Esercitazione 3

Problema 6: Sia $G = (V, E)$ un grafo con pesi distinti sugli archi ed $e \in E$ un arco di G. Progettare un algoritmo lineare in grado di determinare se esiste un MST di G che contiene l'arco e.

Soluzione Osserviamo che, dato un grafo con pesi distinti, questo ammette un unico MST.

Dimostrazione. Supponiamo per assurdo che esistano due MST distinti, diciamo T e T'. Sia e l'arco di peso minimo fra tutti gli archi che appartengono ad un solo albero. Senza perdita di generalità si assuma che e appartenga a T. L'insieme $\{e\} \cup E(T')$ deve per forza contenere un ciclo, ed almeno uno degli archi di tale ciclo, indicato con f non è in T (altrimenti T non sarebbe aciclico). Dal fatto che il peso degli archi è distinto ed f appartiene ad un solo albero, si ha che $w(f) > w(e)$ e quindi l'insieme $\{e\} \cup E(T') \setminus \{f\}$ costituisce un albero ricoprente di peso minore rispetto a T', contraddicendo l'ipotesi che T' sia un MST. □

Dato che il grafo G in input al problema ha pesi distinti, possiamo sfruttare la proprietà appena dimostrata nel seguente modo: si applica un algoritmo per il calcolo dell'MST e una volta ottenuto l'albero T risultante, essendo questo unico, ci basta verificare se $e \in T$ per rispondere alla domanda postaci dal problema.

Per quanto riguarda la complessità della procedura descritta, dobbiamo effettuare il calcolo dell'MST che, può essere effettuato con l'algoritmo di Prim in tempo $O(|E| + |V| \log |V|)$. Una volta ottenuto l'albero T per verificare se l'arco e si trova in T o no, basta effettuare una visita di tale albero in tempo $O(|V|)$.

In definitiva l'algoritmo proposto ha complessità totale $O(|E| + |V| \log |V|)$, quindi non in linea con i vincoli di efficienza posti dal problema.

È quindi necessario trovare un modo più "furbo" per determinare se l'arco appartiene o meno all'MST.

L'idea è quella di non calcolare l'intero MST ma di sfruttare le regole del taglio (dato un taglio l'arco più leggero che lo attraversa appartiene all'MST) e del ciclo (dato un ciclo l'arco più pesante di tale ciclo non appartiene all'MST) per determinare se l'arco in questione appartiene o meno al minimo albero ricoprente.

In particolare dato G costruiremo un grafo ausiliario G' rimuovendo da G tutti gli archi con peso maggiore o uguale al peso di e (e incluso), una volta ottenuto il grafo G' sarà sufficiente effettuare una visita da uno dei nodi agli estremi dell'arco e, nel caso l'altro estremo sia raggiungibile diremo che $e \notin MST(G)$ altrimenti diremo che $e \in MST(G)$.

Di seguito è riportato un semplice pseudocodice di quanto appena descritto:

```
   Data: G = (V, E), e = (u, v) ∈ E
1  begin
2      forall (x, y) ∈ E do
3          if w(x, y) ≥ w(e) then
4              E ⟵ E \ {(u, v)}
5      T ⟵ BFS(u)
6      if v ∈ T then
7          return (e ∉ MST)
8      return (e ∈ MST)
```

Per quanto riguarda la complessità, considerando istanze in cui G è connesso, abbiamo che la fase di rimozione degli archi ha costo $O(|E|)$, la visita BFS ha costo $O(|E|)$, ed infine per verificare che il nodo v sia raggiungibile da u si ha costo $O(|V|)$ (visita dell'albero T). Quindi in totale l'algoritmo proposto ha costo $O(|E|)$ (lineare nella dimensione dell'istanza).

In figura è mostrata l'esecuzione dell'algoritmo per due diversi valori dell'arco e.

