centro appunti
www.centroappunti.it
Corso Luigi Einaudi, 55 - Torino

Appunti universitari
Tesi di laurea
Cartoleria e cancelleria
Stampa file e fotocopie
Print on demand
Rilegature

NUMERO : 122 DATA : 01/07/2011

A P P U N T I

STUDENTE : Regis

MATERIA : Algoritmi e Programmazione Avanzata

Il presente lavoro nasce dall'impegno dell'autore ed è distribuito in accordo con il Centro Appunti.
Tutti i diritti sono riservati. È vietata qualsiasi riproduzione, copia totale o parziale, dei contenuti inseriti nel presente volume, ivi inclusa la memorizzazione, rielaborazione, diffusione o distribuzione dei contenuti stessi mediante qualunque supporto magnetico o cartaceo, piattaforma tecnologica o rete telematica, senza previa autorizzazione scritta dell'autore.

ATTENZIONE: QUESTI APPUNTI SONO FATTI DA STUDENTI E NON SONO STATI VISIONATI DAL DOCENTE.
IL NOME DEL PROFESSORE, SERVE SOLO PER IDENTIFICARE IL CORSO.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 3 di 69

```
int f();    DICHIARAZIONE PROTOTIPO
int f() {   DEFINIZIONE FUNZIONE
.....
}
```

$\lfloor\ \rfloor$ parte intera approssimata per difetto
$\lceil\ \rceil$ parte intera approssimata per eccesso
} non influiscono sul comp. asintotica

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 3 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 5 di 69

CLASSI di MEMORIZZAZIONE

VARIABILE <classe di memorizzazione> <tipo> <nome>

variabile locale : definita in un blocco

variabile globale : definita fuori da un blocco (nella parte di codice successiva)

SCOPE (ambiente) : porzione di programma in cui la variabile è utilizzabile

locale o globale in base alla classe di memorizzazione

CLASSI di MEMORIZZAZ.

- OGGETTI AUTOMATICI
  - AUTOMATIC
    - classe di default
    - variabile visibile nel blocco → scope locale
    - distrutta al termine del blocco
    - non inizializzata
  - REGISTER
    - il compilatore tenta di memorizzarla in un registro, non molto usate perché il compilatore tenta di farlo autonomamente
    - deve essere specificato "register"
    - ⊖ non ne posso usare l'indirizzo
    - ⊖ pochi registri → rischio di rallentare
- OGGETTI STATICI (esterne)
  - STATIC → LINKAGGIO interno, oggetto unico nel file
    - specificare "static"
    - classe statica riferita al contesto di un blocco ma il valore persiste anche fuori dal blocco di definizione
    - allocazione permanente (occhio a sprechi)
    - scope limitato al file di definizione
    - inizializzate a 0. (a compile-time)
    - nel blocco : per estendere la validità
    - fuori blocco : per limitarla a quel file
  - EXTERN → dichiarazioni riferite all'oggetto unico in tutto il programma
    - specificare "extern"
    - scope globale (è l'unica)
    - memoria allocata altrove
    - inizializzata a 0
    - nel blocco : definizione fuori blocco
    - fuori blocco : definizione in altro file
    - UNA DEFINIZIONE esterna ai blocchi senza "extern" alloca la memoria
    - PIÙ DICHIARAZIONI nei blocchi ("extern") facolt.
    - ⊖ difficoltà nel tenere traccia.
    - ⊖ allocazione permanente → spreco

una variabile globale è automatic. static

PREPROCESSORE : elabora il file sorgente prima di passarlo alla compilazione.

- definizione di MACRO (costanti a nomi simbolici) o MACRO CON ARGOMENTI, simili a chiamate a funzione ⊕ prestazione: espande il codice in linea così che non aumenta la dimensione del programma, ⊕ adattabili a qualunque tipo ⊖ l'assenza di tipi può causare errori difficili da trovare ⊖ problemi quando si passano espressioni: è necessario specificare parentesi.
- inclusione condizionale

```
# if defined (<nome macro>)   eq. # ifdef (nome)        # elif
# if <nome> == <valore>       eq.                       # else
# if ! defined (HEADER)       eq  # ifndef (nome)
# define HEADER               # endif
```

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 5 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 7 di 69

MEMORIA DINAMICA

ALLOCAZIONE STATICA di MEMORIA: tramite dichiarazione di variabili

ALLOCAZIONE DINAMICA: su richiesta, per quantità definite al tempo di esecuzione (assegna un'area della memoria fisicamente separata da quella riservata alle variabili dichiarate staticamente).

SPAZIO di INDIRIZZAMENTO

STACK	→ dati temporanei, ossia variabili locali e cresce verso il basso ↓
HEAP	→ memoria allocata dinamicamente, cresce dal text in su ↑
TEXT	→ codice + dati (funzioni, variabili globali) del programma (funzioni generano un ambiente locale sullo stack LIFO temporaneamente) occupa uno spazio fisso al tempo di esecuzione

void *malloc (size_t <size>) → ritorna puntatore al 1° byte a area pari a "size" byte di contenuto indefinito
con operatore cast è possibile allocare qualunque tip

void free (void * <pointer>) libera memoria prima allocata con la malloc
poi annullo il puntatore pointer = NULL

```
int *p;
p= malloc (sizeof(int));   NO!
p = (int *) malloc (sizeof(int));   (OPERATORE di CAST)
free (p);   p ora punta a un'area di memoria che non è più nostra
p= NULL;
x=*p;   area di memoria non valida! Inoltre se non abbiamo annullato p c'è il rischio di accedere a un'area di memoria che può esser stata assegn ad altri.
```

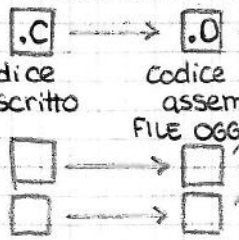
Fallimento della malloc: restituisce NULL se:

- non c'è più lo spazio richiesto disponibile in memoria;
- lo spazio ancora disponibile è frammentato (allora alloco multipli di un n° piccolo o faccio dump su disco).

! È necessario un controllo sulla malloc

MODULARITÀ

Il C consente la gestione di più MODULI (file sorgenti) per maneggevolezza
→ MAIN + moduli che costituiscono la libreria.



.H dichiarazione prototipi (# ifndef (.h) # define (.h) # endif)
.C contiene le definizioni (#include ".h"
MAIN # include " .h".

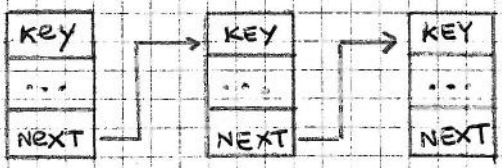
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 7 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 9 di 69

LISTE LINEARI

- elementi sistemati secondo ordine lineare determinato da puntatori

```
struct item {
    int Key;
    struct item *next;}
```



le aree di memoria non sono contigue come nei vettori

- individuata dal puntatore alla testa (e opzionalmente una coda)

uso di struct ricorsive che vengono create così:

```
{ struct item *h;
  h = (struct item*) malloc(sizeof(struct item));
  h -> Key = 0;
  h -> next = NULL;
}
```

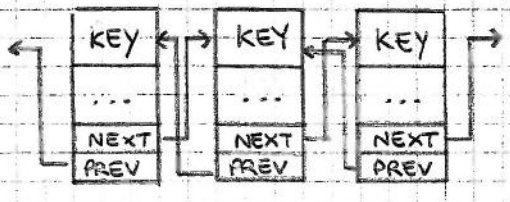
inizializzazione elemento generico e isolato.

LISTE LINKATE: lista di elementi dello stesso tipo connessi da puntatori;

LISTA SEMPLICE: quella rappresentata prima ⊕ semplice ⊖ solo passaggi avanti;

LISTA DOPPIO LINK: ogni elemento possiede puntatore a next e a previous

⊖ più complessa ⊕ consente passaggio avanti/ind

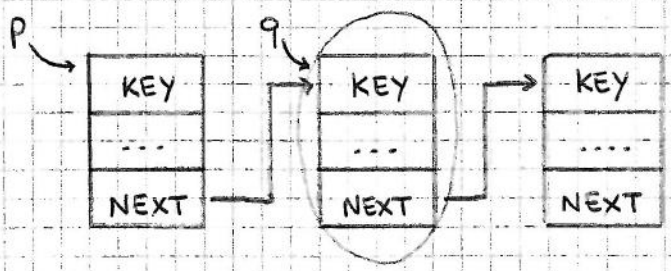


LISTA CON SENTINELLE: esiste un elemento fittizio corrispondente a lista vuota per gestire i casi speciali tipo lista vuota → evita il controllo head == NULL.

⊕ semplifica algoritmo e libera puntatore ⊖ occupa memoria (è necessario?)

LISTA CIRCOLARE: la coda si richiude sulla testa.

NB:



per inserire un elem. dello invertire le operazioni!

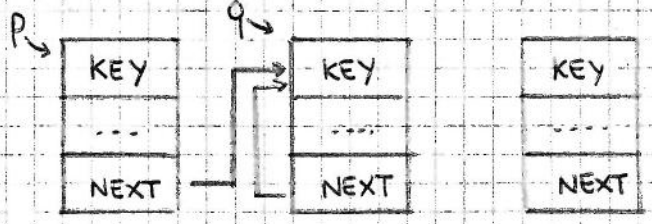
```
Struct item *q;
q = (item*) malloc(sizeof(ite
p -> next = q;
q -> next = p -> next;
```

abbiamo creato una lista chiusa circolare ma attenzione!

abbiamo perso un elemento! che ora non è puntato da alcun puntatore → perso area di memoria e dati in essa contenuti

per avere circolare

```
q -> next = p;
```



LISTA ORDINATA: elementi dati in ordine crescente/decrescente.

