Problem Set 1

Prof. Luciano Gualà

Da consegnare entro lunedì 12 dicembre 2022 agli indirizzi
`guala@mat.uniroma2.it`, `alessandrostr95@gmail.com`

Problema 1

Dato un array $V[1:n]$ di n *interi*, progettare un algoritmo che trovi in *tempo lineare* ($O(n)$) e *memoria ausiliaria costante* ($O(1)$) il minimo intero *positivo* mancante in V.

Esempi

- con $V = [9, 12, 14, -3, 0, 8, 11, -2, -3, 1]$ il risultato è 2.
- con $V = [9, 8, 0, 6, 6, 0, 9, 0, 0, 0]$ il risultato è 1.
- con $V = [0, 4, 2, 1, 1, 3, 1, 4, 4, 5]$ il risultato è 6.

Problema 2

Alessandro e Luciano sono due amici a cui piace molto mangiare. Decidono quindi di andare ad un grande evento culinario in cui vengono proposti un insieme di n *buffet*. Ogni buffet si svolge in una finestra temporale prestabilita, e visto che sono tanti, potrebbe capitare che due buffet si sovrappongano. Un esempio è mostrato in figura.

Ogni buffet è valutato dalle riviste di critica culinaria con un punteggio espresso in *stelle* ★. Un buffet valutato con 9 stelle sarà migliore di un altro valutato con 6 stelle.

Data quindi la tabella degli orari dei buffet, Alessandro e Luciano vogliono organizzare al meglio il loro tour culinario. Più precisamente loro vogliono massimizzare il tempo passato a mangiare e - soprattutto - la qualità dei buffet ai quali partecipano. Sono tipi raffinati, loro. Anche se non sembra.

Nome Buffet	Orario Inizio	Orario Fine	☆☆☆
A	10:10	12:40	4
B	14:50	18:55	5
C	16:45	20:20	7
D	10:10	13:30	12
E	16:45	17:05	5
F	16:35	17:15	6
G	18:15	20:30	14
H	10:10	14:00	10
I	14:05	18:40	13
L	13:05	14:50	12

Table 1: Esempio tabella buffet.

In particolare, la partecipazione ad ogni buffet non vincola necessariamente la partecipazione *per intero*: è possibile in ogni momento abbandonare il buffet a cui si sta partecipando per andare in un altro buffet in corso (che magari è più buono). Diciamo che la regola che i nostri eroi mangioni si sono data è semplice: in ogni momento vogliono essere al buffet che si sta svolgendo con la valutazione più alta in termini di stelle.

Potete immaginare che, se la tabella dei buffet è molto grande, non è tanto facile per Alessandro e Luciano formulare il miglior piano. Una aggravante, poi, è che nessuno dei due ha mai seguito un corso di Algoritmi e Strutture Dati.[1] Perciò dovete aiutarli voi progettando un algoritmo efficiente.
L'algoritmo dovrà calcolare un piano d'azione per i due amici, codificato come una *sequenza* di triple $\langle h_i, b_i, v_i \rangle$, con le seguenti proprietà:

1. h_i indica l'orario in cui eseguire la "mossa" i-esima della strategia.
2. b_i indica il buffet in cui i due devono spostarsi nell'ora h_i; v_i indica semplicemente il *valore* (numero di stelle ☆) del buffet b_i. Se per esempio abbiamo la tripla $\langle 10{:}30,\ C,\ 7 \rangle$ vuol dire che Alessandro e Luciano dovranno andare al buffet C di valore 7☆ alle ore 10:30 se vogliono massimizzare la qualità della loro esperienza.
3. il risultato deve essere *ordinato* per orario, in modo tale che i due amici possano leggere facilmente cosa fare. In altre parole, per ogni coppia di indici $1 \leq i < j \leq k$ (dove k è il numero di coppie del risultato) deve neccessariamente essere vero che $h_i \leq h_j$.

C'è un'ultima complicamzione. Alessandro, il più magro dei due ma sicuramente il più ingordo, è una persona diffidente (non si fida in particolar modo né di voi né degli algoritmi in generale). Dovrete quindi convincerlo *dimostrando* la correttezza dell'algoritmo proposto per il calcolo della strategia.

[1] Anche se ci sono voci di corridoio che dicono che almeno uno dei conosca a menadito gli alberi AVL.

Osservazione. *Questo a cui partecipano Alessandro e Luciano è un evento davvero molto grande: ci sono tantissimi buffet. Un algoritmo di complessità quadratica potrebbe non terminare la sua computazione in tempo utile. Cercate quindi di progettare un algoritmo con complessità* $o(n^2)$.

Problema 3

Per evitare che i ragazzi copino all'esame di Algoritmi e Strutture Dati, il professor Gualà ha disposto i ragazzi in maniera furba. La disposizione è ad *albero*, ovvero:

- in fila 1 è presente un solo (sfortunato) studente.
- in fila 2, dietro lo studente in prima fila, ne sono disposti altri due ai suoi lati. Chiamiamoli, rispettivamente, *back-vicino* sinistro e *back-vicino* destro.
- i due ragazzi in seconda fila avranno a loro volta due rispettivi back-vicini.
- e così via...

Tale disposizione, se vista dall'alto, sarà un *albero binario*. Per rendere il tutto ancora più casuale, l'albero non è regolare, ovvero non è detto che ogni studente abbia *esattamente* due back-vicini: uno studente potrebbe averne uno solo oppure nessuno.

Il professore distribuisce poi le fotocopie della traccia d'esame (esattamente n fotocopie, dove n è il numero di studenti), ma in maniera sparsa. Ovvero ogni studente i riceverà k_i fotocopie, con $k_i \geq 0$ e tali che $k_1 + ... + k_n = n$.

Prima di iniziare l'esame, è necessario distribuire le fotocopie in modo che tutti gli studenti possano svolgere il compito. Ogni studente può solamente passare un insieme di fotocopie ai due back-vicini oppure al suo (chiamiamolo) *front-vicino* nella fila d'avanti. A questo punto il professor Gualà, da buon professore di algoritmi, chiede di calcolare la *"migliore strategia"* per distribuire le fotocopie, altrimenti boccerà tutti! Quando lo studente in prima fila chiede "Migliore rispetto a che?", il professore specifica che ci sono almeno due possibili *funzioni obiettivo* da considerare.

1. Nella prima funzione obiettivo si vuole trovare una strategia che minimizzi il numero totale di fotocopie scambiate. Se uno studente passa 4 fotocopie (anche se tutte insieme) a uno dei suoi vicini, lo scambio costerà 4. Si vuole minimizzare la somma dei costi di tutti gli scambi.
2. Nella seconda funzione obiettivo, invece, si cerca una strategia che minimizzi il numero totale di *scambi* effettuati. Se uno studente per esempio passa prima un blocco di quattro fotocopie ad un suo vicino e poi un blocco di sette fotocopie da un altro vicino, lui complessivamente avrà effettuato due scambi.

Progettate dunque due algitmi che in tempo $O(n)$ calcolino le migliori strategie rispetto alle due funzioni obiettivo. Per ogni algoritmo dovete dimostrare anche la correttezza argomentando sul perché la strategia calcolata è la migliore possibile rispetto alla funzione obiettivo considerata.