

# ESERCIZI Ricorrenze Lineari con MT

- $T(n) = 5T(n/2) + h$

$$(a \text{ cost } a)S = (a)T$$

$$d = \log 5 / \log 2 = 2 > 1 \Rightarrow T(n) = O(n^2)$$

- $T(n) = 5T(n/2) + n^2$

$$d = \log 5 / \log 2 = 2 = 2 \Rightarrow T(n) = O(n^2 \log n)$$

- $\bar{T}(n) = 5\bar{T}(n/2) + n^3$

$$d = \log 5 / \log 2 = 2 < 3 \Rightarrow \bar{T}(n) = O(n^3)$$

- $\bar{T}(n) = T(n/4) + T(\frac{3}{4}n) + n$

LIVELLO

0	$T(n/4)$	$+ T(\frac{3}{4}n)$	$+ n$	$n$
1	$\overbrace{T(\frac{n}{16}) + T(\frac{3}{16}n)}^{(a)T} + T(\frac{3}{16}n) + T(\frac{9}{16}n) + n$	$(a)T$	$\frac{n}{16} + \frac{6}{16}n + \frac{9}{16}n$	$\frac{n}{16} + \frac{15}{16}n$
[::]	[::]	[::]	$m + (m)2$	$[::]2$
$h$	$T(1)$	$m + T(1)$	$\frac{n}{4^h}$	$\frac{n}{4^h}$

$$\Rightarrow \sum_{i=0}^{\log_4 n} \frac{n}{4^i} = n \sum_{i=0}^{\log_4 n} \frac{1}{4^i} = O(n \log n)$$

$$(a \text{ cost } a)S = (a)T$$

$$\bullet T(n) = T(n-1) + \log n$$

$$T(n) \stackrel{?}{=} O(n \log n)$$

HYP IND:  $T(m) \leq cm \log m$   $\forall m < n$  e vogliamo dimostrare che la proprietà vale per  $n$ .

$$\begin{aligned} T(n) &= T(n-1) + \log n \\ &\leq c(n-1) \log(n-1) + \log n \\ &\leq c(n-1) \log n + \log n \\ &= cn \log n - c \log n + \log n \\ &\leq cn \log n \quad \text{per } c \geq 1 \end{aligned}$$

$$\bullet T(n) = 2T(\sqrt{n}) + \log n = 2T(n^{1/2}) + \log n$$

$$\text{Poniamo } n = 2^m$$

$$T(2^m) = 2T(2^{m/2}) + \log 2^m$$

$$\text{Poniamo } S(m) = T(2^m)$$

$$S(m) = 2S(m/2) + m$$

che per TTT ( $\alpha = \log 2 / \log 2 = 1 = \beta = 1$ ) si ha che  $S(m) = \Theta(m \log m)$ ; vedi sopra che:

$$n = 2^m \Rightarrow m = \log n$$

$$T(n) = \Theta(\log n \log \log n)$$

## ESERCIZI:

$$T(n) = \begin{cases} 2T(n/8) + 2T(n/4) + n & n > 1 \\ 1 & n \leq 1 \end{cases}$$

- Cerco di dimostrare per induzione che  $T(n) = O(n)$

$$T\left(\frac{n}{4}\right) = 1 \leq 1 \cdot c \Rightarrow c \geq 1$$

$\exists c > 0, \exists m \geq 0$  t.c.  $T(n) \leq cn \quad \forall n \geq m$

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{8}\right) + 2T\left(\frac{n}{4}\right) + n \\ &\leq 2c\left(\frac{n}{8}\right) + 2c\left(\frac{n}{4}\right) + n \\ &= c\frac{n}{4} + c\frac{n}{2} + n \\ &= c\frac{3n}{4} + n \leq cn \\ n\left(\frac{3}{4}c + 1\right) &\leq cn \\ \frac{3}{4}c + 1 &\leq c \Rightarrow c \geq 5 \end{aligned}$$

- Cerco di dimostrare che  $T(n) = \Omega(n)$

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

Essendo  $T(n)$  sicuramente  $\geq n$ , ed  $n = \Omega(n)$ , allora si può dire che  $T(n) = \Omega(n)$

$$T(n) = \begin{cases} 2T(n/8) + 2T(n/4) + n\log n & n > 1 \\ 1 & n \leq 1 \end{cases}$$

• Cerco di dimostrare per induzione che  $T(n) = \mathcal{O}(n \log n)$

$$T(1) = 1 \leq c \cdot 1 \log_2 1 \Rightarrow 1 \leq c \cdot 0 \text{ FALSO}$$

$$\begin{aligned} T(2) &= 2 \cdot T\left(\frac{2}{8}\right) + 2T\left(\frac{2}{4}\right) + 2 \cdot \log_2 2 \\ &= 2 \cdot 1 + 2 \cdot 1 + 2 = 6 \leq c \cdot 2 \cdot \log_2 2 \Rightarrow c \geq 3 \end{aligned}$$

$\exists c > 0, \exists m > 0$  t.c.  $T(n) \leq cn \log n \forall n \geq m$

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{8}\right) + 2T\left(\frac{n}{4}\right) + n \log n \\ &\leq 2c \cdot \frac{n}{8} \log \frac{n}{8} + 2c \cdot \frac{n}{4} \log \frac{n}{4} + n \log n \end{aligned}$$

$$\begin{aligned} \textcircled{1} &\leq 2cn \left( \frac{1}{8} \log \frac{n}{8} - \log 8 + \frac{1}{4} \log \frac{n}{4} - \log 4 \right) + n \log n \\ &\leq 2cn \left( \frac{1}{8} \log n + \frac{1}{4} \log n \right) + n \log n \\ &= \frac{3}{4}cn \log n + n \log n \leq cn \log n \\ &c \geq 4 \end{aligned}$$

$$\log_a \frac{b}{c} = \log_a b - \log_a c$$

① posso togliere le costanti negative ( $-\log 8, -\log 4$ ) perché sto usando un  $\leq$   $\Rightarrow f(n) + c \leq f(n)$

[con  $c > 0$ ]

$$T(n) = \begin{cases} T(n/5) + T(\frac{7}{10}n) + \frac{11}{5}n & n > 1 \\ 1 & n \leq 1 \end{cases}$$

- Cerco di dimostrare per induzione che  $T(n) = O(n)$

$$T(1) = 1 \leq c \Rightarrow c \geq 1$$

$$\exists c > 0, \exists m \geq 0 \text{ t.c. } T(n) \leq cn \quad \forall n \geq m$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + \frac{11}{5}n \\ &\leq c\left[\frac{n}{5}\right] + c\left[\frac{7}{10}n\right] + \frac{11}{5}n \\ &\leq c\frac{n}{5} + c\frac{7}{10}n + \frac{11}{5}n \\ &= c\frac{2n}{10} + c\frac{7}{10}n + \frac{22}{10}n \\ &= c\frac{9}{10}n + \frac{22}{10}n \leq cn \\ &\times \left(c\frac{9}{10} + \frac{22}{10}\right) \leq cx \end{aligned}$$

$$c\frac{9}{10} - \frac{10}{10}c \leq -\frac{22}{10}$$

$$-\frac{1}{10}c \leq -\frac{22}{10}$$

$$-c \leq -22$$

$$c \geq 22$$

$$\Rightarrow T(n) = O(n) \text{ e } T(n) = \Omega(n) \Rightarrow T(n) = \Theta(n)$$

$$\bar{T}(n) = \begin{cases} \min_{1 \leq k \leq n-1} \{ T(k) + T(n-k) \} + 1 & n > 1 \\ 1 & n \leq 1 \end{cases}$$

