# Esercitazione 2

**Problema 4:** Dato un grafo $G = (V, E)$ con pesi positivi sugli archi ed un insieme di $k$ centri $C = \{c_1, c_2, \cdots c_k\} \subseteq V$, si richiede di partizionare l'insieme $V$ in $k$ insiemi $V_1, V_2, \cdots V_k$ in modo che tutti i vertici in un insieme $V_i$ siano più vicini al vertice $c_i$ che ad ognuno degli altri centri in $C$. Formalmente, se $d(u, v)$ indica la distanza tra i vertici $u$ e $v$ in $G$, deve valere:

$$d(u, c_i) \leq d(u, c_j) \forall i, j = 1, \cdots, k \; \forall u \in V_i$$

Progettare un algoritmo che risolva tale problema in tempo $O(|E| + |V| \log |V|)$.

**Soluzione** Vogliamo trovare una partizione dei nodi del grafo in $k$ sottoinsiemi $V_1, \cdots V_k$ in modo che, i nodi nel generico sottoinsieme $V_i$, si trovino più vicini al centro $c_i$ che agli altri centri.

Una prima idea è quella di applicare l'algoritmo di Dijkstra $k$ volte, usando ogni volta come sorgente uno dei centri in $C$. Una volta terminata tale procedura per ogni nodo in $V$ avremo la distanza verso ognuno dei $k$ centri, quindi sarà facile a questo punto assegnare ogni nodo al sottoinsieme corrispondente al centro che minimizza tale distanza. Vediamo ora un possibile pseudocodice dell'algoritmo appena descritto:

```
   Data: G = (V, E)
   Result: V_1, ··· , V_k
1  begin
2      V_i ⟵ ∅ ∀i = 1, ··· , k
3      forall c_i ∈ C do
4          calcola d(c_i, v) ∀v ∈ V
5      forall v ∈ V do
6          i ⟵ argmin_{j=1,···,k} d(c_j, v)
7          V_i ⟵ V_i ∪ {v}
8      return V_1, ··· , V_k
```

Argomentiamo ora sulla complessità dell'algoritmo proposto: per quanto riguarda il calcolo delle distanze (righe 3-4) è richiesta un'esecuzione dell'algoritmo di Dijkstra per ogni centro, quindi si ha complessità $O(k(|E| + |V| \log |V|))$. Mentre per il calcolo della partizione (righe 5-7) si ha complessità $O(k|V|)$. Il nostro algoritmo in definitiva ha complessità $O(k|E| + k|V| \log |V|)$ e quindi non in linea con le richieste sull'efficienza del problema (si noti che $k$ potrebbe essere anche $\Theta(|V|)$, quando i centri sono una frazione costante dei nodi).

Dobbiamo quindi trovare un modo più furbo per calcolare la partizione.

Un'ulteriore idea è quella di aggiungere un nodo fittizio al grafo (chiamiamolo $s$) e collegarlo a ogni centro con un arco di peso zero. Una volta ottenuto tale grafo, per trovare la partizione richiesta sarà sufficiente eseguire l'algoritmo di Dijkstra usando come sorgente il nodo $s$. In particolare nell'albero ottenuto, al livello uno (quello in cui si trovano i figli della radice) ci saranno tutti e soli i nodi centro (in quanto sono gli unici collegati ad $s$). Considerato un generico sottoalbero $T_i$ radicato nel nodo centro $c_i$, avremo che tutti e soli i nodi in $T_i$ devono appartenere al sottoinsieme $V_i$ di $V$. La cosa non è banale e va dimostrata. Ma prima diamo un possibile pseudocodice dell'algoritmo proposto:

