع کنید و یکی از یال‌های خروجی آن را با حدس انتخاب کرده و پس از رسیدن به هر رأسی همین کار را تکرار کنید، در نهایت با مسیری با طول بیشتر یا مساوی k به t خواهید رسید. البته این موضوع در صورتی برقرار است که شما (یعنی الگوریتم چندجمله‌ای شما) در هر بار انتخاب یال واقعاً «خوش‌شانس» باشید!

- مسئله‌ی زیرمجموعه‌ی مستقل[۱۶] :

[13] version
[14] LongestPath problem
[15] HamiltonianCycle
[16] IndependentSet problem

ورودی. گراف $G = (V, E)$ و عدد $k \in \mathbb{Z}$.
خروجی. «بله» یا «خیر» (آیا G زیر مجموعه‌ی مستقلی از V با اندازه‌ی حداقل k دارد؟).

قبلاً دیدیم در صورتی که گراف ورودی درخت باشد، مسئله با استفاده از برنامه‌ریزی پویا قابل حل است. ولی در حالت کلی، تاکنون الگوریتم بهینه‌ای برای آن ارائه داده نشده.

مشابه مسئله‌ی قبل، اگر مجموعه‌ی رئوس گراف چنین زیرمجموعه‌ای داشته و الگوریتم شما واقعاً خوش‌شانس باشد، با هر بار انتخاب رأس درست، در نهایت به جواب رسیده و در خروجی «بله» چاپ می‌کند. در غیر این صورت به مجموعه‌ای مستقل با کمتر از k رأس رسیده و «خیر» را خروجی می‌دهد.

- بازی تتریس[۱۷]:
ورودی. چینش[۱۸] خاصی از بازی، دنباله‌ای kتایی از قطعات.
خروجی. «بله» یا «خیر» (آیا چینشی از این k قطعه، با ترتیبِ داده شده، وجود دارد که در نهایت منجر به برد شود؟).

برای حل این مسئله نیز، کافیست الگوریتم چندجمله‌ای خوش‌شانسی داشته باشید که در صورت وجود چینش مناسب برای برد، آن را یافته و جواب «بله» بدهد!

تعریف ۵. به مجموعه مسائلی که در صورت «بله» بودن جوابشان، آن جواب در زمان چندجمله‌ای قابل چک کردن باشد، کلاس ان‌پی گفته می‌شود. معادلاً:

$$\mathbf{NP} = \{\text{مسائلی که اگر جوابشان «بله» باشد، قابل چک کردن در زمان چندجمله‌ای است.}\}$$

برای درک بهتر تعریف، فرض کنید برای ورودی خاصی، جواب مسئله‌ی زیرمجموعه‌ی مستقل «بله» باشد. در این صورت، با داشتن زیرمجموعه‌ی مورد نظر، کافیست بررسی کنیم که آیا مستقل هست یا نه؛ که این در زمان چندجمله‌ای کاملاً امکان‌پذیر است. ولی اگر جواب «خیر» باشد، برای حل مسئله باید تمام زیرمجموعه‌های k عضوی رئوس را چک کرد که زمان اجرای آن زیاد است. پس به طور کلی، چک کردن درستی یک جواب «آسان» است و پیدا کردن آن «سخت».

نکته اینجاست که اکثر مسائل، متعلق به این کلاس هستند.

سوال. چرا تعریف اخیر، با تعریف اولیه معادل است؟ در تعریف اولیه، فرض بر این است که الگوریتمی خوش‌شانس و چندجمله‌ای داریم که جواب‌ها را حدس می‌زند و در تعریف رسمی، فرض این است که جواب‌ها داده شده‌اند و تنها باید درستی آن‌ها با الگوریتمی چندجمله‌ای بررسی شود. بنابراین، الگوریتم خوش‌شانس جواب را با زمان اجرای چندجمله‌ای می‌یابد و در تعریف رسمی الگوریتمی در زمان مشابه، درستی همان جواب را بررسی می‌کند.

رابطه‌ی کلاس ان‌پی و دیگر کلاس‌ها

طبق تعریف هر دو کلاس پی و ان‌پی به وضوح می‌توان رابطه‌ی $\mathbf{P} \subseteq \mathbf{NP}$ را نتیجه گرفت. چون هر مسئله‌ی پی، فارغ از خوش‌شانس بودن یا نبودن الگوریتم آن، دارای الگوریتمی چندجمله‌ای است. همچنین برقراری رابطه $\mathbf{NP} \subseteq \mathbf{EXP}$ نیز روشن است. زیرا می‌توان هر مسئله‌ی ان‌پی را با استفاده از یک حلقه روی تمام حدس‌های موجود آن بررسی کرده و جواب آن را به دست آورد. تعداد این حدس‌ها نیز از مرتبه‌ی نمایی است. چیزی که هنوز ثابت نشده، وجود مسئله‌ای در مجموعه‌ی $\mathbf{NP} \setminus \mathbf{P}$ است. احتمالی مبنی بر عضوِ پی بودنِ تمام مسائل ان‌پی وجود دارد. ولی هنوز مشخص نیست.

[17] Tetris game
[18] configuration

در علوم کامپیوتر امروزین، مهم‌ترین مسئله‌ی حل نشده اثبات برقراری یا عدم برقراری رابطه‌ی $\mathbf{P} = \mathbf{NP}$ است. اکثر دانشمندان این حوزه بر این باورند که تساوی فوق برقرار نیست[۱۹]. اگر $\mathbf{P} = \mathbf{NP}$، آنگاه قادر خواهیم بود که حتی با الگوریتمی غیر خوش‌شانس و با کامپیوترهای معمولی، به جواب برسیم.

مشابه با سخت‌ترین مسائلی که برای کلاس نمایی معرفی شد، مسائلی هم هستند که به عنوان سخت‌ترین مسائل انپی ثابت و شناخته شده‌اند و اگر تنها برای یکی از آن‌ها الگوریتمی چندجمله‌ای ارائه شود، مانند این خواهد بود که برای تمام مسائل انپی الگوریتم چندجمله‌ای پیدا شده و به این ترتیب برقراری رابطه‌ی $\mathbf{P} = \mathbf{NP}$ نیز ثابت می‌گردد.

۲ تحویل[۲۰] (تبدیل)

پیش از این دیدیم که برای حل بسیاری از مسائل، می‌توانیم آن‌ها را به مسئله‌ای ازپیش‌حل‌شده مدل کنیم؛ مانند مسئله‌ی یافتن تطابق بیشینه در گراف دو بخشی[۲۱] که با استفاده از مسئله‌ی جریان بیشینه به راحتی پاسخ داده شد. تحویل، رایج ترین شیوه‌ی طراحی الگوریتم است. خوب است همیشه پیش از تلاش برای طراحی الگوریتمی جدید برای یک مسئله، سعی کنید آن را با استفاده از مسائل قبلی حل کنید.

برای نمایش تحویل چندجمله‌ای مسئله‌ی A به مسئله‌ی B، از نماد $\leq_{\mathbf{P}}$ استفاده شده و به صورت $A \leq_{\mathbf{P}} B$ نشان داده می‌شود. در واقع چنین عبارتی، بیانگر این است که اگر الگوریتمی چندجمله‌ای برای B موجود باشد، می‌توان با استفاده از آن الگوریتمی چندجمله‌ای برای مسئله‌ی A ارائه داد.

تحویل چندجمله‌ای چند به یک[۲۲] (تابعی یا کارپ[۲۳])

مسئله‌ی A قابل تحویل چندجمله‌ای به مسئله‌ی B است اگر و تنها اگر تابعی قابل محاسبه در زمان چندجمله‌ای مانند f موجود باشد به طوری که

$$\forall x \quad x \in A \iff f(x) \in B$$

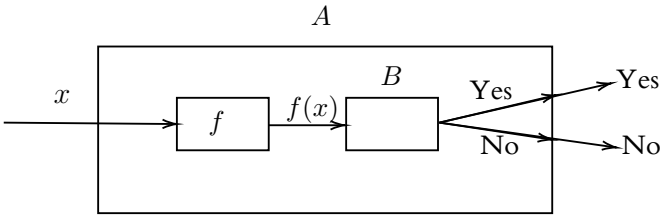
یعنی ورودی‌ای از مسئله‌ی A به تابع f (با زمان اجرای چندجمله‌ای) داده شود و این تابع خروجی متناسب با ورودی مسئله‌ی B را تولید کند. بنابراین اگر به ازای ورودی داده شده، جواب مسئله‌ی A «بله» باشد، جواب مسئله‌ی B نیز به ازای مقدار تابع f که بر آن ورودی اعمال شده، «بله» است و بالعکس.

مثال: فرض کنید x گرافی دو بخشی است و به دنبال پاسخی برای سوال «آیا تطابقی با اندازه‌ی k در x وجود دارد؟» هستیم. در مبحث جریان در شبکه، دیدیم که با مدل‌کردن این مسئله به جریان بیشینه، جواب به دست می‌آید. در اینجا نیز مطابقاً تابع f شبکه‌ای از جریان‌ها را در تناظر با x می‌سازد. این شبکه به همراه عدد k، ورودی مسئله‌ی جریان بیشینه می‌گردد. حال سوال این است که «آیا جریانی با اندازه‌ی بزرگ‌تر یا مساوی k در این شبکه می‌تواند برقرار باشد؟». ☒

[۱۹] از جمله اریک دیمین، استاد دانشگاه MIT، که جمله‌ی معروفی در این باره دارد: «نمی‌توانید به طور مکانیکی عملی انجام دهید که الگوریتم شما به اندازه‌ی الگوریتم خوش‌شانس ایده‌آل، خوب کار کند!»

"You can't engineer luck!"– Erik D. Demaine

[20] reduction
[21] MaximumMatching in Bipartite Graphs
[22] polynomial-time many-one reduction
[23] Karp



شکل ۲: با اعمال عمل‌گر f بر x (ورودی مسئله‌ی A)، ورودی مسئله‌ی B تولید می‌شود. خروجی الگوریتم مسئله‌ی B، همان جواب مسئله‌ی A است.

مثال:تحویل مسئله‌ی طولانی‌ترین مسیر به کوتاه‌ترین مسیر. تابع f، وزن هر یال گراف ورودی را در -1 ضرب کرده و گراف جدید را به عنوان ورودی به الگوریتم مسئله‌ی کوتاه‌ترین مسیر می‌دهد. ⊠

بنابراین طبق آنچه گفته شد، (طراحی الگوریتم) اگر $A \leq_{\mathbf{P}} B$ و برای B الگوریتمی با زمان اجرای چندجمله‌ای وجود داشته باشد، برای A نیز وجود دارد.

پس کافیست، مطابق با شکل، ورودی A به تابع (برنامه) f داده شود و f پس از اعمال تغییرات لازم برای تبدیل آن به ورودی مسئله‌ی B، $f(x)$ را به عنوان ورودی به الگوریتم مسئله‌ی B بدهد. توجه کنید که عکس این گزاره برقرار نیست؛ (اثبات عدم وجود الگوریتم) اگر $A \leq_{\mathbf{P}} B$ و برای A الگوریتمی با زمان اجرای چندجمله‌ای یافت نشود، برای B نیز چنین الگوریتمی وجود نخواهد داشت. پس برای نشان‌دادن سختی B، پیدا کردن مسئله‌ای مانند A که قبلا سخت بودن آن ثابت شده و به B قابل تحویل است، کفایت می‌کند.

۳ مسائل انپی_تمام[۲۴]

تعریف ۶. مسئله‌ی Q، انپی_تمام است اگر:

۱. $Q \in \mathbf{NP}$

۲. $\forall A \in \mathbf{NP} \quad A \leq_{\mathbf{P}} Q$

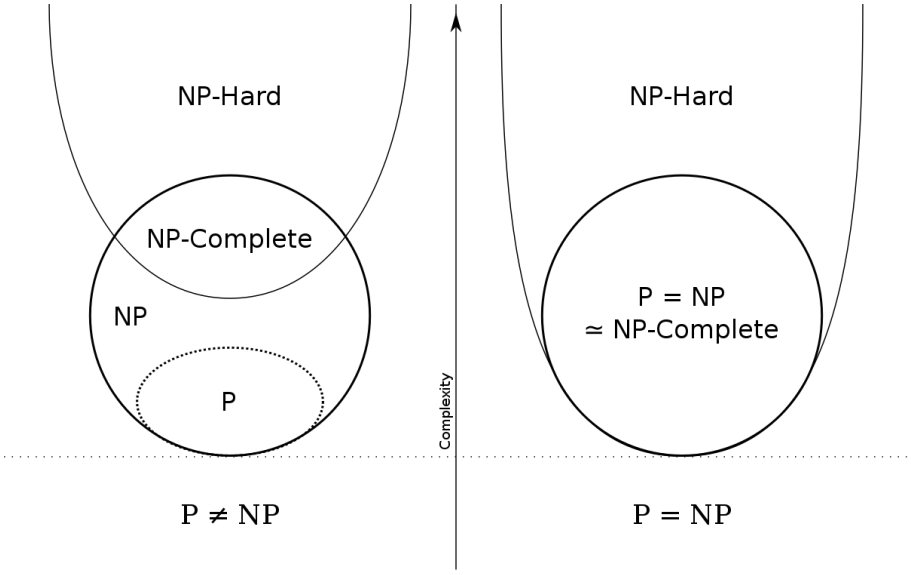
طبق این تعریف، اگر برای مسئله‌ای مانند Q مورد دوم برقرار باشد، آنگاه Q سخت‌ترین مسئله‌ی کلاس انپی است؛ چون، همان طور که قبلاً اشاره کردیم، میزان سختی Qحداقل با میزان سختی هر مسئله‌ی انپی_تمام دیگری از انپی مانند A یکسان خواهد بود ($A \leq_{\mathbf{P}} Q$). پس اگر برای Q الگوریتمی چندجمله‌ای یافت شود، معادل یافتن الگوریتمی چندجمله‌ای برای تمام مسائل انپی است و معادلاً، اگر تنها برای یک مسئله از کلاس انپی الگوریتمی چندجمله‌ای موجود نباشد، برای Q نیز وجود ندارد.

نکته‌ی جالب در نظریه‌ی انپی_تمامیت این است که نه تنها مسائلی مثل مسئله‌ی Q، که در بالا گفته شد، وجود دارند بلکه تعداد آن‌ها بسیار زیاد است! تقریباً هر مسئله‌ای که در عمل با آن روبه‌رو می‌شوید، از دو حالت خارج نیست: یا متعلق به کلاس پی است و یا به کلاس انپی . تعداد ناچیزی از آن‌ها خارج از محدوده‌ی انپی هستند (شکل ۳).

تعریف ۷. مجموعه مسائلی که خاصیت دوم در تعریف ۶ را دارا باشند، انپی_سخت[۲۵] نام دارد.

[24] **NP-complete**
[25] **NP – Hard**

بنابراین یک مسئله انپی‌ـ تمام است اگر متعلق به هر دو کلاس انپی و انپی‌ـ سخت باشد.

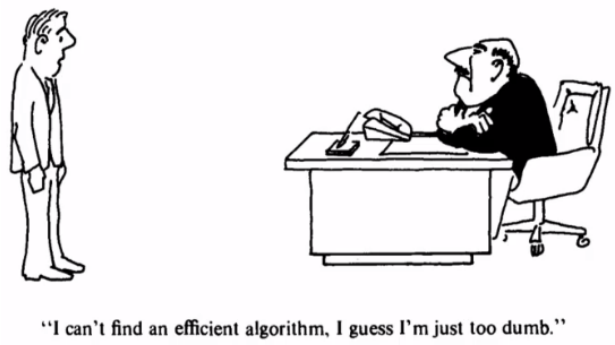


شکل ۳: [۲]نمودار سمت چپ باور کنونی ماست و تصویر سمت راست در صورتی محقق می‌شود که ثابت گردد $\mathbf{P} = \mathbf{NP}$.

قضیه ۲. مسئله‌ی زیرمجموعه‌ی مستقل، بلندترین مسیر، تتریس و . . . انپی‌ـ تمام هستند.

بنابراین برای نشان‌دادن «خیلی سخت»بودن یک مسئله، کافیست انپی‌ـ تمام بودن آن را ثابت کنیم، بدون نیاز به اثبات دیگری. و اگر الگوریتمی چندجمله‌ای برای آن پیدا شود (مثلاً برای مسئله‌ی تتریس چنین الگوریتمی بیابیم)، به این معنا خواهد بود که برای تمام مسائل انپی‌ـ تمامِ دیگر نیز الگوریتم چندجمله‌ای وجود دارد. برای درک بهتر انپی‌ـ تمامیت، به مثال[۲۶] زیر توجه کنید:

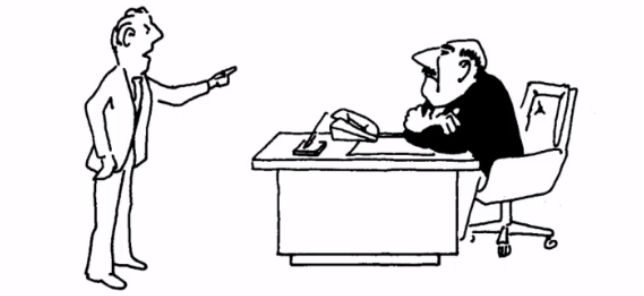
فرض کنید که در محل کارتان مسئله‌ای پیش آمده و از شما خواسته‌اند تا آن را حل کنید و الگوریتمی بهینه برای آن بیابید. اما پس از چند هفته، نتوانسته‌اید که الگوریتم دلخواه را طراحی کنید. بنابراین به پیش رئیس خود خواهید رفت تا این موضوع را با او در میان بگذارید؛ اما مطمئناً آن را با مثلاً نادانی خود استدلال نمی‌کنید:



شکل ۴: قطعاً به رئیس خود نخواهید گفت : «من نمی‌توانم الگوریتم بهینه را پیدا کنم، فکر کنم زیادی احمقم!»

برای این که احمق به نظر نرسید و جایگاهتان به خطر نیفتد، ایده‌آل این است که ثابت کنید این مسئله ذاتاً دشوار است و هیچ الگوریتم سریعی برایش وجود ندارد:

[۲۶]این مثال برگرفته از فصل اول کتابی است که در سال ۱۹۷۹ توسط مایکل گری و دیوید جانسون نوشته شد[۱]. این کتاب به معرفی و بررسی انپی‌ـ تمامیت و تحویل می‌پردازد. در انتهای آن نیز تعداد زیادی از مسائل انپی‌ـ تمام و ۱۲ مسئله‌ی باز (open problem) معرفی گردیده است.



"I can't find an efficient algorithm, because no such algorithm is possible!"

شکل ۵: می‌توانید به رئیس خود بگویید: «من نمی‌توانم الگوریتم بهینه را پیدا کنم، چون اصلاً چنین الگوریتمی وجود ندارد!»

متاسفانه اثبات سخت‌بودن مسئله نیز، به نوعی، به اندازه‌ی پیدا کردن الگوریتم بهینه‌ی آن دشوار است و بهتر است این گونه دلیل بیاورید:

"I can't find an efficient algorithm, but neither can all these famous people."

شکل ۶: می‌توانید بگویید: «من نمی‌توانم الگوریتم بهینه را پیدا کنم، و هیچ یک از این افراد مشهور هم نتوانسته‌اند.»

صحبت از ان‌پی‌ـ تمامیت نیز بی‌شباهت به مورد سوم نیست؛ شما نمی‌توانید اثبات ریاضی برای سخت بودن یک مسئله و نبودن الگوریتم چندجمله‌ای برای آن بیاورید. از طرفی اگر مسئله‌ی شما ان‌پی‌ـ تمام باشد و الگوریتم بهینه‌ی آن را بدست آورید، توانسته‌اید ثابت کنید برای تمام مسائل ان‌پی‌ـ تمام دیگر نیز الگوریتم مورد نظر وجود دارد. پس:

«سختی سایر مسائلی که الگوریتم مناسبشان پیدا نشده است $\geq$ سختی مسئله‌ی شما»

سوال. چگونه ثابت کنیم مسئله‌ای مانند Q، ان‌پی‌ـ تمام است؟

۱. تحویل چندجمله‌ای تمام مسائل ان‌پی‌ـ تمام دیگر به Q، طبق تعریف.

۲. تحویل چندجمله‌ای یک مسئله، مانند A، به Q: $A \leq_{\mathrm{P}} Q$.

اثبات ان‌پی‌ـ تمام بودن یک مسئله با استفاده از مورد اول کاری بسیار دشوار است، چرا که باید تمام مسائل ان‌پی‌ـ تمام دیگر به Q تحویل شوند. اما طبق مورد دوم، یک بار اجرای این کار کفایت می‌کند. زیرا قبلاً ثابت شده که مسئله‌ی A، ان‌پی‌ـ تمام است و اکنون با تحویل A به Q، به نتیجه دلخواهمان خواهیم رسید و این نیز به استناد قضیه‌ی زیر واضح است:

قضیه ۳. تحویل چندجمله‌ای، خاصیت تراگذری دارد. به بیانی دیگر: $L \leq_{\boldsymbol{P}} A \wedge A \leq_{\mathrm{P}} \implies L \leq_{\mathrm{P}} Q$

اثبات. به خواننده واگذار می‌شود. □

در نتیجه، پس از اثبات انپی‌ـ تمام بودن Q، می‌توان برای نشان‌دادن انپی‌ـ تمام بودن مسئله‌ی دیگری از آن استفاده نمود.

در ادامه مسئله‌ای انپی‌ـ تمام را معرفی کرده و انپی‌ـ تمام بودن مسئله‌ی زیرمجموعه‌ی مستقل را با تحویل آن نشان می‌دهیم.

مثال: (۳ـ صدق‌پذیری)[۲۷]. مجموعه‌ای شامل گزاره‌های اتمی مانند $x_1, \dots, x_n$ را در نظر بگیرید. به هر گزاره‌ی اتمی x_i و یا نقیض آن $\neg x_i$ یک لیترال[۲۸]، گفته می‌شود. فرم فصلی[۲۹] لیترال‌ها را نیز عبارت[۳۰] می‌نامند. فرض کنید فرمولی از عطف عباراتی که هر یک حداکثر متشکل از سه لیترال است[۳۱]، داده شده. آیا می‌توان هر لیترال را به گونه‌ای ارزش‌گذاری کرد که در نهایت ارزش فرمول داده شده TRUE گردد؟ برای مثال، در فرمول زیر، آیا می‌توان به هر یک از x_1, x_2, x_3, x_4 طوری TRUE یا FALSE نسبت داد که فرمول TRUE شود؟

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

ثابت می‌شود که این مسئله انپی‌ـ تمام است. ⊠

از منظر تاریخی، در سال ۱۹۷۱، لوین[۳۲] و استیون کوک[۳۳] اولین کسانی بودند که ثابت کردند یک مسئله‌ی انپی‌ـ تمام وجود دارد. قضیه‌ی وجود داشتن یک مسئله‌ی انپی‌ـ تمام به قضیه‌ی کوک‌ـ لوین نیز شهرت دارد[۳۴]. آن مسئله[۳۵] ، مشابه مسئله‌ی ۳ـ صدق‌پذیری است. ریچارد کارپ[۳۶] نیز، در سال ۱۹۷۲، انپی‌ـ تمام بودن بیست مسئله‌ی دیگر را با استفاده از تحویل چندجمله‌ای ثابت کرد؛ از جمله مسائل مجموعه‌ی رئوس مستقل، پوشش رأسی[۳۷]، دور هامیلتونی و ....

تحویل چندجمله‌ای مسئله‌ی ۳ـ صدق‌پذیری به مسئله‌ی مجموعه‌ی مستقل

اکنون با فرض انپی بودن دو مسئله‌ی ۳ـ صدق‌پذیری (3SAT) و مجموعه‌ی مستقل (IndSet)، می‌شود مسئله‌ی اول را به دومی تحویل کنیم؛ یعنی باید تابع fی ارائه کنیم که در گزاره‌ی زیر صدق کند:

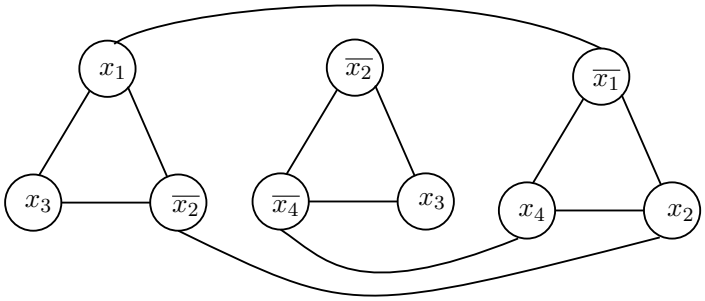
$$\varphi \in \mathsf{3SAT} \iff f(\varphi) \in \mathsf{IndSet}$$

در واقع نقش تابع f این است که ورودی x را که به ازای آن جواب مسئله‌ی TRUE می‌شود، گرفته و در زمان چندجمله‌ای به ورودی مسئله‌ی IndSet یعنی گراف متناظر با فرمول φ و عدد k، تبدیل کند به طوری که جواب این مسئله نیز TRUE گردد.

حال سوال این است که «گراف مذکور چگونه ساخته شود تا ما را به جواب مطلوب برساند؟» فرض کنید به ازای هر لیترال، و برای هر یک به تعداد دفعات تکرار آن در φ (فرمول ورودی)، رأس در نظر بگیریم. و بین رئوسِ در تناظر با لیترال‌هایی که در یک عبارت هستند، و همچنین بین هر رأسِ یک لیترال و نقیضش یال قرار دهیم. شکل ۳ گراف متناظر مثال قبل را نشان می‌دهد.

حال این گراف به همراه عدد «تعداد عبارت‌های φ» $:= m$ همان $f(\varphi)$ مورد نظر می‌باشد.

[27] 3 – Satisfiability problem
[28] literal
[29] Disjunctive Normal Form
[30] clause
[31] 3-Conjunctive Normal Form
[32] Leonid Levin
[33] Stephen Cook
[34] Cook–Levin theorem
[35] BooleanSatisfabity problem
[36] Richard M. Karp
[37] VertexCover



شکل ۷: «$\varphi(x_1, x_2, x_3, x_4) = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2 \vee x_4)$».

ادعا ۱. فرمول φ ارضا پذیر است اگر و تنها اگر گراف ساخته شده مجموعه‌ی مستقلی با سایز m داشته باشد.

اثبات. ($\Rightarrow$) فرض کنید هر لیترال به گونه‌ای مقداردهی شده باشد که مقدار φ، TRUE گردد؛ یعنی در هر عبارت یک لیترال TRUE باشد. حال تمام رئوسی را که مقدار متناظرآن‌ها TRUE می‌باشد از گراف متناظر بردارید؛ این رئوس، مجموعه‌ای مستقل تشکیل می‌دهند. برای درک بهتر، فرمول شکل ۳ را این گونه مقداردهی کنید:

$$x_1 = \text{TRUE}, x_2 = \text{FALSE}, x_3 = \text{FALSE}, x_4 = \text{TRUE}$$

($\Leftarrow$) اگر گراف مجموعه‌ای مستقل با سایز m داشته باشد، به وضوح از هر مثلث یک رأس در آن مجموعه وجود دارد. زیرا در غیر این صورت یا کمتر از m رأس و یا حداقل دو رأس غیر مستقل خواهد داشت. برای مقداردهی معتبر φ کافیست به هر لیترال متناظر با رأسی از مجموعه‌ی مستقل، TRUE نسبت داده شود. روشن است که این گونه امکان ندارد یک لیترال و نقیضش مقداری یکسان داشته باشند، چون اگر رأس نظیر یکی از آن‌ها در مجموعه‌ی مستقل باشد و در نتیجه مقدار TRUE را اخذ کند، رأس نظیر نقیضش در آن مجموعه نیست (از هم مستقل نیستند) و بدون ایجاد مشکلی مقدار FALSE را اخذ می‌کند. ممکن است در این بین لیترال‌هایی نیز باشند که هیچ مقداری به آن‌ها نسبت داده نشده، در این حالت هر گونه مقداردهی آن‌ها تفاوتی در مقدار φ ایجاد نخواهد کرد. □

مراجع

[1] Michael R. Garey, David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness* 1st ed., W. H. Freeman and Company, 1979, pp. 2-4.

[2] Wikipedia contributors. *"NP-completeness."* Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 14 Mar. 2020. Web.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۱۸: مسائل انپی_ تمام ۱ نگارنده: شایان طاهری جم

در جلسه گذشته دسته بندی‌ای از مسائل ارائه دادیم و راجع به مسائل انپی صحبت کردیم. تعریف انپی به صورت شهودی اینگونه بود که اگر دنباله‌ای از حدس‌های خیلی خوب برای جواب داشته باشیم می‌توانیم جواب را در زمان چندجمله‌ای پیدا کنیم.
یا به عبارتی الگوریتمی چندجمله‌ای وجود داشته باشد که اگر جواب مسئله بله باشد الگوریتممان با احتمال ناصفر می‌گوید بله و اگر جواب مسئله خیر باشد با احتمال صفر می‌گوید خیر.

این خطر وجود دارد که این الگوریتم‌ها با الگوریتم‌های تصادفی چندجمله‌ای اشتباه شوند. الگوریتم‌های تصادفی چندجمله‌ای همین تعریف اخیر را دارند ولی احتمال بله گفتن آن‌ها باید بالا باشد ولی برای انپی می‌تواند کم هم باشد، مثلا اگر احتمال درست جواب دادن 10^{-100} باشد خوب است چون اگر الگوریتم را به تعداد زیادی انجام دهیم به احتمال خوبی به جواب می‌رسیم و اگر احتمال درست جواب دادن $\frac{1}{n^{100}}$ باشد باز هم خوب است زیرا کافیست الگوریتم را چندجمله‌ای بار تکرار کنیم تا با احتمال خوبی به جواب درست برسیم، ولی مثلاً احتمال $\frac{1}{2^{n^c}}$ در عمل خوب نیست زیرا اگر این الگوریتم را چندجمله‌ای بار هم اجرا کنیم باز هم با احتمال کمی به جواب درست می‌رسیم.

مسئله Q در انپی است اگر و فقط اگر الگوریتم چندجمله‌ای V وجود داشته باشد به طوری که $x \in \mathrm{Q}$ (یعنی جواب ورودی x برای مسئله Q بله باشد) اگر و تنها اگر رشته w به طول چندجمله‌ای (بر حسب x) وجود داشته باشد که $V(x,w)=1$.
برای مثال در مسئله وجود دور همیلتونی در گراف می‌توانیم الگوریتمی طراحی کنیم که ترتیب رئوس را بگیرد و بررسی کند که آیا این دور همیلتونی است یا خیر.

این جلسه راجع به مسائل انپی_ تمام صحبت می‌کنیم.

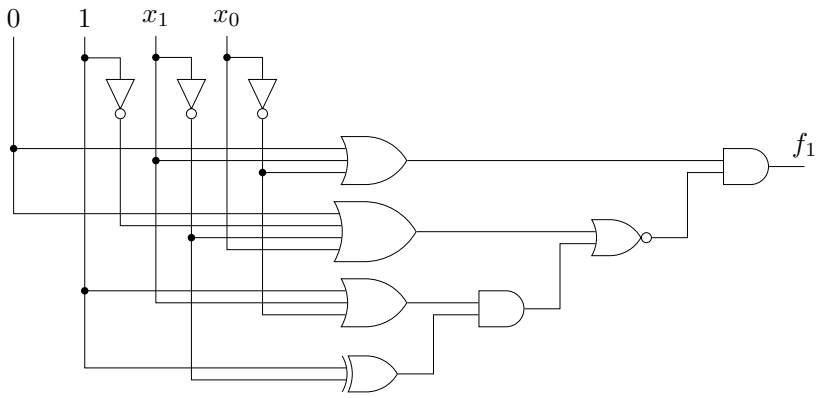
می‌خواهیم راجع به مسئله‌ای صحبت کنیم که تمام مسائل انپی به آن تحویل می‌شوند.

۱ مسئله صدق پذیری مدار [1]

مدار منطقی‌ای شامل ۳ نوع گیت And , Or , Not به ما داده می‌شود مانند شکل زیر و تعدادی از ورودی‌ها مشخص شده اند.و تنها یک خروجی داریم. حال سؤال این است که آیا می‌توان بقیه ورودی‌ها را طوری تعیین کرد که خروجی ۱ شود؟

می‌توان راحت دید که این مسئله انپی است. زیرا می‌توانیم با داشتن مقدار ورودی به راحتی خروجی را چک کنیم. حال می‌خواهیم ثابت کنیم هر مسئله انپی را می‌توانیم به این مسئله تحویل کنیم.

[1] CircuitSAT



فرض کنید مسئله Q انپی است می‌خواهیم نشان دهیم Q به صدق پذیری مدار تحویل میشود بنابر تعریف، الگوریتم چندجمله‌ای V برای Q وجود دارد به طوری که $x \in \mathrm{Q}$ (منظور این است که مسئله Q با ورودی x خروجی ۱ می‌دهد) اگر و تنها اگر رشته w با طول چندجمله‌ای (بر حسب x) وجود داشته باشد که $V(x,w)=1$.
باید الگوریتم چندجمله‌ای f را ارائه دهیم به طوری که:

$$\forall x: \quad x \in \mathrm{Q} \iff f(x) \in \mathrm{CircuitSAT}$$

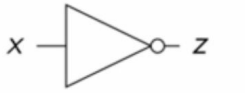
قرار نیست در این درس به اثبات دقیق مسئله بپردازیم. اثبات دقیق را می‌توانید در درس‌های پیچیدگی محاسباتی یا در نظریه‌ی‌محاسبه یا چنین درس‌هایی مطالعه کنید. در این درس صرفاً سعی داریم شهود کافی را منتقل کنیم. می‌خواهیم ثابت کنیم مسئله را می‌توانیم به مدار تبدیل کنیم. چون خود سخت‌افزار کامپیوتر هم جوری مدار منطقی است مثلا CPU کامپیوتر نوعی مدار منطقی است. با این گیت‌ها می‌توان هر الگوریتمی را پیاده‌سازی کرد. مثلا می‌توانیم مدل مداری الگوریتم V را بسازیم و تبدیل می‌شود به مسئله صدق پذیری مدار (البته این فقط ایده مسئله است).
در واقع هر الگوریتمی را که اجرا می‌کنیم می‌توانیم هر مرحله الگوریتم را به صورت یک رشته‌ی دودویی نمایش دهیم مثلا محتویات حافظه و مقادیر متغیرها در زمان i را می‌توانیم در یک ردیف نگه داریم. از هر مرحله به هر مرحله دیگر برنامه عملیات‌های محلی[۲] انجام می‌دهد. می‌توانیم بین هر دو مرحله یک مدار بسازیم که ورودی‌های آن محتویات وضعیت خانه‌های حافظه در زمان $t=i$ و خروجی، محتویات وضعیت خانه‌های حافظه در زمان $t=i+1$ باشد. می‌توانیم در آخر یک مدار بسازیم که برای خروجی‌های مطلوب 1 خروجی دهد.
پس اینگونه می‌توانیم هر مسئله انپی را به صدق پذیری مدار کاهش دهیم.

می‌دانیم مسئله‌ی صدق پذیری مدار ، انپی_ تمام است. می‌خواهیم آن را به یک مسئله‌ی انپی کاهش دهیم و ثابت کنیم این مسئله نیز انپی_ تمام است.
مثلاً می‌توانیم ثابت کنیم مسئله صدق پذیری مدار به مسئله ۳_صدق پذیری کاهش پیدا میکند.
مسئله‌ی ۳_صدق پذیری خیلی مسئله‌ی خوبی برای اثبات انپی_تمام بودن دیگر مسئله‌ها است. یعنی می‌توانیم ۳_صدق پذیری را به مسئله‌ی انپی مذکور کاهش دهیم تا ثابت شود که آن مسئله انپی_ تمام است.
می‌خواهیم به ازای یک مدار یک فرمول برای ۳_صدق پذیری بسازیم به طوری که این فرمول ارضاپذیر باشد اگر و تنها اگر خروجی مدار 1 باشد.
نکته اینجا است که هر گیت را می‌توان به یک عبارت از فرمول‌ها مدل کرد و در نهایت این عبارت‌ها را با هم And می‌کنیم و به فرمول می‌رسیم.
این مدار ساخته شده خروجی 1 دارد اگر و تنها اگر فرمول ارضا پذیر باشد.
برای مدارمان خروجی S در نظر می‌گیریم و برای فرمول کل فرمول را با S ، And می‌کنیم.پس در نهایت فرمول ارضا پذیر است اگر و تنها

[2] local

اگر خروجی مدار ۱ شود. پس این تحویلی از مسئله‌ی صدق پذیری مدار به ۳_صدق پذیری است. پس چون ۳_صدق پذیری ، انپی است طبق این تحویل نتیجه می‌گیریم که انپی_تمام نیز است.

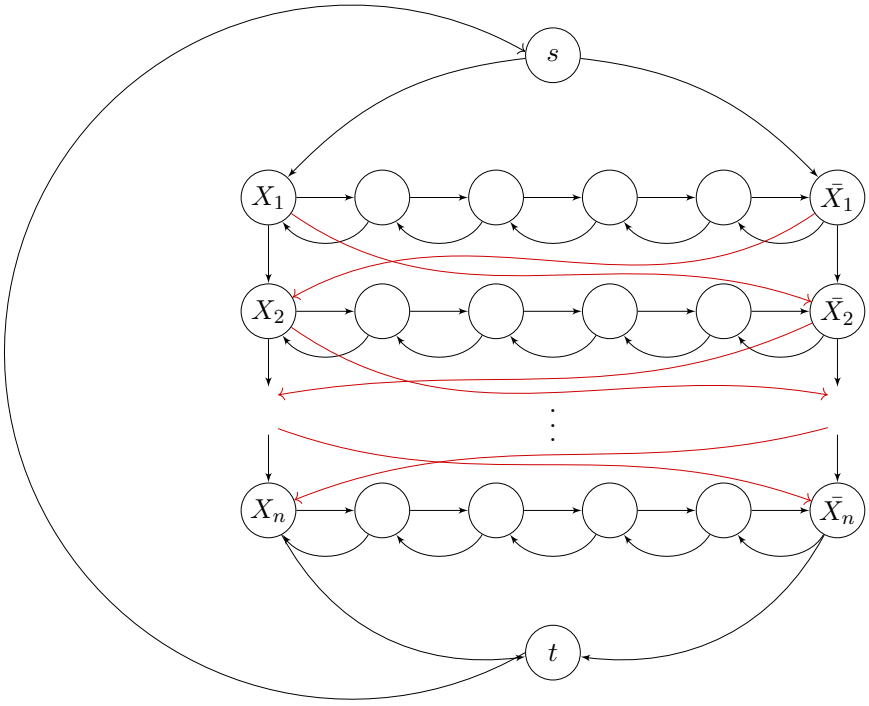
در شکل زیر مدل چند گیت آمده است:

Gate	Formula
	$(\neg x \vee \neg y \vee z) \wedge (x \vee \neg z) \wedge (y \vee \neg z)$
	$(x \vee y \vee \neg z) \wedge (\neg x \vee z) \wedge (\neg y \vee z)$
	$(x \vee z) \wedge (\neg x \vee \neg z)$

حال می‌توانیم از ۳_صدق پذیری استفاده کنیم و آن را به مسائل دیگری تحویل کنیم و ثابت کنیم آن مسئل نیز انپی_تمام هستند.

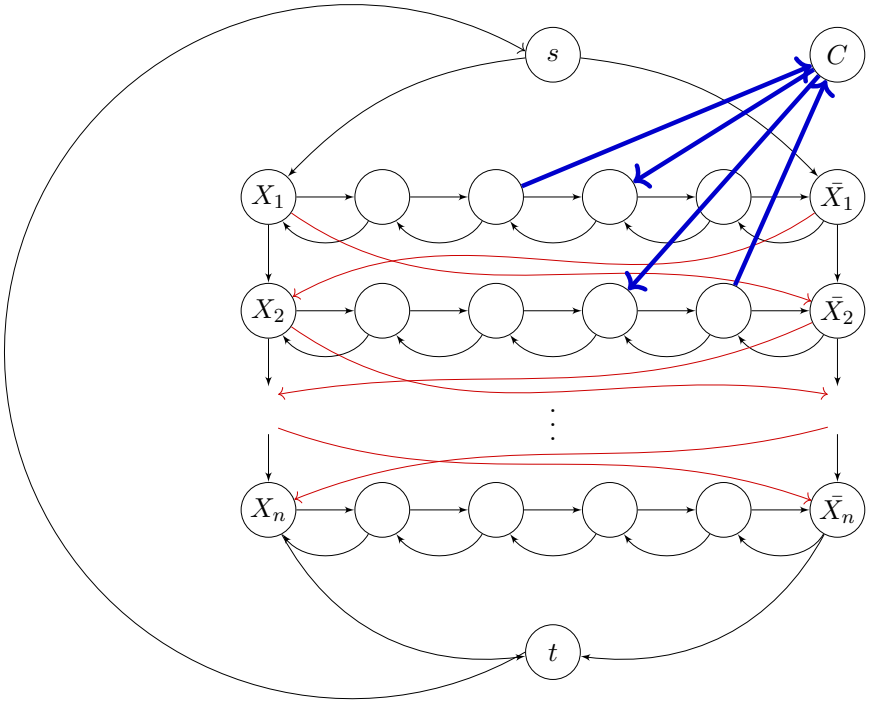
۲ تحویل مسئله ۳_صدق پذیری به مسئله وجود دور همیلتونی [۳]

فرض کنید $X_1, X_2, \ldots, X_n$ متغیرهای فرمول باشند. گراف زیر را تشکیل می‌دهیم :



[3] DHAMcycle

هر ردیف این گراف متناظر با یکی از متغیرها است.
حال فرض کنید عبارتی مانند $(X_1 \vee \bar{X}_2)$ در فرمول ظاهر شده باشد. رأس C_1 را به گراف اضافه میکنیم یال‌های مربوطه را رسم می‌کنیم.



این رأس را وقتی در دور همیلتونی از مسی‌های X_1 به سمت $\bar{X}_1$ یا $\bar{X}_2$ به سمت X_2 حرکت می‌کنیم می‌توانیم پیمایش کنیم یعنی اگر دور همیلتونی داشته باشیم حداقل یکی از ۲ مسیر بالا را باید پیمایش کنیم و می‌توانیم پیمایش مسیر X_i به سمت $\bar{X}_i$ را متناظر با true بودن X_i و مسیر $\bar{X}_i$ به سمت X_i را متناظر با false بودن X_i بدانیم.
حال ثابت می‌کنیم که فرمول ارضا شدنی است اگر و تنها اگر این گراف دور همیلتونی داشته باشد.
برای هر X_i اگر این متغیر true بود مسیر X_i را از چپ به راست پیمایش می‌کنیم و اگر false بود مسیر را از راست به چپ.
با حالت‌بندی هم می‌توان ثابت کرد که دور همیلتونی یک شرایط ارضاپذیر برای فرمول به ما ارائه می‌دهد (برای هر عبارت که رأسی به گراف اضافه کردیم و از رأس‌های میانی مسیرها به آن یال وصل کردیم این رأس‌های میانی منحصر به فرد و یکبار مصرف هستند و همینطور بین هر جفت رأس انتخابی یک رأس را خالی می‌گذاریم که برای اثبات اینکه دور همیلتونی شرایط ارضاپذیر می‌دهد کاربرد دارد.).

تمرین: نشان دهید مسئله وجود دور همیلتونی در گراف بدون جهت انپی_ تمام است.

می‌توانید ۳_صدق پذیری را به این مسئله تحویل کنید و حتی می‌توانید دور همیلتونی با جهت را هم تحویل کنید.

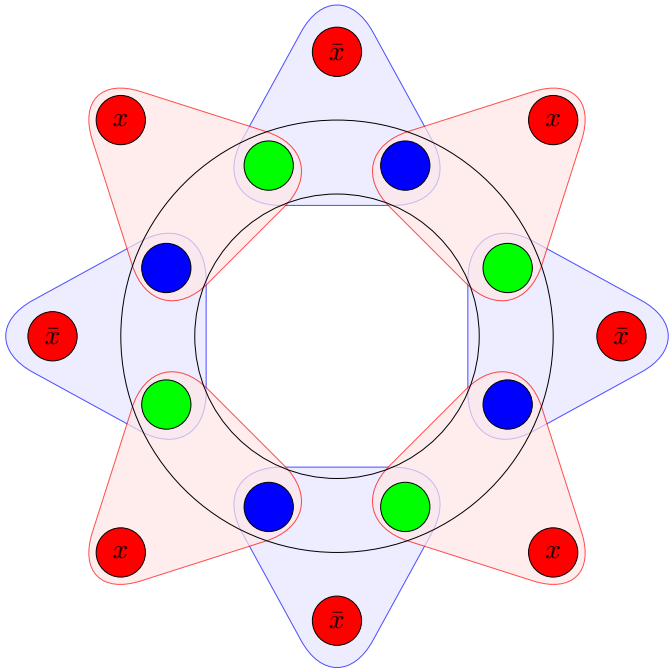
۳ مسئله تطابق سه بعدی [۴]

ورودی : مجموعه‌های n عضوی مجزای X, Y, Z و $T \subseteq X \times Y \times Z$.

خروجی : آیا $S \subseteq T$ وجود دارد که هر عضو $X \cup Y \cup Z$ دقیقاً در یکی از اعضای S آمده باشد؟

[4] 3-dimensional matching (3DM)

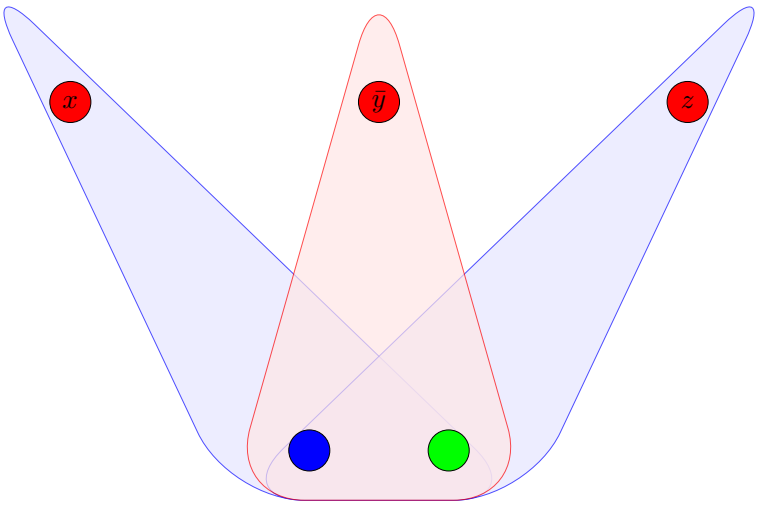
به ازای هر متغیر در فرمول مانند x شکل زیر را رسم می‌کنیم.



تعداد نقاط درونی شکل برابر $2n_x$ است که n_x تعداد دفعات ظاهر شدن x در فرمول است.

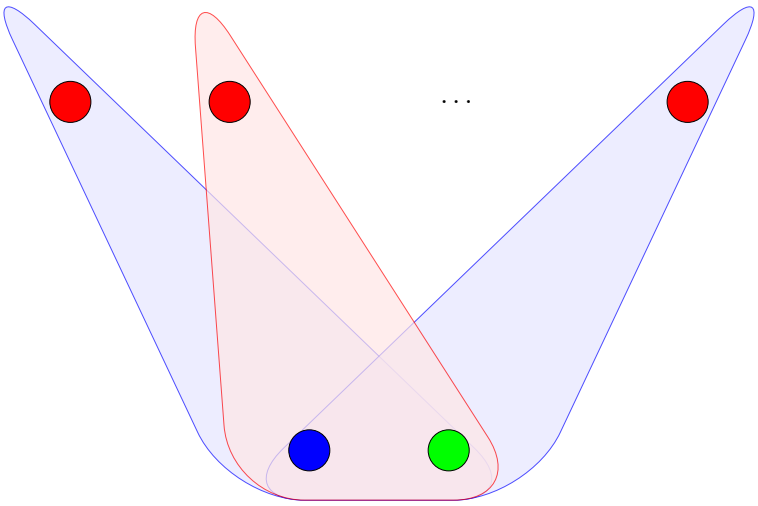
نقاط قرمز اعضای X ، نقاط آبی اعضای Y و نقاط سبز اعضای Z هستند. مثلث‌ها اعضای T هستند که هرکدام متشکل از 3 نقطه با رنگ‌های متمایز هستند. اگر برای متغیر x مثلث‌های آبی را انتخاب کنیم یعنی متغیر x ، true است و اگر قرمزها را انتخاب کنیم یعنی متغیر x ، false است. پس رأس‌های قرمز مثلث‌های قرمز متناظر با x هستند و رأس‌های قرمز مثلث‌های آبی متناظر با $\bar{x}$ هستند. یعنی اگر x ، true باشد رئوس متناظر با x قابل دسترسی هستند و اگر x ، false باشد رئوس متناظر با $\bar{x}$ قابل دسترسی اند.

به ازای هر عبارت مانند $(x \vee \bar{y} \vee z)$ ابزارک شماره ۱ را مانند زیر رسم می‌کنیم.



طریقه ساخت این ابزارک به این صورت است که دو رأس با رنگ‌های سبز و آبی رسم می‌کنیم و و این دو را با یکی از رأس‌های x که قبلاً انتخاب نشده‌اند در یک دسته قرار می‌دهیم و این عمل را برای $\bar{y}$ و z نیز انجام می‌دهیم.

به ازای هر ابزارک باید یکی از مثلث‌ها انتخاب شوند تا رئوس آبی و سبز آن پوشیده بشود و هر ابزارک یکی از متغیرهای عبارت مربوط به خودش را می‌پوشاند پس جمعاً ($\sum n_{x_i}$ − (تعداد عبارت‌ها)) از از رئوس بدون پوشش باقی خواهند ماند. برای این رئوس ابزارک‌های زباله روب[۵] را مانند زیر را رسم می‌کنیم.



این ابزارک متشکل از یک رأس آبی و یک رأس سبز و تمامی رئوس بدون پوشش است.
از این ابزارک به تعداد ($\sum n_{x_i}$ − (تعداد عبارت‌ها)) رسم می‌کنیم تا بتوانند تمام رئوس باقی‌مانده را بپوشانند.

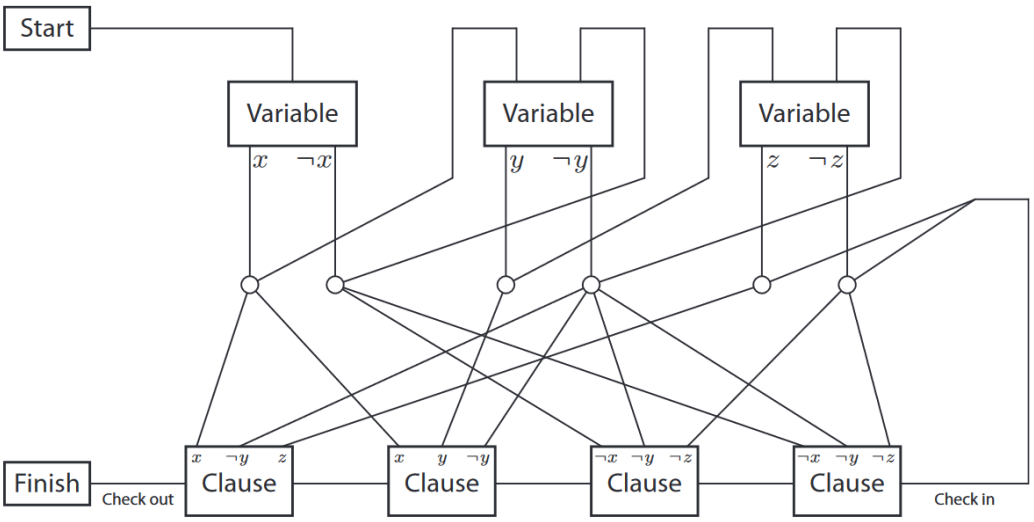
در نهایت یک پوشش از نقاط داریم اگر و تنها اگر فرمول مسئله ۳_صدق پذیری ارضا پذیر باشد، پس مسئله ۳_صدق پذیری را به این مسئله تحویل کردیم و چون این مسئله به وضوح انپی است پس نتیجه می‌شود که انپی_تمام نیز است.

۴ سوپر ماریو[۶]

قضیه ۱. مسئله‌ی سوپر ماریو، انپی_سخت است.
ورودی: یک نقشه از بازی و نقطه‌ی شروع و نقطه‌ی هدف.
خروجی: آیا از نقطه‌ی شروع می‌توان به نقطه‌ی هدف رسید؟

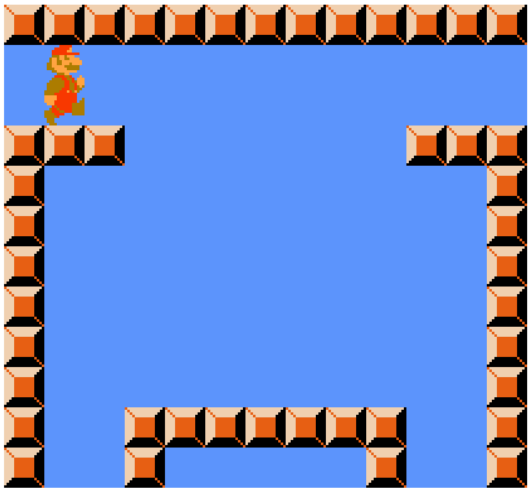
اثبات. یک تحویل چندجمله‌ای از مسئله‌ی ۳_صدق پذیری به بازی سوپر ماریو ارایه می‌دهیم. تناظر فضای بازی با مسئله‌ی ۳_صدق پذیری را می‌توان در شکل زیر خلاصه کرد:

[5] GarbageCollection
[6] Super Mario



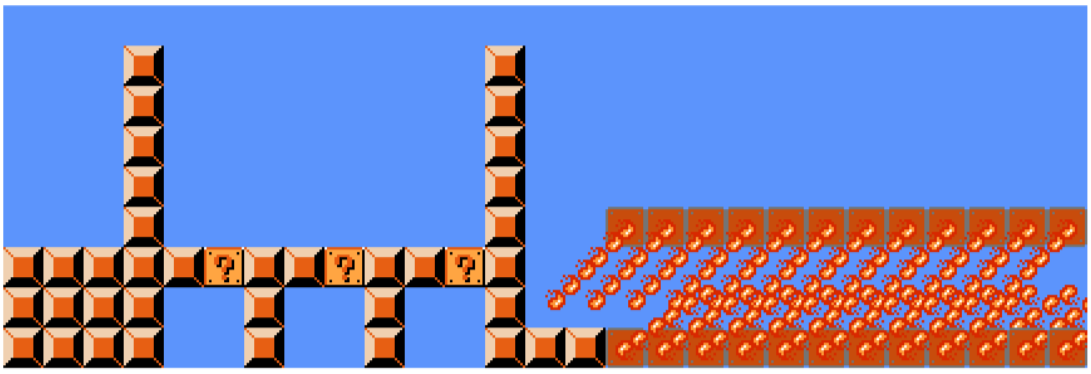
مانند قسمت‌های قبل برای هر متغیر و برای هر پرانتز یک ابزار ارائه می‌دهیم. هر ابزار قسمتی از نقشه‌ی بازی است. متغیر دلخواه x_i را در نظر بگیرید. از ابزار متغیر x_i می‌توان به ابزار متغیر x_{i+1} رفت و همچنین اگر x_i برابر با true باشد، می‌توان به ابزار پرانتزهای شامل x_i و در غیر این صورت به ابزار پرانتزهای شامل $\bar{x}_i$ رفت. ابزار هر پرانتز نیز تنها یک راه خروج دارد و این راه خروج زمانی باز می‌شود که حداقل یکی از عناصر پرانتز مقدار true داشته باشد. طبق شکل، ماریو زمانی می‌تواند به نقطه‌ی پایان برسد که از همه‌ی پرانتزها عبور کرده باشد، یعنی مقدار همه‌ی پرانتزها برابر با true باشد. در صورتی که مقدار حداقل یکی از پرانتزها برابر با false باشد، راهی برای خروج از آن پرانتز وجود نداشته و ماریو برای همیشه در آن قسمت از بازی باقی می‌ماند. با توجه به این که بازی در فضای دوبعدی است، مسیرهای بین ابزارهای مختلف ممکن است با هم تقاطع داشته باشند، برای پیشگیری از تغییر مسیر در قاطع‌ها، یک ابزار برای تقاطع‌ها نیز ارائه می‌شود.

ابزار برای متغیرها:



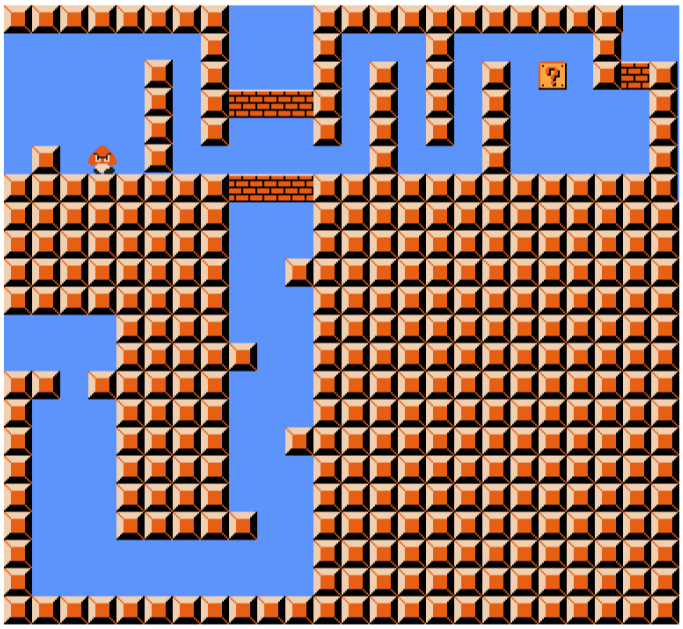
فرض کنید که این ابزار برای متغیر x_i است. دو ورودی وجود دارند که یکی از آن‌ها از x_{i-1} و دیگری از $\bar{x}_{i-1}$ است. زمانی که ماریو از ورودی به پایین می‌افتد، امکان بازگشت به بالا را ندارد و در نتیجه نمی‌تواند به متغیر قبلی و به طور خاص نقیض متغیر قبلی باز گردد. برای تعیین این که مقدار متغیر x_i چه باشد، ماریو یا به چپ (false) یا به راست (true) می‌رود.

ابزار برای پرانتزها:



سه ورودی در قسمت پایین معادل سه عنصر داخل پرانتز هستند. زمانی که ماریو به پرانتز می‌رسد، به جعبه‌ی اشیاء بر می‌خورد و پس از برخورد، یک ستاره در بالای ورودی‌ها پدیدار می‌شود. بعد از این که ماریو ستاره را گرفت، می‌تواند بدون آسیب از میان آتش که در سمت راست شکل قرار دارد عبور کند. بدون ستاره ماریو امکان عبور از این قسمت را ندارد. پس ماریو تنها زمانی از پرانتز می‌تواند عبور کند که از طریق درگاه معادل با یکی از عناصر در پرانتز وارد شود، یعنی مقدار متناظر با آن عنصر true باشد.

ابزار برای نقاط تقاطع:



اگر ماریو از سمت چپ وارد شود، با هیولا برخورد کرده و به دنبال آن، کوچک می‌شود. با کوچک شدن ماریو، او می‌تواند از دالان‌های تنگ سمت راست تصویر عبور کرده و به جعبه اشیا برسد و با برخورد با آن قدرتش ارتقا پیدا کرده و بتواند دیوار آجری را شکسته و از تقاطع عبور کند. به دلیل کوچک بودن ماریو در این قسمت، او نمی‌تواند دیوارهای آجری بزرگ را بشکند و از خروجی بالای تقاطع خارج شود. اگر ماریو از سمت پایین وارد تقاطع شود، با توجه به اندازه‌ی خود می‌تواند دیوارهای آجری را شکسته و عبور کند. با توجه به این که

ورودی به تقاطع از سمت پایین یک‌طرفه است، ماریو می‌تواند دیوارها را بشکاند ولی نمی‌تواند از راست خارج شود چون اندازه‌اش کوچک است پس فقط می‌تواند از بالا خارج شود.

به این ترتیب تحویل چندجمله‌ای از ۳_صدق پذیری به بازی سوپر ماریو وجود دارد. ولی چون نمی‌دانیم که سوپر ماریو، انپی است یا نه هنوز ثابت نشده که سوپر ماریو، انپی_ تمام است.
درواقع ثابت شده است که سوپر ماریو، کلاسی بزرگتر از انپی است به نام پی‌اسپیس_ تمام[۷] است. این کلاس مسائلی هستند که در حافظه چندجمله‌ای حل می‌شوند و دسته بزرگتری از انپی است. □

مراجع

[1] Erik Demaine. *Design and Analysis of Algorithms*. Lecture 16, MIT Open Courseware, 2015. URL: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2015/lecture-notes/$MIT6_046JS15_writtenlec16.pdf$

[7] PSPACE-Complet

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۱۹: مسائل انپی_ تمام[1] ۲

نگارنده: علیرضا دادگرنیا

در جلسهٔ پیش ابتدا درباره مسئلهٔ صدق‌پذیری مدار[2]، که می‌توان آن را به عنوان اولین مسئله‌ای که انپی_تمام بودن آن اثبات شده است مطرح کرد، صحبت کردیم و مقداری نیز راجع به اثبات آن بحث کردیم. سپس مسئلهٔ صدق‌پذیری مدار را به مسئلهٔ ۳_صدق‌پذیری[3] تحویل کردیم و با استفاده از مسئلهٔ ۳_صدق‌پذیری، انپی_تمام بودن تعدادی مسئله را اثبات کردیم.

۱ مسائل صدق‌پذیری

تا به حال چند حالت مختلف از مسائل صدق‌پذیری را دیده‌ایم مانند صدق‌پذیری مدار، ۲_صدق‌پذیری و ۳_صدق‌پذیری. مسئلهٔ ۳_صدق‌پذیری را می‌توان به مسئلهٔ k_صدق‌پذیری تعمیم داد که مشابه مسئلهٔ ۳_صدق‌پذیری تعریف می‌شود تنها با این تفاوت که در هر پرانتز حداکثر k متغیر وجود دارد. حالت کلی‌تر این مسئله نیز مسئلهٔ صدق‌پذیری است که محدودیتی روی تعداد متغیرهای داخل پرانتز وجود ندارد. از آن‌جا که مسئلهٔ ۳_صدق‌پذیری انپی_تمام است حالت‌های کلی‌تر آن نیز انپی_تمام هستند. همچنین می‌دانیم مسئلهٔ ۲_صدق‌پذیری الگوریتم چندجمله‌ای دارد پس داخل $\mathbf{P}$ است. حالا تعدادی مسئلهٔ صدق‌پذیری جدید را معرفی می‌کنیم.

Horn SAT. این مسئله مانند حالت کلی مسئلهٔ صدق‌پذیری تعریف می‌شود تنها با این تفاوت که داخل هر پرانتز حداکثر یک متغیر مثبت و به تعداد دلخواهی متغیر منفی وجود دارد (منظور از متغیر مثبت x و منظور از متغیر منفی $\overline{x}$ است).
در جلسات گذشته دیدیم که یک راه برای حل مسئلهٔ ۲_صدق‌پذیری استفاده از گراف قویاً همبند بود. راه دیگری نیز وجود دارد که چندان وارد جزئیات آن نمی‌شویم. می‌توانیم یکی از متغیرها، به عنوان مثال x_1، را True در نظر بگیریم و ببینیم به چه نتایجی منجر می‌شود. از آن‌جا که می‌خواهیم حاصل هر پرانتز True شود اگر $\overline{x_1}$ در یکی از پرانتزها آمده باشد، از آن‌جا که مقدار آن False است، متغیر دیگری که در آن پرانتز وجود دارد باید True شود و به همین ترتیب مقدار تعدادی از متغیرها مشخص می‌شود. اگر در این فرایند مقدار یک پرانتز False شد به این نتیجه می‌رسیم که انتخاب اولیه اشتباه بوده و x_1 باید False باشد. اگر همین روند را ادامه دهیم به یک الگوریتم چندجمله‌ای منجر می‌شود. پیچیدگی زمانیِ الگوریتم مشابه برای مسئلهٔ ۳_صدق‌پذیری نمایی است و الگوریتم‌های با زمان بهتر اما همچنان نمایی، برای مسئلهٔ ۳_صدق‌پذیری وجود دارند. با ایده‌ای مشابه می‌توان صدق‌پذیری هورن را در زمان چندجمله‌ای حل کرد که نتیجه می‌دهد صدق‌پذیری هورن عضو $\mathbf{P}$ است.

DNF SAT. در این مسئله در هر پرانتز بین همهٔ متغیرها and و بین پرانتزها or قرار دارد و هدف این است که ببینیم آیا یک مقداردهی وجود دارد که حاصل عبارت True شود یا خیر. مانند عبارت زیر

$$(x \wedge y \wedge z) \vee (\overline{x} \wedge y) \vee (y \wedge \overline{z})$$

که در آن اگر هر سه متغیر True باشند حاصل نیز True می‌شود.
مسئلهٔ DNF SAT نیز عضو $\mathbf{P}$ است، زیرا تنها کافی است که مقدار یک پرانتز True شود و اگر در یک پرانتز x و $\overline{x}$ همزمان وجود نداشته

[1] **NP**-Complete
[2] Circuit SAT
[3] 3-SAT

باشند می‌توان با مقداردهی مناسب آن پرانتز را True کرد. در غیر این‌صورت در هر پرانتز مثبت و منفیِ یک متغیر همزمان وجود دارند و مقدار همهٔ پرانتزها False می‌شود. لازم به ذکر است که به حالت معمولی مسئلهٔ صدق‌پذیری CNF SAT نیز گفته می‌شود.

حالت‌های دیگری از مسئلهٔ صدق‌پذیری وجود دارد که آن‌ها انپی_کامل هستند مانند ۳_صدق‌پذیری_۵ و Monotone 3-SAT. تفاوت مسئلهٔ ۳_صدق‌پذیری_۵ با مسئلهٔ ۳_صدق‌پذیری این است که هر متغیر حداکثر ۵ بار ظاهر می‌شود و تفاوت مسئلهٔ Monotone 3-SAT با مسئلهٔ ۳_صدق‌پذیری نیز این است که همهٔ متغیرهای داخل یک پرانتز باید مثبت یا منفی باشند.

یک‌سری دیگر از مسائل صدق‌پذیری وجود دارند که نمی‌توان گفت حالت خاص ۳_صدق‌پذیری هستند و در واقع یک گونهٔ[۴] دیگری از آن هستند. این مسائل، هم مشهور هستند، هم در مسائل انپی_تمامیت و تحویل‌پذیری کاربرد زیادی دارند.

1 in 3-SAT یا Exactly 1 in 3-SAT. در این مسئله تعدادی پرانتز که در هر کدام ۳ متغیر وجود دارند داریم که بین آن‌ها or قرار دارد. مقدار یک پرانتز True است، اگر و تنها اگر دقیقا یک متغیر در آن پرانتز True باشد. حالت خاص این مسئله Monotone 1 in 3-SAT است که همهٔ متغیرها مثبت یا منفی‌اند.

NAE 3-SAT (Not All Equal 3-SAT). در این مسئله تعدادی پرانتز که در هر کدام ۳ متغیر وجود دارند داریم که بین آن‌ها or قرار دارد. مقدار یک پرانتز True است، اگر و تنها اگر حداقل یک متغیر True و حداقل یک متغیر False در آن پرانتز وجود داشته باشد. هر پرانتز را با (x, y, z) نشان می‌دهیم.
می‌خواهیم نشان دهیم مسئلهٔ NAE 3-SAT انپی_تمام است. ثابت می‌کنیم 3-SAT $\leq_{\mathbf{P}}$ NAE 3-SAT. شاید ابتدا فکر کنید با استفاده از رابطهٔ

$$(x, y, z) = (x \vee y \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$

حکم واضح است اما ما باید برعکس این را انجام دهیم یعنی 3-SAT را به NAE 3-SAT تحویل کنیم. عبارت زیر را در نظر بگیرید

$$\Phi = (x_1 \vee y_1 \vee z_1) \wedge (x_2 \vee y_2 \vee z_2) \wedge \cdots \wedge (x_m \vee y_m \vee z_m)$$

باید عبارت Φ' را به شکلی بسازیم که Φ' در NAE 3-SAT، True شود اگر و تنها اگر Φ در 3-SAT، True شود. این کار را در دو مرحله انجام می‌دهیم. ابتدا ثابت می‌کنیم 3-SAT $\leq_{\mathbf{P}}$ NAE 4-SAT سپس نشان می‌دهیم NAE 3-SAT $\leq_{\mathbf{P}}$ NAE 4-SAT. تعریف می‌کنیم

$$\Phi' = (x_1, y_1, z_1, s) \wedge (x_2, y_2, z_2, s) \wedge \cdots \wedge (x_m, y_m, z_m, s)$$

یک مقداردهی ارضاکننده برای Φ در نظر بگیرید یعنی در هر پرانتز حداقل یک متغیر True شده است. پس اگر قرار دهیم $s = \text{False}$ در هر پرانتز Φ' حداقل یک مقدار True و حداقل یک مقدار False داریم در نتیجه مقدار نهایی آن True می‌شود. برعکس فرض کنید یک مقداردهی ارضاکننده برای Φ' داشته باشیم. دقت کنید که مسئلهٔ NAE SAT بین True و False تقارن دارد یعنی برای یک مقداردهیِ ارضا کننده اگر همهٔ مقادیر متغیرها را برعکس کنیم (True به False و False به True) مقداردهی جدید نیز ارضاکننده است. در واقع ما از مفهوم True و False استفاده‌ای نمی‌کنیم و می‌توانیم به جای آن‌ها از دو رنگ آبی و قرمز استفاده کنیم و مسئله را با این دو رنگ تعریف کنیم. در نتیجه می‌توانیم فرض کنیم $s = \text{False}$. از آن‌جا که در هر پرانتز حداقل یک مقدار True داریم حداقل یکی از x_i، y_i و z_i، True است پس Φ نیز True می‌شود و می‌توانیم نتیجه بگیریم ۳_صدق‌پذیری به NAE 3-SAT تحویل‌پذیر است. در نهایت باید نشان دهیم NAE 3-SAT $\leq_{\mathbf{P}}$ NAE 4-SAT. تبدیل زیر را در نظر بگیرید.

$$(x, y, z, w) \Longrightarrow (x, y, s) \wedge (z, w, \overline{s}) \tag{۱}$$

فرض کنید سمت چپ تبدیل (۱)، True باشد. بین x، y، z و w حداقل یک مقدار True و یک مقدار False وجود دارد. اگر $x \neq y$ مقدار (x, y, s)، True می‌شود و به همین ترتیب اگر $z \neq w$ مقدار $(z, w, \overline{s})$ نیز True می‌شود پس بنابر تقارن فرض می‌کنیم $x = y = \text{True}$ (دقت کنید که مسئلهٔ NAE SAT بین True و False تقارن دارد) و قرار می‌دهیم $s = \text{False}$. از آن‌جا که مقدار حداقل یکی از z و

[۴] variant

w، False است هر دو پرانتز (x, y, s) و $(z, w, \overline{s})$، True می‌شوند. برعکس فرض کنید سمت راست تبدیل (۱)، True شود. اگر $x = y = z = w$، یکی از دو پرانتز (x, y, s) و $(z, w, \overline{s})$، False می‌شوند که تناقض است پس دو مقدار مختلف بین آن‌ها وجود دارد که نتیجه می‌دهد (x, y, z, w)، True است. در کل نتیجه می‌شود NAE 4-SAT به NAE 3-SAT تحویل‌پذیر است و انپی‌ـ تمام بودن NAE 3-SAT ثابت می‌شود.

انپی‌ـ تمام بودن تعدادی از مسائل کلاسیک را می‌توان با استفاده از مسئلهٔ NAE 3-SAT اثبات کرد که در قسمت تمرین‌ها به آن‌ها اشاره خواهد شد.

۲ مسئلهٔ Subset Sum

در این بخش انپی‌ـ تمام بودن مسئلهٔ Subset Sum را اثبات می‌کنیم. ورودی این سوال مجموعهٔ n عضوی $A = \{a_1, a_2, \ldots, a_n\}$ و عدد t است و باید بررسی کنیم که آیا مجموعهٔ $S \subseteq A$ وجود دارد که $\sum S = t$ (منظور از $\sum S$ همان $\sum_{x \in S} x$ است).

دقت کنید که اگر مجموعهٔ S به ما داده شود در زمان چندجمله‌ای می‌توانیم بررسی کنیم که خاصیت مسئله را دارد یا خیر پس مسئلهٔ Subset Sum داخل انپی است. در جلسهٔ قبل راجع‌به مسئلهٔ تطابق ۳ـ بعدی[۵] یا به اختصار 3-DM صحبت کردیم و نشان دادیم که انپی‌ـ تمام است. حالا می‌خواهیم این مسئله را به مسئلهٔ Subset Sum تحویل کنیم. فرض کنید مجموعهٔ $B = \{x_1, x_2, \ldots, x_m\}$ و سه‌تایی‌های (x_i, x_j, x_k) از این مجموعه به ما داده شده باشند. فرض کنید n_i تعداد دفعات ظاهر شدن x_i در سه‌تایی‌های داده شده باشد. به هر کدام از سه‌تایی‌ها مانند (x_i, x_j, x_k) عدد

$$a_{i,j,k} = \left(\underbrace{0 \ldots 0 \underset{i}{1} 0 \ldots 0 \underset{j}{1} 0 \ldots 0 \underset{k}{1} 0 \ldots 00}_{m} \right)_{1+\max\limits_{s} n_s}$$

را نظیر می‌کنیم که $1 + \max\limits_{s} n_s$ مبنای آن است. دقت کنید که لزوماً ترتیب i، j و k به شکل بالا نیست. همچنین قرار می‌دهیم

$$t = \left(\underbrace{11 \ldots 11}_{m} \right)_{1+\max\limits_{s} n_s}$$

حالا واضح است که اگر هر کدام از اعضای مجموعهٔ B دقیقاً یک‌بار در سه‌تایی‌ها ظاهر شده باشد مجموع همهٔ $a_{i,j,k}$ها برابر با t می‌شود و برعکس اگر مجموع همهٔ $a_{i,j,k}$ها برابر با t شود، از آن‌جا که مبنا را برابر با $1 + \max\limits_{s} n_s$ قرار داده‌ایم در مجموع هیچ دهبَرِیکی اتفاق نمی‌افتد پس هر عضو مجموعهٔ B باید دقیقاً در یکی از سه‌تایی‌ها آمده باشد. در نتیجه مسئلهٔ Subset Sum به مسئلهٔ 3-DM تحویل می‌شود.

حالا مسئلهٔ Partition را تعریف می‌کنیم و ثابت می‌کنیم مسئلهٔ Subset Sum را می‌توان به آن تحویل کرد. مسئلهٔ Partition حالت خاص مسئلهٔ Subset Sum است و تفاوتی که با آن دارد این است که تنها مجموعهٔ $C = \{c_1, c_2, \ldots, c_n\}$ داده شده است و ما به دنبال مجموعهٔ $S \subseteq C$ می‌گردیم که $\sum S = \frac{\sum C}{2}$.

فرض کنید مجموعهٔ $A = \{a_1, a_2, \ldots, a_n\}$ و عدد t به ما داده شده باشد. قرار می‌دهیم $\sum A = \sigma$ و مجموعهٔ A' را به شکل زیر تعریف می‌کنیم

$$A' = \{a_1, a_2, \ldots, a_n, a_{n+1}, a_{n+2}\}, \quad a_{n+1} = \sigma + t, \quad a_{n+2} = 2\sigma - t$$

دقت کنید که

$$\sum A' = \sum A + a_{n+1} + a_{n+2} = 4\sigma. \tag{۲}$$

[۵] 3-Dimensional Matching

اگر $S \subseteq A$ وجود داشته باشد که $\sum S = t$، قرار می‌دهیم $S' = S \cap \{a_{n+2}\}$. حالا داریم

$$\sum S' = \sum S + a_{n+2} = 2\sigma \overset{(2)}{=} \frac{\sum A'}{2}$$

پس مجموعهٔ S' یک جواب برای مسئلهٔ Partition است. برعکس اگر مجموعهٔ $S' \subseteq A'$ وجود داشته باشد که $\sum S' = \frac{\sum A'}{2}$، تعریف می‌کنیم $\overline{S'} = A' \backslash S'$. واضح است که $\sum \overline{S'} = \frac{\sum A'}{2}$. دقت کنید که از بین دو عدد a_{n+1} و a_{n+2} یکی از آن‌ها باید در S' و دیگری در $\overline{S'}$ باشد زیرا

$$a_{n+1} + a_{n+2} = 3\sigma > 2\sigma \overset{(2)}{=} \frac{\sum A'}{2}.$$

بنابر تقارن فرض می‌کنیم $a_{n+1} \in S'$ و قرار می‌دهیم $S = S' \backslash \{a_{n+1}\}$. در نهایت داریم

$$\sum S = \sum S' - a_{n+1} = t$$

پس طرف دیگر هم اثبات شد و نتیجه می‌گیریم که مسئلهٔ Subset Sum به مسئلهٔ Partition تحویل‌پذیر است.

۳ مسئلهٔ Rectangle Packing

در این بخش مسئلهٔ Rectangle Packing را بررسی می‌کنیم. یک مستطیل بزرگ و تعدادی مستطیل کوچک‌تر داده شده است و سوال این است که آیا می‌توان مستطیل بزرگ را با مستطیل‌های کوچک پوشاند (مستطیل‌های کوچک نباید به جز نقاط مرزی با هم اشتراک داشته باشند). چرخش نود درجه برای مستطیل‌ها مجاز است.

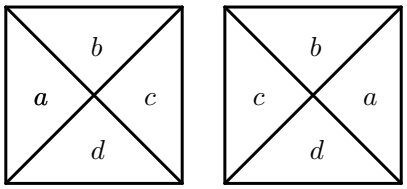
نکتهٔ بدیهی این است که باید جمع مساحت مستطیل‌های کوچک با مساحت مستطیل بزرگ برابر باشد اما این شرط کافی نیست. دقت کنید که این مسئله انپی است زیرا اگر مکان مستطیل‌ها و میزان چرخش آن‌ها را داشته باشیم می‌توانیم در زمان چندجمله‌ای بررسی کنیم که مستطیل بزرگ پوشانده می‌شود یا خیر. نشان می‌دهیم مسئلهٔ Partition به این مسئله تحویل‌پذیر است که انپی_تمام بودن آن را نشان می‌دهد. فرض کنید مجموعهٔ $A = \{a_1, a_2, \dots, a_n\}$ داده شده است. مستطیل بزرگ را یک مستطیل $2 \times \frac{\sum A}{2}$ و مستطیل‌های کوچک را برای هر $1 \leq i \leq n$، $1 \times a_i$ در نظر می‌گیریم. اگر مجموعهٔ $S \subseteq A$ وجود داشته باشد که $\sum S = \frac{\sum A}{2}$ کافی است مستطیل نظیر a_iهایی که عضو S هستند را در نیمهٔ بالای مستطیل بزرگ و به ترتیب قرار دهیم و مستطیل نظیر a_iهایی که عضو S نیستند را نیز در نیمهٔ پایین مستطیل بزرگ قرار دهیم و یک طرف حکم اثبات می‌شود. اما طرف دیگر حکم را نمی‌توانیم نشان دهیم زیرا ممکن است مستطیلی نود درجه چرخیده باشد و به‌طور عمودی قرار گرفته باشد. برای حل این مشکل طول همهٔ مستطیل‌های کوچک و مستطیل بزرگ را ۳ برابر می‌کنیم. در این‌صورت طرف اول حکم مشابه قبل اثبات می‌شود و مستطیل‌ها دیگر نمی‌توانند به‌طور عمودی قرار بگیرند زیرا طول آن‌ها حداقل ۳ است اما عرض مستطیل بزرگ ۲ است. حالا کافی است a_iِ نظیر مستطیل‌هایی که در نیمهٔ بالای مستطیل بزرگ قرار دارند را در مجموعهٔ S قرار دهیم و حکم نتیجه می‌شود.

۴ قویاً انپی_تمامیت[۶]

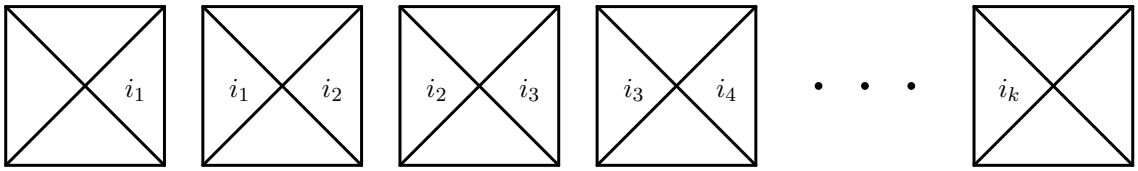
ابتدا قصد داریم انواعی از پازل ها را بررسی کنیم.

نوعی از پازل که این‌جا می‌خواهیم آن را بررسی کنیم به این‌صورت است که تکه‌های آن مربعی شکل و 1×1 است و هر ضلعِ هر تکهٔ آن یک رنگ دارد. دو تکه از پازل را در صورتی می‌توانیم کنار یک‌دیگر قرار دهیم که ضلع‌هایی که با یک‌دیگر تماس پیدا می‌کنند، همرنگ باشند. به عنوان مثال دو تکهٔ زیر می‌توانند کنار هم قرار گیرند:

[۶] Strong NP-Completeness



این نوع از پازل‌ها از قرن نوزدهم میلادی وجود داشته‌اند و یکی از انواع معروف آن‌ها نیز پازل Eternity است که درست کردن آن دو میلیون دلار جایزه داشته است (لازم به ذکر است که حل مسئلهٔ $\mathbf{P}$ vs $\mathbf{NP}$ یک میلیون دلار جایزه دارد).
حالا فرض کنید قطعات یک پازل به‌هم‌ریخته به ما داده شده است. می‌خواهیم ثابت کنیم که حل این نوع از پازل یک مسئله انپی_تمام است. در واقع مسئله به این‌صورت است که تعدادی از این مربع‌ها و یک مستطیل بزرگ به ما داده شده است و سوال این است که آیا می‌توان این تکه‌ها را در مستطیل بزرگ، با شرط گفته شده، قرار داد (رنگ وجه اشتراک مربع‌ها با مرز مستطیل شرطی ندارد).
این مسئله تا حدودی شبیه به مسئلهٔ قبل است. ابتدا سعی می‌کنیم مسئلهٔ Partition را به این مسئله تحویل کنیم. مانند مسئلهٔ Rectangle Packing به ازای هر i، $3a_i$تا مربع 1×1 درست می‌کنیم، مستطیل بزرگ را هم یک مستطیل $2 \times \frac{3\sum A}{2}$ در نظر می‌گیریم و می‌خواهیم رنگ مربع‌ها به شکلی باشد که مجبور باشند در مستطیل بزرگ کنار هم قرار بگیرند. می‌توانیم مربع‌ها را به شکلی بسازیم که دقیقا دو رنگ i_1، دو رنگ i_2 و ... داشته باشیم. مانند شکل زیر:



مشکلی که این‌جا به وجود می‌آید این است که اگر دو مربع داریم که رنگ i_1 در آن‌ها ظاهر شده است، باید مطمئن شویم که این رنگ وجه اشتراک مربع با مستطیل بزرگ نباشد. مشکل دیگر این است که برای هر i باید $3a_i$ تا مربع کوچک تولید می‌کردیم و ورودی مسئلهٔ پازل، همهٔ این مربع‌ها هستند و هر کدام از این مربع‌ها باید به طور جداگانه در این مسئله به عنوان ورودی به طور کامل تعریف شوند اما در این حالت تعداد مربع‌ها ممکن است نمایی باشد. پس این روش کار نمی‌کند و باید روش دیگری را در پیش بگیریم.
در حقیقت مشکل اصلی این است که مسئلهٔ Partition که از آن به عنوان مسئلهٔ مرجع استفاده کردیم به طور ضعیف انپی_تمام است. یعنی اگر عددها کوچک باشند، می‌توانیم مسئله را حل کنیم ولی اگر مقدار اعداد نمایی باشد، در حل آن دچار مشکل می‌شویم.

در مقابل آن مفهومی به نام قویاً ان پی_تمام وجود دارد.

مسئلهٔ قویاً انپی_تمام. یک مسئله قویاً انپی_تمام است اگر در صورتی که اعداد ورودی آن را به صورت یکانی بنویسیم، باز هم انپی_تمام بماند.

یکانی نوشتن یعنی عدد مورد نظر را، به صورت تعدادی یک نمایش دهیم. مثلا به جای 6 قرار دهیم 111111. در این‌صورت تعداد حافظه‌ای که برای نمایش یک عدد نیاز است، به اندازهٔ مقدار آن عدد است.

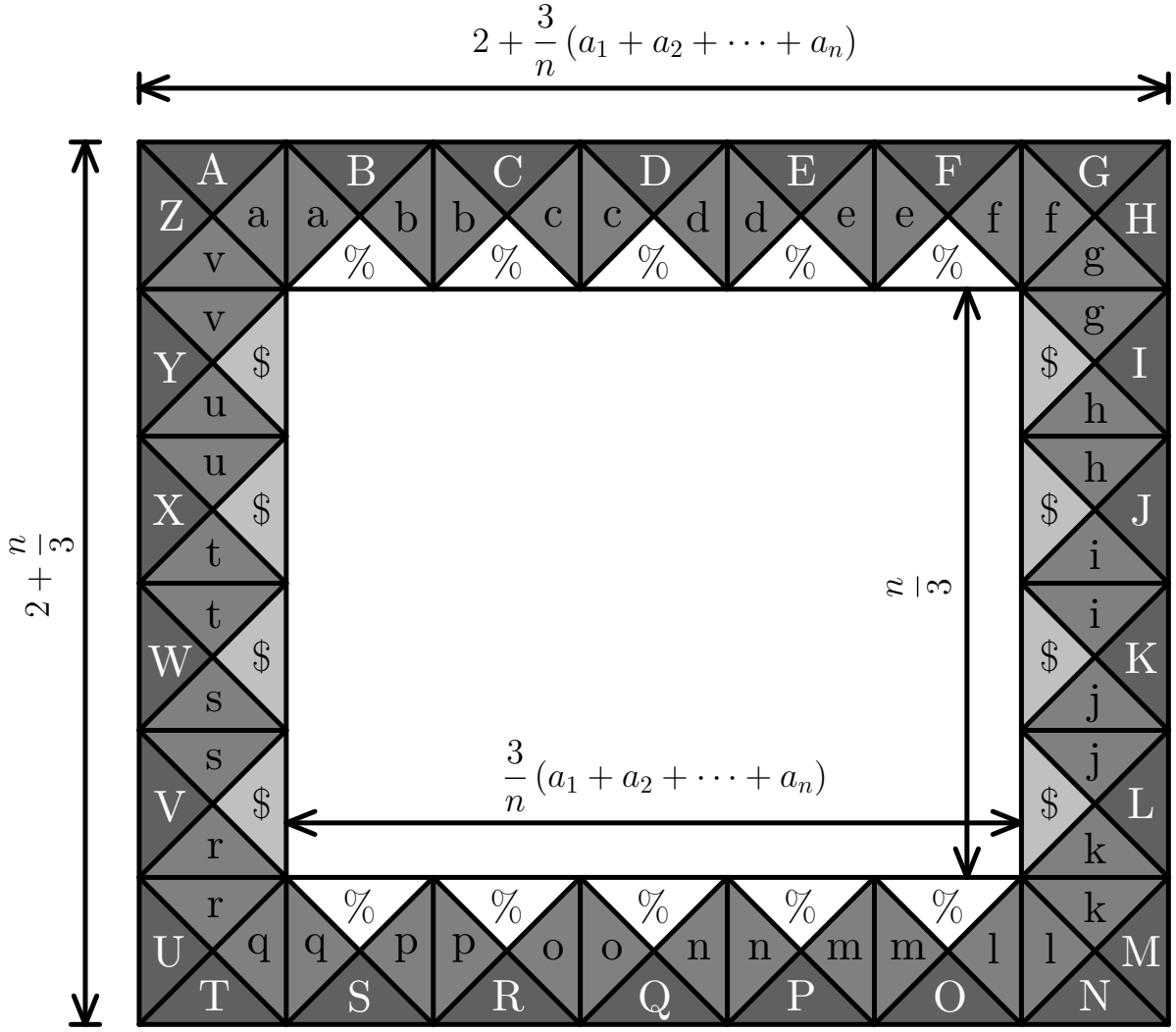
یک مسئله که قویاً انپی_تمام است و از آن زیاد استفاده می‌شود مسئلهٔ 3-Partition است که ورودی آن مجموعهٔ $A = \{a_1, a_2, \ldots, a_n\}$ است و سوال این است که آیا می‌توان مجموعهٔ A را به مجموعه‌های $A_1, A_2, \ldots, A_{\frac{n}{3}}$ افراز کرد طوری‌که

$$\forall 1 \le i \le \frac{n}{3} : \sum A_i = \frac{\sum A}{n/3} = t, \quad \forall 1 \le i \le n : a_i \in \left(\frac{t}{4}, \frac{t}{2}\right).$$

دقت کنید که شرط $a_i \in \left(\frac{t}{4}, \frac{t}{2}\right)$ نتیجه می‌دهد هر کدام از A_iها باید دقیقاً سه عضو داشته باشند، زیرا اگر یکی از آن‌ها کم‌تر از ۳ عضو

داشته باشد، مجموع اعضای آن اکیداً کم‌تر از t می‌شود و اگر بیش‌تر از ۳ عضو داشته باشد مجموع اعضای آن اکیداً بیش‌تر از t می‌شود. اثبات قویاً ان پی‌ـ تمام بودن این مسئله را این‌جا بررسی نمی‌کنیم و در [۱] آمده است.

قبلاً دیدیم که مسئلهٔ پازل‌ها را نمی‌توان با استفاده از مسئلهٔ Partition تحویل کرد اما می‌خواهیم نشان دهیم که این کار با مسئلهٔ 3-Partition امکان‌پذیر است. مشکلی که با آن مواجه شدیم این بود که نمی‌توانستیم مربع‌ها را در زمان چندجمله‌ای بسازیم اما این‌جا دیگر این مشکل را نداریم و برای هر $1 \leq i \leq n$، a_i $(\frac{n}{3}+1)$ تا مربع 1×1 می‌سازیم. یک مشکل دیگر این بود که مطمئن شویم مربع‌هایی که ساختیم در مرز مستطیل بزرگ قرار نگیرند. این مشکل را می‌توانیم با تولید چند مربع دیگر که فقط می‌توانند در مرز قرار گیرند حل کنیم. این مربع‌ها باید یک‌طرفشان رنگی باشد که فقط یک‌بار آمده است و عرض مستطیل بزرگ باید به‌شکلی باشد که بعد از قرار دادن این مربع‌ها عرض مستطیل باقی‌مانده $\frac{n}{3}$ باشد. مانند شکل زیر:



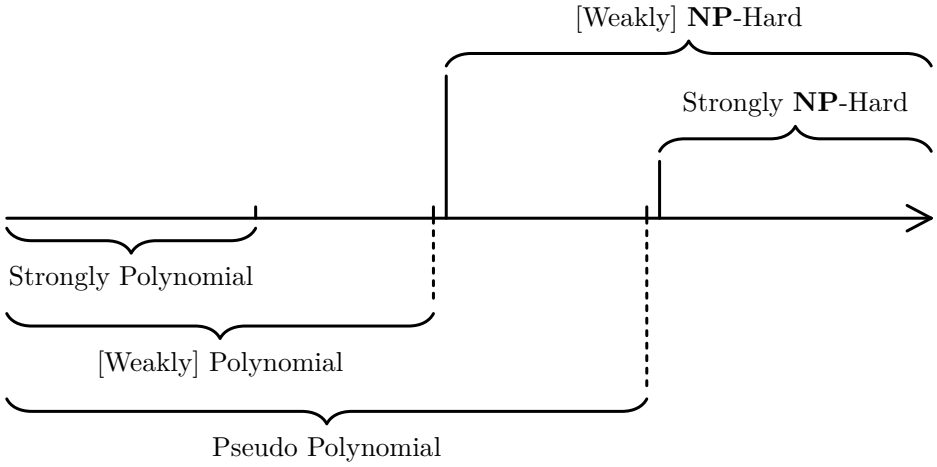
دقت کنید که اگر یک قاب مانند شکل بالا بسازیم دیگر نیازی نیست که a_i $(\frac{n}{3}+1)$ تا مربع بسازیم زیرا می‌توانیم در هر مربع دو رنگ % روبه‌روی هم قرار دهیم و در این‌صورت دیگر مربع‌ها قابل چرخاندن نیستند و با ساختن a_i مربع نیز می‌توانیم به هدف برسیم. شکل بالا نیز با فرض داشتن a_i مربع ساخته شده است و به همین دلیل طول مستطیل داخلی برابر با

$$\frac{3}{n}(a_1+a_2+\cdots+a_n)$$

در نظر گرفته شده است. پس نتیجه‌ای که می‌توانیم بگیریم این است که قویاً ان‌پی‌ـ تمام بودن علاوه بر این‌که نتیجهٔ قوی‌تری است برای

اثبات بعضی مسائل لازم هم هست. برای اثبات انپی_تمام بودن مسئلهٔ پازل در ابتدا سعی کردیم از مسئلهٔ Partition استفاده کنیم و به مشکل خوردیم اما دیدیم که با استفاده از مسئلهٔ 3-Partition این کار به سادگی قابل انجام بود.

با توجه به مفهوم جدید قویاً انپی_تمام بودن که با آن در این جلسه آشنا شدیم می‌توانیم نمودار سختی که قبلاً داشتیم را با جزئیات بیش‌تری رسم کنیم:



با توجه به نمودار، مسائلی که به طور ضعیف انپی_سخت هستند می‌توانند الگوریتم شبهِ چندجمله‌ای داشته باشند. به عنوان مثال مسائل Subset Sum و Partition به طور ضعیف انپی_سخت هستند اما الگوریتم شبه چندجمله‌ای دارند و اگر اعداد ورودی مسئله کوچک باشد چندجمله‌ای محسوب می‌شوند، اما اگر اعداد ورودی بزرگ باشند دیگر الگوریتم بر حسب طول ورودی چندجمله‌ای نیست. در کل برای مسائل انپی_سخت الگوریتمی که بر حسب طول ورودی چندجمله‌ای باشد پیدا نشده است و حدس زده می‌شود که وجود ندارد. در نهایت بین مسائلی که الگوریتم چندجمله‌ای دارند، مسائلی وجود دارند که الگوریتم قویاً چندجمله‌ای دارند. به عنوان مثال در مسئلهٔ شار بیشینه[۷]، الگوریتم فورد_فالکرسون[۸] شبهِ چندجمله‌ای است زیرا در زمان اجرای آن ظرفیت بزرگ‌ترین یال ظاهر می‌شود، الگوریتم Scaling چندجمله‌ای است زیرا در زمان اجرای آن تعداد رئوس، تعداد یال‌ها و لگاریتم بزرگ‌ترین وزن ظاهر می‌شود که بر حسب طول ورودی چندجمله‌ای است و پس از آن نیز الگوریتمی ارائه دادیم که فقط تعداد اجزای ورودی ظاهر شود که این الگوریتم قویاً چندجمله‌ای است.

دقت کنید از آن‌جا که ما با استفاده از مسئلهٔ Partition ثابت کردیم مسئلهٔ Rectangle Packing انپی_تمام است نتیجه می‌شود مسئلهٔ Rectangle Packing انپی_تمام ضعیف است اما با استفاده از مسئلهٔ 3-Partition می‌توان قویاً انپی_تمام بودن آن را اثبات کرد که بسیار شبیه به همان اثبات قبلی است.

۵ تمرین‌ها

۱. در یک گراف به هر یال وزنی مثبت نسبت داده‌ایم و می‌خواهیم برشی را پیدا کنیم که جمع وزن یال‌های بین دو برش بیشینه شود. به این مسئله، برش بیشینه[۹] می‌گوییم. مسئلهٔ NAE 3-SAT را به مسئلهٔ برش بیشینه تحویل کنید.

۲. گراف G داده شده است و می‌خواهیم بررسی کنیم که آیا می‌توان رئوس G را با سه رنگ، رنگ کرد طوری‌که دو سر هیچ یالی تک‌رنگ نباشد. به این مسئله، ۳_رنگ‌آمیزی[۱۰] می‌گوییم. مسئلهٔ NAE 3-SAT را به مسئلهٔ ۳_رنگ‌آمیزی تحویل کنید.

[7] Max Flow
[8] Ford-Fulkerson
[9] Max Cut
[10] 3-Coloring

۳. مسئلهٔ 3-SAT را به مسئلهٔ ۳ـ رنگ‌آمیزی تحویل کنید.

مراجع

[1] Michael R. Garey, David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1st ed., 1979

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

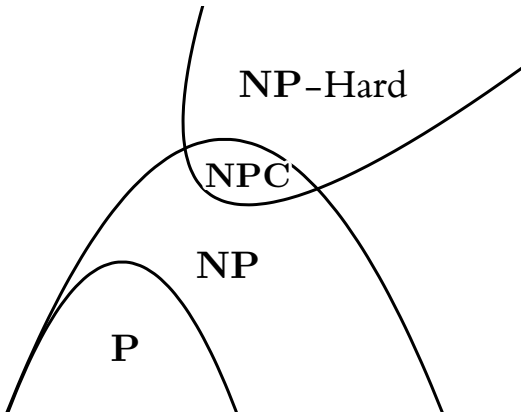
[بهار ۹۹]

جلسه ۲۰: گشت و گذاری در دنیای پیچیدگی (اختیاری) نگارنده: علیرضا دادگرنیا

در جلسهٔ پیش با انواع مختلفی از مسائل صدق‌پذیری آشنا شدیم و انپی‌ـ تمام[1] بودن تعدادی از آن‌ها را اثبات کردیم سپس مسائل دیگری از جمله Subset Sum و Rectangle Packing را بررسی کردیم و با مفهوم قویاً انپی‌ـ تمامیت[2] آشنا شدیم. در این جلسه قرار است اطلاعات بیش‌تری راجع‌به کلاس انپی و کلاس‌هایی که مرتبط با آن هستند کسب کنیم.

۱ کلاس انپی

کلاس انپی، کلاس بسیار مهمی است و در تحلیل پیچیدگی مسائلِ زیادی ظاهر می‌شود. همان‌طور که احتمالاً حدس زده‌اید بخشی از مهم بودن آن به این دلیل است که مسئلهٔ $\mathbf{P}$ vs $\mathbf{NP}$ بسیار مسئلهٔ مهمی است. در ادامه دربارهٔ چرایی مهم بودن این مسئله نیز توضیح خواهیم داد. یک نکتهٔ بسیار جالب این است که اکثر مسائل محاسباتی که با آن‌ها روبه‌رو می‌شویم در کلاس انپی قرار دارند یعنی مستقل از این‌که این مسئله از کدام حیطه آمده باشد، عمدتاً یا می‌توانیم یک الگوریتم چندجمله‌ای برای آن پیدا کنیم، یا می‌توانیم ثابت کنیم که انپی‌ـ سخت[3] یا انپی‌ـ تمام است. تصوری که از این کلاس‌ها داریم مانند شکل زیر است:



اکثر مسائلی که با آن‌ها مواجه می‌شویم، نه تنها عضو کلاس انپی هستند بلکه در یکی از دو قسمت انپی‌ـ تمام یا پی قرار می‌گیرند، به همین علت می‌گوییم یک حالت دو قطبی دارند. حال چند سوال نظری مطرح می‌شود. اول این‌که آیا مسائلی وجود دارند که بین پی و انپی‌ـ تمام قرار بگیرند؟ یعنی عضو کلاس انپی باشند اما نه انپی‌ـ تمام باشند و نه الگوریتم چندجمله‌ای برای آن‌ها وجود داشته باشد. قضیهٔ زیر تا حدی به این سوال جواب می‌دهد:

قضیهٔ Ladner (دههٔ ۷۰ میلادی). مسائلی در کلاس انپی وجود دارند که عضو هیچ‌یک از کلاس‌های انپی‌ـ تمام و پی نیستند، اگر و تنها اگر $\mathbf{P} \neq \mathbf{NP}$.

[1] **NP**-Complete
[2] Strong **NP**-Completeness
[3] **NP**-Hard

در واقع اگر $\mathbf{P} = \mathbf{NP}$ باشد که دو کلاس یکی هستند و مسائلی بین آن‌ها وجود ندارد اما اگر این چنین نباشد بی‌نهایت مسئله بین دو کلاس انپی‌ـ تمام و پی وجود دارد. همچنین می‌توان این مسائل را سطح‌بندی کرد طوری‌که مسائل هر سطح از سطح قبلی سخت‌تر هستند. اما نکته این است که به‌طور عملی ما معمولاً با مسائلی که بین این دو کلاس باشند مواجه نمی‌شویم. می‌دانیم مسائلی که خارج از کلاس انپی باشند وجود دارند و به تعدادی از آن‌ها قبلاً اشاره کردیم، مثل تعیین استراتژی برد در شطرنج یا بازی‌های مشابه در کلاسی قرار دارند که به نظر بزرگ‌تر از کلاس انپی است اما آن هم مانند مسئلهٔ $\mathbf{P}$ vs $\mathbf{NP}$ اثبات نشده است.

چند کاندیدا برای مسائل بین دو کلاس پی و انپی‌ـ تمام وجود دارد. مسئلهٔ اول، مسئلهٔ یکریختی گراف‌ها[۴] است. در واقع دو گراف به ما داده شده است و باید تعیین کنیم که این دو گراف یکریخت هستند یا نه. برای این مسئله نه الگوریتم چندجمله‌ای وجود دارد و نه ثابت شده است که انپی‌ـ تمام است اما به وضوح عضو کلاس انپی است. شواهدی وجود دارد که این مسئله انپی‌ـ تمام نیست زیرا ثابت شده است اگر انپی‌ـ تمام باشد یکی از حدس‌های مهم پیچیدگی نقض می‌شود و احتمال این‌که پی باشد بیش‌تر است اما ممکن است واقعاً بین این دو کلاس قرار داشته باشد. یک الگوریتم خوب برای این مسئله حدود پنج سال پیش داده شده است که چندجمله‌ای نیست اما Quasi-polynomial time است و این مسئله به کلاس پی نزدیک‌تر شده است.

مسئلهٔ دوم، مسئلهٔ Factoring است که بسیار مهم و کاربردی است و برای مثال از سخت بودن این مسئله در بعضی از پروتکل‌های رمزنگاری مانند پروتکل RSA استفاده می‌شود. برای این مسئله نیز نه الگوریتم چندجمله‌ای وجود دارد و نه ثابت شده است که انپی‌ـ تمام است اما به وضوح عضو کلاس انپی است. در کل به جز چند مسئلهٔ محدود اکثر مسائل یا پی هستند یا انپی‌ـ تمام و در واقع عضو کلاس انپی هستند. حال ممکن است این سوال به وجود بیاید که دلیل این اتفاق چیست. جهت یادآوری، به دسته مسائلی که اگر پاسخ مسئله مثبت باشد و شاهدی برای آن به ما بدهند، بتوانیم در زمان چندجمله‌ای درستی آن را بررسی کنیم، کلاس انپی می‌گوییم. این تعریف بسیار کلی است و خیلی از موقعیت‌های زندگی را در بر می‌گیرد. در واقع شهود آن به این شکل است که اگر حل یک مسئله را به ما بدهند بتوانیم به‌راحتی درستی آن را چک کنیم، شبیه ضرب‌المثل "معما چو حل گشت آسان شود!" و اکثر معماها به این شکل هستند. اگر به اهداف انسان‌های مختلف فکر کنید، مثلاً یک ریاضی‌دان به دنبال اثبات یک قضیه است که ممکن است سال‌ها برای رسیدن به اثبات تلاش کند اما زمانی‌که اثبات کامل شود می‌توان درستی آن را در زمان کوتاهی بررسی کرد. اگر اثبات در یک سیستم منطقی نوشته شده باشد حتی می‌توان به وسیلهٔ کامپیوتر درستی آن را چک کرد. یا می‌توانیم هدف کلی دانشمندان را به این‌صورت تعریف کنیم که یک سری مشاهدات از طبیعت دارند و می‌خواهند تئوری بدهند که این مشاهدات را توجیه کند. یا در مهندسی معمولاً یک سری محدودیت‌هایی داریم و می‌خواهیم طرحی بدهیم که این محدودیت‌ها را رعایت کند. در کل انسان‌ها به دنبال حل مسائلی هستند که اگر جواب آن پیدا شود به‌سادگی معلوم می‌شود که این همان جوابی است که به دنبال آن بودند. به این دلیل است که کلاس انپی بسیاری از مسائل را شامل می‌شود.

۲ انپی‌ـ تمامیت در علوم مختلف

حال به مفهوم انپی‌ـ تمامیت می‌پردازیم. اگر دقت کنید می‌بینید که این مفهوم نیز در بسیاری از حیطه‌های علمی وجود دارد و به این شکل نیست که فقط مسائلی که در علوم کامپیوتر مطرح شده‌اند را شامل شود. هر زمینه از علم را که در نظر بگیریم بسیاری از مسائل ثابت شده‌اند که انپی‌ـ تمام هستند. تعدادی از این مسائل به شرح زیر هستند:

Aerospace engineering. Optimal mesh partitioning for finite elements

Biology. Phylogeny reconstruction.

Chemical engineering. Heat exchanger network synthesis.

Chemistry. Protein folding.

Civil engineering. Equilibrium of urban traffic flow.

Economics. Computation of arbitrage in financial markets with friction.

[4] Graph isomorphism

Electrical engineering. VLSI layout.

Environment engineering. Optimal placement of contaminant sensors.

Financial engineering. Minimum risk portfolio of given return.

Game theory. Nash equilibrium that maximizes social welfare.

Mathematics. Given integers $a_1, a_2, \ldots, a_n$, compute $\int_0^{2\pi} \cos(a_1\theta) \times \cos(a_2\theta) \times \cdots \times \cos(a_n\theta)\ \mathrm{d}\theta$

Mechanical engineering. Structure of turbulence in sheared flows.

Medicine. Reconstructing 3d shape from biplane angiocardiogram.

Operations research. Traveling salesperson problem.

Physics. Partition function of 3d Ising model.

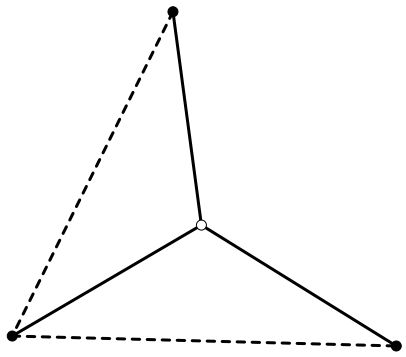
Politics. Shapley-Shubik voting power.

Recreation. Versions of Sudoku, Checkers, Minesweeper, Tetris, Rubik's Cube.

Statistics. Optimal experimental design.

خیلی از کسانی که انپی‌ـتمام بودن مسائل بالا را اثبات کرده‌اند، علوم‌کامپیوتردان نیز نبوده‌اند برای مثال زیست‌شناس یا شیمی‌دان یا اقتصاددان بوده‌اند و در حدی پیچیدگی محاسباتی یاد گرفته‌اند که انپی‌ـتمام بودن این مسائل را اثبات کنند. شاید بتوان گفت مفهوم انپی‌ـتمامیت از لحاظ بروزی که در بسیاری از حیطه‌های علمی دارد و استفاده‌ای که از آن می‌شود نزدیک به بی‌نظیر است. یک مفهوم بسیار جامع است که در همه جا وجود دارد. حال تعدادی سوال مطرح می‌شود. فرض کنید شما در حیطه‌ای از علم ثابت کرده‌اید یک مسئله انپی‌ـتمام است. برای مثال کمینهٔ انرژی که یک مولکول می‌تواند بگیرد چقدر است و مسائل مشابه. نکتهٔ جالبی که وجود دارد این است که به نظر می‌آید این مسائل در طبیعت خودبه‌خود حل می‌شوند. یک برداشت این است که واقعاً این امکان در طبیعت، یعنی حل مسائل انپی‌ـتمام، وجود دارد و ما می‌توانیم یک ماشین محاسباتی بسازیم که مسائل را به یک‌سری مولکول نظیر کند و آن‌ها را در طبیعت قرار دهیم تا مسئله را برای ما حل کند.

زمانی یک مسئلهٔ انپی‌ـسخت بسیار پررنگ شده بوده و این ادعا وجود داشته است که با استفاده از تعدادی حباب می‌توان آن را حل کرد. نام این مسئله درخت اشتاینر است که صورت‌های مختلفی دارد. یک صورت آن به این شکل است: تعدادی نقطهٔ ثابت در صفحه داریم، می‌توانیم تعدادی نقطهٔ کمکی به آن‌ها اضافه کنیم و درخت فراگیر کمینهٔ[۵] گراف حاصل را حساب کنیم. (در واقع فرض می‌کنیم بین هر دو نقطه یک یال وجود دارد که وزن آن، طول پاره‌خط واصل دو نقطه است.) بین همهٔ حالات ممکن، به دنبال درختی هستیم که کمینهٔ طول را دارد. برای مثال در شکل زیر اگر نقطهٔ سفید را اضافه کنیم طول درخت فراگیر کمینهٔ حاصل کم‌تر از حالت اولیه، که با خط‌چین مشخص شده است، می‌شود:



حال این مسئله را به‌طور فیزیکی می‌سازیم. دو شیشه برمی‌داریم و به‌جای نقاطی که مسئله به ما داده است روی یکی از شیشه‌ها تعدادی

[5] Minimum Spanning Tree

پایه قرار می‌دهیم و شیشهٔ دیگر را نیز روی پایه‌ها قرار می‌دهیم. سپس این وسیله را درون یک محلولی که از آب و صابون درست کرده‌ایم فرو می‌کنیم که بین پایه‌های قرار داده شده تعدادی حباب تشکیل شود. نکته این است که این حباب‌ها خودشان را به شکلی تنظیم می‌کنند که به حالت درخت کمینه در مسئلهٔ اشتاینر برسیم. زمانی این تصور وجود داشته است که این کار مسئلهٔ درخت اشتاینر را حل می‌کند اما واقعیت این است که تنها در صورتی که تعداد نقاط کم باشد، حباب‌ها مسئله را حل می‌کنند و زمانی که تعداد نقاط زیاد باشد صرفاً یک کمینهٔ موضعی را پیدا می‌کنند. پس نمی‌توان مسائل انپی‌ـ‌تمام را با حباب و صابون حل کرد! در حال حاضر این ایده که طبیعت می‌تواند مسائل انپی‌ـ‌تمام را حل کند چندان ایدهٔ پرطرفداری نیست و تصور بر این است که پروسه‌های طبیعی قرار نیست کار قوی‌تری نسبت به کامپیوترهای معمولی و با مدل‌های محاسباتی معمولی، انجام دهند.

سوال دیگر این است که اگر ما یک مسئله که در طبیعت اتفاق می‌افتد را به شکلی مدل کنیم، یک مسئلهٔ محاسباتی از آن به‌دست آوریم و ثابت شود که این مسئله انپی‌ـ‌تمام است، در حالی‌که به نظر طبیعت این مسئله را حل می‌کند، چه نتیجه‌ای می‌توانیم بگیریم؟ در جواب باید بگوییم که چند گزینه وجود دارد. گزینهٔ اول این است که این مسئله واقعاً در طبیعت به‌طور بهینه حل نمی‌شود و صرفاً به نظر می‌آید که طبیعت قادر به حل کردن آن است، گزینهٔ دوم این است که ما در مدل‌کردن آن اشتباه کرده‌ایم و گزینهٔ سوم این است که در طبیعت آن ورودی‌های سخت مسئله اتفاق نمی‌افتند و به همین علت طبیعت قادر به حل کردن آن است. این نیز نتیجهٔ بسیار مهمی است و اگر ورودی‌های مسئله در طبیعت را بررسی کنیم درک خیلی بهتری از مسئله به‌دست می‌آوریم. ما نیز می‌توانیم به ازای ورودی‌های خاص، بعضی از مسائل انپی‌ـ‌تمام را حل کنیم. مثلاً برای مسئلهٔ SAT، حل‌کننده‌هایی وجود دارند که در عمل بسیار خوب عمل می‌کنند اما به ازای یک‌سری ورودی‌ها ممکن است گیر کنند و نتوانند آن را حل کنند. اهمیت مسئلهٔ انپی‌ـ‌تمام بودن در طبیعت آن‌قدر زیاد است که بعضی از محققین می‌گویند بهتر است $\mathbf{P} \neq \mathbf{NP}$ را یک قانون طبیعت در نظر بگیریم مانند قانون دوم ترمودینامیک یا قوانین دیگری که در علوم دیگر وجود دارند. یعنی در طبیعتی که ما زندگی می‌کنیم نمی‌شود مسائل انپی‌ـ‌تمام را در زمان کارآمد[۶] حل کرد. یک علوم‌کامپیوتردان مشهور می‌گوید اگر علوم‌کامپیوتردانان هم مثل فیزیک‌دانان بودند رسماً اعلام می‌کردند که $\mathbf{P} \neq \mathbf{NP}$ و این یکی از قوانین طبیعت است که ما کشف کرده‌ایم! بعد از آن نیز به دلیل کشف این قانون مهم به خودشان جایزهٔ نوبل می‌دادند. اگر احیاناً کسی ثابت می‌کرد $\mathbf{P} = \mathbf{NP}$ به او نیز جایزهٔ نوبل می‌دادند! بنابراین دیدگاه پیچیدگی محور می‌تواند در علوم دیگر نیز بسیار کمک کند یعنی در یک مدل یا تئوری که از یک‌سری پروسه‌های طبیعی داده می‌شود، منابع مصرفی نیز لحاظ شود و اگر مدل به شکلی است که قابلیت اجرا کردن توسط طبیعت را به صورت کارآمد ندارد باید حدس بزنیم که احتمالاً مدل ما یک ایرادی دارد.

۳ NP، coNP و فراتر از آن!

ابتدا کلاس **coNP** را معرفی می‌کنیم. کلاس **coNP** دقیقاً مکمل کلاس انپی است یعنی برای هر مسئلهٔ A،

$$A \in \mathbf{NP} \iff \overline{A} \in \mathbf{coNP}.$$

مسئلهٔ $\overline{A}$ همان مسئلهٔ A است با این تفاوت که جای بله و خیر را در آن تغییر داده‌ایم. برای مثال دور همیلتونی نداشتن، ارضا پذیر نبودن یک مدار منطقی و tautology بودن یک عبارت منطقی (یک عبارت منطقی داده شده است و سوال این است که برای هر مقدار دهی دلخواه، حاصل آن True می‌شود یا نه)، همه مسائلی از کلاس **coNP** هستند. در جلسات قبل مسئلهٔ A از کلاس انپی را به شکل زیر تعریف کردیم:

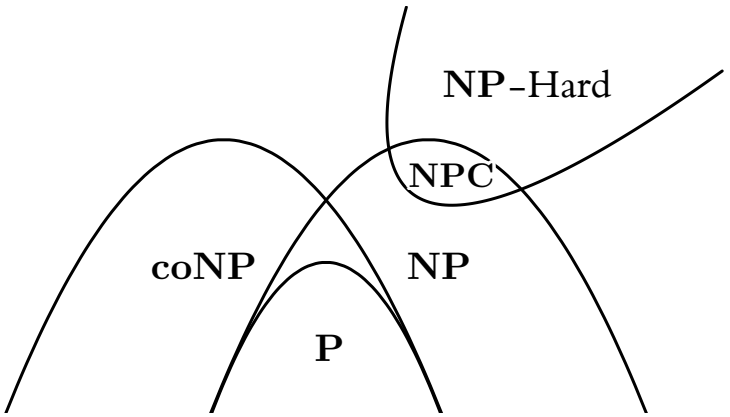
$$x \in A \iff \exists w,\ V(x, w) = 1$$

که V یک تصدیق‌کننده[۷] برای مسئلهٔ A است. حال مسئلهٔ B از کلاس **coNP** را می‌توانیم به شکل زیر تعریف کنیم:

$$x \in B \iff \forall w,\ V(x, w) = 1$$

[6] Efficient
[7] Verifier

که می‌توانید بررسی کنید چرا این رابطه درست است. حال می‌توانیم کلاس **coNP** را به نمودار قبل اضافه کنیم:

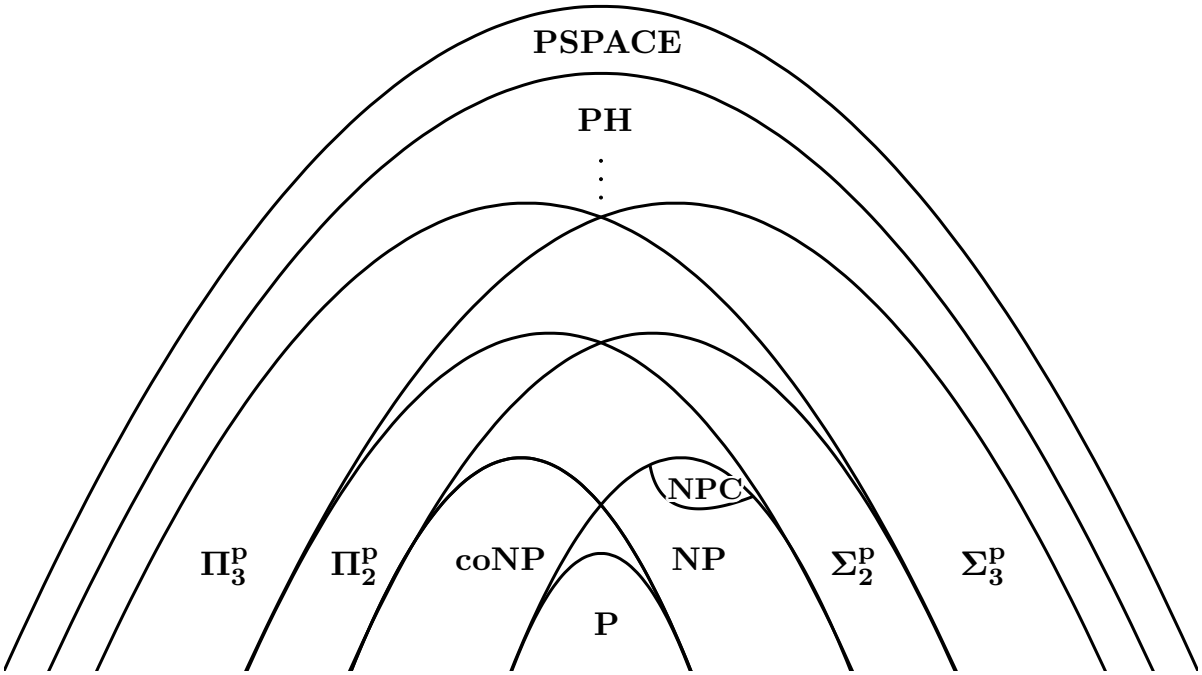


دقت کنید که پی زیرمجموعهٔ ان‌پی و **coNP** است. در واقع اگر مسئله‌ای عضو $\mathbf{NP} \cap \mathbf{coNP}$ باشد به این معناست که هم برای جواب مثبت و هم برای جواب منفی، شاهد چندجمله‌ای وجود دارد. برای مثال، مسئلهٔ تطابق کامل در گراف دو بخشی را در نظر بگیرید. ما از قبل می‌دانیم که این مسئله در پی است اما فرض کنید که نمی‌دانیم و می‌خواهیم نشان دهیم که عضو $\mathbf{NP} \cap \mathbf{coNP}$ است. اگر گراف تطابق کامل داشته باشد که یال‌های تطابق شاهدی بر این ادعا هستند پس این مسئله عضو ان‌پی است. حال فرض کنید گراف تطابق کامل نداشته باشد. در جلسات جریان بیشینه[۸] با قضیهٔ ازدواج هال[۹] آشنا شدیم. طبق این قضیه اگر گراف دو بخشی تطابق کامل نداشته باشد، مجموعهٔ S وجود دارد به طوری‌که $|N(S)| < |S|$ و همین مجموعه شاهدی بر این ادعا است که گراف تطابق کامل ندارد پس این مسئله عضو **coNP** نیز است. به‌طور مشابه می‌توان نشان داد تطابق کامل در گراف دلخواه نیز عضو $\mathbf{NP} \cap \mathbf{coNP}$ است (طبق قضیهٔ تات[۱۰]). در کل بسیاری از مسائل محاسباتی مربوط به روابط کمینه بیشینه‌ای که در ترکیبیات وجود دارند، عضو $\mathbf{NP} \cap \mathbf{coNP}$ هستند. مانند نسخهٔ تصمیم‌گیری مسئلهٔ جریان بیشینه و برش کمینه[۱۱] (نسخهٔ تصمیم‌گیری این مسائل به این شکل تعریف می‌شوند که یک عدد k داریم و می‌خواهیم ببینیم جریانی با اندازهٔ بزرگ‌تر یا مساوی k یا برشی با اندازهٔ کوچک‌تر یا مساوی k داریم یا نه). ان‌پی بودن این دو مسئله واضح است و از طرف دیگر، اگر بخواهیم نشان دهیم جریانی با اندازهٔ بزرگ‌تر یا مساوی k نداریم کافی است برشی با اندازهٔ کوچک‌تر از k ارائه دهیم. پس مسئلهٔ جریان بیشینه در **coNP** نیز است و به‌طور مشابه ثابت می‌شود برش کمینه نیز در این کلاس است. توجه کنید که برای اکثر مسائلی که در $\mathbf{NP} \cap \mathbf{coNP}$ قرار دارند، ثابت می‌شود که عضو پی نیز هستند. حال مسئلهٔ اول بودن[۱۲] را در نظر بگیرید. واضح است که این مسئله عضو **coNP** است زیرا برای شاهد می‌توانیم تجزیهٔ آن به عوامل اول را ارائه دهیم و به‌سادگی با ضرب کردن اعداد درستی آن را ثابت کنیم. این‌که این مسئله عضو ان‌پی است مقداری سخت‌تر است اما این نیز قابل اثبات است و در همان دههٔ هفتاد، که ابتدای شکل‌گیری پیچیدگی محاسباتی بوده است، اثبات شده است. پس از همان زمان می‌دانستند که این مسئله عضو $\mathbf{NP} \cap \mathbf{coNP}$ است و الگوریتم‌های تصادفی با زمان چندجمله‌ای داشته است اما الگوریتم قطعی چندجمله‌ای که نشان دهد این مسئله در پی است نداشته که آن هم حدود پانزده سال پیش پیدا شده است. حال این سوال پیش می‌آید که آیا $\mathbf{P} \stackrel{?}{=} \mathbf{NP} \cap \mathbf{coNP}$ و نظر اکثر پیچیدگی‌دانان این است که برابر نیستند. مانند نسخهٔ تصمیم‌گیری مسئلهٔ تجزیهٔ اعداد که حدس زده می‌شود الگوریتم چندجمله‌ای ندارد اما عضو $\mathbf{NP} \cap \mathbf{coNP}$ است. خوب است به این نکته اشاره کنیم که بعضی از محققین باور داشته‌اند که $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$ و یکی از مشهورترین آنان آقای جک ادموندز[۱۳] است که کارهای بسیار مهمی در بهینه‌سازی ترکیبیاتی و الگوریتم‌های گراف انجام داده است مانند الگوریتم ادموندز_کارپ[۱۴] که در جلسات تطابق بیشینه با آن آشنا شدیم، تطابق در گراف‌های غیر دو بخشی و مسئلهٔ intersection Matroid که در جلسات حل تمرین به آن اشاره شد. البته مثل این‌که اخیراً دیگر چنین باوری ندارد. نکتهٔ آخر دربارهٔ کلاس $\mathbf{NP} \cap \mathbf{coNP}$ این است که به مسائلی که

[8] Maximum Flow
[9] Hall's marriage theorem
[10] Tutte's theorem
[11] Minimum Cut
[12] Primality testing
[13] Jack Edmonds
[14] Edmonds-Karp algorithm

عضو این کلاس هستند Well characterized گفته می‌شود یا مسئله‌ای که Good characterization دارد یعنی هم برای جواب‌های مثبت مسئله و هم برای جواب‌های منفی مسئله یک توصیف خوب وجود دارد.

همان‌طور که در تعریف کلاس **coNP** دیدیم می‌توانیم با تغییر یا اضافه کردن سورهای جدید کلاس‌های پیچیدگی دیگری نیز بسازیم. برای مثال مسئلهٔ روبه‌رو را در نظر بگیرید: آیا گراف G زیرگرافی با حداکثر $\frac{n}{2}$ راس دارد که ۳ـ رنگ‌پذیر نباشد؟ اگر به صورت این مسئله دقت کنیم می‌بینیم که در آن یک *وجود دارد* ($\exists$)، داریم و یک *به‌ازای هر* ($\forall$). پس به نظر می‌رسد که این مسئله سخت‌تر از ان‌پی و **coNP** است زیرا نه می‌توانیم آن را با یک *وجود دارد* توصیف کنیم و نه با یک *به‌ازای هر* و به هر دو با هم نیاز داریم. برای این مسئله یک کلاس پیچیدگی جدید تعریف می‌کنند به اسم $\mathbf{\Sigma_2^p}$. این کلاس هر دو کلاس ان‌پی و **coNP** را شامل می‌شود، همچنین یک کلاس مکمل نیز دارد به اسم $\mathbf{\Pi_2^p}$، که در واقع در این کلاس ترتیب دو صور برعکس است. به همین ترتیب می‌توانیم کلاس‌ها را زیاد کنیم و کلاس‌های $\mathbf{\Sigma_3^p}, \mathbf{\Pi_3^p}, \ldots$ را بسازیم یعنی مسائلی که با دو صور نیز نمی‌توان آن‌ها را توصیف کرد و به ۳ صور یا بیش‌تر نیاز داریم. حال مجموعهٔ همهٔ این بی‌نهایت کلاس را **PH**[15] یا سلسله مراتب چندجمله‌ای می‌نامیم.



حدس قوی‌تر از $\mathbf{P \neq NP}$ این است که به‌ازای هر i، $\mathbf{\Sigma_i^p \neq \Sigma_{i+1}^p}$ یعنی در نمودار بالا بی‌نهایت کلاس متفاوت داریم. دقت کنید که هر $\mathbf{\Sigma_k^p}$ را می‌توان با k صور توصیف کرد که k یک عدد ثابت است. مسائلی وجود دارند که تعداد صورها یک عدد ثابت نیست و تعداد چندجمله‌ای‌تا صور دارند. برای مثال بازی شطرنج را در نظر بگیرید. اگر نفر سفید استراتژی برد داشته باشد یعنی حرکت w_1 برای نفر سفید وجود دارد که به‌ازای هر حرکت b_1 نفر مشکی حرکت w_2 برای نفر سفید وجود دارد که به‌ازای هر حرکت b_2 نفر مشکی و ... که نفر سفید ببرد! این سورها به همین شکل ادامه پیدا می‌کنند و تعداد ثابتی ندارند. قضیهٔ بسیار جالبی که ثابت شده است این است که کلاس این‌گونه مسائل همان کلاس **PSPACE** است یعنی کلاس مسائلی که در حافظهٔ چندجمله‌ای قابل حل هستند. یک طرف این حکم ساده‌تر است یعنی می‌توان نشان داد همهٔ حالات مختلفی که صورها دارند را می‌توان در حافظهٔ چندجمله‌ای ذخیره کرد و سپس همهٔ آن‌ها را بررسی کرد اما ثابت شده است که در واقع این دو کلاس معادل یک‌دیگر هستند. می‌توان مانند کلاس‌های پی و ان‌پی کلاس **NPSPACE** را تعریف کرد که مسائلی هستند که می‌توان آن‌ها را به‌طور غیر قطعی در حافظهٔ چندجمله‌ای حل کرد. باز هم این سوال پیش می‌آید که آیا $\mathbf{PSPACE \stackrel{?}{=} NPSPACE}$ و نکتهٔ جالب این است که این مسئله حل شده است و این دو کلاس با هم مساوی‌اند! یعنی برای حافظهٔ چندجمله‌ای این‌که ما بتوانیم غیر قطعی رفتار کنیم تفاوتی ایجاد نمی‌کند. در واقع این قضیه، حالت خاصی از قضیهٔ Savitch است

[15] **Polynomial Hierarchy**

که اثبات آن بسیار شبیه سوال ۴ میان‌ترم است. پس در کل روابط زیر را داریم:

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{\Sigma_2^p} \subseteq \mathbf{\Sigma_3^p} \subseteq \cdots \subseteq \mathbf{PH} \subseteq \mathbf{PSPACE}$$

و حدس این است که همهٔ این کلاس‌ها متفاوت‌اند یعنی روابط زیرمجموعه‌ها اکید است اما حتی ثابت نشده است که $\mathbf{P}$ مخالف $\mathbf{PSPACE}$ است! یعنی از لحاظ نظری حتی $\mathbf{P}$ نیز ممکن است با $\mathbf{PSPACE}$ برابر باشد. همچنین داریم $\mathbf{PSPACE} \subseteq \mathbf{EXP}$ که کلاس $\mathbf{EXP}$ مسائلی هستند که در زمان نمایی حل می‌شوند و به‌سادگی می‌توان دید که کلاس $\mathbf{EXP}$ شامل کلاس $\mathbf{PSPACE}$ است. یک نکتهٔ دلگرم کننده این است که ثابت شده است $\mathbf{P} \neq \mathbf{EXP}$. در نتیجه اگر رابطهٔ $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP}$ را در نظر بگیریم، ثابت نشده است که هیچ‌کدام از دو رابطهٔ شمول اکید هستند اما می‌دانیم که حداقل یکی از آن‌ها اکید است زیرا $\mathbf{P} \neq \mathbf{EXP}$ و حدس این است که هر دو رابطه اکید هستند.

۴ کلاس‌های پیچیدگی تصادفی

ما کلاس پی را در واقع کلاس مسائلی تعریف کردیم که در زمان کارآمد قابل حل هستند اما ممکن است این ایراد وارد شود که در عمل می‌توانیم از الگوریتم‌های تصادفی نیز استفاده کنیم و در کلاس پی این الگوریتم‌ها را در نظر نمی‌گیریم. پس خوب است کلاسی را معادل با مسائلی در نظر بگیریم که در زمان کارآمد قابل حل هستند، که در آن استفاده از الگوریتم‌های تصادفی با زمان چندجمله‌ای نیز مجاز است. با روش‌های مختلفی می‌توان این کلاس را تعریف کرد که یکی از مشهورترین و شاید مهم‌ترین آن‌ها کلاس $\mathbf{BPP}$ است. یک مسئلهٔ A را می‌گوییم عضو کلاس $\mathbf{BPP}$ است اگر الگوریتم تصادفی T با زمان چندجمله‌ای وجود داشته باشد که

$$x \in A \iff \mathrm{P}(T(x) = 1) \geq \frac{2}{3}$$
$$x \notin A \iff \mathrm{P}(T(x) = 0) \geq \frac{2}{3}$$

پس در واقع چه جواب مسئله مثبت باشد چه منفی الگوریتم T به احتمال $\frac{2}{3}$ جواب درست را می‌دهد. به‌طور مشابه می‌توان کلاس‌های پیچیدگی دیگری نیز تعریف کرد مانند $\mathbf{RP}$ و $\mathbf{coRP}$. تعریف کلاس $\mathbf{RP}$ این است که اگر جواب مسئله منفی بود حتماً باید الگوریتم خروجی منفی بدهد و اگر جواب مسئله مثبت بود به احتمال حداقل $\frac{1}{2}$ جواب درست را بدهد. کلاس $\mathbf{coRP}$ نیز دقیقاً برعکس این کلاس تعریف می‌شود. حال به بررسی کلاس $\mathbf{BPP}$ می‌پردازیم که به وضوح هر دو کلاس $\mathbf{RP}$ و $\mathbf{coRP}$ را شامل می‌شود. ابتدا دقت کنید که عدد $\frac{2}{3}$ اهمیتی ندارد و هر عددی بزرگ‌تر از $\frac{1}{2}$ را می‌توانیم در تعریف استفاده کنیم زیرا می‌توانیم الگوریتم را چند بار اجرا کنیم تا احتمال خطا از عدد مورد نظر کم‌تر شود و این که چند بار باید الگوریتم اجرا شود نیز به‌راحتی قابل محاسبه است. پس می‌توانیم احتمال خطا را هر چه‌قدر بخواهیم کم کنیم.

مسائلی وجود دارند که الگوریتم تصادفی خوب برای آن‌ها پیدا شده است اما الگوریتم غیر تصادفی خوب برای آن‌ها پیدا نشده است. به عنوان مثال مسئلهٔ اول بودن تا بیست سال پیش الگوریتم تصادفی کارآمد داشت اما الگوریتم غیر تصادفی کارآمد برای آن پیدا نشده بود. امروز نیز چنین مسائلی وجود دارند. برای مسائلی که الگوریتم قطعی دارند هم، گاهی الگوریتم‌های تصادفی ساده‌تر و کارآمدتر در عمل وجود دارند. اما با همهٔ این نکات حدس عجیبی که وجود دارد این است که $\mathbf{P} = \mathbf{BPP}$. در واقع شواهدی وجود دارد که اگر مساوی نباشند اتفاقات دیگری در دنیای پیچیدگی رخ می‌دهد که بسیار عجیب‌تر قلمداد می‌شوند به همین دلیل از حدود بیست و پنج سال پیش این حدس زده می‌شود که این دو کلاس با هم مساوی‌اند اما پیش از آن چنین حدسی زده نمی‌شده. بنابراین اگر این تساوی را بپذیریم، در مسائلی که برای ما فقط وجود الگوریتم چندجمله‌ای اهمیت دارد، تصادف چیزی به ما اضافه نمی‌کند. اما با این حال در رمزنگاری، تبادل اطلاعات و ساده‌سازی الگوریتم‌ها، تصادف به ما کمک بسیار زیادی می‌کند. نکتهٔ عجیب دیگری که در رابطه با کلاس $\mathbf{BPP}$ وجود دارد این است که اگر $\mathbf{P} = \mathbf{BPP}$ آن‌گاه به وضوح $\mathbf{BPP} \subseteq \mathbf{NP}$ اما هنوز این رابطه اثبات نشده است. در واقع بسیاری از روابط شمولی که در تئوری پیچیدگی وجود دارد صرفاً حدس زده می‌شوند. برای کران پایین‌ها هم اطلاعات کمی داریم و برای مثال حتی ثابت نشده است که

مسائل انپی‌ـ تمام الگوریتم خطی ندارند. یعنی از لحاظ نظری نه تنها امکان دارد که مسئلهٔ ۳ـ صدق‌پذیری[۱۶] الگوریتم چندجمله‌ای داشته باشد بلکه هنوز اثبات نشده است که برای حل آن به زمانی بیش‌تر از $O(n)$ نیاز داریم!

۵ $\mathbf{P} = \mathbf{NP}$

در این بخش قصد داریم بررسی کنیم که اگر $\mathbf{P} = \mathbf{NP}$ باشد چه اتفاقاتی می‌افتد. می‌دانیم که تعداد زیادی مسئلهٔ انپی‌ـ تمام وجود دارد و خیلی از آن‌ها نیز مسائل بسیار مهمی در حیطه‌های مختلف هستند. مانند مسائلی که در صنعت و علم با آن‌ها مواجه می‌شویم. بسیاری از مسائل بهینه‌سازی نیز انپی‌ـ تمام هستند. البته به یک فرض دیگر هم نیاز داریم. مثلاً ممکن است یک الگوریتم چندجمله‌ای برای مسئلهٔ ۳ـ صدق‌پذیری پیدا شود اما پیچیدگی زمانی آن $\Omega\left(n^{1000}\right)$ باشد. یکی از کسانی که فکر می‌کند ممکن است $\mathbf{P} = \mathbf{NP}$ باشد آقای کنوث[۱۷] است که یک علوم‌کامپیوتردان بسیار بزرگ است و تالیفات بسیار مهمی در زمینهٔ الگوریتم‌ها دارد. مانند کتاب The Art of Computer Programming که مرجع الگوریتم‌ها است و بیش از پنجاه سال است که آقای کنوث در حال نوشتن آن است. آقای کنوث بر این باور دارد که ما نمی‌دانیم الگوریتم‌های با مرتبهٔ n^{100} یا n^{1000} چه کارهایی می‌توانند انجام دهند و ممکن است برای مسائل انپی‌ـ تمام الگوریتم چندجمله‌ای با مرتبهٔ بسیار بالا وجود داشته باشد. در واقع الگوریتم‌های با مرتبهٔ بالا خیلی پیچیده‌اند و پیچیده‌تر از آن که انسان‌ها بتوانند آن‌ها را طراحی و تحلیل کنند. اکثر علوم‌کامپیوتردان‌ها اعتقاد دارند که $\mathbf{P} \neq \mathbf{NP}$، تعداد محدودی مانند آقای کنوث به تساوی اعتقاد دارند و عدهٔ دیگری فقط می‌گویند نمی‌دانیم.

حال فرض می‌کنیم که $\mathbf{P} = \mathbf{NP}$ و الگوریتم چندجمله‌ای که برای این مسائل وجود دارد در عمل نیز کارآمد هستند. قصد داریم نتایج آن را بررسی کنیم. به جز نتایجی که در ابتدای بخش به آن اشاره کردیم، یکی از نتایج آن **اثبات قضایای ریاضی** است زیرا قابل اثبات بودن یک مسئلهٔ ریاضی را می‌توان به یک مسئلهٔ انپی مدل کرد. به این شکل که فرض کنید یک مسئلهٔ ریاضی و یک عدد k به ما داده شده است. سوال این است که آیا برای این مسئله اثباتی (به‌صورت یکانی) وجود دارد که حداکثر k صفحه باشد یا خیر. اگر چنین اثباتی وجود داشته باشد و خیلی دقیق نوشته شود می‌توان درستی آن را به‌صورت مکانیکی بررسی کرد. پس اگر $\mathbf{P} = \mathbf{NP}$ باشد می‌توانیم اثبات‌پذیری قضایای ریاضی را به‌صورت الگوریتمی بررسی کنیم و ممکن است ریاضی‌دانان به یک معنایی بی‌کار شوند!

یک نتیجهٔ دیگر این است که در مسائلی که می‌توان شاهدی برای جواب آن ارائه داد (مسائل انپی)، می‌توان در زمان کارآمد خود جواب را نیز پیدا کرد. به زبانی دیگر، اگر بتوان جواب یک مسئله را خوب توصیف کرد می‌توان الگوریتمی نیز برای پیدا کردن آن در زمان کارآمد ارائه داد. برای مثال مسئلهٔ پیدا کردن دور همیلتونی[۱۸] را در نظر بگیرید. می‌دانیم $\mathbf{P} = \mathbf{NP}$ اما الگوریتمی برای پیدا کردن آن بلد نیستیم و فقط می‌توانیم یک تصدیق‌کننده برای آن بنویسیم. این تصدیق‌کننده علاوه بر گراف یک جایگشت از رئوس را نیز می‌گیرد و چک می‌کند که این جایگشت یک دور همیلتونی می‌سازد یا نه. نکتهٔ جالب این است که با همین تصدیق‌کننده می‌توانیم الگوریتمی را پیاده‌سازی کنیم که خود دور همیلتونی را نیز در زمان کارآمد پیدا کند. پس اگر بخواهیم مطالب گفته شده را خلاصه کنیم، **از روی توصیف یک مسئله به‌صورت یک مسئلهٔ انپی، می‌توان برای حلش الگوریتم تولید کرد.** زیرا همان‌طور که در جلسهٔ ۱۸ توضیح دادیم هر مسئلهٔ انپی را می‌توان به مسئلهٔ صدق‌پذیری مدار[۱۹] تبدیل کرد. در واقع هر مسئلهٔ انپی یک تصدیق‌کننده دارد که می‌توان به‌شکلی روند اجرای آن را به مدار تبدیل کرد. حال می‌توان یک برنامه نوشت که یک تصدیق‌کننده را بگیرد و به مدار یا ۳ـ صدق‌پذیری تبدیلش کند سپس با الگوریتم کارآمدی که برای این مسائل داریم، جواب مسئلهٔ اصلی را پیدا کند. یعنی هر مسئلهٔ انپی که داشته باشیم را با نوشتن یک تصدیق‌کننده می‌توانیم به‌طور اتوماتیک حل کنیم. مثلاً می‌توان این را در محیط برنامه‌نویسی پیاده‌سازی کرد. فقط کافی است ما تصدیق‌کنندهٔ مسئله‌ای که با آن مواجه شدیم را بنویسیم و سپس با کلیک کردن روی یک گزینه، به‌طور جادویی، برنامه جواب مسئله را به ما می‌دهد! دنیایی که در آن $\mathbf{P} = \mathbf{NP}$ است همین‌قدر شگفت‌انگیز است.

[16] 3-SAT
[17] Donald Knuth
[18] Hamiltonian cycle
[19] Circuit SAT

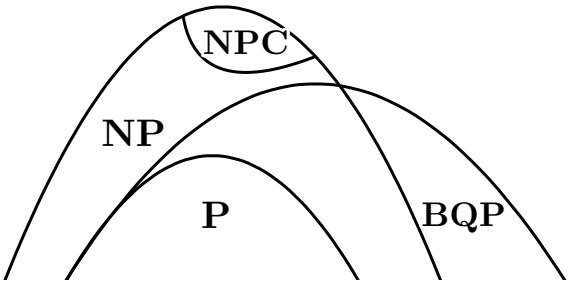
یکی از مسائل دیگری که می‌توان آن را به یک مسئلهٔ انپی مدل کرد **مسائل یادگیری**[۲۰] است. برای مثال یک برنامه یا مدلی تولید کنیم که روی یک مجموعه دادهٔ training بزرگ خوب جواب بدهد. یا فرض کنید مثلاً یک میلیون پاراگراف انگلیسی داریم که یک فرد آن‌ها را به فارسی ترجمه کرده است و با استفاده از آن‌ها می‌خواهیم یک برنامهٔ مترجم خوب بسازیم. مسئلهٔ انپی که می‌توانیم در این رابطه بسازیم این است که آیا برنامه‌ای با سایز حداکثر یک مگابایت وجود دارد طوری‌که برای همهٔ پاراگراف‌هایی که داریم عیناً ترجمهٔ یکسانی ارائه دهد. در این مسئله کوچک بودن سایز برنامه بسیار اهمیت دارد زیرا برنامه‌ای با سایز دلخواه که همهٔ آن یک میلیون عبارت را درست ترجمه کند، به وضوح وجود دارد. صرفاً کافی است ترجمهٔ همهٔ عبارات را در برنامه ذخیره کنیم و برنامه هر ورودی که گرفت را بین عبارات ذخیره شده جستجو کند، اگر موجود بود همان را خروجی دهد و در غیر این‌صورت یک عبارت دلخواه را خروجی بدهد. اما این برنامه برای ما کارآیی ندارد و ما دنبال برنامه‌ای هستیم که حجم آن به‌طور قابل ملاحظه‌ای از حجم عبارت‌هایی که داریم کم‌تر باشد. زیرا در این‌صورت معلوم می‌شود که این برنامه واقعاً دارد عبارات را ترجمه می‌کند و به این شکل نیست که همهٔ عبارات را فقط حفظ کرده باشد. پس اگر بتوانیم وجود داشتن این برنامه را به‌طور کارآمد چک کنیم می‌توانیم روی حجم آن جستجوی دودویی[۲۱] بزنیم و چنین برنامه‌ای را پیدا کنیم. نکته‌ای که لازم است به آن اشاره کنیم این است که ما بیش‌تر راجع‌به مسائل تصمیم‌گیری بحث کردیم اما اگر $\mathbf{P} = \mathbf{NP}$ باشد در خیلی از مسائل ما نیاز داریم که اگر پاسخ مسئله مثبت باشد خود جواب را نیز پیدا کنیم. در واقع این کار امکان‌پذیر است و در جلسات پیش توضیح مختصری راجع‌به آن داده شده. برای دیدن یک مثال نیز می‌توانید مسئلهٔ ۱ از تمرین سری ۱۰ را ببینید.

یکی دیگر از مسائلی که پیدا کردن آن سخت اما تصدیق کردن آن ساده است، **تولید آثار هنری** است. مثلاً زمانی که یک نقاشی زیبا می‌بینیم یا یک موسیقی زیبا می‌شنویم، زیبایی آن را به‌راحتی می‌توانیم تشخیص دهیم. یک جملهٔ مشهور نیز وجود دارد که اگر $\mathbf{P} = \mathbf{NP}$ آن‌گاه همهٔ افراد موتزارت[۲۲] می‌شوند! دقت کنید که زیبایی اثر هنری یک چیز شهودی است اما در صورتی که بتوان آن را با یک الگوریتم تشخیص داد (الگوریتم تصدیق‌کننده)، واقعاً می‌توانیم ادعا کنیم که با یک مسئلهٔ انپی مواجه هستیم. حال همان‌طور که در مسائل یادگیری دیدیم، می‌توان این پروسهٔ تشخیص را نیز مانند آن‌ها انجام داد. مثلاً تعدادی نقاشی را انتخاب کنیم و نظر چند متخصص را دربارهٔ آن‌ها بپرسیم. سپس برنامه‌ای تولید کنیم که نظرش تا جای ممکن بر نظر متخصصین منطبق باشد و زیبایی آثار جدید را با استفاده از این برنامه تشخیص دهیم. البته نکات گفته شده صرفاً گمانه‌زنی است راجع‌به اتفاقاتی که خواهد افتاد و چندان دقیق نیست. اگر علاقه‌مند هستید که نکات بیش‌تری را در این رابطه ببینید می‌توانید به کتاب The Golden Ticket از آقای Lance Fortnow مراجعه کنید که پیش‌نیاز خاصی نیز ندارد و هر کسی می‌تواند آن را مطالعه کند. در واقع این کتاب سعی کرده است مسائل پی، انپی و مسائل مرتبط را به زبانی ساده و جذاب ارائه کند. یک سناریوی جالبی که در این کتاب گفته شده، این است که فرض کنید ثابت شده است $\mathbf{P} = \mathbf{NP}$ اما الگوریتمی که برای مسئلهٔ صدق‌پذیری پیدا شده است خیلی کارآمد نیست، مثلاً $O\left(n^8\right)$ است و ثابت خوبی نیز ندارد. سپس بعد از چند سال و با تلاش‌های بسیار توانسته‌اند الگوریتم را به $O\left(n^6 \log n\right)$ برسانند. اما همچنان این الگوریتم کارآمد نیست. حال ما بهترین سوپر کامپیوتر دنیا را داریم که مثلاً دو هفته طول می‌کشد تا مسئلهٔ صدق‌پذیری را برای $n = 10^6$ حل کند. سوالی که پیش می‌آید این است که از این سوپر کامپیوتر برای حل چه مسئله‌ای استفاده کنیم؟ ممکن است با توجه به شرایط حال حاضر بگویید از آن استفاده می‌کنیم تا واکسن کرونا را پیدا کنیم یا اثبات قضایای ریاضی را پیدا کنیم. در کل کاندیداهای مختلفی وجود دارد اما سوال ما این است که بهترین کار چیست؟ البته ممکن است افراد مختلف نظرات مختلف داشته باشند و این سوال چندان خوش‌تعریف نیست! یک ایدهٔ بسیار خوب این است که وقت این کامپیوتر را صرف این کنیم که الگوریتم ما خودش را بهتر کند! این مسئله را در نظر بگیرید که ما به دنبال الگوریتمی هستیم که به‌ازای ورودی‌هایی که از یک عدد مشخص کوچک‌تر هستند پاسخ یکسانی با الگوریتم قبلی می‌دهد اما زمان اجرای الگوریتم جدید بسیار بهتر است. می‌توان دید که این مسئله یک مسئلهٔ انپی است که با روش‌های مختلفی نیز می‌توان آن را تعریف کرد. پس در واقع بهترین مسئله‌ای که می‌توانیم حل کنیم خود آن مسئله است. وقتی این کار را انجام دهیم ممکن است ابرکامپیوتر ما بعد از چند ماه یک الگوریتم جدید پیدا کند با زمان $O\left(n^5\right)$ و بلافاصله می‌تواند الگوریتم جدید را با الگوریتم قبلی تعویض کند و این بار با الگوریتم جدید به دنبال الگوریتم بهتر بگردد تا باز به الگوریتم‌های با زمان اجرای بهتر برسد. ممکن است برای بار دوم بعد از فقط چند روز به الگوریتمی با زمان اجرای $O\left(n^2 \log n\right)$ برسد. پس این یک راه‌کار بسیار جالب، برای سوال مطرح‌شده است. یک نگاه جالب به این سوال که در کتاب آقای

[20] Learning
[21] Binary search
[22] Wolfgang Amadeus Mozart

Lance Fortnow آمده است، این است که اگر یک غول چراغ جادو داشته باشیم که فقط یک آرزوی ما را برآورده می‌کند، بهترین آرزو چیست؟ جواب این است که یک غول چراغ جادویی به ما بدهد که همهٔ آرزوهای ما را برآورده کند!

تا این‌جا فقط راجع‌به مزایای برابر شدن پی و انپی بحث کردیم اما واقعیت این است که معایبی هم دارد. به عنوان مثال بخش‌هایی از پروتکل‌های رمزنگاری بر پایهٔ سخت‌بودن مسائل بنا شده‌اند و اگر سختی مسائل شکسته شود بسیاری از پروتکل‌های رمزنگاری حال حاضر نیز شکسته می‌شوند و مشکلات بسیار زیادی در این حیطه پیش می‌آید. به‌طور خاص یک‌سری از پروتکل‌های رمزنگاری بر مبنای سختی مسئلهٔ تجزیهٔ اعداد بنا شده‌اند. اگر ثابت شود این مسئله عضو پی است، که واقعاً احتمال دارد که عضو پی باشد، کل امنیت تجارت الکترونیکی یک شبه به هم می‌ریزد. در واقع بسیاری از سیستم‌های اقتصادی و امنیتی جهان بر مبنای سختی مسائل بنا شده‌اند که اهمیت بسیار زیاد پیچیدگی محاسباتی را نشان می‌دهد. دقت کنید که برخلاف مسائل انپی‌ـ‌تمام که تصور اکثریت علوم‌کامپیوتردان‌ها این است که الگوریتم چندجمله‌ای ندارند، در رابطه با مسئلهٔ تجزیهٔ اعداد چنین تصوری وجود ندارد. همچنین این مسئله توسط کامپیوترهای کوانتومی قابل حل است یعنی الگوریتم کوانتومیِ چندجمله‌ای برای این مسئله وجود دارد. این تصور غلط نیز وجود دارد که مسائل انپی‌ـ‌تمام الگوریتم کوانتومیِ چندجمله‌ای دارند در حالی‌که تا امروز چنین چیزی ثابت نشده است. اما یک‌سری از مسائلی که تصور می‌شود الگوریتم چندجمله‌ای ندارند، الگوریتم کوانتومیِ چندجمله‌ای دارند. به کلاس مسائلی که الگوریتم کوانتومیِ چندجمله‌ای دارند **BQP** گفته می‌شود و حدس زده می‌شود که رابطهٔ آن با کلاس انپی به شکل زیر باشد:



یعنی کلاس پی، بعضی از مسائل بین پی و انپی‌ـ‌تمام، و بعضی از مسائل خارج از انپی را شامل می‌شود اما کلاس **BQP** هنوز چندان شناخته‌شده نیست.

یک رمان جالب نیز در این زمینه اخیراً نوشته شده است به نام Factor Man. این رمان داستان آدمی را روایت می‌کند که الگوریتمی سریع برای مسئلهٔ صدق‌پذیری پیدا کرده است. در نتیجه می‌تواند اعداد را خیلی سریع تجزیه کند و سیستم‌های رمزنگاری را بشکند. در واقع یک وبلاگی را ایجاد می‌کند و هر روز یک عدد بزرگ‌تر از عدد روز قبل را تجزیه می‌کند. در ابتدا چیز عجیبی نیست اما کم‌کم که اعداد بزرگ می‌شوند بسیار عجیب می‌شود تا زمانی که همه باور می‌کنند به الگوریتم سریع تجزیهٔ اعداد دست یافته است. سپس سازمان‌های اطلاعاتی آمریکایی و چینی به دنبالش می‌افتند اما Factor Man بعد از مدتی الگوریتمش را به مزایده می‌گذرد و یکی از غول‌های فناوری یعنی اپل، مایکروسافت، گوگل و ... مزایده را برنده می‌شود. در ادامه نیز یک روز اعلام می‌کند که می‌خواهد خودش را نشان بدهد و تا آن ساعت‌های آخر به دنبالش هستند که دستگیرش کنند. خلاصه تعلیق خوبی را ایجاد کرده و داستان موفقی بوده است.

۶ تعامل و تصادف در اثبات‌ها

در بخش آخر راجع‌به اثبات صحبت خواهیم کرد. همان‌طور که قبلاً دیدیم تعریف مسائل انپی این بود که اگر پاسخ مسئله‌ای مثبت باشد بتوان اثباتی برای آن ارائه کرد که در زمان چندجمله‌ای صحت آن قابل بررسی باشد. در مسائل **coNP** نیز زمانی که پاسخ مسئله منفی باشد می‌توان برای آن اثباتی ارائه داد. حال می‌خواهیم ببینیم که آیا می‌توان مفهوم اثبات را مقداری منعطف‌تر کرد تا مثلاً برای حالتی که پاسخ یک مسئلهٔ **coNP** مثبت است نیز بتوان، به معنایی، یک اثبات ارائه داد. زمانی‌که یک فرد قصد دارد حکمی را برای ما اثبات کند، یکی از اِلمان‌هایی که می‌تواند در اثباتش وجود داشته باشد تعامل است. یعنی زمانی‌که آن فرد در حال اثبات گزاره برای ما است می‌توانیم

از او سوال بپرسیم. همین تعامل ممکن است باعث شود که یک حکم به‌طور کارآمدتری برای ما اثبات شود. مفهوم دیگری که وجود دارد نیز تصادف است. یعنی فردی که در حال اثبات حکم است ممکن است از تصادف استفاده کند و به همین دلیل احتمال دارد که بتواند اثبات کارآمدتری ارائه دهد. دقت کنید که در این قسمت هدف ما کارآمد بودن اثبات است. زیرا در بسیاری از مسائل اثبات وجود دارد اما کارآمد نیستند. مثلاً در مسئلهٔ پیدا کردن دور همیلتونی می‌توانیم با امتحان کردن همهٔ جایگشت‌های ممکن از رئوس ثابت کنیم که گراف دور همیلتونی ندارد اما به وضوح این اثبات برای ما کارآمد نیست زیرا تعداد حالات آن نمایی است. نکته این است که اگر ما تعامل و تصادف را به اثبات‌ها اضافه کنیم، آن‌گاه مسائل خیلی بیش‌تری را می‌توان به‌طور کارآمد اثبات کرد.

مسئلهٔ یکریخت نبودن گراف‌ها را در نظر بگیرید. اگر دو گراف یکریخت باشند که می‌توان جایگشتی که ماتریس مجاورت گراف اول را به ماتریس مجاورت گراف دوم تبدیل می‌کند، را به عنوان اثبات ارائه داد. پس این مسئله در **coNP** است اما معلوم نیست که عضو انپی نیز باشد. گاهی دو گراف تفاوت‌های واضحی دارند مثلاً تعداد یال‌های متفاوتی دارند پس به وضوح یکریخت نیستند اما در حالت کلی معلوم نیست که با چه روشی می‌توان یکریخت نبودن دو گراف را به‌طور کارآمد اثبات کرد. در عین حال یک اثبات زیبا با استفاده از تعامل و تصادف برای آن وجود دارد. فرض کنید یک نفر که قدرت محاسباتی نامحدودی دارد، می‌داند که دو گراف یکریخت نیستند (در واقع همه چیز را می‌داند!) و می‌خواهد به ما که قدرت محاسباتی چندجمله‌ای داریم این را ثابت کند. توجه کنید که ما می‌توانیم با این فرد تعامل داشته باشیم. در ابتدا یکی از دو گراف G_1 و G_2 را به‌طور تصادفی انتخاب می‌کنیم. مثلاً فرض کنید G_1 انتخاب شده است. سپس یک جایگشت تصادفی نیز انتخاب می‌کنیم و آن جایگشت را روی ماتریس مجاورت گراف G_1 اعمال می‌کنیم. گراف حاصل را به آن فرد همه‌چیزدان می‌دهیم و از او می‌پرسیم که این گراف G_1 است یا G_2؟ اگر G_1 و G_2 واقعاً یکریخت نباشند، باید بتواند درست تشخیص دهد زیرا فرض کرده‌ایم که قدرت محاسباتی نامحدودی دارد اما اگر G_1 و G_2 یکریخت باشند، با همهٔ آن قدرت محاسباتی که دارد نیز نمی‌تواند تشخیص دهد که ما کدام گراف را انتخاب کرده بودیم زیرا هر سه گراف یکریخت هستند و بهترین کاری که می‌تواند انجام دهد این است که یک جواب تصادفی بدهد که به احتمال $\frac{1}{2}$ جواب غلط می‌دهد. این کار را ۱۰۰ بار تکرار می‌کنیم و اگر هر بار جواب درستی داد آن‌گاه تا حد خیلی خیلی زیادی می‌توانیم مطمئن شویم که دو گراف یکریخت نیستند زیرا اگر یکریخت باشند به احتمال 2^{-100} همهٔ این ۱۰۰ بار را می‌تواند درست جواب دهد که بسیار بعید است. پس دیدیم که با تعامل و تصادف می‌توان اثباتی را با احتمال بسیار بالا (با احتمال خطای خیلی کم) فهمید یعنی اثبات تعاملی ارائه کرد برای مسئله‌ای که در انپی نیست و پاسخ آن مثبت است. در واقع با استفاده از این اثبات‌های تعاملی می‌توان یک کلاس پیچیدگی جدید تعریف کرد به نام **IP**[۲۳] که علاوه بر انپی، **coNP** را نیز شامل می‌شود. یک قضیهٔ بسیار مهمی که در اوایل دههٔ نود میلادی ثابت شده است، این است که $\mathbf{IP} = \mathbf{PSPACE}$. همان‌طور که قبلاً دیدیم کلاس **PSPACE**، کلاس بسیار بزرگی است و علاوه بر کلاس‌های انپی و **coNP** کل کلاس **PH** را نیز شامل می‌شود که این نتیجهٔ خیلی جالبی است زیرا نشان می‌دهد اگر ما تعامل و تصادف را به اثبات‌ها اضافه کنیم دسته مسائل خیلی گسترده‌تری را می‌توانیم به‌طور کارآمد اثبات کنیم.

یک نوع اثبات دیگر که ممکن است بسیار مفید باشد و در درس رمزنگاری نیز کاربرد فراوانی دارد، اثبات‌های ناتراوا[۲۴] هستند. این خیلی خوب است که ما بتوانیم گزاره‌ای را برای یک نفر اثبات کنیم و او را قانع کنیم که این گزاره درست است بدون این که چیز بیش‌تری را لو دهیم. مثلاً فرض کنید یک حدس خیلی مهم ریاضی را اثبات کرده‌اید و می‌خواهید آن را به یک متخصص نشان بدهید. اما نگرانید که وقتی اثبات را به آن متخصص نشان می‌دهید، چون او از شما مشهورتر است اثبات را به اسم خودش چاپ کند. حال اگر شما بتوانید او را قانع کنید که شما قضیه را اثبات کرده‌اید اما خود اثبات را به او لو ندهید بسیار جالب است. در نگاه اول انجام چنین کاری بسیار عجیب و غیر شهودی به نظر می‌رسد زیرا قرار است طرف مقابل قانع شود که گزاره درست است اما خودش نتواند یک نفر دیگر را قانع کند که گزاره درست است. مثال یکریخت نبودن گراف‌ها، که قبلاً آن را بررسی کردیم، می‌تواند شهود خوبی به ما در این زمینه بدهد، هر چند نیاز داریم یک‌سری نکات را برطرف کنیم تا بتوانیم به آن اثبات ناتراوا بگوییم. در همان مثال اگر مثلاً ۱۰۰۰ بار از طرف مقابل سوال بپرسیم واقعاً قانع می‌شویم که دو گراف یکریخت نیستند، در عین حال هیچ چیزی از اثبات آن نیز نمی‌فهمیم و ما نمی‌توانیم به یک نفر دیگر ثابت کنیم که دو گراف یکریخت نیستند. روی این نوع اثبات‌ها نیز کار شده و ثابت شده که با فرض‌هایی این نوع اثبات‌ها وجود دارند. یک

[23] **Interactive Proof**

[24] Zero-knowledge proofs

فرض که در رمزنگاری معمول است، وجود توابع یک‌طرفه است. اگر این فرض را بپذیریم آنگاه هر مسئله که عضو انپی است اثبات ناتراوا نیز دارد. یعنی اگر جواب مسئله‌ای مثبت باشد، یک اثبات‌کننده[۲۵] که قدرت محاسباتی نامحدودی دارد می‌تواند یک تصدیق‌کننده که قدرت محاسباتی چندجمله‌ای دارد را قانع کند که جواب مسئله مثبت است بدون این‌که تصدیق‌کننده جواب را یاد بگیرد. توجه کنید که ما خیلی از نکات فنی اثبات‌های ناتراوا را نگفتیم و بحث را در سطح شهودی نگه داشتیم اما نکتهٔ جالب این است که چنین اثبات‌هایی وجود دارند و می‌توانند کاربرد زیادی داشته باشند. به عنوان مثال در خیلی از پروتکل‌های رمزنگاری ممکن است تعدادی شرکت کننده باشند و بخواهند با یکدیگر کاری را انجام دهند و هر کسی بخواهد اطلاعات خصوصی خودش را لو ندهد اما در عین حال به بقیه ثابت کند که آن تکه‌ای از پروتکل که بر عهده‌اش بوده را انجام داده است. به وضوح این‌جا اثبات ناتراوا کاربرد دارد.

آخرین اثباتی که قرار است راجع‌به آن صحبت کنیم، Probabilistically Checkable Proofs (PCP) هستند. فرض کنید می‌خواهید برگه‌های امتحان میان‌ترم دانشجوها را تصحیح کنید. بعضی از دانشجوها راه حل‌های خیلی طولانی نوشته‌اند و خواندن کل راه حل کار سختی است. اگر می‌توانستید از کل اثبات تنها یک خط آن را بخوانید و مطمئن شوید که اثبات درست است یا نه، بسیار جالب بود و در زمان نیز بسیار صرفه‌جویی می‌شد. اما به نظر می‌رسد که کار خیلی عجیبی است و بعید است که بتوان چنین کاری انجام داد. در واقع اگر شما یک بخشی از اثبات را به عنوان نمونه در نظر بگیرید، همواره این احتمال وجود دارد که ایراد اثبات در قسمتی باشد که شما آن را در نظر نگرفته‌اید حتی اگر قسمت کوچکی را چک نکرده باشید، چه برسد به این‌که فقط یک خط اثبات را در نظر بگیرید. اما راهی وجود دارد که اثبات‌ها به شکلی نوشته شوند که واقعاً نیاز نباشد همهٔ اثبات را بررسی کنید. یعنی یک نحوهٔ نوشتن اثبات وجود دارد، برای اثبات‌هایی که برای همهٔ مسائل انپی داده می‌شود (همان شاهدی که برای نشان دادن مثبت بودن جواب مسئله ارائه می‌شود)، که آن الگوریتم چندجمله‌ای که قرار است اثبات را تصدیق کند لازم نباشد همهٔ اثبات را بخواند و با خواندن تکهٔ کوچکی از اثبات، که به‌صورت احتمالی انتخاب می‌شود، می‌تواند با احتمال خیلی خیلی بالا مطمئن شود که اثبات درست یا غلط است. در واقع فقط کافی است ۳۰ بیت (حدود ۴ بایت) از آن اثبات را بخواند و با احتمال 0.999 بفهمد که آن اثبات درست یا غلط است. به این قضیه، قضیهٔ PCP گفته می‌شود که یکی از عمیق‌ترین قضایایی است که در علوم کامپیوتر ثابت شده است و تبعات زیادی داشته است. این قضیه نیز در اوایل دههٔ ۹۰ میلادی ثابت شده است. به عنوان مثال در مسئلهٔ پیدا کردن دور همیلتونی می‌توانیم تصدیق‌کننده‌ای بنویسیم که یک گراف را بگیرد و اگر دور همیلتونی دارد بتوانیم اثباتی به آن تصدیق‌کننده بدهیم که تنها با نگاه کردن ۳۰ بیت از آن با احتمال 0.999 بتواند تشخیص دهد که اثبات داده شده درست بوده است یا غلط. نکتهٔ جالب دیگر این است که اگر گراف دور همیلتونی داشته باشد، تصدیق‌کننده همواره جواب مثبت می‌دهد و در این حالت اصلاً خطا ندارد. در واقع احتمال خطای 0.001 برای مواقعی است که تصدیق‌کننده باید جواب منفی بدهد که با تکرار می‌توانیم هر چه‌قدر که بخواهیم این خطا را کم کنیم. قضیهٔ PCP کاربردهای دیگری نیز دارد. به‌طور خاص برای ثابت کردن تقریب‌ناپذیری مسائل. یکی از مسائلی که در تئوری انپی‌ـ‌تمامیت وجود دارد این است که اگر $\mathbf{P} \neq \mathbf{NP}$ آنگاه یک‌سری از مسائل راه حل دقیق ندارند. بعضی از مسائل انپی‌ـ‌تمام را می‌توان با یک ضریب تقریب حل کرد یعنی الگوریتم چندجمله‌ای وجود دارد که جوابی تولید می‌کند که مثلاً از دو برابر جواب بهینه بدتر نیست. اما نکته این است که این ضریب تقریب‌ها یک حدی دارند. توجه کنید که از نظر حل‌پذیری در زمان چندجمله‌ای مسائل انپی‌ـ‌تمام با یکدیگر معادل هستند اما از نظر تقریب‌پذیری با یکدیگر معادل نیستند. بعضی از آن‌ها را می‌توان خوب تقریب زد و بعضی از آن‌ها را نمی‌شود. قضیهٔ PCP یک ابزار خیلی خوب است برای اثبات این‌که بعضی از مسائل را نمی‌شود تقریب زد. برای مثال اگر $\mathbf{P} \neq \mathbf{NP}$ آنگاه ثابت می‌شود هیچ الگوریتمی وجود ندارد که بتواند جواب مسئلهٔ رنگ‌آمیزی گراف‌ها را با یک ضریب ثابت یا حتی ضریب تقریب لگاریتمی تقریب بزند. به‌طور دقیق‌تر ضریب تقریب $n^{1-\epsilon}$ ندارد و این‌ها نتایجی هستند که تا پیش از این‌که قضیهٔ PCP ثابت شود نمی‌توانستند به آن‌ها دست پیدا کنند.

[25] Prover

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۲۱: جست‌وجوی هوشمندانه

نگارنده: امین کشیری و فاطمه توحیدیان

در این جلسه راجع به روش‌های حل مسائل سخت صحبت خواهیم کرد. در جلسات قبل و به طور خاص در ۱۶ جلسه‌ی ابتدایی درس الگوریتم‌هایی کارا و با زمان اجرای چندجمله‌ای برای حل مسائل کلاسیک ارائه دادیم. همچنین بررسی کردیم که چگونه می‌توان مسائلی را که در نگاه اول راه‌حل مشخصی ندارند، به صورت کارا به مسائل دیگر کاهش داد [۱] . برای مثال در مبحث جریان در شبکه‌ها [۲] تعدادی مسئله را به کمک مسائل جریان بیشینه [۳] و یا برش کمینه [۴] و در مبحث کوتاه‌ترین مسیر [۵] به کمک مسئله ی کوتاه‌ترین مسیر مدل کردیم. در این جلسه به طور خاص سعی می‌کنیم درمورد حل مسائلی که در چند جلسه‌ی قبلی ثابت کردیم ان‌پی‌ـ سخت [۶] هستند صحبت کنیم.

۱ مواجهه با مسائل ان‌پی‌ـ سخت

هنگام حل مساله اولین روشی که به ذهن میرسد این است که مسئله را به مسائلی که راه‌حل آنها را میدانیم تحویل کنیم یا اینکه از تکنیک‌هایی که برای حل مسئله یاد گرفتیم استفاده کنیم (مانند برنامه‌ریزی پویا [۷]). اما بعضی اوقات روش‌های ذکر شده منجر به حل مسئله نمی شوند. در این حالت می‌توانیم سعی کنیم نشان دهیم مسئله ان پی‌ـ سخت است. اما نکته‌ی قابل توجه این است که با ثابت کردن ان پی‌ـ سخت بودن مسئله ماجرا تمام نمی‌شود زیرا این مسئله در جایی کاربردی داشته و صرف ان‌پی‌ـ سخت بودن آن از لزوم حل شدن آن کم نمی‌کند. در این صورت باید بررسی کنیم که بهترین کاری که می‌توان انجام داد چیست. خواسته‌ی اولیه‌ی ما این است که مسئله را را در کلی‌ترین حالت، به طور دقیق و با زمان اجرای خوب حل کنیم. در نتیجه اگر مسئله‌ی ما ان‌پی‌ـ سخت باشد لازم است که یکی از خواسته‌های خود را فدا کنیم. حال روش‌های حمله به مسائل ان‌پی‌ـ سخت را بررسی میکنیم.

۱. حالت خاص مسئله را حل کنیم.

برای مثال در مسئله‌ی پوشش راسی [۸] اگر گراف مورد نظر درخت [۹] باشد می‌توان مسئله را برای این حالت خاص به راحتی حل کرد. در حالت کلی اگر مسئله‌ی مورد نظر از کاربردی واقعی ایجاد شده باشد می‌توانیم با دقت‌کردن به ورودی‌های مسئله ببینیم که گراف‌هایی که در عمل با آن‌ها سر و کار داریم معمولاً گراف‌های خاصی هستند. در حقیقت مدل‌سازی اولیه ی ما برای حل مسئله دقیق نبوده است و می‌توانیم با دقیق‌تر کردن ورودی به حل مسئله کمک کنیم.

بسیاری از مطالعات الگوریتمی انجام شده راجع به حل مسائل روی کلاس‌های خاصی از گراف ها است. برای مثال مسائل ان‌پی‌ـ سخت زیر برای کلاس‌های زیر حل شده اند (در زمان چندجمله‌ای):

[1] Reduction
[2] Network Flow
[3] Max Flow
[4] Min Cut
[5] Shortest Path
[6] NP-Hard
[7] Dynamic Programming
[8] Vertex Cover
[9] Tree

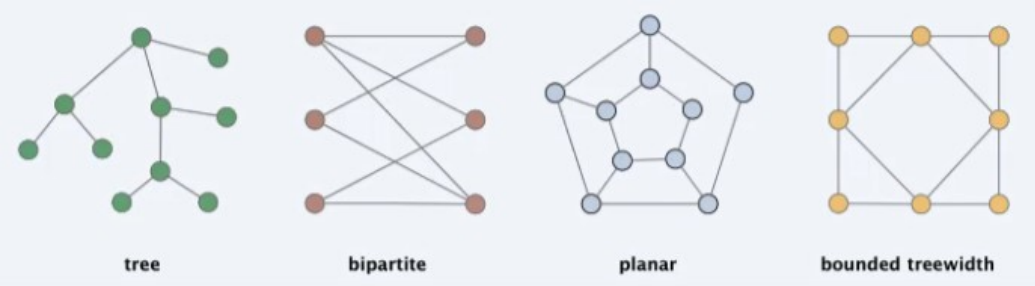
درخت ها: پوشش راسی ـ مجموعه‌ی مستقل [۱] ـ طولانی‌ترین مسیر [۲]

گراف‌های دوبخشی [۳] : پوشش راسی ـ مجموعه‌ی مستقل ـ ۳ رنگ‌پذیری [۴] ـ رنگ آمیزی یال‌ها [۵]

گراف‌های مسطح [۶] : برش بیشینه [۷] ـ ۴ رنگ‌پذیری [۸]

گراف‌های با عرض درختی کراندار [۹] : دور هامیلتونی [۱۰] ـ مجموعه‌ی مستقل

اعداد با اندازه‌ی کم: جمع زیرمجموعه‌ها [۱۱] ـ کوله‌پشتی [۱۲] ـ تقسیم [۱۳]



برای مثال بسیاری از مسائلی که الگوریتم چندجمله‌ای دارند در گراف‌های مسطح الگوریتم با زمان اجرای بهتری دارند و یا اندازه‌ی کوچکترین پوشش راسی در گراف‌های دوبخشی برابر با اندازه‌ی بزرگترین تطابق موجود در گراف است. همچنین دیده بودیم که بعضی از مسائل الگوریتم شبه‌چندجمله‌ای دارند پس اگر اندازه‌ی ورودی کوچک باشد در عمل می‌توانیم این مسائل را در زمان چندجمله‌ای حل کنیم.

با این حال، در بسیاری از مواقع خواسته‌ی اصلی ما حل حالت کلی مسئله است. حتی ممکن است با این که ورودی ما حالت کلی نیست تصریح دقیق آن کاری دشوار باشد. برای مثال گراف‌های ورودی حالت کلی نباشند اما توصیف این که این گراف در چه رده‌ای از گراف‌های مورد نظر قرار می‌گیرد کار دشواری باشد.

۲. الگوریتم نمایی خوب برای مسئله طراحی کنیم.

برای مثال برای حل مسئله‌ی فروشنده‌ی دوره‌گرد [۱۴] الگوریتم نمایی از مرتبه‌ی زمانی $O(poly(n) \times 2^n)$ ارئه دادیم که نسبت به الگوریتم بدوی از اردر $O(n!)$ ، الگوریتمِ نمایی بهتری بود . به طور خاص روش پسگرد [۱۵] و روش انشعاب‌وکران [۱۶] در این دسته قرار میگیرند. با انتخاب این روش چندجمله‌ای بودن زمان الگوریتم را فدا می کنیم. خیلی از اوقات نیز در حقیقت سعی می‌کنیم الگوریتم ارائه شده بر حسب کل ورودی نمایی نباشد، بلکه بر حسب پارامتر یا پارامترهایی از ورودی نمایی باشد.

[1] Independant Set
[2] Longest Path
[3] Bipartite
[4] 3 Coloring
[5] Edge Coloring
[6] Planar
[7] Max Cut
[8] 4 Coloring
[9] Bounded Treewidth
[10] Hamiltonian Cycle
[11] Subset Sum
[12] Knapsack
[13] Partition
[14] Travelling Salesman Problem (TSP)
[15] Backtracking
[16] Branch And Bound

۳. الگوریتم تقریبی برای مسئله ارائه دهیم.

با انتخاب این روش دقیق بودن الگوریتم را فدا می‌کنیم. این روش برای حل مسائل بهینه‌سازی کاربرد دارد و به وضوح نمی‌توان در حل مسائل تصمیم‌گیری از آن بهره برد.

۴. از روش مکاشفه‌ای [۱] استفاده کنیم.

در این حالت ایده‌هایی ارائه می‌دهیم که مسئله را بهینه‌تر حل کنند اما زمان اجرای چنین الگوریتم‌هایی و همچنین میزان نزدیکی آنها به جواب واقعی دقیقاً مشخص نیست. به طور کلی معمولاً تحلیل‌های تئوری خوبی برای آنها نداریم. به طور خاص روش‌های جست‌وجوی محلی [۲] در این دسته قرار می‌گیرند. در حالتی که برای الگوریتم‌ها تحلیل خوبی داریم مشابه حالت ۳ از حل دقیق مسئله دست برداشتیم اما اگر تحلیل خوبی نیز برای این الگوریتم‌ها نداشته باشیم در واقع از لحاظ تئوری از حل مسئله دست کشیدیم.

۲ حل مسائل با روش پسگرد

در بسیاری از موارد، مسائلی داریم که الگوریتم خوبی برای آنها وجود ندارد، اما معمولاً همیشه الگوریتم اصطلاحاً «بد»ی برای آنها وجود دارد. روش پسگرد در حقیقت جست‌وجوی هوشمندانه‌ی همه‌ی حالت‌ها است. درواقع در این روش حالت‌هایی را که به وضوح نمی‌توانند جواب مسئله باشند را بررسی نمی‌کنیم. این روش برای حل مسائل جست‌وجو به کار می‌رود. مثال زیر جست‌وجوی هوشمندانه‌ی همه‌ی حالت‌ها برای حل یک مسئله‌ی SAT نشان می‌دهد.

مثال ۱. مسئله‌ی SAT با n متغیر را در نظر بگیرید. برای پاسخ به این سوال که آیا مقداردهی‌ای وجود دارد که تمام شروط مسئله را ارضا کند می‌توان همه‌ی 2^n حالت ممکن را بررسی کرد، که این روش را جست‌وجوی همه‌ی حالت‌ها [۳] می‌نامند.

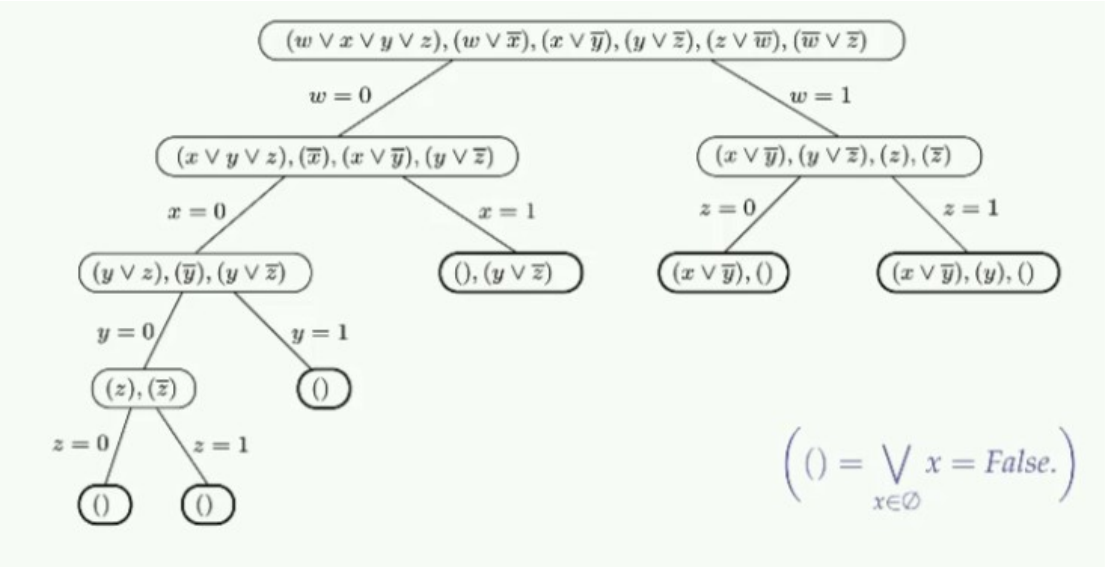
$$\phi = (w \vee x \vee y \vee z) \wedge (w \vee \bar{x}) \wedge (x \vee \bar{y}) \wedge (y \vee \bar{z}) \wedge (z \vee \bar{w}) \wedge (\bar{w} \vee \bar{z})$$

یک متغیر دلخواه را در نظر می‌گیریم و هر دو حالت $true$ و $false$ (یا ۰ و ۱) را برای آن امتحان می‌کنیم. مثلا اگر $w = false$ آنگاه عبارت پنجم و ششم ارضا می‌شوند و برای عبارت‌های اول و دوم یکی از متغیرهایی که می‌توانسته آنها را ارضا کند از بین می‌رود. پس هردو عبارت پنجم و ششم را حذف می‌کنیم و از عبارت‌های اول و دوم نیز عبارت w را حذف می‌کنیم و به عبارت زیر می‌رسیم:

$$\phi' = (x \vee y \vee z) \wedge (\bar{x}) \wedge (x \vee \bar{y}) \wedge (y \vee \bar{z})$$

با همین ایده می‌توان یک درخت ساخت:

[1] Heuristic
[2] Local Search
[3] Exhaustive Search or Brute Force Search



اگر در گره‌ای از درخت به یک عبارت تهی رسیدیم به این معناست که با مقداردهی داده شده برای متغیرها تا این مرحله مقداردهی معتبری وجود ندارد (در واقع هیچ متغیری دیگر وجود ندارد که بتواند آن عبارت را ارضا کند).

می‌توان دید که در این روش همه‌ی حالات بررسی نمی‌شوند. برای مثال برای تمام مقداردهی‌هایی که در آن‌ها $z = 0$ و $w = 1$ فقط یک بار زمان صرف می‌شود زیرا مقداردهی معتبری با $z = 0$ و $w = 1$ وجود ندارد و نیازی نیست حالات مختلف x و y را بررسی کرد.

صورت کلی الگوریتم پسگرد برای حل مسائل به صورت زیر است. مجموعه‌ای از تمام زیرمسئله‌های فعال تعریف می‌کنیم و تا وقتی تهی نشده است، هربار زیر مسئله‌ای از بین تمام زیرمسئله‌های فعال انتخاب می‌کنیم، آن را از این مجموعه حذف می‌کنیم و از روی آن تعدادی زیرمسئله دیگر می‌سازیم. سپس به کمک تابعی تمام این زیرمسئله‌ها را بررسی می‌کنیم و با توجه به جواب تابع برای آن زیر مسئله تصمیم می‌گیریم که آیا آن را به مجموعه‌ی زیرمسئله‌های فعال اضافه کنیم یا نه. گاهی نیز ممکن است این زیرمسئله به منزله‌ی جواب مسئله‌ی ما باشد و در آن صورت الگوریتم را متوقف می‌کنیم و جواب را اعلام می‌کنیم.

```
BACKTRACK()
1  Start with some problem P_0
2  S ← {P_0} // the set of active subproblems
3  while S ≠ ∅
4      Choose a P ∈ S; S ← S − {P}
5      Expand P into P_1, ... , P_k
6      for i = 1 to k
7          case TEST(P_i) of
8              success: announce solution and halt
9              failure: discard P_i
10             uncertainty: add P_i to S
11 Announce that there is no solution.
```

برای مثال درمسئله‌ی SAT داریم:

- **success** : به حالتی رسیدیم که کل عبارت ارضا شده.
- **failure** : مقداردهی معتبری برای کل عبارت با توجه به مقداردهی‌های فعلی متغیرها تا این مرحله وجود ندارد.
- **uncertainty** : هیچ یک از دو حالت بالا رخ نداده (و معلوم نیست آیا از این طریق می‌توان به جواب زسید یا نه).

در روند حل این مسئله یک گراف را پیمایش کردیم. نکته‌ای که باید به آن دقت کنید این است که راه‌های مختلفی برای رسیدن به یک گره در این گراف وجود دارد. در این روش پیمایش، در واقع یک درخت جست‌وجو می‌سازیم و آن را پیمایش می‌کنیم تا به گره‌ای برسیم که در آن هیچ عبارتی نیست. برای اینکه در کوتاه‌ترین زمان ممکن به جواب برسیم می‌توانیم توابع $Expand$ و $Choose$ را هوشمندانه انتخاب کنیم (در واقع به صورتی پیمایش کنیم که فکر می‌کنیم زودتر به جواب می‌رسد).

هدف روش پسگرد این است که تعدادی از گره‌ها را بررسی نکنیم. برای مثال می‌توان اولویتی روی گره‌ها تعیین کرد به طوری که گره‌هایی با تعداد متغیر کمتر اولویت بالاتری داشته باشند و تابع **Choose** گره با اولویت بالاتر را انتخاب کند. در تابع **Expand** نیز متغیری از عبارتی با تعداد متغیر کمتر انتخاب می‌کنیم و آن را مقداردهی می‌کنیم. تعریف پیشنهاد شده برای توابع **Expand** و **Choose** به طور شهودی مناسب به نظر می‌رسد.

برای مثال، حالت خاص‌تر مسئله‌ی SAT یعنی 3-SAT را بررسی می‌کنیم. شبه‌کدی برای الگوریتم اولیه‌ی این مسئله به صورت زیر است:

3-SAT(Φ)

1 **if** Φ is empty **return** TRUE
2 $(l_1 \vee l_2 \vee l_3) \wedge \Phi' \leftarrow \Phi$
3 **if** 3-SAT($\Phi' \mid l_1 = \text{TRUE}$) **return** TRUE
4 **if** 3-SAT($\Phi' \mid l_2 = \text{TRUE}$) **return** TRUE
5 **if** 3-SAT($\Phi' \mid l_3 = \text{TRUE}$) **return** TRUE
6 **return** FALSE

دقت کنید این الگوریتم حتی بدتر از جست‌وجوی همه‌ی حالت ها ($O(2^n)$) است:

قضیه ۱. *زمان اجرای این الگوریتم از اردر* $O(poly(n) \times 3^n)$ *است*

اثبات. از حل رابطه‌ی بازگشتی $T(n) \leq 3T(n-1) + poly(n)$ به زمان موردنظر می‌رسیم. □

می‌توان مشاهده کرد که اگر 3-SAT($\Phi' \mid l_1 = \text{TRUE}$) ، FALSE برگرداند، به این معناست که که مقداردهی معتبری به شرط $l_1 = \text{TRUE}$ وجود ندارد و درنتیجه $l_1 = \text{FALSE}$ است. پس در خط بعدی می‌توان 3-SAT($\Phi' \mid l_1 = \text{FALSE},\ l_2 = \text{TRUE}$) را فراخوانی کرد. مشابهاً اگر این فراخوانی FALSE برگرداند می‌توان دید که l_2 باید FALSE باشد. پس در خط بعدی می‌توانیم تابع 3-SAT($\Phi' \mid l_1 = \text{FALSE},\ l_2 = \text{FALSE},\ l_3 = \text{TRUE}$) را بررسی کنیم. شبه‌کد این الگوریتم به صورت زیر است:

3-SAT(Φ)

1 **if** Φ is empty **return** TRUE
2 $(l_1 \vee l_2 \vee l_3) \wedge \Phi' \leftarrow \Phi$
3 **if** 3-SAT($\Phi' \mid l_1 = \text{TRUE}$) **return** TRUE
4 **if** 3-SAT($\Phi' \mid l_1 = \text{FALSE},\ l_2 = \text{TRUE}$) **return** TRUE
5 **if** 3-SAT($\Phi' \mid l_1 = \text{FALSE},\ l_2 = \text{FALSE},\ l_3 = \text{TRUE}$) **return** TRUE
6 **return** FALSE

قضیه ۲. *زمان اجرای این الگوریتم از مرتبه‌ی زمانی* $O(1.84^n)$ *است*

اثبات. از حل رابطه‌ی بازگشتی $T(n) \leq T(n-2) + T(n-3) + O(m+n)$ به زمان موردنظر می‌رسیم. □

که از $O(2^n)$ بهتر است. برای 3-SAT الگوریتم از $O((\frac{4}{3})^n)$ نیز وجود دارد.

با این که گاهی خیلی نمی‌توانیم روی تئوری تحلیل همچین مسائلی دقیق باشیم، اما در عمل روی همچین الگوریتم‌هایی بسیار کار شده است. حتی برای مسئله‌ی SAT «حل کننده [1] »های اختصاصی زیادی ارائه شده است که در عمل خیلی خوب مسئله را حل می‌کنند و آنقدر زیاد هستند که بررسی آنها از حوصله‌ی بحث خارج است. حتی گاهاً مسابقاتی نیز بین این اصطلاحاً «حل کننده»ها برگزار می‌شود.

مثال ۲. یک صفحه ی $n \times n$ را در نظر بگیرید. می‌خواهیم n وزیر را در این صفحه قرار دهیم به طوری که هیچ دو وزیری یکدیگر را تهدید نکنند. دو وزیر یکدیگر را تهدید می‌کنند اگر در یک سطر یا ستون یا قطر یکسان باشند.

اگر بخواهیم بدون هوشمندی خاصی الگوریتمی ارائه دهیم می توانیم همه‌ی دنباله‌های n تایی که هر عضو آن عددی بین ۱ تا n است را بررسی کنیم. یعنی اگر دنباله برابر $(a_1, ..., a_n)$ باشد a_i بیانگر این است که وزیر مربوط به ستون i ام در سطر a_i ام قرار بگیرد. اگر کمی هوشمندانه‌تر بررسی کنیم کافی است همه‌ی جایگشت‌های ۱ تا n را بررسی کنیم. زیرا هیچ دو وزیری نباید در یک سطر باشند و کافی است تهدیدهای قطری را بررسی کنیم.

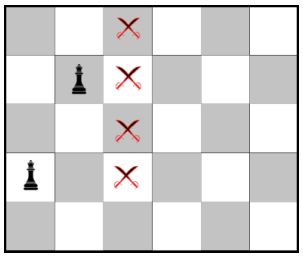
حال می‌خواهیم این مسئله را با روش پسگرد حل کنیم. آرایه‌ی Q را تعریف می‌کنیم به طوری که $Q[i]$ نشان‌دهنده‌ی این است که وزیر ستون i ام در کدام سطر قرار دارد. اگر c بیانگر شماره‌ی ستون باشد آنگاه شبه‌کد الگوریتم ما به صورت زیر خواهد بود. درواقع این الگوریتم به نحوی یک جست‌وجوی اول عمق [2] در درخت حالات است.

[1] SAT Solver
[2] Depth First Search (DFS)

```
BACKTRACK(Q, c)
1  if c == n + 1
2      ans += 1
3  else
4      for j = 1 to n
5          flag = TRUE
6          for i = 1 to c − 1
7              if Q[i] == j or |Q[i] − j| == c − i
8                  flag = FALSE
9                  break
10         if flag == TRUE
11             Q[i] = j
12             BACKTRACK(Q, c + 1)
```

خط هفتم کد بررسی می‌کند که اگر وزیر ستون c در سطر j ام باشد وزیرهای ستون‌های ۱ تا $c-1$ را تهدید می‌کند یا نه.که شرط اول تهدید سطری و شرط دوم تهدید قطری را بررسی می‌کند.

اما این الگوریتم را می‌توانیم بسیار بهینه‌تر کنیم. در الگوریتم فوق تمام خانه‌های ستون c ام را بررسی می‌کنیم و چک می‌کنیم اگر وزیر را در این خانه قرار دهیم، وزیرهای ستون‌های عقب‌تر را تهدید می‌کند یا نه. به جای این کار می‌توانیم با روشی از قبل تمام این شرط‌ها را محاسبه کنیم. برای مثال در شکل زیر بعد از قرار دادن وزیر ستون ستون اول در سطر چهارم، دیگر نمی‌توانیم وزیری را در این سطر قرار دهیم پس اگر بخواهیم موقعیت‌های ممکن برای وزیر ستون سوم را بررسی کنیم، سطر چهارم را بررسی نخواهیم کرد. به همین ترتیب در ستون سوم سطر دوم را بررسی نخواهیم کرد چرا که وزیر ستون اول را قطری تهدید می کند. به طور مشابه خانه‌هایی که وزیر ستون دوم را تهدید می‌کند را بررسی نمی‌کنیم پس در کل فقط کافی است سطر پنجم را بررسی کنیم.



برای پیاده‌سازی این ایده آرایه‌های صفر و یکی زیر را در نظر می گیریم:

- rw (row) :کدام خانه از ستون c به طور سطری توسط وزیر های ستون های ۱ تا $c-1$ تهدید می شوند.
- ld (left diagonal) :کدام خانه‌ها از ستون c به طور قطری در جهت ↗ توسط وزیرهای ستون‌های ۱ تا $c-1$ تهدید می‌شوند.
- rd (right diagonal) :کدام خانه‌ها از ستون c به طور قطری در جهت ↘ توسط وزیرهای ستون‌های ۱ تا $c-1$ تهدید می‌شوند.

می توان به جای آرایه‌های صفر و یکی فوق ۳ عدد در نظر گرفت و عملیات روی آرایه‌ها را با عملیات بیتی پیاده‌سازی کرد. در ابتدای کار دو متغیر $AllOne$ و ans را به صورت زیر مقداردهی می‌کنیم.

$AllOne = \underbrace{111...1}_{n}$

$ans = 0$

حال شبه‌کد الگوریتم جدید به صورت زیر خواهد بود:

BIT-BACK(rw, ld, rd)

```
1  if rw = AllOne
2        ans += 1
3  pos = AllOne  &  (~ (rw | ld |rd))
4  while pos
5        p = pos  &  (-pos)
6        pos ^= p
7        BIT-BACK(rw|p,  (ld|p) >> 1,  (rd|p) << 1)
```

می‌توان در خط اول عبارت شرط را با $c = n + 1$ جایگزین کرد. اما دلیل $\&$ کردن با $AllOne$ در خط سوم چیست؟ درواقع در کامپیوتر تعداد بیت‌ها معمولا عدد ثابتی است و هنگامی که در خط ۳، نقیض آن عبارت را محاسبه می‌کنیم، ممکن است تعداد زیادی ۱ اضافه در آخر عدد ایجاد شود که با این کار آنها را از بین می‌بریم.

این کد را با شکلی که در بالا آمده بود بررسی می‌کنیم. قبل از بررسی ستون سوم مقادیر ۳ آرایه به صورت زیر است :

$rw = 01010$

$ld = 00011$

$rd = 00100$

و در نتیجه خواهیم داشت:

$pos = 10000 \quad \Rightarrow p = 10000$

که این دقیقاً نشان‌دهنده‌ی تنها خانه‌ای است که می‌توانیم وزیر را در آن قرار دهیم. پس از قرار دادن وزیر ستون سوم آرایه ها به صورت زیر تغییر می‌کنند:

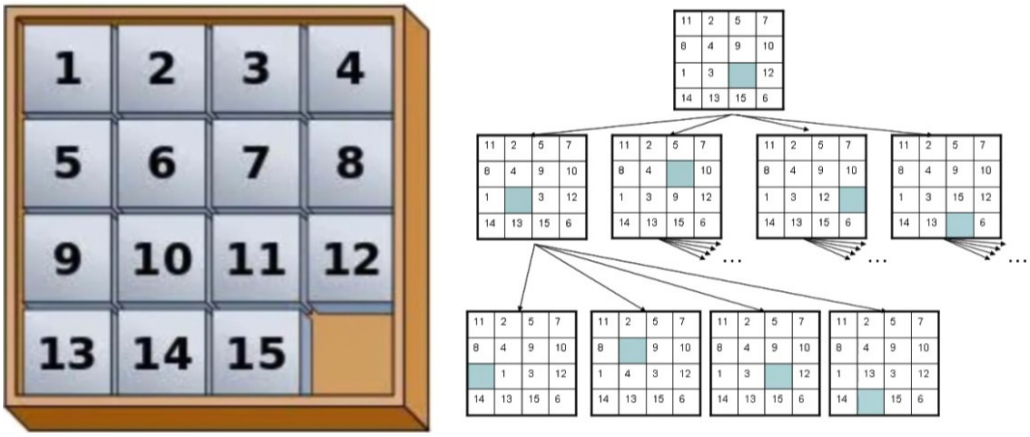
$rw = rw|p = 11010$

$ld = (ld|p) \gg 1 = 01001$

$rd = (rd|p) \ll 1 = 01010$

مثال ۳. مسئله‌ی پازل ۱۵ تایی را می‌توان با الگوریتم‌های پیمایش گراف مانند جست‌وجوی اول سطح [1] یا جست‌وجوی اول عمق حل کرد.

[1] Breadth First Search (BFS)



اگر به ازای هر حالت بازی یک راس در یک گراف در نظر بگیریم و تمام حالاتی که با یک حرکت به هم تبدیل می‌شوند را با یک یال در این گراف به هم وصل کنیم، گرافی خواهیم داشت که تمام حالات این پازل را نمایش می‌دهد (در واقع گراف را به طور ضمنی داریم). درچنین شرایطی یک درخت جست‌وجو تعریف می‌کنیم که تنها نشان می‌دهد از یک گره (حالت) به چه گره‌های دیگری می‌توانیم برویم (درخت شکل بالا). در نتیجه ممکن است در همچین درختی راس‌های تکراری زیادی داشته باشیم (چون از راه‌های زیادی می‌توان به یک حالت خاص رسید). دقت کنید که ما نمی‌توانیم خود گراف اصلی را پیمایش کنیم زیرا حافظه‌ای نمایی نیاز دارد که معمولا آن را در اختیار نداریم. به منظور بهینه کردن حافظه‌ی الگوریتم برخلاف جست‌وجوی اول عمق (و سطح) راس‌ها را نشانه‌گذاری نمی‌کنیم درنتیجه ممکن است به راس‌های تکراری برسیم (که این معادل است با این که در درخت جست‌وجوی گفته شده پیمایش را انجام دهیم). در این حالت ممکن است الگوریتم هیچ‌گاه متوقف نشود. درنتیجه در الگوریتم جست‌وجوی اول عمق یک محدودیت برای عمق جست‌وجو اعمال می‌کنیم. برای مثال فقط گره‌هایی تا عمق d را بررس می‌کنیم. همچنین فقط راس‌هایی که در مسیر ریشه تا راس فعلی قرار گرفته اند را نگهداری می‌کنیم در نتیجه حافظه‌ی مورد استفاده برای جست‌وجوی اول عمق از $O(d)$ است.

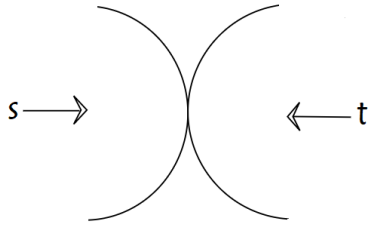
برای بررسی حافظه‌ی مصرفی جست‌وجوی اول سطح ضریب انشعاب [1] را برای درخت تعریف می‌کنیم که برابر با تعداد حالت‌هایی از پازل است که از حالت فعلی می‌توان به آن رسید. اگر این مقدار برابر b باشد، حافظه‌ی مصرفی جست‌وجوی اول سطح از $O(b^d)$ خواهد بود. به وضوح حافظه‌ی جست‌وجوی اول عمق کمتر است اما سوالی که وجود دارد این است که مقدار d را چگونه تعیین کنیم. یک راه استفاده از الگوریتم جست‌وجوی اول عمق تکرارشونده [2] است، به این صورت که ابتدا برای $d = 1$ بررسی می‌کنیم که آیا با جست‌وجو تا حداکثر عمق ۱ به جواب می‌رسد یا نه. اگر به هدف نرسیدیم الگوریتم را با $d = 2$ اجرا می‌کنیم و این روند را آن قدر ادامه می‌دهیم که به هدف خود برسیم.

در نگاه اول به نظر می‌رسد که ما برای عمق‌های اولیه چندبار تمام راس‌ها را پیمایش می‌کنیم و شاید بهتر باشد از همان ابتدا d را عدد بزرگتری در نظر بگیریم، اما درواقع الگوریتم برای $d = 1$ ، b راس، برای $d = 2$ ، b^2 راس و به همین ترتیب برای $d = k$ ، b^k راس را بررسی می‌کند. حال چون از $O(b^k)$ است، پس در کل زمان اجرای کل الگوریتم از اردر بررسی برای $d = k$ خواهد بود. از نظر حافظه نیز اگر راس متناظر با حالت هدف در عمق k باشد، حافظه‌ی مصرفی از $O(k)$ است.

حال برای بهینه کردن این الگوریتم، ایده‌ی دیگری که به ذهن می‌رسد این است که برای هر حالت از پازل یک عدد در نظر بگیریم که به نوعی نشان‌دهنده‌ی فاصله‌ی حالت فعلی از حالت نهایی است. برای نمونه، یک روش به این صورت است که جایگاه این ۱۵ عدد در حالت فعلی را در نظر می‌گیریم. برای هر عدد کمترین تعداد حرکات لازم برای رسیدن به جایگاه آن عدد در حالت نهایی در صورتی که هیچ مانعی بر سر راه عدد نباشد را محاسبه می‌کنیم و عدد (فاصله) مورد نظر این حالت را جمع فاصله‌ی اعدادش از جایگاه‌شان در حالت نهایی تعریف می‌کنیم. در نتیجه عددی که برای هر حالت تعریف کردیم (فاصله) نشان‌دهنده‌ی یک کران پایین برای تعداد حرکات لازم برای رسیدن از حالت فعلی به حالت هدف است.

[1] Branching Factor
[2] Iterative Deepening DFS

حال در گراف از هر راس ابتدا فرزندی را بررسی می‌کنیم که فاصله‌ی کمتری از حالت هدف دارد. می‌توان نشان داد که اگر این فاصله واقعاً کران پایین تعداد حرکات لازم برای رسیدن به هدف باشد، این الگوریتم بهینه است.
ایده‌ی دیگر برای حل این مسئله استفاده از ایده‌ی ملاقات در وسط [1] است یعنی برای پیدا کردن مسیر بین s و t می‌توان هم‌زمان از هر دو راس s و t گراف را با جست‌وجوی اول سطح یا جست‌وجوی اول عمق پیمایش کرد و در نقطه‌ای که دو مسیر پیمایش به هم می‌رسند الگوریتم را متوقف کرد.



با این روش می‌توان زمان الگوریتم جست‌وجوی اول سطح را از $O(b^k)$ به $O(b^{\frac{k}{2}})$ کاهش داد.

روش ملاقات در وسط در نگاه اول مانند روش تقسیم و حل به نظر می‌رسد اما تفاوت آن‌ها در این است که در الگوریتم‌های تقسیم و حل مسئله‌ای که بعد از تقسیم به دست می‌آید دقیقا از جنس مسئله‌ی اولیه است در صورتی که در روش ملاقات در وسط نمی‌توان مساله‌ی به دست آمده را به مسئله‌های کوچکتر تقسیم کرد (مفهوم متفاوتی دارد).

حال می‌خواهیم مسئله ی زیر را با روش ملاقات در وسط حل کنیم:

مثال ۴. ورودی : $S = \{s_1, ..., s_n\}$
خروجی : آیا می‌توانیم S را به ۴ مجموعه‌ی A , B , C , D افراز کنیم به طوری که داشته باشیم:

$$\sum_{x \in A} x = \sum_{x \in B} x = \sum_{x \in C} x = \sum_{x \in D} x$$

الگوریتم بدوی : برای هر عضو از S ، ۴ حالت عضویت در یکی از مجموعه‌های A , B , C , D وجود دارد. اگر بخواهیم همه‌ی حالت‌های ممکن را بررسی کنیم زمان اجرای الگوریتم از $O(4^n)$ خواهد بود. می‌خواهیم با ایده‌ی ملاقات در وسط الگوریتم با زمان اجرای $O(4^{\frac{n}{2}})$ ارائه دهیم.

همه‌ی حالت‌های $\frac{n}{2}$ عدد اول را در یک جدول درهم‌سازی H ذخیره می‌کنیم (در جلسات ابتدایی در مورد ذخیره کردن انواع متغیرها در جدول درهم‌سازی صحبت کرده بودیم، که این کار نیز از $O(1)$ انجام می‌شد) سپس همه ی حالت‌های $\frac{n}{2}$ عدد دوم را یکی‌یکی در نظر می‌گیریم و برای هر کدام یکی‌یکی چک می‌کنیم که آیا مکملش در H موجود است یا نه.

$$(a, b, c, d) \in H \xrightarrow{?} (a - t, b - t, c - t, d - t) \in H$$

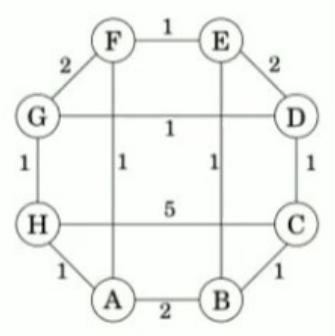
که در آن $t = \frac{\sum_{x \in S} x}{4}$. پس در $O(1)$ می‌توانیم بررسی کنیم که آیا حالت در نظر گرفته شده برای هر چینش $\frac{n}{2}$ عدد دوم معتبر است یا نه. هر دو مرحله زمانی از $O(4^{\frac{n}{2}})$ می‌گیرند و درنتیجه در کل الگوریتم از $O(4^{\frac{n}{2}})$ است. دقت کنید که هرچند در ظاهر پیشرفت خیلی بزرگی نکرده‌ایم، اما در عمل مثلا برای $n = 20$ الگوریتم بدوی تقریباً غیرقابل اجرا است اما الگوریتم دوم امکان عملی شدن دارد.

[1] Meet In The Middle

۳ حل مسائل با روش انشعاب و کران

اشاره کردیم که روش پسگرد برای حل مسائل جست‌وجو به کار می‌رود. برای مسائل بهینه سازی از چه روشی می‌توانیم استفاده کنیم؟ در واقع روش انشعاب و کران به کار بردن ایده‌ی روش پسگرد در مسائل بهینه‌سازی است. برای مثال مسئله‌ی فروشنده‌ی دوره‌گرد را در نظر بگیرید. اگر در مراحل پیدا کردن جواب بهینه، تا اینجای کار یک جواب پیدا کرده باشیم (لزوماً بهینه نیست) و در حال ساخت یک جواب دیگر برای مسئله باشیم (یعنی بخشی از دور را ساخته باشیم)، اگر جواب فعال (جوابی که تا الان بخشی از آن را ساخته‌ام) شانسی برای بهتر شدن از جوابی که تا الان پیدا کرده‌ایم نداشته باشد، دیگر آن را ادامه نمی‌دهیم. یعنی مشابه مسائل قبل یک درخت از وضعیت‌های مختلف در نظر می‌گیریم و آن برای پیدا کردن جواب بهینه پیمایش می‌کنیم. در هر مرحله اگر یکی از گره‌های میانی شانس نزدیکتر شدن به جواب اصلی نسبت به جوابی که تا به حال پیدا کرده‌ایم را نداشته باشد، آن شاخه را ادامه نمی‌دهیم و تنها شاخه‌هایی که شانس رساندن ما به جواب بهینه دارند را بررسی می‌کنیم.

مثال زیر را در نظر بگیرید. می‌خواهیم دور هامیلتونی در گراف بدون جهت زیر را پیدا کنیم. چون دور هامیلتونی شامل تمام راس‌ها می‌شود فرقی نمی‌کند که از کدام راس شروع می‌کنیم. پس بدون کاستن از کلیت فرض کنید از A دور را شروع کنیم. هر زیرمسئله را به صورت (A, S, t) تعریف می‌کنیم. که t راسی است که تا الان از A به آن رسیده‌ایم و S مجموعه رئوسی هستند که در مسیر بین A و t آمده‌اند.

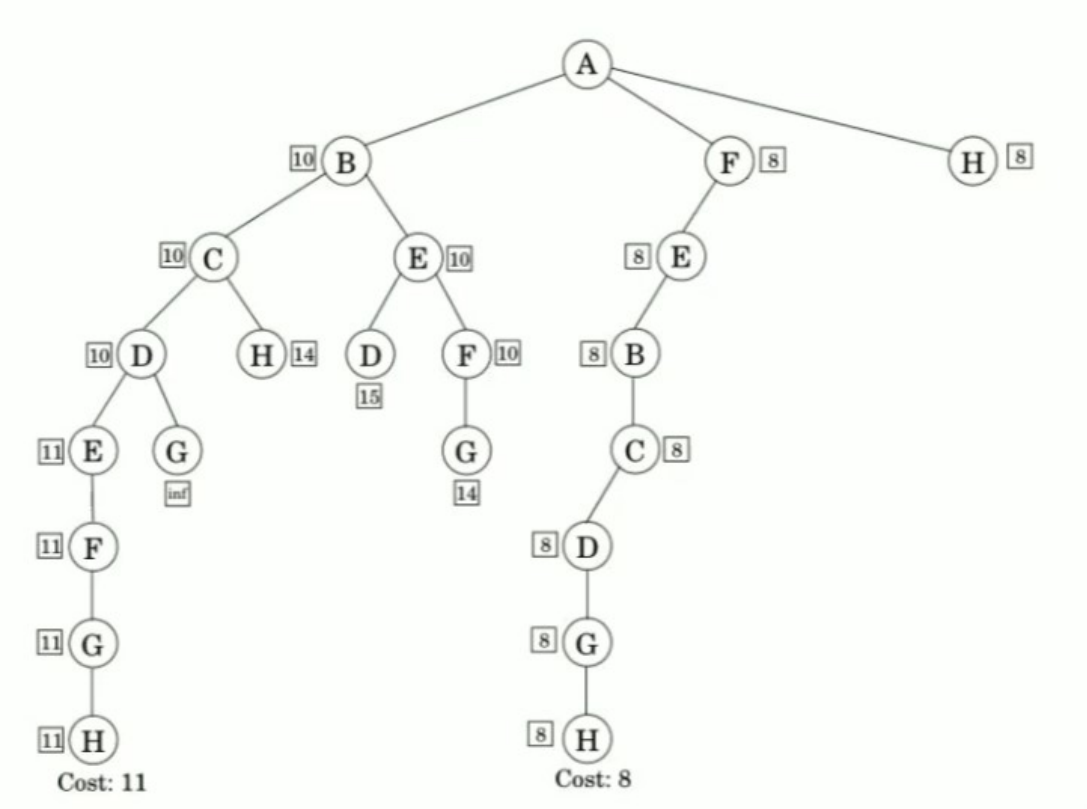


حال درخت زیر را در نظر بگیرید، اعداد ثبت شده در کنار راس ها بیانگر یک کران پایین برای هزینه‌ی یک دور هامیلتونی هستند. مثلاً برای B عدد ۱۰ ثبت شده که این به معناست که اگر از A به B برویم هزینه‌ی دور هامیلتونی در نهایت بزرگتر یا مساوی ۱۰ خواهد شد. فرض کنید درخت را با جست‌وجوی اول عمق پیمایش کنیم از A به B به C به D به E به F به G و در نهایت به H یک دور هامیلتونی با هزینه‌ی ۱۱ است. در درخت با برگشت به سمت بالا از H به D می‌رسیم. می‌توان دید که در این حالت اگر به G برویم نمی‌توانیم یک دور هامیلتونی را کامل کنیم. سپس از D به C می‌رویم. در این حالت اگر از C به H برویم، می‌بینیم که کران پایین هزینه برای ادامه این حالت ۱۴ است (یعنی کم‌هزینه‌ترین دور هامیلتونی‌ای که از وضعیت فعلی می‌توانیم به آن برسیم،حداقل هزینه‌ای برابر با ۱۴ دارد)، لذا بررسی این شاخه را ادامه نمی‌دهیم چون قبلاً یک دور هامیلتونی با هزینه‌ی کمتر از ۱۱ پیدا کرده بودیم. به همین ترتیب کل گراف را پیمایش می‌کنیم و اگر در هر مرحله برای گره‌ای به کران پایین کمتر از ۱۱ رسیدیم آن گره را ادمه می‌دهیم تا هزینه‌ی نهایی‌اش را محاسبه کنیم.

حال کافی است نحوه‌ی تعیین این کران پایین‌ها را مشخص کنیم. تعریف زیرمسئله‌ی (A, S, t) را در نظر بگیرید. برای تشکیل یک دور A باید به یکی از رئوس $V \setminus S$ و t نیز به یکی دیگر از رئوس $V \setminus S$ وصل شود. سپس این دو راس باید با مسیری که تمام راس‌های $V \setminus S$ را دربرمی‌گیرد به هم وصل شوند تا یک دور هامیلتونی ایجاد شود. برای پیدا کردن یک کران پایین تمام یال‌هایی که A را به یکی از رئوس $V \setminus S$ وصل میکنند را در نظر می‌گیریم و یال با کمترین وزن را انتخاب می‌کنیم و شبیه همین کار را برای t انجام می‌دهیم (دو یالی که A و t را به دو عضو متمایز $V \setminus S$ وصل می‌کنند و روی هم کمترین وزن ممکن را دارند). همچنین برای اعضای $V \setminus S$ یک درخت فراگیر کمینه [1] محاسبه می‌کنیم. چون هر مسیر یک درخت است و مسیری که دنبال آن هستیم حتما از تمامی راس‌ها عبور می‌کند و

[1] Minimum Spanning Tree (MST)

درنتیجه یک درخت فراگیر نیز هست (برای راس‌های $V \setminus S$)، پس هزینه‌ی آن بزرگتر مساوی با هزینه‌ی یک درخت فراگیر کمینه خواهد بود. مجموع این ۳ مقدار و هزینه‌ی مسیر از A به t در زیرمسئله‌ی فعلی برابر با کران پایین مد نظر است.



البته دقت کنید لزومی ندارد درخت را با جست‌وجوی اول عمق پیمایش کنیم، می‌توانیم همه‌ی همسایه‌های A را بسازیم و ابتدا آن همسایه‌ای که امیدوار کننده‌تر از بقیه است را ادامه دهیم. شمای کلی همچین الگوریتم‌هایی را می‌توانید در شبه‌کد زیر ببینید:

BRANCH-AND-BOUND()

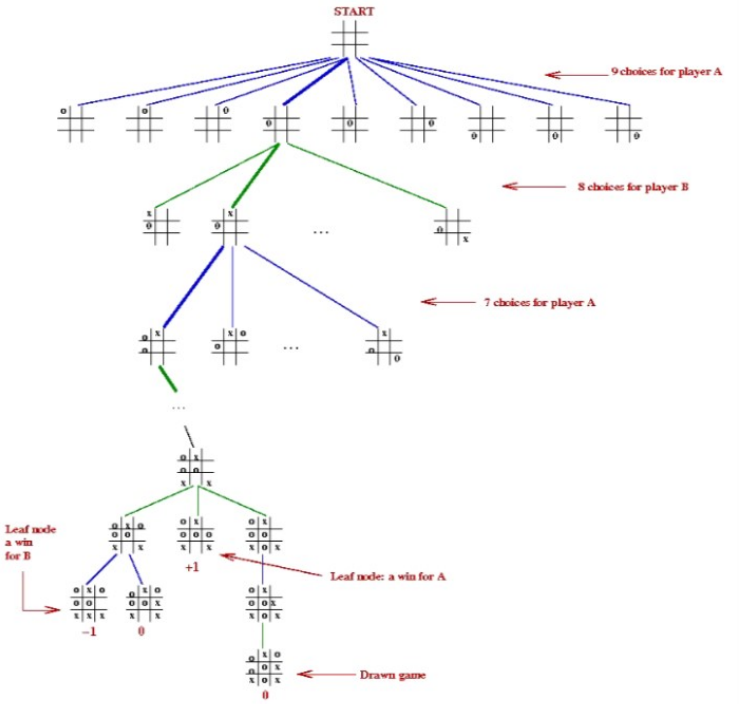
```
1  Start with some problem P_0
2  S ← {P_0} // the set of active subproblems
3  bestSoFar ← ∞
4  while S ≠ ∅
5      Choose a P ∈ S; S ← S − {P}
6      Expand P into P_1, ... , P_k
7      for i = 1 to k
8          if P_i is a complete solution then update bestSoFar
9          else if (LOWERBOUND(P_i) < bestSoFar) then add P_i to S
10     return bestSoFar
```

معمولاً معنایی که برای $Expand$ در نظر گرفته می‌شود این است که وقتی به یک گره رسیدیم تمام فرزندان آن را تولید کنیم و به S اضافه کنیم و گره اصلی را غیرفعال کنیم. در الگوریتم قبل کمی از این روش تخطی کردیم. برای اینکه دقیقاً طبق شبه‌کد ارائه شده عمل کنیم، باید برای هر گره همه‌ی فرزندان آن را به درخت اضافه کنیم.

