Minimum Spanning Tree

Jocelyne Elias
https://www.unibo.it/sitoweb/jocelyne.elias
Moreno Marzolla
https://www.moreno.marzolla.name/

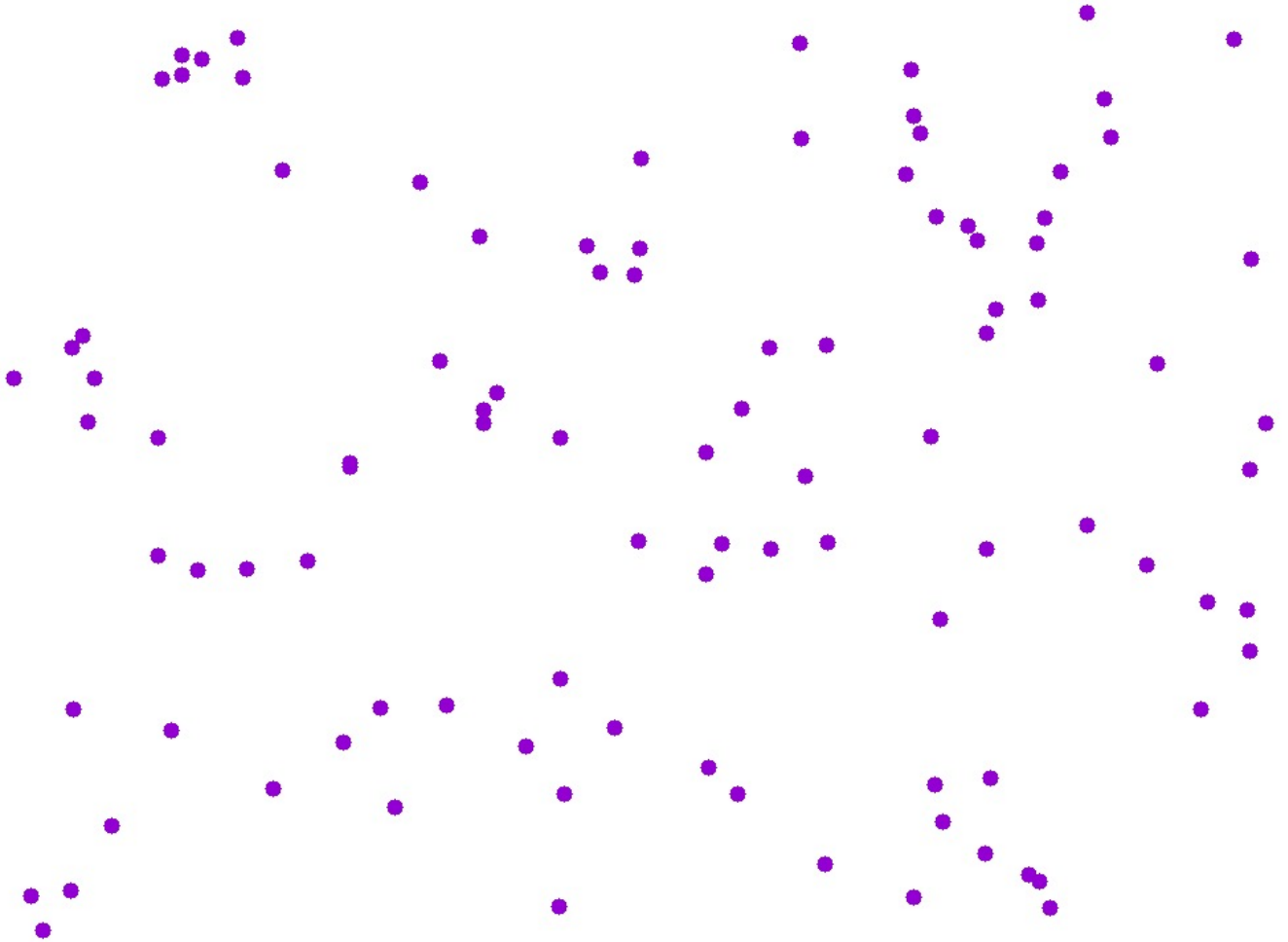
Dipartimento di Informatica—Scienza e Ingegneria (DISI)
Università di Bologna

Introduzione

- Un problema di notevole importanza:
  - determinare come interconnettere diversi elementi fra loro considerando e minimizzando certi vincoli sulle connessioni
- Esempio classico:
  - progettazione dei circuiti elettronici dove si vuole minimizzare la quantità di filo elettrico per collegare fra loro i diversi componenti
- Questo problema prende il nome di:
  - albero di copertura (di peso) minimo
  - albero di connessione (di peso) minimo
  - Minimum Spanning Tree (MST)

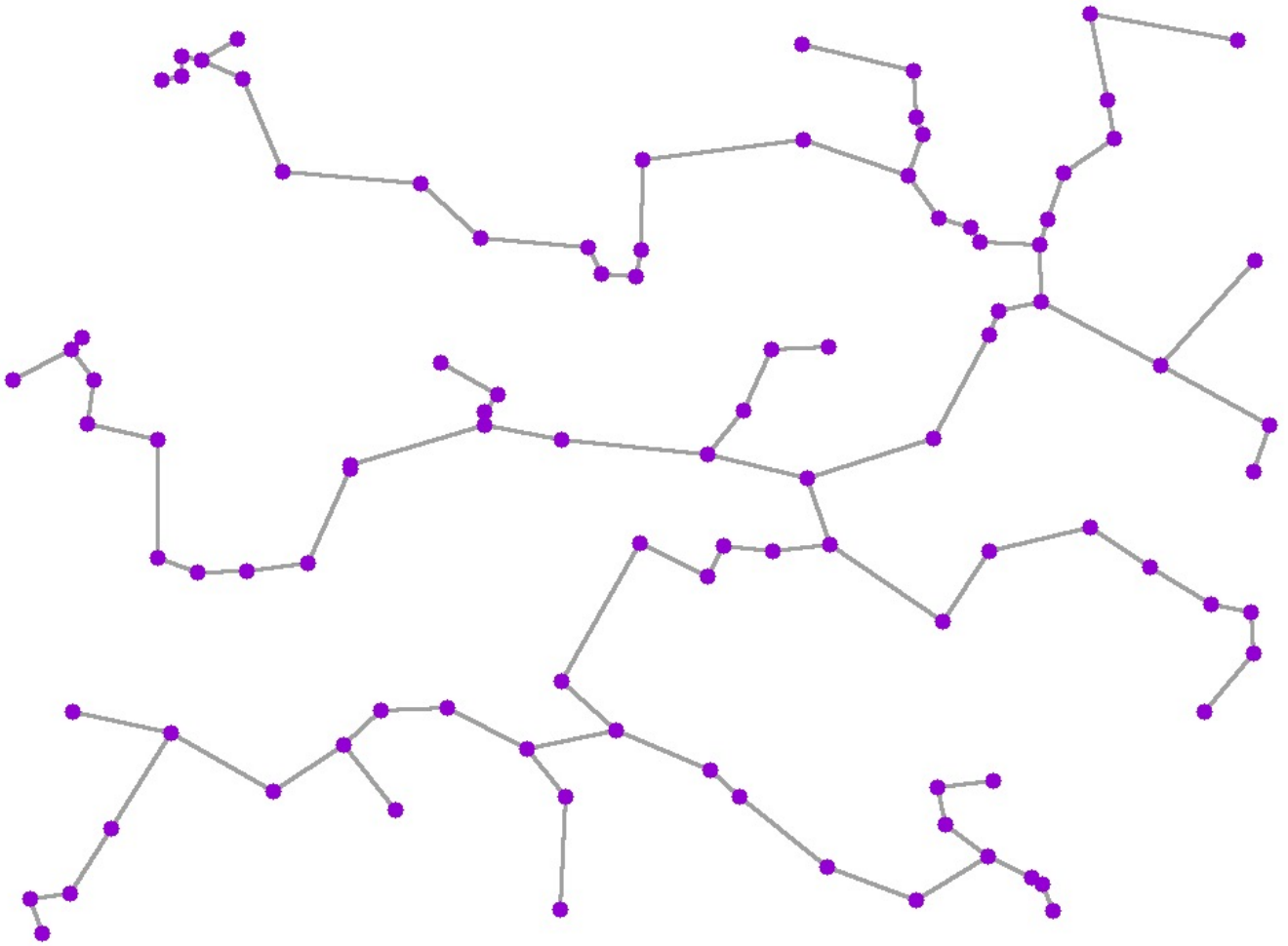
Esempio

- *n* pin su un circuito stampato da collegare con una traccia di lunghezza minima



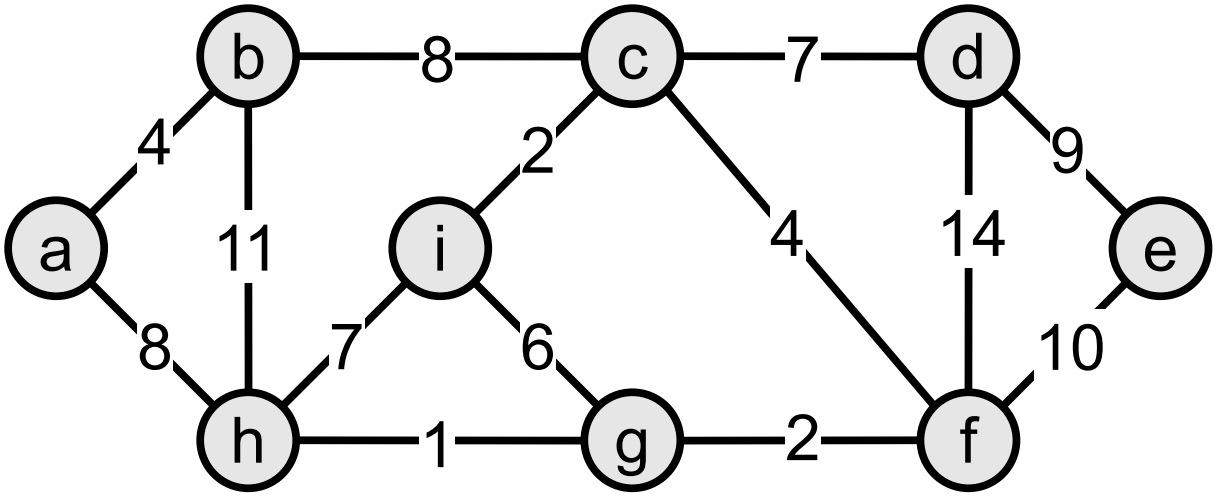
Esempio

- *n* pin su un circuito stampato da collegare con una traccia di lunghezza minima



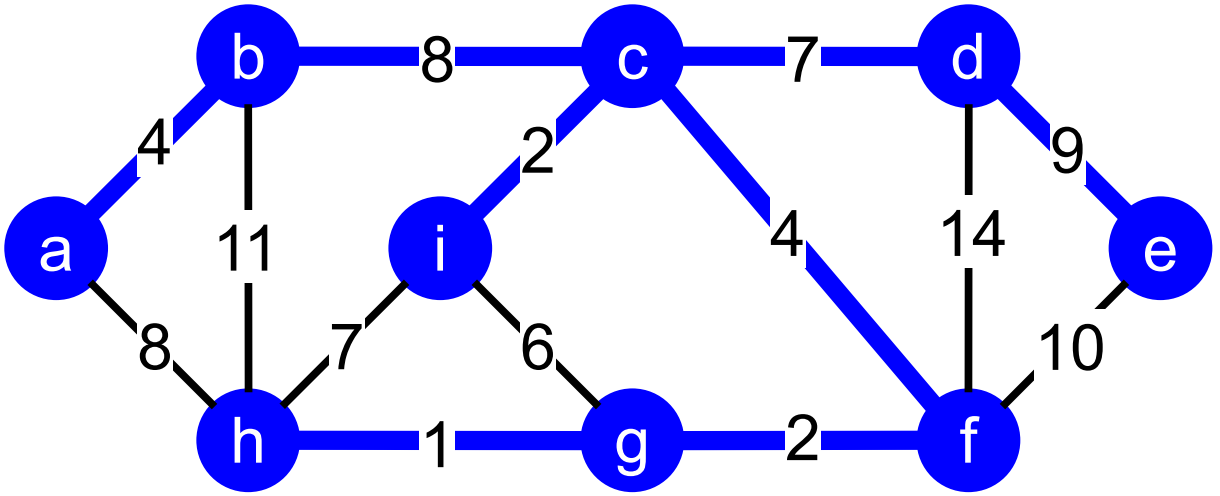
Definizione del problema

- Input:
  - $G = (V, E)$ un grafo non orientato e connesso
  - $w: V \times V \rightarrow R$ una funzione peso
    - se {u, v} $\in$ E, allora $w(u, v)$ è il peso dell'arco {u, v}
    - se {u, v} $\notin$ E, allora $w(u, v) = \infty$
  - Poiché G non è orientato, $w(u, v) = w(v, u)$



Definizione del problema

- Albero di copertura (spanning tree)
  - Dato un grafo $G = (V, E)$ non orientato e connesso, un *albero di copertura* di G è un sottografo $T = (V, E_T)$ tale che
    - T è un albero
    - $E_T \subseteq E$
    - T contiene tutti i nodi di G



Definizione del problema

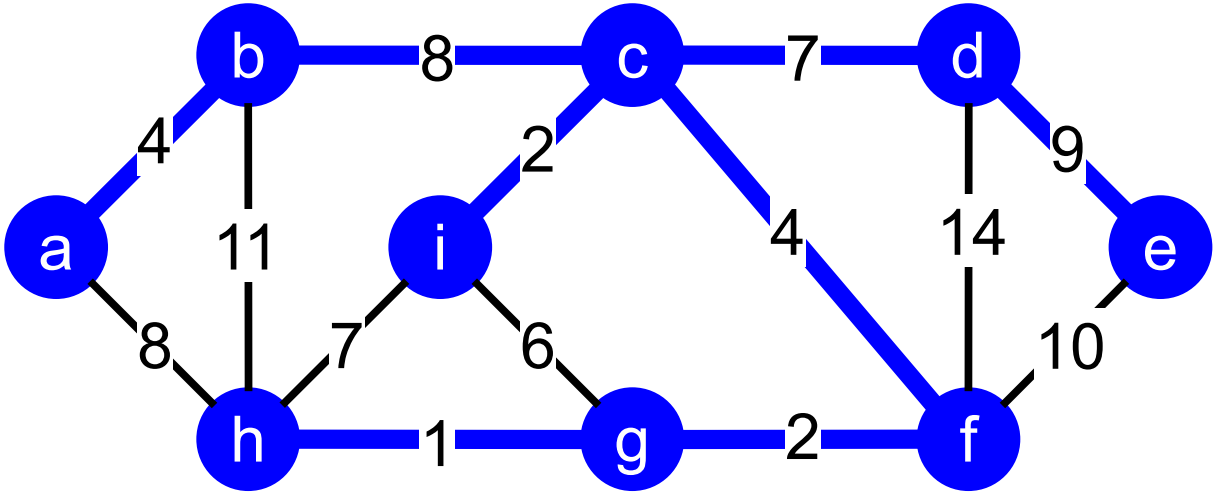
- **Output**: albero di copertura di peso minimo (*minimum spanning tree*)
  - Un albero di copertura il cui peso totale

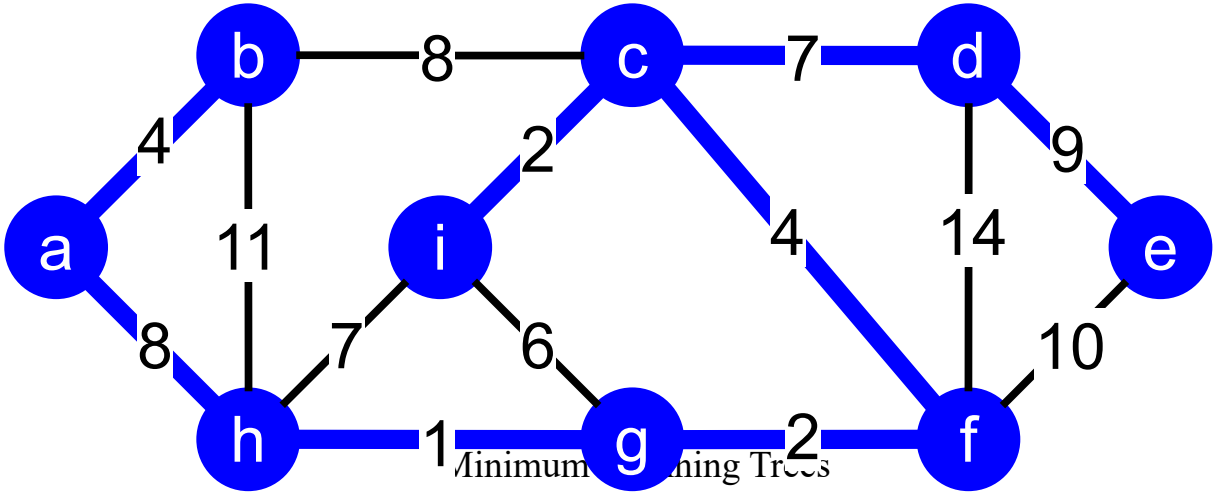
$$w(T) = \sum_{(u,v) \in T} w(u,v)$$

sia minimo, tra tutti i possibili alberi di copertura *T*

Osservazione

- Il MST non è necessariamente unico





Algoritmo generico

- Vediamo
  - Un algoritmo greedy generico
  - Due istanze di questo algoritmo: Kruskal e Prim
- L'idea è di accrescere un sottoinsieme *T* di archi in modo tale che venga rispettata la seguente condizione:
  - *T* è un sottoinsieme di qualche albero di copertura minimo
- Un arco $\{u, v\}$ è detto sicuro per T se $T \cup \{u, v\}$ è ancora un sottoinsieme di qualche MST

Algoritmo generico

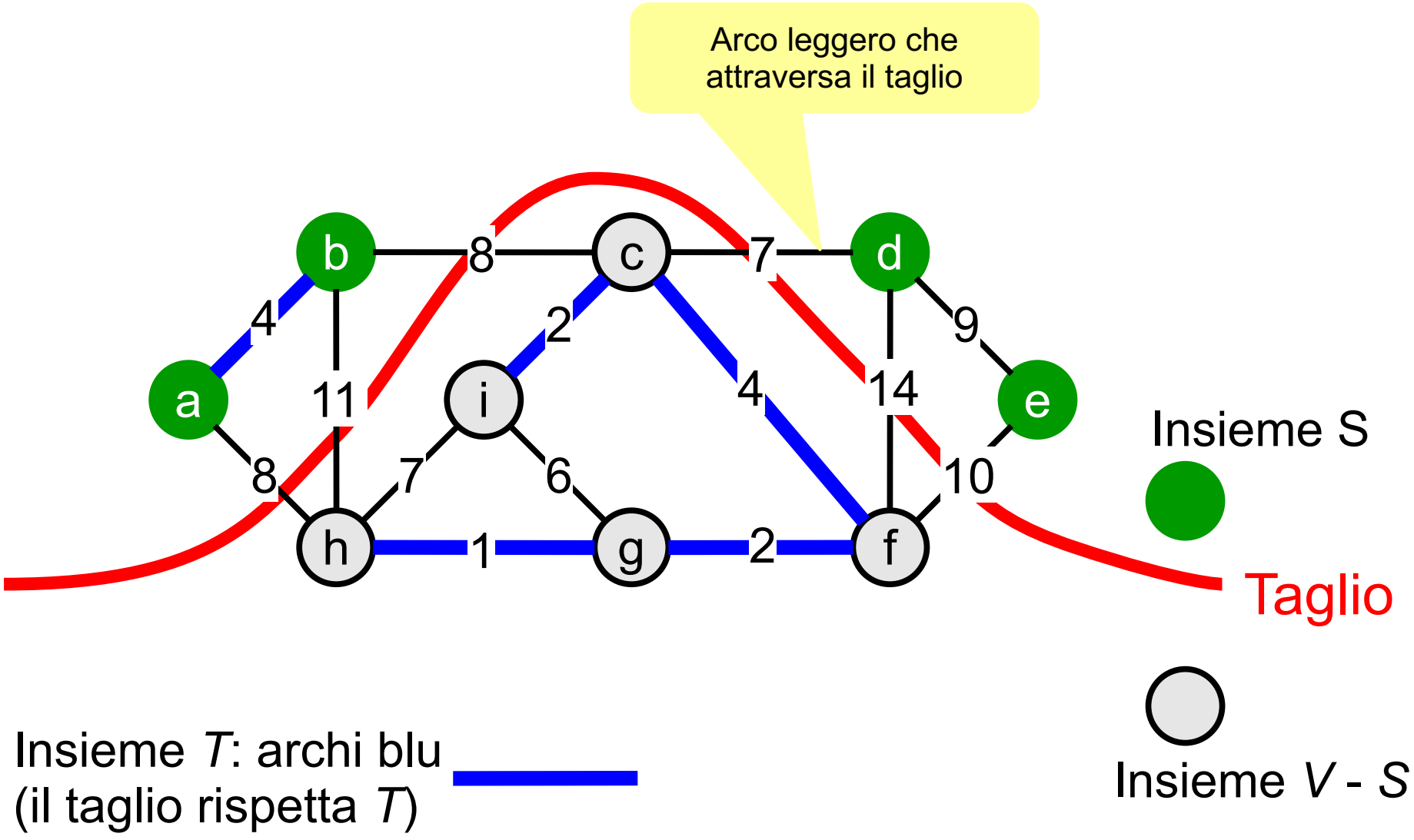
```
Tree Generic-MST(Grafo G=(V,E,w))
      Tree T ← Albero vuoto
      while T non forma un albero di copertura do
            trova un arco sicuro {u, v}
            T ← T U {u, v}
      endwhile
      return T
```

- Archi blu
  - sono gli archi che fanno parte del MST
- Archi rossi
  - sono gli archi che non fanno parte del MST

Definizioni

- Per caratterizzare gli archi sicuri dobbiamo introdurre alcune definizioni:
  - Un taglio (S, $V - S$) di un grafo non orientato $G = (V, E)$ è una partizione di V in due sottoinsiemi disgiunti
  - Un arco $\{u, v\}$ attraversa il taglio se $u \in S$ e $v \in V - S$
  - Un taglio rispetta un insieme di archi T se nessun arco di T attraversa il taglio
  - Un arco che attraversa un taglio è leggero se il suo peso è minimo fra i pesi di tutti gli archi che attraversano quel taglio

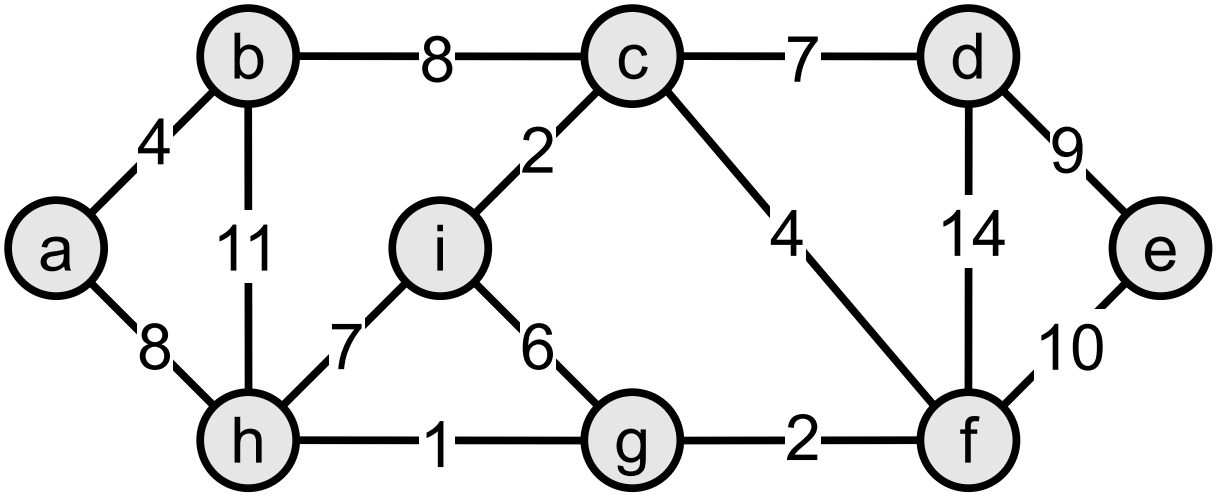
Esempio


Arco leggero che attraversa il taglio
a
b
c
d
e
f
g
h
i
4
8
8
11
7
2
4
14
9
7
6
10
1
2
Insieme S
Taglio
Insieme V - S
Insieme T: archi blu
(il taglio rispetta T)