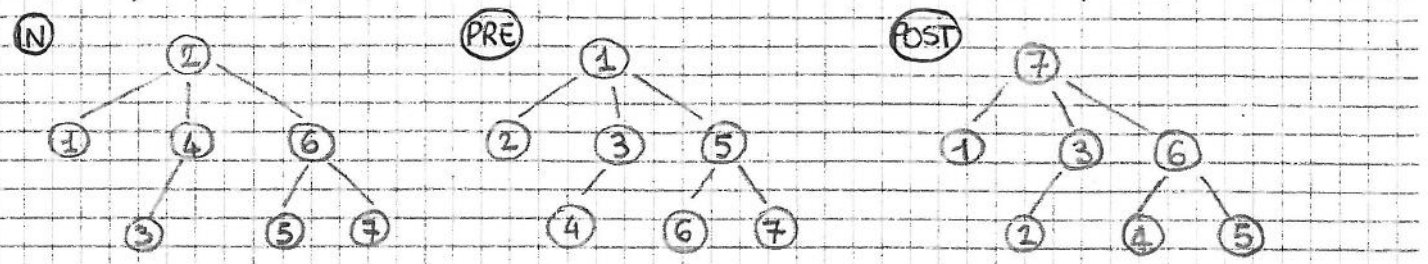
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 9 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 11 di 69

Primitive:

VISITA — INORDER — figlio sx, nodo, figlio dx
PREORDER — nodo, figlio sx, figlio dx
POSTORDER — figlio sx, figlio dx, nodo

SEARCH, INSERT, DELETE costruite adattando la visita all'operazione.




```
void inorder (*head)
{ if (head==NULL) return;
  inorder(head→left);
  visit(head);
  inorder(head→right);}
```

```
void preorder (*head)
{ if (head == NULL) return;
  visit(head);
  preorder(head→left);
  preorder(head→right);}
```

```
void postorder(head)
{ if (head == NULL) return
  postorder(head→left);
  postorder(head→right);
  visit(head);}
```

HEAP

Albero binario in cui ogni figlio ha valore ≤ al valore del padre

serve per implementare code prioritarie (con priorità associata al campo chiave).

È un albero quasi completo: { sbilanciamento max = 1 livello
{ completo su tutti i livelli tranne eventualmente sull'ultimo riempito da sinistra.

PROPRIETÀ:

- elemento maggiore nella radice; (max heap)
- ogni nodo interno ha valore ≥ della discendenza;
- l'altezza di uno heap di n elementi è $\Theta(\log_2(n))$ SHAPE PROPERTY tutti i livelli completi eccetto ultimo

Rappresentabile come albero binario o come vettore (figli di i in 2i e 2i+1)
length = dimensione max vettore
heapsize = n° elementi dello heap

PRIMITIVE:

HEAPIFY RICORSIVA — forza l'applicazione della proprietà dello heap; ricorsiva:
dato l'elemento i → i due sottoalberi di i sono heap;
→ i può essere < dei figli (violazione proprietà!)
→ fa scendere A[i] nella prima posizione utile.
tempo di esecuzione $O(h_i)$ con h_i altezza del nodo i (foglie: livello 0)
→ complessità $O(\log_2 n)$

(prima confronta i due figli e sceglie il maggiore che poi viene confrontato col padre e scambiato se è maggiore del predecessore).

Si può usare la heapify bottom-up per convertire un vettore in uno heap.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 11 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 13 di 69

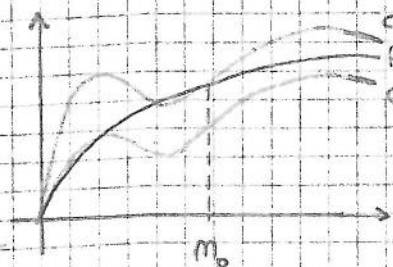
ANALISI di COMPLESSITÀ di ALGORITMI

La complessità di un algoritmo deve essere valutata in termini di:
tempo di calcolo $T(n)$ o di memoria occupata $S(n)$.
L'incidenza sul tempo e sulla memoria cresce al crescere della dimensione n dei dati in ingresso, quindi si conduce un' ANALISI ASINTOTICA, per $n \to \infty$.
L'analisi effettuata deve essere indipendente dal tipo di calcolatore e assumiamo che sia anche indipendente dai valori dei dati in ingresso.

Notazione asintotica (definisce una famiglia di funzioni)

Θ TETA GRANDE

$$\Theta(g(n)) = \{ f(n) : \exists c_1, c_2 > 0 \text{ e } n_0 \geq 0 \text{ t.c. } c_1 g(n) \leq f(n) \leq c_2 g \ \forall n \geq n_0 \}.$$



si dice $f(n) \in \Theta(g(n))$ o $f(n) = \Theta(g(n))$
anche se l'uguaglianza non è proprio stretta.
$f(n) = an^2 + bn + c$, $f(n) = \Theta(n^2)$.
dato un polinomio $p(n)$ di grado d $p(n) = \Theta(n^d)$.
costanti $= \Theta(n^0) = \Theta(1)$.

sia O che $\Omega(g(n))$

esempio. $\frac{1}{2}n^2 - 3n = \Theta(n^2)$.

Devo trovare $c_1, c_2, n_0 \geq 0$ t.c. $c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \ \forall n > n_0$

divido i due membri per n^2: $c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

$\frac{1}{2} - \frac{3}{n} \leq c_2 \quad \forall n \geq 1$ e $c_2 \geq \frac{1}{2}$

$\frac{1}{2} - \frac{3}{n} \geq c_1 \quad \forall n \geq 7$, e $c_2 \leq \frac{1}{14}$ $\quad \frac{7-6}{14} = \frac{1}{14} \leq \frac{1}{14}$

$\to c_1 = \frac{1}{14}$, $c_2 = \frac{1}{2}$, $n_0 = 7$ $\to \frac{1}{2}n^2 - 3n = \Theta(n^2)$

sicuramente esistono molte altre possibilità di scelta, ma ciò che conta è che almeno qualcuna esista.

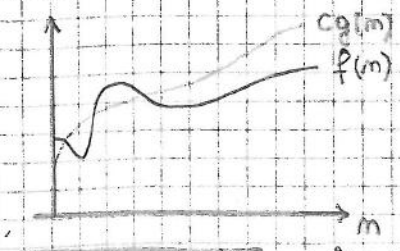
esempio. $6n^3 \neq \Theta(n^2)$

supponiamo che esistano c_2 e n_0 tali che $6n^3 \leq c_2 n^2$, $\forall n \geq n_0$

$\to n \leq \frac{c_2}{6}$ che è impossibile per un n arbitrariamente grande dato che c_2 è una costante arbitraria.

O O GRANDE

$$O(g(n)) = \{ f(n) : \exists c > 0 \text{ e } \exists n_0 \geq 0 \text{ t.c. } \forall n \geq n_0 \quad f(n) \leq c g(n) \}.$$



$\Theta(g(n)) \subseteq O(g(n))$
qualunque funzione lineare $an + b = O(n^2)$
facilmente verificabile prendendo $c = a + |b|$ e $n_0 = 1$

o O piccolo limite superiore che non è asintoticamente stretto. $2n = o(n^2)$

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 13 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 15 di 69

Esempi:

1. RICERCA LINEARE

```
for i=1 to n do
    if A[i] = key  return trovato;
return non trovato;
```

caso migliore: key è il primo elemento $T(n) = O(1)$.

caso peggiore: key è l'ultimo elemento $T(n) = O(n)$ complessità lineare

caso medio: assumiamo una distribuzione equiprobabile $P_i = \frac{1}{n}$

$$T(n) = \sum \frac{1}{n} i = \frac{1}{n} \sum i = \frac{1}{n} \frac{n(n+1)}{2} = \frac{n+1}{2} = \boxed{O(n)}$$

2. INSERTION SORT

(t_j = volte che l'istruzione viene ripetuta)

```
for i=2 to n do
    key = A[j] : inserisci A[j] nella sequenza spostando
                 a destra gli elementi maggiori di A[i]
    i = j-1
    while i>0 and A[i]>key do
        A[i+1] = A[i]
        i = i-1
    A[i+1] = key
    j = j+1
```

COSTO	N. VOLTE
C_1	n
C_2	$n-1$
C_4	$n-1$
C_5	$\sum t_j$ $(j=2..n)$
C_6	$\sum (t_j-1)$ "
C_7	$\sum (t_j-1)$ "
C_8	$n-1$
C_9	$n-1$

$$T(n) = f + e \cdot n + g \sum_{2}^{n} t_j$$

caso migliore: vettore già ordinato: $t_j = 1 \ \forall j \rightarrow T(n) = f + en + g(n-1) = \Theta(n)$

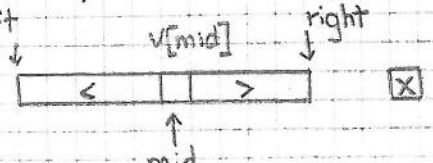
caso peggiore: vettore in ordine inverso: $t_j = j \ \forall j \rightarrow T(n) = f + e \cdot n + g\left(\frac{n(n+1)}{2} - 1\right)$

$$\boxed{T(n) = \Theta(n^2)}$$

3. RICERCA BINARIA (V, left, right, x)

```
if (left > right) return FALSE
mid = (left+right)/2
if x = V[mid] return TRUE
if (x < V[mid])
    binarysearch (V, left, mid-1, x)
if (x > V[mid])
    binarysearch (V, mid+1, right, x)
```

COSTO	n. VOLTE
C_1	1
C_2	1
C_3	1
Problema con dim = $\frac{n}{2}$	?
Problema con dim = $\frac{n}{2}$	?



$$T(n) = C_1 + C_2 + C_3 + T\left(\frac{n}{2}\right) = a + T\left(\frac{n}{2}\right) \qquad \boxed{T(n) = \Theta(\log_2 n)}$$

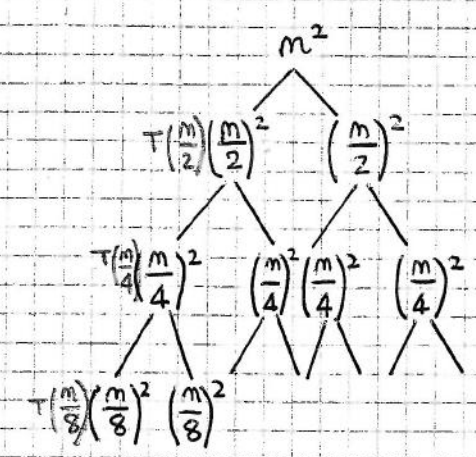
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 15 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 17 di 69

ALBERO della RICORSIONE utile per visualizzare il metodo iterativo.

