Minimum Spanning Tree

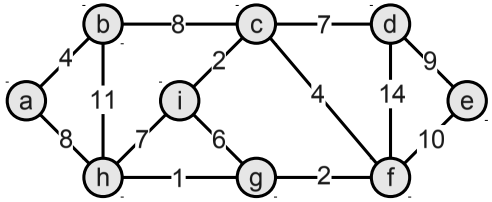
Moreno Marzolla
http://www.moreno.marzolla.name/

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

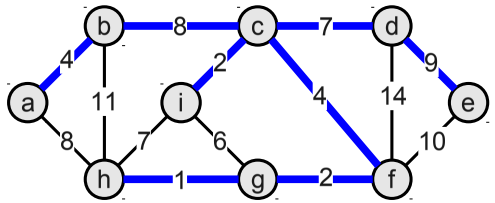
Definizione del problema

- Input:
  - G=(V,E) un grafo non orientato e connesso
  - $w: V \times V \rightarrow \mathbb{R}$ una funzione di peso (costo di connessione)
    - 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 vertici di *G*



Definizione del problema

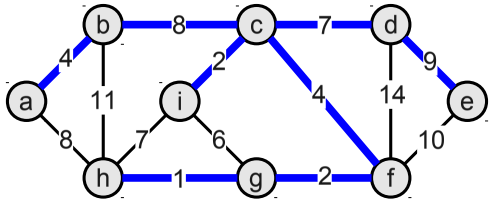
- Output: albero di copertura minimo (minimum spanning tree)
  - Un albero di copertura il cui peso totale

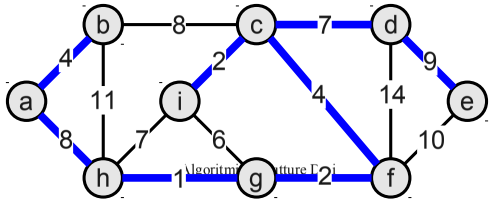
$$w(T) = \sum_{(u,v)\in T} w(u,v)$$

sia minimo

Definizione del problema

- Nota: L'albero di copertura di peso minimo non è necessariamente unico





Algoritmo generico

- Vediamo
  - Un algoritmo di tipo "goloso" *generico*
  - Due "istanze" di questo algoritmo: Kruskal e Prim
- L'idea è di *accrescere* un sottoinsieme *A* di archi in modo tale che venga rispettata la seguente condizione:
  - *A* è un sottoinsieme di qualche albero di connessione minimo
- Un arco *{u,v}* è detto sicuro per *A* se $A \cup \{u,v\}$ è ancora un sottoinsieme di qualche albero di connessione minimo.

Algoritmo generico

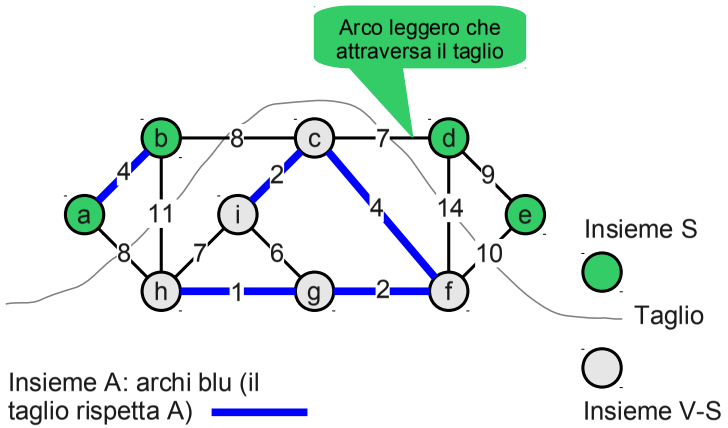
```
algoritmo Generic-MST(Grafo G, w)→albero
1  A := ∅
2  while A non forma un albero di copertura do
3     trova un arco sicuro {u,v}
4     A := A ∪ {u,v}
5  return A
```

- 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 $v \in V\text{-}S$
  - Un taglio *rispetta* un insieme di archi *A* se nessun arco di *A* attraversa il taglio
  - Un arco che attraversa un taglio è *leggero* nel taglio se il suo peso è minimo fra i pesi degli archi che attraversano un taglio

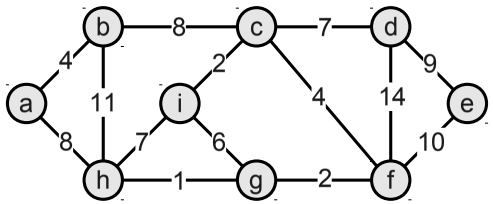
Esempio



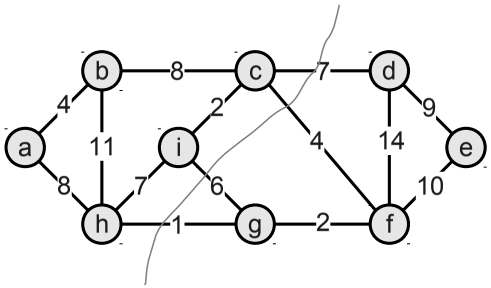
Regole del ciclo e del taglio

- 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 che si possa usare)