Regole del ciclo e del taglio

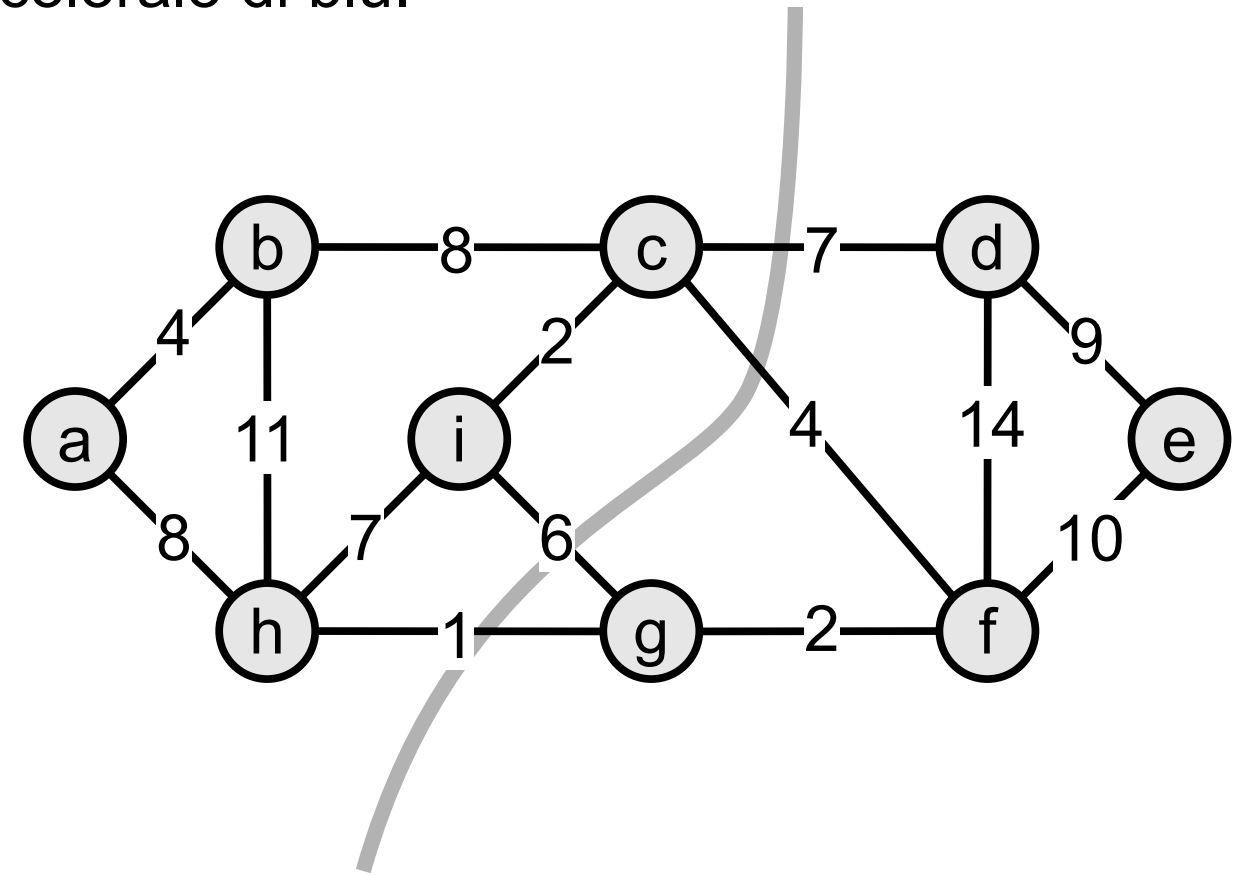
- Regola del ciclo
  - Scegli un ciclo semplice in *G* che non contenga archi rossi. Tra tutti gli archi non colorati del ciclo, seleziona un arco di costo massimo e coloralo di rosso
- Regola del taglio
  - Scegli un taglio in *G* che non contenga archi blu. Tra tutti gli archi non colorati che attraversano il taglio seleziona un arco di costo minimo e coloralo di blu
- Metodo greedy
  - Costruisce un MST applicando, ad ogni passo, una delle due regole precedenti (una qualunque, purché si possa usare)

Esempio

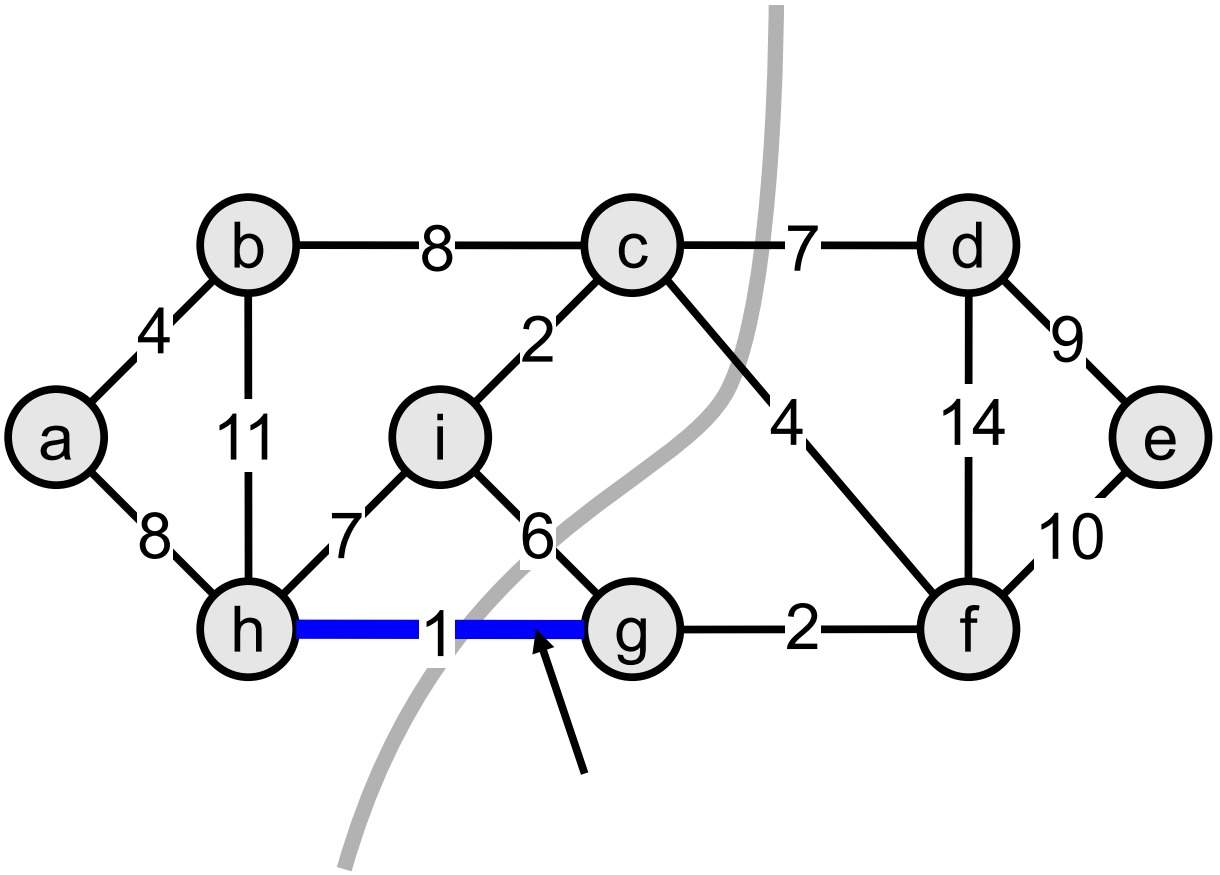


Esempio

Scegli un taglio in G che non contenga archi blu. Tra tutti gli archi non colorati che attraversano il taglio seleziona un arco di costo minimo e coloralo di blu.

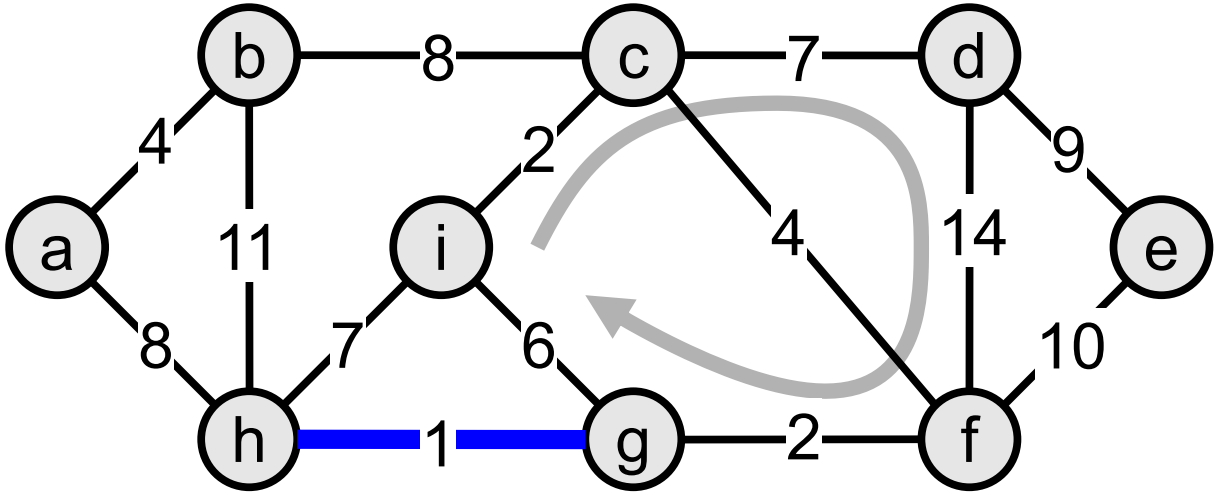


Esempio

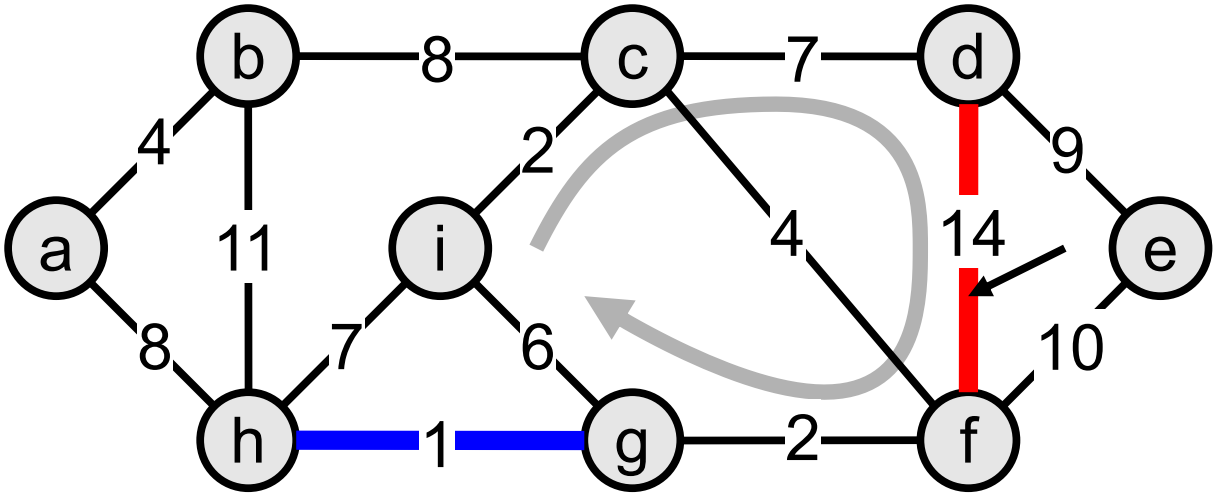


Esempio

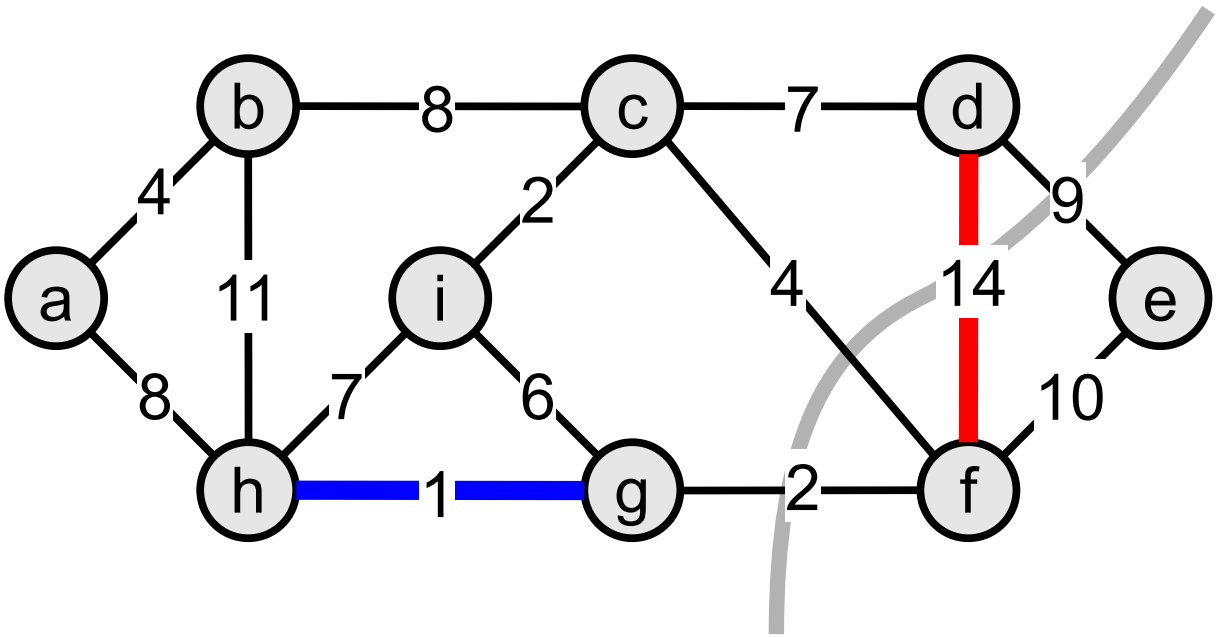
Scegli un ciclo semplice in G che non contenga archi rossi. Tra tutti gli archi non colorati del ciclo, seleziona un arco di costo massimo e coloralo di rosso



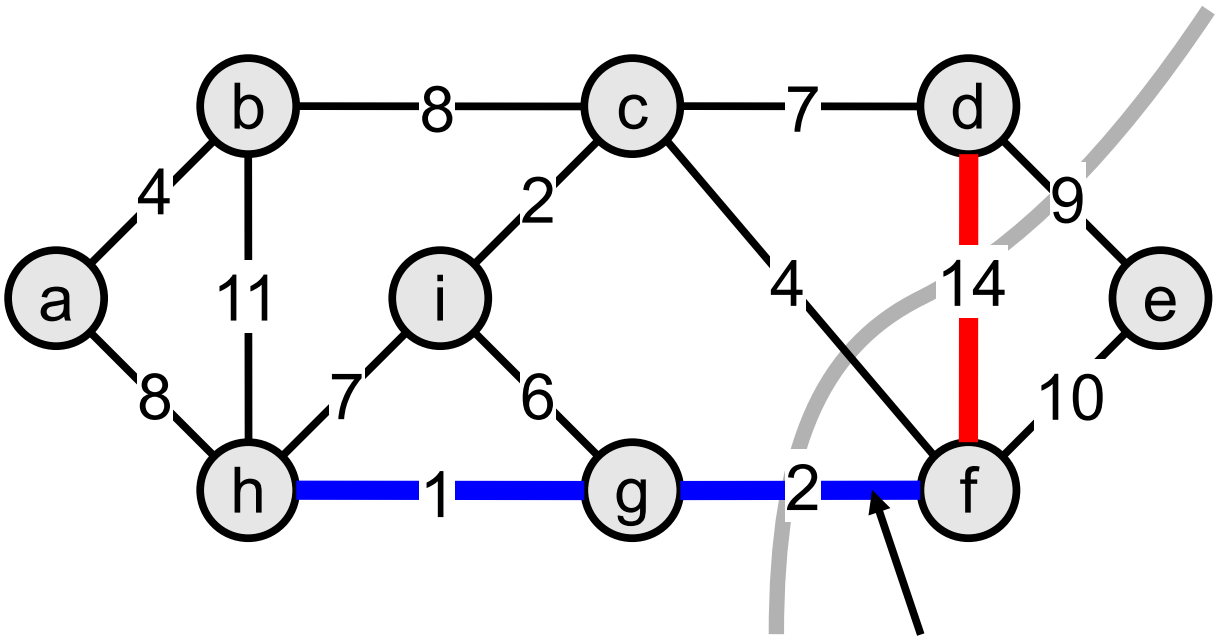
Esempio



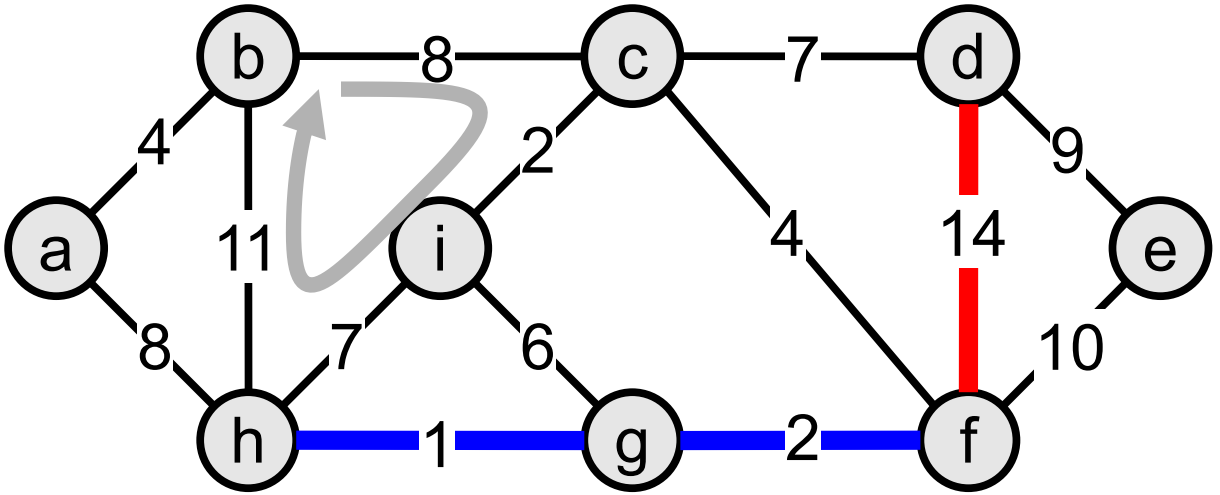
Esempio



Esempio

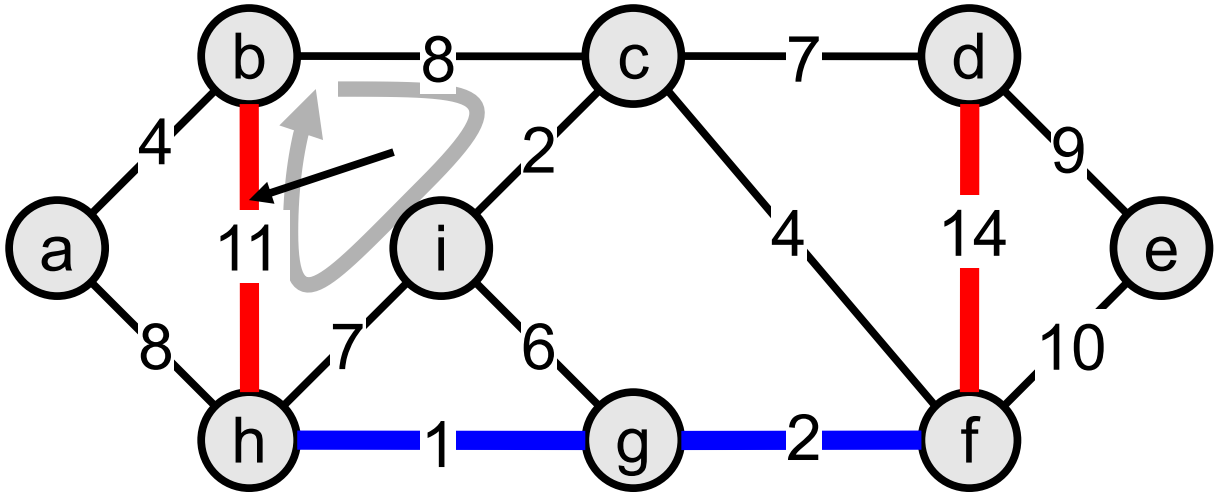


Esempio

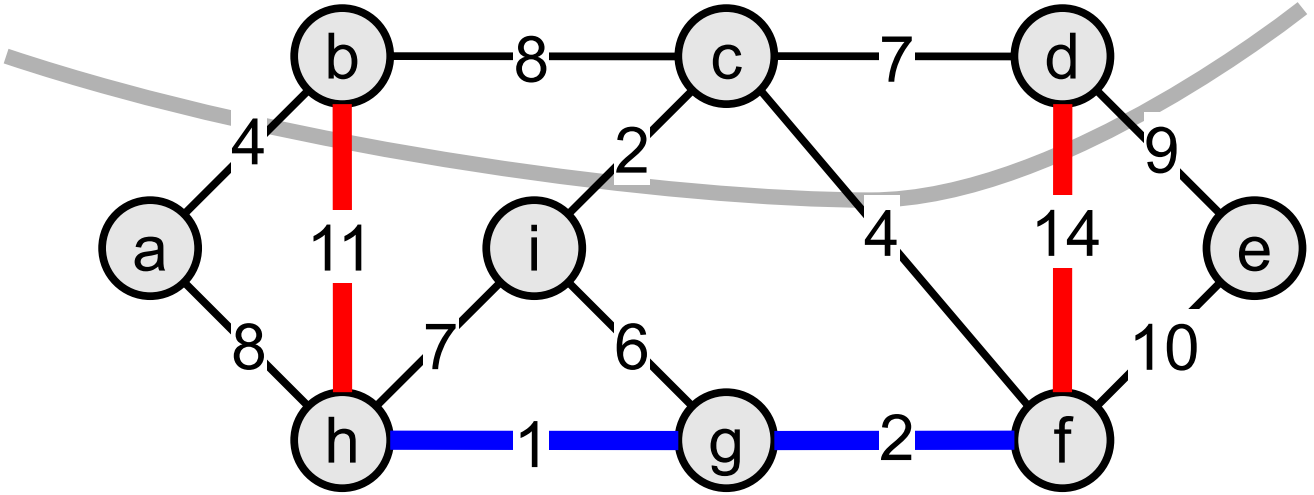


Esempio

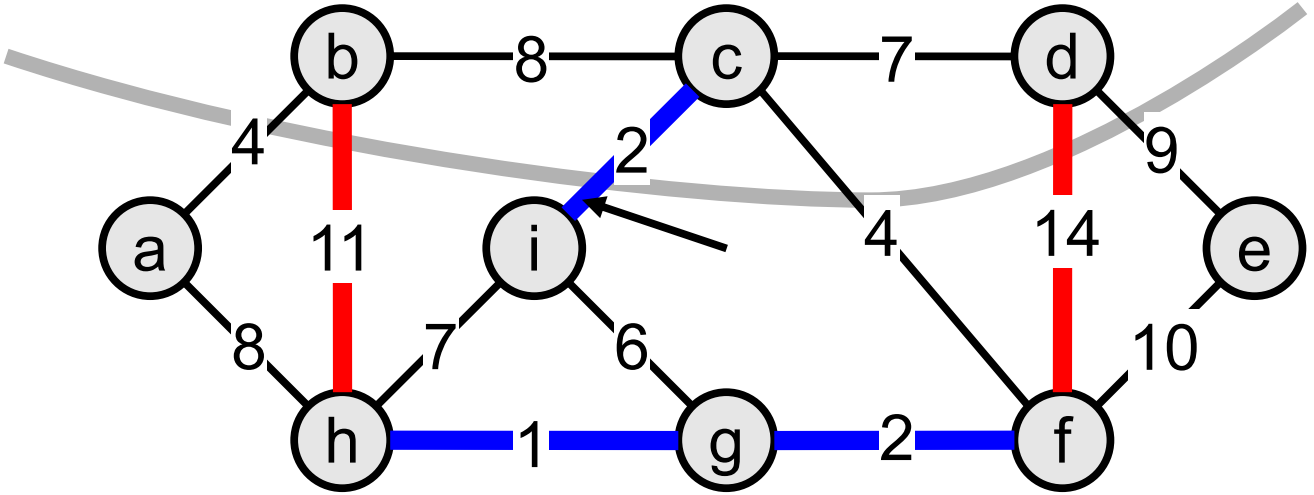
Scegli un ciclo semplice in G che non contenga archi rossi. Tra tutti gli archi non colorati del ciclo, seleziona un arco di costo massimo e coloralo di rosso