1. si espande livello per livello — nodo = costo suddivisione
   figli = termine ricorsivo.
2. si sommano i valori su ogni livello (valori in totale pari all'altezza dell'albero → "costo complessivo"

esempio. $T(n) = 2T\left(\frac{n}{2}\right) + n^2$



COSTO

n^2

$\left(\frac{n}{2}\right)^2 + \left(\frac{n}{2}\right)^2 = 2\frac{n^2}{4} = \frac{n^2}{2}$

$\left(\frac{n}{4}\right)^2 \cdot 4 = \frac{n^2}{4}$

$\left(\frac{n}{8}\right)^2 \cdot 8 = \frac{n^2}{8}$

... $\frac{n^2}{2^i}$ $\log_2 n$

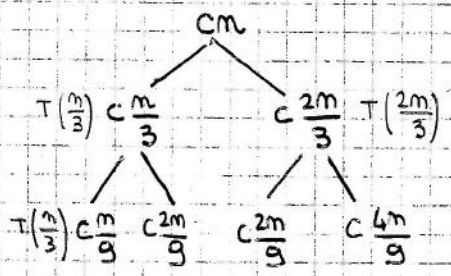
TOTALE = $\Theta(n^2)$

n° livelli = $\log_2 n$ ricavato dal caso banale come prima.

$$T(n) = \sum_{0}^{\log_2 n} \frac{n^2}{2^i} = \sum \frac{n^2}{2^i} = n^2 \sum \frac{1}{2^i} = \Theta(n^2)$$

→ converge a 1

esempio. $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$



COSTO

cn

cn

cn

TOTALE = $cn \cdot h$

h altezza dell'albero la ricavo dal caso banale del ramo più lungo dell'albero dato che questa volta sarà sbilanciato.

$\left(\frac{2}{3}\right)^i n = 1$ $\left(\frac{3}{2}\right)^i = n$ $i = \log_{\frac{3}{2}} n$

$$T(n) = \sum_{i=0}^{\log_{\frac{3}{2}} n} cn = cn \sum_{i=0}^{\log_{\frac{3}{2}} n} = cn \log_{\frac{3}{2}} n = \Theta(n \log n)$$

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 17 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 19 di 69

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

$a = 2 \quad b = 2 \quad f = n$

$n^{\log_2 b} = n \rightarrow$ caso 2: $T(n) = \Theta(n \log n)$

$T(n) = 3T\left(\frac{n}{4}\right) + n^2$

$a = 3 \quad b = 4 \quad f(n) = n^2$

$n^{\log_4 3} \rightarrow$ caso 3: $T(n) = \Theta(n^2)$

Ricorrenze lineari

$T(n) = b_1 T(n-1) + b_2 T(n-2) + \dots + b_k T(n-k)$

con k condizioni iniziali per $T(0), \dots, T(k-1)$

Metodo:

1. Si costruisce l'equazione caratteristica $x^k - b_1 x^{k-1} - b_2 x^{k-2} - \dots - b_k = 0$
2. se ne calcolano le soluzioni $r_1, \dots, r_k$.
3. La soluzione è $T(n) = c_1 r_1^n + c_2 r_2^n + \dots + c_k r_k^n$.
   con $c_1, \dots, c_k$ ricavati dalle condizioni iniziali.

esempio: $T(n) = 2T(n-1)$

$T(0) = 1 \qquad k = 1$

equazione caratteristica $+x - 2 = 0 \quad x = 2$

$T(n) = c_1 2^n \rightarrow T(0) = 1, \; 1 = c_1 \rightarrow T_n = 2^n$.

$T(n) = T(n-1) + T(n-2)$

$T(0) = 1$

$T(1) = 1$

$k = 2 \rightarrow$ eq. caratteristica $x^2 - x - 1 = 0 \rightarrow x = \frac{1 \pm \sqrt{5}}{2}$

$T(n) = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$

$$\begin{cases} c_1 + c_2 = 1 \\ c_1 \left(\frac{1+\sqrt{5}}{2}\right) + c_2 \left(\frac{1-\sqrt{5}}{2}\right) = 1 \end{cases} \qquad \begin{matrix} c_1 = 0{,}723 \\ c_2 = 0{,}276 \end{matrix}$$

$T(n) \simeq \left(\frac{1+\sqrt{5}}{2}\right)^n$.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 19 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 21 di 69

IMPLEMENTAZIONE

```
void Selsort (int V[], int n) {
  int i, j, min;
  for(i=0, i<n-1; i++) {
    min=i;
    for(j=i+1; j<n; j++){
      if ( V[j]<V[min])
        min=j;}
  swap (V[min], V[i]);}}}
```

esempio

7	11	3	4	9	8	2	(1)
1	11	3	4	9	8	(2)	7
1	2	3	4	9	8	11	(7)
1	2	3	4	7	8	11	(9)
1	2	3	4	7	8	9	11

HEAPSORT (ALGORITMO LOGARITMICO O(nlogn)).

Opera come selection sort ma é basato su una struttura dati più efficiente.

1. Buildheap a partire da un vettore;
2. Scarto della radice e modifica dello heap per forzare un ordine (iteraz. heapify

PSEUDOCODICE

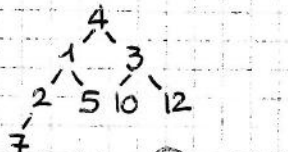
```
Heapsort (A) {
Buildheap (A,n);
for i=n to 2 {
  scambia (A[1], A[i]);
  n=n-1;
  Heapify(A,1);}
}
```

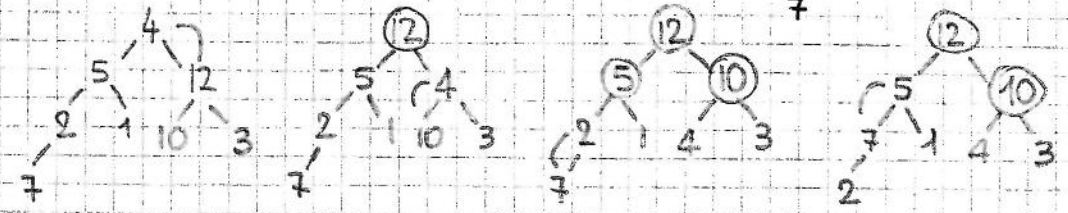
IMPLEMENTAZIONE

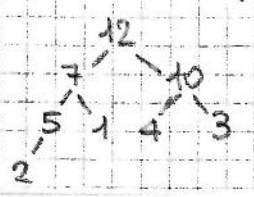
```
void Heapsort (int V[], int dim) {
  int i;
  heap_size =dim;
  buildheap (V, heap_size);
  for(i= dim; i>=2; i--) {
    swap(V,1,i);
    heapify (V,1, --heap_size);}
}
```

Si costruisce l'ordinamento a partire dal fondo.

esempio: A = [4 1 3 2 5 10 12 7]







abbiamo innanzitutto costruito lo heap a partire dal vettore.

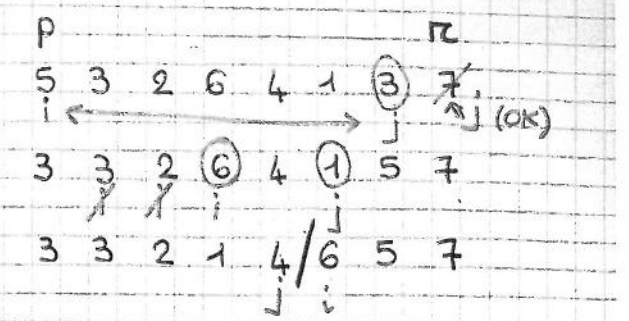
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 21 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 23 di 69

L'elemento "spartiacque" del vettore viene scelto con algoritmo partition.
q sarà indice della posizione in cui si troverà alla fine l'elemento minore o uguale al pivot di valore più grande.

IMPLEMENTAZIONE

```
Partition (A, p, r) {
  x = A[p];
  i = p-1;
  j = r+1;
  while {
    do j = j-1;                 // 1° elemento da dx ≤ x
    while (A[j] ≤ x);
    do i = i+1;                 // 1° elemento da sx ≥ x
    while (A[i] ≥ x);
    if (i < j)                  // se l'array non è stato visto completam. scambia
      scambia (A[i], A[j]);
    else return j;
  }
```



La scelta avrebbe sicuramente effetti positivi sull'algoritmo, ma costa troppo, così si prende il primo elemento del vettore.

COMPLESSITÀ (PARTITION) = $O(n)$.

Analisi del quicksort — ipotesi bilanciata

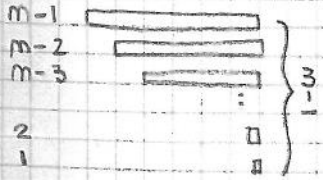
CASO MIGLIORE: si producono due partizioni bilanciate.

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

$\text{costo} = n + \frac{n}{2}\cdot 2 + \frac{n}{4}\cdot 4 + \dots + \frac{n}{n} n = \sum_{i=0}^{\log n} 2^i \frac{n}{2^i} = n(\log n + 1)$

$n^{\log_b a} = n$; $f(n) = n \rightarrow T(n) = O(n \log n)$

CASO PEGGIORE: si produce una partizione in cui una parte ha un solo elemento mentre l'altra ha i restanti $n-1$ elementi.



