

# RIPASSO

## 02. ANALISI DELLA COMPISSITÀ:

• PRENDERE UN ALGORITMO UNA VOLTA, IN TERMINI DI SPazio E TEMPO

→ E' IMPORTANTE DARE MACCHINALE DA DATI (DIPENDE DA N SESSI)

• CLASSIFICAZIONE:

- 1: COSTANTE
- log n: LOGARITMICO
- n: LINEARE
- n log n: LINEARITICO
- $n^2$ : QUADRATICO
- $n^3$ : CUBICO
- $2^n$ : ESPONENZIALE

• ANALISI ASINTOTICA: FACHO UNA STIMA PER  $n \rightarrow \infty$ , NEL CASO PREGIUDIZIO

• NOTAZIONE ASINTOTICA O: (PIÙ USATA)

$$\underline{\Omega} < \Theta < \overline{O} \cdot T(n) = O(y(n)) \quad / y(n) \text{ è una funzione UNIMINIMA SUPERiore}$$

caso di  $T(n)$   
 $n \rightarrow \infty$  MAX

$$\rightarrow 0 \leq T(n) \leq c y(n)$$

$$\rightarrow \text{TEOREMA: se } f(n) = a_1 n^m + a_2 n^{m-1} + \dots + a_0 \rightarrow y(n) = n^m$$

• NOTAZIONE ASINTOTICA  $\underline{\Omega}$ :

$$\cdot T(n) = \underline{\Omega}(y(n)) \quad / y(n) \text{ è una funzione uniminima inferiore}$$

caso di  $T(n)$

$$\rightarrow 0 \leq c y(n) \leq T(n)$$

• NOTAZIONE ASINTOTICA  $\Theta$

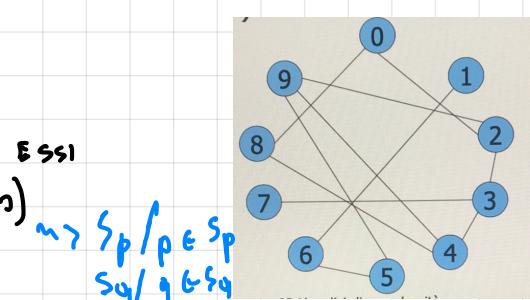
$$\cdot T(n) = \Theta(y(n)) \quad / y(n) \text{ UNITA' ASINTOTICA SISTEMATO con } \bar{T}(n)$$

$$\rightarrow 0 \leq c_1 y(n) \leq T(n) \leq c_2 y(n)$$

$$\rightarrow \text{TEOREMA: se } T(n) = \Theta(y(n)) \Rightarrow T(n) = O(y(n)) \text{ e } T(n) = \underline{\Omega}(y(n))$$

## • ALGORITHMS OF ONLINE CONNECTIVITY :

- $\forall$  COPPIA DI NUMERI  $p \neq q$ , CONSIDERANDO SE ESSI SONO COME APP' IN UN'ALTRA RIGA (PIN) 



- SE SON GOVERN -> PROSEGUN

- \* SE NON SONO COINVOLTI: VERSO OLTRE 100 PERSONE (UN 10%)

3;2 → LO SCOPO È DI RIDURRE IL FUMIGAZIONE AL MINIMO POSSIBILE

QUICK FIND  $\rightsquigarrow$  PREVIEWS IL FINI :  $O(1)$  UNION  $O(N^2)$

```
#include <stdio.h>
#define N 1000
int main()
{
    int t, p, q, i, j, k;
    char str[100];
    for(t = 1; t <= N; t++) {
        scanf("Input p q: %d %d", &p, &q);
        while (scanf("Input %c", &k) != EOF) {
            if (k == 'd') {
                id[p] = id[q];
                printf("%d %d already connected\n", p, q);
            } else if (k == 'u') {
                for (t = id[p]; t <= id[q]; t++) {
                    if (id[t] == -1) {
                        id[t] = t;
                        break;
                    }
                }
                printf("%d %d Not yet connected\n", p, q);
            }
            printf("Input p q: %d %d\n");
        }
    }
}
```

→ LO SCOPÙ È DI EVIDENZIARE UN ARCHETIPO INFORMATIVO

$$\sim O(N_{\text{cores}} * N_{\text{edges}})$$

-> È UN ACCORDATO DI ONLINE CONNECTIVITY, POSSIBILE SU UN TUTTO INCLUSO

**VERIFICATION**: if  $(i \in [p] : i \in [q]) \rightarrow_{\text{def}} \text{relevant sons : PAPP}$   
 $\uparrow_p$  are PAPP.  
a for " "  $i \in [p]$   $i \in [q]$

$\rightarrow \text{final}(p, q) \rightarrow \text{continuous if } (\text{iol}[p] = \text{iol}[q])$

$\rightarrow \text{Union}(p, q) \hookrightarrow \text{id}[p] = \text{id}[q]$  (o inverso)

• "n" COPPIE \* QM. VETRORE "► OP. DA FARRE

• QUICK UNION ~> quick union

Page 1 of 1

- curve is BACK FIND, m if  $(\text{iol}[p] \neq \text{iol}[q])$

$$\rightarrow \text{vol } [p] = q$$

↪ NOUN C'E UN UNICO RAPPRESENTATORE, CREDO PUÒ SERVIRE

•  $n^6$  COPPIJS  $\times$  LUMMEZOR COTTON → OP. OA PARIS

- WEIGHTED QUICK UNION:  $O(\lg n)$  (UNION  $O(1)$ )

- CORVZ IS QUICK UNION, MA CONSEGNA IL MODO PER CONNETTERE 2 MIEGL.

-> PERMANTMENT CATERING TROPPO LUNGHETTI, PACCIO IN MOOD CHIUSI

's/θəʊt̪/ pl̪ ɿ placeɪθ̪ vFvθ̪ VMT̪ ɹ̪ ɹ̪z(ɹ̪) m̪ ɿəʊθ̪  
θ̪ ɿəm̪

• M<sup>o</sup> CUPRIF<sup>+</sup> + CUNNUS CATRNA → Gr. 04 fm

$\hookrightarrow$   $\log \eta_{\text{viscous}}$   $\log \eta_{\text{copper}}$   $\log \eta_{\text{water}}$

① log n ?  $\rightarrow$  THE COMPLEXITY IN CASE PSEUDO (of values)

→ IN ~~mein~~ ~~der~~: ALTBRÜCKE FÜR MEINEN HANDEL MIT NORD

UNION PEGASUS: 2 DISJOINT CON STRINGS  $h \rightarrow h$  FINITE  $\vdash h+1 \perp 2^{h+1}$  WNL  $\Rightarrow 1 = 2 \perp 2$

$\rightarrow \forall$  passo  $i$ : la  $gim$  rec' nello ADMING rappresenta:  $i$  passi  $\rightarrow 2^i$  bit.  $\Rightarrow 2^i \leq N$  FINO  $\Rightarrow i = \log_2 N$

### • 03. GLI ALGORITMI DI ORDINAMENTO

- ORDINAMENTO
  - ✓ INIZIALE: DATI IN MEM. CENTRALE (accesso puro)
  - ↳ RESTAURO: DATI SU MEM. RESTAURO
- ORDINAMENTO
  - ✓ IN LOCO: USA LA STESSA STRUTTURA DATI
  - ↳ STABILITÀ: MANTIENE L'ORDINE DEI DATI FINO ALLA FINE

#### • CLASSIFICAZIONE (COMPLESSITÀ):

- $O(n^2)$ : INVERSIONE SINT, SELECTION SINT, BUBBLE SINT (BASATO SU CONFRONTO)
- $O(n^{3/2})$ : SHELL SINT (CON KMM)
- $O(n \log n)$ : MERGE SINT, QUICK SINT, HEAP SINT (APPLICABILI SOLO CON HP RESTRIZIONE MIGLIORI SUL DATI)
- $O(n)$ : COUNTING SINT, RADIX SINT, BIN/BUCKET SINT

→ SECONDO OPERAZIONI DI CONFRONTO E STABILITÀ:

→ PER UN MEDIO TIPO DI VARIABILE LIBERI, IN CUI SI PRESUNTE UNA

CHIUSURA DI ORDINAMENTO SÌ I DATI SONO UNI

→ CREO POCHE FUNZIONI CHE DEFINISCONO LE VARIABILI RELEVANTI PER UN DATO TIPO

• ALGORITMI DI CONFRONTO:  $\bullet a_i : a_j \rightarrow (a_i < a_j) \sqcup (a_i = a_j)$

COSTO PER COMPARA

IN WORST CASE:

$P_n$

DE VARIE

ESIGENZE

ORDINAMENTO POSSIBILE

CONFRONTO:  $h$   
SOMMA:  $z^h$   
MAX:  $n^h$

•  $n$  VARI DI GRADO:  $n! \sim n^n$  ORDINAMENTO

• COMPLESSITÀ:  $n^h$  DI CONFRONTI ( $h$  è il grado dell'ordine)

•  $n^h$  FORMA:  $2^h$

• APPROXIMAZIONE DI STERZIN:  $n! > \left(\frac{n}{e}\right)^n$

$$\sim z^h \geq n! > \left(\frac{n}{e}\right)^n \rightarrow h = \log_2 \left(\frac{n}{e}\right)^n = n \log_2 \frac{n}{e} =$$

↳ È IL MASSIMO

N° di relazioni possibili

NEI CASI PEGGIORI, CON MAX H

$$= n \log n \cdot n \log e = \sqrt{2} (n \log n)$$

NON O!

↳ È UN LIMITE INFERIORE

### • 6.1 ALGORITMI I PEGAZUH D. ORDINATENYU:

$$A[j] > A[j+1]$$

; BUBBLE SORT:  $(\text{IN } \text{LOCO}, \text{ STABILUS})$   $O(n^2)$

OPPURE  
XOMME  
DRI

- \* 2 sotto vett.: uno a dx ordinato, uno a sx da ordinare

- V CICLU CONGRUENTE  $A[i] \cup A[j+i]$ , scambi si  $A[j]$  manterrà

- A CICLO CRESCENTE US 9M. DEC VERT. ORDINARIO A DX

- $$\bullet \text{LEMMA DE STIRLING: } \frac{c_{1, n} \text{ F.Y.T. : } n-1}{c_{1, n} \text{ INT. : } n-1-i} \rightarrow T(n) = (n-1) + (n-2) + \dots = \frac{1}{2} n(n-1) = D(n^2)$$



$O(n^2)$  • SELECTION SORT: (IN LUCO, NON STABUE)

L> SEZIONI IL MINIMO DA SX L> IN ELEMENTI LONTANI  
PIÙ SINGOLARE ANCHE C'È UNA 6 3

- 2 sotto vir. : SX ordinato, DX DISORDINA

- $\forall$  ciclo souz A[i] = mim

- $\text{COSW} + DX$  SE C<sub>E</sub> UN MIN  $\rightarrow$  IF  $\rightarrow$  SENSIBIL

- consciousness (Analol. BS):  $O(n^2)$

• **INSERTION SORT:** (IN LOC, STABLU)  $O(n^2)$

L> INSEKTYCO SFOLUJĄCO K WETERYNARZ

- Z serne vekt:  $SX$  original,  $DX$  as anslag.  $\rightarrow X^2 A[j]$

- A few suggestions to consider for next year's organization.

