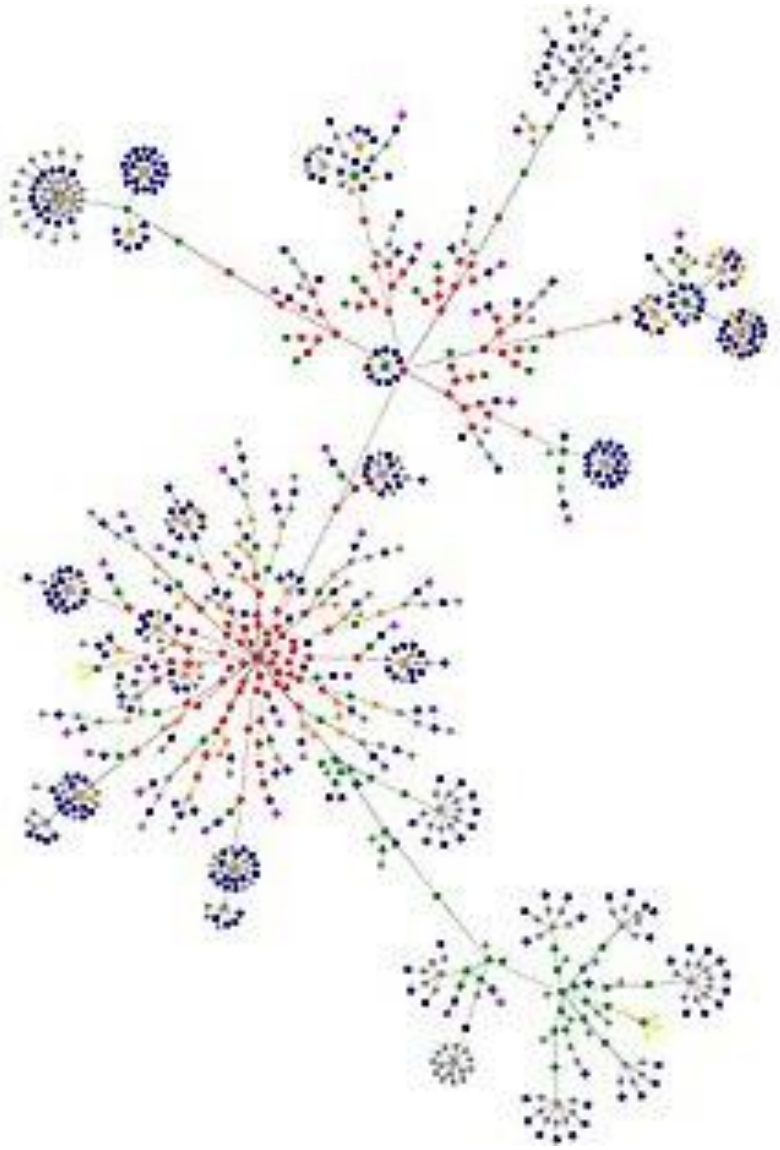
Minimum Spanning Tree: algoritmi e implementazioni

par. 4.5 e 4.6

18 Maggio 2023



Algoritmi greedy su grafi

4.4 Shortest Paths in a Graph (only with positive costs):

Dijkstra Algorithm

4.5 Il problema del MST

4.6 Implementazione degli algoritmi di Prim e di Kruskal:

la struttura dati Union-Find

4.7 Clustering: Applicazione dell'algoritmo di Kruskal

Due piccioni con una fava

Alcuni algoritmi di ottimizzazione su grafi (Prim, Kruskal, Dijkstra) li studierete anche in Ricerca Operativa

Perché?

Punto di vista differente:

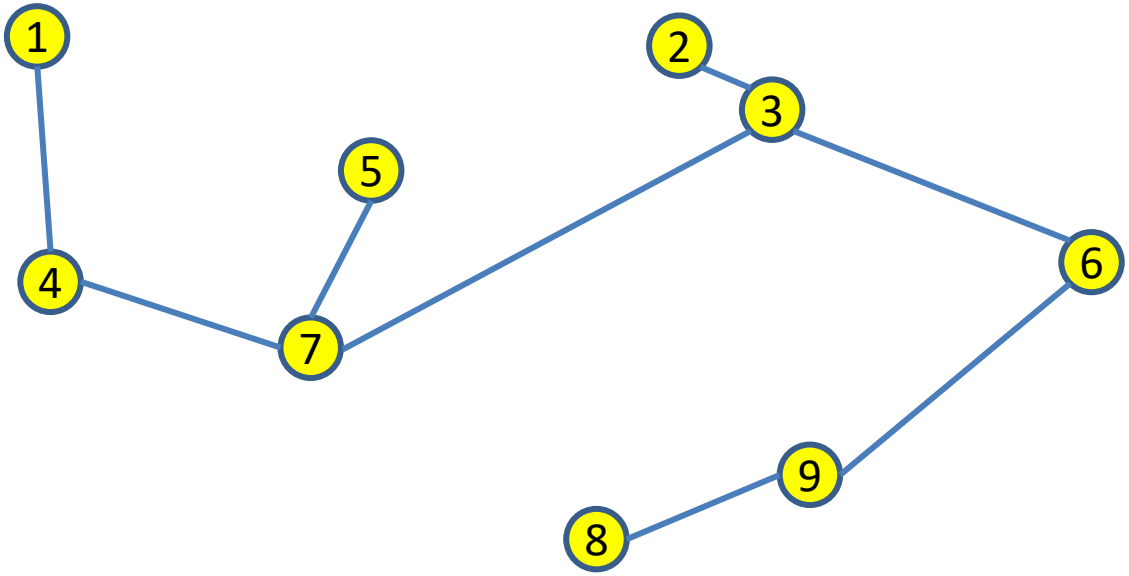
RO: modello matematico

PA: tecniche di progettazione e analisi

Connessione di nodi con costo minimo

Supponiamo di avere un'azienda con varie sedi dislocate in diversi punti della città e di volerle collegare con dei cavi di fibra ottica col minimo costo possibile.

Due sedi possono essere collegate direttamente o attraverso delle altre sedi già collegate.



Connettere nodi con costo minimo

Il problema si può porre anche in altri ambiti.

I nodi potrebbero rappresentare città, case, computer etc. etc., i collegamenti autostrade, tubi idraulici, cavi elettrici, etc. etc. e i costi quelli del pedaggio, della benzina, dei tubi, fili, cavi, etc. etc.

Vediamo di formalizzare il problema.

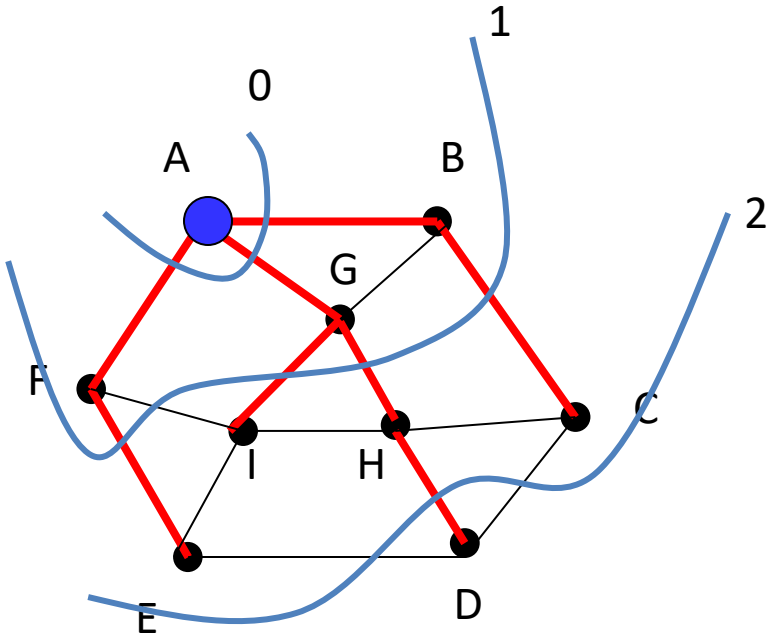
Spanning Tree

- Assume you have an undirected connected graph $G = (V,E)$
- Spanning tree of graph G is a tree $T = (V,E_T)$, with $E_T \subseteq E$
  - Tree has same set of nodes
  - All tree edges are graph edges
- Think: "smallest set of edges needed to connect every node together"

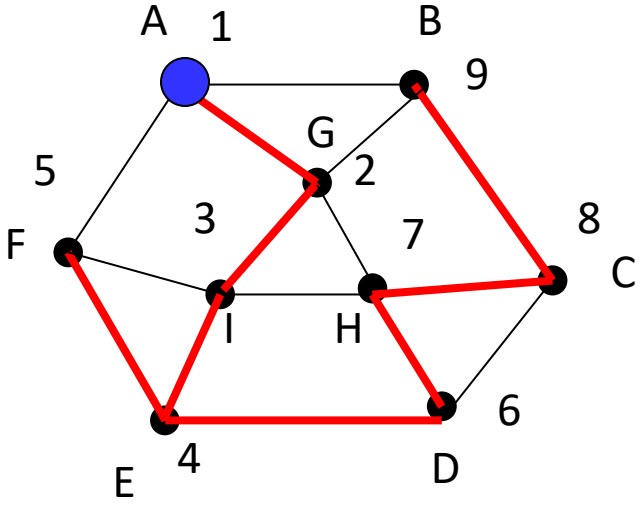
NOTA

- In italiano: albero di copertura (o di ricoprimento)
- nel seguito T potrà indicare semplicemente l'insieme degli archi, sottintendendo che i vertici sono tutti i vertici di V

Spanning trees



Breadth-first Spanning Tree



Depth-first spanning tree

Weighted Spanning Trees

- Assume you have an undirected graph G = (V,E) with weights on each edge
- Spanning tree of graph G is tree T = (V,E_T) with $E_T \subseteq E$
  - Tree has same set of nodes
  - All tree edges are graph edges
  - Weight of spanning tree = sum of tree edge weights

- Minimum Spanning Tree (MST)
  - Any spanning tree whose weight is minimal
  - In general, a graph has several MST's
  - Applications: circuit-board routing, networking, etc.

Applications

MST is a fundamental problem with diverse applications.

Network design.
telephone, electrical, hydraulic, TV cable, computer, road

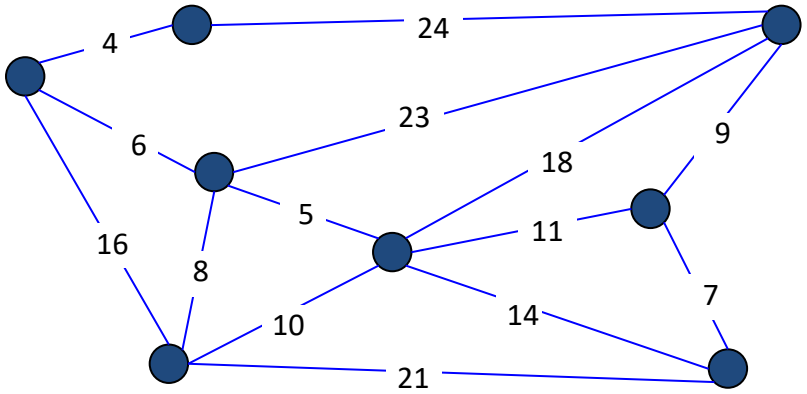
Approximation algorithms for NP-hard problems.
traveling salesperson problem, Steiner tree

Indirect applications.
max bottleneck paths
LDPC codes for error correction
image registration with Renyi entropy
learning salient features for real-time face verification
reducing data storage in sequencing amino acids in a protein
model locality of particle interactions in turbulent fluid flows
autoconfig protocol for Ethernet bridging to avoid cycles in a network

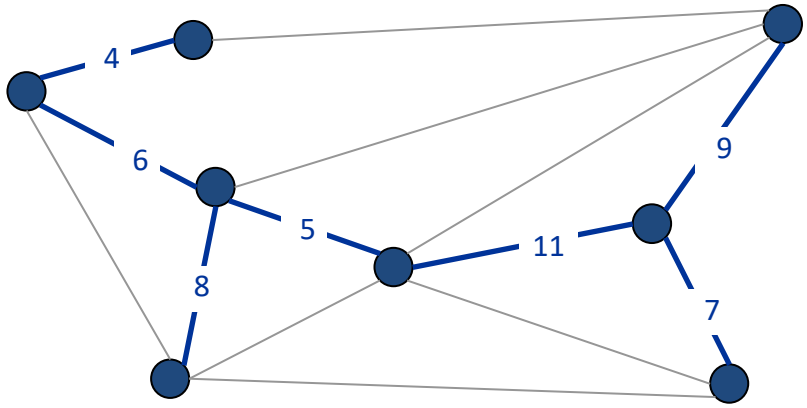
Cluster analysis.

Minimum Spanning Tree (MST)

Minimum spanning tree. Given a connected graph G = (V, E) with real-valued edge weights c_e, an **MST** is a subset of the edges $T \subseteq E$ such that (V,T) is a tree (connected and acyclic), denoted spanning tree, whose sum of edge weights is minimized.



G = (V, E)



T, $\Sigma_{e \in T} c_e = 50$

Recall: a tree with n nodes has n-1 edges.

Cayley's Theorem. There are n^{n-2} spanning trees of K_n: can't solve by brute force!

Greedy Algorithms: possible choices

1. Sort by increasing edge costs (keeping connectivity): Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

2. Sort by increasing edge costs (keeping acyclity): Start with T = ϕ. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

3. Sort by decreasing edge costs: Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

Quale può funzionare?

Greedy Algorithms

All three algorithms produce an MST!!!