$T(n) = T(n-1) + O(n)$

$\text{costo} = (n-1) + (n-2) + \dots + 2 + 1 = O(n^2)$.

si ha solo nel caso di sbilanciamento a valore costante $T(n) = T(a) + T(n-a) + O(n)$

si ha quando il vettore è già ordinato. In questo caso è meglio l'insertion!

CASO MEDIO: coincide con il caso migliore (fortunatamente)
anche nel caso di sbilanciamento a proporzionalità costante complessità $O(n \log n)$ (per qualunque sbilanciamento tra 10% e 90%).

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 23 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 25 di 69

COUNTING SORT (ALGORITMO LINEARE O(m))

Si assume che si conosca l'intervallo in cui sono compresi i valori $[1, \dots, k]$

COMPLESSITÀ $= O(k+m) = O(m)$ se $k = O(m)$

non usa i confronti per l'ordinamento

compromesso velocità / memoria: usa 2 vettori ausiliari < output ordinato B O(m); Vettore conteggi C O(k)

se $k \gg n$ memoria richiesta può essere eccessiva

1. Determina per ogni elemento il n° di elementi minori di x
2. usa questa informazione per porre l'elemento al posto giusto

IMPLEMENTAZIONE

```
Countingsort (A, m) {
  for i=1 to K
    C[i]=0          | inizializzo il vettore C
  for i=1 to m
    C[A[i]] = C[A[i]]+1;   | C[i] contiene tutte le occorrenze (n° elementi uguali ad A[i]) e segna n.
  for i=2 to K
    C[i] = C[i]+C[i-1];    | n° elementi totali che ci sono prima di quello
  for i=m to 1
    B[C[A[i]]] = A[i];     → inserisce nella posizione corretta
    C[A[i]] = C[A[i]] -1   → toglie l'occorrenza dell'elemento inserito
```

esempio.

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

le range in cui sono compresi i valori è $[1, 6]$

	1	2	3	4	5	6
C	0	0	0	0	0	0
	+1		+1	+1		+1
	+1		+1	+1		
				+1		

(inizializzato)

$A[m] = 4$

$C[A[m]] = C[4] = 7$

C	2	0	2	3	0	1	$C[A[i]] = C[A[i]] + 1$
C	2	2	4	7	7	8	$C[i] = C[i] + C[i-1]$

a questo punto inizio ad ordinare nel vettore

	1	2	3	4	5	6	7	8
B	1	1	3	3	4	4	4	6

↑ 1° elemento inserito (si mette dal fondo)

(NB) ORDINAMENTO STABILE: preserva l'ordine relativo dei dati con chiavi uguali all'interno della sequenza da ordinare!

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 25 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 27 di 69

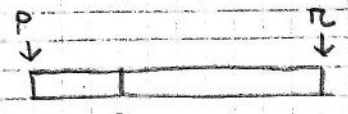
Heapselect applicata alla ricerca del mediano ricade nella complessità dell'ordinamento $\left(k = \frac{n}{2} = O(n)\right) \longrightarrow T(n) = O(n \log n)$

Selezione randomizzata

approccio divide-et-impera simile al quicksort, ma non serve cercare in entrambe le partizioni: basta in una di esse.

Random select (A, p, r, k) cerca k-esimo elemento più piccolo in A

```
if (p=r) return A[p]
q = randompartition (A,p,r)
s = q-p+1 (indice di q in [p.....r])
if (k≤s)
   return Randomselect (A,p,q,k)
else return Randomselect (A,q+1,r,k-s)
```



se $k < s$ è a sinistra, se no è a destra

COMPLESSITÀ

- CASO MIGLIORE $T(n) = T\left(\frac{n}{2}\right) + \theta(n) = \theta(n)$ qualunque rapporto partizione
- CASO PEGGIORE $T(n) = T(n-1) + \theta(n) = \theta(n^2)$
- CASO MEDIO $T(n) \leq \frac{2}{n} \sum T(k) + \theta(n)$ $O(n)$

Randompartition = versione partition in cui la scelta pivot è random.

NB Quando si ricorre sulla partizione destra si cerca l'elemento i-kesimo.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 27 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 29 di 69

MASSIMO e MINIMO: non richiedono confronti per come è stato costruito!

```
Tree_min (T) {
  while (left[T] ≠ NULL)
    T = left[T];
  return T; }
```

```
Tree_max (T) {
  while (right[T] ≠ NULL)
    T = right[T];
  return T; }
```

SUCCESSORE E PREDECESSORE — Successore = chiave min sottoalbero dx

```
Tree_successor (x) {
  if (right[x] ≠ NULL)
    return Tree_min (right[x]);
  y = p[x];
  while (y ≠ NULL and x = right[y])
    x = y;
    y = p[y];
  return y; }.
```

Se x ha figlio dx, successore è minimo del sottoalbero dx

risale l'albero fino a trovare un nodo che è figlio sx del padre di x.

CANCELLAZIONE

3 casi:
1. nodo senza figli → elimina
2. nodo ha un figlio → collega padre del nodo al figlio del nodo
   poi elimina
3. nodo ha 2 figli → cerca il successore (che ha solo figlio dx)
   elimina successore (come caso 2)
   copia campi valore del successore nel nodo da eliminare.

```
Tree_Delete (T, z) {
  if (left[z] = NULL or right[z] = NULL)
    y = z;
  else y = Successor (z);
  if (left[y] != NULL)
    x = left[y];
  else x = right[y];
  if (x != NULL)
    p[x] = p[y]
  if (p[y] = NULL) root[T] = x;
  else if (y = left[p[y]]) left[p[y]] = x;
       else right[p[y]] = x
  if (y != z) chiave[z] = chiave[y];
    return y; }
```

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 29 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 31 di 69

TABELLE HASH

Sono tabelle ad accesso diretto, in cui la ricerca non è più basata sul confronto. L'efficienza non è più legata all'universo di dati (U), quanto piuttosto all'insieme di chiavi effettivamente usate (dimensione $= m$).

Se fattore di carico $\alpha = \frac{n}{m}$ è basso → tabelle hash $[|U| = m]$

Si tratta di una generalizzazione di vettori: vettore di puntatori a struct:

- a ogni chiave corrisponde una posizione nella tabella (un indice)
- la tabella è di dimensione $m << |U|$ (dim. pari all'universo delle chiavi)
- funzione hash $h: U \to \{0, \dots, m-1\}$ calcola indice a partire da chiave
  indice $= h(k)$.
- elemento chiave k viene memorizzato nella posizione $h(k)$.

→ il range degli indici si riduce da $|U| = m$ a n.

→ tempi di esecuzione $O(1)$.

Funzione hash non è iniettiva → COLLISIONI (chiavi distinte-stesso valore hash)

per ridurre le collisioni:

- scelta della funzione hash;
- tecniche di gestione delle collisioni ← concatenazione (liste di collisione)
  indirizzamento aperto.

Funzione hash

caratteristiche:

CRITERIO di UNIFORMITÀ SEMPLICE: il valore hash di una chiave k è uno dei valori $0 \dots m-1$ in modo equiprobabile $\sum_{k:h(k)=j} P(k) = \frac{1}{m}$

scegliendo una chiave a caso questa finisce nella j-esima cella.

In genere la distribuzione di probabilità delle chiavi P non è nota; nella pratica si usano tecniche euristiche che permettono di creare funzioni hash che si comporta bene con buona probabilità.

Una buona funzione hash deve usare TUTTE le cifre della chiave.

METODO della DIVISIONE $\boxed{h(k) = k \bmod m}$ $m \neq 2^p$, di solito m primo.

METODO della MOLTIPLICAZIONE/DIVISIONE $\boxed{h(k) = \lfloor m \overbrace{(kA \bmod 1)}^{\text{parte frazionaria di } kA} \rfloor}$ $k \in [0,1]$

di solito $m = 2^p$ per semplicità

Se si hanno chiavi non numeriche:

caratteri: riconducibili a dati numerici (codice ASCII)

stringhe: rappresentazione come dato numerico in base

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 31 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 33 di 69

Gestione delle collisioni – indirizzamento aperto. ($n \leq m$, $\alpha \leq 1$)

Tutti gli elementi vengono memorizzati in tabella. La collisione è risolta cercando un'altra cella libera.

→ La funzione hash deve essere anche funzione delle posizioni esaminate $h(k,i)$

prende una chiave e un indice e genera una nuova posizione.

$$h: U \times \{0,1,\dots,m-1\} \to \{0,1,\dots,m-1\}$$

data una chiave k: la prima volta $h(k,0)$, poi $h(k,1)$, la generica $h(k,i)$.

```
Hash-Insert (T,k) {
  i=0;
  repeat j=h(k,i)
    if (T[j]=NULL){
      T[j]=k;
      return j}
    else i=i+1;
  until i=m;
  error "overflow";}
```

```
Hash-Search (T,k) {
  i=0;
  repeat j=h(k,i)
    if(T[j]=k){
      return j;
    i=i+1;}
  until i=m or T[j]=NULL;
  return NULL;}
```

1) SCANSIONE LINEARE — fz hash primaria $h(k,i) = (h'(k)+i) \% m$

problema di concatenazione primaria: le collisioni vengono distribuite ma si concentrano attorno a una posizione di hash.

Esempio.

