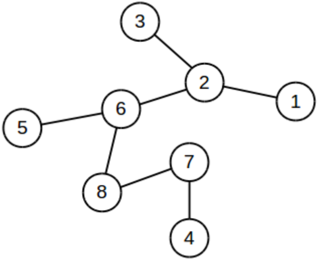
Problema da Árvore Geradora Mínima

Prof. Fábio Dias

20 de janeiro de 2025

Definições

- Uma árvore é um grafo conexo e acíclico;



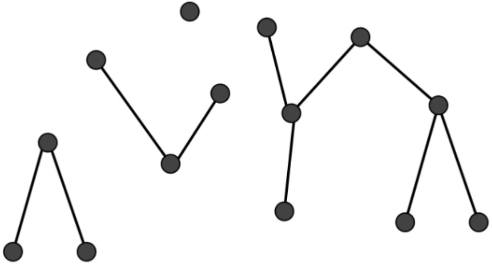
- Conexo: Para qualquer par de vértices do grafo, existe pelo menos um caminho ligando esses dois vértices.
- Acíclico: Esse caminho é único.

Resultados Importantes

- Em toda árvore, temos que $m = n - 1$, onde $n = |V|$ e $m = |E|$;
- Dado uma árvore T, ao adicionarmos uma nova aresta em T, criamos um ciclo. Podemos criar uma nova árvore ao removermos uma aresta do ciclo, diferente da aresta adicionada.
- Se removermos uma aresta da árvore, ela deixa de ser conexa.
- Ou seja, uma árvore é um grafo onde os vértices estão minimamente conectados.

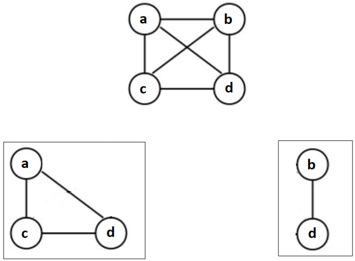
Definições

- Uma floresta é um grafo acíclico, ou seja, é um conjunto de árvores.



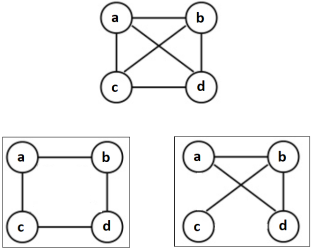
Definições

- Um subgrafo $G' = (V', E')$ de um grafo $G = (V, E)$ é uma parte do grafo, ou seja, $V' \subseteq V$ e $E' \subseteq E$:



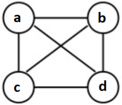
Definições

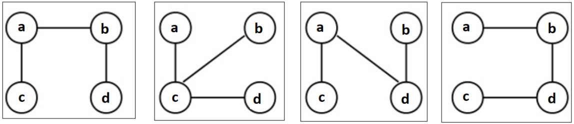
- Um subgrafo $G' = (V', E')$ é gerador quando $V' = V$ e $E' \subseteq E$:



Definições

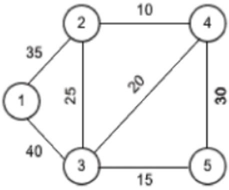
- Uma árvore geradora é um subgrafo gerador que é uma árvore:

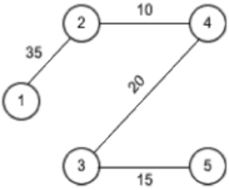




Problema da Árvore Geradora Mínima

- Seja um grafo $G = (V, E)$ conexo e ponderado, ou seja, para cada aresta $(u, v) \in E$ temos um valor associado $w(u, v)$.
- Dado uma árvore geradora T de G, temos que o peso de será $w(T) = \sum_{(u,v) \in T} w(u, v)$.





- No problema da árvore geradora mínima, deseja-se encontrar a árvore geradora T de peso mínimo.

Aplicações

- No mercado de ações, gestão de risco diz respeito ao gerenciamento do nível de risco de sua carteira de investimento.
- Uma das forma de fazer a gestão de risco é através da diversificação da carteira: Não colocar os ovos numa mesma cesta.
- Além disso, varias métrica são usadas para avaliar a correlação dos ativos da carteira. Carteira com baixa correlação possui um risco baixo.
- As ações são os vértices e as arestas ligam as ações com peso sendo a correlação entre as duas ações.

Aplicações

- Instalação de fibras óticas entre os prédios dos Campus de uma universidade.
- Cada trecho de fibra ótica entre os prédios possui um custo associado para sua implantação.
- Desejamos conectar todos os prédios sem redundância com a finalidade de diminuir os custos.
- Os prédios são os vértices e as arestas ligam o projeto de ligação entre dois prédios sendo o peso o seu custo.

Algoritmos

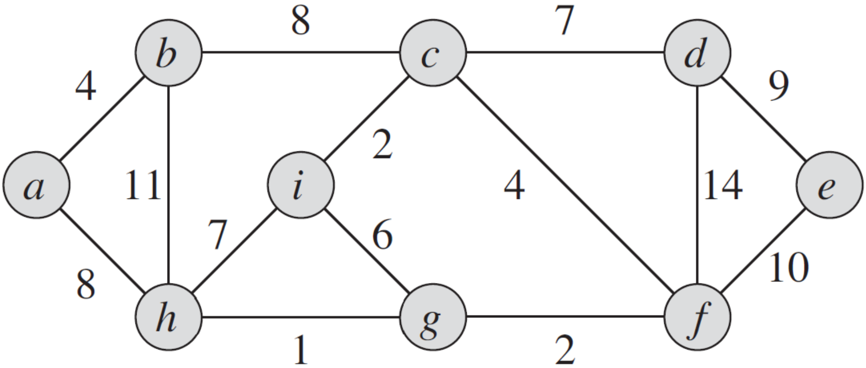
- Podemos gerar todas as árvores geradoras e selecionar a de menor peso (Algoritmo de Força Bruta).
- Dado um grafo completo, o número de árvores geradoras distintas é n^{n-2} (Fórmula de Cayley).
- Lembram da analise assintótica, estamos interessados em valores grande da entrada.
- Precisamos de algoritmos com complexidade polinomial.
- Iremos ver dois algoritmos que resolve o problema com tempo polinomial: Algoritmo Kruskal e Prim;

