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, tale che $f(n) = O(g(n))$. Si dimostri o si confuti la seguente relazione: $2^{f(n)} = O(2^{g(n)})$.

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
L'elemento *mediano* di un insieme S di m elementi (distinti) è quell'elemento che ha esattamente $\lfloor m/2 \rfloor$ elementi minori in S. Per esempio, l'insieme $S = \{1, 4, 6, 8, 9, 12\}$ ha mediano 8. Siano dati due insiemi A e B di n elementi ciascuno, rappresentati come sequenze ordinate memorizzate negli array $S_A[1:n]$ e $S_B[1:n]$. Progettare un algoritmo che trovi il mediano di $A \cup B$ in tempo $O(n)$ e che usi memoria ausiliaria $O(1)$. Si faccia attenzione al fatto che i due insiemi possono avere intersezione non vuota. Si fornisca lo pseudocodice dettagliato dell'algoritmo.

Esercizio 4
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 5
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 6
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 7 (*sottosequenza crescente di lunghezza massima*)
Sia dato un vettore V di n elementi. Una sottosequenza crescente di V è una sequenza di indici $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ tale che $V[i_1] \leq V[i_2] \leq \ldots V[i_k]$. La lunghezza di una tale sottosequenza è k. Progettare un algoritmo di programmazione dinamica che calcoli la lunghezza della sottosequenza crescente più lunga del vettore.

Esercizio 8 (*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 $\sum_{v \in I} val(v)$. Progettare un algoritmo con complessità polinomiale in n che, dato un albero, calcola il valore di un insieme indipentente di valore massimo.

Esercizio 9 (*il signor Marche va in vacanza*)
Il signor Marche sta pianificando i suoi n giorni di vacanza fra Roma e Firenze. Avendo a disposizione un budget limitato, vuole trovare un piano che gli faccia spendere il meno possibile. Ha raccolto i seguenti dati. Passare il giorno i-esimo a Roma gli costerebbe r_i euro, mentre a Firenze spenderebbe f_i euro. Ogni giorno può decidere se restare nella città in cui è o spostarsi nell'altra. Spostarsi di città, però, ha un costo fisso di α euro. Inoltre, dopo aver controllato i costi dei voli, si è accorto che arrivare/partire a/da Roma o Firenze è indifferente. Più formalmente, un piano è una sequenza $x = x_1 x_2 \ldots x_n$, dove per ogni $i = 1, \ldots, n$, $x_i \in \{R, F\}$ specifica se il signor Marche passa il giorno i-esimo a Roma (R) o Firenze (F). Il costo di un piano x è dato dalla somma dei costi giornalieri più gli eventuali spostamenti, ovvero $\sum_{i=1}^{n} c(x_i) + \sum_{i=2}^{n} s_i(x)$, dove $c(x_i) = r_i$ se $x_i = R$, altrimenti (se $x_i = F$) $c(x_i) = f_i$, mentre $s_i(x) = \alpha$ se $x_{i-1} \neq x_i$, 0 altrimenti.

(a) Mostrare che il seguente algoritmo non produce (sempre) un piano ottimo: per ogni $i = 1, \ldots, n$, imposta $x_i = R$ se $r_i \leq f_i$, e $x_i = F$ altrimenti.

(b) Progettare un algoritmo con complessità temporale polinomiale che, dati α, r_i, f_i, per ogni $i = 1, \ldots, n$, calcoli il costo di un piano di costo minimo.

Esercizio 10 (*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 11 *Homer e le sue donut*
Homer Simpson sta facendo una sfida in cui deve mangiare più donuts (morbide ciambelline fritte ricoperte di glassa) possibile. La sfida funziona in n round. Al round i vangono servite a Homer x_i donuts. La quantità di donuts che Homer riesce a mangiare in un certo round i dipende da quanto tempo non ha mangiato e cresce esponenzialmente con la fame. Per essere più precisi, se al round i Homer ha passato (cioè non ha mangiato ne)i precedenti j round, lui riuscirà a mangiare $\min\{2^j, x_i\}$ donuts, e la sua fame si azzerà, così che nel prossimo turno ne potrà mangiare una sola, ma se saltasse il turno successivo a quello dopo ne potrebbe mangiare 2 e così via. Assumeremo che Homer arrivi alla sfida con la pancia piena e che quindi al primo turno la sua fame gli permette di mangiare $2^0 = 1$ sola donut.

Progettare un algoritmo di programmazione dinamica che calcoli il numero massimo di donuts che Homer riesce a mangiare.

Di seguito è mostrato un esempio con $n = 4$. La strategia ottima, in questo caso, è mangiare al terzo e al quarto round.

round i:	1	2	3	4
x_i	1	10	10	1
fame:	1	2	4	8

Esercizio 12 (*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.