In questo caso min sceglie tra:

$$\begin{aligned} T(1) + T(n-1) + 1 \\ T(2) + T(n-2) + 1 \end{aligned}$$

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

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

$\Rightarrow$  Può essere limitata superiormente da:

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

che per RT:

$$T(n) = O(n)$$

Provo a dimostrare per induzione che  $T(n) = O(n)$

$$T(1) = 1 \leq c \Rightarrow c > 1$$

$\exists c > 0 \ \exists m \geq 0$  t.c.  $T(n) \leq cn \ \forall n \geq m$

$$T(n) = \min_{1 \leq k \leq n-1} \{ T(k) + T(n-k) \} + 1$$

$$\leq \min_{1 \leq k \leq n-1} \{ ck + c(n-k) \} + 1$$

$$= \min_{1 \leq k \leq n-1} \{ ck + cn - ck \} + 1$$

$$= cn + 1 \leq cn \Rightarrow \text{non esiste } c \text{ che soddisfa la disegualità}$$

$\exists c > 0, \exists m \geq 0$  t.c.  $T(n) \leq cn - b \ \forall n \geq m$

$$T(n) = \min_{1 \leq k \leq n-1} \{ T(k) + T(n-k) \} + 1$$

$$\leq \min_{1 \leq k \leq n-1} \{ ck - b + c(n-k) - b \} + 1$$

$$= \min_{1 \leq k \leq n-1} \{ cn - 2b \} + 1$$

$$= cn - 2b + 1 \leq cn - b$$

$$-2b + b \leq -1$$

$$b \geq 1$$

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

Scrivere un algoritmo ricorsivo che riceva come  
parametro un vettore di  $n$  elementi t.c. il suo  
tempo di esecuzione  $T(n)$  verifichi la relazione

$$T(n) = \begin{cases} d & n \leq k \\ 5T(n/2) + 2T(n-1) + cn^2 & n > k \end{cases}$$

```

int T(item[A, int s, int e])
    int c ← 0
    if (s > e)
        for int i ← 1 to 5 do
            c ← c + T(A, s+1, ⌊(s+e)/2⌋) } 5T(n/2)
        c ← c + T(A, s, e-1) } 2T(n-1)
        c ← c + T(A, s, e-1)
        for int j ← 1 to e do
            for int k ← 1 to e do
                c ← c + 1 } O(n^2)
    return c

```

Scrivere l'equazione di ricorrenza e la complessità del seguente algoritmo:

int M(int A[], int n)

    int k ← 0

    for int i ← 1 to n do

        k ← k + A[i]/n

    if (n > 44)

        return 3 · M(A, n-2) + k

    else

        return k.

$$T(n) = \begin{cases} T(n-2) + 2 & n > 44 \\ n & n \leq 44 \end{cases}$$

Per T(n),  $T(n) = O(n^2)$

NB: ① è eseguito ad ogni chiamata ricorsiva, quindi se  $n \leq 44$  la complessità è  $n$

② è diverso da for int i ← 1 to 3 do { $M(A, n-2) + k$ } perché è fatta solo una chiamata ricorsiva.

Calcolare la complessità dell'equazione:

$$T(n) = \begin{cases} \sqrt{n} T(\sqrt{n}) + \sqrt{n} & n > 1 \\ 1 & n \leq 1 \end{cases}$$

Sostituisco  $\boxed{n = 2^k}$

$$T(2^k) = \begin{cases} \sqrt{2^k} T(\sqrt{2^k}) + \sqrt{2^k} & k > 1 \\ 2^{k/2} T(2^{k/2}) + 2^{k/2} & k \leq 1 \end{cases}$$

Divido per  $2^k$  e calcolo la complessità sulla nuova funzione  $S(k)$ .

$$\frac{T(2^k)}{2^k} = \frac{T(2^{k/2})}{2^{k/2}} + \frac{1}{2^{k/2}}$$

$\Downarrow$

$$S(k) = S(k/2) + \frac{1}{2^{k/2}} = S(k/2) + O(1)$$

La complessità di  $S(k)$  è  $O(\log k)$  per TT; per tornare alla complessità di  $T(n)$ :

$$S(k) = \frac{T(2^k)}{2^k} = O(\log k)$$

$$T(2^k) = 2^k O(\log k)$$

$$\boxed{K = \log n}$$

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

Dimostriamo il limite inferiore:

$$T(n) \stackrel{?}{=} \Omega(n)$$

$\exists c, \exists m > 0 \quad T(n) \geq cn \quad \forall n \geq m$

$$\begin{aligned} T(n) &= \sqrt{n} T(\sqrt{n}) + \sqrt{n} \\ &\geq \sqrt{n} c \sqrt{n} + \sqrt{n} \\ &= cn + \sqrt{n} \geq cn \end{aligned}$$

$$c > 0 \Rightarrow T(n) = \Omega(n)$$

Si può stringere il limite superiore fino a  $O(n)$ ?

$$T(n) \stackrel{?}{=} O(n)$$

$\exists c, \exists m > 0 \quad T(n) \leq cn \quad \forall n \geq m$

$$\begin{aligned} T(n) &= \sqrt{n} T(\sqrt{n}) + \sqrt{n} \\ &\leq \sqrt{n} c \sqrt{n} + \sqrt{n} \\ &= c\sqrt{n} + \sqrt{n} \leq cn \end{aligned}$$

$\nexists c \Rightarrow$  non riesco a trovare una costante

Restringo la ricerca con:

$\exists c, \exists m > 0 \quad T(n) \leq cn - b\sqrt{n} \quad \forall n \geq m, b > 0$

$$\begin{aligned} T(n) &= \sqrt{n} T(\sqrt{n}) + \sqrt{n} \\ &\leq \sqrt{n}(c\sqrt{n} - b\sqrt{\sqrt{n}}) + \sqrt{n} \\ &= cn - b\sqrt{n}\sqrt{\sqrt{n}} + \sqrt{n} \leq cn - b\sqrt{n} \\ &\quad - b\sqrt{\sqrt{n}} + 1 \leq b \end{aligned}$$