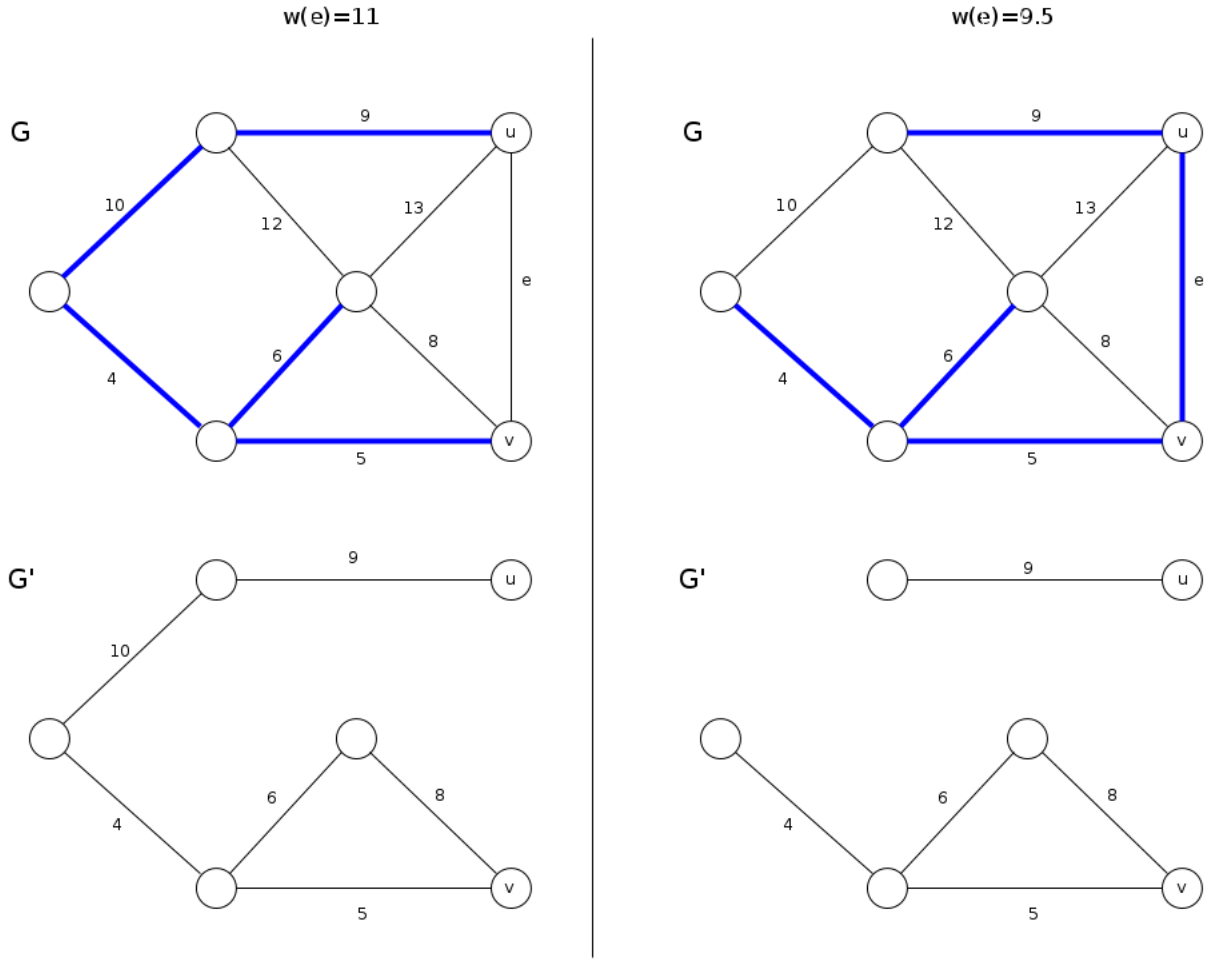
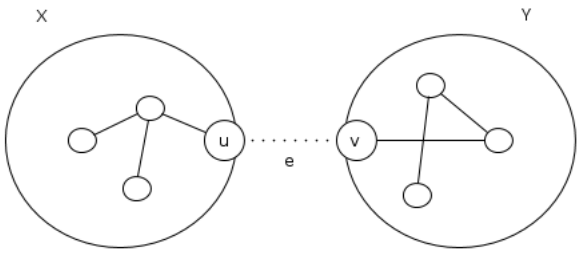


Figura 1: Esempio riportante due istanze del problema, nei grafi sono evidenziati in blu gli archi appartenenti all'MST. Nella figura di sinistra (caso $w(e) = 11$) possiamo vedere che $e \notin MST(G)$ e in effetti in G' il nodo v è ancora raggiungibile da v (e viceversa). Nell'esempio di destra (caso $w(e) = 9.5$) abbiamo che $e \in MST(G)$ ed in effetti in G' si ha che il nodo v non è più raggiungibile da u (e viceversa).

Dimostriamo ora la correttezza dell'algoritmo. Seppur molto semplice non è immediato capire il perché la procedura proposta sia corretta. Nella dimostrazione seguente faremo vedere come, sia nel caso che l'algoritmo risponda *sì* (arco appartiene all'MST) o che risponda *no* (arco non appartiene all'MST) è possibile applicare la regola del ciclo o del taglio che ci assicura che la risposta data è corretta. Siano u e v gli estremi dell'arco e.