**Data:** $G = (V, E)$
**Result:** $V_1, \cdots, V_k$

**begin**
  $V_i \longleftarrow \emptyset \; \forall i = 1, \cdots, k$
  $V \longleftarrow V \cup \{s\}$
  **forall** $c \in C$ **do**
    $E \longleftarrow E \cup \{(s, c)\}$
    $p(s, c) \longleftarrow 0$
  $T \longleftarrow Dijkstra(G, s)$
  siano $T_1, \cdots T_k$ i sottoalberi di $T$ radicati nei nodi $c_1, \cdots, c_k$
  **for** $i = 1$ *to* $k$ **do**
    $V_i \longleftarrow$ insieme dei nodi in $T_i$
  return $V_1 \cdots V_k$

Vediamo ora un esempio di come l'algoritmo proposto opera:

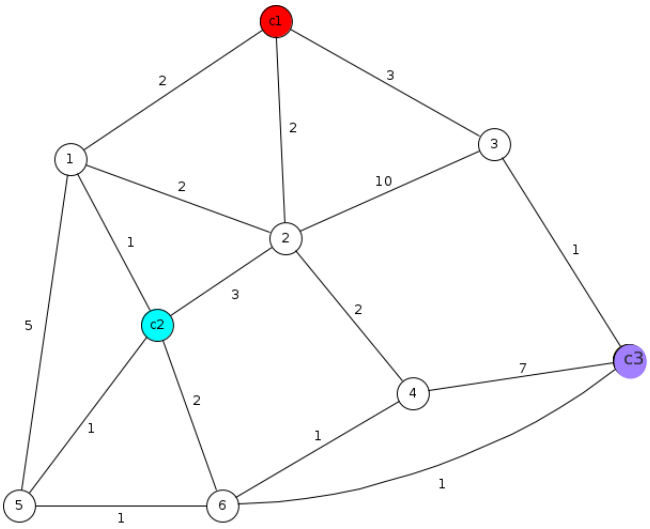



Figura 1: grafo di partenza (i centri sono evidenziati con colori differenti)

