

Problema del cammino minimo

Un viaggiatore vuole trovare la via più corta per andare da una città ad un'altra.

Possiamo rappresentare ogni città con un nodo e ogni collegamento tra due città con un arco etichettato con la distanza che separa le due città.

Tra tutti i cammini che collegano le due città interessate, scegliamo quello più corto.

Problema del cammino minimo

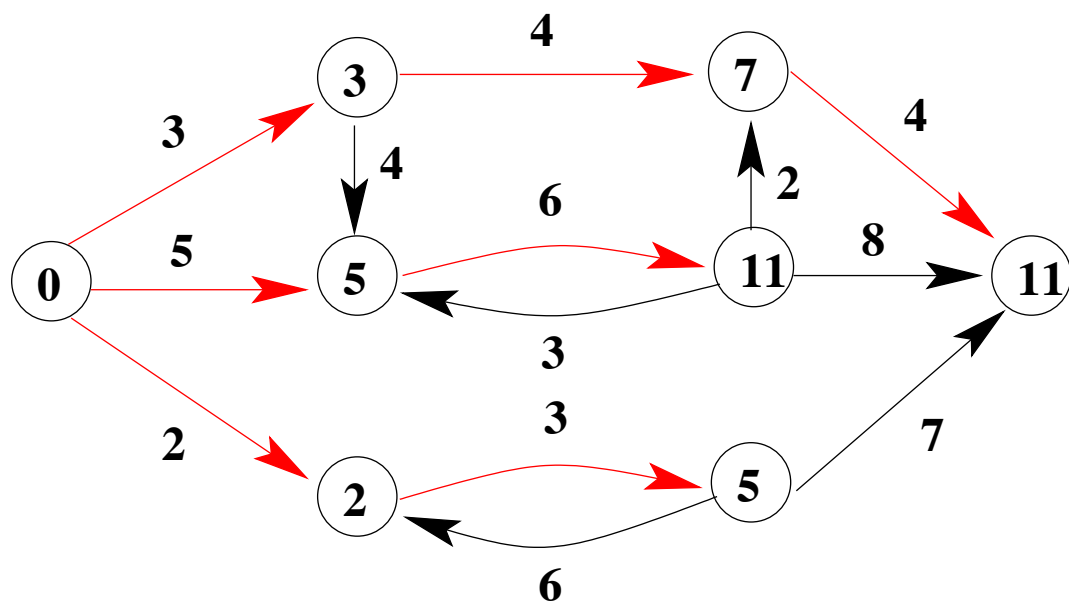
Un grafo diretto **pesato** è una tripla $G = (V, E, w)$ ove (V, E) è un grafo diretto e $w : E \rightarrow \mathbb{R}$ è una funzione peso che associa ad ogni arco un **peso** (o **costo**) reale.

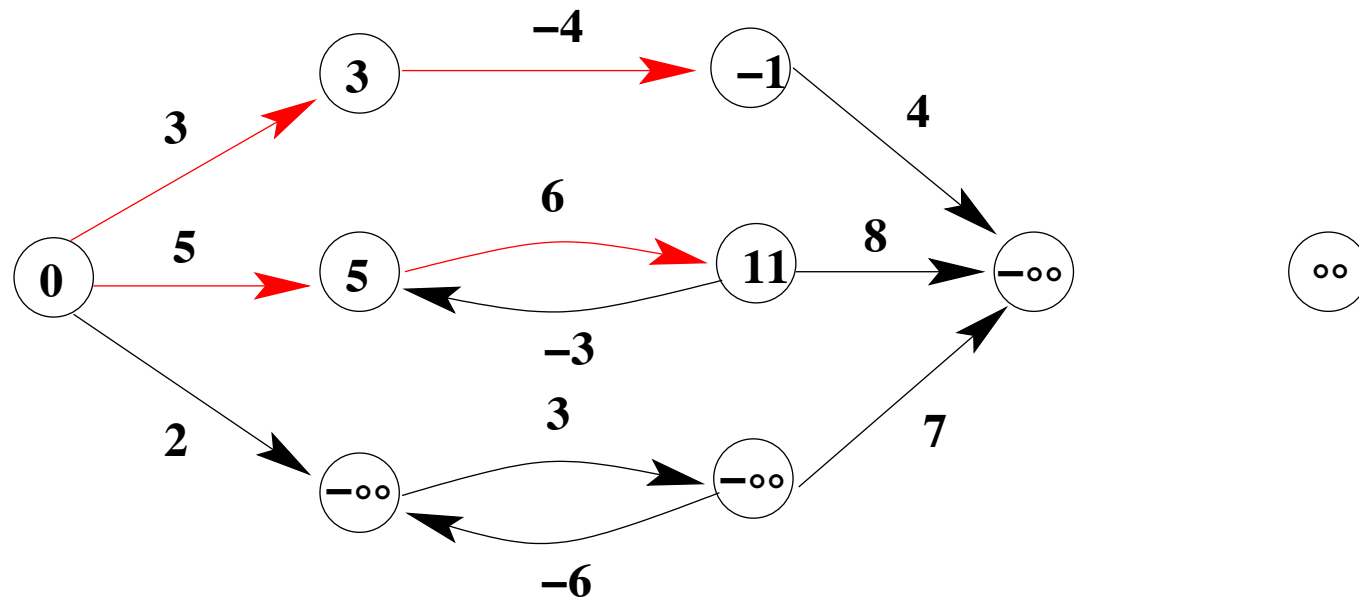
- Il **peso** di un cammino è la somma dei pesi dei propri archi;
- la **lunghezza** di un cammino è il numero di archi del cammino; collega i due nodi.

Dati due nodi u e v , un **cammino minimo** tra due nodi è, se esiste, un cammino di **peso minimo** che li collega. La **distanza** tra due nodi è il peso del cammino minimo che collega i due nodi.

Varianti del problema

- **Cammino minimo tra due nodi:** dati due nodi u e v , trovare un cammino minimo da u e v ;
- **Cammino minimo con una sorgente:** dato un nodo sorgente s , trovare, per ogni nodo del grafo v , un cammino minimo da s a v ;
- **Cammino minimo con una destinazione:** dato un nodo destinazione d , trovare, per ogni nodo del grafo v , un cammino minimo da v a d ;
- **Cammino minimo per tutte le coppie di nodi:** per ogni coppia di nodi u, v del grafo trovare un cammino minimo da u a v .





Proprietà del cammino minimo

La **proprietà della sottostruttura ottimale** afferma che un cammino minimo contiene altri cammini minimi al suo interno.

Teorema Sia $p = \langle v_0, v_1, \dots, v_k \rangle$ un cammino minimo da v_0 a v_k e $p_{i,j}$ il sottocammino di p da v_i a v_j . Allora $p_{i,j}$ è un cammino minimo da v_i a v_j .

Osservazione Un cammino minimo è un cammino semplice, cioè non contiene cicli.

Rappresentazione del cammino minimo

Sia $G = (V, E, w)$ un grafo pesato e s una sorgente di G . Per ogni nodo v raggiungibile dalla sorgente s , rappresentiamo un solo cammino minimo da s a v .

Scriviamo nella variabile $\pi[v]$ il **predecessore** di v lungo il cammino minimo rappresentato dalla sorgente s a v . $\pi[v]$ è NIL se $v = s$ oppure se non esiste alcun cammino minimo da s a v .

Inoltre scriviamo nella variabile $d[v]$ una **stima** (per eccesso) del peso del cammino minimo che va da s a v .

Rappresentazione del cammino minimo

Il sottografo $G_\pi = (V_\pi, E_\pi)$ tale che:

$$V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$$

e

$$E_\pi = \{(\pi[v], v) \in E : \pi[v] \neq \text{NIL}\}$$

è detto **albero dei cammini minimi** con radice s .

Per ogni nodo $v \in V_\pi$, l'unico cammino in G_π da s a v corrisponde ad un cammino minimo in G da s a v .

Inizializza

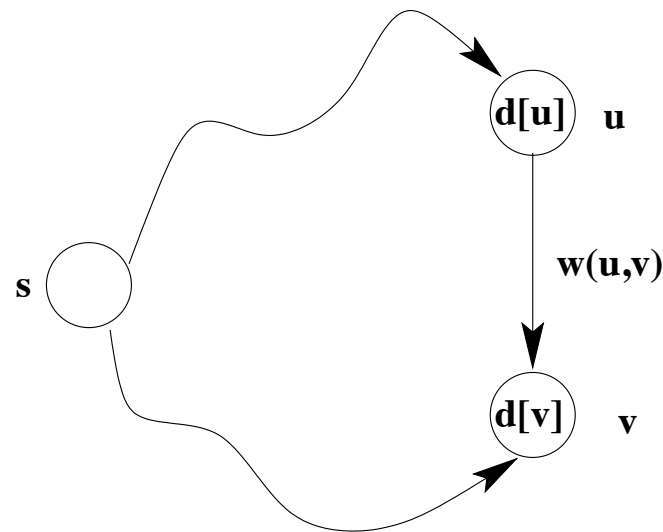
Initialize(G, s)

```
1: for all  $v \in V[G]$  do  
2:    $d[v] \leftarrow \infty$   
3:    $\pi[v] \leftarrow \text{NIL}$   
4: end for  
5:  $d[s] \leftarrow 0$ 
```

Il costo è $\Theta(n)$, con $n = |V|$.

Rilassa**Relax(G, u, v)**

- 1: **if** $d[u] + w(u, v) < d[v]$ **then**
- 2: $d[v] \leftarrow d[u] + w(u, v)$
- 3: $\pi[v] \leftarrow u$
- 4: **end if**



Algoritmo di Bellman-Ford

L'algoritmo risolve il problema del cammino minimo con una sorgente e pesi reali (possibilmente negativi).

L'algoritmo ritorna FALSE se esiste un ciclo di peso negativo raggiungibile dalla sorgente. In questo caso, non tutti i cammini minimi dalla sorgente esistono.

Altrimenti, l'algoritmo ritorna TRUE e, per ogni nodo v , calcola un cammino minimo dalla sorgente s a v e il suo peso.

