Problem Set 2

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

Università di Roma "Tor Vergata"

Prof. Luciano Gualà

Importanti avvertenze preliminari:

1. Il gruppo deve essere formato da almeno tre persone!
2. Per lo svolgimento si consiglia fortemente l'utilizzo del template LaTeX disponibile nella pagina web del corso.

Linee guida per la consegna. Gli elaborati dovranno essere consegnati entro venerdì 31/01/2025 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 2 - 2024/2025`";
3. il file dovrà essere in formato pdf;
4. il nome del file dovrà essere "`PS_2_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 *(Bloxorz)*
Bloxorz è un puzzle game in cui ogni livello consiste in una mappa a griglia. Per completare un livello è necessario far "rotolare" un parallelepipedo di dimensione $2 \times 1 \times 1$ fino a raggiungere una specifica posizione della mappa. Un esempio di livello è riportato in Figura 2.

All'inizio del livello, il parallelepipedo è *sempre posizionato in piedi* su una determinata cella della mappa. L'input del problema è quindi costituito dalla mappa del livello, dalla posizione iniziale del parallelepipedo e dalla posizione finale desiderata.

Il parallelepipedo può essere fatto rotolare in quattro possibili direzioni: su, giù, destra e sinistra. Ogni movimento modifica sia la posizione sia l'orientamento del parallelepipedo (vedi Figura 1 per un esempio di movimento e Figura 2 per una soluzione di un livello).

L'obiettivo è raggiungere, con il *minor numero di mosse possibile*, la configurazione finale in cui il parallelepipedo si trova in piedi sulla cella finale, assicurandosi che nessuna parte del parallelepipedo esca mai dalla mappa.

La mappa del livello è rappresentata da una matrice M di dimensione $n \times n$, dove

- $M[i, j] = 1$ indica una cella percorribile della mappa;
- $M[i, j] = -1$ indica una cella non percorribile della mappa (in cui il parallelepipedo non può mai finirci, né per intero né parzialmente).

L'input del problema comprende quindi la matrice M, la posizione iniziale (i, j) del parallelepipedo (posizionato in piedi), e la posizione finale (s, t).

Progettare un algoritmo che determini se esiste una soluzione al livello e, in tal caso, restituisca la sequenza più corta di mosse necessaria per raggiungere la posizione finale. In caso non esista soluzione, l'algoritmo dovrà restituire `False`. L'algoritmo dovrà in oltre avere complessità temporale e spaziale *lineare* nel numero di celle, ovvero $O(n^2)$.

Parte 2 *(Bloxorz Level 2)* Se siete arrivati a questo punto, vuol dire che avete superato il primo livello: ben fatto![1]

Nel secondo livello, la mappa include dei buchi che impedisco il parallelepipedo di raggiungere la posizione finale. In particolare ci sono buchi di colore *blu* e buchi di colore *rosso*. Nella mappa sono anche presenti dei *bottoni*, alcuni di colore rosso ed altri di colore blu. Quando il parallelepipedo tocca uno qualsiasi dei bottoni blu allora tutti i buchi blu diventano percorribili da quel momento in poi. Stessa cosa per i bottoni rossi. La Figura 3 mostra un esempio di livello con relativa soluzione.

In questo caso la matrice M assume i seguenti valori:

- $M[i, j] = 1$ indica una cella percorribile della mappa;
- $M[i, j] = -1$ indica una cella non percorribile della mappa (in cui il parallelepipedo non può mai finirci, né per intero né parzialmente);

[1] In caso contrario vuol dire che avete barato e avete saltato il primo livello.

- $M[i, j] = B$ indica un buco blu;
- $M[i, j] = R$ indica un buco rosso;
- $M[i, j] = b$ indica un bottone blu;
- $M[i, j] = r$ indica un bottone rosso.

Progettare un algoritmo che risolva il secondo livello, con stessa complessità temporale e spaziale per l'algoritmo del primo livello.

