26/03/2024 | LEZ. 23

PROGR. DINAMICA : 2° PARTE

problemi che vederemo oggi:
- weighted interval scheduling (con memoization)
-longest increasing subsequence
-esercizio: house coloring problem

1° PROBLEMA : WEIGHTED INTERVAL SCHEDULING.

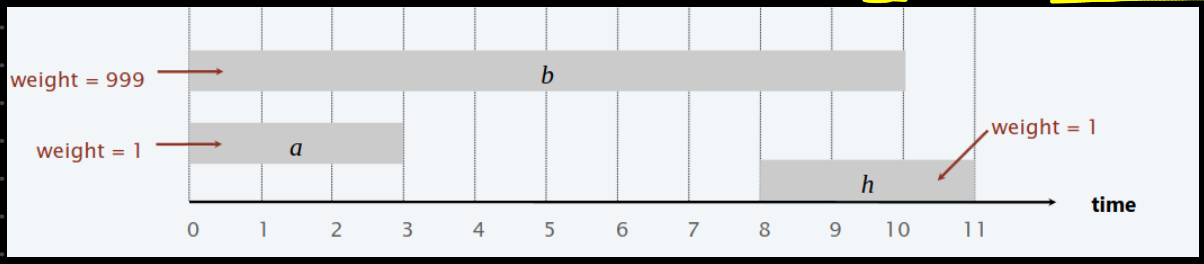
= A INTERVAL SCHEDULING MA OGNI INTERVALLO i HA PESO w_i.

OUTPUT (DA MAX) : SOMMA PESI OF WORK SCHEDULATI (COMPATIBILI)

con interval scheduling si usava l'algo greedy, ma qui funziona?

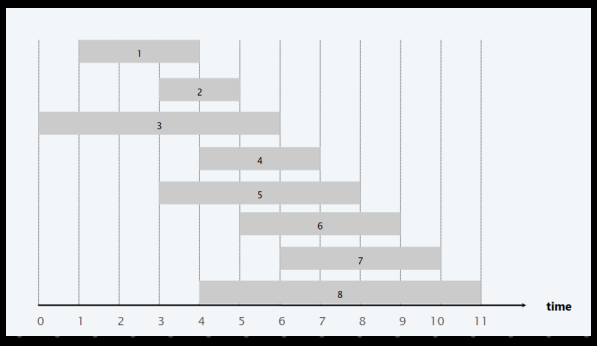
SPOILER : NO

a, h PRESI, MA MA b HA PESO MAX



COSA FARE ?

- SORT JOBS BY FINISH-TIME (↗) → COME I. SCHEDULING



ORA CHE PROPRIETÀ HA?

- CONSIDERIAMO LAST JOB (8) — S = SOL. OTTIMALE
  - IF $8 \notin S$ → SOL.OTT = S^* → SOL. OTTIMA X SET JOB $\setminus\{8\}$
  - IF $8 \in S$ → SOL. OTT. = $8 + S^*$ → SOL. OTT. DI JOB COMPATIBILI U 8

DA QUI...

DEF $P(J)$ = MAX. INDICE $i<J$ t.c JOB_i COMPAT. JOB_J

ALGO

DEF $OPT(J)$ = MAX V.I.2 PER I PRIMI J INTERVALLI

↓

GOAL = $OPT(n)$.

L'ALGO PUÒ FARE 2 COSE. → NEL JOB GENERICO J

1. OPT[J] NON SELECT JOB J
   - ALLORA CALCOLO OPT[J-1] → SOL. OTT. SUI 1° J-1 JOB
2. OPT[J] SCEGLIE J:
   - CALCOLO OPT[P(J)] → SOL. OTT. PER I PRIMI P(J) JOB COMPATIBILI
   - SOMMO U W_J.

(↳ VERO TO BE OPTIMAL)

PSEUDO CODICE → TOP DOWN → BOTTOM-UP → RISOLVO I SOTTOPROBLEMI DAI + PICCOLI.