Algoritmo Kruskal

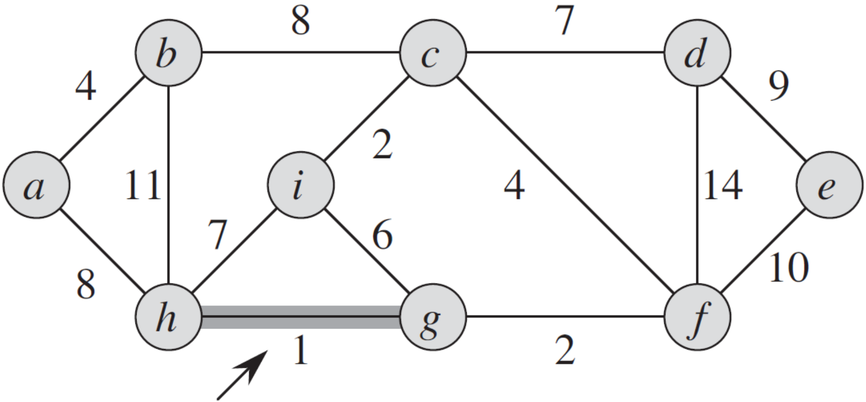
Ideia do Algoritmo Kruskal

- É uma algoritmo construtivo: Gera a solução do problema de forma iterativa, onde a cada iteração, uma parte da solução é selecionada.
- Para o problema da árvore geradora mínima, ele começa com todos os vértices e sem nenhuma aresta (subgrafo gerador sem aresta).
- De forma iterativa, seleciona uma aresta para ser adicionada a solução.
    - **Que não gere ciclo com as arestas já adicionadas.**
    - **Aresta de menor custo.**
- Selecionando $n - 1$ arestas.

