Minimum Spanning Tree

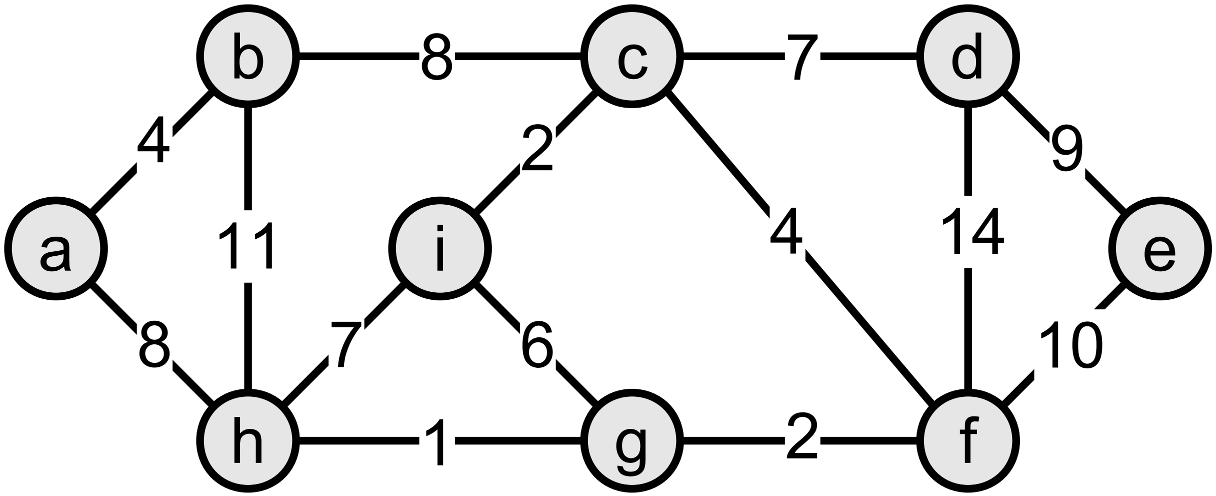
Gianluigi Zavattaro
Dip. di Informatica – Scienza e Ingegneria
Università di Bologna
gianluigi.zavattaro@unibo.it

Introduzione

- Un problema di notevole importanza:
  - determinare come interconnettere diversi elementi fra loro 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

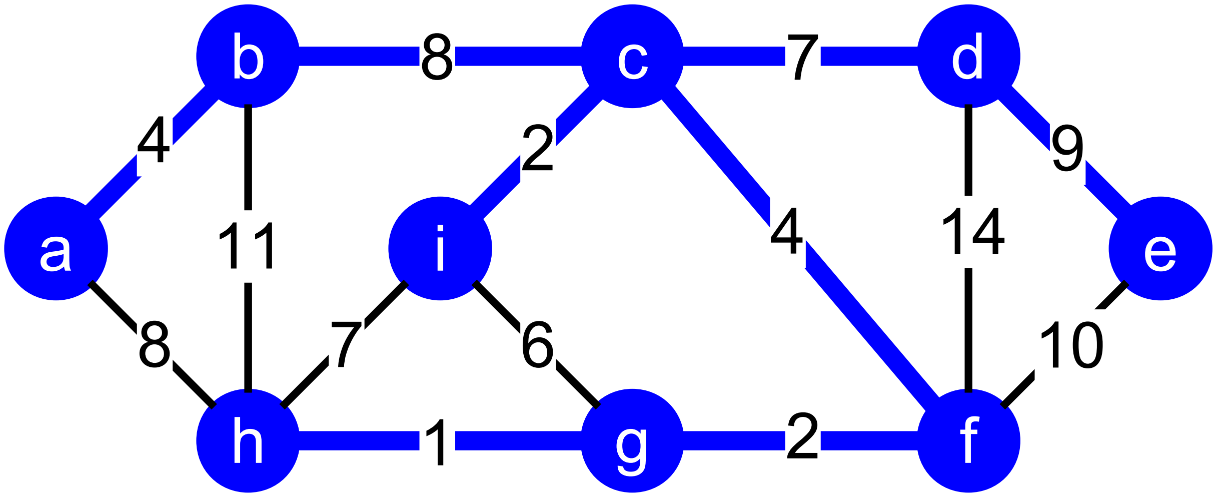
Definizione del problema

- 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

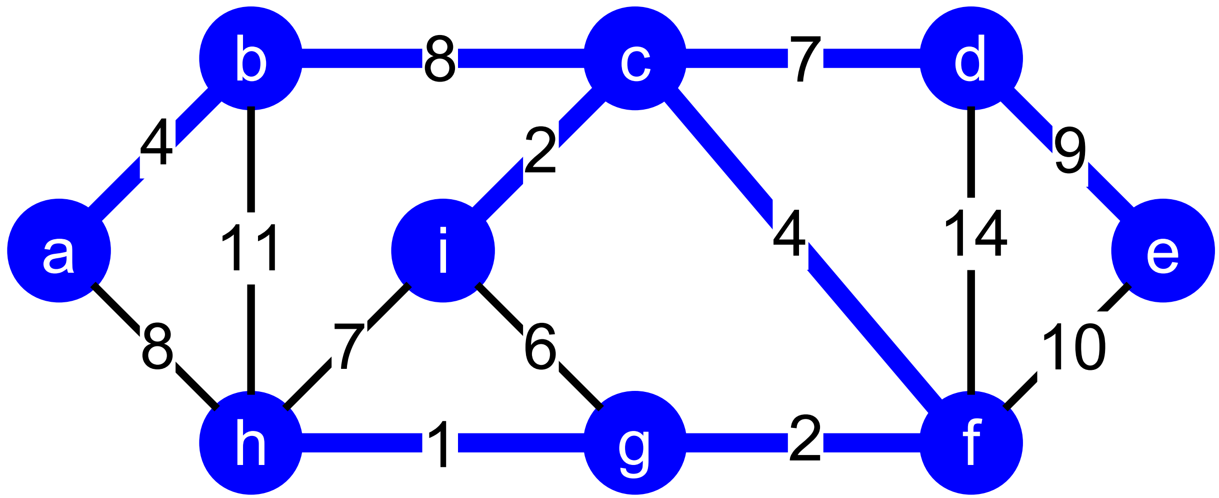
- Output: albero di copertura di peso minimo (minimum spanning tree)
  - Un albero di copertura T il cui peso totale

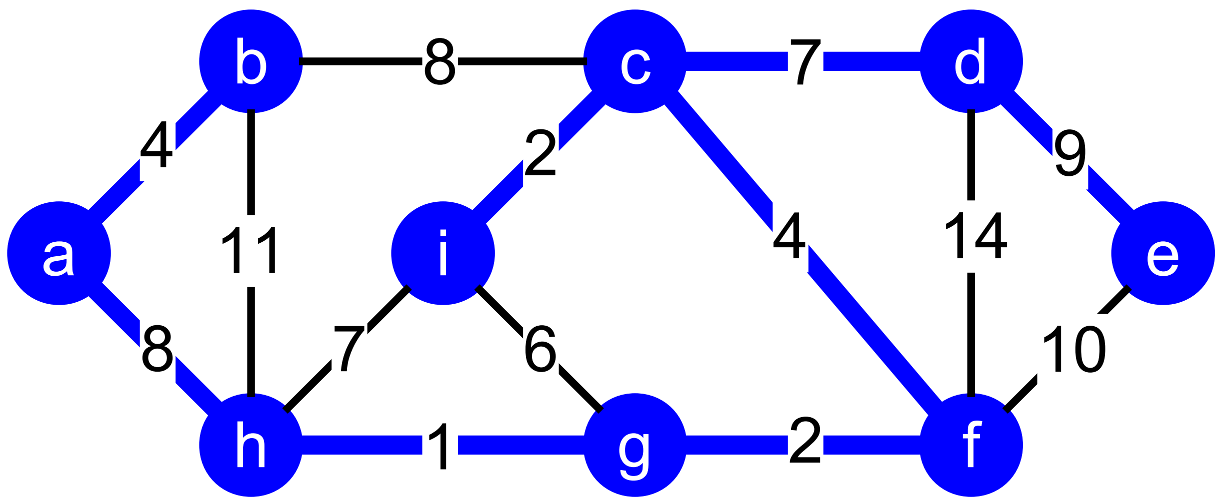
$$w(T) = \sum_{(u,v)\in T} w(u,v)$$

sia minimo, tra tutti i possibili alberi di copertura

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, 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

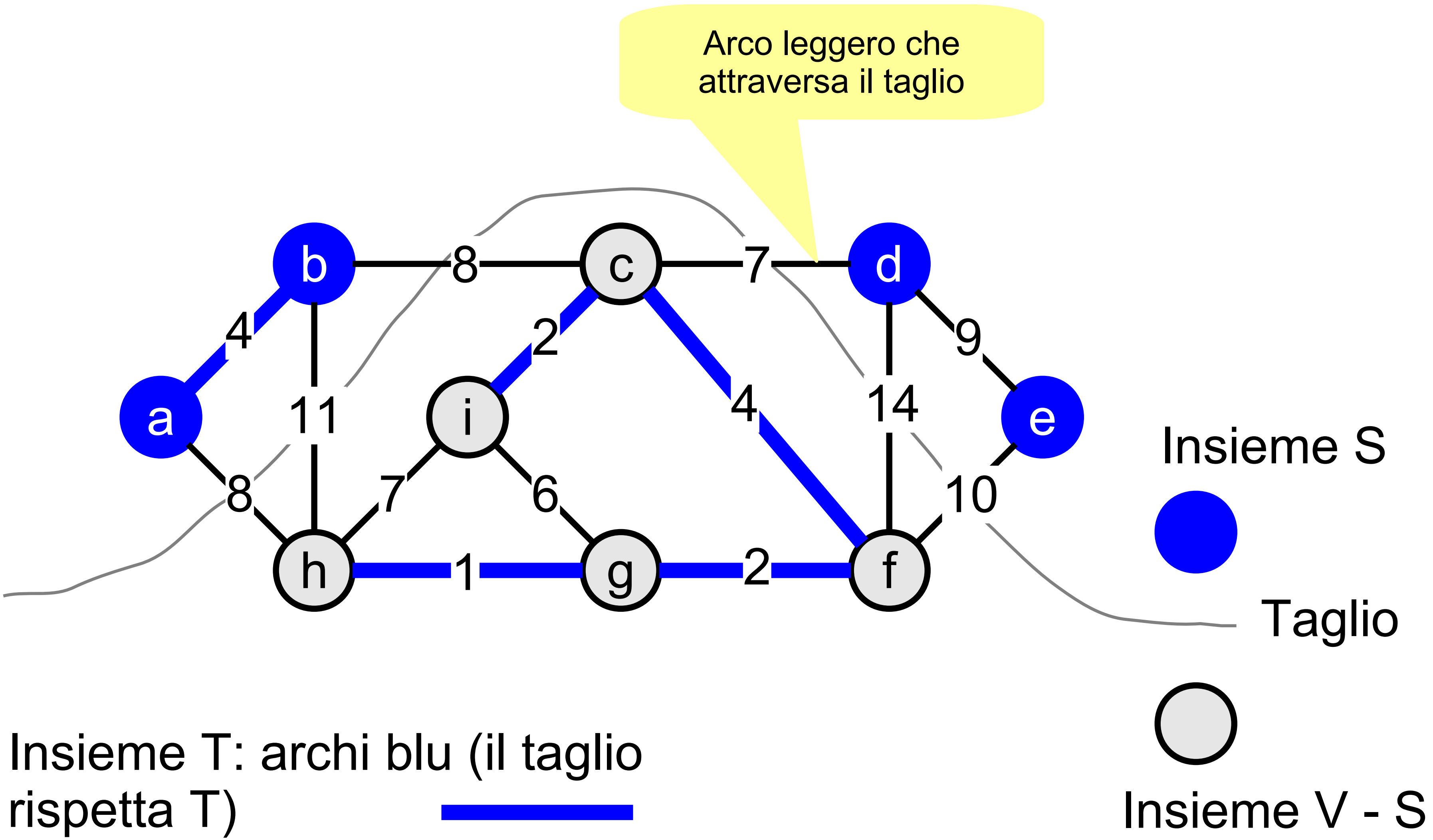
Definizioni

- 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 degli archi che attraversano un taglio

Regole del ciclo e del taglio

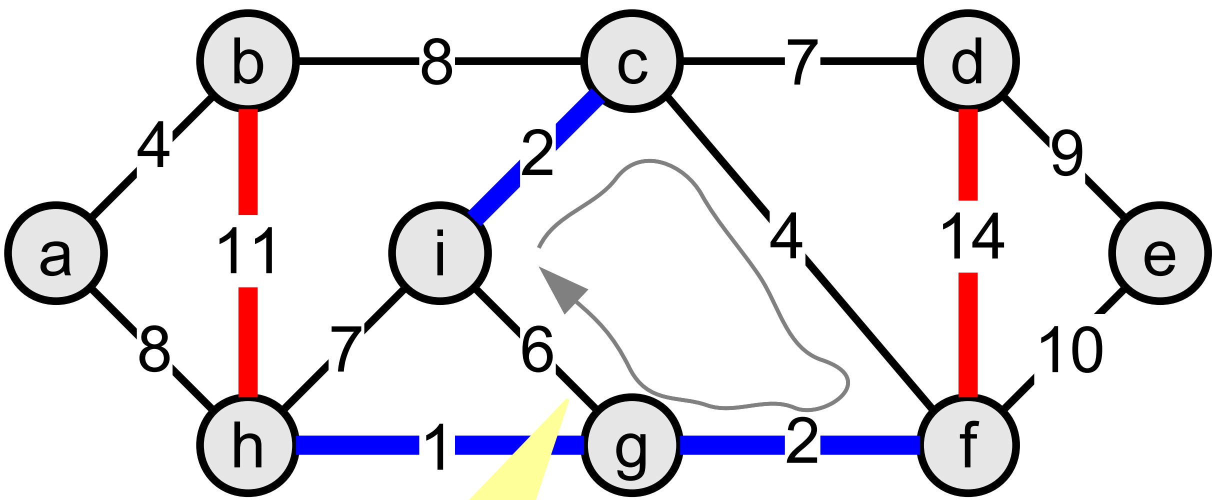
- Regola del taglio
  - Scegli un taglio in *G* che rispetta gli archi già colorati di blu (non attraversato da archi blu). Tra tutti gli archi non colorati che attraversano il taglio selezionane uno leggero (di peso minimo) e coloralo di blu
- Regola del ciclo
  - Scegli un ciclo semplice in *G* che non contenga archi rossi. Tra tutti gli archi non colorati del ciclo, seleziona un arco di costo massimo e coloralo di rosso
- Si può costruire un MST usando tali regole:
  - Costruisce un MST applicando in successione una delle due regole precedenti (una qualunque, purché si possa usare)