0	22
1	88
2	
3	
4	4
5	15
6	17
7	59
8	
9	31
10	10

$m=11$; {22 4 31 10 15 17 88 59}

$h(k,i) = (k \bmod 11 + i) \% 11$

$22 \bmod 11 = 0$ $i=0$ libera ✓

$4 \bmod 11 = 4$ $i=4$ libera ✓

$31 \bmod 11 = 9$ $i=9$ libera ✓

$10 \bmod 11 = 10$ $i=10$ libera ✓

$15 \bmod 11 = 4$ $i=4$ occupata ✗ $i=5$ ✓

$17 \bmod 11 = 6$ $i=6$ libera ✓

$88 \bmod 11 = 0$ $i=0$ occupata ✗ $i=0+1=1$ ✓

$59 \bmod 11 = 4$ $i=4$ occupata ✗ $i=4+1=5$ ✗

$i=4+2=6$ ✗ $i=4+3=7$ ✓

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 33 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 35 di 69

PROGRAMMAZIONE DINAMICA

- Divide in sottoproblemi (non necessariamente indipendenti)
- Li risolve una sola volta e memorizza la soluzione in una tabella (così si evita di risolvere più volte lo stesso problema, da esponenz. a polinomiale)

SOLUZIONE TIPICA: esplorazione delle soluzioni in modo bottom-up

- Applicazione ai casi: se c'è più di una soluzione, a cui è associabile una preferenza e si vuole determinare quella con indice ottimo.

1. Caratterizzazione della struttura di una soluzione ottima, in termini di sottoproblemi: SOTTOSTRUTTURA OTTIMA, costituita da sottosoluzione ottima. (Non tutti i problemi sono caratterizzabili in questo modo!)
2. Definizione ricorsiva del valore di una soluzione ottima (ricorrenza; la soluzione ottima contiene le sottosoluzioni ottime ai sottoproblemi).
3. Calcolo del valore di una soluzione ottima "bottom-up" (prima i + semplici) si usa una tabella per memorizzare le soluzioni ai sottoproblemi, evitando di ripetere quelli già fatti.
4. Costruzione della soluzione ottima (unica).

esempio Parentesizzazione ottima nella moltiplicazione tra matrici (vogliamo minimizzare il numero di moltiplicazioni). sfruttando la proprietà associativa.

c_0 righe A_1
c_1 colonne A_1 = righe A_2
c_{i-1} righe A_i
c_i colonne A_i = righe A_{i+1}

$P(m)$ = n° modi di calcolare il prodotto di m matrici $A_1 \times A_2 \times \dots \times A_m$.

- combinando la parentesizzazione delle prime k matrici e poi delle rimaneuti $m-k$ (esponenziale!)

$$P(m) = \sum_{k=1}^{m-1} P(k)\,P(m-k) = \Omega\left(\frac{4^m}{m^{32}}\right)$$

(1) Caratterizzazione soluzione ottima: divido il problema in due sottoprobl
- → prodotto prime k matrici $A_{1\dots k}$
- → prodotto delle altre $m-k$ matrici $A_{k+1\dots m}$

} devono godere anche di parentesizz. ottima!

la soluzione $A_{1\dots m}$ è il prodotto di $A_{1\dots k}$ e $A_{k+1\dots m}$
il costo del prodotto $A_{1\dots m}$ è la somma del costo dei prodotti $A_{1\dots k}$ e $A_{k+1\dots m}$ più il costo del prodotto finale $= c_0 c_k c_m$.

REQUISITO della SOTTOSTRUTTURA OTTIMA e della SOVRAPPOSIZIONE di SOTTOPROBLEMI

PROP. Ogni soluzione ottima al problema contiene le soluzioni ottime dei due sottoproblemi (dim.)

alcuni sottoproblemi si devono ripetere

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 35 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 37 di 69

④ $S[i,j]$ contiene il valore di k su cui spezzare il prodotto $A_{i\ldots j}$.
→ $A_{i\ldots j} = A_{i\ldots k} \times A_{k+1\ldots j}$

ALGORITMO RICORSIVO

```
MCM (A, s[], i, j)
  if (j>i)
    then k = s[i,j]
         X = MCM (A, s, i, k)
         Y = MCM (A, s, k+1, j)
         return Prod_Matrici (X,Y)
    else return A[i].
```

$A[]$ vettore di lunghezza n contiene le matrici $[A_1 \ldots A_n]$

$S[]$ matrice $n \times n$ contiene k ottimo per ogni coppia di indici

VARIANTE: RICORSIONE CON MEMORIZZAZIONE
(↑ tipico del divide & conquer; ↑ tipico della programmazione dinamica)

Quando risolvo un sottoproblema per la prima volta lo memorizzo in tabella poi per ogni sottoproblema controllo se è già stato risolto: SI → uso il risultato; NO → calcolo e memorizzo

La complessità non può superare il numero di casi distinti.

```
Mem MCO (c[])
  n = lungA (c) - 1
  for i=1 to n
    do for j=i to n
    do m[i,j] = ∞
  return cercaMC (c, 1, n)

CERCA MC (c[], i, j)
  if m[i,j] < ∞
    return m[i,j]
  if (i=j)
    m[i,j] = 0
  else for k=i to j-1
    do costo = cercaMC(c, i, k) + cercaMC(c, k+1, j) + c[i-1]c[k]c[j]
    if costo < m[i,j]
      m[i,j] = costo
  return m[i,j];
```

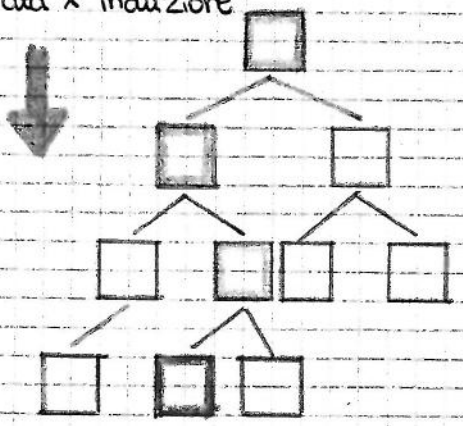
La variante presenta dei vantaggi quando non debbano esser risolti tutti i sottoproblemi (tratta solo quelli necessari).

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 37 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 39 di 69

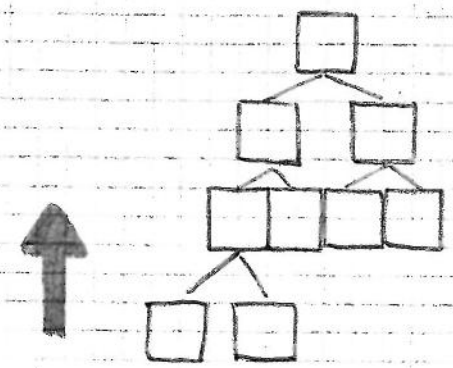
GREEDY

- la scelta fatta da un algoritmo greedy può dipendere dalle scelte fatte fino a quel momento;
- non dipende dalle scelte future né tantomeno da tutte le soluzioni del sottoproblema
- prende la decisione e cambia il percorso verso la soluzione dopo la decis
  NON RICONSIDERA MAI LE DECISIONI GIÀ PRESE
- procede in modo iterativo top-down, viene generata una e una sola sequenza di soluzioni.
- difficoltà: dimostrare che greedy permette di trovare un ottimo globale va dimostrata x induzione



PR. DINAMICA

- ha in comune la sottostruttura ottima
- procede in bottom-up
- esaustiva e garantisce di trovare una soluzione
- prende le decisioni basandosi su tutte le decisioni prese allo stadio precedente.
- la decisione dipende dalla soluzione ai sottoproblemi (tabella)



tutti i nodi hanno un valore.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 39 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 41 di 69

else if $(w = p_i)$

$M(i,w) = \max \{ M(i-1,w), c_i \}$

else $M(i,w) = M(i-1,w) \}$

esempio $M(i,w) = \max \{ M(i-1,w), M(i-1, w-p_i) + c_i \}$

	c_i	p_i
1	24	6
2	12	2
3	25	5

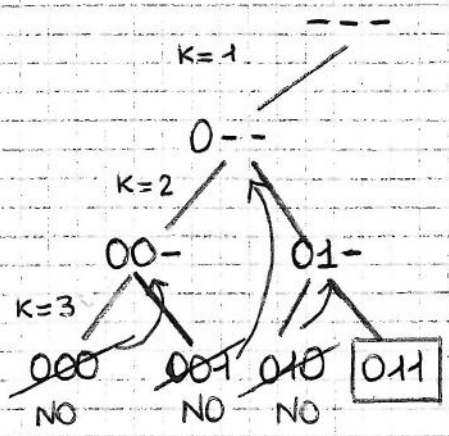
11

i \ w	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	24	24	24	24	24	24
2	0	0	12	12	12	12	24	24	36	36	36	36
3	0	0	12	12	12	25	25	37	37	37	37	49

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 41 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 43 di 69

esempio. $n=3$



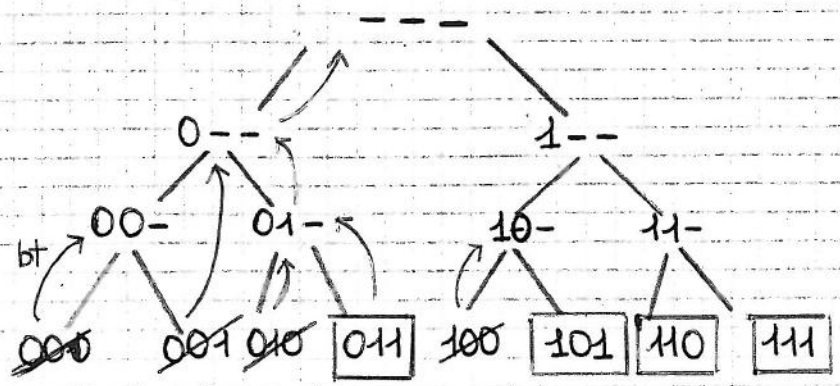
frecce rosse sono backtrack

TROVATO UNA SOLUZIONE!

TUTTE LE SOLUZIONI

```
Backtrack (K)
  for (i = 1 .... m)
    choose K-th candidate
    if (acceptable)
      store it
      if (K<m)
        backtrack (K+1)
      else
        print solution
      Remove from solution
```

esempio



© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 43 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 45 di 69

GRAFI

Sono rilevanti l'insieme di oggetti e le connessioni.

