Alberi di copertura minimi

Alberi di copertura minimi

- Dato un grafo pesato G = (V,E,W),
- si richiede di trovare un albero T = (V,E',W'), E'⊆E,
- tale che la somma dei pesi associati agli archi di T sia minima.

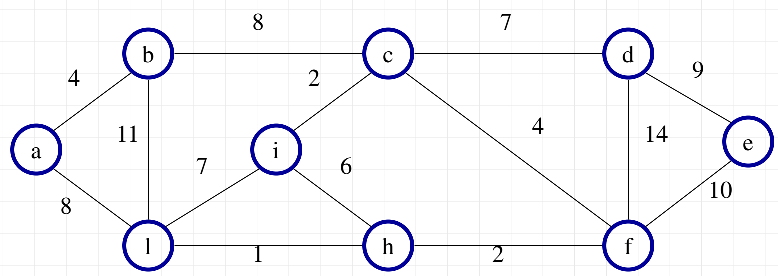
L'albero T è detto albero di copertura minimo (*minimum spanning tree*, MST) di G.

Due algoritmi *greedy* per calcolare un MST: Kruskal e Prim.

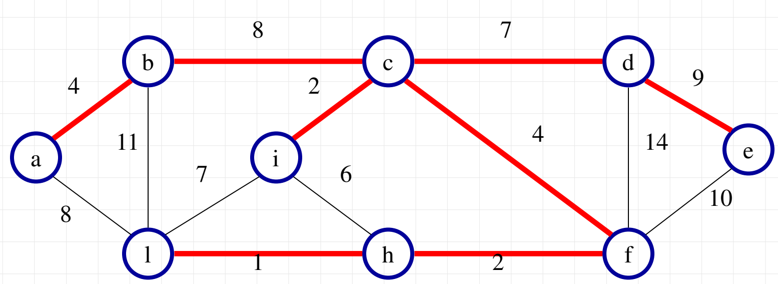
Entrambi basati su uno stesso algoritmo generale che costruisce l'insieme A degli archi dell'MST partendo dall'insieme vuoto e aggiungendo di volta in volta un arco a tale che A ∪ a sia sottoinsieme degli archi di un MST.

Gli algoritmi di Kruskal e di Prim differiscono per il modo in cui viene cercato l'arco da aggiungere.

MST: esempio



MST: esempio

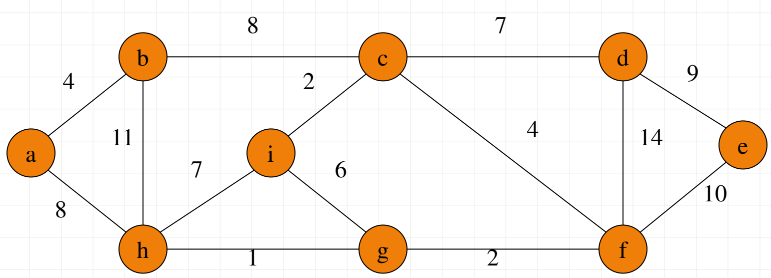


MST - Algoritmo di Kruskal

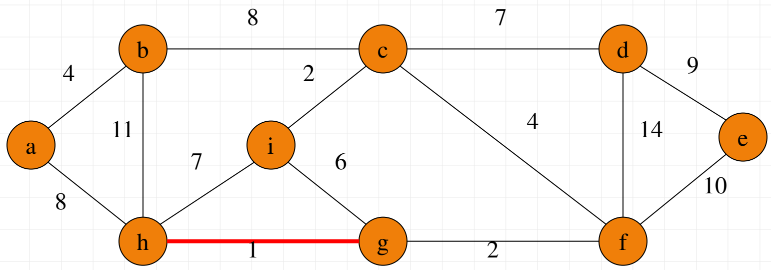
```
MST-Kruskal(G,w)
  A = ∅
  foreach vertice v ∈ V[G] do
     Make-Set(v)
  ordina gli archi di E[G] per pesi non decrescenti
  foreach (u,v) ∈ E[G] in ordine di peso non decr. do
     if Find-Set(u) ≠ Find-Set(v)
     then A = A ∪ {(u,v)}
          Union(u,v)
  return A
```

Make-Set(v): crea un insieme con unico membro v
Find-Set(v): restituisce il rappresentante dell'insieme contenente v
Union(u,v): unisce i due insiemi che contengono u e v

