24/11 →

ALGORITMI

DP + GREEDY

FUNZIONI

HEAP

ARRAY (MERGE)

ALBERI

Domanda A (7 punti) Dare la definizione di max-heap. Dato un insieme S di elementi, memorizzato in parte in un min-heap A e in parte in un max-heap B, entrambi non vuoti, dare un algoritmo `min(A,B)` per trovare il minimo di S nelle due situazioni seguenti:

→ (a) ogni elemento di A è minore o uguale a ogni elemento di B;

(b) ogni elemento di B è minore o uguale a ogni elemento di A.

In entrambi i casi scrivere lo pseudo-codice e valutare la complessità.

A → MIN

B → MAX

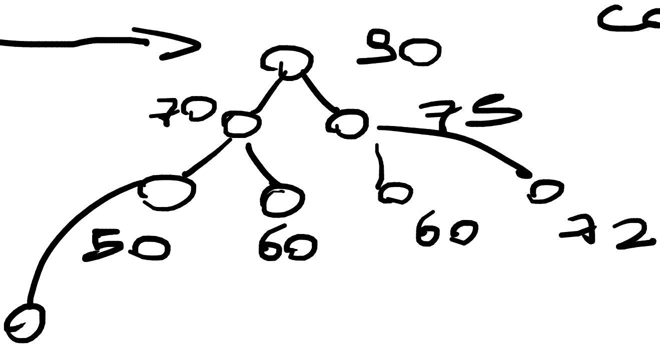
A[1...n]

MIN

$\boxed{1/2}$

MAX

MAX-HEAP → ALBERO BINARIO QUASI COMPLETO



Domanda A (7 punti) Dare la definizione di max-heap. Dato un insieme S di elementi, memorizzato in parte in un min-heap A e in parte in un max-heap B, entrambi non vuoti, dare un algoritmo `min(A,B)` per trovare il minimo di S nelle due situazioni seguenti:

(a) ogni elemento di A è minore o uguale a ogni elemento di B;

(b) ogni elemento di B è minore o uguale a ogni elemento di A.

In entrambi i casi scrivere lo pseudo-codice e valutare la complessità.

A → MIN - HEAP $\quad \forall A \le B \rightarrow \min(A)$

B → MAX - HEAP

```
min_caso_a(A, B)
1  return A[1]
```

(b) $\forall b \le \forall A$

HEAP

MIN (A,B)

- N = HEAPSIZE (IN CASO = STESSA SIZE) ....

1. n = B.HEAPSIZE

IPOTESI: min (S) → B

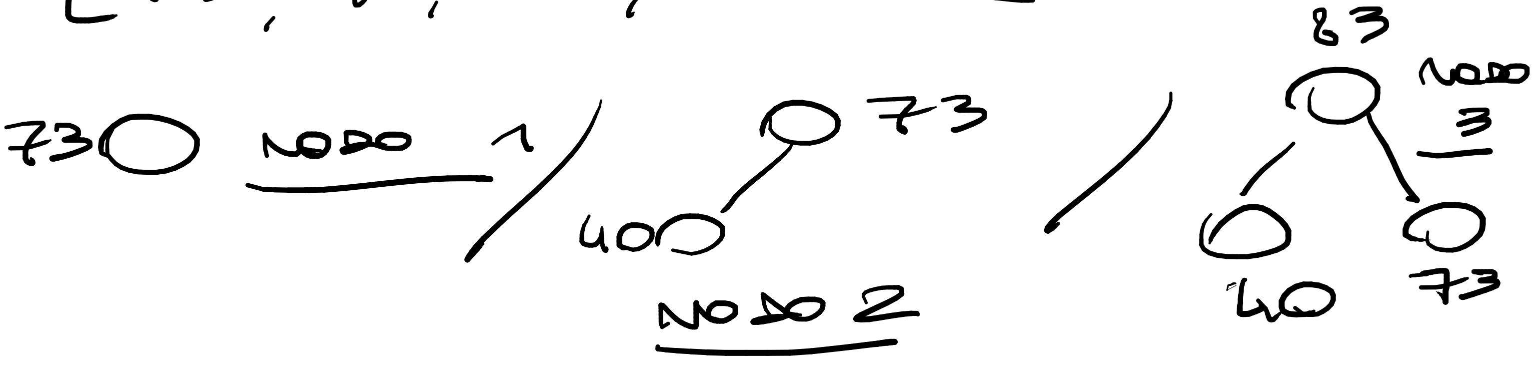
- B → MAX - HEAP (n) → $\lceil n/2 \rceil$