$G = (V, E)$ V = vertici, E = archi (definito come la coppia di vertici che unisce)

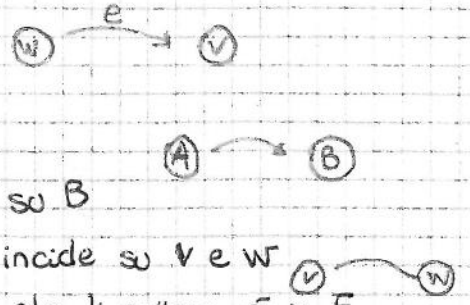
un grafo è <u>orientato</u> se E è una relazione binaria tra i vertici (archi orientati)

un grafo è <u>non orientato</u> se gli archi non sono orientati

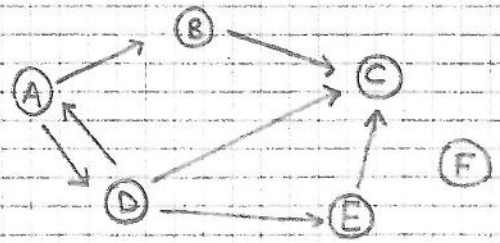
arco $e = (w, v) \in E$

<u>INCIDENZA</u>

- e incidente su v
- e incidente da w
- v è incidente su e
- (A, B) è incidente da A su B
- (grafi non orientati) e incide su v e w



Un vertice w si dice <u>adiacente</u> a un altro v se è l'arco che li collega è in E. (v, w)



- B adiacente ad A
- C adiacente a B e D
- A adiacente a D e viceversa
- B non è adiacente a D né a C
- F non è adiacente ad alcun vertice.

<u>PROP.</u> In un grafo non orientato l'adiacenza è simmetrica.

<u>Grado di un vertice</u> = n° di archi che da esso dipartono;
in un grafo orientato si distinguono il <u>grado entrante</u> e il <u>grado uscente</u>

<u>Grafi pesati</u>: ogni arco ha un peso associato definito da una funzione di costo (quando tra due vertici non esiste arco $\mapsto$ costo infinito).

<u>Sottografo di G</u> $H = (V^*, E^*) \;/\; V^* \subseteq V$, $E^* \subseteq E$ e $E^* \subseteq V^* \times V^*$

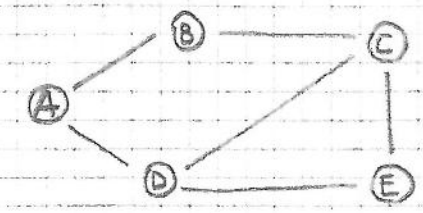
<u>sottografo indotto</u> da V^* è $H = (V^*, E^*)$ tale che $E^* = \{(v, w) \in E \;/\; v, w \in V^*\}$.

<u>Percorso</u> sequenza di vertici $< w_1, w_2, \ldots, w_m >$ tale che $(w_i, w_{i+1}) \in E$ $1 \le i \le n-1$
la lunghezza sarà $m-1$.

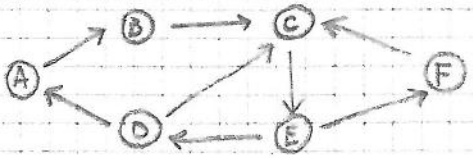
<u>Percorso semplice</u> se tutti i suoi vertici sono distinti

<u>Nodo w raggiungibile</u> da v tramite p se esiste un percorso tra v e w $v \xrightarrow{p} w$

Un grafo non orientato è <u>connesso</u> se esiste un percorso da ogni vertice a ogni altro vertice



Un grafo orientato è <u>fortemente connesso</u> se esiste un percorso da ogni vertice a ogni altro vertice



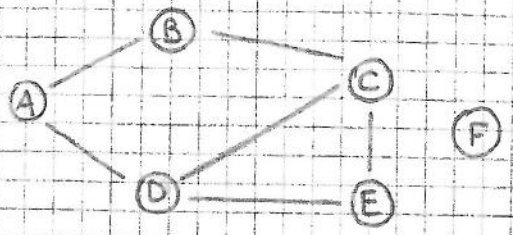
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 45 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 47 di 69

RAPPRESENTAZIONE di GRAFI

Con MATRICE di ADIACENZA $|V|^2$

$$M(v,w) = \begin{cases} 1 & \text{se } (v,w) \in E \\ 0 & \text{altrimenti} \end{cases}$$

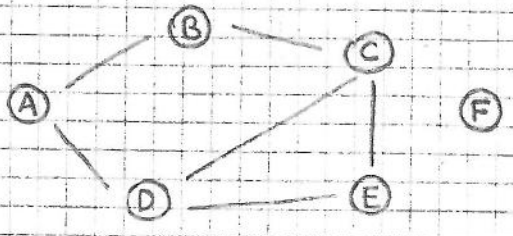


	A	B	C	D	E	F
A	0	1	0	1	0	0
B	1	0	1	0	0	0
C	0	1	0	1	1	0
D	1	0	1	0	1	0
E	0	0	1	1	0	0
F	0	0	0	0	0	0

matrice simmetrica se grafo non orient

CON LISTE di ADIACENZA $a|V| + 2b|E|$

$L(V)$ = lista di W tale che $(V,W) \in E$ per $V \in V$

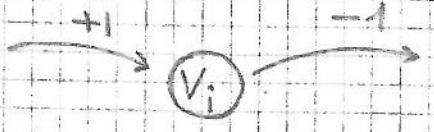


- A → B → D
- B → A → C
- C → B → D → E
- D → A → C → E
- E → C → D
- F

se grafo orientato spazio = $a|V| + b|E|$

CON MATRICE di INCIDENZA $\Theta(M|E|)$

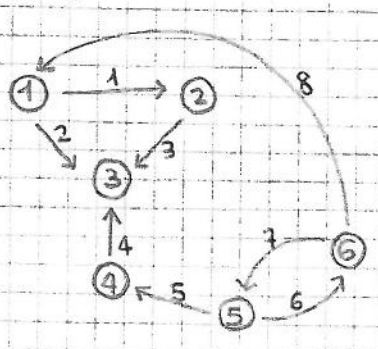
avendo numerato i vertici $1, 2, \dots, |V|$



$$D = (d_{ij}) \quad \begin{cases} +1 & \text{se } e_j \text{ entra in } V_i \\ -1 & \text{se } e_j \text{ esce da } V_i \\ 0 & \text{altrimenti} \end{cases}$$

dim = $|V| \times |E|$

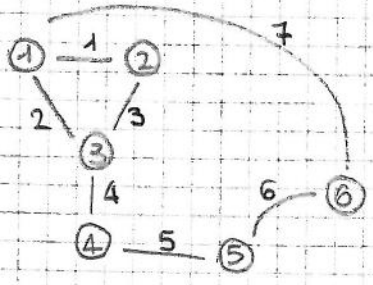
se il grafo non è orientato non si distingue il verso → matrice simmetrica



	1	2	3	4	5	6	7	8
1	-1	-1	0	0	0	0	0	+1
2	+1	0	-1	0	0	0	0	0
3	0	+1	+1	+1	0	0	0	0
4	0	0	0	-1	+1	0	0	0
5	0	0	0	0	-1	-1	+1	0
6	0	0	0	0	0	+1	-1	-1

archi

vertici



	1	2	3	4	5	6	7
1	1	1	0	0	0	0	1
2	1	0	1	0	0	0	0
3	0	1	1	1	0	0	0
4	0	0	0	1	1	0	0
5	0	0	0	0	1	1	0
6	0	0	0	0	0	1	1

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 47 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 49 di 69

VISITE di GRAFI = attraversamento di tutti i vertici a partire dalla sorgente

BIANCO = vertice inesplorato;

GRIGIO = vertice esplorato ma ancora utile per visitare le adiacenze;

NERO = vertice esplorato con tutte le sue adiacenze → non è più utile.

O(V)
```
Initialize (G)
    for ogni u ∈ V do
        color [u] = white
```

O(V+E)
```
Visit (G, S)   S = sorgente
    color[S] = gray
    while ci sono vertici grigi do
        u = vertice grigio
        if ∃ v bianco adiacente a u
            then color [v] = gray   (P[v] = u)
            else color [u] = black
```

OPZIONALE: permette di ricordare l'arco percorso per visitare v, ossia il suo predecessore

al termine tutti i vertici neri hanno un predecessore

sottografo dei predecessori $G_P = (V_P, E_P)$ è un albero o una foresta.

$V_P = \{ v \in V : P[v] \neq NULL \} \cup \{s\}$

$E_P = \{ (P[v], v) \in E : v \in V_P - \{s\} \}$

al termine di visit (G,S) V_P è l'insieme di tutti i vertici neri (quelli raggiungibili da s)

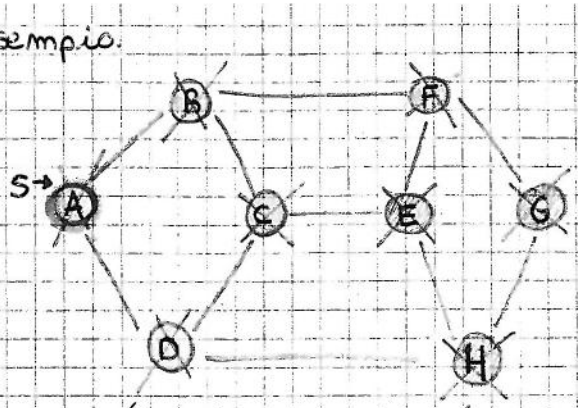
Per gestire l'insieme di vertici grigi si può usare una struttura dati D con le sue primitive.