Kruskal: esempio

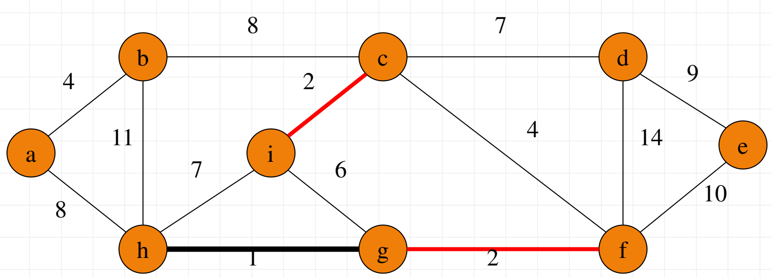


Kruskal: esempio



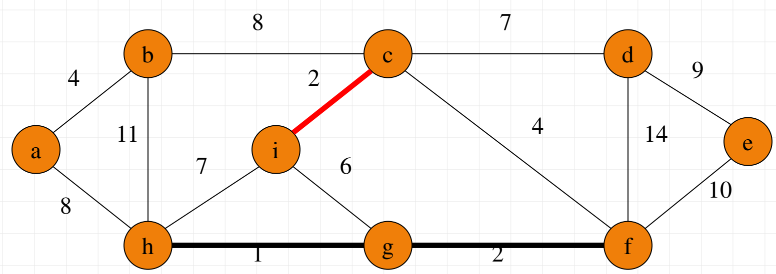
7

Kruskal: esempio

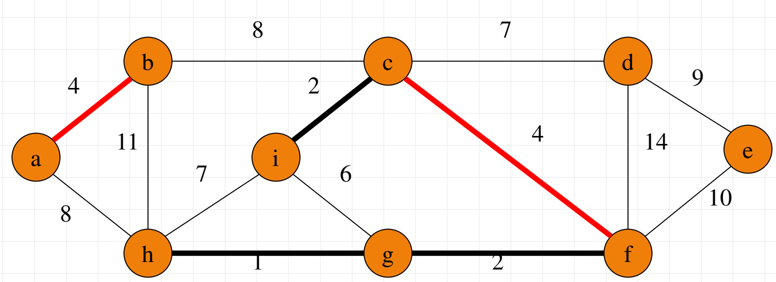


8

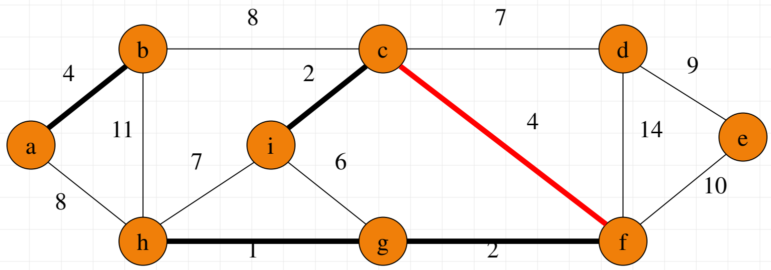
Kruskal: esempio



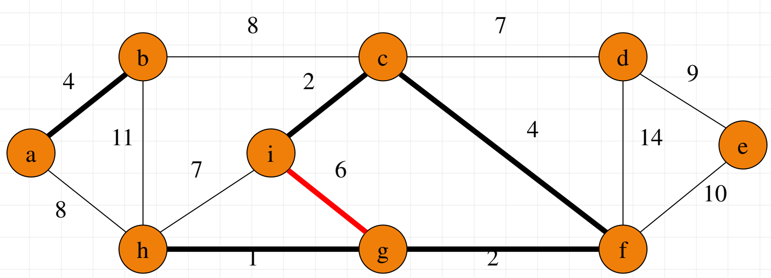
Kruskal: esempio



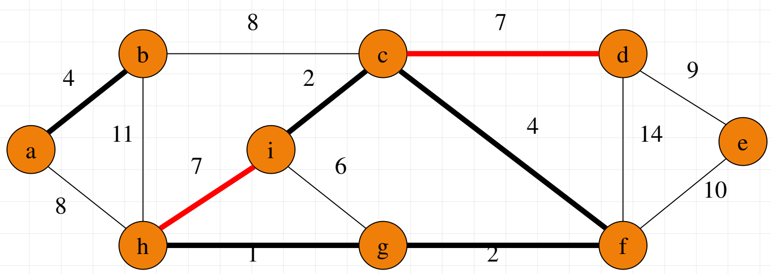
Kruskal: esempio



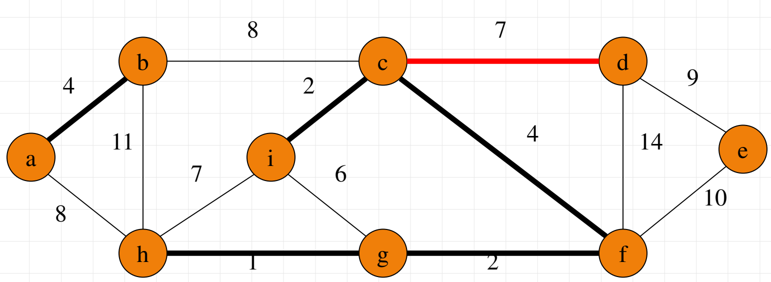
Kruskal: esempio



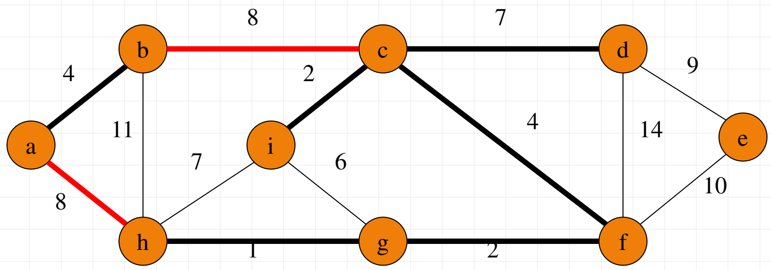
Kruskal: esempio



Kruskal: esempio

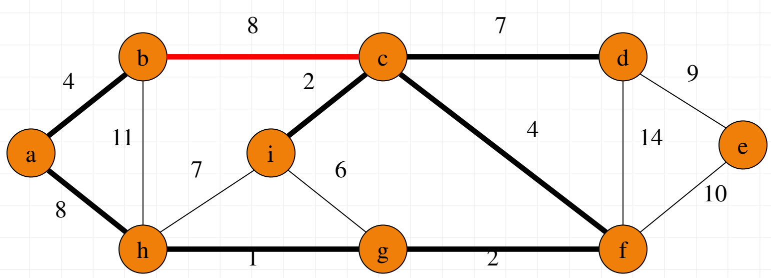


Kruskal: esempio



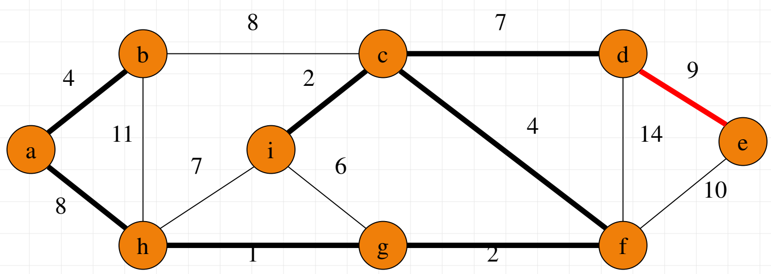
15

Kruskal: esempio

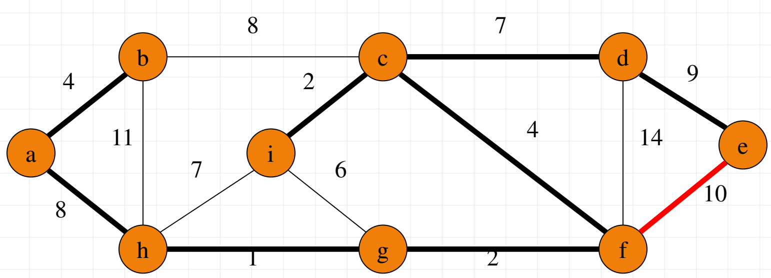


16

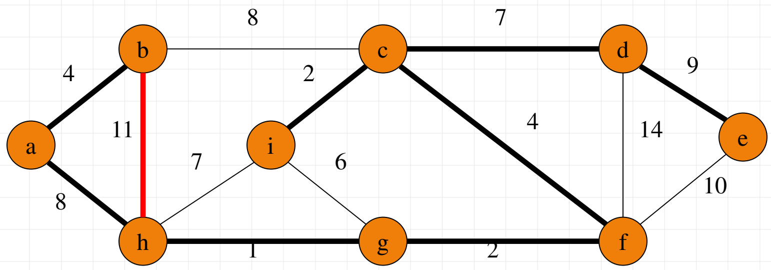
Kruskal: esempio



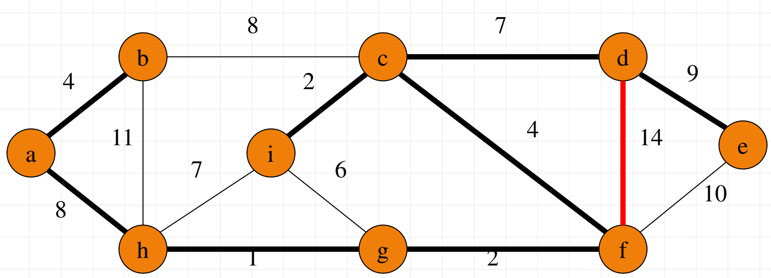
Kruskal: esempio



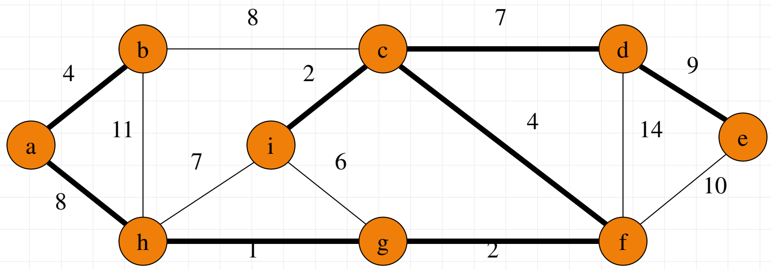
Kruskal: esempio



Kruskal: esempio



Kruskal: esempio