Algoritmo di Bellman-Ford

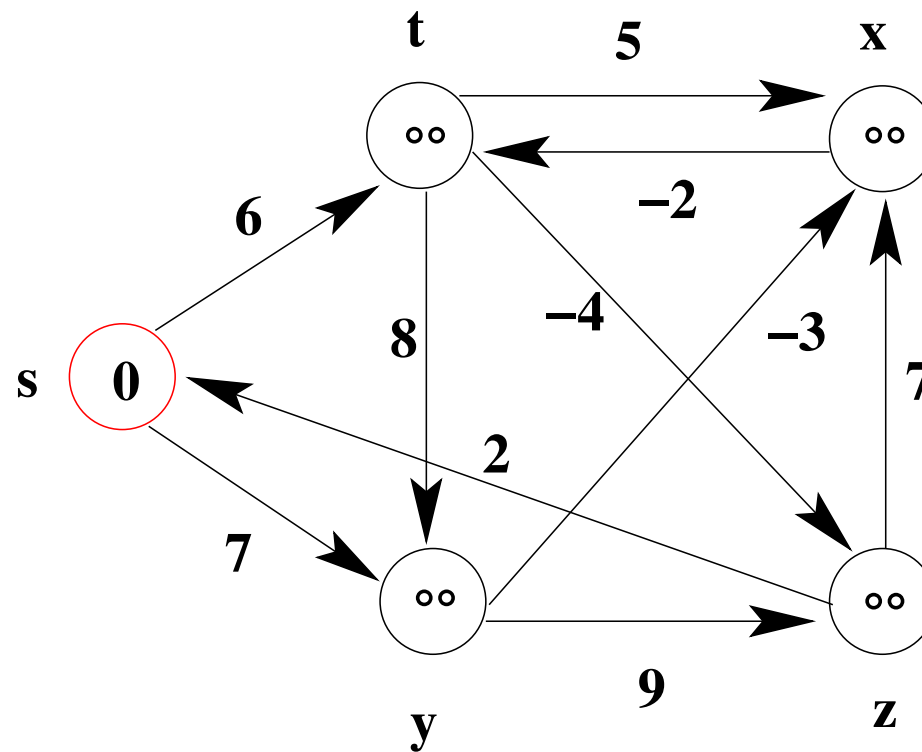
L'idea dell'algoritmo è la seguente:

1. inizializzo i predecessori e le stime per ogni nodo;
2. ripeto, per $n - 1$ volte, il rilassamento di tutti gli archi;
3. se esiste qualche stima che può essere ulteriormente migliorata, allora il grafo contiene un ciclo di peso negativo raggiungibile da s , e quindi ritorno FALSE; altrimenti ritorno TRUE.

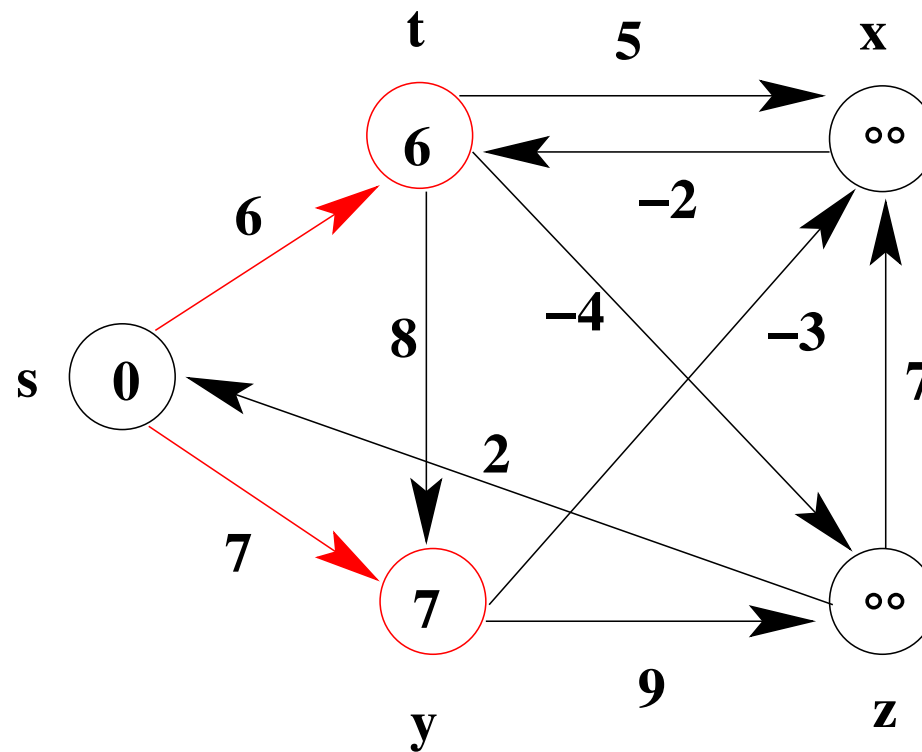
Algoritmo di Bellman-Ford

Bellman-Ford(G, s)

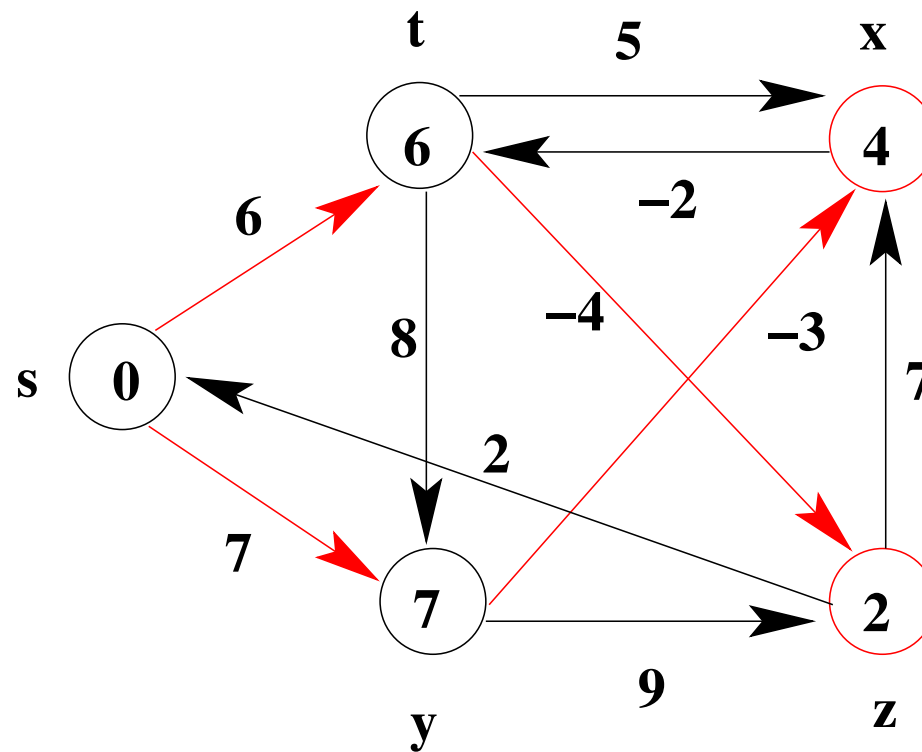
```
1:  $n \leftarrow |V[G]|$ 
2: Initialize( $G, s$ )
3: for  $i \leftarrow 1$  to  $n - 1$  do
4:   for all  $(u, v) \in E[G]$  do
5:     Relax( $G, u, v$ )
6:   end for
7: end for
8: for all  $(u, v) \in E[G]$  do
9:   if  $d[u] + w(u, v) < d[v]$  then
10:    return FALSE
11:  end if
12: end for
13: return TRUE
```



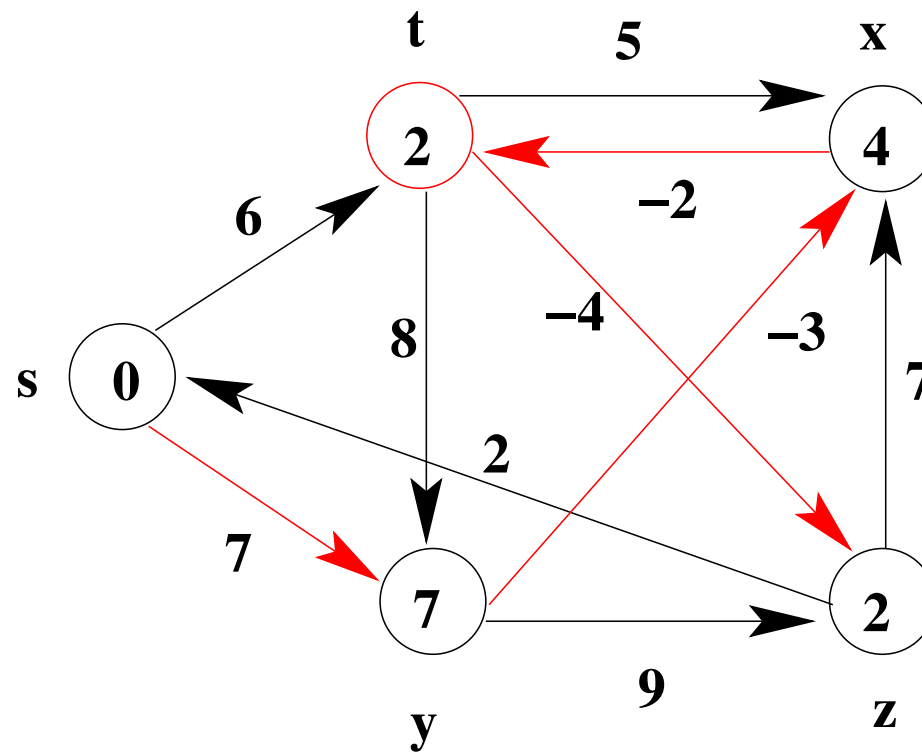
$$E[G] = \{(t,x),(t,y),(t,z),(x,t),(y,x),(z,x),(z,s),(s,t),(s,y)\}$$



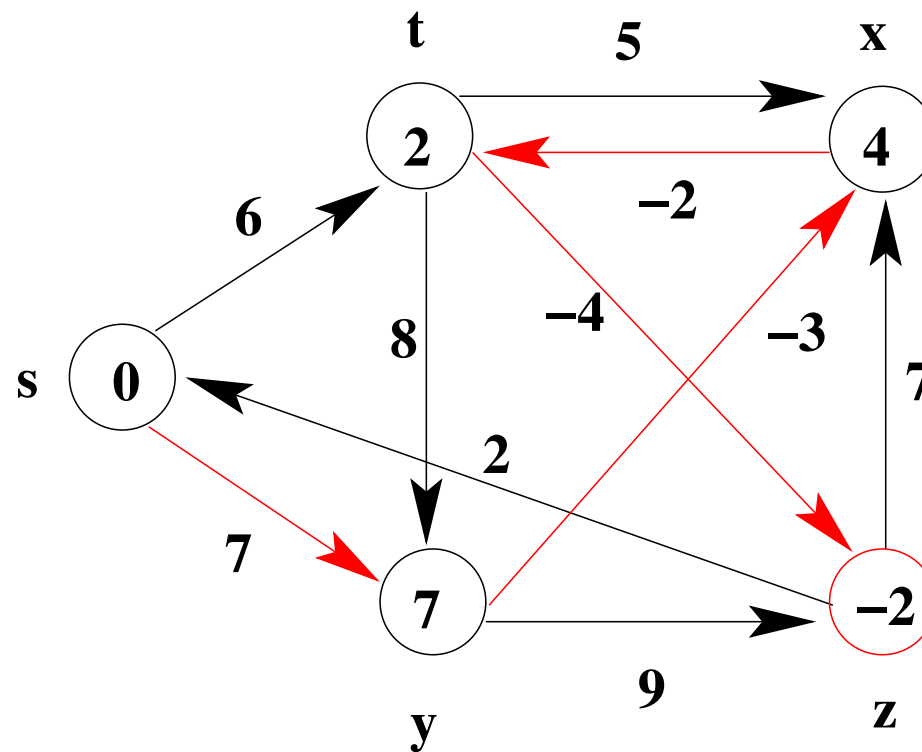
$$E[G] = \{(t,x),(t,y),(t,z),(x,t),(y,x),(z,x),(z,s),(s,t),(s,y)\}$$