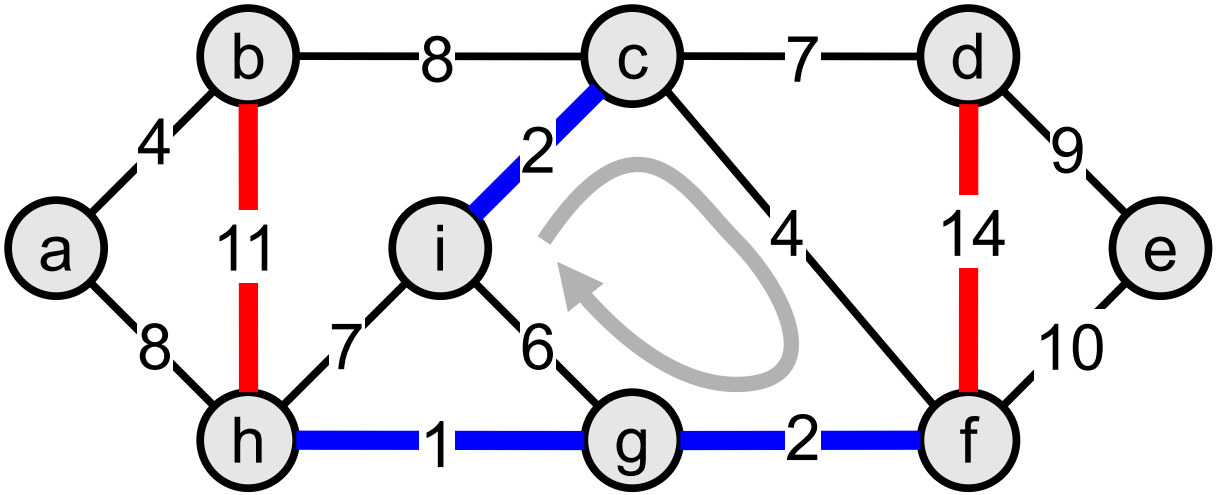
Esempio



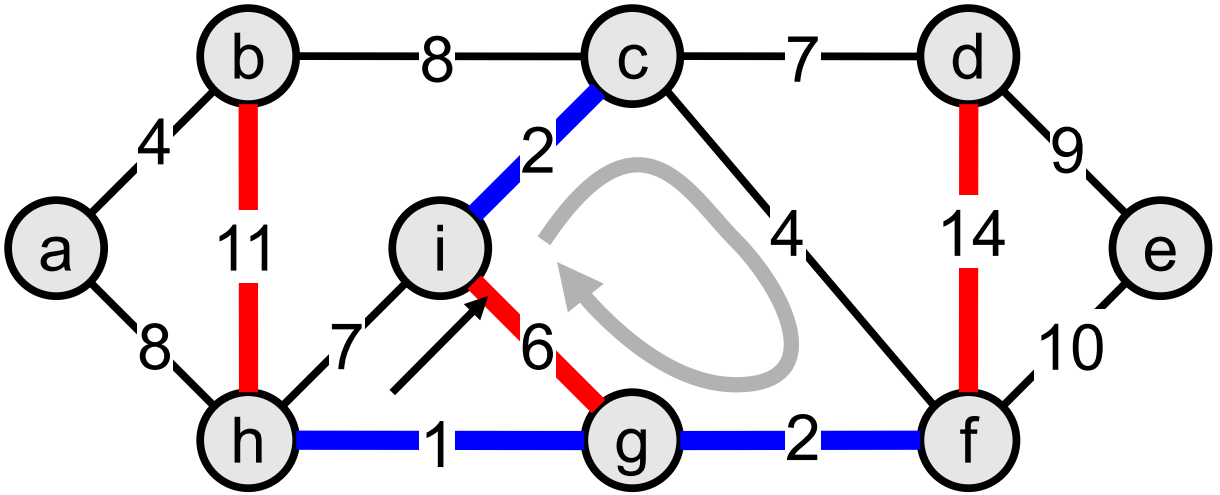
Esempio



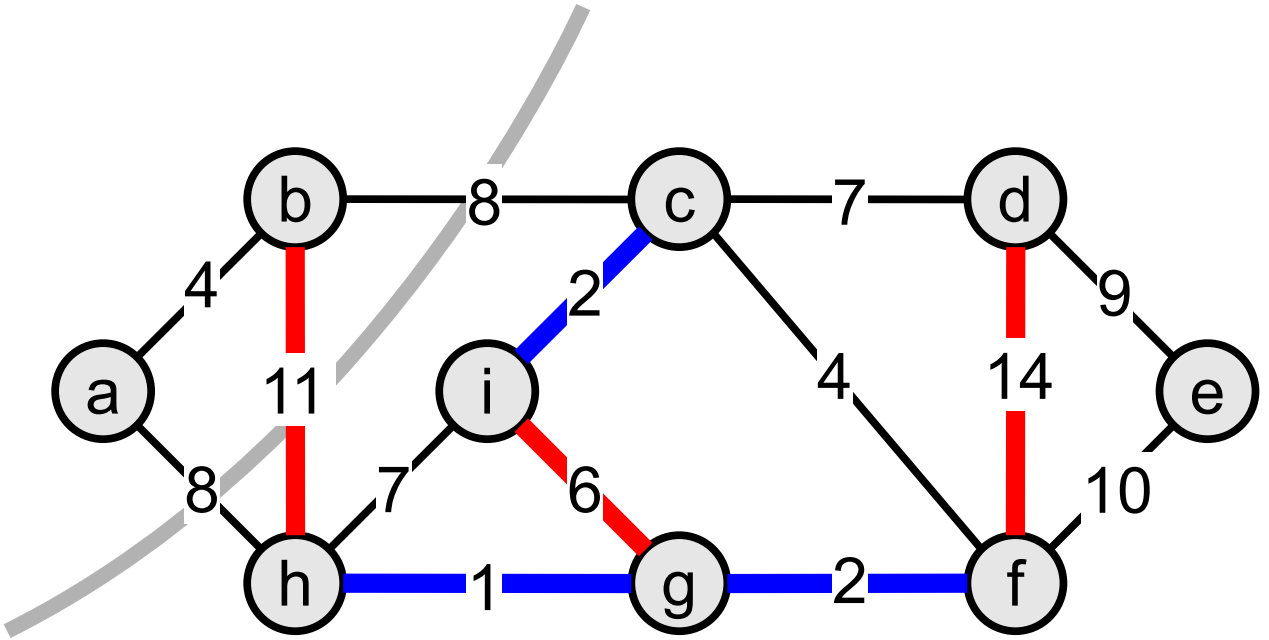
Esempio



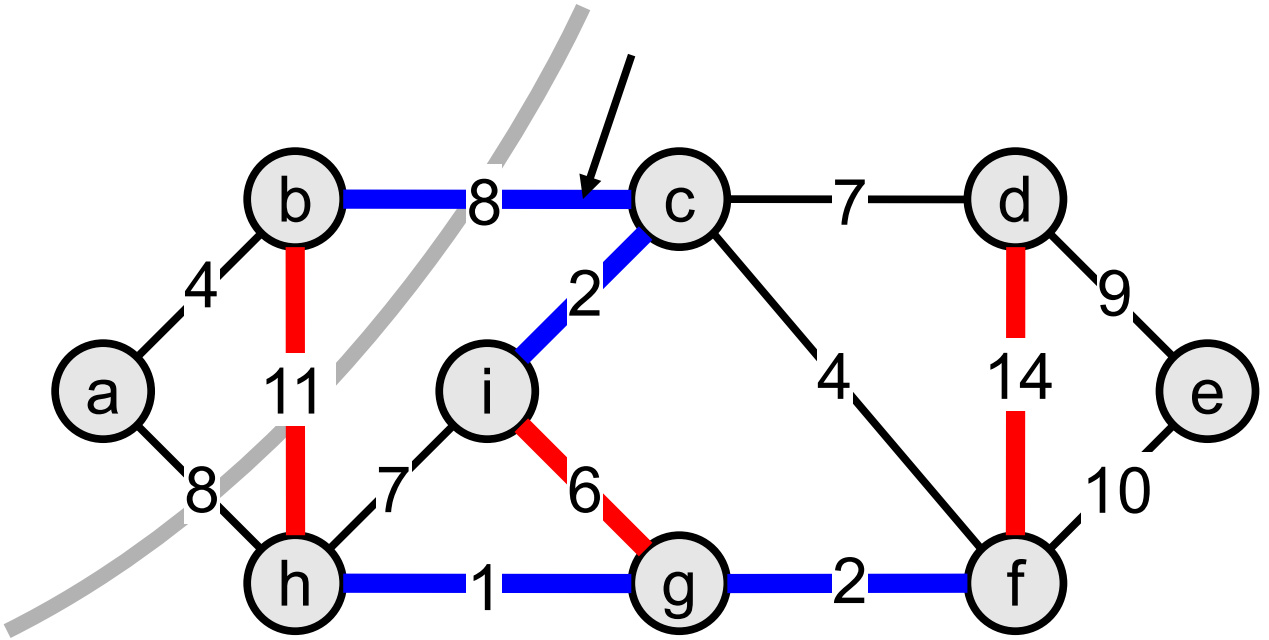
Esempio



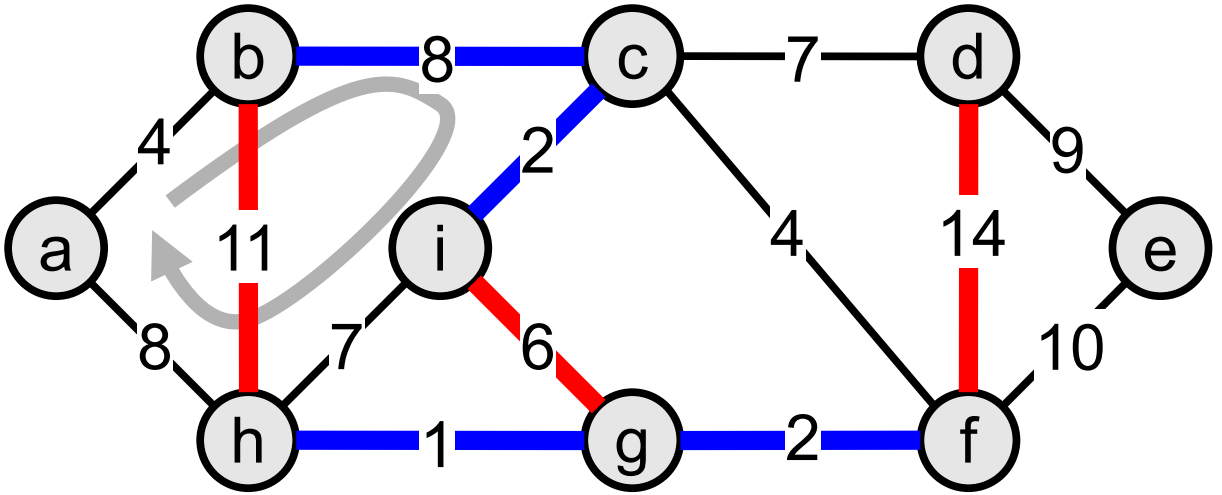
Esempio



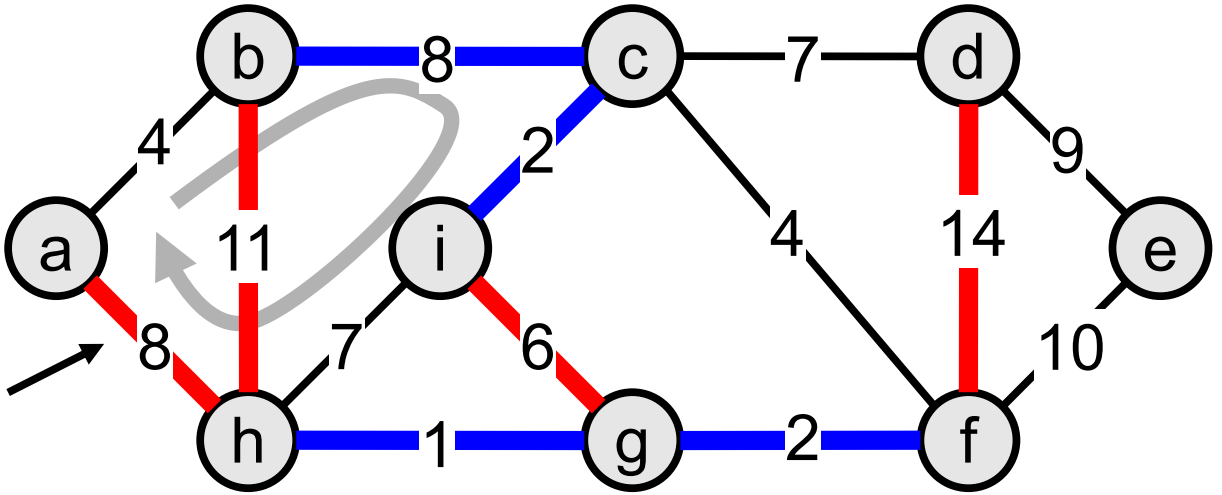
Esempio



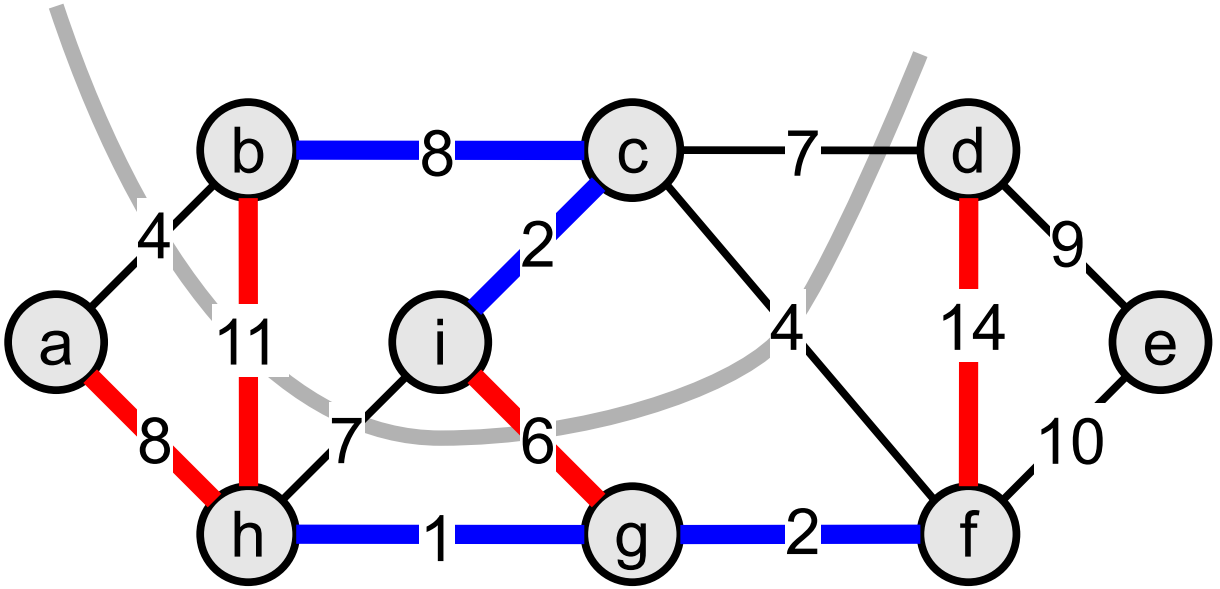
Esempio



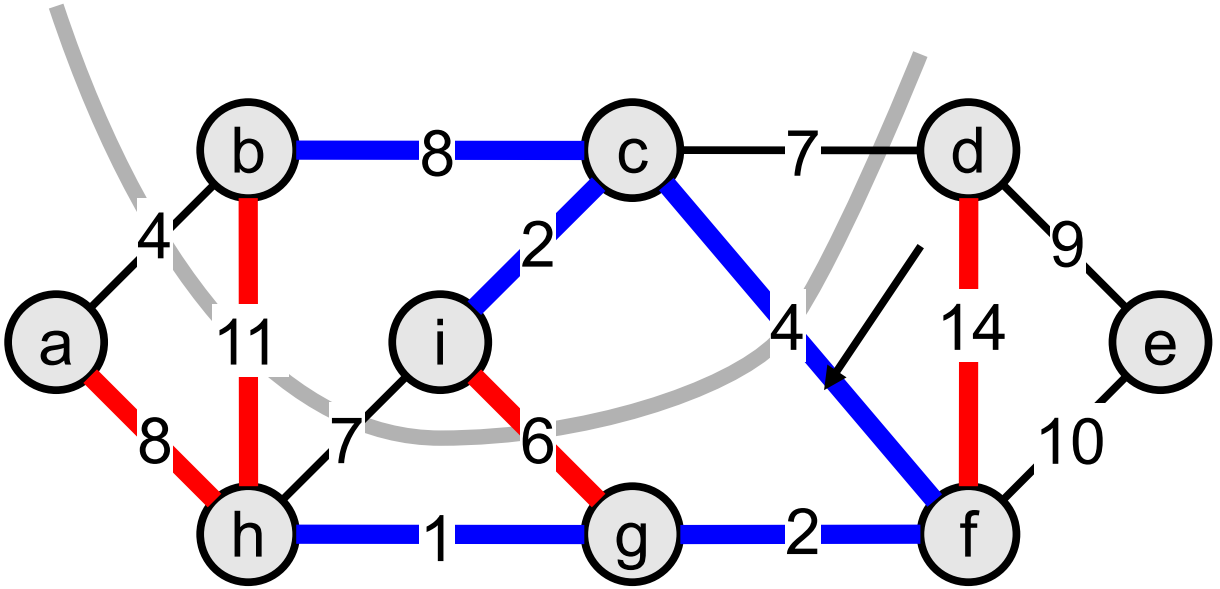
Esempio



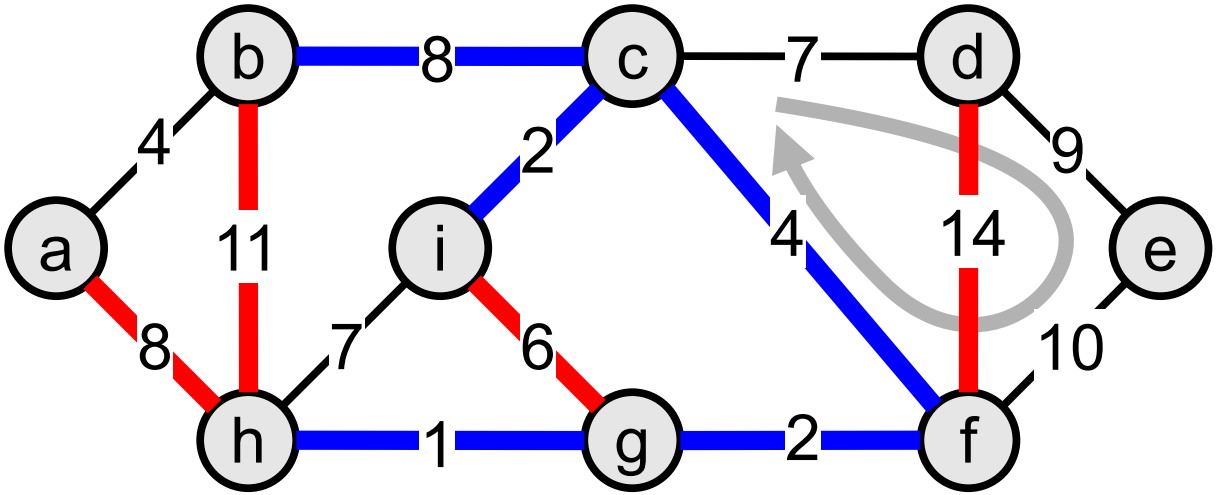
Esempio



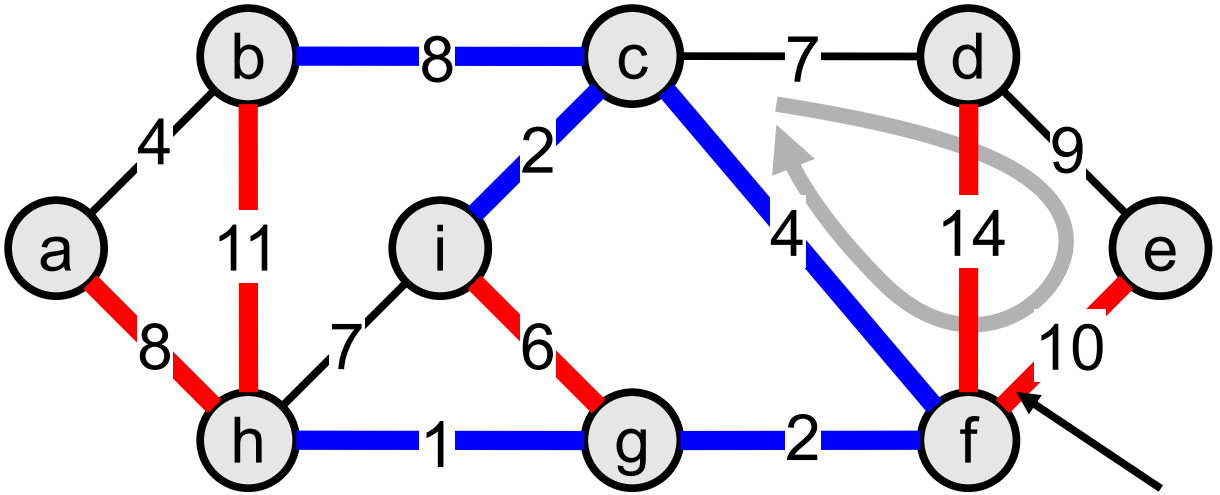
Esempio



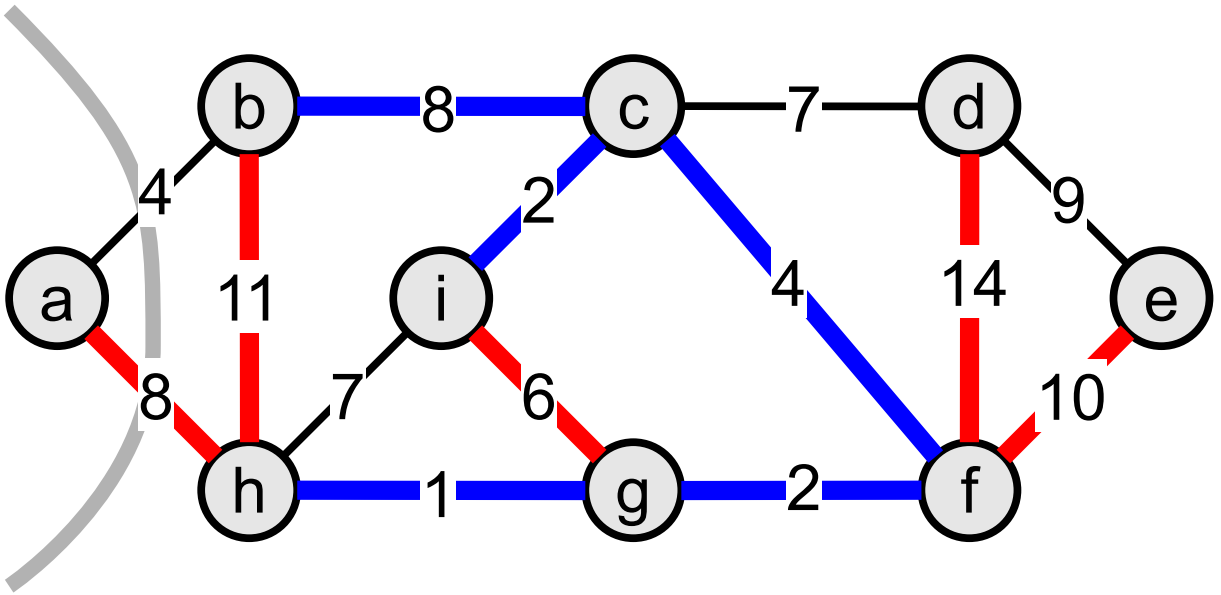
Esempio



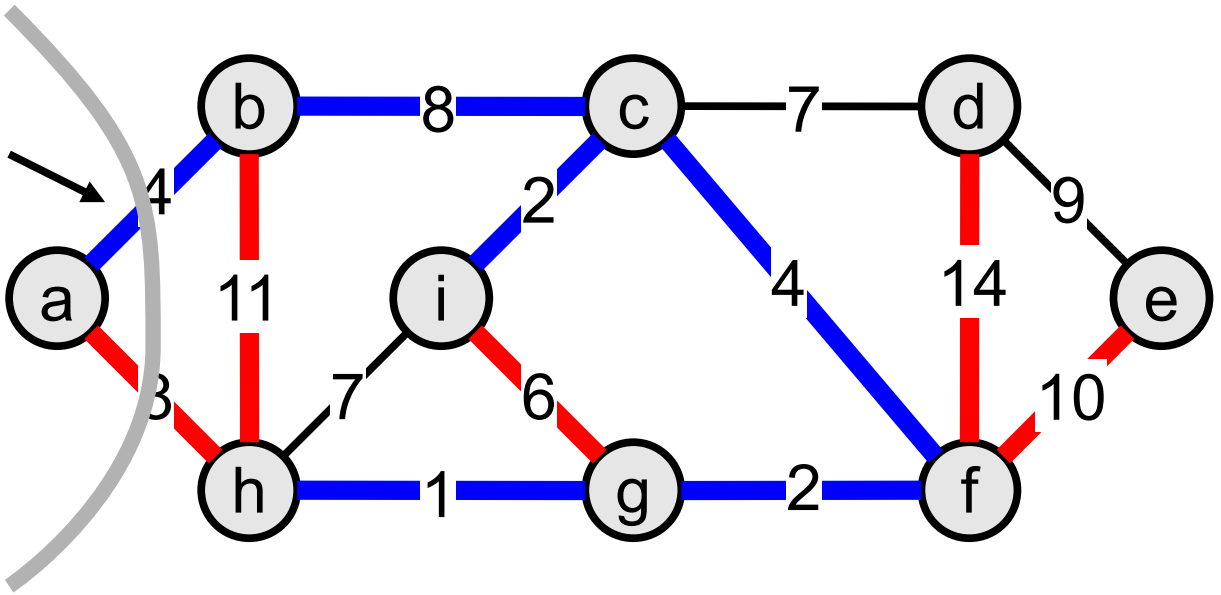
Esempio



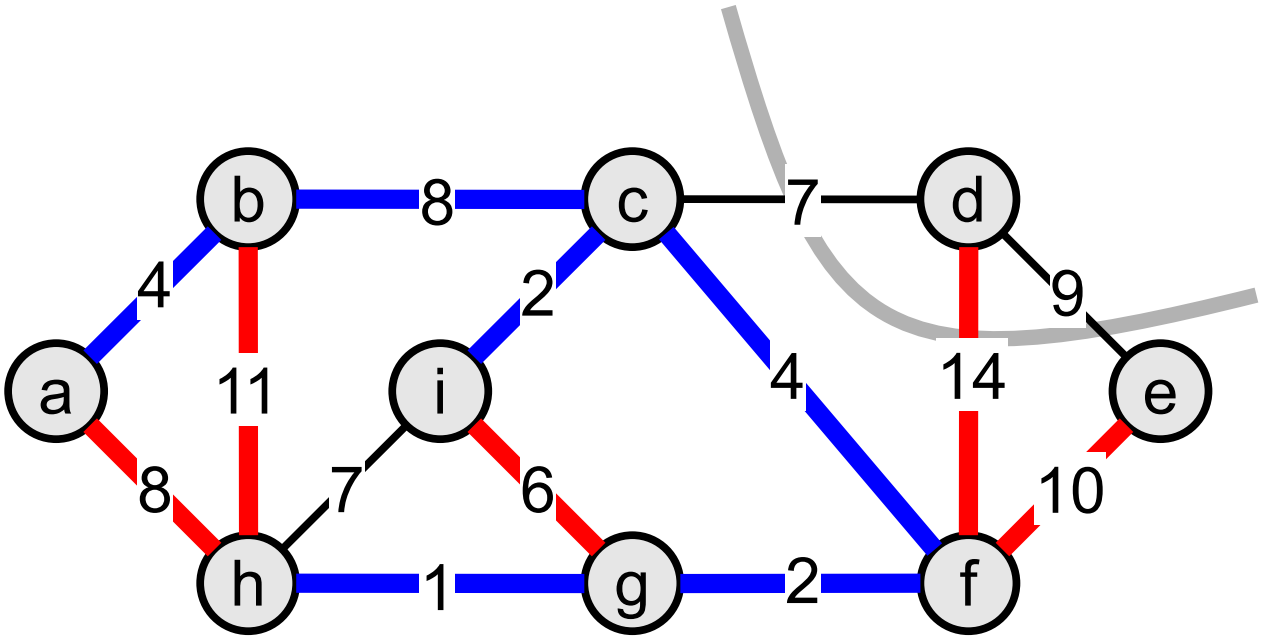
Esempio



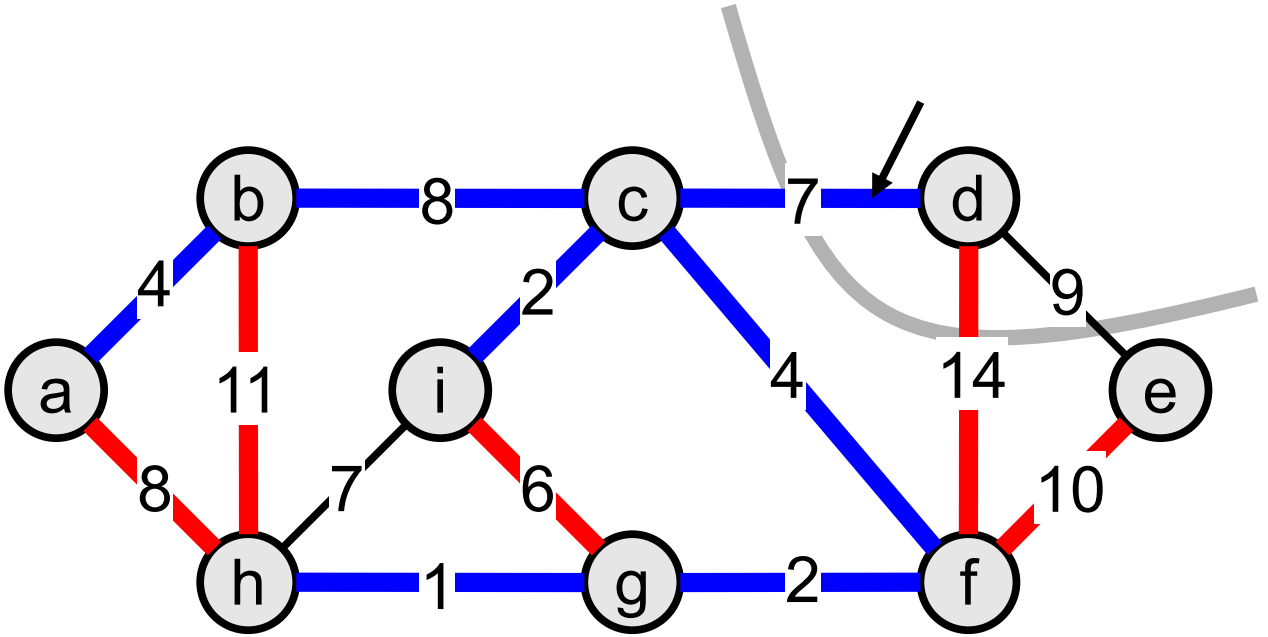
Esempio



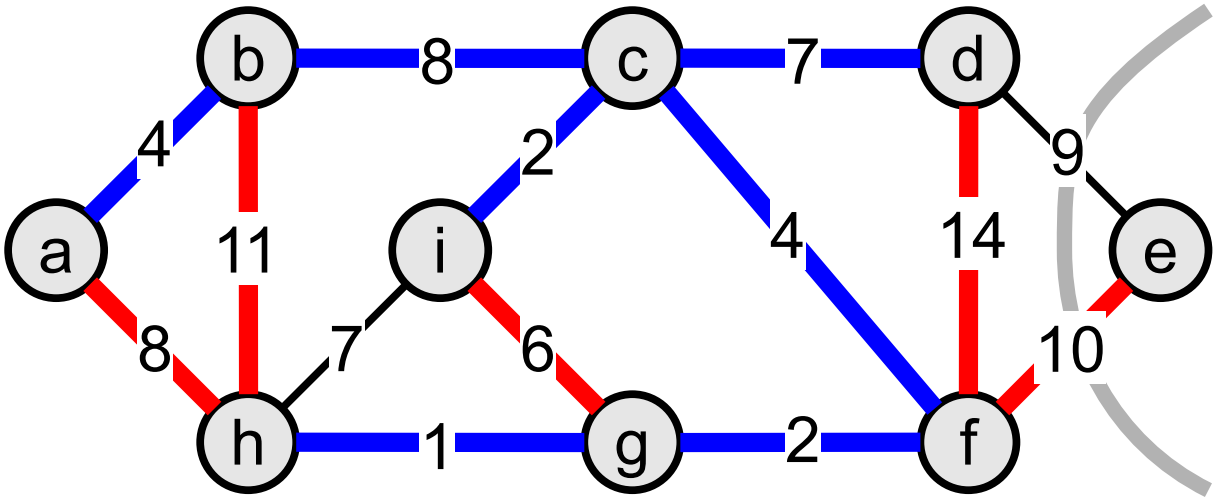
Esempio



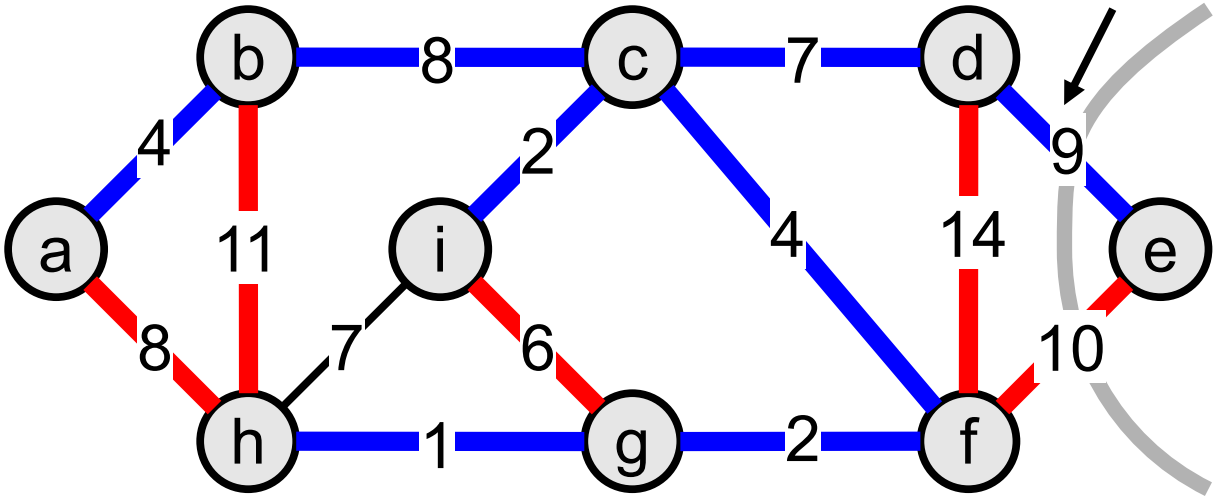
Esempio



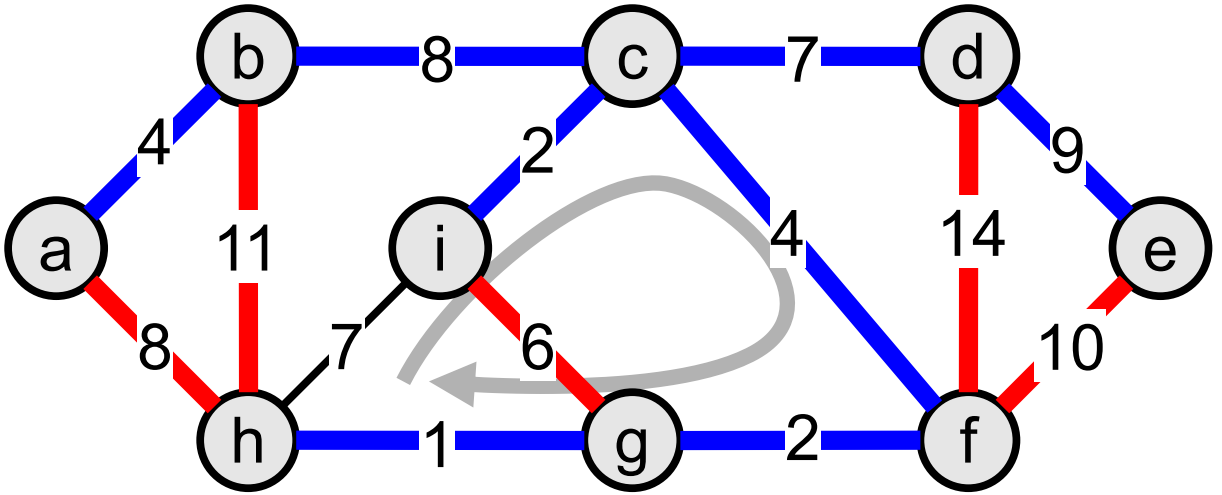
Esempio



