Estruturas de Dados para Conjuntos Disjuntos: *Union-find*

Letícia Rodrigues Bueno

UFABC

Estruturas de Dados para Conjuntos Disjuntos: Introdução

Estruturas de Dados para Conjuntos Disjuntos: Introdução

- Aplicações que envolvem agrupamento de *n* elementos distintos em uma coleção de conjuntos disjuntos;

Estruturas de Dados para Conjuntos Disjuntos: Introdução

- Aplicações que envolvem agrupamento de *n* elementos distintos em uma coleção de conjuntos disjuntos;
- **Operações importantes:**

Estruturas de Dados para Conjuntos Disjuntos: Introdução

- Aplicações que envolvem agrupamento de *n* elementos distintos em uma coleção de conjuntos disjuntos;
- **Operações importantes:**
  1. encontrar o conjunto a que pertence um elemento (**“find”**);

Estruturas de Dados para Conjuntos Disjuntos: Introdução

- Aplicações que envolvem agrupamento de *n* elementos distintos em uma coleção de conjuntos disjuntos;
- **Operações importantes:**
  1. encontrar o conjunto a que pertence um elemento (**“find”**);
  2. unir dois conjuntos (**“union”**).

Operações de Conjuntos Disjuntos

Operações de Conjuntos Disjuntos

- Coleção de conjuntos dinâmicos disjuntos: $S = \{S_1, S_2, \ldots, S_k\}$;

Operações de Conjuntos Disjuntos

- Coleção de conjuntos dinâmicos disjuntos: $S = \{S_1, S_2, \ldots, S_k\}$;
- Cada conjunto é identificado por um representante (um elemento do conjunto);

Operações de Conjuntos Disjuntos

- Coleção de conjuntos dinâmicos disjuntos: $S = \{s_1, s_2, \ldots, s_k\}$;
- Cada conjunto é identificado por um representante (um elemento do conjunto);
- **Operações desejáveis:**

Operações de Conjuntos Disjuntos

- Coleção de conjuntos dinâmicos disjuntos: $S = \{s_1, s_2, \ldots, s_k\}$;
- Cada conjunto é identificado por um representante (um elemento do conjunto);
- **Operações desejáveis:**
  1. **makeSet(x)**: cria conjuntos de único elemento x (representante é o próprio x);

Operações de Conjuntos Disjuntos

- Coleção de conjuntos dinâmicos disjuntos: $S = \{s_1, s_2, \ldots, s_k\}$;
- Cada conjunto é identificado por um representante (um elemento do conjunto);
- **Operações desejáveis:**
  1. **makeSet(x)**: cria conjuntos de único elemento x (representante é o próprio x);
  2. **union(x, y)**: une conjuntos dinâmicos S_x e S_y. Representante é escolhido para novo conjunto;

Operações de Conjuntos Disjuntos

- Coleção de conjuntos dinâmicos disjuntos: $S = \{s_1, s_2, \ldots, s_k\}$;
- Cada conjunto é identificado por um representante (um elemento do conjunto);
- **Operações desejáveis:**
  1. **makeSet(x)**: cria conjuntos de único elemento x (representante é o próprio x);
  2. **union(x, y)**: une conjuntos dinâmicos S_x e S_y. Representante é escolhido para novo conjunto;
  3. **findSet(x)**: retorna representante do conjunto que contém x;

Implementação através de Listas Encadeadas

Implementação através de Listas Encadeadas

- cada conjunto é representado por um lista simplesmente encadeada;

Implementação através de Listas Encadeadas

- cada conjunto é representado por um lista simplesmente encadeada;
- primeiro objeto na lista é o representante;

Implementação através de Listas Encadeadas

- cada conjunto é representado por um lista simplesmente encadeada;
- primeiro objeto na lista é o representante;
- cada nó contém:

Implementação através de Listas Encadeadas

- cada conjunto é representado por um lista simplesmente encadeada;
- primeiro objeto na lista é o representante;
- cada nó contém:
  1. um elemento do conjunto;

Implementação através de Listas Encadeadas

- cada conjunto é representado por um lista simplesmente encadeada;
- primeiro objeto na lista é o representante;
- cada nó contém:
  1. um elemento do conjunto;
  2. ponteiro para próximo nó;

Implementação através de Listas Encadeadas

- cada conjunto é representado por um lista simplesmente encadeada;
- primeiro objeto na lista é o representante;
- cada nó contém:
  1. um elemento do conjunto;
  2. ponteiro para próximo nó;
  3. ponteiro para o representante;

Implementação através de Listas Encadeadas

- cada conjunto é representado por um lista simplesmente encadeada;
- primeiro objeto na lista é o representante;
- cada nó contém:
  1. um elemento do conjunto;
  2. ponteiro para próximo nó;
  3. ponteiro para o representante;
- **makeSet(x)**: tempo $O(1)$;

Implementação através de Listas Encadeadas

- cada conjunto é representado por um lista simplesmente encadeada;
- primeiro objeto na lista é o representante;
- cada nó contém:
  1. um elemento do conjunto;
  2. ponteiro para próximo nó;
  3. ponteiro para o representante;
- **makeSet(x)**: tempo $O(1)$;
- **findSet(x)**: tempo $O(1)$;

