

Algoritmos e Estruturas de Dados II

Árvores geradoras mínimas

Prof. Flávio José M. Coelho

fcoelho@uea.edu.br

Objetivos

Entender o que é uma **árvore geradora mínima**

Entender como funciona o algoritmo de **Prim**

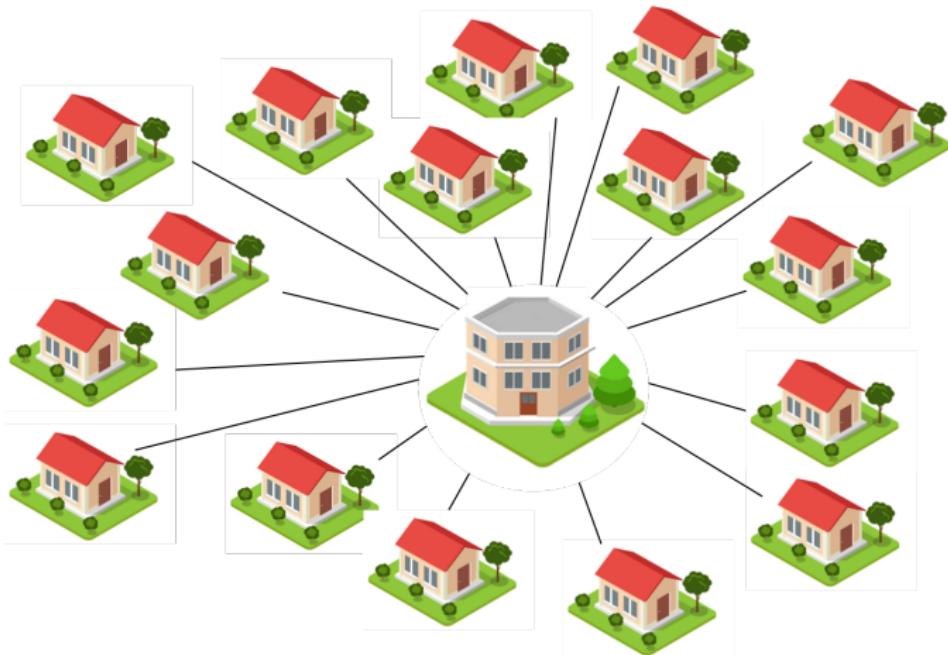
Entender como funciona o algoritmo de **Kruskal**

Árvores geradoras mínimas

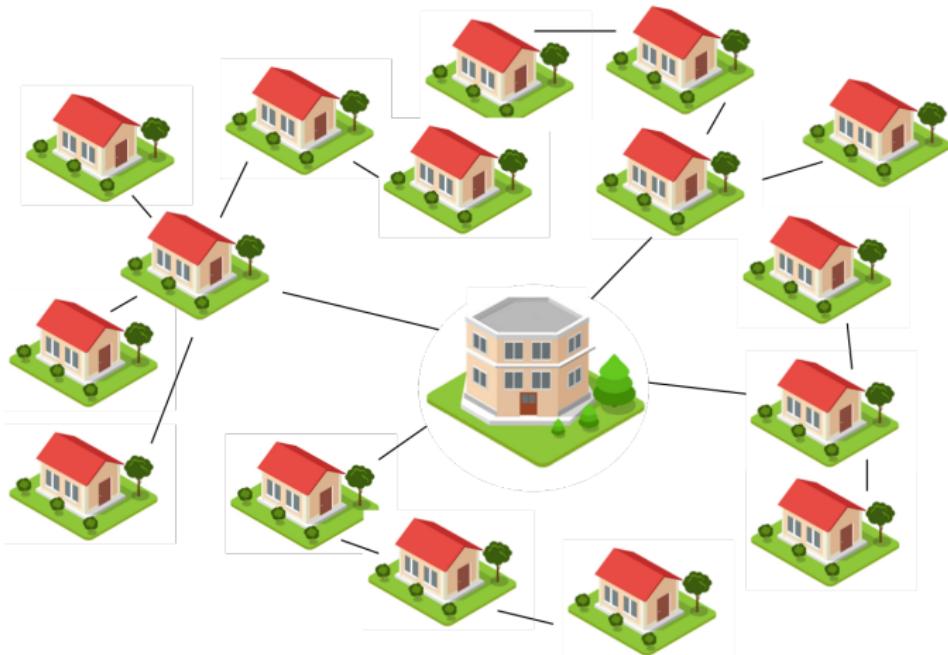
Problema. Uma empresa de TV a cabo precisa levar um cabo de transmissão para residências de um condomínio.



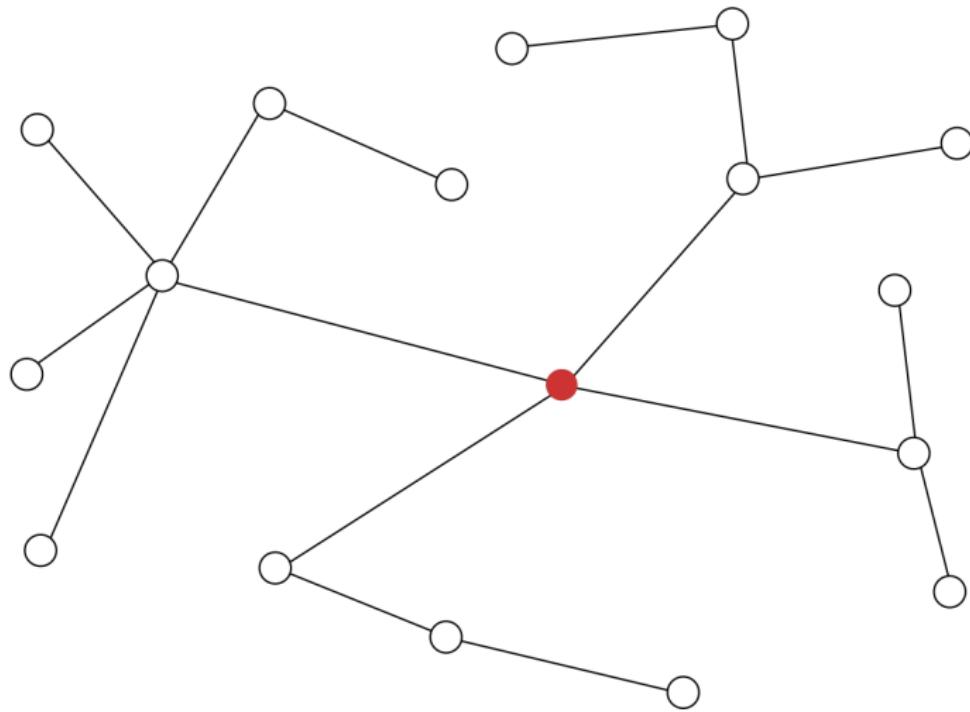
Solução ingênua



Solução racional

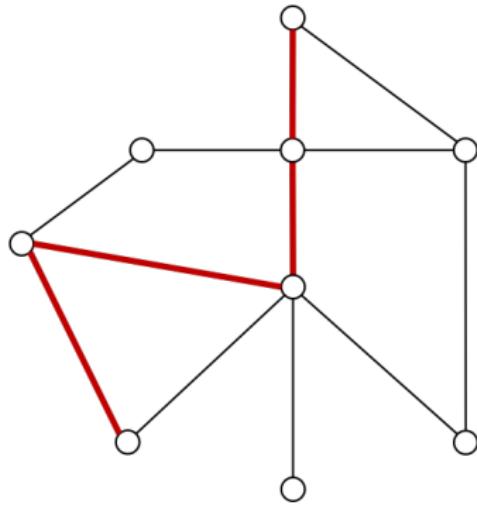


Solução racional (modelada com grafo)

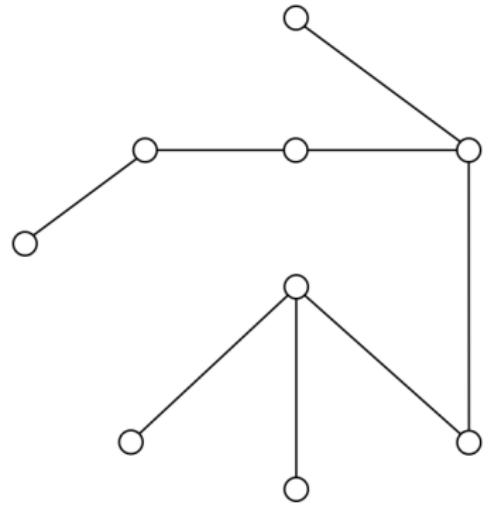


Uma **Árvore Geradora** (*Spanning Tree*) T de um grafo G é um subgrafo de G tal que:

1. T é uma árvore (acíclico e conexo).
2. T conecta todos os vértices de G .

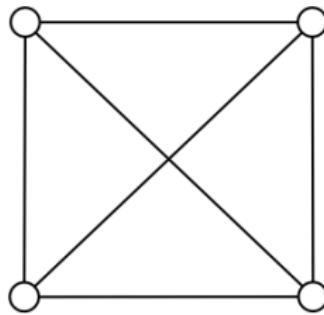


Grafo G

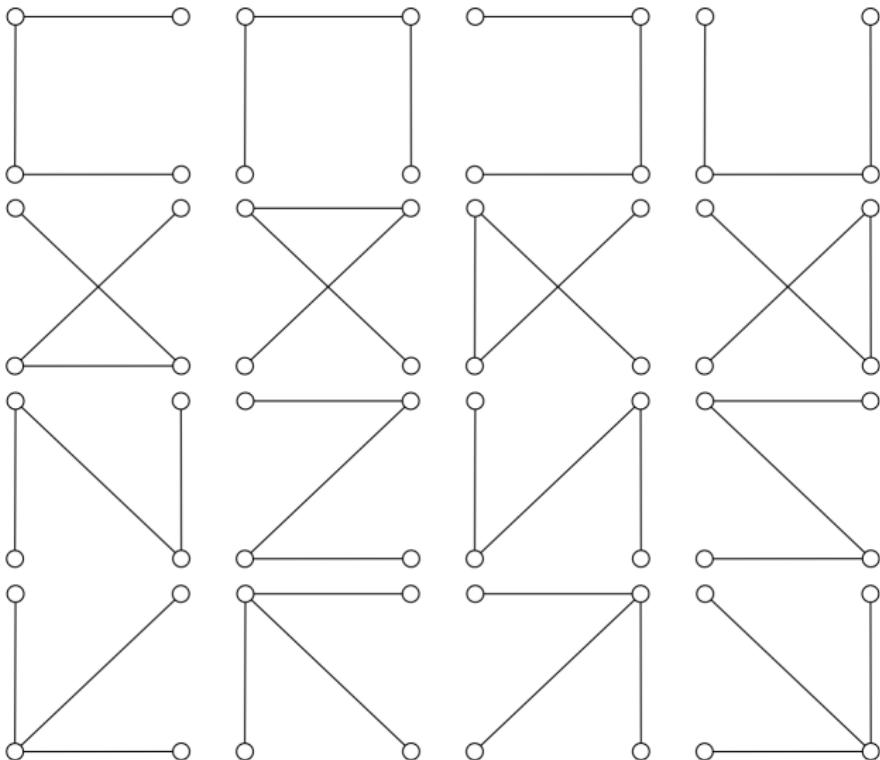


Uma árvore geradora de G

Questão: quantas árvores geradoras tem o K_4 ?

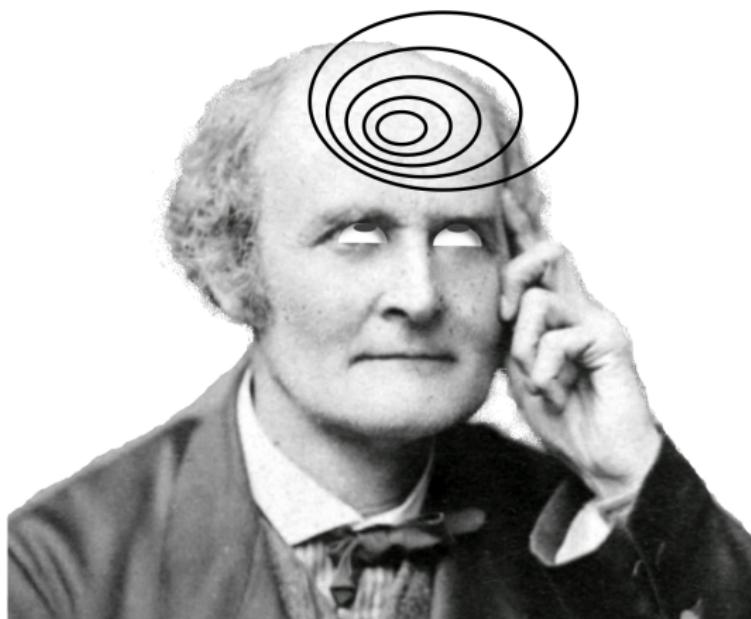


$$K_4$$



K_4 tem 16 árvores geradoras

“Em 1899, um matemático mutante, avô do prof. Charles Xavier... Fez uma descoberta...”