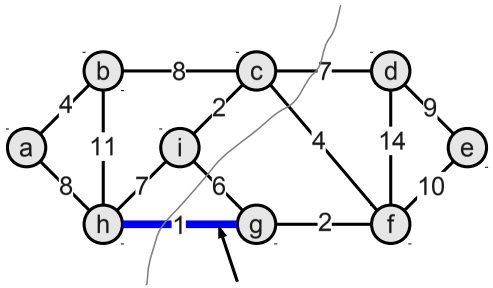
Esempio



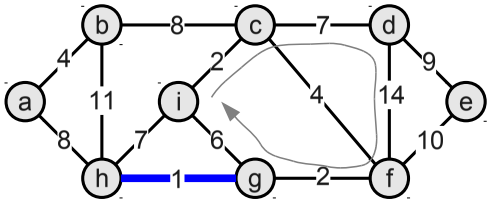
Esempio



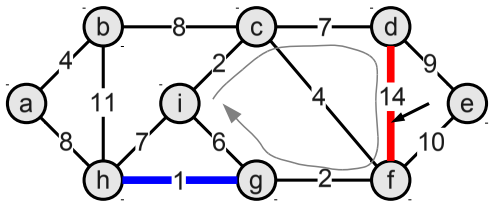
Esempio



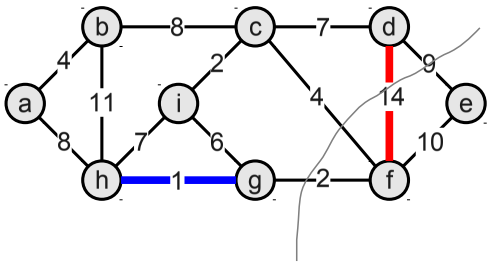
Esempio



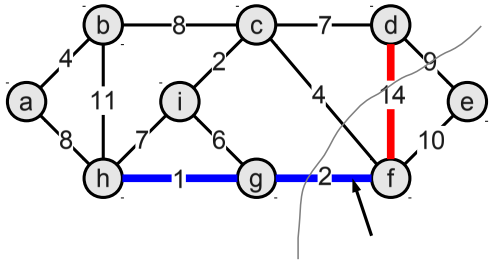
Esempio



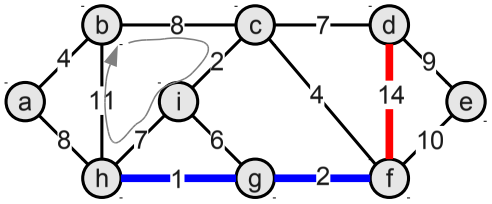
Esempio



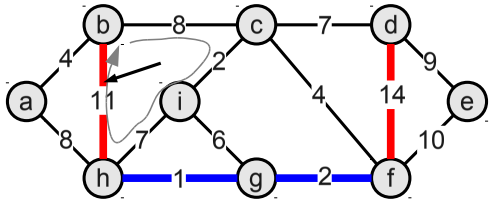
Esempio



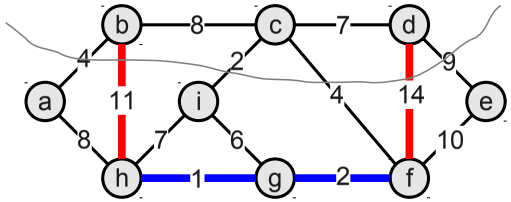
Esempio



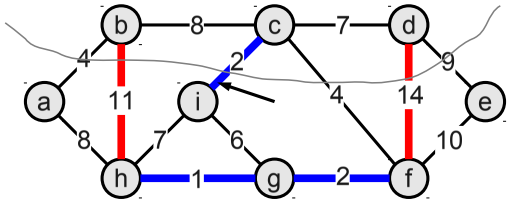
Esempio



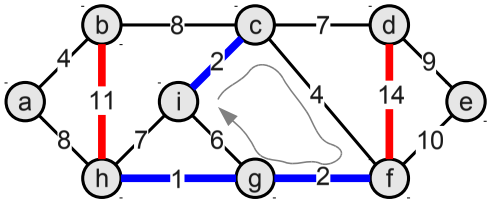
Esempio



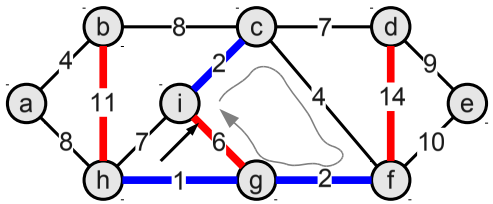
Esempio



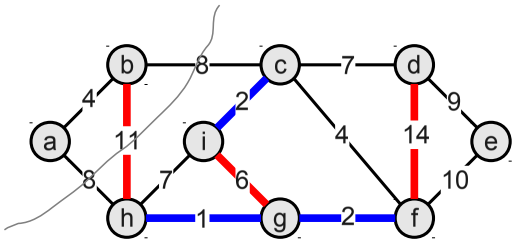
Esempio



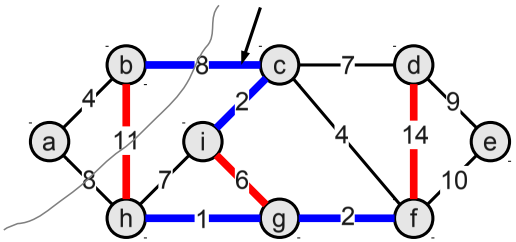
Esempio



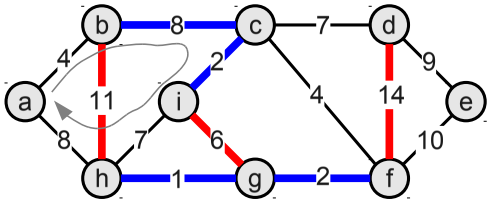
Esempio



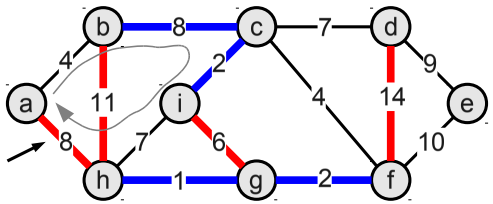
Esempio



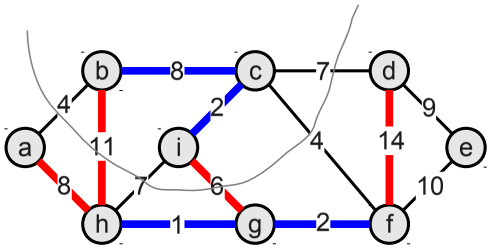
Esempio



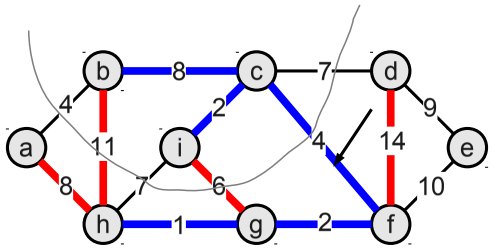
Esempio



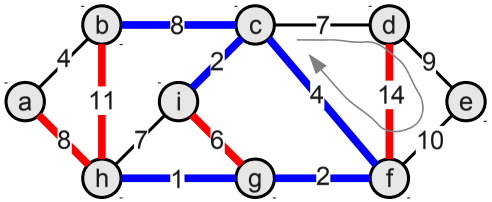
Esempio



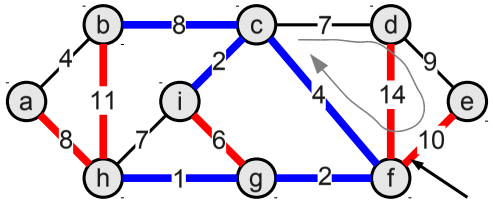
Esempio



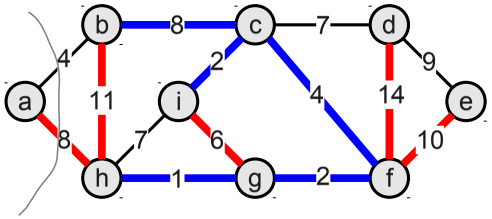
Esempio



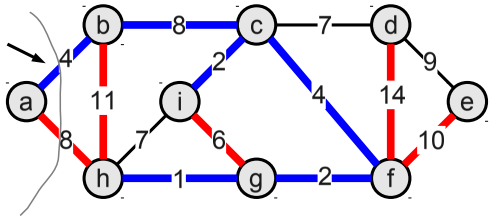
Esempio



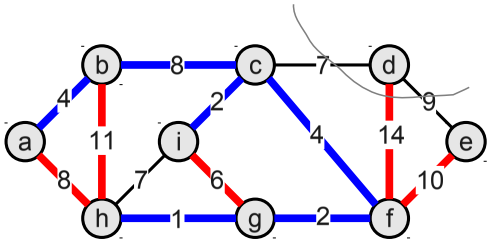
Esempio