از اینجا به بعد مطالب اختیاری هستند.

۴ الگوریتم‌های حل بازی‌های دونفره(اختیاری)

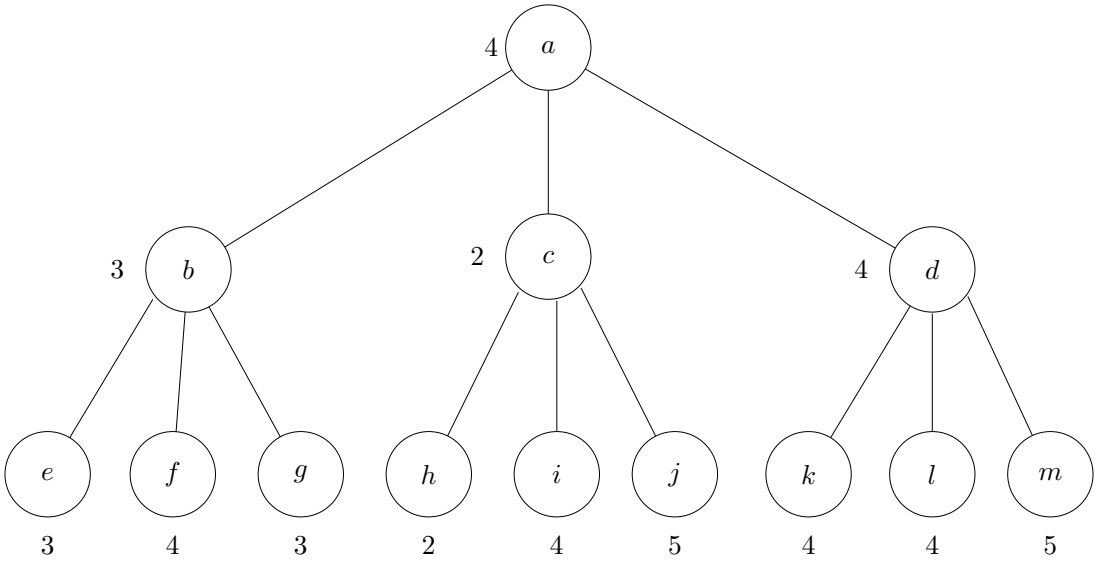
تا اینجا الگوریتم‌های ارائه شده برای حل مسائل به صورت بهینه بود. درواقع یک نفر وجود داشت که می‌خواست مقداری را بهینه کند. اما در یک بازی دونفره هر نفر می‌خواهد سود خود و ضرر طرف مقابل را بیشینه کند (ضرر خود را کمینه کند). می‌توان برای بازی‌های دونفره نیز مشابه مسائل قبل یک درخت حالت در نظر گرفت. برای مثال درخت بازی XO را مشاهده کنید.



درواقع اعداد نسبت داده شده به برگ‌ها سود (یا ضرر) نفر اول را مشخص می‌کنند. حال می‌خوهیم الگوریتمی ارائه دهیم که به جای نفر اول بازی کند (و سود و ضرر او را بهینه کند). فرض ما این است که نفر دوم نیز ایده‌آل بازی می‌کند.

الگوریتم در هر مرحله بین تمام حالت‌های ممکن حالتی را انتخاب می‌کند که منجر به بالاترین سود شود. هر حالت نیز به این ترتیب محاسبه می‌شود که به طور بازگشتی الگوریتم را روی فرزندان این حالت منتها با شروع از نفر دوم محاسبه می‌کنیم و می‌دانیم که نفر دوم می‌خواهد تا جای ممکن سود نفر اول را کم کند و بین تمام فرزندان موجود حالتی را انتخاب می‌کند که منجر به کمترین سود (بیشترین ضرر) برای نفر اول شود.

برای مثال درخت بازی خیالی زیر را در نظر بگیرید که اعداد نشان داده شده برای هر برگ نشان‌دهنده‌ی سود نفر اول در صورت رسیدن به این حالت است. نفر اول می‌خواهد بیشنه عدد ممکن را به دست بیاورد و نفر دوم می‌خواهد کمینه عدد ممکن را برای نفر اول رقم بزند.



در درخت فوق اگر در گره b باشیم که نوبت نفر دوم است بهترین وضعیت برای نفر دوم رفتن به راس e یا g است (حداکثر ۳ واحد سود برای نفر اول) زیرا کمترین سود را به نفر اول می‌رساند. اگر در گره c باشیم به دلیل مشابه نفر دوم گره h را انتخاب خواهد کرد (حداکثر ۲ واحد سود برای نفر اول). همینطور اگر در گره d باشیم، k یا l را انتخاب خواهد کرد (حداکثر سود ۴ برای نفر اول) خواهد رسید. در نتیجه نفر اول گره d را انتخاب می‌کند زیرا بیشترین سود تضمین شده را دارد (دقت کنید فرض بر این است که نفر دوم نیز بهینه بازی می‌کند).

شبه‌کد زیر شمای کلی همچین الگوریتمی را نشان می‌دهد. در الگوریتم فرض می‌کنیم که شرط برگ بودن برای یک حالت بازی مشخص است.

```
MAX(s)
1  if s is leaf
2        return s.value
3  v ← −∞
4  for a in s.children
5        v = max(v, MIN(a))
```

الگوریتم $\text{Min}(s)$ نیز تقریباً متقارن با الگوریتم بالا کار می‌کند. این دو الگوریتم را با هم بیشینه_کمینه [1] می‌نامیم. برای استفاده از این الگوریتم لازم است که درخت وضعیت بازی متناهی باشد. در غیر ای صورت الگوریتم متوقف نمی‌شود. حتی اگر تعداد تعداد انتخاب‌های بازی در هر مرحله نیز محدود باشد، درخت حالت همچین بازی‌هایی یک درخت با رشد نمایی است و بررسی آن زمان‌بر است (و گاهی غیرممکن). برای حل این مشکل می‌توان روی تعداد مراحل پایین رفتن درخت محدودیت اعمال کرد (در واقع تا جایی که توان پردازش آن را داشته باشیم). در این حالت اگر گره‌ای که به آن رسیدیم برگ نباشد می‌توانیم مثلاً تخمینی از میزان برتری نفر اول بر نفر دوم را خروجی دهیم.

پایه‌ی بیشتر الگوریتم‌های بازی‌های دو نفره در حالت کلی به صورت گفته شده است، با این حال باز هم می‌توان بهینه‌سازی‌های زیادی برای این الگوریتم‌ها ارائه داد. مشابه الگوریتم‌های قبل، پیمایش درخت را از گره‌هایی که نمی‌توانند ارزشی بیشتر (از چیزی که تا به حال به دست آورده‌ایم) برای ما بدست بیاورند ادامه نمی‌دهیم. در این نوع مسائل نیز می‌توان از همین ایده استفاده کرد.

[1] $MaxMin$

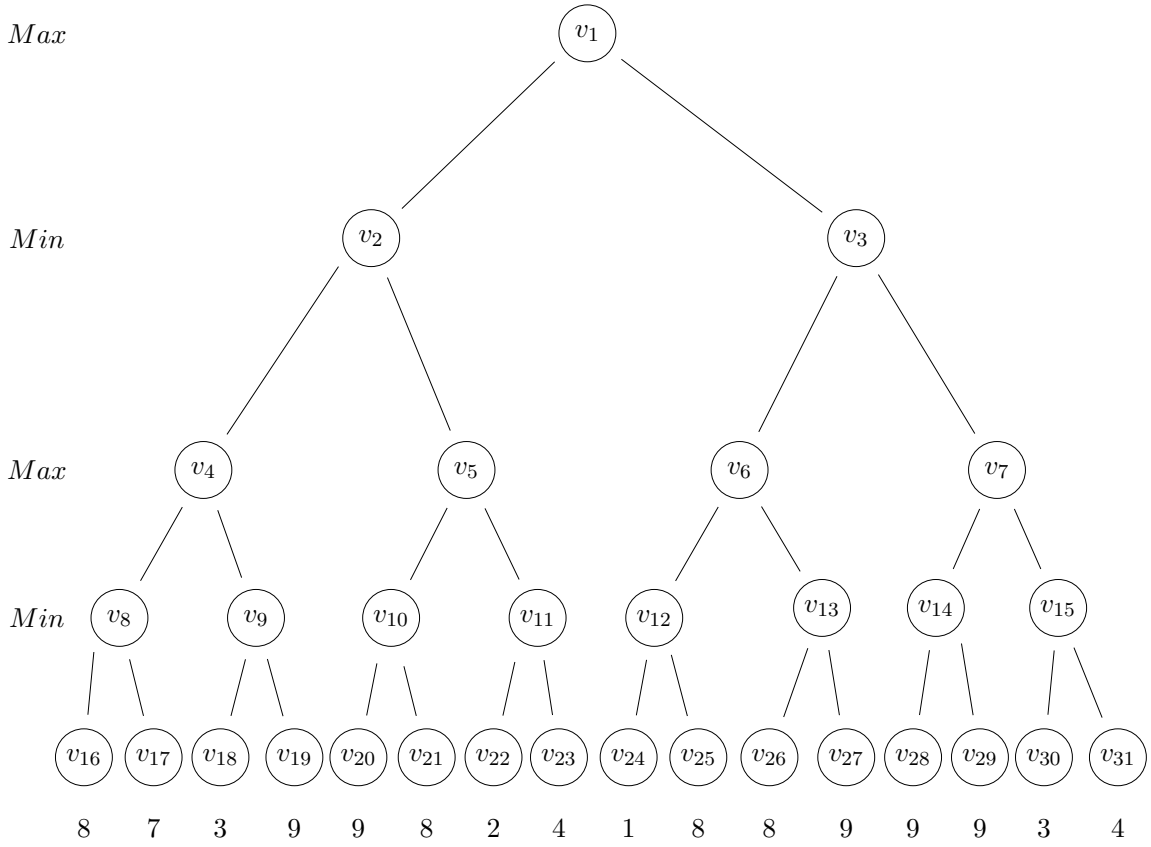
فرض کنید در مثال قبل پس از بررسی گره b به گره a برگردیم و از آنجا گره c وسپس گره h را بررسی کنیم. تا اینجای کار با توجه به سود گره‌های بررسی شده، بازیکن اول سود ۳ واحد را به دست آورده است. به محض مشاهده‌ی گره h می‌توان دریافت که بازیکن اول گره c را انتخاب نخواهد کرد، زیرا هدف نفر دوم کمینه کردن سود برای نفر است و در صورت انتخاب گره c توسط نفر اول بازیکن دوم گره h را انتخاب می‌کند و نفر اول ضرر خواهد کرد (زیرا اگر c را انتخاب نمی‌کرد و b را انتخاب می‌کرد، تضمین حداقل ۳ واحد سود را داشت). در نتیجه نیازی به بررسی گره‌های i و j نیست.

به این روش هرس آلفا_بتا [1] می‌گویند. در الگوریتم زیر α نشان‌دهنده‌ی بهترین مقداری است که یک جد Max گره فعلی می‌تواند به دست بیاورد. و β نشان‌دهنده‌ی بهترین مقداری است که یک جد Min گره فعلی می‌تواند به دست بیاورد. شمای کلی همچین الگوریتمی مانند زیر است:

```
MAX(s, α, β)
1  // handle base case...
2  v ← −∞
3  for a in s.children
4        v = max(v, MIN(a, α, β))
5        if v ≥ β
6              return v
7        α = max(α, v)
```

برای مثال در گراف زیر این الگوریتم را اجرا می‌کنیم (در تمام متن زیر خود را به جای نفر اول بگذارید). از v_1 شروع می‌کنیم به v_2 بعد از آن به v_4 سپس به v_8 و به v_{16} می‌رویم. بعد از بررسی وضعیت v_{16} می‌توان دریافت که در وضعیت v_8 نفر اول سودی کمتر مساوی ۸ به دست خواهد آورد. پس از بررسی v_{17} سود به دست آمده در وضعیت v_8 کمتر مساوی ۷ خواهد بود و چون v_8 دیگری فرزندی ندارد سود نفر اول دقیقاً ۷ واحد خواهد شد. در این مرحله راجع به وضعیت پدر v_8 یعنی v_4 می‌دانیم که سود به دست آمده در آن بزرگتر مساوی ۷ است. حال وضعیت v_9 را بررسی می‌کنیم. بعد از بررسی گره v_{18} می‌دانیم سود به دست آمده در وضعیت v_9 کمتر یا مساوی ۳ خواهد بود. در همین لحظه متوجه می‌شویم که نیازی به بررسی v_{19} نیست چرا که در وضعیت v_4 می‌توان با رفتن به فرزند چپ سود ۷ را به دست آورد پس دلیلی ندارد که به فرزند راست برویم یعنی می‌توان شاخه ی v_4 به v_9 را هرس کرد. پس در وضعیت v_2 سودی کمتر مساوی ۷ حاصل خواهد شد. حال گره v_5 را بررسی می‌کنیم. با بررسی فرزندان v_{10} سود حاصل از این وضعیت ۸ خواهد بود. لذا گره v_5 سودی بزرگتر یا مساوی با ۸ دارد. چون نفر دوم در v_2 سودی کمتر مساوی ۷ به دست آورده در نتیجه دلیلی برای بررسی وضعیتی با سود بزرگتر مساوی ۸ ندارد. پس این شاخه نیز هرس می‌شود. تا اینجای کار وضعیت ریشه سودی بزرگتر مساوی ۷ دارد. حال گره v_3 را بررسی می‌کنیم. پس ابتدا v_6 و بعد v_{12} و بعد از آن v_{24} را بررسی می‌کنیم. بعد از بررسی v_{24} می‌دانیم که ارزش v_{12} کمتر یا مساوی ۱ است. ادعا می‌کنیم که شاخه‌ی v_6 به v_{12} هرس می‌شود. چرا که تا به اینجای کار نفر اول سود ۷ را برای خود تضمین کرده است. اگر بازی به وضعیت v_{12} برسد، نفر دوم وضعیتی که منجر به سود ۱ می‌شود را انتخاب می‌کند که این اتفاق خوشایند نفر اول نیست در نتیجه نفر اول اجازه نمی‌دهد که بازی به این وضعیت برسد. پس از بررسی فرزندان v_{13} می‌دانیم سود این وضعیت ۸ است. بنابراین سود گره v_6 برابر با ۸ است. پس ارزش گره v_3 کمتر مساوی ۸ است. سپس v_7 را بررسی می‌کنیم. در ادامه سود v_{14} برابر با ۹ تعیین می‌شود. پس سود v_7 بزرگتر مساوی ۹ خواهد بود. بنابراین نفر دوم در وضعیت v_3 هرگز به سمت راست نمی‌رود، چرا که با این حرکت سودی بزرگتر یا مساوی ۹ نصیب نفر اول خواهد شد.

[1] α-β Pruning


Max
Min
Max
Min
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}
v_{13}
v_{14}
v_{15}
v_{16}
v_{17}
v_{18}
v_{19}
v_{20}
v_{21}
v_{22}
v_{23}
v_{24}
v_{25}
v_{26}
v_{27}
v_{28}
v_{29}
v_{30}
v_{31}
8 7 3 9 9 8 2 4 1 8 8 9 9 9 3 4

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

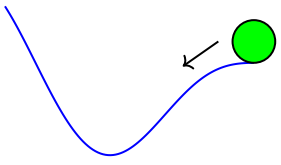
جلسه ۲۲: جست‌وجوی محلّی[1] نگارنده: امیرحسین افشارراد

موضوع این جلسه، روش جست‌وجوی محلّی برای حلّ مسائل مختلف است که در حالت کلّی روشی انعطاف‌پذیر است که برای مسائل بسیار زیادی قابل اعمال است، امّا لزوماً زمان اجرای مناسب و حتّی تضمینی برای منتهی‌شدن به پاسخ بهینه ندارد.

۱ مقدّمه

پیش‌تر روش جست‌وجوی محلّی در مبحث یافتن زیردرخت فراگیر کمینه به عنوان یک تمرین معرّفی شده است. به عنوان یادآوری، می‌توان برای یافتن زیردرخت فراگیر با وزن کمینه در یک گراف، از یک زیردرخت دلخواه شروع کرد و در هر گام، در صورتی که بتوان با حذف یک یال و اضافه کردن یالی دیگر، به زیردرخت جدیدی با وزن کم‌تر دست پیدا کرد، این کار را انجام داد تا نهایتاً دیگر امکان پیشرفت وجود نداشته باشد. می‌توان نشان داد که این الگوریتم به جواب بهینه منتهی می‌شود و علاوه بر آن، در زمان اجرای چندجمله‌ای نیز قابل انجام است (تمرین).

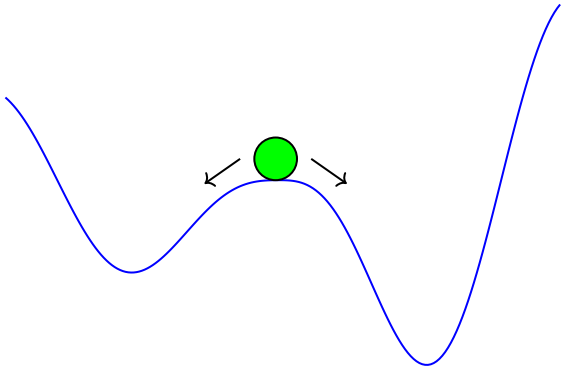
برای در نظر گرفتن الگوریتم‌های جست‌وجوی محلّی، خوب است ابتدا توجّه کنیم که یک شهود مهم مربوط به این روش از سیستم‌های فیزیکی ناشی می‌شود؛ جایی که ساختارها به سمت تعادل و رسیدن به سطوح پایین‌تر انرژی حرکت می‌کنند. به عنوان مثال شکل ۱ را در نظر بگیرید. در این شکل، اگر گوی سبزرنگ رها شود، به سمت قعر درّه حرکت می‌کند و نهایتاً در آن‌جا به پایداری می‌رسد. به عبارتی، سیستم به سمت نقطه‌ی بهینه حرکت می‌کند و این در حالی است که گوی در هنگام حرکت، از محلّ نقطه‌ی بهینه آگاهی ندارد. در واقع گوی در هر لحظه تنها تابعی از شرایط محلّی اطراف خود حرکت می‌کند و به سمتی می‌رود که ارتفاع پایین‌تری داشته باشد. به وضوح در این مثال، چنین رفتاری منجر به رسیدن به نقطه‌ی بهینه می‌شود.



شکل ۱

با این وجود می‌توان به سادگی مثال‌هایی را در نظر گرفت که چنین رفتاری به جواب بهینه‌ی مطلق مسأله همگرا نشود. به عنوان مثال، شکل ۲ را در نظر بگیرید. در این حالت اگر گوی سبزرنگ به سمت چپ حرکت کند، نهایتاً در درّه‌ی سمت چپ به پایداری می‌رسد، و این درحالی است که در ساختار کلّی مسأله نقطه‌ی دیگری وجود دارد که از این نقطه‌ی تعادل بهتر است (در ارتفاع یا سطح انرژی پایین‌تری قرار دارد). به این ترتیب چنین الگوریتمی در این مثال ممکن است منجر به یک جواب بهینه‌ی موضعی شود؛ در حالی که این جواب، بهینه‌ی سراسری مسأله نیست.

[1] Local Search



شکل ۲

حال می‌خواهیم از همین ایده برای مسائل الگوریتمی استفاده کنیم. به طور خاص، می‌خواهیم چنین رویکردی را برای مسائل دشوار، (مسائل انپی–تمام) به کار گیریم و بررسی کنیم که آیا می‌توان به روشی رسید که در عمل برای پیاده‌سازی و حلّ این مسائل مناسب باشد.

۲ صورت‌بندی کلّی روش جست‌وجوی محلّی

به صورت کلّی می‌توان مسائلی که برای حل با روش جست‌وجوی محلّی مورد نظر داریم را به صورت زیر صورت‌بندی کرد:

ورودی. مجموعه Q از جواب‌های ممکن و تابع هزینه‌ی $c : Q \to \mathbb{R}$.
هدف. پیدا کردن $S \in Q$ که $c(S)$ کمینه است.

با استفاده از بیان فوق، روش جست‌وجوی محلّی به صورت زیر خواهد بود:

- گراف همسایگی جواب‌های مسأله را بسازید؛ یعنی برای هر $S \in Q$، تعدادی از جواب‌های دیگر مسأله را به عنوان همسایه‌های S در نظر بگیرید.
- در هر گام از یک جواب به جواب دیگری (که یکی از همسایگان جواب اوّلیه است و احتمالاً مقدار هزینه‌ی آن کمتر است) حرکت کنید تا نهایتاً به یک جواب کمینه (موضعی) برسید.

به عنوان مثال در همان مسأله‌ی زیردرخت فراگیر با وزن کمینه، هر زیردرخت فراگیر از گراف ورودی مسأله یک جواب (یعنی یک عضو از Q مانند S) است، و همسایه‌های هر جواب نیز زیردرخت‌هایی هستند که می‌توان با حذف یک یال از درخت اوّلیه و اضافه کردن یک یال جدید به آن‌ها رسید.

دقّت کنید که با در نظر گرفتن این قید که در هر گام از یک جواب به جواب دیگری با هزینه‌ی اکیداً کمتر حرکت کنیم، با فرض متناهی بودن تعداد جواب‌ها چنین الگوریتمی حتماً در یک نقطه‌ی کمینه‌ی موضعی به پایان می‌رسد؛ با این حال همواره در چنین الگوریتم‌هایی باید دو مسأله‌ی بسیار مهم را مورد بررسی قرار داد:

۱. *زمان اجرای الگوریتم:* چقدر طول می‌کشد تا الگوریتم به نقطه‌ی بهینه برسد؟
۲. *کیفیت جواب:* نقطه‌ی بهینه‌ی موضعی حاصل، چه میزان ممکن است از بهینه‌ی سراسری بدتر باشد؟

ضمناً دقّت کنید که در حالت کلّی لزوماً قید مذکور مبتنی بر حرکت همیشگی به سمت جواب‌های اکیداً بهتر را لحاظ نمی‌کنیم؛ یعنی ممکن است در یک گام الگوریتم اجازه دهیم که از یک پاسخ به پاسخی با هزینه‌ی بیشتر حرکت کنیم. چنین کاری می‌تواند با هدف فرار کردن از کمینه‌های محلّی و حرکت کردن به سمت کمینه‌ی سراسری مسأله صورت پذیرد.

در ادامه، الگوریتم جست‌وجوی محلّی را در مسائل مختلفی به کار خواهیم بست و کارایی آن را در هر یک از این مسائل بررسی خواهیم نمود.

۳ مسأله‌ی پوشش رأسی کمینه

تعریف ۱. *برای گراف بدون جهت $G = (V, E)$، مجموعه رئوس $V_C \subset V$ را یک پوشش رأسی[۲] گراف G گویند، هرگاه برای هر $e = uv \in E$، داشته باشیم $u \in V_C$ یا $v \in V_C$.*

به عبارت دیگر، یک پوشش رأسی برای یک گراف بدون جهت، مجموعه‌ای از رئوس آن گراف است حدّاقل یکی از دو انتهای هر یال گراف، در این مجموعه رئوس باشد. یک بیان ساده آن است که اگر همه‌ی رئوس یک پوشش رأسی را همراه با تمامی یال‌های متّصل با آن‌ها از گراف حذف شوند، هیچ یالی در گراف باقی نماند.

مسأله‌ای که می‌خواهیم در این بخش بررسی کنیم، مسأله‌ی یافتن یک پوشش رأسی با اندازه‌ی کمینه است. این مسأله یک مسأله‌ی ان‌پی–تمام است (زیرا مکمّل یک پوشش رأسی، یک زیرمجموعه‌ی مستقل از گراف خواهد بود و یافتن پوشش رأسی با اندازه‌ی کمینه، معادل با یافتن زیرمجموعه‌ی مستقل با اندازه‌ی بیشینه است که یک مسأله‌ی ان‌پی–تمام است).

برای ارائه دادن یک الگوریتم جست‌وجوی محلّی به منظور یافتن پوشش رأسی با اندازه‌ی کمینه، ابتدا باید همسایگی‌های جواب‌ها را تعریف کنیم. یک الگوریتم ساده که به صورت طبیعی به ذهن می‌رسد آن است که دو جواب احتمالی این مسأله را همسایه‌ی یکدیگر گوییم، هر گاه تنها در یک رأس اختلاف داشته باشند. به عبارت دیگر، دو مجموعه‌ی $A, B \subset V$ را همسایه گوییم، هرگاه $|A \Delta B| = 1$. در این صورت الگوریتم به این شکل خواهد بود که ابتدا همه‌ی رئوس را در نظر بگیریم (به وضوح همه‌ی رئوس، خود یک پوشش رأسی هستند) و در هر گام، در صورتی که می‌توان با حذف یک رأس از این مجموعه به یک پوشش رأسی دیگر رسید، آن رأس را حذف کنیم. شبه کد زیر نیز همین الگوریتم را توصیف می‌کند:

VertexCoverByLocalSearch(G)

```
1  S = V
2  while ∃x ∈ S, S − {x} is a vertex cover
3        S = S − {x}
4  return S
```

حال می‌توانیم عملکرد این الگوریتم را برای چند مثال مختلف بررسی کنیم:

۱. گراف تهی

این الگوریتم برای گرافی که هیچ یالی نداشته باشد به وضوح درست کار می‌کند، چرا که با شروع از $S = V$، در هر گام یکی از رئوس را حذف می‌کند تا نهایتاً به مجموعه‌ی تهی می‌رسد. واضح است که $S = \emptyset$ کوچک‌ترین پوشش رأسی گراف تهی است.

۲. گراف ستاره

گراف ستاره گرافی است با n رأس که یک رأس از درجه‌ی $n-1$ و $n-1$ رأس با درجه‌ی یک دارد. واضح است که کوچک‌ترین پوشش رأسی برای چنین گرافی، مطابق با شکل ۳آ اندازه‌ای برابر با یک دارد. با این حال، الگوریتم توصیف‌شده ممکن است جوابی مانند شکل ۳ب را به دست آورد که بسیار نابهینه است (پوشش‌های رأسی با رنگ قرمز نشان داده شده‌اند). در واقع اگر الگوریتم در اوّلین گام خود، رأس مرکزی را از مجموعه‌ی S کنار بگذارد، دیگر نمی‌تواند هیچ رأس دیگری را حذف کند و به یک پوشش رأسی با اندازه‌ی $n-1$ می‌رسد که جواب نامطلوبی است.

[۲] Vertex cover

(آ) (ب)

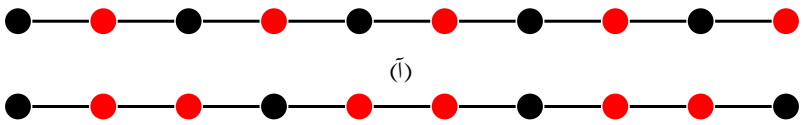
شکل ۳

۳. گراف دوبخشی

در واقع گراف دوبخشی فرم کلّی‌تری از مثال گراف ستاره است که کوچک‌ترین پوشش رأسی آن (با فرض همبندی گراف)، یکی از دو بخش گراف است که اندازه‌ی کمتری دارد. با این حال واضح است که یک پاسخ بهینه‌ی محلّی در این حالت، آن است که بخش دیگر گراف که رئوس بیشتری دارد انتخاب شود. حالت خاص آن، مثال گراف ستاره بود که یک گراف دوبخشی است، به گونه‌ای که یک بخش آن یک رأس و بخش دیگر آن $n-1$ رأس دارد و منجر به دو جواب متفاوت برای پوشش رأسی می‌شود. برای گراف دوبخشی نیز واضح است که بسته به نحوه‌ی انتخاب رئوس حذفی در هر گام، الگوریتم توصیف‌شده ممکن است به بهینه‌ی سراسری یا بهینه‌ی موضعی (غیر سراسری) برسد.

۴. گراف مسیر

مجدّداً مانند مثال‌های قبلی ممکن است الگوریتم جست‌وجوی محلّی برای یافتن پوشش رأسی با اندازه‌ی کمینه در یک نقطه‌ی بهینه‌ی موضعی گیر بیفتد و نتواند پیشرفت کند. به عنوان مثال، برای گراف مسیر، پاسخ بهینه‌ی سراسری در شکل ۴آ و یک پاسخ بهینه‌ی موضعی در شکل ۴ب نشان داده شده است. به صورت کلّی و با ادامه دادن الگویی که در این شکل‌ها نمایش داده شده است، می‌توان مشاهده کرد که پاسخ بهینه اندازه‌ای (تقریباً) برابر با $\frac{n}{2}$ دارد، در حالی که یک بهینه‌ی موضعی با اندازه‌ی (تقریبی) $\frac{2n}{3}$ نیز موجود است. الگوریتم جست‌وجوی محلّی ممکن است به هر یک از این جواب‌ها (و یا هر جواب بهینه‌ی موضعی دیگری) همگرا شود.



(آ)

(ب)

شکل ۴

بنابراین در مجموع مشاهده کردیم که عملکرد الگوریتم جست‌وجوی محلّی برای مسأله‌ی پوشش رأسی (با بیانی از الگوریتم که مدّ نظر قرار دادیم) چندان جالب نیست.

۴ مسأله‌ی فروشنده‌ی دوره‌گرد

ابتدا تعریف مسأله‌ی فروشنده‌ی دوره‌گرد[۳] (TSP) را مرور می‌کنیم:

ورودی. گراف کامل وزن‌دار G و عدد k.
خروجی. کم‌وزن‌ترین دور همیلتونی در گراف G.

در ابتدا و برای استفاده از الگوریتم جست‌وجوی محلّی، باید همسایگی دو دور همیلتونی از گراف را تعریف کنیم. واضح است که نمی‌توان دورهایی که تنها در یک یال متفاوتند را همسایه در نظر گرفت؛ چرا که هیچ دو دوری وجود ندارند که تنها در یک یال متفاوت باشند (و اگر چنین تعریفی را در نظر بگیریم، هیچ یالی همسایه‌ای نخواهد داشت؛ و طبق تعریف الگوریتم جست‌وجوی محلّی، این الگوریتم اصلاً نمی‌تواند پیشروی کند و در حالت اوّلیه‌ی خود باقی خواهد ماند). از این رو، انتخابی که برای همسایگی دو پاسخ از مسأله مدّ نظر قرار می‌دهیم، به صورت زیر است:

دورهای c و c' را همسایه گوییم، هر گاه c و c' در دو یال با هم متفاوت باشند.

به عنوان مثال، دو دور موجود در شکل‌های ۵آ و ۵ب با هم همسایه‌اند. همچنین فرض کنید که وزن هر یال را در شکل‌های ترسیمی، دقیقاً برابر با طول آن یال (با فرض این که یال‌ها به صورت خط‌های مستقیم ترسیم شوند) در نظر بگیریم. به این ترتیب جواب موجود در شکل ۵آ نسبت به شکل ۵ب بهتر است (در مجموع، طول کمتری دارد) و اگر در فرآیند اجرای الگوریتم جست‌وجوی محلّی به شکل ۵ب برسیم، یکی از انتخاب‌های موجود آن است که به همسایه‌ی بهتر آن، یعنی شکل ۵آ برویم.

(آ) (ب)

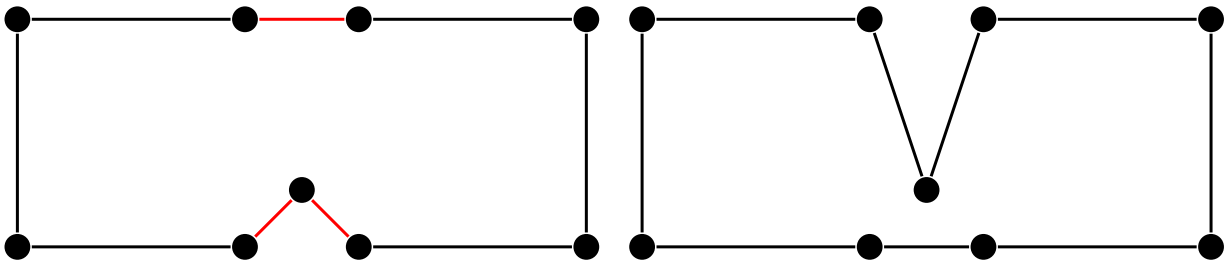
شکل ۵

تا به این‌جا ممکن است به نظر برسد که بیانی از روش جست‌وجوی محلّی که برای مسأله‌ی فروشنده‌ی دوره‌گرد ارائه کردیم، می‌تواند عملکرد مناسبی داشته باشد. با این حال می‌توان مشاهده کرد که این الگوریتم نیز می‌تواند به یک پاسخ بهینه‌ی محلّی (که با بهینه‌ی سراسری متفاوت است) همگرا شود. به عنوان مثال، دور موجود در شکل ۶آ را در نظر بگیرید. طبق تعریف همسایگی دورها، هیچ همسایه‌ای برای این دور وجود ندارد که مجموع طول آن کمتر باشد. با این حال، دور دیگری مانند شکل ۶ب موجود است که همسایه‌ی ۶آ نیست امّا طول کمتری دارد.

به این ترتیب اگر الگوریتم جست‌وجوی محلّی در یکی از گام‌های خود، به شکل ۶آ برسد، دیگر پیشروی نمی‌کند امّا پاسخ بهینه‌ی سراسری مسأله، دور دیگری است.

خوب است در این‌جا دقّت کنیم که چرا الگوریتم جست‌وجوی محلّی قادر به رسیدن به جواب بهینه‌ی سراسری نبود. یک مشاهده‌ی ساده آن است که می‌توان شکل ۶آ را با تغییر سه یال به شکل ۶ب تبدیل کرد؛ و این در حالی است که ما در ابتدا همسایگی دورها را تفاوت

[3] Traveling Salesman Problem



(آ) (ب)

شکل ۶

در دو یال در نظر گرفته بودیم. به این ترتیب به نظر می‌رسد که اگر تعریف همسایگی را گسترش دهیم، و مثلاً آن را به این صورت تعمیم دهیم که دو دور همسایه‌اند، هرگاه در (حدّاکثر) سه یال متفاوت باشند (اصطلاحاً تعریف همسایگی‌ها را از حالت 2–change به حالت 3–change تغییر دهیم)؛ ممکن است الگوریتم به پاسخ بهینه‌ی سراسری همگرا شود (یا با احتمال بیشتری چنین اتّفاقی رخ دهد). به صورت کلّی باید توجّه داشت که اگر همسایگی‌های مسائل به قدر کافی غنی نباشند، احتمال گیر کردن الگوریتم در بهینه‌های موضعی بیشتر می‌شود. با این حال از طرف دیگر اگر تعداد این همسایه‌ها را بیش از حد زیاد کنیم، زمان اجرای الگوریتم می‌تواند بیش از اندازه بالا برود. در یک حالت حدّی می‌توان همه‌ی پاسخ‌های مسأله را همسایه‌ی یک‌دیگر در نظر گرفت؛ که در آن صورت الگوریتم جست‌وجوی محلّی به نوعی تبدیل به الگوریتم بدیهی جست‌وجوی تمامی حالات می‌شود.

۵ مسأله‌ی برش بیشینه

پیش‌تر و در مبحث جریان‌های گراف، مسأله‌ی برش کمینه را بررسی کرده‌ایم. مسأله‌ی برش بیشینه از نظر تعریف بسیار مشابه با مسأله‌ی برش کمینه است، در حالی که از نظر الگوریتم‌های حل و سختی، متفاوت است. به طور خاص، می‌دانیم برای یافتن برش کمینه الگوریتمی چندجمله‌ای موجود است و این در حالی است که مسأله‌ی برش بیشینه یک مسأله‌ی ان‌پی–تمام است. در ادامه صورت‌بندی دقیق این مسأله را ارائه می‌کنیم و تلاش خواهیم کرد الگوریتمی مبتنی بر جست‌وجوی محلّی برای حلّ آن ارائه کنیم:

ورودی. گراف وزن‌دار $G = (V, E)$.
خروجی. افراز (A, B) از V به طوری که جمع وزن یال‌های بین A و B بیشینه شود.

برای ارائه‌ی یک الگوریتم جست‌وجوی محلّی، ابتدا باید همسایگی‌های جواب‌های مسأله را تعریف کنیم. برای این منظور، ساده‌ترین تعریفی که به ذهن می‌رسد را در نظر می‌گیریم:

افرازهای (A, B) و (A', B') را همسایه گوییم، هر گاه رأس $v \in V$ موجود باشد به طوری که
$$(A', B') = (A \cup \{v\}, B - \{v\}) \quad \text{یا} \quad (A', B') = (A - \{v\}, B \cup \{v\})$$

به بیان ساده‌تر، از یک افراز می‌توان با انتقال یکی از رئوس از یک طرف به طرف دیگر به افراز دیگری رفت که همسایه‌ای از افراز اوّلیه است. در این حالت برای آن که از یک پاسخ به پاسخ دیگری که ظرفیت برش متناظر با آن بیشتر باشد برسیم، باید به دنبال رأسی مانند u بگردیم که انتقال آن از یک طرف افراز به طرف دیگر، باعث افزایش ظرفیت برش شود. مثلاً اگر فرض کنید $u \in A$، برای آن که این شرط را داشته باشد، لازم است رابطه‌ی ۱ برقرار باشد:

$$\sum_{v \in A} w_{uv} > \sum_{v \in B} w_{uv} \tag{۱}$$

این یعنی مجموع وزن یال‌های متّصل به u که از A خارج می‌شوند، کمتر باشد از مجموع وزن یال‌های متّصل به u که داخل A قرار می‌گیرند. به این ترتیب با تغییر محلّ u از A به B، آن دسته از یال‌های متّصل به u که پیش‌تر از برش عبور می‌کردند دیگر از برش عبور نمی‌کنند (زیر مبدأ آن‌ها، یعنی u، و مقصد آن‌ها، هر دو در B قرار گرفته‌اند)؛ و یال‌هایی که یک سر آن‌ها u و سر دیگر آن‌ها در A قرار داشت، هم‌اکنون از برش عبور می‌کنند. به این ترتیب طبق رابطه‌ی ۱، انتقال u از A به B باعث افزایش ظرفیت برش می‌شود.

به این ترتیب، الگوریتم جست‌وجوی محلّی به صورت زیر خواهد بود:

```
MaxCutLocal(G, w)
1  (A, B) = random cut
2  while there exists an improving node v
3      if v ∉ A
4          A = A ∪ {v}
5          B = B − {v}
6      else
7          B = B ∪ {v}
8          A = A − {v}
9  return (A, B)
```

حال برای این مسأله نشان خواهیم داد که الگوریتم فوق از نظر کیفیت، *نسبتاً* مناسب است. به عبارت دقیق‌تر، ظرفیت برش حاصل از این الگوریتم حدّاقل به اندازه‌ی نصف ظرفیت برش بیشینه خواهد بود.

برای این منظور، با تعریف نمادگذاری $w(A, B)$ به عنوان وزن برش (A, B) مطابق با رابطه‌ی ۲، می‌توانیم قضیه‌ی ۱ را ثابت کنیم:

$$w(A, B) = \sum_{u \in A\,,\, v \in B} w_{uv} \tag{۲}$$

قضیه ۱. *فرض کنید (A, B) برشی باشد که در تعریف مسأله‌ی برش بیشینه، یک پاسخ بهینه‌ی موضعی (ناشی از الگوریتم جست‌وجوی محلّی) است. همچنین فرض کنید برش (A^*, B^*) پاسخ بهینه‌ی سراسری مسأله باشد. در این صورت رابطه‌ی زیر برقرار است:*

$$w(A, B) \geq \frac{1}{2} \sum_{e \in E} w_e \geq \frac{1}{2} w(A^*, B^*) \tag{۳}$$

اثبات. چون (A, B) یک پاسخ بهینه‌ی موضعی است، هیچ رأسی در گراف وجود ندارد که اگر از A به B منتقل شود (و یا برعکس)، ظرفیت برش حاصل بیشتر شود. به عبارتی، رابطه‌ی ۱ برای هیچ رأسی از گراف برقرار نخواهد بود. به این ترتیب خواهیم داشت:

$$\forall\, u \in A : \sum_{v \in A} w_{uv} \leq \sum_{v \in B} w_{uv} \tag{۴}$$

حال اگر طرفین رابطه‌ی ۴ را برای هر $u \in A$ جمع بزنیم، (با دقّت به این نکته که وزن هر یال مثل uv که هر دو سر آن در A هستند، دو بار جمع زده خواهد شد)، خواهیم داشت:

$$2 \sum_{\{u,v\} \subseteq A} w_{uv} \leq \sum_{u \in A\,,\, v \in B} w_{uv} = w(A, B) \tag{۵}$$

که تساوی آخر در رابطه‌ی فوق، نتیجه‌ی جای‌گذاری تعریف ۲ است. همچنین به طریق مشابه، اگر همه‌ی این معادلات را برای رئوس موجود در B بازنویسی کنیم، خواهیم داشت:

$$2 \sum_{\{u,v\} \subseteq B} w_{uv} \leq \sum_{u \in A\,,\, v \in B} w_{uv} = w(A, B) \tag{۶}$$

حال با در نظر گرفتن تعریف ۲ و دو نامساوی ۴ و ۵ می‌توان نوشت:

$$\begin{aligned} \sum_{e \in E} w_e &= \sum_{\{u,v\} \subseteq A} w_{uv} + \sum_{u \in A\,,\, v \in B} w_{uv} + \sum_{\{u,v\} \subseteq B} w_{uv} \\ &\leq \frac{1}{2} w(A, B) + w(A, B) + \frac{1}{2} w(A, B) \\ &= 2w(A, B) \end{aligned} \tag{۷}$$

همچنین واضح است که مقدار وزن برش بیشینه، حدّاکثر برابر است با مجموع وزن همه‌ی یال‌های گراف. به این ترتیب معادله‌ی ۷ عملاً اثبات را به پایان رسانده است و حکم نتیجه شده است:

$$w(A, B) \geq \frac{1}{2} \sum_{e \in E} w_e \geq \frac{1}{2} w(A^*, B^*) \tag{۸}$$

□

تا به این‌جا برای مسأله‌ی برش بیشینه نشان داده‌ایم که یک الگوریتم جست‌وجوی محلّی می‌تواند پاسخی با کیفیت نسبتاً مناسب را تولید کند. با این وجود در مورد زمان اجرای آن، اطّلاع دقیقی نداریم. به عبارتی با توجّه به متناهی بودن تعداد کلّ پاسخ‌های ممکن، قطعاً این

الگوریتم به پایان می‌رسد؛ با این حال تضمینی برای زمان به پایان رسیدن آن وجود ندارد. در این حالت، یک تکنیک متداول در مسأله‌های جست‌وجوی محلّی وجود دارد که می‌تواند با اندکی کاهش کیفیت جواب، تضمین مناسبی روی زمان اجرای الگوریتم نیز اعمال کند:

الگوریتم پیشروی با ارتقاء قابل توجّه[۴]. الگوریتم جست‌وجوی محلّی را تنها در صورتی ادامه دهید که یک گام پیشروی، حدّاقل به اندازه‌ی ثابت و معلومی مثل Δ، پاسخ موجود را بهبود بخشد.

به عبارت دیگر، ممکن است در الگوریتم‌های جست‌وجوی محلّی بتوان با انجام پیمایش یک گام، مثلاً مقدار جواب مورد نظر را به اندازه‌ی یک درصد بهبود داد. با این حال ممکن است این مقدار کافی نباشد، و در الگوریتم فوق قرارداد کنیم که اگر در یک گام از جست‌وجوی محلّی، امکان پیشرفت حدّاقل ده‌درصدی وجود نداشت، اعلام می‌کنیم که الگوریتم به اتمام رسیده است و به یک جواب بهینه‌ی محلّی رسیده‌ایم. البته طبیعتاً مقدار این حدّ آستانه بسته به کاربرد و شرایط مسأله می‌تواند متفاوت در نظر گرفته شود (و شاید در مسأله‌ای، پیشرفت یک‌درصدی هم بسیار مناسب و قابل توجّه باشد).

حال، این اصلاح را در الگوریتم جست‌وجوی محلّی برای یافتن برش بیشینه به صورت زیر اعمال می‌کنیم:
تنها زمانی یک گام در الگوریتم پیشروی کنید که در اثر این کار، مقدار ظرفیت برش حدّاقل به اندازه‌ی $\frac{2\epsilon}{n}w(A,B)$ *افزایش یابد.*

ادعا ۱. *الگوریتم پیشروی با ارتقاء قابل‌توجّه برای حلّ مسأله‌ی برش بیشینه، برشی مانند* (A,B) *را برمی‌گرداند، به گونه‌ای که*

$$(2+\epsilon)w(A,B) \geq w(A^*,B^*) \tag{۹}$$

اثبات. کافی است در اثبات قضیه‌ی ۱، در هر یک از نامساوی‌های مطرح‌شده، $w(A,B)$ را با $\frac{2\epsilon}{n}w(A,B)$ جایگزین کنید. □

دقّت کنید که در این حالت، ظرفیت برش بیشینه حدّاکثر $(2+\epsilon)$ برابر ظرفیت برش حاصل از الگوریتم جست‌وجوی محلّی است؛ با این حال الگوریتم از مزیت بسیار مهمّی برخوردار شده است و آن، زمان اجرای چندجمله‌ای است:

ادعا ۲. *زمان اجرای الگوریتم پیشروی با ارتقاء قابل‌توجّه برای یافتن برش بیشینه از مرتبه‌ی* $O\left(\frac{n}{\epsilon}\lg W\right)$ *است که* $W=\sum_{e\in E}w_e$.

اثبات. طبق تعریف، در هر گام از اجرای الگوریتم، مقدار جواب حدّاقل $1+\epsilon/n$ برابر می‌شود (به صورت دقیق‌تر، حدّاقل $1+2\epsilon/n$ برابر می‌شود که در این‌جا یک کران پایین‌تر را در نظر گرفته‌ایم). به این ترتیب پس از n/ϵ بار اجرای الگوریتم، مقدار پاسخ حدّاقل $(1+\epsilon/n)^{n/\epsilon}$ برابر شده است که با توجّه به برقراری نامساوی $(1+1/x)^x \geq 2$ برای هر $x \geq 1$ (و با جای‌گذاری $x=n/\epsilon$)، نتیجه می‌شود که پس از n/ϵ بار اجرای الگوریتم، مقدار وزن برش حاصل حدّاقل دو برابر می‌شود.

حال کافی است دقّت کنیم که مقدار $W=\sum_{e\in E}w_e$ یک کران بالا برای ظرفیت برش بیشینه بود؛ و چون در هر n/ϵ گام از الگوریتم پاسخ حدّاقل دو برابر بهتر می‌شود، با طی کردن حدّاکثر $(n/\epsilon)\lg W$ گام از الگوریتم، قطعاً به نقطه‌ی بهینه‌ی موضعی رسیده‌ایم و اجرای الگوریتم به پایان رسیده است. به این ترتیب زمان اجرای الگوریتم پیشروی با ارتقاء قابل‌توجّه برای یافتن برش بیشینه از مرتبه‌ی $O\left((n/\epsilon)\lg W\right)$ می‌باشد. □

به این ترتیب به الگوریتم چندجمله‌ای ساده‌ای برای تقریب‌زدن مسأله‌ی برش بیشینه دست یافته‌ایم که پاسخ این مسأله را با ضریب 0.5 تقریب می‌زند. الگوریتم بهتری برای این کار وجود دارد که این تقریب را با ضریب حدودی 0.878 انجام می‌دهد و جالب است بدانید که ثابت شده است با فرض $\mathbf{P}\neq\mathbf{NP}$، هیچ الگوریتم تقریبی برای این مسأله وجود ندارد که آن را با ضریبی بهتر از 0.942 تقریب بزند. با این حال یک نکته‌ی جالب دیگر نیز این است که مشخّص نیست یافتن تقریباتی که در حدّ فاصل این دو مقدار باشند، در کلاس $\mathbf{P}$ است یا $\mathbf{NP}$. به عنوان مثال، این مسأله که آیا امکان دارد تقریبی برای مسأله‌ی برش بیشینه ارائه دهیم که همواره پاسخ درست را به اندازه‌ی حدّاقل 90 درصد تقریب بزند، مشخّص نیست که در کلاس $\mathbf{P}$ قرار دارد و یا $\mathbf{NP}$.

[۴] Big-Improvement-Flip algorithm

در ادامه می‌خواهیم بررسی کنیم که آیا می‌توان با تغییر مفهوم همسایگی برای پاسخ‌های این مسأله، به الگوریتم جست‌وجوی محلّی بهتری دست یافت. به طور خاص، می‌توان همسایگی‌ها را این طور تعریف کرد که به جای آن که تنها با تغییر یک رأس گراف در برش مربوطه به همسایه‌های آن برسیم (1–flip neighborhood)، با تغییر حدّاکثر k رأس این کار را انجام دهیم (k–flip neighborhood). این یعنی دو برش (A, B) و (A', B') را همسایه در نظر بگیریم، هرگاه این دو برش حدّاکثر در k رأس متفاوت باشند (یا با انتقال حدّاکثر k رأس از یک طرف برش به طرف دیگر آن، بتوان (A, B) را به (A', B') تبدیل کرد). به وضوح چنین کاری منجر به افزایش تعداد همسایه‌های هر برش می‌شود. همان‌طور که در پایان بخش مربوط به مسأله‌ی فروشنده‌ی دوره‌گرد (قسمت ۴) نیز به این موضوع اشاره کردیم، چنین کاری *می‌تواند* احتمال رسیدن الگوریتم به یک جواب بهتر را افزایش دهد، امّا به صورت کلّی زمان اجرای آن را نیز می‌تواند زیاد کند که اتّفاق نامطلوبی است.

یک روش پیشنهادی برای تغییر مفهوم همسایگی در این مسأله، الگوریتم همسایگی KL[5] است که در عمل، کارایی مناسبی از خود نشان می‌دهد. در این الگوریتم، برای یافتن همسایگی یک برش مثل (A, B)، $n - 1$ گام زیر طی می‌شود:

- گام اوّل: رأسی از گراف را پیدا کنید که با تغییر محلّ آن در برش (A, B)، بهترین برش جدید ممکن ایجاد شود (منظور از تغییر محلّ یک رأس، یعنی اگر در A باشد آن را به B منتقل کنیم و برعکس، اگر در B باشد آن را به A منتقل نماییم). در این صورت این رأس را علامت‌دار کنید و محلّ آن را تغییر دهید؛ و برش جدید حاصل را (A_1, B_1) بنامید.
- گام iاُم: رأسی از گراف را بیابید به گونه‌ای اوّلاً علامت‌دار نباشد و ثانیاً با تغییر مکان آن در برش (A_{i-1}, B_{i-1})، بهترین برش جدید ایجاد شود. این رأس را علامت‌دار کنید و محلّ آن را تغییر دهید؛ و برش جدید حاصل را (A_i, B_i) بنامید.

پس از $n - 1$ بار اجرای گام‌های فوق، $n - 1$ برش جدید $(A_1, B_1), \ldots, (A_{n-1}, B_{n-1})$ ایجاد می‌شوند. در این الگویتم، این $n - 1$ برش را همسایه‌های برش (A, B) در نظر می‌گیریم. دقّت کنید که چون مطابق با گام‌های فوق، در هر مرحله تنها مجاز به تغییر محلّ رئوسی هستیم که قبل‌تر دچار تغییر محل نشده‌اند (و علامت‌دار نیستند)، اگر یک گام دیگر این فرآیند را تکرار کنیم خواهیم داشت $(A_n, B_n) = (B, A)$، زیرا پس از گذشت n گام همه‌ی رئوس برش اوّلیه جای خود را عوض کرده‌اند و عملاً دوباره به همان برش رسیده‌ایم.

به این ترتیب می‌توان همسایگی هر برش دلخواه (A, B) را تشکیل داد و در هر گام، مطابق با الگوریتم جست‌وجوی محلّی همسایه‌ای از برش فعلی را انتخاب کرد که ظرفیت آن بیشتر باشد.

این الگوریتم عملکرد مناسبی از خود به نمایش می‌گذارد و در عمل برای استفاده قابل توجّه است؛ با این حال به صورت تئوری توجیه مناسبی از کیفیت عملکرد آن وجود ندارد.

[5] Kernighan–Lin–neighborhood algorithm

۶ اصلاحاتی بر الگوریتم ساده‌ی جست‌وجوی محلّی

تا به این‌جا الگوریتم جست‌وجوی محلّی را به این صورت استفاده می‌کردیم که با تعریف یک همسایگی، در هر گام در صورتی پیشروی می‌کنیم که همسایه‌ای از حالت فعلی موجود باشد که به جواب بهینه نزدیک‌تر باشد. مشاهده کردیم که این روش همواره در پاسخ‌های بهینه‌ی موضعی به دام می‌افتد و امکان فرار از آن‌ها را ندارد. می‌خواهیم الگوریتم‌های جدیدی ارائه دهیم که این امکان را در اختیار قرار دهند که در فرآیند جست‌وجوی محلّی، بتوانیم از یک نقطه‌ی بهینه‌ی موضعی نیز خارج شویم تا این امید وجود داشته باشد که با این کار، به یک نقطه‌ی بهینه‌ی موضعی بهتر (یا یک نقطه‌ی بهینه‌ی سراسری) برسیم. برای این منظور، دو الگوریتم جدید معرّفی خواهیم کرد.

۱.۶ الگوریتم متروپلیس

همان‌طور که نسخه‌ی ابتدایی الگوریتم جست‌وجوی محلّی نیز از یک شهود فیزیکی برخوردار بود؛ برای بهبود آن نیز از شهودهای فیزیکی بهره می‌بریم. برای این منظور، تابع گیبس–بولتزمن[۶] را معرّفی می‌کنیم که تابعی برای توصیف احتمال قرار گیری یک سیستم فیزیکی در یک شرایط معیّن است و به صورت زیر تعریف می‌شود:

معادله‌ی گیبس–بولتزمن. احتمال آن که یک سیستم فیزیکی در وضعیتی با سطح انرژی E قرار داشته باشد، متناسب است با $e^{-E/(kT)}$، که در آن $T > 0$ دمای مطلق سیستم و k ضریبی ثابت است.

با توجّه به ضابطه‌ی این تابع، می‌توان حقایق زیر را نیز در مورد سیستم‌های فیزیکی بیان کرد:

- برای هر دمای $T > 0$، تابع گیبس–بولتزمن تابعی اکیداً نزولی از E است.
- یک سیستم فیزیکی بیشتر محتمل است که در یک سطح انرژی پایین باشد (در مقایسه با یک سطح انرژی بالا)، و این احتمال تابعی از T است:
  - اگر T بزرگ باشد، سطوح انرژی بالا و پایین تقریباً از احتمال یکسانی برخوردارند.
  - اگر T کوچک باشد، سطوح انرژی پایین بسیار محتمل‌ترند.

در ادامه الگوریتمی برای گذار بین حالات مختلف یک سیستم (یا معادلاً، گذار بین پاسخ‌های مختلف یک مسأله) بر مبنای سطوح انرژی آن‌ها (یا معادلاً، مقدار یک تابع هزینه روی هر یک از پاسخ‌های مسأله) ارائه می‌دهیم:

الگوریتم متروپلیس[۷]. برای گذار بین حالات مختلف یک سیستم، گام‌های زیر را طی کنید:

- با فرض دمای ثابت $T > 0$، یک حالت فعلی مثل S را در نظر بگیرید (که یا از گذار از حالت قبلی حاصل شده، و یا یک حالت اوّلیه برای شروع الگوریتم است).
- به صورت تصادفی، حالتی مثل S' را در نظر بگیرید که $S' \in \mathcal{N}(S)$ و منظور از $\mathcal{N}(S)$، مجموعه‌ی همسایه‌های S می‌باشد.
- اگر $E(S') \leq E(S)$، حالت فعلی را به صورت قطعی از S به S' تغییر دهید. در غیر این صورت، حالت فعلی را با احتمال $e^{-\Delta E/(kT)}$ از S به S' تغییر دهید که در آن، $\Delta E = E(S') - E(S) > 0$.

قضیه‌ای که در ادامه می‌آید، کارایی این الگوریتم را در مدل‌سازی سیستم‌ها و حلّ برخی مسائل روشن می‌کند.

[6] Gibbs–Boltzmann function
[7] Metropolis algorithm

قضیه ۲. *فرض کنید $f_S(t)$ نسبتی از t گام اوّل الگوریتم متروپلیس باشد که در آن، حالت فعلی سیستم برابر با S بوده است. در این صورت، (با در نظر گرفتن برخی ملاحظات تکنیکی) رابطه‌ی ریاضی زیر با احتمال ۱ برقرار خواهد بود:*

$$\lim_{t\to\infty} f_S(t) = \frac{1}{Z} e^{-E(S)/(kT)} \tag{۱۰}$$

که در آن

$$Z = \sum_{S'\in\mathcal{N}(S)} e^{-E(S')/(kT)} \tag{۱۱}$$

مطابق با قضیه‌ی ۲، به صورت شهودی مشاهده می‌شود که نتیجه‌ی شبیه‌سازی حاصل از الگوریتم متروپلیس، با در نظر گرفتن معادله‌ی گیبس–بولتزمن، زمان درستی را در هر یک از حالت سیستم سپری می‌کند (یعنی نسبت زمان حضور در هر حالت، به مقدار احتمال حضور در آن حالت میل می‌کند).

دقّت کنید که این الگوریتم، بر خلاف الگوریتم جست‌وجوی موضعی ساده که پیش‌تر بررسی کرده بودیم، به صورت صریح یک پاسخ را در خروجی اعلام نمی‌کند؛ بلکه یک شبیه‌سازی با زمان نامتناهی (که یک فرآیند مارکوف است) ایجاد می‌کند که همواره بین تعدادی حالت در حال جابه‌جایی است. آن‌چه می‌تواند منجر به یافتن یک پاسخ مناسب برای یک مسأله (معادل با یافتن حالت با کمترین انرژی) شود، آن است که این فرآیند را مشاهده کنیم تا حالتی را بیابیم که فرآیند مربوطه، بیشترین سهم زمانی را در آن سپری می‌کند. مطابق با معادله‌ی گیبس–بولتزمن، این حالت دارای انرژی کمینه خواهد بود.

با در نظر گرفتن جمیع این نکات، می‌توان مشاهده کرد که الگوریتم متروپلیس می‌تواند در برخی موارد، ناکارآمدی‌های نسخه‌ی ساده‌ی الگوریتم جست‌وجوی محلّی را بهبود بخشد؛ و در برخی موارد، عملکرد مناسب آن الگوریتم را نیز دچار مشکل سازد. برای فهم بهتر این موضوع، مسأله‌ی پوشش رأسی با اندازه‌ی کمینه (بخش ۴) را در نظر بگیرید. یک حالت از آن مسأله، یافتن پوشش رأسی با اندازه‌ی کمینه برای گراف ستاره بود. دیدیم که در این مسأله، اگر الگوریتم جست‌وجوی محلّی رأس مرکزی گراف را حذف کند دیگر نمی‌تواند به پاسخ بهینه دست یابد و در یک نقطه‌ی بهینه‌ی موضعی (که بسیار بدتر از بهینه‌ی سراسری است) گیر می‌افتد. حال فرض کنید که در این مسأله از الگوریتم متروپلیس استفاده کنیم. در این حالت حتّی اگر در یکی از مراحل رأس مرکزی گراف حذف شود؛ احتمال مثبتی وجود دارد که این رأس دوباره به گراف اضافه شود (معادل با احتمال حرکت از یک حالت کم‌انرژی به یک حالت پرانرژی در الگوریتم متروپلیس) و دوباره این فرصت در اختیار الگوریتم قرار گیرد که به سمت پاسخ‌های بهتر حرکت کند.

از طرف دیگر، این بار مثال گراف تهی را در مسأله‌ی یافتن پوشش رأسی با اندازه‌ی کمینه در نظر بگیرید. دیدیم که برای این مسأله، روش جست‌وجوی محلّی ساده به راحتی به جواب صحیح همگرا می‌شد. حال فرض کنید که از الگوریتم متروپلیس استفاده کنیم. به وضوح حرکت از یک حالت به حالت با انرژی کم‌تر یا بیشتر در این الگوریتم، معادل با آن است که در هر گام، یک پوشش رأسی با اندازه‌ی k داشته باشیم و یک رأس به آن اضافه کنیم یا از آن حذف کنیم. حال فرض کنید مثلا گراف مورد نظر ۱۰۰ رأس داشته باشد. در این صورت هنگامی که تنها ۵ رأس آن در پوشش رأسی باقی مانده باشند و ۹۵ رأس حذف شده باشند، در الگوریتم متروپلیس با احتمال 0.95، حالت $S' \in \mathcal{N}(S)$ از میان حالاتی انتخاب می‌شود که تعداد رئوس موجود در آن‌ها برابر با ۶ است، و با احتمال قابل توجّهی ممکن است در این گام، یک رأس به پوشش رأسی فعلی افزوده شود و از نقطه‌ی بهینه‌ی سراسری دورتر شویم. به عبارتی، در این مسأله هر قدر به حالت بهینه نزدیک و نزدیک‌تر شویم، پیشروی مضاعف سخت‌تر می‌شود و احتمال دورشدن از آن افزایش می‌یابد و باید زمان بسیار زیادی صبر کرد تا بتوان نتیجه‌گیری مناسب را از این الگوریتم انجام داد.

به این ترتیب به نظر می‌رسد که این الگوریتم نیز نیاز به اصلاحاتی برای بهتر شدن دارد. در ادامه، الگوریتم دیگری را (که نسخه‌ی بهبودیافته‌ی الگوریتم متروپلیس است) بررسی خواهیم کرد.

۲.۶ الگوریتم تبرید شبیه‌سازی‌شده

مشاهده کردیم که در صورت الگوریتم متروپلیس، پارامتر T وجود دارد که تا به این‌جا آن را مقداری ثابت در نظر گرفته بودیم و اثر آن را مورد توجّه چندانی قرار ندادیم. همچنین دیدیم که بزرگ‌شدن مقدار T، تقریباً همه‌ی حالات سیستم را به سمت هم احتمال‌شدن سوق می‌دهد و به نظر می‌رسد که با در نظر گرفتن رابطه‌ی ۱۱، منطقی باشد که T را کوچک اختیار کنیم تا احتمال حضور در حالت کم‌انرژی زیاد شود و به هدف خود از این شبیه‌سازی برسیم. با این حال، اتّفاق نامطلوبی که در این صورت می‌تواند رخ بدهد، مشابه با چیزی است که پیش‌تر در ارتباط با مسأله‌ی پوشش رأسی کمینه در گراف تهی بیان کردیم. به عبارت دقیق‌تر، ممکن است حدّ موجود در معادله‌ی ۱۱ به ازای tهای بسیار بزرگ به سمت همگرایی حرکت کند، و برای یافتن حالات کم‌انرژی نیاز به صرف زمان بسیار زیادی داشته باشیم. از طرف دیگر به نظر می‌رسد که با انتخاب مقادیر بزرگ‌تر برای T، سرعت همگرایی رابطه‌ی ۱۱ افزایش یابد امّا احتمال حضور در حالات مختلف به هم نزدیک‌تر شوند.

برای کنترل این موضوع، ابتدا یک شهود فیزیکی دیگر را در نظر می‌گیریم:

- اگر جسم جامدی را به دمای بسیار بالا برسانیم، انتظار نداریم که بلور کریستالی منظّمی تشکیل دهد.
- اگر جسم مذابی را به صورت بسیار ناگهانی سرد کنیم، بازهم انتظار نداریم که بلور کریستالی چندان منظّمی تشکیل شود.
- اگر جسم مذاب به صورت تدریجی سرد شود، می‌تواند به مرور به نقطه‌ی تعادلی نزدیک شود که تشکیل ساختار بلوری دقیق و مناسبی بدهد. این فرآیند را *تبرید*[8] گویند.

با ایده گرفتن از همین شهود فیزیکی، الگوریتم *تبرید شبیه‌سازی‌شده*[9] معرّفی می‌شود که به نوعی همان الگوریتم متروپلیس است، با این تفاوت که ابتدا با مقدار نسبتاً بزرگی از T شروع می‌شود و با طی شدن گام‌های متوالی، به مرور مقدار T آن کاهش می‌یابد. به این ترتیب حدّ میانه‌ای بین سرعت همگرایی رابطه‌ی ۱۱ و تمایز احتمالاتی حالت با انرژی کمتر نسبت به حالت با انرژی بیشتر برقرار می‌شود. به وضوح این که انتخاب مقدار اوّلیه و چگونگی تغییر T متناسب با هر مسأله به صورت مناسب انجام شود، خود یک هنر طرّاحی الگوریتم و حلّ مسأله خواهد بود.

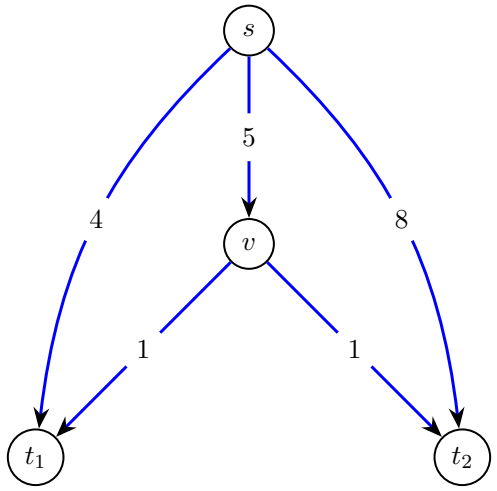
[8] annealing
[9] Simulated annealing algorithm

۷ مسأله‌ی مسیریابی چندتایی و تعادل نَش (اختیاری)

در این بخش به بررسی یک مسأله‌ی جالب به کمک جست‌وجوی موضعی خواهیم پرداخت. این مسأله، از جهاتی، با مثال‌های قبلی بررسی‌شده تفاوت دارد و مسأله‌ای از حوزه‌ی نظریه‌ی بازی است.

مسأله‌ی مسیریابی چندتایی. گراف جهت‌دار $G = (V, E)$ به همراه وزن (هزینه)های $c_e \geq 0$ برای هر یال داده شده است. به علاوه، یک رأس مبدأ، s، و k رأس مقصد $t_1, t_2, \ldots, t_k$ متناظر با k بازیگر (عامل[10]) مسأله داده شده است. هر یک از بازیگران مسأله، می‌خواهد مسیری مثل P_j از s به مقصد خود، t_j بیابد.
قید مهم در این مسأله، قید تسهیم عادلانه[11] است، یعنی اگر تعداد x بازیگر از یال e در مسیر خود استفاده کنند، هزینه‌ی آن یال بین آن‌ها تقسیم می‌شود و هر کدام هزینه‌ای به اندازه‌ی c_e/x برای آن یال می‌پردازند.

به عنوان مثال، فرض کنید تنها دو بازیگر در بازی موجود باشند ($k = 2$)، و گراف مسأله مطابق با شکل ۷ داده شده باشد.



شکل ۷

در این صورت، هر یک از دو بازیگر می‌تواند یکی از دو مسیر ممکن برای خود را انتخاب کند که آن‌ها را (با در نظر گرفتن ساختار موجود در شکل ۷) «مسیر بیرونی» و «مسیر داخلی» می‌نامیم. منظور از مسیر داخلی برای هر یک از دو بازیگر، مسیری است که از یال با وزن ۵ می‌گذرد و منظور از مسیر بیرونی، مسیری است که مستقیماً از s به مقصد هر یک از دو بازیگر مسأله وصل شده است.

در این شرایط، هر بازیگر دو انتخاب برای رسیدن از s به مقصد خود، t_j دارد که در مجموع ۴ استراتژی متفاوت را رقم می‌زند. با در نظر گرفتن اصل تسهیم عادلانه، هزینه‌های پرداختی هر یک از دو بازیگر در حالات مختلف در جدول ۱ قابل مشاهده است.

مسیر بازیگر ۱	مسیر بازیگر ۲	هزینه‌ی بازیگر ۱	هزینه‌ی بازیگر ۲
مسیر بیرونی	مسیر بیرونی	4	8
مسیر بیرونی	مسیر داخلی	4	$5+1$
مسیر داخلی	مسیر بیرونی	$5+1$	8
مسیر داخلی	مسیر داخلی	$5/2+1$	$5/2+1$

جدول ۱

[10] agent
[11] fair share

حال فرض کنید که دینامیک حاکم بر این مسأله به گونه‌ای باشد بازیگران مسأله همواره در صدد کمینه‌کردن هزینه‌ی خود باشند. به این ترتیب، در هر گام، یکی از بازیگران ممکن است مسیر فعلی خود را تغییر دهد و مسیر جدیدی با هزینه‌ی کمتر را انتخاب کند. این رویکرد مشابه با روش جست‌وجوی محلّی است؛ امّا یک نکته‌ی مهم در این مسأله آن است که رفتار بازیگران بر روی یک‌دیگر اثر می‌گذارد و مسیری که یکی از آن‌ها انتخاب کرده است بر هزینه‌های مسیرهای سایرین و در نتیجه بر تصمیم‌گیری‌های آن‌ها اثر می‌گذارد.

به عبارتی، در این ساختار هر بازیگر در هر گام تلاش برای بهبود شرایط خود می‌کند که از این جهت، شبیه به جست‌وجوی محلّی است. با این وجود، ممکن است اقدام یک بازیگر برای بهبود شرایط خود، منجر به افزایش هزینه‌ی یک بازیگر دیگر شود و با توجّه به این که هدف‌های بازیگران متفاوت است، هر کدام تعبیر متفاوتی از بهینگی دارند که ممکن است این تعاریف با هم در تضاد باشند (یعنی انتخاب‌های یکی در جهت بهینگی خودش، منجر به دور شدن دیگری از بهینگی شود). به این ترتیب بر خلاف حالات عادی مسائل جست‌وجوی محلّی، به نظر می‌رسد که ممکن است در چنین مسأله‌ای اصلاً الگوریتم به حالت پایدار نرسد و همواره ادامه یابد.

به عنوان مثال، می‌توانیم ساختار شکل ۷ را بررسی کنیم. اگر فرض کنیم هر دو بازیگر در ابتدا مسیرهای بیرونی را انتخاب کرده‌اند؛ در اوّلین گام بازیگر شماره ۱ مسیر خود را تغییر نمی‌دهد، چرا که مسیر بیرونی برای او 4 واحد، و مسیر داخلی برای او 6 واحد هزینه دارد. در گام بعد، بازیگر شماره ۲ مسیر خود را تغییر می‌دهد، چرا که هزینه‌ی مسیرهای بیرونی و داخلی وی به ترتیب 8 و 6 واحد هستند. در گام بعدی و پس از آن که بازیگر شماره ۲ مسیر داخلی را انتخاب کرد، بازیگر شماره ۱ مشاهده می‌کند که اگر مسیرش را تغییر دهد، (به دلیل اصل تسهیم عادلانه) می‌تواند با هزینه‌ی 3.5 از طریق مسیر داخلی به مقصد خود برسد. بنابراین این کار را انجام می‌دهد و سیستم در این وضعیت به تعادل می‌رسد.

تعادل نَش[۱۲]. در صورتی که چنین سیستمی به تعادل برسد، آن شرایط را یک *تعادل نَش* گویند. به عبارت دقیق‌تر، تعادل نش وضعیتی است که در آن، هیچ یک از بازیگران مسأله انگیزه‌ای برای تغییر استراتژی خود نداشته باشند.

بهینه‌ی اجتماعی[۱۳]. تا به این‌جا بهینگی یک حل برای مسأله را بر مبنای تعاریف هر یک از بازیگران در نظر گرفتیم، و تعادل نش نیز وضعیتی بود که هر یک از بازیگران به نوعی در وضعیت بهینه (موضعی) خود باشند و انگیزه‌ای برای تغییر وضعیت نداشته باشند (شبیه به یک بهینه‌ی موضعی در روش‌های جست‌وجوی محلّی).

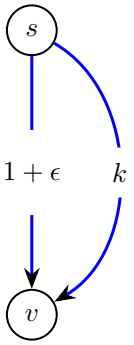
با این حال می‌توان یک تعریف عمومی از بهینگی نیز ارائه داد، به گونه‌ای که وضعیت بهینه را وضعیتی در نظر بگیریم که مجموع هزینه‌های همه‌ی بازیگران در آن کمینه باشد. چنین شرایطی را یک *بهینه‌ی اجتماعی* گوییم.

در حالت کلّی یک مشاهده‌ی مهم آن است که تعادل نش یکتا نیست، و حتّی در صورت یکتا بودن نیز لزوماً با حالت بهینه‌ی اجتماعی یکسان نیست. در واقع ممکن است تعادل نش از نظر اجتماعی بهینه نباشد.

مثال‌های موجود در شکل‌های ۸ و ۹ شواهدی بر این ادّعا هستند. به عنوان مثال فرض کنید k بازیگر در مسأله‌ی متناظر با شکل ۸ موجود باشند. در این صورت یک تعادل نش آن است که همه‌ی k بازیگر از یالی که وزن k دارد عبور کنند، چرا که با تقسیم هزینه‌ی آن بین بازیگران، هر یک هزینه‌ی واحد پرداخت خواهند کرد. همچنین دقّت کنید که هیچ یک از بازیگران انگیزه‌ای برای تغییر مسیر خود ندارد، زیرا اگر مسیر خود را تغییر دهد لازم است هزینه‌ای برابر با $1+\epsilon$ پرداخت کند که از هزینه‌ی فعلی پرداختی‌اش بیشتر است. البته به راحتی مشاهده می‌شود که اگر تمامی بازیگران از مسیر با هزینه‌ی $1+\epsilon$ عبور کنند، تعادل نش دیگری نیز وجود خواهد داشت که بهینه‌ی اجتماعی نیز هست، چرا که مجموع هزینه‌های پرداختی برابر با $1+\epsilon$ خواهد بود که از هر حالت دیگری کمتر است. به این ترتیب در این مثال دو تعادل نش وجود دارد که یکی از آن‌ها معادل با بهینه‌ی اجتماعی است و دیگری چنین نیست.

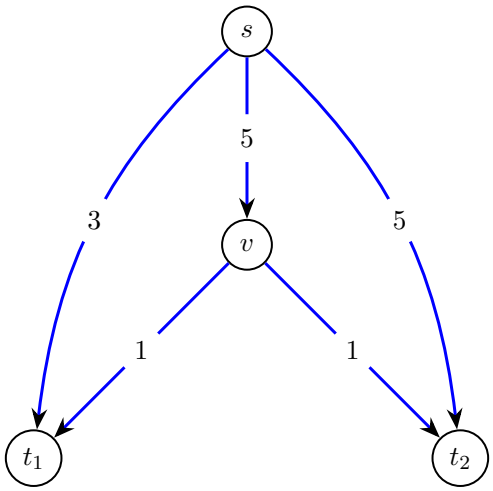
حال مثال شکل ۹ را در نظر بگیرید. در این مثال وضعیت بهینه‌ی اجتماعی آن است که هر دو بازیگر از مسیر داخلی عبور کنند، چرا که در مجموع باید 7 واحد هزینه بپردازند و این کمتر از مجموع هزینه‌های آن‌ها در هر حالت دیگری است. با این حال اگر در چنین وضعیتی

[12] Nash equilibrium
[13] social optimum



شکل ۸

باشیم، بازیگر شماره ۱ انگیزه دارد که مسیر خود را تغییر دهد، چرا که با این کار هزینه‌ی خود را از 3.5 واحد به 3 واحد کاهش خواهد داد. پس از انجام این کار، هزینه‌ی بازیگر شماره ۲ نیز به 6 واحد افزایش یافته (زیرا دیگر باید به تنهایی هزینه‌ی مسیر میانی را بپردازد) و در نتیجه، مسیر خود را تغییر خواهد داد و مسیر بیرونی را (با 5 واحد هزینه) انتخاب خواهد کرد. به این ترتیب این ساختار تنها یک تعادل نش دارد که معادل با عبور کردن هر دو بازیگر از مسیرهای بیرونی مربوط به خود است؛ و این تعادل نش یک وضعیت بهینه‌ی اجتماعی نیست.



شکل ۹

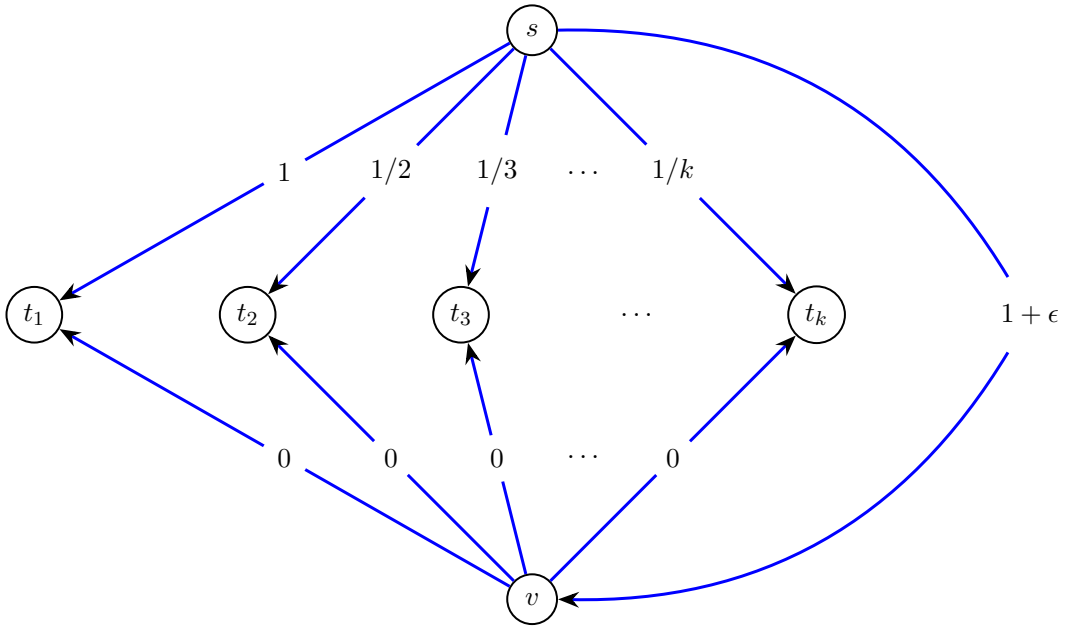
به این ترتیب مشاهده می‌شود که وضعیت بهینه‌ی اجتماعی (متناظر با بهینه‌ی سراسری در مسائل عادی) لزوماً یک تعادل نش (معادل با بهینه‌ی موضعی در مسائل عادی) نمی‌باشد و این یک تفاوت مهم این مسأله با مسائل عادی دیگری است که در بحث روش‌های جست‌وجوی محلّی مورد بررسی قرار داده بودیم.

هزینه‌ی پایداری[۱۴]. دیدیم که حالت بهینه‌ی اجتماعی، لزوماً یک تعادل نش نیست. نسبت هزینه‌ی بهترین تعادل نش به هزینه‌ی حالت بهینه‌ی اجتماعی را *هزینه‌ی پایداری* می‌نامیم. به عبارتی، یک مسأله‌ی جالب این است که هزینه‌ی پایداری چقدر می‌تواند زیاد باشد؛ یعنی چه میزان فاصله بین بهترین تعادل نش تا حالت بهینه‌ی اجتماعی وجود دارد. در مثال شکل‌های ۸ و ۹، این هزینه به ترتیب برابر با 1 (حالتی که بهینه‌ی اجتماعی، خود یک تعادل نش است) و $8/7$ بود.

این بار، مثالی دیگری را بررسی می‌کنیم که هزینه‌ی پایداری آن به مراتب بالاتر باشد. مسأله‌ای با k بازیگر را در نظر بگیرید که گراف آن مطابق با شکل ۱۰ باشد.

در این مثال، نقطه‌ی بهینه‌ی اجتماعی آن است که تمامی بازیگران از مسیر سمت راست که هزینه‌ای برابر با $1+\epsilon$ دارد عبور کنند. در

[14] price of stability



شکل ۱۰

این حالت، کلّ هزینه‌ی پرداختی نیز $1+\epsilon$ خواهد بود. با این حال، اگر در چنین وضعیتی قرار داشته باشیم، بازیگر شماره k انگیزه دارد مسیر خود را تغییر بدهد و از مسیر مستقیم st_k عبور کند که هزینه‌ی آن برابر با $1/k$ است (این بازیگر، قبلاً هزینه‌ای برابر با $1/k+\epsilon/k$ پرداخت می‌کرد و انگیزه‌ی تغییر مسیر دارد). پس از این اتّفاق، هزینه‌ی سایر بازیگران باقی‌مانده برابر با $(1+\epsilon)/(k-1)$ می‌شود و به این ترتیب، بازیگر شماره $k-1$ انگیزه‌ی تغییر مسیر پیدا می‌کند. به همین ترتیب، هر بار که یکی از بازیگران مسیر خود را تغییر می‌دهد، بازیگر بعدی انگیزه‌ی تغییر مسیر پیدا می‌کند تا این که نهایتاً تمامی بازیگران مسیرهای خود را تغییر داده و از مسیر مستقیم st_j عبور می‌کنند. این وضعیت، یک تعادل نش است.

در این حالت هزینه‌ی پایداری برابر با $H(k)/(1+\epsilon)$ خواهد بود که $H(k)$ عدد هارمونیک kاُم بوده و از مرتبه‌ی $\Theta(\lg k)$ است:

$$\text{هزینه‌ی پایداری} = \frac{\text{هزینه‌ی بهترین تعادل نش}}{\text{هزینه‌ی حالت بهینه‌ی اجتماعی}} = \frac{\sum_{i=1}^{k}\frac{1}{i}}{1+\epsilon} = \frac{H(k)}{1+\epsilon} = \Theta(\lg k) \qquad (۱۲)$$

تا به این‌جا به بررسی مثال‌های مختلفی از مسأله‌ی مسیریابی چندتایی و تعادل‌های نش آن پرداخته‌ایم؛ امّا هنوز گزاره‌ای کلّی در مورد پایان‌پذیری قطعی اجرای دینامیک مذکور برای این سیستم نداریم. قضیه‌ی ۳، چنین گزاره‌ای را در اختیار ما قرار خواهد داد.

قضیه ۳. *الگوریتم زیر همواره با یک تعادل نش خاتمه می‌پذیرد (یعنی قطعاً پایان می‌پذیرد، و نقطه‌ی پایانی آن نیز قطعاً یک تعادل نش است):*

BestResponseDynamics(G, c, k)

1 **for** $j = 1$ to k
2 $\quad P_j =$ any path for agent j
3 **while** not a Nash equilibrium
4 $\quad j =$ some agent who can improve by switching paths
5 $\quad P_j =$ better path for agent j
6 return $(P_1, P_2, \ldots, P_k)$

اثبات. در حالت کلّی برای اثبات خاتمه‌پذیری روش‌های جست‌وجوی محلّی، یک رویکرد کلّی و مورد استفاده آن بود که نشان دهیم مقدار هزینه‌ی حالت انتخاب‌شده در هر گام (حدّاقل به اندازه‌ی ثابتی) کاهش می‌یابد، و با توجّه به نامنفی بودن این هزینه، قطعاً الگوریتم باید به یک نقطه‌ی تعادل برسد. در مسأله‌ی حاضر، مشاهده کردیم که مقدار هزینه‌ی اجتماعی (به عنوان یک ملاک عمومی از بهینگی) ممکن است در گام‌های متوالی افزایش یابد؛ و روش مذکور برای اثبات پایان‌پذیری فرآیند در این‌جا مؤثّر نیست. برای حلّ این مشکل، تابع پتانسیلی تعریف می‌کنیم که ویژگی مطلوب را داشته باشد، یعنی در هر گام از اجرای الگوریتم، به اندازه‌ی مقدار مشخّصی کاهش یابد.

دیدیم که هزینه‌ی اجتماعی، مجموع وزن یال‌هایی است که حدّاقل در یک مسیر به کار گرفته شده‌اند (و مستقل از این که از یالی مثل e چند بازیگر عبور می‌کنند، سهم این یال در هزینه‌ی اجتماعی برابر با c_e بود). در تابع پتانسیل جدیدی که تعریف می‌کنیم، تعداد بازیگرانی که یال e را در مسیر خود دارند نیز مؤثّر خواهد بود. اگر تعداد بازیگرانی که یال e را در مسیر خود دارند با x_e نشان دهیم، تابع پتانسیل Φ را به صورت زیر تعریف می‌کنیم:

$$\Phi(P_1, \dots, P_n) = \sum_{e \in E} c_e H(x_e) \tag{۱۳}$$

که در آن، $H(k)$ عدد هارمونیک kاُم با رابطه‌ی $H(k) = \sum_{i=1}^{k} \frac{1}{i}$ است و همچنین تعریف می‌کنیم $H(0) = 0$.

حال نشان می‌دهیم که در هر گام از اجرای الگوریتم، مقدار پتانسیل Φ اکیداً کاهش می‌یابد. برای این منظور، فرض کنید در یک گام از الگوریتم، بازیگر jاُم، مسیر خود را از P_j به P'_j تغییر دهد. بخش‌هایی از این دو مسیر که با هم اشتراک دارند، در هر دو حالت هزینه‌ی ثابتی به این بازیگر تحمیل می‌کنند؛ امّا تفاوت اصلی از یال‌هایی از دو مسیر ناشی می‌شود که مشترک نیستند. به عبارتی، برای آن که این بازیگر تصمیم به تغییر مسیر بگیرد باید مجموع هزینه‌هایی که این بازیگر برای یال‌های P'_j (که با P_j مشترک نیستند) می‌پردازد، کمتر از مجموع هزینه‌ای باشد که پیش‌تر برای یال‌های P_j (که مشترک با P'_j نیستند) می‌پرداخت؛ یعنی:

$$\sum_{f \in P'_j - P_j} \frac{c_f}{x_f + 1} < \sum_{e \in P_j - P'_j} \frac{c_e}{x_e} \tag{۱۴}$$

حال کافی است مقدار تغییرات Φ را محاسبه کنیم. داریم:

$$\begin{aligned} \text{مقدار افزایش } \Phi &= \sum_{f \in P'_j - P_j} c_f \left[H(x_f + 1) - H(x_f)\right] = \sum_{f \in P'_j - P_j} \frac{c_f}{x_f + 1} \\ \text{مقدار کاهش } \Phi &= \sum_{e \in P_j - P'_j} c_e \left[H(x_e) - H(x_e - 1)\right] = \sum_{e \in P_j - P'_j} \frac{c_e}{x_e} \end{aligned} \tag{۱۵}$$

و با مقایسه‌ی روابط ۱۴ و ۱۵ مشاهده می‌شود که مقدار افزایش Φ اکیداً کمتر از میزان کاهش آن است؛ و این یعنی در هر گام از اجرای الگوریتم مقدار Φ اکیداً کم می‌شود. به این ترتیب با توجّه به نامنفی بودن مقادیر Φ، الگوریتم باید پایان بپذیرد. (البته برای آن که دقیق‌تر باشیم، باید نشان دهیم که میزان کاهش Φ در هر گام، از یک مقدار ثابت مثبت بیشتر است. برای این منظور کافی است دقّت کنیم که مقادیر c_e و x_e از میان تعدادی حالت متناهی انتخاب می‌شوند؛ به این ترتیب برای میزان کاهش پتانسیل در هر گام، تعدادی متناهی مقدار مختلف وجود دارد، که اگر کمینه‌ی این مقادیر را در نظر بگیریم و آن را Δ بنامیم، میزان کاهش پتانسیل در هر مرحله حدّاقل برابر با Δ است.) □

در ادامه می‌توانیم به کمک همین تابع پتانسیل، کران بالایی برای مقدار هزینه‌ی پایداری نیز بیابیم. برای منظور، ابتدا یک لم ساده را بیان و اثبات می‌کنیم.

لم ۱. *اگر هزینه‌ی انتخاب مسیرهای $P_1, \dots, P_k$ را با $C(P_1, \dots, P_k)$ نمایش دهیم، آن‌گاه برای هر انتخاب دل‌خواهی از این مسیرها داریم:*

$$C(P_1, \dots, P_k) \le \Phi(P_1, \dots, P_k) \le H(k) C(P_1, \dots, P_k) \tag{۱۶}$$

اثبات. اگر x_e نشان‌دهنده‌ی تعداد مسیرهایی که یال e را در بر دارند، و E^+ نشانگر مجموعه‌ی همه‌ی یال‌هایی باشد که حدّاقل در یک مسیر قرار گرفته‌اند، خواهیم داشت:

$$\begin{aligned} C(P_1,\dots,P_k) &= \sum_{e\in E^+} c_e \\ &\le \sum_{e\in E^+} c_e H(x_e) \\ &= \Phi(P_1,\dots,P_k) \\ &\le \sum_{e\in E^+} c_e H(k) \\ &= H(k)C(P_1,\dots,P_k) \end{aligned} \tag{۱۷}$$

□

حال می‌توانیم قضیه‌ای را برای معرّفی یک کران بالا برای هزینه‌ی پایداری معرّفی و اثبات کنیم:

قضیه ۴. *در هر ساختاری از مسأله‌ی مسیریابی چندتایی، یک تعادل نش وجود دارد که مجموع هزینه‌های همه‌ی بازیگرها در آن، حدّاکثر به اندازه‌ی ضریب $H(k)$ از هزینه‌ی حالت بهینه‌ی اجتماعی بیشتر باشد.*

اثبات. فرض کنید $(P_1^*,\dots,P_k^*)$ مجموعه‌ای از مسیرها باشد که منجر به حالت بهینه‌ی اجتماعی شود. در این صورت فرض کنید الگوریتم مربوط به دینامیک سیستم با شروع از حالت اوّلیه‌ی $(P_1^*,\dots,P_k^*)$ اجرا شود و نهایتاً در حالت $(P_1,\dots,P_k)$ به یک تعادل نش برسد. طبق قضیه‌ی ۳، چنین تعادلی حتماً وجود دارد. همچنین دیدیم که طیّ انجام این فرآیند، مقدار تابع Φ کاهش می‌یابد، بنابراین داریم $\Phi(P_1,\dots,P_k) \le \Phi(P_1^*,\dots,P_k^*)$. حال با در نظر گرفتن لم ۱ می‌توان نوشت:

$$\begin{aligned} C(P_1,\dots,P_k) &\le \Phi(P_1,\dots,P_k) \\ &\le \Phi(P_1^*,\dots,P_k^*) \\ &\le H(k)C(P_1^*,\dots,P_k^*) \end{aligned} \tag{۱۸}$$

که در عبارات فوق، نامساوی‌های اوّل و سوم به ترتیب ناشی از اعمال نتیجه‌ی لم ۱ به ترتیب برای P و P^* هستند و نامساوی دوم نیز نتیجه‌ی قضیه‌ی ۳ است.

به این ترتیب داریم $C(P_1,\dots,P_k) \le H(k)C(P_1^*,\dots,P_k^*)$ و حکم ثابت شده است. □

به این ترتیب برای مسأله‌ی مسیریابی چندتایی، وجود یک نقطه‌ی تعادل نش را ثابت کرده‌ایم و حدّاکثر اختلاف هزینه‌ی آن با هزینه‌ی پایداری را نیز به دست آورده‌ایم؛ با این حال جالب است بدانید که برای مسأله‌ی یافتن این تعادل نش در زمان چندجمله‌ای در حال حاضر حلّی پیدا نشده است.

۸ الگوریتم (تصادفی) جست‌وجوی محلّی برای مسأله‌ی 3SAT (اختیاری)

در جلسه‌ی قبلی، الگوریتمی برای یافتن یک مقداردهی مناسب برای مسأله‌ی 3SAT با استفاده از روش پَسگرد[15] ارائه کردیم. با این حال، الگوریتم‌های مبتنی بر روش پسگرد از نظر زمان اجرای بدترین حالت نمی‌توانند کران زمانی مناسبی در اختیار قرار دهند. در این بخش می‌خواهیم با استفاده از جست‌وجوی محلّی، روشی برای یافتن یک مقداردهی مناسب برای مسأله‌ی 3SAT به دست آوریم که این مشکل را حل کند. به علاوه، زمان اجرای بهترین الگوریتم موجود برای انجام چنین کاری، تقریباً از مرتبه‌ی $O(1.31^n)$ است. الگوریتمی که در این بخش معرّفی می‌کنیم بسیار به این حد نزدیک خواهد بود و زمان اجرای بدترین حالت آن، از مرتبه‌ی $O((\frac{4}{3})^n)$ می‌باشد.

صورت این الگوریتم به این شکل است که در هر بار تکرار آن، یک مقداردهی تصادفی مثل $a \in \{0,1\}^n$ برای متغیّرهای مسأله‌ی 3SAT در نظر می‌گیریم (n تعداد متغیّرهاست) و حدّاکثر n بار، فرآیند اصلاح را انجام می‌دهیم. منظور از فرآیند اصلاح آن است که اگر در وضعیت فعلی a، یک عبارت مانند C از ورودی مسأله موجود باشد که توسّط a ارضا نشده باشد، یکی از لیترال‌های آن را به تصادف انتخاب کرده و مقدار آن لیترال را در a برعکس می‌کنیم.

به این ترتیب یک بار تکرار از فرآیند کلّی الگوریتم را طی کرده‌ایم که شامل یک بار انتخاب تصادفی a و حدّاکثر n بار اجرای فرآیند اصلاح است. کلّ این فرآیند را T بار اجرا می‌کنیم، و اگر در پایان یکی از این T بار اجرای الگوریتم، به یک مقداردهی ارضاکننده برای ورودی مسأله رسیدیم، اعلام می‌کنیم که فرمول داده‌شده ارضاپذیر است. اگر خروجی همه‌ی این T بار اجرا، مقداردهی‌های نامعتبر بود؛ اعلام می‌کنیم که فرمول ورودی ارضاپذیر نمی‌باشد. شبه کد زیر نیز همین فرآیند را بیان می‌کند (منظور از Φ فرمول ورودی مسأله است):

```
LocalSearch3SAT(Φ)
1  for i = 1 to T
2      choose a ∈ {0,1}^n from uniform distribution
3      for j = 1 to n
4          choose an expression C which is not satisfied by a (skip if none exists)
5          choose a literal x from C at random
6          set x = x̄ in a
7      if a satisfies Φ
8          return Yes
9  return No
```

در ادامه می‌خواهیم با یک تحلیل احتمالاتی، احتمال موفّقیت الگوریتم فوق در تشخیص ارضاپذیری یک فرمول داده‌شده را محاسبه کنیم. به وضوح اگر ورودی مسأله ارضاپذیر نباشد، قطعاً خروجی الگوریتم فوق نیز پاسخ منفی خواهد بود. بنابراین تمرکز اصلی باید بر روی آن باشد که بررسی کنیم زمانی که فرمول ورودی ارضاپذیر است، با چه احتمالی الگوریتم قادر به پیدا کردن یک مقداردهی مناسب a و ارائه‌ی پاسخ مثبت خواهد بود. دقّت کنید که در ابتدا در مورد مقدار پارامتر T صحبتی به میان نمی‌آوریم؛ نخست احتمال موفّقیت یک بار اجرای حلقه‌ی اصلی برنامه را محاسبه می‌کنیم و پس از آن، مقدار T را طوری تعیین می‌کنیم که احتمال موفّقیت کلّی الگوریتم به قدر کافی بالا باشد.

برای این منظور، فرض کنید a^* یک مقداردهی ارضاکننده برای فرمول ورودی باشد. $\mathrm{d}(a, a^*)$ را *فاصله‌ی* همینگ[16] دو رشته‌ی a و a^* تعریف می‌کنیم که منظور از آن، تعداد عناصری از این دو رشته است که با هم متفاوتند. به عبارت دیگر، a و a^* دو رشته‌ی متشکّل از صفر و یک‌ها با طول n هستند و فاصله‌ی همینگ آن‌ها برابر است با تعداد اندیس‌هایی مثل j که $a_j \neq a^*_j$.

[15] backtracking
[16] Hamming distance

حال می‌توان احتمال موفّقیت در یک بار اجرای حلقه‌ی اصلی الگوریتم را با شرطی‌سازی بر روی فاصله‌ی همینگ مقدار اوّلیه‌ی a با a^* به صورت زیر نوشت:

$$\mathbb{P}\left[\text{success}\right] = \sum_{k=0}^{n} \mathbb{P}\left[\,\mathrm{d}(a,a^*)=k\right] \mathbb{P}\left[\text{success}\middle|\,\mathrm{d}(a,a^*)=k\right] \tag{۱۹}$$

حال احتمال آن که یک رشته‌ی تصادفی مثل a فاصله‌ای برابر با k تا a^* داشته باشد برابر است با تعداد کلّ چنین رشته‌هایی ضرب در 2^{-n}، که تعداد آن‌ها نیز برابر با $\binom{n}{k}$ خواهد بود. حال برای احتمال موفّقیت به شرط آن که فاصله‌ی a و a^* برابر با k باشد یک کران پایین معرّفی می‌کنیم. برای این منظور، به طور سخت‌گیرانه‌ای فرض کنید برای موفّقیت بخواهیم در k اجرای اوّل حلقه‌ی داخلی الگوریتم (منظور حلقه‌ای از الگوریتم است که n بار اجرا می‌شود و در خط سوم شبه‌کد شروع می‌شود)، هر بار فاصله‌ی a و a^* یک واحد کم شود تا نهایتاً در گام kاُم a و a^* برابر شوند (دقّت کنید که در حالت کلّی برای موفّقیت لازم نیست حتماً چنین اتّفاقی بیفتد و ممکن است در برخی گام‌ها فاصله‌ی a و a^* زیاد شود و سپس دوباره کم شود. حتّی یک گام هم فراتر، ممکن است مقداردهی‌های درست متفاوتی برای فرمول ورودی وجود داشته باشد و حتّی لازم نباشد a حتماً مشابه a^* شود تا موفّقیت حاصل شود. با وجود همه‌ی این موارد، چون به دنبال یک کران پایین برای احتمال موفّقیت هستیم، یک حالت سخت‌گیرانه را در نظر گرفته‌ایم که a در k گام ابتدایی از حلقه‌ی داخلی الگوریتم به a^* برسد). برای این منظور کافی است به فرآیند اجرای حلقه‌ی داخلی الگوریتم دقّت کنیم که در آن، پس از انتخاب یک عبارت ارضانشده مثل C، یکی از لیترال‌های آن به تصادف انتخاب شده و مقداردهی آن برعکس می‌شود. با توجّه به این که a^* یک مقداردهی ارضاکننده برای کلّ فرمول ورودی است، قطعاً C نیز به وسیله‌ی a^* ارضا می‌شود؛ پس حدّاقل یکی از لیترال‌های C وجود دارد که با تغییر مقداردهی فعلی آن در a، مقداری به خود بگیرد که مشابه مقداردهی آن توسّط a^* شود (زیرا اگر برای هیچ یک از لیترال‌های C این اتّفاق نیفتد، a^* نمی‌توانسته C را ارضا کند). به این ترتیب (و با انتخاب آن لیترال خاص) حدّاقل به احتمال $\frac{1}{3}$، در این گام فاصله‌ی a و a^* یک واحد کم می‌شود. به این ترتیب می‌توان رابطه‌ی ۱۹ را به صورت زیر تکمیل کرد:

$$\begin{aligned}
\mathbb{P}\left[\text{success}\right] &= \sum_{k=0}^{n} \mathbb{P}\left[\,\mathrm{d}(a,a^*)=k\right] \mathbb{P}\left[\text{success}\middle|\,\mathrm{d}(a,a^*)=k\right] \\
&\geq \sum_{k=0}^{n} \binom{n}{k} 2^{-n} \left(\frac{1}{3}\right)^k \\
&= 2^{-n} \sum_{k=0}^{n} \binom{n}{k} \left(\frac{1}{3}\right)^k \\
&= 2^{-n} \left(1+\frac{1}{3}\right)^n \\
&= \left(\frac{2}{3}\right)^n
\end{aligned} \tag{۲۰}$$

به این ترتیب احتمال موفّقیت در یک بار اجرای حلقه‌ی اصلی الگوریتم به دست آمد. حال می‌خواهیم مقداری برای T انتخاب کنیم که الگوریتم کلّی پس از T بار اجرا، با احتمال مناسبی به یک مقداردهی مناسب رسیده باشد و پاسخ مثبت را اعلام کند.

برای بررسی این موضوع، لم زیر را معرّفی و اثبات می‌کنیم:

لم ۲. *اگر احتمال موفّقیت الگوریتمی در یک بار اجرا برابر با p باشد، برای آن که احتمال موفّقیت را به $1-\frac{1}{n^c}$ برسانیم، کافی است الگوریتم را $\Theta\left(\frac{\lg n}{p}\right)$ بار اجرا کنیم. (c یک عدد ثابت است.)*

اثبات. برای اثبات، فرض می‌کنیم الگوریتم به اندازه‌ی $c\frac{\ln n}{p}$ بار اجرا شود و در این حالت، احتمال شکست را محاسبه می‌کنیم (منظور از

شکست، شکست در تمامی دفعات اجرای الگوریتم است؛ چرا که اگر الگوریتم یک بار موفّق شود، عملاً موفّق شده است). خواهیم داشت:

$$\begin{aligned}\mathbb{P}[\text{failure}] &= (1-p)^{c(\ln n)/p} \\ &\le \left(e^{-p}\right)^{c(\ln n)/p} \\ &= e^{-c\ln n} \\ &= \frac{1}{n^c}\end{aligned} \tag{۲۱}$$

که برای نامساوی مورد استفاده در روابط ۲۱، از رابطه‌ی $1+x \le e^x$ استفاده کرده‌ایم.

به این ترتیب اثبات شده است که اگر الگوریتم را $c\frac{\ln n}{p}$ بار اجرا کنیم، احتمال شکست حدّاکثر برابر با $\frac{1}{n^c}$ بوده و به احتمال حدّاقل $1-\frac{1}{n^c}$ موفّق خواهیم شد. □

به این ترتیب با توجّه به لم ۲ و با جایگذاری احتمال موفّقیت از نتیجه‌ی رابطه‌ی ۲۰، خواهیم دید که می‌توان با قرار دادن مقدار $T = \Theta\left(\frac{1}{p}\lg n\right) = \Theta\left(\left(\frac{3}{2}\right)^n \lg n\right)$، احتمال موفّقیت را از هر مقدار دلخواهی بزرگ‌تر کرد.

حال می‌خواهیم مرتبه‌ی زمانی الگوریتم، یعنی مقدار T را کوچک‌تر کنیم. برای این منظور، تحلیل احتمالاتی خود را کمی تغییر می‌دهیم. ابتدا یک تغییر کوچک در الگوریتم اعمال می‌کنیم و فرض می‌کنیم حلقه‌ی داخلی آن (خطّ سوم شبه‌کد الگوریتم) به جای n، به اندازه‌ی $3n$ بار اجرا شود. در ادامه، تعریف خود از موفّقیت الگوریتم در هر بار اجرای حلقه‌ی بیرونی آن را کمی ساده‌تر می‌کنیم. پیش‌تر این موفّقیت را منحصر به آن کرده بودیم که در نخستین k بار اجرای حلقه‌ی داخلی الگوریتم، در هر گام فاصله‌ی a و a^* یک واحد کم شود. این بار تعریف خود از موفّقیت را به این صورت در نظر می‌گیریم که در $3k$ بار اجرای حلقه‌ی داخلی الگوریتم، k بار این امکان وجود داشته باشد که فاصله‌ی a و a^* زیاد شود؛ و در $2k$ بار دیگر این فاصله کم شود. به این ترتیب، پس از $3k$ بار اجرای این حلقه، فاصله‌ی a و a^* در مجموع k واحد کاهش یافته و با توجّه به شرطی‌سازی مسأله و این فرض که فاصله‌ی اوّلیه‌ی آن‌ها برابر با k بوده، این بدان معناست که a برابر با a^* شده است. همچنین پیش‌تر احتمال آن که در یک گام فاصله‌ی a و a^* یک واحد کم شود را حدّاقل برابر با $\frac{1}{3}$ در نظر گرفته بودیم؛ بنابراین می‌توان با اعمال تمهیدات جدید در الگوریتم، رابطه‌ای مشابه با رابطه‌ی ۲۰ را به صورت زیر نوشت:

$$\mathbb{P}[\text{success}] \ge \sum_{k=0}^{n} 2^{-n}\binom{n}{k}\binom{3k}{k}\left(\frac{1}{3}\right)^{2k}\left(\frac{2}{3}\right)^{k} \tag{۲۲}$$

برای ساده‌سازی رابطه‌ی ۲۲، از فرمول تقریبی استرلینگ استفاده می‌کنیم:

$$n! = \Theta\left(\sqrt{n}\left(\frac{n}{e}\right)^n\right) \tag{۲۳}$$

و می‌توان مقدار $\binom{3k}{k}$ را به کمک رابطه‌ی ۲۳ به صورت زیر ساده کرد:

$$\begin{aligned}\binom{3k}{k} &= \frac{(3k)!}{(2k)!k!} \\ &= \Theta\left(\frac{\sqrt{3k}}{\sqrt{k}\sqrt{2k}}\frac{\left(\frac{3k}{e}\right)^{3k}}{\left(\frac{k}{e}\right)^{k}\left(\frac{2k}{e}\right)^{2k}}\right) \\ &= \Theta\left(\frac{1}{\sqrt{k}}\frac{3^{3k}}{2^{2k}}\right)\end{aligned} \tag{۲۴}$$

و با جای‌گذاری نتیجه‌ی رابطه‌ی ۲۴ در رابطه‌ی ۲۲ خواهیم داشت:

$$\begin{aligned}\mathbb{P}[\text{success}] &\geq \sum_{k=0}^{n} 2^{-n}\binom{n}{k}\binom{3k}{k}\left(\frac{1}{3}\right)^{2k}\left(\frac{2}{3}\right)^{k} \\ &= \sum_{k=0}^{n} 2^{-n}\binom{n}{k}\Theta\left(\frac{2^{-k}}{\sqrt{k}}\right) \\ &\geq c2^{-n}\sum_{k=0}^{n}\binom{n}{k}\frac{2^{-k}}{\sqrt{k}} \\ &\geq \frac{c}{\sqrt{n}}2^{-n}\sum_{k=0}^{n}\binom{n}{k}2^{-k} \\ &= \frac{c}{\sqrt{n}}2^{-n}\left(1+\frac{1}{2}\right)^{n} \\ &= \frac{c}{\sqrt{n}}\left(\frac{3}{4}\right)^{n}\end{aligned} \tag{۲۵}$$

به این ترتیب احتمال موفّقیت در الگوریتم جدید به دست می‌آید و با استفاده از نتیجه‌ی لم ۲، انتخابی از T که احتمال موفّقیت را به قدر کافی بزرگ می‌کند به صورت زیر محاسبه می‌شود:

$$T = \Theta\left(\frac{1}{p}\lg n\right) = \Theta\left(\left(\frac{4}{3}\right)^{n}\sqrt{n}\lg n\right) \tag{۲۶}$$

و بحث در مورد این الگوریتم به پایان رسیده است.

به عنوان یک نکته‌ی پایانی، شاید این سؤال به ذهن بیاید که بهترین زمان اجرای ممکن برای یافتن پاسخ یک مسأله‌ی 3SAT چقدر است. در این مورد، فرضیه‌ای به نام *فرضیه‌ی زمان نمایی*[۱۷] (ETH) وجود دارد که شرح آن به صورت زیر است:

فرضیه‌ی زمان نمایی. زمان اجرای هر الگوریتم برای 3SAT، در بدترین حالت c^n است که $c > 1$. دقّت کنید که این فرضیه، حتّی از فرضیه‌ی $\mathbf{P} \neq \mathbf{NP}$ نیز قوی‌تر است، چرا که با فرض $\mathbf{P} \neq \mathbf{NP}$ هنوز ممکن است الگوریتم‌هایی با زمان اجراهای بهتر از زمان نمایی (مثلاً الگوریتمی با زمان اجرای $O(n^{\lg n})$) برای مسأله‌ی 3SAT موجود باشد؛ امّا فرضیه‌ی زمان نمایی حتّی این امکان را نیز رد می‌کند. اکثر محقّقین این فرضیه را معتبر می‌دانند.

علاوه بر این، فرضیه‌ی دیگری با نام *فرضیه‌ی زمان نمایی قوی*[۱۸] (SETH) نیز وجود دارد که بیان می‌دارد برای یک مسأله‌ی SAT در حالت کلّی (که تعداد لیترال‌های هر عبارت بتواند مقادیر دلخواهی باشد)، هیچ الگوریتمی وجود ندارد، مگر آن که زمان اجرای آن از مرتبه‌ی c^n باشد و $c \geq 2$. این فرض بسیار قوی است و محقّقین کمتری (در مقایسه با فرضیه‌ی زمان نمایی) آن را قبول دارند. این فرضیه در صورت صحّت، نتایج جالبی را به دنبال دارد. به عنوان مثال، در حوزه‌ی *پیچیدگی ریزدانه*[۱۹] (حوزه‌ای که در مورد پیچیدگی محاسباتی مسائل با دقّت بیشتری بحث می‌کند و به عنوان مثال، بررسی این که آیا مسأله‌ای با زمان اجرای $O(n^2)$ می‌تواند الگوریتمی از مرتبه‌ی $O(n\lg n)$ داشته باشد را نیز هدف قرار می‌دهد) کاربردهای جالبی دارد. به عنوان مثال، با فرض درستی SETH، نشان داده می‌شود که مسأله‌ی فاصله‌ی ویرایش[۲۰] الگوریتمی با زمان اجرای $O(n^2)$ دارد، امّا هیچ الگوریتمی با زمان اجرای $O(n^{2-\epsilon})$ ندارد. البته با توجّه به این که فرضیاتی نظیر فرضیه‌ی زمان نمایی قوی، حتّی از فرض $\mathbf{P} \neq \mathbf{NP}$ نیز قوی‌ترند، انتظار نمی‌رود که به سادگی اثبات شوند؛ مگر آن که امکان رد کردن آن‌ها فراهم شود.

[17] Exponential Time Hypothesis
[18] Strong Exponential Time Hypothesis
[19] fine-grained complexity
[20] edit distance

مراجع

[1] Kleinberg, Jon and Tardos, Eva. *Introduction to Algorithms*. 1st ed., 2005, pp. 661-700.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس مهمان: هانی احمدزاده

[بهار ۹۹]

جلسه ۲۳: برنامه‌ریزی خطی ۱ نگارنده: جواد فرخ‌نژاد

در جلسه‌ی پیش الگوریتم‌هایی را بررسی کردیم که برای بعضی مسأله‌ها جواب‌های بهینه‌ی موضعی پیدا می‌کردند و این جواب را به عنوان تخمینی از جواب بهینه‌ی کل در نظر می‌گرفتند. در این جلسه قصد داریم مفهوم جدیدی به نام برنامه‌ریزی‌خطی[۱] را معرفی و بررسی کنیم.

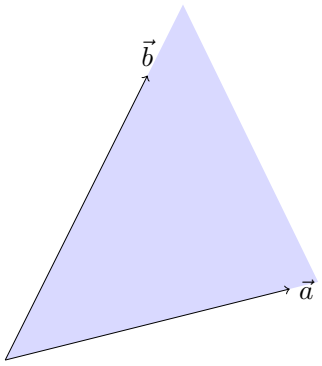
۱ برنامه‌ریزی خطی

۱.۱ یادآوری

ابتدا چند نکته از ریاضی عمومی و جبرخطی، که در ادامه از آن‌ها استفاده می‌شود را یادآوری می‌کنیم.

منظور از یک ترکیب‌خطی برای بردارهای $\vec{a_1}, \vec{a_2}, \dots, \vec{a_n}$ برداری به شکل $\lambda_1\vec{a_1} + \lambda_2\vec{a_2} + \cdots + \lambda_n\vec{a_n}$ است که ضرایب آن یعنی λ_iها از میدان موردبحث انتخاب می‌شوند. مثلا اگر دو بردار $\vec{a}$ و $\vec{b}$ در فضای اقلیدسی $\mathbb{R}^2$ را درنظر بگیریم که مستقل خطی‌اند، ترکیب‌خطی‌های این دو بردار برابر با کل فضا می‌شود.

منظور از ترکیب‌خطی نامنفی این است که ضرایب نامنفی باشند. مثلا در فضای $\mathbb{R}^2$ ترکیب‌خطی‌های نامنفی دوبردار $\vec{a}$ و $\vec{b}$، ناحیه‌ی رنگ شده در شکل را می‌پوشانند.



ناحیه‌ای که ترکیب‌خطی نامنفی $\vec{a}$ و $\vec{b}$ می‌پوشاند

رتبه[۲] ی یک ماتریس برابر است با تعداد سطرهای مستقل خطی آن یا تعداد ستون‌های مستقل خطی آن که هردو برابرند.

فرض کنید $f : \mathbb{R}^n \to \mathbb{R}$ یک تابع باشد و $c \in \mathbb{R}$ در برد f باشد. در این‌صورت منحنی ترازِ[۳] c برای f را که با L_c نمایش می‌دهیم،

[۱] Linear Programming
[۲] rank
[۳] contour line

مجموعه‌ی همه‌ی نقاطی از $\mathbb{R}^n$ تعریف می‌کنیم که مقدار f روی آن‌ها برابر با c است یعنی:

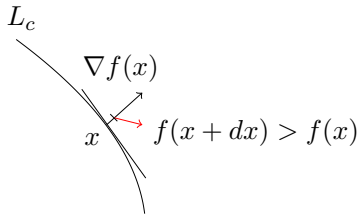
$$L_c = \{x \in \mathbb{R}^n \mid f(x) = c\}$$

اگر تمام cهای مختلف را درنظر بگیریم و به ازای تمام آن‌ها منحنی تراز را در $\mathbb{R}^n$ رسم کنیم، این منحنی‌ها دامنه‌ی تابع f را افراز می‌کنند. درواقع دامنه‌ی f به بخش‌هایی افراز می‌شود که مقدار تابع روی هرکدام از بخش‌ها ثابت است و این درمورد رفتار تابع اطلاعات خوبی به ما می‌دهد.

برای یک تابع $f : \mathbb{R}^n \to \mathbb{R}$ اگر مشتقات پاره‌ای[۴] تابع در نقطه‌ی $x \in \mathbb{R}^n$ موجود باشند آنگاه گرادیان[۵] این تابع در نقطه‌ی x که با نماد $\nabla f(x)$ نمایش داده می‌شود، برداری (یا بنابر کاربرد ماتریس ستونی) است که درایه‌های آن مشتقات پاره‌ای f در نقطه‌ی x اند، یعنی:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x) \\ \frac{\partial f}{\partial x_2}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{bmatrix} \in \mathbb{R}^n$$

نکته‌ی بسیار مهم این است که گرادیان تابع در هر نقطه بر منحنی تراز تابع در آن نقطه عمود است و اگر یک نقطه در دامنه مثل x در نظر بگیریم و در راستای گرادیان به اندازه خیلی کوچک حرکت کنیم به نقطه‌ای مثل $x + dx$ می‌رسیم که مقدار تابع f روی این نقطه از مقدارش روی x بیشتر است. یعنی حرکت در راستای گرادیان باعث افزایش مقدار تابع می‌شود.



۲.۱ تعریف مسأله

هدف از برنامه‌ریزی خطی بیشینه یا کمینه کردن یک تابع خطی برحسب تعدادی متغیر روی یک ناحیه از فضای اقلیدسی است. به‌طور دقیق‌تر فرض کنید $c_1, c_2, \ldots, c_n \in \mathbb{R}$ ثوابتی حقیقی باشند و تعریف کنید $z = c_1x_1 + c_2x_2 + \ldots + c_nx_n$ که به آن تابع هدف[۶] می‌گویند و می‌خواهیم بیشینه یا کمنیه‌ی z را با شروط[۷] زیر بیابیم.

$$\forall i \in \{1, 2, \ldots, m\} \quad a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \quad (\leq = \geq) \quad b_i$$

$$\forall j \in \{1, 2, \ldots, n\} \quad l_j \leq x_j \leq u_j$$

که در آن a_{ij}ها و b_iها همگی اعداد حقیقی‌اند و u_jها و l_jها می‌توانند به‌ترتیب $+\infty$ و $-\infty$ نیز باشند.
به x_iها متغیرهای تصمیم[۸] می‌گویند و در بعضی منابع به a_{ij}ها ضرایب تکنولوژی می‌گویند اما این‌جا خیلی به این نام‌گذاری تاکیدی نداریم. در این‌جا می‌توان فرض‌کرد که برای هر $1 \leq j \leq n$ داریم $l_j < u_j$ چرا که درحالت تساوی l_j و u_j به وضوح x_j تنها مقداری که می‌تواند اتخاذ کند همین $l_j = u_j$ است و بنابراین تکلیف x_j مشخص است.

چون فرمول‌بندی بالا خیلی کلی است ابتدا حالت خاصی از آن را با نام برنامه‌ریزی خطی استاندارد[۹] بررسی می‌کنیم سپس ثابت می‌کنیم

[۴] partial derivatives
[۵] gradient
[۶] Objective function
[۷] constraints
[۸] Decision variable
[۹] Standard LP

که هر چنین فرم کلی را می‌توان به حالت استاندارد تبدیل کرد. فرم استاندار برنامه‌ریزی خطی به صورت زیر است:

$$\begin{cases} \min & z = c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{s.t.} & \forall i \in \{1, 2, \ldots, m\} \quad a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = b_i \\ & \forall j \in \{1, 2, \ldots, n\} \quad x_j \geq 0 \end{cases}$$

به‌طور شهودی حالت استاندارد، درواقع یافتن کمینه‌ی تابع هدف در دامنه‌ایست که به اشتراک mتا ابرصفحه در فضای$\mathbb{R}^n$ و با مؤلفه‌های نامنفی محدود شده‌است.

همانطور که از صورت فرمول‌ها برمی‌آید نوشتن آن‌ها به فرم ماتریسی می‌تواند خیلی سودمند باشد. پس قرار دهید:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n, \quad c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \in \mathbb{R}^n, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \in \mathbb{R}^m, \quad A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

با این تعاریف فرم استاندارد برنامه‌ریزی خطی را می‌توان به صورت زیر نوشت:

$$\begin{cases} \min & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{cases}$$

در فرم استاندارد فرض می‌شود که مؤلفه‌های A، b و c همگی ثابت‌اند و حالت متغیرتصادفی یا ... ندارند، همچنین فرض می‌شود که $rank(A) = m$ و $b \geq 0$.

۳.۱ تبدیل به فرم استاندارد

ثابت می‌کنیم می‌توان هر مسأله‌ی برنامه‌ریزی خطی را استاندارد کرد. اگر به‌طور کلی شرطی به صورت $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \geq b_i$ داشته باشیم می‌توان یک متغیر s_i اضافه کرد و شرط

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n - s_i = b_i, \quad s_i \geq 0$$

را جایگزین قبلی کرد. به چنین متغیری که اضافه می‌کنیم متغیر مازاد می‌گویند.
اگر هم شرطی به صورت $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i$ داشته باشیم می‌توان یک متغیر s_i اضافه کرد و شرط

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n + s_i = b_i, \quad s_i \geq 0$$

را جایگزین قبلی کرد. به چنین متغیری که اضافه می‌کنیم متغیر کمبود می‌گویند.
به‌طور کلی به متغیرهای مازاد و کمبود، متغیر سستی[۱۰] نیز می‌گویند.
پس با این کار می‌توانیم تمام شروطی که بزرگترمساوی یا کوچکترمساوی هستند را به تساوی تبدیل کنیم.
اگر مسأله بیشینه کردن تابع هدف بود مثلا بیشینه کردن c^Tx، می‌توانیم مسأله را با کمینه کردن $-c^Tx$ جایگزین کنیم چرا که می‌دانیم بیشینه کردن یک تابع معادل است با کمینه کردن قرینه‌ی آن تابع و اگر مثلا تابع $f(x)$ به ازای $x = x_0$ بیشینه شود آنگاه تابع $-f(x)$ دقیقا

[۱۰] slack variable

به ازای همان $x = x_\circ$ کمینه می‌شود. به زبان ریاضی $\max f(x) = -\min -f(x)$ و $argmax(f(x)) = argmin(-f(x))$.
پس می‌توانیم همواره مسأله را به یک مسأله‌ی کمینه سازی تبدیل کنیم.
تنها مشکل دیگری که در تبدیل فرم کلی به فرم استاندارد باقی می‌ماند کران‌های $l_i \le x_i \le u_i$ اند. برای تبدیل این کران‌ها به $x_i \ge \circ$ اگر هرکدام از l_i یا u_i عدد حقیقی بودند (یعنی $+\infty, -\infty$ نبودند) می‌توان شرط مورد نظر را همانند شروط قبلی بررسی کرد. مثلا فرض کنید u_i عددی حقیقی باشد. در این‌صورت شرط $x_i \le u_i$ را می‌توان به شکل زیر نوشت و آن را دقیقا به دید یکی از شرط‌های قبلی نگاه کرد.

$$\circ \times x_۱ + \cdots + ۱ \times x_i + \cdots + \circ \times x_n \le u_i$$

بنابراین کلا شرط $l_i \le x_i \le u_i$ را به شکل $x_i \le u_i$ و $l_i \le x_i$ و $-\infty \le x_i \le +\infty$ می‌نویسیم که دوتای اول بررسی شده‌اند. نهایتا باید شرط $-\infty \le x_i \le +\infty$ را بررسی کنیم.
پس فرض کنید متغیر x_i هم می‌تواند مقادیر مثبت و هم مقادیر منفی اتخاذ کند. برای هر عدد حقیقی مثل a داریم:

$$a = \max\{a, \circ\} - \max\{-a, \circ\}$$

به $\max\{a, \circ\}$ ترم مثبت a می‌گویند و با علامت $[a]^+$ نشان می‌دهند و به $\max\{-a, \circ\}$ ترم منفی a می‌گویند و با علامت $[a]^-$ نشان می‌دهند. دقت کنید که $[a]^+, [a]^- \ge \circ$. درواقع:

$$[a]^+ = \max\{a, \circ\} = \begin{cases} a & a \ge \circ \\ \circ & a < \circ \end{cases} \qquad [a]^- = \max\{-a, \circ\} = \begin{cases} \circ & a > \circ \\ -a = |a| & a \le \circ \end{cases}$$

نکته‌ی مهم دیگر درباره‌ی نمادگذاری بالا این است که $|a| = [a]^+ + [a]^-$، که کمک می‌کند قدرمطلق را حذف کنیم.
بنابراین می‌توانیم متغیر x_i را به صورت $x'_i - x''_i$ بنویسیم که $x'_i = [x_i]^+$ و $x''_i = [x_i]^-$ متغیرهای نامنفی‌اند. پس در حالتی که x_i آزاد در علامت باشد نیز می‌توان با تعریف متغیرهای جدید، شرط آزاد بودن x_i را حذف کرد.
این روش شاید بهترین روش برای تبدیل به فرم استاندارد نباشد اما اثبات می‌کند که هرفرم کلی قابل تبدیل به فرم استاندارد خواهد بود.

۴.۱ چند مثال

مثال: مسأله‌ی برنامه‌ریزی خطی زیر را به فرم استاندارد تبدیل کنید.

$$\begin{cases} \min & ۲x_۱ + ۵x_۲ \\ \text{s.t.} & x_۱ + x_۲ \ge ۶ \\ & -x_۱ - ۲x_۲ \ge -۱۸ \\ & x_۱, x_۲ \ge \circ \end{cases}$$

ابتدا می‌خواهیم شرط $b \ge \circ$ محقق شود برای این منظور طرفین رابطه‌ی $-x_۱ - ۲x_۲ \ge -۱۸$ را در -۱ ضرب می‌کنیم. حال می‌خواهیم $x_۱ + x_۲ \ge ۶$ و $x_۱ + ۲x_۲ \le ۱۸$ را به تساوی تبدیل کنیم. در رابطه‌ی $x_۱ + x_۲ \ge ۶$ متغیر جدیدی مثل $x_۳$ تعریف می‌کنیم که نمایانگر فاصله‌ی $x_۱ + x_۲$ از ۶ است و شرط $x_۱ + x_۲ \ge ۶$ را با $x_۱ + x_۲ - x_۳ = ۶$ جایگزین می‌کنیم. مشابها برای شرط دوم یعنی $x_۱ + ۲x_۲ \le ۱۸$ یک متغیر جدید مثل $x_۴$ تعریف می‌کنیم و این شرط را با $x_۱ + ۲x_۲ + x_۴ = ۱۸$ جایگزین می‌کنیم. نهایتا به مسأله‌ی برنامه‌ریزی خطی استاندارد زیر می‌رسیم.

$$\begin{cases} \min & ۲x_۱ + ۵x_۲ \\ \text{s.t.} & x_۱ + x_۲ - x_۳ = ۶ \\ & x_۱ + ۲x_۲ + x_۴ = ۱۸ \\ & x_۱, x_۲, x_۳, x_۴ \ge \circ \end{cases}$$

⊠

مثال: همان مثال قبل را در نظر بگیرید فقط به‌جای $\min ۲x_۱ + ۵x_۲$ قرار دهید $\max ۲x_۱ + ۵x_۲$ و این مسأله را به فرم استاندارد تبدیل کنید.

کافی‌است همان فرم استاندارد مثال قبل را در نظر بگیریم و فقط به جای $\max ۲x_۱ + ۵x_۲$ قرار دهیم $\min -۲x_۱ - ۵x_۲$ یعنی :

$$\begin{cases} \min & -۲x_۱ - ۵x_۲ \\ \text{s.t.} & x_۱ + x_۲ - x_۳ = ۶ \\ & x_۱ + ۲x_۲ + x_۴ = ۱۸ \\ & x_۱, x_۲, x_۳, x_۴ \geq ۰ \end{cases}$$

⊠

مثال: مسأله‌ی برنامه‌ریزی خطی زیر را به فرم استاندارد تبدیل کنید.

$$\begin{cases} \max & -x_۱ + |x_۲| + ۲x_۳ \\ \text{s.t.} & x_۱ - x_۲ + ۳x_۳ = -۴ \\ & x_۱ - x_۳ \leq ۲ \\ & x_۲ + x_۳ \leq -۳ \\ & x_۱ \geq ۰, x_۳ \leq ۰ \end{cases}$$

شروط اول و سوم را باید در -۱ ضرب کرد تا سمت راست آن‌ها مثبت شود. شرط $x_۳ \leq ۰$ با تعریف $x'_۳ = -x_۳$ استاندارد می‌شود. برای حذف قدرمطلق در $|x_۲|$ تعریف می‌کنیم $x_۲ = x'_۲ - x''_۲$، که $x'_۲ = [x_۲]^+, x''_۲ = [x_۲]^-$. بیشینه سازی را هم باید به کمینه سازی تبدیل کنیم پس ضرایب تابع هدف قرینه می‌شوند. برای شروط دوم و سوم هم باید متغیرهای مازاد و کمبود تعریف کنیم. با این توضیحات فرم استاندارد شده‌ی مسأله بالا به این‌صورت خواهد بود:

$$\begin{cases} \min & x_۱ - x'_۲ - x''_۲ + ۲x'_۳ \\ \text{s.t.} & -x_۱ + x'_۲ - x''_۲ + ۳x'_۳ = ۴ \\ & x_۱ + x'_۳ + x_۴ = ۲ \\ & -x'_۲ + x''_۲ + x'_۳ - x_۵ = ۳ \\ & x_۱, x'_۲, x''_۲, x'_۳, x_۴, x_۵ \geq ۰ \end{cases}$$

⊠

اینکه در تبدیل به فرم استاندارد تعداد متغیرها زیاد می‌شوند از لحاظ جبری مشکلی ندارد اما وقتی تعداد متغیرها کمتر باشد شهود هندسی مسأله راحت‌تر است مثلا در مثال اول فرم استاندارد شده در فضای $\mathbb{R}^۴$ است که تجسم آن سخت است اما فرم غیراستاندارد آن را می‌توان به راحتی در $\mathbb{R}^۲$ رسم کرد.

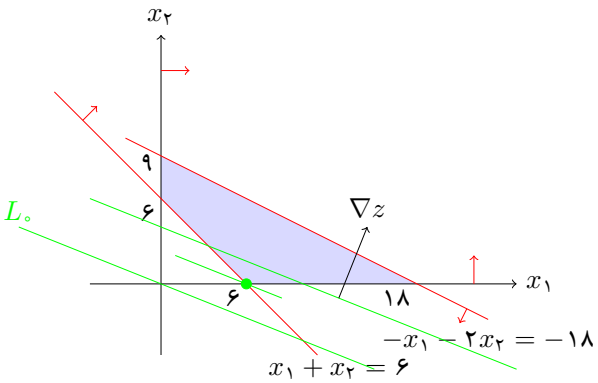
مثال: مسأله‌ی برنامه‌ریزی خطی زیر را حل کنید.

$$\begin{cases} \min & ۲x_۱ + ۵x_۲ \\ \text{s.t.} & x_۱ + x_۲ \geq ۶ \\ & -x_۱ - ۲x_۲ \geq -۱۸ \\ & x_۱, x_۲ \geq ۰ \end{cases}$$

همان‌طور که گفته‌شد می‌خواهیم برای این مسأله نمودار رسم کنیم. برای رسم این نمودار ابتدا خطوط $x_۱+x_۲ = ۶$ و $-x_۱-۲x_۲ = -۱۸$ را رسم می‌کنیم سپس برای اینکه بفهمیم ناحیه‌ی $x_۱ + x_۲ \geq ۶$ بالای خط است یا پایین آن یک نقطه از نمودار مثلا $(۰, ۰)$ را در رابطه قرار می‌دهیم و می‌بینیم آیا در رابطه صدق می‌کند یا نه، اگر صدق کرد ناحیه‌ی مورد نظر همان سمتی است که $(۰, ۰)$ در آن قرار دارد وگرنه ناحیه‌ی موردنظر طرف دیگر است (در شکل با فلش قرمز مشخص شده‌است). مشابها برای $-x_۱ - ۲x_۲ \geq -۱۸$ همین کار را انجام می‌دهیم.

شرط‌های $x_۱, x_۲ \geq ۰$ هم بیان می‌کنند که باید ناحیه را به بالای محور $x_۱$ و سمت راست محور $x_۲$ محدود کنیم. پس نهایتا ناحیه‌ی رنگ شده مجموعه مقادیر مجاز برای $(x_۱, x_۲)$ است و اصطلاحا به این ناحیه، ناحیه‌ی شدنی[۱۱] می‌گویند و آن را با Ω نشان می‌دهند، یعنی مجموعه تمام نقاطی که شرط‌های مسأله را برآورده می‌کنند.

$$\Omega = \{x \mid x_۱ + x_۲ \geq ۶, \quad -x_۱ - ۲x_۲ \geq -۱۸, \quad x_۱ \geq ۰, \quad x_۲ \geq ۰\}$$



می‌بینیم که در این مثال ناحیه‌ی شدنی کران‌دار است. برای یافتن کمینه‌ی $۲x_۱ + ۵x_۲$ در این ناحیه یک منحنی تراز برای آن، مثلا $L_۰$ را درنظر می‌گیریم

$$L_۰ = \left\{ \begin{bmatrix} x_۱ \\ x_۲ \end{bmatrix} \in \mathbb{R}^۲ \mid ۲x_۱ + ۵x_۲ = ۰ \right\}$$

این منحنی یک خط در $\mathbb{R}^۲$ است و به سادگی دیده می‌شود که منحنی‌های تراز $۲x_۱ + ۵x_۲$ خطوط موازی‌اند که با خطوط سبز در شکل مشخص شده‌اند. حال به جهت گرادیان تابع نگاه می‌کنیم (گرادیان در تمام نقاط یکسان است).

$$\nabla z = \begin{bmatrix} \frac{\partial(۲x_۱+۵x_۲)}{\partial x_۱} \\ \frac{\partial(۲x_۱+۵x_۲)}{\partial x_۲} \end{bmatrix} = \begin{bmatrix} ۲ \\ ۵ \end{bmatrix} = c$$

اگر در راستای منفی گرادیان حرکت کنیم مقدار تابع کم می‌شود بنابراین برای یافتن کمینه‌ی تابع با توجه به شکل باید پایین‌ترین منحنی تراز

[۱۱] Feasible region

تابع را بیابیم که با ناحیه‌ی شدنی برخورد دارد و مقدار x^* که تابع در آن کمینه شود پیدا می‌شود.

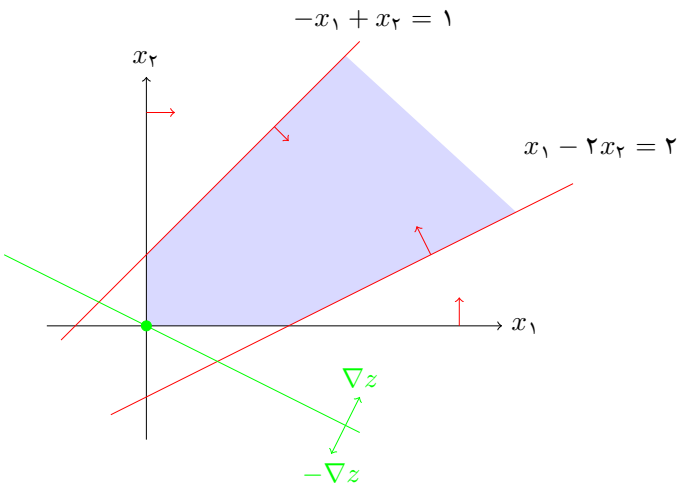
$$x^* = \begin{bmatrix} ۶ \\ ۰ \end{bmatrix}, \quad z^* = z(۶, ۰) = ۲ \times ۶ + ۵ \times ۰ = ۱۲$$

⊠

مثال: مسأله‌ی برنامه‌ریزی خطی زیر را حل کنید.

$$\begin{cases} \min & x_۱ + ۲x_۲ \\ \text{s.t.} & -x_۱ + x_۲ \leq ۱ \\ & x_۱ - ۲x_۲ \leq ۲ \\ & x_۱, x_۲ \geq ۰ \end{cases}$$

چون مسأله کمینه سازی است پس باید در جهت مخالف با گرادیان حرکت کرد که با توجه به شکل پایین‌ترین منحنی ترازی که با ناحیه‌ی شدنی برخورد دارد این ناحیه را در $(۰, ۰)$ قطع می‌کند بنابراین جواب بهینه به ازای $(۰, ۰)$ رخ می‌دهد.

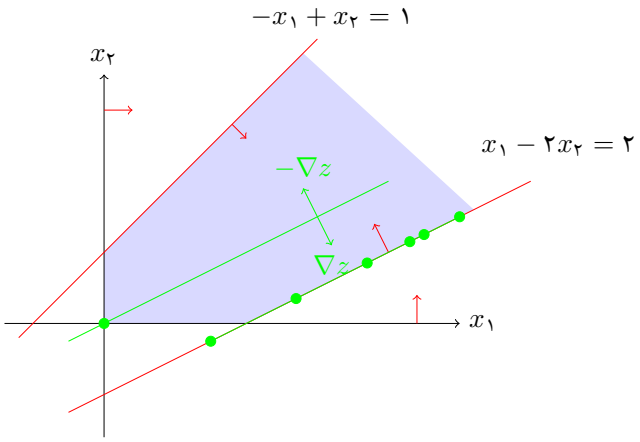


⊠

مثال: مسأله‌ی برنامه‌ریزی خطی زیر را حل کنید.

$$\begin{cases} \min & -x_۱ + ۲x_۲ \\ \text{s.t.} & -x_۱ + x_۲ \leq ۱ \\ & x_۱ - ۲x_۲ \leq ۲ \\ & x_۱, x_۲ \geq ۰ \end{cases}$$

در این مثال نیز باید در جهت مخالف با گرادیان حرکت کرد که با توجه به شکل پایین‌ترین منحنی ترازی که با ناحیه‌ی شدنی برخورد دارد این ناحیه را در بی‌نهایت نقطه روی خط $x_۱ - ۲x_۲ = ۲$ قطع می‌کند بنابراین جواب بهینه به ازای هر نقطه روی خط $x_۱ - ۲x_۲ = ۲$ با مؤلفه‌های نامنفی رخ می‌دهد.

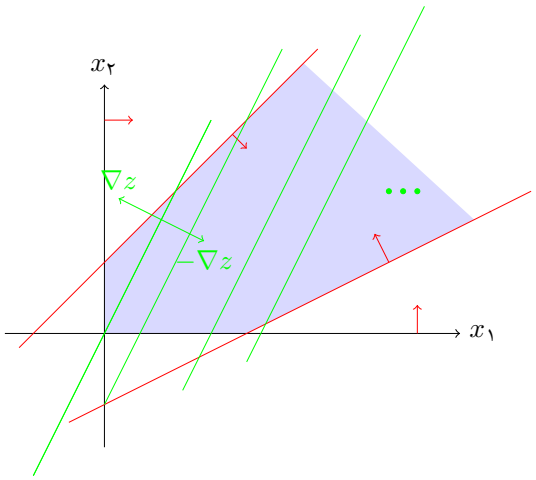


⊠

مثال: مسأله‌ی برنامه‌ریزی خطی زیر را حل کنید.

$$\begin{cases} \min & -۲x_۱ + x_۲ \\ \text{s.t.} & -x_۱ + x_۲ \leq ۱ \\ & x_۱ - ۲x_۲ \leq ۲ \\ & x_۱, x_۲ \geq ۰ \end{cases}$$

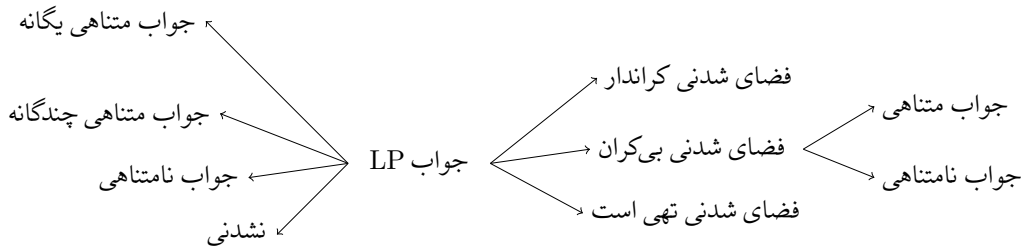
مشابها باید در جهت مخالف با گرادیان حرکت کرد اما با توجه به شکل می‌بینیم که تابع هدف روی ناحیه‌ی شدنی کمترین مقدار ندارد و هرچقدر درجهت مخالف با گرادیان حرکت کنیم همواره منحنی تراز با ناحیه‌ی شدنی برخورد دارد.



⊠

۵.۱ حالات مختلف برای جواب

با توجه به مثال‌هایی که دیدیم به‌طور کلی حالات زیر برای جواب یک مسأله‌ی برنامه‌ریزی خطی می‌تواند رخ دهد:



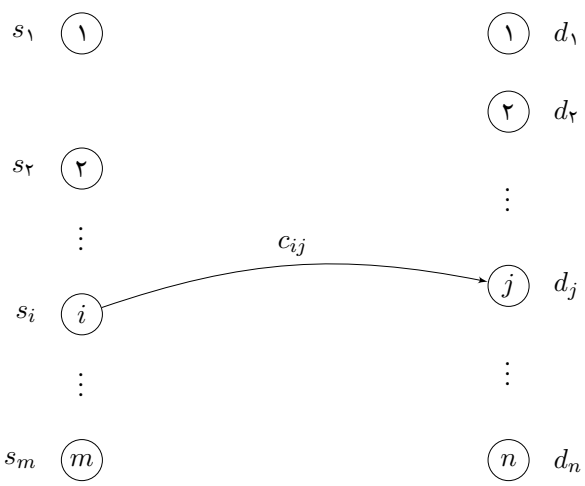
درمورد حالاتی که برای جواب یک مسأله‌ی برنامه‌ریزی خطی رخ می‌دهد:
جواب متناهی یگانه: به حالتی می‌گویند که تابع هدف کمینه (یا بیشینه) می‌شود و تنها یک نقطه‌ی x^* موجود است که به ازای آن، مقدار تابع هدف کمینه (یا بیشینه) می‌شود.
جواب متناهی چندگانه: به حالتی می‌گویند که تابع هدف کمینه (یا بیشینه) می‌شود اما بیش از یک نقطه مثل x^* وجود دارد که تابع هدف به ازای آن کمینه (یا بیشینه) می‌شود.
جواب نامتناهی: به حالتی می‌گویند که تابع هدف مقدار کمینه (یا بیشینه) که موردنظر است را ندارد و به هراندازه می‌توان آن را کوچک (یا بزرگ) کرد.
مسأله نشدنی: حالتی است که شروط مسأله با هم سازگاری ندارند و فضای شدنی درواقع تهی است (از این نوع، مثال ندیدم اما به سادگی می‌توان مثالی زد که این حالت رخ دهد).

درمورد حالاتی که برای فضای شدنی در یک مسأله‌ی برنامه‌ریزی خطی رخ می‌دهد:
فضای شدنی کراندار: به مفهوم آنالیزی یعنی یک گوی[۱۲] باز (یا بسته) حول نقطه‌ای از فضا موجود باشد که فضای شدنی زیرمجموعه‌ی این گوی باشد.
فضای شدنی بی‌کران: وقتی فضای شدنی کراندار و تهی نباشد. که در این حالت ممکن است جواب مسأله‌ی برنامه‌ریزی خطی متناهی یا نامتناهی شود.
فضای شدنی تهی: حالتی که شروط مسأله با هم سازگار نباشند و کلا هیچ نقطه‌ای مثل x موجود نباشد که در تمام شروط مسأله صدق کند.

۶.۱ کاربردی از برنامه‌ریزی خطی

مسأله‌های مختلفی را می‌توان با استفاده از برنامه‌ریزی خطی حل کرد. حتی بعضی از مسأله‌هایی که ماهیت خطی ندارند را می‌توان به صورت خطی نوشت و تقریبی از جواب بهینه برای آن‌ها بدست آورد. می‌خواهیم مسأله‌ی زیر را که به مسأله‌ی حمل‌ونقل معروف است به مسأله‌ی برنامه‌ریزی خطی تبدیل کنیم.
مثال: فرض کنید m مرکز عرضه[۱۳] داریم که میزان عرضه‌ی مرکز iام s_i است و همچنین n مرکز تقاضا[۱۴] داریم که میزان تقاضای مرکز iام d_i است. فرض می‌کنیم میزان هزینه‌ای که داده می‌شود تا یک واحد کالا از مرکز عرضه‌ی iام به مرکز تقاضای jام برسد برابر با c_{ij} است.
همچنین فرض می‌کنیم $\sum_{j=1}^{n} d_j \leq \sum_{i=1}^{m} s_i$ که حالت ایده‌آل این است که برابر باشند (اگر میزان تقاضا بیشتر از عرضه باشد به وضوح مشکلات زیادی پیش می‌آید. در حالتی هم که میزان عرضه بیشتر از تقاضا باشد ممکن است موجب ورشکستگی تعدادی از مراکز عرضه شود).
مسأله این است که برای هر $1 \leq i \leq m$ و $1 \leq j \leq n$، چه مقدار کالا از مرکز عرضه‌ی iام به مرکز تقاضای jام بفرستیم تا مراکز تقاضا حداقل به اندازه‌ی ظرفیتشان کالا دریافت کنند، هیچ مرکز عرضه‌ای بیش از ظرفیتش کالا ارسال نکند و درکل کمترین هزینه را داشته باشیم.

[۱۲] Ball
[۱۳] source
[۱۴] demand



متغیر x_{ij} را برابر با تعداد واحدهای کالایی که از مرکز عرضه‌ی iام به مرکز تقاضای jام می‌فرستیم، تعریف می‌کنیم. هزینه‌ای که این ارسال به ما تحمیل می‌کند $c_{ij}x_{ij}$ است. بنابراین هزینه‌ی کل برابر است با $z = \sum_{j=1}^{n}\sum_{i=1}^{m} c_{ij}x_{ij}$.

نیاز تقاضای مرکز jام باید برآورده شود یعنی باید داشته باشیم $\sum_{i=1}^{m} x_{ij} \geq d_j$.

همچنین مرکز عرضه‌ی iام حداکثر می‌تواند s_i کالا ارسال کند پس باید داشته باشیم $\sum_{j=1}^{n} x_{ij} \leq s_i$.

شرط نهایی این است که x_{ij}ها اعداد صحیح و نامنفی‌اند. البته این شرط که x_{ij}ها صحیح باشند را می‌توانیم وارد نکنیم و می‌توان بررسی کرد که این شرط خودبه‌خود محقق می‌شود (در جلسه ۲۶ به این مورد اشاره می‌شود).

بنابراین مسأله‌ی بالا به یک مسأله‌ی برنامه‌ریزی خطی به صورت زیر ترجمه می‌شود:

$$\begin{cases} \min & \sum_{j=1}^{n}\sum_{i=1}^{m} c_{ij}x_{ij} \\ \text{s.t.} & \sum_{i=1}^{m} x_{ij} \geq d_j \quad \forall j \\ & \sum_{j=1}^{n} x_{ij} \leq s_i \quad \forall i \\ & x_{ij} \geq 0 \quad \forall i,j \end{cases}$$

نکته‌ی قابل توجه در این مثال این است که اگر $\sum_{j=1}^{n} d_j = \sum_{i=1}^{m} s_i$، آنگاه فرم بالا خودبه‌خود درحالت استاندارد است یعنی برای هر $1 \leq i \leq m$ داریم $\sum_{j=1}^{n} x_{ij} = s_i$ و برای هر $1 \leq j \leq n$ داریم $\sum_{i=1}^{m} x_{ij} = d_j$. برای اثبات این موضوع داریم:

$$\forall 1 \leq j \leq n: \quad \sum_{i=1}^{m} x_{ij} \geq d_j \Rightarrow \sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij} \geq \sum_{j=1}^{n} d_j$$

$$\forall 1 \leq i \leq m: \quad \sum_{j=1}^{n} x_{ij} \leq s_i \Rightarrow \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} \leq \sum_{i=1}^{m} s_i$$

حال با برهان خلف فرض کنید مثلا یکی از نامساوی‌های $\sum_{i=1}^{m} x_{ij} \geq d_j$ اکید باشد و حالت تساوی رخ ندهد. در این‌صورت خواهیم داشت

$$\sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij} > \sum_{j=1}^{n} d_j = \sum_{i=1}^{m} s_i \geq \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} = \sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij} \Rightarrow \sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij} > \sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij}$$

که تناقض است. مشابها در تمام $\sum_{j=1}^{n} x_{ij} \leq s_i$ ها نیز باید تساوی رخ دهد. ⊠

۷.۱ نگاهی هندسی دیگر به مسأله

تا اینجا تمام مثال‌هایی که بررسی کردیم دارای دو متغیر بودند و تعبیر هندسی این مسأله‌ها راحت بود. در این قسمت نمایش هندسی دیگری برای مسأله‌هایی بیان می‌کنیم که تعداد متغیرهای آن‌ها بیشتر است.

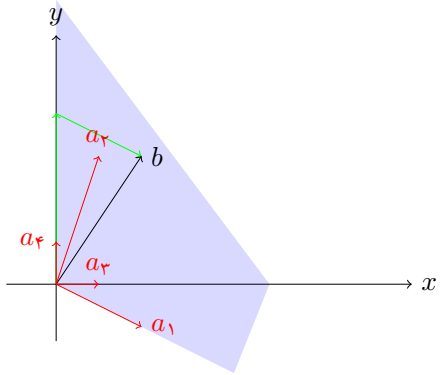
مثال: شروط زیر را درنظر بگیرید.

$$\begin{cases} ۲x_۱ + x_۲ + x_۳ = ۲ \\ -x_۱ + ۲x_۲ + x_۴ = ۳ \\ x_۱, x_۲, x_۳, x_۴ \geq ۰ \end{cases}$$

دو شرط اول را می‌توان به‌صورت ماتریسی نوشت:

$$x_۱ \begin{bmatrix} ۲ \\ -۱ \end{bmatrix} + x_۲ \begin{bmatrix} ۱ \\ ۳ \end{bmatrix} + x_۳ \begin{bmatrix} ۱ \\ ۰ \end{bmatrix} + x_۴ \begin{bmatrix} ۰ \\ ۱ \end{bmatrix} = \begin{bmatrix} ۲ \\ ۳ \end{bmatrix}$$

می‌خواهیم به این سؤال پاسخ دهیم که آیا می‌توان $\begin{bmatrix} ۲ \\ ۳ \end{bmatrix}$ را به صورت ترکیب‌خطی بردارهای $a_۱ = \begin{bmatrix} ۲ \\ -۱ \end{bmatrix}$ ، $a_۲ = \begin{bmatrix} ۱ \\ ۳ \end{bmatrix}$ ، $a_۳ = \begin{bmatrix} ۱ \\ ۰ \end{bmatrix}$ ، $a_۴ = \begin{bmatrix} ۰ \\ ۱ \end{bmatrix}$ با ضرایب نامنفی نوشت یا نه؟ برای پاسخ به این سؤال ۴ بردار ذکر شده را در صفحه‌ی $\mathbb{R}^۲$ رسم می‌کنیم و تمام نقاطی را می‌یابیم که به صورت ترکیب‌خطی نامنفی ۴ بردار ذکر شده قابل بیان‌اند.



با توجه به شکل می‌بینیم که $\begin{bmatrix} ۲ \\ ۳ \end{bmatrix}$ را می‌توان مثلا به صورت ترکیب‌خطی نامنفی از $a_۱ = \begin{bmatrix} ۲ \\ -۱ \end{bmatrix}$ و $a_۳ = \begin{bmatrix} ۱ \\ ۰ \end{bmatrix}$ یا $a_۴ = \begin{bmatrix} ۰ \\ ۱ \end{bmatrix}$ نوشت اما نمی‌توان آن را به صورت ترکیب‌خطی نامنفی از $a_۱ = \begin{bmatrix} ۲ \\ -۱ \end{bmatrix}$ و $a_۳ = \begin{bmatrix} ۱ \\ ۰ \end{bmatrix}$ نوشت.

⊠

مثال: مسأله‌ی برنامه‌ریزی خطی زیر را حل کنید.

$$\begin{cases} \min & z = -۲x_۱ - ۳x_۲ \\ \text{s.t.} & x_۱ + ۲x_۲ \leq ۲ \\ & x_۱, x_۲ \geq ۰ \end{cases}$$

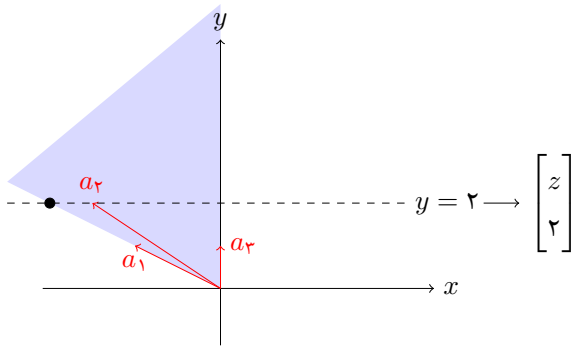
ابتدا مسأله را به فرم زیر می‌نویسیم و از تکنیک جدید که در بالا مطرح شد استفاده می‌کنیم.

$$\begin{cases} \min & z \\ \text{s.t.} & -۲x_۱ - ۳x_۲ = z \\ & x_۱ + ۲x_۲ + x_۳ = ۲ \\ & x_۱, x_۲, x_۳ \geq ۰ \end{cases}$$

به بیان ماتریسی

$$x_۱ \begin{bmatrix} -۲ \\ ۱ \end{bmatrix} + x_۲ \begin{bmatrix} -۳ \\ ۲ \end{bmatrix} + x_۳ \begin{bmatrix} ۰ \\ ۱ \end{bmatrix} = \begin{bmatrix} z \\ ۲ \end{bmatrix}$$

بنابراین، هدف یافتن کمترین مقدار برای z است به طوری که بردار $\begin{bmatrix} z \\ ۲ \end{bmatrix}$ را بتوان به‌صورت ترکیب‌خطی نامنفی از بردارهای $a_۱ = \begin{bmatrix} -۲ \\ ۱ \end{bmatrix}$, $a_۲ = \begin{bmatrix} -۳ \\ ۲ \end{bmatrix}$, $a_۳ = \begin{bmatrix} ۰ \\ ۱ \end{bmatrix}$ نوشت.



درشکل ناحیه‌ای که ترکیب‌خطی نامنفی سه بردار موردنظر است، رنگ شده‌است. انتهای تمام بردارهای به فرم $\begin{bmatrix} z \\ ۲ \end{bmatrix}$ روی خط $y = ۲$ قرار دارند. بنابراین باید سمت چپ‌ترین نقطه روی خط $y = ۲$ را بیابیم که با ناحیه رنگی اشتراک دارد (که معادل است با کمترین مقدار z). این نقطه در شکل مشخص شده که اگر بخواهیم آن را به صورت ترکیب خطی نامنفی از بردارهای ذکر شده بنویسیم، می‌بینیم ضریب $\begin{bmatrix} -۳ \\ ۲ \end{bmatrix}$ و $\begin{bmatrix} ۰ \\ ۱ \end{bmatrix}$ باید صفر باشد و ضریب $\begin{bmatrix} -۲ \\ ۱ \end{bmatrix}$ باید ۲ باشد. پس جواب بهینه به ازای $x_۱^* = ۲, x_۲^* = x_۳^* = ۰$ رخ می‌دهد. در این‌صورت مقدار کمینه‌ی z برابر است با -۴. ⊠

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس مهمان: هانی احمدزاده

[بهار ۹۹]

جلسه ۲۴: برنامه‌ریزی خطی ۲ نگارنده: امیرمحمد شعبانی

جلسه گذشته سعی کردیم با مفاهیم اولیه LP آشنا شویم و در این جلسه قصد داریم ابتدا مفاهیمی از جبر خطی و آنالیز محدب بیان کنیم و سپس وارد نظریه جواب LP شویم و در انتها سعی کنیم یک الگورتم برای پیدا کردن جواب LP پیدا کنیم.

۱ مفاهیم اولیه

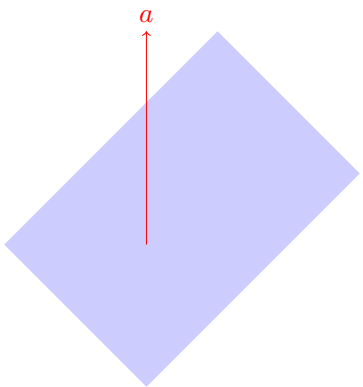
۱.۱ ابرصفحه

تعریف ابرصفحه گذرنده از نقطه $x_0 \in \mathbb{R}^n$ و با نرمال $a \in \mathbb{R}^n$:

$$a^T * (x - x_0) = 0$$

اگر $a^T x_0 = b$ بنابراین داریم:

$$a^T x = b \Rightarrow \sum_{j=1}^{n} a_j x_j$$



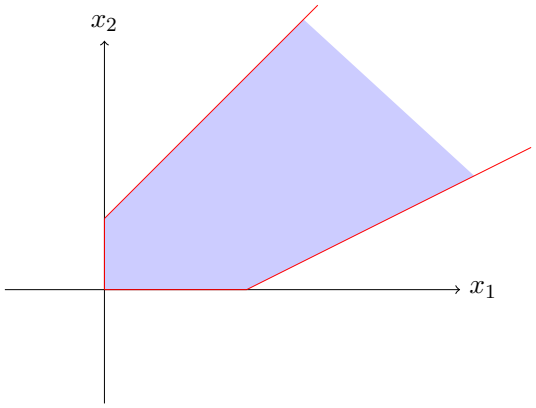
بنابراین هرقیدی در LP نشان‌دهنده یک ابرصفحه است و ناحیه شدنی‌ما اشتراک این ابرصفحه‌ها است. همچنین از لحاظ هندسی ناحیه‌شدنی تشکیل یک چندوجهی می‌دهد.

مثال: فضای R^2

☒

مثال: برای فضای R^3 می‌توان مکعب را مثال زد. ☒

مثال: همچنین برای ناحیه‌شدنی بی‌کران داریم:



☒

۲.۱ ترکیب خطی

تعریف: برای بردار های $S = \{a_1, a_2, \cdots, a_n\} \subset R^n$

$$\lambda_1 a_1 + \lambda_2 a_2 + \cdots + \lambda_n a_n \quad \lambda_j \in R$$

همچنین $Span(S)$ را فضای برداری تولید شده به وسیله ترکیب خطی بردارهای مجموعه S گویند.

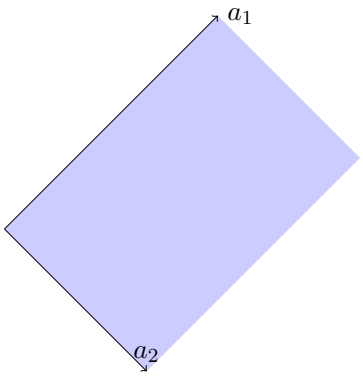
مثال: برای $S = \{a_1, a_2\}$ ، $Span(S)$ تشکیل یک صفحه در قضای R^2 می‌دهد.

☒

۱.۲.۱ ترکیب خطی نامنفی

$$\lambda_1 a_1 + \lambda_2 a_2 + \cdots + \lambda_n a_n \quad \lambda_j \geq 0$$

مثال: ترکیب خطی نامنفی دو بردار a_1 و a_2 :



☒

تعریف مخروط: $C \neq \varnothing \subset R^n$ را مخروط گویند هرگاه:(x یک بردار است)

$$x \in C \quad \lambda \geq 0 \Rightarrow \lambda x \in C$$

مثال: مثال بالا نماینده یک مخروط است. ☒

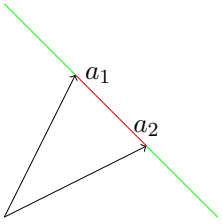
۲.۲.۱ ترکیب خطی آفین

$$\begin{cases} \lambda_1 a_1 + \lambda_2 a_2 + \cdots + \lambda_n a_n \\ \lambda_1 + \lambda_2 + \cdots + \lambda_n = 1 \end{cases}$$

مثال: برای دو بردار a_1 و a_2 ترکیب خطی آفین عبارتند از:

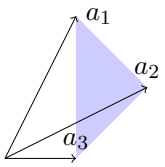
$$\begin{cases} \lambda_1 a_1 + \lambda_2 a_2 \\ \lambda_1 + \lambda_2 = 1 \Rightarrow \lambda_2 = 1 - \lambda_1 \end{cases} \Rightarrow a_2 + \lambda_1 (a_1 - a_2)$$

بنابراین نشان دهنده یک خط است که تحلیل هندسی آن عبارت است از:(کل رنگ قرمز و آبی شامل ترکیب خطی آفین است)



⊠

مثال: برای ۳ بردار نیز داریم: (صفحه شامل مثلث آبی ، ترکیب خطی آفین است.)

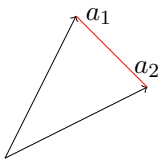


⊠

۳.۲.۱ ترکیب خطی محدب

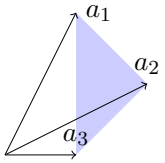
$$\begin{cases} \lambda_1 a_1 + \lambda_2 a_2 + \cdots + \lambda_n a_n \\ \lambda_1 + \lambda_2 + \cdots + \lambda_n = 1 \\ \lambda_1, \lambda_2, \cdots, \lambda_n \geq 0 \end{cases}$$

مثال: برای دو بردار a_1 و a_2 ترکیب خطی محدب عبارتند از:(خط قرمز ترکیب خطی مورد نظر است.)



⊠

مثال: برای ۳ بردار نیز داریم: (فقط مثلث آبی ، ترکیب خطی محدب است.)



⊠

۳.۱ مجموعه محدب

مجموعه $\varnothing \neq C \subset R^n$ را محدب گوییم هرگاه :

$$\forall x, y \in C \quad \forall \lambda \in [0,1] \rightarrow \lambda x + (1-\lambda) y \in C$$

مثال: دایره یک مجموعه محدب است. ⊠

نکته: فضای هر مساله برنامه‌ریزی خطی یک مجموعه محدب است.

اثبات. مجموعه $\Omega = \{x \in R^n | Ax = b, x \geq 0\}$ را فضای‌شدنی برنامه‌ریزی خطی LP فرض کنید. باید بگوییم:

$$\begin{cases} \forall \bar{x}, \hat{x} \in \Omega \\ \lambda \in [0,1] \end{cases} \Rightarrow \lambda\bar{x} + (1-\lambda)\hat{x} \in \Omega$$

برای اثبات باید ۲ شرط را چک کنیم:

شرط ۱: $Ax = b$

$$\begin{cases} \bar{x} \in \Omega \Rightarrow A\bar{x} = b, \bar{x} \geq 0 \\ \hat{x} \in \Omega \Rightarrow A\hat{x} = b, \hat{x} \geq 0 \end{cases} \Rightarrow A(\lambda\bar{x} + (1-\lambda)\hat{x}) = \lambda A\bar{x} + (1-\lambda)A\bar{x} = b$$

شرط ۲: $\lambda\bar{x} + (1-\lambda)\hat{x} \geq 0$

به وضوح مشخص است که جمع دو عدد بزرگتر از صفر ، مثبت خواهد شد.

بنابراین اثبات کردم که ناحیه‌شدنی تشکیل یک مجموعه محدب می‌دهد.

□

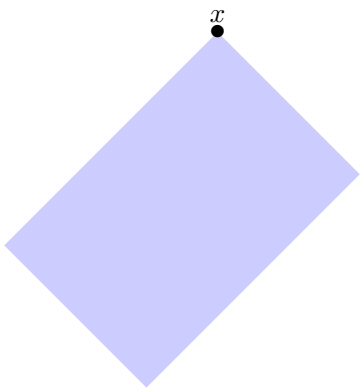
۴.۱ نقطه راسی(نقطه گوشه‌ای)

برای مجموعه محدب $\varnothing \neq C \subset R^n$ ، $x \in C$ را نقطه راسی یا گوشه‌ای نامیم هرگاه:

$$\begin{cases} \nexists \bar{x}, \hat{x} \in C \quad \bar{x} \neq \hat{x} \\ \lambda \in [0,1] \end{cases} \rightarrow x = \lambda\bar{x} + (1-\lambda)\hat{x}$$

به عبارتی x درون پاره خط واصل بین هیپ دو نقطه متمایزی از فضای‌شدنی قرار نداشته باشد.

مثال: نقطه x یک نقطه راسی در فضای شدنی زیر است.



⊠

سوال: آیا دایره نقطه راسی دارد؟

جواب: تمام نقاط روی محیط دایره یک نقطه راسی به حساب می‌آیند.

۵.۱ جهت شدنی

تعریف: d را جهت شدنی در نقطه $\bar{x}$ در فضای‌شدنی $\Omega = \{x \in R^n | Ax = b, x \geq 0\}$ گوییم هرگاه :

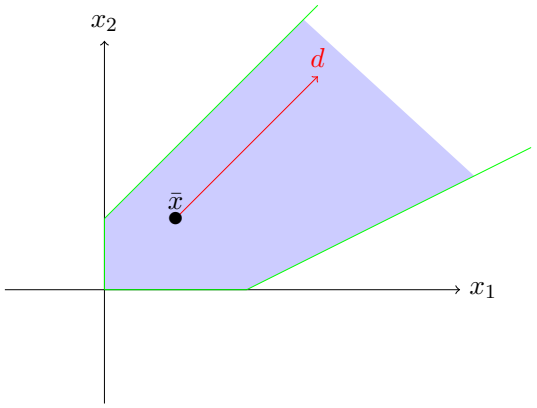
$$\exists \alpha > 0 \quad , \quad \bar{x} + \alpha d \in \Omega$$
$$\Rightarrow A(\bar{x} + \alpha d) = b \Rightarrow Ad = 0$$

۶.۱ جهت دورشونده

تعریف: d را جهت دورشونده در نقطه $\bar{x}$ در فضای‌شدنی $\Omega = \{x \in R^n | Ax = b, x \geq 0\}$ گوییم هرگاه :

$$\forall \alpha > 0 \quad , \quad \bar{x} + \alpha d \in \Omega$$

مثال: d را یک جهت دورشونده گوییم.



⊠

نکته: شرط لازم و کافی برای وجود چنین جهت دورشونده‌ای ، وجود فضای‌شدنی بی‌کران است.

نکته: جهت دورشونده به نقطه وابسته نیست و تنها باید دو شرط را دارا باشد:

شرط ۱: $Ax = b$

مانند جهت‌های شدنی برای قرار گرفتن در فضای‌شدنی باید دارا باشد.

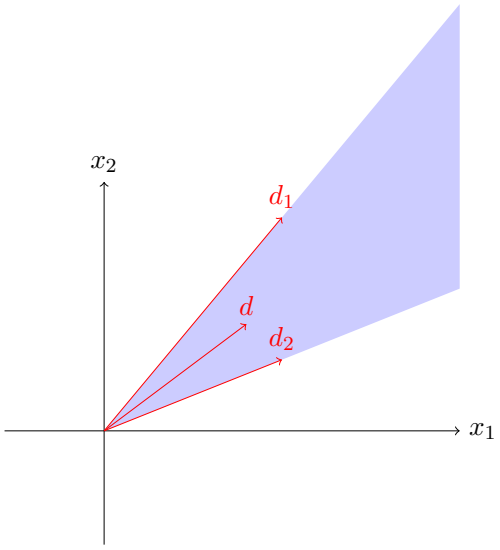
شرط ۲: $\bar{x} + \alpha d \geq 0, \quad \forall \alpha \geq 0$

و این اتفاق تنها زمانی خواهد افتاد که $d \geq 0$ زیرا در غیر اینصورت اگر α را در حد کافی بزرگ بگیریم نقطه‌ای خارج از محدود شرطمان خواهد افتاد.

۷.۱ جهت دورشونده راسی

تعریف: جهت دورشونده‌ای که ترکیب محدب دو جهت دورشونده دیگر نباشد.

مثال: در این مثال d_1 و d_2 جهت دورشونده راسی هستند اما d نیست.



⊠

حالکه تمام مفاهیم اولیه بیان شده‌است، زمان آن است که به سراغ قضیه‌ای که در طراحی الگوریتم حل LP به ما کمک می‌کند برویم.

۲ قضیه نمایش

۱.۲ قضیه نمایش برای چندوجهی کراندار

فرض کنید چندوجهی کراندار به صورت $\varnothing \neq P = \{x | Ax = b, x \geq 0\}$ باشد. آنگاه مجموعه نقاط راسی چندوجهی کراندار P ناتهی و متناهی است. اگر $x_1, x_2, \cdots, x_p$ را نقاط راسی P بنامیم آنگاه:

$$\begin{cases} \forall x \in P \\ \exists \lambda_1, \cdots, \lambda_p \end{cases} \rightarrow \begin{cases} x = \sum_{j=1}^{p} \lambda_j x_j \\ \sum_{j=1}^{p} \lambda_j = 1 \\ \lambda_j \geq 0 \end{cases}$$

۲.۲ قضیه نمایش برای چندوجهی بی‌کران

فرض کنید چندوجهی بی‌کران به ضورت $\varnothing \neq P = \{x | Ax = b, x \geq 0\}$ باشد. آنگاه مجموعه نقاط راسی چندوجهی کراندار P ناتهی و متناهی است. همچنین جهت‌های(دورشونده) راسی چندوجهی P ناتهی و متناهی است.اگر $x_1, x_2, \cdots, x_k$ را نقاط راسی P بنامیم و $d_1, d_2, \cdots, d_p$ جهت‌های راسی p آنگاه:

$$\begin{cases} \forall x \in P \\ \exists \lambda_1, \cdots, \lambda_k \\ \exists \mu_1, \cdots, \mu_p \end{cases} \rightarrow \begin{cases} x = \sum_{j=1}^{k} \lambda_j x_j + \sum_{j=1}^{p} \mu_j d_j \\ \sum_{j=1}^{p} \lambda_j = 1 \\ \lambda_j, \mu_j \geq 0 \end{cases}$$

۳.۲ کاربرد قضیه نمایش در حل LP

اگرچه شاید مساله LP اینگونه بنظر برسد که ماهیت پیوسته دارد اما این پیوستگی تنها در حرکت کردن بر روی آن ، کار را آسان می‌کند. در واقعا نقطه بهینه مساله LP تنها امکان دارد در نقاط گوشه‌ای وجود داشته باشد.(در صورتی که نقطه بهینه بی‌کران نباشد)

برای تحلیل دقیق‌تر مساله را به دو بخش بی‌کران و کراندار تقسیم کرده و بررسی می‌نماییم:

۱.۳.۲ حل Lp چندوجهی کراندار با استفاده از قضیه نمایش

فرض کنید مساله LP زیر را داریم:

$$\begin{cases} \min c^T x \\ \text{s.t.} \\ \quad Ax = b \\ \quad x \geq 0 \end{cases}$$

و ناحیه‌شدنی کراندار که تشکیل چندوجهی می‌دهد به صورت زیر باشد: $\varnothing \neq \Omega = \{x | Ax = b, x \geq 0\}$ که $x_1, \cdots, x_k$ نقاط راسی ω است. طبق قضیه نمایش داریم:

$$\forall x \in \Omega \rightarrow \begin{cases} x = \sum_{j=1}^{k} \lambda_j x_j \\ \sum_{j=1}^{p} \lambda_j = 1 \\ \lambda_j \geq 0 \end{cases} \Rightarrow \begin{cases} \min c^T (\sum_{j=1}^{k} \lambda_j x_j) = \min \sum_{j=1}^{k} \lambda_j c^T x_j \\ \sum_{j=1}^{p} \lambda_j = 1 \\ \lambda_j \geq 0 \end{cases}$$

در اینصورت کافیست کمینه عبارت‌های $c^T x_j$ را به عنوان جواب خروجی دهیم ؛ زیرادر ترکیب محدب درحال انجام میانگین گیری هستیم(به طور مثال بین دو نقطه x_1 و x_2 و برای همین مقدار آن همیشه بین مقدار خود این دو نقطه است.).

نکته: این جواب یکتا نیست! ممکن است دو نقطه x_1 و x_2 هردو جواب‌های بهینه و یکسانی داشته‌باشند. آنگاه هردو جواب مساله هستند. به طور کلی ترکیب محدب این دو نقطه جواب مساله است.

۲.۳.۲ حل Lp چندوجهی بی‌کران با استفاده از قضیه نمایش

فرض کنید مساله LP زیر را داریم:

$$\begin{cases} \min c^T x \\ \text{s.t.} \\ \quad Ax = b \\ \quad x \geq 0 \end{cases}$$

و ناحیه‌شدنی کراندار که تشکیل چندوجهی می‌دهد به صورت زیر باشد: $\varnothing \neq \Omega = \{x | Ax = b, x \geq 0\}$ که $x_1, \cdots, x_k$ نقاط راسی و $d_1, \cdots, p$ جهت‌های راسی ω است. طبق قضیه نمایش داریم:

$$\forall x \in \Omega \rightarrow \begin{cases} x = \sum_{j=1}^{k} \lambda_j x_j + \sum_{j=1}^{p} \mu_j d_j \\ \sum_{j=1}^{p} \lambda_j = 1 \\ \lambda_j, \mu_j \geq 0 \end{cases} \Rightarrow \begin{cases} \min c^T (\sum_{j=1}^{k} \lambda_j x_j + \sum_{j=1}^{p} \mu_j d_j) = \min \sum_{j=1}^{k} \lambda_j c^T x_j + \sum_{j=1}^{p} \mu_j c^T \\ \sum_{j=1}^{p} \lambda_j = 1 \\ \lambda_j, \mu_j \geq 0 \end{cases}$$

در اینصورت اگر $1 \leq j \leq p$ که $c^T d_j < 0$ آنگاه مساله جواب بی‌کران دارد ؛ زیرا اگر $\mu_j \rightarrow \infty$ آنگاه $c^T \rightarrow \infty$. در اینصورت شرط لازم برای جواب کراندار c^T به ازای هر j است. حال اگر شرط لازم برای داشتن جواب کراندار را دارا بود ، به دلیل مثبت بودن $c^T d_j$ برای مینیمم سازی بهتر است که $\mu_j = 0$ باشد و اینگونه باقی مراحل مانند حل LP برای چندوجهی کراندار است.

۳.۳.۲ جمع‌بندی

مساله برنامه ریزی استاندارد زیر را در نظر بگیرید.

$$\begin{cases} \min c^T x \\ \text{s.t.} \\ \quad Ax = b \\ \quad x \geq 0 \end{cases}$$

فضای‌شدنی آن $\varnothing \neq \Omega = \{x | Ax = b, x \geq 0\}$ است. حالت های زیر برای جواب LP به وجود می‌آید:

۱) $\Omega = \varnothing$ که نتیجه می‌گیریم مساله نشدنی است.(قیدها سازگاری ندارند.)

۲) $\Omega \neq \varnothing$ ولی کراندار که نتیجه می‌گیریم جهت دورشونده نداریم و حداقل یک نقطه راسی داریم. حتما یک نقطه راسی بهینه وجود دارد.

۳) $\Omega \neq \varnothing$ ولی بی‌کران که نتیجه می‌گیریم حداقل یک جهت راسی و حداقل یک نقطه راسی دارد.

حالت ۱: وجود دارد $1 \leq j \leq p$ که $c^T < 0$ که در اینصورت جواب بهینه نامتناهی است.

حالت ۲: در غیر اینصورت حتما یک نقطه راسی بهینه وجود دارد.

۳ الگوریتم حل برنامه ریزی خطی

$$\begin{cases} \min c^T x \\ \text{s.t.} \\ \quad Ax = b \\ \quad x \geq 0 \end{cases}$$

۱.۳ گام اول

اگر $\Omega = \varnothing$ بود ، متوقف شود و تهی چاپ کند. (قیدها ناسازگاری دارند)

۲.۳ گام دوم

نقاط راسی $x_1, \cdots, x_k$ و جهت های راسی $d_1, \cdots, d_p$ (درصورت بی‌کران بودن فضای‌شدنی در غیر اینصورت مجموعه جهت‌های راسی تهی است) محاسبه کنید.

۳.۳ گام سوم

اگر $c^T < 0$ برای برخی $1 \leq j \leq p$ وجود دارد، مساله جواب نامتناهی دارد و متوقف شود.

۴.۳ گام چهارم

$\min c^T x_j$ را به عنوان جواب بهینه چاپ کند و متوقف شود.

الگوریتم بالا ایده اولیه الگوریتم $simplex$ است.

۴ جلسه آینده

در جلسه آینده روند شهودی الگوریتم بالا را کامل کرده و در مورد فرآیند محاسبه نقاط راسی ارائه می‌دهیم. همچنین نقاط پایه‌شدنی را تعریف می‌کنیم و سپس ادعا می‌کنیم هر نقطه راسی یک پایه‌شدنی است. همچنین اگر زمان کافی وجود داشت به طور مختصر درمورد نظریه دوگان صحبت خواهیم کرد.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس مهمان: هانی احمدزاده

[بهار ۹۹]

جلسه ۲۵: برنامه‌ریزی خطی[1] ۳ نگارنده: سید پوریا فاطمی

در این جلسه ابتدا به سراغ قضیه‌ی مربوط به جواب پایه‌ای[2] رفته و سپس مسئله دوگان[3] را تعریف می‌کنیم و سعی می‌کنیم به کمک مسئله دوگان کرانی از جواب مسئله‌ی اصلی[4] بدست آوریم و در آخر شرایط یکسان بودن جواب مسئله اصلی و دوگان را بدست می‌آوریم.

۱ جواب پایه‌ای

یک فرض اضافی به مسئله اضافه می‌کنیم و آن این‌که: $rank(A) = m$ یعنی فرض می‌کنیم ماتریس ضرایب رتبه کامل[5] سطری باشد. این فرض محدودکننده نیست زیرا اگر رنک A عدد m نباشد و مسئله نشدنی[6] نباشد حتما می‌توانیم قیدهای اضافه(سطرهای غیر مستقل خطی از A) را حذف کنیم تا به یک ماتریس ضرایب با تعداد سطر کمتر و مستقل خطی برسیم. پس داریم:

$$rank(A) = m \rightarrow \text{ماتریس } A \text{ دارای } m \text{ ستون مستقل خطی است.}$$

دقت کنید که A یک ماتریس $m \times n$ است یعنی n ستون دارد که هر ستون آن m درایه دارد.

مجموعه $\{1, 2, ..., n\}$ را مجموعه اندیس‌های ماتریس A در نظر بگیرید. فرض کنید مجموعه $\beta \subset \{1, 2, ..., n\}$ یک مجموعه m عضوی باشد. حال تعریف می‌کنیم: $B = A_\beta$ به این معنا که ماتریس B ماتریسی شامل ستون‌های ماتریس A می‌باشد که اندیس‌های آن در مجموعه β قرار داد.

مجموعه $\mathcal{N} = \{1, 2, ..., n\} \backslash \beta$ را تعریف کنید و ماتریس N را بگیرید ستون‌هایی از ماتریس A که اندیس‌های آن در مجموعه $\mathcal{N}$ قرار دارد. $N = A_\mathcal{N}$

پس در واقع توانستیم ماتریس A را به دو قسمت تقسیم کنیم. ستون‌هایی از A که در B قرار دارد و بقیه ستون‌ها در N: $A \leftrightarrow [B \quad N]$
هم‌چنین داریم:

$$x \leftrightarrow \begin{bmatrix} x_B \\ x_N \end{bmatrix}$$

ماتریس B را یک پایه نامیم هرگاه $det(B) \neq 0$ (ستون‌های B مستقل خطی باشند.)

حال متناظر با این پایه انتخابی داریم:

$$Ax = b \rightarrow Bx_B + Nx_N = b$$

[1] Linear Programming
[2] Basic solution
[3] Dual
[4] Primal
[5] Full rank
[6] Infeasible

علت درستی عبارت این است که هر سطر از x در ستون متناظر در ماتریس A ضرب می‌شود و در ادامه با ضرب کردن طرفین مساوی در B^{-1} داریم:

$$x_B = B^{-1}b - B^{-1}Nx_N$$

پس هر جواب از دستگاه $Ax = b$ به صورت زیر نوشته می‌شود:

$$\begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} B^{-1}b - B^{-1}Nx_N \\ x_N \end{bmatrix}$$

به این نکته دقت کنید که ممکن است ترتیب درایه‌های بردار x در محاسبات بالا همان ترتیب اولیه نباشد.(با توجه به روش ساخت x_B و x_N) رابطه بالا را به شکل زیر می‌توانیم بازنویسی کنیم:

$$\begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} B^{-1}b - B^{-1}Nx_N \\ x_N \end{bmatrix} = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix} + \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} x_N$$

برای هر $x_N \in \mathbb{R}^{n-m}$ جواب به شکل بالا از دستگاه $Ax = b$ حاصل میشود.

متناظر به پایه B، جواب پایه با قرار دادن $x_N = 0$ حاصل میشود.

$$x \leftrightarrow \begin{cases} x_B = B^{-1}b \\ x_N = 0 \end{cases}$$

اگر $B^{-1}b \geqslant 0$ جواب x بالا را یک جواب پایه‌ای شدنی[۷] می‌نامیم.

قضیه ۱. *هر جواب پایه‌ای شدنی یک نقطه راسی است.*

اثبات. برای آنکه نشان دهیم نقطه راسی است باید نشان دهیم به صورت ترکیب محدب اکید از دو نقطه متمایز چندوجهی نمی‌توانیم نقطه مربوطه را بنویسیم.

پس B را یک ماتریس پایه از A چنان در نظر بگیرید که $B^{-1}b \geqslant 0$ پس ما جواب پایه‌ای به صورت

$$x : \begin{cases} x_B = B^{-1}b \\ x_N = 0 \end{cases}$$

داریم. حال فرض کنید $\bar{x}, \widetilde{x} \in P, \lambda \in (0,1)$ (منظور از P همان فضای شدنی مسئله است.) چنان یافت شود که $x = \lambda\bar{x} + (1-\lambda\widetilde{x})$ نوشته شده است. پس داریم:

$$\rightarrow \begin{cases} x_B = \lambda\overline{x_B} + (1-\lambda)\widetilde{x_B} \\ x_N = \lambda\overline{x_N} + (1-\lambda)\widetilde{x_N} \end{cases}$$

دقت کنید که $\overline{x_N}$ و $\widetilde{x_N}$ هر دو بزرگتر مساوی صفر هستند زیرا هر دو عضو چند وجهی P هستند و λ و $(1-\lambda)$ نیز بزرگتر از صفر هستند. ولی با توجه به جواب پایه‌ای می‌دانیم $x_N = 0$ است. پس هر دو بردار موجود رابطه بالا برای تشکیل x_N، صفر اند (چون ثابت کردیم هر دو نامنفی اند.) پس داریم:

$$\rightarrow \begin{cases} \lambda\overline{x_N} = 0 \\ (1-\lambda)\widetilde{x_N} = 0 \end{cases} \xrightarrow{\lambda \in (0,1)} \overline{x_N} = \widetilde{x_N} = 0$$

[7] Basic feasible solution

$$A\bar{x} = B\overline{x_B} + N\overline{x_N} = B\overline{x_B} = b \to \overline{x_B} = B^{-1}b \to \bar{x} \leftrightarrow \begin{cases} \overline{x_B} = B^{-1}b \\ \overline{x_N} = 0 \end{cases} \to \bar{x} = x$$

به طور مشابه $\widetilde{x} = x$.

یعنی ثابت کردیم اگر نقطه پایه‌ای شدنی x بخواهد به صورت ترکیب محدب اکید دو نقطه از فضای شدنی باشد آن دو نقطه همان x اند. پس x حتما نقطه راسی است. □

قضیه ۲. *برای هر نقطه راسی حداقل یک ماتریس پایه B چنان موجود است که نقطه پایه‌ای شدنی نظیر به B، همان نقطه راسی است.*

اثبات. قضیه بالا را به صورت دقیق اثبات نمی‌کنیم فقط به این نکته توجه کنید که هر نقطه راسی در کنج افتاده است پس حتما در فضای $\mathbb{R}^n$ به تعداد n ابرصفحه از آن می‌گذرد.در واقع داریم که از هر نقطه راسی حتما n ابر صفحه مستقل خطی عبور می‌کند.

m تا از این صفحات از دستگاه $Ax = b$ می آید و $n - m$ تا از این ابرصفحه‌ها باید از $x \geqslant 0$ باشد. پس $n - m$ تا از مولفه‌های x باید مساوی با صفر باشند. پس mتا درایه از x **می تواند** غیر صفر باشد.
ستون‌های نظیر این m درایه را در ماتریس A در نظر بگیرید. به کمک این ستون‌ها می‌توان B یا همان ماتریس پایه را شکل داد. پس داریم:

$$\begin{cases} Ax = b \\ x \geqslant 0 \end{cases} \to \begin{cases} Bx = b \\ x \geqslant 0 \end{cases} \to \begin{cases} x = B^{-1}b \\ B^{-1}b \geqslant 0 \end{cases}$$

پس هر نقطه راسی با توجه به نکات بالا حتما یه نقطه پایه‌ای شدنی می‌شود. □

پس تا الان فهمدیم نقاط راسی همان نقاط پایه‌ای شدنی هستند و نقاط پایه‌ای شدنی قابل محاسبه هستند و جواب بهینه در نقطه راسی اتفاق می‌افتد.

حال اگر جواب مسئله بی‌کران بود باید چه کار کنیم؟ در بالا داشتیم:

$$x \leftrightarrow \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix} + \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} x_N$$

که قسمت اول با شرط $B^{-1}b \geqslant 0$ همان نقطه پایه‌ای شدنی ماست و قسمت دوم یک ماتریس $n \times (n - m)$ است که در یک بردار ضرب شده است. پس ستون jام ماتریس

$$\begin{bmatrix} -B^{-1}N \\ I \end{bmatrix}$$

را در نظر بگیرید. که به صورت

$$d_j = \begin{bmatrix} -y_j \\ e_j \end{bmatrix}$$

است.

قضیه ۳. *اگر $d_j \geqslant 0$ باشد آن‌گاه d_j یک جهت دورشونده است.*

اثبات. مقادیر زیر را در نظر بگیرید:

$$\begin{cases} x_j = \theta \\ x_k = 0 \quad \forall k \in \mathcal{N} \backslash \{j\} \end{cases} \xrightarrow{\text{با فرض درست کردن ترتیب درایه ها}} x(\theta) = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix} + \begin{bmatrix} -y_j \\ e_j \end{bmatrix} \theta$$

پس داریم:

$$\begin{cases} Ax(\theta) = b \\ x(\theta) \geqslant 0 \qquad \forall \theta \geqslant 0 \end{cases}$$

پس هر θ مثبتی $x(\theta)$ ای را به ما می‌دهد که در فضای شدنی وجود دارد و طبیعتا d_j یک جهت دورشونده است. □

حال اگر $c^T d_j < 0$ باشد مسئله جواب بی‌کران دارد. زیرا:

$$c^T x(\theta) = \underbrace{c^T \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}}_{\text{مقدار ثابت}} + \theta \underbrace{c^T d_j}_{\text{منفی}} \xrightarrow{\theta \to \infty} -\infty$$

پس ما می‌توانیم جهت‌های دورشونده را نیز به کمک ماتریس پایه بدست آوریم. به صورت خلاصه الگوریتمی که می‌خواهیم اجرا کنیم (الگوریتم سیمپلکس[۸]) به صورت زیر است:

$$\text{ماتریس پایه شدنی } B \text{ را انتخاب کن} \longrightarrow x \leftrightarrow \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$$

حال می‌توان $c^T x$ را محاسبه کرد و از این نقطه x می‌توان به یک سری نقطه همسایه حرکت کنیم. برای این کار کافی است از روی ماتریس B ماتریس پایه $\bar{B}$ را طوری پیدا کنید که تنها در یک ستون با B متفاوت باشد. در واقع ماتریس $\bar{B}$ باید علاوه بر شرط تفاوت تنها یک سطر با B باید دو شرط زیر را نیز داشته باشد:

$$\begin{cases} det(\bar{B}) \neq 0 \\ \bar{B}^{-1}b \geqslant 0 \end{cases}$$

پس ما از یک نقطه راسی به نقطه راسی مجاور می‌رویم. این حرکت را به شرطی انجام می‌دهیم که مقدار $c^T x$ کاهش یابد. این فرآیند را آن‌قدر ادامه می‌دهیم تا بالاخره به نقطه‌ای برسیم که حرکت به نقطه‌های مجاور به ما مقدار $c^T x$ کمتری ندهد. توضیحات داده شده برای مسئله در حالت کران‌دار بود حال اگر جواب بی‌کران بود چه کنیم؟

می‌دانیم اگر جواب مسئله بی‌کران باشد برای ماتریس پایه‌شدنی B جهتی مانند d_j وجود دارد که $d_j \geqslant 0$ است به طوری که $c^T d_j < 0$. اگر در جایی از الگوریتم به این مورد برخوردیم الگوریتم را خاتمه می‌دهیم و می‌گوییم جواب مسئله بی‌کران است.(جزئیات پیاده‌سازی در یک ویدیو جداگانه توضیح داده خواهد شد.)

پس خلاصه الگوریتم سیمپلکس این بود که روی نقاط راسی حرکت می‌کنیم تا به نقطه راسی با کمترین مقدار $c^T x$ برسیم و نتیجه زیر را داریم:

در نقاط راسی حداکثر m مولفه ناصفر داریم.

[8] Simplex

۲ دوگانی

فرم متعارف مسئله برنامه‌ریزی خطی منیمم‌سازی:

$$\begin{cases} \min\ c^T x \\ s.t.\ \ Ax \geqslant b, x \geqslant 0 \end{cases}$$

فرم متعارف مسئله برنامه‌ریزی خطی ماکزیمم‌سازی:

$$\begin{cases} \max\ c^T x \\ s.t.\ \ Ax \leqslant b, x \geqslant 0 \end{cases}$$

هدف ما در دوگانی این است که به جای این که فرم متعارف مسئله برنامه‌ریزی مینیمم‌سازی را حل کنیم یک کران پایین برای جواب بهینه بدست آوریم.

فرض کنید فرم متعارف مسئله برنامه‌ریزی مینیمم‌سازی، شدنی[9] و دارای جواب بهینه x^* باشد. و داشته باشیم $z^* = c^T x^*$ برای $y \geqslant 0$ که $y \in \mathbb{R}^m$ است مسئله زیر را تعریف می‌کنیم:

$$\phi(y) = \min\{c^T x - y^T(Ax - b) | x \geqslant 0\}$$

قضیه ۴. *برای هر $y \geqslant 0$ داریم: $\phi(y) \leqslant z^*$*

اثبات. داریم:

$$z^* = \min\{c^T x | Ax - b \geqslant 0, x \geqslant 0\}$$

$$Ax - b \geqslant 0 \xrightarrow{y \geqslant 0} y^T(Ax - b) \geqslant 0 \to c^T x - y^T(Ax - b) \leqslant c^T x \quad \forall x \geqslant 0, Ax \geqslant b$$

و اگر از دو طرف نامساوی بالا مینیمم بگیریم به حکم می‌رسیم. □

اگر به تابع $\phi(y)$ دقت کنیم در خیلی از موارد مقدار منفی بی‌نهایت دارد و کران مفیدی به ما نمی‌دهد و داریم:

$$\phi(y) = \min_{x \geqslant 0} c^T - y^T(Ax - b) = \min_{x \geqslant 0} y^T b + (c^T - y^T A)x = y^T b + \min_{x \geqslant 0} (c^T - y^T A)x$$

اگر یکی از مولفه‌های بردار $c^T - y^T A$ منفی باشد درایه متناظر در بردار x را $+\infty$ قرار می‌دهیم و مقدار تابع $\phi(y)$ به $-\infty$ میل میکند. پس داریم:

$$\phi(y) = \begin{cases} -\infty & c^T - y^T A \ngeqslant 0 \\ y^T b & c^T - y^T A \geqslant 0 \end{cases}$$

پس از بین $y \geqslant 0$ تنها مواردی به ما کران مفید می‌دهند که $c^T - y^T A \geqslant 0$ و در این حالت کران پایین برای z^* مقدار $y^T b$ است. حال برای این‌که کران پایین بهتری برای مسئله بدهیم می‌خواهم مقدار این کران پایین را بیشینه کنیم و مسئله زیر را تعریف می‌کنیم و آن را مسئله دوگان متناظر با مسئله اصلی می‌نامیم:

$$P: \begin{cases} \min\ c^T x \quad x \in \mathbb{R}^n \\ s.t.\ \ Ax \geqslant b \ \leftarrow y \\ \qquad x \geqslant 0 \end{cases} \implies D: \begin{cases} \max\ b^T y \quad y \in \mathbb{R}^m \\ s.t.\ \ A^T y \leqslant c \\ \qquad y \geqslant 0 \end{cases}$$

[9] Feasible

پس دوگان یک مسئله متعارف مینیمم‌سازی یک مسئله متعارف ماکزیمم‌سازی است.
نکته بسیار مهم این‌که متغیرهای مسئله دوگان متناظر با قیدهای غیرکرانی مسئله‌ی اولیه هستند.

دو مجموعه زیر را تعریف می‌کنیم:

$$F_P = \{x | Ax \geqslant b, x \geqslant 0\} \qquad F_D = \{y | A^T y \leqslant c, y \geqslant 0\}$$

قضیه ۵. (*قضیه ضعیف دوگانی*[10]) *اگر* $x \in F_P, y \in F_D$ *آن‌گاه داریم:* $c^T x \geqslant b^T y$

اثبات.

$$x \in F_P : Ax \geqslant b, x \geqslant 0 \qquad y \in F_D : A^T y \leqslant c, y \geqslant 0$$

$$c \geqslant A^T y \rightarrow \begin{cases} c^T \geqslant y^T A \\ x \geqslant 0 \end{cases} \rightarrow \begin{cases} c^T x \geqslant y^T A x \\ \begin{cases} Ax \geqslant b \\ y \geqslant 0 \end{cases} \rightarrow y^T A x \geqslant y^T b \end{cases} \rightarrow c^T x \geqslant y^T b$$

□

نتایج زیر از قضیه ضعیف دوگانی بدست می‌آید:

نتیجه ۱. اگر $\bar{x} \in F_P, \bar{y} \in F_D$ چنان یافت شوند که $c^T \bar{x} = b^T \bar{y}$ آنگاه $\bar{x}$ بهینه برای P و $\bar{y}$ بهینه برای D است.

اثبات.

$$\forall x \in F_P, c^T x \geqslant b^T \bar{y} = c^T \bar{x} \Longrightarrow \bar{x} \text{ بهینه برای } P$$

$$\forall y \in F_D, b^T y \leqslant c^T \bar{x} = b^T \bar{y} \Longrightarrow \bar{y} \text{ بهینه برای } D$$

□

برای $x \in F_P, y \in F_D$ شکاف دوگانی[11] به صورت رو به رو تعریف می‌شود: $duality\_gap(x, y) = c^T x - b^T y$
پس نتیجه ۱ معادل آن است که در حالت بهینه شکاف دوگانی برابر صفر می‌شود.

نتیجه ۲. اگر یکی از مسئله‌های اولیه یا دوگان بی‌کران باشد، مسئله دیگر ناشدنی است.

اثبات. برهان خلف: فرض کنید P بی‌کران باشد و D شدنی باشد. داریم:

$$b^T y \leqslant c^T x \quad \forall\, x \in F_P$$

که این با بی‌کران بودن P در تناقض است زیرا کران پایین دارد. □

به طور خلاصه در مسئله‌ی مینیمم‌سازی می‌خواهیم مقدار $z = c^T x$ را کم کنیم و در مسئله‌ی ماکزیمم‌سازی(دوگان) می‌خواهیم مقدار $w = b^T y$ را افزایش دهیم که مقدار بهینه وقتی رخ می‌دهد که این دو مقدار مساوی شوند. حال به سراغ قضیه قوی دوگانی می‌رویم.

[10] Weak duality
[11] Duality-gap

قضیه ۶. (قضیه قوی دوگانی[۱۲]) *اگر $F_P \neq \varnothing, F_D \neq \varnothing$ آنگاه هر دو مسئله دارای جواب بهینه متناهی هستند که مقدار بهینه دو مسئله یکسان است در واقع داریم:*

$$\begin{cases} F_P \neq \varnothing \\ F_D \neq \varnothing \end{cases} \rightarrow \exists\, x^* \in F_P, y^* \in F_D \mid c^T x^* = b^T y^* \Rightarrow x^* \text{ بهینه برای اولیه و } y^* \text{ بهینه برای دوگان}$$

به طور خلاصه برای جمع بندی داریم: (جهت فلش ها نشان میدهد که هر کدام از حالات مسئله اصلی و دوگان در مسئله دیگر چه تاثیری دارد و چه نتیجه ای به ما می دهد.)

P	D
مینیمم سازی	ماکزیمم سازی
مینیمم سازی متعارف	ماکزیمم سازی متعارف
$\longrightarrow$ نامتناهی	ناشدنی
ناشدنی	نامتناهی $\longleftarrow$
$\longrightarrow$ شدنی متناهی	شدنی متناهی $\longleftarrow$
ناشدنی	ناشدنی

دقت کنید که در حالت شدنی متناهی جواب‌های بهینه دو مسئله یکسان هستند.

[12] Strong duality

۳ نوشتن مسئله دوگان

قاعده کلی به این صورت است که اگر فرم متعارف در یکی از طرفین رعایت‌شده باشد باید در طرف دیگر هم رعایت شود و اگر رعایت نشده بود نباید در طرف دیگر هم رعایت شود. قید مساوی در مسئله اصلی به متغیر با علامت آزاد در دوگان تبدیل میشود و متغیر آزاد در مسئله اصلی قید مساوی در حالت دوگان را می سازد. در حالت کلی داریم:

$$P: \begin{cases} \min\ c^T x & \\ s.t.\ a_i' x = b_i & i \in I_1 \to y_i \\ \quad a_i' x \geqslant b_i & i \in I_2 \to y_i \\ \quad a_i' x \leqslant b_i & i \in I_3 \to y_i \\ \quad x_j \geqslant 0 & j \in J_1 \\ \quad x_j \leqslant 0 & j \in J_2 \\ \quad x_j \text{ آزاد در علامت} & j \in J_3 \end{cases} \implies D: \begin{cases} \max\ \sum_i b_i y_i = b^T y & \\ s.t.\ (x_j \text{ ستون ضرایب})^T y \leqslant c_j & j \in J_1 \\ \quad (x_j \text{ ستون ضرایب})^T y \geqslant c_j & j \in J_2 \\ \quad (x_j \text{ ستون ضرایب})^T y = c_j & j \in J_3 \\ \quad y_i \text{ آزاد در علامت} & i \in I_1 \\ \quad y_i \geqslant 0 & i \in I_2 \\ \quad y_i \leqslant 0 & i \in I_3 \end{cases}$$

مثال:

$$P: \begin{cases} \min\ 2x_1 - 3x_2 + x_3 & \\ s.t.\ x_1 - x_2 + x_3 = 1 & \leftarrow y_1 \\ \quad -x_1 + 2x_2 + x_3 \leqslant -2 & \leftarrow y_2 \\ \quad 3x_1 + 2x_2 - 3x_3 \geqslant 1 & \leftarrow y_3 \\ \quad x_1 \text{ آزاد در علامت} & \\ \quad x_2 \geqslant 0 & \\ \quad x_3 \leqslant 0 & \end{cases} \implies D: \begin{cases} \max\ y_1 - 2y_2 + y_3 & \\ s.t.\ y_1 - y_2 + 3y_3 = 2 & \leftarrow x_1 \\ \quad -y_1 + 2y_2 + 2y_3 \leqslant -3 & \leftarrow x_2 \\ \quad y_1 + y_2 - 3y_3 \geqslant 1 & \leftarrow x_3 \\ \quad y_1 \text{ آزاد در علامت} & \\ \quad y_2 \leqslant 0 & \\ \quad y_3 \geqslant 0 & \end{cases}$$

به کمک همان قاعده کلی گفته‌شده می‌توان به سادگی این مثال را حل کرد.

برای تابع هدف که اعداد سمت راست مساوی‌ها و نامساوی‌ها را در متغیر تعریف‌شده قید متناظر ضرب می‌کنیم و جمع می‌بندیم.

برای قید اول دوگان چون x_1 به صورت علامت آزاد در مسئله اصلی ظاهر شده است قید مساوی است. برای x_2 چون فرم متعارف در مسئله اصلی رعایت شده است در مسئله دوگان نیز رعایت شده است و برای x_3 چون فرم متعارف در مسئله اصلی تعریف نشده است در مسئله دوگان نیز رعایت نشده است.

برای y_1 چون قید مربوطه در مسئله اصلی تساوی است در مسئله دوگان علامت y_1 آزاد است. برای قید متناظر با y_2 چون فرم معارف در مسئله اصلی رعایت نشده است در مسئله دوگان نیز رعایت نمی‌شود و برای y_3 نیز چون فرم متعارف در مسئله اصلی برای قید متناظر رعایت شده است در مسئله دوگان نیز رعایت می شود.

⊠

۴ چند قضیه تکمیلی

قضیه ۷. *دوگان دوگان مسئله ≡ مسئله اولیه*

قضیه بالا را اثبات نمی‌کنیم ولی در فرم متعارف ویژگی گفته‌شده را به راحتی می‌توان چک کرد. داریم:

$$\begin{cases} \min\ c^T x \quad x \in \mathbb{R}^n \\ s.t.\ \ Ax \geqslant b \ \leftarrow y \\ \qquad x \geqslant 0 \end{cases} \implies \begin{cases} \max\ b^T y \quad y \in \mathbb{R}^m \\ s.t.\ \ A^T y \leqslant c \ \leftarrow u \\ \qquad y \geqslant 0 \end{cases} \implies \begin{cases} \min\ c^T u \quad u \in \mathbb{R}^n \\ s.t.\ \ Au \geqslant b \\ \qquad u \geqslant 0 \end{cases}$$

همان‌طور که واضح است مسئله اول و سوم معادل‌اند.

نکته دیگر این‌که مسئله دوگان فرم استاندارد به شکل زیر است.

$$P: \begin{cases} \min\ c^T x \quad x \in \mathbb{R}^n \\ s.t.\ \ Ax = b \ \leftarrow y \\ \qquad x \geqslant 0 \end{cases} \implies D: \begin{cases} \max\ b^T y \quad y \in \mathbb{R}^m \\ s.t.\ \ A^T y \leqslant c \\ \qquad y \text{ آزاد در علامت} \end{cases}$$

قضیه ۸. *x جواب بهینه برای مسئله P است اگر و فقط اگر:*

$$\begin{cases} y \text{ چنان یافت شود که شدنی در دوگان باشد} \\ x \text{ شدنی در مسئله اولیه باشد} \\ c^T x = b^T y \end{cases}$$

مثلا برای فرم استاندارد داریم:

$$\begin{cases} A^T y \leqslant c \\ Ax = b, x \geqslant 0 \\ c^T x = b^T y \end{cases}$$

قضیه ۹. (***قضیه لنگی مکمل***[۱۳]) *x^*, y^* جواب‌های بهینه مسئله اولیه و دوگان هستند اگر و تنها اگر* ***هر متغیر ضرب در قید نظیرش صفر شود.*** *به‌عنوان مثال برای $\forall j = 1, 2, ..., n$ و $\forall i = 1, 2, ..., m$ داشته باشیم:*

در فرم استاندارد:

$$\begin{cases} x_j^*(c_j - a_j^T y^*) = 0 \\ y_i^*(a_i' x^* - b_i) = 0 \ \checkmark \end{cases}$$

در فرم متعارف:

$$\begin{cases} x_j^*(c_j - a_j^T y^*) = 0 \\ y_i^*(a_i' x^* - b_i) = 0 \end{cases}$$

[13] Complementary Slackness

اثبات. اگر فرض کنیم قضیه قوی دوگانی برقرار است این قضیه به سادگی اثبات می‌شود. در این‌جا برای فرم استاندارد اثبات می‌کنیم.

$$\left.\begin{cases} x^* \in F_P: & Ax^* = b, x^* \geqslant 0 \\ y^* \in F_D & A^T y^* \leqslant c \end{cases}\right\} \xrightarrow{\text{بهینگی دو نقطه}} c^T x^* = b^T y^*$$

$$\begin{aligned} (a^T b = b^T a \text{ می‌دانیم}) \quad &\iff c^T x^* = (Ax^*)^T y^* = y^{*T}(Ax^*) \\ &\iff (c^T - y^{*T} A)x^* = 0 \\ &\iff (c - A^T y^*)^T x^* = 0 \\ &\iff \sum_j \underbrace{(c_j - a_j^T y^*)}_{\geqslant 0} \underbrace{x_j^*}_{\geqslant 0} = 0 \\ (\text{تک‌تک مولفه‌ها صفر}) \quad &\iff (c_j - a_j^T y^*) x_j^* = 0 \quad \forall\, j = 1, 2, ..., n \end{aligned}$$

□

در واقع قضیه لنگی مکمل معادل با این است که شکاف دوگانی برابر صفر شود.

شرط لازم و کافی برای بهینگی به صورت زیر است:

$$KKT^{14}\ conditions\ for\ standard\ form: \begin{cases} Ax^* = b \\ x^* \geqslant 0 \\ A^T y^* \leqslant c \\ c^T x^* = b^T y^* \Leftrightarrow x_j^*(c_j - a_j^T y^*) = 0 \quad \forall\, j = 1, 2, ..., n \end{cases}$$

شرایط KKT پایه و اساس الگوریتم‌های حل مسئله LP هستند. مثلا الگوریتم‌های مختلف سیمپلکس موجود دو یا سه شرط از چهار شرط بالا را نگه می‌دارد و در راستای برقراری شرط یا شروط دیگر حرکت می‌کنند. روش نقطه درونی نیز شرط لنگی مکمل را به جای این که برابر با صفر بگذارد برابر با τ می‌گذارد و در هر مرحله سعی می‌کند دستگاه معادلات غیرخطی را به کمک روش نیوتن حل کند و در هر مرحله مقدار τ را کاهش دهد. در واقع روش‌های نقطه درونی برعکس روش سیمپلکس پرهیز می‌کنند از این که در مرز قرار داشته باشند.

الگوریتم سیمپلکس به دلیل آنکه ممکن است مجبور شود همه‌ی نقاط راسی را چک کند و تعداد نقاط راسی می‌تواند تا $\binom{n}{m}$ زیاد شود، اردر زمانی نمایی دارد ولی الگوریتم نقطه درونی با تعداد مراحل چندجمله‌ای جواب را بدست می‌آورد. در واقع روش‌های نقطه درونی از ماهیت پیوسته مسئله LP استفاده می‌کنند. به این‌صورت که از نقطه‌ای اکیدا شدنی شروع می‌کند و در مسیری به نام مسیر مرکزی[15] حرکت می‌کند و روی مسیری در داخل فضای شدنی (و نه مرز) به جواب بهینه می‌رسد.

می‌توان ثابت کرد روش نقطه درونی با تعداد مراحل چندجمله یک ϵ تقریب از جواب بهینه را برای ما برآورده می‌کند. البته روش‌های سیمپلکس کاربردهای خاص خودشان را دارند و در عمل تعداد محاسبات خیلی کمتری انجام می‌شود.

در واقع وقتی بعد مسئله بزرگ است و شما می‌خواهید فقط یک مسئله حل کنید روش نقطه درونی بهترین روش است ولی اگر بعد مسئله متوسط یا کوچک باشد و بخواهیم یک دنباله از مسائل LP را حل کنیم، آن وقت روش نقطه درونی هیچ مزیت نسبت به روش نقطه درونی ندارد و بهترین گزینه استفاده از روش سیملکس می باشد.

یک دسته دیگر از روش ها به نام روش‌های بیضوی[16] نیز وجود دارد که بیشتر جنبه تئوری دارند.

[14] Karush-Kuhn-Tucker
[15] Central path
[16] Ellipsoid method

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس مهمان: هانی احمدزاده

[بهار ۹۹]

جلسه ۲۶: برنامه ریزی خطی ۴ نگارنده: مهدی مستانی

در جلسات گذشته با برنامه ریزی خطی آشنا شدیم و شرایط بهینگی برای مسائل برنامه ریزی خطی مورد مطالعه قرار دادیم و مشاهده کردیم که نقاط راسی، کاندید های جواب بهینه هستند؛ بدین معنا که اگر مسئله جواب بهینه داشته باشد،حتما یک نقطه راسی بهینه دارد.بنابراین یافتن نقاط راسی ، استراتژی ما برای یافتن جواب بهینه مسئله بود.

در ادامه مسئله دوگانی را به طور مختصر مورد مطالعه قرار دادیم و مشاهده کردیم که برای هر مسئله برنامه ریزی خطی ، زوج مسئله(یا مسئله همزاد) که آن را مسئله دوگان می‌نامیم، وجود دارد که مقدار بهینه مسائل دوگان یکسان است که یکی از آن مسائل کمینه کردن و دیگری بیشنه کردن تابع هدف بود و ارتباط های این مسائل با یکدیگر را مورد بررسی قرار دادیم. سپس قضیه ضعیف و قوی دوگانی و قضیه لنگی مکمل را مورد بررسی قرار دادیم .

اما مسئله‌ای که همچنان وجود دارد این است که بعضی از متغیر های آن، تنها می‌توانند مقادیر صحیحی اختیار کنند که این شرایط منجر به تعریف دو مدل مسئله خواهد شد که تا حدی کاربردی بیشتر از مسائل برنامه ریزی خطی دارند که در ادامه قصد بررسی این نوع مسائل را داریم.

۱ برنامه‌ریزی عددصحیح

در این بخش قصد داریم که مسئله برنامه‌ریزی عدد صحیح [1] را بررسی کنیم. در این مسئله، هدف یافتن

$$\min c^T x + d^T y$$

$$s.t. \quad Ax + By = b$$

$$x \geq 0$$

$$y \geq 0$$

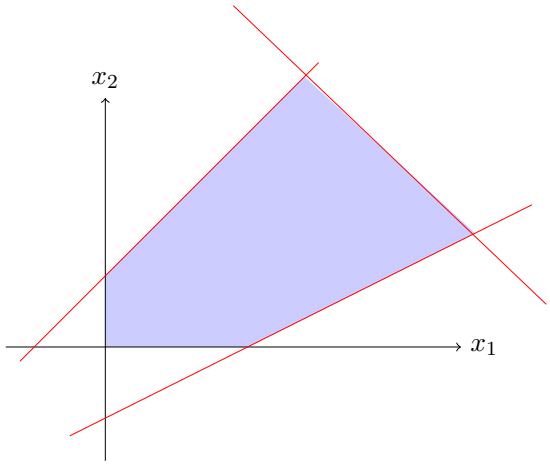
$$x \in \mathbb{R}^n, y \in \mathbb{Z}^k$$

است.در این مسئله ،هدف ما یافتن کیمنه [2] تابع خطی هدف است که در آن تابع را به دو قسمت تقسیم می‌کنیم؛ یک قسمت به صورت $c^T x$ است که متغیر x ، متغیری پیوسته است و قسمت دیگر آن $d^T y$ است که متغیر y ، متغیری است که قید عددصحیح بودن را دارا است.

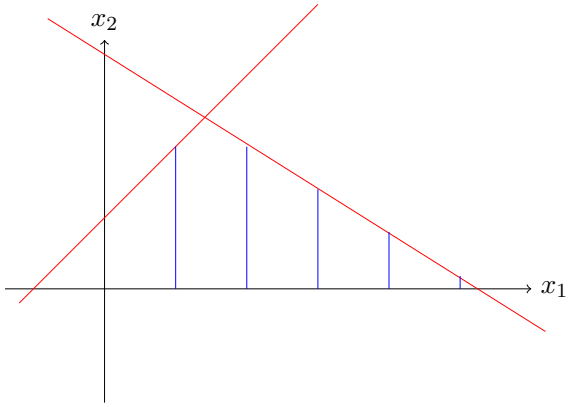
این مسئله بدین صورت است که برخی از متغیر های آن صحیح و برخی از متغیر های آن عددی حقیقی هستند.

[1] Mixed Integer Linear Programming(MILP)
[2] minimum

در قسمت های قبل دیدیم که در حالتی که همه متغیرها در فضای اعداد حقیقی باشند، فضای شدنی ما به صورت زیر است(یک چند وجهی):



اما اگر مثلا متغیر x_1 شرط صحیح بودن را داشته باشد $(x_1 \in \mathbb{Z})$ ، یعنی مثلا x_1 می‌تواند مقادیر $1, 2, 3, \ldots$ را اختیار کند . همینطور فرض کنید که متغیر x_2 ،متغیری است که مقادیر خود را در فضای حقیقی اختیار می‌کند. آنگاه فضای شدنی مسائل ما می‌تواند این شکل را داشته باشد:



که در آن خطوط آبی رنگ، فضای شدنی این مسئله است.

یک نوع دیگر این مسائل که آن را با IP [۳] یاد می‌کنیم ، بدین صورت است که در آن تمامی متغیر های ما مقادیر خود را در اعداد صحیح اختیار می‌کنند ؛ در واقع داریم:

$$\min c^T x$$

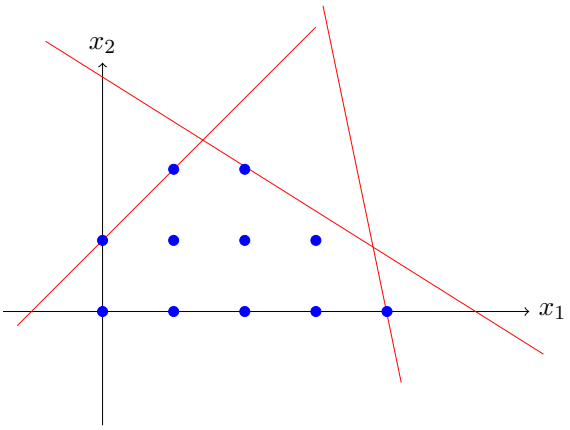
$$s.t. \quad Ax = b$$

$$x \geq 0$$

$$x \in \mathbb{Z}^n$$

[3] Integer Linear Programming

یک مثال از فضای شدنی(در اینجا نقاط شدنی هستند)، بدین صورت است.



که در آن نقاط آبی‌رنگ، نقاط(فضا) شدنی این مسئله است.(با فرض $x_1, x_2 \in \mathbb{Z}$)

پیوسته بودن مسئله برنامه ریزی خطی، کمک زیادی در روند حل مسئله به ما می‌کرد ؛ زیرا هنگامی که در یک نقطه قرار داشتیم ، می‌توانستیم به صورت پیوسته در ناحیه شدنی حرکت کنیم.اما هنگامی که فرض صحیح بودن را به مسئله اضافه می‌کنیم ، دیگر نمی توانیم به صورت پیوسته در ناحیه شدنی حرکت کنیم .در این مسئله مجبوریم که تمامی نقاط را پیدا کنیم و مقدار تابع هدف را در آن نقاط محاسبه و مقایسه کنیم. هدف ما در این قسمت این است که تا آنجا که امکان دارد ، حجم محاسبات را کاهش دهیم و نقاط اضافی را بررسی نکنیم .

قبل از بررسی حل این مسائل ، قصد داریم چند مدل برنامه ریزی عدد صحیح IP معروف را که کاربرد زیادی دارد را بررسی کنیم .