Applicazione regola del taglio



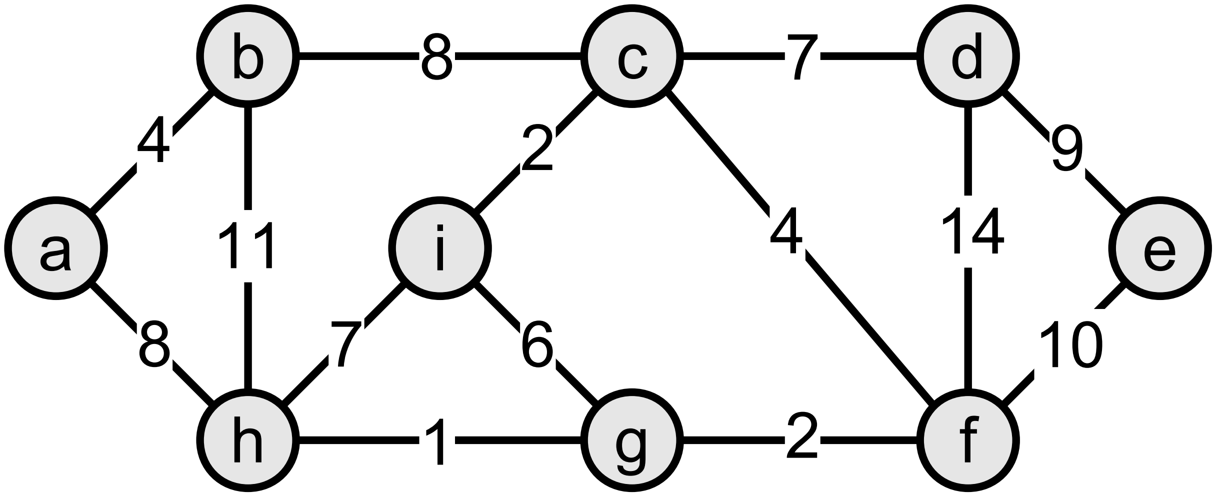
Applicazione regola del ciclo

Seleziona un ciclo semplice
che non contiene archi rossi

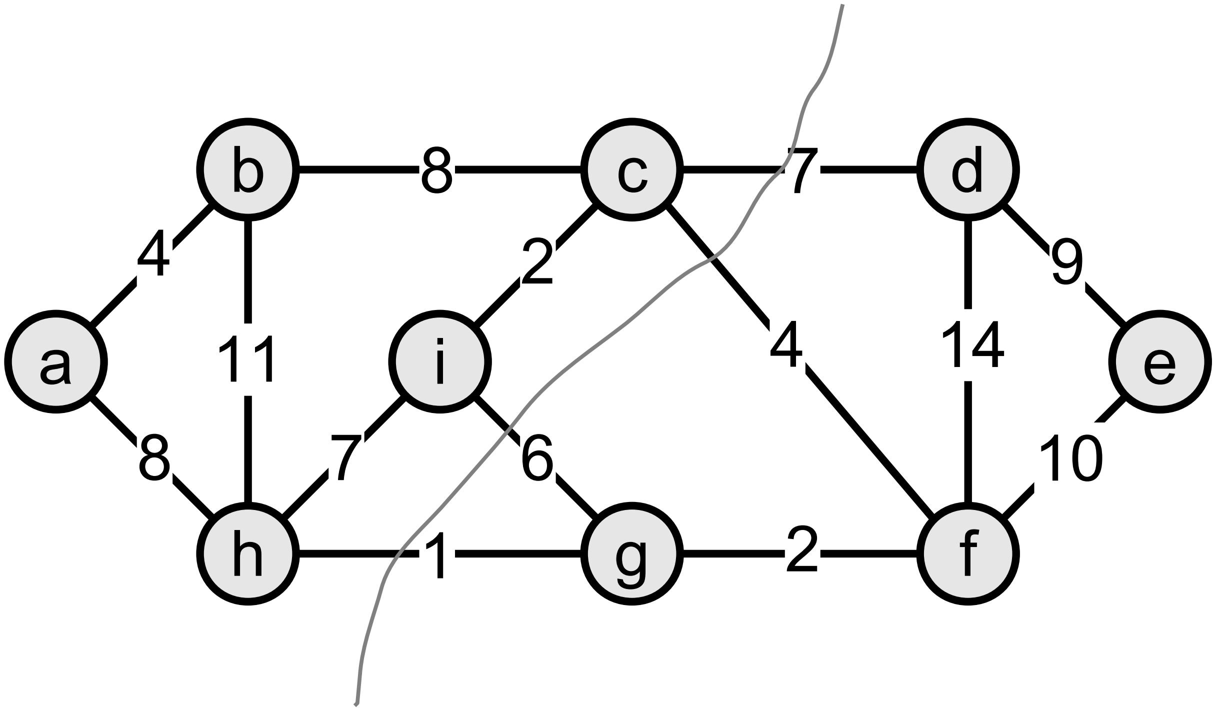


L'arco di peso massimo
del ciclo può essere
colorato di rosso

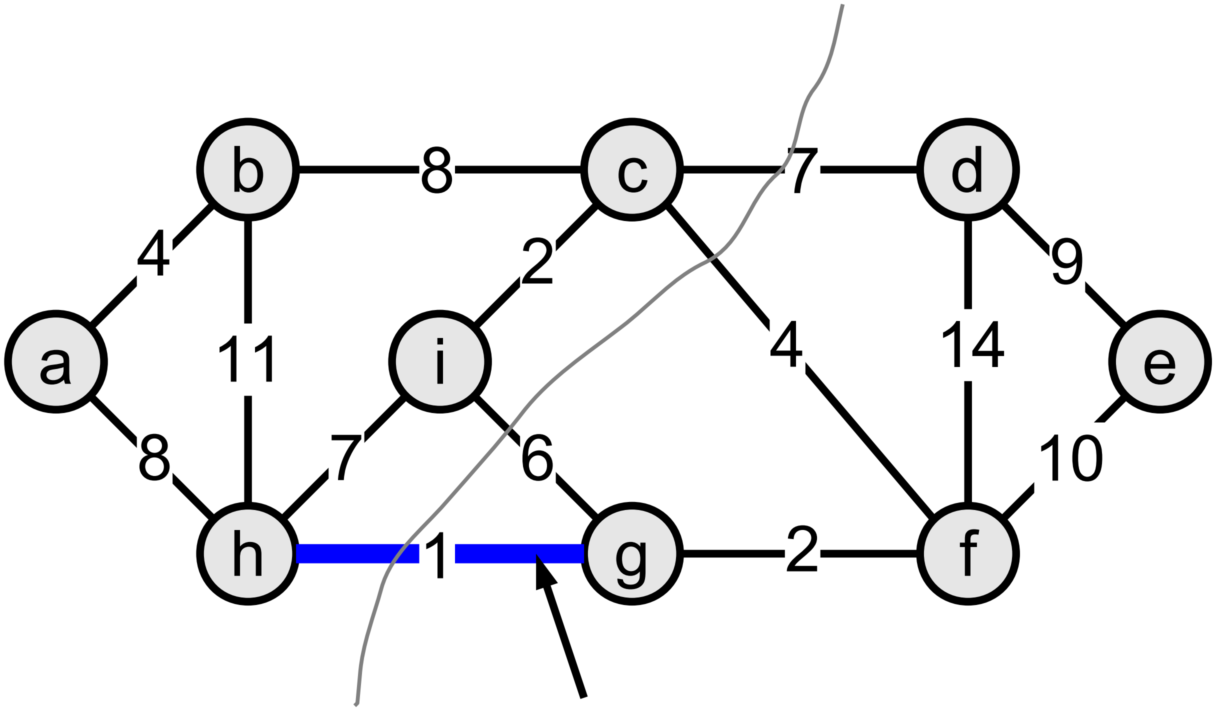
Esempio



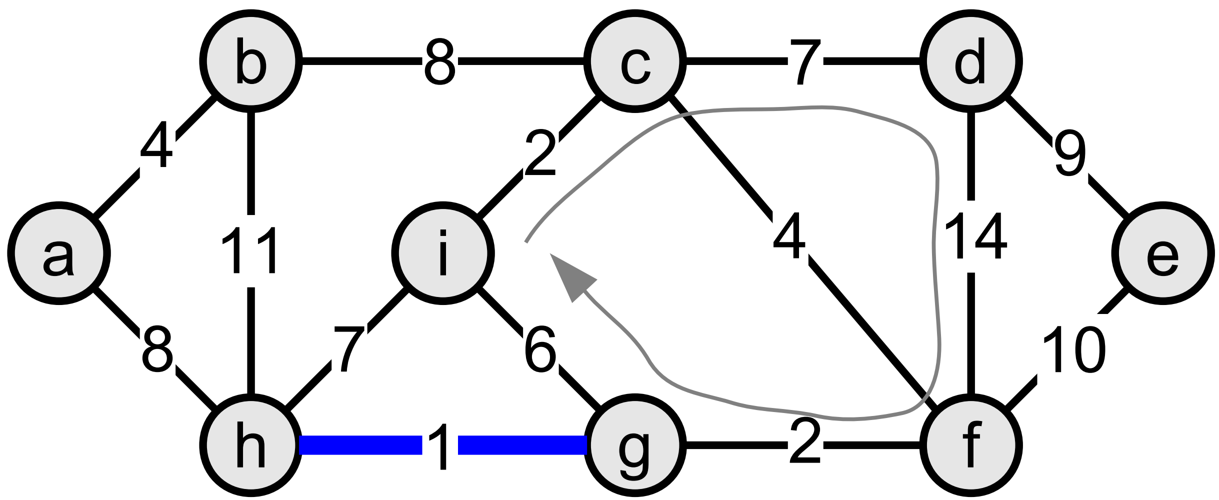
Esempio



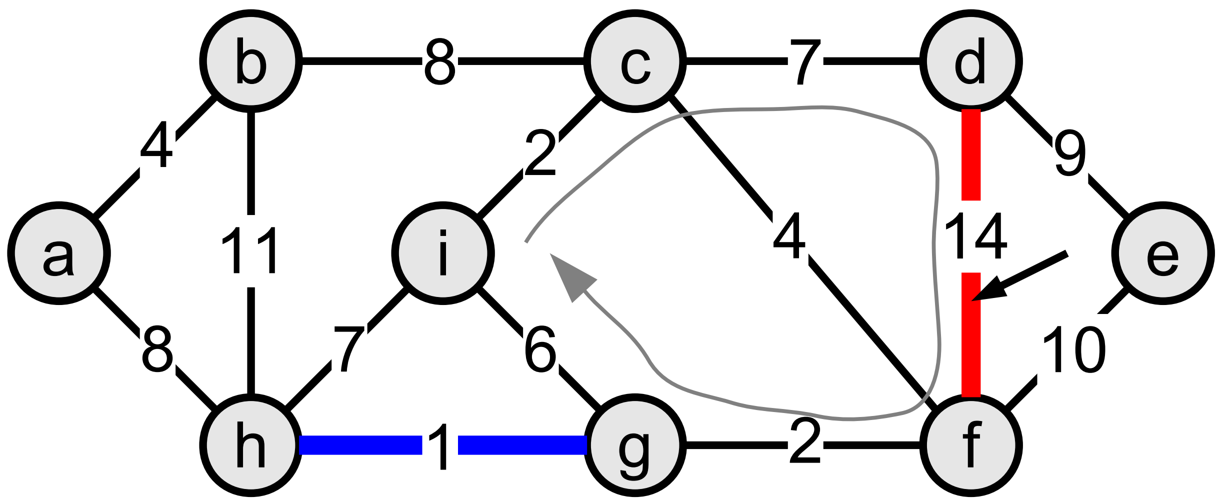
Esempio



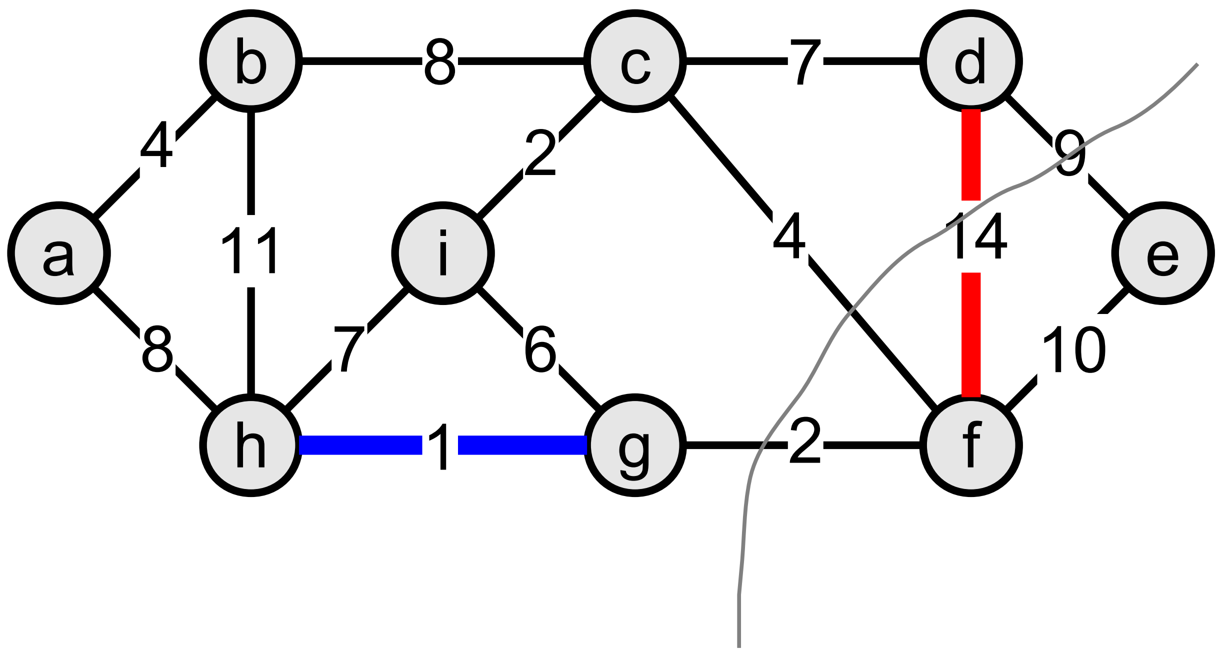
Esempio



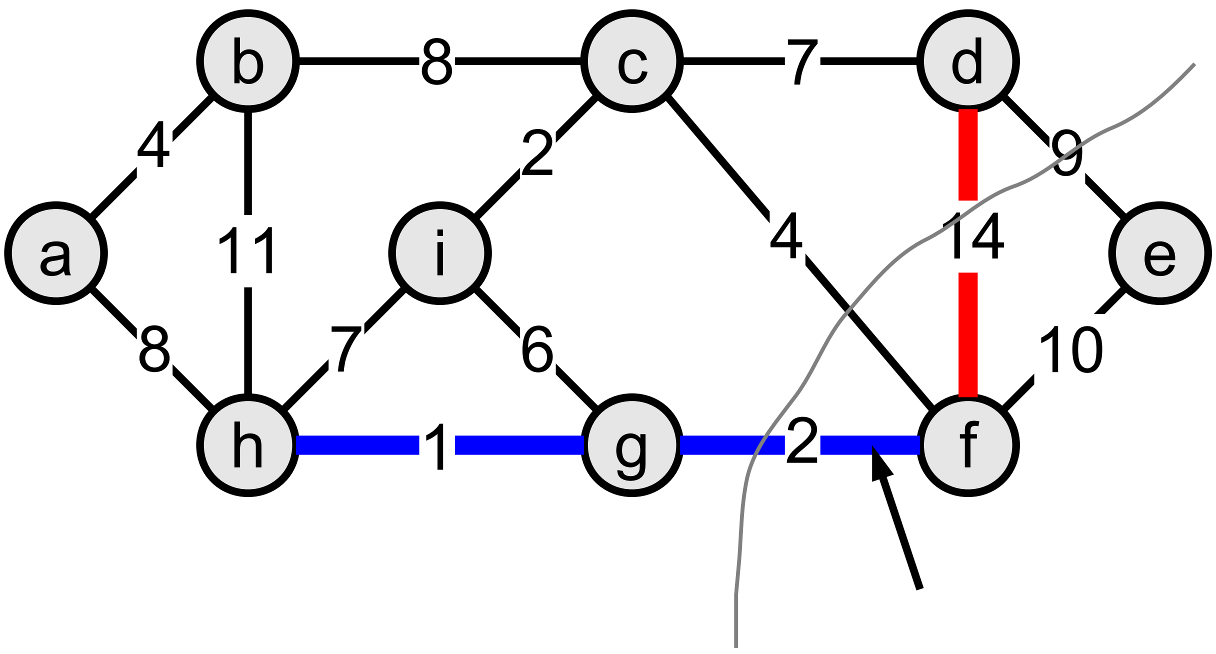
Esempio



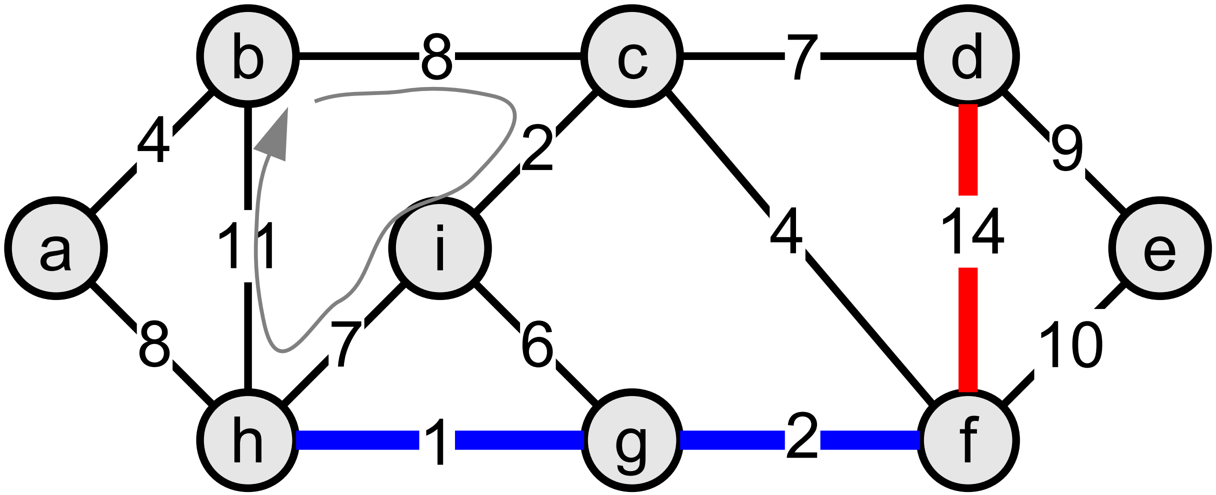
Esempio



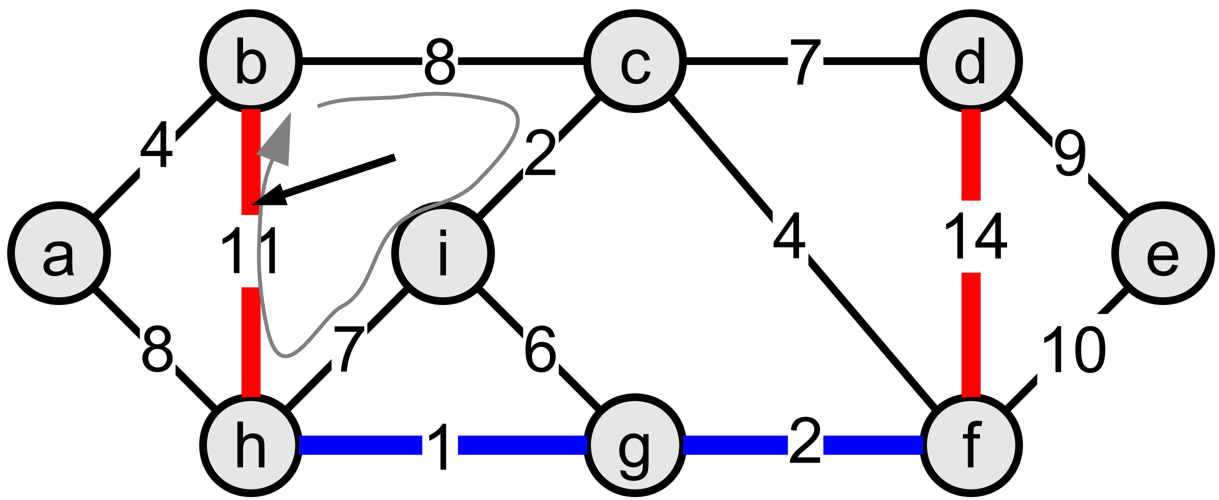
Esempio



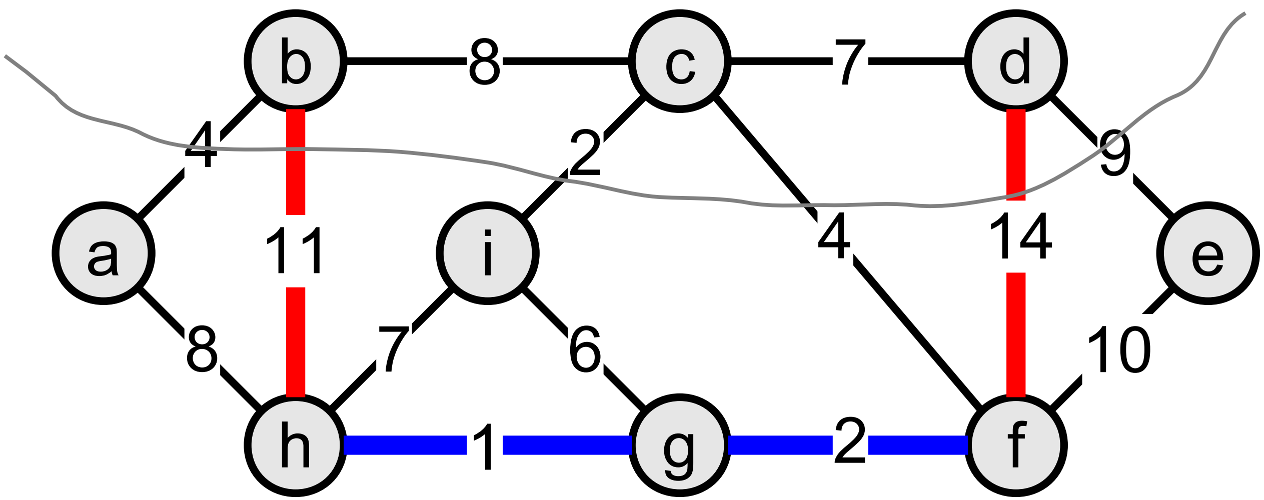
Esempio



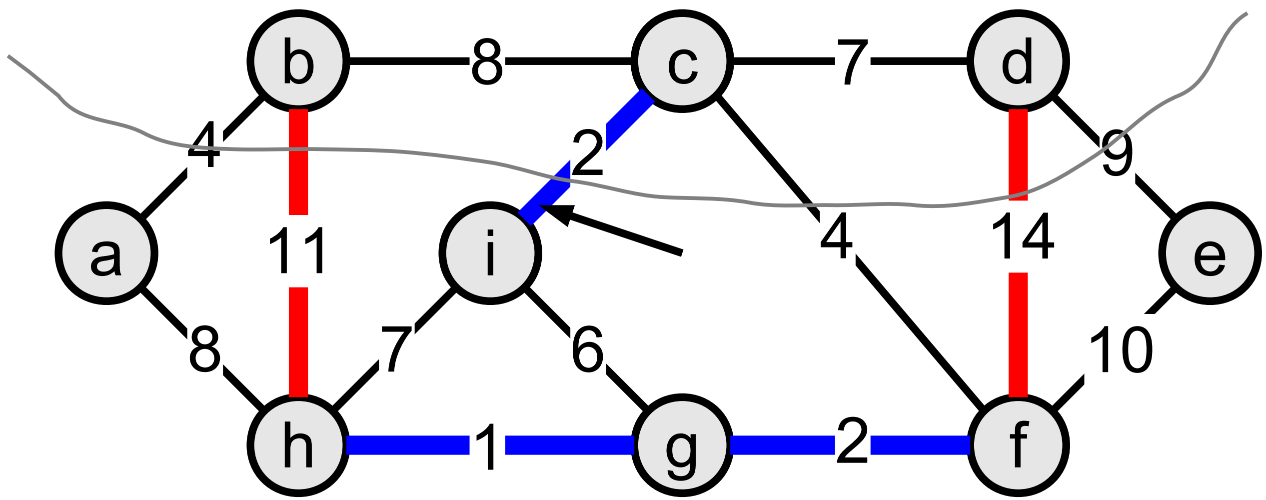
Esempio