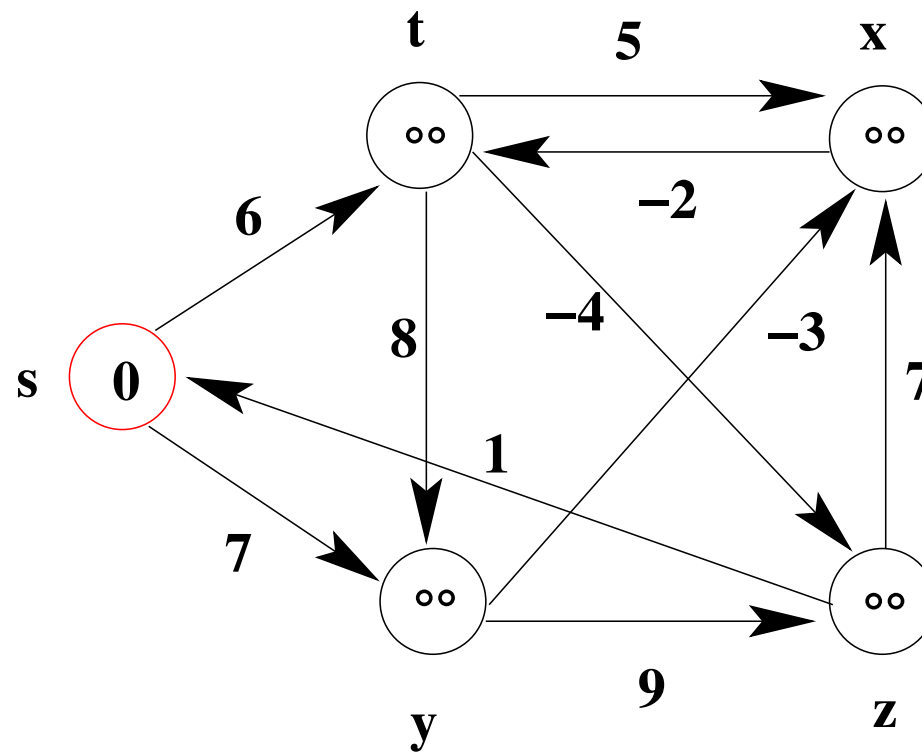
$$E[G] = \{(t,x),(t,y),(t,z),(x,t),(y,x),(z,x),(z,s),(s,t),(s,y)\}$$



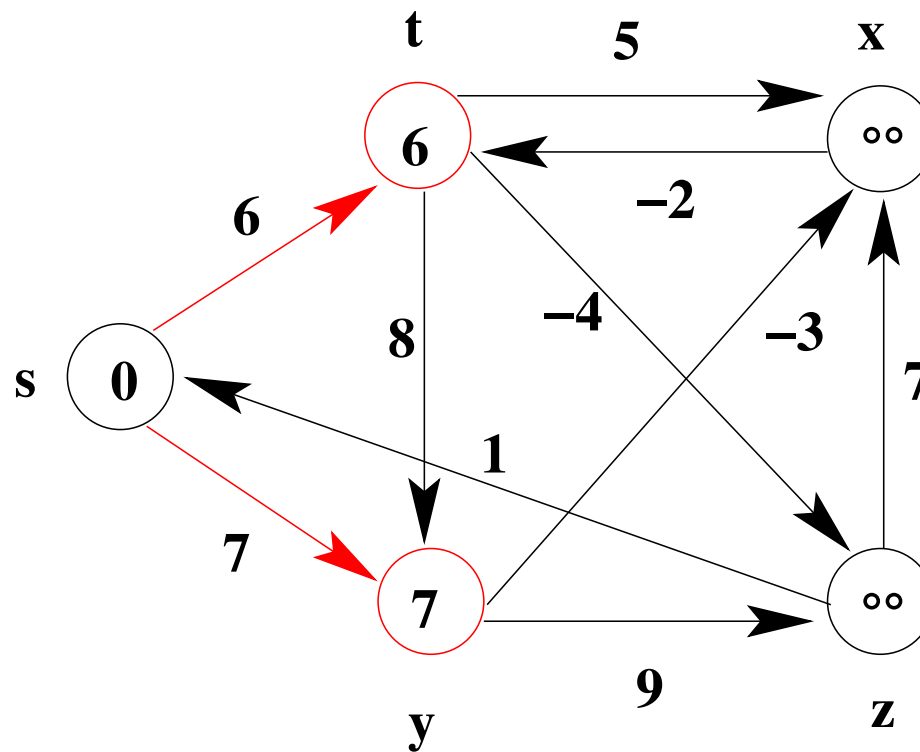
$$E[G] = \{(t,x),(t,y),(t,z),(x,t),(y,x),(z,x),(z,s),(s,t),(s,y)\}$$



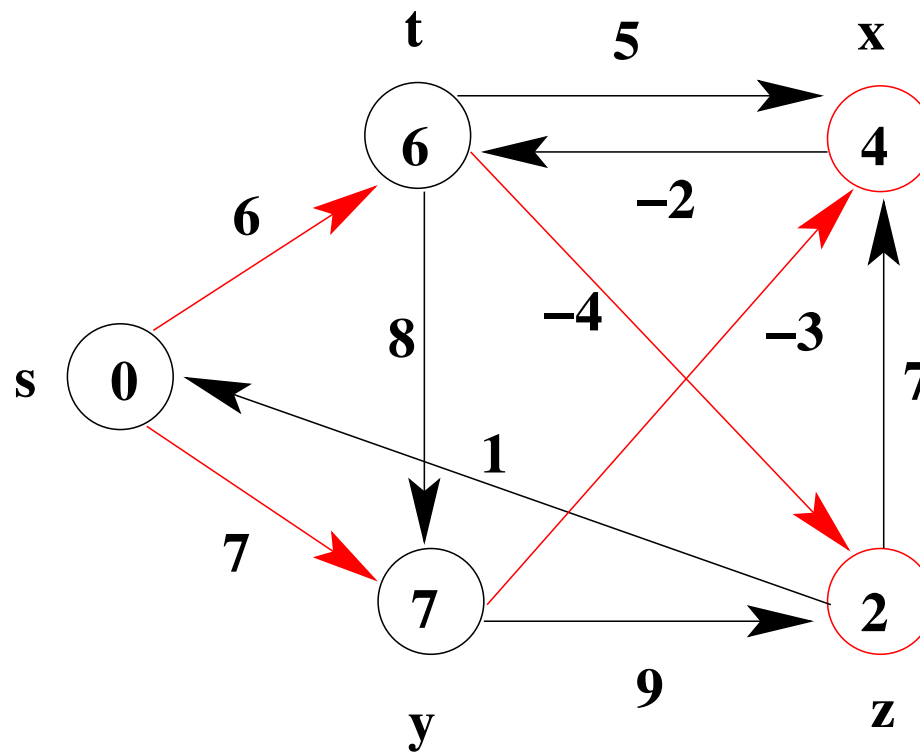
$$E[G] = \{(t,x),(t,y),(t,z),(x,t),(y,x),(z,x),(z,s),(s,t),(s,y)\}$$



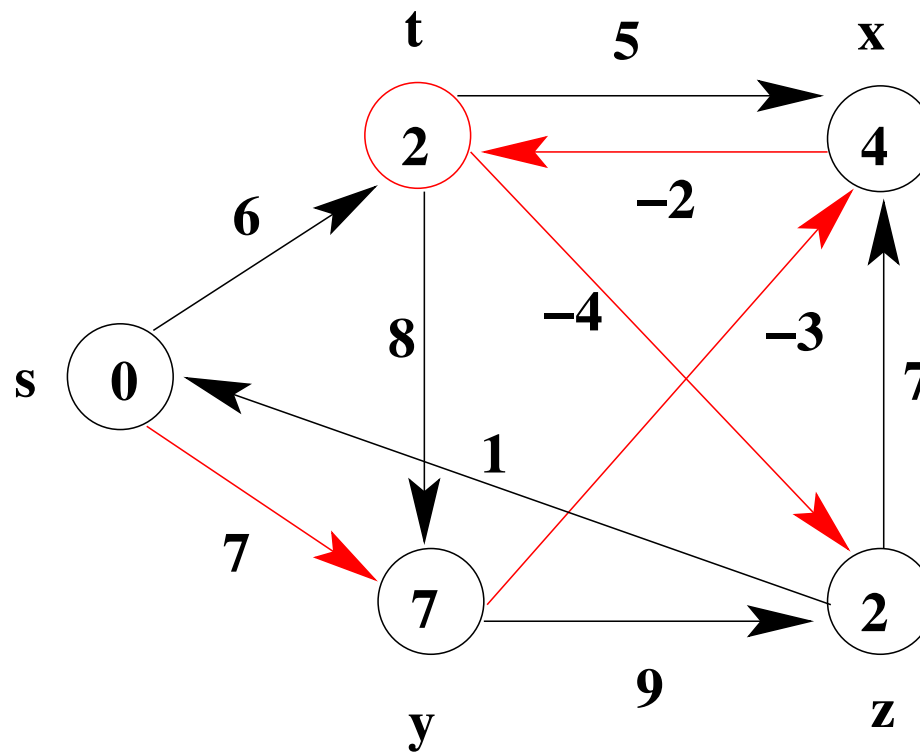
$$E[G] = \{(t,x),(t,y),(t,z),(x,t),(y,x),(z,x),(z,s),(s,t),(s,y)\}$$



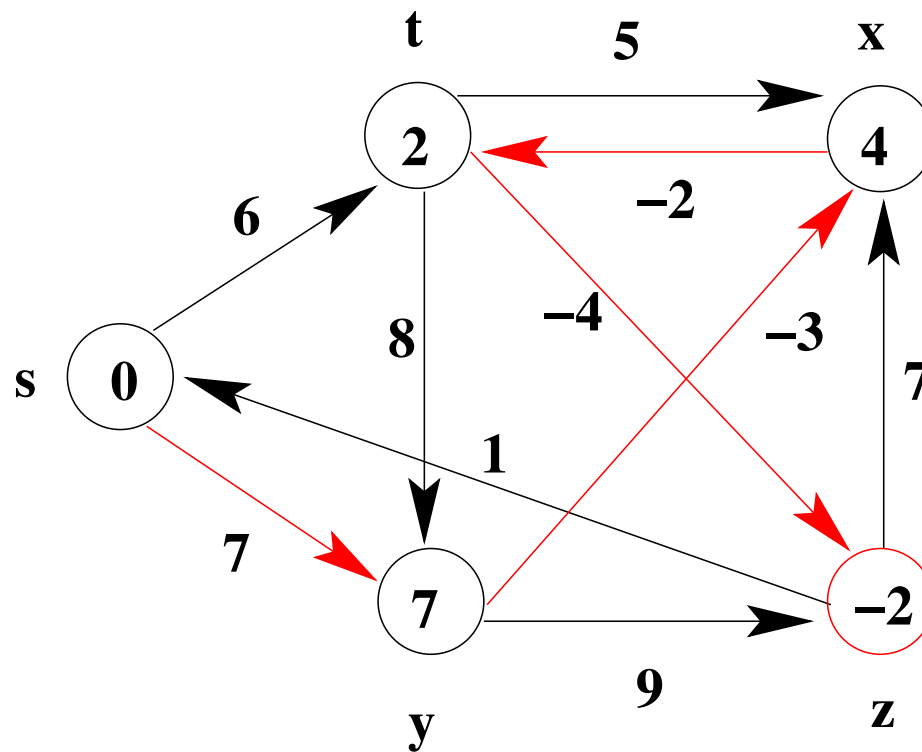
$$E[G] = \{(t,x),(t,y),(t,z),(x,t),(y,x),(z,x),(z,s),(s,t),(s,y)\}$$



