Algoritmi e Strutture Dati - Prova d'esame
18/07/11

Esercizio 1

Questo esercizio è sottile. È simile, ma non identico, ad un esercizio degli appunti. Le somiglianze possono trarre in inganno.
Esplicitando le costanti, e ponendo $n = 2^k$ (oppure $k = \log n$) nella ricorrenza, otteniamo:

$$\begin{aligned} T(n) &= \sqrt{n}T(\sqrt{n}) + \sqrt{n} \Rightarrow \\ T(2^k) &= 2^{\frac{k}{2}}T(2^{\frac{k}{2}}) + 2^{\frac{k}{2}} \end{aligned}$$

Dividiamo tutto per 2^k, e otteniamo:

$$\frac{T(2^k)}{2^k} = \frac{T(2^{\frac{k}{2}})}{2^{\frac{k}{2}}} + \frac{1}{2^{\frac{k}{2}}}$$

Poniamo ora $S(k) = \frac{T(2^k)}{2^k}$; otteniamo quindi:

$$S(k) = S(k/2) + \frac{1}{2^{\frac{k}{2}}}$$

Notando che $\frac{1}{2^{\frac{k}{2}}}$ è decrescente per $k \geq 1$ e tende a 0 per k tendente all'infinito, è quindi possibile osservare che $\frac{1}{2^{\frac{k}{2}}}$ è limitata superiormente da un valore costante. Si potrebbe essere tentati di riformulare il tutto in questo modo:

$$S(k) = S(k/2) + \Theta(1)$$

e utilizzando il Master Theorem, affermare che:

$$S(k) = \Theta(\log k)$$

Ma in realtà si deve scrivere:

$$S(k) \leq S(k/2) + O(1)$$

e quindi $S(k) = O(\log k)$.
A questo punto, possiamo tornare indietro e ottenere:

$$\begin{aligned} \frac{T(2^k)}{2^k} &= O(\log k) \Rightarrow \\ T(2^k) &= O(\log k)2^k \Rightarrow \\ T(n) &= O(\log \log n)n \Rightarrow \\ T(n) &= O(n \log \log n) \end{aligned}$$

L'esercizio, fino a qui, richiede di applicare tecniche note. Il punto fondamentale è notare che quello che abbiamo calcolato è solo un limite superiore, per via della maggiorazione che abbiamo fatto ad un certo punto nella catena algebrica.
Per dimostrare il limite inferiore, possiamo fare un ragionamento di questo genere: dimostrare che $T(n) = \Omega(n \log \log n)$ sembra difficile dal punto di vista algebrico, si potrebbe dimostrare invece il più semplice $T(n) = \Omega(n)$. Proviamo.
Dobbiamo provare che $\exists c > 0, \exists m \geq 0 : T(n) \geq cn, \forall n \geq m$. Il passo base, al solito, è semplice:

$$T(1) = 1 \geq c \cdot 1 \Leftrightarrow c \leq 1$$

Nel passo induttivo, assumiamo che la proprietà $T(n') \geq cn'$ sia già stata provata per tutti gli n' tali che $1 \leq n' < n$, e proviamo che proprietà è vera per n.

$$\begin{aligned} T(n) &= \sqrt{n}T(\sqrt{n}) + \sqrt{n} \\ &\geq \sqrt{n} \cdot c \cdot \sqrt{n} + \sqrt{n} \\ &= cn + \sqrt{n} \geq cn \end{aligned}$$

L'ultima disequazione, banalmente vera, dimostra che $T(n) = \Omega(n)$.
Arrivare ad una risposta di questo tipo dà diritto ad un punteggio pieno dell'esercizio, vista la difficoltà. Ma si può fare un passo ulteriore, che dà certamente diritto ad un bonus.
Potremmo cercare di verificare se questo limite superiore può essere migliorato, provando per esempio che $T(n) = O(n)$, utilizzando il metodo di sostituzione. Dobbiamo quindi provare che $\exists c > 0, \exists m > 0 : T(n) \leq cn, \forall n \geq m$. Otteniamo quindi:

$$\begin{aligned} T(n) &= \sqrt{n}T(\sqrt{n}) + \sqrt{n} \\ &\leq \sqrt{n} \cdot c \cdot \sqrt{n} + \sqrt{n} \\ &= cn + \sqrt{n} \\ &\not\leq cn \end{aligned}$$

