

# ① MA SNOB THORON

HOAP  $\rightarrow$

HASH  $\rightarrow$

HUFFMAN

6 - 7 PUNT

Limite	Risultato	Soluzione
$k = l$	$T(n) = \Theta(n^{\log_b a} \log(n))$	
$k = 0$	$T(n) = \Theta(n^{\log_b a})$	se $\exists k > 0 \mid \lim_{n \rightarrow \infty} \frac{f(n)}{n^{\log_b a-\varepsilon}} = k$
$\lim_{n \rightarrow \infty} \frac{f(n)}{n^{\log_b a}} = k$		$T(n) = \Theta(f(n))$
$k = \infty$		se $\exists k > 0 \mid \lim_{n \rightarrow \infty} \frac{f(n)}{n^{\log_b a-\varepsilon}} = k$ se $\exists k, 0 < k < 1 \mid a f\left(\frac{n}{b}\right) \leq kf(n)$
		CASO 3 - COND. DI REGOLARITÀ

Cosa devo dimostrare

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

$$T(n) = \Omega(f(n))$$

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

Che calcoli devo fare

$$T(n) \leq d \cdot f(n)$$

$$T(n) \geq d \cdot f(n)$$

$$T(n) \leq \geq d \cdot f(n)$$

CLASSE SOTTO → RISOLVO CON FORMULA

RICORSIONE

Domanda 10 Si consideri la ricorrenza

$$T(n) = T\left(\frac{4n}{5}\right) + \frac{n}{2} + \log n$$

(0) CASO 1  $\rightarrow \Theta$

(1) CASO 2  $\rightarrow K$

Soluzione: Si può utilizzare il master theorem dato che la ricorrenza è nella forma CASO 3  $\rightarrow \infty$

$$\left( T(n) = aT\left(\frac{n}{b}\right) + f(n) \right) \xrightarrow{\text{STO}} \text{CASO 3} \rightarrow \infty$$

con  $a = 1$ ,  $b = 5/4$  e  $f(n) = n/2 + \log n$ .

Vale che  $n^{\log_b a} = n^{\log_{5/4} 1} = n^0 = 1$ , quindi

$$f(n) = \Omega(n^{\log_b a+\epsilon}) = \Omega(n^\epsilon)$$

CASO 3  $\rightarrow$  PÙ LUNGO

per  $0 < \epsilon < 1$ .

Inoltre vale la condizione di regolarità  $a f(n/b) \leq kf(n)$  per qualche  $0 < k < 1$ . Infatti

$$\begin{aligned} a f(n/b) &= f(4n/5) \\ &= 2n/5 + \log 4n/5 \\ &= 2n/5 + \log n - \log(5/4) \\ &< k(n/2 + \log n) \end{aligned}$$

BASTA ANDARE UNA SEMPRE

che risulta soddisfatta per  $k > 4/5$  (quindi  $k/2 > 2/5$ ) e  $n$  abbastanza grande. Quindi  $T(n) = \Theta(n + \log n) = \Theta(n)$ .

PÙ CAPITANO CIO

$$\Theta \rightarrow T(n) \leq \underline{d} \cdot f(n)$$

FALISCA LA CONDIZIONE

$d > 0$ ,  $n \in \mathbb{N}$

$$(5 \leq 3) \rightarrow \text{NON SI PUÒ}$$

↓ RAPPORTE INIZIALE (FOLIO)

$$T\left(\frac{4m}{3}\right) + \frac{m}{2} + \log(m)$$

$$T(m) \leq d(m)$$

$$f\left(\frac{4m}{3}\right) + \frac{m}{2} + \log(m) \leq d(m)$$

RISOLVIMENTO PER DIREZIONE  
RISOLVIMENTO PER DIREZIONE,  $\forall m > 0$   
 $C > 0$

$$\underline{T(m-1) \leq d \cdot f(m-1)}$$

$$T\left(\frac{4}{3} \cdot m - 1\right) + \frac{m-1}{2} + \log(m-1) \leq d(m-1)$$

STANDARD

DIMOSTRAZIONE

Soluzione: Si ha che

$$\Omega(f(n)) = \{g(n) \mid \exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 \leq cg(n) \leq f(n)\} \rightarrow \text{DEFINIZIONE}$$

Se  $f(n) = \Omega(g(n))$  e  $g(n) = \Omega(h(n))$  allora esistono  $c_1, c_2 > 0$ ,  $n_1, n_2 \in \mathbb{N}$  tali che per ogni  $n \geq n_1$

$$0 \leq c_1 g(n) \leq f(n) \quad (1)$$

e per ogni  $n \geq n_2$

$$0 \leq c_2 h(n) \leq g(n) \quad (2)$$

Ne consegue che per ogni  $n \geq \max\{n_1, n_2\}$ , moltiplicando (1) per  $c_1$  si ha

$$0 \leq c_1 c_2 h(n) \leq c_1 g(n) \leq f(n)$$

NOTA PERTO INSERITO ...

ovvero, indicato con  $n_0 = \max\{n_1, n_2\}$  e  $c = c_1 c_2$ , per ogni  $n \geq n_0$ ,

$$0 \leq ch(n) \leq f(n)$$

Domanda A (8 punti) Si consideri la seguente funzione ricorsiva con argomento un intero  $n \geq 0$

3° RT

```

val(n)
  if n <= 2
    return 1
  else
    return val(n-1) + val(n-2) + val(n-3)
  
```

$$T(n-1) + 2\underline{T(n-2)} \\ T(n-2) + T(n-2)$$

Determinare la ricorrenza che esprime la complessità della funzione e mostrare che la soluzione è  $\Omega(2^n)$ .  
La complessità è anche  $O(2^n)$ ? Motivare le risposte.

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

2 / 0

**Soluzione:** Nel caso ricorsivo, si effettuano tre chiamate ricorsive, una con argomento  $n - 1$  e due con argomento  $n - 2$ , e inoltre due somme (operazioni di tempo costante) si ottiene quindi

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

Dimostriamo con il metodo di sostituzione che  $T(n) = \Omega(2^n)$ , ovvero dimostriamo che esistono  $d > 0$  e  $n_0$  tali che  $T(n) \geq d \cdot 2^n$  per  $n \geq n_0$ . Si procede per induzione su  $n$ :