۲ مسئله کوله پشتی

همانطور که در جلسات قبل این مسئله را دیدیم، مسئله کوله‌پشتی [۴] بدین صورت است که کوله پشتی ای با ظرفیت W داریم و n کالا داریم که حجم کالا i ام برابر با w_i است($1 \leq i \leq n$). همینطور کالای شماره i ، دارای ارزش v_i است .($1 \leq i \leq n$) از هر یک از این کالا ، یک عدد موجود است و در صورت برداشتن کالای i ، این کالا حجم w_i از کوله پشتی را اشغال می‌کند(در صورت جا داشتن کوله پشتی) و ارزش v_i را به ما می‌دهد. همینطور این فرض را داریم که $\sum_{i=1}^{n} w_i > W$ (یعنی قادر به برداشتن همه این کالاها نیستیم). هدف ما این است که کالاهایی را برداریم که از حجم کوله پشتی بیشتر نشود و نیز بیشترین ارزش را به ما بدهند.

برای مدل سازی این مسئله ، متغیر تصمیم x_i را بدین صورت تعریف می‌کنیم($1 \leq i \leq n$):

$$x_i = \begin{cases} 1 & \text{اگر کالای شماره } i \text{ انتخاب شود} \\ 0 & \text{در غیر این صورت} \end{cases}$$

حال مسئله برنامه ریزی خطی عدد صحیح متناظر با این مسئله بدین صورت تعریف می‌شود:

$$\max \sum_{i=1}^{n} v_i x_i$$

$$s.t. \quad \sum_{i=1}^{n} w_i x_i \leq W$$

[4] Knapsack problem

$$x_i \in \{0,1\} \quad \forall i : 1 \leq i \leq n$$

این مسئله یک حالت خاص از مسئله برنامه ریزی عدد صحیح است که در آن متغیرها دودویی [۵] هستند.به این نوع مسائل ، مسائل برنامه‌ریزی دودویی [۶] نیز گویند.

این مسئله ، علاوه بر آنکه به تنهایی مسئله مهمی است، در حل بسیاری از مسائل ترکیبیاتی ، به عنوان زیر مسئله بررسی می‌شود.

۳ فروشنده دوره‌گرد

مسئله فروشنده دوره‌گرد [۷] را نیز قبلا بررسی کرده ایم.بدین صورت است که n شهر داریم که راه ارتباطی بین هر دو شهر موجود است. هزینه سفر از شهر i به شهر j برابر با c_{ij} است. هدف ما این است که از یک شهر شروع به حرکت کنیم و از هر شهر به شهر دیگری برویم و همه شهر ها را نیز ببینیم و به هر شهر ، حداکثر ۱ بار وارد شویم و در نهایت به شهر اولیه برگردیم و هزینه سفر نیز کمینه شود.

بیان مسئله به زبان گراف بدین صورت است که گراف کامل وزن دار داریم و در آن به دنبال دور همیلتونی [۸] با کمترین مجموع وزن ممکن هستیم.

حال مدل سازی مسئله ، داریم که متغیر تصمیم

$$x_{ij} = \begin{cases} 1 & \text{اگر یال } (i,j) \text{ انتخاب شود} \\ 0 & \text{در غیر این صورت} \end{cases}$$

در این صورت داریم که مسئله برنامه ریزی خطی ما به صورت زیر است:

$$\min \sum_{\substack{j=1 \\ i \neq j}}^{n} \sum_{i=1}^{n} c_{ij} x_{ij}$$

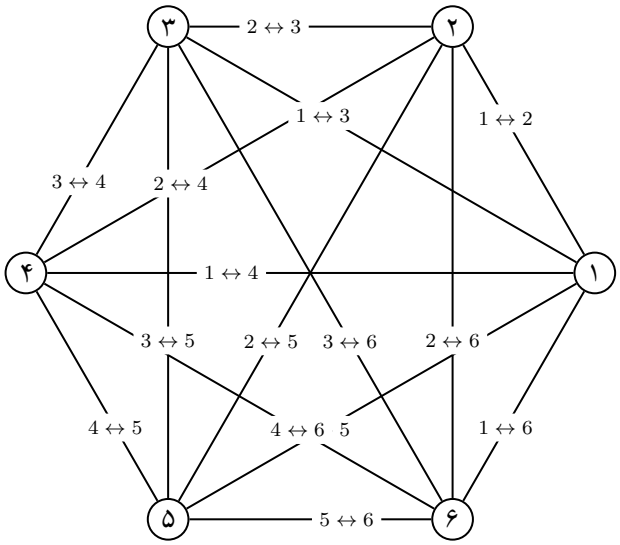
$$s.t. \quad \sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1 \quad \forall i : 1 \leq i \leq n$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1 \quad \forall j : 1 \leq j \leq n$$

$$x_{ij} \in \{0,1\}$$

اما تنها با این شرط ها ممکن است که یک زیر تور حاصل شود ؛ یعنی مثلا در گراف کامل ۶ راسی زیر

[5] binary
[6] Binary Linear Programming
[7] Traveling Saleman Problem (TSP)
[8] Hamiltonian Cycle



اگر داشته باشیم: $x_{12} = x_{23} = x_{31} = 1$ و همینطور $x_{45} = x_{56} = x_{64} = 1$ و سایر $x_{ij} = 0$ است. آنگاه داریم که این مقادیر خواسته سوال را برطرف می‌کند(به هر راس یکبار وارد و از آن یک بار خارج می‌شویم) ولی دور همیلتونی نیست.

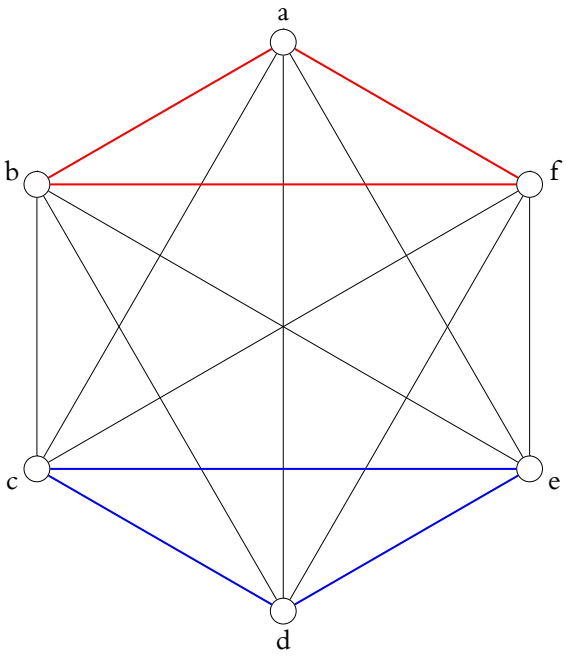
پس باید قید هایی را اضافه کنیم که مسئله ما، زیر تور ها را دیگر در نظر نگیرد. رویکرد های متفاوتی برای این قسمت داریم که یکی از آنها مربوط به Miller - Tucker - Zemlin است بدین صورت که برای راس های ۲ تا n ،متغیر u_i را برای راس i تعریف می‌کنیم بدین صورت که ($2 \le i, j \le n, i \ne j$) $u_i - u_j + nx_{ij} \le n - 1$ و ($2 \le i \le n$) $0 \le u_i \le n - 1$. در واقع مسئله بدبن صورت بیان می‌شود:

$$\begin{cases} \min \sum_{\substack{j=1 \\ i \ne j}}^{n} \sum_{i=1}^{n} c_{ij} x_{ij} \\ s.t. \quad \sum_{\substack{i=1 \\ i \ne j}}^{n} x_{ij} = 1 \quad \forall i : 1 \le i \le n \\ \sum_{\substack{i=1 \\ i \ne j}}^{n} x_{ij} = 1 \quad \forall j : 1 \le j \le n \\ x_{ij} \in \{0, 1\} \\ u_i - u_j + nx_{ij} \le n - 1 \quad 2 \le i, j \le n, i \ne j \\ 0 \le u_i \le n - 1 \quad 2 \le i \le n \end{cases}$$

اضافه کردن این شرط جدید باعث می‌شود که دیگر زپر تور نداشته باشیم. برای دیدن این مطلب فرض کنید که زیر تور داشته باشیم؛ پس حتما دوری وجود دارد که راس شماره ۱ را ندارد.(این دور را C بنامید).برای دور C همه قید های جدید را باز نویسی می‌کنیم:

$$u_i - u_j + n \le n - 1 \quad \forall i, j \in C, x_{ij} = 1$$

در اینجا برای هر راس i که در دور C است، یک بار به این راس وارد و یک بار خارج می‌شویم و یال این دور از راس i به j است و بنابراین $x_{ij} = 1$ است.



برای مثال در گراف کامل زیر:

اگر که دور آبی رنگ را در نظر بگیریم،داریم که ($x_{45} = x_{56} = x_{64} = 1$) و شرایط به صورت زیر است:

$$u_4 - u_5 + 6 \leq 5$$

$$u_5 - u_6 + 6 \leq 5$$

$$u_6 - u_4 + 6 \leq 5$$

که با جمع کردن طرفین این نامساوی ها به تناقض می‌خوریم($15 \leq 18$).

برای حالت کلی برای دور C به همین صورت، به تناقض می‌خوریم.

۴ مسئله محل تاسیسات

مسئله بعدی که آن را بررسی می‌کنیم ، مسئله محل تاسیسات [۹] است . بدین صورت که n تاسیسات [۱۰] داریم و m عدد مشتری [۱۱] داریم. باید به وسیله تاسیسات مان به مشتری‌ها خدمت‌رسانی کنیم .همینطور می‌دانیم که نیاز مشتری j ام ، مقدار d_j کالا نیاز دارد.

همینطور همه تاسیسات ها باز نیستند و بنابر نیاز ، باید آنها را ایجاد(باز) کنیم .بنابر این هزینه احداث تاسیسات i ام برابر با f_i است.

c_{ij} نیز هزینه ارسال کالا از تاسیسات i ام به مشتری j ام است. همینطور تاسیسات i ام حداکثر میزان تولید کالا دارد که آن را u_i می‌نامیم. در اینجا قصد داریم که این مسئله را با برنامه ریزی خطی مدل کنیم.

در اینجا دو نوع متغیر تصمیم داریم ؛ اول اینکه آیا تاسیسات i ایجاد شود یا خیر که آن را به صورت

[9] Facility Location Problem
[10] facility
[11] Cumstumer

$$x_i = \begin{cases} 1 & \text{تاسیسات } i \text{ ام ایجاد شود} \\ 0 & \text{در غیر این صورت} \end{cases}$$

متغیر تصمیم دوم نیز بدین صورت که چه مقدار از نیاز مشتری j ام توسط تاسیسات j ام تامین شود، یعنی داریم:

$= y_{ij}$ بخشی از نیاز مشتری j ام که توسط تاسیسات i ام تامین می‌شود.

پس با این تعاریف ،مدل شده مسئله بدین صورت است:

$$\begin{cases} \min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} d_j y_{ij} + \sum_{i=1}^{n} f_i x_i \\ s.t. \quad \sum_{i=1}^{n} y_{ij} = 1 \quad \forall j : 1 \le j \le m \\ \sum_{j=1}^{m} x_{ij} \le u_i x_i \quad \forall i : 1 \le i \le n \\ y_{ij} \ge 0 \quad 1 \le i \le n, 1 \le j \le m \\ x_{ij} \in \{0, 1\} \end{cases}$$

توضیحات مدل سازی: میخواهیم هزینه کل را کمینه کنیم ، که اگر تاسیسات i ام ایجاد شود ، باید هزینه f_i پرداخت شود که این متغیر در $\sum_{i=1}^{n} f_i x_i$ آمده است. اگر بخواهیم کسر y_{ij} تا از نیاز مشتری j ام توسط تاسیسات i ام تامین شود، هزینه $c_{ij} d_j y_{ij}$ را باید پرداخت کنیم. بنابر این قسمت $\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} d_j y_{ij}$ بیانگر این هزینه ها است.

قید $\sum_{i=1}^{n} y_{ij} = 1$ برای مشتری j ام بیان کننده این است که نیاز مشتری j باید تماما توسط همه n تاسیسات موجود ، تامین شود.

قید $\sum_{j=1}^{m} d_j y_{ij} \le u_i x_i$ برای متغیر i ام بیان کننده این است که اگر تاسیسات i ام باز شود(یعنی $x_i = 1$ باشد) آنگاه حداکثر می‌تواند به اندازه u_i کالا را تامین کند و $\sum_{j=1}^{m} d_j y_{ij}$ بیان کننده این است که این تاسیسات چه تعداد کالا از مشتری ها را تامین می‌کند(مقدار کالای تامین شده توسط تاسیسات i ام برای مشتری j ام برابر $d_j y_{ij}$ است).

در اینجا برخی از مدل ها بررسی شد. در ادامه رویکرد های لازم برای حل مسائل را بررسی می‌کنیم.

۵ رویکرد حل مسائل برنامه‌ریزی خطی عددصحیح

در قسمت های قبل برخی از مدل های برنامه‌ریزی عدد صحیح بررسی شد. در ادامه رویکرد های لازم برای حل این نوع مسائل را مورد بررسی قرار می‌دهیم.

در دسته ای مسائل IP نیازی به اعمال قید صحیح بودن نداریم. اگر که نقطه بهینه راسی چند وجهی $Ax = b, x \ge 0$ ، شرط صحیح بودن را داشته باشد ، می‌توانیم مسئله

$$\begin{cases} \min c^T x \\ s.t. \quad Ax = b \\ x \ge 0 \end{cases}$$

را حل کنیم و حل این مسئله به ما نقطه x^* را می‌دهد که این نقطه ، نقطه بهینه راسی است و شرط صحیح بودن را نیز دارد که نتیجه می‌دهد

این x^* جواب بهینه مسئله

$$\begin{cases} \min c^T x \\ s.t. \quad Ax = b \\ x \geq 0, x \in \mathbb{Z}^n \end{cases}$$

است.

اما باید توجه داشته باشیم که این این اتفاق اغلب رخ نمی دهد و در اکثر موارد جواب مسئله قید صحیح بودن آن را حذف کرده ایم ، جواب بهینه آن صحیح نیست. بنابراین اگر قید صحیح بودن مسئله را برداریم ، به مسئله حاصل ، مسئله آزاد سازی [۱۲] شده (RLP) [۱۳] مسئله بهینه سازی خطی عددصحیح است.

ممکن است که ادعا شود که ما مسئله RLP متناظر را حل می‌کنیم و جواب نهایی آن را اگر عدد صحیح نبود، رند می‌کنیم ؛ اما این روش جواب الزاما درستی را به ما نمی دهد و ممکن است جواب بهینه مقدار دیگری باشد.ممکن است که به تقریبی از جواب مسئله بهینه برسیم اما جواب دقیق آن را الزاما به دست نمی‌آوریم.همینطور این مشکل وجود دارد که اگر جواب مقدار صحیح نشد، به چه صورت رند کردن را انجام دهیم و امثال آن.

از آنجایی که مسئله IP فضای شدنی محدود تری نسبت فضای شدنی مسئله متناظر RLP دارد ، پس داریم که : $OPT(IP) \geq OPT(RLP)$.

حال می‌خواهیم بررسی کنیم که تحت چه شرایطی ،نقاط راسی چند وجهی می‌توانند شرط صحیح بودن را داشته باشند .

یادآوری می‌کنیم که مسئله ای که در حال بررسی آن هستیم به صورت:

$$\begin{cases} \min c^T x \\ s.t. \quad Ax = b \\ x \geq 0, x \in \mathbb{Z}^n \end{cases}$$

تعریف ۱. *ماتریس A را TU [۱۴] نامیم ، هرگاه دترمینان هر زیر ماتریس مربعی آن برابر با ± 1 یا 0 باشد.*

نتیجه ۱. *اگر ماتریس A ،ماتریسی TU باشد ، حتما درایه‌های آن ± 1 یا 0 است.*

اگر ماتریس ضرایب ما TU باشد،تمام نقاط راسی فضای شدنی، همگی صحیح خواهد بود؛ زیرا که نقطه راسی جتما متناظر با یک نقطه پایه‌ای شدنی بود و نقطه پایه ای شدنی بودن، متغیرهای پایه‌ای ما به صورت $x_B = B^{-1}b$ و متغیر های غیر پایه‌ای آن $x_N = 0$ است. ماتریس B زیرماتریسی از ماتریس A است $B^{-1} = \frac{1}{det(B)} adj(B)$ است که درایه های ماتریس $adj(B) = C^T$ که در آن درایه C_{ij} آن به‌صورت $C_{ij} = (-1)^{i+j} det(M_{ij})$ است که در آن ماتریس M_{ij} ماتریسی از که از حذف سطر i ام و ستون j ام ماتریس A حاصل می‌شود. اگر ماتریس A ماتریس TU باشد،در آنصورت $det(B) = 0$ یا ± 1 است که چون B وارون‌پذیر است ، پس باید $det(B) = \pm 1$ باشد و ماتریس $adj(B)$ درایه‌هایش مربوط به دترمیان تعدادی ماتریس که زیرماتریس B هستند که خود B هم زیرماتریسی ماتریس A است و دترمینان زیرماتریس‌های مربعی A نیز 0 یا ± 1 است . بنابراین B^{-1} ماتریسی صحیح است. اگر B نیز صحیح باشد، آنگاه تمام نقاط پایه ای شدنی، صحیح هستند.

نتیجه ۲. *اگر ماتریس A ،ماتریسی TU و $b \in \mathbb{Z}^m$ باشد ، آنگاه نقاط راسی چندوجهی حتما صحیح هستند.*

[12] relax

[13] Relax Integer linear Programming(RLP)

[14] *Totally Unimodular Matrix*

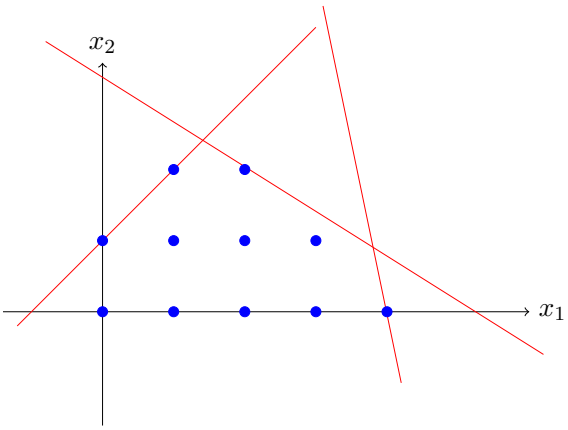
برای اینکه بررسی کنیم که آیا ماتریسی TU هست یا خیر ،کار همیشه امکان پذیری نیست. باید توجه داشته باشیم که نتیجه اخیر ، برعکسش الزاما برقرار نیست؛ یعنی اگر نقاط راسی یک چندوجهی صحیح باشد، A الزاما ماتریسی TU نیست .

مشکلاتی که داریم این است که چک کردن TU بودن ماتریس A کار راحتی نیست و اینکه ممکن است که نقاط راسی چندوجهی صحیح باشد اما ماتریس A ، ماتریسی TU نباشد.

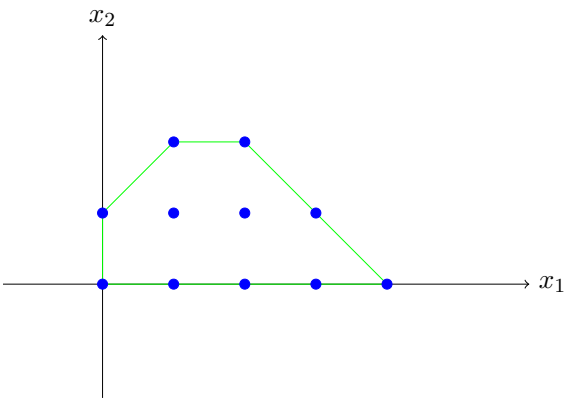
برای ارائه مثالی برای ماتریس TU ، مسئله حمل و نقل مطرح شده در جلسه اول را در نظر بگیرید.مسئله بدین صورت است که:

$$\begin{cases} \min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \\ s.t. \quad \sum_{j=1}^{m} x_{ij} = 1 \\ \sum_{i=1}^{n} x_{ij} = 1 \\ x_{ij} \geq 0, x_{ij} \in \mathbb{Z} \end{cases}$$

می‌توانیم ثابت کنیم که ماتریس ضرایب مسئله فوق، ماتریسی TU است . پس تمام نقاط راسی مسئله حمل و نقل ، حتما صحیح هستند و قید صحیح بودن را نیازی در نظر بگیریم(حالت RLP را حل می‌کنیم).



در شکل بالا نقاط شدنی مسئله، نقاط آبی رنگ هستند اما چند وجهی ما، است که محدود فضای محدود به خطوط قرمز رنگ است.



اگر بتوانیم به طریقی چندوجهی بالا را درست کنیم ، آنگاه نقاط راسی چندوجهی حاصل ، نقاط صحیح هستند و آنگاه جواب بهینه یکی از این نقاط راسی چند وجهی جدید(سبز رنگ) اتفاق می‌افتد.حال اگر چندوجهی سبزرنگ را P بنامیم، آنگاه داریم که مسئله :

$$\begin{cases} \min c^T x \\ s.t. \quad Ax = b \\ x \geq 0, x \in \mathbb{Z}^n \end{cases}$$

به مسئله زیر تبدیل می‌شود:

$$\begin{cases} \min c^T x \\ s.t. \quad x \in P \\ x \geq 0, x \in \mathbb{Z}^n \end{cases}$$

که جواب این مسئله با جواب مسئله IP نیز یکسان است و اما این مسئله جدید، یک مسئله برنامه ریزی خطی معمول(مسئله RLP) تبدیل می‌شود و بنابراین جواب آن ، در نقطه راسی رخ می‌دهد و آن نقطه، جواب بهینه هر دو مسئله است .

چندوجهی سبزرنگ را پوسته محدب [۱۵] نقاط شدنی گویند؛ یعنی برای نقاط آبی رنگ ، اگر پوسته محدب آن را در نظر بگیریم، آنگاه با یک مسئله برنامه ریزی خطی در اعداد حقیقی مواجه هستیم و آن را می‌توانیم با روش‌هایی که در جلسات قبل دیدیم، حل نماییم.

در حقیقت تعریف دقیق‌تر پوسته محدب بدین صورت است:

$$convexhull(S) = \{\cap C | S \subseteq C\}$$

که در آن C مجموعه محدب است.

اگر S شامل تعداد متناهی نقطه باشد یعنی $S = \{x^{(1)}, x^{(2)}, \ldots, x^{(k)}\}$ می‌شود ثابت کرد که رابطه زیر برقرار است:

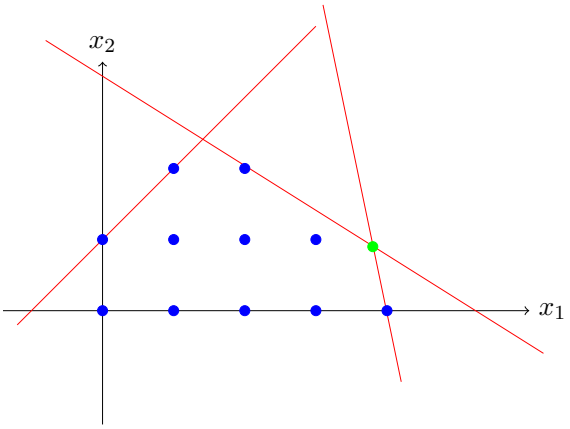
$$convexhull(S) = \{\sum_{i=1}^{k} \lambda_i x^{(i)} | \sum_{i=1}^{k} \lambda_i = 1, \lambda_i \geq 0\}$$

می‌توان ثابت کرد که اگر S تعدادی نقطه داخل یک چندوجهی باشد، آنگاه پوسته محدب آن ، یک چندوجهی است.

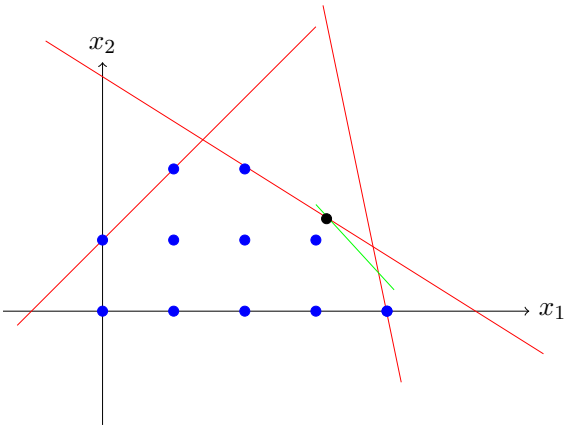
مشکلی که با آن مواجه هستیم، این است که در ابتدا ما همه نقاط شدنی را نداریم و باید تک به تک آن‌ها را بیابیم و نمی‌توانیم تعداد محاسباتمان را بر حسب ورودی مسئله کنترل کنیم ، اما می‌توانیم در هر تکرار ، تقریب بهتری از پوسته محدب داشته باشیم.فرض کنید ما مسئله RLP متناظر را توانسته‌ایم که حل کنیم.

فرض کنید که در مثال زیر، نقطه سبزرنگ(x^*)، نقطه بهینه RLP باشد.

[15] convex hull



اما این نقطه ، نقطه عددصحیح نیست. بنابراین این نقطه ، نقطه مد نظر سوال نیست و باید آن را کنار بگذاریم. برای این کار، یک برش [۱۶] می‌زنیم به طوری که هیچ نقطه صحیحی از چند ضلعی ما حذف نکند اما نقطه بهینه غیرصحیح(در اینجا x^*) را حذف کند. برای مثال به شکل زیر می‌رسیم:



که مجدد نقطه مشکی رنگ ، نقطه بهینه جدید ماست ولی همچنان عددصحیح نیست. مجدد یک برش دیگر اضافه می‌کنیم و همین روند را ادامه می‌دهیم تا جواب به اندازه کافی صحیح را پیدا کنیم.(در هر مرحله به جواب صحیح، نزدیک تر می‌شویم). این ایده را به عنوان روش صفحه برشی [۱۷] می‌شناسند.

در واقع اگر S را مجموعه نقاط با مولفه های صحیح از چندوجهی $Ax = b, x \geq 0$ باشد، آنگاه مسئله IP ما معادل با حل مسئله زیر است:

$$\begin{cases} \min c^T x \\ s.t. \quad x \in convexhull(S) \end{cases}$$

است .

هدف از روش صفحه برشی این است که در هر تکرار برش هایی را اضافه کنیم به گونه‌ای که هیچ نقطه صحیحی حذف نشود ولی جواب بهینه مسئله آزادسازی شده حذف شود و به پوسته محدب S نزدیک شویم.

[16] cut
[17] cut plane

فرض کنید x^* جواب بهینه پایه‌ای متناظر به پایه بهینه B مسئله RLP باشد.

بنابراین قید را می‌توانستیم به صورت زیر بازنویسی کنیم:

$$Ax = b \Rightarrow x_B = B^{-1}b - B^{-1}Nx_N$$

که در آن N ماتریس شامل ستون‌های A که متناظر با متغیرهای غیر پایه‌ای هستند.

قسمت $B^{-1}N = B^{-1}[\dots a_j \dots]_{j\in N}$ است. بنابراین ستون j ام ماتریس $[B^{-1}N]_j = B^{-1}a_j = \bar{a}_j$ و همینطور $B^{-1}b = \bar{b}$ نام‌گذاری می‌کنیم.

بنابراین داریم که :

$$x_B = \bar{b} - [\dots \bar{a}_j \dots]_{j\in N} \begin{bmatrix} \vdots \\ x_j \\ \vdots \end{bmatrix}_{j\in N} \Rightarrow x_B = \bar{b} - \sum_{j\in N} \bar{a}_j x_j \qquad (۱)$$

اگر که x^* شرط صحیح بودن را نداشته باشد ، با توجه به اینکه $x^*_B = \bar{b}, x^*_N = 0$ پس $k \in B$ وجود دارد که $x^*_k \notin \mathbb{Z}$.

پس اگر رابطه (۱) را بازنویسی نماییم داریم که :

$$(۱) \Rightarrow x_i + \sum_{j\in N} \bar{a}_{ij} x_j = \bar{b}_i \quad i \in B$$

آنگاه چون برای هر j داریم که $x_j \geq 0$ هستند،آنگاه

$$x_i + \sum_{j\in N} [\bar{a}_{ij}] x_j \leq x_i + \sum_{j\in N} \bar{a}_{ij} x_j = \bar{b}_i \Rightarrow x_k + \sum_{j\in N} [\bar{a}_{kj}] x_j \leq x_k + \sum_{j\in N} \bar{a}_{kj} x_j = \bar{b}_k \qquad (۲)$$

داریم که $\forall j \in B \cup N \rightarrow x_j \in \mathbb{Z}$ است . از طرفی سمت چپ رابطه (۲) عددی صحیح است که برابر با مقدار صحیحی کوچکتر مساوی $\bar{b}_k$ است که ممکن است $\bar{b}_k$ صحیح نباشد. بنابر این می‌توانیم رابطه زیر را داشته باشیم:

$$(۱) \Rightarrow x_k + \sum_{j\in N} [\bar{a}_{kj}] x_j \leq [\bar{b}_k] \qquad (۳)$$

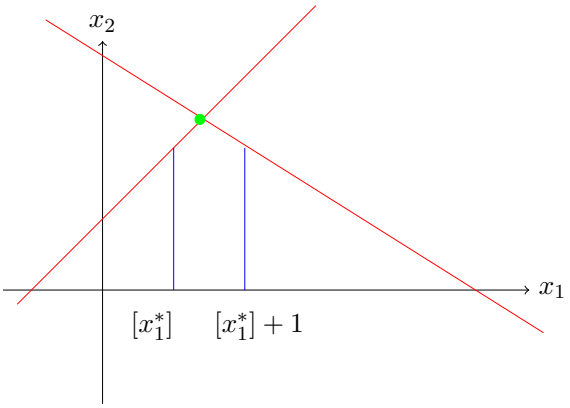
به رابطه (۳) ، **برش گوموری** [۱۸] گویند. این برش که برای x^*_k که صحیح نیست ، نوشته شده است،تضمین می‌کند که هیچ نقطه صحیحی حذف نمی‌شود ولی x^* که صحیح نبود، در این برش صدق نمی‌کند.

پس اگر ما مسئله RLP را حل کردیم و x^* یافت شد که متناظر با پایه بهینه B بود و قید صحیح بودن را نداشت،آنگاه با اضافه کردن برش گوموری (۳) به مسئله RLP ، x^* برای مسئله جدید نشدنی است ولی هیچ نقطه صحیح از فضای شدنی حذف نمی‌شود.

رویکرد دیگر برای حل این نوع مسائل رویکرد **شاخه و محدود** [۱۹] است.

[18] Gomory Cut
[19] Branch and Buond

توضیح این رویکرد بدین صورت است که جواب بهینه مسئله RLP برابر با $x^* = \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix}$ که مثلا $x_1^* \notin \mathbb{Z}$ برقرار است. مطمئنیم که جواب صحیح مسئله IP حتما $x_1 \leq [x_1^*]$ یا $x_1 \geq [x_1^*]+1$ است.



در شکل فوق ، نقطه سبزرنگ ، نقطه x^* است. دو نوار آبی رنگ به ترتیب از چپ برابر با $x_1 = [x_1^*]$ و $x_1 = [x_1^*] + 1$ هستند . حال می‌دانیم که در مجدوده بین این دو نوار، مسئله IP ما جواب ندارد.

حال شرط های $x_1 \leq [x_1^*]$ و $x_1 \geq [x_1^*] + 1$ را به مسئله RLP خود اضافه می‌کنیم. پس از این ، به دو مسئله جدید می‌رسیم که هر کدام مجدد دو مسئله RLP هستند که هر کدام جواب هایی دارند و پس از حل هرکدام ، بررسی می‌کنیم که آیا جواب این مسئله عددصحیح است یا خیر.اگر صحیح بود، دیگر نیازی نیست آن شاخه را ادامه دهیم و اگر جواب، عدد صحیح نبود، آن شاخه را مشابه قبل به دو شاخه تقسیم می‌کنیم تا زمانی که تمام شاخه ها به بسته شود(منظور از بسته شدن شاخه ها رسیدن به جواب صحیح برای آن مسئله است و یک راه دیگر این است که مقدار تابع هدف برای این شاخه از مقدار بهینه به دست آمده تا این لحظه برای مسئله IP بیشتر شود ــ حتی اگر جواب بهینه عدد صحیح نباشد ــ ، آنگاه دیگر امیدی به پیدا کردن جواب بهتر برای زیرشاخه های این مسئله نداریم و آن شاخه را دیگر ادامه نمی دهیم). ممکن است با اضافه کردن برش ها، به مسئله نشدنی برسیم و در این صورت آن شاخه را نیز می‌بندیم.

بخش دوم

جلسات اضافه

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه اضافه ۱: الگوریتم‌های برخط

نگارنده: مهسا فتحی

در این جلسه قصد داریم با الگوریتم‌های برخط [1] آشنا شویم. تاکنون تمام الگوریتم‌هایی که خواندیم قابل استفاده برای ورودی‌های برون خط [2] بوده‌اند. در این جلسه تعدادی از مسائل و تعاریف مهم این موضوع را خواهیم دید.

۱ مقدمه

تاکنون هر آنچه خواندیم با ورودی‌های برون‌خط بوده است. این بدان معناست که از ابتدا تمام ورودی‌های مسئله مشخص هستند و تصمیم‌گیری ما بر اساس همین ورودی‌ها انجام می‌شود. اما در دنیای واقعی مشاهده می‌شود بسیاری از مسائل به این شکل نیستند و ورودی‌های ما در طول زمان بیشتر شده و یا تغییر می‌کنند. به عنوان مثال می‌توان به مسئله خرید و فروش سهام اشاره کرد. شما در هر لحظه با استفاده از داده‌هایی که از ارزش هر سهم دارید می‌توانید تصمیم‌گیری کنید که آیا با گذر زمان نگه‌داشتن این سهم به سود یا ضرر شما خواهد بود. یا می‌توان به تصمیم‌گیری برای خرید یا کرایه وسیله‌ای مانند چوب اسکی اشاره کرد. از آنجا که وقتی برای بار اول به اسکی می‌رویم نمی‌دانیم قرار است در آینده باز هم اینکار را انجام می‌دهیم و یا تعداد دفعاتی که در آینده دوباره به اسکی خواهیم آمد را نمی‌دانیم، نیاز به تصمیمی داریم که داده‌های آن با گذشت زمان تکمیل می‌شود و انتخاب اینکه ارزش خرید بالاتر است یا ارزش کرایه، با گذر زمان ساده‌تر شود.

در مسئله برخط، ورودی به صورت دنباله‌ای از درخواست‌ها می‌باشد که آن را با σ نشان می‌دهیم، که این درخواست‌ها در طول زمان به‌روز می‌شوند. با مشخص کردن هزینه‌های تصمیم‌هایی که می‌گیریم اگر الگوریتم A را بر دنباله‌ی σ اجرا کنیم به $C_A(\sigma)$ هزینه ناشی از اجرای این الگوریتم گویند. ما با دانستن این هزینه به دنبال کمینه کردن تفاوت هزینه الگوریتم اجرایی بر روی داده‌های برخط و الگوریتم بهینه داده‌های برون‌خط هستیم. که این الگوریتم بهینه داده‌های برون‌خط را با OPT نشان می‌دهند. در الگوریتم بهینه فرض ما بر این است که تمام ورودی‌های آینده و گذشته مسئله را یکجا داریم و در این حالت سعی می‌کنیم با الگوریتم‌هایی که می‌شناسیم مسئله را حل کنیم. در واقع هدف ما این است که آنچه پیش‌بینی می‌کنیم خیلی دور از آنچه در واقعیت رخ می‌دهد نباشد.

تعریف ۱. *الگوریتم A را kـ رقابتی می‌گوییم اگر*

$$\forall \sigma : C_A(\sigma) \leq kC_{OPT}(\sigma) + O(1)$$

انتظار آنکه برای هر مسئله‌ای بتوانیم الگوریتمی ارائه دهیم که توانایی رقابت با OPT را داشته باشد، کمی دور از ذهن است. مثال زیر را برای حالتی می‌بینیم که این دو الگوریتم توانایی رقابت با هم را ندارند.

مثال: مسئله پیش‌بینی عدد بعدی. فرض کنید

$$\sigma \in \{-1, +1\}^*$$

[1] online
[2] offline

می‌خواهیم الگوریتمی ارائه دهیم که با مشاهده n عدد ابتدایی دنباله، عدد $n+1$ را حدس بزند. الگوریتم بهینه برون‌خط هزینه صفر را دارد زیرا طبق فرض، این نوع الگوریتم تمام داده‌ها را از همان ابتدا دارد. برای الگوریتم برخط به صورت شهودی واضح است که اگر الگوریتم ارائه شده به صورت تصادفی عمل کند انتظار می‌رود حداقل هزینه نصف تعداد داده‌ها را بپردازد. در واقع حداقل نصف حالات جواب اشتباه می‌دهد. همچنین اگر یک فیلمنامه دشمنانه را در نظر بگیریم، دشمن می‌تواند الگوریتمی که ما ارائه داده‌ایم را شبیه سازی کند و در هر سری متفاوت با جوابی که ما می‌دهیم را به عنوان درخواست بعدی وارد کند. که در این حالت هزینه بیشینه می‌شود.

⊠

۲ مسئله کرایه کردن چوب اسکی

در مسئله کرایه چوب اسکی[3] دو انتخاب داریم، یکی آنکه چوب اسکی را بخریم و یکی آنکه هر سری چوب اسکی را کرایه کنیم. برای این مسئله از فرض‌های ساده‌کننده‌ای استفاده می‌کنیم که می‌توان به تاثیر نداشتن وضع مالی بر انتخاب هرکدام از حالات و یا عدم توانایی فروش دوباره پس از خرید چوب اسکی، اشاره کرد.

فرض کنید هزینه کرایه چوب اسکی $r = 50$ دلار و هزینه خرید آن $b = 500$ دلار است. این هزینه‌ها نشان می‌دهد که اگر ما از ابتدا بدانیم کمتر از ۱۰ روز قصد داریم اسکی برویم بهتر است کرایه کنیم و در غیر این صورت خرید اسکی به صرفه‌تر است. پس می‌خواهیم برای هر روز تصمیم بگیریم که اسکی بخریم یا کرایه کنیم. در این مسئله دنباله درخواست‌ها به صورت $\sigma = 111111$ است که در واقع تنها طول این دنباله اهمیت دارد و تعداد روزهایی که به اسکی رفته‌ایم را مشخص می‌کند. یک راه حل برای این مسئله آن است که همان روز اول اسکی را بخریم که در این صورت ۵۰۰ دلار هزینه کرده‌ایم. در بدترین شرایط اگر ما فقط یک روز به اسکی برویم یعنی دنباله درخواست ما تنها شامل یک ۱ باشد ۱۰ برابر ضرر کرده‌ایم زیرا در حالت بهینه تنها ۵۰ دلار هزینه می‌کردیم.

راه حل اصلی: اگر ۹ روز اول اسکی را کرایه کنیم و روز ۱۰ام آن را بخریم، در این حالت مشاهده می‌کنیم که ضریب رقابتی با الگوریتم بهینه کمتر از ۲ است. به این صورت که اگر تعداد روزهایی که به اسکی می‌رویم بزرگتر یا مساوی ۱۰ باشد آنگاه در الگوریتم بهینه باید همان روز اول اسکی را می‌خریدیم که در این صورت ۵۰۰ دلار هزینه می‌کردیم. اما در این الگوریتم پیشنهادی ۵۰۰+۴۵۰ دلار هزینه کرده‌ایم که ۱/۹ برابر حالت بهینه است.

$$2 - \frac{r}{b} = 1.9$$

می‌توان نشان داد که نمی‌توان الگوریتمی با ضریب رقابتی بهتر پیشنهاد داد. اگر زودتر از ۱۰ روز، مثلا روز ۵ام، اسکی را بخریم و تعداد روزهایی که به اسکی می‌رویم ۵ روز باشد، آنگاه هزینه بهینه ۲۵۰ دلار است که ما ۵۰۰+۲۰۰ دلار در این روش هزینه کرده‌ایم که ضریب رقابتی کمتر از ۳ دارد. همچنین اگر دیرتر از روز ۱۰ام اسکی را بخریم آنگاه هزینه بهینه همان ۵۰۰ باقی می‌ماند و ما پول بیشتری نسبت به راه حل اصلی هزینه کرده‌ایم و ضریب رقابتی بزرگتر از ۱/۹ به ما خواهد داد.

در این مسئله و مسائل مشابه ما نهایتا می‌توانیم به ضریب رقابتی نزدیک به ۲ برسیم (نسبت $\frac{r}{b}$ مشخص می‌کند تا چه حد می‌توان به ۲ نزدیک شد. اگر b مضربی از r باشد که این نسبت از ۲ کم می‌شود و اگر نباشد ضریب رقابتی ۲ یا نزدیک ۲ می‌شود). در حل این مسئله استراتژی این‌گونه است که دقیقا وقتی به این فکر کردیم که اگر اسکی را اول خریده بودیم بهتر بود، همان موقع بخریم. در واقع از سیاست better late than never استفاده می‌کنیم.

حل مسئله با الگوریتم‌های تصادفی: فرض کنید در روز ۸ام سکه می‌اندازیم و اگر شیر آمد اسکی را می‌خریم. جدول ۲ ضریب رقابتی را برای طول‌های مختلف دنباله درخواست‌ها نشان می‌دهد.

[3] ski rental (rent or buy) problem

	هزینه الگوریتم تصادفی	هزینه الگوریتم بهینه	ضریب رقابتی
$n < 8$	n	n	۱
$n = 8$	$\frac{1}{2}.17 + \frac{1}{2}.8 = 12.5$	8	$\frac{12.5}{8}$
$n = 9$	$\frac{1}{2}.17 + \frac{1}{2}.9 = 13$	9	$\frac{13}{8}$
$n \leq 10$	$\frac{1}{2}.17 + \frac{1}{2}.19 = 18$	10	$\frac{18}{10}$

مشاهده می‌کنید که ضریب رقابتی به ۱/۸ کاهش پیدا کرده است. با تغییر روز و احتمالات ممکن است این ضریب را حتی به ۱/۶ رساند. در واقع اگر b خیلی بزرگ‌تر از r باشد می‌توان ضریب رقابتی را به $\frac{e}{e-1}$ رساند.

۳ مسئله آسانسور

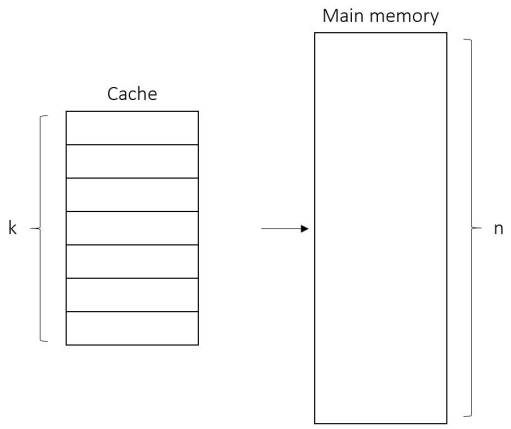
در مسئله آسانسور[۴] انتخاب بین با پله رفتن و با آسانسور رفتن است. اگر فرض کنیم که رفتن به طبقه‌ای با آسانسور $\epsilon = 15$ ثانیه و با پله $s = 60$ ثانیه طول می‌کشد. یک راه حل مانند حالت قبل آن است که از همان ابتدا با پله برویم. اما راه حل اصلی این مسئله آن است که ۴۵ ثانیه صبر کنیم و اگر آسانسور نیامد با پله برویم. استراتژی این مسئله نیز مانند حالت قبل است که دقیقا زمانی که حس کردیم اگر با پله می‌رفتیم بهتر بود همینکار را بکنیم. با این کار هزینه ۶۰+۴۵ را می‌پردازیم که الگوریتم بهینه به ما هزینه ۶۰ را متحمل می‌شد و در نتیجه ضریب رقابتی این راه حل $\frac{105}{60} = 1.75$ است.

در واقع این دو مسئله معادل یکدیگرند. به این صورت که هر ϵ ثانیه در مسئله آسانسور همان یک روز در مسئله کرایه چوب اسکی است. با این تفاوت که مسئله آسانسور پیوسته است. همچنین می‌توان مسائل دیگری را در دنیای واقعی معادل این مسئله دانست. به عنوان مثال اگر یک طراحی مشکل دار و نامرتب از یک الگوریتم داشته باشیم، بعد از پیدا کردن یک مشکل می‌توان با رخنه [۵] آن را برطرف کرد و یا با عوض کردن طراحی، مشکل را اساسی برطرف کرد. در اینجا حل مشکل با رخنه قطعا زمان و انرژی کمتری می‌گیرد اما با اینکار ممکن است در آینده با مشکلات دیگری روبه‌رو شویم که در نتیجه نیاز به رخنه‌های بیشتری خواهد داشت. پس در واقع انتخاب بین این دو کار همانند مسئله کرایه کردن چوب اسکی است. یک مثال دیگر دیسک سخت است که بعد از هر درخواست چرخش خود را متوقف کند یا اینکار را ادامه دهد. زیرا اگر بداند که درخواست بعدی قرار است در زمان کوتاهی داده شود، بهتر است چرخش خود را متوقف نکند که در این صورت در مصرف انرژی صرفه‌جویی شود.

۴ مسئله صفحه‌بندی

در سیستم کامپیوتری، صفحه، بخشی پیوسته و با طول ثابت از حافظه مجازی[۶] است که کوچکترین واحد داده در مدیریت حافظه[۷] محسوب می‌شود. این صفحات هر کدام جداگانه می‌توانند به حافظه نهان[۸] برای دسترسی سریع‌تر منتقل شوند، در حالی که تمام صفحات در حافظه اصلی با دسترسی کندتر ذخیره هستند. در مسئله صفحه‌بندی[۹]، هدف آن است که آنچه در حافظه نهان ذخیره می‌کنیم به گونه‌ای باشد که تعداد دفعاتی که لازم است صفحات داده را از حافظه اصلی به حافظه نهان منتقل کنیم کمینه باشد یا به طور معادل نیاز به آنچه در حافظه نهان است بیشینه باشد.

[4] the elevator problem
[5] hacking
[6] virtual memory
[7] memory management
[8] cache
[9] paging/caching problem



در جلسات قبل مشاهده کردیم که بهینه‌ترین الگوریتم برای حل این مسئله در حالت برون‌خط، الگوریتم دورترین در آینده[۱۰] است که در آن نیاز به پیش‌بینی آینده است. زیرا آن صفحه‌ای که در آینده دیرتر به آن نیاز خواهیم داشت را از حافظه بیرون می‌اندازیم و آنچه می‌خواهیم را وارد حافظه می‌کنیم. که این الگوریتم در واقعیت قابل پیاده سازی نیست. با این حال الگوریتم‌هایی برای حالت برخط وجود دارند که معمولا عملکرد خوبی نشان می‌دهند. این الگوریتم‌ها ضریب رقابتی پایینی دارند.

اخیرا کمتر استفاده شده[۱۱]: در این الگوریتم صفحه‌ای که در آینده دورتری استفاده شده، بیرون انداخته می‌شود.

خروج به ترتیب ورود[۱۲]: در این الگوریتم صفحه‌ای که دیرتر وارد حافظه نهان شده بیرون انداخته می‌شود.

لم ۱. *هیچ الگوریتم قطعی مسئله صفحه‌بندی نمی‌تواند بهتر از kـ رقابتی باشد.*

اثبات. اگر در یک مسئله $n = k + 1$ صفحه داشته باشیم و یک فیلمنامه دشمنانه را در نظر بگیریم، دشمن می‌تواند الگوریتم ما را شبیه سازی کند و در هر مرحله تنها صفحه‌ای که در حافظه نهان نیست را درخواست دهد. پس در بدترین حالت باید هزینه n را برای این الگوریتم پرداخت کنیم.

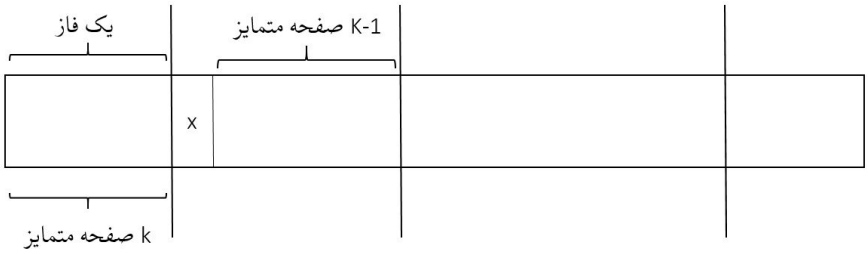
در الگوریتم OPT هزینه حداکثر $\frac{n}{k}$ را می‌پردازیم. زیرا در الگوریتم دورترین در آینده، آن صفحه‌ای را بیرون می‌اندازیم که در آینده دورتری از آن استفاده شده است. پس ما $k - 1$ صفحه در حافظه نهان داریم که زودتر از آن صفحه‌ای که بیرون انداخته‌ایم به آن احتیاج پیدا می‌کنیم. پس برای این $k - 1$ صفحه هیچ هزینه‌ای پرداخت نمی‌کنیم. □

تعریف ۲. *الگوریتم A را محافظه‌کار می‌گوییم اگر روی هر زیردنباله‌ای که شامل حداکثر k صفحه متمایز است حداکثر k واحد هزینه داشته باشد.*

لم ۲. *هر الگوریتم محافظه‌کار kـ رقابتی است.*

اثبات. این لم را به این شکل تغییر می‌دهیم: در مقایسه با الگوریتم بهینه‌ای که اندازه حافظه نهان آن h است هر الگوریتم محافظه‌کار A، $\frac{k}{k-h+1}$ـ رقابتی است. برای این کار دنباله درخواست‌ها را به چند فاز تقسیم می‌کنیم که هر فاز دقیقا k صفحه متمایز دارد. در تعیین این فازها به اولین xای که رسیدیم و با شمارش آن تعداد صفحات متمایز $k+1$ می‌شد، آن را یک فاز حساب می‌کنیم. می‌خواهیم الگوریتم بهینه را روی هر فاز بررسی کنیم. مشاهده می‌کنیم در هر فاز الگوریتم A، k واحد هزینه می‌دهد. حال باید نشان دهیم الگوریتم بهینه $k - h + 1$ هزینه می‌دهد.

[10] Farthest in future
[11] LRU
[12] FIFO



در هر فاز بعد از x، $k-1$ عنصر متمایز وجود دارد که با توجه به اینکه x در حافظه نهان است پس حافظه نهان $h-1$ تا از آنها را در خود دارد. پس ما در هر فاز حداقل باید $k-1-(h-1)=k-h$ هزینه بدهیم. برای بررسی x دو حالت داریم، اگر x در حافظه نباشد که باید هزینه ۱ را بدهیم. در غیر این صورت x در حافظه نهان بوده است و از آنجا که در فاز قبلی این صفحه وجود نداشته پس این صفحه در تمام یک فاز یک صفحه از حافظه نهان را بیهوده اشغال کرده است. پس اگر لازم نباشد هزینه‌ای برای این صفحه در این فاز بدهیم این هزینه را در فاز قبل داده بوده‌ایم و می‌توان در کل گفت که در هر فاز $k-h+1$ را بدهیم. پس اثبات تمام است. □

در این اثبات مشاهده کردیم که در جابه‌جایی صفحات داده به حافظه نهان گاهی تصمیماتی گرفته می‌شود که حافظه نهان با صفحات بیهوده پر می‌شود که ارتباطی به طول ورودی ندارد. این همان بخش $O(1)$ است که در تعریف kـ رقابتی آورده شد.

۱.۴ الگوریتم علامت زدن تصادفی

ساده‌ترین الگوریتم تصادفی که می‌توان برای حل مسئله صفحه‌بندی پیشنهاد داد، انتخاب تصادفی صفحه برای بیرون انداختن از حافظه نهان است. الگوریتم علامت زدن تصادفی[۱۳] یا RMA با شبیه‌سازی اینکار را کمی بهتر انجام می‌دهد. پیاده سازی این الگوریتم به شکل زیر است.

صفحه p درخواست داده می‌شود: اگر p در حافظه نهان نباشد،اگر همه صفحات در حافظه نهان علامت خورده باشند، علامت همه را برمی‌داریم. سپس یک صفحه بدون علامت را به صورت تصادفی انتخاب کرده و بیرون می‌اندازیم. بعد از اطمینان از آنکه صفحه p در حافظه است آن را علامت‌دار می‌کنیم.

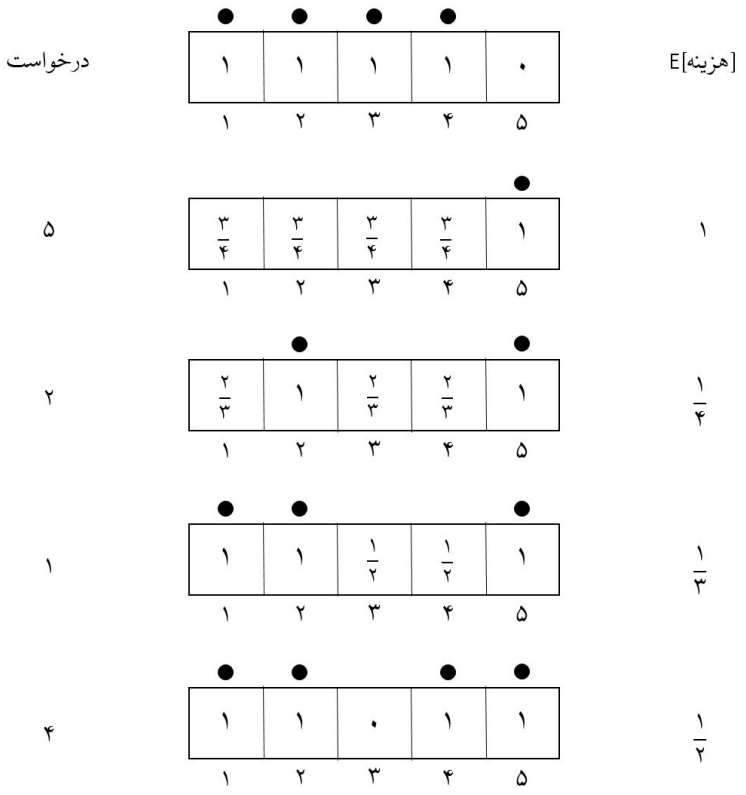
مثال: فرض کنید حافظه نهانی با اندازه $k=4$ داریم و تعداد کل صفحات موجود $n=5$ است. در شکل ۱.۴ مشاهده می‌کنید که طبق این الگوریتم تصمیم گیری‌ها و علامت‌گذاری‌ها به چه صورت است. ۱ در هر شکل نشان دهنده حضور آن صفحه در حافظه نهان و دایره توپر بالای هر صفحه نشان از علامت‌دار بودن آن صفحه است. همچنین در سمت راست هر تصویر امید ریاضی هزینه پرداختی در هر مرحله را مشاهده می‌کنید.

⊠

لم ۳. *الگوریتم RMA $O(lgk)$ـ رقابتی است. (در مقابل یک دشمن بی‌توجه[۱۴])*

ابتدا توضیح می‌دهیم که دشمن بی‌توجه به چه معناست. در الگوریتم‌های قطعی تنها یک نوع دشمن داریم. دشمنی که می‌تواند کد الگوریتم ما را بخواند و انتخاب ما را شبیه‌سازی کند. در مقابل برای الگوریتم‌های تصادفی سه نوع دشمن داریم. دشمن بی‌توجه دشمنی است که می‌تواند کد الگوریتم را بخواند اما متوجه تصادفی بودن انتخاب نمی‌شود بنابراین نمی‌تواند انتخاب الگوریتم را حدس بزند. دشمن نوع دوم دشمن قابل تطبیق[۱۵] است که می‌تواند متوجه بشود که انتخاب‌های تصادفی الگوریتم چه بوده اما نمی‌تواند انتخاب‌های آینده را

[13] Random Marking Algorithm
[14] oblivious
[15] adaptive



حدس بزند. دشمن نوع سوم دشمن کاملا قابل تطبیق[۱۶] است که می‌تواند انتخاب‌های آینده الگوریتم را نیز تشخیص دهد.

اثبات. در اینجا نیز مانند اثبات قبل دنباله‌ی درخواست‌ها را به فازهای kتایی تقسیم می‌کنیم. برای اثبات، عملکرد الگوریتم RMA را بر هر فاز بررسی می‌کنیم. می‌توان به راحتی نشان داد با عبور از فاز $i-1$ علامت همه‌ی صفحه‌ها غیر از اولین صفحه فاز بعد، برداشته می‌شود (مانند ممرحله اول مثالی که در شکل ۱.۴ دیدیم). در فاز i دو دسته درخواست صفحه داریم، صفحه‌هایی که در حافظه نهان حضور دارند و صفحه‌هایی که در حافظه نهان نیستند.

n_i: تعداد صفحات متمایزی از فاز i که در فاز $i-1$ نیستند. که به این درخواست‌ها جدید می‌گویند.

o_i: تعداد صفحات متمایزی از فاز i که در فاز $i-1$ بوده‌اند. که به این درخواست‌ها قدیمی می‌گویند.

الگوریتم مذکور به ازای دیدن هر یک از درخواست‌های جدید باید ۱ واحد هزینه بپردازد. پس هزینه‌ی این الگوریتم برای صفحه‌های جدید n_i است.

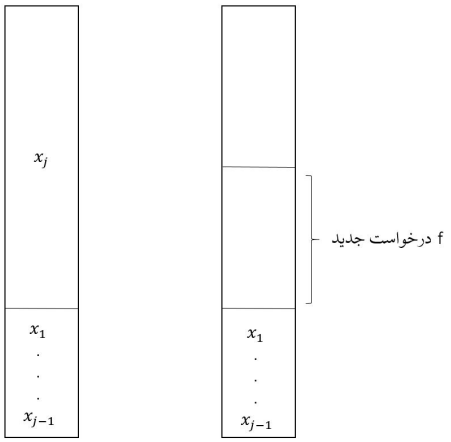
در ادامه هزینه الگوریتم RMA را برای درخواست قدیمی jام بررسی می‌کنیم. فرض کنید دنباله زیر، تمامی صفحات قدیمی باشد که در فاز iام درخواست شده‌اند.

$$x_1, x_2, x_3, ..., x_{o_i}$$

بدین ترتیب $x_1, x_2, ..., x_{j-1}$ در حافظه نهان هستندو همگی علامت دارند و ما می‌خواهیم هزینه درخواست x_j را بررسی کنیم. برای بررسی وضعیت حافظه نهان در این لحظه فرض کنید تا لحظه دیدن درخواست x_j برای اولین بار، f درخواست جدید داشته‌ایم.

پس اگر $j-1$ صفحه از حافظه برای درخواست‌های قدیمی علامت‌دار، پر شده باشد، fتای دیگر برای درخواست‌های جدید پرشده

[16] fully adaptive



و علامت‌دار هستند. این به آن معناست که از $k-(j-1)$ صفحه در حافظه نهان fتای آن به صورت تصادفی بیرون انداخته شده است که ممکن است x_j در این بین بوده باشد. پس می‌توان گفت x_j با احتمال کوچکتر از $\frac{f}{k-(j-1)}$ از حافظه بیرون انداخته شده است.

$$\frac{f}{k-(j-1)} \leq \frac{n_i}{k-(j-1)}$$

$$E[C_{RMA}(old)] \leq n_i \sum_{j=1}^{o_i} \frac{1}{k-(j-1)} \leq n_i \sum_{j=1}^{k} \frac{1}{k-(j-1)} = n_i H(k)$$

$$E[C_{RMA}(\sigma_i)] \leq n_i + n_i H(k) = O(n_i) lgk$$

$$E[C_{RMA}(\sigma)] \leq O(H(k)) \sum_{phases} n_i$$

با داشتن کران بالا برای امید ریاضی هزینه اجرای این الگوریتم بر روی دنباله درخواست‌ها، اگر هزینه اجرای الگوریتم بهینه را به دست آوریم اثبات تمام است.

ادعا ۱. *هزینه OPT برابر است با $\Omega(\sum_{phases} n_i)$.*

ما می‌خواهیم نشان دهیم که الگوریتم بهینه در هر فاز باید هزینه n_i را بپردازد. به شکل ۱.۴ توجه کنید. اگر در فاز iام k درخواست صفحه داشته باشیم، در فاز $i+1$ به تعداد درخواست‌های جدید n_i صفحه جدید به حافظه نهان اضافه می‌شود. یعنی در الگوریتم بهینه از آنجا که آینده را می‌دانیم پس صفحه‌های درخواست‌های قدیمی را تا جایی که ممکن است از حافظه نهان بیرون نمی‌اندازیم و فقط برای درخواست‌های جدید هزینه می‌دهیم. پس اگر i زوج باشد و یا فرد می‌توان دو رابطه زیر را داشت و از آنها حالت کلی را به دست آورد.

$$C_{OPT}(\sigma) \geq \sum_{iodd} n_i$$

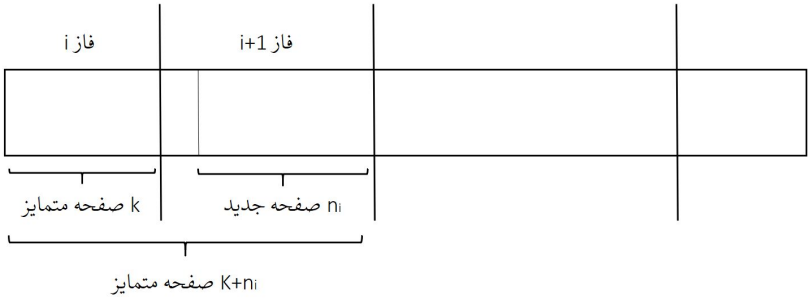
$$C_{OPT}(\sigma) \geq \sum_{ieven} n_i$$

$$2C_{OPT}(\sigma) \geq \sum_{i} n_i$$

$$C_{OPT}(\sigma) \geq \frac{1}{2} \sum_{i} n_i$$

بنابراین ادعا و درنتیجه لم ثابت می‌شود. □

طبق آنچه اثبات کردیم این الگوریتم $O(lgk)$ـ رقابتی است . در ادامه نشان می‌دهیم الگوریتم‌های تصادفی برای این مسئله نمی‌توانند



از $O(lgk)$ بهتر عمل کنند. در نتیجه این الگوریتم بهینه است.

قضیه ۱. *ضریب رقابتی هر الگوریتم (تصادفی) برای مسئله صفحه‌بندی $\Omega(loqk)$ است.*

طراحی الگوریتم مانند بازی با دشمن است که در آن همیشه دشمن سعی می‌کند ورودی‌ای بدهد که الگوریتم پیشنهادی اشتباه عمل کند. پس تلاش الگوریتم باید برای این باشد که بر روی ورودی‌های دشمن بهترین عملکرد را داشته باشد و تا جای ممکن زمان اجرا را پایین بیاورد. الگوریتم‌های تصادفی به صورت مشابه، مانند تصادفی بازی کردن است. هر الگوریتم تصادفی را می‌توان به صورت توزیع احتمالی بر روی الگوریتم‌های قطعی نگاه کرد. به عنوان مثال بازی سنگ، کاغذ، قیچی را در نظر بگیرید. اگر استراتژی طرف مقابل را ندانیم بهینه‌ترین راه حل برای ما آن است که به صورت تصادفی انتخاب کنیم، یعنی هر سه حالت را با احتمال برابر انتخاب کنیم. اما اگر طرف مقابل استراتژی تصادفی داشته باشد و با احتمال برابر، هر یک از سه حالت را انتخاب کند، آنگاه نیازی به تصادفی بازی کردن ما نیست و می‌توانیم همیشه مثلا قیچی را بازی کنیم. برای ورودی‌هایی که دشمن می‌دهد نیز این حالت برقرار است . یعنی اگر در الگوریتم ما انتخاب به صورت تصادفی باشد، دشمن می‌تواند الگوریتم ما را نگاه کند و در هر مرحله یک ورودی ثابت که امید ریاضی الگوریتم ما برای آن ورودی خوب نیست را درخواست دهد. برعکس این حالت نیز درست است. یعنی اگر ورودی الگوریتم ما به صورت تصادفی باشد، بهترین راه حل برای این ورودی یک الگوریتم قطعی است.

برای اثبات قضیه کران پایین الگوریتم صفحه‌بندی ابتدا اصل زیر را ثابت می‌کنیم و سپس با استفاده از آن قضیه را ثابت می‌کنیم.

تعریف ۳. *اصل مینیمکس یائو[17]. برای هر توزیع q روی ورودی‌ها، متوسط زمان اجرای بهترین الگوریتم قطعی با فرض اینکه ورودی از توزیع q انتخاب شود، یک کران پایین برای زمان اجرای بهترین الگوریتم تصادفی روی بدترین ورودی است.*

اثبات. فرض کنید $\mathcal{A}$ و $\mathcal{X}$ فضای الگوریتم‌ها و ورودی‌ها باشند. توزیع‌های p و q را روی $\mathcal{A}$ و $\mathcal{X}$ به صورت $p_a = Pr[A = a]$ و $q_x = Pr[X = x]$ در نظر بگیرید که A و X متغیرهای تصادفی برای الگوریتم‌ها و ورودی‌ها باشند. همچنین $C(a, x)$ را هزینه اجرای الگوریتم a روی ورودی x تعریف کنید. داریم:

$$C = \max_{x\in\mathcal{X}} E_p[C(A, x)] \geq \min_{a\in\mathcal{A}} E_q[C(a, X)] = D$$

پس باید نامساوی بالا را برای اثبات اصل ثابت کنیم.

$$C = \sum_x q_x.C$$

$$\geq \sum_x q_x E_p[C(A, x)] = \sum_x q_x \sum_a P_a C(a, x)$$

$$= \sum_a P_a \sum_x q_x C(a, x) = \sum_a P_a E_q[C(a, X)]$$

[17] Yao's minimax principle

$$\geq \sum_{a} P_a.D = D$$

□

اثبات. فرض کنید $n = k + 1$ و دنباله درخواست‌ها به صورت زیر باشد. و این دنباله را مانند اثبات‌های قبل فازبندی کرده‌ایم.

$$p_1, p_2, p_3, ..., p_m$$

هزینه الگوریتم برون‌خط بهینه برای هر فاز ۱ است. زیرا در هر فاز تنها یک درخواست جدید صورت می‌گیرد و باقی تکراری هستند. حال می‌خواهیم ببینیم بهترین الگوریتم قطعی چه هزینه‌ای را باید در هر فاز بدهد.

برای هر الگوریتم قطعی، در هر مرحله k صفحه در حافظه نهان موجود است و با احتمال $\frac{1}{k+1}$، درخواست بعدی در این حافظه نیست و باید یک صفحه را بیرون انداخته و صفحه درخواستی را وارد حافظه کنیم. بنابراین هزینه هر الگوریتم قطعی برای این مسئله در هر فاز برابر است با: طول فاز.$\frac{1}{k+1}$

$$E[phase\ length] = (k+1)H(k+1)$$

$$\frac{1}{k+1}.E[phase\ length] = H(k+1) = \theta(lgk)$$

پس باید نشان دهیم که طول هر فاز $(k+1)H(k+1)$ است. این مسئله معادل یک مسئله کلاسیک به نام جمع‌کننده کوپن[۱۸] است. در این مسئله n سطل داریم و توپ‌ها را به صورت تصادفی در این سطل‌ها می‌اندازیم و به دنبال آن هستیم که به طور متوسط باید چند توپ پرت کنیم تا در هر سطل حداقل یک توپ باشد. یا به صورت معادل اگر n کوپن از حالات مختلف داشته باشیم و در هر بار خرید یک حالت آن را به دست آوریم، به طور متوسط باید چند کوپن بخریم تا از هر نوع حداقل یکی داشته باشیم.

فرض کنید i صفحه را دیده‌ایم و در هر مرحله بعد، احتمال اینکه صفحه از نوع $i+1$ باشد $\frac{k+1-i}{k+1}$ است. پس انتظار داریم که پس از $\frac{k+1}{k+1-i}$ گام به صفحه نوع $i+1$ برسیم. بنابراین:

$$E[\sum_{i=1}^{k} x_i] = \sum_{i=1}^{k} E[x_i] = \sum_{i=1}^{k} \frac{k+1}{k+1-i} = (k+1)H(k+1)$$

پس طبق اصل مینیمکس یائو هیچ الگوریتم تصادفی بهتر از $\theta(lgk)$ نمی‌توان یافت.

□

۵ مسئله kـ سرویس دهنده

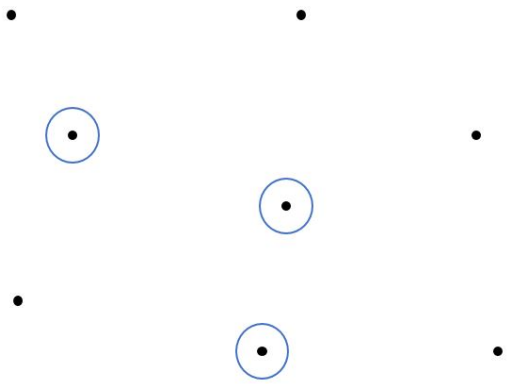
مسئله kـ سرویس دهنده[۱۹] تعمیمی از مسئله صفحه‌بندی است. در این مسئله یک فضای متریک (V, d) داریم که در آن d یک تابع فاصله به صورت زیر است.

$$d : V \times V \to \Re$$

در این مسئله k سرویس دهنده داریم که در ابتدا به گونه‌ای بر نقطه‌های d مشخص شده‌اند. در شکل ۵ نقاط با دایره‌های توپر و سرویس‌دهنده‌ها با دایره‌های آبی در اطراف بعضی از این نقاط قرار دارند. حال دنباله‌ای از درخواست‌ها داریم که در هر مرحله یکی از این

[18] coupon collector
[19] k-server problem

نقاط درخواست داده می‌شوند. اگر سرویس دهنده‌ها روی این نقاط باشند که هزینه‌ای پرداخت نمی‌کنیم. اما اگر آن نقطه سرویس دهنده‌ای نداشته باشد باید یکی از سرویس دهنده‌ها را جابه‌جا کنیم و به اندازه فاصله جابه‌جایی هزینه بدهیم. هدف آن است که الگوریتمی ارائه دهیم تا این هزینه‌ها کمینه شود.



در این مسئله اگر هر نقطه را یک صفحه در مسئله صفحه‌بندی در نظر بگیریم، و فاصله میان تمام نقاط را ۱ بگیریم، در واقع مسئله صفحه‌بندی را شبیه سازی کرده‌ایم.

برای این مسئله تاکنون الگوریتم قطعی با ضریب رقابتی $O(2k-1)$ پیشنهاد شده است، اگرچه حدس زده می‌شود که می‌توان الگوریتمی با ضریب رقابتی $O(k)$ نیز پیدا کرد. در سال ۲۰۱۷ میلادی الگوریتم تصادفی با $O(lg^6 k)$ پیشنهاد شد. اما حدس زده می‌شود که می‌توان الگوریتم تصادفی با $O(lgk)$ نیز یافت.

مراجع

[۱] ویدیو درس آنالیز الگوریتم، جلسه اضافه ۱.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه اضافه ۲: مسأله خبرگان و روش به‌روزرسانی ضربی وزن‌ها نگارنده: کیانا عسگری

در این جلسه درباره‌ی مسأله خبرگان و روشی برای حل این مسأله به نام به‌روزرسانی ضربی وزن‌ها بحث می‌کنیم. این روش، به نوعی یک شبه الگوریتم[۱] است و به همین دلیل برای آن بجای لفظ الگوریتم، از لفظ «روش» استفاده می‌شود؛ همچنین بسته به نحوه استفاده از آن، می‌توان الگوریتم‌هایی شبیه به هم را از طریق آن ایجاد کرد.

این مسأله به گونه‌ای در ادامه مسائل برخط است و می‌توان آن را آموزش برخط[۲] نامید.

خیلی اوقات، ما نیاز داریم نسبت به اتفاقاتی که قرار است در آینده بیفتد، اطلاعاتی کسب کنیم و با این‌کار به سود برسیم. به‌عنوان مثال، وضع هوای فردا یا قیمت سهام در هفته آتی از این قبیل هستند. در این مسأله تعدادی فرد خبره داریم و می‌خواهیم آینده را با کمک آنها به طور «خوبی» پیش‌بینی کنیم به‌طوری‌که در نهایت پیش‌بینی ما نسبت به خبره‌ها بدتر نباشد؛ یعنی بدون اینکه بهترین خبره را بشناسیم، بتوانیم به‌نوعی طبق نظرات او پیش برویم تا نهایتاً در دراز مدت کل اشتباهات ما در پیش‌بینی‌مان نسبت به بهترین خبره خیلی «بدتر» نشود .

مسأله خبرگان[۳]

فرض کنید n خبره[۴] داریم. قرار است اتفاقی در طول چند روز (مرحله) بیفتد که تعداد مراحل آن هم مجهول است. در هر روز نظر خبرگان را در مورد اتفاقاتی که فردا قرار است بیفتد می‌پرسیم و می‌خواهیم بر اساس این نظرات، برای اتفاقات مرحله بعد پیش‌بینی کنیم. سپس با پایان روز، نتیجه اتفاق معلوم می‌شود و ما می‌فهمیم که آیا خودمان و بقیه خبرگان اشتباه کرده‌ایم یا خیر. این فرآیند به مدت نامعلومی ادامه پیدا می‌کند.

هدف ما ارائه الگوریتمی است که در هر زمان (مثلاً بعد t مرحله)، تعداد اشتباهات ما در آن نسبت به بهترین فرد خبره خیلی بیشتر نباشد (منظور از بهترین فرد خبره در روز t ام خبره‌ای است که تاکنون کمترین پیش‌بینی اشتباه را کرده است. بهترین خبره ممکن است در هر روز تغییر کند). در این مسأله فرض می‌کنیم که رخدادها (اتفاقاتی که در مورد آن‌ها پیش‌بینی صورت می‌گیرد) حالت صفر و یکی دارند؛ یعنی به‌عنوان مثال در هر روز به دو صورت بالا[۵] و پایین[۶] هستند و پیش‌بینی می‌کنیم که نرخ زیاد یا کم می‌شود.

دقت کنید برای حالاتی که نتایج پیوسته یا گسسته با تعداد حالات بیشتر از دو باشند هم این روش‌ها کار خواهند کرد که در ادامه به آن اشاره می‌کنیم. اکنون مسأله را با ساختارهای گوناگون بررسی می‌کنیم.

[1] Meta algorithm
[2] Online learning
[3] MWU experts
[4] Expert
[5] up
[6] down

بررسی حالات مختلف

حالت اول :فرض کنید n تعداد خبرگان باشد (دقت کنید تعداد روزها متغیر ولی تعداد خبرگان ثابت است).

فرض می‌کنیم می‌دانیم که بهترین خبره، هیچ‌گاه اشتباه نمی‌کند. با این فرض چگونه می‌توانیم طوری عمل کنیم که تعداد اشتباهات ما حداکثر $\lg n$ باشد؟

حل : کافی است هر بار طبق نظر اکثریت خبرگانی که تاکنون اشتباه نکرده‌اند، پیش‌بینی کنیم. به‌عبارتی، در مرحله اول که هنوز هیچ‌کس اشتباه نکرده است، نظر تمام خبره‌ها را در نظر میگیریم و طبق نظر اکثریت برای اتفاق فردا تصمیم‌گیری می‌کنیم. در هر مرحله بعد از مشخص شدن نتیجه، خبرگانی که پیش‌بینی آن‌ها غلط بود را کلاً کنار گذاشته و از این به بعد به نظرات آن‌ها توجه نمی‌کنیم.

قضیه ۱. *تعداد اشتباهات الگوریتم بالا، حداکثر $\lg n$ است.*

اثبات. علت حکم بالا این است که ما از نظر اکثریت خبرگانی که اشتباه نکرده‌اند تبعیت می‌کنیم. پس هر بار که الگوریتم پیش‌بینی اشتباهی انجام می‌دهد، باید اکثریت خبرگانی که تاکنون اشتباه نکرده‌بودند، اشتباه کرده باشند. در نتیجه تعداد خبرگانی که تاکنون اشتباه نکرده‌اند، در این مرحله حداقل نصف می‌شود. پس حداکثر $\lg n$ بار اشتباه می‌کنیم. □

تمرین : نشان دهید در این شرایط این الگوریتم بهینه است؛ یعنی نمی‌توانیم الگوریتمی داشته باشیم که در بدترین حالت خود کمتر از $\lg n$ بار اشتباه کند.

نکته: توجه کنید که تحلیل‌های ما، تحلیل بدترین حالت هستند؛ یعنی فرض می‌کنیم اتفاقات توسط دشمنی تعیین می‌شوند که هدف او بیشینه کردن اشتباهات ما (نسبت به بهترین خبره) است.

حالت دوم

اکنون فرض می‌کنیم بهترین خبره، حداکثر M بار اشتباه می‌کند. چگونه می‌توانیم عمل کنیم که اشتباهات ما حداکثر $(M+1)(\lg n+1)$ باشد؟

حل : ایده این است که به صورت مرحله به مرحله، الگوریتم حالت اول را اجرا کنیم؛ یعنی از بین خبرگانی که اشتباه نکرده‌اند طبق نظر اکثریت عمل کنیم و هر خبره ای که اشتباه کرد را کنار می‌گذاریم. هر زمان که این مرحله تمام شود؛ یعنی تمام خبره‌ها کنار گذاشته شوند، دوباره همه را از اول وارد الگوریتم کنیم و مرحله بعدی شروع می‌شود.

قضیه ۲. *الگوریتم بالا حداکثر $(M+1)(\lg n+1)$ بار اشتباه می‌کند.*

اثبات. مشابه حالت قبل، در هر مرحله حداکثر $\lg n+1$ بار اشتباه می‌کنیم. همچنین چون در هر مرحله تمام خبرگان اشتباه می‌کنند پس حداکثر می‌توانیم $M+1$ مرحله داشته باشیم و در مرحله آخر تعداد خبرگان به صفر نمی‌رسد. پس در کل تعداد اشتباهات الگوریتم حداکثر برابر $M(\lg n+1)+\lg n$ خواهد بود. □

تمرین : فرض کنید پیش‌آمدها به‌جای حالت دودویی، k تایی باشند. نشان دهید در این حالت هم روشی وجود دارد که $O(M\lg n)$ بار اشتباه کند.

الگوریتم اکثریت وزن دار[۷]

تاکنون، خبره‌هایی که یک‌بار اشتباه کرده بودند را در هر مرحله کاملاً کنار می‌گذاشتیم و به نظر بقیه توجه می‌کردیم و کرانی که پیدا کردیم، برا اساس تعداد خبرگان لگاریتمی بود. حال می‌خواهیم الگوریتمی ارائه دهیم که ضریب $\log n$ در حالت قبلی را به عدد ثابت تقلیل دهد. برای رسیدن به این هدف کافی است به نظرات خبرگان هوشمندانه‌تر توجه کنیم و خبره‌ای که یک‌بار اشتباه کرد را بلافاصله کنار نگذاریم.

برای این کار، به هر خبره یک وزن نسبت می‌دهیم. هر بار که خبره‌ای اشتباه کرد، وزن او را در $\frac{1}{2}$ ضرب می‌کنیم. اکنون اکثریت را وزن‌دار انتخاب می‌کنیم؛ یعنی در هر مرحله اگر جمع وزن خبرانی که بالا را پیش‌بینی می‌کنند بیشتر بود، بالا را پیش‌بینی می‌کنیم وگرنه پایین را پیش‌بینی می‌کنیم.

با همین ایده ساده‌ی برخورد «نرم‌تر» با اشتباهات، به کران بهتری می‌رسیم:

قضیه ۳. *اگر در دنباله‌ای از روزها، بهترین خبره M بار اشتباه کند، الگوریتم اکثریت وزن‌دار حداکثر*

$$2.41(M + \log n)$$

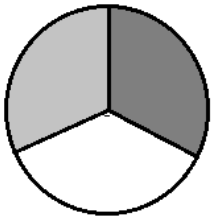
بار اشتباه می کند.

اثبات. فرض کنید Φ_t جمع وزن تمام خبرگان بعد از t روز باشد (تابع پتانسیل یا وزن).

ادعا : هر بار که الگوریتم WM اشتباه می کند، داریم:

$$\phi_t \leq \frac{3}{4}\phi_{t-1}$$

برهان: از روی شکل 1، فرض کنید ناحیه رنگی وزن اکثریتی باشد که اشتباه کردند. بعد از این مرحله نیمی از این ناحیه دور ریخته می‌شود.



شکل ۱: ناحیه رنگی مربوط به وزن بخشی از خبرگان است که اشتباه کرده‌اند و ناحیه تیره‌تر نیمی از آن است که کنار گذاشته می‌شود

حال فرض کنید تعداد کل اشتباهات الگوریتم، m باشد و فرض کنید در ابتدا به همه‌ی خبره‌ها وزن یک بدهیم. آنگاه چون جمع وزن خبرگان در روز آخر از وزن بهترین خبره در روز آخر بیشتر خواهد بود داریم:

$$(\frac{1}{2})^M \leq \Phi_{final} \leq (\frac{3}{4})^m \Phi_0 = (\frac{3}{4})^m.n$$

$$\Rightarrow (\frac{1}{2})^m \leq (\frac{3}{4})^m.n \Rightarrow m \leq \frac{1}{\log \frac{4}{3}}(M + \log n)$$

□

[7] Weighted Majority Algorithm, WM

تمرین : نشان دهید اگر هنگام اشتباه یک خبره، وزن او را در $(1-\epsilon)$ ضرب کنیم، آنگاه تعداد اشتباهات الگوریتم حداکثر برابر

$$2(1+\epsilon)M + O(\frac{\log n}{\epsilon})$$

می شود و عدد ثابت 2 مقدار بهینه است.

الگوریتم اکثریت وزن دار تصادفی[۸]

در این الگوریتم، ابتدا وزن همه خبره‌ها را 1 قرار می‌دهیم و هربار گزینه بالا را با احتمال $\frac{\sum_{j:up} w_j}{\sum w_j}$ انتخاب می‌کنیم. سپس در هر مرحله وزن خبرگانی که اشتباه پیش‌بینی کرده‌اند را در ضریب $(1-\epsilon)$ ضرب می‌کنیم. (یک شیوه بیان دیگر این روش این است که هر خبره را با احتمالی متناسب با وزن او انتخاب کنیم و پیش‌بینی خود را متناسب با پیش‌بینی آن خبره انجام دهیم.)

قضیه ۴. *فرض کنید $\epsilon \leq \frac{1}{2}$ باشد. آنگاه اگر در دنباله‌ای از روزها بهترین خبره M اشتباه داشته باشد، امید ریاضی تعداد اشتباهات الگوریتم* RWM *حداکثر برابر $(1+\epsilon)M + \frac{\ln n}{\epsilon}$ است.*

اثبات. تعریف کنید $\Phi_t := \sum w_i^t$ جمع وزن‌ها در مرحله t ام باشد. فرض کنید F_t کسری از کل وزن‌ها باشد که مربوط به خبرگانی است که در آن مرحله اشتباه کرده‌اند. آنگاه احتمال خطا کردن RWM در مرحله t برابر F_t است. همچنین امید ریاضی تعداد اشتباهات الگوریتم در مرحله t هم برابر F_t است.

در هر مرحله برای تغییرات ϕ داریم:

$$\Phi_t = \Phi_{t-1}(1-\epsilon F_t) \Rightarrow (1-\epsilon)^M \leq \Phi_T = n.\Pi_{t=1}^{T}(1-\epsilon F_t) \leq ne^{-\epsilon\sum F_t}$$

$$\Rightarrow (1-\epsilon)M \leq ne^{-\epsilon\sum F_t} \Rightarrow \epsilon\sum F_t \leq M(-\ln(1-\epsilon)) + \ln n$$

حال چون داریم $-\ln(1-\epsilon) \leq \epsilon + \epsilon^2$ و چون $\sum F_t$ برابر امید ریاضی خطای کل RWM است، پس:

$$\epsilon\sum F_t \leq M(\epsilon+\epsilon^2) + \ln n \Rightarrow E[\text{number of errors}] \leq (1+\epsilon)M + \frac{\ln n}{\epsilon}$$

□

حال برای محاسبه نرخ اشتباهات الگوریتم RWM در هر روز داریم:

$$E[\text{number of errors}] \leq \text{optimum number of errors} + \epsilon T + \frac{\ln n}{\epsilon}$$

$$E[\text{error rate}] \leq \text{optimum error rate} + \epsilon + \frac{\ln n}{\epsilon T}$$

برای کمینه کردن نرخ اشتباه در هر روز اگر قرار دهیم $\epsilon = \sqrt{\frac{\ln}{T}}$ آنگاه:

$$E[\text{error rate}] \leq \text{optimum error rate} + 2\sqrt{\frac{\ln}{T}}$$

نکته: نتایج بدست آمده تاکنون برای وقتی بود که پیش‌بینی هر خبره یا درست بود یا غلط. می‌توان ثابت کرد که اگر هر خبره پیش‌بینی خود را در مجموعه‌ای بزرگ‌تر انجام دهد و همچنین ضرر خبره iام هم بصورت $c_i \in [0,1]$ باشد و در هر مرحله قرار دهیم $w_i = w_i(1-\epsilon c_i)$؛ نتایج همچنان برقرار است.

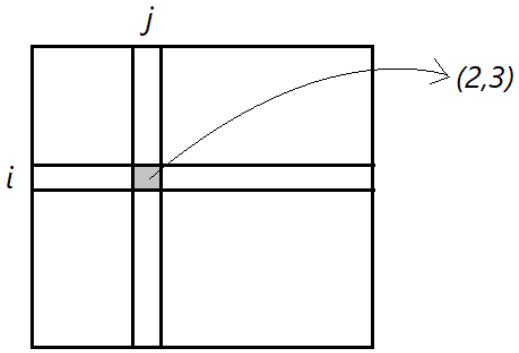
[۸] Randomized weighted majority algorithm, RWM

اثبات قضیه min-max

بازی‌های دو نفره جمع صفر

فرض کنید یک بازی دو نفره داریم. این بازی توسط یک جدول $n \times n$ نمایش داده می‌شود که درایه (i, j) آن بیانگر سود نفر اول است اگر بازیکن اول استراتژی i و بازیکن دوم استراتژی j را اجرا کند.

نفر اول همواره می‌خواهد به‌گونه‌ای بازی کند که سود خود را بیشینه کند. نفر دوم می‌خواهد سود نفر اول را کاهش دهد. همچنین همواره سود نفر اول برابر ضرر نفر دوم است (به همین خاطر به این بازی، بازی جمع صفر می‌گویند). شکل زیر مربوط به یک بازی دو نفره در حالت کلی‌تر است که در آن اگر نفر اول استراتژی iام و نفر دوم استراتژی j ام را اجرا کند آنگاه به ترتیب سود بازیکنان برابر 2 و 3 خواهد شد.



شکل ۲: جدول مربوط به یک بازی دو نفره

حالت تصادفی: در این حالت به هر بازیکن یک بردار احتمال نسبت می‌دهیم بطوری که درایه i ام آن مربوط به احتمال این‌که بازیکن طبق استراتژی i بازی کند باشد. اگر ماتریس استراتژی صورت سوال را M و بردار مربوط به بازیکنان را به ترتیب p و q در نظر بگیریم، آنگاه امید ریاضی سود نفر اول برابر $p^T Mq = \sum p_i q_j M_{ij}$ خواهد بود.

حالت اول: فرض کنید نفر اول در ابتدای بازی بردار احتمال استراتژی خود را اعلام کند. پس نفر دوم از بردار احتمال نفر اول مطلع می‌شود و بر حسب آن تصمیم‌گیری می‌کند؛ یعنی برای p اعلام شده، q را طوری انتخاب می‌کند که امید ریاضی سود نفر اول را کمینه کند. در نتیجه هدف بازیکن اول این است که مقدار $\min_q\{p^T Mq\}$ را بیشینه کند (در این صورت می‌توانیم بازی نفر دوم را احتمالاتی در نظر نگیریم.). یعنی اگر تعریف کنیم $V_R = \max_p \min_q p^T Mq$ آنگاه سود نفر اول برابر با V_R خواهد بود.

حالت دوم: در حالت دوم فرض کنید بازیکن دوم (بازیکن ستونی) اولین بازی را انجام دهد و بازیکن اول از بردار احتمال بازیکن دوم خبر دار شود. پس بازیکن دوم می‌خواهد مقدار $\max_p\{p^T Mq\}$ را کمینه کند. پس اگر تعریف کنیم $V_C = \min_q \max_p p^T Mq$ آنگاه سود نفر اول برابر V_C خواهد شد.

قضیه ۵. *min-max*

ثابت کنید در بازی دو نفره جمع صفر داریم:

$$V_C = V_R$$

اثبات. رابطه $V_R \leq V_C$ به‌وضوح برقرار است؛ زیرا اگر نفر ستونی استراتژی خود را آشکارکند مطمئناً به نفع بازیکن سطری خواهد شد و ممکن است در نهایت بتواند سود بیشتری را بدست بیاورد.

حال با برهان خلف فرض کنید δای وجود دارد به‌طوری‌که:

$$V_R = V_C - \delta, \quad \delta > 0$$

می‌توانیم بدون کاسته شدن از کلیت داریه‌های ماتریس بازی را به بازه $[0, 1]$ تصویر می‌کنیم.

حال فرض می‌کنیم ستون ها، خبرگان باشند. بازی را در T مرحله انجام می‌دهیم و در ابتدا وزن همه را یک قرار می‌دهیم. اتفاقی که در هر روز میفتد، معادل سطرها خواهد بود؛ یعنی عدد هر درایه (i, j) مربوط به میزان اشتباه خبره j در روز i است. پس ما هر بار طبق الگوریتم RWM یک ستون را انتخاب می‌کنیم و سپس سطری را انتخاب می‌کنیم که با بازی بهینه بازیکن سطری متناسب باشد. در واقع استراتژی بازیکن سطری مربوط به توزیع احتمالی است که در هر مرحله روی ستون‌های مختلف داریم. بعد از T گام امید ریاضی سود بازیکن سطری کم‌تر مساوی $(M + \epsilon T + \frac{\ln n}{\epsilon})$ خواهد بود که M ضرر بهترین خبره بعد از T گام است. (اگر کل استراتژی‌هایی که بازیکن سطری در این T گام انجام داده را در نظر بگیریم، یک بردار احتمال بدست می‌آوریم. سپس بهترین خبره ستونی است که اگر کلا با آن بازی می‌کردیم، بهینه بود.)

از طرفی سود بازیکن سطری همواره بزرگ‌تر مساوی $V_C.T$ است؛ زیرا در هرگام اجرای الگوریتم فرض کردیم بازیکن سطری بردار احتمال آن مرحله از الگوریتم RWM را می‌داند و بهترین بازی ممکن را انجام می‌دهد. پس اگر بازیکن ستونی در ابتدا استراتژی خود را آشکار کند، بازیکن سطری می‌تواند جوری بازی کند که حداقل به اندازه V_C سود کند.

هم‌چنین اگر بازیکن سطری اول بازی کند، آنگاه بازیکن ستونی می‌تواند به‌گونه‌ای بازی کند که سود بازیکن اول حداکثر V_R شود. پس یعنی $M \leq V_R T$. در کل:

$$\Rightarrow (V_C - V_R)T \leq \epsilon T + \frac{\ln n}{\epsilon}$$

حال می‌توانیم فرض کنیم $T = \frac{\ln n}{\epsilon^2}$ و $\epsilon = \frac{\delta}{3}$:

$$\Rightarrow \delta T \leq \frac{2\delta}{3} T$$

که با مثبت بودن دلتا در تناقض است. پس باید داشته باشم:

$$V_C = V_R$$

□

مراجع

[۱] درس آنالیز الگوریتم، جلسه اضافه دوم، مسأله خبرگان و روش به‌روزرسانی ضربی وزن‌ها، مرتضی علیمی

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه اضافه ۳: الگوریتم‌های تصادفی

نگارنده: فراز فرح‌وش

در جلسات گذشته نمونه‌هایی از الگوریتم‌های تصادفی را دیدیم. در چند جلسه‌ی آینده این الگوریتم‌ها را بیشتر بررسی می‌کنیم. استفاده از تصادف در علوم کامپیوتر ایده‌ی مهمی است و کاربردهای فراوانی دارد. از کاربردهای مشهور آن می‌توان به استفاده از تصادف برای جلوگیری از پیش آمدن بدترین حالت در الگوریتم مرتب‌سازی سریع [۱] اشاره کرد .

یکی از کاربردهای مهم تصادف ، استفاده از توابع درهم‌ساز و یک ایده‌ی مشابه به آن تحت عنوان ایده‌ی انگشت‌نگاری است [۲] . در این ایده‌ها، یک فضای بزرگ را به فضایی کوچک‌تر تصویر می‌کنیم. این سبک استفاده از تصادفی‌سازی کاربردهای بسیاری دارد از جمله:

- داده‌ساختارهای سریع و با حجم اشغالی کم
- اثبات‌های تعاملی
- تطبیق رشته[۳]

کاربرد مهم دیگر نمونه‌گیری تصادفی [۴] است. در این روش یک مجموعه‌ی بزرگ وجود دارد و ما می‌خواهیم تعدادی از اعضای آن (که امیدواریم typical باشد) را انتخاب کرده و عملیاتی روی آن‌ها انجام بدهیم. امید این است که نتیجه به دست آمده با نمونه‌گیری تصادفی به نتیجه‌ی محاسبه شده با کل داده نزدیک باشد. این کار می‌تواند به دلیل محدودیت حافظه یا کند بودن محاسبه با کل مجموعه انجام شود.

از مسائل دیگری که در آن استفاده از تصادف راهگشا است می‌توان به مسائل توزیع یا پخش کردن اشاره کرد. یک نمونه آن مسئله‌ی پخش کردن بار [۵] است که در آن سعی می کنیم بین چند پردازنده که محاسبات موازی انجام می‌دهند، کارها را به طوری تقسیم کنیم که بار هر پردازنده با بقیه تقریبا یکسان باشد.

کاربرد دیگر شکستن تقارن است. گاهی در سیستم‌های توزیع‌شده برای پیش‌رفتن الگوریتم یا پروتکل، لازم است تقارن موجود در سیستم شکسته شود و در این مواقع استفاده از تصادف می‌تواند کارگشا باشد.

هرگاه راجع به زمان اجرای متوسط الگوریتم‌های تصادفی صحبت می‌کنیم، منظور متوسط‌گیری روی انتخاب‌های تصادفی الگوریتم است؛ یعنی یک الگوریتم تصادفی را انتخاب و فیکس می‌کنیم و دشمن [۶] ورودی را طوری طراحی می‌کند که برای الگوریتم تا جای ممکن نابهینه است. زمان اجرای متوسط این الگوریتم همان متوسط‌گیری روی انتخاب‌های تصادفی خود الگوریتم روی بدترین ورودی ممکن است.

[1] Quick sort
[2] finger printing
[3] string matching
[4] random sampling
[5] load balancing
[6] adversary

گاهی نیز منظور از زمان اجرای متوسط، متوسط گیری روی ورودی‌ها و تصادفی در نظر گرفتن ورودی‌ها با یک توزیع مشخص است. به این کار تحلیل حالت متوسط [۷] گفته می‌شود که هدف ما در این بحث نیست.

۱ مثال ساده برای شروع

ابتدا یک مسئله‌ی ساده را بررسی می‌کنیم.

مسئله:

آرایه‌ای به طول n که $\frac{n}{4}$ اعضای خانه‌های آن یک و بقیه صفر است، داریم. می‌خواهیم عدد i را طوری بیابیم که $A[i] = 1$.

الگوریتم بدیهی و قطعی $(O(n))$:

یک حلقه‌ی for از اول تا آخر آرایه را بررسی می‌کند و خانه‌ای را که مقدار آن یک باشد بازمی‌گرداند:

```
DETEMINISTIC(A)
1  for i = 1 to n
2      if A[i] == 1
3          return i
```

این الگوریتم در بدترین حالت باید $\frac{3}{4}n + 1$ خانه را بررسی کند.

می‌توان نشان داد که برای هر الگوریتم قطعی دیگر، مهاجم می‌تواند طوری ورودی را تنظیم کند که مشابه الگوریتم بالا دیدن حداقل $\frac{3}{4}n + 1$ خانه لازم باشد.

الگوریتم تصادفی:

در هر گام عدد رندم i در بازه‌ی ۱ تا n انتخاب می‌کنیم و اگر $A[i] = 1$ باشد، i را بازمی‌گردانیم.

```
RANDOMIZED(A)
1  repeat:
2      i = random(1, n)
3      if A[i] == 1
4          return i
```

تحلیل: چون احتمال موفقیت در هر تکرار برابر $\frac{1}{4}$ است، به طور متوسط چهار بار تکرار لازم است تا الگوریتم به نتیجه برسد (یک خانه با مقدار ۱ یافت شود). پس زمان اجرای متوسط این الگوریتم ۴ مرحله است.

ایراد الگوریتم: زمان اجرای بدترین حالت این الگوریتم معلوم نیست و حتی محدود [۸] نیز نیست. هر چند احتمال این‌که تعداد گام‌ها زیاد باشد(مثلا ۱۰۰) بسیار کم است ولی همچنان از لحاظ تئوری ممکن است.

[7] average case analysis
[8] bounded

این امر باعث می‌شود که الگوریتم‌های تصادفی به دو دسته تقسیم شوند:

- **لاس‌وگاس:** این الگوریتم‌ها همیشه جواب درست می‌دهند و معمولا سریع هستند ولی ممکن است زمان اجرای بدترین حالت آن حتی نامحدود باشد.
- **مونت‌کارلو:** این الگوریتم‌ها معمولا درست هستند(نه لزوما همیشه) ولی همیشه سریع هستند.

نکته: هر الگوریتم لاس‌وگاس قابل تبدیل به یک الگوریتم مونت‌کارلو است. برای مثال برای الگوریتم بالا می توان حد بالا را ۱۰۰ قرار داد و الگوریتم جدید مطابق زیر می‌شود.

MONTE CARLO(A)

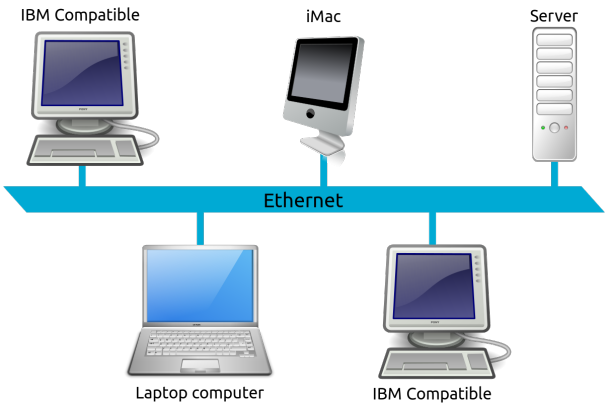
```
1  repeat 100 time:
2      i = random(1, n)
3      if A[i] == 1
4          return i
5  return 1
```

نکته: برعکس نکته قبل درست نیست و هر الگوریتم مونت‌کارلو را نمی‌توان به الگوریتم لاس‌وگاس تبدیل کرد. بنابراین داشتن الگوریتم لاس‌وگاس نتیجه‌ی قوی‌تری نسبت به داشتن الگوریتم مونت‌کارلو برای یک مسئله است.

مثال بعدی درباره‌ی ایده‌ی شکستن تقارن است.

۲ اترنت [۹]

یک سیم یا خط ارتباطی داریم که چند کامپیوتر به آن وصلند و هر کامپیوتر می‌تواند پیامی را روی این سیم بفرستد و بقیه پیام را دریافت کنند. اگر در یک لحظه بیش از یک کامپیوتر روی این سیم پیام بفرستد تداخل ایجاد می‌شود و همه‌ی پیام‌های ارسالی در آن لحظه خراب می‌شوند. بنابراین نیازمند روشی هستیم که تا حد امکان تداخل‌ها را کم کند و در هر زمان حداکثر یک کامپیوتر پیام بفرستد.



[9] Ethernet

حال هماهنگی با روش‌های قطعی [10] به دو صورت محتمل است:

- ارسال پیام برای هماهنگی: ارسال پیام برای هماهنگی خود نیازمند حل این مشکل است! بنابراین این روش ممکن نیست.
- الگوریتم قطعی برای هر کامپیوتر به صورت مستقل: در این حالت نیز اگر دو کامپیوتر در یک لحظه هر دو در یک حالت [11] مشابه باشند، تا ابد در حالات یکسان باقی می‌مانند و تداخل تا ابد ادامه خواهد داشت.

پس راه‌حل قطعی نمی‌تواند مشکل ما را حل کند. باید از روش‌های تصادفی استفاده کنیم.

یک راه‌حل تصادفی ساده:

هر کامپیوتر در هر واحد زمان با احتمال p پیغام می‌فرستد.

حال باید احتمال این که در یک لحظه دقیقا یک کامپیوتر پیام بفرستد را محاسبه کنیم . این احتمال برابر است با:

$$P[\text{exactly one computer sends data}] = np(1-p)^{n-1}$$

می‌خواهیم این عبارت را بر حسب p ماکسیمم کنیم. ماکسیمم این عبارت در $p = \frac{1}{n}$ رخ می‌دهد. با جای‌گذاری داریم:

$$P[\text{exactly one computer sends data}] = (1-\frac{1}{n})^{n-1} \approx \frac{1}{e}$$

پس در هر لحظه با احتمال یک عدد ثابت پیام درست می‌رسد و این نتیجه مثبتی است.

این الگوریتم چندین دغدغه برای پیاده سازی دارد، از جمله این که کامپیوترها تعدادشان را نمی‌دانند و در قرار دادن p به مشکل می‌خورند. برای حل این مشکل هر کامپیوتر تخمینی از n می‌زند(در ابتدا $n = 1$) و در ادامه هرگاه به تداخل برخورد کرد، تخمین خود را از n دو برابر می‌کند. می‌توان مشاهده کرد که پس از $\log n$ بار تداخل، تخمین تقریبا درست خواهد بود.

در عمل می‌توان در صورت عدم تداخل تخمین را از n را به صورت ملایمی کاهش داد تا الگوریتم عملکرد بهتری داشته باشد.

۳ مساله جمع‌آوری کوپن [12]

در ادامه می‌خواهیم مساله جمع‌آوری کوپن را بررسی کنیم.

مسئله:

اعداد 1 تا n وجود دارد و در هر مرحله(واحد زمان) یک عدد به طور تصادفی انتخاب می‌کنیم و می‌بینیم. چند مرحله طول می‌کشد تا تمامی اعداد را ببینیم؟

تحلیل اول:

متغیر تصادفی X را به صورت تعداد مراحل لازم برای دیدن دیدن همه‌ی اعداد تعریف می‌کنیم. در واقع می خواهیم امید ریاضی X را بیابیم.

[10] deterministic
[11] state
[12] Coupon Collecting Problem

برای این منظور n متغیر تصادفی کمکی بدین صورت تعریف می‌کنیم:

X_i متغیر تصادفی مدت زمانی است که پس از دیدن $i-1$ عدد متمایز طول می‌کشد تا عدد متمایز جدیدی ببینیم.

به وضوح داریم:

$$X = \sum_{i=1}^{n} X_i \ , \ X_1 = 1$$

از خطی بودن امید ریاضی استفاده می‌کنیم و داریم:

$$E[X] = \sum_{i=1}^{n} E[X_i]$$

برای محاسبه‌ی $E[X_i]$ توجه کنید که احتمال این که یک مرحله بعد از دیدن $i-1$ عدد متمایز، عدد متمایز جدیدی ببینیم برابر است با $\frac{n}{n-i+1}$ و همچنین X_i توزیع هندسی دارد. بنابراین داریم:

$$E[X_i] = \frac{n}{n-(i-1)}$$

در نتیجه:

$$E[X] = \sum_{i=1}^{n} \frac{n}{n-i+1} = n(1 + \frac{1}{2} + ... + \frac{1}{n}) = nH_n = \Theta(n \log n)$$

تحلیل دوم:

ابتدا احتمال این که بعد از t گام کوپن شماره یک را ندیده باشیم محاسبه می‌کنیم. می‌دانیم احتمال این که بعد از یک گام، کوپن مدنظر را ندیده باشیم برابر $1 - \frac{1}{n}$ است. بنابراین احتمال این‌که بعد از t کوپن شماره یک را ندیده باشیم برابر $(1 - \frac{1}{n})^t$ خواهد بود. از طرفی می‌دانیم:

$$1 + x \leq e^x$$

بنابرین:

$$P[\text{not seeing coupon 1 after t steps}] = (1 - \frac{1}{n})^t \leq e^{-\frac{t}{n}}$$

با قراردادن $t \geq \beta n \ln n$ داریم:

$$P \leq e^{-\frac{t}{n}} \leq n^{-\beta}$$

حال می‌دانیم احتمال اجتماع یک سری پیشامد، کوچک‌تر یا مساوی مجموع احتمال آن‌هاست [۱۳] :

$$Pr[\bigcup A_i] \leq \sum Pr[A_i]$$

پس احتمال این‌که بعد از t مرحله حداقل یک کوپن را ندیده باشیم، حداکثر n برابر مقدار بالا و برابر $n^{1-\beta}$ است.

برای مثال برای $t \geq 10n \ln n$ مرحله، احتمال عدم موفقیت n^{-9} خواهد بود که احتمال پایینی است.

به طور کلی منظور از الگوریتمی که با احتمال بالا موفق است این است که احتمال خطا در آن الگوریتم حداکثر برابر $n^{-\alpha}$ باشد.

نکته: اگر چندجمله‌ای بار یک الگوریتم با احتمال موفقیت بالا را پشت سرهم اجرا کنیم، همچنان الگوریتم ما احتمال موفقیت بالایی دارد.

[13] union bound

۴ مسئله توپ و سطل [۱۴]

مساله‌ی بعدی که بررسی خواهیم کرد مساله توپ و سطل نام دارد.

مسئله:

n توپ و n سطل داریم. هدف پخش‌کردن متوازن توپ‌ها در سطل‌هاست.

حالت اول:

فرض کنید هر توپ را به طور تصادفی در یک سطل بیندازیم. داریم:

$$E[\text{number of balls in bins}] = 1$$

می‌خواهیم نشان دهیم با احتمال بالا (طبق تعریفی که در بخش قبل کردیم) تعداد توپ‌ها در یک سطر از یک حدی بیشتر نیست.

احتمال این که دقیقا k توپ در یک سطل معین باشد را محاسبه می‌کنیم:

$$P[\text{k balls in bit number t}] = \binom{n}{k}(\frac{1}{n})^k(1-\frac{1}{n})^{n-k} \leq \binom{n}{k}(\frac{1}{n})^k$$

حال از نامساوی $\binom{n}{k} \leq (\frac{en}{k})^k$ استفاده می‌کنیم و داریم:

$$P[\text{k balls in bit number t}] \leq \binom{n}{k}(\frac{1}{n})^k \leq (\frac{en}{k})^k(\frac{1}{n})^k = (\frac{e}{k})^k$$

حال مقدار این نامساوی را به ازای $k = O(\frac{\log n}{\log\log n})$ محاسبه می‌کنیم:

$$(\frac{e}{k})^k \leq (\frac{\log\log n}{\log n})^{O(\frac{\log n}{\log\log n})} \leq (\frac{\sqrt{\log n}}{\log n})^{O(\frac{\log n}{\log\log n})} = (\frac{1}{\log n})^{O(\frac{\log n}{\log\log n})} = [(\frac{1}{\log n})^{\frac{1}{\log\log n}}]^{O(\log n)}$$

که برابر است با:

$$[(\frac{1}{\log n})^{\frac{1}{\log\log n}}]^{O(\log n)} = (\frac{1}{e})^{O(\log n)} = n^{-O(1)}$$

که همان تعریف احتمال موفقیت بالاست و با استفاده از کران اجتماع، با احتمال بالا بیش‌ترین تعداد توپ در یک سبد کمتر از $O(\frac{\log n}{\log\log n})$ خواهد بود.

سوال: اگر به جای انتخاب رندم کمی هوشمندی به خرج بدهیم این حد بالا چه تغییری می‌کند؟

[14] Balls in Bins

حالت دوم:

برای هر توپ به طور تصادفی دو سطل را انتخاب می‌کنیم و توپ را داخل سطلی می‌اندازیم که تا اینجا توپ کمتری دارد.

می‌توان نشان داد که در این حالت بیشینه تعداد توپ‌های یک سطل در پایان $O(\log \log n)$ خواهد بود که بسیار حد بهتری نسبت به حالت قبل است. (به این نکته قدرت انتخاب دوگانه [۱۵] می‌گویند.)

تحلیل دقیق این موضوع خارج از بحث ماست ولی سعی خواهیم کرد با یک تحلیل نادقیق شهودی نسبت به مسئله ارائه کنیم.

تحلیل نادقیق:

β_i را برابر تعداد سطل‌هایی که در پایان حداقل i توپ دارند، تعریف می‌کنیم. داریم:

$$\beta_2 \leq \frac{n}{2}$$

تلاش می‌کنیم کرانی برای β_{i+1} برحسب β_i بیابیم. لازمه‌ی $(i+1)$ ـ تایی شدن یک سطل انتخاب دو سطل است که هر دو حداقل i توپ داشته باشند. بنابراین:

$$\beta_{i+1} \leq n(\frac{\beta_i}{n})^2 = \frac{\beta_i^2}{n}$$

حال به صورت استقرایی و شروع از β_2 داریم:

$$\beta_i \leq (\frac{n^2}{2^{2^i}})$$

حال با قرار دادن $\log \log n$ داریم:

$$\beta_{\log \log n} \leq 1$$

این تحلیل نشان می‌دهد که ماکسیمم لود یک سطل $O(\log \log n)$ است.

[15] Power of two choices

۵ مساله برش کمینه‌ی سراسری در گراف‌های بدون جهت

قبلا دیدیم که با استفاده از مسئله جریان بیشینه [۱۶] می‌توان مسئله را برای حالتی که حتما دو راس مشخص در دو طرف این برش باشند حل کرد. بنابراین مسئله برش سراسری را می‌توان با اجرای $\binom{n}{2}$ بار الگوریتم جریان بیشینه حل کنیم(به ازای هر دو راس ممکن). می‌توان به سادگی دید در گراف بدون جهت اجرای $n-1$ بار الگوریتم جریان بیشینه نیز کافی است(با ثابت نگه داشتن راس مبدا و تغییر راس مقصد).

از آن‌جایی که بهترین الگوریتم برای جریان بیشینه $O(mn)$ است، الگوریتمی با زمان اجرای $O(mn^2)$ برای مسئله برش کمینه سراسری وجود دارد.

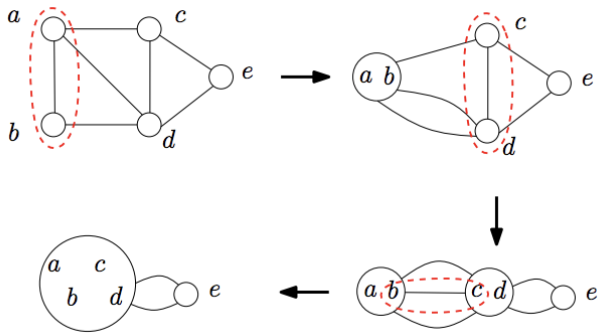
الگوریتم‌های قطعی دیگری وجود دارند که با زمان $O(mn)$ مسئله برش کمینه سراسری را حل می‌کنند. [۱۷] [۱۸]

حال سعی می‌کنیم الگوریتم تصادفی با زمان اجرای بهتر بدهیم:

الگوریتم تصادفی Karger-Stein :

فرآیند منقبض کردن یک یال: منظور از منقبض کردن یک یال این است که دو سر آن را بر هم منطبق کنیم.

مثالی از فرآیند منقبض کردن:



فرآیند منقبض کردن را تا جایی ادامه می‌دهیم که تنها دو راس باقی بماند و مجموعه‌ی یال‌های بین این دو راس را به عنوان برش کمینه خروجی می‌دهیم.

ایده‌ی الگوریتم:

امید این است که یال‌های برش کمینه را در فرآیند منقبض کردن انتخاب نکنیم و در نهایت در حالتی که دو راس باقی مانده برش کمینه را به دست بیاوریم. برای افزایش احتمال موفقیت، می‌توان چند بار الگوریتم را اجرا کنیم و میان نتایج آن‌ها مینیمم‌گیری کرد.

نکته: اگر احتمال موفقیت یک الگوریتم p باشد، کافی است الگوریتم را $\Theta(\frac{\log n}{p})$ بار اجرا کنیم تا الگوریتم با احتمال بالا موفق باشد(با تعریفی که انجام شده)

اثبات:

$$Pr[\text{wrong result after t time}] = (1-p)^t \leq e^{-pt} = e^{-c\ln n} = n^{-c}$$

[16] max-flow
[17] Hao-Orlin algorithm
[18] Nagamuchi-Ibaraki algorithm

تحلیل الگوریتم:

زمان اجرای الگوریتم برای یک بار $O(m)$ است.

لم ۱. *اگر اندازه‌ی برش کمینه‌ی یک گراف c باشد، گراف حداقل $\frac{nc}{2}$ یال دارد.*

اثبات. درجه‌ی هر راس حداقل c است (در غیر این‌صورت یال‌های یک راس با درجه کمتر از c یک برش با تعداد یال کمتر است که تناقض است) بنابراین تعداد کل یال‌ها حداقل $\frac{nc}{2}$ است.. □

حال احتمال خطا را محاسبه می‌کنیم:

$$Pr[\text{error in edge selection}] \leq \frac{c}{\frac{nc}{2}} = \frac{2}{n}$$

بنابراین:

$$Pr[\text{right result from algorithm}] \geq (1-\frac{2}{n})(1-\frac{2}{n-1})...(1-\frac{2}{3}) = \frac{n-2}{n}\frac{n-3}{n-1}...\frac{3-2}{3} = \frac{2}{n(n-1)}$$

در ادامه از $\log n$ صرف‌نظر می‌کنیم.(به یک احتمال موفقیت ثابت راضی می‌شویم)

برای بالا بردن احتمال موفقیت الگوریتم بالا باید $\Theta(n^2)$ بار تکرار شود و زمان اجرای کل الگوریتم برابر $O(n^2m)$ خواهد بود که بدتر از الگوریتم‌های قطعی است!

مشکل: در مراحل پایانی الگوریتم احتمال موفقیت بسیار پایین می‌آید.

ایده: تا جایی الگوریتم تصادفی را اجرا می‌کنیم و سپس از الگوریتم قطعی استفاده می‌کنیم.

Contract Algorithm 2 :

الگوریتم را اجرا می‌کنیم تا جایی که به k راس برسیم و سپس الگوریتم قطعی را اجرا می‌کنیم.

چون احتمال موفقیت وقتی که تا k راس ادامه بدهیم برابر $\Omega(\frac{k^2}{n^2})$ است، تعداد تکرارهای لازم برابر $O(\frac{n^2}{k^2})$ خواهد بود.

CA2(G)

1 repeat $\frac{n^2}{k^2}$ time:

2 contract G to k vertex G_k

3 Hao_Orlin(G_k)

زمان اجرا:

$$O(\frac{n^2}{k^2}(m+k^3)) = O(n^2(\frac{m}{k^2}+k))$$

حال باید عبارت بالا را برحسب k کمینه کرد. k را برابر $m^{\frac{1}{3}}$ قرار می‌دهیم و زمان اجرا برابر $O(n^2m^{\frac{1}{3}})$ خواهد بود که بهتر از الگوریتم قطعی است.

حال که الگوریتم بهتری برای پیداکردن برش کمینه داریم می‌توان از آن به جای الگوریتم قطعی استفاده کرد:

CA3(G)

1 repeat $\frac{n^2}{k^2}$ time:
2 contract G to k vertex G_k
3 CA2(G_k)

البته مقدار k در این الگوریتم بیشتر از الگوریتم قبل خواهد بود و زمان اجرای آن نیز کم‌تر از $CA2$ خواهد بود.

حال می‌توان این روند را ادامه داد:

CA4(G)

1 repeat $\frac{n^2}{k^2}$ time:
2 contract G to k vertex G_k
3 CA3(G_k)

و الی آخر...

ادامه دادن این روند تا جایی که به بی‌نهایت میل کند، یک الگوریتم بازگشتی نتیجه می‌دهد که آن را Randomized Contraction می‌نامیم. الگوریتم مطابق زیر است:

RCA(G)

1 repeat twice:
2 contract G to $\frac{n}{\sqrt{2}}$ vertex G'
3 RCA(G')

زمان اجرا:

$$T(n) = 2T(\frac{n}{\sqrt{2}}) + O(n^2)$$

که طبق قضیه‌ی مستر داریم:

$$T(n) = O(n^2 log(n))$$

ادعا:

احتمال موفقیت الگوریتم بالا $\Omega(\frac{1}{\log n})$ است که در صورت اثبات زمان اجرای کلی الگوریتم Karger-Stein برابر $O(n^2 \log^3 n)$ خواهد بود.

اثبات ادعا:

$p(n)$ را برابر احتمال موفقیت الگوریتم روی گراف n راسی تعریف می‌کنیم. داریم:

$$p(n) = 1 - (1 - \frac{1}{2}p(\frac{n}{\sqrt{2}}))^2$$

که $\frac{1}{2}$ احتمال خطا در انقباض یال‌ها و $p(\frac{n}{\sqrt{2}})$ احتمال خطا در بخش بازگشتی الگوریتم است.

متغیر کمکی r_k را برابر $p(\sqrt{2}^k)$ تعریف می‌کنیم. داریم:

$$r_2 = 1$$

$$r_k = 1 - (1 - \frac{1}{2}r_{k-1})^2 = r_{k-1} - \frac{1}{4}r_{k-1}^2$$

متغیر z_k را مطابق زیر تعریف می‌کنیم:

$$z_k = \frac{4}{r_k} - 1 \to r_k = \frac{4}{z_k+1}$$

حال رابطه‌ی بازگشتی را بر حسب z_k می‌نویسیم:

$$\frac{4}{z_k+1} = \frac{4}{z_{k-1}+1}(1 - \frac{1}{z_{k-1}+1})$$

پس از کمی ساده‌سازی به روابط زیر می‌رسیم:

$$z_k + 1 = (z_{k-1} + 1)(1 + \frac{1}{z_{k-1}}) = z_{k-1} + 2 + \frac{1}{z_{k-1}}$$

پس داریم:

$$z_k + 1 = z_{k-1} + 1 + \frac{1}{z_{k-1}}$$

و از آن‌جایی که $z_k \geq 1$ داریم:

$$2 \geq z_k - z_{k-1} \geq 1$$

بنابراین:

$$2k \geq z_k \geq k - 1$$

و با جایگذاری در روابط r_k :

$$r_k = \Theta(\frac{1}{k})$$

در نهایت:

$$p(n) = r_{2\log n} = \Theta(\frac{1}{\log n})$$

ادعا ثابت می‌شود.

۶ فرآیند شاخه‌ای [۱۹]

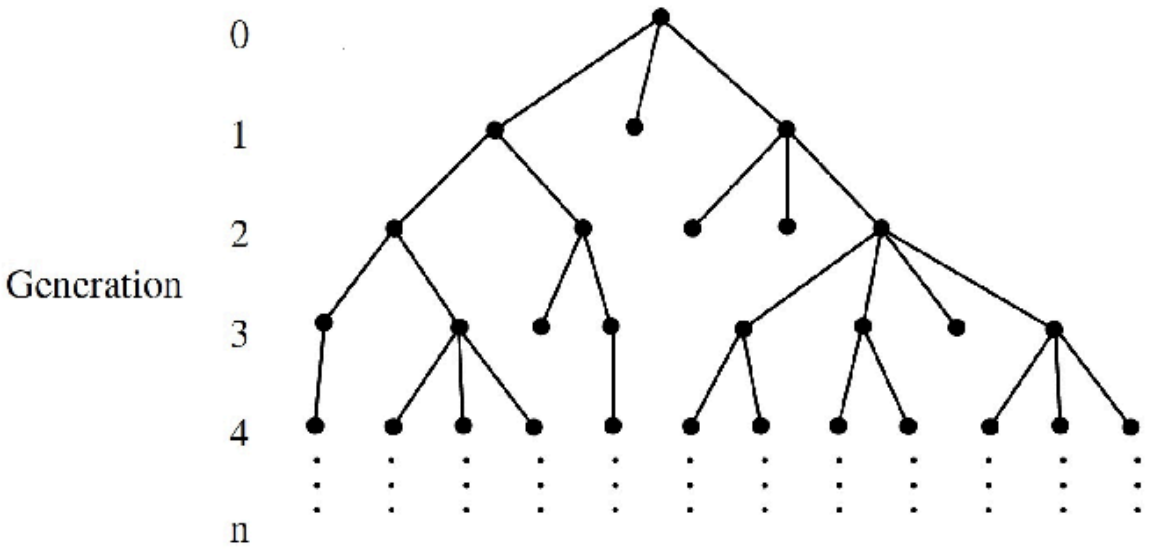
می‌توان به الگوریتم قبل بدین صورت نگاه کرد که یک درخت بازگشت [۲۰] داریم و موفقیت الگوریتم در هر بازگشت به معنای وجود شاخه‌ای است که برش کمینه خراب نشده باشد.

می‌توان دید که امید ریاضی تعداد فرزندهای سالم برای هر نود برابر ۱ است. این درخت نمونه‌ای از مسائلی به نام فرآیند شاخه‌ای یا branching process است.

در این مسائل، نکته‌ی مهم امید ریاضی تعداد فرزندان یک گره است. داریم:

$$E[\text{number of children}] = \begin{cases} \leq 1, & \text{extinction} \\ = 1, & \text{probability of continuation for at least k steps is approximately } \frac{1}{k} \\ > 1, & \text{there is a non-zero probability that process never stops} \end{cases}$$

یک نمونه فرآیند شاخه‌ای در ادامه آورده شده است.



مراجع

[۱] ویدئوی جلسه اضافه سوم (الگوریتم‌های تصادفی)، درس آنالیز الگوریتم

[19] Branching process
[20] recursion tree

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه اضافه ۴: نامساوی‌های مارکف و چبیشف

نگارنده: سعید هدایتیان

در جلسات پیش در مورد الگوریتم های تصادفی صحبت کردیم. به طور خاص، برای مسئله جمع‌آوری کوپن و مسئله توپ و سطل الگوریتم های تصادفی ارائه کردیم. همان طور که دیدیم در تحلیل این الگوریتم ها از روش های خاصی استفاده کردیم که با توجه به ویژگی های خاص و ذاتی مسائل مورد بحث به دست آمده بودند و به صورت کلی، در تحلیل الگوریتم های تصادفی مختلف قابل استفاده نبودند. در این جلسه ابتدا مروری بر مفاهیم پایه‌ای در احتمال خواهیم داشت. سپس نامساوی های مارکف [۱] و چبیشف [۲] را بیان و اثبات می‌کنیم و با استفاده از آن ها در بررسی مسائلی که قبلاً با آن ها مواجه شده بودیم، کاربرد این نامساوی ها در تحلیل الگوریتم های تصادفی دلخواه را نشان می دهیم. در واقع این نامساوی های احتمالاتی به ما در ارائه کرانی برای احتمال دور شدن مقدار یک متغیر تصادفی از میانگینش کمک می‌کنند. در ادامه، یک الگوریتم تصادفی برای محاسبه میانه n عدد ارائه می‌کنیم و آن را تحلیل می‌کنیم. در انتها در مورد پیچیدگی محاسباتی الگوریتم های تصادفی صحبت می کنیم و کلاس های پیچیدگی جدیدی را معرفی می کنیم.

۱ مروری بر مفاهیم پایه‌ای احتمال

تعریف ۱. *امید ریاضی* [۳] (میانگین یا متوسط) متغیر تصادفی گسسته X با نمادهای $\mathbb{E}[X]$ یا μ_X نشان داده می‌شود و به صورت زیر تعریف می‌شود.

$$\mu_X = \mathbb{E}[X] = \sum_{x \in \text{supp}(X)} xPr[X = x]$$

لم ۱. *امید ریاضی خطی است. یعنی به ازای هر دو متغیر تصادفی X و Y،*

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y].$$

تعریف ۲. *واریانس* [۴] متغیر تصادفی گسسته X با نماد $\text{Var}(X)$ نشان داده می‌شود و به صورت زیر تعریف می شود.

$$\text{Var}(X) = \mathbb{E}[(X - \mu_X)^2]$$

لم ۲. *می‌توان نشان داد که تعریف فوق از واریانس معادل تعریف زیر است.*

$$\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}^2[X]$$

لم ۳. *اگر دو متغیر تصادفی X و Y مستقل از یکدیگر باشند، واریانس مجموع آن ها برابر مجموع واریانس آن ها است. یعنی*

$$X \perp Y \implies \text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y).$$

[1] Markov's inequality
[2] Chebyshev's inequality
[3] Expected value
[4] Variance

اثبات.

$$\begin{aligned}\mathrm{Var}(X+Y) &= \mathbb{E}[(X+Y)^۲] - \mathbb{E}^۲[X+Y] = \mathbb{E}[(X+Y)^۲] - (\mathbb{E}[X]+\mathbb{E}[Y])^۲ \\ &= \mathbb{E}[X^۲ + Y^۲ + ۲XY] - \mathbb{E}^۲[X] - ۲\mathbb{E}[X]\mathbb{E}[Y] - \mathbb{E}^۲[Y] \\ &= \mathbb{E}[X^۲] - \mathbb{E}^۲[X] + \mathbb{E}[Y^۲] - \mathbb{E}^۲[Y] + ۲\mathbb{E}[X]\mathbb{E}[Y] - ۲\mathbb{E}[X]\mathbb{E}[Y] \\ &= \mathrm{Var}(X) + \mathrm{Var}(Y).\end{aligned}$$

□

تعریف ۳. *انحراف معیار* [۱] متغیر تصادفی X که با نماد $\sigma(X)$ نمایش داده می‌شود، طبق تعریف برابر جذر واریانس این متغیر تصادفیست.

$$\sigma(X) = \sigma_X = \sqrt{\mathrm{Var}(X)}$$

مثال: توزیع دوجمله‌ای

اگر متغیر تصادفی X دارای توزیع دوجمله‌ای با پارامترهای n و p باشد($X \sim Binomial(n,p)$)، آنگاه $X = \sum_{i=۱}^{n} X_i$ که هر X_i یک متغیر تصادفی برنولی [۲] است که با احتمال p برابر ۱ است و با احتمال $p - ۱$ برابر ۰ است.

در واقع X را می‌توان تعداد *شیرهای* مشاهده شده در پرتاب n سکه در نظر گرفت که هر سکه به احتمال p شیر و به احتمال $p - ۱$ خط می‌آید.

داریم

$$\mathbb{E}[X] = \mathbb{E}[\sum_{i=۱}^{n} X_i] = \sum_{i=۱}^{n} \mathbb{E}[X_i] = np.$$

همچنین چون X_i ها مستقل هستند،

$$\mathrm{Var}(X) = \sum_{i=۱}^{n} \mathrm{Var}(X_i) = n\mathrm{Var}(X_۱) = n(\mathbb{E}[X_۱^۲] - \mathbb{E}^۲[X_۱]) = n(p - p^۲) = np(۱-p).$$

بنابراین می‌توانیم از رابطه زیر استفاده کنیم.

$$\sigma_X = \sqrt{np(۱-p)} \leqslant \sqrt{n}.$$

⊠

مثال: توزیع هندسی

اگر X تعداد دفعات انجام یک آزمایش برنولی با احتمال موفقیت p تا مشاهده اولین موفقیت باشد، X را یک متغیر تصادفی هندسی با پارامتر p گویند. به سادگی می توان دید که $\mathbb{E}[X] = \frac{۱}{p}$. برای محاسبه واریانس X ابتدا $\mathbb{E}[X^۲]$ را محاسبه می‌کنیم.

$$\begin{aligned}&\mathbb{E}[X^۲] = p + (۱-p)(\mathbb{E}[(X+۱)^۲]) = p + ۱ - p(\mathbb{E}[X^۲] + ۲\mathbb{E}[X] + ۱) = ۱ + (۱-p)(\mathbb{E}[X^۲] + ۲\mathbb{E}[X]) \\ &\implies p\mathbb{E}[X^۲] = ۱ + (۱-p)\frac{۲}{p} \implies \mathbb{E}[X^۲] = \frac{۲-p}{p}.\end{aligned}$$

[1] Standard deviation

[2] Bernoulli random variable

حال می‌توانیم واریانس X را محاسبه می‌کنیم.

$$\mathrm{Var}(X) = \mathbb{E}[X^۲] - \mathbb{E}^۲[X] = \frac{۱-p}{p^۲}.$$

ضمناً می‌توان از نامساوی زیر هم استفاده کرد.

$$\mathrm{Var}(X) = \frac{۱-p}{p^۲} < \frac{۱}{p^۲}.$$

⊠

۲ نامساوی مارکف

فرض کنید X یک متغیر تصادفی گسسته و نامنفی باشد. آنگاه به ازای هر t مثبت،

$$Pr[X \geqslant t\mathbb{E}[X]] \leqslant \frac{۱}{t}.$$

یا به طور معادل برای هر k مثبت،

$$Pr[X \geqslant k] \leqslant \frac{\mathbb{E}[X]}{k}.$$

اثبات.

$$\mathbb{E}[X] = \sum_i iPr[X=i] \geqslant \sum_{i \geqslant k} iPr[X=i] \geqslant \sum_{i \geqslant k} kPr[X=i] = kPr[X \geqslant k].$$

□

مثال: استفاده از نامساوی مارکف در مسئله جمع‌آوری کوپن [۱]

فرض کنید تعداد مراحلی که طول می‌کشد تا همه n کوپن دیده‌شود را با متغیر تصادفی X نشان دهیم. فرض کنید می‌دانیم متوسط مقدار X برابر $n \ln n$ است. در این صورت با استفاده از نامساوی مارکف می‌توان کران زیر را برای احتمال اینکه تعداد مراحل بیش از t برابر متوسط تعداد مراحل شود، ارائه کرد.

$$Pr[X \geqslant tn \ln n] \leqslant \frac{۱}{t}.$$

به عنوان مثال داریم

$$Pr[X \geqslant ۲n \ln n] \leqslant \frac{۱}{۲}.$$

که البته از کرانی که قبلاً به دست آورده بودیم ضعیف‌تر است. ⊠

مثال: استفاده از نامساوی مارکف در مسئله توپ و سطل

می‌دانیم متوسط تعداد توپ‌هایی که در یک سطل خاص قرار می‌گیرند، ۱ است. طبق نامساوی مارکف، احتمال قرار گرفتن حداقل t توپ در یک سطل خاص حداکثر $\frac{۱}{t}$ است.

$$Pr[\#\text{of balls in ith bin} \geqslant t] \leqslant \frac{۱}{t}$$

[۱] برای مشاهده صورت دقیق مسائل به [1] مراجعه کنید.

اگر بخواهیم مشابه استدلال جلسه قبل از کران اجتماع [۲] استفاده کنیم، احتمال وجود سطلی با حداقل t توپ، حداکثر $\frac{n}{k}$ است. یعنی،

$$Pr[\text{Existence of a bin with at least } t \text{ balls}] \leqslant nPr[\#\text{of balls in ith bin} \geqslant t] \leqslant \frac{n}{t}.$$

اما دقت کنید که نامساوی فوق به ازای هر $1 \leqslant t \leqslant n$ یک کران بدیهی را نتیجه می‌دهد. در واقع به ازای هر t، سمت راست نامساوی بالا بزرگتر یا مساوی ۱ است که نتیجه‌ای بدیهی است.

پس در این مورد، استفاده از نامساوی مارکف و کران اجتماع در محاسبه کرانی برای احتمال وجود یک سطل با تعداد توپ زیاد، بی‌نتیجه است. ⊠

۳ نامساوی چبیشف

فرض کنید X یک متغیر تصادفی گسسته با متوسط μ_X و انحراف معیار σ_X باشد. در این صورت به ازای هر $k \in \mathbb{R}^+$ داریم

$$Pr[|X - \mu_X| \geqslant k] \leqslant \frac{\sigma_X^2}{k^2}.$$

یا معادلاً

$$Pr[|X - \mu_X| \geqslant t\sigma_X] \leqslant \frac{1}{t^2}.$$

اثبات. ابتدا توجه کنید که

$$Pr[|X - \mu_X| \geqslant t\sigma_X] = Pr[(X - \mu_X)^2 \geqslant t^2\sigma_X^2].$$

تعریف کنید $Y = (X - \mu_X)^2$. داریم

$$\mathbb{E}[Y] = \mathbb{E}[(X - \mu_X)^2] = \mathrm{Var}(X) = \sigma_X^2.$$

حال با اعمال نامساوی مارکف روی متغیر تصادفی Y حکم اثبات می‌شود.

$$Pr[|X - \mu_X| \geqslant t\sigma_X] = Pr[Y \geqslant t^2\mathbb{E}[Y]] \leqslant \frac{1}{t^2}.$$

□

مثال: استفاده از نامساوی چبیشف در مسئله توپ و سطل

می خواهیم با استفاده از نامساوی چبیشف کران بهتری برای احتمال وجود حداقل t توپ در یک سطل به خصوص پیدا کنیم تا پس از استفاده از کران اجتماع، بتوانیم به یک کران احتمالاتی غیربدیهی برسیم. به طور دقیق‌تر، می‌خواهیم کوچکترین t را پیدا کنیم که $Pr[\#\text{of balls in ith bin} \geqslant t] \leqslant \frac{1}{n}$. برای سادگی فرض کنید تعداد توپ‌ها در سطل iام X_i باشد. در این‌جا هر X_i توزیع دوجمله‌ای با پارامترهای n و $\frac{1}{n}$ دارد. پس $\mu_{X_i} = 1$ و $\sigma_{X_i} = \sqrt{n\frac{1}{n}(1 - \frac{1}{n})} \leqslant 1$. بنابراین اگر قرار دهیم $t = \sqrt{n} + 1$ با استفاده از نامساوی چبیشف می‌توان نوشت

$$Pr[X_i \geqslant \sqrt{n} + 1] = Pr[X - 1 \geqslant \sqrt{n}] \leqslant \frac{1}{n}.$$

که یک کران غیربدیهی است و اکنون با استفاده از نامساوی بول می‌توان کرانی برای احتمال وجود سطلی با تعداد توپ‌های زیاد ارائه کرد. البته این کران هنوز هم نسبت به کران لگاریتمی که جلسه پیش به دست آوردیم ضعیف‌تر است. ⊠

[۲] کران اجتماع(Union bound) یا نامساوی بول(Boole's inequality) را اینجا ببینید.

مثال: استفاده از نامساوی چبیشف در مسئله جمع‌آوری کوپن X_i را زمانی که طول می‌کشد تا پس از دیدن کوپن $i-1$ام، کوپن iام را ببینیم تعریف می‌کنیم. با استفاده از آنچه در جلسه قبل در مورد این مسئله گفته شد، می‌دانیم که هر X_i یک متغیر تصادفی هندسی است که احتمال موفقیت آن در هر گام $\frac{n-i+1}{n}$ است. پس

$$\mathbb{E}[X_i] = \frac{n}{n-i+1}$$

و

$$\mathrm{Var}(X_i) < (\frac{n}{n-i+1})^2.$$

حال اگر X را مدت زمانی که طول می‌کشد تا همه کوپن‌ها را دیده باشیم تعریف کنیم، به وضوح $X = \sum_{i=1}^{n} X_i$. چون X_iها مستقل هستند می‌توان گفت

$$\mathrm{Var}(X) < \sum_{i=1}^{n} (\frac{n}{n-i+1})^2 = n^2 \sum_{i=1}^{n} \frac{1}{i^2} < n^2 \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6} n^2$$

بنابراین

$$\mathrm{Var}(X) = \mathcal{O}(n^2) \implies \sigma_X = \mathcal{O}(n).$$

اکنون با استفاده از نامساوی چبیشف می توانیم کران زیر را به‌دست آوریم.

$$Pr[X \geqslant 2n \ln n] = Pr[X - n\ln n \geqslant n \ln n] \leqslant \frac{\mathcal{O}(n^2)}{(n\ln n)^2} = \mathcal{O}(\frac{1}{\ln^2 n}).$$

این کران نسبت به کران $\frac{1}{2}$ که با استفاده از نامساوی مارکف به‌دست آمد بسیار بهتر است. ⊠

۴ الگوریتم تصادفی برای محاسبه میانه

مسئله پیدا کردن میانه را می توان به شکل زیر تعریف کرد.

ورودی: n عنصر $a_1, a_2, \cdots, a_n$.

خروجی: عنصر $\frac{n}{2}$ از نظر بزرگی. یعنی $y_{\frac{n}{2}}$، اگر $y_1, y_2, \cdots, y_n$ جایگشتی از عناصر ورودی باشد که به ترتیب صعودی مرتب شده‌است.

در این مسئله منظور از زمان اجرای یک الگوریتم تعداد مقایسه های انجام شده بین عناصر توسط آن الگوریتم است.

برای پیدا کردن میانه n عنصر الگوریتم قطعی با زمان اجرای cn وجود دارد. همچنین ثابت شده‌است که هر الگوریتم قطعی برای این مسئله حداقل $(2+\epsilon)n$ مقایسه بین عناصر را انجام می‌دهد. به عبارتی در هر الگوریتم قطعی این مسئله، $c \geqslant 2+\epsilon$. در این بخش می‌خواهیم با استفاده از تصادف در طراحی الگوریتم ضریب c را کاهش دهیم. برای این کار از تکنیک نمونه‌گیری تصادفی استفاده می‌کنیم.

فرض کنید $y_1, y_2, \cdots, y_n$ مرتب شده عناصر ورودی هستند (که البته در حین اجرای الگوریتم به این لیست مرتب شده دسترسی نداریم و صرفاً برای تحلیل الگوریتم از آن استفاده می‌کنیم.)

ابتدا s عدد را به طور تصادفی از بین a_i ها و با جایگذاری انتخاب می‌کنیم. فرض کنید نمونه‌ها $X_1 \leqslant X_2 \leqslant \cdots \leqslant X_s$ باشند. به طور شهودی $X_{\frac{s}{2}}$ تقریبی از $y_{\frac{n}{2}}$ است. بر اساس لم زیر، با احتمال خوبی $X_{\frac{s}{2}}$ در چارک اول یا چهارم عناصر ورودی نیست و به تعبیری «نزدیک به میانه» است.

لم ۴. *احتمال اینکه* $X_{\frac{s}{2}}$ *در چارک اول یا چهارم باشد زیاد(!) نیست.*

اثبات. ابتدا با استفاده از نامساوی چبیشف کران بالایی برای احتمال اینکه $X_{\frac{s}{4}}$ در چارک اول n عنصر اولیه باشد به‌دست می آوریم. اگر تعداد نمونه هایی که در چارک اول هستند را با متغیر تصادفی X نشان دهیم، داریم

$$Pr[X_{\frac{s}{4}} < y_{\frac{n}{4}}] = Pr[\text{At least half of the samples are in the first quartile}] = Pr[X \geqslant \frac{s}{2}].$$

همچنین ملاحظه می‌شود که $X \sim Binomial(s, \frac{1}{4})$. پس

$$\mathbb{E}[X] = \frac{s}{4},\ \mathrm{Var}(X) = \frac{3s}{16}.$$

حال با استفاده از نامساوی چبیشف،

$$Pr[X \geqslant \frac{s}{2}] \leqslant \frac{\frac{3s}{16}}{(\frac{s}{4})^2} = \frac{3}{s} = \mathcal{O}(\frac{1}{s}).$$

به طور مشابه احتمال اینکه میانه نمونه در چارک چهارم از اعداد اولیه باشد هم $\mathcal{O}(\frac{1}{s})$ است. حال با استفاده از کران مجموع، می‌توان دید که میانه نمونه‌ها($X_{\frac{s}{2}}$) به احتمال $1 - \mathcal{O}(\frac{1}{s})$ در دو چارک میانی قرار دارد. □

پس از انجام نمونه‌گیری، می‌خواهیم دو عدد a و b از نمونه‌ها را به گونه‌ای انتخاب کنیم که اولاً میانه n عنصر بین این دو باشد و ثانیاً تعداد عناصر بین a و b در لیست اصلی زیاد نباشد تا بتوانیم با تعداد کمی مقایسه میانه را پیدا کنیم. به این منظور مقدار s را برابر $n^{\frac{3}{4}}$ قرار می‌دهیم و همچنین $a = X_{\frac{s}{2}-\sqrt{n}}$ و $b = X_{\frac{s}{2}+\sqrt{n}}$. حال لیست اولیه اعداد را اسکن می‌کنیم و با مقایسه هر عنصر با a و b، عناصر بین a و b را در لیست C ذخیره می‌کنیم و در همین حال تعداد عناصر کوچک‌تر از a را هم می‌شماریم و در متغیر t ذخیره می‌کنیم. در انتها عنصر $\frac{n}{2} - t$ ام لیست C را به عنوان جواب خروجی می‌دهیم. شبه‌کد این الگوریتم به شکل زیر است.

RandomizedMedian(A, n)

```
1  Pick n^(3/4) elements from A, chosen independently and uniformly at random with replacement;
   call this multiset of elements R.
2  Sort R
3  l = max(1, s/2 − √n)
4  h = min(n, s/2 + √n)
5  a = R[l]
6  b = R[h]
7  t = 0
8  C = {}
9  for i = 1 to n
10     if A[i] < a
11         t = t + 1
12     elseif A[i] < b
13         C.insert(A[i])
14 Sort C
15 if n/2 − t > 0
16     return R[n/2 − t]
17 else
18     return RandomizedMedian(A, n)
```

لم زیر بیانگر این موضوع است که الگوریتم پس از یک دور نمونه‌گیری احتمال موفقیت خوبی دارد و در نتیجه زمان اجرای متوسط الگوریتم چندجمله‌ای است.

لم ۵. *$y_{\frac{n}{2}}$ با احتمال خوبی بین a و b است.*

اثبات. کرانی برای $Pr[X_{\frac{s}{2}-\sqrt{n}} > y_{\frac{n}{2}}]$ ارائه می‌کنیم. اگر تعداد عناصری از نمونه که بزرگتر از $y_{\frac{n}{2}}$ هستند را با متغیر تصادفی X نمایش دهیم، به وضوح $X \sim Binomial(s, \frac{1}{2})$. همچنین

$$Pr[X_{\frac{s}{2}-\sqrt{n}} > y_{\frac{n}{2}}] = Pr[X_{\frac{s}{2}-\sqrt{n}}, \cdots, X_s \geqslant y_{\frac{n}{2}}] = Pr[X > \frac{s}{2}].$$

با استفاده از نامساوی چبیشف داریم

$$Pr[X - \frac{s}{2} \geqslant \sqrt{n}] \leqslant \frac{\frac{n^{\frac{3}{4}}}{4}}{(\sqrt{n})^2} = \frac{1}{4n^{\frac{1}{4}}} = \mathcal{O}(\frac{1}{\sqrt[4]{n}}).$$

بنابراین پس از یک دور نمونه‌گیری و انجام محاسبات، با احتمال $1 - \mathcal{O}(\frac{1}{\sqrt[4]{n}})$ موفق به یافتن میانه خواهیم شد. هر چند پس از یک دور نمونه‌گیری *با احتمال بالا* [1] موفق به یافتن نمونه نمی‌شویم، اما همین کران به دست آمده هم کران «خوبی» است زیرا اگر $n \to \infty$ احتمال موفقیت پس از یک دور نمونه‌گیری هم به ۱ میل می‌کند. □

علاوه بر لم فوق، باید نشان دهیم تعداد عناصر بین a و b نیز با احتمال خوبی کم هستند تا مرتب کردن اعداد موجود در لیست C هم در زمانی کمتر از $\Theta(n)$ صورت گیرد. اثبات لم زیر به عنوان تمرین به خواننده واگذار می‌شود.

لم ۶. *تعداد عناصر بین a و b یعنی همان تعداد اعضای لیست C در شبه‌کد بالا، به احتمال خوبی $2n^{\frac{3}{4}} = o(n)$ تا است.*

حال زمان اجرای الگوریتم را محاسبه می‌کنیم. به این منظور تعداد کل مقایسه های انجام شده در طول یک دور نمونه‌گیری و انجام محاسبات لازم را می‌شماریم.

(i) (*مرتب کردن R*) چون لیست R شامل $n^{\frac{3}{4}} = o(n)$ عنصر است، پس می توانیم با استفاده از یک الگوریتم مرتب‌سازی بهینه مانند مرتب‌سازی ادغامی، در زمان $o(n^{\frac{3}{4}} \lg n^{\frac{3}{4}}) = o(n)$ همه نمونه ها را مرتب کنیم.

(ii) (*مرتب کردن C*) طبق ۶ تعداد اعضای C به احتمال خوبی $o(n)$ تا است. پس مشابه لیست R، لیست C را هم می‌توان در زمان $o(n)$ مرتب کرد.

(iii) (*مقایسه‌ها در حلقه for*) الگوریتم بیشترین زمان را صرف مقایسه عناصر با a و b در خطهای ۱۰ و ۱۲ می‌کند. در واقع در این حلقه، هر یک از n عنصر ورودی با a و b مقایسه می‌شوند. پس مجموعاً تعداد $2n$ مقایسه در حلقه انجام می‌شود.

طبق نکات بالا، متوسط تعداد مقایسه های انجام شده توسط این الگوریتم تصادفی در یک در نمونه‌گیری $2n + o(n)$ تا است که از هر الگوریتم قطعی برای پیدا کردن میانه بهتر است.

البته با کمی دقت بیش‌تر می‌توان دید که در حلقه اصلی، چون اعداد کوچک‌تر از a دیگر با b مقایسه نمی‌شوند(خط ۱۲ شبه‌کد) و از آن جا که میانگین تعداد اعداد کوچک‌تر از a در لیست ورودی تقریباً $\frac{n}{2}$ است پس حدود نصف عناصر فقط یک بار در حلقه اصلی مقایسه می‌شوند. این یعنی تعداد کل مقایسه‌های انجام شده در یک دور نمونه‌گیری توسط الگوریتم در واقع $\frac{3}{4}n + o(n)$ است.

در نهایت چون احتمال موفقیت الگوریتم در هر دور نمونه‌گیری $1 - \mathcal{O}(\frac{1}{\sqrt[4]{n}})$ و تعداد مقایسه‌ها در هر دور هم $\frac{3}{4}n + o(n)$ محاسبه شد، پس متوسط تعداد مقایسه‌هایی که الگوریتم تصادفی انجام می‌دهد، $(\frac{3}{4}n + o(n))(1 + \mathcal{O}(\frac{1}{\sqrt[4]{n}}))$ است.

[1] With high probability

۵ پیچیدگی محاسباتی الگوریتم‌های تصادفی

در جلسات گذشته با دو نوع الگوریتم تصادفی آشنا شدیم. الگوریتم‌های مونت‌کارلو زمان اجرای چندجمله‌ای دارند اما ممکن است جواب صحیح را بدست نیاورند. در مقابل، الگوریتم های لاس‌وگاس همواره جواب درست را محاسبه می‌کنند اما زمان اجرای بدترین حالت آن‌ها ممکن است حتی محدود هم نباشد(یعنی الگوریتم هرگز متوقف نشود). در این قسمت با دو کلاس پیچیدگی محاسباتی جدید آشنا می‌شویم که متناظر با این دو نوع الگوریتم تصادفی هستند.

کلاس RP [1] کلاس مسائلی مانند $\mathcal{L}$ که برایشان الگوریتم تصادفی(در بدترین حالت) چندجمله‌ای مانند $\mathcal{A}$ وجود دارد به طوری که به ازای هر ورودی x،

$$x \in \mathcal{L} \implies Pr[\mathcal{A}(x) = ۱] \geqslant \frac{۱}{۲},$$
$$x \notin \mathcal{L} \implies Pr[\mathcal{A}(x) = ۰] = ۱.$$

به عبارتی اگر الگوریتم $\mathcal{A}$ وجود داشته باشد که به ازای هر ورودی x از مسئله $\mathcal{L}$ که برای آن جواب مسئله «بله» یا معادلاً «۱» باشد، الگوریتم $\mathcal{A}$ به احتمال حداقل $\frac{۱}{۲}$ جواب درست «بله» را تولید کند و به ازای هر ورودی x که جواب مسئله «خیر» یامعادلاً «۰» باشد، الگوریتم $\mathcal{A}$ با احتمال ۱ جواب درست «خیر» را تولید کند، آنگاه می‌گوییم *مسئله* $\mathcal{L}$ *از کلاس* RP *است*.

بایستی به این نکته توجه شود که به‌جای احتمال $\frac{۱}{۲}$ گفته شده در تعریف می‌توان هر عدد دیگری را هم قرار داد و به تعریف معادلی دست یافت. در واقع چنان‌چه الگوریتم $\mathcal{A}$ در تعریف بالا صدق کند، با اجرای این الگوریتم به تعداد چندجمله‌ای بار می‌توان به الگوریتم چند جمله‌ای دیگری دست یافت که احتمال درستی خروجیش، اگر جواب درست «بله» باشد، حداقل $۱ - \frac{۱}{۲^{p(n)}}$ است.

این کلاس و مکمل آن که به شکل زیر تعریف می‌شود، متناظر با الگوریتم‌های مونت‌کارلو هستند.

کلاس coRP کلاس مسائلی مانند $\mathcal{L}$ که برایشان الگوریتم تصادفی(در بدترین حالت) چندجمله‌ای مانند $\mathcal{A}$ وجود دارد به طوری که به ازای هر ورودی x،

$$x \in \mathcal{L} \implies Pr[\mathcal{A}(x) = ۱] = ۱,$$
$$x \notin \mathcal{L} \implies Pr[\mathcal{A}(x) = ۰] \geqslant \frac{۱}{۲}.$$

مثال: حالت تصمیم‌گیری مسئله برش کمینه(در ورودی عدد m هم داده می‌شود و جواب مسئله ۱ است اگر اندازه برش کمینه همان m باشد و در غیر این صورت ۰ است.) که در جلسه پیش در مورد آن صحبت شد، یک مسئله از کلاس RP است. در جلسه پیش الگوریتمی ارائه کردیم که با احتمال بالا در زمان چندجمله‌ای اندازه برش کمینه را بدست آورد. با تغییراتی جزیی می‌توان همین الگوریتم را طوری نوشت که اگر m اندازه برش کمینه باشد خروجی الگوریتم با احتمال بالا «۱» و در غیر این صورت خروجی الگوریتم قطعاً «۰» باشد. پس مسئله برش کمینه در کلاس RP قرار دارد. همچنین بعضی الگوریتم‌های تشخیص اعداد اول از کلاس coRP هستند. یعنی اگر ورودی عددی مرکب باشد، الگوریتم ممکن است به اشتباه عدد را اول تشخیص دهد. ⊠

به راحتی می توان نشان داد $\mathrm{RP} \subset \mathrm{NP}$. در واقع اگر رشته بیت‌های تصادفی استفاده شده توسط یک الگوریتم را مشابه ورودی در نظر بگیریم، یک الگوریتم از کلاس RP است اگر در صورت «بله» بودن جواب، به ازای **حداقل نیمی** از رشته بیت‌های تصادفی خروجی الگوریتم هم «بله» باشد. همچنین این مسئله از کلاس NP است اگر به ازای **حداقل یک** رشته از بیت‌های تصادفی خروجی الگوریتم «بله» باشد. با استفاده از این شیوه تعریف، گزاره‌ی $\mathrm{RP} \subset \mathrm{NP}$ بدیهتاً درست است.

کلاس ZPP [2] کلاس مسائلی مانند $\mathcal{L}$ که برایشان الگوریتم تصادفی با زمان اجرای متوسط چندجمله‌ای وجود دارد که همیشه درست جواب بدهد.

[1] Randomized Polynomial time
[2] Zero-error Probabilistic Polynomial time

توجه کنید که زمان اجرای چنین الگوریتم‌هایی در بدترین حالت ممکن است حتی متناهی هم نباشد اما به طور متوسط زمان اجرا چندجمله‌ای است.

این کلاس از مسائل دقیقاً متناظر با مسائلی هستند که برایشان الگوریتم لاس‌وگاس با زمان اجرای متوسط چندجمله‌ای وجود دارد.

قضیه ۱. $\mathrm{ZPP} = \mathrm{RP} \cap \mathrm{coRP}$.

اثبات. فرض کنید مسئله $\mathcal{L}$ از کلاس ZPP باشد و برای آن الگوریتم لاس‌وگاس $\mathcal{A}$ با زمان اجرای متوسط $T(n)$ وجود داشته باشد. الگوریتم $\mathcal{B}$ زیر را برای حل همین مسئله در نظر بگیرید.

$\mathcal{B}(x)$
1 Run $\mathcal{A}(x)$ for time $2T(n)$
2 **if** $\mathcal{A}(x)$ found an answer
3 **return** that answer
4 **else return** 0.

اگر جواب درست مسئله به ازای ورودی x برابر ۰ باشد، خروجی $\mathcal{B}$ هم قطعاً ۰ است. همچنین اگر جواب درست مسئله ۱ باشد، به احتمال حداقل $\frac{1}{2}$ خروجی الگوریتم $\mathcal{B}$ هم ۱ است زیرا با استفاده از نامساوی مارکف، می‌دانیم زمان اجرای $\mathcal{A}$ به احتمال حداقل $\frac{1}{2}$ کمتر از $2T(n)$ است. پس

$$Pr[\mathcal{B} \text{ outputs wrong answer}] \leqslant Pr[\mathcal{A} \text{ needs more than } 2T(n)\text{time }] \leqslant \frac{1}{2}.$$

بنابراین الگوریتم $\mathcal{B}$ در شرایط گفته شده در تعریف کلاس RP صدق می‌کند و در نتیجه مسئله $\mathcal{L}$ از کلاس RP است. به طریق مشابه می‌توان نشان داد اگر مسئله $\mathcal{L}$ از کلاس ZPP باشد آنگاه از کلاس coRP هم هست. پس ثابت کردیم

$$\mathrm{ZPP} \subseteq \mathrm{RP} \cap \mathrm{coRP}.$$

حال فرض کنید الگوریتم های RP و coRP $\mathcal{A}$ و $\mathcal{A}'$ برای مسئله $\mathcal{L}$ داده شده‌اند. الگوریتم $\mathcal{B}$ تا هنگامی که یکی از $\mathcal{A}$ یا $\mathcal{A}'$ جواب غیر پیش‌فرض بدهد، هر دو الگوریتم را به طور پی‌درپی اجرا می‌کند. منظور از خروجی پیش‌فرض، خروجی ۱ برای $\mathcal{A}$ و خروجی ۰ برای $\mathcal{A}'$ است.

توجه کنید که اگر هر یک از دو الگوریتم $\mathcal{A}$ و $\mathcal{A}'$ خروجی غیر پیش‌فرض خود را تولید کنند قطعاً جوابشان درست است. ضمناً در هر دور، پس از اجرای هر دو الگوریتم به احتمال حداقل $\frac{1}{2}$ یکی از دو الگوریتم خروجی غیر پیش‌فرض تولید می‌کند و موفق می‌شویم. بنابراین تعداد دور هایی که دو الگوریتم را اجرا می‌کنیم دارای توزیع هندسی با احتمال موفقیت $\frac{1}{2}$ در هر دور است. پس به طور متوسط اجرای الگوریتم دو دور طول می‌کشد و جواب نهایی هم هیچ خطایی ندارد. این یعنی $\mathcal{B}$ یک الگوریتم لاس‌وگاس با متوسط زمان اجرای چندجمله‌ای است. پس مسئله $\mathcal{L}$ عضو کلاس ZPP نیز هست و در واقع ثابت کردیم

$$\mathrm{RP} \cap \mathrm{coRP} \subseteq \mathrm{ZPP}.$$

و حکم ثابت می‌شود. □

تمرین: ثابت کنید تعریف زیر معادل تعریف گفته شده برای کلاس ZPP است.

« برای مسئله $\mathcal{L}$ الگوریتمی با زمان اجرای چندجمله‌ای در بدترین حالت وجود دارد که خروجی آن با احتمال $\frac{1}{2}$ جواب درست مسئله است و به احتمال $\frac{1}{2}$ «نمی‌دانم» است. »

مراجع

[۱] جزوه جلسه اضافه سوم(الگوریتم‌های تصادفی)، درس آنالیز الگوریتم

[۲] ویدیو جلسه اضافه چهارم(نامساوی های مارکف و چبیشف)، درس آنالیز الگوریتم

[3] Motwani, Rajeev, and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 2018.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه اضافه ۵: کران چرنوف[1] نگارنده: کیانا عسگری

همان‌طور که دیدیم، خیلی اوقات در تحلیل الگوریتم‌های تصادفی نیاز به محاسبه کرانی برای احتمال دور شدن یک متغیر تصادفی از میانگین خودش داریم و نیاز به تضمینی داریم که متغیر خیلی از میانگین خودش دور نشود. کران‌های مارکوف و چبیشوف و تعدادی کاربرد از آن‌ها را در جلسات قبل دیدیم. دیدیم که کران چبیشف بطور خاص با توزیع های مستقل دوجانبه[2] عمل می‌کند.

کران چرنوف، از دو کران قبلی قوی‌تر است و برای متغیرهای تصادفی مستقل استفاده می‌شود. می‌توان گفت نامساوی چرنوف پرکاربردترین کران در تحلیل الگوریتم‌های تصادفی برای تعیین کران بالای احتمال دور شدن متغیر تصادفی از میانگین است. صورت‌های کلی‌تری از کران چرنوف وجود دارند که به کران هافدینگ[3] معروف هستند و در جلسات حل تمرین بررسی می‌شوند.

۱ کران چرنوف

قضیه ۱. *فرض کنید X_i متغیرهای تصادفی برنولی (صفر و یکی) و مستقل باشند و همچین:*

$$X = \sum_i X_i, \quad E[X] = \mu, \quad \Pr[X_i = 1] = p_i$$

آنگاه نامساوی زیر برقرار است:

$$\Pr[X > (1+\epsilon)\mu] \leq \left[\frac{e^{\epsilon}}{(1+\epsilon)^{(1+\epsilon)}}\right]^{\mu}$$

عبارت سمت راست (نامساوی داخل کروشه)، به ϵ و توان آن به μ وابسته است. چون عبارت داخل کروشه کم‌تر از یک است، هرچقدر میانگین متغیر بزرگ‌تر شود، کران سمت راست کوچک‌تر می‌شود. همچنین هرچقدر ϵ بزرگ‌تر شود کران سمت راست کوچک‌تر می‌شود.

اثبات. تابع f را بصورت $f(X) = e^{tX}$[4] تعریف کنید. با استفاده از کران مارکوف داریم:

$$\Pr[f(X) > y] \leq \frac{E[f(X)]}{y}$$

حال چون Exp یک تابع صعودی است:

$$\Pr[X > (1+\epsilon)\mu] = \Pr[e^{tX} \geq e^{t(1+\epsilon)\mu}] \leq \frac{E[e^{tx}]}{e^{t(1+\epsilon)\mu}} \qquad (۱)$$

[1] Chernoff bound
[2] pairwise independent random variables
[3] Hoeffding
[4] Moment generating function

چون X_i ها متغیرهای تصادفی مستقل هستند داریم:

$$E[e^{tX}] = E[e^{t\sum_i X_i}] = E[\Pi_i e^{tX_i}] = \Pi_i E[e^{tX_i}] \quad (۲)$$

$$E[e^{tX_i}] = p_i e^t + (1-p_i)e^0 = 1 + p_i(e^t - 1) \overset{1+x\le e^x}{\le} e^{p_i(e^t-1)} \quad (۳)$$

$$\overset{(2),(3)}{\Longrightarrow} E[e^{tX}] \le \Pi_i e^{p_i(e^t-1)} = e^{(e^t-1)\sum_i p_i} = e^{\mu(e^t-1)} \quad (۴)$$

$$\overset{(1),(4)}{\Longrightarrow} \Pr[X > (1+\epsilon)\mu] \le \frac{e^{\mu(e^t-1)}}{e^{t(1+\epsilon)\mu}} \overset{t=\ln(1+\epsilon)}{\Longrightarrow} \Pr[X > (1+\epsilon)\mu] \le \left[\frac{e^\epsilon}{(1+\epsilon)^{(1+\epsilon)}}\right]^\mu$$

□

۱.۱ شکل‌های ساده‌تر نامساوی چرنوف

کران بدست آمده در حالت کلی، پیچیده است که استفاده از آن را دشوار می‌سازد. برای ساده‌تر شدن کار با کران چرنوف، بر اساس مقدار ϵ، می‌توان سه حالت زیر را که حالت‌های ساده شده‌ای از حالت اصلی هستند بدست آورد:

$$\Pr[X > (1+\epsilon)\mu] \le \left[\frac{e^\epsilon}{(1+\epsilon)^{(1+\epsilon)}}\right]^\mu \le \begin{cases} e^{-\mu\frac{\epsilon^2}{2+\epsilon}} & \forall\epsilon \\ e^{-\mu\frac{\epsilon^2}{3}} & \epsilon \le 1 \\ 2^{-(1+\epsilon)\mu} & \epsilon \ge 6 \end{cases}$$

همچنین نامساوی مشابهی برای حالت زیر داریم:

$$\Pr[X \le (1-\epsilon)\mu] \le e^{-\mu\frac{\epsilon^2}{2}}$$

اگر دقت کنید، می‌بینید که در تمام حالات بالا، کرانی که داریم بر حسب میانگین متغیر، نمایی است. پس خیلی اوقات اگر بتوانیم نشان دهیم که میانگین یک متغیر تصادفی تقریباً از مرتبه $\lg n$ است، آنگاه می‌توانیم نتیجه بگیریم که با احتمال بالایی از میانگین متغیر خیلی دور نخواهیم شد.

نکته : حالت ساده‌شده سوم این خاصیت را دارد که نیازی نداریم مقدار دقیق ϵ و μ را بدانیم بلکه دانستن $(1+\epsilon)\mu$به تنهایی برای انجام محاسبات کافی است.

نکته : اگر داشته باشیم $\epsilon \cong \frac{1}{\sqrt{\mu}}$ ، آنگاه نامساوی، به ما یک عدد ثابت می‌دهد که کران کارآمدی نیست. به‌عنوان مثال در متغیر تصادفی‌دوجمله‌ای این عبارت در واقع می‌گوید که احتمال دور شدن به اندازه یک انحراف معیار از میانگین کم نیست که دور از ذهن هم نبود.

نکته : در کران چرنوف وقتی ϵ بزرگ شود، نتیجه بدست آمده شروع به کارآمد شدن می‌کند. نکته‌ای که در کران چرنوف دیده می‌شود این است که دنباله هر توزیع، به توزیع گاوسی شباهت دارد که به‌گونه‌ای شبیه به قضیه حد مرکزی است.

نکته : برای حالاتی که X_i ها متغیرهای گسسته نباشند هم می‌توان کران‌های مشابهی داد که به کران‌های هافدینگ معروف‌اند.

مساله توپ و سطل[۵]

می‌خواهیم تعداد توپ‌های داخل یک سطل را با استفاده از کران چرنوف و نامساوی بول[۶] بررسی می‌کنیم. متغیرهای تصادفی زیر را تعریف کنید:

[۵] Balls In Bins
[۶] union bound

$X :=$ تعداد توپ‌های داخل سطل اول

$X_i :=$ آیا توپ i ام داخل سطل یک است یا خیر

$$X = \sum_i X_i, \quad E[X] = 1, \quad (1+\epsilon)\mu = 2\lg n$$

با توجه به مقدار ϵ، می‌توانیم از حالت ساده‌شده‌ی سوم کران چرنوف استفاده کنیم (دیدیم که در این حالت نیازی به تعیین ϵ نیست و دانستن مقدار $(1+\epsilon)\mu$ کافی است).

$$\Pr[X > 2\lg n] \leq 2^{-2\lg n} = n^{-2}$$

در نهایت می‌توانیم با استفاده از نامساوی بول نشان دهیم که این پیشامد برای تمام سطل‌ها با احتمال بالا برقرار است.

با استفاده از کران چرنوف بدست آوردیم که با احتمال بالا بیشترین تعداد توپی که داخل یک سطل قرار می‌گیرد، لگاریتمی است که بطور نمایی از کرانی که نامساوی چبیشف به ما می‌داد ($\sqrt{(n)}$) بهتر است. درواقع کرانی که با استفاده از نامساوی چرنوف بدست آوردیم به کران واقعی ($O(\frac{\lg n}{\lg\lg n})$) خیلی نزدیک است که نشان‌دهنده قدرت کران چرنوف است. بنابراین در خیلی از مسأله‌ها نیازی نیست که از ابزاری قوی‌تر از کران چرنوف استفاده کنیم.

مسیریابی در شبکه[۷]

فرض کنید یک شبکه (گراف) به همراه یک تعداد زوج رأس مبدأ و مقصد در اختیار داریم که برای هر زوج، رأس مبدأ می‌خواهد پیغامی را به رأس مقصد می‌فرستد. می‌خواهیم تعدادی مسیر بین زوج‌های مبدأ و مقصد بیابیم به‌طوری‌که بیشینه زمان رسیدن پیام‌ها به مقصد کمینه شود.

فرض می‌کنیم گراف بدون وزن است، یال‌ها جهت‌دار دوطرفه هستند و هم‌چنین در هر واحد زمانی از طریق هر یالی حداکثر یک پیغام می‌تواند فرستاده شود. بنابراین شهوداً هدف کاهش بار هر یال است (منظور از کم کردن بار یک یال این است که تعداد زیادی پیغام هم‌زمان درخواست عبور از آن یال را ندهند).

در حقیقت به شبکه داده شده به چشم یک سیستم توزیع‌شده[۸] نگاه می‌کنیم؛ یعنی بطور خاص الگوریتمی که می‌دهیم برای هر زوج مبدأ و مقصد، مستقل از بقیه زوج‌ها مسیری را تعیین می‌کند، که مناسب پیاده‌سازی در یک محیط توزیع شده است.

معمولاً در این سیستم‌ها توپولوژی و نحوه ساخت شبکه در دست خودمان است. در حالت ایده‌آل که شبکه ما گراف کامل باشد، فرستادن پیام‌ها برای هر زوج مبدأ و مقصد در یک گام ممکن است؛ اما هزینه و پیچیدگی ساخت این حالت زیاد است و اکثر اوقات شبکه‌های ما تنک[۹] هستند؛ یعنی تعداد یال‌ها نسبت به حداکثر تعداد یال ممکن کم است و در این حالت پیدا کردن مسیرهای بهینه چالش‌برانگیز است.

الگوریتم مسیریابی جایگشتی[۱۰]

فرض می‌کنیم هر رأس، مبدأ و مقصد دقیقاً یک پیام است (دقیقا N زوج مبدأ و مقصد داریم و هر راس دقیقاً یک‌بار بعنوان مبدأ و یک‌بار به‌عنوان مقصد ظاهر شده است). برای این نوع از ورودی، الگوریتمی که ارائه می‌دهیم، الگوریتم بی‌توجه[۱۱] است. (یعنی عمل‌کرد الگوریتم

[7] Routing
[8] distributed
[9] Sparse
[10] Permutation Routing
[11] Oblivious

برای هر زوج، از زوج‌های دیگر مستقل است.)

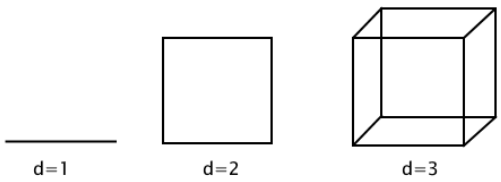
- **الگوریتم قطعی**

ثابت می‌شود که هر الگوریتم قطعیِ بی‌توجه برای مسیریابی جایگشتی نیاز به $\Omega(\sqrt{\frac{N}{d}})$ گام روی یک شبکه N راسی با قطر d دارد. (قطر گراف برابر اندازه‌ی بزرگ‌ترین کوتاه‌ترین مسیر در گراف است؛ یعنی طول کوتاهترین مسیر بین دو رأسی که فاصله آن‌ها بیشینه باشد.)

- **الگوریتم تصادفی**

اکنون می‌خواهیم الگوریتمی را بررسی کنیم که نیاز به تعداد گام‌های لگاریتمی بر حسب تعداد رأس‌های شبکه داشته باشد و همچنین از کران پایینی که برای الگوریتم‌های قطعی وجود دارد، بطور نمایی بهتر باشد.

تعریف : منظور از شبکه ابرمکعب[۱۲]، گراف‌هایی به شکل (۱) هستند که بطور بازگشتی از روی هم ساخته می‌شوند. برای ساختن یک ابرمکعب $d+1$ بعدی کافیست دو نسخه از ابر مکعب d بعدی را کنار هم قرار داده و رأس‌های متناظر آن‌ها را به هم وصل کنیم. نوع دیگر بیان این دسته از گراف‌ها بر اساس رأس‌های آن‌هاست. یک ابرمکعب d بعدی، دارای 2^d راس است که هر رأس آن رشته‌ای از بیت‌ها به طول d است و دو رأس u و v به هم متصل‌اند اگر و تنها اگر رشته‌های متناظر آن‌ها تنها در یک بیت تفاوت داشته باشند. پس درجه هر رأس برابر d وقطر شبکه هم برابرd است.



شکل ۱: نمونه هایی از ابر مکعب

فرض می‌کنیم شبکه به‌صورت یک ابر مکعب باشد. اکنون این فرض را که مقصدها بطور تصادفی انتخاب می‌شوند را نیز اضافه می‌کنیم. در اینصورت همچنان هر رأس، حتماً بعنوان مبدأ ظاهر می‌شود ولی این قضیه برای مقصد برقرار نخواهد بود. در حقیقت این‌که هر رأس به‌عنوان مقصد چند مسیر ظاهر می‌شود تبدیل به همان مسأله توپ و سطل می‌شود.

نکته : همان‌طور که در جلسه الگوریتم‌های تصادفی دیدیم، ما معمولاً فرض نمی‌کنیم که ورودی ما تصادفی است؛ بلکه الگوریتمی ارائه می‌دهیم که خود الگوریتم تصادفی است و برای هر ورودی دشمنانه‌ای به احتمال خوبی کار کند. اما دقت کنید به‌عنوان مثال اینجا فرض می‌کنیم خود ورودی تصادفی است.

الگوریتم تصحیح بیتی[۱۳]

در این الگوریتم زوج مبدأ و مقصد s و t را در نظر گرفته و به بیت‌هایی که رشته‌های متناظر این دو راس با هم تفاوت دارند دقت می‌کنیم. مسیر انتخابی الگوریتم به این صورت است که با شروع از مبدأ به ترتیب از بیت‌های پرارزش به کم‌ارزش، بیت‌های متفاوت رشته s با رشته t را تصحیح کند:

$$s = 10110, t = 11010 \Rightarrow p : 10110 \longrightarrow 11110 \longrightarrow 11010 = t$$

[12] Hypercube
[13] BIT FIXING algorithm

پس طول هر مسیر انتخابی $O(d)$ است (طول هر مسیر به اندازه تعداد بیت‌های متفاوت رأس مبدأ و مقصد است که حداکثر d است) و متوسط طول مسیرها برابر $\frac{d}{2}$ است؛ زیرا هر بیت رأس مقصد، به احتمال $\frac{1}{2}$ با بیت متناظر راس مبدأ متفاوت است.

حال تعریف کنید:

$T(e)$: تعداد مسیرهایی که از یال e عبور می‌کنند.

می‌خواهیم میانگین متغیر تصادفی $T(e)$ را برای هریال e محاسبه کنیم. چون گراف ما کاملاً متقارن است مقدار میانگین بدست آمده باید برای تمام یال‌ها یکسان می‌شود. این نکته به محاسبه زیر می‌انجامد: (فرض می‌کنیم یال‌های گراف جهت‌دار و دوطرفه است)

$$E[T(e)] = \frac{E[\text{جمع طول مسیرها}]}{\text{تعداد یال‌ها}} = \frac{N\frac{d}{2}}{Nd} = \frac{1}{2}$$

از طرفی $T(e)$ جمع تعدادی متغیر تصادفی نشانگر مستقل $y_i(e)$ است که $y_i(e)$ نشانگر عبور یا عدم عبور مسیر با مبدأ رأس i ام از یال e است. این متغیرها بخاطر بی‌توجه بودن الگوریتم مستقل هستند.

$$\Rightarrow T(e) = \sum_i y_i(e)$$

حال اگر قرار دهیم $(1+\epsilon)\frac{1}{2} = 2\lg N$ آنگاه طبق حالت ساده‌شده‌ی سوم کران چرنوف داریم:

$$\Pr[T(e) \geq 2\lg N] \leq 2^{-2\lg N} = N^{-2}$$

$$\overset{\text{union bound}}{\Longrightarrow} \Pr[\text{بار یک یال بیشتر از } 2\lg N \text{ باشد}] \leq \frac{N\lg N}{N^2} = \frac{\lg N}{N}$$

حال برای محاسبه زمان مسیریابی یک پیغام، کافیست دقت کنید که مسیر آن حداکثر طول d دارد. چون بار هر یال هم با احتمال بالا $O(\lg N)$ است؛ پس کل زمانی که طول می‌کشد یک پیغام به مقصد برسد با احتمال بالا برابر $O(\lg^2 N)$ است که از الگوریتم قطعی خیلی بهتر است.

یک تحلیل بهتر

- **تحلیل غلط** : واضح است که زمان فرستاده شدن یک پیغام برابر مجموع طول مسیر طی شده و تأخیر تجربه شده است. طول مسیر بر حسب N لگاریتمی است. پس کافیست ثابت کنیم تأخیر پیغام هم لگاریتمی است.

$$E[\text{تأخیر پیغام } i \text{ ام}] \leq \sum_{e \in \text{مسیر } i \text{ ام}} E[T(e)] = \frac{d}{2}$$

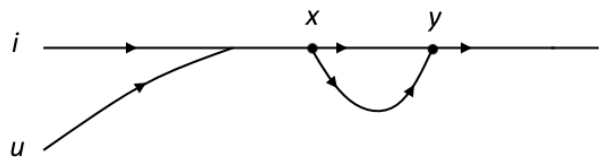
پس چون میانگین یک مقدار لگاریتمی است طبق نکته‌ای که در کران چرنوف بیان شد، با احتمال بالا از مقدار میانگین دور نمی‌شویم:

$$\Pr[X \geq 10\frac{d}{2}] \leq 2^{-5d}$$

اما تحلیل بالا اشتباه است؛ زیرا متغیرهای تصادفی $T(e)$ مستقل نیستند. بعنوان مثال اگر بار یال e زیاد باشد با احتمال بالایی بار یال‌هایی که به انتهای e وصل هستند هم زیاد خواهد بود و نمی‌توانیم از نامساوی چرنوف استفاده کنیم.

- **تحلیل درست** : این‌بار دسته دیگری از متغیرها را پیدا می‌کنیم که آن‌ها مستقل هستند و می‌توانیم کران چرنوف را روی آن‌ها اعمال کنیم.

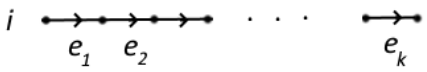
لم ۱. *فرض کنید s پیغام مسیرشان با مسیر پیغام iام یال مشترک داشته باشد. در این‌صورت تأخیر پیغام iام حداکثر s است.*



شکل ۲: این حالت هیچ‌وقت رخ نمی‌دهد.

اثبات. فرض کنید مسیر پیام iام بصورت شکل (۲) باشد و مسیر پیغام u آن‌را قطع کند. وقتی این دو مسیر از یک‌دیگر جدا شدند، دیگر با هم اشتراکی نخواهند داشت؛ زیرا در غیر اینصورت شکل (۲) اتفاق میفتد اما چون الگوریتم تصحیح بیتی یک استراتژی قطعی است و در مسیر بین x به y بیت‌ها را از ترتیب پرارزش به کم‌ارزش تصحیح می‌کند، پس امکان ندارد که دو مسیر متمایز از x به y داشته باشیم.

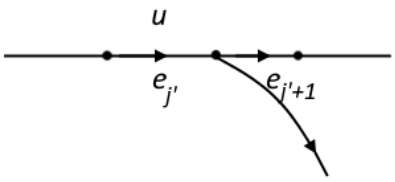
حال فرض کنید مسیر پیام i ام بصورت شکل (۳) باشد. فرض کنید در زمان t، پیام u یال e_j از مسیر p را پیمایش کند. در اینصورت عقب‌افتادگی[۱۴] پیام u در زمان t را بصورت $t-j$ تعریف می‌کنیم. حال لحظه‌ای را در نظر بگیرید که تأخیر پیام iام از l به $l+1$ افزایش یابد. در این‌صورت $t=j+l$. در این لحظه پیغام i روی یال e_j به تأخیر می‌خورد و پس یعنی پیغام دیگری از این یال در آن لحظه استفاده می‌کند. ایده استدلال این است که هربار تأخیر پیام i ام یک واحد افزایش یافت این یک واحد را «از پیامی که منجبر به تأخیر پیام iام شده



شکل ۳: مسیر p

بگیریم».

فرض کنید t' آخرین زمانی باشد که یک پیام با عقب‌افتادگی l مسیر p را پیمایش می‌کند. این پیغام را u بنامید. ادعا می‌کنیم که مسیر پیام u بعد از پیمایش یال $e_{j'}$، مسیر p را ترک می‌کند (و هیچ‌گاه برنمی‌گردد). دلیل درست بودن این ادعا این است که اگر مسیر u بعد از یال $e_{j'}$، یال $e_{j'+1}$ را پیمایش کند آنگاه بعد از زمان t' هم‌چنان پیامی وجود دارد که با عقب‌افتادگی l یکی از یال‌ها را پیمایش کرده که تناقض است.



شکل ۴: اشتراک دو مسیر یال e_j است

حال کافیست افزایش کل تأخیر پیام iام از l به $l+1$ را به پیام u نسبت دهیم. این پیام‌ها متمایز هستند؛ یعنی اگر u رأسی باشد که افزایش تأخیر پیام i از l به $l+1$ را به آن نسبت می‌دهیم و u' رأسی باشد که افزایش تأخیر پیام i از l' به $l'+1$ را به آن نسبت می‌دهیم، چون l' با l متفاوت است و u و u' آخرین یالی که در مسیر p طی می‌کنند به ترتیب با عقب‌افتادگی l و l' است پس u' با u متمایز است پس حکم اثبات می‌شود.

[14] lag

□

اکنون تعریف کنید:

H_{ij} :$=$ متغیر بولی اشتراک مسیر پیغام i ام و پیغام j ام.

هر j که مسیرش با مسیر پیغام iام اشتراک دارد حداقل یک واحد به $T(e)$ هایی که در مسیر iام است اضافه می‌کند. بنابراین:

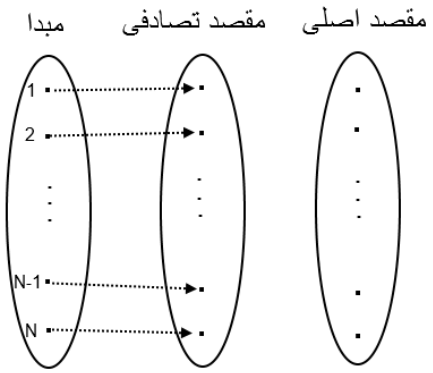
$$E[\text{تأخیر پیام } i\text{ام}] \leq E[\sum H_{ij}] \leq E[\sum_{e \in \text{مسیر } i\text{ام}} T(e)] = \frac{d}{2} = \frac{\log N}{2} \overset{lemma(1)}{\Rightarrow}$$

اکنون H_{ij} ها مستقل هستند، زیرا وقتی مسیر پیغام iام انتخاب شد اشتراک داشتن این مسیر با مسیر kام و اشتراک داشتنش با مسیر پیغام jام از هم مستقل‌اند. پس در نتیجه می‌توانیم روی متغیرهای $\sum_i H_{ij}$ از چرنوف استفاده کنیم و تحلیل غلط قبلی را عیناً برای H_{ij} ها تکرار کنیم و به نتیجه دل‌خواه برسیم.

تاکنون دیدیم که با استفاده از الگوریتم تصحیح بیتی با احتمال بالا زمانی که طول می‌کشد یک پیغام از مبدأ به مقصد برسد لگاریتمی است. حال در قدم پایانی فرض اضافی تصادفی بودن مقصد پیام‌ها را برمی‌داریم و فرض می‌کنیم مقصدها جایگشتی از رأس‌ها هستند (مسأله اصلی). در این حالت هم هم‌چنان می‌توانیم از الگوریتم تصحیح بیتی استفاده کنیم؛ اما نکته این است که دیگر هیچ عامل تصادفی‌سازی وجود ندارد که می‌تواند باعث شود هر استراتژی در بدترین حالت بد عمل کند. برای رفع این مشکل یک لایه میانی در نظر می‌گیریم؛ یعنی برای هر رأسی یک مقصد تصادفی تولید می‌کنیم و پیغام‌ها را از مبدأ به مقصد تصادفی و سپس از مقصد تصادفی به مقصد واقعی می‌فرستیم؛ یا به معنایی:

۱. پیام‌ها را به مقصدهای تصادفی می‌فرستیم

۲. پیام‌ها را از مقصد تصادفی خود به مقصد واقعی می‌فرستیم



شکل ۵: نحوه عملکرد لایه میانی

هر کدام از دو مرحله در حقیقت یک‌بار اجرای الگوریتم در حالت مقصد تصادفی است که دیدیم با احتمال بالا زمان لگاریتمی طول خواهد کشید (برای قدم دوم الگوریتم کافیست به صورت برعکس به آن نگاه کنیم). پس در الگوریتم جدید، هر رسیدن پیغام مدت زمان $O(\log N)$ به طول می‌انجامد.

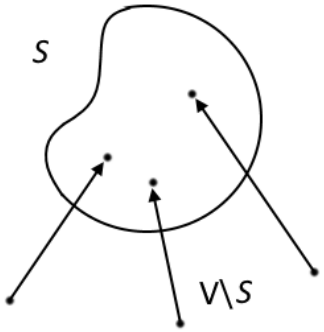
نکته : ایده استفاده‌شده و تحلیل بیان‌شده برای شبکه‌های متقارن دیگر هم کار می‌کند.

مسأله ۳ـ رنگ‌آمیزی گراف های ۳ـ رنگ‌پذیر چگال

تعریف : گراف G را δـ چگال می‌گوییم اگر درجه هر رأس آن حداقل δn باشد.

می‌دانیم مسأله ۳ـ رنگ‌پذیری **انپی‌ـ سخت**[15] است و در حالت کلی الگوریتم چندجمله‌ای برای حل آن وجود ندارد. ولی ما فرض می‌کنیم ورودی ما یک گراف ۳ـ رنگ‌پذیر و δـ چگال باشد. می‌خواهیم نشان دهیم در این حالت الگوریتم چندجمله‌ای وحود دارد که می‌تواند یک رنگ‌آمیزی قابل قبول بسازد.

یک زیرمجموعه تصادفی S از رأس‌ها به اندازه‌ی $O\left(\frac{\ln n}{\delta}\right)$ انتخاب می‌کنیم. نشان می‌دهیم به احتمال بالا هر رأس G حداقل یک همسایه در S خواهد داشت. سپس همه‌ی ۳ـ رنگ‌آمیزی‌های ممکن برای S را درنظر می‌گیریم و بررسی می‌کنیم که آیا هرکدام از آن‌ها را می‌توان به یک ۳ـ رنگ‌آمیزی کل گراف G گسترش داد یا خیر ($3^{|S|} = poly(n)$).



شکل ۶: هر رأس خارج از S یک همسایه در S دارد.

هر رأس خارج از S، با احتمال بالا یک همسایه در S دارد. پس برای هر راس در $V - S$ دو گزینه برای رنگ‌آمیزی آن خواهیم داشت. پس طبق تمرین اول سری دوم، می‌توانیم در زمان چندجمله‌ای بررسی کنیم که آیا می‌توان همه رأس‌های مجموعه $V - S$ را با رنگ‌های انتخاب شده به‌طور معتبری رنگ‌آمیزی کرد یا نه. حال چون $3^{|S|} = poly(n)$ پس در کل یک الگوریتم چندجمله‌ای برای مسأله داریم که یک رنگ‌آمیزی برای کل گراف پیدا می‌کند یا می‌فهمد که گراف ۳ـ رنگ‌پذیر نیست.

نحوه انتخاب نمونه تصادفی

حکم : با احتمال بالا تعداد اعضای نمونه کم است.

هر رأس گراف G با احتمال $3c\frac{\ln n}{\delta n}$ انتخاب می‌کنیم. در این‌صورت:

$$E[|S|] = \frac{3c\ln n}{\delta}$$

پس اگر قرار دهیم $\epsilon = 1$، طبق حالت دوم کران چبیشف خواهیم داشت:

$$pr[|S| > \frac{c\delta\ln n}{\delta}] \leq e^{\frac{-\mu\epsilon^2}{3}} e^{\frac{-3c\ln n}{\delta * 3}} = n^{\frac{-c}{\delta}}$$

پس با احتمال بالا تعداد اعضای نمونه کم خواهد بود.

حکم : با احتمال بالا، هر رأس در $V - S$ حداقل یک همسایه در S دارد:

$$E[S \text{ تعداد همسایه‌های یک رأس مشخص خارج از }] > 3c\ln n$$

[15] $\mathbf{NP - hard}$

حال اگر قرار دهیم $\epsilon = 1$ طبق چرنوف:

$$\Pr[S \text{ از خارج مشخص رأس یک همسایه‌های تعداد} = 0] \leq e^{-\mu\frac{\epsilon^2}{2}} = n^{\frac{3}{2}c}$$

$$\Rightarrow \Pr[S \text{ صفر باشد در } V - S \text{ های رأس از یکی همسایه‌های تعداد}] \leq n^{-\frac{3}{2}c+1}$$

پس با احتمال بالا می‌توانیم یک نمونه با ویژگی‌های مد نظرمان بدست آوریم.

مراجع

[۱] درس آنالیز الگوریتم، جلسه اضافه پنجم، کران چرنوف.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه اضافه ۶: کاربرد چندجمله‌ای‌ها در طراحی الگوریتم

نگارنده: سجاد ولایی

جبر از اولین زمینه‌های مورد بررسی ریاضیات و دانشمندان بوده است و پیشرفت در عرصه‌ی جبر چندجمله‌ای ها از قرن ۱۵ شروع شده است. در نگاه اول، شاید قضایا و مسائل جبری جایی در طراحی الگوریتم نداشته باشد. در این مقاله سعی داریم چند مثال از کاربرد چندجمله‌ای‌ها در طراحی الگوریتم را بررسی کنیم.

۱ قضایای ابتدایی

ابتدا به بررسی چند قضیه‌ی ابتدایی میپردازیم. همانطور که می‌دانیم یک چندجمله‌ای بر حسب x و درجه‌ی d به صورت زیر است:

$$P(x) = c_d x^d + c_{d-1} x^{d-1} + ... + c_1 x + c_0$$

توجه کنید که دنباله‌ی $(c_d, c_{d-1}, ..., c_0)$ چندجمله‌ای مورد نظر را به صورت یکتا مشخص می‌کند.

درجه‌ی چند جمله‌ای $P(x)$ برابر بزرگ‌ترین توان x در آن است که معمولا آن را با d نشان می‌دهند. توجه کنید که c_d ناصفر است.

همچنین می‌دانیم که چندجمله‌ای‌ها نسبت به جمع و ضرب بسته‌اند. به این معنا که اگر $G(x)$ و $P(x)$ دو چندجمله‌ای از درجه‌ی d_1 و d_2 باشد. چندجمله‌ای $S(x) = G(x) + P(x)$ وجود دارد که حداکثر درجه‌ی آن بیشینه‌ی d_1 و d_2 است. چندجمله‌ای $R(x) = G(x) \times P(x)$ هم با درجه‌ی $d_1 \times d_2$ وجود دارد.

۱.۱ محاسبهٔ چندجمله‌ای

برای محاسبهٔ چندجمله‌ای یک روش ساده و مستقیم آن است که هر جمله cx^i را به صورت مجزا و با محاسبهٔ x^i و ضرب آن در ضریب c محاسبه کنیم. این مقدار با استفاده از روش سریع‌تری به نام روش هرنر[1] با پیچیدگی زمانی $O(d)$ قابل محاسبه است که d درجه‌ی چندجمله‌ای مورد نظر است.

CALCULATE$(c_d, c_{d-1}, ..., c_0, x)$

1 $ans = 0$
2 **for** $i = d$ to 0
3 $\quad ans = ans \times x + c_i$
4 **return** ans

[1] Horner's Rule

همان‌طور که در شبه‌کد می‌بینید در این روش هر بار یک بار عدد ورودی در جواب آخر ضرب شده و بزرگ‌ترین ضریب اضافه نشده به آن اضافه می‌شود. با بررسی ساده‌ای می‌توان درستی این روش را متوجه شد به این صورت که بعد از اضافه کردن ضریب c_0 و اتمام همه‌ی مراحل عدد ورودی به تعداد i بار در ضریب c_i ضرب شده است.

۲.۱ ریشه‌های چندجمله‌ای

می‌دانیم مجموعه ریشه‌های چندجمله‌ای $P(x)$ مجموعه تمام مقادیر x است که $P(x) = 0$ برقرار باشد.

قضیه ۱ (قضیه اساسی جبر)**.** هر چندجمله ای ناصفر با ضرائب حقیقی و از درجه d دارای حداکثر d ریشه است.

لم ۱. دو چندجمله‌ای متقاوت $A(x)$ و $B(x)$ حداکثر در d نقطه اشتراک دارند.

اثبات. برای اثبات این لم کافی‌است توجه کنیم به این که چندجمله‌ای $Q(x) = A(x) - B(x)$ دارای حداکثر d ریشه است. و هر ریشه‌ی آن نقطه‌ی تقاطع دو چندجمله‌ای مذکور است. □

یکی از نتیجه‌های قضیه‌ی مذکور آن است که با داشتن مقدار چندجمله‌ای در $d + 1$ نقطه‌ی x_0 و x_1 و ... و x_d خود چندجمله‌ای به صورت یکتا مشخص می‌شود.

قضیه ۲ (بازسازی یکتا)**.** برای $d + 1$ زوج مرتب (a_0, b_0) و (a_1, b_1) و ... و (a_d, b_d) در صورتی که a_i ها متمایز باشند دقیقا یک چندجمله‌ای $P(x)$ با حداکثر درجه‌ی d وجود دارد که به ازای هر i داشته باشیم $P(a_i) = b_i$.

اثبات. به ازای هر i $(0 \le i \le d)$ تعریف می‌کنیم:

$$R_i(x) = \frac{\Pi_{j \neq i}(x - a_j)}{\Pi_{j \neq i}(a_i - a_j)}$$

به وضوح $R_i(x)$ چندجمله‌ای درجه d می‌باشد و به ازای هر j $(0 \le j \le d, j \neq i)$ مقدار $R_i(a_j)$ برابر صفر و مقدار $R_i(a_i)$ برابر 1 است. بنابراین $P(x)$ به این صورت ساخته می‌شود:

$$P(x) = \sum_{i=0}^{d} b_i \times R_i(x_i)$$

به وضوح $P(x)$ یک چندجمله‌ای درجه d است که شرط مساله را راضی می‌کند و با توجه به لم ۱ تنها چندجمله‌ای است که در تمام $d + 1$ نقطه صدق می‌کند.

□

۳.۱ چندجمله‌ای‌های چندمتغیره

درجه‌ی یک چندجمله‌ای برابر بزرگ‌ترین درجه‌ی جمله‌های آن است. درجه‌ی یک جمله‌ی چندمتغیره برابر مجموع توان‌های متغیر هاست. مثال: درجه‌ی جمله‌ی $cx_1^{i_1}x_2^{i_2}x_3^{i_3}$ برابر $i_1 + i_2 + i_3$ است. ⊠

مثال: درجه‌ی چندجمله‌ای $P(x_1, x_2, x_3, x_4) = x_1x_2^2x_3 + x_3x_4^2 + x_1x_2^2x_3^2x_4$ برابر ۶ است.

⊠

قضیه ۳. [شوارتز ـ زیپل[۲]] اگر $P(x)$ یک چندجمله‌ای ناصفر، m ـ متغیره و با درجه‌ی d باشد و S زیرمجموعه‌ی میدان F باشد و متغیرهای تصادفی X_i به صورت مستقل از هم از زیرمجموعه‌ی S انتخاب شوند داریم:

$$Pr[P(X_1, X_2, ..., X_m) = 0] \leq \frac{d}{|S|}$$

۲ کاربرد ها

۱.۲ مساله‌ی تصحیح کدهای اشتباه

۱.۱.۲ مدل اول

می‌خواهیم دنباله‌ی $\mathrm{x}\langle a_d, a_{d-1}, ...a_0\rangle$ را به عنوان پیام به مقصدی برسانیم. برای این منظور دنباله اعدادی را از طریق کانال نویزداری ارسال می‌کنیم. همچنین می‌خواهیم کمترین تعداد عدد را از طریق کانال بفرستیم. حداکثر k عدد از اعدادی که از طریق کانال فرستاده می‌شوند حذف خواهند شد. توجه کنید که حذف اعداد جایگاهشان را تغییر نخواهد داد. به این معنی که اگر عددی حذف نشده باشد و در جایگاه i ام دنباله‌ی اولیه بوده است پس از حذف اعداد نیز هم‌چنان در همان جایگاه i ام خواهد ماند.

مثال: اگر دنباله‌ی فرستاده شده به صورت $\langle 5, 19, 2, 3, 2\rangle$ باشد و k برابر ۲ باشد دنباله‌ی خروجی کانال ممکن است به صورت $\langle 5, *, 2, 3, *\rangle$ باشد. ⊠

یک راه‌حل ساده این است که هر عدد دنباله را با $k + 1$ تکرار پشت سرهم بفرستیم تا حذف‌شدن هیچ k عددی باعث از بین رفتن هیچ‌کدام از اعداد پیام نشود. در این روش $(k + 1)(d + 1)$ عدد باید از طریق کانال فرستاده شود.

مثال: اگر دنباله‌ی پیام به صورت $\langle 5, 19, 2, 3, 2\rangle$ باشد و k برابر ۲ باشد. با فرستادن دنباله‌ی $\langle 5, 5, 5, 19, 19, 19, 2, 2, 2, 3, 3, 3, 2, 2, 2\rangle$ می‌توان پیام را انتقال داد. به وضوح با حذف هیچ دو عددی پیام اصلی خدشه‌دار نخواهد شد. ⊠

حال می‌خواهیم راه حلی ارائه دهیم که با فرستادن دنباله‌ی d+k+۱ تایی پیام را ارسال کند. برای این منظور چندجمله‌ای $P(x)$ را به صورت زیر تعریف می‌کنیم:

$$P(x) = \sum_{i=0}^{d} a_i \times x_i$$

همانطور که مشخص است دنباله‌ی پیام همان دنباله‌ی ضرایب این چندجمله‌ای است. دنباله‌ای که از طریق کانال می‌فرستیم به صورت $\langle P(0), P(1), P(2), ..., P(d + k - 1), P(d + k)\rangle$ خواهد بود. در صورت حذف حداکثر k عدد دنباله، حداقل $d + 1$ عدد باقی می‌ماند. و با توجه به خاصیت حفظ جایگاه این $d + 1$ عدد، در دنباله‌ی خروجی کانال می‌توان حداقل $d + 1$ جایگاه $x_0, x_1, ..., x_d$ یافت که عدد x_iام حذف نشده باشد و در نتیجه $d + 1$ زوج مرتب به صورت $(x_i, P(x_i))$ وجود دارد. بنابر قضیه‌ی ۲ تنها یک چندجمله‌ای وجود دارد که نقاط آن در زوج مرتب‌ها صدق کند. که این چند جمله‌ای به وضوح همان $P(x)$ است. و در نتیجه ضرایب این چند جمله‌ای دنباله‌ی پیام را مشخص می‌کند.

[۲] Schwartz–Zippel

۲.۱.۲ مدل دوم

این مدل از مساله همانند مدل قبل است با این تفاوت که دیگر k عدد حذف نمی‌شود بلکه اعدادی جایگزین k عدد از دنباله‌ی ورودی می‌شوند. همانند مدل قبل روش ساده و نزدیک به ذهنی برای حل این مساله وجود دارد به این صورت که از هر عدد دنباله‌ی پیام $2 \times k + 1$ تکرار در دنباله‌ی اولیه قرار دهیم و در آخر در دنباله‌ی تغییریافته در هر بخش $2 \times k + 1$ عددی دنباله (که متعلق به یک عدد دنباله‌ی پیام است) عدد غالب(عددی که بیشتر از نصف این بخش را پوشانده) عدد متناظر تمام بخش مذکور است.

راه‌حل بهتری مانند مدل قبل برای این مساله موجود است که آن را به نام الگوریتم (برلکمپ_ولچ)[۳] می‌شناسند. مانند مدل قبل چندجمله‌ای $P(x)$ را به صورت زیر تعریف می‌کنیم:

$$P(x) = \sum_{i=0}^{d} a_i \times x_i$$

می‌دانیم دنباله‌ی پیام همان دنباله‌ی ضرایب این چندجمله‌ای است. دنباله‌ای که از طریق کانال می‌فرستیم به صورت زیر خواهد بود.

$$\langle P(0), P(1), P(2), ..., P(d + 2 \times k - 1), P(d + 2 \times k) \rangle$$

حال با فرض تغییر یافتن حداکثر k عدد دنباله $P(x)$ را می‌یابیم. هر چندجمله‌ای که $d + k + 1$ نقطه از نقاط داخل دنباله‌ی تغییر یافته را در نمودار خود داشته باشد حداقل $d + 1$ نقطه از نمودار $P(x)$ را دارد که طبق قضیه‌ی ۱ چندجمله‌ای مذکور با $P(x)$ برابر است. و چون حتما چنین چندجمله‌ای وجود دارد پس دنباله‌ی تغییر یافته به صورت یکتا می‌تواند چندجمله‌ای $P(x)$ را مشخص کند.

در ادامه به این می‌پردازم که چگونه چندجمله‌ای مورد نظر را بیابیم. فرض کنید دنباله‌ی تغییر یافته برابر $\langle r_0, r_1, ..., r_{d+2\times k+1} \rangle$ باشد. مجموعه‌ی Z را به این صورت تعریف می‌کنیم:

$$Z = \{i | (0 \leq i \leq d + 2 \times k + 1), r_i \neq P(i)\}$$

به وضوح اندازه‌ی مجموعه‌ی Z حداکثر برابر k است. تعریف می‌کنیم:

$$E(x) = \Pi_{i \in Z}(x - i)$$

با توجه به اینکه مقدار $E(x)$ در دامنه‌ی اعضای Z صفر است، داریم:

$$\forall x (0 \leq x \leq d + 2 \times k + 1) : P(x) \times E(x) = r_x \times E(x)$$

چندجمله‌ای‌های $E(x)$ و $\times E(x)P(x)$ را ما ضرایب مجهول می‌نویسیم:

$$E(x) = x^k + e_{k-1}x^{k-1} + e_{k-2}x^{k-2} + ... + e_0$$

$$P(x) \times E(x) = f_{d+k}x^{d+k} + f_{d+k-1}x^{d+k-1} + ... + f_0$$

در نتیجه برای $d + 2 \times k + 1$ مقدار x داریم:

[3] Berlekamp-Welch

$$(1 \leq x \leq d+2\times k+1) \qquad f_{d+k}x^{d+k} + f_{d+k-1}x^{d+k-1} + ... + f_0 = r_x(x^k + e_{k-1}x^{k-1} + e_{k-2}x^{k-2} + ... + e_0)$$

تعداد مجهولات ما برابر تعداد f_iها و تعداد e_i هاست که در کل برابر $d+2\times k+1$ است. با توجه به اینکه تعداد معادلات هم برابر با همین تعداد است می‌توان مجهولات را بدست آورد. پس از بدست آوردن مجهولات و در نتیجه بدست آوردن دو چندجمله‌ای $P(x) \times E(x)$ و $E(x)$ میتوان چندجمله‌ای $P(x)$ را از تقسیم آن‌ها به صورت $\frac{P(x)\times E(x)}{E(x)}$ بدست آورد. اگر برای بدست آوردن مجهولات از روش حذف گاوسی[۴] استفاده شود پیچیدگی زمانی این الگوریتم برابر $O(n)$ است.

۲.۲ یافتن تطابق

۱.۲.۲ یافتن تطابق کامل

برای هر گراف $G = (V, E)$ با راس‌های $v_1, v_2, ..., v_n$ ماتریس $|V|\times|V|$ تات[۵] را به صورت زیر تعریف می‌کنیم:

$$M(G)_{i,j} = \begin{cases} x_{i,j} & i < j \text{ و } (v_i, v_j) \in E \text{ اگر} \\ -x_{j,i} & i > j \text{ و } (v_i, v_j) \in E \text{ اگر} \\ 0 & (v_i, v_j) \notin E \text{ اگر} \end{cases}$$

دترمینان این ماتریس چندجمله‌ای چندمتغیره‌ی $P_G(x)$ است که حداکثر درجه‌ی آن $|V|$ است. داریم:

قضیه ۴ (تات). یک گراف G دارای تطابق کامل است اگر و فقط اگر $P_G(x)$ چندجمله‌ای متحد با صفر نباشد.

در نتیجه برای اثبات وجود تطابق کامل در گراف G کافی‌است که ثابت کنیم $P_G(x)$ ناصفر است. یافتن دترمینان ماتریس مذکور ممکن است $|V|!$ عملیات زمان ببرد. اما با استفاده از قضیه ۳ اگر متغیرهای چندجمله‌ای $P_G(x)$ را از زیر مجموعه‌ای از اعداد مانند S به صورت تصادفی و مستقل از هم انتخاب کنیم و مقدار آن را محاسبه کنیم، می‌دانیم که در صورت ناصفر بودن چندجمله‌ای مذکور آنگاه احتمال صفر شدن مقدار محاسبه شده برابر $\frac{|V|}{|S|}$ است و چون اندازه‌ی زیرمجموعه‌ی S هر قدر می‌تواند بزرگ باشد این احتمال ناچیز است. در نتیجه همان‌طور که گفته شد مقدار چندجمله‌ای مورد نظر را محاسبه کرده و اگر این مقدار ناصفر بود به این معناست که چندجمله‌ای متحد با صفر نیست و گراف G دارای تطابق کامل است و اگر برابر صفر بود آنگاه جز به احتمال ناچیزی گراف G دارای تطابق کامل نیست.

گفتنی است که دترمینان یک ماتریس $n \times n$ را می‌توان با پیچیدگی زمانی $O(n^{2.373})$ یافت.

۲.۲.۲ تطابق بیشینه

برای آنکه دریابیم که آیا گراف G تطابق به اندازه‌ی k دارد یا نه، کافی است گراف H را از گراف G بسازیم به این صورت که H دارای گراف G به عنوان زیرگراف القایی به علاوه‌ی مجموعه رئوس A که $|V|-2\times k$ راس است و همه‌ی این راس ها به همه‌ی راس‌های G متصل هستند. حال کافی‌است گراف H دارای تطابق کامل باشد آنگاه به این دلیل که راس های مجموعه‌ی A حداکثر $n-2\times k$ راس از G را داخل تطابق پوشانده اند G حداقل تطابق به اندازه k باید داشته باشد تا H دارای تطابق کامل بشود.

[4] Gaussian Elimination
[5] Tutte

۳.۲ اثبات تعاملی مکمل مساله‌ی ۳ـ صدق‌پذیری

می‌دانیم مسائل ان‌پی مسائلی هستند که الگوریتم بررسی‌کننده‌ای[۶] با زمان چندجمله‌ای می‌تواند درستی جواب بله آن‌ها را که از الگوریتم اثبات‌کننده[۷] میگیرد بررسی کند و درست یا غلط آن را تشخیص دهد. در مقابل مسائل ان‌پی‌ـ‌مکمل نیز آن دسته مسائلی هستند که بررسی‌کننده‌ای با زمان چندجمله‌ای می‌تواند جواب نه را در آن‌ها بررسی کند. توجه کنید که اثبات‌کننده دارای توان محاسباتی نامحدود است.

مثال: مساله‌ی ۳ـ صدق‌پذیری [۸] یک مساله‌ی ان‌پی است. این جمله بدین معناست که اگر اثبات‌کننده برای ورودی مساله‌ی ۳ـ صدق‌پذیری حالتی از درست و نادرست بودن متغیرها تحویل دهد آنگاه بررسی‌کننده‌ای وجود دارد که با زمان چندجمله‌ای می‌تواند تشخیص دهد که این حالت می‌تواند شرط‌های مساله‌ی ۳ـ صدق‌پذیری را ارضا کند یا خیر. که به روش ساده‌ای می‌توان با جای‌گذاری متغیرها و بررسی همه‌ی شرط‌ها با پیچیدگی زمانی از مرتبه‌ی طول ورودی درستی یا نادرستی ادعای اثبات‌کننده را تشخیص داد. ☒

مکمل مساله‌ی ۳ـ صدق‌پذیری یک مساله‌ی ان‌پی‌ـ‌مکمل است به این معنا که اثبات‌کننده باید بررسی‌کننده را قانع کند که شروط ورودی صدق‌پذیر نیست. تاکنون بررسی‌کننده‌ای برای این مساله وجود نداشته است. اما این مساله دارای اثبات تعاملی[۹] است که در ادامه به آن می‌پردازیم. به این معنا که به اندازه‌ی چندجمله‌ای‌بار بررسی‌کننده می‌تواند از اثبات‌کننده سوال بپرسد و اثبات‌کننده در همان لحظه جواب آن را بدهد. توجه کنید اثبات‌کننده همه‌چیزدان است و وقتی از چندجمله‌ای صحبت می‌کنیم منظورمان چندجمله‌ای برحسب طول ورودی است. این سری مسائل که الگوریتم بررسی‌کننده به صورت تعاملی در زمان چندجمله‌ای درستی ادعای اثبات‌کننده را بررسی می‌کند را آی‌پی مینامیم. گفتنی است که مسائل ان‌پی‌ـ‌مکمل زیر مجموعه‌ی مسائل آی‌پی هستند.

مساله‌ی قوی‌تری تعریف می‌کنیم به این صورت که تعداد حالت‌های درست و نادرست متغیرها که همه‌ی شروط را ارضا کند چندتاست. آیا این مقدار برابر k است؟ می‌دانیم تعداد کل این حالات برابر 2^n است. بررسی جواب صفر مساله‌ی جدید همان مساله‌ی مکمل ۳ـ صدق‌پذیری است. برای حل این مساله ابتدا چندجمله‌ای nـ متغیره $P(x)$ را به این صورت تعریف می‌کنیم که n متغیر مساله‌ی ۳ـ صدق‌پذیری در چندجمله‌ای وجود داشته باشند و درست بودن یک متغیر در مساله متناظر ۱ بودن آن در چندجمله‌ای و نادرست بودن آن متغیر متناظر صفر بودنش در چندجمله‌ای باشد و در صورت راضی‌شدن همه‌ی شروط مقدار محاسبه‌شده‌ی چند جمله‌ای برابر یک و در غیر این صورت برابر صفر شود. به راحتی می‌توان دید برای هر شرط t مساله که به صورت $(x_i \vee x_j \vee x_k)$ است چندجمله‌ای درجه سه‌ای وجود دارد مانند $P_t(x) = (1-(1-x_i)(1-x_j)(1-x_k))$ که اگر شرط ارضا شود این چندجمله‌ای همان‌طور که پیش‌تر توضیح داده شد برابر یک می‌شود.(اگر به جای x_r در شرط $\bar{x_r}$ وجود داشته باشد کافی‌است در چندجمله‌ای متناظرش به جای $(1-x_r)$ از خود x_r استفاده کنیم). تعریف می‌کنیم:

$$P(x) = \prod_{i=0}^{m} P_i(x)$$

راضی شدن همه‌ی شرط‌ها بدین معناست که تمام $P_i(x)$ها برابر یک شود و درنتیجه چندجمله‌ای $P(x)$ برابر یک خواهد شد و اگر یکی از شرط‌ها ارضا نشود برابر صفر خواهد شد. درجه‌ی $P(x)$ حداکثر برابر $3 \times m$ است. همه‌ی این چندجمله‌ای ها را در میدان $\mathbf{Z}_p$ تعریف می‌کنیم که $p \geq 2^{2n}$ و عدد اول است. حال مساله‌ی ما تبدیل به بررسی درستی این نامساوی خواهد شد:

$$\sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \cdots \sum_{x_n=0}^{1} P(x_1, x_2, ..., x_n) = k$$

توجه کنید که محاسبهٔ این نامساوی با توان محاسباتی چندجمله‌ای امکان پذیر نیست. می‌خواهیم اثباتی تعاملی برای این نامساوی بیاوریم.

[6] verifier
[7] prover
[8] 3-SAT
[9] interactive

الگوریتم بررسی‌کننده را V و الگوریتم اثبات‌کننده را Q می‌نامیم. الگوریتم به این صورت خواهد بود:

۱. ابتدا V از Q چندجمله‌ای $G(y) = \sum_{x_2=0}^{1} \sum_{x_3=0}^{1} ... \sum_{x_n=0}^{1} P(y, x_2, x_3, ..., x_n)$ را طلب می‌کند.

۲. Q چندجمله‌ای $G'(y)$ را می‌فرستد.

۳. V بررسی می‌کند که آیا $G'(0) + G'(1) = k$ است یا نه. اگر برابر نباشد مستقل از آنکه $G(y)$ برابر $G'(y)$ باشد یا نباشد جواب V منفی‌است و الگوریتم تمام می‌شود.

۴. V عدد $q_1 \in \mathbf{Z_p}$ را به صورت تصادفی انتخاب می‌کند. فرض کنید $k_2 = G'(q_1)$.

۵. V از Q می‌خواهد که اثبات کند $G(q_1) = k_2$.

داریم:

$$G(q_1) = \sum_{x_2=0}^{1} \sum_{x_3=0}^{1} ... \sum_{x_n=0}^{1} P(q_1, x_2, x_3, ..., x_n) \stackrel{?}{=} k_2$$

چون q ثابت است می‌توانیم بنویسیم:

$$P^2(x_2, x_3, ..., x_n) = P(q_1, x_2, x_3, ..., x_n)$$
$$G(q_1) = \sum_{x_2=0}^{1} \sum_{x_3=0}^{1} ... \sum_{x_n=0} P^2(x_2, x_3, ..., x_n) \stackrel{?}{=} k_2$$

باید Q به V ثابت کند که:

$$x_1 = q_1$$
$$\sum_{x_2=0}^{1} \sum_{x_3=0}^{1} ... \sum_{x_n=0} P^2(x_1, x_2, ..., x_{n-1}) = k_2$$

بنابراین با درخواست قسمت ۵ الگوریتم توانستیم با حذف یک متغیر دوباره به مساله‌ای مانند مساله‌ی اول برسیم. با تکرار همین الگوریتم از این نامساوی به نامساوی زیر میرسیم:

$$x_2 = q_2$$
$$\sum_{x_3=0}^{1} \sum_{x_4=0}^{1} ... \sum_{x_n=0} P^2(x_1, x_2, ..., x_{n-2}) = k_2$$

و آنقدر این الگوریتم را تکرار می‌کنیم تا به نامساوی $\sum_{x_n=0}^{1} P^n(x_n) = k_n$ برسیم و با یک‌بار تکرار همان الگوریتم به $P^{n+1} = k_{n+1} = P(q_1, q_2, ..., q_n)$ می‌رسیم که V می‌تواند آن را محاسبه کند و درستی نامساوی را بررسی کند و اگر نامساوی برقرار نبود جواب منفی برگرداند. توجه کنید که طی الگوریتم اگر یک‌بار جواب قسمت سه‌ی الگوریتم منفی باشد جواب کل منفی می‌شود.

حال می‌خواهیم ثابت کنیم احتمال خطای V ناچیز است. اگر الگوریتم اثبات‌کننده همواره حقیقت جواب را ارسال کند به وضوح هیچ خطایی رخ نخواهد داد. اما اگر بخواهد فریب دهد ممکن است جواب مثبت V اشتباه باشد. در هربار که قسمت‌های ۱ تا ۵ الگوریتم اجرا می‌شود اگر Q تابع $G'(x)$ را در قسمت اول الگوریتم صادقانه به V بفرستد مشکلی وجود ندارد اما اگر $G(x) \neq G'(x)$ آنگاه احتمال آنکه برای q(عدد تصادفی که در قسمت ۴ انتخاب می‌شود) نامساوی $G(q) = G'(q)$ برقرار باشد طبق قضیه‌ی ۳ برابر $\frac{d}{p} = \frac{d}{2^{2n}}$ است که d حداکثر برابر $n \times 3$ می‌باشد. اگر ادعای Q در مرحله‌ای نادرست باشد به احتمال $(1 - \frac{d}{p})$ در مرحله‌ی بعدی نیز V درحال اثبات ادعای نادرست است(یعنی خطایی رخ نداده) و طی n مرحله اجرای الگوریتم احتمال کلی خطا حداکثر به مقدار $\frac{nd}{2^{2n}} = \frac{3 \times n^2}{2^{2n}}$ است. فلذا V می‌تواند جز با احتمال ناچیزی درستی یا نادرستی ادعای Q مبنی بر تعداد حالات ارضای شروط تشخیص دهد.

مراجع

[۱] http://www.cs.cmu.edu/ 15451-s20/lectures/lec25.pdf

[۲] https://en.wikipedia.org/wiki/Schwartz%E2%80%93Zippel_lemma

[۳] https://en.wikipedia.org/wiki/Co-NP

[۴] https://fa.wikipedia.org/wiki/%D9%85%D8%B3%D8%A6%D9%84%D9%87_%D8%B5%D8%AF%D9%82%E2%8

[۵] https://fa.wikipedia.org/wiki/%D8%A7%D9%86%E2%80%8C%D9%BE%DB%8C_%DA%A9%D8%A7%D9%85%

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

نمونه‌گیری[1]، مسأله بستار متعدی

نگارنده: آرمیتا کاظمی نجف آبادی

موضوع مورد بحث این جلسه، نمونه‌گیری تصادفی است که قبلا هم با آن در جلسات مربوط به کران چبیشف (پیدا کردن میانه) و چرنف (در مسأله سه_رنگ آمیزی گراف های چگال) و جلسه حل تمرین ۱۳، برخورد کرده ایم.
از آنجایی که نمونه‌گیری ایده ای پراهمیت است، خوب است بررسی بیشتری روی آن انجام دهیم.