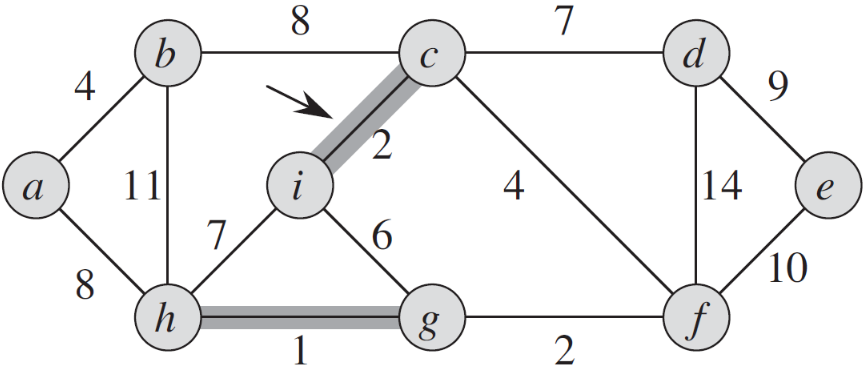
Algoritmo Kruskal



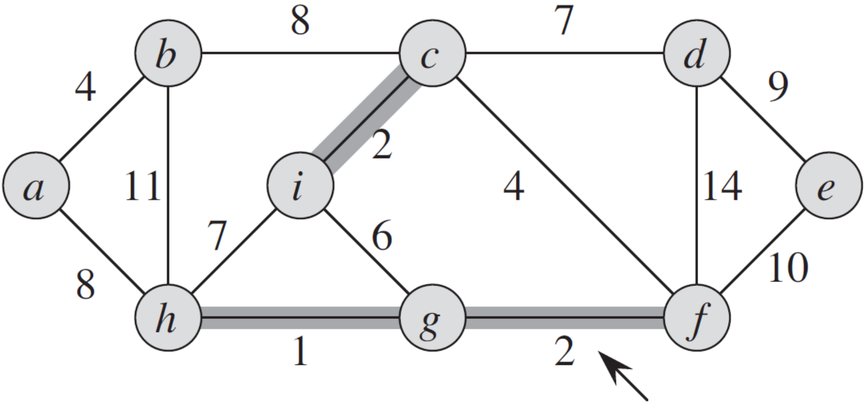
Algoritmo Kruskal



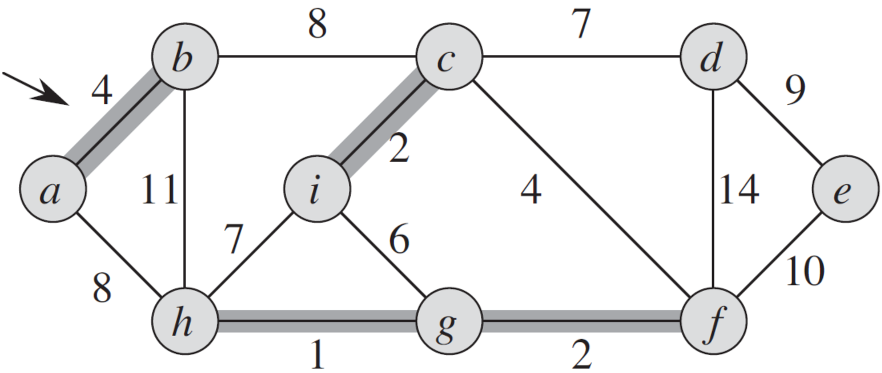
Algoritmo Kruskal



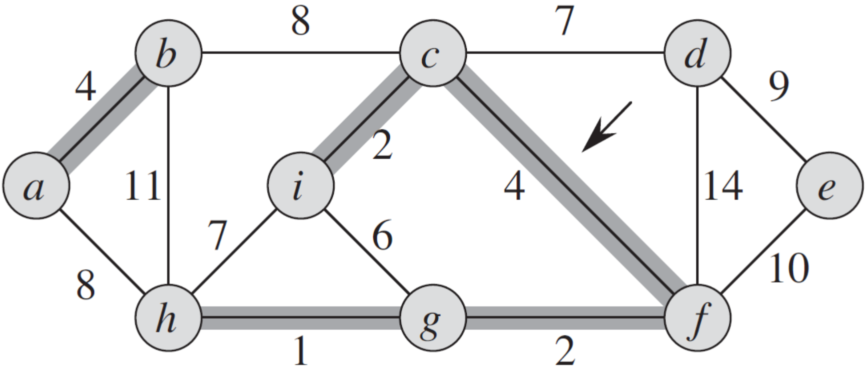
Algoritmo Kruskal



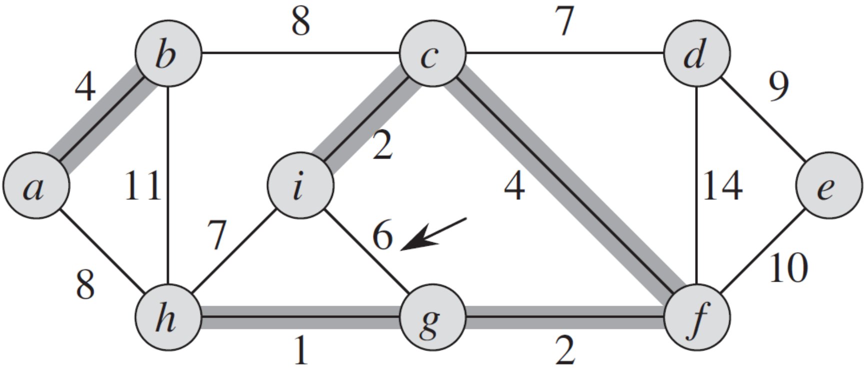
Algoritmo Kruskal



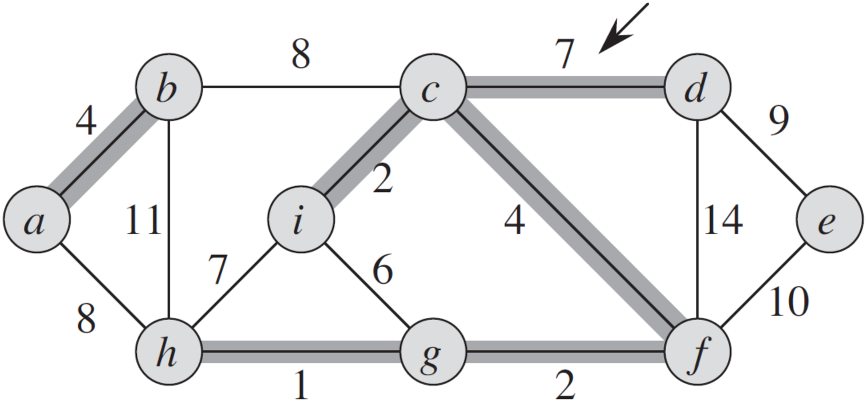
Algoritmo Kruskal



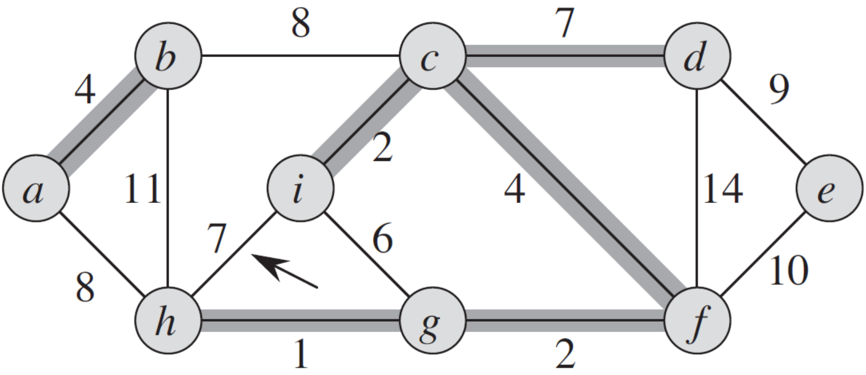
Algoritmo Kruskal



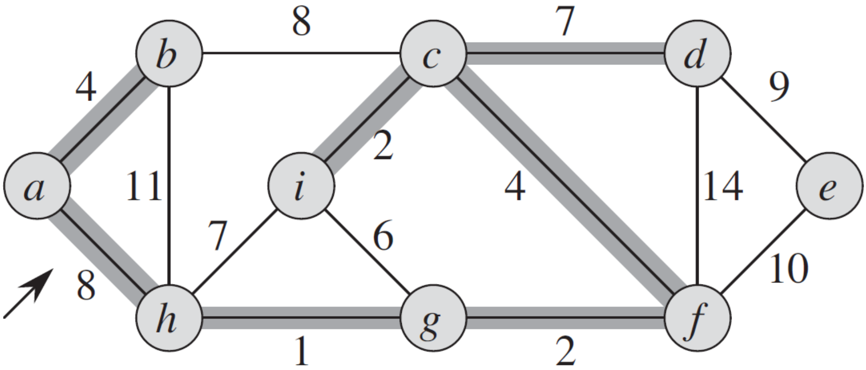
Algoritmo Kruskal



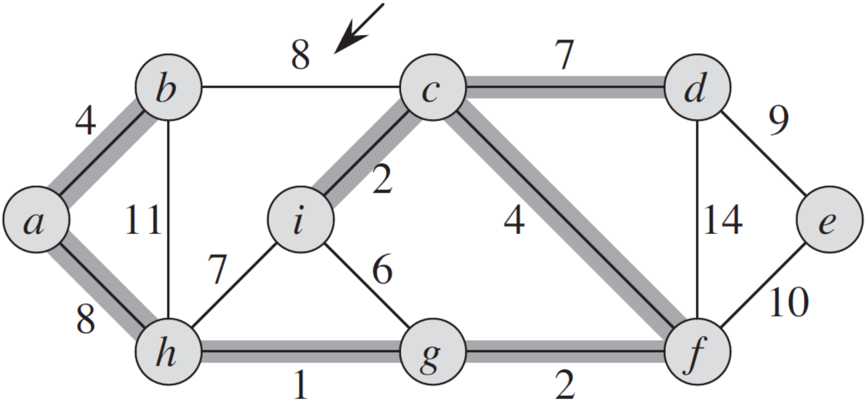
Algoritmo Kruskal



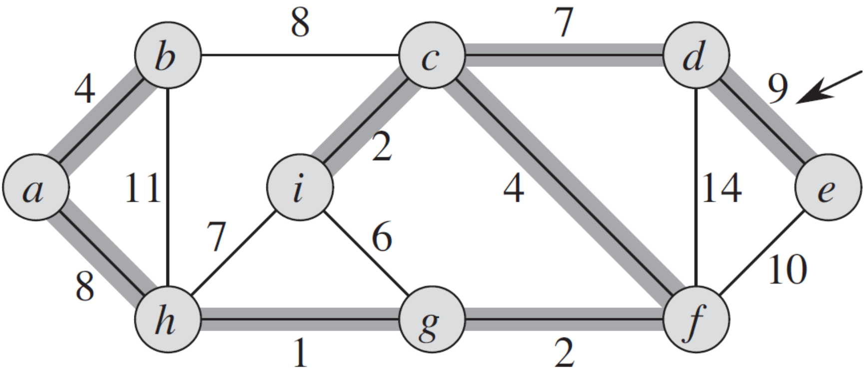
Algoritmo Kruskal



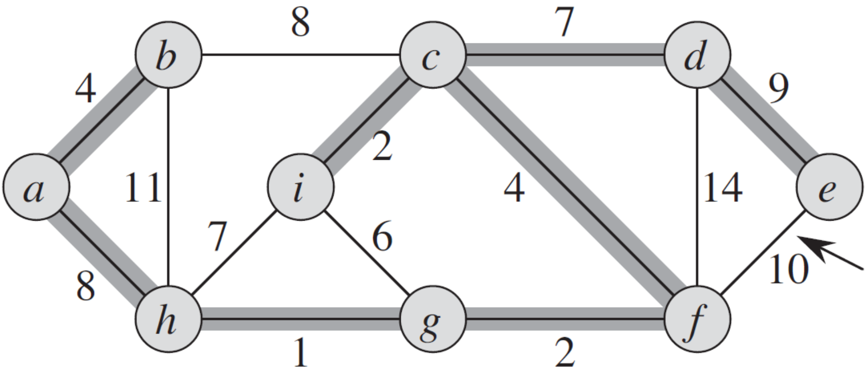
Algoritmo Kruskal



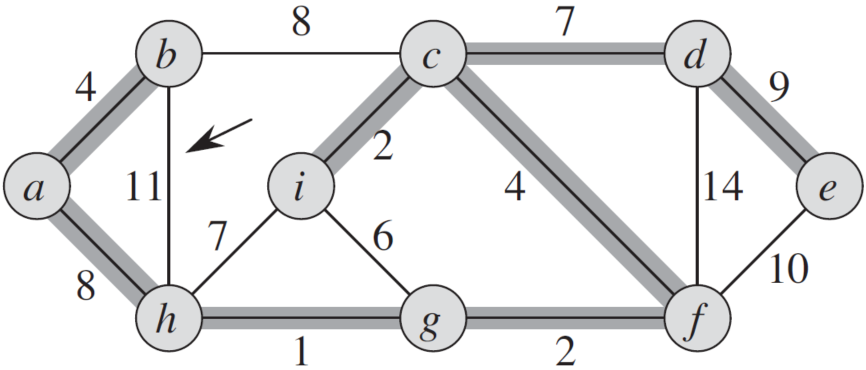
Algoritmo Kruskal



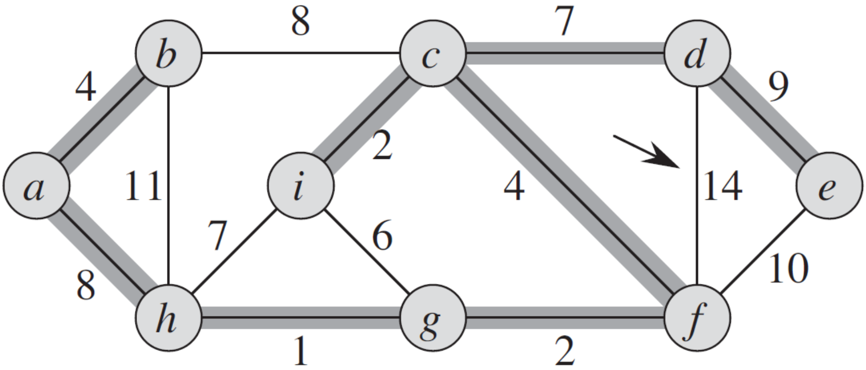
Algoritmo Kruskal



Algoritmo Kruskal



Algoritmo Kruskal



Algoritmo Kruskal

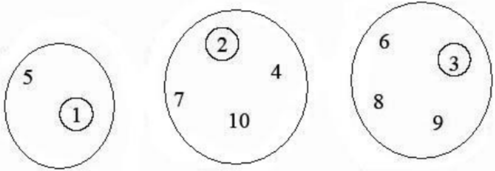
Entrada: Grafo conexo $G = (V, E)$ e função de peso w.
Saída: Árvore Geradora Mínima

$T = \emptyset$;
