Algoritmi e Strutture Dati (modulo II) - Esercitazione 06/04/2023
(Soluzioni)

Cognome: .... Nome: .... Matricola: ....

Esercizio 1 [MST] Sia $G = (V, E)$ un grafo non diretto e pesato con pesi non negativi e tutti distinti. Dire quali tra le seguenti frasi sono vere e quali false.

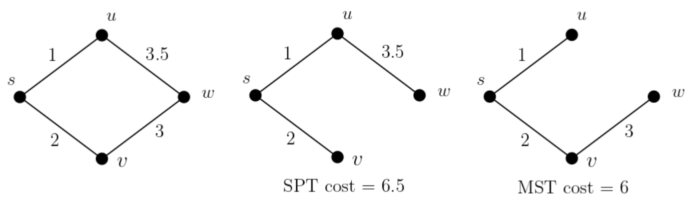
- [ ] Sia T_1 un MST calcolato con l'algoritmo di Prim, e T_2 il MST calcolato con l'algoritmo di Kruskal. Allora non è detto che T_1 e T_2 siano uguali. FALSO. Dato che i pesi sono tutti distinti, allora il MST è unico.

- [ ] Siano T_1, T_2 due MST calcolati entrambi con l'algoritmo di Prim, ma partendo da due sorgenti differenti $s_1 \neq s_2$. Allora $T_1 \not\equiv T_2$. FALSO. Dato che i pesi sono tutti distinti, allora il MST è unico.

- [ ] Sia e l'arco di peso minimo presente nel MST calcolato dall'algoritmo di Kruskal. Allora e appartiene ad ogni MST. VERO. Dato che i pesi sono tutti distinti, allora il MST è unico. Si può verificare anche con la CUT property.

- [ ] Sia e l'arco di peso massimo presente nel MST calcolato dall'algoritmo di Kruskal. Allora e appartiene ad ogni MST. VERO. Dato che i pesi sono tutti distinti, allora il MST è unico.

- [ ] Sia s il nodo da cui parte l'algoritmo di Prim per calcolare il MST. Dato che i pesi sono tutti diversi, allora l'albero risultante è anche uno shortest path tree radicato in s. FALSO.



- [ ] Siano $C_1, ..., C_k$ le *componenti connesse* calcolate dopo $n - k$ iterazioni dell'algoritmo di Kruskal. Siano due nodi u, v appartenenti ad una stessa componente connessa C_i (per qualche $1 \leq i \leq k$). Sia invece T il MST calcolato alla fine dell'algoritmo. Definiamo con $d_{C_i}(u, v)$ la distanza tra u, v all'interno della componente connessa C_i, e con $d_T(u, v)$ la distanza tra u, v all'interno del MST T. Allora è vero che $d_{C_i}(u, v) \leq d_T(u, v)$. VERO. In realtà $d_{C_i}(u, v) = d_T(u, v)$.

- [ ] Supponiamo che il grafo G sia *non connesso*, ed eseguendo l'algoritmo di Prim partendo da un generico nodo s otteniamo un albero T. Allora T è un MST per la sola componente connessa C_s alla quale s appartiene. VERO. Perché l'algoritmo di Primo esegue una visita partendo dal nodo s, la quale esplorerà solo la sua componente connessa C_s.

- [ ] Supponiamo che il grafo G sia *non connesso*. Eseguendo l'algoritmo di Prim partendo da un generico nodo s otteniamo due alberi T_1, T_2. T_1 è il MST per la componente connessa C_s alla quale s appartiene, e T_2 è il MST per il restante sottografo indotto dai nodi $V \setminus C_s$. FALSO. La visita si limiterà alla sola componente connessa C_s.

- [ ] Supponiamo che il grafo G sia non connesso. Allora eseguendo l'algoritmo di Kruskal otterremo due MST per ciascuna delle componenti connesse. FALSO. In realtà otterremo un solo MST per ciascuna delle componenti connesse di G.

Esercizio 2 [Union-Find] Sia $U = \{1, 2, ..., n\}$ un universo di n elementi. Si consideri la struttura dati *Quick-Find* con euristica *union-by-size*, su tale universo U. A seguito di n operazioni di `makeSet`, partiamo da una configurazione iniziale di n singleton.