۱ ایده کلی نمونه‌گیری

ایده ی این مبحث به طور کلی این است که از یک داده یا ورودی بزرگ، بخش کوچکی را انتخاب کرده و الگوریتمی که قرار بود روی کل ورودی اجرا کنیم، تنها روی همان بخش کوچک (نمونه) اجرا کنیم یا اگر می‌خواهیم تحلیلی روی یک ورودی بزرگ انجام دهیم، روی بخش کوچکتری از نمونه تحلیل را انجام دهیم و بر اساس آن، برای کل ورودی تحلیل ارائه دهیم.
مزیت این روش، افزایش سرعتِ ارائه‌ی تحلیل، بر اثر کاهش زمان اجرای الگوریتم است. (طبعا وقتی زمان اجرای الگوریتم به اندازه ورودی وابسته باشد، هر چه اندازه ورودی کوچکتر باشد زمان اجرا کمتر است.)
عیب این روش نیز از بین رفتن دقت است. ممکن است نتیجه حاصل از بررسی نمونه با نتیجه ای که از بررسی کل ورودی حاصل می‌شد تفاوت زیادی داشته باشد. (خطای آن زیاد باشد.)
از اهداف ما این است که تعادلی بین سرعت و خطا ایجاد کنیم و درواقع تحلیل کنیم که برای رسیدن به یک سرعت خاص، طوری که خطا هم از حدی معین بیشتر نباشد؛ عمل نمونه‌گیری به چه شکلی انجام شود.

۲ مسأله نظرسنجی[2]

شاید ساده ترین مثال برای اِعمال ایده نمونه‌گیری، مسأله نظرسنجی باشد.
این مسأله را در جلسه ۱۳ حل تمرین نیز بررسی کرده بودیم ولی در اینجا نکات بیشتری را مطرح می‌کنیم.
فرض کنیم یک جمعیت داریم که هر فرد از آن، در مورد یک مسأله مشخص، نظر مثبت یا منفی دارد.
می‌خواهیم بدانیم چه درصدی از جمعیت، نظری مثبت درباره این مسأله مشخص دارند.
روش نمونه‌گیری اینطور عمل می‌کند که نمونه ای تصادفی از جمعیت انتخاب کرده و از روی نتیجه نظرسنجی برای آن نمونه، به برآوردی از نظرات کل جمعیت می‌رسد.
مسأله را با زبانی دیگر بیان می‌کنیم: فرض کنید n گوی در دو رنگ آبی و قرمز داریم. هر گوی به احتمال p آبی است. (np گوی، آبی و بقیه قرمز هستند.)

[1] random sampling
[2] polling

می‌خواهیم بدانیم چند درصد توپ ها آبی ست. یا به عبارتی می‌خواهیم p را تخمین بزنیم.
برای تخمین p می‌بایست مشخص کنیم چه دقت تخمین و چه احتمال خطایی برایمان قابل پذیرش است.

الگوریتم های FPRAS : برخی اوقات الگوریتمی‌احتمالاتی داریم که احتمال عدم موفقیتی در ارائه تخمین مناسب دارد و همچنین تخمینی که ارائه می‌دهد نیز تقریبی از جواب واقعی مسأله است.
اگر جواب مسأله واقعی را p بنامیم، یک زوج (ϵ, δ) می‌توان به الگوریتم نسبت داد که نشان می‌دهد این الگوریتم به احتمال حداقل $1-\delta$ جوابی می‌دهد که بین $(1-\epsilon)p$ و $(1+\epsilon)p$ باشد.
به این الگوریتم ها FPRAS می‌گویند.
زمان اجرای الگوریتم نیز باید تابعی از ϵ و δ باشد یا شاید بهتر باشد بگوییم تابعی از $\frac{1}{\epsilon}$ و $\frac{1}{\delta}$.
اگر بخواهیم دقیق تر بگوییم، زمان اجرای این مدل از الگوریتم ها بر حسب $(n, \frac{1}{\epsilon})$ (که n طول ورودی ست) چندجمله‌ای ست و احتمال موفقیت آن ثابت است. (موفقیت به این معناست که خروجی الگوریتم در بازه $(1-\epsilon)p$ و $(1+\epsilon)p$ بیفتد.)
در این تعریف δ نقشی ندارد. برای آنکه به الگوریتمی برسیم که احتمال خطای آن نیز حداکثر δ باشد، می‌توانیم الگوریتم اخیر با احتمال موفقیت ثابت را آن قدر تکرار کنیم که احتمال خطا پایین بیاید.
می‌توان نشان داد اگر $O\left(\frac{1}{\delta}\right)$ بار الگوریتم را تکرار کرده و سپس میانه ی تخمین های به دست آمده از هر بار اجرا را به عنوان تخمین p خروجی دهیم، احتمال خطا حداکثر δ خواهد بود.
بنابراین می‌توانیم پارامتر δ را در خودِ تعریف وارد نکنیم و با تکرار همان الگوریتم به میزان کافی، احتمال خطا را از δ ی معین کمتر کنیم.

دنبال یک الگوریتم تقریبی تصادفی چندجمله‌ای با پارامتر های داده شده ی ϵ و δ (الگوریتم FPRAS) برای مسأله نظرسنجی هستیم.
به این الگوریتم توجه کنید: ابتدا k توپ را به طور تصادفی با جایگذاری نمونه‌گیری می‌کنیم. (جایگذاری باعث می‌شود نمونه‌گیری ها از هم مستقل باشند و تحلیل ساده شود. به علاوه وقتی ورودی بزرگ باشد، احتمال اینکه عضوی تکراری انتخاب کنیم؛ مثلا از یک آدم دو بار نظرسنجی کنیم، کم است.)
سپس اگر تعداد آبی ها در این نمونه l باشد. p را $\frac{l}{k}$ تخمین زده و خروجی دهیم.
برای تحلیل بهتر می‌توان X_i را متغیر تصادفی شاخصِ آبی بودنِ نمونه ی iـام در نظر گرفت.
در این صورت $X = \sum_{i=1}^{k} X_i$ متغیر تصادفی تعداد آبی های نمونه ی k عضوی از ورودی هاست.
و همینطور $E\left[\frac{X}{k}\right] = p$ است؛
چون شاخص بودنِ متغیرِ X_i نتیجه می‌دهد که $E[X_i] = P_r\left(X_i = 1\right) = p$ باشد و خطی بودنِ امید ریاضی نتیجه می‌دهد $E[X] = E[\sum_{i=1}^{k} X_i] = \sum_{i=1}^{k} E[X_i] = kp$ بنابراین $E[\frac{X}{k}] = p \Rightarrow \frac{E[X]}{k} = p \Rightarrow E[X] = kp$ است.

بنابراین اگر خروجی الگوریتم $\frac{X}{k}$ باشد، متوسط مقدار خروجی آن، همان p است. در چنین حالتی (که متوسط تخمین خروجی داده شده توسط الگوریتم برابر مقدار واقعی باشد)، به الگوریتم، یک تخمین‌گر ناریب [3] می‌گوییم.
اما بعضا پیدا کردن یک تخمین‌گر ناریب برای یک مسأله خیلی هم سخت نیست. نکته اصلی این است که نشان دهیم تخمین مان با احتمال خوبی به مقدار واقعی نزدیک است. (مثلا برای یک تخمین‌گر ناریب که FPRAS است، پارامتر های ϵ و δ به قدر کافی کم باشد.) این مطلب را به کمک کران هایی که قبلا دیده ایم (کران مارکف، چبیشف و چرنف) نشان می‌دهیم.
مثلا برای این الگوریتم با توجه به کران چرنف و با فرض $\epsilon \leqslant 1$ می‌توانیم بنویسیم:

$$P_r\left[X > (1+\epsilon)\,\mu\right] \leqslant e^{\frac{-\mu\epsilon^2}{3}}$$

در اینجا $\mu = kp$ است. می‌خواهیم احتمال عدم موفقیت حداکثر δ باشد. برای این منظور می‌بایست $e^{\frac{-kp\epsilon^2}{3}} \leqslant \delta$ باشد. طرفین نامساوی را معکوس کرده، لگاریتم می‌گیریم. بنابراین می‌بایست $kp \geqslant \frac{3}{\epsilon^2}\ln\frac{1}{\delta}$ یا $k \geqslant \frac{1}{p}\frac{3}{\epsilon^2}\ln\frac{1}{\delta}$ باشد.
یعنی در صورتی که تعداد نمونه هایمان $k = \Omega\left(\frac{1}{p} \times \frac{1}{\epsilon^2} \times \ln\frac{1}{\delta}\right)$ باشد، به احتمال حداقل $1-\delta$، تخمین مان از $(1+\epsilon)p$ بیشتر نیست.

[3] unbiased estimator

(یا به عبارتی تخمین مان به احتمال زیادی، بیش از ϵp از مقدار واقعی که همان p است، فاصله ندارد.)
می‌توان به طور شهودی وابستگی k به هر یک از مقادیر $\frac{1}{p}$، $\frac{1}{\epsilon^2}$ و $\ln \frac{1}{\delta}$ را بررسی کرد.
اگر p خیلی کم باشد، امید تعداد دفعات لازم نمونه‌گیری برای دیدن یک گوی آبی $\frac{1}{p}$ است پس مثلا حداقل $\frac{1}{p}$ بار باید نمونه‌گیری را انجام دهیم تا بتوانیم تخمینی مناسب از p ارائه دهیم. (در غیر اینصورت احتمالا تخمین مان از p برابر صفر خواهد بود چون در تعداد کمتر نمونه‌گیری، امیدی به دیدن گوی آبی نداریم.)
به طور شهودی $\ln \frac{1}{\delta}$ نیز به ایده های تکرار و کاهش خطا وابسته ست.
همین طور در مورد شهود برای $\frac{1}{\epsilon^2}$ می‌توان گفت به این خاطر است که احتمال دور شدن از میانگین به اندازه $\epsilon\mu$، نزدیک به احتمال دور شدن به اندازه حداقل یک انحراف معیار از میانگین شود.
در این الگوریتم چون وابستگی k به δ، لگاریتمی ست، اگر بخواهیم درجه اطمینان از تخمین (احتمال موفقیت) را بیشتر کنیم؛ خیلی لازم نیست نمونه های بیشتری بگیریم. اما در عوض، برای آنکه مقدار تخمین زده شده به مقدار واقعی نزدیک تر شود، می‌بایست تلاش بیشتری کنیم (تعداد نمونه های خیلی بیشتری بگیریم.)

تا اینجا یک مشکل اساسی را نادیده گرفتیم و آن، وابستگی k به مقدار p است. و با توجه به اینکه از ابتدا می‌خواستیم p را تخمین بزنیم، یک حالت دوری ایجاد می‌شود.
می‌توان تصور کرد مسأله ای که می‌خواهیم حل کنیم، ابتدا یک تخمین نه چندان بد از p دارد و هدف ما دقیق تر کردن آن است.
اگر هم از ابتدا ایده ای درمورد حدود p نداشته باشیم باز راهکار هایی برای تخمین آن وجود دارد.
مثلا در ابتدا فرض کنیم p عدد بزرگی ست و بر همین اساس نمونه‌گیری را انجام دهیم. اگر در نمونه k عضوی مان، تعداد آبی ها زیاد بود، فرض بزرگی p تایید می‌شود وگرنه تخمین مان از p را پایین تر آورده و تعداد بیشتری نمونه می‌گیریم تا وقتی که فرضمان از حدود p با تخمینی که از نمونه به دست می‌آید، همخوان باشد.
به طور شهودی می‌خواهیم آن قدر نمونه‌گیری را انجام دهیم تا تعداد موفقیت ها با فرضی که روی p گذاشته ایم سازگار باشد.

می‌خواهیم بنابر آنچه گفته شد الگوریتم را تغییر دهیم. قبل از آن به این نکته توجه می‌کنیم که برای این k که در قسمت های قبل محاسبه کردیم، امید تعداد موفقیت ها در نمونه ی تصادفی k عضوی مان، kp است. پس طبق نامساوی های قبلی، متوسط تعداد موفقیت های دیده شده حداقل $\frac{3}{\epsilon^2} \ln \frac{1}{\delta}$ است.
می‌توان الگوریتم را به این شکل تغییر داد: آن قدر عمل نمونه‌گیری را تکرار کنیم تا $\mu = \frac{3}{\epsilon^2} \ln \frac{1}{\delta}$ موفقیت ببینیم.
اگر وقتی μ بار موفقیت می‌بینیم، k نمونه گرفته باشیم؛ تخمین مان از p را $\frac{\mu}{k}$ قرار می‌دهیم.
در اینجا مشابه استدلال های قبلی، کران چرنف به ما می‌گوید که اگر k بزرگتر یا مساوی $\frac{1}{p}\frac{3}{\epsilon^2} \ln \frac{1}{\delta}$ باشد، احتمال اینکه تعداد موفقیت های دیده شده خارج از بازه ی $\mu(1-\epsilon)$ و $\mu(1+\epsilon)$ باشد؛ کمتر از $\frac{1}{\delta}$ است.
به علاوه باز به کمک چرنف می‌توان نشان داد احتمال اینکه قبل از $\frac{\mu}{p}(1-\epsilon)$ نمونه، μ موفقیت ببینیم، کم است.
یعنی انتظار داریم بعد از $\frac{\mu}{p}$ بار نمونه‌گیری، μ موفقیت ببینیم. (احتمال اینکه قبل از این تعداد نمونه‌گیری μ موفقیت ببینیم، طبق چرنف کم است.)
به طور مشابه هم می‌توان نشان داد احتمال اینکه بعد از $\frac{\mu}{p}(1+\epsilon)$ نمونه، μ موفقیت ببینیم، کم است.

به طور شهودی گفته بودیم اگر اندازه ورودی بزرگ باشد، خیلی فرقی ندارد که مسأله را با جایگذاری یا بدون آن حل کنیم و شاید حتی به طور کلی شهوداً بتوان گفت بدون جایگذاری، اوضاع بهتر هم می‌شود. چون مثلا فرض کنید نمونه به دست آمده در حالت بدون جایگذاری، تعداد گوی های آبی اش بیشتر از امید تعداد آبی هایی که باید می‌دیدیم باشد. بنابراین نسبت گوی های قرمز داخل کیسه بیشتر از نسبت اولیه می‌شود و شانس اینکه در نمونه‌گیری های بعد گویی قرمز به دست آید بیشتر می‌شود.
انگار اگر جایگذاری نداشته باشیم، خود به خود به سمت برقراری توازن و μ واقعی سوق داده می‌شویم.

یک نمادگذاری کوچک: همان طور که در الگوریتم پیشین دیدیم، رویکرد این بود که نمونه‌گیری را آن قدر انجام دهیم تا μ موفقیت ببینیم. مقدار این μ نیز در اکثر تحلیل هایمان به پارامتر های ϵ و δ بستگی دارد. بنابراین در ادامه به جای μ از نماد $\mu_{\epsilon\delta}$ استفاده می‌کنیم.

۳ مسأله بستار متعدی[۴]

بستار متعدی یک گراف جهت دار، گرافی ست با همان مجموعه رئوس که در آن از راس v به u یال جهت دار وجود دارد اگر و تنها اگر در گراف اصلی مسیری جهت دار از v به u موجود باشد.
در اینجا فقط می‌خواهیم برای هر راسِ v از گراف اصلی، تعداد راس هایی که v به آن ها مسیر دارد به دست آوریم. بنابراین می‌توان نوشت:
ورودی: گراف جهت دار G
خروجی: برای هر راسِ v از G تعداد راس هایی که v به آن ها مسیری جهتدار دارد خروجی دهید.
حل این مسأله کاربردهایی در حوزه پایگاه داده [۵] دارد.

این مسأله را در زمان $O(mn)$ (با BFS زدن روی هر راس) می‌توانیم حل کنیم.
به کمک ضرب ماتریس ها نیز می‌توان در $O(n^3)$ مسأله را حل کرد و از آنجایی که الگوریتم های بهتری نیز برای ضرب ماتریس وجود دارد می‌توان زمان اجرا را تا $O(n^{2.373})$ کاهش داد.
اما در اینجا می‌خواهیم با روش نمونه‌گیری مسأله را در زمان تقریبا $O(m)$ (با مثلا ضریبی لگاریتمی مربوط به ϵ و δ که خطایمان را مشخص می‌کنند.) حل کنیم.

برای ارائه دادن چنین الگوریتمی، ابتدا به این سوال پاسخ می‌دهیم که چگونه می‌توان تعداد راس هایی که از v به آنها مسیر جهتدار وجود دارد تخمین زد؟
یک ایده این است که نمونه تصادفی از راس ها بگیریم و محاسبه کنیم که v به چه کسری از آنها مسیر دارد.
این ایده عیناً همان ایده ای ست که برای مسأله نظرسنجی استفاده کردیم. (در اینجا نظر یک راس مثبت است اگر مسیر جهت داری از v به آن وجود داشته باشد و در غیر اینصورت نظر راس منفی ست!)
البته اگر دقیقا همان ایده حل مسأله نظرسنجی را پیاده کنیم، می‌توانیم برای یک راسِ خاصِ v کسری از رئوس گراف که v به آن ها مسیر جهتدار دارد به دست آوریم. بنابراین تخمین مان از تعداد رئوس مطلوب، n ضربدر مقدار کسری اخیر خواهد بود.
درواقع اگر همان روش را پیاده کنیم، اینجا نیز به قدری نمونه می‌گیریم که $\mu_{\epsilon\delta}$ موفقیت مشاهده کنیم و این مقدار تقسیم بر تعداد نمونه های گرفته شده، تخمینی از کسرِ تعداد رئوسی می‌دهد که v به آنها مسیر دارد. $\left(\frac{\mu_{\epsilon\delta}}{k}\right)$ بنابراین تخمین مان از تعداد رئوسی که v به آنها مسیر دارد $\left(n \times \frac{\mu_{\epsilon\delta}}{k}\right)$ است.
اما با کمی دقت مشخص می‌شود که این رویکرد زمان اجرای خوبی ندارد چون برای اینکه برای هر نمونه بتوانیم بگوییم از v به آن مسیر جهتدار وجود دارد یا نه، می‌بایست یک BFS از راس v اجرا کنیم. (برخلاف این پیاده سازی، در مسأله نظرسنجی در $O(1)$ می‌توانستیم بفهمیم نظر یک فرد مثبت یا منفی است.)
درواقع اگر k بار نمونه‌گیری و هر بار یک BFS انجام شود، زمان اجرا $O(mk)$ یا $O\left(m \times \frac{\mu_{\epsilon\delta}}{p}\right)$ خواهد بود و با توجه به اینکه کوچکترین مقدار ممکن برای p ، $\frac{1}{n}$ است؛ زمان اجرا $O(mn\mu_{\epsilon\delta})$ خواهد بود.
به علاوه توجه داریم که این زمان اجرا تنها برای یک راس بود. اما می‌توان طوری الگوریتم را تغییر داد که برای همه راس ها به طور همزمان در همین زمان الگوریتم انجام شود.
تغییر به این صورت است که در هر بار نمونه‌گیری به جای آنکه فقط ببینیم از راس v به راس نمونه مسیر وجود دارد یا خیر، ببینیم از همه رئوس گراف به راس نمونه مسیر جهتدار وجود دارد یا خیر. این کار نیز با تنها یکبار BFS قابل انجام است. کافی ست جهت همه یال ها را معکوس در نظر بگیریم و رئوسی که از راس نمونه به آنها مسیر جهتدار وجود دارد شناسایی کنیم.
عمل نمونه‌گیری را نیز تا جایی تکرار می‌کنیم که تعداد موفقیت هر یک از رئوس به $\mu_{\epsilon\delta}$ برسد.
بنابراین با همان زمان اجرای قبلی که $O(mn\mu_{\epsilon\delta})$ بود، می‌توانیم تخمین را ارائه دهیم.
یکسری ملاحظات نیز در اینجا وجود دارد. برای مثال می‌خواهیم δ ای که در اینجا درنظر گرفته شده، به این معنا باشد که احتمال خطای

[۴] Transitive Closure
[۵] database

عدد تخمین زده شده برای حتی یکی از رئوس گراف، حداکثر δ باشد. برای رسیدن به این هدف، یک $\delta' = \frac{\delta}{n}$ تعریف می‌کنیم. و در الگوریتم از آن استفاده می‌کنیم. یعنی نمونه‌گیری را تا جایی تکرار می‌کنیم که تعداد موفقیت هر یک از رئوس به $\mu_{\epsilon\delta'}$ برسد.
به این ترتیب می‌توان نتیجه گرفت زمان اجرا به احتمال بالا در همان حدود $O(mn\mu_{\epsilon\delta})$ می‌ماند. (ضرب در احتمالا یک فاکتور لگاریتمی) با وجود تمام این کار ها زمان اجرا کماکان از حالت اولیه (BFS روی همه رئوس) بهتر نمی‌شود.

برای بهتر کردن الگوریتم سعی می‌کنیم نمونه‌گیری را کاراتر انجام دهیم؛ فرض کنید هر راس یک شمارشگر داشته باشد که تعداد موفقیت های آن راس را نشان دهد. هر بار که BFS اجرا می‌شود، رئوسی از گراف که در BFS به آن ها بر می‌خوریم، تعداد موفقیت هایی که دیده اند، یک واحد زیاد می‌شود. برای آنکه عملیات را کارا تر کنیم، اگر به راسی رسیدیم که شمارنده آن به مقدار $\mu_{\epsilon\delta}$ رسیده بود، ادامه BFS از آن راس را متوقف می‌کنیم.
با توجه به این نکته که شمارنده ی هر راس از شمارنده همه راس هایی که به آن ها مسیر جهتدار دارد کمتر است، اگر به راسی رسیده باشیم که شمارنده آن به مقدار $\mu_{\epsilon\delta}$ رسیده است، همه رئوسی که بهشان مسیر جهتدار دارد نیز شمارنده شان به مقدار $\mu_{\epsilon\delta}$ رسیده است و لزومی ندارد BFS را روی آنها انجام دهیم تا شمارشگرشان را یک واحد زیاد کنیم.
زمان اجرای الگوریتم با توجه به اینکه هر یال حداکثر $\mu_{\epsilon\delta}$ بار توسط BFS دیده می‌شود، $O(m\mu_{\epsilon\delta})$ خواهد بود. البته اگر دقیق تر نگاه کنیم، زمان اجرا $O(m\mu_{\epsilon\delta} + n\mu_{\epsilon\delta})$ است. چون ممکن است راس هایی را نمونه‌گیری کنیم که منجر به دیده شدن یالی توسط BFS نشوند.
نکته الگوریتم اخیر (الگوریتمِ EDITH COHEN) این است که در آن از ایده‌ی پیچیده‌ای استفاده نکردیم و صرفا با نمونه‌گیری مقتصدانه، به زمان اجرای مناسب دست پیدا کردیم.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۲۸: شمارش و نمونه‌گیری تصادفی

نگارنده: مریم محمدی یکتا

در جلسه‌ی قبل چند مثال از نمونه‌گیری تصادفی دیدیم. در مسأله‌ی نظرسنجی دیدیم با روش نمونه‌گیری تصادفی همراه با جای‌گذاری، اگر بخواهیم با تقریب $(1 \pm \epsilon)$ و با خطای حداکثر δ تعداد توپ‌های آبی را تخمین بزنیم، لازم است حداقل $\frac{1}{p} \times \frac{3}{\epsilon^2} ln\frac{1}{\delta}$ بار نمونه‌گیری کنیم که این متغیر را $\mu_{\epsilon\delta}$ می‌نامیم. برای رویدادهای نادر (رویدادهایی که برای آن‌ها، p خیلی کوچک باشد)، مقدار $\mu_{\epsilon\delta}$ خیلی بزرگ می‌شود و لازم است این رویدادها را به نحوی مدیریت کنیم.

مسأله‌ی شمارش مقداردهی‌های ارضا کننده [1]

یک فرمول منطقی در حالت نرمال فصلی [2] است اگر به صورت فصل تعدادی پرانتز باشد که هر پرانتز شامل عطف تعدادی متغیر [3] است.
به عنوان مثال $(x_1 \wedge \bar{x_2}) \vee (x_3 \wedge x_1 \wedge \bar{x_1}) \vee (x_4 \wedge \bar{x_4})$ یک فرمول نرمال فصلی‌ست.
ورودی، فرمول Φ درحالت نرمال فصلی‌ست و می‌خواهیم تعداد مقداردهی‌های ارضا کننده‌ی آن را خروجی بدهیم.
مسأله شمارش مقداردهی‌های ارضاکننده در حالت کلی #p-complete است (#p-complete نسخه‌ی شمارشی مسائل np-complete است) .
برای حل مسأله‌ی اصلی، کافی‌ست احتمال این‌که یک مقداردهی تصادفی ارضاکننده باشد را بیابیم، اما در این‌جا سعی می‌کنیم تقریبی از این احتمال را با استفاده از نمونه‌گیری بیابیم. پس تعدادی مقداردهی تصادفی انتخاب می‌کنیم و می‌بینیم چه کسری از آن‌ها ارضاپذیر هستند.
در ابتدا بررسی می‌کنیم که آیا Φ ارضاپزیر هست یا خیر. اگر ارضاپذیر نبود، تکلیف مشخص است. در غیر این‌صورت حداقل یکی از 2^n مقداردهی مختلف وجود دارد که Φ را ارضا می‌کند پس داریم $p \geq \frac{1}{2^n}$ و بنابراین تعداد نمونه‌های لازم ($\mu_{\epsilon\delta}$) ممکن است نمایی باشد.

فرض کنید A_i مجموعه‌ی همه‌ی مقداردهی‌هایی باشد که پرانتز i م را ارضا می‌کند. اگر m پرانتز داشته باشیم، می‌خواهیم مقدار $|\bigcup_{i=1}^{m} A_i|$ را تقریب بزنیم. فرض کنید پرانتز iم ارضاپذیر باشد و شامل s_i متغیر باشد. برای هر مقداردهی‌ای که این پرانتز را ارضا می‌کند، مقدار این s_i متغیر ثابت است اما مقادیر بقیه‌ی متغیرها اهمیتی ندارد بنابراین $|A_i| = 2^{n-s_i}$. مجموعه‌ی X را اجتماع مجزای A_i ها تعریف می‌کنیم، یعنی X اجتماع A_i هاست اما با حساب کردن تکرر اعضا، مثلا اگر a در سه تا از A_i ها آمده باشد، a با سه برچسب مختلف در X ظاهر می‌شود. بنابراین داریم $|X| = \sum_{i=1}^{m} |A_i|$. یک جدول در نظر بگیرید که سطرهای آن نشان‌دهنده‌ی مقداردهی‌های ارضاکننده‌ی فرمول Φ باشد و هر ستون آن متناظر یک پرانتز باشد. در محل تقاطع سطر i و ستون j یک بگذارید اگر مقداردهی i برای پرانتز j ارضاکننده باشد. در هر سطر، دور اولین درایه‌ی یکی که در این سطر ظاهر می‌شود را دایره بکشید.

[1] DNF-counting
[2] DNF
[3] literal

۱	۱	۰	۱	۰	۱
۰	۰	۱	۱	۱	۰
			. . .		
۱	۰	۰	۱	۰	۰

در این صورت، تعداد سطرهای این جدول برابرست با m ، تعداد ستون‌های این جدول برابرست با $|\bigcup_{i=1}^{m} A_i|$ که همان جواب اصلی مسئله است، و از طرفی برابرست با تعداد یک‌هایی که دور آن‌ها خط کشیده‌ایم و جمع تعداد یک‌های جدول برابرست با $|X|$.

به این شکل عمل می‌کنیم:

یکی از یک‌های جدول را به صورت تصادفی و یکنواخت انتخاب می‌کنیم و بررسی می‌کنیم آیا دور آن دایره کشیده‌شده است یا خیر. به دنبال الگوریتمی هستیم که به ما اجازه دهد به صورت یکنواخت و تصادفی یکی از یک‌ها را انتخاب کنیم.

یک ستون را با احتمال متناسب با تعداد یک‌های آن انتخاب می‌کنیم. می‌دانیم تعداد یک‌های ستون i برابرست با $|A_i|$ (بنابراین در زمان چندجمله‌ای می‌توان این گام را انجام داد). سپس به صورت یکنواخت و تصادفی یکی از یک‌های این ستون را انتخاب می‌کنیم (این مرحله معادل انتخاب کردن یکی از اعضای A_i به صورت تصادفی‌ست که در زمان چندجمله‌ای قابل انجام است) و بررسی می‌کنیم آیا دور این درایه‌ی یک دایره کشیده شده است یا خیر. یعنی اگر درایه‌ی یکی که انتخاب کرده باشیم در در سطر i و ستون j باشد می‌خواهیم بررسی کنیم آیا $k < j$ وجود دارد به‌طوری که مقداردهی i م پرانتز k م را ارضا کند یا خیر. که این مرحله نیز در زمان چندجمله‌ای قابل انجام است.

پس زمان هر بار نمونه‌گیری چندجمله‌ای است. حال بررسی می‌کنیم در کل چند بار نمونه‌گیری نیاز داریم. نسبت تعداد یک‌هایی که دور آن‌ها دایره است به کل یک‌ها در جدول حداقل $\frac{1}{m}$ است زیرا در هر سطر حداکثر m تا یک وجود دارد و تنها یکی از آن‌ها دایره‌دار است پس تعداد نمونه‌های لازم حداکثر $m\mu_{\epsilon\delta}$ است که چندجمله‌ای است.

پس توانستیم تعداد مقداردهی‌های ارضاکننده را تقریب بزنیم. حال می‌خواهیم یکی از این مقداردهی‌های ارضاکننده را بیابیم.

تولید مقداردهی ارضاکننده

آن قدر با الگوریتم قبلی، یک تولید می‌کنیم تا به یک درایه‌ی یکِ دایره‌دار برسیم و مقداردهی متناظر آن را خروجی می‌دهیم. چون احتمال انتخاب شدن یک‌ها در الگوریتم قبل با هم یکسان است پس هر یکِ دایره‌دار با احتمال یکسان انتخاب می‌شود. دیدیم احتمال این‌که یک درایه‌ی یک دایره‌دار باشد، حداقل $\frac{1}{m}$ است. پس به طور متوسط حداکثر باید m بار نمونه‌گیری کنیم تا یک درایه‌ی یک دایره‌دار تولید شود.

الگوریتم دیگری برای این‌کار ارائه می‌دهیم:

یک درایه‌ی یک تصادفی در نظر بگیرید. فرض کنید c تا یک در سطر مربوط به این درایه وجود داشته باشد. این درایه را با احتمال $\frac{1}{c}$ نگه می‌داریم و مقداردهی متناظر با آن را خروجی می‌دهیم.

فرض کنید می‌خواهیم یک نمونه‌ی تصادفی یک‌نواخت از مجموعه‌ی A تولید کنیم و الگوریتمی داریم که x را با احتمال $P_x > 0$ تولید می‌کند. می‌خواهیم از روی این الگوریتم، الگوریتمی ارائه دهیم که هر عضو A را با احتمال یکسان تولید کند. مانند بالا عمل می‌کنیم. عنصر x را با الگوریتم بالا تولید می‌کنیم و با احتمال $\frac{\epsilon}{P_x}$ نگه می‌داریم و خروجی می‌دهیم که ϵ عددی‌ست که برابر کمینه‌ی P_x باشد. پس یک عنصر را با احتمال P_x تولید می‌کنیم و با احتمال $\frac{\epsilon}{x}$ نگه می‌داریم پس با احتمال $P_x \times \frac{\epsilon}{P_x} = \epsilon$ یک عنصر را خروجی می‌دهیم. پس اگر بخواهیم احتمال موفقیت در تولید کردن یک نمونه را در نظر بگیریم داریم:

$$Pr[success] = \sum_{x \in A} P_x \times \frac{\epsilon}{P_x} = \epsilon |A| = \{_x min \quad P_x\}|A|$$

مثلا اگر برای هر $x \in A$ داشته باشیم $P_x = \frac{1}{|A|}$ در این صورت احتمال موفقیت یک است، اما اگر توزیع اولیه یکنواخت نباشد، احتمال موفقیت پایین می‌آید.

ارائه‌ی الگوریتم تولید از روی الگوریتم شمارش

می‌خواهیم با استفاده از یک الگوریتم شمارش DNF دلخواه، یک الگوریتم تولید DNF دلخواه ارائه بدهیم.
تعداد مقداردهی‌های ارضاکننده‌ای که در آن‌ها متغیر x_1 برابر با true باشد را با استفاده از الگوریتم شمارش میشماریم، به این صورت که در فرمول اصلی مقدار x_1 را جایگذاری می‌کنیم و تعداد مقداردهی‌های ارضا کننده برای این فرمول جدید را با استفاده از الگوریتم شمارشی که در اختیار داریم، محاسبه می‌کنیم. فرض کنید t مقداردهی ارضا کننده در این حالت وجود داشته باشد. به طریق مشابه تعداد مقداردهی‌های ارضاکننده‌ای که در آن‌ها x_1 برابر با false باشد را f می‌نامیم.
سپس مقدار متغیر x_1 را با احتمال $\frac{t}{t+f}$ برابر با true و با احتمال $\frac{f}{t+f}$ برابر با false قرار می‌دهیم و به صورت بازگشتی یک مقداردهی ارضاکننده‌ی تصادفی برای فرمول باقیمانده تولید می‌کنیم.
در این الگوریتم به ازای هر متغیر در فرمول، دو بار از الگوریتم شمارش استفاده می‌کنیم پس در کل $2n$ بار از این الگوریتم استفاده می‌کنیم و چون الگوریتم شمارش، در زمان چندجمله‌ای کار می‌کند پس کل الگوریتم تولید نیز چندجمله‌ای است.
دیدیم که الگوریتم شمارشی که در زمان چندجمله‌ای کار کند نداریم، اما الگوریتم شمارشی تقریبی‌ای که در زمان چندجمله‌ای کار کند، وجود دارد. با استفاده از این الگوریتم شمارشی تقریبی، به یک تولیدکننده‌ی تقریبا یکنواخت می‌رسیم:
مقداردهی $s_1, ..., s_n$ را برای متغیرهای $x_1, ..., x_n$ در نظر بگیرید. (یعنی $x_i = s_i$). احتمال این‌که با استفاده از یک الگوریتم شمارشی به مقداردهی $s_1, ..., s_n$ برسیم برابرست با:

$$Pr[s_1, ..., s_n] = Pr[x_1 = s_1] \times Pr[x_2 = s_2 | x_1 = s_1] \times Pr[x_3 = s_3 | x_1 = s_1, x_2 = s_2] \times ...$$

اگر t, f را با تقریب $(1 \pm \epsilon)$ و با خطای δ محاسبه می‌کنیم، مقادیر $\frac{t}{t+f}$ و $\frac{f}{t+f}$ نیز با تقریب $(1 \pm \epsilon)$ و با خطای δ محاسبه می‌شوند. پس احتمال تولید یک مقداردهی دل‌خواه، با احتمال حداقل $1 - n\delta$، حداکثر $(1 + \epsilon)^n$ با احتمال درست فاصله دارد، زیرا احتمال این‌که یکی از عبارت‌های $Pr[x_i = s_i | x_1 = s_1, ..., x_{i-1} = s_{i-1}]$ با مقدار واقعی $(1\pm)$ فاصله داشته باشد، حداکثر δ است پس احتمال این‌که یکی از این n عبارت با مقدار واقعی فاصله داشته باشند حداکثر $n\delta$ است، پس احتمال تولید یک مقداردهی، با احتمال $1 - n\delta$، حداکثر $(1 \pm \epsilon)^n$ با احتمال درست فاصله دارد.
پس اگر بخواهیم یک الگوریتم تولید مقداردهی با تقریب $(1 + \epsilon')$ و با خطای حداکثر δ' عمل کند، خوب است الگوریتم تقریبی شمارش ما با تقریب $(1 + \frac{\epsilon'}{n})$ و با خطای حداکثر $\frac{\delta'}{n}$ عمل کند.

تولید الگوریتم شمارش از روی الگوریتم تولید

می‌توانیم از الگوریتم تولید استفاده کنیم و تعدادی نمونه‌ی تصادفی تولید کنیم و کسری از مقداردهی‌ها که در آن‌ها داشته مقدار x_1 برابر true باشد را به دست آوریم، سپس فرض کنیم مقدار x_1، true باشد و این‌کار را به صورت بازگشتی برای بقیه‌ی متغیرها انجام دهیم.
می‌دانیم تعداد مقداردهی‌هایی که در آن‌ها مقدار متغیر x_1 برابر true باشد برابر است با تعداد کل مقداردهی‌ها ضرب در $Pr[x_1 = true]$.
با روش کلاسیک نمونه‌گیری می‌توانیم مقدار $Pr[x_1 = true]$ را تخمین بزنیم، به این صورت که نمونه‌گیری را تا جایی ادامه دهیم که به $\mu_{\epsilon\delta}$ تا نمونه برسیم که در آن‌ها مقدار x_1، true باشد و از روی آن مقدار $Pr[x_1 = true]$ تخمین زده می‌شود. بعد از تخمین زدن، مقدار x_1 را true قرار می‌دهیم و تعداد مقداردهی‌های ارضا کننده که در آن‌ها مقدار x_1، true باشد را به صورت بازگشتی تخمین می‌زنیم و در نهایت تخمینی از تعداد کل مقداردهی‌های ارضا کننده به دست می‌آید.
مشکلی که این راه ممکن است داشته باشد این است که اگر مقدار $Pr[x_1 = true]$ کوچک باشد، باید نمونه‌گیری‌های زیادی انجام دهیم تا

تخمین خوبی با روش کلاسیک نمونه‌گیری به دست آوریم که باعث بد شدن زمان اجرا می‌شود. اما اگر مقدار $Pr[x_1 = true]$ کم باشد، مقدار $Pr[x_1 = false] = 1 - Pr[x_1 = true]$ زیاد است. پس می‌توانیم نمونه‌گیری را تا جایی ادامه دهیم که یکی از پیشامدهای $x_1 = true$ یا $x_1 = false$ ، به حداقل $\mu_{\epsilon\delta}$ موفقیت برسد، در نتیجه فرایند نمونه‌گیری در هر مرحله حداکثر $2\mu_{\epsilon\delta}$ بار تکرار می‌شود. مانند تحلیلی که برای حالت قبل داشتیم، در هر مرحله ضریب خطای $(1 + \epsilon)$ به ما تحمیل می‌شود و مانند قبل اگر بخواهیم یک الگوریتم شمارش با تقریب $(1 + \epsilon')$ و با خطای حداکثر δ' داشته باشیم، خوب است الگوریتم تولید مقداردهی با تقریب $(1 + \frac{\epsilon'}{n})$ و با خطای حداکثر $\frac{\delta'}{n}$ عمل کند.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

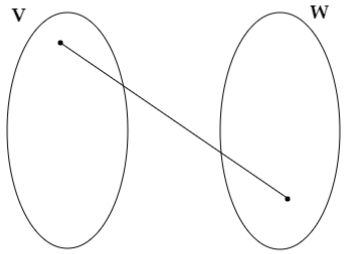
[بهار ۹۹]

جلسه ۱: شکستن تقارن، جواب‌های یکتا، لم ایزوله‌سازی

نگارنده: فاطمه توحیدیان

در این جلسه درمورد یکی از روش‌های مهم شکستن تقارن و به طور خاص کاربرد لم ایزوله‌سازی[1] را می‌بینیم. با مسئله‌ی آشنای تطابق کامل در گراف دوبخشی [2] این بخش را آغاز می‌کنیم:

۱ تطابق کامل در گراف دوبخشی



دیده بودیم که می‌توانیم این مسئله را به کمک یه روش جبری حل کنیم. فرض کنید گراف دوبخشی‌ای داریم که $n \times n$ است و از دو بخش V و W تشکیل شده است. ماتریس M را صورت زیر تعریف می‌کردیم:

$$M_{ij} = \begin{cases} x_{ij} & e_{ij} \in E \\ 0 & \text{در غیر این صورت} \end{cases} \quad (۱)$$

یعن اگر راس i از V به راس j از W وصل بود، آنگاه درایه (i,j) ماتریس M را مساوی x_{ij} می‌گذاشتیم. دترمینان این ماتریس یک چندجمله‌ای می‌شد:

$$det(M) = \sum_{\Pi} (-1)^{sign(\Pi)} \prod_{i=1}^{n} M_{i,\Pi(i)}$$

درصورتی گراف حداقل یک تطابق کامل داشت که این چندجمله‌ای متحد با صفر نباشد. برای بررسی کردن آن نیز یک آزمون تصادفی ساده طراحی کردیم. به این صورت که از دامنه‌ی خوب به اندازه‌ی کافی بزرگی مقدار x_{ij} ها را به صورت تصادفی انتخاب می‌کردیم و برای

[1] Isolating Lemma

[2] Perfect Matching in Bipartite Graph

این مقدار تصادفی بررسی می‌کردیم که آیا چند جمله‌ای ما صفر می‌شود یا نه. سپس به کمک خاصیت صلبیت[1] چندجمله‌ای‌ها که توسط لم شوارتز_زیپل[2] دیده می‌شد، دیدیم که با احتمال خوبی می‌توانستیم بفهمیم که چندجمله‌ای ما متحد با صفر است یا نه.

مسئله‌ی بالا حالت تصمیم‌گیری داشت و مزیتی که نسبت به الگوریتم‌های دیگر دارد این است که قابل موازی‌سازی است، زیرا برای پیدا کردن دترمینان ماتریس الگوریتم موازی وجود دارد. به طور خاص در کلاس **RNC** است که **NC** کلاس مسائلی است که به صورت کارا قابل موازی‌سازی‌اند، یعنی به کمک تعداد چند جمله‌ای پردازنده در زمان چندلگاریتمی[3] حل می‌شوند. همچنین اشاره کردیم همین کار را برای گراف‌های کلی‌تر نیز می‌توان انجام داد (تعریف ماتریس تفاوت داشت). همچنین نسخه‌ی جست‌وجوی این مسئله را نیز می‌توانیم به کمک روش خود تحویلی[4] حل کنیم. به این صورت که یک یال را حذف می‌کنیم و می‌بینیم آیا در گراف باقی‌مانده تطابق کامل وجود دارد یا نه. اگر وجود نداشت یعنی به آن یال نیاز نداریم و آن یال را حذف می‌کنیم. اگر وجود داشت یعنی به آن یال احتیاج داریم، و می‌توانیم آن یال را به همراه راس‌های دو سر آن حذف کنیم و ادامه دهیم... . اما استفاده از این روش خود تحویلی برای حل مسئله‌ی جست‌وجو، باعث خراب شدن خاصیت موازی‌سازی ما می‌شود زیرا باید همه‌ی یال‌ها را یکی یکی امتحان کنیم و ادامه دهیم و نمی‌توانیم همه آن‌ها را همزمان چک کنیم. زیرا ممکن است چند تطابق کامل داشته باشیم، و هر یالی را حذف کنیم ممکن است گراف همچنان تطابق کامل داشته باشد. پس این که چند جواب داریم، برای ما مشکل ایجاد می‌کند (در واقع انگار تقارنی بین همه‌ی تطابق‌های کامل داریم) و اگر تنها یک تطابق کامل داشتیم، کار ما آسان‌تر بود.

می‌خواهیم پیدا کردن تطابق کامل را به صورت موازی انجام دهیم. کاری می‌کنیم که تطابق کامل ما یکتا شود.
ایده: سعی می‌کنیم یکی از تطابق‌های کامل را «مشخص» کنیم.
به یال‌ها به گونه‌ای وزن نسبت می‌دهیم که تطابق کامل کمینه یکتا باشد. سپس الگوریتمی ارائه می‌دهیم که تطابق کامل کمینه را پیدا کند.
حال فرض کنیم توانستیم این وزن‌ها را به یال‌ها نسبت دهیم (این کار را تصادفی انجام می‌دهیم و نشان می‌دهیم اگر به صورت خوبی این کار را انجام دهیم، با احتمال خوبی تطابق کامل کمینه یکتا پیدا خواهد شد).
تابع وزن $w : E \to \mathbb{N}$ را در نظر بگیرید.
شبیه ماتریس M ماتریس Q را به صورت زیر تعریف می‌کنیم:

$$Q_{ij} = \begin{cases} 2^{w_{ij}} & e_{ij} \in E \\ 0 & \text{در غیر این صورت} \end{cases} \qquad (۲)$$

داریم:

$$det(Q) = \sum_{\Pi} (-1)^{sign(\Pi)} \times 2^{\sum_{i=1}^{n} w_{i\Pi(i)}}$$

عبارتی که در توان ۲ ظاهر شده است، درواقع وزن تطابق متناظر با آن قطر پراکنده (جایگشتی خاص از درایه‌ها). پس هرکدام از جمله‌های این دترمینان، اگر این جایگشت متناظر با هیچ تطابق کاملی نباشد، صفر می‌شود در غیر این صورت دو به توان وزن تطابق متناظر است.

حال فرض کنید وزن تطابق کمینه W باشد. یکی از جمله‌های $det(Q)$ ، 2^W است. و بقیه جمله‌ها یا صفر هستند و یا به صورت $2^{W'}$ که $W' > W$. پس 2^W بزرگترین توانی از ۲ است که $det(Q)$ را می‌شمارد. در واقع $det(Q) = 2^w(\pm 1 + ...)$ که این یعنی $det(Q)$ برابر است با ضرب 2^W در یک عدد فرد. این یعنی اگر بتوانیم $det(Q)$ را محاسبه کنیم (که می‌توانیم آن را به صورت موازی انجام دهیم)، آنگاه کافی است کم ارزش‌ترین بیتی که مقدار آن ۱ می‌شود را پیدا کنیم که متناظر با وزن تطابق کامل کمینه می‌شود (دقت کنید فرض کردیم که تطابق کامل کمینه یکتا است). پس با این راه می‌توانیم وزن تطابق کامل کمینه را پیدا کنیم. حال می‌توانیم یکی یکی

[1] Rigidity
[2] Schwartz–Zippel lemma
[3] Polylogarithmic
[4] Self Reduction

یال‌ها را به صورت موازی بررسی کنیم و دیگر مشکلی از نظر داشتن چند جواب نداریم.

حال تابع وزن w را چگونه پیدا کنیم؟ یک سوال این است که دامنه‌ای که از آن وزن را انتخاب می‌کنیم چقدر بزرگ است. واضح است که هرچه این دامنه بزرگتر باشد، احتمال وجود دو تطابق با وزن برابر کمتر می‌شود. همچنین هر چه این مجموعه بزرگتر باشد، انتخاب یک عدد تصادفی سخت‌تر می‌شود و زمان بیشتری نیز می‌خواهد.

حال به لم مهم زیر می‌رسیم:

لم ۱. ***(ایزوله‌سازی)*** *فرض کنید خانواده‌ی $\mathscr{A}$ از زیر مجموعه‌های $X = \{x_1, \cdots, x_m\}$ داده شده باشد. فرض کنید به هر x_i یک وزن تصادفی از مجموعه‌ی $\{1, 2, 3, \cdots, 2m\}$ نسبت دهیم. در این صورت با احتمال حداقل $\frac{1}{2}$ مجوعه با وزن کمینه از $\mathscr{A}$ یکتا است.*

اثبات. برای هر عضوی از X مجموعه‌هایی از $\mathscr{A}$ که آن را شامل هستند و آن‌هایی که شامل نیستند را در نظر می‌گیریم و کمینه‌ی هرکدام را جداگانه بررسی می‌کنیم و سپس روی این نتایج بحث می‌کنیم. تعریف می‌کنیم:

$$\forall x \in X : A_x = \{S \in \mathscr{A} \mid x \in S\}, B_x = \{S \in \mathscr{A} \mid x \notin S\}$$

$$Y_x = \{\text{مجموعه‌های کمینه } A_x\}$$

$$N_x = \{\text{مجموعه‌های کمینه } B_x\}$$

$$M_x = \{\text{مجموعه‌های کمینه } \mathscr{A}\}$$

حال سه حالت می‌تواند رخ دهد.

- اگر $x = -\infty$ ، آنگاه $M_x = Y_x$
- اگر $x = +\infty$ ، آنگاه $M_x = N_x$
- اگر وزن x را از $x = -\infty$ به $x = +\infty$ افزایش دهیم، دقیقاً یک نقطه وجود دارد که $M_x = Y_x \cup N_x$

اگر حالت سوم رخ دهد یعنی $M_x = Y_x \cup N_x$ ، x را مبهم می‌نامیم. داریم:

$$Pr[\text{ مبهم } x\text{ }] \leq \frac{1}{2m}$$

زیرا اگر مقدار بقیه متغیرها هرچیزی باشد (یک مقدار ثابت برای بقیه در نظر بگیرید)، حداکثر یکی از $2m$ مقدار ممکن برای x می‌تواند باعث شود که x مبهم شود (دقت کنید اگر آن نقطه در مجموعه‌ی وزن‌های ما نباشد، این احتمال برابر صفر است). البته اگر بخواهیم دقیق‌تر بنویسیم باید احتمال شرطی بنویسیم (احتمال این که x مبهم باشد به شرط این که بقیه متغیر ها مقدار ثابتی داشته باشند) و همه‌ی حالات مختلف را با هم جمع بزنیم و ... . با این روش باز هم به $\frac{1}{2m}$ می‌رسیم. حال با استفاده از قانون کران اجتماع داریم:

$$Pr[\text{ یکی از } x_i\text{ها مبهم }] \leq \frac{1}{2}$$

در نتیجه به احتمال حداقل $\frac{1}{2}$ هیچ کدام از x_i ها مبهم نیست، و اثبات کامل می‌شود. زیرا مبهم نبودن x_i ها به این معنی است که تکلیف هر x_i معلوم است و x_i یا در مجموعه‌ی کمینه می‌آید یا نمی‌آید.

□

دقت کنید که در این لم، اندازه‌ی مجموعه‌ی $\mathscr{A}$ مهم نیست (مثلاً می‌تواند نمایی باشد). البته اگر همه‌ی زیر مجموعه‌ها را بگیریم،

بدیهتاً مجموعه‌ی تهی این مجموعه‌ی یکتا خواهد شد. اما باز هم می‌توانیم جوری آن را تعریف کنیم که اندازه‌ی آن نمایی باشد و مجموعه‌ی کمینه نیز بدیهی نباشد.

حال به مسئله‌ی اصلی برمی‌گردیم. به هر یال وزنی از مجوعه $\{1, 2, \cdots, 2m\}$ نسبت می‌دهیم که m تعداد یال‌ها است، و به احتمال حداقل $\frac{1}{2}$ تطابق کامل کمینه یکتا است. حال با فرض این که این تطابق یکتا است، چگونه یال‌های این تطابق را تشخیص دهیم؟ $Q^{i,j}$ را تعریف می‌کنیم ماتریسی که از حذف سطر i م و ستون j م ماتریس Q حاصل می‌شود.

قضیه ۱. *یال (i, j) در تطابق کامل کمینه یکتا وجود دارد اگر و فقط اگر $\frac{det(Q^{i,j}) \times 2^{w_{ij}}}{2^W}$ فرد باشد (که در آن W وزن تطابق کامل کمینه است). یعنی 2^W بزرگترین توانی از ۲ باشد که صورت کسر را می‌شمارد.*

اثبات. در واقع در صورت کسر می‌خواهیم دترمینان ماتریس Q را که سطر i م و j م آن را به صورت زیر تغییر داده‌ایم محاسبه کنیم:

$$
\begin{array}{c} \\ i \end{array}
\begin{array}{c} \begin{array}{ccccccc} & & & j & & & \end{array} \\
\begin{bmatrix}
 & & & 0 & & & \\
 & & & \vdots & & & \\
 & & & 0 & & & \\
0 & \cdots & 0 & 2^{w_{ij}} & 0 & \cdots & 0 \\
 & & & 0 & & & \\
 & & & \vdots & & & \\
 & & & 0 & & &
\end{bmatrix}
\end{array}
$$

درواقع داریم:

$$det(Q^{i,j}) \times 2^{w_{ij}} = \sum_{\Pi:\Pi(i)=j} sign(\Pi) \times \prod_{k=1}^{n} Q_{k\Pi(k)}$$

اگر یال (i, j) جزو تطابق کمینه باشد، آنگاه یکی از جمله‌های جمع بالا برابر با 2^W می‌شود. بقیه جمله‌ها نیز یا صفرند یا به صورت $2^{W'}$ که $W' > W$ ، درنتیجه اگر تقسیم بر 2^W کنیم، عدد حاصل فرد می‌شود. اگر یال (i, j) جزو تطابق کمینه نباشد نیز باز هم یا تمام جملات صفرند یا به صورت $2^{W'}$ که $W' > W$ ، که به این معنی است که وقتی تقسیم بر 2^W می‌کنیم عدد حاصل حتماً زوج است. □

حال، ما باید m بار دترمینان حساب کنیم. اگر فرض کنیم محاسبه‌ی دترمینان از $O(n^3)$ باشد، می‌توانیم آن را با $O(n^3)$ پردازنده به طور موازی محاسبه کنیم. چون این کار را برای تمام یال‌ها باید انجام دهیم پس ضریب m نیز در آن ضرب می‌شود و درنهایت تعداد پردازنده‌ها از $O(mn^3)$ می‌شود.

بعد از این که این آزمون‌ها را برای یال‌ها انجام دادیم، کاندیدهایی برای یال‌های تطابق نمونه بدست می‌آوریم، و سپس بررسی می‌کنیم که آیا به ما یک تطابق کمینه می‌دهند یا خیر. اگر نمی‌دادند، این مراحل را تکرار می‌کنیم.

نکته‌ی دیگری که وجود دارد این است که هرکدام از درایه‌های این ماتریس می‌تواند تعداد بیت‌های زیادی داشته باشد. در واقع این مقدار می‌تواند از $O(2^m n!)$ باشد و بنابرین تعداد بیت‌های لازم از $O(m + n \log n)$ می‌شود. حال اگر فرض کنیم تعداد یال‌ها بزرگتر از $n \log n$ باشد، می‌توانیم برای تعداد پردازنده‌ها یک ضریب m نیز اضافه کنیم (زیرا ضرب و جمع نیز قابل موازی‌سازی هستند و می‌توانیم با داشتن پردازنده‌هایی به اندازه‌ی تعداد بیت‌های اعداد این اعمال را موازی‌سازی کنیم). به هرحال، تعداد این پردازنده‌ها چند جمله‌ای می‌شود و در تعریف کلاس **RNC** صدق می‌کند.

البته راه دیگری نیز برای این محاسبات وجود دارد (بدون این که تعداد زیادی دترمینان حساب کنیم). ماتریس الحاقی Q را در نظر بگیرید:

$$\mathrm{a}dj(Q) = {}_i \begin{bmatrix} & \vdots^{\,j} & \\ \cdots & (-1)^{i+j} det(Q^{j,i}) & \cdots \\ & \vdots & \end{bmatrix} = Q^{-1} \times det(Q)$$

این یعنی کافی است یک بار Q^{-1} را حساب کنیم و همه $det(Q^{j,i})$ هایی که می‌خواهیم به کمک این ماتریس به دست می‌آیند.

نکاتی در مورد این الگوریتم

همان‌طور که دیدیم، این الگورتیم پیچیدگی $\mathbf{RNC}$ داشت. حال سوال این است که آیا الگورتیم $\mathbf{NC}$ نیز داریم یا نه (تصادفی بودن را حذف کنیم). این یک سوال باز مهم است. البته نتایج مهمی در سال‌های اخیر به دست آمده است. مثلاً ثابت شده است که این مسئله در $\mathbf{QNC}$ است. یعنی می‌تونیم آن را به صورت قطعی و در زمان پلی‌لگاریتمی ($polylog$) و با تعداد $n^{polylog(n)}$ پردازنده حل کنیم (تعداد پردازنده‌ها نمایی نیست ولی چندجمله‌ای نیز نیست).

همچنین در سال‌های اخیر، برای گراف‌های کلی، مسئله‌ی پیدا کردن یک تطابق کامل را به صورت موازی به نسخه‌ی تصمیم‌گیری تطابق کامل کاهش داده‌اند.

مسائل دیگری نیز وجود دارند که می‌توانیم آن‌ها را با استفاده از این روش حل کنیم (که می‌توانید به عنوان تمرین این کار را انجام دهید). برای مثال:

- پیدا کردن تطابق بیشینه[1]
  می‌توانیم با کاهش دادن آن به مسئله‌ی تطابق کامل این کار را انجام دهیم.
- پیدا کردن تطابق با وزن کمینه[2] (در حالتی که اندازه‌ی وزن یال‌ها چندجمله‌ای باشد)
- جریان بیشینه[3] (در حالتی که ظرفیت یال‌ها ۱ باشد)
- تطابق دقیق[4]
  یال‌های گراف قرمز یا آبی هستند. هدف این است که تطابق کاملی با k یال قرمز پیدا کنیم. الگوریتمی که برای پیدا کردن تطابق کامل دیدیم را می‌توانیم تعمیم دهیم تا این مسئله را نیز حل کند و در نتیجه:

$\mathsf{ExactMatching} \in \mathbf{RNC}$

اما برای مسئله‌ی Exact Matching هنوز الگوریتم چند جمله‌ای پیدا نشده است، یعنی:

$\mathsf{ExactMatching} \overset{?}{\in} \mathbf{P}$

[1] Maximum Matching
[2] Minimum Matching
[3] Maximum Flow
[4] Exact Matching

این نکته ازین لحاظ جالب است که: $\mathbf{NC} \subseteq \mathbf{P}$ (البته حدس زده می‌شود که زیرمجموعه اکید است). به همین دلیل وقتی مسئله‌ای عضو $\mathbf{RNC}$ است، همچنان حدسی وجود دارد که باید یک الگوریتم چند جمله‌ای نیز داشته باشد (ولی تا به امروز پیدا نشده است). می‌توانیم به این صورت تعبیر کنیم که تصادفی بودن یک عنصر خیلی مهم برای موازی‌سازی این الگوریتم است.

حال می‌خواهیم در مورد یک کاربرد دیگر لم ایزوله‌سازی صحبت کنیم. زمینه‌ی دیگری که در آن یکتا بودن جواب می‌تواند کمک کننده باشد (یا حداقل جالب به نظر می‌رسد)، مسائل سخت هستند.

۲ مسائل سخت

آیا در مسائل سخت نیز، این که بدانیم یک جواب یکتا وجود دارد می‌تواند کمک‌کننده باشد؟ برای مثال، اگر بدانیم دقیقا یک مقداردهی ارضا پذیر برای یک مسئله‌ی 3-SAT وجود دارد، یا این که ارضا پذیر نیست. درواقع خیلی اوقات اضافه کردن محدودیت‌های بیشتر، ممکن است به حل مسئله کمک کند. برای مثال فرض کنید یک پازل داریم. اگر با گذاشتن هر قطعه، قطعه‌های مجاور بسیار محدود شوند، این محدودیت‌های بیشتر به ما کمک می‌کند که راحت‌تر پازل را حل کنیم. اما دقت کنید ما با یکتا بودن سراسری[۱] سروکار داریم. در مورد مثال پازل، محدودیت ما یک محدودیت محلی[۲] بود. حال نشان می‌دهیم که در این حالت (مسائل سخت) به ما کمک نمی‌کند! یعنی این که می‌توانیم مسائل حالت کلی را به مسائلی که تنها یک جواب دارند کاهش دهیم:

$\mathsf{SAT} \leq_{RP} \mathsf{USAT}$ [۳]

البته کاهش‌هایی که ما دیده بودیم، کاهش چندجمله‌ای بودند، اما اینجا کاهش ما نیاز به تصادف دارد (یعنی خود تحویل ما یک الگوریتم تصادفی چند جمله‌ای است). البته ما این کاهش را درواقع درباره‌ی مسئله‌ی خوشه‌ها انجام می‌دهیم، یعنی نشان می‌دهیم:

$\mathsf{Clique} \leq_{RP} \mathsf{UClique}$ [۴]

- **مسئله‌ی خوشه‌ها**

ورودی: گراف G و عدد k

خروجی: آیا گراف G خوشه‌ای (k راس که همه به هم وصل هستند) با k راس دارد یا نه؟

حال کاهش ما باید چه خاصیتی داشته باشد؟ فرض کنید ورودی و خروجی کاهش ما به صورت زیر باشد:
ورودی: (G, k)
خروجی: (G', k')

فرض کنید این خاصیت‌ها را داریم:

$$(G, k) \notin \mathsf{Clique} \Rightarrow (G', k') \notin \mathsf{Clique}$$

یعنی در صورتی که ورودی امکان پذیر نباشد، خروجی نیز قطعاً امکان پذیر نخواهد بود (طبیعی بود که ما نتیجه‌گیری کنیم که $(G', k') \notin \mathsf{UClique}$ ، اما در اینجا ما فرض قوی‌تری می‌گذاریم، یعنی اگر (G, k) برای مسئله‌ی اصلی ارضا پذیر نباشد، (G', k')

[1] global
[2] local
[3] Unique SAT
[4] Unique Clique

نه تنها جوابی یکتا ندارد، بلکه اصلا هیچ خوشه‌ای با اندازه‌ی k' ندارد). اما درصورتی که (G,k) برای مسئله‌ی اصلی ارضا پذیر باشد، با قطعیت کامل نمی‌توانیم نتیجه گیری کنیم که جوابی یکتا برای (G',k') وجود دارد، اما با احتمال مثبتی می‌توانیم این را بیان کنیم. یعنی:

$$(G,k) \in \mathsf{Clique} \Rightarrow Pr[(G',k') \in \mathsf{UClique}] = \Omega(\frac{1}{n^2})$$

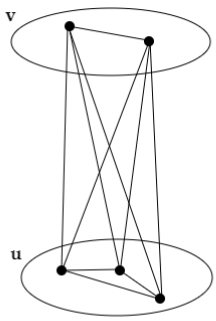
با این که در نگاه اول ناامید کننده به نظر می‌رسد، اما باز هم با تکنیک تکرار می‌توانیم این احتمال را افزایش دهیم. حال اگر یک الگوریتم تصادفی چندجمله‌ای برای مسئله‌ی UClique داشته باشیم، آنگاه نتیجه می‌شود که برای Clique نیز داریم. به این صورت که اگر خوشه‌ی مورد نظر وجود نداشته باشد، با این الگوریتم به جواب قطعی درست می‌رسیم، و اگر وجود داشته باشد نیز با احتمال $\frac{1}{n^2}$ جواب درست می‌گیریم (و حتی می‌توانیم $O(n \log n)$ بار اجرا کنیم تا با احتمال بالا[1] موفق شویم). و این در واقع به این معنی است که:

$$\mathsf{UClique} \in \mathbf{RP} \Rightarrow \mathbf{NP} = \mathbf{RP}$$

که اتفاق عجیب و دور از ذهنی است . همچنین می‌توانید بررسی کنید که اگر به جای حکم $(G',k') \notin \mathsf{Clique}$ در خاصیت اول حکم $(G',k') \notin \mathsf{UClique}$ را داشتیم، آنگاه نمی‌توانستیم نتیجه‌گیری بالا را انجام دهیم.

حال چگونه این کاهش را انجام دهیم؟

- تابع $w : V \to [2n]$ را به صورت تصادفی تعریف می‌کنیم (منظور از $[2n]$ ، $[1, 2, \cdots, 2n]$ است).
- طبق لم ایزوله‌سازی، خوشه‌ی با اندازه‌ی k با وزن بیشینه با احتمال حداقل $\frac{1}{2}$ یکتا است.
- گراف (G',k') را به این صورت تعریف می‌کنیم: برای هر راس v در G ، $2nk + w(v)$ راس به صورت یک خوشه در G' می‌گذاریم.
- برای هر یال $uv \in G$ همه‌ی راس های خوشه‌ی متناظر با u را به همه‌ی راس‌های خوشه‌ی متناظر با v (در گراف G') وصل می‌کنیم.
- عدد $r \in [2nk]$ را به صورت تصادفی انتخاب می‌کنیم.
- قرار می‌دهیم $k' = 2nk^2 + r$.



و تحویل ما پایان می‌یابد. حال خاصیت‌های مطرح شده را بررسی می‌کنیم:

- فرض کنید G خوشه‌ای با اندازه‌ی k ندارد. در این صورت اندازه‌ی بزرگترین خوشه‌ی G' حداکثر $(k-1) \times (2nk + 2n)$ است. حال داریم:

[1] With High Probability

$(k-1) \times (2nk+2n) < 2nk^2 < k'$

- فرض کنید G خوشه‌ای با اندازه‌ی k داشته باشد. به احتمال حداقل $\frac{1}{2}$ یک $k-$خوشه با وزن بیشینه یکتا مثل C در G وجود دارد (طبق لم ۱ . همچنین از اینجا به بعد فرض کنید این خوشه یکتا تعیین شده است). اندازه‌ی خوشه‌ی C در G' برابر است $2nk^2 + w(C)$. حال داریم:

$(2nk^2+1) \leq (2nk+1)k \leq 2nk^2 + w(C) \leq k(2nk+2n) = 2nk^2 + 2nk$

حال با احتمال $\frac{1}{2nk}$ ، G' خوشه‌ی با اندازه‌ی k' دارد (زیرا r عددی تصادفی در بازه $[2nk]$ بود). همچنین دقت کنید که اگر همچین خوشه‌ای موجود باشد، یکتا نیز هست. زیرا:

- هر خوشه‌ی با اندازه‌ی k در G را بگیریم، وزنی کمتر از $w(C)$ دارد، و در نتیجه خوشه‌ی متناظر با آن در G' وزنی کمتر از $k' = 2nk^2 + w(C)$ دارد.
- اگر خوشه‌ای با اندازه‌ی حداکثر $k-1$ در G داشته باشیم، طبق قسمت قبل دیدیم که سایز بزرگترین خوشه با این شرایط حتما کمتر از k' است.
- اگر خوشه‌های با اندازه‌ی حداقل $k+1$ در G را درنظر بگیریم، یک خوشه با سایز حداقل $(2nk+1)(k+1)$ در G' می‌دهد، اما این نتیجه نمی‌دهد که خوشه‌ی مورد نظر ما یکتا است! زیرا می‌توانیم یک زیر مجموعه از خوشه‌ی ایجاد شده در G' بگیریم که اندازه‌ی آن k' باشد. اما باز هم می‌توانیم آن نتیجه‌ای که میخواستیم را به دست بیاوریم. مسئله‌ی Clique را به PClique [1] تبدیل می‌کنیم، که یعنی به ما این اطمینان را می‌دهند که G خوشه‌ای با اندازه‌ی بیشتر از k ندارد. در این حالت به مشکل گفته‌شده برنمی‌خوریم و داریم:

$(G,k) \notin \text{PClique} \Rightarrow (G',k') \notin \text{Clique}$

$(G,k) \in \text{PClique} \Rightarrow Pr[(G',k') \in \text{UClique}] = \Omega(\frac{1}{n^2})$

نشان می‌دهید همین نتایج برای نشان دادن نتیجه‌ی زیر کافی است:

$\text{UClique} \in \mathbf{RP} \Rightarrow \mathbf{NP} = \mathbf{RP}$

زیرا فزض کنید یک ورودی (G,k) برای مسئله‌ی Clique داشته باشیم. حال ورودی (G,n) را برای PClique در نظر می‌گیریم. سپس این مسئله را به UClique کاهش می‌دهیم. به تعداد کافی مسئله را تکرار می‌کنیم (الگوریتم تصادفی چندجمله‌ای را اجرا می‌کنیم)، اگر مطمئن شدیم که G خوشه‌ای با اندازه‌ی n ندارد، آنگاه می‌توانیم n را کاهش دهیم زیرا مسئله‌ی $(G,n-1)$ را برای PClique می‌توانیم تعریف کنیم، و سپس همین مراحل را تکرار می‌کنیم. اگر در مرحله‌ای مسئله‌ی PClique خوشه‌ای با اندازه‌ی m داشته باشد که $k \leq m$ ، یعنی حتما خوشه‌ی با اندازه‌ی k نیز دارد، و الگوریتم پایان می‌یابد، زیرا به این معنی است که گراف G یک خوشه‌ی با اندازه‌ی k دارد.

$(G,k) \xrightarrow{to\ \text{PClique}} (G,n) \xrightarrow{to\ \text{UClique}} (G',k') \xrightarrow{\text{UClique}\ algorithm} \cdots$

$(G,k) \xrightarrow{to\ \text{PClique}} (G,n-1) \xrightarrow{to\ \text{UClique}} (G',k') \xrightarrow{\text{UClique}\ algorithm} \cdots$

$(G,k) \xrightarrow{to\ \text{PClique}} (G,n-2) \xrightarrow{to\ \text{UClique}} (G',k') \xrightarrow{\text{UClique}\ algorithm} \cdots$

$\vdots$

$(G,k) \xrightarrow{to\ \text{PClique}} (G,k) \xrightarrow{to\ \text{UClique}} (G',k') \xrightarrow{\text{UClique}\ algorithm} \cdots$

در نتیجه توانستیم مسئله‌ی Clique را به PClique و درنهایت به UClique کاهش دهیم. برای بقیه مسائل نیز، می‌توانیم به کمک همین کاهش سخت بودن آن‌ها را نشان دهیم. برای مثال می‌توانیم نشان دهیم:

$\text{UClique} \leq_P \text{USAT}$

[1] Promise Clique

می‌دانیم کاهشی به شکل زیر برای دو مسئله‌ی اصلی وجود دارد یعنی:

$$\text{Clique} \leq_P \text{SAT}$$

به کمک ورودی Clique ورودی SAT را به صورت زیر می‌سازیم:

$$(G, k) \to \phi$$

اگر این کاهش این خاصیت را داشته باشد که تعداد خوشه‌های با اندازه k در G برابر با تعداد مقداردهی‌های ارضا کننده ϕ باشد، آن‌گاه همین کاهش برای کاهش مورد نظر ما نیز کار می‌کند (که معمولاً کاهش‌های ما این خاصیت را نیز دارند[1]).

[1] Parsimonious Reductions

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

الگوریتم‌های پیشرفته برای حل مسأله درخت فراگیر کمینه — نگارنده: کیانا عسگری

در این جلسه، قرار است در مورد مسأله درخت فراگیر کمینه[1] صحبت کنیم. با این مسأله در جلسات قبل آشنا شده و تعدادی الگوریتم نیز برای حل آن دیدیم. هدف ما در این جلسه، ارائه یک‌سری الگوریتم برای یافت درخت فراگیر کمینه با زمان اجرای خطی است؛ به خصوص در ادامه‌ی الگوریتم‌های قطعی که معرفی می‌شوند، یک الگوریتم تصادفی زیبا برای حل در زمان خطی ارائه می‌شود.

درخت فراگیر کمینه

۱ یادآوری

در گذشته، الگوریتم‌های کراسکال[2]، پریم[3] و برووکا[4] را برای حل مسأله زیردرخت فراگیر کمینه مشاهده کردیم. اساس کار اکثر الگوریتم‌های مقدماتی درخت فراگیر کمینه، مشابه یگدیگر است. به طور خاص، دو قاعده‌ی زیر وجود دارند:

- **قاعده آبی**: کوچک‌ترین (سبک‌ترین) یال یک برش در یک گراف، حتماً در درخت فراگیرکمینه آن گراف موجود خواهد بود.
- **قاعده قرمز**: سنگین‌ترین یال یک دور گراف، هیچ‌گاه در درخت فراگیر کمینه موجود نخواهد بود.

دیدیم که از روی این دو قاعده می‌توانیم یک الگوریتم تلفیقی ارائه کنیم به این صورت که از روی قاعده آبی، یک یال را آبی کنیم و سپس از بین یال‌هایی که رنگ نشده‌اند یک یال را انتخاب کنیم و رنگ آن را تعیین کنیم تا نهایتاً بتوانیم از روی یال‌های آبی، یک درخت فراگیر کمینه را بیابیم.

سه الگوریتم کراسکال، پریم و برووکا اصولاً بر مبنای قاعده آبی عمل می‌کنند؛ هربار از روی یک برش، یک یالی که حتماً در زیردرخت فراگیر کمینه خواهد آمد را انتخاب می‌کنند و نهایتاً یک درخت ساخته می‌شود.

تمرین: سعی کنید قاعده آبی و قرمز را اثبات کنید.

الگوریتم‌های ما اصولاً بر مبنای مدل مقایسه‌ای عمل می‌کنند. در این جلسه فرض می‌کنیم ورودی‌های ما گراف وزن‌داری هستند که وزن تمام یال‌ها در آن متمایز است. بنابراین درخت فراگیر کمینه یکتا است (می‌توانید این قضیه را هم به عنوان یک تمرین اثبات کنید). پس در کل صحبت‌ما در مورد درخت فراگیر کمینه‌ی یکتا خواهد بود. دقت کنید اگر وزن یال‌ها متمایز نبود، می‌توانیم یک قاعده کانونی برای شکستن تساوی بین وزن یال‌ها بگذاریم؛ به عنوان مثال می‌توانیم در ابتدا یک ترتیب به کل یال‌ها بدهیم و اگر دو یال دارای وزن یکسان

[1] MinimumSpanningTree, MST
[2] Kruskal algorithm
[3] Prim algorithm
[4] Borůvka algorithm

بودند، یالی که در ترتیب بیان‌شده زودتر ظاهر شده را به‌عنوان یال کوچک‌تر در نظر بگیریم. دقت کنید چون الگوریتم‌های ما بر مبنای مدل مقایسه‌ای هستند، مقدار عددی یال‌ها برای ما اهمیتی ندارند و چیزی که اهمیت دارد قاعده‌ی از پیش مشخص‌شده‌ای است که ترتیب یال ها را نسبت به هم تأیین می‌کند. بنابراین طبق روش گفته‌شده می‌توانیم بدون کاسته شدن از کلیت مسأله فرض کنیم هیچ‌گاه تساوی رخ نخواهد داد.

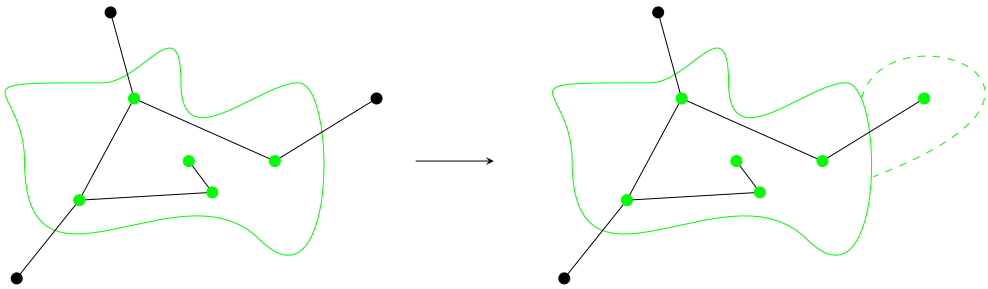
۱.۱ الگوریتم کراسکال

الگوریتم کراسکال، در ابتدا یال‌ها را از کوچک به بزرگ مرتب کرده و سپس یال‌ها را به ترتیب در نظر می‌گیرد، هر کدام را که بشود به جنگلی که تاکنون ساخته شده اضافه کرد به‌طوری‌که دور ایجاد نشود و همچنان جنگل باقی بماند، به آن اضافه می‌کند. زمان اجرای این الگوریتم اگر تعداد رأس‌ها n و تعداد یال‌ها m باشد، $O(m \lg n)$ است. (مساوی با زمان اجرای الگوریتم مرتب‌سازی که در اولین گام استفاده شد.)

هنگام بررسی یک یال، برای اینکه بفهمیم که آیا می‌توانیم آن را به جنگل خود اضافه کنیم، باید ببینیم که آیا دو سر آن یال در دو مؤلفه مختلف از جنگل فعلی ما است یا خیر و اگر دو سر آن در دو مؤلفه مختلف بود پس از انتخاب این یال باید آن دو مؤلفه را تبدیل به یک مؤلفه کنیم. برای این هدف، از ساختمان داده‌ی مجموعه‌های مجزا[5] استفاده می‌کنیم.

۲.۱ الگوریتم پریم

در پیاده‌سازی الگوریتم پریم، از هرم[6] استفاده می‌شود و نحوه کار آن به این صورت است که در هر لحظه از الگوریتم، یک مؤلفه همبندی داریم (درخت) و کم‌وزن ترین یالی را که از این مؤلفه خارج می‌شود پیدا کرده، سر دیگر آن‌را به مجموعه‌ی فعلی خود اضافه می‌کنیم.



بنابراین تمام رأس‌هایی را که در درخت فعلی‌مان ظاهر نشده‌اند در یک هرم که کلید آن کم‌وزن‌ترین یالی است که به درخت فعلی دارند، قرار می‌دهیم. هربار که یک رأس را به درخت فعلی اضافه می‌کنیم، باید همه یال‌های خروجی از آن را بررسی کنیم و در صورت نیاز کلید رأس‌هایی را که سر دیگر این یال‌ها هستند کاهش دهیم. هربار هم که می‌خواهیم یک رأس را بررسی کنیم کافیست عنصر کمینه آن هرم را پیدا کنیم. پس نهایتاً زمان اجرا $O(m \lg n)$ می‌شود؛ زیرا باید m بار برای هر رأسی که به هرم اضافه می‌کنیم سر خارج از درخت فعلی یال مربوطه را به‌روزرسانی کنیم یا به‌عبارتی برای راس‌هایی که به هرم اضافه می‌کنیم به اندازه‌ی درجه‌ی آن رأس باید کلید سر دیگر یال‌هارا به‌روزرسانی کنیم و هر تغییری در کلید مستلزم $O(\lg n)$ هزینه است؛ و چون n بار باید کمینه هرم را در زمان $O(\lg n)$ بیابیم پس کل زمان اجرا $O(m \lg n + n \lg n)$ می‌شود. اما چون معمولاً فرض می‌کنیم که m بزرگتر از n است فاکتور $n \lg n$ را در نظر نمی‌گیریم. البته اگر گراف اولیه ناهمبند باشد یا $m < n$ باشد، استدلال به این صورت درست نخواهد بود و ما روی هر مؤلفه همبندی آن بحث می‌کنیم تا به زیر جنگل فراگیر برسیم. پس می‌توانیم فرض کنیم گراف اولیه‌مان همبند است و ما دنبال زیردرخت هستیم.

[5] union find
[6] heap

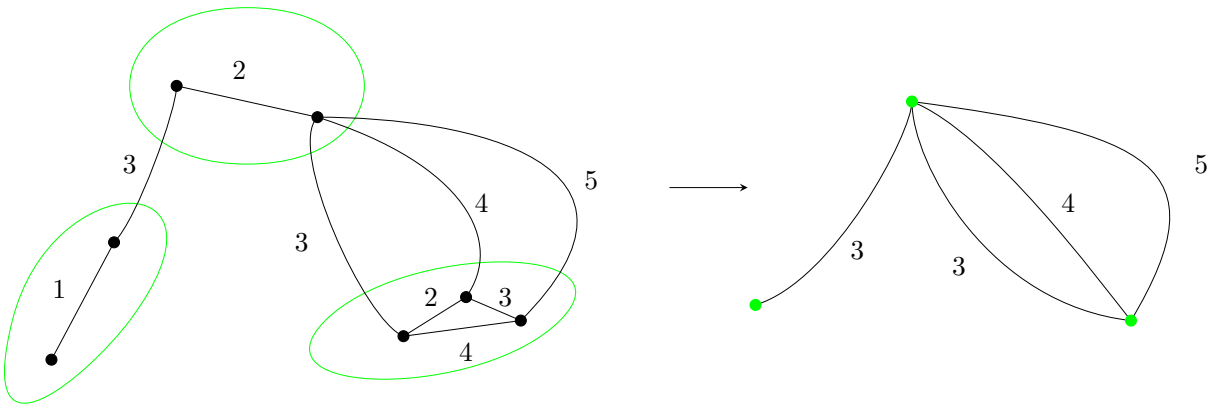
۱.۲.۱ هرم فیبوناچی

الگوریتم پریم را می‌توان به صورت دیگری نیز پیاده سازی کرد: می‌توان بجای استفاده از هرم معمولی، از هرم فیبوناچی[۷] استفاده کرد. این کار به ما اجازه می‌دهد که الگوریتم کاهش کلید[۸] را در زمان سرشکن $O(1)$ انجام دهیم. دیدیم که هر بار به‌روزرسانی کلید یک رأس، به معنای کوچک‌تر کردن کلید آن عنصر است. پس اکنون، عامل $\lg n$ در زمان اجرا «ازبین می‌رود» و زمان اجرا به $O(m + n \lg)$ کاهش می‌یابد.

در هرم فیبوناچی،الگوریتم یافتن کوچک‌ترین کلید[۹] را در زمان سرشکن $O(\lg N)$ انجام می‌شود، که منظور از N بیشترین تعداد عناصر هرم در طول اجرای الگوریتم است.

۳.۱ الگوریتم برووکا

در الگوریتم برووکا، برای هر رأس سبک‌ترین یالی را که به آن وصل است میابیم و نهایتاً یال‌های انتخاب شده، تشکیل تعدادی مؤلفه همبندی می دهند (یا به‌عبارتی یک جنگل می‌سازند) و هرکدام از مؤلفه‌های جنگل بدست آمده را «منقبض» می‌کنیم. بعنوان تمرین می‌توانید اثبات کنید چرا وقتی برای هر رأسی سبک‌ترین یال مجاور آن را انتخاب کنیم، کل یال‌های انتخاب شده تشکیل دور نمی‌دهند. در نهایت هم تمام یال های انتخاب شده تشکیل دور نمی‌دهند. همچنین تمام یال‌های انتخاب شده طبق قاعده آبی باید حتماً در درخت فراگیر کمینه حضور داشته باشند. بنابراین می‌توانیم تمام مؤلفه‌های همبندی تشکیل شده را منقبض کنیم و روی گرافی که بدست می‌آید همین فرآیند را بطور بازگشتی انجام دهیم:



در شکل قبلی یک‌بار اجرا[۱۰] الگوریتم برووکا را مشاهده کردیم. این مرحله در $O(m)$ اجرا می‌شود؛ زیرا یک‌بار باید همه‌ی یال‌ها را پیمایش کنیم و برای هر رأسی کم‌وزن‌ترین یال را انتخاب کرده سپس مؤلفه های همبندی گراف تشکیل شده توسط یال‌های انتخاب شده را پیدا کنیم و هر مؤلفه همبندی آن را تبدیل به یک رأس برای گراف جدید مرحله بعدی الگوریتم کنیم؛ و با پیمایش دوم روی یال‌ها، یال‌هایی که دو سرشان در دو مؤلفه همبندی متمایز هستند، به گراف جدید اضافه کنیم. کل این کارها در $O(m)$ انجام می‌شوند. نکته الگوریتم برووکا این است که تعداد مراحل اجرا تا وقتی که نهایتاً به یک رأس برسیم $O(\lg n)$ خواهد بود؛ زیرا هرکدام از مؤلفه‌های همبندی تشکیل

[7] Fibonacci heap
[8] DECREASE KEY
[9] EXTRACT MIN
[10] Iteration

شده در هر مرحله، حداقل شامل دو رأس است، بنابراین بعد از تبدیل هرمؤلفه همبندی به یک رأس، تعداد رأس‌های گراف حداقل نصف می‌شود و پس بعد از حداکثر $\lg n$ مرحله، الگوریتم به اتمام می رسد.

الگوریتم برووکا، قابلیت‌هایی دارد که باعث می‌شود بعضی جاها بر الگوریتم‌های کروسکال و پریم که الگویتم‌های معروف‌تری برای محاسبه زیردرخت فراگیر کمینه زیردرخت فراگیر کمینه هستند، ارجحیت داشته باشد.

مزیت‌های الگوریتم برووکا

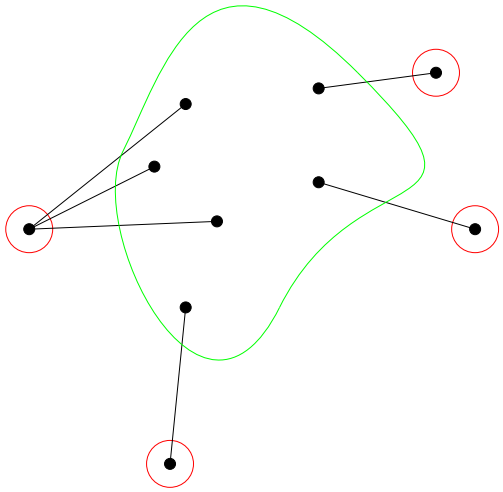
۱. الگوریتم بروفکا قابل موازی‌سازی است.

۲. الگوریتم بروفکا بر خلاف الگوریتم کراسکال و پریم، نیاز به هیچ ساختمان داده خاصی بجز آرایه و احتمالاً لیست مجاورت، ندارد.

۲ الگوریتم تارجان

عمل‌کرد هرم و بطور خاص هرم فیبوناچی در الگوریتم پریم، این سؤال را ایجاد می‌کند که الگوریتم را چگونه اجرا کنیم و در چه حالتی زمان اجرا بهبود می‌یابد.

نحوه‌ای که بطور معمول الگوریتم پریم پیاده‌سازی می‌شود، این است که در ابتدا کلید تمام رأس‌ها بی‌نهایت وکلید رأس اولیه را صفر بگذاریم و سپس تمام رأس‌ها را در هرم قرار داده و الگوریتم را اجرا کنیم. در این‌صورت N تعریف شده در هرم فیبوناچی، همان n می‌شود. اما فرض کنید بجای این کار، تنها رأس ابتدایی با کلید صفر را در هرم قرار دهیم. سپس وقتی این رأس را به‌عنوان عضو کمینه انتخاب کردیم، همسایه‌های آن را در نظر بگیریم و اگر همسایه‌ای تاکنون در هرم نبوده است، آن را اکنون به هرم اضافه کنیم. یعنی رأس‌هایی که با درخت فعلی همسایه نیستند و هنوز به آن‌ها نرسیده‌ایم (به اصطلاح در مرز درخت فعلی نیستند، که منظور از مرز درخت، رأس‌های خارج از آن هستند که به داخل درخت یال دارند.) را اول کار با کلید بی نهایت در هرم قرار نمی‌دهیم. مزیت این کار این است که مرز درخت فعلی در کل مراحل اجرای الگوریتم همواره کوچک باشد؛ بعنوان مثال مرز درخت فعلی همواره $\lg n$ باشد. مثلاً به ازای کران ۴ گراف به شکل زیر خواهد بود:



در این مدل پیاده‌سازی الگوریتم پریم با هرم فیبوناچی، پیچیدگی $O(m + n \lg \lg n)$ خواهد داشت؛ زیرا N برابر با $\lg n$ می‌شود و زمان پیدا کردن عنصر کمینه به $O(\lg \lg n)$ کاهش میابد. پس یک ایده این است که سعی کنیم الگوریتم را جوری پیاده سازی کنیم که

این اتفاق بیفتد. این ایده، ایده‌ی الگوریتم فردمن و تارجان[11] است که زمان اجرای آن $O(m \lg^* n)$ است. این الگوریتم خیلی به هدف ما، یعنی زمان اجرای خطی بر حسب تعداد یال‌ها، نزیک است؛ زیرا رشد تابع $\lg^*$ ،که برابر تعداد بارهایی است که باید از یک عدد لگاریتم گرفته شود تا اینکه به یک برسیم، به شدت کند است و حتی برای اعداد خیلی بزرگ هم از ۵ یا ۶ بیشتر نمی‌شود. اما همچنان از لحاظ تئوری، خطی نیست و ما همچنان دنبال یک الگوریتم با زمان اجرای بهتر هستیم.

۱.۲ نحوه عملکرد

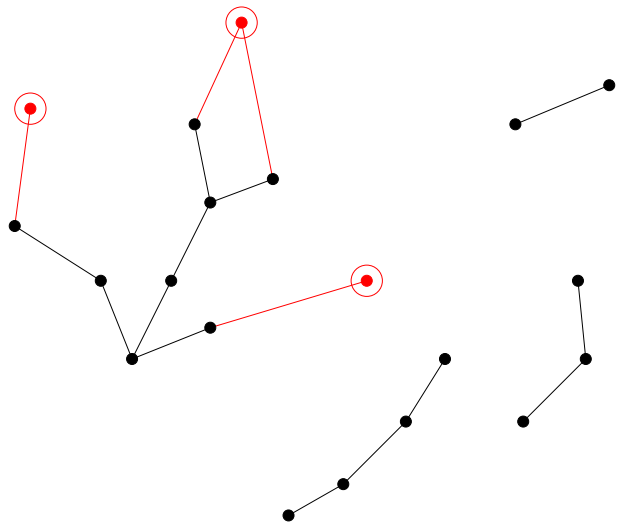
الگوریتم تارجان، از تعدادی دور یا مرحله تشکیل شده است. در هر دور، تعدادی درخت پیدا می‌کنیم و سپس درخت‌هارا منقبض کرده و در ادامه تقریباً مشابه الگوریتم برووکا ادامه می‌دهیم.

۱.۱.۲ هر مرحله از الگوریتم

در ابتدا، همه رأس‌ها را علامت نخورده در نظر می‌گیریم و در طول اجرای هر مرحله آن‌ها را علامت می‌زنیم. یک آستانه k را هم در نظر می‌گیریم که بیشینه اندازه درخت است که حاضریم متحمل شویم (مشابه محدودیتی که روی اندازه مرز درخت در الگوریتم پریم قرار دادیم) و مقدار آن بر حسب شماره مرحله‌ای که در آن هستیم تعیین می‌شود.

یک رأس علامت نخورده مثل u انتخاب می‌کنیم، سپس به شیوه الگوریتم پریم شروع به ساخت درخت T_u از روی u می‌کنیم. اگر به جایی رسیدیم که $|N(T_u)| \geq k$ و یا یک رأس علامت خورده به T_u اضافه شد، ساخت درخت فعلی را متوقف می‌کنیم و همه رأس‌های T_u را علامت می‌زنیم.

یک مرحله تا جایی ادامه می‌یابد که تمام رأس‌های گراف علامت بخورند. بعنوان مثال به ازای $k = 3$ درخت‌های انتخاب شده می‌توانند به شکل زیر می‌شوند:



در نهایت تمام مؤلفه‌های همبندی بدست آمده را منقبض می‌کنیم، و روی گراف باقی‌مانده الگوریتم را ادامه می‌دهیم.

[11] Fredman-Tarjan

۲.۲ اثبات درستی

اثابت درستی الگوریتم، مشابه درستی الگوریتم‌های پریم و برووکا و با استفاده از قاعده‌های آبی و قرمز است.

۳.۲ تحلیل مرتبه زمانی

سوال : در هر دور، حداکثر چند درخت بر حسب تعداد یال‌ها و k پیدا می‌کنیم؟

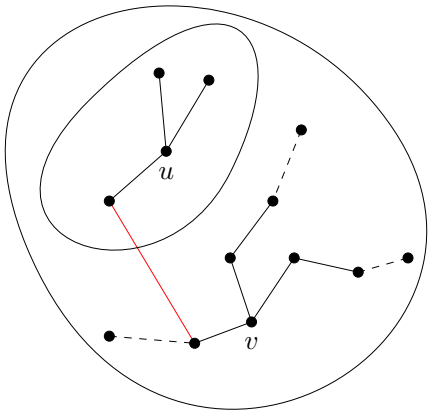
دقت کنید تعداد درخت‌هایی که در یک دور پیدا می‌کنیم، تعداد رأس‌های گراف مرحله بعدی می‌شوند. پس پاسخ این سوال در تحلیل کمک‌کننده است.

لم ۱. *جمع درجات رأس‌های هرکدام از زیردرخت‌ها، حداقل k است.*

اثبات. یکی از زیردرخت‌های تولید شده را در نظر بگیرید. ساخت این زیر درخت به دو دلیل می‌تواند متوقف شده باشد:

۱. مرز درخت حداقل k شده باشد؛ آنگاه هر کدام از رأس های روی مرز درخت، حداقل باعث اضافه شدن یک درجه در جمع درجات درخت می‌شود.

۲. یک رأس علامت‌خورده انتخاب شده باشد (به یک درخت دیگری که قبلاً ساخته بودیم برخورد کرده باشیم)؛ یعنی یالی مانند e که زیر درخت T_u و T_v را به هم متصل می‌کند را انتخاب کرده‌ایم. در این حالت هم حکم مشخصاً برقرار است؛ زیرا بصورت استقرایی فرض می‌کنیم تمام زیر درخت‌هایی که قبل از T_v ساخته‌ایم جمع درجاتشان حداقل k بوده باشد. پس جمع درجات T_u حداقل k است. اکنون این زیر درخت با T_v ترکیب شده که درنتیجه جمع درجات آن‌ها بیشتر می‌شود که حکم را اثبات می‌کند.



□

فرض کنید درخت‌های پیدا شده، $T_1, ..., T_l$ باشند. می‌خواهیم کران بالایی برای l بیابیم. با استفاده از لم داریم:

$$2m = \sum_{v \in V} d(v) = \sum_{i=1}^{l} \sum_{v \in T_i} d(v) \geq lk \Rightarrow l \leq \frac{2m}{k}$$

فرض کنید در مرحله i ام، گرافی که روی آن الگوریتم را اجرا می‌کنیم، n_i تا رأس و حداکثر m یال داشته باشد. آنگاه زمان اجرای الگوریم در مرحله iام، $O(m + n_i \lg k_i)$ است که k_i مقدار k در مرحله i ام است و هنوز تعیین نشده است. زیرا مشابه اجرای الگوریتم پریم با هرم فیبوناچی است که هیچوقت تعداد عناصر هرم از k_i بیشتر نمی‌شود. همچنین در انتها منقبض‌کردن مؤلفه‌ها هم در زمان خطی $O(m + n_i)$ انجام می‌شود. بنابراین کل زمان اجرا $O(m + n_i \lg k_i)$ است.

سوال : بهترین مقدار برای k_i چه می‌تواند باشد؟

می‌خواهیم به k_i مقداری بدهیم که زمان $O(m + n_i \lg k_i)$، تبدیل به زمان خطی بشود. پس اگر قرار دهیم:

$$k_i = 2^{\frac{2m}{n_i}}$$

در این‌صورت، به زمان اجرای $O(m)$ می‌رسیم. پس زمان اجرای الگوریتم در هر مرحله خطی بر حسب تعداد یال است.

سوال : چه کران بالایی برای تعداد مراحل الگوریتم وجود دارد؟

دیدیم که تعداد درخت‌های مرحله i ام حداکثر $\frac{2m}{k_i}$ است. هرکدام از این مؤلفه‌ها تبدیل به یک رأس در مرحله بعدی می‌شوند. پس:

$$n_{i+1} \leq \frac{2m}{k_i} \Rightarrow k_i \leq \frac{2m}{n_{i+1}} = \lg k_{i+1} \Rightarrow k_{i+1} \geq 2^{k_i}$$

$$\Rightarrow k_1 = 1, k_2 \geq 2, k_3 \geq 2^2, k_4 \geq 2^{2^2}, k_5 \geq 2^{2^{2^2}}, ...$$

پس تعداد مراحلی که باید الگوریتم را اجرا کنیم تا مقدار k به n برسد (که مرحله آخر خواهد بود و الگوریتم تبدیل به الگوریتم پریم می‌شود)، حداکثر $\lg^* n$ خواهد بود:

$$k \lg^* n \geq n$$

پس کل زمان اجرای الگوریتم، $O(m \lg^* n)$ خواهد شد.

زمان اجرای الگوریتم تارجان به الگوریتم خطی خیلی نزدیک است؛ اما هم‌چنان سؤالی که از لحاظ نظری وجود دارد این است که آیا الگوریتم قطعی با زمان اجرای خطی برای یافتن زیردرخت فراگیر کمینه وجود دارد یا خیر. بعد از الگوریتم تارجان، الگوریتم قطعی با پیچیدگی $O(m \lg \lg^* n)$ برای درخت فراگیر کمینه یافت شد. همچنین الگوریتم دیگری با زمان اجرای $O(m\alpha(m, n))$ یافت شده است که α معکوس تابع اکرمن[۱۲] است که رشد آن به شدت کند است.

۳ تأیید زیردرخت فراگیر کمینه

این بخش، نسخه تصمیم گیری از مسأله درخت فراگیر کمینه[۱۳] را بررسی می‌کنیم:

ورودی : یک گراف وزن‌دار و یک زیردرخت از آن.

خروجی : آیا زیردرخت داده شده یک زیردرخت فراگیر کمینه از گراف داده شده هست یا خیر.

[12] Ackermann Function: $A(0, n) = n + 1; A(i, 0) = A(i - 1, 1)$ for $i > 0$; and $A(i, n) = A(i1, A(i, n))$for $i, n > 0$.

[13] MSTverification

برای این مسأله، الگوریتم خطی قطعی یافت شده است.

کملوس[۱۴] برای این مسأله، یک الگوریتم با $O(m)$ مقایسه ارائه کرد. اما پیاده‌سازی این مقایسه‌ها زمان اجرا را افزایش می‌دهد. بعد از کملوس،الگوریتمی توسط دیکسون[۱۵] ، راش[۱۶] و تارجان ارائه شد که زمان اجرای آن خطی است. اساس کار این الگوریتم، انجام مقایسه‌های بهینه برای درخت‌های با اندازه خیلی کم (مثلاً یا اندازه حداکثر$(O(\lg^* n)$ با جست‌جوی تمام حالات[۱۷] است و از یک الگوریتم بازگشتی استفاده می‌کند که پایه بازگشت، همان درخت‌های با اندازه کوچک هستند.

پس مسأله تأیید زیردرخت فراگیر کمینه در زمان خطی و بصورت قطعی قابل حل است و ما می‌خواهیم با استفاده از وجود این الگوریتم، یک الگوریتم تصادفی با زمان اجرای $O(m)$ برای حل مسأله زیردرخت فراگیر کمینه ارائه کنیم.

۴ الگوریتم تصادفی

در الگوریتمی که ارائه خواهیم کرد، از مسأله تأیید زیردخت فراگیر کمینه در زمان خطی به‌عنوان زیررویه استفاده می‌شود. درواقع الگوریتم‌های تأیید زیردرخت فراگیر کمینه کار بیشتری از صرفاً تشخیص زیردرخت فراگیر کمینه بودن یک زیردرخت انجام می‌دهند؛ این الگوریتم‌ها یال‌هایی از گراف را که می‌توان از آن‌ها استفاده کرد تا زیردرخت را بهبود داد نیز به‌عنوان خروجی می‌دهند (اگر درخت ورودی خود زیردرخت فراگیر کمینه باشد این مجموعه تهی است وگرنه با اضافه و حذف کردن یک‌سری یال که الگوریتم به ما می‌دهد می‌توان وزن درخت بهبود داد).

ایده این الگوریتم تصادفی، نمونه‌گیری[۱۸] است. با نمونه‌گیری در گذشته آشنا شدیم اما عمده‌ی استفاده ما از آن در تخمین احتمال یا شمارش بوده است. اکنون از نمونه‌گیری برای پیدا کردن جواب واقعی یک مسأله بهینه‌سازی ترکیبیاتی استفاده خواهیم کرد.

سؤالی که ایجاد می‌شود این است که نمونه‌گیری چه کمکی به ما می‌کند و چگونه این نمونه‌گیری را انجام دهیم. طبیعی‌ترین راهی که به ذهن می‌رسد، یک نمونه‌گیری از یال‌های گراف است؛ تعدادی از یال‌های گراف را انتخاب کنیم و سپس در این مجموعه که کوچک‌تر از گراف اصلی است زیردرخت فراگیر کمینه را بیابیم:

۱. یک نمونه F از یال‌های گراف انتخاب کرده و گراف تشکیل شده را $G(p)$ می‌نامیم (هر یال G را با احتمال p انتخاب می‌کنیم).

۲. زیر درخت فراگیر کمینه F را پیدا می‌کنیم.

حال سوال پیش آمده این است که زیردرخت فراگیر کمینه پیدا شده چه اطلاعاتی به ما می‌دهد و چگونه از آن برای یافتن زیردرخت فراگیر کمینه کل گراف استفاده کنیم؟

مشابه مطلبی که در مسأله میانه[۱۹] بیان شد، اکنون هم نمی‌توانیم درمورد ارزش عددی زیردرخت فراگیر کمینه کل گراف از روی ارزش عددی زیردرخت فراگیر کمینه بدست آمده حرفی بزنیم. یعنی زیردرخت فراگیر کمینه بدست آمده از نمونه تصادفی به ما اطلاعاتی درمورد جمع وزن یال‌های زیردرخت فراگیر کمینه کل گراف نمی‌دهد. پس ما فرض می‌کنیم که با ترتیب یال‌ها کار می‌کنیم و مقدار عددی وزن آن‌ها برای ما اهمیتی ندارد.

به چه معنایی می‌توانیم بگوییم زیردرخت فراگیر کمینه F به زیردرخت فراگیر کمینه کل گراف نزدیک است؟

[14] Koml´os algorithm
[15] Dickson
[16] Rush
[17] brute force
[18] Sampling
[19] Median

یک روش برای تحلیل این حرف این است که مطلب گفته شده را به معنای اشتراک زیاد یال‌های دو زیر درخت تفسیر کنیم. اما اکنون معنای زیر را درنظر میگیریم:

زیردرخت فراگیر کمینه‌ی مجموعه F احتمالاً زیردرخت فراگیر کمینه کل گراف نیست پس می شود از یک سری یال‌هایی را که در G هستند اما در F نیستند استفاده کرد تا زیردرخت به‌دست آمده را بهبود داد. حال نکته این است که اگر تعداد این یال‌ها کم باشد، به معنایی زیردرخت یافت شده «خوب» است؛ یعنی اگر تعداد کمی از یال‌های G را حذف کنیم، آن‌وقت زیردرخت فراگیر کمینه کل گراف با زیردرخت فراگیر کمینه‌ی F برابر می‌شود.

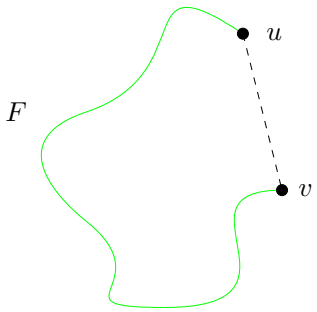
برای دقیق‌تر شدن، تعریف زیر را درنظر بگیرید:

یال $e = uv$ را $F-$**سبک** می‌نامیم اگر در F مسیری از u به v با یال‌های سبک‌تر از e وجود داشته باشد. درغیر این صورت e را $F-$**سبک** می‌نامیم. به‌عنوان مثال در گراف زیر، یال uv در سمت چپ $F-$سبک و در سمت راست $F-$سنگین است.

دقت کنید $F-$سنگین بودن یال e به این معناست که طبق قاعده‌ی قرمز، یال e در زیردرخت فراگیر کمینه گراف $F+e$ ظاهر نمی‌شود:

$$e \notin \mathsf{MST}(F+e)$$

زیرا سنگین ترین یال دوری مانند دور زیر است:



بنابراین یال e نمی‌تواند زیردرخت فراگیر کمینه F را بهبود دهد. پس درواقع:

یال e ، $F-$سنگین است اگر و فقط اگر e، $\mathsf{MST}(F)-$سنگین باشد.

اثبات نکته بالا با استفاده از قاعده‌ی قرمز سرراست است.

بنابراین از این به بعد می‌توانیم درمورد استفاده از الفاظ F یا زیردرخت فراگیر کمینه Fو بین تعریف $F-$سنگین یا $F-$سبک بودن یال، کمی بی‌دقتی به خرج دهیم.

به‌طور مشابه برای سبک بودن یک یال نیز حکم مشابه برقرار است.

پس وقتی نمونه ما F باشد، یال‌های $F-$سبک تنها یال‌هایی هستند که می‌توانند زیردرخت فراگیر کمینه F را بهبود دهند. اکنون می‌خواهیم نشان دهیم تعداد این‌گونه یال‌ها کم است. به معنایی:

لم ۲. *اگر F هر یال G را بطور مستقل و با احتمال P شامل شود، آنگاه:*

$$E[\text{تعداد یال‌های } F-\text{سبک}] \leq \frac{n}{p}$$

یعنی اگر مثلاً هر یال با احتمال $\frac{1}{2}$ انتخاب کنیم، آنگاه میانگین تعداد یال‌هایی که انتخاب می‌شوند نصف تعداد کل یال‌هاست؛ اما تعداد یال‌های $F-$سبک که با زیردرخت فراگیر کمینه F مشکل دارند و می‌توانند آن‌را بهبود دهند، حداکثر $2n$تا است که اگر فرض کنیم تعداد یال‌ها خیلی بیشتر از تعداد رأس‌ها است، این تعداد خیلی کم است. پس زیردرخت فراگیر کمینه بدست آمده برای اکثر یال‌ها «درست» است و یال‌های اندکی با زیردرخت فراگیر کمینه بدست آمده مشکل دارند.

همین‌طور دقت کنید که

$$\mathsf{MST}(G) \subseteq \mathsf{MST}(F) + \text{یال‌های } F-\text{سبک}$$

زیرا اگر یالی $F-$سنگین باشد، طبق قاعده‌ی قرمز در زیردرخت فراگیر کمینه ظاهر نخواهد شد؛ زیرا دوری وجود دارد که سنگین‌ترین یال آن است. پس اگر لم بیان‌شده اثبات شود، می‌توانیم یک نمونه تصادفی گرفته، زیردرخت فراگیر کمینه آن و یال‌های $F-$سبک را پیدا کرده و درنهایت در مجموعه $\{\text{یال‌های } F-\text{سبک} + \mathsf{MST}(F)\}$ که تعداد اعضای آن هم کم است، دنبال زیردرخت فراگیر کمینه گراف اصلی با روش‌هایی مانند روش بازگشتی یا با استفاده از الگوریتم‌هایی که دیدیم بگردیم.

دیدیم که $F-$سبک بودن با $\mathsf{MST}(F)-$سبک بودن به یک معنا هستند و برای ساختن زیردرخت فراگیر کمینه F کاری که انجام می‌دهیم به ترتیب زیر است:

۱. نمونه تصادفی F خود را انتخاب می‌کنیم.

۲. زیردرخت فراگیر کمینه آن‌را می‌سازیم.

اثبات. برای اثبات لم و تحلیل، خوب است که به این فرآیند به چشم دیگری نگاه کنیم. در حقیقت به گونه‌ای دو مرحله را هم‌زمان انجام دهیم. یعنی به‌عنوان مثال الگوریتم کراسکال را شروع به اجرا می‌کنیم بدون این‌که هنوز نمونه‌گیری صورت گرفته باشد و نمونه‌گیری را حین اجرای الگوریتم انجام دهیم.

فرآیند گفته شده به این صورت است: فرض کنید یال های G به ترتیب مرتب شده‌اند. نمونه‌گیری تصادفی به این معنا است که برای هر یالی یک سکه که به احتمال p شیر است بیندازیم و اگر شیر آمد یال را انتخاب کنیم. پس در ترتیب مرتب‌شده یال‌ها، تعدادی از آن‌ها باقی‌مانده و بقیه حذف می‌شوند. سپس روی یال‌های باقی‌مانده الگوریتم کراسکال را اجرا کنیم.

اکنون می‌توانیم به این فرایند به این دید نگاه کنیم که برای هر یال یک سکه انداخته‌ایم اما نتیجه را نمی‌دانیم. پس در ابتدا در الگوریتم کروسکال همه یال‌ها را مرتب می‌کنیم و بعد یکی یکی یال‌هارا بررسی می‌کنیم. برای یال‌هایی که طبق الگوریتم کروسکال باید به زیرجنگل فعلی ما اضافه شوند، در لحظه اضافه شدن نتیجه سکه‌ی انداخته‌شده برای این یال را نگاه می‌کنیم. این معادل همین است که در لحظه انتخاب شدن یک یال، برای آن سکه بیندازیم. اگر نتیجه شیر بود یال را به زیردرخت فراگیر کمینه F اضافه کرده وگرنه آنرا اضافه نمی‌کنیم. پس به خلاصه برای هر یال e :

۱. اگر نباید به جنگل فعلی اضافه شود (مسیری بین دو سر e وجود دارد) از آن عبور می‌کنیم و به نتیجه سکه‌ی آن یال کاری نداریم؛ زیرا هدف ما نهایتاً زیردرخت فراگیر کمینه‌ی نمونه مدنظر است نه خود نمونه و این یال طبق کروسکال نباید در زیردرخت فراگیر کمینه F موجود باشد (درواقع اما اگر بخواهیم خود F را داشته باشیم، باید در این مرحله به نتیجه سکه یال دقت کنیم، ولی هدف ما تنها زیردرخت فراگیر کمینه F است).

۲. وگرنه با احتمال p، یال e را به زیردرخت فراگیر کمینه F اضافه می‌کنیم (و به F، اگر F را می‌خواهیم).

(آ) اگر یال e اضافه نشد، به این معنی است که $F-$سبک است.

حال اگر به فرآیند نمونه‌گیری و ساخت زیردرخت فراگیر کمینه F را به صورت بیان شده نگاه کنیم، تعداد یال‌های $F-$سبک برابر با تعداد یال‌های بخش ۲.(آ) می‌شوند؛ زیرا یال $F-$سبک زمانی ایجاد می‌شود که یک یال e یافته شده که باید به زیردرخت فراگیر کمینه اضافه شود ولی نشده است. حال سوال این است که تعداد این یال‌ها چقدر می‌شود.

تعداد یال‌های زیردرخت فراگیر کمینه F حداکثر $n-1$ می‌شود زیرا در نهایت زیردرخت فراگیر کمینه همبند است. پس بعد از اینکه $n-1$ یال به آن اضافه شد، دیگر یال $F-$سبک نخواهیم داشت. نکته این است که یالی $F-$سبک است اگر در کل مسیری بین دو سر آن در مجموعه یال‌های فعلی نباشد؛ زیرا اگر مسیری باشد حتما مسیر سبک‌تر است (زیرا یال‌ها را به ترتیب وزن بررسی می‌کنیم) که با $F-$سبک بودن در تناقض است. پس در کل گویی آزمایشی داریم که با احتمال p یال را به زیردرخت فراگیر کمینه F اضافه کرده (موفقیت) و با احتمال $p-1$ نمی‌کند (شکست) و در نهایت ما قرار است n موفقیت بدست آوریم. پس میانگین تعداد دفعاتی که باید یک سکه نامتقارن را بندازیم تا n بار شیر بیاید را می‌خواهیم که این مقدار برابر $\frac{n}{p}$ است (توزیع دوجمله‌ای). پس یعنی میانگین کل دفعات اجرای مرحله ۲ برابر $\frac{n}{p}$ است؛ پس میانگین جمع کل یال‌های زیردرخت فراگیر کمینه F و کل یال‌های $F-$سبک، حداکثر $\frac{n}{p}$ است. □

توجه : بعضی اوقات در تحلیل‌هایمان به یال‌های زیردرخت فراگیر کمینه‌ی F هم$F-$سبک می‌گوییم.

در اثبات لم، از اصلی به نام اصل تعویق تصمیم[20] استفاده شد؛ به این معنا که تصمیم برای اینکه چه یال‌هایی را در مجموعه‌مان بیاوریم را قرار است که در وهله‌ی اول انجام دهیم ولی برای تحلیل می‌توانیم فرض کنیم که سکه‌های انداخته‌شده برای نمونه‌گیری را دیر به آنها نگاه می‌کنیم (حین اجرای الگوریتم).

با استفاده از این لم، در همین‌جا به یک الگوریتم تقریبی برای مسأله‌ی زیردرخت فراگیر کمینه رسیده‌ایم؛ به این معنا که یال‌های کمی هستند که با زیردرخت فراگیر کمینه انتخاب شده «مشکل» دارند. برای رسیدن به زیردرخت فراگیر کمینه اصلی گراف، الگوریتمی که داریم بصورت زیر است:

۱. ابتدا $G(P)$، با مجموعه یال‌های F و زیر درخت فراگیر کمینه‌ی $G(P)$ را محاسبه کنید.

۲. یال‌های$F-$سبک را پیدا کنید.

۳. زیردرخت فراگیر کمینه‌ی یال‌های $F-$سبک را بیابید.

همانطور که گفتیم در الگویتم بالا خود زیر درخت فراگیر کمینه‌ی $G(P)$ را هم جزو یال‌های $F-$سبک درنظر می‌گیریم. پس در نهایت MST کل گراف طبق لم، باید در مجموعه‌ی (یال‌های $F-$سبک)MST باشد. در واقع مرحله دوم الگوریتم ما به نوعی مرحله «حرس کردن» است؛ در ابتدا یک نمونه‌ای می‌گیریم و زیردرخت فراگیر کمینه آن‌را حساب می‌کنیم و این زیردرخت به ما کمک می‌کند که تعدادی از یال‌های G را در نظر نگیریم و توجه خودمان را به یال‌های $F-$سبک معطوف کنیم که تنها یال‌های مفید هستند و می‌توانند در ساختن زیردرخت فراگیر کمینه استفاده شوند.

[20] Principle of Deferred Decisions

۱.۴ زمان اجرا

۱. برای محاسبه زیردرخت‌های فراگیر کمینه، یک راه این است که از بهترین الگوریتمی که تا کنون داریم استفاده کنیم؛ که مرتبه زمانی آن $O(m \lg^* n)$ است. از طرفی میانگین تعداد یال‌های نمونه، $O(mp)$ است، پس زمان اجرا $O(mp \lg^* n)$ است.

۲. همان‌طور که بیان شد، این را می‌پذیریم که الگوریتم‌های تأیید زیردرخت فراگیر کمینه در زمان $O(m)$ قابل انجام هستند و علاوه بر این، این الگوریتم‌ها یال‌های $F-$سبک را نیز میابند.

۳. طبق لم دیدیم که میانگین تعداد یال‌های $F-$سبک برابر $O(\frac{n}{p})$ است پس زمان پیدا کردن زیردرخت پوشای کمینه $O(\frac{n}{p} \lg^* n)$ می‌شود.

پس در کل، زمان اجرای الگوریتم $O(m + (mp + \frac{n}{p})lg^*n)$ است. حال چون p دست ماست، مقدار آن‌را طوری تعریف می‌کنیم که زمان اجرا کمینه شود:

$$mp = \frac{n}{p} \Rightarrow p = \sqrt{\frac{n}{m}}$$

پس زمان اجرا برابر:

$$O(m + \sqrt{mn} \lg^* n)$$

می‌شود که خیلی به زمان خطی مدنظر نزدیک است؛ یعنی اگر تعداد یال‌ها $m\Omega(n\lg^{*2})$ باشد، الگوریتم در زمان خطی اجرا می‌شود.

۱.۱.۴ بهبود زمان اجرا با تغییر بازگشت‌ها

در بخش قبل اگر تعداد یال‌ها به تعداد رأس‌ها نزدیک باشند و به‌عبارتی گراف تنک[۲۱] باشد، زمان اجرا خطی نخواهد بود. کاری که انجام می‌دهیم این است که برای پیدا کردن زیردرخت فراگیر کمینه از همین الگوریتم استفاده کنیم؛ یعنی به‌عنوان زیرفرآیند بجای الگوریتم قطعی با زمان اجرای $O(m \lg^* n)$، بصورت بازگشتی از خود این الگوریتم در قدم‌های یک و سه استفاده کنیم. با این‌کار به یک الگوریتم سریع‌تر می‌رسیم که دوباره می‌توانیم از خودش بعنوان زیربرنامه استفاده کنیم، و با ادامه این استدلال با وضعیتی مشابه به الگوریتم برش کمینه[۲۲] روبرو می‌شویم ونهایتاً ایده این است که خود همین الگوریتم را بصورت بازگشتی صدا کنیم و از الگوریتم دیگری استفاده نکنیم. در این‌صورت برای زمان اجرا به رابطه بازگشتی زیر می‌رسیم:

$$T(m, n) = T(mp, n) + O(m) + T(\frac{n}{p}, n)$$

حال اگر مقدار p را برابر $\frac{1}{2}$ قرار دهیم، آنگاه:

$$T(m, n) = T(\frac{m}{2}, n) + O(m) + T(2n, n)$$

برای حالتی که $m = 2n$، فرآیند نمونه‌گیری دیگر فایده نخواهد داشت و بهترین کار استفاده از الگوریتم‌های قطعی است. پس زمان اجرا به :

$$T(m, n) = T(\frac{m}{2}, n) + O(m) + O(n \lg^* n) = m + n \lg^* n + \frac{m}{2} + n \lg^* n + \frac{n}{4} + n \lg^* n + ...$$

$$\Rightarrow T(m, n) = O(m + n \lg^* n \lg n)$$

تغییر میابد.

[21] Sparse
[22] MinimumCut

۲.۱.۴ بهبود زمان اجرا با کاهش تعداد رأس‌ها

در واقع مشکلی که باعث خطی نشدن زمان اجرا می‌شود این است که با این‌که نمونه‌گیری باعث می‌شود یال‌ها به خوبی حرس شوند و نهایتاً زمان اجرا تشکیل یک سری هندسی بر حسب تعداد یال‌ها می‌دهد که درنتیجه بر حسب تعداد یال‌ها خطی می‌شود، اما نمونه‌گیری ما تعداد رأس‌ها را کاهش نمی‌دهد، بنابراین عباراتی که شامل n هستند در مراحل بازگشت، کوچک نمی‌شوند و درنهایت باعث مشکل می‌شوند.

پس اگر بتوانیم تعداد رأس‌ها را هم همزمان با تعداد یال‌ها کاهش دهیم، آنگاه می‌توانیم به زمان اجرای خطی برسیم. برای این کار کافیست از الگوریتم برووکا استفاده کنیم؛ در یک‌بار اجرای فرآیند برووکا، تعداد رأس‌ها حداقل نصف می‌شود. پس تلفیق ایده الگوریتم برووکا با ایده الگوریتم تصادفی باعث می‌شود همزمان بتوانیم هم تعداد یال‌ها و هم تعداد رأس‌ها را در هر تکرار کاهش دهیم تا نهایتاً جواب رابطه بازگشتی، خطی شود.

۱. سه گام الگوریتم برووکا را روی G اجرا کنید تا گراف G' را بدست آورید.

۲. سپس زیردرخت فراگیر کمینه $G'(\frac{1}{2})$ را محاسبه کنید.

۳. یال‌های $\mathsf{MST}(G'(\frac{1}{2}))-$سبک را محاسبه کنید و آنهارا L بنامید.

۴. سپس زیردرخت فراگیر کمینه L را محاسبه کنید.

با اجرای مرحله اول، گرافی بدست می‌آید که تعداد رأس‌های آن حداکثر $\frac{1}{8}$ تعداد رأس‌های گراف اصلی است. بقیه مراحل را روی گراف جدید بدست آمده انجام می‌دهیم؛ یال‌های آنرا با احتمال $\frac{1}{2}$ نمونه‌گیری می‌کنیم و زیردرخت فراگیر کمینه آنرا محاسبه کرده، یال‌های سبک نسبت به زیردرخت فراگیر کمینه‌ی بدست آورده را حساب می‌کنیم (با الگوریتم‌های تأیید زیردرخت فراگیر کمینه). فرض کنید مجموعه یال‌های سبک را L بنامیم. اکنون در نهایت زیردرخت فراگیر کمینه L را محاسبه می‌کنیم. گام یک و سه خطی هستند. گام دو فراخواندن بازگشتی الگوریتم روی گرافی با میانگین تعداد یال $\frac{m}{2}$ و تعداد رأس $\frac{n}{8}$ است. گام چهارم هم دیدیم که میانگین تعداد یال‌های سبک $\frac{n}{p}$ است که اینجا n برابر $\frac{n}{8}$ است پس داریم:

$$T(m,n) = O(m+n) + T(\frac{m}{2},\frac{n}{8}) + T(\frac{n}{4},\frac{n}{8}) = c_1(m+n) + T(\frac{m}{2},\frac{n}{8}) + T(\frac{n}{4},\frac{n}{8})$$

قضیه ۱. *زمان اجرای الگوریتم خطی است:*

$$T(m,n) \leq c(m+n)$$

اثبات. حکم را با استقرا نشان می‌دهیم. طبق فرض و رابطه بالا، اگر $c \geq 2c_1$ باشد، داریم:

$$T(m.n) = c_1(m+n) + c(\frac{m}{2}+\frac{n}{8}) + c(\frac{n}{4}+\frac{n}{8}) = (\frac{c}{2}+c_1)(m+n) \leq c(m+n)$$

□

پس اگر c را به اندازه‌ی کافی بزرگ انتخاب کنیم، حکم ثابت می‌شود. بعنوان تمرین می‌توانید ببینید که دوبار اجرای برووکا درگام اول، درنهایت موجب می‌شود زمان اجرا خطی نباشد.

پس نهایتاً به یک الگوریتم تصادفی رسیدیم که میانگین زمان اجرای آن خطی است؛ درواقع زمان اجرای آن با احتمال بالا خطی است که این نکته اثبات نشده باقی‌ماند.

دقت: نکته‌ای که وجود دارد این است که رابطه‌های بازگشتی نوشته‌شده مانند خیلی از الگوریتم‌های تصادفی دیگر، تصادفی هستند؛ یعنی زمان اجرا خود یک متغیر تصادفی است و عبارت های نوشته شده، میانگین متغیرها هستند. کاری که انجام دادیم لزوماً قابل اجرا

نیست. اما این‌جا روابط نوشته شده معتبر است چون زمان اجرا واقعاً خطی است. اما در حالت کلی معمولاً امید رابطه بازگشتی را نمی‌توانیم به امید متغیرهای کوچک‌تر ربط دهیم که دلیل آن هم این است که امید و تابع بصورت کلی نسبت به هم جابجا نمی‌شوند:

$$E[f(x)] \neq f(E[x])$$

$$E[x^2] \neq E[x]^2$$

بنابراین این گونه روابط همیشه معتبر نیستند. اما در حالت خاص، وقتی الگوریتم خطی است معتبر هستند پس حرف‌هایی که زدیم برقرار هستند.

نهایتاً توانستیم یک الگوریتم تصادفی با میانگین زمان اجرای خطی بدست آوردیم. اکنون می‌خواهیم ببینیم آیا الگوریتم قطعی با زمان اجرای خطی وجود دارد یا خیر.

۵ الگوریتم قطعی با زمان اجرای خطی

الگوریتم با زمان اجرای $O(m\alpha(m,n))$ که متعلق شزل[۲۳] است، از ایده‌ای مشابه نمونه‌برداری استفاده می‌کند. ما در نمونه‌گیری تصادفی، یک سری اشتباهات در ساخت زیردرخت فراگیر کمینه انجام می‌دهیم؛ یعنی یک تعدادی از یال‌هایی را که باید درنظر بگیریم، در نظر نمی‌گیریم (یال‌های سبک نسبت به درختی که نهایتاً ساخته می‌شود) و درنهایت به یک زیردرخت فراگیر کمینه می‌رسیم که «خوب» است و تعداد یال‌های اشتباه آن کم است و این زیردرخت به ما کمک می‌کند که زیردرخت فراگیر کمینه واقعی کل گراف را سریع‌تر بسازیم.

شزل، ایده اشتباه کردن در انتخاب یال‌ها را در جای دیگری نیز استفاده کرده است. در الگوریتم‌های استاندارد مسأله درخت فراگیر کمینه مانند الگوریتم پریم، یک صف الویت یا یک هرم داریم که در هر گام کمینه تعدادی عضو را به ما برمی‌گرداند. شزل ایده‌ای مطرح کرد که این ساختمان داده را هم جوری بسازیم که بتواند در هر مرحله تعدادی اشتباه کند، ولی در عوض سریع‌تر باشد. یعنی ساختمان داده‌ای که قطعی است ولی در عین حال گاهی ممکن است اشتباه کند؛ اکثراً عنصری که برمی‌گرداند عنصر کمینه است ولی گاهی ممکن است نباشد. این الگوریتم دقیقاً خطی نیست ولی خیلی به زمان اجرای خطی نزدیک است.

در اواخر سال ۲۰۰۰ پتی[۲۴] و راماچاردران[۲۵] الگوریتمی برای زیردرخت فراگیر کمینه اراعه کردند که ثابت شد این الگوریتم، بهینه است (یعنی اگر، الگوریتم بهینه در زمان T انجام شود این الگوریتم از مرتبه $O(T)$ خواهد بود). ولی در عین حال موفق به یافتن زمان دقیق اجرا نشدند! و نمی‌دانیم که این الگوریتم خطی است یا نیست.

ایده کلی اثبات بهینه بودن این الگوریتم، این است که اگر فرض کنیم الگوریتم بهینه را داشته باشیم، می‌توانیم روی گراف های کوچک از آن استفاده کرده و زیردرخت فراگیر کمینه آن‌ها را پیدا کنیم و از آن‌ها به‌عنوان حالت پایه استفاده کرده و مسأله را به‌صورت استقرایی برای گراف‌های بزرگ‌تر حل کنیم. حالا با اینکه الگوریتم بهینه را نداریم، می‌توانیم با جست‌وجوی تمام حالات روی گراف‌های کوچک، الگوریتم بهینه را پیدا کنیم.

۶ الگوریتم کوملس

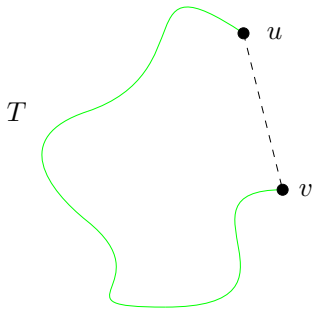
در این بخش می‌خواهیم الگوریتم کوملس[۲۶] را که نشان می‌دهد مسأله تأیید زیردرخت فراگیر کمینه با $O(m)$ مقایسه انجام می‌شود، بررسی کنیم. هدف درواقع پیدا کردن یال‌های سبک است.

[23] Chazelle
[24] Pettie
[25] Ramachardran
[26] Komlos algorithm

ورودی:درخت T.

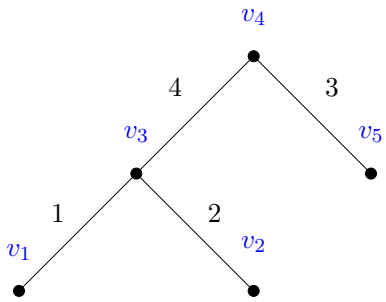
خروجی: یال‌های $T-$سبک .

درواقع کاری که می‌خواهیم انجام دهیم این است که به ازای هر یال گراف، بررسی کنیم که آیا مسیری که بین دوسر این یال در درخت T هست، آیا همه یال‌های آن سبک‌تر از یال uv هستند یا یک یال سنگین‌تر وجود دارد. پس درواقع می‌خواهیم وزن سنگین‌ترین یال مسیر uv در درخت T رابیابیم.

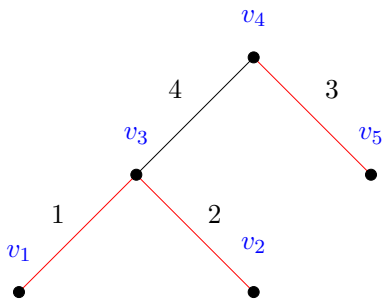


۱.۶ کاهش مسأله

می‌خواهیم درخت T را به درختی دیگر تبدیل کنیم به‌طوری که انجام محاسبات خواسته‌شده آسان‌تر شود. یک‌بار الگوریتم برووکا را روی درختی که داریم اجرا می‌کنیم و رأس‌های مراحل میانی الگوریتم را ذخیره می‌کنیم و با اینکار یک درخت T' ساخته می‌شود که پرسمان[۲۷] ها را روی آن بررسی می‌کنیم؛ به‌عنوان مثال فرض کنید درخت داده شده مشابه شکل زیر باشد:

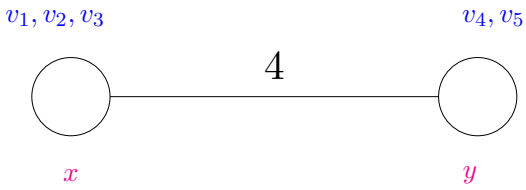


بعد از یک مرحله اجرای الگوریتم برووکا، یال‌های قرمز انتخاب می‌شوند:

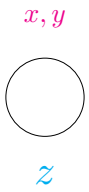


[27] Query

پس درخت T تبدیل به درختی مانند شکل زیر می‌شود.

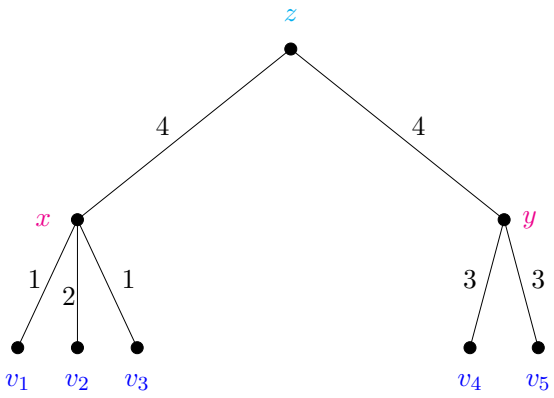


و بعد یک‌بار دیگر اجرای مراحل برووکا، به درخت زیر می رسیم.



حال، دقت کنید روند ساختن درخت T' به این صورت است که مرحله اول رأس‌های T را بعنوان برگ‌های درخت T' قرار می‌دهیم؛ درواقع هر عمق از درخت T'، متناظر با رأس‌های درخت‌مان در یکی از مراحل اجرای الگوریتم برووکا است. پایین‌ترین عمق، یا برگ‌های درخت T'، متناظر با رأس‌های درخت اولیه است، و بالاترین عمق آن‌هم متناظر با وقتی است که تعداد رأس‌های درخت در الگوریتم برووکا، یک می‌شود. وزن یال‌ها را هم برابر وزن کوچک‌ترین یالی که به آن رأس در آن مرحله از برووکا متصل بوده، می‌گذاریم. پس از یکبار اجرای برووکا، عمق دوم درخت T' مانند شکل زیر می‌شود:

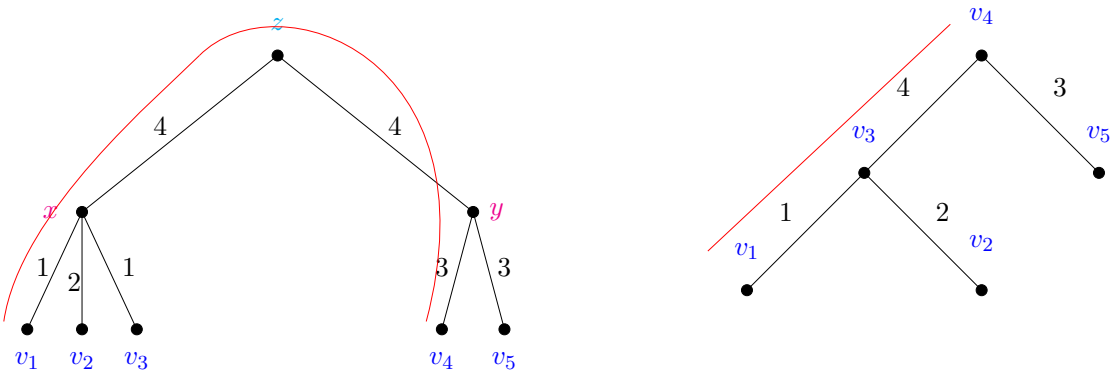
و بعد از دور سوم از الگوریتم برووکا، درخت T' به شکل زیر ایجاد می‌شود:



پس درخت T' به‌صورت سلسه مراتبی از روی درخت T بدست می‌آید.

نکته : با اجرای الگوریتم برووکا روی یک گراف، الگوریتم در هر دور خود تعداد رأس‌ها را نصف می‌کند. اینجا چون در هر دور از الگوریتم گرافی که ساخته می‌شود، درخت است، پس تعداد یال‌ها نیز نصف می‌شود. بنابراین زمان اجرای الگوریتم برووکا روی درخت T خطی است. بنابراین می‌توانیم بدون اینکه ازهدف زمان خطی خود دور شویم، درخت T' را تشکیل دهیم.

نکته : پرسمان‌هایی که داشتیم، به این صورت بودند که با دریافت دو رأس که متناظر با دو سر یال گراف اصلی هستند، سنگین‌ترین یال در مسیر بین این دو رأس در درخت T را بیابیم. ادعا می‌کنیم که جواب این پرسمان،در درخت‌های T و T' یکسان است. بعنوان مثال به مسیر بین v_1 و v_4 در دو درخت بدست آمده دقت کنید:



در هر دو مسیر مشخص شده، سنگین‌ترین یال ۴ است. بعنوان تمرین برای بقیه رأس‌ها هم می‌توانید این ادعا را بررسی کنید:

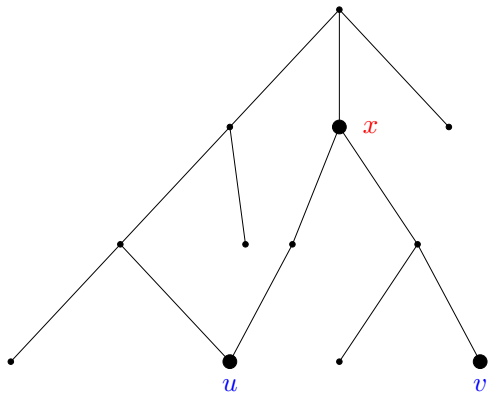
- مسیر v_1 به v_2 در هر دو: ۲
- مسیر v_5 به v_2 در هر دو: ۴

تمرین : ادعای بیان‌شده را در حالت کلی اثبات کنید.

پس از این به بعد، روی درخت T' بحث می‌کنیم. بطور خلاصه، m تا پرسمان به ما داده‌شده (برای هر یال گراف اصلی یک پرسمان) و می‌خواهیم این m پرسمان را در زمان $O(m)$ پاسخ بدهیم. هر پرسمان که دو رأس از گراف اصلی است، دو برگ از درخت T' خواهند بود. پس هدف پیدا کردن سنگین‌ترین یال یکتای بین این دو رأس در گراف T' است.

از طرفی اگر به درخت T' دقت کنید، هر پرسمان را می‌توانیم به دو تیکه بشکنیم. در کل وقتی یک درخت ریشه‌دار داریم و می‌خواهیم درباره مسیر بین دو برگ آن اطلاعاتی بدست آوریم، می‌توانیم این مسیر را به دو تیکه تقسیم کنیم و روی دو مسیر بدست آمده بحث کنیم. بعنوان مثال اگر در درخت زیر مسیر بین u و v را بخواهیم، کافیست پایین‌ترین جد مشترک[۲۸] آن دو رأس را یافته، که در مثال زیر رأس x است، و مسیر بین u و v را به دو مسیر بین u و x و مسیر بین x و v بشکنیم. پس در کل اگر بتوانیم برای دو رأس داده شده، پایین‌ترین جد مشترک آن‌دو را بیابیم، می‌توانیم فرض کنیم هر پرسمان، شامل دو رأس است که یکی از آن‌ها جد دیگری است. پس در نهایت اگر بتوانیم در $O(m)$ به m پرسمان به این شکل پاسخ دهیم و همچنین بتوانیم پایین‌ترین جد مشترک دو رأس خواسته‌شده را در $O(1)$ محاسبه کنیم، آنگاه مسأله اصلی را نیز می‌توانیم حل کنیم.

[28] LowestCommonAncestor, LCA



۲.۶ حل مسأله پایین‌ترین جد مشترک

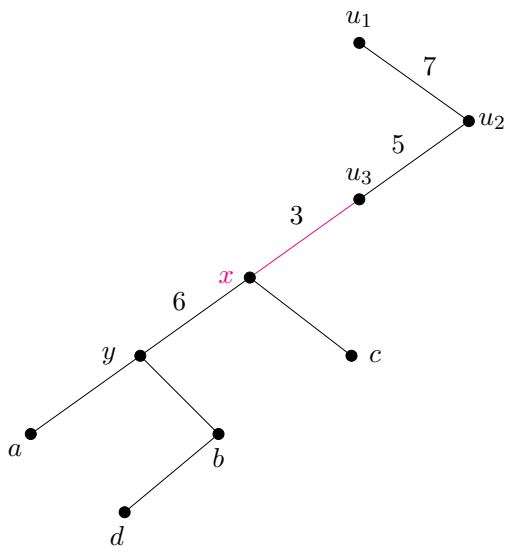
حل مسأله پایین‌ترین جد مشترک را قبلاً دیده‌ایم؛ برای این‌کار کافیست با یک‌بار پیش‌پردازش روی رأس‌ها با زمان $O(n)$ که n تعداد رأس های درخت است، یک ساختمان داده‌ای طراحی کنیم که با استفاده از آن برای هر دو رأسی که به ما می‌دهند، بتوانیم پایین‌ترین جد مشترک آن‌دو را در $O(1)$ محاسبه کنیم. برای حل دقیق‌تر می‌توانید به جلسه ۲۵ درس ساختمان داده مراجعه کنید.

پس پایین‌ترین جد مشترک در زمانی که می‌خواهیم قابل محاسبه است. مسأله‌ی باقی‌مانده، پیداکردن سنگین‌ترین یال بین یک رأس و جد آن است.

۳.۶ پیدا کردن سنگین‌ترین یال بین یک رأس و جد آن

فرض کنید در درخت T' دو رأس به ما داده‌شده که یکی جد دیگری است و می‌خواهیم در مسیر بین این دو، سنگین‌ترین یال را بیابیم. درواقع می‌خواهیم $O(m)$ پرسمان از این نوع را هم‌زمان پردازش کنیم به‌طوری‌که تعداد مقایسه‌ها $O(m)$ بشود. (دقت کنید ممکن است کل زمان الگوریتم از این کران بیشتر شود.)

برای این‌کار، یک یال خاص درخت T' را درنظر بگیرید. فرض x رأس پایینی این یال باشد. حال تمام پرسمان‌هایی که رأس پایینی آن‌ها یکی از نوادگان x است و رأس بالایی آن‌ها بالاتر از x است (یکی از اجداد آن) را در نظر بگیرید:



فرض کنید پرسمان‌های ما بصورت $(c,u_3),(b,u_2),(a,u_1)$ باشند. می‌خواهیم هر سه‌تای این پرسمان‌ها را بطور همزمان پردازش کنیم. تمام این پرسمان‌ها به‌گونه‌ای از یال xu_3 عبور می‌کنند. نحوه‌ی پردازش ما این است که سر پایینی از پرسمان‌ها با اینکه متفاوت ازهم‌اند را x درنظر می‌گیریم و پرسمان‌ها را مرحله به مرحله بروزرسانی می‌کنیم. پس فعلاً می‌خواهیم پرسمان‌های $(x,u_1),(x,u_2),(x,u_3)$ را جواب بدهیم. می‌توانیم رشته‌ی پرسمان‌ها را بصورت (u_1,u_2,u_3) بنویسیم که با توجه به شکل پاسخ آنها $(7,5,3)$ می‌شود.

- $(a,u_1)\longrightarrow(x,u_1)\longrightarrow 7$
- $(b,u_2)\longrightarrow(x,u_2)\longrightarrow 5$
- $(c,u_3)\longrightarrow(x,u_3)\longrightarrow 3$

اکنون می‌خواهیم مرحله به مرحله پرسمان‌ها را با توجه به سر پایینی آن‌ها به‌روزرسانی کنیم. در حقیقت تاکنون توانسته‌ایم بخشی از هر پرسمان را پاسخ دهیم.

در مرحله بعد، یال بعدی، مثلاً یال yx را پردازش می‌کنیم. گویی با شروع از بالای درخت، تیکه به تیکه تمام پرسمان‌ها را محاسبه کرده و هر مرحله، یک قدم پایین‌تر آمده، پرسمان‌ها را به‌روزرسانی کنیم. پس، چون x دو بچه دارد، هر کدام از دو یال آن به بچه‌هایش را جدا بررسی می‌کنیم. با در نظر گرفتن یال xy، از سه پرسمان اصلی که داشتیم، دوتا از آن پرسمان‌ها از این یال رد می‌شوند؛ یعنی رشته پرسمان‌های (u_1,u_2). در این مرحله ممکن است پرسمان‌هایی که رأس بالایی آن‌ها x است نیز به پرسمان‌های مرحله قبلی اضافه شوند. بعنوان مثال اکنون می‌تواند پرسمان (x,d) اضافه شود. دوباره هنگام پردازش یال yx، سر پایینی پرسمان‌ها را y درنظر می‌گیریم. بنابراین رشته‌ی پرسمان‌ما در مرحله پردازش یال yx، تبدیل به (u_1,u_2,x) می شود. تاکنون برای u_1 و u_2 یک جواب پیدا کرده‌ایم. پس اکنون باید برای آن دو بین پاسخ بدست آمده و 6، وزن یال جدید، بیشینه بگیریم (در حقیقت انگار طبق مسیر اصلی، یک یال دیگر را هم به مسیر بررسی شده اضافه کرده و بررسی را ادامه می‌دهیم):

- $(a,u_1)\longrightarrow(y,u_1)\longrightarrow 7$
- $(b,u_2)\longrightarrow(y,u_2)\longrightarrow 6$
- $(x,d)\longrightarrow(y,x)\longrightarrow 6$

نکته الگوریتم این است که ما می‌توانیم از یک مرحله به مرحله بعدی، از روی رشته‌ی جواب قبلی، رشته جواب کنونی را بصورت کارا محاسبه کنیم (و صرفاً رشته جواب قبلی را به‌روزرسانی کنیم). درواقع کاری که باید انجام شود بیشینه گرفتن بین جوابی که برای هر پرسمان بدست آورده بودیم با وزن یال جدیدی که بررسی می‌کنیم است. نهایتاً باید تمام یال‌ها پردازش شوند.

حال، این نکته مهم است که لازم نیست در هر مرحله، روی کل رشته جواب برای هر عنصر آن یکبار بیشینه‌گیری کنیم، بلکه کافیست لیست جواب‌ها را بصورت مرتب‌شده نگه‌داری کنیم و تنها یا یک‌بار جست‌وجوی دودویی[۲۹] وزن یال جدید را در آن پیدا کرده، از آن اندیس به بعد رشته جواب را با وزن یال جدید جایگزین کنیم. پس ما در زمان لگاریتمی بر حسب اندازه‌ی رشته‌ی تولید شده در هر مرحله، می‌توانیم پاسخ رشته‌ی یال بچه آن را بیابیم.

همچنین در هر عمق درخت، جمع کل تعداد پرسمان‌هایی که از یال‌های آن عمق می‌گذرند، حداکثر m است؛ زیرا در کل m پرسمان داریم و هر پرسمانی از حداکثر یک یال هر مرحله عبور می‌کند زیرا پرسمان‌ها بصورت یک رأس با جد آن رأس هستند.

پس درنهایت، اگر تمام این لگاریتم‌ها را جمع کنیم، به جواب $O(m)$ می‌رسیم. برای دیدن محاسبات دقیق‌تر اینجا را مشاهده کنید.

[29] BinarySearch

مراجع

[۱] الگوریتم‌های سریع‌تر برای مسئله درخت فراگیر کمینه، درس آنالیز الگوریتم، دکتر مرتضی علیمی

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

الگوریتم‌های پارامتر ثابت [1] نگارنده: شایان طاهری جم

این نوع الگورینم‌ها یک جور نگاه دیگر و یک روش حمله دیگر به مسائل انپی_تمام هستند.
ما دیدیم که مسائل انپی_تمام را نمی‌توانیم (بلد نیستیم) با الگوریتم‌های چندجمله‌ای حل کنیم و حدس زده می‌شود که برای چنین مسائلی الگوریتم چندجمله‌ای وجود نداشته باشد، ولی می‌توانیم این مسائل را با الگوریتم‌های نمایی حل کنیم و کاری کنیم که در عمل خوب جواب بدهند و مفید باشند.
در الگوریتم‌های پارامتر ثابت یک پارامتر از مسئله را در نظر می‌گیریم و الگوریتمی ارائه می‌دهیم که فقط بر حسب آن پارامتر نمایی باشد نه کل ورودی مسئله، حال این را با یک مثال نشان می‌دهیم:

۱ مسئله پوشش رأسی:

در این مسئله می‌خواهیم بدانیم آیا زیرمجموعه‌ی حداکثر k عضوی از مجموعه رئوس گراف ورودی مسئله وجود دارد به طوری که هر یال از گراف حداقل به یکی از اعضای این مجموعه متصل باشد؟
ورودی: گراف G و عدد k.
خروجی: آیا G پوشش رأسی با اندازه حداقل k دارد؟

الگوریتم بدیهی: با جست‌وجوی همه حالت‌ها در زمان اجرا $O(2^n E)$ می‌توانیم مسئله را حل کنیم مثلا اگر $k = 20$ و $n = 10000$ زمان اجرا تقریباً برابر 10^{300} می‌شود.

با الگوریتم جست‌وجو تمام زیرمجموعه‌های k عضوی می‌توانیم مسئله را در زمان $O(n^k E)$ حل کنیم که برای $k = 20$ و $n = 10000$ زمان اجرا تقریباً برابر 10^{80} می‌شود.
از الگوریتم قبل بهتر است ولی هنوز مطلقاً قابل استفاده نیست

در الگوریتم اول الگوریتم بر حسب ورودی که $O(n^2)$ است نمایی است ولی در الگوریتم دوم اگر k را ثابت فرض کنیم این الگوریتم چندجمله‌ای خواهد بود ولی همچنان اگر گراف ما بزرگ باشد و k حداقل 2 یا 3 باشد زمان اجرا بد خواهد بود.
حال اگر بتوانیم الگوریتمی طراحی کنیم که زمان اجرای آن به شکل $O(f(k).n^c)$ باشد که در آن c به k وابسته نباشد این الگوریتم بهتر خواهد بود.
برای مسئله پوشش رأسی الگوریتم پارامتر ثابت با زمان اجرای $O(2^k.n)$ داریم که اگر $k = 20$ و $n = 10000$ زمان اجرای آن حدوداً 10^{10} می‌باشد که زمان خوب و قابل قبولی است.
پس همانطور که مشاهده کردیم اگر الگوریتمی بسازیم که قسمت نمایی زمان اجرای آن فقط بر حسب k باشد و بر حسب n یا سایر ورودی‌های دیگر چندجمله‌ای باشد، این الگوریتم کاربردی خواهد بود.

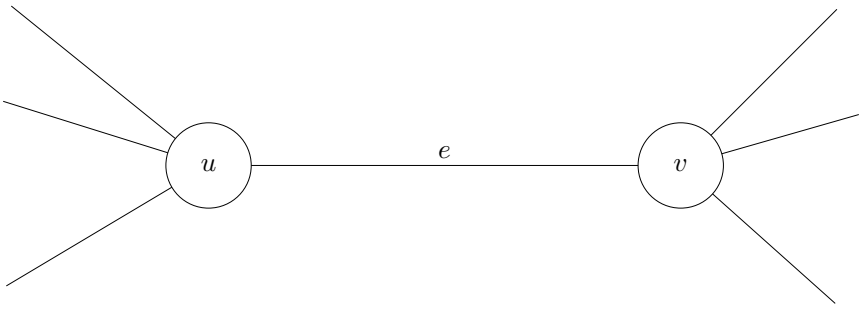
[1] Fixed Parameter Algorithms

در این مسئله پارامتر k در مسئله حضور داشت و خیلی وقت‌ها پارامتری که ما می‌خواهیم آن را ثابت در نظر بگیریم و بر حسب آن الگوریتم پارامتر ثابت مطرح کنیم در مسئله حضور دارد مثلا سایز بزرگترین مجموعه مستقل. ولی خیلی وقت‌ها ممکن است ما پارامتر را انتخاب کنیم، و اینکه چه پارامتری انتخاب کنیم و الگوریتمی طراحی کنیم که زمان اجرا آن بر حسب آن پارامتر نمایی شود یا در واقع $f(k)$ چه باشد دست ما است.
به اینگونه مسائل که خود پارامتر مناسب را دارند مسائل پارامتریزه شده می‌گویند.

پارامتر تابعی از ورودی مانند $k(X)$ است که X ورودی مسئله است.
گاهی ممکن است این پارامتر در مسئله ظاهر شده باشد یا به راحتی قابل محاسبه باشد مانند بیشترین درجه در گراف، و گاهی ممکن است محاسبه این پارامتر خود انپی_سخت باشد. مثلا گاهی ممکن است پارامتر ما اندازه خروجی باشد و الگوریتمی می‌دهیم که برای اندازه خروجی کوچک خوب کار می‌کند، یا گاهی می‌توان پارامتر را عرض درختی[۲] گرفت که در این جلسه راجع به آن صحبت نمی‌شود ولی خیلی از مسائل انپی_سخت هستند که روی درخت‌ها الگوریتم چندجمله‌ای دارند، مانند مجموعه مستقل و ما می‌توانیم برای گراف‌هایی که نزدیک به درخت هستند الگوریتمی خوب با پارامتر ثابت عرض درختی ارائه کنیم.

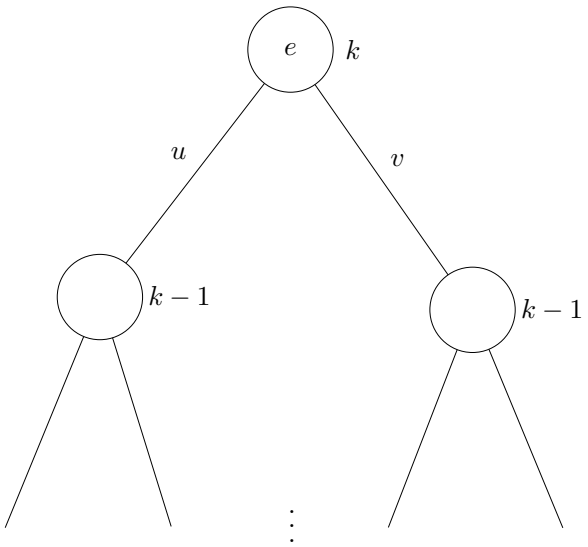
۲ الگوریتم درخت جست‌وجوی کراندار[۳] برای مسئله پوشش رأسی:

یال دلخواهی مانند e را درنظر می‌گیریم:



برای پوشانده شدن این یال یا باید رأس u انتخاب شود یا رأس v. فرض کنید رأس u انتخاب شود. آنگاه یال‌های متصل به u را از گراف حذف می‌کنیم و این الگوریتم را ادامه می‌دهیم.
درختی مانند شکل زیر پدید می‌آید:

[2] tree width
[3] Bounded search tree



در هر طبقه آن پارامتر ثابت آن یکی کم می‌شود و هر رأس آن متناظر با یک یال از گراف است.
شبه کد این الگوریتم :

```
BOUNDED STVC(G,k)
1  if k = 0
2      if G = ∅
3          return TRUE
4      else
5          return FALSE
6  Pick uv ∈ E
7  t1 = BOUNDED STVC(G-u , k - 1)
8  t2 = BOUNDED STVC(G-v , k - 1)
9  return t1 Or t2
```

قضیه ۱. برای هر مسئله پارامتریزه، الگوریتم با زمان اجرای $O(f(k).n^c)$ وجود دارد اگر و تنها اگر الگوریتم با زمان $O(f'(k) + n^{c'})$ وجود داشته باشد.

اثبات. یک طرف قضیه بدیهی است و برای طرف دیگر آن دو حالت در نظر می‌گیریم .

حالت ۱: $n \leq f(k)$
که در این حالت نتیجه می‌شود $f(k).n^c \leq f(k)^{c+1}$
حالت ۲: $n \geq f(k)$
که در این حالت نتیجه می‌شود $f(k).n^n \leq n^{c+1}$

پس در هر دو صورت داریم: $f(k)n^c \leq f(k)^{c+1} + n^{c+1}$ پس اگر قرار دهیم $f'(k) = f(k)^{c+1}$ و $c' = c + 1$ آنگاه $f(k)n^c \leq f'(k) + n^{c'}$ پس طرف دیگر قضیه نیز اثبات می‌شود.

□

می‌گوییم مسئله A عضو کلاس پارامتر ثابت است اگر الگوریتم با زمان اجرای $O(f(k).n^c)$ داشته باشد.

۳ هسته سازی [۴]

هسته سازی نوعی ساده سازی از طریق تحویل یک نمونه از یک مسئله به یک نمونه ساده‌تر از همان مسئله در زمان چندجمله‌ای است. درواقع. نمونه‌ی (X,k) را به نمونه (X',k') تبدیل می‌کنیم به طوری که $(x \in A) \leftrightarrow (x' \in A)$ و $|X'| \leq f(k)$.

برای مثال فرض کنید می‌خواهیم یک پوشش رأسی روی گراف G پیدا کنیم. بیاییم یک گراف کوچک تر مانند G' پیدا کنیم که G' پوشش رأسی به اندازه حداقل k دارد اگر و تنها اگر G داشته باشد و اندازه G' کوچکتر از $f(k)$ باشد.
حال اگر جواب مسئله را برای G' در زمان $g(|V(G')|)$ داشته باشیم درواقع برای گراف G الگوریتم در زمان $O(g(f(k)) + n^c)$ خواهیم داشت که n^c در آن زمان تبدیل G به G' است که باید چندجمله‌ای باشد.
حال طبق ۱. این مسئله پارامتر ثابت است.

قضیه ۲. مسئله پارامتریزه شده‌ی P ،پارامتر ثابت است، اگر و تنها اگر هسته سازی داشته باشد.

اثبات. یک طرف قضیه را که در بالا اثبات کردیم. حال فرض کنید P ، پارامتر ثابت است یعنی الگوریتمی با زمان اجرای $O(f(k).n^c)$ دارد. می‌خواهیم ثابت کنیم هسته سازی دارد.
حالت ۱ : اگر $f(k) \leq n$ الگوریتم را اجرا می‌کنیم و برحسب این که جواب بله یا خیر باشد یک نمونه کانونیک بله یا خیر خروجی می‌دهیم. در واقع با الگوریتمی چندجمله‌ای مسئله را به مسئله‌ای کوچکتر تبدیل کرده‌ایم که جواب آن مشخص است. مثلاً اگر جواب الگوریتم بله بود مسئله را گراف k_2 و k' را برابر با 1 درنظر می‌گیریم. و اگر جواب خیر بود همان گراف و با k' برابر 0 درنظر می‌گیریم.

حالت ۲ : اگر $f(k) \geq n$ ، $|X| \leq f(k)$ پس همان X را خروجی می‌دهیم. (درواقع خود مسئله هسته سازی‌شده است چون اندازه آن کوچک است)

حال ممکن است از اول ندانیم که حالت ۱ برقرار است یا حالت ۲.
الگوریتم را اینگونه اجرا می‌کنیم:
ابتدا الگوریتمی که داریم را اجرا می‌کنیم و یک زمان سنج قرار می‌دهیم اگر الگوریتم در زمان کمتر از n^{c+1} پایان یافت حالت اول است و اگر نیافت حالت دوم است و مانند همان حالات عمل می‌کنیم. □

۴ هسته سازی برای مسئله پوشش رأسی

می‌توانیم فرض کنیم گراف ورودی ساده است. اگر طوقه داشتیم آن رأس را حذف و k را یک واحد کم می‌کنیم و اگر یال‌های موازی داشتیم یکی از آن‌ها را نگه می‌داریم و بقیه را حذف می‌کنیم.

مراحل زیر را تا جای ممکن تکرار می‌کنیم:
۱ اگر رأس ایزوله داریم حذف‌اش می‌کنیم.
۲ اگر رأسی مانند u وجود داشته باشد که $d(w) > k$ آن رأس باید حتما در پوشش باشد (در غیر این صورت تمام همسایه‌های آن که بیشتر از k تا هستند باید در پوشش باشند که نمی‌شود) آن رأس را حذف می‌کنیم و k را یک واحد کاهش می‌دهیم..

[4] kernelization

در پایان درجه هر رأس کمترمساوی k است (منظور از k در واقع k نهایی است که در مراحل بالا حساب شده است).
اگر تعداد یال‌ها بیشتر از k^2 بود یک نمونه کانونیک NO خروجی می‌دهیم.(زیرا با k رأس با درجه حداکثر k نمی‌توان بیش‌از k^2 یال را پوشاند)
وگرنه تعداد رئوس حداکثر $2k^2$ است. پس گراف باقی‌مانده کوچک است.(تعداد رئوس و یال‌های آن برحسب k چندجمله‌ای هستند) پس این گراف را خروجی می‌دهیم.

زمان اجرای الگوریتم هسته سازی

به طور بدیهی الگوریتم در زمان V^2 دارد.

الگوریتم در زمان $O(V+E)$ هم دارد که به شرح زیر است:
در ابتدا درجه تمام رئوس را حساب می‌کنیم و دو لیست تهیه می‌کنیم. در یکی رئوس ایزوله و در دیگری رئوس با درجه بیشتر از k را قرار می‌دهیم. هر مرحله یک رأس از یکی از لیست‌ها حذف می‌کنیم و همسایه‌های آن را به‌روز می‌کنیم.
انتخاب هر رأس زمان $O(1)$ می‌خواهد پس انتخاب رئوس مجموعاً در زمان $O(V)$ انجام می‌شوند و به‌روز کردن رئوس مجموعاً در زمان $O(E)$ انجام می‌شود پس هسته سازی در زمان $O(V+E)$ انجام می‌شود.

حال که هسته سازی کرده‌ایم می‌توانیم الگوریتم‌هایی که برای پوشش رأسی داشته‌ایم را روی گراف تولید شده توسط هسته سازی اجرا کنیم.

۱ الگوریتم انتخاب همه زیرمجموعه‌های k عضوی :
زمان این الگوریتم $O(V^kE)$ است که اگر روی هسته اعمال شود خواهیم داشت $V \leq 2k^2$ و $E \leq k^2$ پس زمان حل مسئله با این الگوریتم (هسته سازی بعلاوه اجرا الگوریتم روی هسته) برابر خواهد بود با:

$$O(V+E+(2k)^k.k^2)$$

۲ الگوریتم (درخت جست‌وجوی کراندار):
زمان این الگوریتم $O(2^kV)$ است که اگر روی هسته اعمال شود خواهیم داشت $V \leq 2k^2$ پس زمان حل مسئله با این الگوریتم (هسته سازی بعلاوه اجرا الگوریتم روی هسته) برابر خواهد بود با:

$$O(V+E+2^k.k^2)$$

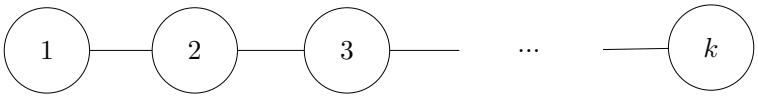
برای این مسئله الگوریتم با زمان اجرای $O(kV+1.274^k)$ نیز وجود دارد.

۵ کدگذاری رنگی[۵]

مثال: وجود مسیر به طول k :
این مسئله ان‌پی ـ تمام است زیرا با آن می‌توان وجود مسیر همیلتونی را حل کرد.
الگوریتم با زمان اجرای $O(n^{O(k)})$ داریم.

رأس‌ها را به طور تصادفی با k رنگ، رنگ می‌کنیم و چک می‌کنیم آیا مسیری به شکل زیر وجود دارد یا خیر؟

[۵] color coding



اگر در گراف اصلی مسیر به طول k وجود نداشته باشد جواب مسئله فوق خیر است و اگر مسیر به طول k وجود داشته باشد جواب این مسئله به احتمال حداقل k^{-k} بله است (هر رأس از یک مسیر به طول k به احتمال $\frac{1}{k}$ درست رنگ می‌شود)

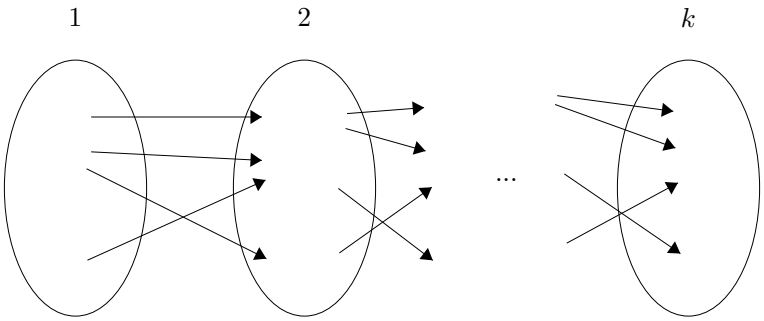
اگر احتمال موفقیت p باشد، بعد از $\frac{1}{p}$ بار تکرار الگوریتم احتمال شکست برابر خواهد بود با:

$$\Pr[failure] = (1-p)^{\frac{1}{p}} \leq (e^{-p})^{\frac{1}{p}} = \frac{1}{e}$$

پس اگر الگوریتم بالا را $O(k^k)$ بار اجرا کنیم به احتمال بالا جواب درست را می‌گیریم. پس اگر برای مسئله فوق الگوریتم چندجمله‌ای $(O(n^c))$ داشته‌باشیم برای مسئله الگوریتم احتمالی با زمان اجرای $O(k^k.n^c)$ خواهیم داشت. ☒ حال باید برای مسئله فوق الگوریتم چندجمله‌ای ارائه دهیم.

فرض کنید رئوس با k رنگ رنگ شده‌اند. از بین یال‌های گراف فقط یال‌هایی را نگه می‌داریم که بین دو رأس با دو رنگ به شماره‌های متوالی باشند(باقی یال‌ها نمی‌توانند در مسیری که می‌خواهیم پیدا کنیم حضور داشته باشند، پس بود یا نبود آن‌ها تأثیری ندارد). حال به یال‌ها جهت می‌دهیم. جهت یال‌ها را از طرف رأس با شماره رنگ کوچک‌تر به طرف رأس با شماره رنگ بزرگ‌تر قرار می‌دهیم.

گراف حاصل به شکل می‌باشد که رئوس به رنگ i در دسته i ام قرار دارند:



پس اکنون به یک گراف جهت دار بدون دور[۶] رسیده‌ایم که الگوریتم پیدا کردن بلند ترین مسیر این نوع گراف در زمان چندجمله‌ای را قبلاً داشته‌ایم.

پس اکنون الگوریتم احتمالاتی با زمان $O(k^k.n^c)$ داریم.

می‌خواهیم الگوریتم سریع تری برای مسئله ارائه دهیم.

ایده این است که رئوس را مانند حالت قبل رنگ کنیم، ولی به جای این که مسیری با ترتیب رنگی 1 تا k پیدا کنیم این بار مسیری به طول k پیدا کنیم به طوری که رئوس آن هر k رنگ را داشته باشند(تفاوت این حالت با حالت قبل این است که ترتیب رنگ‌های ظاهر شده در رئوس در حالت قبل مهم بود ولی در این حالت مهم نیست).

حال می‌خواهیم احتمال موفقیت در این حالت را محاسبه کنیم:

اگر مسیر به طول k نداشته‌باشیم قطعاً جواب مسئله فوق خیر خواهد بود و در این حالت به احتمال 1 جواب ما درست خواهد بود.

حال اگر مسیری به طول k در گراف موجود باشد تعداد حالات رنگ آمیزی این رئوس با k رنگ متفاوت برابر با $k!$ می‌باشد و کل حالات رنگ آمیزی این رئوس k^k حالت دارد. پس به احتمال $\frac{k!}{k^k}$ الگوریتم جواب مسئله را درست می‌دهد.

[6] DAG

داریم:

$$\frac{k!}{k^k} > \frac{(\frac{k}{e})^k}{k^k} = e^{-k}$$

پس این الگوریتم به احتمال e^{-k} درست جواب می‌دهد و این احتمال از احتمال درست جواب دادن حالت قبل (k^{-k}) خیلی بهتر است.
حال این الگوریتم را باید $O(e^k)$ مرتبه تکرار کنیم تا الگوریتممان با احتمال بالا جواب درست بدهد.
ولی چک کردن این که آیا در گرافی که با k رنگ، رنگ شده است مسیری به طول k با k رنگ متمایز وجود دارد یا خیر کار راحتی نیست و الگوریتم آن نمایی می‌شود.
این مسئله مانند مسئله‌ی فروشنده‌ی دوره‌گرد با برنامه‌ریزی پویا در زمان $O(2^k.n^{c'})$ حل می‌شود که می‌توانید خودتان روی آن فکر کنید.
اکنون برای مسئله‌ی وجود مسیر به طول k می‌توانیم الگوریتم فوق را که زمان اجرای آن $O(2^k.n^{c'})$ است را $O(e^k)$ بار تکرار کنیم و به الگوریتمی احتمالاتی با زمان اجرای $O((2e)^k.n^{c'})$ برسیم که از زمان اجرای الگوریتم قبل ($O(k^k.n^c)$) خیلی بهتر است.

۶ پیچیدگی مسائل

در جلسات قبل راجع به کلاس‌های پیچیدگی انپی و انپی_تمام صحبت کرده‌ایم.
یک تحویل چندجمله‌ای معرفی کردیم که اگر به کمک این تحویل می‌توانستیم مسئله‌ای از کلاس انپی_تمام مانند A را به یک مسئله از کلاس انپی مانند B تحویل کنیم آنگاه ثابت می‌شد که مسئله B نیز از کلاس انپی_تمام است.
ولی این تحویل چندجمله‌ای برای مسائل کلاس پارامتر ثابت صحیح عمل نمی‌کند.

برای مثال می‌توانیم مسئله مجموعه مستقل را به مسئله پوشش رأسی تحویل کنیم .
این تحویل بسیار ساده است. فرض کنید مجموعه A پوششی رأسی برای گراف G است. اکنون می‌دانیم هر یال G حداقل به یکی از اعضای A متصل است، پس هیچ یالی از G نمی‌تواند دو رأس از مجموعه $G\backslash A$ را به هم وصل کند. پس $G\backslash A$ یک مجموعه مستقل است.
و به طریق مشابه اگر A یک مجموعه مستقل از رئوس G باشد هیچ کدام از یال‌های G نمی‌توانند بین دو تا از اعضای A باشند، پس حداقل به یکی از اعضای $G\backslash A$ وص هستند. پس مجموعه $G\backslash A$ یک پوشش رأسی است.

با توجه به نکات بالا جواب مسئله‌ی پوشش رأسی با ورودی (G,k) برابر با جواب مسئله مجموعه مستقل با ورودی $(G,n-k)$ است. پس مسئله مجموعه مستقل به مجموعه پوشش رأسی تحویل می‌شود.
ولی در این تحویل پارامتر دوم تغییر می‌کند و خراب می‌شود. به طور مثال اگر k خیلی کوچک باشد آن وقت $n-k$ بسیار بزرگ می‌شود و چون الگوریتم ما برای پوشش رأسی برحسب مؤلفه دوم ورودی آن $(n-k)$ نمایی است پس الگوریتم ما که توسط این تحویل برای برای مسئله مجموعه مستقل به دست می‌آید برحسب $n-k$ نمایی است. چون k کوچک است پس $n-k$ نزدیک به n است پس الگوریتم ما بر حسب n نمایی می‌شود که خوب نیست.

پس تحویل آن چندجمله‌ای که قبلاً بلد بوده‌ایم اینجا به کار نمی‌آید.
حال می‌خواهیم یک تحویل دیگر معرفی کنیم.

۷ تحویل پارامتری

تحویل پارامتری A به B از ورودی‌های مسئله‌ی A که به صورت (X,k) هستند ورودی‌ای برای B به صورت (X',k') می‌سازد به طوری که سه شرط زیر را داشته باشد:
۱_ $X \in A \iff X' \in B$

۲ ـ زمان اجرای تحویل $f(k).|X|^c$ باشد ($f(k)$ می‌تواند بر حسب k نمایی باشد).
۳ ـ $k' \leq g(k)$ (این خاصیت نشان می‌دهد که پارامتر مسئله دوم نباید خیلی بزرگتر از پارامتر مسئله اول باشد).

حال به راحتی می‌توان دید که اگر A در کلاس پارامتر ثابت باشد آنگاه B نیز در این کلاس خواهد بود.

۸ کلاس‌های پیچیدگی سلسله مراتب W

بر خلاف ان‌پی‌ـ تمام که کلی از مسائلی که الگوریتم چندجمله‌ای برای آن‌ها نداریم و سخت هستند معدل‌اند، برای مسائلی که الگوریتم پارامتر ثابت نداریم این طور نیست که با هم معادل باشند و بینهایت درجه سختی دارند که به آن‌ها به ترتیب درجه سختی $W[1]$ و $W[2]$ و ... گفته می‌شود. به طور مثال مسئله مجموعه مستقل در کلاس $W[1]$ است و مسئله مجموعه چیره[۷] در کلاس $W[2]$ قرار دارد.
یعنی تحویل پارامتری از مجموعه مستقل به مجموعه چیره داریم.

این مقدمه کوتاهی بر الگوریتم‌های پارامتر ثابت بود.

[7] Dominating Set

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه اضافه ۱۴: تجزیه‌ی درختی

نگارنده: امیرحسین افشارراد و احسان شریفیان

۱ مقدّمه

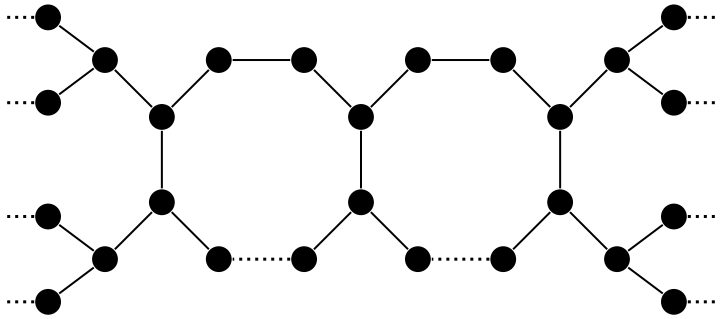
از جلسات مربوط به پیچیدگی محاسباتی به خاطر داریم که یکی از راه‌های درگیرشدن با مسائل سخت، حل کردن آن‌ها بر روی ورودی‌های خاص بود. به عنوان مثال مسأله‌ی یافتن پوشش رأسی در گراف‌های کلّی یک مسأله‌ی ان‌پی–سخت است؛ با این حال بر روی گراف‌های دوبخشی الگوریتمی چندجمله‌ای برای حلّ آن موجود است. یکی از کلاس‌های قابل توجّه از گراف‌ها که بسیاری از مسائل سخت برای آن قابل حل هستند، درخت‌ها می‌باشند. مسأله‌ی یافتن زیرمجموعه‌ی مستقل در گراف، نمونه‌ای از این مسائل است. این موضوع، سؤالی را پدید می‌آورد که آیا می‌توان معیاری برای نزدیک بودن گراف‌های کلّی به درخت‌ها تعریف کرد؛ به گونه‌ای که حتّی اگر گرافی درخت نباشد امّا طبق این معیار شبیه به یک درخت باشد، شاید بتوان بسیاری از مسائل سخت را بر روی آن نیز با الگوریتم مناسبی حل نمود. به عبارت دقیق‌تر، سؤال این است که آیا می‌توان پارامتری مانند k برای گراف‌ها تعیین کرد، به گونه‌ای که k ملاکی از نزدیکی گراف به درخت باشد و در برخی مسائل سخت بتوان الگوریتمی با زمان اجرای $f(k)\mathsf{poly}(n)$ بر روی چنین گرافی ارائه داد، به گونه‌ای که تنها ممکن باشد زمان اجرا بر حسب k نامطلوب باشد (مثلاً f تابعی نمایی از k باشد). در چنین حالتی حتّی ممکن است الگوریتمی با زمان اجرای $O(n^k)$ نیز برای حلّ مسأله‌ای یافت شود که اگر k عددی ثابت باشد، آن نیز الگوریتمی چندجمله‌ای خواهد بود. به این ترتیب حتّی اگر گرافی درخت نباشد امّا عدد k برای آن کوچک باشد (یعنی با توجّه به معیار تعریف‌شده شبیه به درخت باشد)، آن‌گاه شاید بتوان برخی مسائل سخت را بر روی آن نیز با زمان اجرای مناسب حل کرد.

پیش از آن که در ادامه به بررسی پاسخ پرسش فوق و یافتن معیار مناسب بپردازیم، خوب است به این سؤال نیز توجّه کنیم که کدام ویژگی درخت‌ها را خاص می‌کند تا برخی مسائل سخت بر روی آن‌ها حل‌های مناسب‌تری داشته باشند. به صورت مختصر می‌توان پاسخ این سؤال را به این نحو بیان کرد که «تعامل موجود بین قسمت‌های مختلف گراف در درخت‌ها محدود است». شاهدی بر این مفهوم آن است که هر یال یک درخت برشی است؛ یعنی با حذف آن درخت به دو قسمت مجزّا تبدیل می‌شود که هیچ ارتباطی با یکدیگر ندارند. به عنوان مثال اگر مسائلی که به کمک برنامه‌ریزی پویا روی درخت‌ها حل می‌شدند را به خاطر بیاورید، در بسیاری از موارد ایده‌ی کلّی حل آن بود که درخت را از یک رأس دلخواه ریشه‌دار کنیم و زیرمسأله‌های مربوطه را بر روی زیردرخت‌های موجود در هر یک از فرزندان ریشه حل کنیم. با توجّه به این که این زیردرخت‌ها هیچ‌گونه تعاملی (به جز از طریق خود ریشه) با یکدیگر نداشتند، نوعی استقلال در زیرمسأله‌های متناظر با آن‌ها ایجاد می‌شد که نهایتاً این امکان را فراهم می‌کرد که بتوان از یافتن حلّ هر یک از آن‌ها، به یک حلّ کلّی برای مسأله‌ی اصلی رسید. چنین امکانی در گراف‌های عادی وجود ندارد، چرا که امکان ایجاد چنین تقسیم‌بندی‌ای بر روی گراف (به گونه‌ای که مسأله‌ی اصلی به چند زیرمسأله‌ی مستقل از هم شکسته شود) در حالت کلّی وجود ندارد. به این ترتیب شاید ایده‌ای مناسب برای تعیین نزدیکی یک گراف به درخت، امکان ایجاد چنین ساختاری (با بخش‌های مجزّا یا با کمترین تعامل) در آن گراف باشد.

به صورت کلّی می‌توان ویژگی‌های مختلفی از درخت‌ها را به عنوان معیار نزدیکی گراف‌های کلّی به درخت‌ها در نظر گرفت. در ادامه سه مورد از این ویژگی‌ها را به عنوان مثال در نظر می‌گیریم:

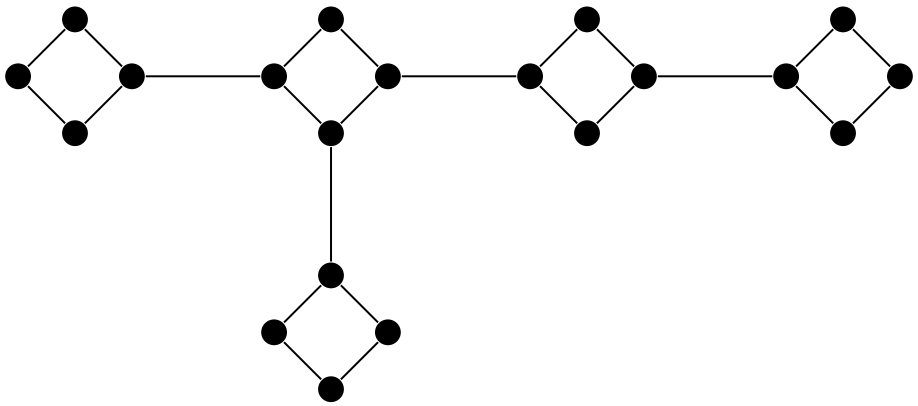
۱. محدود بودن تعداد دورها

با توجّه به این که یکی از ویژگی‌های بنیادین درخت‌ها نداشتن دور است؛ یک ایده برای ایجاد یک معیار برای شباهت گراف‌های کلّی به درخت‌ها می‌تواند کم بودن تعداد دورها باشد. به عنوان مثال، گراف شکل ۱ نمونه‌ای از چنین گراف‌هایی است. این گراف تنها سه دور دارد (دو دور بدون وتر و یک دور که از ادغام دو دور دیگر حاصل می‌شود) و سایر انشعابات آن فاقد دور می‌باشند. چنین گرافی را می‌توان تا حدّی شبیه به یک درخت دانست، طوری که با حذف دو یال از آن تبدیل به یک درخت می‌شود.



شکل ۱

با این حال این تعریف برای گرافی مانند شکل ۲ مناسب نیست؛ چرا که تعداد دورهای چنین گرافی می‌تواند از مرتبه‌ی تعداد رئوس آن باشد. با این حال این گراف ساختاری شبیه به درخت دارد؛ طوری که به جای هر کدام از رئوس درخت، می‌توان فرض کرد که یک دور به طول ۴ قرار گرفته است.



شکل ۲

۲. حذف تعداد ثابتی رأس منجر به ایجاد یک درخت شود

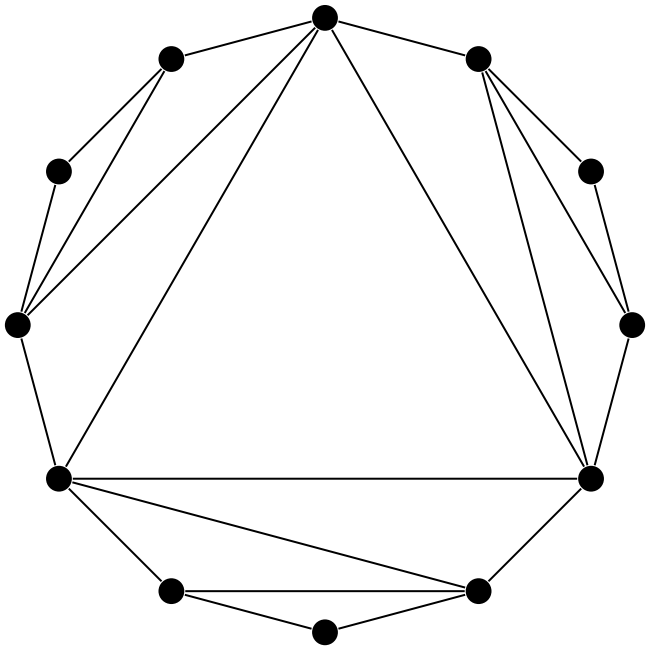
این ایده نیز می‌تواند ملاکی برای نزدیک بودن یک گراف به درخت باشد و می‌توان گرافی در نظر گرفت که از مرتبه‌ی تعداد رئوس خود دور دارد، اما با حذف یک رأس و یال‌های متّصل به آن تمامی این دورها از بین بروند. همچنین این ایده برای گراف شکل ۱ نیز مناسب است؛ امّا با این حال باز هم گراف شکل ۲ با این تعریف شباهتی به درخت نخواهد داشت.

۳. وجود تکّه‌های کوچکی که با یک ساختار درختی به هم متّصل شده باشند

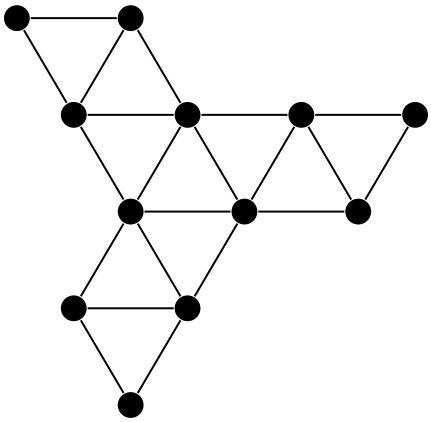
این تعریف علاوه بر این که به صورت شهودی مناسب است، شباهت ساختار شکل ۲ به درخت را نیز شامل می‌شود؛ با این حال نمی‌تواند شباهتی بین گراف شکل ۱ و درخت‌ها بیابد، چرا که در آن گراف، اگر چه تعداد دورها کم است امّا طول هر دور می‌تواند از مرتبه‌ی تعداد رئوس گراف باشد.

لازم به ذکر است مثال‌هایی از گراف‌های شبیه به درخت و نوع شباهت آن‌ها با درخت‌ها به نمونه‌های ذکرشده محدود نیستند. به عنوان

نمونه، گراف شکل ۳ را در نظر بگیرید. این گراف در نگاه اوّل شباهتی به یک درخت ندارد، با این حال می‌توان آن را به صورت شکل ۴ رسم کرد.



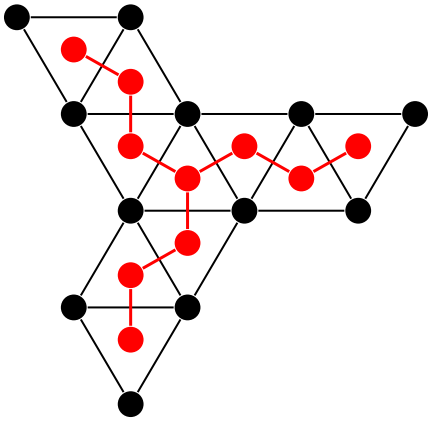
شکل ۳



شکل ۴

مشاهده می‌شود که گراف شکل ۴ با وجود آن که از منظر هیچ یک از ایده‌های سه‌گانه‌ای که برای شباهت گراف به درخت مطرح شد، شباهتی به یک درخت ندارد؛ امّا می‌توان با رویکرد دیگری آن را شبیه به یک درخت دانست. در واقع ساختار این گراف به گونه‌ای است که اگر به هر دور به طول ۳ از آن یک رأس نسبت دهیم و رئوس متناظر با دورهایی که در یک ضلع مجاور هستند را با یال‌هایی به هم متّصل کنیم، گراف جدیدی حاصل می‌شود که یک درخت خواهد بود. دقّت کنید که تنها دورهایی که در یک یال مشترک هستند مجاور در نظر گرفته می‌شوند و آن دسته از دورهایی که تنها در یک رأس مشترک باشند، با یکدیگر مجاور در نظر گرفته نخواهند شد. شکل ۵ این موضوع را به خوبی نشان می‌دهد. رئوس متناظر با دورهای به طول ۳ از گراف اصلی با رنگ قرمز نشان داده شده‌اند و اتّصالات آن‌ها مطابق با قاعده‌ی گفته‌شده با یال‌هایی از همان رنگ انجام شده است. به این ترتیب در این شکل مشاهده می‌شود که گراف قرمزرنگ حاصل، یک درخت است که ساختار کلّی گراف اوّلیه را توصیف می‌کند و نشان می‌دهد که آن گراف نیز از جهاتی شبیه به یک درخت بوده است.

به این ترتیب تا به این جا در مثال‌های متعدّدی شاهد گراف‌هایی بودیم که هر کدام از جهتی ساختاری مشابه با درخت‌ها داشتند. در ادامه می‌خواهیم تعریفی ارائه دهیم که بتواند به صورت گسترده‌ای، انواع و اقسام شباهت‌های گراف‌ها به درخت‌ها را در خود بگنجاند تا به



شکل ۵

کمک آن بتوانیم در صورت امکان، از وجود شباهت بین گراف‌های کلّی و درخت‌ها در حلّ مسائل سخت بهره ببریم.

۲ تجزیه‌ی درختی و عرض درختی یک گراف

تعریف ۱. *گراف $G = (V, E)$ را در نظر بگیرید. زوج $(T, \{V_t : t \in T\})$ را که در آن، $V_t \subseteq V$ها زیرمجموعه‌هایی از رئوس G هستند، یک* ***تجزیه‌ی درختی*** *برای G گوییم هر گاه سه خاصیت زیر برقرار باشند:*

۱. *هر رأس G حدّاقل در یکی از V_tها آمده باشد:*

$$\forall v \in V \ , \ \exists t \in T \ , \ v \in V_t$$

۲. *رئوس انتهایی هر یال از G در یکی از V_tها آمده باشند:*

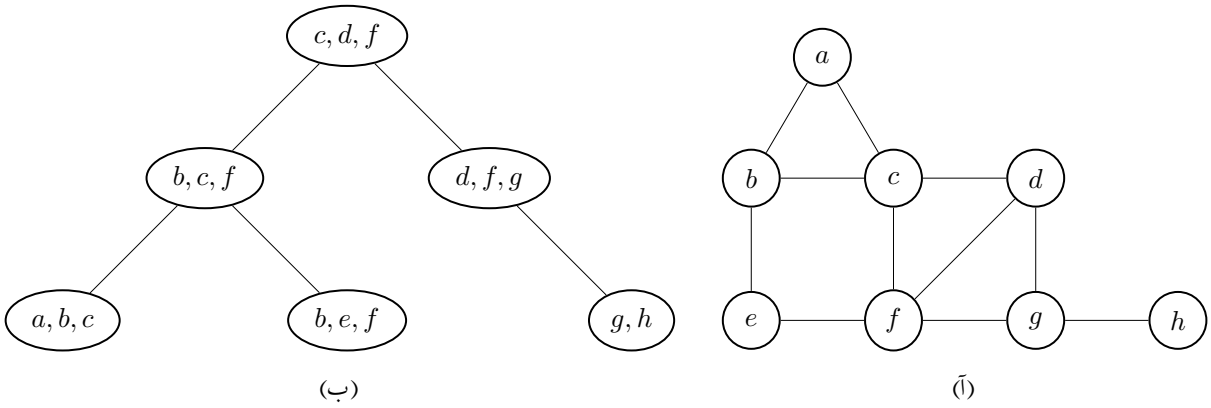
$$\forall uv \in E \ , \ \exists t \in T \ , \ \{u, v\} \subseteq V_t$$

۳. *مجموعه گره‌هایی از T که V_t متناظر با آن‌ها شامل رأسی از G مانند v باشد، تشکیل یک زیردرخت همبند از T بدهند.*

در تعریف فوق، خاصیت سوم از اهمّیت بالایی برخوردار است، به گونه‌ای که بسیاری از الگوریتم‌های مورد استفاده از این خاصیت بهره می‌برند. به صورت شهودی منظور از این خاصیت آن است که همه‌ی رئوسی از T که یک رأس دلخواه مانند $v \in G$ را در بر دارند (یا به عبارت دقیق‌تر، v در V_t آن‌ها موجود است) باید به هم راه داشته باشند و در کل، تشکیل یک زیردرخت همبند از T را بدهند. بیان دیگری از همین موضوع آن است که اگر رئوس $t_1 \neq t_2$ از T موجود باشند که $v \in V_{t_1}$ و $v \in V_{t_2}$، آنگاه در صورتی که رأس دیگری مانند $t_3 \in T$ وجود داشته باشد که در مسیر بین t_1 و t_2 در T حاضر باشد (مسیر بین t_1 و t_2 در درخت T را با نماد $t_1 T t_2$ نشان می‌دهیم)، باید داشته باشیم $v \in V_{t_3}$؛ یعنی

$$v \in V_{t_1} \ , \ v \in V_{t_2} \ , \ t_3 \in t_1 T t_2 \Rightarrow v \in V_{t_3}$$

به عنوان مثال، گراف موجود در شکل ۶آ را در نظر بگیرید. یک تجزیه‌ی درختی برای این گراف در شکل ۶ب مشاهده می‌شود. به سادگی می‌توان برقراری خواصّ سه‌گانه‌ی یک تجزیه‌ی درختی را برای این گراف تحقیق کرد.



شکل ۶

تا به این جا طبق تعریفی که برای یک تجزیه‌ی درختی ارائه کرده‌ایم، برای هر درختی حدّاقل یک تجزیه‌ی درختی بدیهی وجود دارد و آن، حالتی است که T را درختی با تنها یک رأس در نظر بگیریم، به گونه‌ای که V_t متناظر با آن همه‌ی رئوس گراف اوّلیه را شامل شود. بدیهی است که هر سه خاصیت یک تجزیه‌ی درختی برای T برقرار است (چرا که هر رأس و هر یال از گراف اوّلیه در یکی از رئوس T ظاهر شده‌اند و به‌علاوه، شرط همبندی نیز همواره برای درخت تک‌رأسی برقرار است). به این ترتیب به نظر می‌رسد که با این تعریف، وجود یک

تجزیه‌ی درختی از اهمیّت خاصّی برخوردار نیست. آن‌چه در عمل به نظر دارای اهمّیت است آن است که برای هر $t \in T$ از تجزیه‌ی درختی، اندازه‌ی V_t تا حدّ امکان کوچک باشد. این موضوع انگیزه‌ای برای ارائه‌ی تعاریفی نظیر عرض یک تجزیه‌ی درختی را فراهم می‌آورد.

تعریف ۲. ***عرض یک تجزیه‌ی درختی*** *مانند* $(T, \{V_t : t \in T\})$ *به صورت زیر تعریف می‌شود:*

$$\max_{t \in T} |V_t| - 1$$

به عنوان مثال، عرض تجزیه‌ی درختی موجود در شکل ۶ب برابر با ۲ است، چرا که اندازه‌ی بزرگ‌ترین V_t موجود در آن ۳ می‌باشد.

در ادامه تعریفی کلّی برای یک گراف بر مبنای تجزیه‌های درختی آن ارائه می‌دهیم:

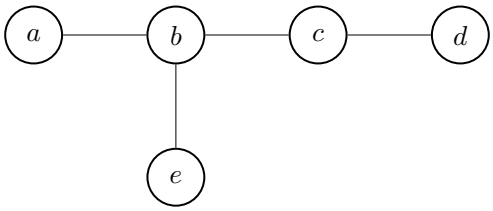
تعریف ۳. ***عرض درختی*** [1] ***یک گراف*** *G را کمینه‌ی عرض یک تجزیه‌ی درختی G تعریف می‌کنیم و آن را با $tw(G)$ نشان می‌دهیم.*

ادّعا می‌کنیم که کوچک بودن عرض درختی یک گراف، ملاکی از نزدیک بودن ساختار آن گراف به درخت است. قضیه‌ی زیر، یکی از جنبه‌های درستی این ادّعا را نشان می‌دهد. در ادامه نیز طرحی از اثبات این قضیه را ارائه کرده و دقیق‌تر کردن آن به خواننده واگذار می‌شود.

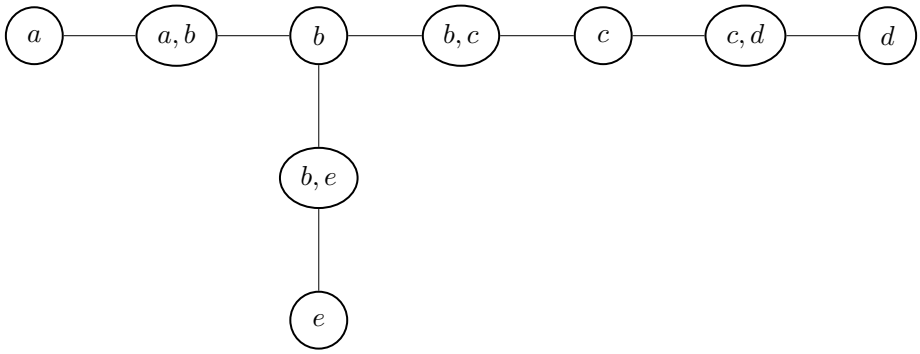
قضیه ۱. *عرض درختی گراف G برابر با یک است، اگر و تنها اگر G یک جنگل باشد.*

طرحی از اثبات. ابتدا فرض می‌کنیم یک درخت دلخواه داده شده باشد. باید نشان دهیم عرض درختی آن برابر با یک است؛ و این یعنی باید یک تجزیه‌ی درختی از آن بیابیم، طوری که اندازه‌ی تمامی مجموعه‌های V_t موجود در آن حدّاکثر برابر با ۲ باشد. برای این منظور، تجزیه‌ی درختی را ابتدا برابر با درخت اوّلیه قرار می‌دهیم؛ یعنی اگر v رأسی از درخت اوّلیه باشد، رأس متناظر با آن در تجزیه‌ی درختی را t_v نامیده و داریم $V_{t_v} = \{v\}$. در ادامه هر یال مانند $t_u t_v$ از تجزیه‌ی درختی را با مسیری به طول ۲ جایگزین می‌کنیم که اگر رأس میانی این مسیر را t_{uv} بنامیم، داریم $V_{t_{uv}} = \{u, v\}$.

به عنوان مثال، تجزیه‌ی درختی معرّفی‌شده در توضیحات فوق، برای درخت شکل ۷، درخت جدیدی مانند شکل ۸ خواهد بود.



شکل ۷



شکل ۸

[1] tree width

می‌توان به سادگی تحقیق کرد که خواصّ سه‌گانه‌ی یک تجزیه‌ی درختی برای روش ارائه‌شده برقرار است. به این ترتیب در این تجزیه‌ی درختی، V_tها حدّاکثر ۲ عضو دارند. از طرفی طبق خاصیت دوم تجزیه‌های درختی (به جز برای درخت تک‌عضوی) همواره باید حدّاقل یک V_t با اندازه‌ی حدّاقل ۲ (متناظر با یکی از یال‌های درخت) وجود داشته باشد؛ و به این ترتیب طبق تعریف، عرض درختی برای یک درخت (با حدّاقل دو رأس) برابر با یک خواهد بود. با برقراری حکم برای درخت‌ها، به وضوح برقراری آن برای جنگل‌ها نیز نتیجه می‌شود.

برای اثبات طرف دیگر حکم، باید نشان دهیم که اگر عرض درختی گرافی برابر با یک باشد، آن‌گاه آن گراف باید جنگل باشد. این فرض یعنی برای گراف داده‌شده یک تجزیه‌ی درختی موجود است که در آن، هر یک از V_tها اندازه‌ای حدّاکثر برابر با ۲ دارند. با برهان خلف فرض کنید چنین شرایطی برقرار است امّا گراف مربوطه جنگل نیست. بنابراین چنین گرافی باید یک دور داشته باشد. برای سادگی فرض کنید طول این دور برابر با ۴ باشد و رئوس موجود در آن، به ترتیب a و b و c و d نامیده شوند. رئوس متناظر با یال‌های ab و bc را در تجزیه‌ی درختی در نظر بگیرید و آن‌ها را t_{ab} و t_{bc} بنامید. چون تجزیه‌ی درختی خود یک درخت است، بین t_{ab} و t_{bc} مسیری یکتا وجود دارد. طبق خاصیت سوم تجزیه‌ی درختی، اگر t رأسی از این مسیر باشد، باید داشته باشیم $b \in V_t$. بنابراین نتیجه می‌شود که هیچ یک از رئوس این مسیر نمی‌توانند متناظر با یال‌های cd یا da از گراف اصلی باشند (چرا که V_tها حدّاکثر دو عضو دارند و در طول این مسیر، هر V_t باید شامل b باشد و نمی‌تواند در عین حال شامل c و d یا a و d باشد). به این ترتیب رأسی از تجزیه‌ی درختی که متناظر با یال cd است (مثلاً t_{cd}) باید جایی خارج از مسیر $t_{ab}Tt_{bc}$ باشد. با استدلال مشابه، t_{da} نیز نمی‌تواند عضو مسیرهای $t_{ab}Tt_{bc}$ یا $t_{bc}Tt_{cd}$ باشد. تا به این جا، مشاهده می‌شود که از اجتماع سه مسیر $t_{ab}Tt_{bc}$، $t_{bc}Tt_{cd}$، و $t_{cd}Tt_{da}$ یک مسیر از t_{ab} به t_{da} ایجاد شده است. چون T یک تجزیه‌ی درختی است، اگر t عضوی از این مسیر باشد باید داشته باشیم $a \in V_t$ (چرا که در V_tهای ابتدا و انتهای این مسیر، رأس a وجود دارد). از طرفی $V_{t_{bc}}$ در این مسیر موجود است که b, c عضو آن هستند. به این ترتیب باید داشته باشیم $\{a, b, c\} \subseteq V_{t_{bc}}$ که این یعنی $|V_{t_{bc}}| > 2$ و این با فرض اوّلیه در تناقض است؛ پس فرض خلف باطل است و گرافی که دور داشته باشد، نمی‌تواند عرض درختی برابر با ۱ داشته باشد.

به سادگی می‌توان توضیحات فوق را (که برای یک دور با طول ۴ ارائه شدند)، برای دور با طول دلخواه تعمیم داد و اثبات را کامل کرد.

□

در ادامه یک قضیه‌ی مهم دیگر در مورد تجزیه‌های درختی را بررسی می‌کنیم.

قضیه ۲. *فرض کنید G یک گراف دلخواه و $(T, \{V_t : t \in T\})$ یک تجزیه‌ی درختی متناظر با آن باشد. در این صورت برای هر $t \in T$، گراف $T - \{t\}$ یک جنگل است که مؤلّفه‌های همبندی (درخت‌ها) آن را با $T_1, T_2, \ldots, T_d$ نشان می‌دهیم. همچنین فرض کنید مؤلّفه‌ی متناظر با T_j در گراف G را با G_{T_j} نشان دهیم. به عبارت دقیق‌تر:*

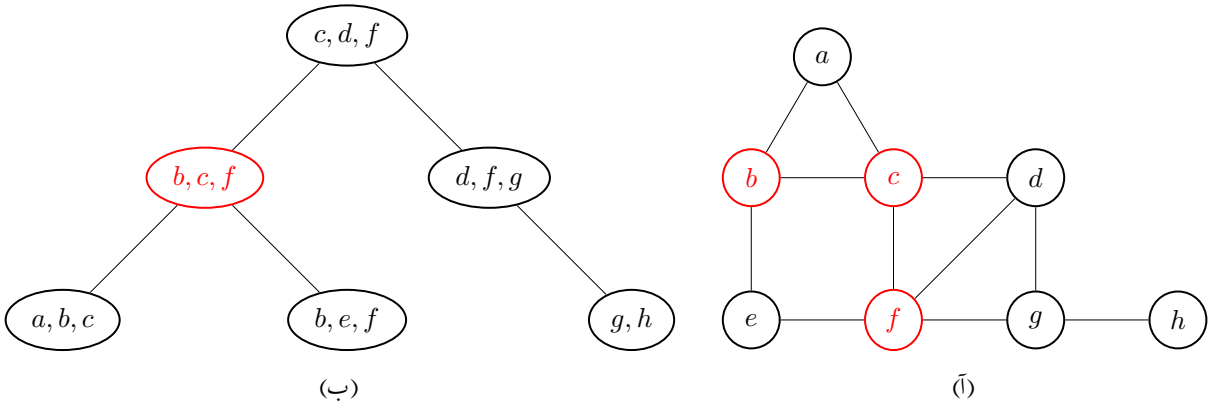
$$G_{T_j} = \bigcup_{\tau \in T_j} V_\tau$$

در این صورت، $G_{T_1} - V_t, \ldots, G_{T_d} - V_t$ رأس مشترکی ندارند و یالی هم بینشان نیست.

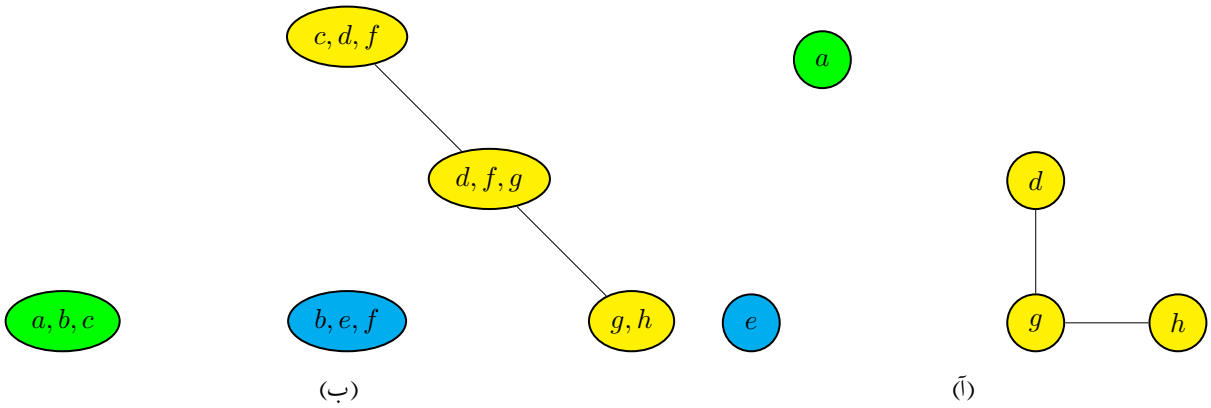
پیش از آن که در مورد علّت برقراری قضیه‌ی فوق استدلالی ارائه دهیم، درستی آن را در یک مثال تحقیق می‌کنیم. گراف شکل ۶آ و تجزیه‌ی درختی آن در شکل ۶ب را در نظر بگیرید. فرض کنید قرار دهیم. اگر t را رأسی از تجزیه‌ی درختی را که متناظر با رئوس b و c و f از گراف اصلی است در نظر بگیریم، آن‌گاه با حذف آن درخت T به جنگلی با سه درخت تبدیل خواهد شد. رأس t و رئوس متناظر با آن در گراف اصلی در شکل ۲۳ با رنگ قرمز مشخّص شده‌اند.

شکل ۱۰ب تجزیه‌ی درختی را پس از حذف رأس t نشان می‌دهد و شکل ۱۰آ نیز گراف اصلی را پس از حذف V_t نشان می‌دهد. مشاهده می‌شود که G_{T_j}ها (که با رنگ‌ها متمایز نشان داده شده‌اند) هیچ رأس مشترکی ندارند و علاوه بر آن، هیچ یالی نیز بین آن‌ها وجود ندارد.

در ادامه اثبات قضیه ۲ را بیان می‌کنیم.



شکل ۹



شکل ۱۰

اثبات. ابتدا تعریف می‌کنیم $G_i = G_{T_i} - V_t$. باید نشان دهیم که G_iهای متمایز رأس مشترک ندارند و یالی نیز بین آن‌ها موجود نیست. ابتدا با برهان خلف فرض کنید G_i و G_j موجود هستند که $i \neq j$ و رأسی مانند u از گراف اصلی پیدا شده که $u \in G_i \cap G_j$. در این صورت رئوسی مانند τ و τ' از تجزیه‌ی درختی موجود هستند، طوری که $\tau \in T_i$ و $\tau' \in T_j$ و $u \in V_\tau \cap V_{\tau'}$. حال با توجّه به خاصیت سوم از تجزیه‌های درختی، هر رأسی از T که در مسیر بین τ و τ' باشد، در مجموعه‌ی متناظر خود باید شامل رأس u باشد. حال چون T_i و T_j زیردرخت‌هایی بودند که تنها از طریق رأس t به هم مرتبط بودند، t قطعاً عضوی از مسیر بین τ و τ' بوده و باید داشته باشیم $u \in V_t$ و این تناقض است؛ چرا که فرض کرده بودیم $u \in G_{T_i} - V_t$. بنابراین فرض خلف باطل است و امکان ندارد G_iها رأس مشترکی داشته باشند.

برای اثبات بخش دیگر قضیه، یعنی عدم وجود یال بین G_i و G_j متمایز در گراف اصلی، مجدّداً از برهان خلف استفاده می‌کنیم. فرض کنید $u \in G_i$ و $v \in G_j$ موجودند که $i \neq j$ و یال uv بین این دو رأس موجود است. طبق خاصیت دوم تجزیه‌ی درختی، می‌دانیم $t_{uv} \in T$ باید موجود باشد که $\{u, v\} \subseteq V_{t_{uv}}$. از طرفی رئوس $t_u \in T$ و $t_v \in T_j$ نیز موجودند که $u \in V_{t_u}$ و $v \in V_{t_v}$. حال اگر $t_{uv} \in T_i$ باشد، آنگاه با در نظر گرفتن مسیر بین t_{uv} و t_v، مشاهده می‌کنیم که رأس t عضوی از این مسیر است و باید داشته باشیم $v \in V_t$ که تناقض است. تناقض مشابهی برای حالتی که $t_{uv} \in T_j$ باشد نیز حاصل می‌شود. تنها حالت باقی‌مانده آن است که $t_{uv} \in T_k$ که $k \notin \{i, j\}$. در آن صورت نیز t عضوی از مسیر بین t_{uv} و t_u می‌باشد که این یعنی $u \in V_t$ و مجدّداً تناقض مشابه حاصل می‌شود. به این ترتیب وجود یال uv در گراف اصلی در هر حالت منجر به تناقض می‌شود؛ بنابراین چنین یالی وجود نخواهد داشت. □

بنابراین قضیه ۲ به صورت شهودی بیان می‌کند که بخش‌های مجزّایی که از حذف یک رأس از تجزیه‌ی درختی مانند t حاصل می‌شوند؛ به صورت متناظر در گراف اصلی نیز مجزّا خواهند بود و تنها راه ارتباطی آن‌ها از طریق رئوس متناظر با t بوده است. قضیه‌ی مشابهی در

مورد حذف یک یال از تجزیه‌ی درختی نیز وجود دارد که در ادامه بیان می‌شود و اثبات آن به خواننده واگذار می‌شود.

قضیه ۳. *فرض کنید G یک گراف دلخواه و $(T, \{V_t : t \in T\})$ یک تجزیه‌ی درختی متناظر با آن باشد. همچنین فرض کنید x و y دو رأس متّصل‌به‌هم از T باشند. در این صورت حذف مجموعه‌ی $V_x \cap V_y$ از G، گراف G را به دو مؤلّفه‌ی $G_x - (V_x \cap V_y)$ و $G_y - (V_x \cap V_y)$ تقسیم می‌کند که رأس مشترک ندارند و یالی نیز بین آن‌ها نیست.*

همچنین قضیه‌ی جالب دیگری نیز در مورد تجزیه‌های درختی قابل بیان است که در ادامه ذکر می‌شود و اثبات آن به کمک *استقرا* به خواننده واگذار می‌شود.

قضیه ۴. *فرض کنید G یک گراف دلخواه و $(T, \{V_t : t \in T\})$ یک تجزیه‌ی درختی متناظر با آن باشد. در این صورت اگر X یک خوشه[۲] از G باشد (منظور از یک خوشه از یک گراف بدون جهت، مجموعه‌ای از رئوس است که همگی با هم مجاور باشند)، آن‌گاه کلّ X باید در یک گره از T آمده باشد.*

$$\exists t \in T\ ,\ X \subseteq V_t$$

در واقع قضیه ۴ برای خوشه‌هایی با اندازه‌ی ۱ و ۲ همان شروط اوّل و دوم از تعریف تجزیه‌ی درختی است؛ و برای خوشه‌ها با اندازه‌های بیشتر نیز با استقرا قابل اثبات است.

پیش از به پایان بردن این بخش، یک خاصیت مهمّ دیگر از عرض درختی گراف را بیان می‌کنیم و اثبات آن را به عنوان تمرین به خواننده واگذار می‌کنیم.

گزاره ۱. *اگر هر یال دلخواه از یک گراف را حذف یا منقبض کنیم، عرض درختی گراف زیاد نمی‌شود.*

[2] clique

۳ تعاریف دیگری برای شباهت گراف‌ها به درخت

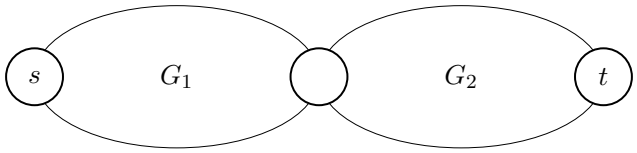
۱.۳ گراف سری–موازی

پیش از تعریف تجزیه‌ی درختی (تعریف ۱)، تعاریف دیگری نیز برای شباهت گراف‌ها به درخت‌ها ارائه شده بودند. نمونه‌ای از این موارد، گراف‌های سری–موازی هستند.

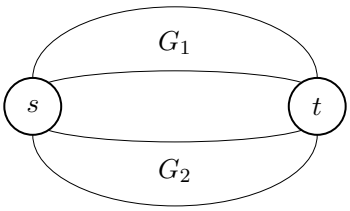
تعریف ۴. *گراف G یک **گراف سری–موازی** است، هر گاه G دارای دو رأس پایانه‌ای[3] s و t باشد و در صورتی که بیش از دو رأس دارد، از اتّصال سری یا موازی دو گراف سری–موازی دیگر ایجاد شده باشد.*

دقّت کنید که تعریف فوق، یک تعریف استقرایی است؛ یعنی هر گراف سری–موازی با بیش از دو رأس به صورت ترکیب سری یا موازی دو گراف سری–موازی کوچک‌تر بیان شده است. منظور از ترکیب موازی دو گراف آن است که دو رأس s و دو رأس t مربوط به دو گراف، با یکدیگر ادغام شوند و رئوس s و t گراف جدید را پدید آورند. همچنین منظور از ترکیب سری دو گراف آن است که رأس s یکی با رأس t دیگری ادغام شده، و دو رأس پایانه‌ای ادغام‌نشده به عنوان s و t گراف جدید در نظر گرفته شوند. شماتیکی از اتّصال‌های سری و موازی دو گراف در شکل‌های ۱۱، ۱۲، و ۱۳ مشاهده می‌شود.

شکل ۱۱: دو گراف سری–موازی کلّی



شکل ۱۲: اتّصال سری گراف‌های شکل ۱۱



شکل ۱۳: اتّصال موازی گراف‌های شکل ۱۱

همان طور که پیش‌تر بیان شد، گراف سری–موازی یکی از نمونه‌های موجود از تعاریفی است که شباهت یک گراف کلّی را با درخت می‌سنجند. قضیه ۵ این ارتباط را روشن‌تر می‌کند:

قضیه ۵. *عرض درختی گراف G کوچک‌تر یا مساوی ۲ است، اگر و تنها اگر G زیرگرافی از یک گراف سری–موازی باشد.*

با وجود ارائه‌شدن تعاریف متعدّد، می‌توان مشاهده کرد که تعریف عرض درختی یک گراف به نوعی یک تعریف طبیعی است؛ طوری که افراد و گروه‌های مختلفی به صورت مستقل از یک‌دیگر به چنین تعریفی برای بیان میزان شباهت گراف به درخت دست یافته‌اند. این

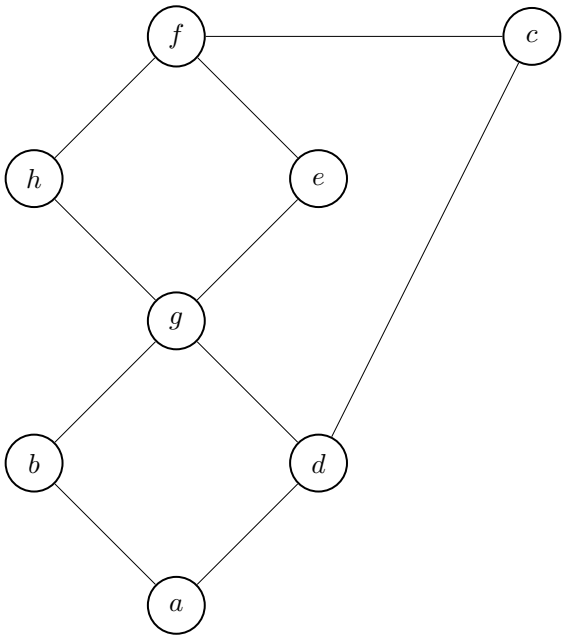
[3] terminal vertices

تعریف را می‌توان در حالات خاص‌تری مانند مسیر نیز بیان کرد (که در آن، همه‌ی تعاریف مشابه است و تنها تفاوت آن است که به جای کار کردن با تجزیه‌های درختی، با تجزیه‌های مسیری سر و کار داریم. در این حالات، سایر خواصّ یک تجزیه‌ی مسیری مشابه با یک تجزیه‌ی درختی است و به کمک آن‌ها، می‌توان مفاهیمی نظیر عرض مسیری یک گراف را تعریف کرد). علاوه بر این موضوع، تعاریف عرض و تجزیه‌ی درختی به صورت‌های معادل دیگری نیز قابل بیان هستند که در ادامه، مثال‌هایی از این صورت‌های معادل را بررسی خواهیم کرد.

۲.۳ گراف‌های وتری

تعریف ۵. *گوییم گراف G یک **گراف وتری**[۴] است، هر گاه G دور القایی با طول بزرگ‌تر یا مساوی ۴ نداشته باشد.*

به عبارت دیگر، یک گراف وتری است هر گاه هر دوری از آن که طولی بزرگ‌تر یا مساوی ۴ داشته باشد، حتماً دارای وتر باشد (و منظور از وتر یک دور، یالی از گراف است که بین دو رأس غیرمجاور درون آن دور موجود باشد). به عنوان مثال، گراف شکل ۱۴ وتری نیست، چرا که دورهای القایی با طول ۴ یا بیشتر دارد. برای آن که این گراف وتری شود، لازم است تعدادی یال به آن اضافه کنیم. گراف شکل ۱۵ با اضافه کردن سه یال، گراف شکل ۱۴ را به یک گراف وتری تبدیل کرده است (دقّت کنید که در شکل ۱۵، دور $fegd$ دارای وتر fg است امّا در ترسیم، این وتر خارج از دور مشاهده می‌شود).



شکل ۱۴: نمونه‌ای از یک گراف غیروتری

حال می‌توان عرض درختی را با کمک گراف‌های وتری تعریف کرد. قضیه ۶ چنین امکانی را فراهم می‌کند.

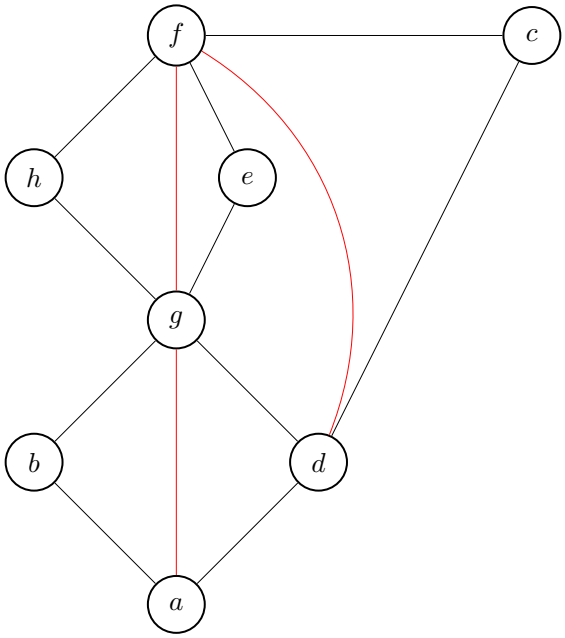
قضیه ۶. *اگر عرض درختی گراف G را با $tw(G)$ نشان دهیم، داریم:*

$$tw(G) \leq k \iff \exists \text{ chordal } G' \,,\, G \subseteq G' \,,\, \omega(G') \leq k+1$$

که منظور از $\omega(G')$، اندازه‌ی بزرگ‌ترین خوشه‌ی گراف G' است.

قضیه‌ی فوق بیان می‌دارد که اگر G زیرمجموعه‌ای از یک گراف وتری مثل G' باشد که اندازه‌ی بزرگ‌ترین خوشه‌ی آن حدّاکثر $k+1$ است؛ آن‌گاه عرض درختی G بیشتر از k نخواهد بود.