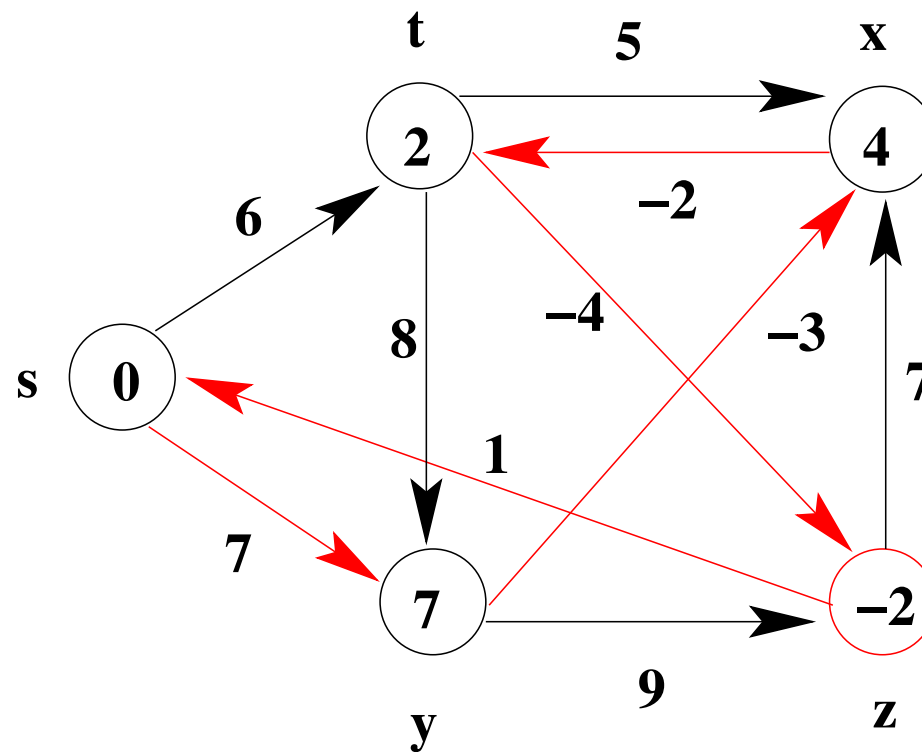
$$E[G] = \{(t,x),(t,y),(t,z),(x,t),(y,x),(z,x),(z,s),(s,t),(s,y)\}$$



$$E[G] = \{(t,x),(t,y),(t,z),(x,t),(y,x),(z,x),(z,s),(s,t),(s,y)\}$$



$$E[G] = \{(t,x),(t,y),(t,z),(x,t),(y,x),(z,x),(z,s),(s,t),(s,y)\}$$



$$E[G] = \{(t,x),(t,y),(t,z),(x,t),(y,x),(z,x),(z,s),(s,t),(s,y)\}$$

Complessità

Bellman-Ford(G, s)

```
1:  $n \leftarrow |V[G]|$ 
2: Initialize( $G, s$ )
3: for  $i \leftarrow 1$  to  $n - 1$  do
4:   for all  $(u, v) \in E[G]$  do
5:     Relax( $G, u, v$ )
6:   end for
7: end for
8: for all  $(u, v) \in E[G]$  do
9:   if  $d[u] + w(u, v) < d[v]$  then
10:    return FALSE
11:   end if
12: end for
13: return TRUE
```

1. la fase di inizializzazione costa $\Theta(n)$;
2. il primo ciclo for costa $(n - 1) \cdot m \cdot \Theta(1) = \Theta(nm)$;
3. il secondo ciclo for costa $\Theta(m)$;

Quindi in totale il costo di Bellman-Ford è

$$\Theta(n + m + nm) = \Theta(nm)$$

cioè $\Theta(n^3)$ qualora il grafo sia denso.

Grafi diretti aciclici (DAG)

Supponiamo di avere un grafo $G = (V, E, w)$ diretto e aciclico (DAG) con pesi reali (possibilmente negativi).

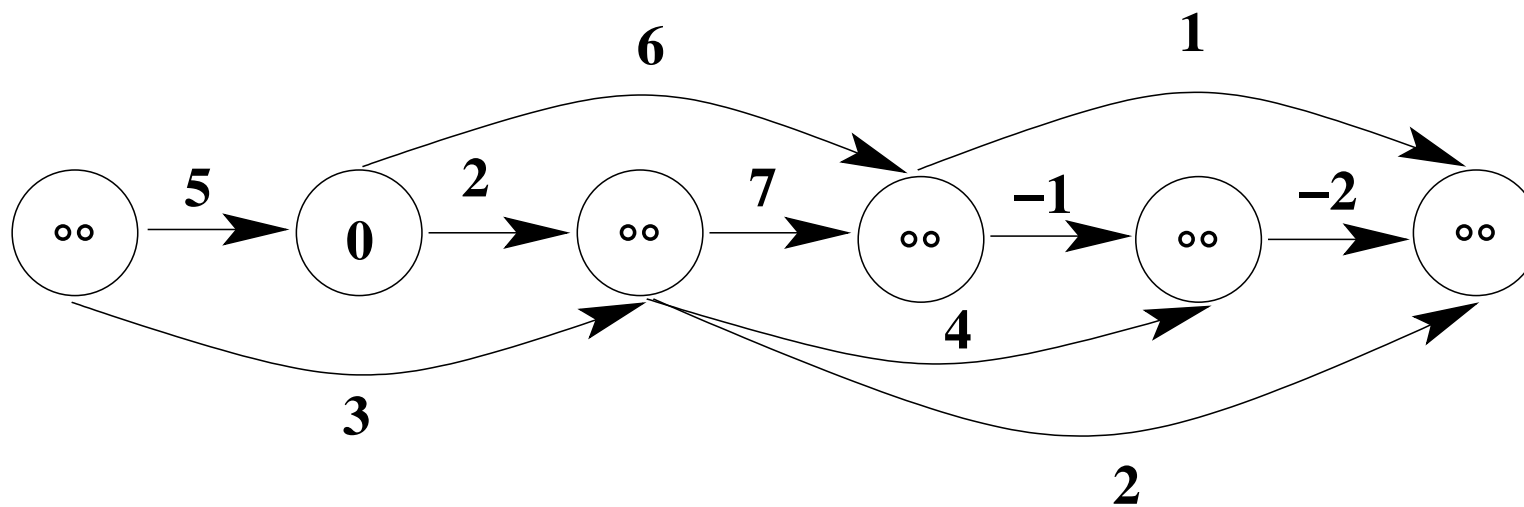
Essendo G aciclico, esso non contiene cicli di peso negativo, quindi tutti i cammini minimi esistono e l'algoritmo di Bellman-Ford ritorna TRUE.

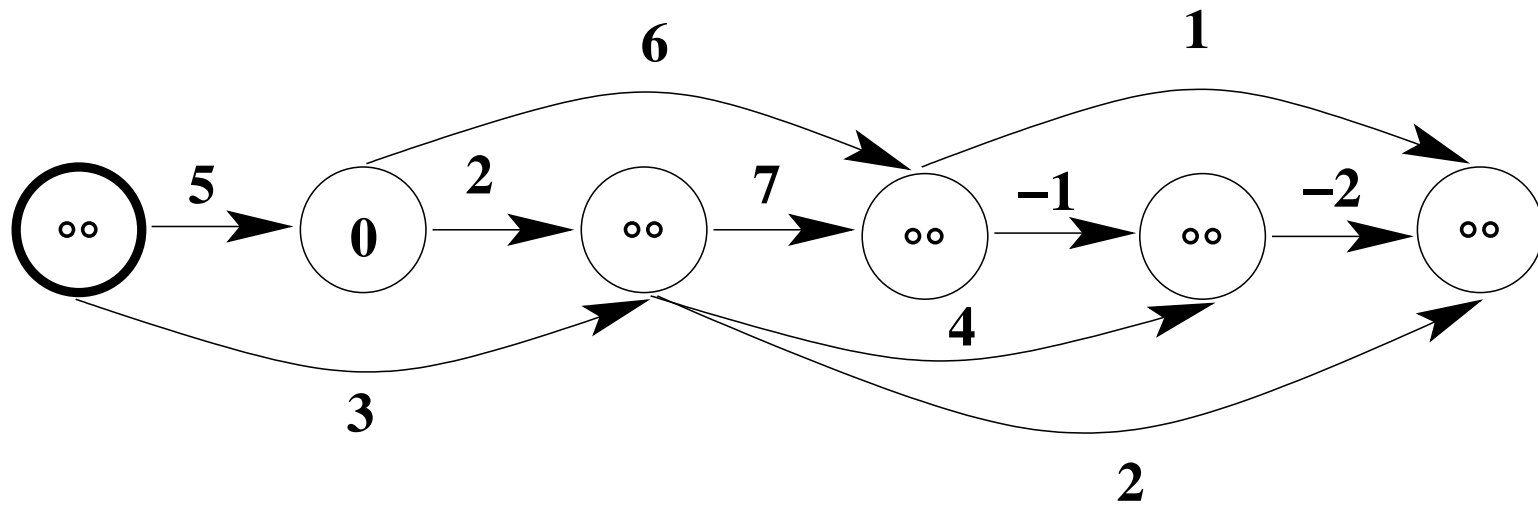
Ordiniamo topologicamente i nodi del grafo, e inseriamo in E gli archi in quest'ordine: prima tutti gli archi (in ordine qualsiasi) uscenti dal primo nodo in ordine topologico, poi tutti gli archi uscenti dal secondo nodo in ordine topologico, e così via.

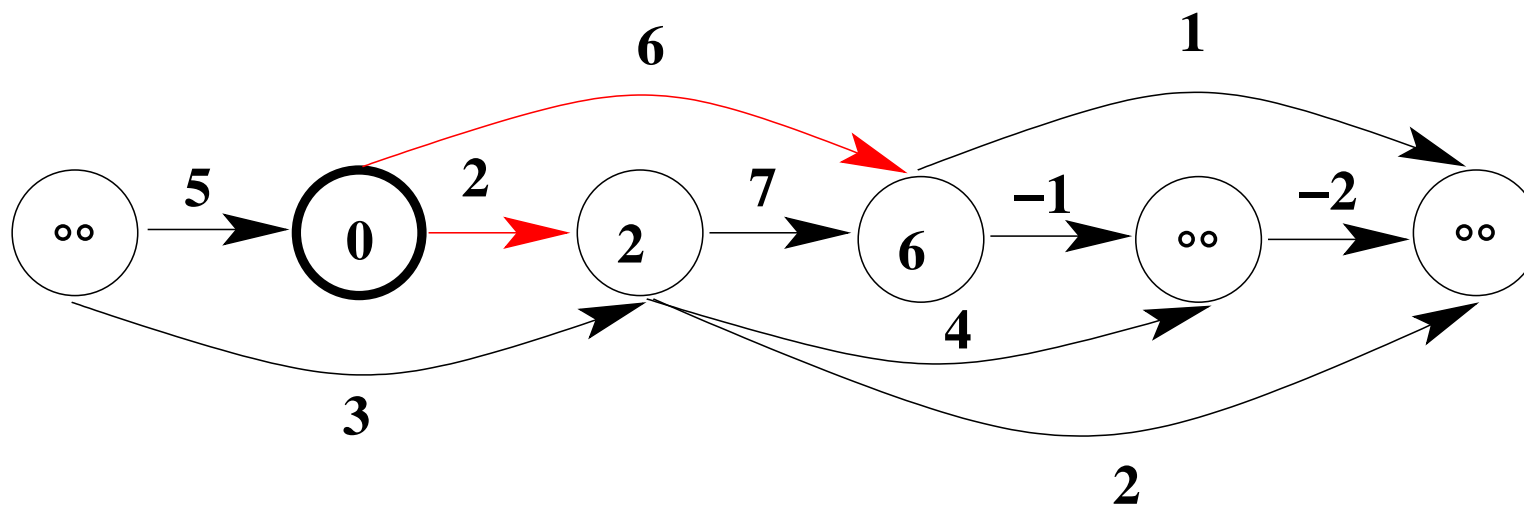
Ora lanciamo Bellman-Ford sul grafo G . Cosa succede?

