Grafi: alberi di copertura minimi

Fulvio CORNO - Matteo SONZA REORDA
Dip. Automatica e Informatica
Politecnico di Torino

Sommario

- Introduzione
- Algoritmo generico
- Algoritmo di Kruskal
- Algoritmo di Prim.

Sommario

- Introduzione
- Algoritmo generico
- Algoritmo di Kruskal
- Algoritmo di Prim.

Introduzione

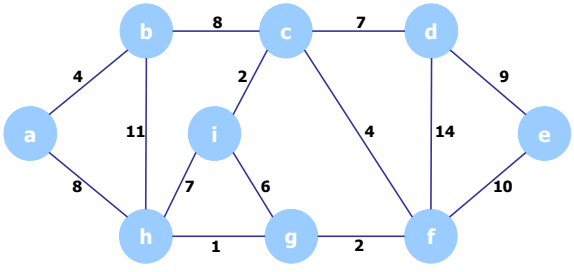
Si consideri un grafo non orientato, pesato e connesso G=(V,E), con una funzione peso w: E→***R***.
Il grafo G' avente le seguenti caratteristiche:

- è composto da
  - stessi vertici di G
  - un sottoinsieme T degli archi E ($T \subseteq E$)
- è aciclico
- minimizza la quantità

$$w(T)=\sum w(u,v)$$

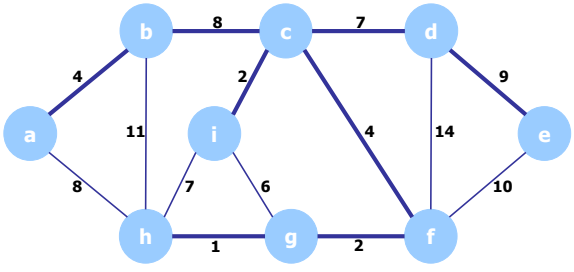
è detto albero di copertura minimo (o *Minimum Spanning Tree*, MST).

Esempio



Esempio

Albero di copertura minimo
Peso totale = 37



Nota

L'albero di copertura minimo per un grafo in generale non è unico.

Esempio di applicazione

Si supponga di dover realizzare l'impianto di distribuzione dell'energia elettrica di una zona, nella quale esistono un certo numero di abitazioni che devono essere servite.

Di ogni abitazione si conosce la posizione, nonché la distanza da tutte le altre.

L'albero di copertura minimo rappresenta la soluzione che consente la minimizzazione della lunghezza dei collegamenti da realizzare.

Sommario

Algoritmo generico

Gli algoritmi per il calcolo di un albero di copertura minimo che seguono si basano sullo stesso schema elementare.

Tale schema implementa un approccio di tipo greedy.

Algoritmi greedy

Sono una particolare categoria di algoritmi di ricerca o ottimizzazione.

In molti problemi ad ogni passo l'algoritmo deve fare una scelta: seguendo la strategia greedy (*ingordo*), l'algoritmo fa la scelta più conveniente in quel momento.

In tal modo non è in generale garantito che si ottenga la soluzione globalmente ottima.

Pseudo-codice

```
Generic-MST(G,w)
1  A←∅
2  while A non forma un albero di copertura
3     do trova un arco (u,v) che sia sicuro per A
4           A←A∪{(u,v)}
5  return A
```

Pseudo-codice

```
Generic-MST(G,w)
1  A←∅
2  while A non forma un albero di copertura
3     do trova un arco (u,v) che sia sicuro per A
4           A←A∪{(u,v)}
5  return A
```

Si definisce *sicuro* un arco che aggiunto ad un sottoinsieme di un albero di copertura minimo produce ancora un sottoinsieme di un albero di copertura minimo

Tagli

Dato un grafo non orientato G=(V,E), un taglio è una partizione di V in due sottoinsiemi.

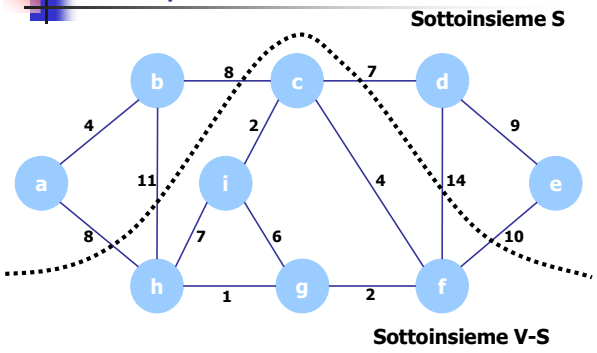
Un arco attraversa il taglio se uno dei suoi estremi è in una partizione, e l'altro nell'altra.