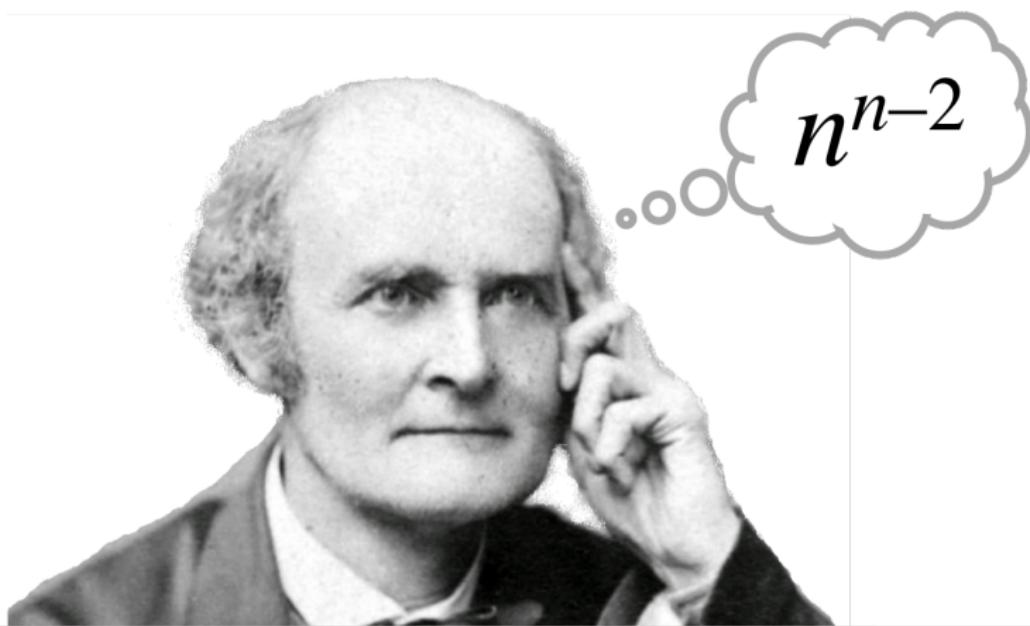
Arthur Cayley
(1821 - 1895)

“...Que derrubou governos ao redor do mundo,
e um disco voador em Roswell!”

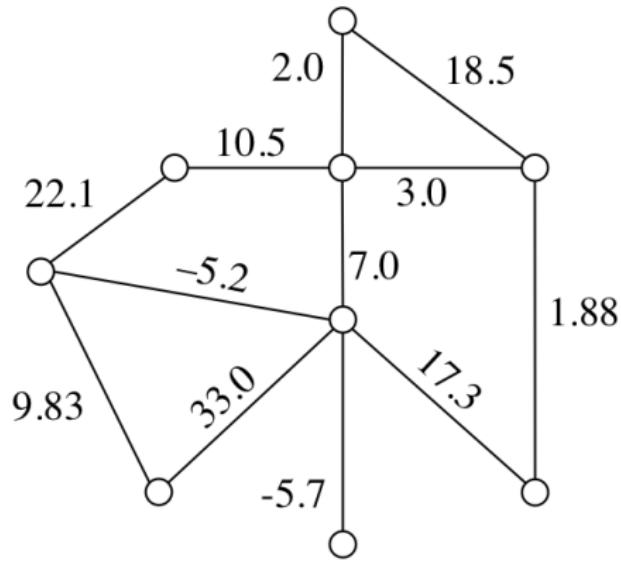


Arthur Cayley
(1821 - 1895)

Teorema (Cayley, 1889). Há n^{n-2} árvores geradoras sobre um grafo completo com n vértices.



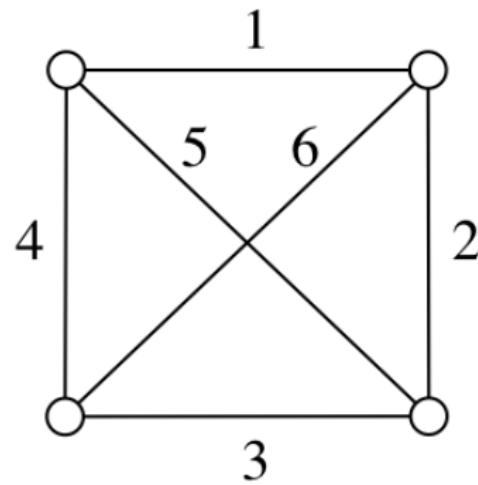
Seja um grafo ponderado $G = (V, E)$ com uma função $w : E \rightarrow \mathbb{R}$ de pesos sobre seus vértices.

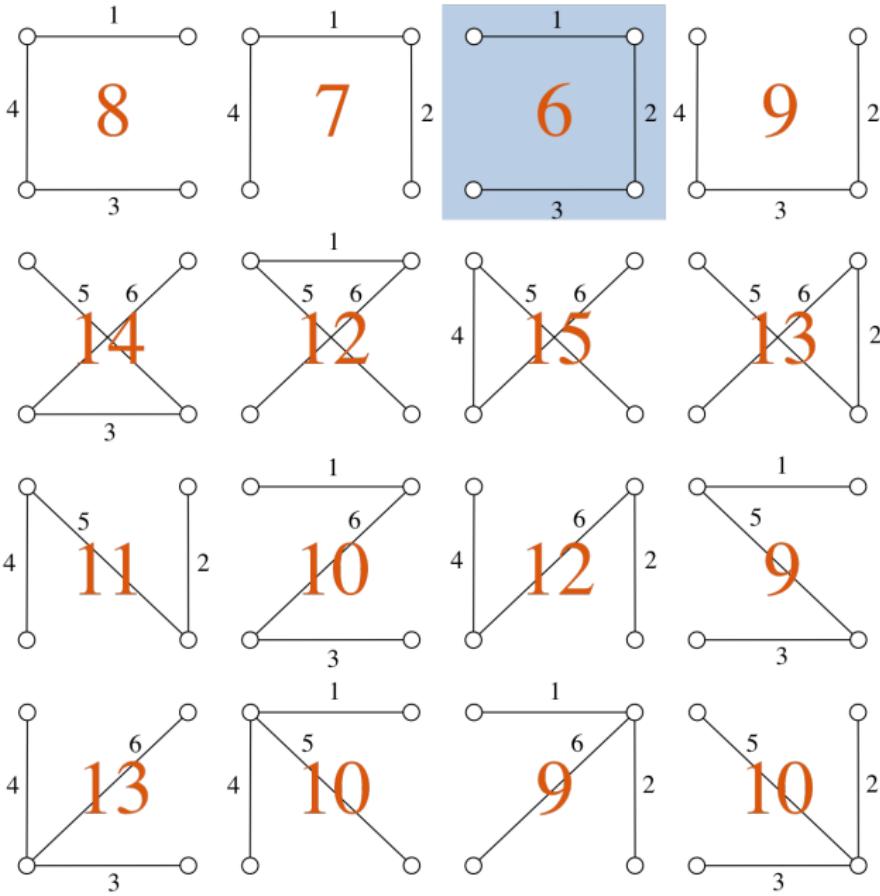


Uma **Árvore Geradora Mínima** (*Minimum Spanning Tree - MST*) T de G é uma árvore geradora cujo peso $w(T)$ (a soma dos pesos de suas arestas) é mínimo.

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

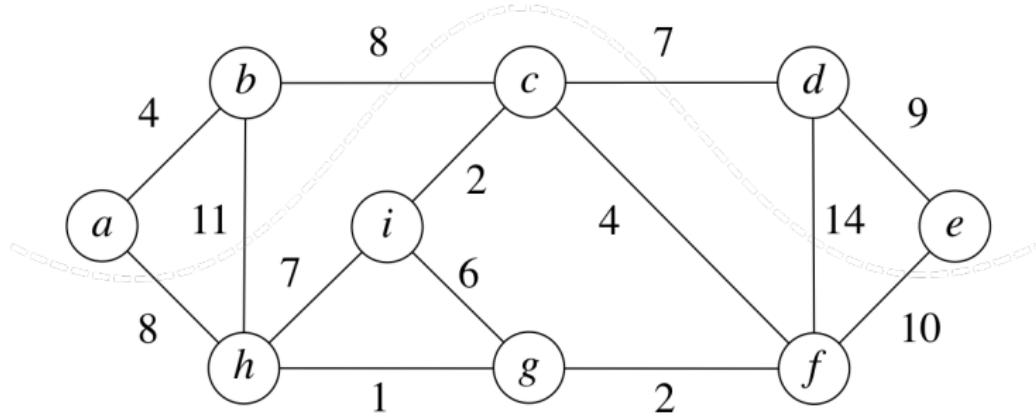
Exemplo. Dado um K_4 ponderado, determine sua MST.



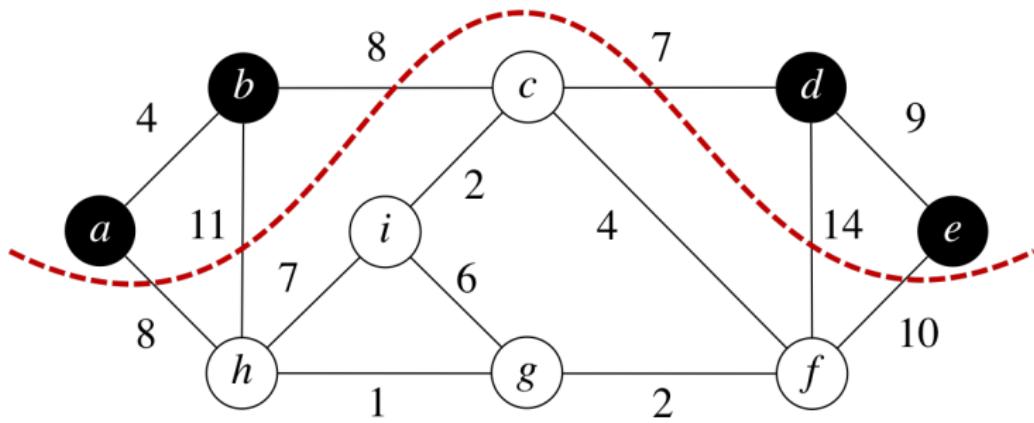


Algoritmo de Prim

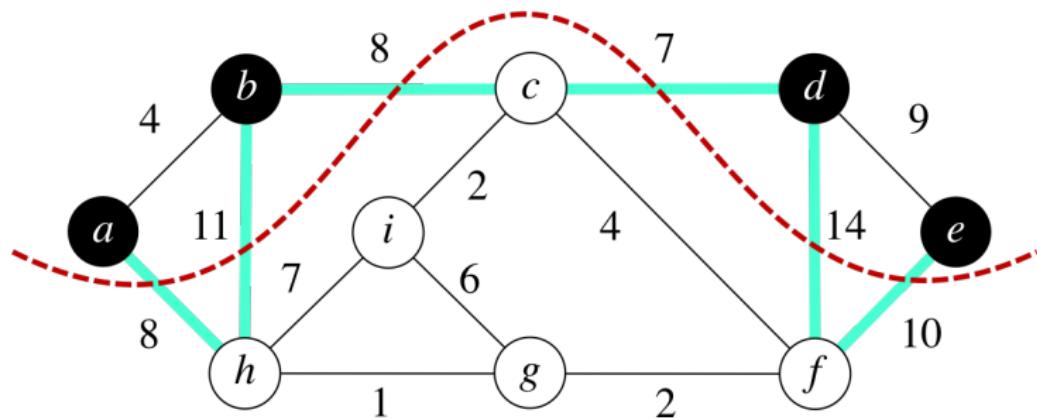
Seja um grafo não-direcionado $G = (V, E)$.



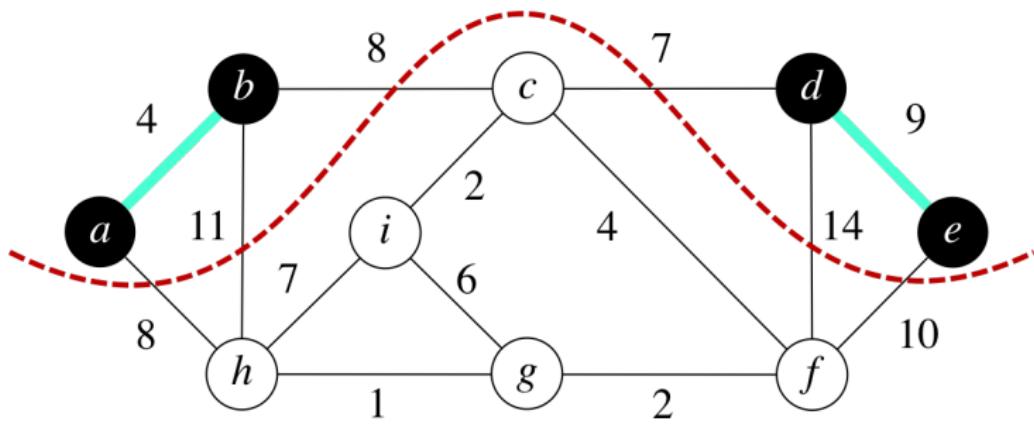
Um **corte** $(S, V - S)$ em G é uma bipartição de V ($S = \{a, b, d, e\}$ e $V - S = \{h, i, g, c\}$).