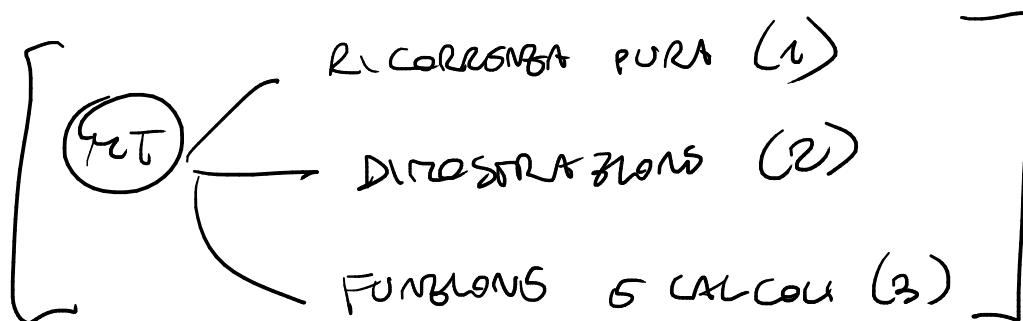
$$\begin{aligned} T(n) &= T(n-1) + 2T(n-2) + c && [\text{dalla definizione della ricorrenza}] \\ &\geq d2^{n-1} + 2d2^{n-2} + c && [\text{ipotesi induttiva}] \\ &\geq d2^{n-1} + 2d2^{n-2} && [\text{poiché } c > 0] \\ &= d(2^{n-1} + 2 \cdot 2^{n-2}) && 2^{n-1} + 2 \cdot 2^{n-2} \\ &= d2^n && = 2^{n-1} + 2^{n-1} \\ & && = 2 \cdot 2^{n-1} = \\ & && = 2^{n-1+1} = 2^n \rightarrow d2^n \end{aligned}$$

qualunque siano  $d$  e  $n_0$ .

Vale anche  $T(n) = O(2^n)$ , ovvero esistono  $d > 0$  e  $n_0$  tali che  $T(n) \leq d \cdot 2^n$  per un'opportuna costante  $d > 0$  e  $n \geq n_0$ . Ai fini della prova induttiva è opportuno dimostrare che  $T(n) \leq d(2^n - 1)$ . Infatti:

$$\begin{aligned} T(n) &= T(n-1) + 2T(n-2) + c && [\text{dalla definizione della ricorrenza}] \\ &\leq d(2^{n-1} - 1) + 2d(2^{n-2} - 1) + c && [\text{ipotesi induttiva}] \\ &= d(2^{n-1} + 2 \cdot 2^{n-2}) - 3d + c \\ &= d2^n - 3d + c \\ &\leq d2^n \end{aligned}$$

dove l'ultima diseguaglianza vale se  $-3d + c \leq 0$ , e quindi per  $d \geq c/3$ .



### Esempio 2

Si consideri una tabella hash di dimensione  $m = 9$  gestita con indirizzamento aperto e che si basa su  $h_1(x) = k \bmod m$  e su  $h_2(x) = 1 + k \bmod (m-2)$ . Si descriva come avviene l'inserimento delle chiavi in sequenza: 12, 3, 22, 14, 38.

### Soluzione

Le due funzioni hash sono  $h_1(x) = k \bmod 9$  e su  $h_2(x) = 1 + k \bmod 7$ . Ricordiamo che  $h(k, i) = (h_1(k) + i * h_2(k)) \bmod 9$ . Dunque:

0	
1	
2	38
3	12
4	22
5	14
6	
7	3
8	

1.  $h(12, 0) = (12 \bmod 9) + 0 * (1 + 12 \bmod 7) = 3$   
Inserisco la chiave 12 in posto 3
2.  $h(3, 0) = (3 \bmod 9) + 0 * (1 + 3 \bmod 7) = 3 \rightarrow \text{collistione}$   
 $h(3, 1) = (3 \bmod 9) + 1 * (1 + 3 \bmod 7) = 7$   
Inserisco la chiave 3 in posto 7 perché prima ho avuto una collisione; ho incrementato i di 1
3.  $h(22, 0) = (22 \bmod 9) + 0 * (1 + 12 \bmod 7) = 4$   
Inserisco la chiave 22 in posto 4
4.  $h(14, 0) = (14 \bmod 9) + 0 * (1 + 12 \bmod 7) = 5$   
Inserisco la chiave 14 in posto 5
5.  $h(38, 0) = (38 \bmod 9) + 0 * (1 + 12 \bmod 7) = 2$

Collisioni

DOPPIO HASH

$$f(x) = h_1(x) + i \cdot h_2(x)$$

$$\begin{aligned}
 f(x) &= [h_1(x) + 0 \cdot h_2(x)] \bmod m \\
 &= [k \bmod m] + 0 \cdot (1 + k \bmod 7) \rightarrow \text{caso 1} \\
 &\quad \underbrace{\{[k \bmod m] + 1 \cdot (h_2(x))\}}_{h_1} \bmod m
 \end{aligned}$$

LINKED  
LISTS

Domanda 34 Si consideri una tabella hash di dimensione  $m = 8$ , gestita mediante chaining (liste di trabocco) con funzione di hash  $h(k) = k \bmod m$ . Si descriva in dettaglio come avviene l'inserimento della sequenza di chiavi: 14, 10, 22, 18, 19.