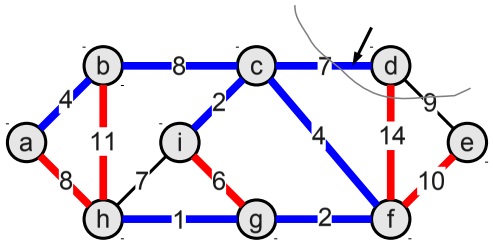
Esempio



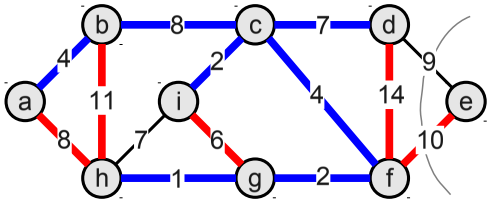
Esempio



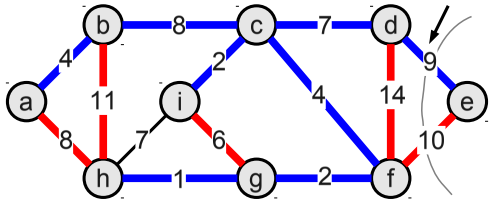
Esempio



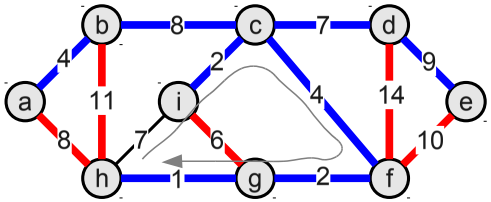
Esempio



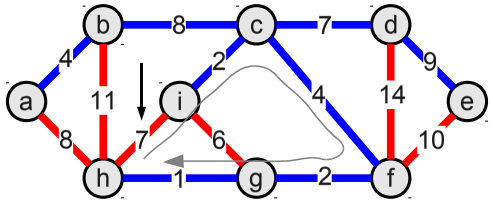
Esempio



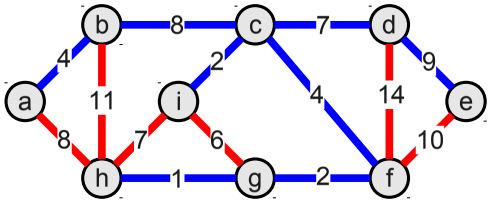
Esempio



Esempio



Finito!



Archi sicuri (regola del taglio)

Teorema:

Sia $G=(V,E)$ un grafo non orientato e connesso

Sia w una funzione peso a valori reali definita su E.

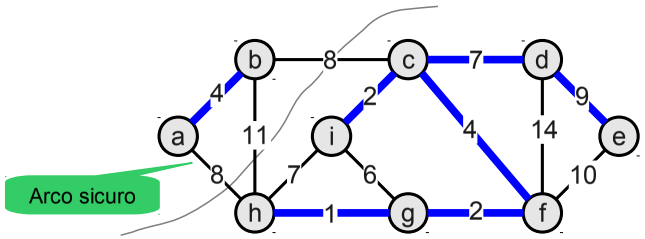
Sia $A \subseteq E$ composto di soli archi blu (A è contenuto in un albero di copertura minimo per G).

Sia (S,V-S) un taglio che rispetta A, ossia che non contiene archi blu

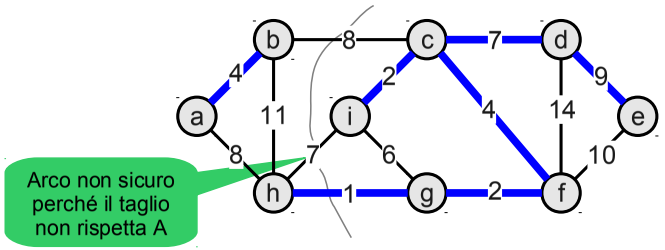
Sia (u,v) un arco leggero che attraversa il taglio.

Allora l'arco (u,v) è sicuro per A

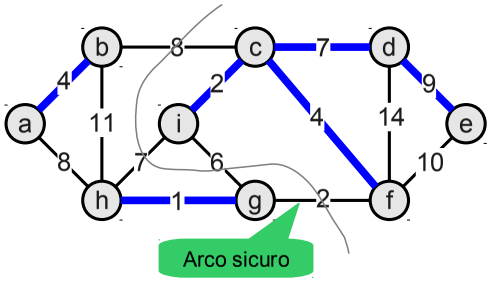
Esempio



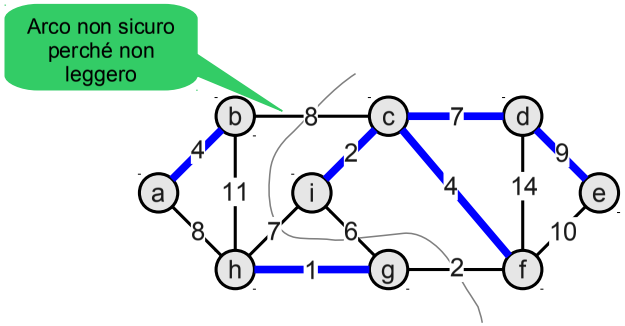
Esempio



Esempio



Esempio



Archi sicuri

Corollario:

Sia $G=(V,E)$ un grafo non orientato e connesso

Sia w una funzione peso a valori reali definita su E.

Sia $A \subseteq E$ contenuto in un albero di copertura minimo per G.