- POSISCONO L'URINARIO NELLA POSIZIONE COMUNA A DX SHIFTANDO I VESICALI PIÙ GRANDI A QUESINA

- SHELL SORT :** (IN LOC, NON STABLING)

↳ KMM

- CONSIGN UN DRAFT INVOICE DEI SOCI RIVARO 1 01/01.

- Organ in blood low IN SITUATION SORE

- **dimensiōn** h (sociaal uva seeranža: **KUNTH, SPÖRER (1974)**)

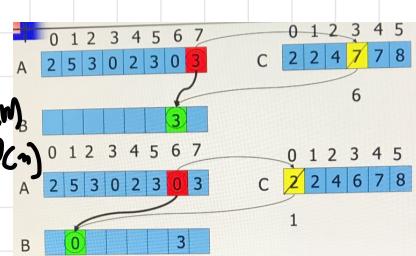
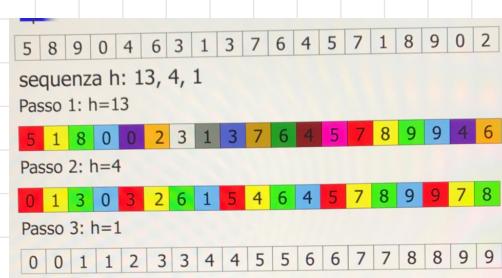
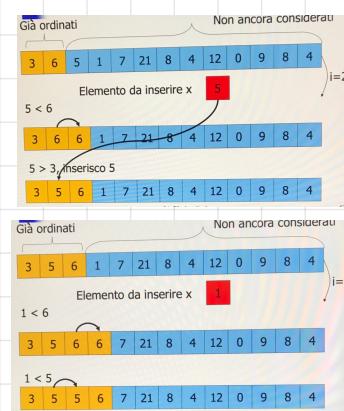
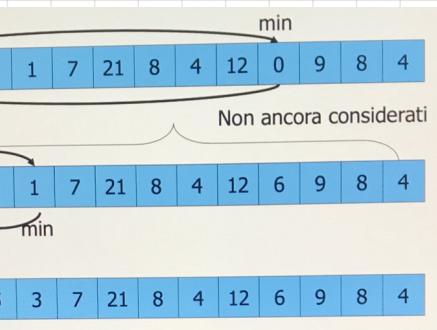
$O(n)$

• CONVESSITÀ:  $O(3/2)$  con seq. di KNNM:  $b = 3b + 1$

COUNTING SONG: (NON IN COCO, STARSUE), OMICELLO DA  
CANTERÀ O(?)

- 3 verryzzi:  $A[\cdot]$  on ordinare,  $B[\cdot]$  vett. risultato  
 $C[i]$  corrispondente valore in  $A[\cdot]$  ->  $S[i]$

- $\forall_{\text{POST}} : (\text{INORDER } A[J] \text{ IN } C[], C[A[J]]) \rightarrow \text{IN POS-1}$



## • 04. GRAFI e ALBERI

- $G = \{V, E\}$ ,  $V$ : VERTICI,  $E$ : ARCFMI

- POSSONO ESSERE ORIENTATI o NON ORIENTATI

- GRADO DI UN VERTICE  $a$ :

out-degree( $a$ ): n° arcini

in-degree( $a$ ): acci entranti

out-degree( $a$ ): arcini uscenti

Per  
orientati

- CAMMINO ( $p$ )

- È IL PERCORSO DA UN VERTICE PER ARRIVARE A UN ALTRO:  $p: U \rightarrow U'$

- K = LUNGHEZZA di  $p = n^0$  ARCFMI ATTIVAMENTE

- $p$  È SEMPRE SE NON PASSO PIÙ VOLTE SULLO STESSO ARCO

- CICLO:  $V_0 = V_K$ , CICLO:  $V_0 = V_K \wedge K \geq 1$

- GRAFO CONNESSO:  $\forall v_i, v_j \rightarrow \exists p: v_i \rightarrow v_j$ , COMPONENTE CONNESSA: SOTTOGRAFO CONNESSO MASSIMALE

- GRAFO FORTEMENTE CONNESSO:  $\exists p_1, p_2 / \forall v_i, v_j \rightarrow \exists p_1: v_i \rightarrow v_j \wedge \exists p_2: v_j \rightarrow v_i$

- GRAFO COMPLETO: OGNI VERTICE È COLLEGATO DIRETTAMENTE CON OGNI ALTRO VERTICE ( $K=1$ )

$$\hookrightarrow \text{QUANTO ARCFMI?} \xrightarrow{\text{non orientato}} |E| = \frac{n^0 \text{ connessioni in } V}{2} = \frac{|V|!}{((|V|-2)! \cdot 2!)}, V = \frac{|V|!}{(|V|-2)!} = \frac{1}{2} |V| \cdot (|V|-1)$$

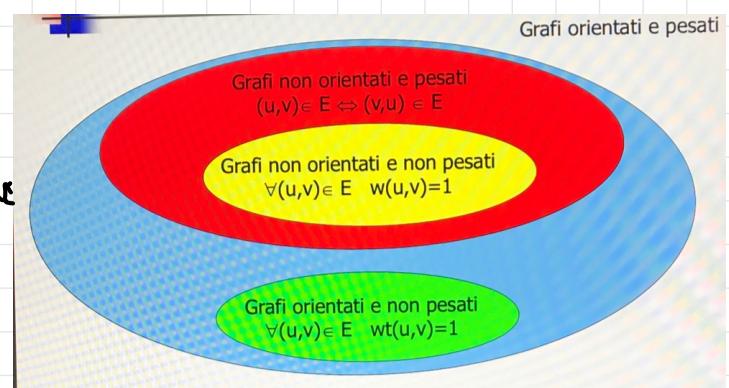
$$\cdot |E|_{\text{ARCFMI}}, |V| \text{ VERTICI} \xrightarrow{\text{orientato}} |E| = \frac{n^0 \text{ orientazioni}}{2} = \frac{|V|!}{(|V|-2)!} = |V| \cdot (|V|-1)$$

- GRAFO DENSE:  $|E| \approx |V|^2$

- GRAFO SPARSO:  $|E| \ll |V|^2$

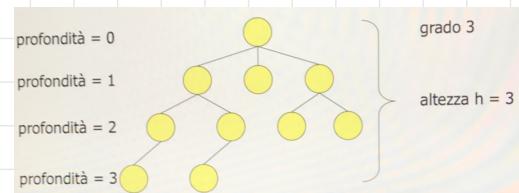
- GRAFO PESATO: OGNI ARCO HA UN PESO

- GRAFO BIPARTITO: SI SUDDIVIDE IN 2 SOTTOSSETTI/OGNI VERTICE È CONNESSIONE SOLO A VERTICI DELLA ALTRA PARTE



## ALBERI

- ALBERI SONO RADICALI (0 LIANI) : GRADICI SONO ORIENTATI, CONNESSI, ACCIUCI  
↳ IL LORO INTITUTO È UNA FORESTA
- $G = \{V, E\} / |E| = |V| - 1$
- } UN SOLO CAMMINO RADICE  $\rightarrow$  SI PUÒ INFORMATO UNO MEMBRA P' OLTRE: RISULTA  $\rightarrow \dots \rightarrow$  FORESTA
- PROPRIETÀ:
  - GRADO ( $T$ ) = MAX DI FIGLI DI UN VERGATE
  - PROFONDITÀ ( $x$ ) = LUNGHEZZA DEL CAMMINO DA  $R$  A  $X$
  - ALTEZZA ( $T$ ) = PROFONDITÀ MAX



### • RAPPRESENTAZIONE:

- K PUNTATORI A K PUNTI  $\rightarrow$  INEFFICIENTE IN TERMINI DI SPAZIO (HANNA BASSO IC)
  - RAPP. LEFT·CHILD·RIGHT·SIBLING, Ogni NODO HA:
    - 1 PUNTATORE AL FIGLIO
    - 1 PUNTATORE AL FRATELLO A DX (o NULL)
    - 1 PUNTATORE AL FIGLIO DAL P.M. (o NULL)
- $\nwarrow$  2 PUNTATORI
- EFFICIENTE IN TERMINI DI SPAZIO,  
NO IN TERMINI DI TEMPO  $O(k)$

### • ALBERI BINARI:

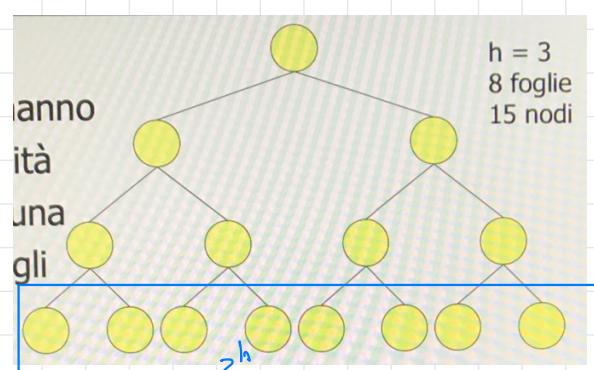
- GRADO ( $T$ ) = 2, Ogni NODO È  $\begin{cases} \text{NULL} \\ \text{OPPURE} \\ \text{RADICE, SOTTOSETTO SX o DX} \end{cases}$

### • ALBERO BINARIO COMPLETO:

- TUTTI I FOLI HANNO STESSA PROFONDITÀ
- OGNI NODO HA 2 FIGLI (TRanne PER I FOGLI)

#### • PROPRIETÀ

$$\text{SE ALBERO } h: \begin{aligned} & \cdot 2^h : n^{\circ} \text{ FOGLI} \\ & \cdot 2^{h+1}-1 = n^{\circ} \text{ NODI} \end{aligned}$$



