ALGORITMI → 16/11/2024

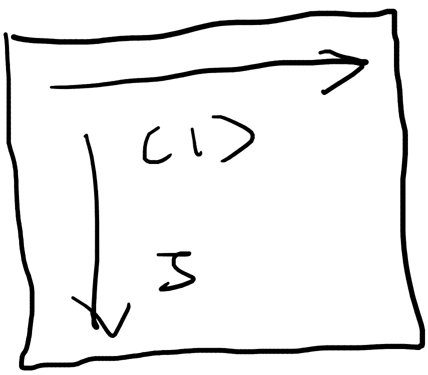
PROG. DINAMICA

↓

SALVARE LA SOLUZIONE BUONA SENZA RICALCOLARLA

- ITER.
- RIC.

ITERATIVO



(SCANSIONE DELLA MATRICE)

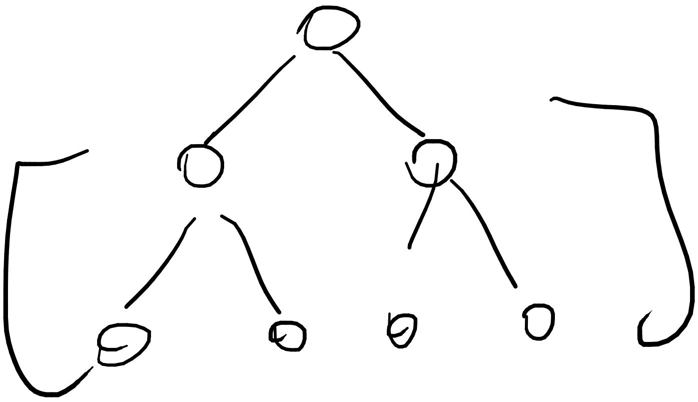
RICORSIVO

↓

MEMOIZZAZIONE

USE - CASE → FIBONACCI

```
REC-FIB(n)
if (n = 0) or (n = 1)
    return 1
return REC-FIB(n − 1) + REC-FIB(n − 2)
```



← TROVO DI NUOVO LE STESSE SOLUZIONI

DIVIDE AND CONQUER (D & C) → PROG. DINAMICA!

PROBLEMI DI OTTIMIZZAZIONE → MAX / → MIN

- STRINGHE

LCS → LONGEST COMMON SUBSTRING

STRINGA

[ABC] ...... [GH]

PREFISSO SUFFISSO

[ABC] DEF [GH] →

[ABC] XAF [GH]

A → $A\underbrace{BCD}E$

B $B\underbrace{BCD}$ → LCS(3)

MASTER - THOREM → RICORRENZE

PROG. DINAMICA → RICORRENZE SUI COSTI

SERVE A RIEMPIRE LA MATRICE ←

X / Y → STRINGHE

SOTTOSTRINGA COMUNE PIÙ LUNGA

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

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

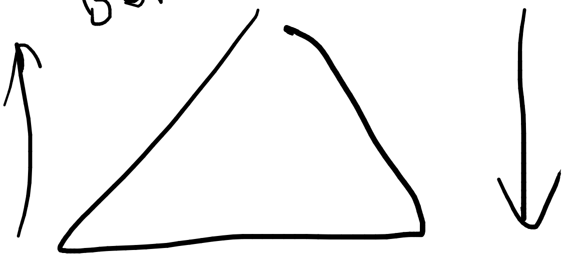
RICORR. SUI COSTI

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

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

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

BOTTOM-UP



TOP-DOWN

PROG. DINAMICA CON MATRICE

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

INIT →

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

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

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

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

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

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

LCS(X, Y)

```
m = X.[length]
n = Y.length
for i = 0 to m                      ] INIT.
    L[i,0] = 0
for j = 0 to n
    L[0,j] = 0
for i = 1 to m                      ] → n^2
    for j = 1 to n
        if x_i = y_j
            L[i,j] = L[i-1, j-1] + 1
            B[i,j] = '↖'            ] MATCH
        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)
```

B = B

SE HAI AVUTO UN MATCH

UNA PARTE AL MATTINO

NON C'È STATO ORA, MA C'È STATO PRIMA

		A	B	C	D	E	F
	0	0	0	0	0	0	
B	0	0	0	0	0	0	0
B	0	1	1	1	1	1	1
C	0	1	2	2	2	2	2
D	0						
E	0						

CALCOLO

F

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

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

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

$\leftarrow$ = VICINO MATCH

$\uparrow$ = NO MATCH ORA, MA PRIMA SI

NO, ALTRO MATCH

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

Pseudocodice Memoizzato

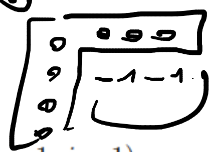
```
INIT-LCS(X, Y)
    m = X.length
    n = Y.length
    if (m = 0) or (n = 0)
        return 0
    for i = 0 to m
        L[i, 0] = 0
    for j = 0 to n
        L[0, j] = 0
    for i = 1 to m
        for i = 1 to n
            L[i, j] = −1
    return R-LCS(X, Y, m, n)
```

INIT.