Kruskal: correttezza

Teorema

Siano

G(V,E,W) il grafo dato in input

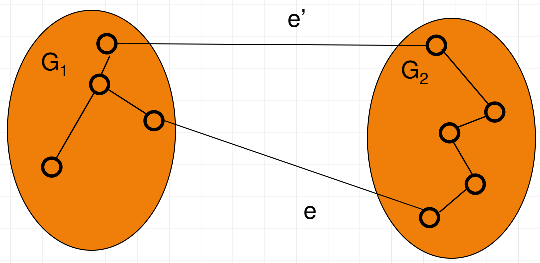
G'(V,E',W') sottografo di un qualche MST di G, contenente tutti i vertici e alcuni archi di G
(NON è un albero di copertura. Restano vertici isolati)

$G_1(V_1,E_1,W_1) \cup G_2(V_2,E_2,W_2)$ - partizione di G' in due componenti non connesse (*taglio* di G)

$e \in E$ arco di peso minimo che connette G_1 e G_2

allora anche $G'(V,E' \cup \{e\}, W' \cup \{w_e\})$ è un sottografo di qualche MST

Kruskal: correttezza



Sia T' un MST contenente e'. Se w(e')≥w(e) e si sostituisce e' con e si ottiene un altro albero di copertura T con w(T') ≥ w(T).

Quindi T sarebbe un albero di copertura di costo inferiore a T': o w(e') = w(e) o T' non era un MST.

Kruskal: complessitá

- Inizializzazione: O(V)
- Ordinamento archi: O(E lg E)
- Operazioni nella foresta di insiemi disgiunti: O(E)
- Tempo complessivamente richiesto per la costruzione: O(E α(E,V)) o anche O(E lg* V), v. up trees (*E operazioni find-set, union e V make-set*).

T(V,E) = O(V) + O(E log E) + O(E log* V) = O(E log E)

Dato che log|E| = O(log|V|^2) = O(log|V|), la complessità è anche O(|E| log|V|).

MST: algoritmo di Prim

Nell'algoritmo di Prim gli archi dell'insieme in costruzione formano sempre un unico albero, A.

L'algoritmo costruisce l'albero di connessione minimo partendo da un vertice prescelto come radice ed estendendolo finché non connette tutti i vertici.

Usa una coda di priorità Q in cui memorizza i vertici non ancora raggiunti dall'albero in costruzione.

MST: algoritmo di Prim

Strutture dati:

vertici NON in A: code di priorità, chiave key.

- key[v]: min peso degli archi che connettono v ad un nodo dell'albero.
- π[v]: predecessore di v nell'albero A.

Alla fine A = $\{(v, \pi[v]): v \in V - \{r\}\}$ è l'MST.