• ALBERO BINARIO COMPLETO A SIMMETRIA: L'ULTIMO LEVEL È COMPOSTO SOLO DI SX

• ALBERO BINARIO BIANCCHIATO: OGNI CAMMINO  $R \rightarrow$  FOGLI HA STESSA LUNGHEZZA

$\hookrightarrow$  QUASI BIANCHIATO SE DIFFERENZE DI MAX DI 1

## SEQUENZE UNERI:

✓ VETTORI: DATI CONTENUTI IN MEMORIA  $\rightarrow$  ACCESSO ORETTU  $\Theta(1)$

✓ LISTE: DATI NON CONTENUTI IN MEMORIA  $\rightarrow$  ACCESSO SEQUENZIALE  $\Theta(n)$   
 $\rightarrow$  SEGUO SCORRERE TUTTI I DATI

## OPERAZIONI SULLE LISTE:

### • RICERCA

(NO ORDINE) (NO ORDINE) (ORDINATO)

• INSERZIONE: IN TESTA, IN CODA  $\rightarrow$  IN POSIZIONE  
(NO ORDINE) (ORDINATO)

• ESTRAZIONE: IN TESTA, IN POSIZIONE

## • CODE GENERALIZZATE: COLLEZIONI DI DATI CON OPERAZIONI:

• INSERT

- SEARCH

• DELETE

## Criteri per DELETE:

### • CRONOLOGICO:

• Estraggo l'elemento inserito PIÙ RECENTEMENTE

$\rightarrow$  LAST-IN FIRST-OUT (LIFO), STACK o PILA, INSERIMENTO: PUSH, ESTRAZIONE: POP

• Estraggo l'ultimo inserito MAIOR RECENTEMENTE  
 $\rightarrow$  FIRST-IN FIRST-OUT (FIFO), NUOVE o CODA, INSERIMENTO: IN CODA, ESTRAZIONE: DALLA CODA

### • PREFERITA:

• Si lavora con l'elemento DALLA TESTA, si estraie l'ultimo inserito

$\rightarrow$  CODA A PREFERITA

### • CONTINUO

• Estrarre in PAGE AD UN CERTO

$\rightarrow$  TABELLA DI SIMBOLO

• 05

RICORSIONE e DIVIDE et CONQUER

$$\sum_{i=0}^k x^i = \frac{x^{k+1}-1}{x-1} / \sum_{i=0}^n i = \frac{1}{2} n(n+1)$$

- $f$  RICORSIVA: SI PUÒ ESPRIMERE COME  $S(D) = f(S(D'))$ ,  $S(D_0) = S_0$

→

- DIVIDE: DIVISO IL PROBLEMA IN  $\alpha$  SOTTOPROBLEMI DI DIMENSIONE  $n' < n$
- CONQUISTA: RISOLVIMENTO DI PROBLEMI ELEMENTARI
- RECOMBINAZIONE: COSTRUZIONE SOLUZIONE COMPLESSA

→

- $\alpha = 1$  → RICORSIONE LINEARE
- $\alpha > 1$  → RICORSIONE MUROVIA

 $a$ :  $n'$  sottoproblem $b$ : fattore di divisione

→

- $b = \text{cost.}$  →  $n'$  si riduce di UN'UNITA' COST. AD OGNI PASSO
- $b = \text{FATTORE COST.}$  →  $n'$  si DIVIDE CON UN FATTORE  $b$
- $\hookrightarrow n^{\theta}$  VARIABILI

- DIVIDE AND CONQUER:  $\alpha > 1$ ,  $b = \text{cost.}$
- DECREASING AND CONQUER:  $\alpha > 1$ ,  $b = \text{decreas.}$

ANALISI DI COMPLESSITÀ:

program  $T(n)$ :

```

if  $n = 1$ 
    soluz  $B(1)$ 
else
    soluz  $D(n)$ 
    sotto prob  $T(\frac{n}{b})$  -
    unione  $C(n)$ 

```

IN GENERALE, L'ES. DI UN ALGORITMO DELL'RICORSIONE SI PUÒ SCRIVERE COME:

$$\cdot T(n) = D(n) + \alpha T\left(\frac{n}{b}\right) + C(n)$$

SOTTOPROBLEMI  
COSTO  
NUOVA DIVISIONE.

SE DIVIDE AND CONQUER  
COSTO  
DI DIVISIONE  
COSTO  
DI RECORSIONE

$$\cdot T(n) = D(n) + \sum_{i=0}^{n-1} T(n-i) + C(n)$$

SE DECREASING AND CONQUER  
( $n_i = \frac{n}{b^i}$ )

ESEMPIO:

$$\cdot \underline{\text{FATTORIALE}}: \begin{cases} n! = n \cdot n-1! \\ 0! = 1 \end{cases} \rightarrow \begin{cases} T(n) = 1 + T(n-1) \\ C(n) = \Theta(1), D(n) = \Theta(1) \end{cases} \rightarrow T(n) = \sum_{i=0}^{n-1} 1 = 1 + (n-1) = n$$

$$\rightarrow T(n) = O(n)$$

$$\cdot \underline{\text{FIBONACCI}}: \begin{cases} \text{FIB}_n = \text{FIB}_{n-1} + \text{FIB}_{n-2} \\ \text{FIB}_1 = 1, \text{FIB}_0 = 0 \end{cases} \rightarrow \begin{cases} T(n) = 1 + T(n-1) + T(n-2) \\ T(0) = 1, T(1) = 1 \end{cases}$$

ESISTONO ANCHE:

$$(a+b):a = b$$

$$\rightarrow \text{se } \varphi = \frac{a}{b}$$

$$\rightarrow \frac{\varphi+1}{\varphi} = \varphi \rightarrow \varphi+1 = \frac{1+\sqrt{5}}{2}$$

$$\rightarrow \text{FIB}_n = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

→ POSSO APPROSSIMARE  $T(n-2) + T(n-1) \rightarrow$  APPR. CONSERVATIVA

$$\rightarrow T(n) = 1 + 2T(n-1) \quad \text{APPROXIMAZIONE}$$

$$\rightarrow T(n) = 1 + 2 + 4 + 2^3 T(n-3) = \sum_{i=0}^{n-1} 2^i = 2^n - 1$$

$$\rightarrow O(2^n)$$

$$\cdot \sum_{i=0}^k x^i = \frac{x^{k+1}-1}{x-1}$$

MCD:  $\rightarrow$  ALGORITMO DI EUCLIDE - IAMSI - Dijkstra: if ( $x > y$ )  
 $T(x, y) = 1 + T(n+1, n)$   
 $T(x, 0) = 1$

TERMINAZIONE: se  $y=0$   
 ritorna  $x$

Caso particolare se  $m+1$  è un solo 2 MM. di FIBONACCI (conservazione):

$$\begin{aligned} \rightarrow T(x, y) &= 1 + T(FIB_{m+1}, FIB_m) = 1 + \bar{T}(FIB_m, FIB_{m+1}, \dots, FIB_n) = \\ &= 1 + T(FIB_m, FIB_{m-1}) = \sum_0^{n-1} 1 = n \end{aligned}$$

$\hookrightarrow FIB_{m+1} \neq FIB_m = FIB_{m-1}$

$$\rightarrow T(x, y) = O(n)$$

$$\rightarrow m \leq y \leq m \text{ MM di FIBONACCI: } y = FIB_m = \frac{\varphi^n - \rho^n}{\sqrt{5}} \Rightarrow y = \Theta(\varphi^n)$$

$$\rightarrow m = \lfloor \log_{\varphi} y \rfloor \rightarrow \bar{T}(y) = T(x, y) = O(\log(y))$$

MAX di VETT.:

\* caso base:  $a=2, b=2 \rightarrow T(m) = 1 + 2T(\frac{m}{2})$ ,  $C(n) = \Theta(1)$ ,  $D(n) > \Theta(1)$

$0 \leq i \leq \frac{m}{2}$   
 $\rightarrow \frac{m}{2^i} = n$

$$\rightarrow T(m) = 1 + 2 + 2 \cdot 4 \cdot 2^3 \left( T\left(\frac{m}{2}\right) \right) = \sum_0^{\log_2 m} 2^i = 2 \frac{2^{\log_2 m + 1} - 1}{2 - 1} = 2 \cdot 2^{\log_2 m} - 1 = 2m - 1$$

$$\rightarrow T(m) = O(n)$$

PROBLEMA D'INTESI ( $a=4, b=2$ ):

Moltiplicazione di  $x \cdot y$  con n cifre

if( $n=1$ )

TC(1 cifra)  $x \cdot y$

else

$$\text{dove } x \text{ in 2: } x = 10^{\frac{n}{2}} x_s + x_0$$

$$\text{e } y \text{ in 2: } y = 10^{\frac{n}{2}} y_s + y_0$$

scrivere  $x_s y_s, x_s y_0, x_0 y_s, x_0 y_0$

$$\text{TC(1 cifra) } x \cdot y = 10^m x_s y_s + 10^{\frac{m}{2}} (x_s y_0 + x_0 y_s) + x_0 y_0$$

TC(n cifre)  $\Theta(n)$ :

$$\cdot \text{ mult. per } 10^k = \Theta(k)$$

$$\cdot somma di nmm su k cifre  $\Theta(n)$   $\rightarrow$  se puoi  $k=2m \rightarrow \Theta(m)$$$

multimul: come nella recursione

$$\cdot D(n) = \Theta(n), C(n) = O(n)$$

$$\begin{cases} T(m) = 4T\left(\frac{m}{2}\right) + n \\ T(1) = 1 \end{cases} \rightarrow \bar{T}(m) = n + 4 \left(\frac{m}{2}\right) + 4^2 \left(\frac{m}{2^2}\right) + 4^3 \left(\frac{m}{2^3}\right)$$

$$\rightarrow T(n) = \sum_0^{4^{n/2}} 4^i \cdot \frac{n}{2^i} = \sum_0^{4^{n/2}} 2^i \cdot n = n \sum_0^{4^{n/2}} 2^i = n(2^{n/2} - 1) = 2n^2 - n$$

$$= O(n^2)$$

$$+ \sum_0^{4^{n/2}} 2^i = \frac{(2^{4^{n/2}+1} - 1)}{2-1} = \frac{(2 \cdot n \cdot 1)}{1} = 2n - 1$$