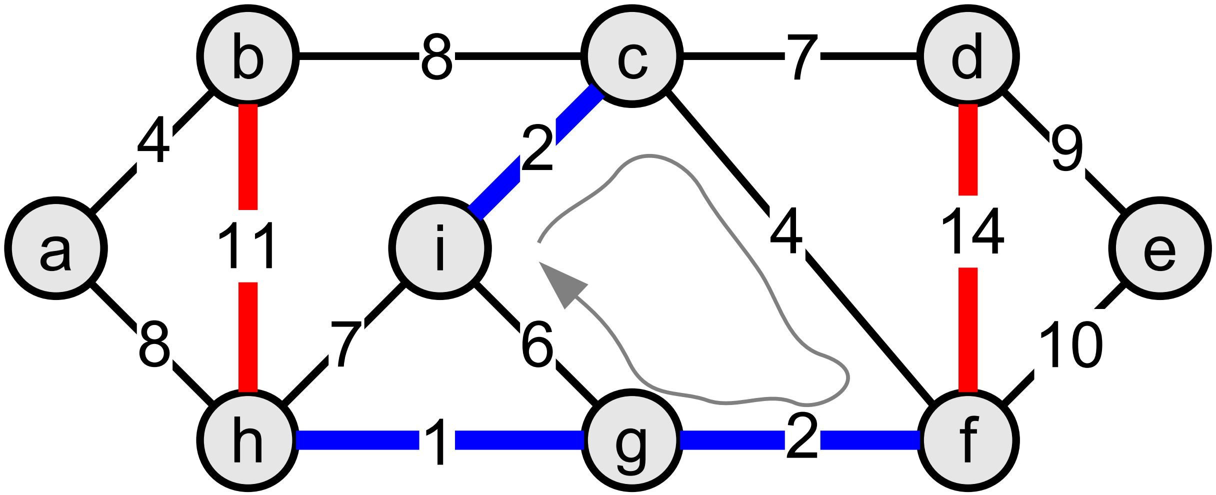
Esempio



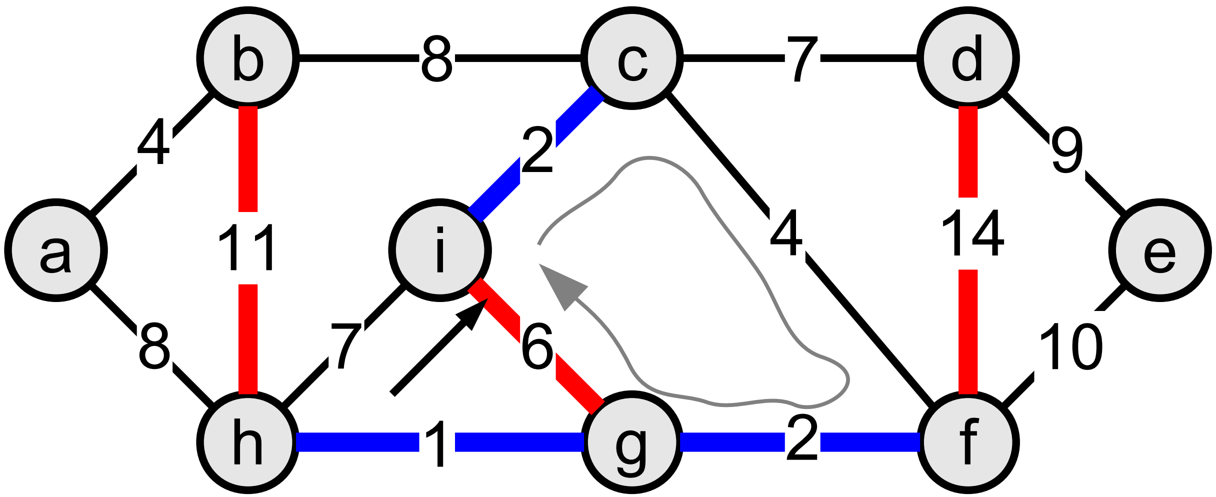
Esempio



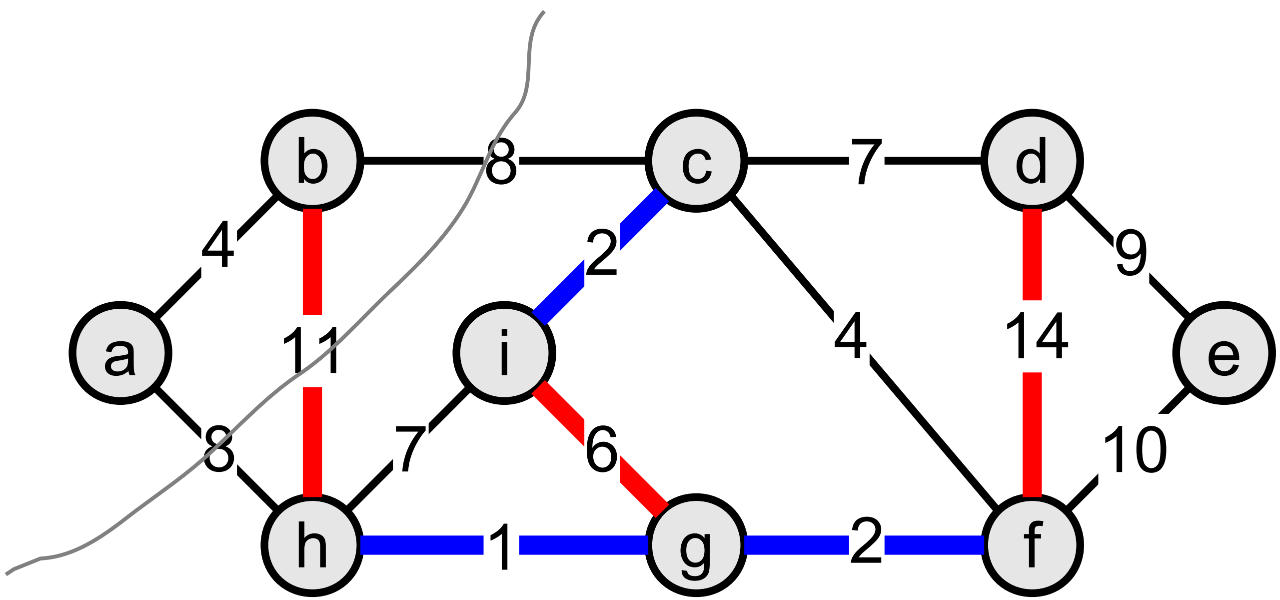
Esempio



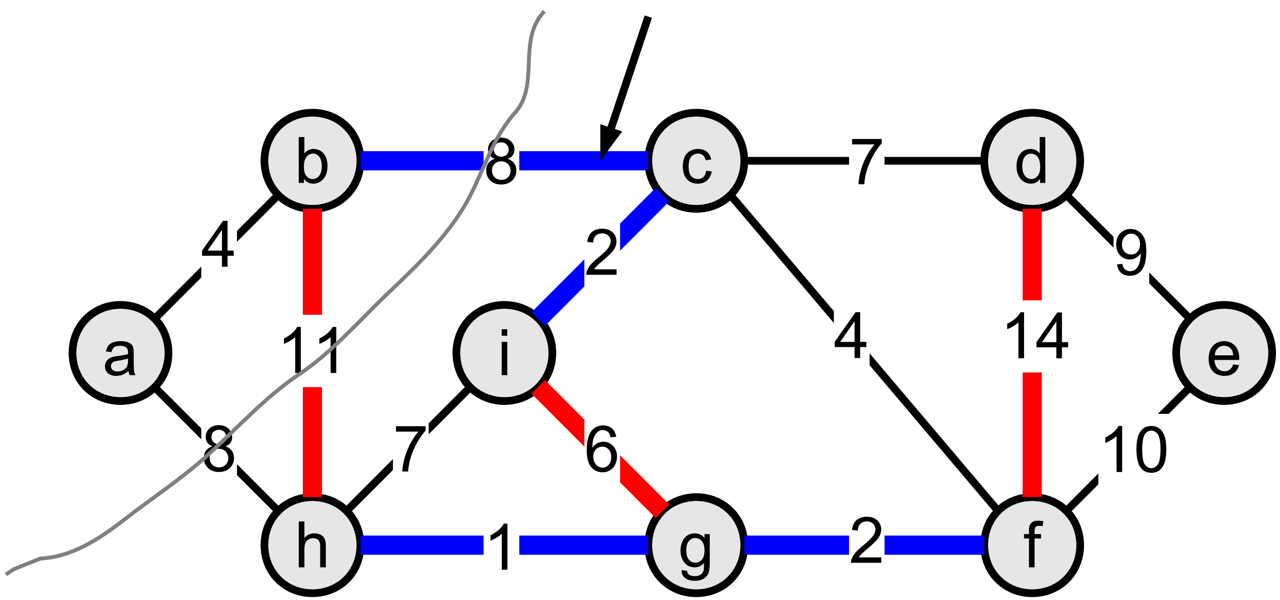
Esempio



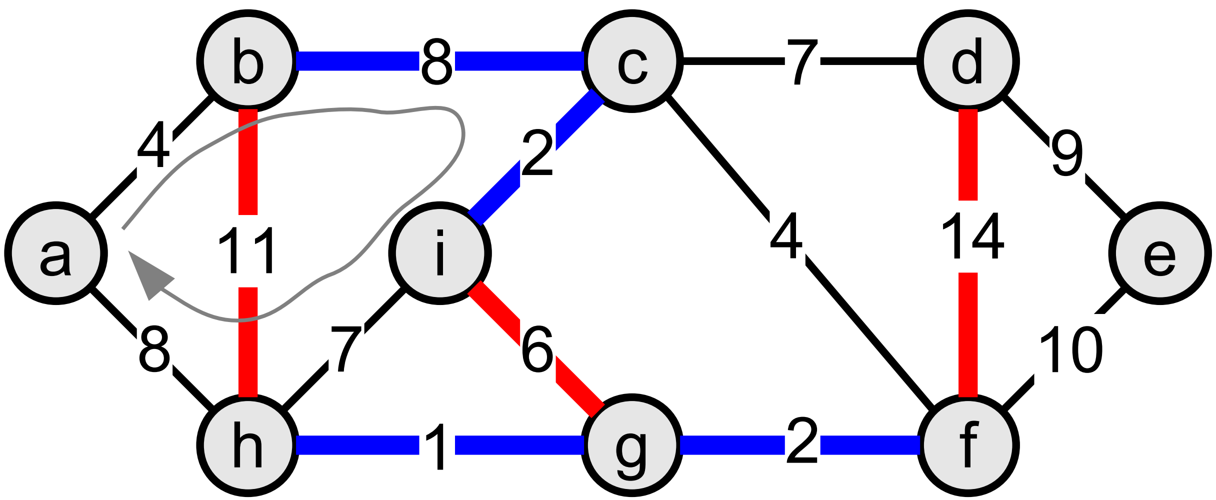
Esempio



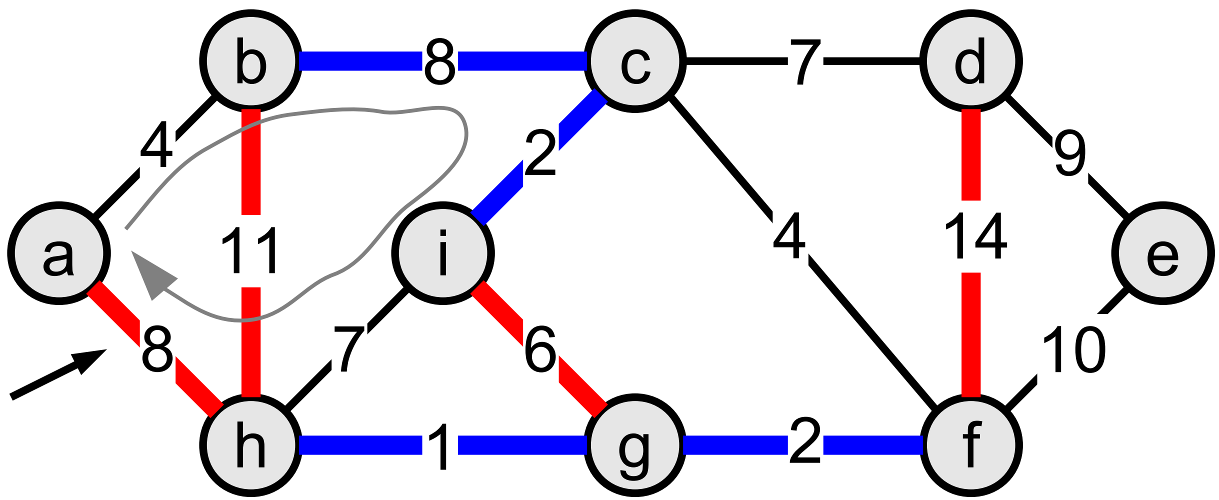
Esempio



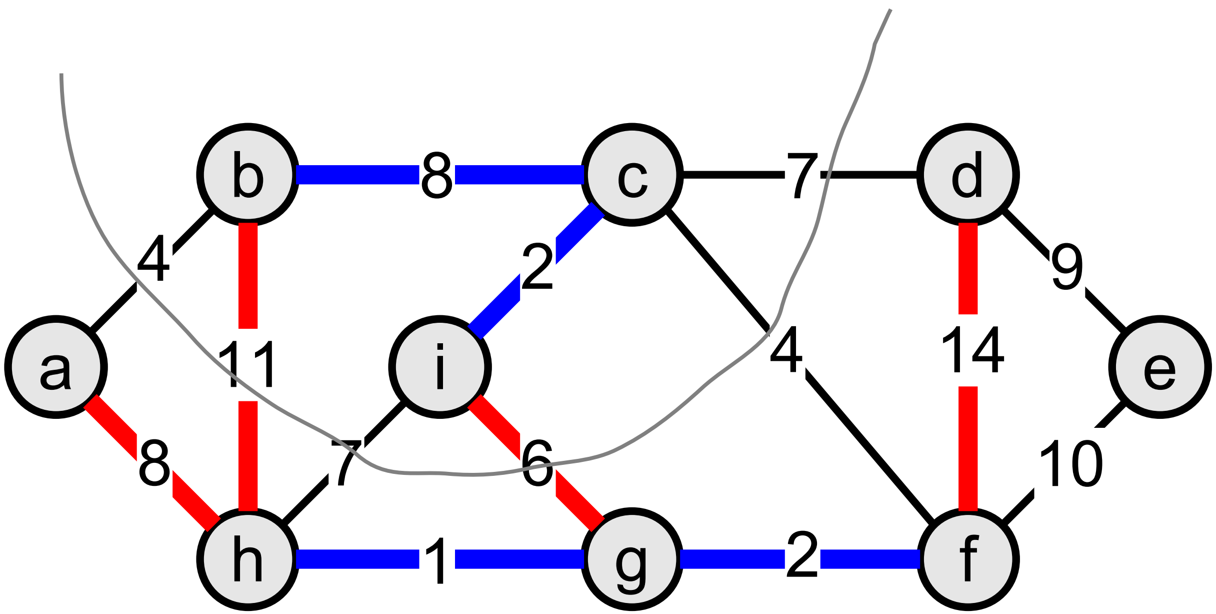
Esempio



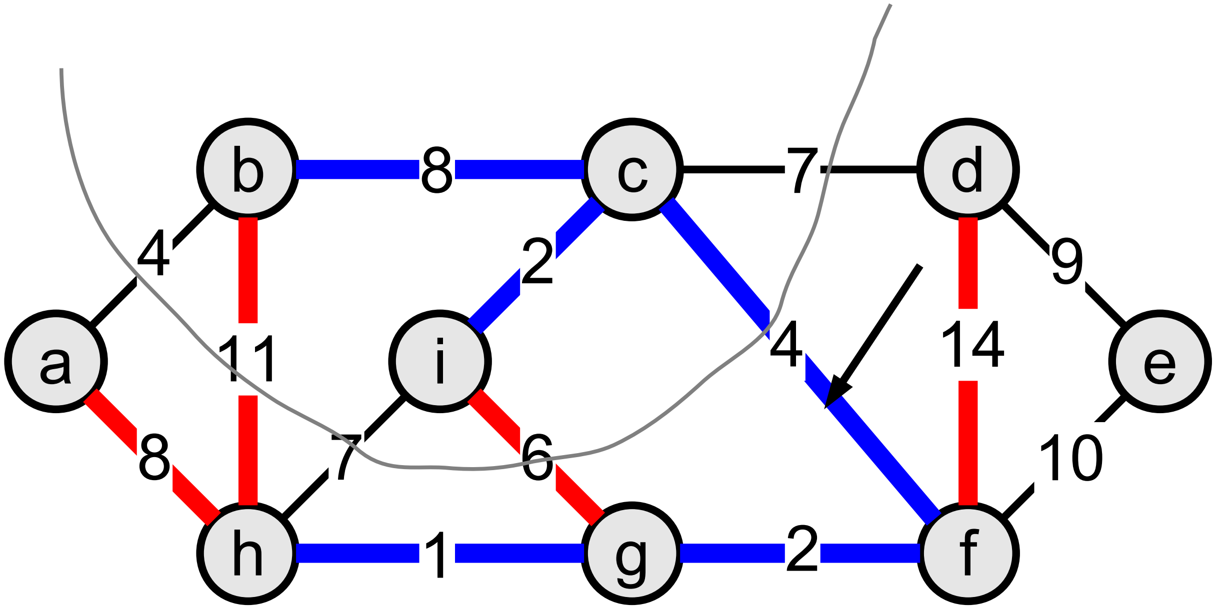
Esempio



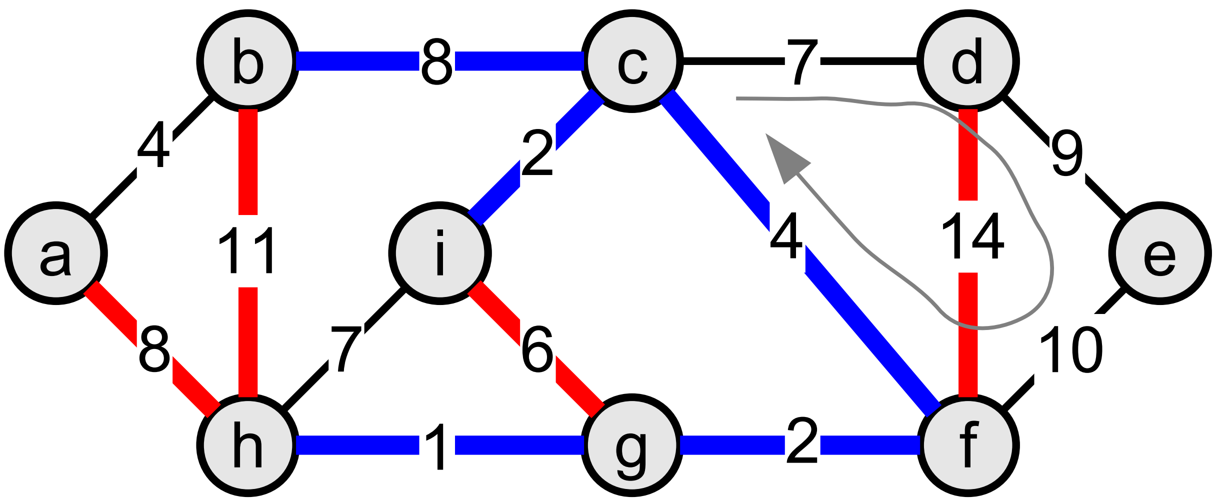
Esempio



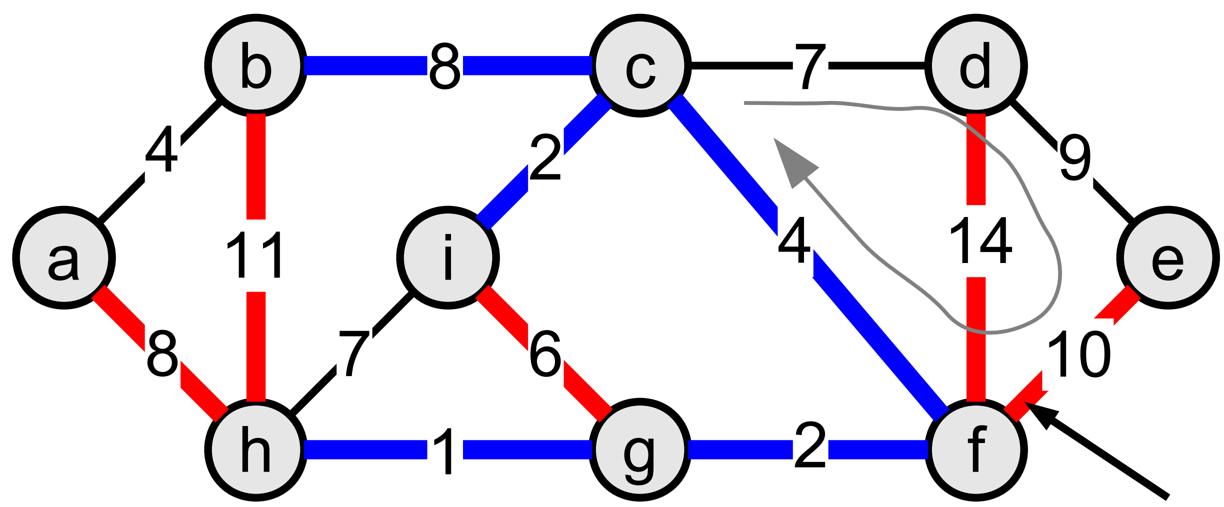
Esempio



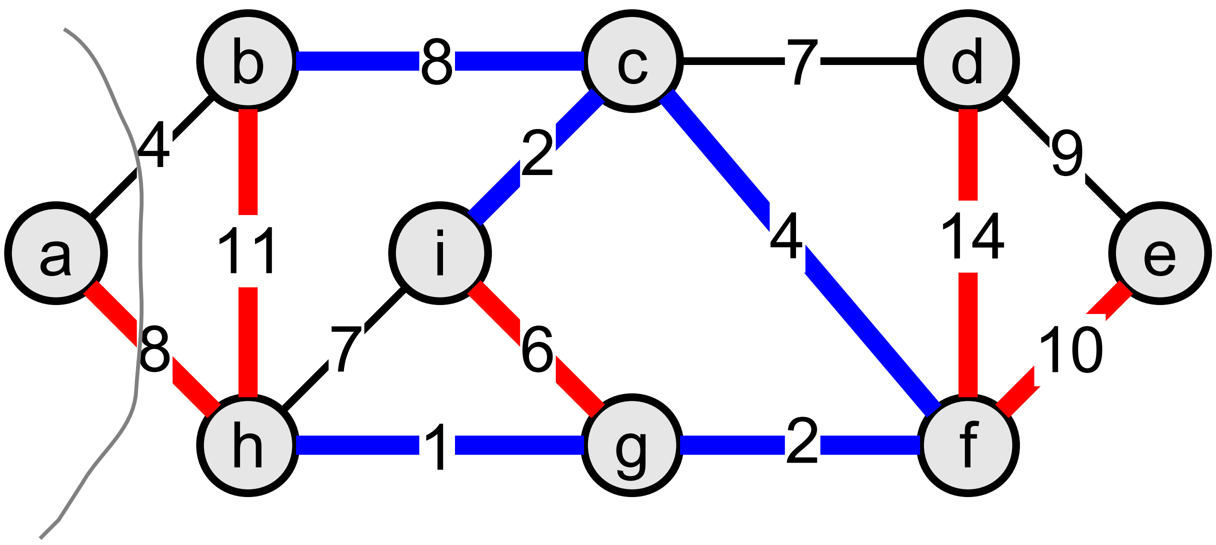
Esempio



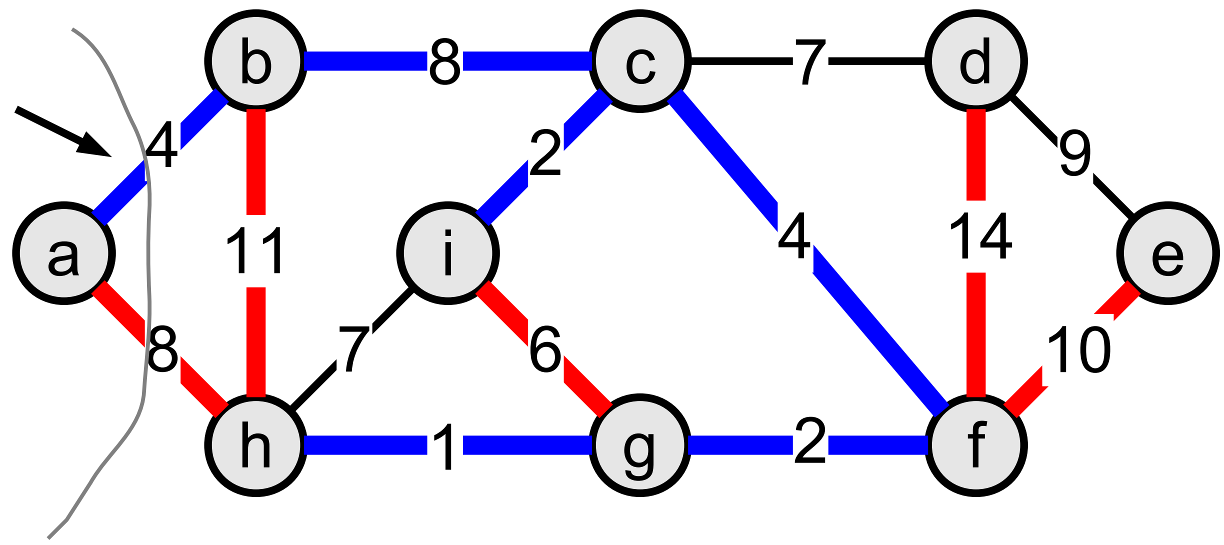
Esempio



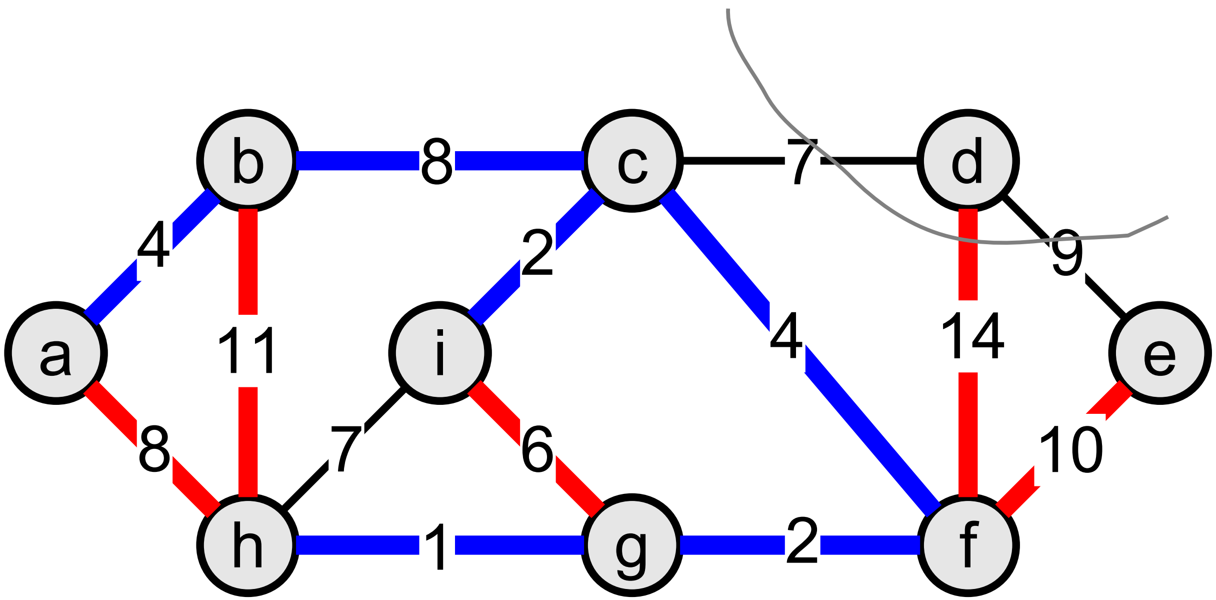
Esempio



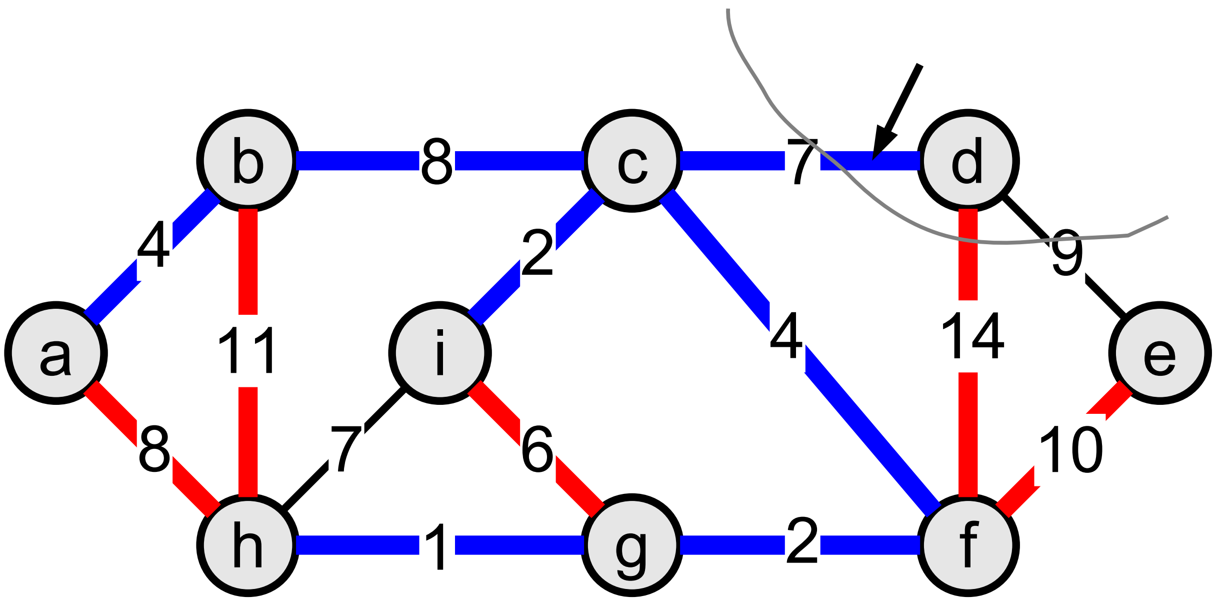
Esempio



