Algoritmi e Strutture Dati
22/07/2013

Esercizio 1

In questo particolare esercizio, non è esplicitamente richiesto di far uso del metodo di sostituzione (da qui il basso punteggio). Quindi è possibile utilizzare il master theorem, e notare che $\log_{\frac{3}{2}} 2 < 2$, e quindi siamo nel caso in cui $T(n) = O(n^2)$.
Volendo comunque utilizzare la sostituzione, cerchiamo di dimostrare che il limite superiore per $T(n)$ è n^2, ovvero che

$$\exists c > 0, \exists m \geq 0 : T(n) \leq cn^2, \forall n \geq m$$

- Caso base: $T(1) = 1 \leq c \cdot 1 \Leftrightarrow c \geq 1$
- Ipotesi induttiva: $\forall n' < n, T(n') \leq c(n')^2$
- Passo induttivo:

$$\begin{aligned} T(n) &= 2T(2n/3) + n^2 \\ &\leq 2c\frac{4}{9}n^2 + n^2 \\ &= \frac{8}{9}cn^2 + n^2 \\ &\leq cn^2 \end{aligned}$$

L'ultima disequazione è vera per $c \geq 9$; abbiamo quindi dimostrato la nostra ipotesi per $c = 9$, $m = 1$.

Per quanto riguarda il limite inferiore, è facile notare che $T(n) = 2T(2n/3) + n^2 \geq n^2$.

Esercizio 2

Questo esercizio è praticamente identitico alla massima somma di sottovettori, che abbiamo visto il primo giorno di lezione, è stato usato come secondo esempio per l'utilizzo di judge, e la cui soluzione è presente come esercizio 1.8 negli esercizi su divide-et-impera: http://disi.unitn.it/~montreso/asd/appunti/esercizi/12-divide.pdf.
Quindi, era sufficiente prendere il codice della migliore soluzione mostrata in quel file, che lavora in tempo $O(n)$, e adattarla tenendo conto che la sequenza vuota ha "produttoria" pari a 1:

```
integer maxprod(integer[] A, integer n)
    integer max ← 1                          % Massimo valore trovato
    integer here ← 1                         % Massimo valore che termina nella posizione attuale
    for i ← 1 to n do
        here ← max(here · A[i], 1)
        max ← max(here, max)
    return max
```

Esercizio 3

Nel compito originale, la formulazione era poco chiara e ha dato origine a soluzioni a problemi diversi dall'originale. La formulazione che vedete invece può essere risolta tramite la rete di flusso di Figura 1, dove i maschi e le femmine sono nodi, collegati ad una supersorgente ed ad un superpozzo, rispettivamente, con archi di peso 10; e le registrazioni fra coppie maschio-femmina sono rappresentate da archi con peso 3.
Per quando riguarda la complessità, il valore del massimo flusso è limitato superiormente da $\min\{10n, 10m\}$. Quindi utilizzando il limite definito per l'algoritmo di Ford-Fulkerson, si ottiene:

$$T(n) = O((m + n + k) \cdot \max\{m, n\})$$

dove k è il numero di registrazioni, limitato superiormente da $O(mn)$.

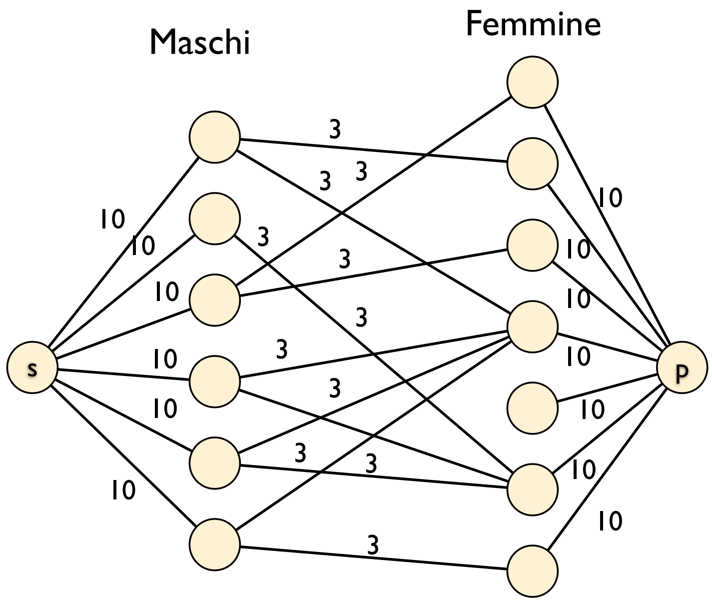

Figura 1: Rete di flusso

Esercizio 4

Per risolvere l'esercizio, è possibile osservare che il problema può essere diviso in sottoproblemi, dove il sottoproblema $P[i, j]$ è definito come la riga di monete che vanno da i a j, estremi inclusi, tale per cui $j - i + 1$ è pari. Il problema iniziale è definito da $P[1, n]$. A questo punto, è possibile definire $M[i, j]$ come il valore massimo che posso ottenere da $P[i, j]$, assumendo di muovere per primo.
Per calcolare $M[i, j]$, possono darsi due casi: devo scegliere fra i e j, *massimizzando* il mio guadagno; in entrambi i casi toccherà poi all'avversario, che sceglierà una moneta fra quelle restanti che *minimizzi* il mio guadagno (essendo un avversario intelligente). A questo punto sono state scelte 2 monete, e resto con un sottoproblema di dimensione comunque pari, in cui muovo per primo.
Dato il problema $P[i, j]$, possono darsi quindi quattro casi:

- Scelgo i: a questo punto l'avversario può scegliere fra $i + 1$ e j, lasciandomi con i sottoproblemi $P[i + 2, j]$ e $P[i + 1, j - 1]$.
- Scelgo j: a questo punto l'avversario può scegliere fra i e $j - 1$, lasciandomi con i sottoproblemi $P[i + 1, j - 1]$ e $P[i, j - 2]$.

Il problema può essere definito nel seguente modo ricorsivo:

$$M[i,j] = \begin{cases} \max\{\min\{M[i+2,j], M[i+1,j-1]\} + V[i], \min\{M[i+1,j-1], M[i,j-2]\} + V[j]\} & j > i \\ 0 & i = j \end{cases}$$

Il codice può essere espresso tramite memoization come segue:

```
integer play(integer[] V, integer i, integer j, integer[][] M)
  if i = j then
    return 0
  if M[i, j] = ⊥ then
    M[i, j] ←
    max(min(play(V, i + 2, j, M), play(V, i + 1, j − 1, M)) + V[i]), min(play(V, i + 1, j − 1, M), play(V, i, j − 2, M)) + V[j])
  return M[i, j]
```

La complessità è $O(n^2)$, in quanto è necessario riempire la matrice (in realtà non tutta; solo le celle tale per cui $i < j$ e $j - i + 1$ è pari).