Prim's algorithm. Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

Kruskal's algorithm. Start with $T = \phi$. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

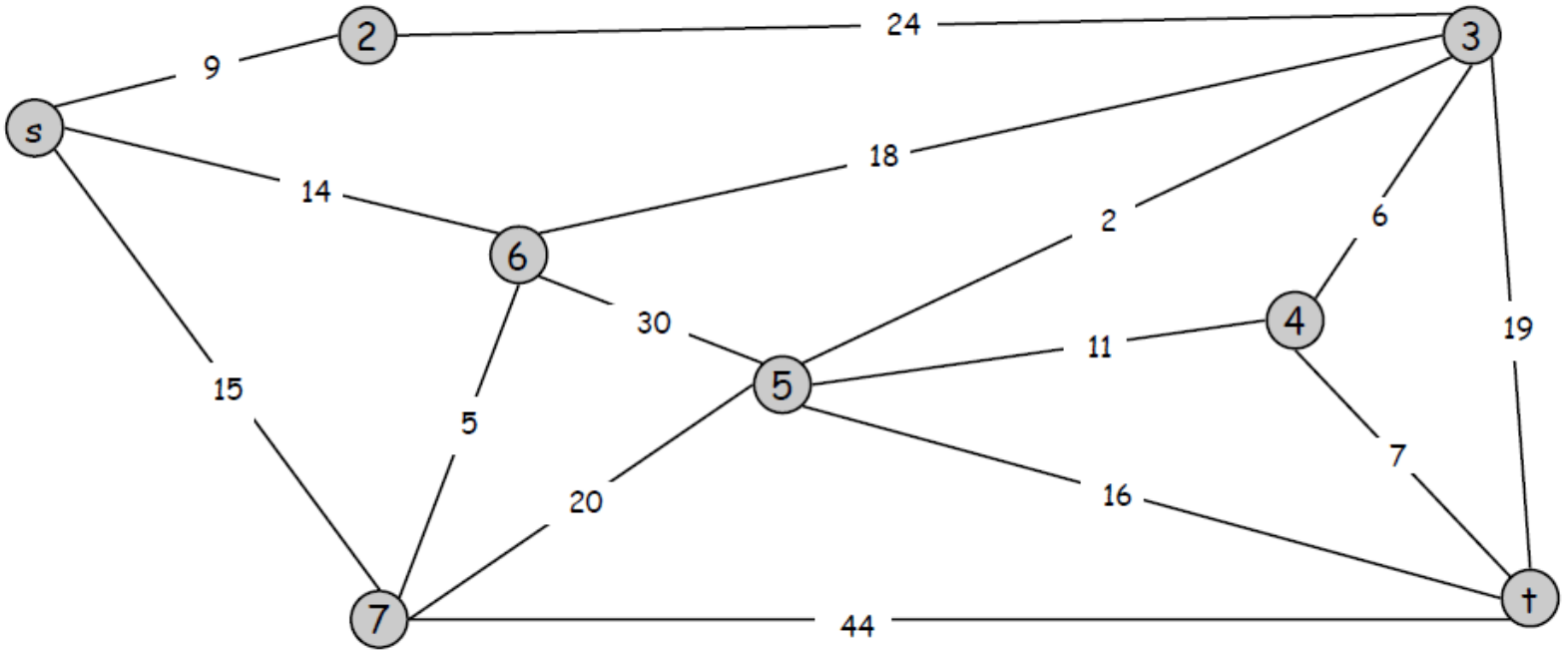
Reverse-Delete algorithm. Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

Algoritmo di Prim

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ e con un nodo radice s.
- Aggiungi arco a T che è incidente solo su un nodo in T e con costo minimo.

T = ϕ



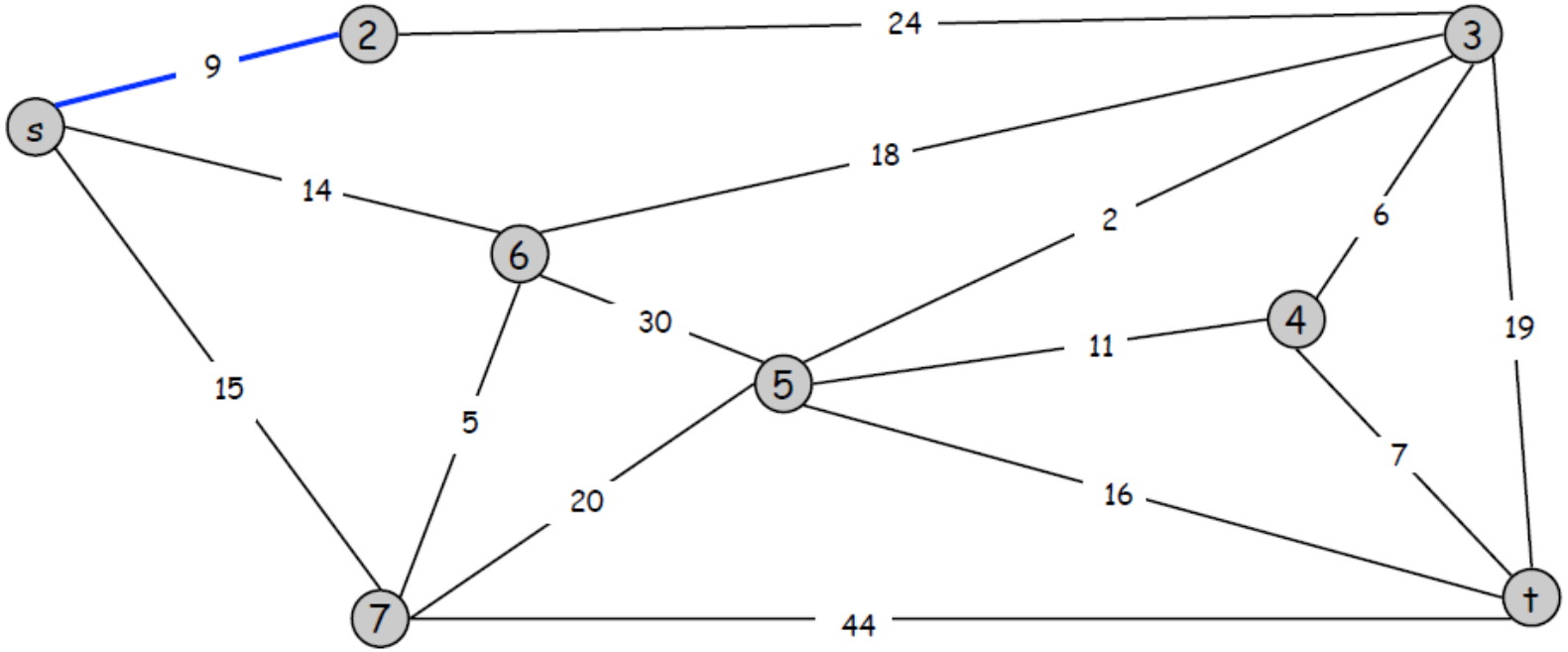
Nota: simile ad algoritmo di Dijkstra (che vedremo per il problema dei cammini minimi)

Algoritmo di Prim

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ e con un nodo radice s.
- Aggiungi arco a T che è incidente solo su un nodo in T e con costo minimo.

T = { {s,2} }

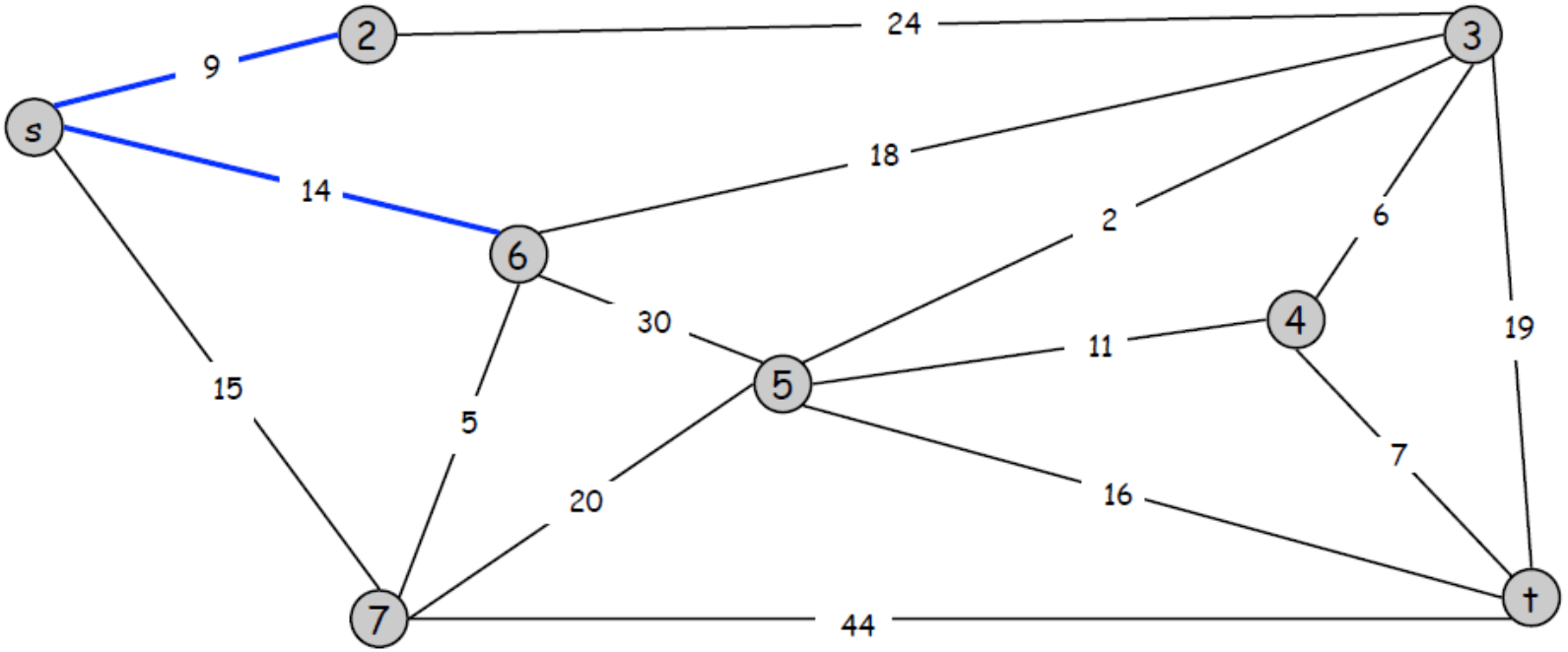


Algoritmo di Prim

Trovare Minimo Spanning Tree.

- Inizia con $T = \phi$ e con un nodo radice s.
- Aggiungi arco a T che è incidente solo su un nodo in T e con costo minimo.

T = { {s,2}, {s,6} }

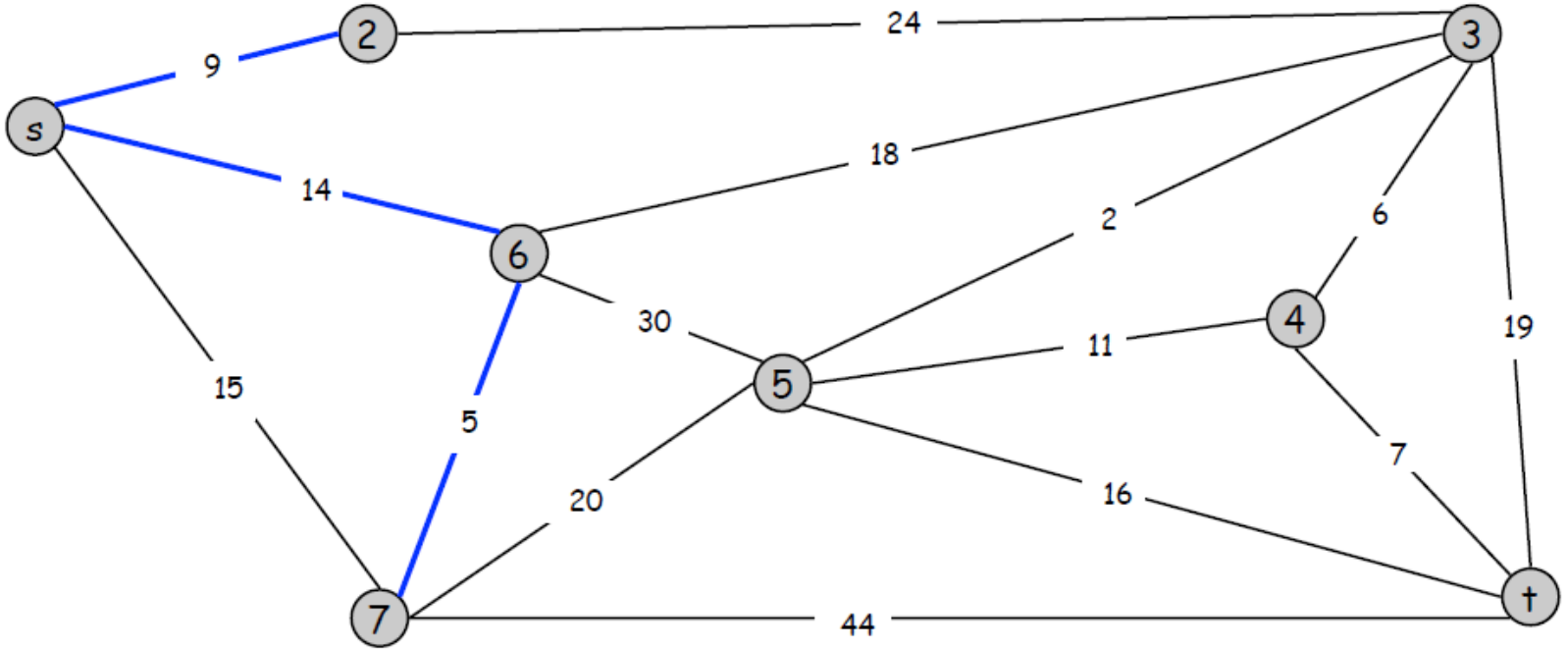


Algoritmo di Prim

Trovare Minimo Spanning Tree.

- Inizia con $T = \phi$ e con un nodo radice s.
- Aggiungi arco a T che è incidente solo su un nodo in T e con costo minimo.

