Algoritmos e Estruturas de Dados

Árvore Geradora Mínima

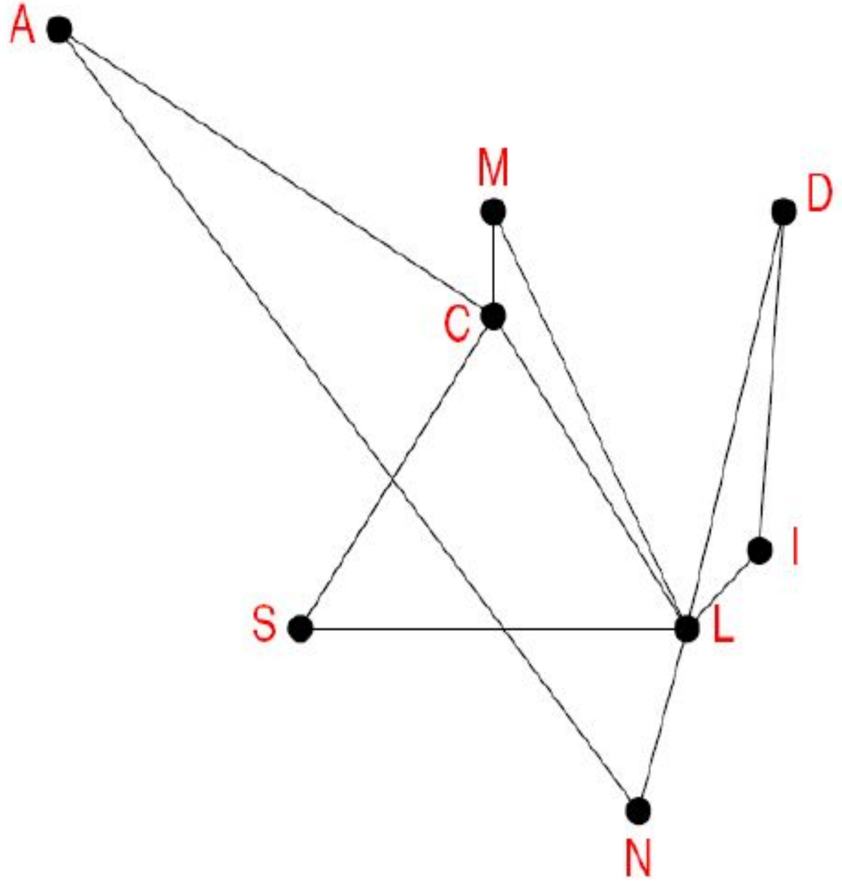
Slides baseados em:

- **ZIVIANI, N. Projetos de Algoritmos - com implementações em Java e C++. Thomson Learning, 2007. Cap 7.**
- **CORMEN, H.T.; LEISERSON, C.E.; RIVEST, R.L. Introduction to Algorithms, MIT Press, McGraw-Hill, 1999.**
- **Slides Humberto C. Brandão de Oliveira**

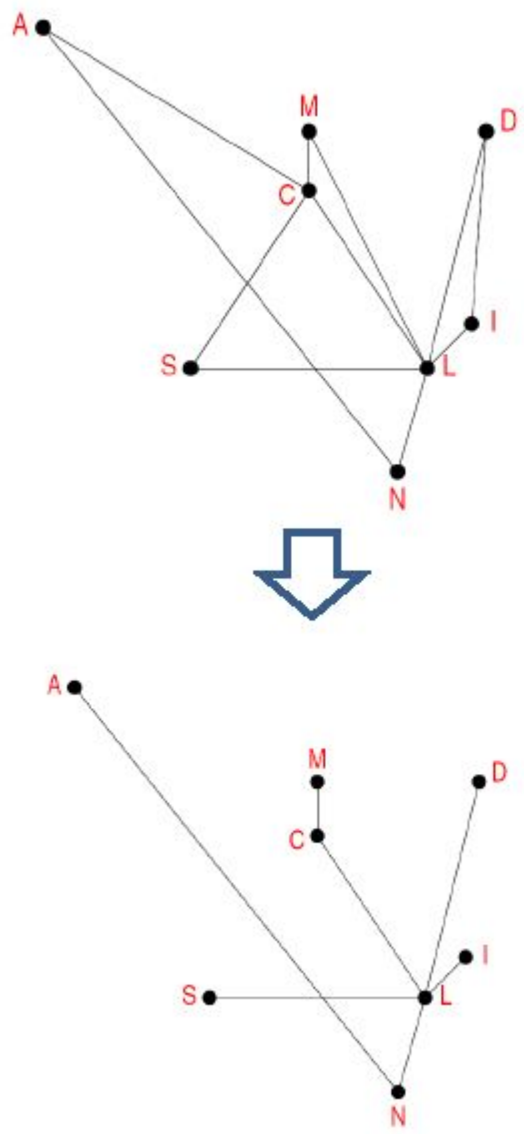
Profa. Karina Valdivia Delgado
EACH-USP

Motivação

- Suponha que uma companhia aérea recebeu permissão para voar nas rotas da figura.
- Por questões de economia, a empresa não irá operar em todas as vias.
- A empresa precisa atender a toda a demanda aérea do país (afinal, os passageiros podem fazer conexões)



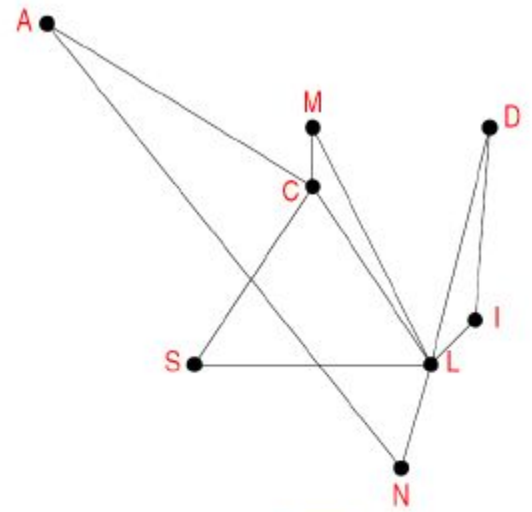
Motivação

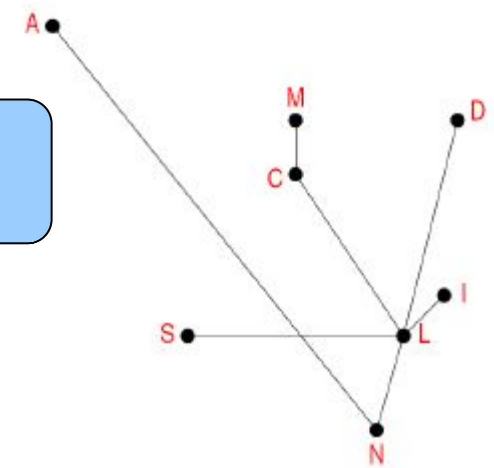


- Uma forma de atender toda a demanda é interconectando todas as cidades.
- Sem considerar pesos (distância), este conjunto de rotas é mínimo?

Motivação

$G=(V,A)$





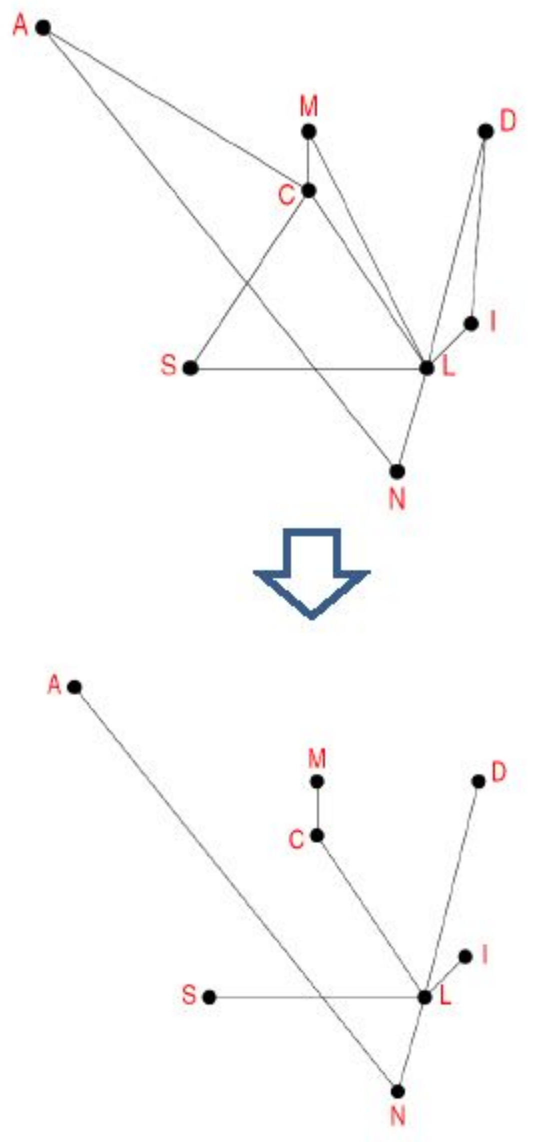
$G'=(V,T)$

$|T| = |V| - 1$

- Uma forma de atender toda a demanda é interconectando todas as cidades.
- Sem considerar pesos (distância), este conjunto de rotas é mínimo? Sim

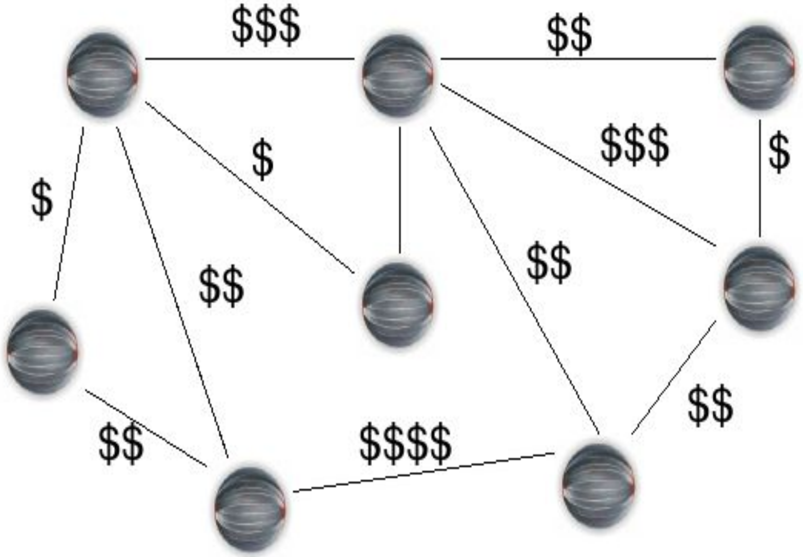
Motivação

- Se consideramos os pesos (distâncias), como encontrar o conjunto de rotas com peso mínimo?



Aplicações

- Transporte aéreo
- Transporte terrestre
- Redes de computadores
- Redes elétricas
- Circuitos integrados



O problema da Árvore Geradora Mínima

- O problema da Árvore Geradora Mínima consiste em encontrar um subconjunto acíclico $T \subseteq A$, que conecta todos os vértices e cujo peso total é mínimo.

$$G = (V, A) \longrightarrow G' = (V, T)$$

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

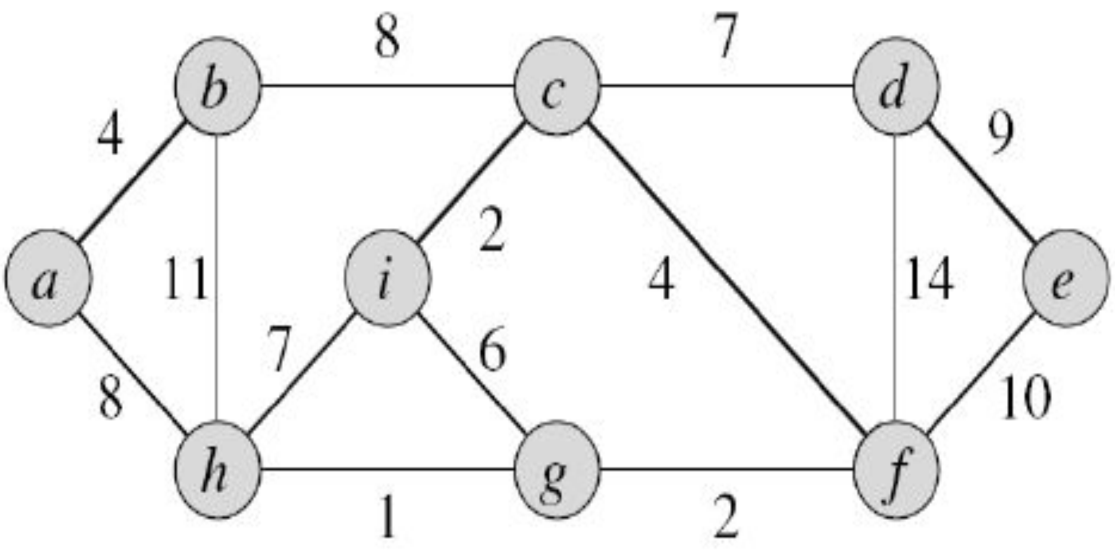
Em que para cada aresta (u,v) pertencente a A, temos um peso w(u,v) especificando o custo para interconectar (u,v).

Árvore Geradora Mínima (AGM)

- Uma vez que T é acíclico e conecta todos os vértices, ele deve formar uma árvore, que é chamada de:
  - Árvore geradora, ou
  - Árvore espalhada, ou
  - Árvore de extensão;
- O problema de determinar a árvore de menor custo é conhecido como:
  - Problema da Árvore Geradora Mínima, ou
  - Problema da Árvore Espalhada Mínima.
  - Problema da Árvore de Extensão Mínima.

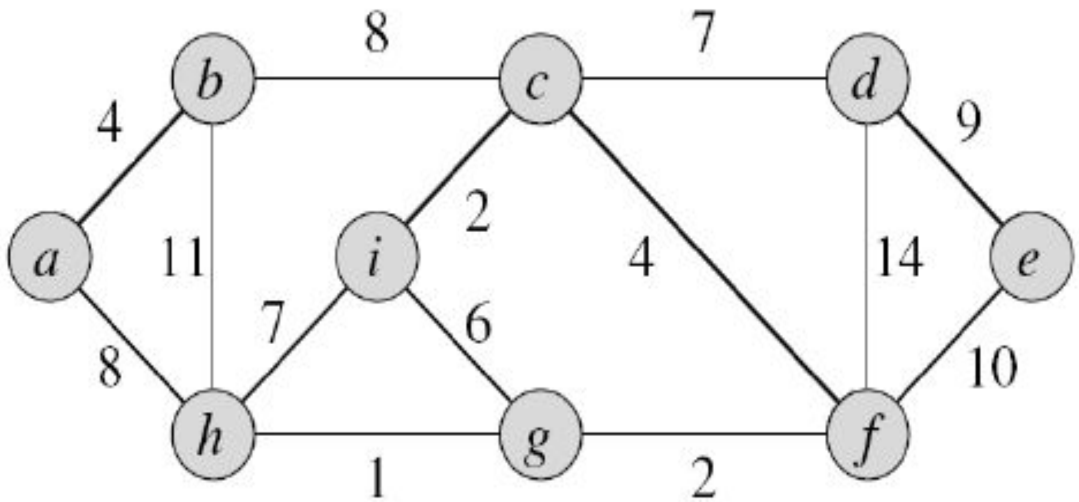
Árvore Geradora Mínima: Exemplo

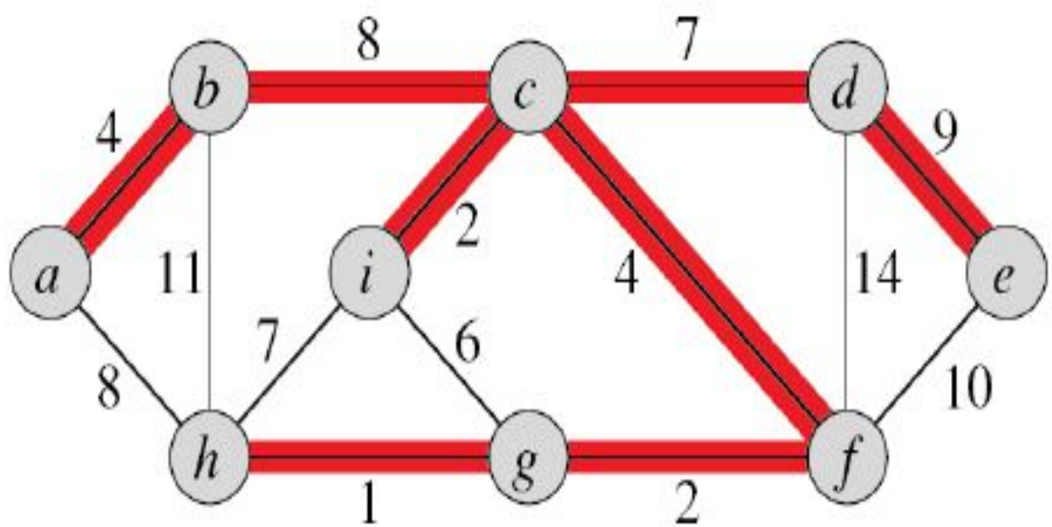
- Qual a árvore geradora mínima?
- Qual o peso total?
- É única?



Árvore Geradora Mínima: Exemplo

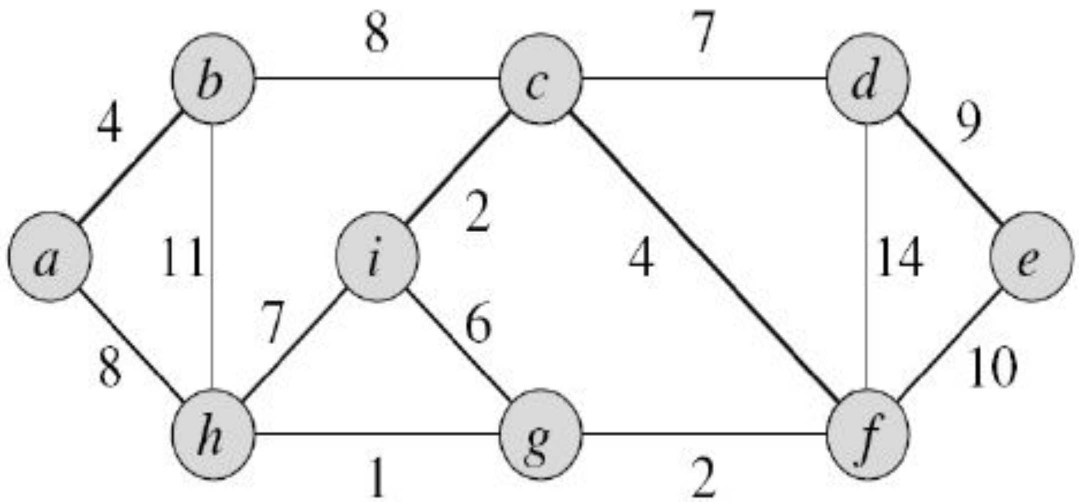
- Qual a árvore geradora mínima?
- Qual o peso total?
- É única?



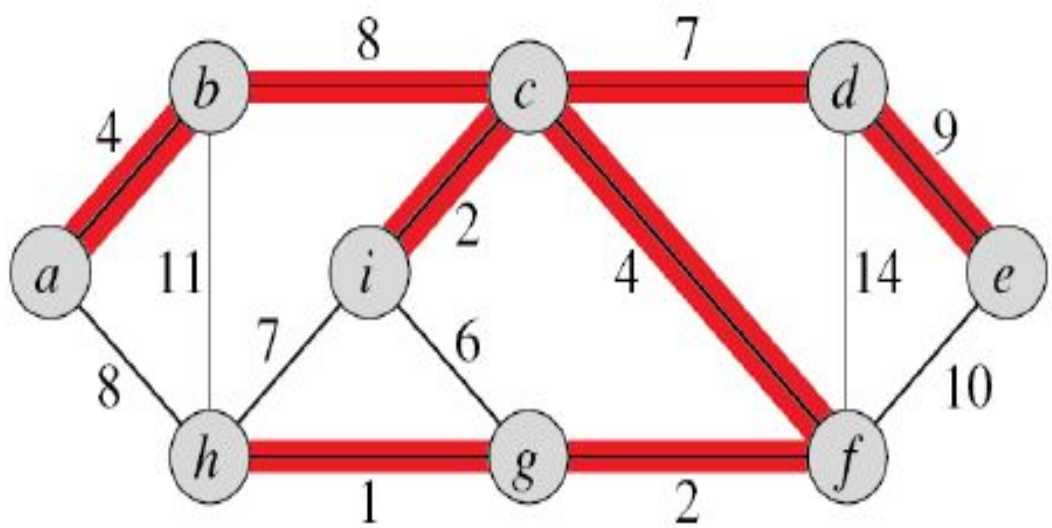


Árvore Geradora Mínima: Exemplo

- Qual a árvore geradora mínima?
- Qual o peso total?
- É única?



O peso total da árvore é 37.
A aresta (b,c) pode ser substituída pela aresta (a,h)



Algoritmos

- Dois algoritmos gulosos:
  - –Algoritmo de Kruskal;
  - –Algoritmo de Prim;
- Na estratégia gulosa é feita a melhor escolha a cada passo (MELHOR ESCOLHA LOCAL), mesmo que tal escolha não nos leve a uma solução ótima ao final da execução.
- Esses dois algoritmos, mesmo usando a estratégia gulosa, encontram a solução ótima.

Algoritmos

- Veremos primeiro um algoritmo genérico, que constrói a árvore geradora mínima adicionando uma aresta segura de cada vez.
- Seja X um subconjunto de arestas de uma árvore geradora mínima. Uma aresta (u,v) é segura em relação a X se $(u,v) \notin X$ e $X \cup \{(u,v)\}$ também é um subconjunto de uma árvore geradora mínima.