[۴] chordal graph



شکل ۱۵: تبدیل گراف شکل ۱۴ به یک گراف وتری با اضافه کردن سه وتر

همچنین قضایایی در مورد وتری بودن یک گراف نیز وجود دارند که در ادامه به دو مورد از آن‌ها اشاره می‌کنیم:

قضیه ۷. *گراف G وتری است، اگر و تنها اگر G یک تجزیه‌ی درختی داشته باشد که در آن، هر انبان[۵] (هر کدام از V_tها) یک خوشه باشد.*

قضیه ۸. *گراف G وتری است، اگر و تنها اگر G یک ترتیب حذفی تام[۶] داشته باشد که منظور از یک ترتیب حذفی تام روی گراف، ترتیبی روی رئوس گراف است که برای هر رأس v، v و همسایگانی از v که در ترتیب بعد از v قرار می‌گیرند تشکیل یک خوشه بدهند.*

۳.۳ تعریف عرض درختی بر مبنای بازی دزد و پلیس

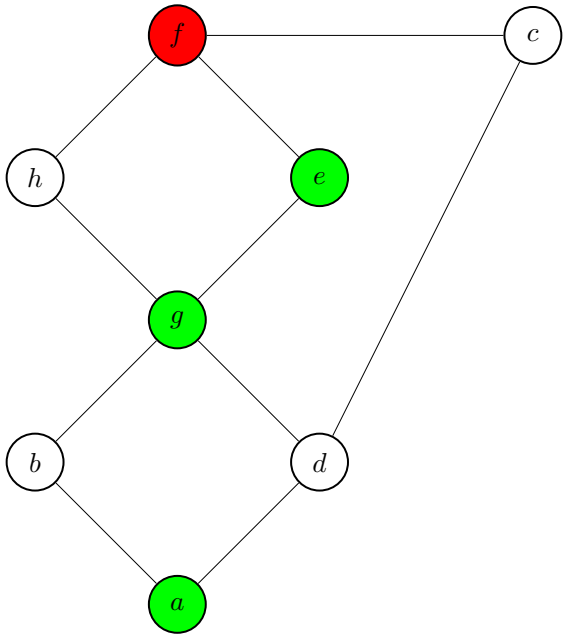
ابتدا بازی دزد و پلیس را بر روی یک گراف تعریف می‌کنیم. فرض کنید k پلیس و یک دزد وجود دارند که در ابتدای بازی، هر کدام در یکی از رئوس گراف داده‌شده قرار دارند (پلیس‌ها و دزد، هر کدام می‌توانند محلّ شروع خود را تعیین کنند). می‌توان فرض کرد پلیس‌ها مجهّز به بالگرد[۷]هایی هستند که به آن‌ها این امکان را می‌دهد تا در یک واحد زمانی، از رأس فعلی خود بلند شده و به رأس دیگری پرواز کنند و در آن فرود بیایند. به عبارت دیگر، در طول یک واحد زمانی، هر پلیس می‌تواند از محلّ فعلی خود بدون محدودیت به هر رأس دیگری برود و این کار شامل فرآیند پرواز از مبدأ به سمت مقصد می‌باشد. از طرف دیگر دزد به چنین امکانی مجهّز نیست و تنها می‌تواند بر روی یال‌های گراف بدود و خود را از رأسی به رأس دیگر برساند. فرض می‌کنیم سرعت دویدن دزد بسیار بالاست و در هر لحظه می‌تواند خود را از طریق یال‌ها بین رئوس جابه‌جا کند. تنها محدودیت دزد آن است که نمی‌تواند از رئوسی که پلیس در آن‌ها حاضر است عبور کند (چرا که در آن حالت دستگیر می‌شود). در این بازی هدف پلیس‌ها آن است که دزد را دستگیر کنند، و منظور از دستگیری آن است که در لحظه‌ای، یکی از پلیس‌ها در رأسی فرود بیاید که دزد در آن رأس قرار دارد. در این بازی فرض می‌شود که استراتژی دزد همواره بهینه است؛ بنابراین امکان دستگیری دزد وجود ندارد، مگر آن که دزد در رأسی قرار گرفته باشد که تمامی راه‌های خروج آن توسّط پلیس‌ها مسدود شده باشد. در این حالت، پلیس دیگری با بالگرد خود به محلّ دزد پرواز می‌کند و او را دستگیر می‌کند. دقّت کنید که این پلیس باید با پلیس‌هایی که راه‌های فرار دزد را مسدود کرده‌اند متفاوت باشد، چرا که اگر یکی از آن‌ها از محلّ خود به سمت دزد پرواز کند، یکی از راه‌های فرار باز می‌شود و دزد که سرعت دویدن بی‌نهایت دارد، پیش از آن که دستگیر شود فرار می‌کند (در واقع این نکته در بازی مهم است که سرعت حرکت دزد

[5] bag
[6] perfect elimination ordering
[7] helicopter

بی‌نهایت است، امّا سرعت حرکت پلیس‌ها از یک رأس به رأس دیگر بی‌نهایت نیست و در حدّ فاصل جابه‌جایی یک پلیس، دزد می‌تواند در صورت لزوم حرکت آن را مشاهده کند و فرار کند).

به عنوان مثال، واضح است که یک پلیس هیچ‌گاه به تنهایی نمی‌تواند دزد را دستگیر کند چرا که به محض آن که پلیس به سمت یک رأس پرواز کند، دزد خود را در محلّی به جز آن رأس قرار می‌دهد. می‌توانید مثال ساده‌ی این حالت را برای یک گراف با دو رأس و یک یال بررسی کنید که یک پلیس هیچ‌گاه نمی‌تواند دزد را دستگیرد کند و همواره دزد و پلیس جای خود را روی گراف با یک‌دیگر عوض می‌کنند.

به عنوان نمونه، گراف شکل ۱۶ را به عنوان حالت اوّلیه‌ی این بازی با سه پلیس (رئوس سبز) و یک دزد (رأس قرمز) در نظر بگیرید.



شکل ۱۶: حالت اوّلیه از یک بازی دزد و پلیس با ۳ پلیس

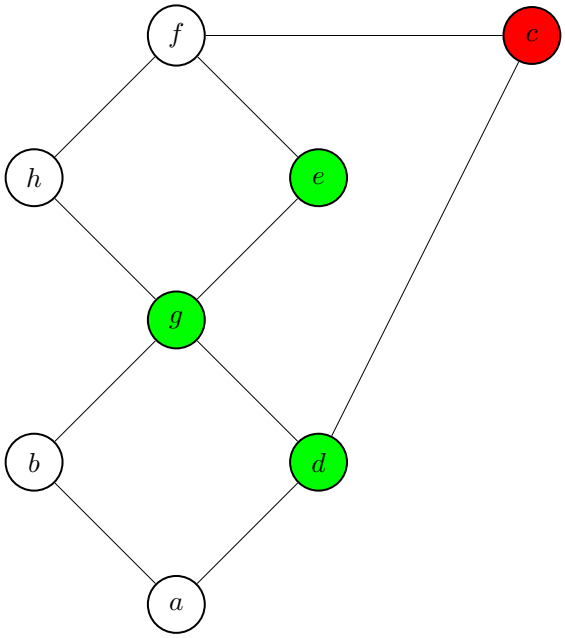
فرض کنید با گذر یک واحد زمانی، یکی از پلیس‌ها از رأس a به d رفته و سایر پلیس‌ها بدون حرکت باقی می‌مانند. در این میان، دزد می‌تواند در f باقی بماند و یا از طریق راه‌هایی که توسّط پلیس‌ها مسدود نیست به یکی از رئوس h، c، a، یا b برود. (دقّت کنید که زمانی که پلیس در مسیر بین a و d است، دزد می‌تواند از این دو رأس عبور کند). مثلاً فرض کنید دزد از f به c برود. در این حالت شرایط بازی به صورت شکل ۱۷ در می‌آید.

در گام بعدی پلیس حاضر در رأس e می‌تواند خود را به f برساند و در این زمان، دزد می‌تواند در c باقی بماند یا خود را به یکی از دو رأس e و h برساند. مثلاً فرض کنید دزد خود را به e برساند. در این حالت ساختار بازی به صورت شکل ۱۸ در می‌آید.

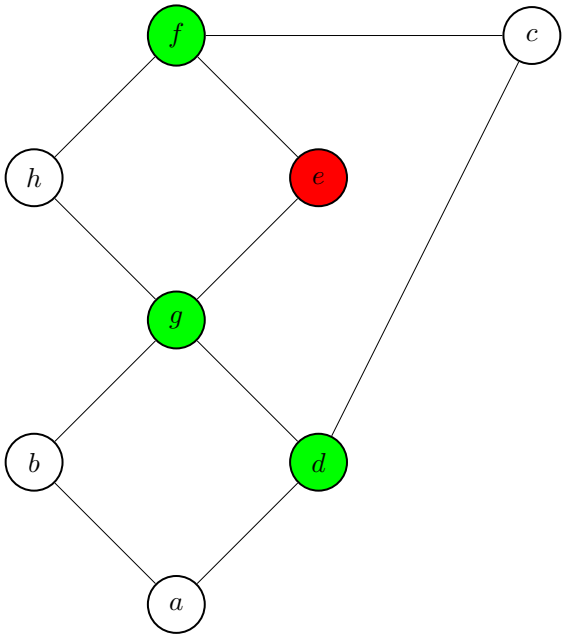
در این حالت مشاهده می‌شود که دزد بین دو پلیس حاضر در خانه‌های f و g محصور شده است و هیچ راهی برای فرار ندارد. حال کافی است پلیس حاضر در d از محلّ خود بلند شود و در e فرود بیاید و دزد را دستگیر کند.

به این ترتیب یک نمونه از اجرای بازی دزد و پلیس را مشاهده کردیم. واضح است که چگونگی تعیین استراتژی توسّط دزد و پلیس به ساختار گراف بستگی دارد. به صورت کلّی مسأله‌ی مورد نظر این است که روی یک گراف داده‌شده، حدّاقل چه تعداد پلیس نیاز است تا حتماً یک استراتژی برای دستگیری دزد وجود داشته باشد (با فرض آن که دزد از استراتژی بهینه استفاده کند و بهترین عملکرد ممکن را برای فرار داشته باشد). قضیه ۹ پاسخی به این پرسش ارائه می‌کند.

قضیه ۹. *اگر بازی دزد و پلیس بر روی گراف G انجام شود، عرض درختی این گراف کوچک‌تر یا مساوی k است، اگر و تنها اگر استراتژی‌ای موجود باشد که $k+1$ پلیس حتماً بتوانند در بازی موفّق شوند و دزد را دستگیر کنند.*



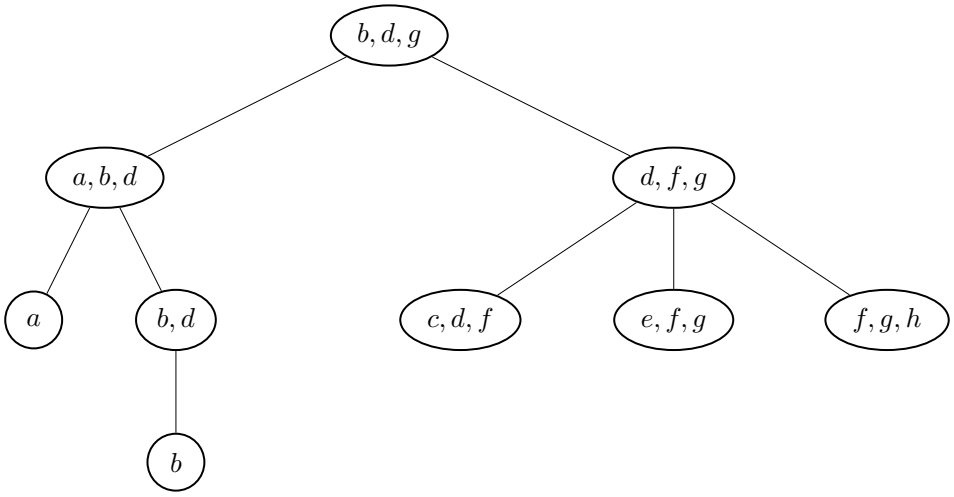
شکل ۱۷: وضعیت بازی دزد و پلیس با گذشتن یک واحد زمانی



شکل ۱۸: وضعیت بازی دزد و پلیس با گذشتن دو واحد زمانی

به عنوان نمونه می‌توان مشاهده کرد که درخت شکل ۱۹ یک تجزیه‌ی درختی برای گراف موجود در شکل ۱۷ است. با توجّه به این که این گراف درخت نیست، عرض درختی آن حدّاقل برابر با ۲ است. از طرفی مشاهده می‌شود که عرض تجزیه‌ی درختی موجود در شکل ۱۹ نیز برابر با ۲ است، بنابراین عرض درختی گراف شکل ۱۷ دقیقاً برابر با ۲ خواهد بود. در نتیجه طبق قضیه ۹ کافی است ۳ پلیس بر روی آن گراف قرار گیرند تا حتماً امکان دستگیری دزد وجود داشته باشد و این نشان می‌دهد مثالی که مورد بررسی قرار دادیم، در هر حالت دیگری (یعنی انتخاب‌های دیگری توسّط دزد برای حرکت کردن) نیز با انتخاب یک استراتژی مناسب توسّط پلیس‌ها منجر به دستگیری دزد می‌شد.

نکته‌ی دیگری که با استفاده از قضیه ۹ می‌توان بیان کرد آن است که اگر گراف بازی دزد و پلیس یک درخت باشد، آنگاه دو پلیس حتماً می‌توانند دزد را دستگیر کنند. استراتژی مورد استفاده برای پلیس‌ها در این حالت ساده است. یکی از پلیس‌ها روی ریشه‌ی درخت



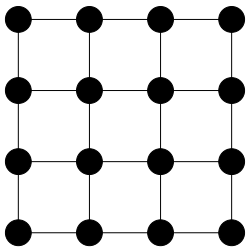
شکل ۱۹

قرار می‌گیرد. در این حالت، دزد در یکی از زیردرخت‌های منشعب از ریشه قرار دارد و تا زمانی که پلیس حاضر در ریشه از جای خود بلند نشود، دزد در همان زیردرخت محصور مانده است. حال پلیس دیگر می‌تواند خود را به ریشه‌ی آن زیردرخت (که یکی از فرزندان ریشه‌ی درخت اصلی است) برساند. به این صورت، دزد در یک درخت کوچک‌تر محصور شده و می‌توان هر بار به صورت استقرایی با همین الگو، زیردرختی که دزد در آن قرار دارد را کوچک‌تر کرد تا نهایتاً در یک رأس دستگیر شود.

همین الگو برای حالت کلّی قضیه ۹ نیز برقرار است. در واقع اگر فرض کنیم تجزیه‌ی درختی گراف مسأله را در اختیار داشته باشیم، می‌توان استراتژی مشابهی برای دستگیری دزد ارائه کرد. به عنوان مثال، تجزیه‌ی درختی موجود در شکل ۱۹ را در نظر بگیرید. یک استراتژی می‌تواند آن باشد که ابتدا پلیس‌ها در ریشه‌ی این درخت، یعنی در رئوس b و d و g قرار بگیرند و بررسی کنند که دزد در کدام‌یک از زیردرخت‌هاست. به عنوان مثال فرض کنید دزد در زیردرخت سمت راست باشد. ضمناً توجّه کنید که طبق قضیه ۲، این دو زیردرخت در گراف اصلی با هم اشتراکی (به جز در b یا d یا g) ندارند. حال کافی است پلیس‌ها به رأس dfg بروند؛ یعنی دو پلیسی که در رئوس b و g قرار داشتند محلّ خود را تغییر ندهند و پلیسی که در رأس b قرار داشت خود را به f برساند. به این ترتیب با توجّه به جدا از هم بودن رئوس متناظر زیردرخت‌ها در گراف اصلی، دزد نمی‌تواند در این زمان خود را از زیردرخت سمت راست خارج کند. حال مسأله به صورت استقرایی بر روی درختی که ریشه‌ی آن رأس dfg است تکرار می‌شود. کافی است پلیس‌ها بررسی کنند که دزد در کدام‌یک از زیردرخت‌های سه‌گانه وجود دارد و در ادامه آن را دستگیر کنند.

چنین روشی با داشتن هر تجزیه‌ی درختی با عرض k، این امکان را می‌دهد که دزد توسّط $k+1$ پلیس دستگیر شود و عملاً با دقیق‌تر کردن همین بیان، اثبات یکی از طرفین قضیه ۹ نیز انجام می‌شود. اثبات طرف دیگر این قضیه دشوارتر است.

با پذیرفتن قضیه ۹، می‌توان نتایج دیگری نیز به دست آورد. یک نمونه از آن، بررسی شبکه‌های $k \times k$ است. به عنوان مثال، شکل ۲۰ یک شبکه‌ی $k \times k$ را به ازای $k = 4$ نشان می‌دهد.



شکل ۲۰

ادّعا می‌کنیم عرض درختی چنین شبکه‌ای حدّاقل برابر با $k-1$ است. این بدان معناست که طبق قضیه ۹، اگر بازی دزد و پلیس روی چنین شبکه‌ای صورت بگیرد، برای دستگیری دزد حدّاقل به k پلیس نیاز است. می‌توان به سادگی مشاهده کرد که چنین ادّعایی درست است. در واقع اگر تعداد پلیس‌ها کم‌تر از k باشد، می‌توان مشاهده کرد که در هر لحظه از زمان، حدّاقل یک سطر یا یک ستون از جدول خالی است و این باعث می‌شود که دزد استراتژی‌ای داشته باشد که هیچ‌گاه دستگیر نشود.

به این ترتیب قضیه ۹ این نتیجه را حاصل می‌کند که عرض درختی شبکه‌ی $k \times k$ حدّاقل برابر با $k-1$ است. البته می‌توان به صورت کلّی نشان داد که عرض درختی این شبکه دقیقاً برابر با k است، امّا این اثبات به استدلالی فراتر از تنها به کار بردن قضیه ۹ نیاز دارد.

به صورت کلّی نیز چنین شبکه‌هایی مثال‌های جالبی در بحث عرض درختی هستند. در واقع گراف‌هایی هستند با n رأس که بیشترین درجه‌ی رئوس موجود در آن‌ها برابر ۴ است و در عین حال، عرض درختی آن‌ها $\sqrt{n}$ است. قضایایی وجود دارند که نشان می‌دهند وجود چنین شبکه‌هایی مانع پایین آمدن عرض درختی یک گراف می‌شود، به گونه‌ای که زیاد بودن عرض درختی یک گراف را حتماً می‌توان به وجود یک ساختار شبکه‌ای در آن نسبت داد.

۴ یافتن تجزیه‌ی درختی برای یک گراف

فرض کنید عرض درختی یک گراف $tw(G) = t$ باشد. سوال ما این است که در چه زمانی می‌توانیم یک تجزیه‌ی درختی با عرض t برای گراف پیدا کنیم. اولا خوب است دقت داشته باشیم که پیدا کردن عرض درختی یک گراف یک مساله‌ی ان‌پی–سخت است. بنابراین الگوریتمی که پیدا می‌کنیم یک ارتباط نمایی با پارامتر t دارد. البته این مساله یک مساله‌ی مهارشدنی با پارامتر ثابت[۸] برحسب t است، بدین معنا که الگوریتم‌هایی وجود دارد که فقط برحسب t نمایی هستند و برحسب n یا همان تعداد رئوس چندجمله‌ای هستند.

به عنوان مثال یک الگوریتمی که در سال 1996 توسط آقای بُدلَندر [۹] ارائه شد، در زمان $2^{\tilde{O}(t^3)}n$ یک تجزیه‌ی درختی با عرض درختی t می‌یابد. در نتیجه به ازاء یک t ثابت، این الگوریتم خطی می‌باشد و از جهت تئوری یک نتیجه‌ی بسیار مطلوب است. اما دقت داریم که ضریب ثابت این الگوریتم به ازاء مقادیر کوچک t حتی $t = 3$ بسیار بزرگ می‌شود.

نکته اینجاست که الگوریتم‌های بهتر در زمان‌های کوتاه‌تری وجود دارند که یک تجزیه‌ی درختی برمی‌گردانند که عرض درختی آن‌ها کمینه نیست. در حقیقت اگر ما به دنبال یک تجزیه‌ی درختی خوب باشیم لزومی ندارد که همیشه تجزیه درختی با کمینه عرض درختی را داشته باشیم. بسته به کارکرد و الگوریتمی که می‌خواهیم از تجزیه‌ی درختی در آن استفاده کنیم، می‌توان از تجزیه‌های درختی با عرض درختی بیشتر از حالت مینیمم استفاده کنیم. مثلا الگوریتمی وجود دارد که در زمان $2^{O(t)}n$ یک تجزیه‌ی درختی با عرض $5t + 4$ برمی‌گرداند، با فرض اینکه عرض درختی گراف اولیه t باشد. حتی الگوریتمی با زمان $\mathsf{poly}(n)$ وجود دارد که یک تجزیه‌ی درختی با عرض $O(t \log(t))$ برمی‌گرداند. پس، از این به بعد فرض می‌کنیم که برای یک گراف داده شده تجزیه‌ی درختی مناسب برای آن را داریم و دیگر بررسی نمی‌کنیم این تجزیه‌ی درختی از کجا آمده است.

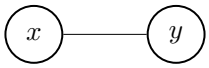
[8] fixed-parameter tractable

[9] bodlaender

۵ حل برخی مسائل بر روی گراف‌های با عرض درختی کم

می‌توان نشان داد که بسیاری از مسائل انپی–سخت مانند یافتن مجموعه‌ی مستقل، کلیک، مسائل رنگ‌پذیری و یافتن دور همیلتونی را می‌توان در زمان‌های کارایی برای گراف‌های با عرض درختی کم حل کرد. حتی یک سری مسائل که به صورت مستقیم به گراف مربوط نیستند را می‌توان با استفاده از ساخت یک گراف متناظر با آن مساله در زمان چندجمله‌ای (اگر عرض درختی گراف ساخته شده به حد کافی کوچک باشد) حل کرد.

به عنوان مثال مساله‌ی 3-SAT یا انواع دیگر مساله‌های SAT یا در حالت کلی‌تر مسائلی که تحت عنوان ارضاپذیری قیود[۱۰] مطرح می‌شوند را می‌توان با استفاده از روش ذکر شده حل کرد. مساله‌ی 3-SAT را در نظر بگیرید. از روی فرمول متناظر با مساله، می‌توان یک گراف اولیه‌ی [۱۱] فرمول ساخت. این گراف بدین نحو ساخته می‌شود که برای هر دو متغیر x و y یا نقیض آن‌ها که به هر شکل در یک *عبارت* از فرمول ظاهر شده اند، دو گره با نام‌های x و y مانند شکل زیر در گراف اولیه قرار می‌دهیم و بین آن‌ها یک یال می‌گذاریم.



شکل ۲۱

در گراف اولیه بسیاری از اطلاعات فرمول از بین می‌رود. به عنوان مثال هر تعداد بار خود متغیرهای x و y یا نقیض آن‌ها در حالت‌های مختلف در یک عبارت از یک فرمول ظاهر شوند، ما تنها دو راس با همان نام‌ها (حالت بدون نقیضشان) با یک یال بین‌شان مانند شکل ۲۱ قرار می‌دهیم. حال اگر درخت اولیه‌ی ساخته شده از روی فرمول دارای عرض درختی ثابت و کوچکی باشد، می‌توان نشان داد که فرمول 3-SAT متناظر با آن در زمان چندجمله‌ای قابل حل است. البته این مساله یک مساله‌ی مهارشدنی با پارامتر ثابت[۱۲] نیست یعنی عرض درختی در نمای n (تعداد راس‌ها) ظاهر می‌شود. اما به هر حال به ازای یک عرض درختی ثابت، نشان داده می‌شود که این چنین مسائل 3-SAT را می‌توان در زمان چندجمله‌ای حل کرد. برای حل چنین فرمول‌هایی از همان ابزار ذکر شده در گراف‌های وتری با نام ترتیب حذفی تام استفاده می‌شود که ما به ذکر جزئیات آن نمی‌پردازیم.

یک مطلب جالب دیگر که در حد اشاره به آن می‌پردازیم و جزئیات آن را بررسی نمی‌کنیم، یک قضیه‌ی مهم در منطق مرتبه‌ی دوم تکین[۱۳] یا MSO2 است. بیان کلی این قضیه این است که می‌گوید هر خاصیتی از گراف‌ها را که بتوان با این زبان منطقی (MSO2) نشان داد، برحسب عرض درختی یک مساله‌ی مهارشدنی با پارامتر ثابت است. منطق مرتبه‌ی دوم تکین یک زبان منطقی خاصی است که در منطق گراف‌ها کاربرد زیادی دارد. علت اصلی کاربرد آن نیز همین قضیه‌ای است که بیان کلی آن ذکر شد. به عنوان مثال عبارت منطقی زیر در MSO2 را در نظر بگیرید.

$$\exists\, C \subseteq V \quad \forall\, v \in C \quad \exists\, u_1, u_2 \in C \;:\; u_1 \neq u_2 \;\wedge\; \mathrm{adj}(u_1, v) \;\wedge\; \mathrm{adj}(u_2, v)$$

در عبارت منطقی فوق V مجموعه‌ی تمام رئوس است و تابع $\mathrm{adj} : V \times V \to \{T, F\}$ یک تابع تعریف شده در این منطق است که دو راس از گراف را به عنوان ورودی می‌گیرد و در صورتی که یالی بین آن‌ها باشد (همسایه باشند)، خروجی آن True و در غیر این صورت False برمی‌گرداند. به صورت کلی در یک گزاره‌ی منطقی که می‌خواهد در MSO2 بیان شود، می‌توانیم سورهای وجودی و عمومی روی هر زیرمجموعه‌ی دلخواه از رئوس یا یال‌ها داشته باشیم. همچنین توابعی به فرم $P : X \to \{\mathsf{True}, \mathsf{False}\}$ را که X یک زیرمجموعه از رئوس یا یال‌ها باشد، می‌توانیم در یک گزاره از این منطق استفاده کنیم. معمولا این توابع تحت عنوان predicate شناخته می‌شوند. گزاره‌ای که در فوق نوشته شده هنگامی True برمی‌گرداند که یک زیرمجموعه از رئوس با درجه‌ی حداقل ۲ یافت شوند، یا به تعبیر دیگر

[10] constraint satisfaction
[11] primal graph
[12] fixed-parameter tractable
[13] monadic second-order logic

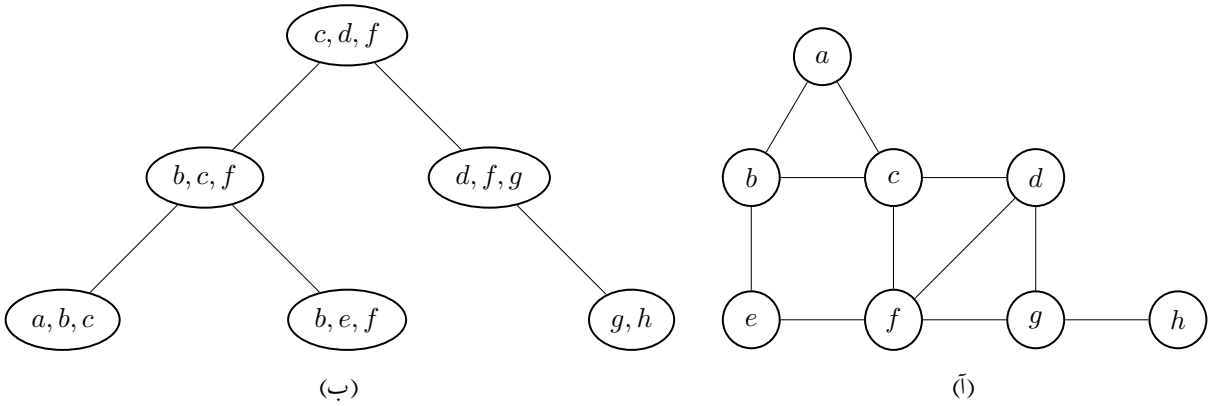
هنگامی که گراف شامل دور باشد. در ادامه صورت دقیق قضیه‌ای که بیان کلی آن در جلوتر آمد را ذکر می‌کنیم. این قضیه تحت عنوان قضیه‌ی کورسِل[۱۴] شناخته می‌شود.

قضیه ۱۰. ***قضیه‌ی کورسِل:*** *فرض کنید F یک فرمول در $MSO2$ باشد. برای هر t عدد ثابت $K_{t,F}$ وجود دارد به طوری که می‌توان در زمان $O(K_{t,F}n)$ برای گراف‌های با عرض درختی حداکثر t درست بودن فرمول F را تصمیم گرفت.*

این قضیه‌ی بسیار کلی است و در نتیجه می‌توان برای بسیاری از مسائل به دنبال یک الگوریتم با پارامتر ثابت بر حسب عرض درختی بود. به عنوان مثال مسائلی مانند رنگ‌پذیری و دور همیلتونی در قالب MSO2 قابل بیان‌اند و درنتیجه می‌توان برای آن‌ها الگوریتمی با پارامتر ثابت بر حسب عرض درختی یافت. البته در اکثر موارد ثابت $K_{t,F}$ وابستگی بسیار شدیدی به طول فرمول و عدد t دارد و اگر طول فرمول و عدد t اندکی افزایش یابند، این ثابت بسیار بزرگ می‌شود.

۱.۵ حل مسأله‌ی زیرمجموعه‌ی مستقل با داشتن تجزیه‌ی درختی یک گراف

گراف شکل ۲۲آ و تجزیه‌ی درختی متناظر با آن در شکل ۲۲ب را در نظر بگیرید. این گراف را در مثال‌های قبل‌تر نیز بررسی کرده بودیم. می‌خواهیم مساله‌ی یافتن زیرمجموعه‌ی مستقل را روی این گراف به کمک تجزیه‌ی درختی آن و با ابزار برنامه‌ریزی پویا حل کنیم.



(ب) (آ)

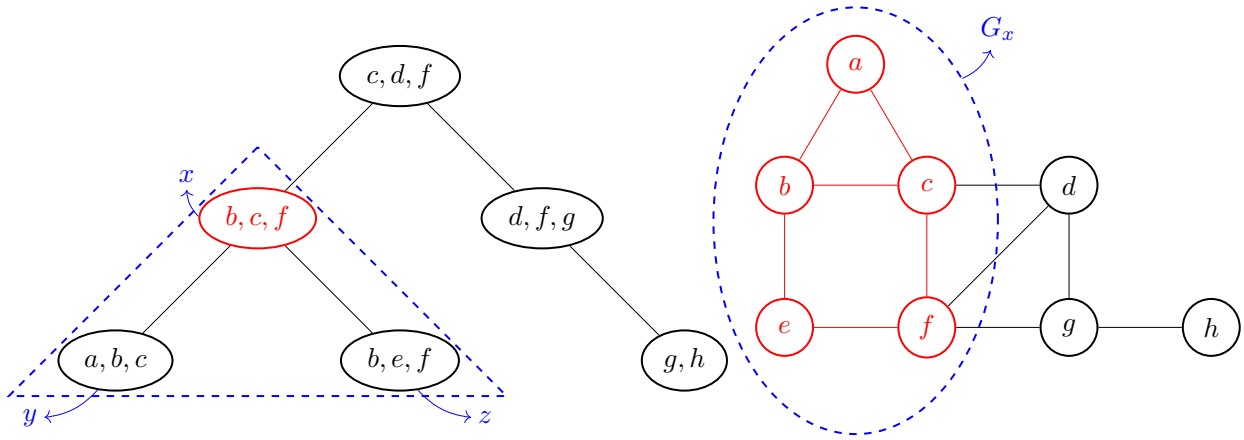
شکل ۲۲

اگر در خاطر داشته باشیم ما مسأله‌ی بزرگ‌ترین زیرمجموعه‌ی مستقل را برای درخت‌ها در جلسات مربوط به برنامه‌ریزی پویا حل کردیم. در حل این مسأله برای درخت‌ها، برای هر گره مانند x دو زیرمساله تعریف کردیم. یکی اندازه‌ی بزرگ‌ترین زیرمجموعه‌ی مستقل در زیردرختی که x ریشه‌ی آن است و شامل x است و یکی اندازه‌ی بزرگ‌ترین زیرمجموعه‌ی مستقل در زیردرختی که x ریشه‌ی آن است و شامل خود x نیست. در ادامه مشاهده کردیم که از روی حل این زیرمسائل برای فرزندان یک گره، می‌توانیم پاسخ این دو زیر مسأله را برای خود آن گره بیابیم. برای گراف‌هایی که تجزیه‌ی درختی آن‌ها را داریم نیز می‌توانیم یک الگوریتم بسیار مشابه الگوریتمی که بر روی درخت‌ها داشتیم اجرا کنیم. برای یک گره مانند x در تجزیه‌ی درختی و یک زیر مجموعه از رئوس متناظر با آن گره یعنی $S \subseteq V_x$ زیرمسأله‌ی $M[x, S]$ را به صورت زیر تعریف می‌کنیم.

$$M[x, S] : \text{سایز بزرگ‌ترین مجموعه‌ی مستقل مانند } I \text{ در } G_x \text{ که } I \cap V_x = S$$

که در آن G_x زیرگراف القایی از گراف اصلی با مجموعه‌ی همه‌ی رئوسی است که در زیردرخت با ریشه‌ی x در تجزیه درختی وجود دارند. به عنوان مثال گره‌ی x را گره‌ی به رنگ قرمز در شکل ۲۳ب در نظر بگیرید. گراف G_x به عنوان زیرگراف القایی ناشی از گره‌ی x در گراف اصلی در شکل ۲۳آ با رنگ قرمز مشخص شده است.

[14] Courcelle's theorem



(آ) (ب)

شکل ۲۳

حال می‌خواهیم زیرمسأله‌ی تعریف شده برای یک گره را برحسب پاسخ زیرمسأله‌های فرزندانش محاسبه کنیم. برای این منظور می‌توان نوشت

$$M[x,S] = \sum_{c\in\mathcal{C}(x)} \max\ M[c,S'] \rightarrow \boxed{\text{به ازا هر } S'\subseteq V_c \text{ که } S \text{ و } S' \text{ سازگار هستند}}$$

که در رابطه‌ی بالا $\mathcal{C}(x)$ مجموعه‌ی همه‌ی فرزندان گره x در تجزیه‌ی درختی است و سازگاری S و S' بدین معناست که هر رأسی که در S حضور دارد حتما در S' هم حضور داشته باشد و هم‌چنین S' شامل هیچ رأس از مجموعه‌ی $V_x \setminus S$ نباشد چرا که طبق تعریف، $M[x,S]$ اندازه‌ی بزرگ‌ترین زیرمجموعه‌ی مستقلی است که از مجموعه رئوس V_x شامل همه‌ی رئوس داخل S است و هیچ یک از رئوس $V_x \setminus S$ را شامل نمی‌شود.

برای جا افتادن مطلب، این رابطه را برای گره‌ی x نشان داده شده در شکل ۲۳ب و $S = b \subseteq V_x$ می‌نویسیم. مطابق شکل فرض کنید که گره‌های y و z به ترتیب فرزندان چپ و راست گره‌ی x هستند. خواهیم داشت

$$M[x,\{b\}] = \max_{S_1\subseteq V_y} M[y,S_1] + \max_{S_2\subseteq V_z} M[z,S_2]$$

به شرط آنکه

$$b \in S_1 \quad , \quad c \notin S_1$$
$$b \in S_1 \quad , \quad f \notin S_1$$

که شرایط فوق همان شرایط سازگاری S با S_1 و S_2 است. چون $S = b$ و $V_x \setminus S = \{c,f\}$، طبق توضیحات فوق S_1 و S_2 حتما باید شامل b باشند و هم‌چنین نباید در آن‌ها هیچ یک از رئوس $\{c,f\}$ دیده شود.

برای محاسبه‌ی زمان اجرا کافی است ما تعداد زیر مساله‌ها و زمان حل هر زیر مساله را به دست آوریم. تعداد زیر مساله‌ها برابر است با تعداد گره‌های تجزیه‌ی درختی ضرب در بزرگ‌ترین تعداد حالات ممکن برای مجموعه‌ی S که $2^{tw(G)+1}$ است. چرا که S زیرمجموعه‌ای دلخواه از V_x است و چون بیشترین مقدار ممکن برای اندازه‌ی مجموعه‌ی V_x همان $tw(G)+1$ است، تعداد حالات مختلف برای یک زیرمجموعه‌ی دلخواه از آن برابر با $2^{tw(G)+1}$ می‌شود. هم‌چنین زمان حل هر زیرمسأله نیز برحسب عرض درختی یک زمان نمایی است، چرا که باید همه‌ی زیرمجموعه‌های ممکن و سازگار از انبان گره‌های فرزند نیز بررسی شوند و ماکزیمم آن‌ها انتخاب شود. در نهایت مشاهده می‌کنیم که زمان حل این مسأله یک زمان نمایی برحسب عرض درختی ضرب در یک زمان چندجمله‌ای برحسب n است که همان زمان مطلوبی است که به دنبال آن هستیم.

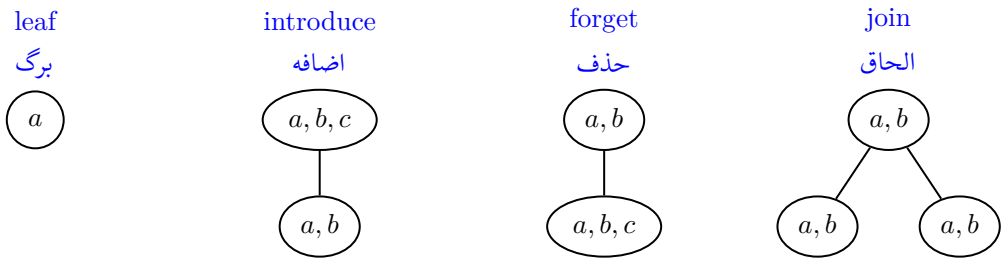
همچنین در حل این مسأله می‌توان مشاهده کرد که خاصیت سوم ذکر شده برای یک تجزیه‌ی درختی یک خاصیت مهم و بنیادین است. به عنوان مثال همچنان گراف شکل ۲۳آ و تجزیه‌ی درختی آن در شکل ۲۳ب و گره‌ی مشخص شده‌ی x روی آن و دو فرزند این گره یعنی y و z را در نظر بگیرید. دقت کنید که زیردرخت‌هایی از تجزیه درختی که y و z ریشه‌های آن هستند، با یکدیگر هیچ اشتراکی در رئوس متناظرشان در گراف اصلی ندارند مگر آن رئوسی که در V_x یعنی $\{b, c, f\}$ نیز آمده است. اگر چنین نبود و یک راس دیگری مانند u در این دو زیر درخت با ریشه‌های y و z یافت می‌شد در حل زیرمسأله‌ها به صورت بازگشتی دچار مشکل می‌شدیم. چرا که گاهی ممکن بود u را به عنوان یک راس در مجموعه‌ی S_1 متناظر با y بیاوریم و برعکس در مجموعه‌ی S_2 متناظر با z نیاوریم و در نتیجه نمی‌توانستیم همزمان u را داخل و خارج مجموعه‌ی مستقل داشته باشیم و برای این کار نیاز به حالت‌بندی‌های زیادی روی چنین رئوسی می‌شد که عملا زمان ما را به یک زمان نمایی برحسب تعداد رئوس تبدیل می‌کرد. در کل نیز نکته در تجزیه‌ی درختی همین جاست که بگوییم اشتراک گره‌های داخل انبان‌های زیردرخت‌های به ریشه‌ی y و z محدود و ثابت است و همگی در انبان گره‌ی پدرشان یعنی x حضور دارند و درنتیجه مجموعه‌ی S متناظر با x تعیین می‌کند که کدام یک از رئوس باید در مجموعه‌ی S' متناظر با هر یک از فرزندان حتما باشند و کدام‌ها نباید باشند و بقیه‌ی رئوسی که در انتخاب آن‌ها برای زیرمجموعه‌ی مستقل آزاد هستیم دارای قید دیگری نخواهند بود.

نکته‌ی آخر این است که برای این که بتوانیم رابطه‌های بازگشتی خود را به فرم بهتر و راحت‌تری بنویسیم، خوب است که فرض کنیم تجزیه‌ی درختی ما دارای خواص خوبی است. این خواص به ما کمک می‌کنند تا در برنامه‌ریزی‌های پویایی که در حل مسائل با آن‌ها مواجه می‌شویم، بتوانیم راحت‌تر روابط بازگشتی را بنویسیم. چنین تجزیه‌های درختی‌ای را در قسمت بعد توضیح می‌دهیم.

۲.۵ تجزیه‌ی درختی خوب

تعریف ۶. *یک تجزیه‌ی درختی خوب*[15] *یک درخت دودویی ریشه‌دار است که شامل چهار نوع گره می‌باشد.*

- ***گره‌های برگ***[16] *که شامل هیچ فرزندی نیستند و اندازه‌ی انبان آن‌ها ۱ است.*
- ***گره‌های اضافه***[17] *که تنها شامل یک فرزند هستند. انبان فرزند شامل همان رئوس داخل انبان پدر می‌باشد به غیر از یک راس که نسبت به رئوس داخل انبان پدر حذف می‌شود.*
- ***گره‌های حذف***[18] *که تنها شامل یک فرزند هستند. انبان فرزند شامل همان رئوس داخل انبان پدر می‌باشد به غیر از یک راس که نسبت به رئوس داخل انبان پدر اضافه می‌شود.*
- ***گره‌های الحاق***[19] *که شامل دو فرزند هستند. انبان هر دو فرزند کاملا مشابه انبان پدر می‌باشد.*



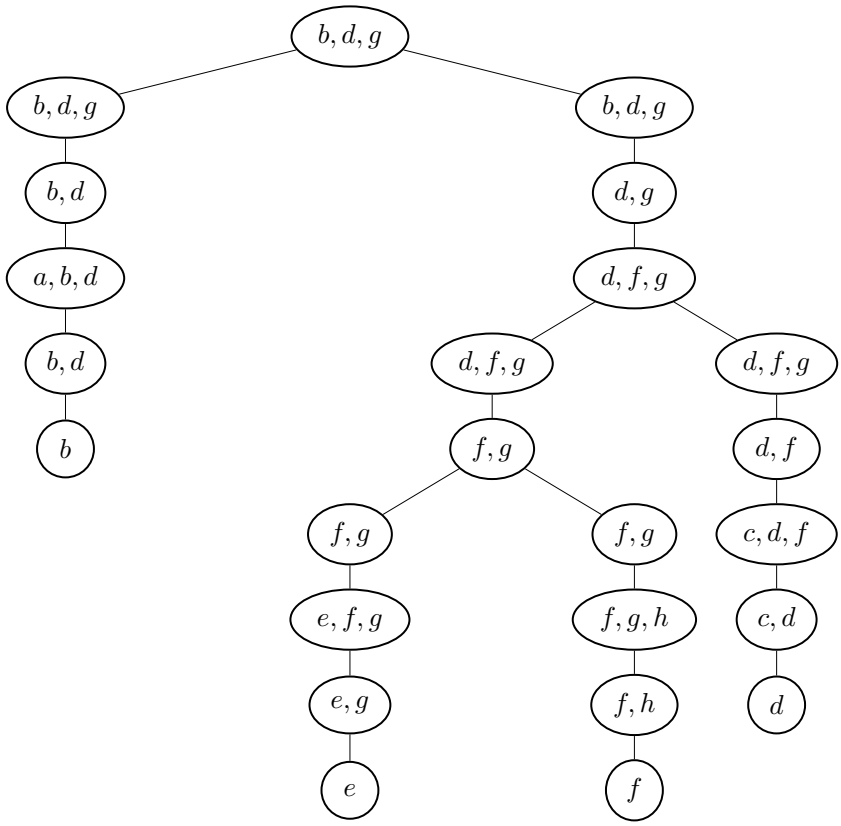
شکل ۲۴: انواع گره‌ها در یک تجزیه‌ی درختی خوب

[15] nice tree decomposition
[16] leaf
[17] introduce
[18] forget
[19] join

در مورد تجزیه‌ی درختی خوب می‌توان قضیه‌ی زیر را ثابت کرد.

قضیه ۱۱. *اگر $tw(G) = k$ باشد، آنگاه G حتما یک تجزیه‌ی درختی خوب دارد. هم‌چنین اگر T یک تجزیه‌ی درختی با عرض k باشد، در زمان $O\left(k^2 \max(|V(T)|, |V(G)|)\right)$ می‌توان یک تجزیه‌ی درختی خوب از G با عرض k و با $O(k|V(G)|)$ رأس یافت.*

پس ما مطمئن می‌شویم که همواره می‌توان یک تجزیه‌ی درختی خوب با $tw(G)$ برای G یافت. حال برای نمونه گراف شکل ۱۴ را در نظر بگیرید. ما برای این گراف یک تجزیه‌ی درختی غیر خوب در شکل ۱۹ دیدیم. اما حال می‌خواهیم برای آن یک تجزیه درختی خوب نشان دهیم. به شکل ۲۵ دقت کنید. این شکل یک تجزیه‌ی درختی خوب برای گراف شکل ۱۹ است.



شکل ۲۵: یک تجزیه‌ی درختی خوب برای گراف شکل ۱۹

لفظ "خوب" در تجزیه‌ی درختیِ خوب بدین معنا نیست که تعداد گره‌ها خیلی کم باشد یا تجزیه‌ی درختی ظاهر جالبی داشته باشد، بلکه همان‌طور که از شکل ۲۵ پیداست ممکن است یک تجزیه با تعداد گره‌های زیاد و درختی عریض و طویل باشد اما نکته‌ی مفید در این درخت‌ها آن است که اجرا کردن الگوریتم‌های برنامه‌ریزی پویا بر روی آن‌ها آسان‌تر است.

گاهی در تعریف تجزیه‌ی درختی خوب در منابع دیگر، می‌گویند که باید برگ‌ها یا ریشه‌ی درخت تهی باشند. به عنوان مثال در شکل ۲۵ باید به برگ‌های انتهایی هر کدام یک برگ جدید و تهی اضافه کرد و به بالای درخت در محل ریشه نیز به ترتیب سه گره اضافه کرد که بتوانیم طبق قوانین تجزیه‌ی درختی خوب، اعضای داخل انبان ریشه را یکی یکی حذف کنیم تا در انتها به یک ریشه‌ی تهی برسیم.

یک نکته‌ی دیگر که خوب است به آن توجه کنیم، این است که در یک تجزیه‌ی درختی خوب یک رأس حداکثر یک بار حذف می‌شود ولی ممکن است چند بار اضافه شود. می‌توانید به اثبات این موضوع نیز فکر کنید. شمای کلی اثبات به کمک خاصیت سوم در تعریف تجزیه‌های درختی است. به کمک این خاصیت می‌توانید نشان دهید که اگر یک رأس باشد که دو بار حذف شده باشد، قسمت‌های مربوط به آن ناهم‌بند می‌شود.

۳.۵ حل مسأله‌ی سه رنگ‌پذیری به کمک تجزیه‌ی درختی خوب

فرض کنید گراف G و یک تجزیه‌ی درختی خوب از آن به ما داده شده است. می‌خواهیم به کمک این تجزیه‌ی درختی خوب داده شده مسأله‌ی سه رنگ‌پذیری را با برنامه‌ریزی پویا بر روی تجزیه‌ی درختی حل کنیم. ابتدا زیر مسأله‌ی خود را این‌گونه تعریف می‌کنیم که برای یک گره‌ی x در تجزیه‌ی درختی و یک تابع رنگ‌آمیزی روی رئوس داخل انبان آن مانند $c : V_x \to \{0, 1, 2\}$ متغیر بولی $D[x, c]$ را True تعریف می‌کنیم اگر و تنها اگر c را بتوان به یک رنگ‌آمیزی معتبر برای G_x گسترش داد. (برای یادآوری G_x زیرگراف القایی از گراف اصلی با مجموعه‌ی همه‌ی رئوسی است که در زیردرخت با ریشه‌ی x در تجزیه درختی وجود دارند.) حال می‌خواهیم بدانیم آیا $D[r, c]$ به ازای یک رنگ‌آمیزی c روی V_r که در آن r ریشه‌ی تجزیه‌ی درختی است، برابر با True می‌شود یا نه.

ما می‌توانیم $D[x, c]$ را به صورت بازگشتی و با توجه به انواع گره‌های موجود در تجزیه‌ی درختی خوب به صورت زیر محاسبه کنیم.

- اگر x یک گره‌ی برگ باشد: هر نوع رنگ‌آمیزی معتبر می‌باشد. به تعبیری $D[x, c]$ به‌ازای همه‌ی حالات c مقدار True را اختیار می‌کند.
- اگر x یک گره‌ی اضافه باشد: در این حالت فرض کنید نام گره‌ی فرزند y باشد و گره‌ی پدر نسبت به گره‌ی فرزند، رأس u را بیشتر داشته باشد. ما می‌خواهیم رأس u را به گونه‌ای رنگ کنیم که هیچ یک از رنگ‌آمیزی‌های قبلی خراب نشود. به وضوح $D[x, c]$ مقدار false را اختیار می‌کند، اگر رنگ u با رنگ یکی از همسایه‌هایش در V_x یکسان شود. در غیر این حالات می‌توان نوشت $D[x, c] = D[y, c|_{V_y}]$ که $c|_{V_y}$ همان رنگ‌آمیزی c است که دامنه‌ی آن به V_y محدود شده است.
- اگر x یک گره‌ی حذف باشد: در این حالت فرض کنید نام گره‌ی فرزند y باشد واین بار گره‌ی فرزند نسبت به گره‌ی پدر، رأس u را بیشتر داشته باشد. برای اینکه ببینیم آیا می‌توان از یک رنگ‌آمیزی روی گره‌ی y می‌توان به یک رنگ‌آمیزی برای گره‌ی x رسید، باید روی رنگ رأس u حالت‌بندی داشته باشیم. اگر به ازای یکی از این حالات نیز ما موفق به گسترش رنگ‌آمیزی شدیم، پس گسترش رنگ‌آمیزی به رأس x موفقیت‌آمیز بوده است. به بیان ریاضی می‌توان نوشت

$$D[x, c] = \vee_{i=0}^{2} D[y, c'] \qquad \begin{cases} c'(v) = c(v) & v \in V_x \\ c'(v) = i & \text{o.w.} \end{cases}$$

- اگر x یک گره‌ی الحاق باشد: در این حالت فرض کنید که نام فرزندان x، y و z باشد. در این صورت رنگ‌آمیزی باید بر روی هر دو فرزند معتبر باشد تا رنگ‌آمیزی بر روی گره‌ی پدر یعنی x نیز معتبر باشد. به بیان ریاضی می‌توانیم بنویسیم

$$D[x, c] = D[y, c] \wedge D[z, c]$$

۶ تجزیه و عرض درختی در گراف‌های مسطح

به علت کاربردهای زیادی که گراف‌های مسطح دارند، علاقه‌مندیم که عرض درختی روی چنین گراف‌هایی کم باشد. در قسمت ۳.۳ یک مثال از یک گراف مسطح دیدیم که در شکل ۲۰ ترسیم شده است. با پذیرفتن قضیه‌ی دزد و پلیس یا همان قضیه‌ی ۹، نشان داده شد در حالت کلی نیز گراف‌های به صورت یک شبکه‌ی $m \times m$ دارای عرض درختی m هستند. حال می‌خواهیم یک قضیه‌ی جالب و کلی‌تر زیر را مطرح کنیم.

قضیه ۱۲. *عرض درختی هر گراف مسطح n رأسی از مرتبه‌ی $O(\sqrt{n})$ است.*

بلافاصله به کمک این قضیه و قضیه‌ی ۱۰ (یا همان قضیه‌ی کورسل که بیان می‌کرد بسیاری از مسائلی که در منطق مرتبه‌ی دوم تکین روی گراف‌ها قابل بیان هستند بر حسب عرض درختی یک مساله‌ی مهار شدنی با پارامتر ثابت‌اند) می‌توان بیان کرد که برای خیلی از مسائل NP-Hard روی گراف‌های مسطح الگوریتم با زمان اجرای $O(2^{O(\sqrt{n})}\text{poly}(n))$ وجود دارد. در نتیجه در این گونه گراف‌ها ما دارای یک سری الگوریتم زیرنمایی[20] هستیم. یعنی با اینکه زمان اجرای ما چندجمله‌ای نیست اما از هر زمان اجرای نمایی به فرم c^n کم‌تر خواهد بود و در گراف‌های کلی ما چنین الگوریتم‌هایی با این زمان اجرا نداریم.

در ادامه یک قضیه را می‌آوریم که به کمک آن می‌توان یک الگوریتم تقریبی برای حل یکی از مسائل NP-Hard روی گراف‌های مسطح ارائه نمود. این قضیه، به قضیه‌ی اشتاین[21] معروف است.

قضیه ۱۳. *اگر G یک گراف مسطح با قطر d باشد، در این صورت عرض درختی آن حداکثر $3d-2$ خواهد بود و در نتیجه خواهیم داشت $tw(G) = O(d)$.*

۱.۶ یک الگوریتم تقریبی با زمان چندجمله‌ای برای حل مسأله‌ی بزرگ‌ترین زیرمجموعه‌ی مستقل بر روی گراف‌های مسطح

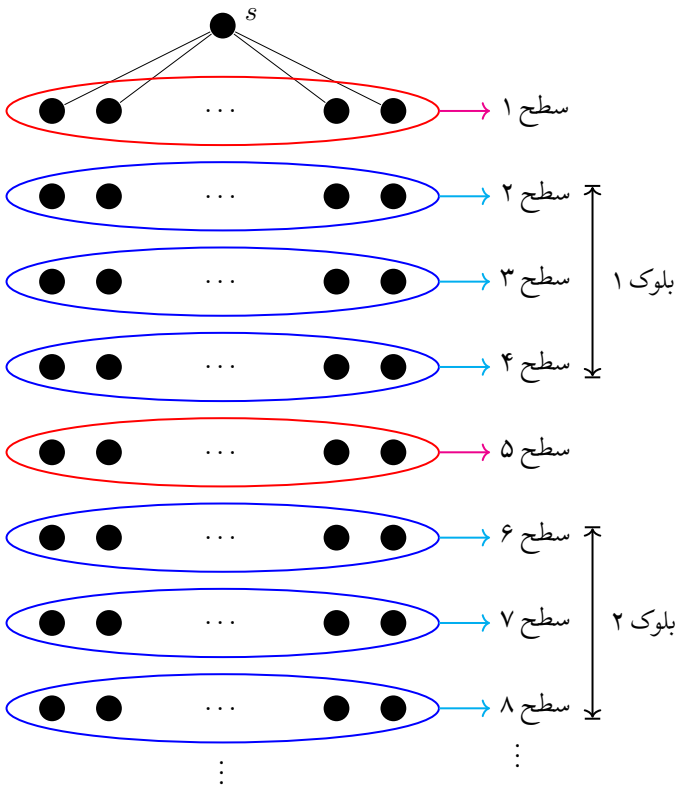
در این قسمت می‌خواهیم یک الگوریتم با ضریب تقریب $1-\epsilon$ برای مسأله‌ی بزرگ‌ترین زیرمجموعه‌ی مستقل روی گراف‌های مسطح ارائه کنیم. همان‌طور که می‌دانیم این الگوریتم‌ها در حالت کلی PTAS[22] نامیده می‌شوند و الگوریتم با ضریب تقریب $1-\epsilon$ در این حوزه، بدین معناست که این الگوریتم به ازای یک ϵ ثابت، در زمانی چندجمله‌ای می‌تواند پاسخی برای مساله بیابد که حداکثر به اندازه‌ی یک ضریب $1-\epsilon$ از جواب بهینه فاصله داشته باشد. به عنوان مثال فرض کنید اندازه‌ی زیرمجموعه‌ی مستقل در یک گراف مسطح برابر با OPT باشد. الگوریتم با ضریب تقریب $1-\epsilon$ که در ادامه خواهیم گفت، به ازای یک ϵ ثابت، در زمانی چندجمله‌ای می‌تواند یک زیرمجموعه‌ی مستقل از گراف را برگرداند که اندازه‌ی آن حداکثر به اندازه‌ی ϵOPT از OPT کم‌تر باشد.

الگوریتم:

- قرار دهید $L = \frac{1}{\epsilon}$ و $r \in \{0, 1, \ldots, L-1\}$ را به صورت کاملا تصادفی انتخاب کنید.
- از یک رأس دلخواه مانند s الگوریتم جست‌وجوی اول سطح یا همان BFS را اجرا کرده و سطح متناظر با هر گره را نگه دارید.
- رأس‌های سطح r، $r+L$، $r+2L$ و ... را حذف کنید.
- هر $L-1$ سطح متوالی حذف نشده را یک بلوک بنامید و مسأله‌ی یافتن بزرگ‌ترین زیرمجموعه‌ی مستقل را برای هر بلوک حل کنید.
- اجتماع جواب‌ها روی بلوک‌ها را برگردانید.

[20] subexponential
[21] Eppstein
[22] polynomial-time approximation scheme

به عنوان مثال فرض کنید که $\epsilon = \frac{1}{4}$ است و درنتیجه $L = 4$ و عدد تصادفی r برابر با 1 انتخاب شده است. همچنین فرض کنید از یک رأس دلخواه s الگوریتم جست‌وجوی اول سطح اجرا شده و سطح‌ها مطابق با شکل ۲۶ تعیین شده‌اند. مطابق گام‌های الگوریتم باید سطوح 1، 5، 9 و ... حذف شوند و هر سه سطر متوالی حذف نشده به عنوان یک بلوک شناخته شود. برای درک این مطلب سطرهای حذف شده با رنگ قرمز و سطرهای بلوک‌ها با رنگ آبی در شکل ۲۶ جدا شده‌اند. همچنین بلوک‌ها نیز در سمت راست شکل نام‌گذاری شده‌اند.



شکل ۲۶

درستی الگوریتم: جوابی که در نهایت به عنوان مجموعه‌ي مستقل برگردانده می‌شود، به درستی یک زیرمجموعه‌ی مستقل خواهد بود. چرا که اگر بین دو رأس متمایز یالی باشد، این دو راس حتما در یک سطح یا در دو سطح با حداکثر اختلاف ۱ قرار خواهند گرفت. چون بین هر دو بلوک حداقل یک سطر فاصله است، پس بین رأس‌های دو بلوک مجزا نمی‌تواند هیچ یالی وجود داشته باشد و در نتیجه مجموعه رئوس حاصل از اجتماع زیرمجموعه‌های مستقل بلوک‌ها همچنان یک زیر مجموعه‌ی مستقل خواهد بود. حال باید دو مطلب را ثابت کنیم. یکی زمان اجرای مناسب و یکی این‌که چرا این الگوریتم یک PTAS با ضریب تقریب $1 - \epsilon$ خواهد بود.

زمان اجرای الگوریتم: برای اینکه بتوانیم ثابت کنیم زمان اجرای این الگوریتم به ازای یک ϵ ثابت چندجمله‌ای است، ابتدا نیاز داریم تا لم زیر را ثابت کنیم.

لم ۱. *عرض درختی هر بلوک از مرتبه‌ی $O(\frac{1}{\epsilon})$ یا $O(L)$ است.*

اگر این لم را بپذیریم، زمان اجرا به سادگی به فرم مطلوب در می‌آید. چرا که در این صورت می‌دانیم با در نظر گرفتن یک ϵ ثابت، عرض درختی هر بلوک یک مقدار ثابت می‌شود و می‌دانیم که برای گراف‌های با عرض درختی ثابت می‌توانیم مسأله‌ی بزرگ‌ترین زیرمجموعه‌ی مستقل را در زمان چندجمله‌ای حل کنیم و در نتیجه زمان اجرا در کل نیز مطلوب است. در انتهای این بخش این لم را ثابت خواهیم کرد.

ضریب تقریب الگوریتم: حال می‌خواهیم ثابت کنیم که چرا ضریب تقریب $1-\epsilon$ برای این الگوریتم درست است. فرض کنید اندازه‌ی بزرگ‌ترین مجموعه‌ی مستقل OPT باشد. در این صورت خواهیم داشت

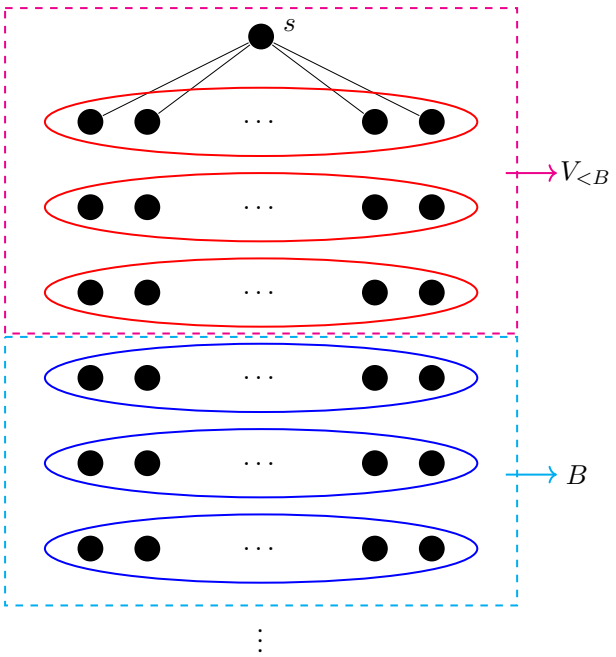
$$\mathbb{E}[\text{ تعداد راس‌های حذف شده از جواب بهینه }] = \epsilon\, OPT$$

چرا که ما از هر L سطر یک سطر را حذف می‌کنیم و در نتیجه به صورت متوسط از همه‌ی n رأس گراف ϵn رأس و هم‌چنین از همه‌ی رأس‌های جواب بهینه نیز به طور متوسط $\epsilon\, OPT$ رأس را حذف می‌کنیم. حال جوابی که الگوریتم به ما برمی‌گرداند، $\sum_B \mathsf{MaxIS}(G[B])$ است که در آن جمع زدن روی B یا همان بلوک‌های ساخته شده در الگوریتم است. هم‌چنین $G[B]$ نماد گراف القایی در گراف اصلی است که شامل رئوس بلوک B است. می‌دانیم اندازه‌ی بزرگ‌ترین زیرمجموعه‌ی مستقل در هر بلوک از اندازه‌ی اشتراک جواب بهینه با آن بلوک بیشتر است. (چون اشتراک جواب بهینه با یک بلوک نیز خود یک زیرمجموعه‌ی مستقل از آن بلوک است که به وضوح از بزرگ‌ترین زیرمجموعه‌ی مستقل آن بلوک اعضای کم‌تری دارد) در نتیجه می‌توان نامساوی زیر را نوشت

$$\begin{aligned}\sum_B \mathsf{MaxIS}(G[B]) &\geq \sum_B |OPT \cap B| \\ &= \text{تعداد تمام رئوسی از جواب بهینه که حذف نشده‌اند} \\ &= OPT - (\text{تعداد راس‌های حذف شده از جواب بهینه}) \\ &= (1-\epsilon)OPT\end{aligned}$$

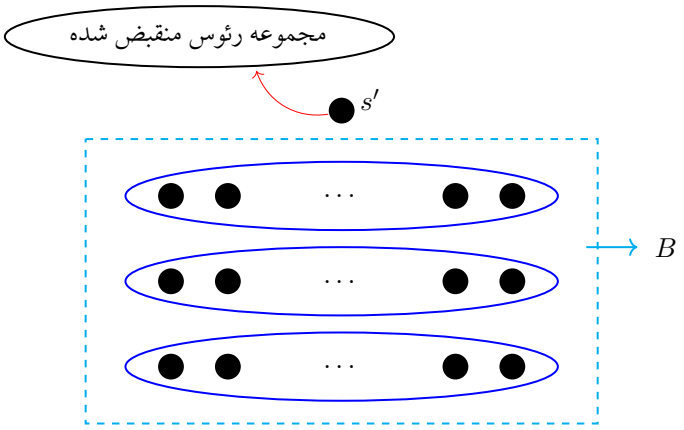
که البته تساوی اخیر به صورت متوسط نوشته شده است. روابط بالا ثابت می‌کنند که ضریب تقریب این الگوریتم $1-\epsilon$ است. حال فقط اثبات لم باقی می‌ماند که به عنوان آخرین بخش از این جلسه آن را ذکر می‌کنیم.

اثبات لم ۱. یک بلوک خاص مانند B را در نظر بگیرید. منظور از B مجموعه‌ی همه‌ی رئوس آن بلوک است. نام مجموعه‌ی همه‌ی رئوس در سطوح بالاتر از این بلوک را $V_{<B}$ بنامید و گراف القایی حاصل از $B \cup V_{<B}$ روی گراف اصلی را در نظر بگیرید. نام این زیرگراف القایی را $G[B \cup V_{<B}]$ بنامید.



شکل ۲۷

حال همه‌ی رئوس $V_{<B}$ و یال‌های متصل به آن‌ها را در گراف $G[B \cup V_{<B}]$ منقبض کنید و گراف حاصل را H بنامید.



شکل ۲۸: گراف H

چون گراف اصلی مسطح بود و عمل انقباض یال‌ها، مسطح بودن گراف را حفظ می‌کند، در نتیجه گراف H هم‌چنان مسطح است. نکته‌ی نهایی این است که $\text{diam}(H) = O(\frac{1}{\epsilon}) = O(L)$ است. چرا که برای هر دو رأس دلخواه u و v در گراف H یک مسیر با طول حداکثر $2L$ وجود دارد. زیرا هر لایه به لایه‌ی بالایی و پایینی خود متصل است و حتی اگر بخواهیم بین دو رأس هم‌سطح یک مسیر بیابیم، با حداکثر جابه‌جایی بین $2L$ سطح این مسیر وجود دارد. بنابراین طبق قضیه‌ی ۱۳ عرض درختی گراف H نیز $O(L)$ خواهد بود که این نشان می‌دهد عرض درختی بلوک B هم $O(L)$ است و اثبات لم به پایان می‌رسد. □

مراجع

[1] Kleinberg, Jon and Tardos, Eva. *Introduction to Algorithms*. 1st ed., 2005, pp. 572-591.

[2] *"Series-parallel graph"*, Wikipedia. Available at: https://en.wikipedia.org/wiki/Series-parallel_graph (Accessed: August 5, 2020).

[3] *"Clique (graph theory)"*, Wikipedia. Available at: https://en.wikipedia.org/wiki/Clique_(graph_theory) (Accessed: August 5, 2020).

[4] *"Chordal graph"*, Wikipedia. Available at: https://en.wikipedia.org/wiki/Chordal_graph (Accessed: August 5, 2020).

[5] *"Monadic second-order logic"*, Wikipedia. Available at: https://en.wikipedia.org/wiki/Monadic_second-order_logic (Accessed: August 6, 2020).

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

الگوریتم‌های تقریبی ۱: روش حریصانه — نگارنده: آرمیتا کاظمی‌نجف آبادی

در جلسات قبل (جلسه ۲۱) بحث کردیم که مسائل سخت [1] را به صرف سخت بودن نمی‌شود از حلشان صرف نظر کرد؛ بنابراین از رویکردهای متنوعی برای برخورد با آنها استفاده می‌کنیم. مثلا برای به دست آوردن جواب بهینه، سعی می‌کنیم یک الگوریتم نمایی یا به طور کلی غیر چندجمله‌ای که زمان آن تا جای ممکن خوب باشد ارائه دهیم یا اینکه به طور کلی از به دست آوردن جوابِ بهینه صرف نظر کرده و یک جواب تقریبی (تا جای ممکن نزدیک به جواب بهینه) در زمان بهتر (مثلا چندجمله‌ای) ارائه دهیم.
در این جلسه و جلسات بعد به بررسی رویکرد دوم خواهیم پرداخت.

۱ الگوریتم تقریبی

الگوریتم A را برای مسأله Q، یک αـ تقریب می‌گوییم، اگر برای ورودی I از Q، نابرابری $c(A(I)) \leqslant \alpha c(OPT(I))$ برقرار باشد.
در اینجا منظور از $c(A(I))$، هزینه خروجی الگوریتم A روی ورودی I و منظور از $c(OPT(I))$، هزینه خروجی الگوریتم بهینه روی ورودی I از مسأله Q است. همچنین α می‌تواند ثابت یا متغیر (بر حسب اندازه ورودی) باشد.
مثلا اگر مسأله Q را یافتن پوشش راسی کمینه در نظر بگیریم و الگوریتم A، یک ۲ـ تقریب برای مسأله Q باشد، در این صورت برای هر گراف مانند I که به الگوریتم A داده شود، خروجی A یک پوشش راسی از I است که اندازه آن حداکثر دو برابر پوشش راسی کمینه ی گراف I باشد.

هدف ما در ادامه این است که برای مسائلِ سختِ بهینه سازی (مسائل **NP-Hard**)، الگوریتم چندجمله ای با ضریب تقریب کوچک (ثابت یا متغیر)، ارائه دهیم. به طور کلی تکنیک‌های متنوعی برای این کار وجود دارد. ما در این جلسه بر تکنیک حریصانه تمرکز کرده و دو مسأله کلاسیکِ الگوریتم‌های تقریبی را بررسی می‌کنیم.
بخش دیگری از موضوعاتِ مورد بحث این زمینه، در این راستاست که ثابت کنند ضریب تقریب یک مسأله از حدی معین کوچکتر نمی‌شود.

به طور کلی واضح است که ضریب تقریب هر الگوریتم چندجمله‌ای برای مسائل سخت، عددی بزرگتر از یک است. (اگر برابر یک باشد؛ مسأله، الگوریتم بهینه ی چندجمله ای دارد و بنابراین سخت نیست.) همچنین برای برخی مسائل ثابت می‌شود الگوریتمی با ضریب تقریب $1+\epsilon$ وجود دارد که می‌توان $\epsilon > 0$ را به قدر دلخواه به صفر نزدیک کرد.

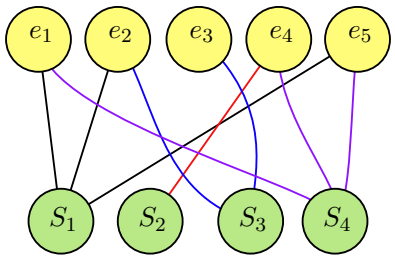
۲ مسأله پوشش مجموعه‌ای [2]

ورودی: مجموعه $U = \{e_1, ..., e_n\}$ و مجموعه $Q = \{S_1, ..., S_m\}$ طوری که برای هر i داشته باشیم $S_i \subseteq U$ که $\bigcup_{i=1}^{m} S_i = U$ باشد و همچنین تابع هزینه ی $c : Q \to \mathbb{R}_+$ که روی اعضای Q تعریف شده است.

[1] NP-Hard
[2] Set Cover

خروجی: زیرمجموعه T از اعضای Q طوری که $\bigcup_{A\in T} A = U$ باشد و $c(T) = \sum_{A\in T} c(A)$ کمینه شود.

به کمک گراف دوبخشی، صورت بندی دیگری از این مسأله ارائه می‌دهیم. در یک بخش رئوس متناظر با S_iها و در بخش دیگر رئوس متناظر با e_iها را قرار دهیم و تنها در صورتی بین e_i و S_j یال بگذاریم که $e_i \in S_j$ باشد.
مثلا اگر در گراف زیر، تابع هزینه به صورت $c(S_i) = i^2$ باشد؛



به راحتی می‌توان دید T حتما باید شامل S_3 باشد. (چون فقط S_3 شامل e_3 است.) اگر T شامل S_4 باشد، حداقل هزینه ای که در این حالت به دست می‌آید 25 است و اگر شامل S_4 نباشد، حداقل هزینه ی ممکن 14 خواهد بود. (چون در این حالت برای پوشاندن کل U توسط اعضای T، می‌بایست S_1 و S_2 هم انتخاب شوند.)

$$c\left(\{S_3, S_4\}\right) = 3^2 + 4^2 = 25$$

$$c\left(\{S_1, S_2, S_3\}\right) = 1^2 + 2^2 + 3^2 = 14$$

بنابراین در اینجا $T = \{S_1, S_2, S_3\}$ است.

می‌خواهیم برای این مسأله الگوریتمی تقریبی به کمک تکنیک حریصانه به دست آوریم.
یک ایده ی ساده ی اولیه این است که S_iها را به ترتیب هزینه شان مرتب کنیم و T را از روی همان ترتیب بسازیم. به راحتی می‌توان دید که این الگوریتم خوب نیست زیرا اجتماع S_iهای کم هزینه تر ممکن است e_iهای کمتری را هم پوشش دهد و مجبور شویم S_iهای بیشتری به عضویت T درآوریم که هزینه نهایی را بالا می‌برند.
مثلا فرض کنید: $S_1 = \{e_1\}$، $S_2 = \{e_2\}$، $S_n = \{e_n\}$، $S_{n+1} = \{e_1, e_2, ..., e_n\}$
و تابع هزینه به صورت $S_1 = 1$ و $S_2 = 1$ و ... و $S_n = 1$ و $S_{n+1} = 1 + \epsilon$ باشد.
در این صورت واضح است که جواب بهینه مسأله $T = \{S_{n+1}\}$ با هزینه $1 + \epsilon$ است در حالی که الگوریتم حریصانه مذکور به ما $T = \{S_1, ..., S_n\}$ با هزینه ی بیشترِ n می‌دهد. که واقعا تقریب خوبی از جواب بهینه نیست. اما همین رویکرد ساده، زمینه ساز ایده ی بهتری برای انتخاب الگوریتم حریصانه است.

الگوریتم حریصانه: مجموعه‌های S_i را با توجه به نسبتِ **هزینه، به تعداد عضوهای جدید** U **که توسط** S_i **پوشانده می‌شوند** مرتب کنیم و T را از روی همان ترتیب بسازیم.
برای بیان دقیق تر می‌توان چنین توصیفی ارائه داد:

GreedySetCover(U)

1 $T = \emptyset$
2 $C = \emptyset$
3 **while** $C \neq U$
4 $\quad S_i = \arg\min_{S_i \in Q \setminus T} \frac{c(S_i)}{|S_i \setminus C|}$
5 $\quad T = T \cup \{S_i\}$
6 $\quad$ // **for** $e \in S_i \setminus C$
7 $\quad\quad$ // PRICE(e) $= \frac{c(S_i)}{|S_i \setminus C|}$
8 $\quad C = C \cup S_i$
9 **return** T

در اینجا مرحله به مرحله T را می‌سازیم؛ به این شکل که در هر بار اجرای حلقه While، یک S_i را به T اضافه می‌کنیم.
مجموعه C نیز اجتماع S_iهای داخل T است و درواقع اعضایی از U را نشان می‌دهد که حداقل در یکی از S_iهای عضو T آمده اند.
الگوریتم زمانی متوقف می‌شود که اجتماع S_iها که همان C است، برابر U شود زیرا از آنجایی که تابع هزینه مثبت است، در هر مرحله ای که $C = U$ شود، دیگر لازم نیست عضو جدیدی به T اضافه کنیم چون فقط هزینه آن را بیشتر می‌کند.
در ادامه از نماد C_t، برای نشان دادن مجموعه C در tـاُمین اجرای حلقه While استفاده می‌کنیم.

تکنیک حریصانه، خود را در انتخاب S_i مناسب برای افزودن به T نشان می‌دهد؛ در هر مرحله (هر بار اجرای حلقه While)، S_i ای انتخاب می‌شود که نسبتِ هزینه به تعداد اعضای متمایزش که در C نیامده اند، کمتر باشد.

تحلیل درستی: *خطوط 6 و 7 در شبه کد الگوریتم اخیر، نقشی در اجرای الگوریتم ندارند و صرفا کار تحلیل را ساده می‌کنند.*
فرض کنید برای هر عضو U، تابع $price$ تعریف شود که ادعا می‌کنیم مقدار آن به طور یکتا در خط 7 شبه کد اخیر مشخص می‌شود.
علت درستی ادعای اخیر این است که اولاً درنهایت اجتماع S_iهای عضو T برابر U است پس حتما S_j وجود دارد که شامل عضو دلخواهی از U مانند e باشد؛ ثانیاً بدون کاستن از کلیت می‌توان فرض کرد S_j اولین مجموعه ی شامل e بوده که آن را به T افزوده ایم. بنابراین $price(e) = \frac{c(S_j)}{|S_j \setminus C|}$ خواهد بود و یا کمی‌دقت به شرط خط 6 شبه کد، مشخص است که در تکرارهای بعدی حلقه تا اتمام اجرای الگوریتم، مقدار $price(e)$ تغییر نخواهد کرد.
پس بنابر آنچه گفتیم می‌توان دید $price$ برای همه اعضای U تعریف شده و دقیقا یکبار مقداردهی می‌شود. ضمنا بنا به نحوه تعریف تابع $price$، در هر مرحله (مانند مرحله t) که یک S_i انتخاب می‌شود، هزینه $c(S_i)$ برابر $\sum_{e \in S_i, e \notin C_t} price(e)$ می‌باشد که در اینجا منظور از C_t، همان مجموعه C در tـاُمین اجرای حلقه While است. (به طور شهودی انگار هزینه S_i بین eهایی که عضو $S_i \setminus C_t$ اند پخش می‌شود.)
گزاره‌های اخیر نشان می‌دهند رابطه $\sum_{i=1}^{n} price(e) = \sum_{S_i \in T} c(S_i)$ برقرار است. در آینده از این رابطه استفاده خواهیم کرد.
در ادامه ضریب تقریب الگوریتم را بررسی می‌کنیم.

قضیه ۱. *الگوریتمِ* GreedySetCover *یک H_nـ تقریب برای مسأله پوشش مجموعه‌ای است.*

اثبات. منظور از H_n عددهارمونیک nـام است که از رابطه $H_n = \sum_{i=1}^{n} \frac{1}{i}$ به دست می‌آید.
همچنین از قبل می‌دانیم $H_n = O(\ln n) = O(\lg n)$ است.
اعضای U را با $e_1, e_2, ..., e_k, ..., e_n$ برچسب گذاری می‌کنیم طوری که e_k، kـاُمین عضو U باشد که مقدار تابع $price$ (در kـاُمین اجرای خط 7 شبه کد) به ازای آن تعیین می‌شود.
برای یک k ثابت، مجموعه E را چنین تعریف می‌کنیم: $E = \{e_k, ..., e_n\}$

همچنین OPT_k را بزرگترین زیرمجموعه از OPT تعریف می‌کنیم که اشتراک اعضای آن با E ناتهی است؛
یا به عبارتی $OPT_k = \{O_1, ..., O_p\}$ طوری که برای هر i طبیعی بین 1 تا p نامساوی $O_i \cap E \neq \varnothing$ برقرار است.

ادعای اول: اگر $i < j$ آنگاه $e_i < e_j$ است.
(ادعای اول نقشی در اثبات قضیه ندارد و تنها یک نکته ست که شاید توجه به آن خالی از لطف نباشد.)

اثبات درستی: برای اثبات درستی ادعا کافی ست فرض کنید $price(e_i) = \frac{c(S_t)}{|S_t \setminus C_v|}$ و $price(e_j) = \frac{c(S_l)}{|S_l \setminus C_{v'}|}$ باشد.
از آنجایی که $i < j$ ، بنابه نحوه برچسب گذاری اعضای U، مشخص است که $price(e_i)$ زودتر از $price(e_j)$ مقداردهی می‌شود.
بنابراین $|C_v| \subseteq |C_{v'}|$ (منظور از C_v، همان مجموعه C در vـاُمین اجرای حلقه While است.)
با توجه به گزاره‌های قبل و نحوه انتخاب S_t و S_l در خط 4 شبه کد، نتیجه بگیرید:

$$price(e_i) = \frac{c(S_t)}{|S_t \setminus C_v|} \leqslant \frac{c(S_l)}{|S_l \setminus C_v|} \leqslant \frac{c(S_l)}{|S_l \setminus C_{v'}|} = price(e_j)$$

که نامساوی اخیر نشان دهنده درستی ادعاست.

لم ۱. *برای k طبیعی دلخواه که $1 \leqslant k \leqslant n$ و هر i که $e_k \in O_i$، نامساوی $price(e_k) \leqslant \frac{c(O_i)}{|O_i \cap E|}$ برقرار است.*

اثبات. با توجه به نحوه تعریف OPT_k، می‌دانیم تمام مجموعه‌های به فرم S_i که شامل e_k اند، عضو OPT_k هستند. (چون اشتراکشان با E ناتهی ست)
فرض کنید $price(e_k) = \frac{c(S_r)}{|S_r \setminus C_t|}$، همچنین فرض کنید O_i یک عضو دلخواه OPT_k شامل e_k باشد. از آنجایی که O_i عضوی از Q نیز هست، j وجود دارد که $O_i = S_j$ باشد. می‌دانیم $price(e_k)$ برای اولین و آخرین بار توسط S_r مقداردهی شده؛ یعنی در مرحله ای که S_r انتخاب می‌شود (مرحله tـام)، هنوز S_j عضو T نیست. بنابراین شرط انتخاب حریصانه S_r (خط 4 شبه کد) نتیجه می‌دهد:

$$\frac{c(S_r)}{|S_r \setminus C_t|} \leqslant \frac{c(S_j)}{|S_j \setminus C_t|}$$

بنا به نحوه برچسب گذاری اعضای U، می‌دانیم C_t که تا این مرحله به دست آمده، زیرمجموعه‌ای از $\{e_1, ..., e_{k-1}\}$ است. بنابراین داریم:

$$|S_j \setminus C_t| \geqslant |S_j \setminus \{e_1, ..., e_{k-1}\}| = |S_j \cap \{e_k, ..., e_n\}| = |S_j \cap E|$$

از دو نامساوی اخیر نتیجه می‌شود:

$$price(e_k) = \frac{c(S_r)}{|S_r \setminus C_t|} \leqslant \frac{c(S_j)}{|S_j \setminus C_t|} \leqslant \frac{c(S_j)}{|S_j \cap E|} = \frac{c(O_i)}{|O_i \cap E|}$$

$$\Rightarrow price(e_k) \leqslant \frac{c(O_i)}{|O_i \cap E|}$$

□

لم ۲. *برای k طبیعی دلخواه که $1 \leqslant k \leqslant n$، $price(e_k) \leqslant \frac{c(OPT)}{n-k+1}$ است.*

اثبات. بنا به تعریف $E = \{e_k, ..., e_n\}$ است. با توجه به نحوه تعریفِ OPT_k، اجتماع اعضای OPT_k باید E را بپوشاند یا به عبارتی:
$E = \bigcup_{i=1}^{p} O_i = \bigcup_{i=1}^{p} O_i \cap E$
بنابراین نامساوی $|E| \leqslant \sum_{i=1}^{p} |O_i \cap E|$ برقرار است. (هر عضو E ممکن است توسط چند O_i پوشانده شود.)
نکته دیگر، درستی لم ۱ و در نتیجه برقراری نامساوی زیر است:

$$price(e_k) \leqslant \frac{c(O_i)}{|O_i \cap E|} \Rightarrow price(e_k) \cdot |O_i \cap E| \leqslant c(O_i)$$

نهایتاً با کنار هم قرار دادن نامساوی‌های اخیر، می‌توان روند زیر را طی کرد:

$$price(e_k)\cdot(n-k+1)=price(e_k)\cdot|E|\leqslant price(e_k)\cdot\sum_{i=1}^{p}|O_i\cap E|\leqslant\sum_{i=1}^{p}c\left(O_i\right)=c\left(OPT_k\right)\leqslant c\left(OPT\right)$$

$$\Rightarrow price(e_k)\cdot(n-k+1)\leqslant c\left(OPT\right)\Rightarrow price(e_k)\leqslant\frac{c\left(OPT\right)}{n-k+1}$$

□

برای اثبات حکم قضیه از لم ۲ بهره می‌گیریم:

$$price(e_k)\leqslant\frac{c\left(OPT\right)}{n-k+1}\Rightarrow\sum_{i=1}^{n}price(e_k)\leqslant\sum_{i=1}^{n}\frac{c\left(OPT\right)}{n-k+1}=c\left(OPT\right)\sum_{i=1}^{n}\frac{1}{n-k+1}=H_n\times c\left(OPT\right)$$

با استفاده از نامساوی اخیر و رابطه ی $\sum_{i=1}^{n}price(e)=\sum_{S_i\in T}c(S_i)$ که درستی آن را قبلا اثبات کردیم، ثابت می‌شود:

$$\sum_{S_i\in T}c\left(S_i\right)\leqslant H_n\times c\left(OPT\right)$$

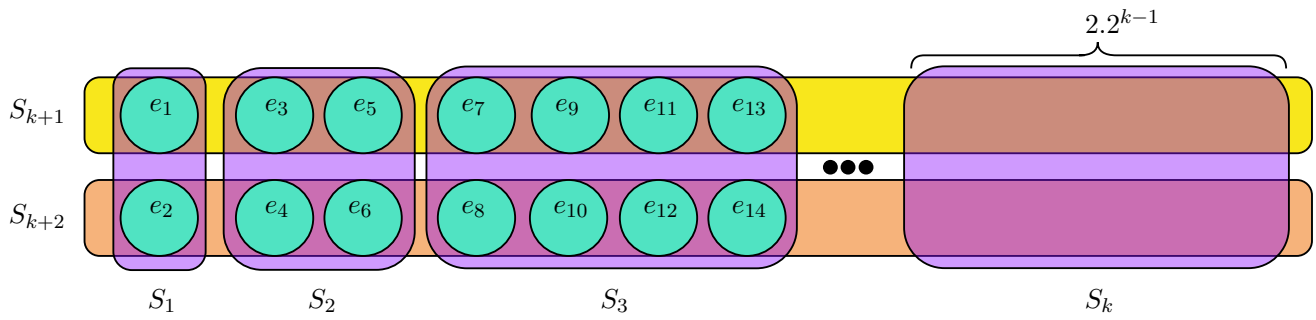
که همان حکم قضیه است؛ یعنی ثابت کردیم الگوریتم GreedySetCover یک H_nـتقریب برای مسأله پوشش مجموعه‌ای است.

□

در ادامه برای اینکه نشان دهیم الگوریتم حریصانه مان ضریب تقریبی بهتر از $H_n=O(\lg n)$ ندارد، یک مثال ارائه می‌دهیم.

در این مثال، جواب الگوریتم حریصانه، $\Omega\left(\lg n\right)$ برابر جواب بهینه است که نشان می‌دهد ضریب تقریب الگوریتم حریصانه ی ارائه شده، از همین مرتبه است.

اعضای U مانند قبل به فرم e_i نمایش داده شده اند (تعداد اعضای U برابر $n=2^{k+1}-2$ است.) S_1 تا S_{k+2} نیز مجموعه‌هایی هستند که برای پوشش U می‌توان به کار برد. (درواقع $Q=\{S_1,...,S_{k+2}\}$ است.) همچنین تابع هزینه، ثابت و برابر یک است. $(c\left(S_i\right)=1)$



با کمی‌دقت می‌توان دید $|S_{k+1}|=|S_{k+2}|=2^k-1$.

این نیز واضح است که $OPT=\{S_{k+1},S_{k+2}\}$ و در نتیجه $c\left(OPT\right)=2$ است.

می‌توان دید الگوریتم حریصانه در گام اول S_k را انتخاب می‌کند و به کمک استقرا می‌توان نشان داد مجموعه T که توسط الگوریتم حریصانه به دست آمده، برابر $\{S_1,...,S_k\}$ خواهد بود که هزینه آن بر حسب تعداد اعضای U، لگاریتمی‌است.

$$c\left(T\right)=k=\Theta\left(\lg(2^{k+1}-2)\right)=\Theta\left(\lg n\right)$$

بنابراین ضریب تقریب الگوریتم حریصانه ای که ارائه کردیم، از $\Omega(\lg n)$ بهتر نیست.

خوب است ثابت کنیم مسأله پوشش مجموعه‌ای متعلق به کلاس **NP-Hard**، و جزو مسائل سخت است. یک تحویل چندجمله ای از مسأله پوشش راسی به حالت خاصی از مسأله پوشش مجموعه‌ای ارائه می‌دهیم.
یک گراف دلخواه مانند G را به عنوان نمونه برای مسأله پوشش راسی در نظر بگیرید. در زمان چندجمله‌ای مجموعه U، متشکل از یال‌های گراف G و مجموعه‌های S_i که S_i شامل یال‌های گذرنده از iـاُمین راس G است می‌سازیم. تابع هزینه را نیز ثابت و برابر یک در نظر می‌گیریم. با انجام این اعمال، اکنون یک نمونه برای مسأله پوشش مجموعه‌ای به دست آورده ایم.
با کمی‌دقت در نحوه تعریف S_iها می‌توان دید اگر گراف G پوششی راسی با اندازه k داشته باشد، آنگاه نمونه ای که از روی G برای مسأله پوشش مجموعه‌ای ساختیم جوابی با حداکثر هزینه ی k خواهد داشت. عکس این موضوع هم به طریق مشابه برقرار است. بنابراین تحویل ارائه شده معتبر است و چون از قبل می‌دانستیم مسأله پوشش راسی جزو مسائل سخت است، اکنون نتیجه می‌شود حالت خاص مسأله پوشش مجموعه‌ای و در نتیجه خود آن، در زمره مسائل سخت هستند.

به طور کلی در مورد مسأله پوشش مجموعه‌ای ثابت شده الگوریتمی با ضریب تقریب بهتر از $\Omega(\lg n)$ برای آن وجود ندارد. و اخیراً نتیجه قوی تری نیز به اثبات رسیده که برای این مسأله، به ازای هر $\epsilon > 0$، ضریب تقریب بهتر از $(1-\epsilon)\ln n$ وجود ندارد مگر آنکه $\mathbf{P} = \mathbf{NP}$ باشد.

علاوه بر مسأله پوشش راسی که به نوعی حالتی خاص از مسأله پوشش مجموعه‌ای بود، حالات خاص دیگری هم می‌توان متصور شد که در ادامه آنها را بررسی می‌کنیم. تعریف کنید:
f: بیشترین تعداد مجموعه‌هایی (مانند S_i) که یک عنصر در آن ظاهر شده است.
Δ: اندازه بزرگترین S_i : $\max_{S_i \in Q} |S_i|$

قضیه ۲. *الگوریتمِ* GreedySetCover *یک* H_Δـ*تقریب برای مسأله پوشش مجموعه‌ای است.*

اثبات. یک جواب بهینه مسأله مانند $OPT = \{O_1, ..., O_p\}$ و O_i دلخواه، عضو OPT، طوری که $O_i = \{e_1', ..., e_d'\}$ در نظر بگیرید. می‌دانیم $d \leqslant \Delta$ است.
اعضای O_i طوری نام گذاری شده اند که اگر $r < s$، آنگاه $price(e_r')$ زودتر از $price(e_s')$ مقداردهی شده است. بنابراین اگر k دلخواه که $1 \leqslant k \leqslant d$ را در نظر بگیریم و فرض کنیم $price(e_k') = \frac{c(S_j)}{|S_j \setminus C_t|}$ باشد، در این صورت یا $O_i = S_j$ است یا تا مرحله tـام، O_i انتخاب نمی‌شود. (اگر تا قبل این مرحله انتخاب شود، در تناقض با مقدار مفروض $price(e_k')$ است.) پس در مرحله tـام، با توجه به انتخاب حریصانه (خط 4 شبه کد)، نامساوی زیر برقرار است:

$$\frac{c(S_j)}{|S_j \setminus C_t|} \leqslant \frac{c(O_i)}{|O_i \setminus C_t|}$$

می‌دانیم در مرحله tـام، $price(e_k')$ مقداردهی می‌شود. با توجه به شیوه نام گذاری اعضای O_i، تعداد اعضایی از O_i که تا قبل از مرحله t مقدار تابع $price$ شان معلوم شده، حداکثر $k-1$ است. پس داریم:

$$|O_i \setminus C_t| \geqslant d - (k-1)$$

$$\Rightarrow \frac{c(O_i)}{|O_i \setminus C_t|} \leqslant \frac{c(O_i)}{d-k+1}$$

با کنار هم گذاشتن نامساوی‌های به دست آمده، نتیجه می‌شود:

$$\Rightarrow price(e_k') \leqslant \frac{c(O_i)}{d-k+1}$$

از آنجایی که نامساوی اخیر برای k دلخواه که $1 \leqslant k \leqslant d$ برقرار است، همه نابرابری‌های از این نوع به ازای همه kهای ممکن را جمع می‌زنیم و روابط زیر به دست می‌آید:

$$\Rightarrow \sum_{k=1}^{d} price\left(e_k{}'\right) \leqslant \sum_{k=1}^{d} \frac{c\left(O_i\right)}{d-k+1} = c\left(O_i\right) \cdot \sum_{k=1}^{d} \frac{1}{d-k+1} = H_d \times c\left(O_i\right) \leqslant H_\Delta \times c\left(O_i\right)$$

می‌دانیم اجتماع اعضای OPT برابر U است پس با کمی‌دقت و به کمک روابط قبلی می‌توان نوشت:

$$c\left(T\right) = \sum_{k=1}^{n} price\left(e_k{}'\right) \leqslant \sum_{i=1}^{p} \sum_{e \in O_i} price\left(e\right) \leqslant \sum_{i=1}^{p} H_\Delta \cdot c\left(O_i\right) = H_\Delta \cdot \sum_{i=1}^{p} c\left(O_i\right) = H_\Delta \cdot c\left(OPT\right)$$

در نتیجه $c\left(T\right) \leqslant H_\Delta \cdot c\left(OPT\right)$ و حکم ثابت است. □

بنابراین اگر Δ به میزان کافی کوچک باشد، تقریب H_Δ از H_n بهتر خواهد بود.

در حالتی که مقدار f کوچک باشد، می‌توان الگوریتم حریصانه ساده تری با ضریب تقریب بهتر ارائه کرد. (درواقع می‌توان الگوریتمی fـ تقریب ارائه داد.) مثلا وقتی $f = 2$ است در حالتی که تابع وزن برابرِ ثابتِ یک باشد و هر عضو U در حداقل دو تا از S_i ظاهر شود؛ می‌توان مسأله را به مسأله پوشش راسی تبدیل کرد.
می‌خواهیم برای مسأله پوشش راسی الگوریتم با ضریب تقریب دو ارائه دهیم. بین مسأله پوشش راسی و تطابق بیشینه ارتباطی وجود دارد که در این الگوریتم از آن بهره می‌گیریم.

لم ۳. *برای گراف ساده ی بدون وزن و بدون جهتِ دلخواه، اگر OPT_{VC} پوشش راسی کمینه و $OPT_{matching}$ تطابق بیشینه باشد؛ $c\left(OPT_{VC}\right) \geqslant c\left(OPT_{matching}\right)$ است.*

اثبات. یک تطابق دلخواه و یک پوشش راسی در نظر بگیرید. هر راسی که در پوشش راسی انتخاب شده، حداکثر یکی از یال‌های تطابق را می‌پوشاند. از آنجایی که پوشش راسی، می‌بایست همه یال‌ها من جمله یال‌های تطابق را بپوشاند، به ازای هر یالِ تطابق، حداقل یکی از رئوس دو سر آن عضو پوشش راسی است. بنابراین اندازه هر پوشش راسی بزرگتر از هر تطابق است. خصوصا اندازه (هزینه) پوشش راسی کمینه از اندازه (هزینه) تطابق بیشینه بیشتر است. □

توصیف الگوریتم:
ـ یک تطابق ماکزیمال مانند M انتخاب می‌کنیم. (هر بار یک یال از گراف انتخاب می‌کنیم که رئوس دو سر آن با رئوس یال‌های انتخاب شده در مراحل قبلی اشتراک نداشته باشد. زمانی که الگوریتم به پایان می‌رسد، یک تطابق ماکزیمال داریم.)
ـ همه رئوسی که دو سر یکی از یال‌های این تطابق اند به عنوان پوشش راسی T معرفی می‌کنیم.

ادعا می‌کنیم الگوریتم توصیف شده، 2ـ تقریب است. ابتدا می‌بایست ثابت کنیم رئوس عضو T پوشش راسی تشکیل می‌دهند. فرض خلف کنید که این گونه نباشد. بنابراین یک یال مانند e وجود دارد که هیچ یک از رئوس دو سر آن، عضو T نیست. اما اگر چنین یالی داشته باشیم، می‌توانیم آن را به تطابق ماکزیمال M بیفزاییم و تطابقی بزرگتر به دست آوریم که با ماکزیمال بودن M در تناقض است. بنابراین فرض خلف باطل است و T یک پوشش راسی با اندازه ی دو برابر تطابق M معرفی می‌کند. با استفاده از لم ۳ و گزاره‌های اخیر می‌توان نوشت:

$$2 \times c\left(OPT_{VC}\right) \geqslant 2 \times c\left(OPT_{matching}\right) \geqslant 2 \times |M| \geqslant |T| = c\left(T\right)$$

بنابراین داریم $c\left(T\right) \leqslant 2 \times c\left(OPT_{VC}\right)$ که نشان می‌دهد الگوریتم 2ـ تقریب است.

به عنوان تمرین می‌توانید همین ایده را برای fهای بزرگتر پیاده کنید. (هر بار یک عضو که تاکنون انتخاب نشده در نظر گرفته و تمام مجموعه‌های شامل این عنصر را انتخاب کرده، به مجموعه T بیفزایید.)

۳ مسأله فروشنده دوره‌گرد[۳]

ورودی: گراف کامل $G = (V, E)$ با تابع وزن $c : E \to \mathbb{R}_+$
خروجی: یک دور هامیلتونی با وزن کمینه از G

نشان می‌دهیم در حالت کلی هیچ ضریب تقریبی برای این مسأله وجود ندارد.
فرض کنید که A الگوریتمی با ضریب تقریب $\alpha(n)$ برای مسأله TSP باشد. نشان می‌دهیم $\mathbf{P} = \mathbf{NP}$ است.
فرض کنید G گرافی دلخواه باشد. از روی گراف G، گراف کامل H را طوری می‌سازیم که مجموعه رئوس H همان رئوس G باشد و تابع وزن روی یال‌های H به صورت زیر تعریف شود: $c(e) = \begin{cases} 1 & e \in G.E \\ n\alpha & e \notin G.E \end{cases}$
منظور از $G.E$ مجموعه یال‌های G است.
اگر G دور هامیلتونی داشته باشد؛ مشاهده می‌کنیم که $c(OPT_{TSP}(H)) = n$ است.
زیرا هزینه هر یال از H که در G نیامده، بیش از هزینه یک دور همیلتونی از یال‌های G خواهد بود. پس دور هامیلتونی بهینه ی H، همه یال‌هایش در G هم وجود دارد.
اگر G دور هامیلتونی نداشته باشد؛ مشاهده می‌کنیم که $c(OPT_{TSP}(H)) > n\alpha$ است.
زیرا وقتی G دور هامیلتونی ندارد، در یک دور هامیلتونی از H حتما یکی از یال‌هایی که در G نیست ظاهر می‌شود. پس هزینه هر دور هامیلتونی H حداقل به اندازه هزینه آن یال است.
حال الگوریتم A را روی H اجرا می‌کنیم.
به طور کلی می‌دانیم $c(A) \geqslant c(OPT_{TSP}(H))$ است. حال اگر G دور هامیلتونی نداشته باشد با توجه به گزاره‌های قبل نتیجه می‌شود $c(A) > n\alpha$.
اگر G دور هامیلتونی داشته باشد، می‌توان نوشت:

$$c(A) \leqslant c(OPT_{TSP}) \leqslant \alpha(n) \cdot c(OPT_{TSP}) = \alpha(n) \cdot n = \alpha \cdot n$$

یعنی $c(A) \leqslant n\alpha$؛ پس با توجه به اینکه هزینه A کمتر مساوی یا بیشتر از $n\alpha$ است، می‌توانیم بفهمیم G هامیلتونی ست یا خیر.

کاری که اکنون کردیم، نشان می‌دهد تحویلی چندجمله ای از مسأله فروشنده دوره‌گرد برای گراف ساده دلخواه به مسأله فروشنده دوره‌گردِ تقریبی، وجود دارد و چون قبلا نشان داده بودیم مسأله فروشنده دوره‌گرد جزو مسائل سخت است، اکنون ثابت می‌شود مسأله فروشنده دوره‌گردِ تقریبی نیز سخت است. از آنجایی که الگوریتم A زمان چندجمله‌ای داشت؛ یعنی برای یک مسأله سخت الگوریتم چندجمله ای یافته ایم که نتیجه می‌دهد $\mathbf{P} = \mathbf{NP}$ است. پس کارمان در اینجا تمام شده و هدف بعدی مان بررسی حالت خاصی از این مسأله است.

۱.۳ مسأله فروشنده دوره‌گرد متریک[۴]

ورودی: گراف کامل $G = (V, E)$ با تابع وزن $c : E \to \mathbb{R}_+$ به طوری که تابع وزن شرط نامساوی مثلث دارد. به این معنا که برای هر سه راس از G مانند x و y و z، نابرابری $c(xy) \leqslant c(xz) + c(zy)$ برقرار باشد. (منظور از xy، یال بین x و y است.)
خروجی: یک دور هامیلتونی با وزن کمینه از G

با توجه به وزن درخت فراگیر کمینه G: $c(MST(G))$ و اینکه حذف یک یال از دور هامیلتونی G، یک درخت فراگیر از آن را مشخص می‌کند، می‌توان نوشت: $c(MST(G)) \leqslant c(OPT_{TSP}(G))$ که در انتها از این رابطه استفاده می کنیم.

[3] Traveling Salesman Problem
[4] Metric Traveling Salesman Problem

حال یک الگوریتم 2ـ تقریب برای این مسأله ارائه می‌دهیم.

لم ۴. *دوری هامیلتونی از گراف G وجود دارد که هزینه اش حداکثر دو برابر هزینه درخت فراگیر کمینه باشد.*

اثبات. گراف G' را با توجه به گراف G و با همان مجموعه رئوس می‌سازیم؛ یک درخت فراگیر کمینه از G در نظر می‌گیریم و به ازای هر یالِ این درخت، دو یال با همان هزینه، روی همان رئوس در گراف G' قرار می‌دهیم.
درجه هر راس گراف G' زوج است بنابراین تور اویلری دارد. این تور اویلری از یک راس شروع شده، از تمام یال‌های G'، دقیقا یکبار می‌گذرد و درنهایت به راس اول باز می‌گردد؛ از آنجا که راس‌ها ایزوله نیستند، هر راس هم حداقل یکبار در تور دیده می‌شود.
یال‌های G' مضاعف شده ی یال‌های درخت فراگیر کمینه ی G هستند. بنابراین هزینه ی این تور اویلری، دو برابر هزینه درخت فراگیر کمینه G است. فرض کنید این تور اویلری را به صورت $v_{i_1}v_{i_2}v_{i_3}...v_{i_m}$ نشان دهیم که v_{i_j} راسی از G' است. (می‌دانیم راس آخر این تور همان راس اول است، بنابراین موقع نوشتن دنباله رئوس تور اویلری، آخرین راسی که دیده می‌شود را نمی‌نویسیم.)
می‌دانیم v_{i_j} راسی از G نیز هست. همین دنباله رئوس در G را پیمایش کرده و هر جا به راسی تکراری برخوردید آن را حذف کنید. با توجه به کامل بودن گراف G، دنباله جدیدی که از حذف رئوس تکرای به دست می‌آید، دنباله ای از همه رئوس است (همانطور که گفتیم هر راس حداقل یکبار در تور دیده می‌شود.) ادعا می‌کنیم این دنباله جدید یک دور هامیلتونی معرفی می‌کند که هزینه آن از هزینه تور و در نتیجه از دو برابر هزینه درخت فراگیر کمینه، کمتر است.
هامیلتونی بودن آن واضح است چون همه رئوس دقیقا یکبار آمده اند. منتها در محاسبه هزینه، می‌بایست یال بین راس اول و آخر دنباله را فراموش نکنیم. درستی ادعا در مورد هزینه این دور هامیلتونی ، با توجه به شرط نامساوی مثلث و استقرا قابل اثبات است.
درواقع با هر بار حذف یک راس تکراری، به دنباله ای با هزینه کمتر می‌رسیم. برای مثال هزینه دنباله فرضی $y_1...y_tAx_1...x_kAz_1...z_r$ از رئوس G، برابر است با:

$$\sum_{i=1}^{t-1} c(y_iy_{i+1}) + c(y_tA) + c(Ax_1) + \sum_{i=1}^{k-1} c(x_ix_{i+1}) + c(x_kA) + c(Az_1) + \sum_{i=1}^{r-1} c(z_iz_{i+1}) + c(z_ry_1)$$

در حالی که بعد از حذف عضو تکراری A، دنباله جدید $y_1...y_tAx_1...x_kz_1...z_r$ با هزینه زیر به دست می‌آید:

$$\sum_{i=1}^{t-1} c(y_iy_{i+1}) + c(y_tA) + c(Ax_1) + \sum_{i=1}^{k-1} c(x_ix_{i+1}) + c(x_kz_1) + \sum_{i=1}^{r-1} c(z_iz_{i+1}) + c(z_ry_1)$$

نامساوی مثلث ایجاب می‌کند:

$$c(x_kz_1) \leqslant c(x_kA) + c(Az_1)$$

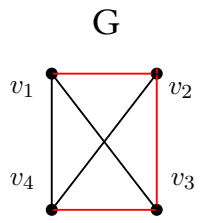
و با کمی‌دقت می‌توان فهمید هزینه دو دنباله جدید و قدیم تنها در همین جملات با هم فرق می‌کنند. پس با افزودن جملاتِ یکسانِ هزینه ی دو دنباله به طرفین نامساوی اخیر، می‌توان دید هزینه دنباله جدید کمتر است. پس هزینه دنباله نهایی که یک دور همیلتونی را مشخص می‌کند، کمتر از هزینه تور اویلری اولیه است. هزینه تور اویلری مذکور نیز دو برابر هزینه درخت فراگیر کمینه G بود. بنابراین حکم ثابت است. □

حال با استفاده از لم اخیر، می‌توانیم یک دور هامیلتونی مانند T ارائه دهیم که $c(T) \leqslant 2 \times c(OPT_{MST})$ ضمنا از رابطه $c(OPT_{MST}) \leqslant c(OPT_{TSP})$ که درستی آن را قبلا اثبات کرده ایم استفاده می‌کنیم. با کنار هم گذاشتن روابط نتیجه می‌شود:

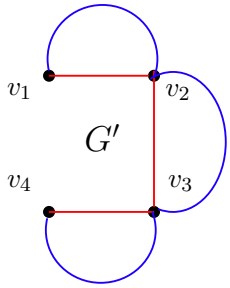
$$c(T) \leqslant 2 \times c(OPT_{MST}) \leqslant 2 \times c(OPT_{TSP})$$

پس $c(T) \leqslant 2 \times c(OPT_{TSP})$ که نشان می‌دهد الگوریتم ارائه شده 2ـ تقریب است.

برای واضح شدن روند طی شده، گراف G را مطابق شکل زیر در نظر بگیرید. فرض کنید هزینه هر یال برابر طول آن در صفحه اقلیدسی باشد. بنابراین یال‌های به رنگ قرمز، یک درخت فراگیر بهینه G را مشخص می‌کنند:



گراف G' رئوس G را دارد و یال‌هایش مضاعف شده ی یال‌های درخت فراگیر G است. تور اویلری G' را می‌توان به فرم دنباله $v_1v_2v_3v_4v_3v_2$ از رئوس نوشت. به ترتیب روی رئوس پیمایش کرده و رئوس تکراری را حذف می‌کنیم تا مثلا دنباله $v_1v_2v_4v_3$ به دست بیاید.



از آنجا که الگوریتم مورد استفاده برای به دست آوردن دنباله $v_1v_2v_4v_3$ که دوری هامیلتونی مانند T را مشخص می‌کند 2ـ تقریب بود، نتیجه می‌گیریم هزینه T از دو برابر هزینه دور هامیلتونی بهینه ($v_1v_2v_3v_4$) بیشتر نیست.

می‌توانیم الگوریتمی با ضریب تقریب بهتر از 2 برای مسأله فروشنده دوره‌گرد متریک ارائه دهیم.
این بار گراف G' را به شکلی دیگر می‌سازیم. راس‌های گراف G' را، همان رئوس G در نظر می‌گیریم و از بین یال‌های G، تنها یال‌های مربوط به درخت فراگیر کمینه ی G را با همان هزینه، روی رئوس‌های متناظرشان در گراف G' قرار می‌دهیم. (یک درخت فراگیر کمینه ی ثابت را در نظر می‌گیریم.) گرافی مانند G'' را نیز به این شیوه می‌سازیم: راس‌های درجه فرد گراف G' را در نظر گرفته و به ازای هر یال بین دو راس درجه فرد در G'، یک یال با همان هزینه بین دو راس متناظر در G''، قرار می‌دهیم.
حال اگر یال‌های یک تطابق کامل از گراف G'' را به گراف G' اضافه کنیم، گراف H به دست می‌آید که درجه همه رئوس آن زوج است و بنابراین اویلری ست.
دقیقا مشابه استدلال‌هایی که برای قسمت قبل انجام دادیم، اینجا نیز می‌توان این تور اویلری را به فرم دنباله مرتب رئوس نوشت و هر بار راسی تکراری را حذف کرد تا در نهایت دنباله ای از رئوس به دست آید که مشخص کننده ی یک دور هامیلتونی مانند T در گراف G ست که هزینه آن از هزینه تور اولیه کمتر است.

حال رئوس درجه فرد در دور هامیلتونی بهینه G را در نظر بگیرید. همه رئوس در دور هامیلتونی بهینه آمده اند بنابراین تعداد رئوس درجه فرد آن، زوج است. آن‌ها را به ترتیبی که در دور ظاهر می‌شوند، $x_1, ..., x_{2r}$ بنامید. دو نوع تطابق از روی همین رئوس می‌توان به دست آورد. تطابقی که هر راس با اندیس فرد را به راس قبلی اش تناظر می‌دهد (راس اول و آخر، دو سر یک یال تطابق می‌شوند) و تطابقی که هر راس با اندیس فرد را به راس بعدی اش تناظر می‌دهد. به کمک نامساوی مثلث می‌توان نشان داد جمع هزینه ی این دو تطابق حداکثر به اندازه هزینه دور هامیلتونی بهینه است. پس حداقل یکی از این دو تطابق، هزینه اش از نصف هزینه ی OPT_{TSP} بیشتر نیست. پس هزینه تطابق کمینه هم از نصف هزینه OPT_{TSP} بیشتر نیست. یعنی:

$$c\left(OPT_{(matching)}(G'')\right) \leqslant \frac{c\left(OPT_{TSP}\right)}{2}$$

هزینه توری که در گراف H به دست آمد، جمع هزینه همه یال‌های H است. جمع هزینه یال‌های H نیز برابر جمع هزینه یال‌های تطابق کمینه گراف G'' و هزینه یال‌های G' (برابر با هزینه درخت فراگیر کمینه G) است. با توجه به اینکه هزینه T نیز از هزینه تور H کمتر است می‌توان نوشت:

$$c(T) \leqslant c\left(OPT_{(matching)}(G'')\right) + c(OPT_{MST}) \leqslant \frac{c(OPT_{TSP})}{2} + c(OPT_{MST})$$

$$\leqslant \frac{c(OPT_{TSP})}{2} + c(OPT_{TSP}) = \frac{3}{2} \times c(OPT_{TSP})$$

در نتیجه $c(T) \leqslant \frac{3}{2} \times c(OPT_{TSP})$ و ضریب تقریب الگوریتم جدید، $\frac{3}{2}$ است.

الگوریتم‌هایی که در این بخش برای مسأله فروشنده دوره‌گرد متریک ارائه دادیم، همگی مربوط به گراف‌های بدون جهت بودند. برای گراف‌های جهت دار، الگوریتم با ضریب تقریب $\lg n$ وجود داشت تا اینکه اخیرا الگوریتمی با ضریب تقریب بهتر $O(\frac{\lg n}{\lg \lg n})$ برای آن بدست آمد و اکنون حتی الگوریتم با ضریب تقریب ثابت نیز برای آن وجود دارد (در حدود 2000) و حدس زده می‌شود الگوریتم با ضریب 2 نیز برای آن وجود داشته باشد.

در مورد این مسأله بهترین ضریب تقریب داده شده، همین $\frac{3}{2}$ است و حدس زده می‌شود که می‌توان ضریب بهتری مانند $\frac{4}{3}$ نیز ارائه داد. در عین حال ثابت شده الگوریتم تقریبی با ضریب تقریب به دلخواه خوب $\alpha = (1+\epsilon)$ برای این مسأله وجود ندارد مگر $\mathbf{P} = \mathbf{NP}$ باشد. برای حالت خاص تر این مسأله که مترِ تعریف شده، اقلیدسی باشد؛ یعنی فرض کنیم رئوس، نقاطی در صفحه اند و وزن هر یال، فاصله پاره خط بین آن دو راس است، ثابت شده است که الگوریتم‌های تقریبی با ضریب تقریب به دلخواه خوب $\alpha = (1+\epsilon)$ که $\epsilon > 0$ برای آن وجود دارد. در مورد این مدل از الگوریتم‌های تقریبی، در جلسات آینده صحبت خواهیم کرد.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

الگوریتم‌های تقریبی ۲: گردکردن داده و برنامه‌ریزی پویا

نگارندگان: کیانا عسگری و محبوبه عنایتی

در این جلسه، بحث الگوریتم‌های تقریبی را ادامه می‌دهیم و در مورد تکنیک گردکردن عددهای ورودی و استفاده از برنامه‌ریزی پویا در طراحی الگوریتم‌های تقریبی صحبت می‌کنیم.

در جلسه‌های قبلی الگوریتم حریصانه برای مسأله پوشش مجموعه‌ای و همچنین برای مسأله فروشنده دوره‌گرد و الگوریتم با ضریب تقریب دو برای مسأله برش بیشینه را دیدیم. سوالی که اکنون ممکن است مطرح شود این است که الگوریتم تقریبی ارائه شده برای این مسأله‌ها تا چه حد می‌توانند «خوب» باشند؛ یعنی آیا تقریب‌هایی که داریم قابل بهبود هستند یا خیر. در جلسات قبل اشاره‌ای به تعدادی نتایج تقریب‌ناپذیری شد که برای بعضی مسأله‌ها نمی‌توان الگوریتم تقریبی را از یک حدی بیش‌تر بهبود داد؛ اما درعین حال مسأله‌هایی هستند که می‌توان الگوریتم‌های تقریبی به دلخواه خوب برای آن‌ها ارائه کرد. عمدتاً مسأله‌هایی که الگوریتم تقریبی با تقریب به‌دلخواه خوب برای آن‌ها وجود دارد، مسأله‌هایی هستند که می‌توان از تکنیک‌های این جلسه برای حل آن‌ها استفاده کرد.

۱ گردکردن داده و برنامه‌ریزی پویا

برای اینکه درمورد الگوریتم‌های تقریبی به‌دلخواه خوب برای یک مسأله صحبت کنیم، دو اصطلاح زیر را معرفی می‌کنیم:

- **PTAS**[1]

  برای یک مسأله π، منظور از **PTAS**، یک خانواده از الگوریتم‌های چندجمله‌ای $\{\forall \epsilon \geq 0, A_\epsilon\}$ است که الگوریتم A_ϵ، یک االگوریتم $1+\epsilon$ تقریب برای مسأله بیشینه‌سازی و یا $1-\epsilon$ تقریب برای مسأله کمینه‌سازی است.

  دقت کنید الگوریتم‌هایی که در نظر می‌گیریم، بر حسب طول ورودی چندجمله‌ای هستند. و هیچ محدودیت دیگری روی الگوریتم‌ها نیست. به‌عنوان مثال ممکن است زمان اجرای الگوریتم‌ها بر حسب ϵ بد باشد؛ اما به ازای هر ϵ ثابت، زمان الگوریتم A_ϵ برحسب ورودی چندجمله‌ای است.

- **FPTAS**[2]

  منظور از **FPTAS**، یک **PTAS** است که زمان اجرای A_ϵ بر حسب $\frac{1}{\epsilon}$ هم چندجمله‌ای است.

اکنون چند مثال از روش بالا می‌بینیم.

تاکنون با تعدادی مسأله **انپی**ـ تمام[3] آشنا شدیم که با روش برنامه‌ریزی پویا قابل حل هستند و برای آن‌ها الگوریتم شبه چندجمله‌ای وجود دارد (یعنی زمان اجرای الگوریتم به مقدار اعداد ورودی وابسته است). بنابراین برحسب طول ورودی لزوماً چندجمله‌ای نیستند ولی اگر مقدار اعداد کم باشد،مثلاً مقدار بزرگ‌ترین عدد یک چندجمله‌ای بر حسب تعداد اعداد ورودی باشد، زمان اجرای الگوریتم چندجمله‌ای می‌شود.

[1] Polynomial Time Approximation Scheme
[2] Fully Polynomial Time Approximation Scheme
[3] $\mathbf{NP-complete}$

۱.۱ مسأله کوله‌پشتی

طبق توضیحات بالا، اگر می‌توانستیم برای یک ورودی مسأله کوله‌پشتی[۴] کاری کنیم که مقدار اعداد بر حسب تعداد اعداد از یک حدی بیش‌تر نباشد آنگاه الگوریتمی که با برنامه‌ریزی پویا داشتیم، چندجمله‌ای به حساب می‌آمد. پس هدف ما کاهش مقدار اعداد در عوض از دست دادن کمی دقت است تا در زمان چندجمله‌ای جوابی نزدیک به مقدار بهینه بیابیم.

برای رسیدن به این هدف، ایده این است که فقط تعدادی بیت پرارزش از اعداد ورودی را در نظر بگیریم و بقیه بیت‌های کم‌ارزش را دور بریزیم. در این‌صورت کمی دقت از دست می‌دهیم اما مقدار اعداد ورودی کاهش میابند و به نوعی بین دقت حل مسأله و زمان اجرا یک بده‌بستان[۵] وجود دارد؛ هرچقدر که بخواهیم دقیق‌تر باشیم، زمان اجرای الگوریتم افزایش می‌یابد.

مسأله کوله‌پشتی

ورودی. n عنصر با اندازه s_i و ارزش v_i داده شده. هم‌چنین ظرفیت کوله B است.

خروجی. زیرمجموعه‌ای از عناصر بصورتی که جمع اندازه آن‌ها حداکثر B باشد و ارزش آن‌ها تا جای ممکن زیاد باشد.

همان طور که قبلاً دیدیم، این مسأله را می‌توان به‌صورت‌های مختلف با برنامه‌ریزی پویا حل کرد؛ بعنوان مثال می‌توانیم زیرمسأله $V(i,b)$ را به معنای بیشینه ارزش یک زیرمجموعه از عناصر ۱ تا i با اندازه b تعریف کنیم و مقدار آن را بر حسب زیرمسأله‌های کوچک‌تر بیان کنیم. در روشی دیگر می‌توانیم زیرمسأله $A(i,p)$ را به معنای کمینه اندازه یک زیرمجموعه از عناصر ۱ تا i با ارزش p تعریف کنیم و مقدار آن را بر حسب زیرمسأله‌های کوچک‌تر بیان کنیم:

$$A(i+1,p) = \begin{cases} \min\{A(i,p), s_{i+1} + A(i,p-v_{i+1})\} & \text{if} \quad v_{i+1} \le p \\ A(i,p) & \text{oth} \end{cases}$$

و جواب بهینه از رابطه زیر محاسبه می‌شود:

$$OPT = \max_{j \le nP, A(n,j) \le B} \{j\}$$

در این‌صورت زمان اجرا، $O(n.nP)$ می‌شود که P برابر بیش‌ترین ارزش یک عنصر است.

اکنون می‌خواهیم روشی را بررسی کنیم که مقدار P را کاهش دهد تا در نهایت در قبال از دست دادن مقداری دقت، زمان اجرا چندجمله‌ای شود.

فرض می‌کنیم که برای تمام عناصر، $s_i \le B$ باشد؛ یعنی هیچ عنصری بی‌فایده نیست. اگر عنصری در این شرط صدق نکند می‌توانیم اول کار آن را دور بیندازیم.

در تعیین الگوریتم A_ϵ، کاری که باید انجام دهیم این است که مطمئن شویم بیش‌ترین وزن در بین عناصر، خیلی زیاد نیست. پس در حقیقت می‌خواهیم عددی مانند k درنظر بگیریم و سپس تمام وزن‌ها را به آن تقسیم کنیم:

$$\forall i : v'_i = \left\lfloor \frac{v_i}{k} \right\rfloor$$

اکنون باید مقدار k را تعیین کنیم. برای اینکه الگوریتم چندجمله‌ای شود، باید ضریب P در مقدار k موجود باشد. قرار می‌دهیم $k = \frac{P}{n}$.در این صورت بیش‌ترین مقدار ارزش عناصر، n می‌شود که چندجمله‌ای است. حال چون می‌خواهیم با کاهش مقدار ϵ، دقت الگوریتم افزایش یابد، درنهایت قرار می‌دهیم:

$$k = \frac{\epsilon P}{n}$$

[۴] Knapsack problem
[۵] Trade-off

سپس کافیست جواب بهینه را برای مسأله (s, v', B) با برنامه‌ریزی پویا بیابیم. پس در کل:

الگوریتم A_ϵ:

- $k = \frac{\epsilon P}{n}$
- $\forall i : v'_i = \lfloor \frac{v_i}{k} \rfloor$
- جواب بهینه را برای مسأله (s, v', B) بیابید و آن را O' بنامید.
- O' را خروجی بدهید.

اکنون باید ضریب تقریب و زمان اجرا را بررسی کنیم.

زمان اجرا : دیدیم که زمان اجرای الگوریتم برنامه‌ریزی پویا، $O(n^۲ . \max_i v_i)$ بود. در این صورت، اکنون زمان اجرا

$$O(n^۲ . \left\lfloor \frac{P}{K} \right\rfloor) = O(n^۲ . \left\lfloor \frac{n}{\epsilon} \right\rfloor)$$

می‌شود که بر حسب n و $\frac{۱}{\epsilon}$ چندجمله‌ای است.

کیفیت تقریب:

قضیه ۱.

$$v(O') \geq (۱ - \epsilon) v(O)$$

اثبات. با توجه به تعریف v' و این‌که $v_i - k \leq k v'_i \leq v_i$ و همچنین این نکته که تعداد اعضای O حداکثر n تاست، داریم:

$$v(O') \geq k v'(O') \geq k v'(O) \geq v(O) - nk = v(O) - \epsilon P$$

حال چون $v(O) \geq P$ است نتیجه می‌شود:

$$v(O') \geq P - \epsilon P \geq (۱ - \epsilon) v(O)$$

□

پس در کل، A_ϵ ها **FPTAS** هستند.

۲.۱ حالت کلی مسأله

اکنون می‌خواهیم ببینیم در حالت کلی چه مسأله هایی، **FPTAS** دارند. در مورد اکثر مسأله های ان پی، **FPTAS** وجود ندارد و حتی الگوریتم تقریبی خوبی ندارند. دیدیم که اگر مسأله‌ای **تمام انپی_سخت** باشد،آنگاه الگوریتم شبه‌چندجمله‌ای برای آن وجود ندارد. اکنون می‌خواهیم حکم مشابهی را ثابت کنیم. فرض می‌کنیم تمام اعداد، صحیح هستند.

قضیه ۲. *فرض کنید π یک مسأله بهینه‌سازی (کمینه‌سازی) **انپی_سخت** است و p یک چندجمله‌ای باشد به طوری که برای هر ورودی I از π،*

$$OPT(I) < p(|I_u|)$$

*در این‌صورت اگر π یک **FPTAS** داشته باشد، یک الگوریتم شبه‌چندجمله‌ای دارد. (منظور از $|I_u|$، اندازه ورودی است زمانی که بصورت یکانی نمایش داده شود.)*

دقت کنید **FPTAS**، الگوریتمی است که با هر ϵ، یک تقریبی از جواب بهینه را خروجی می‌دهد. اما برای الگوریتم شبه چندجمله‌ای، می‌خواهیم جواب دقیق را بدست آوریم. نکته این است که مقدار ϵ را درست تعیین کنیم؛ اگر ϵ را به قدر کافی کوچک اختیار کنیم، آن وقت خود **FPTAS**، جواب درست را خروجی خواهد داد؛ در حقیقت چون تمام اعداد صحیح هستند، اگر میزان خطای **FPTAS** کم‌تر از یک باشد،آنگاه خودبه‌خود باید جواب بهینه را خروجی بدهد.

اثبات. فرض کنید خانواده A_ϵ یک **FPTAS** برای π باشد که زمان اجرای A_ϵ روی I، $q(|I|, \frac{1}{\epsilon})$ باشد که q یک چندجمله‌ای است.

روی ورودی I، قرار دهید:

$$\epsilon = \frac{1}{p(|I_u|)}$$

(همان کران بالای مقدار تابع هدف حکم). سپس الگوریتم A_ϵ را روی I اجرا کنید. طبق کران بالای فرض حکم برای تابع هدف داریم:

$$\text{مقدار تابع هدف برای خروجی} \leq (1+\epsilon)OPT(I) < OPT(I) + \epsilon P(|I_u|) = OPT(I) + 1$$

پس نتیجه می‌شود که خروجی الگوریتم باید همان OPT باشد. پس A_ϵ جواب بهینه را برمی‌گرداند.

در این‌صوت زمان اجرای الگوریتم،

$$q(|I|, p(|I_u|))$$

می‌شود که بر حسب طول ورودی نمایش یکانی I چندجمله‌ای است؛ پس در کل A_ϵ یک الگوریتم شبه‌چندجمله‌ای است. □

نتیجه: اگر π یک مسأله قویاً **انپی‌ـسخت** باشد و یک‌سری شرط‌های ضعیف دیگر را داشته باشد، آنگاه **FPTAS** ندارد مگر اینکه $\mathbf{P} = \mathbf{NP}$ باشد.

۳.۱ مسأله جای‌گذاری در سطل

ورودی. n عنصر با اندازه‌های $a_1, \ldots, a_n \in (0, 1]$ داده شده است.

خروجی. می‌خواهیم این عناصر را به‌گونه‌ای در سطل‌های با اندازه ۱ جای دهیم به طوری که تا جای ممکن تعداد کم‌تری سطل استفاده کنیم.

مسأله جای‌گذاری در سطل[6] یکی از مسائل کلاسیک و محوری بهینه‌سازی و بهینه‌سازی ترکیبیاتی است که زیاد بررسی شده‌است و کاربردهای زیادی دارد. برای این مسأله، الگوریتم تقریبی با ضریب تقریب ۲ وجود دارد که الگوریتم ساده‌ای است. الگوریتم FirstFit یک الگوریتم ۲ـ تقریب برای این مسأله است.

FirstFit: تعدادی سطل که تاکنون از آن‌ها استفاده شده را در نظر بگیرید؛ در ابتدا یک سطل داریم که خالی است و سپس عناصر را یکی یکی (با ترتیب دل‌خواه) در نظر می‌گیریم و در مرحله iام، عنصر a_i را برمی‌داریم و در اولین سطلی که به اندازه‌ی کافی ظرفیت داشت، قرار می‌دهیم و اگر چنین سطی وجود نداشت، یک سطل جدید اضافه می‌کنیم و a_i را به تنهایی در آن سطل قرار می‌دهیم.

قضیه ۳. *الگوریتم* FirstFit*، ۲ـ تقریب است*

اثبات. فرض کنید الگوریتم از m سطل استفاده کند. هم‌چنین تمامی اعداد صحیح هستند.

[6] Bin Packing

ادعا ۱. *اگر الگوریتم* FirstFit *از m سطل استفاده کرده باشد، تقریباً حدّاقل نیمی از حجم تمامی سطل‌ها، به جز حدّاکثر یک سطل، اشغال شده است. به بیان دیگر، حدّاقل $m-1$ سطل حدّاقل نصفشان پر است و غیر ممکن است که دو سطل بیش از نصف فضایشان خالی باشد.*

اثبات. هنگامی که سطل جدیدی اضافه می‌شود، تا حدّاقل نیمی از آن پر نشده، طبق الگوریتم FirstFit دلیلی ندارد که سطل جدید دیگری پس از آن اضافه گردد و تنها کم‌تر از نصف حجمش اشغال شده باشد. چرا که عنصری (یا عناصری) که در سطل جدیدتر جا می‌شود و کمتر از نصف آن را می‌گیرد، در سطل قبلی (جدید) نیز جا می‌گیرد (چون حدّاقل نصف فضای سطل قبلی خالی است). □

بنابراین طبق ادعای ۱ داریم:

$$OPT \leq \sum_{i=1}^{n} a_i < \frac{m-1}{2}$$

پس:

$$m - 1 < 2OPT \xrightarrow{\text{تمامی اعداد صحیح‌اند}} m \leq 2OPT$$

□

قضیه ۴. *برای هر $\epsilon > 0$، الگوریتم با ضریب تقریب $\frac{3}{2} - \epsilon$ برای مسأله‌ی جای‌گذاری در سطل وجود ندارد؛ مگر این‌که $\boldsymbol{P = NP}$.*

اثبات. به برهان خلف فرض کنید چنین الگوریتمی وجود داشته باشد. در ادامه با استفاده از مسأله‌ی افراز[7] به تناقض می‌رسیم.

تعریف ۱. (***مسأله‌ی افراز***) *n عدد صحیح و مثبت $a_1, a_2, \ldots, a_n$ داده شده است. هدفْ افراز مجموعه‌ی $\{1, 2, \ldots, n\}$ به دو مجموعه‌ی S_1 و S_2 است به طوری که $\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i$.*

می‌خواهیم با تحویل[8] مسأله‌ی افراز به مسأله‌ی جای‌گذاری در سطل، جوابی برایش بیابیم. تأکید می‌کنیم که تمامی اعدادْ صحیح هستند.

می‌توان a_iها را طوری نرمال‌سازی کرد که جمع آن‌ها ۲ شود. a_iهای جدیدْ ورودی مسأله‌ی جای‌گذاری در سطل خواهند بود. بنابراین جواب مسأله‌ی افراز «بله» است اگر و تنها اگر تمامی عناصر مسأله‌ی جای‌گذاری در سطل در دو سطل جا گیرند؛ چرا که در این صورت مجموع اندازه‌ی دو سطل و اندازه‌ی a_iها برابر ۲ است و هر دو سطل کاملاً پر می‌شوند.

اکنون اگر الگوریتمی با ضریب تقریب ۱٫۴ داشته باشیم، قادریم مسأله‌ی افراز را در زمان اجرای چندجمله‌ای حل کنیم. اگر جواب الگوریتم حدّاقل ۳ باشد، جواب مسأله‌ی افراز «خیر» خواهد بود و در غیر این صورت «بله». پس اگر جواب مسأله‌ی افراز «بله» باشد، جواب الگوریتم حدّاکثر $1.4 \times 2 = 2.8$ است که با توجه به صحیح بودن تعداد سطل‌ها، ۲ را خروجی خواهد داد. ولی از قبل می‌دانیم که مسأله‌ی افراز **انپی**ـتمام[9] است و در نتیجه الگوریتمی چندجمله‌ای ندارد مگر $\mathbf{P = NP}$. پس فرض خلف باطل است. □

[7] Partition
[8] reduce
[9] **NP**-complete

۴.۱ PTAS مجانبی[10]

شاید از خود بپرسید که چرا چنین مسئله‌ای (مسأله‌ی افراز) را در این‌جا طرح کردیم در حالی که هدف از ابتدا یافتن الگوریتمی با ضریب تقریب $(\epsilon + ۱)$ بود. نکته این‌جاست که در مثال اخیر تفاوت بین ۲ و ۳ بسیار مهم بود و جواب مسأله در هر یک از این حالات اطلاعات مهمی به ما می‌داد؛ از جمله این که حل یک مسأله‌ی **انپی‌ـ سخت** امکان‌پذیر است یا نه. اگر اجازه دهیم الگوریتم تقریبی ما به اندازه‌ی ۱ واحد اشتباه کند، یعنی با فرض این‌که I مجموعه‌ی a_iها $(۱ \leq i \leq n)$ و $f(A_\epsilon(I))$ برابر تعداد سطل‌های خروجی الگوریتم A_ϵ باشد داشته باشیم:

$$f(A_\epsilon(I)) \leq (۱ + \epsilon).OPT(I) + ۱$$

آنگاه خانواده‌ای از الگوریتم‌ها خواهیم داشت که اصطلاحاً **PTAS مجانبی** نامیده می‌شوند. به بیانی دیگر، الگوریتم‌هایی چندجمله‌ای که هر کدام به ازای یک $\epsilon > ۰$ ثابت خاص خودش، جواب را با هزینه‌ی حدّاکثر $(۱ + \epsilon).OPT + ۱$ بیابد و برای آن نمونه[11] هایی از مسأله که جواب بهینه‌اش (OPT) آن) از یک $n_۰$ی بزرگ‌تر است، ضریب تقریب $(\epsilon + ۱)$ داشته باشد، اعضایی از خانواده‌ی **PTAS مجانبی** هستند. در ادامه به اثبات وجود چنین خانواده‌ای از الگوریتم‌ها خواهیم پرداخت. در واقع ابتدا دو محدودیت برای مسأله‌ی جای‌گذاری در سطل در نظر گرفته و سپس در دو مرحله این دو محدودیت را حذف می‌کنیم تا به حالت کلی آن برسیم.

لم ۱. *فرض کنید $\epsilon > ۰$ و k عددی صحیح و مثبت است. حالت خاصی از مسأله‌ی جای‌گذاری در سطل را در نظر بگیرید که اندازه‌ی هر عنصر a_i $(۱ \leq i \leq n)$ حدّاقل ϵ است و تعداد اندازه‌های متمایز حدّاکثر k است (دو محدودیت). یک الگوریتم چندجمله‌ای برای این حالت خاص مسأله‌ی جای‌گذاری در سطل وجود دارد.*

در واقع ایده این است که تمام حالات ممکن را چک کنیم. با توجه به این که اندازه‌ی عناصر از یک مقداری (ϵ) بیش‌تر نیست و این‌که تعداد اندازه‌های متمایز نیز از k بیش‌تر نیست، تعداد کل حالاتْ یک چندجمله‌ای بر حسب n خواهد بود (که البته درجه‌ی این چندجمله‌ای بسیار بزرگ است!).

اثبات. با توجه به این‌که اندازه‌ی هر سطل برابر ۱ و اندازه‌ی هر عنصر حدّاقل ϵ است، تعداد عناصری که می‌توانند داخل یک سطل قرار گیرند حدّاکثر $M = \lfloor \frac{۱}{\epsilon} \rfloor$ است. بنابراین حدّاکثر $R = \binom{M+k}{M}$ نوع سطل متمایز وجود دارد. دو سطل متمایزند اگر به ازای حدّاقل یک s $(۱ \leq s \leq k)$ تعداد عناصرِ با اندازه‌ی s در یکی، با تعداد عناصرِ با اندازه‌ی s در دیگری متفاوت باشد. به وضح R عددی ثابت بر حسب ϵ و k است. به عبارتی دیگر، R همان تعداد حالات ممکن مقداردهی صحیح و مثبت x_iها در نامعادله‌ی ترکیبیاتی زیر است:

$$x_۱ + x_۲ + \cdots + x_k \leq M$$

از طرف دیگر، تعداد کل سطل‌های استفاده‌شده حدّاکثر n می‌باشد. بنابراین تعداد کلّ جواب‌های شدنی[12] حدّاکثر $P = \binom{n+R}{R}$ بوده که بر حسب n چندجمله‌ای می‌باشد. استدلالی مشابهِ آن‌چه که برای محاسبه‌ی R به کار گرفته شد برای محاسبه‌ی P نیز برقرار است. هم‌چنین واضح است که $P \leq (n + R)^R$ و مقدار آن از مرتبه‌ی چندجمله‌ای بر حسب n است. پس می‌توانیم در زمان چندجمله‌ای تمام حالات را بررسی کنیم و جواب دقیق را به‌دست آوریم.

□

لم ۲. *اگر محدویت تعداد اندازه‌ها را از لم ۱ حذف کنیم، آنگاه الگوریتمی چندجمله‌ای وجود دارد که مسأله را با تقریب $\epsilon + ۱$ حل می‌کند (پس دقیق‌بودنی را که در جواب حالت قبلی داشتیم از دست خواهیم داد).*

ایده‌ی اثبات این است که مسأله را به حالت قبل تحویل کنیم؛ به این معنا که تعداد اندازه‌های متمایز عدد ثابتی شود. پس باید بر حسب تقریبی که لازم داریم این عدد ثابت را تنظیم کنیم.

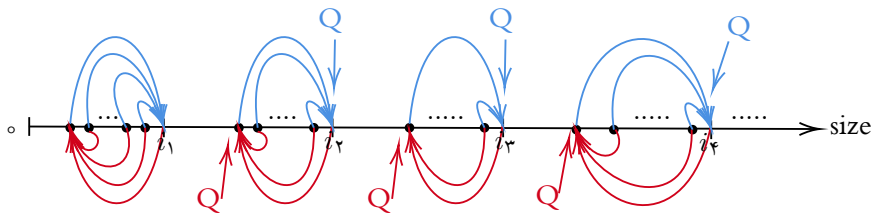
[10] **asymptotic PTAS**
[11] instance
[12] feasible

اثبات. – عناصر نمونه‌ی ورودی (I) را بر حسب اندازه مرتب کنید.

– این مجموعه‌ی مرتب‌شده را به $k = \lfloor \frac{۱}{\epsilon^۲} \rfloor$ گروه که هر کدام حدّاکثر $Q = \lfloor n\epsilon^۲ \rfloor$ عضو دارد افراز کنید. فقط گروه آخر ممکن است کم‌تر از Q عنصر داشته باشد. هم‌چنین ممکن است دو گروه عناصری با اندازه‌ی یکسان داشته باشند.

– نمونه‌ی J را از روی I به این صورت می‌سازیم که اندازه‌ی هر عنصر را به اندازه‌ی بزرگ‌ترین عنصر گروهش گرد می‌کنیم (بازه‌ی اندازه‌ی گروه‌ها، یعنی اختلاف اندازه کوچک‌ترین و بزرگ‌ترین عنصر، ممکن است با یک‌دیگر متفاوت باشد). بنابراین J حدّاکثر k اندازه متمایز دارد و با استفاده از لم ۱ به طور دقیق قابل حل است.

– نشان می‌دهیم $OPT(J) \leq (۱ + \epsilon).OPT(I)$:



شکل ۱: Q عنصر گروه‌های نمونه‌ی J را به بیش‌ترین اندازه‌ی موجود در گروه خود، و Q عنصر گروه‌های J' را به کم‌ترین اندازه‌ی موجود در گروه خود گرد می‌کنیم.

– نمونه‌ی J' را از روی I به این صورت می‌سازیم که اندازه‌ی هر عنصر را به اندازه‌ی کوچک‌ترین عنصر گروهش گرد می‌کنیم. داریم:

$$OPT(J') \leq OPT(I)$$

چرا که اندازه‌ی هر عنصر در J' کوچک‌تر یا مساوی اندازه‌ی عنصر نظیرش در I می‌باشد. هم‌چنین داریم:

$$OPT(J) \leq OPT(J') + Q$$

چون اگر گروه i_t را در نمونه‌ی J و گروه $i_{t+۱}$ را در نمونه‌ی J' در نظر بگیریم، اندازه‌ی Q عنصر در اولی به مقدار عنصری با اندازه‌ی بیشینه در آن گروه و در دومی اندازه‌ی Q عنصر به مقدار عنصری با اندازه‌ی کمینه در آن گروه گرد شده‌اند و این دو عنصر، یکی با اندازه‌ی بیشینه و دیگری با اندازه‌ی کمینه که به ترتیب در J و J' هستند، پشت سر هم قرار گرفته‌اند. می‌توان همه‌ی Q عنصر i_t را جایگزین Q عنصر $i_{t+۱}$ در سطل‌های جواب بهینه‌ی J' کرد؛ چرا که اندازه‌ی هر کدام کوچک‌تر از اندازه‌ی گردشده‌ی عناصر $i_{t+۱}$ در J' است. بنابراین تنها گروه عناصری از J که نمی‌توانند جایگزین عناصر دیگری در $OPT(J')$ شوند گروه آخر خواهد بود که می‌توان در بدترین حالت برای هر عنصرش یک سطل مجزا در نظر گرفت، یعنی حدّاکثر Q سطل و در این صورت بیش‌ترین تعداد سطل Q است. پس تعداد سطل‌های $OPT(J)$ حدّاکثر Q تا بیش‌تر از تعداد سطل‌های $OPT(J')$ است. پس:

$$OPT(J) \leq OPT(J') + Q \leq OPT(I) + Q$$

$$\xrightarrow[\epsilon n \leq OPT(I)]{\lfloor n\epsilon^۲ \rfloor \leq \epsilon.(\epsilon n)} OPT(J) \leq (۱ + \epsilon).OPT(I)$$

□

اکنون بدون هیچ یک از دو محدودیت پیشین به طرح الگوریتم **PTAS مجانبی** برای مسأله‌ی جای‌گذاری در سطل خواهیم پرداخت و در نهایت ثابت می‌کنیم که چنین خانواده‌ای از الگوریتم‌ها وجود دارد. روند این الگوریتم بدین شرح است:

- عناصری را که اندازه‌ی کمتر از ϵ دارند کنار بگذارید تا نمونه‌ی I' به‌دست آید.
- I' را با حدّاکثر $(۱+\epsilon).OPT(I')$ سطل حل کنید (طبق لم ۲ چنین عملی امکان‌پذیر است).
- عناصر کوچک، یعنی آن‌هایی را که در مرحله‌ی اوّل نادیده گرفتیم، با الگوریتم FirstFit اضافه می‌کنیم.
- اگر در مرحله‌ی قبل سطل جدیدی اضافه نشد، کار تمام است. زیرا:

$$(۱+\epsilon).OPT(I') \leq (۱+\epsilon).OPT(I) \leq (۱+\epsilon).OPT(I)+۱$$

- در غیر این صورت (اگر سطل(های) جدیدی اضافه گردد)،
  - فرض کنید M برابر تعداد کلّ سطل‌های استفاده‌شده باشد.
  - حجم حدّاقل $M-۱$ سطل، به اندازه‌ی حدّاقل $\epsilon-۱$ اشغال شده است. چون دستِ کم یک سطل جدید اضافه شده و در نتیجه حدّاقل یک عنصر کوچک، که اندازه‌اش کوچک‌تر از ϵ است، در هیچ یک از سطل‌های اولیه جا نشده و نیاز به سطل جدید داشته. پس حجم فضای خالی سطل‌های اوّلیه کم‌تر از ϵ است. بنابراین

$$(M-۱).(۱-\epsilon) < OPT \xRightarrow{\epsilon \leq \frac{۱}{۲}} M < \frac{OPT}{۱-\epsilon}+۱ \leq (۱+۲\epsilon).OPT+۱$$

بنابراین جواب الگوریتم $A_{\frac{\epsilon}{۲}}$ حدّاکثر برابر $(۱+\epsilon).OPT+۱$ است.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

الگوریتم‌های تقریبی ۳: روش‌های مبتنی بر برنامه‌ریزی خطی — نگارنده: سید پوریا فاطمی

در این جلسه در مورد روش‌های مبتنی بر برنامه‌ریزی خطی[۱] برای طراحی الگوریتم‌های تقریبی[۲] صحبت می‌کنیم و به طور خاص راجع به روش گرد کردن[۳] و روش اولیه ـ دوگان[۴] حرف خواهیم زد. هر دوی این روش‌ها را روی مسئله‌ی پوشش مجموعه‌ای[۵] بررسی خواهیم کرد. برای یادآوری تعریف مسئله‌ی پوشش مجموعه‌ای به صورت زیر است.

ورودی: مجموعه‌ای از عناصر $\mathcal{U} = \{e_1, e_2, ..., e_n\}$ و مجموعه‌ای از زیرمجموعه‌های $\mathcal{U}$ به صورت $\mathcal{Q} = \{S_1, S_2, ..., S_m\}$ که برای هر $i = 1, 2, ..., m$ داریم: $S_i \subseteq \mathcal{U}$ و یک تابع هزینه هم برای هر مجموعه به صورت روبه‌رو داریم: $c : \mathcal{Q} \to \mathcal{R}_+$
فرض این است که هر e حداکثر در f تا از S_iها ظاهر شده است.

خروجی: به دنبال پیدا کردن تعدادی S_i هستیم که اجتماعشان $\mathcal{U}$ شود و هزینه‌شان تا جای ممکن کم باشد. به زبان ریاضی داریم:

$$T \subseteq \mathcal{Q}, \quad \bigcup_{A \in T} A = \mathcal{U}, \quad \text{کمینه شود.} \quad c(T) = \sum_{A \in T} c(A)$$

در این جلسه می‌خواهیم این مسئله را به صورت یک برنامه‌ریزی صحیح[۶] و بعد از آن به صورت یک برنامه‌ریزی خطی مدل کنیم.

۱ مدل‌سازی مسئله‌ی پوشش مجموعه‌ای

مدل‌سازی مسئله به کمک برنامه‌ریزی صحیح به صورت زیر است.

$$\begin{cases} \min \sum_{S \in \mathcal{Q}} c_S x_S \\ s.t. \ \sum_{S \ni e} x_S \geqslant 1 & \forall\, e \in \mathcal{U} \\ \quad x_S \in \{0, 1\} & \forall\, S \in \mathcal{Q} \end{cases}$$

می‌دانیم یک مسئله‌ی برنامه‌ریزی صحیح را لزوما نمی‌توانیم در زمان چندجمله‌ای حل کنیم ولی برنامه‌ریزی خطی را می‌توانیم در زمان چندجمله‌ای حل کنیم. اگر برنامه ریزی صحیح بالا را ریلکس[۷] کنیم به برنامه‌ریزی خطی زیر می‌رسیم:

$$\begin{cases} \min \sum_{S \in \mathcal{Q}} c_S x_S \\ s.t. \ \sum_{S \ni e} x_S \geqslant 1 & \forall\, e \in \mathcal{U} \\ \quad 0 \leqslant x_S \leqslant 1 & \forall\, S \in \mathcal{Q} \end{cases}$$

[1] Linear Programming
[2] Approximate algorithms
[3] Rounding
[4] Primal-Dual
[5] Set cover
[6] Integer Programming
[7] Relax

البته می‌توانیم شرط کران بالای x_S را نیز حذف کرد زیرا در جواب بهینه هیچ‌گاه لزوم ندارد که x_S از یک بیش‌تر شود و قید $x_S \leqslant 1$ خودبه‌خود برقرار است. در واقع چون هدف، کمینه کردن تابع هدف[۸] است و هر x_S ای که بزرگ‌تر از یک باشد و در یک جواب شدنی[۹] برنامه‌ریزی خطی قرار داشته باشد، می توان مقدار x_S را به یک تغییر دهید و باز هم جواب مربوطه جواب شدنی باقی می‌ماند و مقدار تابع هدف کمتر می‌شود. پس مسئله به برنامه‌ریزی خطی زیر تبدیل می‌شود.

$$\begin{cases} \min \sum_{S \in \mathcal{Q}} c_S x_S \\ s.t. \ \sum_{S \ni e} x_S \geqslant 1 & \forall \, e \in \mathcal{U} \\ \quad x_S \geqslant 0 & \forall \, S \in \mathcal{Q} \end{cases}$$

حال می‌توانیم برنامه‌ریزی خطی بالا را حل کنیم ولی نکته این است که حل کردن برنامه‌ریزی خطی بالا لزوما جوابی برای مسئله‌ی اصلی به ما نمی‌دهد زیرا در برنامه‌ریزی خطی بالا متغیرها عدد حقیقی می‌باشند درحالی که ما به مقادیر صحیح برای x_S نیاز داریم. حال می‌خواهیم از روی حل برنامه‌ریزی خطی بالا جوابی پیدا کنیم که جواب مسئله اصلی باشد و نشان دهیم جواب بهینه آن از جواب بهینه مسئله‌ی اصلی خیلی بدتر نیست!

روش تحلیل به این‌گونه است که جواب بهینه برنامه ریزی خطی را اگر با OPT_{LP} و جواب بهینه برنامه‌ریزی صحیح را اگر با OPT_{IP} نشان دهیم داریم:

$$OPT_{LP} \leqslant OPT_{IP}$$

زیرا داریم تابع هدف را کمینه می‌کنیم و در برنامه‌ریزی خطی شروط کم‌تری داریم و متغیرها آزادی عمل بیشتری دارند و به همین دلیل جواب بهینه برنامه‌ریزی خطی می‌تواند کمتر از جواب بهینه برنامه‌ریزی صحیح شود.

حال اگر هزینه جوابی که الگوریتم $\mathcal{A}$ روی هر ورودی تولید می‌کند را با $c(\mathcal{A})$ نشان دهیم. ما به دنبال الگوریتمی هستیم که داشته باشیم:

$$c(\mathcal{A}) \leqslant \alpha . OPT_{LP} \Rightarrow c(\mathcal{A}) \leqslant \alpha . OPT_{IP}$$

نکته این است برای طراحی یک الگوریتم تقریبی مبتنی بر برنامه‌ریزی خطی و برای تحلیل و پیدا کردن ضریب تقریب[۱۰] باید بتوانیم جواب تقریبی را با جواب بهینه مسئله اصلی مقایسه کنیم ولی جواب بهینه مسئله را نداریم و برای همین جواب تقریبی بدست آمده را با OPT_{LP} مقایسه می‌کنیم و اگر ثابت کنیم مثلا جواب بهینه الگوریتم $\mathcal{A}$ از ۵ برابر جواب بهینه برنامه‌ریزی خطی بیش‌تر نیست به صورت خودبه‌خود اثبات شده است که جواب بهینه الگوریتم $\mathcal{A}$ از ۵ برابر جواب بهینه برنامه‌ریزی صحیح که همان مسئله اصلی هست نیز بیشتر نیست.

[8] Objective function
[9] Feasible
[10] Approximation factor

۲ روش گرد کردن قطعی

برای حل مسئله پوشش مجموعه‌ای الگوریتم زیر($\mathcal{A}$) را در نظر بگیرید.

۱. برنامه‌ریزی خطی تعریف شده در بالا را حل می‌کنیم و جواب را x^* می‌نامیم.

۲. برای هر مجموعه‌ی S اگر $x^*_S \geqslant \frac{1}{f}$ بود، مجموعه‌ی S را انتخاب کنید.

قضیه ۱. *الگوریتم بالا f تقریب است.*

اثبات. ابتدا دقت کنید در این مسئله ورودی الگوریتم را به صورت غیرصریح[11] در نظر گرفته‌ایم یعنی فرض می‌کنیم ورودی الگوریتم را می‌دانیم. فرض کنید T مجموعه‌ی همه مجموعه‌های انتخاب شده باشد. حال برای هزینه الگوریتم داریم:

$$c(\mathcal{A}) = \sum_{S \in T} c_S = \sum_{S \in T} f \times \frac{1}{f} c_S = f \sum_{S \in T} \frac{1}{f} c_S \leqslant f \sum_{S \in T} c_S x^*_S$$

علت نامساوی آخر این است که مجموعه‌هایی که انتخاب می‌کنیم خاصیت $x^*_S \geqslant \frac{1}{f}$ را دارند. حال چون همه‌ی x_Sها و همه‌ی هزینه‌ها مثبت‌اند داریم:

$$c(\mathcal{A}) \leqslant f \sum_{S \in T} c_S x^*_S \leqslant f \underbrace{\sum_{S \in Q} c_S x^*_S}_{\text{تابع هدف } LP} = f.OPT_{LP} \leqslant f.OPT_{IP} \Longrightarrow c(\mathcal{A}) \leqslant f.OPT_{IP}$$

پس حکم مربوطه اثبات شد. □

پس ابتدا برنامه‌ریزی خطی را حل می‌کنیم سپس به کمک گرد کردن مجموعه‌هایی که x_Sشان از یک حدی بیشتر است را به بالا گرد می‌کنیم و از جواب حقیقی موجود یک جواب صحیح می‌سازیم.

روش بالا را می‌توانیم به صورت تصادفی نیز انجام دهیم و به نظر نیز مفید می‌آید زیرا در برنامه‌ریزی خطی تعریف‌شده برای هر x_S داریم $0 \leqslant x_S \leqslant 1$ پس به صورت طبیعی می‌توانیم به چشم احتمال به x_Sها نگاه کنیم. یعنی اگر در جواب برنامه‌ریزی خطی، x_S یک مجموعه خیلی نزدیک به یک باشد این شهود را منتقل می‌کند که برداشتن مجموعه S مفید است و اگر نزدیک صفر باشد برعکس.

[11] Implicit

۳ روش گرد کردن تصادفی

برای حل مسئله پوشش مجموعه‌ای الگوریتم زیر($\mathcal{A}'$) را در نظر بگیرید.

۱. برنامه‌ریزی خطی تعریف شده را حل می‌کنیم و جواب را x^* می‌نامیم.

۲. مجموعه‌ی S را با احتمال x^*_S انتخاب می‌کنیم.

امید ریاضی هزینه الگوریتم مطابق میل ما و به صورت زیر است:

$$E[c(\mathcal{A}')] = \sum_{S \in \mathcal{Q}} x^*_S c_S = OPT_{LP} \leqslant OPT_{IP}$$

درواقع طراحی الگوریتم به نحوی بود که امید ریاضی برابر OPT_{LP} شود.

ولی در این‌جا مسئله مهم این است که ممکن است یک عنصر پوشانده نشود. یعنی ممکن است در فرآیند الگوریتم برای یک e شرط $\sum_{S \ni e} x_S \geqslant 1$ برقرار نباشد و درواقع یک عضو پوشانده نشود. پس ممکن است الگوریتم گفته‌شده در بالا جواب درست ندهد. البته می‌توانیم ادعا کنیم که هر عنصر در امید ریاضی پوشانده می‌شوند. در واقع اگر متغیرهای صفر و یکی که متناظر با انتخاب‌های مجموعه‌ها طبق الگوریتم x'_S بنامیم داریم: $\sum_{S \ni e} E[x'_S] \geqslant 1$

حال می‌خواهیم احتمال پوشانده نشدن یک عنصر را بدست آوریم.(در این قسمت برای اشتباه نشدن اسم عنصر با عدد نپر اسامی عنصرها را با z نشان می‌دهیم.) داریم:

$$\begin{aligned}
\mathbb{P}(\text{پوشانده نشدن یک عنصر}) &= \prod_{S \ni z} (1 - x^*_S) \\
(1 - x \leqslant e^{-x}) \Rightarrow &\leqslant \prod_{S \ni z} e^{-x^*_S} \\
&= e^{-\sum_{S \ni z} x^*_S} \\
(\sum_{S \ni z} x_S \geqslant 1) \Rightarrow &\leqslant \frac{1}{e} \implies \mathbb{P}(\text{پوشانده نشدن یک عنصر}) \leqslant \frac{1}{e}
\end{aligned}$$

پس احتمال پوشانده شدن هر عنصر حداقل $1 - \frac{1}{e}$ می‌باشد ولی مطلوب ما پوشانده شدن همه‌ی عنصرها می‌باشد. برای این‌که الگوریتم بالا خطا نداشته باشد و همه‌ی عنصرها را بپوشاند الگوریتم زیر($\mathcal{A}''$) را انجام می‌دهیم.

گام دوم الگوریتم $\mathcal{A}'$ را $\theta(\log n)$ بار تکرار می‌کنیم و همه‌ی مجموعه‌های همه‌ی تکرارها را برمی‌داریم.

می‌دانیم اگر گام دوم الگوریتم $\mathcal{A}'$ را $\theta(\log n)$ بار تکرار کنیم امید ریاضی هزینه حداکثر $\theta(\log n)$ برابر می‌شود پس داریم:

$$E[c(\mathcal{A}'')] = O(\log n).OPT_{IP}$$

اگر تکرار الگوریتم را به صورت دقیق‌تر $k.\log n$ بار درنظر بگیریم، احتمال پوشانده نشدن یک عنصر در الگوریتم $\mathcal{A}''$ حداکثر برابر $(\frac{1}{e})^{k.\log n} = \frac{1}{n^k}$ می‌باشد. پس به کمک کران اجتماع[۱۲] به احتمال حداقل $1 - \frac{1}{n^{k-1}}$ همه‌ی عناصر پوشانده می‌شوند. پس در این‌جا الگوریتمی تقریبی داریم که ضریت تقریب الگوریتم $O(\log n)$ است.(مشابه الگوریتم حریصانه[۱۳]) و به احتمال بالا[۱۴] همه‌ی عناصر را می‌پوشاند.

نکته این است که الگوریتم‌های تقریبی مبتنی بر برنامه‌ریزی خطی معمولا جواب‌های خوبی در عمل به ما می‌دهند زیرا ساختار مسئله تا حد خوبی در فرمول‌بندی برنامه‌ریزی خطی مدل شده است و ممکن است در عمل جواب‌ها از ضریب تقریب بدست آمده نیز بهتر باشند.

[12] Union bound
[13] Greedy
[14] With high probability

۴ روش اولیه_دوگان

دوگان برنامه‌ریزی خطی تعریف‌شده در بالا را می‌نویسیم داریم: (y_e متغیر تعریفی برای هرکدام از قیدهای مربوط به عناصر مسئله‌ی اصلی می‌باشد.)

$$P: \begin{cases} \min \sum_{S\in\mathcal{Q}} c_S x_S & \\ s.t.\ \sum_{S\ni e} x_S \geqslant 1 & \forall\, e\in\mathcal{U} \\ x_S \geqslant 0 & \forall\, S\in\mathcal{Q} \end{cases} \Longrightarrow D: \begin{cases} \max \sum_{e\in\mathcal{U}} y_e & \\ s.t.\ \sum_{e\in S} y_e \leqslant c_S & \forall\, S\in\mathcal{Q} \\ y_e \geqslant 0 & \forall\, e\in\mathcal{U} \end{cases}$$

در روش اولیه_دوگان به صورت هم‌زمان یک جواب برای مسئله‌ی اولیه و یک جواب برای مسئله‌ی دوگان بدست می‌آوریم و با استفاده از این دو جواب تحلیلی انجام می‌دهیم که جواب مسئله‌ی P خیلی بدتر از جواب مسئله‌ی اصلی نیست. در واقع زوج جواب‌های برنامه‌ریزی خطی اولیه و دوگان را طوری می‌سازیم که تابع هدف‌هایشان خیلی از هم فاصله نداشته باشد بعد به کمکی تحلیلی که در زیر انجام خواهیم داد نتیجه می‌گیریم که مقدار تابع هدف جواب مسئله‌ی P با جواب اصلی مسئله‌ی اصلی پوشش مجموعه‌ای نیز خیلی فاصله ندارد. به زبان دقیق‌تر داریم:

زوج جواب x^* و y^* را برای P و D به‌گونه‌ای بدست می‌آوریم که:

$$c_P(x^*) \leqslant \alpha.c_D(y^*)$$

اگر ویژگی بالا برقرار باشد به کمک دوگانی ضعیف داریم:

$$c_P(x^*) \leqslant \alpha.c_D(y^*) \leqslant \alpha.OPT_{LP}(P) \leqslant \alpha.OPT_{IP}(P) \Longrightarrow c_P(x^*) \leqslant \alpha.OPT_{IP}(P)$$

دقت کنید x^* جواب صفر و یکی است و متناظر با جوابی برای مسئله‌ی اصلی پوشش مجموعه‌ای می‌باشد.
پس اگر x^* و y^*ای پیدا کنیم که دارای ویژگی $c_P(x^*) \leqslant \alpha.c_D(y^*)$ باشند الگوریتم ما آلفا تقریب خواهد بود و به مطلوب خود دست یافته‌ایم. حال برای این‌که به این نامساوی دست پیدا کنیم می‌توانیم از قضیه لنگی مکمل[15] استفاده کنیم.
به کمک قضیه لنگی مکمل برای مسئله‌ی بالا داریم:

۱. اگر $y_e > 0$ باشد، آنگاه $\sum_{S\ni e} x_S = 1$ است.

۲. اگر $x_S > 0$ باشد، آنگاه $\sum_{e\in S} y_e = c_S$ است.

قضیه ۲. *اگر یک زوج جواب برای مسئله‌ی اصلی و دوگان داشته باشیم که شرایط لنگی مکمل را برآورده کنند آنگاه دوگانی قوی داریم و مقدار توابع هدف مسئله‌ی اصلی و دوگان برابر است.(البته این قضیه را به صورت کلی در جلسات برنامه‌ریزی خطی اثبات کرده‌ایم)*

اثبات.

$$\begin{aligned} c_P(x^*) &= \sum_{S\in\mathcal{Q}} c_S x_S^* \\ \text{(نتیجه‌ی دوم لنگی مکمل)} \Rightarrow &= \sum_{S\in\mathcal{Q}} x_S^* \sum_{e\in S} y_e^* = \sum_{S\in\mathcal{Q}} \sum_{e\in S} x_S^* y_e^* \\ \text{(جابه‌جایی سیگما)} \Rightarrow &= \sum_{e\in\mathcal{U}} \sum_{S\ni e} y_e^* x_S^* = \sum_{e\in\mathcal{U}} y_e^* \sum_{S\ni e} x_S^* \\ \text{(نتیجه‌ی اول لنگی مکمل)} \Rightarrow &= \sum_{e\in\mathcal{U}} y_e^* = c_D(y^*) \Longrightarrow c_P(x^*) = c_D(y^*) \end{aligned}$$

□

[15] Complementary slackness

حال دقت کنید که ما نمی‌توانیم برای مسئله جواب دقیق ارائه دهیم و نکته‌ی روش اولیه‌ـ دوگان این است که شرایط لنگی مکمل نیز برای زوج x^* و y^* به صورت تقریبی برقرار باشد یعنی شرایط لنگی مکمل تقریبی که به صورت زیر تعریف می‌شود را می‌خواهیم برآورده کنیم.

شرایط لنگی مکمل تقریبی:

۱. اگر $y_e > 0$ باشد، آنگاه $\sum_{S \ni e} x_S \leqslant f$ است.

۲.اگر $x_S > 0$ باشد، آنگاه $\sum_{e \in S} y_e = c_S$ است.

دقت کنید که فقط در قید اول تقریب وارد کردیم به این صورت که در مسئله‌ی P می‌دانیم باید $\sum_{S \ni e} x_S \geqslant 1$ می‌باشد و در شرط اول لنگی مکمل دقیق باید $\sum_{S \ni e} x_S = 1$ باشد حال شرط موجود را با خیلی زیاد نبودن $\sum_{S \ni e} x_S$ تقریب زده‌ایم.

با تعریف بالا، تغییر زیر در اثبات قضیه ۲ به وجود می‌آید و داریم:

$$
\begin{aligned}
c_P(x^*) &= \sum_{S \in \mathcal{Q}} c_S x_S^* \\
&\Rightarrow = \sum_{S \in \mathcal{Q}} x_S^* \sum_{e \in S} y_e^* = \sum_{S \in \mathcal{Q}} \sum_{e \in S} x_S^* y_e^* \quad \text{(نتیجه‌ی دوم لنگی مکمل تقریبی)} \\
&\Rightarrow = \sum_{e \in \mathcal{U}} \sum_{S \ni e} y_e^* x_S^* = \sum_{e \in \mathcal{U}} y_e^* \sum_{S \ni e} x_S^* \quad \text{(جابه‌جایی سیگما)} \\
&\Rightarrow \leqslant f \sum_{e \in \mathcal{U}} y_e^* = f.c_D(y^*) \Longrightarrow c_P(x^*) \leqslant f.c_D(y^*) \quad \text{(نتیجه‌ی اول لنگی مکمل تقریبی)}
\end{aligned}
$$

دقت کنید ما به دنبال x^* و y^*ای بودیم که ویژگی $c_P(x^*) \leqslant \alpha.c_D(y^*)$ را داشته باشند و به کمک لنگی مکمل تقریبی به این خواسته رسیدیم. حال نیاز است الگوریتمی ارائه دهیم که x^* و y^*ای به ما بدهد که در شرایط لنگی مکمل تقریبی صدق کنند.

الگوریتم به صورت زیر است:

۱. $T = \varnothing$ و داریم: $y_e = 0 \quad \forall e$

۲. تا زمانی که عنصر پوشانده‌نشده‌ای وجود دارد ($g \notin \bigcup_{A \in T} A$):

۳. y_g را آن‌قدر زیاد کنید تا برای یک مجموعه‌ی S داشته باشیم $\sum_{e \in S} y_e = c_S$.

۴. S را به T اضافه می‌کنیم.

دقت کنید که در شروع الگوریتم به دلیل صفر بودن y_gها یک جواب شدنی برای دوگان داریم.(چون c_Sها مثبت‌اند و همه‌ی محدودیت‌ها ارضا می‌شوند.) حال در روند الگوریتم کم‌کم مقدار y_gها را طوری زیاد می‌کنیم که همیشه جواب مسئله دوگان شدنی باقی بماند. پس با توجه به روند الگوریتم هیچ‌گاه شرط دوم لنگی مکمل نقض نمی‌شود.

به زبان دیگر ابتدا از جوابی برای مسئله‌ی P شروع می‌کنیم و با توجه به این‌که $T = \varnothing$ می‌باشد درواقع از جواب معادل با همه‌ی x_Sها مساوی صفر شروع می‌کنیم ولی دقت کنید این جواب ناشدنی[۱۶] است و در روند الگوریتم بالا این هدف را داریم که این جواب را به یک جواب شدنی تبدیل کنیم. هم‌چنین در شروع الگوریتم با توجه به صفر بودن y_gها جواب شدنی برای مسئله‌ی دوگان داریم. کم‌کم در روند اجرای الگوریتم جواب مسئله‌ی P شدنی‌تر می‌شود یعنی تعداد کم‌تری محدودیت نقض می‌شود و عناصر بیش‌تری پوشانده می‌شوند و هم‌چنین در هر مرحله مقدار تابع هدف برای دوگان نیز به علت افزایش y_gها بهبود پیدا می‌کند.

در انتهای الگوریتم بالا به جوابی می‌رسیم که در برنامه‌ریزی خطی P شدنی‌ست و جواب مسئله‌ی دوگان نیز همیشه شدنی بود و هم‌چنین

[16] Infeasible

مقدار جواب مسئله‌ی P صفر و یکی می‌باشد پس به جوابی از مسئله‌ی پوشش مجموعه‌ای رسیده‌ایم. شرط دوم لنگی مکمل تقریبی که خودبه‌خود در دل الگوریتم برقرار می‌ماند و فقط باید بررسی کنیم که شرط اول لنگی مکمل تقریبی نیز برقرار می‌ماند. در این‌جا کافی است f را مشابه حالت گرد کردن قطعی همان بیشینه فرکانس یک عنصر قرار دهیم و با توجه به این تعریف شرط $\sum_{S \ni e} x_S \leqslant f$ بدیهی می‌شود.

با توجه به توضیحات بالا **الگوریتم گفته‌شده یک الگوریتم f تقریب برای مسئله‌ی پوشش مجموعه‌ای می‌باشد.**

حالت خاصی از مسئله‌ی پوشش مجموعه‌ای، مسئله‌ی پوشش راسی[۱۷] در گراف می‌باشد. دقت کنید در این مسئله $f = 2$ می‌باشد و عناصر متناظر با یال‌ها و مجموعه‌ها متناظر با راس‌ها می‌باشند که مجموعه متناظر با هر راس مجموعه‌ی یال‌هایی می‌باشد که به آن راس وصل‌اند.

چون در این مسئله مقدار f برابر ۲ است پس روند توضیح داده شده در بالا یک الگوریتم ۲ تقریب را به ما می‌دهد.

روش اولیه_دوگان به طور کلی روش پرکاربردی می‌باشد و در حل دقیق مسائل نیز استفاده می‌شود. یعنی سعی می‌کنیم جوابی پیدا کنیم که شرایط لنگی مکمل را به صورت دقیق ارضا کند و به طور هم‌زمان جوابی برای مسئله‌ی اولیه و دوگان پیدا کند. مثلا الگوریتم کلاسیک پیدا کردن تطابق بیشینه[۱۸] در گراف دوبخشی[۱۹] و حتی الگوریتم فورد فالکرسون[۲۰] برای مسئله‌ی شار بیشینه[۲۱] مسائلی هستند که با روش اولیه_دوگان قابل تعبیر می‌باشند.

[17] Vertex cover
[18] Maximum matching
[19] Bipartite graph
[20] Ford-Fulkerson
[21] Max flow

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

الگوریتم‌های جویباری

نگارنده: محبوبه عنایتی

دز این جلسه قصد داریم به معرفی الگوریتم‌های جویباری [۱] بپردازیم. مدل جویباری مدلی مناسب برای برخورد با جریان زیادی از اطلاعات است؛ به خصوص در قرن حاضر که بیش از هر زمان دیگری با داده‌های حجیمی سر و کار داریم و حجمشان بیش از میزانی است که بتوان تمامی آن‌ها را ذخیره کرد.

۱ مقدّمه

گاهی با میزان زیادی از اطّلاعات روبه‌رو می‌شویم که نمی‌توانیم به تمامی آن‌ها دسترسی تصادفی [۲] داشته باشیم. برای مثال هنگامی که این اطّلاعات روی یک نوار [۳] (نوارها بیشتر در گذشته مورد استفاده قرار می‌گرفتند) ذخیره شده باشند و یا اگر بخواهید آن‌ها را از روی دیسکی بخوانید، خواندن به صورت ترتیبی [۴] سریع‌تر از دسترسی تصادفی خواهد بود و مدل جویباری به کمک خواهد آمد. بنابراین این مدل در قدیم بیشتر مورد استفاده قرار می‌گرفت و پس از مدتی از دور خارج شد؛ چرا که نوارها نیز روز به روز کمتر استفاده می‌شدند. با این وجود با ظهور عصر داده‌های عظیم [۵] و پیدایش اینترنت و دیگر منابع که محتوی اطّلاعات زیادی هستند، مدل جویباری بیش از پیش مورد توجه و مطالعه و بررسی قرار گرفت. به عنوان یک مثال کاربردی و نسبتاً جدید، فرض کنید یک روتر [۶](مسیریاب) دارید که در هر واحد زمانی (مثلاً ثانیه) حجم زیادی از اطّلاعات (مثلاً چند گیگابایت) یا بسته‌ها از آن عبور می‌کنند و فقط قادر است بخش کوچکی از آن را نگه دارد، که در این صورت مدل جویباری به‌کار خواهد آمد. بر خلاف یک نوار که ممکن است حاوی کلّ اطّلاعات باشد (گرچه رَم [۷] مورد استفاده فضای کوچک‌تری داشته باشد و نتواند کلّ آن را یک‌جا نگه دارد) و می‌توان چند بار آن را خواند، جویبارهای بسته‌ها تنها یک بار از روتر عبور می‌کنند و در این حالت فقط یک دفعه می‌توان آن‌ها را مورد پردازش قرار داد. این‌جاست که نقش حافظه پررنگ‌تر می‌شود و سعی بر این است که تا جای ممکن با حافظه‌ی کمْ مسائل بیشتری را حل کنیم.

به صورت کلی، ورودی الگوریتم‌های جویباری، جویباری از n عنصر [۸] است و هدفْ ذخیره‌سازی آن‌ها در حافظه‌ای با اندازه‌ی بسیار کوچک‌تر از n، مثلاً در مرتبه‌ی $\log n$ و یا غالباً $poly\log(n)$، می‌باشد. در اکثر موارد مقدار n و زمان پایان جویبار مشخص نیست. گاهی نیز ممکن است تخمینی از آن موجود باشد. برای مثال یکی از کارهایی که می‌توانیم با چنین حافظه‌ای انجام دهیم، شمردن تعداد عناصر است که به راحتی می‌توان در حافظه‌ی لگاریتمی ذخیره کرد.

در ادامه به بررسی چند مسئله و استفاده‌ی الگوریتم‌های جویباری در حلّ هریک خواهیم پرداخت.

[1] Streaming
[2] random access
[3] tape
[4] sequential
[5] big data era
[6] router
[7] RAM
[8] item

۲ مسئله‌ی شمردن تعداد عناصر

یکی از مسائل بنیادی در مدل جویباری، محاسبه‌ی گشتاور kام[۹] یک جویبار است. مسئله‌ی شمردن تعداد عناصر نیز از این دست مسائل است.

ورودی. جریانِ $s = a_1, a_2, \ldots, a_n$ به طوری که $a_i \in \{1, \ldots, l\}$.

خروجی. محاسبه‌ی $\sum_{j=1}^{l} (f_j)^k$ (تعداد رونوشت‌های عنصر a_j در جویبار ورودی $= f_j$).

به وضوح اگر $k = 1$، آنگاه خروجی برابر تعداد عناصر (n) خواهد بود. در صورتی که $k = 0$، و با این فرض که $0^0 = 0$ آنگاه جواب برابر تعداد عناصر متمایز در جویبار است. برای حالتی که $k = 2$ نیز الگوریتم‌های زیبایی وجود دارد که حاصل را با حافظه‌ی چندلگاریتمی محاسبه می‌کنند. هر یک از گشتاورها اطلاعات خوبی درباره‌ی جویبار مورد نظر می‌دهند، ولی گشتاورهای بالاتر گشتاور دوم را نمی‌توان با حافظه‌ی چندلگاریتمی حساب کرد (با حافظه‌ی کمتر از n امکان‌پذیر است).

شاید در نگاه اول شمارش تعداد عناصر با حافظه‌ی لگاریتمی بسیار ساده و سطحی به نظر برسد؛ کافیست شمارش‌گری[۱۰] را هر بار یک واحد زیاد کنیم. ولی در واقع هدف ما این است که حافظه‌ی مصرفی را کمتر از لگاریتمی کنیم که صورت قطعیِ آن بعید است. ولی همان طور که در ادامه خواهیم دید، با تقریب خوبی به هدف خواهیم رسید.

فرض کنید شمارنده‌ی c را بعد از دیدن هر عنصر به احتمال $\frac{1}{2}$ زیاد کنیم. پس در نهایت خواهیم داشت

$$\mathbb{E}[c] = \frac{n}{2}$$

بنابراین می‌توانیم تخمین n را دو برابر c در نظر بگیریم:

$$\hat{n} = 2c$$

و در این حالت یک بیت صرفه‌جویی خواهیم کرد، چرا که مقدار شمارش‌گر در انتها نصف تعداد عناصر است.

راه دیگر این است که هر بار مقدار شمارنده را به احتمال $\frac{1}{2^k}$ زیاد کنیم و k بیت صرفه‌جویی داشته باشیم. ولی با این حال تعداد بیت‌هایی که صرفه‌جویی می‌کنیم ثابت است و حدّاکثر $n - k$ بیت ذخیره خواهیم داشت. ولی هدف این است که به طور نمایی بیت‌های کمتری مصرف گردد و به جای مرتبه‌ی $\log n$، در مرتبه‌ی $\log \log n$ بیت استفاده شود.

ایده. تعداد بیت‌های شمارش‌گر ایده‌آل را نگه داریم (برای مثال زمانی که شمارش‌گر، عددی بین 2^{i-1} و 2^i باشد، $i - 1$ بیت مصرف شود).

[9] moment
[10] counter

۱.۲ الگوریتم شمارش موریس [11]

MorrisCounter

```
1  x = 0 // x := counter
2  for each item in stream
3      r = random integer ∈ [1, 2^x]
4      if r = 2^x
5          x = x + 1
6  return 2^x − 1
```

طبق این الگوریتم، همان طور که در شبه‌کد آن نیز مشاهده می‌کنید، شمارش‌گر xی داریم که به ازای هر عنصر جویبار (خط ۲)، به احتمال $\frac{1}{2^x}$ یک واحد افزایش می‌یابد (خط ۵). یعنی به این شرط که عدد تصادفی r که از بازه‌ی 1 تا 2^x انتخاب می‌شود (خط ۳)، برابر 2^x گردد (خط ۴). در نهایت تخمین ما از تعداد عناصر جویبار $2^x - 1$ خواهد بود (خط ۶). اکنون نشان می‌دهیم که امید ریاضی خروجی الگوریتم، همان تعداد عناصر است و تخمین‌گری ناارایب[12] برای n است.

قضیه ۱. اگر x_m مقدار شمارش‌گر بعد از دیدن m عنصر باشد، آنگاه $\mathbb{E}[2^{x_m} - 1] = m$.

اثبات. به وسیله‌ی استقرا درستی قضیه را ثابت می‌کنیم.

◊ حالت پایه: واضح است که اگر $m = 0$، آنگاه $\mathbb{E}[2^{x_0} - 1] = \mathbb{E}[2^0 - 1] = 0$.

◊ فرض: $\mathbb{E}[2^{x_m} - 1] = m$.

◊ گام استقرا $(m + 1)$:

$$\begin{aligned}
\mathbb{E}[2^{x_{m+1}}] = \sum_{j=1}^{m} \mathbb{E}[2^{x_{m+1}} | x_m = j] &= \sum_{j=1}^{m} (2^{j+1} \cdot 2^{-j} + 2^j \cdot (1 - 2^{-j})) \cdot \Pr[x_m = j] \\
&= \sum_{j=1}^{m} (2^j + 1) \cdot \Pr[x_m = j] \\
&= \sum_{j=1}^{m} 2^j \cdot \Pr[x_m = j] + \sum_{j=1}^{m} \Pr[x_m = j] \\
&= \mathbb{E}[2^{x_m}] + 1 \\
&= m + 2.
\end{aligned}$$

□

در نتیجه تخمین‌گری ناارایب برای تعداد عناصر داریم و به طور متوسط $\log \log n$ بیت مصرف می‌کنیم. اما بیشتر اوقات در تحلیل چنین الگوریتم‌هایی، این که تنها به طور متوسط خوب عمل کنیم کفایت نمی‌کند و لازم است که نشان دهیم تخمین ما به مقدار واقعی نزدیک است که مثلاً با محاسبه‌ی واریانس می‌توان این را استدلال کرد.

[11] Morris
[12] unbiased

لم ۱. (با توجه به تعریف x_m در قضیه‌ی ۱) $\mathbb{E}[2^{2x_m}] = \frac{3}{2}m^2 + \frac{3}{2}m + 1$.

اثبات. به عنوان تمرین به خواننده واگذار می‌شود. □

قضیه ۲. در تحلیل الگوریتم موریس، برای متغیر تصادفی $Z = 2^{x_m} - 1$ داریم: $\mathrm{Var}[Z] = \mathrm{E}[(2^{x_m} - 1 - m)^2] \leq \frac{m^2}{2}$.

اثبات. به عنوان تمرین به خواننده واگذار می‌شود (با استفاده از لم ۱). □

قضیه ۳. برای $\epsilon > 0$،

$$\Pr[|(2^{x_m} - 1 - m)| > \epsilon m] \leq \frac{1}{2\epsilon^2}$$

.

اثبات. با توجه به قضیه‌ی ۲ و نامساوی چبیشف[۱۳] داریم:

$$\begin{aligned}\Pr[|(2^{x_m} - 1 - m)| > \epsilon m] &\leq \frac{\mathrm{Var}[Z]}{(\epsilon m)^2} \\ &\leq \frac{\frac{m^2}{2}}{(\epsilon m)^2} \\ &= \frac{1}{2\epsilon^2}.\end{aligned}$$

□

حال می‌توان با تکرار و میانگین‌گیری واریانس را کاهش داد. به بیانی دیگر، اگر الگوریتم موریس را k بار اجرا و میانگین را گزارش کنیم، واریانس در $\frac{1}{k}$ ضرب می‌شود. به صورت کلی اگر میانگین k رونوشت مستقل از یک متغیر تصادفی را محاسبه کنیم، واریانس جدید، $\frac{1}{k}$ برابر واریانس اولیه می‌شود. بنابراین به منظور کوچک‌تر کردن احتمال دور شدن تخمین محاسبه‌شده از مقدار واقعی، کافیست با انتخاب k مناسب، k شمارش‌گر مستقل داشته و الگوریتم را k بار مستقلاً اجرا کنیم. برای مثال با قرار دادن $k = \frac{5}{\epsilon^2}$ احتمال مذکور حدّاکثر $\frac{1}{10}$ خواهد بود. مقدار ϵ متناسب با میزان خطایی است که حاضر به قبول آن هستیم.

۳ مسئله‌ی شمردن تعداد عناصر متمایز جویبار

همان طور که پیش‌تر اشاره شد، شمردن تعداد عناصر متمایز یک جویبار در واقع همان محاسبه‌ی گشتاور صفرم آن جویبار است. از جمله مثال‌های کلاسیک این مسئله، شمارش تعداد کلمات متمایز آثار ادبی است[۱۴].

از روش‌های پرکاربرد در حلّ این مسئله، نمونه‌گیری[۱۵] و استفاده از تابع درهم‌سازی[۱۶] است. حال با روش دوم به حلّ آن می‌پردازیم.

فرض کنید $h : U \to [0, 1]$ یک تابع درهم‌سازی تصادفی باشد. منظور از U جهان عناصر ورودی مسئله است که این‌جا فرض می‌کنیم $U = \{1, \ldots, l\}$. در عمل نمی‌توان توزیع یکنواختی[۱۷] روی اعداد حقیقی داشت، پس باید دقتی[۱۸] برای این تابع درهم‌سازی (h) تعریف

[13] Chebyshev's inequality

[۱۴] برای مثال اثبات شده [۱] که با داشتن حافظه‌ای معادل ۵ خط چاپی و تنها با یک بار خواندن کلّ آثار شکسپیر، می‌توان با دقت 95% تعداد کلمات متمایز آن‌ها را محاسبه نمود. این تعداد ۳۵۰۰۰ کلمه تخمین زده شد.

[15] sampling

[16] hash function

[17] uniform distribution

[18] resolution

کنیم. همچنین باید حافظه‌ی نگه‌داری این تابع را بررسی کرد که در تحلیل انتزاعی آن را نادیده می‌گیریم. پس فرض می‌کنیم می‌توانیم چنین تابعی انتخاب کرده و با استفاده از آن به شمارش تعداد عناصر متمایز می‌پردازیم.

برای هر عنصر a_i، $h(a_i)$ را حساب می‌کنیم و فقط کمینه‌ی $h(a_i)$ها را نگه می‌داریم. این کمینه $Y := h(a_i)$; باشد.

لم ۲. اگر $Y := h(a_i)$، آنگاه $\mathbb{E}[\mathbb{Y}] = \frac{1}{N+1}$ که N تعداد عناصر متمایز است.

با فرض درستی این لم می‌توان $\hat{N} = \frac{1}{Y} - 1$ را به عنوان تخمینی از تعداد عناصر متمایز در نظر گرفت.

اثبات. به دو روش می‌توان این لم را اثبات کرد:

۱. نکته. $\int_0^1 \Pr[Y \geq z]dz = \int_0^1 z \cdot \Pr[Y = z]dz$ (برای حالت گسسته نیز این تساوی برقرار است. می‌توانید آن را ثابت کنید!).

$$\mathbb{E}[Y] = \int_0^1 \Pr[Y \geq z]dz$$

اگر بخواهیم مقدار کمینه بزرگتر از z باشد، باید مقدار تابع به ازای تمامی عناصر متمایز بزرگتر از z باشد و چون h تصادفی عمل می‌کند داریم:

$$\begin{aligned} &= \int_0^1 (1-z)^N dz \\ &= \Big[-\frac{(1-z)^{N+1}}{N+1} \Big]_0^1 \\ &= \frac{1}{N+1} \end{aligned}$$

۲. $\mathbb{E}[Y]$ برابر با احتمال این است که $N+1$ عدد در بازه‌ی $[0,1]$ به تصادف انتخاب کنیم و آخرین عددْ کمینه باشد؛ چرا که اگر کمینه‌ی N عدد اول Y باشد، آنگاه احتمال این که عدد بعدی کوچکتر از Y انتخاب شود، برابر Y است (دقت کنید که اعداد از بازه‌ی 0 تا 1 انتخاب می‌شوند)؛ یعنی احتمال این که عدد $N+1$م در بازه‌ی $[0,Y)$ باشد. بنابراین به علت تقارن، احتمال این که آخرین عدد کمینه باشد، $\frac{1}{N+1}$ است؛ چون احتمال کمینه بودن هر عدد با احتمال کمینه بودن هر یک از اعداد دیگر یکسان است.

□

مشابه آن‌چه در تحلیل الگوریتم موریس بیان کردیم، این تخمین به طور متوسط خوب است ولی باید سعی کنیم که واریانس آن را تا جای ممکن کوچک کنیم. پس طبق ایده‌ای که برای تحلیل الگوریتم مسئله‌ی قبل به کار گرفتیم، باید این الگوریتم را نیز k بار مستقلاً اجرا کنیم، یعنی Y نهایی، حاصل از میانگین کمینه‌های متناظر با k تابع درهم‌سازیِ مستقل باشد. به وضوح در این صورت حافظه‌ی مصرفی تابع درهم‌سازی k برابر خواهد شد، که اگر آن را در نظر نگیریم، کافی خواهد بود که برد تابع درهم‌سازی را، به جای $[0,1]$، مجموعه رشته‌های دودویی تعریف کنیم. در این صورت حافظه‌ی نگه‌داری عدد کمینه حدّاکثر $\log n$ است (چون برد h رشته‌های دودویی به طول $\log n$ می‌باشد). در نهایت کل حافظه‌ی مصرفی $k \cdot \log n$ خواهد بود. مشابه قبل، مقدار k بستگی به دقت مدّنظر ما دارد.

۴ مسئله‌ی یافتن عناصر پرتکرار [۱۹]

همان طور که از نام آن برداشت می‌شود، هدف این مسئله یافتن عناصری است که بیش از حدّی تکرار شده‌اند. کاربردهای فراوانی برای آن می‌توان تصور کرد؛ از جمله یافتن مواردی که تعداد دفعاتِ زیادی در گوگل جست‌وجو شده‌اند، در بسته‌های عبوری از یک روتر زوج‌های مبدأ و مقصدی که بیشتر تکرار شدند، توییت‌های پرتکرار و .... بسته به این‌که منظور از «پرتکرار» چه باشد، این مسئله را می‌توان به حالت‌های مختلفی صورت‌بندی[۲۰] کرد که احتمالاً برای هر حالت الگوریتم خاص آن باید طرح شود. در ادامه یکی از صورت‌بندی‌های کلاسیک آن را بررسی می‌کنیم.

فرض کنید جریان ورودی عناصر $a_1, \ldots, a_n$ باشند. می‌خواهیم k عنصر از آن‌ها را خروجی دهیم به گونه‌ای که هر عنصری که حدّاقل $\frac{n}{k+1} + 1$ بار تکرار شده است، در لیست خروجی ظاهر شود. برای مثال اگر $k = 4$، آنگاه خروجی 4 عنصری خواهد بود که اکیداً بیش از 20% عناصر را تشکیل می‌دهند. واضح است که در این صورت حدّاکثر k عنصر این خاصیت را دارند. مثلاً حدّاکثر 4 عنصر بیش از 20% ظاهر شده‌اند. ولی این تعداد می‌تواند کمتر از k باشد. پس امکان دارد لیست خروجی الگوریتم عنصر(های) درستِ منفی *(false positive)* داشته باشد؛ یعنی هر عنصری که خاصیت مطلوب را داراست در لیست ظاهر شده ولی ممکن است عناصر فاقد این خاصیت نیز در لیست خروجی باشند.

این مسئله یک الگوریتم قطعی جالب دارد: الگوریتم Misra-Gries.

۱.۴ الگوریتم Misra-Gries

```
Misra-Gries
1   list = ∅
2   for each item x
3       if x ∈ list
4           increament x's counter m
5       elseif length(list) < k
6           add x to list with counter = 1
7       else
8           throw away x
9           decrement the counter of every item in list
10          delete items which their counters become zero
```

در آغاز الگوریتم، لیست تهی است و حدّاکثر می‌تواند k عنصر در خود جا دهد (خط ۱). برای هر عنصرِ x (خط ۲)، وجود داشتنش را در لیست بررسی می‌کنیم. اگر در لیست موجود باشد (خط ۳)، شمارنده‌ی مخصوصش را یک واحد زیاد می‌کنیم (خط ۴). در غیر این صورت، اگر لیست کمتر از k عنصر داشته باشد (خط ۵)، x را به لیست اضافه می‌کنیم و شمارنده‌ی آن را برابر 1 قرار می‌دهیم (خط ۶). وگرنه، x را نادیده می‌گیریم (خط ۸). و در این حالت از شمارنده‌ی هر k عنصر موجود در لیست، 1 واحد کم می‌کنیم (خط ۹) و بعد عنصر نظیر هر شمارنده‌ای را که برابر صفر شد از لیست حذف می‌کنیم (خط ۱۰). بنابراین اواسط اجرای الگوریتم ممکن است تعداد عناصر لیست کمتر از k باشد.

[۱۹] FrequentItems که به مسئله‌ی HeavyHitters نیز شهرت دارد.

[20] formalize

حافظه‌ی مصرفی الگوریتم به اندازه‌ی طول k شمارنده به همراه عنصر متناظرشان است. اگر فرض کنیم مقدار هر عنصر از مرتبه‌ی چندجمله‌ای بر حسب n باشد، طول آن از مرتبه‌ی لگاریتمی است. بنابراین طول شمارنده‌ها و عناصر ازمرتبه‌ی $\log n$ خواهد بود و میزان حافظه‌ی مصرف‌شده حدّاکثر $k \log n$ است.

اکنون به اثبات درستی الگوریتم و این که تمامی عناصرِ جواب را خروجی می‌دهد، می‌پردازیم.

گزاره. در پایان اجرای الگوریتم Misra-Gries شمارنده‌ی مربوط به عنصر x حدّاقل $f_x - \frac{n}{k+1}$ است (با فرض این که شمارنده‌ی عناصر خارج از لیست برابر صفر باشد).

در نتیجه، مقدار شمارنده‌ی هر عنصری که بیش از $\frac{n}{k+1}$ دفعه در ورودی ظاهر شده باشد، حدّاقل 1 است و باید در لیست خروجی باشد.

اثبات. طبق این الگوریتم، هر بار که عنصری مانند x را می‌بینیم، مقدار شمارنده‌ی آن را یک واحد افزایش می‌دهیم (هنگام نادیده‌گرفتن x در خط ۸ شبه‌کد، فرض کنید شمارنده‌ی x را یک واحد افزایش و سپس بلافاصله یک واحد کاهش می‌دهیم). بنابراین f_x بار شمارنده‌ی x زیاد می‌شود. هر بار که شمارنده‌ی x کاهش یابد، در واقع از هر شمارنده‌ی $k+1$ عنصر یک واحد کم می‌شود (k عنصر داخل لیست به علاوه‌ی عنصری که در همان دور حلقه نادیده گرفته شد، چون فرض کردیم که شمارنده‌اش یک واحد افزایش و بلافاصله یک واحد کاهش می‌یابد). پس حدّاکثر $\frac{n}{k+1}$ بار این اتفاق می‌افتد؛ چرا که دقیقاً تعداد دفعات افزایش مقدار شمارنده‌ها نیز n است.

□

به روش‌های دیگر نیز می‌توان به حلّ مسئله‌ی یافتن عناصر پرتکرار پرداخت؛ از جمله روش نمونه‌گیری تصادفی.

۲.۴ نمونه‌گیری تصادفی [۲۱]

فرض کنید می‌خواهیم عناصری را که با احتمال حدّاقل $p = \Omega(1)$ (یعنی p عددی ثابت است و به n بستگی ندارد)[۲۲] در جریان ورودی ظاهر می‌شوند، بیابیم. تعریف کنید $k = \frac{6}{\epsilon^2 p} \ln n$. k عنصر را به صورت تصادفی نمونه‌گیری می‌کنیم. عنصر خاصی را در نظر بگیرید که به احتمال p در جویبار ورودی ظاهر شده است. در این صورت امید ریاضی μ تعداد دفعات ظاهر شدن یک عنصر خاص در نمونه برابر است با: $\mu = kp = \frac{6}{\epsilon^2} \ln 2$.

با استفاده از کران چرنوف[۲۳] نشان می‌دهیم به احتمال کمی تعداد دفعات ظاهر شدن یک عنصر خاص در نمونه که با احتمال p در ورودی ظاهر شده، کمتر از $(1-\epsilon) \cdot \mu$ می‌باشد:

$$\Pr[X_i < (1-\epsilon) \cdot \mu] \le e^{-\frac{\mu\epsilon^2}{2}}$$
$$= n^{-3}$$

و بنابراین با استفاده از کران اجتماع[۲۴] برای $X := \sum X_i$ که متغیر تصادفی تعداد تکرار یک عنصر در نمونه است، نتیجه خواهیم گرفت که به احتمال حدّاکثر n^{-2} مقدار X کمتر از $(1-\epsilon) \cdot n\mu$ است. به بیان ساده، احتمال کم‌بودن تعداد یک عنصر پرتکرار در نمونه، کمتر

[21] Random Sampling

[۲۲] البته به طور خاص اگر از مرتبه‌ی $\frac{1}{\log n}$ یا کوچکتر باشد موردی ندارد. در غیر این صورت مشکل‌ساز است، چون حافظه‌ی مصرفی بیش از مرتبه‌ی لگاریتمی خواهد بود. ولی در حالت کلی آن را عددی ثابت در نظر می‌گیریم تا به حافظه‌ی مصرفی مطلوب برسیم (مانند k بار تکرار الگوریتم Misra-Gries که ثابت فرض شد).

[23] Chernoff bound

[24] Union bound

از n^{-2} است و عنصری که به احتمال حدّاقل p در جویبار وجود دارد، به احتمال بالا[۲۵] در نمونه‌ی ما ظاهر می‌شود. با استدلالی مشابه، با استفاده از کران چرنوف، می‌توان نشان داد که اگر عنصری با احتمال $(1-\epsilon)^2 \cdot p$ در جویبار ورودی باشد، آنگاه احتمال این که کمتر از $(1-\epsilon) \cdot kp$ بار یا $(1-\epsilon)$ برابر میانگین X_i در نمونه ظاهر شود، زیاد است. به بیانی دیگر، احتمال این که به تعداد حدّاقل $1+\epsilon$ برابر میانگین ظاهر شود، طبق کران چرنوف، کم است. بنابراین با احتمال بالا عناصر پرتکرار در نمونه‌گیری تصادفی ظاهر می‌شوند و احتمال وجود عناصر بسیار کم‌تکرار در آن کم است.

اکنون شاید این به ذهن برسد که الگوریتم نمونه‌گیری تصادفی برای این سؤال نکته‌ی خاصی ندارد و الگوریتم Misra-Gries بسیار بهتر عمل کرده و به طور قطعی جواب می‌دهد. پس چرا از این روش برای حلّ مسئله استفاده کردیم؟ باید گفت که به صورت کلی نمونه‌گیری تصادفی روشی بسیار پرکاربرد است و همان‌طور که در جلسات پیشین دیدیم و همچنین در مدل جویباری برای بسیاری از مسائل به‌کار می‌رود و بر خلاف الگوریتم Misra-Gries تنها به یک مسئله‌ی خاص محدود نیست. ضمناً در صورتی که از ابتدا مشخص نشده باشد که به دنبال چه آمار یا اطّلاعاتی از جویبار هستیم و سؤال خوش‌تعریفی نداشته باشیم، می‌توانیم با داشتن نمونه‌ای از آن به سؤالات احتمالی پاسخ دهیم. ولیکن گاهی نیز خوب کار نمی‌کند و باید در استفاده از آن احتیاط نمود و یا حدّاقل هوشمندانه عمل کرد. برای مثال مسئله‌ی تعداد عناصر یکتا در جویبار را بررسی می‌کنیم.

مثال: (استفاده از نمونه‌گیری در تخمین تعداد عناصر یکتا در جویبار) فرض کنید $\frac{n}{2}$ عناصر ورودی یکتا باشند و $\frac{n}{4}$ عناصر، هر یک، دو بار ظاهر شده باشند. پس جواب مسئله در این حالت خاص $\frac{n}{2}$ است و اگر به طور تصادفی عنصری را انتخاب کنیم، به احتمال $\frac{1}{2}$ یکتاست. مثلاً فرض کنید از 10% ورودی نمونه‌برداری کنیم و تعداد عناصر یکتا در نمونه را به عنوان $\frac{1}{10}$ تعداد عناصر جویبار در نظر بگیریم. یعنی جواب مسئله را 10 برابر تعداد عناصر نمونه تخمین بزنیم. این روش جواب نخواهد داد؛ چرا که انتظار می‌رود به طور متوسط نیمی از اعضای نمونه از عناصر یکتا انتخاب شده باشند. اما به طور متوسط نیمی دیگر متشکل از عناصری است که دو بار در جویبار ظاهر شدند و اکثر آن‌ها به طور یکتا در نمونه حضور دارند. در این صورت بیشتر اعضای نمونه (بیش از 90%) یکتا به نظر خواهند رسید.

اگر طوری نمونه‌گیری را انجام دهیم که به ازای هر عنصری که به عنوان نمونه انتخاب شد، همه‌ی رونوشت‌های دیگر آن نیز در نمونه ظاهر شوند این مشکل بر طرف خواهد شد. برای این منظور، می‌توان از تابع درهم‌سازی بهره جست. مثلاً برای انتخاب نمونه‌ای از 10% عناصر می‌توانیم از تابع $h : U \to \{1, \dots, 10\}$ استفاده کنیم و تمامی عناصری را انتخاب کنیم که مقدار تابع h به ازای آن‌ها برابر 1 است. در این حالت، تنها از 10% عناصر نمونه‌برداری شده و چون مقدار تابع برای همه‌ی رونوشت‌های یک عنصر یکسان است، عناصر نمونه‌ی انتخابی به همراه رونوشت‌های خود ظاهر شده‌اند. ⊠

۱.۲.۴ نمونه‌گیری انباری[۲۶]

در ابتدای بخش ۲.۴ عدد k را با وابستگی نرمی به n تعریف کردیم. در این صورت اگر تنها تخمینی از n داشته باشیم، برای مثال n در بازه‌ی $[\frac{x}{1000}, 1000x]$ به ازای عددی مانند x باشد، آنگاه می‌توانیم با تقریب جمعی 10، $\ln n$ را تخمین زنیم و k را محاسبه کنیم. اکنون فرض کنیم k را ثابت و بدون هر گونه وابستگی‌ای به n تعریف کنیم و هیچ اطلاعی از n در دسترس نیست. در این صورت می‌توان از روش نمونه‌گیری انباری برای حلّ مسئله اسفاده نمود.

ایده‌ی این الگوریتم آن است که در لحظه، به طور استقرایی، k عنصر تصادفی از عناصری را که تاکنون خوانده‌ایم ذخیره کنیم و به هر عنصر جدیدی که می‌رسیم لیست ذخیره شده را به‌روزرسانی کنیم. به بیان دیگر، فرض کنید تاکنون (تا لحظه‌پیش از خواندن عنصر a_{i+1}ام)، k عنصر تصادفی از $a_1, \dots, a_i$ را ذخیره کرده‌ایم و پس از خواندن عنصر a_{i+1} می‌خواهیم این k عنصر، k عنصر تصادفی از $a_1, \dots, a_{i+1}$ باشد. در این صورت عنصر a_{i+1} را با احتمال $\frac{k}{i+1}$ به نمونه‌ی خود اضافه می‌کنیم. اگر اضافه شود، با احتمال یکسانی یکی از k عنصر قبل را از نمونه حذف می‌کنیم تا تعداد اعضای آن همان k باقی بماند. بنابراین همیشه نمونه‌ای k عنصری داریم و احتمال

[25] **w**ith **h**igh **p**robabilty
[26] Reservoir Sampling

انتخاب‌شدن تمامی عناصر یکسان است.

پیشتر با مزیت استفاده از توابع درهم‌سازی آشنا شدیم. در ادامه نیز استفاده‌ی دیگری از آن‌ها را می‌بینیم.

۳.۴ روش Count-Min Sketch[۲]

تاکنون فرض بر این بود که عناصر به جویبار تنها اضافه می‌شوند و فرکانس عناصر (همان f_i برای عنصر iم) تنها می‌تواند افزایش یابد. در این بخش فرض قابلیت حذف‌شدن عناصر ($\text{del}(a_i)$) را نیز به صورت مسئله اضافه می‌کنیم. همچنین فرض می‌کنیم فرکانس یک عنصر هیچ موقع منفی نمی‌شود. مشابه صورت کلی، هدف تخمین تعداد عناصر پرتکرار است و این که در پایانْ نمونه یا طرحی (sketch) از عناصر داشته باشیم تا به کمک آن تخمین خوبی از تعداد عناصر پرتکرار داشته باشیم (برای مابقی عناصر می‌تواند تخمین خوبی نباشد).

پس فرض کنید در طی الگوریتم طرحی از عناصر داریم و در پایان آن باید برای عنصر ورودیِ x، فرکانس آن با خطای جمعی ϵn برگردانده شود. به این معنا که به ازای هر عنصر x باید f_x به گونه‌ای تخمین زده باشیم که حدّاکثر به اندازه‌ی ϵn با فرکانس واقعی آن اختلاف داشته باشد. البته چنین تقریبی تنها برای عناصر پرتکرار معنی دارد، چون اگر فرکانس واقعی x کمتر از ϵn باشد، فرکانس تخمینی آن امکان دارد صفر شود که معنی ندارد. در ادامه به شرح این روش Count-Min Sketch خواهیم پرداخت.

تابع درهم‌سازی $h_i : U \to \{1, \dots, b\}$ را در نظر بگیرید. هنگام دیدن عنصر a_j ($\text{add}(a_j)$)، $c[h_i(a_j)]$ را در جدول درهم‌سازی c یک واحد زیاد می‌کنیم (و به طور مشابه هنگام دیدن دستور حذف a_j یا $\text{del}(a_j)$ یک واحد از آن کم می‌کنیم). در پایان، $c[h_i(x)]$ را به عنوان تخمین $f(x)$ برمی‌گردانیم. در واقع این عدد کران بالایی برای فرکانس x است؛ چرا که با هر بار دیدن این عنصر یک واحد به مقدار $c[h_i(x)]$ اضافه می‌کنیم و همچنین ممکن است تابع درهم‌سازی تصادم داشته باشد، یعنی عنصری مانند y موجود باشد که $h_i(x) = h_i(y)$. بنابراین چیزی که برمی‌گردانیم، تخمین‌گری اریب برای $f(x)$ است. فرض کنید $Z_i = c[h_i(x)]$. در واقع داریم $Z = f_x + \sum_{\substack{y \neq x \\ h(y)=h(x)}} f_y$ پس:

$$\begin{aligned}
\mathbb{E}[Z_i] &= f_x + \sum_{y \neq x} f_y \cdot \Pr[h(y) = h(x)] \\
&= f_x + \frac{1}{b} \sum_{y \neq x} f_y \\
&\leq f_x + \frac{1}{b} \sum_{y \in U} f_y \\
&= f_x + \frac{n}{b}.
\end{aligned}$$

اگر قرار دهیم $b = \frac{2}{\epsilon}$، آنگاه $\mathbb{E}[Z_i] \leq f_x + \frac{\epsilon n}{2}$. همچنین طبق نامساوی مارکوف[۲۷] داریم: $\Pr[Z_i - f_x \geq \epsilon n] \leq \frac{1}{2}$ و بنابراین احتمال این که مقدار متغیر تصادفی Z_i و فرکانس واقعی یک عنصر، به ازای تابع درهم‌سازی h_i ، حدّاقل ϵn باشد، حدّاکثر $\frac{1}{2}$ است. حال به علت این که $\frac{1}{2}$ کران چندان قابل اطمینانی نیست، می‌توانیم k تابع درهم‌سازی در نظر بگیریم و در نهایت مقدار متغیر تصادفی $Z := \min_{1 \leq i \leq k} Z_i$ (هر Z_i در تناظر با h_i است) را به عنوان تخمین خواسته‌شده خروجی دهیم؛ چرا که هر Z_i کران بالایی برای x است پس کمینه‌ی آن‌ها نیز کران بالای x می‌باشد. بنابراین برای متغیر تصادفی z و یا همان تخمین نهایی که از f_x خواهیم داشت، داریم:

$$\Pr[Z - f_x \geq \epsilon n] \leq \frac{1}{2^k}$$

و در این صورت اگر تعریف کنیم که $k := O(\lg n)$ آنگاه:

$$\leq \frac{1}{n^{O(1)}}$$

و بنابراین خطای تخمین‌گر از ϵn کمتر خواهد بود.

[27] Markov's inequality

اگر در همین سیستم فرض کنید که بخواهیم مسئله‌ای مشابه آن‌چه در بخش ۲.۴ را درنظر گرفتیم، یعنی همه‌ی عناصری را که با احتمال p ظاهر شده‌اند برگردانیم، می‌توانیم $\epsilon = \frac{p}{2}$ در نظر گرفته و همه‌ی عناصری که تخمین فرکانسی بیش از np را دارند، به عنوان خروجی گزارش دهیم. در این صورت، به طور قطعی عناصری که حدّاقل np بار در جویبار ظاهر شده‌اند، در خروجی حضور دارند و همین‌طور عناصری که کمتر از $\epsilon n = n \cdot \frac{p}{2}$ دفعه در ورودی بودند، با احتمال بالا در خروجی نخواهند بود.

مراجع

[1] M Durand, P Flajolet. *Loglog counting of large cardinalities*, Annual European Symposium on Algorithms (ESA03), 2003.

[2] G Cormode, S Muthukrishnan. *An improved data stream summary: the count-min sketch and its applications*, Journal of Algorithms 55 (1), 58-75, 2005.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه: فشرده سازی گراف

نگارنده: محمد حسین مروجی

فرض کنید گراف خیلی بزرگی داریم و می‌خواهیم مسئله ای را روی آن حل کنیم مثلا مسئله کوتاهترین مسیر روی گراف ولی چون اندازه گراف بزرگ است پس زمان لازم برای حل این مسئله نیز زیاد می‌شود همچنین حافظه مورد استفاده نیز زیاد می‌شود. در این جلسه سعی می‌کنیم زیرگراف مناسبی از گراف اصلی پیدا کنیم که سایز زیرگراف به طور قابل توجهی کوچکتر از گراف اصلی باشد ولی با حل مسئله مورد نظر روی این زیرگراف، تقریب قابل قبولی از جواب مسئله در گراف اصلی بدست آوریم.

۱ فشرده سازی گراف

ایده‌ای که می‌خواهیم بررسی کنیم این است که از روی گراف ورودی گرافی کوچکتر بسازیم که خواص گراف ورودی را تا حدی داشته باشد و مسئله مورد نظر را روی آن حل کنیم.

ورودی : گراف $G = (V, E)$

خروجی : زیرگراف $H = (V, E')$ از G که سایز E' تا حد امکان کوچک باشد و همچنین $\forall u, v \in V : d_H(u, v) = d_G(u, v)$

اگر کمی فکر کنیم متوجه می‌شویم رابطه بالا معقول نیست زیرا برای مثال اگر گراف کامل را در نظر بگیریم فاصله هر دو راس یک است ولی با حذف حتی یک یال دیگر رابطه بالا برقرار نیست. بنابراین فرض زیر معقول تر است:

$$\forall u, v \in V : d_G(u, v) \leq d_H(u, v) \leq \alpha d_G(u, v) + \beta$$

تعریف : زیرگراف H را یک پوشاننده$-(\alpha, \beta)$ [1] می‌گویند.

اگر $\beta = 0$ در اینصورت H را یک پوشاننده ضربی می نامند.

اگر $\alpha = 1$ در اینصورت H را یک پوشاننده جمعی می نامند.

اگر G گرافی جهت دار باشد ممکن است به هیچ وجه نتوانیم α, β مناسبی پیدا کنیم که تقریب خوبی ارائه دهد. برای مثال گراف جهت دار G را یک گراف کامل دوبخشی در نظر بگیرید که تمام یال ها از قسمت A خارج و به B وارد می شوند. در این صورت با حذف یک یال به وضوح دیگر فاصله آن دو راسی که یال بین آن ها حذف شده، بی نهایت می شود و دیگر در نامساوی مورد نظر صدق نمی کند. پس ایده فشرده سازی گراف را فقط برای گراف های بدون جهت بررسی می‌کنیم.

۲ پوشاننده ضربی

ابتدا نشان می‌دهیم اگر $\alpha = 2$ در اینصورت لزوما زیرگرافی از G وجود ندارد به طوری که $\forall u, v \in V : d_H(u, v) \leq 2d_G(u, v)$

[1] Spanner

برای مثال گراف دوبخشی کامل را در نظر بگیرید. اگر یک یال از آن را حذف کنیم فاصله بین این دو راس در گراف باقی مانده حداقل باید ۳ باشد زیرا اگر ۲ باشد یک دور به طول ۳ باید در G وجود داشته باشد که چون G دوبخشی است پس دور به فرد ندارد که تناقض است پس فاصله بین این دو راس حداقل باید ۳ باشد که با توجه به رابطه بالا، برای اینکه زیر گراف مورد نظر خوب باشد باید فاصله بین هر دو راس کمتر مساوی ۲ باشد، پس باید تمام یال های G را انتخاب کنیم. پس زیرگرافی کوچکتر از G با خاصیت مورد نظر وجود ندارد.

قضیه : فرض کنید $2t-1$ یا $2t$ $\alpha =$ ،در اینصورت یک « $\alpha-$ پوشاننده ضربی » با $O(n^{1+\frac{1}{t}})$ یال وجود دارد.

الگوریتمی برای بدست آوردن زیرگراف مورد نظر:

FindSubgraph(G)

```
1  H = (V, ∅)
2  for each e = uv ∈ E
3      if d_H(u, v) > α
4          add e to H
5  return H
```

اثبات درستی الگوریتم: بوضوح طبق فرآیند الگوریتم اگر بین دو راس در G یال باشد در این صورت فاصله این دو راس در H کمتر مساوی α است. حال دو راس دلخواه x, y در G را در نظر می‌گیریم. کوتاهترین مسیر بین این دو راس در G را در نظر می‌گیریم. بین هر دو راس متوالی در مسیر یال وجود دارد پس بین هر دو راس متوالی در مسیر در H یک مسیر با حداکثر α یال وجود دارد پس در کل یک گشت بین x, y در H وجود دارد. پس داریم که $d_H(x,y) \leq \alpha d_G(x,y)$.پس الگوریتم درست کار می‌کند.

حال قبل از اینکه ثابت کنیم در زیرگراف بدست آمده، تعداد یال ها برابر $O(n^{1+\frac{1}{t}})$ است، لم زیر را ثابت می‌کنیم.

لم: فرض کنید $\bar{d} = \frac{2E}{V}$ که $\bar{d}$ برابر متوسط درجه یک راس در H باشد. آنگاه یک زیرگراف H' از H با حداقل درجه $\frac{\bar{d}}{2}$ وجود دارد.

اثبات: الگوریتم زیر را در نظر بگیرید.

```
1  V_0 = V, E_0 = E, i = 0
2  while ∃v ∈ V : d(v) ≤ E_i / V_i
3      i = i + 1
4      delete v from (V_i, E_i) to obtain (V_{i+1}, E_{i+1})
5  return (V_i, E_i)
```

نکته‌ای که وجود دارد این است:

$$\frac{E_i}{V_i} \geq ... \geq \frac{E_1}{V_1} \geq \frac{E}{V}$$

چون در پایان الگوریتم $\frac{E_i}{V_i}$ مخالف صفر است پس چند راس در V_i باقی می‌مانند و تمام رئوس در V_i به صورت $d(v) \geq \frac{E_i}{V_i} \geq \frac{E_0}{V_0}$ که این همان خواسته مسئله است پس کافی است نامساوی $\frac{E_i}{V_i} \geq ... \geq \frac{E_1}{V_1} \geq \frac{E}{V}$ اثبات کنیم.

$$\frac{E_j}{V_j} = \frac{x}{y} \rightarrow \frac{E_{j+1}}{V_{j+1}} = \frac{x-t}{y-1}, t \leq \frac{x}{y}$$

$$\frac{x-t}{y-1} \geq \frac{x-\frac{x}{y}}{y-1} = \frac{x(1-\frac{1}{y})}{y(1-\frac{1}{y})} = \frac{x}{y} \Rightarrow \frac{E_{j+1}}{V_{j+1}} \geq \frac{E_j}{V_j}$$

پس لم به طور کامل ثابت شد.

تعریف: به طول کوتاهترین دور از گراف، کمر گراف می‌گویند.

لم: کمر گراف H (زیرگراف G که توسط الگوریتم فشرده‌سازی بدست آمده است) بیشتر مساوی $\alpha + 2$ است.

برهان خلف: فرض کنید دوری در گراف با طول کمتر از $\alpha + 2$ وجود داشته باشد، آن را c می‌نامیم. فرض کنید آخرین یالی از c که در الگوریتم به H اضافه شده است $e = uv$ باشد. در الگوریتم ابتدا بررسی می‌کنیم که اگر $d_H(u, v) > \alpha$ باشد e را به H اضافه کنیم. اما چون با اضافه شدن e به H دوری به طول حداکثر $\alpha + 1$ ایجاد می‌شود پس در کل مسیری به طول کمتر از α در H بین دو راس u, v وجود داشته است که این با فرض اضافه شدن e به H در تناقض است. پس فرض خلف باطل و حکم برقرار است.

زیرگراف H' از H را مانند لم، در نظر می‌گیریم. بوضوح چون H' زیرگراف H است پس کمر H' بزرگتر مساوی $\alpha + 2$ است.

یک راس از H' مانند v در نظر می‌گیریم و یک درخت جست‌وجوی اول سطح از v ایجاد می‌کنیم و تا $t = \lceil \frac{\alpha}{2} \rceil$ سطح جلو می‌رویم. در درخت حاصل هر راس در سطح $i \leq t$ نمی‌تواند به دو راس در سطح $i - 1$ وصل باشد چون در غیر اینصورت دوری به طول حداکثر $2t < \alpha + 2$ بوجود می‌آید که تناقض است.

حداقل راس های در سطح t برابر $\left(\frac{\bar{d}}{2} - 1\right)^t$ است چون هر راس در سطح i، یک یال به پدرش دارد و حداقل $\left(\frac{\bar{d}}{2} - 1\right)$ یال به سطح $i + 1$ دارد و چون v دارای حداقل درجه $\frac{\bar{d}}{2}$ است پس در کل در سطح t حداقل $\left(\frac{\bar{d}}{2} - 1\right)^t$ حداقل $\left(\frac{\bar{d}}{2} - 1\right)^{t-1}.\frac{\bar{d}}{2} \geq \left(\frac{\bar{d}}{2} - 1\right)^t$ راس وجود دارد. حال داریم که:

$$\left(\frac{\bar{d}}{2} - 1\right)^t \leq n \rightarrow \left(\frac{m}{n} - 1\right)^t \leq n \rightarrow m \leq n + n^{1+\frac{1}{t}} \rightarrow O(m) = O(n^{1+\frac{1}{t}})$$

پس اثبات قضیه کامل شد.