Sia C una componente connessa (un albero) nella foresta $G_A=(V,A)$

Sia (u,v) un arco leggero che connette C a qualche altra componente in G_A

Allora l'arco (u,v) è sicuro per A

Teorema

- Sia G un grafo non orientato, connesso e pesato sugli archi. Ogni passo di colorazione del metodo greedy mantiene la seguente invariante
  - *"Esiste sempre un minimo albero ricoprente di G che contiene tutti gli archi blu, e che non contiene alcun arco rosso"*
- Inoltre, il metodo goloso colora tutti gli archi di G

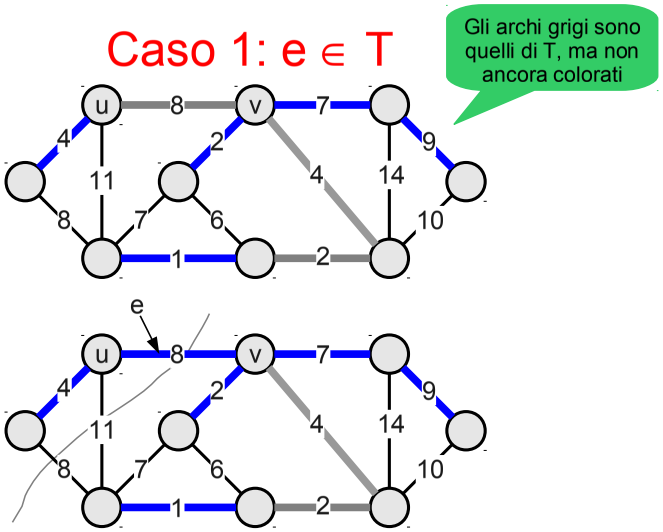
Dimostrazione
per induzione sul numero di passi

- All'inizio (nessun passo di colorazione) l'invariante è banalmente mantenuta perché non ci sono archi colorati, né rossi né blu

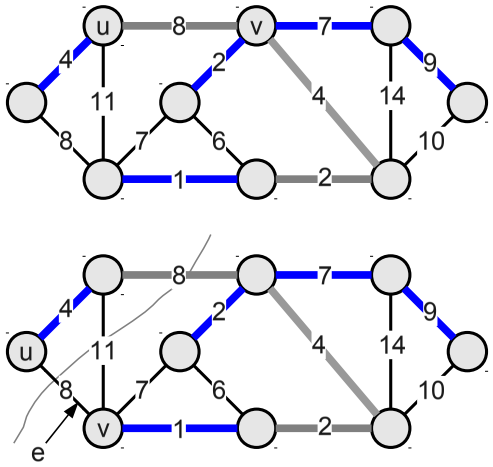
Dimostrazione
per induzione sul numero di passi

- Supponiamo l'invariante sia vera prima di applicare la regola del taglio
- Sia (S, V-S) il taglio considerato, e={u,v} l'arco del taglio di peso minimo che è stato colorato di blu, T il MST di G che verificava l'invariante prima della colorazione di e

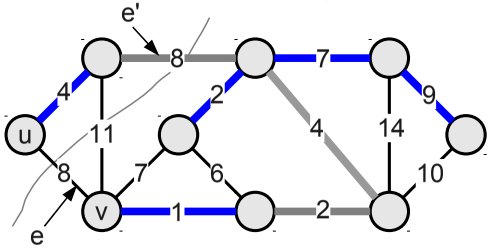
Caso 1: e ∈ T



Caso 2: e ∉ T

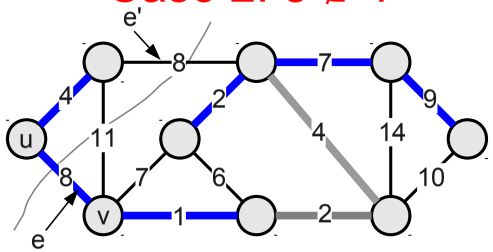


Caso 2: e ∉ T



- Consideriamo il cammino che collega u e v in T
- Deve esistere un arco e' di T che attraversa il taglio
  - Grazie all'invariante, e' sicuramente non è rosso
- Poiché abbiamo applicato la regola del taglio:
  - e' sicuramente non era colorato;
  - w(e) ≤ w(e')

Caso 2: e ∉ T



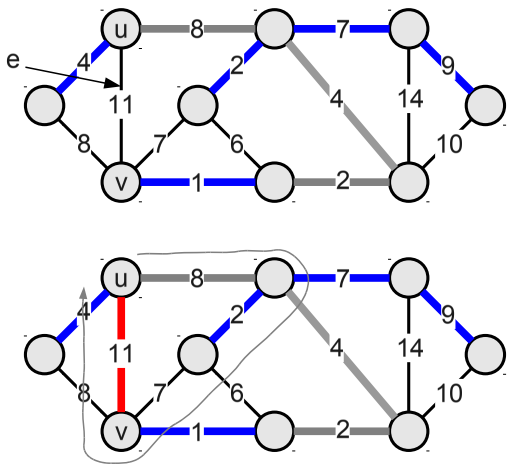
- Poiché abbiamo applicato la regola del taglio:
  - e' sicuramente non era colorato;
  - w(e) ≤ w(e')
- Ma allora l'albero (T – e') U e
  - è un MST
  - soddisfa l'invariante dopo la colorazione di e

Dimostrazione

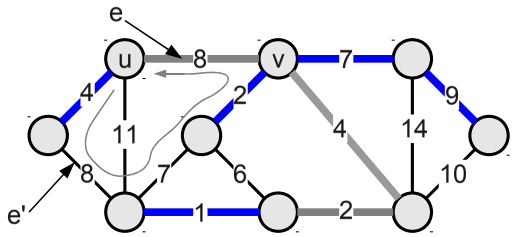
per induzione sul numero di passi

- Supponiamo l'invariante sia vera prima di applicare la regola del ciclo
- Sia e={u,v} l'arco del ciclo che è stato colorato di rosso, T il MST di G che verificava l'invariante prima della colorazione di e

Caso 1: e ∉ T

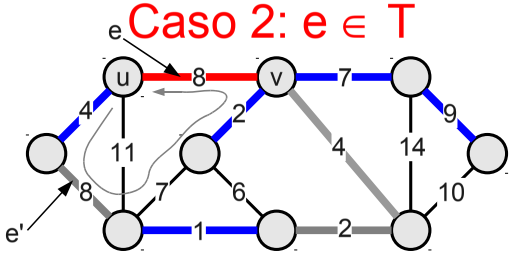


Caso 2: e ∈ T



- Se coloriamo di rosso e, spezziamo l'albero T in due sottoalberi, uno che contiene u e l'altro che contiene v
- Consideriamo il ciclo che abbiamo usato per colorare e={u,v}
  - in tale ciclo deve esserci un altro arco e' con gli estremi nei due sottoalberi
  - e' non era rosso, altrimenti non potevamo applicare la regola del ciclo
  - e' non era nemmeno blu, altrimenti T non sarebbe stato un albero