- ho trovato un fattore

$b$  che rende vera la

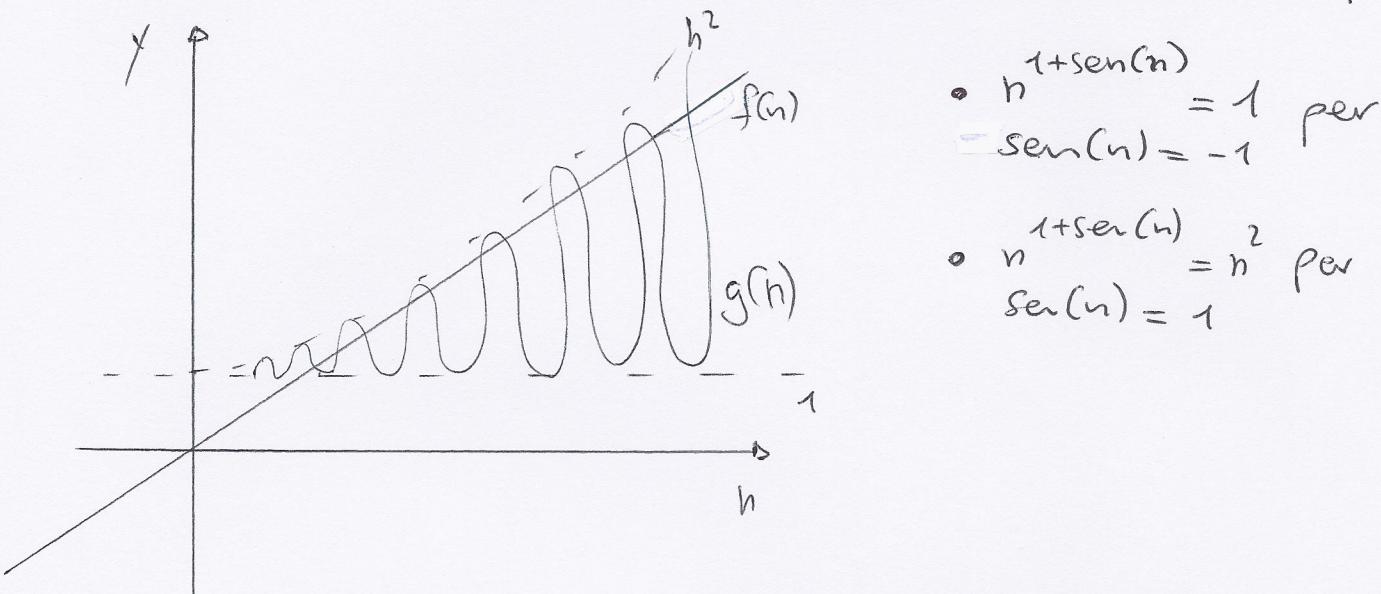
dimostrazione  $T(n) = O(n)$

$$b \geq \frac{1}{\sqrt{n'} - 1}$$

## Esercizio

Si considerino le funzioni  $f(n) = n$  e  $g(n) = n^{1+\sin(n)}$ .  
 Si dimostri che le due funzioni non sono confrontabili  
 in ordine di grandezza, cioè che non vale né che  
 $f(n) = O(g(n))$ , né  $f(n) = \Omega(g(n))$ .

→ Se si considera la funzione  $g(n)$ , si nota che  
 oscilla tra  $1$  e  $n^2$  in ordine di grandezza,  
 in quanto  $\sin(n)$  è compreso tra  $-1$  e  $1$ .



⇒ non si riesce a trovare un punto "m" oltre il quale  $f(n)$  è dominato superiormente da  $g(n)$ .  
 Stesso ragionamento vale per  $\Omega(g(n))$  solo per  
 il limite inferiore.

Dato un albero radicato  $T$ , calcolare il num tot di nodi.

Integer nodiABR(Tree  $T$ , integer  $c$ )

```
| if (T.left = nil AND T.right = nil)  
|   return c  
else  
|   return nodiABR(T.left, c++) +  
|       nodiABR(T.right, c++)
```

Integer nNodiABR(Tree  $T$ )

```
| if (T = nil) return 0  
else return nodiABR(T, 1)
```

Dato un albero radicato  $T$ , stampare tutti i nodi a profondità  $h$ .

printH (Tree  $T$ , integer  $h$ )

```
| if (h == 0)  
|   print(T.val)  
else  
|   if (T.right != nil)  
|       printH(T.right, h--)  
|   else if (T.left != nil)  
|       printH(T.left, h--)  
|   else return
```

```
| if (T != nil)  
|   if (h > 0)  
|       printH(T.left, h-1)  
|       printH(T.right, h-1)  
|   else  
|       print(T.val)
```

Scrivere un algoritmo che, dato un array ordinato A di numeri interi con ripetizioni, e dato un numero K da cercare in A, restituisce l'indice corrispondente all'ultima occorrenza di K in A. (-1 altrimenti)

$$A = [2, 2, 4, 5, 5, 7, 7], K=5 \Rightarrow i=5$$

- ① ... 2  $\underset{\uparrow}{K}$  K ... - ... 2  $\underset{\uparrow}{K}$  2 ...
- ② ... K  $\underset{\uparrow}{K}$  K ...
- ③ ... K  $\underset{\uparrow}{K}$  2 ... - ... 2  $\underset{\uparrow}{K}$  2 ...

integer F(integer[] A, integer k)

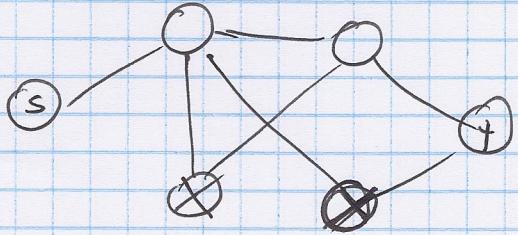
```
integer i = binarySearch(A, k)
if (i = -1) return -1
return moveToLastOccurrence(A, k, i)
```

integer moveToLastOccurrence(integer[] A, integer K, int i)

```
while (A[i] = K AND i+1 ≠ A.length) // invertire le condizioni
    i = i+1
return i
```

boolean allow (Graph G, Node n, Node[] p)

| for i ← 1 to p.length do  
| | if n = p[i] return false  
| return true



boolean cammino (Graph G, Node s, Node t, Node[] p)

| s.visitato ← true → if s = t return true  
| for v ∈ G.adj(s) do  
| | if ((not v.visitato) AND allow(v, p))  
| | | v.visitato ← true  
| | | return cammino(G, v, t, p)  
| return false

$O((n+m) \cdot j)$

n: num nodi

m: num svchi

j: num nodi proib.

| p.length è sempre ≤ n-2 perché suppongo che  
| s ≠ p e s ≠ t.

Si scriva un algoritmo che, dato un grafo G non orientato, un nodo di partenza  $s^*$  e un nodo di arrivo  $t$ , ed un array di nodi proibiti  $p$ , dice se esiste un cammino da  $s^*$  a  $t$  che non tocca nessun nodo dell'insieme  $p$ .  
(si supponga sempre  $s \neq t$ ).

Dato un albero radicato in  $T$ , calcolare la sua altezza.

integer altezza(Tree  $T$ , integer  $h$ )  
| if( $T.\text{left} = \text{nil}$  OR  $T.\text{right} = \text{nil}$ )  
| | return  $h$   
| else  
| | return  $\max(\text{altezza}(T.\text{left}, h+1),$   
| | |  $\text{altezza}(T.\text{right}, h+1))$ ,

integer altezzaABR(Tree abr)

| if( $\text{abrv} = \text{nil}$ )  
| | return 0  
| else  
| | return altezza(abr, 1)

integer altezzaABR(Tree  $T$ )

| if ( $T \neq \text{nil}$ )  
| | return  $\max(\text{altezzaABR}(T.\text{left}) + 1,$   
| | |  $\text{altezzaABR}(T.\text{right}) + 1)$   
| else  
| | return 0

## ESERCIZI

Ricavare l'albero binario dalle seguenti visite:

A, E, B, F, G, C, D, I, H (anticipata)

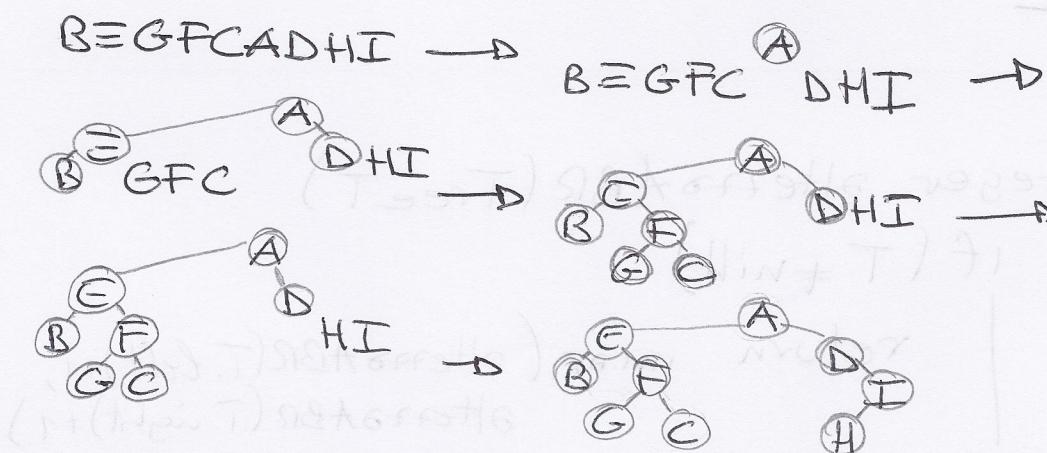
B, G, C, F, E, H, I, D, A (posticipata)

B, E, G, F, C, A, D, H, I (simmetrica)

Risoluzione:

Per risolvere questo problema basta considerare la visita anticipata e simmetrica:

- trovo la radice  $\rightarrow$  primo elemento visitato nella visita anticipata.
- per ogni nodo della visita anticipata (da sx a dx) sposto ogni suo nodo in giù di un livello dalla visita simmetrica.