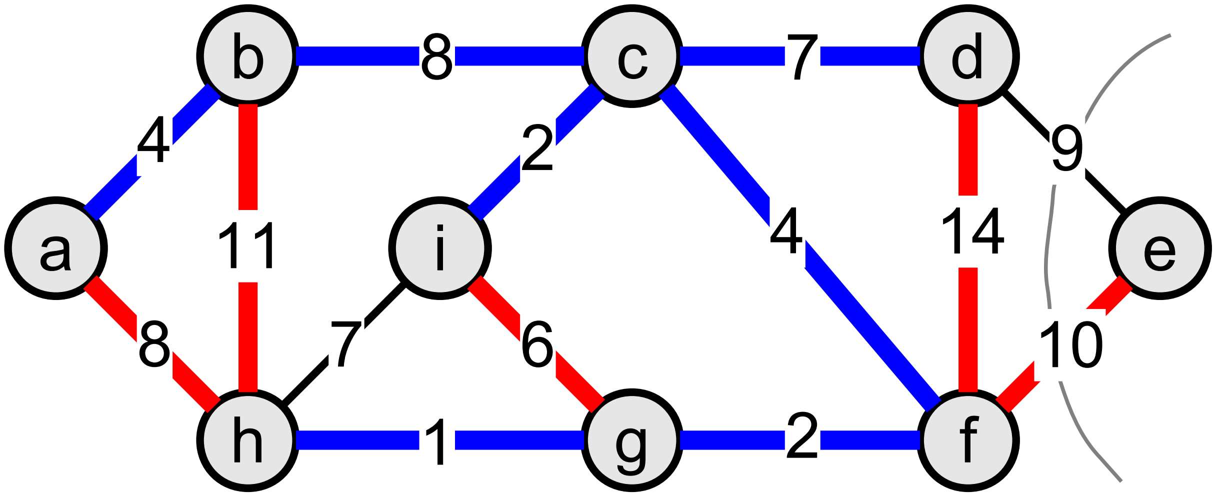
Esempio



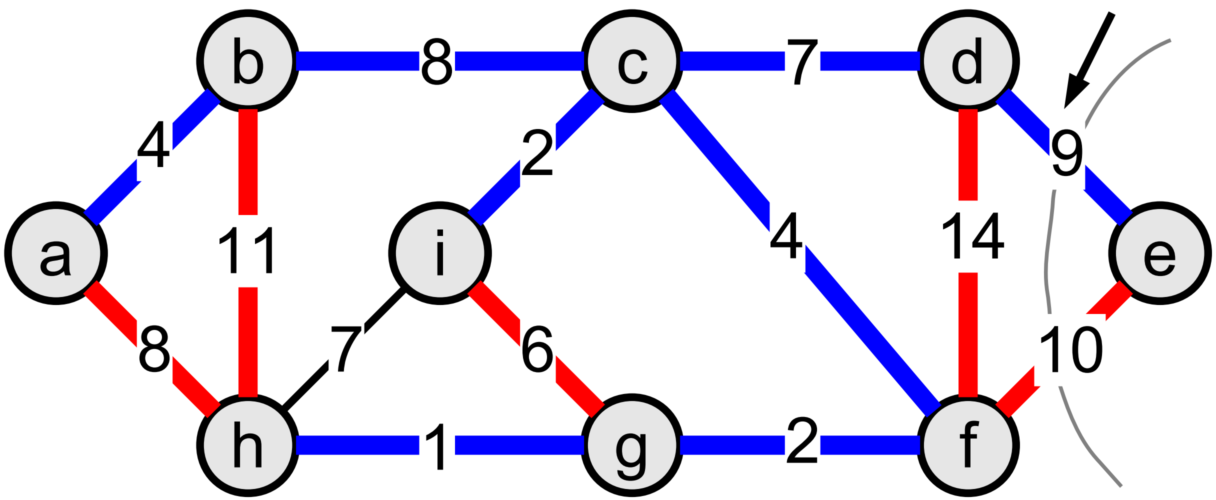
Esempio



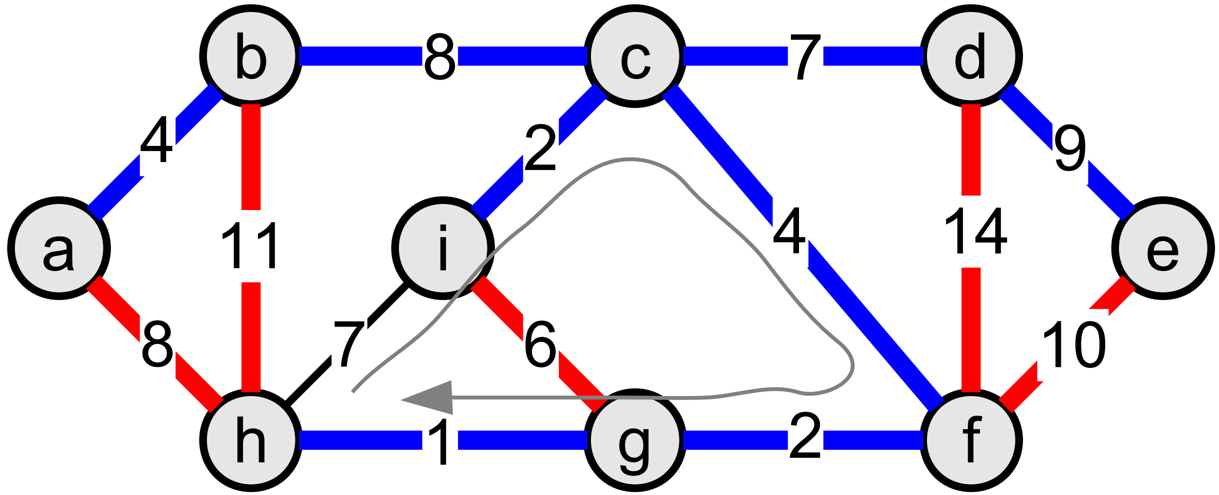
Esempio



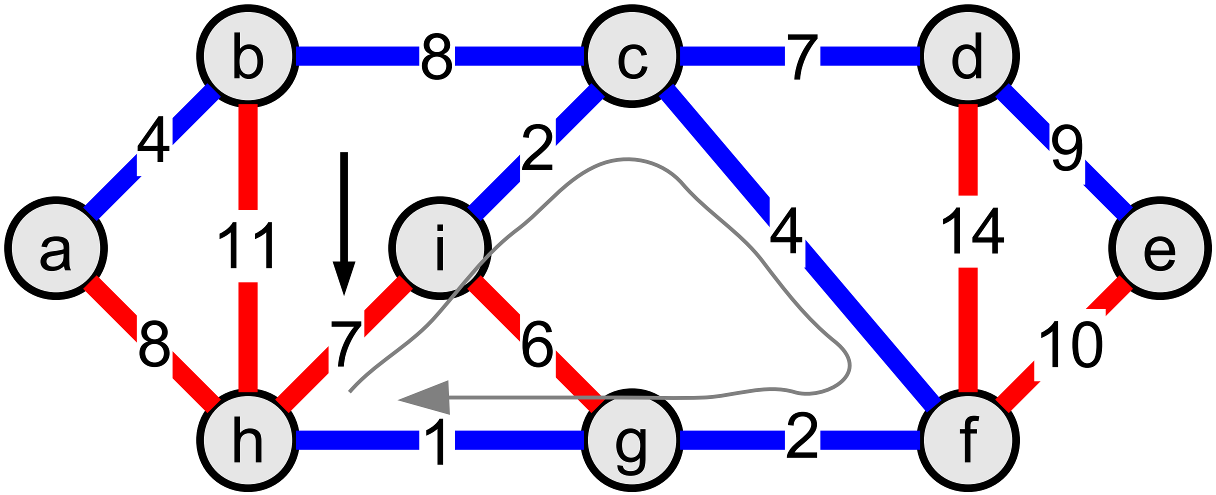
Esempio



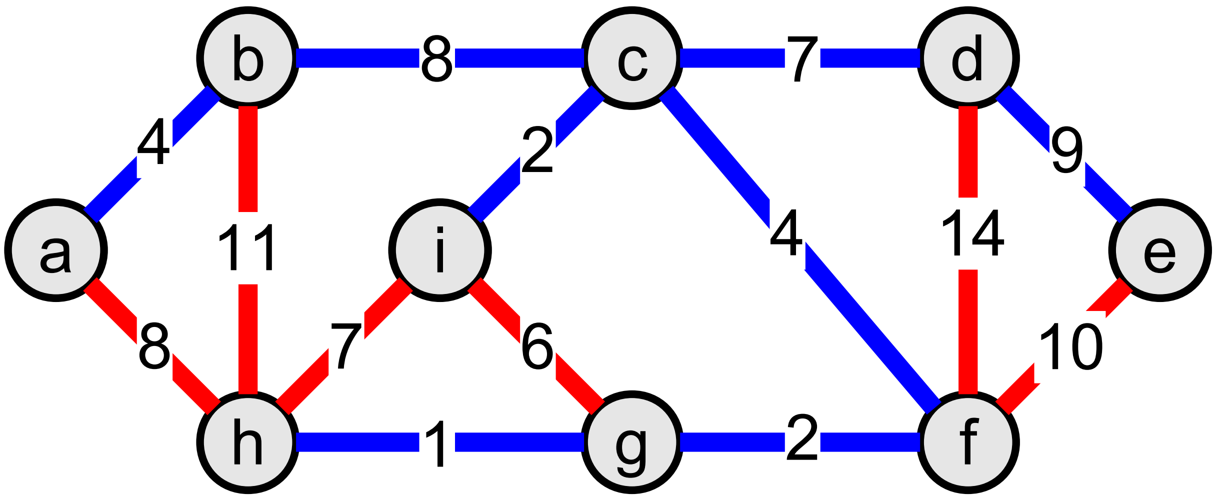
Esempio



Esempio



Finito!

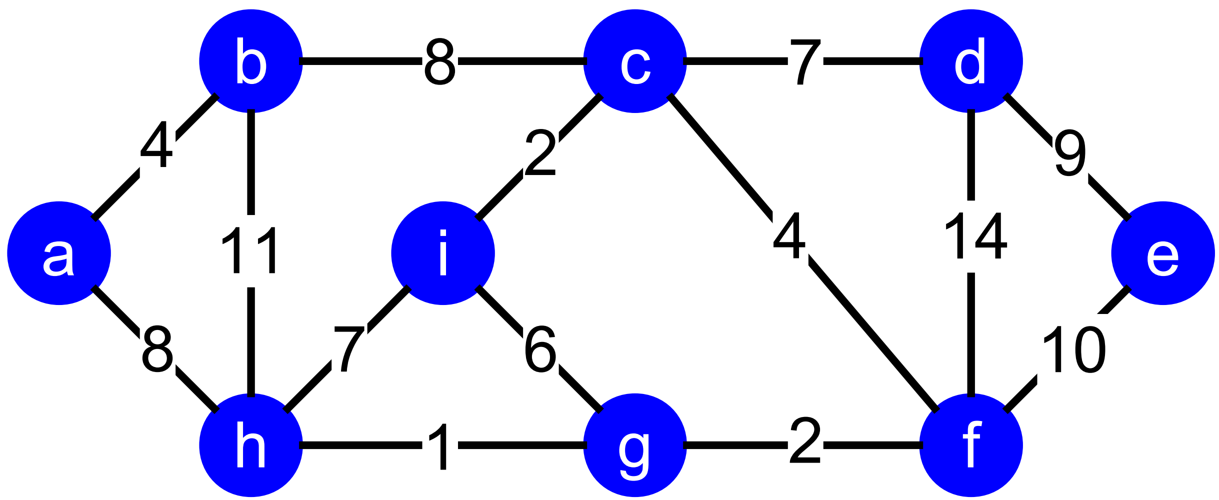


Algoritmo di Kruskal

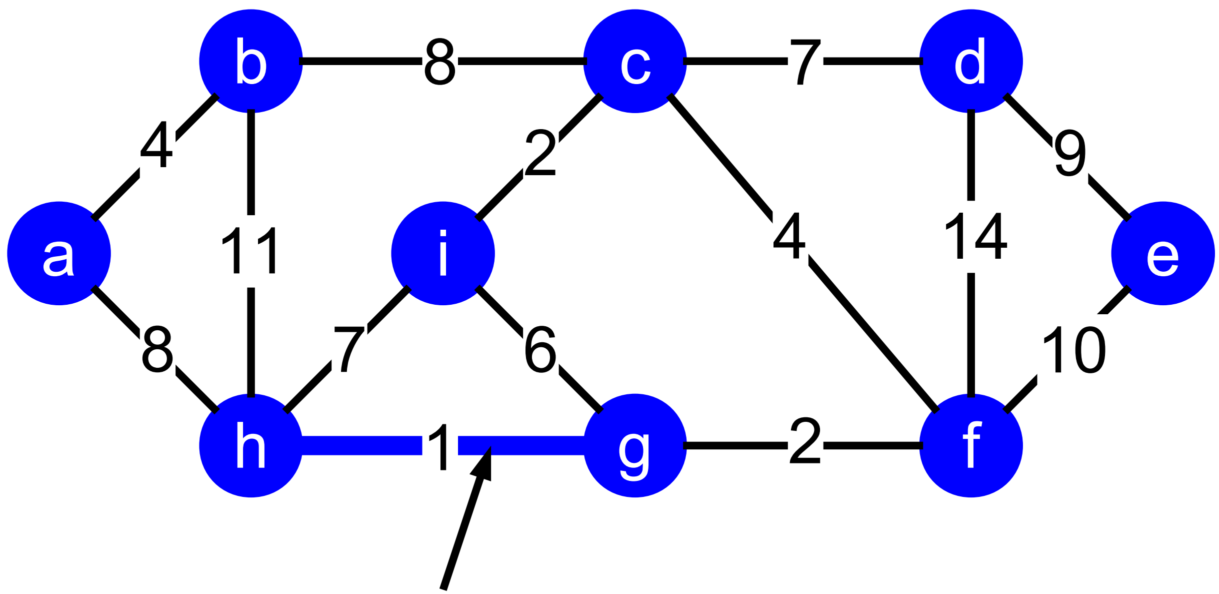
- Quanto visto in precedenza non è un algoritmo (non indica in modo deterministico come applicare le regole)
- Kruskal fissa un ordine di applicazione delle regole:
  - Idea: ingrandire sottoinsiemi disgiunti di un albero di copertura minimo connettendoli fra di loro fino ad avere l'albero finale
    - 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