Caso 2: e ∈ T



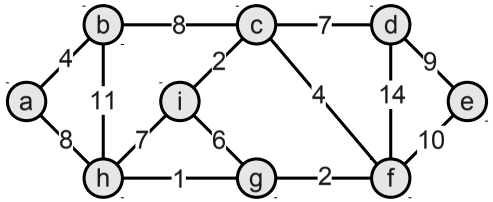
- Poiché la regola del ciclo ha colorato e, risulta che w(e) ≥ w(e')
- Ma allora l'albero T' = (T – e) U e' verificherà l'invariante dopo l'applicazione della regola del ciclo

Algoritmo di Kruskal

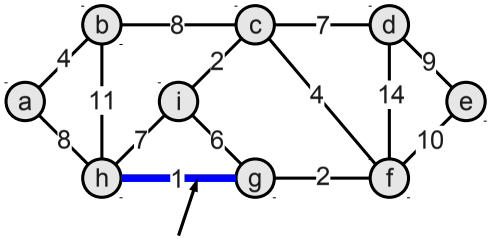
- Idea
  - Ingrandire sottoinsiemi disgiunti di un albero di copertura minimo connettendoli fra di loro fino ad avere l'albero complessivo
    - Inizialmente la foresta di copertura è composta da n alberi, un albero per ciascun nodo, e nessun arco
  - Si considerano gli archi in ordine non decrescente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

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

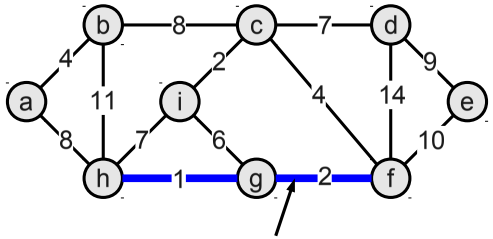
Esempio
Algoritmo di Kruskal



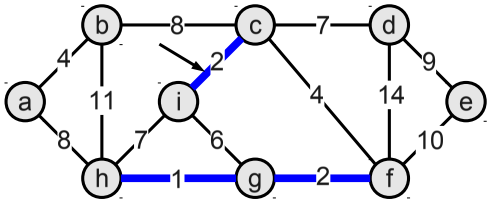
Esempio
Algoritmo di Kruskal



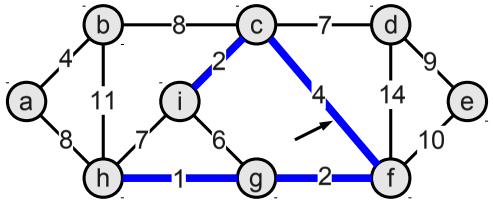
Esempio
Algoritmo di Kruskal



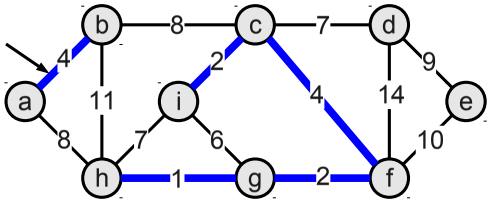
Esempio
Algoritmo di Kruskal



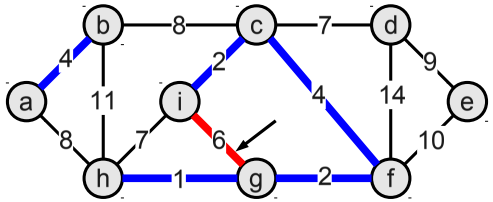
Esempio
Algoritmo di Kruskal



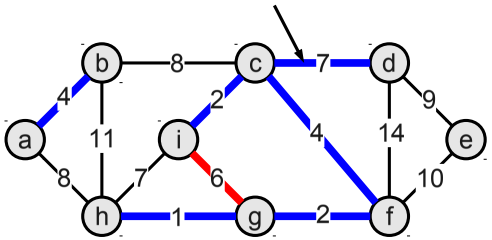
Esempio
Algoritmo di Kruskal



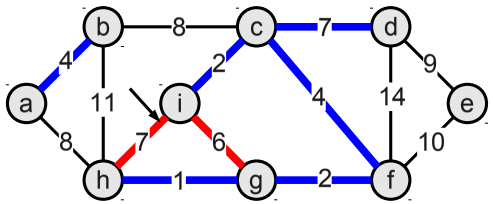
Esempio
Algoritmo di Kruskal



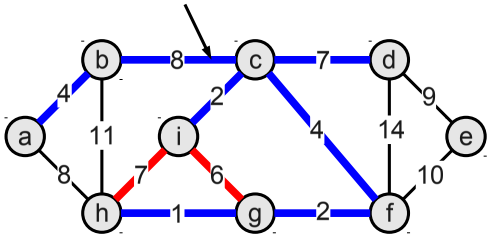
Esempio
Algoritmo di Kruskal



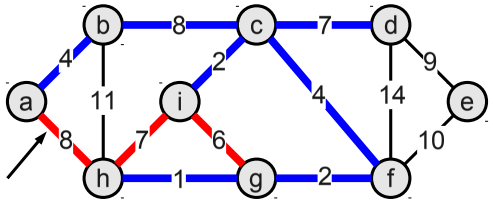
Esempio
Algoritmo di Kruskal



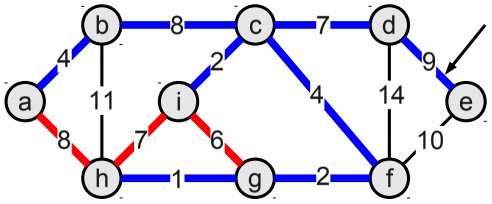
Esempio
Algoritmo di Kruskal



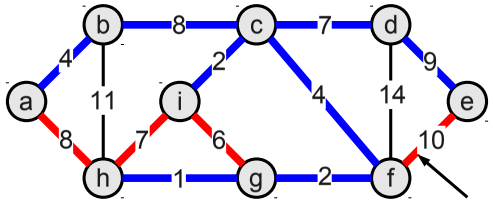
Esempio
Algoritmo di Kruskal



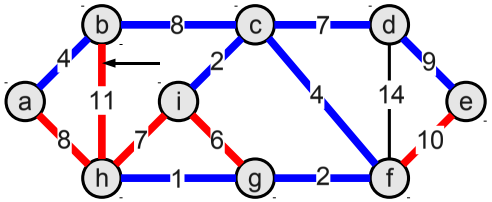
Esempio
Algoritmo di Kruskal



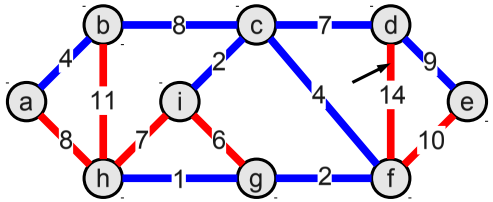
Esempio
Algoritmo di Kruskal



Esempio
Algoritmo di Kruskal



Esempio
Algoritmo di Kruskal



- Finito! Il MST è composto dai soli archi blu

Algoritmo di Kruskal: 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

```
Algoritmo Kruskal-MST(Grafo G=(V,E), w)->albero
    T := albero vuoto          // foresta iniziale
    UnionFind UF;
    for each v in V do
        UF.makeSet(v)
    endfor
    // 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 {u,v}      // aggiungi arco
            UF.union(Tx,Ty)     // unisci componenti
        endif
    endfor
    return T
```