T = { {s,2}, {s,6}, {6,7} }

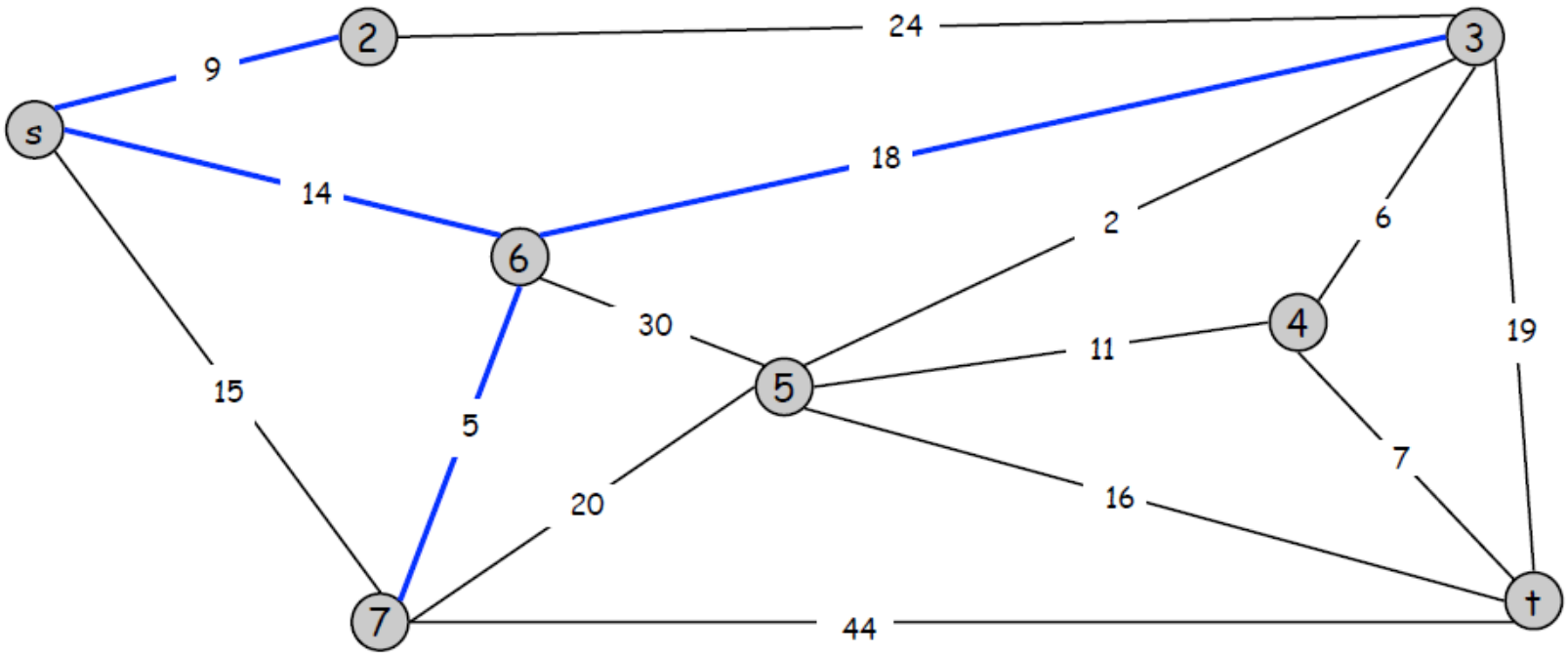


Algoritmo di Prim

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ e con un nodo radice s.
- Aggiungi arco a T che è incidente solo su un nodo in T e con costo minimo.

T = { {s,2}, {s,6}, {6,7}, {6,3} }

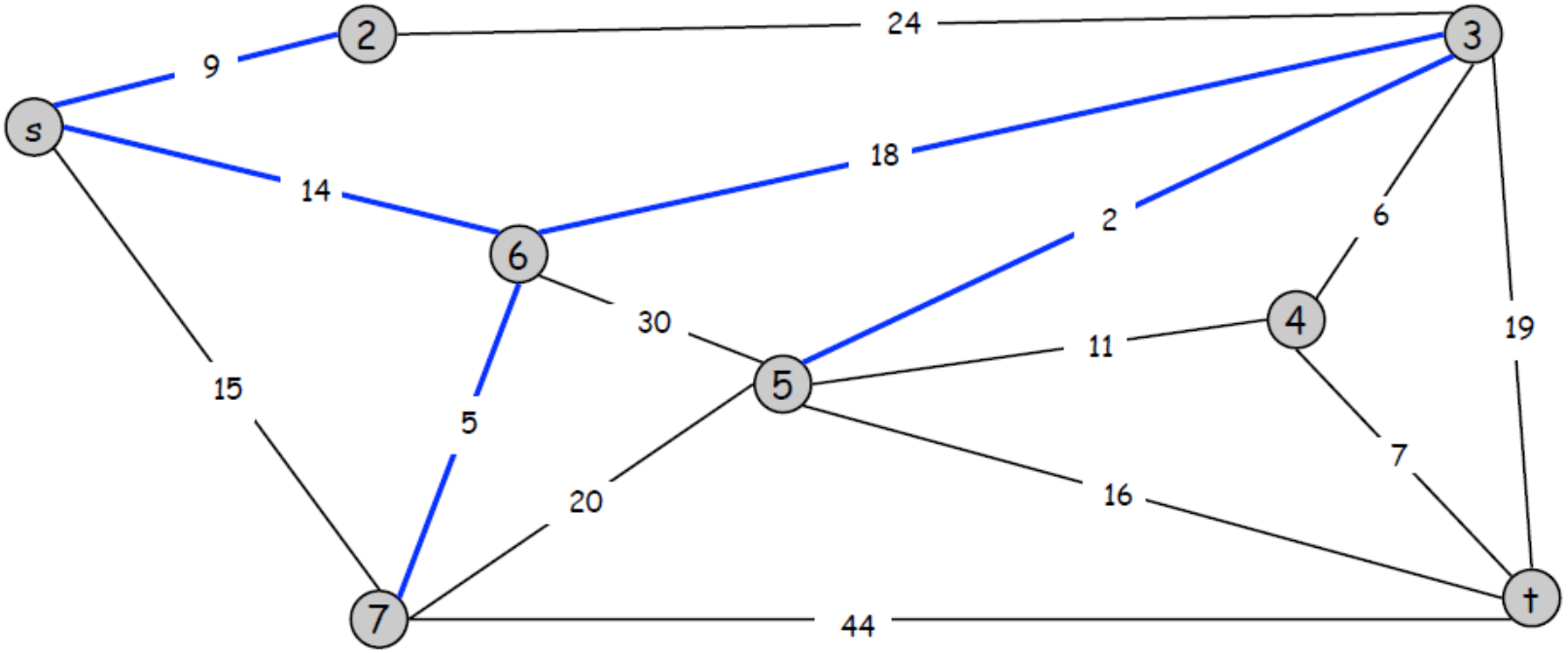


Algoritmo di Prim

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ e con un nodo radice s.
- Aggiungi arco a T che è incidente solo su un nodo in T e con costo minimo.

T = { {s,2}, {s,6}, {6,7}, {6,3}, {3,5} }

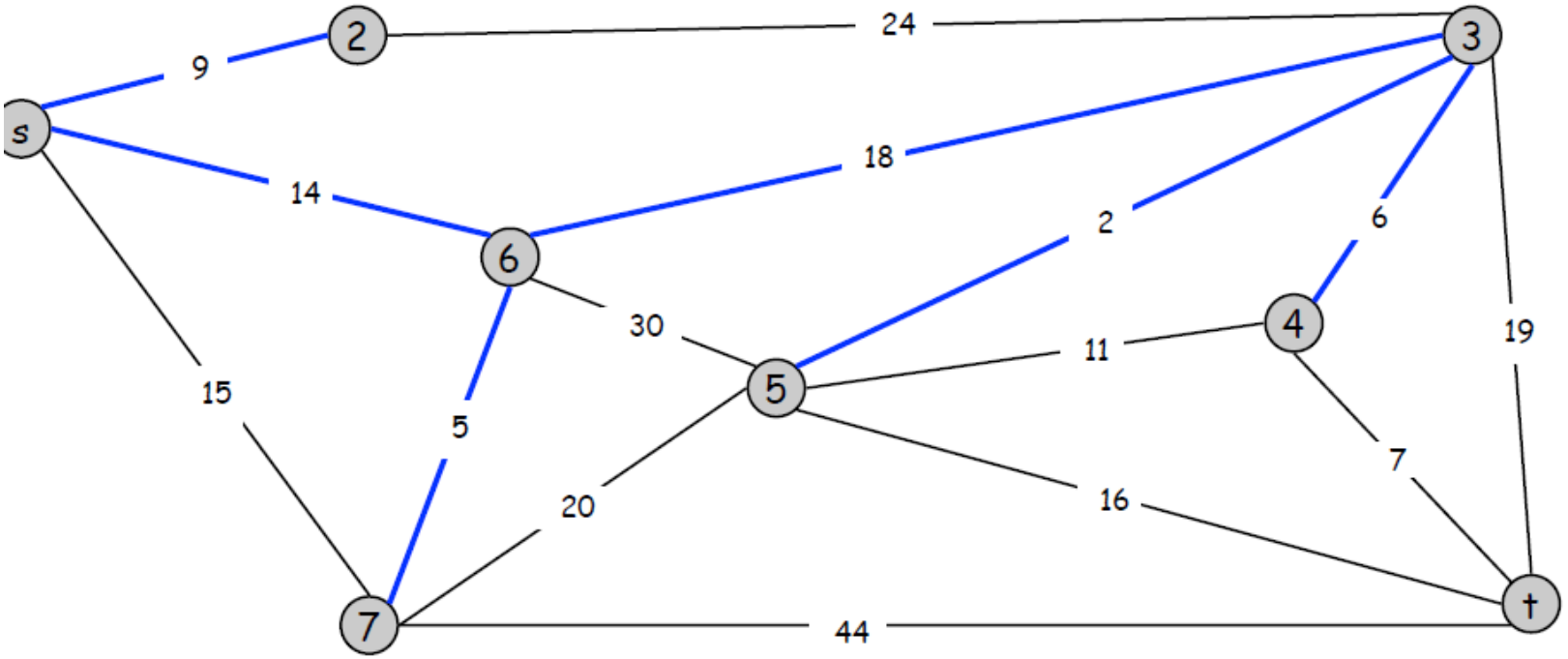


Algoritmo di Prim

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ e con un nodo radice s.
- Aggiungi arco a T che è incidente solo su un nodo in T e con costo minimo.

T = { {s,2}, {s,6}, {6,7}, {6,3}, {3,5}, {3,4} }

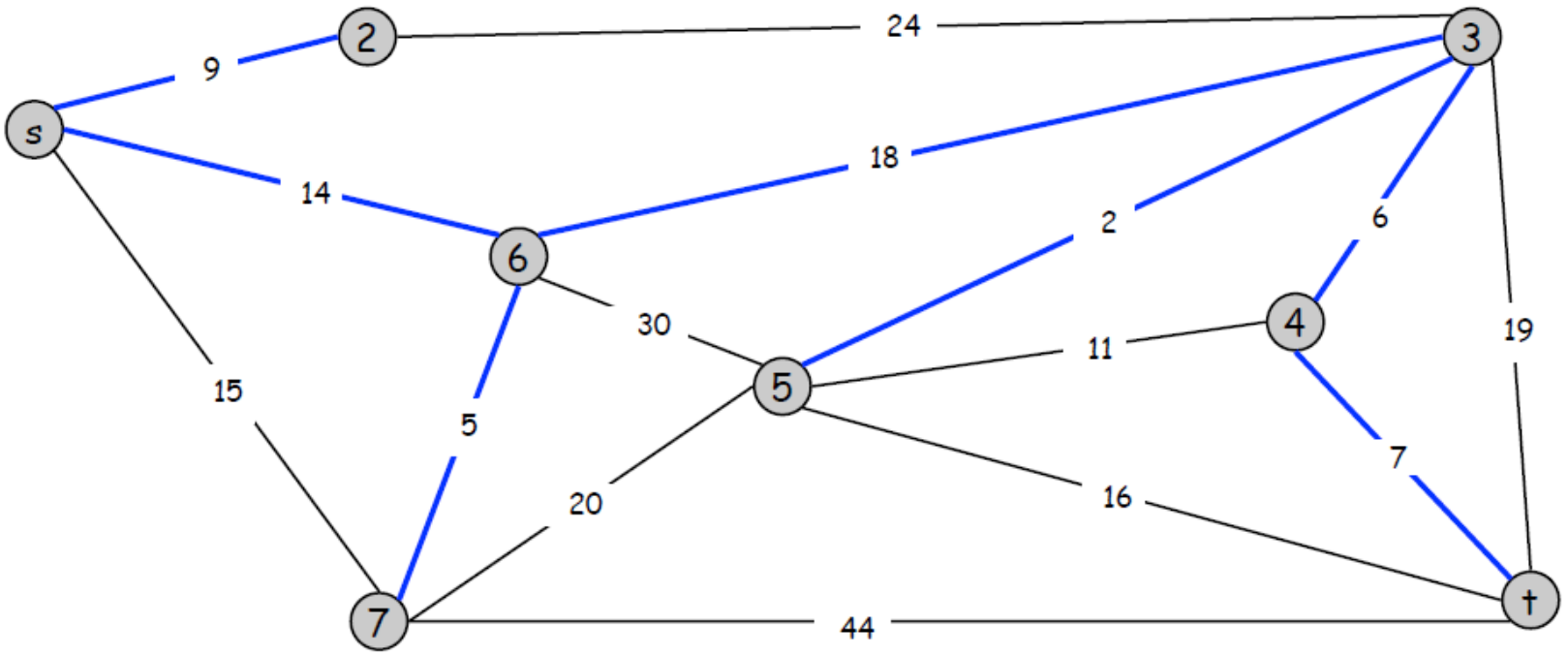


Algoritmo di Prim

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ e con un nodo radice s.
- Aggiungi arco a T che è incidente solo su un nodo in T e con costo minimo.

