[04/11/2024 ⇒ Algoritmi]

[OVERVIEW]

→ Intro a prog. dinamica + alberi esercizi .....

PROGR. DINAMICA → ~~OOP~~

DIVIDE & CONQUER

SALVO LE CHIAMATE RICORSIVE

[PROGR. DINAMICA] → MEMOIZZAZIONE

→ GREEDY

TOP-DOWN ↓

↑ BOTTOM UP

LCS → mini esercizio

No risolvo → trovi subito la soluzione

LCS → SOTTOSTRINGA COMUNE PIÙ LUNGA

ARCOBALENO

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

USO STRUTT. DATI ⟷ MATRICI

```
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)
```

INIZIALIZZAZIONE

CALCOLO SOLUZIONE

Complessità

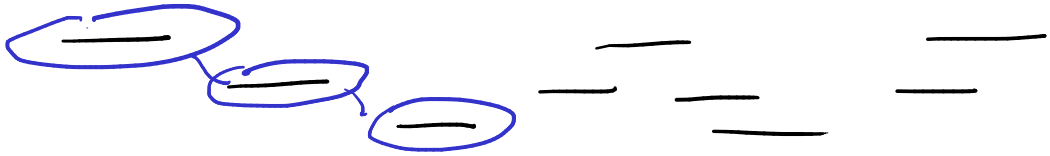
$T(m, n) = \Theta(m \cdot n)$

GREEDY

SALVO LA MIGLIORE SOLUZIONE

→ → →

ACTIVITY SELECTION



INPUT → SUBITO PRIMA ATTIVITÀ

Versione iterativa:

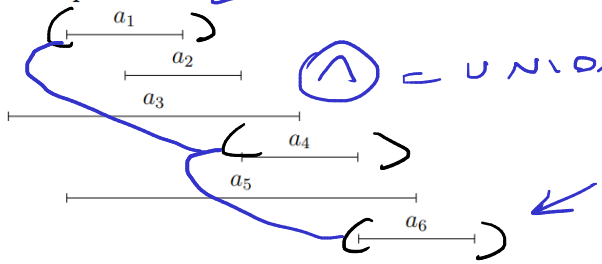
GREEDY-SEL(S, f)
1 $n = S.length$
2 $A = \{a_1\}$
3 $last = 1$ // indice dell'ultima attività selezionata
4 **for** $m = 2$ **to** n
5 **if** $s_m \geq f_{last}$
6 $A = A \cup \{a_m\}$
7 $last = m$
8 **return** A

UNIONE SOLO ALTRE → OTTIMIZZATO USANDO I MIGLI INPUT

Esempio

SORTA OTTIMA = INIZIO / FINE

∧ = UNIONE (NO SOVRAPPOSIZIONE)



$A = a_1$ $last = 1$ | $A = a_1, a_4$ $last = 4$ | $A = a_1, a_4, a_6$ $last = 6$