• CON Koeffizienten / DF MNV: zulässige OT L. Menge  $\rightarrow 3$

$$x_s^* x_d + x_d^* x_s = x_c^* r_s + x_d^* r_d \sim (x_c - x_d)^* (r_s - r_d)$$

$$\rightarrow \text{zu. min. Kosten: } \begin{cases} T(n) = 3T\left(\frac{n}{2}\right) + n \\ T(1) = 1 \end{cases}$$

$$\rightarrow T(n) = \sum_0^{4^{n/2}} 3^i \cdot \frac{n}{2^i} = n \sum_0^{4^{n/2}} \left(\frac{3}{2}\right)^i = n \left(\frac{\frac{3}{2}^{4^{n/2}+1} - 1}{\frac{3}{2} - 1}\right) =$$

$$= 2n \left(\left(\frac{3}{2}\right)^{\frac{4^{n/2}}{2}} - 1\right) = 2n \left(\frac{3}{2}^{\frac{4^{n/2}}{2}} - 1\right) \stackrel{*}{=} 3^{\frac{\log_2 n}{2}} = n^{\log_2 3}$$

$$= 2n \left(\frac{3^{\frac{\log_2 n}{2}}}{n} - 1\right) = 3n^{\frac{\log_2 3}{2}} \sim 2n = O(n^{\log_2 3})$$

• DETERMINANTE ( $Q = n$ ,  $k_i = 2n-1$ ):

• Ausrechnung der Laplace  $\sum_0^m (-1)^{i+j} M[i][j] \cdot \det(M_{minore})$   
( $n \times n$ )

• if ( $n=2$ )

    create  $\det()$  si  $2 \times 2$

    recuse ( $n > 2$ )

$$tmp = M[0][j]$$

    create recus.  $\det(M_{minore[i,j]})$

    sum = sum +  $M[0][k] \times \text{pow}(-1, k) \times \det(M_{minore[i,k]})$



$$\rightarrow T(n) = O(n!)$$

$\rightarrow$  NO D.M.

Ricerca binaria ricorsiva:  $(a=1, b=2)$

- CONFRONTO  $K$  CON ELEMENTO CENTRALE DI  $V[ ]$

→ SE C'È UN OGGETTO NEL VETTORE

$\text{if } (K \leq V[m])$  PROSEGUE RICORSO SU SOLO METà SX  
 $\text{else } (K > V[m])$  " " " " DX

{ COND. TERM.  
 $V[m] = K$ }

- ANALISI DI COMPLESSITÀ:

$$D(n) = \Theta(1), C(n) = \Theta(1), a=1, b=2$$

$$\rightarrow \begin{cases} T(n) = 1 + T(\frac{n}{2}) \\ T(1) = 1 \end{cases} \rightarrow T(n) = \sum_{i=0}^{\log_2 n} 1 = 1 + \log_2 n = D(\log n)$$

Stampa in ordine inverso:

- se ricorsiva FINO a cui il char non è "10"  
 $\rightarrow$  return, torna a stampare il char prima

- ANALISI DI COMPLESSITÀ:

$$D(n) = \Theta(1), C(n) = \Theta(1), a=1, k_i=1$$

$$\rightarrow \begin{cases} T(n) = 1 + T(n-1) \\ T(1) = 1 \end{cases} \rightarrow \sum_{i=1}^{n-1} 1 = 1 + n-1 = n \rightarrow D(n)$$

(skip 125 ÷ 202)

Tower di Hanói:

SPOSTARE  $n$  DISCHI NEGLI POSIZIONE PRIMA CON  $K$  PROBLEMI / SPOSTARE OGNI DISCO PIÙ ALTO → OLD PLACE

RICORSIONE:  
 $n \cdot 1$  DISCHI IN POS. DI DEPOSITO  
 DISCO FUORI IN POS. 2  
 $n \cdot 1$  DISCHI IN POS. 2

- ANALISI DI COMPLESSITÀ:

→ COPIE DISCHI  
 $z$  volte per ogni

$$D(n) = \Theta(1), C(n) = \Theta(1), a=2, k_i=1$$

$$\rightarrow \begin{cases} T(n) = 1 + 2T(n-1) \\ T(1) = 1 \end{cases} \rightarrow 1+2+2+T(n-3) \rightarrow T(n) = \sum_{i=0}^{n-1} 2^i = \frac{2^{n-1+1}-1}{2-1} = 2^n - 1 \rightarrow D(2^n)$$

• 12 RICORSO :  $(a=2, b=2)$

→ ricorsivamente risolto

$(l, m, h-1)$  : TACCA  $\eta_1$  SX

mettere  $(m, h)$  : TACCA CENTRALE

$(m, h, l-1)$  : TACCA DI DX

$(l, m, h)$

• COND. TERM.:  $h = 0$

$$T\left(\frac{n}{2}\right) = 1 + 2 \bar{T}\left(\frac{n}{4}\right)$$

$$T\left(\frac{n}{4}\right) = 1 + 2 \bar{T}\left(\frac{n}{8}\right)$$

ANALISI DI COMPLESSITÀ:

$$D(n) = \Theta(1), C(n) = \Theta(1), a=2, b=2$$

$$\begin{cases} T(n) = 1 + 2 T\left(\frac{n}{2}\right) \\ T(1) = 1 \end{cases} \rightarrow T(n) = 1 + 2 + 4 + 8 T\left(\frac{n}{8}\right) = \sum_{i=0}^{\log_2 n} 2^i = \underbrace{2^{n+1}}_{1} - 1 = 2 \cdot n - 1 = \Theta(n)$$

• MECCANISMO WESTONE  $\downarrow$  RECORSIVE:

• LO STACK WESTONE CHIUSO E RETRO NELL'ORDINE DI

→ È UN DATO ADT, OPERAZIONI: • PUSH: INSERIMENTO IN CIMA  
• POP: ESTRAZIONE E CONCERNENTE IN CIMA } LIFO

→ STACK FRAME: STRUTTURA ORI CON: PASSAGGI MOLTIPI DI  $\downarrow$ , VAR. LOCALI, INITIALIZED A UN ROMBO, INITIALIZING DI  $\downarrow$  PER OGNI.

• C'È UNA QUANTITÀ DI NUMERO DI DATI SUO STACK, SE SUPERATA → STACK OVERFLOW

• STACK POINTER SP: CONSEGNA I'INITIALIZED DEL PRIMO STACK FRAME DISPONIBILE

•  $\downarrow$  NON-TAIL RECORSIVE: FASE DI DISCESA: RICORSIONE, RICORSITA: CONCETTO

$\hookrightarrow$  C'È BISOGNO DI UNO STACK-FRAME

•  $\downarrow$  TAIL-RECORSIVE: L'UNICA OP. È LA RECORSIONE → ACCIUMERE GIÙ IL RETORNO DELLA VERSOGE  
 $\rightarrow$  CONCETTO IN FASE DI DISCESA

$\hookrightarrow$  EFFICIENTI, NO STACK (NO STACK-OVERFLOW)

•  $\downarrow$  TAIL-RECORSIVE PUÒ ESSERE TRASFORMATO IN ITERATIVA, SENZA USE DI STACK

→ SE  $\downarrow$  NON-TAIL RECORSIVE, POSSONO TRASFORMATI IN ITERATIVA → WESTONE DANNEGGIA NUOVO STACK

• 06

## GLI ORDINAMENTI RICORSIVI

- MERGE SORT:  $O(n \log n)$



→ PRIMAVERA DIVIDE E' IMPORTANTE: SVOLGONO IL VETTORE IN 2 PARTI

AL CASO TERMINALE  $l = r$  o  $l > r$

→ ORDINA IN FASE DI RECORSANZA

- $\text{merge}()$ : UNI I 2 ESTREMETRI DEI VETTORI, ORDINA CONFRONTO

SCORDA I 2 SORTEMENTI: SX:  $i = l$ , DX:  $j = q + 1$

→ CONFRONTO IN SEZIONE MASSIMA VETTORE, SVIZ. IL MINORE E' MIGLIORE L'INDICE

Algoritmo	In loco	Stabile	Caso peggiore
Bubble sort	sì	sì	$O(n^2)$
Selection sort	sì	no	$O(n^2)$
Insertion sort	sì	sì	$O(n^2)$
Shellsort	sì	no	dipende
Mergesort	no	sì	$O(n \log n)$
Quicksort	sì	no	$O(n^2)$
Counting sort	no	sì	$O(n)$

• NON IN LOC: USO VETT. ANTERIORI

• STABILE: •  $\text{merge}()$  PRESERVA ORDE SOTTO VETT. SX IN CASO GIÀ CHIUDI VETTORE

- ANALISI DI COMPLICATITÀ:

$$D(n) = \Theta(1), C(n) = \Theta(n), a = 2, b = 2 \rightarrow \text{caso 2 della ricorrenza}$$

$$\rightarrow \text{ES. CASO BASE: } T(n) = 2T\left(\frac{n}{2}\right) + m$$

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

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4}$$

$$\begin{aligned} \rightarrow T(n) &= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n \\ &= 2\left(2\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + \frac{n}{2}\right) + n \\ &\vdots \end{aligned}$$

$$\therefore \sum_{0 \leq i \leq \log_2 n} n = n \sum_{i=1}^{\infty} 1 = n (\log_2 n + 1) = n + n \log n = O(n \log n)$$

## BOTTOM-UP MERGE SORT

→ VERSIONE NON RECURSIVA: POSSIBILE FARLO SENZA VETTORI INTERMEDI

- BOTTUM SORT:  $O(n^2)$  ( $O(n \log n)$  CASO MEDIO)

• SI EFFETTUANO UN DIVISIONE DIVISE, IC COMBINEZIONI IN POSIZIONE FINO ALLA UNA CONFRONTO FINALE

→ CICLO INTERNO SI FONO SOVRAPPONE  $i \rightarrow e \ j \leftarrow \rightarrow z$  dove ( $\Rightarrow A[i] > x \ \& \ A[j] < x$ )

- IN LOC, NON STABILE (permutazione)

→ CASO MIGLIOR: 2 VETTORI GIÀ STABILI GLI.