Algoritmo Genérico

GENERIC-MST (G, w)

X = { }
while X não forma uma árvore geradora mínima
    encontrar uma aresta (u, v) que seja segura para X
    X = X $\cup$ {(u, v)}
return X

Algoritmo Genérico

GENERIC-MST (G, w)

X = { }
while X não forma uma árvore geradora mínima
    encontrar uma aresta (u, v) que seja segura para X
    X = X ∪ {(u, v)}
return X

Ponto chave

Algoritmo Genérico

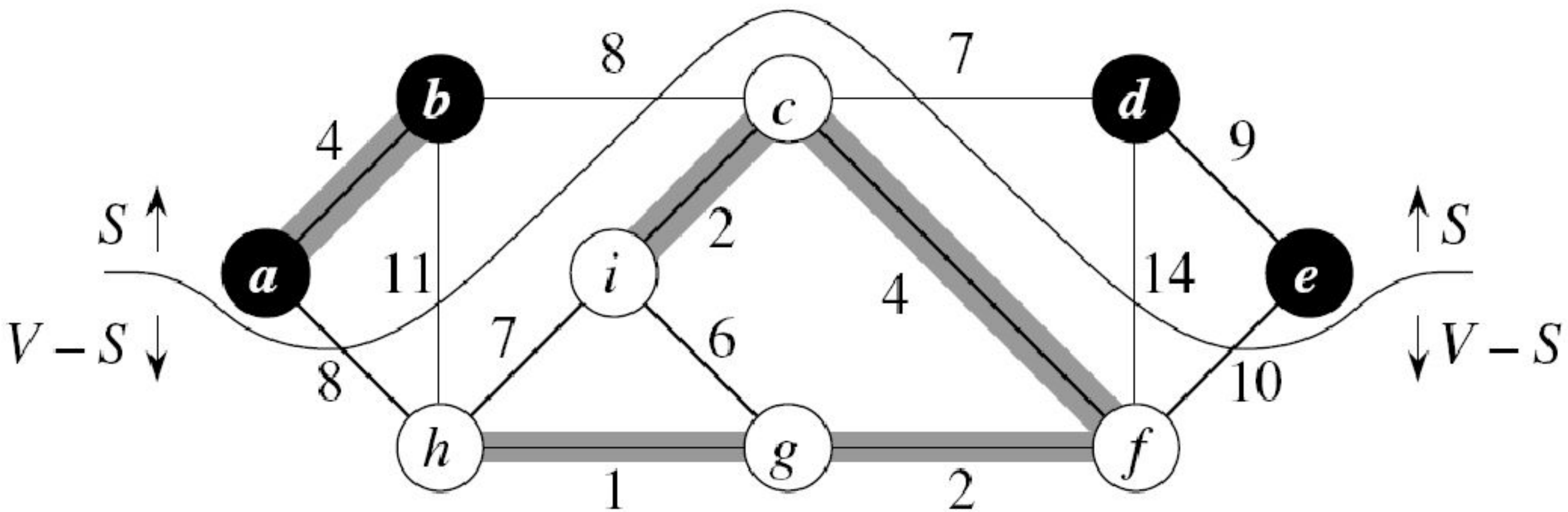
- Precisamos definir o conceito de corte antes de fornecer uma regra para reconhecer arestas seguras
- Corte é uma partição do conjunto de vértices. Dado o grafo G = (V , A) o corte é:

$$(S , V - S)$$

Algoritmo Genérico

- Corte é uma partição do conjunto de vértices. Dado o grafo G = (V , A) o corte é:

$$(S , V - S)$$



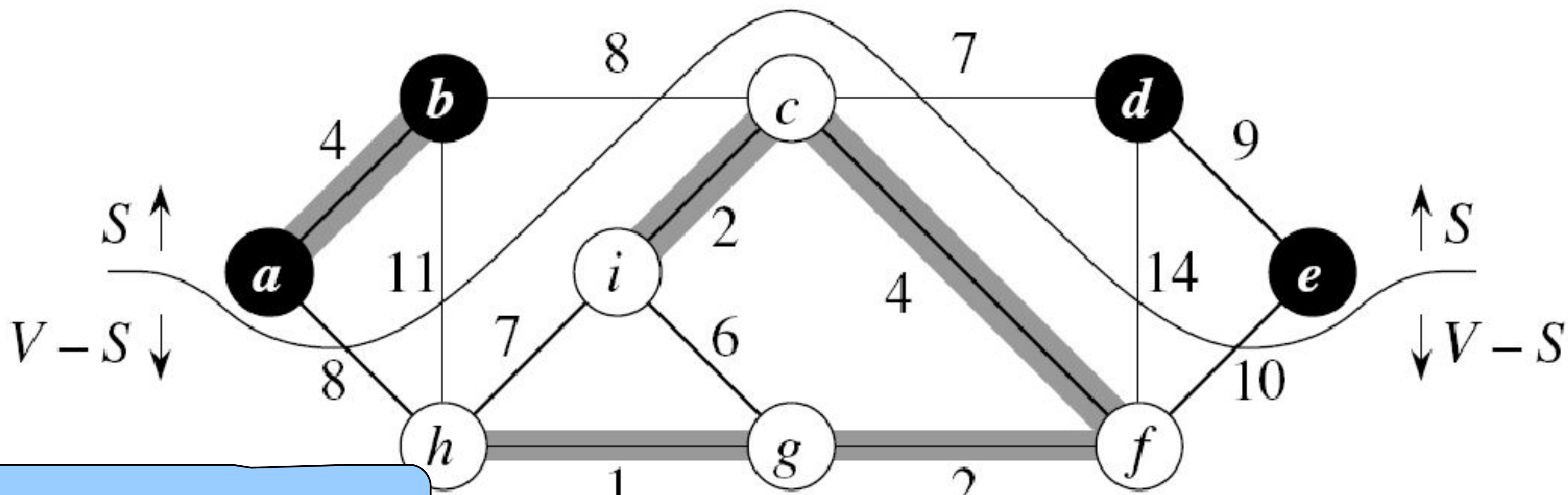
Algoritmo Genérico

- Corte é uma partição do conjunto de Dado o grafo G = (V , A) o corte

  S={a, b, d, e}

  V-S={h, i, g, c, f}

Nós pretos



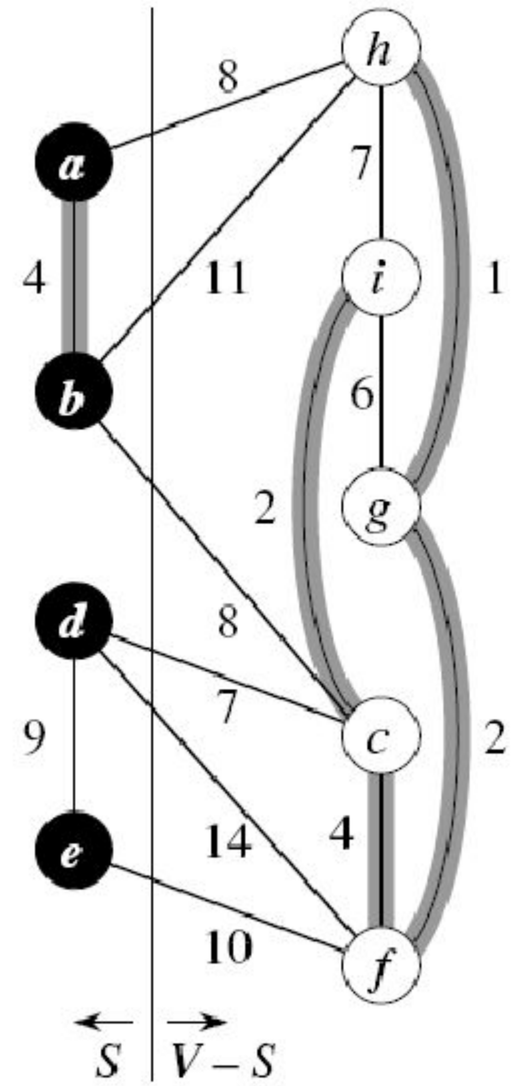
Nós brancos

Algoritmo Genérico

- Outra forma de visualizar o mesmo corte:

S={a, b, d, e}

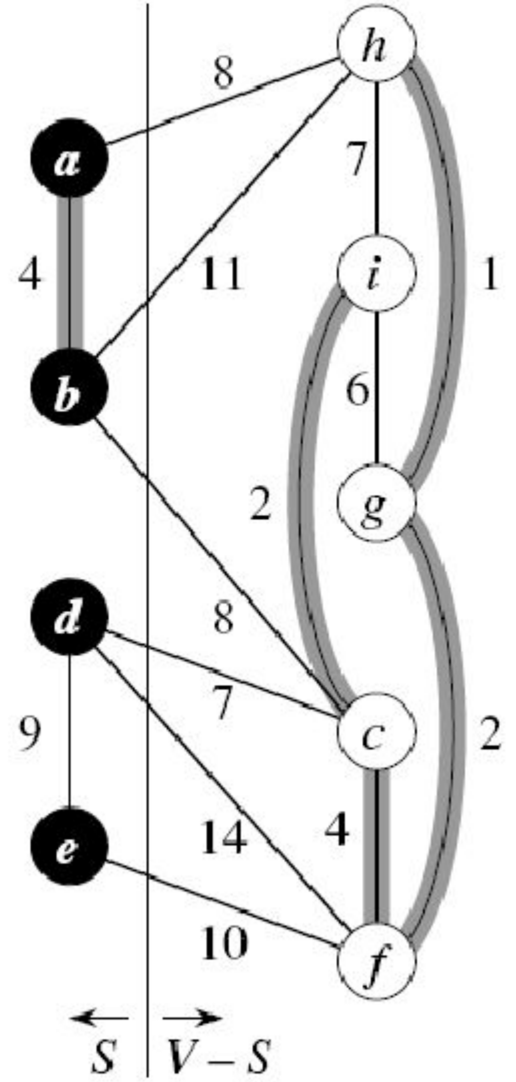
V-S={h, i, g, c, f}



Algoritmo Genérico

- Definição: Aresta que cruza o corte

Dizemos que a aresta (u,v) cruza o corte se conecta um vértices que está em S, e outro que está em V-S.

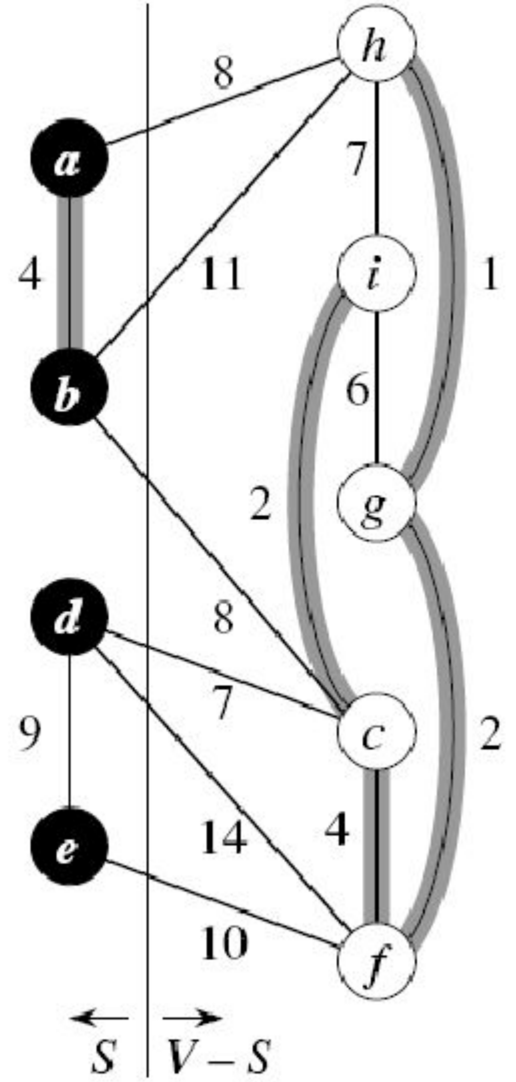


Algoritmo Genérico

- Definição: Corte respeita X

Dizemos que um corte respeita o conjunto X se nenhuma aresta de X cruza o corte

O conjunto X de arestas está sombreado

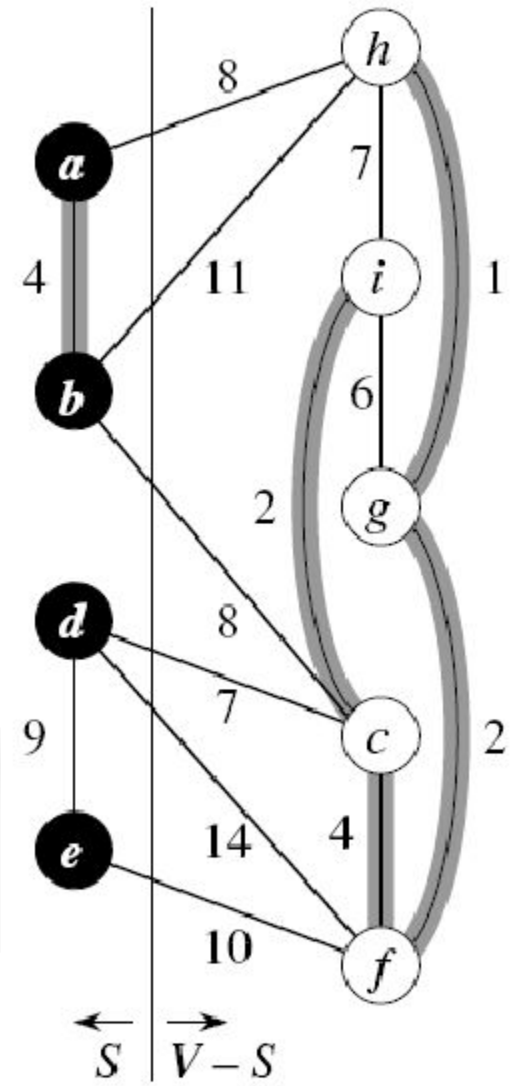


Algoritmo Genérico

- Definição: Corte respeita X

Dizemos que um corte respeita o conjunto X se nenhuma aresta de X cruza o corte

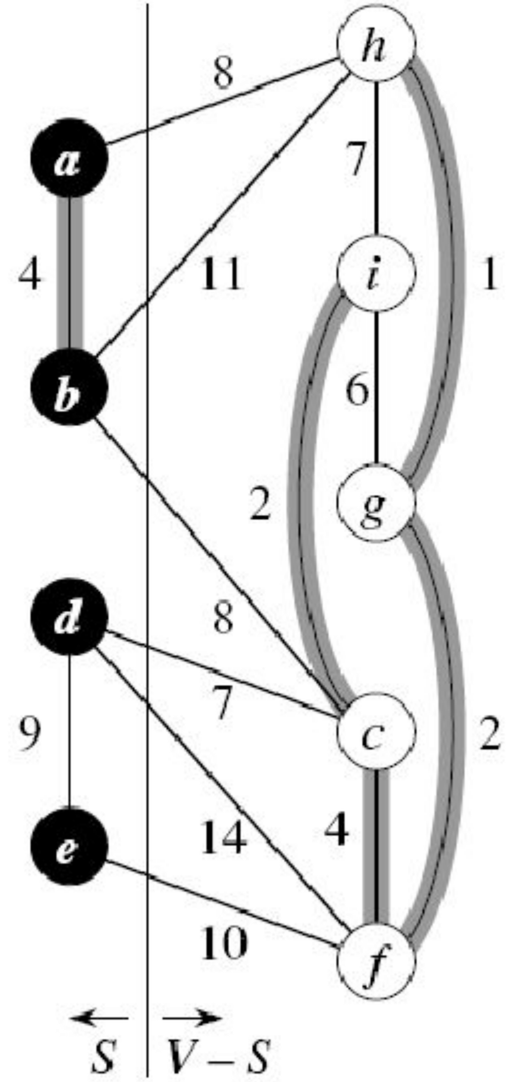
Observe que o corte S={a, b, d, e}
V-S={h, i, g, c, f}
respeita X



Algoritmo Genérico

- Definição: Aresta leve

Dizemos que uma aresta é uma aresta leve cruzando o corte se o seu peso é o menor, quando comparado as outras arestas que cruzam o corte.

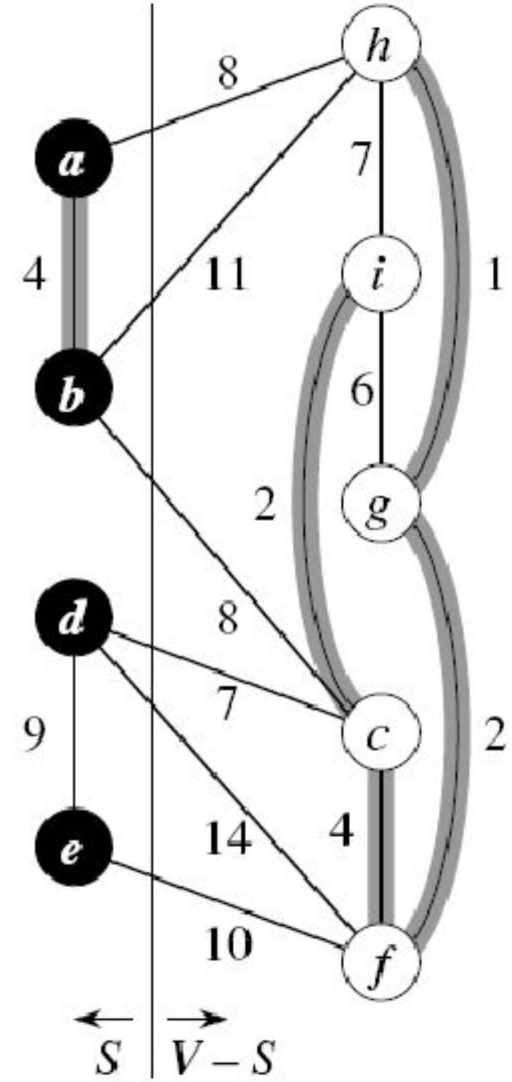


Algoritmo Genérico

- Definição: Aresta leve

Dizemos que uma aresta é uma aresta leve cruzando o corte se o seu peso é o menor, quando comparado as outras arestas que cruzam o corte.

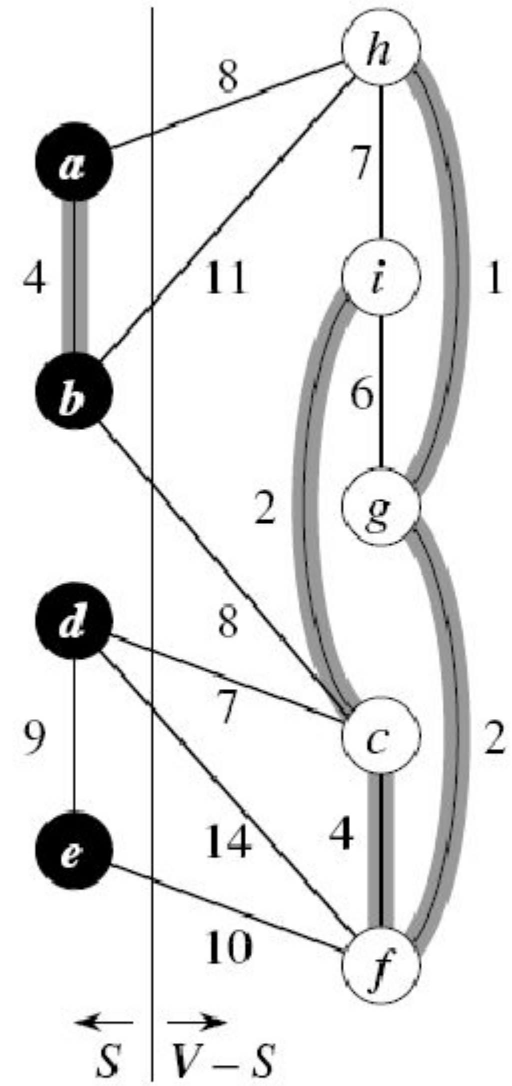
A aresta **(d,c)** é a única aresta leve que cruza o corte



Algoritmo Genérico

- **Teorema: Arestas seguras**

  Seja (S, V-S) qualquer corte de G que respeita X e seja (u,v) uma aresta leve cruzando (S,V-S), então a aresta **(u,v) é segura** para X

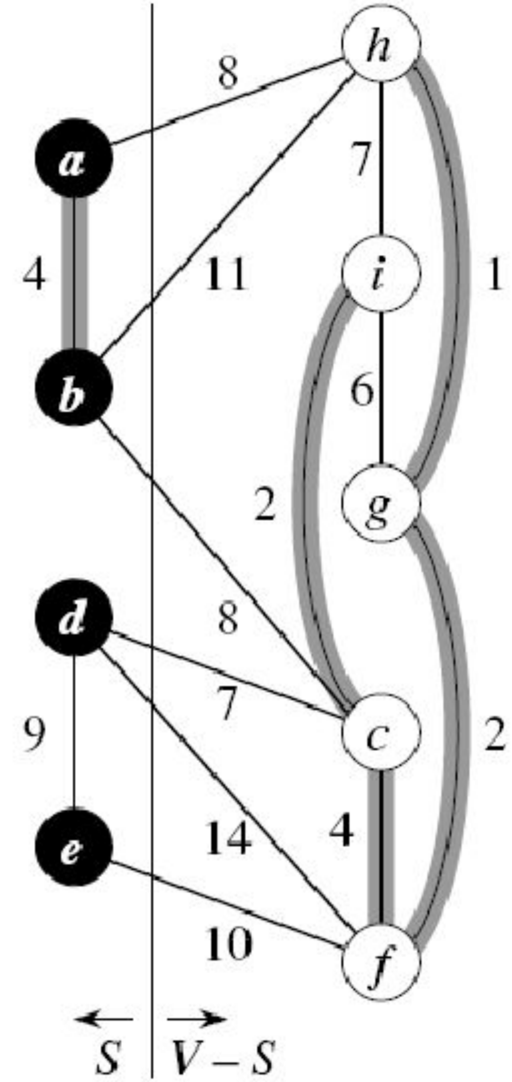


Algoritmo Genérico

- **Teorema: Arestas seguras**

  Seja (S, V-S) qualquer corte de G que respeita X e seja (u,v) uma aresta leve cruzando (S,V-S), então a aresta **(u,v) é segura para X.**

A aresta **(d,c)** é uma aresta segura para X

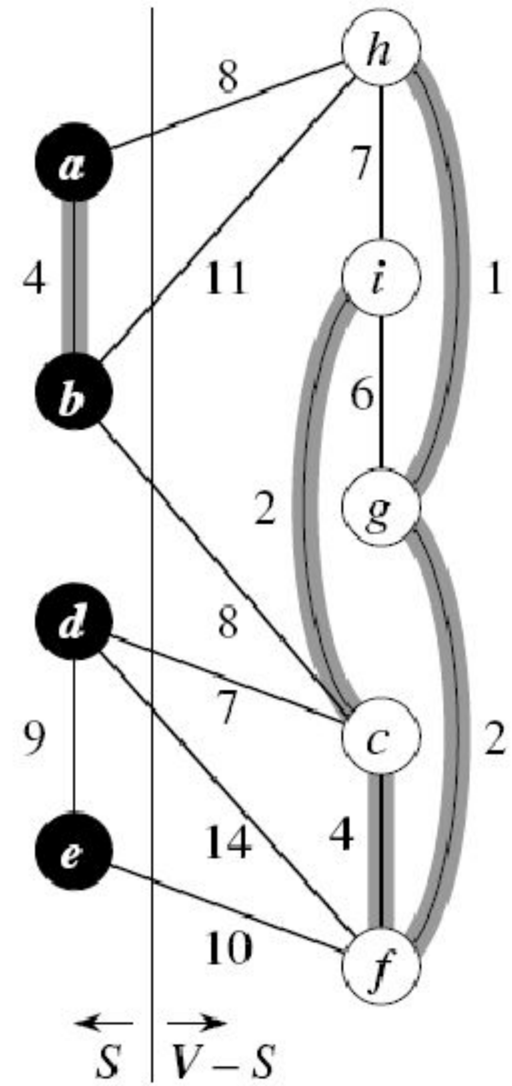


Algoritmo Genérico

- **Teorema: Arestas seguras**

  Seja (S, V-S) qualquer corte de G que respeita X e seja (u,v) uma aresta leve cruzando (S,V-S), então a aresta **(u,v) é segura para X.**

Como fazer o corte?



