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
Sia $A[1:n]$ un vettore di n elementi booleani, 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?

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 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 6
Dovete valutare l'infrangibilità di un certo tipo di bicchiere di vetro e organizzate un test

per determinare l'altezza massima da cui potete far cadere il bicchiere senza che esso si rompa. Per il test, avete a disposizione una scala con n pioli e voi dovete capire qual è il più alto piolo da cui è possibile far cadere il bicchiere senza conseguenze. Chiameremo questo piolo il *più alto piolo sicuro.*

Essendo dei bravi algoritmisti, dopo un attimo di riflessione capite che se avete a disposizione tanti bicchieri uguali e non vi preoccupate di quanti ne rompete durante il processo, allora potete simulare una ricerca binaria e trovare il più alto piolo sicuro facendo al massimo $\lceil \log_2 n \rceil$ tentativi. Se invece avete a disposizione un solo bicchiere, non potete far altro che provare tutti pioli partendo dal basso; in questo modo trovate il più alto piolo sicuro in al più n tentativi.

1. Supponete di avere a disposizione 2 bicchieri. Riuscite a determinare una strategia che trovi il più alto piolo sicuro rompendo al massimo 2 bicchieri e facendo al più $f_2(n)$ tentativi, con $f_2(n) = o(n)$?

2. Generalizzate l'idea del punto precedente a un numero $k \geq 2$ di bicchieri che avete a disposizione. Più precisamente, per ogni $k = 2, 3, \ldots, \lfloor \log_2 n \rfloor$ dovete determinare una strategia che trovi il più alto piolo sicuro rompendo al massimo k bicchieri e facendo al più $f_k(n)$ tentativi, in modo tale che risulti $f_k(n) = O\left(f_{k-1}(n)\right)$ per ogni k.

Esercizio 7
Progettare un algoritmo che prende in input un vettore di numeri $A[1 \ldots n]$ e in tempo $\mathcal{O}(n)$ restituisce $\max\{A[j] - A[i] \,:\, i < j\}$.
(Per esempio, se $A = [6, 2, 4, 5, 1, 3]$, l'algoritmo deve restituire 3, ossia la differenza fra 5 e 2).

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 (*vendere al meglio una stecca di cioccolata*) Il signor Valter Bianchi, dopo aver fatto un bel gruzzoletto vendendo cristalli non proprio legali, ha deciso di diversificare la sua attività e si è messo a vendere della cioccolata. Lui ne ha una stecca. La stecca di cioccolata può essere venduta tutta intera o può essere spezzata in segmenti più piccoli da vendere separatamente. La lunghezza della stecca di cioccolata è di L centimetri, con L

intero. Si assuma che nello spezzare la stecca la lunghezza dei pezzi ottenuti (in centimetri) debba essere ancora un numero intero. Per esempio un pezzo lungo 3 centimetri può essere venduto così, spezzato in tre pezzi da 1 centimetro o in due pezzi: uno da 2 centimetri e l'altro da 1 centimetro (mentre non si possono fare due pezzi da mezzo centimetro e due centimetri e mezzo). Il guadagno che il signor Valter Bianchi riesce a fare se vende un pezzo lungo t centimetri è $G(t)$, $t = 1, 2, \ldots, L$. Progettare un algoritmo di programmazione dinamica che aiuti il signor Valter Bianchi a guadagnare il più possibile. La complessità temporale dell'algoritmo deve essere polinomiale in L.

Esercizio 10 (*sottosequanza crescente più lunga di un vettore*) 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 11 (*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.

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.

Esercizio 13 *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