SE DIVIDONO  
IN MODO  
OPPURE J  
 $n-1 / 1$

$$\rightarrow \text{EFFICIENZA POSSIBILE SECONDO QUESTA DEDUZIONE VETTORI DI DIM. 1 E N-1}$$

$$\cdot \text{CASO PEGGIOR: } T(n) = T(n-1) + n \rightarrow = \frac{n(n-1)}{2} = O(n^2) \cdot \text{CASO MIGLIOR: } T(n) = 2T\left(\frac{n}{2}\right) + n \rightarrow O(n \log n)$$

07

## PROBLEMI DI RICERCA E OTIMIZZAZIONE

- RIPASSO CALCOLO COMBINATORIO:

- PRINCIPIO DI MOLTIPLICAZIONE:

OGGETTO  $X_i$  CON  $p_i$  SCELTE  $\rightarrow$  SCELTE TOT:  $p_1 \cdot p_2 \cdot \dots \cdot p_k$

→  $0 \leq k \leq m$

• DISPOSIZIONI SEMPLICI:  $D_{m,k} = \frac{m!}{(m-k)!} \rightarrow$  CONTA L'ORDINE, NO RIPETIZIONI

• DISP. CON RIPETIZIONI:  $D'_m,k = \eta^k \rightarrow$  CONTA L'ORDINE, SI RIPETIZIONI

• PERMUTAZIONI SEMPLICI:  $P_m = D_{m,m} = m! \rightarrow$  CONTA L'ORDINE, NO RIPETIZIONI

• PERMUTAZ. CON RIPETIZIONI:  $P_m^{(\alpha, \beta, \dots)} = \frac{m!}{\alpha! \beta! \dots} \rightarrow$  CONTA L'ORDINE, SI RIPETIZIONI

→  $0 \leq k \leq m$

• COMBINAZIONI SEMPLICI:  $C_{m,k} = \binom{m}{k} = \frac{m!}{k!(m-k)!} \rightarrow$  NON CONTA L'ORDINE, NO RIPETIZIONI

$$\binom{m+k-\gamma}{k}$$

• COMBINAZ. CON RIPETIZIONI:  $C'_{m,k} = \frac{(m+k-\gamma)!}{k!(m-\gamma)!} \rightarrow$  NON CONTA L'ORDINE, SI RIPETIZIONI

• POWER SET (o INSIEME DEI SOTTOinsiemi): INSIEME DEI SOTTOinsiemi

• PARTIZIONI DI UN INSIEME:  $n^0$  È UNA PA' MOLTIPLICATORE DI PARTIZIONE:  $B_{n+1} = \sum_{k=0}^n \binom{n}{k} \cdot B_k$

- PRINCIPIO DI MOLTIPLICAZIONE:

```
int princ_molt(int pos, Livello *val, int *sol, int n, int count) {
    int i;
    if (pos >= n) { → end, val[i];
        for (i=0; i < n; i++)
            printf("%d ", sol[i]);
        printf("\n");
        return count; ← POSSIBILE
    }
    for (i=0; i < n; i++) {
        if (mark[i] == 0) {
            mark[i] = 1; → some is marked[i] == 0
            sol[pos] = val[i];
            count = princ_molt(pos+1, val, sol, mark, n, count);
            mark[i] = 0; ← BACKTRACK
        }
    }
    return count; ← REGISTRA IL NUM. DI SOLUZ.
    TERMINATE
}
```

(BACKTRACK)

↳ PUZZLE DI CHIUSI

- $D_{m,k}$ :

↳ MARK[]  
↳ BACKTRACK

```
int disp(int pos,int *val,int *sol,int n,int k,int count){
    int i;
    if (pos >= k){ → some is marked[i] == 0
        for (i=0; i < k; i++)
            printf("%d ", sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=0; i < n; i++) {
        if (mark[i] == 0) {
            mark[i] = 1; → some is marked[i] == 0
            sol[pos] = val[i];
            count = disp(pos+1, val, sol, mark, n, k, count);
            mark[i] = 0; ← BACKTRACK
        }
    }
    return count;
}
```

$\downarrow m=k$

- $P_m$ :

↳ CORRE  $D_{m,k}$   
↳  $m=k$

```
int perm(int pos,int *val,int *sol,int *mark,
        int n, int k,int count){
    int i;
    if (pos >= n) { → terminazione
        for (i=0; i < n; i++)
            printf("%d ", sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=0; i < n; i++) {
        if (mark[i] == 0) {
            mark[i] = 1; → some is marked[i] == 0
            sol[pos] = val[i];
            count = perm(pos+1, val, sol, mark, n, k, count);
            mark[i] = 0; ← BACKTRACK
        }
    }
    return count;
}
```

- $D'_m,k$ :

↳ CORRE  $D_{m,k}$   
↳ NO MARK[]

```
int disp_rip(int pos,int *val,int *sol,int n,int k,int count){
    int i;
    if (pos >= k) {
        for (i=0; i < k; i++)
            printf("%d ", sol[i]);
        printf("\n");
        return count+1;
    }
    for (i = 0; i < n; i++) {
        sol[pos] = val[i];
        count = disp_rip(pos+1, val, sol, n, k, count);
    }
    return count;
}
```

↳ SIMILI

- $P_m^{(0,1,0,..)}$ :

↳ MARK[] CONTINUE  
IL # DI OGNI  
V. CERTA  
ORDINA  
SI SIMILA A  $P_m$

```
int perm_r(int pos,int *val,int *sol,int n,int dist,int count) {
    int i, j;
    if (pos >= n) { → some constant / sufficient algorithm
        int mark, int n, int n_dist, int count;
        for (i=0; i < n; i++)
            printf("%d ", sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=0; i < n; i++) {
        if (mark[i] > 0) {
            mark[i] = -1;
            sol[pos] = val[i];
            count = perm_r(pos+1,dist_val,sol,mark,n,n_dist,count);
            mark[i]++;
        }
    }
    return count;
}
```

↳ Ogni → 0,1,0,...

- $C_{m,k}$ :

↳ USED START FOR  
INITIALIZATION, SE CHE APPO  
SI VERA INIZIALE LA  
RISORVA  
↳ NO MARK[]

```
int comb(int pos, int *val, int *sol, int n, int k,
        int start, int count) {
    int i, j;
    if (pos >= k) { → terminazione
        for (i=0; i < k; i++)
            printf("%d ", sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=start; i < n; i++) {
        sol[pos] = val[i];
        count = comb(pos+1, val, sol, n, k, i+1, count);
    }
    return count;
}
```

- $C'_{m,k}$ :

↳ START IT  
SOLO A FINI  
RISORVA  
↳ NO MARK[]

```
: comb_r(int pos,int *val,int *sol,int n,int k,
        int start, int count) {
    int i, j;
    if (pos >= k) {
        for (i=0; i < k; i++)
            printf("%d ", sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=start; i < n; i++) {
        sol[pos] = val[i];
        count = comb_r(pos+1, val, sol, n, k, start, count);
        start++;
    }
    return count;
}
```

## • INSERIRE DUEI PERM (POWERSET)

### • CON RICORSIONE :

```
int powerset(int pos, int *val, int *sol, int k, int start, int count) {
    int i;
    if (start >= k) {
        for (i = 0; i < pos; i++)
            printf("%d ", sol[i]);
        printf("\n");
        return count+1;
    }
    for (i = start; i < k; i++) {
        sol[pos] = val[i];
        count = powerset(pos+1, val, sol, k, i+1, count);
    }
    count = powerset(pos, val, sol, k, k, count);
    return count;
}
```

### • CON D<sub>M</sub>K :

```
int powerset(int pos, int *val, int *sol, int k, int count) {
    int j;
    if (pos >= k) {
        printf("{ ");
        for (j=0; j<k; j++)
            if (sol[j]==0)
                printf("%d \t", val[j]);
            printf("} \n");
        return count+1;
    }
    sol[pos] = 0;
    count = powerset(pos+1, val, sol, k, count);
    sol[pos] = 1;
    count = powerset(pos+1, val, sol, k, count);
    return count;
}
```

### • CON C<sub>M</sub>K E WRAPPER :

```
int powerset(int* val, int k, int* sol) {
    int count = 0, j;
    printf("{ }\n");
    count++;
    for(j = 1; j <= k; j++){
        count += powerset_r(val, k, sol, j, 0, 0);
    }
    return count;
}
```

→ POWERSET PER J RECURSION

```
int powerset_r(int* val, int k, int* sol, int j, int pos, int start){
    int count = 0, i;
    if (pos == j){
        printf("{ ");
        for (i = 0; i < j; i++)
            printf("%d ", sol[i]);
        printf("} }\n");
        return 1;
    }
    for (i = start; i < k; i++) {
        sol[pos] = val[i];
        count += powerset_r(val, k, sol, n, pos+1, i+1);
    }
    return count;
}
```

### • PARTIZIONAMENTO DI UN INSIEME :

```
void disp_rripet(int pos, int *val, int *sol, int n, int k) {
    int j, t, ok=1, *occ;
    occ = calloc(k, sizeof(int));
    for (j=0; j<k; j++)
        occ[sol[j]]++;
    i=0;
    while ((i < k) && ok) {
        if (occ[i]==0) ok = 0;
        i++;
    }
    free(occ);
    if (ok == 0) return;
    else { /*STAMPA SOLUZIONE */
    }
    for (i = 0; i < k; i++) {
        sol[pos] = i;
        disp_rripet(pos+1, val, sol, n, k);
    }
}
```

• OCC [i]: CONTIENE LE OCCORRENZE DEL VALORE i ( $\text{occ}[2] = 3$ )  
 ↳ contiene le occorse nel blocco i

↳ AL OKOKO Z  
 CI SONO 3 BLOCCHI

• SOL [j]: CONTIENE IL BLOCCO i A CUI APPARTIENE j ( $\text{sol}[4] = 2$ )  
 ↳ è appartenente al blocco 2

### • ALGORITMO DI ER:

CALCOLO DI TUTTE LE PARTIZIONI DI N DIVISI IN K BLOCCHI

```
void SP_rec(int n, int m, int pos, int *sol, int *val) {
    int i, j;
    if (pos >= n) {
        printf("partizione in %d blocchi: ", m); COND. TERM.
        for (i=0; i<m; i++)
            if (sol[i]==i)
                printf("%d ", val[i]);
        printf("\n");
        return;
    }
    for (i=0; i<m; i++) {
        sol[pos] = i;
        SP_rec(n, m, pos+1, sol, val); → ric. su omitti
    }
    sol[pos] = m;
    SP_rec(n, m+1, pos+1, sol, val); → ric. su omitti + blocco
}
```

• SE VALORE DI PARTIZIONE IN ESISTENTE K BLOCCHI :

• FILTRA NEGLI STESSI SOL [ ]: if ( $m = i$ )

• PASSA K CORSE PARTIZIONI

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

$$n = \frac{1}{2} 2^i$$

$$2n = 2^i$$

$$i = \log_2 2n = \log_2 2 + \log_2 n$$

• 09

## LA PROGRAMMAZIONE DINAMICA

PROCEDIMENTO ↗ BOTTOM UP: COSTRUISSA LA SOLUZIONE

VERIFICA APPLICABILITÀ: ↗ TOP - DOWN: SE VINCIBILE CON MEMORIZZAZIONE, E CON RICORSIONE

DEF. RECURRENCE

RECURSIVE

SOLUZIONE

RECURRENCE

LA COSTR.

