PROG. DINAMICA → <u>TEORIA</u>

<u>ESERCIZI</u> → TUTTI GLI ES
↳ RACCOLTA PD + GREEDY ...

SOLUZIONE → OTTIMIZZATA
↳ SALVATA DENTRO MATRICI O STRUTTURE

- ITERATIVI < BOTTOM-UP / TOP-DOWN

FIB = FIB(n-1)

- RICORSIVI → MEMOIZZAZIONE

ABACO
ABBA

↑

LCS =

LONGEST COMMON SUBSEQUENCE

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

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

RICORRENZA DA COSTI

COSTO

Alla fine ci interessa calcolare $l(m,n)$.
Per calcolare $l(i,j)$ mi possono servire tre valori:

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

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

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

$$b(i,j) = \begin{cases} '\nwarrow' & \text{se } x_i = y_j \\ '\leftarrow' & \text{se } x_i \neq x_j \text{ e } max = LCS(i,j-1) \\ '\uparrow' & \text{se } x_i \neq y_j \text{ e } max = LCS(i-1,j) \end{cases}$$

→ DOVE TROVO LE SOLUZIONI

		A	B	A	C	O
	0	0	0	0	0	
A	0	1	1	2	2	2
B	0	1	2	2	2	2
B	0	1	2	2	2	2
A	0	1	2	3	3	3

= (*)

→ 6 PUNTI = CALCOLO MANUALE LCS

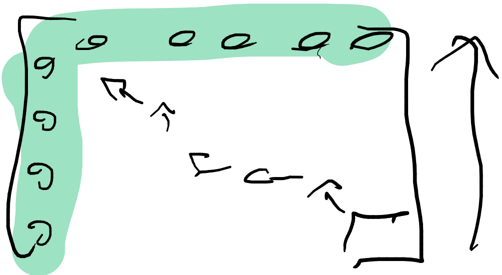
SOLITAMENTE, TANTI DEI SEGNI PER OGNI LCS (*) CI DANNO LA SOL.

Pseudocodice

```
LCS(X, Y)
m = X.[length]
n = Y.length
for i = 0 to m
        L[i, 0] = 0
for j = 0 to n
        L[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i = y_j
                        L[i, j] = L[i − 1, j − 1] + 1
                        B[i, j] = '↖'
                else if L[i − 1, j] ≥ L[i, j − 1]
                        L[i, j] = L[i − 1, j]
                        B[i, j] = '↑'
                else
                        L[i, j] = L[i, j − 1]
                        B[i, j] = '←'
return (L[m, n], B)
```

→ n^2

SOLUZIONE



BOTTOM UP

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

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

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

$$L = \begin{array}{c} \\ \\ b \\ d \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccccc} & a & b & c & b & d \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 2 \\ 0 & 0 & 1 & 2 & 2 & 2 \\ 0 & 0 & 1 & 2 & 2 & 3 \end{bmatrix} \end{array} \qquad B = \begin{array}{c} \\ b \\ d \\ c \\ d \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & b & d \end{array} \\ \begin{bmatrix} \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow \\ \uparrow & \uparrow & \uparrow & \uparrow & \nwarrow \\ \uparrow & \uparrow & \nwarrow & \leftarrow & \uparrow \\ \uparrow & \uparrow & \uparrow & \uparrow & \nwarrow \end{bmatrix} \end{array}$$

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

Pseudocodice Memoizzato

MEMOIZZAZIONE

```
INIT-LCS(X, Y)
m = X.length
n = Y.length
if (m = 0) or (n = 0)
        return 0
for i = 0 to m
        L[i, 0] = 0
for j = 0 to n
        L[0, j] = 0
for i = 1 to m
        for i = 1 to n
                L[i, j] = −1
return R-LCS(X, Y, m, n)
```

SALVA SOLUZIONE IN MATRICE RICORSIVAMENTE

```
R-LCS(X, Y, i, j)
if L[i, j] = −1
        if x_i = y_j
                L[i, j] = R-LCS(X, Y, i − 1, j − 1)
        else if R-LCS(X, Y, i − 1, j) ≥ R-LCS(X, Y, i, j − 1)
                L[i, j] = L[i − 1, j]
        else
                L[i, j] = L[i, j − 1]
return L[i, j]
```

→ RICORRENZA COSTO

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:

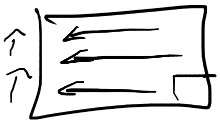
$$c(i,j) = \begin{cases} a_i & \text{se } 0 < i \leq n-1 \text{ e } j = n-1, \\ b_j & \text{se } i = 0 \text{ e } 0 \leq j \leq n-1, \\ c(i-1, j-1) \cdot c(i, j+1) & 0 < i \leq j < n-1. \end{cases}$$

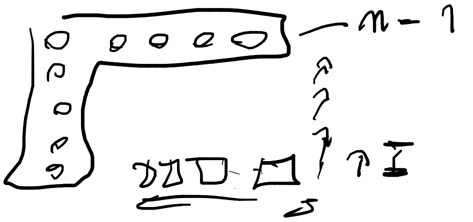
→ RICORRENZA SUL COSTO

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.

```
1. COMPUTE(a, b)
n = length(a)
// calcolo minimo
→ metto un num. più grande possibile per essere sicuri di salvare il primo minimo
m = +infty

for i = 1 to n-1
        C[i, n-1] = a_i
        m = MIN(m, C[i, n-1])

// Uso un indice solo per rendere più efficiente; mi scrive lui che indice usare..

for j = 0 to n-1
        C[0, j] = b_j
        m = MIN(m, C[0, j])

for i = 1 to n - 2
        for j = n - 2 downto i

        C[i,j] = C[i - 1, j - 1] * C[i, j + 1]
        m = MIN(m, C[i,j])

return m
```





$c(i-1, j-1) \cdot c(i, j+1) \quad 0 < i \leq j < n - 1.$

tà $m = \min\{c(i, j) : 0 \leq i \leq j \leq n - 1\}$.

```
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
```

OPTIMIZE

LCS(X, Y)

```
m = X.[length]
n = Y.length
for i = 0 to m
      L[i,0] = 0
for j = 0 to n
      L[0,j] = 0
for i = 1 to m
      for j = 1 to n
            if x_i = y_j
                  L[i,j] = L[i-1,j-1] + 1
                  B[i,j] = '↖'
            else if L[i-1,j] ≥ L[i,j-1]
                  L[i,j] = L[i-1,j]
                  B[i,j] = '↑'
            else
                  L[i,j] = L[i,j-1]
                  B[i,j] = '←'
return (L[m,n], B)
```

CALCOLO SOLUZIONE...

BOTTOM-UP

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 $0 \le i \le j \le n-1$, definita come segue:

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

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

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.

$$\sum_{i=1}^{n-2} \sum_{j=i}^{(n-2)} 1 \to (n-2) \cdot i$$

```
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
```

COSTO SOLUZIONE

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

$n - 1 - i$

90%

$$\left[\sum_{i=1}^{n} k \to \frac{n(n-1)}{2} \quad \text{GAUSS} \right]$$

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

COSTO ESATTO!

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

$$M(i,j) = \begin{cases} 2 & \text{se } i = j, \\ 3 & \text{se } j = i+1, \\ M(i+1,j-1) \cdot M(i+1,j) + M(i,j-1) & \text{se } j > i+1. \end{cases}$$

$\Rightarrow$ RICORRENZA COSA

(a) Si scriva una coppia di algoritmi INIT_M(n) e REC_M(i,j) per il calcolo memoizzato di $M(1,n)$.

(b) Si calcoli il numero esatto $T(n)$ di moltiplicazioni tra interi eseguite per il calcolo di $M(1,n)$.

RICORSIONE

① INIZIALIZZA PER SOLU

② CALCOLA RICORSI VALORE SOLUZIONE

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

$$M(i,j) = \begin{cases} 2 & \text{se } i = j, \\ 3 & \text{se } j = i+1, \\ M(i+1,j-1) \cdot M(i+1,j) + M(i,j-1) & \text{se } j > i+1. \end{cases}$$

```
INIT_M(n)
    // esiste un solo indice → i=j, sono la stessa cosa
    if n = 1 return 2
    // esistono due indici → i/i+1,j, ritorno 3
    if n = 2 return 3

    for i = 1 to n (oppure da 1 to n-1)
        M[i, i] = 2
        M[i, i + 1] = 3

    // due cicli for per mettere a 0 i lati della matrice
    for i = 1 to n - 1 (oppure n - 2)
        for j = i + 2 to n // perché j = i + 1, già "inizializzato" prima
            M[i, j] = 0
```