Un taglio rispetta un insieme A di archi se nessun arco di A attraversa il taglio.

Un arco si dice leggero se ha peso minimo tra gli archi che attraversano il taglio.

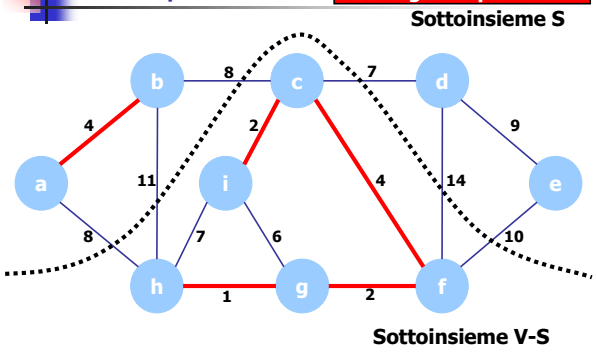
Esempio

Esempio di taglio



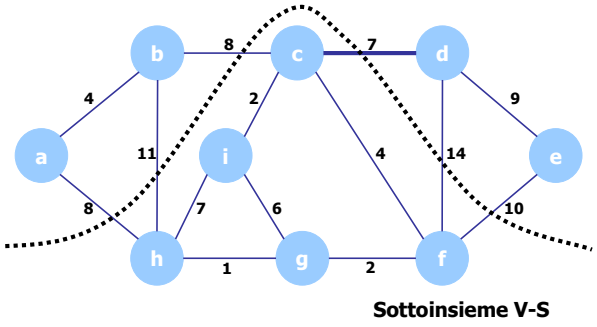
Esempio

In rosso un sottoinsieme di archi A. Il taglio rispetta A.



Esempio

L'arco (c,d) è leggero rispetto al taglio.



Teorema

Sia G=(V,E) un grafo non orientato e connesso con una funzione peso w a valori reali definita su E.

Sia A un sottoinsieme di E contenuto in un qualche albero di copertura minimo per G.

Sia (S,V-S) un qualunque taglio che rispetta A.

Sia (u,v) un arco leggero che attraversa (S,V-S).

Allora l'arco (u,v) è sicuro per A.

Corollario

Sia G=(V,E) un grafo non orientato e connesso con una funzione peso w a valori reali definita su E.
Sia A un sottoinsieme di E contenuto in un un albero di copertura minimo per G.
Sia C una componente connessa (ossia un albero) nella foresta G_A=(V,A).
Sia (u,v) un arco leggero che connette C ad un'altra componente in G_A.
Allora l'arco (u,v) è sicuro per A.

Sommario

- Introduzione
- Algoritmo generico
- Algoritmo di Kruskal
- Algoritmo di Prim.

Algoritmo di Kruskal

È riconducibile all'algoritmo generico, rispetto al quale utilizza un modo particolare per determinare un arco sicuro, basato sul precedente corollario.

Pseudo-codice

```
MST-Kruskal(G,w)
 1  A←∅
 2  for ogni vertice v∈V[G]
 3     do Make-Set(V)
 4  ordina gli archi di E per peso w non decrescente
 5  for ogni arco (u,v)∈E, in ordine di peso non decrescente
 6     do if Find-Set(u)≠Find-Set(v)
 7        then A←A∪{(u,v)}
 8             Union(u,v)
 9  return A
```


Pseudo-codice

```
MST-Kruskal(G,w)
 1  A←∅
 2  for ogni vertice v∈V[G]
 3     do Make-Set(V)
 4  ordina gli archi di E per peso w non decrescente
 5  for ogni arco (u,v)∈E, in ordine di peso non decrescente
 6     do if Find-Set(u)≠Find-Set(v)
 7        then A←A∪{(u,v)}
 8             Union(u,v)
 9  return A
```

Crea un insieme per ogni vertice in V

Pseudo-codice

```
MST-Kruskal(G,w)
 1  A←∅
 2  for ogni vertice v∈V[G]
 3     do Make-Set(V)
 4  ordina gli archi di E per peso w non decrescente
 5  for ogni arco (u,v)∈E, in ordine di peso non decrescente
 6     do if Find-Set(u)≠Find-Set(v)
 7        then A←A∪{(u,v)}
 8             Union(u,v)
 9  return A
```