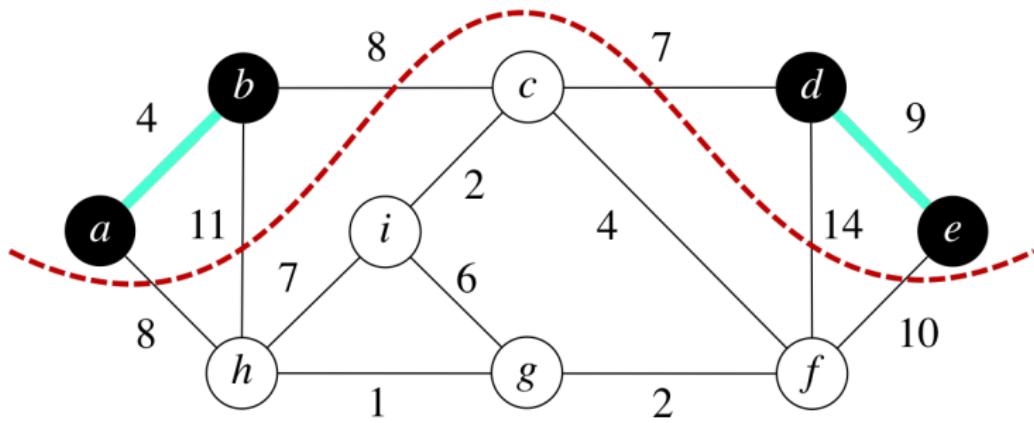
Algumas arestas cruzam S e $V - S$.



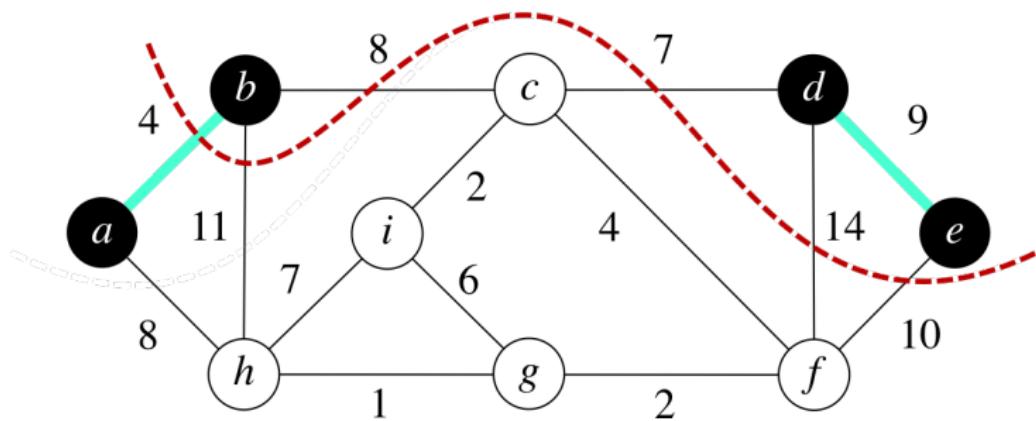
Um corte **respeita** um conjunto de arestas A se nenhuma aresta em A cruza o corte.



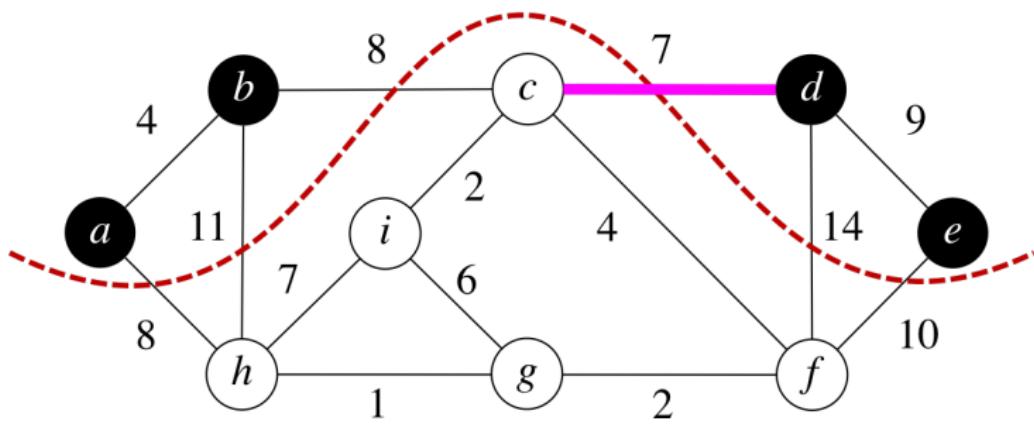
Este corte respeita $A = \{ab, de\}$.



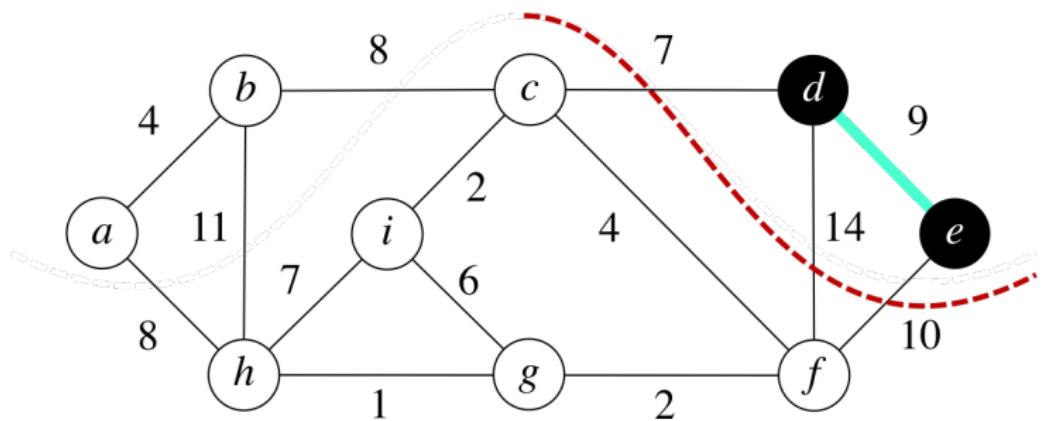
Este outro corte **não** respeita $A = \{ab, de\}$.



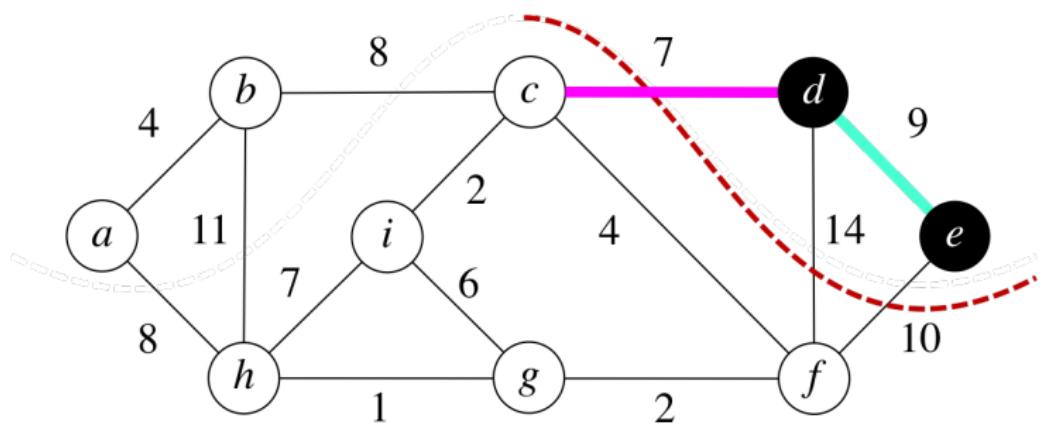
Uma aresta de cruzamento com peso mínimo é chamada de **aresta leve**.



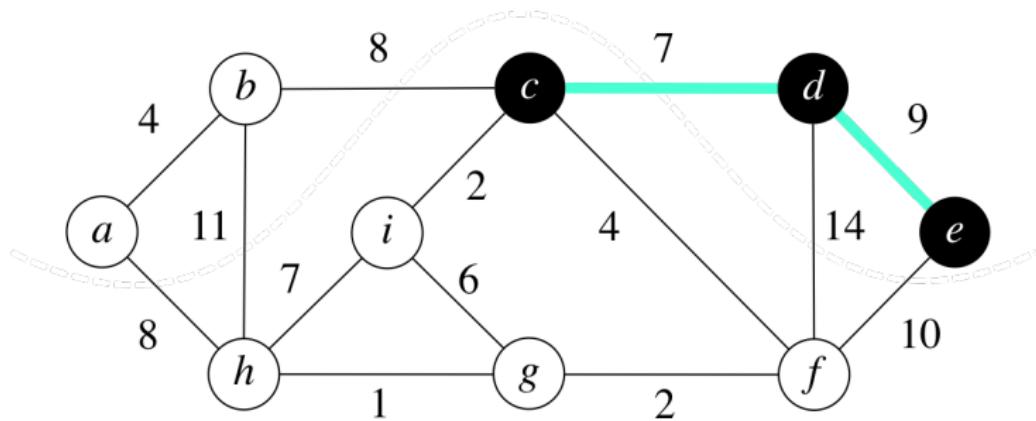
Seja T uma MST de um grafo $G = (V, E)$, um corte $(S, V - S)$ e $A \subseteq T$.



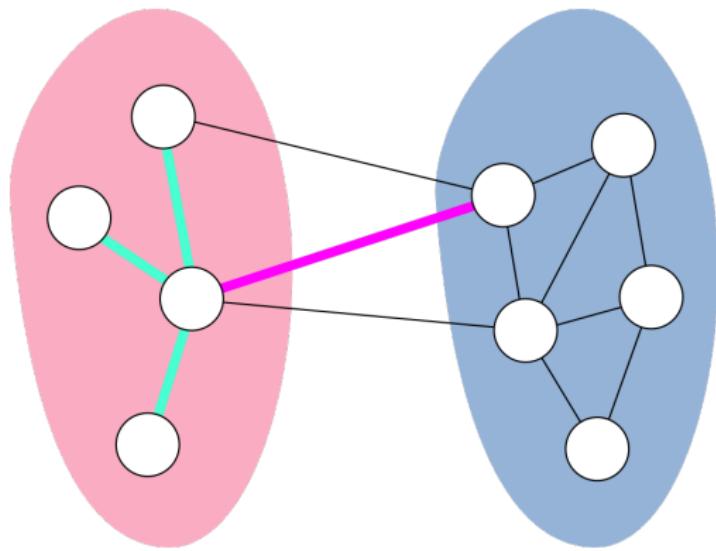
Uma **aresta segura**: (1) é leve; (2) forma uma MST com A .



Uma **aresta segura**: (1) é leve; (2) forma uma MST com A .

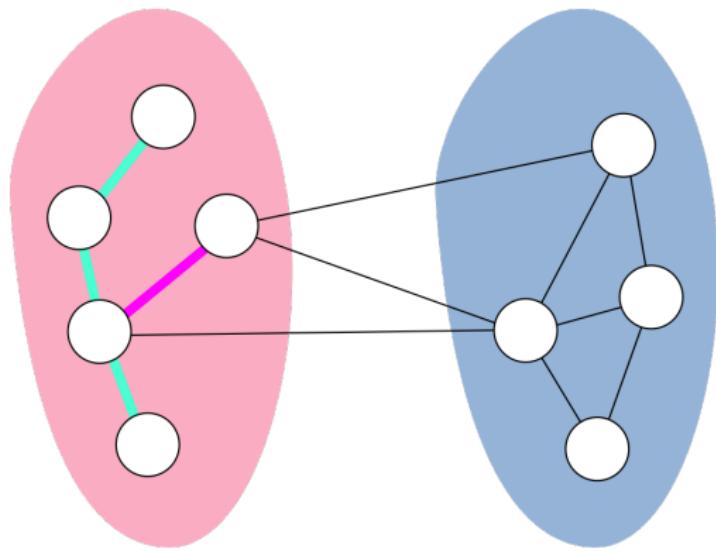


Algoritmo de Prim¹ cria um MST adicionando arestas seguras à uma MST inicial.



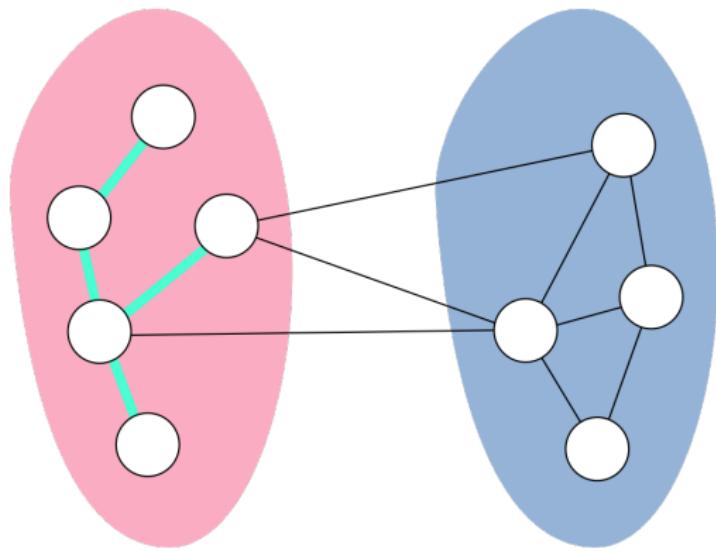
(1) Criado por Vojtech Jarník (1930), redescoberto por Robert C. Prim (1957) e Edsger W. Dijkstra (1959)

Algoritmo de Prim¹ cria um MST adicionando arestas seguras à uma MST inicial.