— MINI- ASSOMAZ.

```
j=1  j=2  j=3  j=4  j=5
i=1   2    3    0    0    0
i=2   -    2    3    0    0
i=3   -    -    2    3    0
i=4   -    -    -    2    3
i=5   -    -    -    -    2
```

```
REC_M(i, j)
```

$$M(i,j) = \begin{cases} 2 & \text{se } i = j, \\ 3 & \text{se } j = i+1, \\ M(i+1,j-1) \cdot M(i+1,j) + M(i,j-1) & \text{se } j > i+1. \end{cases}$$

```
if M[i, j] = 0
    M[i, j] = REC_M(i + 1, j - 1) * REC_M(i + 1, j) + REC_M(i, j - 1)

return M[i, j]
```

(a)
```
INIT_M(n)
if n=1 then return 2
if n=2 then return 3
for i=1 to n-1 do
    M[i,i] = 2
    M[i,i+1] = 3
M[n,n] = 2
for i=1 to n-2 do
    for j=i+2 to n do
        M[i,j] = 0
return REC_M(1,n)
```

—2 / 3

→ 0

```
REC_M(i,j)
if M[i,j] = 0 then
    M[i,j] = REC_M(i+1,j-1) * REC_M(i+1,j) + REC_M(i,j-1)
return M[i,j]
```

```
INIT-LCS(X, Y)
m = X.length
n = Y.length
if (m = 0) or (n = 0)
    return 0
for i = 0 to m
    L[i, 0] = 0
for j = 0 to n
    L[0, j] = 0
for i = 1 to m
    for i = 1 to n
        L[i, j] = −1
return R-LCS(X, Y, m, n)
```

→ 0

→ −1

```
R-LCS(X, Y, i, j)
if L[i, j] = −1
   if x_i = y_j
       L[i, j] = R-LCS(X, Y, i − 1, j − 1)
   else if R-LCS(X, Y, i − 1, j) ≥ R-LCS(X, Y, i, j − 1)
       L[i, j] = L[i − 1, j]
   else
       L[i, j] = L[i, j − 1]
return L[i, j]
```

RICORR. COST

(b) Si calcoli il numero esatto $T(n)$ di moltiplicazioni tra interi eseguite per il calcolo di $M(1, n)$.

Soluzione:

(a)
```
INIT_M(n)
if n=1 then return 2
if n=2 then return 3
for i=1 to n-1 do
   M[i,i] = 2
   M[i,i+1] = 3
M[n,n] = 2
for i=1 to n-2 do
   for j=i+2 to n do
      M[i,j] = 0
return REC_M(1,n)
```

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

GAUSS

```
REC_M(i,j)
if M[i,j] = 0 then
   M[i,j] = REC_M(i+1,j-1) * REC_M(i+1,j) + REC_M(i,j-1)
return M[i,j]
```

1

0

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

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

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

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.

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
 for i=1 to n-2 do
   for j=i+2 to n do
     L[i,j] <- 0
 return REC_L(X,1,n)
```

Prendo la lunghezza della stringa iniziale

Similmente, anche n=1 e n=2 vengono dai due casi base (array di 1/2 elementi max), quindi o ha un elemento oppure ne ha due soli

(ultimo elemento diagonale inizializzato)

Una volta inizializzato secondo i casi base, tutto il resto della matrice viene riempita di zeri. Poi, si considera, essendo una scansione per diagonale, noi partiamo dal basso e riempiamo solo la parte sopra delle diagonali principali, ovviamente riempita anche questa di zeri.

```
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]
```

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

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

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 $z(n, 1)$. (vedo (n,1) e la scansione è per colonna)
2. Si valuti il numero esatto $T_Z(n)$ di moltiplicazioni tra interi eseguite dall'algoritmo sviluppato al punto (1).

Reverse column major:

a11	a12	a13
a21	a22	a23
a31	a32	a33

Soluzione:

1. Date le dipendenze tra gli indici nella ricorrenza, un modo corretto di riempire la tabella è attraverso una scansione "reverse column-major", in cui calcoliamo gli elementi della tabella in ordine decrescente di indice di colonna e, all'interno della stessa colonna, in ordine crescente di indice di riga. Il codice è il seguente.