−1

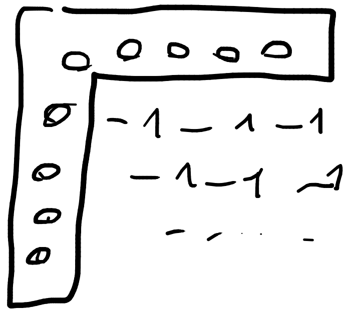
```
R-LCS(X, Y, i, j)
    if L[i, j] = −1
        if x_i = y_j
            L[i, j] = R-LCS(X, Y, i − 1, j − 1)
        else if R-LCS(X, Y, i − 1, j) ≥ R-LCS(X, Y, i, j − 1)
            L[i, j] = L[i − 1, j]
        else
            L[i, j] = L[i, j − 1]
    return L[i, j]
```

RICORDO



← SOL.

INIT →



COPPIA

↖ (IF $x_i = y_j$)

↑ (IF $x_{i-1} \geq y_j$)

← (IF $x_i \geq y_{j-1}$)

LCS → LIS

↑

LONGEST INCREASING SUBSEQUENCE

SOTTO ALGORITMO

⇓

FOCUS ON THE MEANING

```
Algorithm MatrixLIS(arr)
    // Input: Array arr of length n
    // Output: Length of LIS and one possible LIS sequence

    if arr is empty
        return 0, empty_sequence

    n = length of arr

    // Initialize DP matrix with zeros
    // dp[i][j] represents LIS length considering elements from index i to j
    dp[n][n] = matrix of zeros

    // Initialize path matrix for sequence reconstruction
    // path[i][j] stores the index k where arr[k] was the last element before arr[j]
    path[n][n] = matrix of null values

    // Base case: every single element is an increasing subsequence of length 1
    for i = 0 to n-1
        dp[i][i] = 1

    // Fill the matrix diagonally
    for diff = 1 to n-1
        for i = 0 to n-diff-1
            j = i + diff

            // First consider best result excluding current element
            dp[i][j] = max(dp[i+1][j], dp[i][j-1])

            // Try to include arr[j] by finding a valid previous element
            for k = i to j-1
                if arr[k] < arr[j] then
                    current = dp[i][k] + 1
                    if current > dp[i][j] then
                        dp[i][j] = current
                        path[i][j] = k

    // Sequence reconstruction procedure
    Procedure BuildSequence(i, j)
        if i ≥ j then
            if i equals j then
                return [arr[i]]
            else
                return empty_sequence

        if path[i][j] is null then
            return BuildSequence(i+1, j)

        k = path[i][j]
        return BuildSequence(i, k) concatenated with [arr[j]]

    return dp[0][n-1], BuildSequence(0, n-1)

// Time Complexity: O(n³)
// Space Complexity: O(n²)

Example:
Input array: [10, 22, 9, 33, 21, 50]

DP Matrix structure:
dp[i][j] represents: Length of LIS from index i to j
   j→ 0  1  2  3  4  5
i↓
0   1  2  2  3  3  4
1   -  1  1  2  2  3
2   -  -  1  2  2  3
3   -  -  -  1  1  2
4   -  -  -  -  1  2
5   -  -  -  -  -  1

Path Matrix will store indices k where arr[k] < arr[j] and leads to optimal solution
This helps in reconstructing the actual sequence.
```

ALTRO PROBLEMA → COMPLETAMENTO A PALINDROMO

= SOTTOSTRINGA PALINDROMA

7.7.3 Ricorrenza sulle Lunghezze

Definisco

$l(i,j) = |CP(X_{i\ldots j})|$

A = A

$$l(i,j) = \begin{cases} 1 & \text{se } i = j \rightarrow 1 \\ 2 & \text{se } j = i+1 \text{ e } x_i = x_j \\ 3 & \text{se } j = i+1 \text{ e } x_i \neq x_j \\ 2 + l(i+1, j-1) & \text{se } j > i+1 \text{ e } x_i = x_j \\ 2 + \min\{l(i+1,j), l(i,j-1)\} & \text{se } j > i+1 \text{ e } x_i \neq x_j \end{cases}$$

AB = BA

CALCOLO

← SALVIAMO SOLUZIONE

	c	i	a	o
c	1	3	5	6
i		1	3	5
a			1	3
o				1

Legenda:

- Diagonale (1): caso base, singolo carattere

- Prima diagonale (3): coppie di caratteri

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

$$L = \left[\begin{array}{ccc} (i,j{-}1) & \leftarrow & (i,j) \\ & \swarrow & \downarrow \\ (i{+}1,j{-}1) & & (i{+}1,j) \end{array} \right]$$

Riempio la tabella per diagonali, dall'alto verso il basso e da sinistra verso destra.