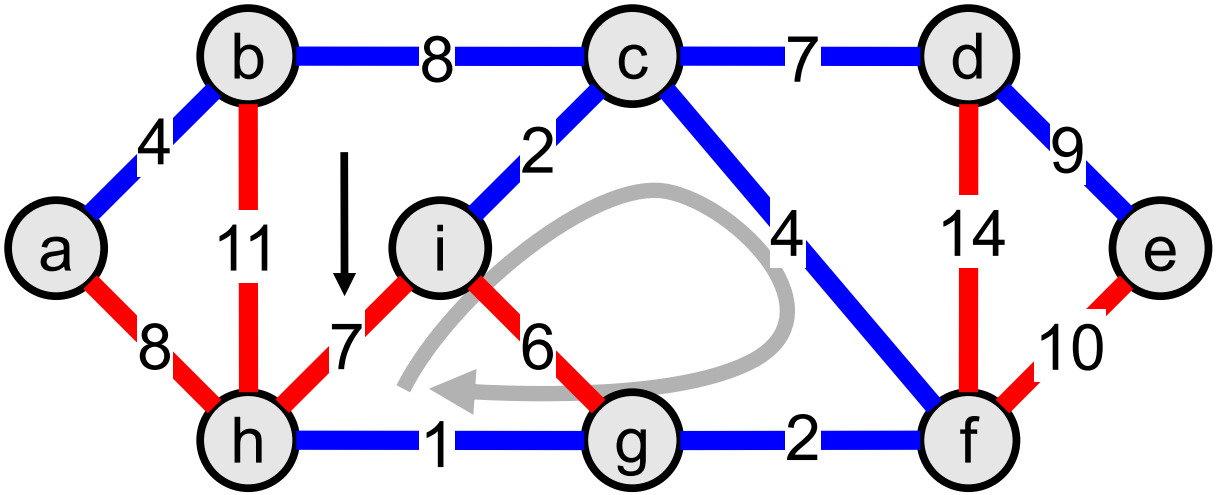
Esempio



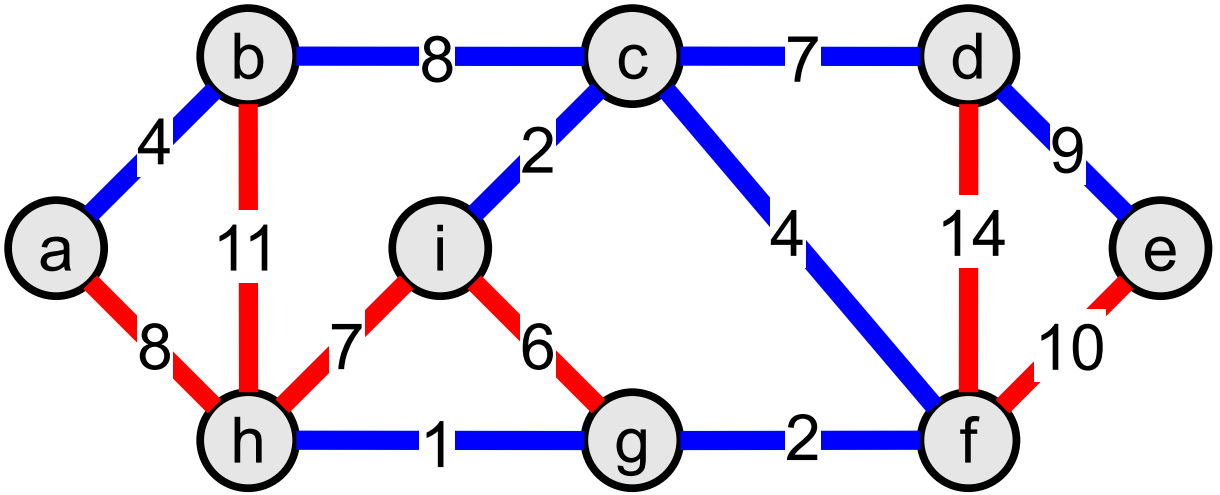
Esempio



Esempio



Finito!

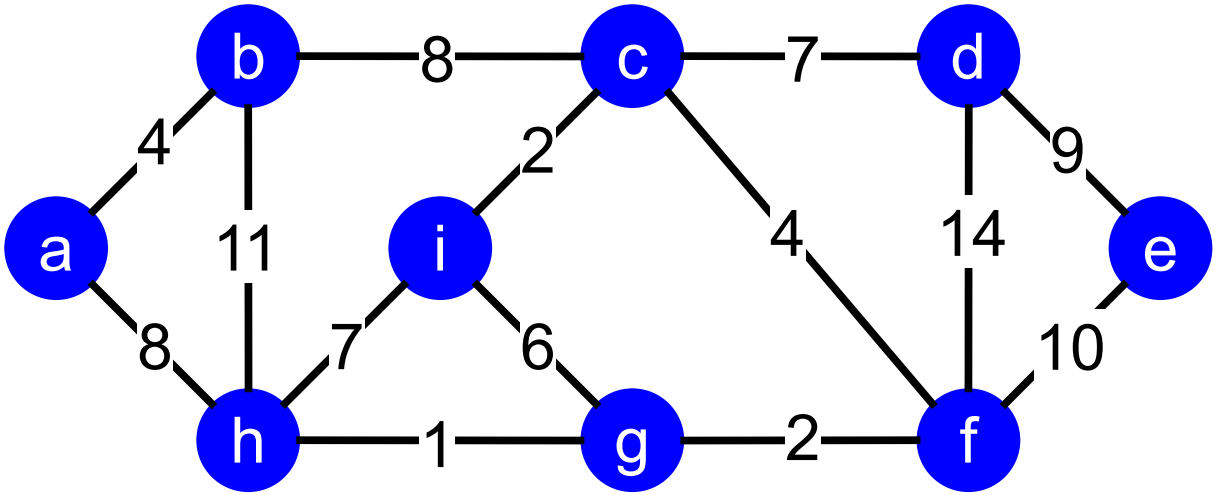


Algoritmo di Kruskal

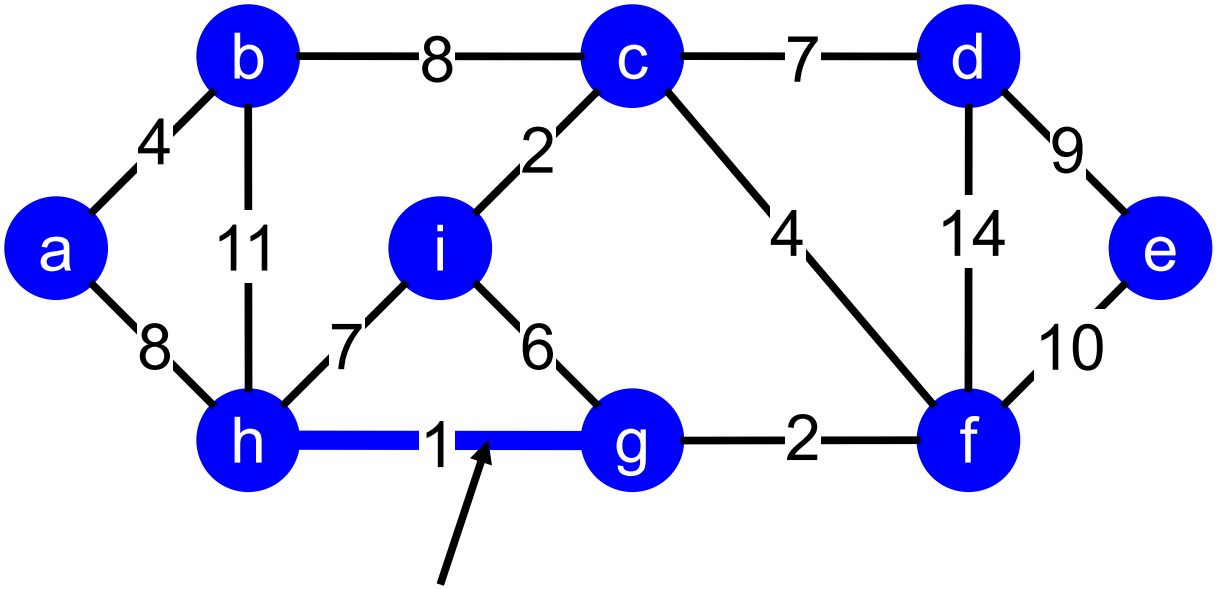
- Idea
  - Ingrandire sottoinsiemi disgiunti di un albero di copertura minimo connettendoli fra di loro fino ad avere l'albero finale
    - Inizialmente la foresta di copertura è composta da *n* alberi, uno per ciascun nodo, e nessun arco
  - Si considerano gli archi in ordine non decrescente di peso
    - Se l'arco $e = \{u, v\}$ connette due alberi blu distinti, lo si colora di blu. Altrimenti lo si colora di rosso
  - L'algoritmo è greedy perché ad ogni passo si aggiunge alla foresta un arco con il peso minimo

Joseph B. Kruskal: *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem*. In: Proceedings of the American Mathematical Society, Vol 7, No. 1 (Feb, 1956), pp. 48–50