```
Z(A)
n = length(A)
for i=1 to n do
   z[1,i] = a_i
   z[i,n] = a_{n+1-i}
for j=n-1 downto 1 do
   for i=2 to n do
      z[i,j] = z[i-1,j] * z[i,j+1] * z[i-1,j+1]
return z[n,1]
```

Piuttosto che usare per l'inizializzazione due indici, se ne usa solo uno.
Quando si ha "j", si sa che "i" vale 1, da cui i=j e quindi [1, i]
Quando invece si ha "n+1-i" come indice, si vede che "j" è =n e quindi avremo che per [i, n] avremo [n+1-i].

Vedendo che "j" parte da n e invece "i" parte da 1, per effettuare una scansione giusta per colonne (avendo che la prima colonna/riga è stata inizializzata dal caso base), allora "j" parte da (n-1) piuttosto che da (n) e "i" parte da 2 piuttosto che da (1)

Esercizio 2 (11 punti) Una *longest common substring* di due stringhe X e Y è una sottostringa di X e di Y di lunghezza massima. Si vuole progettare un algoritmo efficiente per calcolare la lunghezza di una longest common substring. Per semplicità si assuma che entrambe le stringhe di input abbiano stessa lunghezza n.

→ LCS

BRUTE-FORCE

(a) Qual è la complessità dell'algoritmo esaustivo che analizza tutte le possibili sottostringhe comuni?

(b) Assumendo di conoscere un algoritmo che determina se una stringa di m caratteri è sottostringa di un'altra stringa di n caratteri in tempo $O(m+n)$, come si può modificare l'algoritmo del punto precedente per renderlo più efficiente?

(c) Progettare un algoritmo di programmazione dinamica più efficiente di quello del punto precedente. Sono richiesti relazione di ricorrenza sulle lunghezze (senza dimostrazione) e algoritmo bottom-up. (Suggerimento: considerare la lunghezza della longest common substring dei prefissi $X_i = \langle x_1, \ldots, x_i \rangle$ e $Y_j = \langle y_1, \ldots, y_j \rangle$ *che termina con* x_i *e* y_j, rispettivamente.)

→ LCS

[X, Y → STRINGHE — $\underline{X \subseteq Y}$

SUBSTRING → LCS

(LIS / PALINDROMO)

n, m

$\simeq O(n^2)$ →

(b) $O(n^2)$ → X, → Y

$O(m+n)$

$X \subseteq Y$ →

MAX = MAX + $\underline{C[i,j]}$

MAX ATTUALE

(b)

```
def longest_common_substring(X, Y):
    n = len(X)
    L = [[0] * (n + 1) for _ in range(n + 1)]  # Matrice di lunghezza
(n+1)x(n+1)
    maxLength = 0  # Variabile per salvare la lunghezza massima trovata

    for i in range(1, n + 1):
        for j in range(1, n + 1):
            if X[i - 1] == Y[j - 1]:  # Se i caratteri corrispondono
                L[i][j] = L[i - 1][j - 1] + 1
                maxLength = max(maxLength, L[i][j])  # Aggiorna la lunghezza
massima
            else:
                L[i][j] = 0  # Altrimenti resettiamo a 0