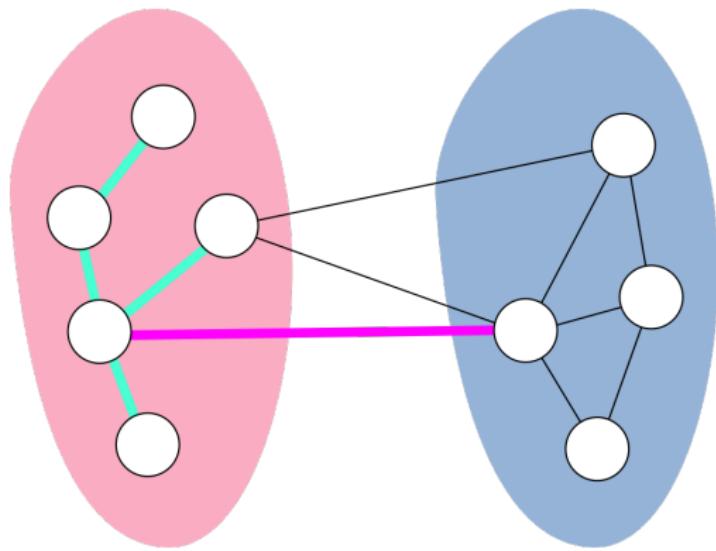
(1) Criado por Vojtech Jarník (1930), redescoberto por Robert C. Prim (1957) e Edsger W. Dijkstra (1959)

Algoritmo de Prim¹ cria um MST adicionando arestas seguras à uma MST inicial.



(1) Criado por Vojtech Jarník (1930), redescoberto por Robert C. Prim (1957) e Edsger W. Dijkstra (1959)

Algoritmo de Prim¹ cria um MST adicionando arestas seguras à uma MST inicial.



(1) Criado por Vojtech Jarník (1930), redescoberto por Robert C. Prim (1957) e Edsger W. Dijkstra (1959)

Elementos do algoritmo

Elementos do algoritmo

- ▶ Algoritmo constrói a MST vértice por vértice: $A = \{(v, v.\pi) : v \in V - \{r\}\}$.

Elementos do algoritmo

- ▶ Algoritmo constrói a MST vértice por vértice: $A = \{(v, v.\pi) : v \in V - \{r\}\}$.
- ▶ $(v, v.\pi)$ aresta de A com um vértice v e seu pai $v.\pi$.

Elementos do algoritmo

- ▶ Algoritmo constrói a MST vértice por vértice: $A = \{(v, v.\pi) : v \in V - \{r\}\}$.
- ▶ $(v, v.\pi)$ aresta de A com um vértice v e seu pai $v.\pi$.
- ▶ V é o conjunto de vértices e r é o vértice inicial de A .

Elementos do algoritmo

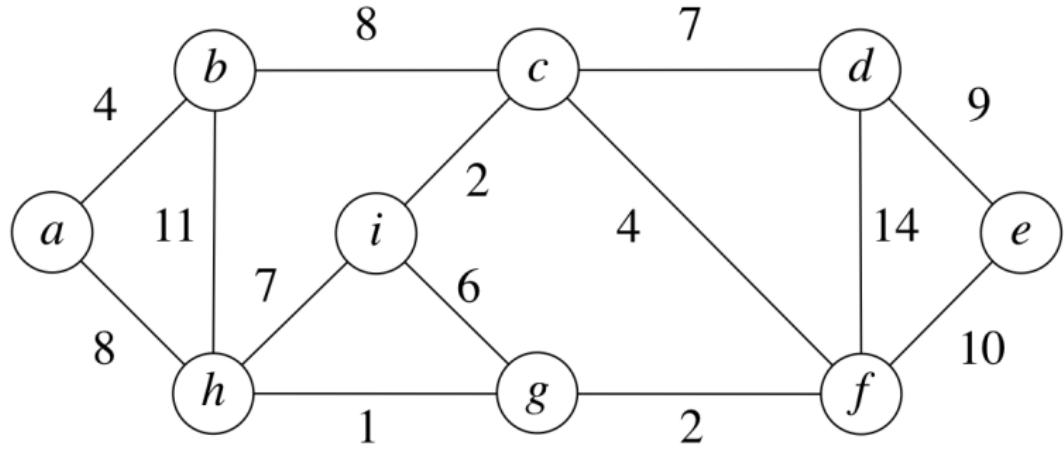
- Q , uma fila de prioridades mínimas, contém os vértices que ainda não entraram em A .

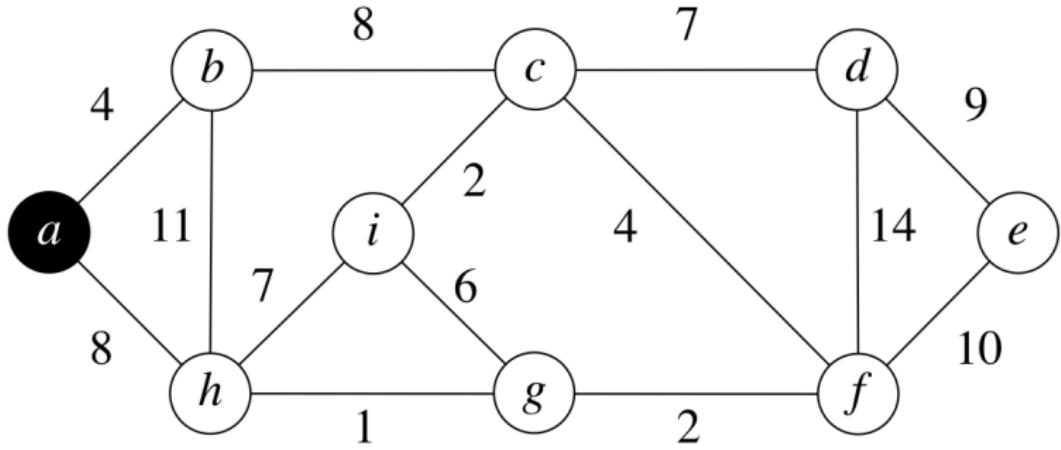
Elementos do algoritmo

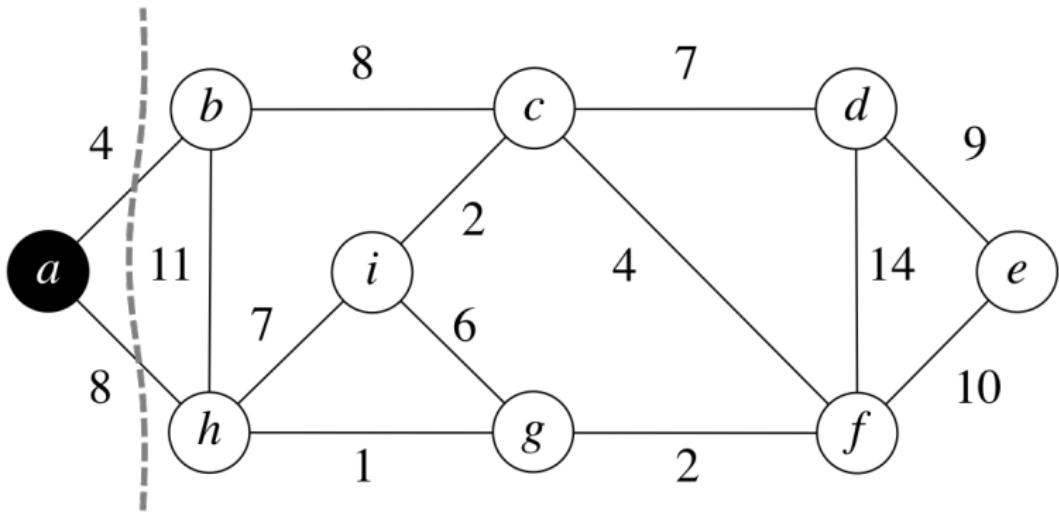
- ▶ Q , uma **fila de prioridades mínimas**, contém os vértices que ainda não entraram em A .
- ▶ Para cada vértice v em Q , $v.chave$ é o peso mínimo de alguma aresta conectando v a algum vértice em A .

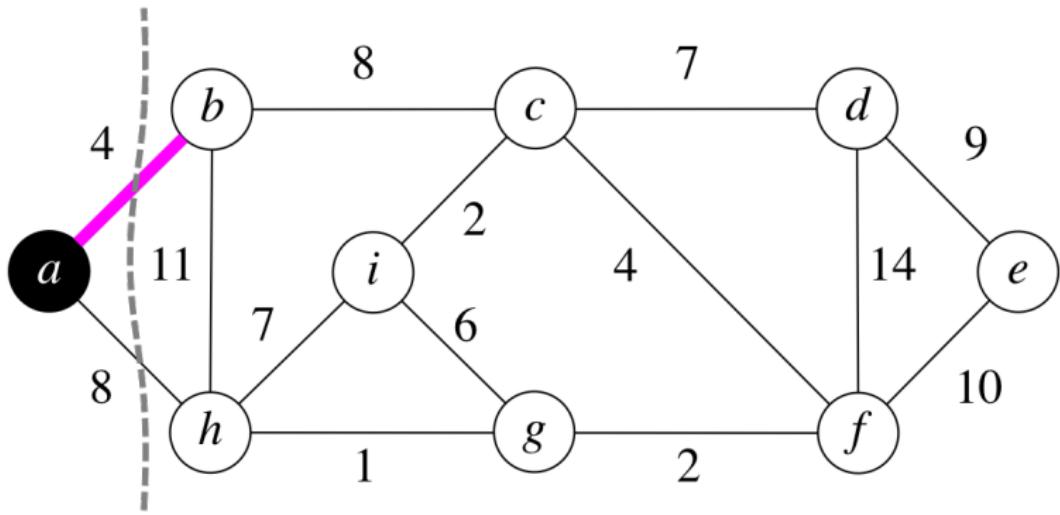
MST-PRIM(G, r, w)

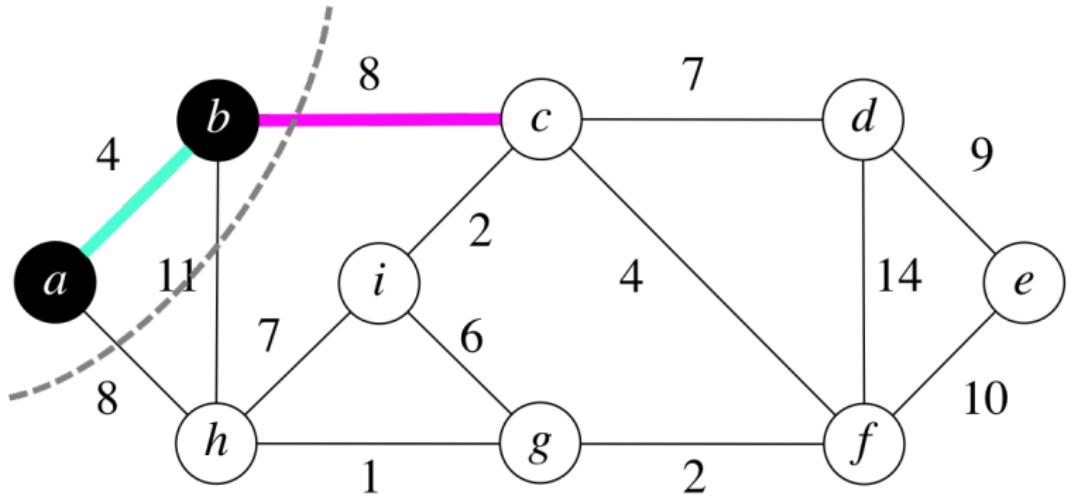
- 1 **para cada** $u \in G.V$
- 2 $u.chave = \infty$
- 3 $u.\pi = \text{NIL}$
- 4 $r.chave = 0$
- 5 $Q = G.V$
- 6 **enquanto** $Q \neq \emptyset$
- 7 $u = \text{EXTRACT-MIN}(Q)$
- 8 **para cada** $v \in G.Adj[u]$
- 9 **se** $v \in Q$ **e** $w(u, v) < v.chave$
- 10 $v.\pi = u$
- 11 $v.chave = w(u, v)$