- **caso in cui l'algoritmo risponde** $e \notin MST(G)$ **:** In questo caso si ha che in G' esiste un cammino π tra u e v (notare che $e \notin G'$) , aggiungendo l'arco e a G' si formerà un ciclo, tale ciclo è presente anche in G ed inoltre si ha che e è l'arco più pesante di tale ciclo (altrimenti il cammino π non sarebbe in G'). Quindi per la regola del ciclo è giusto affermare che $e \notin MST(G)$ □
- **caso in cui l'algoritmo risponde** $e \in MST(G)$ **:** In questo caso in G' esiste un taglio (X, Y) tale che e è l'arco più leggero che lo attraversa. Si ha che $X = \{x \in V : x$ è raggiungibile da u in $G'\}$ e $Y = V \setminus X$. Si noti che $u \in X$ e $v \in Y$ e quindi $e = (u, v)$ è un arco che attraversa il taglio.



Si può affermare che e è l'arco più leggero che attraversa tale taglio. Infatti, si consideri un qualsiasi arco (x, y) di G che attraversa il taglio (X, Y), ovvero con $x \in X$ e $y \in Y$. Allora (x, y) non può appartenere a G', perché altrimenti anche y si troverebbe in X. Quindi, dato che e è l'arco di peso minore tra quelli rimossi da G, vale $w(u, v) < w(x, y)$, e quindi e è l'arco più leggero che attraversa il taglio (X, Y). Quindi per la regola del taglio è giusto affermare che $e \in MST(G)$ □

In definitiva è stato quindi trovato un algoritmo che, dato un grafo con pesi distinti e un arco di quest'ultimo, è in grado di determinare se tale arco appartiene o meno all'unico MST senza calcolare l'MST stesso, facendo in modo che la complessità sia lineare.

Problema 7: Dato un grafo $G = (V, E, w)$ connesso e pesato sugli archi ed un insieme $U \subset V$ di nodi, progettare un algoritmo che calcoli uno spanning tree T di G di costo minimo con il vincolo che tutti i nodi in U siano foglie di T. L'algoritmo deve avere complessità $O(|E| \log |V|)$.

Soluzione: Dato un albero T, denoteremo con $cost(T)$ il costo di T, ovvero $cost(T) = \sum_{e \in E(T)} w(e)$. Al fine di progettare un algoritmo per il problema proposto osserviamo prima una importante proprietà.

Proprietà 1. Sia T^* la soluzione ottima del problema, allora $T^* \setminus U$ è un MST di $G = (V \setminus U, E, w)$. Ovvero, informalmente, dato l'albero ottimo avente i nodi in U come foglia, l'albero ottenuto rimuovendo tali nodi è un MST per il grafo ottenuto rimuovendo i nodi in U da V.

Dimostrazione. Chiamiamo con G' il grafo $G = (V \setminus U, E, w)$ e supponiamo per assurdo che $T^* \setminus U$ non sia un MST per G' . Allora esiste un MST T' per il grafo G' tale che $cost(T') < cost(T^* \setminus U)$. Possiamo ora "aggiungere" tutti i nodi in U all'albero T' collegandoli agli stessi nodi a cui erano collegati in T^* (quindi utilizzando anche gli stessi archi **con gli stessi pesi**). È facile osservare che l'albero $T' \cup U$ ha tutti i nodi in U come foglie, ed inoltre $cost(T' \cup U) < cost(T^*)$. Ma l'albero T^* era l'albero ottimo quindi avendo raggiunto un assurdo deve necessariamente essere che $T^* \setminus U$ è un MST per $G = (V \setminus U, E, w)$. □

Nella figura di seguito si può vedere in un esempio come in effetti l'albero $T^* \setminus U$ corrisponda all'MST calcolato sui soli nodi in $V \setminus U$.