## ESERCIZI

Trovare il massimo grado di sbilanciamento dell'albero  $T$ . Dove grado di sbilanciamento di un nodo  $v$  si intende la differenza fra il numero di foglie presenti nel sotto albero sinistro e nel sotto albero destro di  $v$ .

tra tutti i nodi

integer  $S(\text{TREE } n, \text{ integer } m)$

if ( $n.\text{left} = \text{nil}$  AND  $n.\text{right} = \text{nil}$ )

return 1

else

integer  $d = |S(n.\text{left}) - S(n.\text{right})|$

if ( $d > m$ )

$m = d$

return  $d$

integer Sbilanciamento( $\text{TREE } T$ )

if ( $T = \text{nil}$ ) return -1

$\max = 0$

$S(T, \max)$

return  $\max$

Dato un albero binario  $T$ , trovare il più lungo  
cammino radice - foglia monotona crescente

integer CaminoMonotono(TREE T)

if ( $T.\text{right} = \text{nil}$  AND  $T.\text{left} = \text{nil}$ )

    return 1

else

    if ( $T.\text{left}.val > T.val$ )

        lmax = Camino( $T.\text{left}$ ) + 1

    else if ( $T.\text{right}.val > T.val$ )

        rmax = Camino( $T.\text{right}$ ) + 1

    return max(lmax, rmax)

b = m

b nroter

( $T \in ST$ ) et. 3.6.3.6.1.1.2 vegetari

~ nroter ( $\text{Lan} = T$ )

O = x6m

(x6m, T) 2

x6m nroter

Scrivere una funzione che, dato un vettore A di n interi distinti, dove esiste un valore K t.c.:

- $A[1..K]$  sono ordinati in modo crescente
- $A[K..n]$  sono ordinati in modo decrescente.

Si trovi l'indice dell'elemento K.

① Soluzione  $\Theta(n)$ : scorro il vettore fino a trovare la discesa dei valori

② Soluzione  $\Theta(\log n)$ : divide-et-impera

```
int search(int[] A, int s, int e)
{
    if (s > e)
        int k <- L(s+e)/2
        if (A[k] < A[k+1])
            return searchK(A, k, e)
        else if (A[k] < A[k-1])
            return searchK(A, s, k)
        else
            return k
}
```

Dato un vettore di numeri interi  $A$ , dire se esistono due elementi in  $A$  la cui somma è esattamente  $V$ :

- Soluzione  $\Theta(n \log n)$ :

```
bool isSumInVector(int[] A, int V)
    sort(A)
    for int i < 1 to n
        [ return binarySearch(A, 1, n, V - A[i]) ]
```

( $\Theta(n \log n)$ )  $\Rightarrow N$  passi

( $\Theta(n^2)$ )  $\Rightarrow 1$

$[O(\log n)] \Rightarrow N$  passi

( $\Theta(n)$ )  $\Rightarrow O(N)$  passi

( $\Theta(n^2)$ )  $\Rightarrow N$  passi

( $\Theta(n^2)$ )  $\Rightarrow O(N^2)$  passi

$N$  passi

- Elemento in Posizione: Sia dato un vettore ordinato A di elementi distinti. Trovare se esiste un indice del vettore t.c.  $A[i] = i$ . e restituirlo.

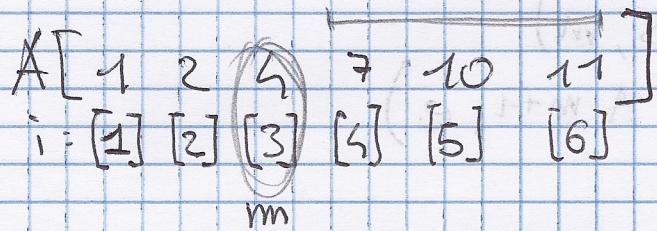
int F(int[] A, int s, int e)

$$m = \lfloor (e-s)/2 \rfloor$$

if ( $A[m] == m$ ) return m

else if ( $A[m] < m$ ) ret F(A, s, m-1)

else if ( $A[m] > m$ ) ret F(A, m+1, e)



Idea della soluzione: l'indice m spezza a metà il vettore e controlla solo la parte sx del vettore se  $A[m] < m$  perché nella parte dx essi saranno i indici ed elementi sempre crescenti (sfruttando A ordinato). Stesso ragionamento si applica quando  $A[m] > m$  solo applicato alla parte dx del vettore.

Il caso base è quando  $A[m] == m$  che è l'obiettivo.

- Chi manca? Dato un vettore ordinato  $A[1..n]$  contenente  $n$  elementi ~~int~~ interi distinti appartenenti all'intervallo  $(1, n+1)$ . Restituire quale elemento manca nel vettore.

```
int F(int[] A, int s, int e)
| m = [(e-s)/2]
| if(m >= A[s] == s)
|   | ret s+1
| else ret s
| if(m < A[m]) ret F(A, s, m)
| else if(m == A[m]) ret F(A, m+1, e)
```

$A[1, 2, 3, 4, 5]$   
[1] [2] [3] [4] [5]

L'idea alla base per non guardare tutto il vettore è la seguente: se l'indice  $m$  è  $<$  di  $A[m]$  significa che dall'indice  $m$  all'inizio del vettore ci sarà l'elemento mancante, (sfottando il fatto che il vettore è ordinato, l'indice deve corrispondere agli elementi di  $A$  sono nell'intervallo  $(1, n+1)$ ). Nel caso l'indice  $m$  coincida con  $A[m]$  significa che nella parte a sinistra ~~dell'indice~~ di  $m$  non sarà possibile trovare l'elemento mancante, per le ragioni appena spiegate.

## ESERCIZI

- Vettore ordinato di  $n$  interi distinti, determinare se esiste un indice  $i$  t.c.  $A[i]=i$  in  $O(\log n)$

```

bool FixPoint(integer A[], integer s, integer e)
| if (s > e)
|   integer p = ⌊(s+e)/2⌋
|   if (A[p] == p)
|     return true
|   else if (A[p] > p)
|     return FixPoint(A, s, p)
|   else if (A[p] < p)
|     return FixPoint(A, p+1, e)
|   else return false
  
```

- Vettore  $A[1..n]$  t.c.  $A[1] < A[n]$ . Trovare l'indice di una coppia di elementi  $i, i+1$  t.c.  $A[i] < A[i+1]$ . Scrivere un algoritmo in  $O(\log n)$ .

```

integer check(integer A[], integer s, integer e)
| if (e - s + 1 == 1)
|   return -1
| else
|   integer p = ⌊(s+e)/2⌋
|   if (A[s] < A[p])
|     return check(A, s, p)
|   else
|     return check(A, p, e)
  
```

Scrivere un algoritmo ricorsivo che riceva come  
parametro un vettore di  $n$  elementi t.c. il suo  
tempo di esecuzione  $T(n)$  verifichi la relazione:

$$T(n) = \begin{cases} d & n \leq k \\ 5T(n/2) + 2T(n-1) + cn^2 & n > k \end{cases}$$

INTEGER  $T(\text{INTEGER } A, \text{ INTEGER } s, \text{ INTEGER } e)$

INTEGER  $c = 0$

if ( $s > e$ )

for integer  $i \leftarrow 1$  to 5 do

$c \leftarrow c + T(A, s+1, \lfloor (s+e)/2 \rfloor)$

$c \leftarrow c + T(A, s, e-1)$

$c \leftarrow c + T(A, s, e-1)$

for  $j \leftarrow 1 + e$  do

for  $k \leftarrow 1$  to  $e$  do

$c \leftarrow c + 1$

return  $c$

Dato un grafo orientato  $G = (V, E)$ .  
 Un nodo  $v$  è di tipo "roma" se  $\forall$  altro nodo c'è  
 un cammino verso  $v$ .