T = { {s,2}, {s,6}, {6,7}, {6,3}, {3,5}, {3,4}, {4,t} } è MST di costo 9+14+5+18+2+6+7= 61

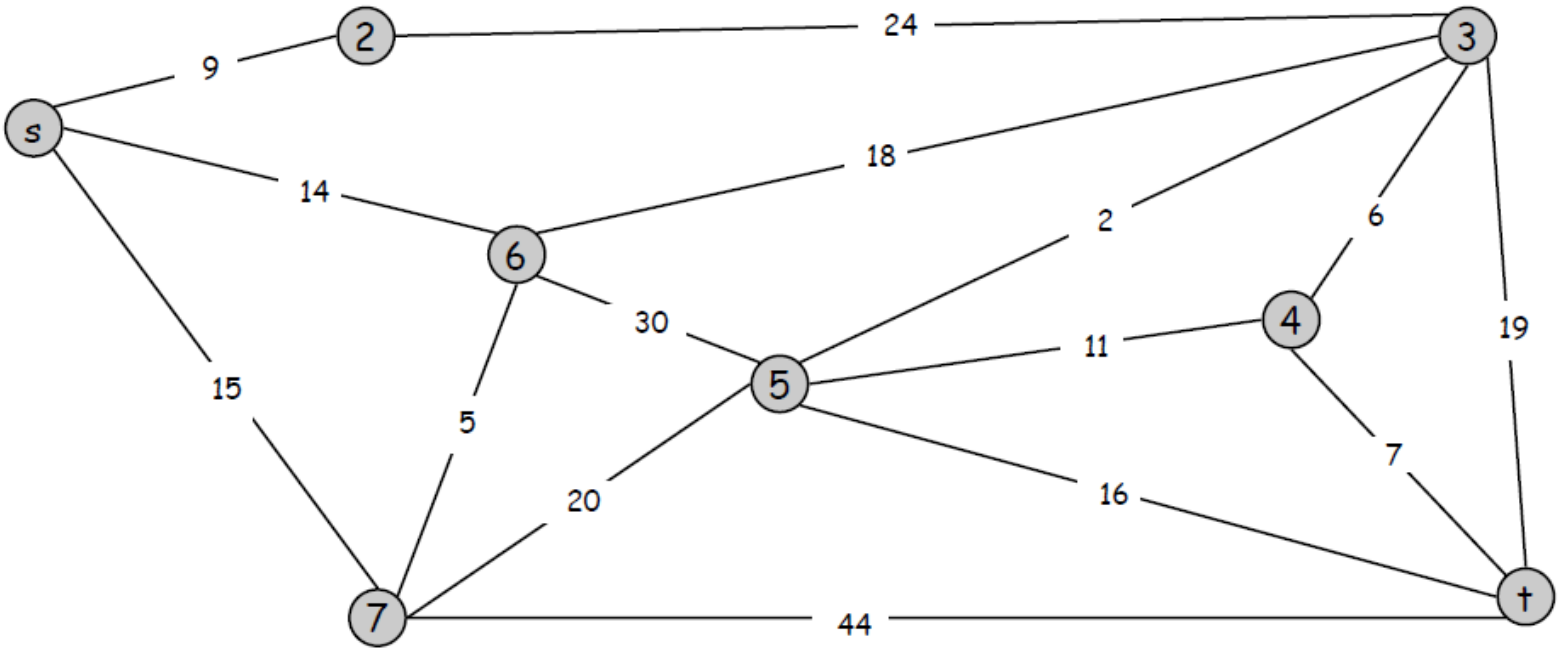


Algoritmo di Kruskal

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ.
- Considera archi in ordine crescente di costo.
- Inserisci l'arco in T se non crea un ciclo.

T = ϕ

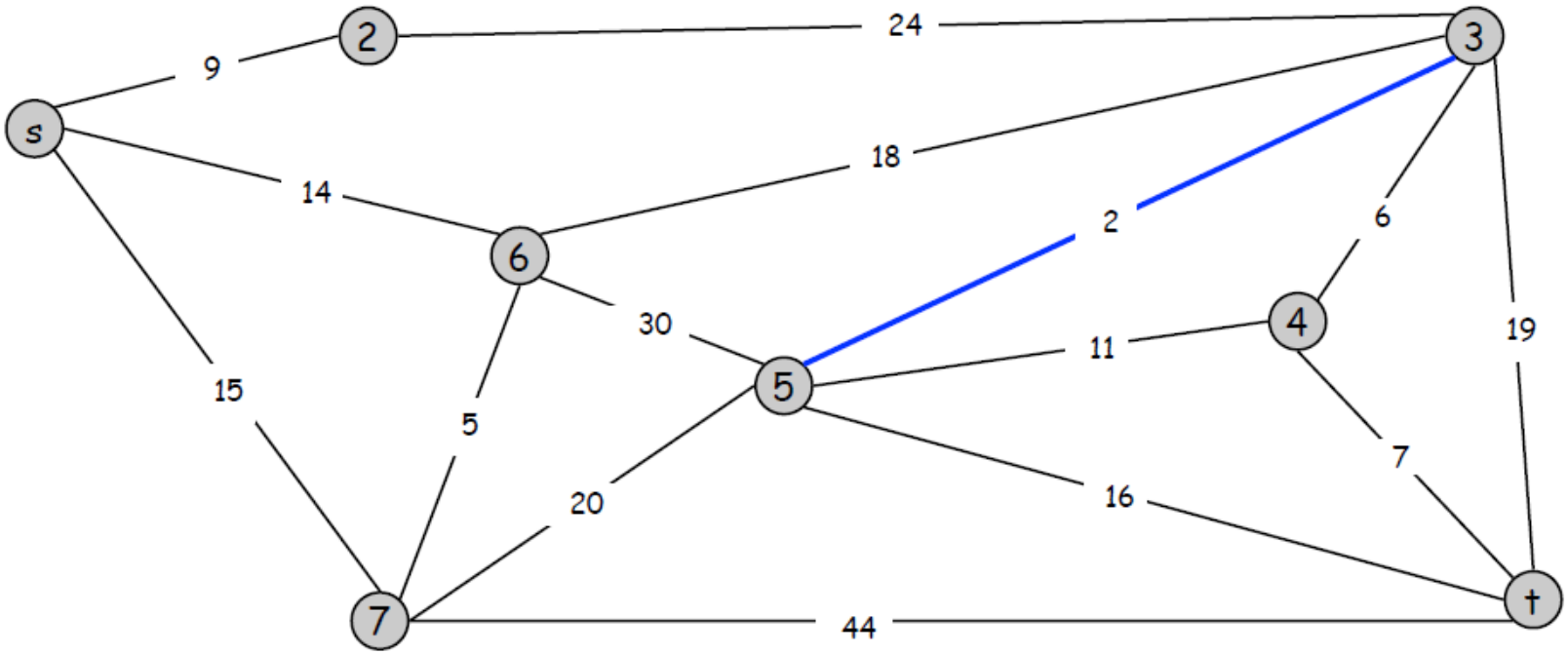


Algoritmo di Kruskal

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ.
- Considera archi in ordine crescente di costo.
- Inserisci l'arco in T se non crea un ciclo.

T={(3,5)}

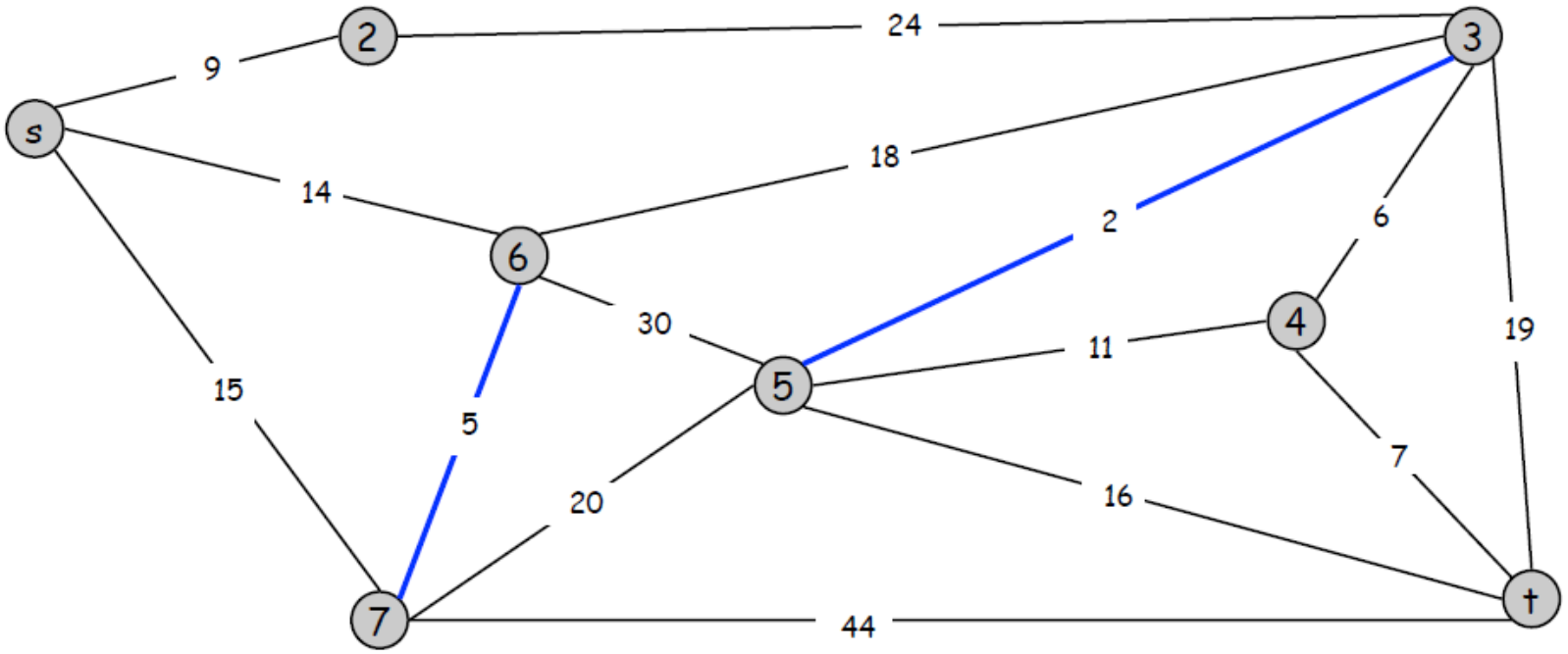


Algoritmo di Kruskal

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ.
- Considera archi in ordine crescente di costo.
- Inserisci l'arco in T se non crea un ciclo.

T={(3,5), (6,7)}

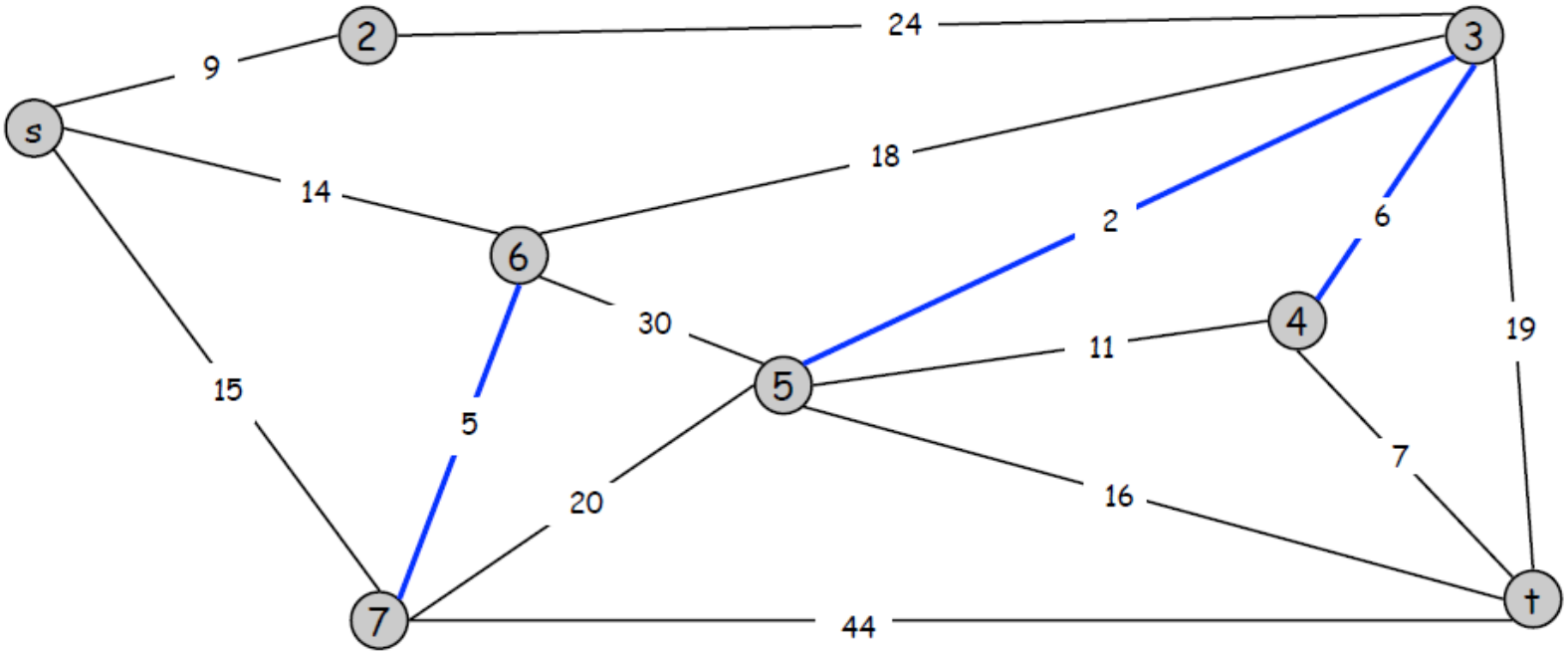


Algoritmo di Kruskal

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ.
- Considera archi in ordine crescente di costo.
- Inserisci l'arco in T se non crea un ciclo.

T={(3,5), (6,7), (3,4)}

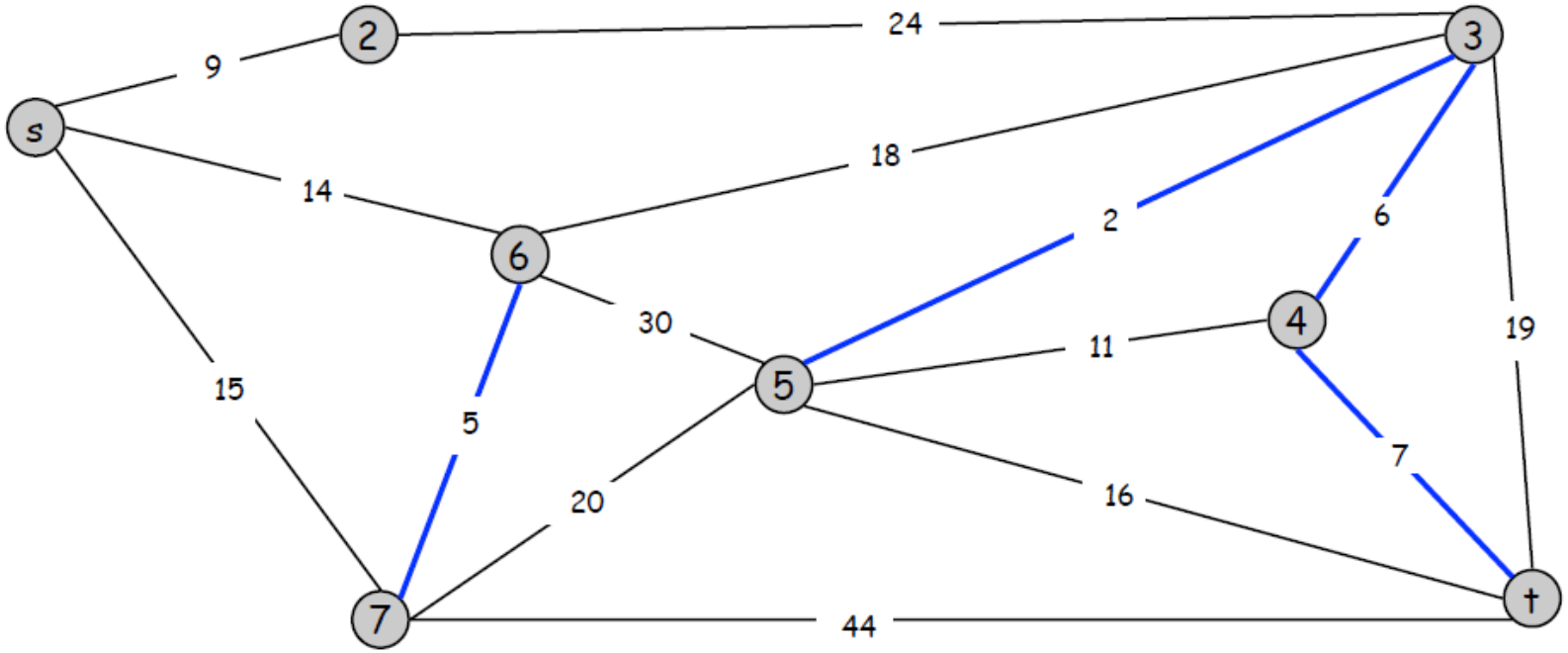


Algoritmo di Kruskal

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ.
- Considera archi in ordine crescente di costo.
- Inserisci l'arco in T se non crea un ciclo.