Ordene as arestas E em ordem crescente de peso w;
para *cada aresta* $(u, v) \in E$ **faça**
    se $(u.v)$ *não gera ciclo com as arestas de* T **então**
        $T = T \cup (u, v)$;
    fim se
fim para
return T;

- Como saber se a aresta (u, v) gera ciclo com as arestas de T?????

Estrutura de dados para Conjuntos Disjuntos

- Em matemática, dois conjuntos são ditos disjuntos se não tiverem nenhum elemento em comum.
- Essa ED é usada para representar situação que precisamos saber se dois elementos pertencem ao mesmo conjunto disjuntos.
- Uma estrutura de dados conjuntos-disjuntos é uma coleção $S = S_1, ..., S_k$ de conjuntos dinâmicos disjuntos.
- Cada conjunto S_k é identificado por um representante, que é um membro do conjunto.
- Tipicamente não importa quem é o representante.



	1	2	3	4	5	6	7	8	9	10
C	1	2	3	2	1	3	2	3	3	2

Estrutura de dados para Conjuntos Disjuntos

- $Make - Set(x)$: Cria um novo conjunto cujo único elemento é apontado por x. x não pode pertencer a outro conjunto da coleção.
- $Union(x, y)$: Executa a união dos conjuntos que contêm x e y, digamos S_x e S_y, em um conjunto único.
- $Find - Set(x)$: Retorna o representante do conjunto que contém x.

Algoritmo Kruskal

```
MST-KRUSKAL(G, w)
    A = ∅
    for cada vértice v ∈ G.V
        MAKE-SET(v)
    ordene as arestas de G.E em ordem não decrescente de peso w
    for cada aresta (u, v) ∈ G.E, tomada em ordem não decrescente de peso
        if FIND-SET(u) ≠ FIND-SET(v)
            A = A ∪ {(u, v)}
            UNION(u, v)
    return A
```

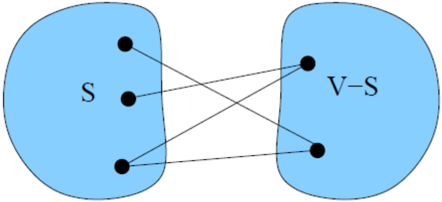
Complexidade do Kruskal

- Depende da implementação da estrutura de dados Conjuntos Disjuntos;
- Linha 2-3 custo $O(n)$;
- Ordenação consome $O(m \log m)$;
- Cada operação sobre Conjuntos Disjuntos é da ordem de $O(\log n)$;
- No pior caso teremos m operação sobre o Conjuntos Disjuntos, portanto $O(m \log n)$;
- Como $m < n^2$, temos $\log m = O(\log n)$ e, portanto, o Kruskal tem complexidade da ordem de $O(m \log n)$.