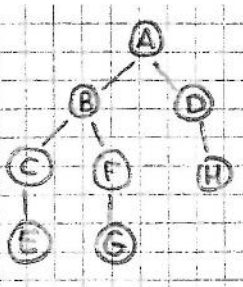
```
Visit (G,s)
    D = create_empty
    color[s] = gray
    add (D,s)
    while not_empty(D) do
        u = first (D)
        if ∃ v bianco adiacente a u
            then color[v] = gray
                 add (D,v)
            else color [u] = black
                 remove_first (D)
```

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 49 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 51 di 69

esempio

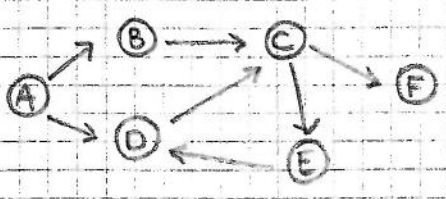




Q	A	B	D	C	F	H	E	G
	0	1	1	2	2	2	3	3

DFS DEPTH-FIRST-SEARCH

VISITA IN PROFONDITÀ si scoprono prima i vertici adiacenti all'ultimo adiacente scoperto; generalizzazione della visita in preordine per alberi binari.



A, B, C, E, D, F

1. Viene processato il vertice s;
2. Ricorsivamente si processano tutti i vertici adiacenti a s; necessario evitare di riprocessare vertici già visitati (attenzione ai cicli)
3. Il sottografo dei predecessori può formare una foresta di alberi perché la visita può esser ripetuta da più sorgenti.

$G_p = (V, E_p)$

$E_p = \{(p[v], v) \in E : v \in V \text{ e } p[v] \neq NULL\}$.

4. Usa uno stack per memorizzare i nodi grigi.

```
DFS (G,S)
    S = make_empty_stack
    color[s] = gray
    push(S,s)
    while not_empty(S) do
        while c'è un v adiacente a top(S) non visto do
            if color[v] = white
                color[v] = gray
                p[v] = top(S)
                push(S,v)
        color[top(S)] = black
        pop(S)
```

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 51 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 53 di 69

DFS con GRAFI ORIENTATI

CLASSIFICAZIONE di ARCHI → dell'albero (tree, T)
- all'indietro (backward, B) — collega a un antenato
- in avanti (forward, F) — collega a un discendente
- di attraversamento (cross, C) — se collega vertici che non sono antenato/discendente

in base al colore del vertice raggiunto

1. BIANCO → (u, v) è T dell'albero
2. GRIGIO → (u, v) è B all'indietro
3. NERO → (visita di v è terminata) → (u, v) è F se $d[u] < d[v]$
   → (u, v) è C se $d[v] < d[u]$

Applicazioni:
- ordinamento topologico di un DAG
- calcolo delle componenti fortemente connesse.

ORDINAMENTO TOPOLOGICO di un DAG

ordinamento lineare di tutti i vertici tale che se esiste l'arco (u, v) in E, allora u precede v. (Non è unico!)

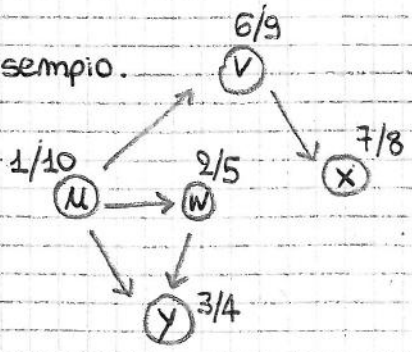
TOPSORT (G)

Chiama DFS(G) per calcolare i tempi $f[v]$

Ogni volta che un vertice è "terminato" aggiungilo in testa a una lista

Ritorna la lista di vertici

esempio.



OUTPUT u V x w y

CALCOLO delle COMPONENTI CONNESSE

componente connessa = sottografo massimale connesso di un grafo non orientato (massimale = non è possibile aggiungere vertici o archi lasciandolo connesso)

CC componenti connesse (di grafi non orientati) = classi di equivalenza dei vertici rispetto alla relazione di connettività. ("raggiungibile da")

SCC componenti fortemente connesse (di grafi orientati) = classi di equivalenza rispetto alla relazione di connettività forte ("esiste un cammino tra")

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 53 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 55 di 69

ALBERI di COPERTURA MINIMI — MST

Dato un grafo non orientato, connesso, con archi pesati

ALBERO di COPERTURA MINIMO = sottografo aciclico $T \subseteq E$ che connette tutti i vertici minimizzando il peso totale. $W(t) = \sum W(u,v)$

T è un albero perché è aciclico e connesso (tocca tutti i vertici)

Non è unico!

$A \subseteq T$

IdeA accrescere un sottoinsieme A di archi di un albero di copertura aggiungendo ogni volta un ARCO SICURO (mantiene la proprietà per A di essere un sottoinsieme di archi di un albero di copertura)

A inizialmente è vuoto

```
Generic MST (G, w)
    A = ∅
    while A non forma un albero di copertura
    do trova un arco sicuro (u,v)        approccio greedy
        A = A ∪ {(u,v)}
    return A
```

Taglio = partizione dei vertici V

Arco attraversa il taglio = gli estremi dell'arco sono in partizioni diverse

Taglio rispetta un insieme di archi = nessun arco attraversa il taglio

Arco leggero = arco con peso minimo tra' quelli che attraversano un dato taglio

Teorema $G(V,E)$ grafo non orientato, connesso, con una fz di peso w
$A \subseteq E$ contenuto in qualche albero di copertura minimo per G.
$(S, V-S)$ è un taglio che rispetta A
(u,v) arco leggero che attraversa il taglio
$\Rightarrow$ (u,v) è sicuro per A.

Oss. A è sempre aciclico altrimenti un MST T conterrebbe un ciclo
$G_A = (V,A)$ è una foresta, e ogni sua cc è un albero. gli archi sicuri connettono cc

Corollario $G(V,E)$ grafo non orientato, connesso, con una fz di peso w
$A \subseteq E$ contenuto in un albero di copertura minimo per G
C componente connessa (albero) nella foresta $G_A = (V,A)$
(u,v) arco leggero che connette C a qualche componente in G_A
$\Rightarrow$ (u,v) è sicuro per A.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 55 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 57 di 69

Algoritmo di Prim

mantiene A come singolo albero (una singola componente connessa)
- albero parte da un vertice arbitrario (radice) e cresce fino a coprire tutto.
- ogni volta si aggiunge un arco leggero che collega un vertice in A con uno in V-A (sicuro per il corollario, dato che A è una CC)
- greedy perché aggiunge sempre l'arco di peso minimo.

```
MST-PRIM (G, w)
A = r        albero formato da un solo nodo
ordina gli archi di E per peso w non decrescente
while (A ha meno di |V| nodi) do
    trova l'arco (u, v) di costo minimo incidente su A
    A = A ∪ {(u, v)}
return A.
```

EFFICIENZA dipende da scelta del nuovo arco. Soluzione:
1. $Q = V - A$ (coda con priorità con tutti i vertici che non sono nell'albero in costruz)
2. la priorità è basata su key[v] = minimo tra i pesi degli archi che collegano v a un qualunque vertice dell'albero in costruzione A.
3. per ogni nodo si introduce parent p[x] che serve per ricostruire l'albero. [visita con l'aggiunta dell'ordine di priorità!].

$A = \{(v, p[v]) : v \in V - \{r\} - Q\}$
alla fine $Q = \emptyset$, $A = \{(v, p[v]) : v \in V - \{r\}\}$.

```
MST-PRIM (G, w, r)   con coda di priorità
Q = V[G]   tutti i vertici
for ∀ u in Q
    do key[u] = ∞
key[r] = 0
p[r] = NULL
while Q ≠ ∅
  do u = EXTRACT-MIN (Q)
    for ogni v in Adj[u]
        if v in Q e w(u, v) < key[v]
            then p[v] = u
                 key[v] = w(u, v).
```

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 57 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 59 di 69

CAMMINI MINIMI

costo del cammino $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$ — $p = (v_0, \dots, v_k)$

peso di un cammino minimo $\delta(u,v) = \begin{cases} \min \{w(p) : u \xrightarrow{p} v\} & \text{se } \exists \text{ cammino} \\ \infty & \text{altrimenti} \end{cases}$

cammino p da u a v minimo se $w(p) = \delta(u,v)$

può non essere definito in presenza di archi o cicli negativi.

Peso unitario → BFS

ALBERO dei CAMMINI MINIMI: sottografo orientato $G'(V',E')$ di G

- G' albero con radice in s
- V' insieme vertici raggiungibili da s (vertici distinti)
- $\forall v \in V'$ l'unico percorso semplice da s a v è un percorso minimo.

Proprietà algoritmi → sottostruttura ottima cammini minimi
→ tecnica rilassamento

Lemma 1 – SOTTOSTRUTTURA OTTIMA dei CAMMINI MINIMI

$G = (V,E)$ grafo ~~non~~ orientato con funzione di peso w

$p = \langle v_1, \dots, v_k \rangle$ percorso minimo tra v_1 e v_k

$p_{ij} = \langle v_i, \dots, v_j \rangle$ sottopercorso di p tra v_i e v_j $\quad \forall i,j$

$\Rightarrow$ p_{ij} è un percorso minimo tra v_i e v_j

Corollario 1

$G = (V,E)$ grafo pesato orientato con funzione di peso w.

percorso minimo p decomponibile in $s \xrightarrow{p'} u \to v$ per qualche u

$\Rightarrow$ peso del percorso minimo tra s e v è $\delta(s,v) = \delta(s,u) + w(u,v)$

Lemma 2.