2. MIN = B [n/2] +1
3. FOR I = N/2 TO N
4. IF B[i] < MIN
5. MIN = B[i]
6. RETURN MIN

$\Theta(n)$

- INSERIMENTO NELLO HEAP { MIN / MAX }

[73, 40, 83, 72, ...]



PRIORITÀ → HEAP (PRIORITY QUEUE)

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

(a) Definire il problema.

(b) Descrivere brevemente l'algoritmo ottimo GREEDY-SEL visto in classe.

(c) Fornire un esempio di algoritmo greedy *non* ottimo, motivandone la non ottimalità.

GREEDY { ALGORITMO CHE FA ASSEGNAZ. INTELLIGENTE / OTTIMO → SOTTOSTR. OTTIMA }

→ GREEDY

(A) ATTIVITÀ COMPATIBILI

→ $S = \{a_1, \dots a_n\}$ $\quad a_i \begin{cases} S_i \\ F_j \end{cases}$

$\forall \{a_i, a_j\} \in A^*$

S = START

F = FINE

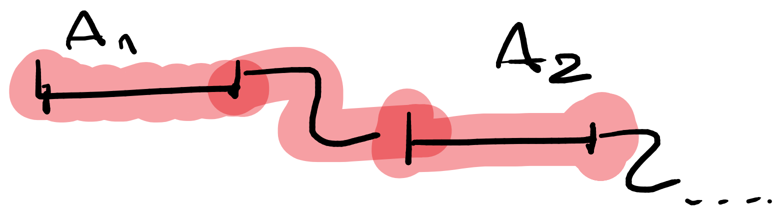
A (*) — INSIEME OTTIMO

(B)

```
GREEDY-SEL(s, f, n)
// Assunzione: le attività sono ordinate per tempo di fine crescente
A = {a₁}
k = 1
for i = 2 to n
    if s[i] ≥ f[k]
        A = A ∪ {aᵢ}
        k = i
return A
```

→ GREEDY

ORDINATE



ORDINAMENTO → "FAI MONO STRADA" ≤ END

(C) SOTTOSTR. OTTIMA → ESEMPIO "CHE VA BENE"

$$S^* = \{ a_i \in S : s_i \geq f_j \}$$

$$S' = a_i \qquad S^* = S^* \cup S' \text{ OTTIMO}$$

$$S^j = S \setminus S^* \rightarrow \text{GREEDY}$$

$$\forall a_i \rightarrow s_i \geq f_j$$

$$s_{i+1} \geq f_{j-1} \ldots$$

Esercizio 1 (10 punti) Sia dato un array `V[1..n]` i cui valori rappresentano la variazione giornaliera del valore di un titolo azionario. È noto che il titolo è stato prima perdita, con valori sempre negativi, poi ha iniziato a oscillare in giorni *consecutivi* tra valori positivi e negativi, e infine si è stabilizzato su valori positivi (dunque nella sequenza non ci possono essere due giorni positivi seguiti da un negativo). Realizzare un algoritmo *divide et impera* `Split(V)` che individua il giorno in cui il titolo ha iniziato a essere stabile su valori positivi, ovvero il minimo indice $i \in [1, n]$ tale che per ogni $j \geq i$ vale $V[j] > 0$. Se il titolo non si stabilizza su valori positivi, ritornare 0. Ad es., se l'array `V = [-1,-2,2,-1,6,3]` l'indice da tornare sarà 5, mentre invece per `V=[-1,-2,2,-1,6,-3]` si ritornerà 0. Fornire lo pseudocodice di `Split(V)`, motivarne la correttezza e individuarne la complessità. Si assuma che non ci siano valori nulli.