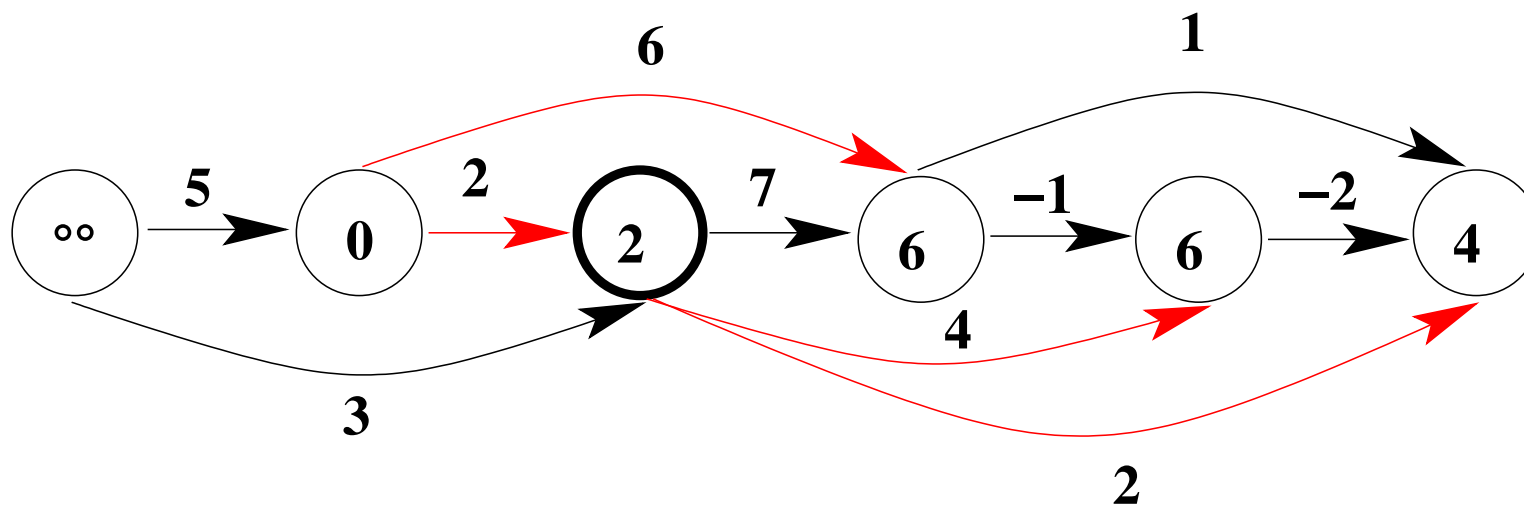
Bellman-Ford(G, s)

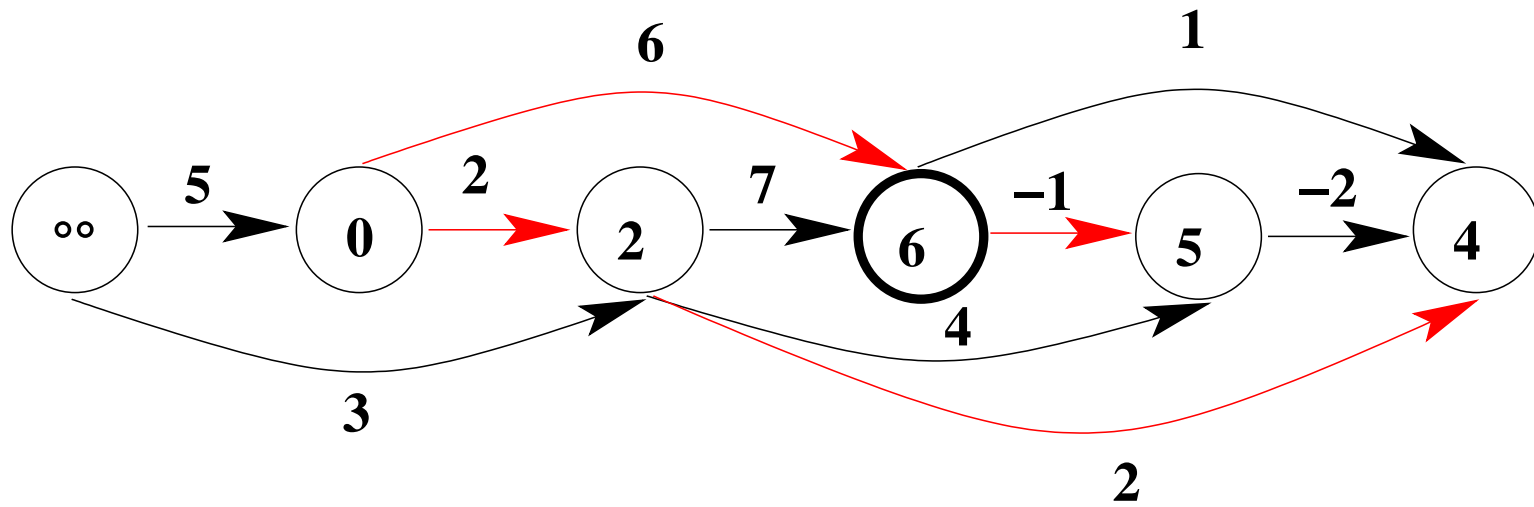
```
1:  $n \leftarrow |V[G]|$ 
2: Initialize( $G, s$ )
3: for  $i \leftarrow 1$  to  $n - 1$  do
4:   for all  $(u, v) \in E[G]$  do
5:     Relax( $G, u, v$ )
6:   end for
7: end for
8: for all  $(u, v) \in E[G]$  do
9:   if  $d[u] + w(u, v) < d[v]$  then
10:    return FALSE
11:   end if
12: end for
13: return TRUE
```

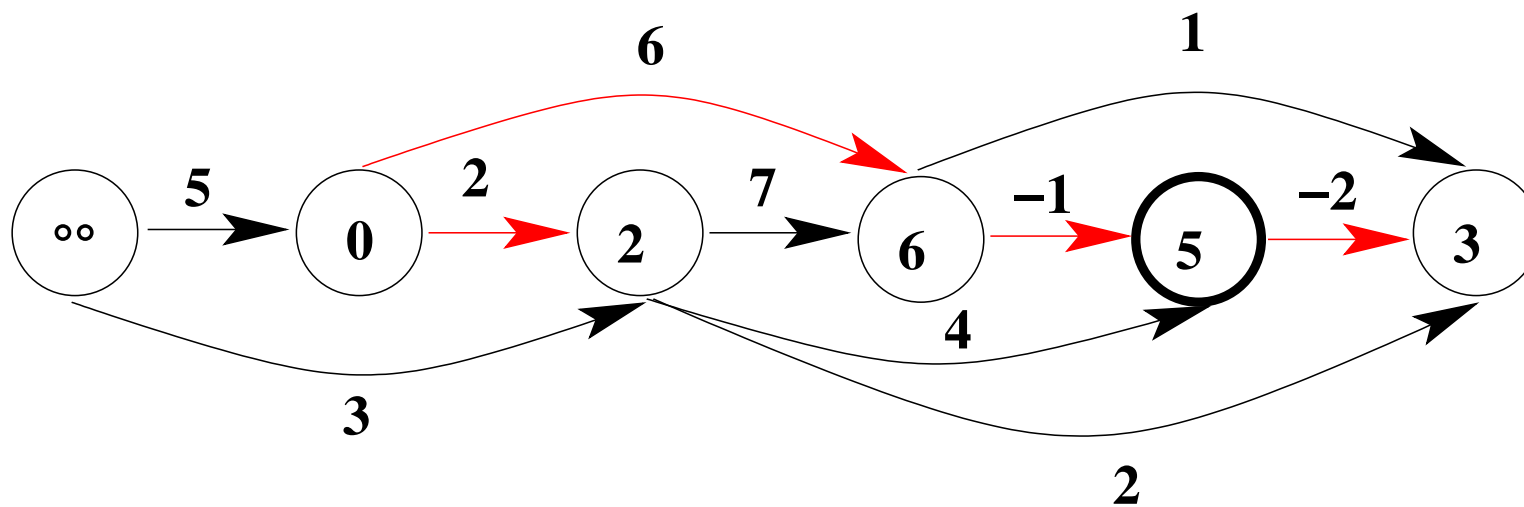


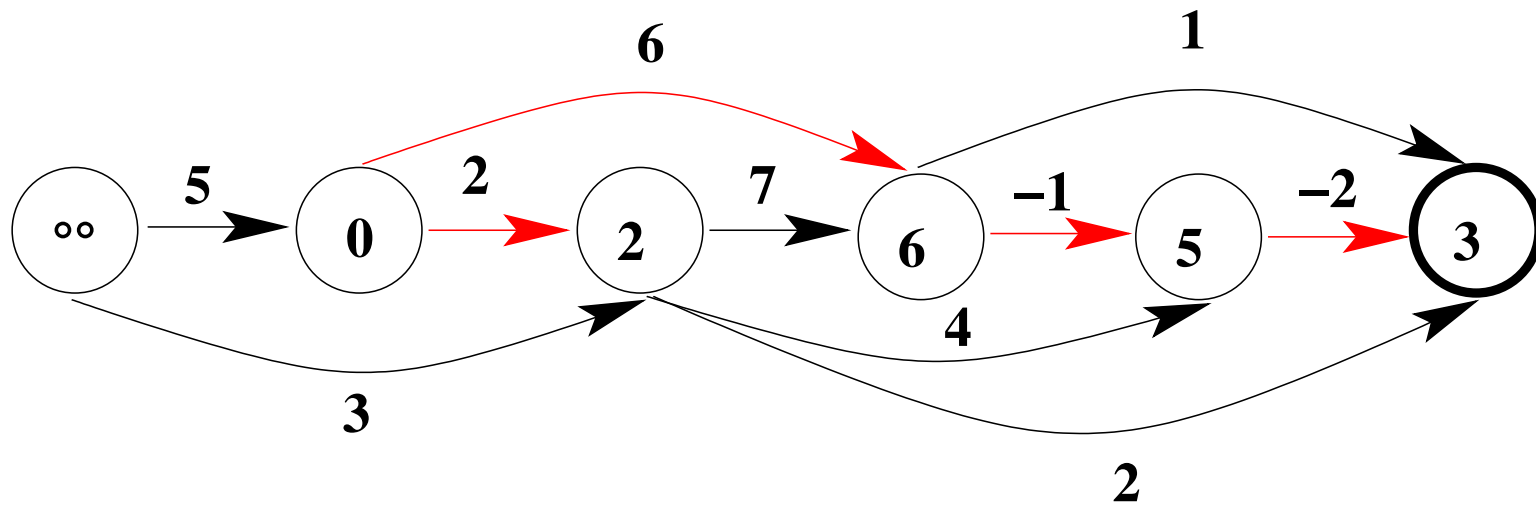












In sostanza, posso suddividere i grafi diretti pesati sui numeri reali in tre categorie in funzione del comportamento di Bellman-Ford su tali grafi:

1. grafi aciclici: in questo caso, ordinando opportunamente gli archi, Bellman-Ford raggiunge uno stato stabile in un passo;
2. grafi ciclici privi di cicli negativi: in questo caso Bellman-Ford raggiunge uno stato stabile in k passi, con $1 \leq k \leq n - 1$;
3. grafi ciclici con cicli negativi: in questo caso Bellman-Ford non raggiunge mai uno stato stabile.