L'ultima parte della disequazione è chiaramente falsa, e quindi sembra che il limite superiore non sia dimostrabile. Il problema potrebbe derivare dal termine di ordine inferiore: $\sqrt{n}$. Passiamo quindi a dimostrare che $\exists c > 0, \exists b > 0, \exists m > 0 : T(n) \leq cn - b\sqrt{n}, \forall n \geq m$. Otteniamo quindi:

$$\begin{aligned} T(n) &= \sqrt{n}T(\sqrt{n}) + \sqrt{n} \\ &\leq \sqrt{n}(c\sqrt{n} - b\sqrt[4]{n}) + \sqrt{n} \\ &= cn - b\sqrt{n}\sqrt[4]{n} + \sqrt{n} \\ &\leq cn - b\sqrt{n} \end{aligned}$$

L'ultima disequazione è vera per ogni c, ma b deve rispettare le seguenti condizioni:

$$\begin{aligned} & - b\sqrt{n}\sqrt[4]{n} + \sqrt{n} \leq -b\sqrt{n} \\ \Leftrightarrow \quad & - b\sqrt[4]{n} + 1 \leq -b \\ \Leftrightarrow \quad & b(\sqrt[4]{n} - 1) \geq 1 \\ \Leftrightarrow \quad & b \geq \frac{1}{\sqrt[4]{n} - 1} \end{aligned}$$

Poiché $\frac{1}{\sqrt[4]{n}-1}$ è una funzione positiva decrescente per $n \geq 1$ e tende a 0 per n tendente all'infinito, possiamo scegliere un valore b che soddisfi la condizione per $n = 2$, sapendo che soddisfarrà la condizione per tutti i valori $n \geq 2$. Scegliamo quindi $b = \frac{1}{\sqrt[4]{2}-1} > 0$; questo conclude l'analisi del passo induttivo.
Ci resta quindi da dimostrare un caso base e avendo dimostrato la disequazione $n \geq 2$, scegliamo $n = 1$.

$$T(1) = 1 \leq c - b$$

che è vera per ogni b e per ogni $c \geq b + 1$.
Abbiamo quindi dimostrato che $T(n) = O(n)$, e quindi $T(n) = \Theta(n)$.

Esercizio 2

Il problema può essere risolto da un algoritmo pseudo-polinomiale con costo $O(nt)$. È possibile definire un sottoproblema $S(i, k)$ come il problema di ottenere il valore k usando i primi i numeri. Il problema iniziale è definito come $S(n, t)$. $S(i, t)$ è definito ricorsivamente nel modo seguente:

$$S(i,k) = \begin{cases} \textbf{true} & k = 0 \\ \textbf{false} & k > 0 \wedge i = 0 \\ S(i-1,k) \vee S(i, k - A[i]) & \text{altrimenti} \end{cases}$$

La formula ricorsiva può essere letta come segue:

- Se k è uguale a zero, allora la risposta è **true**: selezionando zero oggetti da un qualunque insiem di oggetti, anche vuoto, si può ottenere $k = 0$.
- Se k è maggiore di zero e non ho più oggetti, la risposta è **false**.
- Altrimenti, ho due possibilità: non prendo l'oggetto i-esimo cercando di ottenere k; prendo l'oggetto i-esimo e cerco di ottenere $k - A[i]$, ma senza eliminarlo, ovvero posso prenderlo ancora. Se uno questi sottoproblemi dà origine a **true**, allora $S(i, k)$ è **true**; da cui l'operazione **or**.

Utilizzando memoization e assumendo un vettore **boolean**$S[0 \dots n][0 \dots t]$ inizializzato a $\perp$, il codice può essere scritto nel modo seguente:

```
multisetSum(int[] A, int i, int k, boolean[][] S)
if t = 0 then
    return true
else if i = 0 then
    return false
else
    if S[i,k] = ⊥ then
        S[i,k] = multisetSum(A, i−1, k, S) or multisetSum(A, i, k−A[i], S)
    return S[i,k]
```