SPLIT → SE C'È → 4

V [-1 , -2 , 2 , -1 , 6 , 3]   SE NO → Ø   V[i] → SPLIT

NEG.   ALTERN.   POS.

V → DIVIDE ET IMPERA

L — log

M = Q — PIVOT / ELEM. DI METÀ

R — log

L+R/2 → M

SPLIT → V [- - - - -]

L R

① SE TUTTO RIGHT POSITIVO

SX — INIZIO DX

② SE RIGHT CONTIENE ALMENO UN NEGATIVO → SOLUZIONE A DX

SPLIT(V)

RETURN SPLIT (V, 1, V.LENGTH)

V

SPLIT-REC (V, LEFT, RIGHT)

1. IF (LEFT == RIGHT)
    IF (V[LEFT] > 0)
        RETURN LEFT
    ELSE RETURN 0

"NON TROPPO DIVIDE ET IMPERA"

V [-3, -2, 5, 7] → SPLIT

$M = \lfloor (LEFT + RIGHT)/2 \rfloor$

- CASO 1: $V[M] \leq 0$ → DX DI MID

IF $V[MID] \leq 0$
    RETURN SPLIT_REC (V, M+1, RIGHT)

V → [-5; -7; -3; 1, -2, 3, 4]

TRADOTTA → FORSE IL PRIMO INDICE POSITIVO È SPLIT!

$V[M] \leq 0$

> 0

RIGHT ≤ 0

RIGHT > 0

CASO 2 $\rightarrow V[M] > 0$;

$V[RIGHT] \leq 0$

TRA M E RIGHT

SPLIT ?

$[3 \; ; \; \ldots - 5] \ldots$

IF $V[RIGHT] \leq 0$

RETURN SPLIT (V, M+1, RIGHT)

- CASO 3 : $\left\{ \begin{array}{l} V[M] > 0 \\ V[RIGHT] > 0 \rightarrow SX \end{array} \right\}$

$[\underbrace{1 \; ; \; -3}_{LEFT} \; ; \; 5 \; ; \; 7]$

RESULT = SPLIT_RIC (V, LEFT, M-1)

IF RESULT == 0

RETURN MID

ELSE

RETURN RESULT

SPLIT

$-1 \; ; \; -2 \; ; \; \boxed{3} \; ; \; 4 \; ; \; 5$

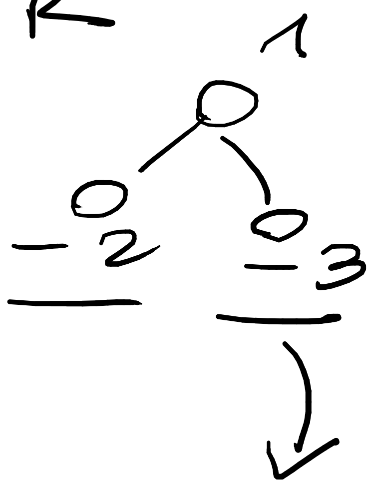
```
SPLIT-OPTIMIZED(V)
return SPLIT-OPT-REC(V, 1, V.length)

SPLIT-OPT-REC(V, left, right)
if left == right
    if V[left] > 0
        return left
    else
        return 0

if left > right
    return 0

mid = ⌊(left + right) / 2⌋

// Caso 1: V[mid] ≤ 0 → soluzione a destra di mid
if V[mid] ≤ 0
    return SPLIT-OPT-REC(V, mid + 1, right)

// Caso 2: V[mid] > 0 e V[right] ≤ 0 → soluzione tra mid e right
if V[right] ≤ 0
    return SPLIT-OPT-REC(V, mid + 1, right)

// Caso 3: V[mid] > 0 e V[right] > 0 → verifica a sinistra
result = SPLIT-OPT-REC(V, left, mid - 1)
if result == 0
    return mid
else
    return result
```

$\forall j \geq i, V[j] > 0$

DEBUG → D&I

V [-3 ; -2 ; $\boxed{1}$; 4 ; 5]

L M R



[-3 ; $\boxed{-2}$; 1]