T={(3,5), (6,7), (3,4), (4,t)}

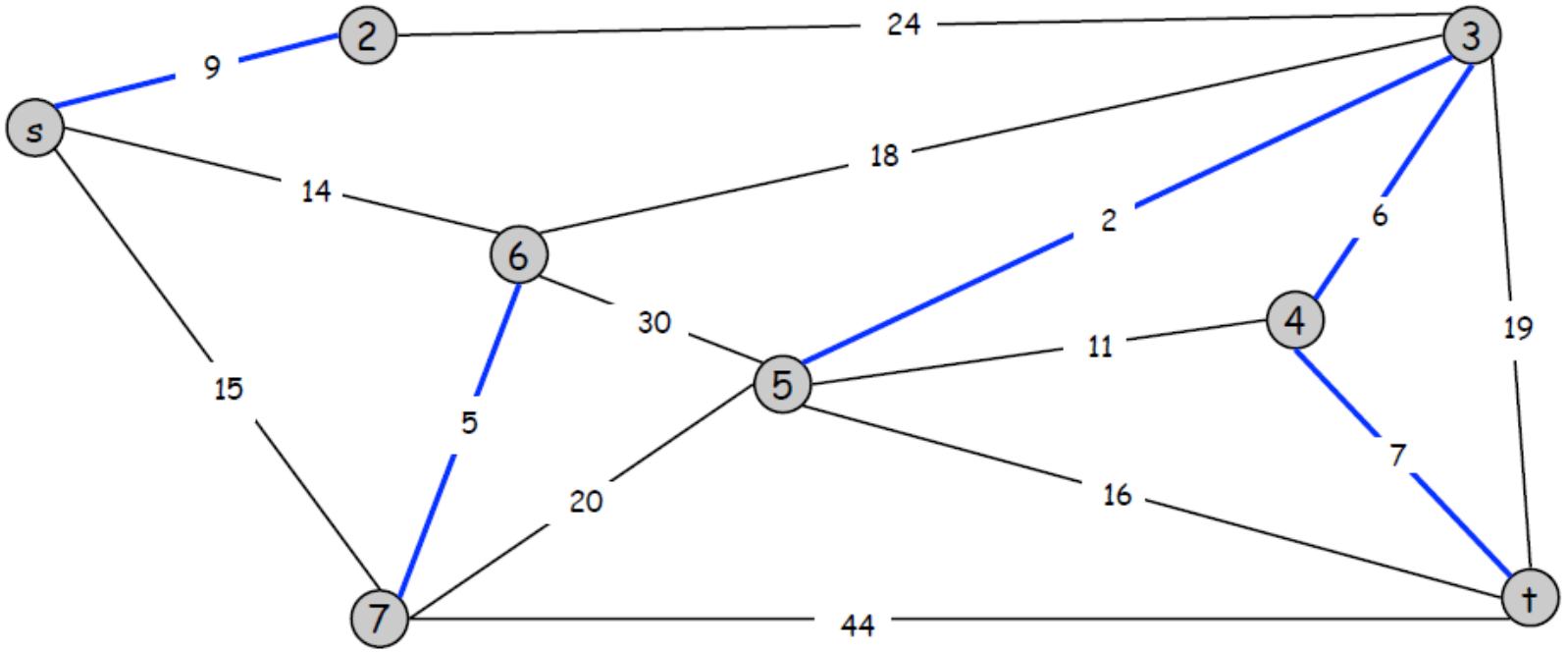


Algoritmo di Kruskal

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ.
- Considera archi in ordine crescente di costo.
- Inserisci l'arco in T se non crea un ciclo.

T={(3,5), (6,7), (3,4), (4,t), (s,2)}

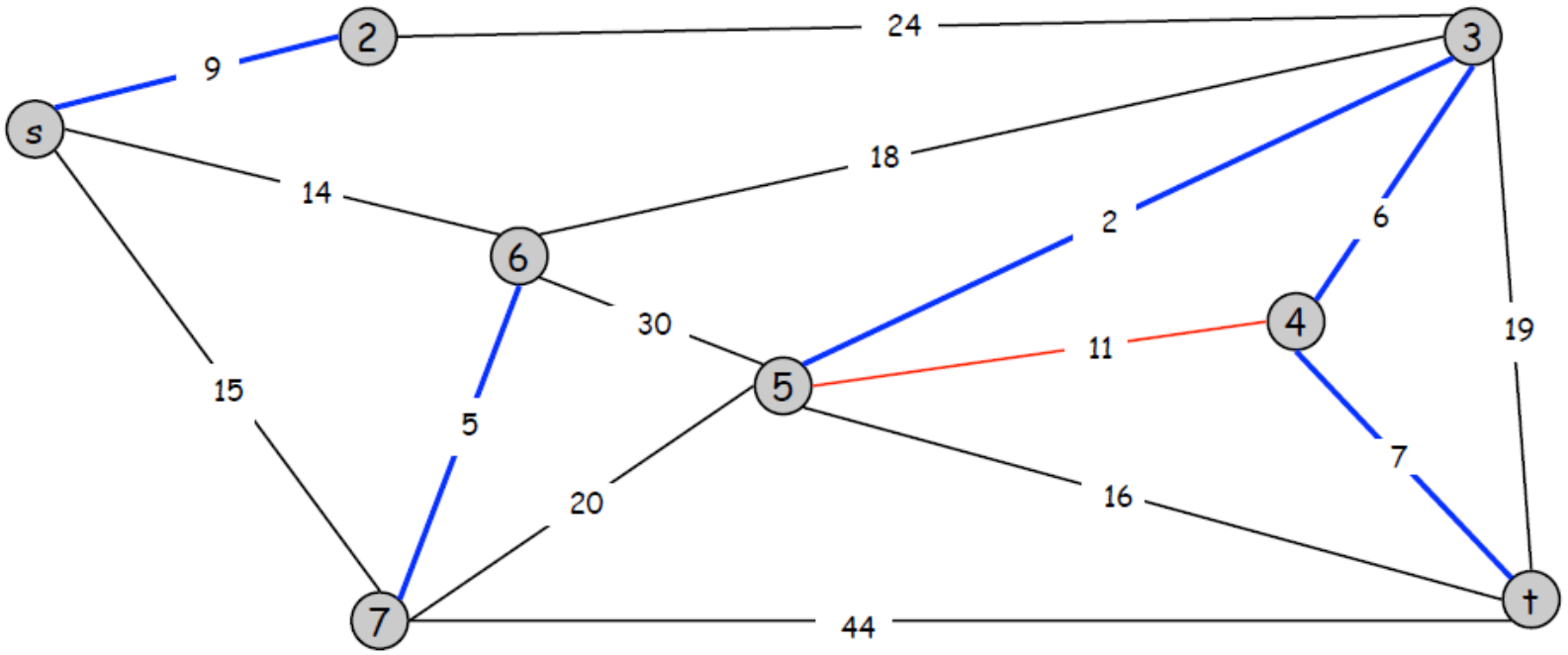


Algoritmo di Kruskal

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ.
- Considera archi in ordine crescente di costo.
- Inserisci l'arco in T se non crea un ciclo.

T={(3,5), (6,7), (3,4), (4,t), (s,2)}

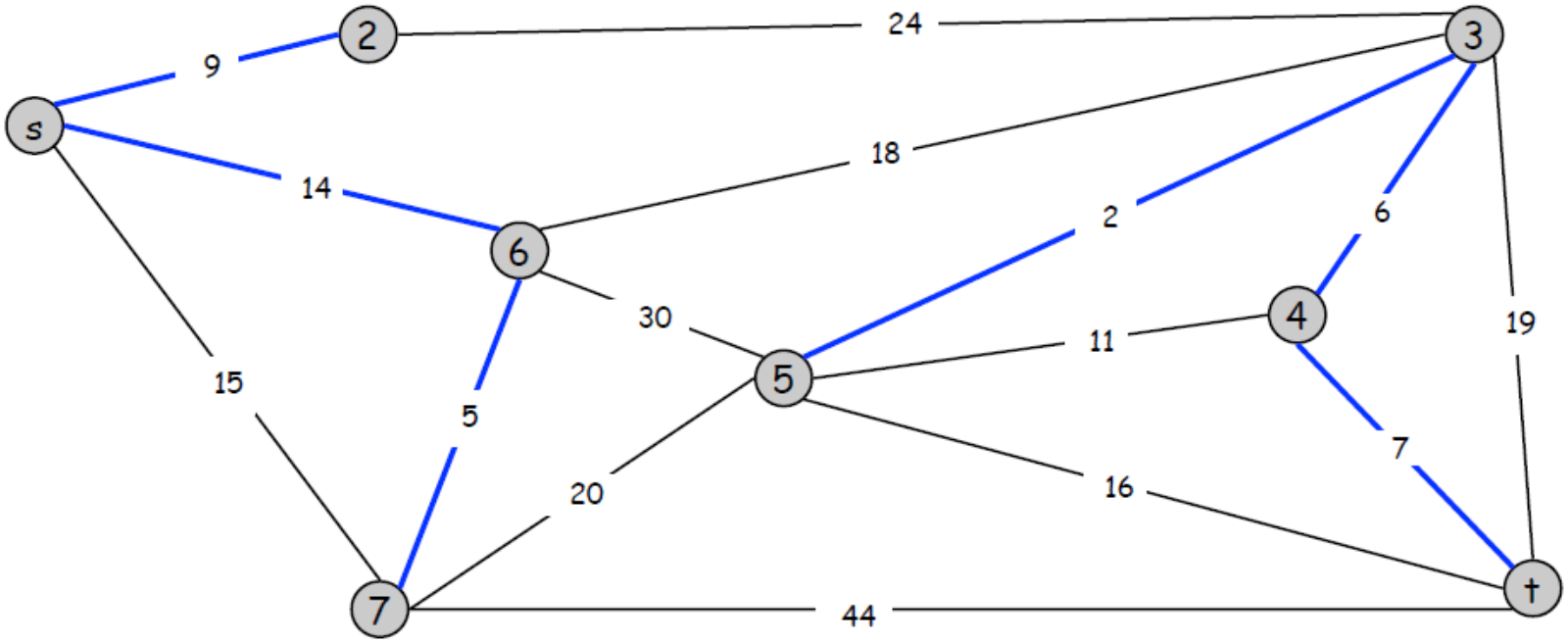


Algoritmo di Kruskal

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ.
- Considera archi in ordine crescente di costo.
- Inserisci l'arco in T se non crea un ciclo.

T={(3,5), (6,7), (3,4), (4,t), (s,2), (s,6)}

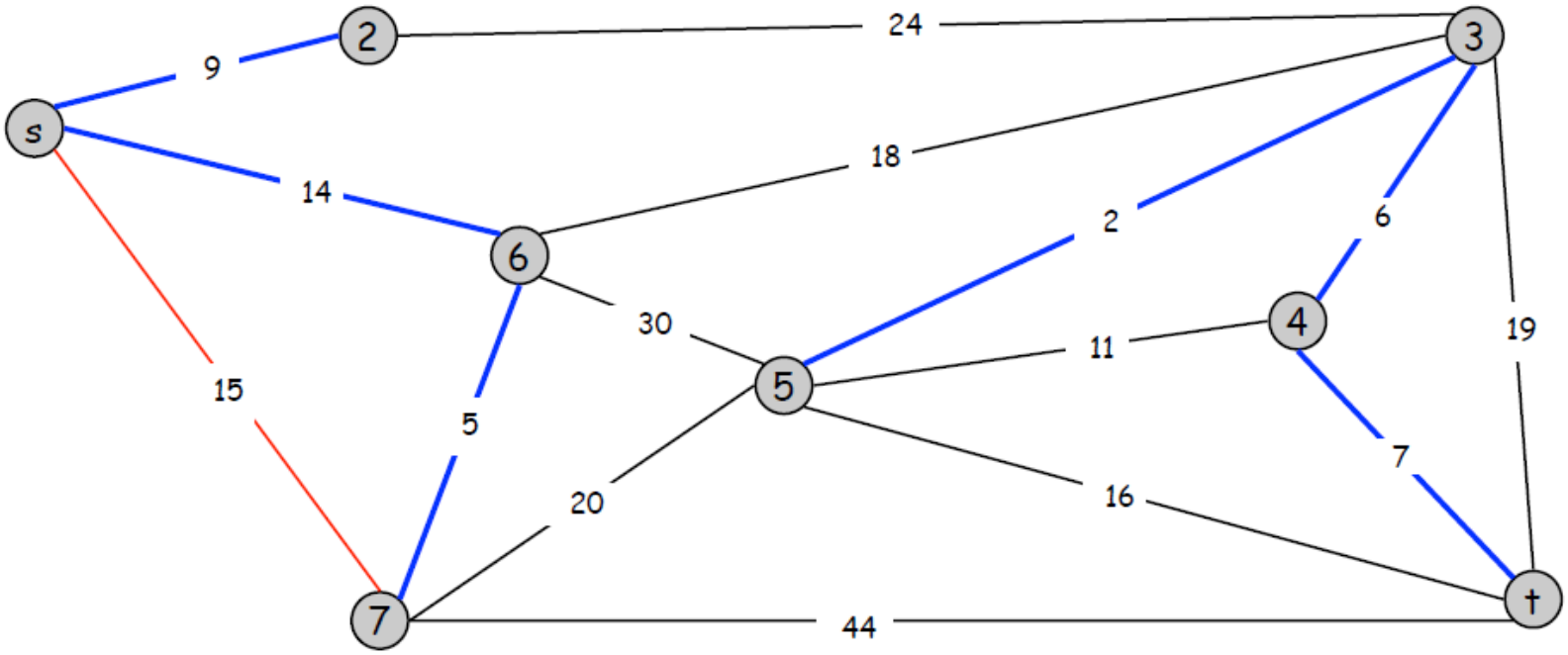


Algoritmo di Kruskal

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ.
- Considera archi in ordine crescente di costo.
- Inserisci l'arco in T se non crea un ciclo.