① Scrivere una funzione che dato un nodo dice se  
 è di tipo romo:

```

 $G^T \leftarrow \text{Trasposto}(G)$ 
 $E \leftarrow \text{new Integer}[1 - G.n]$ 
Erdos( $G^T, v, E, 0$ )
for  $n \in G.V()$ 
    if ( $E[n] = +\infty$ )
        return false
return true
    
```

- visita il grafo  $G^T$  dal  
 nodo  $v$  e vedo se  
 raggiunge tutti altri  
 nodi.

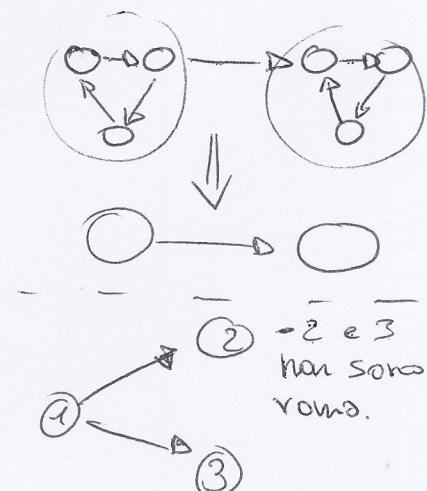
② Scrivere una funzione che controlla se nel grafo esiste  
 un nodo romo.

```

Roma(Graph  $G^T$ )
    stack  $s \leftarrow \text{TopSort}(G^T)$ 
    Node  $n \leftarrow s.pop()$ 
    return isRoma( $G^T, n$ )
    
```

N.B.: isRoma è la funzione del punto ① senza il calcolo di  $G^T$ .  
 Oppure:

- calcolare le componenti fortemente connesse
- topSort del grafo ottenuto
- per tutti i nodi ultimi nel topSort  
 controllare se sono romo



Scrivere una funzione che prenda come parametro un vettore  $S$  di interi ordinati e un intero  $x$ , dire se  $\exists i, j, i \neq j$  t.c.  $S[i] + S[j] = x$ .

① soluzione in  $\Theta(n^2)$ :

- due cicli for nested che controllano ogni coppia di valori.

② soluzione in  $\Theta(n \log n)$ :

- indice  $i$  scorre il vettore dall'inizio
- si fa partire una ricerca sulla porzione di vettore  $A[i+1, n]$  dell'elemento  $x - S[i]$ .

③ soluzione in  $\Theta(n)$ :

- indice  $i$  scorre il vettore dall'inizio
- indice  $j$  scorre il vettore dalla fine
- calcolo  $sum = S[i] + S[j]$
- if ( $sum > x$ )  $i = i + 1$
- else if ( $sum < x$ )  $j = j + 1$
- else return  $(i, j)$
- continuo l'iterazione fino a  $i = j$

## Esercizi

- Dato un grafo  $G$  non orientato, dire il numero di archi da aggiungere per renderlo连通的.
  - Idea: un grafo è连通的 se ha solo una componente连通的.
  - Soluzione: modificare la funzione  $cc()$  perché ritorni il valore di "conta - 1".
- 
- Dato un grafo  $G$  non orientato, aggiungere un numero minimo di archi per renderlo连通的.
  - Soluzione non efficiente:
 

```
integer[] ccID = cc(G, ordire)
id = 1
for i=id to ccID.size() do
  if (ccID[i] != id)
    collega G.V[i] con G.V[id]
    id = i
```
  - Soluzione efficiente: modificare la funzione  $cc()$  aggiungendo un vettore chiamato rappresentante dove ogni elemento è un nodo rappresentante della componente连通的 a cui appartiene; e facendolo ritornare dalla  $cc()$ . Ciclando su questo vettore si collega un nodo con il successivo.

## GRAFO BIPARTITO

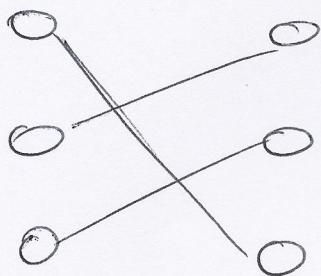
Un grafo  $G$  non orientato si dice bi-partito se

$$V_1 \subset V, V_2 \subset V$$

$$V_1 \cup V_2 = V, V_1 \cap V_2 = \emptyset$$

ovvero:

$$\forall (u, v) \in E \Rightarrow (u \in V_1, v \in V_2) \vee (u \in V_2, v \in V_1)$$



$G$  è bipartito sss  $G$  è duo-colorabile: cioè se è possibile colorare i nodi con due colori senza che ci siano due vicini con due colori uguali.  
 $G$  è bipartito sss  $G$  non contiene cicli di lunghezza dispari.

Come bi-colorare un Grafo:

boolean isBicolorable(Graph G, Node r)

```

Queue S ← Queue()
S.enqueue(r);
boolean[] visitato ← new boolean[G.n]
boolean[] colorato ← new boolean[G.n]
foreach u ∈ G.V - {r} do
    visitato[u] ← false; colorato[u] ← null
    colorato[r] = true
while (not S.isEmpty()) do
    Node u ← S.dequeue();
    foreach v ∈ G.adj(u) do
        if (not visitato[v])
            visitato[v] ← true
            colorato[v] ← !colorato[u]
            S.enqueue(v)
        else
            if (colorato[v] = colorato[u])
                return false
return true
    
```

## Esercizi

Scrivere una funzione F che ritami il numero di caratteri da inserire per rendere la stringa in input palindroma.

$$F[i,j] = \begin{cases} 0 & S[i] = S[j] \\ \min\{F[i+1,j]+1, F[i,j-1]+1\} & S[i] \neq S[j] \end{cases}$$

L'idea è che i due indici controllano il primo ( $i$ ) e l'ultimo ( $j$ ) carattere; se sono uguali, quella parte di stringa è considerata Palindroma, se sono diversi devo fare un inserimento e spostare i cursori; prendo il minimo tra la scelta di spostare in avanti  $i$  (inserisco un carattere in testa) oppure spostato in dietro  $j$  (inserisco un carattere in coda).

## Osando Neutralization:

```

F(char[s], int i, int j, int[n])
if (i < j) ret 0
if (n[i, j] = ⊥)
    if (s[i] = s[j]) ret 1
        ret F(s, i+1, j-1, n)
    else
        n[i, j] = min(F(s, i+1, j, n)+1, F(s, i, j-1, n))+1
    ret n[i, j]
}

```

Calcolare tutte le possibili sequenze composte da  $n$  pallini gialli e  $m$  pallini rossi.

Essendo che si deve stampare tutte le possibili sequenze, l'unica strategia è di trovarle tutte.