    return maxLength
```

Domanda (5 punti): Indicare, in forma di albero binario, il codice prefisso ottenuto tramite l'algoritmo di Huffman per l'alfabeto $\{a, b, c, d, e\}$ supponendo che ogni carattere appaia con le seguenti frequenze

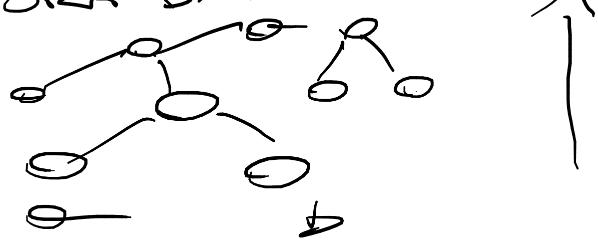
a	b	c	d	e
12	10	13	57	8

→ TREE BY FREQUENCY SORT CHARS

Spiegare il processo di costruzione del codice

HUFFMAN → OTTIMO (COMPRESSO)

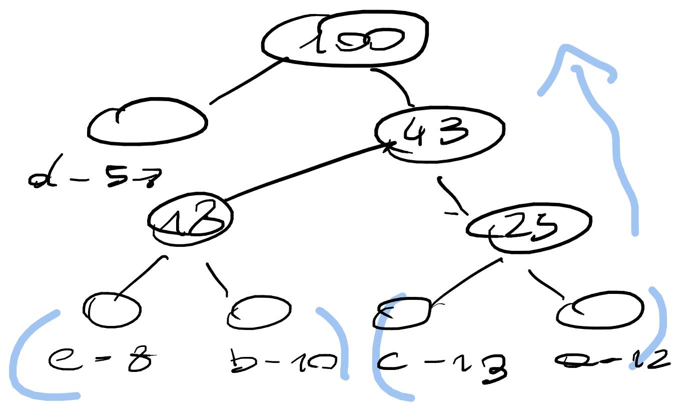
→ LIBERA DA PREFISSI



Domanda (5 punti): Indicare, in forma di albero binario, il codice prefisso ottenuto tramite l'algoritmo di Huffman per l'alfabeto $\{a, b, c, d, e\}$ supponendo che ogni carattere appaia con le seguenti frequenze

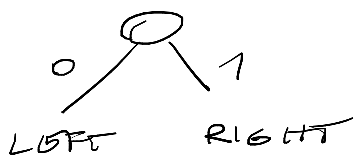
a	b	c	d	e
12	10	13	57	8

Spiegare il processo di costruzione del codice



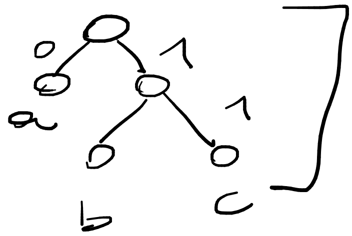
Esercizio (9 punti): Si consideri un file definito sull'alfabeto $\{a, b, c\}$, con frequenze $f(a), f(b), f(c)$. Per ognuna delle seguenti codifiche determinare, se esiste, un opportuno assegnamento di valori alle 3 frequenze per cui l'algoritmo di Huffman restituisce tale codifica, oppure argomentare che tale codifica non è mai ottenibile

1) $e(a) = 0, \quad e(b) = 10, \quad e(c) = 11$
2) $e(a) = 1, \quad e(b) = 0, \quad e(c) = 11$
3) $e(a) = 10, \quad e(b) = 01, \quad e(c) = 00$



PREMESSA

1



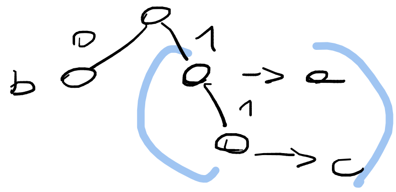
BUONO

$f(a) = 50 \quad f(b) = 25$

$f(c) = 25$

$(f(b), f(c)) < f(a)$

(2) →



→ NON LIBERO DA PREFISSI

ALBERO NON-COMPLETO

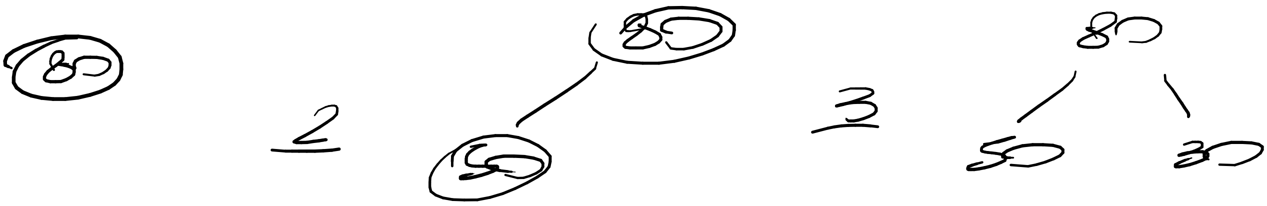
HEAP → $\boxed{\text{QUASI}}$-COMPLETO

DIGRESSIONE

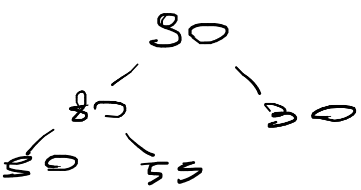
A = { 80, 50, 30, 90, 55 }

MAX-HEAP → PARENT > FIGLI

1

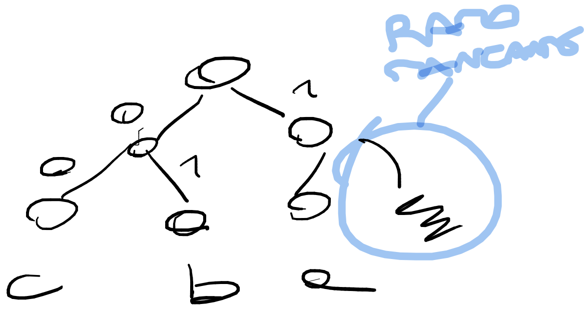


4 / 5



← MAX-HEAP

(3) 10 / 01 / 00
a b c



NON COMPLETO / NON OTTIMO