1. Dopo una sequenza arbitraria di $< \lceil n/2 \rceil$ operazioni di `union`, rimane almeno un insieme contentente un singolo oggetto? Giustificare brevemente la risposta (Max 2 righe)
   Nel caso di n pari è banale, in quanto servono almeno $n/2$ operazioni di `union` per non avere più insiemi con un singolo elemento. Nel caso di n dispari, basta osservare che servono almeno $(n-1)/2 = \lfloor n/2 \rfloor$ `union` per non avere più singleton per i primi $n-1$ elementi (i quali sono pari). Ne rimarrà quindi uno scoperto.

2. Si enunci in modo preciso le prestazioni della struttura dati, in termini di costi delle operazioni della struttura dati. (Max 5 righe) Il costo delle singole operazioni di `makeSet` è costante, in quanto basta creare un singleton. Anche le singole operazioni di find hanno costo costante, in quanto gli alberi che rappresentano gli insiemi hanno tutti altezza 1. In conclusione, dato che si sta adottando l'euristica *union-by-size*, si può dimostrare che a fronte di $n-1$ operazioni di `union` ogni elemento cambia padre al più $\log n$ volte. Perciò, il costo di n operazioni di `makeSet`, $n-1$ `union` ed m `find` sarà $O(m + n + n \log n) \subseteq O(m + n \log n)$.

Esercizion 3 [Interval Partitioning] Si consideri il problema di ottimizzazione *Interval Partitioning*.

1. Definire formalmente l'istanza del problema, quali sono le soluzioni ammissibili e cosa si vuole ottimizzare. (Max 2 righe)
   - **Input:** $\langle \mathcal{I} \equiv \{I_1, ..., I_n\};\ s, f : \mathcal{I} \to \mathbb{R}^+ \rangle$ t.c. $\forall j = 1, ..., n\ \ s(I_j) \leq f(I_j)$.
   - **Feasible Solution:** una *partizione* $\mathcal{C} \equiv \{C_1, ..., C_k \subseteq \mathcal{I}\}$, ovvero t.c. $\forall 1 \leq i < j \leq k\ C_i \cap C_j \equiv \emptyset$ e $\bigcup_{i=1}^{k} C_i \equiv \mathcal{I}$, e tale che per ogni $i = 1, ..., k$ tutti i job di C_i sono *compatibili*, ovvero $\forall I \neq I' \in C_i$ abbiamo che $(s(I), f(I)) \cap (s(I'), f(I')) \equiv \emptyset$.
   - **Goal:** minimizzare $|\mathcal{C}|$.

2. Data una generica istanza del problema, qual è il minor numero di risorse necessarie per una qualsiasi soluzione ammissibile? Giustificare brevemente la risposta. (Max 2 righe) Il minor numero di risorse necessarie per una generica istanza è la sua *depth* d, ovvero il maggior numero di job non compatibili. Ovviamente con $d-1$ risorse disponibili non sarebbe possibile schedulare i d jobs non compatibili in maniera appropriata.

3. Se come politica greedy utilizzassimo l'*earliest finish time* (anziché l'*earliest starting time*) l'algoritmo visto a lezione garantirebbe ancora l'ottimalità? Se NO, mostrare un controesempio. Se SI, arogmentare brevemente il perché. (Max 5 righe) L'algoritmo è ancora ottimale, e si può tranquillamente applicare la stessa argomentazione della politica greedy *earliest starting time*. Consideriamo i d jobs *non compatibili* $I_1, ..., I_d$, enumerati in ordine di *finish time*. Dato che sono tutti incompatibili, allora certamente è vero che $\forall 1 \leq i < j \leq d$ abbiamo che $s(I_j) \leq f(I_i)$ (convincersi di questo). Senza perdita di generalità, assumiamo che nel momento in cui viene considerato l'ultimo job I_d abbiamo $d-1$ risorse allocate. Dato che $s(I_d) \leq f(I_j)$ per ogni $j < d$, allora sarà necessaria per forza un'ulteriore risorsa, per un totale di d risorse complessive $\square$.