Ritorna l'insieme a cui appartiene il vertice u

Ritorna l'insieme a cui appartiene il vertice v

Pseudo-codice

```
MST-Kruskal(G,w)
 1  A←∅
 2  for ogni vertice v∈V[G]
 3     do Make-Set(V)
 4  ordina gli archi di E per peso w non decrescente
 5  for ogni arco (u,v)∈E, in ordine di peso non decrescente
 6     do if Find-Set(u)≠Find-Set(v)
 7        then A←A∪{(u,v)}
 8             Union(u,v)
 9  return A
```

Verifica se l'arco (u,v) collega due vertici appartenenti allo stesso albero. Se così è, aggiungendo l'arco all'albero si introdurrebbe un ciclo.

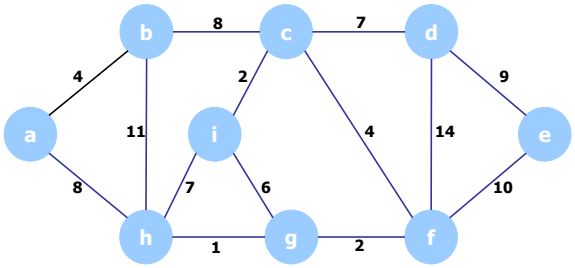
Pseudo-codice

```
MST-Kruskal(G,w)
 1  A←∅
 2  for ogni vertice v∈V[G]
 3     do Make-Set(V)
 4  ordina gli archi di E per peso w non decrescente
 5  for ogni arco (u,v)∈E, in ordine di peso non decrescente
 6     do if Find-Set(u)≠Find-Set(v)
 7        then A←A∪{(u,v)}
 8             Union(u,v)
 9  return A
```

Gli insiemi cui appartengono i vertici u e v vengono fusi in un unico insieme

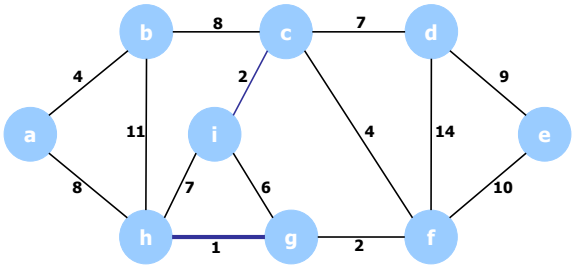
Esempio (0)

Inizializzazione: ogni vertice costituisce un insieme a sé.



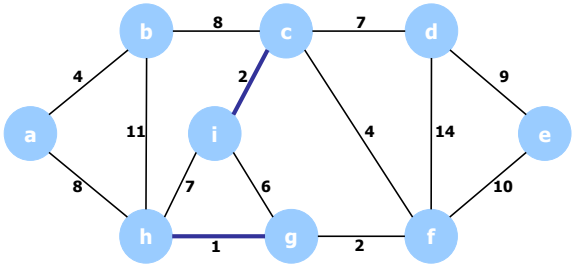
Esempio (1)

Si considera l'arco (h,g). Lo si inserisce in A.



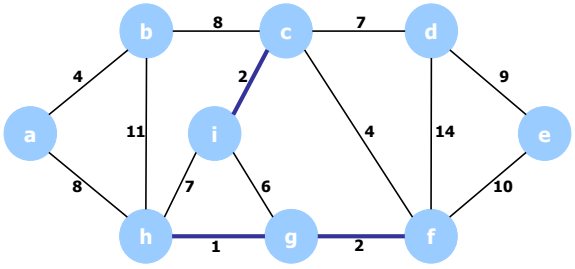
Esempio (2)

Si considera l'arco (i,c). Lo si inserisce in A.



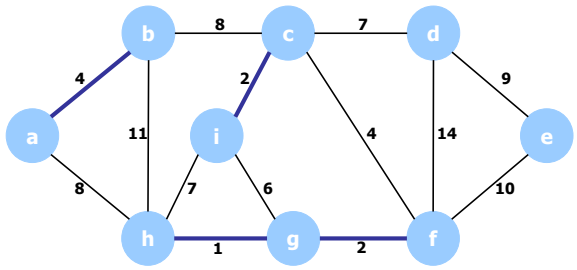
Esempio (3)

Si considera l'arco (g,f). Lo si inserisce in A.