Laurea Informatica - Università di Padova

Paolo Baldan

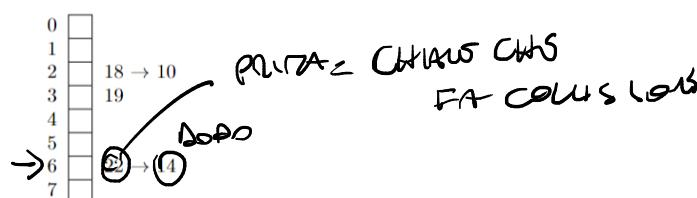
PUNZIONE DI HASH

②

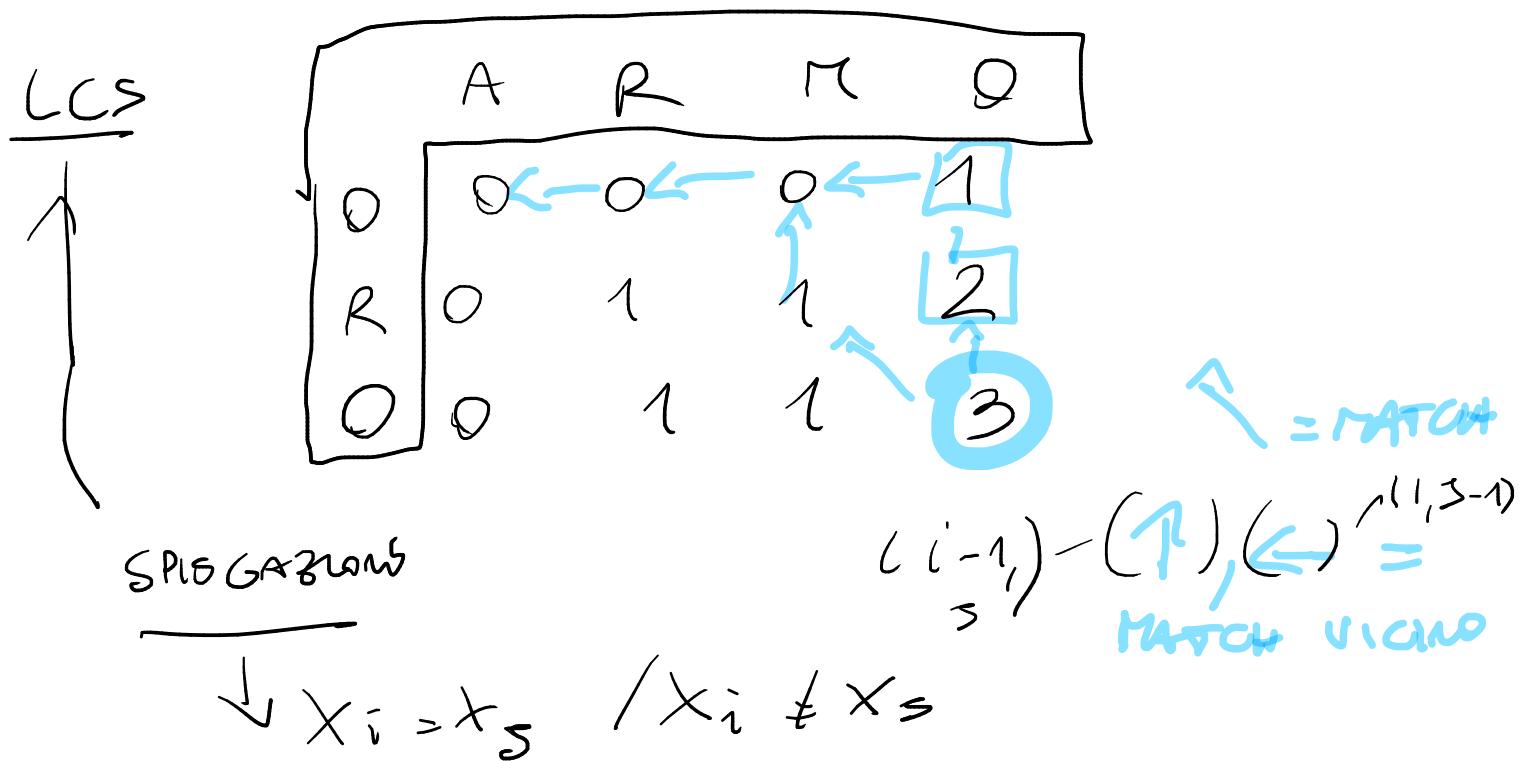
26

Algoritmi e Strutture Dati - 2018/2019

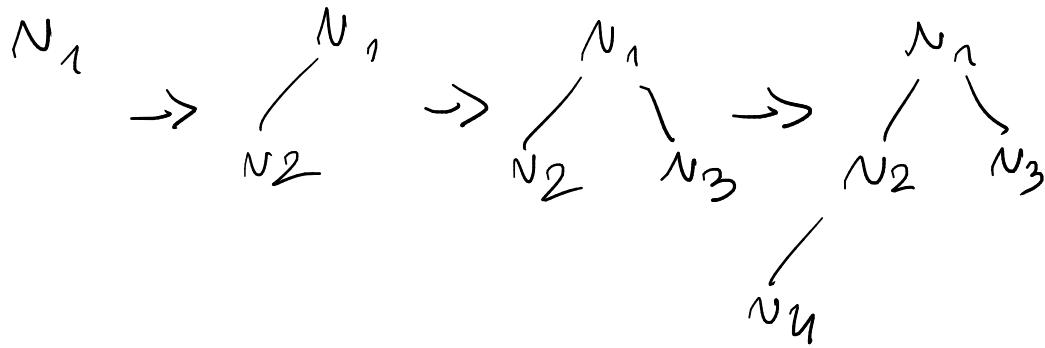
Soluzione: Si ottiene



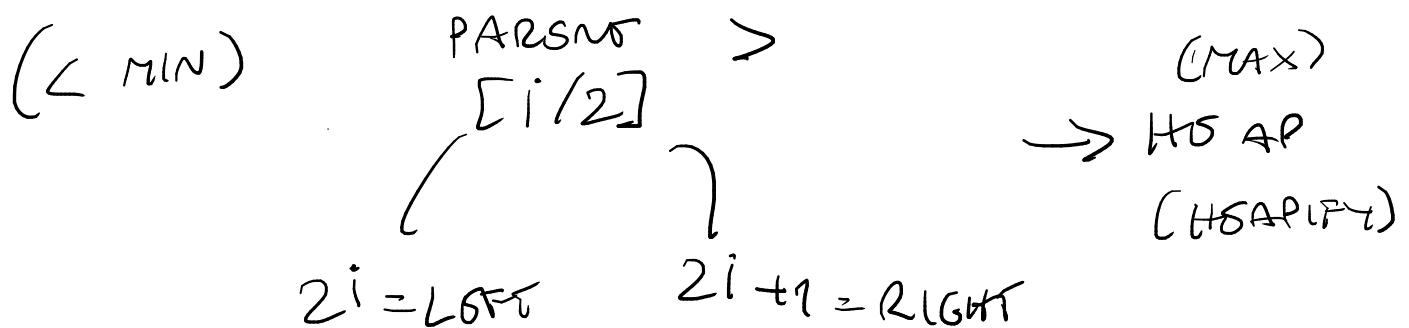
HUFFMAN → BNG CHILLING



**Domanda 25** Dare la definizione di max-heap. Dato un array  $A[1..12]$  con sequenza di elementi [60, 6, 45, 95, 30, 24, 15, 80, 19, 38, 21, 70] si indichi il risultato della procedura BuildMaxHeap applicata ad  $A$ . Si descriva sinteticamente come si procede per arrivare al risultato.