Implementação através de Listas Encadeadas

- cada conjunto é representado por um lista simplesmente encadeada;
- primeiro objeto na lista é o representante;
- cada nó contém:
  1. um elemento do conjunto;
  2. ponteiro para próximo nó;
  3. ponteiro para o representante;
- **makeSet(x)**: tempo $O(1)$;
- **findSet(x)**: tempo $O(1)$;
- **union(x, y)**:

Implementação através de Listas Encadeadas

- cada conjunto é representado por um lista simplesmente encadeada;
- primeiro objeto na lista é o representante;
- cada nó contém:
  1. um elemento do conjunto;
  2. ponteiro para próximo nó;
  3. ponteiro para o representante;
- **makeSet(x)**: tempo $O(1)$;
- **findSet(x)**: tempo $O(1)$;
- **union(x, y)**: **??????**

Implementação através de Listas Encadeadas

Implementação de **union**(**x**, **y**):

Implementação através de Listas Encadeadas

Implementação de **union**(**x**, **y**):

- lista de *x* anexada ao final da lista de *y*;

Implementação através de Listas Encadeadas

Implementação de **union**(**x**, **y**):

- lista de x anexada ao final da lista de y;
- representante da nova lista é y;

Implementação através de Listas Encadeadas

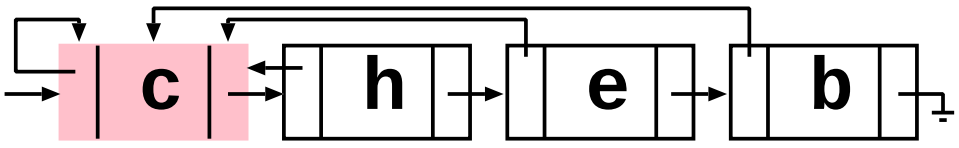
Implementação de **union**(**x**, **y**):

- lista de x anexada ao final da lista de y;
- representante da nova lista é y;
- nós de x devem apontar para representante y: custa comprimento da lista de x;

Implementação através de Listas Encadeadas

Implementação de **union**(**x**, **y**):

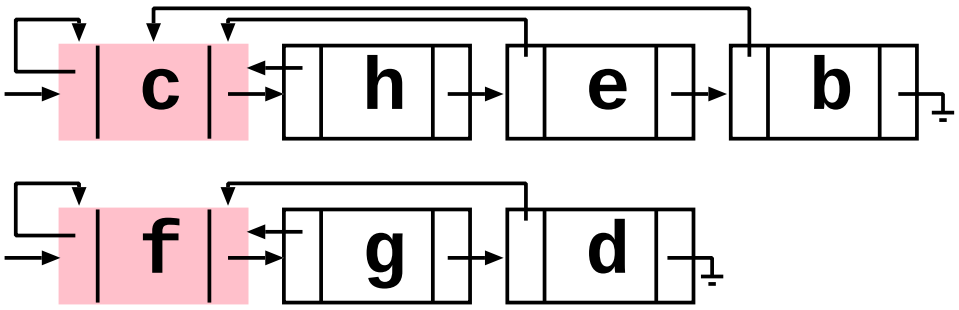
- lista de *x* anexada ao final da lista de *y*;
- representante da nova lista é *y*;
- nós de *x* devem apontar para representante *y*: custa comprimento da lista de *x*;



Implementação através de Listas Encadeadas

Implementação de **union**(**x**, **y**):

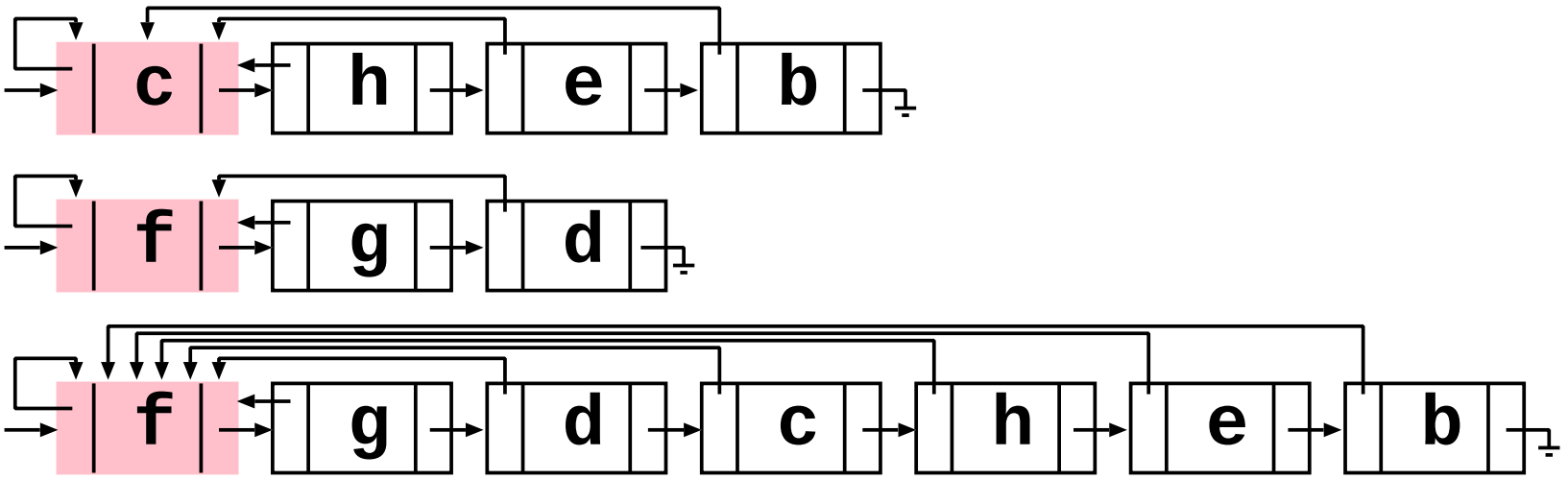
- lista de *x* anexada ao final da lista de *y*;
- representante da nova lista é *y*;
- nós de *x* devem apontar para representante *y*: custa comprimento da lista de *x*;



