Problem Set 1

Algoritmi e Strutture Dati – a.a. 2023/2024

Università di Roma "Tor Vergata"

Prof. Luciano Gualà

Da consegnare entro lunedì 04/12/2023 agli indirizzi
`guala@mat.uniroma2.it`, `alessandrostr95@gmail.com`

Problema 1 *(Il barman algoritmista)*

Il dott. Straziota, per arrotondare la misera borsa di dottorato, durante i week end ha cominciato a lavorare come barman in un pub sulla Tuscolana. Una sera arrivano n persone e si siedono agli n sgabelli del bancone. Le ragazze ordinano tutte una pinta di Pale **A**le (tipo A), un tipo di birra leggera, di colore chiaro e delicati aromi di luppolo. I ragazzi, invece, ordinano tutti una pinta di **B**ock (tipo B), una birra rossa dalla gradazione alta caratterizzata da note maltate ricche e cremose e una punta di cacao sul finale.

Ora, succede che il collega del dott. Straziota, probabilmente un dottorando in lettere e filosofia, dopo aver spillato le birre, le ha servite agli n clienti senza badare alle tipologie. Così, adesso c'è da riordinare i boccali in modo che ognuno possa bere la birra che ha ordinato. I clienti non vogliono alzarsi dagli sgabelli e lanciare i bicchieri lungo il bancone è assolutamente da escludere, dato che rischierebbe di rovesciare la bevanda. Inoltre il bancone è stretto e quindi c'è posto per un solo boccale davanti ad ogni cliente. L'unica operazione possibile è quindi quella di *scambio* di due boccali adiacenti. Riuscite ad aiutare il dott. Straziota a rimediare alla situazione (la cosa potrebbe fargli guadagnare una buona dose di mance)?

In particolare:

- Dimostrate che $O(n^2)$ scambi sono sempre sufficienti per riordinare le pinte.
- Dimostrate che $\Omega(n^2)$ scambi sono necessari nel caso peggiore.
- Progettate un algoritmo che calcolare la sequenza più corta di scambi necessaria per riordinare i boccali. L'algoritmo deve avere complessità $O(n+K)$, dove K è la lunghezza di una sequenza ottima di scambi. Ovviamente dovete dimostrare che il vostro algoritmo è corretto, ovvero calcola

sempre una sequenza di scambi di lunghezza minima, e argomentare sulla complessità temporale dell'algoritmo.

Problema 2

In questo esercizio dovete progettare un *Game Over Checker* per il gioco del *Tetris* con pezzi rettangolari, ovvero un algoritmo che, preso in input una sequenza di mosse giocate da un giocatore, calcola efficientemente se il giocatore ha perso la partita o meno.

Più formalmente, avete una griglia bidimensionale larga n celle e alta m, e una sequenza di N mosse. L'i-esima mossa T_i è una tripla di interi positivi (x_i, w_i, h_i), dove:

- w_i e h_i sono rispettivamente la larghezza e l'altezza dell'i-esimo pezzo rettangolare;
- $x_i \in \{1, \ldots, n\}$ specifica la coordinata a cui il pezzo i-esimo è lasciato cadere. Più precisamente, il pezzo è lasciato cadere da altezza $m+1$ dopo aver posizionato il rettangolo in modo che lo spigolo in basso a sinistra si trovi ad ascissa x_i. Una volta lasciato cadere il pezzo si muove verso il basso soggetto a forza di gravità e si ferma appena incontra un pezzo precedentemente piazzato nella griglia o il bordo inferiore della griglia stessa.[1]

Una generica mossa innesca il *game over* se, una volta lasciato cadere, il pezzo corrispondente non è interamente contenuto nella griglia, ovvero parte del rettangolo è ad un'altezza superiore ad m. Un esempio di sequenza è mostrata in figura, dove la terza mossa innesca il game over.

Il vostro obiettivo è progettare un algoritmo che, preso in input la sequenza delle N mosse, determina, se esiste, l'indice della prima mossa che innesca il game over, o dichiara che il giocatore non ha ancora perso la partita. La complessità temporale dell'algoritmo deve essere $O(N\sqrt{n})$, assumendo $N = \Omega(n)$.

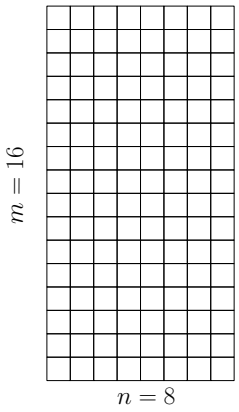
Esempio

Considera la seguente istanza

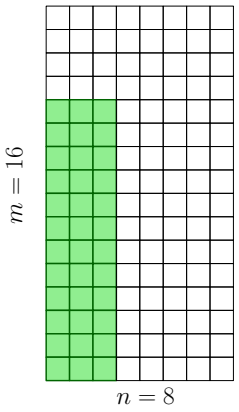
- $n = 8$;
- $m = 16$;
- sequenza di $N = 7$ mosse $(1, 3, 12)$, $(6, 3, 3)$, $(2, 5, 2)$, $(8, 1, 5)$, $(4, 2, 3)$, $(2, 2, 2)$, $(4, 3, 7)$.

L'esito della partita è un game over in 5 mosse.

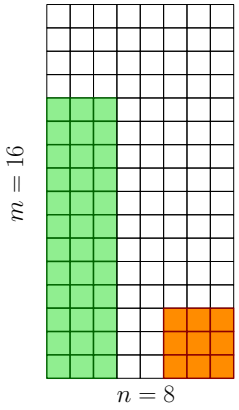
[1] Si assuma per semplicità che, diversamente dal gioco classico del Tetris, le linee piene non vengono rimosse e che quindi un pezzo posizionato nella griglia resta lì per tutta la durata della partita.