DAG-Shortest-Paths(G, s)

```
1: Initialize( $G, s$ )
2: TopologicalSort( $G, S$ )
3: while  $S \neq \emptyset$  do
4:    $u \leftarrow \text{Pop}(S)$ 
5:   for all  $v \in \text{Adj}[u]$  do
6:     Relax( $G, u, v$ )
7:   end for
8: end while
```

Complessità

- la fase di inizializzazione costa $\Theta(n)$;
- l'ordinamento topologico costa $\Theta(n + m)$;
- durante il ciclo while, faccio n di estrazioni dalla pila, ognuna con costo costante. Quindi il costo di tutte le estrazioni risulta $\Theta(n)$;
- durante il ciclo while, chiamo m volte la procedura *Relax*. Tale procedura ha costo costante, quindi il costo di tutti i rilassamenti ammonta a $\Theta(m)$.

Il costo di DAG-Shortest-Paths è dunque

$$\Theta(3n + 2m) = \Theta(n + m)$$

contro

$$\Theta(nm)$$

nel caso di Bellman-Ford.

Algoritmo di Dijkstra

L'algoritmo di Dijkstra risolve il problema del cammino minimo con una sorgente e pesi reali non negativi. Dunque assumiamo che $w(u, v) \geq 0$ per ogni arco (u, v) del grafo.

L'algoritmo mantiene un insieme S , inizialmente vuoto, che contiene i nodi terminati (per i quali un cammino minimo è già stato calcolato), e un insieme $Q = V \setminus S$ di nodi ancora da scoprire.

Fino a quando Q contiene qualche elemento, l'algoritmo:

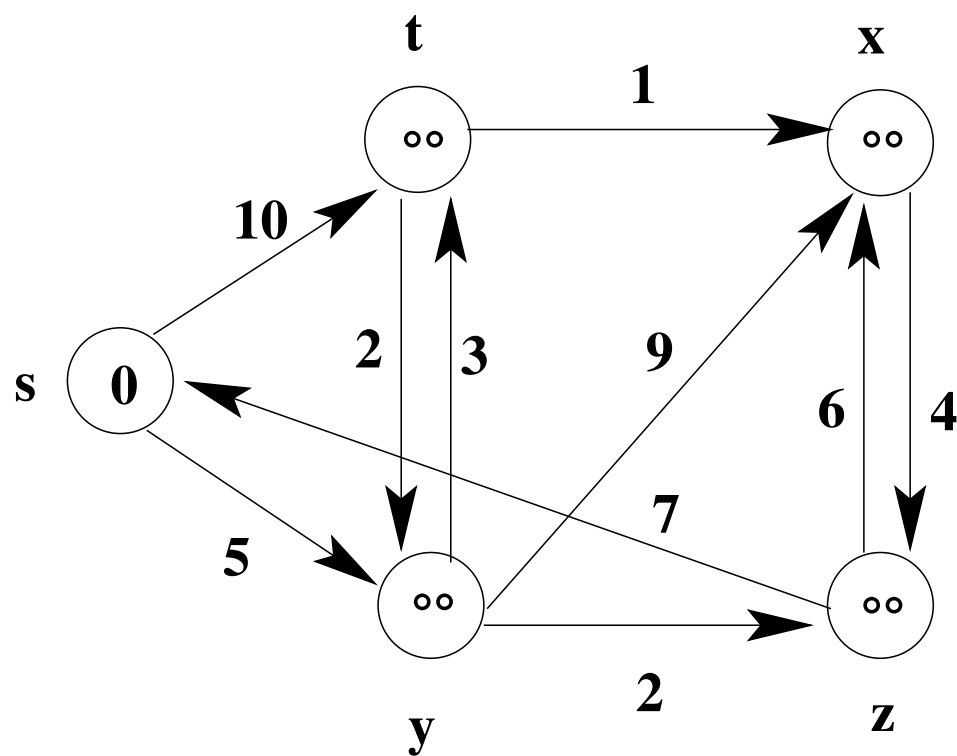
- estrae da Q il nodo u più vicino alla sorgente (cioè con la più piccola stima della distanza dalla sorgente);
- aggiunge u a S ;
- rilassa tutti gli archi che escono da u .

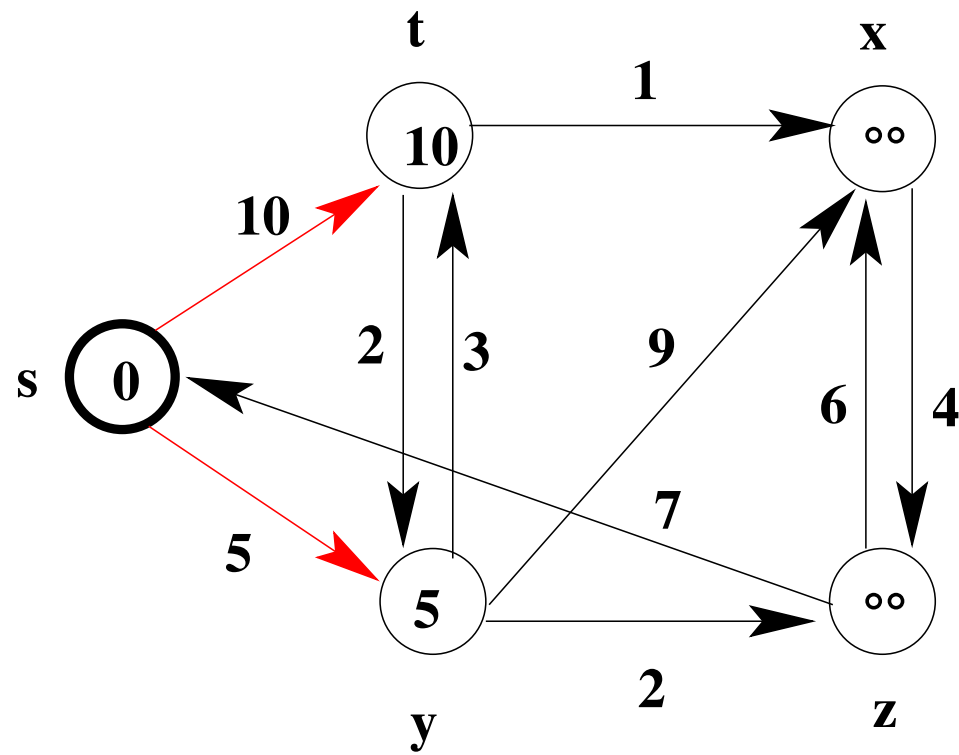
L'insieme Q viene implementato con una coda con priorità basata sul minimo. Ogni nodo v viene inserito nella coda con priorità pari a $d[v]$.

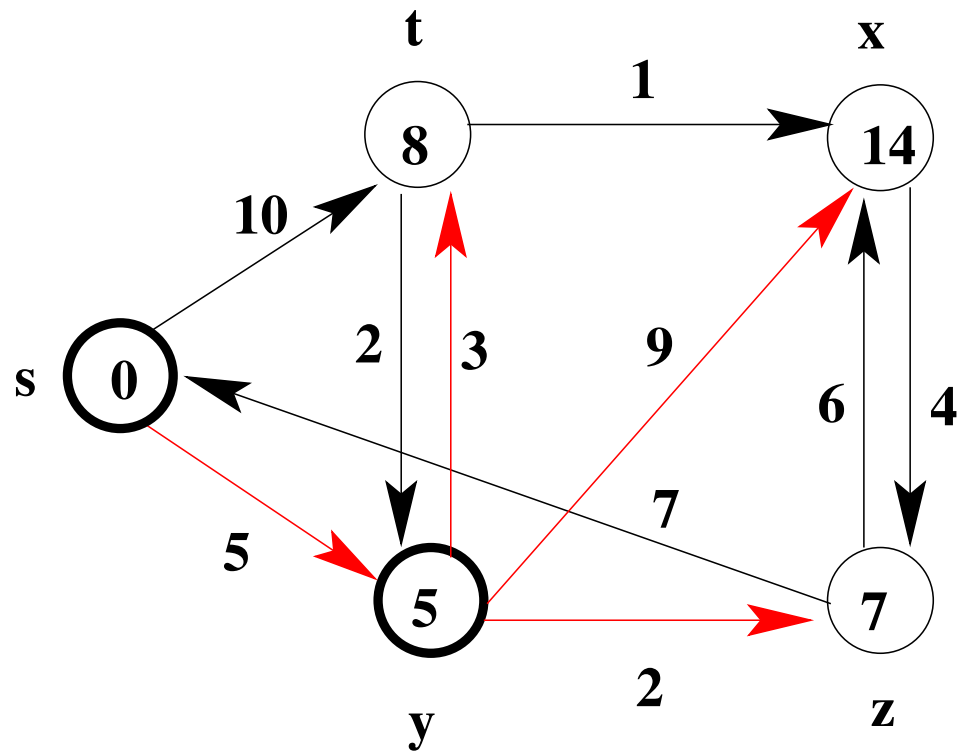
Algoritmo di Dijkstra

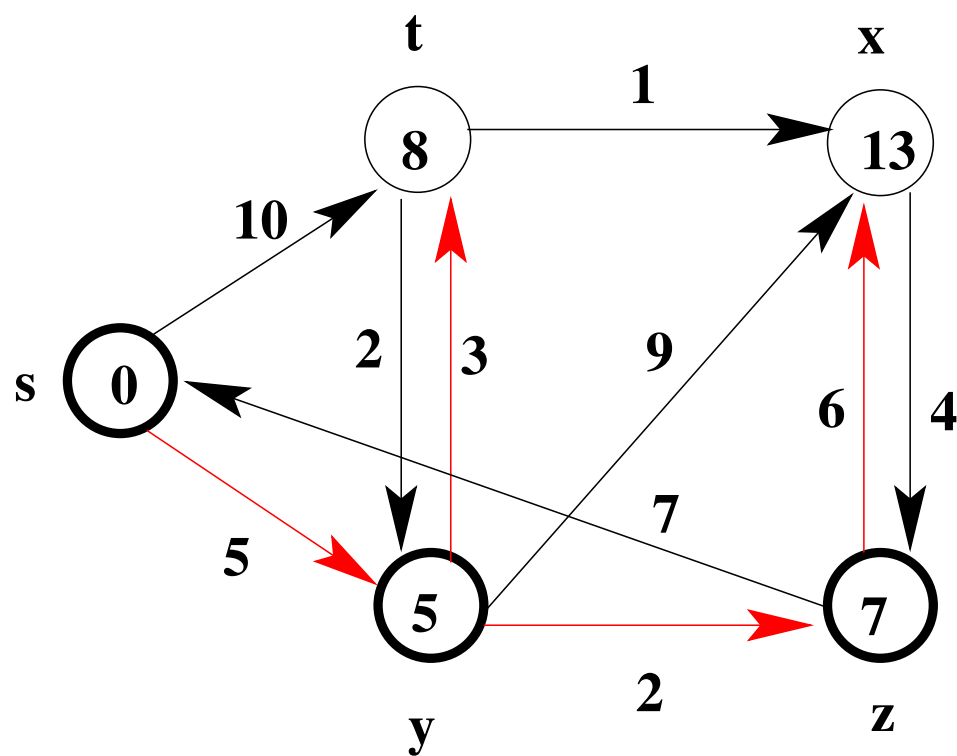
Dijkstra(G, s)