Implementação através de Listas Encadeadas

Implementação de **union**(**x**, **y**):

- lista de *x* anexada ao final da lista de *y*;
- representante da nova lista é *y*;
- nós de *x* devem apontar para representante *y*: custa comprimento da lista de *x*;



Implementação através de Listas Encadeadas

Implementação de **union**$(\mathbf{x}, \mathbf{y})$:

Implementação através de Listas Encadeadas

Implementação de **union**(**x**, **y**):

- **usando heurística de união ponderada:**

Implementação através de Listas Encadeadas

Implementação de **union**$(\mathbf{x}, \mathbf{y})$:

- **usando heurística de união ponderada:** anexa lista menor à maior;

Implementação por Florestas de Conjuntos Disjuntos

Implementação por Florestas de Conjuntos Disjuntos

- Implementação mais rápida que por listas encadeadas;

Implementação por Florestas de Conjuntos Disjuntos

- Implementação mais rápida que por listas encadeadas;
- estrutura de dados de conjuntos disjuntos assintoticamente **mais rápida** conhecida;

Implementação por Florestas de Conjuntos Disjuntos

- Implementação mais rápida que por listas encadeadas;
- estrutura de dados de conjuntos disjuntos assintoticamente **mais rápida** conhecida;
- conjuntos representados por árvores enraizadas:

Implementação por Florestas de Conjuntos Disjuntos

- Implementação mais rápida que por listas encadeadas;
- estrutura de dados de conjuntos disjuntos assintoticamente **mais rápida** conhecida;
- conjuntos representados por árvores enraizadas:
  1. cada nó contém um elemento;

Implementação por Florestas de Conjuntos Disjuntos

- Implementação mais rápida que por listas encadeadas;
- estrutura de dados de conjuntos disjuntos assintoticamente **mais rápida** conhecida;
- conjuntos representados por árvores enraizadas:
  1. cada nó contém um elemento;
  2. cada nó aponta somente para seu pai;

Implementação por Florestas de Conjuntos Disjuntos

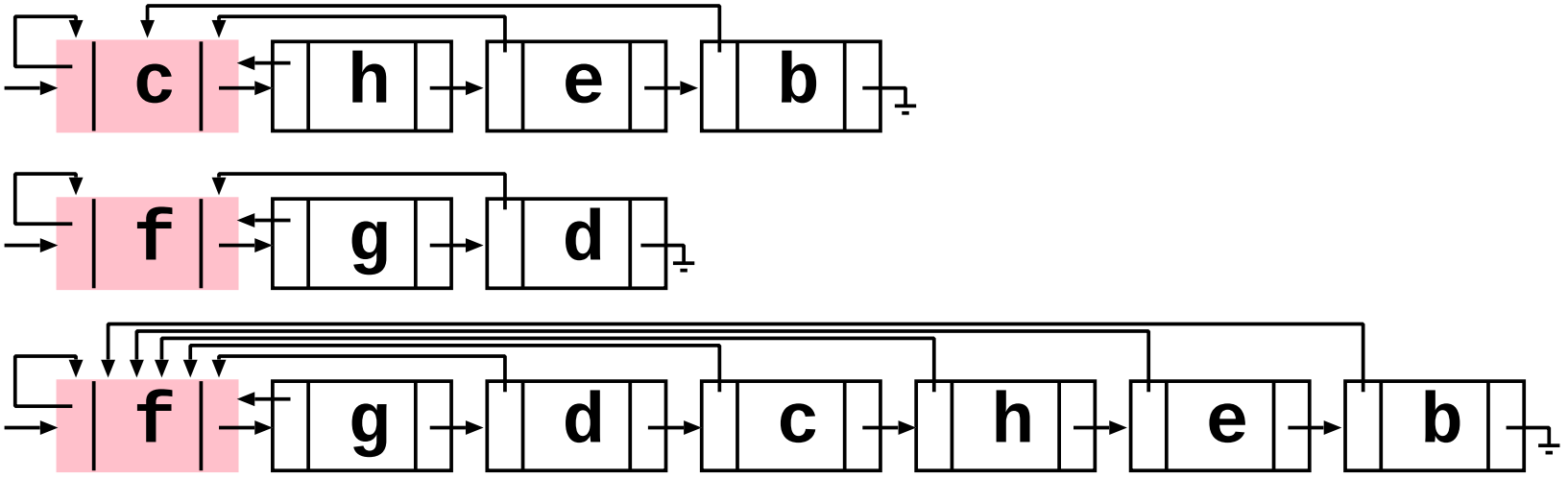
- Implementação mais rápida que por listas encadeadas;
- estrutura de dados de conjuntos disjuntos assintoticamente **mais rápida** conhecida;
- conjuntos representados por árvores enraizadas:
  1. cada nó contém um elemento;
  2. cada nó aponta somente para seu pai;
  3. cada árvore representa um conjunto;

