

LEZIONE 6

6. ANALISI DELL'ALTEZZA DI UN ALBERO ROSSO-NERO

Un albero rosso-nero è un albero binario con un extra bit di informazione per ogni nodo: il suo colore, ROSSO o NERO.

Proprietà rosso-nero:

1. ogni nodo è rosso oppure nero;
2. la radice è nera;
3. tutte le foglie (NIL) sono nere;
4. se un nodo è rosso, allora entrambi i suoi figli sono neri;
5. per ogni nodo, tutti i cammini semplici dal nodo alle foglie discendenti contengono lo stesso numero di nodi neri.

CONVENZIONE: Utilizzerò un singolo nodo per rappresentare NIL, T. null, che ha gli stessi attributi di un nodo nero eccetto dell'altezza.

DEF.: ALTEZZA NERA di un nodo x , $bh(x)$: # modi neri su un cammino
semplice da x a una
foglia, escluso x stesso.

(ben definita, per s.)

DEF.: ALTEZZA NERA di un albero rosso-nero: altezza nera delle due radici.

Lemma: Un albero rosso-nero con n nodi interi ha altezza al
più $\overbrace{2 \log(n+1)}$.

$$\overbrace{O(\log(n))}$$

Proof: \otimes Dimostriamo per induzione che il sottoalbero radicato ad un
nodo x contiene almeno $2^{bh(x)} - 1$ nodi interi, per
l'induzione sull'altezza di x .

base dell'induzione: Se $h(x) = 0 \Rightarrow x$ è una foglia, i.e., $x = T.nil$,
e il sottoalbero radicato a x contiene
 $2^{bh(x)} - 1 = 2^0 - 1 = 1 - 1 = 0$. ✓

ipotesi induttiva: Supponiamo che \otimes sia vera per tutti i nodi

di altezza h , tale che $0 \leq h < h(x)$.

passo riuttivo: proviamo che \otimes è vero per $bh=bh(x)$.

Se $bh(x) > 0$, x ha due figli, ciascuno dei quali può essere una foglia.

Se un figlio è nero, allora da un contributo 1 all'altezza nera di x non alle proprie.

Se un figlio è nero, allora non contribuisce alla altezza nera di x e neanche alle proprie.

Allora, ogni figlio di x ha altezza nera:

- $bh(x) - 1$, se è nero . } $bh(x.\text{left/right}) \geq bh(x) - 1$
- $bh(x)$, se è nero

L'altezza di un figlio di x è minore di $bh(x)$, perciò possiamo applicare l'ipotesi riuttiva,

e quindi concludere che ogni figlio di x ha almeno $2^{\frac{bh(x.\text{left/right})}{V_1}} - 1$ nodi riferiti.

$$2^{\frac{bh(x)-1}{V_1}} - 1$$

Allora i nodi criterici al sottoalbero

radicato ad x sono almeno

$$(\geq) 2(2^{bh(x)} - 1) + 1 = 2^{bh(x)} - 2 + 1 = 2^{bh(x)} - 1.$$

Abbiamo dimostrato \otimes .

Per completezza la dimostrazione, sia h^* l'altezza dell'albero.

Per la proprietà 4, almeno metà dei nodi su un cammino
semplice dalla radice alle foglie, escludendo la radice, deve
essere fatto di nodi veri. Perché, l'altezza vera delle

$$\text{radice } bh(\text{root}) \geq \frac{h^*}{2} \Rightarrow$$

$$\otimes \quad bh(\text{root}) \geq \frac{h^*}{2}$$

$$\underbrace{n}_{\# \text{ nodi}} \geq 2^{\frac{h^*}{2}} - 1 \geq 2^{h^*} - 1 \Rightarrow$$

$\# \text{ nodi}$
criterici
all'albero

$$\Rightarrow 2^{\frac{h^*}{2}} \leq n+1 \Rightarrow \frac{h^*}{2} \leq \log(n+1) \Rightarrow h^* \leq 2 \log(n+1).$$

□

Conseguenze: Le operazioni su inservizi dinamici (SEARCH, INSERT, DELETE, MINIMUM, MAXIMUM, SUCCESSOR, PREDECESSOR) su un albero sono verso l'alto $O(\log n)$ -time.