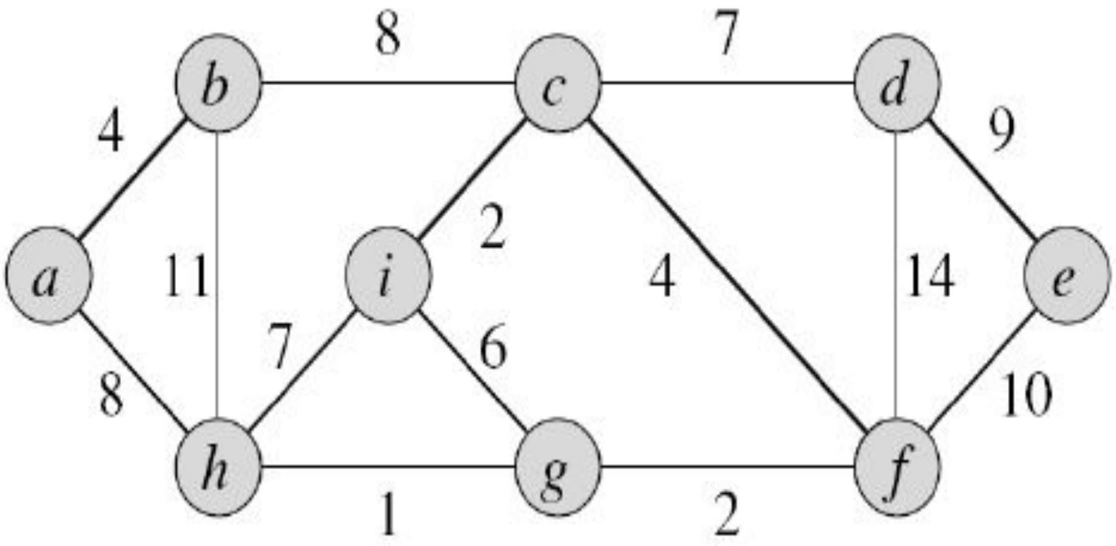
Algoritmo de Kruskal

- Se baseia diretamente no algoritmo genérico.
- Gera uma floresta, antes de gerar a Árvore Geradora Mínima.
- Depois da última iteração do algoritmo a floresta é apenas uma árvore.

Algoritmo de Kruskal

- A aresta segura adicionada a X é sempre uma aresta de peso mínimo no grafo que conecta dois componentes distintos (duas árvores distintas na floresta).

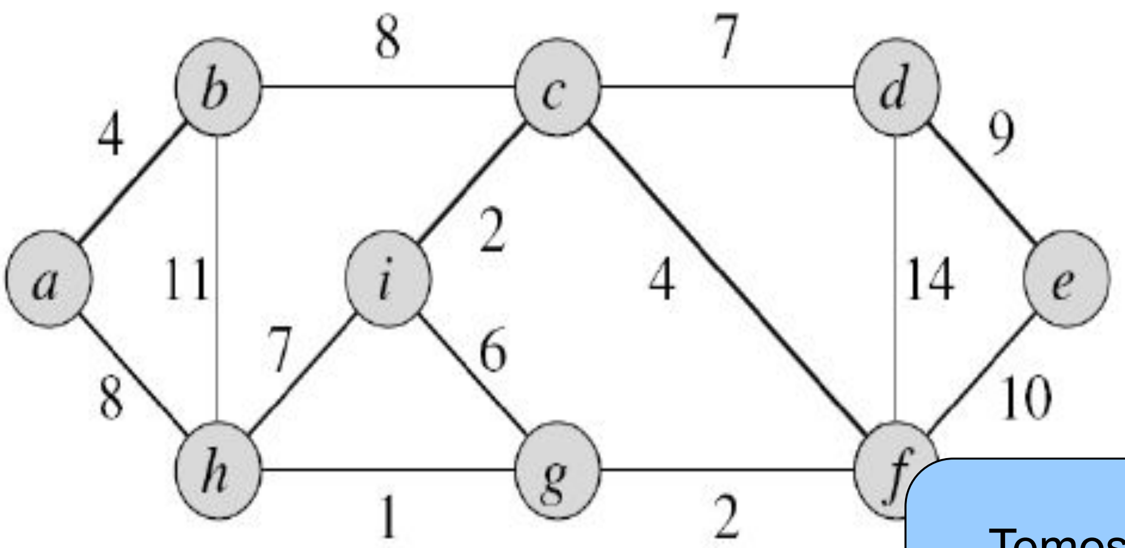
Algoritmo de Kruskal



|V| árvores/conjuntos são criadas

{{a},{b},{c},{d },{e},{f},{g},{h},{i}}

Algoritmo de Kruskal

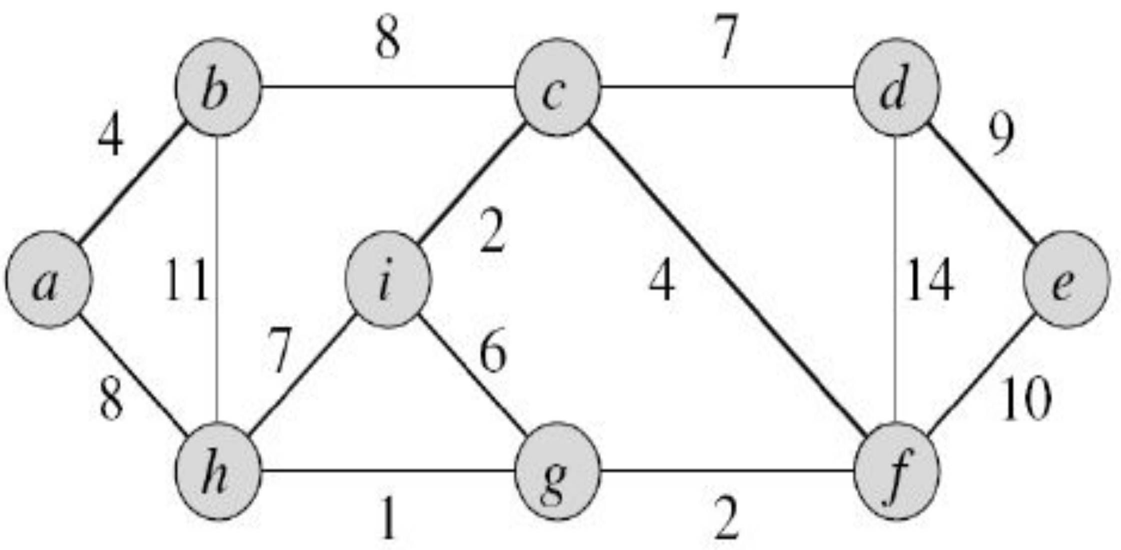


Temos 9 árvores com apenas um vértice cada uma

|V| árvores/conjuntos são criadas

{{a},{b},{c},{d },{e},{f},{g},{h},{i}}

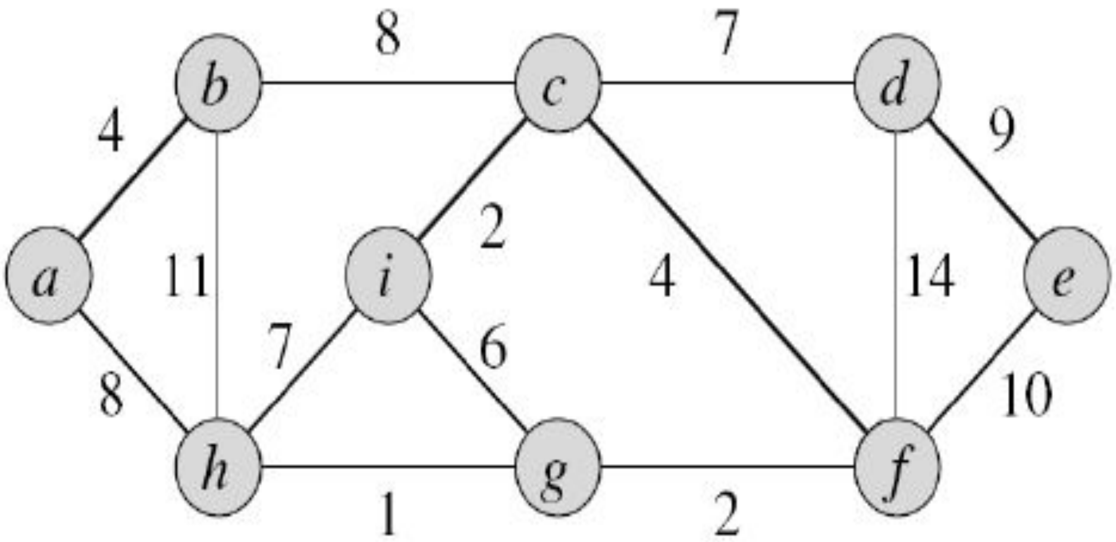
Algoritmo de Kruskal



O conjunto de arestas é ordenado pelo peso

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Floresta:

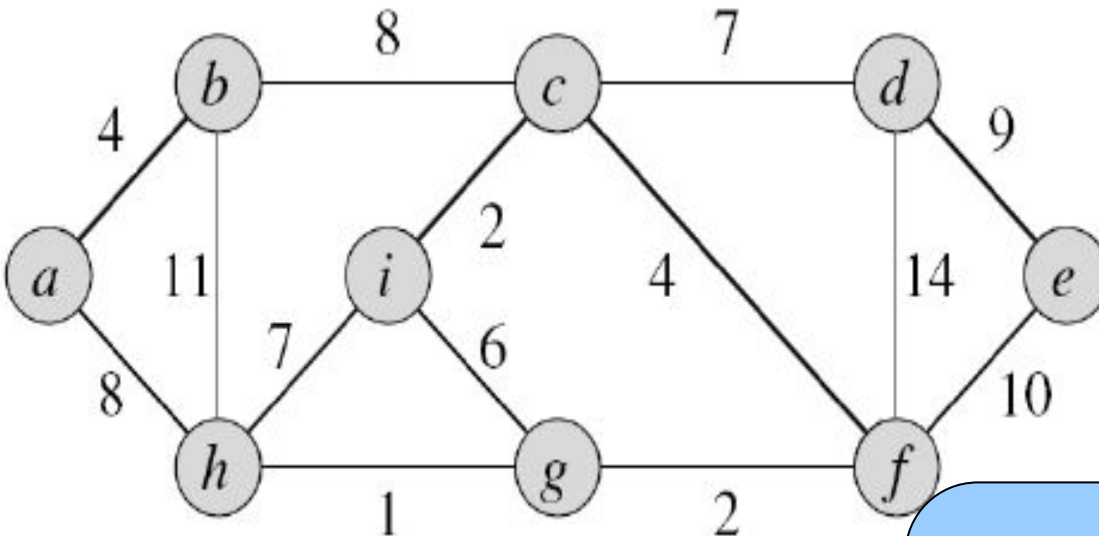
{{a},{b},{c},{d },{e},{f},{g},{h},{i}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

g e h pertencem a árvores distintas?

Algoritmo de Kruskal



Sim.
Faz a união das árvores de g e h e adiciona a aresta a X (conjunto que guarda as arestas da AGM)

Para cada aresta ordenada (u,v), u e v perten[...] árvores/conjuntos distintos?

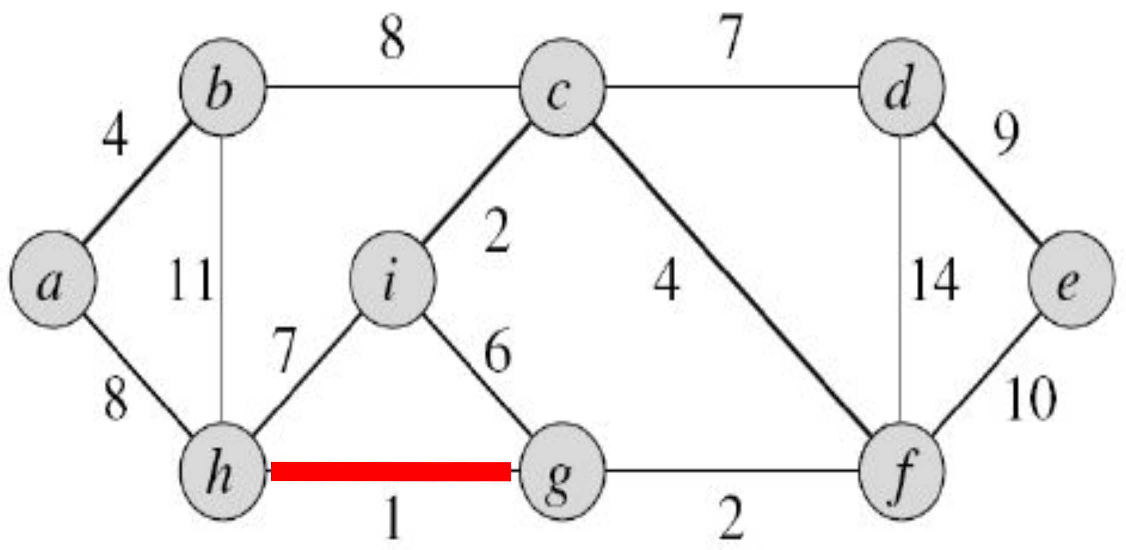
Floresta:

{{a},{b},{c},{d },{e},{f},{g},{h},{i}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

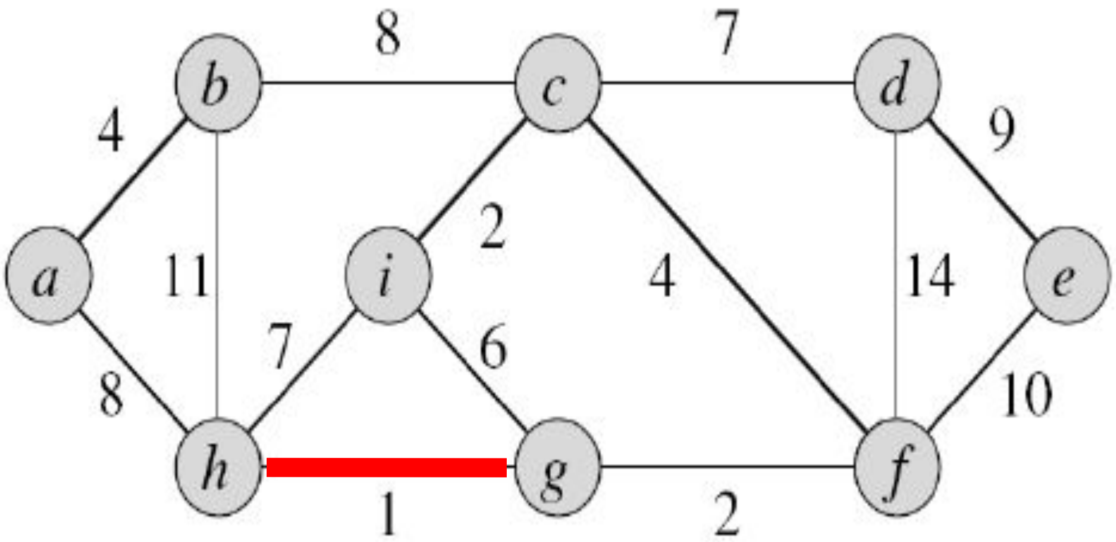
Floresta:

{{a},{b},{c},{d },{e},{f},{g,h},{i}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Floresta:

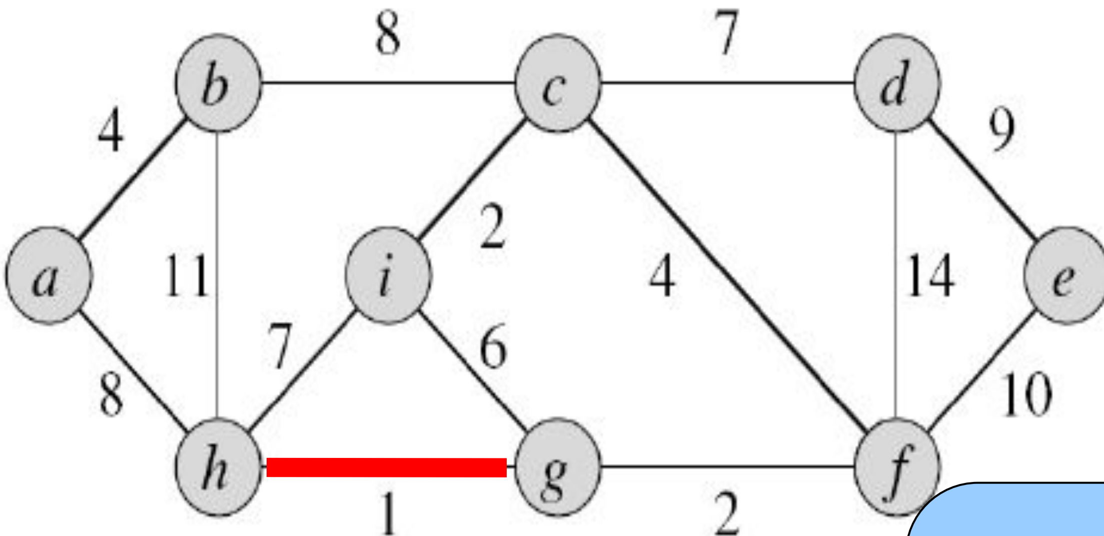
{{a},{b},{c},{d },{e},{f},{g,h},{i}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

c e i pertencem a árvores distintas?

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v perten[...]
árvores/conjuntos distintos?

Sim.
Faz a união das árvores de c e i e adiciona a aresta a X (conjunto que guarda as arestas da AGM)

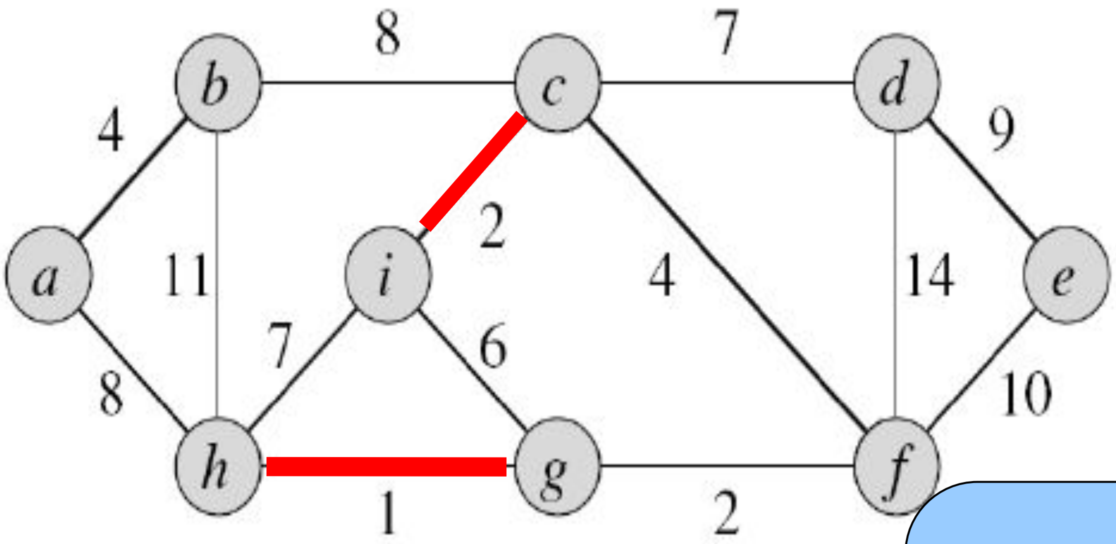
Floresta:

{{a},{b},{c},{d },{e},{f},{g,h},{i}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Sim.
Faz a união das árvores de c e i e adiciona a aresta a X (conjunto que guarda as arestas da AGM)

Para cada aresta ordenada (u,v), u e v perten
árvores/conjuntos distintos?

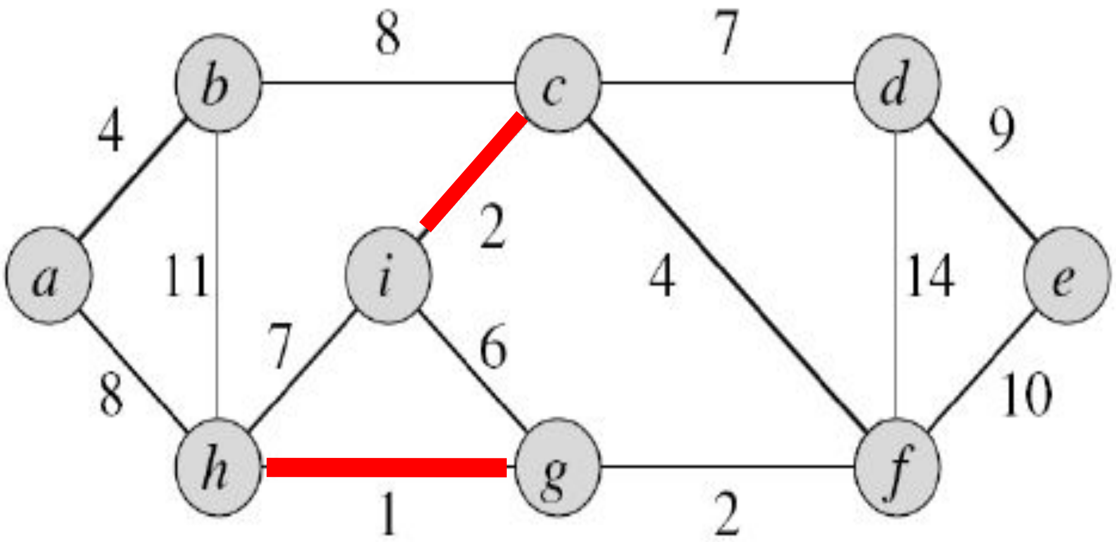
Floresta:

{{a},{b},{c,i},{d },{e},{f},{g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

f e g pertencem a árvores distintas?