Implementação por Florestas de Conjuntos Disjuntos

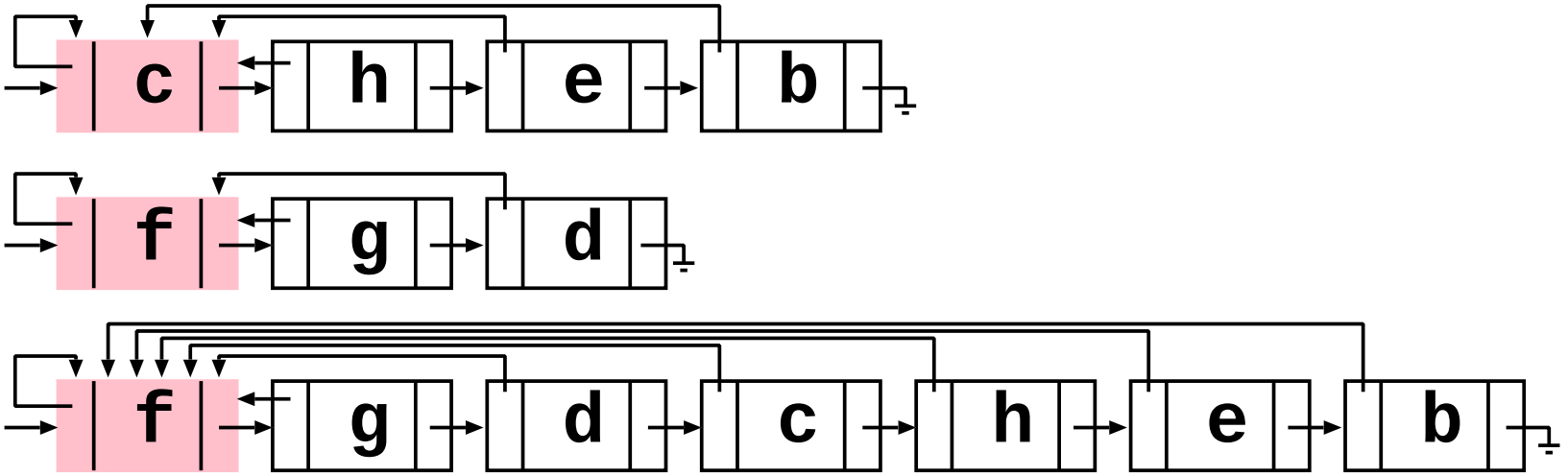
- Implementação mais rápida que por listas encadeadas;
- estrutura de dados de conjuntos disjuntos assintoticamente **mais rápida** conhecida;
- conjuntos representados por árvores enraizadas:
  1. cada nó contém um elemento;
  2. cada nó aponta somente para seu pai;
  3. cada árvore representa um conjunto;
  4. **representante do conjunto**: raiz da árvore;