$\text{f}(\text{char}[] A, \text{int } n, \text{int } m, \text{int } i)$

if (if ( $n=0$  and  $m=0$ ) print(A))

if ( $n > 0$ )

$A[i] = 'R'$

$F(A, n-1, m, i+1)$

if ( $m > 0$ )

$A[i] = 'G'$

$F(A, n, m-1, i+1)$

$$n((\pi_{j,i,(1,2)} + \pi_{j,i,(1,2)}))^{n+m-2} = [i,j]^{n+m}$$

[i,j]<sup>n+m</sup>

### 1.7 Mosse su scacchiera

Supponete di avere una scacchiera con  $n \times n$  caselle e un pedone che dovete muovere dall'estremità inferiore a quella superiore. Un pezzo si può muovere (1) una casella in alto, oppure (2) una casella in diagonale alto-destra, oppure (3) una casella in diagonale alto-sinistra. Non si può tornare indietro. Le caselle sono denotate da una coppia di coordinate  $(x, y)$ . Quando una cella  $(x, y)$  viene visitata, guadagnate un valore reale  $p(x, y)$ .

Calcolare un percorso da una qualunque casella dell'estremità inferiore ad una qualunque casella dell'estremità superiore, massimizzando il profitto.

Esempio di tabella  $p$  per una scacchiera  $5 \times 5$ ; sono evidenziati (uno in grassetto e uno con sottolineatura) due percorsi ottimi dalla riga inferiore a quella superiore, entrambi con valore 33.

6	7	4	7	8
7	<b>6</b>	1	1	<u>4</u>
3	5	<u>7</u>	<u>8</u>	2
2	<b>6</b>	<u>7</u>	0	2
7	3	5	<u>6</u>	1

$x_1 \in [1, n]$  è scelta dal giocatore a priori.

Definiamo percorso:  $X = (x_1, x_2, \dots, x_n)$  che porta da  $(x_1, 1)$  a  $(x_n, n)$  con guadagno massimo.

DICOSTRAZIONE SOTTOSTRUTTURA OTTIMA:

Dato il percorso  $X = x_1, \dots, x_n$  definiamo il sottopercorso  $X[\bar{y}] = (x_y, x_{y+1}, \dots, x_n)$ . Diciamo che  $X[\bar{y}]$  è ottimo (TOP IND). Allora  $X[\bar{y}+1]$  è un percorso ottimo fra  $x_{y+1}$  e  $x_n$ .

Supponiamo che esista un percorso  $X'[y+1] = x_{y+1}, x'_{y+2}, \dots, x'_n$  che ha guadagno maggiore di  $X[\bar{y}+1]$

$$G(X'[y+1]) > G(X[\bar{y}+1])$$

Allora  $X''[\bar{y}] = (x_y, x_{y+1}, x'_{y+2}, x_n)$  ha guadagno superiore a  $X[\bar{y}]$

$$G(X''[\bar{y}]) > G(X[\bar{y}])$$

Il che contraddice l'HIND  $\Rightarrow$  assurdo

□

## Equazione di ricorrenza:

$$x < 1 \quad \text{or} \quad x > 5$$

$$G[x, y] = \begin{cases} e(x, y) & y = h \\ \max \left\{ G[x-1, y+1], G[x, y+1], G[x-1, y+1] \right\} + P(x, y) & \text{altre} \end{cases}$$

Il caso base è  $y=n$  dove i guadagni sono fissati:

I guadagni della riga  $y=h-1, h-2, \dots, 1$  dipendono dalla riga precedente.

Nel vettore  $m$  vengono segnate le scelte  $d \in \{1, 0, -1\}$  della  $x$  che hanno portato ad un miglioramento del guadagno.

	-00	-00		
n	-00	$x_1$	$x_2$	
n-1				
1	-00			
0	-00			
	0	1		

tabella  $y: x_1, x_2, \dots, x_n$  valori  
del caso base

for  $x=1$  to  $n$  do  $g[x,n] = p(x,n)$  // case base

for  $y=h-1$  down to 1 do

for x=1 to n do

$g[x, y] = -\infty$  //setto un valore più piccolo di quanto.

for  $d \in \{-1, 0, +1\}$  do

$$x' = x + d$$

If  $x' \geq 1$  &  $x' < n$

localSum = p(x,y) + g[x',y+1] // y+1 won't be longer.

if localSum > g[x, y]

$g[x, y] = \text{localSum}$

$$m[x, y] = d$$

## Esercizi

Dato un testo (potenzialmente molto lungo), ritrovare quante volte contiene una clausa sottoseguente.

Due sotto seguenti sono considerate diverse (quindi contate separatamente) se esiste almeno una differenza negli insiemi di caratteri utilizzati.

ESEMPIO: la stringa "did you go" contiene due volte la ~~sua~~ sottoseguente "dog"

IDEA DELLA SOLUZIONE: questo tipo di problema è molto simile a cercare un pattern  $P(j)$  nel testo  $T(i)$ , con l'unica differenza che il pattern può essere diviso in più parole del testo.

La dimostrazione di sotto-struttura ottima si fa in modo analogo in tutti i problemi di questo tipo: si definisce  $T(i)$  come prefisso del testo fino all'iesimo carattere e  $P(j)$  come prefisso del pattern fino al  $j$ -esimo carattere.

Si definisce  $D[i,j]$  come sottoproblema: ovvero il numero di sottoseguente del testo  $T(i)$  con il pattern  $P(j)$

$$D[i,j] = \begin{cases} D[i-1,j] & T(i) \neq P(j) \text{ AND } j > 0 \\ D[i-1,j-1] + D[i-1,j] & T(i) = P(j) \text{ AND } j > 0 \\ 0 & i = 0 \text{ AND } j > 0 \\ 1 & j = 0 \end{cases}$$

Se  $T(i) \neq P(j)$ , l'ultimo carattere del pattern non matcha con l'ultimo carattere del testo, continuo a leggere il testo.  
Se  $T(i) = P(j)$  devo sommare il risultato ottenuto "consumando" un carattere del pattern ( $j-1$ ) oppure continuando a cercare nel testo ( $i-1,j$ )

Quando ho finito il testo ( $i=0$ ) non ho più sottoseguente del pattern e quando consumo l'ultimo carattere del pattern, ho trovato una sottoseguente completa.

La soluzione ricorsiva permette di scrivere un algoritmo con complessità superpolinomiale.

Una soluzione basata su memoization può essere scritta:

sottosequenza (char[T], char[P])

int[D][T, P] D = new int[T.n][P.n]

for k=0 to T.n do D[0,k] = 0

for k=0 to P.n do D[k,0] = 1

for i=1 to T.n do

for j=1 to P.n do

if P(j) = T(i)

D[i,j] = D[i-1,j-1] + D[i-1,j]

else

D[i,j] = D[i-1,j]

return D[T.n][P.n]

for i=0 to T.n

for j=0 to P.n

if P(j) = T(i)

c=1

D[i,j]

D[i-1,j-1] + D[i-1,j]

D[i-1,j]

D[i,j]

D[i-1,j-1]

D[i-1,j]

# DATA CENTER - ESERCIZIO PD PAG 5

$n$ : giorni

$x_i$ : quantità di dati da gestire al giorno

$s_j$ : quantità di dati gestibili al giorno,  $s_1 > s_2 > s_3 \dots$

Massimizzare la quantità di dati gestibili in totale.

Idea per la soluzione:  $p(i, j)$  massima quantità di dati gestibili il giorno  $i$  nel caso in cui sia stato fatto l'ultimo reboot  $j$  giorni prima di  $i$ .

In sostanza  $j$  è come se fosse la capacità di memorare dati del nostro sistema.