T={(3,5), (6,7), (3,4), (4,t), (s,2), (s,6)}

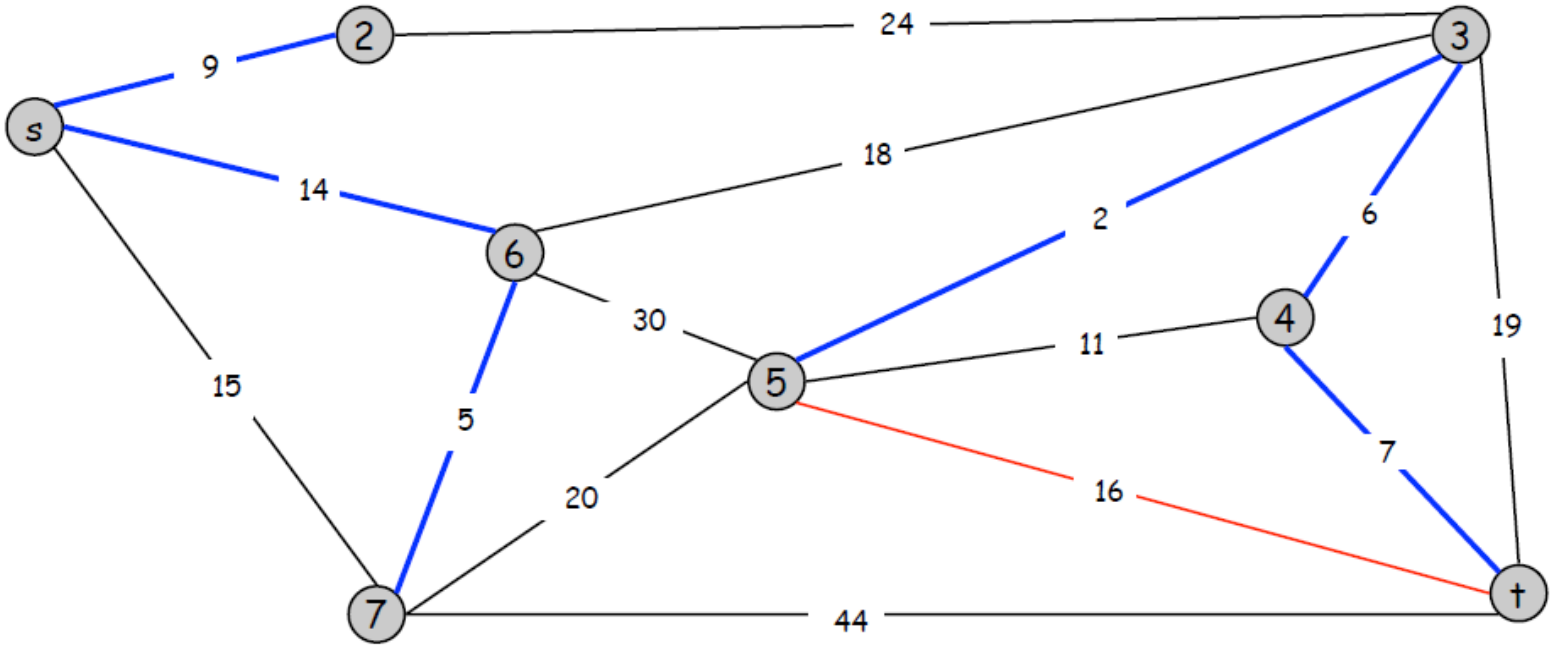


Algoritmo di Kruskal

Trovare Minimo Spanning Tree.

- Inizia con T = ϕ.
- Considera archi in ordine crescente di costo.
- Inserisci l'arco in T se non crea un ciclo.

T={(3,5), (6,7), (3,4), (4,t), (s,2), (s,6)}

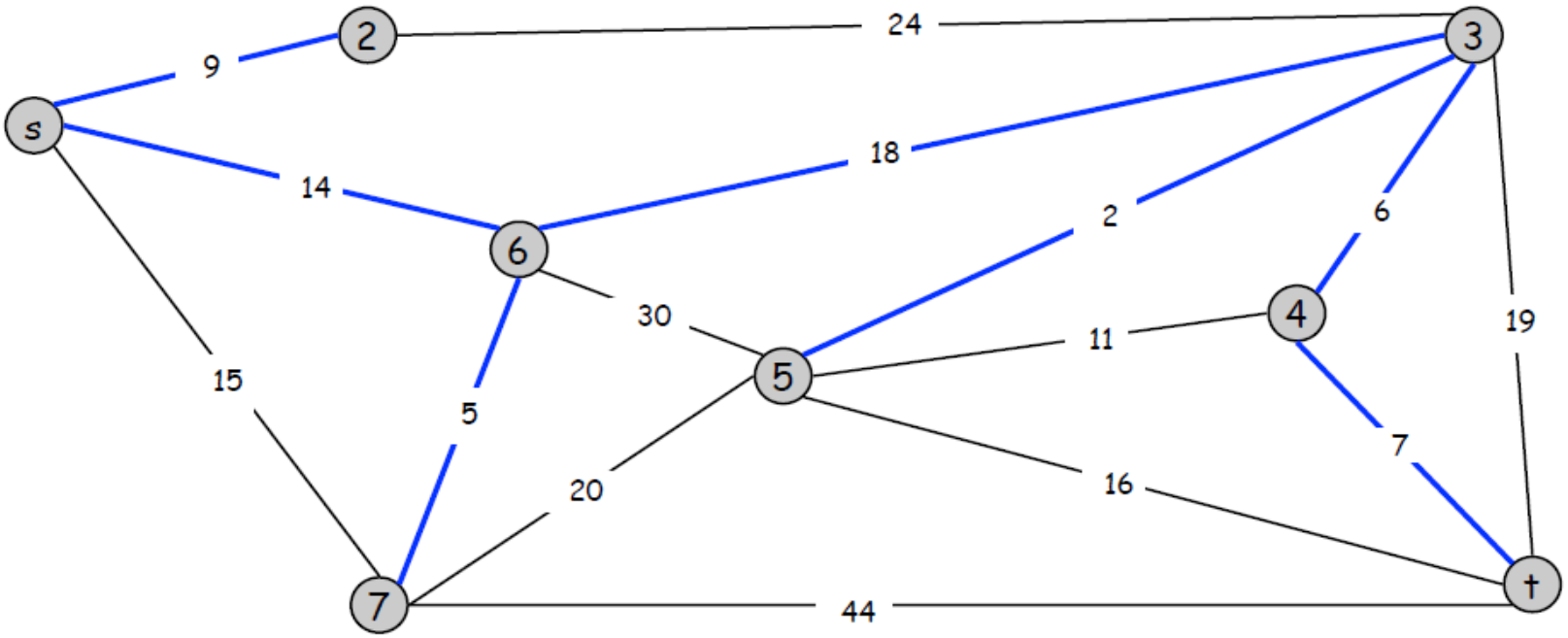


Algoritmo di Kruskal

Trovare Minimo Spanning Tree.

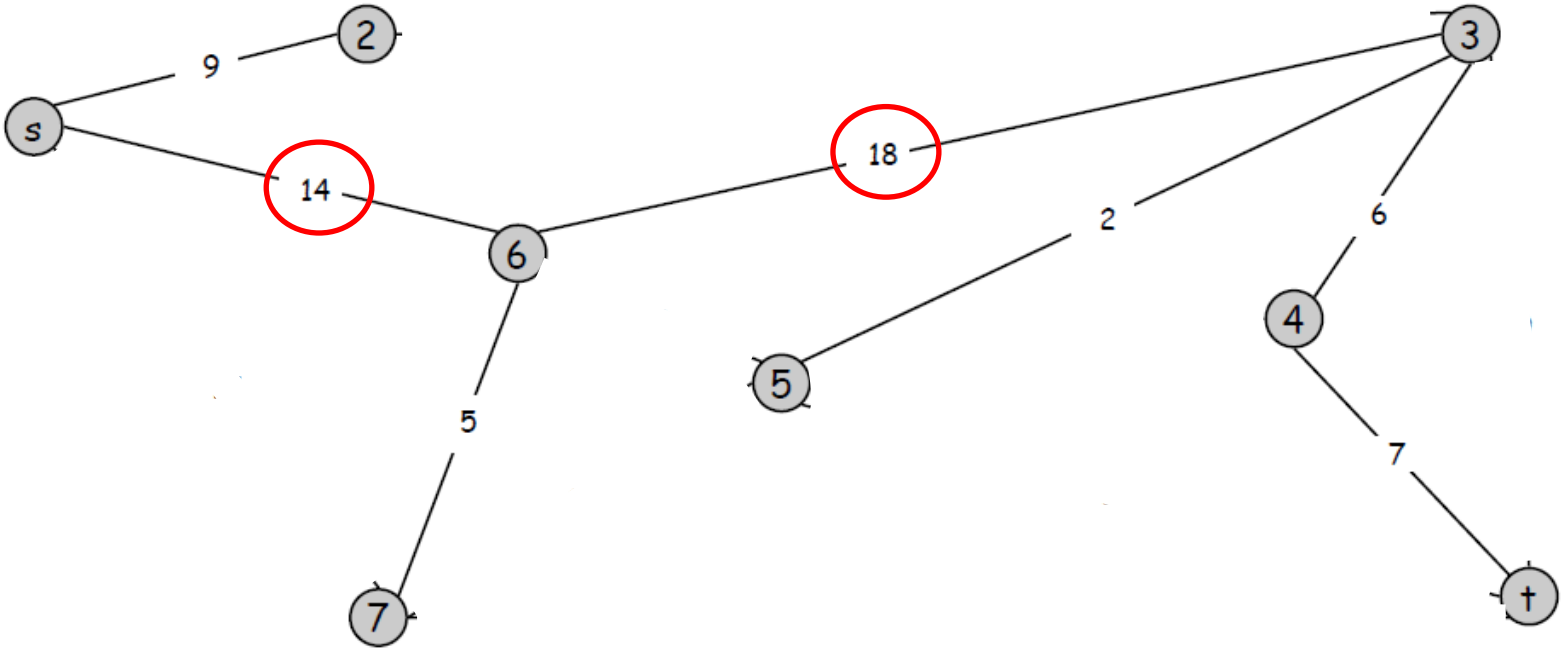
- Inizia con T = ϕ.
- Considera archi in ordine crescente di costo.
- Inserisci l'arco in T se non crea un ciclo.

T={(3,5), (6,7), (3,4), (4,t), (s,2), (s,6), (3,6)} è MST di costo 9+14+5+18+2+6+7= 61



Reverse-Delete: an example

Reverse-Delete algorithm. Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.



Confronto fra algoritmi di Prim e di Kruskal

Durante l'esecuzione:

l'algoritmo di Prim mantiene un singolo albero

l'algoritmo di Kruskal un insieme di alberi (foresta).

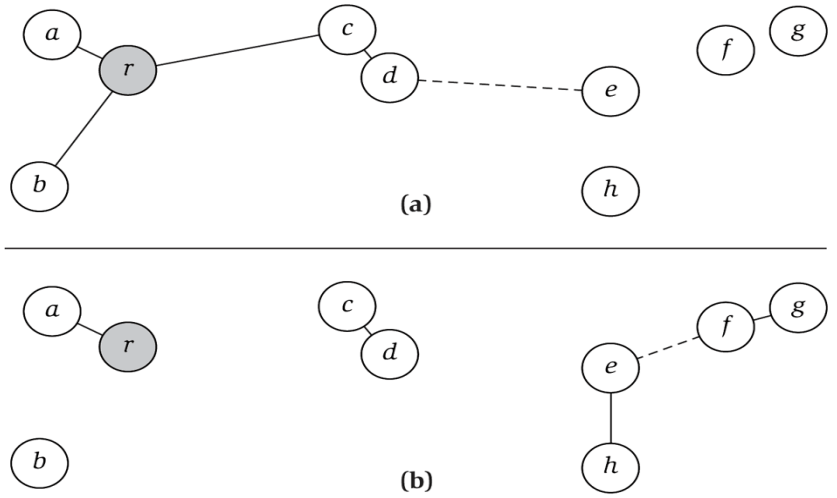


Figure 4.9 Sample run of the Minimum Spanning Tree Algorithms of (a) Prim and (b) Kruskal, on the same input. The first 4 edges added to the spanning tree are indicated by solid lines; the next edge to be added is a dashed line.

Gli alberi restituiti sono gli stessi?

Se i costi sono distinti, il MST è unico. (es. 8, p.192: usa proprietà seguenti).

In generale no.

Osservazioni (per il seguito).

In un albero: se tolgo un arco disconnetto; se aggiungo un arco fra due suoi vertici creo un ciclo.

In un grafo connesso: se, togliendo un arco, non disconnetto, l'arco apparteneva ad un ciclo.

Correctness

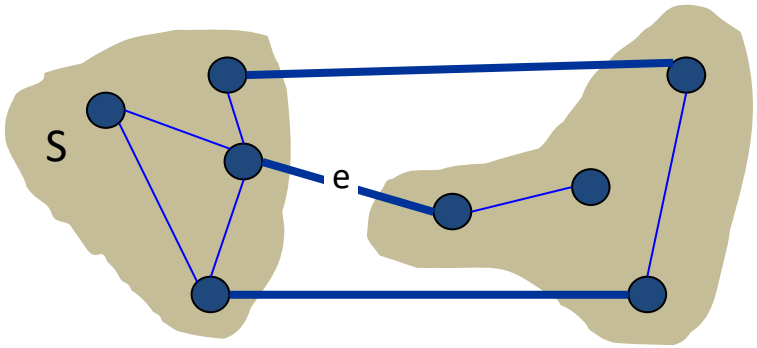
Correctness of Prim's and Kruskal's algorithms is based on Cut property. Correctness of Reverse-Delete on Cycle property.