Esercizio 3

Vi sono almeno un paio di soluzioni. La peggiore è in $O(n^3)$, considerando tutti i possibili sottovettori e calcolando la somma per ognuno di essi.
Evitando di ricalcolare la somma di tutti i sottovettori e utilizzando le somme precedenti, è possibile ottenere un algoritmo che lavora in tempo $O(n^2)$.

```
closeToZero(int[] V, int n)
int[][] M = new int[1...n][1...n]
int min = +∞
int m_i, m_j
for i = 1 to n do
    M[i][i] = V[i]
    for j = i+1 to n do
        M[i][j] = M[i][j−1] + V[j]
        if |M[i][j]| < min then
            min = |M[i][j]|
            m_i = i
            m_j = j
return (m_i, m_j)
```

Esercizio 4

Il problema è risolvibile tramite una rete di flusso, mostrata in Figura 1. Tutti gli archi non marcati hanno peso 1.
L'idea è la seguente:

- per ogni bambino, creiamo un nodo $B_1 \dots B_n$;
- per ogni giorno della settimana g e per ogni bambino i, creiamo un nodo gadget $G_{i,g}$ (questo è necessario perchè dobbiamo porre un limite al numero di corsi che un bambino può fare al giorno, ovvero 1);
- per ogni corso tenuto nel giorno g e nell'ora o, creiamo un nodo $C_{g,o}$;
- per ogni istruttore, creiamo un nodo $i_1 \dots I_k$;
- infine creiamo i nodi sorgente e pozzo.

Per quanto riguarda gli archi,

- gli archi (S, B_i) con peso 5 servono a limitare il numero di corsi che un bambino può fare;
- gli archi $(B_i, G_{i,g})$ con peso 1 servono a limitare il numero di corsi che un bambino può fare al giorno
- gli archi $(G_{i,g}, C_{g,o})$ servono a esprimere le preferenze dei bambini;

- gli archi $(C_{g,o}, I_j)$ con peso 6 servono ad esprimere le preferenze degli istruttori;
- gli archi (I_j, P) servono a completare il grafo e hanno peso $+\infty$

Il numero di nodi con n bambini e k istruttori è pari a $1 + n + 7n + 70 + k + 1 = 8n + k + 72$. Il numero di archi è limitato superiormente da $n + 7n + 70n + 70k + k = 78n + 71k$. Il flusso massimo è limitato superiormente da $5n$; quindi il costo è $5n(8n + 4 + 72 + 78n + 71k) = O(n^2 + nk)$.

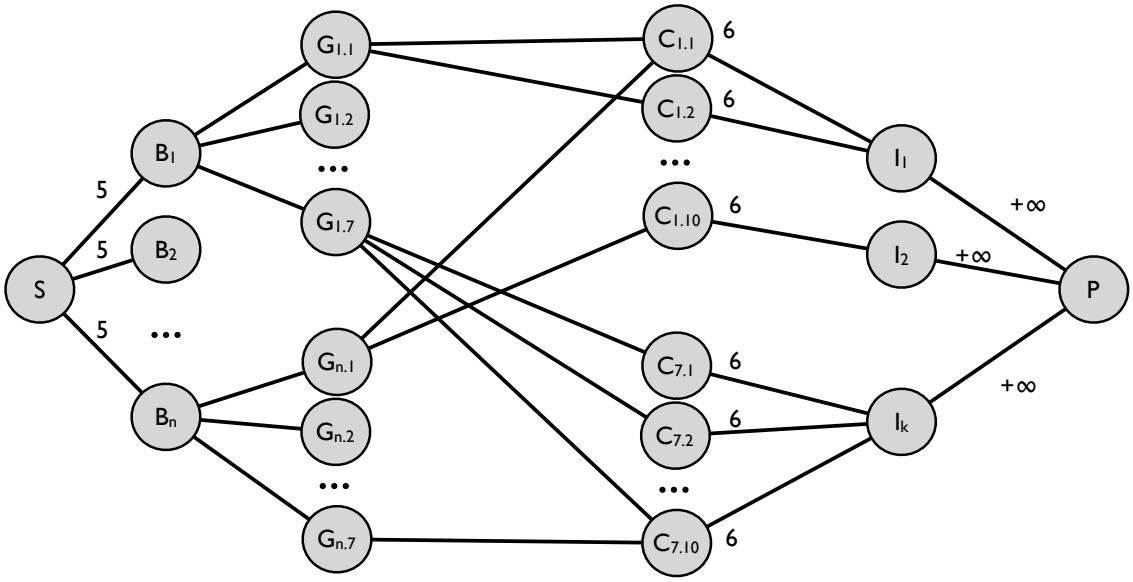

Figura 1: Rete di flusso