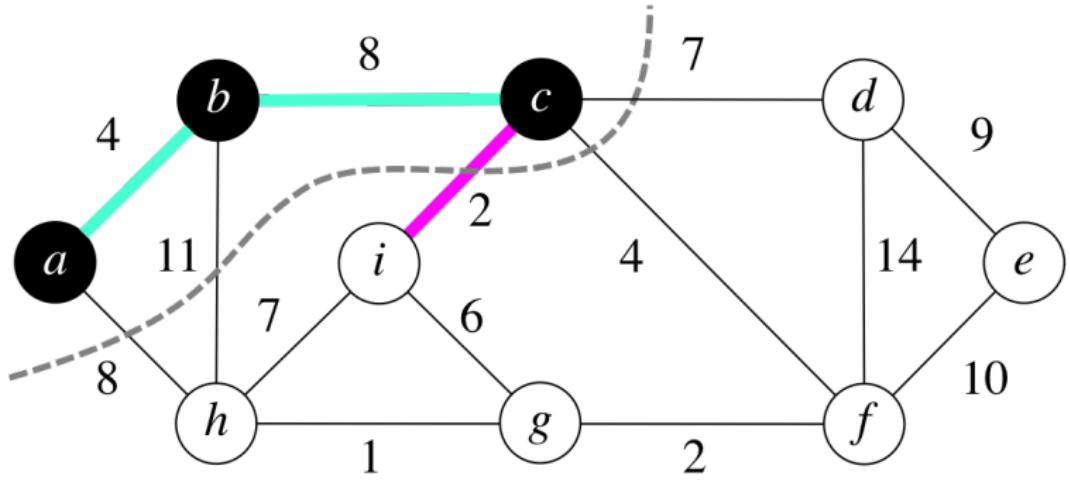


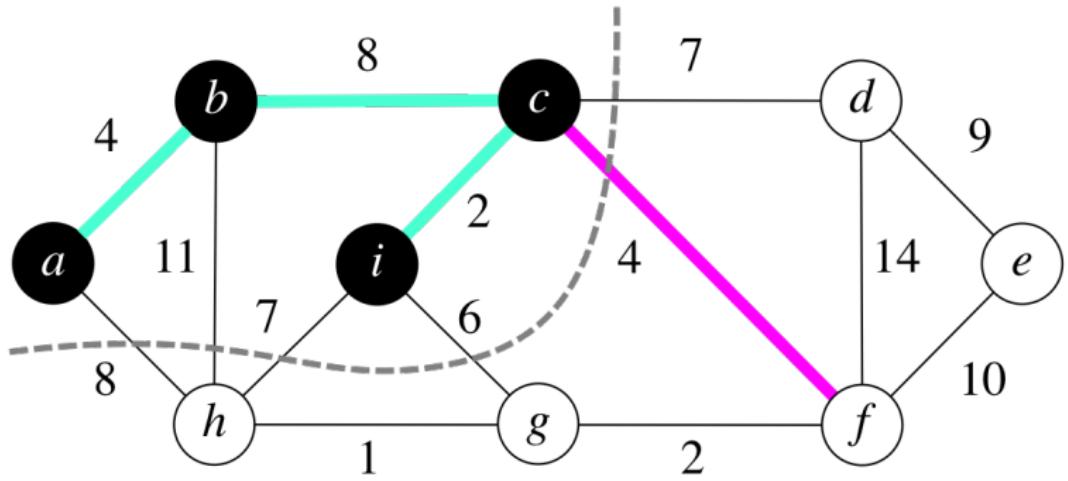


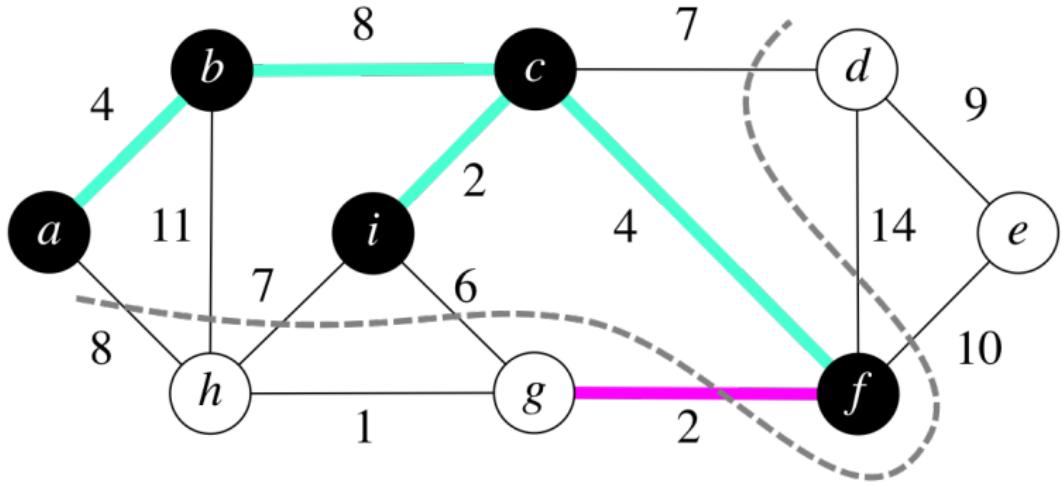


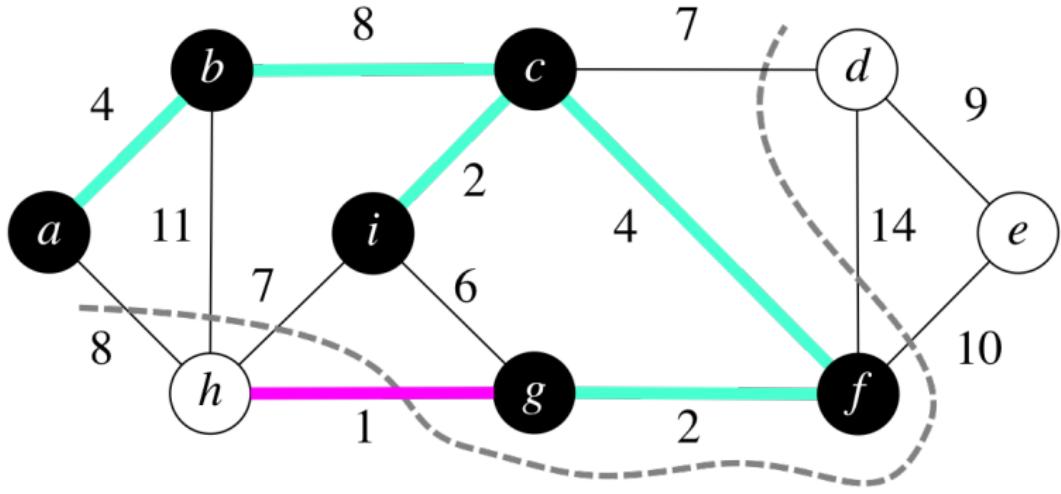


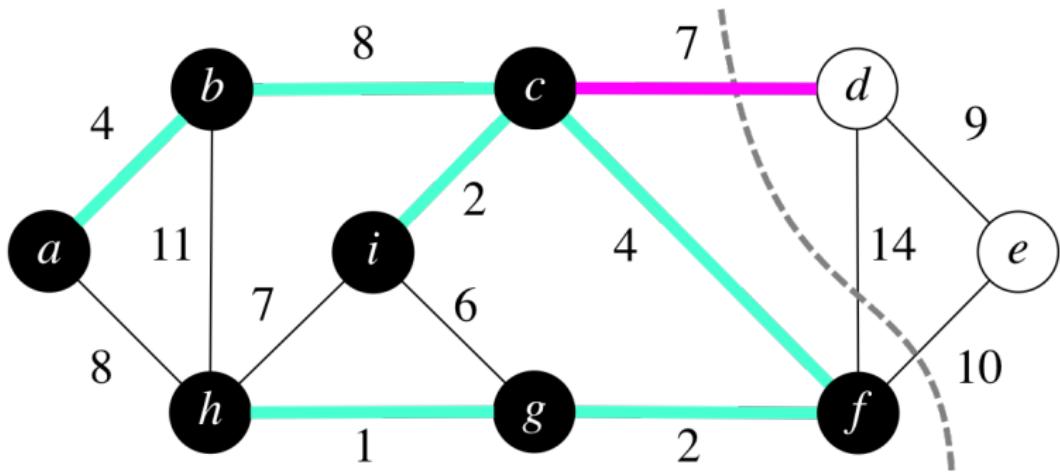


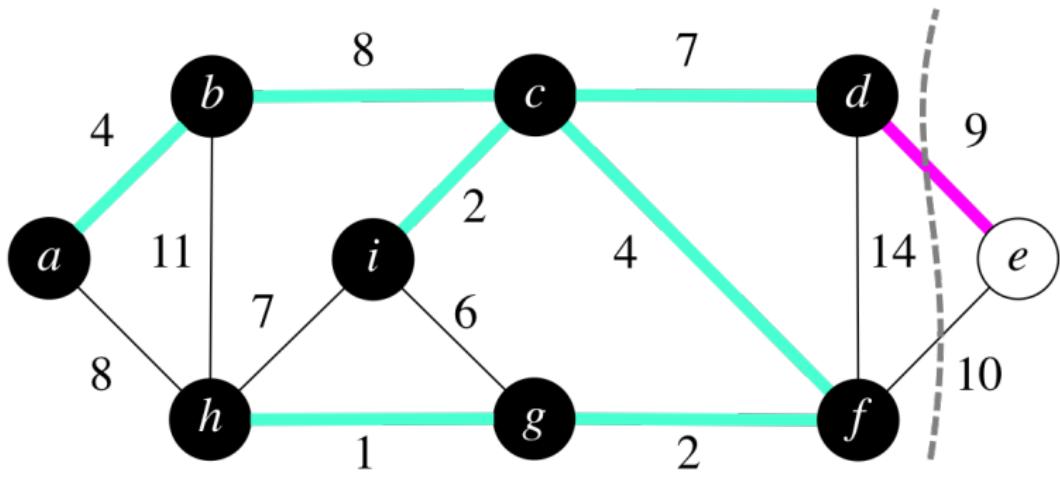


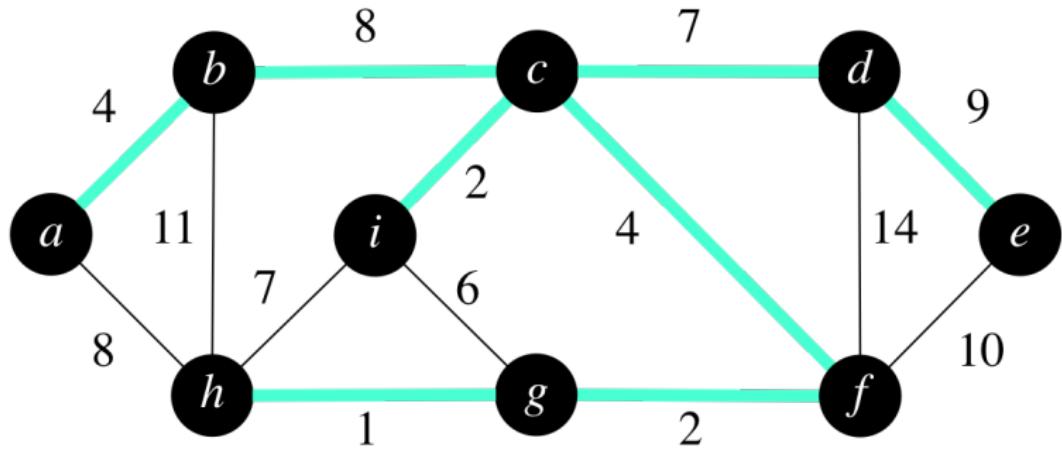


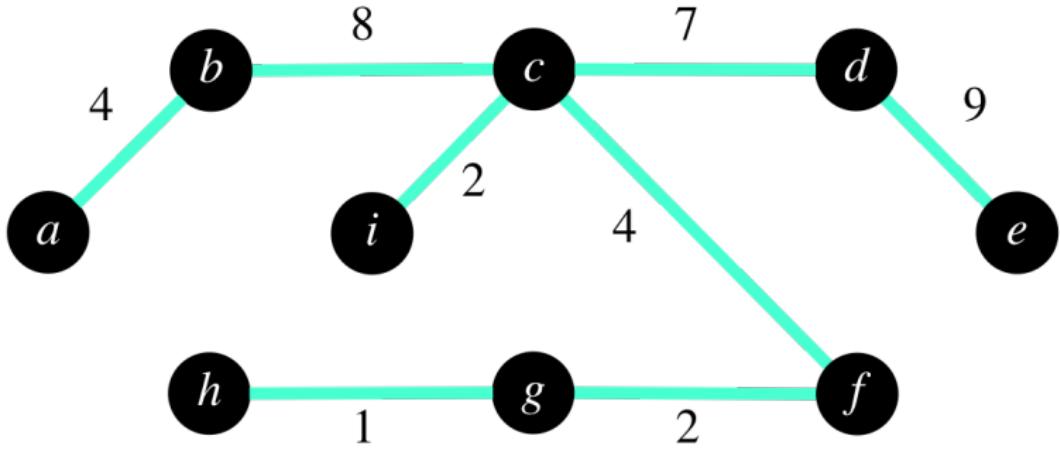


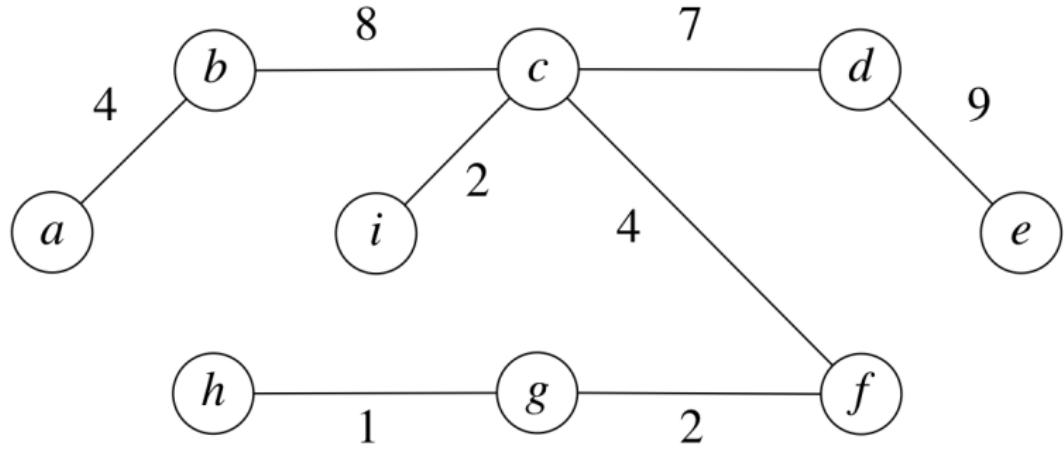












Uma **fila de prioridades** (*Priority Queue - PQ*) é uma estrutura de dados que mantém um conjunto S de elementos, cada um associado a uma chave.

