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. Fanno eccezione soltanto gli Esercizi 2 e 3, che sono da considerarsi come argomenti "jolly" e che quindi non fanno parte del programma d'esame. Quando sono stati presentati in aula, lo scopo del docente è stato quello di mostrare come la teoria sviluppata sulle equazioni di ricorrenza non è legata esclusivamente all'analisi di algoritmi ricorsivi, ma può essere utilizzata in altri ambiti. Nel caso specifico, l'ambito è quello della matematica/informatica ricreativa. In particolare, l'Esercizio 3 fa riferimento a un puzzle che chi scrive trova particolarmente divertente. Lo studente interessato o semplicemente incuriosito può fare riferimento all'articolo originale scaricabile al seguente url: http://arxiv.org/abs/1203.3602.

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

Esercizio 2 (*embedding di un albero binario completo su una griglia bidimensionale*)
Sia dato un albero binario completo T con n foglie e una griglia bidimensionale (foglio a quadretti). Si vuole disegnare T sulla griglia in modo da non far intrecciare gli archi e minimizzando lo spazio utilizzato. Più precisamente, un *embedding* di T è un disegno di T sul foglio tale che: (i) i nodi di T sono disegnati sulle intersezioni della griglia (ovvero come cerchi centrati su un qualche spigolo di un qualche quadratino), (ii) gli archi sono delle linee "seghettate" che seguono le linee del foglio, (iii) le linee che rappresentano gli archi non possono incrociarsi reciprocamente. Dato un embedding di T il suo *bounding box* è il rettangolo più piccolo che contiene l'intero embedding. Trovare un embedding di T che minimizza (asintoticamente) l'area del bounding box.

Esercizio 3 (*Picture-Hanging Puzzles*)
La versione più semplice di questo puzzle è la seguente. Siano dati due chiodi allineati su un muro, una corda e un quadro. Se si appendesse il quadro al muro nel modo classico (ovvero attaccando le estremità della corda al quadro e poi appendendo il quadro al muro facendo passare la corda sopra entrambi i chiodi) il quadro resterebbe comunque appeso anche rimuovendo uno dei due chiodi. L'obiettivo di questo rompicapo, invece, è quello di appendere il quadro al muro arrotolando opportunamente la corda intorno ai chiodi in modo tale che rimuovendo *uno qualsiasi* dei due chiodi il quadro (per forza di gravità) cada. Come si può fare? E se i chiodi fossero n (e si volesse appendere il quadro in modo tale che rimuovendo *uno qualsiasi* degli n chiodi il quadro cada)? Quanto deve essere lunga la corda (in funzione di n)?

Esercizio 4
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 5
Sia T un albero binario di n nodi in cui ogni nodo v ha associata una chiave reale positiva $k(v)$ 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 delle chiavi dei nodi del cammino. Progettare un algoritmo che, dato T, restituisce il valore del cammino (di tipo nodo-radice) rosso 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 A un array di n valori reali. Progettare un algoritmo che, dato A, restituisca una coppia di indici i, j con $1 \leq i < j \leq n$ che massimizza il valore $A[j] - A[i]$, ovvero tale che per ogni altra coppia di indici i', j' con $1 \leq i' < j' \leq n$ si ha $A[j] - A[i] \geq A[j'] - A[i']$. L'algoritmo deve avere complessità $o(n^2)$.

Esercizio 7
Sia T un albero AVL di n nodi con chiavi intere, distinte e positive. Dati due nodi v e v', definiamo la *distanza* fra v e v' come $d(v, v') =\mid ch(v) - ch(v') \mid$, dove $ch(u)$ denota la chiave di un generico nodo u. Progettare degli algoritmi per i seguenti problemi:

(a) dato un valore x, dire se esistono due nodi in T tali che la loro distanza è almeno x. L'algoritmo deve avere complessità temporale $O(\log n)$.

(b) trovare due nodi (distinti) in T che minimizzano la loro distanza, ovvero, trovare due nodi v e v' per i quali vale $d(v, v') \leq d(u, u')$ per ogni coppia di nodi (distinti) u, u' in T. L'algoritmo deve avere complessità temporale $o(n^2)$.

(c) dato un nodo v restituire tutti i nodi la cui distanza da v è al più 2. L'algoritmo deve avere complessità temporale $O(\log n)$.

(d) dire se esistono in T tre nodi distinti v_1, v_2, v_3 tale che $ch(v_1) + ch(v_2) + ch(v_3) = 20$. L'algoritmo deve avere complessità temporale $o(n^3)$.

Esercizio 8 (*aiutate il Re Imprenditore*)
In un paese non troppo diverso dal nostro governa un Re Imprenditore (RI). Il RI, essendo un re, ha il potere di fare le leggi e, essendo un imprenditore, ha diverse aziende da cui trae un discreto profitto. Il RI governa per n anni e può fare delle leggi che qui chiameremo Leggi Dubbie (LD). Una LD fatta nell'i-esimo anno incrementa l'utile delle aziende del re di un valore $v_i > 0$. La democrazia però, si sa, impone i suoi vincoli e infatti il RI deve far passare almeno tre anni fra una LD e l'altra (o altrimenti il popolo a gran voce si rivolterebbe!). Questo scenario suggerisce il seguente problema di ottimizzazione. Sia data una sequenza di valori positivi $v_1, \ldots, v_n$. Si vuole calcolare una strategia dubbia che

consiste in un sottoinsieme $I \subseteq \{1, 2, \ldots, n\}$ tale che per ogni $i, j \in I$ vale $|i - j| > 3$. Il valore di una strategia dubbia I è quindi $\sum_{i \in I} v_i$. Lo scopo di questo esercizio è aiutare il RI a fare il massimo profitto progettando un algoritmo polinomiale (di programmazione dinamica) che calcola la strategia dubbia ottima, ovvero quella di valore massimo.

Esercizio 9 e 10 (*insieme indipendente di valore massimo*)
Sia T un albero di n nodi in cui ad ogni nodo v è associato un valore positivo $val(v)$. Un *insieme indipendente* di un albero è un sottoinsieme I di nodi dell'albero tale che per ogni coppia $u, v \in I$, l'arco (u, v) non fa parte dell'albero, ovvero u non è padre di v e v non è padre di u. Il valore di un insieme indipendente I è definito come $\sum_{v \in I} val(v)$. Un *cammino* è un albero in cui tutti i nodi interni hanno esattamente un figlio. Progettare un algoritmo con complessità polinomiale in n che, dato un albero, calcola il valore di un insieme indipentente di valore massimo. Si studi prima il caso speciale in cui l'albero in input è un cammino.

Esercizio 11 (*il problema del distributore automatico*)
Si consideri il problema di restituire un resto R usando il minimo numero di monete di n tagli diversi $v_1, v_2, \ldots, v_n$. Si progetti un algoritmo (di programmazione dinamica) per il problema che abbia complessità temporale $O(n^2 R)$, assumendo di avere a disposizione per ogni taglio una quantità illimitata di monete.

Esercizio 12
Sia T un albero binario con n nodi e radice r in cui ogni nodo ha una chiave intera non negativa associata. Sia assuma che T è mantenuto attraverso una struttura collegata e che ogni nodo v abbia associato i seguenti campi: puntatori al padre e ai figli ($v.p$, $v.s$, $v.d$) e chiave del nodo ($v.ch$). Si progetti un algoritmo con complessità temporale $O(n)$ che, preso T, dica se esiste o meno una foglia v tale che le chiavi lungo il cammino fra r e v sono in ordine strettamente crescente. Si fornisca lo pseudocodice dettagliato dell'algoritmo.