SOL. OPTIMA

RECURRENCE

COMPLESSITÀ:  $T(n) = \Theta(n) = \Theta(n^{\text{numero di casi di ricorsione sol. ottimi}})$

MEMORIZZAZIONE: RICORDAZIONE SOLUZIONI FRUITI DURANTE LA RICORSIONE

• 10

## IL PARADIGMA GREEDY

↗ SOLO LOCALMENTE

COSTRUTTONE DI UNA SOLUZIONE OTTIMA, TRAMITE FUNZIONE CHE CONSIDERA BLOCCHE IN PASSO AD UN INDICE DI APPETIBILITÀ ↗ Ogni blocco

→ LA SOLUZIONE NON È DETTO CHE SIA IN TUTTI I CASI OTTIMA

• CONCI ↗ LUNGHEZZA FISSA, SIMBOLI (SO) FREQUENTI

↗ GRADO DI INFERIORE (SOPRAVINTENZA)

• CONCI ↗ LUNGHEZZA VARIABILE: DIFFERENZA CODIFICATI, RISPARMIO DI SPATI

• CODICI PREDEFINITI: NESSUN CODICE È PREFISSO DI UN ALTRO CODICE

• CODIFICA: TRASFERIRE IL CODICE, UNISSO I BLOCCHI TRA DI LORO UNA VOLTA CONFIRMATO

• DECODIFICA: PER RICORDARE C'È UNO BINARIO PER TRASFERIRE IL CODICE ORIGINALE

## COSTRUZIONE DEL 'ALBERO BINARIO':

• UTILIZZO UNA CODA A PROPRITÀ, ORDINATA PER FREQUENZA CRESCENTE

↗ PASSO i :

• ESTRAZIONE I 2 SIMBOLI (O 2000 ACCESI) CON MINOR FREQUENZA CON:

- I 2 NONI CON I SIMBOLI (O 1000), UNO PER IL RAMO D E UNO PER E
- FREQUENZA: SOMMA DEGLI ACCESI PRECEDENTI

• INSERISCI NELLA CODA A PROPRITÀ IL SIMBOLI (O 4000), IN BASE ALLE ↗

→ FINIRSI SE LA CODA È Vuota

• 11

## TABELLE DI SIMBOLI

UTORI/USTR  
↗

• VERSIONI DEL' ADT: TABELLE AD ACCESSO DIRITTO / STRUTTURE LINELLI O AD ALBERO / DI HASH

• 12

## ALBERI BINARI DI RICERCA BST :

CHIAVE  
RADICALE  
( $\text{h} = \sum_{i=1}^n m_i$ )

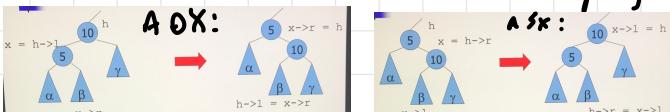
$\forall \text{ nodo} : \text{ SIFORZIONE } \text{SX} \text{ e } \text{DX} : \begin{cases} \text{sx} : \text{KEY}[\text{v}] < \text{KEY}[\text{x}] \\ \text{dx} : \text{KEY}[\text{v}] > \text{KEY}[\text{x}] \end{cases}$



- $\text{SEARCH}(\cdot)$  : ricorsione a partire da ROOT :  $\forall \text{nodo} \begin{cases} \text{SEARCH SU SX}, \text{SE KEY}[\text{v}] < \text{KEY}[\text{x}] \\ \text{SEARCH SU DX}, \text{SE KEY}[\text{v}] > \text{KEY}[\text{x}] \end{cases}$
- CERCA IL MIN : ricorsione su SX / CERCA IL MAX : ricorsione su DX
- $\text{INSERT}(\cdot)$  : inserisco facendo un confronto tra KEY[], finisce quando arriva per la prima volta alla radice

•  $\text{SEARCH}(\cdot)$  : EFFETTUO ITEM IN ORDINE INCRESCENTE :  $O(\log n) \leq T(n) \leq O(n)$

•  $\text{ROTATORI}(\cdot)$ .



SE X  
A SX  
G + DX X

• INSERIMENTO IN RADICE : inserisco ITEM come根, EFFETTUO ROTAZIONE A DX E SX PER PASSARE A RADICE

• SUCCESSORE() : cerca il minimo subito dopo h :  $\begin{cases} \text{RIGHT}(h) : \min(\text{RIGHT}(h)) \\ \text{RIGHT}(h) : \text{ITEM ANTERIORI DI } h \\ \text{RIGHT}(h) : \text{ITEM SX DI ANTERIORI DI } h \end{cases}$

• PREDECESSORE() : ricorreva l'accedi, con ITEM CON CHIAVI PIÙ PICCOLE IN REPUBBLICA → TRACCI IN CHIAVE E VERSO IL ROT. A DX E SX

✓ CIOÈ IL PREDECESSORE  
NON PARTE DAL PREDECESSORE

• REVERSE() : cerca il succ() di ITEM, APERTURA ROOT, TRAMITE PREDECESSORE() E RICORRENZA

• SELECT() : TRACCI L'h-ESIMO (cerca in base alle possibili)  $\begin{cases} t=r : \text{RADICE TROVATA} \\ t>r : \text{NONS. SU SX} \\ t < r : \dots \text{DX, CASO IC } h-t-1 \end{cases}$

$$\rightarrow r = (h-N+1)/2 - 1$$

• BILANCIAZIONE() : PREMIANA RISPELTO AL ITEM INSERITO E BALANCE NEI SOTTOALBERI

REVERSE  
ROTATION

→ ACCIUNO INFO A BST POSSO FARLO IN OPERAZIONI PIÙ EFFICACE

POST-FIX

↳ as. 1-BST

• EXPRESSION: BST AS EXP: POST-order: RPN

PRE-ORDER: NOTAZIONE PREPESA

• 15 ADT GRAFO

↳ INSERT(): GUARDA LOWER[i]

SEARCH(): CONFRONTO KEY CON MAX AND MIN.  
PRINT CON  $\sim$  ROOT

FIND SX :

• CICLO DI HAMILTON: CAMMINO SERPENTE CHE VISITA OGNI VERGA UNA VOLTA SOLO

• STRUTTURE PER IL GRAFO:

↳ INTERNA o ESTERNA

• TABECCA DI SITZBOEL, PER LA CORRISPONDENZA NODI-INDICI DE NODI

•  $|V|$ : SE NON ESPLICATO, SOVRASSUNTO CON:  $\geq^* |V| \cdot |\text{ARCHI}|$  ITEM

•  $|F|$ : EFFICIENTE SE UNICO ORBIO:  $S(n) = O(|V|^2)$

$\rightarrow S(n) = O(|V||E|)$ ,  $\rightarrow$  UTILE PER

• MULTRARIO: VERSOGLI CONNESSIONI DA PIÙ DI UN ARCO

- UNIRAZIONI 0) DISSETI ↗ CASUAL  
CON probabilità  $p = 2|E| / (|V| * (|V|-1))$
  - CAMMINO SEMPLICE DA  $V \in W$ : ricorsione per numero dei nodi adiacenti non visitati  
, se non ci sono adiacenze i nodi visitati  $T(n) = O(|V| + |E|)$
  - CAMMINO DI HAMILTON: come algoritmo del cammino semplice, ma controlla che le connesioni siano  $|V|-1$   
↳ VERTICI ↗ ANCHE UNA SEMPLICE
  - CAMMINO DI EULEO: cammino che connette 2 vertici attraversando ogni arco  
↳ ANARCHY ↗ CICLO  $\Leftrightarrow$  (CONNESSO & Tutte le aree ha grado pari)  
↳ CAMMINO  $\Leftarrow$  (CONNESSO &  $\exists$  2 vertici di grado dispari)

## • 16 ALGORITMI DI VISITA DEI GRAFI

- time
- pre[], post [y]
- set [] aggiunt. dei nodi

IN PROFOUNDITÀ : ④ DFS (Depth-First Search)

'IN AMPUSZBA : ②  $\beta FS$  (zeroon · first wəzəʊn) ~> FOR UNA PQ

↳ si può unire due periferiche  
in uno stesso percorso

$\sim O(|V| + |F_D|) \rightarrow$  CON LISTA ADJACENCIA

7. prec[i]: tempo di scoperta, post[i]: tempo di fine elaborazione  
 > area orientati: Tree, Backtrack, Forward, Cross

→ WEIßER FORESTA DI ALBERI

•  $\text{set } [i] := \text{SEARCH } \text{LL} \text{ PROBE } \text{REG } \text{ VERTUE } |$

IL VERSOLO SU CUI INIZIA B  
È CHI STAM SCOPERA (PRE[1]:=1)  
MA NON È STAM • ARCH  
COMPLETATI  
(POST[1]:=1) ↗

## • Arch1 :

• T: APEN DEU' ALPACOURANTE LA VISITA IN PROPOSALTA

5 UN ARCO B SE  
post[i] = -1 DEC  
VERSE SCADUTA

PER  
WIFI  
ORIENTAM

- B: CONNETTE UN VERTICE AD UN SO ANTENNA:  $\text{post}[i] \leftarrow \text{post}[i] > \text{post}[j]$   
 $\text{post}[i] = -1$ , vorrei sia sempre
- F: CONNETTE UN VERTICE A UN SW PRECEDENTE:  $\text{pre}[i] < \text{pre}[j]$   
 $\hookrightarrow$  chi scorrazza
- C: NELLA RETE DISSONANTE  $(\text{pre}[i] > \text{pre}[j]) \rightarrow$  MARCA IN DIREZIONE 2  
MOLTI CHE NON SONO NEGLIGIBILI

DAPM  $\text{dfs}()$  e  $\text{dfsR}()$   $\sim T(n) = \Theta(|E| + |V|)$ , con uscita in ALIASING

(2) If wolo wosue, constans i svol aram & netwo vj codo kusigibzilne in cosa  $Q \sim T(n) = O(|V|^3)$   $\sim$  con matrix si nolame

- ACICLICITÀ: ovante DFG non si incontrano nodi  $B$

UN GRAPPO  
È CONNESSO  
 $\Leftrightarrow$  C'È UNA  
SOLA CC.