Uma **fila de prioridades** (*Priority Queue - PQ*) é uma estrutura de dados que mantém um conjunto S de elementos, cada um associado a uma chave.

Comumente implementada com **heap binário**.

Operações básicas

Operações básicas

- ▶ $\text{INSERT}(S, x)$. Insere o elemento x na PQ S .

Operações básicas

- ▶ $\text{INSERT}(S, x)$. Insere o elemento x na PQ S .
- ▶ $\text{MINIMUM}(S)$. Retorna o elemento de S com chave mínima.

Operações básicas

- ▶ $\text{INSERT}(S, x)$. Insere o elemento x na PQ S .
- ▶ $\text{MINIMUM}(S)$. Retorna o elemento de S com chave mínima.
- ▶ $\text{EXTRACT-MIN}(S)$. Remove e retorna o elemento de S com chave mínima.

Operações básicas

- DECREASE-MIN(S, x, k). Atualiza o valor do elemento x com o novo valor de chave k .

Supõe-se que a chave antiga seja maior que a nova chave.

Complexidade de tempo de Prim: $O(E \lg V)$.

Algoritmo de Kruskal

Adiciona uma aresta por vez em uma floresta A , na ordem não-decrescente dos pesos.

Uma aresta é incluída se conecta vértices de árvores distintas de A . Ao final, A se torna uma MST.

Elementos do algoritmo

- ▶ Algoritmo constrói a MST no conjunto de arestas A .

Elementos do algoritmo

- ▶ Algoritmo constrói a MST no conjunto de arestas A .
- ▶ Utiliza uma estrutura de dados **Union-Find** para representar a floresta de MSTs.

Union-Find

- ▶ Estrutura de dados que trata da partição de uma coleção de conjuntos disjuntos $S = \{S_1, S_2, \dots, S_n\}$.

Union-Find

- ▶ Estrutura de dados que trata da partição de uma coleção de conjuntos disjuntos $S = \{S_1, S_2, \dots, S_n\}$.
- ▶ S inicia com n subconjuntos com um único elemento, e depois sofre modificações por meio da união de alguns destes subconjuntos.

Union-Find: operações

- ▶ $\text{MAKE-SET}(x)$. Cria um novo conjunto com x como membro único (representante do conjunto).

Union-Find: operações

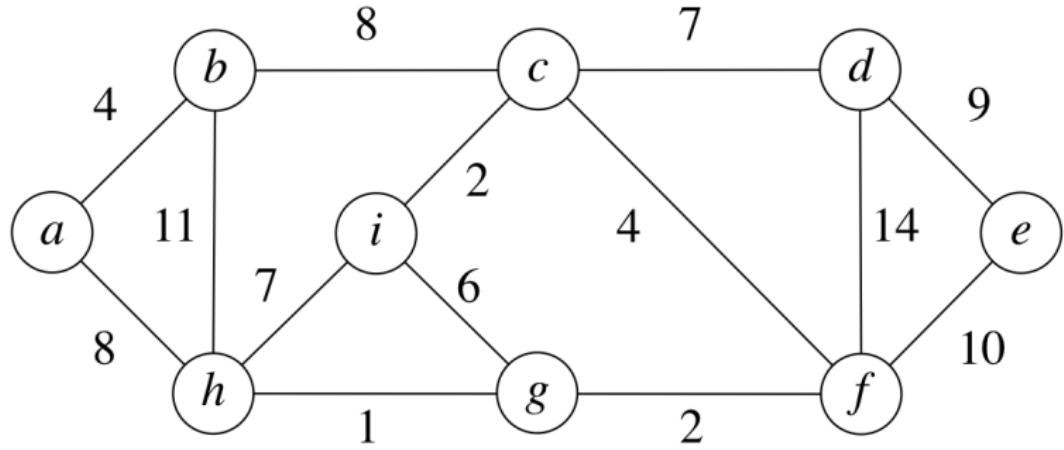
- ▶ MAKE-SET(x). Cria um novo conjunto com x como membro único (representante do conjunto).
- ▶ UNION(x, y). Une os dois conjuntos disjuntos S_x e S_y cujos membros são x e y , respectivamente, em um novo conjunto disjunto, digamos, S_{xy} .

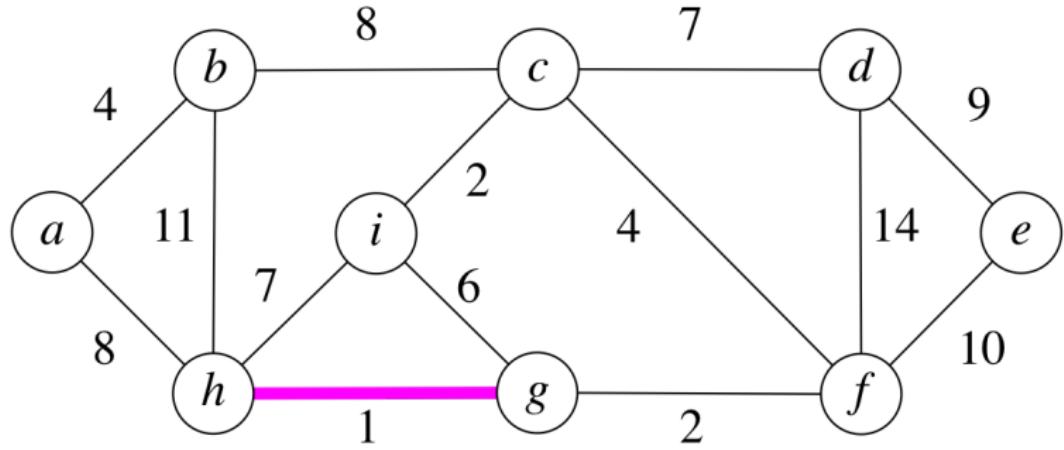
Union-Find: operações

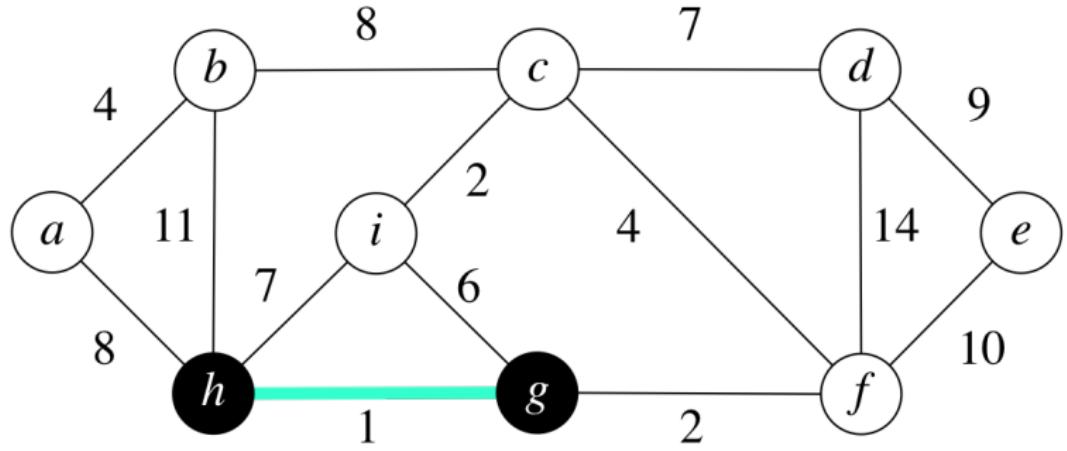
- ▶ FIND-SET(x). Retorna um ponteiro para o elemento representante do único conjunto contendo x .

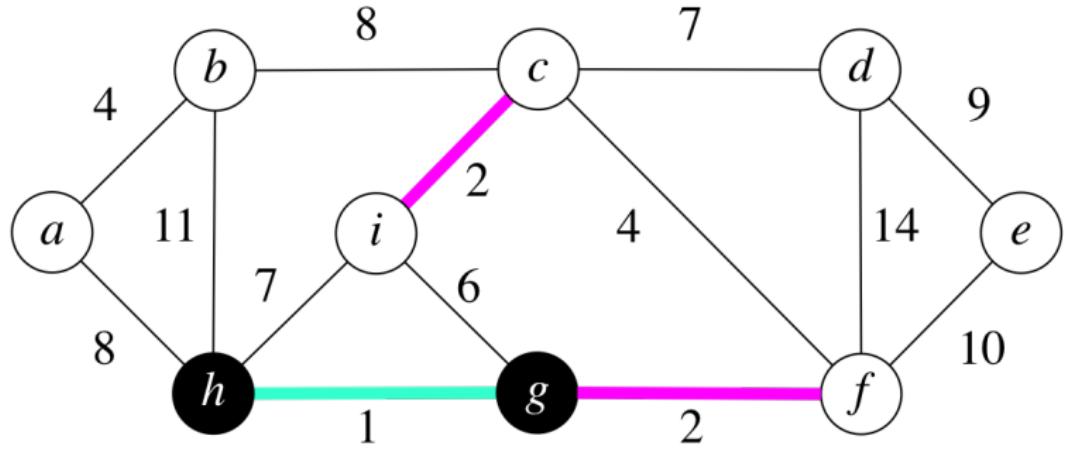
MST-KRUSKAL(G, w)

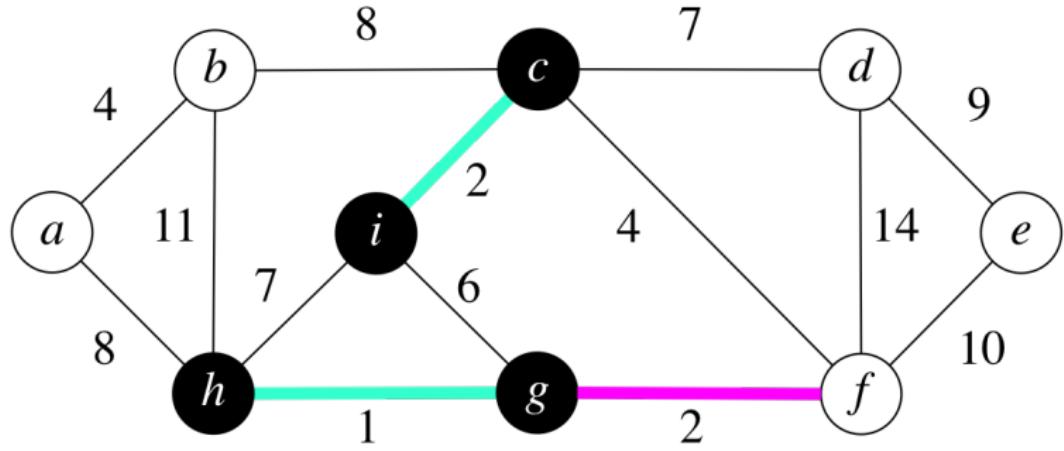
- 1 $A = \emptyset$
- 2 **para cada** vértice $v \in G.V$
 - 3 MAKE-SET(v)
- 4 ordene as arestas de $G.E$ em ordem
não-decrescente de peso
- 5 **para cada** aresta $\{u, v\} \in G.E // ordem acima$
 - 6 **if** FIND-SET(u) \neq FIND-SET(v)
 - 7 $A = A \cup \{u, v\}$
 - 8 UNION(u, v)
 - 9 **retorne** A