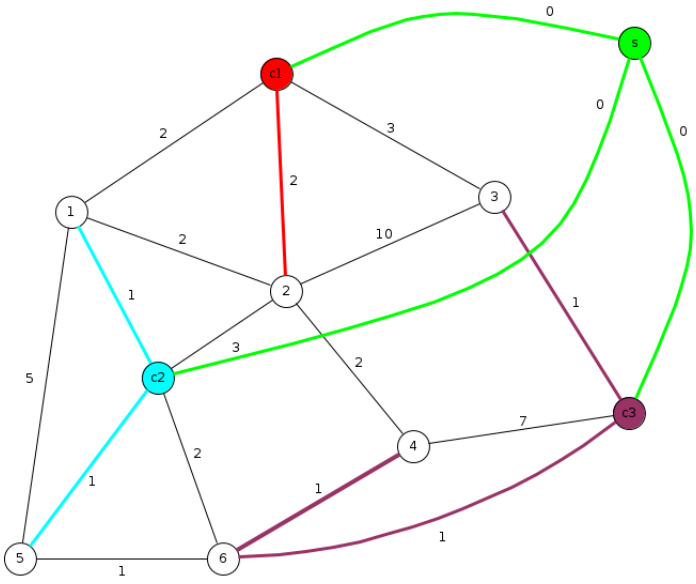



Figura 2: risultato dell'algoritmo proposto

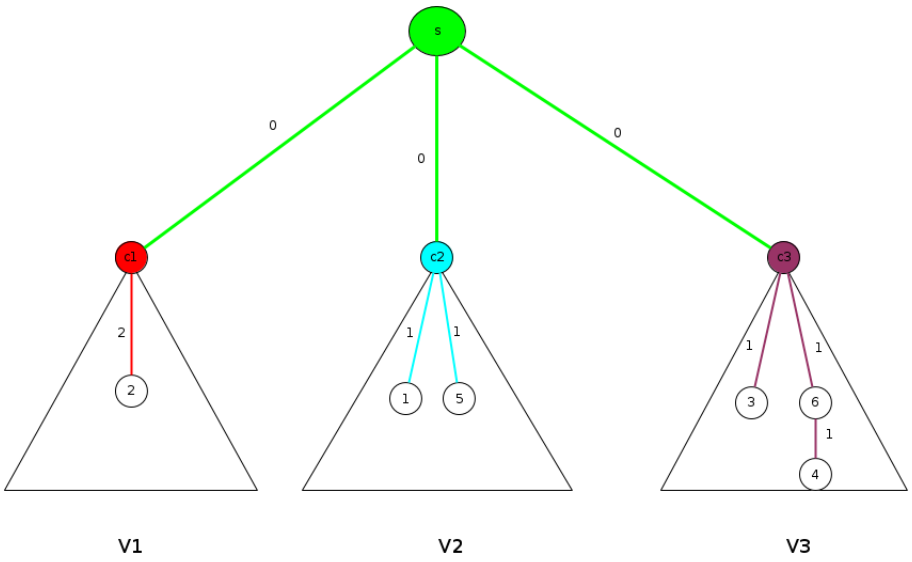


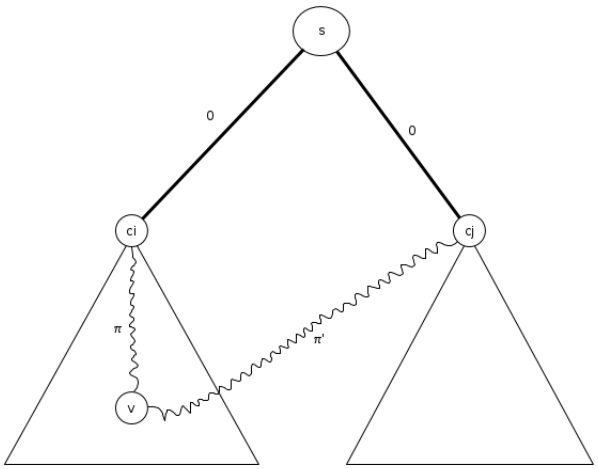

Figura 3: i sottoalberi evidenziati costituiscono la partizione richiesta

Andiamo ora ad analizzare la complessità dell'algoritmo proposto.

Chiamiamo $G' = (V', E')$ il grafo ottenuto aggiungendo il nodo $s$ e gli archi a lui incidenti, si ha che: $|V'| = |V| + 1$ e $|E'| = |E| + k$, restringendoci ai casi in cui $G$ è connesso e osservando che $k \leq n$ possiamo concludere che $|V'| = O(|V|)$ e $|E'| = O(|E|)$. Quindi l'applicazione dell'algoritmo di Dijkstra su $G'$ ha complessità $O(|E| + |V| \log |V|)$, mentre la costruzione degli insiemi $V_i$ (righe 8-10) può essere effettuata con una visita dell'albero $T$ e quindi in tempo $O(|V|)$. In definitiva l'algoritmo proposto ha complessità $O(|E| + |V| \log |V|)$ (come richiesto).

Dimostriamo infine la correttezza dell'algoritmo facendo vedere che, dato un nodo $v \in T(c_i)$ (indichiamo con $T(c_i)$ il sottoalbero di $T$ radicato in $c_i$) il centro $c_i$ è effettivamente il centro più vicino al nodo $v$.

Supponiamo per assurdo che esista un centro $c_j \neq c_i$ tale che $d(c_j, v) < d(c_i, v)$, indichiamo con $\pi'$ il cammino minimo in $G$ tra $c_j$ e $v$. Inoltre, indichiamo con $\pi$ il cammino in $T$ tra $c_i$ e $v$. Si noti che $\pi$ è un cammino in $G$ e quindi $d(c_i, v) \leq w(\pi)$. Vogliamo argomentare sul fatto che $\pi'$ non può essere strettamente più corto di $\pi$ e quindi non può essere $d(c_j, v) < d(c_i, v)$, da cui l'assurdo.