FUNCTION  $\rightarrow$  HSAF (CREATE ORGANIZZAZIONE A MIN / MAX HEAP)



MAX-HEAPIFY( $A$ )  $\rightarrow$  (ASSORRE PUNT. 25% CORSO)

MAX-HEAPIFY( $A, i$ )

```

1   $l = \text{LEFT}(i)$ 
2   $r = \text{RIGHT}(i)$ 
3  if ( $l \leq A.\text{heapsiz}$ e) and ( $A[l] > A[i]$ )
4     $max = l$ 
5  else
6     $max = i$ 
7  if ( $r \leq A.\text{heapsiz}$ e) and ( $A[r] > A[max]$ )
8     $max = r$ 
9  if ( $max \neq i$ )
10    $A[i] \leftrightarrow A[max]$ 
11   MAX-HEAPIFY( $A, max$ )

```

BUILD-MAX-HEAP( $A$ )

```

1   $A.\text{heapsiz} = A.length$ 
2  for  $i = \lfloor A.length/2 \rfloor$  down to 1
3    MAX-HEAPIFY( $A, i$ )

```

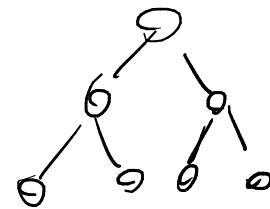
$O(n \log n)$

# FUNZIONI

QUICK

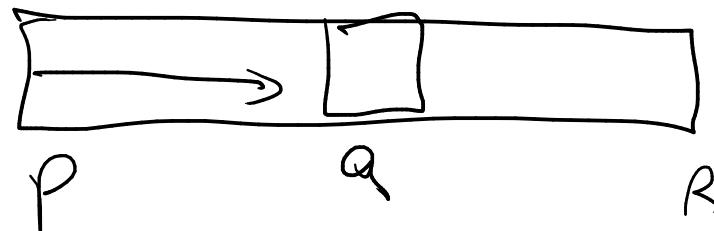
METHODS

LEARN



PARENTHESIS  
(REC.)

$$\rightarrow \lfloor q \rfloor = \lfloor p + r \rfloor / 2$$



SANS

FUNZ1 ( $A, P+1, Q$ )  $\left[ \begin{smallmatrix} (P, Q-1) \\ (Q, R) \end{smallmatrix} \right]$

FUNZ1 ( $A, Q+1, R$ )  $\left[ \begin{smallmatrix} (Q+1, R) \end{smallmatrix} \right]$

PARENTHESIS  $\rightarrow$  ARRAY ORDINATO

ORDINATO

{	$P+1 / P+2 - \dots - Q$	}
	$Q+1 - \dots - R$	

②  $i, j, k \rightarrow A[i] + A[j] = k$

WHILE ( $A[i] + A[j] < > k$ )

$\rightarrow$  ORDINATO  
TRA  
INDICI

IF ( $A[i] + A[j] < k$ )

$i++$

$j--$

ELSE  $[A[i] + A[j] > k]$   $j++$

$i--$

**Domanda 21** Realizzare una funzione Double(A,n) che, dato un array A[1,n] ordinato in senso crescente, verifica se esiste una coppia di indici i, j tali che  $A[j] = 2 * A[i]$ . Restituisce la coppia se esiste e (0,0) altrimenti. Scrivere lo pseudocodice e valutare la complessità.

**Soluzione:**

Il codice può essere:

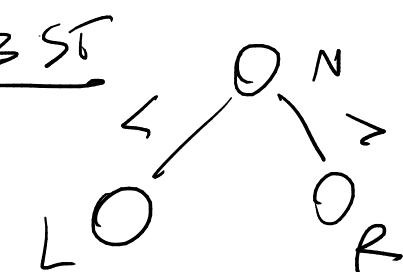
```
double(A,n) {
    n = A.length
    // assume A[1,n] ordinato
    // ritorna una coppia i, j
    // tale che 2*A[i] = A[j]
    i=1
    j=1
    while (i<=n) and (j<=n) and (2*A[i] <= A[j])
        if (2*A[i] > A[j])
            j++
        else
            i++
    if (i<=n) and (j<=n)
        return (i,j)
    else
        return (0,0)
```

ES.  
RISULTATO  
= 0 0 N.  
INDICI

ALGORITMO

BINARY

[P]  
[L] [R]



$x \cdot \text{root} = [\text{root}, \text{left}, \text{right}]$

RIC.  $\rightarrow$  CASO BASE  $(x = T_{\text{root}})$



ES. COME A FOGGIO DI UN ALBERO  $T$

$\boxed{\begin{array}{l} \text{CONTATI } (T) \\ \text{RETURN CONTATI RIC } (T_{\text{root}}) \end{array}}^{\textcircled{1}}$

CONTATI(X)

$\rightarrow$  IF ( $X_{\text{root}} = \text{NIL}$ ) THEN ("...")

CHIAVAN

RVC'

$$\left[ \begin{array}{l} \text{FOCUS\_SX} = \text{CONTANO}(x - \text{LEFT}) \\ \text{FOCUS\_DX} = \text{CONTANO}(x + \text{RIGHT}) \end{array} \right]$$

Return (FOCUS\\_SX + FOCUS\\_DX)

### COMPLESSITÀ

DIVIDI E TI MOLTA  $\rightarrow O(n \log(n))$

MAX/MIN HEAP

ALBORI

$\rightarrow O(\frac{n}{h})$

$\lfloor \lg_2(n) + 1 \rfloor$



### GRISI

① ALGORITMO  $\rightarrow$  IF / WHILE (TORNARE =  
SUSCETTO  
ATTIVITÀ)

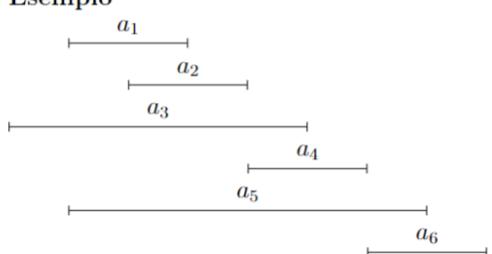
GREEDY-SEL( $S, f$ )  $\rightarrow$  ORDINATO

```

1  n = S.length
2  A = {a1}  $\rightarrow$  PRIMO SELEZIONATO
3  last = 1 // indice dell'ultima attività selezionata
4  for m = 2 to n
5      if sm  $\geq$  flast
6          A = A  $\cup$  {am}
7          last = m
8  return A

```

Esempio



$$A = a_1$$

$$A = a_1, a_4$$

$$A = a_1, a_4, a_6$$

$$last = 1$$

$$last = 4$$

$$last = 6$$

CUT - AND - PASTS (BUONO!)



SORTEATURA OTTIMA

$$C^* = C^k \cup C_i$$

$$C' = C \setminus C^* = \cancel{\bigcup_{i=1}^{n-1} C_i}$$

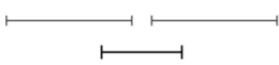
non  
più  
PLANTACCA

#### 8.2.4 Scelte Greedy Alternative

Oltre alla scelta greedy vista precedentemente, ne esistono altre:

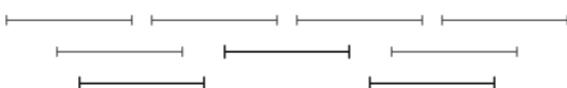
- Scegli l'attività di durata inferiore → non è ottima.

Controesempio:



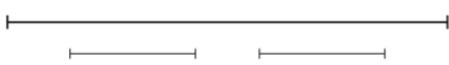
- Scegli l'attività col minor numero di sovrapposizioni → non è ottima.

Controesempio:



- Scegli l'attività che inizia per prima → non è ottima.