حدس کمر: برای هر $\alpha = 2t, 2t - 1$ و n به اندازه کافی بزرگ، گراف هایی با کمر $\alpha + 2$ و $\Omega(n^{1+\frac{1}{t}})$ یال وجود دارد.

این حدس برای $\alpha = 1, 2, 3, 5$ اثبات شده‌است.

این حدس نشان می‌دهد که گرافی که برای قضیه بالا ارائه دادیم بهینه است. چون گرافی در نظر بگیرید که در حدس بالا صدق کند. در اینصورت اگر یک یال از این گراف را در نظر بگیریم مانند $e = uv$، این یال را به هیچ وجه نمی‌توانیم از گراف حذف کنیم زیرا کوتاهترین مسیری که در گراف حاصل از حذف e وجود دارد، طولش حداقل $\alpha + 1$ زیرا اگر مسیری با طول کمتر وجود داشته باشد آن مسیر با یال e تشکیل یک دور با طول کمتر از $\alpha + 2$ را می‌دهد که تناقض است.

۳ پوشاننده جمعی

قضیه: یک ۲ ـ پوشاننده جمعی با $O(n\sqrt{n}\ln n)$ یال وجود دارد.

الگوریتم ساخت زیرگراف مورد نظر: ۱ ـ همه یال‌های متصل به راس‌های با درجه حداکثر $\sqrt{n}$ را به H اضافه می‌کنیم.

۲ ـ به طور تصادفی، $2\sqrt{n}\ln n$ را انتخاب می‌کنیم و از هر کدام یک درخت جست‌وجوی اول سطح به ریشه آن راس را به H اضافه می‌کنیم.

توجه کنید تعداد یال‌ها در گراف حاصل از الگوریتم، برابر $O(n\sqrt{n}\ln n)$ است.

دو راس دلخواه u, v را در نظر می‌گیریم. کوتاه‌ترین مسیر بین این دو راس در G را نظر می‌گیریم. اگر تمام رئوس بین u, v دارای

حداکثر درجه $\sqrt{n}$ باشند. پس مسیر بین این دو راس عینا در H وجود دارد و مسئله حل است. پس فرض کنید تعدادی از راس‌های این مسیر دارای درجه بیش تر از $\sqrt{n}$ باشند. در اینصورت روی این مسیر در H حرکت می‌کنیم و فرض کنید اولین جایی که نمی‌توان روی مسیر حرکت کرد بین x, y باشد. پس درجه x, y بیش‌تر از $\sqrt{n}$ است. اگر یکی از همسایه های x مانند z وجود داشته باشد که جزو راس‌های انتخاب شده در مرحله دوم الگوریتم باشد، در اینصورت در درخت جست‌وجوی اول سطح با ریشه z، کوتاهترین مسیر از z به v کوتاهتر از مسیری است که از z به x می‌رویم و از x از روی مسیر بین u, v به سمت v می‌رویم. پس چون درخت جست‌وجوی اول سطح با ریشه z به H اضافه شده‌است می‌توانیم از u به x برویم و سپس از x به z و سپس روی درخت جست‌وجوی اول سطح از z به v برویم که طول این مسیر حداکثر ۲ واحد بیش تر از کوتاهترین مسیر بین u, v در G است.

پس کافی است ثابت کنیم با احتمال بالا z وجود دارد.

$$\Pr\left[\text{هیچ همسایه از } x \text{ انتخاب نشود}\right] \leq \left(1 - \frac{\sqrt{n}}{n}\right)^{2\sqrt{n}\ln n} \leq \frac{1}{n^2}$$

پس به احتمال $\left(1 - \frac{1}{n^2}\right)$ برای هر راس حداقل یکی از همسایه‌هایش در مرحله دوم الگوریتم انتخاب شده‌است. پس اثبات کامل شد.

در پایان یک سری قضیه را بدون اثبات می‌آوریم.

قضیه: یک ۲ـ پوشاننده جمعی با $O(n^{\frac{3}{2}})$ یال وجود دارد.

قضیه: یک ۴ـ پوشاننده جمعی با $\tilde{O}(n^{\frac{7}{5}})$ یال وجود دارد.

قضیه: یک ۶ـ پوشاننده جمعی با $\tilde{O}(n^{\frac{4}{3}})$ وجود دارد.

قضیه: برای هر ثابت $c > 0$ ، یک cـ پوشاننده جمعی در بدترین حالت با $\Omega(n^{\frac{4}{3}})$ وجود دارد.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه اضافه: هندسه محاسباتی

نگارنده: علی جباری

در این جلسه قرار است بطور مختصر راجب به هندسه محاسباتی صحبت شود و دو مساله بطور خاص در این حیطه بررسی گردد. هندسه محاسباتی در حوزه‌های فراوانی کاربرد دارد که از میان آن‌ها می‌توان به گرافیک کامپیوتری، روباتیک، طراحی به کمک کامپیوتر و سیستم اطلاعات هندسی [1] اشاره کرد.

پیش از این مواجهاتی با هندسه محاسباتی داشتیم. در جلسه‌ی دوم مساله‌ی نزدیک‌ترین زوج نقطه را دیدیم. بعدتر هم الگوریتم افزایشی تصادفی [2] را دیدیم که به هندسه‌ی محاسباتی مربوط می‌شود. در این جلسه هم می‌خواهیم دو مساله را بررسی کنیم که عبارتند از:

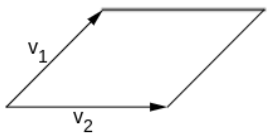
ـ پیدا کردن پوش محدب [3] تعدادی از نقاط

ـ پیدا کردن تمام تقاطع‌های تعدادی پاره‌خط

پیش از صحبت درباره‌ی این دو سوال لازم است تعدادی از اعمال مقدماتی هندسی [4] معرفی شوند.

۱ اعمال مقدماتی

۱.۱ مساحت متوازی‌الاضلاع حاصل از دو بردار غیر هم‌راستا



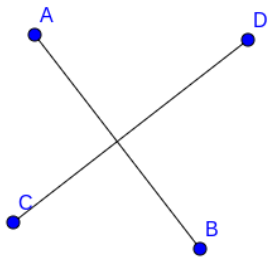
اگر دو بردار مورد نظر، $v_۱(x_۱, y_۱)$ و $v_۲(x_۲, y_۲)$ باشند، مساحت متوازی‌الاضلاع گفته شده، برابر با اندازه‌ی ضرب خارجی دو بردار می‌شود:

$$\text{Area} = |v_۱ \times v_۲| = |\begin{vmatrix} x_۱ & x_۲ \\ y_۱ & y_۲ \end{vmatrix}| = |x_۱y_۲ - x_۲y_۱|$$

علامت $x_۱y_۲ - x_۲y_۱$ نیز با توجه به برون‌سو یا درون‌سو بودن بردار $v_۱ \times v_۲$ نسبت به صفحه‌ی متوازی‌الاضلاع، به ترتیب مثبت و منفی خواهد شد. معادلا، اگر نقطه‌ی انتهایی بردار $v_۲$ سمت چپ بردار $v_۱$ باشد، این علامت مثبت می‌شود و در غیر این‌صورت، منفی است. بدین صورت می‌توان فهمید یک نقطه کدام طرف خطی جهت‌دار است. کافیست یک بردار هم جهت روی خط مذکور انتخاب کرده و به روش بالا چک کنید نقطه کدام طرف این بردار است.

[1] Geometric Information System
[2] Randomized Incremental Algorithm
[3] Convex hull
[4] Geometric Primitives

۲.۱ چک کردن متقاطع بودن دو پاره‌خط



یک راه این است که با استفاده از معادله‌ی خطوط منطبق بر دو پاره‌خط، نقطه‌ی تقاطع را پیدا کنید و چک کنید که روی دو پاره‌خط قرار دارد یا خیر.

راه دیگر که از مساله‌ی قبلی کمک می‌گیرد، به این شکل است که چک کنید C و D در دو طرف خط گذرنده از A و B باشند و متقابلاً، A و B در دو طرف خط گذرنده از C و D باشند. اگر این دو شرط برقرار باشند، می‌توان ثابت کرد که دو پاره‌خط متقاطعند.

۲ پیدا کردن پوش محدب

۱.۲ تعریف پوش محدب

پوش محدب n نقطه، مجموعه نقاطی از صفحه است که می‌توانند به شکل ترکیب محدبی از آن n نقطه نوشته شوند. تجسم شهودی این مفهوم در دو بعد به این شکل است که n نقطه را میخ‌هایی در صفحه در نظر بگیرید و کش بزرگی را دور همه‌ی آنها قرار داده و رها کنید. پس از مدتی کش به شکل یک چند ضلعی محدب در می‌آید که همان پوش محدب است. در ابعاد بالاتر، پوش محدب به شکل یک چندوجهی محدب است. تمام نقاط محصور شده در پوش محدب را می‌توان به صورت ترکیب محدبی از نقاط راسی آن نوشت.

۲.۲ صورت دقیق مساله‌ی پوش محدب

تعدادی نقطه داریم. پوش محدب این نقاط را پیدا کرده و رئوس آن را به ترتیبی مشخص(در اینجا ترتیب پادساعتگرد مدنظر است) خروجی دهید.

۳.۲ شباهت مساله‌ی پوش محدب و مساله‌ی مرتب‌سازی

مساله‌ی مرتب‌سازی را می‌توان با استفاده از پوش محدب به شکل زیر حل کرد:

فرض کنید میخواهیم n عدد x_1، x_2، ...، x_n را به صورت صعودی مرتب کنیم. نقاط (x_1, x_1^2)، (x_2, x_2^2)، ...، (x_n, x_n^2) را به مساله‌ی پوش محدب می‌دهیم. از جایی که این نقاط روی سهمی $y = x^2$ هستند، تمام این نقاط در پوش محدب ظاهر خواهند شد. در نتیجه، خروجی الگوریتم پیدا کننده‌ی پوش محدب، ترتیبی از همان n نقطه‌ی ورودی خواهد بود که مولفه‌ی x آن‌ها به ترتیب، $O = (o_1, o_2, ..., o_n)$ است. از جایی که ترتیب نقاط خروجی، پادساعتگرد بود، دنباله‌ی O نیز به شکل یک پسوند و سپس یک پیشوند از ترتیب هدف مساله‌ی مرتب‌سازی است. برای مثال، اگر ورودی مساله، اعداد ۱, ۲, ۳ باشند، دنباله‌ی O می‌تواند (۱, ۲, ۳)، (۲, ۳, ۱) و یا (۳, ۱, ۲) باشد. از روی دنباله‌ی O هم میتوان دنباله‌ی مرتب شده را در زمان $O(n)$ با پیدا کردن عضو کمینه بدست آورد.

بدین ترتیب، می‌توان کران پایین $\Omega(n \log n)$ را برای مساله‌ی پوش محدب ثابت کرد زیرا در غیر این صورت، مساله‌ی مرتب‌سازی را خواهیم توانست در زمانی کمتر از $\Theta(n \log n)$ حل کنیم، در حالی که در کلاس داده‌ساختارها ثابت شد این کار ممکن نیست.

حال سعی می‌کنیم مساله‌ی پوش محدب را در زمان $O(n \log n)$ حل کنیم. برای سادگی فرض کنید هیچ ۳ نقطه‌ای هم‌خط نباشند.

الگوریتم $O(n^۳)$:

پاره‌خط بین هر زوج نقطه را در نظر بگیرید. اگر تمام نقاط دیگر تنها در یک طرف خط گذرنده از این دو نقطه باشند، پاره خط مذکور، یکی از اضلاع پوش محدب خواهد بود. هر کدام از نقاط اگر در بین رئوس پوش محدب باشد، دقیقا با دو نقطه‌ی دیگر پاره خطی می‌سازد که شرط بالا را دارد. برای هر نقطه، این نقاط همسایه را نگهداری میکنیم(در صورت وجود).

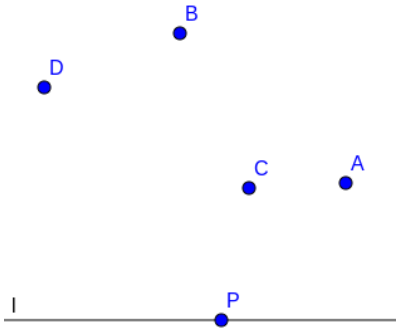
در نهایت، برای تمام نقاط راسی، همسایه‌های آن‌ها روی پوش محدب را داریم که به کمک آن می‌توانیم رئوس پوش محدب را در زمان $O(n)$ به ترتیب بدست بیاوریم.

در این الگوریتم، $O(n^۲)$ پاره‌خط داریم که برای چک کردن اینکه هر پاره خط روی پوش محدب باشد، $O(n)$ عملیات نیاز داریم. در نهایت هم یک پس‌پردازش با زمان $O(n)$ داریم. پس پیچیدگی زمانی این روش، $O(n^۳)$ است.

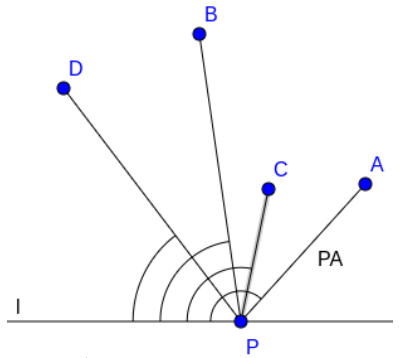
با مقداری تغییر میتوان به الگوریتمی سریعتر رسید:

الگوریتم $O(n^۲)$:

نقطه‌ای که عرض آن کمینه است، حتما باید راس پوش محدب باشد، وگرنه نمی‌توان آن را به شکل ترکیب محدب نقاط راسی پوش محدب نوشت. نام این نقطه را P بگذارید و فرض کنید روی خطی افقی به نام l باشد.



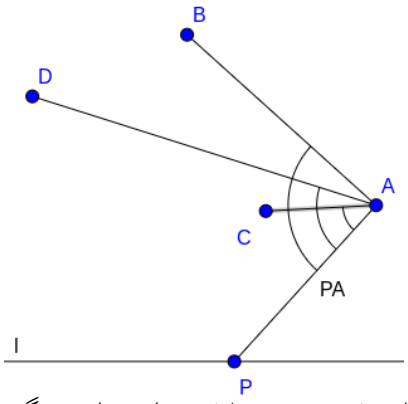
زاویه‌ی همه‌ی نقاط دیگر با P و جهت منفی l را حساب کنید و فرض کنید بیشترین زاویه را نقطه‌ی A بسازد. باز هم به طرز مشابه ثابت می‌شود A باید روی پوش محدب باشد.



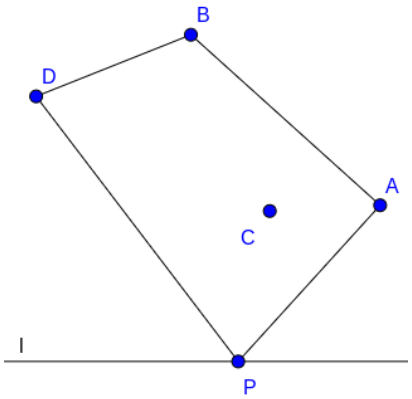
برای مقایسه‌ی دو زاویه، مثل $\angle lPC$ و $\angle lPB$ کافیست به صورتی که بالاتر گفته شد، چک کنیم B سمت راست $\overrightarrow{PC}$ است یا سمت چپ آن. در شکل بالا، از جایی که همه‌ی نقاط سمت چپ $\overrightarrow{PA}$

هستند، می‌توان فهمید زاویه‌ی $\angle lPC$ از بقیه‌ی زوایا بزرگتر است.

در گام بعدی زاویه‌ی همه‌ی نقاط با نقطه‌ی A و پاره‌خط PA را حساب میکنیم. نقطه‌ای که بیشترین زاویه را می‌سازد را نقطه‌ی B می‌نامیم. این نقطه هم باید روی پوش محدب باشد.



همین کار را تا زمانی انجام می‌دهیم که کل پوش محدب پیدا شود، یا به عبارتی دیگر، دوباره به نقطه‌ی P برسیم.



اگر پوش محدب k راس داشته باشد، زمان اجرای این الگوریتم $O(kn)$ خواهد شد، زیرا در هر گام $O(n)$ عملیات انجام می‌شود و در کل هم k مرحله داریم. اگر k زیاد باشد، زمان الگوریتم حداکثر $O(n^2)$ خواهد شد.

برای حل این سوال در زمان $O(n \log n)$ الگوریتم های زیادی وجود دارند. برای مثال میتوان از تکنیک تقسیم و حل [5] استفاده کرد. الگوریتمی که در ادامه گفته می‌شود نیز این مساله را در زمان $O(n \log n)$ حل میکند.

۴.۲ الگوریتم اسکن گراهام [6]

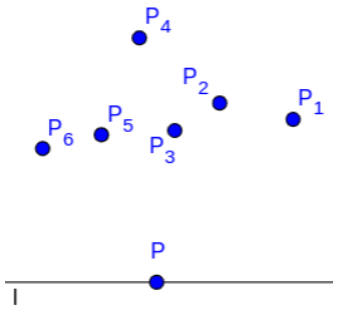
ابتدا مشابه الگوریتم $O(n^2)$ نقطه‌ای که کمترین عرض را دارد انتخاب می‌کنیم و نامش را P می‌گذاریم. سپس خطی که P روی آن است را l می‌نامیم. همه‌ی نقاط را با توجه به زاویه‌ای که با P و جهت منفی l می‌سازند، در زمان $O(n \log n)$ بصورت نزولی مرتب می‌کنیم. برای مقایسه‌ی زوایا هم به همان شکلی که پیش‌تر گفته شد عمل می‌کنیم. سپس به رئوس مرتب شده به ترتیب اسم‌های P_1 تا P_{n-1} را نسبت می‌دهیم.

یک پشته به نام Q نگه می‌داریم که در هر گام، نقاطی که تا آن زمان کاندیدای راس پوش محدب شدن هستند را در آن می‌ریزیم. در ابتدا رئوس P و P_1 را در Q می‌ریزیم زیرا به سادگی می‌توان ثابت کرد این دو راس حتما روی محیط پوش محدب قرار دارند. سپس در مرحله‌ی i، $(1 < i < n)$ چک می‌کنیم P_i سمت چپ بردار حاصل از دو عضو آخر Q باشد(برداری که از عضو یکی‌مانده‌به‌آخر Q شروع می‌شود و به عضو آخر Q می‌رود). تا زمانی که P_i سمت راست برداری باشد که به شکل گفته‌شده ساخته می‌شود، آخرین عضو Q را بیرون می‌اندازیم. در نهایت، زمانی که بردار حاصل از دو عضو آخر Q و P_i شرط گفته شده را داشتند، P_i را به انتهای Q اضافه می‌کنیم.

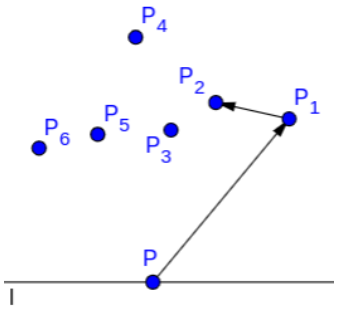
هر عضو ۱ بار به Q اضافه می‌شود و حداکثر ۱ بار از آن حذف می‌شود پس این بخش از الگوریتم هزینه‌ی $O(n)$ دارد. پس پیچیدگی زمانی کل الگوریتم، $O(n \log n)$ است.

[5] Divide and Conquer
[6] Graham Scan

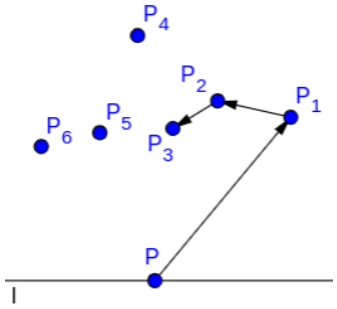
برای مثال فرض کنید الگوریتم را روی نقاط زیر اجرا کنیم:



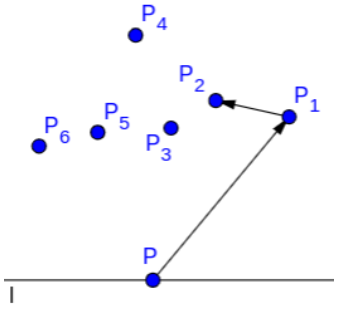
در ابتدا P و P_1 داخل Q هستند و در گام شماره‌ی ۲، P_2 به Q اضافه می‌شود زیرا P_2 سمت چپ بردار $\overrightarrow{PP_1}$ قرار دارد.



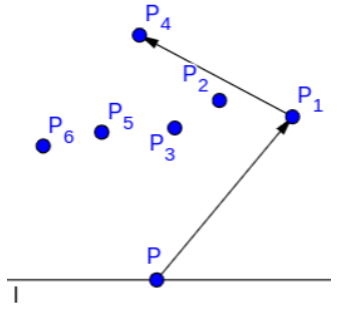
در مرحله‌ی بعد به طرز مشابه، P_3 وارد Q می‌شود.



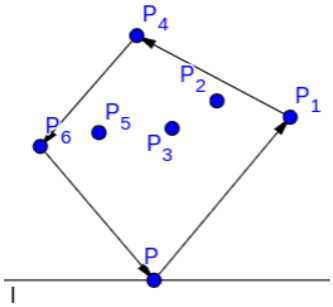
سپس از جایی که P_4 سمت راست $\overrightarrow{P_2P_3}$ قرار دارد، P_3 را از Q حذف میکنیم.



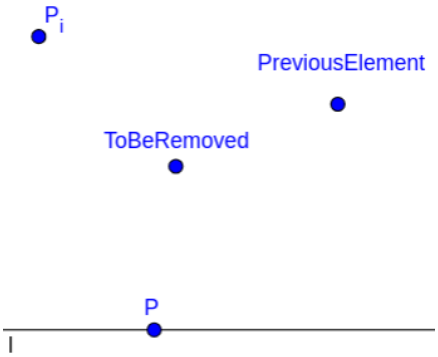
P_2 نیز به طرز مشابه از Q حذف می‌شود. سپس P_4 به Q اضافه می‌شود زیرا سمت چپ بردار $\overrightarrow{PP_1}$ قرار دارد.



در گام بعدی نیز P_5 وارد Q می‌شود. سپس در مرحله‌ی ششم از آن حذف می‌گردد. در نهایت هم در مرحله‌ی آخر P_6 وارد Q می‌شود و الگوریتم Q را به عنوان جواب خروجی می‌دهد.



در این الگوریتم هرگاه در مرحله‌ی i نقطه‌ای از Q حذف شود، یعنی واقعا نمی‌توانسته روی محیط پوش محدب حضور داشته باشد، زیرا آن نقطه را میتوان به شکل ترکیب محدب نقطه‌ی قبلی‌اش در Q، P و P_i نوشت.



همچنین هر برداری در Q نسبت به بردار قبلیش به سمت داخل پوش محدب چرخیده، پس اگر هریک از اعضای نهایی Q در پوش محدب نباشد، آن عضو را دیگر نمی‌توان به صورت ترکیب محدب تعداد دیگری از اعضای Q نوشت. طبق این استدلال، الگوریتم اسکن گراهام، تنها نقاط پوش محدب را خروجی خواهد داد. اثبات دقیق درستی الگوریتم به دانشجویان سپرده می‌شود.

مثال: n نقطه داده شده‌است و می‌خواهیم دورترین زوج نقطه را بیابیم.

برای این مثال الگوریتم بدیهی $O(n)$ وجود دارد ولی می‌خواهیم با کمک پوش محدب، آن را در زمان $O(n \log n)$ حل کنیم.

الگوریتم ما به این شکل است که پوش محدب را حساب می‌کنیم. سپس برای هر ضلع، دورترین نقطه روی پوش محدب از آن پاره‌خط را می‌یابیم و آن را P می‌نامیم. به ازای هر کدام از اضلاع، P و دو راس اطراف آن ضلع، از کاندیداهای جواب هستند. در نهایت دورترین زوج نقطه‌ی کاندیدا به عنوان جواب نهایی انتخاب می‌شوند.

برای حل این مثال، پس از یافتن پوش محدب در زمان $O(n \log n)$ یک ضلع را می‌گیریم و دورترین نقطه از آن را در زمان $O(n)$ بدست می‌آوریم. سپس زوج‌های حاصل از دو راس آن ضلع و دورترین نقطه از آن ضلع(اسمش را P می‌گذاریم) را به لیست کاندیداها اضافه می‌کنیم. سپس سراغ ضلع بعدی در ترتیب پادساعتگرد می‌رویم. ادعا میکنیم دورترین نقطه نسبت به این ضلع، یا همان نقطه‌ی P است، یا از آن در ترتیب پادساعتگرد جلوتر است. پس برای پیدا کردن دورترین نقطه‌ی جدید، کافیست تا جایی، با شروع از P به شکل پادساعتگرد جلو برویم که جلورفتن ما باعث دورتر شدن نقطه‌ی مورد نظر از پاره‌خط شود.

این بخش از الگوریتم در هر گام بطور متوسط [۷] $O(1)$ عملیات طول می‌کشد پس در کل زمان $O(n)$ می‌خواهد. بدین ترتیب، زمان نهایی الگوریتم $O(n \log n)$ است. درستی این الگوریتم را هم بطور دقیق ثابت نمی‌کنیم.

$\boxtimes$

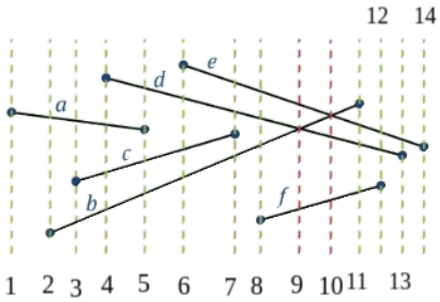
[7] amortized

۳ پیدا کردن تقاطع‌های تعدادی پاره‌خط

n پاره خط در صفحه داریم و می‌خواهیم تقاطع‌های آنها را چاپ کنیم. برای سادگی فرض کنید هیچ دو نقطه‌ای (دو سر پاره‌خط‌ها و نقاط تقاطع) مولفه‌ی x یکسان ندارند.

الگوریتم بدیهی این سوال $O(n^۲)$ است که در آن متقاطع بودن تمام جفت پاره‌خط را چک می‌کنیم.

در اینجا می‌خواهیم الگوریتمی با زمان اجرای $O((n+k)\log n)$ برای این مساله ارائه دهیم که در آن، k تعداد کل تقاطع‌های پاره‌خط‌هاست. این الگوریتم اگر تعداد تقاطع‌ها کم باشد، بهتر از الگوریتم بدیهی عمل می‌کند. برای این‌کار از تکنیکی کاربردی به نام خط جاروب [۸] استفاده می‌کنیم. نمونه‌ی زیر را در نظر بگیرید.



فرض می‌کنیم خط بزرگی به اسم خط جاروب در سمت چپ همه‌ی پاره‌خط‌ها داریم که آن را بصورت پیوسته و به آرامی به سمت راست حرکت می‌دهیم و در هر لحظه نگاه می‌کنیم که چه چیزهایی را قطع می‌کند.

هرگاه خط جاروب به خطوط نقطه‌چین می‌رسد، رویدادی اتفاق افتاده‌است. برای مثال در موقعیت ۱(خط نقطه‌چین ۱)، پاره‌خط a به چیزهایی که خط جاروب را قطع می‌کنند اضافه می‌شود و در موقعیت ۲ پاره‌خط b به این مجموعه افزوده می‌شود.

به عبارت دیگر، در هر لحظه، همه‌ی پاره‌خط‌هایی که خط جاروب را قطع می‌کنند به ترتیب عرض نقطه‌ی تقاطع آنها با خط جاروب را در داده‌ساختاری به اسم SL نگه می‌داریم. بدین ترتیب، بین موقعیت های ۱ و ۲، $SL=(a)$، بین موقعیت ۲ و ۳ $SL=(a,b)$، و بین موقعیت ۳ و ۴ $SL=(a,c,b)$ است.

این تغییرات هم تنها در خطوط نقطه‌چین اتفاق می‌افتند پس کافیست به جای پیوسته حرکات دادن خط جاروب، تنها تغییرات آن را در خطوط نقطه‌چین، که پیشامدها اتفاق می‌افتند، چک کنیم. این پیشامدها را باید در داده‌ساختاری نگهداری کرد که زمان اتفاقات مختلف را در خود نگه می‌دارد و در هر لحظه که بخواهیم، نزدیک ترین اتفاق پیش رو را به ما می‌گوید. داده ساختار هرم برای این کار مناسبست. نام آن را EQ (Event Queue) می‌گذاریم که در ابتدا، نقاط سر و ته تمام پاره‌خط‌ها درون آن هستند و در هر لحظه، میتوانیم نقطه‌ای که کمترین مولفه‌ی x را دارد را از آن خارج کنیم.

دقت کنید لحظه‌ای قبل از آن‌که دو پاره‌خط با هم تقاطع پیدا کنند، همیشه با هم در SL مجاورند.پس کافیست در هر لحظه، همسایه ها را با هم چک کنیم و ببینیم متقاطع هستند یا خیر. اگر متقاطع بودند، نقطه‌ی تقاطع آن‌ها را هم به EQ اضافه می‌کنیم.

پس از هر پیشامد، حداکثر ۲ همسایگی جدید بین پاره‌خط‌ها در SL ایجاد می‌شود که نیازست وجود تقاطع بین آن پاره‌خط‌های همسایه را چک کنیم:

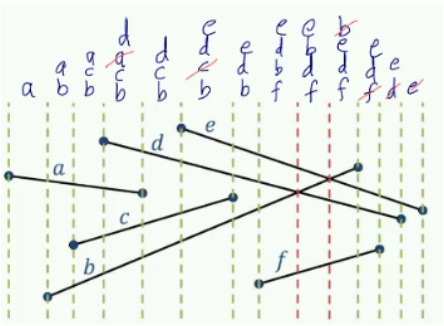
ـ در پیشامدهایی که پاره‌خط جدیدی شروع به قطع کردن خط جاروب می‌کند، حداکثر دو همسایگی جدید بین پاره‌خط‌ها ایجاد می‌شود که باید متقاطع بودن آن‌ها را چک کرد و اگر متقاطع بودند، آن‌ها را به EQ اضافه کرد.

ـ درپیشامدهایی که یک پاره‌خط از SL حذف می‌شود، حداکثر یک همسایگی جدید ایجاد می‌شود.

[8] Sweep Line

ـ پس از پیشامدهایی که دو پاره‌خط یکدیگر را در یک نقطه قطع می‌کنند، جای آن دو پاره‌خط(که قبل از تقاطع با هم همسایه بودند) با هم عوض می‌شود پس حداکثر دو همسایگی جدید بوجود می‌آید.

در تصویر زیر، مقدار SL در زمان‌های مختلف دیده می‌شود:



اگر SL داده‌ساختاری از نوع درخت جستجوی دودویی متعادل [۹] باشد، در هر نقطه‌ی پیشامد، می‌توان در زمان $O(\log n)$ همسایگی‌ها را بدست آورد و پاره‌خط‌های جدید را با ترتیبی که گفته شد از SL حذف و یا به آن اضافه کرد. برای هر پاره خط i در SL زوجی به شکل (m_i, b_i) را به عنوان کلید نگه می‌داریم که همان شیب و عرض‌ازمبدأ خط گذرنده از پاره‌خط iام هستند. سپس هنگام انجام عملیات مختلف در SL، برای مقایسه‌ی دو پاره‌خط i و j در طول x، مقادیر $m_ix + b_i$ و $m_jx + b_j$ را با هم مقایسه می‌کنیم. مقادیر گفته شده، در اصل همان مولفه‌ی y تقاطع پاره‌خط‌ها با خط جاروب در طول x می‌باشند.

شبه کد این الگوریتم را در ادامه مشاهده می‌کنید:

FindSegmentIntersections(S)

```
1  create an empty priority queue EQ
2  for each segment S, insert its start and end events into EQ
3  create an empty balanced tree SL
4  while EQ is not empty
5      E = deletemin(EQ)
6      if E is start of segment s
7          insert s into SL
8          CheckForIntersection(EQ, s, successor of s)
9          CheckForIntersection(EQ, s, predecessor of s)
10     elseif E is the end of segment s
11         CheckForIntersection(EQ, pred of s, succ of s)
12         delete s from SL
13     elseif E is cross event for segment s1 and s2
14         println "s1 and s2 intersect"
15         remove s1 and s2 from SL
16         re-insert s1 and s2 into SL in the opposite order
17         CheckForIntersection(EQ, s1, new neighbor of s1)
18         CheckForIntersection(EQ, s2, new neighbor of s2)
```

[9] Balanced Binary Search Tree

بررسی پیچیدگی زمانی:

در زمان رخداد هر پیشامد، الگوریتم برای استفاده از SL و برای فهمیدن پیشامد بعدی به کمک EQ، هزینه‌ی زمانی $O(\log n)$ می‌دهد. در کل هم $2n$ پیشامد بخاطر حذف و اضافه شدن پاره‌خط‌ها و k پیشامد بخاطر تقاطع‌ها داریم. در نتیجه، زمان اجرای نهایی این الگوریتم $O((n+k)\log n)$ است.

از تکنیک خط جاروب می‌توان در ابعاد بزرگ‌تر هم استفاده کرد، ولی در آن صورت، بجای خط جاروب، با صفحه‌ی جاروب مواجه می‌شوید. در این مثال، حرکت خط جاروب مستقیم و در راستای محور طول بود، ولی در بعضی از مواقع، می‌توان آن را به شکل‌های دیگری حرکت داد. برای مثال خط جاروب می‌تواند حول یک نقطه‌ی ثابت دوران کند.

مراجع

[۱] هندسه محاسباتی، درس آنالیز الگوریتم، دکتر مرتضی علیمی

بخش سوم

جلسات حل تمرین

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس حل تمرین: مجید گروسی

[بهار ۹۹]

جلسه حل تمرین ۱: روش تقسیم و حل و جستجوی همه‌ی حالت‌ها نگارنده: جواد فرخ‌نژاد

در کلاس درس با نمادهای O و Ω برای محاسبه‌ی زمان اجرای الگوریتم‌ها و ایده‌ی تقسیم و حل برای حل مسائل آشنا شدیم. اکنون به حل چند سوال می‌پردازیم.

۱. فرض کنید $۳n$ عدد طبیعی مانند $a_۱, a_۲, \ldots, a_n, b_۱, b_۲, \ldots, b_n, c_۱, c_۲, \ldots, c_n$ داریم. همچنین فرض کنید $p_۱p_۲\ldots p_n$ یک جایگشت از اعداد ۱ تا n باشد. در هر مرحله می‌توانیم یکی از عملیات زیر را انجام دهیم:

A : به ازای هر $۱ \le i \le n$ مقدار a_i را با مقدار a_{p_i} جایگزین کنیم.

B : به ازای هر $۱ \le i \le n$ مقدار a_i را با مقدار $a_i \oplus c_i$ [۱] جایگزین کنیم.

الگوریتمی طراحی کنید که در زمان

الف) $O(۲^k n)$

ب) $O(nF_k)$

تعیین کند آیا بعد از k بار انجام هر کدام از عملیات فوق با هر ترتیب دلخواهی آیا می‌توان از دنباله‌ی $a_۱, a_۲, \ldots, a_n$ به دنباله‌ی $b_۱, b_۲, \ldots, b_n$ رسید یا خیر؟ (منظور از F_k عدد kـام فیبوناچی[۲] است).

حل:

الف) با استفاده از یک الگوریتم بروت‌فورس همه حالات را چک می‌کنیم. برای انجام k عملیات هرکدام دو حالت دارند، یا A انجام می‌شود یا B. بنابراین در کل می‌توان تمام ۲^k دنباله‌ای که از $a_۱, a_۲, \ldots, a_n$ ساخته می‌شوند را شناسایی کرد.
زمان اجرای هر عملیات $O(n)$ است چون باید n مقدار را عوض کنیم و در کل ۲^k دنباله از عملیات‌ها داریم پس زمان اجرای کل الگوریتم برابر خواهد بود با $O(۲^k n)$.

ب) مشابه قسمت قبل عمل می‌کنیم و به نوعی تمام حالات را چک می‌کنیم فقط کافی است به این خاصیت از عمل $\oplus$ دقت کنیم که برای هر $u, v \in \mathbb{N}$ داریم: $(u \oplus v) \oplus v = u$ بنابراین می‌بینیم که اگر در مرحله‌ای B را انجام داده باشیم اینکه در مرحله بعدی دوباره B را انجام دهیم کار بیهوده ایست!! ☺
بنابراین در کل می‌توان حالاتی که برای انجام دادن عملیات روی دنباله داریم را اینطوری درنظر گرفت که در هر مرحله یا A انجام می‌شود یا اگر B انجام شد، بلافاصله پس از آن A انجام شود.
فرض کنید زمان اجرای این الگوریتم T_k باشد. در این‌صورت خواهیم داشت:

$$T_k = T_{k-۱} + T_{k-۲} + O(n)$$

زیرا یا ابتدا A را انجام می‌دهیم که در این‌صورت باید $k-۱$ عملیات دیگر انجام دهیم که زمان اجرای آن $T_{k-۱}$ است، یا ابتدا B را انجام می‌دهیم و سپس A را و در این‌صورت باید از این به بعد $k-۲$ عملیات دیگر انجام دهیم که زمان اجرای آن $T_{k-۲}$ است.
$O(n)$ هم برای این است که سه بار یکی از A یا B را انجام دادیم.با حل رابطه‌ی بالا داریم : $T_k = O(nF_k)$.

---

[۱] xor

[۲] Fibonacci

۲. عدد طبیعی x داده شده‌است. الگوریتمی بیابید که پرمقسوم‌علیه‌ترین عدد کوچترمساوی x را بیابد در زمان

الف) $O(x\sqrt{x})$

ب) $O(x \log x)$

ج) $O(x \log \log x)$

د) $O(x)$

حل:

الف) دوباره با استفاده از یک الگوریتم بروت‌فورس تعداد مقسوم‌علیه‌های تمام اعداد کمترمساوی x را می‌شماریم. پیدا کردن تعداد مقسوم‌علیه‌های عدد n در زمان $O(\sqrt{n})$ قابل انجام است، چرا که برای یافتن تجزیه‌ی n کافی است اعداد اول کمترمساوی $\sqrt{n}$ را چک کنیم که با فرض $n \leq x$ زمان اجرای کل این الگوریتم برابر است با:

$$O(\sqrt{۱} + \sqrt{۲} + \cdots + \sqrt{x}) = O(x\sqrt{x})$$

ب) این‌دفعه رویکردمان را تغییر می‌دهیم و به جای اینکه برای هر $۱ \leq n \leq x$ تعداد مقسوم‌علیه‌های n را بیابیم برای هر $۱ \leq i \leq x$ تمام اعدادی را می‌یابیم که i مقسوم‌علیه‌ای از آنهاست.
برای هر $۱ \leq k \leq x$ ابتدا d_k را مساوی صفر قرار داده سپس برای هر $۱ \leq i \leq x$ و هر $۱ \leq t \leq \frac{x}{i}$ مقدار $d_{i\times t}$ را یک واحد اضافه می‌کنیم. نهایتا d_k تعداد مقسوم‌علیه‌های k خواهد بود. زمان اجرای الگوریتم برابر است با:

$$O\big(\frac{x}{۱} + \frac{x}{۲} + \cdots + \frac{x}{x}\big) + O(x) = O\big(x \times (\frac{۱}{۱} + \frac{۱}{۲} + \cdots + \frac{۱}{x})\big) + O(x) = O(x \log x) + O(x) = O(x \log x)$$

ج) در اینجا بیشتر از مفاهیم نظریه‌اعداد استفاده می‌کنیم. این‌دفعه به جای i های قسمت قبل فقط توان‌های اعداد اول را درنظر می‌گیریم و به نوعی هر دفعه به ازای عدد اول p توان آن را در تجزیه‌ی تمام اعداد کوچکترمساوی x می‌یابیم.
اگر این کار را انجام دهیم که ابتدا به هر عدد کمترمساوی x، یک عدد، که نهایتا برابر با توان عامل p در تجزیه آنها می‌شود، نسبت دهیم، سپس اول تمام مضارب p را در نظر بگیریم و به عدد نسبت داده شده به هر کدام یک واحد اضافه کنیم، بعد مضارب $p^۲$ را در نظر بگیریم و به عدد نسبت داده شده به هرکدام یک واحد اضافه کنیم، سپس مضارب $p^۳$ و ... تا جایی که $x < p^k$. نهایتا عدد نسبت داده شده به هر عدد همان توان عامل p در تجزیه آنهاست چرا که اگر مثلا برای عدد n بزرگترین توان p که n را عاد می‌کند برابر با p^t باشد، عدد نسبت داده شده به n در هر کدام از مراحل مربوط به $p^۱$ و $p^۲$ و ... و p^t یک واحد اضافه می‌شود و نهایتا برابر با t می‌شود که همان عامل p در تجزیه n است.
حال برای هر عدد اول این کار را انجام می‌دهیم و در این‌صورت توان عامل اول p در تجزیه‌ی هر عدد کوچکترمساوی x پیدا می‌شود.
زمان اجرای این الگوریتم برابر است با:

$$O\Big(\sum_{p\text{اول}} (\frac{x}{p} + \frac{x}{p^۲} + \ldots)\Big) = O\Big(\sum_{p\text{اول}} \frac{x}{p-۱}\Big) = O\Big(x \times \sum_{p\text{اول}} \frac{۱}{p-۱}\Big) = O\Big(x \times \sum_{p\text{اول}} \frac{۱}{p}\Big) = O(x \log \log x)$$

د) می‌توان فرض‌کرد عددی که به دنبالش هستیم $A = {p_۱}^{a_۱}{p_۲}^{a_۲} \ldots {p_r}^{a_r}$ است به‌طوری‌که p_i ها اولند و $p_۱ < p_۲ < \cdots < p_r$ و $a_۱ \geq a_۲ \geq \cdots \geq a_r$. چرا که اگر مثلا $a_i < a_{i+۱}$ آنگاه می‌توان عدد ${p_۱}^{a_۱} \ldots {p_{i-۱}}^{a_{i-۱}} {p_i}^{a_{i+۱}} {p_{i+۱}}^{a_i} {p_{i+۲}}^{a_{i+۲}} \ldots {p_r}^{a_r}$ را در نظر گرفت که تعداد مقسوم‌علیه‌هایش با تعداد مقسوم‌علیه‌های A برابر است.

حال در بین اعدادی که این خاصیت را دارند عددی با بیشترین تعداد مقسوم‌علیه را می‌یابیم یعنی الگوریتمی ارائه می‌دهیم که در بین تمام اعداد کمترمساوی x به شکل ${p_۱}^{a_۱}{p_۲}^{a_۲} \ldots {p_r}^{a_r}$ که $p_۱ < p_۲ < \cdots < p_r$ و $a_۱ \geq a_۲ \geq \ldots \geq a_r$ پرمقسوم‌علیه‌ترین را بیابد. شبه‌کد زیر برای این الگوریتم است که زمان اجرای آن $O(x)$ است اما در کلاس اثبات نشد.

```
FUNC(curr_A, min_a, i, curr_div)
1  if min_a == 0
2      ans = max{ans, curr_div}
3      return
4  for a_i = 0 to min_a
5      if curr_A × p_i^{a_i} ≤ x
6          FUNC(curr_A × p_i^{a_i}, a_i, i + 1, curr_div × (a_i + 1))
```

$curr\_A$ همان عددی است که کاندیدای پرمقسوم‌علیه‌ترین عدد است. min_a کوچکترین توان برای عوامل اول $curr\_A$ است که تا به حال ظاهر شده‌است. i اندیس عدد اول است یعنی منظور از p_i، iـامین عدد اول است. $curr\_div$ هم تعداد مقسوم‌علیه‌های عدد $curr\_A$ است.

در ابتدای کار تابع را با مقادیر $(۰, ۱, \infty, ۱)$ صدا می‌زنیم و در انتهای کار مقدار ans همان بیشترین تعداد مقسوم‌علیه را می‌دهد.

دقت کنید که متغیر ans یک متغیر $global$ است که در ابتدای کار صفر بوده است.

همچنین با نگه‌داشتن یک پوینتر به عدد $curr\_A$ هنگام بزرگتر شدن ans، نهایتا این پوینتر به پرمقسوم‌علیه‌ترین عدد اشاره می‌کند.

پس در کل با این روش می‌توان در زمان $O(x)$ پرمقسوم‌علیه‌ترین عدد کمترمساوی x را یافت.

۳. عدد طبیعی k و یک آرایه از اعداد مانند $A[]$ داده شده‌است. می‌خواهیم برای هر k مقدار متوالی از این آرایه مینیمم این k مقدار را بیابیم. الگوریتمی طراحی کنید که این کار را در زمان

الف) $O(nk)$

ب) $O(n)$

انجام دهد.

حل :

الف) با استفاده از یک الگوریتم بروت‌فورس تمام kـتایی‌های مختلف را چک می‌کنیم یعنی برای هر $۱ \le i \le n - k + ۱$ بین تمام مقادیر $A[i], A[i + ۱], \ldots, A[i + k - ۱]$ مینیمم را در زمان $O(k)$ می‌یابیم. زمان اجرای کل الگوریتم برابر است با $O(nk)$.

ب) ابتدا یک جواب دیگر با استفاده از روش تقسیم و حل برای قسمت قبل ارائه می‌کنیم. آرایه را به ۲ بخش به طول‌های $\frac{n}{۲}$ تقسیم می‌کنیم و به طور بازگشتی مسئله را برای هر بخش حل می‌کنیم. حال کافی است مینیمم بازه‌های kـتایی را بیابیم که هم شامل اعدادی از نصفه‌ی چپ و هم شامل اعدادی از نصفه‌ی راست‌اند.

یک روش این است که تمام kـتایی‌های ممکن را در نظر بگیریم و برای هر کدام مینیمم را با $O(k)$ بیابیم. تعداد این k تایی‌ها k تاست و در کل هزینه‌ی این قسمت برابر است با $O(k^۲)$. پس با این روش رابطه‌ی بازگشتی که برای زمان الگوریتم بدست می‌آید برابر است با:

$$T(n) = ۲T(\frac{n}{۲}) + O(k^۲)$$

که با حل آن به زمان اجرای $O(nk^۲)$ می‌رسیم که حتی از زمان قسمت قبل هم بدتر است!!

هدف این است که $O(k^۲)$ را به $O(k)$ کاهش دهیم. برای این‌منظور بعد از اینکه آرایه دو بخش شد فرض کنید اندیس‌های ۱ تا i در سمت چپ آرایه و اندیس‌های $i + ۱$ تا n در سمت راست آرایه باشند.

به ازای هر $i - k + ۱ \le t \le i + ۱$ عدد b_t را مینیمم اعداد بین $A[i + ۱]$ تا $A[t]$ تعریف می‌کنیم. همچنین برای هر $i + k \le t \le i$ عدد c_t را مینیمم اعداد بین $A[t]$ تا $A[i]$ تعریف می‌کنیم.

محاسبه‌ی c_t ها و b_t ها در زمان $O(k)$ قابل انجام است. زیرا با شروع از $t = i$ تا $t = i - k + ۱$ هر دفعه مینیمم دو عدد $c_{t+۱}$

و $A[t]$ را به جای c_t قرار می‌دهیم و به‌طور مشابه با شروع از $t = i + ۱$ تا $t = i + k$ هر دفعه مینیمم دو عدد $b_{t-۱}$ و $A[t]$ را به‌جای b_t قرار می‌دهیم.

حال برای محاسبه‌ی مینیمم اعداد بین $A[r]$ تا $A[r + k - ۱]$ به‌طوری‌که $r \leq i < r + k - ۱$ کافی است مینیمم دو عدد c_r و $b_{r+k-۱}$ را محاسبه کنیم. پس در کل با این محاسبات مینیمم هر k عدد متوالی که هم شامل عددی از سمت چپ و هم شامل عددی از سمت راست است در زمان $O(k)$ قابل محاسبه است و رابطه‌ی بازگشتی به این شکل تغییر می‌یابد:

$$T(n) = ۲T(\frac{n}{۲}) + O(k)$$

این‌دفعه با حل رابطه‌ی بازگشتی به زمان $O(kn)$ می‌رسیم که همان زمان بخش قبل است اما برای هر $i < k$ داریم $T(i) = ۰$ چرا که وقتی طول آرایه کمتر از k باشد اصلا k مقدار متوالی نداریم که بخواهیم آنها را چک کنیم.

بنابراین اگر با شرایط جدید رابطه‌ی بازگشتی را حل کنیم به زمانی که می‌خواهیم یعنی $T(n) = O(n)$ می‌رسیم.

درواقع اگر درخت بازگشتی برای محاسبه $T(n)$ را رسم کنیم کل ارتفاع $\log n$ است اما از ارتفاع $\log n - \log k$ به پایین هزینه رئوس صفر است به عبارت دیگر $\log k$ سطر آخر از درخت همگی هزینه صفر دارند و به طور کلی زمان اجرای الگوریتم برابر خواهد بود با:

$$O(k \times ۲^{\log n - \log k}) = O(k \times \frac{n}{k}) = O(n)$$

به عنوان تمرین می‌توانید سعی کنید با استفاده از صف دوطرفه مسئله آخر را حل کنید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس حل تمرین: مرتضی علیمی

[بهار ۹۹]

جلسه ۲: مؤلفه‌های قویاً همبند و رأس‌های برشی

نگارنده: مهدی مستانی

در جلسه چهارم درس با جستجوی اول عمق(DFS) [1] در گراف‌ها آشنا شدیم. در این جلسه حل تمرین قصد داریم که با جستجوی اول‌عمق و کاربردهای آن بیشتر آشنا شویم.

۱ مؤلفه‌های قویاً همبند

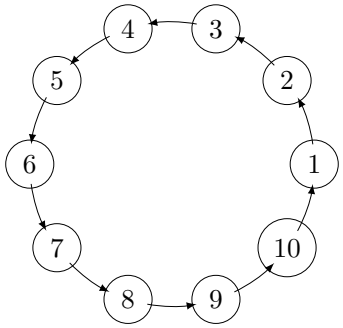
تعریف ۱. *در گراف جهت‌دار $D = (V, A)$ ، مجموعه‌ای از رئوس مانند w ($w \subseteq V$) را یک مؤلفه‌ی قویاً همبند [2] گوییم هرگاه w یک زیرمجموعه‌ی ماکسیمال [3] از رئوس باشد به‌طوری‌که برای هر $s, t \in w$ مسیری از s به t و همینطور مسیری از t به s در گراف موجود باشد .*

قضیه ۱. *مؤلفه‌های قویاً همبند ، راس‌های گراف جهت‌دار را افراز می‌کنند(به عبارتی هیچ دو مؤلفه‌ی‌قویاً همبندی یک رأس مشترک ندارند).*

اثبات. فرض کنید که دو مؤلفه‌ی قویاً همبند مانند c_1 و c_2 در رأسی مانند s مشترک باشند.از آن‌جایی که از هر رأس مؤلفه‌ی c_1 به رأس s و از s به هر رأس مؤلفه‌ی c_1 مسیری وجود دارد و برای مؤلفه‌ی c_2 شرایط مشابهی را داریم،آنگاه بین هر رأس از c_1 و هر رأس از c_2 مسیری وجود دارد و این با ماکسیمال بودن c_1 و c_2 در تناقض است.پس هیچ دو مؤلفه‌ی‌قویاً همبندی با یکدیگر اشتراک ندارند. □

مثال: مؤلفه‌های قویاً همبند برای گراف جهت‌دار بدون دور(DAG) [4] تک رأس‌ها می‌باشند؛زیرا که اگر بین دو رأس مانند s و t مسیری از s به t و مسیری از t به s داشته‌باشیم،آنگاه گراف دارای دور است و با بدون دور بودن گراف در تناقض است. ⊠

مثال: اگر گراف دوری جهت دار باشد، آنگاه تنها یک مؤلفه قویاً همبند دارد.(شامل همه رئوس گراف)



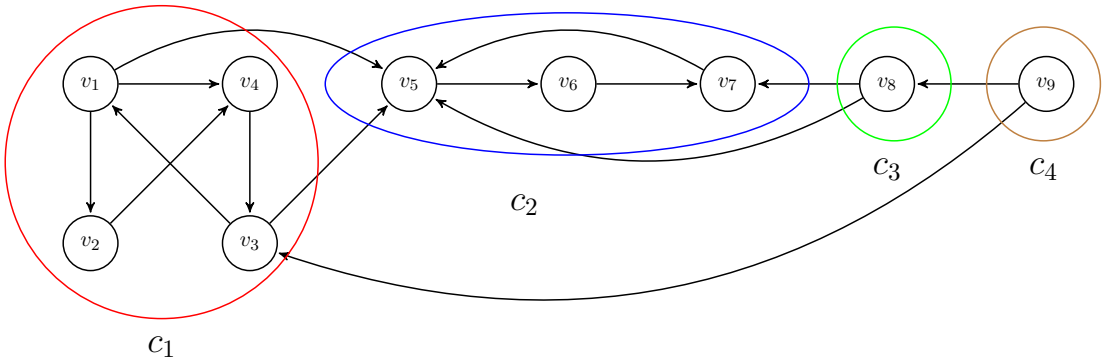
[1] Deep First Search
[2] strong component
[3] maximal subset
[4] Directed Acyclic Graph

⊠

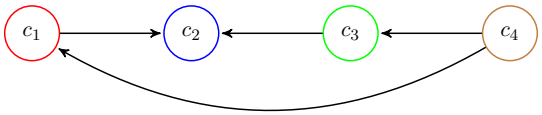
۲ الگوریتمی برای یافتن مؤلفه‌های قویاً همبند

برای گراف جهت دار $D = (V, A)$ گراف H را بدین صورت می‌سازیم که رئوس آن مؤلفه‌های قویاً همبند گراف D هستند و یال‌های آن بدین صورت است که بین دو رأس از H (که همان مؤلفه‌های قویاً همبند D هستند) مانند c_1 و c_2 ، یال جهت‌داری از c_1 به c_2 رسم می‌کنیم اگر راسی داخل مؤلفه c_1 مانند u و راسی مانند v در مؤلفه c_2 موجود باشد که $uv \in A$ باشد(یال جهت‌داری از u به v در گراف اولیه D داشته باشیم).

حال چون برای دو رأس دلخواه مانند $u, v \in V(H)$ نمی‌توانیم همزمان هم مسیری از u به v و مسیری از v به u داشته باشیم(با ماکسیمال بودن مؤلفه‌های قویاً همبند متناظر این دو راس در تناقض است)، پس نتیجه می‌گیریم که گراف جهت‌دار H فاقد دور است و در نتیجه H یک DAG است.



شکل ۱: گراف جهت دار D که در آن مؤلفه‌های قویاً همبند مشخص شده اند



شکل ۲: گراف جهت دار H متناظر با گراف ۱

بر روی گراف D الگوریتم DFS را اجرا می‌کنیم و برای هر $v \in V(D)$ مشابه الگوریتم DFS جلسه چهارم درس،مقادیر $d(v), f(v)$ را برای v مشخص می‌کنیم. مشابهاً برای هر $c \in V(H)$ تعریف کنیم:

$$d(c) = \min\{d(v) | v \in c\}$$

$$f(c) = \min\{f(v) | v \in c\}$$

از لحاظ مفهومی ،برای هر $c \in V(H)$ داریم که $d(c)$ زمانی که اولین بار به کل مجموعه رئوس این مؤلفه‌ی همبندی وارد شده‌ایم و مشابهاً $f(c)$ زمانی است که در نهایت از این مجموعه رئوس خارج شده‌ایم.

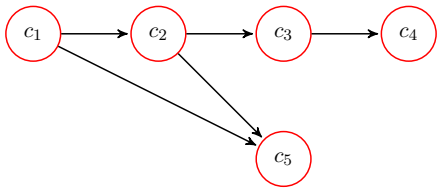
حال اگر برای رئوس $c_i, c_j \in V(H)$ داشته باشیم که $c_ic_j \in A(H)$ آنگاه می‌توانیم نتیجه بگیریم که $f(c_i) > f(c_j)$ ؛ زیرا که در الگوریتم DFS زمانی از یک رأس خارج می‌شویم که تمام یال‌هایی که از آن رأس خارج شده‌اند را پیمایش کرده باشیم و چون یال از c_i به c_j موجود است و گراف H نیز بدون دور است(به عبارتی DAG است)، پس c_i دیر تر از c_j تمام خواهد شد .

$$f(c_i) > f(c_j)$$

شکل ۳: زمان پایان الگوریتم DFS بر روی رئوس گراف H

حال از رابطه ۳ استفاده می‌کنیم تا مؤلفه‌های قویاً همبند را با یک بار دیگر اجرای الگوریتم DFS به دست بیاوریم.چون به دست آوردیم که H(همان گراف مؤلفه‌های قویاً همبند) یک DAG است، اگر بتوانیم که مرتب‌سازی توپولوژیک[5]این DAG را به دست آوریم، آنگاه می‌توانیم که مؤلفه‌ها را به دست آوریم.اگر بدانیم که کدام رأس در H در اجرای الگوریتم DFS دیرتر تمام می‌شود کدام است،آنگاه می‌توانیم از آن رأس الگوریتم DFS را شروع کنیم و مؤلفه قویاً همبند آن را به دست آوریم و سپس به سراغ رأس بعدی در H برویم و تا در نهایت همه مؤلفه‌های قویاً همبند را به دست بیاوریم.

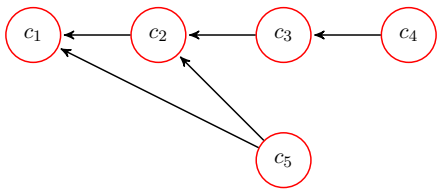
DAG زیر را در نظر بگیرید:



شکل ۴: گراف G

رابطه ۳ بیان می‌کند که برای مثال روابط $f(c_3) > f(c_4), f(c_2) > f(c_3), f(c_2) > f(c_5)$ برقرار است و در بین تمامی مؤلفه‌ها، $f(c_1)$ از سایر مؤلفه‌ها بزرگتر است؛پس در DFS اجرا شده بر روی گراف اصلی(D) ،رأسی که دیرتر از همه اجرای الگوریتم بر روی آن تمام می‌شود، باید در مؤلفه c_1 باشد.

حال ایده‌ای که باعث می‌شود تا بتوانیم این مسئله را حل کنیم، این است که از روی گراف اصلی(D) ، گراف معکوس[6]گراف اصلی را بسازیم(آن را D^T و گراف مؤلفه‌های قویاً همبند متناظر با D^T را H^T بنامید) و بر روی D^T الگوریتم DFS را اجرا کنیم.منظور از گراف معکوس متناظر با گراف D که آن را D^T می‌نامیم، این است که رئوس این گراف، همان رئوس D هستند ($V(H) = V(D)$) و جهت یال ها D را در گراف D^T برعکس کنیم؛یعنی $uv \in A(D)$ اگر و تنها اگر $vu \in A(D^T)$



شکل ۵: گراف معکوس ۴ (G^T)

[5]Topological Sorting
[6]reverse

به سادگی می‌توان بررسی کرد که اگر معکوس یک گراف جهت دار را در نظر بگیریم ، مؤلفه‌های قویاً همبند آن با گراف اولیه یکسان است؛زیرا اگر بین دو رأس u, v در گراف اصلی مسیری در هر دو جهت(هم از v به u و هم از u به v)وجود داشته است،در گراف معکوس نیز همچنان مسیر در هر دو جهت موجود است.(مسیر v به u به مسیر از u به v تبدیل شده است و برعکس)

پس الگوریتم یافتن مؤلفه‌های قویاً همبند گراف D بدین صورت است که ابتدا یک بار الگوریتم DFS را بر روی گراف D اجرا می‌کنیم.سپس گراف D^T را می‌سازیم و بر روی آن الگوریتم DFS را با ترتیب نزولی $f(v)$ به دست آمده از DFS بر روی D اجرا می‌کنیم و سپس رئوس هر درخت DFS از گراف D^T را به عنوان یک مؤلفه قویاً همبند از D خروجی می‌دهیم.

دلیل درستی این الگوریتم این است که مثلاً در گراف ۳ ، $f(c_1)$ از سایر مؤلفه‌ها بزرگتر است.هنگامی که DFS به ترتیب نزولی f بر روی D^T اجرا کنیم،اولین رأسی که الگوریتم بر روی آن اجرا می‌شود(آن را v بنامید و داریم که $f(v)$ بیشینه است)،در c_1 قرار دارد.با اجرای DFS بر روی v ،به وضوح به تمامی رئوس موجود در مؤلفه c_1 می‌رسیم(چون در یک مؤلفه قویاً همبند هستند) .به‌علاوه به هیچ رأس خارج از این مؤلفه نیز نمی‌رسیم؛زیرا اگر اینگونه نباشد و به راسی مانند u از مؤلفه‌ی دیگری مانند c_2 برسیم،آنگاه باید از مؤلفه‌ی c_1 به c_2 در D^T یال باشد و این یعنی از c_2 به c_1 در D یال موجود است و با بیشینه بودن $f(c_1)$ در تناقض است.(با توجه به رابطه ۳) . بنابراین الگوریتم ارائه شده به درستی کار می‌کند .

۳ مسئله ۲ـ صدق پذیری

یکی از کاربردهای الگوریتم DFS ، حل مسئله ۲ـ صدق پذیری [۷](یا به صورت خلاصه $2-SAT$) است.پیش از بیان مسئله ابتدا باید چند مفهوم را بیان می‌کنیم.

لیترال [۸]:یک متغیر یا نقیض آن را **لیترال** می گویند .

عبارت: منظور از عبارت در این مسئله، یا(OR) تعدادی لیترال است.

حال مسئله $2-SAT$ را شرح می‌دهیم(صورت پیشرفته تر این مسئله که $3-SAT$ نام دارد، در جلسات آتی بررسی خواهد شد) :

ورودی:یک فرمول که AND تعدادی عبارت است که هر عبارت شامل ۲ لیترال است.

خروجی: آیا مقداردهی‌ای برای متغیرها موجود است که فرمول داده شده ، مقدار $true$ شود ؟ در این صورت به فرمول داده شده، **ارضاپذیر** گویند.

مثال: فرمول زیر یک نمونه از ورودی مسئله $2-SAT$ است:

$$(x_1 \vee \bar{x_4}) \wedge (\bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_2) \wedge (\bar{x_1} \vee \bar{x_2}) \wedge (x_3 \vee x_1) \wedge (x_4 \vee \bar{x_3})$$

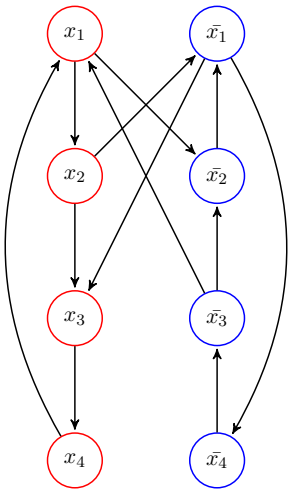
⊠

حال می‌خواهیم که مسئله را به گراف جهت‌داری مدل کنیم و آن را به کمک یافتن مؤلفه‌های قویاً همبند حل کنیم.واضح است که برای این‌که کل فرمول $true$ شود، باید تک تک عبارت ها $true$ شود.

برای رأس‌های گراف جهت‌داری که قصد ساختن آن را داریم،هر متغیر و نقیض آن را در نظر می‌گیریم.حال باید یال‌های آن را به طرز مناسبی تعریف کنیم تا مدل‌سازی کامل شود.

[7] 2-CNF satisfiability
[8] literal

برای ساختن یال ها، به طور مثال اگر عبارت $(l_i \vee l_j)$ در فرمول موجود باشد،برای این عبارت دو یال در گراف متناظر اضافه می‌کنیم: یال $\bar{l}_i l_j$ (یال از $\bar{l}_i$ به l_j) و یال $\bar{l}_j l_i$ (یال از $\bar{l}_j$ به l_i). دلیل آن به طور خلاصه این است که اگر l_i، دارای مقدار $true$ نباشد(یعنی $\bar{l}_i$ مقدارش $true$ باشد) ، آنگاه الزاماً برای این‌که مقدار این عبارت $true$ شود ، باید که l_j مقدار $true$ داشته باشد و برعکس. پس گراف را بدین صورت می‌سازیم.برای مثال گراف متناظر با عبارت ۳ به صورت زیر است:



شکل ۶: گراف متناظر با عبارت ۳

ادعا ۱. *فرمول F ارضاپذیر است اگر و تنها اگر برای هیچ متغیر x_i ، لیترال‌های x_i و $\bar{x_i}$ در یک مؤلفه قویاً همبند از $G(F)$ (گراف متناظر با فرمول F) نباشند.*

اثبات. یک طرف ادعا واضح است.یعنی اگر متغیری مانند x_i موجود باشد که x_i و $\bar{x_i}$ در یک مؤلفه قویاً همبند باشند،آنگاه فرمول ارضا پذیر نیست؛فرض کنید که اینگونه باشد، یعنی مسیری از x_i به $\bar{x_i}$ و مسیری از $\bar{x_i}$ به x_i داریم.با توجه نحوه تعریف یال‌ها،اگر یالی از x_i به x_j باشد، آنگاه اگر x_i مقدارش $true$ باشد، آنگاه باید x_j نیز $true$ باشد.پس چون مسیری از x_i به $\bar{x_i}$ داریم،پس اگر x_i بخواهد مقدارش $true$ باشد، باید $\bar{x_i}$ مقدارش $true$ که تناقض است.مشابه همین استدلال برای مسیر از $\bar{x_i}$ به x_i نیز به تناقض می‌انجامد.بنابراین نمی‌توان تمامی این محدودیت‌ها را ارضا کرد و فرمول ارضاپذیر نیست.برای مثال در گراف ۶ تمامی رئوس در یک مؤلفه قویاً همبند هستند و این نتیجه می‌دهد که فرمول ۳ ارضاپذیر نیست.

حال برعکس فرض کنید که در گراف ساخته شده ، هیچ متغیری با نقیضش در یک مؤلفه نیست.در ادامه می‌خواهیم روشی برای یافتن مقداردهی مناسب به متغیرها بیابیم طوری‌که فرمول متناظر با گراف، مقدارش $true$ شود.

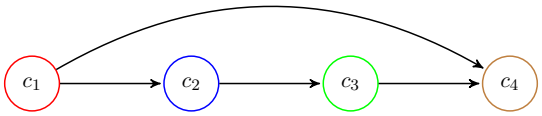
برای یافتن این مقداردهی مناسب،به روش زیر عمل می‌کنیم:

مرتب سازی توپولوژیک گراف H (گراف مؤلفه‌های قویاً همبند) را در نظر می‌گیریم. سپس مؤلفه‌ای که در این مرتب سازی در انتها قرار داد(هیچ یالی از این مولفه خارج نمی شود) را در نظر می‌گیریم (آن را c بنامید)و تمامی لیترال‌های موجود در آن را $true$ می‌کنیم. مثلاً اگر x_i در این مؤلفه باشد،آنگاه مقدار x_i را برابر با $true$ قرار می‌دهیم و اگر مثلاً $\bar{x_j}$ در این مؤلفه بود، $\bar{x_j}$ را $true$ می‌کنیم که معادل با این است که x_j را برابر با $false$ قرار دهیم.پس هر لیترال داخل مؤلفه c را $true$ می‌کنیم و سپس تمام متغیرهایی که مقدار آن‌ها مشخص شده‌است را از گراف حذف می‌کنیم(مثلا اگر x_i را $true$ کنیم،آنگاه باید $\bar{x_i}$ را نیز از گراف حذف کنیم.سپس برای گراف باقیمانده همین کار را انجام می‌دهیم.(مؤلفه‌ی بدون یال خروجی در گراف باقیمانده را در نظر می‌گیریم و لیترال‌های آن را $true$ می‌کنیم).

بیان مرحله به مرحله آن بدین صورت است:

۱ـ مرتب سازی توپولوژیک مؤلفه‌های قویاً همبند را می‌یابیم.

۲ـ مؤلفه آخر(مؤلفه بدون یال خروجی) را در نظر می‌گیریم(آن را c بنامید):



برای مثال در گراف بالا،مؤلفه‌ی c_4 مؤلفه‌ی‌آخر است .

۳ـ مقدار تمامی رئوس موجود در c را $true$ کنید.سپس رئوس متناظر با لیترال‌های داخل c و نقیض‌های آن‌ها را حذف می‌کنیم .

۴ـ (در صورت وجود داشتن رأس در گراف باقیمانده) به مرحله ۲ می‌رویم.

حال ثابت می‌کنیم که این مقداردهی فرمول را ارضا می‌کند. برای اثبات درستی آن اگر محدودیت‌ها را درست ایجاد کنیم یعنی متغیر و نقیض‌آن در یک مؤلفه قویاً همبند نباشند حتما فرمول ارضا پذیر است.فرض کنید که این مقداردهی خراب باشد.اولین جایی را در نظر بگیرید که این اتفاق رخ دهد.فرض کنید متغیر l_i را $true$ کرده‌ایم و یال $l_i l_j$ موجود است و باید که l_j مقدارش $true$ شود ولی قبلا مقدار آن $false$ تعیین شده است.بنا به تقارنی که بین ایجاد یال‌ها وجود داشت،یال $\bar{l_j}\bar{l_i}$ نیز باید وجود داشته باشد و چون مقدار l_j برابر با $false$ تعیین شده است، پس باید قبل از آن مقدار $\bar{l_j}$ برابر با $true$ باشد و سپس $l_j, \bar{l_j}$ را حذف کرده‌ایم.حال چون یال $\bar{l_j}\bar{l_i}$ وجود دارد،آنگاه باید $\bar{l_i}$ در همان مرحله‌ی تعیینِ مقدارِ $\bar{l_j}$ تعیین می‌شده است و برابر با $true$ می‌شده است که پس مقدار l_i نیز در اینجا تعیین می‌شود که تناقض است.پس این مقدار دهی فرمول را ارضا می‌کند. □

۴ پیدا کردن رأس‌های برشی در گراف بدون جهت

تعریف ۲. *رأسی که با حذف شدن آن تعداد مؤلفه‌های همبندی آن افزایش یابد ، رأس برشی گویند.*

ورودی: یک گراف بدون جهت

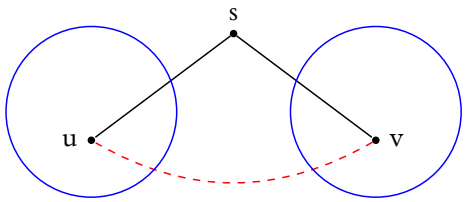
خروجی: رأس‌های برشی گراف ورودی

برای حل مسئله، ابتدا چند خصوصیت از رأس‌های برشی را ثابت می‌کنیم:

۱ـ فرض کنید رأس s ریشه‌ی یک درخت DFS باشد.در این‌صورت رأس s برشی است اگر و تنها اگر حداقل دو فرزند در این درخت داشته باشد.

اثبات. در ابتدا فرض کنید که رأس s برشی باشد. در ابتدا توجه کنید که s نمی‌تواند به یک مؤلفه همبندی پس از حذفش مثل c_1 در درخت DFS مذکور دو یال داشته باشد؛ زیرا مثلا اگر این‌گونه باشد،آنگاه اگر برای اولین بار رأس u از این مؤلفه را ببیند،آنگاه چون سایر رئوس مؤلفه c_1 از طریق u دیده می‌شوند،آنگاه دیگر s نمی‌تواند به این مؤلفه یال دیگری در درخت DFS داشته باشد.حال پس از حذف s واضح است که بین هیچ دو تا از مؤلفه‌های باقیمانده یالی نیست(در غیر این‌صورت یکی از این مؤلفه توسط مؤلفه دیگر که در ابتدا دیده می‌شود،بررسی می‌شود و دیگر در مؤلفه‌های مجزا قرار نمی گیرد). پس چون s رأس برشی است و ریشه درخت،پس باید حداقل ۲ فرزند در این درخت داشته باشد.

حال فرض کنید که که s حداقل دو فرزند در درخت DFS داشته باشد.(رئوس u, v) فرض کنید که در حین اجرای‌الگوریتم DFS



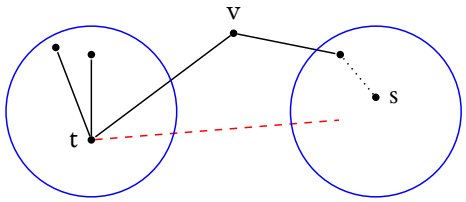
شکل ۷: درخت حاصل از DFS با ریشه s

در ابتدا رأس u را بررسی کنیم. حال اگر مسیری از u به v موجود باشد که از s نگذرد(مانند مسیر خط چین قرمز در ۷)،تا قبل از اینکه همه رئوسی که از u می‌گذرد را ملاقات[9] نکرده باشیم(از جمله رأس v) اجرای‌الگوریتم DFS بر روی D تمام نمی‌شود و بنابراین که رأس v توسط v ملاقات می‌شود و دیگر در درخت DFS به s متصل نیست که تناقض است.پس چنین مسیری وجود ندارد.پس s رأس برشی است.

□

۲ ـ اگر رأس v رأس غیر ریشه از درخت DFS باشد،آنگاه رأس v برشی است اگر و تنها اگر v فرزندی مانند t داشته‌باشد به‌طوری‌که هیچ یال بازگشتی از t یا یکی از نوادگان t به یک v (غیر از v) وجود نداشته باشد.

اثبات. فرض کنید که v رأس برشی باشد.



شکل ۸: درخت حاصل از DFS با ریشه s

حال در شکل ۸ مشابه استدلال در حالت قبلی، نتیجه می‌شود که نمی‌توان مسیری از t به مؤلفه‌همبندی c_2 (که شامل اجداد v است)، داشت(مسیر خط چین قرمز رنگ شکل ۸)؛زیرا که اگر چنین مسیری موجود باشد،آنگاه این دو مؤلفه پس از حذف v به یکدیگر از طریق این مسیر متصل‌اند و با برشی بودن v در تناقض است.

حال برعکس، فرض کنید v فرزندی مانند t داشته‌باشد به‌طوری‌که هیچ یال بازگشتی از t یا یکی از نوادگان t به یک v (غیر از v) وجود نداشته باشد.پس برای مثال در شکل ۸ ، هیچ یالی از مؤلفه c_1 به اجداد v وجود ندارد. ادعا می‌کنیم با حذف v ، مسیری بین t و s وجود نخواهد داشت. فرض کنید چنین مسیری وجود دارد.این مسیر جایی از مؤلفه c_1 خارج خواهد شد(این یال را e بنامید). در هنگام DFS در مؤلفه‌ی c_1 این یال بررسی می‌شود.حال می‌خواهیم بررسی کنیم که رنگ آن رأس از e که خارج از c_1 است(این رأس را نیز u بنامید) چیست(رنگ‌دهی ها در جلسه چهارم بررسی شده است). واضح است که رنگ u نمی‌توانسته سفید باشد(زیرا که اگر این‌گونه باشد،باید

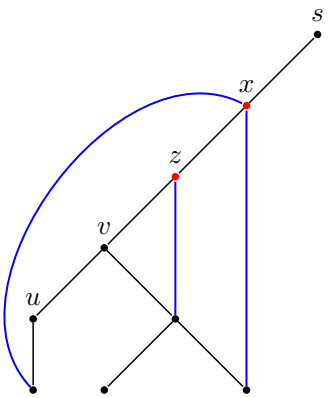
[9] visit

u در مؤلفه c_1 قرار گیرد).حال اگر u سیاه باشد،آنگاه یال e یا یال روبه‌جلو است یا یال تقاطعی و چون گراف بدون جهت است آنگاه تنها می‌توانیم یال درختی و بازگشتی داشته باشیم(به عنوان تمرین این مطلب را ثابت کنید) و بنابراین رنگ u نمی‌تواند سیاه باشد و تنها می‌تواند خاکستری باشد و یال e یال بازگشتی باشد و چون یال e بازگشتی است، پس باید u یکی از اجداد طرف دیگر یال e باشد(و نتیجه می‌شود که رأس u از اجداد v نیز هست) که با فرض اولیه ما در تناقض است.پس این قسمت نیز ثابت می‌شود. □

۳ـ پارامتر low را برای هر رأس v بدین صورت تعریف کنید:

$$v.low = \min \begin{cases} v.d \\ w.d \quad (u,w) \text{ is a backedge for some descendant } u \text{ of } v \end{cases}$$

برای مثال در درخت DFS زیر داریم که رئوس z, x گزینه‌های موجود برای w هستند.(به واسطه یال‌های آبی رنگ):



حال ثابت کنید که می‌توانیم که $v.low$ را می‌توان برای همه $v \in V$ در زمان $O(E)$ محاسبه کرد(E تعداد یال‌های گراف است).

برای اثبات این بخش می‌توانیم تعریف کنیم :

$$v.low = \min \begin{cases} v.d \\ u.low \quad u \text{ is a child of } v \\ w.d \quad (u,w) \text{ is a backedge for some descendant } u \text{ of } v \end{cases}$$

حال برای مثال در شکل فوق، فرض کنید هنگام محاسبه $v.low$ به طور بازگشتی مقدار $u.low$ محاسبه شده است و وقتی که DFS بر روی u به اتمام رسید،هنگامی‌که به رأس v باز می‌گردیم، مقداری که برای $v.low$ داریم که در ابتدا برابر با $v.d$ است را با $u.low$ مقایسه می‌کنیم و اگر کمتر بود آن را آپدیت می‌کنیم و مشابهاً برای سایر فرزندان رأس v و برای یال‌های بازگشتی نیز به همین طریق می‌توانیم آپدیت کنیم و می‌توان بررسی کرد که در زمان گفته شده قابل انجام است.

۴ـ نشان دهید که همه رأس‌های برشی G را می‌توان در $O(E)$ پیدا کرد.

برای رأس‌های برشی دسته اول (ریشه درخت DFS) می‌توان بررسی کرد که دو فرزند دارد یا نه که به راحتی قابل انجام است.

برای بررسی رأس‌های دسته‌ی‌دوم(غیر ریشه)،برای رأس غیر ریشه‌ای مانند v ، برای هر یک از فرزندان v که در آستانه تمام شدن است و باید مقدار $v.f$ را تعیین کنیم و از تابع DFS این رأس خارج شویم، مقدار low فرزندان v را بررسی می‌کنیم(مثل u) ،مقدار $u.low$ را با $v.d$ مقایسه می‌کنیم و اگر این مقدار کمتر از $v.d$ بود،بدین معناست که این رأس و نوادگانش دارای یال بازگشتی(به اجداد v) هستند و

اگر همه فرزندان v در درخت DFS دارای این ویژگی بودند،آن‌گاه دیگر v نمی‌تواند که برشی باشد ولی اگر یک رأس بود که این ویژگی را نداشت، آن‌گاه v رأس برشی است و این قسمت نیز کامل می‌شود.

مراجع

[1] CLRS Cormen, Thomas H., et al. Section 22, Problem 2 Introduction to Algorithms. 3rd ed., MIT Press, 2009

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس حل تمرین: محمد امین ولی

[بهار ۹۹]

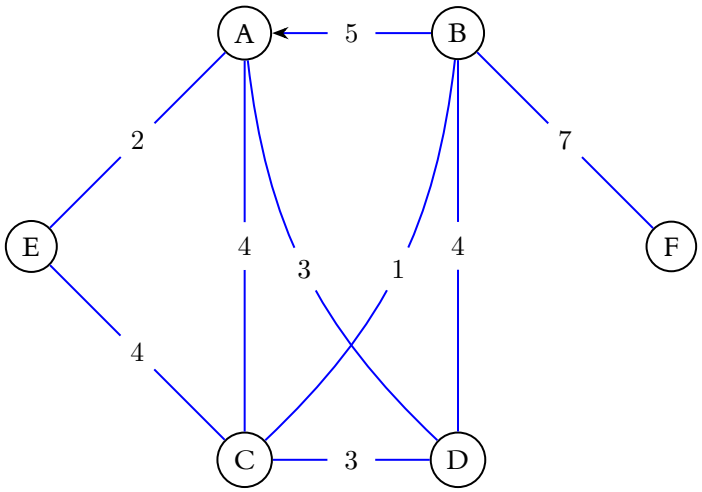
جلسه حل تمرین ۳: پیاده سازی الگوریتم‌های درخت فراگیر کمینه

نگارنده: آیسان نیشابوری

در این جلسه راه‌های پیاده‌سازی یک سری الگوریتم برای پیدا کردن درخت فراگیر کمینه [1] مورد بحث قرار گرفته شد.

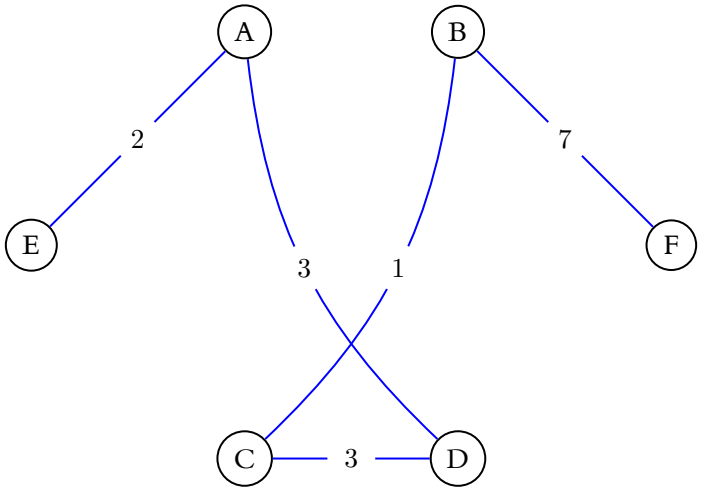
۱ الگوریتم کروسکال (Kruskal)

کلیت این الگوریتم، به این شکل است که یک سری مؤلفه ی همبندی داریم و در هر مرحله یالی که کم‌ترین وزن را دارد و تا به حال چک نشده است را چک می‌کنیم و می‌بینیم که آیا دو سر این یال، در یک مؤلفه ی همبندی گراف هستند یا نه. اگر در یک مؤلفه بودند دوباره این کار را تکرار می‌کنیم (یعنی به سراغ یال بعدی می‌رویم) در غیر این صورت این یال را به درخت‌مان اضافه می‌کنیم.



برای مثال در گراف بالا که وزن یال‌ها روی آن‌ها نوشته شده است، برای پیدا کردن درخت فراگیر کمینه این گراف با الگوریتم کروسکال، ابتدا یال با وزن یک را برمی‌داریم، سپس یال با وزن دو و پس از آن یال‌هایی که وزن سه دارند را با ترتیب دلخواهی برمی‌داریم بعد از آن به تصادف یکی از یال‌هایی که وزن چهار دارند را بر می‌داریم و می‌بینیم دو سر این یال در یک مؤلفه است، پس آن را برای درخت‌مان انتخاب نمی‌کنیم و برای همه‌ی یال‌ها به وزن چهار و سپس یال با وزن پنج این روند تکرار می‌شود و این یال‌ها به درخت‌مان اضافه نمی‌شوند تا این که یال با وزن هفت را به درخت اضافه می‌کنیم و در نهایت به درخت درخت فراگیر کمینه زیر می‌رسیم.

[1] Minimum Spanning Tree (MST)



برای پیاده‌سازی این الگوریتم ابتدا یک آرایه از یال‌ها نگه می‌داریم و فرض می‌کنیم اندازه‌ی این آرایه E باشد. Merge Sort را روی آرایه‌مان اجرا می‌کنیم که از زمان $O(E\log(E))$ می‌باشد.

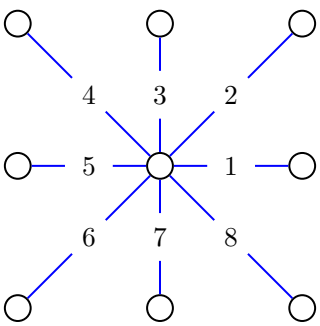
حال اگر این طور فرض کنیم که زمان لازم برای این که چک کنیم دو رأس در یک مؤلفه هستند یا نه و این که مؤلفه‌هایشان را ادغام کنیم از $O(1)$ است، کل الگوریتم در زمان $O(E\log(E)+E)$ انجام می‌شود زیرا در ابتدا که از $O(E)$ آرایه‌ی یال‌ها را می‌سازیم، سپس از $O(E\log(E))$ این آرایه را مرتب می‌کنیم و پس از این به ازای هر یال ماکزیمم یک بار دو سرش را چک می‌کنیم که درون یک مؤلفه هستند یا نه پس با این فرض ما، این کار از $O(E)$ است.

اما واقعیت این است که بررسی این که دو یال درون یک مؤلفه هستند یا نه با $O(1)$ بدیهی نیست. برای این کار ما راس‌ها را یک سری آدم فرض می‌کنیم که یک سری قبیله دارند و هر قبیله‌ای رئیسی دارد و در هر مرحله برای چک کردن این که دو رأس در یک قبیله هستند یا نه رئیس‌شان را نگاه می‌کنیم و اگر یکی بود در یک قبیله هستند، در غیر این صورت در یک قبیله نیستند. مشکلی که وجود دارد زمانی است که می‌خواهیم یالی را اضافه کنیم و دو مؤلفه از گراف را به یک مؤلفه تبدیل کنیم و در واقع گراف را آپدیت کنیم. برای این کار چند رویکرد وجود دارد:

ـ رویکرد اول این است که از بین دو رئیس دو قبیله یکی را به طور رندوم انتخاب کرده و برای تک تک آدم‌هایی که در قبیله‌ی مخالف هستند رئیس را عوض کنیم که این کار اردر زمانی خوبی ندارد زیرا برای مثال اگر گراف‌مان به شکل ستاره (مانند گراف زیر) باشد در ابتدا مؤلفه‌ها تک رأس‌ها هستند و اگر هر بار به طور تصادفی گراف تک رأس به عنوان رئیس قبیله انتخاب شود باید همه‌ی رأس‌های دیگر که درون درختی که در لحظه داریم وجود دارند رئیس‌شان عوض می‌شود و اگر گراف‌مان به شکل ستاره باشد و V رأس داشته باشد برای ادغام کردن تمام رأس‌ها در بدترین حالت باید زمان

$$1+2+...+V=O(V^2)$$

داشته باشیم و این زمان خوبی نیست.



ـ رویکرد دوم این است که از ساختار DSU [۲] استفاده کنیم که در این ساختار ادغام کردن برای یک مجموعه‌ی V تایی از $O(\log V)$ می‌باشد. چون ما در ابتدا V تا رأس و به همین تعداد مؤلفه‌ی همبندی داریم و در آخر الگوریتم کروسکال یک مؤلفه داریم، پس V تا ادغام انجام می‌دهیم و اردر کل ادغام‌هایمان $O(V \log V)$ می‌باشد.

پس در کل اگر با رویکرد دوم جلو برویم الگوریتم کروسکال اردر زمانی $O(V \log V + E \log E)$ خواهد داشت.

سوال: اگر یک گراف با یال های قرمز و آبی داشته باشیم در چه صورت درخت فراگیری از این گراف وجود دارد که تعداد یال‌های قرمز و آبی آن برابر باشد؟

۲ الگوریتم پریم (Prim)

در این الگوریتم، ما ابتدا یک مجموعه‌ی خالی مثل T داریم که قرار است درخت‌مان در نهایت درون آن قرار بگیرد و یک مجموعه‌ی G که در ابتدا از همه‌ی رأس‌ها و یال‌های گراف تشکیل شده است و در آخر هیچ رأسی ندارد. در مرحله‌ی اول یک رأس تصادفی مانند v از G برمی‌داریم و به مجموعه‌ی T اضافه می‌کنیم. در مراحل بعد تمام یال‌هایی که یک سرشان در T و سر دیگرشان درG است را بررسی می‌کنیم و از بین این‌ها آنی که کم‌ترین وزن را دارد انتخاب می‌کنیم، فرض کنید این یال درT به u و در G به v وصل باشد. این یال را همراه با v به مجموعه‌ی T اضافه می‌کنیم و v را از G حذف می‌کنیم.

حال اگر بخواهیم تحلیل زمانی کنیم، در هر مرحله باید یالی که وزن مینیمم دارد و بین مجموعه‌ی T و G است را انتخاب کنیم. در بدترین حالت بین این دو $|T| \times |G|$ تا یال وجود دارد که چون $|T| + |G| = n$ است (n تعداد رأس‌های گراف است) این مقدار در بدترین حالتی که $|G| = |T| = \frac{n}{2}$ باشد برابر $\frac{n^2}{2}$ است که از $O(n^2)$ است. در کل هم n بار این کار را انجام می‌دهیم، چون هر بار یک رأس از G به T اضافه می‌کنیم و در کل الگوریتم از $O(n^3)$ می‌شود.

برای این که این اتفاق نیافتد یک آرایه از رأس‌ها نگه می‌داریم و این آرایه را این طور تعریف می‌کنیم که اگر رأس i ام در T بود خانه‌ی i ام آرایه را برابر -2 قرار میدهیم و اگر این رأس به T یال نداشته باشد، خانه‌ی آرایه را برابر -1 قرار می‌دهیم در غیر این صورت اگر e یالی با کم‌ترین وزن باشد که رأس i ام را به رأس u در G وصل می‌کند درون خانه‌ی i ام آرایه دوتایی (u, e) را نگه می‌داریم. حال در هر مرحله برای پیدا کردن یالی که بین T و G است و کم‌ترین وزن را دارد کافی است یک دور آرایه را طی کنیم و برای هر خانه ای که -1 و -2 نباشد، وزن یال‌اش را چک کنیم و یال مورد نظرمان را پیدا کنیم که این کار از $O(n)$ انجام می‌شود.

برای آپدیت کردن آرایه هم کافی است خانه‌ی رأسی که به T منتقل شده (که آن را v می‌نامیم) را برابر -2 قرار دهیم و سپس لیست رأس‌های مجاور v را چک کنیم و برای هرکدام مانند u که درون G بود و با یال e به v وصل بود، ببینیم وزن e از وزن یال موجود در خانه‌ی رأس u در آرایه، کم‌تر است یا نه و اگر کم‌تر بود آن را آپدیت کنیم و اگر خانه‌ی متناظر با u برابر -1 بود آن را برابر (v, e) قرار دهیم. این کار از $O(n)$ انجام می‌شود زیرا v حداکثر $n - 1$ راس مجاور دارد.

پس با این کار، ما در هر مرحله کاری با زمان $O(n)$ انجام می‌دهیم و همان طور که قبلا گفته شد در کل n مرحله داریم، نتیجتا الگوریتم ما در زمان $O(n^2)$ انجام می‌شود.

[۲] Disjoint-set/Union-find Forest

۳ الگوریتم برووکا (Boruvka)

کلیت الگوریتم برووکا، به این شکل است که یک سری مؤلفه داریم (در ابتدا هر کدارم از رأس‌هایمان یک مؤلفه است) و در هر مرحله هر کدام از مؤلفه‌ها مانند U_0 میبیند که کم وزن‌ترین یالش به فضای بیرون (به رأس‌هایی که درون مؤلفه‌ی خودش نیستند) کدام یال است، که فرض کنیم این یال e باشد و به رأسv درون مؤلفه‌ی U_1 وصل باشد در این صورت با اضافه کردن e به MST این دو مؤلفه را یکی می‌کنیم. خوبی این الگوریتم این است که به طور موازی با چند کامپیوتر می‌تواند انجام شود، چرا که یال‌های بین مؤلف ها به طور جداگانه بررسی می‌شوند.

نکته‌ای که درون این الگوریتم وجود دارد، این است که وزن یال‌ها نباید یکسان باشد، زیرا اگر برای مثال بین دو مؤلفه‌ی U_0 و U_1 دو یال e_1 و e_2 هر دو با وزن k وجود داشته باشد و این دو یال کم وزن‌ترین یال‌های خارج شده از U_0 و U_1 باشند و در مرحله‌ای U_0 ، e_1 را انتخاب کند و U_1 ، e_2 در این صورت بین این دو مؤلفه دو یال اضافه می‌شود و دور ایجاد می‌شود که باعث می‌شود در نهایت به درخت نرسیم و این مشکل ایجاد می‌کند.

برای این الگوریتم، یک آرایه‌ی به اندازه‌ی E از یال‌ها نگه می‌داریم و اگر آن یال درون مؤلفه‌ای باشد، خانه‌ی متناظر با آن را برابر -1 قرار می‌دهیم. در غیر این صورت درون آن خانه‌ی آرایه وزن یال و رأس سر و ته آن را نگه می‌داریم و یک سری مجموعه داریم که مؤلفه‌هایمان هستند و آن‌ها را با DSU پیاده‌سازی می‌کنیم و برای هر مؤلفه کم وزن‌ترین یال‌اش را به بیرون نگه می‌داریم.

در هر مرحله، کم وزن‌ترین یال هر مؤلفه به بیرون از خودش را داریم و مؤلفه های دو سر این یال ها را ادغام می‌کنیم و می‌دانیم همه‌ی ادغام‌ها با زمان $V \log V$ انجام می‌شود (مانند الگوریتم پریم). حال کافی است آرایه‌ی یال‌هایمان را آپدیت کنیم و برای هر مؤلفه هم کم وزن‌ترین یالش را به بیرون آپدیت کنیم.

برای آپدیت کردن آرایه‌ی یال‌ها، کافی است برای هر یال سر و ته آن را نگاه کنیم و ببینیم در یک مؤلفه هستند یا نه و اگر بودند در خانه‌ی متناظر با آن یال 1- بگذاریم. در غیر این صورت وزن یال و رأس سر و ته آن درون این خانه قرار دارد و تغییری در آن ایجاد نمی‌کنیم، این کار برای هر یال از $O(1)$ انجام می‌شود، پس برای همه‌ی یال‌ها از $O(E)$ انجام می‌شود.

برای آپدیت کردن کم وزن‌ترین یال هر مؤلفه، روی آرایه‌ی یال‌ها حرکت می‌کنیم و برای هر یال مانند e مؤلفه‌های رأس‌های دو سر آن را از $O(1)$ می‌یابیم، حال فرض می‌کنیم این مؤلفه‌ها U_0 و U_1 باشند، اگر کم وزن‌ترین یال U_0 ، k باشد، می‌بینیم که وزن e از وزن k کم‌تر است یا نه. اگر کم تر بود e را کم وزن‌ترین یال مؤلفه‌ی U_0 قرار می‌دهیم در غیر این صورت آن را تغییر نمی‌دهیم. برای U_1 و e هم همین کار را تکرار می‌کنیم. این عملیات برای هر یال از $O(1)$ است، پس در کل از $O(E)$ می‌باشد.

در کل هم حداکثر $\log V$ مرحله داریم زیرا در هر مرحله هر مؤلفه حداقل با یک مؤلفه ی دیگر ادغام می‌شود، پس در هر مرحله تعداد مؤلفه‌ها حداقل نصف می‌شوند، در نتیجه مراحل بیش‌تر $\log V$ تا نمی‌توانند باشند. در هر مرحله هم عملیاتی از $O(E)$ انجام می‌دهیم، پس در کل عملیاتی از $O(E \log V)$ انجام می‌دهیم و خود ادغام‌ها هم از $O(V \log V)$ است (مانند الگوریتم پریم)، پس زمان الگوریتم‌مان در کل از $O(E \log V + V \log V)$ است و چون گراف‌مان قرار است درخت فراگیر داشته باشد، تعداد یال‌هایش حداقل $V - 1$ تاست و کل الگوریتم از زمان$O(E \log V)$ می‌باشد.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: علیرضا توفیقی محمدی

[بهار ۹۹]

جلسه حل تمرین ۴: برنامه ساز پویا

نگارنده: مهدی مستانی

در این جلسه قصد داریم راجع به برخی از مسائل برنامه ساز پویا [1] (DP) بحث کنیم.

۱ اعداد شلخته

به عدد a **شلخته** گوییم اگر صفر های سمت چپ عدد را در نظر نگیریم،اختلاف هر دو رقم مجاور آن حداقل دو واحد باشد. برای مثال عدد ۱۵۹۳ عددی شلخته است ولی عدد ۱۵۳۲ شلخته نیست ($(3-2=1)<2$) . حال در ورودی دو عدد l و r ($r \geq l$) و می خواهیم تعداد اعداد شلخته در بازه $[l,r]$ را به دست آوریم.

فهمیدن اینکه یک عدد شلخته است ، اگر عدد ما k رقمی باشد، در زمان $o(k)$ می‌توانیم بفهمیم این رقم شلخته است یا خیر.

یک راه حل ساده برای حل این سوال این است که تمام اعداد در بازه $[l,r]$ را بررسی کنیم و تعداد آنها را به عنوان خروجی بدهیم.

یک راه دیگر این است که جواب را برای بازه $[1,r]$ و از جواب های در بازه $[1,l]$ کم کنیم. پس مسئله تبدیل به این می‌شود که تعداد اعداد شلخته در بازه $[1,r]$ به دست آوریم.

برای این سوال می خواهیم که الگوریتمی ارائه دهیم که در زمان $o(|r|)$ مسئله را حل کنیم که $|r|$ اندازه طول عدد r است.

اگر تعریف کنیم $dp[i][j]$ را بدین صورت که اعداد شلخته i رقمی که رقم سمت چپ آن j است.در این صورت عدد ما شلخته است، اگر $i-1$ رقم سمت راست آن شلخته باشد و اگر رقم $i-1$ ام را آن را x بنامیم، آنگاه باید $|x-j| \leq 2$ باشد. حال اعداد حداکثر k رقمی شلخته را می یابیم که k تعداد ارقام عدد r است.

حال واضح است که می توانیم رابطه بازگشتی زیر را برای $dp[i][j]$ را بنویسیم:

$$dp[i][j] = \sum_{\substack{x \in \{0,1,\ldots,9\} \\ |x-j| \leq 2}} dp[i-1][x]$$

حال اگر عدد ما بدین صورت باشد : $r = \overline{r_1 r_2 \ldots r_k}$ و عدد x ما بدین صورت : $x = \overline{x_1 x_2 \ldots x_t}$ ، آنگاه برای عدد x دو حالت داریم:

۱) $|x| < |r|$: در حالت داریم تعداد اعداد شلخته ۱ رقمی ، ۲ رقمی ، ...و $|r-1|$ رقمی را محاسبه می‌کنیم . برای محاسبه این مقادیر، می‌توانیم مقدار زیر را بیابیم :

[1] dynamic programing

$$\sum_{i=1}^{|r-1|}\sum_{j=1}^{9} dp[i][j]$$

مقدار بالا تمام اعداد حداکثر $|r-1|$ رقمی و شلخته را محاسبه می‌کند.

۲) $|x| = |r| = k$: اگر r و x را به صورت زیر در نظر بگیریم، $r = \overline{r_1 r_2 \dots r_k}$ و $x = \overline{x_1 x_2 \dots x_k}$ آنگاه اگر $x < r$ باشد، آنگاه وجود دارد i که $x_1 = r_1$ و $x_2 = r_2$ و ...و $x_i = r_i$ ولی $x_{i+1} < r_{i+1}$. حال اگر روی مقدار i حالت بندی کنیم و تعداد اعدادی را حساب کنیم که k رقمی هستند و تا رقم i ام با r یکسان است ولی در رقم $i+1$ ام از r کوچک تر است و همینطور الی آخر. در واقع می‌خواهیم به ازای هر i ($0 \le i \le k-1$) تعداد اعداد شلخته را حساب کنیم که k رقمی است و تا رقم $i-1$ ام با r یکسان اند و در رقم i ام از r کوچکتر اند. برای مثال اگر $r_1 > x_1$ آنگاه باید مقدار $\sum_{r_1<p} dp[k][p]$ را محاسبه کرد و همین کار را برای تمامی i ها انجام می دهیم و در نهایت مقدار نهایی اعداد شلخته کوچکتر از r را محاسبه می کنیم.(توجه کنید مثلا اگر $|(r_2 - r_3)| = 1$ بود ، دیگر برای هر $i > 3$ ، مقدار dp آنها صفر است ؛ زیرا که دیگر x نمی تواند شلخته باشد)

می توان با کمی محاسبه، تعداد دقیق این اعداد را با استفاده از توضیحات بالا به دست آورد.

۲ شطرنج بلاک شده

یک صفحه شطرنج $n \times n$ داریم که تعدادی از خانه های آن مسدود [۲] شده است و در آن خانه ها نمی توانیم مهره ای را قرار دهیم. می خواهیم در خانه های غیر مسدود این جدول n مهره قلعه(رخ) قرار دهیم به طوری که هیچ دو تایی یکدیگر را تهدید نکنند (مهره رخ در هر خانه ای که قرار گیرد، تمام خانه های آن سطر و آن ستون را تهدید می‌کند). می‌خواهیم تعداد حالت های قرار دادن این n مهره را بیابیم.

یک راه حل برای این سوال این است که همه جایگشت ها را چک کنیم. برای این کار $n!$ حالت داریم که برای چک کردن این که یک حالت درست است یا خیر ، به $o(n)$ زمان نیاز داریم که نتیجه می‌دهد این راه حل به زمان $o(n \times n!)$ نیاز دارد.

حال راه دیگر این است که تعریف کنیم [ستون های بلاک شده] dp[i] بدین صورت که این مقدار برابر است با تعداد حالاتی که یک رخ را در i سطر اول جدول قرار دهیم و هیچ رخی در ستون های مسدود شده قرار نگیرد.

برای این کار اگر خانه های مسدود شده را در لیستی به نام list قرار دهیم داریم و ستون های مسدود شده را در مجموعه ای به نام S قرار دهیم ، داریم:

$$dp[i][S] = \sum_{\substack{(i,j)\notin list \\ j\in\{1,2,\dots,n\} \\ j\notin S}} dp[i-1][s \cup \{j\}]$$

در واقع در این راه حل ، اگر یک مهره را توانستیم در یک خانه (i,j) قرار دهیم،آنگاه دیگر در هیچ یک از خانه های ستون j ، نمی توان مهره ای قرار داد(در غیر این صورت توسط مهره قرار داده شده تهدید می‌شود) و در واقع خانه های این ستون، مسدود می‌شوند .

ما میخواهیم برای حل این سوال مقدار $dp[n][\emptyset]$ را محاسبه کنیم.

حال این کار ما $n \times 2^n$ حالت دارد و هر حالت به زمان $o(n)$ نیاز دارد و نتیجه می‌شود که این روش به $o(n^2 \times 2^n)$ زمان نیاز دارد.

اگر کمی دقت کنیم ، در ابتدا $dp[n][\emptyset]$ را اجرا می کنیم و سپس [مجموعه یک عضوی] dp[n-۱] و ...(در واقع در مرحله i+۱ ام [مجموعه $n-i$ عضوی] dp[i] فراخوانی می‌شود) .

[۲] block

حال این گونه فراخوانی از بالا به پایین در زمان $o(n \times 2^n)$ اجرا می‌شود ولی حافظه مصرفی ای برابر $o(n \times 2^n)$ دارد.

۳ بازه ای از آرایه ها

k آرایه به طول n داریم. می‌خواهیم یک آرایه $k+1$ عضوی بسازیم به مانند $p_0, p_1, \ldots, p_k$ به طوری‌که $p_0 = 0, p_k = n$ و همینطور داشته باشیم: $p_i \leq p_{i+1}$ ساختن این دنباله هزینه ای دارد که برابر است با: ($\sum_{i=1}^{k} \sum_{j=p_{i-1}+1}^{p_i} a_{ij}$) . هدف ما این است که آرایه P را طوری بیابیم که این هزینه کمترین مقدار باشد. می توان این مسئله را با یک جدول $k \times n$ متناظر کرد که در خانه های این جدول اعدادی نوشته شده اند (همان اعداد k آرایه n عضوی) و می خواهیم بازه هایی از خانه های جدول را انتخاب کنیم که هیچ دو بازه ای اشتراک نداشته باشند و اگر بازه‌ی i از سطر j ام انتخاب شده باشد و شامل خانه های k ام تا s ام باشد، برای بازه های با شماره بزرگتر از i ، باید این بازه از سطر بزرگتر از j انتخاب شود و همینطور داشته باشیم که شامل خانه های بزرگتر با شماره ستون بزرگتر از s است.همینطور از هر ستون باید دقیقا یک خانه انتخاب شود.

برای مثال فرض کنید ۳ آرایه ۴ عضوی داریم بدین صورت:

9	4	10	7
4	5	2	6
1	2	5	3

تعدادی از حالات انتخاب بازه ها را در زیر مشاهده می‌فرمایید:

9	4	10	7
4	5	2	6
1	2	5	3

9	4	10	7
4	5	2	6
1	2	5	3

9	4	10	7
4	5	2	6
1	2	5	3

9	4	10	7
4	5	2	6
1	2	5	3

حال تعبیر دیگری از مسئله تعریف شده بدین صورت است که فرض می‌کنیم که می خواهیم از خانه بالا راست جدول به پایین چپ جدول برویم و تنها به راست یا پایین جدول برویم و به هر خانه ای که سمت راست برویم ، باید به اندازه مقدار آن خانه هزینه بدهیم و پایین بیاییم، هزینه ای نداریم و می خواهیم کمترین هزینه را بدهیم.

حال اگر بخواهیم زیر مسئله را تعریف کنیم بدین صورت است که فرض کنیم در خانه (i, j) هستیم ، یا سمت راست می‌رویم و هزینه a_{ij} را می دهیم و در خانه $(i, j-1)$ قرار می‌گیریم و یا پایین می آییم و در خانه $(i-1, j)$ قرار می‌گیریم و باید ببینیم که حرکت ما به سمت پایین هزینه کمتری دارد یا حرکت به سمت راست .بیان توضیحات داده شده به زبان dp به صورت زیر است:

$$dp[i][j] = min\{dp[i-1][j], dp[i][j-1] + a_ij\}$$

پاسخ نهایی و مطلوب مسئله با محاسبه مقدار $dp[k][n]$ برابر است .

۴ سوال پیشنهادی

فرض کنید شهری داریم که n برج دارد و بین هر دو برج دیواری کشیده ایم. ($n-1$ دیوار) . دشمنی به این شهر حمله کرده است. یک پهلوان(که از قضا هیکلی نیز هست)از این شهر، تصمیم می گیرد که روی دیوار های شهر حرکت کند تا دشمن را بترساند اما دیوار i ام ظرفیت a_i دارد و پس از a_i بار حرکت روی آن ، دیوار فرو می‌ریزد. این پهلوان می تواند از هر یک از برج ها شروع کند و روی دیوار ها حرکت کند. اما می‌خواهد طولانی ترین مسیر ممکن را طی کند تا دشمن را بترساند و دیوار ها فرو نریزد(اگر فرو بریزد بد جوری ضایع می‌شود). این طولانی ترین مسیر را بیابید.

مراجع

[1] Cormen, Thomas H., et al. *Introduction to Algorithms*. 3rd ed., MIT Press, 2009, pp. 18-22.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس حل تمرین : علیرضا توفیقی محمدی

[بهار ۹۹]

جلسه ۵: کاربردهایی از مساله ی تطابق و جریان بیشینه

نگارنده: فاطمه توحیدیان

در این جلسه مسائلی بر مبنای الگوریتم های ارائه شده در جلسه های ۱۳ و ۱۴ درس مانند یافتن جریان بیشینه در یک شبکه ، الگوریتم تطابق بیشینه در گراف دوبخشی و برش کمینه خواهیم دید.

۱ مروری بر مباحث جلسات ۱۳ و ۱۴

در مساله ی جریان بیشینه گراف جهت دار $G(V,E)$ ودو راس متمایز s و t و برای همه ی یال ها یک ظرفیت نامنفی c داده شده است . تابع f را به گونه ای بیابید که جریان نسبت داده شده به هر یال از ظرفیت یال کمتر باشد و مجموع جریان های ورودی یک راس با مجموع جریان های خروجی از آن راس برابر باشد.

$cap(A,B) = \sum_{eoutof\,A} c(e)$ را در گراف $G(V,E)$ در نظر بگیرید به طوری که $s \in A$ و $t \in B$ باشد. میدانیم $cut(A,B)$ طبق قضیه ای داشتم به اگر f یک جریان و $cut(A,B)$ یک برش باشد داریم : $val(f) \leqslant cap(A,B)$

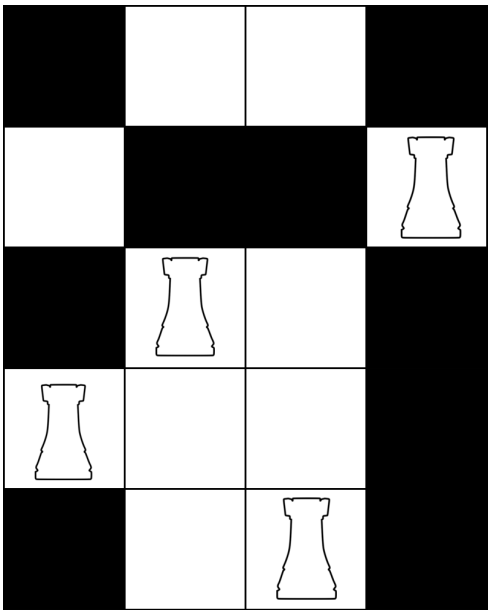
علاوه بر آن ظرفیت برش کمینه با مقدار جریان بیشینه برابر است . مساله ی جریان بیشینه را با استفاده از گراف باقی مانده ی گراف داده شده حل میکردیم . هر گاه دیگر نمی توانستیم در گراف باقی مانده نسیری از s به t پیدا کنیم الگوریتم متوقف می شد . برای پیدا کردن برش کمینه نیز تمام رئوس قابل دسترس از s را در مجموعه ی A قرار میدادیم و بقیه ی رئوس را در مجموعه ی B قرار میدادیم. همچنین راهکار هایی برای انتخاب هوشمندانه ی کوتاه ترین مسیر از s به t ارائه دادیم که در نهایت با انتخاب کوتاه ترین مسیر با استفاده از BFS توانستیم به زمان بهتری برسیم .

برای حل مساله ی تطابق بیشینه در گراف دو بخشی نیز رئوس فرضی s و t را به گراف اضافه کردیم و از s به همه ی رئوس موجود در بخش چپ گراف یال با ظرفیت ۱ و همچنین از تمام رئوس مجود در بخش راست گراف به راس t یال با ظرفیت ۱ اضافه کردیم. سپس مساله ی جریان بیشینه را برای گراف جدید حل کردیم.

۲ سوالات :

سوال ۱. یک جدول شطرنجی $n \times m$ که تعدادی از خانه های آن مسدود است داده شده است . می خواهیم در این صفحه تعدادی مهره ی رخ قرار دهیم به طوری که هیچ دو مهره ی رخی همدیگر را تهدید نکنند . دو مهره ی رخ همدیگر را تهدید می کنند اگر در یک سطر مشترک یا در یک ستون مشترک باشند همچنین در خانه های مسدود نمی توان مهره قرار داد . بیشینه تعداد مهره رخی را که میتوان در صفحه قرار داد به طوری که هیچ دو مهره رخی یکدیگر را تهدید نکنند بیابید.

مثال: شکل زیر نشان دهنده ی یک چینش مناسب مهره های رخ در صفحه هستند .



راه حل : گراف دو بخشی $G(L \cup R, V)$ را تعریف میکنیم به طوری که به ازا ی هر سطر i راس $v_i \in L$ و به ازای هر ستون j راس $u_j \in R$. از راس متناظر با سطر i به راس متناظر با ستون j یال وجود دارد اگر خانه ی متناظر با سطر i و j گراف مسدود نباشد . حال مساله ی تطابق بیشینه رابرای گراف G حل میکنیم.

تمرین : تناظر یک به یک میان یک تطابق بیشینه در G و یک چینش مناسب مهره ها در صفحه را ثابت کنید.

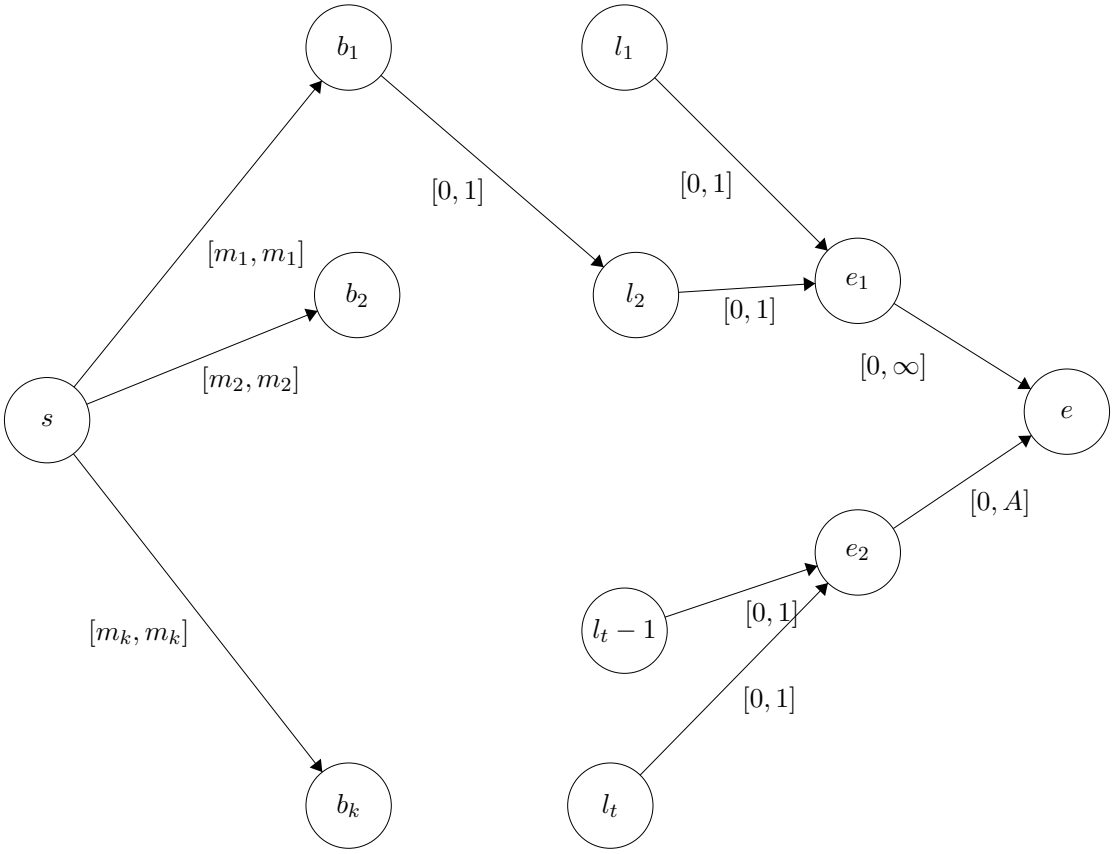
سوال ۲. یک دانشجو می خواهد فارغ التحصیل شود . برای فارغ التحصیل شدن باید از هر سبد $B_i = \{l_{1,1}, ...l_{1,n_i}\}$ که l_j ها دروس هستند ، m_i درس را پاس کند تعداد کل سبد ها k است . سبد ها می تواند با هم اشتراک داشته باشند اما یک درس را فقط می تواند به ازای یک سبد مشخص گذراند .همچنین دانشجو دروس $l'_1, .., l'_t$ را تا کنون گذرانده است.

الف)آیا دانشجو می تواند فارغ التحصیل شود؟

ب)کمینه تعداد درسی را که داشنجو باید پاس کند تا فارغ التحصیل شود بیابید.

راه حل : در جلسه ی چهاردهم مساله ای را حل کردیم که در آن مساله ی جریان بیشینه با شرط حداکثر b_i و حداقل a_i بودن جریان از یال i ام گراف حل شد . حال گراف دو بخشی $G(L \cup R, V)$ را تعریف کنید که L مجموعه ای شامل تمام سبد ها و R مجموعه ای شامل تمام دروس است.

حل ۱ :حال دو راس e_1 و e_2 را در نظر بگیرید از مجموعه ی R رئوسی که درس متناظر با آنها گذرانده شده را به e_1 و بقیه ی رئوس را به e_2 با یال هایی با ظرفیت حداکثر ۱ و حداقل ۰ وصل میکنیم . از e_1 به e یالی با ظرفیت حداکثر ∞ و حداقل صفر وصل میکنیم . از e_2 به e یالی با حداکثر ظرفیت A و حداقل ظرفیت ۰ وصل میکنیم. حل مساله ی تطابق بیشینه روی گراف G نشان میدهد آیا می توان با پاس کردن حداکثر A درس فارغ اتحصیل شد یا نه. هدف کمینه کردن یال های خروجی e_2 است . با جست و جوی دو دویی روی A مساله را حل میکنیم.



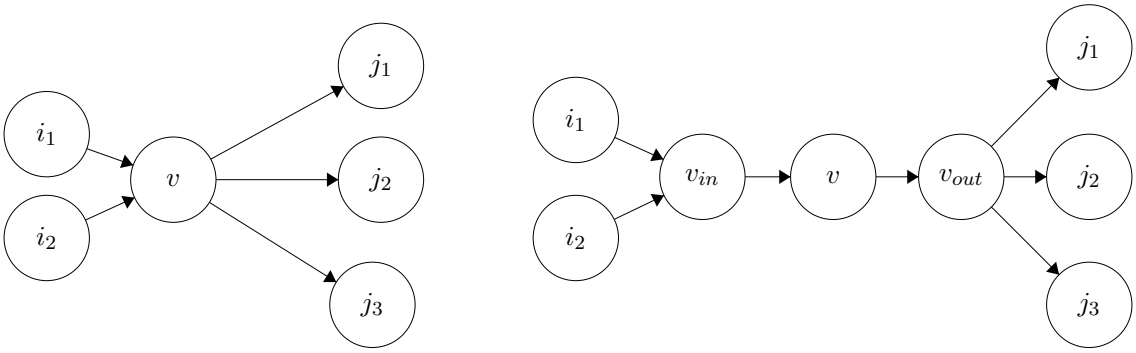
حل۲ :در حل مساله ی تطابق بیشینه اگر در هنگام پیدا کردن مسیر از s به t با استفاده از BFS بر روی یال ها یک ترتیب مشخص کنیم ، می توان مساله ی تطابق بیشینه را به گونه ای حل کرد که یال های خروجی به ترتیب الفبایی مرتب شده باشند . با توجه به این موضوع اگر در گراف G برای یال های متناظر با دروسی که گذرانده ایم اولویت بالایی انتخاب کنیم این تضمین وجود دارد که این دروس حتما انتخاب می شوند ، حال کافی است تعداد دروس گذرانده شده را از جواب نهایی کم کنیم.

سوال ۳. یک گراف جهت دار $G(V,E)$ داده شده است. $v \in V$ را مرکز گوییم به طوری که برای هر $u \in V$ داشته باشیم $uv, vu \in E$ (از مرکز به خودش نیز باید یالی وجود داشته باشد) . گراف را خوب میگوییم هر گاه برای هر $v \in V$ که راس v مرکز نیست ، درجه ورودی و خروجی برابر ۲ باشد. تغییر در گراف شامل حذف یک یال یا اضافه کردن یک یال است .کمینه تعداد تغییر لازم در گراف را بیابید به طوری که گراف پس از تغییرات خوب باشد.

راه حل : به ازای هر $v \in V$ راس v را مرکز گراف فرض میکنیم سپس تغییرات لازم را اعمال کرده و در نهایت میان تمام v ها کمینه را به عنوان جواب چاپ میکنیم. اگرi i مرکز باشد باید از همه ی راس های گراف و از همه ی رئوس گراف به i یال وجود داشته باشد. اگر d_i^+ درجه ی ورودی راس i و d_i^- درجه ی خروجی این راس باشد باید $n - d_i^+ + n - d_i^- - 1$ یال به گراف اضافه کرد تا شرط مرکز بودن i برقرار شود. حال راس i را از گراف حذف میکنیم در یک گراف خوب بعد از حذف مرکز درجه ی ورودی و خروجی رئوس باقی مانده باید ۱ باشد .گراف $G'(L \cup R, V)$ را تعریف می کنیم به طوری که به ازای هر $j \in V$، $j \neq i$ یک راس v_j در L و یک راس u_j در R وجود داشته باشد. اگر در گراف اصلی از راس k به راس z یال وجود داشته باشد ما در G' از v_k به u_z یال وصل میکنیم . با پیدا کردن یک تطابق بیشینه در G' در حقیقت داریم بیشترین تعداد یال ها از گراف اصلی را که می توان نگه داشت تا درجه ورودی و خروجی رئوس ۱ باشند را پیدا می کنیم اگر تطابق بیشینه k یال داشته باشد باید $n - k$ یال اضافه کنیم.

سوال ۴. در جلسه ی ۱۴ دیدیم که برای یک گراف $G(V,E)$ مسیرهای مجزای یالی از s به t را چگونه می توان پیدا کرد. حال می خواهیم مسیرهای مجزای راسی را از s به t پیدا کنیم.

راه حل : گراف G' را تعریف میکنیم به طوری که به ازای هر راس $v \in V$ ما ۳ راس v_{in}, v, v_{out} را به G' اضافه میکنیم. همچنین همه ی یال های ورودی v را به v_{in} و همه ی یال های خروجی از v را از v_{out} وصل میکنیم . با پیدا کردن یک مسیر مجزای یالی در G' یک مسیر مجزای راسی در G پیدا میشود. چرا که راس v در G' حداکثر یک بار ظاهر میشود.



تمرین $multiSet$ مجموعه ای ست که میتواند عضو تکراری داشته باشد اما همچنان ترتیب در آن اهمیت ندارد. یک$multiSet$ از Set ها داده شده است .یک $multiSet$ A را خوب می گوییم هر گاه داشته باشیم :$\forall S \in A : \forall x \in S \Rightarrow x \in A$. میخواهیم A را به گونه ای تغییر دهیم که خوب باشد. تغییر شامل اضافه کردن یک عضو به A یا حذف یک عنصر از A است . کمینه تعداد تغییراتی را که می توان اعمال کرد تا A خوب شود را بیابید.

مراجع

[۱] سوال ۱

[۲] hackerearth.com

[۳] سوال ۳

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس حل تمرین: مرتضی علیمی

[بهار ۹۹]

جلسه ۶: الگوریتم‌های گراف

نگارنده: کیانا عسگری

در این جلسه تعدادی از مسأله‌هایی که بیان شده‌اند ولی حل نشده‌اند را بررسی می‌کنیم.

حل تعدادی مسأله

مسأله یک: فرض کنید یک گراف وزن‌دار G داده شده، و یال‌ها با دو رنگ آبی یا قرمز رنگ شده‌اند. هدف پیدا کردن یک درخت پوشای کمینه[1] است بطوری‌که نصف یال‌های آن آبی و نصف دیگر قرمز باشند (فرض می‌کنیم این زیردرخت حتماً وجود دارد و ما فقط می‌خواهیم آن را پیدا کنیم).

اثبات. برای حل، ابتدا دو لم زیر را ثابت می‌کنیم:

لم ۱. *فرض کنید در گراف وزن دار G دو زیر درخت پوشای کمینه $\mathbf{MST}_1$ و $\mathbf{MST}_2$ داریم. فرض کنید از $\mathbf{MST}_1$ یک یال دل‌خواه e را حذف کرده‌ایم. در این‌صورت یالی در $\mathbf{MST}_2$ مانند e' وجود دارد که با اضافه کردن آن به $\mathbf{MST}_1$ دوری ایجاد نشود و وزن آن با وزن یال حذف شده برابر باشد.* (*در حقیقت کافیست بگوییم یالی با وزن کم‌ترمساوی وجود دارد و طبق ویژگی درخت پوشای کمینه واضح است که وزن این یال نمی‌تواند کمتر از وزن یال حذف شده باشد*).

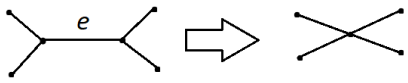
اثبات. با حذف یک یال e از درخت اول، دو مولفه باقی می‌مانند که هرکدام روی مجموعه رأس‌های خود درخت هستند و رأس‌های گراف را افراز می‌کنند؛ پس اکنون در حقیقت یک برش[2] روی گراف G داریم. حال تمام یال‌های این برش را که با یال‌های درخت $\mathbf{MST}_2$ مشترک هستند را مجموعه S در نظر بگیرید. چون $\mathbf{MST}_2$ همبند است پس S ناتهی است و همچنین واضح است اضافه کردن هر کدام از این یال‌ها به $\mathbf{MST}_1$ باعث همبند شدن آن می‌شود بدون آن که دوری ایجاد شود. حال با برهان خلف فرض کنید وزن تمام یال‌های مجموعه S از وزن یال e (حذف شده در $\mathbf{MST}_1$) بیش‌تر باشند. در این‌صورت با حذف یال دل‌خواهی مانند e' در مجموعه S از $\mathbf{MST}_2$ و هم‌چنین اضافه کردن یال e به $\mathbf{MST}_2$، این درخت همبند باقی می‌ماند ولی چون وزن یال اضافه شده اکیداً کم‌تر از وزن یال حذف شده است، پس ما توانسته‌ایم وزن $\mathbf{MST}_2$ را کاهش دهیم که تناقض است. پس فرض اولیه ما اشتباه بود و حتماً یالی با ویژگی بیان شده وجود دارد. □

لم ۲. *فرض کنید $\mathbf{MST}_1$ یک زیر درخت پوشای کمینه دل‌خواه از گراف اصلی باشد، ولی تعداد یال‌های قرمز آن از تعداد یال‌های آبی آن بیشتر باشند. همچنین فرض کنید $\mathbf{MST}_2$ یک زیردرخت پوشای کمینه از گراف اصلی باشد که تعداد یال‌های قرمز آن با تعداد یال‌های آبی آن برابر است. در این صورت یال قرمزی در $\mathbf{MST}_1$ وجود دارد به‌طوری‌که اگر برش حاصل از حذف آن را در نظر بگیریم، تمام یال‌های این برش که در $\mathbf{MST}_2$ نیز آمده‌اند، آبی باشند.*

[1] MinimumSpanningTree

[2] Cut

اثبات. ابتدا تمام یال‌های آبی در $\mathbf{MST}_1$ را منقبض کنید (شکل ۱). حال تمام یال‌های قرمز در $\mathbf{MST}_2$ را به گراف منقبض شده اضافه کنید. چون این تعداد، از تعداد یال‌های قرمز در $\mathbf{MST}_1$ کم‌تر است، پس گراف ایجاد شده ناهمبند خواهد بود و پس یال قرمزی از $\mathbf{MST}_1$ وجود دارد که دو سر آن بین دو مولفه همبندی آن باشند. این یال قرمز همان یال مد نظر ما است. □



شکل ۱: انقباض یال e

حال با توجه به دو لم بیان شده، کافیست در ابتدا یک درخت پوشای کمینه دلخواه را در گراف بدون توجه به رنگ یال‌ها با استفاده از الگوریتم‌هایی که خوانده‌ایم، پیدا کنیم. اگر تعداد یال‌های قرمز و آبی در درخت ما باهم برابر باشند الگوریتم به پایان رسیده، وگرنه فرض کنید به تقارن تعداد یال‌های قرمز از آبی بیشتر باشد. آنگاه می‌دانیم یال قرمزی وجود دارد که وقتی آن‌را حذف کنیم در برش بدست آمده در گراف اصلی، یالی آبی وجود داشته باشد که وزن آن با یال آبی حذف شده برابر باشد. پس با حذف یال قرمز و اضافه کردن یال آبی، وزن درخت تغییری نمی‌کند و اختلاف تعداد یال‌های قرمز و آبی یک واحد کم‌تر می‌شود. پس اگر اختلاف تعداد یال‌های قرمزو آبی k باشد، الگوریتم بعد از k مرحله به پایان می‌رسد و در پایان الگوریتم درخت بدست آمده هم پوشای کمینه است و هم تعداد یال‌های قرمز آن با یال‌های آبی آن با هم برابراند.

□

سوال: زمان اجرای این الگوریتم را بیابید. آیا این زمان اجرا چندجمله‌ای است؟

مسأله دو: دیدیم یکی از روش‌های اجرای الگوریتم فورد فارکرسون [۳] این است که در هربار پیدا کردن مسیر افزایشی [۴] ، مسیری را پیدا کنیم که کم‌وزن‌ترین یال آن بیشینه باشد. نحوه پیاده‌سازی این الگوریتم را توضیح دهید.

حل: کافیست به نحوه اجرای الگوریتم دایکسترا در حالت کلی‌تر نگاه کنیم. ما در هر لحظه برای هر رأس پارامتر $d(v)$ را نگه می‌داریم که یک تخمینی از بیشینه اندازه کم‌وزن‌ترین یال در مسیر از s به v است. کافیست هربار رأسی که برای اضافه کردن به مجموعه خود انتخاب می‌کنیم، مقدار تخمین زده‌شده برای آن کمینه باشد. همچنین فرایند ریلکس کردن را به‌صورت زیر تغییر می‌دهیم:

Relax (u,v)

1 **if** $min\{d(u), w(u,v)) > d(v)\}$

2 $\quad d(v) = min\{d(u), w(u,v)\}$

نحوه اثبات این‌که این الگوریتم به هر رأس بیشینه مقدار کم‌وزن‌ترین یال در مسیری از s نسبت می‌دهد درست مانند اثبات الگوریتم دایکسترا[۵] است. همچنین اثبات اینکه این الگوریتم در نهایت بیشینه شار را خروجی می‌دهد درست مانند اثبات الگوریتم ادموندز_کارپ[۶] است.

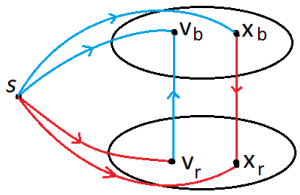
مسأله سه: فرض کنید یک گراف وزن‌دار با یال‌های آبی و قرمز داریم و می‌خواهیم از رأس s به بقیه رأس‌ها، اندازه وزن کوتاه‌ترین مسیر ممکن را بیابیم با این شرط که در این مسیرها یال‌ها باید یک در میان آبی قرمز باشند. الگوریتمی کارا برای حل این مسأله ارائه کنید.

[3] Ford-Fulkerson algorithm
[4] augmenting path
[5] Dijkstra algorithm
[6] Edmonds–Karp algorithm

حل:کافیست از الگوریتم بلمن فورد[۷] استفاده کنیم به این صورت که برای هر رأس دو مقدار $d_r(v)$ و $d_b(v)$ را ذخیره کنیم و در هر بار ریلکس کردن یال‌ها از الگوریتم جدید زیر استفاده کنیم:

```
Relax (u,v)
1  if (u,v) is blue
2        if d_r(u) + w(u,v) < d_b(v)
3              d_b(v) = d_r(u) + w(uv)
4  else if (u,v) is Red
5              if d_b(u) + w(u,v) < d_r(v)
6                    d_r(v) = d_b(u) + w(u,v)
```

حل با دایکسترا: می توانیم گراف G' را از روی گراف اولیه خود به این صورت بسازیم که به ازای هر رأس v در G، دو رأس جدید v_b و v_r را در G' در نظر بگیریم. اگر یال (x,y) در گراف اولیه آبی بود، بین x_b به y_r یال قرار می‌دهیم وگرنه از x_r به y_b یال می‌گذاریم. حال رأس s را به G' اضافه می‌کنیم به این صورت که اگر از s به v یال قرمز بود از s به v_r وگرنه به v_b یال می‌دهیم (شکل ۲). اکنون مشخص است که هر مسیر در گراف جدید بصورت یک در میان آبی قرمز خواهد بود. پس کافی است بعد از اجرای یکی از الگوریتم‌های کوتاه‌ترین مسیر برای هر رأس v، کمینه $d(s,v_b)$ و $d(s,v_r)$ را به‌عنوان کوتاه‌ترین مسیر با شرایط خواسته‌شده نسبت دهیم.



شکل ۲: G'

مسأله چهار: فرض کنید f یک جریان باشد. ثابت کنید همواره می‌توان f را بصورت

$$f = \sum_{p \in P \cup C} \alpha_p x_p \quad \alpha_p > 0$$

بیان کرد بطوری‌که عبارت بالا حداکثر E جمله داشته باشد (P مجموعه کل $s-t$ مسیرهای گراف و C مجموعه تمام دورهای گراف است).

اثبات. اگر گراف m یال داشته باشد و به یال‌ها یک ترتیب دل‌خواه یک تا m دهیم، هر جریان را می‌توان بصورت یک بردار m مولفه‌ای بیان کرد که مولفه iام آن بیانگر جریان یال iام باشد. هدف ما این است که این بردار را بصورت ترکیب خطی مثبت بردار مشخصه‌های $s-t$ مسیرها و دورهای گراف بنویسیم. باقی اثبات به کمک استقرا روی تعداد یال‌هایی که جریان عبوری از آن‌ها غیر صفر است می‌باشد که بعنوان تمرین ماند. □

[7] Bellman-Ford algorithm

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس حل تمرین: مرتضی علیمی
[بهار ۹۹]

جلسه حل تمرین ۷: تحویل چندجمله‌ای و جریان بیشینه نگارنده: محبوبه عنایتی

این جلسه، ابتدا بر مفهوم تحویل چندجمله‌ای [۱] مرور کرده و در ادامه با تکیه بر این موضوع، به چند مسئله در رابطه با مباحث مجموعه مستقل[۲]، پوشش رأسی[۳]، درخت فراگیر کمینه[۴]، اشتراک ماتروییدها[۵]، کوتاه‌ترین مسیر[۶]، برش کمینه [۷] و در آخر نیز به اثبات قضیه‌ای مرتبط با جریان بیشینه[۸] خواهیم پرداخت.

همان‌طور که در جلسه ۱۷ درباره تحویل چندجمله‌ای صحبت کردیم، برای این که بتوان مسئله‌ای (مثلاً A) را با استفاده از مسئله دیگری (مثلاً B) حل نمود، به تابعی نیاز است که با زمان اجرای چندجمله‌ای، ورودی‌های بالقوه A را به ورودی‌های بالقوه B ببرد:

$$x \in A \iff f(x) \in B$$

یعنی تابع f با زمان اجرای چندجمله‌ای امکان حل مسئله A با استفاده از مسئله B را فراهم کند؛ مانند تحویل مسئله تطابق دو بخشی به جریان بیشینه. اصولاً مسائلی که در پیچیدگی محاسباتی با آن‌ها سر و کار داریم، از مسائل تصمیم‌گیری («بله» یا «خیر») هستند. اکثر مسائلی که حالت جست‌وجو یا بهینه‌سازی را دارند، قابل تبدیل به مسائل تصمیم‌گیری‌اند.

۱. A: مسئله مجموعه مستقل.

ورودی. گراف $G = (V, E)$ و $k \in \mathbb{N}$.

خروجی. «بله» یا «خیر» (آیا G مجموعه مستقلی با سایز حداکثر k دارد؟).

B: مسئله پوشش رأسی.

ورودی. گراف $H = (V, E)$ و $k' \in \mathbb{N}$.

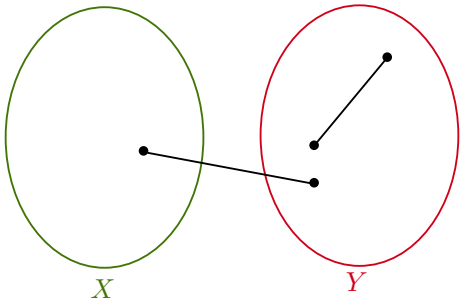
خروجی. «بله» یا «خیر» (آیا H پوشش رأسی با سایز حداکثر k' دارد؟ منظور از پوشش رأسی، انتخاب یک سری رأس از گراف است به گونه‌ای که برای یکی از دو رأس سر هر یال انتخاب شده باشد).

آیا تبدیل $A \leq_{\mathrm{P}} B$ امکان‌پذیر است؟

حل:

مکمل مجموعه رئوس مستقل یک گراف، مجموعه تشکیل‌دهنده پوشش رأسی است. در شکل صفحه بعد فرض کنید X شامل رئوس مستقل و Y مکمل آن باشد. واضح است که هر یالی از G حداقل یک سر در Y دارد؛ چون هر دو رأس سر یک یال نمی‌توانند در X باشد (واضح است که در غیر این صورت X مجموعه مستقلی نخواهد بود).

[1] polynomial-time reduction
[2] IndependentSet problem
[3] VertexCover problem
[4] MinimumSpanningTree (MST) problem
[5] Matroid Intersection
[6] ShortextPath problem
[7] MinimumCut problem
[8] MaximumFlow problem



حال تابع f را این‌طور می‌توان تعریف کرد که ورودی A را گرفته (G,k) و ورودی B $(H,k^{'})$ را خروجی دهد. H همان G و $k^{'}$ برابر با $|V|-k$ خواهد بود:

$$f((G,k))=(H,k^{'})$$

بنابراین G مجموعه مستقلی با سایز k خواهد داشت اگر $H=G$ پوشش رأسی با سایز $k^{'}=|V|-k$ داشته باشد.

به بیان کلی‌تر، G مجموعه مستقلی با سایز حداقل k دارد اگر و تنها اگر H پوشش رأسی با سایز حداکثر $k^{'}$ داشته باشد.

۲. به تبدیل چندجمله‌ای مسئله مجموعه مستقل به مسئله خوشه[۹] (تعدادی رأس که دو به دو به هم یال دارند) فکر کنید.[۱۰]

۳. A: گرافی با یال‌های آبی و قرمز و با وزن‌دهی مثبت داده شده است. آیا می‌توان زیردرخت فراگیر کمینه‌ای از آن را پیدا کرد که نصف یال‌هایش آبی و نصف دیگر قرمز باشد؟

B: مسئله پیدا کردن مجموعه مستقل با وزن بیشینه در یک ماتروید (قابل حل با الگوریتم حریصانه[۱۱]).

ورودی. $M_1=(S,I_1)$، $M_2=(S,I_2)$ و $w:S\to\mathbb{R}$.

خروجی. مجموعه مستقل مشترک M_1 و M_2 با وزن بیشینه.

برای حل مسئله A، می‌توانیم از ایده اشتراک ماتروئیدها استفاده کنیم؛ یعنی با توجه به ورودی و هدف مسئله، دو ماتروید را به گونه‌ای بسازیم که الگوریتمِ چندجمله‌ای مسئله B (در اینجا به شرح آن نخواهیم پرداخت) مجموعه مستقل مشترک آن دو را بیابد، طوری که وزنش بیشینه باشد. (دقت کنید که بیشینه یا کمینه تفاوتی ایجاد نمی‌کند. یعنی اگر هدف مسئله A یافتن زیر درخت فراگیر کمینه با شروط اشاره‌شده باشد، می‌توان بدون تغییر در کلیت مسئله، تابع وزن جدیدی برای آن تعریف کرد که به هر یال وزنی قرینه وزن اولیه نسبت دهد. در این صورت هدف مسئله یافتن زیر درخت فراگیر بیشینه خواهد بود که با «وزن بیشینه» اشاره‌شده در مسئله B سازگار است).

(یادآوری) $M=(S,I)$ یک ماتروید است اگر:

$$I\subseteq 2^S, \emptyset\in I$$

$$\forall A\in I \quad \forall B\subseteq A \quad B\in I \ \text{I}$$

$$\forall A,B\in I \quad |A|>|B|\to\exists x\in A\setminus B \quad x+B\in I \ \text{II}$$

حل:

[9] Clique problem

[۱۰] به بیان دقیق‌تر، گراف G دارای مجموعه مستقلی با سایز k باشد اگر و تنها اگر گراف $G^{'}$ خوشه‌ای (زیر گرافی کامل) با سایز $k^{'}(=?)$ داشته باشد.

[11] Greedy Algorithm

اگر S را مجموعه یال‌های گراف در نظر بگیریم آنگاه تعریف I_1 و I_2 به این شرح خواهد بود: I_1 را شامل مجموعه‌ای از یال‌ها که فاقد دور هستند و جنگل‌های گراف را تشکیل می‌دهند[۱۲]، در نظر می‌گیریم و I_2 را نیز شامل مجموعه‌هایی از یال‌ها که تعداد یال‌های قرمز در هر یک حداکثر $\frac{n-1}{2}$ و تعداد یال‌های آبی نیز حداکثر $\frac{n-1}{2}$ است (تعداد رئوس گراف $= n$).

قبلاً دیدیم که خواص ماتروید برای M_1 برقرار است. اکنون به بررسی این خواص در M_2 می‌پردازیم. بدیهی است که M_2 دو خاصیت اول را دارد. فرض کنیم که دو مجموعه A و B عضوی از I_2 باشند به طوری که $|A| > |B|$. بنابراین یا تعداد یال‌های قرمز A از یال‌های قرمز B بیشتر است و یا تعداد یال‌های آبی آن از یال‌های آبی B بیشتر است. در حالت اول کافیست یکی از یال‌های قرمز A به B اضافه شود و در حالت دوم یکی از یال‌های آبی آن. پس M_2 خاصیت II را دارد و ماتروید است.

فرض کنید الگوریتمی جادویی داریم که با گرفتن M_1 و M_2 و اشتراک‌گیری آن‌ها مجموعه X را به عنوان مجموعه مستقل با وزن بیشینه و دارای تعداد یال‌های قرمز و آبیِ برابر تولید می‌کند. از قبل وزن زیردرخت فراگیر بیشینه جدا حساب شده است. برای دادن جواب نهایی، کافیست این وزن با وزن X مقایسه شود؛ اگر برابر باشند جواب مسئله A «بله» خواهد بود و در غیر این صورت «خیر». دقت کنید که این‌جا X همان درختی است که وزنش با ارضاء شرط برابری تعداد یال‌های قرمز و آبی، بیشینه است و با مجموعه‌ای مقایسه خواهد شد که روی رنگ یال‌هایش شرطی گذاشته نشده.

این نکته نیز حائز اهمیت است که این‌گونه اشتراک‌گیری ماترویدها برای تعمیم این مسئله هم پاسخ‌گو است. برای مثال می‌توانید زیردرخت فراگیر بیشینه‌ای را بیابید که تعداد یال‌های آبی آن k و تعداد یال‌های قرمز آن $(n-1)-k$ باشد.

۴. یافتن کم‌یال‌ترین کوتاه‌ترین مسیر.

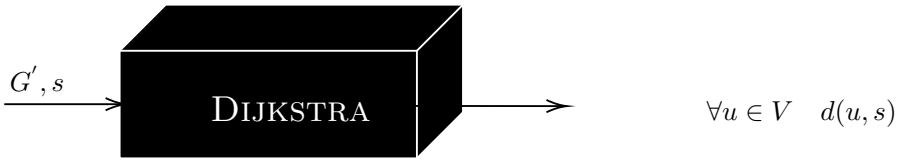
ورودی. $G = (V, E)$، $s \in V$ و $w : E \to \mathbb{R}^+$.

خروجی. $d(u)$ ($u \in V$، تعداد یال‌های کم‌یال‌ترین کوتاه‌ترین مسیر از s به u $(d(u) = u$).

حل:

ایده حل استفاده از الگوریتم دایکسترا[۱۳] است. آن را جعبه سیاهی در نظر بگیریم که اگر گراف و رأس شروع را به عنوان ورودی بگیرد، کوتاه‌ترین مسیر را با ویژگی خواسته‌شده در مسئله خروجی دهد. پس باید در گراف ورودی تغییراتی ایجاد کنیم و سپس آن را به جعبه سیاه دهیم.

اگر گراف جدید همان گراف ورودی باشد و وزن یال‌ها این گونه تغییر کند که مقدار خیلی کوچکی به وزن هر یال اضافه شود، مثلاً $\varepsilon < \frac{1}{n}$، کوتاه‌ترین مسیری که الگوریتم دایکسترا در این گراف جدید پیدا می‌کند معادل با کوتاه‌ترین مسیر در گراف اصلی خواهد بود.



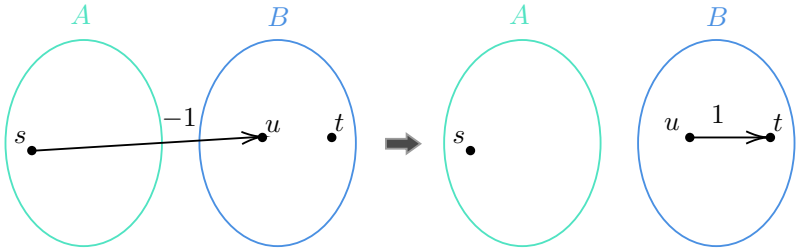
۵. شبکه‌ای را فرض کنید که تنها یال‌های با وزن منفی آن می‌تواند یال‌های خروجی رأس s و یال‌های ورودی رأس t باشد. وزن بقیه یال‌ها در آن نامنفی است. الگوریتمی برای یافتن برش کمینه در این شبکه ارائه کنید.

حل:

با ایجاد تغییراتی در شبکه ورودی، شبکه جدیدی خواهیم ساخت و برش کمینه را با الگوریتم‌های قبلی می‌یابیم. فرض کنید u رأسی در شبکه اصلی باشد به طوری که وزن یال (s, u) منفی است $(w((s, u)) < 0)$. شبکه جدیدْ مشابه شبکه اصلی خواهد بود با این تفاوت که یالی مانند (s, u) در آن حذف شده و یال (u, t) با وزن نامنفی $-w((s, u))$ به آن اضافه شده است (یال‌های بین s و t

[12] graphic matroid
[13] DIJKSTRA

را نیز می‌توان حذف کرد چون به هر حال در هر برش وجود دارند). ثابت کنید برش کمینه در شبکه جدید معادل با برش کمینه در شبکه اصلی است.



دقت کنید اگر در شبکه ورودی، امکان منفی بودن وزن تمامی یال‌ها وجود داشت، آنگاه با مسئله‌ای ان‌پی‌ـتمام[14] روبه‌رو می‌بودیم.

تمرین. با فرض مثبت بودن وزن‌ها ثابت کنید مسئله برش بیشینه[15] ان‌پی‌ـتمام است.

۶. در هر شبکه نشان دهید گردش با تقاضای d وجود دارد اگر و تنها اگر برای هر برش (A, B) در آن داشته باشیم

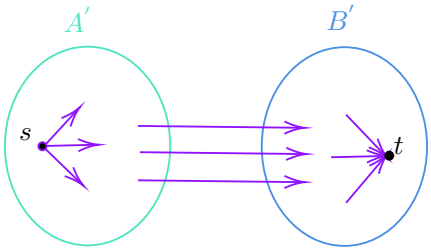
$$c(A, B) \geq \sum_{v \in B} d(v)$$

حل:

درباره شرط لازم، جلسه ۱۴ صحبت کردیم.

برای اثبات شرط کافی، می‌توانید از قضیه گردش هافمن استفاده کنید. اثبات پیش رو، با استفاده از قضیه «جریان بیشینه‌ـبرش کمینه»[16] است:

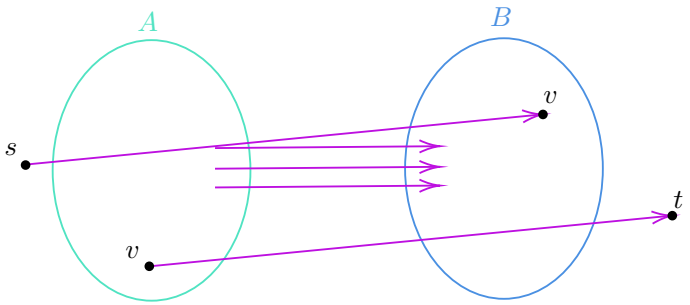
همان‌طور که به یاد دارید، برای اثبات وجود گردش در شبکه‌هایی با عرضه و تقاضا، گراف اصلی را به گرافی با دو رأس جدید s و t تبدیل می‌کردیم. در گراف جدید از s به هر رأس v که $d(v) < 0$ یالی با ظرفیت $-d(v)$ وصل کرده و همچنین از هر رأس u که $d(u) > 0$، نیز یالی با ظرفیت $d(u)$ به t متصل می‌کردیم. بنابراین وجود گردش در گراف اصلی معادل وجود جریان بیشینه برابر با مجموع ظرفیت یال‌های خروجی رأس s است. در این مسئله نیز، چنین تبدیلی لازم است.



اکنون عکس نقیض را در نظر بگیرید: اگر گردشی با تقاضای d در شبکه وجود نداشته باشد آنگاه برشی مانند (A, B) وجود دارد که $c(A, B) < \sum_{v \in B} d(v)$.

معادل فرض این گزاره را برای شبکه حاصل از تبدیل شبکه اصلی در نظر بگیرید: اگر در شبکه اصلی گردشی وجود نداشته باشد، آنگاه جریان بیشینه در شبکه، کوچک‌تر از ظرفیت یال‌های خروجی s است ($c(A', B') < c(\delta(s)) = \sum_{e=(s,u)} c(e)$) که (A', B') برش معادل برش اولیه در شبکه اصلی یعنی (A, B) است). می‌دانیم ظرفیت برش کمینه برابر جریان بیشینه است. بنابراین در شبکه اصلی نیز برش (A, B) وجود خواهد داشت که $c(A, B) < \sum_{v \in B} d(v)$.

[14] **NP-complete**
[15] MaximumFlow problem
[16] Max-flow Min-cut Theorem



$$c(A^{'}, B^{'}) = c(A, B) + \sum_{v \in B, d(v) < 0} - d(v) + \sum_{v \in A, d(v) > 0} d(v) < \sum_{v \in V, d(v) < 0} - d(v)$$

$$\implies c(A, B) + \sum_{v \in B, d(v) < 0} - d(v) + \sum_{v \in A, d(v) > 0} d(v) < \sum_{v \in A, d(v) < 0} - d(v) + \sum_{v \in B, d(v) < 0} - d(v)$$

$$\implies c(A, B) < \sum_{v \in A, d(v) < 0} - d(v) = \sum_{v \in B} d(v)$$

برقراری تساوی آخر به دلیل برابری تقاضاها و عرضه‌هاست.

□

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس حل تمرین: مرتضی علیمی

[بهار ۹۹]

جلسه حل تمرین ۸: مسائل انپی–تمام[1] نگارنده: امیرحسین افشارراد

در این جلسه به بررسی بیشتر مسائل انپی–تمام و همچنین حلّ چند مثال مربوط تحویل چندجمله‌ای[2] مسائل به یکدیگر پرداخته خواهد شد.

۱ حلّ نسخه‌های دیگر مسأله‌ی زیرمجموعه‌ی مستقل در گراف با استفاده از نسخه‌ی تصمیم‌گیری

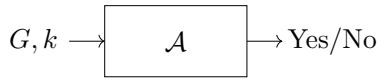
می‌دانید در بررسی پیچیدگی محاسباتی مسائل، همواره فرض می‌کنیم که پاسخ مسأله به صورت «بله یا خیر» تعیین می‌شود (مسأله‌های تصمیم‌گیری). این رویکرد ممکن است ابتدا یک رویکرد ضعیف به نظر برسد؛ چرا که در بسیاری از مسائل عملی نیاز به پاسخی فراتر از یک بله یا خیر وجود دارد. در این بخش و در قالب یک مثال، به بررسی این موضوع خواهیم پرداخت که با فرض وجود الگوریتمی برای پاسخ به نسخه‌ی تصمیم‌گیری (بله یا خیر) یک مسأله، چگونه می‌توان الگوریتم‌هایی برای حلّ نسخه‌های دیگر آن ارائه کرد. مسأله‌ی مورد نظر، مسأله‌ی زیرمجموعه‌ی مستقل در یک گراف است. در ابتدا تعریف زیرمجموعه‌ی مستقل را مرور می‌کنیم:

تعریف ۱. *اگر $G = (V, E)$ یک گراف باشد، $U \subseteq V$ را یک زیرمجموعه‌ی مستقل از رئوس گراف G گوییم، هرگاه بین هیچ دو رأس از U یالی از گراف G وجود نداشته باشد.*

همچنین می‌توان نسخه‌ی تصمیم‌گیری مسأله‌ی زیرمجموعه‌ی مستقل را به صورت زیر تعریف کرد:

ورودی. گراف G و عدد k.
خروجی. آیا G زیرمجموعه‌ی مستقل با اندازه‌ی حدّاقل k دارد؟

همان طور که پیش‌تر بیان شد، در این جا فرض خواهیم کرد که مسأله‌ی فوق حل شده است و به کمک آن، نسخه‌های دیگری از مسأله را حل می‌کنیم. برای این منظور فرض کنید الگوریتمی مانند $\mathcal{A}$ برای حلّ این مسأله وجود دارد؛ و به صورت یک جعبه‌ی سیاه (که دسترسی به داخل آن امکان‌پذیر نیست) داده شده است. شکل ۱ به صورت نمادین چنین ساختاری را توصیف می‌کند:



شکل ۱: الگوریتم حلّ نسخه‌ی تصمیم‌گیری مسأله‌ی زیرمجموعه‌ی مستقل گراف

در ادامه دو نسخه‌ی دیگر از این مسأله را بررسی خواهیم کرد.

[1] NP–Complete
[2] polynomial-time reduction

۱.۱ مسأله‌ی یافتن اندازه‌ی بزرگ‌ترین زیرمجموعه‌ی مستقل در گراف

می‌خواهیم *اندازه‌ی* بزرگ‌ترین زیرمجموعه‌ی مستقل گراف داده‌شده‌ی G را بیابیم. برای این منظور، کافی است از الگوریتم $\mathcal{A}$ استفاده کنیم و همه‌ی مقادیر ممکن برای k را از کوچک به بزرگ آزمایش کنیم تا به اوّلین مقداری از k مثل $k^* + 1$ برسیم که به ازای آن، خروجی الگوریتم No باشد. در این صورت به وضوح پاسخ مورد نظر k^* است.

همچنین می‌توان زمان اجرای چنین الگوریتمی را نیز کمی بهبود داد؛ و به جای آزمودن تمامی حالات k تا رسیدن به مقدار k^*، از جست‌وجوی دودویی استفاده کرد. برای این منظور بین دو کران پایین و بالای $k = 1$ و $k = |V|$ جست‌وجوی دودویی انجام می‌دهیم تا اوّلین مقداری از k مثل $k^* + 1$ را بیابیم که به ازای آن، خروجی الگوریتم No است. به این ترتیب k^* پاسخ مسأله خواهد بود. واضح است که در این حالت برای هر k که $k \leq k^*$ پاسخ مسأله برابر با Yes، و برای هر k که $k > k^*$ پاسخ مسأله No خواهد بود. بنابراین خروجی الگوریتم توصیف‌شده، k^*، همان اندازه‌ی بزرگ‌ترین زیرمجموعه‌ی مستقلّ گراف G خواهد بود.

۲.۱ مسأله‌ی یافتن (یک) بزرگ‌ترین زیرمجموعه‌ی مستقل در گراف

حال فرض کنید یک گام فراتر برویم و بخواهیم به جای اندازه‌ی بزرگ‌ترین زیرمجموعه‌ی مستقل، خود آن زیرمجموعه را به صورت مجموعه‌ای از رئوس گراف بیابیم. در واقع ممکن است پاسخ این مسأله یکتا نباشد. در این جا به دنبال یافتن یک زیرمجموعه‌ی مستقل از گراف داده‌شده‌ی G هستیم. (فرض کنید الگوریتم معرّفی‌شده در قسمت ۱.۱ را مورد استفاده قرار داده‌ایم و می‌دانیم اندازه‌ی بزرگ‌ترین زیرمجموعه‌ی مستقل گراف برابر با k^* است.)

برای این منظور با پیمایش بر روی رئوس گراف و استفاده از الگوریتم $\mathcal{A}$ یک زیرمجموعه‌ی مستقل با بزرگ‌ترین اندازه را می‌سازیم. فرض کنید این زیرمجموعه را با S نشان دهیم. در ابتدا S را مجموعه‌ی تهی در نظر می‌گیریم و با یک ترتیب دلخواه شروع به پیمایش رئوس گراف می‌کنیم. فرض کنید در شروع این فرآیند، رأس $v_1 \in V$ را انتخاب کرده‌ایم. ابتدا فرض می‌کنیم این رأس از گراف حذف شده باشد؛ یعنی گراف $G - \{v_1\}$ را در نظر می‌گیریم. (دقّت کنید که نمادگذاری $G - \{v_1\}$ نادقیق است؛ امّا قرارداد می‌کنیم که منظور از آن، گرافی است که از حذف رأس v_1 و یال‌های متصّل به آن از گراف G حاصل شود.) حال حاصل $\mathcal{A}(G - \{v_1\}, k^*)$ را محاسبه می‌کنیم (یعنی $G - \{v_1\}$ و k^* را به عنوان ورودی به الگوریتم $\mathcal{A}$ می‌دهیم). اگر خروجی Yes باشد، یعنی می‌توان v_1 را از G حذف کرد و همچنان یک زیرمجموعه‌ی مستقل k^*عضوی به دست آورد. در این حالت v_1 را از G حذف می‌کنیم و الگوریتم را بر روی گراف باقی‌مانده ادامه می‌دهیم. اگر خروجی No باشد، یعنی با حذف v_1 دیگر هیچ زیرمجموعه‌ی مستقل k^*عضوی در گراف باقی‌مانده وجود نخواهد داشت؛ و این یعنی باید v_1 را به S اضافه کنیم. در این حالت برای ادامه‌ی الگوریتم، باید هم v_1 و هم تمامی همسایگان آن را از گراف G حذف کنیم (زیرا طبق تعریف زیرمجموعه‌ی مستقل هیچ یک از همسایگان آن نمی‌توانند در S حاضر شوند) و الگوریتم را برای یافتن یک زیرمجموعه‌ی مستقل با $k^* - 1$ عضو از گراف $G - \{v_1\} \cup \mathcal{N}(v_1)$ ادامه دهیم که منظور از $\mathcal{N}(v_1)$ مجموعه‌ی همسایگان v_1 است. کافی است همین روال را ادامه دهیم تا S به یک مجموعه‌ی k^*عضوی تبدیل شود. این فرآیند را می‌توان به صورت شبه‌کد زیر نیز پیاده‌سازی کرد:

```
FindIndependentSet(G, k)
1  if k == 0
2      return ∅
3  choose v ∈ G
4  if A(G − {v}, k) == Yes
5      return FindIndependentSet(G − {v}, k)
6  return {v} ∪ FindIndependentSet(G − {v} ∪ N(v), k − 1)
```

نکته: در بسیاری از مسائل دیگر نیز می‌توان با رویکردی مشابه، با استفاده از نسخه‌ی تصمیم‌گیری مسأله به حلّی برای نسخه‌ی عمومی‌تر آن دست یافت. به عنوان مثال، در مسأله‌ی ۳–صدق‌پذیری (3–SAT) (با فرض آن که یک الگوریتم برای نسخه‌ی تصمیم‌گیری مسأله موجود است و به واسطه‌ی آن، از وجود جواب معتبر اطمینان داریم) می‌توان برای یافتن مقدار مناسب برای هر یک از متغیّرهای مسأله مثل x، ابتدا آن را صحیح در نظر گرفت و مسأله را به یک مسأله‌ی 3–SAT کوچک‌تر تبدیل کرد. اگر مسأله‌ی 3–SAT جدید پاسخ معتبر داشته باشد؛ انتخابی که برای متغیّر x انجام داده بودیم مناسب بوده و کافی است برای سایر متغیّرها انتخاب‌های مناسب انجام دهیم. اگر مسأله‌ی جدید حلّ معتبر نداشته باشد، باید متغیّر x را غلط در نظر بگیریم و در ادامه، برای سایر متغیّرها در قالب یک مسأله‌ی 3–SAT کوچک‌تر به دنبال پاسخ درست بگردیم.

ساختاری که در دو مثال اخیر مشاهده شد را **خودکاهش‌پذیری**[۳] گوییم. در چنین ساختاری، با انجام یک انتخاب در مسأله، آن را به مسأله‌ی کوچک‌تری از همان جنس تبدیل می‌کنیم و فرآیند حل را ادامه می‌دهیم.

[3] self-reducibility

۲ مسأله‌ی رنگ‌پذیری گراف

در این قسمت به بررسی مسأله‌ی رنگ‌پذیری گراف[۴] خواهیم پرداخت. در ابتدا یک رویکرد کلّی برای تحویل چندجمله‌ای دو مسأله‌ی رنگ‌پذیری را بررسی می‌نماییم؛ و در ادامه توجّه خود را به مسائل ۲–رنگ‌پذیری و ۳–رنگ‌پذیری گراف معطوف خواهیم کرد. پیش از بررسی این مسائل، تعریف رنگ‌پذیری را مرور می‌کنیم:

تعریف ۲. *گراف $G = (V, E)$ را k–رنگ‌پذیر[۵] گویند، هرگاه بتوان رئوس G را با k رنگ متمایز رنگ‌آمیزی کرد؛ به گونه‌ای که برای هر یال گراف، رنگ رئوس ابتدا و انتهای آن متفاوت باشد.*

۱.۲ تحویل چندجمله‌ای مسائل k–رنگ‌پذیری

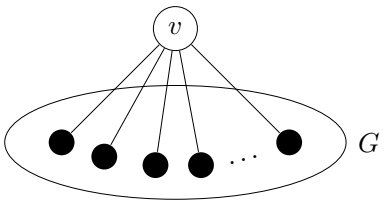
ابتدا مسأله‌ی k–رنگ‌پذیری را به صورت زیر تعریف می‌کنیم:

ورودی. گراف G و عدد k.
خروجی. آیا گراف G، k–رنگ‌پذیر است؟

اگر مسأله‌ی k–رنگ‌پذیری را با نماد k–COL نشان دهیم، در این قسمت ثابت خواهیم کرد که می‌توان مسأله‌ی k–رنگ‌پذیری را به مسأله‌ی $(k+1)$–رنگ‌پذیری کاهش داد:

$$k\text{–COL} \leq_p (k+1)\text{–COL} \tag{۱}$$

برای این منظور کافی است گراف G را در نظر گرفته و مطابق با شکل ۲ یک رأس جدید مانند v به آن اضافه کرده و v را به تمامی رئوس گراف G متصّل کنیم. اگر این گراف جدید را G' بنامیم، ادّعا می‌کنیم که حلّ مسأله‌ی $(k+1)$–رنگ‌پذیری در گراف G' معادل است با حلّ مسأله‌ی k–رنگ‌پذیری در گراف G.



شکل ۲: فرآیند کاهش مسأله‌ی k–رنگ‌پذیری به مسأله‌ی $(k+1)$–رنگ‌پذیری

اثبات هر دو طرف این ادّعا ساده است. با توجّه به این که v به تمام رئوس گراف G متصّل شده است، در یک $(k+1)$–رنگ‌آمیزی گراف G' هیچ رأسی نمی‌تواند با v هم‌رنگ باشد، بنابراین سایر رئوس با حدّاکثر k رنگ متمایز رنگ‌آمیزی شده‌اند که اگر رنگ‌آمیزی اوّلیه یک $(k+1)$–رنگ‌آمیزی معتبر برای G' بوده باشد، آنگاه با در نظر گرفتن رئوس G در G' یک k–رنگ‌آمیزی معتبر برای G خواهیم داشت. همچنین اگر یک k–رنگ‌آمیزی معتبر برای G در دست باشد، می‌توانیم رنگ v را رنگی متمایز با k رنگ اوّلیه قرار دهیم و به یک $(k+1)$–رنگ‌آمیزی معتبر برای G' برسیم.

به این ترتیب توانسته‌ایم مسأله‌ی k–رنگ‌پذیری را به مسأله‌ی $(k+1)$–رنگ‌پذیری کاهش دهیم و با توجّه به خاصیت تراگذری در تحویل چندجمله‌ای مسائل، برای هر $m \geq k$ خواهیم داشت:

$$k\text{–COL} \leq_p m\text{–COL} \tag{۲}$$

[4] graph colorability
[5] k–colorable

۲.۲ مسأله‌ی ۲–رنگ‌پذیری

در بخش بعدی نشان خواهیم داد که مسأله‌ی ۳–رنگ‌پذیری یک مسأله‌ی انپی–تمام است و در نتیجه، با توجّه به نتیجه‌ی ۲، نتیجه خواهد شد که برای هر $k \geq 3$ مسأله‌ی k–رنگ‌پذیری مسأله‌ای انپی–تمام است. در این میان تنها نقطه‌ی ابهام باقی‌مانده به ازای $k = 2$ خواهد بود. در این بخش نشان می‌دهیم که مسأله‌ی ۲–رنگ‌پذیری را می‌توان با یک الگوریتم چندجمله‌ای حل کرد.

اثبات این موضوع ساده است؛ کافی است با شروع از یک رأس دلخواه مانند s از گراف داده‌شده یک بار الگوریتم BFS را اجرا کنیم و تا عمق هر رأس گراف در درخت BFS مشخّص شود. حال همه‌ی رئوسی که عمق آن‌ها در این درخت زوج است را قرمز و تمامی رئوسی که عمق آن‌ها در این درخت فرد است را آبی رنگ‌آمیزی می‌کنیم. (واضح است که اگر چنین نکنیم، حتماً یالی وجود خواهد داشت که دو سر آن هم‌رنگ باشد.) حال کافی است یک بار برای تمامی یال‌های گراف (یا به طور دقیق‌تر، برای تمامی یال‌هایی از گراف که در درخت BFS حاضر نیستند) شرط رنگ‌آمیزی صحیح را بررسی کنیم. اگر هیچ یالی با دو سر هم‌رنگ وجود نداشت پاسخ مسأله مثبت است و یک ۲–رنگ‌آمیزی مناسب برای گراف وجود دارد. در صورتی که یالی از گراف باشد که دو سر آن هم‌رنگ باشد قطعاً هیچ ۲–رنگ‌آمیزی معتبری برای گراف وجود ندارد. البته دقّت کنید که شاید طرف دوم این گزاره در نگاه اوّل واضح به نظر نرسد؛ یعنی ممکن است در نگاه اوّل چنین به ذهن آید که ممکن است رنگ‌آمیزی دیگری موجود باشد که شرایط مسأله را برآورده کند؛ امّا کافی است درخت BFS محاسبه‌شده در یک بار اجرای الگوریتم را در نظر بگیریم. برای این درخت، تنها یک روش رنگ‌آمیزی یکتا وجود دارد (به طور دقیق‌تر دو روش، که یکی دقیقاً برعکس دیگری است. پس می‌توان با ثابت در نظر گرفتن رنگ رأس s، رنگ‌آمیزی صحیح این درخت را یکتا در نظر گرفت). بنابراین با توجّه به یکتایی رنگ‌آمیزی صحیح این درخت و با در نظر گرفتن این که این درخت (با فرض همبندی گراف) یک زیردرخت فراگیر از گراف اصلی است؛ به وضوح نتیجه می‌شود که اگر این رنگ‌آمیزی با شرط مسأله در تعارض باشد، هیچ رنگ‌آمیزی معتبر دیگری نیز برای این گراف موجود نخواهد بود. (توجّه کنید که اگر گراف ناهمبند باشد باید الگوریتم فوق را در هر مؤلّفه‌ی همبندی به صورت جداگانه انجام دهیم.)

الگوریتم معرّفی‌شده به وضوح یک الگوریتم چندجمله‌ای است و در زمان $O(|V| + |E|)$ قابل پیاده‌سازی است.

۳.۲ مسأله‌ی ۳–رنگ‌پذیری

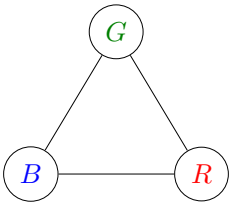
در این بخش نشان خواهیم داد که مسأله‌ی ۳–رنگ‌پذیری گراف یک مسأله‌ی انپی–تمام است. برای این منظور، مسأله‌ی 3–SAT را به مسأله‌ی ۳–رنگ‌پذیری کاهش خواهیم داد:

$$3\text{SAT} \leq_p 3\text{–COL} \tag{۳}$$

به این ترتیب باید متناظر با یک ورودی استاندارد مسأله‌ی 3–SAT گرافی بسازیم که شرط لازم و کافی برای وجود یک ۳–رنگ‌آمیزی معتبر در آن گراف، وجود مقداردهی مناسب برای متغیّرهای مسأله‌ی 3–SAT باشد.

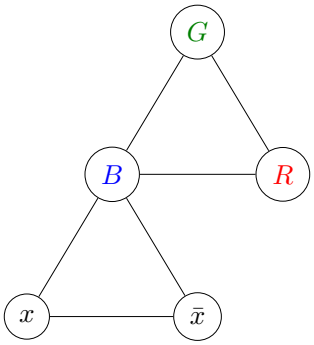
ابتدا دقّت می‌کنیم که مقداردهی متغیّرها دوحالته است؛ و این در حالی است که رنگ‌آمیزی هر رأس سه حالت دارد. با توجّه به این که ابتدایی‌ترین ایده در ساخت گراف متناظر آن است که برای متغیّرها رأس‌هایی در گراف در نظر بگیریم؛ به نظر می‌آید که این تفاوت ساختاری (دوحالته بودن متغیّرها و سه‌حالته بودن وضعیت رنگ‌آمیزی رئوس گراف) مشکل‌ساز است. برای حلّ‌این مشکل، در ادامه سعی خواهیم کرد ساختار گراف را طوری تعیین کنیم که هر متغیّر و نقیض آن در یک رنگ‌آمیزی معتبر حتماً یکی از دو رنگ متناظر با درست یا نادرست را اخذ کنند. در اوّلین گام، یک حلقه‌ی مثلّثی مطابق با شکل ۳ در گراف ایجاد می‌کنیم. در ادامه‌ی این اثبات و برای سادگی بیان، گراف مثلّثی‌شکل نشان‌داده‌شده در شکل ۳ را **مثلّث مرجع** می‌نامیم.

قرارداد می‌کنیم که سه رأس R، G، و B در شکل ۳ نمایانگر سه رنگ قرمز، سبز، و آبی در رنگ‌آمیزی گراف باشند. البته دقّت کنید که ممکن است در ابتدا به نظر برسد که تعیین رنگ رئوس در رنگ‌آمیزی مناسب در اختیار ما نیست و ممکن است در یک رنگ‌آمیزی صحیح،



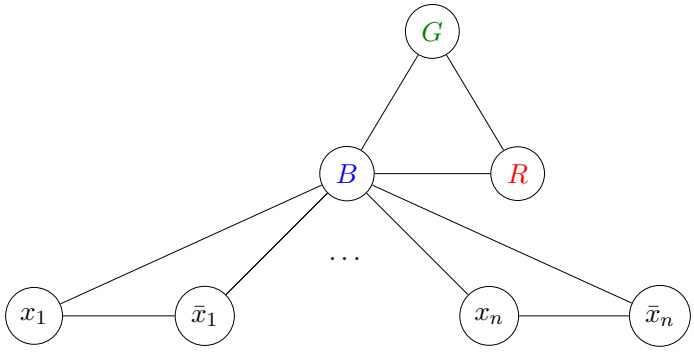
شکل ۳: مثلّث مرجع سه رنگ اصلی در گراف مورد رنگ‌آمیزی

رأس‌ها رنگ‌های دیگری به خود بگیرند؛ امّا دقّت کنید که این کار به معنای تعیین رنگ رئوس نیست. در واقع می‌توان با بیان دیگری قرارداد کرد که رأس با اندیس R هر رنگی که در یک رنگ‌آمیزی معتبر به دست آورد، آن رنگ را «قرمز» می‌نامیم. همچنین در مورد رئوس G و B برای رنگ‌های سبز و آبی. حال رنگ سبز را متناظر با درست بودن یک متغیّر و رنگ قرمز را به معنای غلط بودن آن در نظر می‌گیریم. به این ترتیب اوّلین گامی که برای ساخت گراف مطلوب باید طی کنیم، آن است که هر متغیّر دلخواه مسأله مثل x را مجبور کنیم که یکی از دو رنگ قرمز یا سبز را در رنگ‌آمیزی معتبر به خود بگیرد؛ و به علاوه، اگر رنگ x سبز (قرمز) است، رنگ $\bar{x}$ قرمز (سبز) باشد. برای دستیابی به این هدف، برای هر متغیّر x آن را در ساختاری مانند شکل ۴ قرار می‌دهیم که در آن، x و $\bar{x}$ به رأس B از مثلّث مرجع متصّل هستند.



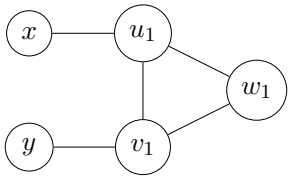
شکل ۴: ساختار پیشنهادی برای تحمیل درست و غلط بودن به رئوس متناظر با متغیّرهای منطقی برای یک متغیّر دلخواه x

به این ترتیب هیچ یک از دو رأس x و $\bar{x}$ نمی‌توانند آبی باشند؛ و به علاوه از آن جا که خود این دو رأس نیز متصّل هستند، قطعاً در یک رنگ‌آمیزی معتبر گراف باید یکی از آن‌ها سبز و دیگری قرمز باشد. به این ترتیب شکل عمومی رئوسی از گراف که متناظر با متغیّرهای منطقی مسأله‌ی 3–SAT هستند به صورت شکل ۵ خواهد بود.



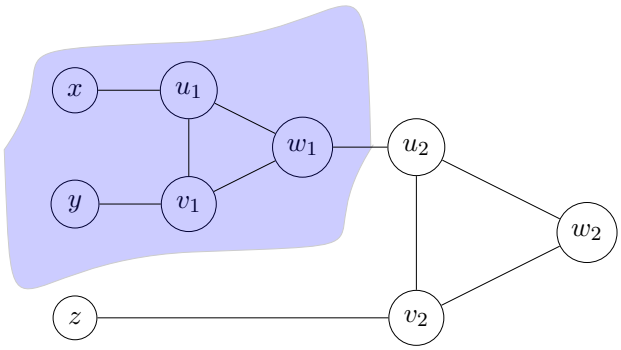
شکل ۵: ساختار پیشنهادی برای تحمیل درست و غلط بودن به رئوس متناظر با متغیّرهای منطقی برای همه‌ی متغیّرها

حال به سراغ چالش اصلی مسأله می‌رویم. فرض کنید $(x \vee y \vee z)$ یکی از عبارات موجود در ورودی مسأله‌ی 3–SAT باشد. می‌خواهیم ساختاری در گراف مربوطه ایجاد کنیم که در صورت وجود ۳–رنگ‌آمیزی معتبر، حدّاقل یکی از این سه متغیّر به رنگ سبز درآید. ابتدا ساختار مقدّماتی شکل ۶ را در نظر بگیرید.



شکل ۶: ساختار نیمه‌تمام پیاده‌سازی عمل منطقی OR در گراف

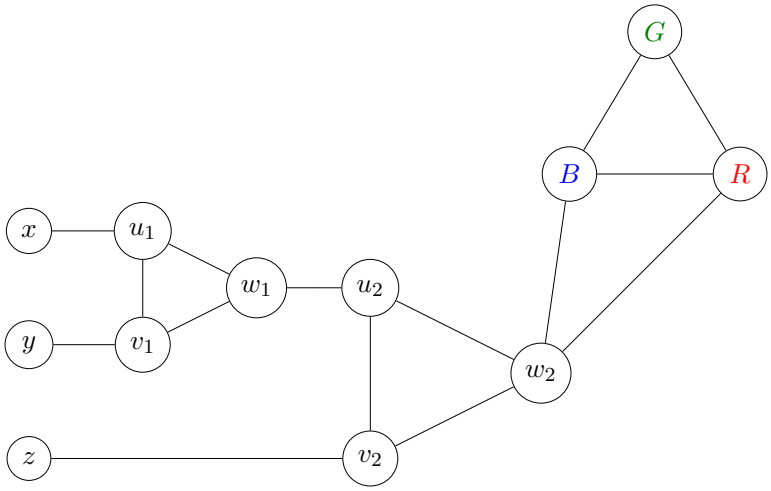
هدف از ساختار شکل ۶ آن است که به نوعی عمل OR منطقی را پیاده‌سازی کنیم. در صورتی که x و y هر دو غلط باشند، u_1 و v_1 دو رنگ متمایز و مخالف قرمز خواهند داشت، بنابراین یکی باید سبز و دیگری آبی باشد، و در نتیجه w_1 حتماً باید قرمز باشد. به این ترتیب در صورتی که هر دو ورودی این ساختار قرمز باشند، خروجی آن نیز قطعاً قرمز خواهد بود و از این جهت، موفّق به مدل‌سازی OR منطقی شده‌ایم. با این حال، در صورتی که یکی از ورودی‌ها سبز باشد؛ الزامی برای سبز بودن خروجی وجود ندارد (البته دقّت کنید که اگر هر دو ورودی سبز باشند، با استدلالی مشابه با آن‌چه بیان شد، قطعاً خروجی سبز خواهد بود؛ امّا به سادگی می‌توانید مشاهده کنید که اگر یکی از دو ورودی سبز و دیگری قرمز باشد، w_1 می‌تواند هر رنگی به خود بگیرد). این مشکل را در این جا رها می‌کنیم و در آینده‌ی نزدیک به حلّ آن می‌پردازیم. در این مرحله، ساختار شکل ۶ را گسترش می‌دهیم. در واقع این ساختار، مدل نیمه‌کاره‌ای برای OR منطقی است. با توجّه به این که عبارت منطقی اصلی مورد نظر، $(x \vee y \vee z)$ است؛ کلّ ساختار شکل ۶ را به عنوان ورودی همان ساختار در نظر می‌گیریم و ورودی دیگر را برابر با z قرار می‌دهیم. حاصل این کار، شکل ۷ خواهد بود.



شکل ۷: ساختار تکمیل‌شده‌ی پیاده‌سازی عمل منطقی OR در گراف

همان‌طور که گفته شد، ساختار شکل ۷ ترکیب دو نمونه از ساختار شکل ۶ است که یکی از ورودی‌های آن، بر روی شکل ۷ به صورت رنگی مشخّص شده است که خود، همان ساختار شکل ۶ است. به این ترتیب می‌خواهیم ساختار شکل ۷ نماینده‌ای از عبارت منطقی $x \vee y \vee z$ باشد. پیش‌تر نشان دادیم که در صورت قرمز بودن ورودی‌های این ساختار، خروجی آن (در این جا w_2) قطعاً قرمز خواهد بود. همچنین اگر هر سه ورودی سبز رنگ باشند، با استدلال مشابه خروجی نیز قطعاً سبز خواهد بود. با این وجود برای حالاتی که ورودی‌ها ترکیبی از رنگ‌های سبز و قرمز باشند، خروجی می‌تواند رنگ‌های مختلفی را در یک رنگ‌آمیزی مناسب اخذ کند. با توجّه به این که هدف ما این است که خروجی حتماً به رنگ سبز درآید، آن را مستقیماً به رئوس قرمز و آبی از مثلّث مرجع متصّل می‌کنیم. به این ترتیب این اجبار را در یک رنگ‌آمیزی معتبر اعمال خواهیم کرد که خروجی متناظر با $(x \vee y \vee z)$ برای هر عبارت از ورودی مسأله به رنگ سبز باشد. در این حالت ساختار نهایی مورد نظر به صورت شکل ۸ درمی‌آید.

در این حالت اگر یک مقداردهی معتبر برای متغیّرهای مسأله‌ی 3–SAT موجود باشد، کافی است رئوس متناظر با متغیّرها را مطابق با همان مقداردهی به رنگ‌های قرمز و سبز درآوریم، و همه‌ی رئوسی مانند w_2 در شکل ۸ نیز به درستی به رنگ سبز درخواهند آمد (زیرا طبق



شکل ۸: ساختار نهایی پیاده‌سازی عمل منطقی OR در گراف و تحمیل درست بودن خروجی آن

درستی مقداردهی متغیّرها، حدّاقل یکی از سه متغیّر x و y و z سبز خواهد بود؛ و طبق توضیحات فوق رنگ‌آمیزی معتبری برای ساختار شکل ۸ وجود دارد که در آن، w_2 نیز سبز باشد).

همچنین برعکس، اگر فرض کنیم یک ۳–رنگ‌آمیزی معتبر برای گراف ساخته‌شده موجود است؛ در این صورت تمامی رئوس مانند w_2 در شکل ۸ به رنگ سبز در آمده‌اند؛ و این یعنی حدّاقل یکی از سه متغیّر x و y و z سبز هستند (زیرا استدلال کردیم که اگر هر سه قرمز باشند، w_2 نیز باید قرمز باشد) و این یعنی در اثر وجود یک ۳–رنگ‌آمیزی معتبر برای گراف، می‌توان یک مقداردهی مناسب برای مسأله‌ی 3–SAT به دست آورد.

به این ترتیب هر دو طرف معادل بودن حل در دو مسأله را نشان داده‌ایم و با ساختن گراف توصیف‌شده در مراحل فوق، توانسته‌ایم مسأله‌ی $3 - - SAT$ را با یک تحویل چندجمله‌ای به مسأله‌ی ۳–رنگ‌پذیری گراف کاهش دهیم. به این ترتیب مسأله‌ی ۳–رنگ‌پذیری گراف یک مسأله‌ی انپی–تمام خواهد بود؛ و با توجّه به عبارت ۲ برای هر $k \geq 3$ مسأله‌ی k–رنگ‌پذیری گراف نیز انپی–تمام خواهد بود.

۳ مسأله‌ی فروشنده‌ی دوره‌گرد

ابتدا تعریف (نسخه‌ی تصمیم‌گیری) مسأله‌ی فروشنده‌ی دوره‌گرد[۶] (TSP) را مرور می‌کنیم:

ورودی. گراف کامل وزن‌دار G و عدد k.
خروجی. آیا G دور همیلتونی با هزینه‌ی حداّکثر k دارد؟

از جلسه‌ی هجدهم درس می‌دانیم مسأله‌ی یافتن دور همیلتونی در گراف بدون وزن یک مسأله‌ی انپی–تمام است. در این‌جا مسأله‌ی یافتن دور همیلتونی در گراف بدون وزن را به مسأله‌ی فروشنده‌ی دوره‌گرد (مسأله‌ی یافتن دور همیلتونی در گراف کامل وزن‌دار) کاهش می‌دهیم.

برای این منظور فرض کنید در مسأله‌ی یافتن دور همیلتونی، گراف بدون وزن G داده شده است. در این صورت گراف H را از روی آن می‌سازیم. ابتدا همه‌ی یال‌های G را با وزن واحد به H اضافه می‌کنیم. با توجّه به این که H باید ورودی مسأله‌ی فروشنده‌ی دوره‌گرد باشد، لازم است یک گراف کامل باشد. از این رو همه‌ی یال‌های مورد نیاز برای تبدیل شدن H به یک گراف کامل را با وزن ۲ (یا هر عدد دلخواه بزرگ‌تر از یک) به آن اضافه می‌کنیم. به این ترتیب H گرافی کامل خواهد بود که وزن هر یال آن یا برابر با ۱ است و یا ۲؛ و یال e از این گراف وزن ۱ دارد اگر و تنها اگر e یالی از گراف G نیز باشد. در ادامه گراف H را به همراه عدد n (تعداد رئوس G یا H) به عنوان ورودی مسأله‌ی فروشنده‌ی دوره‌گرد در نظر می‌گیریم.

حال ابتدا فرض کنید گراف G یک دور همیلتونی داشته باشد. در این صورت همان دور همیلتونی در گراف H یک دور همیلتونی با وزن n خواهد بود. از طرف دیگر فرض کنید در گراف H دور همیلتونی با وزن حداّکثر n موجود باشد (با توجّه به این که این گراف n رأس دارد و وزن هر یال حداّقل یک است، وزن چنین دوری باید دقیقاً برابر با n باشد). در این صورت هیچ کدام از یال‌های H که وزن بیشتر از ۱ دارند نمی‌توانند در این دور حاضر باشند؛ چرا که این دور باید n یال داشته باشد و وزن هر یال حداّقل ۱ است. بنابراین دور به وزن n باید دقیقاً از n یال با وزن ۱ تشکیل شده باشد؛ و این یعنی این مجموعه یال‌ها در گراف G نیز حاضر هستند؛ به این ترتیب در آن گراف نیز تشکیل یک دور همیلتونی می‌دهند.

به این ترتیب هر دو طرف تناظر دو مسأله ثابت شده است؛ و نشان داده‌ایم که مسأله‌ی یافتن دور همیلتونی در گراف بدون وزن با یک تحویل چندجمله‌ای به مسأله‌ی یافتن دور همیلتونی در گراف کامل وزن‌دار کاهش یافته است. به این ترتیب مسأله‌ی فروشنده‌ی دوره‌گرد یک مسأله‌ی انپی–تمام است.

تمرین. با روشی مشابه نشان دهید هیچ الگوریتم تقریبی با زمان اجرای چندجمله‌ای برای مسأله‌ی فروشنده‌ی دوره‌گرد وجود ندارد. (منظور از الگوریتم تقریبی، الگوریتمی است که بتواند یک خروجی از مسأله را ارائه دهد که حداّکثر k برابر بدتر از جواب بهینه باشد که در آن، k عددی ثابت است. به عنوان مثال الگوریتمی که بتواند یک دور همیلتونی پیدا کند که وزن آن، حداّکثر k برابر بزرگ‌تر از وزن سبک‌ترین دور همیلتونی باشد.)

تمرین. نسخه‌ی تقریبی مسأله‌ی فروشنده‌ی دوره‌گرد (تمرین قبل) را به صورت یک مسأله‌ی تصمیم‌گیری مدل کنید.

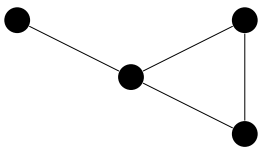
[۶] Traveling Salesman Problem

۴ تحویل مسأله‌ی ۳–رنگ‌پذیری گراف به مسأله‌ی زیرمجموعه‌ی مستقل گراف

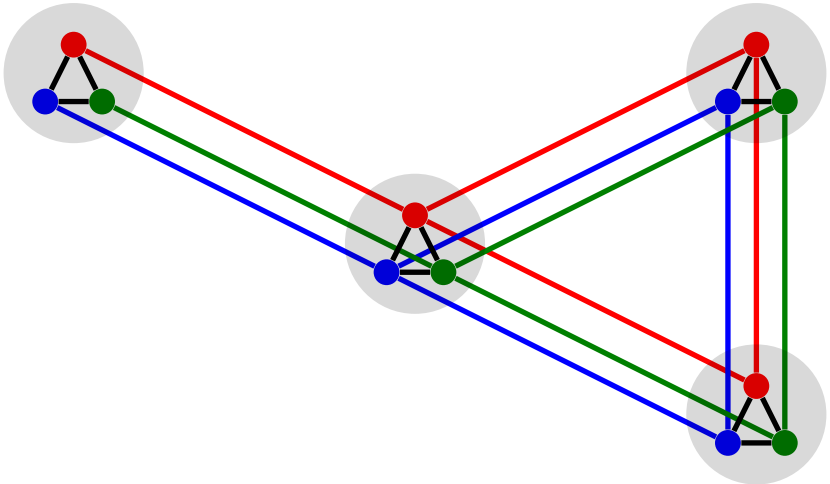
به عنوان یک تمرین دیگر، می‌خواهیم نشان دهیم می‌توان مسأله‌ی ۳–رنگ‌پذیری گراف را به مسأله‌ی یافتن زیرمجموعه‌ی مستقل در گراف کاهش داد؛ یعنی:

$$3\text{–COL} \leq_p \text{IndSet} \tag{۴}$$

برای این منظور باید مسأله‌ی ۳–رنگ‌پذیری گراف را تبدیل به مسأله‌ی مجموعه‌ی مستقل در گراف کنیم. فرض کنید گراف G به عنوان ورودی مسأله‌ی ۳–رنگ‌پذیری داده شده است. می‌خواهیم گراف H و عدد k را به عنوان ورودی‌های مسأله‌ی زیرمجموعه‌ی مستقل طوری تعیین کنیم که گراف H زیرمجموعه‌ی مستقلّی با اندازه‌ی حدّاقل k داشته باشد، اگر و تنها اگر گراف G یک ۳–رنگ‌آمیزی معتبر داشته باشد. ایده‌ی کلّی آن است که به ازای هر رأس از گراف G مانند v سه رأس در H مانند v_r ، v_g، و v_b قرار دهیم که نماینده‌ی اخذ یکی از سه رنگ قرمز، سبز، یا آبی توسّط رأس v باشند. به عبارت دیگر، می‌خواهیم این ساختار را طوری تنظیم کنیم که اگر v در رنگ‌آمیزی G مثلاً رنگ قرمز را به خود بگیرد، در حلّ مسأله‌ی زیرمجموعه‌ی مستقل در گراف G نیز رأس v_r در این مجموعه انتخاب شود. بنابراین به وضوح اوّلین اقدام آن است که سه رأس متناظر با v در گراف H را به یکدیگر متّصل کنیم تا حدّاکثر یکی از آن‌ها مجاز به حضور در زیرمجموعه‌ی مستقل باشد (زیرا متناظراً رأس v در گراف G نمی‌تواند بیشتر از یک رنگ در رنگ‌آمیزی اخذ کند). بنابراین تا به این جا برای هر $v \in G$، سه رأس v_r ، v_g، و v_b را در H قرار داده‌ایم. برای تکمیل فرآیند حل، در صورتی که دو رأس u و v در G به هم وصل باشند، رنگ‌های متناظر آن‌ها را در H به هم متصّل می‌کنیم؛ یعنی به ازای هر یال $e = uv$ در گراف G، سه یال $e_r = u_r v_r$، $e_g = u_g v_g$، و $e_b = u_b v_b$ را در گراف H اضافه می‌کنیم. شکل‌های ۹ و ۱۰ نمونه‌هایی از دو گراف G و H را نمایش می‌دهند.



شکل ۹: نمونه‌ای از گراف G برای مسأله‌ی ۳–رنگ‌پذیری



شکل ۱۰: نمونه‌ای از گراف H برای مسأله‌ی زیرمجموعه‌ی مستقل (متناظر با گراف G در شکل ۹)

تا به این جا گراف H را به صورت کامل تشکیل داده‌ایم. برای تکمیل صورت مسأله‌ی زیرمجموعه‌ی مستقل، باید پارامتر k را نیز انتخاب کنیم. مقدار این ورودی را $k = n$ در نظر می‌گیریم؛ چرا که می‌خواهیم متناظر با هر یک از رئوس گراف G، یک رأس از H را در مجموعه‌ی مستقل انتخاب کرده باشیم.

حال ابتدا فرض کنید یک ۳–رنگ‌آمیزی معتبر در گراف G موجود باشد. در این صورت اگر رنگ انتخابی برای رأس v برابر c باشد که $c \in \{r, g, b\}$، آنگاه در انتخاب زیرمجموعه‌ی مستقل از گراف H رأس v_c را انتخاب خواهیم کرد. با توجّه به این که هر یال از G در رنگ‌آمیزی معتبر این ویژگی را دارد که رنگ دو سر آن متمایز است، هیچ یک از رئوس انتخابی از H نیز به یکدیگر متّصل نیستند و مجموعه‌ی n رأس انتخابی در H یک زیرمجموعه‌ی مستقل n عضوی خواهد بود؛ و به این ترتیب یک پاسخ معتبر برای مسأله‌ی زیرمجموعه‌ی مستقل به دست آمده است.

از طرف مقابل اگر فرض کنیم یک زیرمجموعه‌ی مستقل n عضوی از گراف H موجود است؛ اوّلاً می‌دانیم که امکان ندارد دو رأس مانند v_c و v_d در این مجموعه انتخاب شده باشند که $\{c, d\} \subset \{r, g, b\}$ و $c \neq d$، زیر هر سه رأس v_r و v_g و v_b در گراف H به یکدیگر متّصل هستند. به این ترتیب هر یک از n رأس انتخابی متناظر با یکی از رئوس گراف G خواهند بود. از طرفی چون مجموعه رئوس انتخابی یک زیرمجموعه‌ی مستقل از گراف H هستند، هیچ دوتایی به یکدیگر متّصل نیستند؛ و این یعنی برای هر یال uv از G، رئوس u_c و v_d انتخابی در زیرمجموعه‌ی مستقل ($\{c, d\} \subset \{r, g, b\}$) این ویژگی را دارند که $c \neq d$، به این ترتیب کافی است در رنگ‌آمیزی گراف G رنگ c را برای رأس u و رنگ d را برای رأس v انتخاب کنیم. در نتیجه رنگ دو سر هر یال از G متمایز خواهد بود و یک ۳–رنگ‌آمیزی معتبر برای این گراف خواهیم داشت.

بنابراین هر دو طرف از تناظر دو مسأله اثبات شده است و نشان داده‌ایم مسأله‌ی ۳–رنگ‌پذیری گراف با یک تحویل چندجمله‌ای به مسأله‌ی یافتن زیرمجموعه‌ی مستقل گراف کاهش یافته است.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

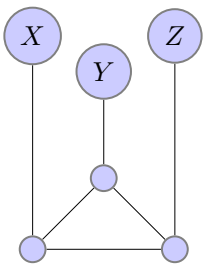
جلسه حل تمرین ۹: مثال‌های بیشتری از ان‌پی_ تمامیت نگارنده: آرمیتا کاظمی‌نجف آبادی

در این جلسه تمرکز مان را بر ایده‌هایی که در تحویل کردن مسائل وجود دارند می‌گذاریم. به این منظور به بررسی چند مسأله که در جلسه نوزدهم با عنوان تمرین مطرح شدند و یک مسأله دیگر با ظاهری چالش برانگیز می‌پردازیم.

۱ تحویل‌هایی از مسأله NAE3SAT:

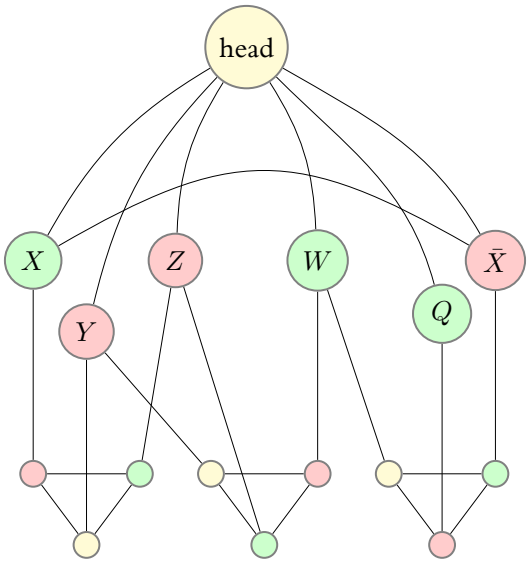
در جلسه نوزدهم دو مسأله به عنوان تمرین مطرح شد که در اینجا آن‌ها را حل می‌کنیم. (تعریف مسأله NAE3SAT در جزوه همان جلسه موجود است.)

تحویل NAE3SAT به 3COL : یک نمونه از مسأله NAE3SAT در نظر می‌گیریم. می‌خواهیم از روی این نمونه، در زمان چندجمله‌ای، گرافی بسازیم که سه رنگ پذیر باشد اگر و تنها اگر مقداردهی مناسب برای لیترال‌های مسأله نمونه وجود داشته باشد.
یکی از ایده‌های مطرح شده برای تحویل، این بود که ابتدا NAE3SAT را به 3SAT و سپس 3SAT را به 3COL تحویل کنیم. اما از آنجایی که قبلا با روش انجام این تحویل‌ها آشنا شده‌ایم و نکته آموزشی جدیدی در آن نیست؛ این کار را انجام نمی‌دهیم.
هدف این است که تحویل را مستقیم انجام دهیم.
به این منظور به ازای هر لیترال در نمونه، یک راس در نظر گرفته و به ازای هر سه لیترالی که در یک عبارت (پرانتز) آمده اند، سه راس اضافه در نظر می‌گیریم که بین دو به دویشان یال است. به هر یک از رئوس این دور سه تایی، یکی از سه راسِ متناظر با لیترال‌های داخل آن پرانتز را نسبت می‌دهیم و بین شان یال می‌کشیم. (طوری که نسبت دادنمان یک به یک باشد.)
رئوسی از گراف که متناظر با یک لیترال بوده اند را رئوس برچسب دار و بقیه رئوس را بدون برچسب می‌نامیم. مثلا در شکل زیر برای عبارت (X, Y, Z)، به ازای هر لیترال راسی برچسب دار وجود دارد و یک دور سه تایی از رئوس بدون برچسب نیز داریم که راس‌های این دور به طور یک به یک، به رئوسِ برچسب دارِ متناظر با لیترال‌های (X, Y, Z) تناظر داده شده اند.

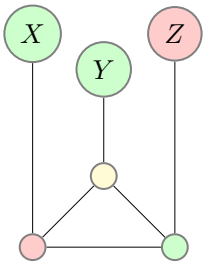


علاوه بر یال‌هایی که تا به حال کشیده‌ایم، بین هر متغیر و نقیضش نیز یالی اضافه می‌کنیم.
بعلاوه برای کامل شدن اثبات می‌بایست یک راس اضافه در نظر بگیریم که به همه رئوس برچسب دار (رئوسی که متناظر با یک لیترال

هستند) یال داشته باشد. این راس خاص را $head$ می‌نامیم.
برای مثال شکل زیر، گراف مربوط به عبارت $(X,Y,Z) \wedge (Y,Z,W) \wedge (W,Q,\bar{X})$ می‌باشد.



ابتدا نشان می‌دهیم هر جوابی برای مسأله نمونه را می‌توان به جوابی برای مسأله سه‌ـرنگ پذیری در گرافی که از روی نمونه ساخته‌ایم تبدیل کرد.
یک مقداردهی درست به لیترال‌های نمونه را در نظر بگیرید. کافی ست رئوسی از گراف که متناظر با هر لیترال هستند را بسته به مقداری که گرفته اند، به رنگ $True$ یا $False$ در بیاوریم. (مثلا رنگ $True$ را در شکل به رنگ سبز و رنگ $False$ را با قرمز نشان داده‌ایم.)
راس $head$ را نیز به رنگ B در می‌آوریم. (رنگ B در گراف با زرد نشان داده شده است.) این کار مشکلی ایجاد نمی‌کند چون راس $head$ فقط به رئوس برچسب دار متصل بود که آنها هم به رنگ $True$ یا $False$ درآمده اند.
بقیه راس‌های گراف نیز قابل رنگ آمیزی هستند؛ زیرا اکنون تنها رئوس بدون برچسب باقی مانده اند. این رئوس نیز هر یک در دوری سه تایی ظاهر شده اند و به جز همسایه‌هایشان در آن دور، دقیقا یک همسایه دیگر دارند که راسی از میان رئوس برچسب دار است (راسی متناظر با یک لیترال از عبارتی سه تایی)
از آنجایی که هیچ سه لیترالی که در یک عبارت آمده اند همرنگ نیستند، با حالت بندی می‌توان نشان داد این دورِ سه تاییِ رئوس که به یک عبارت متناظر شده اند را می‌توان با سه رنگ $True$ و $False$ و B رنگ کرد طوری که هیچ یک از رئوس این دور، با همسایه خود همرنگ نباشند.
درواقع با توجه به اینکه رئوس برچسب دار از قبل رنگ شده اند، برای هر عبارت، بیشتر از دو حالت کلی (که دو لیترال $True$ و یکی $False$ باشد یا بالعکس) پیش نمی‌آید. که این دو حالت نیز درواقع یکی هستند و رنگ آمیزی مطلوبی مطابق شکل برایشان وجود دارد:



به این ترتیب اگر نمونه ای که برای مسأله NAE3SAT در نظر می‌گیریم، مقداردهی مناسب برای لیترال‌هایش موجود باشد، گرافی که از روی آن (در زمان چندجمله‌ای) ساخته‌ایم نیز سه رنگ پذیر است.

حال برای تکمیل تحویل، کافی ست نشان دهیم سه رنگ پذیریِ گراف ساخته شده نیز وجود مقداردهی مناسب برای لیترال‌های نمونه را نتیجه می‌دهد.

یک ۳_رنگ آمیزی مطلوب برای گراف در نظر بگیرید. در این رنگ آمیزی، رنگ متناظر با راس $head$ را B می‌نامیم.

راسِ $head$ فقط به رئوس برچسب دار متصل بود، بنابراین رنگ رئوس متناظر با لیترال‌ها را می‌توان $True$ و $False$ نامید! (درواقع رئوس متناظر با لیترال‌ها یا همان رئوس برچسب دار، دو رنگ به جز رنگ B می‌توانند داشته باشند که یکی از این رنگ‌ها را $True$ و دیگری را $False$ می‌نامیم.)

یک پرانتز دلخواه از نمونه مسأله NAE3SAT در نظر بگیرید و به رئوس متناظر با لیترال‌های آن در گراف نگاه کنید. فرض کنید این رئوس X و Y و Z باشند.

به دور سه تایی متناظر با این عبارت توجه می‌کنیم. در یک رنگ آمیزی مطلوب، هر یک از این سه راسی که در این دور آمده، رنگش با دیگری فرق دارد. بنابراین رئوس X و Y و Z که هر یک به یکی از این سه یال دارند نمی‌توانند همگی همرنگ باشند.

پس فرض کنید به لیترال‌هایی که رئوس متناظر با آنها به رنگ $True$ درآمده، مقدار $True$ و به لیترال‌هایی که رئوس متناظر با آنها به رنگ $False$ درآمده، مقدار $False$ نسبت بدهیم.

ادعا می‌کنیم این مقداردهی، معتبر و صدق پذیر است.

علت معتبر بودن مقداردهی این است که در گراف، بین هر لیترال و متناقض اش یال کشیده بودیم بنابراین، رئوس متناظر با هیچ دو لیترالی که نقیض هم باشند نمی‌توانند به یک رنگ درآمده باشند.

علت صدق پذیریِ مقداردهی هم همان طور که گفتیم، این است که هیچ سه تایی از لیترال‌ها که در یک پرانتز آمده اند، نمی‌توانند هر سه مقدار $True$ یا هر سه مقدار $False$ گرفته باشند.

پس نشان داده‌ایم تحویل ارائه شده معتبر است.

تحویل NAE3SAT به MaxCut: در اینجا فرض می‌کنیم مسأله MaxCut به این صورت است که گراف G بدون جهت و بدون وزن دهی یال‌ها (انگار هر یال وزن یک دارد) و عدد k به عنوان ورودی داده شده و سوال این است که آیا برشی با اندازه حداقل k در G وجود دارد یا خیر.

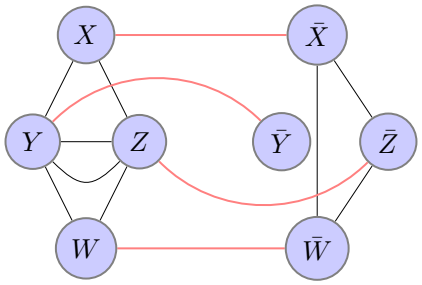
برای تحویل مسأله NAE3SAT، نمونه ای از آن را در نظر می‌گیریم. می‌خواهیم در زمان چندجمله‌ای گرافی از روی این نمونه بسازیم به طوری که بین پاسخ مسأله MaxCut در این گراف و پاسخ مسأله NAE3SAT روی نمونه اولیه، تناظری برقرار شود.

به ازای هر متغیر در نمونه مذکور، راسی در گراف قرار می‌دهیم. (اگر نقیض آن متغیر در نمونه نیامده، به ازای نقیض آن متغیر هم راسی در گراف قرار می‌دهیم.) بین راس‌هایی که متغیر متناظرشان در نمونه در یک عبارت (پرانتز) قرار دارند، یال می‌کشیم. به این یال‌ها، یال **نوع اول** می‌گوییم. توجه می‌کنیم که ممکن است دو متغیر در بیش از یک عبارت با هم آمده باشند. در نتیجه بین دو راس از گراف ساخته شده، ممکن است بیش از یک یال وجود داشته باشد. (درواقع در این راه حل، مسأله NAE3SAT را به مسأله MaxCut برای گراف‌های چندگانه تحویل می‌کنیم.)

به عنوان تمرین، سعی کنید مسأله NAE3SAT را به مسأله MaxCut برای گراف‌های ساده تبدیل کنید.

به علاوه، بین هر راسِ متناظر با یک متغیر و راسِ متناظر با نقیضِ آن متغیر نیز یال می‌کشیم. به این یال‌ها یال **نوع دوم** می‌گوییم.

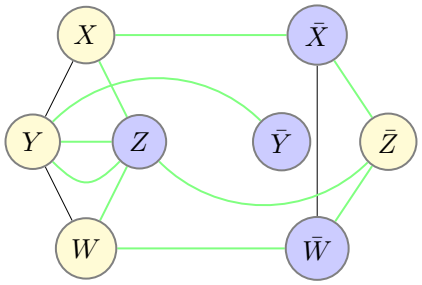
مثلا گرافی که برای نمونه ی $(X,Y,Z) \wedge (Y,Z,W) \wedge (\bar{X},\bar{Z},\bar{W})$ ساخته می‌شود، گرافی مانند شکل زیر است که در آن یال‌های نوع اول با رنگ سیاه و یال‌های نوع دوم با رنگ قرمز مشخص شده اند.



حال یک مقداردهی معتبر برای لیترال‌های نمونه داده شده در نظر بگیرید. از روی این مقداردهی معتبر، می‌توانیم برشی از گراف با حداقل اندازه $2m + n$ به دست آوریم که در اینجا m تعداد عبارت‌ها و n تعداد متغیرها در نمونه اولیه است.

برش مذکور به این ترتیب به دست می‌آید که رئوس متناظر با متغیرهایی که مقدار $True$ به آنان داده شده در یک بخش و متغیرهای با مقدار $False$ را در بخش دیگر قرار دهیم. با توجه به اینکه هیچ سه متغیری که در یک عبارت آمده اند در یک دسته قرار نمی‌گیرند (هر سه $True$ یا هر سه $False$ نیستند)، به ازای هر یک از m عبارت موجود در نمونه، دو یال متمایز از یال‌های **نوع اول** در برش وجود دارد. به علاوه هیچ متغیری با نقیض خود در یک بخش قرار نمی‌گیرد. بنابراین به تعداد متغیرها یعنی n تا، از یال‌های **نوع دوم** متمایز، در برش معرفی شده داریم. پس برشی با حداقل اندازه $2m + n$ را پیدا کردیم. و جواب مسأله وجود برش با حداقل اندازه $2m + n$ در این نمونه مثبت است.

مثلا در گراف زیر که متناظر با نمونه $(X, Y, Z) \wedge (Y, Z, W) \wedge (\bar{X}, \bar{Z}, \bar{W})$ بود، مجموعه یال‌هایی که بین دو راس که یکی زرد و دیگری آبی باشد، یال‌های یک برش بیشینه است. (یال‌های برش با رنگ سبز مشخص شده اند.)



برای آنکه اثبات تحویل پذیری کامل شود، می‌بایست نشان دهیم وجود یک برش با حداقل اندازه ی $2m + n$ در گرافِ ساخته شده از روی نمونه، وجود یک مقداردهی مطلوب برای لیترال‌های نمونه را نتیجه می‌دهد.

پس فرض کنید برش با حداقل اندازه $2m + n$ در گراف مان وجود دارد که راس‌ها را به دو دسته با نام‌های T و F تقسیم کرده است. از بین یال‌های **نوع اول**، حداکثر $2m$ تا می‌توانند در برش آمده باشند. چون هر یک از m عبارت در مسأله نمونه، حداکثر سه لیترال دارد و سه راس متناظر با این لیترال‌ها به هر شکلی که بین دو بخشِ تقسیم شوند، بیش از دو یال نمی‌توانند در برش ایجاد کند. از طرفی برای آنکه اندازه برش، به $2m + n$ برسد، حداقل به n یالِ دیگر نیاز داریم. که مجبور هستیم از بین یال‌های **نوع دوم** انتخابشان کنیم. این یال‌ها بین هر متغیر و نقیض آن متغیر هستند و تعداد متغیرها هم بیشتر از n نیست. بنابراین می‌بایست هر متغیر و نقیضش در بخش‌های متفاوتی از برشِ مشخص شده قرار بگیرند. بسته به اینکه راس متناظر یک متغیر در دسته T یا F قرار گیرد، می‌توانیم به آن مقدار $True$ یا $False$ بدهیم. و با توجه به آنچه گفتیم، می‌توان نشان داد این مقداردهی برای لیترال‌های نمونه معتبر است. زیرا اولا راسِ متناظر با یک متغیر و راسِ متناظر با نقیض آن، در دو دسته متفاوت اند که نتیجه می‌دهد مقدار دو متغیر نقیض، یکسان نخواهد شد؛ دوما از هر عبارت (پرانتز) در نمونه، حداکثر به دو لیترال مقدار $True$ یا $False$ داده شده. اگر اینطور نباشد و هر سه لیترال در یک پرانتز یک مقدار گرفته باشند؛ به این معناست که هر سه در یکی از مجموعه‌های T یا F قرار گرفته اند. اما برای اینکه اندازه برش بیشینه باشد، باید هر سه تاییِ متناظر با یک عبارت، حداقل دو یال در برش داشته باشد. پس چنین حالتی رخ نمی‌دهد و مقداردهی برای نمونه معتبر است.

در نهایت تحویل ارائه داده شده نیز معتبر است.

به عنوان تمرین می‌توان ثابت کرد ورژن دیگری از مسأله NAE3SAT*، که همان شروط بعلاوه شرط مثبت بودن لیترال‌ها را دارد، ان‌پی‌ـ تمام است. (مثبت بودن لیترال‌ها به این معناست که نقیض هیچ لیترالی در نمونه نیامده باشد.)*

۲ تحویل مسأله 3COL به مسأله Exactly1in3SAT :

در جلسه نوزدهم با تعریف مسأله NAE3SAT آشنا شدیم. برای اینکه اثبات کنیم ان‌پی‌ـ تمام است، کافی ست تحویلی از یک مسأله ان‌ـ پی تمام به آن ارائه دهیم. (زیرا از قبل می‌دانستیم که در کلاس ان‌پی هست.)
در اینجا سعی داریم مسأله 3COL را به آن تحویل کنیم. بنابراین فرض می‌کنیم گراف G داده شده است. به ازای هر یال بین دو راس u و v در گراف، سه لیترال r_{uv}، g_{uv} و b_{uv} و همچنین برای هر راس مانند v نیز لیترال‌های r_v، g_v و b_v را در نظر می‌گیریم.
اگر بین u و v یالی قرار دارد، عبارت‌های (r_{uv}, r_v, r_u)، (g_{uv}, g_v, g_u)، (b_{uv}, b_v, b_u)، (b_{uv}, r_{uv}, g_{uv})، (b_u, r_u, g_u)، (b_v, r_v, g_v) را در مجموعه سه تایی‌هایی که باید با هم and شوند تا نمونه ای از مسأله Exactly1in3SAT بسازند قرار می‌دهیم.
حال اگر گراف G سه‌ـ رنگ پذیر باشد، به ازای هر دو راسی که به هم وصل اند، و رنگ دلخواهی مانند r ، یا دقیقا یکی از دو راس به رنگ r هستند یا اینکه می‌توانیم فرض کنیم که انگار یال بین این دو راس به رنگ r در آمده است. درواقع انگار دقیقا یک لیترال از عبارت (r_{uv}, r_v, r_u)، مقدار $True$ دارد. برای هر عبارتی که در نمونه ساخته شده داریم، همین اتفاق رخ می‌دهد. یعنی مثلا اگر بدون کاستن از کلیت فرض کنیم دو راس u و v، یکی سبز و دیگری قرمز باشد، در این صورت لیترال‌های b_{uv}, r_v, g_u، مقدار $True$ و لیترال‌های $g_{uv}, r_{uv}, g_v, b_v, r_u, b_u$، مقدار $False$ خواهند گرفت. بنابراین از هر یک از پرانتزهای (r_{uv}, r_v, r_u)، (g_{uv}, g_v, g_u)، (b_{uv}, b_v, b_u)، (b_{uv}, r_{uv}, g_{uv})، (b_u, r_u, g_u)، (b_v, r_v, g_v)، دقیقا یک لیترال مقدار $True$ می‌گیرد. بنابراین اگر گراف نمونه، سه‌ـ رنگ پذیر باشد، برای نمونه ای که از روی آن در زمان چندجمله‌ای برای مسأله Exactly1in3SAT ساخته‌ایم، مقداردهی مطلوب وجود دارد.

برای تکمیل تحویل کافی ست نشان دهیم که اگر مقداردهی مطلوبی برای نمونه ساخته شده مسأله Exactly1in3SAT وجود داشته باشد، گراف نمونه سه‌ـ رنگ پذیر است.
این را می‌دانیم که متناظر با دو راس u و v در گراف که بین شان یالی باشد، پرانتزهای (r_{uv}, r_v, r_u)، (g_{uv}, g_v, g_u)، (b_{uv}, b_v, b_u)، (b_{uv}, r_{uv}, g_{uv})، (b_u, r_u, g_u)، (b_v, r_v, g_v) در نمونه ساخته شده وجود دارند. می‌دانیم لیترال‌ها طوری مقداردهی شده اند که از هر پرانتز دقیقا یکی مقدار $True$ گرفته باشد. اگر رنگ راس v را همان لیترالی در نظر بگیریم که در (b_v, r_v, g_v) مقدار $True$ گرفته و رنگ u را نیز مشابها لیترالی در نظر بگیریم که در (b_u, r_u, g_u)، مقدار $True$ گرفته، در این صورت یک ۳ـ رنگ آمیزی معتبر برای گراف به دست خواهیم آورد. چون با توجه به اینکه تنها یک مقدار از لیترال‌های هر یک از عبارت‌های (r_{uv}, r_v, r_u)، (g_{uv}, g_v, g_u)، (b_{uv}, b_v, b_u)، مقدار $True$ می‌گیرد؛ u و v نمی‌توانند همرنگ باشند. درنهایت تحویل چندجمله‌ای ارائه داده شده معتبر است.

۳ مسأله ای با ظاهر ترسناک!

فرض کنید اعداد صحیح $x_1, x_2, ..., x_n$ به عنوان ورودی داده شده اند.
آیا $\int_{-\pi}^{\pi}(\cos x_1\theta)(\cos x_2\theta)...(\cos x_n\theta)d\theta \neq 0$ برقرار است؟

درواقع می‌خواهیم نشان دهیم مسأله ان‌پی‌ـ تمام است.