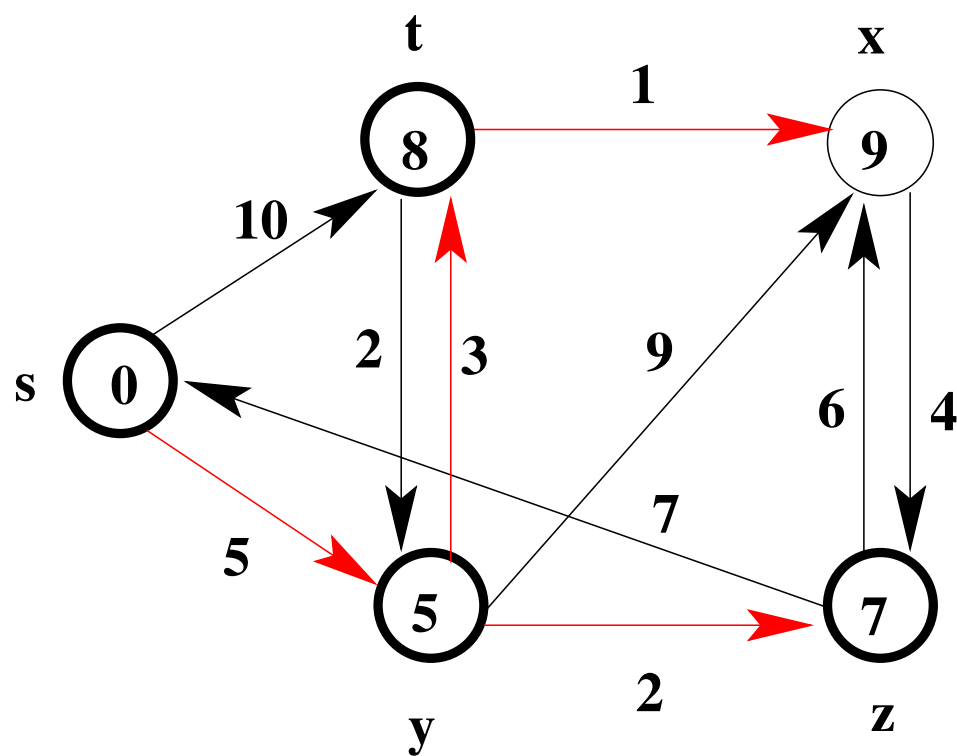
```
1: Initialize( $G, s$ )
2:  $S \leftarrow \emptyset$ 
3:  $Q \leftarrow V[G]$ 
4: while  $Q \neq \emptyset$  do
5:    $u \leftarrow \text{ExtractMin}(Q)$ 
6:    $S \leftarrow S \cup \{u\}$ 
7:   for all  $v \in \text{Adj}[u]$  do
8:     Relax( $G, u, v$ )
9:   end for
10: end while
```

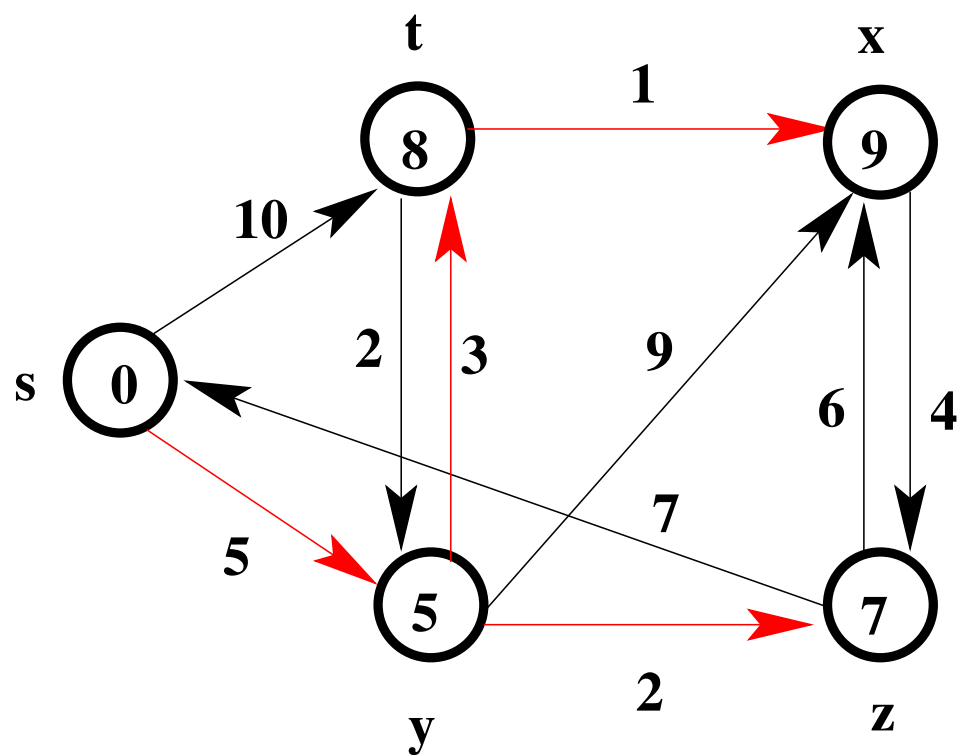












Complessità

Dijkstra(G, s)

```
1: Initialize( $G, s$ )
2:  $S \leftarrow \emptyset$ 
3:  $Q \leftarrow V[G]$ 
4: while  $Q \neq \emptyset$  do
5:    $u \leftarrow \text{ExtractMin}(Q)$ 
6:    $S \leftarrow S \cup \{u\}$ 
7:   for all  $v \in \text{Adj}[u]$  do
8:     Relax( $G, u, v$ )
9:   end for
10: end while
```

- la fase di inizializzazione costa $\Theta(n)$;
- la creazione del min-heap implicita nell'istruzione $Q \leftarrow V[G]$ costa $\Theta(n)$;
- durante il ciclo while, faccio n estrazioni del minimo. Ogni estrazione costa $\Theta(\log n)$, quindi il costo di tutte le estrazioni risulta $\Theta(n \log n)$;
- durante il ciclo while, chiamo m volte la procedura *Relax*. Durante la chiamata di $Relax(G, u, v)$, la stima $d[v]$ può essere aggiornata. Questo implica la chiamata implicita di *ModifyKey* che costa $\Theta(\log n)$. Quindi il costo di tutti i rilassamenti ammonta a $\Theta(m \log n)$.

Il costo di Dijkstra è dunque

$$\Theta(2n + n \log n + m \log n) = \Theta((n + m) \log n)$$

contro

$$\Theta(nm)$$

nel caso di Bellman-Ford.

Se

$$m > \frac{n \log n}{n - \log n}$$

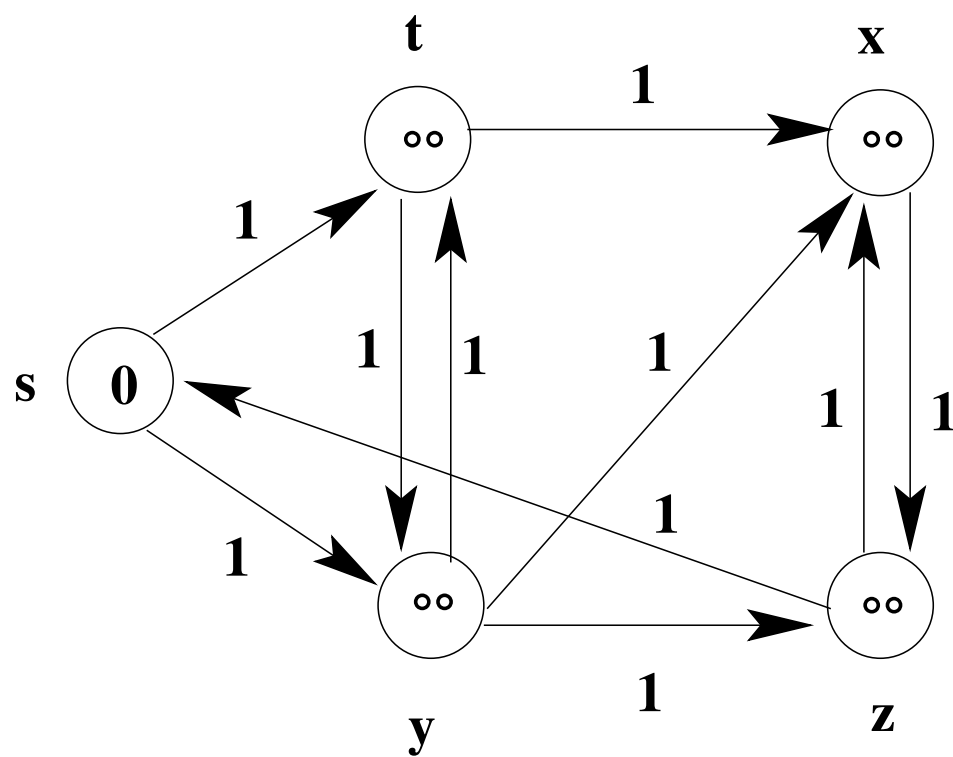
allora Dijkstra è da preferire. In particolare, se $m \geq n$, Dijkstra è più veloce.