Analisi

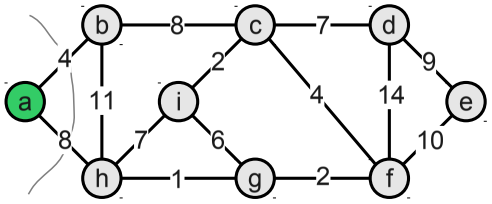
- L'ordinamento richiede
  $O(m \log m) = O(m \log n^2) = O(m \log n)$
- Il tempo di esecuzione dipende dalla realizzazione della struttura dati per insiemi disgiunti
  - Vengono effettuate n makeset, 2m find e (n-1) union
- Se usiamo quickUnion con euristica sul rango, la sequenza di operazioni costa in tutto $O(m + n \log n)$
- Totale: $O(m \log n + m + n \log n) = O(m \log n)$

Algoritmo di Prim

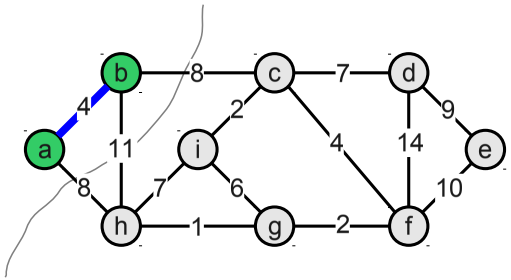
- L'algoritmo di Prim procede mantenendo in *A* un singolo albero
  - L'albero parte da un vertice arbitrario *r* (la *radice*) e cresce fino a quando non ricopre tutti i vertici
  - Ad ogni passo viene aggiunto un arco leggero che collega un vertice in V_A con un vertice in $V\text{-}V_A$
    - dove V_A è l'insieme di nodi raggiunti da archi in *A*
- Correttezza
  - $(V_A, V\text{-}V_A)$ è un taglio che rispetta *A* (per definizione)
  - Per il corollario, gli archi leggeri che attraversano il taglio sono sicuri

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

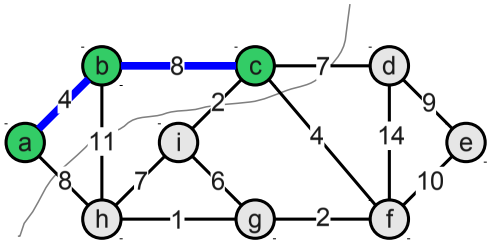
Esempio Algoritmo di Prim



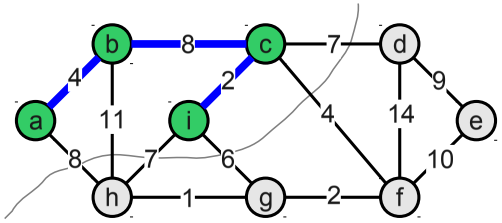
Esempio
Algoritmo di Prim



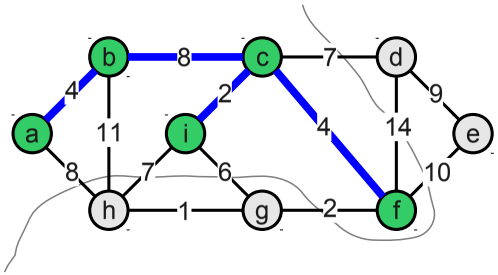
Esempio
Algoritmo di Prim



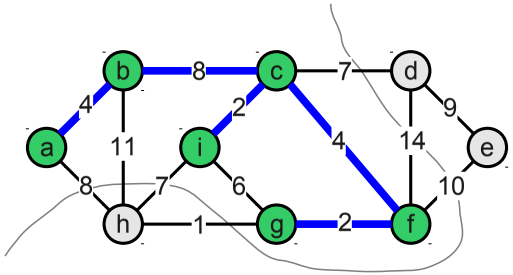
Esempio
Algoritmo di Prim



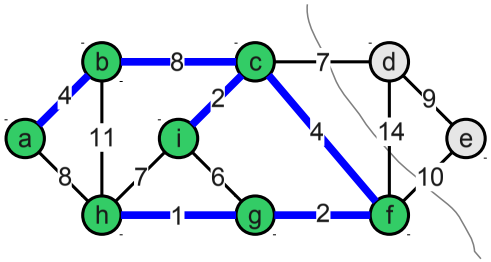
Esempio
Algoritmo di Prim



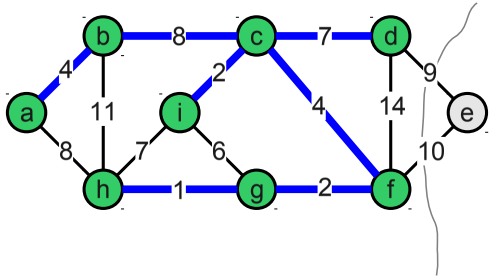
Esempio
Algoritmo di Prim



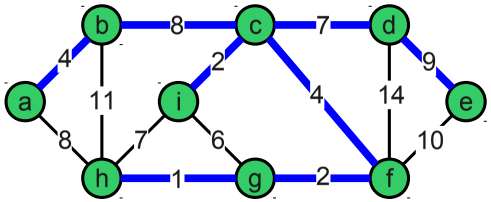
Esempio
Algoritmo di Prim



Esempio
Algoritmo di Prim



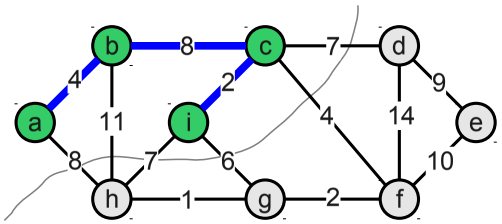
Esempio
Algoritmo di Prim



Implementazione

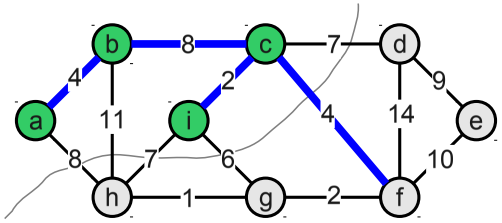
- Una struttura dati per i nodi non ancora nell'albero
  - Durante l'esecuzione, i vertici non ancora nell'albero si trovano in una *coda con priorità Q* ordinata in base ad un campo *d(v)*
  - *d(v)* è il peso minimo di un arco che collega *v* ad un vertice nell'albero, o $+\infty$ se tale arco non esiste
- Come mantenere l'albero
  - Ogni nodo v mantiene un puntatore al padre *v.p*
  - T è mantenuto implicitamente: $T = \{ (v, v.p) \mid v \in V\text{-}Q\text{-}\{r\} \}$
- Terminazione: quando l'insieme Q è vuoto
  - Tutti i nodi tranne la radice conoscono il proprio padre

