Esercizi a lezione

docente: Luciano Gualà

Di seguito si trovano alcuni degli esercizi che sono stati discussi durante il corso (oltre quelli già presenti nelle slide e quelli dei Problem Set). Lo studente che vuole avere una preparazione profonda per la prova scritta è caldamente invitato a rivedere (o, meglio, studiare) le soluzioni proposte a lezione per questi esercizi.

Esercizio 1 (*notazioni asintotiche, costanti ed esponenti*)
Siano $f(n)$ e $g(n)$ due funzioni sempre positive. Si dimostri o si confuti la seguente relazione: $f(n) = O(g(n))$ implica $2^{f(n)} = O(2^{g(n)})$.

Esercizio 2
Sia $A[1;n]$ un array di n interi distinti ordinato in modo crescente. Progettare un algoritmo con complessità temporale $o(n^2)$ che, preso in input A e un valore x, dice se esistono due indici i e j tale che $A[i] + A[j] = x$.

Esercizio 3
Un array A di n elementi è detto *unimodale* se consiste di una sequenza crescente seguita da una sequenza decrescente o più precisamente se esiste un indice $m \in \{1, 2, \ldots, n\}$ tale che:

- $A[i] < A[i+1]$, per ogni $1 \leq i < m$, e
- $A[i] > A[i+1]$, per ogni $m \leq i < n$.

In particolare $A[m]$ è il massimo elemento ed è l'unico che è circondato da due elementi più piccoli ($A[m-1]$ e $A[m+1]$).

(a) Si progetti un algorimo con complessità temporale $o(n)$ che, dato un array unimodale A, restituisce l'indice dell'elemento massimo.

(b) Si progetti un algorimo con complessità temporale $o(n \log n)$ che ordina (in ordine crescente) un array unimodale A.

Esercizio 4
Sia $A[1;n]$ un vettore di n numeri. Si progetti un algoritmo che restituisca due indici i^* e j^*, con $i^* < j^*$, che massimizzano $A[j^*] - A[i^*]$, ovvero due indici tale che $A[j^*] - A[i^*] \geq A[j] - A[i]$, per ogni i, j con $i < j$. L'algoritmo deve impiegare tempo $O(n)$.

Esercizio 5
Sia T un albero binario di n nodi in cui ad ogni nodo v è associato un valore reale positivo $k(v)$ e un colore $c(v) \in \{rosso, nero\}$. Diciamo che un cammino da un nodo v alla radice è rosso se tutti i nodi lungo il cammino sono di colore rosso; inoltre definiamo il *valore* di un tale cammino come la somma dei valori dei nodi del cammino. Progettare un algoritmo che, dato T, restituisce il valore del cammino rosso di tipo nodo-radice di valore massimo.

L'algoritmo deve avere complessità temporale $O(n)$. Si assuma che l'albero è mantenuto attraverso una struttura dati collegata e che per ogni nodo siano disponibili, oltre alla chiave e al colore, i puntatori al padre e ai due figli.

Esercizio 6
Sia T un albero binario di n nodi. La profondità di un nodo v è la lunghezza (misurata in termini di numero di archi) del cammino da v alla radice. Progettare un algoritmo che, dato T e un intero h, restituisce il numero di nodi di T che hanno profondità almeno h. L'algoritmo deve avere complessità temporale $O(n)$. Si assuma che l'albero è mantenuto attraverso una struttura dati collegata e che per ogni nodo siano disponibili i puntatori al padre e ai due figli.

Esercizio 7
Sia T un albero binario di n nodi in cui ad ogni nodo v è associato un valore reale positivo $k(v)$. Diciamo che un nodo v è *bilanciato* se la somma dei valori degli antenati di v è uguale alla somma dei valori dei discendenti di v. Progettare un algoritmo che, dato T, restituisce il numero di nodi bilanciati di T. L'algoritmo deve avere complessità temporale $O(n)$. Si assuma che l'albero è mantenuto attraverso una struttura dati collegata e che per ogni nodo siano disponibili i puntatori al padre e ai due figli.

Esercizio 8
Si consideri il problema di trovare, dato un vettore ordinato $A[1;n]$ di n bit, ovvero dove $A[i] \in \{0,1\}$ per ogni i, il numero k di zeri presenti in A. Si progetti un algoritmo con complessità $O(\log n)$ e poi un miglior algoritmo con tempo $O(\log k)$.

Esercizio 9
Sia $A[1;n]$ un vettore di n bit, ovvero di zeri e di uni. Si progetti un algoritmo che restituisca un indice k tale che il numero di zeri prima di k, ovvero in $A[1;k]$, è uguale al numero di uni dopo k, ovvero in $A[k+1;n]$. L'algoritmo deve impiegare tempo $O(n)$. E' possibile risolvere il problema con la stessa complessità computazionale e usando memoria ausiliaria costante?