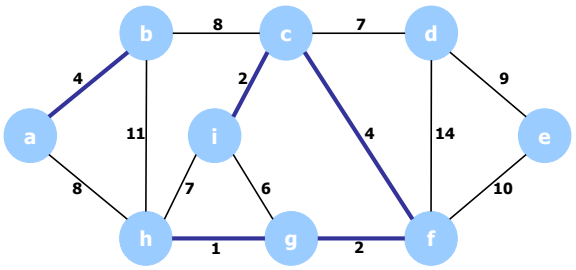
Esempio (4)

Si considera l'arco (a,b). Lo si inserisce in A.



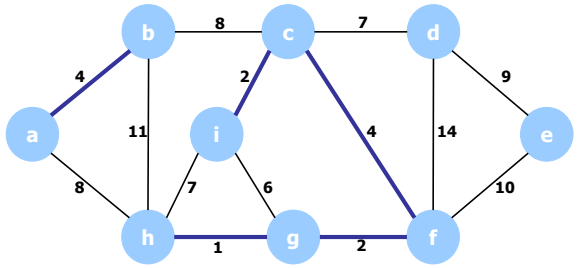
Esempio (5)

Si considera l'arco (c,f). Lo si inserisce in A.



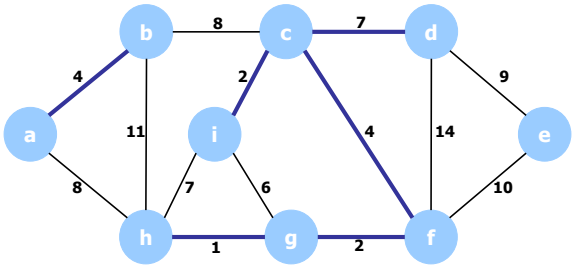
Esempio (6)

Si considera l'arco (i,g). Non lo si inserisce in A.



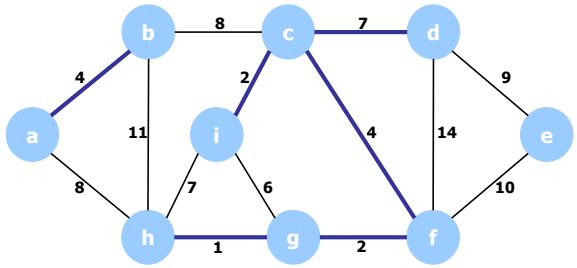
Esempio (7)

Si considera l'arco (c,d). Lo si inserisce in A.



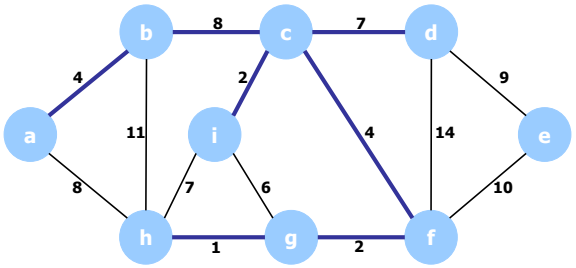
Esempio (8)

Si considera l'arco (i,h). Non lo si inserisce in A.



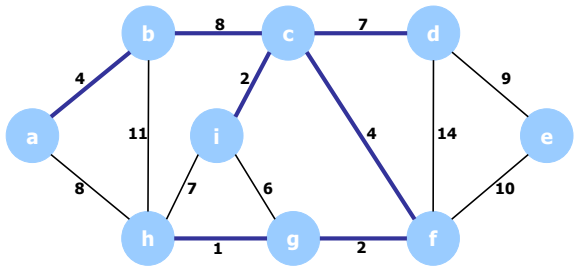
Esempio (9)

Si considera l'arco (b,c). Lo si inserisce in A.



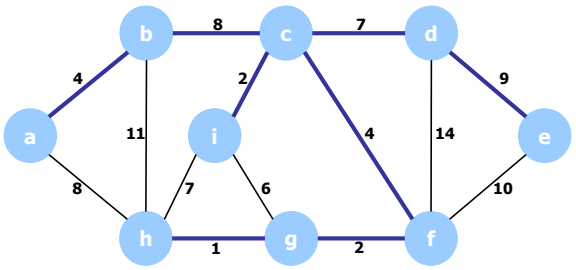
Esempio (10)

Si considera l'arco (a,h). Non lo si inserisce in A.