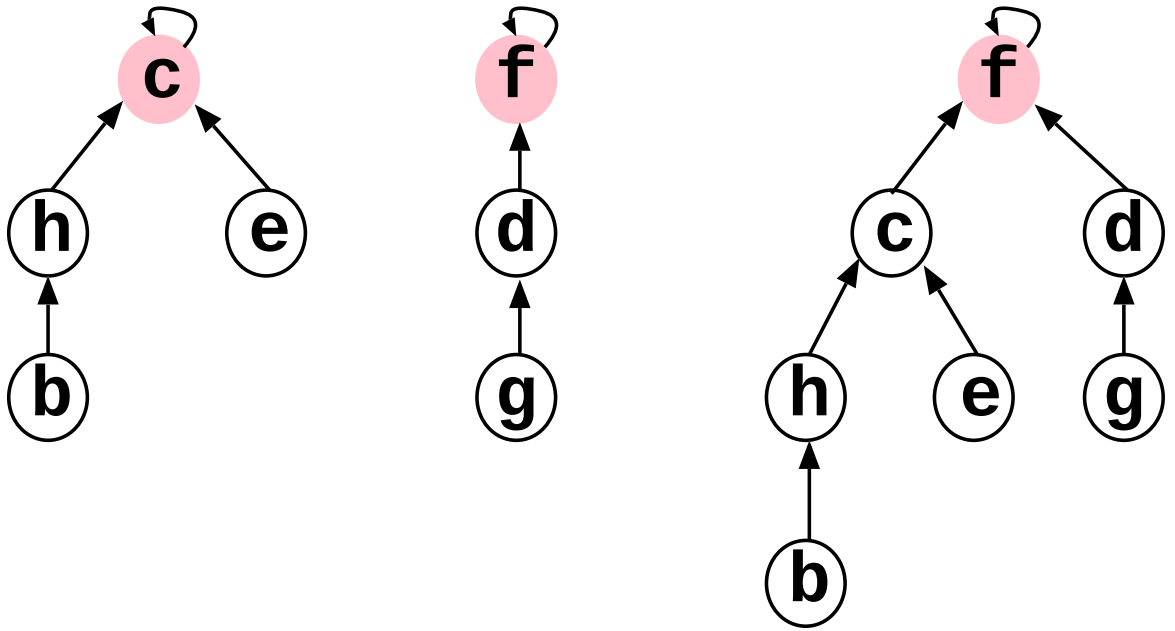
Implementação por Florestas de Conjuntos Disjuntos

Implementação por Florestas de Conjuntos Disjuntos



Implementação por Florestas de Conjuntos Disjuntos





Implementação por Florestas de Conjuntos Disjuntos

Implementação por Florestas de Conjuntos Disjuntos

- **makeSet**(**x**): cria árvore com um nó contendo *x*;

Implementação por Florestas de Conjuntos Disjuntos

- **makeSet**(**x**): cria árvore com um nó contendo x;
- **union**(**x**, **y**): raiz da árvore de x aponta para a raiz da árvore de y;

Implementação por Florestas de Conjuntos Disjuntos

- **makeSet**(**x**): cria árvore com um nó contendo x;
- **union**(**x**, **y**): raiz da árvore de x aponta para a raiz da árvore de y;
- **findSet**(**x**): segue ponteiros de pais até encontrar a raiz da árvore;

Implementação por Florestas de Conjuntos Disjuntos

- **makeSet(x)**: cria árvore com um nó contendo x;
- **union(x, y)**: raiz da árvore de x aponta para a raiz da árvore de y;
- **findSet(x)**: segue ponteiros de pais até encontrar a raiz da árvore;
- sequência de $n - 1$ operações **union(x, y)** pode criar árvore que é lista;

Implementação por Florestas de Conjuntos Disjuntos

- **makeSet(x)**: cria árvore com um nó contendo x;
- **union(x, y)**: raiz da árvore de x aponta para a raiz da árvore de y;
- **findSet(x)**: segue ponteiros de pais até encontrar a raiz da árvore;
- sequência de $n - 1$ operações **union(x, y)** pode criar árvore que é lista;
- uso de duas heurísticas para melhorar desempenho:

Implementação por Florestas de Conjuntos Disjuntos

- **makeSet(x)**: cria árvore com um nó contendo x;
- **union(x, y)**: raiz da árvore de x aponta para a raiz da árvore de y;
- **findSet(x)**: segue ponteiros de pais até encontrar a raiz da árvore;
- sequência de $n - 1$ operações **union(x, y)** pode criar árvore que é lista;
- uso de duas heurísticas para melhorar desempenho:
  1. **união por ordenação**: raiz da menor árvore aponta para raiz da maior árvore;

Implementação por Florestas de Conjuntos Disjuntos

- **makeSet(x)**: cria árvore com um nó contendo x;
- **union(x, y)**: raiz da árvore de x aponta para a raiz da árvore de y;
- **findSet(x)**: segue ponteiros de pais até encontrar a raiz da árvore;
- sequência de $n - 1$ operações **union(x, y)** pode criar árvore que é lista;
- uso de duas heurísticas para melhorar desempenho:
  1. **união por ordenação**: raiz da menor árvore aponta para raiz da maior árvore;
  2. **compressão de caminho**: cada nó aponta diretamente para a raiz;

Aplicação: árvore geradora mínima

Aplicação: árvore geradora mínima

- **projeto de circuitos eletrônicos:** tornar pinos de componentes eletricamente equivalentes juntando a fiação de todos eles.