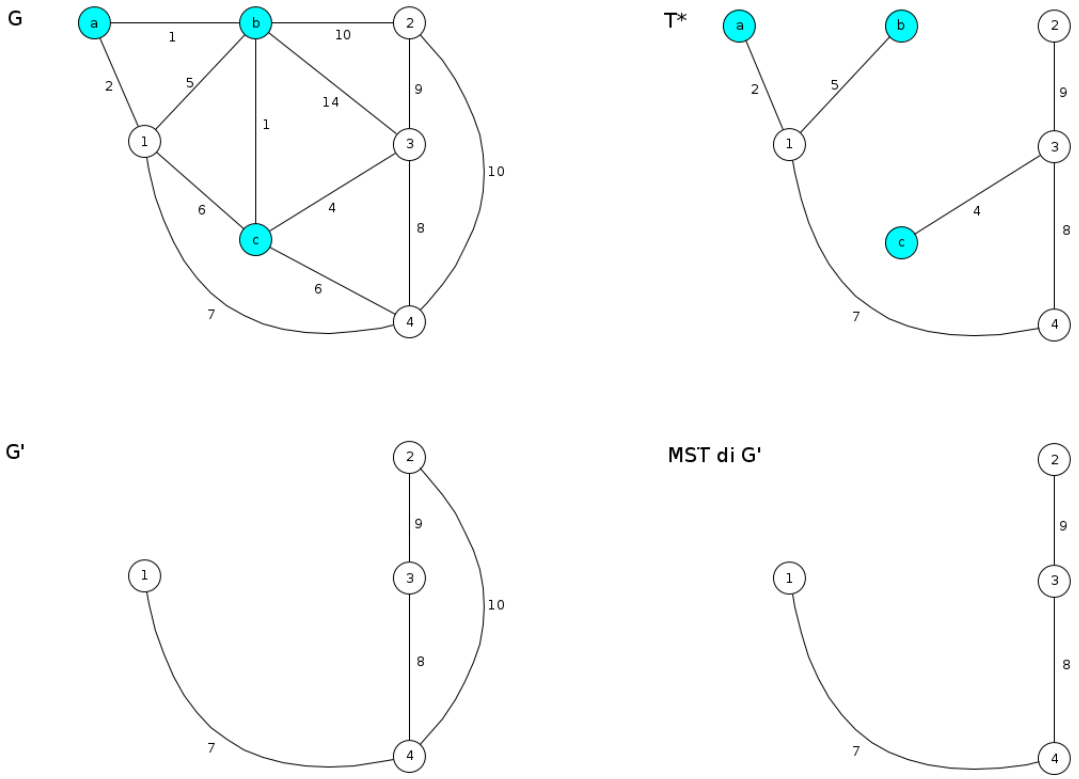


Figura 2: in nodi in U sono colorati in azzurro

Una volta compreso che l'MST T del grafo $G \setminus U$ è sottografo dell'albero richiesto, per trovare la soluzione del nostro problema non ci resta che trovare un metodo per aggiungere i nodi di U come foglie a T a costo totale

minimo. E' facilmente intuibile che l'approccio "vincente" è quello di collegare ogni nodo in U ad un nodo in T utilizzando l'arco di peso minore tra quelli disponibili.

E' semplice verificare che tale approccio porta alla soluzione cercata in quanto i nodi in U saranno certamente delle foglie, ed inoltre l'albero ottenuto sarà ottimo (se non si scegliesse l'arco di peso minore banalmente l'albero ottenuto avrebbe un costo più alto; si può dimostrare ciò formalmente usando argomentazioni di tipo "cut&paste" usate nella prova della Proprietà 1).

Quindi in virtù della proprietà 1 e della correttezza della procedura appena descritta per trovare una soluzione al nostro problema possiamo usare il seguente algoritmo:

```
   Data: G = (V, E, w), U ⊂ V
1  begin
2      T ⟵ MST di G \ U.
3      forall u ∈ U do
4          sia (u, v) l'arco più leggero in E con v ∉ U
5          T ⟵ T ∪ {(u, v)}
6      return T
```

La correttezza dell'algoritmo descritto è conseguenza diretta della correttezza della proprietà 1 e dell'approccio con cui aggiungiamo i nodi in U all'albero.

Analizziamo ora la complessità dell'algoritmo proposto. Per calcolare un MST T di $G \setminus U$ in tempo $O(|E| \log |V|)$ possiamo utilizzare l'algoritmo di Kruskal su $G \setminus U$. Per aggiungere i nodi in U a T possiamo scorrere l'insieme degli archi in ordine crescente di peso (possiamo ordinare gli archi in tempo $O(|E| \log |V|)$) ed aggiungere un arco (u, v) a T se e soltanto se $u \in U$ e $v \in V \setminus U$ (o viceversa) e u (o v nel viceversa) non è già stato aggiunto a T.

In definitiva la complessità totale dell'algoritmo è $O(|E| \log(|V|))$ (come richiesto).

Problema 8: Dato un grafo diretto aciclico (DAG) $G = (V, E)$ e due nodi $s, t \in V$ progettare un algoritmo lineare (in $|V|$ e $|E|$) che calcoli il numero di cammini distinti in G da s a t. Suggerimento: si usi la programmazione dinamica.