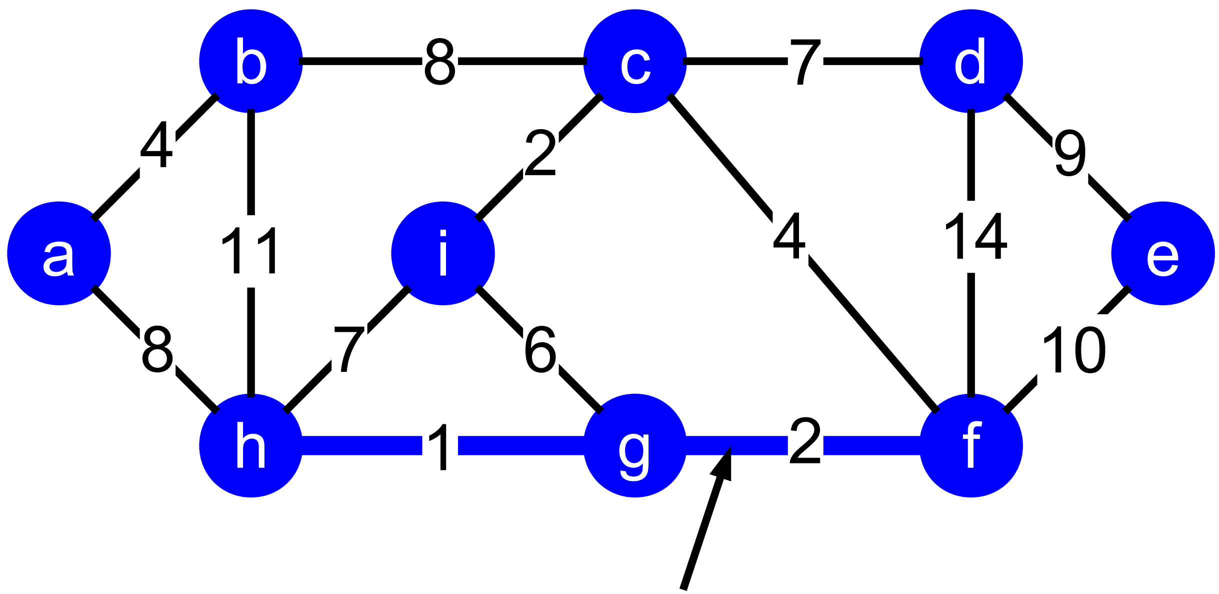
Esempio
Algoritmo di Kruskal



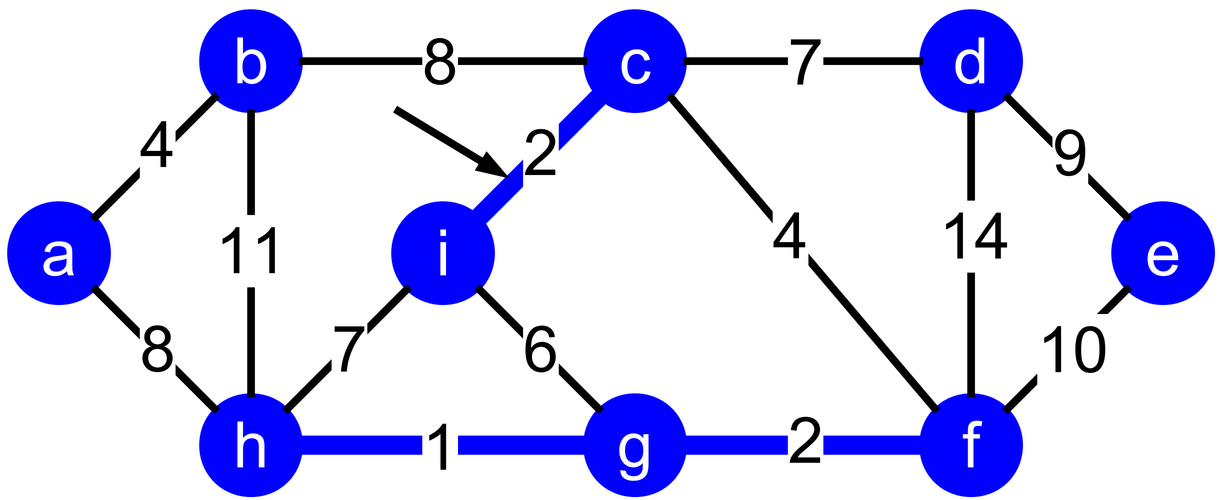
Esempio
Algoritmo di Kruskal



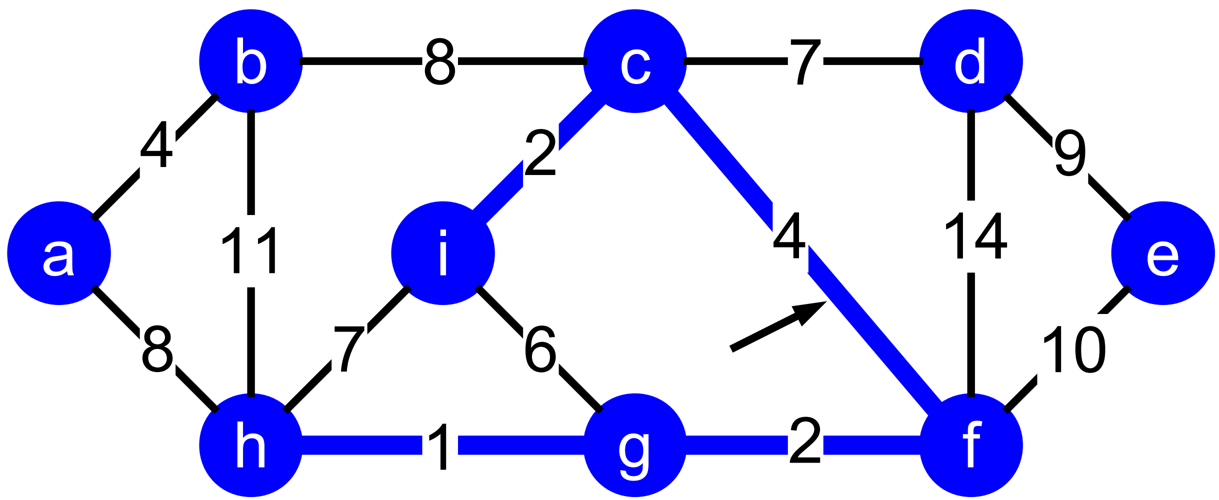
Esempio
Algoritmo di Kruskal



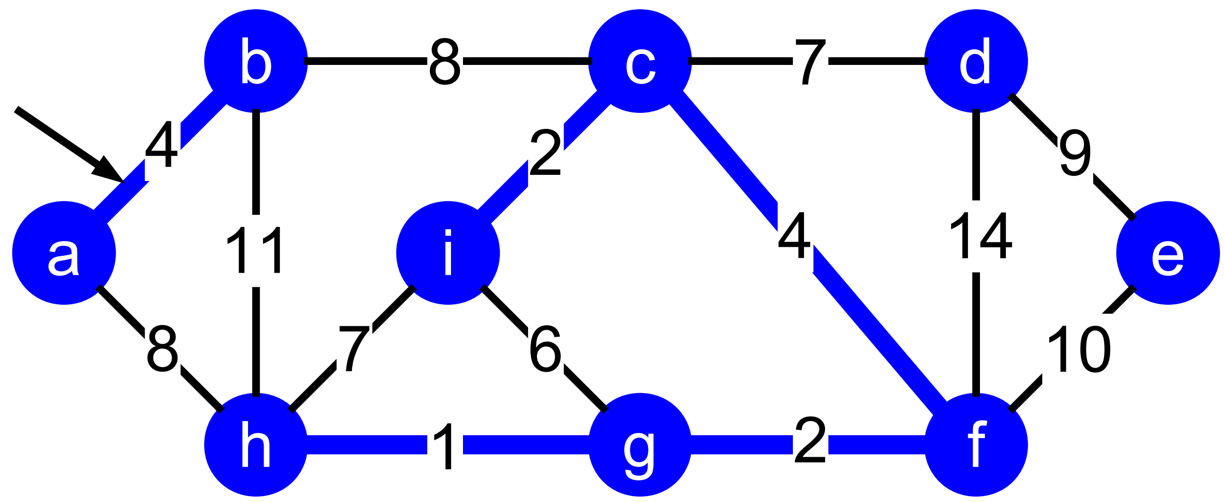
Esempio
Algoritmo di Kruskal



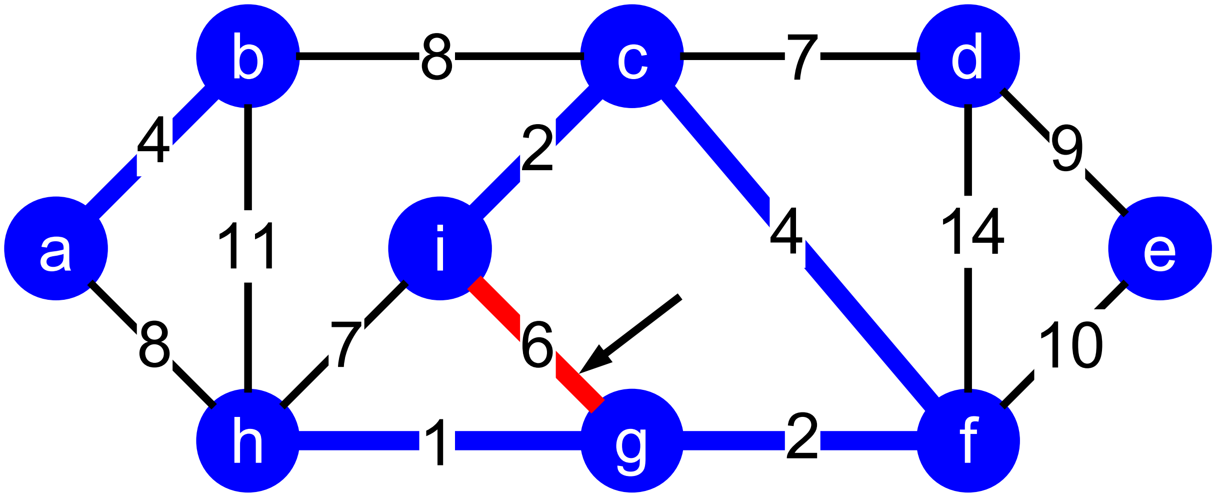
Esempio
Algoritmo di Kruskal



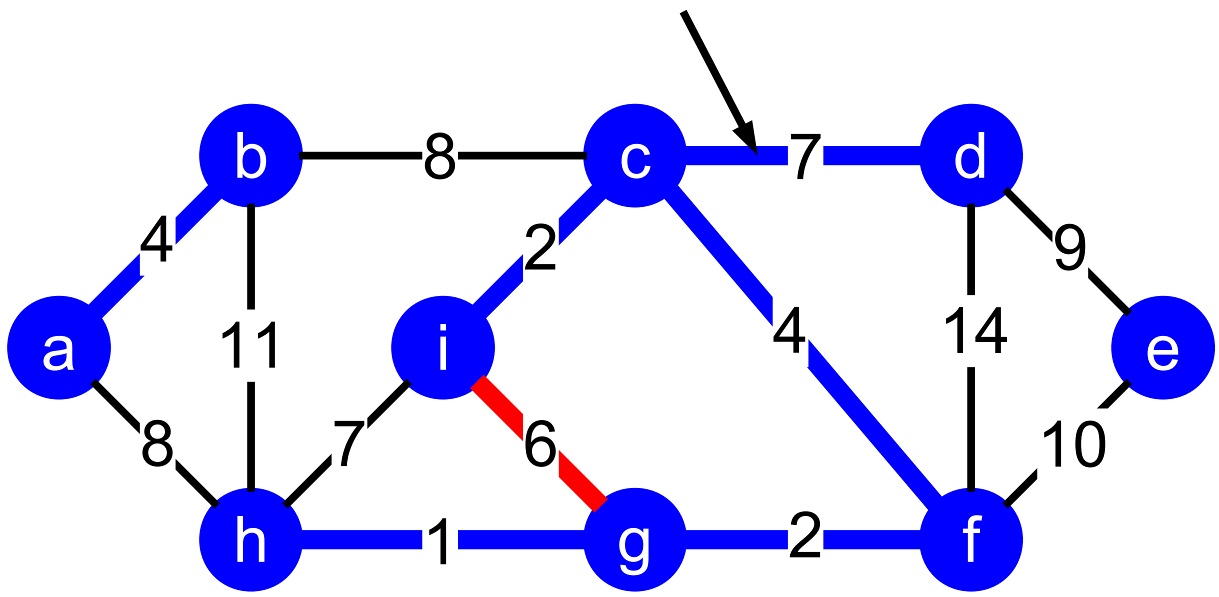
Esempio
Algoritmo di Kruskal



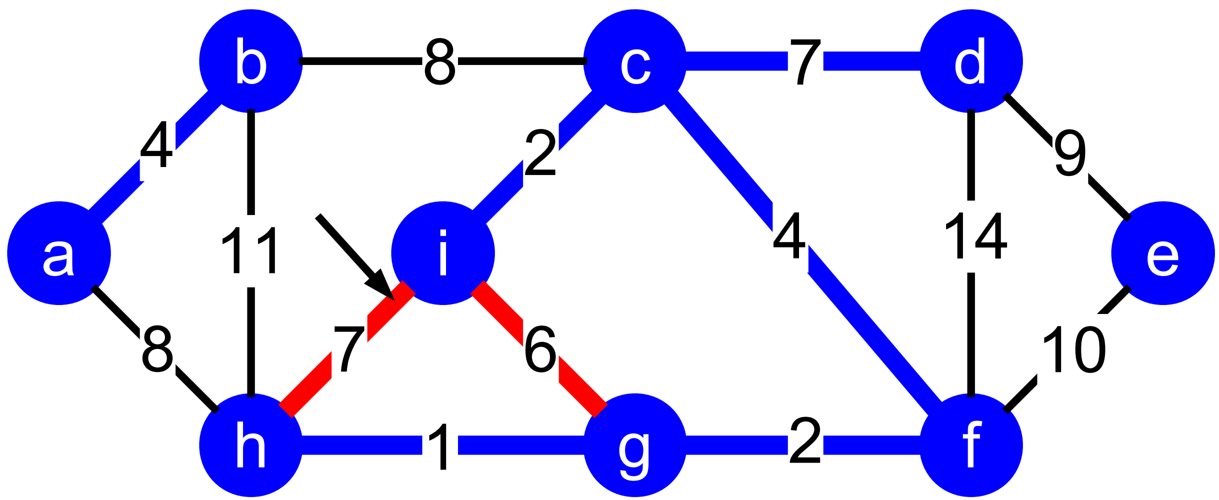
Esempio
Algoritmo di Kruskal



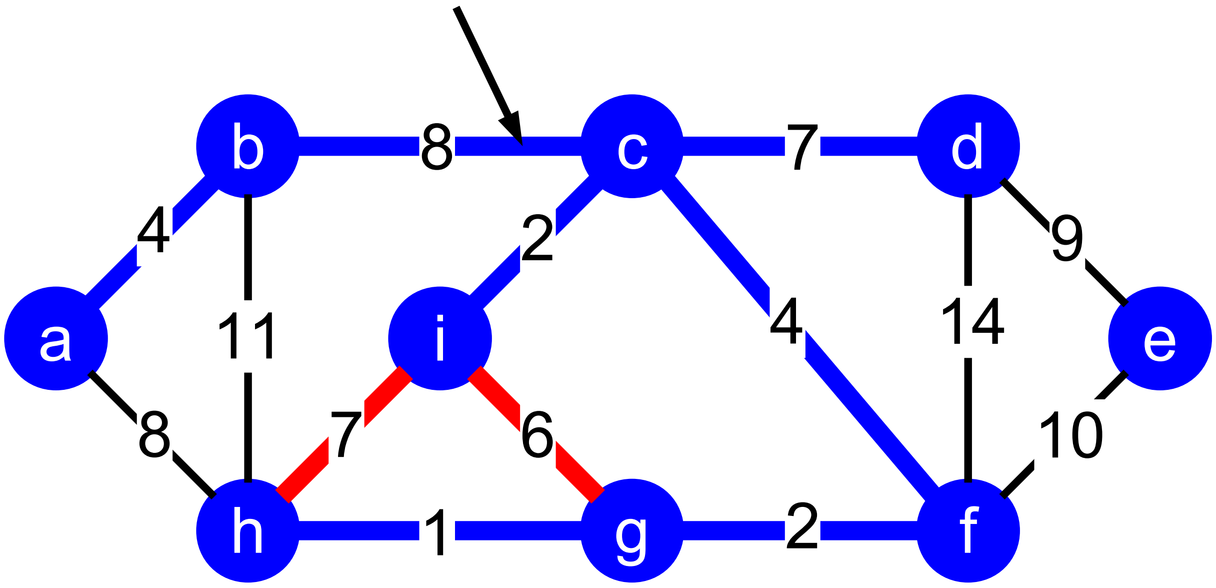
Esempio
Algoritmo di Kruskal



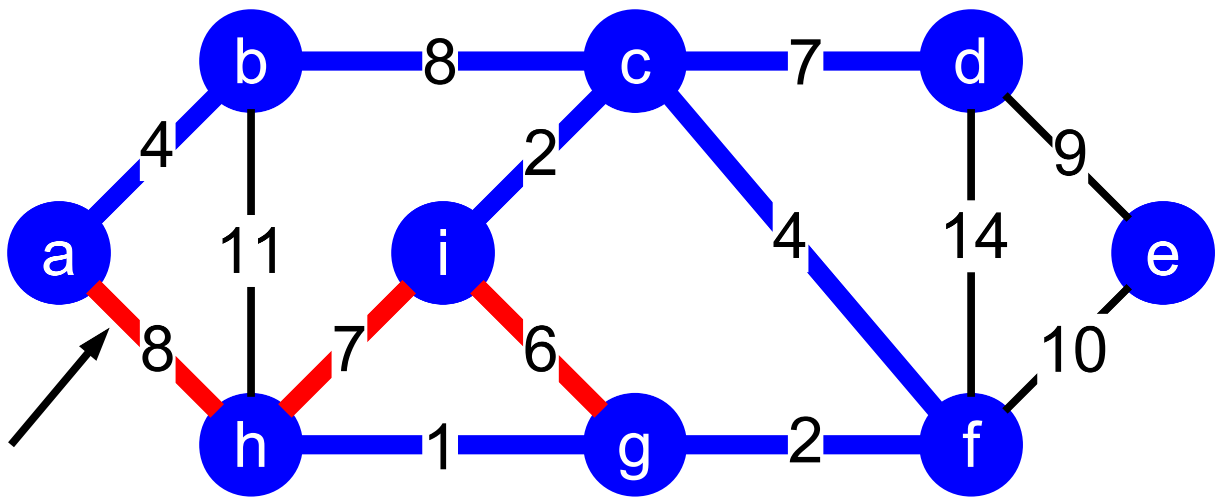
Esempio
Algoritmo di Kruskal



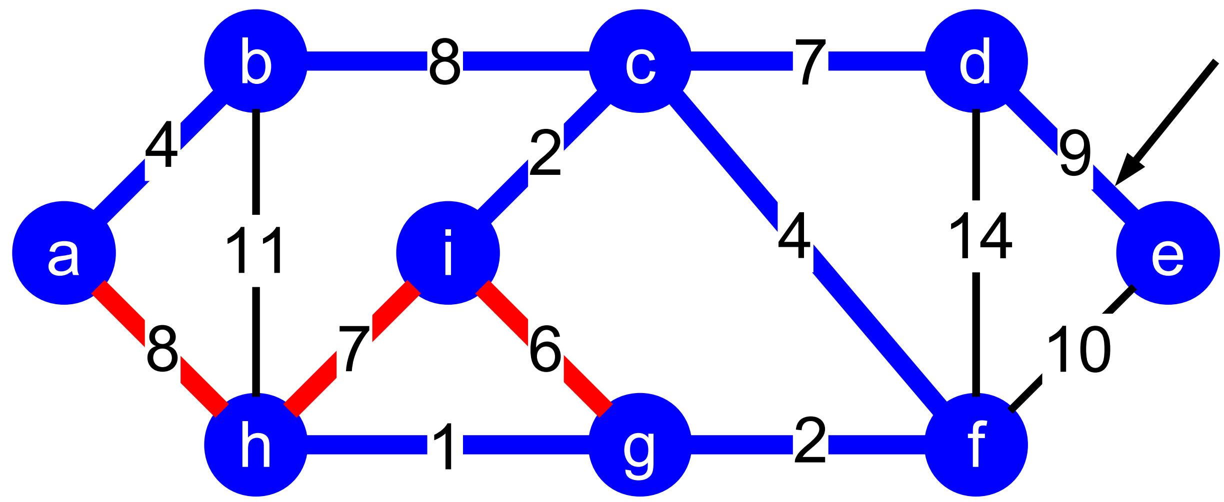
Esempio
Algoritmo di Kruskal



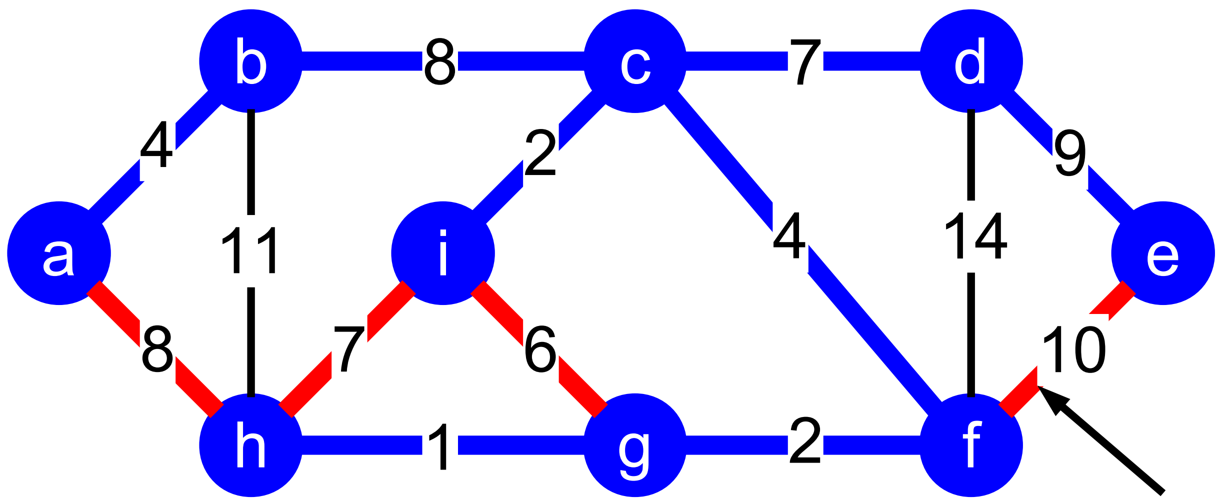
Esempio
Algoritmo di Kruskal



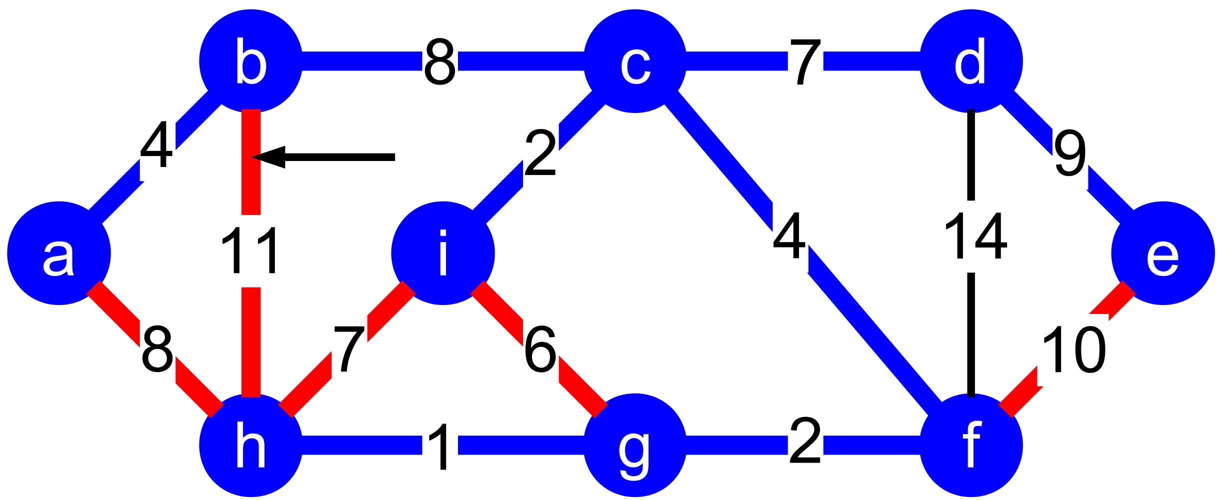
Esempio
Algoritmo di Kruskal



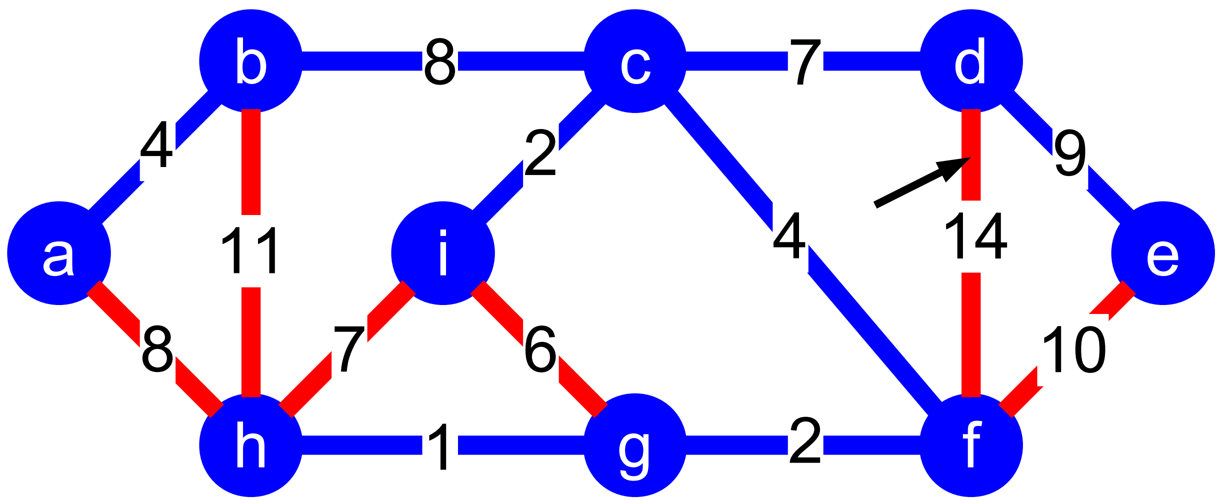
Esempio
Algoritmo di Kruskal



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 union-find!

Algoritmo di Kruskal

```
Tree Kruskal-MST(Grafo G=(V,E,w))
    UnionFind UF
    Tree T ← albero vuoto
    for i ← 1 to G.numNodi() do UF.makeSet(i)
    // ordina gli archi di E per peso w crescente
    sort(E, w)
    for each {u,v} in E do
        Tu ← UF.find(u)
        Tv ← UF.find(v)
        if (Tu ≠ Tv) then          // evita i cicli
            T ← T ∪ {u, v}         // aggiungi arco
            UF.union(Tu, Tv)       // unisci componenti
        endif
    endfor
    return T
```

Analisi

- L'ordinamento richiede

$$O(m \log m) = O(m \log n^2) = O(m \log n)$$

  dove m è il numero di archi e n il numero di nodi
- Il tempo di esecuzione dipende dalla realizzazione della struttura dati per insiemi disgiunti
  - Vengono effettuate n makeSet, $2m$ find e $(n - 1)$ union
- Se usiamo quickUnion con euristica sul rango, la sequenza di operazioni costa in tutto $O(n+m \log n+n)$
- Totale: $O(2m \log n + 2n) = O(m \log n)$

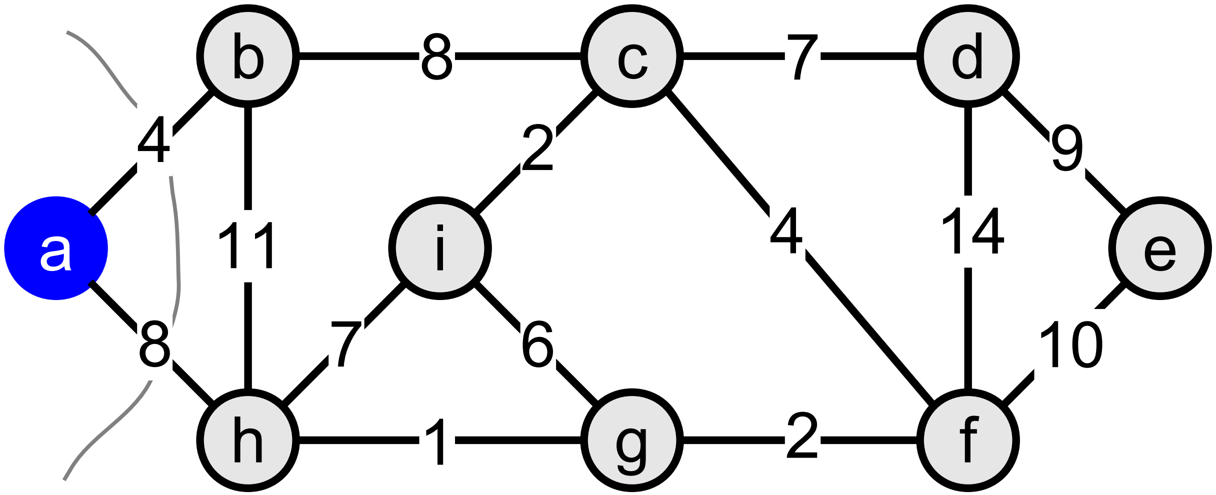
In un grafo connesso si ha sempre $m \geq n - 1$

Algoritmo di Prim

- L’algoritmo di Prim utilizza solo la regola del taglio
  - L'ordine di applicazione della regola dipende da un nodo *r*, detto *radice*, da cui si assume di far partire l'algoritmo
- Si procede mantenendo in un singolo albero *T* che viene fatto via via “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

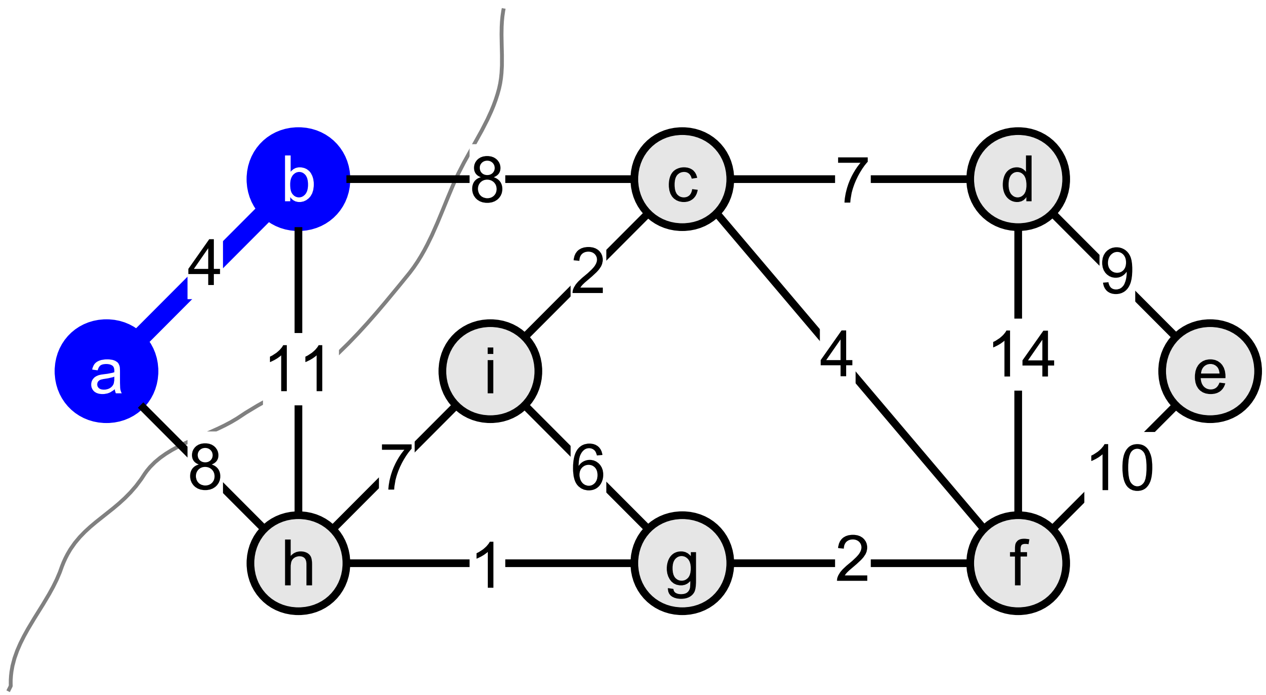
R. C. Prim: *Shortest connection networks and some generalizations*. In: Bell System Technical Journal, 36 (1957), pp. 1389–1401