Aplicação: árvore geradora mínima

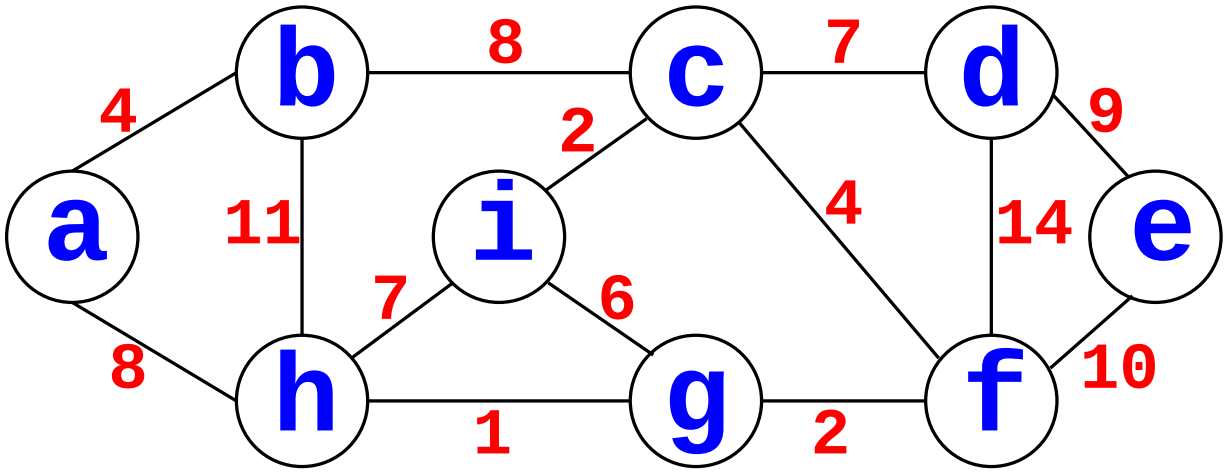
- **projeto de circuitos eletrônicos:** tornar pinos de componentes eletricamente equivalentes juntando a fiação de todos eles.
- **problema:** queremos minimizar a quantidade de fios;

Aplicação: árvore geradora mínima

- **projeto de circuitos eletrônicos:** tornar pinos de componentes eletricamente equivalentes juntando a fiação de todos eles.
- **problema:** queremos minimizar a quantidade de fios;
- **problema modelado por grafos:** árvore geradora mínima;

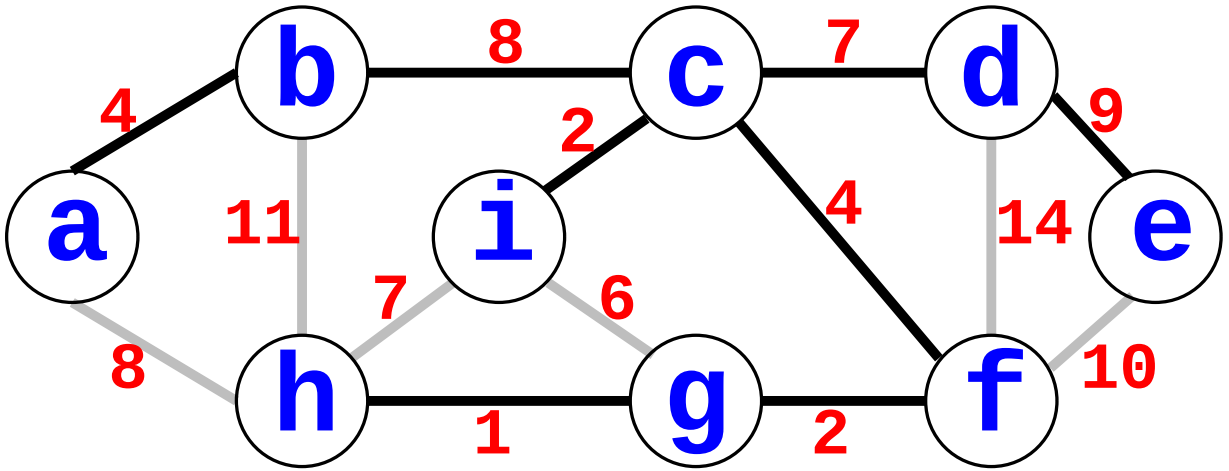
Aplicação: árvore geradora mínima

- **projeto de circuitos eletrônicos:** tornar pinos de componentes eletricamente equivalentes juntando a fiação de todos eles.
- **problema:** queremos minimizar a quantidade de fios;
- **problema modelado por grafos:** árvore geradora mínima;