Esempio Algoritmo di Prim



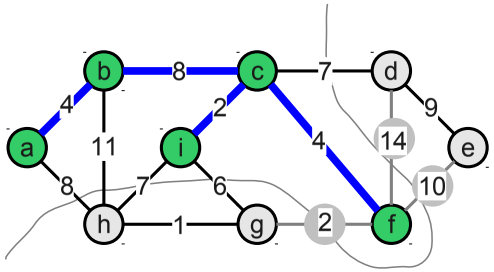
Q = { (f,4), (g,6), (h,7), (d,7) }

Esempio Algoritmo di Prim



Q = { ~~(f,4)~~, (g,6), (h,7), (d,7) }

Esempio Algoritmo di Prim



Q = { (g,6), (h,7), (d,7) }

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

Algoritmo di Prim

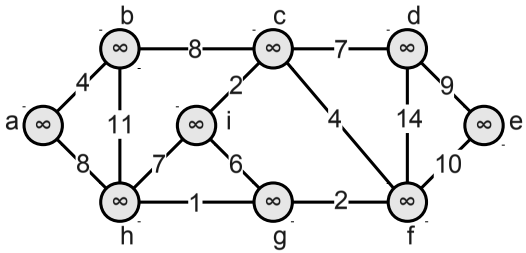
```
algoritmo Prim(Grafo G=(V,E), nodo s)→albero
   for each v in V do d(v) := ∞ endfor
   T := albero formato dal solo nodo s
   CodaPriorita Q;
   d(s) := 0;
   Q.insert(s,0);
   while (not Q.isEmpty()) do
      u := Q.deleteMin();
      for each {u,v}∈E do
         if (d(v) = ∞) then
            Q.insert(v,w(u,v));
            d(v) := w(u,v);
            rendi u padre di v in T;
         else if (w(u,v) < d(v)) then
            Q.decreaseKey(v,w(u,v));
            d(v) := w(u,v);
            rendi u nuovo padre di v in T;
         endif
      endfor
   endwhile
   return T;
```

d(v) è il peso minimo dell'arco che connette v con l'albero T; +∞ se tale arco non esiste

setta nuovo peso di v a w(u,v)