Algoritmo Prim

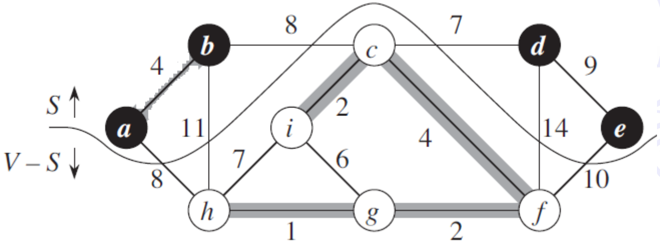
Corte em um Grafo

- Um corte em um grafo $G = (V, E)$ é uma partição de V em $(S, V - S)$.



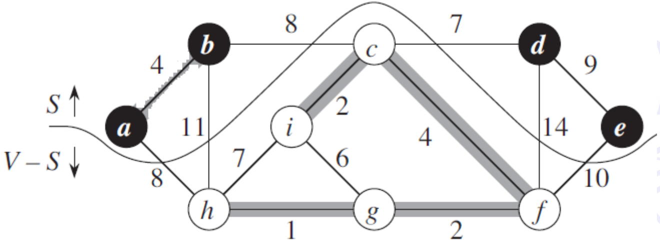
Corte em um Grafo

- Dizemos que uma aresta (u, v) cruza o corte $(S, V - S)$ se $u \in S$ e $v \in V - S$ ou vice-versa;
- Em um grafo conexo, qualquer corte possui pelo menos uma aresta que o cruza. Logo, em uma árvore também.

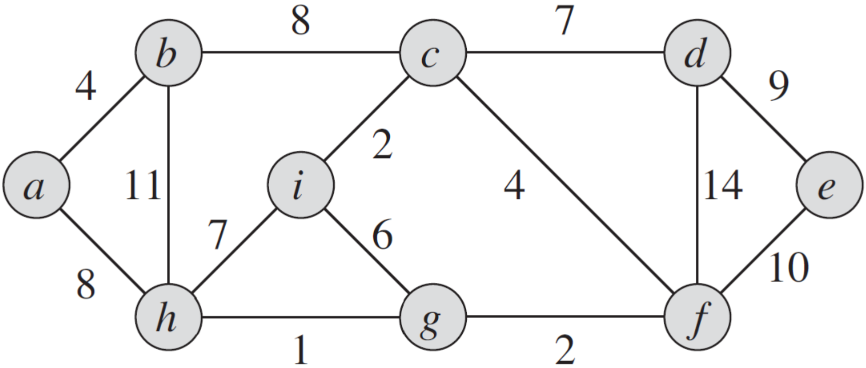


Corte em um Grafo

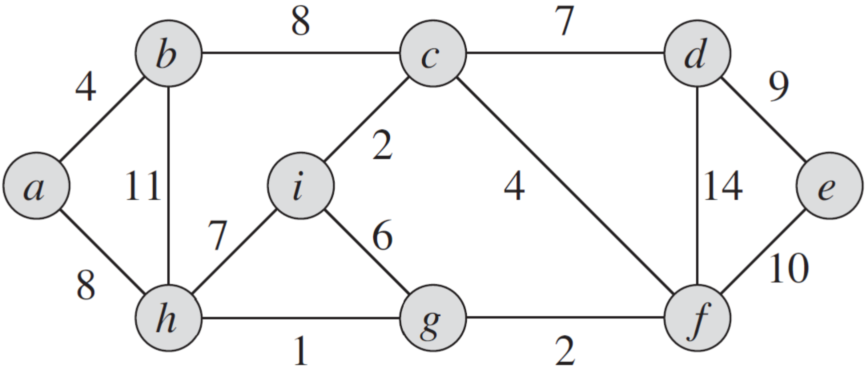
- Uma aresta será leve para um corte, se essa aresta cruza o corte e possui peso mínimo entre todas as arestas que cruzam o corte.



Como Gerar uma Solução Viável???

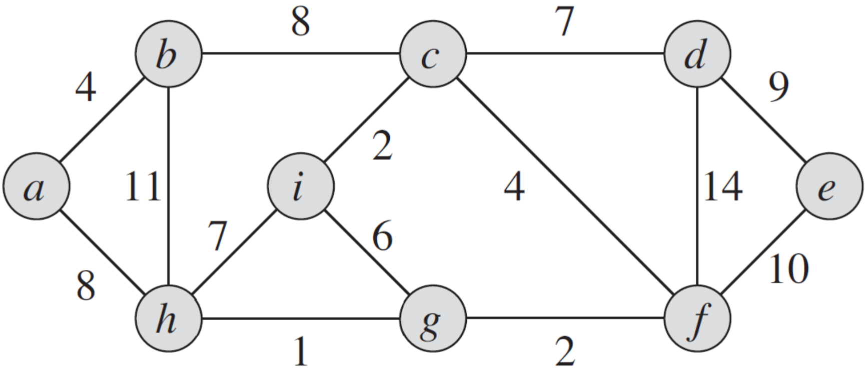


Como Gerar uma Solução Viável???



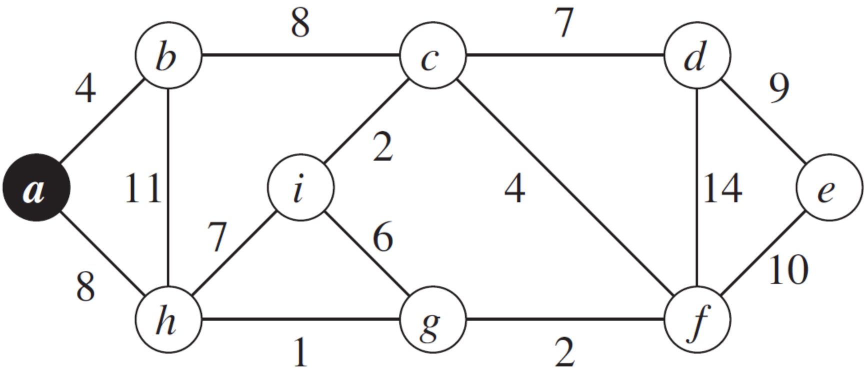
- Partindo de um corte $S = \{q\}$, com $q \in V$ qualquer, selecione uma aresta que cruza um corte e adicione o vértice em S;
- Continue até que $S = V$.