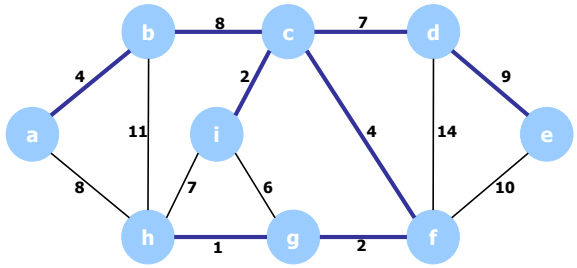
Esempio (11)

Si considera l'arco (d,e). Lo si inserisce in A.



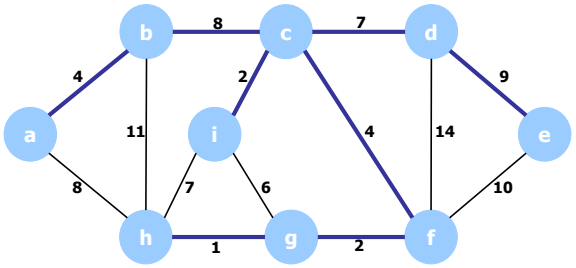
Esempio (12)

Si considera l'arco (e,f). Non lo si inserisce in A.



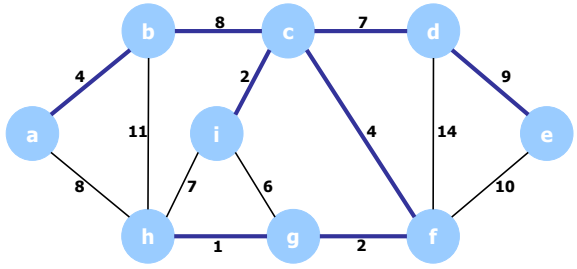
Esempio (13)

Si considera l'arco (b,h). Non lo si inserisce in A.



Esempio (14)

Si considera l'arco (d,f). Non lo si inserisce in A.



Complessità

```
MST-Kruskal(G,w)
1  A←∅
2  for ogni vertice v∈V[G]
3     do Make-Set(V)
4  ordina gli archi di E per peso w non decrescente
5  for ogni arco (u,v)∈E, in ordine di peso non decrescente
6     do if Find-Set(u)≠Find-Set(v)
7           then A←A∪{(u,v)}
8                Union(u,v)
9  return A
```

Righe 1,2,3 (inizializzazione): O(V)

Riga 4 (ordinamento): O(E lg E)

Ogni operazione sugli insiemi ha costo O(lg E)

Righe 5-8: il ciclo è ripetuto E volte

Complessità

Il tempo di esecuzione dell'algoritmo di Kruskal è O(E lg E).

Sommario

- Introduzione
- Algoritmo generico
- Algoritmo di Kruskal
- Algoritmo di Prim.

Algoritmo di Prim

Anche l'algoritmo di Prim segue lo schema proposto dall'algoritmo generico, utilizzando un meccanismo per la scelta degli archi basato sul teorema introdotto in precedenza.

L'algoritmo parte da un nodo radice r ed espande ad ogni passo l'albero di copertura minimo A, sino a che questo non copre tutti i vertici.

Ad ogni passo viene aggiunto all'albero un arco leggero che collega un vertice in A ad un vertice in V-A.

Per la scelta dell'arco leggero si usa una coda prioritaria.

Coda prioritaria

Ad ogni istante contiene tutti i vertici non appartenenti all'albero A.

La posizione di ciascun vertice v nella coda dipende dal valore di un campo chiave key[v], corrispondente al minimo tra i pesi degli archi che collegano v ad un qualunque vertice in A. Se v non è collegato a nessun vertice in A, allora key[v]=∞.

Pseudo-codice

```
MST-Prim(G,w,r)
 1  Q←V[G]
 2  for ogni u∈Q
 3     do key[u]←∞
 4  key[r]←0
 5  π[r]←NIL
 6  while Q≠∅
 7     do u←Extract-Min(Q)
 8        for ogni v∈Adj[u]
 9           do if v∈ Q e w(u,v)<key[v]
10                 then  π[v]←u
11                       key[v]←w(u,v)
```