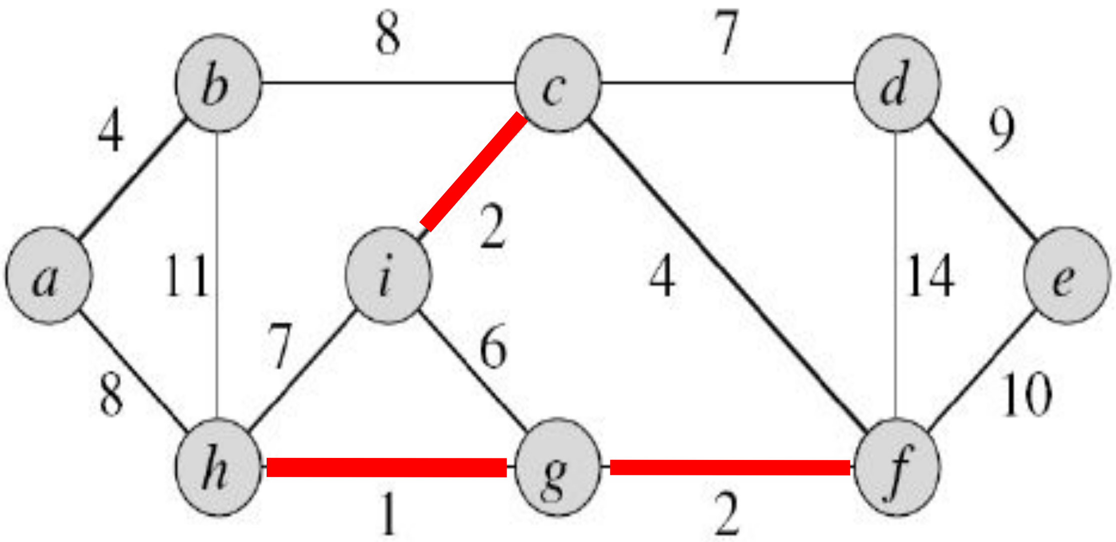
Floresta:

{{a},{b},{c,i},{d },{e},{f},{g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Sim

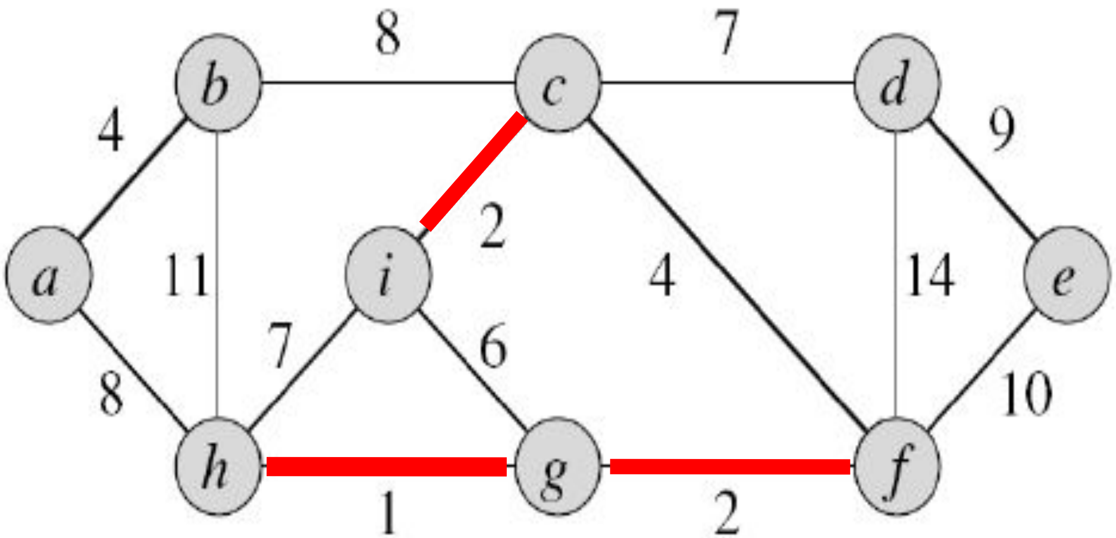
Floresta:

{{a},{b},{c,i},{d },{e},{f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Floresta:

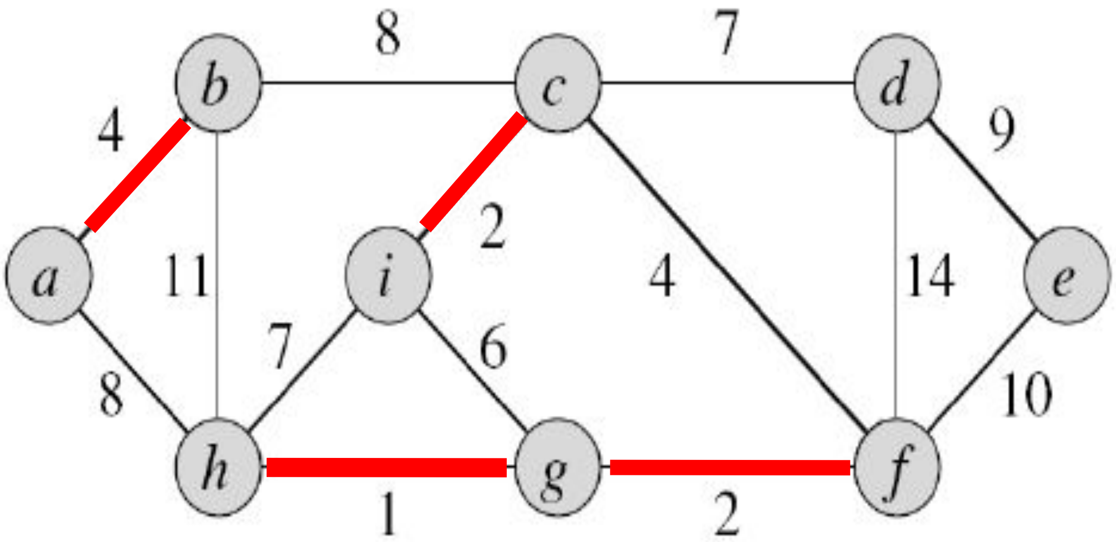
{{a},{b},{c,i},{d },{e},{f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

a e b pertencem a árvores distintas?

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Sim

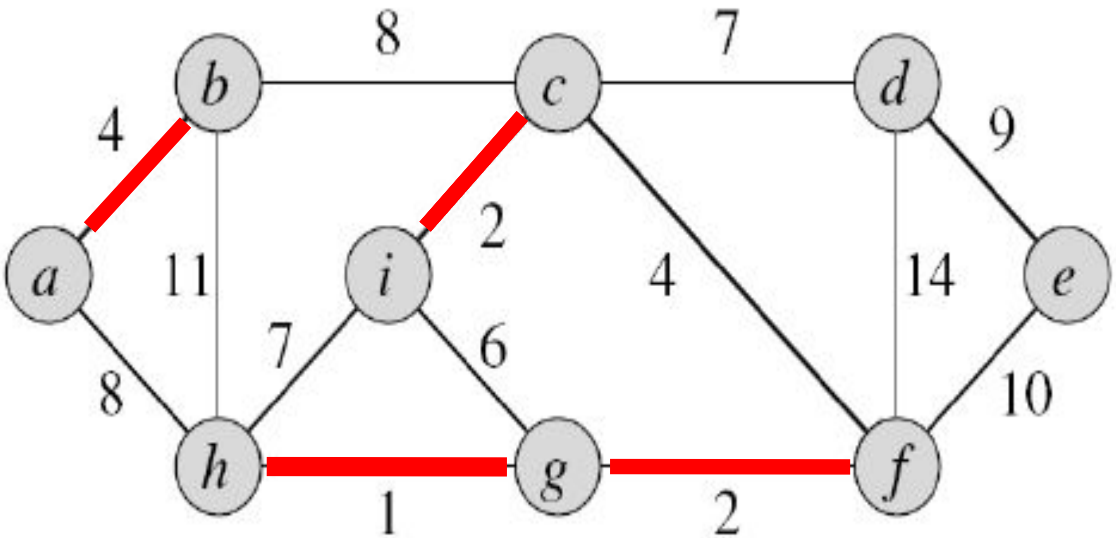
Floresta:

{{a,b},{c,i},{d },{e},{f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

c e f pertencem a árvores distintas?

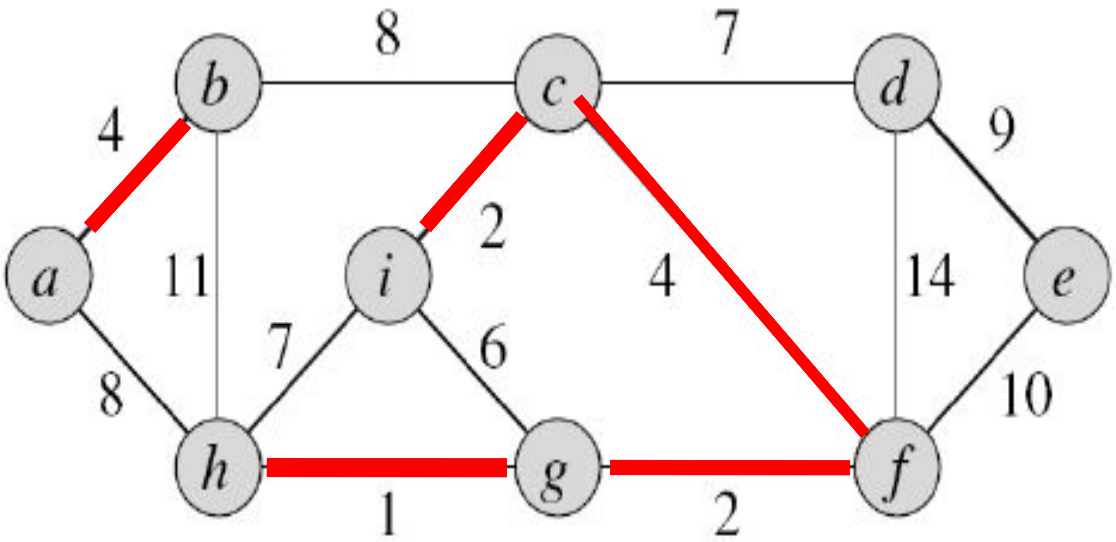
Floresta:

{{a,b},{c,i},{d },{e},{f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Sim

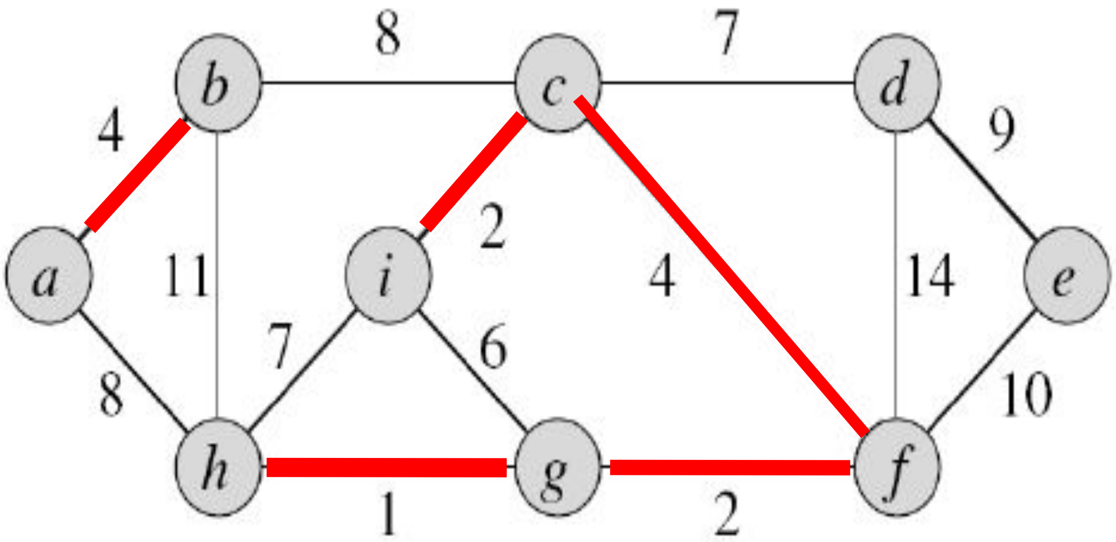
Floresta:

{{a,b},{d },{e},{c,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Floresta:

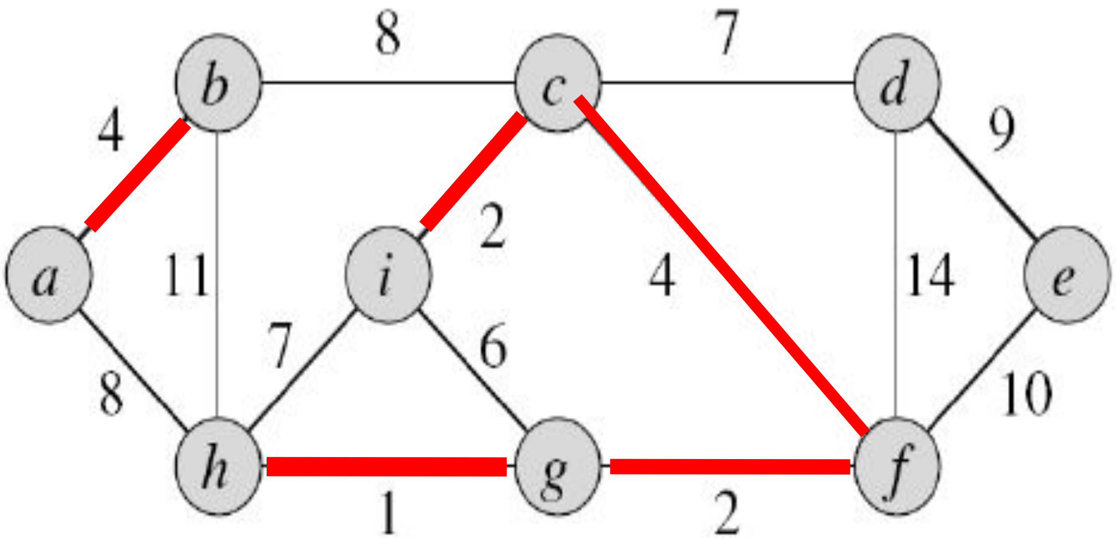
{{a,b},{d },{e},{c,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g ,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

g e i pertencem a árvores distintas?

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Não!!!!

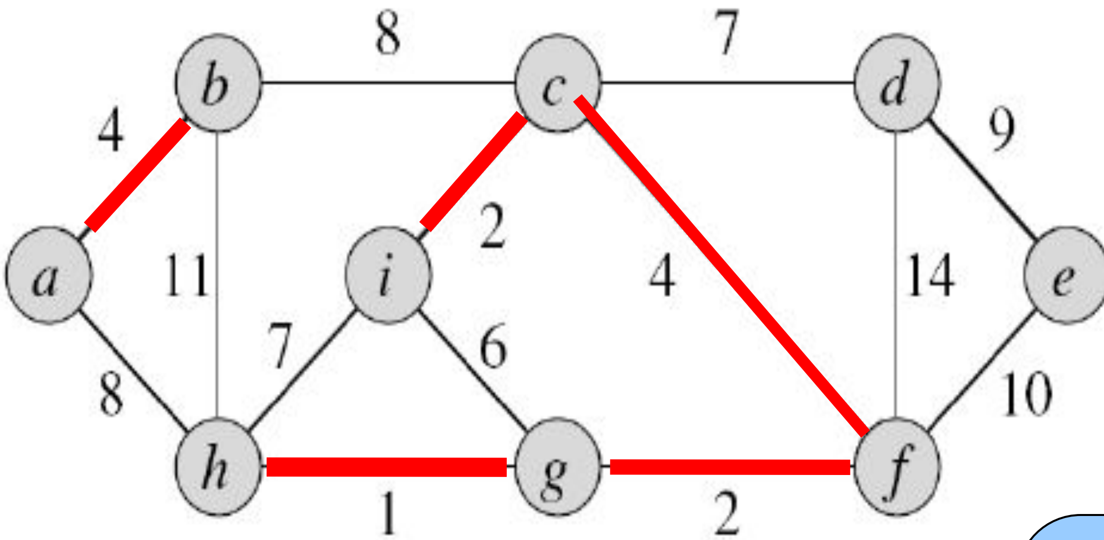
Floresta:

{{a,b},{d },{e},{c,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Não posso adicionar a aresta a X porque a aresta criaria um ciclo. Nada a ser feito

Para cada aresta ordenada (u,v), u e v pertencem árvores/conjuntos distintos?

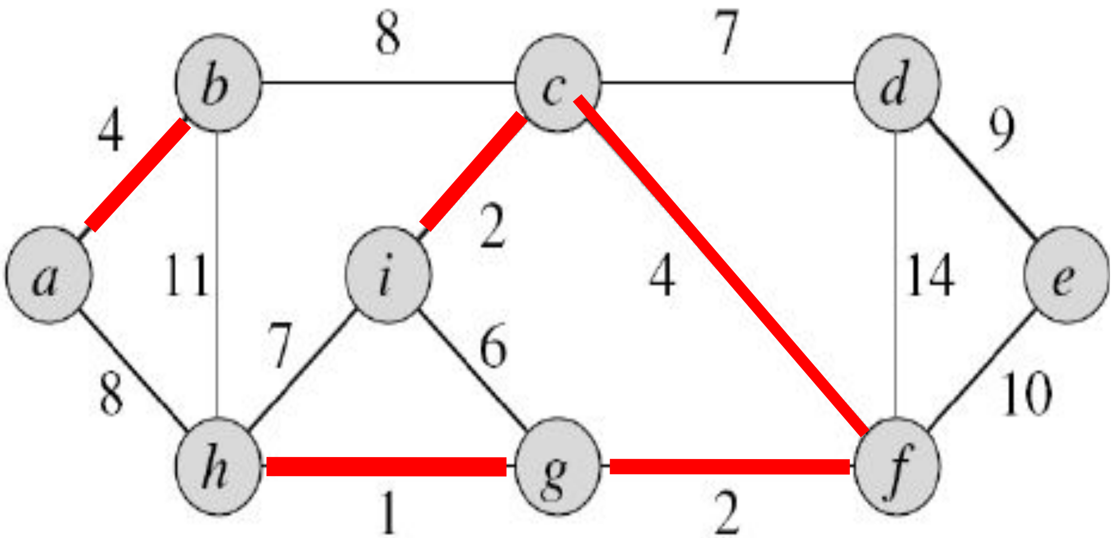
Floresta:

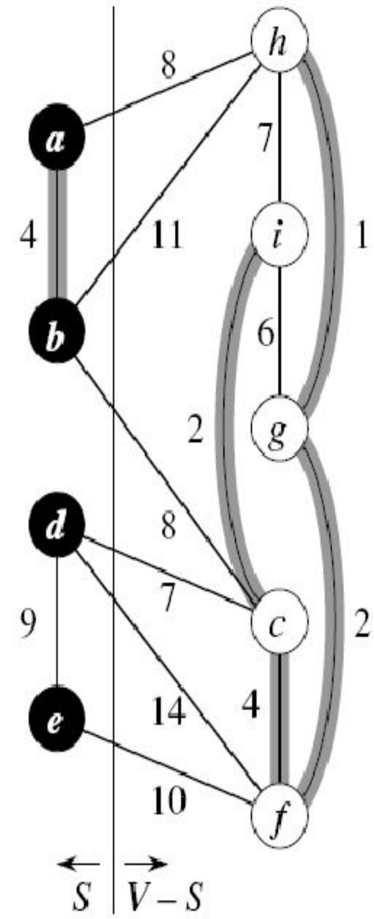
{{a,b},{d },{e},{c,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal





Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Floresta:

{{a,b},{d },{e},{c,i,f,g,h}}

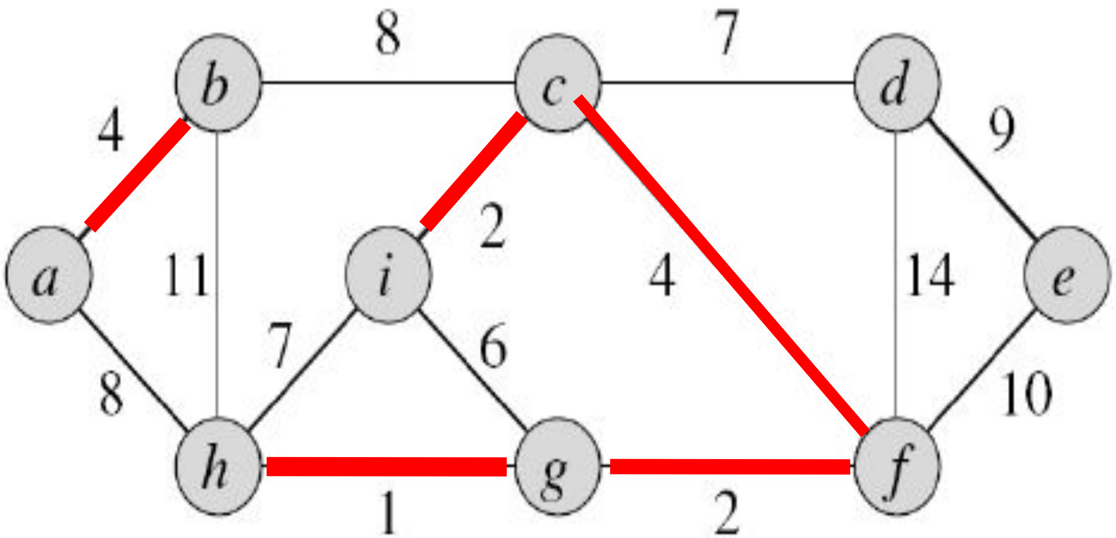
O conjunto de arestas é ordenado pelo pe

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h

(e,f); (b,h); (d,f)

Como isso está relacionado com o algoritmo genérico?

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

c e d pertencem a árvores distintas?