- COMPONENTI CONNESSE: array [] che memorizza le componenti connesse

(1) PIÙ DI  
ARCO CONNESSO PONTE (2)

- CONNESSIONE: verifica se troviamo un nodo o arco il quale dirigere compreso

NON SONO NEI VERTICI IN PROGRESSIONE

- (1) ROOT DI  $G$  è TRUE  $\Leftrightarrow$  HA ALMENO 2 FIGLI

- Verso sinistra è TRUE  $\Rightarrow$  UN FIGLIO S /  $\not\exists$  ARCO  $B$  O S / FIGLIO DISCONNESSO A UN SOLO ANTEPRECEDENTE

- (2)  $B$ : NON LO È MAI

$T \Leftrightarrow \not\exists$  ARCO  $B$  DA UN DISCONNESSO DI  $V$  AD UN GRANDE DI  $V$  ( $v, w$ )

• ALGORITMO: RICHIESTA 1 SU 1 MI SEMPRE E' VERDIERE SE  $G$  È CONNESSO

- DAG (Directed Acyclic Graph):

LEADER ORIENTATO ACCIAIO: NON SI PUÒ TORNARE INDIFERENTEMENTE

$\rightarrow$  POSSONO PIÙ ARCOGNI DI SCHEMATICHE (RISOLVERE COMBINAZIONI CON VERSO DI PRECEDENZA)

NON SOLENTE: in-degree = 0

PORREZZO: out-degree = 0

DAG.POP:  $\downarrow$  POST[] • ORDINAMENTO TOPOLOGICO: ORDINA IL DAG. IN ORDINE "TORONTO"

DAG.TOF.INN.:  $\nearrow$  POST[]  $\downarrow$  CON LA RICORSIONE STAMPO L'ORDINE INVERSO

- COMPONENTI FORTEMENTE CONNESSE:

- ALGORITMO DI KOSARAJU:

$O(|V| * |E|)$

```
void SCCDFSGraph(G, int v, int *scce, int *time0, int timel, int *post) {
    link t;
    scce[v] = timel;
    for (t = G->adj[v]; t != G->z; t = t->next) {
        if (scce[t->v] == -1) {
            SCCDFSGraph(G, t->v, scce, timel, post);
            post[*(time0)] = v;
        }
    }
}

void SCCESSGraph(G) {
    int v, time0 = 0, *scce, *scceR, *postG, *postR;
    G = GRAPHreverse(G);
    SCCESS(G, time0, scce, postG);
    SCCESS(G, time0 + 1, scceR, postR);
    for (v = 0; v < G->V; v++) {
        if (scce[v] < scceR[v]) {
            SCCDFSGraph(G, postR[v], scce, time0, postG);
        } else {
            SCCDFSGraph(G, postG[v], scceR, time0 + 1, postR);
        }
    }
}
```

• TRASPOSTO IL DAGO  $G \rightarrow G^T$ ; INVERTO L'ORDINE DEI FIGLII DELMI

• VISITA DFS SU  $G^T$ , SALVANDO PRE[] E POST[]

• VISITA DFS SU  $G$ , ORDINANDO COME ORDINE I POST[], IN ORDINE DECRESCENTE

$\rightarrow$  RICHO COSÌ LE EDG. FORTE CONNESSIONI

LE MIE MIE DEDICATE  
 $\rightarrow$  CHI SE SONO DEDICATE DA

• POSSO ESTRAE  $G^T$ , DOVE HAM. FORTE CONNESSIONI CONSIDERA UN NODO RAPPRESENTATO  
 $\hookrightarrow$  KERNEL DAG: SE CI POSSE UN CICLO STAMPA UN MESSAGGIO DI DFS

(USARE MEGLIO ALGORITMO)

### 13. TABELLE DI HASH

- OCCUPAZIONE SPAZIO  $O(1/k)$   $\leadsto$  numero di chiavi
- ACCESSO  $O(1)$
- f di HASH  $\leadsto$  può portare a collisioni:  $h(k_i) = h(k_j), k_i \neq k_j$
- PREZIAMENTE COLLISIONI, CONVIENE AMPLIARE le DIFFERENZE TRA CHIAVI SIMILI e SCORRERE  $h(k)$  DA K e DISTRIBUIRE IN MODO UNIFORME

- PROCEDURE: MOLTIPLICATRICE  $\rightarrow$  MODULAZIONE  $\leadsto$  MMDDI PRIMO  
 $\hookrightarrow$  scarto  $M$   $\hookrightarrow$  K.P.M  $\leadsto$  numero di eliminare i multipli di M
- SE  $K = \text{string}$ : METODI: POLINOMIALI  $\rightarrow$  HORNER  $\leadsto$  numero di stringhe minori di M

- LE COLLISIONI SONO INEVITABILI: SI POSSANO RIMUOVERE o AVOIDERE

- GESTIONE COLLISIONI:
  - $M \geq \text{num chiavi} / p \leadsto$  O.M LISTA
  - $\hookrightarrow$  con scorrere la O.M. Dopo trovarsi

```
int hash (char *v, int M){
    int h = 0, base = 128;
    for (; *v != '\0'; v++)
        h = (base * h + *v) % M;
    return h;
}
```

- LINEAR CHAINING: CREA UNA LISTA SE CI Sono COLLISIONI

- FATTORE DI CARICO  $\alpha = N/M$  /  $N$ : NUM. ELEMENTI TABELLARI  
 $M$ : O.M. TABELLA HASH

- OPEN ADDRESSING:  $\alpha < 1 \leadsto M \geq 2^k$  numeri

- UNA CELLA PUÒ CONTENERE 1 SOLO ELEMENTO

- > SE CI Sono COLLISIONI: PROBING: rientrante di loschi

- f di PROBING:

$$\hookrightarrow i = (i+1) \% M$$

- LINEAR PROBING: SE TROVA OCCUPATO, MUOVE POSIZIONE DI 1 FINO A CELLA VUOTA

$$\cdot \text{se } M = 2^k: c_1 = c_2 = \frac{1}{2}$$

- LA OPERAZIONE () È COMPLESSA (COLLISIONE PROB.).

$\hookrightarrow$  SI PUÒ RUSTARE UNA ZONE DELLA TAB.  
 DIRE CHE SI È ONUATO  $\hookrightarrow$  METODO "STATUS"  
 $\hookrightarrow$  REINSEGNARLO CONSTATR

- QUADRATIC PROBING:  $i = (h(k) + c_1 i + c_2 i^2) \% M$

- DOUBLE HASHING: SI RICAVANO  $h_2(k)$

$\hookrightarrow$  bisogna fare attenzione a un breve ciclo  
 $\hookrightarrow i = h_1(k)$

$$\begin{cases} h_2 \neq 0 \\ h_2 \% M \neq 0 \end{cases}$$

$$\rightarrow j = h_2(k) \rightarrow i = (i+j) \% M$$

## 74. CODE A PRIORITÀ e HEAP

- HEAP: ALBERO BINARIO CON

USATO TRAMITE VETTORI:

RADICE IN  
 $A[0]$

$$\text{ORDINE COMPLETO} \quad (\text{RISPARMIO DI SX E DX})$$

$$\text{KEY}(PARENTE(i)) \geq \text{KEY}(i) \rightarrow \text{CHIAVE MAX}$$

PADRE  $A[\frac{i}{2}(i-1)]$

FIGLIO SX  $A[i]$

FIGLIO DX

$A[z_{i+1}]$   $A[z_{i+2}]$

- heapify(): TRANSFORMA IN HEAP I  
 $\hookrightarrow$  ASCIENDI IN  $A[i]$  IL  
 MAX ENTRO I,  $\text{RIGHT}(i)$ ,  $\text{LEFT}(i)$

- heap build(): TRASFORMA VN ALBERO BINARIO IN HEAP  
 $\hookrightarrow O(m)$   
 $\hookrightarrow$  MAX IN  
 FIGLIO SX E DX

PROBLEMA PADRE  
 NON UNICO FIGLIO  
 E heapify, i --

- heapsort(): TRANSFORMA IL VETTORE TRAMITE heap build()  $\rightarrow$  heap build()  
 $\hookrightarrow O(n \log n)$   $\hookrightarrow$  ORDINA IL VETTORE

heapsort()  
 SINGOLARE VERSO  
 KEY SIZE  $\rightarrow$   $O(DX)$

## CODE A PRIORITÀ:

- HEAP CON CAMPO "PRIORITÀ"

$\rightarrow$  PER PADRE

- PQ INSERT()  $\rightarrow O(\log n)$ : INSERISCE IN FOGLIO, CONFRONTANDO NEW NODE FINO TUTTI PARENTI

SE KEY() NIENTE

- PQ EXTRACT MAX()

- PQ CHANGE(): MODIFICA LA PRIORITÀ DI UN ELEMENTO

RISULTATO IN POSIZIONE

$\rightarrow$  heapify SE KEY() OLTRE  
 $\hookrightarrow O(n) + O(\log n) = O(n)$

18

6.1 ALBERI RICOPRENDO MINIMO (MSI),  $T \subseteq E$ , ACQUACO, PESO

$$\text{SIA } G = (V, E) \rightarrow G' (V, T)$$

PRIM  
S

MIMMIZZA  $wt(u, v)$

LORO CHE COPRIRRE tutti i nodi

- ARCO INESISTENTE,  $wt : +\infty$ , 0, 1  $\rightarrow$  SE NON È AMMESSO PESO NEGATIVO
- SE  $G = (V, E) \rightarrow G'$  ANCHE ESATTAMENTE  $V - 1$  ARCFI
- ALGORITMO GENERICO: AGLIARBI ARCFI SICURI, FINO A CHE NON È RICOPRENDO MINIMO
  - $\rightarrow$  ARCO CHE, SE AGGIUNTO, LORO CHE COPRIRRE ancora
  - $\rightarrow$  SOLUZIONE OTIMA
- TABUN: FORMAZIONE DI UN GRADFO  $S \in V - S$ 
  - $\rightarrow$  ARCO  $(u, w)$ ; nello A PESO MINIMO TRA QUELLI CHE ATTENDONO IL TACCO
- T. ARCFI SICURI: SIA  $G = (V, E)$  NON ORIENTATO, PESATO, CONNESSO
  - $A \subseteq E, (S, V - S)$  TABUN CHE RICOPRE A

• SE  $(u, v)$  ARCO LEGATO CHE ATTENDE  $(S, V - S) \rightarrow$  È UN ARCO SICURO PER A

- ALGORITMO DI KRUSKAL CON SIMMETRIE EFF.  $T(\pi) = (|E| \ln |E|)$ 
  - $\rightarrow$  SIMMETRIE
  - $\rightarrow$  id[] =  $\pi[\cdot]$

• CONSIDERA UNA FORESTA DI ALBERI (IMPLEMENTA VERICI SIMBOLI)