X	10	1	7	3
S	8	5	2	1

$$p[i, j] = \begin{cases} \min\{x[i], s[j]\} & i = n \\ \max\{p[i+1, 1], \min\{x[i], s[j]\} + p[i+1, j+1]\} & i < n \end{cases}$$

Caso in cui decidiamo di fare il reboot al giorno  $i$   $\Rightarrow$  non si gestiscono dati e la capacità di elaborarne si riporta da  $s[-1]$ .

se non effettuiamo il reboot si può elaborare una quantità di dati che è il minimo tra  $x[i]$  e  $s[i]$  e quel minimo si somma il passo ricorsivo, consumando 1 giorno ( $i+1$ ) e usando la capacità di elaborazione del giorno seguente ( $j+1$ )

D20

n dadi, dadi i-esimo dotato di  $F[i]$  facce numerate da 1- $F[i]$ , trovare il numero di modi possibili con cui è possibile ottenere una certa somma  $X$ .

es Con 2 dadi e 5 facce, numerate da 2 a 6, ottenere il valore 7.

$$F[1] \quad F[2]$$

$$\sum_{j=1}^i D[i, j]$$

$$D[i, x] = \begin{cases} 0 & x < 0 \\ 1 & x = 0 \\ 0 & x > 0 \\ \sum_{j=1}^{F[i]} D[i-1, x-j] & \text{altrimenti} \end{cases}$$

### Rete di Flusso

$G$  = rete di flusso, si conosce il flusso massimo, e un arco  $(v, u) \in E$  viene aumentato di 1.

Progettare un algoritmo che aggiorna il flusso massimo

$$c[u, v] = c[u, v] + 1$$

$g \leftarrow$  cammino sventante()

$f \leftarrow f + g$

## Esercizio - ABBASSO TRENITALIA - PD pag 5

$X$ : prezzo abbonamento valido per ~~20~~<sup>30</sup> giorni

$d_1, \dots, d_n$ : giorni di viaggio

$f_1, \dots, f_n$ : tariffa intera per il viaggio d  
minimizzare il costo totale del viaggio.

IDEA DELLA SOLUZIONE: si definisce  $P(i)$  come  
il costo minimo per viaggiare al giorno di  $i$ .

Da notare che i giorni  $d_1, \dots, d_n$  possono anche  
essere non consecutivi. (N.B. ①)

La soluzione al problema è  $P(1)$

$$P(i) = \begin{cases} +\infty & i > n \\ \min \left\{ P(i+1) + f_i, P \left( \min \{ j : d[j] \geq d[i+30] \} + X \right) \right\} & \text{altrimenti} \end{cases}$$

-  $P(i+1) + f_i$ : indica la scelta di pagare la tariffa  
intera.

- altrimenti deve pagare ~~20~~, il costo  $X$  dell'abbonamento  
e poi saltare un ammontare di giorni t.c. mi  
permettano di saltare 30 in totale.

(La soluzione  $P(i+30)$  non funziona per  
(N.B. ①))

## Esercizi

Abbiamo un tubo lungo  $L$ . Si deve tagliare in  $n$  segmenti lunghi rispettivamente  $S[1], S[2], \dots, S[n]$ .

- (1) Scrivere un algoritmo che dati il vettore  $S$  e la lunghezza  $L$ , ritorni il numero massimo di pezzi ottenibili.
- (2) Dimostrarre la correttezza.
- (3) Calcolare il costo.

1) Con la programmazione dinamica si ottiene:

$$G[L, n] = \begin{cases} 0 & h=0 \\ \max \left\{ \begin{array}{l} G[L - S[n], n-1] + 1, \\ G[L, n-1] \end{array} \right\} & \text{altrimenti} \end{cases}$$

dove  $G[L, n]$  è il massimo numero di pezzi che posso fare partendo dalla lunghezza  $L$ .

$F(\text{int}[I][J] G, \text{int } L, \text{int}[I] S, \text{int } n)$

```

if(n=0)
    return 0
if(G[L,n]=1)
    G[L,n]=max{G[L-S[n],n-1]+1,G[L,n+1]}
return G[L,n]

```

complessità  $\Theta(n^2)$  perché potenzialmente si deve riempire tutta la matrice.



E' possibile abbassare la complessità dell'algoritmo effettuando una scelta greedy:

**GREEDY:** Prendere in considerazione sempre la lunghezza più piccola, ordinandole in modo crescente.

DIM: dimostrare che una soluzione ottima contiene sempre la lunghezza più piccola.

μ, ρ :

- ① O è una soluzione ottima.
  - ② I<sub>1</sub> è l'elemento con lunghezza minore dopo il sort.

se  $I_1 \in \mathcal{O}$ : dimostrazione effettuata.

se In \$ o;

sia  $E_j$  la lunghezza del tubo più piccolo in cui costruisco la nuova soluzione.

$$O' = O - \{I_3\} \cup \{I_1\}$$

ovvero dalla soluzione ottima tolgo il suo elemento  $I_j$  e lo sostituisco col mio  $I_i$ .

Essendo  $S[I_i] \subset S[I_j]$  per HP@ questo scelto mi porta ad una soluzione che contiene la lunghezza più piccola, quindi ancora ad una soluzione ottima.

$F(\text{int}[JS], \text{int } L, \text{int } n)$

sort(s)

```
int i=7
```

while  $i < n$  and  $L - S[i] > 0$

$$|L = L - SE_i\rangle$$

$i = i + 1$

return i-1

1

Complexity  $O(n \log n)$

## Esercizi

Scrivere un algoritmo che dato un vettore  $A$  di  $n$  interi ( $n = \text{pari}$ ), ritorna  $\star$  vero se è possibile partizionare  $A$  in  $p$  coppie di elementi che hanno tutte la stessa somma, falso altrimenti.

ESEMPIO:  $A = \{7, 4, 5, 2, 6, 3\}$  può essere partizionato in  $7+2=4+5+3+6$

IDEA DELLA SOLUZIONE: Essendo che si deve cercare una somma comune tra tutti i valori di  $A$  presi a coppie; la coppia con somma massima sarà data da  $(M+m)$  dove  $M$  è il valore massimo e  $m$  è il minimo.

Partendo da quest'idea si può ordinare il vettore  $A$  e accoppiare i valori associando sempre il primo e l'ultimo; ovvero il massimo e il minimo.

$$A = \{7, 4, 5, 2, 6, 3\}$$

$$\begin{array}{cccccc} A & = & \{ & 2 & , & 3 & , & 5 & , & 5 & , & 6 & , & 7 \} \\ & & \boxed{g} & \end{array}$$

DIMOSTRAZIONE = SCATTA GREEDY:

Prendiamo una soluzione ottima, ovvero un vettore che rispetti tutte le somme.

Assumiamo per assurdo che l'elemento maggiore  $M$  sia associato ad un elemento  $M'$  maggiore

del minimo  $m$  ( $m < M'$ ).

Quindi il minore  $m$  è associato con un elemento  $m'$  diverso dal massimo  $M$  ( $m' < M$ ).

Allora  $m + m' < M + M' \Rightarrow$  il che contraddice l'ipotesi che sia una soluzione ottima  $\Rightarrow$  ASSURDO

$$\begin{array}{ccccccc} m & m' & \dots & M' & M \\ \boxed{L} & \boxed{L'} & \dots & \boxed{L'} & \boxed{L} & \end{array}$$

$$m + M' < m' + M \Rightarrow \text{ASSURDO}$$

□

checkPairs (int A[], int n)

sort(A, n)

int s = A[1] + A[n]

Complessità

for i=2 to  $n/2-1$  do

[if  $A[i] + A[n-i+1] \neq s$ ,

[return false

$O(n \log n)$

return true

Da questa soluzione si può ricavare una soluzione in  $O(n)$ , la quale non testa tutte le coppie del vettore ordinato ma calcola la somma totale dei valori e la confronta con la somma del minimo e il massimo moltiplicato per  $n/2$ .