GREEDY → Efficiente

CUT & PASTE → Meglio la sel. ottima

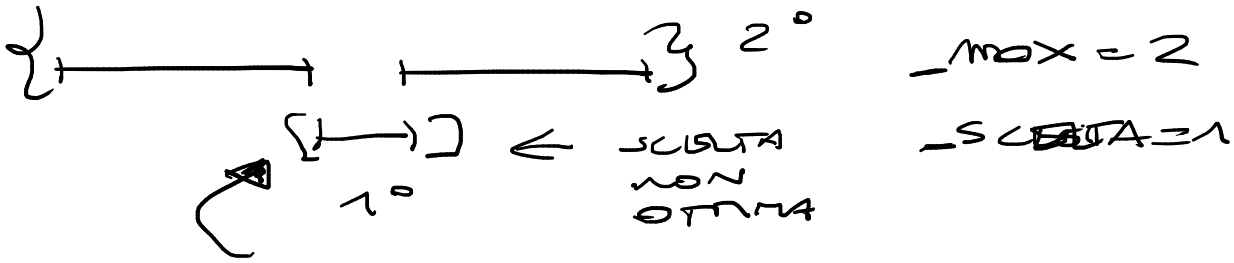
→ SCELTA GREEDY

$A = \{a_1 \dots a_n\}$ a_i

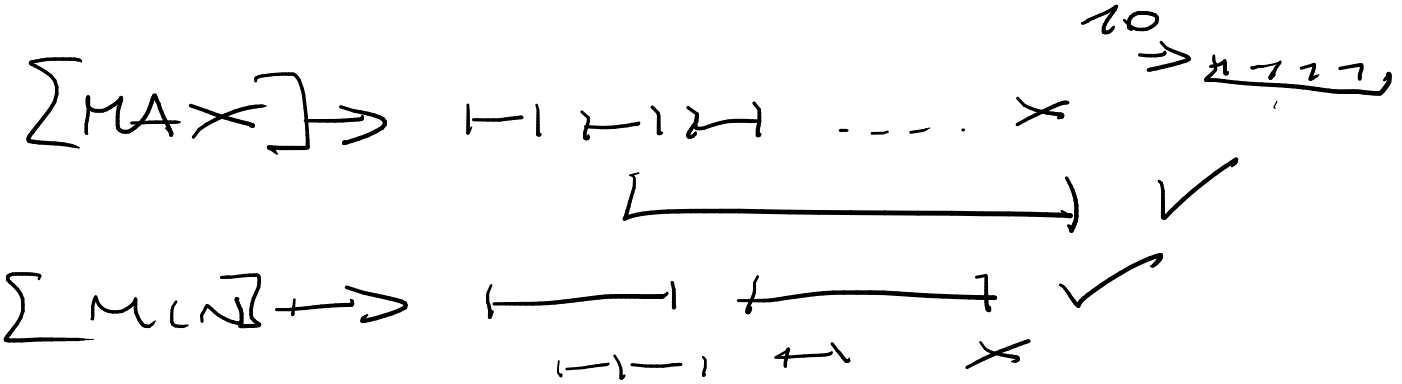
→ s_i / f_i

CINEMA (FILM)

- ATTIVITÀ CHE DURA MENO



- MAX = 2
- SCELTA = 1



- GREEDY → ARRAY / RIC.
- PROG. DINAMICA → FOR (i TO N) FOR (j TO N)

(OTTIMA) → MAX MIN

↑ MATRICI

Esercizio 2 Scrivere una procedura di tipo divide et impera *over* che dato un array di interi distinti $A[1..n]$, ordinato in modo crescente, e un intero x restituisce l'indice del più piccolo elemento in A strettamente maggiore di x. Se nessun elemento di A soddisfa la condizione, si restituisca $n+1$. Valutare la complessità dell'algoritmo.

OVER(A) ← {FIRMA ...}

A[1 2 3 4 . . .]

x = 3 → 4

OVER(A, x, p, r)

[ ] [ ] [ ]
p q r

[IF (p > r)
  return [r + 1]]

⇒ CASO BASE (NON ORDINATO)

ELSE

q = FLOOR((p + r)/2)

(APPROX. PER DIFETTO)

- FLOOR ⌊x⌋ ← 0
- CEILING → CEIL ⌈x⌉ → ∞

p — q — r

1 2 3 4 5 6

x = 3 4 → OUTPUT

IF A[q] < x
  RETURN OVER(A, q+1, r, x)

ELSE
  RETURN OVER(A, p, q-1, x)

CORRETT. 0 → n

$T(n) = \begin{cases} \log(n) & A[q] > x \rightarrow A[q-1, p] \\ T(\frac{n}{2}) * c & A[q] <= x \rightarrow A[q+1, r] \end{cases}$

Domanda 14 Dare una soluzione asintotica per la ricorrenza $T(n) = 3T(n/2) + n(n+1)$.

$$\lim_{n\to\infty} \frac{f(n)}{n^{\log_b a}} = \frac{n(n+1)}{n^{\log_2 3}} = \infty$$

Master theorem.

Metodo alternativo a quello di sostituzione e non infallibile. Risolve un sottoinsieme di equazioni di ricorrenza che hanno forma:

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

Il teorema ci dice che il risultato finale sarà $f(n)$ o $n^{\log_b a}$ e per stabilire il vincitore si fa il limite del rapporto che tende ad infinito. Si distinguono tre casi:

Limite	Risultato	Soluzione
$\lim_{n\to\infty} \frac{f(n)}{n^{\log_b a}} = k$	$k = l$	$T(n) = \theta\left(n^{\log_b a} \log(n)\right)$
	$k = 0$	$T(n) = \theta\left(n^{\log_b a}\right)$ se $\exists k > 0 \mid \lim_{n\to\infty} \frac{f(n)}{n^{\log_b a - \varepsilon}} = k$
	$k = \infty$	$T(n) = \theta\left(f(n)\right)$ se $\exists k > 0 \mid \lim_{n\to\infty} \frac{f(n)}{n^{\log_b a - \varepsilon}} = k$ se $\exists k, 0 < k < 1 \mid a f\left(\frac{n}{b}\right) \le k f(n)$