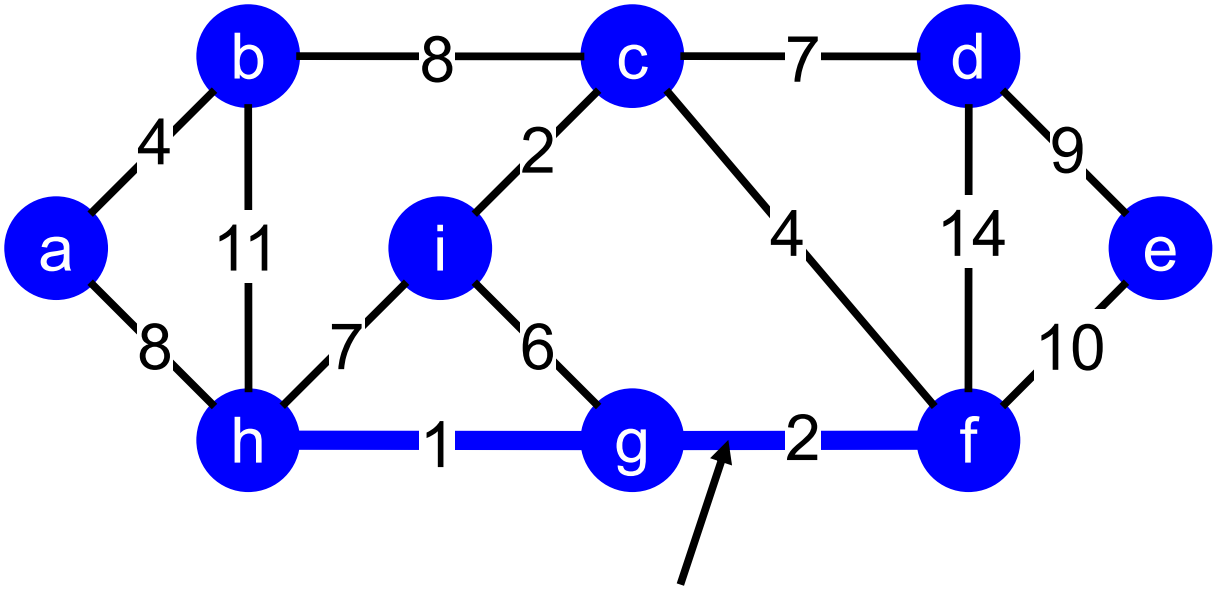
Esempio
Algoritmo di Kruskal



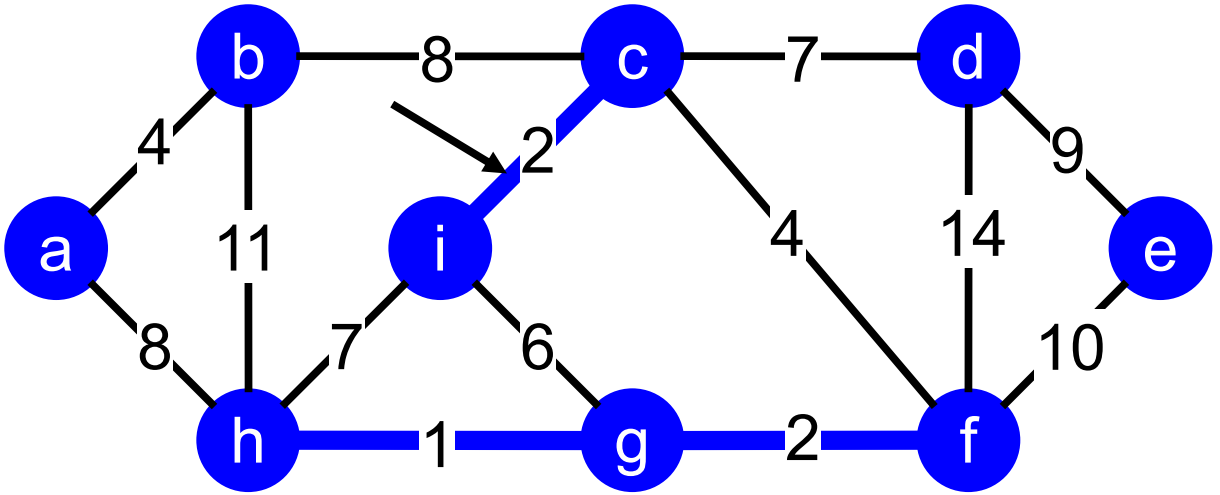
Esempio
Algoritmo di Kruskal



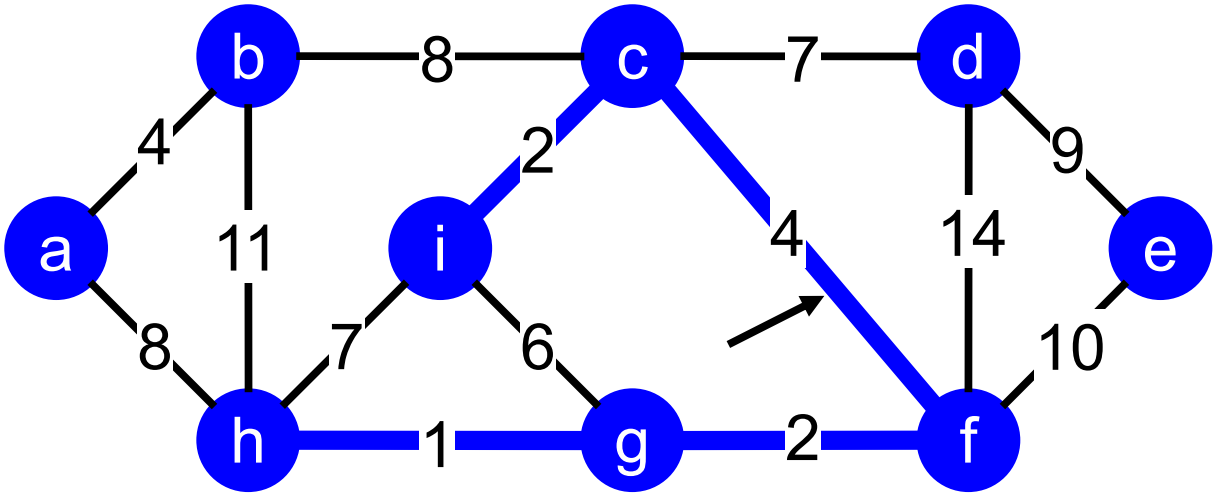
Esempio
Algoritmo di Kruskal



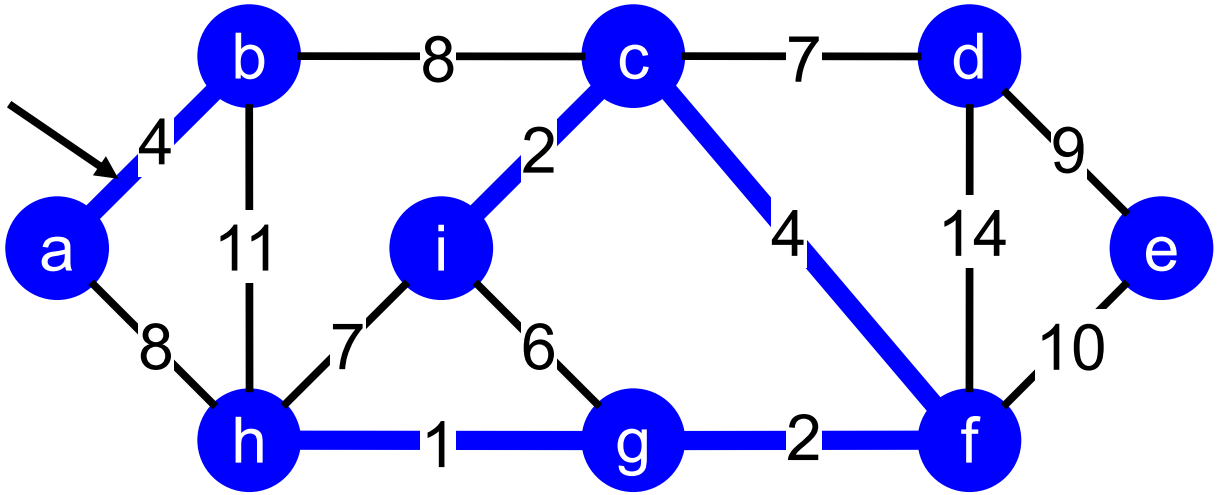
Esempio
Algoritmo di Kruskal



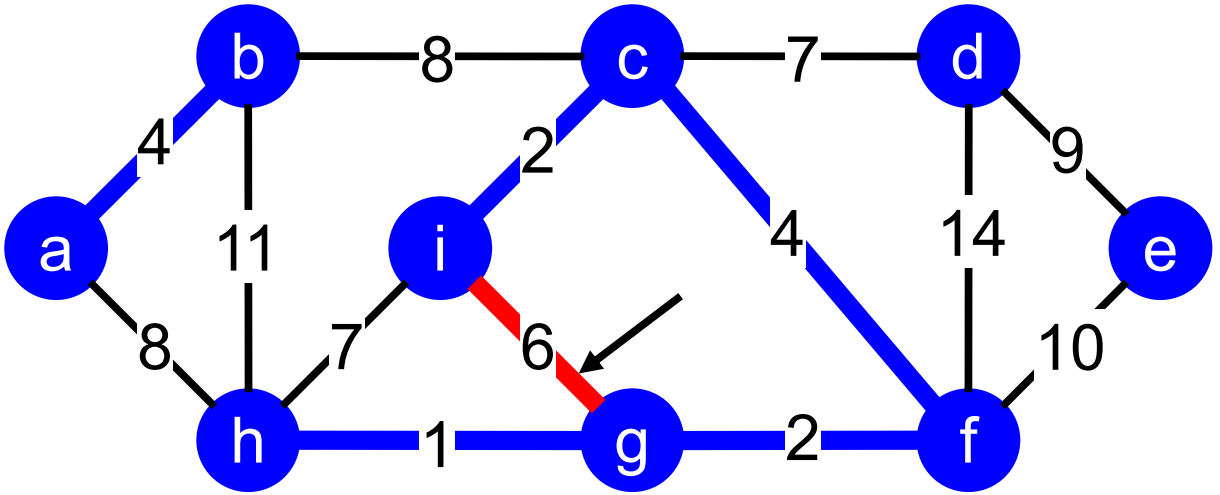
Esempio
Algoritmo di Kruskal



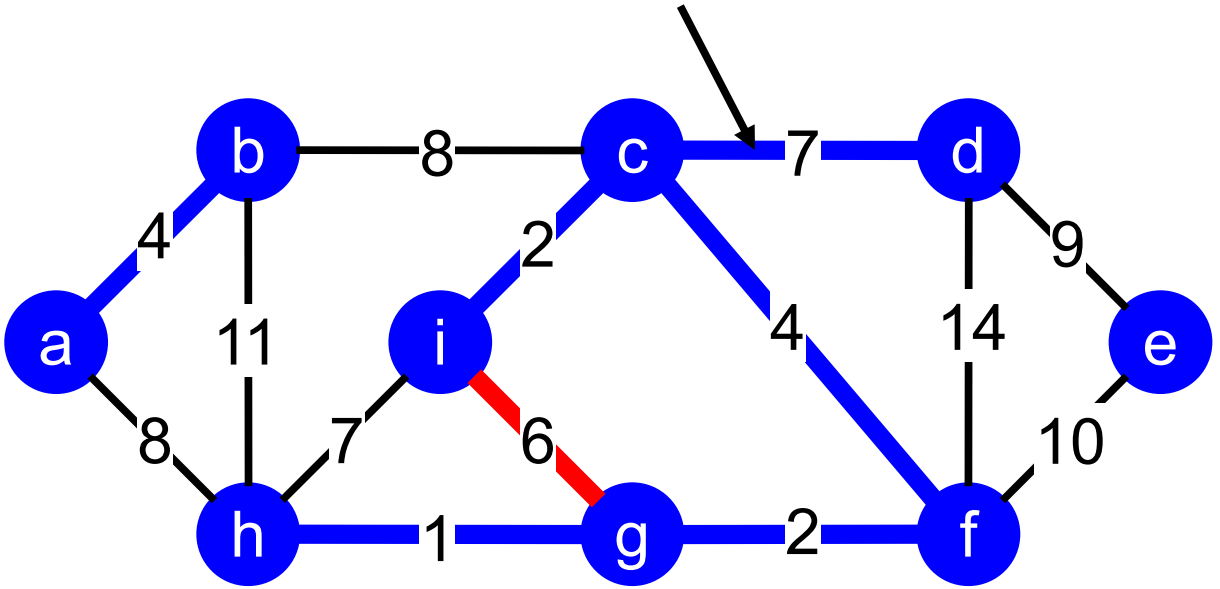
Esempio
Algoritmo di Kruskal



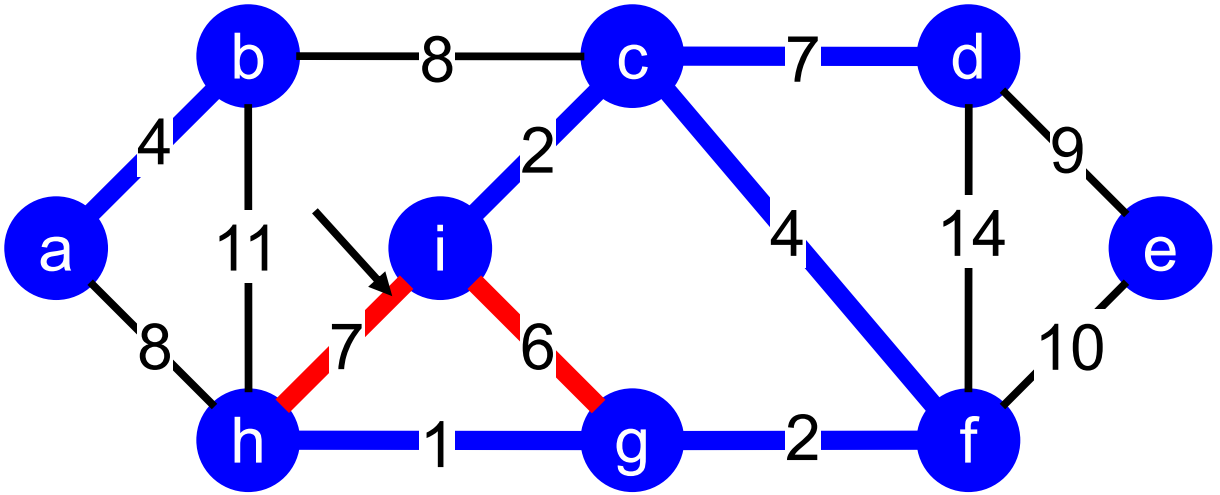
Esempio
Algoritmo di Kruskal



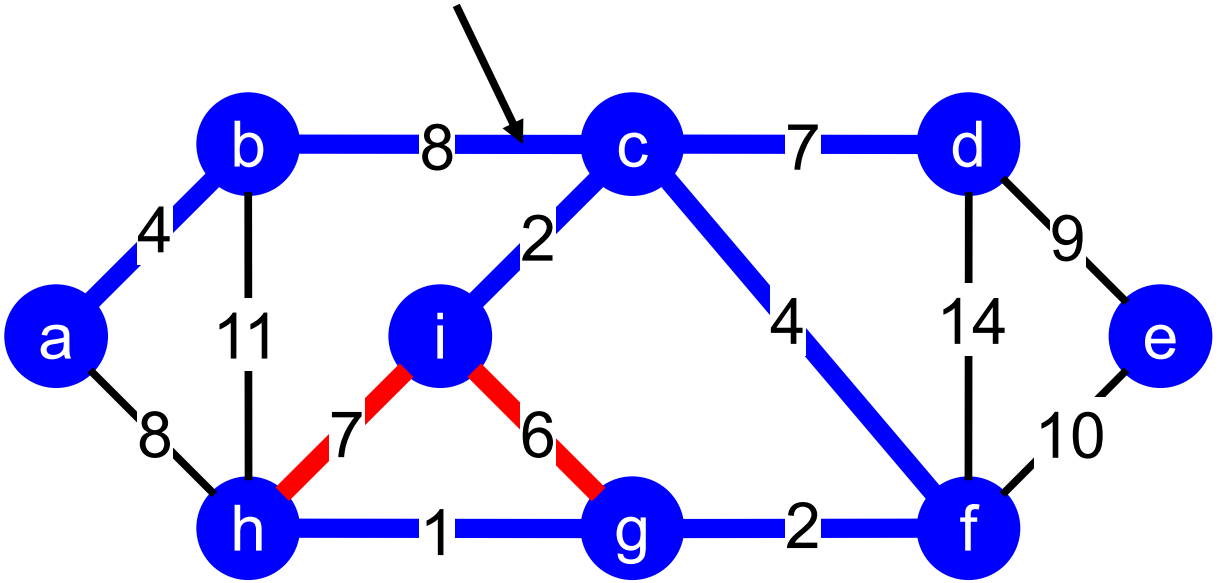
Esempio
Algoritmo di Kruskal



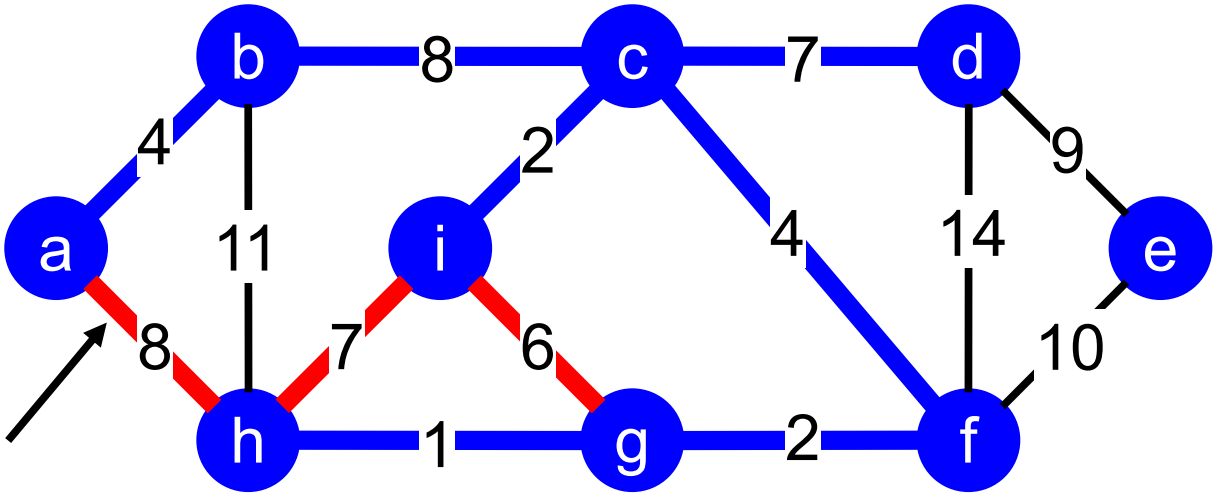
Esempio
Algoritmo di Kruskal



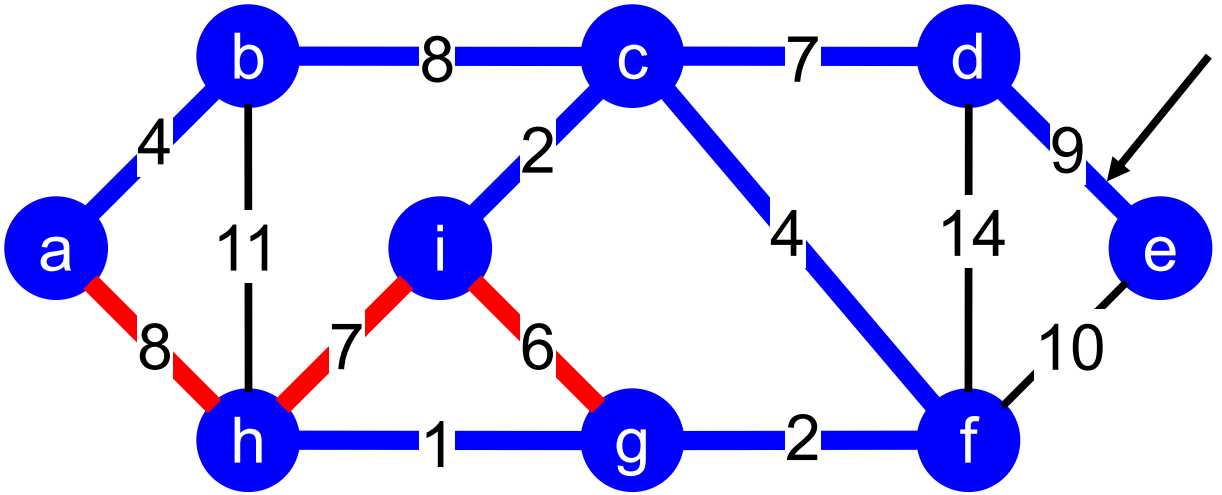
Esempio
Algoritmo di Kruskal



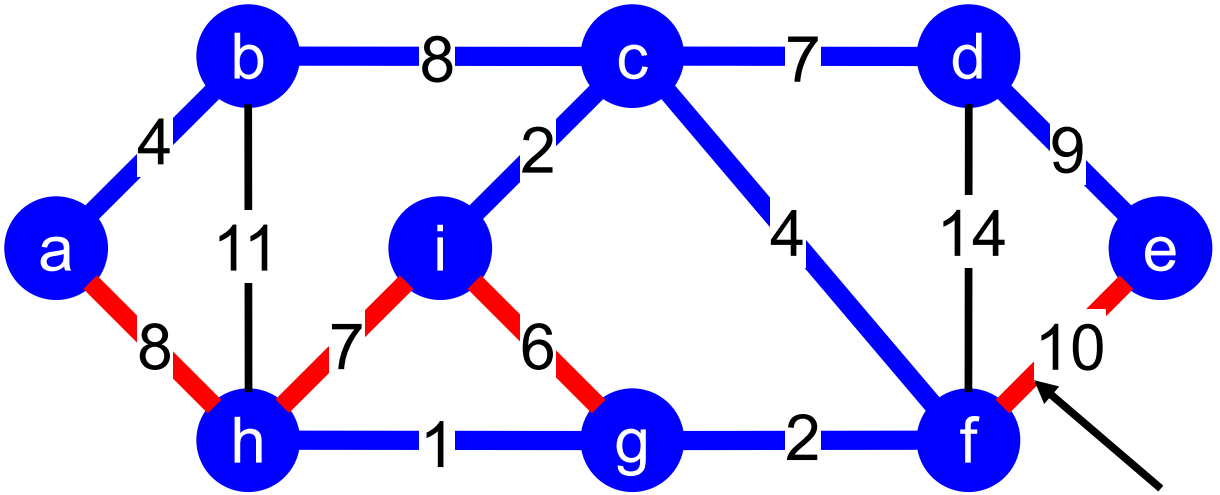
Esempio
Algoritmo di Kruskal



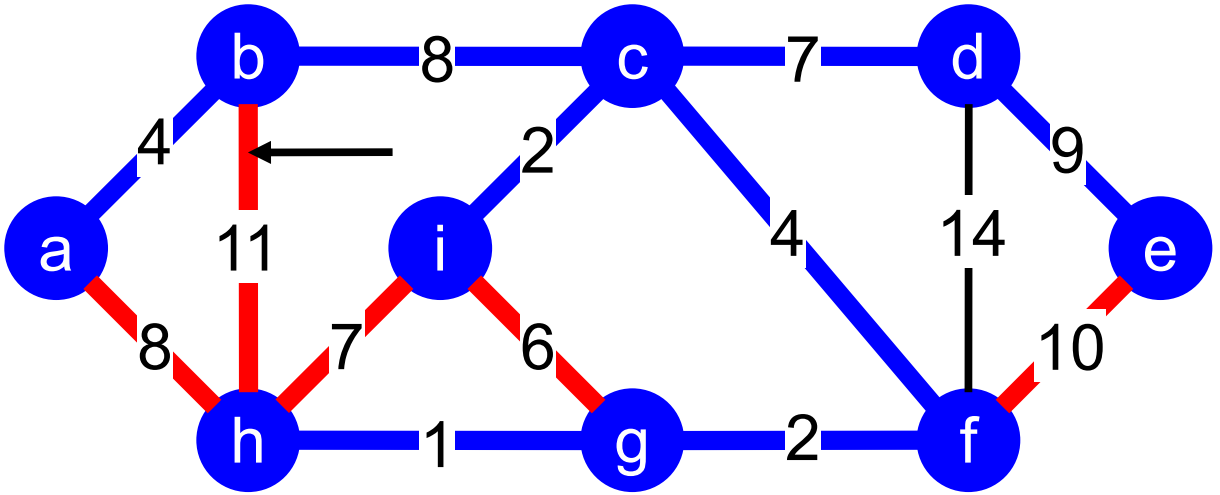
Esempio
Algoritmo di Kruskal



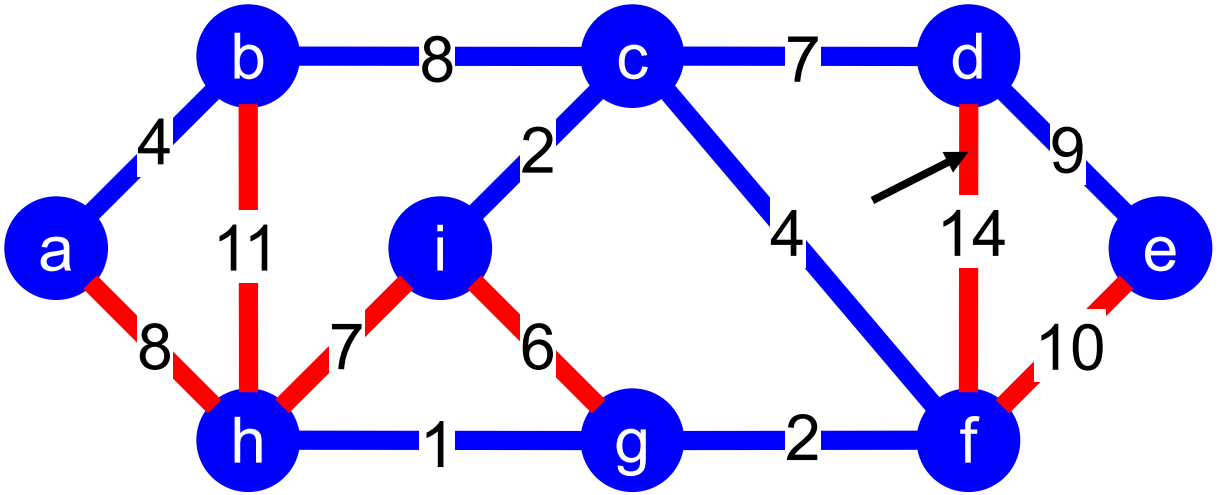
Esempio
Algoritmo di Kruskal



Esempio
Algoritmo di Kruskal



Esempio
Algoritmo di Kruskal



- Finito! Il MST è composto dai soli archi blu

Implementazione

- Ordinare gli archi in ordine non decrescente di peso
  - Sappiamo come fare
- Determinare se gli estremi di un arco appartengono allo stesso albero oppure no
  - Anche qui, sappiamo come fare...
  - ...usando le strutture ***merge-find***!

Algoritmo di Kruskal

```
Tree Kruskal-MST(Grafo G=(V,E,w))
      Tree T ← albero vuoto
      MergeFind MF ← new mfset( G.numNodi() );
      // ordina gli archi di E per peso w crescente
      sort(E, w)
      for each {u, v} ∈ E do
            Tu ← MF.find(u)
            Tv ← MF.find(v)
            if (Tu ≠ Tv) then              // evita i cicli
                  T ← T U {u, v}           // aggiungi arco
                  MF.merge(Tu, Tv)         // unisci componenti
            endif
      endfor
      return T
```

Analisi

- L'ordinamento richiede tempo
$$O(m \log m) = O(m \log n^2) = O(m \log n)$$

dove m è il numero di archi e n il numero di nodi

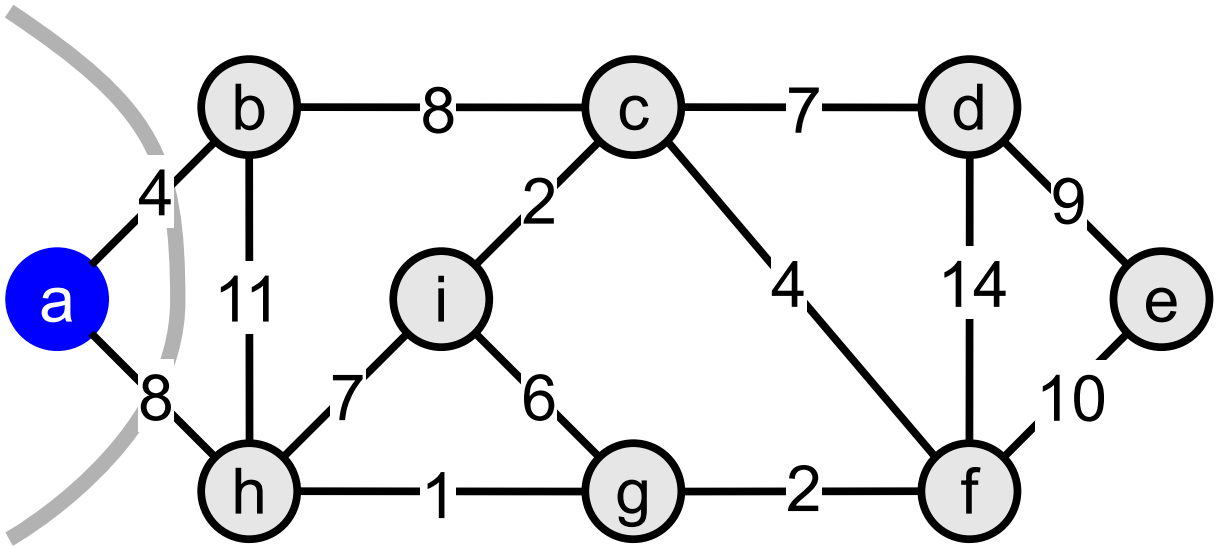
- Il tempo di esecuzione dipende dalla realizzazione della struttura Merge-Find
  - Vengono effettuate $2m$ Find e $(n - 1)$ Merge, oltre alla creazione della struttura Merge-Find
- Se usiamo QuickUnion con euristica sul rango, la sequenza di operazioni costa in tutto $O((m + n) \log n)$
- Totale: $O(2m \log n + n \log n) = O(m \log n)$

Algoritmo di Prim

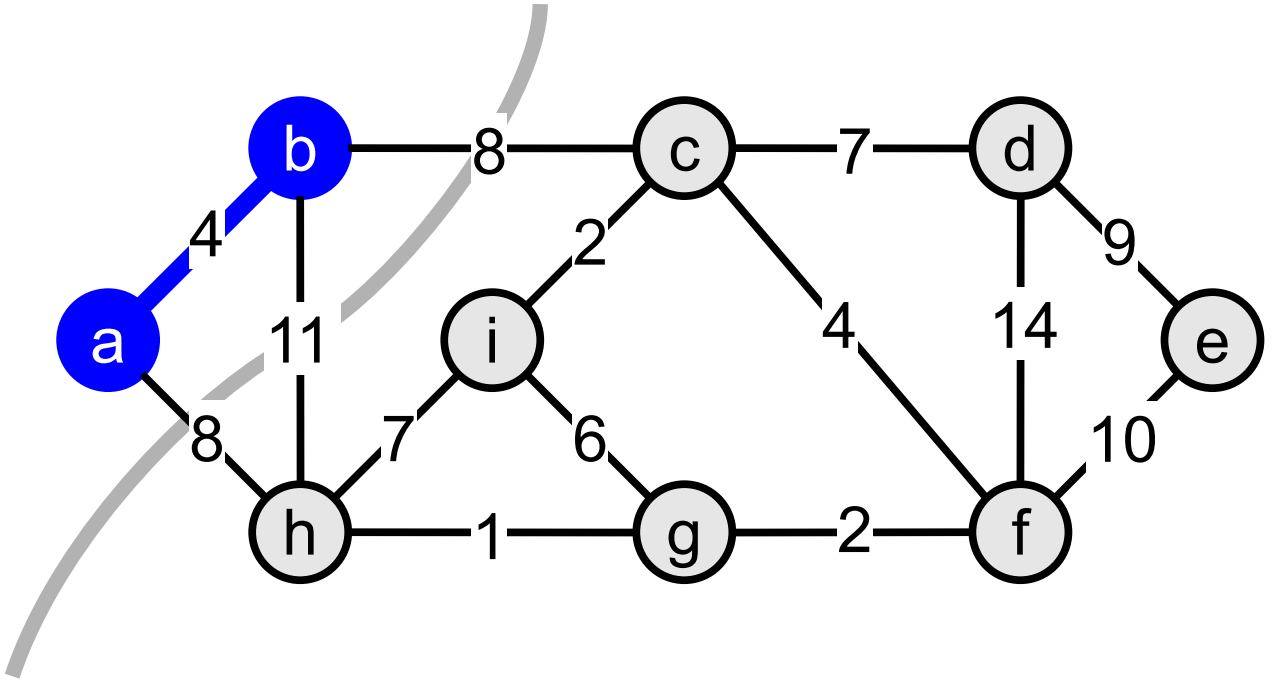
- L'algoritmo di Prim procede mantenendo un singolo albero *T* che viene fatto "crescere"
  - L'albero parte da un nodo arbitrario *r* (la *radice*) e cresce fino a quando ricopre tutti i vertici
  - Ad ogni passo viene aggiunto l'arco di peso minimo che collega un nodo già raggiunto dell'albero con uno non ancora raggiunto