• ORDINA GLI ARCFI PER PESO INCREASING  $u[] \rightarrow \text{sort}()$

• ITERAZIONE: SELEZIONA UN ARCO SICURO DI PESO MINIMO CHE UNISCE 2 ALBERI
 

- $\rightarrow$  TERMINA QUANDO HO CONSIDERATO tutti i VERICI

id[]: rappresenta gli alberi,  $S \geq [\cdot]$ : UNIONE DEGLI ALBERI

- ALGORITMO DI PRIMM:  $\rightarrow$  ASSUNTA DI ARCO:  $+\infty$ , 0 [ $|E| \ln |V|$ )

• USO I SOGLIALI S ( $\emptyset$  INIZIANO)  $\rightarrow$   $V - S$  (PIÙ MINIMO)

• ITERAZIONE: CONSIDERA ARCO CHE ATTENDE IL TACCO A PESO MINIMO

• ALLARNO  $st[\cdot]$ : VETTORE DEI PADRI DEL NODO IN S

$wt[\cdot]$ : PESO MINIMO TRA 2 VERICI  $\left\{ \begin{array}{l} S \neq \emptyset \\ V - S \neq S \end{array} \right.$

$fr[i]$ : NODO DI S  $\rightarrow$  GIÀ CONSIDERATO

• min: MINIMIZZA VERICI, V-S PIÙ PICCOLO, V. DI S

• TERMINAZIONE: QUANDO HO CONSIDERATO tutti i VERICI

```
void mst(Graph G, int *st, int *wt) {
    int v, w, min, *fr = malloc(G->V*sizeof(int));
    for (v=0; v < G->V; v++) {
        st[v] = -1; fr[v] = v; wt[v] = maxWT;
    }
    st[0] = 0; wt[0] = 0; wt[G->V] = maxWT;
    for (min = 0; min != G->V; ) {
        v = min; st[min] = fr[min];
        for (w = 0, min = G->V; w < G->V; w++)
            if (G->adj[v][w] < wt[w]) {
                if (G->adj[v][w] < wt[min])
                    if (wt[w] < wt[min]) min = w;
                }
            if (wt[w] < wt[min]) min = w;
        }
    }
}
```

• USO IL VERICO, TRA GLI 2 PESI MINIMI

ARCO LIBERO  
TRA GLI 2 PESI MINIMI  
&  
PESO MINIMO

19. CAMMINI MINIMI  $\rightarrow$   $w(p)$ ,  $S(s, v)$  CAMMINO MINIMO

• PESO DI UN CAMMINO:  $\sum \text{wt}(v_{i-1}, v_i)$

• ALGORITMI PER CAMMINI MINIMI TRA SOGLIALE  $\leq$  A GVI  $V$ : DIJKSTRA  
RELLMAN-FORD

•  $\exists$   $v, u$   $w(u, v) < 0$ , ma  $\exists$  CICLO: DIJKSTRA: SOL. OTTIMA NON VERA  
BELLMAN-FORD: SOL. OTTIMA GARANTITA

•  $\exists$   $\exists$  CICLO CON  $\text{wt}(\cdot) \leq 0$  (PSILOFURSI). DIJKSTRA: RISULTATO SPERATO SICURO  
BELLMAN-FORD: RISULTATO CICLO  $< 0$

• I CAMMINI NERI NON POSSONO AVERE CICLI A PESO NEGATIVO O POSITIVO

$\hookrightarrow$  I CAMMINI MINIMI SONO SEMPLICI, CON AL PIÙ  $|V|-1$  ARCI NEMICO

• LETTERA: UN SOTTOCAMMINO DI UN CAMMINO MINIMO È UN CAMMINO MINIMO  
 $\hookrightarrow$  SE CONSIDERO UN SOTTOARCO, ESSO È MINIMA TRA I 2 NUOVI ARCOLI CONSIDERATI

• DECAYATION: SONO STILICI LA SISTEMA DI V DA S PER PIZZI PIZZA  
UNIFORME: SE CONSIDERO IL CAMMINO DA  $s$  A  $u$  +  $\text{wt}(u, v)$

```
if (d[v] > d[u] + w(u,v)) {
    d[v] = d[u] + w(u,v);
    st[v] = u;
}
```

• DISUG. TRAVERSATE:  $S(s, v) \leq S(s, u) + w(u, v)$  GRADO MINIMO

• FORMA:  $d(v) \geq d(s, v)$

• SE  $\nexists$  ARCO:  $d(v) = S(s, v) = \infty$

• DOPO IL RISASSIEMO:  $d(v) \leq d(u) + w(u, v)$

$\hookrightarrow$  DIJKSTRA: 1 VOLTA AD ARCO  
BELLMAN-FORD: DUE VOLTE PIZZA ARCO

FIN A CHE PQ È PIANA  
 $\rightarrow$  ESTRAGGI IL VERDE Y OUT.  
NEMICO E APPENDI REGOLE AI NUOVI

DIJKSTRA

```
while (!PQempty(pq)) {
    if (d[v] == PQextractMin(pq, d)) != maxWT)
        for (t=G->adj[v]; t!=G->z; t=t->next)
            if (d[t->v] + t->wt < d[t->v]) {
                d[t->v] = d[v] + t->wt;
                PQchange(pq, d, t->v);
                st[t->v] = v;
            }
}
```

• DIJKSTRA:  $O(|V|)$ ,  $O(|E||V|)$ ,  $O(|E||V|)$ ,  $O(|E|)$   $\rightarrow O((|V|+|E|)\log|V|)$

SI PUÒ APPROPRIARE DI PQ

• USO UNA COSA A POLARITÀ PIZZA / VERDURA CON CONSIDERAZIONI

• CICLO CAMMINO MINIMO VERDURA DECAYATION

$d[\cdot]$   
 $st[\cdot]$  e PQ

• CON PESI  $\leq 0$  SUCCIDE OLTRE NON SONO VERTICI: LA DECAYATION SI PUÒ APPROPRIARE: SOLO 1 VOLTA PER ARCO  $\rightarrow$  NON POSSO VERIFICARE L'ARCO MINIMA MA CICLO

• BELLMAN-FORD:  $T(n) = O(|V||E|)$

o<sub>sgn</sub>  $d[\cdot]$  e  $st[\cdot]$

• PROBLEMA DINAMICO  $d[\cdot]$  e  $st[\cdot]$

• NON CONTIENE LA LISTA MINIMA TRA S E V, ALCUNA DECAYATION

• QUOTIDIANO IL NUMERO MAX DI ARCI PIÙ A  $|V|-1 \rightarrow$  AL PASSO  $\exists$  UN CICLO CONTENUTO SE

• LA DECAYATION PUÒ ESSERE APPLICATA PIÙ VOLTE A UN ARCO

```
for (w=0; w<=V-1; w++) V[i]=1 CHECK FOR DECAYATION
for (v=0; v<=V; v++) V[v]=0
if (d[v] < maxWT)
    for (t=G->adj[v]; t!=G->z; t=t->next)
        if (d[t->v] > d[v] + t->wt) {
            d[t->v] = d[v] + t->wt;
            st[t->v] = v;
        }
}
negacycFound = 0;
for (v=0; v<=V; v++)
    if (d[v] < maxWT)
        for (t=G->adj[v]; t!=G->z; t=t->next)
            if (d[t->v] > d[v] + t->wt)
                negacycFound = 1;
```

negacyc  
CICLO  $\leq 0$

# • STRUCTURE DAM

## • BST :

BST.h

```
typedef struct binarysearchtree *BST;
```

```
BST.c
#include <stdlib.h>
#include "Item.h"
#include "BST.h"

typedef struct BSTnode* link;
struct BSTnode { Item item; link l; link r; };
struct binarysearchtree { link root; link z; };
```

NODO  
DI SC.  
SOTTO  
ESSO

ROOT

LINK

CNT

BST.NODE

BST.NODE

## • GRAPHI :

```
typedef struct edge { int v; int w; int wt; } Edge;
```

```
typedef struct graph *Graph;
```

```
struct graph {int V; int E; int **adj; ST tab;};
```

```
123 void GRAPHdfs(graph_p G, int id){
124
125     int v, time=0, *pre, *post, *st;
126
127     pre = malloc(G->V * sizeof(int));
128     post = malloc(G->V * sizeof(int));
129     st = malloc(G->V * sizeof(int));
130
131     for(v=0; v<G->V; v++){
132
133         pre[v]=-1; post[v]=-1; st[v]=-1;
134
135         dfsR(G, EDGEcreate(id, id, 0, '\0'), &time, pre, post, st);
136
137         for(v=0; v<G->V; v++){
138             if (pre[v]==-1)
139                 dfsR(G, EDGEcreate(v, v, 0, '\0'), &time, pre, post, st);
140
141     }
142
143     link t = malloc(sizeof(*t));
144     int v, w = e.w;
145     //Edge x;
146
147     if (e.v != e.w)/*
148     printf("(%d, %d): T\n", STsearchByIndex (G->tab, e.v),
149           STsearchByIndex (G->tab, e.w)); */
150     st[e.w] = e.v;
151
152     pre[w] = (*time)++;
153
154     for(t = G->list_adj[w]->next; t != NULL; t=t->next){ // ERRORE !!!
155         if (pre[t->w] == -1)
156             dfsR(G, EDGEcreate(w, t->w, e.wt, e.type), time, pre, post, st);
157         else{
158             v = t->w; // w->v
159
160             if (post[v] == -1){
161                 for(t = G->list_adj[w]->next; t != NULL; t = t->next)
162                     if(t->w == v)
163                         t->type = 'B';
164             }
165             else if(pre[v] > pre[w]){
166                 for(t = G->list_adj[w]->next; t != NULL; t = t->next)
167                     if(t->w == v)
168                         t->type = 'F';
169             }
170             else{
171                 for(t = G->list_adj[w]->next; t != NULL; t = t->next)
172                     if(t->w == v)
173                         t->type = 'C';
174             }
175             //assert(t != NULL);
176             //t=t->next;
177         }
178         post[w] = (*time)++;
179     }
180 }
```

$D = \{(x, y) \in \mathbb{R}^2 : x \leq y \leq x(2-x)\}$ . La coordinata  $y_G$  del baricentro di  $D$  è

## • DOMANDA TEORIA :

- DIFF. PROPR. DINAMICA - METODI DI B.
- COSA VENGONO TRA DI RETTE DI BARRON PER D?
- SONO RICORSIVI IN LODO → NON

## • PROB.

- INIZIALIS. INSIEMISTI
- MOL COMBIN. TIPO  $b_1, b_2$