L M R

[-3 , -2] L == R

```
SPLIT(V)
return SPLIT-REC(V, 1, V.length)

SPLIT-REC(V, left, right)
if left == right
    if V[left] > 0
        return left
    else
        return 0

mid = ⌊(left + right) / 2⌋

// Verifica se tutta la metà destra è positiva
all_positive_right = true
for i = mid + 1 to right
    if V[i] ≤ 0
        all_positive_right = false
        break

if all_positive_right
    // Soluzione nella metà sinistra o è mid+1
    result = SPLIT-REC(V, left, mid)
    if result == 0
        return mid + 1
    else
        return result
else
    // Soluzione nella metà destra
    return SPLIT-REC(V, mid + 1, right)
```

$\Theta \log(n)$

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

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

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

(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.

LCS

X → ABACO

Y → AB|BECCDARLO

PROG. DINAMICA → $O(n^2)$

→ $O(n^3)$

→ CICLI FOR MATRICI

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

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

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

(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.

_ BOTTOM _ UP (1 ALGORITMO SOLO CON CICLI) → ITERATIVO

_ TOP - DOWN (RICORSIVO) → MEMOIZZATO

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

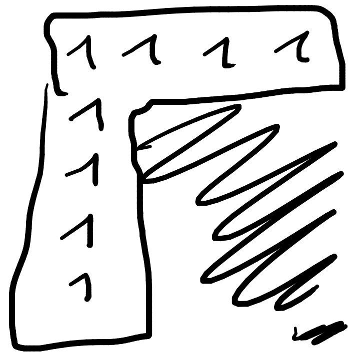
$$\ell(i,j) = \begin{cases} 1 & \text{se } i = 0 \text{ o } j = 0 \\ 3\ell(i, j-1) & \text{se } i, j > 0 \text{ e } x_i = y_j \\ 2\ell(i-1, j-1) - \ell(i-1, j) & \text{se } i, j > 0 \text{ e } x_i \neq y_j. \end{cases}$$

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

(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.

```
COMPUTE(X, Y, m, n)

1. Q = - INFTY
2. FOR i = 0 TO M
3.     L[i, 0] = 1
4. FOR J = 0 TO N
5.     L[0, J] = 1
6. FOR i = 1 TO M - 1
7.     FOR J = N - 1 DOWNTO i
```



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

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

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

(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.

8. IF (X[i] == Y[j])
9. L[i, j] = 3 * L[i, j-1]
10. m = MAX (m, L[i, j-1])
11. ELSE

L[i, j] = 2 * L[i-1, j-1] - L[i-1, j]

m = MAX (m, L[i-1, j])

```
COMPUTE-Q(X, Y, m, n)
// Inizializzazione
for i = 0 to m
    L[i, 0] = 1
for j = 0 to n
    L[0, j] = 1

q = 1  // Massimo iniziale (dai casi base)

// Calcolo bottom-up
for i = 1 to m
    for j = n-1 downto I
        if X[i] == Y[j]
            L[i, j] = 3 * L[i, j-1]
        else
            L[i, j] = 2 * L[i-1, j-1] - L[i-1, j]

        // Aggiornamento del massimo
        if L[i, j] > q
            q = L[i, j]

return q
```

m = MAX (m, L[i, j-1])

m = MAX (m, L[i-1, j])

```
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 i 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)

ESEMPIO TIPO ...

```
COMPUTE-Q(X, Y, m, n)
  // Inizializzazione
  for i = 0 to m
      L[i, 0] = 1
  for j = 0 to n
      L[0, j] = 1

  q = 1  // Massimo iniziale (dai casi base)

  // Calcolo bottom-up
  for i = 1 to m -1
      for j = n-1 downto I
          if X[i] == Y[j]
              L[i, j] = 3 * L[i, j-1]
          else
              L[i, j] = 2 * L[i-1, j-1] - L[i-1, j]

