Problem Set 1

Algoritmi e Strutture Dati – a.a. 2024/2025

Università di Roma "Tor Vergata"

Prof. Luciano Gualà

Linee guida per la consegna. Gli elaborati dovranno essere consegnati entro venerdì 13/12/2024 alle ore 23:59. Si prega di seguire le seguenti indicazioni:

1. le soluzioni dovranno essere consegnate via email a entrambi gli indirizzi `guala@mat.uniroma2.it` e `alessandrostr95@gmail.com`;
2. l'oggetto dell'email dovrà essere "`Consegna problem set 1 - 2024/2025`";
3. il file dovrà essere in formato pdf;
4. il nome del file dovrà essere "`PS_1_XXX.pdf`" dove `XXX` è il primo cognome (in lower case) in ordine alfabetico dei membri del gruppo;
5. l'email dovrà essere inviata tramite l'indirizzo dello studente `XXX` presente nel nome del file;
6. inserire nel file nome, cognome, indirizzo email e matricola di tutti i membri del gruppo (meglio ancora se tutti i membri del gruppo sono in indirizzo nell'email della consegna).

Buon lavoro 🔥😎!

Problema 1 *(parantesi k-bilanciate)*
Si consideri una sequenza di parentesi aperte e chiuse. La sequenza è detta *bilanciata* se ogni parentesi aperta ha una corrispondente parentesi chiusa e le coppie di parentesi sono correttamente annidate. Esempi di sequenze di parentesi bilanciate, sono:

$$(()(()())), \quad ()(()())$$

mentre esempi di sequenze non bilanciate sono:

$$)()(, \quad ())$$

Ora, una sequenza è detta *k-bilanciata* se, aggiungendo k parentesi aperte all'inizio della sequenza e k parentesi chiuse alla fine della sequenza, si ottiene un sequenza bilanciata. Alcuni esempi:

- la sequenza `)()(` è 1-bilanciata, poiché la sequenza `(()()()` è bilanciata;
- la sequenza `)()())((` è 2-bilanciata ma non 1-bilanciata;
- la sequenza `()())` non è k-bilanciata per nessun $k \geq 0$.

Progettare un algoritmo che, data una sequenza lunga n di parentesi aperte e chiuse, calcola il minimo valore di k per cui la sequenza è k-bilanciata, o restituisce $+\infty$ se non è possibile mai bilanciarla. L'algoritmo deve avere complessità temporale $O(n)$ e usare memoria ausiliaria costante. Si argomenti sulla correttezza dell'algoritmo proposto.

Problema 2 *(Il posto fisso)*
Ti hanno recentemente assunto come unico impiegato al piccolo ufficio postale del tuo quartiere. Il lavoro non ti piace e lo fai controvoglia. Però, come dice tua madre, è un posto fisso e il posto fisso è perfetto per te che, come sottolinea sempre lei, sei una persona pigra.

Ogni giorno devi servire un certo numero di clienti, diciamo n. Visto che l'ufficio è piccolo e tu sei l'unico impiegato, i clienti devono prenotarsi con una apposita app. Così tu sai quando verranno. Il cliente i-esimo arriva a tempo t_i. Questo non vuol dire che un cliente venga servito subito. Infatti, se il cliente arriva e trova lo sportello occupato da altri clienti si mette in coda e aspetta il suo turno.

Le pratiche sono sempre le stesse e sono tutte uguali. Se lavorassi alla massima velocità, ogni pratica richiederebbe Δ minuti. Ma tu, come sai perché tua madre non perde occasione per ricordartelo, sei pigro e vuoi lavorare il più lentamente possibile. Diciamo che puoi scegliere un *regime giornaliero di velocità*, ovvero puoi scegliere un valore *intero* $\Delta' \geq \Delta$ che rappresenta il numero di minuti che impiegherai per evadere *ogni singola* pratica della giornata. Ovviamente, essendo pigro, vuoi massimizzare Δ'. Il tuo unico vincolo è non essere licenziato. Tua madre ne soffrirebbe troppo. Per questo devi assicurarti di evadere tutte le pratiche entro il tempo M di chiusura.