Esempio
Algoritmo di Prim



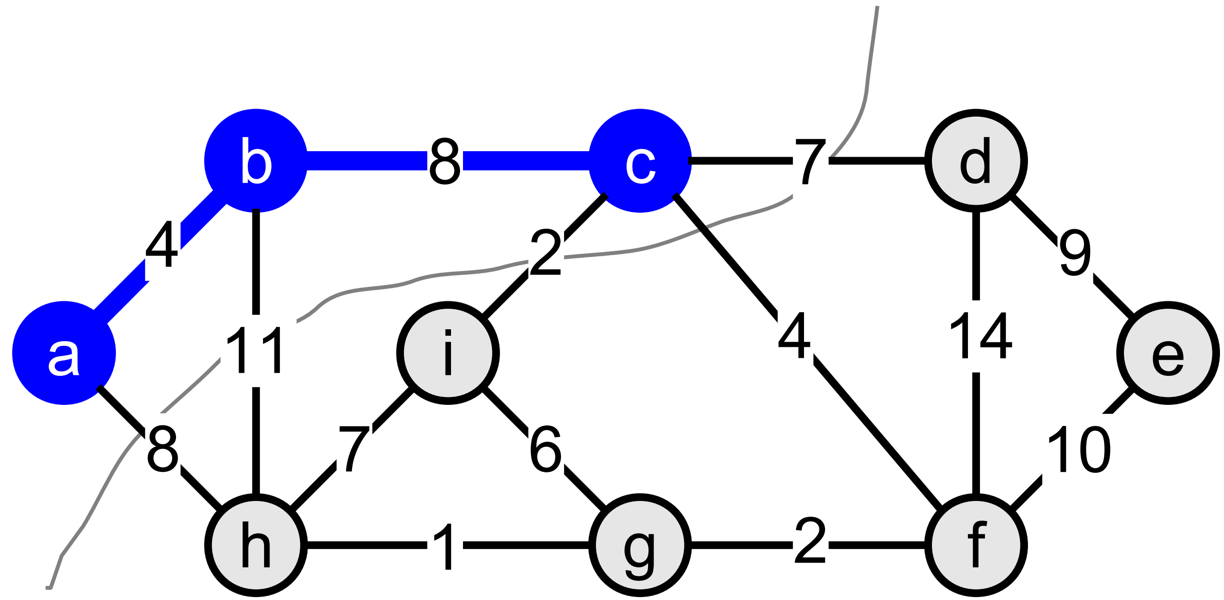
$\min(4, 8) = 4$

Esempio
Algoritmo di Prim



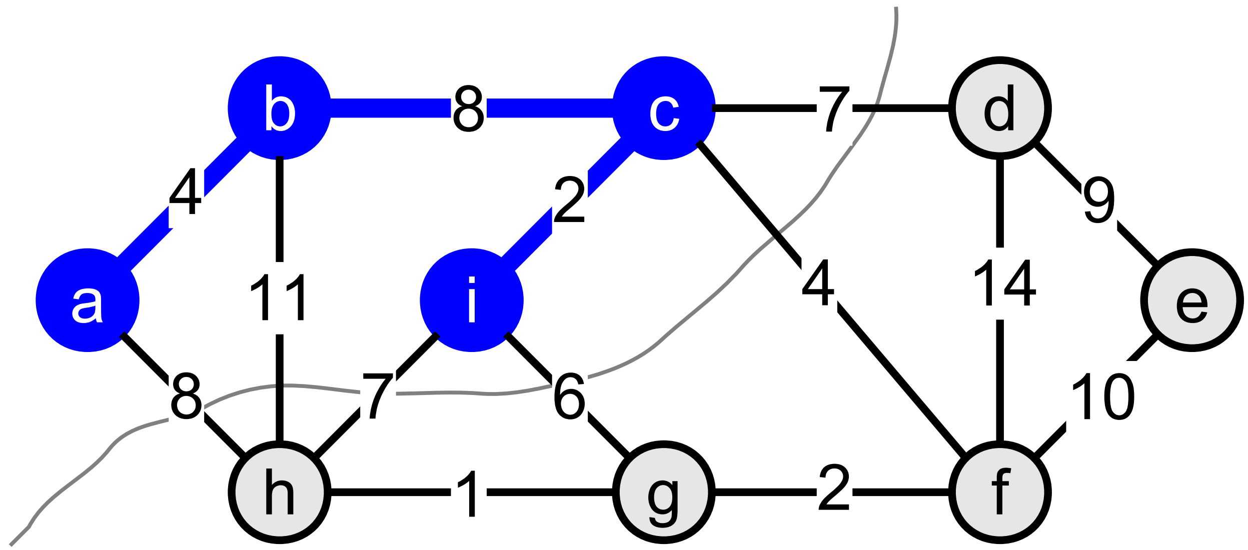
$\min(8, 11, 8) = 8$

Esempio
Algoritmo di Prim



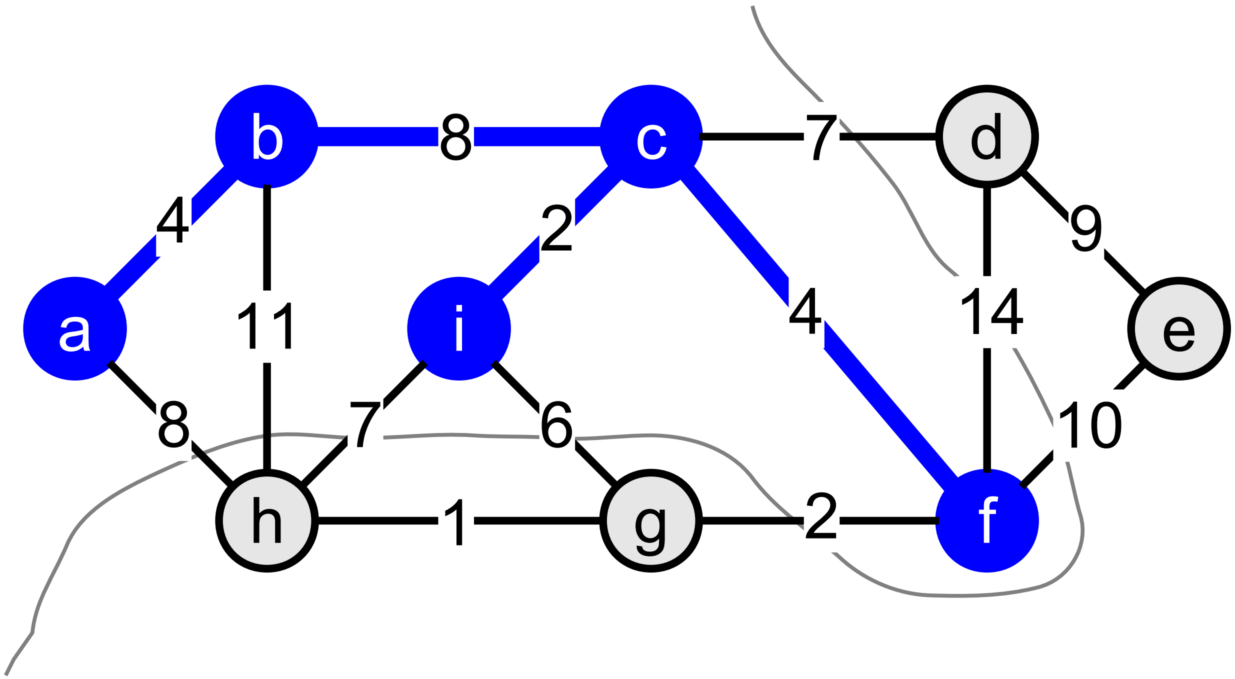
$\min(8, 11, 2, 7) = 2$

Esempio
Algoritmo di Prim



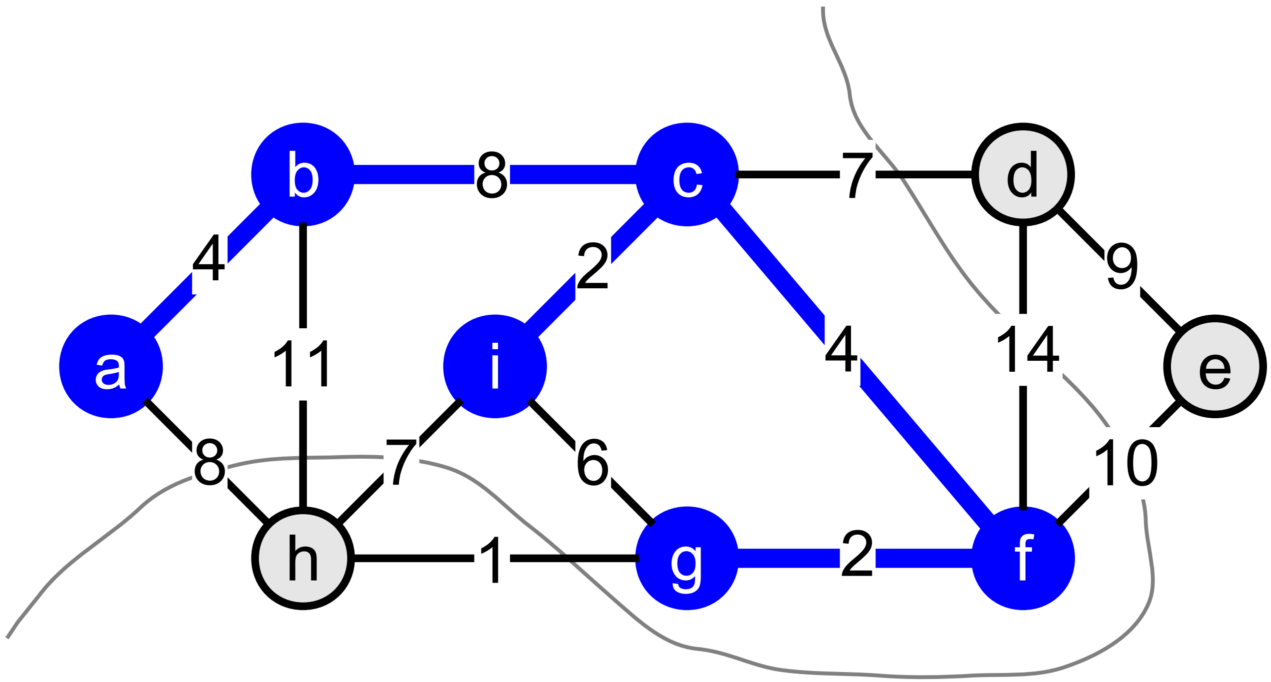
$\min(8, 7, 6, 4, 7) = 4$

Esempio
Algoritmo di Prim



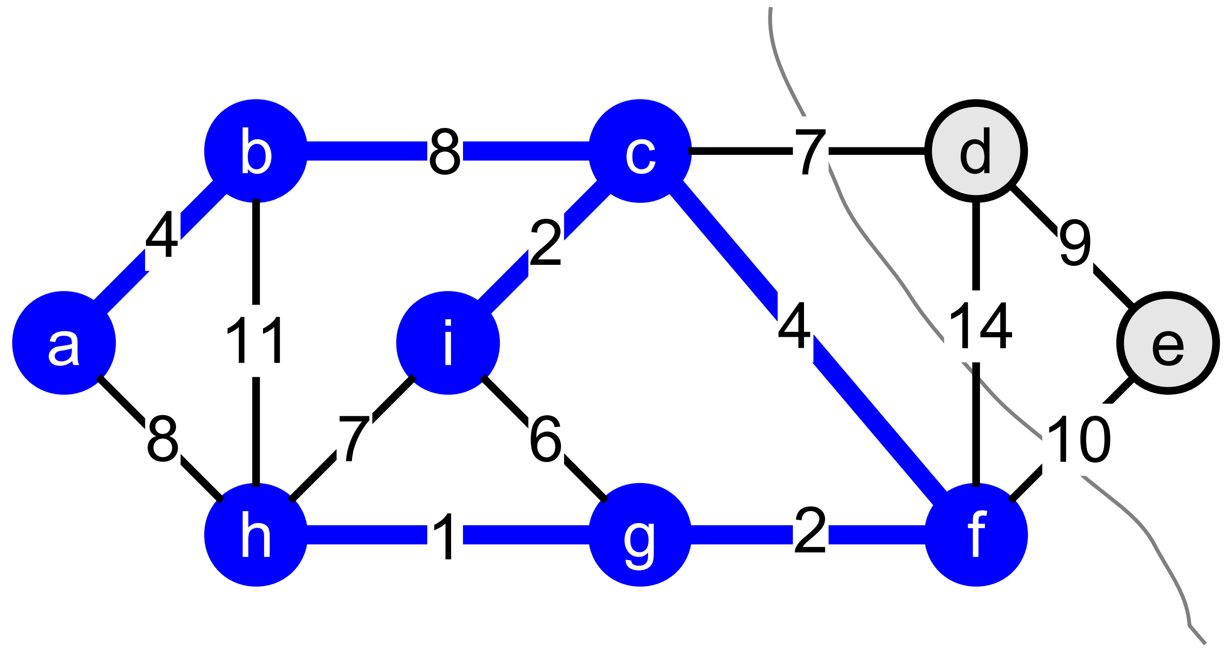
min (8, 7, 6, 2, 14, 7) = 2

Esempio
Algoritmo di Prim



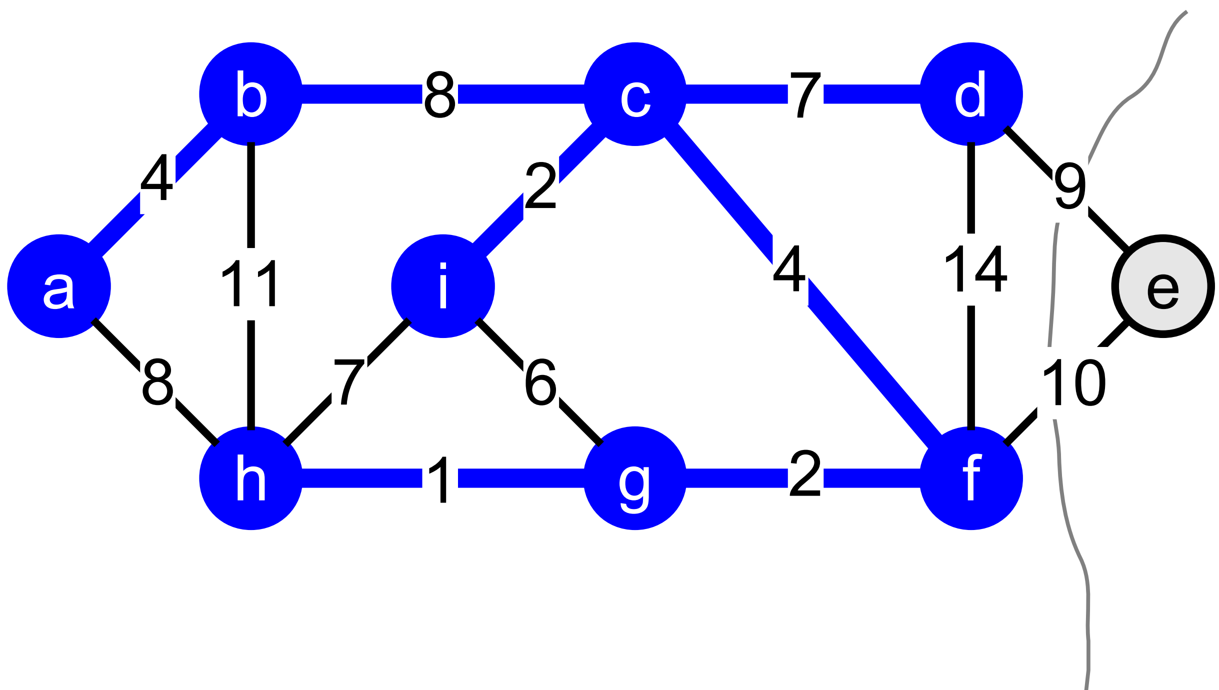
min (8, 7, 1, 10, 14, 7) = 1

Esempio
Algoritmo di Prim



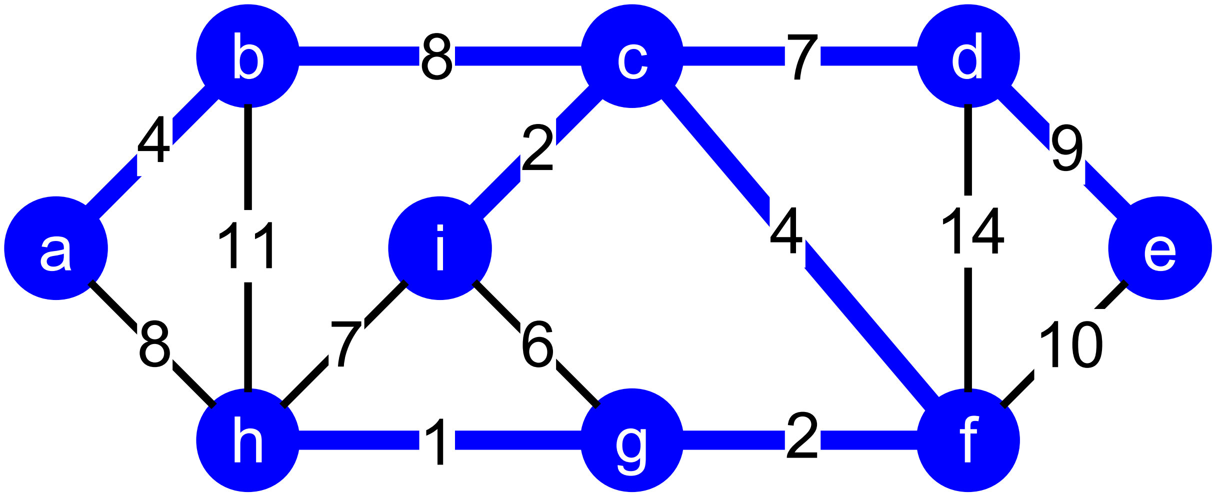
$\min(7, 14, 10) = 7$

Esempio
Algoritmo di Prim



min (10, 9) = 9

Esempio
Algoritmo di Prim



stop

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 $d[v]$
    - Più precisamente, la coda viene pian piano popolata quando un nodo risulta essere collegato ad un nodo già nel MST
  - $d[v]$ 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 (in questo caso il nodo non è ancora entrato nella coda con priorità)
- Albero rappresentato mediante il vettore padri $p[v]$
- Array di booleani per ricordare i nodi già nel MST
- Terminazione: quando la coda Q è vuota
  - Tutti i nodi tranne la radice conoscono il proprio padre