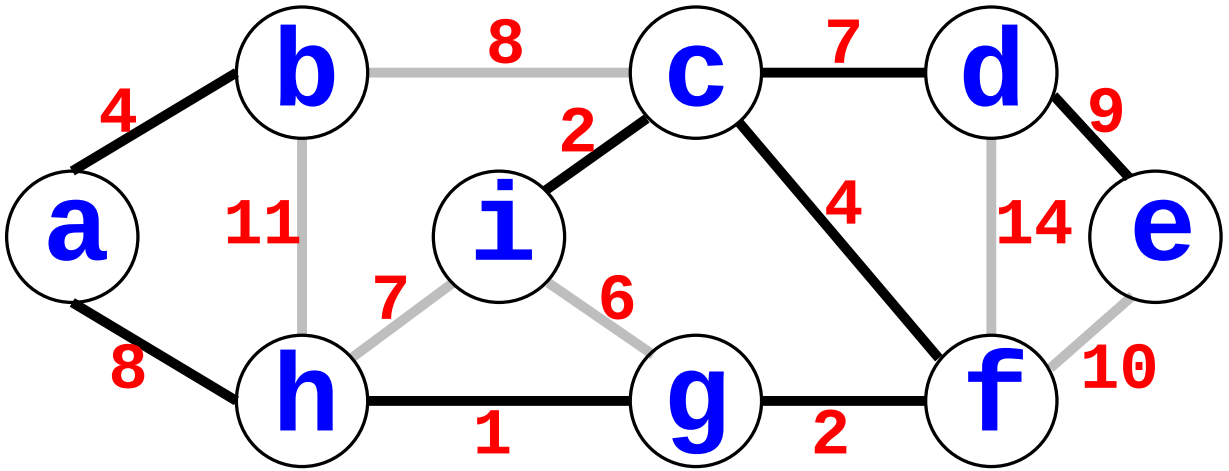
Aplicação: árvore geradora mínima

- **projeto de circuitos eletrônicos:** tornar pinos de componentes eletricamente equivalentes juntando a fiação de todos eles.
- **problema:** queremos minimizar a quantidade de fios;
- **problema modelado por grafos:** árvore geradora mínima;



Aplicação: árvore geradora mínima

- **projeto de circuitos eletrônicos:** tornar pinos de componentes eletricamente equivalentes juntando a fiação de todos eles.
- **problema:** queremos minimizar a quantidade de fios;
- **problema modelado por grafos:** árvore geradora mínima;



Árvore geradora mínima: algoritmo geral

Árvore geradora mínima: algoritmo geral

Estratégia gulosa:

Árvore geradora mínima: algoritmo geral

Estratégia gulosa:

```
1  generico(G):
2      A ← ∅
3      enquanto A não é árvore geradora faça
4          encontre aresta (u, v) segura para A
5          A ← A ∪ {(u, v)}
6      retorne A
```

Árvore geradora mínima: algoritmo geral

Estratégia gulosa:

```
1  generico(G):
2      A ← ∅
3      enquanto A não é árvore geradora faça
4          encontre aresta (u, v) segura para A
5          A ← A ∪ {(u, v)}
6      retorne A
```