R. C. Prim: *Shortest connection networks and some generalizations*. In: Bell System Technical Journal, 36 (1957), pp. 1389–1401

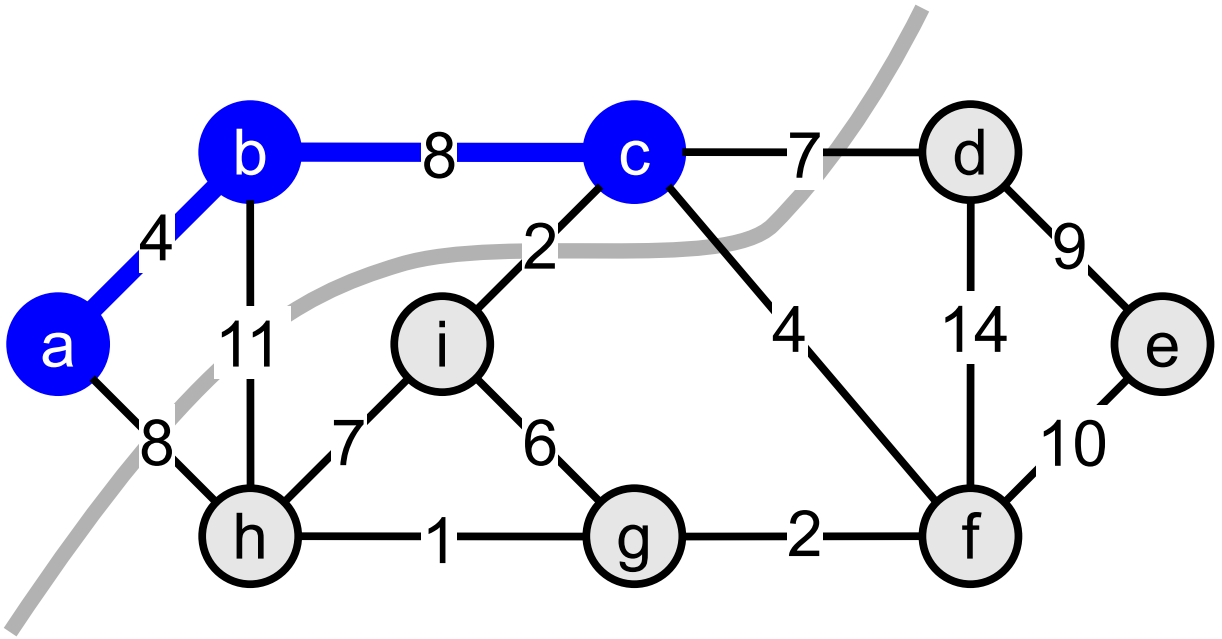
Esempio
Algoritmo di Prim



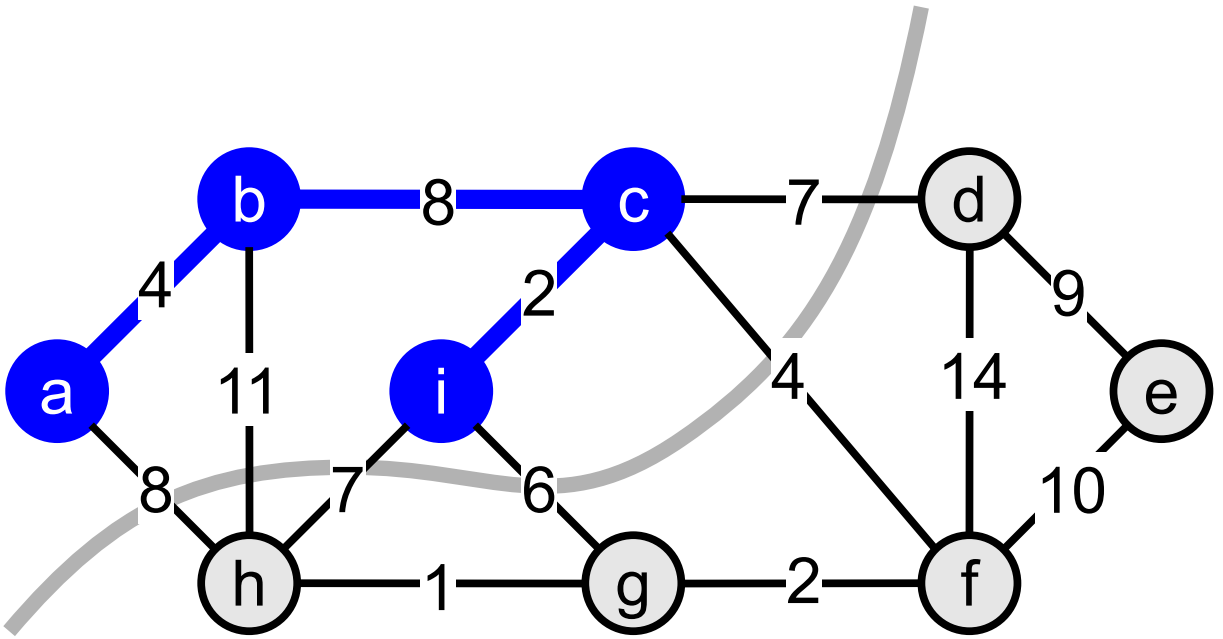
Esempio
Algoritmo di Prim



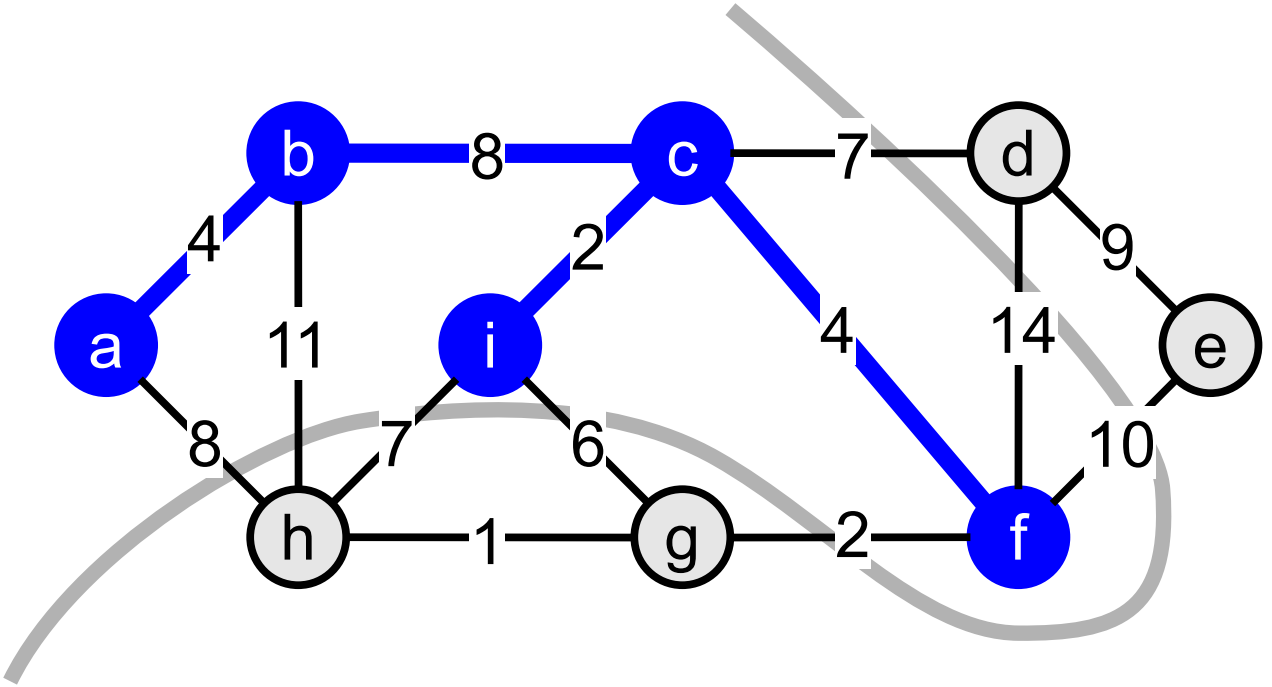
Esempio
Algoritmo di Prim



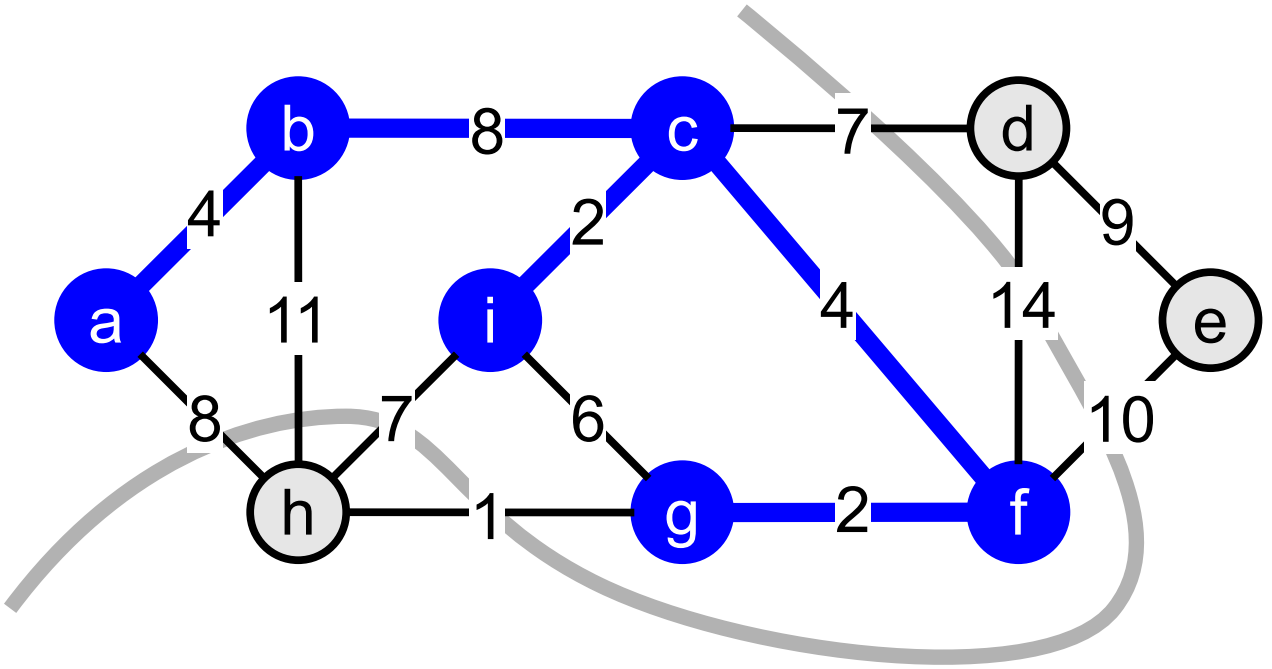
Esempio
Algoritmo di Prim



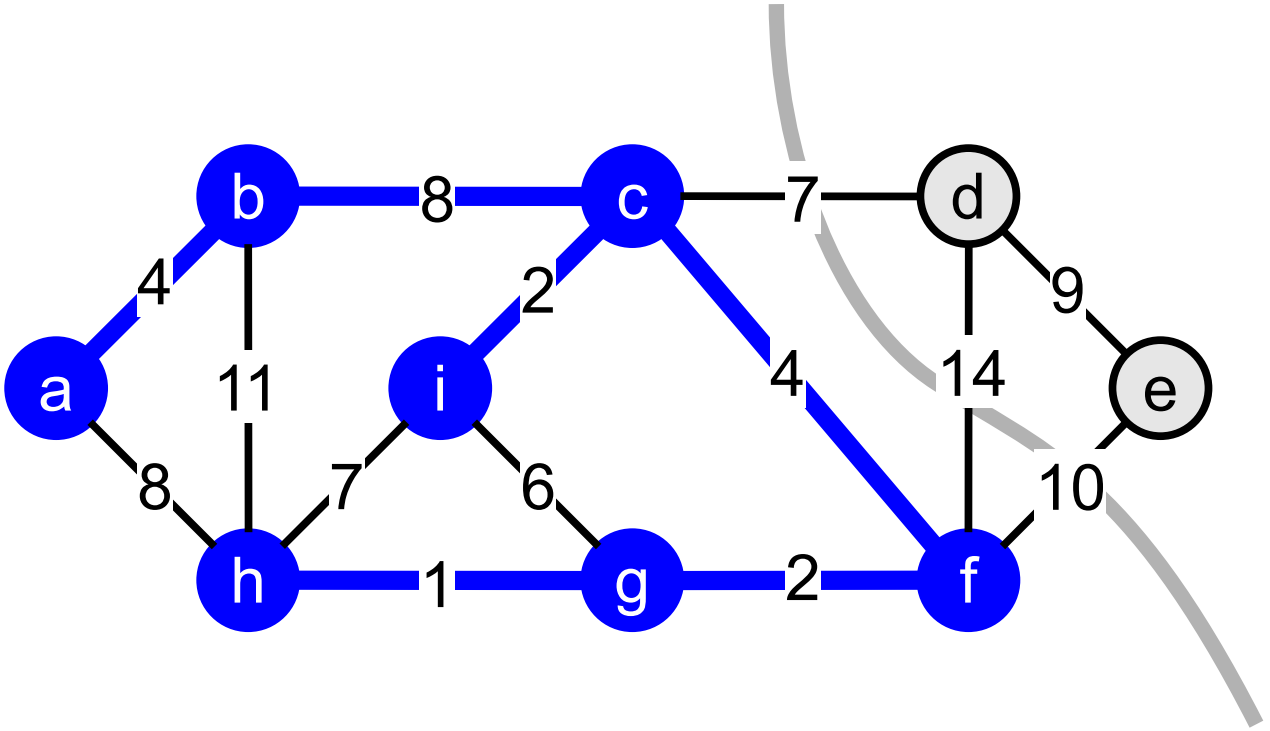
Esempio
Algoritmo di Prim



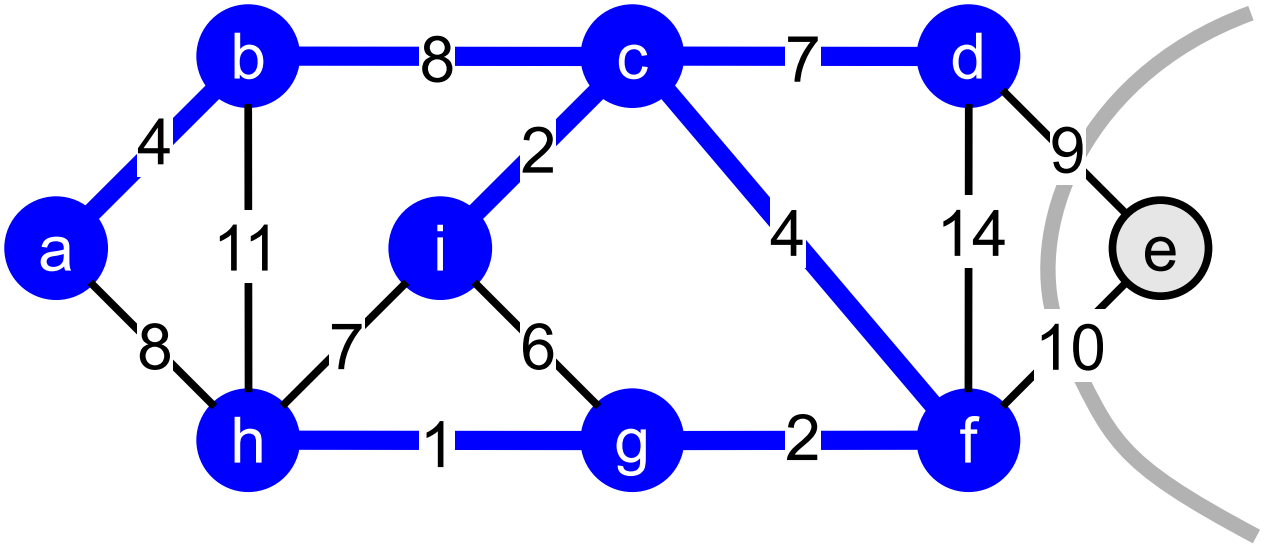
Esempio
Algoritmo di Prim



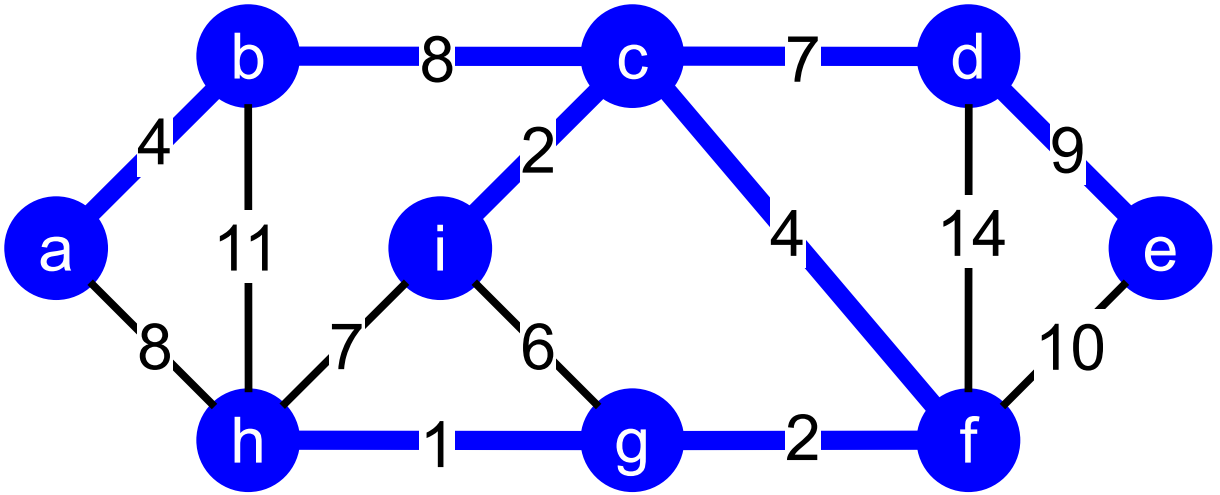
Esempio
Algoritmo di Prim



Esempio
Algoritmo di Prim



Esempio
Algoritmo di Prim



Implementazione

- Una struttura dati per i nodi non ancora nell'albero
  - i nodi non ancora nel MST si trovano in una coda con priorità Q ordinata in base ad un valore $d[v]$
  - $d[v]$ è il peso minimo di un arco che collega il nodo v, che non appartiene all'albero, ad un nodo già nell'albero
    - $+\infty$ se tale arco non esiste
- Come mantenere l'albero
  - Mediante il vettore padri $p[v]$

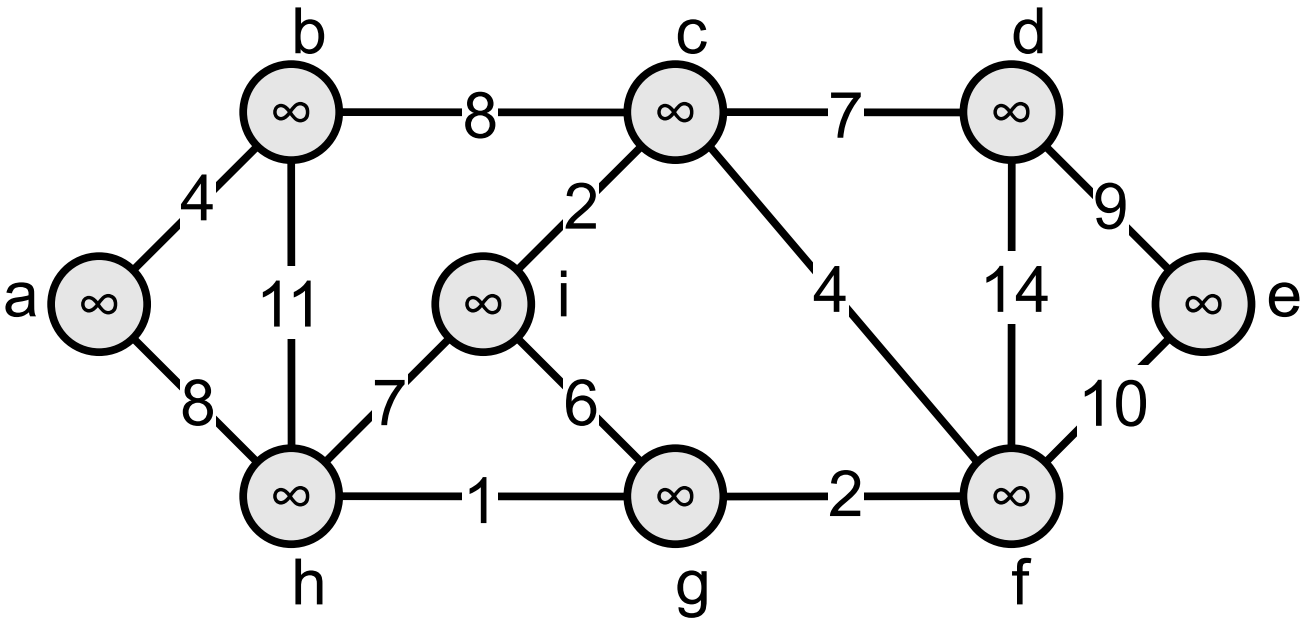
- Terminazione: quando l'insieme Q è vuoto
  - Tutti i nodi tranne la radice conoscono il proprio padre

Algoritmo di Prim

```
int[] Prim-MST(Grafo G=(V,E,w), nodo s)
      double d[1..n];
      int p[1..n];
      boolean added[1..n];
      CodaPriorita<int, double> Q;
      for v ← 1 to n do
            p[v] ← -1; added[v] ← false;
            if (v==s) then d[v] ← 0; else d[v] ← +∞ endif
            Q.insert(v, d[v]);
      endfor
      while (not Q.isEmpty()) do
            u ← Q.find();
            Q.deleteMin();
            added[u] ← true;
            for each v adiacente a u do
                  if (not added[v] and w(u,v) < d[v]) then
                        d[v] ← w(u,v);
                        Q.decreaseKey(v, d[v]);
                        p[v] ← u;
                  endif
            endfor
      endwhile
      return p;
```