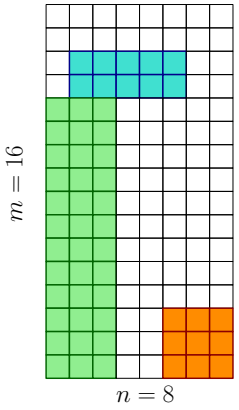
(a) Inizio della partita



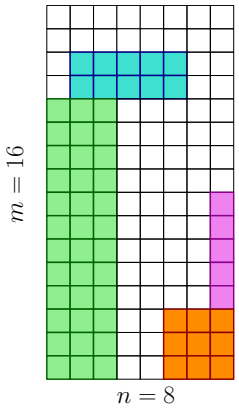
(b) Mossa 1: $(1, 3, 12)$



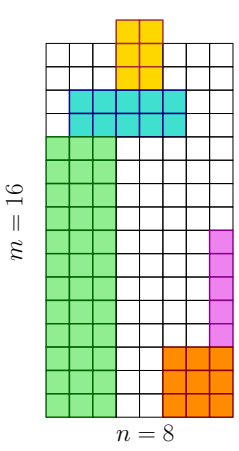
(c) Mossa 2: $(6, 3, 3)$



(d) Mossa 3: $(2, 5, 2)$



(e) Mossa 4: $(8, 1, 5)$



(f) Mossa 5: $(4, 2, 3)$ – GAME OVER!

Figure 1

Problema 3

In questo esercizio vi è richiesto di progettare tre diverse strutture dati in grado di rispondere a tre diversi tipi di *query* definite su un albero preso in input. In particolare, sia $T = (V, E)$ un albero *pesato* di n nodi con funzione di peso $w : E \to \mathbb{N}^+$ che associa ad ogni arco e di T un peso intero positivo $w(e)$.

In ognuno dei tre problemi vi è chiesto di processare T in modo da costruire una struttura dati che sia in grado di rispondere a query di un certo tipo. Mentre non ci sono vincoli sulla complessità temporale dell'algoritmo che costruisce la struttura dati, saranno invece imposti dei vincoli sulla dimensione massima della struttura dati costruita e sul tempo necessario alla struttura dati per rispondere a una generica query.[2]

Di seguito sono definiti i tre problemi.

Distance

Processate l'albero T in modo da costruire una struttura dati in grado di rispondere alla seguente query:

- `dist`(v, u): dati due nodi $v, u \in V$, con il vincolo che u è un antenato di v (non ti dovrai preoccupare di fare questo controllo, assumi sempre che riceverai in input un nodo u che è antenato di v), restituire la lunghezza (intesa come somma dei pesi degli archi) del cammino in T che collega v a u.

La struttura dati deve avere dimensione $O(n)$ e il tempo di query deve essere $O(1)$.

Level Ancestor

Processate l'albero T in modo da costruire una struttura dati in grado di rispondere alla seguente query:

- `LA`(v, k): dati un nodo $v \in V$ e un intero positivo k, restituire il k-esimo antenato di v lungo il cammino da v verso la radice, ovvero, restituire il nodo lungo tale cammino che è k livelli sopra v.

La struttura dati deve avere dimensione $O(n \log n)$ e il tempo di query deve essere $O(\log n)$.

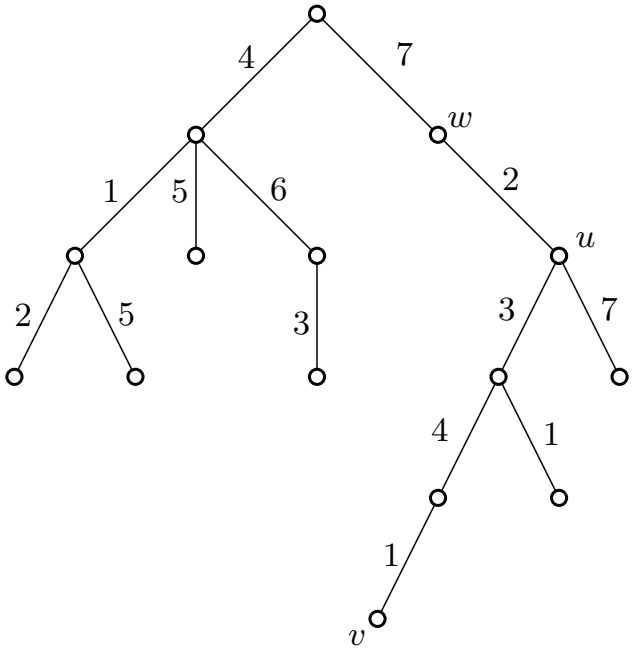
Weighted Level Ancestor (extra)

Il seguente problema è una generalizzazione del precedente. Processate l'albero T in modo da costruire una struttura dati in grado di rispondere alla seguente query:

[2] Sarà il vincolo sulla dimensione della struttura dati a rendere i problemi interessanti. Infatti senza alcun vincolo sullo spazio, per tutti i problemi considerati sarebbe possibile pre-calcolare le risposte a tutte le possibili query che, una volta memorizzate in una tabella, permetterebbero alla struttura dati di rispondere in tempo costate semplicemente accedendo all'opportuna casella della tabella. Troppo facile così, no?

- `wLA`(v, Δ): dati un nodo $v \in V$ e un intero positivo Δ, restituire il più profondo antenato u di v tale che la distanza da v ad u è almeno Δ.[3]

La struttura dati deve avere dimensione $O(n \log n)$ e il tempo di query deve essere $O(\log n)$.



$\texttt{dist}(v, u) = 8$

$\texttt{LA}(v, 3) = u$

$\texttt{wLA}(v, 9) = w$

Figure 2

[3] Riesci a vedere perché questo problema è una generalizzazione del precedente?