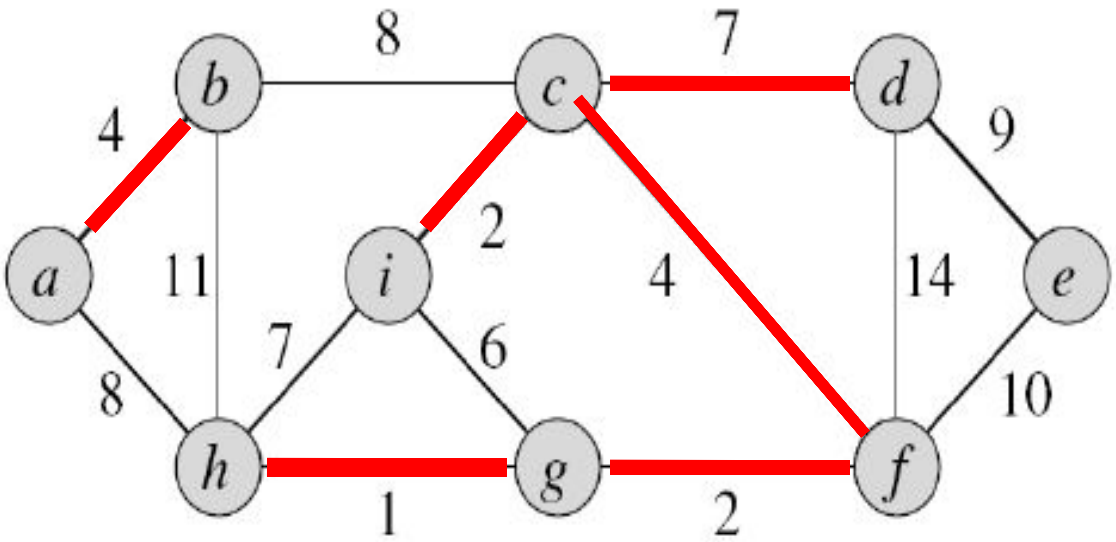
Floresta:

{{a,b},{d },{e},{c,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Sim

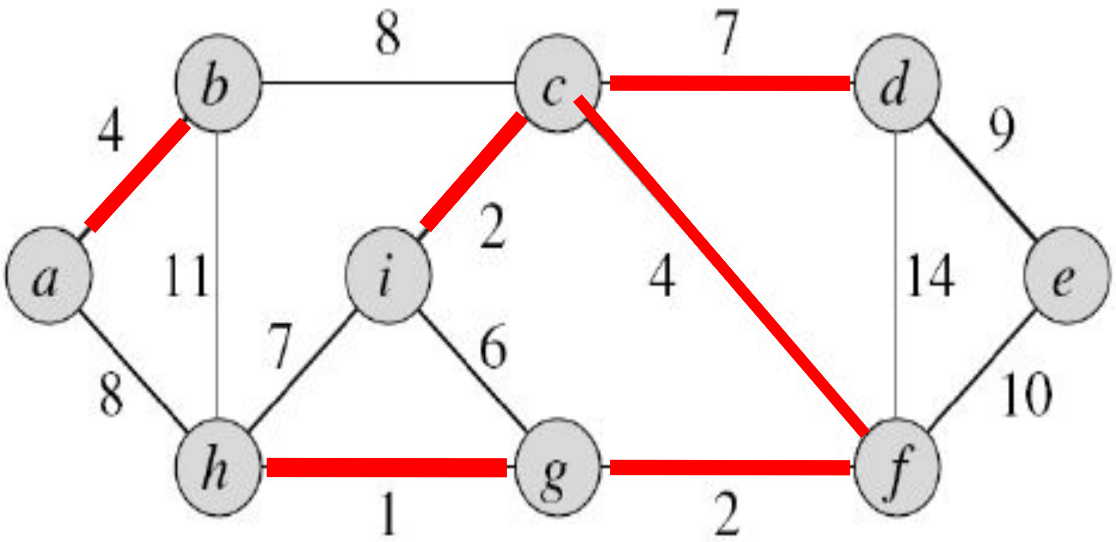
Floresta:

{{a,b},{e},{c,d,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

h e i pertencem a árvores distintas?

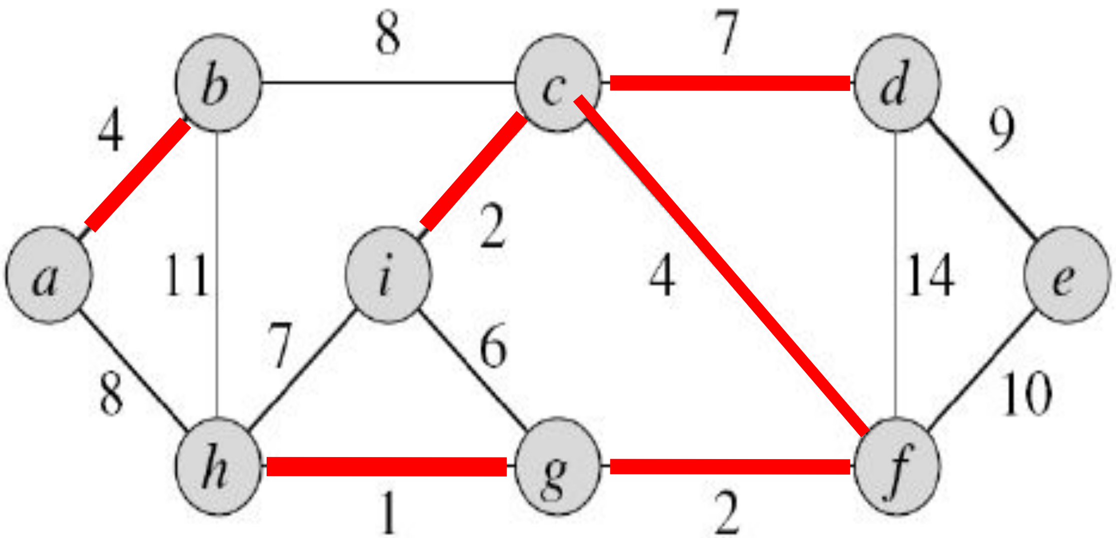
Floresta:

{{a,b},{e},{c,d,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Nao!!!

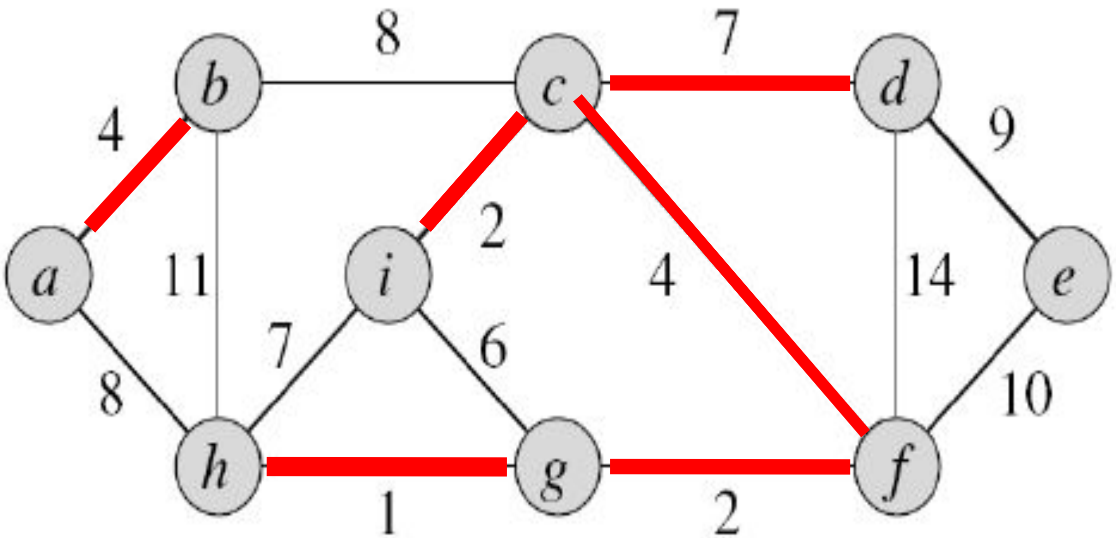
Floresta:

{{a,b},{e},{c,d,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

a e h pertencem a árvores distintas?

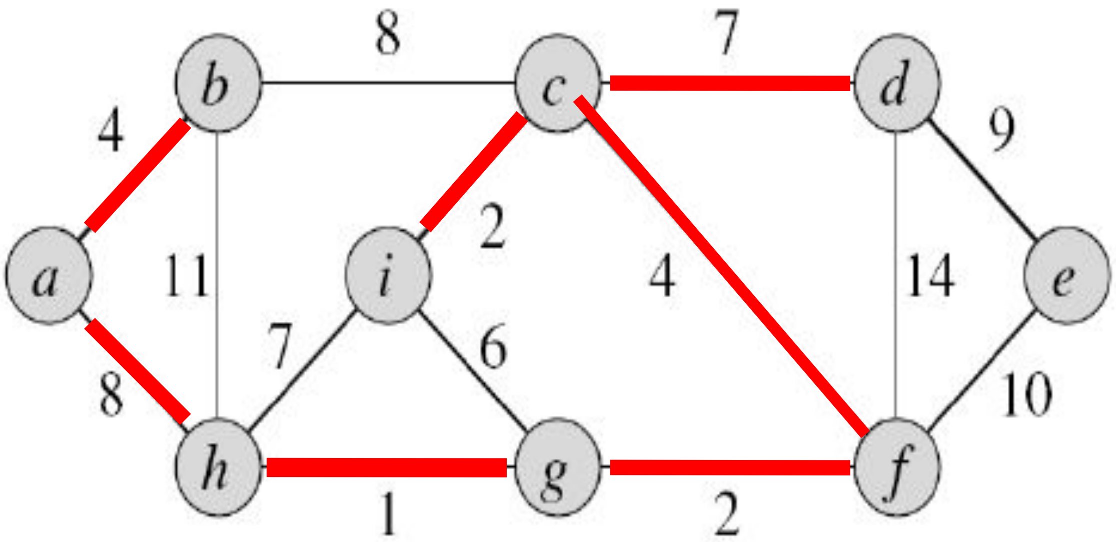
Floresta:

{{a,b},{e},{c,d,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Sim

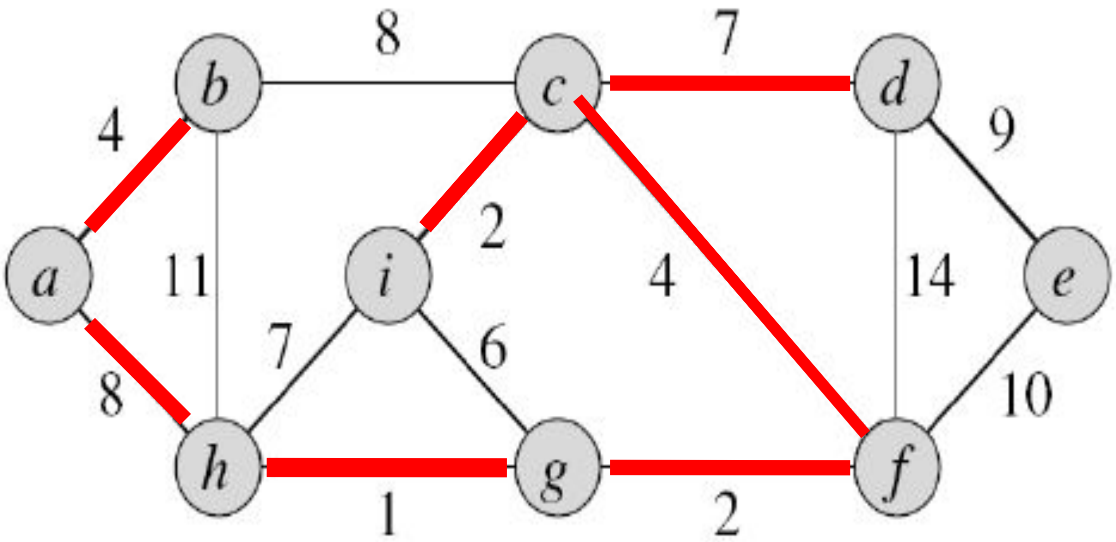
Floresta:

{{e},{a,b,c,d,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

b e c pertencem a árvores distintas?

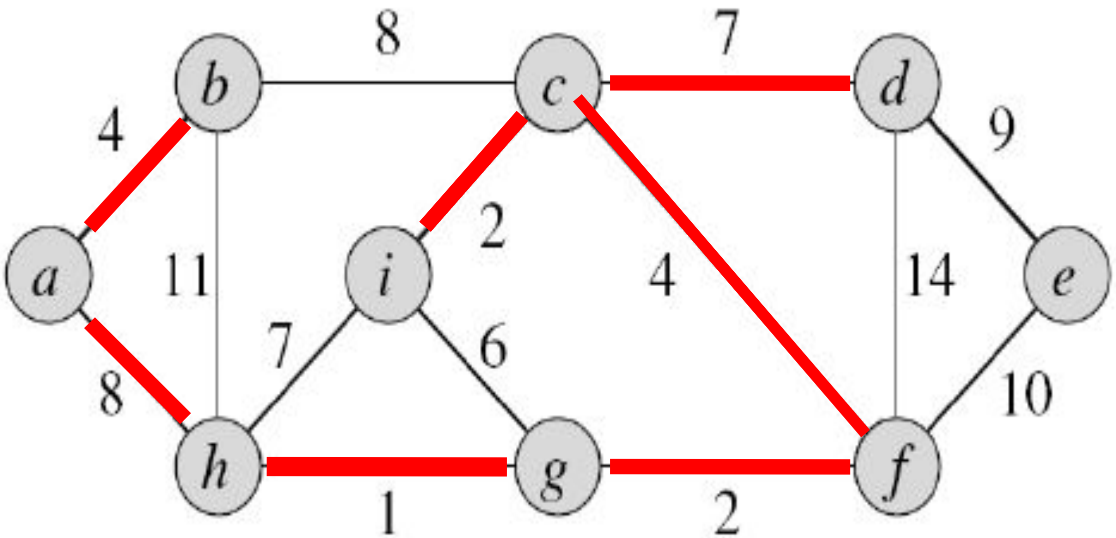
Floresta:

{{e},{a,b,c,d,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Não!!!

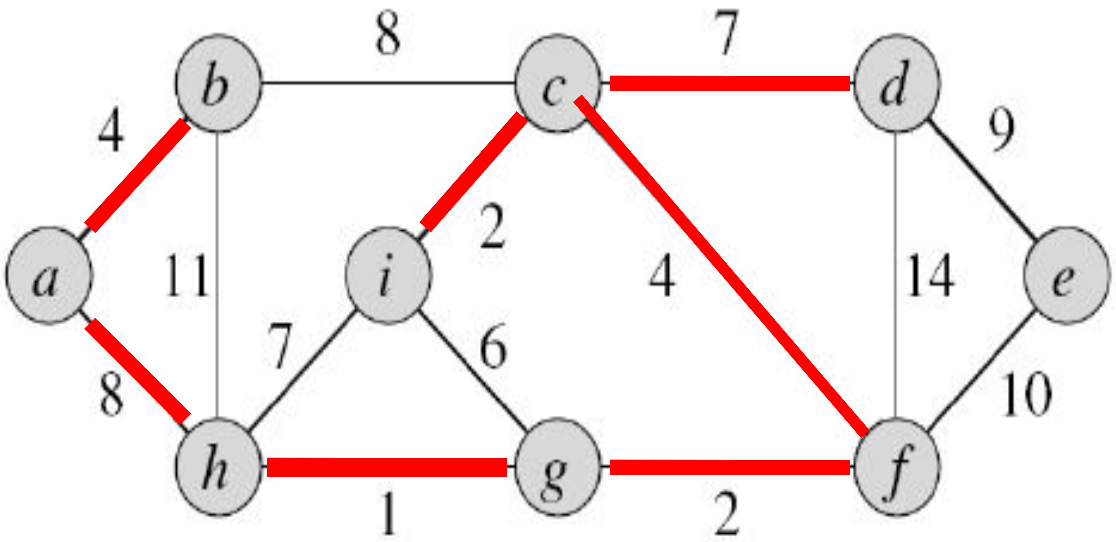
Floresta:

{{e},{a,b,c,d,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

d e e pertencem a árvores distintas?

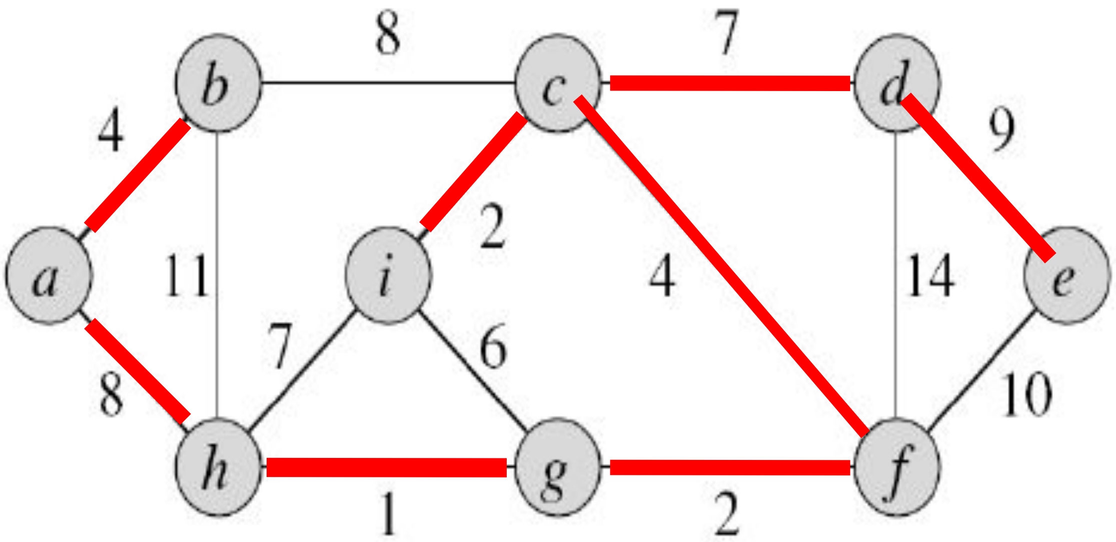
Floresta:

{{e},{a,b,c,d,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Sim

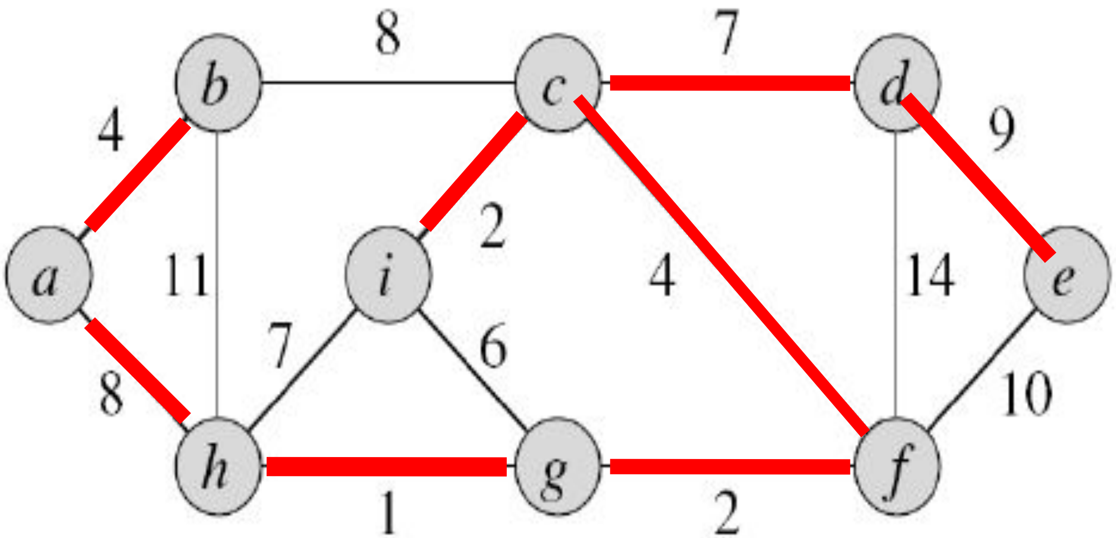
Floresta:

{{a,b,c,d,e,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

e e f pertencem a árvores distintas? Não

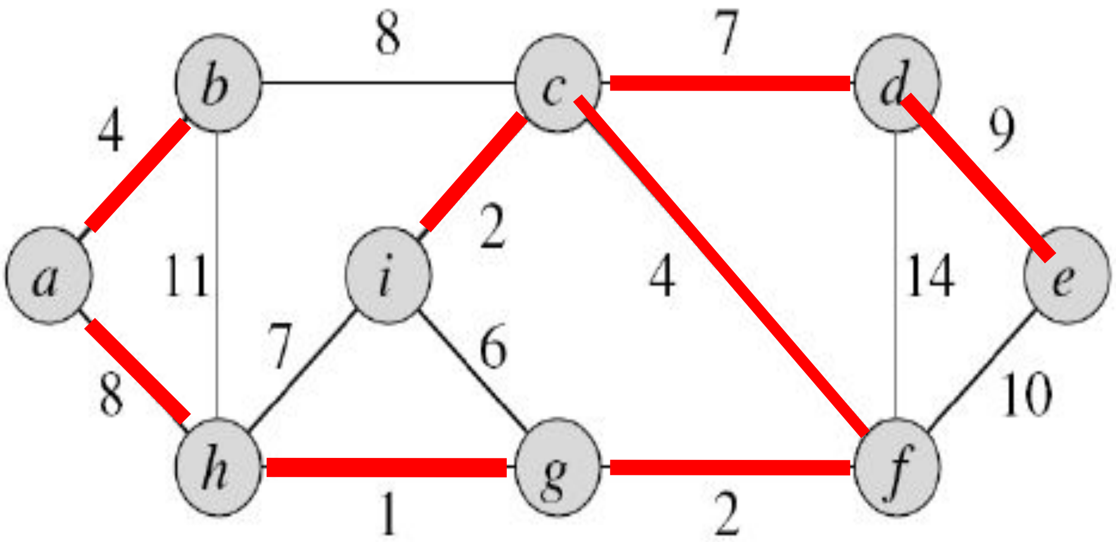
Floresta:

{{a,b,c,d,e,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

Floresta:

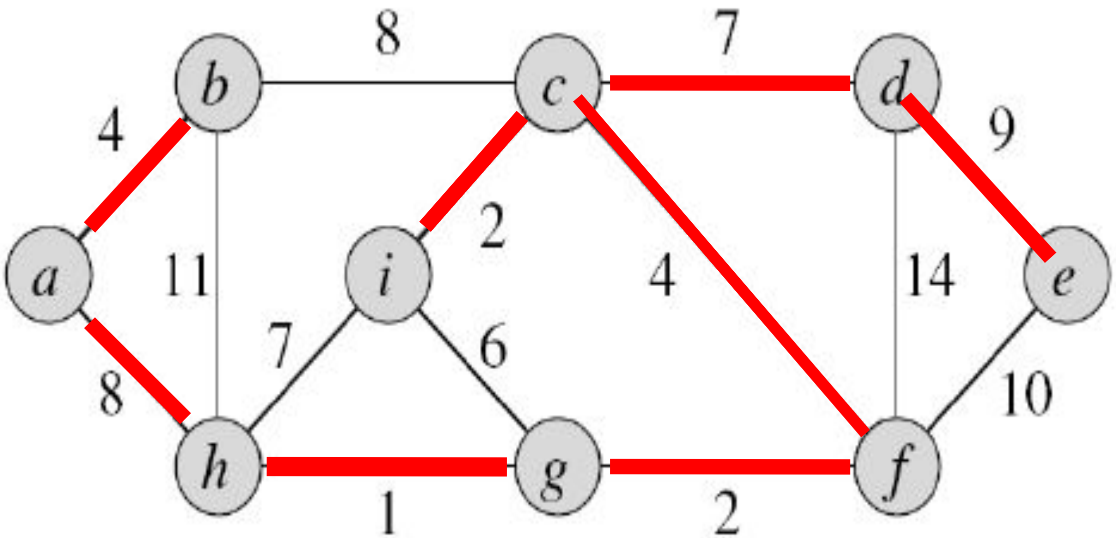
{{a,b,c,d,e,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

b e h pertencem a árvores distintas? Não

Algoritmo de Kruskal



Para cada aresta ordenada (u,v), u e v pertencem a árvores/conjuntos distintos?

d e f pertencem a árvores distintas? Não

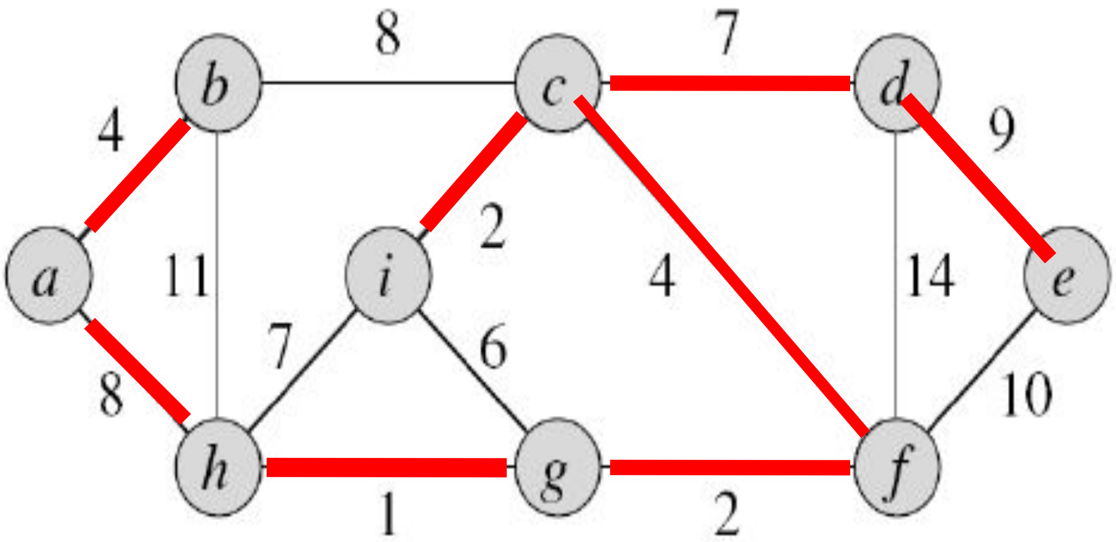
Floresta:

{{a,b,c,d,e,i,f,g,h}}

O conjunto de arestas é ordenado pelo peso:

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal



O algoritmo devolve o conjunto X (arestas marcadas em vermelho no grafo)

(g,h); (c, i); (f, g); (a,b); (c,f); (g,i); (c,d); (h,i); (a,h); (b, c); (d,e); (e,f); (b,h); (d,f)

Algoritmo de Kruskal

- A implementação usa uma estrutura que mantém vários conjuntos disjuntos de elementos.
- Cada conjunto contém os vértices em uma árvore da floresta.
- A operação FIND-SET(u) retorna o elemento que representa o conjunto que contém o vértice u.
- Para determinar se dois vértices u e v pertencem a mesma árvore testamos se FIND-SET(u) é igual a FIND-SET(v).
- A combinação de árvores é realizada pelo procedimento UNION(u,v).

Algoritmo de Kruskal

```
KRUSKAL(V, A, w)

X ← { }        ▷ O conjunto X que guarda as arestas AGM inicialmente é vazio
for each vertex v in V
     MAKE-SET(v)
sort A into nondecreasing order by weight w
for each (u, v) taken from the sorted list
       if FIND-SET(u) ≠ FIND-SET(v)
       then X ← X ∪ {(u, v)}
            UNION(u, v)
return X
```

Algoritmo de Kruskal

```
KRUSKAL(V, A, w)

X ← { }          ▷ O conjunto X que guarda as arestas AGM inicialmente é vazio
for each vertex v in V
     MAKE-SET(v)   ▷ |V| árvores são criadas
sort A into nondecreasing order by weight w
for each (u, v) taken from the sorted list
       if FIND-SET(u) ≠ FIND-SET(v)
       then X ← X ∪ {(u, v)}
            UNION(u, v)
return X
```

Algoritmo de Kruskal

```
KRUSKAL(V, A, w)

X ← { }                     ▷ X no final terá as arestas da AGM
for each vertex v in V
    MAKE-SET(v)    ▷ |V| árvores são criadas
sort A into nondecreasing order by weight w
for each (u, v) taken from the sorted list
      if FIND-SET(u) ≠ FIND-SET(v)
      then X ← X ∪ {(u, v)}
           UNION(u, v)
return X
```

O conjunto de arestas é ordenado pelo peso.

Algoritmo de Kruskal

KRUSKAL(V, A, w)

X ← { } ▷ X no final terá as arestas da AGM
for each vertex v in V
    MAKE-SET(v) ▷ |V| árvores são criadas
sort A into nondecreasing order by weight w
for each (u, v) taken from the sorted list
    if FIND-SET(u) ≠ FIND-SET(v)
    then X ← X ∪ {(u, v)}
        UNION(u, v)
return X

Para cada aresta da lista ordenada

Algoritmo de Kruskal

KRUSKAL(V, A, w)

X ← { } ▷ X no final terá as arestas da AGM
for each vertex v in V
MAKE-SET(v) ▷ |V| árvores são criadas
sort A into nondecreasing order by weight w
for each (u, v) taken from the sorted list
if FIND-SET(u) ≠ FIND-SET(v)
then X ← X ∪ {(u, v)}
UNION(u, v)
return X

Se u e v pertencem a árvores distintas (a aresta conecta duas árvores distintas)

Algoritmo de Kruskal

KRUSKAL(V, A, w)

X ← { } ▷ X no final terá as arestas da AGM
for each vertex v in V
    MAKE-SET(v) ▷ |V| árvores são criadas
sort A into nondecreasing order by weight w
for each (u, v) taken from the sorted list
    if FIND-SET(u) ≠ FIND-SET(v)
    then X ← X ∪ {(u, v)}
        UNION(u, v)
return X

Se u e v pertencem a árvores distintas (a aresta conecta duas árvores distintas)
A aresta não cria um ciclo.

Algoritmo de Kruskal

```
KRUSKAL(V, A, w)

X ← { }                    ▷ X no final terá as arestas da AGM
for each vertex v in V
     MAKE-SET(v)    ▷ |V| árvores são criadas
sort A into nondecreasing order by weight w
for each (u, v) taken from the sorted list
       if FIND-SET(u) ≠ FIND-SET(v)
       then X ← X ∪ {(u, v)} ▷ a aresta  segura é adicionada ao conjunto X
            UNION(u, v)
return X
```

Algoritmo de Kruskal

KRUSKAL(V, A, w)

X ← { } ▷ X no final terá as arestas da AGM
for each vertex v in V
    MAKE-SET(v) ▷ |V| árvores são criadas
sort A into nondecreasing order by weight w
for each (u, v) taken from the sorted list
    if FIND-SET(u) ≠ FIND-SET(v)
    then X ← X ∪ {(u, v)} ▷ a aresta segura é adicionada ao conjunto X
        UNION(u, v) ▷faço a união das árvores de u e v
return X

Algoritmos

- Algoritmo de Kruskal:
  - Trabalha com uma floresta antes de gerar a Árvore Geradora Mínima.
  - Depois da última iteração do algoritmo a floresta é apenas uma árvore.
- Algoritmo de Prim:
  - Trabalha com uma árvore durante e no final do algoritmo

Algoritmos

- Algoritmo de Kruskal:
  - A aresta segura adicionada a X é sempre uma aresta de peso mínimo no grafo que conecta dois componentes distintos (duas árvores distintas na floresta).
- Algoritmo de Prim:
  - A aresta segura é sempre a aresta de peso mínimo que conecta a árvore a um vértice não presente no conjunto X.

Algoritmo de Prim

- Muito similar ao algoritmo de Dijkstra
- Começa com um vértice árbitrário r e cresce a AGM em cada iteração partindo de r.
- Corte:
  - S corresponde aos vértices que fazem parte da AGM parcialmente construída
  - V-S os outros vértices

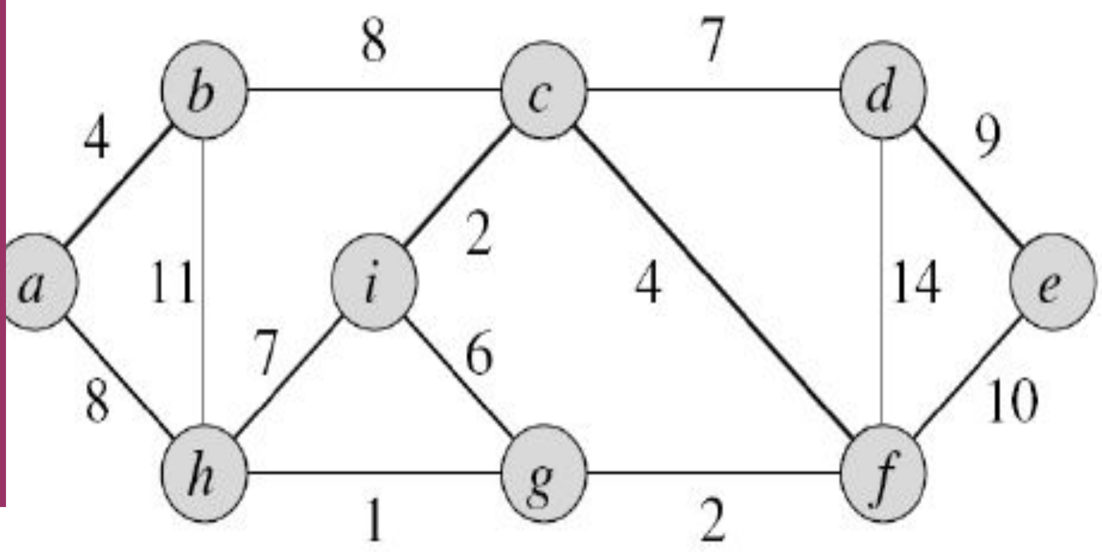
Algoritmo de Prim

- Algoritmo de Prim utiliza:
  - Fila de prioridade Q: Vértices que ainda não fazem parte da AGM parcial
  - key[v]: peso da aresta mais leve que conecta v a um vértice da AGM parcialmente construída;
  - π[v]: vértice pai do vértice v;

Algoritmo de Prim

PRIM(V, A, w, r)

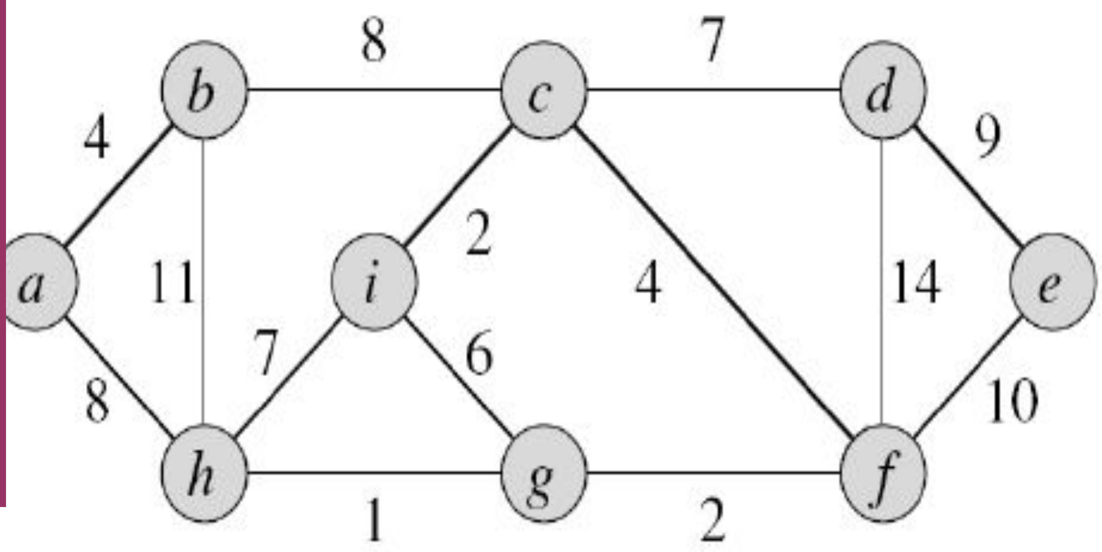
Q ← { }
for each u in V
    key[u] ← ∞
    π[u] ← NIL
    INSERT(Q, u)
DECREASE-KEY(Q, r, 0) ▹ key[r] ← 0



vértice	a	b	c	d	e	f	g	h	i
chave									
π									
Q									

Algoritmo de Prim

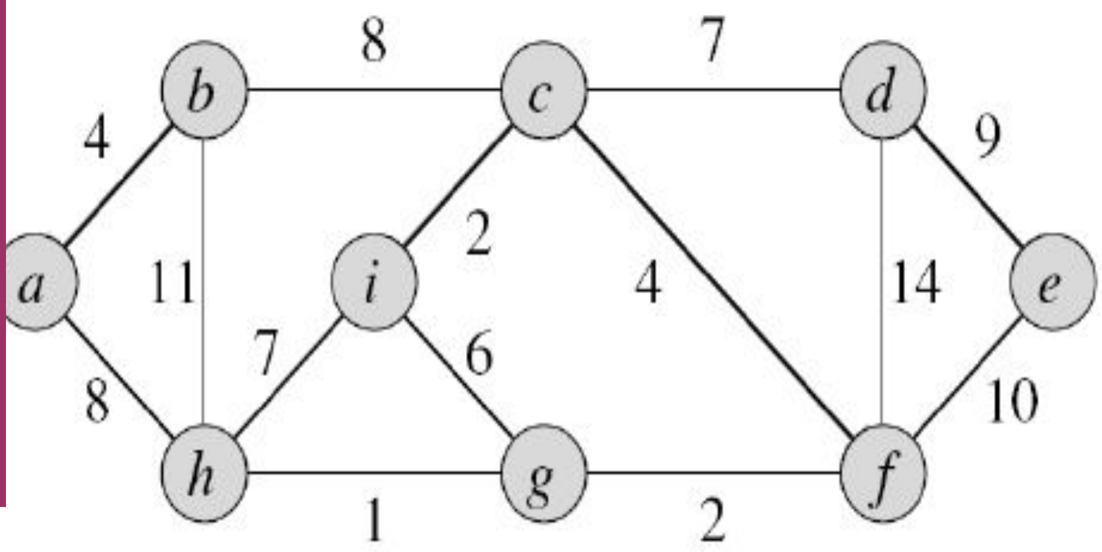
```
PRIM(V, A, w, r )

Q ← { }
for each u in V
        key[u] ← ∞
        π[u] ← NIL
        INSERT(Q, u)
DECREASE-KEY(Q, r, 0) ▹ key[r ] ← 0
```



vértice	a	b	c	d	e	f	g	h	i
chave	∞	∞	∞	∞	∞	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
Q	✓	✓	✓	✓	✓	✓	✓	✓	✓

Algoritmo de Prim

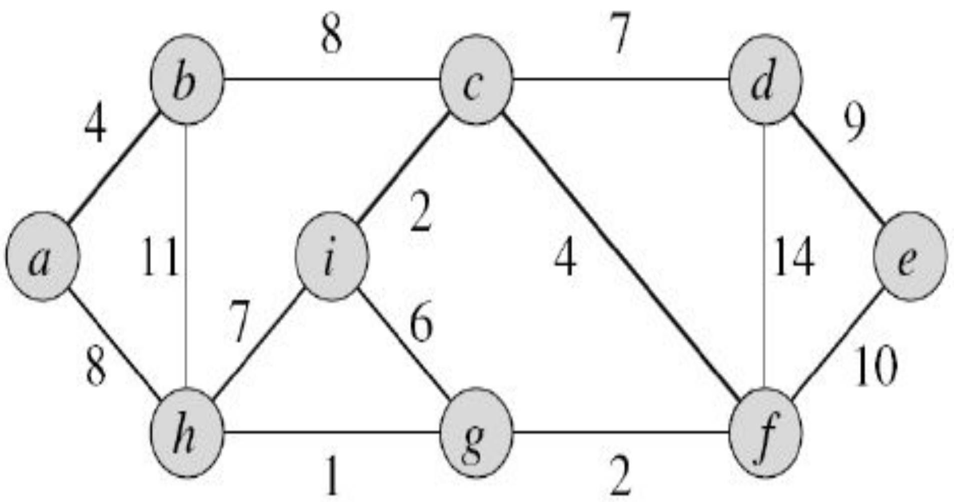
```
PRIM(V, A, w, r )

Q ← { }
for each u in V
        key[u] ← ∞
        π[u] ← NIL
        INSERT(Q, u)
DECREASE-KEY(Q, r, 0) ▹ key[r ] ← 0
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	∞	∞	∞	∞	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
Q	✔	✔	✔	✔	✔	✔	✔	✔	✔

Algoritmo de Prim

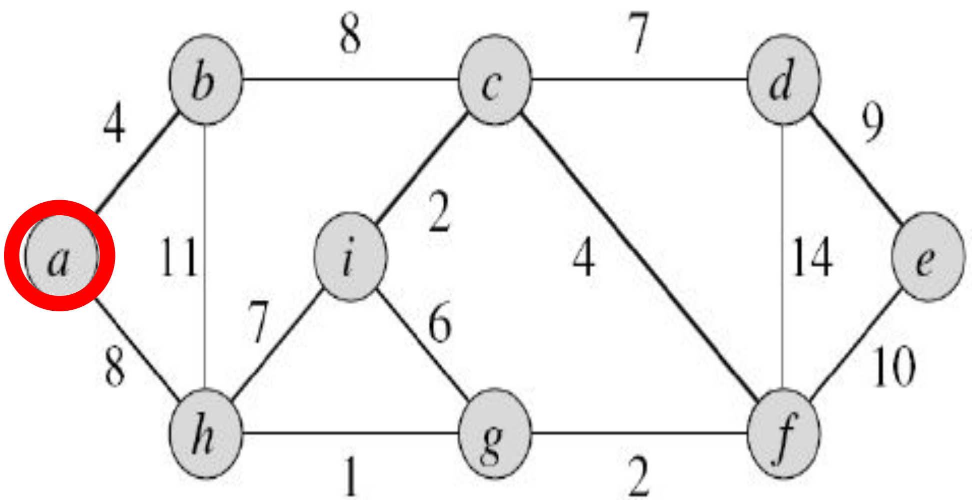
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	∞	∞	∞	∞	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
Q	✔	✔	✔	✔	✔	✔	✔	✔	✔

Algoritmo de Prim

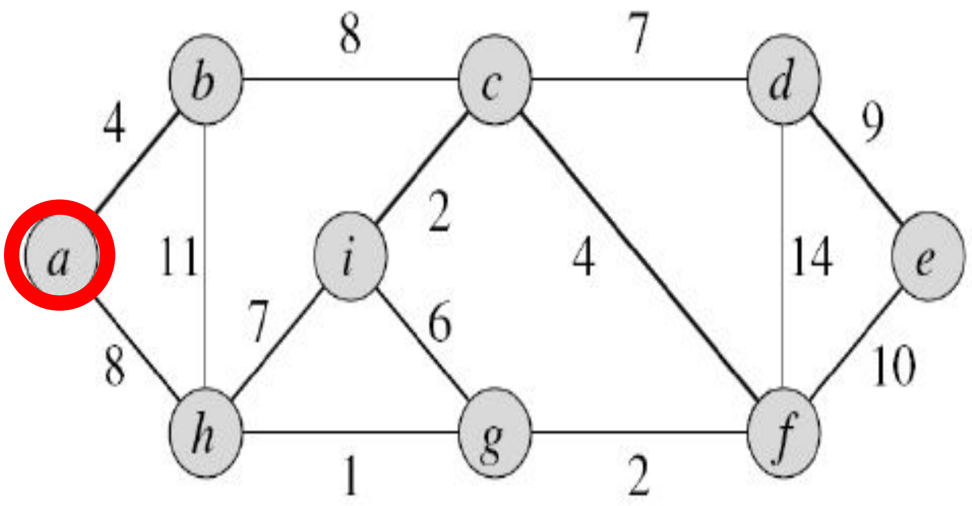
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	∞	∞	∞	∞	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
Q		✔	✔	✔	✔	✔	✔	✔	✔

Algoritmo de Prim

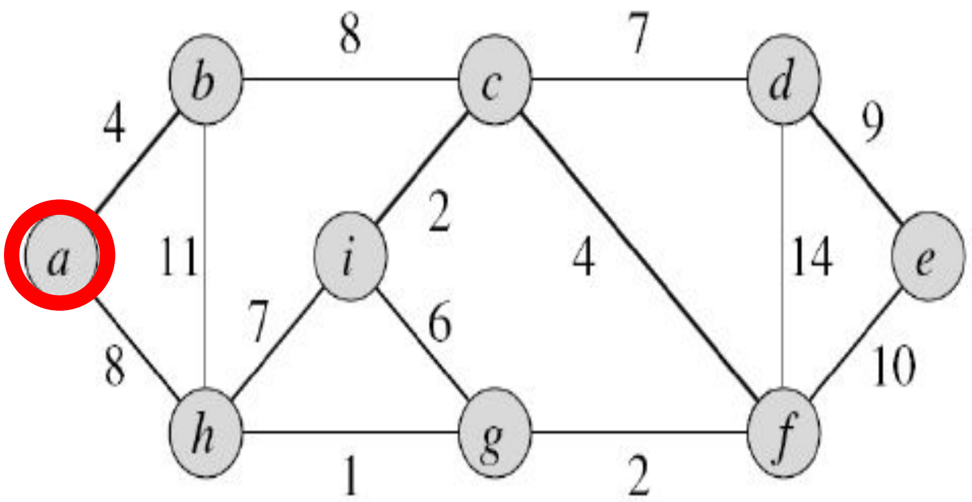
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	∞	∞	∞	∞	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
Q		✓	✓	✓	✓	✓	✓	✓	✓

Algoritmo de Prim

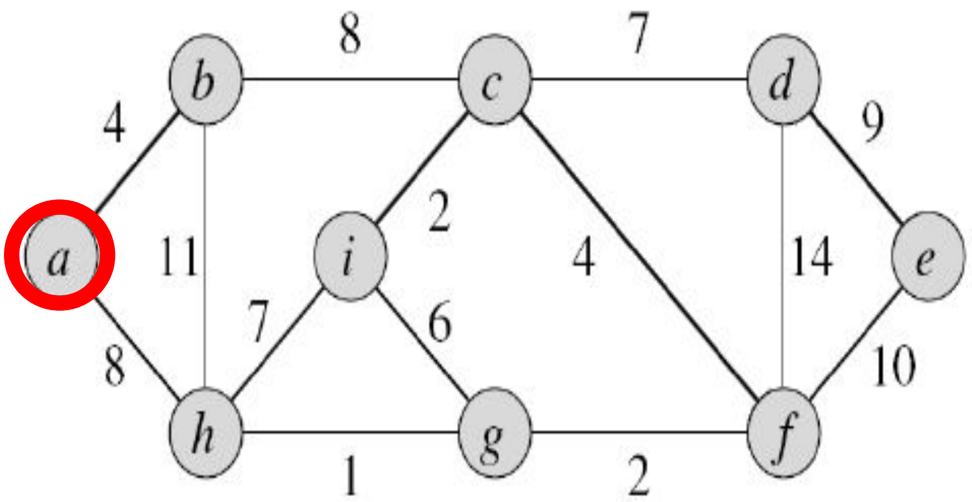
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	∞	∞	∞	∞	∞	8	∞
π	NIL	a	NIL	NIL	NIL	NIL	NIL	a	NIL
Q		✔	✔	✔	✔	✔	✔	✔	✔

Algoritmo de Prim