MST: algoritmo di Prim

```
MST-Prim(G, w, r)
Q = V[G]
foreach u ∈ V do key[u] = ∞
key[r] = 0
π[r] = nil
while Q ≠ ∅ do
   u = Extract-Min(Q)
   foreach v ∈ Adj(u) do
      if v ∈ Q and w(u,v) < key[v]
      then π[v] = u
           key[v] = w(u,v)
```


MST: algoritmo di Prim

Strutture dati:

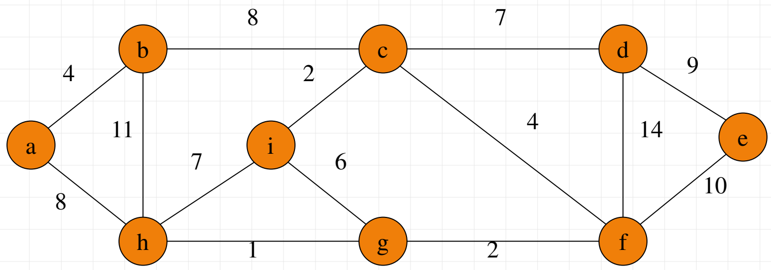
vertici NON in A: code di priorità, chiave key.

key[v]: min peso degli archi che connettono v ad un nodo dell'albero.

π[v]: predecessore di v nell'albero A.

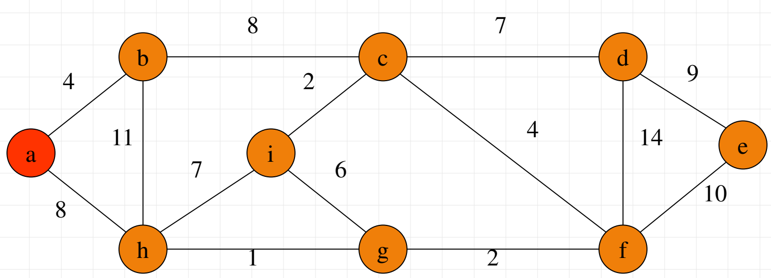
Alla fine A = {(v, π[v]): v ∈ V – {r}} è l'MST.