Progettate un algoritmo che calcola il massimo valore Δ' ammissibile in tempo $o(nM)$. Si discuta la correttezza e la complessità dell'algoritmo.

Problema 3 *(Algo Run)*
È stato rilasciato un nuovo gioco per smartphone chiamato *Algo Run*. Algo Run è un arcade game ispirato a *Temple Run* in cui il protagonista, uno studente di algoritmi e strutture dati, corre lungo un corridoio suddiviso in k corsie parallele. Su ogni corsia, ci sono delle **sequenze** di *gettoni CFU*, monete di gioco (di diverso valore) che vanno raccolte se il protagonista vuole laurearsi (e quindi vincere la partita). Ogni sequenza i è descritta tramite una tupla (s_i, t_i, v_i, k_i) dove:

i) s_i indica l'istante (ovvero il punto del percorso) in cui la sequenza inizia;

ii) t_i indica l'istante (ovvero il punto del percorso) in cui la sequenza termina;

iii) v_i indica il valore in CFU di ciascun gettone della sequenza;

iv) k_i indica la corsia su cui si trova la sequenza.

Assumi le seguenti proprietà:

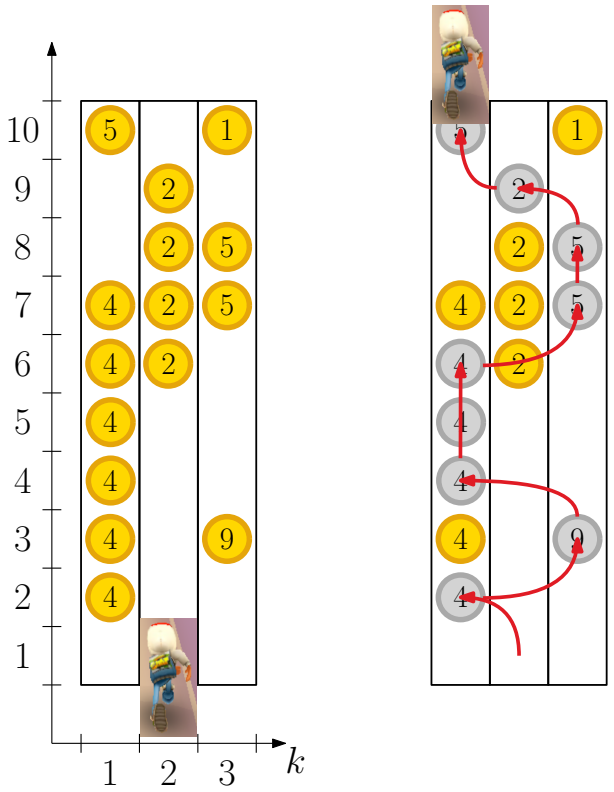
1. ogni sequenza occupa un intervallo continuo tra s_i e t_i lungo la corsia k_i, e $t_i \geq s_i$;
2. due sequenze lungo la stessa corsia non si sovrappongono mai: se $k_i = k_j$ per qualche $i \neq j$ allora $s_i > t_j$ oppure $s_j > t_i$.

Ti è permesso *saltare* da una corsia verso un'altra qualunque, e quando sei sopra una moneta essa verrà raccolta automaticamente. Lo score finale della tua partita è pari alla somma dei valori delle monete raccolte, e tu vuoi massimizzare il tuo score.

L'input del gioco è quindi:

- un intero k: il numero di corsie;
- un intero n: il numero di sequenze di monete;
- una lista S di n tuple (s_i, t_i, v_i, k_i).

Un esempio di istanza e la corrispondente strategia che massimizza lo score è mostrata in Figura 1. Progettate un algoritmo efficiente di tempo $O(n \log n)$ che calcola il massimo score ottenibile.



Input:

- $k = 3$;
- $n = 6$;
- $S = [\,(2,7,4,1),\ (3,3,9,3),\ (6,9,2,2),\ (7,8,5,3),\ (10,10,5,1),\ (10,10,1,3)]$.

Figure 1: Lo score (ottimo) ottenuto dalla strategia mostrata è 42.