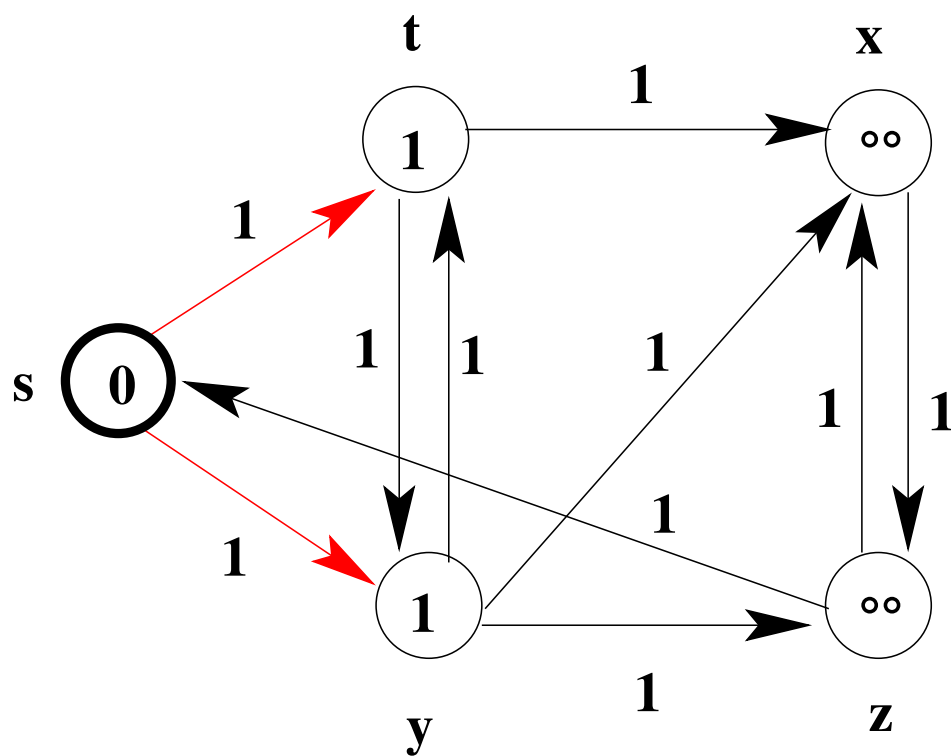
Dijkstra e BFS

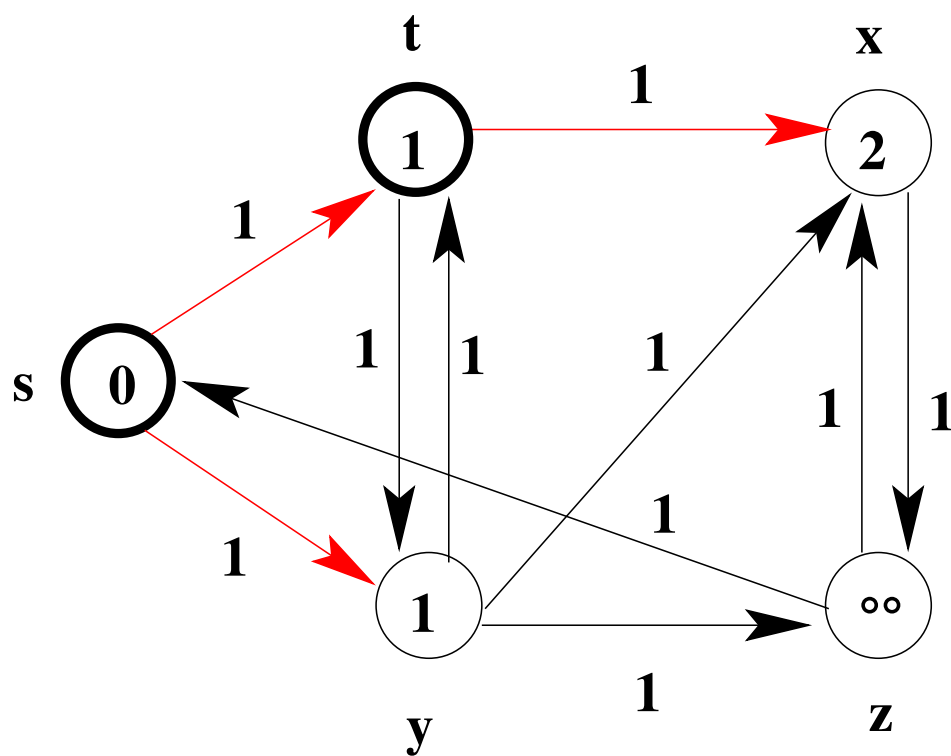
Dijkstra è un algoritmo **greedy**: ad ogni passo, esso inserisce in S il nodo più vicino alla sorgente s .

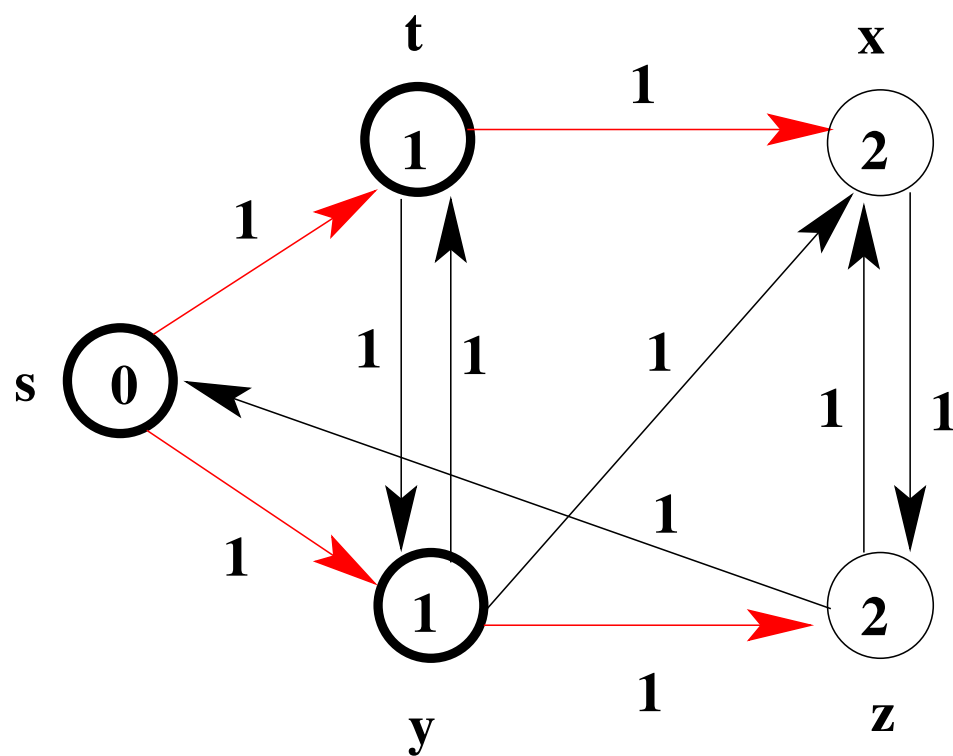
Inoltre, se associamo un peso unitario ad ogni arco, Dijkstra simula la procedura **BFS** che visita in ampiezza un grafo a partire da una sorgente.

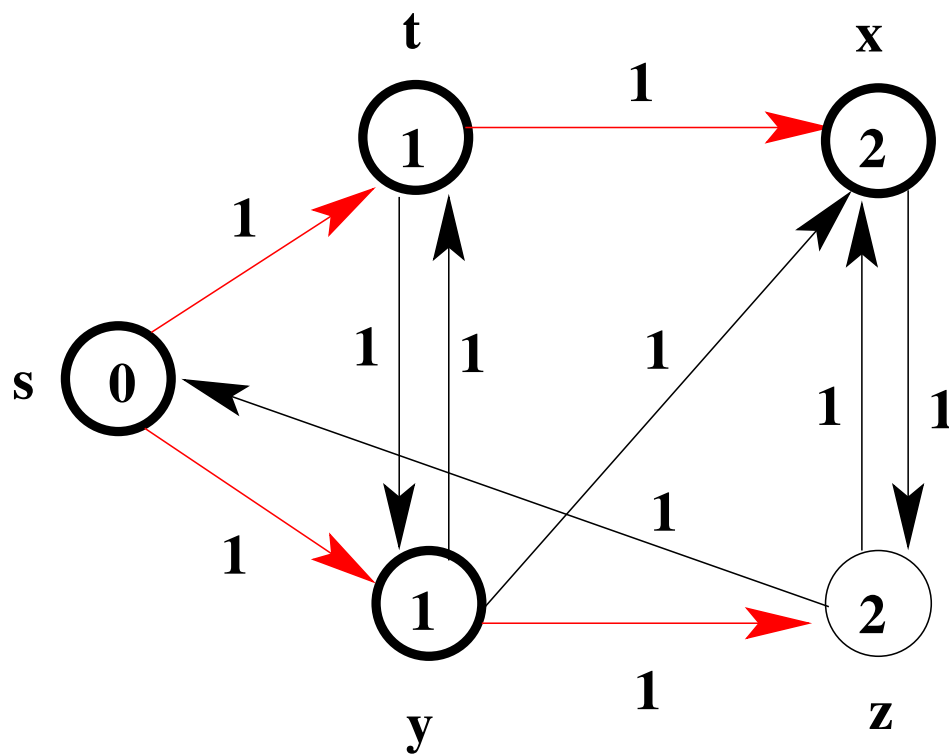
I nodi vengono visitati in ampiezza e inseriti in S una volta scoperti. Un cammino di peso minimo corrisponde ad un cammino di lunghezza minima. L'albero dei cammini di peso minimo corrisponde al BF-albero dei cammini di lunghezza minima.

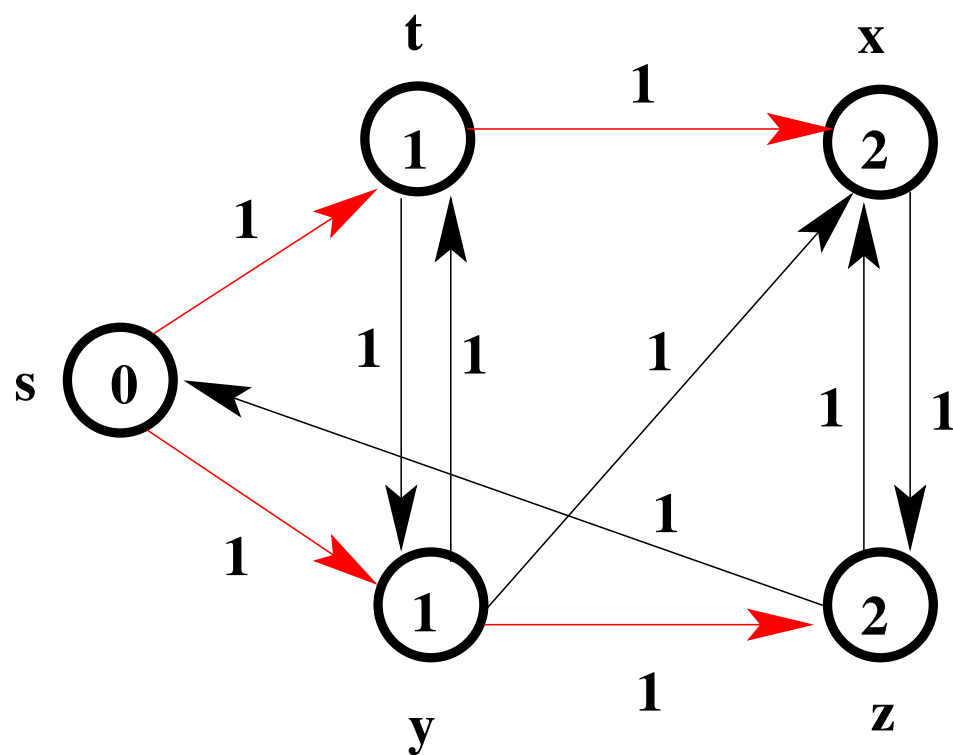












Ricostruzione dei cammini minimi

PrintPath(G, s, v)

```
1: if  $v = s$  then  
2:   print  $s$   
3: else  
4:   if  $\pi[v] = \text{NIL}$  then  
5:     print “no path exists”  
6:   else  
7:     PrintPath( $G, s, \pi[v]$ )  
8:     print  $v$   
9:   end if  
10: end if
```