Controesempio:



- Scegli l'attività che inizia per ultima → è ottima.

(non è ottima)



Dura meno

ma l'altra

non è

ottima

RISPOSTA (1)

**Domanda B** (7 punti) Si consideri il problema di selezione di attività compatibili:

- Definire il problema.
- Descrivere brevemente l'algoritmo ottimo GREEDY\_SEL visto in classe.
- Fornire un esempio di algoritmo greedy *non* ottimo, motivandone la non ottimalità.

**Esercizio 2** (10 punti) Si consideri il problema di selezione di attività compatibili, con  $n$  attività  $a_1, \dots, a_n$  che ci vengono date attraverso due vettori  $s$  e  $f$  di tempi di inizio e fine, e ordinate per tempo di *inizio* (cioè  $0 < s_1 \leq s_2 \leq \dots \leq s_n$ ).

- Scrivere un algoritmo greedy iterativo che implementa la scelta greedy di selezionare l'attività che inizia per ultima.
- Determinare l'insieme di attività restituito dall'algoritmo al punto (a) quando eseguito sul seguente insieme di 6 attività, caratterizzate dai seguenti vettori  $s$  e  $f$  di tempi di inizio e fine:

$$s = (1, 2, 3, 5, 7, 10) \quad f = (3, 9, 10, 7, 11, 12)$$

- Dimostrare la proprietà di scelta greedy, cioè che esiste soluzione ottima che contiene l'attività che inizia per ultima.



ottima



(inizio a ultimo)



(ultimo a inizio)

**Esercizio 2** (10 punti) Dato un insieme di  $n$  numeri reali positivi e distinti  $S = \{a_1, a_2, \dots, a_n\}$ , con  $0 < a_i < a_j < 1$  per  $1 \leq i < j \leq n$ , un  $(2,1)$ -boxing di  $S$  è una partizione  $P = \{S_1, S_2, \dots, S_k\}$  di  $S$  in  $k$  sottoinsiemi (cioè,  $\bigcup_{j=1}^k S_j = S$  e  $S_r \cap S_t = \emptyset, 1 \leq r \neq t \leq k$ ) che soddisfa inoltre i seguenti vincoli:

$$|S_j| \leq 2 \quad \text{e} \quad \sum_{a \in S_j} a \leq 1, \quad 1 \leq j \leq k.$$

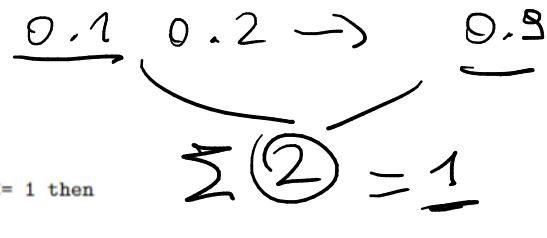
In altre parole, ogni sottoinsieme contiene al più due valori la cui somma è al più uno. Dato  $S$ , si vuole determinare un  $(2,1)$ -boxing che minimizza il numero di sottoinsiemi della partizione.

- (a) Progettare un algoritmo greedy che restituisce un  $(2,1)$ -boxing ottimo in tempo lineare. (Suggerimento: si creino i sottoinsiemi in modo opportuno basandosi sulla sequenza ordinata.)
- (b) Enunciare la proprietà di scelta greedy per l'algoritmo sviluppato al punto precedente e la si dimostri, cioè si dimostri che esiste sempre una soluzione ottima che contiene la scelta greedy.

**Soluzione:**

1. L'idea è provare ad accoppiare il numero più piccolo ( $a_1$ ) con quello più grande ( $a_n$ ). Se la loro somma è al massimo 1, allora  $S_1 = \{a_1, a_n\}$ , altrimenti  $S_1 = \{a_n\}$ . Poi si procede analogamente sul sottoproblema  $S \setminus S_1$ .

```
(2,1)-BOXING(S)
n <- |S|
P <- empty_set
first <- 1
last <- n
while (first <= last)
    if (first < last) and a_first + a_last <= 1 then
        P <- P U {{a_first, a_last}}
        first <- first + 1
    else
        P <- P U {{a_last}}
        last <- last - 1
return P
```



Questo algoritmo scansiona ogni elemento una sola volta, quindi la sua complessità è lineare.

2. La scelta greedy è  $\{a_1, a_n\}$  se  $n > 1$  e  $a_1 + a_n \leq 1$ , altrimenti  $\{a_n\}$ . Ora dimostriamo che esiste sempre una soluzione ottima che contiene la scelta greedy. I casi  $n = 1$  e  $a_1 + a_n > 1$  sono banali, visto che in questi casi ogni soluzione ammessa deve contenere il sottoinsieme  $\{a_n\}$ . Quindi assumiamo che la scelta greedy sia  $\{a_1, a_n\}$ . Consideriamo una qualsiasi soluzione ottima dove  $a_1$  e  $a_n$  non sono accoppiati nello stesso sottoinsieme. Quindi, esistono due sottoinsiemi  $S_1$  e  $S_2$ , con  $a_1 \in S_1$  e  $a_n \in S_2$ . Sostituendo questi due sottoinsiemi con  $S'_1 = \{a_1, a_n\}$  (cioè, la scelta greedy) e  $S'_2 = S_1 \cup S_2 \setminus \{a_1, a_n\}$ ,  $|S'_2| \leq 2$  e, se  $|S'_2| = 2$ , allora  $S'_2 = \{a_s, a_t\}$  con  $a_s \in S_1$  e  $a_t \in S_2$ . Siccome  $a_t$  era precedentemente accoppiato con  $a_n$ , a maggior ragione può essere accoppiato con  $a_s < a_n$ , quindi la nuova soluzione così creata è ammessa e ancora ottima.

**Esercizio 2** (10 punti) Abbiamo  $n$  programmi da eseguire sul nostro computer. Ogni programma  $j$ , dove  $j \in \{1, 2, \dots, n\}$ , ha lunghezza  $\ell_j$ , che rappresenta la quantità di tempo richiesta per la sua esecuzione. Dato un ordine di esecuzione  $\sigma = j_1, j_2, \dots, j_n$  dei programmi (cioè, una permutazione di  $\{1, 2, \dots, n\}$ ), il tempo di completamento  $C_{j_i}(\sigma)$  del  $j_i$ -esimo programma è dato quindi dalla somma delle lunghezze dei programmi  $j_1, j_2, \dots, j_i$ . L'obiettivo è trovare un ordine di esecuzione  $\sigma$  che minimizza la somma dei tempi di completamento di tutti i programmi, cioè  $\sum_{j=1}^n C_j(\sigma)$ .

- (a) Dare un semplice algoritmo greedy per questo problema, e valutarne la complessità.
- (b) Dimostrare la proprietà di scelta greedy dell'algoritmo del punto (a), cioè che esiste un ordine di esecuzione ottimo  $\sigma^*$  che contiene la scelta greedy.