Soluzione: Dato un grafo diretto $G = (V, E)$ e due nodi $s, t \in V$, due cammini uscenti da s ed entranti in t sono distinti se differiscono per almeno un arco.

L'idea è quella di calcolare inizialmente un ordinamento topologico del grafo in input, ottenendo cosi un ordine di "attraversamento". Trovato l'ordinamento possiamo applicare la programmazione dinamica per trovare il numero di cammini distinti tra s e t.

In particolare rivolveremo $O(|V|)$ sottoproblemi della seguente forma: calcolare il numero di cammini distinti da u a t, dove u è un generico nodo. Una volta risolti tutti questi sottoproblemi abbiamo risolto il nostro problema, in quanto la soluzione che cerchiamo è quella del sottoproblema relativo a s.

Ma quale è la relazione fra diversi sottoproblemi? Come possiamo comporre le soluzioni di certi sottoproblemi per trovare la soluzione al generico sottoproblema relativo al (generico) nodo u? Per rispondere a tale domanda bisogna osservare che dato un nodo u, il numero di cammini distinti da u a t è decomponibile come la somma del numero di cammini distinti che partono dai successori di u e terminano in t.

Nella figura seguente è rappresentato quanto appena detto:

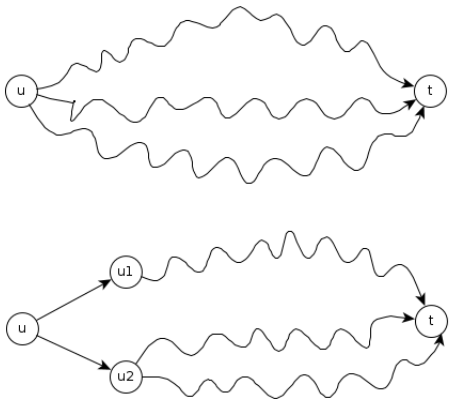


Figura 3: per calcolare il numero di cammini distinti da u a t possiamo ricondurci a calcolare il numero di cammini distinti dai successori di u a t

Più formalmente, sia OPT[u] il numero di cammini distinti da u a t. Valgono le seguenti relazioni fra sottoproblemi:

- casi base:
  - OPT[u] $= 0$ se u si trova dopo t nell'ordinamento topologico (non ci sono cammini da u a t);
  - OPT[u] $= 1$ se $u = t$ (c'è un unico cammino degenere da t a t che non ha archi);
  - OPT[u] $= 0$ se u non ha archi uscenti e $v \neq t$ (non ci sono cammini da u a t).
- caso generico (quando non si applica nessun caso base):
  - OPT[u] $= \sum_{v:(u,v)\in E}$ OPT[v] (il numero di cammini distinti che da u vanno a t è la somma del numero di cammini distinti che usano i diversi archi che escono da u).

Rimane soltanto da stabilire in che ordine andare a risolvere i sottoproblemi. Dalle relazioni risulta evidente che è sufficiente considerare i nodi u (e quindi i sottoproblemi) in ordine topologico inverso. Tutti i nodi u che seguono t nell'ordinamento topologico sono casi basi, così come è un caso base $u = t$. Poi si andranno a risolvere i sottoproblemi relativi a nodi che precedono t nell'ordinamento, fino a raggiungere s.

Vediamo ora un possibile pseudocodice dell'algoritmo descritto:

```
   Data: G = (V, E, w), U ⊂ V
 1 begin
 2     ord ⟵ ordinamentoTopologico(G)
 3     forall u in ordine topologico inverso do
 4         if ord(u) > ord(t) then
 5             OPT[u] ⟵ 0
 6         if u = t then
 7             OPT[u] ⟵ 1
 8         else
 9             OPT[u] ⟵ 0
10             forall (u, v) ∈ E do
11                 OPT[u] ⟵ OPT[u] + OPT[v]
12     return OPT[s]
```

Per quanto riguarda la complessità, assumendo (ragionevolmente) che il grafo G ci è dato con una rappresentazione a liste di adiacenza, abbiamo che l'ordinamento topologico può essere calcolato in tempo $O(|V| + |E|)$. Per quanto riguarda la fase dell'algoritmo in cui il vettore OPT viene popolato, per ogni nodo stiamo scorrendo la sua lista di adiacenza quindi si che il tempo speso è proporzionale alla somma dei gradi uscenti di tutti i nodi, e quindi anche questa fase cosa $O(|V| + |E|)$. Quindi in definitiva l'algoritmo ha complessità totale $O(|V| + |E|)$ (come richiesto).