Algoritmo di Prim

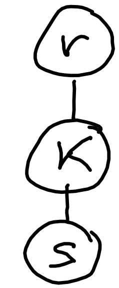
```
integer[] Prim-MST(Grafo G=(V,E,w), nodo s)
    double d[1..n]; integer p[1..n]; boolean b[1..n];
    for v ← 1 to n do
        d[v] ← ∞;
        p[v] ← -1;
        b[v] ← false;
    endfor
    d[s] ← 0;
    CodaPriorita<integer, double> Q; Q.insert(s, d[s]);
    while (not Q.isEmpty()) do
     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

parent vector

equivalente di null

ovvero la radice non ha padre.

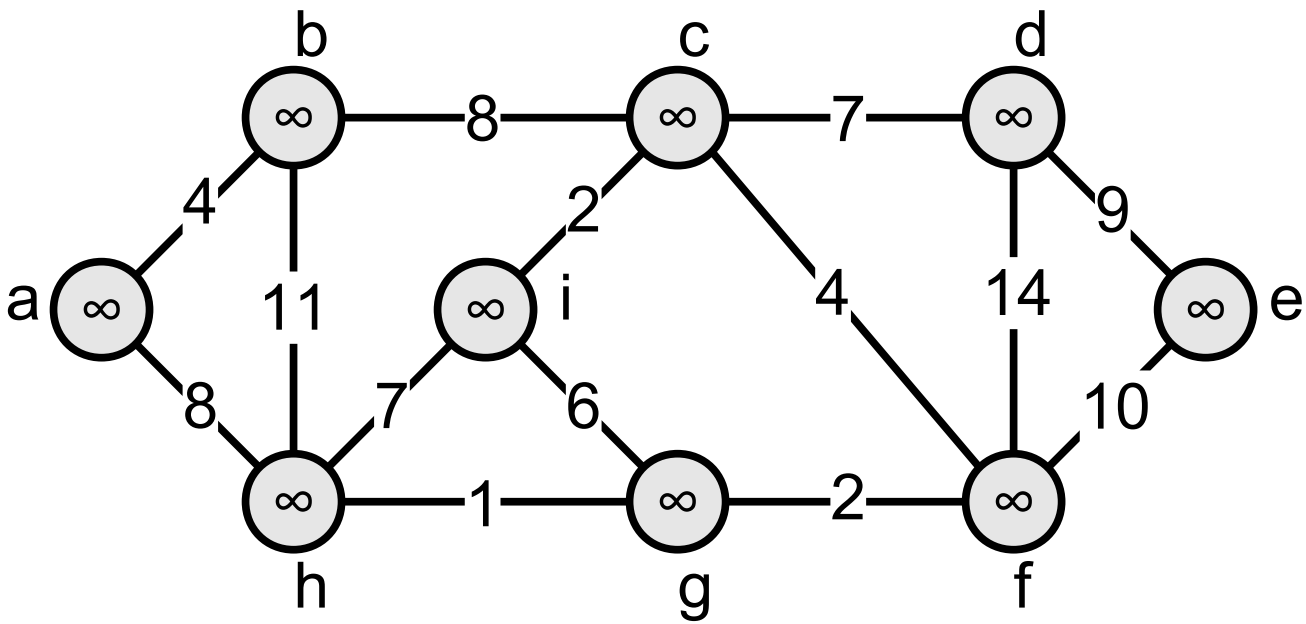
p[r]=-1
p[K]=r
p[S]=K



Minimum Spanning Tree = MST

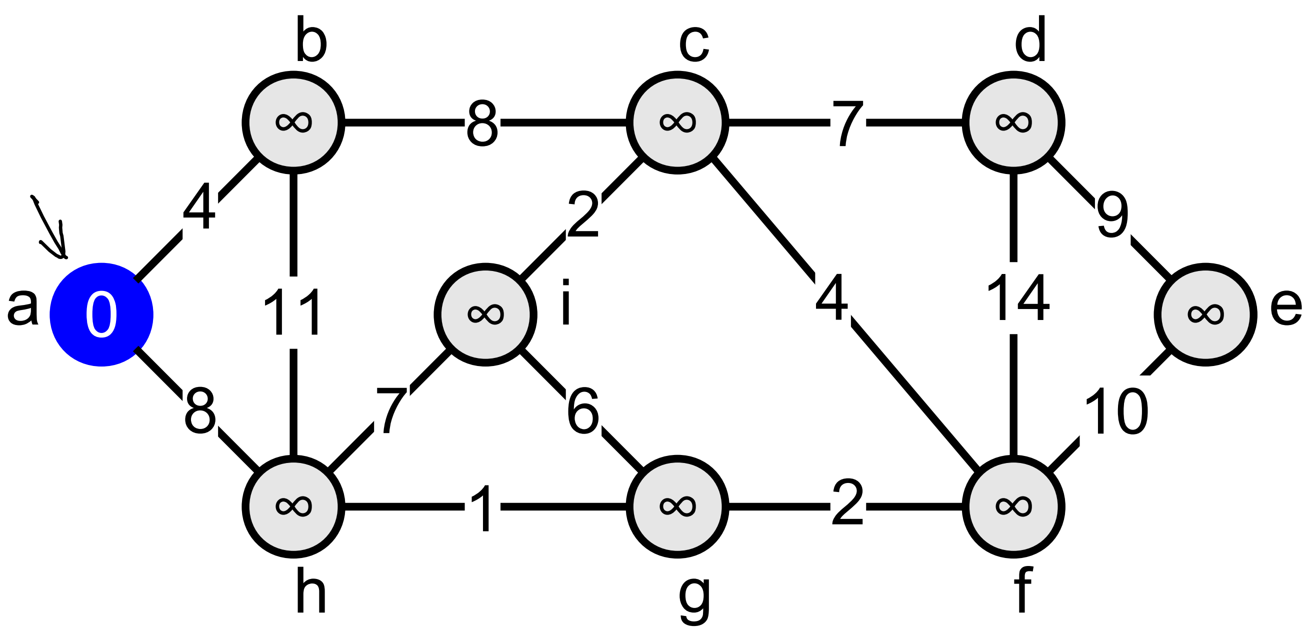
Esempio di esecuzione

Q = { }



Esempio di esecuzione

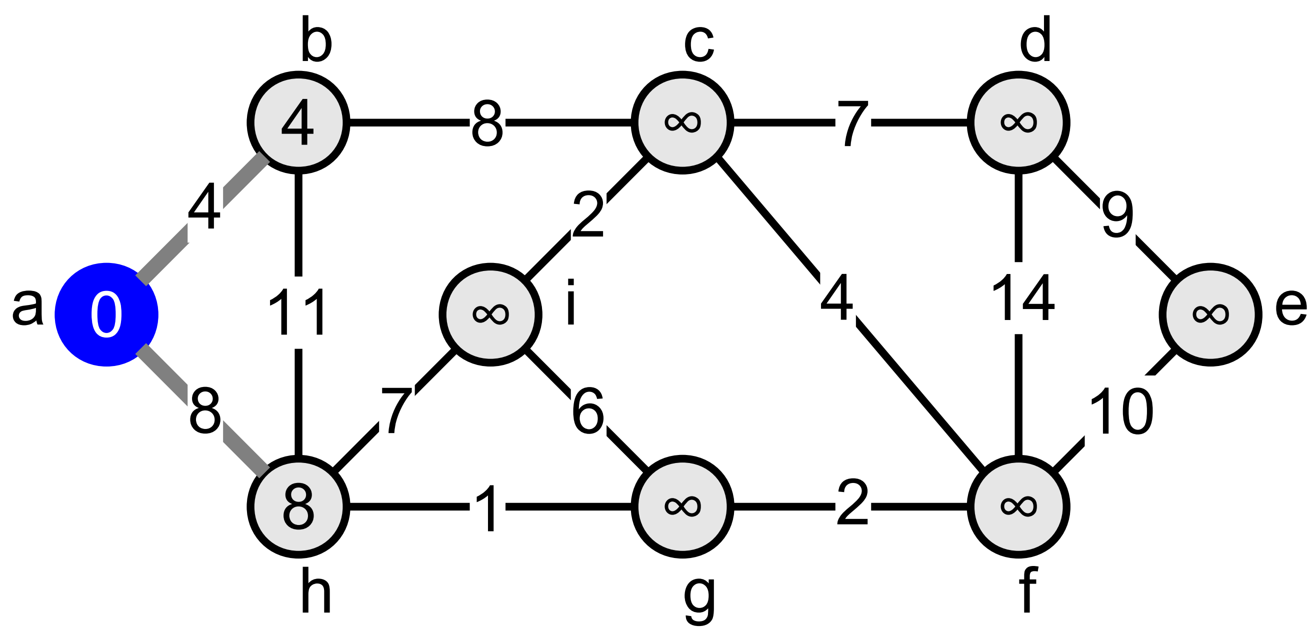
$Q = \{ \underbrace{(a,0)} \}$



Esempio di esecuzione

nodo costo

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

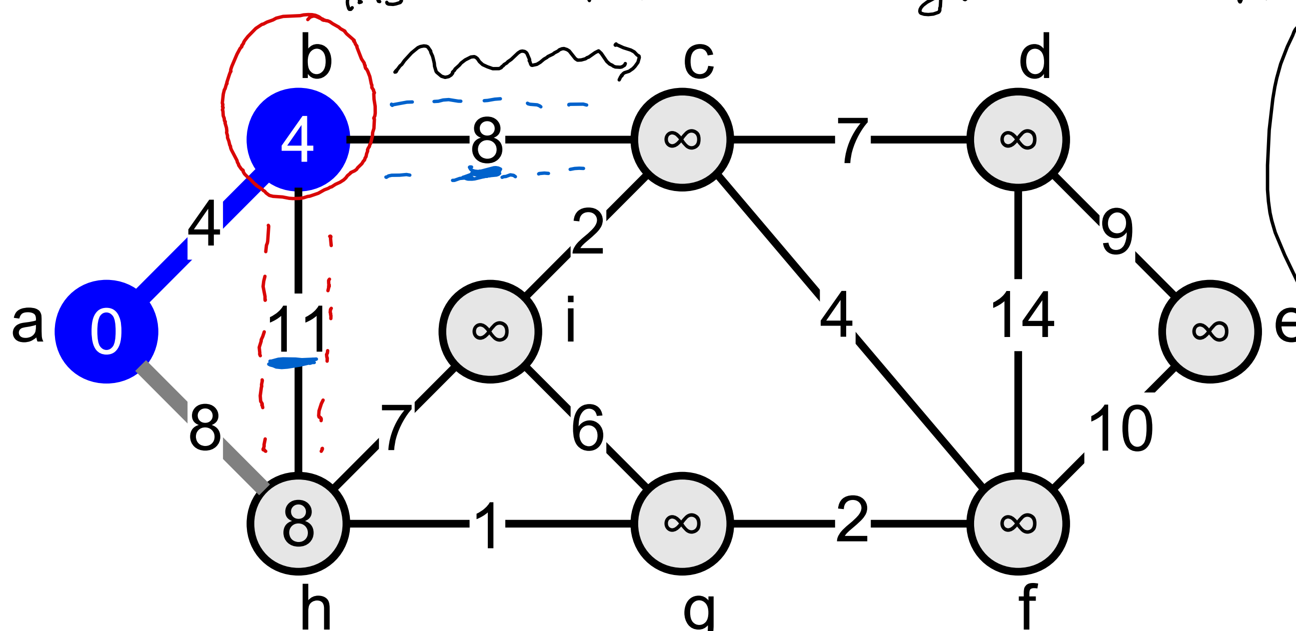


Esempio di esecuzione

estrazione peso minimo

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

inserire nella coda gli adiacenti a b



gli adiacenti ad a sono già trattati

ora posso confrontare il costo 8 con 11.

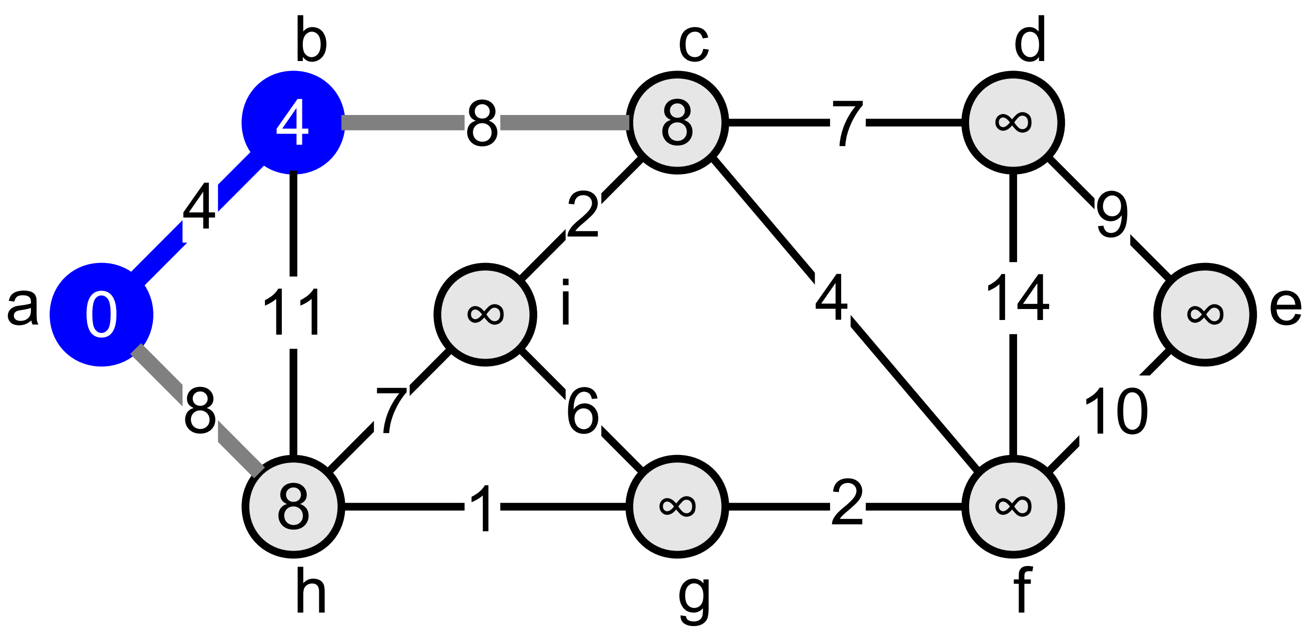
inserisco (c,8) nella coda e (h,11) NON SERVE

E NON LO USO

sappiamo già che h è raggiungibile con costo 8
quindi non serve costo 11.

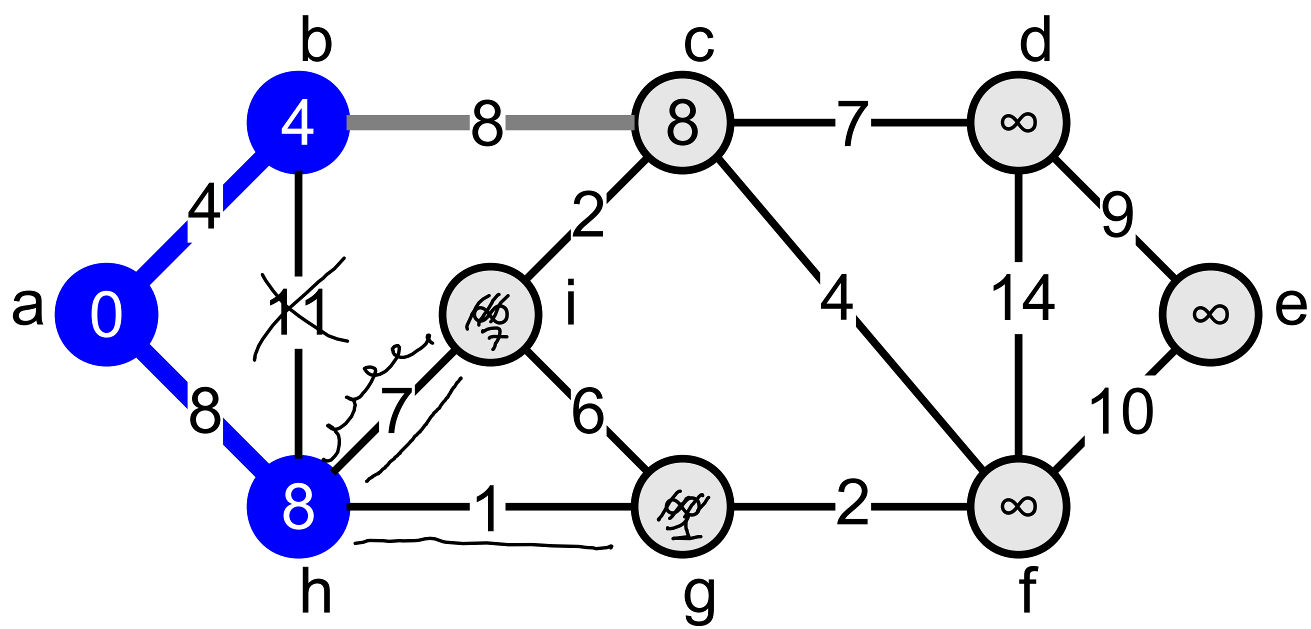
Esempio di esecuzione

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



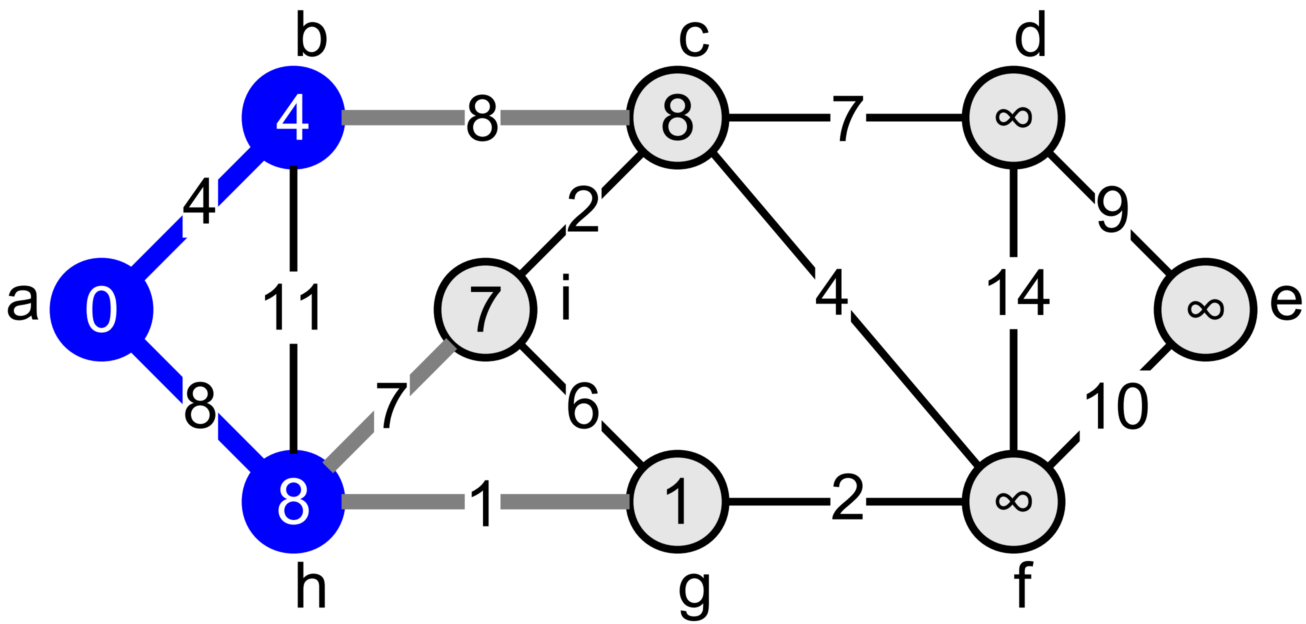
Esempio di esecuzione

$Q = \{ \cancel{(h,8)}, (c,8) \}$



Esempio di esecuzione

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



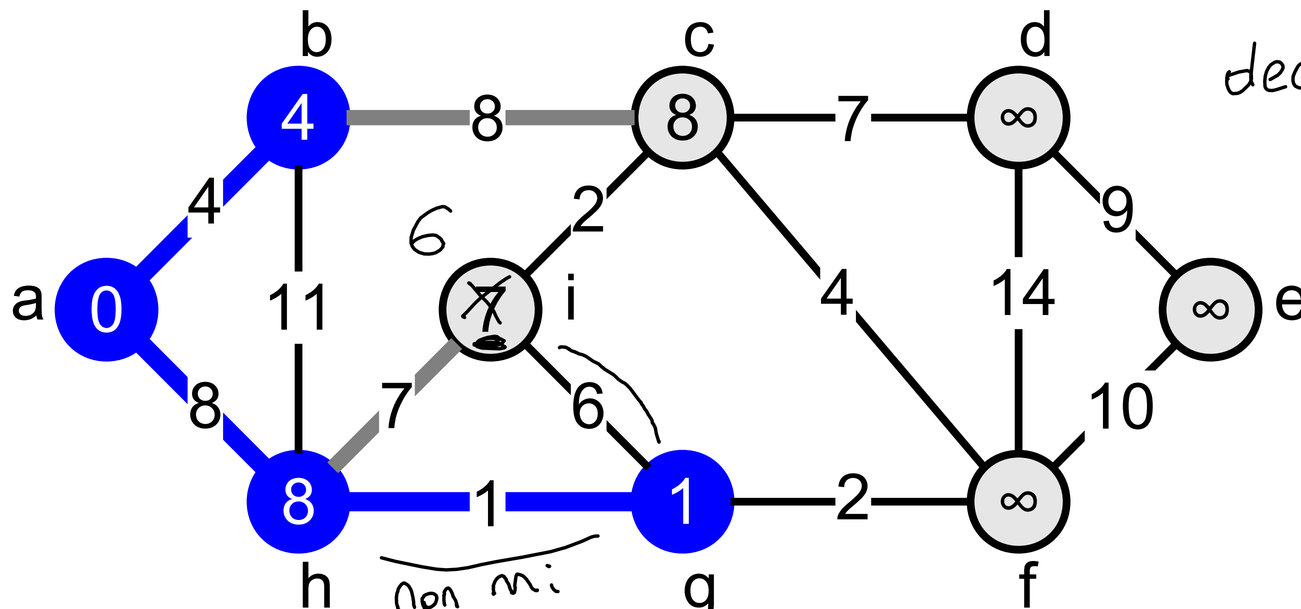
Esempio di esecuzione

↓6 (f,2)

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

obsoleta

devo modificare decreaseKey( )



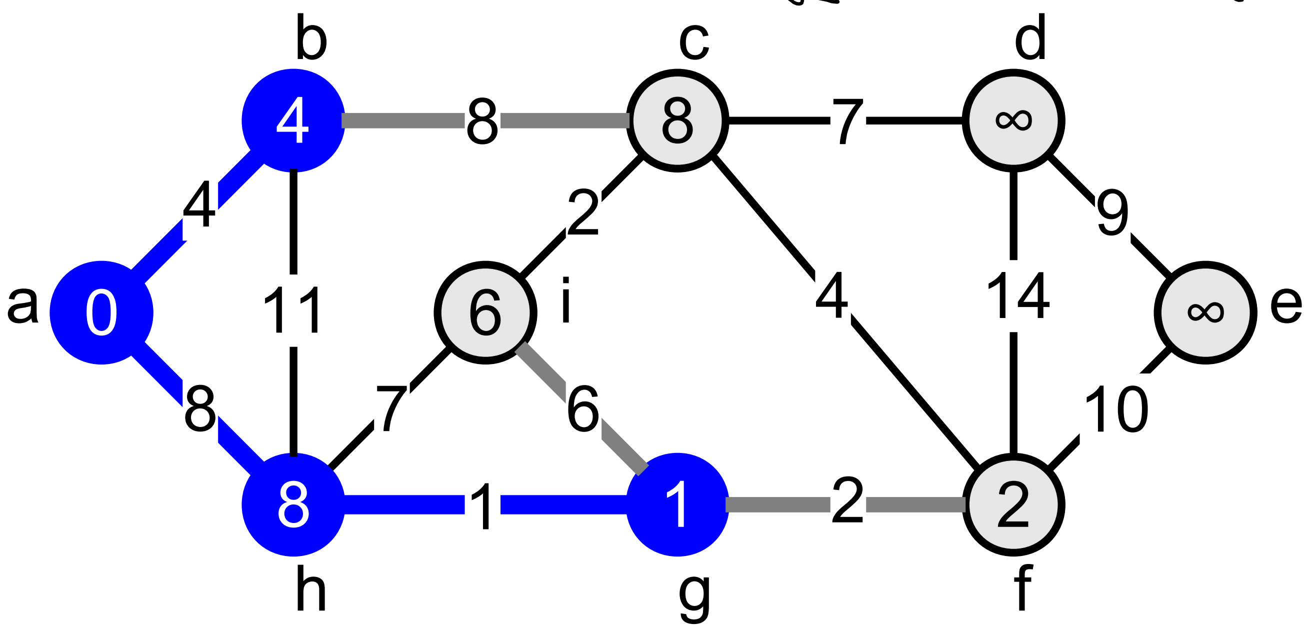
non mi interessa "è già blu"

i risulta raggiungibile con costo inferiore

Esempio di esecuzione

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

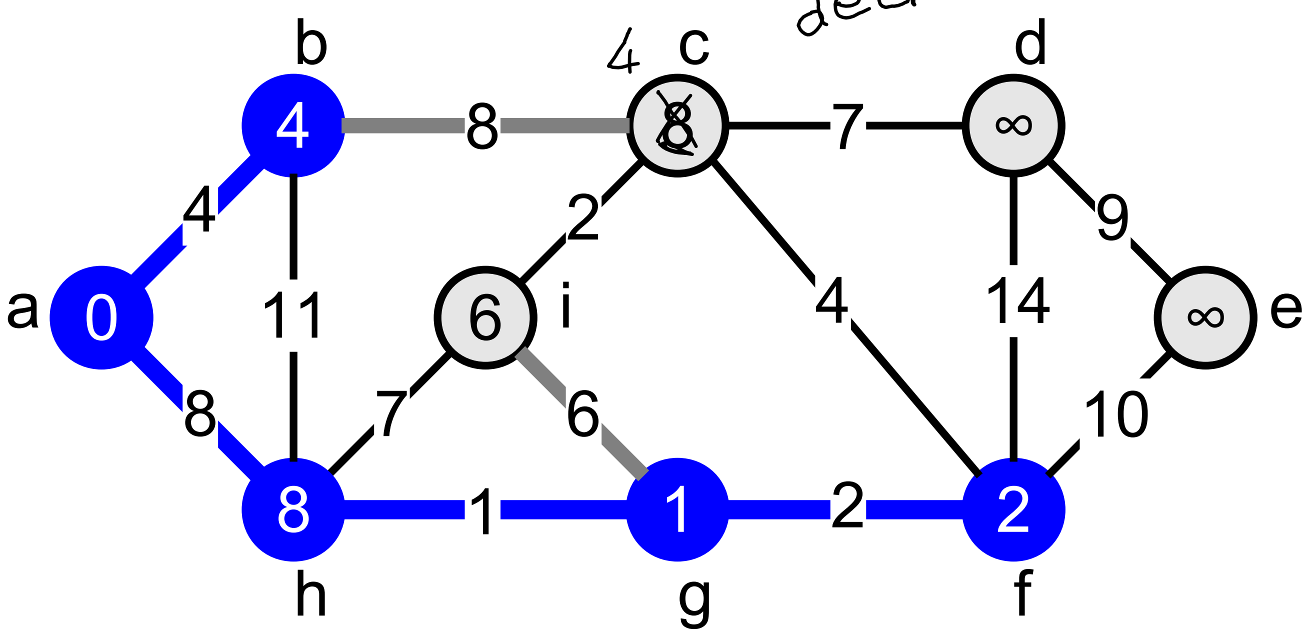
aggiornato dal passo prima



Esempio di esecuzione

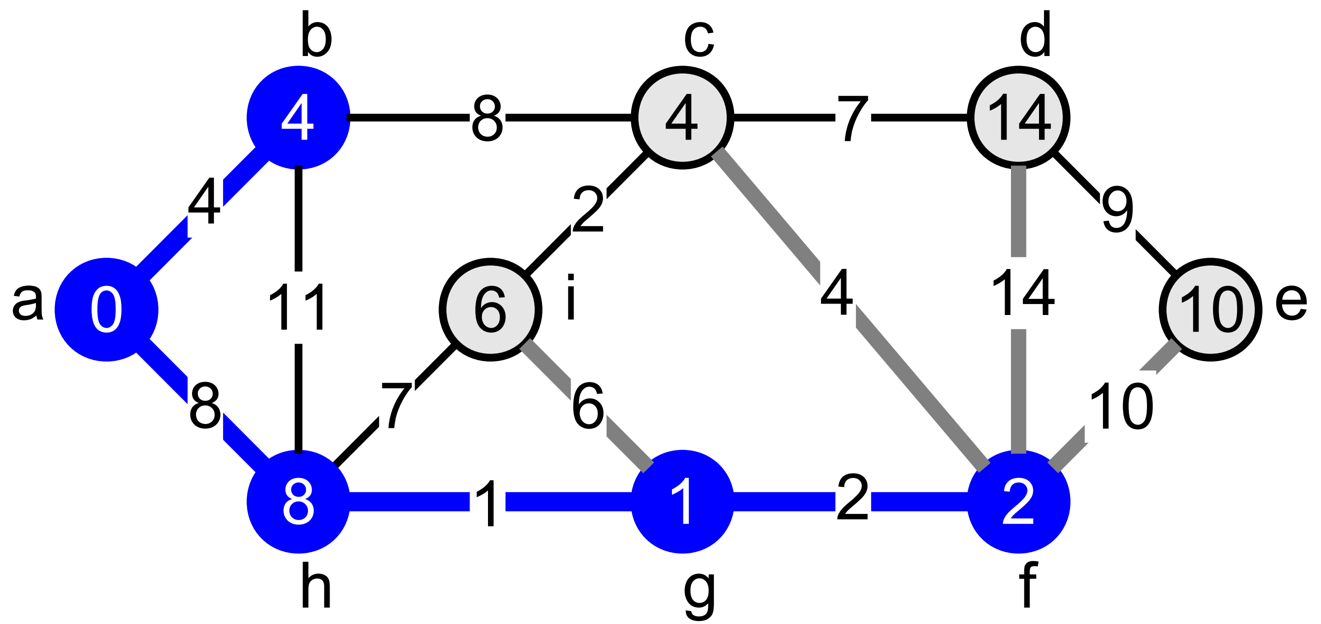
Q = { ~~(f,2)~~, (i,6), (c,8) }

decrease Key ( )



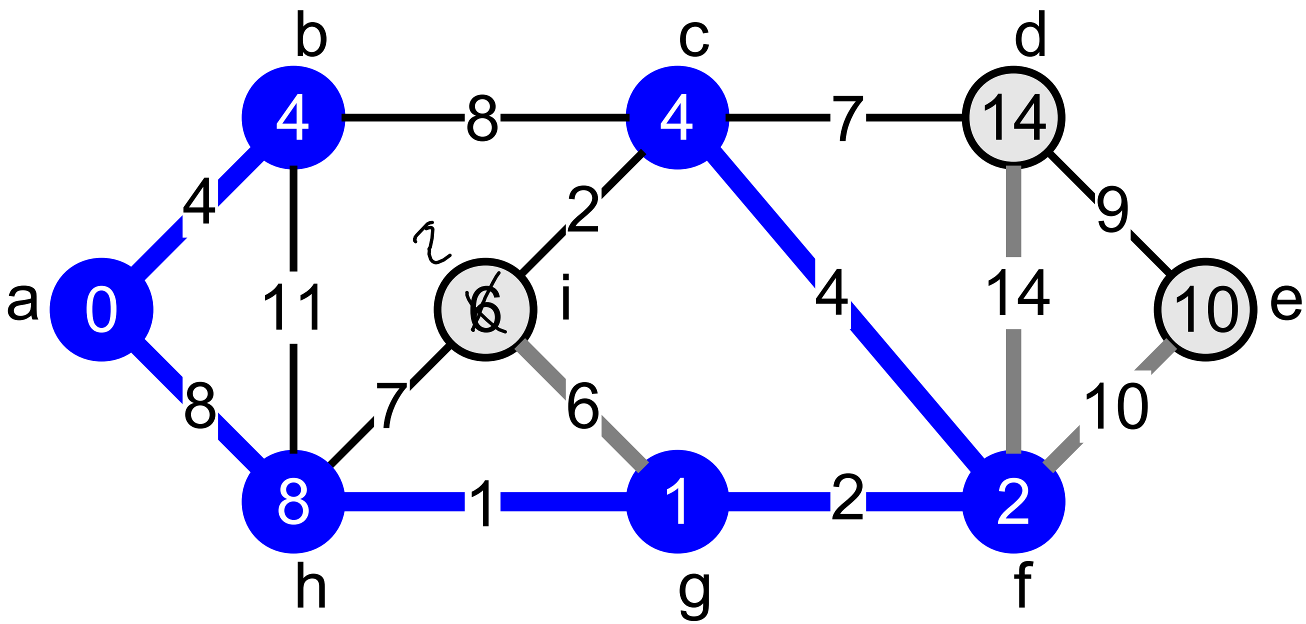
Esempio di esecuzione

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