7. ESEMPLI E ANALISI DELLA PROGRAMMAZIONE DINAMICA

7.1 PROPRIETÀ DI SOTTOSTRUTTURA OTTIMA

Due proprietà necessarie per l'applicabilità delle programmazioni dinamiche:

- sottostruzione ottima (optimal substructure);
- sottoproblemi sovrapposti (overlapping subproblems).

Un problema esibisce le proprietà di sottostruzione ottima se una sua soluzione ottima contiene al suo interno sottosoluzioni ottime ai sottoproblemi.

Generalmente, per dimostrare le proprietà di sottostruzione ottima si suppone che le restrizioni delle soluzioni ottime

ai sotto problemi non siamo soltanto e se ne deduce una
combinazione.

7.1.1 ESEMPIO: ROD - CUTTING

Input: una barra di lunghezza n e valori tabellati di pezzi
 p_i , per $i \in \{1, \dots, n\}$.

Goal: Determinare il guadagno massimo che è possibile
ottenere tagliando la barra e vendendone i pezzi.

Potremmo esprimere il problema come quello di
determinare

$$\begin{aligned} r_n &= \max_{\substack{i_1, \dots, i_k: \\ i_1 + \dots + i_k = n}} p_{i_1} + \dots + p_{i_k} = \max \left\{ p_{i_1}, r_{i_1} + r_{n-i_1}, r_{i_2} + r_{n-i_2}, \dots \right\} = \\ &= \max \left\{ p_i + r_{n-i} \mid 1 \leq i \leq n \right\}, \quad r_0 = 0. \end{aligned}$$

In questa formulazione:

- se una soluzione contiene le soluzioni di un sotto problema;
- suggerisce un algoritmo ricorsivo che però ha complessità esponenziale 
- Suggerisce una soluzione basata sulla programmazione dinamica che riduce il tempo polinomiale.  Risolve i sottoproblemi di dimensione $j = 0, 1, \dots, n$ in questo ordine (bottom-up).

ROD-CUTTING soddisfa la proprietà:  una soluzione ottima di taglio di una barra di lunghezza n è una unica soluzione di dimensione ($n-i$). 

$$\begin{aligned}
 r_n &= r_{n-i} + p_i, \quad \text{se } r_{n-i} \text{ non è ottima} \\
 &\quad \text{vuo dire che esiste una} \\
 &\quad \text{sol di taglio: } r'_{n-i} > r_{n-i} \\
 &\Rightarrow r'_n = r'_{n-i} + p_i > r_{n-i} + p_i \\
 &\Rightarrow r'_n > r_n \quad \text{perché } r_n \text{ è max.}
 \end{aligned}$$

7.1.2 ESEMPIO: MATRIX-CHAIN MULTIPLICATION (MATRIX MULTIPLICATION)

INPUT: una sequenza $\langle A_1, A_2, \dots, A_n \rangle$ di matrici da moltiplicare (non necessariamente quadrate).

GOAL: "Parentenizzare" il prodotto $A_1 \cdot A_2 \cdots A_n$ per minimizzare il numero di moltiplicazioni scalari.

Il problema soddisfa le proprietà di sottostruttura, ovvero: Sia $1 \leq i < j \leq n$. Supponiamo che per parentenizzare $A_i \cdot A_{i+1} \cdots A_j$ si spezzino tra A_k e A_{k+1}

$$(A_i \cdot A_{i+1} \cdots A_k) (A_{k+1} \cdots A_j)$$

costo(P) = # mult scalari di P

$$P_{\text{tutto}} (A_i \cdots A_k A_{k+1} \cdots A_j) = P_1 (A_i \cdots A_k) P_2^c (A_{k+1} \cdots A_j)$$

$$\text{costo}(P_{\text{tutto}}) = \text{costo}(P_1) + \text{costo}(P_2) + h \left(\begin{array}{l} h: \text{costo per mult} \\ (A_i \cdots A_k)(A_{k+1} \cdots A_j) \end{array} \right)$$