Como Gerar uma Solução Viável???

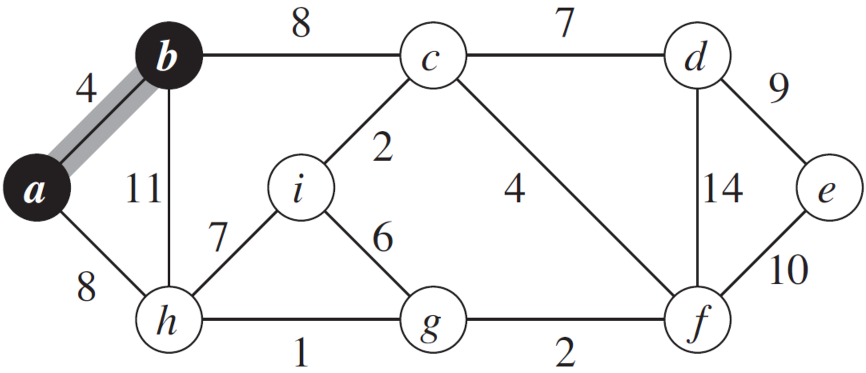


- **Selecione a aresta que cruza o corte S de custo mínimo (aresta leve).**

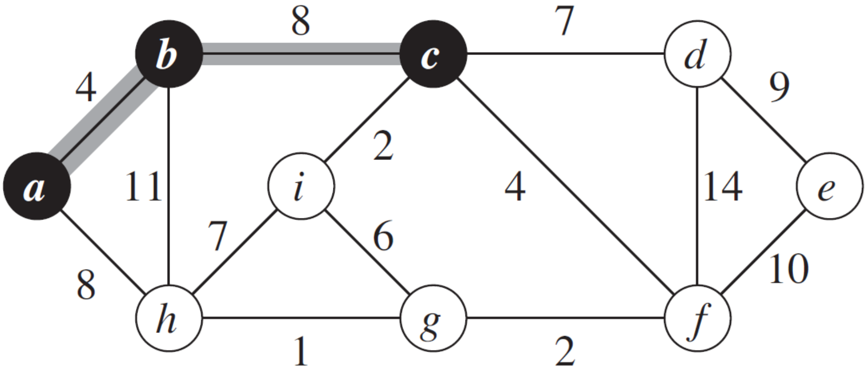
Algoritmo Prim



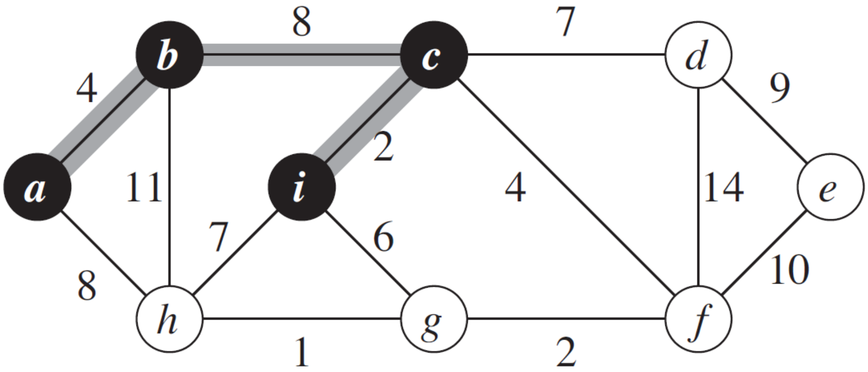
Algoritmo Prim



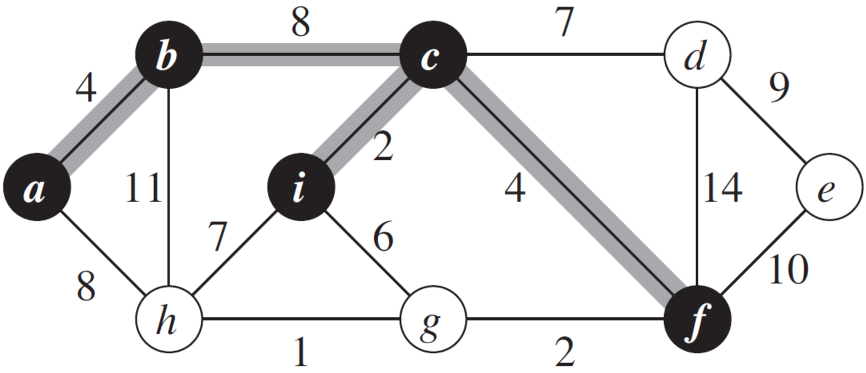
Algoritmo Prim



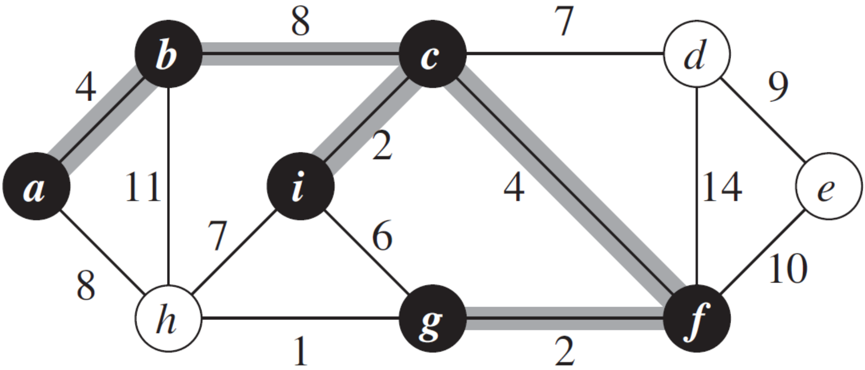
Algoritmo Prim



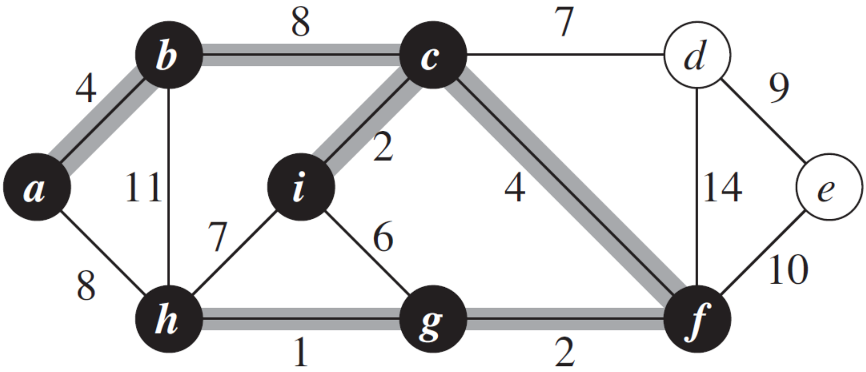
Algoritmo Prim



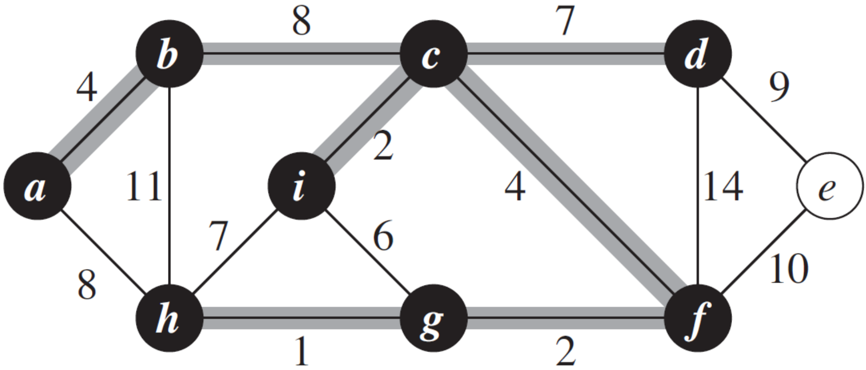
Algoritmo Prim



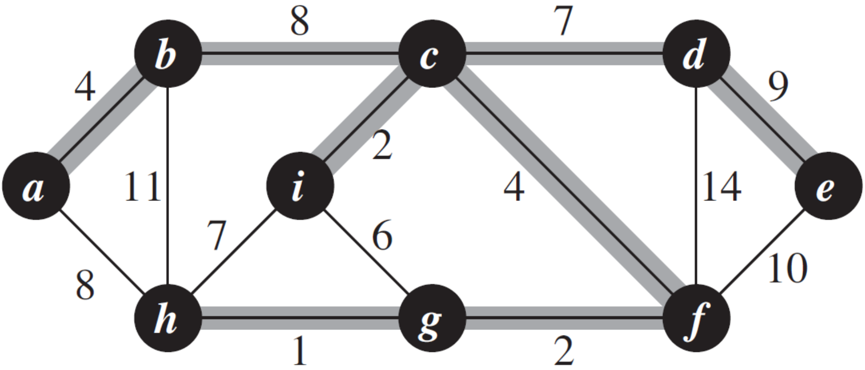
Algoritmo Prim



Algoritmo Prim



Algoritmo Prim



Algoritmo Prim

Entrada: Grafo conexo $G = (V, E)$ e função de peso w.
Saída: Árvore Geradora Mínima

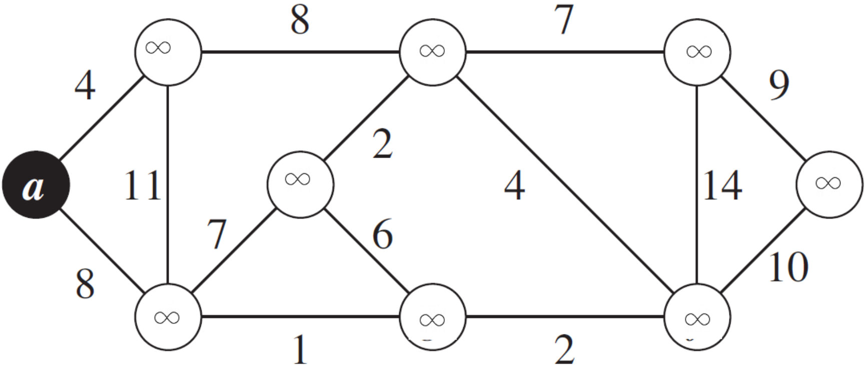
```
S = ∅;
Add um vértice q ∈ V em S;
enquanto |S| < |V| faça
    Selecione a aresta que cruza o corte S de custo mínimo;
    Seja (v, u) essa aresta, tal que v ∈ S e u ∉ S;
    S = S ∪ u;
fim enqto
```

- Como selecionar a aresta leve que cruza o corte?????