Simplifying assumption. All edge costs c_e are distinct (hence the MST is unique).

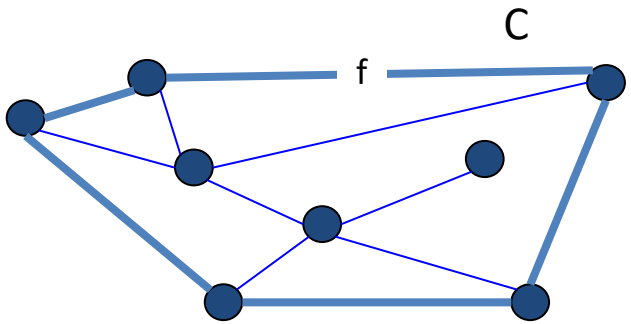
Cut property. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the MST contains e.

(In altri testi: l'arco leggero che attraversa il taglio è sicuro per S)

Cycle property. Let C be any cycle, and let f be the max cost edge belonging to C. Then the MST does not contain f.



e is in the MST



f is not in the MST

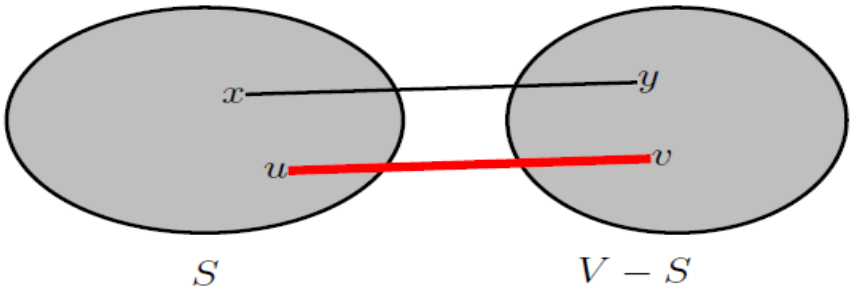
Proof of Cut Property

Simplifying assumption. All edge costs c_e are distinct.

Cut property. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the MST contains e.

Pf. (exchange argument)

Supponiamo per assurdo che ciò non sia vero e sia T il MST che NON contiene e=(u,v).



I vertici u e v saranno connessi da un (unico) cammino all'interno di T; su tale cammino esiste un arco a = (x,y), diverso da (u,v), con un estremo in S e uno in V\S. Per ipotesi c(e) < c(a).
Consideriamo adesso T' = T ∪ { e } \ { a}.
Aggiungendo l'arco e a T si forma un ciclo; togliendo l'arco a si elimina tale ciclo cosicchè T' è ancora aciclico. Inoltre anche T' connette tutti i vertici di V.

Quindi T' è un albero di ricoprimento. Inoltre il suo costo sarebbe costo(T') < costo (T), contro l'ipotesi di minimalità di T.

Proof of Cycle Property

Simplifying assumption. All edge costs c_e are distinct.

Cycle property. Let C be any cycle in G, and let *e* be the max cost edge belonging to C. Then the MST T* does not contain *e*.

Pf. (exchange argument)

Suppose *e* belongs to T*, and let's see what happens.

Deleting *e* from T* disconnects T* and partitions V in S and V\S.

In the cycle C there exists another edge, say *e*'=(u',v'), with u' in S and v' in V\S.

T' = T* $\cup$ { *e*' } - { *e*} is also a spanning tree.

Since $c_{e'} < c_e$ then cost(T') < cost(T*).

This is a contradiction. ▪

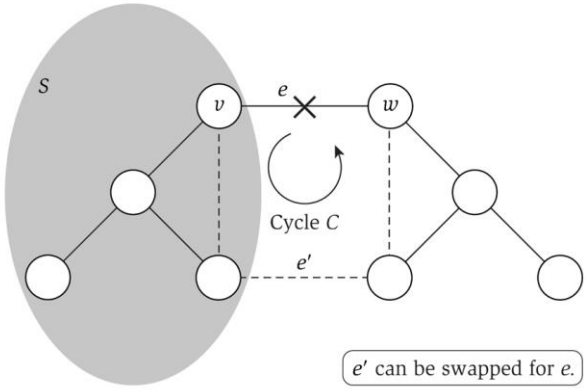


Figure 4.11 Swapping the edge e' for the edge e in the spanning tree T, as described in the proof of (4.20).

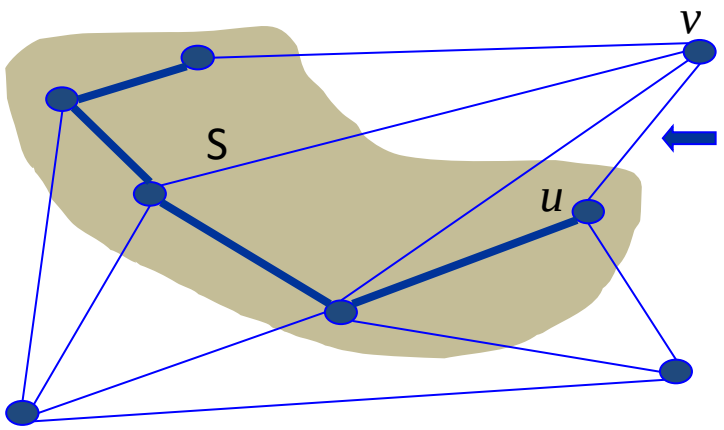
Prim's Algorithm: Proof of Correctness

Prim's algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959]

Prim's algorithm. Start with some root node s and greedily grow a tree T from s outward. At each step, add the cheapest edge e to T that has exactly one endpoint in T.

Pf. Che l'algoritmo di Prim produca un albero è ovvio, visto che aggiunge archi solo da nodi già tra di loro connessi in S a nuovi nodi "fuori" di S (quindi non crea cicli). Inoltre, ad ogni passo aggiunge a T l'arco di minimo costo che ha un estremo u in S (insieme dei nodi su cui un albero ricoprente parziale è stato già costruito) ad un nodo $v \in V - S$.

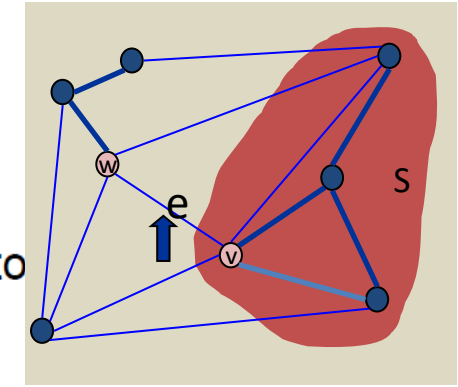
Dalla proprietà prima vista, tale arco appartiene ad ogni MST del grafo (cioè, di nuovo l'algoritmo non inserisce mai archi che non appartengono a MST, quindi produce effettivamente un MST).



L'algoritmo di Kruskal produce un MST

Aggiungi a T uno ad uno gli archi del grafo, in ordine di costo crescente, saltando gli archi che creano cicli con gli archi già aggiunti.

- Sia $e = (v, w)$ un generico arco inserito in T dall'algoritmo di Kruskal, e sia S l'insieme di tutti i nodi connessi a v attraverso un cammino, al momento appena prima di aggiungere (v, w) a T.
- Ovviamente vale che $v \in S$, mentre $w \notin S$, altrimenti l'arco (v, w) creerebbe un ciclo.
- Inoltre, negli istanti precedenti l'algoritmo non ha incontrato nessun arco da nodi in S a nodi in $V - S$, altrimenti un tale arco sarebbe stato aggiunto, visto che non creava cicli.
- Pertanto l'arco (v, w) è il primo arco da S a $V - S$ che l'algoritmo incontra, ovvero è l'arco di minor costo da S a $V - S$ che, abbiamo visto appartiene ad ogni MST.
- Ci rimane da mostrare che l'output dell'algoritmo di Kruskal è un albero



Facciamolo:

- Sicuramente, per costruzione, l'output (V, T) non contiene cicli.
- Potrebbe (V, T) non essere connesso? Ovvero potrebbe esistere un $\emptyset \neq S \subset V$ per cui in T non esiste alcun arco da S a $V - S$?
- Sicuramente no! Infatti, poichè il grafo G è connesso, un tale arco e esiste sicuramente in G e poichè l'algoritmo di Kruskal esamina tutti gli archi di G, prima o poi incontrerà tale arco e e lo inserirà, visto che non crea cicli.

Correttezza Reverse-Delete

Reverse-Delete algorithm. Start with T = E. Consider edges in descending order of cost. Delete edge e from T unless doing so would disconnect T.

Prova. Sia *e* un arco eliminato dall'algoritmo. Poiché *e* non disconnette il grafo, appartiene ad un ciclo. Di questo ciclo è l'arco di costo massimo (per la scelta effettuata). Per la proprietà del ciclo *e* non appartiene a nessun MST.

Alla fine (V,T) sarà un albero: connesso per costruzione dell'algoritmo; aciclico, altrimenti l'algoritmo non si sarebbe arrestato.

Implementation: Prim's Algorithm

Implementation. Use a priority queue Q "a la Dijkstra".

Maintain set of explored nodes S and set of unexplored nodes Q.

For each unexplored node v in Q, maintain as priority an attachment cost

a[v] = cost of cheapest edge v to a node in S.

For each node v, π[v] will be the parent of v in the MST.

```
Prim(G,c,s) {
foreach (v∈V) {a[v]← ∞; π[v]= NULL}; a[s]←0;
Initialize an empty priority queue Q
foreach (v ∈ V) insert v onto Q
Initialize set of explored nodes S ← ϕ

 while (Q is not empty) {
      u ← delete min element from Q
      S ← S ∪ { u }
      foreach (edge e = (u, v) incident to u)
          if ((v ∉ S) and (c_e < a[v]))
           {decrease priority a[v] to c_e; π[v]=u;}
}
```

Prim's Algorithm: Implementation

```
Prim(G,c,s) {
foreach (v∈V) {a[v]← ∞; π[v]= NULL}; a[s]←0;
Initialize an empty priority queue Q
foreach (v ∈ V) insert v onto Q
Initialize set of explored nodes S ← ϕ

 while (Q is not empty) {
      u ← delete min element from Q
      S ← S ∪ { u }
      foreach (edge e = (u, v) incident to u)
          if ((v ∉ S) and (c_e < a[v]))
           {decrease priority a[v] to c_e; π[v]=u;}
}
```

Priority Queue

PQ Operation	Prim	Array	Binary heap	d-way Heap	Fib heap †
Insert	n	1	log n	$d \log_d n$	1
ExtractMin	n	n	log n	$d \log_d n$	log n
Decr		1	log n	$\log_d n$	1
I		1	1	1	1
Total		n^2	m log n	$m \log_{m/n} n$	m + n log n

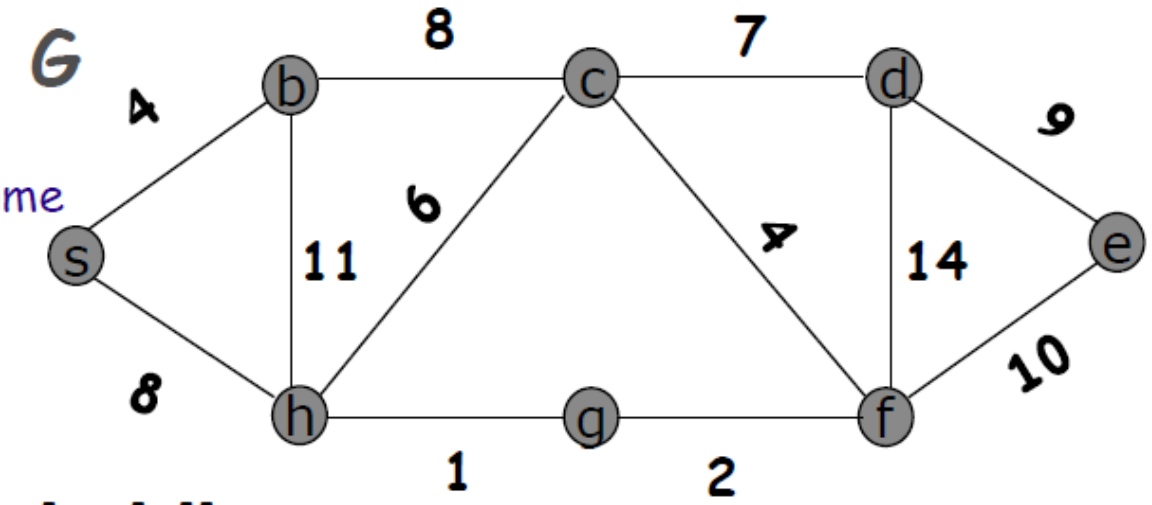
Quale delle due è preferibile?

† Individual ops are amortized bounds

Un esempio

In questo esempio, per ciascun nodo u manteniamo anche un campo $\pi[u]$ che alla fine è uguale al padre di u nello MST

s = nodo scelto come radice



[u,a[u]]

$Q = \{[s, 0],[b, \infty],[c, \infty],[d, \infty],[e, \infty],[f, \infty],[g, \infty],[h, \infty]\}$ $S = \{\ \}$

Si estrae s da Q e si aggiornano i campi a e π dei nodi adiacenti ad s

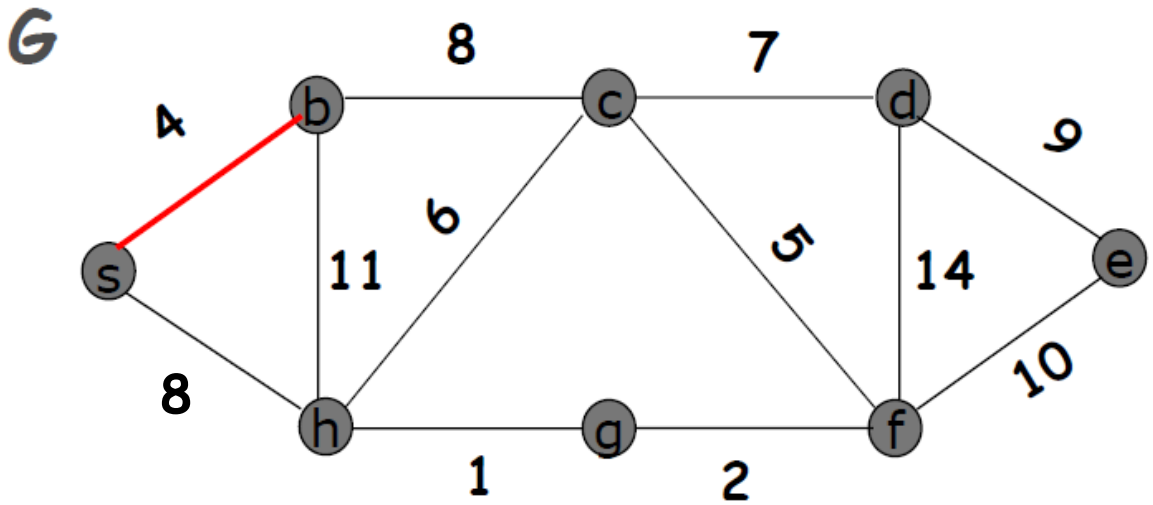
$S = \{s\}$

$Q = \{[b, 4],[c, \infty],[d, \infty],[e, \infty],[f, \infty],[g, \infty],[h, 8]\}$

$\pi(b)=s$

$\pi(h)=s$

Un esempio



$Q = \{[b,4],[c,\infty],[d,\infty],[e,\infty],[f,\infty],[g,\infty],[h,8]\}$ $S = \{s\}$

- Si estrae b da Q.