Esempio di esecuzione

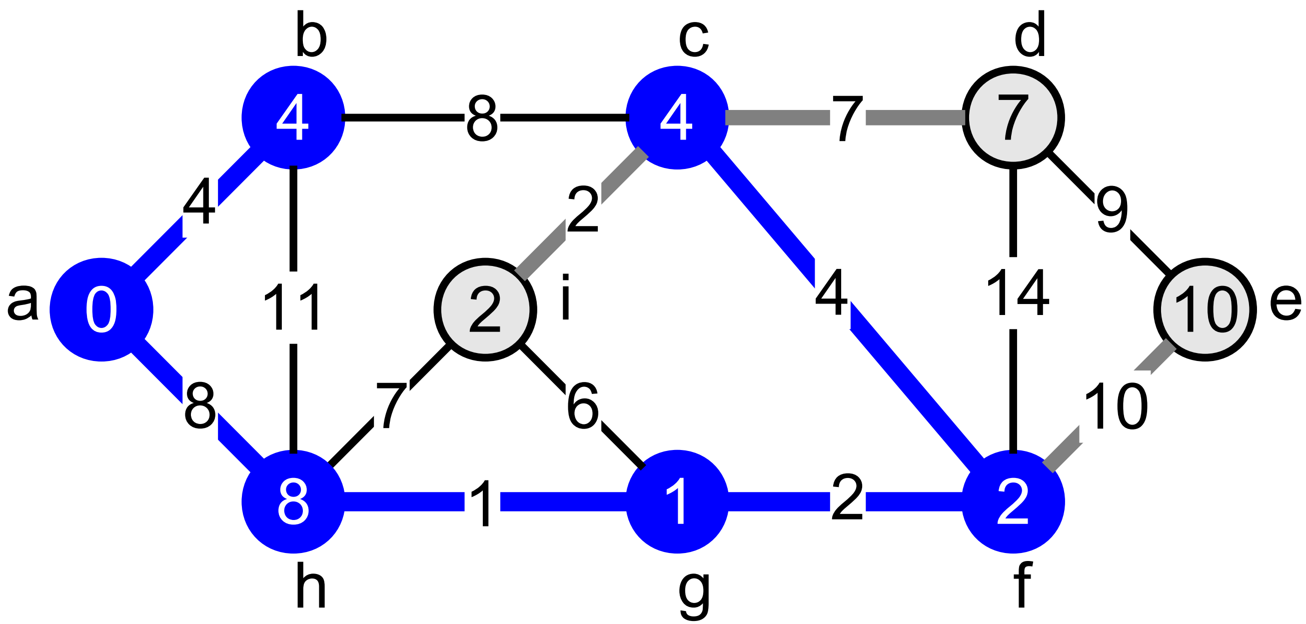
Q = { ~~(c,4)~~, (i,6), (e,10), (d,14) }



decrease Key

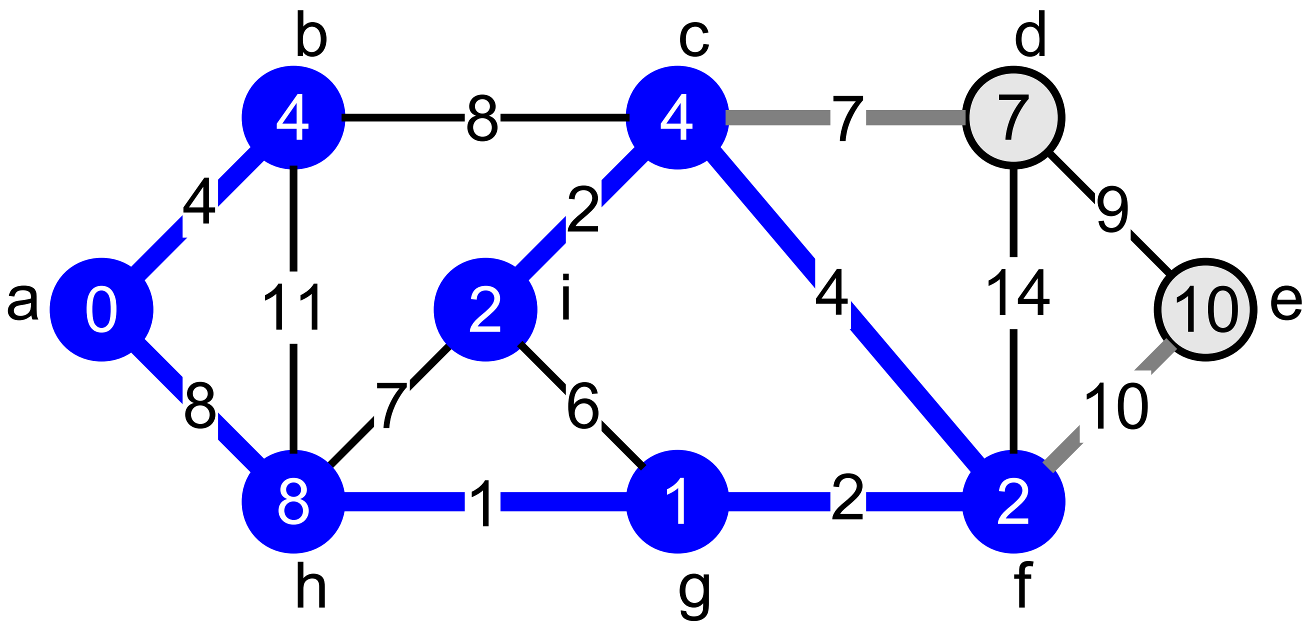
Esempio di esecuzione

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



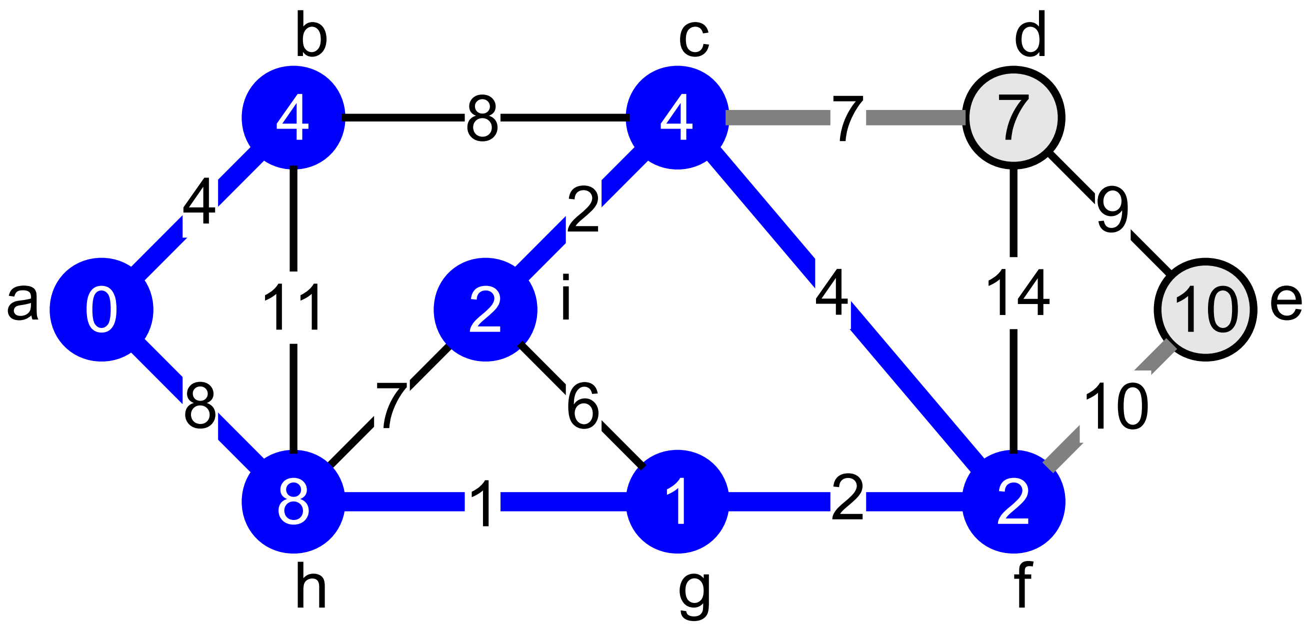
Esempio di esecuzione

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



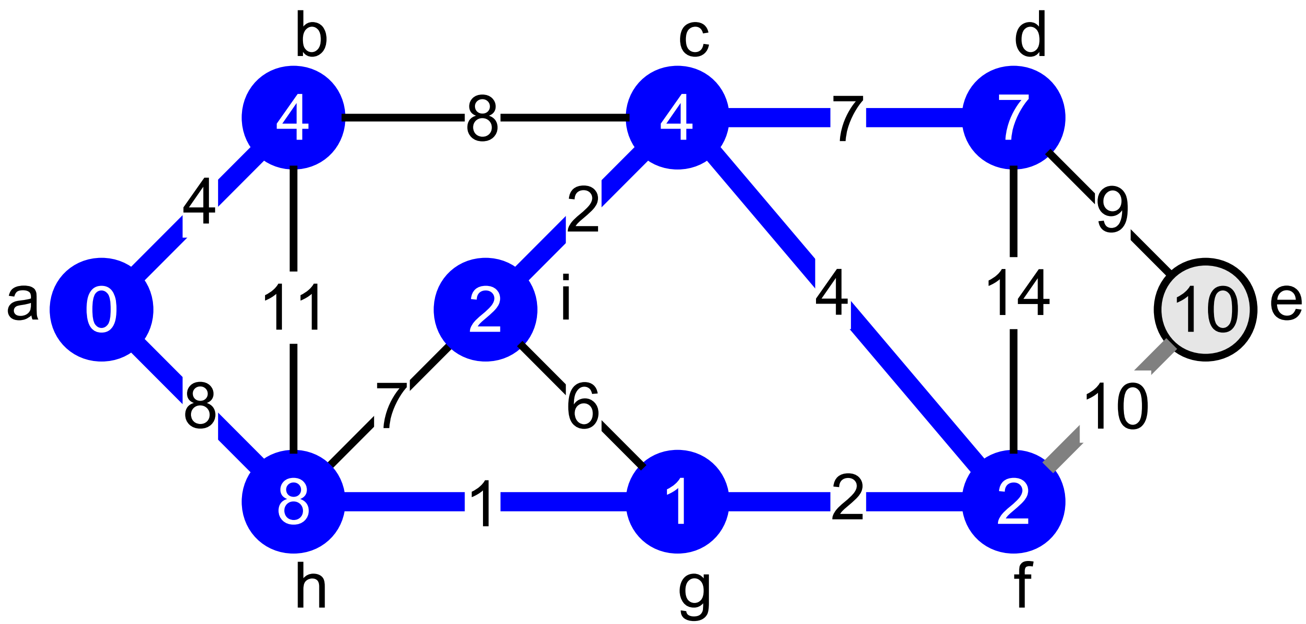
Esempio di esecuzione

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



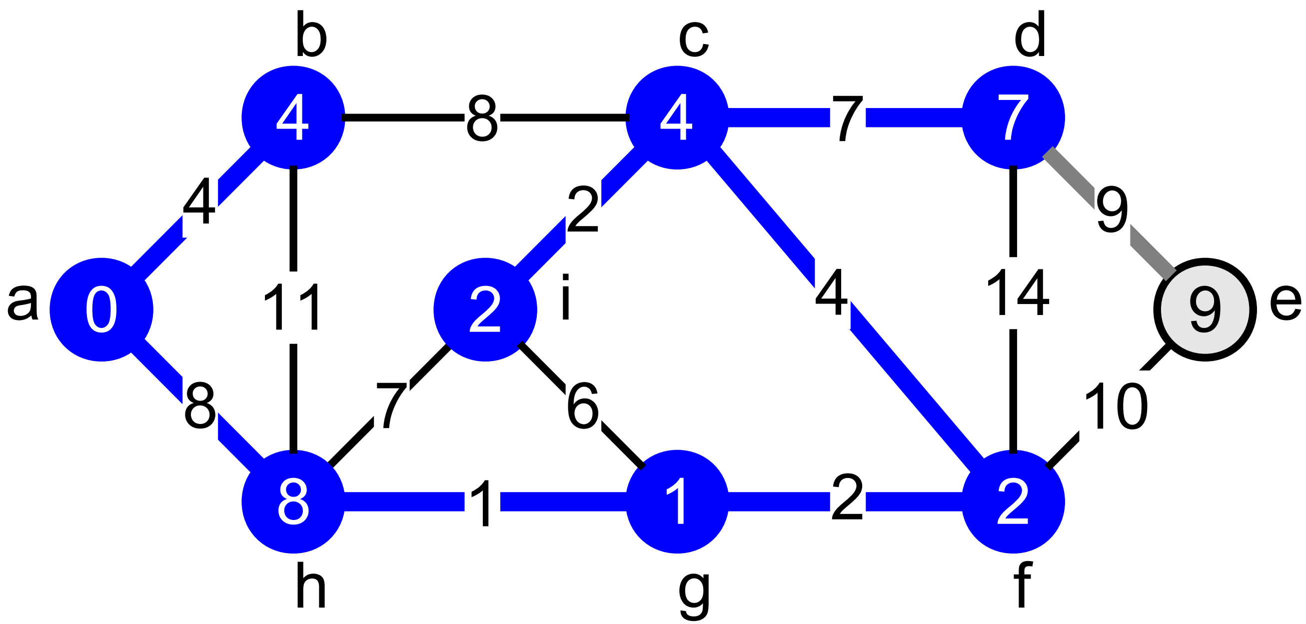
Esempio di esecuzione

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



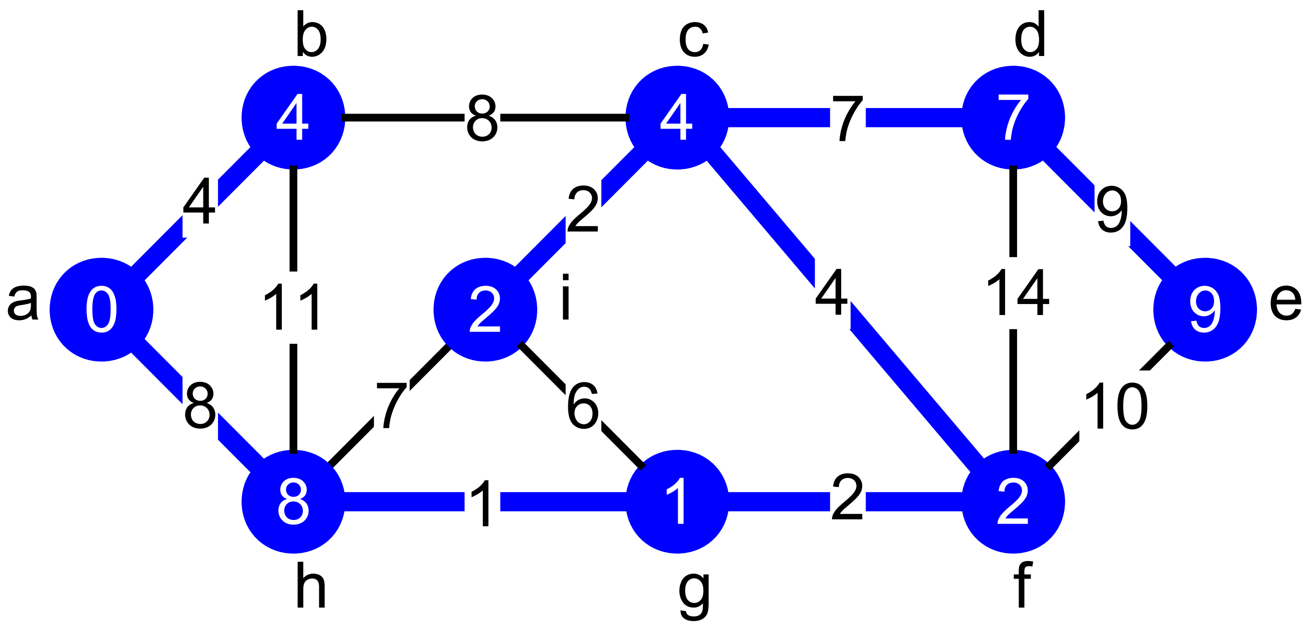
Esempio di esecuzione

Q = { (e,9) }



Esempio di esecuzione

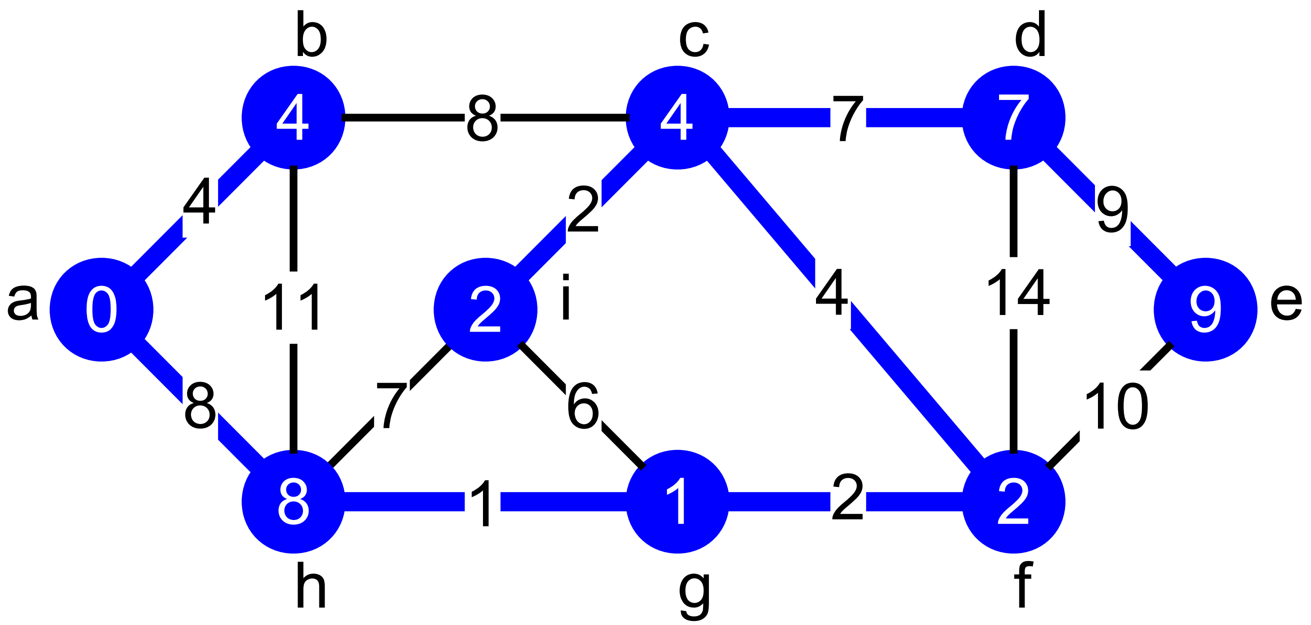
Q = { ~~(e,9)~~ }



Esempio di esecuzione

$Q = \{ \}$

coda svuotata e l'algoritmo termina



Algoritmo di Prim

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

n deleteMin()

n insert()
(inclusa Q.insert(s,0))

O(*m*) decreaseKey()

n cicli

n cicli

viola

Algoritmo di Prim
Costo computazionale

- Utilizzando una coda di priorità basata su min-heap
  - n deleteMin() costano $O(n \log n)$
  - n insert() costano $O(n \log n)$
  - $O(m)$ decreaseKey() costano $O(m \log n)$
- Totale
  - $O(n \log n + n \log n + m \log n) =$
    $O(m \log n + n \log n) =$
    $O(m \log n)$

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

Esercizio 1

- Si consideri un grafo non orientato connesso G = (V, E) i cui archi abbiano tutti lo stesso peso w > 0. Descrivere un algoritmo efficiente che, dato in input il grafo G e il peso w di tutti i suoi archi, determini un Minimum Spanning Tree di G. Determinare il costo computazionale dell'algoritmo proposto.

Esercizio 2

- Proporre un algoritmo efficiente per calcolare un Maximum Spanning Tree di un grafo non orientato pesato G = (V, E, w), in cui ad ogni arco *e* è associato un peso reale *w(e)* non necessariamente positivo (un Maximum Spanning Tree è un albero di copertura di peso totale massimo).

Esercizio 3

- Consideriamo un grafo non orientato, connesso e pesato G = (V, E, w). Mostrare, fornendo un controesempio, che non è sempre possibile costruire un MST contenente un dato arco *e*.