SPC(X)

```
n = X.length
for i = 1 to n − 1
    L[i, j] = 1
    if x_i = x_{i+1}
        L[i, i + 1] = 2
    else
        L[i, i + 1] = 3
L[n, n] = 1
for l = 3 to n // scansione diagonale con l indice della diagonale
    for i = 1 to n − l + 1
        j = i + l − 1
        if x_i = x_j
            L[i, j] = 2 + L[i + 1, j − 1]
        else
            L[i, j] = 2 + min{L[i + 1, j], L[i, j − 1]}
return L[1, n]
```

INIZIALIZZO

CALCOLO SOLUZIONE

ESEMPIO!

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 $1 \le i \le j \le n$:

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

(caso base)

(vedo (1,n) e so che la scansione è per diagonale)

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

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

RACCOLTA ESERCIZI → PROG. DINAMICA DI BALDAN (NO!)

DA VECCHI ESAMI ←

O

TUTTI GLI ESERCIZI (PROG. DINAMICA)

SEMPRE GLI STESSI → 11 PUNTI FACILI

CICLO DI INIZ. CON 1 SOLO INDICE

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

(1) I = J

(2) J = I+1 ELEMENTI

METODO (PERMETTE RIC.) PER DIAG

~QUI RISPARMIO)

(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)
```

I = J

J = I+1

Se gli indici sono uno solo o due soli, allora, seguendo i casi base si ritorna 2 e 3.

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)

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.

HAI ESCLUSO UNA RIGA PER CASI BASE

N-2 / I+2

QUI CALCOLI

```
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]
```

(b)

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

2 => I = J

3 = I = J+1

CALCOLO -> RIC

INIT

IF N=2
IF N=3

FOR I
FOR J
M[I,J]=2
M[I,J+1]=3

FOR I
FOR J
M[I,J]=0

REC.

IF M[I,J]=0

M[I,J] = RIC. .....

RETURN M[I,J]

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:

Si capisce che si scansione in ordine descrescente 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 $m = \min\{c(i,j) : 0 \leq i \leq j \leq n-1\}$. → MINIMO TRA INDICI ORDINA

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.

(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.

→ FUNZIONE ITERATIVA

```
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)

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

La sommatoria interna è costituita da soli 1, in quanto richiede una moltiplicazione tra tre numeri interi, nello specifico (n-2), (n-1), n.

Quanti uno?
Beh, da j=i+2 a\ n ci sono esattamente n-(i+2) +1=n-i-1 termini (sostituisco j in i, perché la serie è basata su i e il +1 si ha per il fatto che i=1).
Poi sostituire con k accorgendoti che sono esattamente gli stessi termini della sommatoria, se provi a svilupparli, e l'ultima sommatoria la puoi riscrivere in quel modo, ricordandoti che la somma di 1...n in generale è n(n+1)/2.
Non avendo il termine 2 per linearità della sommatoria, non viene moltiplicato con il "fratto 2" di (n-2)(n-1) e quindi il risultato è proprio (n-2)(n-1)/2

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

① →

② ↗

FOR[I = 1 TO N − 1] → $\begin{cases} 1 \leq 0 \quad \cancel{X} \quad 1 \leq 0 \\ 1 \leq n-1 \end{cases}$

J = N − 1 → CICLO È SU I

MINIMO SU I ✓

$C[1, n-1] = a_i$ $\quad$ $m = \min(C[i, n-1]$

FOR [J=0 TO N−1] → {0 ≤ J ≤ n−1}

C [0, J] = B_J

m = min (C[0, J])

(3) ↗ [n − 1 × n − 1] → INIZIALIZZAZIONE SU MATRICE N − 1 × N − 1

PROD CARTESIANO = RIGHE · COLONNE

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

CALCOLO SOMMATORIE

```
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
```

- "j" va da (n-1) a 0 compresi, quindi dato che abbiamo inizializzato, parte da (n-2)

(b)

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)/

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

APPLICO GAUSS

PASSO AD 1 INDICE

$$\sum_{i} k = \frac{n(n-1)}{2} \rightarrow \text{GAUSS}$$