ESEMPIO:

$A = 7, 5, 5, 2, 3, 6$

$n = 6$

$s = 27$

se  $(7+2) \cdot \frac{6}{2} = 27$

return true

else return false

checkPairs (int A[], int n)

int s = 0

for i=1 to n do

[ $s = s + A[i]$ ]

return  $\min(A, n) + \max(A, n) \cdot \frac{n}{2}$

$= s$

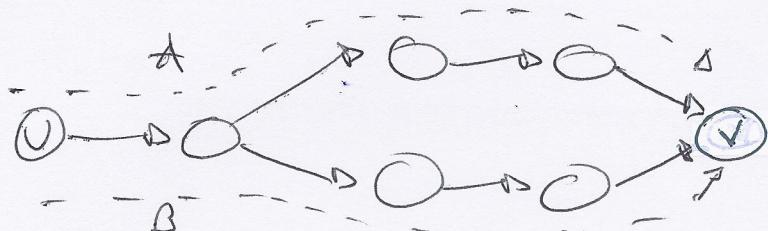


cheese  $\Leftarrow H + H' + H'' + H'''$

## ESERCIZI RETE DI FLUSSO

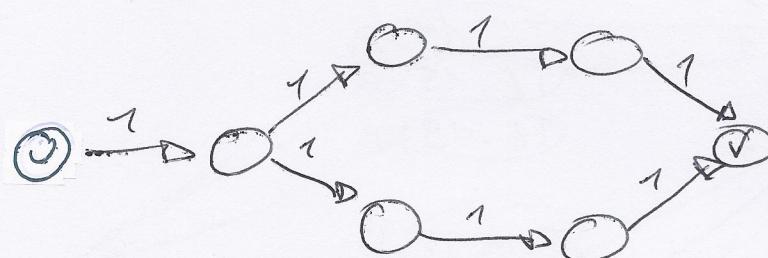
Dato un grafo orientato  $G = (V, E)$  e due vertici  $u, v$ , dire se il grafo è edge-disjoint; ovvero che se esistono due cammini da  $u \rightarrow v$  la loro intersezione degli archi deve essere nulla.

Esempio:



In questo caso il grafo non è edge-disjoint perché c'è il cammino da  $u \rightarrow v$  A che condivide un arco con il cammino da  $u \rightarrow v$  B.

Soluzione: trasformare il grafo in una rete di flusso in modo tale da rendere  $u$  la sorgente,  $v$  il pozzo e la capacità sugli archi pari a 1.



Questo implica che ogni volta che viene cercato un cammino partente da  $s \rightarrow p$ , l'arco verrà assegnato permanentemente a quel cammino.

Se l'algoritmo ritorna una funzione di flusso con più flussi possibili, allora è edge-disjoint, falso altrimenti.

### 1.8 McDonald

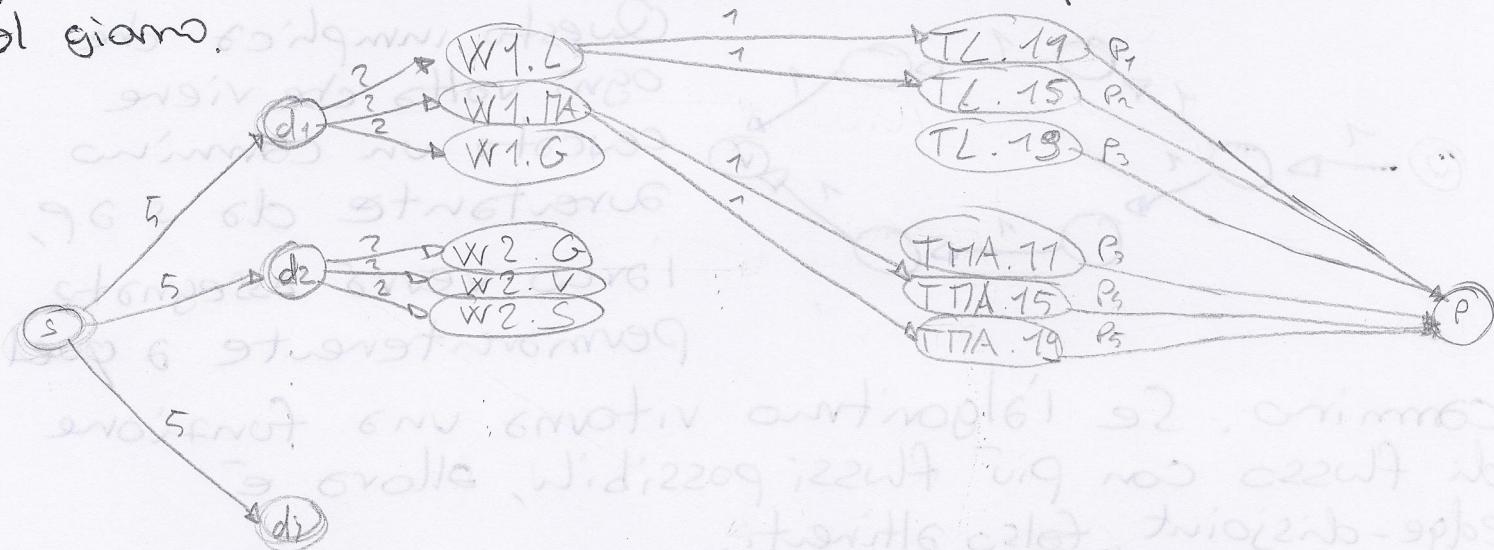
Gestire un McDonald non è semplice. La giornata lavorativa è suddivisa in 3 turni da quattro ore, dalle 11 alle 23. Il management ha stabilito che per ognuno dei turni settimanali contenuti nell'insieme  $T$  ( $7 \cdot 3 = 21$  turni totali), una certa quantità di personale è necessaria. Ad esempio, sabato dalle 19 alle 23 c'è la richiesta massima. Dato il turno  $t_i \in T$ , la richiesta di personale è  $p_i$ . Avete a disposizione un insieme  $D$  di dipendenti; ogni dipendente  $d_j \in D$  dichiara un insieme di turni  $n_j \subseteq T$  in cui non può lavorare. Ad esempio,  $d_j$  non può lavorare nei turni  $\{t_1, t_7, t_{21}\}$ . Per contratto aziendale, ogni lavoratore non può lavorare per più di 5 turni. Ogni turno  $t_i$  deve essere coperto esattamente da  $p_i$  personale. Ogni giorno, un dipendente non può lavorare per più di due turni (qualsiasi, anche non consecutivi). Progettare un algoritmo che produca uno scheduling che illustri, per ogni turno, il personale associato e discutere la complessità dell'algoritmo risultante.

3 turni di 4 ore

11 - 15  
15 - 19  
19 - 23 } ogni giorno della settimana =  $T$

Più richiesta personale al turno  $t \in T$

$D$  insieme dei dipendenti. Lavoratore  $d_j$  non può lavorare più di 5 turni alla settimana e non più di 2 turni al giorno.



Ogni dipendente  $d_j$  è collegato al nodo  $W_{j,d}$  dove  $d$  è il giorno. L'arco ha capacità 2 perché può fare al massimo due turni quel giorno. Ogni nodo  $W_{j,d}$  è collegato con al massimo due nodi  $T_{d,h}$ , dove  $d$  è il giorno e  $h$  l'ora di inizio turno.

La sorgente è collegata con un arco di capacità 5 ad ogni dipendente perché ogni dipendente può scegliere al massimo 5 turni settimanali.