Witwo  $C_{last} + C_s \leq \underline{\sigma}$   
 $C^* = C^* \cup C_s$

**Esercizio 2** (9 punti) Supponiamo di avere un numero illimitato di monete di ciascuno dei seguenti valori: 50, 20, 1. Dato un numero intero positivo  $n$ , l'obiettivo è selezionare il più piccolo numero di monete tale che il loro valore totale sia  $n$ . Consideriamo l'algoritmo greedy che consiste nel selezionare ripetutamente la moneta di valore più grande possibile.

- Fornire un valore di  $n$  per cui l'algoritmo greedy *non* restituisce una soluzione ottima.
- Supponiamo ora che i valori delle monete siano 10, 5, 1. In questo caso l'algoritmo greedy restituisce sempre una soluzione ottima: dimostrare che ogni insieme ottimo  $M^*$  di monete di valore totale  $n$  contiene la scelta greedy.

**Soluzione:**

- Per esempio  $n = 60$ , perché la soluzione ottima è 3 monete da 20, mentre l'algoritmo greedy restituisce 11 monete (una da 50 e 10 da 1).
- Sia  $M^*$  una soluzione ottima. Sia  $x$  il valore maggiore tra 10, 5, e 1 che sia non superiore a  $n$ . Se  $M^*$  contiene una moneta di valore  $x$ , la proprietà è dimostrata. Altrimenti, sia  $M \subseteq M^*$  un insieme di (2 o più) monete di valore totale  $x$  (si osservi che tale insieme esiste sempre quando i valori delle monete sono 10, 5, 1); consideriamo  $M' = M^* \setminus M \cup X$ , dove  $X$  è l'insieme contenente una moneta di valore  $x$ .  $M'$  è un insieme di monete di valore totale  $n$  e di cardinalità inferiore a quella di  $M^*$ : assurdo, quindi questo secondo caso non può verificarsi, e quindi  $M^*$  contiene necessariamente una moneta di valore  $x$ .

moneta più valora

Esercizio: matching sulla linea

Sia  $S = \{s_1, s_2, \dots, s_m\}$  un insieme di punti ordinati sulla retta reale, rappresentanti dei server. Sia  $C = \{c_1, c_2, \dots, c_n\}$  un insieme di punti ordinati sulla retta reale, rappresentanti dei client. Il costo di assegnare un client  $c_i$  ad un server  $s_j$  è  $|c_i - s_j|$ . Fornire un algoritmo greedy che assegna ogni client ad un server distinto e che minimizzi il costo totale (equiv., medio) dell'assegnamento.

$C \vdash S \vdash ) C_i - s_j$   
client / server

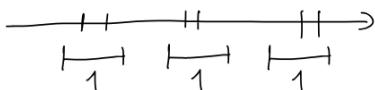
Esercizio: Sia  $X = \{x_1, x_2, \dots, x_m\}$  un insieme di punti ordinati sulla retta reale.

Fornire un algoritmo greedy che determini un insieme  $I$  di cardinalità minima di intervalli chiusi di ampiezza unitaria ( $[a, b] \in I \Rightarrow b-a=1$ ) tale che  $\forall x_i \in X \exists j \in I$  tale che  $x_i \in j$ .

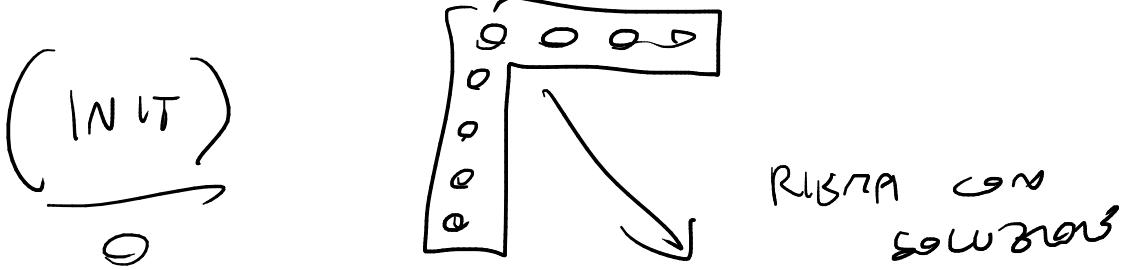
$$\frac{b-a}{1} = 1$$

ASSIGNI  
ODINARIO

Esempio:



# PROG. DINAMICA



$\text{IF } (X[1, \cdot] = 0)$

Ricerca la soluzione

Esercizio 2 (8 punti) Per  $n > 0$ , siano dati due vettori a componenti intere  $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^n$ . Si consideri la quantità  $c(i, j)$ , con  $0 \leq i \leq j \leq n - 1$ , definita come segue:

Si capisce che si scansione in ordine decrescente di colonna perché quando "i" vale 0, allora "j" vale qualcosa

$$c(i, j) = \begin{cases} a_i & \text{se } 0 < i \leq n - 1 \text{ e } j = n - 1, \\ b_j & \text{se } i = 0 \text{ e } 0 \leq j \leq n - 1, \\ c(i-1, j-1) \cdot c(i, j+1) & 0 < i \leq j < n - 1. \end{cases}$$

Si vuole calcolare la quantità  $m = \min\{c(i, j) : 0 \leq i \leq j \leq n - 1\}$ .

1. Si fornisca il codice di un algoritmo iterativo bottom-up per il calcolo di  $m$ .
2. Si valuti la complessità esatta dell'algoritmo, associando costo unitario ai prodotti tra numeri interi e costo nullo a tutte le altre operazioni.

Soluzione:

- (a) Date le dipendenze tra gli indici nella ricorrenza, un modo corretto di riempire la tabella è attraverso una scansione in cui calcoliamo gli elementi in ordine crescente di indice di riga e, per ogni riga, in ordine decrescente di indice di colonna. Il codice è il seguente.

WINGHTS (a)

```

COMPUTE(a, b)
n <- length(a)
m = +infinito
for i=1 to n-1 do
    C[i, n-1] <- a_i
    m <- MIN(m, C[i, n-1])
for j=0 to n-1 do
    C[0, j] <- b_j
    m <- MIN(m, C[0, j])
for i=1 to n-2 do
    for j=n-2 downto i do
        C[i, j] <- C[i-1, j-1] * C[i, j+1]
        m <- MIN(m, C[i, j])
return m

```

ci prendiamo la lunghezza dell'array e inizializziamo il minimo ad infinito (così, qualsiasi prima quantità minima sarà più piccola)

Siccome devo salvare il minimo, ogni volta si fa il confronto tra il minimo attuale e l'indice di  $C[i, j]$  appena salvato.