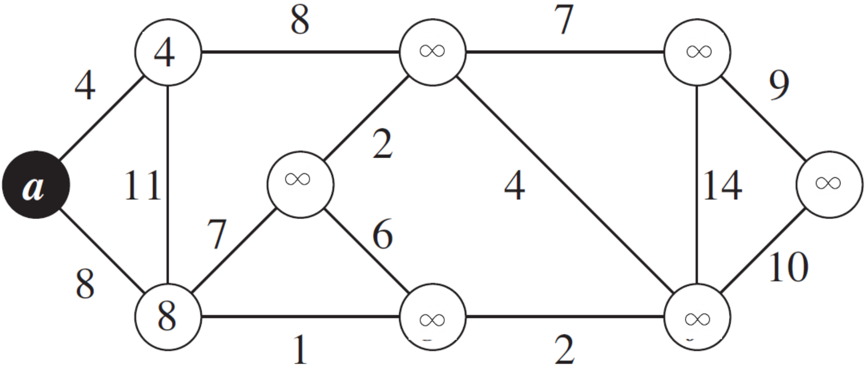
Algoritmo Prim

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

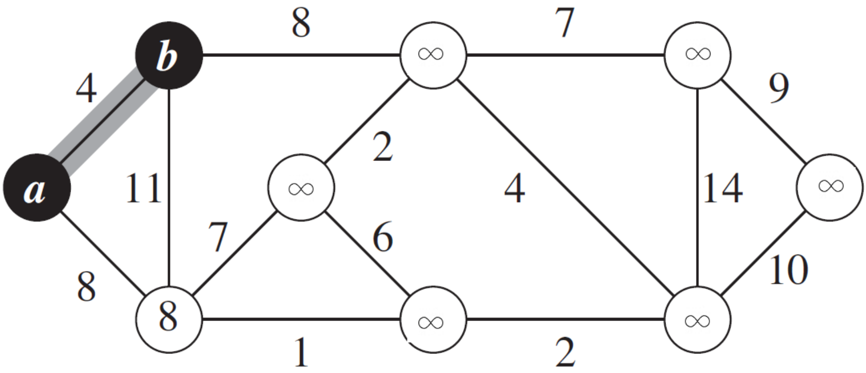
Algoritmo Prim



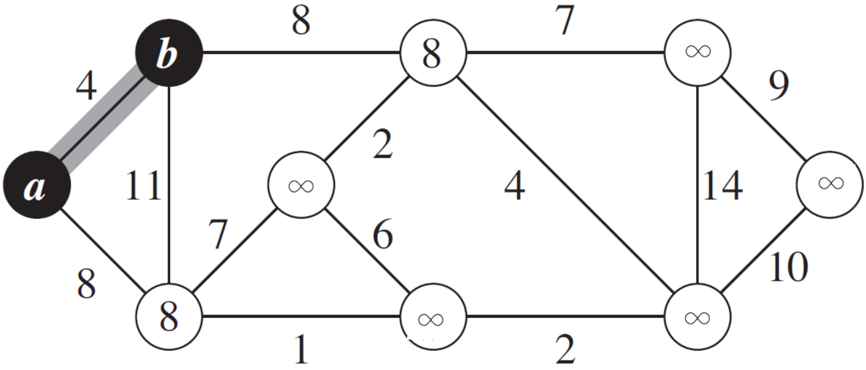
Algoritmo Prim



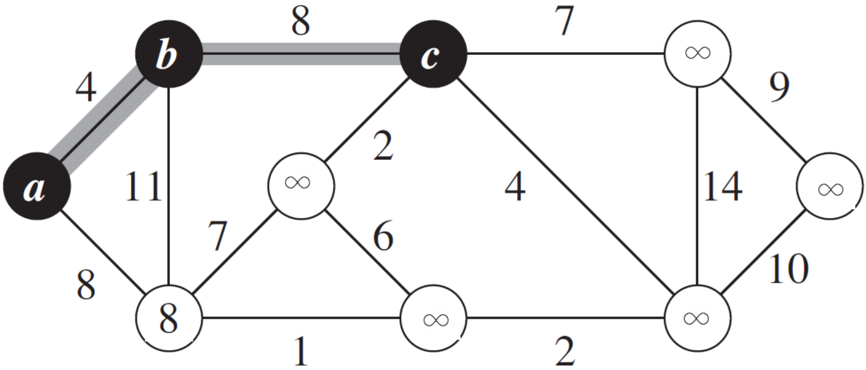
Algoritmo Prim



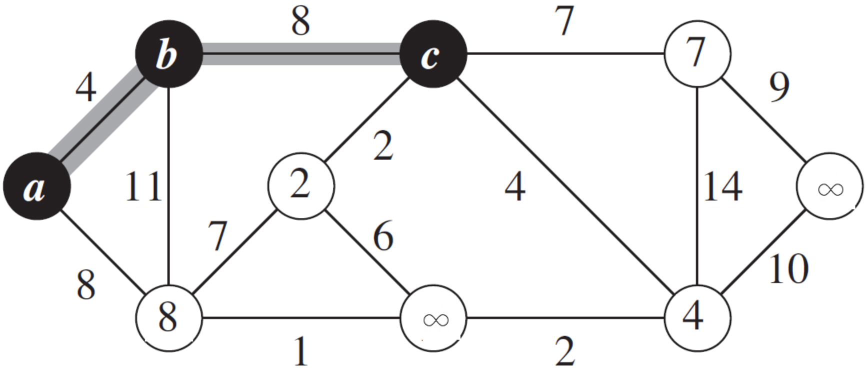
Algoritmo Prim



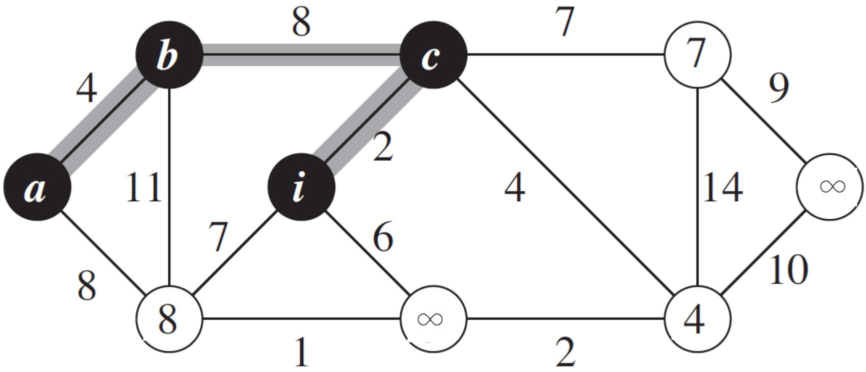
Algoritmo Prim



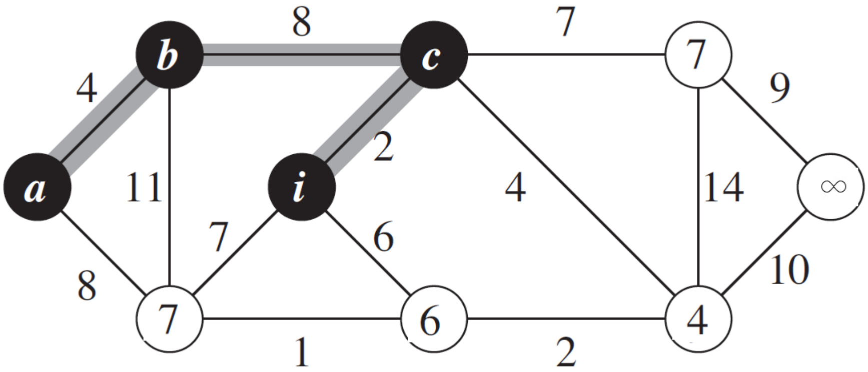
Algoritmo Prim



Algoritmo Prim



Algoritmo Prim



Complexidade do Prim

- Depende da implementação da Fila de Prioridade. Usando Heap Binário;
- Linha 1-5 custo $O(n)$;
- O While executa n vezes. Como *Extract-Min* tem custo $O(\log n)$, teremos custo total de $O(n \log n)$;
- O For executa m vezes. Como alteração da lista de prioridade tem custo $O(\log n)$, teremos custo total de $O(m \log n)$;
- Logo, o Prim tem complexidade da ordem de $O(n \log n + m \log n) \in O(m \log n)$.