BOTTOM-UP($n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n$)

Sort jobs by finish time and renumber so that $f_1 \le f_2 \le \ldots \le f_n$.

Compute $p[1], p[2], \ldots, p[n]$ via binary search.

$M[0] \leftarrow 0$.

FOR $j = 1$ TO n (previously computed values)

  $M[j] \leftarrow \max\{ M[j-1],\ w_j + M[p[j]] \}$.

RETURN $M[n]$.

→ EQ. BELLMAN

$$OPT(J) = \begin{cases} 0 & J = 0 \\ \max \{OPT[J-1],\ W_J + OPT[P(J)]\} \end{cases}$$

COSTO:
- SORT = $O(n \log n)$
- P[i] × n VOLTE = → $n \cdot O(\log n)$.
- FOR CYCLE → $O(n)$

SOL = $O(n \cdot \log n)$.

POSSIAMO USARE LA STRATEGY TOP-DOWN

↳ RISOLVO PROB. DA PICCOLO → GRANDE.

PSEUDO CODICE

BRUTE-FORCE $(n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n)$

Sort jobs by finish time and renumber so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

Compute $p[1], p[2], \ldots, p[n]$ via binary search.

RETURN COMPUTE-OPT(n).

COMPUTE-OPT(j)

IF ($j = 0$)

  RETURN 0.

ELSE

  RETURN max {COMPUTE-OPT($j-1$), w_j + COMPUTE-OPT($p[j]$) }.

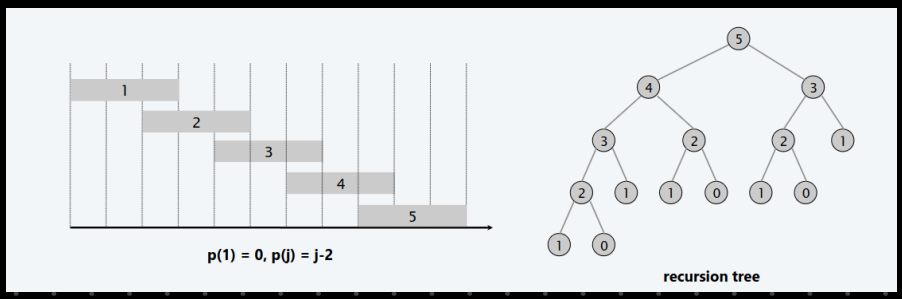
⇒ + PULITO MA COSTOSO

- $\Theta(1) \rightarrow n = 1$
- $T(n) = 2T(n-1) + \Theta(1)$.
  ↳ $n > 1$

⇓

COSTO $= O(2^n)$

PERCHÉ



COME × FIBONACCI

⇓

NUMERO CHIAMATE RIC. RISOLVO PROBLEMI MAGARI GIÀ RISOLTI

possiamo migliorarlo con la memoization → SUB PROBL. RISOLTI MEMORIZZO IN M[J]

⇓

SE M[J] SPORCATO NON LO RISOLVO UN'ALTRA VOLTA

NUOVO CODICE

TOP-DOWN($n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n$)

Sort jobs by finish time and renumber so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

Compute $p[1], p[2], \ldots, p[n]$ via binary search.

$M[0] \leftarrow 0$. ← global array

RETURN M-COMPUTE-OPT(n).

M-COMPUTE-OPT(j)

IF ($M[j]$ is uninitialized) → NON INIZIALIZZATO.

  $M[j] \leftarrow$ max { M-COMPUTE-OPT($j-1$), w_j + M-COMPUTE-OPT($p[j]$) }.

RETURN $M[j]$.

COSTO:

- SORT, SEARCH = A PRIMA $O(n \lg n)$.
- CHIAMATE RICORSIVE:
  - O M[J] GIÀ FATTO $O(1)$
  - O FA 2 CHIAMATE RICORSIVE

QUINDI AL MASSIMO 2n RECUR. CALL.