Pseudo-codice

Inizializzazione della coda: tutti i vertici sono inseriti. Per ogni vertice v≠r key[v]=∞; key[r]=0.

```
MST-Prim(G,w,r)
 1  Q←V[G]
 2  for ogni u∈Q
 3     do key[u]←∞
 4  key[r]←0
 5  π[r]←NIL
 6  while Q≠∅
 7     do u←Extract-Min(Q)
 8        for ogni v∈Adj[u]
 9           do if v∈ Q e w(u,v)<key[v]
10                 then  π[v]←u
11                       key[v]←w(u,v)
```

Pseudo-codice

```
MST-Prim(G,w,r)
 1  Q←V[G]
 2  for ogni u∈Q
 3      do key[u]←∞
 4  key[r]←0
 5  π[r]←NIL
 6  while Q≠∅
 7      do u←Extract-Min(Q)
 8         for ogni v∈Adj[u]
 9             do if v∈ Q e w(u,v)<key[v]
10                   then  π[v]←u
11                         key[v]←w(u,v)
```

Il vettore π contiene l'indice del vertice padre di ciascun vertice in A. Per i vertici fuori da A, π è indefinito.

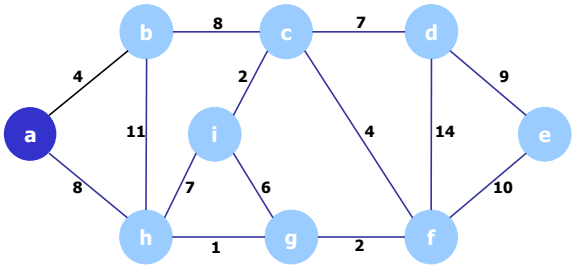
Pseudo-codice

```
MST-Prim(G,w,r)
 1  Q←V[G]
 2  for ogni u∈Q
 3      do key[u]←∞
 4  key[r]←0
 5  π[r]←NIL
 6  while Q≠∅
 7      do u←Extract-Min(Q)
 8         for ogni v∈Adj[u]
 9             do if v∈ Q e w(u,v)<key[v]
10                   then  π[v]←u
11                         key[v]←w(u,v)
```

Estrae un vertice dalla coda e lo aggiunge all'albero. Aggiorna key e π.

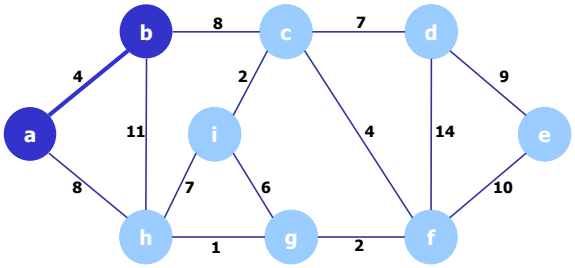
Esempio (0)

Il vertice a è la radice.



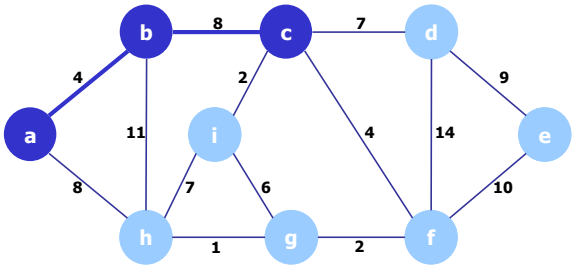
Esempio (1)

Il vertice b viene aggiunto all'albero.



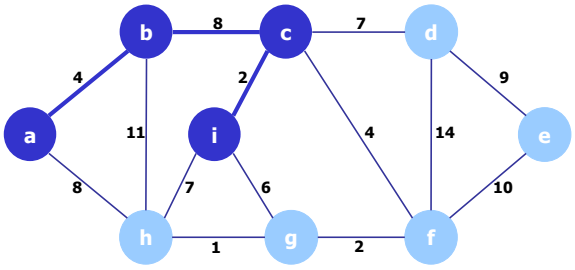
Esempio (2)

Il vertice c viene aggiunto all'albero. Anche il vertice h poteva essere scelto.



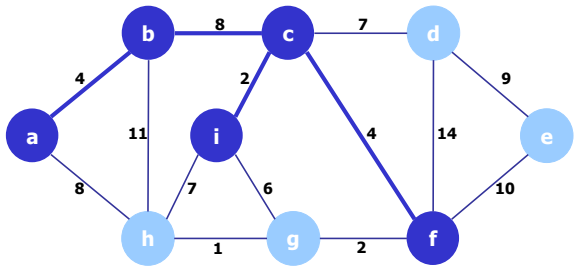
Esempio (3)

Il vertice i viene aggiunto all'albero.