Infatti, abbiamo che, se $w(\pi') < w(\pi)$ allora $w(\pi' \cup \{(s, c_j)\}) < w(\pi \cup \{(s, c_i)\})$ che ci porta ad un assurdo in quanto $\pi \cup \{(s, c_i)\}$ è un cammino minimo in $G'$ da $s$ a $v$ (per la correttezza dell'algoritmo di Dijkstra) e quindi non può essere più lungo del cammino $\pi' \cup \{(s, c_j)\}$ (che è ancora un cammino in $G'$).

Avendo raggiunto un assurdo possiamo affermare che, se un generico nodo $v$ viene collocato in un certo sottoalbero $T(c_i)$ si ha che il centro $c_i$ è il più vicino al nodo $v$.

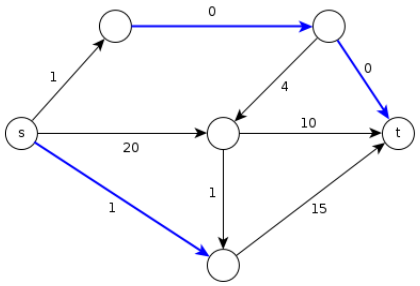
**Problema 5:** Sia $G = (V, E, w)$ un grafo orientato con pesi positivi sugli archi in cui alcuni archi sono speciali e sono chiamati archi *blu*. Progettare un algoritmo che, dato $G$, due nodi $s, t \in V$ e un intero $k$, calcoli un cammino di costo minimo da $s$ a $t$ che usa *al più* $k$ archi blu.

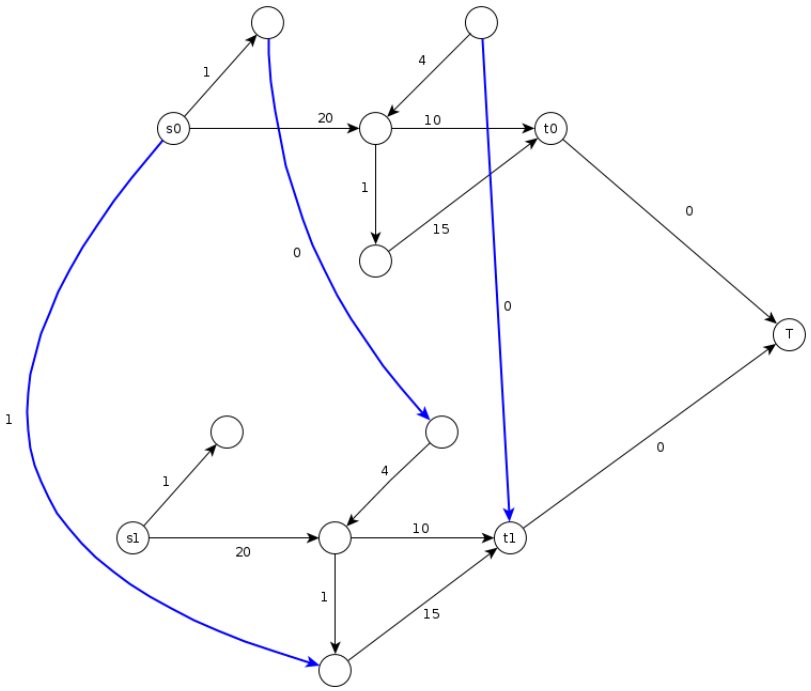
**Soluzione** L'idea è quella di costruire un grafo ausiliario $G'$ che ci vincoli a utilizzare al più $k$ archi blu. In particolare dato il grafo $G$, $G'$ sarà composto da $k+1$ "copie" di $G$, ognuna delle quali è essenzialmente definita come $G$ senza però gli archi blu. Le copie sono organizzate in "livelli", ogni livello indica il numero di archi blu utilizzati. Le copie ai livelli $0, \cdots, k-1$ sono collegate nel seguente modo: per ogni arco blu $(u, v)$ uscente da $u$, si rimpiazza tale arco con l'arco $(u, v')$ dove $v'$ è la replica del nodo $v$ appartenente al livello successivo. Nella copia al livello $k$ non ci sono archi blu. Infine tutti i nodi $t$ di ogni copia sono collegati con un arco diretto di peso 0 ad un unico nuovo nodo destinazione chiamato $T$. L'idea è questa: in ogni livello ci si può spostare liberamente senza usare però archi blu, se si vuole cambiare livello e andare al successivo, invece, si è costretti a usare esattamente un arco blu. Quindi se si considera un generico cammino da $s$ (nella copia di livello 0) a un generico nodo al livello $i$, questo cammino ha esattamente $i$ archi blu.