Prim: esempio



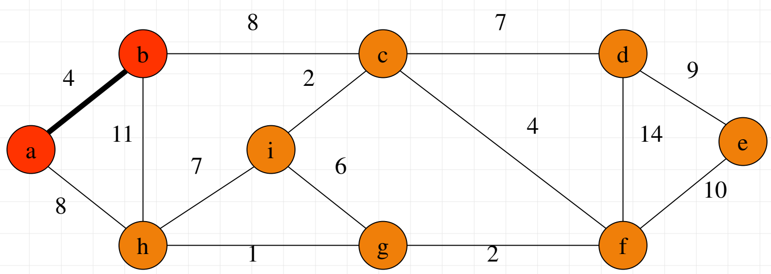
29

Prim: esempio

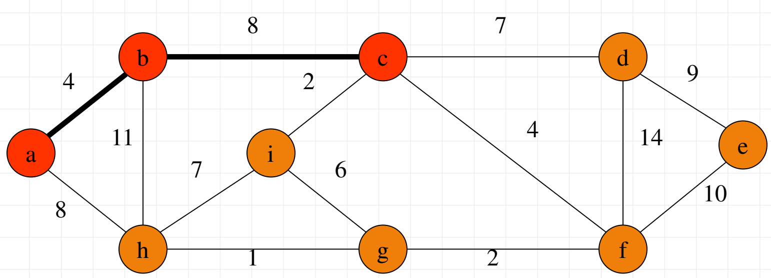


30

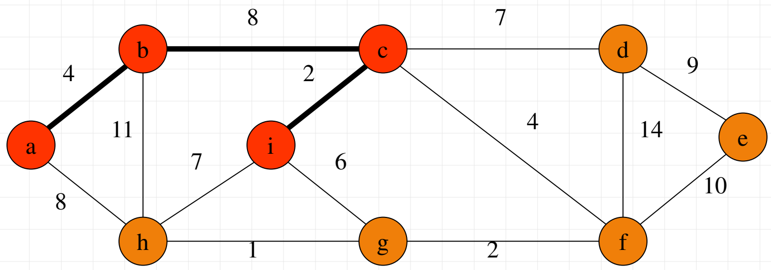
Prim: esempio



Prim: esempio

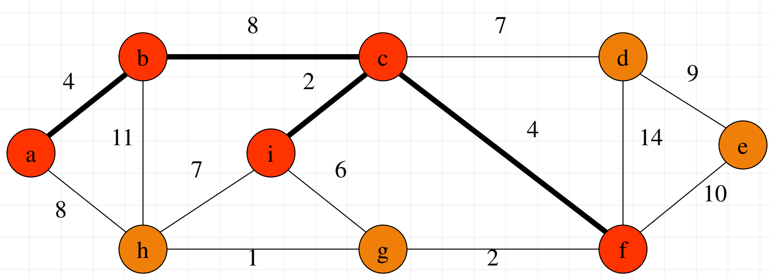


Prim: esempio



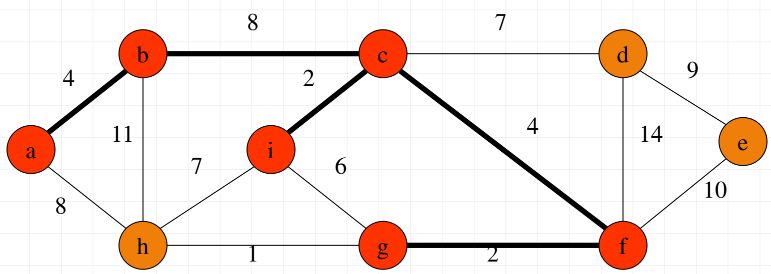
33

Prim: esempio

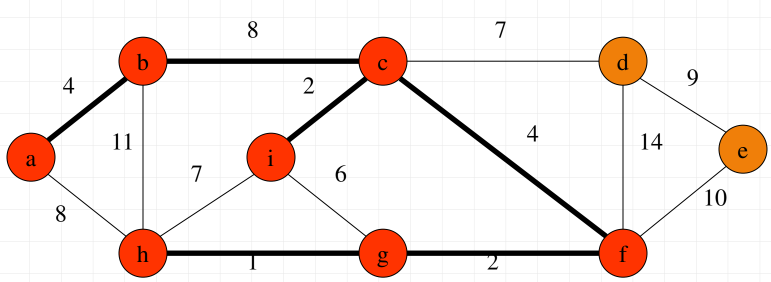


34

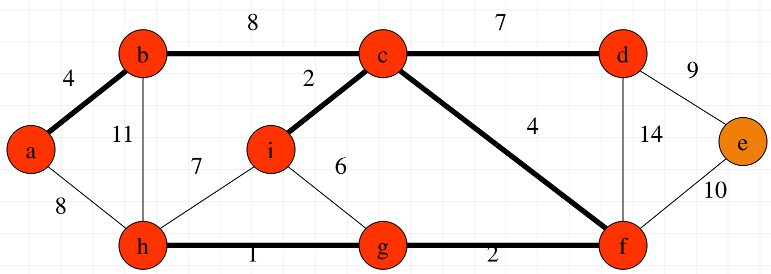
Prim: esempio



Prim: esempio



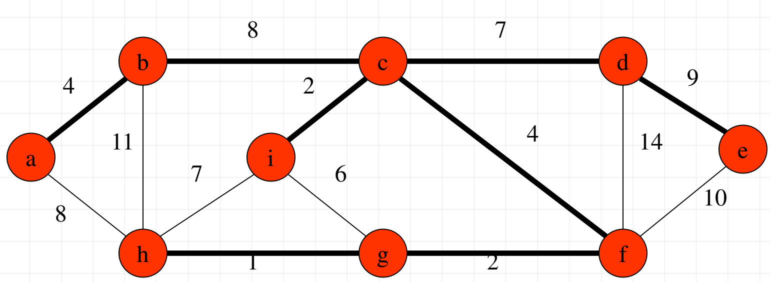
Prim: esempio



Vittorio Maniezzo - Universita di Bologna 37

37

Prim: esempio



Vittorio Maniezzo - Universita di Bologna 38

38

Prim: complessitá

Dipende dall'implementazione delle code di priorità.

Heap binaria:

Init: $O(V)$

Ciclo: V volte, ExtractMin $O(\lg V) \longrightarrow O(V \lg V)$

For: $O(E)$ volte, DecreaseKey $O(\lg V) \longrightarrow O(E \lg V)$

Complessivamente: $O(V + V \lg V + E \lg V) = O(E \lg V)$

Heap di Fibonacci

Init: $O(V)$

Ciclo: V volte, ExtractMin $O(\lg V) \longrightarrow O(V \lg V)$

For: $O(E)$ volte, DecreaseKey $O(1) \longrightarrow O(E)$

Complessivamente: $O(V + V \lg V + E) = O(E + V \lg V)$