ابتدا می‌بایست صورت مسأله را ساده تر کنیم. با توجه به تساوی زیر مقادیر را جایگذاری می‌کنیم.

$$\cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

جایگذاری:

$$\int_{-\pi}^{\pi} (\cos x_1\theta)(\cos x_2\theta)...(\cos x_n\theta)d\theta = \frac{1}{2^n}\int_{-\pi}^{\pi}\prod_{j=1}^{n}(e^{ix_j\theta} + e^{-ix_j\theta})d\theta$$

$$= \frac{1}{2^n}\int_{-\pi}^{\pi}\sum_{A\subseteq[n]}(\prod_{j\in A} e^{ix_j\theta})(\prod_{j\notin A} e^{-ix_j\theta})d\theta = \frac{1}{2^n}\int_{-\pi}^{\pi}\sum_{A\subseteq[n]} e^{i\theta[\sum_{j\in A}x_j - \sum_{j\notin A}x_j]}d\theta$$

درستی عبارت زیر را می‌پذیریم:

$$\int_{-\pi}^{\pi} e^{iy\theta}d\theta = \begin{cases} 2\pi & y = 0 \\ 0 & y \neq 0 \end{cases}$$

بنابراین تنها در حالتی عبارت اولیه مان صفر نمی‌شود که A وجود داشته باشد طوری که:

$$\sum_{j\in A} x_j - \sum_{j\notin A} x_j = 0$$

یا درواقع جواب مسأله partition روی ورودی‌های $x_1, x_2, ..., x_n$ مثبت باشد. (با مسأله partition نیز در جلسه نوزدهم آشنا شدیم.) درواقع با نوشتن عبارات جبری اخیر، نشان دادیم این دو مسأله با هم معادل هستند. بنابراین هر یک قابل تحویل به دیگری ست. (چون دو مسأله معادلند، عملا نیازی به تابع تحویل هم نداریم.) پس چون می‌دانستیم مسأله partition انپی‌ـ تمام است، این مسأله نیز انپی‌ـ تمام خواهد بود.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس حل تمرین: مرتضی علیمی

[بهار ۹۹]

جلسه حل تمرین: جلسه ۱۰ نگارنده: محمد حسین مروجی

در این جلسه به بررسی چند مسئله درباره روش حل جست‌وحوی موضعی [1] پرداخته می‌شود.

۱ زمان بندی روی ماشین های یکسان

سوال: تعداد n کار داده شده است به طوری که کار i ام p_i واحد طول می کشد و همچنین m تا ماشین یکسان داده شده است که می توانند کارها را انجام دهند(زمان شروع همه ماشین ها یکسان است). الگوریتمی ارائه دهید که کارها را به ماشین ها اختصاص دهد به طوری که کارها در حداقل زمان انجام شوند. برای مثال اگر $n = 2$ به حالت کلی تری از مسئله تقسیم بندی مجموعه [2] می رسیم.

برای حل این سوال سعی می کنیم از روش جست و جوی موضعی استفاده کنیم. یکی از کارهایی که می توانیم انجام دهیم این است که کارها را به طور دلخواه بین ماشین ها تقسیم کنیم و سپس سعی کنیم با جابه‌جا کردن کارهای انتهایی جواب را بهینه تر کنیم. یعنی اگر کاری که در انتهای یک ماشین وجود داشت که با اختصاص دادن آن کار به انتهای ماشین دیگری جواب بهینه تر می شد این کار را انجام دهیم و زمانی که نتوانیم با یک جابه جایی جواب را بهینه تر کنیم الگوریتم پایان می یابد.

حال سعی می کنیم الگوریتمی که بیان کردیم را تحلیل کنیم. به این صورت که ابتدا باید بررسی کنیم که این الگوریتم آیا به جواب بهینه می رسد یا خیر و سپس بررسی کنیم که مرتبه زمانی الگوریتم چگونه است.

ابتدا مثالی را بیان می کنیم که این الگوریتم به جواب بهینه نمی رسد. فرض کنید ۲ ماشین و ۵ کار به طول های ۴،۴،۳،۲و۵ داده شده است. حال اگر زمانی که کار هارا به طور دلخواه تخصیص بدهیم و به ماشین اول کارهای ۴،۴ اختصاص یابد و به ماشین دوم کارهای ۲،۳،۵ اختصاص یابد می توانیم به راحتی ببنیم که با تعویض کار ها بین دو ماشین نمی توانیم به حالت بهینه که اختصاص ۲،۴،۳ به ماشین اول و اختصاص ۴،۵ به ماشین دوم است،برسیم زیرا با یک تغییر نمی توانیم جواب را بهینه تر کنیم.

ادعا: جوابی که این الگوریتم می دهد حدااکثر دوبرابر بدتر از جواب بهینه است.

اگر جواب بهینه برای مسئله را برابر OPT در نظر بگیرم به وضوح $OPT \geq p_i$.از طرف دیگر نیز داریم $OPT \geq \frac{\sum_{i=1}^{n} p_i}{m}$. در ادامه ماشینی را در نظر می گیریم که بیش از همه طول کشیده است و طول آخرین کار آن را P_l می گیریم و فرض کنید طول زمان اجرا این ماشین برابر C_m باشد. در این صورت داریم:

$$\begin{cases} P_l \leq OPT \\ C_m - P_l \leq \frac{\sum_{i=1}^{n} P_i}{m} \leq OPT \end{cases} \Rightarrow C_m \leq 2OPT$$

پس اثبات کردیم که الگوریتم دو تقریب است.

[1] Local search
[2] Partition set

برای تحلیل زمانی الگوریتم ادعایی را ثابت می کنیم ولی قبل از آن فرض می کنیم که در الگوریتمی که ارائه دادیم زمانی که یک کار پایانی را می خواهیم جابه‌جا کنیم اگر این کار را می توانستیم به چند ماشین اختصاص دهیم و جواب بهینه تر شود، به آن ماشینی اختصاص دهیم که طول زمان کارش کمتر باشد.

ادعا: در طی اجرای الگوریتم هر کار جداکثر یکبار جابه‌جا می شود.

برهان: کمینه زمان اجرا ماشین ها در طی اجرای الگوریتم صعودی است زیرا در هر گام از اجرای الگوریتم فقط زمان اجرا دو ماشین که کار در آن ها جابه‌جا شده است تغییر می کند که اگر با این تعویض زمان کمینه اجرا تغییر کند نتیجه می گیریم نباید این تعویض انجام می شده است. پس حال که زمان اجرا کمینه در طول اجرای الگوریتم صعودی است اگر یک کار دو بار تعویض شده باشد به این معنی است که کمینه زمان اجرای الگوریتم نسبت به زمانی که ابتدا این کار جابه‌جا شده بود، کمتر است که این با صعودی بودن کمینه زمان اجرا در تناقض است. پس ادعا ثابت می شود.

۲ مسئله پوشش راسی

می خواهیم برای مسئله پوشش راسی که در جلسات گذشته آن را دیده ایم الگوریتمی با زمان نمایی بهتر پیدا کنیم.ابتدا یک نماد را تعریف می کنیم.

$$\tilde{O}(f(n)) := O(f(n) * Poly\log(n))$$

می خواهیم الگوریتمی با مرتبه زمانی $\tilde{O}(3^{\frac{n}{2}})$ برای مسئله پوشش راسی ارائه دهیم.

الگوریتم: ابتدا یک تطابق بیشینه برای گراف پیدا می کنیم.بوضوح برای هر یال که در تطابق بیشینه آمده است ۳ حالت داریم یا باید هر دو راس آن در پوشش راسی کمینه بیایند یا یکی از دو راس آن در آن بیایند. حال روی همین موضوع تمام حالات را بررسی می کنیم که در کل حداکثر $3^{\frac{n}{2}}$ حالت می شود. برای هر حالت اگر یالی وجود داشته باشد که توسط مجموعه انتخابی، پوشانده نشده باشد قطعا یکی از رئوس آن در خارج از تطابق بیشینه قرار دارد که در این صورت آن راس را به مجموعه انتخابی اضافه می کنیم.بوضوح به جواب بهینه در این حالت می رسیم. مرتبه زمانی الگوریتم نیز همان چیزی است که می خواستیم چون هر حالتی را که بررسی می کنیم با زمان چند جمله ای می توانیم به مجموعه مینیمال از پوشش راسی برسیم.

سوال: فرض کنید عددی صحیح داده شده است و در مورد این عدد صحیح اطلاعاتی نداریم.می خواهیم با شروع از روی نقطه صفر اعداد صحیح به عدد مورد نظر برسیم.الگوریتم با ضریب رقابتی ثابت برای این سوال ارائه دهید.

الگوریتم: در مرحله i ام اگر $i-1$ زوج بود 2^{i-1} گام به سمت راست می رویم و اگر فرد بود 2^{i-1} گام به سمت چپ می رویم.

۳ الگوریتم انشعاب و کران برای مسئله پوشش مجموعه‌ای

مسئله پوشش مجموعه‌ای: فرض کنید $U = \{e_1, e_2, ..., e_n\}$ و $S \subseteq 2^U$ و $w : S \to R^+$ هستند.الگوریتمی ارائه دهید به طوری که مجموعه $T \subseteq S$ را بیابد به طوری که $\bigcup_{A \in T} A = U$ کمینه شود و همچنین $\sum_{A \in T} w(A)$.

با استفاده از مسئله پوشش راسی می توانیم ثابت کنیم این مسئله سخت-np است زیرا می توانیم در مسئله پوشش راسی کمینه،عناصر را یال ها و مجموعه هارا نیز راس ها در نظر بگیرم حال اگر یک یال به راسی وصل بود آن را در مجموعه راس مورد نظر قرار دهیم. وزن مجموعه ها را نیز یک می گیریم.بوضوح با این تبدیل مسئله پوشش راسی به یک حالت خاص از مسئله پوشش مجموعه ای تبدیل می شود. پس حکم مورد نظر ثابت می شود.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس حل تمرین: مرتضی علیمی

[بهار ۹۹]

جلسه حل تمرین ۱۱: الگوریتم‌های برخط[1] نگارنده: سنا نادعلی

در این جلسه به بررسی دو مسئله‌ی برخط ارزیابی درخت بازی[2] و پیدا کردن تطابق بیشینه و برخی الگوریتم‌های قطعی[3]، غیرقطعی[4] و تصادفی[5] برای حل آن‌ها می‌پردازیم. همچنین سعی می‌کنیم بهینه بودن برخی از این الگوریتم‌ها را ثابت کنیم.

۱ ارزیابی درخت بازی

۱.۱ تعریف صورت مسئله

برای یک بازی k مرحله‌ای دونفره که در هر مرحله بازیکن دو حق انتخاب دارد، درخت بازی، یک درخت دودویی[6] کامل است که هر رأس آن مقدار True یا False می‌گیرد. مقدار True به این معناست که فردی که دراین مرحله نوبتش است برنده خواهد بود. و برد یک نفر به معنای باخت دیگری است. می‌خواهیم بررسی کنیم که آیا نفر اول استراتژی برد دارد یا نه. در واقع می‌خواهیم مقدار ریشه درخت را مشخص کنیم.

اگر مقدار دو فرزند یک رأس را بدانیم، مقدار آن رأس مشخص می‌شود. مقدار این رأس تنها در حالتی False خواهد بود که که هر دو فرزندش مقادیر True داشته‌باشند. زیرا در این رأس هر انتخابی که انجام گیرد فرد دیگر برنده خواهد بود. و اگر حداقل یکی از فرزندان یک رأس مقدار False داشته‌باشد، فرد با بردن بازی به حالت نظیر این رأس، باخت طرف مقابل که معادل برد خودش خواهد بود را تضمین می‌کند. با دقت در این موضوع درمی‌یابیم که می‌توان هر رأس را معادل یک گیت NAND و درخت حاصل را معادل یک مدار در نظر گرفت.

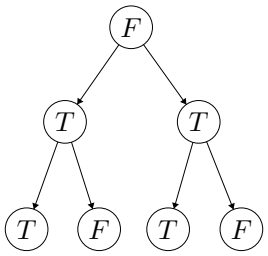
در ادامه فرض شده‌است که عمق این درخت، معادلاً تعداد مراحل بازی، زوج است و $n = 2^k$ تعداد برگ‌های درخت است. همچنین در تحلیل الگوریتم‌های زیر زمان اجرای الگوریتم را تعداد برگ‌های بررسی‌شده آن در نظر می‌گیریم.

۲.۱ الگوریتم قطعی

یک ایده ساده استفاده از الگوریتم‌های بازگشتی یا الگوریتم DFS است. با توجه به قاعده توضیح داده‌شده در بالا از روی مقادیر فرزندان یک رأس مقدار آن رأس مشخص می‌شود. همچنین با توجه به نحوه پر شدن درخت بدیهی است که اگر یکی از فرزندان یک رأس مقدار False باشد، نیاز به بررسی فرزند دیگر این رأس نخواهد بود. و به این صورت می‌توان از انجام محاسبات اضافه جلوگیری کرد.

[1] Online
[2] Game tree evaluation
[3] Determinestic
[4] Non-determinestic
[5] Randomized
[6] Binary

فرض کنید الگوریتم قطعی به این صورت باشد که هر بار به طور بازگشتی ابتدا مقدار فرزند چپ رأس مورد نظر را تأیین کند سپس اگر این مقدار True بود، مقدار فرزند دیگر را نیز محاسبه کند. در بدترین حالت این الگوریتم همه برگ‌ها را بررسی خواهد کرد. برای مثال درخت بازی زیر را در نظر بگیرید:



حکمی مشابه برای حالت کلی‌تر قابل اثبات است:

ادعا ۱. *هر الگوریتم قطعی برای مسئله فوق، در بدترین حالت همه برگ‌ها را بررسی می‌کند.*

اثبات: برای الگوریتم دلخواه A، با استقرا روی k نشان می‌دهیم برگ‌های درخت بازی را می‌توان به گونه‌ای مقداردهی کرد که مقدار ریشه هر زیردرخت، به مقدار آخرین برگی بستگی داشته‌باشد که توسط الگوریتم A بررسی می‌شود.

پایه استقرا: فرض کنید $k = 1$. الگوریتم A یکی از ۲ برگ این درخت را زودتر بررسی می‌کند. مقدار این برگ را True در نظر بگیرید. در این صورت اگر مقدار برگ دیگر False باشد، مقدار ریشه نیز False و در غیر این صورت True خواهدبود. پس پایه استقرا برقرار است.

گام استقرا: فرض کنید برای $k = i$ حکم را می‌دانیم. برای $k = i + 1$ حکم را ثابت می‌کنیم. آن فرزند ریشه که مقدارش زودتر بررسی می‌شود را در نظر بگیرید. طبق فرض استقرا زیردرخت این رأس را می‌توان به گونه‌ای مقداردهی کرد که ارزش این رأس به آخرین برگی که از این زیردرخت بررسی می‌شود بستگی داشته‌باشد. حال این برگ را به گونه‌ای مقدار می‌دهیم که ارزش این فرزند True شود. بنابراین الگوریتم مجبور خواهدبود که ارزش فرزند دیگر را مشخص کند. باز هم با استفاده از فرض استقرا می‌توان زیردرخت این فرزند را به گونه‌ای مقداردهی کرد که ارزشش وابسته به آخرین برگی باشد که از این زیردرخت بررسی می‌شود (این برگ به وضوح آخرین برگ درخت خواهدبود که بررسی می‌شود). اگر آخرین برگ این زیردرخت به‌گونه‌ای مقداردهی شود که ارزش این فرزند True شود، ارزش ریشه False و در غیر این صورت True خواهدبود.

۳.۱ الگوریتم غیرقطعی

منظور از الگوریتم غیرقطعی، همانطور که در جلسات پیچیدگی بررسی شد، الگوریتمی است که به طور جادویی انتخاب‌های درست انجام می‌دهد! در این مسئله یک الگوریتم غیرقطعی می‌تواند به طور جادویی $m < n$ برگ از درخت بازی را انتخاب کند که بررسی آن‌ها برای مشخص کردن مقدار ریشه درخت کافی باشد. ادعای زیر نشان می‌دهد مقدار m را تا $\sqrt{n} = 2^{\frac{k}{2}}$ می‌توان کاهش داد.

ادعا ۲. *برای هر مقداردهی دلخواه یک درخت بازی، می‌توان $2^{\frac{k}{2}}$ برگ از درخت را مشخص کرد که دانستن مقادیر آن‌ها برای مشخص کردن مقدار ریشه درخت کافی باشد.*

اثبات: ابتدا تعریف می‌کنیم:

تعداد برگ‌های زیردرخت به عمق k که باید مقدارشان را بدانیم تا از True بودن ریشه زیردرخت مطمئن شویم. =: $T(k)$

تعداد برگ‌های زیردرخت به عمق k که باید مقدارشان را بدانیم تا از False بودن ریشه زیردرخت مطمئن شویم. =: $F(k)$

حال مقادیر $T(k)$ و $F(k)$ را به صورت بازگشتی محاسبه می‌کنیم. مقادیر $T(0)$ و $F(0)$ به وضوح برابر با ۱ است. زیرا مقدار تنها برگ که همان ریشه است را باید بررسی کنیم. برای هر $k > 0$ برای این که مقدار ریشه این زیردرخت True باشد یعنی حداقل یکی از فرزندانش مقدار False دارد. پس تنها مقدار تعدادی از برگ‌ها را لازم است بررسی کنیم که مطمئن شویم مقدار این فرزند False است. یعنی خواهیم داشت:

$$T(k) = F(k-1) \quad (۱)$$

برای این که مقدار ریشه این زیردرخت False شود هر دو فرزند این رأس باید مقدار True بگیرند. پس برای اطمینان یافتن از این که ریشه زیردرخت False است باید از True بودن هر دو فرزندش اطمینان حاصل کنیم. پس خواهیم داشت:

$$F(k) = 2 \times T(k-1) \quad (۲)$$

حال از ترکیب (1) و (2)، دو نتیجه زیر به دست خواهدآمد:

$$F(k) = 2 \times F(k-2) \ \rightarrow \ F(k) = 2^{\frac{k}{2}}$$
$$T(k) = 2 \times T(k-2) \ \rightarrow \ T(k) = 2^{\frac{k}{2}}$$

پس چه مقدار ریشه False باشد چه True، با انتخاب کردن حداکثر $2^{\frac{k}{2}}$ رأس خوب می‌توان از مقدار ریشه مطمئن شد.

۴.۱ الگوریتم تصادفی

شاید ساده‌ترین الگوریتم تصادفی که برای حل این مسئله به ذهن می‌رسد چنین باشد که با استفاده از الگوریتم DFS درخت بازی را پیمایش کنیم. به این صورت که با احتمال $\frac{1}{2}$ ابتدا فرزند راستی یک رأس را بررسی کنیم. اگر مقدارش False بود، نیاز به بررسی مقدار فرزند دیگر نخواهدبود ولی در غیر این صورت باید فرزند دیگر را نیز بررسی کنیم. در ادامه بررسی خواهیم کرد که این الگوریتم در بدترین حالت، به طور میانگین چه تعداد از برگ‌ها را بررسی خواهدکرد.

تعریف می‌کنیم:

امید تعداد برگ‌های زیردرخت به عمق k که در بدترین حالت مقدارشان بررسی می‌شود تا از True بودن ریشه زیردرخت مطمئن شویم. $:= T(k)$

امید تعداد برگ‌های زیردرخت به عمق k که در بدترین حالت مقدارشان بررسی می‌شود تا از False بودن ریشه زیردرخت مطمئن شویم. $:= F(k)$

اگر مقدار ریشه زیردرخت به عمق $k > 0$ برابر False باشد این ریشه دو فرزند دارد که مقدار هر دو True است. پس الگوریتم هر دو فرزند این رأس را بررسی خواهدکرد که برای هر یک به طور میانگین $F(k)$ تعداد برگ را باید بررسی کند. حال فرض کنید مقدار ریشه برابر با True باشد. روی بدترین ورودی مقدار فرزندان این رأس (True،False) یا (False،True) خواهدبود(چرا؟). پس به احتمال $\frac{1}{2}$ اولین رأس True خواهدبود و بررسی فرزند دیگر لازم می‌شود. پس می‌توان نوشت:

$$F(k) = 2T(k-1)$$
$$T(k) = \frac{1}{2}(T(k-1) + F(k-1)) + \frac{1}{2}F(k-1) = F(k-1) + \frac{1}{2}T(k-1)$$

با جایگذاری عبارت اول در دوم خواهیم داشت:

$$T(k) = \frac{1}{2}T(k-1) + 2T(k-2), T(0) = 1, T(1) = \frac{3}{2}$$

با استفاده از معادله مشخصه عبارت بازگشتی فوق مقدار $T(k)$ برابر $(\frac{1+\sqrt{33}}{4})^k \approx n^{0.753}$ به دست می‌آید.

در ادامه به بررسی این موضوع خواهیم پرداخت که زمان اجرای این الگوریتم تصادفی چقدر با مقدار بهینه فاصله دارد. با توجه به اصل مینمکس یائو، اگر بتوانیم یک تابع توزیع روی مقادیر برگ‌ها ارائه دهیم که بهترین الگوریتم قطعی روی این توزیع، نتواند میانگین زمان اجرایی بهتر از زمان اجرای الگوریتم تصادفی ارائه‌شده در بالا داشته‌باشد به مقصود خود رسیده‌ایم.

فرض کنید احتمال True بودن هر برگ برابر p باشد. در این صورت در عمق یکی بالاتر احتمال True بودن یک رأس برابر با $1-p^2$ خواهد بود. اگر بتوان مقدار p را به گونه‌ای تعیین کرد که این دو احتمال برابر شوند به طور استقرایی ثابت می‌شود که در این توزیع، هر رأس این درخت با احتمال p مقدار True خواهدداشت. و این، موجع به راحتی محاسبات می‌شود. پس:

$$p = 1 - p^2 \rightarrow p^2 + p - 1 = 0 \rightarrow p = \frac{-1 \pm \sqrt{5}}{2}$$

و مقدار $\frac{\sqrt{5}-1}{2} \approx 0.618$ برای p قابل قبول خواهد بود.

می‌توان نشان داد که بررسی رأسی که فرزند رأس فعلی نیست، قبل از مشخص شدن مقدار این رأس، سودی برای الگوریتم نخواهدداشت. بنابراین کافی‌ست میانگین زمان اجرا را برای بهترین الگوریتم قطعی اول‌عمق، فرض کنید الگوریتم A، حساب کنیم.

حال تعریف می‌کنیم:

$E(k) :=$ میانگین زمان اجرای A روی توزیع فوق، وقتی عمق درخت برابر k باشد.

این الگوریتم برای رأسی در عمق $k > 0$ ابتدا مقدار یکی از فرزندانش را مشخص می‌کند که این کار میانگین زمان اجرای $E(k-1)$ لازم دارد. از طرفی این فرزند به احتمال p مقدار True خواهدداشت. بنابراین الگوریتم ملزم خواهد بود مقدار فرزند دیگر را نیز مشخص کند. پس به احتمال p الگوریتم میانگن زمان $E(k-1)$ اضافه‌تر پرداخت می‌کند. پس خواهیم داشت:

$$E(k) = E(k-1) + pE(k-1) = (1+p) \times E(k-1) = (1+p)^k \times E(0) = (\frac{1+\sqrt{5}}{2})^k \approx n^{0.69}$$

برای حالت $k = 0$ هر الگوریتم قطعی باید مقدار این رأس را بررسی کند، پس $E(0) = 1$.

پس آیا الگوریتم تصادفی بالا بهینه نبوده‌است؟ خیر. در واقع کران پایین حساب شده با اندازه کافی تنگ نیست. اگر توزیع را به گونه‌ای انتخاب کنیم می‌کردیم که هیچگاه هر دو فرزند یک رأس مقدار False نداشته‌باشند، به کران بهتری می‌رسیدیم. فرض کنید توزیع را به صورت بازگشتی چنین بسازیم که اگر مقدار رأسی True بود، هر دو فرزندش False باشند و در غیر این صورت، دو مقدار متفاوت داشته‌باشند. رابطه بازگشتی برای محاسبه زمان اجرا همان رابطه الگوریتم بازگشتی خواهد بود. پس الگوریتم تصادفی که ارائه کرده‌بودیم، الگوریتم بهینه بوده‌است.

۲ تطابق بیشینه

در جلسات پیشین درباره مسئله پیدا کردن تطابق بیشینه در گراف دوبخشی صحبت شده‌است. در این جلسه صورت آنلاین این مسئله را بررسی می‌کنیم.

۱.۲ تعریف صورت مسئله

عدد n که اندازه هر بخش گراف دوبخشی را مشخص می‌کند و n رأس، که رئوس بخش اول هستند به الگوریتم داده‌می‌شود. سپس در هر یک از n مرحله بعد ابتدا یک رأس از بخش دیگر و تمام یال‌هایی که به آن وصل است به الگوریتم داده‌می‌شود و الگوریتم باید تصمیم بگیرد

از بین این یال‌ها، یالی را به تطابق اضافه کند یا خیر. در مراحل بعد نمی‌تواند یالی که قبلاً به تطابق اضافه کرده‌است را حذف کند.

۲.۲ الگوریتم قطعی

یک الگوریتم قطعی برای حل این مسئله چنین است که در هر مرحله، در صورت وجود، یالی که سر دیگرش (آن رأسی که در بخش اول است) تا به حال در تطابق ظاهر نشده‌است را به تطابق اضافه می‌کنیم. این الگوریتم در نهایت ما را به تطابق ماکسیمال M می‌رساند(چرا؟).

ادعا ۳. *ضریب رقابتی الگوریتم فوق برابر ۲ است.*

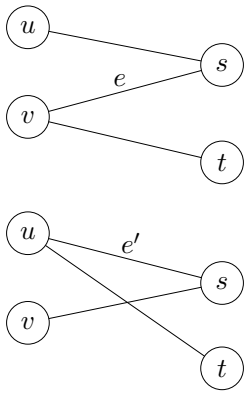
اثبات: فرض کنید OPT تطابق بیشینه در گراف ورودی باشد. حداقل یک سر از هر یال این تطابق در M آمده‌است. زیرا اگر یالی از OPT باشد که هیچ دو سرش در M نیامده‌باشد می‌توان آن یال را به M اضافه کرد. اما می‌دانیم M تطابق ماکسیمال است. پس تعداد رئوس M حداقل به اندازه یال‌های OPT خواهدبود. پس می‌توان نوشت:

$$|OPT| \leq 2|M| \rightarrow \frac{|OPT|}{2} \leq |M|$$

و ادعا ثابت می‌شود.

همچنین می‌توان نشان داد هیچ الگوریتم قطعی با ضریب رقابتی بهتر از ۲ وجود ندارد. در واقع نشان می‌دهیم دشمن می‌تواند ورودی را به گونه‌ای تعیین کند که جواب الگوریتم دلخواه قطعی نصف جواب بهینه شود.

به دو شکل زیر دقت کنید. فرض کنید رأس s از بخش دوم و یال‌های متصل به آن داده‌شود. هر الگوریتم قطعی یکی از دو یال e یا e' را به تطابق فعلی اضافه می‌کند. اگر الگوریتم قطعی یال e را انتخاب کرد دشمن در مرحله بعد رأس t را به رأس v و در غیر این صورت به u وصل می‌کند. و این الگو در مراحل بعدی نیز به همین ترتیب ادامه پیدا می‌کند. واضح است که جواب بهینه ۲ برابر جواب الگوریتم قطعی خواهدبود که با انتخاب دو یال دیگر در هر مرحله حاصل می‌شود.



۳.۲ الگوریتم تصادفی

در این بخش الگوریتم بهینه تصادفی برای این مسئله و ضریب رقابتی آن را بدون اثبات بیان می‌کنیم.

در این الگوریتم پیش از آن که مراحل ورودی گرفتن رأس‌های بخش دوم آغاز شود یک ترتیب تصادفی به رأس‌های بخش اول نسبت می‌دهیم. سپس در هر مرحله یالی را به تطابق اضافه می‌کنیم که شماره رأس آن در بخش دوم کمینه است و تا به حال در تطابق ظاهر نشده‌است. این الگوریتم ضریب رقابتی $(1 - \frac{1}{e})^{-1}$ دارد.

مراجع

[۱] ویدئوی جلسه یازدهم کلاس حل تمرین

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه حل تمرین ۱۲: الگوریتم‌های تصادفی

نگارنده: بردیا آریان فرد

در این جلسه به بررسی چند روش احتمالاتی و تصادفی و قطعی کردن آن‌ها می‌پردازیم.

مسئله Max-SAT

در ورودی یک فرمول مثل مسئله‌ی SAT داده می‌شود، می‌خواهیم یک مقداردهی از متغیرها پیدا کنیم که حداکثر تعداد ممکن از عبارت‌ها را ارضا کند.
هدف ما پیدا کردن یک الگوریتم تصادفی تقریبی برای این مسئله است، چرا که اگر یک الگوریتم دقیق پیدا کنیم، از مقایسه‌ی پاسخ و آن با تعداد عبارت‌ها می‌توان مسئله‌ی SAT را حل کنیم که این اتفاق نمی‌افتد مگر این که $\mathbf{P} = \mathbf{NP}$ باشد.
طبیعی‌ترین ایده: هر متغیر را به طور تصادفی ۰ یا ۱ مقداردهی کنیم.
ابتدا تعاریف زیر را در نظر بگیرید:
Y_i در صورتی که عبارت iاُم ارضا شده‌بود، یک و در غیر این صورت صفر است.

$$W := \sum Y_i$$

به علت خطی بودن امید ریاضی می‌توان گفت:

$$E[W] = E[\Sigma Y_i] = \Sigma E[Y_i]$$

از طرفی می‌توان کران پایین زیر را برای هر $E[Y_i]$ ارائه داد:

$$E[Y_i] \geq \frac{1}{2}$$

چرا که هر عبارت دقیقاً به ازای یک مقداردهی ارضا نمی‌شود و از طرفی به ازای حداقل یک مقداردهی ارضا می‌شود؛ برای مثال برای یک عبارت تک‌متغیره $E[Y_i] = \frac{1}{2}$. در نتیجه می‌توان گفت:

$$E[W] = E[\Sigma Y_i] = \Sigma E[Y_i] \geq \frac{N}{2}$$

که N تعداد کل عبارت‌ها است.
همچنین برای این مسئله، حالت وزن‌دار را نیز می‌توان این‌گونه در نظر گرفت که هر عبارت دارای وزن w_i باشد و هدف مسئله نیز پیدا کردن یک مقداردهی به متغیرهاست به طوری که جمع وزن عبارت‌های ارضا شده بیشینه شود. برای آن‌ها نیز می‌توان گفت:

$$W := \sum w_i Y_i \implies E[W] = E[\sum w_i Y_i] = \sum w_i E[Y_i] \geq \frac{\sum w_i}{2}$$

حال، می‌خواهیم الگوریتم فوق را به الگوریتمی قطعی با همین تقریب تبدیل کنیم. [۱]
یک ایده‌ی ساده: ابتدا یک مقدار اوّلیه (مانند تمام صفر یا هر حالت تصادفی دیگری) به متغیرها می‌دهیم. اگر بیش از نیمی از عبارت‌ها

[۱] اصطلاحاً آن را derandomize کنیم.

ارضا شدند، مقادیر را همانطور نگه می‌داریم و در غیر این صورت مقدار تمامی متغیرها را نقیض می‌کنیم. چرا که در این صورت، حداقل نصف عبارت‌ها ارضا نشده‌اند پس اگر تمامی متغیرها را نقیض کنیم، تمامی عبارت‌های ارضا نشده، که تعدادشان حداقل به اندازه‌ی نصف عبارت‌ها است، ارضا می‌شوند.

الگوریتم فوق، علاوه بر قطعی بودن، تقریب الگوریتم تصادفی مذکور را نیز حفظ کرده. حال، مسئله‌ی $\text{Max k} - \text{SAT}$ را در نظر بگیرید. این مسئله مانند مسئله‌ی قبلی است، با این تفاوت که هر عبارت شامل دقیقاً k متغیر متمایز است. برای مثال در $k = 3$، با ایده‌ی تصادفی فوق می‌توان به طور متوسط $\frac{7}{8}$ عبارت‌ها را ارضا کرد. همچنین ثابت شده‌است که در این حالت تقریبی بهتر از $\frac{7}{8}$ وجود ندارد، مگر این که $\mathbf{P} = \mathbf{NP}$. (با استفاده از قضیه‌ی PCP [۱] چند قضیه‌ی تقریب ناپذیری اثبات می‌شود.)

حال، به قطعی کردن الگوریتم این مسئله می‌پردازیم چرا که روش قبلی برای این مسئله صحیح نیست.

برای این کار، از اوّلین متغیر شروع کرده و هر متغیر را طوری قرار می‌دهیم که متغیرهای بعدی را بتوانیم طوری مقدار دهی کنیم که حداقل $\frac{7}{8}$ عبارت‌ها ارضا شوند. برای این کار، وقتی به هر x_i رسیدیم، ابتدا یک بار به آن مقدار صفر را می‌دهیم تا به فرمول Φ_0 برسیم و یک بار نیز آن را برابر یک قرار می‌دهیم تا به فرمول Φ_1 برسیم. سپس امید ریاضی تعداد عبارات ارضا شده در Φ_0 و Φ_1 در شرایطی که مقدار باقی متغیرها تصادفی انتخاب شوند را محاسبه می‌کنیم ($E[W(\Phi_0)], E[W(\Phi_1)]$) و مقدار x_i را به نحو زیر تعیین می‌کنیم:

$$x_i = \begin{cases} 0 & E[W(\Phi_0)] \geq \frac{7}{8} \\ 1 & E[W(\Phi_1)] \geq \frac{7}{8} > E[W(\Phi_0)] \end{cases}$$

با بررسی روابط امید ریاضی شرطی به نحو زیر می‌توان ثابت کرد که از بین دو مقدار فوق حداقل یکی از آن‌ها حداقل $\frac{7}{8}$ است:

$$\frac{7}{8} \leq E[W] = E[W|x_i = 1]Pr[x_i = 1] + E[W|x_i = 0]Pr[x_i = 0]$$

$$= \frac{1}{2}(E[W|x_i = 1] + E[W|x_i = 0]) \leq \max(E[W|x_i = 1], E[W|x_i = 0])$$

در نتیجه در الگوریتم فوق، از آن جا که در هر مرحله میانگین تعداد عبارات ارضا شده به ازای تمامی مقداردهی‌های متغیرهای باقی‌مانده حداقل $\frac{7}{8}$ است، یک مقداردهی از باقی متغیرها نیز وجود دارد که تعداد عبارات ارضا شده در آن حداقل $\frac{7}{8}$ است. پس وقتی این الگوریتم به مرحله‌ی آخر برسد، شرایط مسئله محقق می‌شود. حال، به مسئله‌ی $\text{Max} - \text{SAT}$ برمی‌گردیم. سعی می‌کنیم برای این مسئله یک راه حل با تقریب بهتر از $\frac{1}{2}$ پیدا کنیم. فرض کنید هر متغیر را به احتمال $p \neq \frac{1}{2}$ یک و به احتمال $1 - p$ صفر قرار می‌دهیم. فرض کنید $p > \frac{1}{2}$ و فرض کنید هیچ عبارتی وجود ندارد که به شکل $(\overline{x_i})$ باشد. (عبارتی به شکل $(\overline{x_i} \vee \overline{x_i})$ مشکلی ندارد.)

ادعا: در این حالت می‌توان به امید ریاضی بیش از $\frac{1}{2}$ دست یافت.

احتمال ارضا شدن یک عبارت، حداقل $\min\{p, 1 - p^2\}$ است. چرا که احتمال ارضا شدن یک عبارت در واقع برابر یک منهای احتمال ارضا نشدن آن است، پس اگر در این عبارت، a بار نقیض متغیرها و b بار خود متغیرها ظاهر شوند، می‌توان گفت:

$$Pr\{Y_i = 1\} = 1 - p^a(1 - p)^b \geq 1 - p^{a+b} \geq 1 - p^2$$

حال، برای رسیدن به بیشترین مقدار می‌توانیم قرار دهیم $p = 1 - p^2$ که حاصل این معادله $p \simeq 0.618$. پس، این الگوریتم دارای تقریب 0.618 است.

برای این الگوریتم ما فرض کردیم که هیچ عبارتی به شکل $(\overline{x_i})$ وجود ندارد ولی این فرض را می‌توان حذف کرد و به جای آن، این فرض را اضافه کرد که برای هر متغیر x_i در حالت وزن‌دار، وزن عبارت (و در حالت عادی، تعداد تکرار) x_i بیش از عبارت $\overline{x_i}$ باشد. که تضمین این شرط ممکن است؛ چرا که به ازای هر متغیری که در این شرط صدق نمی‌کرد، می‌توانیم جای آن متغیر و نقیضش را عوض کنیم تا شرایط برقرار شود.

نکته‌ی دیگر این است که هدف ما در واقع رسیدن به تقریبی است که کسر مناسبی از جواب بهینه (opt) باشد، نه مجموع وزن کل عبارات. در واقع، از کران بدیهی $opt \leq \sum w_i$ استفاده کردیم. یک کران بهتر، به نحو زیر است:

$$opt \leq \sum w_i - \sum v_i$$

که v_i وزن عبارت $\overline{x_i}$ است. (در صورت عدم وجود، صفر در نظر گرفته می‌شود.)
می‌توان اثبات کرد که با الگوریتم فوق، به $0.618opt$ می‌رسیم.

مراجع

[۱] جزوه‌ی جلسه‌ی ۲۰

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس حل تمرین: مرتضی علیمی

[بهار ۹۹]

جلسه حل تمرین ۱۳: کران مارکف، چبیشف و چرنوف نگارنده: امین کشیری

در این جلسه مثال‌های دیگری از کران‌های بررسی شده در جلسه‌ی ۴ و ۵ را خواهیم دید.

مثال: فرض کنید یک تاس را n بار بیندازیم. تعریف کنید:

$X =$ تعداد دفعات آمدن ۶

با استفاده از کران‌های مارکف [1] ، چبیشف [2] و چرنوف [3] کران بالایی برای $Pr[X \geq \frac{n}{4}]$ ارائه دهید.

مارکف:

$$Pr[X \geq \tfrac{n}{4}] \leq \frac{\mu}{\frac{n}{4}} = \frac{\frac{n}{6}}{\frac{n}{4}} = \frac{2}{3} = O(1)$$

چبیشف:

$$Pr[\mid X - \tfrac{n}{6} \mid \geq \tfrac{n}{12}] \leq Pr[X \geq \tfrac{n}{4}] \leq \frac{\sigma^2}{(\frac{n}{12})^2} = \frac{\frac{5n}{36}}{(\frac{n}{12})^2} = O(\tfrac{1}{n})$$

چرنوف:

$$Pr[X \geq (1+\varepsilon)\mu] = Pr[X \geq (1+\tfrac{1}{2})\tfrac{n}{6}] \leq Pr[X \geq \tfrac{n}{4}] \leq e^{-\frac{\mu\varepsilon^2}{3}} = e^{-\frac{\frac{n}{6}\cdot\frac{1}{4}}{3}} = O(e^{-n})$$

$\boxtimes$

مثال: در مسئله‌ی توپ‌ها و سطل‌ها [4] حداقل تعداد توپ‌هایی را پیدا کنید که باعث شود بار هر سطر با احتمال بالا [5] بین نصف و دوبرابر میانگین باشد. فرض کنید تعداد توپ‌ها m و تعداد سطل n باشد.

تعریف کنید:

$X =$ تعداد توپ‌های سطل اول

قرار دهید $m = 24n\ln n$. در این صورت $\mu = 24\ln n$. حال داریم:

[1] Markov
[2] Chebyshev
[3] Chernoff
[4] Balls ans Bins
[5] With High Probability

$$Pr[X \geq 2\mu] \leq e^{-\frac{\mu\varepsilon^2}{2+\varepsilon}} = e^{-8\ln(n)} = n^{-8} \leq n^{-3}$$

$$Pr[X \leq \tfrac{\mu}{2}] \leq e^{-\frac{\mu\varepsilon^2}{2}} = e^{-3\ln(n)} = n^{-3}$$

پس با احتمال بالا تعداد توپ‌های سطر اول (و مشابهاً تک تک سطل‌ها) بین نصف و دوبرابر میانگینش است. حال داریم:

$$Pr[\text{تعداد توپ‌های یکی از سطل‌ها کمتر از نصف یا بیشتر از دوبرابر میانگین باشد}] \leq n \cdot Pr[(\tfrac{\mu}{2} \geq X) \cup (X \geq 2\mu)]$$

$$\leq n \cdot 2n^{-3} = \tfrac{2}{n^2}$$

⊠

مثال: فرض کنید n گوی داریم که در یک کیسه هستند و np تا از آن‌ها آبی و بقیه قرمز هستند (در واقع احتمال آبی بودن یک گوی p است). می‌خواهیم با یک نمونه گیری p را تخمین بزنیم. تعداد k توپ را به تصادف بیرون می‌کشیم و نسبت تعداد آبی‌ها به کل توپ‌ها را محاسبه می‌کنیم و آن را به عنوان تخمین خود بیرون می‌دهیم. فرض کنید می‌خواهیم p را با احتمال $1-\delta$ و با ضریب تقریب $1 \pm \varepsilon$ پیدا کنیم. تعداد نمونه‌های لازم را با استفاده از کران چبیشف و چرنوف بیابید.

درواقع، ما می‌خواهیم به ازای هر ε و δ دلخواه، k مناسبی را پیدا کنیم که تخمینی که می‌زنیم شرایط خواسته شده در مسئله را داشته باشد. در حالت کلی چنین الگوریتم‌هایی در یک دسته‌بندی خاص [۶] قرار می‌گیرند که شامل الگوریتم‌هایی هستند که در زمان چند جمله‌ای تخمینی با ضریب تقریب کافی و احتمال خطایی کم از مقدار خروجی واقعی مسئله به دست می‌آورند (درواقع یعنی با احتمال بالا حداکثر با یک ضریب ϵ از جواب فاصله دارند). حال k باید چقدر باشد؟ به کمک نامساوی چبیشف داریم:

$$Pr[\mid X - kp \mid \geq \varepsilon kp] \leq \tfrac{\sigma^2}{\varepsilon^2 p^2 k^2} = \frac{kp(1-p)}{\varepsilon^2 p^2 k^2} \leq \delta \Rightarrow \tfrac{1}{\varepsilon^2}\tfrac{1-p}{p}\tfrac{1}{k} \leq \delta \Rightarrow \tfrac{1}{\varepsilon^2}\tfrac{1}{p}\tfrac{1}{\delta} \leq k$$

مشابهاً به کمک نامساوی چرنوف:

$$Pr[X \geq (1+\varepsilon)kp] \leq e^{-\frac{kp\varepsilon}{3}} \leq \delta \Rightarrow k = \Omega(\tfrac{1}{\varepsilon^2}\tfrac{1}{p}\ln(\tfrac{1}{\delta}))$$

نکته‌ی جالبی که در این تحلیل وجود دارد وابستگی محاسبات ما به p است، اما ما p را در اختیار نداریم. می‌توانیم به این صورت توجیه کنیم که فرض کنیم p را به ما داده‌اند و ما بیشتر به دنبال تایید درستی این ادعا هستیم تا محاسبه واقعی مقدار p . اما در واقع راه بهتری نیز وجود دارد. می‌توانیم آنقدر نمونه بگیریم تا تعداد موفقیت‌های ما به $\frac{1}{\varepsilon^2}\ln(\frac{1}{\delta})$ برسد. در این صورت، یعنی ما حدود $\frac{1}{\varepsilon^2}\frac{1}{p}\ln(\frac{1}{\delta})$ نمونه گرفته‌ایم و این یعنی بدون داشتن p تقریباً تعداد مناسبی نمونه گرفته‌ایم!

نکته‌ی دیگری که قابل توجه است، حضور $\frac{1}{p}$ در تحلیل‌های ماست. و البته قابل انتظار نیز بود، زیرا به طور میانگین باید حداقل $\frac{1}{p}$ توپ را ببینیم تا حداقل یک توپ آبی دیده باشیم (زیرا تعداد توپ‌های لازم برای دیدن یک توپ آبی یک متغییر هندسی با پارامتر p است).

این تحلیل به ما می‌گوید وقتی احتمال یک اتفاق بسیار پایین باشد، برای یک تخمین خوب باید تعداد نمونه‌ها را خیلی زیاد کنیم، و یعنی یک نمونه‌ی کوچک به ما تخمین دقیقی نمی‌دهد. این مشکل را «مشکل اتفاقات نادر [۷] » می‌نامند، که البته روش‌هایی نیز برای حل آن وجود دارد که ما در اینجا بررسی نمی‌کنیم.

⊠

در جلسات قبل، دیدیم که نامساوی چرنوف معمولاً کران قوی‌تری نسبت به نامساوی چبیشف به ما می‌دهد. اما از طرفی نامساوی

[6] FPRAS
[7] Rare Event

چبیشف به فرض‌های کمتری روی وروی مسئله احتیاج دارد و بر خلاف نامساوی چرنوف که به استقلال همه‌ی متغییرها احتیاج داشت، نامساوی چبیشف تنها به استقلال دو به دوی متغییرها احتیاج دارد. مثال زیر کاربردی ازین نکته را نشان می‌دهد.

مثال: فرض کنید یک الگوریتم مونت‌ـ کارلو [8] داریم که با احتمال $\frac{1}{4}$ جواب غلط می‌دهد. حالا برای کاهش‌دادن احتمال خطا الگوریتم را چندین بار اجرا می‌کنیم و تنها درصورتی که در یکی از اجراها جواب مثبت بود، جواب مثبت را به عنوان خروجی اعلام می‌کنیم. نکته‌ای که در همچین مسائلی وجود دارد، این است که برای هر اجرا نیاز به تعدادی بیت تصادفی داریم. این تعداد را k می‌نامیم. در صورت داشتن k بیت تصادفی یک بار می‌توانیم الگوریتم را اجرا کنیم، در صورت داشتن $2k$ بیت تصادفی دوبار و ... . با هر بار اجرای الگوریتم نیز احتمال خطا ضرب در $\frac{1}{4}$ می‌شود. حال فرض کنید تولید بیت تصادفی نیز هزینه دارد و ما تنها می‌توانیم $2k$ بیت تصادفی تولید کنیم. کران بالایی که می‌توانیم برای خطا پیدا کنیم $\frac{1}{4}$ است که برای بسیاری از کاربردها مناسب نیست. اگر بتوانیم به کمک این k بیت، تعداد بیشتری بیت تصادفی بسازیم می‌توانیم احتمال خطا را کاهش دهیم. ایده‌ای که از آن استفاده می‌کنیم به این صورت است که اگر دو بیت تصافی مثل x و y داشته باشیم، x ، y و $x \oplus y$ دو به دو مستقل هستند (دقت کنید هر سه با هم مستقل نیستند). همانطور که در بحث درهم‌سازی در ساختمان‌داده دیده بودیم، تابع‌هایی وجود داشتند که این استقلال دو به دوی متغییرها [9] را برای ما فراهم می‌کردند. اگر داشته باشیم $h(x) = (ax + b) \mod p$ و $a, b \in \mathbb{Z}_p$ خواهیم داشت: $Pr[h(x) = h(y)] = \frac{1}{p}$. حال به ازای هر $x = 0, 1, ...p - 1$ ، $h(x)$ را محاسبه می‌کنیم. طبق تعریف بالا تمام این عبارات دو به دو مستقل اند. با این روش می‌توانیم p عدد تصادفی بسازیم که دو به دو مستقل هستند و حال به کمک آن ها kp بیت تصادفی می‌سازیم. دقت کنید دیگر نمی‌توانیم بگوییم احتمال شکست $(\frac{1}{4})^p$ است زیرا انتخاب آن‌ها تصادفی نبوده و تنها دو به دو با هم مستقل هستند. داریم:

$$E[\text{ تعداد موفقیت‌ها }] = \frac{p}{2} \Rightarrow Pr[\text{ شکست }] = Pr[\text{ موفقیت } = 0\,]$$

$$Var(\text{تعداد موفقیت‌ها}) = \frac{p}{4} \Rightarrow \sigma = \frac{\sqrt{p}}{2}$$

پس برای این که تعداد موفقیت‌ها صفر شود، نیاز داریم تقریباً به اندازه‌ی یک ضریب $\sqrt{p}$ از انحراف از معیار دور شویم. حال به کمک چبیشف داریم:

$$Pr[\text{ شکست }] \leq \frac{1}{\sqrt{p}^2} = \frac{1}{p}$$

و می‌بینیم توانستیم احتمال خطا به صورت قابل توجهی کاهش دهیم (بدون استفاده از بیت‌های تصادفی بیشتری).

⊠

[8] Monte Carlo

[9] Pairwise Independence

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی

[بهار ۹۹]

جلسه ۱۴ حل تمرین: برنامه‌ریزی خطی

نگارنده: شایان طاهری جم

در این جلسه قصد داریم تعدادی مسئله را به صورت برنامه‌ریزی خطی مدل کنیم.

۱ مسئله جریان بیشینه

می‌خواهیم مسئله‌ی جریان بیشینه را به صورت برنامه‌ریزی خطی مدل کنیم.
f_e : جریانی که از یال e می‌گذرد.

جریان هر یال مانند e باید عددی بین 0 و ظرفیت یال (c_e) باشد. پس شرط‌های زیر لازم اند:

$$\forall e \quad 0 \leq f_e \leq c_e$$

به ازای هر رأس بجز s و t باید مجموع جریان‌های ورودی و خروجی به آن رأس 0 شود. پس شرط‌های زیر را باید داشته باشم.

$$\forall v \neq s,t \sum_{e \in \delta^-(v)} f_e - \sum_{e \in \delta^+(v)} f_e = 0$$

تابع هدف ما در واقع مقدار جریان است که باید بیشینه شود. مقدار جریان برابر تابع زیر است، پس تابع هدف هم به همین صورت است:

$$\max(\sum_{e \in \delta^-(s)} f_e - \sum_{e \in \delta^+(s)} f_e) = ?$$

پس مسئله جریان بیشینه را به صورت برنامه‌ریزی خطی مدل کردیم.
حال فرض کنید می‌خواهیم جریان با کمترین هزینه را به صورت برنامه‌ریزی خطی مدل کنیم.
شرط‌های این مسئله هم مانند مسئله جریان بیشینه است و فقط تابع هدف آن به صورت زیر تغییر می‌کند:

$$\min \sum_{e \in E} k_e f_e$$

و مقدار جریان هم با توجه به تساوی زیر تعیین می‌شود:

$$\sum_{e \in \delta^-(s)} f_e - \sum_{e \in \delta^+(s)} f_e = d$$

۲ مسئله برازش خط

ورودی: n نقطه مانند (x_i, y_i)
خروجی : a, b به طوری که $ax_i + b = y_i$ ولی ممکن است این اتفاق رخ ندهد بنابراین خطی را می‌خواهیم که تا جای ممکن به این

خاصیت نزدیک باشد .
مسئله‌ای شبیه به این در آزمایشگاه فیزیک ۱ داشته‌ایم به نام کمترین مربعات که در آن می‌خواستیم a و b را طوری پیدا کنیم که مقدار $\sum |y_i - ax_i + b|^2$ کمینه شود.

در اینجا می‌خواهیم a و b را طوری پیدا کنیم که مقدار $\sum |y_i - ax_i + b|$ کمینه شود.
در اینجا باید تابع هدف را طور خوبی قرار دهیم که قدرمطلق آن از بین برود.
به ازای هر عبارت داخل قدر مطلق متغیر t_i را اضافه می‌کنیم و شرط‌های زیر را اضافه می‌کنیم.

$$\forall i \quad t_i \geq y_i - ax_i + b$$

$$\forall i \quad t_i \geq -(y_i - ax_i + b)$$

حال باید مقدار $\sum t_i$ کمینه شود پس این عبارت تابع هدف ما است.

پس این مسئله را نیز به صورت برنامه‌ریزی خطی مدل کردیم

۳ مسئله جداسازی نقاط

تعدادی نقطه در صفحه داریم که بعضی از آن‌ها آبی و بقیه قرمز هستند. می‌خواهیم نقاط آبی و قرمز را با خطی از هم جدا کنیم.
منظور از جدا کردن نقاط با خط چیست؟
فرض کنید نقاط آبی ما باشند :

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

و نقاط قرمز ما باشند:

$$(x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_n, y'_n)$$

و ما می‌خواهیم نقاط قرمز زیر خط و نقاط آبی بالای خط باشند.
باید a و b را طوری بیابیم که:

$$\forall i \quad y_i > ax_i + b$$

$$\forall i \quad y'_i < ax'_i + b$$

(این حالت مسئله داخل فضای 2 بعدی است. می‌توان مسئله را در n بعد نیز تعریف کرد.)
یکی از مشکلاتی که این مسئله دارد این است که ما در برنامه‌ریزی خطی بزرگتر اکید ($>$) و یا کوچک تر اکید ($<$) نداریم.
متغیرهای t_i و t'_i را به اینصورت تعریف می‌کنیم:

$$\forall i \quad t_i = y_i - (ax_i + b)$$

$$\forall i \quad t_i = ax_i + b - y_i$$

و می‌خواهیم a و b را طوری بیابیم که مقادیر t_i ها و t'_i ها مثبت باشد پس باید کمینه مقدار t_i و t'_i ها را حساب کنیم اگر مثبت بود مسئله شدنی است.
پس باید این عبارت را حساب کنیم:

$$\max \min\{\min_{0<i\leq n} t_i, \min_{0<i\leq n} t'_i\}$$

ولی تابع هدف ما پیچیده شده است.
با اضافه کردن تعدادی شرط تابع هدف بهتری می‌سازیم:

$$\forall i \quad t_i \geq \delta$$

$$\forall i \quad t'_i \geq \delta$$

و تابع هدف ما می‌شود $\max(\delta)$. حال اگر دقت کنید می‌توانستیم به کمک δ از اول مسئله را طور خوبی بنویسیم و نیازی به t_i ها و t'_i ها نباشد:

$$\forall i \quad y_i \geq ax_i + b + \delta$$

$$\forall i \quad y'_i \leq ax'_i + b + \delta$$

و تابع هدف هم کماکان $\max(\delta)$ است.
پس این مسئله را نیز توانستیم به صورت برنامه‌ریزی خطی مدل کنیم.
حال صورت‌های پیچیده تری از این مسئله را هم می‌توانیم با همین ایده مدل کنیم. مثلاً فرض کنید می‌خواهیم نقاط را به کمک سهمی‌ای از هم جدا کنیم.
مسئله اینگونه مدل می‌شود:

$$\forall i \quad y_i > ax_i^2 + bx_i + c$$

$$\forall i \quad y'_i < ax_i'^2 + bx'_i + c$$

توجه داشته باشید که اعدادی مانند x_i^2 ورودی مسئله هستند و متغیرهای ما a و b و c هستند پس کماکان مسئله خطی است.

۴ مسئله سود بیشینه شرکت

شرکتی می‌خواهد بیش‌ترین سود ممکن را ببرد.
این شرکت دو نوع محصول دارد.
محصول نوع 1 : $\frac{1}{4}$ ساعت زمان برای تولید نیاز دارد، $\frac{1}{8}$ ساعت زمان برای تست نیاز دارد، هزینه مواد خام آن 1.2$ است، و قیمت آن 9 $ است.

محصول نوع 2 : $\frac{1}{3}$ ساعت زمان برای تولید نیاز دارد، $\frac{1}{3}$ ساعت زمان برای تست نیاز دارد، هزینه مواد خام آن 0.9$ است، و قیمت آن 8 $ است.
این شرکت محدودیت‌هایی هم دارد مانند:

هرروز 90 ساعت زمان تولید دارد و 80 ساعت زمان تست.
فرض کنید x_1 و x_2 به ترتیب تعداد محصولات نوع اول و نوع دوم باشد که می‌خواهیم تولید کنیم.
در این مسئله می‌خواهیم سود خود را بیشینه کنیم که سود برابر است با :

$$9x_1 + 8x_2 - 1.2x_1 - 0.9x_2$$

پس تابع هدف ما می‌شود:

$$\max(9x_1 + 8x_2 - 1.2x_1 - 0.9x_2)$$

و محدودیت‌های ما عبارت اند از :

$$\frac{1}{4}x_1 + \frac{1}{3}x_2 \leq 90$$

$$\frac{1}{8}x_1 + \frac{1}{3}x_2 \leq 80$$

$$x_1, x_2 \geq 0$$

این می‌تواند مدلی برای مسئله باشد.
حال فرضی به مسئله اضافه می‌کنیم:
فرض کنید می‌توانیم روزی تا 50 ساعت به زمان تولید اضافه کنیم که هر ساعت 7 دلار هزینه می‌برد.
با تغییراتی جزئی که در زیر مشاهده می‌کنید می‌توان این فرض را نیز اضافه کرد:
تابع هدف ما می‌شود:

$$\max(9x_1 + 8x_2 - 1.2x_1 - 0.9x_2 - 7t)$$

و محدودیت‌های ما عبارت اند از :

$$\frac{1}{4}x_1 + \frac{1}{3}x_2 \leq 90 + t$$

$$\frac{1}{8}x_1 + \frac{1}{3}x_2 \leq 80$$

$$x_1, x_2 \geq 0$$

$$0 \leq t \leq 50$$

می‌توان باز هم یک فرض دیگر به مسئله اضافه کرد:
فرض کنید اگر حداقل از 300 دلار ماده خام استفاده کنیم، می‌توانیم 10 درصد تخفیف بگیریم.
برای مدل کردن این فرض می‌توانیم به راحتی مسئله را به دو حالت تقسیم کنیم و هر حالت را جداگانه مدل کنیم.
حالت اول اینکه کم‌تر از 300 دلار مواد خام خریداری کنیم.
حالت دوم اینکه بیش از 300 دلار مواد اولیه خریداری کنیم.
هر دو حالت را می‌توانیم به راحتی مدل کنیم که آن را بر عهده خودتان می‌گذاریم.

۵ مسئله کارخانه‌ی تولید میله

فرض کنید کارخانه‌ای میله‌های به طول 3 متر تولید می‌کند. شرکتی به این کارخانه سفارشاتی داده :
97 میله 135 سانتی متری
610 میله 108 سانتی متری
395 میله 93 سانتی متری
211 میله 42 سانتی متری
کمترین تعداد میله 3 متری برای سرویس دادن به همه سفارشات چقدر است؟

اینجا نکته این است که چه چیز را متغیر بگیریم ؟
ابتدا بررسی می‌کنیم که یک میله را به چند طریق می‌توانیم به صورت بهینه برش بزنیم؟
1 : دوتا 135 سانتی.
2 : یک 135 سانتی، یک 108 سانتی، یک 82 سانتی.
3 : یک 135 سانتی، یک 93 سانتی، یک 42 سانتی.
4 : یک 135 سانتی، سه‌تا 42 سانتی.

5 : دوتا 108 سانتی، دوتا 42 سانتی.

.

.

.

(در واقع 12 حالت دارد که همه‌ی آن‌ها را نمی‌نویسیم)

حال x_i را تعریف می‌کنیم تعداد میله‌هایی که از نوع i ام بریده می‌شود.
تابع هدف ما باید کمترین تعداد میله‌ها باشد پس تابع هدف می‌شود:

$$\min \sum x_i$$

و محدودیت‌های ما باید به گونه‌ای باشد که سفارشات سرویس دهی شوند. یعنی مثلاً باید مجموعاً حداقل 97 میله 135 سانتی تولید شود پس:

$$2x_1 + x_2 + x_3 + x_4 \geq 97$$

چون در برش اول 2 تا میله 135 سانتی تولید می‌شود و در برش‌های دوم تا چهارم هر کدام یک میله 135 سانتی تولید می‌کنند پس تعداد میله‌های 135 سانتی تولید شده برابر می‌شود با $2x_1 + x_2 + x_3 + x_4$ که باید حداق 97 باشد.
این محدودیت‌ها را برای دیگر سفارشات هم به صورت مشابه اعمال می‌کنیم.

تمرین: فرض کنید Y متغیر تصادفی‌ای باشد که مقدار آن می‌تواند یکی از n مقدار $a_1, a_2, \ldots, a_n$ باشد.
فرض کنید ما می‌دانیم این متغیر تصادفی یا توزیع احتمال p به صورت زیر دارد:

$$\Pr(Y = a_i) = p_i$$

یا توزیع احتمال q به صورت زیر دارد:

$$\Pr(Y = a_i) = q_i$$

می‌خواهیم با یک مشاهده از این متغیر تصادفی حدس بزنیم که توزیع آن p است یا q. عملکرد ما اینگونه است که به ازای هر پیشامد از Y مانند a_i یک عدد مانند x_i بین 0 و 1 انتخاب می‌کنیم و در صورت a_i بودن پیشامد ما به احتمال x_i توزیع p را انتخاب می‌کنیم و تبعاً به احتمال $1 - p$ توزیع q را انتخاب می‌کنیم.
حال ما می‌خواهیم x_i ها را طوری تعیین کنیم که احتمال این که ما بگوییم توزیع p است در حالی که توزیع q باشد بیشتر از β نباشد که β عدد مثبت و کوچکی است (حدوداً 0.05) و می‌خواهیم احتمال این که ما بگوییم توزیع p است و در حقیقت توزیع p باشد بیشینه شود.
این بیشینه سازی را به صورت برنامه‌ریزی خطی مدل کنید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس: مرتضی علیمی
[بهار ۹۹]

جلسه حل تمرین ۱۵: برنامه‌ریزی خطی ۳ نگارنده: احسان شریفیان

در این جلسه تعدادی مثال در زمینه‌ی برنامه‌ریزی خطی[1] را بررسی خواهیم کرد.

مثال ۱: یک جواب شدنی به تو یک جواب بهینه می‌دهد!

همواره یافتن یک جواب شدنی برای مساله‌ی برنامه‌ریزی خطی، ساده‌تر از یافتن یک جواب بهینه برای آن است، چرا که با داشتن جواب بهینه ما یک جواب شدنی برای مساله‌ی برنامه‌ریزی خطی داریم که همان جواب بهینه است. پس الگوریتمی که جواب بهینه را بیابد عملا یک جواب شدنی مساله برنامه‌ریزی خطی را هم یافته است. حال می‌خواهیم نشان دهیم برعکس این اتفاق نیز وجود دارد. به بیان دقیق‌تر می‌خواهیم گزاره‌ی زیر را ثابت کنیم:
اگر الگوریتمی داشته باشیم که برای یک LP یک جواب شدنی[2] پیدا کند، با استفاده از آن می‌توانیم برای هر LP جواب بهینه را پیدا کنیم.

پاسخ

با توجه به دوگان مساله و قضیه‌ی دوگانی قوی، می‌توان این کار را انجام داد. فرض کنید الگوریتمی که یک جواب شدنی برای یک مساله‌ی LP می‌یابد، $\mathcal{A}$ نام داشته باشد، هم‌چنین مساله‌ای که می‌خواهیم جواب بهینه‌ی آن را بیابیم یک مساله‌ی متعارف به فرم

$$\begin{cases} \min \quad c^T x \\ Ax \geq b \\ x \geq 0 \end{cases}$$

باشد که x یک بردار $n \times 1$، A یک ماتریس $n \times m$ و b یک بردار $m \times 1$ باشد. دوگان این مساله‌ی برنامه ریزی خطی فرمی به صورت زیر دارد

$$\begin{cases} \max \quad b^T y \\ A^T y \leq c \\ y \geq 0 \end{cases}$$

که در آن y یک بردار $m \times 1$ است. طبق قضیه‌ی دوگانی قوی، هر گاه یک مساله‌ی برنامه‌ریزی خطی و دوگان آن فضای شدنی غیر تهی داشته باشند، آنگاه هر دو مساله دارای جواب بهینه‌ی متناهی هستند که مقدار بهینه دو مساله یکسان است، یعنی x^* در فضای شدنی مساله‌ی اصلی و y^* در فضای شدنی مساله‌ی دوگان وجود دارد که $c^T x^* = b^T y^*$. در نتیجه ما می‌توانیم یک مساله‌ی برنامه‌ریزی خطی جدید تشکیل دهیم که یک نقطه‌ی شدنی آن، یک جواب بهینه برای مساله‌ی اصلی باشد. این مساله‌ی برنامه‌ریزی خطی جدید را با توجه به قضیه‌ی بیان شده به صورت زیر می‌سازیم.

[1] linear programming
[2] feasilbe

متغیر مساله‌ی جدید بردار $[x \;\; y]^T$ است.(یعنی در این مساله ما $n+m$ متغیر داریم)

$$\begin{cases} \min \;\; c^T x \\ Ax \geq b \\ x \geq 0 \\ A^T y \leq c \\ y \geq 0 \\ y^T b = c^T x \end{cases}$$

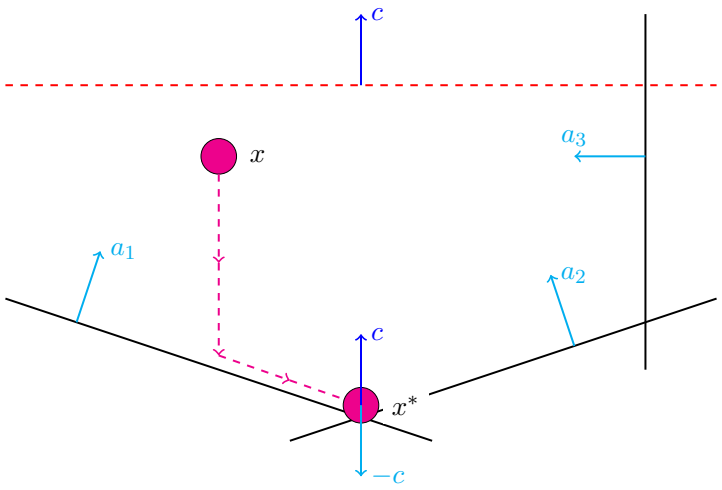
سپس الگوریتم $\mathcal{A}$ را روی آن اجرا می‌کنیم و در صورت وجود یک جواب شدنی برای آن به دست می‌آوریم. جواب شدنی باید در قیود مساله صدق کند و ما قیود مساله را بر اساس قضیه‌ی دوگانی قوی ساخته‌ایم. بنابراین اگر یک جواب شدنی مانند $[x^* \;\; y^*]$ برای آن یافت شده باشد، این بردار حاصل در همه‌ی قیود صدق کرده و در نتیجه در شرایط قضیه‌ی دوگانی قوی صدق می‌کنند و بنابراین x^* یک جواب بهینه برای مساله‌ی اصلی و y^* یک جواب بهینه برای مساله‌ی دوگان می‌باشد. یک نکته‌ی جالب این است که ما به وسیله‌ی الگوریتم $\mathcal{A}$ توانستیم هم‌زمان هر دو مساله‌ی اصلی و دوگان را حل کنیم و جواب بهینه را برای آن بیابیم.

مثال ۲: شهود فیزیکی برای مساله‌ی دوگان و قضیه‌ی لنگی مکمل

یک مساله‌ی برنامه‌ریزی خطی به فرم زیر در نظر بگیرید.

$$\begin{cases} \min \;\; c^T x \\ Ax \geq b \end{cases}$$

این صورت‌بندی نیز از آن حالاتی است که می‌توان همه‌ی فرم‌های مساله‌ی برنامه‌ریزی خطی را به صورت آن درآورد. برای اینکه بتوانیم ترسیم داشته باشیم فرض کنید مساله در حالت دوبعدی باشد اما این توضیحات در ابعاد بالاتر نیز صادق است. در این حالت هر یک از قیود $a_i^T x \geq b_i$ یک نیم‌صفحه را مشخص می‌کنند. همچنین فرض کنید که بردار گردایان تابع هدف یعنی c در جهت عمودی صفحه باشد (محورها را طوری چرخانده‌ایم که این اتفاق بیفتد) و ما جهت جاذبه را منفی جهت گرادیان یعنی $-c$ می‌گیریم. از تحلیل مساله‌ی برنامه‌ریزی خطی می‌دانیم که هر چقدر در جهت جاذبه (منفی گرادیان تابع هدف) حرکت کنیم مقدار تابع هدف کاهش می‌یابد. این بیان مانند این است که شما در این صفحه‌ی دوبعدی، هر یک از شروط $a_i^T x \geq b_i$ را مانند یک دیوار و متغیر اصلی مساله یعنی x را مانند یک گوی که در ابتدا در یک نقطه از فضای شدنی مساله رها شده است، ببینید. این گوی به خاطر برقراری جاذبه، در جهت جاذبه حرکت می‌کند تا در نهایت بین چند دیوار به حالت تعادل برسد. شکل زیر را در نظر بگیرید.



نقطه‌ای که در آن گوی به حالت تعادل می‌رسد همان نقطه‌ی بهینه برای مساله‌ی اصلی است. اما این نقطه از لحاظ فیزیکی نکات جالبی دارد که به مساله‌ی دوگان و قضیه‌ی لنگی مکمل مرتبط است.

۱. دقت داریم که نقطه‌ی تعادل (بهینه) محل برخورد چند دیوار است و برای آن دیوارها قیود $a_i^T x \geq b_i$ به حالت تساوی درآمده‌اند (که به چنین قیدهایی قیدهای فعال می‌گویند). به عنوان مثال در شکل ترسیم شده، قیود $a_1^T x \geq b_1$ و $a_2^T x \geq b_2$ در نقطه‌ی تعادل (که آن را x^* می‌نامیم) به صورت $a_1^T x^* = b_1$ و $a_2^T x^* = b_2$ درآمده‌اند.

۲. قیدهای متناظرِ دیوارهایی که نقطه‌ی تعادل با آن‌ها برخوردی ندارد، همگی به صورت اکید برقرار هستند. به عنوان مثال در شکل ترسیم شده، قید $a_3^T x \geq b_3$ در نقطه‌ی تعادل به صورت $a_3^T x^* > b_3$ برقرار است.

۳. فرض کنید جسمِ رها شده در صفحه وزنی معادل $-c$ داشته باشد. در این صورت هنگامی که جسم در نقطه‌ی x^* به تعادل می‌رسد، باید از طرف دیوارهایی که جسم با آن‌ها برخورد دارد (قیود فعال) نیرویی به جسم وارد شود که وزن آن را خنثی کند. هم‌چنین دیوارهایی که با جسم تماس در نقطه‌ی تعادل تماس ندارند به جسم نیرویی وارد نمی‌کنند. در ضمن دقت داریم که هر دیوار می‌تواند در راستای عمود بر سطح خود به جسم نیرو وارد کند و در نتیجه یک ترکیب خطی مثبت از بردارهای عمود بر دیوارهایی که با جسم در نقطه‌ی تعادل تماس دارند باید وزن جسم را خنثی کند. به بیان ریاضی باید اعداد p_i وجود داشته باشند که

$$\sum_{i=1}^{n} p_i a_i = A^T p = c$$

که p_iهای متناظر با قیود غیر فعال (دیوارهایی که با جسم در نقطه‌ی تعادل یا همان نقطه‌ی بهینه تماس ندارند) صفر است و می‌توان رابطه‌ی بالا را به صورت زیر نوشت

$$\sum_{\text{قیود فعال}} p_i a_i = c$$

در شکل صفحه‌ی قبل این بیان معادل این است که دیوار متناظر با a_3 نیرویی به جسم وارد نمی‌کند و ضریب متناظر با آن یعنی p_3 برابر با صفر است. هم‌چنین برآیند دو نیروی دیگر وارده از طرف دیوارهای متناظر با a_1 و a_2 باید وزن جسم را خنثی کند که این به معنای $p_1 a_1 + p_2 a_2 = c$ است.

۴. بیان‌های قسمت قبل شهودی از قضایای دوگانی قوی و لنگی مکمل است. فرض کنید مساله‌ی دوگان مساله‌ی اصلی تعریف شده در این قسمت به صورت زیر باشد

$$\begin{cases} \max \quad b^T p \\ A^T p = c \\ p \geq 0 \end{cases}$$

این متغیرهای p_i همان ضرایب ترکیب خطی a_iها هستند که قرار است وزن جسم را خنثی کنند. قضیه‌ی لنگی مکمل بیان می‌کند که رابطه‌ی $p_i(b_i - a_i^T x^*) = 0$ برای هر $i = 1, \dots, n$ برقرار است. برای قیودی که فعال نیستند یا همان دیوارهایی که به جسم نیرو وارد نمی‌کنند مقدار $b_i - a_i^T x^*$ مخالف صفر است و در نتیجه قضیه‌ی لنگی مکمل بیان می‌کند که برای آن دسته از قیدها حتما باید $p_i = 0$ باشد که همان بیان این مطلب است که دیوارهایی که در نقطه‌ی تعادل با جسم برخوردی ندارند به آن نیرویی وارد نمی‌کنند.در شکل صفحه‌ی قبل قضیه‌ی دوگانی قوی نیز بدین صورت قابل استنتاج است که در نقطه‌ی x^* قیدهای اول و دوم فعالند و در نتیجه برای آن‌ها داریم

$$\begin{cases} a_1^T x^* = b_1 \\ a_2^T x^* = b_2 \end{cases}$$

و هم‌چنین ضریب $p_3 = 0$ بدین علت که دیوار سوم با جسم در نقطه‌ی تعادل برخوردی ندارد. دو نیروی وارده از طرف دو دیوار دیگر باید جسم را در حالت تعادل نگه دارند پس

$$p_1 a_1 + p_2 a_2 = c$$

حال به سادگی می‌توان در این شکل قضیه‌ی دوگانی قوی را ثابت کرد.

$$\begin{aligned} b^T p = p_1 b_1 + p_2 b_2 &= p_1(a_1^T x^*) + p_2(a_2^T x^*) \\ &= (p_1 a_1^T + p_2 a_2^T) x^* \\ &= c^T x^* \qquad \square \end{aligned}$$

به صورت کاملا مشابه می‌توان این نتایج را در حالت کلی تر از این شکل و در ابعاد بالاتر گرفت.

مثال ۳: بهترین ترکیب خطی قیود مساله‌ی دوگان را می‌سازد!

یک شهود برای مساله‌ی دوگان این بود که ما شروط و قیود مساله را حذف کنیم و به نحوی یک ضریبی از نقض شدن این محدودیت را وارد تابع هدف کنیم. می‌توان از یک طریق دیگر نیز شهود مناسبی را نسبت به مساله‌ی دوگان به دست آورد. مساله‌ی برنامه‌ریزی خطی زیر را در نظر بگیرید.

$$\min \quad 12y_1 + 3y_2 + 4y_3$$

$$s.t. \begin{cases} 4y_1 + 2y_2 + 3y_3 \geq 2 \\ 8y_1 + y_2 + 2y_3 \geq 3 \\ y_1, y_2, y_3 \geq 0 \end{cases}$$

اگر به قیدها توجه کنیم می‌بینیم که به راحتی از روی آن‌ها می‌توان کران‌های پایینی برای تابع هدف به دست آورد. به علت نامنفی بودن y_iها داریم

$$12y_1 + 3y_2 + 4y_3 \geq 4y_1 + 2y_2 + 3y_3 \geq 2$$

پس می‌توان نتیجه گرفت که یک کران پایین برای مقدار بهینه ۲ است. اگر دقت کنیم می‌بینیم که این کار را با قید دوم نیز می‌توان انجام داد و در این صورت خواهیم داشت

$$12y_1 + 3y_2 + 4y_3 \geq 8y_1 + y_2 + 2y_3 \geq 3$$

و در این صورت یک کران پایین بهتر برای مقدار تابع هدف به دست می‌آید. حال اگر کمی دقت کنیم می‌یابیم، در صورتی که ما ترکیب‌های خطی دو قید را بسازیم به نحوی که ضرایب ساخته شده در آن ترکیب خطی برای هر y_i از ضریب متناظر با آن در تابع هدف کوچک‌تر باشد هم‌چنان می‌توان یک کران پایین معتبر برای مقدار تابع هدف به دست آورد. به عنوان مثال اگر قید اول را در $\frac{1}{2}$ ضرب کرده و سپس با قید دوم جمع کنیم خواهیم داشت

$$\left.\begin{cases} 2y_1 + 1y_2 + 1.5y_3 & \geq 1 \\ 8y_1 + y_2 + 2y_3 & \geq 3 \end{cases}\right\} \implies 10y_1 + 2y_2 + 3.5y_3 \geq 4$$

$$12y_1 + 3y_2 + 4y_3 \geq 10y_1 + 2y_2 + 3.5y_3 \geq 4$$

و باز یک کران پایین بهتر برای تابع هدف به دست آوردیم. حال سوال این است که چه ترکیب خطی از قیود بهترین کران پایین را به دست می‌دهد. پاسخ به این سوال منجر به همان ساختن مساله‌ی دوگانی می‌شود. ما ترکیب خطی دو قید را به صورتی در نظر می‌گیریم که

$$12y_1 + 3y_2 + 4y_3 \geq p_1(4y_1 + 2y_2 + 3y_3) + p_2(8y_1 + y_2 + 2y_3) \geq 2p_1 + 3p_2 \qquad (۱)$$

بهترین کران پایین، مقدار بیشینه‌ی $2p_1 + 3p_2$ است به شرطی که $p_1, p_2 \geq 0$ باشند تا هنگامی که در نامساوی ها ضرب می‌شوند جهت نامساوی برنگردد و ضرایب جمله‌ی میانی نامساوی فوق از ضرایب در تابع هدف کوچک‌تر باشد. بدین صورت مساله‌ی زیر به دست آمده است

$$\max \quad 2p_1 + 3p_2$$
$$s.t. \begin{cases} 4p_1 + 8p_2 \leq 12 \\ 2p_1 + p_2 \leq 3 \\ 3p_1 + 2p_2 \leq 4 \\ p_1, p_2 \geq 0 \end{cases}$$

که همان دوگان مساله‌ی اصلی است.

دقت داریم که منطق قواعد ساخت دوگان یک مساله ازروی آن طبق این تعبیر آشکار می‌شوند، به عنوان مثال اگر در مساله‌ی اصلی شرط $y_3 \geq 0$ به شرط $y_3 \leq 0$ تغییر می‌کرد، برای درست ماندن روابط کران پایین در نامساوی‌های نوشته شده در رابطه‌ی (۱) باید ضریب متناظر با y_3 در جمله‌ی میانی بیشتر از ضریب آن در تابع هدف باشد و در نتیجه شرط $3p_1+2p_2 \leq 4$ در مساله‌ی دوگان به شرط $3p_1+2p_2 \geq 4$ تغییر می‌کرد. یا به عنوان مثال اگر علامت y_3 آزاد بود، باید نامعادله‌ی $3p_1 + 2p_2 \leq 4$ به حالت تساوی در می‌آمد.

مثال ۴: دوگان در همه‌ی حالات معادلِ یک مساله یکسان است!

مساله‌ی برنامه‌ریزی خطی زیر را در نظر بگیرید.

حالت(۱):

$$\begin{cases} \min \quad c^T x \\ Ax \geq b \end{cases}$$

می‌توان فرم‌های معادل این مساله را به دست آورد. مثلا فرض کنید که می‌خواهیم فرم معادل این مساله در حالتی که قیدها حتما به صورت تساوی هستند را به دست آوریم.

حالت(۲):

$$\begin{cases} \min \quad c^T x \\ Ax - s = b \\ s \geq 0 \end{cases} \implies \begin{cases} \min \quad \begin{bmatrix} c & 0 \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} \\ \begin{bmatrix} A & -I \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix} = b \\ s \geq 0 \end{cases}$$

یا می‌توان فرم معادل مساله در حالت اول را بدین گونه به دست آورد که باید همه‌ی متغیرهای بهینه‌سازی نامنفی باشند. در این صورت با تبدیل $x_i = x_i^+ - x_i^-$ فرم معادل مساله پیدا می‌شود. **حالت(۳):**

$$\begin{cases} \min \quad c^T x^+ - c^T x^- \\ Ax^+ - Ax^- \geq b \\ x^+, x^- \geq 0 \end{cases}$$

در ادامه نشان می‌دهیم که فرم مساله‌ی دوگان برای هر سه حالت یکی است. هر کدام از دوگان‌ها را بر اساس قواعد ساخت دوگان از مساله‌ی اصلی در جلسه‌ی برنامه‌ریزی خطی (۳) به دست می‌آوریم.

دوگان حالت (۱):

$$\begin{cases} \max \quad b^T y \\ A^T y = c \\ y \geq 0 \end{cases}$$

دوگان حالت (۲):

$$\begin{cases} \max \quad b^T y \\ A^T y = c \\ -I^T y \leq 0 \to -y \leq 0 \\ y \text{ آزاد در علامت} \end{cases} \implies \begin{cases} \max \quad b^T y \\ A^T y = c \\ y \geq 0 \end{cases}$$

دوگان حالت (۳):

$$\begin{cases} \max \quad b^T y \\ A^T y \leq c \\ A^T y \geq c \\ y \geq 0 \end{cases} \implies \begin{cases} \max \quad b^T y \\ A^T y = c \\ y \geq 0 \end{cases}$$

مراجع

[۱] ویدیوی پانزدهمین جلسه‌ی حل تمرین که قابل دسترسی از اینجا است.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس حل تمرین: مرتضی علیمی

[بهار ۹۹]

جلسه حل تمرین ۱۶: استفاده از قضیه‌ی دوگانی برای اثبات قضایای ترکیبیاتی نگارنده: جواد فرخ‌نژاد

در دو جلسه‌ی اخیر به موضوع برنامه‌ریزی خطی پرداختیم و چند مسأله را با استفاده از آن مدل کردیم. این جلسه به بررسی شهودی دوگان مسائل می‌پردازیم و در انتها اثباتی از قضیه‌ی جریان بیشینه ـ برش کمینه با استفاده‌ای هوشمندانه از قضیه‌ی لنگی مکمل می‌بینیم.

۱ مسأله‌ی کوتاهترین مسیر

فرض کنید یک گراف جهت‌دار و وزن‌دار با مجموعه رئوس V و مجموعه یال‌های E داریم. وزن یال uv را با w_{uv} نشان می‌دهیم. دو رأس s و t نیز داده شده‌اند. می‌خواهیم مسأله‌ی یافتن کوتاهترین مسیر از s به t را با برنامه‌ریزی خطی مدل کنیم، سپس دوگان آن را بررسی کنیم.

قبلا در جلسات درس درمورد تابع پتانسیل روی رئوس گراف صحبت کردیم. در اینجا برای هر رأس مثل v یک متغیر p_v تعریف می‌کنیم که متناظر با پتانسیل رأس v است. به طور شهودی می‌توان فرض کرد که رئوس گراف با نخ‌هایی به طول وزن یال‌ها به هم وصل شده‌اند سپس کل گراف را از رأس s آویزان می‌کنیم و p_v اختلاف ارتفاع رأس s و v است.
بنابراین مسأله‌ی برنامه‌ریزی خطی زیر را داریم:

$$\begin{cases} \max & p_t - p_s \\ \text{s.t.} & p_j - p_i \le w_{ij} \quad \forall ij \in E \\ & p_u \quad \text{آزاد در علامت} \quad \forall u \in V \end{cases}$$

اگر بخواهیم مسأله را به طور ماتریسی بیان کنیم قید $Ap \le w$ را خواهیم داشت که A یک ماتریس با بعد $|E| \times |V|$ است و p و w به ترتیب بردارهایی در $\mathbb{R}^{|V|}$ و $\mathbb{R}^{|E|}$ اند. هر سطر ماتریس A متناظر با یک یال است و در هر سطر دقیقا یک درایه‌ی ۱ و یک درایه‌ی -1 وجود دارد و بقیه‌ی درایه‌ها صفرند.

برای نوشتن دوگان این مسأله متناظر با هر قید $p_j - p_i \le w_{ij}$ یک متغیر x_{ij} تعریف می‌کنیم. چون قیدهای $p_j - p_i \le w_{ij}$ در فرم متعارف‌اند بنابراین متغیرهای x_{ij} نامنفی‌اند. متناظر با هر متغیر p_v که آزاد در علامت است نیز یک قید تساوی به مسأله‌ی دوگان اضافه می‌شود به این صورت که هرجا متغیر p_v در قیدی مثل $p_j - p_i \le w_{ij}$ با علامت مثبت ظاهر شود یعنی یال ورودی به v داریم و اگر با علامت منفی ظاهر شود یعنی یال خروجی داریم. ضریب p_s در تابع هدف یک و ضریب p_t منفی یک است و ضریب بقیه‌ی p_iها صفر

است بنابراین دوگان مسأله به این صورت خواهد بود:

$$\begin{cases} \min & \sum_{ij\in E} w_{ij}x_{ij} \\ \text{s.t.} & \sum_{ji\in E} x_{ji} - \sum_{ij\in E} x_{ij} = ۰ \quad \forall i \neq t, s \in V \\ & \sum_{jt\in E} x_{jt} - \sum_{tj\in E} x_{tj} = ۱ \\ & \sum_{js\in E} x_{js} - \sum_{sj\in E} x_{sj} = -۱ \\ & x_{ij} \geq ۰ \quad \forall ij \in E \end{cases}$$

شهود مسأله‌ی دوگان در این مثال، جریان با هزینه‌ی کمینه است. زیرا می‌توان x_{ij} ها را جریانی که از یال‌ها می‌گذرند در نظر گرفت. البته در اینجا برای یال‌ها ظرفیت نداریم بلکه هزینه داریم یعنی هزینه‌ی ردشدن جریان x_{ij} از یال ij برابر با $w_{ij}x_{ij}$ است. می‌خواهیم جریان با مقدار ۱ را از s به t منتقل کنیم که کمترین هزینه را داشته باشد. با کمی تأمل می‌بینیم اگر کوتاهترین مسیر از s به t را در نظر بگیریم و از هر یالش جریان ۱ بگذرانیم دقیقا جریان با کمترین هزینه را خواهیم داشت.

۲ مسأله‌ی جریان بیشینه

فرض کنید یک شبکه با مجموعه رئوس V و مجوعه یال‌های E داریم. رئوس s و t نیز داده شده‌اند. ظرفیت یال ij را نیز c_{ij} درنظر می‌گیریم. می‌خواهیم مسأله‌ی جریان بیشینه در این شبکه را با استفاده از برنامه‌ریزی خطی مدل کنیم.

متناظر با هر یال ij یک متغیر x_{ij} تعریف می‌کنیم. متغیر x_{ts} را نیز درنظر می‌گیریم که انگار از رأس t به s یک یال با ظرفیت بی‌نهایت اضافه کردیم. با اضافه کردن این یال می‌توان فرض کرد جریان ورودی و خروجی برای هر رأس برابرند. پس مسأله‌ی برنامه‌ریزی خطی زیر را داریم:

$$\begin{cases} \max & x_{ts} \\ \text{s.t.} & x_{ij} \leq c_{ij} \quad \forall ij \in E \\ & \sum_{ji\in E} x_{ji} - \sum_{ij\in E} x_{ij} = ۰ \quad \forall i \in V \\ & x_{ij} \geq ۰ \end{cases}$$

در بیان ماتریسی مسأله داریم $x \leq c, Ax = ۰$ که در آن ماتریس A دارای بعد $|V| \times |E|$ است که درواقع ماتریس وقوع گراف جهت‌دار است.

برای نوشتن دوگان این مسأله برای هر i و j متناظر با قید $x_{ij} \leq c_{ij}$ متغیر d_{ij} را درنظر می‌گیریم همچنین برای هر i متناظر با قید $\sum_{ji\in E} x_{ji} - \sum_{ij\in E} x_{ij} = ۰$ متغیر p_i را درنظر می‌گیریم. مشابه مسأله‌ی قبلی به طور شهودی هر کدام از d_{ij}ها به نوعی طول یال‌ها اند و p_i را هم می‌توان فاصله‌ی رئوس از s درنظر گرفت. دقت کنید که اگر تمام p_iها را با $-p_s$ جمع کنیم بازهم درشرایط مسأله‌ی دوگان صدق می‌کنند و مقدار تابع هدف تغییری نمی‌کند پس واقعا می‌توان p_iها را فواصل رئوس از s درنظر گرفت.
شرط $d_{uv} + p_u - p_v \geq ۰$ از اینجا می‌آید که برای هر یال uv که $uv \neq ts$ متغیر x_{uv} دقیقا یکبار در قیدهای $x_{ij} \leq c_{ij}$، به ازای $ij = uv$ و دوبار با علامت‌های مثبت و منفی در قیدهای $\sum_{ji\in E} x_{ji} - \sum_{ij\in E} x_{ij} = ۰$ ، به ازای $i = u$ و $i = v$ ظاهر می‌شود. قید $d_{ij} + p_i - p_j \geq ۰$ در مسأله‌ی دوگان همان نامساوی مثلث را بیان می‌کند.

قید $p_t - p_s \geq ۱$ هم از اینجا می‌آید که ضریب x_{ts} در تابع هدف ۱ است و این متغیر دوجا در قیدهای $\sum_{ji \in E} x_{ji} - \sum_{ij \in E} x_{ij} = ۰$ ظاهر می‌شود.

چون شروط متناظر با متغیرهای p_i تساوی‌اند پس این متغیرها آزاد در علامت‌اند.

$$\begin{cases} \min & \sum_{ij \in E} c_{ij} d_{ij} \\ \text{s.t.} & d_{ij} + p_i - p_j \geq ۰ \quad \forall ij \neq ts \in E \\ & p_t - p_s \geq ۱ \\ & d_{ij} \geq ۰ \end{cases}$$

برای بررسی مسأله‌ی دوگان به طور شهودی می‌توان یال‌ها را به عنوان لوله‌هایی با طول‌های d_{ij} و سطح مقطع c_{ij} درنظر گرفت. هدف این است که حجم کل شبکه را کمینه کنیم با این شرط که فاصله‌ی دو رأس s و t حداقل ۱ باشد.

مثلا می‌توان طول لوله‌ها را اینطور درنظر گرفت که ابتدا مجموعه رئوس را به دو دسته افراز می‌کنیم (درواقع یک stـ برش درنظر می‌گیریم). سپس طول لوله‌های درون هر دسته را صفر و طول لوله‌های بین دودسته را یک تعریف می‌کنیم. این کار معادل با این است که پتانسیل رئوس دسته‌ی شامل s را صفر و پتانسیل رئوس شامل t را یک قرار دهیم. در این حالت می‌بینیم مقدار تابع هدف دقیقا برابر با ظرفیت stـ برش مورد نظر است. بنابراین تعبیر حالت خاصی از مسأله‌ی دوگان (زمانی که d_{ij}ها به طور خاصی که بیان شد مقدار دهی شوند)، همان مسأله‌ی یافتن stـ برش کمینه در گراف است. پس مقدار بهینه‌ی تابع هدف در مسأله‌ی دوگان کران پایین برای stـ برش کمینه در گراف است.

حال بنابر قضیه‌ی دوگانی ضعیف داریم: برش کمینه $\leq \mathrm{OPT}(D) \leq \mathrm{OPT}(P) =$ جریان بیشینه.

شاید به نظر برسد که از قضیه‌ی قوی دوگانی نتیجه می‌شود که جریان بیشینه $=$ برش کمینه، اما این نتیجه‌گیری اشتباه است. درمورد مسأله‌ی اولیه هر مقداردهی برای متغیرها متناظر با یک جریان در گراف بود. بنابراین مقدار بهینه‌ی تابع هدف در مسأله‌ی اولیه دقیقا برابر با جریان بیشینه در گراف است اما این مطلب درباره‌ی مسأله‌ی دوگان درست نیست. درواقع ما نمی‌دانیم که آیا جواب بهینه‌ی مسأله‌ی دوگان الزاما با stـ برش کمینه در گراف برابر است یا نه. فقط این را می‌دانیم که اگر متغیرهای دوگان مقادیر خاصی اتخاذ کنند مقدار تابع هدف برابر با ظرفیت یک stـ برش در گراف است و برای هر stـ برش نیز می‌توان متغیرها را طوری مقداردهی کرد که تابع هدف دقیقا با ظرفیت stـ برش برابر شود. اما اینکه آیا مقدار بهینه‌ی تابع هدف الزاما به ازای چنین مقداردهی‌هایی رخ می‌دهد بدیهی نیست و نیاز به اثبات دارد.

حال با روش هوشمندانه‌ای ثابت می‌کنیم درواقع مقدار بهینه‌ی تابع هدف دقیقا برابر با ظرفیت یک stـ برش در گراف خواهد شد. یعنی با استفاده از جواب بهینه‌ی مسأله‌ی اولیه و دوگان یک stـ برش می‌یابیم.

یک زوج جواب بهینه برای مسأله‌ی اولیه و دوگان درنظر بگیرید. برای سادگی از نماد $*$ استفاده نمی‌کنیم و هرجا متغیری از مسأله‌ی اولیه یا دوگان به کار بردیم منظور مقداردهی بهینه برای متغیر است. می‌توان فرض کرد $p_s = ۰$ زیرا بنابر استدلالی که قبلا گفته شد می‌توان تمام p_iها را با $-p_s$ جمع کرد. حال تعریف کنید:

$$A = \{v | v \in V, \quad p_v < ۱\} \quad B = V \setminus A$$

چون $p_s = ۰ < ۱$ داریم $s \in A$ و چون $p_t = p_t - p_s \leq ۱$ داریم $t \notin A$. پس (A, B) درواقع یک stـ برش برای گراف است. حال ثابت می‌کنیم در زوج جواب بهینه‌ای که در نظر گرفتیم تمام متغیرهای x_{ij} که متناظر با یک یال از A به B اند، الزاما یال اشباع شده‌اند و تمام متغیرهای x_{ij} که متناظر با یک یال از B به A اند، الزاما یال خالی‌اند. اگر این ادعا اثبات شود مقدار جریانی که از برش می‌گذرد دقیقا برابر است با مقدار ظرفیت برش و بدینصورت اثبات جدیدی برای قضیه‌ی جریان بیشینه ـ برش کمینه ارائه می‌شود.

فرض کنید vw یالی از A به B باشد. در اینصورت:

$$\left.\begin{array}{r} v \in A \Rightarrow p_v < ۱ \\ w \in B \Rightarrow p_w \geq ۱ \end{array}\right\} \Rightarrow d_{vw} \geq p_w - p_v > ۰$$

از طرفی چون متغیر d_{vw} متناظر با قید $x_{vw} \leq c_{vw}$ بود، بنابر قضیه‌ی لنگی مکمل داریم $d_{vw}(c_{vw} - x_{vw}) = ۰$. چون d_{vw} اکیدا مثبت است الزاما داریم $x_{vw} = c_{vw}$ یعنی یال vw اشباع شده است.

به طور کاملا مشابه فرض کنید vw یالی از B به A باشد. در اینصورت:

$$\left.\begin{array}{r} w \in A \Rightarrow p_w < ۱ \\ v \in B \Rightarrow p_v \geq ۱ \end{array}\right\} \Rightarrow p_v \geq ۱ > p_w \left.\begin{array}{r} \\ \\ d_{vw} \geq ۰ \end{array}\right\} \Rightarrow d_{vw} + p_v - p_w > ۰$$

متغیر x_{vw} متناظر با شرط $d_{vw} + p_v - p_w \geq ۰$ است. مشابها بنابر قضیه‌ی لنگی مکمل داریم $x_{vw}(۰ - (d_{vw} + p_v - p_w)) = ۰$ و نتیجه می‌شود که $x_{vw} = ۰$. یعنی یال vw خالی است.

پس به طور کلی ادعای مورد نظر اثبات شد.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

مدرس حل تمرین: مرتضی علیمی

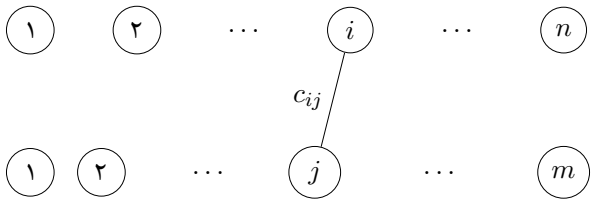
[بهار ۹۹]

جلسه حل تمرین ۱۷: مدل‌سازی مسائل ترکیبیاتی با برنامه‌های صحیح نگارنده: جواد فرخ‌نژاد

در جلسات گذشته مثال‌هایی در رابطه با برنامه‌ریزی خطی دیدیم. این جلسه نیز به حل چند مثال در همین رابطه می‌پردازیم.

۱ مسأله‌ی مکان‌یابی تسهیلات

همانطور که در جلسه‌ی ۲۶ دیدیم مسأله‌ی مکان‌یابی تسهیلات[۱] به این صورت است که n تا تسهیل و m تا مشتری داریم. هزینه‌ی باز شدن iامین تسهیل برابر با f_i است. برای اینکه بتوان از یک تسهیل استفاده کرد حتما باید این تسهیل باز شود. درضمن اگر مشتری jام بخواهد ازتسهیل iام استفاده کند هزینه‌ی c_{ij} دارد. اینجا ورژنی از مسأله را در نظر می‌گیریم که هر مشتری دقیقا بخواهد از یک تسهیل استفاده کند و برای تسهیلات محدودیتی نداریم و هرتعداد مشتری می‌توانند از یک تسهیل استفاده کنند[۲]. هدف برطرف کردن نیازهای مشتری‌ها با کمترین هزینه است.



ابتدا مسأله را به صورت یک مسأله‌ی برنامه‌ریزی خطی صحیح بیان می‌کنیم.
متناظر با iامین تسهیل متغیر y_i را تعریف می‌کنیم که برابر با ۱ است اگر تسهیل iام باز شود وگرنه ۰ است. همچنین برای هر $1 \le i \le n$ و هر $1 \le j \le m$ متغیر x_{ij} را به این صورت تعریف می‌کنیم که برابر با ۱ است اگر مشتری jام از تسهیل iام استفاده کند وگرنه برابر ۰ است.
با این تعاریف هزینه‌ی باز شدن تسهیلات برابر با $\sum_{i=1}^{n} f_i y_i$ و هزینه‌ی استفاده‌ی مشتری jام از تسهیلات برابر با $\sum_{i=1}^{n} c_{ij} x_{ij}$ خواهد بود. از طرفی هر مشتری دقیقا از یک تسهیل استفاده می‌کند بنابرین برای هر $1 \le j \le m$ باید داشته باشیم $\sum_{i=1}^{n} x_{ij} = 1$. همچنین تا زمانی که تسهیل iام باز نشده باشد هیچ کدام از مشتری‌ها نمی‌توانند از آن استفاده کنند بنابراین برای هر $1 \le i \le n$ و هر $1 \le j \le m$ داریم $x_{ij} \le y_i$. نهایتا با این توضیحات می‌توان مسأله را به صورت یک مسأله‌ی برنامه‌ریزی خطی صحیح به شکل زیر بیان کرد:

$$\begin{cases} \min & \sum_{i=1}^{n} f_i y_i + \sum_{j=1}^{m} \sum_{i=1}^{n} c_{ij} x_{ij} \\ \text{s.t.} & \sum_{i=1}^{n} x_{ij} = 1 \quad 1 \le j \le m \\ & x_{ij} \le y_i \quad 1 \le i \le n, \quad 1 \le j \le m \\ & x_{ij}, y_i \in \{0, 1\} \quad 1 \le i \le n, \quad 1 \le j \le m \end{cases}$$

[۱] Facility Location
[۲] uncapacitated facility location

اگر این مسأله را ریلکس کنیم باید فرض کنیم که x_{ij} و y_iها می‌توانند مقادیر حقیقی بگیرند و $۰ \leq x_{ij} \leq ۱$ و $۰ \leq y_i \leq ۱$. در اینجا می‌توان شرط $x_{ij} \leq ۱$ را درنظر نگرفت زیرا برای هر $۱ \leq j \leq m$ داریم $\sum_{i=۱}^{n} x_{ij} = ۱$ و x_{ij}ها نامنفی‌اند بنابراین x_{ij}ها خودبه‌خود کوچکتر یا مساوی ۱ خواهند شد. همچنین می‌توانیم شرط $y_i \leq ۱$ را نیز درنظر نگیریم زیرا اگر به ازای یک i داشته باشیم $y_i > ۱$ آنگاه اگر مقدار y_i را به ۱ تغییر دهیم مقدار تابع هدف به اندازه‌ی $f_i(y_i - ۱)$ کم می‌شود و با این کار خللی در قیدهای $x_{ij} \leq y_i$ وارد نمی‌شود چون تمام x_{ij}ها کوچکتر یا مساوی ۱ اند. پس چون مسأله‌ی کمینه سازی داریم می‌بینیم شرط $y_i \leq ۱$ خودبه‌خود محقق می‌شود.

حال می‌خواهیم دوگان این مسأله‌ی ریلکس شده را به طور شهودی بررسی و پیدا کنیم.

می‌توان اینطور به دوگان مسأله نگاه کرد که قرار است از هر مشتری مقداری هزینه دریافت کنیم. مشتری jام هزینه‌ی v_jمی‌دهد که متناظر با هزینه‌ی وصل شدن به یک تسهیل است (متغیر v_j در دوگان متناظر با قید $\sum_{i=۱}^{n} x_{ij} = ۱$ در مسأله‌ی اصلی است).

همچنین باید هزینه‌ی بازشدن تسهیلات را نیز از مشتری‌ها دریافت کنیم. پس برای هر $۱ \leq i \leq n$ و هر $۱ \leq j \leq m$ یک متغیر w_{ij} تعریف می‌کنیم که متناظر است با هزینه‌ای که از مشتری jام در ازای باز شدن تسهیل iام دریافت می‌شود (متغیر w_{ij} در دوگان متناظر با قید $x_{ij} \leq y_i$ در مسأله‌ی اصلی است).

هر مشتری می‌خواهد کمترین هزینه را بدهد بنابراین برای هر $۱ \leq i \leq n$ و هر $۱ \leq j \leq m$ باید قید $v_j \leq c_{ij} + w_{ij}$ را به دوگان اضافه کرد. یعنی اول برای هر تسهیل و هر مشتری تعیین می‌کنیم که اگر مشتری jام بخواهد از تسهیل iام استفاده کند باید هزینه‌ی $c_{ij} + w_{ij}$ را بدهد و نهایتا مشتری‌ها خودشان انتخاب می‌کنند که هزینه‌ی مربوط به کدام تسهیل را بدهند و قطعا آن تسهیل را انتخاب می‌کنند که کمترین هزینه را بدهند.

درمورد علامت متغیرها نیز به وضوح داریم $۰ \leq w_{ij}$ و $۰ \leq v_j$ که البته می‌توان شرط $v_j \leq ۰$ را بیان نکرد زیرا مسأله بیشینه سازی $\sum_{j=۱}^{m} v_j$ است و قیدی هم نداریم که منفی بودن v_j ها را نتیجه دهد (سمت راست تمام قیدهای $v_j \leq c_{ij} + w_{ij}$ نامنفی‌اند). پس این شرط خودبه‌خود محقق می‌شود. نهایتا دوگان مسأله‌ی مورد نظر به این صورت خواهد بود:

$$\begin{cases} \max & \sum_{j=۱}^{m} v_j \\ \text{s.t.} & v_j \leq c_{ij} + w_{ij} \quad ۱ \leq i \leq n, \quad ۱ \leq j \leq m \\ & \sum_{j=۱}^{m} w_{ij} \leq f_i \quad ۱ \leq i \leq n \\ & w_{ij} \geq ۰ \end{cases}$$

که شرط $\sum_{j=۱}^{m} w_{ij} \leq f_i$ را می‌توان با $\sum_{j=۱}^{m} w_{ij} = f_i$ نیز جایگزین کرد چرا که اگر تساوی رخ ندهد می‌توان یکی از w_{ij} ها را زیاد کرد که تساوی رخ دهد و این کار در بقیه‌ی قیدها خللی ایجاد نمی‌کند و مقدار تابع هدف ممکن است زیاد شود. پس هردو مسأله معادلند.

۲ مسأله‌ی مجموعه‌ی مستقل بیشینه

فرض کنید یک گراف داریم و می‌خواهیم مجموعه‌ی مستقل بیشینه را بیابیم. هدف این است که این مسأله را با یک مسأله‌ی برنامه‌ریزی خطی صحیح مدل کنیم.

برای این منظور فرض می‌کنیم V مجموعه‌ی رئوس گراف و E مجموعه‌ی یال‌های گراف باشد. برای هر رأس $v \in V$ یک متغیر x_v تعریف می‌کنیم که برابر با ۱ است اگر این رأس در مجموعه‌ی انتخابی باشد وگرنه ۰ است. قیدهای این مسأله به این صورت‌اند که برای هر یال uv

حداکثر یکی از رئوس u و v انتخاب شوند، معادلا $x_u + x_v \leq ۱$. پس می‌توان مسأله را به این صورت مدل کرد:

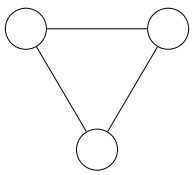
$$\begin{cases} \max & \sum_{v \in V} x_v \\ \text{s.t.} & x_u + x_v \leq ۱ \quad \forall uv \in E \\ & x_v \in \{۰, ۱\} \quad \forall v \in V \end{cases}$$

اگر این مسأله را ریلکس کنیم به این صورت تبدیل می‌شود:

$$\begin{cases} \max & \sum_{v \in V} x_v \\ \text{s.t.} & x_u + x_v \leq ۱ \quad \forall uv \in E \\ & ۰ \leq x_v \leq ۱ \quad \forall v \in V \end{cases}$$

نکته قابل توجه در این مسأله این است که جواب بهینه‌ی مسأله‌ی ریلکس شده الزاما با جواب بهینه‌ی مسأله‌ی اصلی برابر نیست. در واقع اگر با این ریلکس کردن، جواب بهینه‌ی مسأله تغییری نمی‌کرد آنگاه می‌توانستیم ابتدا مسأله‌ی مجموعه مستقل بیشینه را به یک مسأله‌ی برنامه‌ریزی خطی صحیح تبدیل کنیم سپس آن را ریلکس کنیم و به یک مسأله‌ی برنامه‌ریزی خطی عادی برسیم سپس این مسأله‌ی برنامه‌ریزی خطی را با الگوریتم چندجمله‌ای که برایش موجود است حل کنیم و جواب بدست آمده همان جواب مسأله‌ی مجموعه مستقل بیشینه است. اما می‌دانیم مسأله‌ی مجموعه مستقل بیشینه انپی‌ـ تمام است. پس اگر جواب‌های بهینه‌ی دو مسأله‌ی برنامه‌ریزی خطی صحیح و ریلکس شده یکی باشند، اثبات می‌شود که $\mathbf{N} = \mathbf{NP}$.

اما متأسفانه مثال زیر نشان می‌دهد که جواب بهینه‌ی مسأله‌ی ریلکس شده می‌تواند اکیدا از جواب مسأله‌ی اصلی بزرگتر باشد.



جواب بهینه‌ی مسأله‌ی ریلکس شده مربوط به این گراف برابر با ۱٫۵ است که زمانی اتفاق می‌افتد که متغیر متناظر با هر رأس مقدار ۰٫۵ بگیرد. اما جواب بهینه‌ی مسأله‌ی صحیح برابر با ۱ است.

حال می‌خواهیم به همین مسأله‌ی ریلکس شده تعدادی شرط اضافه کنیم که فضای شدنی مسأله، به پوش‌محدب نقاط شدنی مسأله‌ی صحیح نزدیک‌تر شوند. برای این منظور می‌توان شروطی اضافه کرد که بیان کند از هر خوشه[۳] در گراف حداکثر یک رأس می‌توانیم برداریم. پس می‌توان شروط زیر را به مسأله اضافه کرد:

$$\sum_{v \in C} x_v \leq ۱ \quad \forall C \text{ خوشه}$$

تعداد این محدودیت‌ها می‌تواند نمایی باشد اما به هرحال هرکدام از این شرط‌ها را اضافه کنیم احتمالا فضای حالت مسأله را کوچکتر می‌کند و به پوش‌محدب نقاط شدنی مسأله‌ی صحیح نزدیک‌تر می‌شود. اصطلاحا با این کار مدل‌سازی را تقویت کردیم.

۳ مدل‌سازی عملگر or

فرض کنید یک مسأله‌ی برنامه‌ریزی خطی داریم که دارای دو قید $a_۱^t x \geq b_۱$ و $a_۲^t x \geq b_۲$ است. هدف این است که یک مسأله‌ی برنامه‌ریزی خطی بنویسیم که در آن می‌خواهیم حداقل یکی از این دو شرط ارضاء شوند. یعنی فضای حالت این مسأله چنان باشد که برای هر x در

[۳] clique

فضای حالت حداقل یکی از دو شرط $a_۱^t x \geq b_۱$ یا $a_۲^t x \geq b_۲$ رخ دهد. معادلا می‌خواهیم شرط $a_۱^t x \geq b_۱ \quad \text{or} \quad a_۲^t x \geq b_۲$ را مدل‌سازی کنیم.

برای این منظور یک متغیر y تعریف می‌کنیم و شروط زیر را در نظر می‌گیریم:

$$\begin{cases} a_۱^t x \geq b_۱(۱ - y) \\ a_۲^t x \geq b_۲ y \\ y \in \{۰, ۱\} \end{cases}$$

می‌بینیم که هر xای که در شروط بالا صدق کند حتما یکی از دوشرط $a_۱^t x \geq b_۱$ یا $a_۲^t x \geq b_۲$را ارضاء می‌کند زیرا اگر $y = ۰$ شرط اول و اگر $y = ۱$ شرط دوم ارضاء می‌شود. البته ممکن است هر دو را نیز ارضاء کند اما مشکلی بوجود نمی‌آید.

مشابها می‌توان این مسأله را بررسی کرد که m قید داشته باشیم مانند $a_i^t x \geq b_i$ برای هر $۱ \leq i \leq m$ و بخواهیم مسأله‌ای بنویسیم که در آن حداقل k تا از این شروط ارضاء شوند.

برای این مسأله نیز می‌توان متغیر های $y_۱, y_۲, \ldots, y_m$ تعریف کرد و شروط زیر را درنظر گرفت:

$$\begin{cases} a_i^t x \geq b_i y_i \quad ۱ \leq i \leq m \\ \sum_{i=۱}^{m} y_i = ۱ \\ y_i \in \{۰, ۱\} \quad ۱ \leq i \leq m \end{cases}$$

۴ مدل‌سازی تعلق یک متغیر به یک مجموعه‌ی متناهی

فرض کنید می‌خواهیم در یک مسأله‌ی برنامه‌ریزی خطی یکی از متغیرها مثل x را محدود کنیم به این صورت که فقط می‌تواند یکی از مقادیر مجموعه‌ی $\{a_۱, a_۲, \ldots, a_k\}$ را اتخاذ کند.

برای این منظور می‌توان متغیرهای $y_۱, y_۲, \ldots, y_k$ را تعریف کرد و شروط زیر را در نظر گرفت:

$$\begin{cases} x = \sum_{i=۱}^{k} a_i y_i \\ \sum_{i=۱}^{k} y_i = ۱ \\ y_i \in \{۰, ۱\} \quad ۱ \leq i \leq k \end{cases}$$

اگر این شروط محقق شوند، دقیقا یکی از y_iها برابر با ۱ خواهد بود و بقیه صفراند و متغیر x نیز دقیقا با یکی از a_iها برابر می‌شود.

۵ مسأله‌ی درخت فراگیر کمینه

فرض کنید یک گراف داریم که همبند است و یال‌های گراف وزن‌دارند. می‌خواهیم کم‌وزن‌ترین زیردرخت فراگیر برای این گراف را بیابیم. هدف این است که مسأله را به یک مسأله‌ی برنامه‌ریزی خطی تبدیل کنیم.

مجموعه یال‌های گراف را E، مجموعه‌ی رأس‌های گراف را V و وزن یال $e \in E$ را c_e درنظر می‌گیریم. متناظر با هر یال $e \in E$ یک متغیر x_e تعریف می‌کنیم که درصورتی که یال مورد نظر انتخاب شود متغیر مقدار ۱ می‌گیرد وگرنه مقدار ۰ می‌گیرد. باید قیدهایی اضافه کنیم که باعث شود یال‌های انتخابی تشکیل درخت دهند. به دو شیوه این کار را انجام می‌دهیم و آن‌ها را با هم مقایسه می‌کنیم.

رویکرد اول این است که قیدهایی اضافه کنیم که باعث شود یال‌های انتخابی تشکیل دور ندهند. به این رویکرد حذف دور[۴] می‌گویند. برای این منظور برای هر زیرمجموعه از رأس‌های گراف، باید تعداد یال‌های انتخابی که دو سرشان بین آن رئوس است از تعداد رئوس کمتر باشد. قید دیگر هم این است که دقیقا $|V| - ۱$ یال باید انتخاب کنیم که تشکیل درخت فراگیر بدهند. پس مسأله را می‌توان اینطور مدل کرد:

$$\begin{cases} \min & \sum_{e \in E} c_e x_e \\ \text{s.t.} & \sum_{e \subseteq S} x_e \leq |S| - ۱ \quad \forall S \subseteq V \\ & \sum_{e \in E} x_e = |V| - ۱ \\ & x_e \in \{۰, ۱\} \quad \forall e \in E \end{cases}$$

تعداد قیدهای این مسأله می‌تواند نمایی باشد. اما روش‌هایی برای حل مسائل برنامه‌ریزی خطی موجودند به نام روش‌های بیضوی[۵] که تحت شرایط خاصی می‌توانند مسائل برنامه‌ریزی خطی که تعداد نمایی قید دارند را در زمان چندجمله‌ای حل کنند. در واقع اگر برای هر مقداردهی برای متغیرها بتوانیم به روش هوشمندانه‌ای و در زمان چندجمله‌ای بفهمیم آیا یکی از قیدهای مسأله نقض شده است یا نه، روش‌های بیضوی در این موارد کار می‌کنند. درمورد مسأله‌ی درخت فراگیر کمینه این روش وجود دارد بنابراین اینکه تعداد قیدهای مسأله نمایی شده است مشکل خیلی بزرگی نیست.

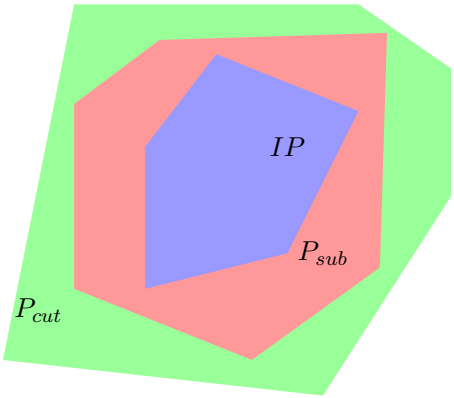
رویکرد دوم این است که قیدهایی اضافه کنیم که همبند بودن را نتیجه دهد. اگر یال‌های انتخابی همبند نباشند یک زیرمجموعه از رئوس مثل A پیدا خواهد شد که بین رئوس A و $V \setminus A$ هیچ یال انتخاب شده‌ای موجود نیست. معادلا یک برش در گراف یافتیم که هیچ یالی از یال‌های برش، انتخاب نشده‌اند. پس می‌توانیم قیدهایی اضافه کنیم به این صورت که برای هر برش حداقل یک یال از برش انتخاب شود. پس مسأله به این صورت مدل می‌شود:

$$\begin{cases} \min & \sum_{e \in E} c_e x_e \\ \text{s.t.} & \sum_{e \in \delta(A)} x_e \geq ۱ \quad \forall A \subseteq V \\ & \sum_{e \in E} x_e = |V| - ۱ \\ & x_e \in \{۰, ۱\} \quad \forall e \in E \end{cases}$$

حال می‌خواهیم این دو مدل‌سازی را مقایسه کنیم. ریلکس شده‌ی هر دو مسأله را درنظر بگیرید. ملاک بهتر بودن در اینجا این است که فضای شدنی مسأله‌ی ریلکس شده چقدر به پوش‌محدب نقاط شدنی مسأله‌ی ریلکس نشده نزدیک است. در واقع ایده‌آل برای یک مسأله‌ی برنامه‌ریزی خطی صحیح زمانی است که فضای شدنی مسأله‌ی ریلکس شده دقیقا مساوی با پوش محدب نقاط مسأله‌ی ریلکس نشده باشند، که در این حالت جواب بهینه‌ی مسأله‌ی ریلکس شده همان جواب بهینه‌ی مسأله‌ی ریلکس نشده است.

ادعا می‌کنیم رویکرد اول بهتر است. فرض کنید فضای شدنی رویکرد اول در حالت ریلکس شده برابر با P_{sub} و درمورد رویکرد دوم P_{cut} باشد. ثابت می‌کنیم $P_{sub} \subseteq P_{cut}$. این نشان می‌دهد که فضای شدنی مسأله در رویکرد اول به پوش‌محدب نقاط شدنی مسأله‌ی ریلکس نشده نزدیک‌تر است و بنابراین رویکرد اول بهتر است.

[۴] subtour elimination
[۵] ellipsoid



فرض کنید $x \in P_{sub}$ یعنی بردار x در شروط مدل اول صدق کند (حالت ریلکس شده). ثابت می‌کنیم این x در شروط مدل دوم نیز صدق می‌کند (در حالت ریلکس شده).

فرض کنید $A \subseteq V$ یک زیرمجموعه از رئوس گراف باشد. داریم:

$$\sum_{e \in A} x_e \leq |A| - ۱ \quad \text{و} \quad \sum_{e \in V \setminus A} x_e \leq |V \setminus A| - ۱ \quad \text{و} \sum_{e \in E} x_e = |V| - ۱$$

بنابراین:

$$\sum_{e \in \delta(A)} x_e = \sum_{e \in E} x_e - \sum_{e \in A} x_e - \sum_{e \in V \setminus A} x_e \geq (|V| - ۱) - (|A| - ۱) - (|V \setminus A| - ۱) = ۱ \quad \Rightarrow \quad \sum_{e \in \delta(A)} x_e \geq ۱$$

پس بردار x در شروط مدل دوم نیز صدق می‌کند و ادعای موردنظر اثبات می‌شود.

بخش چهارم

تمرین‌های هفتگی

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۱ موعد: سه‌شنبه ۲۹ بهمن ساعت ۱۲

---

۱. (آ) فرض کنید موعد تحویل تمرین نظری شما فرداست و هنوز تمرین‌ها را حل نکرده‌اید. همچنین فرض کنید فردا ۳ میان‌ترم دارید. آیا درست است که پاسخ تمرین‌ها را از دوستتان بگیرید؟

(ب) در مسئله بخش قبل همچنین فرض کنید که این آخرین سری تمرین است و اگر نمره آن را نگیرید یا از میان‌ترم‌های فردا نمره کمی بگیرید ممکن است یکی از درس‌هایتان را بیفتید و با توجه به اینکه ترم ۱۰ هستید فارغ‌التحصیلی شما به خطر می‌افتد. در این حالت آیا درست است پاسخ تمرین‌ها را از دوستتان بگیرید؟

(ج) فرض کنید یک سوال تمرین را در اینترنت جستجو کنید و جواب آن را پیدا کنید. بعد از خواندن و متوجه شدن جواب، صفحه جواب را ببندید و جواب را از ذهن خودتان و با بیان خودتان بنویسید. همچنین در بالای تمرین به سایت موردنظر ارجاع دهید. آیا این کار مجاز است؟

(د) فرض کنید موعد تحویل تمرین عملی شما امشب است. یکی از سوال‌ها را نوشته‌اید، اما کد شما خطا می‌دهد و با وجود صرف وقت بسیار نتوانسته‌اید خطای آن را پیدا کنید. یکی از بچه‌هایی که سال گذشته درس الگوریتم را گذرانده، از دوستان نزدیک شماست. آیا درست است از او برای اشکال‌یابی برنامه‌تان کمک بگیرید؟

۲. فرض کنید پنجشنبه ۴ اردیبهشت عروسی خواهرتان است. از یک طرف به هیچ‌وجه نمی‌توانید در عروسی شرکت نکنید و تاریخ عروسی هم قابل تغییر نیست و از طرف دیگر نمی‌خواهید نمره میان‌ترم را از دست بدهید. بهترین کاری که می‌توانید انجام دهید کدام مورد است؟

(آ) در عروسی شرکت می‌کنم و سپس از استاد می‌خواهم تا برای میان‌ترمم جایگزینی تعیین کند.

(ب) قبل از تاریخ مورد نظر از استاد می‌خواهم که تاریخ میان‌ترم را تغییر دهد.

(ج) سعی می‌کنم در بقیه قسمت‌های درس نمره خوبی بگیرم تا در نهایت نمره قابل‌قبولی کسب کنم.

(د) به محض مشخص شدن مشکل، به استاد ایمیل می‌زنم و مشکل را توضیح می‌دهم و درخواست می‌کنم در صورت امکان یک میان‌ترم جداگانه از من گرفته شود یا به هرگونه‌ای که صلاح است نمره میان‌ترمم جبران شود.

۳. درستی یا نادرستی هرکدام از موارد زیر را به همراه توضیحی کوتاه مشخص کنید.

(آ) هر الگوریتم مرتب‌سازی مقایسه‌ای حداقل به زمان $O(n \lg n)$ نیاز دارد.

(ب) زمان اجرای بهترین الگوریتم برای پیدا کردن یک زیرمجموعه مستقل ۱۰ رأسی از یک گراف n رأسی $O(n^{10})$ است.

(ج) زمان اجرای هر الگوریتم برای پیدا کردن یک زیرمجموعه مستقل ۱۰ رأسی از یک گراف n رأسی $\Omega(n^{10})$ است، چون لازم است همه $\binom{n}{10}$ حالت ممکن را تست کنیم.

(د) ساختمان داده‌ای وجود دارد که n عدد را در حافظه $O(n)$ ذخیره کند و با داده شدن یک عدد جدید، بتواند با احتمال ۱ در زمان $O(1)$ تعیین کند که این عدد در بین n عدد اولیه هست یا خیر.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۲ موعد: سه‌شنبه ۶ اسفند ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با andishe.ghasemi.9@gmail.com مطرح کنید.

تمرین‌های پیشنهادی (غیرتحویلی)

CLRS 22.1.5, 22.1.6, 22.1.8, 22.2.1, 22.2.7, 33.4.3.

تمرین‌های تحویلی

۱. روی یک صفحه مختصات n نقطه مشخص شده‌است. به یک وضعیت از n نقطه خوب گوییم اگر به ازای هر جفت نقطه مثل (x, y) و (x', y') یکی از سه شرط زیر برقرار باشد:
۱) $x = x'$
۲) $y = y'$
۳) نقطه‌ی دیگری چون (x'', y'') موجود باشد به طوری‌که $\min(x, x') \le x'' \le \max(x, x')$ و $\min(y, y') \le y'' \le \max(y, y')$.
می‌خواهیم تعدادی نقطه به مجموعه نقاط اضافه کنیم تا در انتها به یک وضعیت خوب برسیم؛ الگوریتمی طراحی کنید که در زمان $O(n \lg n)$ به مقدار $O(n \lg n)$ نقطه به این نقاط اضافه کند طوری که در انتها یک وضعیت خوب از نقاط داشته‌باشیم.

۲. برای یک عدد b که $۲ \le b \le ۱۰$ است، عدد خوب را عددی طبیعی تعریف می‌کنیم که تمام ارقام آن کمتر از b باشد، مثلا برای $b = ۲$، تمام اعداد با ارقام ۰ و ۱ خوب هستند.
برای n و b داده‌شده، می‌خواهیم با کمک مدل‌سازی اعداد در گراف و استفاده از نسخه‌ای تغییر یافته از «الگوریتم جستجوی اول سطح» کوچکترین عدد خوب مضرب n را پیدا کنیم؛ مثلا برای $n = ۳, b = ۲$ کوچکترین عدد خوب مضرب ۳ برابر با ۱۱۱ است.

(آ) سعی کنید با طراحی یک گراف مناسب، الگوریتمی از $O(n)$ ارائه دهید تا با گرفتن n, b کوچکترین عدد خوب مضرب n را بیابد.

(ب) فرض کنید x یک عدد خوب است، همزاد این عدد را عددی مثل y تعریف می‌کنیم که $(x)_b = (y)_{۱۰}$ (منظور از اندیس مبنا است.) مثلا برای $b = ۲$ همزاد عدد خوب ۱۱۱ عدد ۷ است.
الگوریتمی از زمان $O(n^۲)$ ارائه دهید که با ورودی گرفتن n, b کوچکترین عدد خوب که هم خودش و هم همزادش مضرب n باشد بیابید.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۳ موعد تحویل: سه‌شنبه ۱۳ اسفند ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با r.akbarian98@yahoo.com , javadakbari1379@gmail.com مطرح کنید.

تمرین‌های پیشنهادی

CLRS 22.3.1, 22.3.2, 22.3.5, 22.3.8, 22.3.9, 22.3.11, 22.4.4.

تمرین‌های تحویلی

۱. درختی با n رأس داده شده‌است. الگوریتمی طراحی کنید که در زمان $O(n)$:

(آ) طولانی ترین مسیر را پیدا کند.

(ب) کوتاه‌ترین گشت گذرنده از تمام رأس‌ها را بدست آورد.

۲. گوییم زوج مرتب (a, b) می‌تواند از زوج مرتب (c, d) برنده شود، اگر $a \geq c$ یا $b \geq d$.
n زوج عدد به صورت a_i و b_i به همراه m رابطه‌ی برنده‌شدن به صورت اینکه کدام زوج می‌تواند کدام زوج را ببرد داریم.
می‌خواهیم هر کدام از این زوج‌ها را به صورت زوج مرتب (a_i, b_i) یا (b_i, a_i) تبدیل کنیم به طوری که m رابطه‌ی برنده‌شدن بین آن‌ها برقرار باشد.
الگوریتمی از $O(n + m)$ ارائه دهید که بگوید این‌کار ممکن است یا خیر.
راهنمایی: می‌توانید از مسئله 2SAT یا از روش تشخیص دوبخشی بودن گراف استفاده کنید.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۴ موعد: سه‌شنبه ۲۰ اسفند ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با javadakbari1379@gmail.com , andishe.ghasemi.9@gmail.com مطرح کنید.

تمرین‌های پیشنهادی

حالت وزن‌دار مسئله زمان‌بندی بازه‌ها را با برنامه‌ریزی پویا حل کنید.

CLRS Exercises 16.4.1, 23.1.1, 23.1.2, 23.1.3, 23.1.8.

CLRS Problems 15.1, 16.1.

تمرین‌های تحویلی

۱. قصد داریم که در یک مسیر بی انتها(!) تا جای ممکن پیش برویم. در ابتدای مسیر n کامیون داریم که ظرفیت بنزین آن‌ها تکمیل است. حداکثر ظرفیت بنزین هر کامیون یک لیتر است و مصرف سوخت هر کامیون برای طی‌کردن یک کیلومتر از مسیر برابر یک لیتر است. همچنین می‌توانیم در هر جای مسیر هر کامیونی را به دلخواه نگه‌داشته و مقداری از بنزین آن را به کامیونی دیگر منتقل‌کنیم. حداکثر مسیری که می‌توانیم پیش برویم چقدر است؟(با ذکر دلیل و اثبات درستی)

۲. قصد داریم با یک ماشین که با حداکثر ظرفیت بنزینش d کیلومتر حرکت می‌کند به یک سفر دور کشور برویم. از قبل همه‌ی n پمپ‌بنزین موجود در طول مسیرمان را روی نقشه پیدا کرده‌ایم.فاصله‌ی بین پمپ بنزین‌های متوالی بیشتر از d کیلومتر نیست.می‌خواهیم برنامه سفر را طوری بچینیم که کمترین توقف‌های ممکن را برای سوخت‌گیری داشته‌باشیم. الگوریتمی طراحی کنید که با گرفتن d و فاصله‌ی پمپ‌بنزین‌ها از نقطه شروع لیست پمپ‌بنزین‌هایی که باید در آن ها توقف کنیم را خروجی دهد.

۳. در یک تیم والیبال n والیبالیست داریم. قد همه‌ی آن‌ها بین ۱۹۵ سانتی‌متر و ۲۰۵ سانتی‌متر بوده و میانگین قد آن‌ها ۲۰۰ سانتی‌متر است. می‌خواهیم همه بازیکنان به ترتیبی در یک ردیف بایستند. اگر مربی تیم دو بازیکن را به دلخواه انتخاب کند که k بازیکن دیگر بین آن‌ها باشند ، اختلاف میانگین قد این $k+2$ بازیکن از مقدار $200 \times (k+2)$ را حساب کرده و اگر این مقدار بیشتر از ۱۰ سانتی‌متر باشد از تیم خود ناامید می‌شود.الگوریتمی حریصانه برای پیدا کردن ترتیبی از بازیکنان ارائه دهید که مربی را ناامید نکند.

تمرین امتیازی

فرض کنید گراف $G = (V, E)$ با دو تابع وزن $c_1 : E \to Q_+$ و $c_2 : E \to Q_+$ داده شده باشد. الگوریتمی چندجمله‌ای ارائه دهید که تشخیص دهد آیا G زیردرخت فراگیری دارد که هم‌زمان برای c_1 و c_2 کمینه باشد.
راهنمایی: تعمیم مسئله برای ماتروئیدها را حل کنید.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۵ موعد: سه‌شنبه ۲۷ اسفند ساعت ۱۲

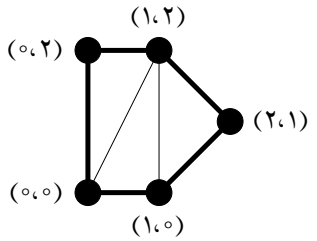
ـ سؤالات خود پیرامون تمرین را با alirtofighim@gmail.com, r.akbarian98@yahoo.com مطرح کنید.

تمرین‌های پیشنهادی

CLRS Exercises 24.1.4, 24.1.5, 24.2.4, 24.3.2, 24.3.9, 24.3.6.
CLRS Problem 15.4.

تمرین‌های تحویلی

۱. یک n ضلعی محدب داریم و می‌خواهیم آن‌را مثلث‌بندی کنیم، منظور از یک مثلث‌بندی برای یک n ضلعی افراز n ضلعی به مثلث‌هایی است که رئوس آن‌ها، رئوس n ضلعی باشند.
هزینه‌ی یک مثلث‌بندی مجموع محیط مثلث‌هاست، الگوریتمی از $O(n^۳)$ ارائه دهید که کمترین هزینه برای مثلث‌بندی را بیابد.



مثال بالا، یک مثلث بندی را نشان می‌دهد که دو مثلث با محیط $\sqrt{۵} + ۳$ و یک مثلث با محیط $۲\sqrt{۲} + ۲$ داریم یعنی هزینه‌ی آن $۲\sqrt{۵} + ۲\sqrt{۲} + ۸$ است.

۲. یک جدول $n \times m$ داریم که خانه‌های آن به رنگ قرمز یا آبی است. می‌خواهیم بزرگترین زیرمربع از جدول را انتخاب کنیم که فقط از یک رنگ تشکیل شده‌باشد. الگوریتمی از $O(nm)$ ارائه دهید تا این زیرمربع را پیدا کند.

۳. شهری داریم که که شامل n تقاطع و خیابان‌هایی بین این تقاطع‌هاست. اگر تقاطع‌های آن را به عنوان رئوس و خیابان‌های آن به عنوان یال‌های یک گراف در نظر بگیریم، این گراف تشکیل یک درخت می‌دهد. می‌خواهیم برای تعدادی از تقاطع‌های شهر چراغ نصب کنیم، در صورت قرار دادن چراغ برای یک تقاطع، تمامی خیابان‌های متصل به آن روشن می‌شود. می‌خواهیم کمترین تعداد چراغ لازم را خریداری کرده تا تمام خیابان‌های شهر روشن شوند، الگوریتمی از $O(n)$ طراحی کنید که کمترین تعداد چراغ لازم و تعداد حالت‌های نصب کمترین چراغ لازم برای اینکه کل خیابان‌های شهر روشن شوند را محاسبه کند.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۶ موعد: سه‌شنبه ۱۲ فروردین ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با alirtofighim@gmail.com, javadakbari1379@gmail.com مطرح کنید.

تمرین‌های پیشنهادی

تمرین سری ۳ (سؤال ۲) و سری ۴ و حل‌تمرین جلسه ۵ از درس الگوریتم کارنگی‌ملون.

تمرین‌های مربوط به مباحث تدریس شده از درس الگوریتم برکلی.

تمرین‌های عملی مربوط به مباحث تدریس شده از درس دنی اسلیتور.

تمرین‌های تحویلی

۱. یک گراف وزن‌دار و جهت‌دار V راسه و E یاله داریم که فاقد دور منفی است، روی این گراف Q امر مطرح می‌شود، هر امر یکی از دو نوع زیر است:

- v را ورودی داده و از ما می‌خواهند راس v را از گراف پاک کنیم.
- دو راس v, u را ورودی داده و وزن کوتاه‌ترین مسیر (از نظر مجموع وزن یال‌ها) از v به u را از ما می‌خواهند.

متاسفانه در لحظه‌ی مطرح شدن این امور، ما قادر به پاسخ‌دهی به آن‌ها نبودیم و حال همه‌ی Q امر را یک‌جا داریم، الگوریتمی ارائه دهید که با گرفتن گراف و Q امر در زمان اجرای $O(V^3 + Q)$ به عنوان خروجی پاسخ امرهای نوع دوم را بدهد.

۲. یک گراف وزن‌دار V راسه و E یاله با وزن یال‌های نامنفی داریم. شخصی روی راس s و شخص دیگری روی راس e ایستاده‌است، عبور از هر یال e دقیقاً یک واحد زمان می‌برد و به اندازه‌ی وزن یال e هزینه دارد. دو شخص مسئله‌ی ما از یک دیگر متنفرند، به همین دلیل نمی‌خواهند لحظه‌ای باشد که هر دو روی یک راس قرار گرفته‌باشند (ولی می‌توانند در یک لحظه در دو سر یک یال باشند و لحظه‌ی بعد جای خود را عوض کنند و ملاقات در وسط یال اشکالی ندارد)، هر دو انسان‌های عجولی هستند به همین دلیل به محض رسیدن به راسی از آن به راس دیگری حرکت می‌کنند و توقفی روی راس‌ها ندارند.
شخص اول می‌خواهد از راس s به راس e و شخص دوم می‌خواهد از راس e به راس s برسد، الگوریتمی طراحی کنید که با گرفتن گراف و راس‌های s و e، کمترین هزینه‌ای که این دو شخص در مجموع برای این جابه‌جایی باید بدهند را حساب کند.

(آ) زمان اجرای الگوریتم از $O(V^2 lgV + E^2)$ باشد.

(ب) زمان اجرای الگوریتم از $O(V^2 lgV + VE)$ باشد.

راهنمایی. سعی کنید گراف جدیدی با V^2 راس بسازید و پاسخ مسئله‌ی بالا را به یک مسئله‌ی کوتاه‌ترین مسیر در گراف جدید تبدیل کنید.

۳. یک روش بهینه‌سازی الگوریتم بلمن‌فورد در عمل، این است که اگر در یکی از مراحل ریلکس‌کردن مشاهده کردیم که هیچ یک از d_iها کاهش پیدا نکرد، الگوریتم را متوقف کنیم؛ می‌خواهیم بررسی کنیم که آیا این کار باعث بهبود زمان اجرای الگوریتم در بدترین حالت می‌شود یا خیر. اثبات یا رد کنید که برای گرافی وزن‌دار با V راس بدون دور منفی و با وجود یک ترتیب روی یال‌ها برای الگوریتم بلمن‌فورد، نیاز به اجرای $\Omega(V)$ بار فرایند ریلکس کردن روی کل یال‌ها را داریم.

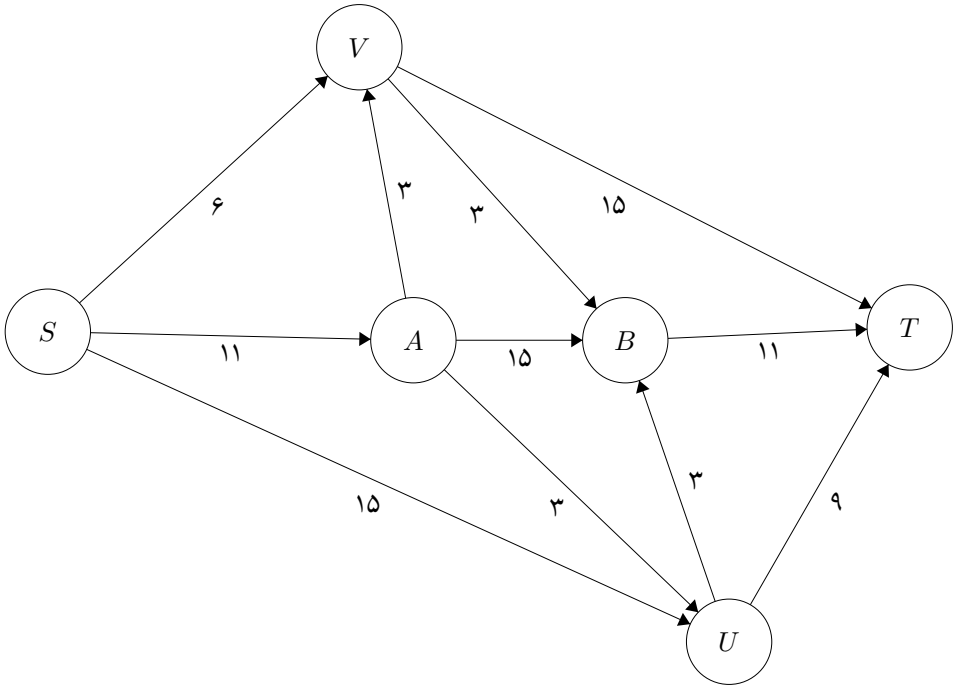
موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۷ موعد: سه‌شنبه ۲۶ فروردین ساعت ۱۲

- سؤالات خود پیرامون تمرین را با andishe.ghasemi.9@gmail.com مطرح کنید.

۱. در گراف زیر ظرفیت هر یال مشخص شده‌است. و S رأس منبع[1] و T رأس چاه[2] در این شبکه است.
آ) یک شار بیشینه بدست آورید و شکل آن را به همراه مقدار جریان عبوری از هر یال مشخص کنید.
ب) یک برش کمینه بدست آورید. مقدار ظرفیت خروجی این برش و همچنین رأس‌های آن را مشخص کنید.
پ) گراف باقی‌مانده نهایی را رسم کنید. کدام رأس از S قابل دسترسی است؟ کدام رأس دسترسی به T دارد؟
ت) به یک یال بحرانی افزایشی می‌گوییم اگر افزایش ظرفیت آن یال منجر به افزایش شار بیشینه شود. به یک یال بحرانی کاهشی می‌گوییم اگر کاهش ظرفیت آن یال منجر به کاهش شار بیشینه شود. یک یال افزایشی بحرانی و یک یال افزایشی کاهشی در این گراف پیدا کنید.(اگر وجود دارد.)
ث) الگوریتمی بهینه برای پیدا کردن یک یال بحرانی کاهشی طراحی کنید.
ج) الگوریتمی بهینه برای پیدا کردن یک یال بحرانی افزایشی طراحی کنید.(امتیازی)



[1] source
[2] sink

۲. درستی یا نادرستی موارد زیر را مشخص کنید. در صورت درست بودن اثبات مختصری ارائه کنید و در صورت نادرست بودن مثال نقض بیاورید.

آ) اگر همه‌ی یال‌های جهت‌دار یک شبکه ظرفیت‌های متفاوتی داشته باشند ، آنگاه شار بیشینه به طریقی یکتا بدست می‌آید.

ب) در مسئله‌ی شار بیشینه با ظرفیت رأس‌ها گراف جهت‌دار $G = (V, E)$ داده شده‌است که S رأس منبع و T رأس چاه است و ظرفیت هر رأس $v \in V$ برابر با $c_v \geq 0$ است.(ولی ظرفیتی برای یال‌ها داده نشده‌است.) یک شار f را معتبر برای گراف G می‌گوییم اگر برای همه‌ی v ها بجز S و T ، مجموع شار ورودی به رأس v حداکثر c_v باشد. اندازه‌ی یک شار معتبر f ، مجموع شار خروجی از S است. با داشتن گراف ورودی ، مسئله‌ی شار بیشینه با ظرفیت رأس‌ها ، محاسبه کردن یک شار معتبر با سایز بیشینه است.

محاسبه‌ی یک شار بیشینه با ظرفیت رأس‌ها را می‌توان به مسئله‌ی شار بیشینه معمولی(با ظرفیت یال‌ها) کاهش داد.

پ) اگر هر یال جهت‌دار با ظرفیت c و بین دو رأس u و v در یک شبکه را با دو یال جهت‌دار با جهت‌های مخالف و ظرفیت c بین دو رأس u و v جایگزین کنیم ، آنگاه مقدار شار بیشینه ثابت می‌ماند.

۳. در یک ساختمان عمومی مثل یک سینما ، داشتن یک نقشه‌ی خروج برای موارد اضطراری نظیر آتش‌سوزی مهم است. در این سوال می‌خواهیم با استفاده از شار بیشینه یک نقشه خروج اضطراری طراحی کنیم. فرض کنید که نقشه سینما یک گراف $G = (V, E)$ است که در آن هر اتاق یا طبقه با یک رأس و هر راهرو یا پله با یک یال مشخص شده‌است. هر راهرو یا پله دارای ظرفیتی c است که نشان می‌دهد حداکثر c نفر همزمان می‌توانند از این راهرو استفاده کنند. پیمایش یک راهرو از یک سر تا سر دیگر یک واحد زمانی طول می‌کشد.(پیمایش یک اتاق صفر واحد زمانی طول می‌کشد.) فرض کنید در ابتدا همه مردم در اتاق S هستند و تنها یک خروجی T به خیابان وجود دارد. نشان دهید که چطور با استفاده از مسئله‌ی شار بیشینه ، سریع‌ترین راه برای خارج کردن همه افراد از ساختمان را پیدا کنیم. (راهنمایی : گراف G' را طراحی کنید که در آن هر رأس نشان‌دهنده‌ی یک اتاق در هر واحد زمانی باشد.)

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۸ موعد: سه‌شنبه ۲ اردیبهشت ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با Mavali1999@gmail.com مطرح کنید.

۱. گراف وزن‌دار با وزن‌های طبیعی و بدون جهت G و دو راس s, t از رئوس G داده شده است.

می‌خواهیم در این گراف دو مسیر متمایز از s به t بیابیم، طوری‌که این دو مسیر در هیچ یالی مشترک نباشند و همچنین وزن هر کدام برابر با وزن کوتاه‌ترین مسیر از s به t باشد. دقت کنید که این دو مسیر می‌توانند در یک یا چند راس مشترک باشند.
با کمک مسئله‌ی شار بیشینه، الگوریتمی برای پیدا کردن این دو مسیر ارائه دهید.

۲. گراف وزن‌دار و بدون جهت G را در نظر بگیرید.
در این گراف به هر رأس و هر یال وزنی طبیعی و کمتر از C نسبت داده شده‌است.
برای هر زیرگراف مثل H در G، وزن H را مجموع وزن یال‌های H منهای مجموع وزن راس‌های H تعریف می‌کنیم. (بدیهی است که اگر یال e در زیرگراف H باشد، آنگاه حتما راس‌های ابتدا و انتهای e نیز در زیر گراف هستند.)

با کمک مسئله‌ی شار بیشینه، الگوریتمی ارائه دهید که زیرگرافی با وزن بیشینه در G را بیابد.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۹ موعد: چهارشنبه ۱۰ اردیبهشت ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با Ghazalkhn99@gmail.com, Mavali1999@gmail.com مطرح کنید.

۱. برای مسائل زیر کوچک‌ترین کلاس پیچیدگی از بین P، NP و NP-Complete که می‌توانید ثابت کنید مسئله در آن است کدام است؟ دلیل مختصری بیاورید.

(آ) یک گراف $G = (V, E)$ داده شده است، آیا $V' \subseteq V$ موجود است که $|V'| = k$ و بین هر دو راس در V' یالی در G باشد؟ (k یک عدد ثابت است و جزء ورودی‌های مسئله نیست.)

(ب) یک گراف $G = (V, E)$ و عدد k داده شده است، آیا $V' \subseteq V$ موجود است که $|V'| = k$ و بین هر دو راس در V' یالی در G باشد؟

(ج) یک گراف $G = (V, E)$ و عدد k داده شده است، آیا $E' \subseteq E$ وجود دارد که $|E'| < k$ و برای هر $v \in V$، یالی مانند e در E' باشد که v یکی از دو سر e باشد؟

(د) یک گراف $G = (V, E)$ و عدد k داده شده است، آیا $V' \subseteq V$ وجود دارد که $|V'| < k$ و برای هر $e \in E$، راسی مانند v در V' باشد که v یکی از دو سر e باشد؟

(ه) مسئله‌ی 3-SAT با این فرض اضافه برای ورودی که در ورودی هر متغیر یا نقیض آن حداکثر ۲ بار در ورودی ظاهر شده است.

(و) مسئله‌ی 3-SAT با این فرض اضافه برای ورودی که در ورودی هر متغیر یا نقیض آن حداکثر ۴ بار در ورودی ظاهر شده است.

راهنمایی. مسئله‌ی پیداکردن بزرگترین تطابق در گراف در زمان چندجمله‌ای بر حسب اندازه‌ی گراف ورودی قابل حل است.

۲. در گراف $G = (V, E)$ یک زیرمجموعه از راس‌ها مثل U را خیلی‌مستقل گوییم هرگاه هیچ مسیری با طول حداکثر ۲ در G بین دو راس از U موجود نباشد. ثابت کنید مسئله‌ی زیر NP-Complete است:
یک گراف $G = (V, E)$ و عدد k داده شده است، آیا G زیرمجموعه‌ی خیلی‌مستقل به اندازه‌ی حداکثر k دارد؟

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۱۰ موعد: سه‌شنبه ۱۵ اردیبهشت ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با alirtofighim@gmail.com مطرح کنید.

۱. با فرض $\mathbf{P} = \mathbf{NP}$ الگوریتمی چند جمله‌ای ارائه دهید که یک 3SAT به عنوان ورودی گرفته و در صورتی که ورودی ارضاپذیر باشد، مقداردهی متغیرها که 3SAT را ارضا کند خروجی دهد.

۲. مسئله‌ی کوله‌پشتی چندگانه به این صورت است که m کوله‌پشتی داریم که ظرفیت کوله‌پشتی iام برابر با c_i است. همچنین n الماس داریم که الماس iام ارزش v_i و وزن w_i دارد. می‌توانیم تعدادی از الماس‌ها را برداشته و هر کدام را داخل یکی از کوله‌پشتی‌ها قرار دهیم، اما مجموع وزن الماس‌های داخل یک کوله‌پشتی نباید از ظرفیت آن کوله‌پشتی بیشتر شود.
ورودی ظرفیت کوله‌پشتی‌ها، مشخصات الماس‌ها و عدد k است و می‌خواهیم ببینیم آیا می‌توانیم حداقل با مجموع ارزش k الماس داخل کوله‌پشتی‌ها جا دهیم.

(آ) برای حالت $m = ۱$ (یک کوله‌پشتی) ثابت کنید مسئله‌ی کوله‌پشتی انپی‌ـتمام است.

(ب) برای $m = ۱$ (یک کوله‌پشتی) و با فرض $\mathbf{P} \neq \mathbf{NP}$ ثابت کنید مسئله‌ی کوله‌پشتی قویاً انپی‌ـتمام نیست.

(ج) ثابت کنید مسئله‌ی کوله‌پشتی چندگانه قویاً انپی‌ـتمام است.

موفّق باشید.

مه‌راد

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

آقای راد که علاقه‌ی زیادی به کپی کردن دارد، دستگاهی ساخته که می‌تواند شکلات‌ها را بدون هزینه کپی کند! شکلات‌ها به صورت ردیف‌های kتایی از شکلات‌های واحد هستند که با چسب مخصوصی به هم چسبیده‌اند. فردا تولد همسر آقای راد، مه جان، است به همین خاطر آقای راد تصمیم می‌گیرد تا شکلاتی ۱ در n که شامل n شکلات واحد است به همسرش هدیه دهد.

کپی کردن شکلات‌ها برای آقای راد هیچ هزینه‌ای ندارد؛ تنها چسباندن دو شکلات به هم است که هزینه‌بر است. او یک شکلات واحد از دوستش سپهر قرض گرفته تا با کمک آن یک شکلات nتایی بسازد، برای این کار او در هر مرحله یکی از دو کار زیر را انجام می‌دهد:

۱. یکی از شکلات‌هایی که تا الآن ساخته (مثلا یک شکلات kتایی) را به دستگاه کپی داده و دو شکلات به همان طول (یعنی دو شکلات kتایی) دریافت می‌کند.
۲. دو شکلات مثل یک شکلات aتایی و یک شکلات bتایی را گرفته و با کمک چسب یک شکلات $a+b$تایی می‌سازد.

به دلیل حباب، قیمت چسب بسیار زیاد شده‌است و آقای راد برای k دفعه چسب زدن نیاز به k ماه پس‌انداز کردن دارد. به او بگویید که باید حداقل چندماه پس‌انداز داشته باشد.

ورودی

در تنها خط ورودی عدد n آمده است.

$$1 \leq n \leq 500$$

خروجی

در تنها خط خروجی، کمینه‌ی تعداد ماه‌هایی که آقای راد باید پس‌انداز کند را چاپ کنید.

مثال

ورودی نمونه ۱

```
1
```

خروجی نمونه ۱

```
0
```

کافی است شکلات واحد را به مه بدهد.

ورودی نمونه ۲

```
15
```

خروجی نمونه ۲

```
5
```

ابتدا یک شکلات واحد داریم. آن را ۲ بار تکثیر می‌کنیم تا ۳ شکلات واحد داشته باشیم. دو تا از آن‌ها را با چسب به هم می‌چسبانیم تا یک شکلات ۲ تایی ساخته شود. شکلات ۲تایی و شکلات واحد را به هم می‌چسبانیم تا شکلات ۳ تایی داشته‌باشیم. شکلات ۳ تایی را ۲ بار تکثیر می‌کنیم تا سه شکلات ۳تایی داشته باشیم. حال دو تا از این شکلات‌ها را به هم می‌چسبانیم و یک شکلات ۶ تایی می‌سازیم. شکلات ۶ تایی را تکثیر کرده و دو شکلات ۶ تایی داریم. حال دو شکلات ۶ تایی را به هم چسبانده و شکلات ۱۲ تایی می‌سازیم و سپس شکلات ۱۲ تایی و شکلات ۳ تایی را به هم چسبانده و شکلات ۱۵ تایی را می‌سازیم. در مجموع ۵ بار عمل چسباندن را انجام دادیم پس به ۵ ماه پس‌انداز نیاز داریم.

ورودی نمونه ۳

```
16
```

خروجی نمونه ۳

```
4
```

برعکس

- محدودیت زمان: ۲/۵ ثانیه (برای جاوا و پایتون ۱۰ ثانیه)
- محدودیت حافظه: **۱۲۸** مگابایت

یک دنباله‌ی n عضوی از اعداد مانند $a_1, a_2, ..., a_n$ داریم، می‌خواهیم یک زیردنباله از این دنباله را انتخاب کرده و ترتیب عضوهای این زیر دنباله را در دنباله‌ی اصلی برعکس کنیم تا طول بزرگ‌ترین زیردنباله‌ی صعودی دنباله‌ی حاصل بیشینه شود.

ورودی

در نخستین خط، عدد n که تعداد اعضای دنباله است آمده‌است.

$$1 \leq n \leq 100$$

در n خط بعد، در خط i ام عدد a_i آمده است.

$$1 \leq a_i \leq 150$$

خروجی

در خط اول خروجی، اندازه‌ی بزرگ‌ترین زیردنباله‌ی صعودی‌ای را که می‌توانیم آن را با یک بار انتخاب یک زیردنباله و برعکس کردن ترتیب عناصر آن زیردنباله بسازیم چاپ کنید.

در خط دوم تعداد عناصر زیردنباله‌ای که انتخاب کرده‌اید را چاپ کنید.

در خط سوم شماره اندیس‌های دنباله‌ی اصلی که این زیردنباله را می‌سازند چاپ کنید. (اندیس‌ها از یک شروع می‌شوند.)

در صورت وجود چندین جواب، هر کدام را چاپ کنید قبول است.

داوری

در این سوال شما اجازه‌ی پنج واحد خطا دارید، یعنی اگر بزرگترین دنباله‌ی صعودی قابل رسیدن با طول k

باشد، با چاپ هر کدام از اعداد $k, k-1, k-2, k-3, k-4, k-5$ پاسخ شما پذیرفته می‌شود.

▼ Tags

Simulated Annealing

▼ راه‌نمایی

کافی‌است یک زیرمجموعه‌ی به اندازه‌ی کافی خوب از $\{1, 2, ..., n\}$ پیدا کنیم، از یک زیرمجموعه‌ی تصادفی شروع کنید، یک همسایگی بین زیرمجموعه‌ها تعریف کنید سپس در هر مرحله یکی از همسایه‌های تصادفی را انتخاب کرده، بزرگترین زیردنباله‌ی صعودی در حالت جدید و قدیم را باهم مقایسه کنید و براساس این دو بزرگترین زیردنباله‌ی صعودی و یک عدد تصادفی مثل r تصمیم بگیرید که ادامه‌ی کار را با زیرمجموعه‌ی جدید ادامه دهید یا قبلی، این کار را به اندازه‌ی کافی تکرار کنید و در پایان اطلاعات مربوط به بزرگترین زیردنباله‌ی صعودی‌ای که در کل فرایند دیدید را چاپ کنید.

مثال

ورودی نمونه ۱

```
9
1
2
3
9
5
6
8
7
4
```

خروجی نمونه ۱

```
9
4
4 7 8 9
```

کافی است زیر دنباله از اندیس‌های ۴ام، ۷ام، ۸ام و ۹ام را انتخاب کنیم و ترتیب آن‌ها را برعکس کنیم تا به

آرایه‌ی ۱ تا ۹ برسیم.

ورودی نمونه ۲

```
5
1
2
3
4
5
```

خروجی نمونه ۲

```
5
0
```

کافی است کاری نکنیم!

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۱۲ موعد: پنجشنبه ۲۲ خرداد ساعت ۱۲

---

۱. چندوجهی

$$P = \{[x_۱, x_۲]^T \mid x_۱ + x_۲ \geq ۱,\ -x_۱ + x_۲ \leq ۲,\ x_۱ - ۲x_۲ \leq ۴,\ x_۱ \geq ۰\},$$

را در نظر بگیرید.

(آ) چندوجهی P را در یک دستگاه مختصات رسم کنید.

(ب) نقاط رأسی P را بیابید.

(ج) جهت‌های دور شونده راسی چندوجهی P را بیابید.

(د) با استناد به قضیه نمایش، نقاط $[۱/۲, ۳/۲]^T$، $[۱, ۱]^T$ و $[۳, ۲]^T$ را به صورت ترکیب نقاط رأسی و جهت‌های دور شونده رأسی بنویسید.

(ه) با استفاده از قضیه نمایش و نمایش نقاط شدنی مساله با استفاده از نقاط رأسی و جهت‌های دورشونده رأسی، مجموعه جواب بهینه مساله برنامه‌ریزی خطی

$$\min\ c^T x \quad \text{s.t.} \quad x \in P, \tag{۱}$$

که در آن $c^T = [۱, ۱]$ را در صورت وجود بیابید.

(و) مشابه قسمت قبل، نشان دهید مساله برنامه‌ریزی خطی (۱) که در آن $c^T = [-۲, -۱]$ جواب بهینه متناهی ندارد.

(ز) مشابه قسمت (ه)، مجموعه جواب بهینه مساله برنامه‌ریزی خطی (۱) که در آن $c^T = [-۲, -۲]$ را بیابید.

(ح) مشابه قسمت (ه)، مجموعه جواب بهینه مساله برنامه‌ریزی خطی (۱) که در آن $c^T = [۰, ۱]$ را بیابید. این مساله برنامه‌ریزی خطی را به صورت یک مساله برنامه‌ریزی خطی استاندارد بازنویسی کنید، سپس جواب بهینه این مساله برنامه‌ریزی استاندارد را بیابید.

۲. مساله برنامه‌ریزی خطی

$$\begin{cases} \min & -x_۱ - x_۲ + ۲x_۳ + x_۴ \\ \text{s.t.} & x_۱ + x_۲ + x_۳ + x_۴ \geq ۶ \\ & x_۱ - x_۲ - ۲x_۳ + x_۴ \leq ۴ \\ & x_۱, x_۲, x_۳, x_۴ \geq ۰. \end{cases}$$

را در نظر بگیرید.

(آ) با افزودن متغیرهای کمبود و مازاد مساله را به صورت استاندارد بازنویسی کنید. فضای التزام (فضای ترکیب خطی نامنفی بردار ضرایب متغیرها) را در صفحه رسم کنید.

(ب) با استناد به فضای التزام، استدلال کنید که چرا این مساله برنامه‌ریزی خطی شدنی است.

(ج) با فرض این که در هر جواب بهینه این مساله حداکثر دو متغیر مقدار غیر صفر می‌گیرند، و با استفاده از فضای التزام، جواب بهینه مساله را بیابید.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۱۳ غیرتحویلی

۱. برنامه خطی زیر را در نظر بگیرید.

$$\begin{array}{lrcrcrcl} \text{maximize} & ۱۳x_۱ & + & ۱۰x_۲ & + & ۶x_۳ & & \\ \text{subject to} & ۵x_۱ & + & x_۲ & + & ۳x_۳ & = & ۸ \\ & ۳x_۱ & + & x_۲ & & & = & ۳ \end{array}$$

$$x_۱, x_۲, x_۳ \geq ۰$$

(آ) دوگان این برنامه را بنویسید.

(ب) فرض کنید بردار $x^* = (۱, ۰, ۱)$ یک جواب بهینه برای این برنامه باشد. با استفاده از شرایط لنگی مکمل، یک جواب بهینه برای دوگان این برنامه پیدا کنید، و با استفاده از آن جواب تحقیق کنید که x^* واقعاً جواب بهینه برای برنامه اصلی است.

۲. یک گراف و تعدادی از رأس‌های آن داده شده‌اند. می‌خواهیم به یال‌های گراف وزن‌های مثبتی نسبت دهیم به‌طوری که مجموع وزن‌ها کمینه باشد و فاصله بین هر دو رأس از بین رأس‌های انتخاب شده از یک بیشتر شود. این مسئله را به صورت یک برنامه خطی مدل کنید و دوگان آن را بنویسید.

۳. برنامه صحیح زیر را در نظر بگیرید.

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{e\in E} c_e x_e + \sum_{v\in V} p_v y_v & \\ \text{s.t} & \displaystyle\sum_{e\in\delta(S)} x_e \geq y_v & \forall S \subseteq V - r, S \neq \emptyset,\ \forall v \in V \\ & y_r = ۱ & \\ & y_v \in \{۰, ۱\} & \forall v \in V \\ & x_e \in \{۰, ۱\} & \forall e \in E \end{array}$$

(آ) یک مسئله الگوریتمی روی گراف‌ها تعریف کنید که توسط این برنامه مدل شود.

(ب) این برنامه صحیح را به یک برنامه خطی ریلکس کنید و دوگان آن را بنویسید.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۱۴ موعد: پنجشنبه ۱۹ تیر ساعت ۱۲

---

۱. فرض کنید یک شرکت می‌تواند هرکدام از پروژه‌های $A, B, \ldots, H$ را انجام دهد. هرکدام از محدودیت‌های زیر را با استفاده از متغیرهای دودویی $x_a, x_b, \ldots, x_h$ مدل کنید.

(آ) حداکثر یکی از پروژه‌های $A, B, \ldots, H$ انجام شوند.

(ب) حداقل یکی از پروژه های $A, B, \ldots, H$.

(ج) اگر A آنگاه B.

(د) اگر A آنگاه B انجام نشود.

(ه) اگر A انجام نشود B انجام شود.

(و) A اگر و فقط اگر B.

(ز) اگر A، آنگاه B و C.

(ح) اگر A، آنگاه B یا C.

(ط) اگر B یا C آنگاه A.

(ی) اگر B و C آنگاه A.

(ک) اگر دو تا یا بیشتر از B, C, D, E آنگاه A.

۲. گراف بدون جهت $G = (V, E)$ را در نظر بگیرید که $V = \{v_1, \ldots, v_n\}$. رأس v_1 نشانگر یک پیتزافروشی است که برای سادگی فرض می‌کنیم فقط یک نوع پیتزا دارد. هرکدام از بقیه رأس‌ها نشانگر یک مشتری است که مشتری v_i می‌خواهد b_i تا پیتزا بخرد. همچنین به هر یال $e \in E$ هزینه c_e نسبت داده شده است. پیتزافروشی m پیک دارد که تعداد پیتزاهایی که هر پیک می‌تواند حمل کند Q است. هر پیک باید از مبدأ (v_1) شروع کند، و بعد از اینکه به تعدادی مشتری سرویس داد در نهایت به v_1 بازگردد. فرض می‌کنیم $b_i \leq Q$ برای هر i، و اینکه سفارش هر مشتری باید توسط یک پیک تحویل داده شود. می‌خواهیم اختصاص دادن سفارش مشتری‌ها به پیک‌های مختلف و مسیر پیک‌ها را طوری برنامه‌ریزی کنیم که کل مسیر پیمایش شده توسط پیک‌ها کمینه شود. این مسئله را توسط یک برنامه صحیح مدل کنید.

موفّق باشید.

بخش پنجم

تمرین‌های اضافه

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین اضافه سری ۱ موعد: چهارشنبه ۳۱ اردیبهشت ساعت ۱۲

---

۱. فرض کنید قرار است برای اولین بار درس الگوریتم‌های پسرفته در دانشکده ارائه شود. n نفر کاندیدای TA شدن در این درس هستند و ما می‌خواهیم دقیقاً یک نفر را انتخاب کنیم. مکانیزم انتخاب به این صورت است که به ترتیب با کاندیداها مصاحبه می‌کنیم، و بعد از مصاحبه با هرکدام، برای اینکه به وی استرس وارد نشود، بلافاصله باید تصمیم بگیریم که آیا این دانشجو را به عنوان TA انتخاب کنیم یا خیر و تصمیممان را به او اعلام کنیم. فرض کنید تنها چیزی که از مصاحبه با یک دانشجو می‌فهمیم این باشد که آیا او از همه کاندیداهای قبلی بهتر است یا خیر. نشان دهید هر الگوریتم تصادفی به احتمال حداکثر $1/n$ (روی بدترین دنباله ورودی برای آن الگوریتم) بهترین TA را انتخاب می‌کند.

راهنمایی ۱: از اصل مینیمکس یائو استفاده کنید.

راهنمایی ۲: دنباله‌هایی از کاندیداها را در نظر بگیرید که تا رسیدن به بهترین کاندیدا صعودی هستند.

۲. نشان دهید مشابه نتیجه‌ای که در درس برای حالت صفر و یکی مسئله خبرگان بیان شد، در حالت پیوسته هم برقرار است. به‌طور دقیق‌تر، فرض کنید بعد از روز tام، ضرر $c_i^t \in [0, 1]$ برای خبره iام مشخص شود، و سپس وزن خبره iام را در $(1 - c_i^t\epsilon)$ ضرب کنیم. همچنین هر خبره را در هر روز با احتمال متناسب با وزنش انتخاب می‌کنیم. نشان دهید بعد از T روز، امید ریاضی ضرر این الگوریتم حداکثر برابر است با $(1+\epsilon)M + \frac{\ln n}{\epsilon}$ که M ضرر بهترین خبره بعد از T روز است.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین اضافه سری ۲ موعد: پنجشنبه ۸ خرداد ساعت ۱۲

---

۱. مسئله ۳ـ رنگ‌پذیری را در نظر بگیرید. الگوریتم بدیهی این مسئله نیاز به زمان 3^n دارد. می‌خواهیم یک الگوریتم تصادفی با زمان $(\frac{3}{2})^n\text{poly}(n)$ برای این مسئله طراحی کنیم. فرض کنید به طور تصادفی رنگ هر رأس را به ۲ تا از ۳ گزینه ممکن محدود کنیم. نشان دهید تشخیص اینکه آیا می‌توان گراف را با این مجموعه رنگ‌هایی که برای هر رأس مشخص کرده‌ایم رنگ کرد یا نه را می‌توان در زمان چندجمله‌ای حل کرد (از مسئله 2SAT استفاده کنید). سپس نتیجه بگیرید که در صورتی که گراف ۳ـ رنگ‌پذیر باشد، می‌توان با احتمال بالا یک ۳ـ رنگ‌آمیزی از آن را در زمان $(\frac{3}{2})^n\text{poly}(n)$ پیدا کرد.

۲. فرض کنید مجموعه n عضوی A به عنوان ورودی داده شده باشد. یک نمونه تصادفی با سایز $s = n^{\frac{3}{4}}$ از A انتخاب می‌کنیم (با جایگذاری). فرض کنید $x_1 \leq x_2 \leq \ldots \leq x_s$ نمونه انتخاب شده باشد. نشان دهید احتمال اینکه تعداد عناصری از A که بین $x_{s-\sqrt{n}}$ و $x_{s+\sqrt{n}}$ هستند بیشتر از $4n^{\frac{3}{4}}$ باشد، $O(n^{-\frac{1}{4}})$ است.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین اضافه سری ۳ موعد: پنجشنبه ۱۵ خرداد ساعت ۱۲

---

۱. یک مجموعه n عضوی A و m مجموعه $S_1, \ldots, S_m \subseteq A$ را در نظر بگیرید. نشان دهید می‌توان اعضای A را به گونه‌ای با دو رنگ آبی و قرمز رنگ کرد که اختلاف تعداد اعضای آبی و قرمز هرکدام از S_iها $O(\sqrt{n \lg m})$ باشد. توجه کنید صرفاً باید وجود چنین رنگ‌آمیزی‌ای را اثبات کنید.

راهنمایی: اعضا را تصادفی رنگ کنید و با استفاده از کران چرنف و کران اجتماع نشان دهید به احتمال غیرصفر خاصیت گفته شده برقرار است. نتیجه بگیرید یک رنگ‌آمیزی خوب وجود دارد.

۲. یک الگوریتم چندجمله‌ای تصادفی ارائه دهید که دو درخت ریشه‌دار را به عنوان ورودی بگیرد و تشخیص دهد آیا این دو درخت یکریخت هستند یا نه. توجه کنید فرزندان گره‌ها ترتیب ندارند.

راهنمایی: برای هر زیردرخت یک چندجمله‌ای تعریف کنید به طوری که یکریخت بودن دو زیردرخت معادل با یکسان بودن چندجمله‌ای‌های آن‌ها باشد.

موفّق باشید.

بخش ششم

پاسخ تمرین‌های هفتگی

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۱

---

۱. (آ) فرض کنید موعد تحویل تمرین نظری شما فرداست و هنوز تمرین‌ها را حل نکرده‌اید. همچنین فرض کنید فردا ۳ میان‌ترم دارید. آیا درست است که پاسخ تمرین‌ها را از دوستتان بگیرید؟

پاسخ: خیر (فایل اطلاعات درس، بخش ۶، پاراگراف ۳).

(ب) در مسئله بخش قبل همچنین فرض کنید که این آخرین سری تمرین است و اگر نمره آن را نگیرید یا از میان‌ترم‌های فردا نمره کمی بگیرید ممکن است یکی از درس‌هایتان را بیفتید و با توجه به اینکه ترم ۱۰ هستید فارغ‌التحصیلی شما به خطر می‌افتد. در این حالت آیا درست است پاسخ تمرین‌ها را از دوستتان بگیرید؟

پاسخ: عیناً مانند بخش قبل.[1]

(ج) فرض کنید یک سوال تمرین را در اینترنت جستجو کنید و جواب آن را پیدا کنید. بعد از خواندن و متوجه شدن جواب، صفحه جواب را ببندید و جواب را از ذهن خودتان و با بیان خودتان بنویسید. همچنین در بالای تمرین به سایت موردنظر ارجاع دهید. آیا این کار مجاز است؟

پاسخ: طبق فایل اطلاعات درس، استفاده از راه‌حل‌های آماده مجاز نیست. بنابراین نمره‌ای از سؤال گفته شده نمی‌گیرید. اما از آنجایی که به منبع ارجاع داده‌اید، به عنوان متقلب هم شناخته نمی‌شوید.

(د) فرض کنید موعد تحویل تمرین عملی شما امشب است. یکی از سوال‌ها را نوشته‌اید، اما کد شما خطا می‌دهد و با وجود صرف وقت بسیار نتوانسته‌اید خطای آن را پیدا کنید. یکی از بچه‌هایی که سال گذشته درس الگوریتم را گذرانده، از دوستان نزدیک شماست. آیا درست است از او برای اشکال‌یابی برنامه‌تان کمک بگیرید؟

پاسخ: در فایل اطلاعات درس تأکید زیادی شده است که باید راه‌حل کار خود شما باشد. در سناریوی گفته‌شده، می‌توانید از دوستتان کمک بگیرید، اما باید احتیاط کنید که کمک دوستتان از حدی فراتر نرود و مثلاً وی در کد شما دست نبرد. در صورت تردید در مورد مجاز بودن یا نبودن هر موردی، بهترین کار پرسیدن مستقیم از مدرس درس است.

۲. فرض کنید پنجشنبه ۴ اردیبهشت عروسی خواهرتان است. از یک طرف به هیچ‌وجه نمی‌توانید در عروسی شرکت نکنید و تاریخ عروسی هم قابل تغییر نیست و از طرف دیگر نمی‌خواهید نمره میان‌ترم را از دست بدهید. بهترین کاری که می‌توانید انجام دهید کدام مورد است؟

(آ) در عروسی شرکت می‌کنم و سپس از استاد می‌خواهم تا برای میان‌ترمم جایگزینی تعیین کند.

(ب) قبل از تاریخ مورد نظر از استاد می‌خواهم که تاریخ میان‌ترم را تغییر دهد.

(ج) سعی می‌کنم در بقیه قسمت‌های درس نمره خوبی بگیرم تا در نهایت نمره قابل‌قبولی کسب کنم.

(د) به محض مشخص شدن مشکل، به استاد ایمیل می‌زنم و مشکل را توضیح می‌دهم و درخواست می‌کنم در صورت امکان یک میان‌ترم جداگانه از من گرفته شود یا به هرگونه‌ای که صلاح است نمره میان‌ترمم جبران شود.

[1] Yeah, really. Sorry. ☺

پاسخ: گزینه ایده‌آل (ج) است ☺ ، اما (د) هم قابل قبول است. در صورت انتخاب این گزینه، ممکن است با درخواست شما موافقت شود یا نشود. در صورت موافقت، محتمل‌ترین سناریو این است که یک میان‌ترم سخت‌تر در تاریخ دیگری از شما گرفته شود.

۳. درستی یا نادرستی هرکدام از موارد زیر را به همراه توضیحی کوتاه مشخص کنید.

(آ) هر الگوریتم مرتب‌سازی مقایسه‌ای حداقل به زمان $O(n \lg n)$ نیاز دارد.

پاسخ: درست است، اما گزاره پوچی است! گزاره بهتر این است: «زمان اجرای هر الگوریتم مرتب‌سازی مقایسه‌ای در بدترین حالت $\Omega(n \lg n)$ است.»

(ب) زمان اجرای بهترین الگوریتم برای پیدا کردن یک زیرمجموعه مستقل ۱۰ رأسی از یک گراف n رأسی $O(n^{۱۰})$ است.

پاسخ: درست. همان‌طور که در کلاس گفته شد، الگوریتم $\Theta(n^{۱۰})$ برای این مسئله وجود دارد (بررسی همه حالت‌های ممکن). بنابراین زمان بهترین الگوریتم هم $O(n^{۱۰})$ است.

(ج) زمان اجرای هر الگوریتم برای پیدا کردن یک زیرمجموعه مستقل ۱۰ رأسی از یک گراف n رأسی $\Omega(n^{۱۰})$ است، چون لازم است همه $\binom{n}{۱۰}$ حالت ممکن را تست کنیم.

پاسخ: استدلال مطرح شده نادرست است. صرف وجود k حالت ممکن برای جواب‌های یک مسئله به معنای نیاز برای چک کردن همه آنها نیست. برای مثال $n!$ حالت بالقوه برای خروجی‌های مسئله مرتب سازی وجود دارد، اما زمان اجرای الگوریتم‌های استاندارد مرتب‌سازی $o(n!)$ است.

(د) ساختمان داده‌ای وجود دارد که n عدد را در حافظه $O(n)$ ذخیره کند و با داده شدن یک عدد جدید، بتواند با احتمال ۱ در زمان $O(۱)$ تعیین کند که این عدد در بین n عدد اولیه هست یا خیر.

پاسخ: درست. درهم‌سازی تام.[۲]

[2] Perfect Hashing

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۲ موعد: سه‌شنبه ۶ اسفند ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با andishe.ghasemi.9@gmail.com مطرح کنید.

تمرین‌های پیشنهادی (غیرتحویلی)

CLRS 22.1.5, 22.1.6, 22.1.8, 22.2.1, 22.2.7, 33.4.3.

تمرین‌های تحویلی

۱. روی یک صفحه مختصات n نقطه مشخص شده‌است. به یک وضعیت از n نقطه خوب گوییم اگر به ازای هر جفت نقطه مثل (x, y) و (x', y') یکی از سه شرط زیر برقرار باشد:

۱) $x = x'$

۲) $y = y'$

۳) نقطه‌ی دیگری چون (x'', y'') موجود باشد به طوری‌که $\min(x, x') \le x'' \le \max(x, x')$ و $\min(y, y') \le y'' \le \max(y, y')$.

می‌خواهیم تعدادی نقطه به مجموعه نقاط اضافه کنیم تا در انتها به یک وضعیت خوب برسیم؛ الگوریتمی طراحی کنید که در زمان $O(n \lg n)$ به مقدار $O(n \lg n)$ نقطه به این نقاط اضافه کند طوری که در انتها یک وضعیت خوب از نقاط داشته‌باشیم.

راهنمایی برای پاسخ: اگر شرط اینکه نقطه‌ها متمایز باشند را نداشته‌باشیم، می‌توانیم دقیقا روی هر نقطه، نقطه‌ای قرار داده و در $O(n)$ مسئله را حل کنیم.

اما اگر شرط متمایز بودن نقاط را داشته‌باشیم، الگوریتمی ارائه می‌دهیم که با داشتن مجموعه نقاطی که بر حسب مولفه اول مرتب شده و همان مجموعه نقاط که بر حسب مولفه‌ی دوم مرتب شده، مسئله را حل می‌کند: فرض کنید n نقطه داریم که $(x_1, y_1), \cdots, (x_n, y_n)$ و $x_i \le x_{i+1}$ و همچنین همین مجموعه نقاط را به صورت $(x'_1, y'_1), \cdots, (x'_n, y'_n)$ و $y'_i \le y'_{i+1}$ داریم. اگر $n = 1$ باشد، مسئله حل است، فرض کنید x' میانه‌ی x_iها باشد، چون نقاط را مرتب شده داریم x' را با $O(1)$ داریم، پس حداکثر $\frac{n}{2}$ تا از نقاط دارای مولفه‌ی اول کمتر از x' و حداکثر $\frac{n}{2}$ از نقاط دارای مولفه‌ی اول بیشتر از x' هستند، این دو مجموعه را با $O(n)$ مجاسبه می‌کنیم و مسئله را به صورت بازگشتی برای قسمت سمت چپ و سمت راست حل می‌کنیم و دو مجموعه نقطه‌ی اضافه‌ی P_L و P_R به‌دست می‌آوریم.

حال آرایه‌ی A را برابر با مولفه‌ی دوم همه‌ی نقاطی که مولفه‌ی اول آن‌ها x' است در زمان $O(n)$ با فور زدن روی (x'_i, y'_i)ها می‌سازیم، پس این آرایه یک آرایه‌ی مرتب است. حال $R = \emptyset$ در نظر بگیرید، به ازای هر $i \in \{1, ..., n-1\}$، اگر $y'_i \ne y'_{i+1}$، نقطه‌ی $(x', \frac{y'_i + y'_{i+1}}{2})$ را در صورتی که $\frac{y'_i + y'_{i+1}}{2} \notin A$ به R اضافه می‌کنیم، با توجه به مرتب بودن A و مرتب بودن $\frac{y'_i + y'_{i+1}}{2}$ها، این کار را در مجموع با $O(n)$ می‌توانیم انجام دهیم.

حال $R \cup P_L \cup P_R$ را به عنوان نقاطی که باید اضافه‌کنیم خروجی می‌دهیم.

خب، الگوریتم ارائه شد، ۳ مورد می‌ماند:

(آ) اثبات درستی الگوریتم:
برای اثبات می‌توانیم از استقرا استفاده کنیم، برای $n = ۱$ درستی الگوریتم واضح است، فرض کنید الگوریتم برای همه‌ی n های کمتر از k درست است و برای $n = k$، زوج نقاط به چند دسته نقسیم می‌شوند: یا هردو مولفه‌ی اول کمتر از x' یا هر دو مولفه‌ی اول بیشتر از x' دارند که طبق استقرا نقطه‌ای برای آن اضافه کرده‌ایم. برای مولفه اول یکی کمتر از x' و مولفه اول دیگری بزرگتر یا مساوی x'، اگر مولفه‌ی دوم این دو نقطه مساوی باشد، سطر دوم برقرار است و در غیر اینصورت طبق عمل آخر، نقطه‌ای با مولفه‌ی اول x' و مولفه‌ی دوم بین این‌دو اضافه کردیم که شرط سوم برای آن برقرار است و برای حالت چهارم نیز مشابه قبل است و در نتیجه این وضعیت یک وضعیت خوب است و گام استقرا ثابت و حکم استقرا ثابت می‌شود.

(ب) اثبات زمان اجرا: فرض کنید $T(n)$ یک کران‌بالا برای زمان اجرای الگوریتم روی n نقطه باشد که تابعی صعودی است، داریم:

$$T(n) = ۲ \times T(\frac{n}{۲}) + O(n)$$

که می‌دانیم $T(n) = O(n \lg n)$ است.

(ج) اثبات کم بودن تعداد نقاط اضافه‌شده:
مشابه قسمت قبل با استقرا روی n و مشابه قسمت قبل نتیجه می‌شود.

حال ابتدا نقطه‌ها را یک‌بار بر حسب مولفه‌ی اول و یک‌بار بر حسب مولفه‌ی دوم در زمان $O(n \lg n)$ مرتب می‌کنیم و سپس با الگوریتم معرفی شده مسئله را در زمان $O(n \lg n)$ حل می‌کنیم.

۲. برای یک عدد b که $۲ \leq b \leq ۱۰$ است، عدد خوب را عددی طبیعی تعریف می‌کنیم که تمام ارقام آن کمتر از b باشد، مثلا برای $b = ۲$، تمام اعداد با ارقام ۰ و ۱ خوب هستند.
برای n و b داده‌شده، می‌خواهیم با کمک مدل‌سازی اعداد در گراف و استفاده از نسخه‌ای تغییر یافته از «الگوریتم جستجوی اول سطح» کوچکترین عدد خوب مضرب n را پیدا کنیم؛ مثلا برای $n = ۳, b = ۲$ کوچکترین عدد خوب مضرب ۳ برابر با ۱۱۱ است.

(آ) سعی کنید با طراحی یک گراف مناسب، الگوریتمی از $O(n)$ ارائه دهید تا با گرفتن n, b کوچکترین عدد خوب مضرب n را بیابد.
راهنمایی: به ازای باقی‌مانده‌ی هر عدد خوب بر n، یک راس در نظر بگیرید.
راهنمایی برای پاسخ: یک آرایه n تایی مثل A در نظر می‌گیریم و ابتدا تمام مقادیر آن را ۰ قرار می‌دهیم. یک گراف فرضی در نظر بگیرید که راس‌های آن اعداد خوب است و از راس متناظر عدد x به راس‌های متناظر با اعداد $x۰, x۱, ..., x(b-۱)$ یال جهت‌دار وجود دارد. همچنین برای عدد ۰ نیز یک راس درنظر بگیرید که به راس‌های $۱, ..., b-۱$ یال جهت دار دارد.
الگوریتم BFS را از راس ۰ شروع کنید با این تفاوت که برای علامت‌گذاری به عنوان دیده‌شده از آرایه‌ی A استفاده کنید و برای اضافه‌کردن راس x به صف BFS ، مقدار $x \mod n$ را در آرایه‌ی A یک کنیم و اگر یک بود x را به صف اضافه نکنیم، فقط ۰ را از این قاعده مستثنی کنیم و به اولین راس متناظر عددی مثل x رسیدیم که $x \mod n = ۰$ بود، x را به عنوان جواب خروجی دهیم، همچنین برای راس x همسایه‌های آن را به ترتیب یعنی ابتدا $x۰$، سپس $x۱$ و ... و در آخر $x(b-۱)$ را مورد بررسی برای اضافه‌کردن به صف قرار می‌دهیم.
در واقع، اگر به راسی با باقی‌مانده‌ی تکراری بر n رسیدیم، الگوریتم BFS را از آن راس ادامه نمی‌دهیم.
باید ثابت کنیم که زمان اجرای الگوریتم $O(n)$ است و همچنین خروجی کمترین عدد ممکن است.
با توجه به اینکه هر یال معادل اضافه‌شدن یک رقم است و با توجه به اجرای الگوریتم BFS، پاسخ کوچکترین طول را دارد، همچنین با توجه به اینکه همسایه‌ها را به ترتیب مشاهده کردیم، می‌توان نشان‌داد که به ازای هر باقی‌مانده، کوچکترین‌عدد، اولین مشاهده‌ی ما از راس‌ها در آن باقی‌مانده است.
همچنین زمان اجرای الگوریتم از $O(n)$ است، گرافی که این الگوریتم راس‌ها و یال‌های آن را بازدید می‌کند در نظر بگیرید،

الگوریتم حداکثر n بار راس با باقی‌مانده‌ی جدید می‌بیند و k بار راس باقی‌مانده‌ی تکراری می‌بیند، همچنین راس‌های با باقی‌مانده‌ی تکراری یال ورودی دارند و هیچ یال خروجی ندارند و راس‌های غیرتکراری b یال خروجی دارند، پس در این گراف درجه‌ی هر راس حداقل یک و تعداد یال‌ها برابر با مجموع درجه‌های خروجی و حداکثر bn است، پس تعداد راس‌های گراف از $O(n)$ و تعداد یال‌های گراف نیز از $O(n)$ است و در مجموع الگوریتم BFS به زمان $O(n)$ طول می‌کشد.

(ب) فرض کنید x یک عدد خوب است، همزاد این عدد را عددی مثل y تعریف می‌کنیم که $(x)_b = (y)_{10}$ (منظور از اندیس مبنا است.) مثلا برای $b = 2$ همزاد عدد خوب ۱۱۱ عدد ۷ است.

الگوریتمی از زمان $O(n^2)$ ارائه دهید که با ورودی گرفتن n, b کوچکترین عدد خوب که هم خودش و هم همزادش مضرب n باشد بیابید.

راهنمایی: به ازای هر زوج مرتب باقی‌مانده‌ی عدد خوب بر n و همزادش بر n، یک راس در نظر بگیرید.

راهنمایی برای پاسخ: دقیقا مشابه الف رفتار کنید، ولی اینبار آرایه‌ی A را یک آرایه‌ی دوبعدی $n \times n$ در نظر بگیرید و برای علامت‌گذاری به عنوان دیده شده از باقی‌مانده‌ی x بر n و همزادش بر n استفاده کنید.

بقیه‌ی روند الگوریتم مشابه قسمت قبل است.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۳

---

۱. درختی با n رأس داده شده‌است. الگوریتمی طراحی کنید که در زمان $O(n)$:

(آ) طولانی ترین مسیر را پیدا کند.

(ب) کوتاه‌ترین گشت گذرنده از تمام رأس‌ها را بدست آورد.

پاسخ:

(آ) یک رأس دلخواه از درخت مانند A را در نظر می‌گیریم. با استفاده از BFS دورترین رأس نسبت به A را پیدا می‌کنیم. (اگر بیش‌تر از یک دورترین رأس وجود داشت یکی از آن‌ها را به دلخواه انتخاب می‌کنیم.) فرض کنید این راس B باشد. ادعا می‌کنیم مسیری با بیش‌ترین طول وجود دارد که یک‌سرش B باشد، در این‌صورت با استفاده از یک BFS دیگر می‌توانیم طولانی‌ترین مسیر را پیدا کنیم. از آن‌جا که پیچیدگی زمانی BFS، $O(m)$ است و برای هر درخت $m = n - 1$، نتیجه می‌گیریم که الگوریتم در زمان $O(n)$ اجرا می‌شود.

حال ادعای بیان‌شده را ثابت می‌کنیم. مسیر بین دو رأس X و Y را با $p(X, Y)$ نشان می‌دهیم. فرض کنید C و D دو سر بلندترین مسیر باشند که هیچ‌کدام رأس B نیز نیستند زیرا در غیر این‌صورت چیزی برای اثبات وجود ندارد. ابتدا فرض می‌کنیم A روی $p(C, D)$ باشد. توجه کنید که $p(A, B)$ حداکثر با یکی از $p(A, C)$ و $p(A, D)$ می‌تواند یال مشترک داشته باشد چون در غیر این‌صورت در گراف دور ایجاد می‌شود. پس بنابر تقارن فرض می‌کنیم با $p(A, D)$ یال مشترک ندارد. اگر در مسیر $p(C, D)$ به‌جای $p(A, C)$، $p(A, B)$ را قرار دهیم، مسیری با همان طول قبلی، $p(C, D)$ یا طولانی‌تر به‌دست می‌آید که یک‌سر آن B است پس در این حالت ادعا ثابت می‌شود و فرض می‌کنیم A روی $p(C, D)$ نباشد. اگر $p(A, B)$، $p(C, D)$ را برای بار اول در E و برای آخرین بار در F قطع کند، طوری‌که در $p(C, D)$ نقاط C، F و E به همین ترتیب ظاهر شوند، مسیری که از اتصال $p(B, F)$ و $P(F, D)$ تشکیل می‌شود طولی بزرگ‌تر یا مساویِ طول $p(C, D)$ دارد زیرا طول $p(A, B)$ از طول $p(A, C)$ کم‌تر نیست. تنها حالتی می‌ماند که $p(A, B)$، $p(C, D)$ را قطع نکند. فرض کنید $p(A, C)$، $p(C, D)$ را برای بار اول در E و $p(B, A)$، $p(A, C)$ را برای بار اول در F قطع کند. در این‌صورت مسیری که از اتصال $p(B, F)$، $p(F, E)$ و $p(E, D)$ به وجود می‌آید مانند حالت قبل طولی بزرگ‌تر یا مساویِ طول $p(C, D)$ دارد. پس در همهٔ حالات ادعا ثابت شد.

(ب) در یک گشت به یال‌هایی که دقیقاً یک بار ظاهر شده‌اند، خوب می‌گوییم. می‌خواهیم گشتی پیدا کنیم که از همهٔ رئوس بگذرد، از هر یال حداکثر دو بار بگذرد و تعداد یال‌های خوب آن بیش‌ترین مقدار ممکن را داشته باشد. واضح است که اگر چنین گشتی پیدا کنیم، کوتاه‌ترین طول ممکن را دارد. فرض کنید رئوس A و B دو سر یک گشت باشند. تعداد یال‌های خوب در این گشت نمی‌تواند بیش‌تر از تعداد یال‌های مسیر بین A و B باشد زیرا در غیر این‌صورت به یک مسیر طولانی‌تر بین A و B می‌رسیم که امکان ندارد. (در درخت بین هر دو رأس دقیقاً یک مسیر وجود دارد.) پس تعداد یال‌های خوب زمانی بیش‌ترین مقدار خود را می‌گیرند، که A و B دو سر طولانی‌ترین مسیر باشند. حال برای ساختن گشت یال‌ها و رئوس $p(A, B)$ را علامت می‌زنیم و به هر رأس علامت‌خورده رسیدیم ابتدا روی یال‌هایی که علامت نخورده‌اند حرکت می‌کنیم تا به برگ برسیم. سپس روی همان یال‌ها بر می‌گردیم تا باز به همان رأس علامت‌خورده برسیم. این کار را تکرار می‌کنیم تا فقط

یال علامت‌خورده باقی بماند و بالاخره روی آن حرکت می‌کنیم. دقت کنید که در این گشت همهٔ یال‌های $p(A, B)$ خوب هستند و یال‌های دیگر همه دقیقاً دو بار ظاهر شده‌اند پس گشت ساخته‌شده کوتاه‌ترین گشت است.

طبق قسمت (آ) پیدا کردن رئوس A و B هزینهٔ $O(n)$ دارد، ساختن گشت نیز به وضوح هزینهٔ $O(n)$ دارد زیرا هر یال را حداکثر دو بار طی کرده‌ایم و $m = n - 1$. در نتیجه هزینهٔ کل الگوریتم $O(n)$ است.

۲. گوییم زوج مرتب (a, b) می‌تواند از زوج مرتب (c, d) برنده شود، اگر $a \geq c$ یا $b \geq d$.

n زوج عدد به صورت a_i و b_i به همراه m رابطه‌ی برنده‌شدن به صورت اینکه کدام زوج می‌تواند کدام زوج را ببرد داریم.

می‌خواهیم هر کدام از این زوج‌ها را به صورت زوج مرتب (a_i, b_i) یا (b_i, a_i) تبدیل کنیم به طوری که m رابطه‌ی برنده‌شدن بین آن‌ها برقرار باشد.

الگوریتمی از $O(n + m)$ ارائه دهید که بگوید این‌کار ممکن است یا خیر.

راهنمایی: می‌توانید از مسئله 2SAT یا از روش تشخیص دوبخشی بودن گراف استفاده کنید.

پاسخ: هر زوج مرتب را یک رأس و هر رابطهٔ برنده‌شدن را یک یال در نظر می‌گیریم. دقت کنید که در گراف یال جهت‌دار نمی‌کشیم و برای هر رأس مشخص می‌کنیم که باید از همسایه‌اش ببرد یا نه. یک مؤلفهٔ همبندی گراف را در نظر می‌گیریم. تغییرات این مؤلفهٔ همبندی روی بقیهٔ گراف تاثیری نمی‌گذارد پس می‌توانیم از ابتدا فرض کنیم فقط یک مؤلفهٔ همبندی داریم. تغییر (a, b) به (b, a) را *برعکس کردن* می‌نامیم. توجه کنید که اگر یک حالت مطلوب (حالتی که همهٔ روابط برنده‌شدن برقرار باشد) داشته باشیم با برعکس کردن همهٔ رئوس همچنان یک حالت مطلوب داریم پس می‌توانیم در ابتدا وضعیت یک رأس را ثابت در نظر بگیریم و روی آن قفل بزنیم، به این معنی که این رأس را نمی‌توانیم برعکس کنیم. حال روی این رأس DFS می‌زنیم. در هر مرحله اگر از رأس u به رأس v رفتیم در صورتی‌که روی حداقل یکی از u و v قفل نباشد می‌توانیم آن‌ها را تغییر دهیم تا رابطهٔ برنده‌شدن برقرار شود. همچنین اگر پس از برقراری رابطهٔ برنده‌شدن همچنان با برعکس‌شدن یکی از u و v رابطهٔ برنده‌شدن برقرار می‌ماند، روی آن قفل نمی‌زنیم اما اگر برعکس‌شدن یکی از دو رأس رابطهٔ برنده‌شدن را از بین ببرد روی آن قفل می‌زنیم. اگر DFS به‌طور کامل انجام شود یعنی این کار امکان‌پذیر است و اگر جایی نتوانیم رابطهٔ برنده‌شدن را برقرار کنیم (به علت قفل‌شدن بعضی رئوس) یعنی این کار امکان‌پذیر نیست. به وضوح پیچیدگی زمانی این الگوریتم همان پیچیدگی زمانی DFS یعنی $O(m + n)$ است.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

[بهار ۹۹]

تمرین سری ۴ موعد: سه‌شنبه ۲۰ اسفند ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با andishe.ghasemi.9@gmail.com , javadakbari1379@gmail.com مطرح کنید.

تمرین‌های پیشنهادی

حالت وزن‌دار مسئله زمان‌بندی بازه‌ها را با برنامه‌ریزی پویا حل کنید.

CLRS Exercises 16.4.1, 23.1.1, 23.1.2, 23.1.3, 23.1.8.

CLRS Problems 15.1, 16.1.

تمرین‌های تحویلی

۱. قصد داریم که در یک مسیر بی انتها(!) تا جای ممکن پیش برویم. در ابتدای مسیر n کامیون داریم که ظرفیت بنزین آن‌ها تکمیل است. حداکثر ظرفیت بنزین هر کامیون یک لیتر است و مصرف سوخت هر کامیون برای طی‌کردن یک کیلومتر از مسیر برابر یک لیتر است. همچنین می‌توانیم در هر جای مسیر هر کامیونی را به دلخواه نگه‌داشته و مقداری از بنزین آن را به کامیونی دیگر منتقل‌کنیم. حداکثر مسیری که می‌توانیم پیش برویم چقدر است؟(با ذکر دلیل و اثبات درستی)

راهنمایی برای پاسخ: ابتدا همه‌ی n کامیون حرکت می‌کنند و بعد از طی کردن $\frac{۱}{n}$ کیلومتر متوقف می‌شوند. کامیون n ام در باک بنزین خود $\frac{n-۱}{n}$ بنزین دارد و به هر کدام از $n-۱$ کامیون دیگر $\frac{۱}{n}$ لیتر بنزین منتقل می‌کند تا بنزینش تمام شود و حذف می‌شود. حال الگوریتم را روی $n-۱$ کامیون باقی‌مانده با باک بنزین پر تکرار می‌کنیم. اگر $n=۱$ باشد یک کیلومتر را طی می‌کند. پس جواب به صورت $\frac{۱}{n}+...+\frac{۱}{۲}+۱$ است. فرض کنید الگوریتم بهینه‌ای وجود داشته باشد که در آن اولین کامیونی که حذف می‌شود k کیلومتر طی کرده باشد. اگر $k<\frac{۱}{n}$ باشد، هنگام حذف شدن این کامیون ظرفیت خالی باک بنزین کامیون‌های دیگر $(n-۱)k$ لیتر است و بنزین باقی مانده برای این کامیون $(۱-k)$ لیتر است.

$$k<\frac{۱}{n}\Rightarrow nk<۱\Rightarrow nk-k<۱-k\Rightarrow (n-۱)k<۱-k$$

مقداری از بنزین این کامیون استفاده نمی‌شود و بدون کم کردن از جواب بهینه می‌توانیم این کامیون را تا $\frac{۱}{n}$ متوقف نکنیم. اگر $k>\frac{۱}{n}$ باشد، در لحظه‌ای که در کیلومتر k ام هستیم، هر کامیون $۱-k$ لیتر بنزین دارد و در مجموع $n(۱-k)$ لیتر بنزین داریم. حال اگر در کیلومتر $\frac{۱}{n}$ متوقف شویم و بنزین‌ها را منتقل کنیم، در کیلومتر k ام مسیر مقدار $(n-۱)(۱-k+\frac{۱}{n})=(n-۱)-(n-۱)(k-\frac{۱}{n})$ بنزین خواهیم داشت.

$$k\le ۱\Rightarrow ۱-k\ge ۰\Rightarrow ۱-k+\frac{۱}{n}>۰$$

$$k>\frac{۱}{n}\Rightarrow nk>۱\Rightarrow n-۱>n-nk\Rightarrow (n-۱)(۱-k+\frac{۱}{n})>n(۱-k)$$

پس با اعمال الگوریتم ما، $n-۱$ کامیون با بنزین بیشتری در کیلومتر k ام مسیر خواهیم داشت و بدون کم شدن از الگوریتم بهینه می‌توانیم اولین کامیون را در $k=\frac{۱}{n}$ متوقف کنیم. به صورت استقرایی اثبات می‌شود که بیشینه مسیر قابل پیمایش برای $n-۱$ کامیون دیگر نیز $\frac{۱}{n-۱}+...+\frac{۱}{۲}+۱$ است.

۲. قصد داریم با یک ماشین که با حداکثر ظرفیت بنزینش d کیلومتر حرکت می‌کند به یک سفر دور کشور برویم. از قبل همه‌ی n پمپ‌بنزین موجود در طول مسیرمان را روی نقشه پیدا کرده‌ایم.فاصله‌ی بین پمپ بنزین‌های متوالی بیشتر از d کیلومتر نیست.می‌خواهیم برنامه سفر را طوری بچینیم که کمترین توقف‌های ممکن را برای سوخت‌گیری داشته‌باشیم. الگوریتمی طراحی کنید که با گرفتن d و فاصله‌ی پمپ‌بنزین‌ها از نقطه شروع، لیست پمپ‌بنزین‌هایی که باید در آن ها توقف کنیم را در زمان $O(n)$ خروجی دهد.

راهنمایی برای پاسخ:

از شهر اول شروع به حرکت می‌کنیم و در هر شهر به پمپ بنزین می‌رویم اگر بنزین کافی برای رسیدن به شهر بعد را نداشته باشیم. فرض کنید در شهرهای $c_1, c_2, ..., c_n$ بنزین زده‌باشیم. و الگوریتم بهینه‌ای وجود داشته باشد که در شهرهای کمتری بنزین زده باشد مثل $c'_1, c'_2, ..., c'_m$. کوچکترین اندیسی را در نظر بگیرید که داشته باشیم $c_i \neq c'_i$. می‌دانیم که شهر c'_i نمی‌تواند بعد از شهر c_i باشد، چون اگر در شهر c_i بنزین نزده‌باشیم بنزین کافی برای رسیدن به شهر بعدی را نخواهیم داشت. اگر c'_i شهری قبل از شهر c_i باشد، الگوریتم بهینه موقع رسیدن به شهر c_i مجموعا i بار توقف داشته است و در این شهر باک بنزینش کامل پر نیست. ولی در الگوریتم حریصانه‌ی ما در شهر c_i باک بنزین پر بوده و مجموعا i بار بنزین زده ایم. پس با تغییر c'_i به c_i همچنان الگوریتم بهینه می‌ماند.

۳. در یک تیم والیبال n والیبالیست داریم. قد همه‌ی آن‌ها بین ۱۹۵ سانتی‌متر و ۲۰۵ سانتی‌متر بوده و میانگین قد آن‌ها ۲۰۰ سانتی‌متر است. می‌خواهیم همه بازیکنان به ترتیبی در یک ردیف بایستند. اگر مربی تیم دو بازیکن را به دلخواه انتخاب کند که k بازیکن دیگر بین آن‌ها باشند ، اختلاف مجموع قد این $k+2$ بازیکن از مقدار $200 \times (k+2)$ را حساب کرده و اگر این مقدار بیشتر از ۱۰ سانتی‌متر باشد از تیم خود ناامید می‌شود.الگوریتمی حریصانه برای پیدا کردن ترتیبی از بازیکنان ارائه دهید که مربی را ناامید نکند.
راهنمایی برای پاسخ: ابتدا از قد هر نفر مقدار ۲۰۰ را کم می‌کنیم که با فاصله‌ی آن‌ها از میانگین که در بازه‌ی $[-5, 5]$ کار کنیم. سپس اعداد را به دو گروه منفی و نامنفی تقسیم می‌کنیم. یک عدد دلخواه را قرار می‌دهیم و هربار در مرحله i ام اگر جمع اعداد از اول تا i کمتر از صفر بود، عددی نامنفی و اگر این مجموع بیشتر مساوی صفر بود عددی منفی قرار می‌دهیم. ادعا می‌کنیم همواره چنین عددی وجود دارد زیرا میانگین قد ها ۲۰۰ است و مجموع تفاضل قدها از میانگین برابر با صفر است. پس اگر مجموع منفی است عددی نامنفی و اگر مجموع مثبت است عددی منفی هنوز وجود دارد. پس به صورت استقرایی ثابت می‌شود که برای هر بازه از ابتدا تا نفر i ام مجموع عددی در بازه‌ی $[-5, 5]$ است.

$$-5 \leq a_1 + a_2 + ... + a_i \leq 0, 0 \leq a_{i+1} \leq 5 \Rightarrow -5 \leq a_1 + a_2 + ... + a_{i+1} \leq 5$$

$$0 \leq a_1 + a_2 + ... + a_i \leq 5, -5 \leq a_{i+1} \leq 0 \Rightarrow -5 \leq a_1 + a_2 + ... + a_{i+1} \leq 5$$

حال به ازای هر بازه‌ی دلخواه از نفر i ام تا نفر j ام، این مجموع برابر تفاضل مجموع بازه‌ی ابتدا تا نفر i ام و مجموع بازه‌ی ابتدا تا نفر j ام خواهد بود.

$$-5 \leq a_1 + a_2 + ... + a_i \leq 5$$

$$5 \leq a_1 + a_2 + ... + a_{i-1} + a_i + a_{i+1} + ... + a_j \leq 5$$

$$\Rightarrow -10 \leq a_{i+1} + a_{i+2} + ... + a_j \leq 10$$

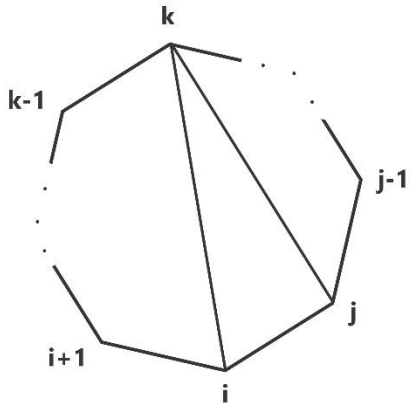
تمرین امتیازی

فرض کنید گراف $G = (V, E)$ با دو تابع وزن $c_1 : E \to Q_+$ و $c_2 : E \to Q_+$ داده شده باشد. الگوریتمی چندجمله‌ای ارائه دهید که تشخیص دهد آیا G زیردرخت فراگیری دارد که هم‌زمان برای c_1 و c_2 کمینه باشد.
راهنمایی: تعمیم مسئله برای ماتروییدها را حل کنید.

موفّق باشید.

۱

نقاط روی چندضلعی محدب را به صورت ساعت‌گرد از ۱ تا n نام‌گذاری می‌کنیم. حال تابع C را این‌گونه تعریف می‌کنیم که $C(i.j)$ مثلث‌بندی بهینه‌ای می‌باشد که می‌توان با مجموعه $\{i.i+۱.\ldots.j\}$ ساخت. جواب مسئله $C(۱.n)$ می‌باشد.
اگر چند ضلعی محدب i، $i+۱$ ... و j را داشته‌باشیم، در هر حالت ضلع ij در یک مثلث قرار می‌گیرد، چون اگر بخواهد در بیش از یک مثلث قرار بگیرد، در این صورت ضلع‌های آن دو مثلث هم‌دیگر را قطع می‌کنند با توجه به این‌که شکل ما محدب می‌باشد. پس راس دیگر آن می‌تواند $j > k > i$ باشد.



حال می‌توانیم ضلع ij را حذف کنیم و در زیرمسئله‌های $C(k.j)$ و $C(i.k)$ به دنبال مثلث‌بندی بهینه بگردیم. بنابراین $C(i.j)$ برابر عبارت زیر می‌شود:

$$C(i.j)=\begin{cases}|ij|. & j=i+۱\\ |ij|+\min\{C(i.k)+C(k.j)\mid \forall k:\ j>k>i\}. & j>i+۱\end{cases}$$

هر مرحله $O(n)$ زمان می‌گیرد و باید برای $O(n^۲)$ خانه این تابع این مرحله را اجرا کنیم پس اردر زمانی $O(n^۳)$ زمان می‌گیرد.

۲

می‌توان ابتدا به دنبال بزرگ‌ترین مربع آبی گشت و دقیقاً همین مراحل را برای پیدا کردن بزرگ‌ترین مربع قرمز دنبال کرد و از مقایسه این دو فهمید که بزرگ‌ترین زیر مربع کدام است. پس ابتدا فرض می‌کنیم به دنبال مربع آبی هستیم و رنگ آبی با ۱ و رنگ قرمز با ۰ مشخص شده است.

ماتریس اولیه را M می‌نامیم و سپس یک ماتریس هم اندازه با M در نظر می‌گیریم و آن را S می‌نامیم. $S(i.j)$ نشان‌دهنده طول ضلع بزرگ‌ترین زیر مربعی که i و j درایه پایین و سمت راست آن در ماتریس M است، می‌باشد. حال اگر $M(i.j)=۱$ باشد، مقدار $(i.j)$ ماتریس S به صورت زیر تعیین می‌شود:

$$S(i.j)=\min\{S(i-۱.j).S(i-۱.j-۱).S(i.j-۱)\}+۱$$

در صورتی که $M(i.j) = ۰$ برابر صفر باشد، $S(i.j) = ۰$ می‌شود. بزرگ‌ترین درایه S می‌شود طول ضلع بزرگ‌ترین زیر مربع آبی در ماتریس M. برای رنگ قرمز هم مراحل مشابه را تکرار می‌کنیم و بین دو مقدار به‌دست آمده بزرگ‌ترین را انتخاب می‌کنیم.

می‌خواهیم اثبات کنیم $S(i.j)$ که طول ضلع بزرگ‌ترین زیر مربعی که i و j درایه پایین و سمت راست آن در ماتریس M است را به ما می‌دهد.

در صورتی که $S(i.j) = ۰$ باشد، چون $M(i.j) = ۰$ است پس بزرگ‌ترین زیرمربع طول صفر دارد.

در صورتی که $S(i.j) = k$ باشد، روی k استفرا می‌زنیم.

برای $k = ۱$ داریم عبارت $\min\{S(i - ۱.j).S(i - ۱.j - ۱).S(i.j - ۱)\}$ را که برابر صفر شده است. که این معنی را می‌دهد که در یکی از همسایه‌های آن رنگ قرمز مشاهده شده است. پس نمی‌توانیم زیر مربعی با طول ۲ داشته باشیم.

گام استقرا به این صورت است که اگر به ازای k، $S(i.j)$ تعداد درست را نتیجه بدهد برای $k + ۱$ باید اثبات کنیم. زمانی که $k + ۱$ را داریم باید عبارت $\min\{S(i - ۱.j).S(i - ۱.j - ۱).S(i.j - ۱)\}$ برابر k شده باشد. که طبق فرض استقرا درست به جواب رسیده است. معنی این عبارت به این صورت است که در هر همسایه $S(i.j)$ یک زیر مربع با طول حداقل k دیده شده است. پس می‌توان از درایه $(i.j)$ حداکثر به طول $k + ۱$ مربع داشت و $S(i.j) = k + ۱$ می‌شود.

این الگوریتم بخاطر این که یک جدول $m \times n$ را پر می‌کند، در زمان $O(mn)$ انجام می‌پذیرد.

۳

یکی از رأس‌های این درخت را انتخاب کرده و این رأس ریشه را r بنامید و روی آن BFS را اجرا می‌کنیم تا فرزندان هر رأس به‌دست بیایند. حال برای هر رأس x دلخواه تعریف می‌کنیم:

$A(x) :=$ کم‌ترین تعداد چراغ‌های لازم برای روشن کردن خیابان‌های زیر درخت با ریشه x، اگر در x چراغ باشد.

$B(x) :=$ کم‌ترین تعداد چراغ‌های لازم برای روشن کردن خیابان‌های زیر درخت با ریشه x، اگر در x چراغ نباشد.

$A'(x) :=$ تعداد حالاتی که می‌توان خیابان‌های شهر را با $A(x)$ چراغ روشن کرد، اگر در x چراغ باشد.

$B'(x) :=$ تعداد حالاتی که می‌توان خیابان‌های شهر را با $B(x)$ چراغ روشن کرد، اگر در x چراغ نباشد.

زمانی که در در رأس x چراغ نباشد، در فرزندان این رأس حتماً باید چراغ وجود داشته باشد ولی زمانی که در رأس x چراغ باشد، در فرزندان این رأس آزادی در انتخاب چراغ داریم. برای هر فرزند v رأس دلخواه x، عبارات زیر محاسبه می‌شوند:

$$A(x) = ۱ + \sum \min\{A(v).B(v)\}$$

$$B(x) = \sum \mathrm{A(v)}$$

$$A'(x) = \prod \min\{A'(v).B'(v)\}$$

$$B'(x) = \prod A'(v)$$

به‌طور بازگشتی این عبارات را محاسبه می‌کنیم، هر رأس حداکثر دو بار بررسی می‌شود و زمانی که همه رأس ها را بررسی کنیم الگوریتم تمام می‌شود. پس مرتبه زمانی این الگوریتم $O(n)$ می‌باشد.

حال برای جواب نهایی مسأله اگر $B(r) > A(r)$ باشد، آنگاه با $A'(x)$ حالت می‌توانیم خیابان را روشن کنیم. اگر $B(r) < A(r)$ آنگاه با $B'(x)$ حالت می‌توانیم خیابان را روشن کنیم. و درصورتی که این دو باهم برابر باشند به $A'(r) + B'(r)$ می‌توانیم خیابان را روشن کنیم. چون در این حالت برای کمینه بودن تعداد لامپ‌ها فرقی نمی‌کند که در ریشه لامپ باشد یا نباشد.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

[بهار ۹۹]

تمرین سری ۶ موعد: سه‌شنبه ۱۲ فروردین ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با alirtofighim@gmail.com, javadakbari1379@gmail.com مطرح کنید.

تمرین‌های تحویلی

۱. یک گراف وزن‌دار و جهت‌دار V راسه و E یاله داریم که فاقد دور منفی است، روی این گراف Q امر مطرح می‌شود، هر امر یکی از دو نوع زیر است:

- v را ورودی داده و از ما می‌خواهند راس v را از گراف پاک کنیم.
- دو راس v, u را ورودی داده و وزن کوتاه‌ترین مسیر (از نظر مجموع وزن یال‌ها) از v به u را از ما می‌خواهند.

متاسفانه در لحظه‌ی مطرح شدن این امور، ما قادر به پاسخ‌دهی به آن‌ها نبودیم و حال همه‌ی Q امر را یک‌جا داریم، الگوریتمی ارائه دهید که با گرفتن گراف و Q امر در زمان اجرای $O(V^3 + Q)$ به عنوان خروجی پاسخ امرهای نوع دوم را بدهد.

پاسخ:

در مجموعه دستورات Q ، دستورات مربوط به حذف یک رأس را به ترتیب $q_1, q_2, \ldots, q_l$ بنامید و بقیه دستورات را نیز $q'_1, q'_2, \ldots, q'_{Q-l}$ (به ترتیب) بنامید.الگوریتم فلوید_وارشال را اجرا می‌کنیم ولی برای اینکه جواب دستورات q'_i را بتوانیم خروجی دهیم($1 \leq i \leq Q-l$) ، می‌بایست رأس‌ها را به شکلی خاص از ۱ تا $|V|$ شماره‌گذاری کنیم.ابتدا اعداد ۱ تا $|V|-l$ را به رئوسی که تحت هیچ دستور q_i ($1 \leq i \leq l$) حذف نمی‌شوند نسبت دهیم.سپس به آخرین رأسی که(تحت q_l) حذف می‌شود، شماره $|V|-l+1$ نسبت داده و به همین ترتیب به رأسی که تحت دستور q_i حذف می‌شود، ($1 \leq i \leq l$) شماره‌ی $|V|-l+(l+1-i)$ نسبت دهیم که همان شماره‌ی $|V|-i+1$ است. به نحوه تعریف $d^k_{u,v}$ در الگوریتم فلوید_وارشال دقت کنید.

$d^k_{u,v}$ = کوتاهترین مسیر از u به v طوری‌که رئوس این مسیر، رأس‌هایی با شماره ۱ تا k باشند.

پیاده سازی الگوریتم فلوید_وارشال نیز طوری‌است که برای یک k ثابت ،به ازای هر u, v به مقدار $d^k_{u,v}$ دست یافته، سپس به سراغ $k = k+1$ می‌رویم. بنابراین زمانی که $k = |V| - l$ باشد،می‌توانیم به همه‌ی $query$ های بعد از q_l که در واقع حالتی است که هیچ یک از رئوس حذف شده توسط q_1 تا q_l حضور ندارند،پاسخ درست بدهیم.

پس همینطور که جلوتر می‌رویم و مثلا $k = |V| - l + (l+1-i) = |V| - i + 1$ است، می‌توانیم به همه‌ی $query$ های بعد از q_{i-1} و قبل از q_i پاسخ درست دهیم.چون $d^k_{u,v}$ ای که محاسبه می‌شود،کمترین فاصله‌ی بین u, v است که از رئوسش شماره ای بین ۱ تا $k = |V| - i + 1$ دارند. یعنی رئوسی که تحت q_1 تا q_{i-1} حذف شده اند، اکیداً در این مسیر نبوده و بقیه‌ی رئوس در محاسبه $d^k_{u,v}$ برای این k خاص در نظر گرفته شده‌اند .یعنی دقیقاً به $query$ های بین q_{i-1}, q_i که به فرم q'_j آن‌ها را معرفی کردیم، می‌توانیم پاسخ درست دهیم و بعد سراغ k بعدی برویم.

نهایتا برای تحلیل زمانی توجه داریم که در $O(V + E + Q)$ ورودی‌ها را می‌گیریم. در $O(Q + V)$ می‌توانیم رأس‌ها را با توجه به q_i ها شماره گذاری کنیم .(به رأسی که در مرحله q_i حذف می‌شود، شماره $|V| - i + 1$ دهیم و رئوس باقی‌مانده هم به ترتیب

دلخواه شماره دهیم. الگوریتم فلوید ـ وارشال نیز با توجه به تغییری که دادیم، در زمان $O(V^3+Q)$ انجام می‌شود.چون تنها تفاوت این است که می‌بایست در $O(1)$ ، قبل از اینکه به سراغ k بعدی برود،پاسخ q_i' های متناظر آن مرحله را بدهد.چون تعداد آن نیز $O(Q)$ است، زمان این قسمت $O(V^3+Q)$ می‌شود که در مجموع زمان اجرای کل الگوریتم همان $O(V^3+Q)$ خواهد بود.

۲. یک گراف وزن‌دار V راسه و E یاله با وزن یال‌های نامنفی داریم. شخصی روی راس s و شخص دیگری روی راس e ایستاده‌است، عبور از هر یال e دقیقاً یک واحد زمان می‌برد و به اندازه‌ی وزن یال e هزینه دارد. دو شخص مسئله‌ی ما از یک دیگر متنفرند، به همین دلیل نمی‌خواهند لحظه‌ای باشد که هر دو روی یک راس قرار گرفته‌باشند (ولی می‌توانند در یک لحظه در دو سر یک یال باشند و لحظه‌ی بعد جای خود را عوض کنند و ملاقات در وسط یال اشکالی ندارد)، هر دو انسان‌های عجولی هستند به همین دلیل به محض رسیدن به راسی از آن به راس دیگری حرکت می‌کنند و توقفی روی راس‌ها ندارند.
شخص اول می‌خواهد از راس s به راس e و شخص دوم می‌خواهد از راس e به راس s برسد، الگوریتمی طراحی کنید که با گرفتن گراف و راس‌های s و e، کمترین هزینه‌ای که این دو شخص در مجموع برای این جابه‌جایی باید بدهند را حساب کند.