Più formalmente, $G'$ è definito nel seguente modo:

- **nodi:** $\forall v \in V$ ho $k+1$ copie $v_0, v_1, \cdots, v_k$, è presente inoltre un unico nodo destinazione $T$
- **archi:**
  - $\forall$ arco $(u, v)$ non blu ho gli archi $(u_i, v_i)$, $i = 0, \cdots, k$ con $w(u_i, v_i) = w(u, v)$;
  - $\forall$ arco $(u, v)$ blu ho gli archi $(u_i, v_{i+1})$, $i = 0, \cdots, k-1$ con $w(u_i, v_{i+1}) = w(u, v)$;
  - ho gli archi $(t_i, T)$ con $w(t_1, T) = 0$ per $i = 0, \cdots, k$.

Al fine di chiarire come $G'$ è costituito mostriamo un esempio in figura: l'istanza $G$ con $k = 1$ e il corrispondente grafo $G'$.



Una volta costruito il grafo $G'$ per trovare la soluzione al nostro problema sarà sufficiente trovare il cammino minimo $\pi'$ in $G'$ tra il nodo $s_0$ ed il nodo $T$ e poi riconvertilo in un cammino $\pi$ in $G$. In particolare sia $\pi'$ il cammino trovato dall'algoritmo su $G'$, il cammino $\pi$ cercato su $G$ è definito nel seguente modo. Prima rimuoviamo da $\pi'$ l'arco $(t_j, T)$, dove $t_j$ è la replica di $t$ da cui abbiamo raggiunto il target $T$. Poi, un generico arco in $\pi'$ del tipo $(u_i, v_i)$ (ovvero un arco che non ci fa cambiare livello) è sostituito con l'arco $(u, v)$, e un generico arco in $\pi'$ della forma $(u_i, v_i)$ (ovvero un arco che ci fa cambiare livello) viene sostituito con l'arco blu $(u, v)$.

È facile osservare che il cammino $\pi$ cosi prodotto contiene al più $k$ archi blu, visto che ogni volta che ne viene utilizzato uno si avanza al livello successivo, ed una volta giunti all'ultimo livello non ci sono più archi blu da utilizzare. Ora una domanda più intrigante è questa: è $\pi$ il cammino di costo minimo fra tutti quelli che usano al più $k$ archi blu? La risposta a questa domanda è sì e deriva dalla seguente proprietà (la cui semplice dimostrazione è lasciata come esercizio allo studente).

*Proprietà:* esiste un cammino in $G$ da $s$ a $t$ che ha al più $k$ archi blu di lunghezza $W$ se e soltanto se esiste un cammino in $G'$ da $s_0$ a $T$ di lunghezza $W$.

Andiamo ora ad analizzare la complessità dell'algoritmo appena proposto: si ha che $|V'| = \Theta(k|V|)$ mentre $|E'| = \Theta(k|E|)$ da ciò la complessità di costruzione di $G'$ risulta essere $O(k|V| + k|E|)$. L'applicazione dell'algoritmo di Dijkstra su $G'$ avrà complessità $O(k|E| + k|V| \log |V|)$ quindi l'algoritmo proposto ha complessità totale $O(k|E| + k|V| \log |V|)$.