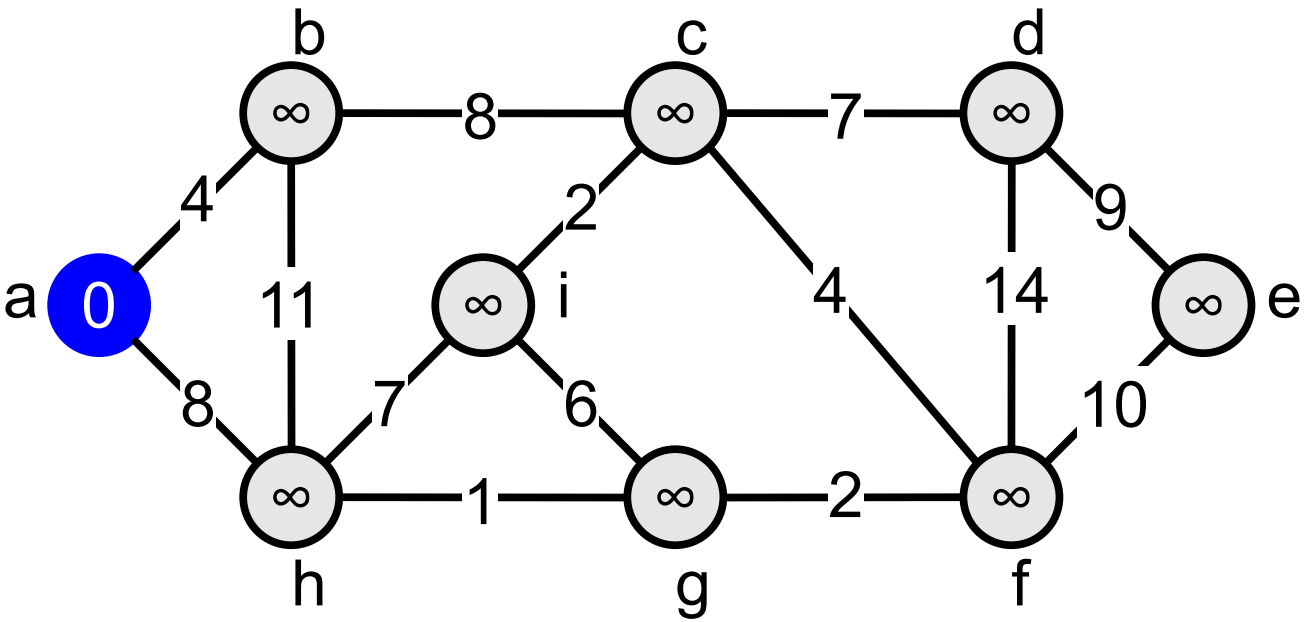
Esempio di esecuzione

Q = { }



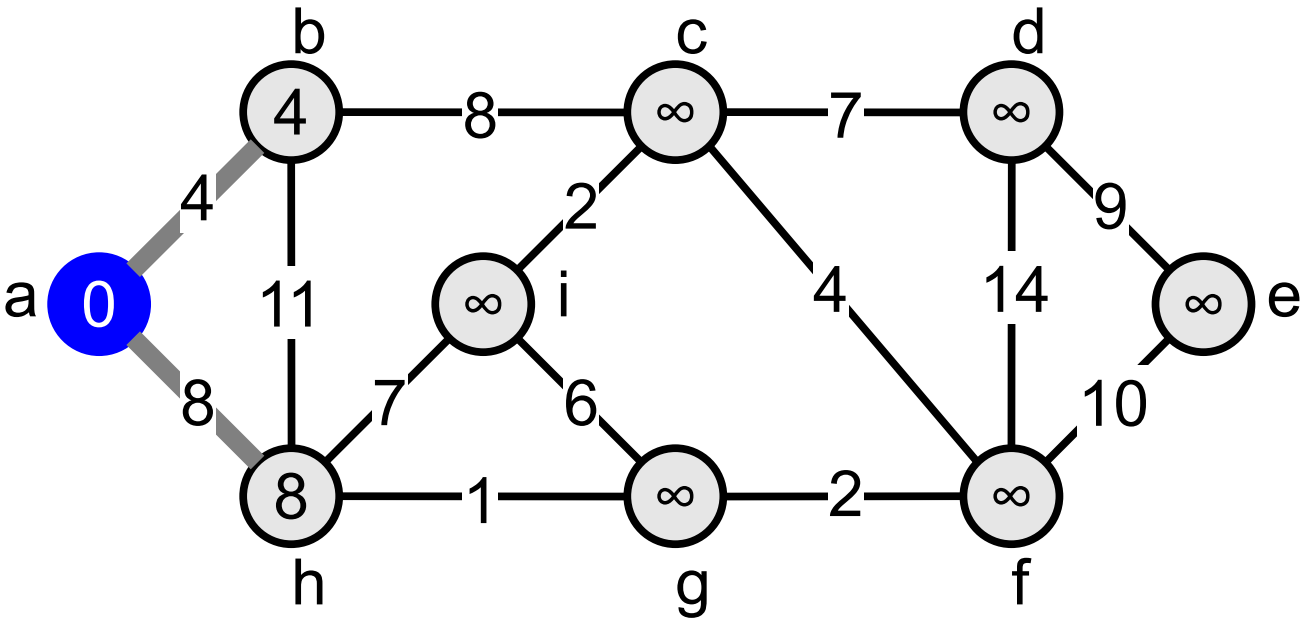
Esempio di esecuzione

Q = { (a,0) }



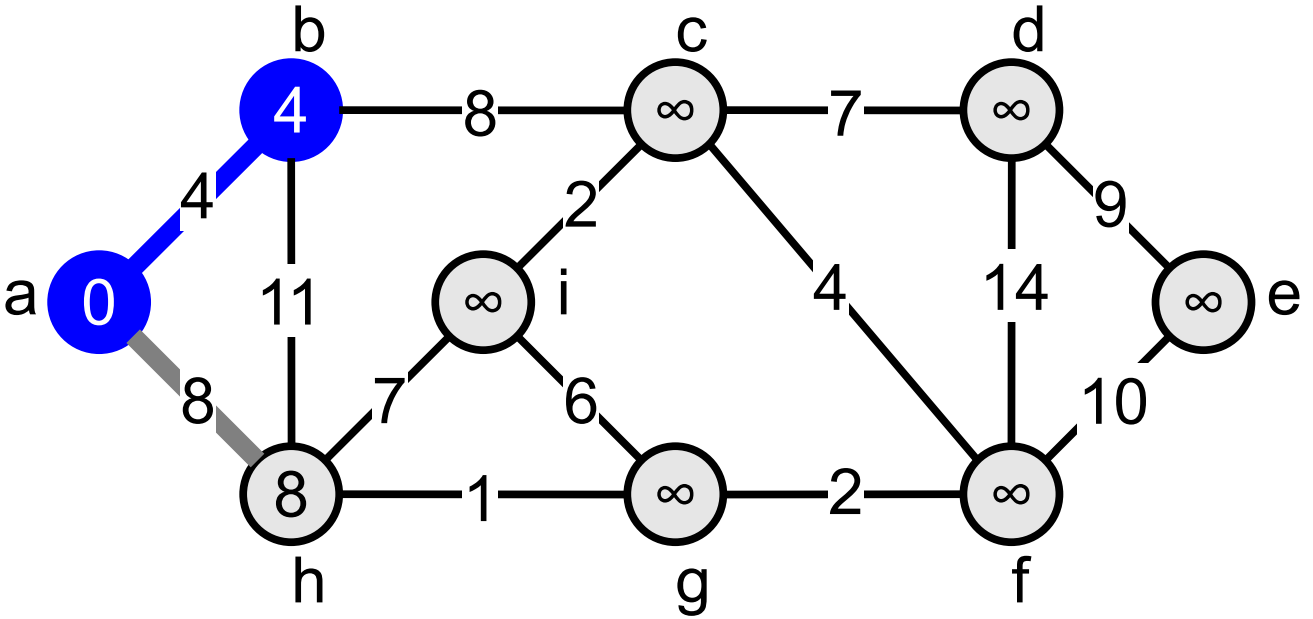
Esempio di esecuzione

Q = { (b,4), (h,8) }



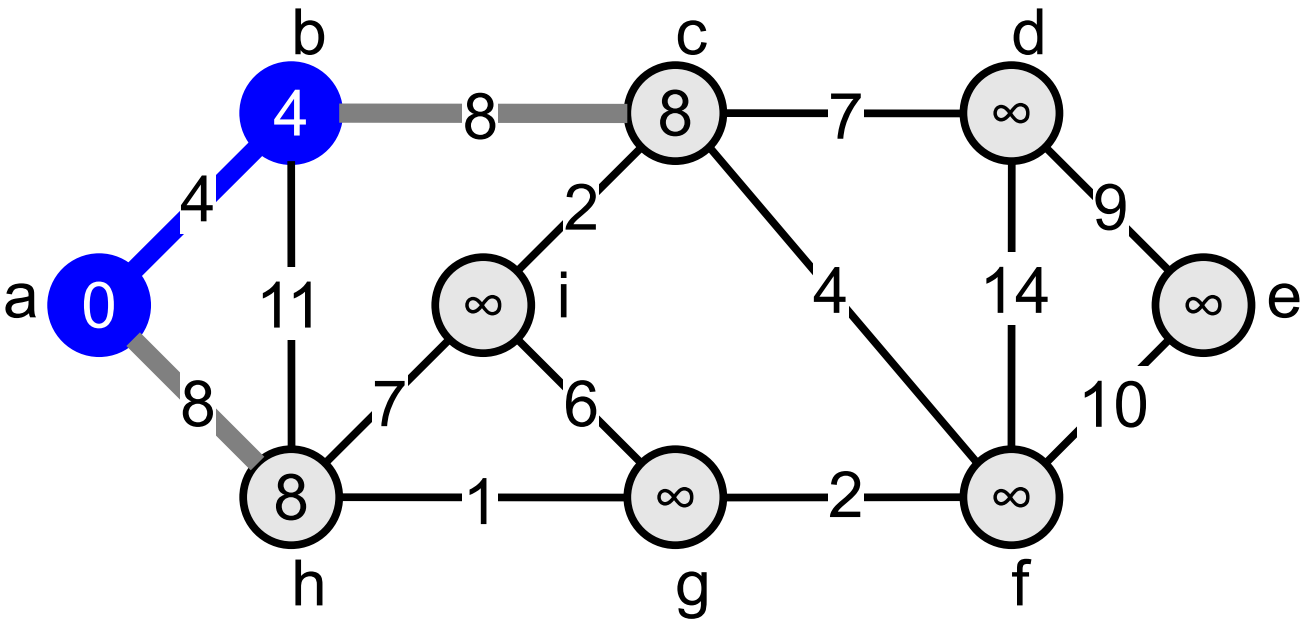
Esempio di esecuzione

Q = { (b,4), (h,8) }



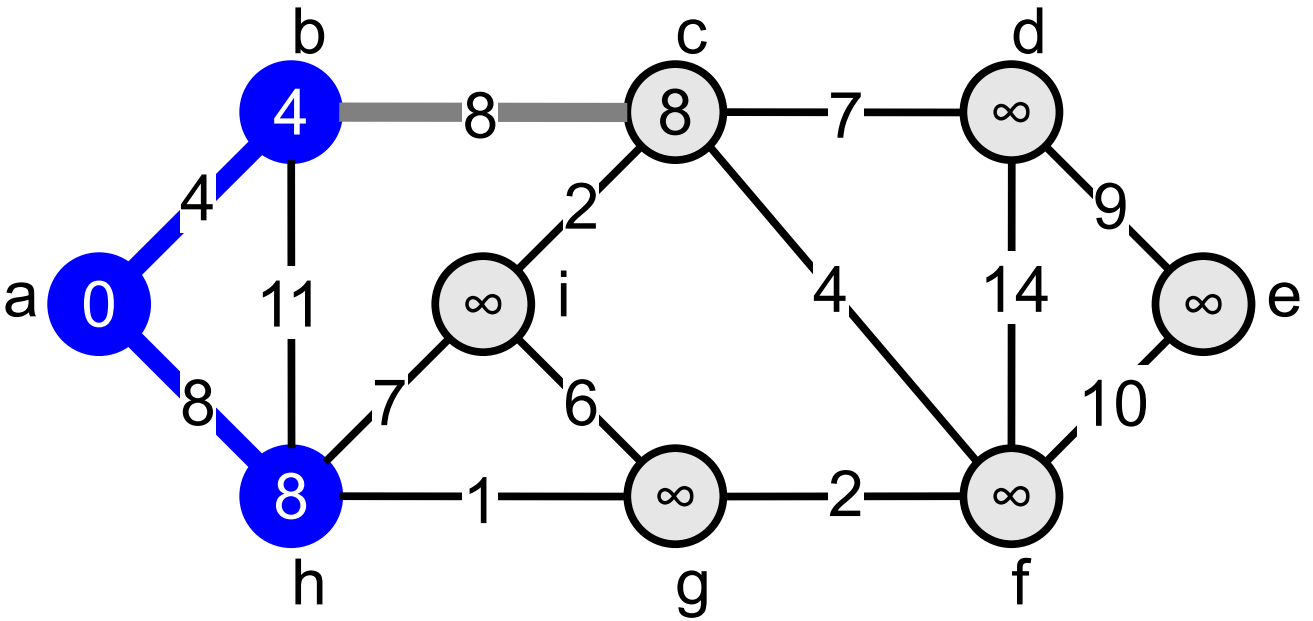
Esempio di esecuzione

Q = { (h,8), (c,8) }



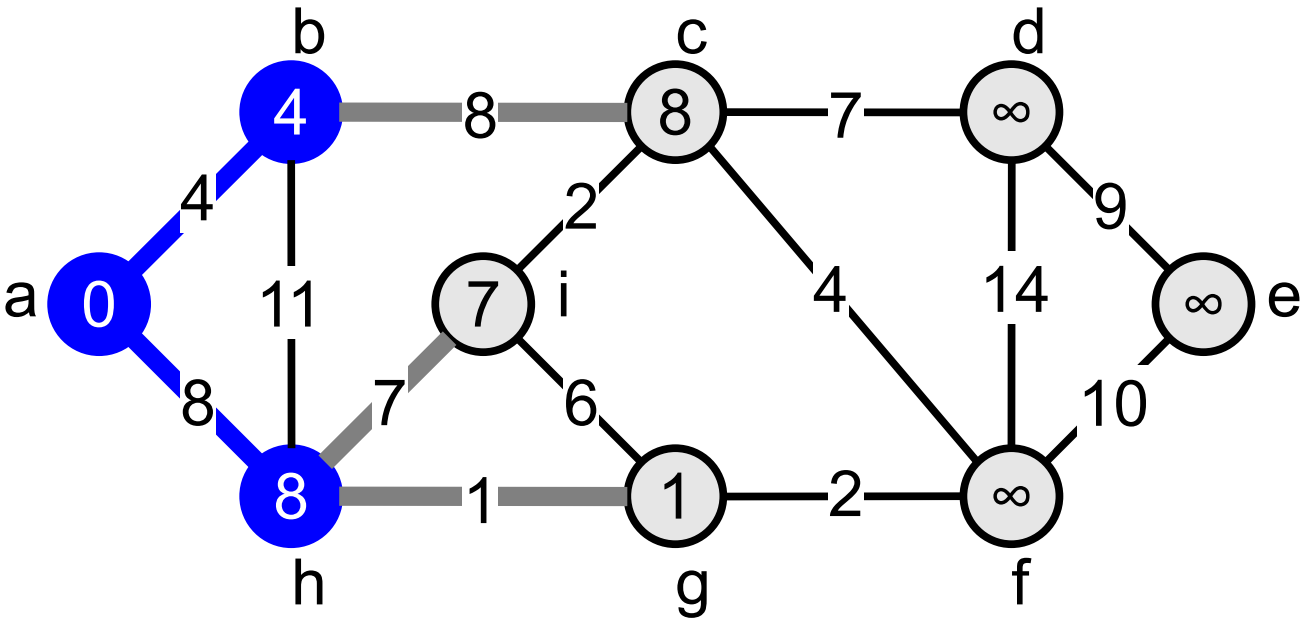
Esempio di esecuzione

Q = { (h,8), (c,8) }



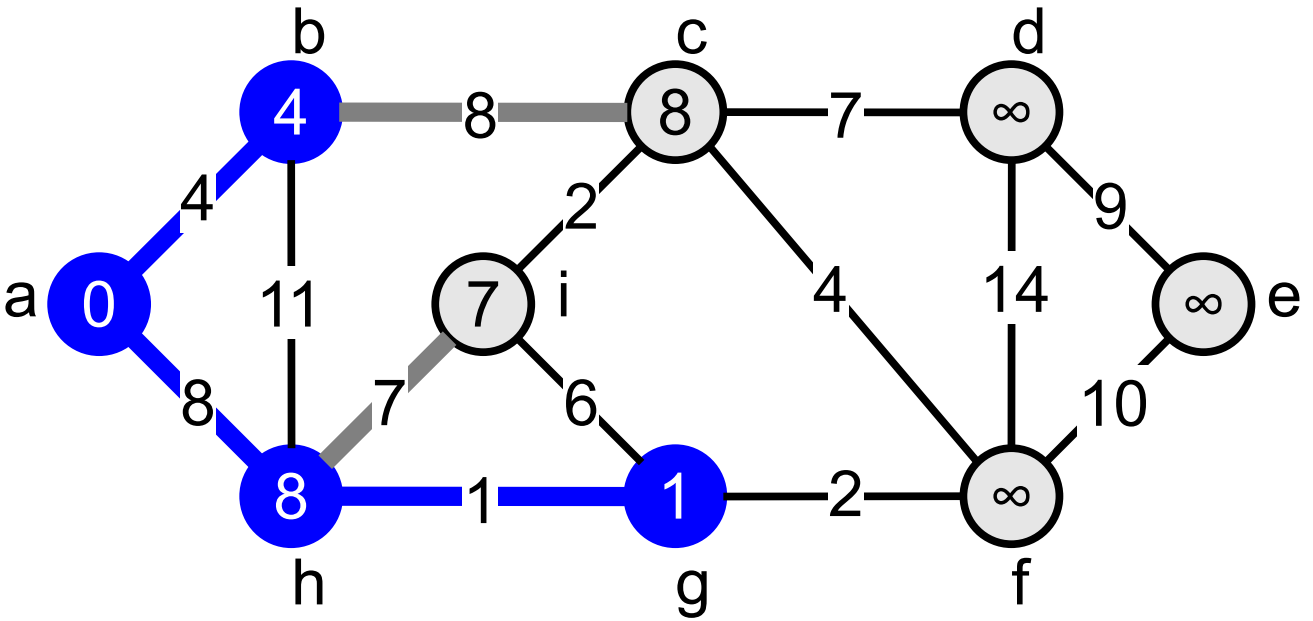
Esempio di esecuzione

Q = { (g,1), (i,7), (c,8) }



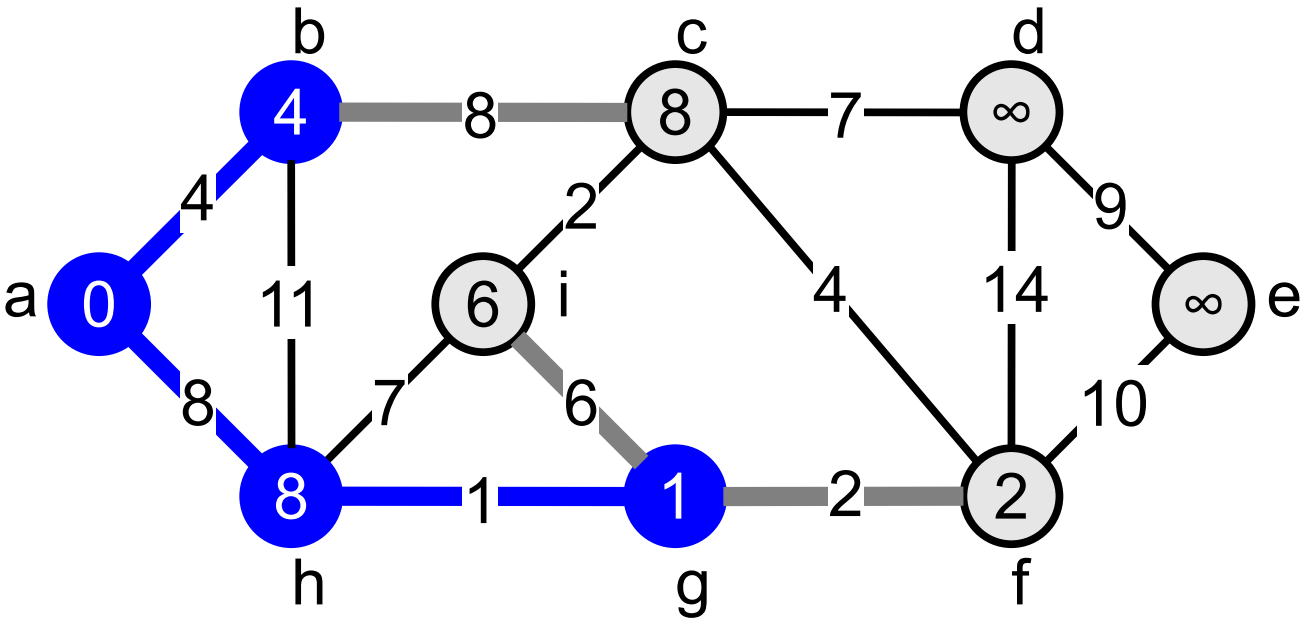
Esempio di esecuzione

Q = { (g,1), (i,7), (c,8) }



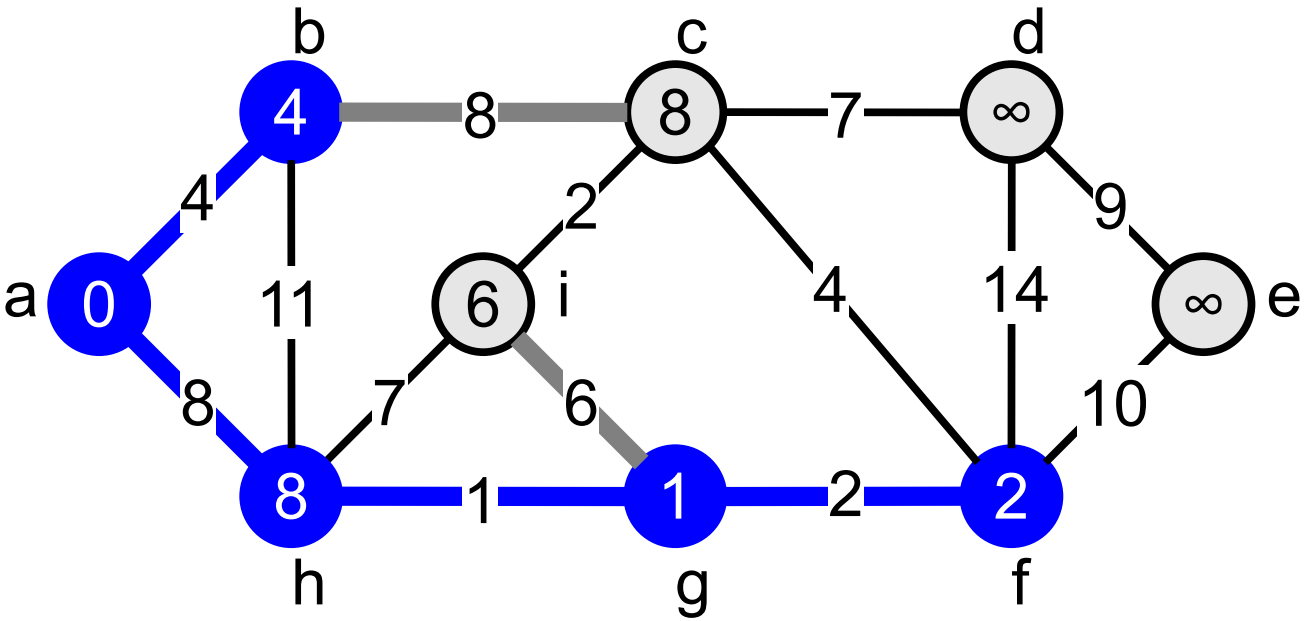
Esempio di esecuzione

Q = { (f,2), (i,6), (c,8) }



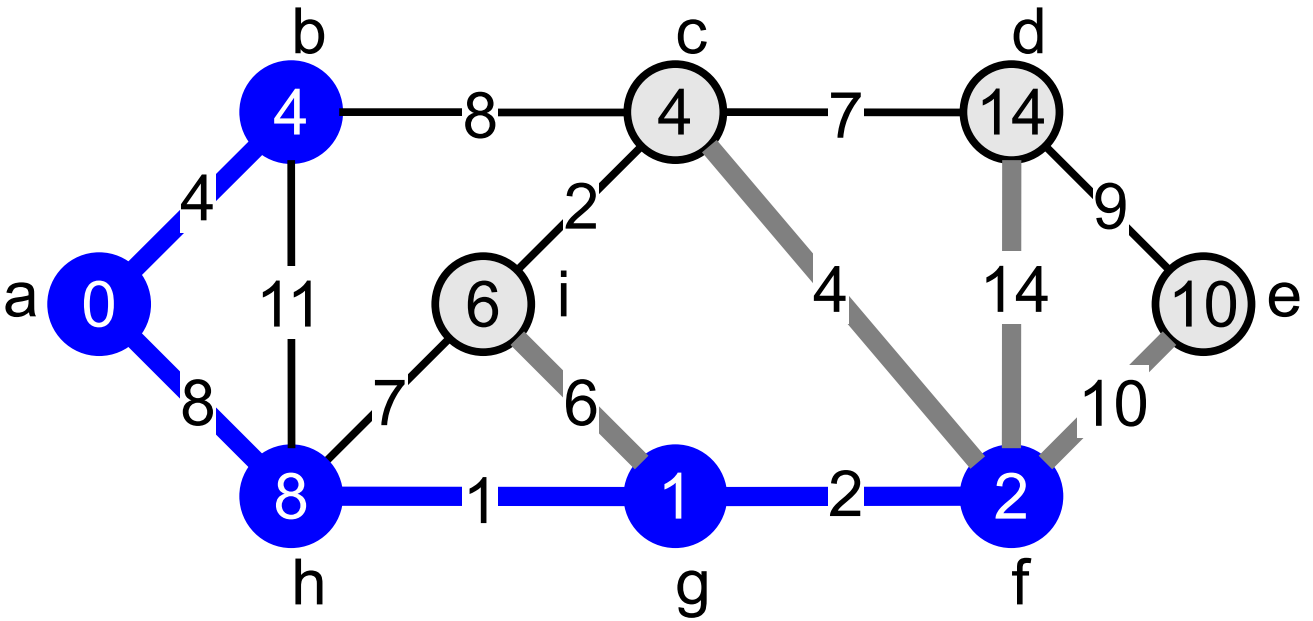
Esempio di esecuzione

Q = { (f,2), (i,6), (c,8) }



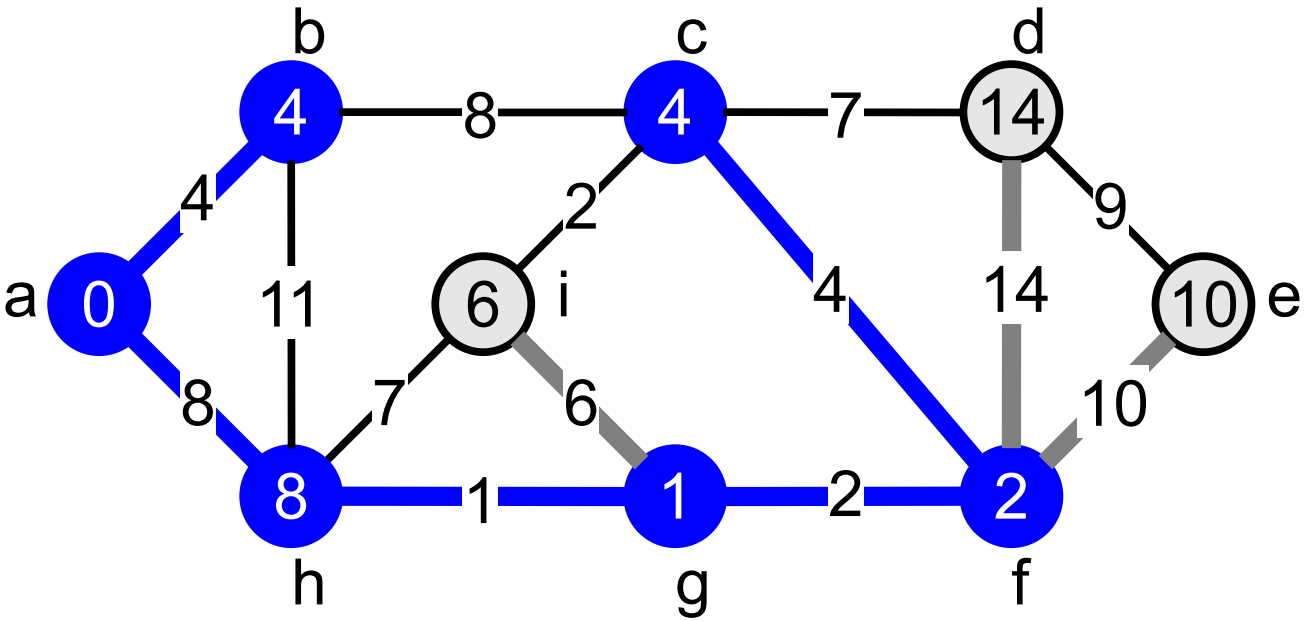
Esempio di esecuzione

Q = { (c,4), (i,6), (e,10), (d,14) }



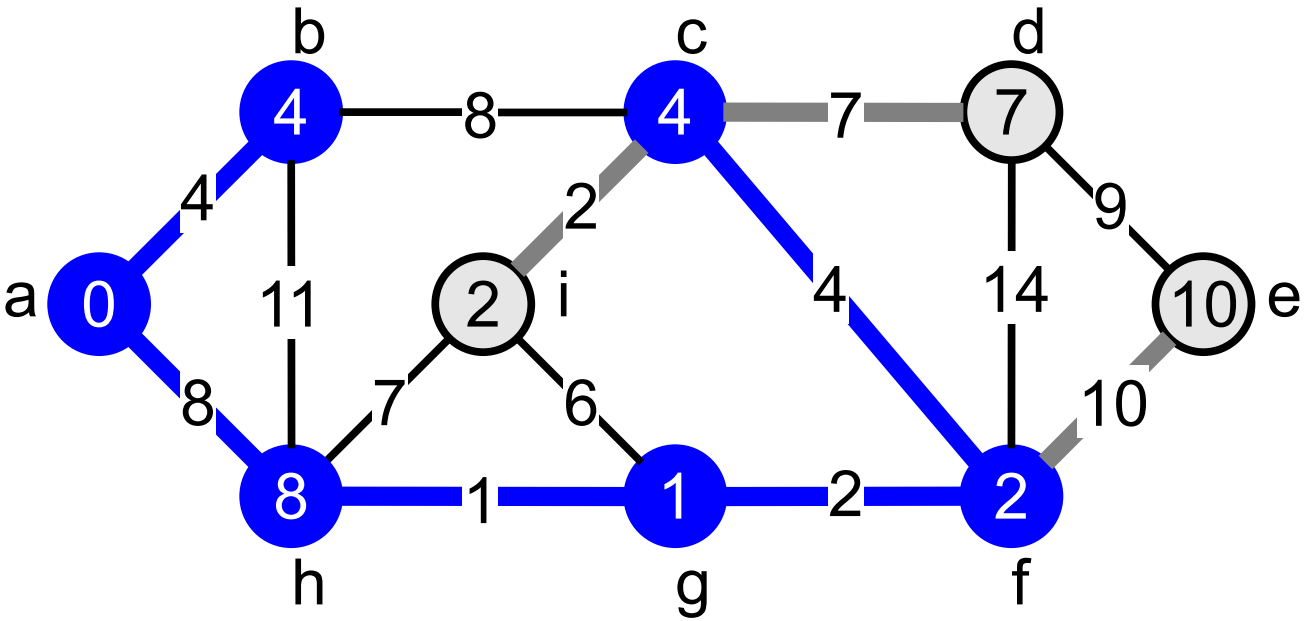
Esempio di esecuzione

Q = { (c,4), (i,6), (e,10), (d,14) }



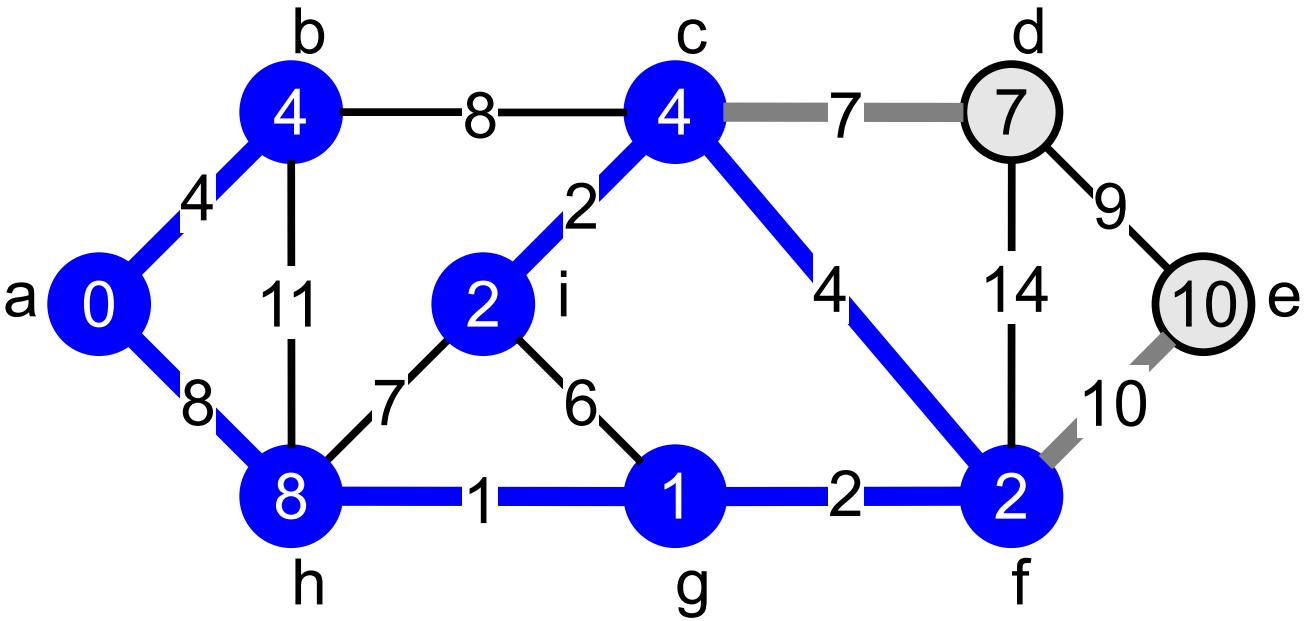
Esempio di esecuzione

Q = { (i,2), (d,7), (e,10) }



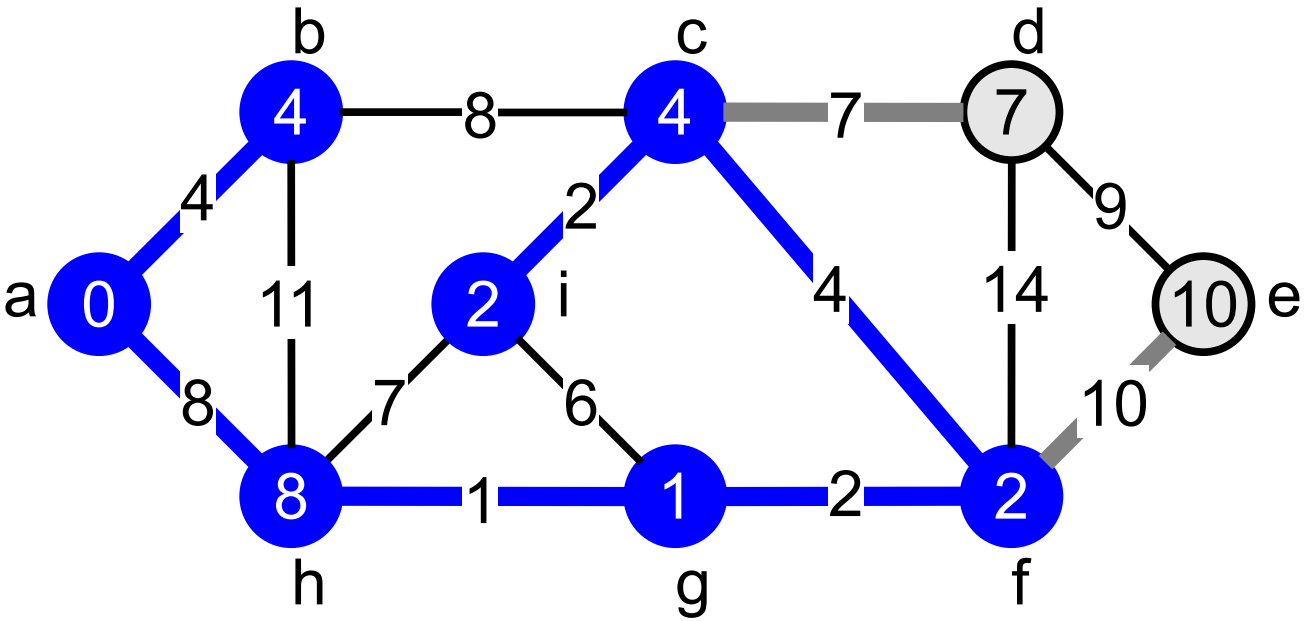
Esempio di esecuzione

Q = { (i,2), (d,7), (e,10) }



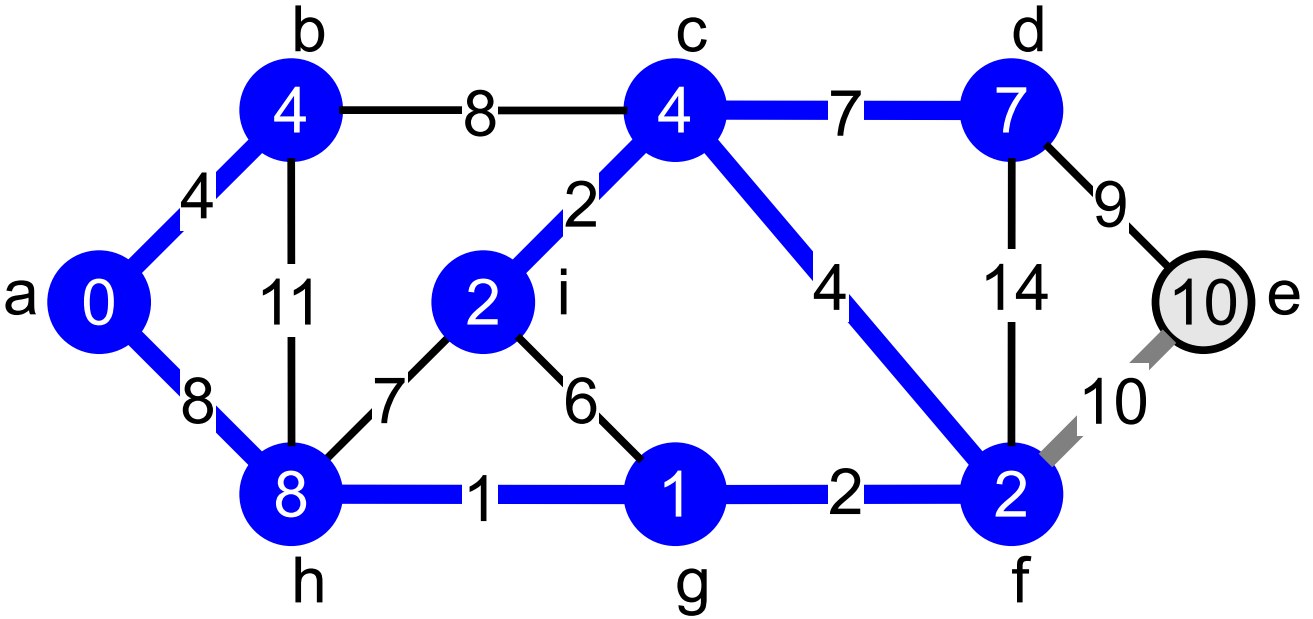
Esempio di esecuzione

Q = { (d,7), (e,10) }



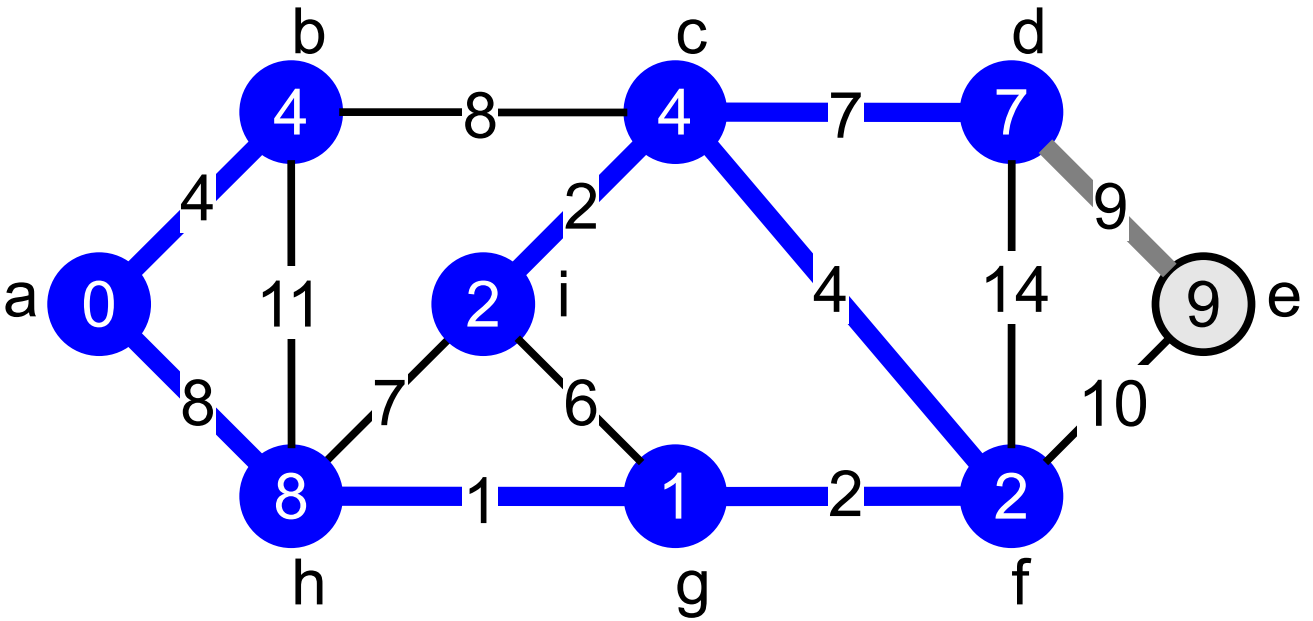
Esempio di esecuzione

Q = { (d,7), (e,10) }



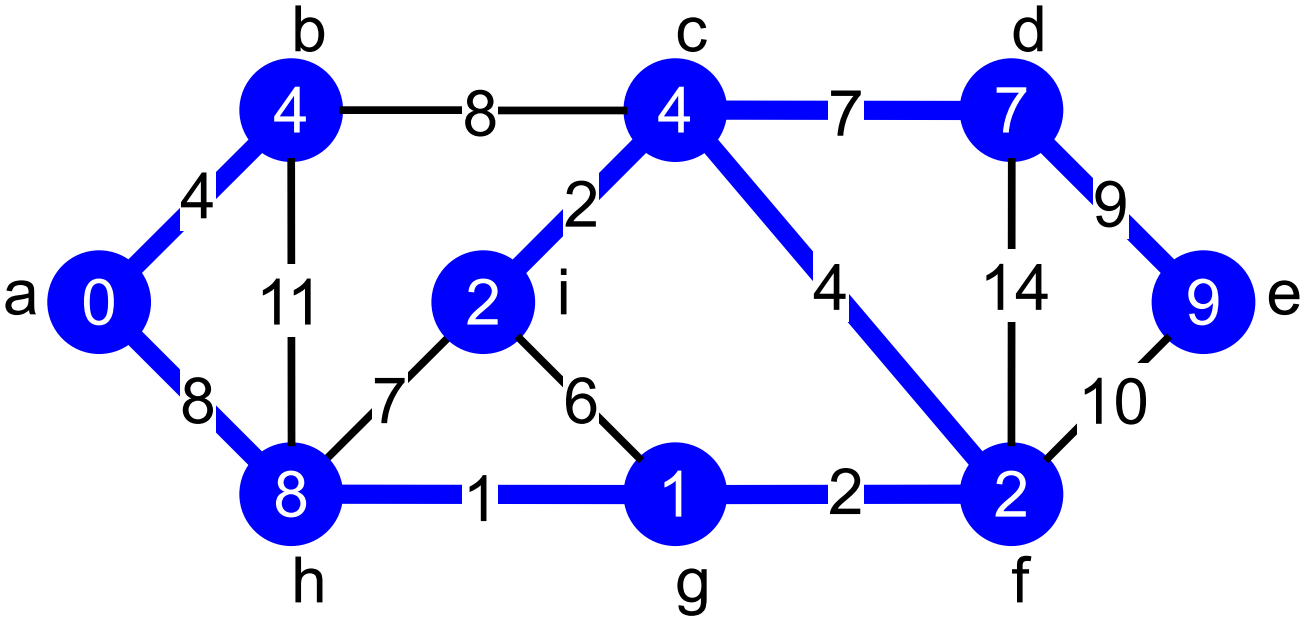
Esempio di esecuzione

Q = { (e,9) }