$S = \{s,b\}$

- Si aggiornano i campi a dei nodi adiacenti a b che si trovano in Q

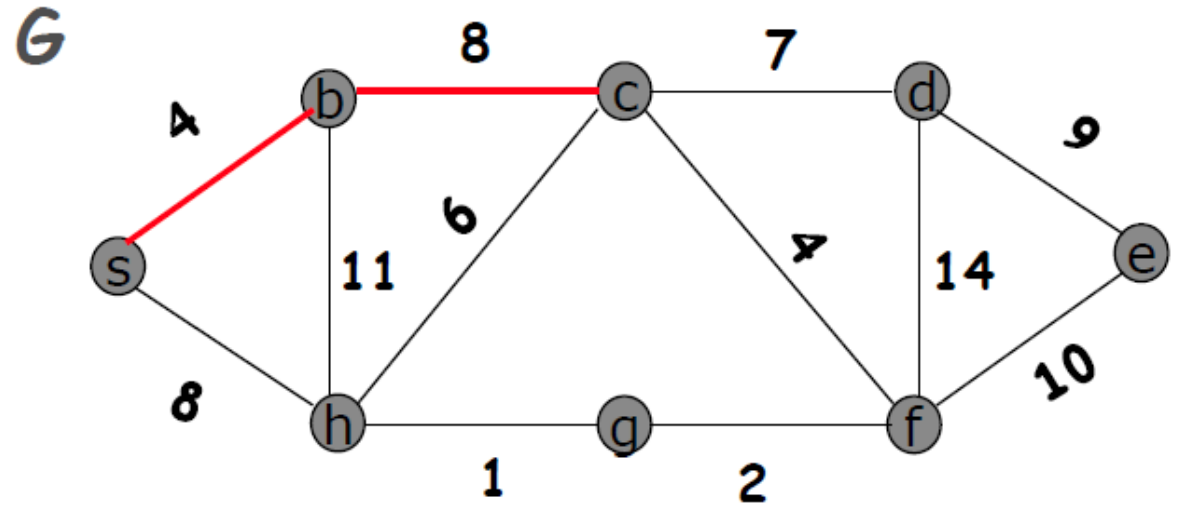
$\pi(b)=s$

$\pi(h)=s$

$Q = \{[c,8],[d,\infty],[e,\infty],[f,\infty],[g,\infty],[h,8]\}$

$\pi(c)=b$

Un esempio



$Q = \{[c, 8],[d, \infty],[e, \infty],[f, \infty],[g, \infty],[h, 8]\}$ $S = \{s,b\}$

- A seconda dell'implementazione di Q si estrae c oppure h da Q.

$S = \{s,b,c\}$

- Assumiamo che venga estratto c: si aggiornano i campi a e π dei nodi adiacenti a c che si trovano in Q

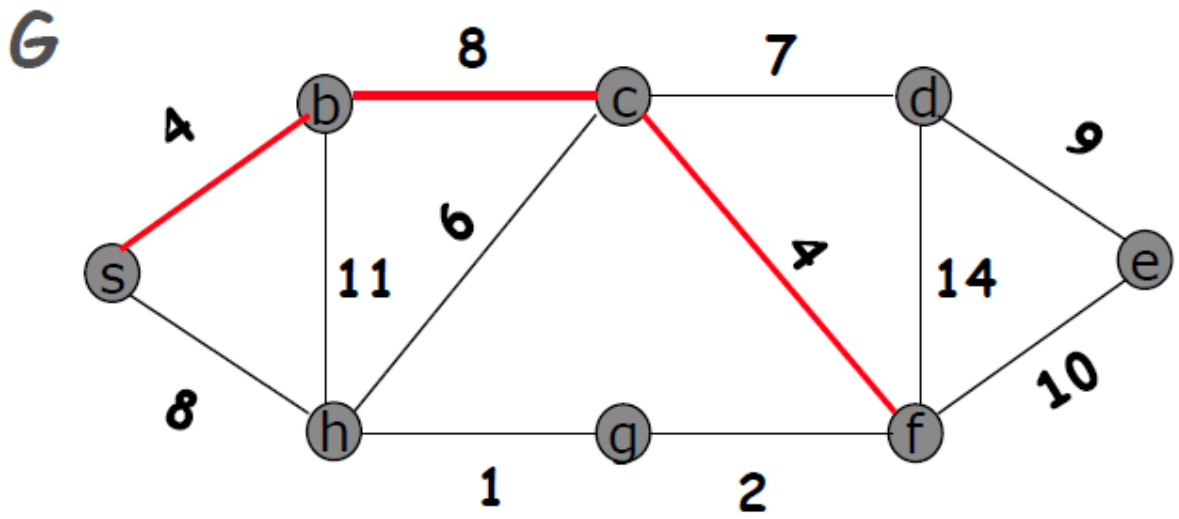
$Q = \{[d, 7],[e, \infty],[f, 4],[g, \infty],[h, 6]\}$

$\pi(b)=s, \pi(c)=b$

$\pi(h)=c, \pi(d)=c$

$\pi(f)=c$

Un esempio



$Q = \{[d, 7],[e, \infty],[f, 4],[g, \infty],[h, 6]\}$ $S = \{s,b,c\}$

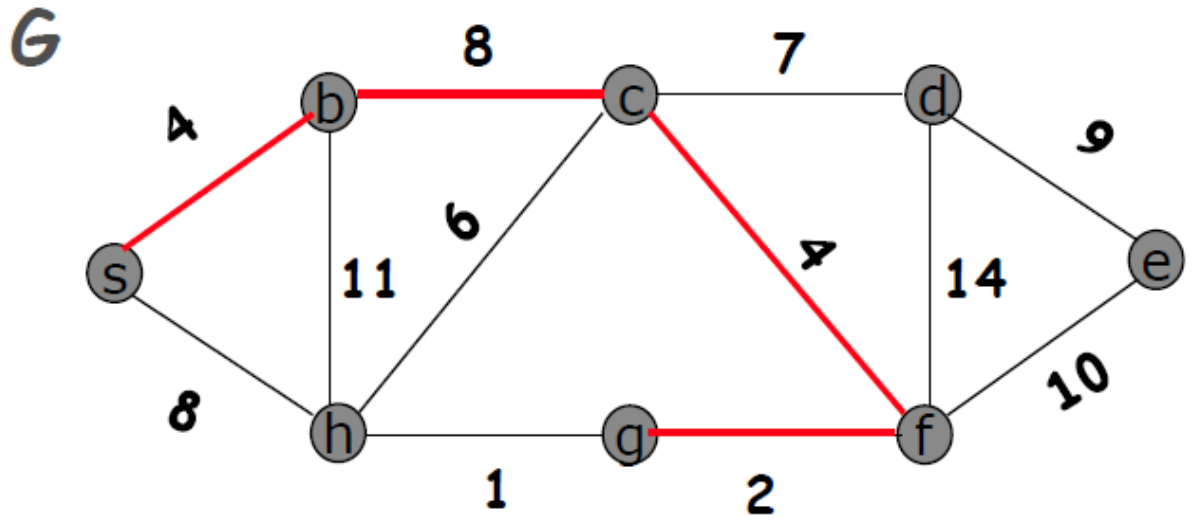
Si estrae f da Q e si aggiornano i campi a e π dei nodi adiacenti a f che si trovano in Q

$S = \{s,b,c,f\}$

$Q = \{[d, 7],[e, 10],[g, 2],[h, 6]\}$

$\pi(b)=s, \pi(c)=b, \pi(f)=c,$
$\pi(h)=c, \pi(d)=c, \pi(g)=f,$
$\pi(e)=f$

Un esempio



Q = {[d, 7],[e, 10],[g, 2],[h, 6]}

S = {s,b,c,f}

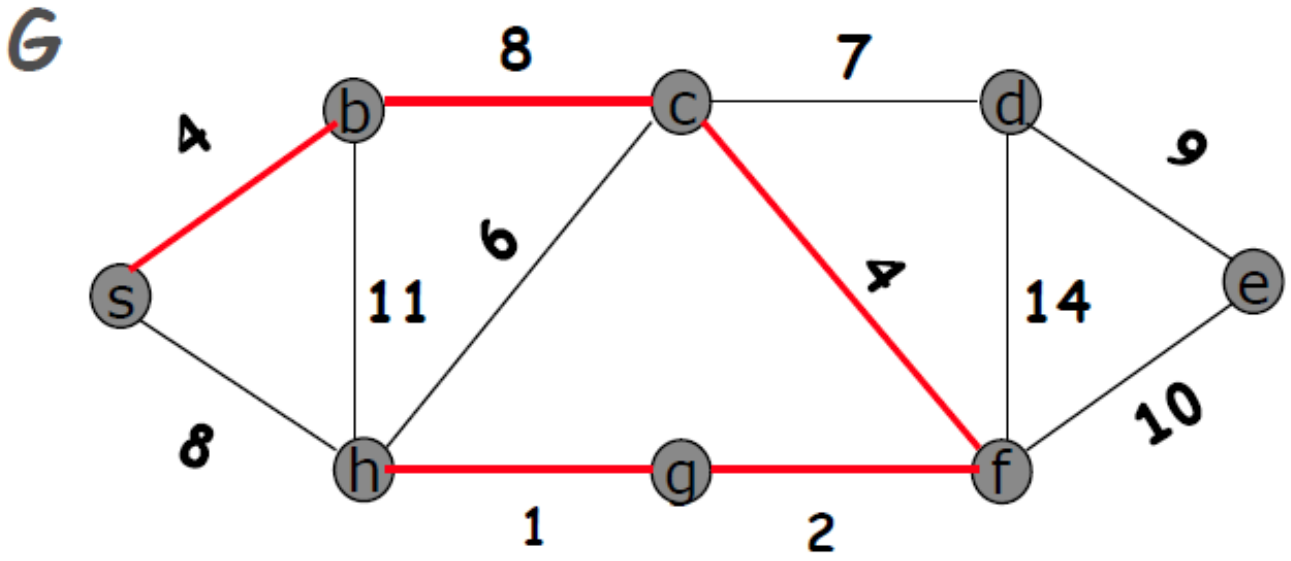
Si estrae g da Q e si aggiornano i campi a e π dei nodi adiacenti a g che si trovano in Q

S = {s,b,c,f, g}

Q = {[d, 7],[e,10],[h, 1]}

π(b)=s,π(c)=b,π(f)=c,π(g)=f,
π(d)=c, π(e)=f, π(h)=g

Un esempio



$Q = \{[d, 7],[e, 10],[h, 1]\}$

$S = \{s,b,c,f,g\}$

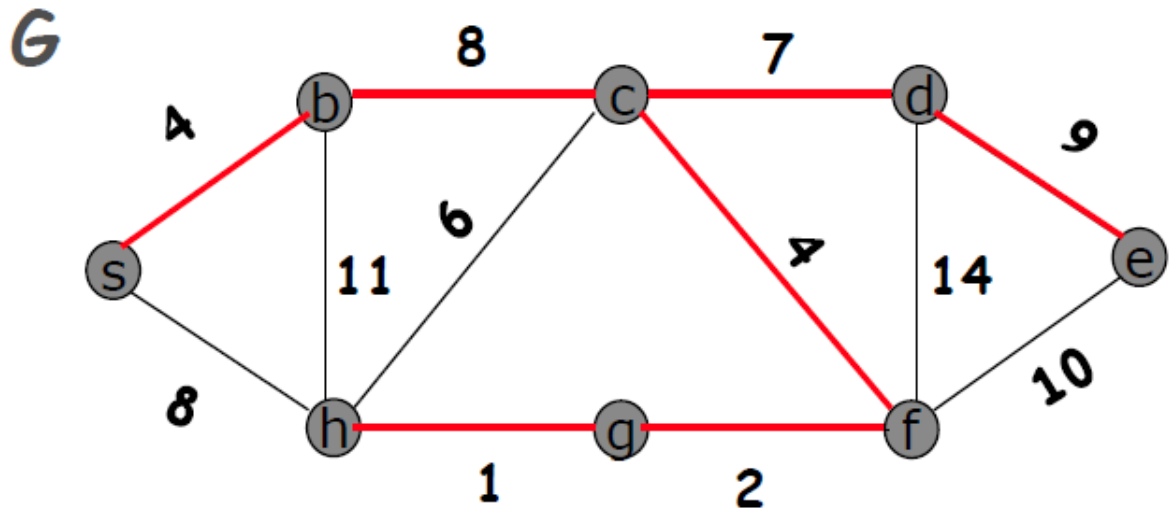
Si estrae h da Q e si aggiornano i campi a e π dei nodi adiacenti a h che si trovano in Q

$S = \{s,b,c,f,g,h\}$

$Q = \{[d, 7],[e, 10]\}$

$\pi(b)=s, \pi(c)=b, \pi(f)=c, \pi(g)=f, \pi(h)=g,$
$\pi(d)=c, \pi(e)=f$

Un esempio



Q = {[d, 7],[e, 10]}

S = {s,b,c,f,g, h}

Si estrae d da Q e si aggiorna il campo a e π di e

S = {s,b,c,f,g,h,d}

Q = {[e, 9]}

Si estrae e da Q

π(b)=s,π(c)=b,π(f)=c,π(g)=f,π(h)=g,
π(d)=c, π(e)=d

Q = { } e l'algoritmo termina

Esercizio

Eseguire l'algoritmo di Prim sul grafo G dell'esempio precedente, ma selezionando come terzo vertice h anziché c.

L'albero MST così ottenuto è lo stesso di quello nella slide precedente?