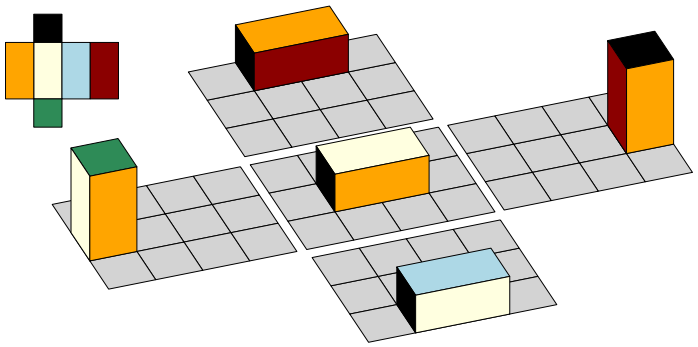
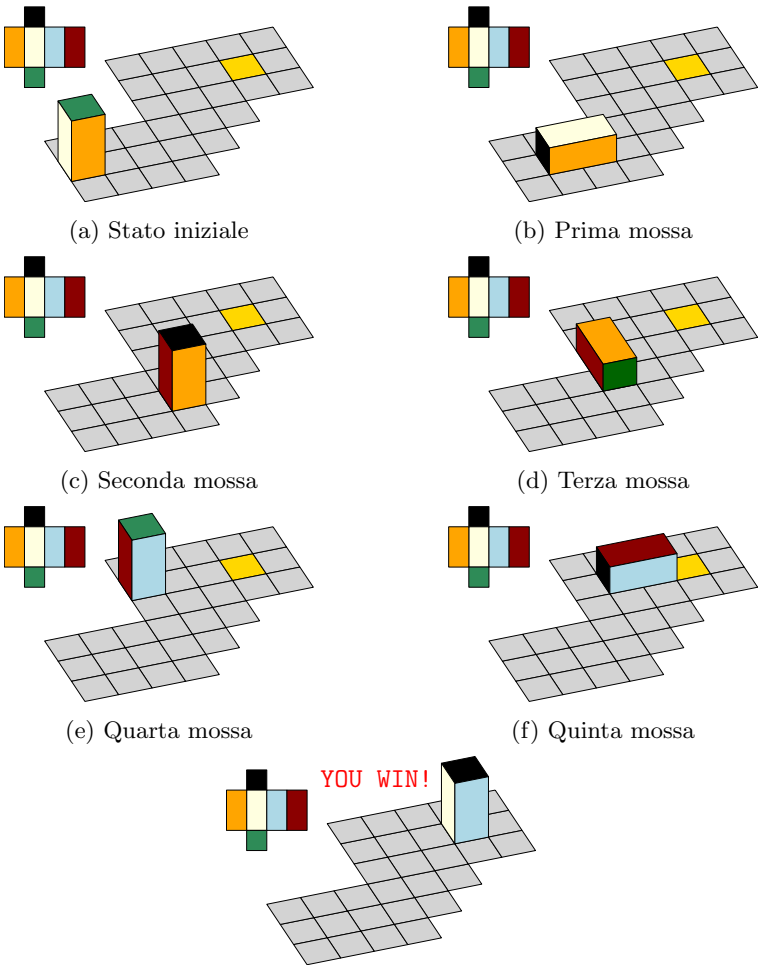


Figure 1: Quattro possibili mosse, a partire dalla configurazione centrale.



(a) Stato iniziale

(b) Prima mossa

(c) Seconda mossa

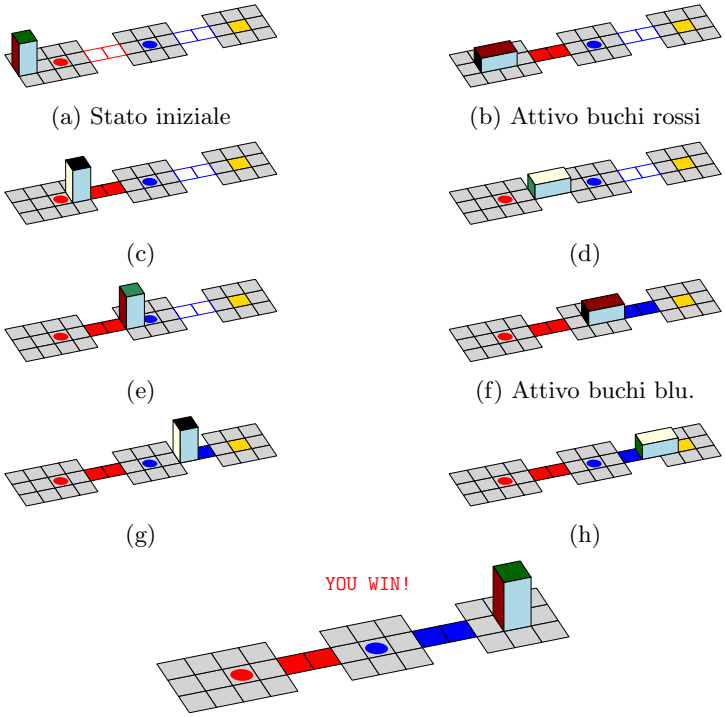
(d) Terza mossa

(e) Quarta mossa

(f) Quinta mossa

(g) Ultima mossa

Figure 2: Esempio di livello con relativa soluzione.