CASO 3 → CONDIZIONE DI REGOLARITÀ

$T(n) = 3T(n/2) + n(n+1)$.

k ? = SOLUZIONE

$$3\left(\frac{n(n+1)}{2}\right) \le k(n(n+1))$$

$$3\frac{n}{2}\left(\frac{n}{2}+1\right) \le k(n(n+1))$$

$$\frac{3n^2}{4} + \frac{3n}{2} \le kn^2 + k \quad \rightarrow k > \frac{3}{4}$$

$\left(\frac{13}{16}\right)$ → (?)

Domanda A (8 punti) Si consideri la seguente funzione ricorsiva con argomento un array A di interi e due indici $1 \leq p \leq r \leq A.length$:

```
Minimum(A,p,r)
   if p=r
      return A[p]
   else
      q = (p+r)/2
      return min(Minimum(A,p,q), Minimum(A,q+1,r))
```

n/2 n/2

Dimostrare induttivamente che la funzione calcola il minimo del sottoarray $A[p..r]$. Inoltre, determinare la ricorrenza che esprime la complessità della funzione e mostrare che la soluzione è $\Theta(n)$, dove n indica la lunghezza del sottoarray. Motivare le risposte.

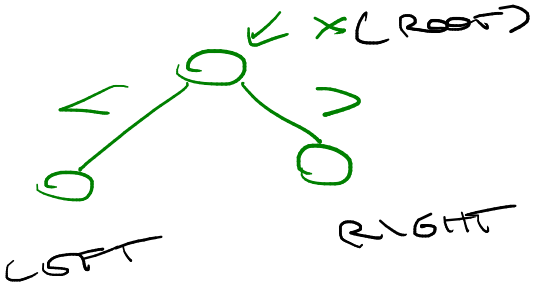
$T(n) = 2T(\frac{n}{2}) + c$ ① RICORRENZA

→ MASTER THEOREM ② RISOLVI

① $= n = 1$ ② TROVI SOLUZIONE

Domanda B (6 punti) Realizzare una funzione `SearchUnique(T,k)` che dato un albero binario di ricerca `T` (che si assume non vuoto), verifica se la chiave `k` è presente in un unico nodo, e, in caso affermativo restituisce il nodo, altrimenti (ovvero se la chiave non è presente oppure è presente in più nodi) restituisce `nil`. Si possono usare le funzioni standard degli alberi binari di ricerca. Valutarne la complessità.

STANDARD → MAX / MIN / INSERT / SEARCH / DELETE

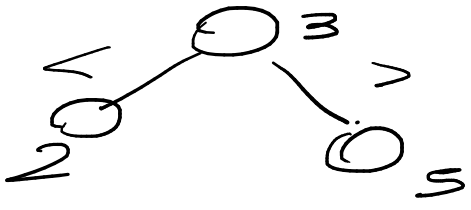


SEARCH

SEARCH(x, k)
1 **if** $(x = \text{NIL})$ **or** $(x.key = k)$
2 **return** x
3 **else if** $k < x.key$
4 **return** SEARCH$(x.left, k)$
5 **else**
6 **return** SEARCH$(x.right, k)$

SEARCH_UNIQUE (T, k)

x = SEARCH (T.ROOT, k)



IF (x == NIL) ∨

SEARCH (x.LEFT, k) <> NIL ∨

SEARCH (x.RIGHT, k) <> NIL

RETURN (x)

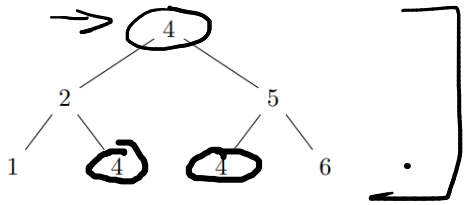
→ [min ≤ k ≤ max]

↑ CORRETT.

ELSE

RETURN (NIL)

Per concludere, è da osservare che l'assunzione che le chiavi duplicate siano solo nel sottoalbero destro (o in quello sinistro) o che siano adiacenti non è legittima. Ad esempio, il seguente è un BST valido:



O(h)

RANGE

y = MAX(x.LEFT)

z = MIN(x.RIGHT)

IF (y != NIL) AND (y.KEY == k)

OR

IF (z != NIL) AND (z.KEY == k)

RETURN NIL

ELSE

RETURN x

ELSE

RETURN NIL