$G = (V,E)$ grafo orientato con funzione di peso w ; s è sorgente

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

Rilassamento verifica se è possibile ottenere un percorso migliore tra s e v passando per u.

```
Relax (u, v, w)
    if d[v] > d[u] + w(u,v)
        then  d[v] = d[u] + w(u,v)
              p[v] = u
```

$d[v]$ dist da sorgente: estremo superiore lunghezza del percorso minimo tra s e v, inizializzato a ∞

$p[v]$ predecessore di v

$w(u,v)$ peso arco (u,v)

cerca di dim. costo cammino minimo – se riesce → nuovo PREDECESSORE

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 59 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 61 di 69

Algoritmo DijKstra

PESI NON-NEGATIVI

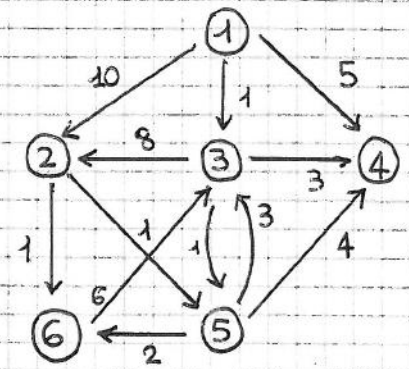
- Mantiene un insieme S di vertici i cui pesi del percorso minimo sono già stati determinati $d[u] = \delta(s,u)$
- seleziona a turno il vertice u in $V-S$ col minimo valore $d[u]$
- inserisce u in S
- rilassa tutti gli archi uscenti da u.
- usa una coda a priorità Q che contiene i vertici usando i valori d[] come chiave

```
        DIJKSTRA (G,S)
O(V)    Inizializza (G,S)        (d=∞ e p=NULL , d[s]=0)
O(V)    S=∅
        Q = V(G)
O(V)    while (Q≠∅)
O(logV)     u = Delete_min (Q)
            S = S∪{u}
O(E)        for each vertice v adiacente a u
O(logV)         Relax (u,v,w)   cambia valore di d[] → corrisponde a
                                Decrease_key (d[v], d[u]+w(u,v))
                                sulla coda a priorità
```

TOT = $O((E+V)\log V)$

COMPLESSITÀ $O(E\log V)$ la struttura dati è efficiente x mantenere priorità

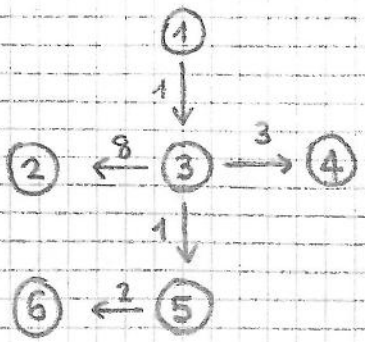
esempio



S =

1	3	5	4	6	2
0	1	2	4	4	9

1	2	3	4	5	6
∅	∞	∞	∞	∞	∞
	10	(1)	5	∞	∞
	9		4	(2)	∞
	9		(4)		4
	9				4



© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 61 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 63 di 69

Algoritmo di Floyd-Warshall

ALGO di PR. DINAMICA
CAMMINI MINIMI TRA TUTTE LE COPPIE

Archi anche negativi (ma cicli no)

$G = (V, E, w)$ grafo orientato pesato con $|V| = n$ rappr. con matrice di adiac.

$$W[i,j] = \begin{cases} 0 & \text{se } i = j \\ w(i,j) & \text{se } i \neq j \text{ e } (i,j) \in E \\ \infty & \text{se } (i,j) \notin E \end{cases}$$ MATRICE di ADIACENZA

$$P_{ij} = \begin{cases} NULL & \text{se } i = j \text{ o se } i \neq j \text{ e } j \text{ non è raggiungibile da } i \\ k & \text{se } i \neq j \text{ e } j \text{ raggiungibile da } i \text{ e } k \text{ predecess. di } j \text{ lungo il percorso minimo.} \end{cases}$$

↑ ogni riga P_i induce un sottografo dei predecessori

1. PROPR. SOTTOSTRUTTURA OTTIMA — dato un cammino ottimo p ogni suo sottocamm. è ottimo.

Data $W = w_{ij}$ considero il percorso minimo p tra i e j (supponiamo p = m archi)
se $i \neq j$ $p = i \xrightarrow{p'} k \to j$ dove p' ha alpiù $m-1$ archi ed è il percorso minimo tra i e k $\quad \delta(i,j) = \delta(i,k) + w_{kj}$

2. DEFINIZIONE RICORSIVA → devo trovare $d_{ij}^{(|V|-1)}$ $\forall i,j$.

$d_{ij}^{(m)}$ = peso percorso min tra V_i e V_j usando alpiù m archi.

caso base: $m = 0$ $\quad d_{ij}^{(0)} = \begin{cases} 0 & i = j \\ \infty & i \neq j \end{cases}$

caso generico: $m \neq 0$ $\quad d_{ij}^{(m-1)}$ matrice dei percorsi minimi con alpiù $m-1$ archi
→ percorso minimo tra V_i e V_j:
- contiene $m-1$ archi → costo $d_{ij}^{(m-1)}$
- contiene m archi → costo $d_{ij}^{(m)} = d_{ik}^{(m-1)} + w_{kj}$

ossia

$$d_{ij}^{(m)} = \min\left(d_{ij}^{(m-1)}, \min_{1 \le k \le n}\left\{d_{ik}^{(m-1)} + w_{kj}\right\}\right) = \min_{1 \le k \le n}\left\{d_{ik}^{(m-1)} + w_{kj}\right\}$$

Se il grafo non contiene cicli di peso negativo, ogni percorso minimo è un percorso semplice con alpiù $|V|-1$ archi

3. CALCOLO del VALORE della SOLUZIONE OTTIMA

algoritmo riceve $W = w_{ij}$ matrice di adiacenza del grafo pesato.
Vengono calcolate $D^{(0)} = D^{(1)} = D^{(2)} = \ldots = (D^{|V|-1})$
$D^{(|V|-1)}$ contiene i pesi dei percorsi minimi
$D^{(1)}$ coincide con W
$D^{(k)} = (d_{ij}^{(k)})$

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 63 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 65 di 69

INTRACTABLE PROBLEMS & NP COMPLETENESS

PROBLEMI TRATTABILI = problemi risolvibili in tempo polinomiale da algoritmi (esponenti piccoli di solito)

PROBLEMI INTRATTABILI = risolvibili solo con algoritmi con tempo di esecuzione $> n^k$

non tutti i problemi sono risolvibili con una soluzione polinomiale, la conoscenza della NP COMPLETEZZA è importante per definire un algoritmo, di solito si ricorre ad algoritmi approssimati (euristici): lavorano bene nella maggior parte dei casi. La teoria della NP-completezza è limitata ad un tipo di problema:

i PROBLEMI di DECISIONE: problemi con risposta binaria $\{0,1\}$.

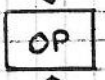
I problemi di ottimizzazione possono essere facilmente ricondotti a problemi di decisione (con artifici atti a restringere il campo delle possibilità).

Finora abbiamo considerato solamente algoritmi deterministici, per i quali cioè il risultato di ciascuna operazione è unicamente definita (l'hardware è determ).

Per la teoria della NP COMPLETEZZA rimuoviamo questa restrizione ed assumiamo che il risultato delle operazioni non sia univocamente determinato.

→ ALGORITMI NON DETERMINISTICI

non esistono macchine non deterministiche



ma danno un aiuto intuitivo per capire quali problemi (non) siano risolvibili con algoritmi determ.

O(1) {
- **choice** $(x_1, \dots, x_n)$ — ritorna un elemento arbitrario (non random) (sceglie il valore giusto)
- **failure** ( ) — segnala una terminazione senza successo
- **success** ( ) — segnala una terminazione con successo

Nota bene che non ritornano un valore. failure e success.

```
ND search (A, n, K) {
    j = choice (1, n)
    if (A[j] == K)
        print j;
        success;
    else
        print '0';
        failure; }
```

cerca K nell'array A

cerca j t.c. $A(j) = K$ oppure $j = 0$ $K \notin A$

sostituisce il ciclo for deterministico.

O(1)

(corrisp. deterministico O(m))

Concettualmente la choice corrisponde a fare tante copie dell'algoritmo quante sono le possibilità di scelta; la prima copia che raggiunge il valore atteso termina anche tutte le altre (spiegazione deterministica non funziona così!).

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 65 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 67 di 69

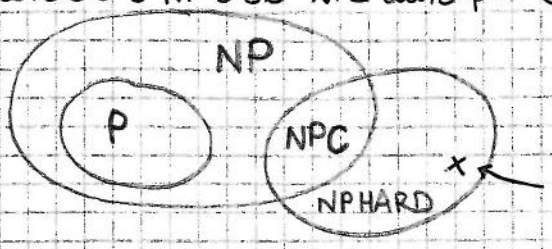
NP-HARD per ogni altro problema A in NP $A \propto A$ (almeno difficile quanto un NP)

NP-COMPLETE { NP (i più difficili problemi in NP)
per ogni altro problema A' in NP $A' \propto A$

devo verificare che è NP e che NPC altro può $\propto$ a questo problema.

$NPC \cap P = \emptyset$

$NPC = NP \cap NP\text{-HARD}$



non verificabile in tempo polinomiale (problemi che non sono di decisione).

SAT BOOLEAN SATISFIABILITY $O(m)$ (D-Complex esponenziale)

- problema NPC (il primo)
- tutti i problemi NP $\propto$ SAT.
- trovare i valori (vero/falso) per x_i così che soddisfi l'equazione booleana. (and, or, not)

```
ND sat (E, m) {
    for (i = 1 to m) do
        x_i = choice (true, false)
    if {E(x_1 — x_n) is true
        success
    else failure }
```

CDP CLIQUE DECISION PROBLEM

- Dato un grafo non orientato, dire se contiene sottografi connessi di dim = K.
- problema NPC (è NP: può essere verificato in tempo polinomiale).

3CNF SAT

- 3-SAT = { f / f è conjunctive normal form: ogni clausola ha 3 lettere e f è soddisfabile }
- 3-SAT $\propto$ CDP

  trasformiamo le clausole 3CNF in un grafo; ogni K-clique corrisponde ad un'assegnazione accettabile.

HAMILTONIAN CYCLE

percorre n vertici di G visitando tutti i vertici una volta e tornando a quello di partenza.

SAT $\propto$ DIRECTED HAMILTONIAN CYCLE

TSP TRAVELLING SALESPERSON PROBLEM

Deve visitare n vertici visitando ciascuno una volta sola e ritornando quello di partenza, c'è ne è uno lungo al massimo d?

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 67 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 69 di 69

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 69 di 69