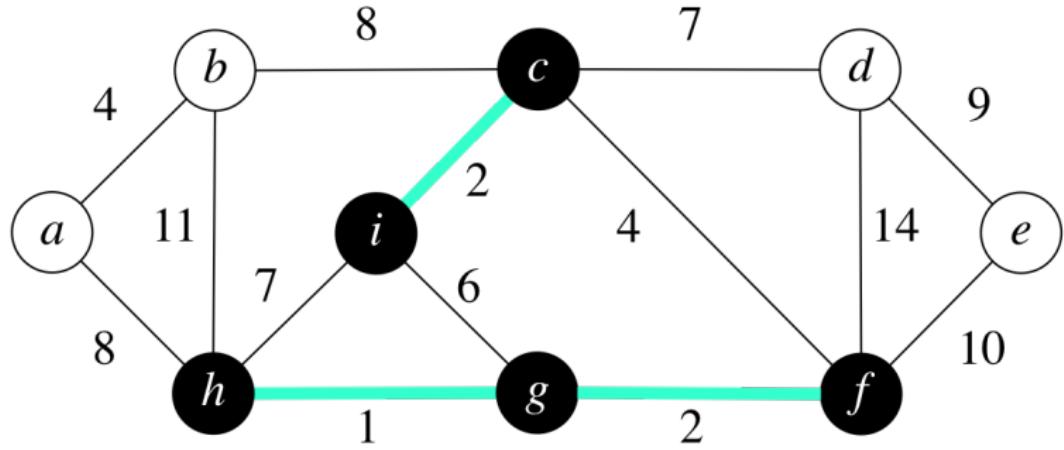


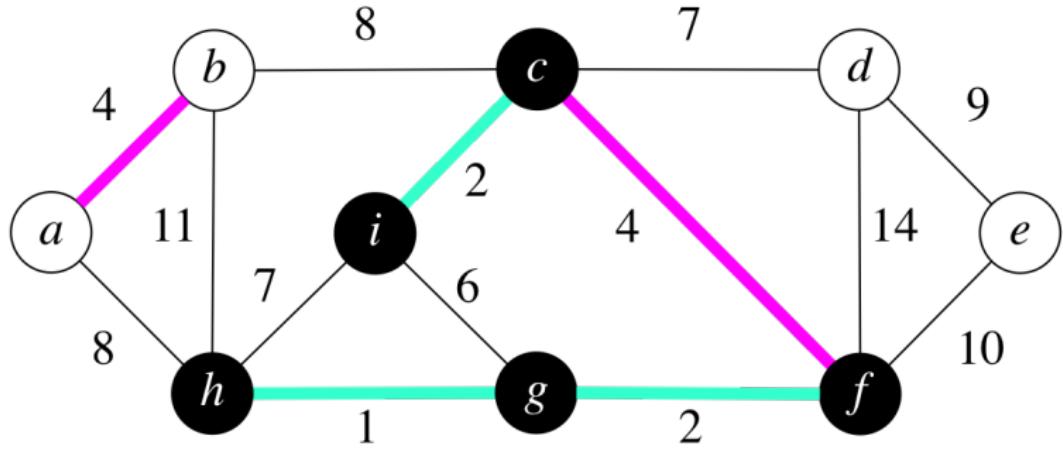


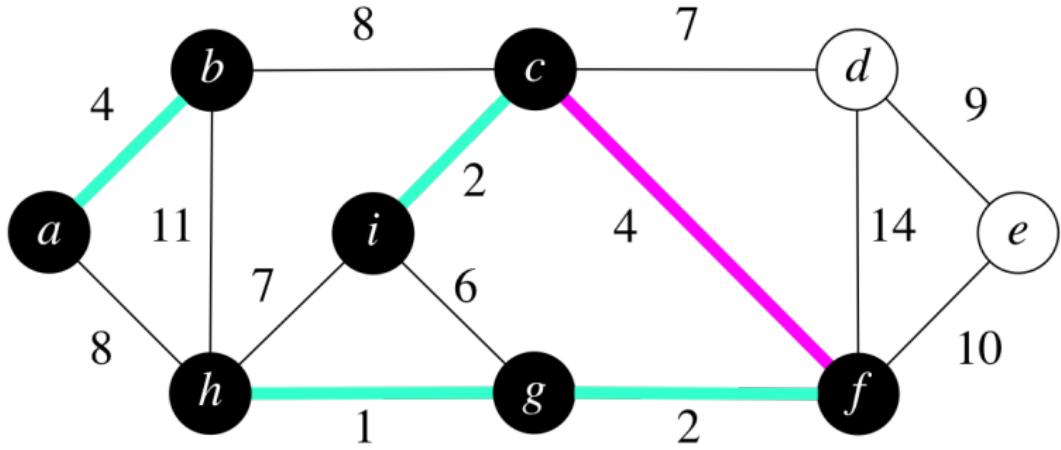


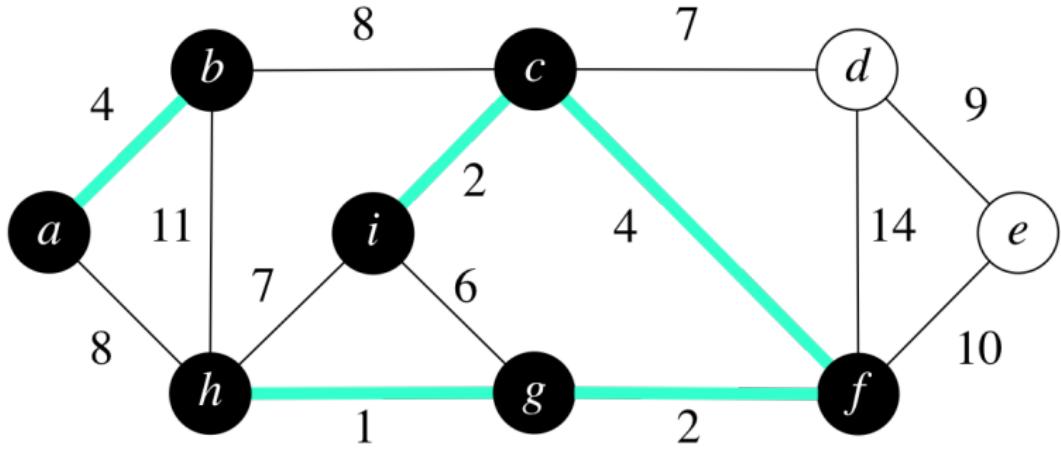


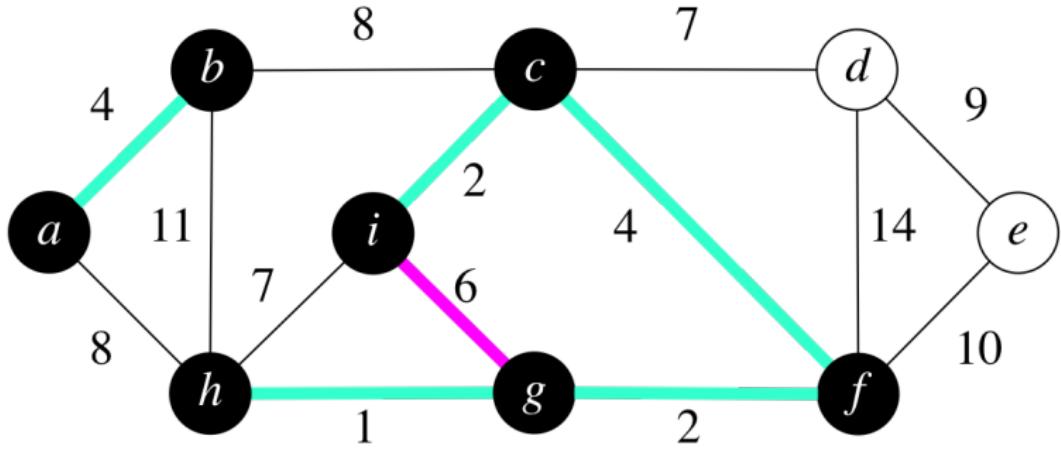


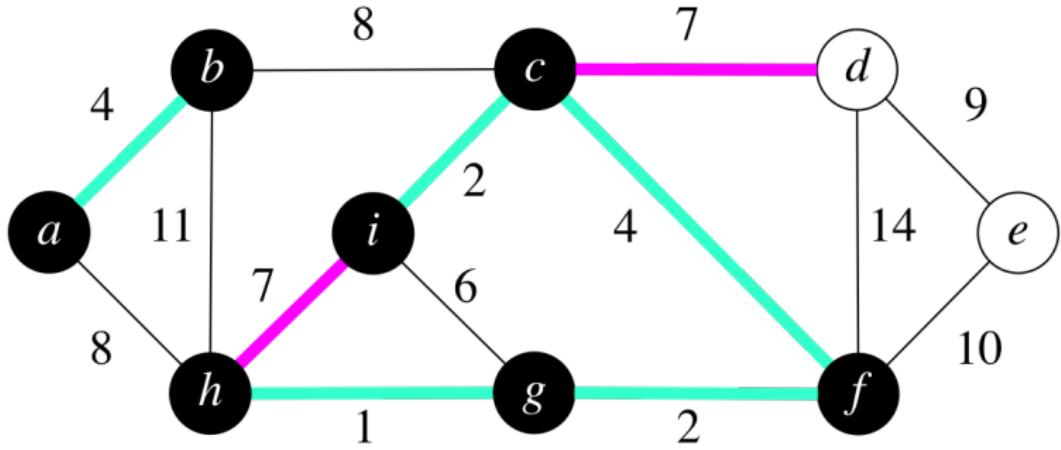


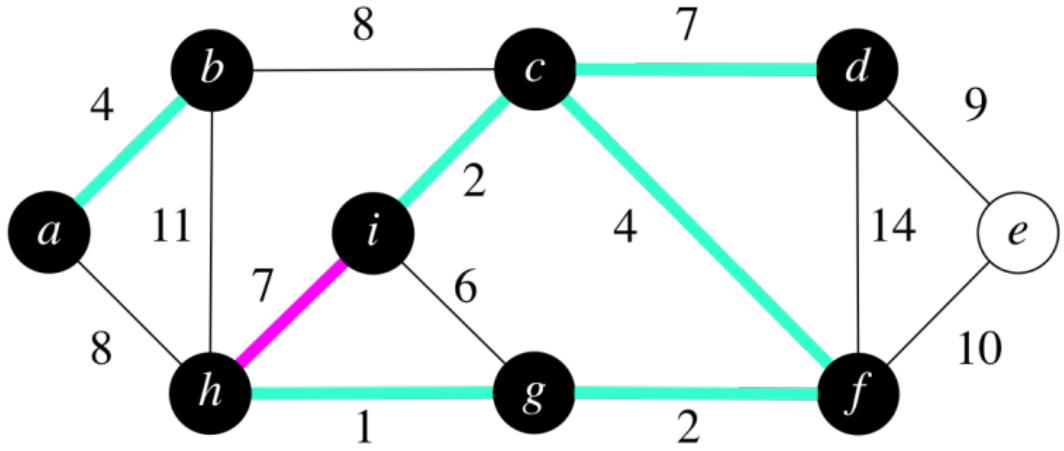


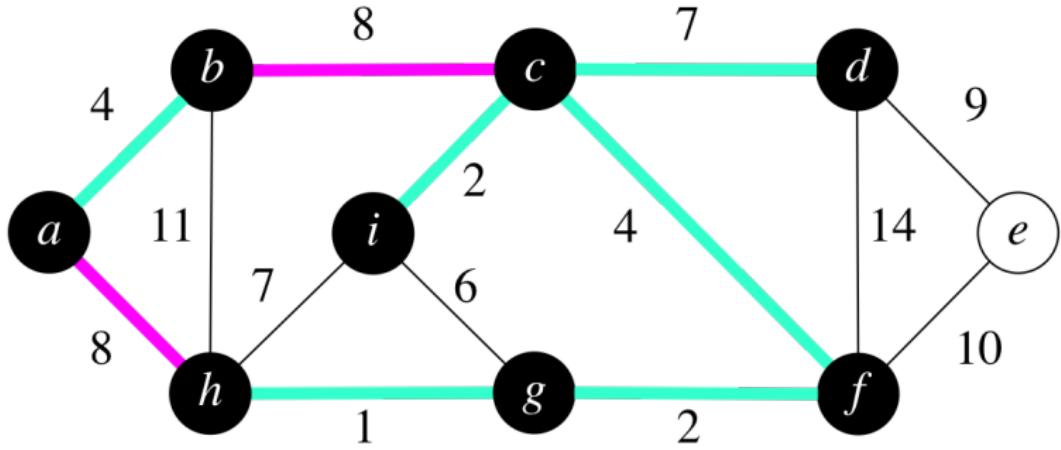


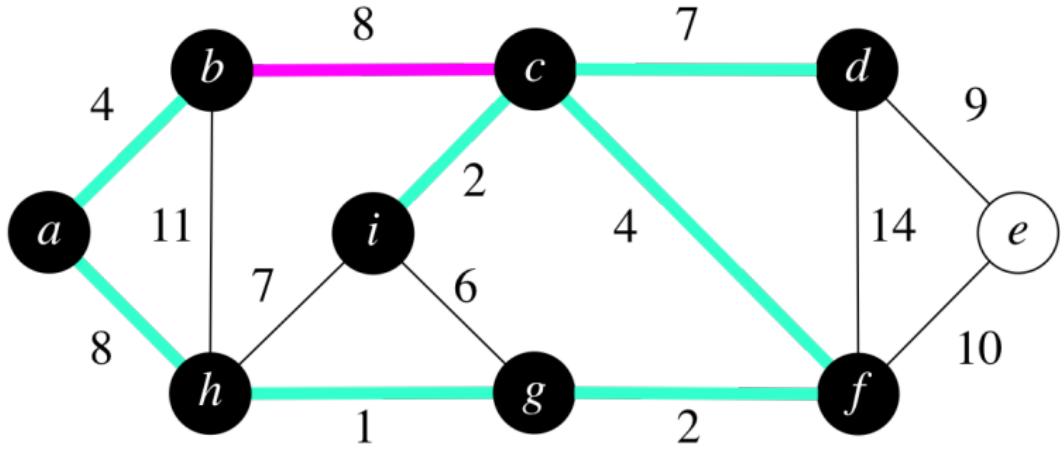


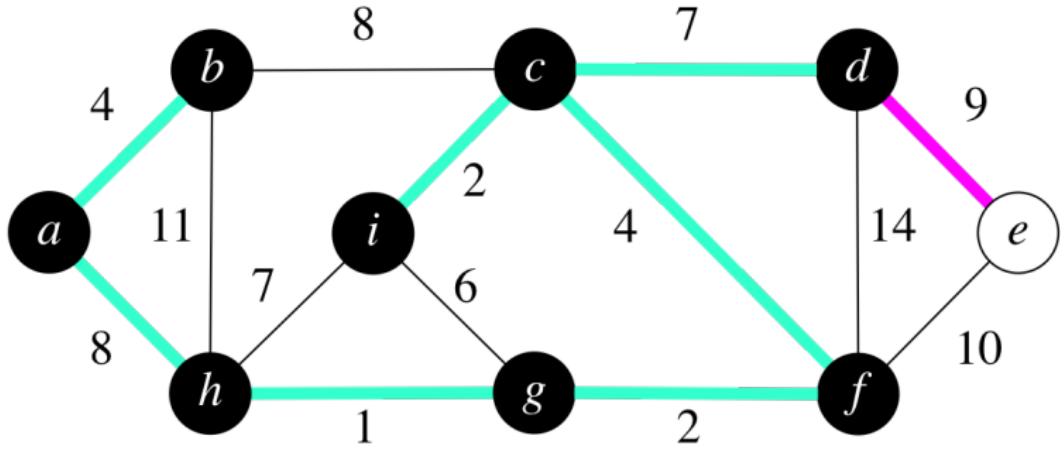


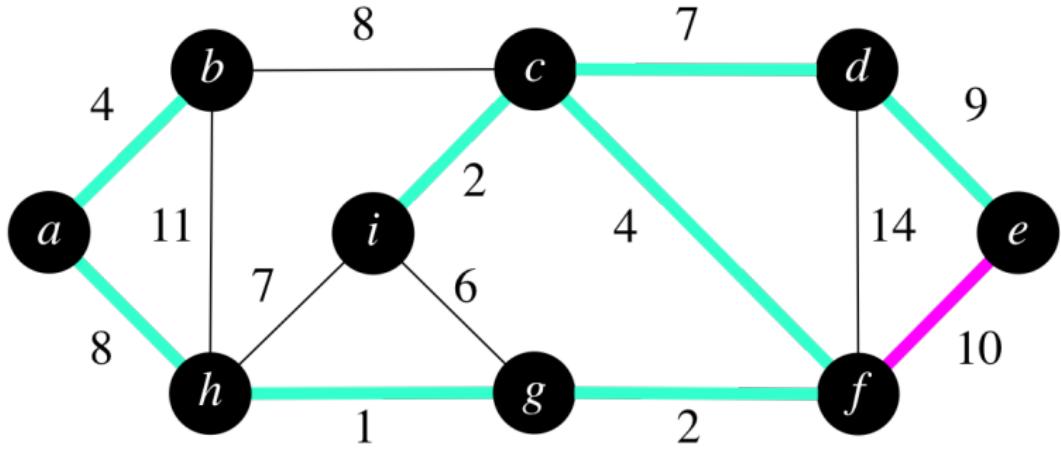


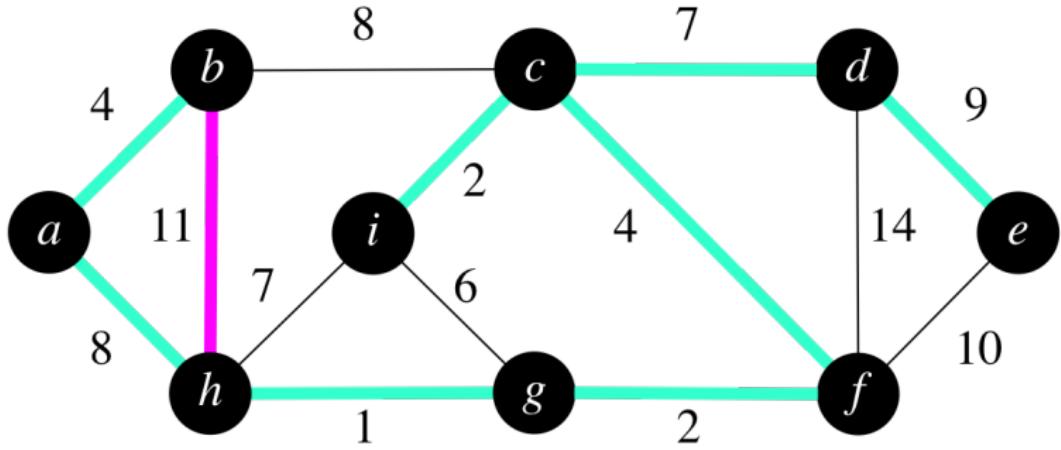


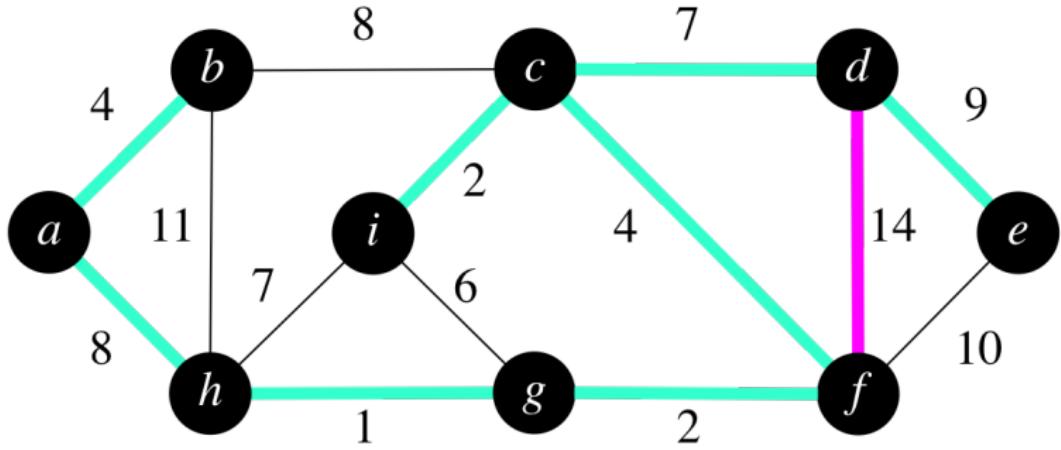


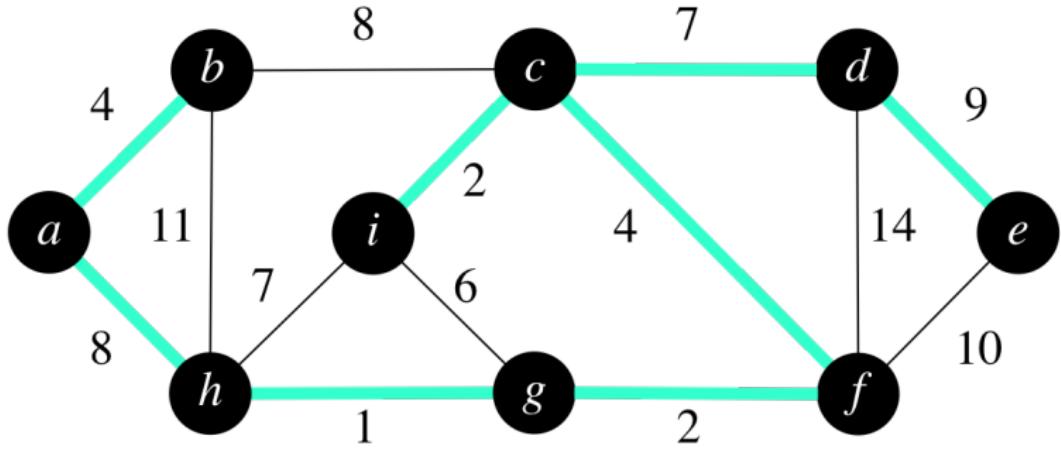


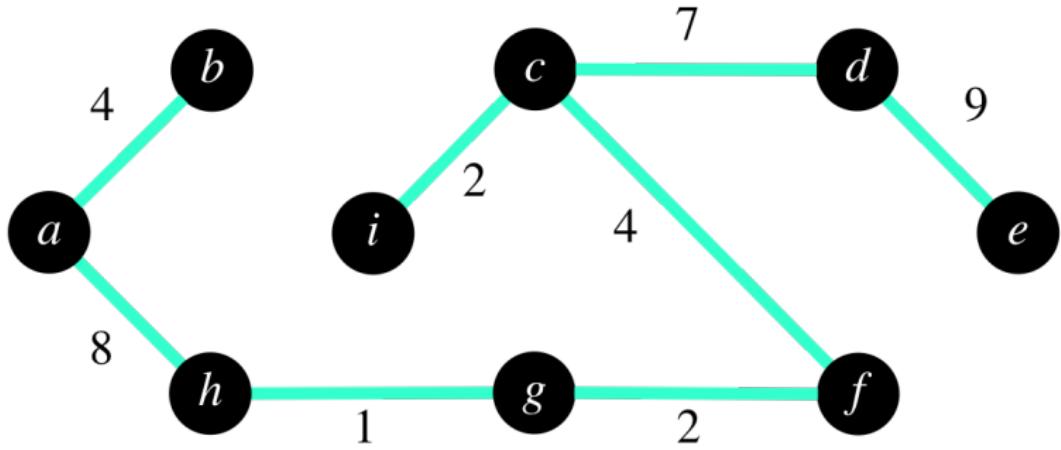


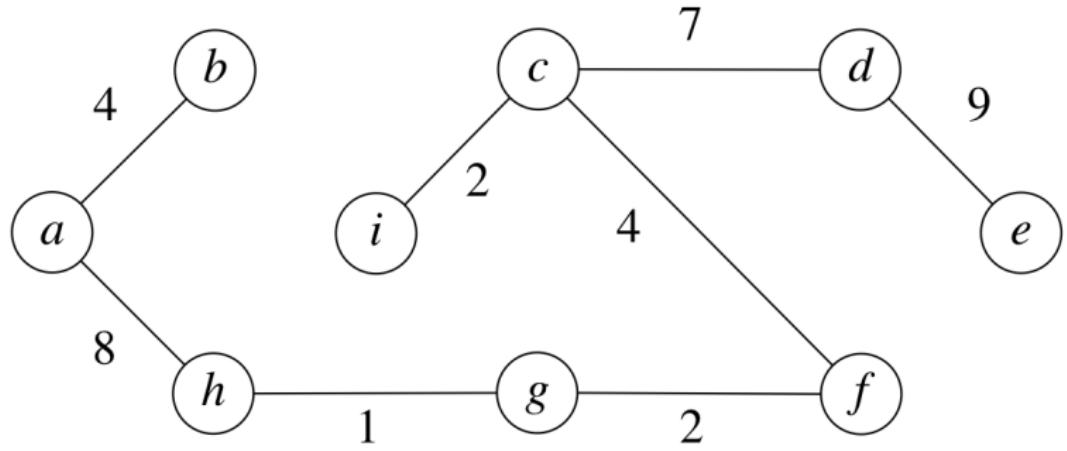












Complexidade de tempo de Kruskal

- ▶ Union-Find implementado como uma **floresta de conjuntos disjuntos** usando as heurísticas “*union by rank*” e “*path compression*”, obtém tempo teórico superlinear ou linear na prática (CRLS, 12.3).

Referências

-  T. H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, MIT Press, 2010
-  A. Levitin. Introduction to the Design and Analysis of Algorithms. 3rd edition. Addison-Wesley, 2007
-  R. Sedgewick, K. Wayne. Algorithms. 4th edition, Addison-Wesley Professional, 2011
-  N. Ziviani. Projeto de Algoritmos com Implementação em Pascal C. Cengage Learning, 2012

Onde obter este material:

est.uea.edu.br/fcoelho