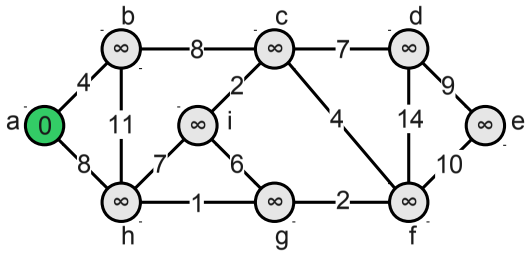
Esempio di esecuzione

Q = { }



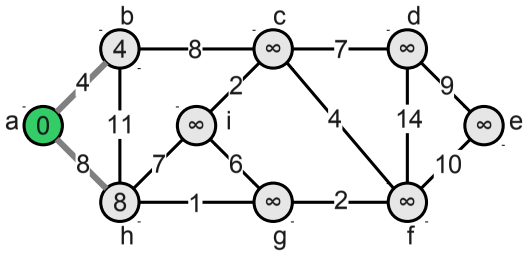
Esempio di esecuzione

Q = { (a,0) }



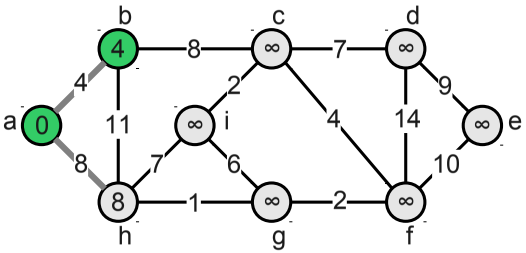
Esempio di esecuzione

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



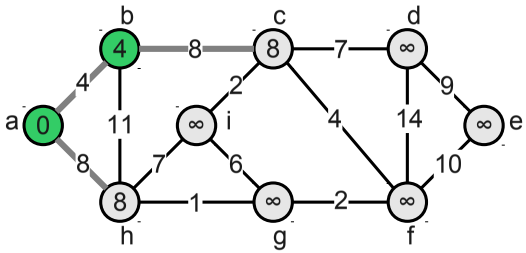
Esempio di esecuzione

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



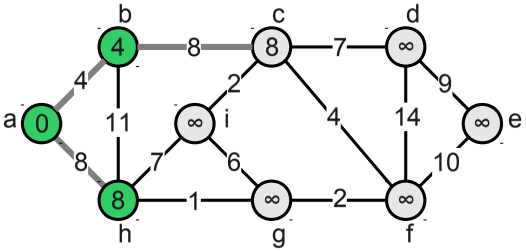
Esempio di esecuzione

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



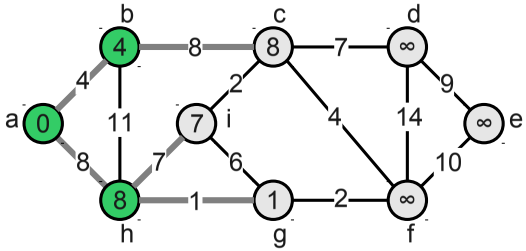
Esempio di esecuzione

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



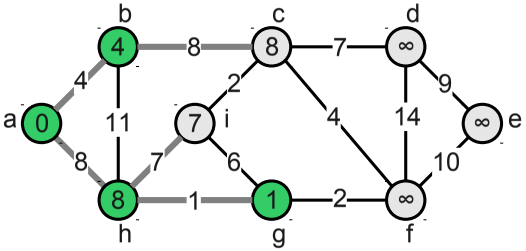
Esempio di esecuzione

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



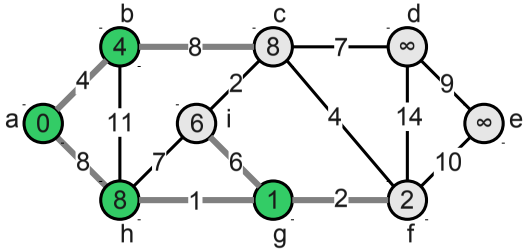
Esempio di esecuzione

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



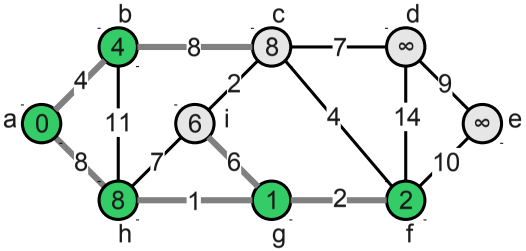
Esempio di esecuzione

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



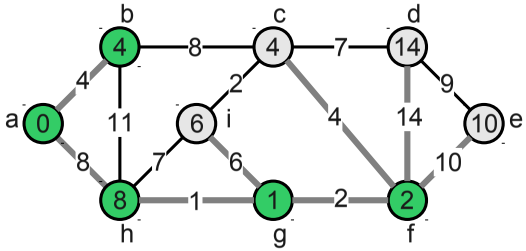
Esempio di esecuzione

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



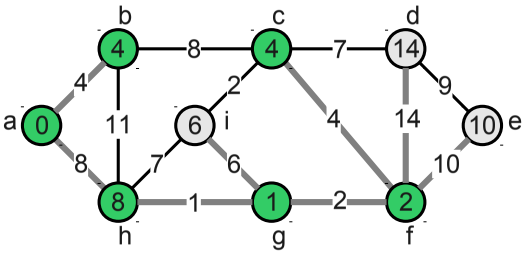
Esempio di esecuzione

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



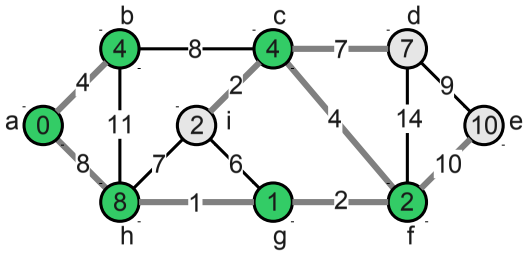
Esempio di esecuzione

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



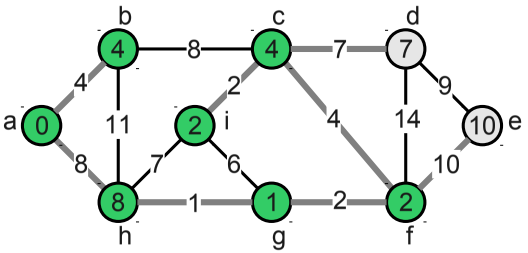
Esempio di esecuzione

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



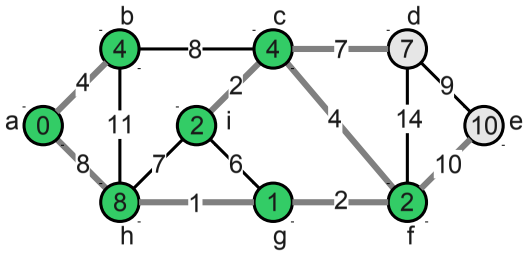
Esempio di esecuzione

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



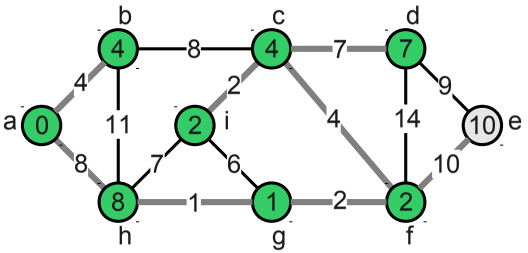
Esempio di esecuzione

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



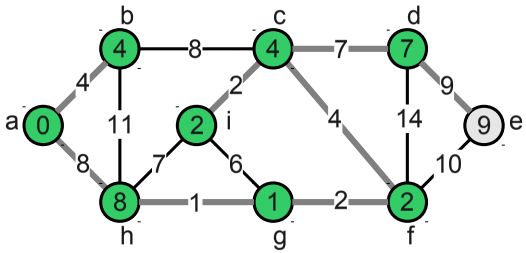
Esempio di esecuzione

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



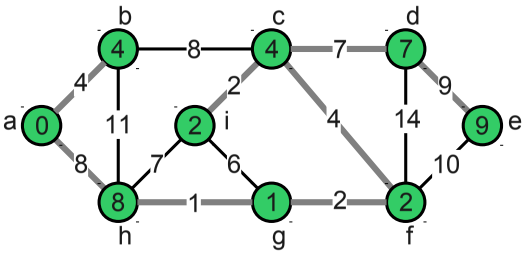
Esempio di esecuzione

Q = { (e,9) }



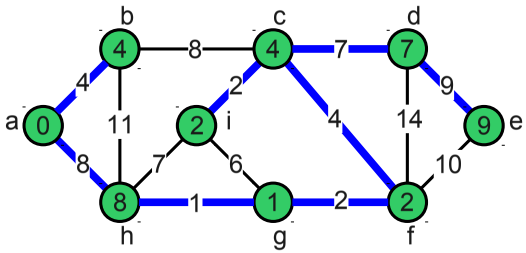
Esempio di esecuzione

Q = { (e,9) }



Esempio di esecuzione

Q = { }



Algoritmo di Prim

```
algoritmo Prim(Grafo G=(V,E), nodo s)→albero
   for each v in V do d(v) := ∞ endfor
   T := albero formato dal solo nodo s
   CodaPriorita Q;
   d(s) := 0;
   Q.insert(s,0);
   while (not Q.isEmpty()) do
      u := Q.deleteMin();
      for each {u,v}∈E do
         if (d(v) = ∞) then
            Q.insert(v,w(u,v));
            d(v) := w(u,v);
            rendi u padre di v in T;
         else if (w(u,v) < d(v)) then
            Q.decreaseKey(v,w(u,v));
            d(v) := w(u,v);
            rendi u nuovo padre di v in T;
         endif
      endfor
   endwhile
   return T;
```

vengono eseguite n operazioni deleteMin()

vengono eseguite n operazioni insert() (inclusa Q.insert(s,0))

vengono eseguite O(m) decreaseKey()

Algoritmo di Prim
Costo computazionale

- Utilizzando heap binari
  - 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)

Ricordiamo che il grafo G deve essere connesso, quindi m ≥ n-1