Ricordandosi che:
- "i" va da 1 ad (n-1), quindi diventa perché abbiamo già inizializzato, (n-2) la scansione
- "j" va da (n-1) a 0 compresi, quindi dato che abbiamo inizializzato, parte da (n-2)

Siccome nuovamente abbiamo un ciclo in cui "i" va ad (n-2) e in quello di inizializzazione andava ad (n-1), la sommatoria è data da  $(n-2)(n-1)/2$

(b)

$$T(n) = \sum_{i=1}^{n-2} \sum_{j=i}^{n-2} 1 = \sum_{i=1}^{n-2} n - 1 - i = \sum_{k=1}^{n-2} k = (n-1)(n-2)/2.$$

BOTTOM-UP  
TOP-DOWN  
=  
ITERATIVO

**Esercizio 2** (9 punti) Data una stringa  $X = x_1, x_2, \dots, x_n$ , si consideri la seguente quantità  $\ell(i, j)$ , definita per  $1 \leq i \leq j \leq n$ :

$$\ell(i, j) = \begin{cases} 1 & \text{se } i = j \\ 2 & \text{se } i = j - 1 \\ 2 + \ell(i+1, j-1) & \text{se } (i < j-1) \text{ e } (x_i = x_j) \\ \sum_{k=i}^{j-1} (\ell(i, k) + \ell(k+1, j)) & \text{se } (i < j-1) \text{ e } (x_i \neq x_j). \end{cases}$$

1. Scrivere una coppia di algoritmi INIT\_L( $X$ ) e REC\_L( $X, i, j$ ) per il calcolo memoizzato di  $\ell(1, n)$ .
2. Si determini la complessità *al caso migliore*  $T_{\text{best}}(n)$ , supponendo che le uniche operazioni di costo unitario e non nullo siano i confronti tra caratteri.

**Soluzione:**

1. Pseudocodice:

```

INIT_L(X)
n <- length(X)
if n = 1 then return 1
if n = 2 then return 2
for i=1 to n-1 do
    L[i,i] <- 1
    L[i,i+1] <- 2
L[n,n] <- 1 (ultimo elemento diagonale inizializzato)
for i=1 to n-2 do
    for j=i+2 to n do
        L[i,j] <- 0
return REC_L(X,1,n)

REC_L(X,i,j)
if L[i,j] = 0 then
    if x_i = x_j then L[i,j] <- 2 + REC_L(X,i+1,j-1)
    else for k=i to j-1 do
        L[i,j] <- L[i,j] + REC_L(X,i,k) + REC_L(X,k+1,j)
return L[i,j]

```

$$l(i, j) = |LCS(X_i, Y_j)|$$

$$l(i, j) = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \quad (\text{caso 0}) \\ l(i-1, j-1) + 1 & \text{se } i, j > 0 \text{ e } x_i = x_j \quad (\text{caso 1}) \\ \max\{l(i, j-1), l(i-1, j)\} & \text{se } i, j > 0 \text{ e } x_i \neq x_j \quad (\text{caso 2}) \end{cases}$$

Alla fine ci interessa calcolare  $l(m, n)$ .

Per calcolare  $l(i, j)$  mi possono servire tre valori:

$$L = \begin{bmatrix} & (i-1, j-1) & (i-1, j) \\ (i, j-1) & \leftarrow & (i, j) \end{bmatrix}$$

Scansione "row-major": riempio la tabella per righe, da sinistra a destra.

Informazione addizionale per costruire la sequenza (vera e propria):

$$b(i, j) = \begin{cases} '↖' & \text{se } x_i = y_j \\ '←' & \text{se } x_i \neq y_j \text{ e } \max = LCS(i, j-1) \\ '↑' & \text{se } x_i \neq y_j \text{ e } \max = LCS(i-1, j) \end{cases}$$

*CALCOLO.*

Pseudocodice

```

LCS(X, Y)
1 m = X.length
2 n = Y.length
3 for i = 0 to m
4     L[i, 0] = 0
5 for j = 0 to n
6     L[0, j] = 0
7 for i = 1 to m
8     for j = 1 to n
9         if x_i = y_j
10            L[i, j] = L[i - 1, j - 1] + 1
11            B[i, j] = '↖'
12        else if L[i - 1, j] ≥ L[i, j - 1]
13            L[i, j] = L[i - 1, j]
14            B[i, j] = '↑'
15        else
16            L[i, j] = L[i, j - 1]
17            B[i, j] = '←'
18 return (L[m, n], B)

```

INIT-LCS( $X, Y$ )

```

1  m = X.length
2  n = Y.length
3  if (m = 0) or (n = 0)
4      return 0
5  for i = 0 to m
6      L[i, 0] = 0
7  for j = 0 to n
8      L[0, j] = 0
9  for i = 1 to m
10     for j = 1 to n
11         L[i, j] = -1
12  return R-LCS(X, Y, m, n)

```

RISULTATI PER

SOLUZIONI

CALCOLA  
SOLUZIONI

R-LCS( $X, Y, i, j$ )

```

1  if L[i, j] = -1
2      if  $x_i = y_j$ 
3          L[i, j] = R-LCS(X, Y, i - 1, j - 1)
4      else if R-LCS(X, Y, i - 1, j) ≥ R-LCS(X, Y, i, j - 1)
5          L[i, j] = L[i - 1, j]
6      else
7          L[i, j] = L[i, j - 1]
8  return L[i, j]

```

Esercizio

$$X = \langle b, d, c, d \rangle$$

$$Y = \langle a, b, c, b, d \rangle$$

Restituisce  $LCS(X, Y)$  e  $|LCS(X, Y)|$

$$L = \begin{matrix} & \begin{matrix} a & b & c & b & d \end{matrix} \\ \begin{matrix} b \\ d \\ c \\ d \end{matrix} & \left[ \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 2 \\ 0 & 0 & 1 & 2 & 2 & 2 \\ 0 & 0 & 1 & 2 & 2 & 3 \end{matrix} \right] \end{matrix} \quad B = \begin{matrix} & \begin{matrix} a & b & c & b & d \end{matrix} \\ \begin{matrix} b \\ d \\ c \\ d \end{matrix} & \left[ \begin{matrix} \uparrow & \nearrow & \leftarrow & \nwarrow & \leftarrow \\ \uparrow & \uparrow & \uparrow & \uparrow & \nearrow \\ \uparrow & \uparrow & \nwarrow & \leftarrow & \uparrow \\ \uparrow & \uparrow & \uparrow & \uparrow & \nwarrow \end{matrix} \right] \end{matrix}$$

$$LCS(X, Y) = \langle b, c, d \rangle \quad |LCS(X, Y)| = 3$$

GAUSS

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

$$\rightarrow \sum_{i=1}^{K=m-2} k = \frac{(m-2)(m-1)}{2}$$

**Esercizio 2** (9 punti) Sia  $n > 0$  un intero. Si consideri la seguente ricorrenza  $M(i, j)$  definita su tutte le coppie  $(i, j)$  con  $1 \leq i \leq j \leq n$ :

$$M(i, j) = \begin{cases} 1 & \text{se } i = j, \\ 2 & \text{se } j = i + 1, \\ M(i+1, j-1) \cdot M(i+1, j) \cdot M(i, j-1) & \text{se } j > i + 1. \end{cases}$$