```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```

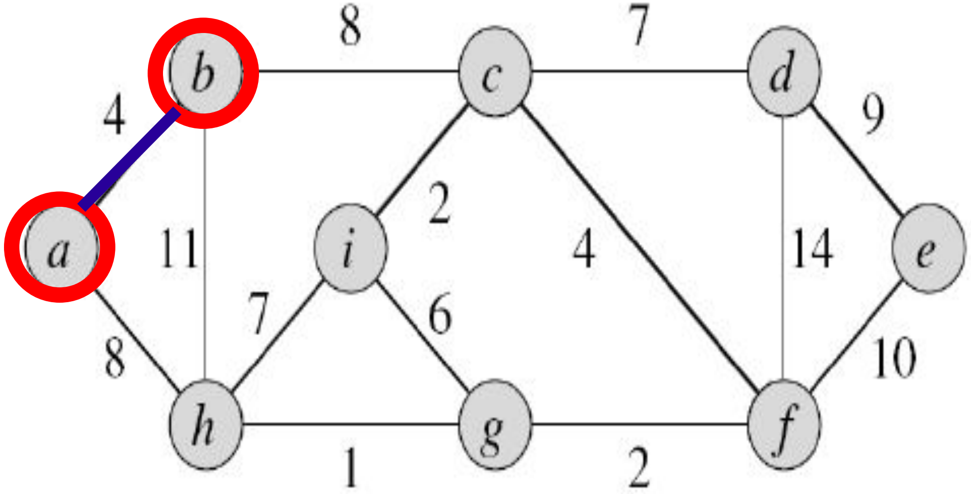


Lembrando que key[v] é o peso da aresta mais leve que conecta v a um vértice da AGM parcialmente construída

vértice	a	b	c	d	e	f	g	h	i
chave	0	4	∞	∞	∞	∞	∞	8	∞
π	NIL	a	NIL	NIL	NIL	NIL	NIL	a	NIL
Q		✓	✓	✓	✓	✓	✓	✓	✓

Algoritmo de Prim

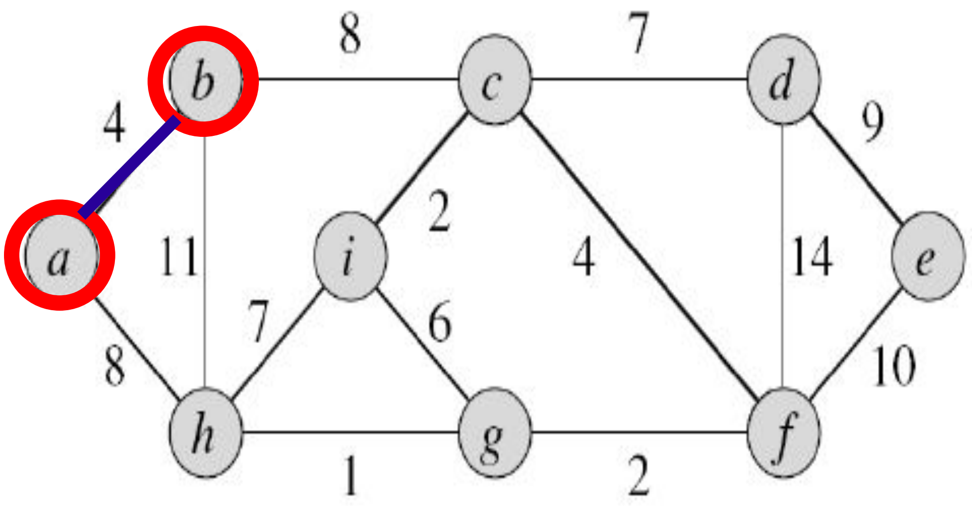
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	∞	∞	∞	∞	∞	8	∞
π	NIL	a	NIL	NIL	NIL	NIL	NIL	a	NIL
Q			✔	✔	✔	✔	✔	✔	✔

Algoritmo de Prim

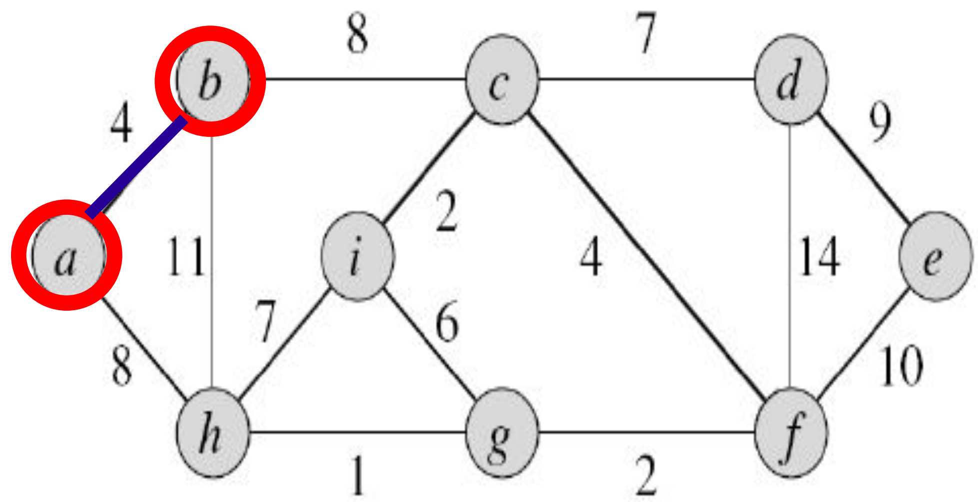
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	∞	∞	∞	∞	∞	8	∞
π	NIL	a	NIL	NIL	NIL	NIL	NIL	a	NIL
Q			✓	✓	✓	✓	✓	✓	✓