Supponiamo che P_1 sia ottenuto per $A_i \cdots A_k$, allora

$\exists P_1'$ di $A_i \dots A_k$ tale che $\text{costo}(P_1') < \text{costo}(P_1)$,
ma se così fosse potrei definire

$$P^*(A_i \dots A_k A_{k+1} \dots A_j) = P_1'(A_i \dots A_k) P_2(A_{k+1} \dots A_j)$$

$$\begin{aligned} \text{costo}(P^*) &= \text{costo}(P_1') + \text{costo}(P_2) + h < \text{costo}(P_1) + \text{costo}(P_2) + h \\ &= \text{costo}(P_{\text{min}}) \end{aligned}$$

$$\Rightarrow \text{costo}(P^*) < \text{costo}(P_{\text{min}})$$

ASSURDO
perché $P_{\text{min}} \leq$ di
qualunque
altra parentesi-
zione
di $(A_i \dots A_j)$.

Analogamente per P_2 .

7.2 ANALISI DELLA PROGRAMMAZIONE DINAMICA

La sostruzione offre varie su base al problema considerato in due modi:

- # di sottoproblemi usati (e.g. su ROD-CUTTING: 1, su MATRIX-MULT: 2);
- # di scelte per i sottoproblemi da usare su cui

soluzione ottima.

In formulante, il tempo computazionale di algoritmi di programmazione dinamica dipende del prodotto di questi due fattori.

E.g.: Per ROD-CUTTING, abbiamo $\Theta(n)$ sottoproblemi in totale, e al più n scelte da eseguire per ognuno di essi $\Rightarrow \Theta(n^2)$

$$\Rightarrow \text{ROD-CUTTING} \in \Theta(n^2) - \text{TIME}.$$

In MATRIX MULTIPLICATION, otteniamo $\Theta(n^3)$ sottoproblemi in totale, e ognuno di essi ha al più $n-1$ scelte $\Rightarrow \Theta(n^3)$.
 \Rightarrow MATRIX MULTIPLICATION $\in \Theta(n^3) - \text{TIME}$.

7.4 QUANDO POSSIAMO APPLICARE LA PROGRAMMAZIONE DINAMICA?

Detto un grafo $G = (V, E)$, definiamo:

- cammino fra due vertici u e v : una sequenza di vertici

$$u \xrightarrow{P} v$$

$\langle v_0, v_1, \dots, v_n \rangle$ tali che $v_0 = u$, $v_n = v$

$$\{ (v_i, v_{i+1}) \in E, \forall i \in \{0, \dots, n-1\} \}.$$

- Lunghezza di un cammino p : # di archi contenuti
- Cammino semplice: un cammino $p = \langle v_0, v_1, \dots, v_m \rangle$ tale che $v_i \neq v_j \quad \forall i, j \in \{0, 1, \dots, m\}$. Non ci sono cicli.

Consideriamo:

(UNWEIGHTED) SHORTEST PATH

INPUT: un grafo $G = (V, E)$, due vertici u e v .

GOAL: Trovare un cammino da u a v di lunghezza minima.

(UNWEIGHTED) LONGEST SIMPLE PATH

INPUT: un grafo $G = (V, E)$, due vertici u e v .

GOAL: Trovare un cammino semplice p da u a v di lunghezza massima.

SHORTEST PATH ha le proprietà di sovrapposizione.

Sia $u \neq v$ (altrimenti il problema è banale). A causa di un p , da u a v deve contenere un vertice intermedio w (può accadere che $w=u$ o $w=v$). Allora possiamo decomporre il cammino in due p_{uw}

$$p_{uw} = p_1 + p_2$$

$$u \xrightarrow{p_1} w \xrightarrow{p_2} v$$

$$p_{uw} = p_1 + p_2$$

$$\ell(p_{uw}) = \ell(p_1) + \ell(p_2).$$

Supponiamo che $\exists p'_1 : u \xrightarrow{p'_1} w$ tale che $\ell(p'_1) < \ell(p_1)$, allora definisca $p^* = p'_1 + p_2$ che è un cammino da u a v .

$$u \xrightarrow{p'_1} w \xrightarrow{p_2} v = u \xrightarrow{p^*} v$$

$$l(p^*) = l(p_1') + l(p_2) < l(p_1) + l(p_2) = l(p^{\text{unif}})$$

Ho trovato un
concetto più
costo di quello
di p^{unif} .

Analogamente, si dimostre per p_2