(آ) زمان اجرای الگوریتم از $O(V^2lgV+E^2)$ باشد.

(ب) زمان اجرای الگوریتم از $O(V^2lgV+VE)$ باشد.

راهنمایی. سعی کنید گراف جدیدی با V^2 راس بسازید و پاسخ مسئله‌ی بالا را به یک مسئله‌ی کوتاه‌ترین مسیر در گراف جدید تبدیل کنید.

پاسخ:

(آ) گراف G' را با توجه به گراف اولیه(G که در مسئله داده شده) می‌سازیم.فرض کنید رئوس گراف اولیه را با $V=\{v_1,v_2,\dots,v_n\}$ نمایش دهیم.رئوس گراف G' را زوج مرتب‌های (v_i,v_j) در نظر بگیریم که $i\neq j$ و $v_i,v_j\in V$ باشد. از رأس (v_i,v_j) به رأس (v_l,v_p) یال جهت‌دار به وزن $c_{jp}+c_{il}$ می‌کشیم اگر و تنها اگر در گراف G ، یال جهت‌دار $v_j\vec{v}_p$ به وزن c_{jp} موجود باشد($i\neq l, j\neq p$)

ادعا: مجموع وزن کم‌وزن‌ترین مسیر بین دو رأس (e,s) و (s,e) در گراف G' ؛کمترین هزینه‌ای است که دو شخص در مجموع برای جابه‌جایی با شروط ذکر شده در مسئله در گراف G باید بپردازند.[فرض کرده‌ایم که هر دو شخص هم‌زمان باید به مقصد خود برسند تا الگوریتم پایان یابد]

در صورتی که درستی ادعا را بپذیریم،نشان می‌دهیم مسئله در حالت الف) حل می‌شود.

با توجه به اینکه گراف G' ، $V'=(V)(V-1)$ راس دارد(تعداد زوج‌های‌مرتب (v_i,v_j) که $i\neq j$ و $1\leq i,j\leq |V|$) و تعداد یال‌های‌آن به اندازه دوتایی‌های مرتب (v_i,v_j) و (v_l,v_p) است که $v_i\vec{v}_l$ و $v_j\vec{v}_p$ یال‌هایی از G باشند.
حال نابرابری زیر را داریم که در آن E' تعداد یال‌های G' است.

$$E(E-1)\geq E'$$

نابرابری فوق به این دلیل رخ می‌دهد که بعضی از انتخاب‌های مرتب دو یال، مثل حالاتی که $v_j=v_i$ است،متناظر با یالی از G' نمی‌شود ،چون رأس $(v_i,v_j=v_i)$ در گراف G' نداریم.

با توجه به اینکه وزن یال‌های G' به فرم $c_{jp}+c_{il}$ است و بنا به فرض مسئله داشته‌ایم که $c_{jp}\geq 0$ و $c_{il}\geq 0$. پس داریم که $c_{jp}+c_{il}\geq 0$.بنابر این وزن یال‌های G' نامنفی است و G' گرافی جهت‌دار با V' راس و E' یال است. حال می‌توانیم کم‌وزن ترین مسیر بین (e,s) و (s,e) (جهت دار با شروع از (e,s)) با کمک الگوریتم دایکسترا در زمان $O(V'\lg V'+E')$ بیابیم.با توجه به اینکه $V'=V(V-1)\leq V^2$ و $E'\leq E^2-E\leq E^2$ می‌توان‌گفت الگوریتم از زمان $O(V^2\lg V^2+E^2)$ است که می‌توان آن را با کمی دقت به $O(V^2\lg V+E^2)$ کاهش داد .پس از اثبات ادعا ، جواب مسئله در زمان $O(V^2\lg V+E^2)$ به دست آمده و قسمت الف) حل می‌شود. پس کافی است که ادعا را ثابت کنیم.

اثبات درستی ادعا: می‌گوییم (۱) یک کم‌وزن‌ترین مسیر از (e,s) به (s,e) در گراف G' ، متناظر با یک جواب مسئله اصلی است و (۲) اگر مسئله اصلی جواب داشته باشد،حتما مسیری از (e,s) به (s,e) و در نتیجه کم‌وزن‌ترین مسیری از (e,s) به (s,e) در گراف G' وجود دارد.

(۱): نشان‌دهیم یک کم‌وزن‌ترین مسیر از (e,s) به (s,e) متناظر با یک جواب مسئله است.

این مسیر را p بنامید و فرض کنید چنین فرمی دارد: $p : (e,s),(e_1,s_1),(e_2,s_2),\dots,(e_l,s_l)$ که در آن $(e_l,s_l) = (s,e)$. ادعا: دو مسیر p_1,p_2 که $p_1 : ee_1e_2\dots e_l$ که در آن $e = e_0$ و $e_l = s$ و $p_2 : ss_1s_2\dots s_l$ که $s = s_0$ و $s_l = e$ ،در گراف G ، دو مسیر مطلوب حرکت این دو شخص یکی از e به s و دیگری از s به e را مشخص می‌کنند. [اینجا فرض کرده‌ایم که منظورمان از مسیر p_1,p_2 در گراف G ، دنباله‌ی رئوسی است که شخص اول و دوم می‌پیمایند_بنابراین ممکن است رأسی تکراری در آن وجود داشته باشد]

اولاً لحظه ای نیست که هر دو نفر در یک رأس باشند چون فرد اول در لحظه‌ی i ام روی رأس e_i و فرد دوم در لحظه i ام روی رأس s_i است و اگر هر دو روی یک رأس باشند یعنی $s_i = e_i$ در حالیکه (e_i,s_i) رأسی از G' در مسیر P بود و می‌دانیم G' همچین رأسی ندارد. ثانیاً هر دو نفر بعد از l لحظه به مقصد خود رسیده اند و الگوریتم پایان یافته‌است.

ثالثاً (*) اگر دو مسیر p'_1,p'_2 برای این دو شخص با شروط ذکر شده در مسئله وجود داشته باشد که مجموع هزینه‌های پیمودن p'_1,p'_2 از مجموع هزینه‌های p_1,p_2 کمتر باشد، نشان دهیم مسیری کم‌وزن‌تر از p ، از (e,s) به (s,e) وجود دارد و تناقض است.پس فرض خلف باطل است و (۱) ثابت می‌شود.

پس فرض خلف را انجام داده،دو مسیر مذکور را $p'_1 : e'_0e'_1e'_{-2}\dots e'_l$ که در آن $e = e'_0$ و $e'_l = s$ و $p'_2 : s'_0s'_1s'_2\dots s'_l$ که $s = s'_0$ و $s'_l = e$ است را در نظر بگیرید(طول هر دو مسیر l' است چون زمانی که دو نفر همزمان به مقصد برسند کار تمام است.) بنا به نحوه تعریف یال در گراف G' ، $p' : (e,s),(e'_1,s'_1),(e'_2,s'_2),\dots,(e'_{l'},s'_{l'})$ یک مسیر از (e,s) به (s,e) در گراف G' است.چون اولاً برای هر $0 \le i \le l'$ ، s'_i,e'_i دو رأس متمایز در گراف G اند. بنابراین زوج مرتب (e'_i,s'_i) رأسی از G' است.ضمناً با توجه به نحوه‌ی تعریف یال در G' ، بین هر دو رأس متوالی این دنباله از رئوس ، یال جهت‌دار وجود دارد طوری که وزن این مسیر p' ،برابر با مجموع وزن یا هزینه‌ای است که دو نفر با طی کردن p'_1,p'_2 می‌پردازند.

توجه داشتیم که وزن مسیر p هم برابر با مجموع هزینه‌ای بود که دو نفر با طی کردن p_1,p_2 می‌پرداختند. پس فرض اولیه مبنی بر اینکه مجموع وزن p'_1,p'_2 از مجموع وزن p_1,p_2 کمتر باشد، با نحوه انتخاب P در تناقض است.پس همانطور که گفتیم (*) و در نتیجه (۱) ثابت شد.

مشابه با اثبات (*) می‌توان (۱) را ثابت کرد.مثلاً اگر فرض کنیم p'_1,p'_2 همانطور که در (*) تعریف شدند، دو مسیر مطلوب(جواب مسئله) در گراف G باشند،بنابراین به همان شکلی که در آنجا توضیح داده‌شد ، p' هم یک مسیر از (e,s) به (s,e) در گراف G' خواهد بود که وزن آن برابر مجموع هزینه‌های p'_1,p'_2 است(در اینجا (۲) ثابت می‌شود).اگر کوتاه‌ترین(کم‌وزن‌ترین) مسیری از (e,s) به (s,e) وجود داشته‌باشد،بنابر (۱) جوابی از مسئله خواهد بود.پس همین مسیر p' بین مسیرهای از (e,s) به (s,e) ،کم‌ترین وزن را دارد وگرنه(مشابه با آنچه در (۲) بیان شد)،می‌توان جواب بهتری بری مسئله اصلی به دست آورد.پس ادعا ثابت شد.(اگر مسئله جواب نداشته باشد، فاصله‌ی رأس (e,s) به (s,e) برابر با ∞ خواهد بود). با توجه به آنچه گفتیم قسمت الف) حل شده است.

(ب) گراف G' را با توجه به گراف اولیه(G که در مسئله داده شده) می‌سازیم.فرض کنید رئوس گراف اولیه را با $V = \{v_1,v_2,\dots,v_n\}$ نمایش دهیم.رئوس گراف G' را به شکل ۳ تایی‌های مرتب $(v_i,v_j,0),(v_i,v_i,0),(v_i,v_j,1)$ در نظر گیریم به‌طوری‌که $(1 \le i,j \le n),(i \ne j)$ باشد.

بنابراین تعداد حالات این ۳ تایی‌های مرتب برابر با $V + V(V-1) + V(V-1)$ است. اگر تعداد رئوس G' را با V' نمایش دهیم داریم که $|V'| = 2V^2 - V$ است. حال یال‌های G' را بررسی می‌کنیم.

حالت ۱: از رأس $(v_i,v_j,1)$ به $(v_l,v_p,0)$ یال جهت‌دار به وزن c_{ip} رسم می‌کنیم اگر و تنها اگر $v_j = v_l$ بوده و از v_i به

v_p در گراف G یال جهت‌دار به وزن c_{ip} داشته باشیم.

حالت ۲ : از رأس $(v_i, v_j, 0)$ به $(v_l, v_p, 1)$ یال جهت‌دار به وزن c_{ip} رسم می‌کنیم اگر و تنها اگر $v_j = v_l$ بوده و از v_i به v_p در گراف G یال جهت‌دار به وزن c_{ip} داشته باشیم.

توجه کنید در حالت ۱ می‌تواند که $v_p = v_l$ هم باشد.

توجه کنید در حالت ۱ می‌تواند که $v_i = v_j$ هم باشد.

بنابراین اگر تعداد یال‌های G' را با E' نمایش دهیم،می‌توان نوشت :

$E' \geq VE + VE$ (مثلا برای حالت ۱ $v_l = v_j$ به V حالت می‌تواند انتخاب شود و v_i, v_p که از v_i به v_p یال جهت‌دار داشته باشیم به E طریق می‌تواند انتخاب شود. اگر یال $v_i\vec{v}_p$ طوری باشد که $v_i = v_j$ باشد،چون رأس به فرم $(v_i, v_i, 1)$ در G' نداریم، یال متناظری هم در E نخواهیم داشت . پس تعدادی حالت را اضافه شمرده‌ایم. شمارش حالت ۲ نیز به طوری مشابه است با این تفاوت که $v_l = v_p$ حالات اضافی اند).

حال ادعا می‌کنیم وزن کم‌وزن‌ترین مسیر از $(e, s, 1)$ به $(s, e, 1)$ ، همان جواب مطلوب مسئله است.

نشان دهیم اگر ادعا درست باشد، قسمت ب حل می‌شود.

اگر ادعا درست باشد، با توجه به اینکه گراف G' گرافی با V' راس و E' یال جهت‌دار و دارای یال‌هایی با وزن‌های نامنفی است،به کمک الگوریتم دایکسترا در $O(V' \lg V' + E')$ می‌توانیم وزن کم‌وزن‌ترین مسیر از $(e, s, 1)$ به $(s, e, 1)$ را بیابیم.

با توجه به اینکه $V' \leq 2V^2$ و $E' \leq 2VE$ است، پس الگوریتم در زمان $O(2V^2 \lg(2V^2) + 2VE)$ که با کمی دقت همان $O(V^2 \lg(V) + VE)$ است، جواب مسئله را به ما می‌دهد.پس تنها کافی است که ادعا را ثابت کنیم.

(۱): نشان‌دهیم مسیری از $(e, s, 1)$ به $(s, e, 1)$ در گراف G' مانند P ،دو مسیر p_1, p_2 برای نفر اول و نفر دوم در گراف G به ما می‌دهد طوری‌که دو نفر در هیچ لحظه‌ای همزمان روی یک رأس نباشند و مسیر p_1 برای نفر اول، از e شروع و بعد از l مرحله به s ختم شود و مجموع هزینه نفر اول و دوم در مسیر p_1, p_2 برابر مجموع وزن یال‌های p در G' است.

(۲): نشان‌دهیم دو مسیر p_1, p_2 در G' به ترتیب برای نفر اول و دوم که شرایط مطلوب مسئله اصلی را دارن، مسیری مانند p در G' از $(e, s, 1)$ به $(s, e, 1)$ مشخص می‌کنند که کم‌ترین وزن را دارد.

(۱): مسیر p در گراف G' را در نظر بگیرید: $p : (e, s, 1), (s, e_1, 0), (e_1, s_1, 1), (s_1, e_2, 0), \ldots, (s_l, e_l, 1)$ که داریم $(e, s, 1) = (e_0, s_0, 1)$ و $(s, e, 1) = (s_l, e_l, 1)$ است(انگار نفر دوم هر بار حرکت خود را پس از نفر اول انجام می‌دهد!).

با توجه به نحوه انتخاب یال‌های G' ، مسیر به همین فرمی است که نوشته‌ایم.

ادعا: دو مسیر p_1, p_2 که $p_1 : ee_1e_2 \ldots e_l$ که در آن $e = e_0$ و $e_l = s$ و $p_2 : ss_1s_2 \ldots s_l$ که $s = s_0$ و $s_l = e$ ،در گراف G ، دو مسیر مطلوب حرکت این دو شخص یکی از e به s و دیگری از s به e را مشخص می‌کنند. [اینجا فرض کرده‌ایم که منظورمان از مسیر p_1, p_2 در گراف G ، دنباله‌ی رئوسی است که شخص اول و دوم می‌پیمایند تا زمانی که هر دو به مقصد خود برسند؛بنابراین ممکن است رأسی تکراری در آن وجود داشته باشد]

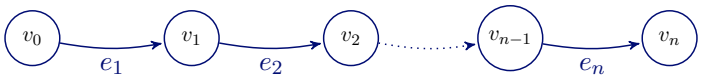
در مسیر p ، وقتی یال جهت‌داری از رأسی با مؤلفه‌ی سوم ۱ شروع و به رأس با مؤلفه‌ی سوم ۰ ختم می‌شود، یعنی نفر اول از e_i به e_{i+1} حرکت کرده(در مسیر p_1) و وقتی یال جهت‌داری با مؤلفه‌ی سوم ۰ به مؤلفه‌ی سوم ۱ می‌رود،یعنی نفر دوم از s_i به s_{i+1} حرکت کرده است.بنابراین در قدم‌های فرد در مسیر p ، نفر اول حرکت خود را انجام داده و نفر دوم مانده است.بنابراین ممکن نیست در لحظه i فرد اول در رأس e_i و فرد دوم در رأس s_i باشد که $s_i = e_i$ است.چون نفر اول زودتر به رأس e_i می‌رسد.بنابراین در قدم بعدی که نفر دوم هم به همان رأس می‌آید،می‌بایست در مسیر p به رأس $(e_i, s_i, 1)$ بخوریم($e_i = s_i$) که می‌دانیم چنین رأسی در گراف وجود ندارد.پس ممکن نیست.

ضمناً اگر دو مسیر p'_1, p'_2 برای این دو نفر با شروط ذکر شده در مسئله وجود داشته باشد که مجموع هزینه‌های پیمودن p'_1, p'_2 از مجموع هزینه‌های پیمودن p_1, p_2 کمتر باشد،نشان دهیم مسیری کم‌وزن‌تر از p از $(e, s, 1)$ به $(s, e, 1)$ وجود

دارد که تناقض است. دو مسیر را $p'_1 : e'_0e'_1e' -_2 \dots e'_l$ که در آن $e = e'_0$ و $e'_l = s$ و $s = s'_0$ که $p'_2 : s'_0s'_1s'_2 \dots s'_l$ و $s'_l = e$ در نظر بگیرید.بنا به نحوه تعریف یال در گراف G' ، $p' : (e, s, 1), (s'_0, e'_1, 0), (e'_1, s'_1, 1), \dots, (s'_{l'}, e'_{l'}, 1)$ یک مسیر از $(e, s, 0)$ به $(s, e, 0)$ در گراف G' است.چون اولاً برای هر $0 \le i \le l'$ ، s'_i, e'_{i+1} ، بنابراین زوج مرتب $(s'_i, e'_{i+1}, 1)$ رأسی از G' است. ضمناً با توجه به نحوه‌ی تعریف یال در G' ، بین هر دو رأس متوالی این دنباله از رئوس p ، یال جهت‌دار وجود دارد طوری که وزن این مسیر p' ،برابر با مجموع وزن یا هزینه‌ای است که دو نفر با طی کردن p'_1, p'_2 می‌پردازند. چون p'_1, p'_2 جوابی برای مسئله اصلی بود،هزینه $p'_1 + p'_2$ کمتر از هزینه $p_1 + p_2$ است.مسیر p' وزن کمتری از مسیر p دارد . پس ادعا ثابت می‌شود.

۳. یک روش بهینه‌سازی الگوریتم بلمن‌فورد در عمل، این است که اگر در یکی از مراحل ریلکس‌کردن مشاهده کردیم که هیچ یک از d_iها کاهش پیدا نکرد، الگوریتم را متوقف کنیم؛ می‌خواهیم بررسی کنیم که آیا این کار باعث بهبود زمان اجرای الگوریتم در بدترین حالت می‌شود یا خیر. اثبات یا رد کنید که برای گرافی وزن‌دار با V راس بدون دور منفی و با وجود یک ترتیب روی یال‌ها برای الگوریتم بلمن‌فورد، نیاز به اجرای $\Omega(V)$ بار فرایند ریلکس کردن روی کل یال‌ها را داریم.

پاسخ: فرض کنید گراف یک مسیر به فرم زیر است :



الگوریتم بلمن فورد ، یال‌های این گراف را به ترتیب عکس (یعنی ابتدا e_n ، سپس e_{n-1} و ...و در انتها e_1) را $relax$ می‌کند.در ابتدا $d(v_0) = 0$ و برای $1 \le i \le n$ قرار می‌دهیم $d(v_1) = \infty$.

بنابراین وقتی در ابتدا e_n را $relax$ می‌کنیم، $d(v_n)$ تغییری نمی‌کند . به همین صورت می‌توان گفت $d(v_{n-1})$ تا $d(v_2)$ هم تغییر نخواهد کرد اما با توجه به اینکه در k امین باری که حلقه for در الگوریتم بلمن فورد،می‌دانیم رئوسی که حداکثر فاصله‌شان(تعداد یال) از v_0 برابر k است،مقدار d آن‌ها به درستی تعیین شده.(*) بنابراین اولین دوری که حلقه اجرا می‌شود ، $d(v_1)$ برابر فاصله‌ی آن از v_0 است و $d(v_2)$ و ...و $d(v_n)$ کماکان ∞ است. با استقرا بر روی k می‌توان نشان داد در انتهای k امین اجرای حلقه for ، $d(v_1)$ و ...و $d(v_k)$ برابر فاصله‌ی هر یک از d است و $d(v_{k+1})$ و ...و $d(v_n)$ همچنان ∞ هستند.(اثبات حکم به صورت مانند چیزی که بیان کردیم، است.اگر از k به $k+1$ برویم،می‌دانیم چون در شروع حلقه $d(v_{k+1})$ =...= $d(v_{k+1})$ = ∞ است و به ترتیب $e_1, e_2, \dots, e_n$ ، $relax$ می‌شوند ، بنابراین $d(v_{k+2})$ =...= $d(v_{k+1})$ = ∞ باقی می‌ماند.چون d برای رأسی که به یک یال می‌دهد،هنگام $relax$ کردن ∞ است. اما با توجه به گزاره(*) و اینکه $d(v_k)$ در مرحله قبل به درستی تعیین شده است، در این مرحله $d(v_{k+1})$ به درستی تعیین خواهد شد).

بنابراین نتیجه می‌گیریم زمانی که d_i ها کاهش پیدا نمی‌کند که همان زمانی است هک d_i ها به درستی تعیین شده‌اند،حداقل به اندازه زمانی‌است که $d(v_n)$ به درستی تعیین شده است و $d(v_n)$ نیز در n ـ امین اجرای حلقه for بنابر استقرا به درستی مقدار دهی می‌شود . در هر حلقه‌ی for (برای k های مختلف)، n یال به ترتیبِ $e_n, e_{n-1}, \dots, e_1$ ، $relax$ می‌شوند و نشان دادیم که n بار اجرای این حلقه ضروری است. توجه کنیم که $\Omega(V)$ بار فرآیند ریلکس کردن کل یال‌ها انجام گرفته است($|V| - 1 = n$)

حال سوال می خواست برسی کنیم که روش بهینه‌سازی پیشنهاد شده‌اش باعث بهبود زمان اجرا در بدترین حالت می‌شود یا نه. ما یک مثال زدیم که به اجرای $\Omega(V)$ نیاز داشت.پس بدترین حالت بهتر از این نخواهد بود .زمان اجرای بدترین حالت با این روش بهینه‌سازی بهبود نخواهد یافت.

موفّق باشید.

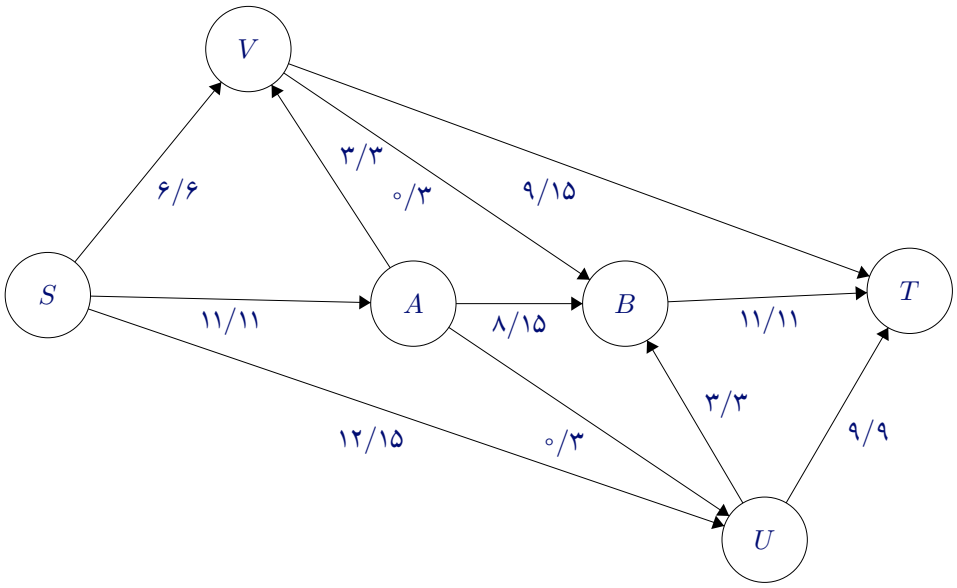
آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۷ موعد: سه‌شنبه ۲۶ فروردین ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با andishe.ghasemi.9@gmail.com مطرح کنید.

۱. در گراف زیر ظرفیت هر یال مشخص شده‌است. و S رأس منبع[1] و T رأس چاه[2] در این شبکه است.
آ) یک شار بیشینه بدست آورید و شکل آن را به همراه مقدار جریان عبوری از هر یال مشخص کنید.

راهنمایی برای پاسخ: بیشینه شار عبوری ۲۹ است.



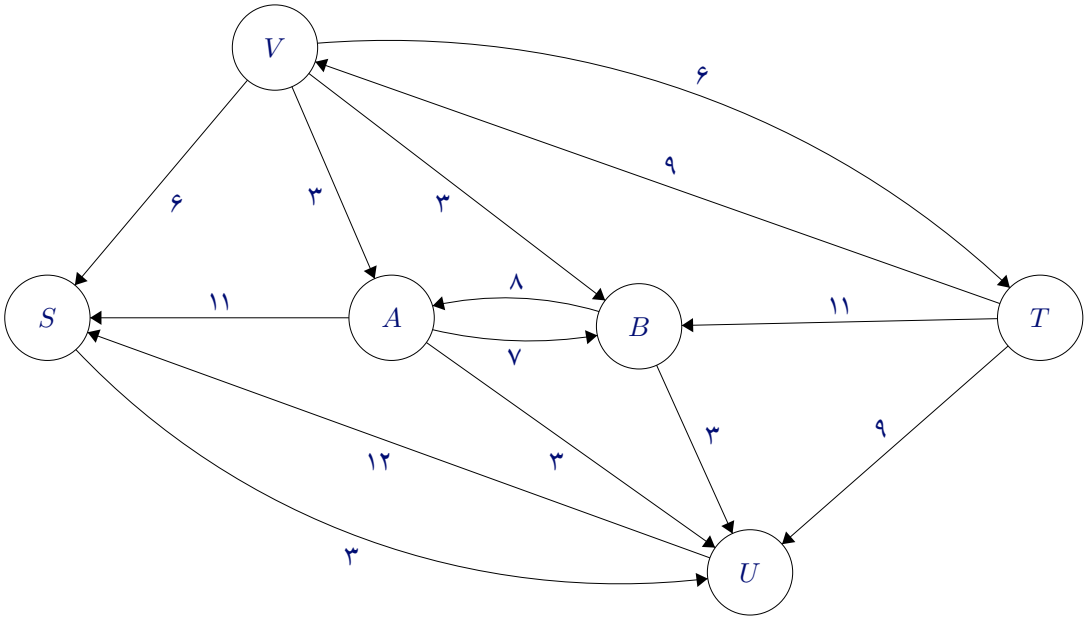
ب) یک برش کمینه بدست آورید. مقدار ظرفیت خروجی این برش و همچنین رأس‌های آن را مشخص کنید.

راهنمایی برای پاسخ: یک برش کمینه به صورت $C_1 = \{S, U\}$ و $C_2 = \{T, V, A, B\}$ است. برش دیگری به صورت $C_1 = \{S, A, B, U\}$ و $C_2 = \{T, V\}$ است.

پ) گراف باقی‌مانده نهایی را رسم کنید. کدام رأس از S قابل دسترسی است؟ کدام رأس دسترسی به T دارد؟

راهنمایی برای پاسخ: رأس S دسترسی به رأس U دارد و رأس V دسترسی به رأس T دارد.

[1] source
[2] sink



ت) به یک یال بحرانی افزایشی می‌گوییم اگر افزایش ظرفیت آن یال منجر به افزایش شار بیشینه شود. به یک یال بحرانی کاهشی می‌گوییم اگر کاهش ظرفیت آن یال منجر به کاهش شار بیشینه شود. یک یال افزایشی بحرانی و یک یال کاهشی بحرانی در این گراف پیدا کنید.(اگر وجود دارد.)

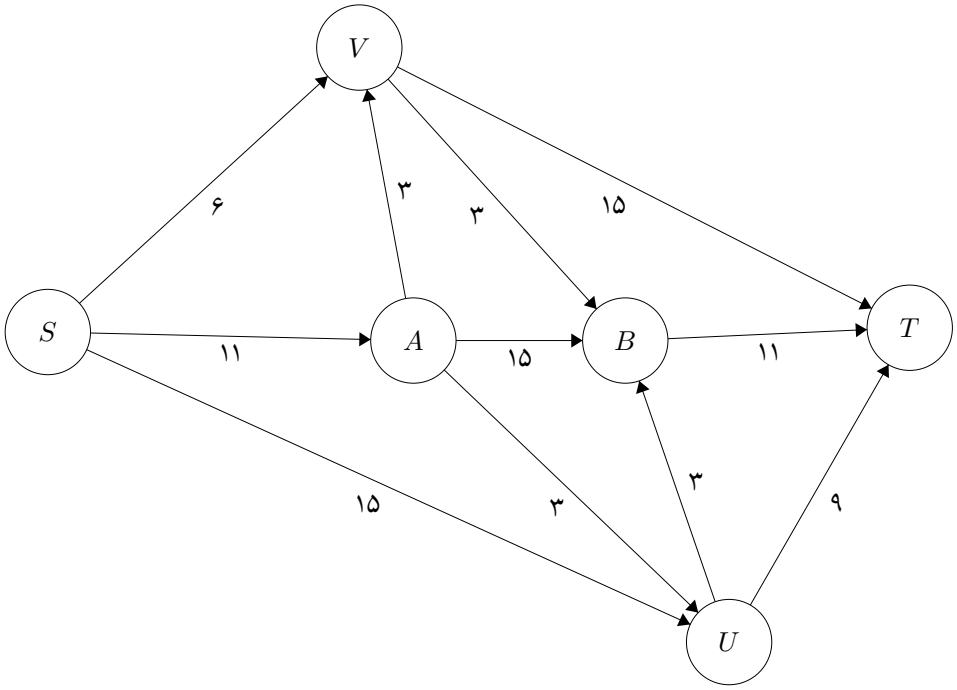
راهنمایی برای پاسخ: دو یال بحرانی افزایشی (U,T) و (S,V) وجود دارد و یال‌های بحرانی کاهشی یال‌های (S,V) و (S,A) و (U,B) و (U,T) و (B,T) و (A,V) هستند.

ث) الگوریتمی بهینه برای پیدا کردن یک یال بحرانی کاهشی طراحی کنید.

راهنمایی برای پاسخ: اثبات کردیم که شار بیشینه برابر با ظرفیت برش کمینه است. پس اگر کاهش ظرفیت یک یال منجر به کاهش ظرفیت برش کمینه شود، آنگاه کاهش ظرفیت آن یال منجر به کاهش شار بیشینه نیز می‌شود. در یک برش کمینه مثل $C_۱$ و $C_۲$ مجموع ظرفیت یال‌ها از $C_۱$ به $C_۲$ برابر با ظرفیت این برش است. پس با کم کردن ظرفیت این یال‌ها ظرفیت برش کاهش یافته و شار بیشینه کاهش می‌یابد.

ج) الگوریتمی بهینه برای پیدا کردن یک یال بحرانی افزایشی طراحی کنید.(امتیازی)

راهنمایی برای پاسخ: اگر یک یال $e=(u,v)$ بحرانی افزایشی باشد، آنگاه پس از اعمال الگوریتم شار بیشینه، این یال در گراف باقی‌مانده نهایی با حداکثر ظرفیت استفاده شده است، به عبارتی در گراف باقی‌مانده یال از v به u با ظرفیت c_e خواهیم داشت. چون اگر حداکثر ظرفیت این یال استفاده نشده باشد آنگاه در گراف باقی‌مانده نهایی یال $e'=(v,u)$ با ظرفیت $c_{e'}>۰$ خواهیم داشت. و چون طبق تعریف یال بحرانی افزایشی، افزایش ظرفیت این یال به اندازه ϵ منجر به افزایش شار بیشینه می‌شود، با افزایش این ظرفیت مسیری از s به t ایجاد می‌شود. پس در گراف باقی‌مانده، پیش از افزایش ظرفیت این یال، همچنان ظرفیت ناصفر است و مسیری از s به t شامل این یال وجود دارد و این گراف باقی‌مانده نهایی نیست. پس همه‌ی یال‌های $e=(u,v)$ که در گراف باقی‌مانده نهایی مسیری از s به u و همچنین مسیری از v به t وجود داشته و یال با ظرفیت c_e از v به u داشته باشیم، یال افزایشی بحرانی هستند. برای پیدا کردن این یال، در گراف باقی‌مانده نهایی از s یک بار BFS را اجرا می‌کنیم و یالهایی که از s قابل دسترسی هستند را پیدا می‌کنیم. بار دیگر جهت یال‌ها را برعکس کرده و از t یک بار BFS را اجرا می‌کنیم و یالهایی که به t دسترسی دارند را پیدا می‌کنیم. حال ظرفیت یال‌هایی که انتهای آن‌ها در مجموعه اول و ابتدای آن‌ها در مجموعه دوم است را با ظرفیت اولیه آن‌ها مقایسه می‌کنیم.



۲. درستی یا نادرستی موارد زیر را مشخص کنید. در صورت درست بودن اثبات مختصری ارائه کنید و در صورت نادرست بودن مثال نقض بیاورید.

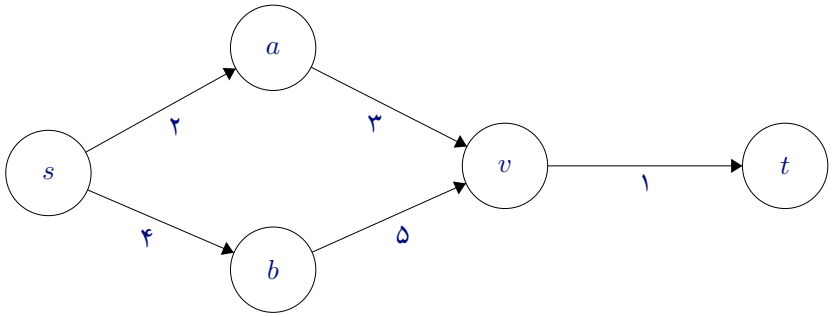
آ) اگر همه‌ی یال‌های جهت‌دار یک شبکه ظرفیت‌های متفاوتی داشته باشند ، آنگاه شار بیشینه به طریقی یکتا بدست می‌آید.

راهنمایی برای پاسخ:

نادرست. در گراف زیر همه‌ی ظرفیت‌ها متفاوت هستند، اما حداقل دو شار بیشینه $f_۱$ و $f_۲$ با ظرفیت ۱ وجود دارند.

$$f_۱(s,a) = f_۱(a,v) = f_۱(v,t) = ۱, f_۱(s,b) = f_۱(b,v) = ۰$$

$$f_۲(s,b) = f_۲(b,v) = f_۲(v,t) = ۱, f_۲(a,v) = f_۲(s,a) = ۰$$



ب) در مسئله‌ی شار بیشینه با ظرفیت رأس‌ها گراف جهت‌دار $G = (V, E)$ داده شده‌است که S رأس منبع و T رأس چاه است و ظرفیت هر رأس $v \in V$ برابر با $c_v \geq ۰$ است.(ولی ظرفیتی برای یال‌ها داده نشده‌است.) یک شار f را معتبر برای گراف G می‌گوییم اگر برای همه‌ی v ها بجز S و T ، مجموع شار ورودی به رأس v حداکثر c_v باشد. اندازه‌ی یک شار معتبر f ، مجموع شار خروجی از S است. با داشتن گراف ورودی ، مسئله‌ی شار بیشینه با ظرفیت رأس‌ها ، محاسبه کردن یک شار معتبر با سایز بیشینه است.

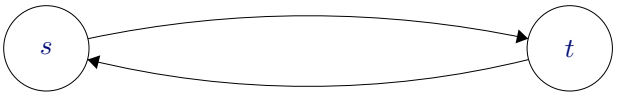
محاسبه‌ی یک شار بیشینه با ظرفیت رأس‌ها را می‌توان به مسئله‌ی شار بیشینه معمولی(با ظرفیت یال‌ها) کاهش داد.

راهنمایی برای پاسخ: درست. به این صورت کاهش می‌دهیم که گراف G' را می‌سازیم که در آن به ازای هر رأس v در G با ظرفیت c_v دو رأس v_1 و v_2 را قرار می‌دهیم. همه‌ی یال‌های ورودی به v ، در گراف G' به رأس v_1 وارد می‌شوند. و همه‌ی یال‌های خروجی از v ، از v_2 خارج می‌شوند. و در نهایت یال (v_1, v_2) با ظرفیت c_v قرار داده می‌شود. حالا هر شار f در G که مطابق قوانین ظرفیت رأس‌ها است، می‌تواند تبدیل به یک شار f' در G' شود که آن هم مطابق قوانین ظرفیت رأس‌ها باشد. برای این تبدیل به ازای هر یال $(u, v) \in G$ قرار می‌دهیم $f'(u_2, v_1) = f(u, v)$ و برای (v_1, v_2) مقدار آن را برابر با مجموع کل شار ورودی به رأس v قرار می‌دهیم. چون f مطابق قوانین ظرفیت رأس‌ها در G است، داریم $f'(v_1, v_2) \leq c_v$ پس f' نیز مطابق قوانین ظرفیت یال‌ها است.

همچنین هر شار f' در G' می‌تواند به یک شار f در G تبدیل شود که مطابق با قوانین رأس‌ها باشد. برای این تبدیل به ازای هر یال (u, v) در G قرار می‌دهیم $f(u, v) = f'(u_2, v_1)$.پس شار ورودی به هر رأس v حداکثر c_v می‌شود. پس در نهایت شار بیشینه در G' برابر با شاری در G با بیشترین مقدار و مطابق قوانین ظرفیت رأس‌ها است.

پ) اگر هر یال جهت‌دار با ظرفیت c و بین دو رأس u و v در یک شبکه را با دو یال جهت‌دار با جهت‌های مخالف و ظرفیت c بین دو رأس u و v جایگزین کنیم ، آنگاه مقدار شار بیشینه ثابت می‌ماند.

راهنمایی برای پاسخ: نادرست. فرض کنید ظرفیت یال‌ها ۱ باشد، در گراف اول اندازه شار بیشینه صفر است ولی در گراف دوم این مقدار برابر با یک است.



۳. در یک ساختمان عمومی مثل یک سینما ، داشتن یک نقشه‌ی خروج برای موارد اضطراری نظیر آتش‌سوزی مهم است. در این سوال می‌خواهیم با استفاده از شار بیشینه یک نقشه خروج اضطراری طراحی کنیم. فرض کنید که نقشه سینما یک گراف $G = (V, E)$ است که در آن هر اتاق یا طبقه با یک رأس و هر راهرو یا پله با یک یال مشخص شده‌است. هر راهرو یا پله دارای ظرفیتی c است که نشان می‌دهد حداکثر c نفر همزمان می‌توانند از این راهرو استفاده کنند. پیمایش یک راهرو از یک سر تا سر دیگر یک واحد زمانی طول می‌کشد.(پیمایش یک اتاق صفر واحد زمانی طول می‌کشد.) فرض کنید در ابتدا همه مردم در اتاق S هستند و تنها یک خروجی T به خیابان وجود دارد. نشان دهید که چطور با استفاده از مسئله‌ی شار بیشینه ، سریع‌ترین راه برای خارج کردن همه افراد از ساختمان را پیدا کنیم. (راهنمایی : گراف G' را طراحی کنید که در آن هر رأس نشان‌دهنده‌ی یک اتاق در هر واحد زمانی باشد.)

راهنمایی برای پاسخ: فرض کنید M نفر وجود دارند که باید آن‌ها را خارج کنیم. اول الگوریتمی طراحی می‌کنیم که مشخص کند آیا همه افراد در T واحد زمانی خارج می‌شوند یا خیر. با داشتن این الگوریتم می‌توانیم یک باینری سرچ روی T از ۱ تا $|V|M/c$ انجام می‌دهیم تا کمترین زمان ممکن برای خروج کل افراد را پیدا کنیم. الگوریتمی که استفاده می‌کنیم، با داشتن گراف G، گراف G' را به صورت زیر می‌سازد:

به ازای هر رأس v در V ، T کپی از v به صورت $v_1, v_2, ..., v_T$ می‌سازیم به طوری که v_i مربوز به مرحله زمانی i ام است. به ازای هر i یالی از v_i به v_{i+1} با ظرفیت بی‌نهایت می‌کشیم.(افراد تنها می‌توانند یک واحد زمانی در اتاق‌ها بمانند.) سپس یالی از v_i به w_{i+1} با ظرفیت c می‌کشیم اگر که یالی از v به w با ظرفیت c در گراف اصلی وجود داشته باشد. فرض کنید همه‌ی افراد در ابتدا در اتاق a هستند و راه خروج اتاق b است. پس قرار می‌دهیم $s = a_1$ و $t = b_T$. برای اینکه بفهمیم همه افراد در T واحد زمانی

از s به t می‌رسند یا خیر، بررسی می‌کنیم که آیا شار بیشینه در G' بزرگتر مساوی تعداد افرادی که در ابتدا در s بودند است یا خیر. اگر برابر بود ما می‌توانیم افراد را در T واحد زمانی خارج کنیم.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۸

---

۱. گراف وزن‌دار با وزن‌های طبیعی و بدون جهت G و دو راس s, t از رئوس G داده شده است.

می‌خواهیم در این گراف دو مسیر متمایز از s به t بیابیم، طوری‌که این دو مسیر در هیچ یالی مشترک نباشند و همچنین وزن هر کدام برابر با وزن کوتاه‌ترین مسیر از s به t باشد. دقت کنید که این دو مسیر می‌توانند در یک یا چند راس مشترک باشند.
با کمک مسئله‌ی شار بیشینه، الگوریتمی برای پیدا کردن این مسیر ارائه دهید.

پاسخ: ابتدا با یک بار اجرای الگوریتم بلمن‌ـفورد برای هر رأس v کمینهٔ فاصلهٔ آن از s را می‌یابیم. (از آن‌جا که وزن همهٔ یال‌ها مثبت است می‌توان گراف G را تبدیل به یک گراف جهت‌دار بدون دور منفی کرد پس اجرای الگوریتم بلمن‌ـفورد امکان‌پذیر است.) فرض کنید برای هر رأس v، $d(v)$ نشان‌دهندهٔ فاصلهٔ s از v و برای هر یال e، $w(e)$ نشان‌دهندهٔ وزن آن یال باشد. گراف G' را با همان رئوس G به این شکل می‌سازیم: برای هر یال $e = uv \in E(G)$، اگر $d(v) = d(u) + w(e)$، در G' یالی جهت‌دار از u به v با ظرفیت ۱ می‌کشیم و اگر $d(u) = d(v) + w(e)$، در G' یالی جهت‌دار از v به u با ظرفیت ۱ می‌کشیم. در واقع تنها یال‌هایی را رسم می‌کنیم که ممکن است در مسیرِ با وزن کمینه از s به t در G ظاهر شوند.

حال فرض کنید $sv_1v_2 \dots v_kt$ مسیری با وزن کمینه در G باشد. در نتیجه

$$\begin{cases} d(v_1) = d(s) + w(sv_1) \\ d(v_2) = d(v_1) + w(v_1v_2) \\ \vdots \\ d(t) = d(v_k) + w(v_kt) \end{cases}$$

پس همهٔ یال‌های $s \to v_1, v_1 \to v_2, \dots, v_k \to t$ در G' ظاهر شده‌اند و مسیری از s به t در G' معادل با این مسیر وجود دارد.
این بار فرض کنید $sv_1v_2 \dots v_kt$ مسیری در G' باشد. چون یال‌های $s \to v_1, v_1 \to v_2, \dots, v_k \to t$ در G' آمده‌اند، نتیجه می‌شود

$$\begin{cases} d(v_1) = d(s) + w(sv_1) \\ d(v_2) = d(v_1) + w(v_1v_2) \\ \vdots \\ d(t) = d(v_k) + w(v_kt) \end{cases}$$

بنابراین $s \to v_1, v_1 \to v_2, \dots, v_k \to t$ یک مسیر با وزن کمینه از s به t در G است. از مطالب گفته‌شده نتیجه می‌شود که هر مسیر از s به t در G' معادل با یک مسیر با وزن کمینه از s به t، با همان رئوس، در G است. پس مسیرهای یال مجزا از s به t در G' معادل با مسیرهای با وزن کمینه و یال مجزا از s به t در G هستند. در نتیجه کافی است شار بیشینه در G' را بیابیم و اگر مقدار شار بیش‌تر از ۱ بود، در G' حداقل ۲ مسیر یال مجزا از s به t وجود دارد که در الگوریتم پیدا کردن شار بیشینه ظاهر شده‌اند.

۲. گراف وزن‌دار و بدون جهت G را در نظر بگیرید.
در این گراف به هر رأس و هر یال وزنی طبیعی و کمتر از C نسبت داده شده‌است.

برای هر زیرگراف مثل H در G، وزن H را مجموع وزن یال‌های H منهای مجموع وزن راس‌های H تعریف می‌کنیم. (بدیهی است که اگر یال e در زیرگراف H باشد، آنگاه حتما راس‌های ابتدا و انتهای e نیز در زیرگراف هستند.)

با کمک مسئله‌ی شار بیشینه، الگوریتمی ارائه دهید که زیرگرافی با وزن بیشینه در G را بیابد.

پاسخ: گراف G' را به این شکل می‌سازیم: برای هر یال از G مانند e، راسی متناظر با این یال به G' و به‌ازای هر راس از G مانند v، راسی متناظر با v به G' اضافه می‌کنیم. حال مجموعهٔ روابط پیش‌نیازی زیر را در نظر بگیرید:
اگر $e = uv \in E(G)$ آنگاه u, v پیش‌نیازهای e هستند یعنی اگر e در زیرگراف مورد نظر ظاهر شود، u, v هم باید در این زیرگراف باشند همچنین اگر یال e را انتخاب کنیم جایزهٔ $+w(e)$ (وزن یال e) و اگر راس v را انتخاب کنیم جایزهٔ $-w(v)$ (وزن راس v) را دریافت می‌کنیم.
تعدادی رابطهٔ پیش‌نیازی داریم، به همراه یک تابع وزن روی اشیاء و می‌خواهیم جایزهٔ ما بیشینه شود پس تنها کافی است مسئلهٔ انتخاب پروژه[1] را با این ورودی‌ها حل کنیم.

[1] Project selection

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۹ موعد: چهارشنبه ۱۰ اردیبهشت ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با Ghazalkhn99@gmail.com, Mavali1999@gmail.com مطرح کنید.

تمرین‌های تحویلی

۱. برای مسائل زیر کوچک‌ترین کلاس پیچیدگی از بین P، NP و NP-Complete که می‌توانید ثابت کنید مسئله در آن است کدام است؟ دلیل مختصری بیاورید.

(آ) یک گراف $G = (V, E)$ داده شده است، آیا $V' \subseteq V$ موجود است که $|V'| = k$ و بین هر دو راس در V' یالی در G باشد؟ (k یک عدد ثابت است و جزء ورودی‌های مسئله نیست.)

(ب) یک گراف $G = (V, E)$ و عدد k داده شده است، آیا $V' \subseteq V$ موجود است که $|V'| = k$ و بین هر دو راس در V' یالی در G باشد؟

(ج) یک گراف $G = (V, E)$ و عدد k داده شده است، آیا $E' \subseteq E$ وجود دارد که $|E'| < k$ و برای هر $v \in V$، یالی مانند e در E' باشد که v یکی از دو سر e باشد؟

(د) یک گراف $G = (V, E)$ و عدد k داده شده است، آیا $V' \subseteq V$ وجود دارد که $|V'| < k$ و برای هر $e \in E$، راسی مانند v در V' باشد که v یکی از دو سر e باشد؟

(ه) مسئله‌ی 3-SAT با این فرض اضافه برای ورودی که در ورودی هر متغیر یا نقیض آن حداکثر ۲ بار در ورودی ظاهر شده است.

(و) مسئله‌ی 3-SAT با این فرض اضافه برای ورودی که در ورودی هر متغیر یا نقیض آن حداکثر ۴ بار در ورودی ظاهر شده است.

راهنمایی. مسئله‌ی پیداکردن بزرگترین تطابق در گراف در زمان چندجمله‌ای بر حسب اندازه‌ی گراف ورودی قابل حل است.

پاسخ:

(آ) (دلایل به صورت مختصر آورده شده است):

در کل $\binom{n}{k} = O(n^k)$ دسته k تایی از رئوس وجود دارد.هر یک را در $O(k^2)$ می‌توان چک کرد که آیا این مجموعه k تایی ،خوشه است یا نه.پس در کل زمان اجرا $O(k^2 n^k)$ است که چون $k = O(1)$ است، زمان چند جمله‌ای است. پس این مسئله در کلاس **P** قرار دارد.

(ب) این مسئله در کلاس $\mathbf{NPC}(NP - complete)$ است. به وضوح در صورت داشتن جواب می‌توان آن را در زمان چند جمله‌ای چک کرد. پس **NP** است.ضمناً مسئله پیدا کردن خوشه k رأسی در گراف G دقیقاً معادل با پیدا کردن مجموعه مستقل k رأسی در $\bar{G}$ است و چون این مسئله اخیر، NPC است. پس پیدا کردن خوشه k رأسی هم **NPC** است.

(ج) به راحتی می‌توان اثبات کرد که حداقل تعداد یال‌هایی که باید انتخاب کنیم تا کل رئوس G پوشیده شوند برابر $n - |M| = n - 2|M| + |M|$ است که $|M|$ تعداد یال های یک تطابق ماکزیمم در G است.(اگر G رأس ایزوله داشته باشد هیچ پوشش کاملی برای رئوس وجود ندارد که این در زمان $O(V)$ قابل بررسی است)این حداقل هم در حالتی رخ می‌دهد که همه یال های یک تطابق ماکزیمم را انتخاب کنیم و از رئوس خارج از تطابق هم یک یال انتخاب کنیم که این یال لزوماً یک برش یکی از رئوسی است که توسط تطابق ماکزیمم پوشیده شده‌است. پیدا کردن تطابق بیشینه و بررسی وجود رأس برشی در زمان چند جمله‌ای ممکن است.پس در کل مسئله‌ی یافتن حداقل تعداد یال‌های لازم برای پوشش همه رئوس در زمان چند جمله‌ای قابل حل است.درنتیجه این مسئله هم در کلاس **P** قرار دارد.

(د) این مسئله در واقع پیدا کردن پوشش راسی با اندازه حداکثر $k - 1$ در گراف G است.طبق قضیه‌ای در گراف ها

$$\text{اندازه هر پوشش رأسی} = n - \text{اندازه یک مجموعه‌ی مستقل}$$

(کافیست رئوسی که در پوشش راسی نیستند را در نظر بگیرید.این رئوس به یکدیگر هیچ یالی ندارند)بنابراین پیدا کردن پوشش راسی با اندازه حداکثر $k - 1$ در G معادل با پیدا کردن یک مجموعه مستقل به اندازه حداقل $n - k + 1$ در است.
چون این مسئله از نوع مسئله **NPC** است،پس مسئله پیدا کردن پوشش راسی با حداکثر $k - 1$ عضو هم **NPC** است.

(ه) این مسئله در کلاس **P** قرار دارد. فرض کنید تعداد عبارت ها و n تعدا لیترال ها باشد. می توانیم در زمان چند جمله ای همه متغیر هایی که دقیقا یکبار ظاهر شده اند یا متغیر هایی که دو بار ظاهر شده اند اما در هر دو بار به یک شکل (x_i یا $\bar{x_i}$ بوده اند را $true$ کنیم و عبارت های شامل آن هارا هم(که اکنون حتما $true$ هستند)حذف کنیم . (چون با هر بار پیدا کردن چنین متغیری حداقل یکی از عبارات حذف می شود)

بنابراین به حالتی می‌رسیم که هر متغیر x_i دقیقاً یکبار به شکل x_i و یکبار به شکل $\bar{x_i}$ در عبارات ظاهر می‌شود.

اگر گراف دو بخشی G را تعریف کنیم که رئوس یک بخش آن متناظر با x_i ها و رئوس بخش دیگر آن متناظر با عبارت‌ها (C_i ها) باشد و $x_iC_j \in E$ اگر در عبارت C_j ، لیترال x_i یا $\bar{x_i}$ آمده باشد،آنگاه به وضوح هر تطابقی که همه رئوس C_i هارا بپوشاند متناظر با یک شیوه مقدار دهی x_j ها است که در آن همه C_i ها $true$ شوند و نیز هر شیوه مقدار دهی x_i ها که همه C_j هارا $true$ کند متناظر با یک تطابق است که همه C_j هارا می پوشاند. پس بعد از صرف زمانی چند جمله ای،این مسئله هم به مسئله تطابق بیشینه در گراف در بخشی تبدیل می شود.بنابراین در کلاس **P** قرار دارد.

(و) این مسئله هم در کلاس **NPC** است.

می توان مسئله $3SAT$ را به این مسئله (آن را $3SAT*$ می نامیم)تحویل کرد. به این صورت عمل می‌کنیم که به ازای هر ورودی برای مسئله $3SAT$ اگر از یک متغیر مثل x ، $k > 4$ بار استفاده شده بود، k متغیر جدید $x_1, x_2, \ldots, x_k$ را تعریف می کنیم ،به جای امین جایی که x آمده بود، x_i را به کار می بریم.ضمنا عبارت های $(x_1 \vee \bar{x_2}), (x_2 \vee \bar{x_3}), \ldots, (x_{k-1} \vee \bar{x_k})$ را هم به مجموعه عبارات اضافه می کنیم.این عبارات تضمین می کند که x_i ها یا با همدیگر $true$ هستند و یا با همدیگر $false$.

بنابراین اگر این عبارت جدید صدق پذیر باشد، عبارت اولیه هم صدق پذیر است و برعکس.

پس در زمان چند جمله ای ، $3SAT$ به $3SAT*$ قابل تحویل است و در نتیجه NP_Hard ، $3SAT$ است.به سادگی می توان دید که این مسئله **NP** هم هست. درنتیجه **NPC** است.

۲. در گراف $G = (V, E)$ یک زیرمجموعه از راس‌ها مثل U را خیلی‌مستقل گوییم هرگاه هیچ مسیری با طول حداکثر ۲ در G بین دو راس از U موجود نباشد. ثابت کنید مسئله‌ی زیر NP-Complete است:
یک گراف $G = (V, E)$ و عدد k داده شده است، آیا G زیرمجموعه‌ی خیلی‌مستقل به اندازه‌ی حداکثر k دارد؟

پاسخ: به راحتی می توان دید که این مسئله **NP** است زیرا باداشتن جواب صحت آن را در زمان $O(E)$ می‌توان بررسی کرد. حال مسئله زیر مجموعه ی مستقل به اندازه ی حداقل k (IS) را به مسئله زیر مجموعه خیلی مستقل به اندازه حداقل k (ID*) تحویل می کنیم.

فرض کنید گراف G به عنوان ورودی داده شده و می‌خواهیم مسئله (IS) را برای آن بررسی کنیم. روی هر یال از G یک راس جدید قرار دهید.این رئوس جدید که $|E|$ هستند را **رئوس میانی** می نامیم و این کار در زمان $O(|E|)$ قابل انجام است. رئوس ایزوله در هر مجموعه مستقلی می توانند بیایند؛ پس بدون از دست دادن کلیت فرض کنید G رأس ایزوله‌ای ندارد.حال اگر همه رئوس میانی را به یکدیگر وصل کنیم،گراف G^* به دست می‌آید.ثابت می‌کنیم وجود زیر مجموعه مستقل k راسی در G $\iff$ وجود زیر مجموعه خیلی مستقل k رأسی در G^*

۱ـ ابتدا دقت کنید که اگر دو راس در G مجاور باشند .فاصله این دو راس در G^* دقیقاً ۲ است. بنابراین اگر دو رأس u, v در هیچ زیر مجموعه مستقلی از G باهم نیایند (یعنی $uv \in E(G)$) آنگاه در هیچ زیر مجموعه‌ی خیلی مستقلی از G^* هم u, v با هم وحود نخواهند داشت.

۲ـ اگر دو راس u, v می‌توانستند در یک مجموعه مستقل از G باهم حضور داشته باشند،یعنی $uv \notin E(G)$ آنگاه این دو رأس در یک مجموعه خیلی مستقل از G^* هم می‌توانند با هم حضور داشته باشند زیرا فاصله u, v در G^* حداقل ۳ است.

۳ـ اگر مجموعه خیلی مستقل در G^* با k رأس وجود داشته باشد ،آنگاه G^* شامل مجموعه خیلی مستقل k رأسی است که هیچ یک از رئوس میانی درآن نیامده است. چون همه رئوس میانی در G^* به همدیگر یال دارند،به وضوح حداکثر ۱ رأس میانی می‌تواند در یک مجموعه خیلی مستقل از G^* حضور داشته باشد .ضمناً اگر در یک مجموعه مستقل یکی از رئوس میانی حضور داشته باشد، هیچ رأسی از گراف اصلی (یعنی رئوس غیر میانی) در این مجموعه خیلی مستقل حضور ندارد.زیرا با صرف نظر کردن از رئوس ایزوله، فاصله هر رأس میانی تا هر رأس اصلی حداکثر ۲ است.بنابراین تنها در صورتی یک مجموعه خیلی مستقل k عضوی در G^* شامل رأسی از رئوس میانی است که $k = 1$ و بدیهتاً با انتخاب یک رأس دلخواه اصلی از G^* می‌توانیم مجموعه خیلی مستقل $k = 1$ عضوی از رئوس غیر میانی در G^* پیدا کنیم.

با توجه به سه لم گفته شده،هر مجموعه مستقل در G متناظر با یک مجموعه‌خیلی مستقل،به همان اندازه ،در G^* است .پس در زمان چند جمله ای هر پرسش به صورت: "آیا G زیر مجموعه ی مستقل به اندازه حداقل k دارد؟" را می‌توان به پرسشی به صورت "آیا G^* زیر مجموعه خیلی مستقلی به اندازه حداقل k دارد ؟" تبدیل کرد.چون پرسش اول **NPC** است،پس پرسش دوم هم $NP-Hard$ است و با توجه به آنچه گفته شد ،مسئله وجود زیر مجموعه خیلی مستقل به اندازه حداقل k هم **NPC** است.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۱۰ موعد: سه‌شنبه ۱۵ اردیبهشت ساعت ۱۲

ـ سؤالات خود پیرامون تمرین را با alirtofighim@gmail.com مطرح کنید.

۱. با فرض $\mathbf{P} = \mathbf{NP}$ الگوریتمی چند جمله‌ای ارائه دهید که یک 3SAT به عنوان ورودی گرفته و در صورتی که ورودی ارضاپذیر باشد، مقداردهی متغیرها که 3SAT را ارضا کند خروجی دهد.

راهنمایی برای پاسخ: فرض کنید الگوریتمی با زمان اجرای $f(n)$ داریم که یک 3SAT با اندازه‌ی n گرفته و در زمان $f(n)$ می‌گوید که ارضاپذیر است یا خیر. همچنین چون مسئله‌ی 3SAT در کلاس $\mathbf{NP}$ است، فرض $\mathbf{P} = \mathbf{NP}$ نتیجه می‌دهد که الگوریتمی با زمان چندجمله‌ای نیز داریم و می‌توانیم فرض کنیم $f(n)$ نسبت به n چند جمله‌ای است.
حال یک 3SAT دلخواه به اندازه‌ی n شامل متغیرهای $x_۱, ..., x_k$ در نظربگیرید، الگوریتمی برای مقداردهی $x_۱, ..., x_k$ ارائه می‌دهیم.
$S_۰$ را برابر با ورودی قرار دهید، $S_۰$ را به الگوریتم بررسی ارضاپذیری 3SAT می‌دهیم، اگر جواب خیر داد چاپ می‌کنیم که 3SAT ارضا پذیر نیست و الگوریتم را تمام می‌کنیم، در غیر این‌صورت یک مقداردهی وجود دارد که $S_۰$ ارضاپذیر است، حال k بار عملیات زیر را انجام می‌دهیم:
در مرحله‌ی iام، $S'_i = S_{i-۱} \wedge (x_i \vee x_i \vee x_i)$ و $S''_i = S_{i-۱} \wedge (\neg x_i \vee \neg x_i \vee \neg x_i)$ را می‌سازیم و هر کدام را به مسئله‌ی بررسی ارضاپذیری 3SAT می‌دهیم، اگربا ورودی S'_i بله داد، مقدار $x_i = \text{true}$ چاپ می‌کنیم و $S_i = S'_i$ قرار می‌دهیم و در غیراینصورت اگر با ورودی S''_i بله داد مقدار $x_i = \text{false}$ چاپ کرده و $S_i = S''_i$ چاپ می‌کنیم. همچنین اگر فرض کنیم $S_{i-۱}$ ارضاپذیراست حتما یکی از این دو حالت رخ می‌دهد که با استقرا می‌توان این را نشان داد.
بعد از پایان k مرحله به S_k می‌رسیم که ارضاپذیر است و همچنین مقداردهی کل متغیرها را چاپ کردیم، برای هر متغیر یکی از دو عبارت $(x_i \vee x_i \vee x_i)$ یا $(\neg x_i \vee \neg x_i \vee \neg x_i)$ در S_k موجود است که به صورت یکتا مقداردهی ارضاپذیر را مشخص می‌کند.
در این الگوریتم $۲k + ۱$ بار الگوریتم قبلی را اجرا کردیم که طول ورودی آن حداکثر $n + ۳k$ بوده پس زمان اجرای آن $(۲k + ۱) * f(۴n)$
$f(n + ۳k) \leq (۲n + ۱) * f(۴n)$ است که با فرض چندجمله‌ای بودن f چند جمله‌ای است و مسئله حل شد.

۲. مسئله‌ی کوله‌پشتی چندگانه به این صورت است که m کوله‌پشتی داریم که ظرفیت کوله‌پشتی iام برابر با c_i است. همچنین n الماس داریم که الماس iام ارزش v_i و وزن w_i دارد. می‌توانیم تعدادی از الماس‌ها را برداشته و هر کدام را داخل یکی از کوله‌پشتی‌ها قرار دهیم، اما مجموع وزن الماس‌های داخل یک کوله‌پشتی نباید از ظرفیت آن کوله‌پشتی بیشتر شود.
ورودی ظرفیت کوله‌پشتی‌ها، مشخصات الماس‌ها و عدد k است و می‌خواهیم ببینیم آیا می‌توانیم حداقل با مجموع ارزش k الماس داخل کوله‌پشتی‌ها جا دهیم.

(آ) برای حالت $m = ۱$ (یک کوله‌پشتی) ثابت کنید مسئله‌ی کوله‌پشتی انپی‌ـتمام است.

راهنمایی برای پاسخ: اولا باید نشان دهیم که این مسئله NP است که به سادگی ممکن است، چراکه اگر لیست شماره الماس‌هایی که باید آن‌ها را در کوله‌پشتی قرار دهیم داشته‌باشیم، به سادگی می‌توان بررسی کرد که آیا وزن الماس‌ها از وزن کوله پشتی کمترمساوی و جمع ارزششان از k بیشتر مساوی است یا خیر.

حال مسئله‌ی $Subset - Sum$ را به این مسئله کاهش چندجمله‌ای می‌دهیم و چون $Subset - Sum$ انپی‌ـ تمام است پس مسئله‌ی کوله‌پشتی نیز انپی تمام است.

ورودی مسئله‌ی $Subset - Sum$ یک آرایه از n عدد است و عدد t است و پرسش این است که آیا زیرآرایه‌ای با مجموع t دارد یا خیر.

کافی است با داشتن ورودی مسئله‌ی $Subset - Sum$، n الماس درنظر بگیریم که وزن و ارزش الماس iام برابر با عدد iام آرایه باشد و ظرفیت کوله و عدد k را برابر با t قرار دهیم.

حال کافی‌است نشان دهیم یک ورودی برای مسئله‌ی $Subset - Sum$ مقدار بله دارد اگر و تنها اگر تبدیل‌شده‌ی این ورودی به مسئله‌ی کوله‌پشتی پاسخ بله داشته باشد. که برای هر دو کافی است شماره اندیس اعضای آرایه که برای زیرآرایه انتخاب شده را متناظر با شماره اندیس الماس‌های انتخاب شده در نظر بگیریم.

(ب) برای $m = ۱$ (یک کوله‌پشتی) و با فرض $\mathbf{P} \neq \mathbf{NP}$ ثابت کنید مسئله‌ی کوله‌پشتی قویاً انپی‌ـ تمام نیست.

راهنمایی برای پاسخ: ابتدا ثابت می‌کنیم این مسئله الگوریتم چندجمله‌ای دارد، سپس با برهان خلف نشان می‌دهیم که انپی‌ـ تمام نیست. الگوریتم چندجمله‌ای این پرسش در مبحث برنامه‌نویسی پویا در کلاس مطرح شده‌است پس می‌دانیم که الگوریتم چندجمله‌ای دارد. همچنین اگر انپی‌ـ تمام باشد همه‌ی الگوریتم‌های انپی به آن کاهش می‌یابند و چون این مسئله راه حل چند جمله‌ای دارد آن‌ها نیز راه حل چندجمله‌ای خواهند داشت و در نتیجه $\mathbf{NP} \subseteq \mathbf{P}$ ، اما هر مسئله در کلاس $\mathbf{P}$ خود $\mathbf{NP}$ نیز است و در واقع $\mathbf{P} \subseteq \mathbf{NP}$ و در نتیجه $\mathbf{P} = \mathbf{NP}$ که این با فرض مسئله در تناقض است.

(ج) ثابت کنید مسئله‌ی کوله‌پشتی چندگانه قویاً انپی‌ـ تمام است.

راهنمایی برای پاسخ: اولا می‌دانیم مسئله‌ی کوله‌پشتی چندگانه انپی تمام است، کافی است ثابت کنیم مسئله‌ی کوله‌پشتی چندگانه با ورودی یکانی انپی تمام است. برای اینکار کافی است یکی از الگوریتم‌هایی که می‌دانیم انپی‌ـ تمام است را به آن کاهش چندجمله‌ای دهیم، مسئله‌ی $3 - Partition$ با ورودی یکانی را در نظر می‌گیریم. ورودی مسئله‌ی $3 - Partition$، یک آرایه از اعداد مثل $a_۱, ..., a_{۳n}$ است که $a_i \in (\frac{\sum_{i=۱}^{۳n} a_i}{۴n}, \frac{\sum_{i=۱}^{۳n} a_i}{۲n})$ و می‌خواهیم ببینیم آیا این آرایه را می‌توانیم به n دسته‌ی ۳ عضوی افراز کنیم که مجموع اعداد هر دسته باهم برابر شود یا خیر،

برای اینکار کافی‌است n کوله‌پشتی با ظرفیت $\frac{\sum_{i=۱}^{۳n} a_i}{n}$ در نظر گرفته و به‌ازای عدد iام، الماسی با وزن a_i و ارزش a_i قرار دهیم و مقدار $k = \sum_{i=۱}^{۳n} a_i$ قرار دهیم. ساختن این ورودی به صورت یکانی چندجمله‌ای است چراکه ورودی مسئله‌ی $3 - Partition$ نیز یکانی در نظر گرفتیم. حال کافی است نشان دهید یک ورودی پاسخ بله برای مسئله‌ی $3 - Partition$ می‌گیرد اگر و تنها اگر تبدیل آن به ورودی مسئله‌ی کوله‌پشتی چندگانه پاسخ بله در مسئله‌ی کوله‌پشتی چندگانه دریافت کند.

موفّق باشید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

[بهار ۹۹]

تمرین سری ۱۲

---

۱. چندوجهی

$$P = \{[x_1, x_2]^T \mid x_1 + x_2 \geq 1,\ -x_1 + x_2 \leq 2,\ x_1 - 2x_2 \leq 4,\ x_1 \geq 0\},$$

را در نظر بگیرید.

(آ) چندوجهی P را در یک دستگاه مختصات رسم کنید.

(ب) نقاط رأسی P را بیابید.

(ج) جهت‌های دور شونده راسی چندوجهی P را بیابید.

(د) با استناد به قضیه نمایش، نقاط $[1/2, 3/2]^T$، $[1, 1]^T$ و $[3, 2]^T$ را به صورت ترکیب نقاط رأسی و جهت‌های دور شونده رأسی بنویسید.

(ه) با استفاده از قضیه نمایش و نمایش نقاط شدنی مساله با استفاده از نقاط رأسی و جهت‌های دورشونده رأسی، مجموعه جواب بهینه مساله برنامه‌ریزی خطی

$$\min\ c^T x \quad \text{s.t.} \quad x \in P, \tag{۱}$$

که در آن $c^T = [1, 1]$ را در صورت وجود بیابید.

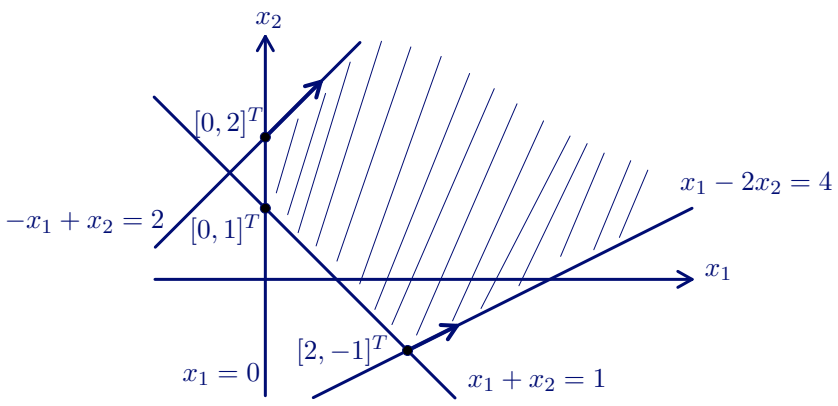
(و) مشابه قسمت قبل، نشان دهید مساله برنامه‌ریزی خطی (۱) که در آن $c^T = [-2, -1]$ جواب بهینه متناهی ندارد.

(ز) مشابه قسمت (ه)، مجموعه جواب بهینه مساله برنامه‌ریزی خطی (۱) که در آن $c^T = [-2, -2]$ را بیابید.

(ح) مشابه قسمت (ه)، مجموعه جواب بهینه مساله برنامه‌ریزی خطی (۱) که در آن $c^T = [0, 1]$ را بیابید. این مساله برنامه‌ریزی خطی را به صورت یک مساله برنامه‌ریزی خطی استاندارد بازنویسی کنید، سپس جواب بهینه این مساله برنامه‌ریزی استاندارد را بیابید.

پاسخ:

(آ) در شکل زیر قسمت هاشور خورده چندوجهی P است.



(ب) مطابق شکل قسمت (آ)، ۳ نقطهٔ رأسی داریم:

$$x^{(1)} = [0,2]^T,\ \ x^{(2)} = [0,1]^T,\ \ x^{(3)} = [2,-1]^T$$

(ج) باز هم مطابق شکل قسمت (آ)، دو جهت دور شوندهٔ رأسی داریم:

$$d_1 = [2,1]^T,\ \ d_2 = [1,1]^T$$

(د)

$$[1/2, 3/2]^T = x^{(2)} + \frac{1}{2}d_2$$
$$[1,1]^T = \frac{3}{4}x^{(2)} + \frac{1}{4}x^{(3)} + \frac{1}{2}d_2$$
$$[3,2]^T = \frac{1}{2}x^{(2)} + \frac{1}{2}x^{(3)} + 2d_2$$

(ه) توجه کنید که

$$c^T d_1 = 2 + 1 = 3 > 0,\ \ c^T d_2 = 1 + 1 = 2 > 0$$

پس مسئله چواب متناهی دارد. از طرف دیگر

$$c^T x^{(1)} = 0 + 2 = 2,\ \ c^T x^{(2)} = 0 + 1 = 1,\ \ c^T x^{(3)} = 2 - 1 = 1$$

پس کم‌ترین مقدار ممکن برابر با 1 است و دسته جواب‌های مسئله به شکل زیر هستند:

$$\{\lambda[0,1]^T + (1-\lambda)[2,-1]^T \mid 0 \le \lambda \le 1\} = \{[2(1-\lambda), 2\lambda - 1]^T \mid 0 \le \lambda \le 1\}$$

(و) دقت کنید که $c^T d_1 = -4 - 1 < 0$ پس مسئله جواب متناهی ندارد.

(ز) مشابه قسمت قبل، $c^T d_1 = -4 - 2 < 0$ پس مسئله جواب متناهی ندارد.

(ح) توجه کنید که

$$c^T d_1 = 1 > 0,\ \ c^T d_2 = 1 > 0$$

پس مسئله چواب متناهی دارد. از طرف دیگر

$$c^T x^{(1)} = 2,\ \ c^T x^{(2)} = 1,\ \ c^T x^{(3)} = -1$$

پس کم‌ترین مقدار ممکن برابر با -1 است که به‌ازای تک‌نقطهٔ $x^{(3)} = [2,-1]^T$ رخ می‌دهد. صورت اولیه و صورت استاندارد مسئله در عبارات زیر آمده است:

$$\begin{cases} \min & x_2 \\ \text{s.t.} & x_1 + x_2 \ge 1 \\ & -x_1 + x_2 \le 2 \\ & x_1 - 2x_2 \le 4 \\ & x_1 \ge 0 \end{cases} \implies \begin{cases} \min & x_2' - x_2'' \\ \text{s.t.} & x_1 + x_2' - x_2'' - x_3 = 1 \\ & -x_1 + x_2' - x_2'' + x_4 = 2 \\ & x_1 - 2x_2' + 2x_2'' + x_5 = 4 \\ & x_1, x_2', x_2'', x_3, x_4, x_5 \ge 0 \end{cases}$$

می‌دانیم جواب مسئله در فرم غیر استاندارد، $[2,-1]^T$ بود. به کمک این جواب، جواب فرم استاندارد را پیدا می‌کنیم. به وضوح $x_1 = 2$ و

$$-1 = x_2 = x_2' - x_2'' \implies \begin{cases} x_2' = 0 \\ x_2'' = 1 \end{cases}$$

۲. حال به‌سادگی مقادیر بقیهٔ متغیرها نیز به‌دست می‌آیند $x_3 = 0,\ x_4 = 5,\ x_5 = 0$. مساله برنامه‌ریزی خطی

$$\begin{cases} \min & -x_1 - x_2 + 2x_3 + x_4 \\ \text{s.t.} & x_1 + x_2 + x_3 + x_4 \geq 6 \\ & x_1 - x_2 - 2x_3 + x_4 \leq 4 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{cases}$$

را در نظر بگیرید.

(آ) با افزودن متغیرهای کمبود و مازاد مساله را به صورت استاندارد بازنویسی کنید. فضای التزام (فضای ترکیب خطی نامنفی بردار ضرایب متغیرها) را در صفحه رسم کنید.

(ب) با استناد به فضای التزام، استدلال کنید که چرا این مساله برنامه‌ریزی خطی شدنی است.

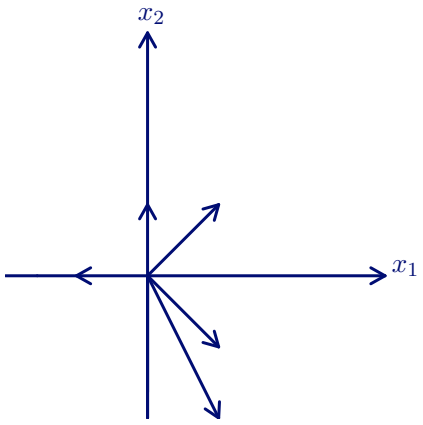
(ج) با فرض این که در هر جواب بهینه این مساله حداکثر دو متغیر مقدار غیر صفر می‌گیرند، و با استفاده از فضای التزام، جواب بهینه مساله را بیابید.

پاسخ:

(آ) فرم استاندارد مسئله:

$$\begin{cases} \min & -x_1 - x_2 + 2x_3 + x_4 \\ \text{s.t.} & x_1 + x_2 + x_3 + x_4 - x_5 = 6 \\ & x_1 - x_2 - 2x_3 + x_4 + x_6 = 4 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0. \end{cases}$$

مطابق شکل زیر فضای التزام مسئله از کل R^2 تشکیل شده است.



(ب) از آن‌جا که نقطهٔ $[6,4]^T$ مطابق شکل قسمت (آ) درون فضای التزام است، مسئله شدنی است.

(ج) قرار می‌دهیم

$$v_1 = [1,1]^T,\ v_2 = [1,-1]^T,\ v_3 = [1,-2]^T,\ v_4 = [1,1]^T,\ v_5 = [-1,0]^T,\ v_6 = [0,1]^T.$$

توجه کنید که مطابق شکل قسمت (آ)، نقطهٔ $[6,4]^T$ در فضای ترکیب خطی نامنفی حاصل از جفت بردارهای (v_1, v_2)، (v_4, v_2)، (v_1, v_3)، (v_4, v_3)، (v_6, v_2) و (v_6, v_3) است. در هر کدام از این حالات جواب مسئله را محاسبه می‌کنیم، به این صورت که برای مثال اگر جفت (v_1, v_2) را انتخاب کردیم، x_1, x_2 را ناصفر و بقیهٔ متغیرها را صفر در نظر می‌گیریم و

با حل دو معادله مقدار x_1, x_2 را به‌دست می‌آوریم. پس حالت‌های زیر را داریم

$$(v_1, v_2) \implies -x_1 - x_2 = -6$$
$$(v_2, v_4) \implies -x_2 + x_4 = 4$$
$$(v_1, v_3) \implies -x_1 + 2x_3 = -4$$
$$(v_3, v_4) \implies 2x_3 + x_4 = \frac{20}{3}$$
$$(v_6, v_2) \implies -x_2 = -6$$
$$(v_6, v_3) \implies 2x_3 = 12$$

در نتیجه کم‌ترین مقدار ممکن -6 است و در دو حالت

$$x_1 = 5,\ x_2 = 1,\ x_3 = x_4 = x_5 = x_6 = 0$$
$$x_2 = 6,\ x_6 = 10,\ x_1 = x_3 = x_4 = x_5 = 0$$

اتفاق می‌افتد.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین سری ۱۴ موعد: پنجشنبه ۱۹ تیر ساعت ۱۲

---

۱. فرض کنید یک شرکت می‌تواند هرکدام از پروژه‌های $A, B, \ldots, H$ را انجام دهد. هرکدام از محدودیت‌های زیر را با استفاده از متغیرهای دودویی $x_a, x_b, \ldots, x_h$ مدل کنید.

(آ) حداکثر یکی از پروژه‌های $A, B, \ldots, H$ انجام شوند.

(ب) حداقل یکی از پروژه های $A, B, \ldots, H$.

(ج) اگر A آنگاه B.

(د) اگر A آنگاه B انجام نشود.

(ه) اگر A انجام نشود B انجام شود.

(و) A اگر و فقط اگر B.

(ز) اگر A، آنگاه B و C.

(ح) اگر A، آنگاه B یا C.

(ط) اگر B یا C آنگاه A.

(ی) اگر B و C آنگاه A.

(ک) اگر دو تا یا بیشتر از B, C, D, E آنگاه A.

پاسخ:

(آ) $x_a + x_b + \cdots + x_h \leq 1$ که حداکثر یکی از آن‌ها یک است.

(ب) $x_a + x_b + \cdots + x_h \geq 1$ که حداقل یکی از آن‌ها یک است.

(ج) $x_a - x_b \leq 0$ که اگر $x_a = 0$ باشد آنگاه x_b متغیر آزاد است و اگر $x_a = 1$ آنگاه $x_b = 1$ است .

(د) $x_a + x_b \leq 1$ که اگر $x_a = 0$ باشد آنگاه x_b متغیر آزاد است و اگر $x_a = 1$ آنگاه $x_b = 0$ است .

(ه) $x_a + x_b \geq 1$ که اگر $x_a = 0$ باشد آنگاه $x_b = 1$ است و اگر $x_a = 1$ آنگاه x_b متغیر آزاد است .

(و) $x_a - x_b = 0$ که اگر $x_a = 0$ باشد آنگاه $x_b = 0$ است و اگر $x_a = 1$ آنگاه $x_b = 1$ است.

(ز) $\begin{cases} x_a - x_b \leq 0 \\ x_a - x_c \leq 0 \end{cases}$ اگر $x_a = 1$ باشد، آنگاه $x_b = 1$ و $x_c = 1$ و اگر $x_a = 0$ آنگاه x_b و x_c متغیر آزاد است.

(ح) $x_a - (x_b + x_c) \leq 0$ که اگر $x_a = 1$ باشد ، آنگاه x_b یا x_c برابر یک هستند و اگر $x_a = 0$ آنگاه x_b و x_c متغیر آزاد است.

(ط) $(x_b + x_c) - 2x_a \leq 0$ اگر B یا C آنگاه $x_b + x_c \geq 1$ و $x_a = 1$ و اگر، نه B و نه C آنگاه $x_b + x_c = 0$ و x_a متغیر آزاد ی) $(x_b + x_c) - 2x_a \leq 0$ اگر B و C آنگاه $x_a = 1$ در غیر این‌صورت x_a متغیر آزاد است.

(ی) $x_b + x_c + x_d + x_e - 4x_a \leq 1$ تا هنگامی که $x_b + x_c + x_d + x_e \leq 1$ آنگاه x_a متغیر آزاد است و اگر $x_b + x_c + x_d + x_e \geq 2$ آنگاه $x_a = 1$ است.

۲. گراف بدون جهت $G = (V, E)$ را در نظر بگیرید که $V = \{v_1, \ldots, v_n\}$. رأس v_1 نشانگر یک پیتزافروشی است که برای سادگی فرض می‌کنیم فقط یک نوع پیتزا دارد. هرکدام از بقیه رأس‌ها نشانگر یک مشتری است که مشتری v_i می‌خواهد b_i تا پیتزا بخرد. همچنین به هر یال $e \in E$ هزینه c_e نسبت داده شده است. پیتزافروشی m پیک دارد که تعداد پیتزاهایی که هر پیک می‌تواند حمل کند Q است. هر پیک باید از مبدأ (v_1) شروع کند، و بعد از اینکه به تعدادی مشتری سرویس داد در نهایت به v_1 بازگردد. فرض می‌کنیم $b_i \leq Q$ برای هر i، و اینکه سفارش هر مشتری باید توسط یک پیک تحویل داده شود. می‌خواهیم اختصاص دادن سفارش مشتری‌ها به پیک‌های مختلف و مسیر پیک‌ها را طوری برنامه‌ریزی کنیم که کل مسیر پیمایش شده توسط پیک‌ها کمینه شود. این مسئله را توسط یک برنامه صحیح مدل کنید.

پاسخ:

فرض کنیم $G_1, G_2, \ldots, G_l$ همه گشت‌هایی از گراف G باشند که از رأس v_1 عبور می‌کنند.هزینه هر یک از این گشت‌ها(مجموع هزینه یال‌های گشت) را با $f(G_i)$ $(1 \leq i \leq l)$ نمایش می‌دهیم($\sum_{e \in G_i} c_e = f(G_i)$)

$(1 \leq i \leq l)$ را یک متغیر دودویی در نظر بگیریم وداریم :

$$\begin{cases} x_i = 1 & \text{اگر دور } x_i \text{ انتخاب شود} \\ x_i = 0 & \text{اگر دور } x_i \text{ انتخاب نشود} \end{cases}$$

اگر بتوانیم شرط‌های برنامه ریزی خطی را طوری تعیین کنیم به طوری‌که $x_i = 1$ است اگر و تنها اگر یک موتورسوار در گشت G_i حرکت کند،آنگاه هدف مسئله یافتن $\min \sum_{i=1}^{l} f(G_i)x_i$ خواهدشد. شرط مورد نیاز در اینجا ، $\sum_{i=1}^{l} x_i \leq m$ است.(پیتزافروشی حداکثر m پیک دارد.)

حال متغیرهای دودویی y_{ij} تعریف می‌کنیم: ($2 \leq i \leq n$ و $1 \leq j \leq l$)

$y_ij = 1$ است اگر و تنها اگر رأس v_i توسط پیک‌موتوری که از گشت j می‌گذرد، سفارش خود را تحویل بگیرد.

شرط‌های مورد نیاز در اینجا

$\forall_{2 \leq i \leq n} i : \sum_{j=1}^{l} y_{ij} = 1$ است(هر مشتری دقیقاً از طریق پیکی که از گشت خاص می‌گذرد،سفارشش را دریافت کند).

به‌علاوه شرطی نیاز داریم که اگر گشت G_t انتخاب نشده است، $\forall_{2 \leq i \leq n} i : y_{it} = 0$ باشد.

با توجه به این‌که n عددی ثابت است، شرط مورد نظر را به این شکل می‌نویسیم: $nx_t - \sum_{i=2}^{n} y_{it} \geq 0$

بدین ترتیب اگر $x_t = 1$ باشد، مشکلی نیست اکا اگر $x_t = 1$ ، همه y_{it} ها باید صفر باشند .

همین‌طور باید به این نکته توجه کنیم که ممکن است از گشت G_i لازم باشد که z_i پیک عبور کنند که می‌دانیم که z_i ها اعداد صحیح نامنفی‌اند و $\sum_{i=1}^{l} z_i \leq m$ باید باشد.

از طرفی اگر گشت G_i انتخاب نشده است، می‌بایست که $z_i = 0$ باشد. همینطور شرط $z_i \leq mx_i$ $(1 \leq i \leq l)$ با توجه به این‌که حداکثر m پیک داریم، کمک‌کننده است.

همینطور متغیر w_{ij} $(1 \leq i \leq m$ و $1 \leq j \leq l)$ را اضافه می‌کنیم و داریم که $w_{ij} = 1$ است اگر و تنها اگر پیک i ام از گشت j ام عبور کند.شرط $\sum_{j=1}^{l} w_{ij} \leq 1$ سبب می‌شود تا پیک i نتواند بیش از یک مسیر ممکن داشته باشد. ضمناً

$\forall_{1 \leq j \leq l} j : \sum_{i=1}^{m} w_{ij} = z_{ij}$ است؛ زیرا از گشت j ، z_j پیک عبور می‌کند.

متغیر دیگری نیز تعریف می‌کنیم که λ_{ij} ($1 \leq j \leq m$ و $2 \leq i \leq n$) که داریم : $\lambda_{ij} = 1$ اگر و تنها اگر سفارش مشتری i ام توسط پیک j ام برطرف شود.

شرط $\sum_{j=1}^{m} \lambda_{ij} = 1$ برای هر i برقرار باشد که هر مطمئن باشیم که هر مشتری دقیقاً از یک پیک سفارش دریافت می‌کند . ضمناً $\sum_{i=2}^{n} \lambda_{ij} b_i \leq Q$ باید برقرار باشد که هر پیک بتواند همه پیتزاهای مشتری‌هایش را تأمین کند. از طرفی اگر $y_{it} = 1$ و $\lambda_{ij} = 1$ است،می‌بایست که پیک t از گشت j عبور کند و این یعنی که $w_{tj} = 1$ است.پس باید برای هر $2 \leq i \leq n$ و $1 \leq t \leq l$ و $1 \leq j \leq m$ باید شرط $yit + \lambda_{ij} - w_{tj} \leq 1$ برقرار باشد که یعنی اگر پیک j نیاز مشتری i را تأمین می‌کند و اگر نیاز مشتری i از طریق گشت G_t تأمین می‌شود، آنگاه پیک j باید از مسیر i عبور کند.

پس در نهایت داریم :

$$
\begin{cases}
\min \sum_{i=1}^{l} f(G_i) x_i & \text{تابع هدف} \\
\\
\sum_{i=1}^{l} x_i \leq m & \\
\sum_{j=1}^{l} y_{ij} = 1 & 2 \leq i \leq n \\
n x_j - \sum_{i=2}^{n} y_{ij} \geq 0 & 1 \leq j \leq l \\
\sum_{i=1}^{l} z_i \leq m & \\
z_i \leq m x_i & 1 \leq i \leq l \\
\sum_{j=1}^{l} w_{ij} \leq 1 & 1 \leq i \leq m \\
\sum_{i=1}^{m} w_{ij} = z_{ij} & 1 \leq j \leq l \\
\sum_{j=1}^{m} \lambda_{ij} = 1 & 1 \leq j \leq m, 1 \leq i \leq n \\
\sum_{i=2}^{n} \lambda_{ij} b_i \leq Q & 1 \leq j \leq m \\
yit + \lambda_{ij} - w_{tj} \leq 1 & 1 \leq j \leq m, 1 \leq t \leq l, 2 \leq i \leq n \\
x_i, \lambda_{ij}, y_{ij}, w_{ij} \in \{0, 1\} & \\
z_i \in \mathbf{N} \cup \{0\} &
\end{cases}
$$

موفّق باشید.

بخش هفتم

پاسخ تمرین‌های اضافه

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین اضافه سری ۱

---

۱. فرض کنید قرار است برای اولین بار درس الگوریتم‌های پسرفته در دانشکده ارائه شود. n نفر کاندیدای TA شدن در این درس هستند و ما می‌خواهیم دقیقاً یک نفر را انتخاب کنیم. مکانیزم انتخاب به این صورت است که به ترتیب با کاندیداها مصاحبه می‌کنیم، و بعد از مصاحبه با هرکدام، برای اینکه به وی استرس وارد نشود، بلافاصله باید تصمیم بگیریم که آیا این دانشجو را به عنوان TA انتخاب کنیم یا خیر و تصمیممان را به او اعلام کنیم. فرض کنید تنها چیزی که از مصاحبه با یک دانشجو می‌فهمیم این باشد که آیا او از همه کاندیداهای قبلی بهتر است یا خیر. نشان دهید هر الگوریتم تصادفی به احتمال حداکثر $1/n$ (روی بدترین دنباله ورودی برای آن الگوریتم) بهترین TA را انتخاب می‌کند.

راهنمایی ۱: از اصل مینیمکس یائو استفاده کنید.

راهنمایی ۲: دنباله‌هایی از کاندیداها را در نظر بگیرید که تا رسیدن به بهترین کاندیدا صعودی هستند.

راهنمایی برای پاسخ: فرض کنید بهترین کاندیدا به‌طور تصادفی و با توزیع یکنواخت یکی از افراد ۱ تا n باشد و تا زمان رسیدن به بهترین کاندیدا، دنباله کاندیداها صعودی باشد. فرض کنید دنباله کاندیداها به صورت $a_1 < a_2 < \ldots < a_k > \ldots > a_n$ باشد. هر الگوریتم قطعی A تا زمان رسیدن به بهترین کاندیدا، تعدادی کاندیدا می‌بیند که هر کدام از همه قبلی‌ها بهتر است. بنابراین نگاه کردن به ورودی هیچ اطلاعات مفیدی به A نمی‌دهد؛ به محض اینکه A کاندیدایی را ببیند که از قبلی‌ها بهتر نیست، شانسی برای انتخاب بهترین کاندیدا نخواهد داشت. بنابراین می‌توانیم فرض کنیم A بدون نگاه به ورودی یک اندیس ۱ تا n مثل k را انتخاب می‌کند، و طبق نحوه تعریف توزیع ورودی، احتمال اینکه بهترین کاندیدا a_k باشد $1/n$ است. پس احتمال موفقیت A $1/n$ است و بنابراین طبق اصل مینیمکس یائو، هیچ الگوریتم تصادفی نمی‌تواند در بدترین حالت به احتمال بهتر از $1/n$ موفق شود.

۲. نشان دهید مشابه نتیجه‌ای که در درس برای حالت صفر و یکی مسئله خبرگان بیان شد، در حالت پیوسته هم برقرار است. به‌طور دقیق‌تر، فرض کنید بعد از روز tام، ضرر $c_i^t \in [0, 1]$ برای خبره iام مشخص شود، و سپس وزن خبره iام را در $(1 - c_i^t\epsilon)$ ضرب کنیم. همچنین هر خبره را در هر روز با احتمال متناسب با وزنش انتخاب می‌کنیم. نشان دهید بعد از T روز، امید ریاضی ضرر این الگوریتم حداکثر برابر است با $(1+\epsilon)M + \frac{\ln n}{\epsilon}$ که M ضرر بهترین خبره بعد از T روز است.

راهنمایی برای پاسخ: فرض کنید w_i^t و c_i^t وزن و ضرر خبره iام در روز tام باشد. همچنین L_t را ضرر الگوریتم RWM در روز tام و L را کل ضرر الگوریتم در نظر بگیرید و تعریف کنید $\Phi_t = \sum_i w_i^t$. داریم

$$\Phi_{t+1} = \sum_i (1 - c_i^t\epsilon) w_i^t = \Phi_t \sum_i (1 - c_i^t\epsilon)\frac{w_i^t}{\Phi_t} = \Phi_t \left[\sum_i \frac{w_i^t}{\Phi_t} - \epsilon \sum_i \frac{w_i^t}{\Phi_t} c_i^t\right] = \Phi_t(1 - \epsilon \mathsf{E}[L_t]).$$

همچنین

$$w_i^T = \prod_{t=1}^{T} (1 - c_i^t\epsilon) \le \Phi_T$$

و

$$\Phi_T = n. \prod_{t=1}^{T} (1 - \epsilon \mathsf{E}[L_t]) \le n e^{-\epsilon \sum_t \mathsf{E}[L_t]} = n e^{-\epsilon \mathsf{E}[L]}.$$

در نتیجه

$$\sum_t \ln(۱ - c_i^t \epsilon) \leq -\epsilon \mathsf{E}[L] + \ln n.$$

پس

$$\begin{aligned}\epsilon \mathsf{E}[L] &\leq -\sum_t \ln(۱ - c_i^t \epsilon) + \ln n \\ &\leq \sum_t \left(c_i^t \epsilon + (c_i^t)^۲ \epsilon^۲ \right)\end{aligned}$$

یا

$$\mathsf{E}[L] \leq \sum_t c_i^t + \epsilon \sum_t (c_i^t)^۲ + \frac{\ln n}{\epsilon} \leq (۱ + \epsilon) \sum_t c_i^t + \frac{\ln n}{\epsilon}.$$

که حکم خواسته شده است، چون $\sum_t c_i^t$ جمع ضرر خبره iام است.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین اضافه سری ۲

---

۱. مسئله ۳ـ رنگ‌پذیری را در نظر بگیرید. الگوریتم بدیهی این مسئله نیاز به زمان ۳^n دارد. می‌خواهیم یک الگوریتم تصادفی با زمان $(\frac{۳}{۲})^n\text{poly}(n)$ برای این مسئله طراحی کنیم. فرض کنید به طور تصادفی رنگ هر رأس را به ۲ تا از ۳ گزینه ممکن محدود کنیم. نشان دهید تشخیص اینکه آیا می‌توان گراف را با این مجموعه رنگ‌هایی که برای هر رأس مشخص کرده‌ایم رنگ کرد یا نه را می‌توان در زمان چندجمله‌ای حل کرد (از مسئله 2SAT استفاده کنید). سپس نتیجه بگیرید که در صورتی که گراف ۳ـ رنگ‌پذیر باشد، می‌توان با احتمال بالا یک ۳ـ رنگ‌آمیزی از آن را در زمان $(\frac{۳}{۲})^n\text{poly}(n)$ پیدا کرد.

راهنمایی برای پاسخ: برای تحویل مسئله رنگ‌آمیزی گفته شده به 2SAT، برای هر رأس $v \in V$، سه متغیر v_r, v_g, v_b را در نظر می‌گیریم. برای هر رأس عبارتی که شامل متغیرهای متناظر با دو رنگی است که برای آن رأس مجاز هستند را در فرمولمان قرار می‌دهیم. مثلاً در صورتی که رنگ v به یکی از دو رنگ آبی و قرمز محدود شده باشد، عبارت $(v_r \vee v_b)$ را قرار می‌دهیم. همچنین برای هر یال $uv \in E$، سه عبارت $(\overline{v_r} \vee \overline{u_r})$، $(\overline{v_b} \vee \overline{u_b})$، و $(\overline{v_g} \vee \overline{u_g})$ را به فرمولمان اضافه می‌کنیم.

حال در صورتی که یک ۳ـ رنگ‌آمیزی C برای گراف وجود باشد و رنگ هر رأس را به صورت تصادفی به ۲ رنگ از ۳ رنگ ممکن محدود کنیم، به احتمال $\frac{۲}{۳}$ رنگ هر رأس در C جزء دو رنگ انتخاب شده برای آن رأس خواهد بود و به احتمال $(\frac{۲}{۳})^n$ رنگ همه رأس‌ها در C جزء دو رنگ انتخاب شده‌شان خواهد بود. بنابراین طبق بحثی که در کلاس درس مطرح شد، برای اینکه با احتمال بالا موفق شویم، کافی است این کار را

$$O(\frac{۱}{(۲/۳)^n}\lg[(۲/۳)^n]) = (\frac{۳}{۲})^n\text{poly}(n)$$

بار تکرار کنیم.

۲. فرض کنید مجموعه n عضوی A به عنوان ورودی داده شده باشد. یک نمونه تصادفی با سایز $s = n^{\frac{۳}{۴}}$ از A انتخاب می‌کنیم (با جایگذاری). فرض کنید $x_۱ \le x_۲ \le \ldots \le x_s$ نمونه انتخاب شده باشد. نشان دهید احتمال اینکه تعداد عناصری از A که بین $x_{\frac{s}{۲}-\sqrt{n}}$ و $x_{\frac{s}{۲}+\sqrt{n}}$ هستند بیشتر از $۴n^{\frac{۳}{۴}}$ باشد، $O(n^{-\frac{۱}{۴}})$ است.

راهنمایی برای پاسخ: فرض کنید اعداد مجموعه ورودی $y_n > \ldots > y_۲ > y_۱$ باشند. نشان می‌دهیم احتمال اینکه $x_{\frac{s}{۲}-\sqrt{n}} < y_{\frac{n}{۲}-۲n^{۳/۴}}$، $O(n^{-۱/۴})$ است و همچنین احتمال اینکه $x_{\frac{s}{۲}+\sqrt{n}} > y_{\frac{n}{۲}+۲n^{۳/۴}}$ هم $O(n^{-۱/۴})$ است. در این صورت طبق کران اجتماع، به احتمال $۱ - O(n^{-۱/۴})$ عناصر $x_{\frac{s}{۲}-\sqrt{n}}$ و $x_{\frac{s}{۲}+\sqrt{n}}$ در بازه $[y_{\frac{n}{۲}-۲n^{۳/۴}}, y_{\frac{n}{۲}+۲n^{۳/۴}}]$ قرار می‌گیرند و بنابراین تعداد عناصری که بینشان است حداکثر $۴n^{۳/۴}$ است.

X را متغیر تصادفی تعدادی از x_iها در نظر بگیرید که کمتر از $y_{\frac{n}{۲}-۲n^{۳/۴}}$ هستند. در این صورت X با توزیع دوجمله‌ای است و $E[X] = \frac{n^{۳/۴}}{۲} - ۲\sqrt{n}$ و $Var[X] = O(n^{۳/۸})$. پس طبق نامساوی چبیشف،

$$Pr[X \ge \frac{n^{۳/۴}}{۲} - \sqrt{n}] = Pr[X - E[X] \ge \sqrt{n}] \le \frac{O(n^{۳/۸})}{\sqrt{n}} = O(n^{-۱/۴}).$$

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

تمرین اضافه سری ۳

---

۱. یک مجموعه n عضوی A و m مجموعه $S_1, \ldots, S_m \subseteq A$ را در نظر بگیرید. نشان دهید می‌توان اعضای A را به گونه‌ای با دو رنگ آبی و قرمز رنگ کرد که اختلاف تعداد اعضای آبی و قرمز هرکدام از S_iها $O(\sqrt{n \lg m})$ باشد. توجه کنید صرفاً باید وجود چنین رنگ‌آمیزی‌ای را اثبات کنید.

راهنمایی: اعضا را تصادفی رنگ کنید و با استفاده از کران چرنف و کران اجتماع نشان دهید به احتمال غیرصفر خاصیت گفته شده برقرار است. نتیجه بگیرید یک رنگ‌آمیزی خوب وجود دارد.

راهنمایی برای پاسخ: عناصر را با احتمال برابر با آبی یا قرمز رنگ می‌کنیم. مجموعه S_1 را در نظر بگیرید. فرض کنید $|S_1| = n$. امید تعداد اعضای آبی S_1 برابر با $n/2$ است. احتمال اینکه تعداد آبی‌های S_1 بیشتر از $n/2 + 4\sqrt{n \ln m}$ باشد طبق چرنوف حداکثر برابر است با $e^{-\frac{n}{2}(4\frac{\sqrt{\ln m}}{\sqrt{n}})^2/3} \leq m^{-2}$. احتمال اینکه تعداد قرمزها بیشتر از این مقدار باشد هم همین قدر است، بنابراین احتمال اینکه اختلاف تعداد آبی‌ها و قرمزها بیشتر از $8\sqrt{n \ln m}$ باشد $O(\frac{1}{m^2})$ است. اگر نشان دهیم در صورتی که تعداد اعضای S_1 کمتر از n باشد احتمال اینکه تعداد آبی‌ها به اندازه $4\sqrt{n \ln m}$ بیشتر از امیدش باشد همچنان $O(\frac{1}{m^2})$ است، می‌توانیم نتیجه بگیریم احتمال اینکه این اتفاق برای حداقل یکی از S_iها بیفتد طبق کران اجتماع $O(\frac{1}{m})$ است.

فرض کنید $|S_1| < n$. کل اعضای A را به‌طور تصادفی و به احتمال برابر رنگ می‌کنیم. X را این پیشامد در نظر بگیرید که تعداد آبی‌های S_1 حداقل $|S_1|/2 + 4\sqrt{n \ln m}$ باشد و Y را این پیشامد در نظر بگیرید که تعداد آبی‌های A حداقل $n/2 + 4\sqrt{n \ln m}$ باشد. از آنجا که طبق تقارن به احتمال حداقل ۱/۲ تعداد اعضای آبی $A \setminus S_1$ حداقل $|A \setminus S_1|/2$ است، $Pr[Y \mid X] \geq 1/2$. پس $Pr[Y] \geq 1/2 Pr[X]$، یا $Pr[X] \leq 2Pr[Y]$.

۲. یک الگوریتم چندجمله‌ای تصادفی ارائه دهید که دو درخت ریشه‌دار را به عنوان ورودی بگیرد و تشخیص دهد آیا این دو درخت یکریخت هستند یا نه. توجه کنید فرزندان گره‌ها ترتیب ندارند.

راهنمایی: برای هر زیردرخت یک چندجمله‌ای تعریف کنید به طوری که یکریخت بودن دو زیردرخت معادل با یکسان بودن چندجمله‌ای‌های آن‌ها باشد.

راهنمایی برای پاسخ: فرض کنید ارتفاع درخت T برابر با h باشد. به‌طور استقرایی به هر گره v از T یک چندجمله‌ای f_v روی متغیرهای $x_0, \ldots, x_h$ نسبت می‌دهیم. در صورتی که v برگ باشد، تعریف می‌کنیم $f_v = x_0$. در صورتی که v یک گره با ارتفاع $k > 0$ باشد و بچه‌های آن $v_1, v_2, \ldots, v_l$ باشند، تعریف می‌کنیم $f_v = (x_k - f_{v_1})(x_k - f_{v_2}) \ldots (x_k - f_{v_l})$.

بخش هشتم

تمرین‌های عملی

تاج‌گذاری در آتلانتیس

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

آکوامن برای شرکت در مراسم تاج‌گذاری شاه‌زاده به آتلانتیس رفته‌است. با وجود این که او بارها وارد قصر شاه‌زاده شده‌است اما به خاطر هوش ضعیفش هیچ‌وقت مسیرها را یاد نمی‌گیرد. او که حوصله‌اش سر رفته‌است از سالن مراسم خارج شده و قصد خروج از محوطه‌ی قصر را دارد. اما متوجه می‌شود که به تنهایی راه برگشت را نمی‌تواند پیدا کند. در محوطه‌ی قصر n تقاطع وجود دارد که با اعداد ۱ تا n شماره‌گذاری شده‌اند. در آن‌جا m جاده‌ی دوطرفه وجود دارد که هر کدام از آن‌ها جفتی از تقاطع‌ها را به هم وصل می‌کند. در k تا از تقاطع‌ها پری‌های دریایی برای راه‌نمایی مهمان‌ها حاضر هستند. مکان این پری‌های دریایی ثابت است و اجازه‌ی جابه‌جایی ندارند. وقتی آکوامن از یک پری دریایی راه خروج را می‌پرسد پری دریایی می‌تواند کل مسیر را به او بگوید. ولی این مسیرها خیلی شبیه به هم هستند و آکوامن که اصلا باهوش نیست تنها می‌تواند q تقاطع (به جز تقاطعی که در آن است) را بعد از پرسیدن مسیر به خاطر بسپارد. او همیشه با پری‌های دریایی گرم می‌گیرد و به آن‌ها می‌گوید که پیدا کردن مسیرها برایش کار دشواری است و از آن‌ها می‌پرسد که آیا مسیری مستقیم با طول حداکثر q (به تعداد جاده‌ها) برای خروج وجود دارد یا نه. در صورتی که چنین مسیری وجود داشته باشد، پری دریایی آن مسیر را به آکوامن می‌گوید. در غیر این صورت پری دریایی او را به پری دریایی دیگری که فاصله‌اش با آن‌جا حداکثر q جاده است هدایت می‌کند. پری‌های دریایی مسیرها را به خوبی بلد هستند و آکوامن را به بهترین نحو هدایت می‌کنند. پس اگر راه خروجی وجود داشته باشد آکوامن حتما آن را خواهد یافت!

مکان اولیه‌ی آکوامن تقاطع s و مکان خروج تقاطع t است. همیشه یک پری دریایی در تقاطع s وجود دارد. کار شما این است که کمینه‌ی مقدار q را طوری پیدا کنید که آکوامن حتما بتواند راه خروج را پیدا کند.

▼ Tags

Binary Search - BFS

▼ راه‌نمایی

ابتدا سعی کنید مسئله را به صورت بله و خیر با داده‌شدن یک q مشخص حل کنید و سپس با جستجوی دودویی روی q مشخص سعی در حل مسئله‌ی اصلی کنید.

ورودی

خط اول ورودی شامل سه عدد صحیح n و m و k است که با فاصله از هم آمده‌اند و به ترتیب نشان‌دهنده‌ی تعداد تقاطع‌ها، تعداد جاده‌ها و تعداد پری‌های دریایی هستند.

$$2 \leq n \leq 10^5,\ 0 \leq m \leq 2 \times 10^5,\ 1 \leq k \leq n$$

خط بعد شامل k عدد صحیح متمایز است که با فاصله از هم جدا شده‌اند و به ترتیب صعودی آمده‌اند و نشان‌دهنده‌ی شماره‌ی تقاطع‌هایی هستند که پری‌های دریایی در آن‌ها حضور دارند.

در m خط بعدی جاده‌ها توصیف می‌شوند. خط iاُم شامل دو عدد صحیح u_i وv_i است که نشان‌دهنده‌ی شماره‌ی تقاطع‌هایی است که جاده‌ی iاُم آن‌ها را متصل می‌سازد. بین هر جفت تقاطع حداکثر یک جاده وجود دارد.

$$1 \leq u_i, v_i \leq n,\ u_i \neq v_i$$

آخرین خط ورودی شامل دو عدد صحیح s و t است که به ترتیب نشان‌دهنده‌ی تقاطعی که آکوامن در آن قرار دارد و تقاطعی که هدف نهایی برای خروج است هستند. ممکن است راهی از تقاطع s به تقاطع t وجود نداشته باشد.

تضمین می‌شود که در تقاطع s یک پری دریایی وجود دارد.

خروجی

خروجی شامل یک عدد صحیح است که نشان دهنده ی کمینه‌ی مقدار q می‌باشد. اگر چنین مقداری وجود نداشت عدد -1 را چاپ کنید.

مثال

ورودی نمونه ۱

```
6 6 3
1 3 6
1 2
```

```
2 3
4 2
5 6
4 5
3 4
1 6
```

خروجی نمونه ۱

```
3
```

ورودی نمونه ۲

```
6 5 3
1 5 6
1 2
2 3
3 4
4 5
6 3
1 5
```

خروجی نمونه ۲

```
3
```

نمرات کونگ فو

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

اوگ‌وی در شهر گنگمن یک استاد معروف کونگ فو است. او پس از یک ترم تدریس قصد دارد نمرات شاگردانش را وارد سامانه‌ی 教育 (سامانه‌ی ثبت نمرات کونگ فو) کند تا اولیای شاگردانش در جریان نمرات قرار بگیرند. اما چون این سامانه سرعت بسیار پایینی دارد او قصد دارد کم‌ترین داده‌ی ممکن را برای وارد کردن نمرات منتقل کند. نمرات هر کدام از شاگردها به صورت یک جدول n در m است. بعضی از خانه‌های جدول خالی هستند که در آن‌ها یک نقطه (".") نوشته شده‌است و بعضی دیگر با یک حرف از حروف الفبای انگلیسی پر شده‌اند (به این دلیل که برخی از شاگردان استاد اوگ‌وی قصد دارند در آینده در دیگر کشورها باشگاه آموزش کونگ فو تأسیس کنند، استاد اوگ‌وی از حروف چینی استفاده نمی‌کند تا شاگردانش نیازی به ترجمه‌ی مدارک خود نداشته باشند.). یک حرف می‌تواند چند بار در جدول نمرات تکرار شده باشد. هنگام منتقل کردن هر جدول مانند A دو کار را می‌توان انجام داد:

۱) می‌توانید کل جدول A را منتقل کنید. برای این کار باید $n \times m$ بایت را منتقل کنید.

۲) در صورتی که جدولی مانند B را قبلا منتقل کرده باشید، می‌توانید اختلاف بین جدول A و جدول B را منتقل کنید. برای این کار باید $d_{A,B} \times w$ بایت را منتقل کنید که $d_{A,B}$ تعداد خانه‌هایی از جدول است که در A و B متفاوت هستند و w یک عدد طبیعی ثابت است. توجه کنید که شما تنها می‌توانید خانه‌های متناظر در جدول‌ها را با هم مقایسه کنید و نمی‌توانید جدول‌ها را تغییر دهید یا نسبت به یک‌دیگر بچرخانید یا تکان دهید.

کار شما این است که به استاد اوگ‌وی کمک کنید تا بفهمد برای انتقال داده‌های مربوط به نمرات k شاگردش حداقل چه مقدار داده را باید جابه‌جا کند.

▾ Tags

MST

▾ راه‌نمایی

یک گراف وزن‌دار بسازید که وزن یال بین راس iم و jم این باشد که اگه بخواهیم یکی از این جدول‌ها را از

دیگری بسازیم چه‌قدر باید هزینه بدهیم.

▼ راه‌نمایی ۲

سعی کنید با راه‌نمایی نخست و تگ سوال، سوال را حل کنید.

▼ راه‌نمایی ۳

با اضافه‌کردن یک راس اضافه به این گراف مشکل ساختن جدول به صورت مستقل را حل کنید.

ورودی

ابتدا اعداد صحیح n و m و k و w به ترتیب داده می‌شوند.

$$1 \leq n, m \leq 10,\ 1 \leq k, w \leq 1000$$

سپس توضیحات مربوط به جدول نمرات هر کدام از k شاگرد داده می‌شود. هر جدول با n خط توضیح داده می‌شود و هر خط شامل m حرف است. هر حرف به صورت حرفی از حروف الفبای انگلیسی و یا یک نقطه (".") است. به کوچک یا بزرگ بودن حروف توجه کنید.

خروجی

در خط اول کمینه‌ی بایت‌های لازم برای انتقال داده‌های مربوط به همه‌ی شاگردان را چاپ کنید.

سپس k جفت عدد $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ را چاپ کنید که نشان‌دهنده‌ی نحوه‌ی انتقال داده‌های مربوط به هر جدول است.

جفت (x_i, y_i) نشان می‌دهد که جدول نمرات x_iاُم به روش y_i منتقل می‌شود. اگر y_i برابر با صفر باشد نشان می‌دهد که این جدول به روش شماره‌ی ۱ منتقل می‌شود. در غیر این صورت y_i شماره‌ی جدولی است که قبل از جدول x_i منتقل شده‌است و برای انتقال جدول x_i اختلاف آن‌ها فرستاده می‌شود.

جفت‌ها را به ترتیبی که جدول‌ها منتقل می‌شوند چاپ کنید. جدول‌ها به ترتیبی که در ورودی داده می‌شوند از ۱ تا k شماره‌گذاری شده‌اند. اگر جواب‌های مختلفی وجود داشت یکی از آن‌ها را چاپ کنید.

مثال

ورودی نمونه ۱

```
2 3 3 2
A.A
...
A.a
..C
X.Y
...
```

خروجی نمونه ۱

```
14
1 0
2 1
3 1
```

ورودی نمونه ۲

```
1 1 4 1
A
.
B
.
```

خروجی نمونه ۲

```
3
1 0
2 0
4 2
3 0
```

ورودی نمونه ۳

```
1 3 5 2
ABA
BBB
```

```
BBA
BAB
ABB
```

خروجی نمونه ۳

```
11
1 0
3 1
2 3
4 2
5 1
```

جنگ خروسی

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۵۱۲ مگابایت

هفته‌ی آینده در شهر «خروسان» مسابقه‌ی خروس‌جنگی‌ها برگزار خواهد شد. اما هنوز شرکت‌کنندگان این مسابقه و هم‌چنین داورها انتخاب نشده‌اند. شهر خروسان دارای n شهروند و n خروس است. هر شهروند دقیقا یک خروس‌جنگی در خانه‌ی خود دارد. شهروندها و خروس‌ها با شماره‌های ۱ تا n شماره‌گذاری شده‌اند. به طوری که خروس‌جنگی iاُم در خانه‌ی شهروند iاُم زندگی می‌کند. هر شهروند خروسان با تعدادی از خروس‌جنگی‌ها رفیق است؛ از جمله خروس‌جنگی خودش که در خانه‌اش زندگی می کند. بدیهی است که کسی دوست ندارد شکست رفیق‌هایش را ببیند.

برای این که یک مسابقه برگزار شود باید حداقل یک خروس‌جنگی و حداقل یک داور داشته باشیم. و برای این که عدالت برقرار شود هیچ داوری نباید با هیچ‌کدام از خروس‌جنگی‌های شرکت‌کننده در مسابقه رفیق باشد. هم‌چنین برای موفقیت مسابقات در جذب تماشاچی باید مجموع تعداد داورها و خروس‌جنگی‌های شرکت‌کننده n باشد.

به شهروندان خروسان کمک کنید تا گروه داوران و خروس‌جنگی‌ها را مشخص کنند و یا به آن‌ها بگویید که این کار غیرممکن است.

▼ Tags

DFS

▼ راه‌نمایی

با اطلاعات داده‌شده برای هر تست‌کیس از t تست‌کیس، یک گراف جهت‌دار با n رأس و m یال بسازید. اگر در این گراف از رأس u به رأس v $(u \neq v)$ مسیر جهت‌دار داشته باشیم، از بین ۴ حالت ممکن برای قرار دادن u و v در یکی از دو دسته‌ی داوران و شرکت‌کنندگان، چه حالت(های)ی معتبر نیست؟

ورودی

خط اول ورودی شامل یک عدد صحیح t است که تعداد تست‌کیس‌ها را مشخص می‌کند.

($1 \leq t \leq 100000$) سپس توضیحات هر کدام از t تست‌کیس ورودی داده می‌شود. توضیحات هر تست‌کیس به صورت زیر است:

خط اول شامل اعداد طبیعی n و m است. ($1 \leq n \leq m \leq 1000000$) عدد n نشان‌دهنده‌ی تعداد شهروندان و عدد m نشان‌دهنده‌ی تعداد رفاقت‌ها بین شهروندان و خروس‌جنگی‌هاست.

هر کدام از m خط بعدی شامل اعداد صحیح a_i و b_i است. ($1 \leq a_i, b_i \leq n$) که نشان‌دهنده‌ی رفاقت شهروند a_iاُم با خروس‌جنگی b_iاُم است. تضمین می‌شود که هیچ دو جفت یکسانی در این m جفت وجود ندارد.

تضمین می‌شود که به ازای هر $1 \leq i \leq n$، جفتی شامل شهروند iاُم و خروس‌جنگی iاُم وجود دارد.

تست‌کیس‌ها با یک خط خالی از هم جدا شده‌اند.

مجموع nها و مجموع mهای تست‌کیس‌ها حداکثر 10^6 است.

خروجی

برای هر تست‌کیس موارد زیر را چاپ کنید:

اگر پیدا کردن گروه داوران و خروس‌جنگی‌ها با ویژگی‌های خواسته‌شده امکان پذیر نیست No را چاپ کنید.

در غیر این صورت در خط نخست Yes را چاپ کنید. سپس در خط دوم j و p را چاپ کنید که به ترتیب تعداد داوران مسابقه و خروس‌جنگی‌های شرکت‌کننده در آن را نشان می‌دهد. سه شرط زیر باید برقرار باشند:

$$p \geq 1,\ j \geq 1,\ j + p = n$$

در خط سوم j عدد طبیعی چاپ کنید که شماره‌های مربوط به داورهای مسابقه هستند.

در خط چهارم p عدد طبیعی چاپ کنید که شماره‌های مربوط به خروس‌جنگی‌های انتخاب‌شده هستند.

در صورتی که چند جواب مختلف برای یک تست‌کیس وجود دارد یکی را به دل‌خواه چاپ کنید.

مثال

ورودی نمونه ۱

```
4
3 4
1 1
2 2
3 3
1 3

3 7
1 1
1 2
1 3
2 2
3 1
3 2
3 3

1 1
1 1

2 4
1 1
1 2
2 1
2 2
```

خروجی نمونه ۱

```
Yes
2 1
1 3
2
Yes
1 2
2
1 3
No
No
```

شب ددلاین

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

دانش‌جوی خوش‌بخت دانشگاه شریف به خاطر حذف شدن فرجه‌ها مجبور است شبانه‌روز درس بخواند. او که دارد از شدت خواب‌آلودگی می‌میرد تصمیم می‌گیرد که بخوابد اما نگران است که سریع خوابش نبرد. برای همین تصمیم می‌گیرد از معجون‌های خواب‌آور هم‌اتاقی‌اش استفاده کند تا هر چه سریع‌تر خوابش ببرد و وقتی را هدر ندهد. (همان‌طور که می‌دانید این کار اصلا کار درستی نیست و هیچ دانش‌جوی عاقلی این کار را نمی‌کند.)

هم‌اتاقی او n فنجان دارد و باید دقیقا دو تا از آن‌ها را مخلوط کند تا یک معجون خواب‌آور ساخته شود. قدرت خواب‌آوری فنجان iاُم برابر با a_i است. قدرت خواب‌آوری ترکیب دو فنجان از رابطه‌ای عجیب پیروی می‌کند؛ به این صورت که اگر فنجان iاُم با فنجان jاُم مخلوط شود، با خوردن معجون حاصل فرد پس از $f(i, j)$ دقیقه به خواب می‌رود. مقدار $f(i, j)$ به صورت زیر محاسبه می‌شود:

$$f(i, j) = (i - j)^2 + (\sum_{k=min(i,j)+1}^{max(i,j)} a_k)^2$$

به او کمک کنید تا دو فنجانی را پیدا کند که زمان مورد نیاز برای به خواب رفتن را کمینه کند.

▼ Tags

Divide and Conquer

▼ راهنمایی

به ازای هر $1 \leq i \leq n$ تعریف کنید:

$$s_i := \sum_{j=1}^{i} a_j$$

حال فرمول $f(x, y)$ را با استفاده از تعریف بالا بازنویسی کرده و به فرم آن توجه کنید.

ورودی

اولین خط ورودی شامل یک عدد طبیعی n است که تعداد معجون‌ها را مشخص می‌کند. خط بعدی شامل n عدد صحیح $a_1, a_2, \ldots, a_n$ است که قدرت خواب‌آوری فنجان‌ها را مشخص می‌کند.

$$2 \leq n \leq 100000$$

$$-10^4 \leq a_i \leq 10^4$$

خروجی

خروجی شامل یک عدد صحیح است که $min_{i \neq j} f(i, j)$ را مشخص می‌کند.

مثال

ورودی نمونه ۱

```
4
2 3 -5 -7
```

خروجی نمونه ۱

```
8
```

داریم:

$$f(3, 1) = (3 - 1)^2 + (3 + (-5))^2 = 8$$

ورودی نمونه ۲

```
3
-1 2 1
```

خروجی نمونه ۲

2

داریم:

$$f(3, 2) = (3 - 2)^2 + (1)^2 = 2$$

سوپر شلدون و کرونا

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

شلدون یک فیزیک‌دان نابغه است که با توجه به شیوع کرونا در کشورش نمی‌تواند به راحتی از خانه خارج شود. او به شدت نگران سلامتی خود و جامعه است. در کشور شلدون n شهر با شماره‌های ۱ تا n وجود دارد. با وجود این که مردم عاقل و فهمیده‌ی کشورش تمام تلاششان را می‌کنند تا بهداشت را رعایت کرده و تا جای ممکن از خانه خارج نشوند ولی بعضی انتقال‌های بین شهری اجتناب‌ناپذیر است. شلدون به تازگی موفق به ساختن دستگاه‌های انتقال شده‌است که در یک چشم بر هم زدن محموله را از شهری به شهر دیگر منتقل می‌کنند. هر دستگاه انتقال مخصوص انتقال دادن اجسام از یک شهر مشخص به شهری دیگر است. و نمی‌توان از دستگاه انتقال از شهر x به شهر y برای فرستادن اجسام از شهر y به شهر x استفاده کرد. هم‌چنین این دستگاه‌ها به گونه‌ای ساخته شده‌اند که اگر دستگاهی برای انتقال از شهر x به شهر y وجود داشته باشد و همچنین دستگاهی برای انتقال از شهر y به شهر z وجود داشته باشد می‌توان فورا اجسام را از شهر x به شهر z منتقل کرد.

شلدون برای کمک به شرایط بحرانی کشور پیش‌نهاد ساختن این دستگاه‌ها را به دولت می‌دهد. ساختن هر دستگاه هزینه‌ی زیادی می‌برد به همین دلیل دولت m جفت شهر که انتقال اجسام بین آن‌ها ضروری است را مشخص می‌کند. شلدون قصد دارد طوری دستگاه‌ها را بسازد که به ازای هر جفت شهر مهم (a_i, b_i) حداقل یک راه رفتن از شهر a_i به شهر b_i وجود داشته باشد. (لزوماً این راه مستقیماً a_i را به b_i وصل نمی‌کند.)

هر چند شلدون یک نابغه است و به کمک شما نیازی ندارد اما وقت فکر کردن به سوال‌های سطحی را ندارد و از شما می‌خواهد که به او بگویید کم‌ترین تعداد دستگاه‌هایی که باید ساخته شود تا دولت راضی باشد چه‌قدر است.

▼ Tags

Topological Sort - SCC

▼ راه‌نمایی

با توجه به تگ سوال سعی کنید به سوال بیشتر فکر کنید!

▼ راه‌نمایی ۲

به گراف تولید شده با راس‌های شهرها و یال‌های جفت‌شهرهای مهم نگاه کنید؛ هر مولفه‌ی همبندی را سعی کنید با کمترین تعداد یال متصل کنید. به دور داشتن یا نداشتن مولفه‌های همبندی دقت کنید.

ورودی

خط اول ورودی شامل دو عدد صحیح n و m است که تعداد شهرها و تعداد جفت شهرهای مهم را مشخص می‌کند.

$$2 \leq n \leq 10^5,\ 1 \leq m \leq 10^5$$

در m خط بعدی جفت شهرهای مهم ورودی داده می‌شوند. در iاُمین خط $(1 \leq i \leq m)$ دو عدد صحیح a_i و b_i ورودی داده می‌شوند که نشان می‌دهد باید راهی برای رفتن از شهر a_i به شهر b_i به وسیله‌ی دستگاه‌های انتقال وجود داشته باشد.

$$1 \leq a_i, b_i \leq n,\ a_i \neq b_i$$

تضمین می‌شود که همه‌ی این جفت‌ها مجزا هستند.

خروجی

در تنها خط خروجی کم‌ترین تعداد دستگاه‌هایی که باید ساخته شود تا تمامی خواسته‌های دولت برآورده شود را چاپ کنید.

مثال

ورودی نمونه ۱

```
4 5
1 2
1 3
1 4
2 3
2 4
```

خروجی نمونه ۱

```
3
```

ورودی نمونه ۲

```
4 6
1 2
1 4
2 3
2 4
3 2
3 4
```

خروجی نمونه ۲

```
4
```

در صورت وجود پرسش از «مهراد زمانی» یا «غزل خلیقی‌نژاد» بپرسید.

گردش در اُز

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

جادوگر شهر اُز که از هنرنمایی در مجالس و جشن‌ها خسته شده‌است (قانون دست و پایش را برای انجام کارهای بزرگ‌تر و خطرناک‌تر بسته‌است.)، تصمیم گرفته‌است که برای تعطیلات سال نو، یک تور گردش‌گری در شهر اُز را برنامه‌ریزی کند و از این راه پول خوبی به جیب بزند. در شهر اُز n جاذبه‌ی گردش‌گری وجود دارد؛ اما با توجه به محدود بودن زمان، جادوگر می‌خواهد برنامه‌ی بازدید از m جاذبه‌ی گردش‌گری متمایز را در تور بگنجاند. او به خوبی می‌داند که هر چه تعداد عکس‌هایی که گردش‌گران در جریان بازدیدهایشان می‌گیرند بیش‌تر باشند، تعداد بیش‌تری از آن عکس‌ها را در شبکه‌های اجتماعی منتشر خواهند کرد و بدین ترتیب در تعطیلات بعدی، تعداد مشتریان تور گردش‌گری او بیش‌تر خواهد شد. بنا بر این می‌خواهد برنامه‌ی بازدیدها را طوری بچیند که تعداد عکس‌هایی که گرفته می‌شود بیشینه شود. جادوگر برای این که بفهمد گردش‌گران چند عکس خواهند گرفت، از قدرت‌های جادویی‌اش در پیش‌بینی آینده استفاده می‌کند. او می‌داند که به ازای هر $1 \leq i \leq n$، اگر گردش‌گران از جاذبه‌ی iاُم بازدید کنند، a_i عکس یادگاری در آن مکان خواهند گرفت. چندین سال اجرای برنامه‌های سرگرم‌کننده در نقاط مختلف شهر باعث شده‌است که جادوگر به نقشه‌ی شهر کاملا مسلط شده باشد و k مسیر یک‌طرفه با چشم‌اندازهای زیبا را بشناسد. مسیر iاُم ($1 \leq i \leq k$) از جاذبه‌ی x_iاُم آغاز و به جاذبه‌ی y_iاُم ختم می‌شود. او می‌داند که اگر برنامه‌ی بازدید از هر دو جاذبه‌ی x_iاُم و y_iاُم را در تور بگنجاند و بازدید از جاذبه‌ی y_iاُم بلافاصله پس از بازدید از جاذبه‌ی x_iاُم انجام شود (تا اتوبوس گردش‌گران از مسیر زیبای iاُم عبور کند)، گردش‌گران در طول مسیر c_i عکس یادگاری خواهند گرفت. دقت کنید که اگر گردش‌گران بلافاصله پس از بازدید از جاذبه‌ی xاُم ($1 \leq x \leq n$)، از جاذبه‌ی yاُم ($1 \leq y \leq n$) بازدید کنند ($x \neq y$) و جاده‌ی زیبایی از جاذبه‌ی xاُم به جاذبه‌ی yاُم وجود نداشته باشد، در طول مسیر هیچ عکسی نمی‌گیرند.

از آن جایی که تعداد برنامه‌های مختلف برای تور می‌تواند بسیار زیاد باشد و جادوگر از پس حساب کردن تعداد عکس‌ها در هر کدام از حالت‌ها برنمی‌آید، از شما کمک می‌خواهد که بیشینه‌ی تعداد عکس‌هایی که می‌تواند در طول تور گرفته شود را به او بگویید. در ازای آن، او با قدرت‌های جادویی خود نمره‌ی شما را در درس آنالیز الگوریتم‌ها افزایش می‌دهد!

▾ Tags

DP - Bitmasks

▼ راه‌نمایی

تابع $f(mask, i)$ را به این صورت تعریف کنید: بیشینه‌ی تعداد عکس‌هایی که می‌تواند در طول تور گرفته شود به شرطی که بیت‌های $mask$ نشان‌دهنده‌ی شماره‌ی جاذبه‌هایی باشد که در طول تور از آن‌ها بازدید می‌شود و آخرین بازدید از جاذبه‌ی شماره‌ی i انجام شود.

پاسخ مسئله بر حسب مقادیر تابع f به چه صورت بیان می‌شود؟ مقادیر تابع f را به صورت بازگشتی محاسبه کنید.

ورودی

در خط نخست ورودی سه عدد n و m و k به ترتیب و با فاصله آمده‌اند.

$$1 \leq m \leq n \leq 18$$

$$0 \leq k \leq n(n-1)$$

در خط دوم، اعداد a_1 تا a_m به ترتیب و با فاصله آمده‌اند. به ازای هر $1 \leq i \leq n$ داریم:

$$0 \leq a_i \leq 10^9$$

به ازای هر $3 \leq j \leq k+2$ در خط jاُم اعداد x_j و y_i و c_j به ترتیب و با فاصله آمده‌اند. تضمین می‌شود که $x_j \neq y_j$ و به ازای هر $1 \leq k, l \leq n$ حداکثر یک مسیر زیبا وجود دارد که از جاذبه‌ی kاُم آغاز و به جاذبه‌ی lاُم ختم شود. داریم:

$$1 \leq x_j, y_j \leq n$$

$$0 \leq c_j \leq 10^9$$

خروجی

در تنها خط خروجی بیشینه‌ی تعداد عکس‌هایی که گردش‌گران در طول تور خواهند گرفت را چاپ کنید.

مثال

ورودی نمونه ۱

```
2 2 1
1 2
2 1 2
```

خروجی نمونه ۱

```
5
```

اگر گردش‌گران ابتدا از جاذبه‌ی دوم و سپس از جاذبه‌ی نخست بازدید کنند، در مجموع ۵ عکس یادگاری خواهند گرفت.

ورودی نمونه ۲

```
4 3 3
3 1 1 4
3 1 2
3 2 1
1 4 3
```

خروجی نمونه ۲

```
13
```

اگر گردش‌گران به ترتیب از جاذبه‌های سوم، نخست و چهارم بازدید کنند، در مجموع ۱۳ عکس یادگاری خواهند گرفت.

شام موش خرمای کوهی

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

قبل از این که زمستان فرا برسد، موش‌های خرمای کوهی به فکر جمع کردن آذوقه برای زمستان می‌افتند و چون نسلشان در خطر انقراض است آذوقه‌ی خیلی خیلی زیادی انبار می‌کنند. آذوقه‌شان شامل گل‌های سفید تک‌شاخه‌ای و دسته‌هایی kتایی از گل‌های قرمز می‌شود. موشی باهوش تصمیم می‌گیرد برای این که حوصله‌اش در این دوران سر نرود، تعداد روش‌هایی را پیدا کند که می‌تواند حداقل a و حداکثر b گل را بخورد (یعنی تعداد گل‌هایی که می‌خورد در بازه‌ی $[a, b]$ باشد.). دقت کنید که اگر موش بخواهد از گل‌های قرمز بخورد باید کل دسته‌ی kتایی آن را بخورد. به موش کمک کنید!

▾ Tags

DP - Partial Sum

▾ راه‌نمایی

تابع $f(x)$ را تعداد روش‌های خوردن دقیقا x گل تعریف کنید. سپس

$$s(x) = \sum_{i=0}^{x} f(i)$$

تعریف کنید و با کمک $s(b_i)$ و $s(a_i - 1)$ مسئله را حل کنید.

همچنین در مورد باقی‌مانده بر $10^9 + 7$ دقت کنید که عمل باقی‌مانده گرفتن روی جمع و ضرب پخش می‌شود و باقی‌مانده عددی مثبت است.

ورودی

حال، برنامه‌ای بنویسید که ابتدا دو عدد طبیعی t و k را دریافت کند که t تعداد تست‌کیس‌ها و k تعداد گل‌های قرمز در یک دسته است.

$$1 \leq t, k \leq 10^5$$

در t خط بعدی دو عدد طبیعی a_i و b_i آمده‌اند که مشخص می‌کنند که در تست‌کیس iام، تعداد گل‌هایی که موش می‌خورد باید در چه بازه‌ای باشد.

$$1 \leq a_i, b_i \leq 10^5$$

خروجی

خروجی برنامه‌ی شما باید شامل t خط باشد که در خط iام باید باقی‌مانده تعداد روش‌های خوردن حداقل a_i و حداکثر b_i گل بر $10^9 + 7$ را چاپ کنید.

مثال

ورودی نمونه

```
3 2
1 3
2 3
4 4
```

خروجی نمونه

```
6
5
5
```

برای $k = 2$ و ۱ گل، یک روش امکان‌پذیر است: (سفید).

دقت کنید که چون دسته‌های گل قرمز دوتایی است، نمی‌تواند ۱ گل قرمز بخورد.

برای $k = 2$ و ۲ گل دو روش وجود دارد: (قرمز-قرمز) و (سفید-سفید).

برای $k = 2$ و ۳ گل سه روش وجود دارد: (سفید-سفید-سفید) و (سفید-قرمز-قرمز) و (قرمز-قرمز-سفید).

دقت کنید که برای $k = 2$ و ۴ گل، موش می‌تواند به روش‌های (قرمز-قرمز-قرمز-قرمز) و (سفید-قرمز-قرمز-سفید) و (قرمز-قرمز-سفید-سفید) گل بخورد ولی به روش‌های (قرمز-سفید-قرمز-قرمز) و (قرمز-سفید-سفید-

قرمز) نمی‌تواند.

بارون درخت‌نشین

- محدودیت زمان: ۲ ثانیه برای سی و سی‌پلاس‌پلاس، ۱۰ ثانیه برای جاوا و پایتون
- محدودیت حافظه: ۲۵۶ مگابایت برای سی و سی‌پلاس‌پلاس، ۱۰۲۴ مگابایت برای جاوا و پایتون

بارون درخت‌نشین برای سال نو می‌خواهد درخت‌تکانی کند. او صاحب q درخت است. درختانی که بارون روی شاخه‌های آن‌ها می‌نشیند حالا دیگر پیر شده‌اند و به همین خاطر بارون می‌خواهد آن‌ها را با آویختن آویزهای رنگارنگ تزئین کند. او به ازای هر $1 \leq i \leq q$، تصویر درخت iاُم را به فرم یک گراف همبند با n_i رأس و $n_i - 1$ یال روی کاغذ می‌کشد. n_i رأس این گراف، n_i نقطه از درخت iاُم را نشان می‌دهند. بارون تصمیم می‌گیرد که از هر کدام از این n_i نقطه دقیقا k_i آویز رنگی آویزان کند. او دوست ندارد که از یک رنگ در کل درخت iاُم بیش‌تر از دو بار استفاده کند؛ زیرا تمایل دارد که درختانش رنگارنگ باشند. زیبایی درخت‌ها در نظر مهمان‌ها برای بارون بسیار مهم است؛ بنا بر این او به ازای هر $1 \leq j \leq n_i - 1$، به یال jاُم گراف iاُم عدد $w_{i,j}$ را نسبت می‌دهد که نشان‌دهنده‌ی میزان در دیدرس بودن آن قسمت از درخت iاُم است. به ازای هر حالت از آویختن آویزها که شرایط مورد نظر بارون را دارد، بارون به این صورت به زیبایی درخت iاُم یک عدد نسبت می‌دهد: او شماره‌ی هر یالی که دو سر آن حداقل یک آویز با رنگ یکسان دارند را روی کاغذ می‌نویسد و مجموعه‌ی این شماره‌ها را S می‌نامد. سپس میزان زیبایی درخت iاُم را با این فرمول محاسبه می‌کند:

$$\sum_{j \in S} w_{i,j}$$

بارون می‌خواهد بیشینه‌ی زیبایی ممکن برای هر کدام از درخت‌هایش را به دست بیاورد. او برای این کار از شما کمک می‌خواهد. در ازای آن، اگر در تعطیلات عید به دیدارش رفتید، به شما عیدی می‌دهد!

▼ Tags

DP - Trees

▼ راه‌نمایی

برای هر کدام از q درخت مسئله را به این صورت حل کنید: ابتدا درخت را از یک رأس دل‌خواه ریشه‌دار کنید. سپس تابع $f(v)$ را به این صورت تعریف کنید: بیشینه‌ی زیبایی زیردرخت رأس v به شرطی که از همه‌ی رئوس آن زیردرخت دقیقا k آویز آویزان کنیم.

پاسخ مسئله بر حسب مقادیر تابع f به چه صورت بیان می‌شود؟ آیا می‌توان مقادیر تابع f را به صورت

بازگشتی از روی دیگر مقادیر این تابع محاسبه کرد؟ اگر خیر، با تعریف یک یا چند تابع کمکی، مقادیر همگی این توابع را به صورت بازگشتی از روی مقادیر دیگر این توابع محاسبه کنید.

ورودی

در خط نخست عدد q آمده‌است.

$$1 \leq q \leq 500000$$

در خطوط بعدی اطلاعات مربوط به درخت‌ها به ترتیب آورده شده‌است. به ازای هر $1 \leq i \leq q$، اطلاعات درخت iاُم به صورت زیر آورده شده‌است:

در خط نخست، دو عدد n_i و k_i به ترتیب و با فاصله آمده‌اند.

$$1 \leq n_i,\ k_i \leq 500000$$

سپس به ازای هر $1 \leq j \leq n_i - 1$، در خط jاُم از $n_i - 1$ خط بعدی، سه عدد $u_{i,j}$ و $v_{i,j}$ و $w_{i,j}$ به ترتیب و با فاصله آمده‌اند که بدان معناست که یال jاُم درخت iاُم بین دو رأس $u_{i,j}$ و $v_{i,j}$ است و میزان در دیدرس بودن آن برابر با $w_{i,j}$ است.

$$1 \leq u_{i,j},\ v_{i,j} \leq n_i$$

$$1 \leq w_{i,j} \leq 10^5$$

تضمین می‌شود که یال‌های داده‌شده تشکیل یک درخت n_i-رأسی می‌دهند و هم‌چنین داریم:

$$\sum_{i=1}^{q} n_i \leq 500000$$

خروجی

به ازای هر $1 \leq i \leq q$، در خط iاُم بیشینه‌ی زیبایی ممکن برای درخت iاُم را چاپ کنید.

مثال

ورودی نمونه ۱

```
2
5 1
2 3 3
1 5 5
2 5 3
4 5 6
7 2
3 6 4
4 1 4
1 5 8
5 6 1
7 1 2
2 7 3
```

خروجی نمونه ۱

```
9
20
```

عیدی‌های آقاجون

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

عید نوروز نزدیک است و آقاجون باید به n تا از نوادگانش عیدی بدهد! این n نواده بسیار حسود هستند. آقاجون m جفت از نوادگانش را می‌شناسد که اولی به دومی حسودی می‌کند. به ازای هر $1 \leq i \leq m$، جفت iاُم را با (a_i, b_i) نشان می‌دهیم. از آن‌جایی که آقاجون اصلا دلش نمی‌خواهد نوادگانش به یک‌دیگر حسودی کنند، باید حواسش را حسابی جمع کند تا به هر کدام از آن‌ها مقدار مناسبی عیدی بدهد. به ازای هر $1 \leq i \leq n$، مقدار عیدی نواده‌ی iاُم را p_i می‌نامیم. به ازای هر $1 \leq i \leq m$، تجربه به آقاجون نشان داده‌است که برای این که نواده‌ی a_iاُم به نواده‌ی b_iاُم به خاطر مقدار عیدی‌شان حسودی نکند، باید نامساوی $p_{b_i} - p_{a_i} \leq w_i$ برقرار باشد (w_i می‌تواند منفی باشد.).

آقاجون باید برای گرفتن اسکناس‌های نو هر چه زودتر به بانک مراجعه کند. به او کمک کنید و تشخیص دهید که آیا ممکن است که مقدار عیدی هر نواده را طوری تعیین کنیم که هیچ‌کدام از نوادگان به خاطر مقدار عیدی‌شان به یک‌دیگر حسودی نکنند؟ اگر بله، مقادیر p_1 تا p_n را طوری تعیین کنید که این اتفاق بیفتد.

▼ Tags

Floyd-warshall - bellman-ford

▼ راه‌نمایی

گراف وزن‌داری از نوادگان طراحی کنید. در صورتی که این گراف دور منفی داشت چه می‌شود؟

اگر دور منفی نداشت به ازای هر راس کوتاه‌ترین مسیر از بقیه‌ی راس‌ها به آن را محاسبه کنید و برای پاسخ به جواب از آن استفاده کنید.

ورودی

در خط نخست دو عدد n و m به ترتیب و با فاصله آمده‌اند.

$$2 \leq n \leq 500$$

$$1 \leq m \leq min(n(n-1),\ 10^5)$$

به ازای هر $1 \leq i \leq m$، در خط iاُم از m خط بعدی، اعداد a_i و b_i و w_i به ترتیب و با فاصله آمده‌اند.

$$1 \leq a_i,\ b_i \leq n$$

$$a_i \neq b_i$$

$$-10^5 \leq w_i \leq 10^5$$

تضمین می‌شود که زوج‌های مرتب $(a_i,\ b_i)$ متمایز هستند.

خروجی

اگر هیچ شیوه‌ای برای تعیین مقادیر p_1 تا p_n وجود ندارد که نوادگان به یک‌دیگر حسودی نکنند، در تنها خط خروجی NO را چاپ کنید.

در غیر این صورت، در نخستین خط خروجی YES را چاپ کنید و در دومین خط خروجی مقادیری که برای p_1 تا p_n تعیین کرده‌اید را به ترتیب و با فاصله چاپ کنید. به ازای هر $1 \leq i \leq n$ شرط زیر باید برقرار باشد:

$$0 \leq p_i \leq 2 \times 10^9$$

تضمین می‌شود که اگر جواب «بله» باشد، می‌توان مقادیر p_1 تا p_n را طوری تعیین کرد که به ازای هر $1 \leq i \leq n$ شرط مذکور برقرار باشد.

در صورتی که چند جواب معتبر وجود دارد، یکی را به دل‌خواه چاپ کنید.

مثال

ورودی نمونه ۱

```
4 3
1 2 12
2 3 -7
4 2 3
```

خروجی نمونه ۱

```
YES
7 7 0 7
```

ورودی نمونه ۲

```
3 3
1 2 -1
2 3 -1
3 1 -1
```

خروجی نمونه ۲

```
NO
```

ماز جومانجی

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۵۱۲ مگابایت

شما در بازی جومانجی گیر افتاده‌اید و تنها راهی که می‌توانید به دنیای واقعی برگردید این است که به دستورات بازی عمل کنید وگرنه تا ابد در این بازی ویدیویی گرفتار خواهید شد.

در مرحله‌ای، شما در مازی به شکل مستطیلی $n \times m$ محبوس شده‌اید و تنها می‌توانید به راست یا پایین حرکت کنید. خانه‌ی jاُم ردیف iاُم را با (i, j) نشان می‌دهیم. اگر در خانه‌ی (x, y) قرار داشته باشید، هر حرکت به راست (در صورت امکان) شما را به خانه‌ی $(x, y + 1)$ می‌برد و هر حرکت به سمت پایین (در صورت امکان) شما را به خانه‌ی $(x + 1, y)$ می‌برد.

هم‌چنین در بعضی از خانه‌های ماز سنگی غلتان وجود دارد. در صورتی که شما به راست حرکت کنید در حالی که در خانه‌ی مقصد سنگی وجود داشته باشد، آن سنگ یک خانه به راست قل می‌خورد و اگر به سنگی دیگر برخورد کند، آن را یک خانه به سمت راست حرکت می‌دهد و این زنجیره از برهم‌کنش‌ها ادامه پیدا می‌کند. در صورتی که به پایین حرکت کنید و در خانه‌ی مقصد سنگی وجود داشته باشد، آن سنگ یک خانه به پایین قل می‌خورد و اگر به سنگی دیگر برخورد کند، آن را یک خانه به سمت پایین حرکت می‌دهد و این زنجیره از برهم‌کنش‌ها ادامه پیدا می‌کند.

دور تا دور این ماز توسط دیوارهای بتونی محاصره شده و نه شما می‌توانید از این دیوارها عبور کنید و نه هیچ سنگی می‌تواند از این دیوارها عبور کند.

در ابتدا شما در خانه‌ی $(1, 1)$ قرار دارید (در این خانه سنگی وجود ندارد.) و می‌دانید که در خانه‌ی (n, m) دری به بیرون وجود دارد (در این خانه ممکن است سنگ وجود داشته باشد.).

شما چون ذهن متفکری دارید تنها به فکر فرار از ماز نیستید بلکه می‌خواهید تعداد تمام راه‌های ممکن برای فرار را نیز به دست بیاورید. دو روش فرار متمایز محسوب می‌شوند اگر یک خانه از ماز وجود داشته باشد که در یک روش وارد آن می‌شوید ولی در روش دیگر خیر.

▼ راهنمایی

به هر خانه‌ی ماز متغیرهایی نسبت دهید. این متغیرها باید در نهایت تعداد کل راه‌های متمایز را از خانه مبدا

به مقصد مشخص کنند. مثلا می‌توانید متغیری نگه دارید که تعداد سنگ‌های زیر خانه را در ابتدا مشخص کند. (یعنی تعداد خانه‌هایی که سنگ دارند و x آن ها مشابه با x خانه است.)

شما می‌خواهید تعداد راه‌ها را از خانه (x, y) به مقصد بدست آورید در صورتی که اولین حرکتتان به پایین باشد. می‌توانید به خانه $(x + 1, y)$ بروید و از آن تمام راه‌های به مقصد را با اولین حرکت به راست بیابید، یا به خانه $(x + 2, y)$ بروید و از آن تمام راه‌های به مقصد را با اولین حرکت به راست بیابید و یا .... اما دقت کنید که تا جایی می‌توانید ادامه دهید که سنگ‌ها را بتوانید به پایین هل دهید. در نهایت جمع تعداد این راه‌ها، تعداد راه‌ها را از خانه (x, y) به مقصد مشخص می‌کند در صورتی که اولین حرکت به پایین باشد.

ورودی

خط اول ورودی شامل دو عدد طبیعی n و m است که ابعاد ماز را مشخص می‌کنند.

$$1 \leq n, m \leq 2000$$

در n خط بعدی که هر خط شامل m کاراکتر است، جاهایی که سنگ وجود دارد مشخص می‌شود. اگر jاُمین کاراکتر از خط iاُم برابر با «R» باشد یعنی در آن خانه سنگ وجود دارد و در صورتی که برابر با «.» باشد یعنی آن خانه خالی است.

خانه‌ی $(1, 1)$ همیشه خالی است.

خروجی

در تنها خط خروجی باقی‌مانده تعداد راه‌های متمایز برای فرار بر $10^9 + 7$ را چاپ کنید.

مثال

ورودی نمونه ۱

```
1 1
.
```

خروجی نمونه ۱

```
1
```

ورودی نمونه ۲

```
2 3
...
..R
```

خروجی نمونه ۲

```
0
```

ورودی نمونه ۳

```
4 4
...R
.RR.
.RR.
R...
```

خروجی نمونه ۳

```
4
```

راه‌های متمایز برای خروج عبارت هستند از:

۱) راست - راست - پایین - راست - پایین - پایین

۲) راست - پایین - راست - پایین - راست - پایین

۳) پایین - راست - پایین - راست - پایین - راست

۴) پایین - پایین - راست - پایین - راست - راست

مبحث این تمرین «کوتاه‌ترین مسیر» و «بیشینه‌ی جریان» است، در صورت وجود مشکل می‌توانید با «مهراد زمانی» یا «علیرضا توفیقی» مطرح کنید.

آزِریلا و روزنامه

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

آزِریلا که حوصله‌اش سر رفته بود و به خاطر قرنطینه نمی‌توانست از خانه بیرون برود، تصمیم گرفت به سراغ روزنامه‌های قدیمی داخل انباری خانه‌اش برود تا یک سرگرمی در آن‌ها پیدا کند و خودش را با آن مشغول کند.

در صفحه‌ی سرگرمی یکی از روزنامه‌ها او به یک بازی جالب برخورد که به این صورت است: یک جدول با n سطر و m ستون داریم که تعدادی از خانه‌های آن به رنگ سیاه و بقیه به رنگ سفید هستند. سطرها از بالا به پایین با اعداد ۱ تا n و ستون‌ها از چپ به راست با اعداد ۱ تا m شماره‌گذاری شده‌اند. به ازای هر $1 \le i \le n$ و هر $1 \le j \le m$، خانه‌ی jاُم سطر iاُم را با (i, j) نشان می‌دهیم. هدف این است که رنگ خانه‌ها را طوری تغییر دهیم که بتوان از خانه‌ی $(1, 1)$ با طی کردن یک مسیر به خانه‌ی (n, m) رسید به طوری که سه شرط زیر برقرار باشند:

- هیچ‌گاه از جدول خارج نشویم.
- در هر حرکت، از خانه‌ای که روی آن قرار داریم به خانه‌ی پایین آن (در صورت وجود) یا خانه‌ی سمت راست آن (در صورت وجود) رفته باشیم.
- در هیچ لحظه‌ای (از جمله لحظه‌ی آغازین و لحظه‌ی پایانی) در یک خانه‌ی سیاه‌رنگ قرار نداشته باشیم.

برای تغییر دادن رنگ خانه‌ها، در هر گام می‌توانیم یک زیرمستطیل دل‌خواه از جدول را انتخاب و رنگ خانه‌هایش را برعکس کنیم. به عبارت دیگر، در هر گام دو عدد $1 \le r_0 \le r_1 \le n$ و دو عدد $1 \le c_0 \le c_1 \le m$ را به دل‌خواه انتخاب و به ازای هر $r_0 \le i \le r_1$ و هر $c_0 \le j \le c_1$، رنگ خانه‌ی (i, j) را برعکس می‌کنیم.

آزِریلا نگاهی به جدول انداخت و متوجه شد که ابعاد آن بیش از حد بزرگ است. به همین دلیل دلش می‌خواهد با کم‌ترین تعداد گام به هدف دست پیدا کند. او برای به دست آوردن این عدد از شما کمک می‌خواهد.

▾ Tags

Shortest Paths

▼ راه‌نمایی

به ازای هر مسیر از $(1, 1)$ به (n, m) که در حین پیمودن آن هیچ‌گاه به سمت بالا یا چپ حرکت نمی‌کنیم، و هر زیرمستطیل دل‌خواه، در حین پیمودن مسیر مذکور حداکثر یک بار وارد زیرمستطیل می‌شویم؛ این یعنی هر زیرمستطیل که بخشی از مسیر را می‌پوشاند، تعدادی از خانه‌های متوالی آن را می‌پوشاند.

هم‌چنین به ازای هر بازه (تعدادی از خانه‌های متوالی) از خانه‌های مسیر مذکور، زیرمستطیلی وجود دارد که دقیقا آن بخش از مسیر را بپوشاند. (چرا؟)

ورودی

در نخستین خط، اعداد n و m به ترتیب و با فاصله آمده‌اند.

$$2 \leq n, m \leq 100$$

سپس در هر کدام از n خط بعد، یک رشته‌ی m-حرفی آمده‌است که وضعیت خانه‌های هر سطر از جدول را نشان می‌دهد. به ازای هر $1 \leq i \leq n$ و هر $1 \leq j \leq m$، اگر حرف jاُم رشته‌ی iاُم برابر با `#` باشد نشان‌دهنده‌ی آن است که خانه‌ی (i, j) به رنگ سیاه است. در غیر این صورت این حرف برابر با `.` است که یعنی رنگ خانه‌ی (i, j) سفید است.

خروجی

در تنها خط خروجی، کمینه‌ی تعداد گام‌های لازم برای رسیدن به هدف بازی را چاپ کنید.

مثال

ورودی نمونه ۱

```
3 3
.##
.#.
##.
```

خروجی نمونه ۱

```
1
```

ورودی نمونه ۲

```
2 2
#.
.#
```

خروجی نمونه ۲

```
2
```

ورودی نمونه ۳

```
4 4
..##
#...
###.
###.
```

خروجی نمونه ۳

```
0
```

گلدان‌های استاد دینْبَلی

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۵۱۲ مگابایت

استاد دینْبَلی، هندسه‌دان برجسته‌ی جهان، به تازگی متوجه شده‌است که گلدان‌هایش به اندازه‌ی کافی خانه‌اش را زیبا نمی‌کنند. او n^2 گلدان در خانه‌اش دارد که آن‌ها را در قالب n ردیف n-تایی در کنار هم چیده‌است، به طوری که به ازای هر $1 \leq i \leq n$، گلدان‌های iام هر ردیف در یک ستون قرار دارند. جنس هر گلدان پلاستیکی یا سفالی‌ست و در هر گلدان یک بوته‌ی گل قرار دارد که نوع آن n حالت مختلف دارد. پس از بررسی‌های فراوان، او فهمید که دلیل این که گلدان‌ها خانه را به اندازه‌ی کافی زیبا نمی‌کنند استفاده از مقدار نادرست عدد π در محاسباتش است. او بار دیگر محاسبات را با مقدار واقعی عدد π انجام داد و به این نتیجه رسید که برای رسیدن به حداکثر زیبایی ممکن، هیچ دو گلدان متمایز هم‌سطر یا هم‌ستونی نباید وجود داشته باشند که هم جنسشان یکسان باشد و هم نوع بوته‌ی گلی که در آن‌ها قرار دارد یکسان باشد.

بنا بر این او می‌خواهد تعدادی از گلدان‌هایش را انتخاب کند و آن‌ها را به همراه بوته‌ی گل داخلشان بفروشد و وقتی همه‌ی آن‌ها را فروخت، به همان تعداد گلدان گل جدید بخرد (که جنس گلدانشان پلاستیکی یا سفالی باشد و نوع بوته‌ی گلشان هم یکی از همان n نوع باشد) و گلدان‌های جدید را به جای گلدان‌های فروخته‌شده قرار دهد تا شرایط مورد نظرش برقرار شوند. توجه کنید که او از هر کدام از $2n$ نوع گلدانِ گل مختلفی که وجود دارد به هر تعدادی که بخواهد می‌تواند بخرد. هم‌چنین دقت کنید که او نمی‌تواند گلدان‌هایی را که برای فروش انتخاب نکرده‌است جابه‌جا کند.

با توجه به شرایط موجود، استاد دینْبَلی تمایل دارد که تا حد امکان در هزینه‌ها صرفه‌جویی کند؛ به همین دلیل می‌خواهد کم‌ترین تعداد گلدان گل ممکن را بخرد. او که مشغول تقسیم زاویه به هفت قسمت مساوی با استفاده از راست‌کش و پرگار است، به دست آوردن این عدد را بر عهده‌ی شما می‌گذارد!

▼ Tags

Flows - Maximum Matching

▼ راه‌نمایی

اگر تعدادی گلدان را در تعدادی از خانه‌های یک جدول n در n طوری بچینیم که در هر خانه حداکثر یک گلدان قرار بگیرد و هیچ دو گلدان متمایزی وجود نداشته باشد که جنس و نوع بوته‌ی گلشان یکسان باشد و

همچنین هم‌سطر یا هم‌ستون باشند، همواره می‌توان در هر کدام از خانه‌های خالی یک گلدان قرار داد به طوری که در نهایت هم‌چنان هیچ دو گلدان متمایزی وجود نداشته باشد که جنس و نوع بوته‌ی گلشان یکسان باشد و هم‌چنین هم‌سطر یا هم‌ستون باشند. (چرا؟) پس کافی‌ست کمینه‌ی تعداد گلدان‌هایی را پیدا کنیم که با حذف کردنشان شرط مذکور برقرار شود.

ورودی

در خط نخست عدد n آمده‌است.

$$2 \leq n \leq 300$$

سپس در هر کدام از n خط بعدی، n عدد با فاصله آمده‌اند که نشان‌دهنده‌ی وضعیت گلدان‌های هر ردیف از n ردیف هستند. به ازای هر $1 \leq i, j \leq n$، عدد jاُم iاُمین خط را $a_{i,j}$ می‌نامیم. علامت $a_{i,j}$ نوع گلدان jاُم ردیف iاُم و $|a_{i,j}|$ نوع بوته‌ی گل این گلدان را مشخص می‌کند.

$$-n \leq a_{i,j} \leq n, \quad a_{i,j} \neq 0$$

خروجی

در تنها خط خروجی کمینه‌ی تعداد گلدان‌هایی را چاپ کنید که استاد دینْبَلی باید بخرد تا شرایط مورد نظرش برقرار شوند. این عدد می‌تواند برابر با صفر هم باشد که به این معناست که محاسبات نادرست استاد تصادفا نتایج مناسبی داشته‌اند!

مثال

ورودی نمونه ۱

```
2
1 2
2 1
```

خروجی نمونه ۱

```
0
```

ورودی نمونه ۲

```
2
1 1
2 1
```

خروجی نمونه ۲

```
1
```

ورودی نمونه ۳

```
2
1 2
1 2
```

خروجی نمونه ۳

```
2
```

ورودی نمونه ۴

```
2
2 2
-2 2
```

خروجی نمونه ۴

```
1
```

تمرین گیتار

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۵۱۲ مگابایت

سِپرمَن که دوست دارد در آینده بتواند قطعات گیتار گیتاریست محبوبش یعنی «جان پتروخانی» را بی‌نقص بنوازد، تصمیم گرفته‌است که از فرصتی که قرنطینه‌ی خانگی در اختیارش گذاشته بیش‌ترین استفاده را کرده و در این روزها از آغاز صبح تا پایان شب نواختن قطعات او را تمرین کند!

برای استفاده‌ی هر چه بهتر از زمان، او تصمیم می‌گیرد که برای تمریناتش برنامه‌ریزی کند. او n تا از بهترین قطعات جان پتروخانی را برای تمرین انتخاب می‌کند و به ازای هر $1 \leq i \leq n$، تصمیم می‌گیرد که قطعه‌ی iاُم را قبل از فرا رسیدن روز a_iاُم و پس از فرا رسیدن روز b_iاُم تمرین **نکند**. او هم‌چنین نمی‌خواهد در یک روز بیش‌تر از یک قطعه را تمرین کند زیرا در غیر این صورت در شکل‌گیری حافظه‌ی عضلانی اختلال ایجاد می‌شود. به دلیل علاقه‌ی بی‌حد و اندازه‌ی او به قطعات جان پتروخانی، سِپرمَن می‌خواهد در نواختن همه‌ی n قطعه‌ای که انتخاب کرده‌است به یک اندازه ماهر شود. این یعنی او باید تعداد یکسانی از روزها را صرف تمرین کردن هر کدام از قطعات کند. هم‌چنین او با این که روزهایی که صرف تمرین یک قطعه‌ی خاص می‌شوند متوالی نباشند مشکلی ندارد.

به سِپرمَن کمک کنید و برنامه‌ای برایش بچینید که شرایط بالا را داشته باشد و تعداد روزهایی را که او مشغول تمرین یک قطعه است بیشینه کند.

▼ Tags

Flows - Binary Search

▼ راه‌نمایی

اگر سِپرمَن بتواند برنامه‌ای بریزد که در آن روی هر قطعه $k > 0$ روز وقت می‌گذارد، می‌تواند برنامه‌ای بریزد که در آن روی هر قطعه $k - 1$ روز وقت می‌گذارد. (چرا؟) پس برای به دست آوردن بیشینه‌ی تعداد روزهایی که می‌تواند هر کدام از قطعات را به آن تعداد از روزها تمرین کند می‌توان از جست‌وجوی دودویی (Binary Search) استفاده کرد.

حال کافی‌ست بتوانیم تشخیص دهیم که برای یک عدد صحیح نامنفی مانند k، آیا سِپرمَن می‌تواند برنامه‌ای

بریزد که در آن روی هر قطعه دقیقا k روز وقت بگذارد یا خیر. سعی کنید این مسئله را به یک مسئله‌ی جریان بیشینه مدل کرده و با استفاده از الگوریتم‌هایی که آموخته‌اید آن را حل کنید.

ورودی

در خط نخست عدد n آمده‌است.

$$1 \leq n \leq 200$$

سپس به ازای هر $1 \leq i \leq n$، در خط iام از n خط بعد، دو عدد a_i و b_i به ترتیب و با فاصله آمده‌اند.

$$0 \leq a_i < b_i \leq 1000$$

خروجی

در تنها خط خروجی بیشینه‌ی تعداد روزهایی که سِپرمَن می‌تواند مشغول تمرین باشد را چاپ کنید.

مثال

ورودی نمونه ۱

```
3
2 4
1 5
6 9
```

خروجی نمونه ۱

```
6
```

سِپرمَن می‌تواند در روزهای دوم و سوم قطعه‌ی نخست، در روزهای نخست و چهارم قطعه‌ی دوم، و در روزهای ششم و هفتم قطعه‌ی سوم را تمرین کند.

ورودی نمونه ۲

```
3
1 2
1 2
1 2
```

خروجی نمونه ۲

```
0
```

علی‌داعشی و بمب‌گذاری

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

امروز روز ۴۰اُم قرنطینه است و علی‌داعشی کاملا از ماندن در خانه خسته شده‌است. او برای رفع خستگی به کمک دوستانش یک روزنامه به زبان انگلیسی آماده کرده و چند نسخه از آن را جلوی در خانه‌ی مردم قرار داده‌است. اما مردم که از بیرون آمدن از خانه واهمه دارند، هیچ‌کدام روزنامه‌ی زیبا و انگلیسی علی‌داعشی را نخوانده‌اند. علی تصمیم گرفته کاری کند که مردم از خانه‌هایشان بیرون بیایند و روزنامه را بردارند تا سرانه‌ی مطالعه‌ی کشور نیز بالا برود. برای این کار او تصمیم می‌گیرد که خیابان‌های شهر را بمباران کند تا سر و صدایی ایجاد شده و مردم برای این که بفهمند چه خبر شده‌است از خانه بیرون بیایند و از قضا روزنامه‌ی علی‌داعشی را نیز مطالعه کنند.

شهر به شکل یک گراف همبند n رأسی و m یالی بدون جهت و وزن‌دار است که خیابان‌ها یال‌های این گراف هستند.

برای یک گراف وزن‌دار و همبند $G = (V, E)$ و رأس $v \in V$، گراف کوتاه‌ترین مسیر برای v را زیرگراف $G' = (V, E')$ تعریف می‌کنیم که $E' \subseteq E$ و G' یک درخت باشد و برای هر $u \in V$ وزن کم‌وزن‌ترین مسیر از v به u در G و G' برابر باشد.

خانه‌ی علی‌داعشی رأس v است و علی‌داعشی به تازگی درباره‌ی مزایای زیرگراف کوتاه‌ترین مسیر چیزهایی شنیده‌است. پس تصمیم می‌گیرد که یکی از زیرگراف‌های کوتاه‌ترین مسیر از رأس v را بمب‌گذاری کند، اما چون بمب گران است، می‌خواهد زیرگرافی را بمب‌گذاری کند که مجموع وزن یال‌های آن کمینه باشد. به علی‌داعشی کمک کنید!

▾ Tags

Dijkstra

▾ راه‌نمایی

الگوریتم دایکسترا را اجرا کنید، فقط سعی کنید در حالت مسیرها با وزن مساوی به یک راس، مسیری را انتخاب کنید که یال آخر آن وزن کمتری دارد. چرا این کار مسئله را حل می‌کند؟

ورودی

در نخستین خط، اعداد n و m به ترتیب و با فاصله آمده‌اند.

$$1 \le n \le 100\,000, \quad 0 \le m \le \min(100\,000, \frac{n(n-1)}{2})$$

سپس در هر کدام از m خط بعد سه عدد v_i و u_i و w_i آمده‌اند که نشان‌دهنده‌ی یال وصل‌کننده‌ی v_i و u_i با وزن w_i هستند.

$$1 \le w_i \le 10^9, \quad 1 \le v_i \ne u_i \le n$$

آخرین خط نیز شامل v است.

$$1 \le v \le n$$

تضمین می‌شود که گراف همبند است و هیچ یالی بیش‌تر از یک بار نیامده‌است.

خروجی

در خط اول خروجی مجموع وزن یال‌های کم‌وزن‌ترین گراف کوتاه‌ترین مسیر برای v را چاپ کنید.

در خط دوم خروجی $n-1$ عدد چاپ کنید که شماره‌ی یال‌هایی که این زیرگراف را می‌سازند باشد که با فاصله آن‌ها را از هم جدا کرده‌اید.

مثال

ورودی نمونه ۱

```
3 3
1 2 1
1 3 2
2 3 1
3
```

خروجی نمونه ۱

```
2
1 3
```

ورودی نمونه ۲

```
4 4
2 3 1
1 2 1
3 4 1
4 1 2
4
```

خروجی نمونه ۲

```
4
1 3 4
```

چرزه بعد از قرنطینه

- محدودیت زمان: ۲ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

چرزه که ایام کرونا به شدت به او فشار آورده‌است، بی‌خیال قرنطینه می‌شود و از خانه پا به فرار می‌گذارد تا به منزل دوستانش سر بزند.

در شهر محل سکونت چرزه، n خانه داریم که با شماره‌های ۱ تا n مشخص شده‌اند. خانه‌ها با m خیابان به هم وصل شده‌اند که خیابان iاُم خانه‌ی v_i را به خانه‌ی u_i به صورت مستقیم و یک‌طرفه وصل کرده‌است و l_i کیلومتر مسافت دارد، هم‌چنین خیابان iاُم متعلق به محله‌ی c_iاُم شهرداری است.

چرزه که یک‌نواختی قرنطینه کاملا حوصله‌اش را سر برده، تصمیم می‌گیرد از خانه بیرون زده و به منزل دوستانش برود. برای چرزه یک دنباله از خانه‌های $h_1, h_2, \cdots, h_k$ یک مسیر غیریک‌نواخت است، اگر اولا برای هر i که $1 \leq i < k$، بین h_i و h_{i+1} خیابانی وجود داشته باشد و دوما اگر $i + 1 < k$ محله‌ی خیابانی که h_i را به h_{i+1} متصل می‌کند با محله‌ی خیابانی که h_{i+1} را به h_{i+2} متصل می‌کند متفاوت باشد.

هم‌چنین او طول یک مسیر غیریک‌نواخت را مجموع طول خیابان‌های آن در نظر می‌گیرد.

چون چرزه حال ندارد، می‌خواهد از بین همه‌ی مسیرهای غیریک‌نواخت از h_1 به h_k آنی را انتخاب کند که کم‌ترین طول را دارد.

چرزه در خانه‌ی s قرنطینه‌ی خانگی‌ست و q پرسش برایش مطرح می‌شود. پرسش iاُم شامل یک t_i است که یعنی او می‌خواهد طول کوتاه‌ترین مسیر غیریک‌نواخت از s به t_i را بداند. جواب پرسش‌های او را بدهید!

▼ Tags

Dijkstra

▼ راه‌نمایی

اگر برای هر راس و هر رنگ یک راس در گراف جدیدی در نظر بگیرید، مسئله قابل حل است ولی در این حالت با محدودیت زمان روبرو می‌شوید؛ به این نکته دقت کنید که برای هر راس تنها نگه‌داشتن دو رنگ با کم‌وزن‌ترین مسیر کافی است.

ورودی

در نخستین خط، اعداد n و m و C و s به ترتیب و با فاصله آمده‌اند که به ترتیب نشان‌دهنده‌ی تعداد خانه‌ها، تعداد خیابان‌ها، تعداد محله‌های شهر و شماره‌ی خانه‌ی چرزه هستند.

$$1 \leq n, m, C \leq 100\,000$$

$$1 \leq m \leq n(n-1)$$

سپس در هر کدام از m خط بعد، اطلاعات خیابان iاُم یعنی v_i و u_i و l_i و c_i به ترتیب و با فاصله از هم آمده‌اند.

$$1 \leq v_i \neq u_i \leq n$$

$$1 \leq c_i \leq C$$

$$1 \leq l_i \leq 1000\,000\,000$$

تضمین می‌شود که یال دیگری از رأس v_i به رأس u_i وجود ندارد.

در خط بعد q آمده‌است که تعداد پرسش‌های چرزه است.

$$1 \leq q \leq 100\,000$$

در خط بعد q عدد آمده‌است که عدد iاُم برابر با t_i است. این اعداد با فاصله از هم جدا شده‌اند.

خروجی

در تنها خط خروجی q عدد که با فاصله از هم جدا شده‌اند را چاپ کنید که عدد iاُم برابر با پاسخ پرسش iاُم چرزه باشد. اگر مسیری بین s و t_i وجود نداشت، -1 چاپ کنید.

مثال

ورودی نمونه ۱

```
5 4 3 1
1 2 10 1
2 3 10 2
3 4 10 2
4 5 10 1
5
1 2 3 4 5
```

خروجی نمونه ۱

```
0 10 20 -1 -1
```

ورودی نمونه ۲

```
5 5 2 1
1 2 10 1
2 3 10 2
3 4 10 1
4 5 10 2
1 5 39 1
5
1 2 3 4 5
```

خروجی نمونه ۲

```
0 10 20 30 39
```

بخش نهم

آزمون‌ها

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

آزمون میان‌ترم زمان: ۳/۵ ساعت

نکته ۱: برای مسئله‌هایی که با برنامه‌ریزی پویا حل می‌کنید لازم است زیرمسئله‌هایی که تعریف می‌کنید را به‌طور کامل و شفاف به فارسی توصیف کنید.

نکته ۲: در مسائلی که زمان اجرای الگوریتم خواسته شده مشخص شده، با ارائه الگوریتمی با زمان اجرای اندکی بدتر ممکن است بخشی از نمره را بگیرید.

۱. فرض کنید $2n$ عدد $a_1^1, a_2^1, \ldots, a_n^1$ و $a_1^0, a_2^0, \ldots, a_n^0$ داده شده باشند. می‌خواهیم یک زیرمجموعه $S = \{x_1, \ldots, x_k\} \subseteq \{1, 2, \ldots, n\}$ و رشته $b \in \{0, 1\}^k$ را بیابیم به‌طوری که برای هر i، b_i و b_{i+1} متفاوت باشند و $\sum_{i=1}^{k} a_{x_i}^{b_i}$ بیشینه شود. الگوریتمی با زمان اجرای $O(n)$ برای این مسئله ارائه دهید.

۲. در یک درخت ریشه‌دار، منظور از یک مسیر *پایین‌رونده* مسیری مانند $v_1, v_2, \ldots, v_{k+1}$ است به‌طوری‌که برای هر $1 \leq i \leq k$، v_i پدر v_{i+1} باشد. همچنین منظور از دو مسیر مجزا، دو مسیر است که در هیچ رأسی با هم مشترک نباشند.

الگوریتمی کارآ ارائه دهید که درخت ریشه‌دار T و عدد k را ورودی بگیرد و بیشینه تعداد مسیرهای مجزای پایین‌رونده به طول k در T را محاسبه کند.

۳. فرض کنید $G = (V, E)$ یک گراف بدون جهت باشد و $s \in V$. یک یال $e \in E$ را *بی‌فایده* می‌گوییم اگر با حذف e طول کوتاهترین مسیر تا هر رأس $v \in V$ یا تغییر نکند یا حداکثر یکی زیاد شود. الگوریتمی با زمان اجرای $O(E)$ برای یافتن همه یال‌های بی‌فایده ارائه کنید.

۴. فرض کنید گراف $G = (V, E)$ رأسی به‌صورت ضمنی به ما داده شده باشد. مجموعه رأس‌های V برابر است با کل رشته‌های صفر و یک به‌طول n. همچنین یک تابع isConnected در اختیار داریم که می‌توانیم آن را هر دو رأس دلخواه از G مثل u و v فراخوانی کنیم و تابع به ما جواب می‌دهد که آیا u به v وصل است یا خیر. همچنین فرض کنید دو رأس $s, t \in V$ هم به ما داده شده باشند. الگوریتمی با حافظه چندجمله‌ای طراحی کنید که تشخیص دهد آیا از s به t مسیر وجود دارد یا خیر.

راهنمایی: حداکثر فاصله s و t چقدر است؟ تابعی بازگشتی بنویسید که عمق درخت بازگشتش حداکثر n باشد. توجه کنید زمان الگوریتمتان لازم نیست چندجمله‌ای باشد.

۵. فرض کنید f یک جریان از s به t با اندازه α در شبکه G باشد. شبکه G' را به‌این‌گونه از روی G می‌سازیم که رأس‌ها و یال‌های G' همان رأس‌ها و یال‌های G هستند و ظرفیت و کران پایین برای هر یال e، به ترتیب $\lceil f(e) \rceil$ و $\lfloor f(e) \rfloor$ است. نشان دهید جریان معتبر f' در G' با اندازه $\lceil \alpha \rceil$ وجود دارد.

۶. گراف $G = (V, E)$ با تابع ظرفیت $c : E \to Q_+$ روی یال‌های آن را در نظر بگیرید. فرض کنید H یک گراف کامل روی V باشد. برای هر یال xy از H، $w(xy)$ را برابر با ظرفیت یک xyـ برش کمینه در G (برشی که x در یک سمت آن است و y در سمت دیگر) تعریف کنید.

(آ) ثابت کنید برای هر $x, y, z \in V$، $w(xy) \geq \min\{w(xz), w(zy)\}$.

(ب) فرض کنید T یک زیردرخت فراگیر از H با وزن بیشینه باشد. نشان دهید برای هر $x, y \in V$، $w(xy)$ برابر است با وزن کوچکترین یال در مسیر بین x و y در T.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

[بهار ۹۹]

پاسخ آزمون میان‌ترم — زمان: ۳/۵ ساعت

نکته ۱: برای مسئله‌هایی که با برنامه‌ریزی پویا حل می‌کنید لازم است زیرمسئله‌هایی که تعریف می‌کنید را به‌طور کامل و شفاف به فارسی توصیف کنید.

نکته ۲: در مسائلی که زمان اجرای الگوریتم خواسته شده مشخص شده، با ارائه الگوریتمی با زمان اجرای اندکی بدتر ممکن است بخشی از نمره را بگیرید.

۱. فرض کنید $2n$ عدد $a_1^1, a_2^1, \ldots, a_n^1$ و $a_1^0, a_2^0, \ldots, a_n^0$ داده شده باشند. می‌خواهیم یک زیرمجموعه $S = \{x_1, \ldots, x_k\} \subseteq \{1, 2, \ldots, n\}$ و رشته $b \in \{0,1\}^k$ را بیابیم به‌طوری که برای هر i، b_i و b_{i+1} متفاوت باشند و $\sum_{i=1}^{k} a_{x_i}^{b_i}$ بیشینه شود. الگوریتمی با زمان اجرای $O(n)$ برای این مسئله ارائه دهید.

پاسخ: زیرمسئله $D(i,t)$ که $1 \le i \le n$، و $t \in \{0,1\}$ را برابر با جواب بهینه مسئله در حالتی که بخواهیم یک زیرمجموعه از بین i تا a^0ها و a^1های اول انتخاب کنیم و آخرین عضو انتخاب شده برابر با یک a_j^t باشد (که $j \le i$) تعریف می‌کنیم. داریم

$$D(i,t) = \max\{D(i-1,t), D(i-1,1-t) + a_i^t\}.$$

۲. در یک درخت ریشه‌دار، منظور از یک مسیر *پایین‌رونده* مسیری مانند $v_1, v_2, \ldots, v_{k+1}$ است به‌طوری‌که برای هر $1 \le i \le k$، v_{i+1} پدر v_i باشد. همچنین منظور از دو مسیر مجزا، دو مسیر است که در هیچ رأسی با هم مشترک نباشند.

الگوریتمی کارآ ارائه دهید که درخت ریشه‌دار T و عدد k را ورودی بگیرد و بیشینه تعداد مسیرهای مجزای پایین‌رونده به طول k در T را محاسبه کند.

پاسخ: زیر مسئله $C(v,j)$ که $1 \le j \le k+1$ را برابر با جواب بهینه برای زیردرخت به ریشه v در حالتی که رأس jام از یک مسیر پایین‌رونده باشد در نظر بگیرید. جواب مسئله اصلی $C(r,1)$ است که r ریشه T است (توجه کنید می‌توانیم فرض کنیم در جواب بهینه یکی از مسیرها ریشه را شامل است). داریم

$$C(v,i) = \begin{cases} 1 + \sum_{u \in v.children} C(u,1) & i = k+1 \\ \max_{x \in v.children} \{ \sum_{\substack{u \in v.children \\ u \neq x}} C(u,1) + C(x,i+1) \} & i \le k \end{cases}$$

با یک‌بار محاسبه $\sum_{u \in v.children} C(u,1)$ می‌توان $C(v,i)$ را در زمان $O(|v.children|)$ محاسبه کرد و بنابراین یک الگوریتم $O(nk)$ برای مسئله داریم.

۳. فرض کنید $G = (V,E)$ یک گراف بدون جهت باشد و $s \in V$. یک یال $e \in E$ را *بی‌فایده* می‌گوییم اگر با حذف e طول کوتاهترین مسیر تا هر رأس $v \in V$ یا تغییر نکند یا حداکثر یکی زیاد شود. الگوریتمی با زمان اجرای $O(E)$ برای یافتن همه یال‌های بی‌فایده ارائه کنید.

پاسخ: با یک بار اجرای BFS از رأس s، $d(u)$ و $p(u)$ یعنی فاصله s تا رأس u و پدر u در درخت BFS را به ازای هر u می‌یابیم. واضح است که هر یال که به صورت $(u, p(u))$ به ازای یک u نباشد بی‌فایده است. همچنین در صورتی که u به رأس $v \neq P(u)$ که $d(v) \le d(u)$ وصل باشد، یال $(u, P(u))$ هم بی‌فایده است؛ وگرنه نیست.

۴. فرض کنید گراف $G = (V, E)$ رأسی به‌صورت ضمنی به ما داده شده باشد. مجموعه رأس‌های V برابر است با کل رشته‌های صفر و یک به‌طول n. همچنین یک تابع isConnected در اختیار داریم که می‌توانیم آن را هر دو رأس دلخواه از G مثل u و v فراخوانی کنیم و تابع به ما جواب می‌دهد که آیا u به v وصل است یا خیر. همچنین فرض کنید دو رأس $s, t \in V$ هم به ما داده شده باشند. الگوریتمی با حافظه چندجمله‌ای طراحی کنید که تشخیص دهد آیا از s به t مسیر وجود دارد یا خیر.

راهنمایی: حداکثر فاصله s و t چقدر است؟ تابعی بازگشتی بنویسید که عمق درخت بازگشتش حداکثر n باشد. توجه کنید زمان الگوریتمتان لازم نیست چندجمله‌ای باشد.

پاسخ: $\textsc{ExistsPath}(s, t, n)$ را فراخوانی می‌کنیم.

```
ExistsPath(u, v, k)
1  // returns True iff there is a path of length at most 2^k from u to v
2  if k == 0
3      return isConnected(u, v)
4  for each x ∈ {0,1}^n
5      if ExistsPath(u, x, k − 1) and ExistsPath(x, v, k − 1)
6          return True
7  return False
```

۵. فرض کنید f یک جریان از s به t با اندازه α در شبکه G باشد. شبکه G' را به‌این‌گونه از روی G می‌سازیم که رأس‌ها و یال‌های G' همان رأس‌ها و یال‌های G هستند و ظرفیت و کران پایین برای هر یال e، به ترتیب $\lceil f(e) \rceil$ و $\lfloor f(e) \rfloor$ است. نشان دهید جریان معتبر f' در G' با اندازه $\lceil \alpha \rceil$ وجود دارد.

پاسخ: اگر یال ts با جریان α به شبکه اضافه کنیم، یک گردش داریم. پس اگر ظرفیت و کران پایین هر یال e را برابر با $f(e)$ بگذاریم، یک گردش معتبر داریم. در نتیجه این ظرفیت‌ها و کران‌پایین‌ها شرط قضیه هافمن را دارند. بنابراین اگر ظرفیت‌ها و کران‌پایین‌ها را طبق صورت سوال تعریف کنیم، همچنان شرط هافمن را داریم. حال اگر ظرفیت و کران پایین ts را از α به $\lceil \alpha \rceil$ تغییر دهیم، چون تغییر اتفاق افتاده کمتر از ۱ است و بقیه ظرفیت‌ها و کران‌پایین‌ها صحیح هستند، شرط هافمن همچنان برقرار است.

توجه کنید که قضیه جریان صحیح که در درس داشتیم یا در مورد جریان بیشینه در حالتی است که کران پایین نداشته باشیم یا در مورد وجود گردش در شرایطی که کران پایین داشته باشیم، و نمی‌توان حکم سؤال را از قضیه جریان صحیح نتیجه گرفت.

۶. گراف $G = (V, E)$ با تابع ظرفیت $c : E \to Q_+$ روی یال‌های آن را در نظر بگیرید. فرض کنید H یک گراف کامل روی V باشد. برای هر یال xy از H، $w(xy)$ را برابر با ظرفیت یک xyـبرش کمینه در G (برشی که x در یک سمت آن است و y در سمت دیگر) تعریف کنید.

(آ) ثابت کنید برای هر $x, y, z \in V$، $w(xy) \geq \min\{w(xz), w(zy)\}$.

(ب) فرض کنید T یک زیردرخت فراگیر از H با وزن بیشینه باشد. نشان دهید برای هر $x, y \in V$، $w(xy)$ برابر است با وزن کوچکترین یال در مسیر بین x و y در T.

پاسخ:

(آ) یک xyـبرش کمینه (A, B) را در نظر بگیرید. اگر $z \in A$، این برش یک zyـبرش هم هست و بنابراین $w(zy) \leq w(xy)$. به‌طور مشابه اگر $z \in B$، داریم $w(xz) \leq w(xy)$. پس در هر حال $w(xy) \geq \min\{w(xz), w(zy)\}$.

(ب) با استقرا روی طول مسیر می‌توان ثابت کرد که برای مسیر P از x به y، $w(xy) \geq \min_{e \in P} w(e)$. در صورتی که $P \subseteq T$، و $w(xy) > \min_{e \in P} w(e)$، می‌توان با اضافه کردن xy به T و حذف کوچکترین یال P درخت بزرگتری به دست آورد.

آنالیز الگوریتم‌ها (۲۲۸۹۱)

[بهار ۹۹]

آزمون پایان‌ترم زمان: ۴ ساعت

---

۱. در مسئله **جریان چندتایی** ورودی یک شبکه $G = (V, E)$ و تابع ظرفیت مثبت c روی E به همراه k رأس مبدأ $s_1, \ldots, s_k$ و k رأس مقصد $t_1, \ldots, t_k$ است. هدف این است که k جریان $f_1, \ldots, f_k$ بیابیم به‌طوری که f_i یک جریان معتبر از s_i به t_i باشد، برای هر یال از گراف جمع کل k جریان عبوری از آن یال بیشتر از ظرفیتش نشود، و جمع کل k جریان بیشینه شود.

نشان دهید مسئله **جریان چندتایی** در زمان چندجمله‌ای قابل حل است.

۲. فرض کنید یک پارچه مستطیلی به ابعاد $X \times Y$ دارید که X و Y اعداد صحیح هستند. همچنین لیستی از n محصول که می‌توانند از پارچه درست شوند در اختیار دارید. محصول iام پارچه‌ای به ابعاد $a_i \times b_i$ مصرف می‌کند و قیمت فروش آن c_i است. ماشینی در اختیار داریم که هر پارچه مستطیلی را می‌تواند افقی یا عمودی ببرد تا دو تکه مستطیل کوچکتر درست شود. می‌خواهیم پارچه‌ای که در اختیار داریم را به گونه‌ای ببریم که با تبدیل تکه‌های ایجاد شده به محصول‌های مختلف، بیشترین سود را ببریم. توجه کنید که می‌توانیم از یک محصول به هر تعداد که دوست داریم تولید کنیم.

(آ) ثابت کنید این مسئله ان‌پی‌ـ تمام است.

(ب) یک الگوریتم شبه‌چندجمله‌ای برای این مسئله ارائه دهید.

۳. فرض کنید گراف جهت‌دار بدون وزن $G = (V, E)$ داده شده باشد. هر یال G ممکن است یکی از چهار رنگ آبی، قرمز، سبز و زرد را داشته باشد یا بی‌رنگ باشد. همچنین دقیقاً یک رأس از G با هرکدام از این چهار رنگ وجود دارد (بقیه $|V| - ۴$ رأس گراف بی‌رنگ هستند). یک زیرمجموعه $U \subseteq V$ از رأس‌ها داده شده است که روی هرکدام یک بستنی قرار دارد. یک رأس شروع r هم داده شده است. هدف این است که از r شروع کنیم، حداقل یک بستنی برداریم و نهایتاً به r برگردیم و تعداد یال‌هایی که طی می‌کنیم کمینه باشد. این قانون وجود دارد که اگر بخواهیم از یک یال رنگی عبور کنیم، پیش از آن حتماً باید از رأسی که رنگش مانند آن یال است عبور کرده باشیم.

الگوریتمی خطی برای حل این مسئله ارائه دهید.

راهنمایی: ابتدا مسئله را برای حالت ساده‌تری که یال‌ها رنگ ندارند حل کنید. همچنین در صورت نوشتن راه‌حل این حالت خاص بخشی از نمره را می‌گیرید.

۴. برنامه صحیح زیر را در نظر بگیرید.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} c_e x_e + \sum_{v \in V} p_v y_v & \\
\text{s.t} \quad & \sum_{e \in \delta(S)} x_e \geq y_v & \forall S \subseteq V - r, S \neq \emptyset,\ \forall v \in S \\
& y_r = ۱ & \\
& y_v \in \{۰, ۱\} & \forall v \in V \\
& x_e \in \{۰, ۱\} & \forall e \in E
\end{aligned}
$$

منظور از $\delta(S)$ یال‌هایی است که دقیقاً یک سرشان در S است.

(آ) یک مسئله الگوریتمی روی گراف‌ها تعریف کنید که توسط این برنامه مدل شود.

(ب) این برنامه صحیح را به یک برنامه خطی ریلکس کنید و دوگان آن را بنویسید.

۵. در مسئله **رنگ‌آمیزی نزدیک** ورودی یک گراف $G = (V, E)$ است و سؤال این است که آیا می‌توان رأس‌های G را با رنگ‌های $\{۰, ۱, ۲, ۳, ۴\}$ به گونه‌ای رنگ‌آمیزی کرد که اگر $uv \in E$ آنگاه $c(v) = c(u) + ۱ \pmod ۵$ یا $c(v) = c(u) - ۱ \pmod ۵$، که $c(v)$ رنگ رأس v است.

نشان دهید مسئله **رنگ‌آمیزی نزدیک** ان‌پی‌ـ‌تمام است.

۶. در مسئله **چهارتکه کردن گراف**، یک گراف $G = (V, E)$ با تابع وزن $w : E \to \mathbb{R}_+$ به عنوان ورودی داده می‌شود. هدف این است که V را به چهار مجموعه افراز کنیم به‌طوری که جمع وزن یال‌هایی که بین این چهار بخش هستند بیشینه باشد.

الگوریتمی چندجمله‌ای ارائه دهید که برای هر ورودی این مسئله جوابی پیدا کند که جمع وزن یال‌هایش حداقل ۰٫۷۴ جواب بهینه باشد.

۷. شبکه $G = (V, E)$ با تابع ظرفیت مثبت c روی یال‌های آن را در نظر بگیرید. همچنین فرض کنید $s, t \in V$. در این سؤال منظور از یک جریان، یک stـ‌جریان معتبر است.

(آ) نشان دهید هر جریان $f \in \mathbb{R}^E$ در G را می‌توان به صورت

$$f = \sum_{p \in \mathcal{P}} \lambda_p \chi_p + \sum_{c \in \mathcal{C}} \lambda_c \chi_c \qquad (۱)$$

نوشت که همه λ_iها نامنفی هستند. در این‌جا $\mathcal{P}$ مجموعه همه مسیرهای s به t در G، و $\mathcal{C}$ مجموعه همه دورهای G است. همچنین $\chi_p \in \mathbb{R}^E$ بردار مشخصه p است؛ یعنی $\chi_p(e) = ۱$ اگر و فقط اگر $e \in p$ و در غیر این صورت $\chi_p(e) = ۰$.

(ب) با استفاده از بخش قبل، مسئله جریان بیشینه را به‌صورت یک برنامه خطی مدل کنید.

(ج) نشان دهید جریان بیشینه‌ای مانند f وجود دارد که در فرمول‌بندی آن به صورت بخش الف، فقط تعداد چندجمله‌ای تا از λ_i ها غیرصفر هستند.

(د) نشان دهید می‌توان به گونه‌ای به هر یال $e \in E$ وزن w_e را نسبت داد که طول (وزن) هر مسیر از s به t حداقل یک باشد، و به ازای هر جریان بیشینه f، اگر f را به‌صورت ۱ بنویسیم، اگر p مسیری از s به t باشد که کوتاهترین مسیر نیست، آنگاه $\lambda_p = ۰$.

(ه) چگونه می‌توان با استفاده از فرمول‌بندی ارائه شده در این سؤال، مقدار جریان بیشینه را در زمان چندجمله‌ای حساب کرد؟

آنالیز الگوریتم‌ها (۲۲۸۹۱)

[بهار ۹۹]

پاسخ آزمون پایان‌ترم

---

۱. در مسئله **جریان چندتایی** ورودی یک شبکه $G = (V, E)$ و تابع ظرفیت مثبت c روی E به همراه k رأس مبدأ $s_1, \ldots, s_k$ و k رأس مقصد $t_1, \ldots, t_k$ است. هدف این است که k جریان $f_1, \ldots, f_k$ بیابیم به‌طوری که f_i یک جریان معتبر از s_i به t_i باشد، برای هر یال از گراف جمع کل k جریان عبوری از آن یال بیشتر از ظرفیتش نشود، و جمع کل k جریان بیشینه شود.

نشان دهید مسئله **جریان چندتایی** در زمان چندجمله‌ای قابل حل است.

پاسخ: مشابه مسئله جریان بیشینه، این مسئله را می‌توان به صورت یک برنامه خطی مدل کرد. ابتدا برای هر $1 \leq i \leq k$ یک یال با ظرفیت بینهایت از t_i به s_i اضافه می‌کنیم.

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{k} f_i(t_i s_i) \\
\text{s.t.} \quad & \sum_{i=1}^{k} f_i(e) \leq c(e) && \forall e \in E \\
& \sum_{vu \in E} f_i(vu) - \sum_{uv \in E} f_i(uv) = 0 && \forall u \in V,\ 1 \leq i \leq k \\
& f_i(e) \geq 0 && \forall 1 \leq i \leq k,\ e \in E
\end{aligned}
$$

برای درک شهودی‌تر این مسئله می‌توانید تصور کنید که مثلاً می‌خواهیم مقداری آب از s_1 به t_1، مقداری آب هویج از s_2 به t_2، مقداری دلستر از s_3 به t_3 و ... بفرستیم، و باید قانون بقای هر مایع در هر گره (به‌جز مبدأها و مقصدها) را داشته باشیم. روش‌های معمول مدل‌سازی یا استفاده از الگوریتم‌های مسئله جریان بیشینه استاندارد باعث می‌شوند این توازن به هم بخورد (مایع‌های مختلف باهم قاطی شوند) و بنابراین کار نمی‌کنند.

۲. فرض کنید یک پارچه مستطیلی به ابعاد $X \times Y$ دارید که X و Y اعداد صحیح هستند. همچنین لیستی از n محصول که می‌توانند از پارچه درست شوند در اختیار دارید. محصول iام پارچه‌ای به ابعاد $a_i \times b_i$ مصرف می‌کند و قیمت فروش آن c_i است. ماشینی در اختیار داریم که هر پارچه مستطیلی را می‌تواند افقی یا عمودی ببرد تا دو تکه مستطیل کوچکتر درست شود. می‌خواهیم پارچه‌ای که در اختیار داریم را به گونه‌ای ببریم که با تبدیل تکه‌های ایجاد شده به محصول‌های مختلف، بیشترین سود را ببریم. توجه کنید که می‌توانیم از یک محصول به هر تعداد که دوست داریم تولید کنیم.

(آ) ثابت کنید این مسئله انپی‌ـتمام است.

(ب) یک الگوریتم شبه‌چندجمله‌ای برای این مسئله ارائه دهید.

پاسخ:

(آ) مسئله SubsetSum را به نسخه تصمیم گیری این مسئله تحویل می‌کنیم.

فرض کنید $S = \{a_1, \ldots, a_n\}, t$ یک نمونه از مسئله SubsetSum باشد که a_iها اعداد صحیح مثبت هستند. تعریف می‌کنیم $M = 1 + \sum a_i$. متناظر با هر a_i دو نوع محصول تعریف می‌کنیم: محصول نوع اول با ابعاد $(M^{n+1} + M^i) \times 1$ و نوع دوم با ابعاد $(M^{n+1} + M^i + a_i) \times 1$. ارزش هر محصول را نیز مساوی طول آن در نظر می‌گیریم. همچنین ابعاد پارچه را برابر با $(nM^{n+1} + \sum_{i=1}^{n} M^i + t) \times 1$ تعریف می‌کنیم و ارزشی که به دنبالش هستیم را نیز برابر با طول پارچه قرار می‌دهیم. می‌توانید

تحقیق کنید که در صورتی می‌توان به ارزش مورد نظر در بریدن پارچه دست پیدا کرد که بتوان پارچه را به گونه‌ای برید که از هرکدام از دو محصول متناظر با هر $1 \le i \le n$ دقیقاً یکی تولید شود و جمع a_iهای متناظر با محصول‌های تولید شده نوع دوم برابر با t شود.

(ب) از برنامه‌ریزی پویا استفاده می‌کنیم. زیرمسئله $D(x, y)$ را برابر با بیشترین سودی که می‌توان از یک پارچه با ابعاد $x \times y$ به دست آورد تعریف می‌کنیم. داریم:

$$D(x,y) = \max\{0, \max_{1 \le k \le x-1}\{D(k,y) + D(x-k,y)\}, \max_{1 \le k \le y-1}\{D(x,k) + D(x,y-k)\}, \max_{\substack{1 \le i \le n \\ a_i \le x,\ b_i \le y}} c_i\}$$

۳. فرض کنید گراف جهت‌دار بدون وزن $G = (V, E)$ داده شده باشد. هر یال G ممکن است یکی از چهار رنگ آبی، قرمز، سبز و زرد را داشته باشد یا بی‌رنگ باشد. همچنین دقیقاً یک رأس از G با هرکدام از این چهار رنگ وجود دارد (بقیه $|V| - 4$ رأس گراف بی‌رنگ هستند). یک زیرمجموعه $U \subseteq V$ از رأس‌ها داده شده است که روی هرکدام یک بستنی قرار دارد. یک رأس شروع r هم داده شده است. هدف این است که از r شروع کنیم، حداقل یک بستنی برداریم و نهایتاً به r برگردیم و تعداد یال‌هایی که طی می‌کنیم کمینه باشد. این قانون وجود دارد که اگر بخواهیم از یک یال رنگی عبور کنیم، پیش از آن حتماً باید از رأسی که رنگش مانند آن یال است عبور کرده باشیم.

الگوریتمی خطی برای حل این مسئله ارائه دهید.

راهنمایی: ابتدا مسئله را برای حالت ساده‌تری که یال‌ها رنگ ندارند حل کنید. همچنین در صورت نوشتن راه‌حل این حالت خاص بخشی از نمره را می‌گیرید.

پاسخ: برای حالت خاص، یک بار از r BFS می‌زنیم تا برای هر رأس $v \in U$، $d(r, v)$ را پیدا کنیم. همچنین با یک بار دیگر اجرای BFS با مبدأ U، $d(v, r)$ را برای هر $v \in U$ پیدا می‌کنیم (یا یال‌ها را برعکس می‌کنیم و از r BFS می‌زنیم). اگر v رأسی باشد که جمع این دو مقدار برایش کمینه باشد، الحاق کوتاهترین مسیر r به v با کوتاهترین مسیر v به r جواب مسئله است.

برای حالت کلی، فرض کنید می‌دانیم در جواب بهینه قبل از اینکه برای اولین بار به یک رأس از U برسیم، از رأس سبز عبور می‌کنیم و بعد از اینکه برای اولین بار یک رأس از U را دیدیم از رأس زرد هم عبور خواهیم کرد و از رأس‌های آبی و قرمز عبور نمی‌کنیم. در این صورت برای یافتن جواب بهینه کافی است سه کوتاهترین مسیر پیدا کنیم (که دومی مشابه حالت خاص نیاز به ۲ BFS دارد): کوتاهترین مسیری از r به رأس سبز که از یال رنگی استفاده نمی‌کند، کوتاهترین مسیر از رأس سبز به رأس زرد که در میانه مسیر از یک رأس U عبور می‌کند و فقط مجاز است از یال‌های سبز یا بی‌رنگ استفاده کند، و کوتاهترین مسیر از رأس زرد به r که می‌تواند از یال‌های بی‌رنگ، سبز و زرد استفاده کند.

چون رأس‌های رنگی مورد استفاده در جواب بهینه و ترتیب دیده شدن آن‌ها را نمی‌دانیم، همه حالت‌های ممکن را امتحان می‌کنیم و بهترین حالت را انتخاب می‌کنیم. هر حالت مشابه توضیح بالا نیاز به زمان $O(V + E)$ دارد و تعداد کل حالت‌ها هم $O(1)$ است پس کلاً زمان اجرای الگوریتم خطی است.

راه دوم: یک گراف H از روی G می‌سازیم به‌طوری که به ازای هر رأس $v \in V$، ۳۲ کپی از رأس v را در H قرار می‌دهیم. هر کپی به شکل $(v, f_b, f_r, f_g, f_y, f_U)$ است که هر کدام از f_iها صفر یا یک هستند و مشخص می‌کنند که آیا در مسیر رسیدن از کپی اول r یعنی $(r, 0, 0, 0, 0, 0)$ به این کپی از v از هر کدام از رأس‌های رنگی و همچنین از یکی از رأس‌های U عبور کرده‌ایم یا نه. یال‌های H هم با توجه به یال‌های G و قوانین گفته شده در سؤال تعریف می‌شوند. به‌طور دقیق‌تر، فرض کنید $uv \in E$. در این صورت بین رأس‌های $(u, f_b, f_r, f_g, f_y, f_U)$ و $(v, f'_b, f'_r, f'_g, f'_y, f'_U)$ یال می‌گذاریم اگر اولاً یال uv بی‌رنگ باشد یا در صورتی که رنگ این یال x باشد، $f_x = 1$، و ثانیاً همه f'_iها کوچکتر یا مساوی f_i متناظرشان باشند، مگر اینکه رنگ رأس v برابر با x باشد که در این صورت $f'_x = 1$ یا اینکه $v \in U$ که در این‌صورت $f'_U = 1$.

حال کافی است در گراف H، کوتاهترین مسیر از رأس $(r, 0, 0, 0, 0, 0)$ به رأس‌های $(r, f_b, f_r, f_g, f_y, 1)$ را پیدا کنیم و بین این ۱۶ مسیر کوتاهترین را انتخاب کنیم.

۴. برنامه صحیح زیر را در نظر بگیرید.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{e\in E} c_e x_e + \sum_{v\in V} p_v y_v & & \\
\text{s.t} \quad & \sum_{e\in\delta(S)} x_e \geq y_v & & \forall S \subseteq V - r, S \neq \emptyset,\ \forall v \in S \\
& y_r = ۱ & & \\
& y_v \in \{۰, ۱\} & & \forall v \in V \\
& x_e \in \{۰, ۱\} & & \forall e \in E
\end{aligned}
$$

منظور از $\delta(S)$ یال‌هایی است که دقیقاً یک سرشان در S است.

(آ) یک مسئله الگوریتمی روی گراف‌ها تعریف کنید که توسط این برنامه مدل شود.

(ب) این برنامه صحیح را به یک برنامه خطی ریلکس کنید و دوگان آن را بنویسید.

پاسخ:

(آ) می‌خواهیم یک زیردرخت (نه لزوماً فراگیر) از G انتخاب کنیم که حتماً شامل رأس r باشد و جمع هزینه یال‌ها و رأس‌های درخت کمینه شود.

اگر فرض کنیم c_eها مثبت و p_vها منفی هستند، مسئله به نوعی طبیعی‌تر می‌شود. از یک طرف می‌خواهیم تا جای ممکن رأس‌های بیشتری در درختمان باشند (هر رأس مقداری سود به ما می‌رساند)، و از طرف دیگر برای یال‌ها باید هزینه بدهیم.

(ب)

$$
\begin{aligned}
\min \quad & \sum_{e\in E} c_e x_e + \sum_{v\in V} p_v y_v & & \\
\text{s.t} \quad & \sum_{e\in\delta(S)} x_e \geq y_v & & \forall S \subseteq V - r, S \neq \emptyset,\ \forall v \in S \\
& y_r = ۱ & & \\
& ۰ \leq y_v \leq ۱ & & \forall v \in V - r \\
& ۰ \leq x_e \leq ۱ & & \forall e \in E
\end{aligned}
\qquad
\begin{aligned}
\max \quad & \sum_{e\in E} r_e + \sum_{v\in V} t_v & & \\
\text{s.t} \quad & \sum_{\substack{S\subseteq V-r,\\ v\in S, e\in\delta(S)}} Z_{S,v} + r_e \leq c_e & & \forall e \in E \\
& \sum_{\substack{S\subseteq V-r,\\ v\in S}} -Z_{S,v} + t_v \leq p_v & & \forall v \in V \\
& Z_{S,v} \geq ۰ & & \forall S \subseteq V - r, v \in S \\
& r_e \leq ۰ & & \forall e \in E \\
& t_v \leq ۰ & & \forall v \in V - r
\end{aligned}
$$

۵. در مسئله **رنگ‌آمیزی نزدیک** ورودی یک گراف $G = (V, E)$ است و سؤال این است که آیا می‌توان رأس‌های G را با رنگ‌های $\{۰, ۱, ۲, ۳, ۴\}$ به گونه‌ای رنگ‌آمیزی کرد که اگر $uv \in E$ آنگاه $c(v) = c(u) + ۱ \pmod ۵$ یا $c(v) = c(u) - ۱ \pmod ۵$، که $c(v)$ رنگ رأس v است.

نشان دهید مسئله **رنگ‌آمیزی نزدیک** ان‌پی‌ـ تمام است.

پاسخ: از مسئله ۵ـ رنگ‌آمیزی رأسی تحویل کنید. به جای هر یال در گراف ورودی، یک مسیر به طول ۳ بگذارید.

۶. در مسئله **چهارتکه کردن گراف**، یک گراف $G = (V, E)$ با تابع وزن $w : E \to \mathbb{R}_+$ به عنوان ورودی داده می‌شود. هدف این است که V را به چهار مجموعه افراز کنیم به‌طوری که جمع وزن یال‌هایی که بین این چهار بخش هستند بیشینه باشد.

الگوریتمی چندجمله‌ای ارائه دهید که برای هر ورودی این مسئله جوابی پیدا کند که جمع وزن یال‌هایش حداقل ۰٫۷۴ جواب بهینه باشد.

پاسخ: مشابه الگوریتم جستجوی محلی برای برش بیشینه (جلسه ۲۲). توجه کنید که مشابه آن مسئله، برای اثبات چندجمله‌ای بودن زمان اجرای الگوریتم باید از ارتقای قابل توجه استفاده کنید.

۷. شبکه $G = (V, E)$ با تابع ظرفیت مثبت c روی یال‌های آن را در نظر بگیرید. همچنین فرض کنید $s, t \in V$. در این سؤال منظور از یک جریان، یک stـ جریان معتبر است.

(آ) نشان دهید هر جریان $f \in \mathbb{R}^E$ در G را می‌توان به صورت

$$f = \sum_{p \in \mathcal{P}} \lambda_p \chi_p + \sum_{c \in \mathcal{C}} \lambda_c \chi_c \qquad (۱)$$

نوشت که همه λ_iها نامنفی هستند. در این‌جا $\mathcal{P}$ مجموعه همه مسیرهای s به t در G، و $\mathcal{C}$ مجموعه همه دورهای G است. همچنین $\chi_p \in \mathbb{R}^E$ بردار مشخصه p است؛ یعنی $\chi_p(e) = ۱$ اگر و فقط اگر $e \in p$ و در غیر این صورت $\chi_p(e) = ۰$.

(ب) با استفاده از بخش قبل، مسئله جریان بیشینه را به‌صورت یک برنامه خطی مدل کنید.

(ج) نشان دهید جریان بیشینه‌ای مانند f وجود دارد که در فرمول‌بندی آن به صورت بخش الف، فقط تعداد چندجمله‌ای تا از λ_i ها غیرصفر هستند.

(د) نشان دهید می‌توان به گونه‌ای به هر یال $e \in E$ وزن w_e را نسبت داد که طول (وزن) هر مسیر از s به t حداقل یک باشد، و به ازای هر جریان بیشینه f، اگر f را به‌صورت ۱ بنویسیم، اگر p مسیری از s به t باشد که کوتاهترین مسیر نیست، آنگاه $\lambda_p = ۰$.

(ه) چگونه می‌توان با استفاده از فرمول‌بندی ارائه شده در این سؤال، مقدار جریان بیشینه را در زمان چندجمله‌ای حساب کرد؟

پاسخ:

(آ) از استقرا روی تعداد یال‌های با جریان غیرصفر استفاده می‌کنیم. اگر جریان خالص خروجی از s مثبت است، حتماً یک مسیر با جریان مثبت از s به t وجود دارد. λ متناظر با این مسیر را برابر با کمترین جریان یک یال آن در نظر می‌گیریم و این مقدار را از جریان همه یال‌های این مسیر کم می‌کنیم. به‌طور مشابه در صورتی که جریان خالص خروجی از s صفر است اما یالی با جریان مثبت وجود دارد، حتماً یک دور با جریان مثبت وجود دارد.

(ب)

$$\begin{aligned} \max \quad & \sum_{p \in \mathcal{P}} \lambda_p \\ \text{s.t.} \quad & \sum_{p \in \mathcal{P}, e \in p} \lambda_p \le c(e) \quad \forall e \in E \\ & \lambda_p \ge ۰ \qquad\qquad \forall p \in \mathcal{P} \end{aligned}$$

(ج) می‌دانیم یک جواب بهینه پایه‌ای برای LP بخش قبل وجود دارد، و در هر جواب پایه‌ای، حداکثر $|E|$ متغیر ناصفر وجود دارد. همچنین تعداد ضرایب ناصفری که اثبات بخش آ به دست می‌دهد چندجمله‌ای است.

(د) دوگان LP بخش آ را در نظر بگیرید.

$$\begin{aligned} \min \quad & \sum_{e \in E} c(e) w_e \\ \text{s.t.} \quad & \sum_{e \in p} w_e \ge ۱ \quad \forall p \in \mathcal{P} \\ & w_e \ge ۰ \qquad \forall e \in E \end{aligned}$$

فرض کنید w یک جواب بهینه برای دوگان، و λ یک جواب بهینه برای LP اولیه باشد. طبق شرط لنگی مکمل، اگر یک محدودیت از دوگان به‌صورت تساوی برقرار نباشد (یعنی مسیر متناظر طبق وزن‌دهی w کوتاهترین مسیر نباشد)، متغیر متناظر با آن محدودیت (مسیر) در برنامه اولیه باید صفر باشد.

(ه) در کلاس حل تمرین گفتیم که اگر الگوریتمی چندجمله‌ای وجود داشته باشد که یک جواب از یک LP را بگیرد و چک کند که آیا جواب شدنی است یا نه و اگر نیست یکی از محدودیت‌های نقض‌شده را برگرداند، آنگاه می‌توان آن LP را با روش بیضوی حل کرد (حتی اگر تعداد محدودیت‌های LP نمایی باشد). این کار را با برنامه دوگان می‌توان انجام داد؛ کافی است با الگوریتم دایکسترا چک کنیم که آیا مسیری از s به t با وزن کمتر از ۱ طبق وزن‌دهی w وجود دارد یا نه.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

آزمونک اختیاری زمان: ۲ ساعت

---

۱. ثابت کنید تعداد برش‌های کمینه متمایز یک گراف بدون جهت بدون وزن n رأسی همبند حداکثر $\binom{n}{2}$ است.

۲. فرض کنید یک مجموعه $A = \{a_1, \ldots, a_n\}$ داریم که هر $a_i \in [0, 1]$. همچنین می‌دانیم که دقیقاً ۵ تا از a_iها صفر هستند. فرض کنید نمی‌توانیم به‌طور مستقیم مقدار a_iها را ببینیم؛ صرفاً می‌توانیم تعدادی از آن‌ها را انتخاب کنیم و بقیه را از لحاظ مقدار با آن‌ها مقایسه کنیم. می‌خواهیم با ایده نمونه‌گیری تصادفی و به‌روزرسانی وزن‌ها، جای اعداد صفر را پیدا کنیم. این الگوریتم را در نظر بگیرید.

۰ ابتدا به همه a_iها وزن ۱ را نسبت می‌دهیم.

۱ مجموعه S شامل ۲۰ عضو از اعضای A را به‌طور تصادفی انتخاب می‌کنیم. هر عنصر با احتمال متناسب با وزنش انتخاب می‌شود.

۲ پنج عضو کوچکترین بین اعضای S را انتخاب می‌کنیم. مجموعه شامل این ۵ عنصر را H می‌نامیم.

۳ مجموعه V شامل همه عناصری از $A \setminus S$ را می‌یابیم که از همه عناصر H (و در نتیجه از همه عناصر S) کوچکتر یا مساوی هستند.

۴ اگر $V = \emptyset$، H را خروجی می‌دهیم، و الگوریتم خاتمه می‌یابد.

۵ وگرنه، اگر $w(V) \leq \frac{w(A)}{10}$، وزن هرکدام از اعضای V را دوبرابر می‌کنیم (منظور از $w(X)$ جمع وزن اعضای مجموعه X است).

۶ به مرحله ۱ می‌رویم.

فرض کنید در هر بار تکرار الگوریتم، شرط گام پنجم به احتمال حداقل $\frac{1}{2}$ درست باشد. نشان دهید امید تعداد دفعات تکرار الگوریتم $O(\lg n)$ است.

راهنمایی: جمع وزن پنج عدد صفر را با جمع وزن کل اعضای A بعد از $5k$ مرحله مقایسه کنید.

آنالیز الگوریتم‌ها (۲۲۸۹۱)
[بهار ۹۹]

پاسخ آزمونک اختیاری

---

۱. ثابت کنید تعداد برش‌های کمینه متمایز یک گراف بدون جهت بدون وزن n رأسی همبند حداکثر $\binom{n}{2}$ است.

پاسخ: در جلسه اضافه ۳ دیدیم اگر $(S, V \setminus S)$ یک برش کمینه باشد، احتمال اینکه الگوریتم انقباض تصادفی کارگر این برش را پیدا کند حداقل $\binom{n}{2}^{-1}$ است. فرض کنید k برش کمینه متمایز داشته باشیم و X_i متغیر تصادفی‌ای باشد که نشان دهد آیا برش کمینه iام توسط الگوریتم برگردانده می‌شود. همچنین فرض کنید X تعداد برش‌های کمینه‌ای باشد که توسط الگوریتم خروجی داده می‌شود. در این صورت $X = \sum X_i$. همچنین $E[X] = \sum E[X_i] \geq k\binom{n}{2}^{-1}$. اما الگوریتم حداکثر یک برش کمینه خروجی می‌دهد، یعنی $E[X] \leq 1$. بنابراین $k \leq \binom{n}{2}$.

۲. فرض کنید یک مجموعه $A = \{a_1, \ldots, a_n\}$ داریم که هر $a_i \in [0, 1]$. همچنین می‌دانیم که دقیقاً ۵ تا از a_iها صفر هستند. فرض کنید نمی‌توانیم به‌طور مستقیم مقدار a_iها را ببینیم؛ صرفاً می‌توانیم تعدادی از آن‌ها را انتخاب کنیم و بقیه را از لحاظ مقدار با آن‌ها مقایسه کنیم. می‌خواهیم با ایده نمونه‌گیری تصادفی و به‌روزرسانی وزن‌ها، جای اعداد صفر را پیدا کنیم. این الگوریتم را در نظر بگیرید.

۰ ابتدا به همه a_iها وزن ۱ را نسبت می‌دهیم.

۱ مجموعه S شامل ۲۰ عضو از اعضای A را به‌طور تصادفی انتخاب می‌کنیم. هر عنصر با احتمال متناسب با وزنش انتخاب می‌شود.

۲ پنج عضو کوچکترین بین اعضای S را انتخاب می‌کنیم. مجموعه شامل این ۵ عنصر را H می‌نامیم.

۳ مجموعه V شامل همه عناصری از $A \setminus S$ را می‌یابیم که از همه عناصر H (و در نتیجه از همه عناصر S) کوچکتر یا مساوی هستند.

۴ اگر $V = \emptyset$، H را خروجی می‌دهیم، و الگوریتم خاتمه می‌یابد.

۵ وگرنه، اگر $w(V) \leq \frac{w(A)}{10}$، وزن هرکدام از اعضای V را دوبرابر می‌کنیم (منظور از $w(X)$ جمع وزن اعضای مجموعه X است).

۶ به مرحله ۱ می‌رویم.

فرض کنید در هر بار تکرار الگوریتم، شرط گام پنجم به احتمال حداقل $\frac{1}{2}$ درست باشد. نشان دهید امید تعداد دفعات تکرار الگوریتم $O(\lg n)$ است.

راهنمایی: جمع وزن پنج عدد صفر را با جمع وزن کل اعضای A بعد از $5k$ مرحله مقایسه کنید.

پاسخ: امید تعداد دفعات تکرار الگوریتم حداکثر دو برابر تعداد دفعات درست بودن شرط گام پنجم الگوریتم است. بعد از $5k$ بار درست بودن شرط گام پنجم الگوریتم، جمع وزن پنج عنصر صفر حداقل $2^k \cdot 5$ است. همچنین جمع وزن کل عناصر حداکثر $n(1 + \frac{1}{10})^{5k} \leq n(\sqrt{e})^k$ است. از آنجا که $\sqrt{e} < 2$ جمع وزن پنج عنصر صفر به‌طور نمایی سریعتر از جمع وزن کل عناصر رشد می‌کند و اگر بعد از $O(\lg n)$ مرحله الگوریتم تمام نشود، جمع وزن پنج عنصر صفر از وزن کل عناصر بیشتر می‌شود که غیرممکن است.