Algoritmo de Prim

```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```

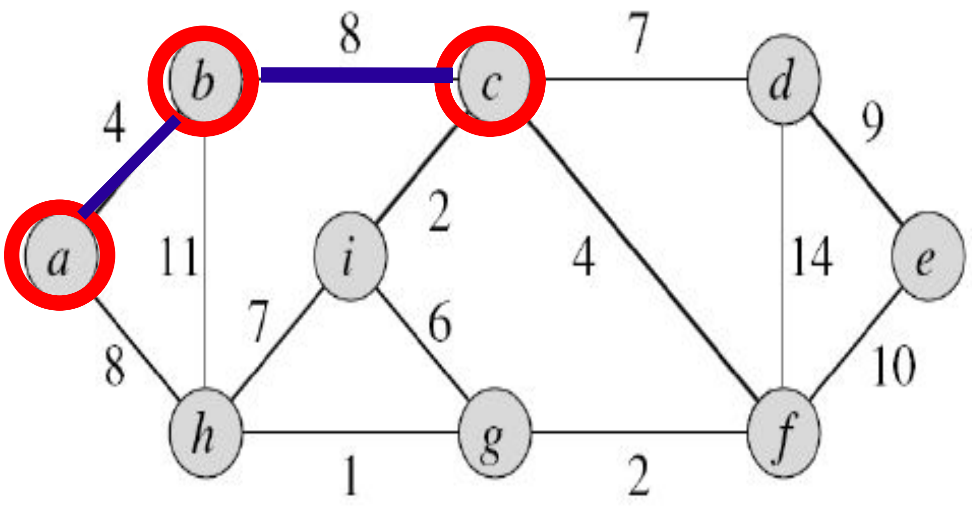


Lembrando que key[v] é o peso da aresta mais leve que conecta v a um vértice da AGM parcialmente

vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	∞	∞	∞	∞	8	∞
π	NIL	a	b	NIL	NIL	NIL	NIL	a	NIL
Q			✓	✓	✓	✓	✓	✓	✓

Algoritmo de Prim

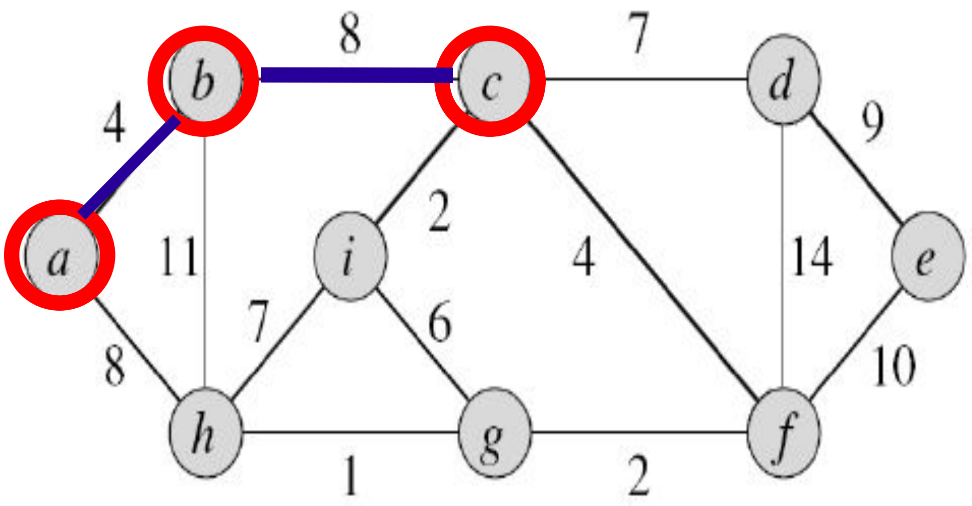
PRIM(V, A, w, r)
...
while Q is not empty
  u ← EXTRACT-MIN(Q)
  for each v in Adj[u]
    if v in Q and w(u, v) < key[v]
    then π[v] ← u
      DECREASE-KEY(Q, v, w(u, v))



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	∞	∞	∞	∞	8	∞
π	NIL	a	b	NIL	NIL	NIL	NIL	a	NIL
Q				✔	✔	✔	✔	✔	✔

Algoritmo de Prim

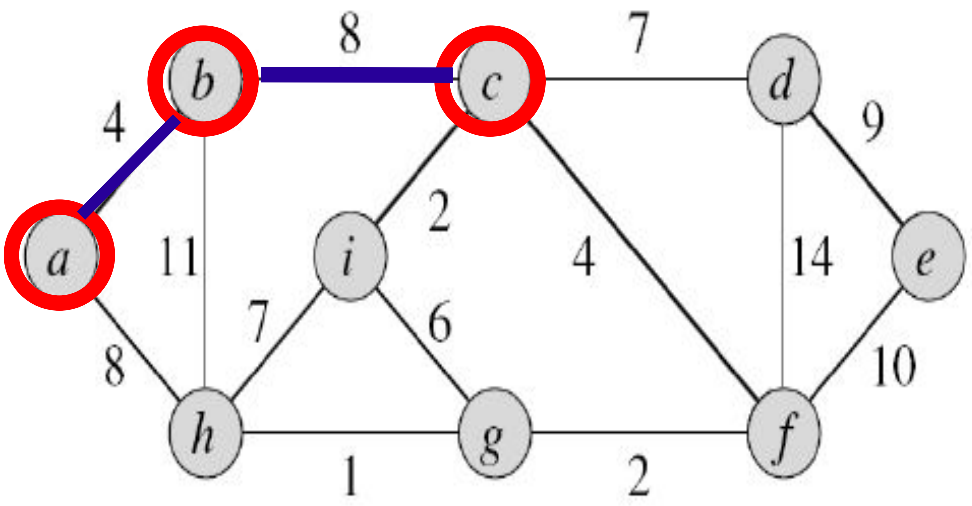
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	∞	∞	∞	∞	8	∞
π	NIL	a	b	NIL	NIL	NIL	NIL	a	NIL
Q				✔	✔	✔	✔	✔	✔

Algoritmo de Prim

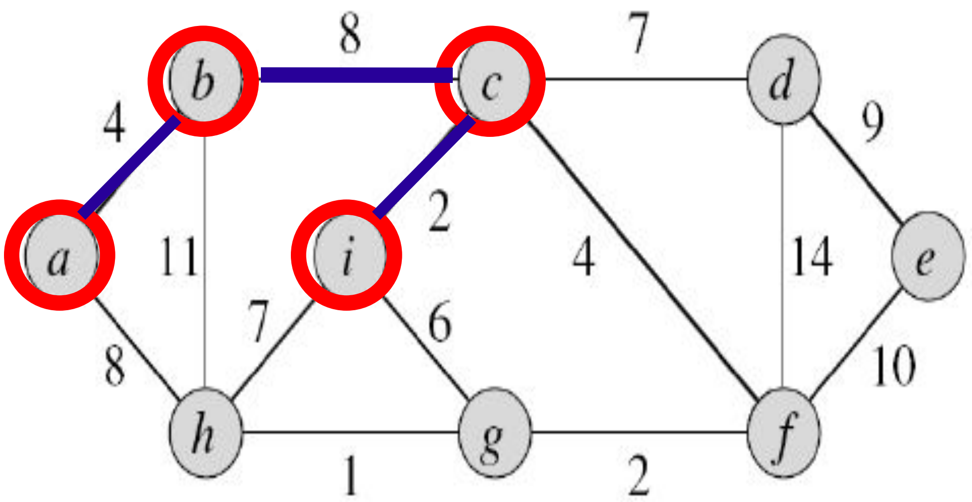
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	∞	4	∞	8	2
π	NIL	a	b	c	NIL	c	NIL	a	c
Q				✓	✓	✓	✓	✓	✓

Algoritmo de Prim

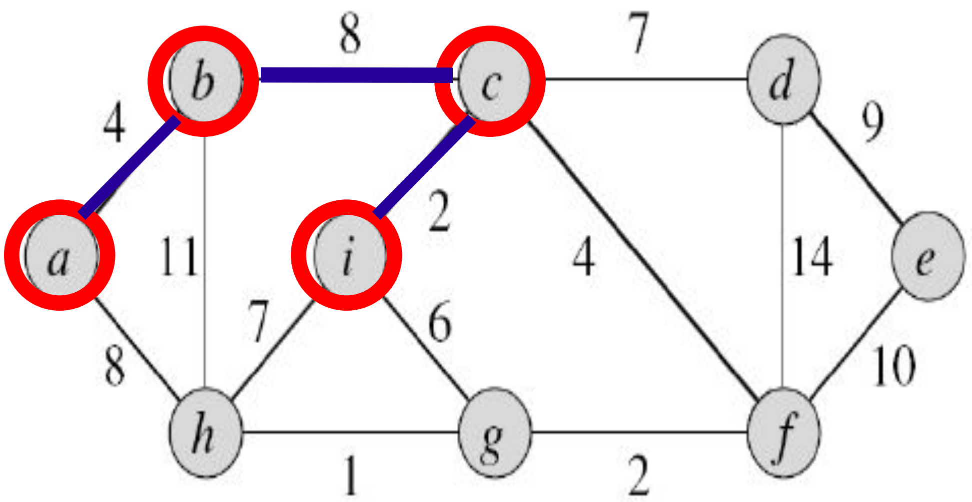
PRIM(V, A, w, r)
...
while Q is not empty
    u ← EXTRACT-MIN(Q)
    for each v in Adj[u]
        if v in Q and w(u, v) < key[v]
        then π[v] ← u
            DECREASE-KEY(Q, v, w(u, v))



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	∞	4	∞	8	2
π	NIL	a	b	c	NIL	c	NIL	a	c
Q				✓	✓	✓	✓	✓	

Algoritmo de Prim

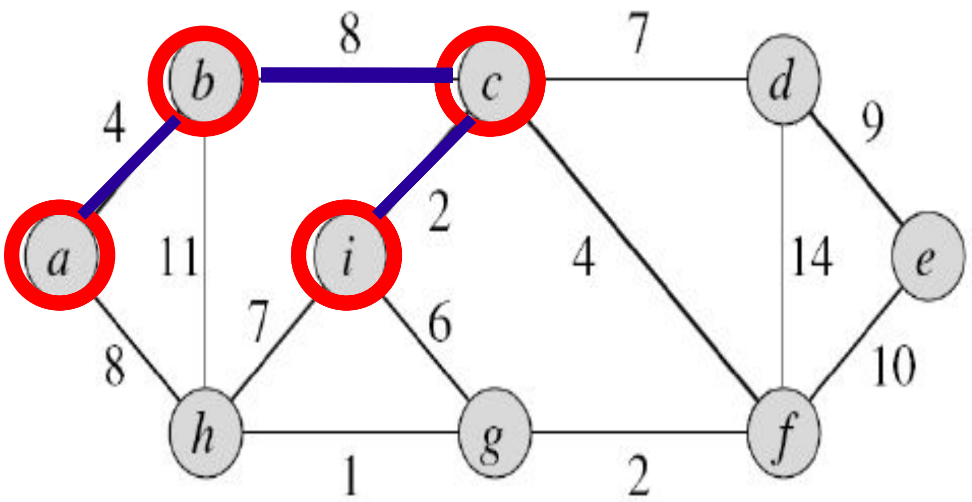
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	∞	4	∞	8	2
π	NIL	a	b	c	NIL	c	NIL	a	c
Q				✓	✓	✓	✓	✓	

Algoritmo de Prim

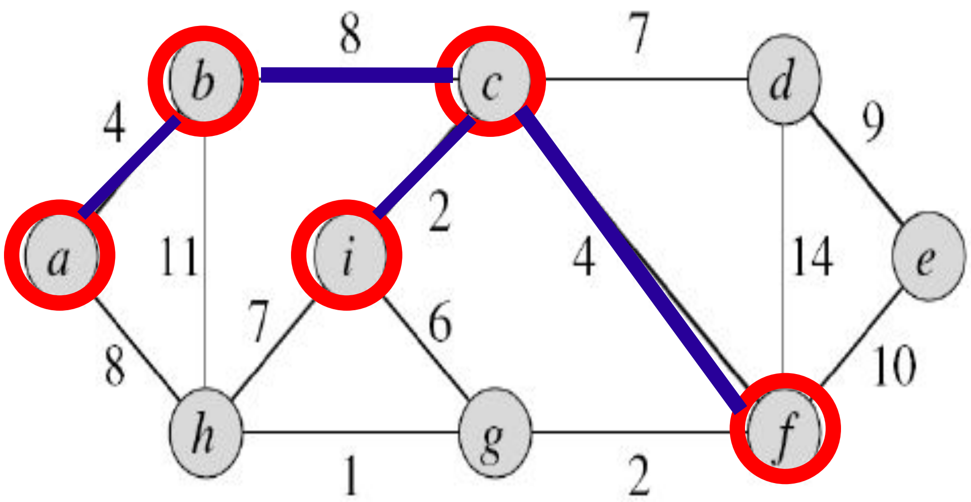
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	∞	4	6	7	2
π	NIL	a	b	c	NIL	c	i	i	c
Q				✔	✔	✔	✔	✔	

Algoritmo de Prim

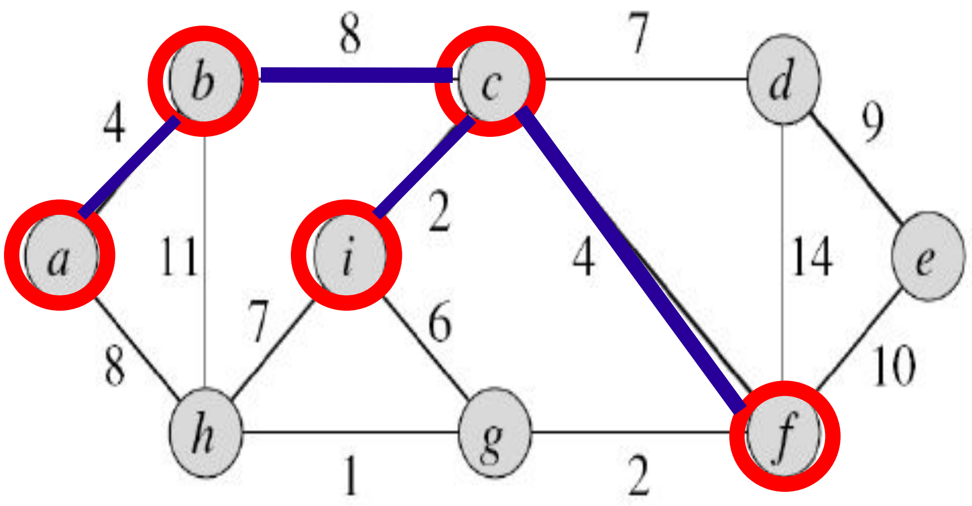
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	∞	4	6	7	2
π	NIL	a	b	c	NIL	c	i	i	c
Q				✔	✔		✔	✔	

Algoritmo de Prim

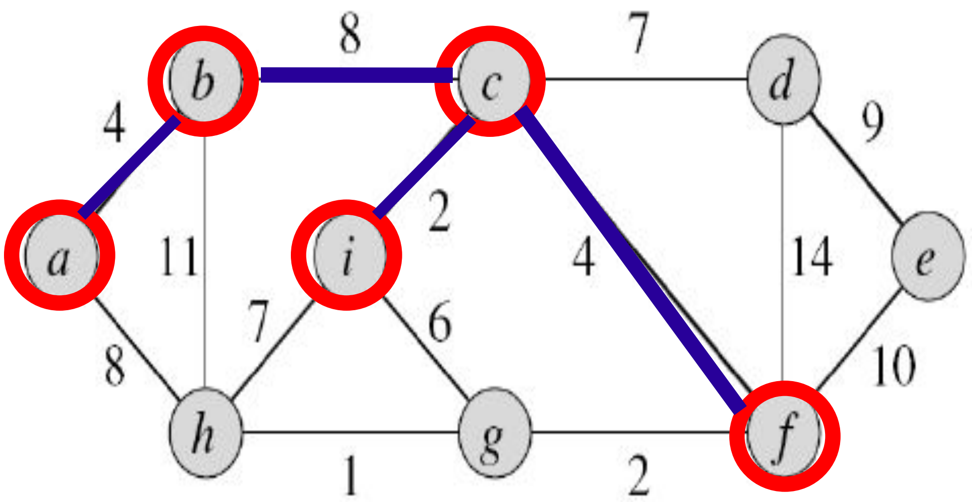
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	∞	4	6	7	2
π	NIL	a	b	c	NIL	c	i	i	c
Q				✓	✓		✓	✓	

Algoritmo de Prim

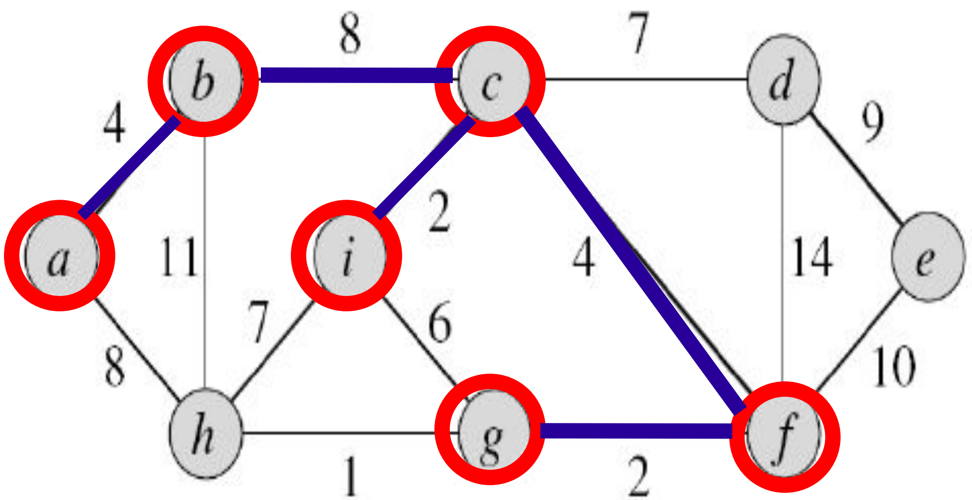
PRIM(V, A, w, r)
...
while Q is not empty
    u ← EXTRACT-MIN(Q)
    for each v in Adj[u]
        if v in Q and w(u, v) < key[v]
        then π[v] ← u
            DECREASE-KEY(Q, v, w(u, v))



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	10	4	2	7	2
π	NIL	a	b	c	f	c	f	i	c
Q				✔	✔		✔	✔	

Algoritmo de Prim

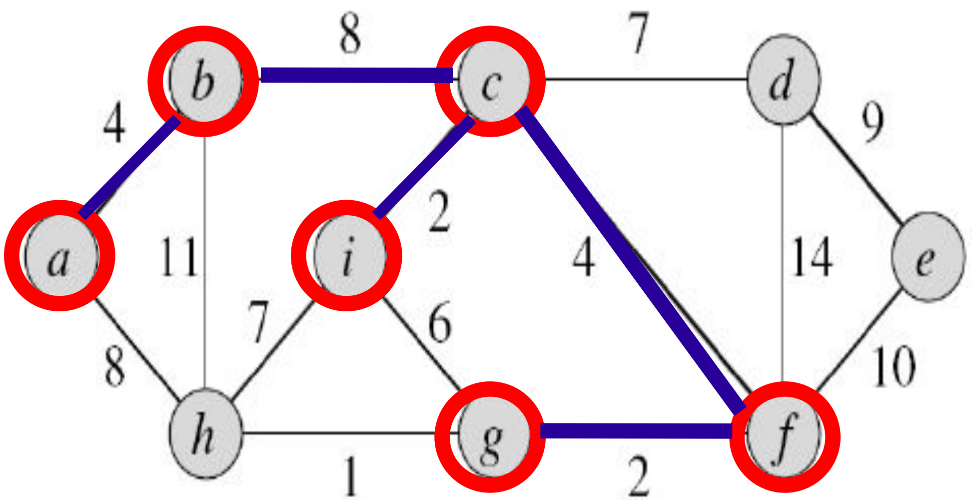
PRIM(V, A, w, r)
...
while Q is not empty
    u ← EXTRACT-MIN(Q)
    for each v in Adj[u]
        if v in Q and w(u, v) < key[v]
        then π[v] ← u
            DECREASE-KEY(Q, v, w(u, v))



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	10	4	2	7	2
π	NIL	a	b	c	f	c	f	i	c
Q				✔	✔			✔	

Algoritmo de Prim

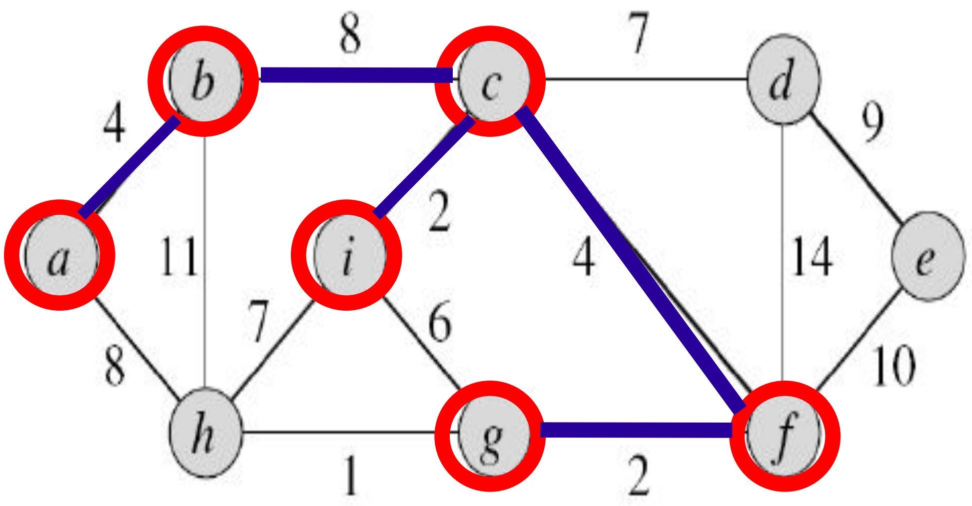
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	10	4	2	7	2
π	NIL	a	b	c	f	c	f	i	c
Q				✔	✔			✔	

Algoritmo de Prim

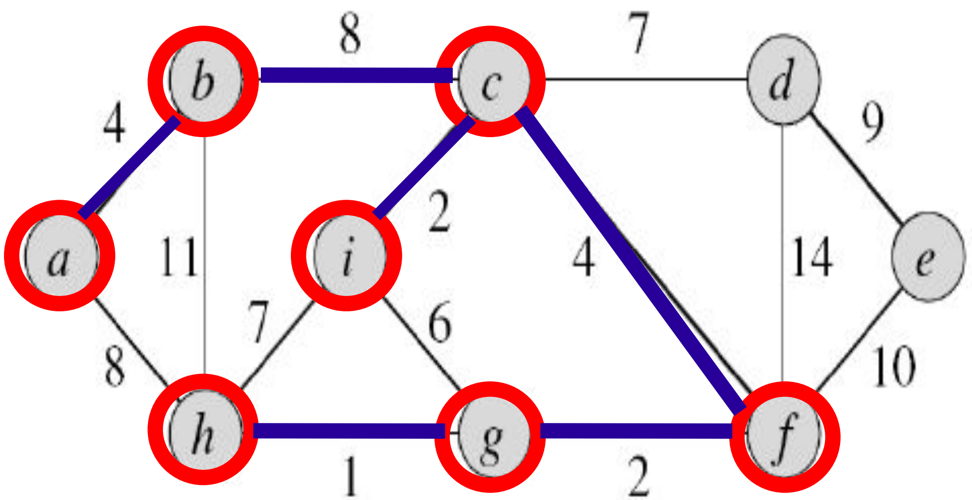
PRIM(V, A, w, r)
...
while Q is not empty
  u ← EXTRACT-MIN(Q)
  for each v in Adj[u]
    if v in Q and w(u, v) < key[v]
    then π[v] ← u
      DECREASE-KEY(Q, v, w(u, v))



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	10	4	2	1	2
π	NIL	a	b	c	f	c	f	g	c
Q				✓	✓			✓	

Algoritmo de Prim

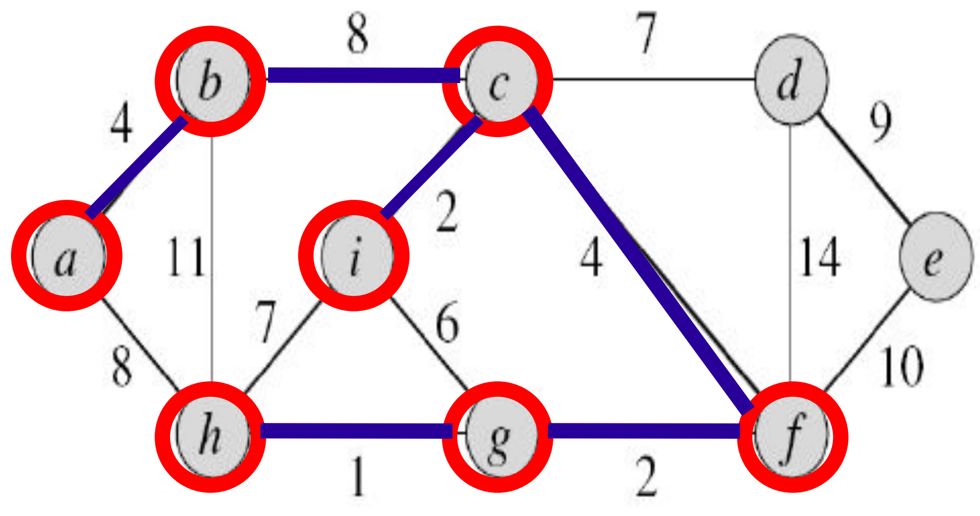
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	10	4	2	1	2
π	NIL	a	b	c	f	c	f	g	c
Q				✔	✔				

Algoritmo de Prim

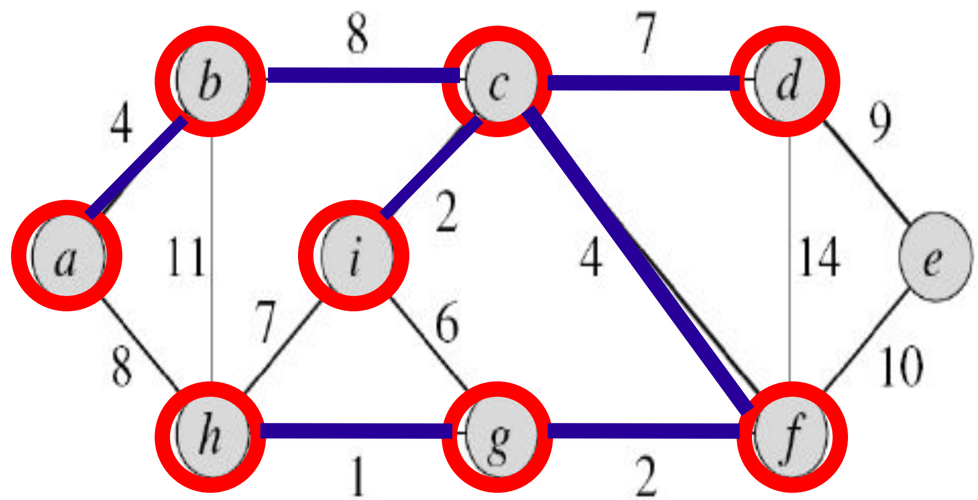
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	10	4	2	1	2
π	NIL	a	b	c	f	c	f	g	c
Q				✔	✔				

Algoritmo de Prim

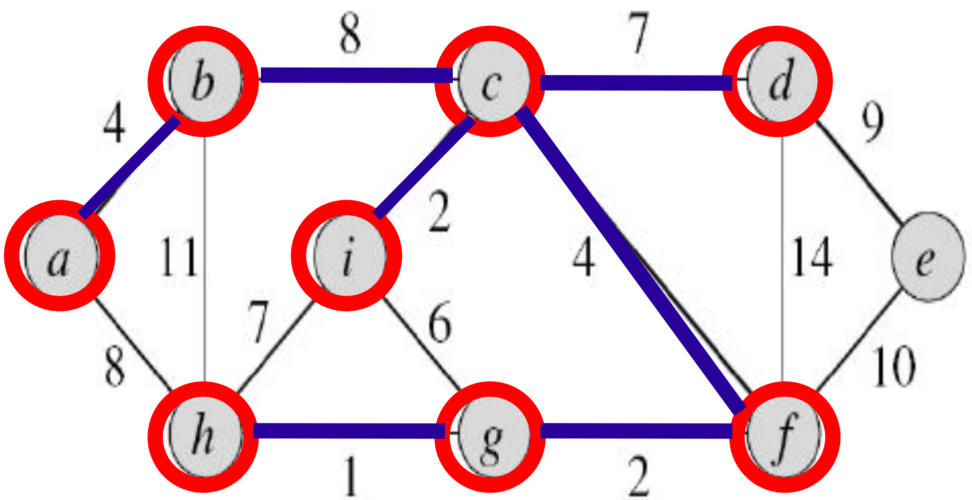
```
PRIM(V, A, w, r )
...
while Q is not empty
        u ← EXTRACT-MIN(Q)
        for each v in Adj[u]
            if v in Q and w(u, v) < key[v]
            then π[v] ← u
                    DECREASE-KEY(Q, v, w(u, v))
```



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	10	4	2	1	2
π	NIL	a	b	c	f	c	f	g	c
Q					✔				

Algoritmo de Prim

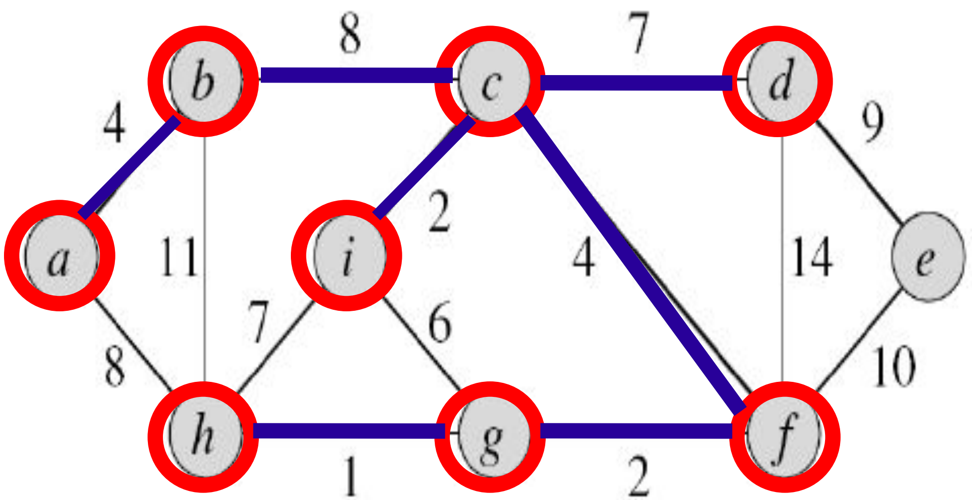
PRIM(V, A, w, r)
...
while Q is not empty
u ← EXTRACT-MIN(Q)
for each v in Adj[u]
if v in Q and w(u, v) < key[v]
then π[v] ← u
DECREASE-KEY(Q, v, w(u, v))



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	10	4	2	1	2
π	NIL	a	b	c	f	c	f	g	c
Q					✔				

Algoritmo de Prim

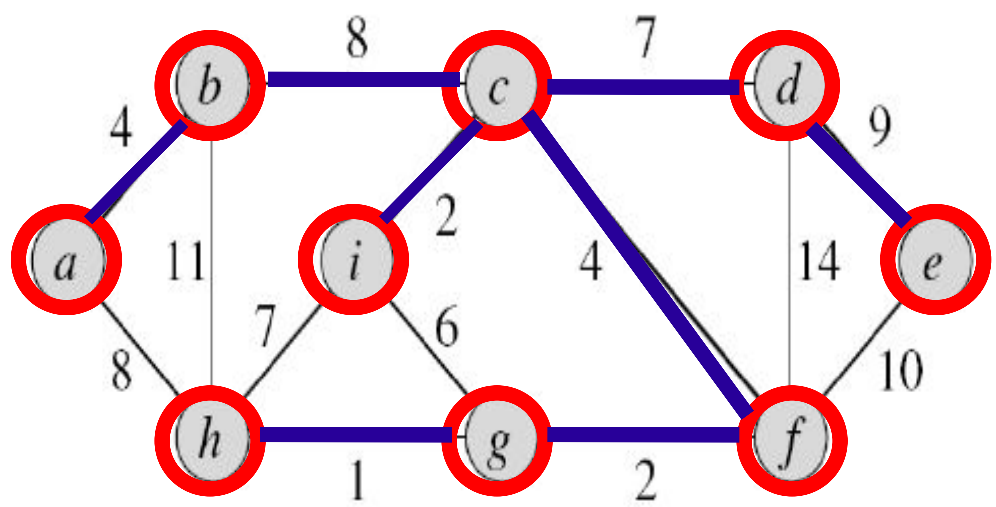
PRIM(V, A, w, r)
...
while Q is not empty
  u ← EXTRACT-MIN(Q)
  for each v in Adj[u]
    if v in Q and w(u, v) < key[v]
    then π[v] ← u
      DECREASE-KEY(Q, v, w(u, v))



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	9	4	2	1	2
π	NIL	a	b	c	d	c	f	g	c
Q					✔				

Algoritmo de Prim

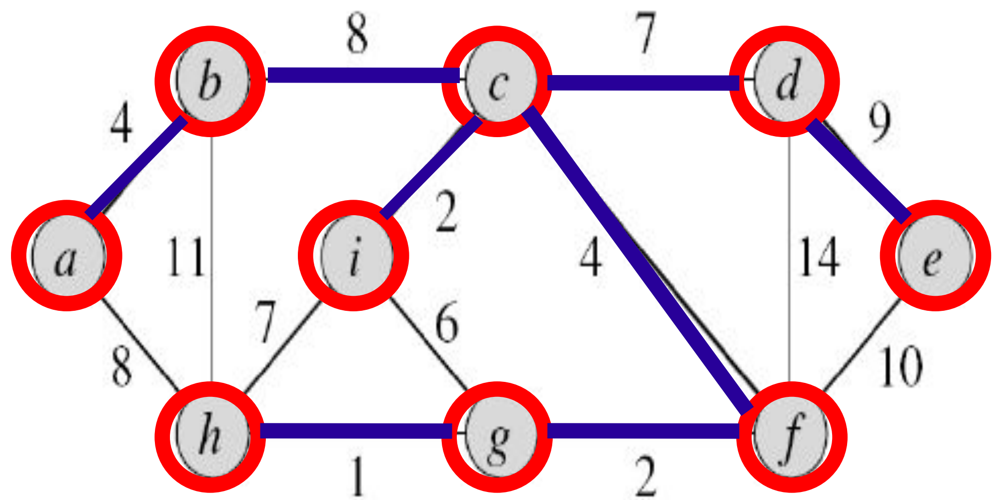
PRIM(V, A, w, r)
...
while Q is not empty
    u ← EXTRACT-MIN(Q)
    for each v in Adj[u]
        if v in Q and w(u, v) < key[v]
        then π[v] ← u
            DECREASE-KEY(Q, v, w(u, v))



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	9	4	2	1	2
π	NIL	a	b	c	d	c	f	g	c
Q									

Algoritmo de Prim

PRIM(V, A, w, r)
...
while Q is not empty
    u ← EXTRACT-MIN(Q)
    for each v in Adj[u]
        if v in Q and w(u, v) < key[v]
        then π[v] ← u
            DECREASE-KEY(Q, v, w(u, v))



vértice	a	b	c	d	e	f	g	h	i
chave	0	4	8	7	9	4	2	1	2
π	NIL	a	b	c	d	c	f	g	c
Q									

Algoritmo de Prim (Jarnik)

```
PRIM(V, A, w, r )

Q ← { }
for each u in V
        key[u] ← ∞
        π[u] ← NIL
        INSERT(Q, u)
DECREASE-KEY(Q, r, 0)    ▷ key[r ] ← 0
while Q is not empty
       u ← EXTRACT-MIN(Q)
       for each v in Adj[u]
                 if v in Q and w(u, v) < key[v]
                 then π[v] ← u
                       DECREASE-KEY(Q, v, w(u, v))
```

Exercício

Existem 8 ilhas em um lago e deseja-se construir sete pontes para conectá-las de forma que cada ilha possa ser alcançada a partir de cada outra. O custo de construir uma ponte é proporcional ao seu comprimento. As distâncias entre os pares de ilhas são dados na seguinte tabela:

-	240	210	340	280	200	345	120
-	-	265	175	215	180	185	155
-	-	-	260	115	350	435	195
-	-	-	-	160	330	295	230
-	-	-	-	-	360	400	170
-	-	-	-	-	-	175	205
-	-	-	-	-	-	-	305
-	-	-	-	-	-	-	-

Quais pontes devem ser construídas para minimizar o custo total de construção? Qual é esse custo?