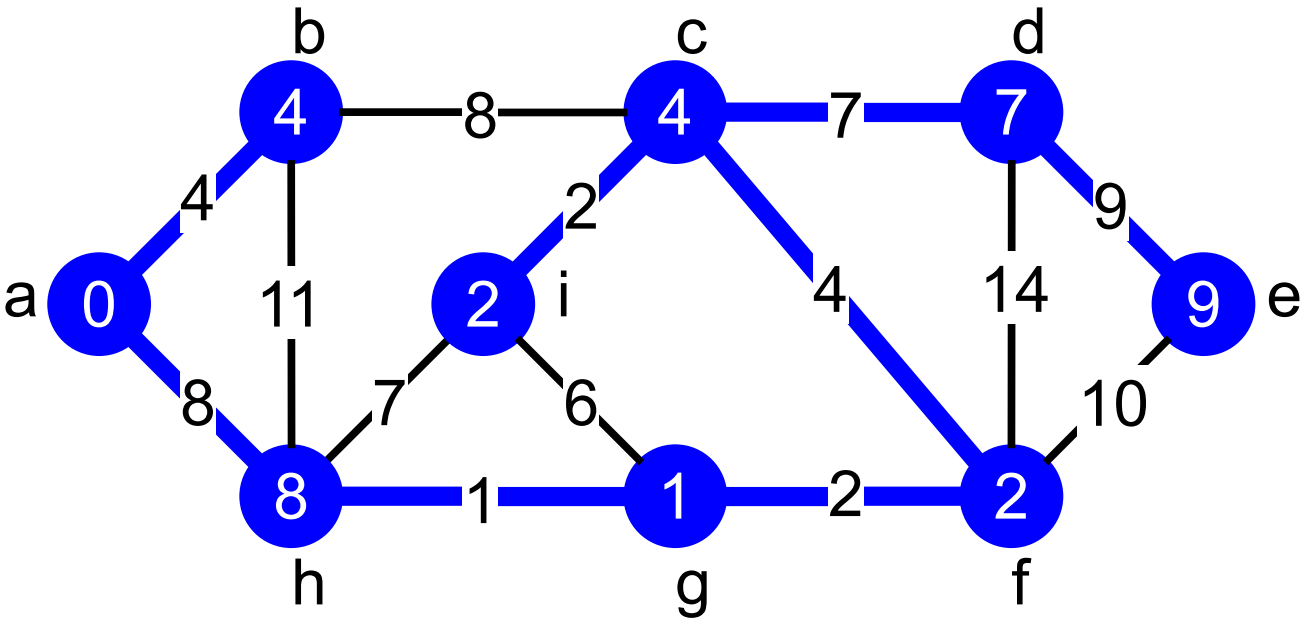
Esempio di esecuzione

Q = { (e,9) }



Esempio di esecuzione

Q = { }



```
int[] Prim-MST(Grafo G=(V,E,w), nodo s)
      double d[1..n];
      int p[1..n];
      boolean added[1..n];
      CodaPriorita<int, double> Q;
      for v ← 1 to n do
            p[v] ← -1; added[v] ← false;
            if (v==s) then d[v] ← 0; else d[v] ← +∞ endif
            Q.insert(v, d[v]);
      endfor
      while (not Q.isEmpty()) do
            u ← Q.find();
            Q.deleteMin();
            added[u] ← true;
            for each v adiacente a u do
                  if (not added[v] and w(u,v) < d[v]) then
                        d[v] ← w(u,v);
                        Q.decreaseKey(v, d[v]);
                        p[v] ← u;
                  endif
            endfor
      endwhile
      return p;
```

n insert()

n find()

n deleteMin()

O(*m*) decreaseKey()

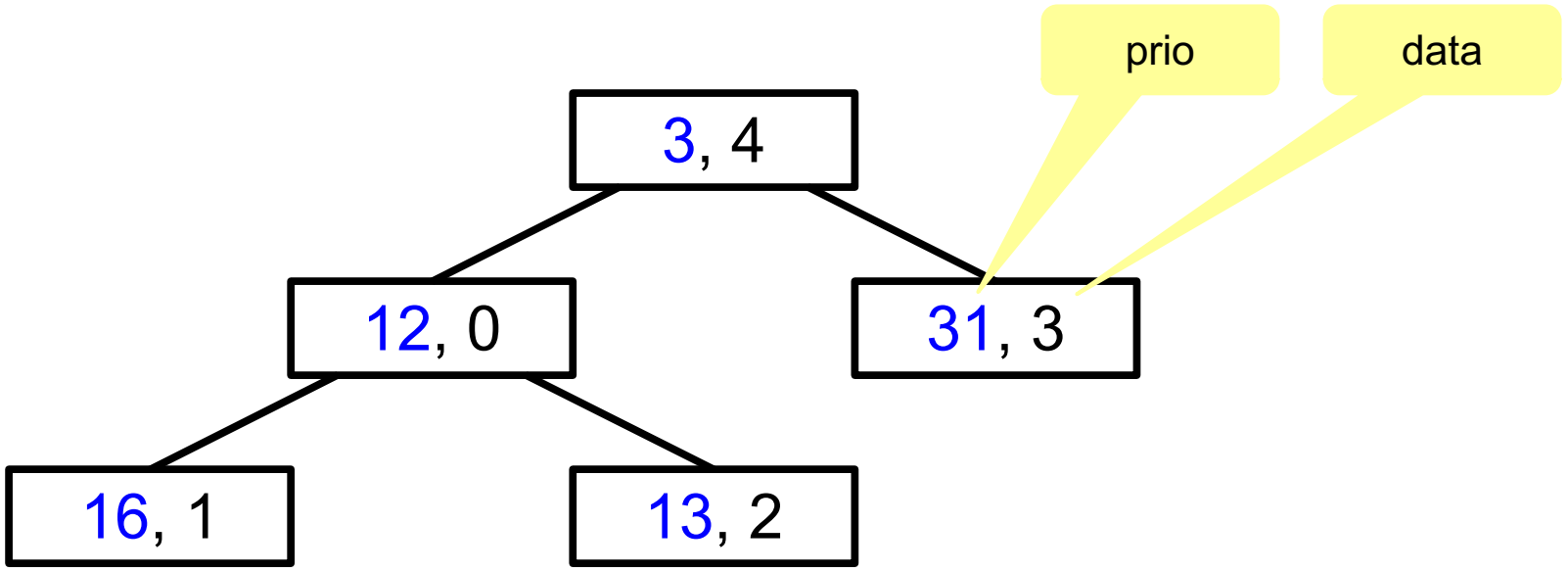
Algoritmo di Prim
Costo computazionale

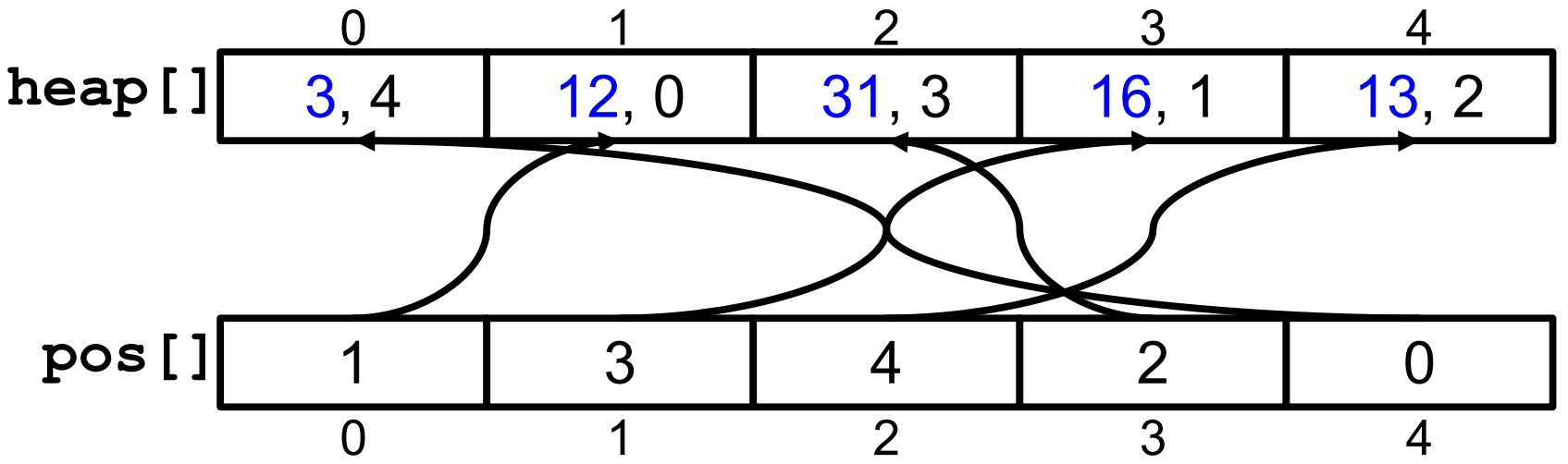
- Utilizzando una coda di priorità basata su min-heap binario
  - n **`deleteMin()`** costano $O(n \log n)$
  - n **`insert()`** costano $O(n \log n)$
  - n **`find()`** costano $O(n)$
  - $O(m)$ **`decreaseKey()`** costano $O(m \log n)$

- Costo computazionale totale:
  - $O(2n \log n + n + m \log n) =$
    $O((n + m) \log n) =$
    $O(m \log n)$ (se il grafo è connesso)

In un grafo connesso si ha sempre $m \geq n - 1$

MinHeap


prio
data
3, 4
12, 0
31, 3
16, 1
13, 2


0
1
2
3
4
heap[]
3, 4
12, 0
31, 3
16, 1
13, 2
pos[]
1
3
4
2
0
0
1
2
3
4