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

Esercizio 1

Questo esercizio è sottile. E' 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:

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

Ora, si potrebbe essere tentati di scrivere:

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

e utilizzando il Master Theorem, scrivere

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

quando in realtà si dovrebbe scrivere:

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

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

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

Notate che questo è un limite superiore. Un modo diverso per risolvere il problema è tramite sostituzione. E' possibile provare un limite più stretto, $O(n)$. Dobbiamo quindi provare che $\exists c > 0, \exists m > 0 : T(n) \leq cn, \forall n \geq m$. Otteniamo quindi:

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

L'ultima parte della disequazione è chiaramente falsa, e quindi sembra che il limite superiore non sia dimostrabile. Il problema deriva tuttavia da un elemento 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:

$$\begin{aligned} T(n) &= n^{\frac{1}{2}}T(n^{\frac{1}{2}}) + n^{\frac{1}{2}} \\ &\leq n^{\frac{1}{2}}(cn^{\frac{1}{2}} - bn^{\frac{1}{4}}) + n^{\frac{1}{2}} \\ &= cn - bn^{\frac{3}{4}} + n^{\frac{1}{2}} \\ &\leq cn - bn^{\frac{1}{2}} \end{aligned}$$

L'ultima disequazione è vera per ogni c, per ogni b e per $n \geq \frac{(b+1)^2}{b^2}$.
Analizzando il caso base, si ottiene:

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

che è vera per ogni b e per ogni $c \geq b + 1$.

Esercizio 2

Il problema può essere risolto da un algoritmo pseudo-polinomiale con costo $O(nt)$. E' 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:

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

La formula ricorsiva può essere letta come segue:

- Se k è maggiore di zero e non ho più oggetti, la risposta è **false**.
- Se k è uguale a zero, allora la risposta è **true**: un insieme vuoto di oggetti genera 0.
- Altrimenti, ho tre possibilità: prendo l'oggetto i-esimo e non lo considero più, cercando di ottenere $k - A[i]$; 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**$S[0\ldots n][0\ldots t]$ inizializzato a $\perp$, il codice può essere scritto nel modo seguente:

```
multisetSum(integer[] A, integer i, integer 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 − A[i], S) or 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(integer[] V, integer n)
integer[][] M ← new integer[1...n][1...n]
integer min ← +∞
integer 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 2 servono a limitare il numero di corsi che un bambino può fare al giorno1
- 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 $n + 7n + 490n + 70k + k = 498n + 71k$. Il flusso massimo è limitato superiormente da $5n$; quindi il costo è $5n(8n + 4 + 72 + 498n + 71k) = O(n^2 + nk)$.

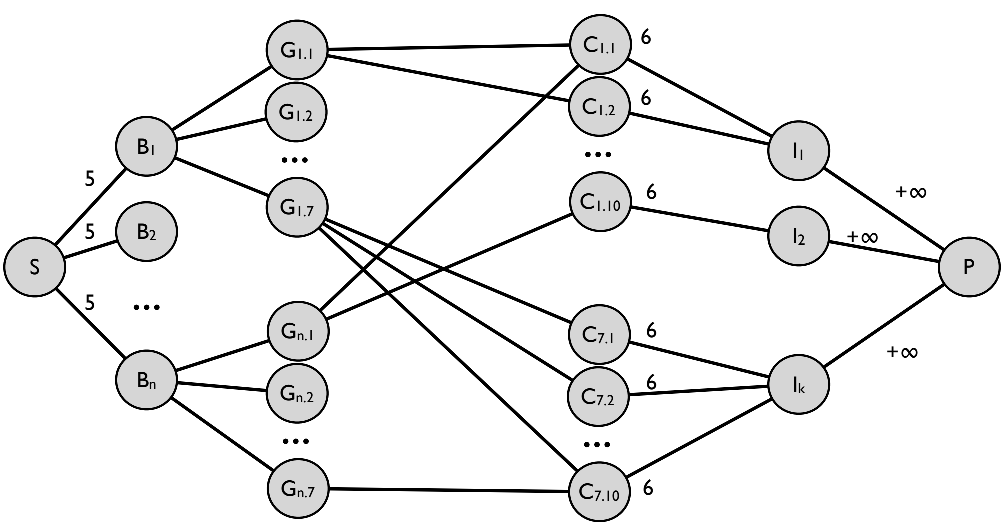

Figura 1: Rete di flusso