(a) Stato iniziale

(b) Attivo buchi rossi

(c)

(d)

(e)

(f) Attivo buchi blu.

(g)

(h)

(i) Ultima mossa

Figure 3: Esempio di secondo livello.

Problema 2 *(le cene fra algoritmisti)*
Il prof. Gualà invita spesso a cena il dot. maestro Straziota che, da buon ospite, non si presenta mai senza aver prima comprato una bottiglia di vino. Ci sono tante enoteche in città e ogni enoteca ha il suo orario di apertura. Questo esercizio vi chiede di aiutare Straziota progettando una struttura dati che lui potrà agevolmente interrogare per scoprire ogni volta la miglior strada che dovrà fare da casa sua a casa del prof. Gualà per arrivare a cena con una bottiglia di vino.

Più formalmente, sia $G = (V, E, c)$ un grafo non orientato e pesato di n nodi ed m archi, dove ad ogni arco e è associato un costo $c(e) \geq 0$. Sia inoltre $W \subseteq V$ un insieme di nodi detti *enoteche*. Un *wine path* da un nodo s ad un nodo t è un cammino da s a t in G, non necessariamente semplice[2], che attraversa almeno un'enoteca, ovvero un nodo di W. Il costo di un wine path da s a t è definito come la somma dei costi degli archi che attraversa (un arco attraversato più volte viene sommato più volte). Il costo di un wine path di costo minimo è detto *wine distance*.

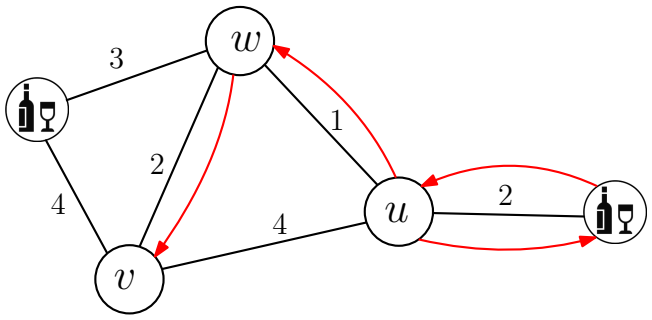


Figure 4: Un wine path da u a v di costo $2+2+1+2=7$, che è anche la wine distance da u a v.

Ora, ogni enoteca $w \in W$ ha un orario di apertura $\tau_w \geq 0$. Un wine path è detto *attivo a tempo* τ se passa per una enoteca che è aperta a tempo τ, ovvero se attraversa un'enoteca w tale che $\tau_w \leq \tau$. Il costo del miglior wine path fra s e t attivo a tempo τ è detta *wine distance a tempo* τ fra s e t.

Il vostro obiettivo è progettare una struttura dati che aiuti il dot. maestro Straziota a capire che strada fare per andare a casa del prof. Gualà dipendentemente dall'orario dell'invito.

Più formalmente, progettate una struttura dati che, dato il grafo G, l'insieme delle enoteche W, e due nodi s e t (rispettivamente la casa di Straziota e di Gualà), è in grado di rispondere alle seguenti query:

- `Wdist`(τ): restituisce la wine distance fra s e t a tempo τ.
- `Wpath`(τ): restituisce il miglior wine path fra s e t attivo a tempo τ.

[2]Si ricorda che un cammino è detto *semplice* se non attraversa alcuno nodo più di una volta.

La prima query deve avere complessità $O(\log n)$, mentre la seconda deve richiedere tempo $O(\log n+k)$, dove k è il numero di archi del wine path restituito. La struttura dati inoltre deve poter essere costruita in tempo $O(m + n \log n)$ e occupare spazio $O(n)$.

Una generalizzazione (per i più impavidi).[3] Modificare la struttura dati precedente in modo da supportare la seguente ulteriore operazione:

- `open`(w, τ_w): dato un nodo $w \in V \setminus W$, apre una nuova enoteca in w il cui orario di apertura è τ_w.

Le complessità sul tempo di costruzione della struttura dati, il tempo delle query e lo spazio occupato devono essere le stesse. La nuova operazione, inoltre, deve richiedere tempo *ammortizzato* $O(\log n)$, assumendo che all'inizio non ci sono enoteche aperte.

[3] Questo punto è opzionale (per una attività già opzionale di per sé come i Problem Set). Da svolgere solo se si è abbastanza coraggiosi e ci si sta divertendo adeguatamente con gli algoritmi.