          // Aggiornamento del massimo
          if L[i, j] > q
              q = L[i, j]

  return q
```

② → costo (?)...

$\Theta(n^2) \to$

$$T(n) = \sum_{i=1}^{m-1} \sum_{j=i}^{n-1} 1$$

$$= \sum_{i=1}^{n-1} n-1-i$$

$$\underbrace{\sum_{i=1}^{n} k \to \frac{n(n-1)}{2}}_{\text{GAUSS}} = \sum_{k=1}^{n-1} k \quad \frac{(n-1)(n-1)}{2}$$

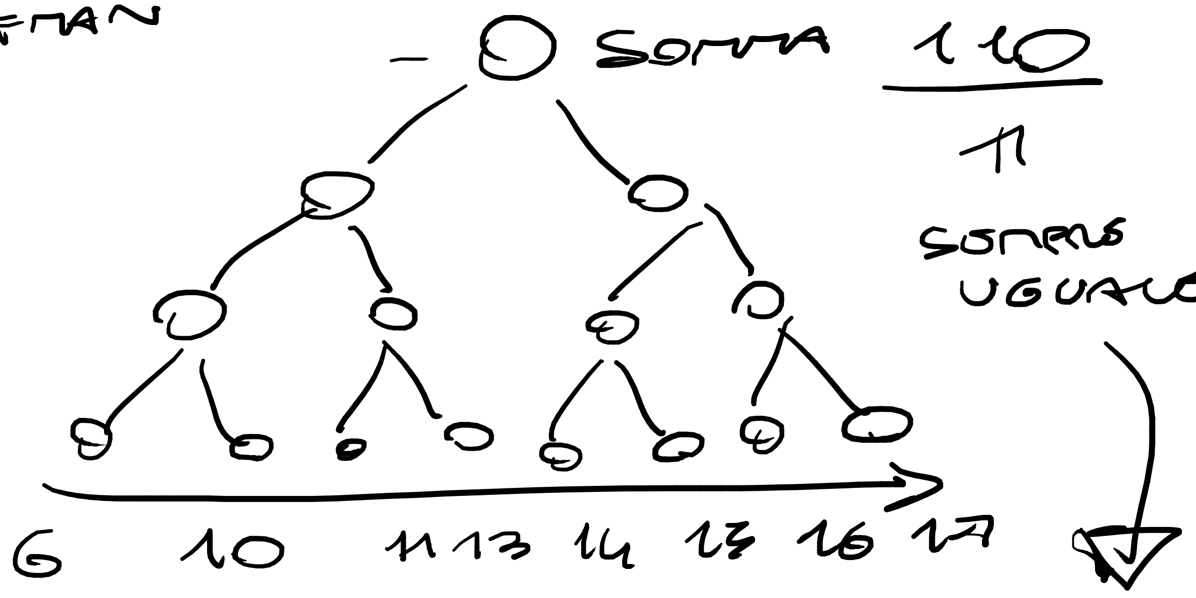
Domanda B (6 punti) Indicare, in forma di albero binario, il codice prefisso ottenuto tramite l'algoritmo di Huffman per l'alfabeto $\{a, b, c, d, e, f\}$, supponendo che ogni simbolo appaia con le seguenti frequenze:

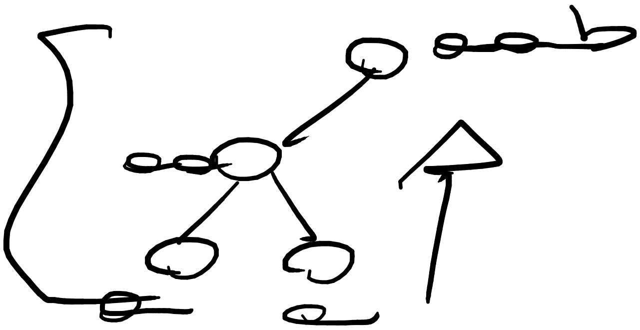
a	b	c	d	e	f
11	6	13	35	10	25

Spiegare brevemente il processo di costruzione del codice.

2

HUFFMAN





LIBERO DA PREFISSI

Domanda B (6 punti) Indicare, in forma di albero binario, il codice prefisso ottenuto tramite l'algoritmo di Huffman per l'alfabeto $\{a, b, c, d, e, f\}$, supponendo che ogni simbolo appaia con le seguenti frequenze:

a	b	c	d	e	f
11	6	13	35	10	25

Spiegare brevemente il processo di costruzione del codice.