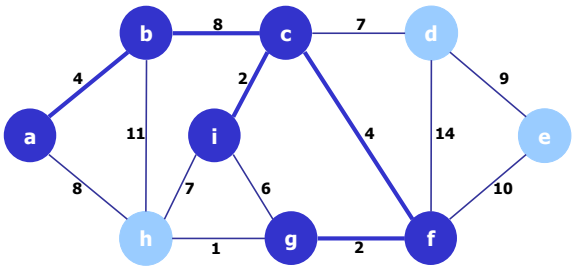
Esempio (4)

Il vertice f viene aggiunto all'albero.



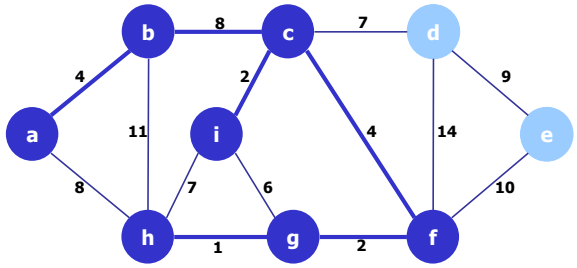
Esempio (5)

Il vertice g viene aggiunto all'albero.



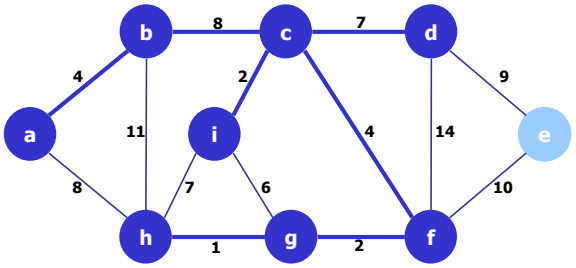
Esempio (6)

Il vertice h viene aggiunto all'albero.



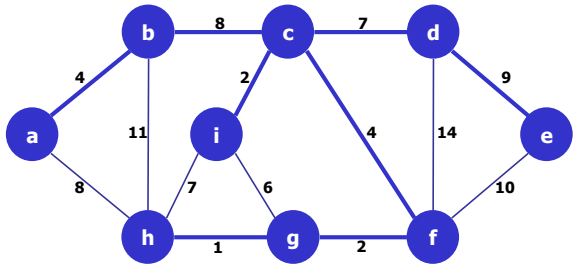
Esempio (7)

Il vertice d viene aggiunto all'albero.



Esempio (8)

Il vertice e viene aggiunto all'albero.



Complessità

Se per realizzare la coda con priorità si usa uno heap l'inizializzazione si esegue con Build-Heap: O(V)

```
MST-Prim(G,w,r)
 1  Q←V[G]
 2  for ogni u∈Q
 3      do key[u]←∞
 4  key[r]←0
 5  π[r]←NIL
 6  while Q≠∅
 7      do u←Extract-Min(Q)
 8          for ogni v∈Adj[u]
 9              do if v∈ Q e w(u,v)<key[v]
10                    then  π[v]←u
11                          key[v]←w(u,v)
```

Complessità

```
MST-Prim(G,w,r)
 1  Q←V[G]
 2  for ogni u∈Q
 3      do key[u]←∞
 4  key[r]←0
 5  π[r]←NIL
 6  while Q≠∅
 7      do u←Extract-Min(Q)
 8         for ogni v∈Adj[u]
 9             do if v∈ Q e w(u,v)<key[v]
10                   then  π[v]←u
11                         key[v]←w(u,v)
```

Il ciclo viene ripetuto V volte. Extract-Min ha complessità O(lg V).

Complessità

```
MST-Prim(G,w,r)
 1  Q←V[G]
 2  for ogni u∈Q
 3      do key[u]←∞
 4  key[r]←0
 5  π[r]←NIL
 6  while Q≠∅
 7      do u←Extract-Min(Q)
 8         for ogni v∈Adj[u]
 9             do if v∈ Q e w(u,v)<key[v]
10                   then  π[v]←u
11                         key[v]←w(u,v)
```

Il ciclo viene ripetuto O(E) volte.

Richiede un riaggiustamento dello heap, con complessità O(lg V).

Complessità

Il tempo di esecuzione dell'algoritmo di Prim è $O(V \lg V + E \lg V) = O(E \lg V)$.

Usando uno heap di Fibonacci la complessità scende a $O(E + V \lg V)$.