Ricorsivo

1. Scrivere una coppia di algoritmi INIT\_M(n) e REC\_M(i, j) per il calcolo memoizzato di  $M(1, n)$ .
2. Calcolare il numero esatto  $T(n)$  di moltiplicazioni tra interi eseguite per il calcolo di  $M(1, n)$ .

ricorrenza  $c(i, j)$  definita per ogni coppia di valori  $(i, j)$  con  $1 \leq i, j \leq n$ :

$$c(i, j) = \begin{cases} a_j & \text{if } i = 1, 1 \leq j \leq n, \\ a_{n+1-i} & \text{if } j = n, 1 < i \leq n, \\ c(i-1, j) \cdot c(i, j+1) \cdot c(i-1, j+1) & \text{altrimenti.} \end{cases}$$

Ricorsivo  $\rightarrow 1$  solo

1. Si fornisca il codice di un algoritmo iterativo bottom-up COMPUTE\_C( $A$ ) che, data in input la stringa  $A$  restituisca in uscita il valore  $c(n, 1)$ .
2. Si valuti il numero esatto  $T_{CC}(n)$  di moltiplicazioni tra interi eseguite dall'algoritmo sviluppato al punto (1).

**Esercizio 2** (8 punti) Per  $n > 0$ , siano dati due vettori a componenti intere  $\mathbf{a}, \mathbf{b} \in \mathbf{Z}^n$ . Si consideri la quantità  $c(i, j)$ , con  $0 \leq i \leq j \leq n - 1$ , definita come segue:

$$c(i, j) = \begin{cases} a_i & \text{se } 0 < i \leq n - 1 \text{ e } j = n - 1, \\ b_j & \text{se } i = 0 \text{ e } 0 \leq j \leq n - 1, \\ c(i-1, j-1) \cdot c(i, j+1) & 0 < i \leq j < n - 1. \end{cases}$$

Si vuole calcolare la quantità  $m = \min\{c(i, j) : 0 \leq i \leq j \leq n - 1\}$ .

1. Si fornisca il codice di un algoritmo iterativo bottom-up per il calcolo di  $m$ .
2. Si valuti la complessità esatta dell'algoritmo, associando costo unitario ai prodotti tra numeri interi e costo nullo a tutte le altre operazioni.

**Esercizio 2** (8 punti) Per  $n > 0$ , siano dati due vettori a componenti intere  $\mathbf{a}, \mathbf{b} \in \mathbf{Z}^n$ . Si consideri la quantità  $c(i, j)$ , con  $0 \leq i \leq j \leq n - 1$ , definita come segue:

$$c(i, j) = \begin{cases} a_i & \text{se } 0 < i \leq n - 1 \text{ e } j = n - 1, \\ b_j & \text{se } i = 0 \text{ e } 0 \leq j \leq n - 1, \\ c(i-1, j) \cdot c(i, j+1) & 0 < i \leq j < n - 1. \end{cases}$$

Si vuole calcolare la quantità  $M = \max\{c(i, j) : 0 \leq i \leq j \leq n - 1\}$ .

1. Si fornisca il codice di un algoritmo iterativo bottom-up per il calcolo di  $M$ .

**Esercizio 2** (9 punti) Data una stringa  $X = x_1, x_2, \dots, x_n$ , si consideri la seguente quantità  $\ell(i, j)$ , definita per  $1 \leq i \leq j \leq n$ :

$$\ell(i, j) = \begin{cases} 1 & \text{se } i = j \\ 2 & \text{se } i = j - 1 \\ 2 + \ell(i+1, j-1) & \text{se } (i < j-1) \text{ e } (x_i = x_j) \\ \sum_{k=i}^{j-1} (\ell(i, k) + \ell(k+1, j)) & \text{se } (i < j-1) \text{ e } (x_i \neq x_j). \end{cases}$$

1. Scrivere una coppia di algoritmi INIT\_L( $X$ ) e REC\_L( $X, i, j$ ) per il calcolo memoizzato di  $\ell(1, n)$ .
2. Si determini la complessità *al caso migliore*  $T_{best}(n)$ , supponendo che le uniche operazioni di costo unitario e non nullo siano i confronti tra caratteri.

**Esercizio 2** (9 punti) Data una stringa di numeri interi  $A = (a_1, a_2, \dots, a_n)$ , si consideri la seguente ricorrenza  $z(i, j)$  definita per ogni coppia di valori  $(i, j)$  con  $1 \leq i, j \leq n$ :

$$z(i, j) = \begin{cases} a_j & \text{if } i = 1, 1 \leq j \leq n, \\ a_{n+1-i} & \text{if } j = n, 1 < i \leq n, \\ z(i-1, j) \cdot z(i, j+1) \cdot z(i-1, j+1) & \text{altrimenti.} \end{cases}$$

1. Si fornisca il codice di un algoritmo iterativo bottom-up  $Z(A)$  che, data in input la stringa  $A$  restituisca in uscita il valore  $z(n, 1)$ .
2. Si valuti il numero esatto  $T_Z(n)$  di moltiplicazioni tra interi eseguite dall'algoritmo sviluppato al punto (1).

**Esercizio 2** (8 punti) Date due stringhe  $X = \langle x_1, x_2, \dots, x_m \rangle$  e  $Y = \langle y_1, y_2, \dots, y_n \rangle$ , si consideri la seguente quantità  $\ell(i, j)$ , definita per ogni coppia di valori  $i, j$  con  $0 \leq i \leq m$  e  $0 \leq j \leq n$ :

$$\ell(i, j) = \begin{cases} 1 & \text{se } i = 0 \text{ o } j = 0 \\ 3\ell(i, j-1) & \text{se } i, j > 0 \text{ e } x_i = y_j \\ 2\ell(i-1, j-1) - \ell(i-1, j) & \text{se } i, j > 0 \text{ e } x_i \neq y_j. \end{cases}$$

Si vuole calcolare la quantità  $q = \max\{\ell(i, j) : 0 \leq i \leq m, 0 \leq j \leq n\}$ .

- (a) Scrivere un algoritmo bottom-up per il calcolo di  $q$ .
- (b) Determinare la complessità *esatta* dell'algoritmo, supponendo che le uniche operazioni di costo unitario e non nullo siano i confronti tra caratteri.