COSTO $O(n \log n)$

TUTTI E 2 GLI APPROCCI SONO VALIDI E HANNO PRO E CONTRO

RICORSIONE ↑ ITERATIVO ↑

Memoization (top-down)	VS	Table-based (bottom-up)
• Top-Down approach (more intuitive)		• Harder to grasp
• Easier to index subproblems by other objects (e.g., sets).		• Need to index subproblems with integers
• Only computes necessary subproblems		• Always computes all subproblems
• Function calls overhead		• No recursion. More cache efficient.
• Time complexity is harder to analyze		• Time complexity is easy to analyze
		• Short and clean code

2° PROBLEMA: LONGEST INCREASING SUBSEQUENCE

DATO UN ARRAY S.

GOAL:

SOTTOSEQUENZA S' DOVE

- $\forall i, k \quad 0 < i, k < m \rightarrow i < k \Rightarrow S[i] < S[k]$
- S' + LUNGA POSSIBILE

P.D

- SUBPROBL. → $LIS[i]$ = LIS OF LENGHT i
- CASO BASE → $LIS[1] = 1$
- GOAL = $LIS[m]$

• FORMULA -> ? COME FARE?

NELLA REC. DI SOTTOPRO. POSSIAMO IMPORRE DEI PARAMETRI

• NEW SUBPROB. → OPT[i] = LENG. OF LIS CHE FINISCE CON S[i].

ESEMPIO

NEW P.D.

• SUBPR → OPT[i] = LIS OF S CHE FINISCE CON S[i]

• GOAL → $\max_{i=1,\dots,n} \{OPT[i]\}$

• FORMULA

$$OPT[i] = 1 + \max\left\{0, \max_{\substack{j=1,\dots,i-1 \\ t.c.. \\ S[j]<S[i]}} OPT[j]\right\}$$

CODE

LIS(S[1:n])

OPT[1]=1

FOR $i = 2$ TO n

$$\text{OPT[i]} = 1 + \max\left\{0, \max_{\substack{j=1,2,\ldots,i-1 \\ \text{tc } S[j]<S[i]}} \text{OPT[j]}\right\}$$

RETURN max_i OPT[i].

↝ MASSIMO TRA 0

OPPURE

MAX DEGLI OPT PRIMA VALIDI

COSTO

- OGNI OP. SU OPT → O(n) ↝ SCORRE LISTA PRIMA
- TOT ↝ $O(n^2)$

EXE

HOUSE COLORING PROBLEM

Goal. Paint a row of n houses red, green, or blue so that

- No two adjacent houses have the same color.
- Minimize total cost, where *cost*(*i*, *color*) is cost to paint *i* given color.

	A	B	C	D	E	F
	7	6	7	8	9	20
	3	8	9	22	12	8
	16	10	4	2	5	7

cost to paint house i the given color

↝ COLORE N CASE

RESTRIZIONI:

- CASE ADIACENTI ↓ NO COLORE UGUALI
- MIN COST

SOLUTION

DEF PROBL:

- CONSID. 3 PROBLEMI (3 TAB.)
  - $R[i]$ = MIN. COST x DIP. CASE $1,..,i$ U i ROSSO
  - $V[i]$ = " " " " " " VERDE
  - $B[i]$ = " " " " " " BLU
- GOAL $\rightsquigarrow \min\{R[n], V[n], B[n]\}$
- FORMULA:

  $\forall i = 1,...,n$

  - $R[i] = COST(i, ROSSO) + \min\{V[i-1], B[i-1]\}$
  - $V[i] = COST(i, VERDE) + \min\{R[i-1], B[i-1]\}$
  - $B[i] = COST(i, BLU) + \min\{R[i-1], V[i-1]\}$

EZ TO CODE

↓

COSTO = $O(n)$