Aresta segura: não cria ciclo em A

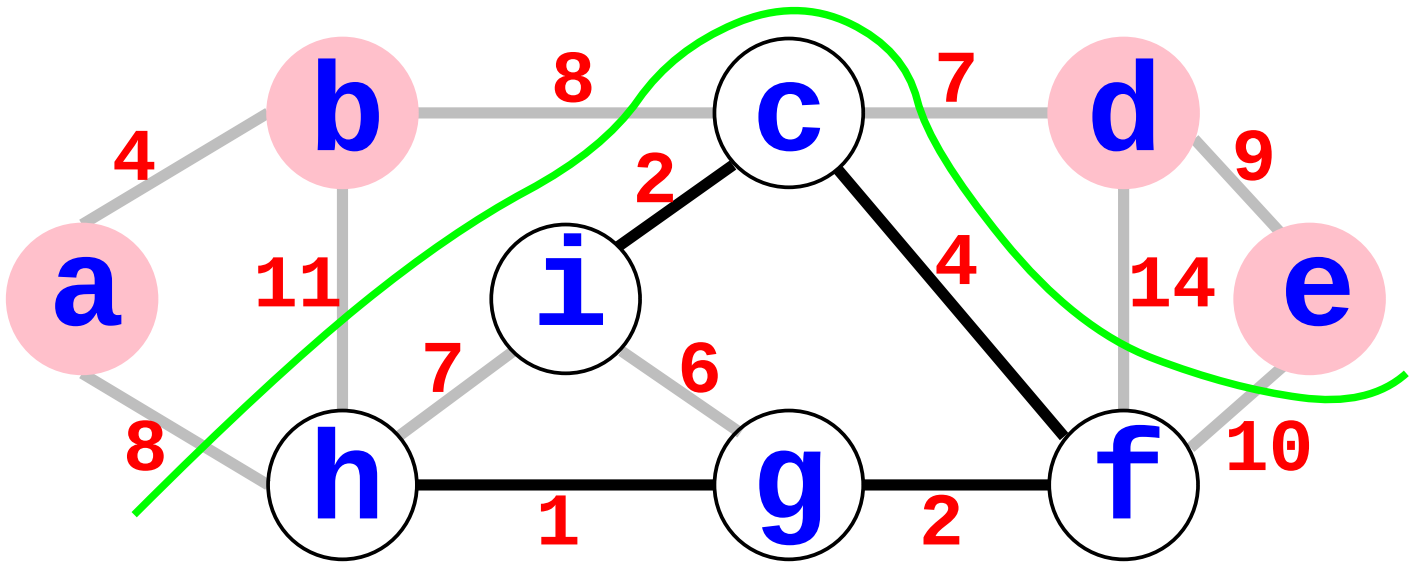
Árvore geradora mínima: algoritmo geral

Árvore geradora mínima: algoritmo geral

Aresta leve: aresta de menor peso que atravessa um corte

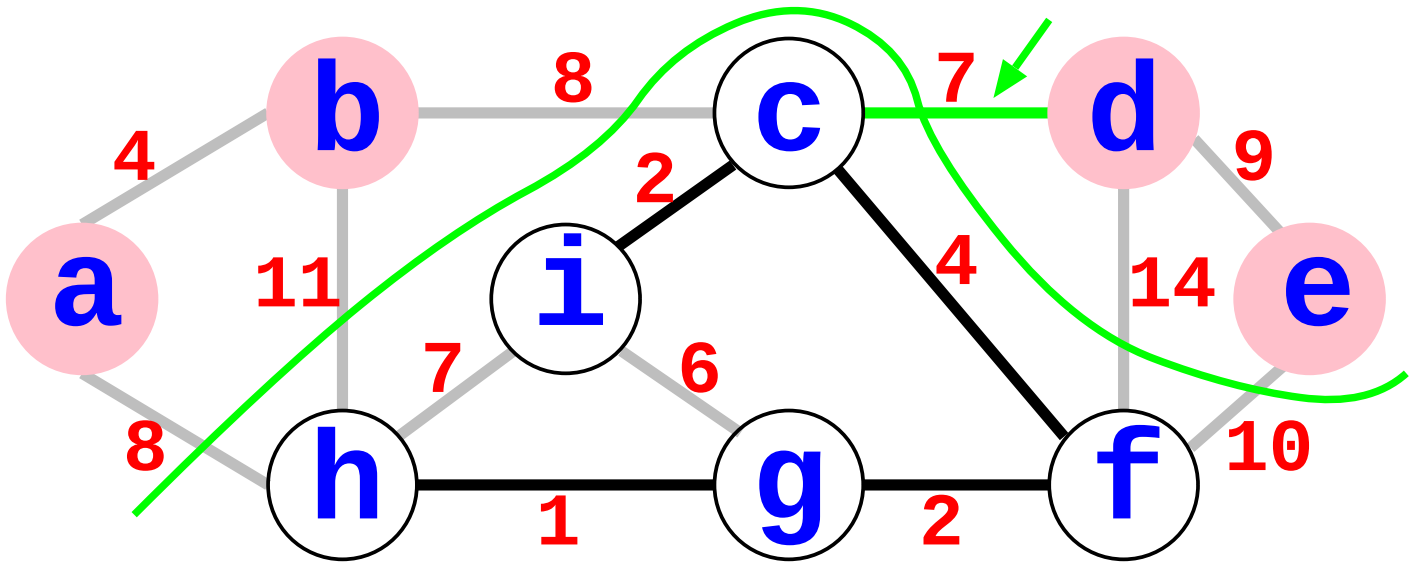
Árvore geradora mínima: algoritmo geral

Aresta leve: aresta de menor peso que atravessa um corte

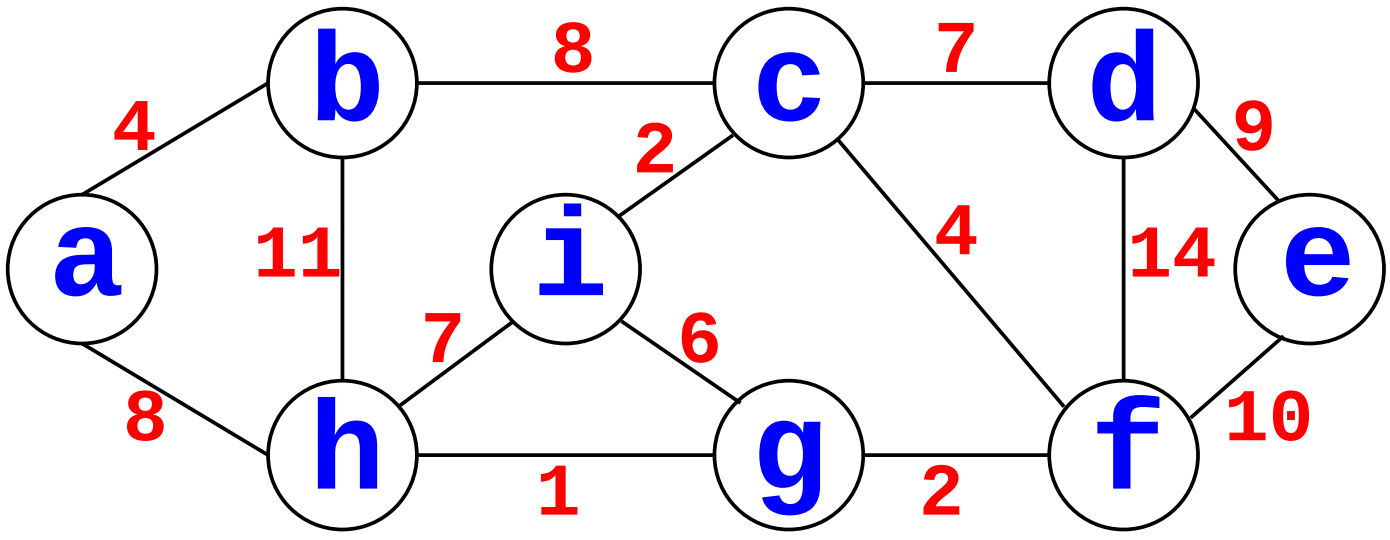


Árvore geradora mínima: algoritmo geral

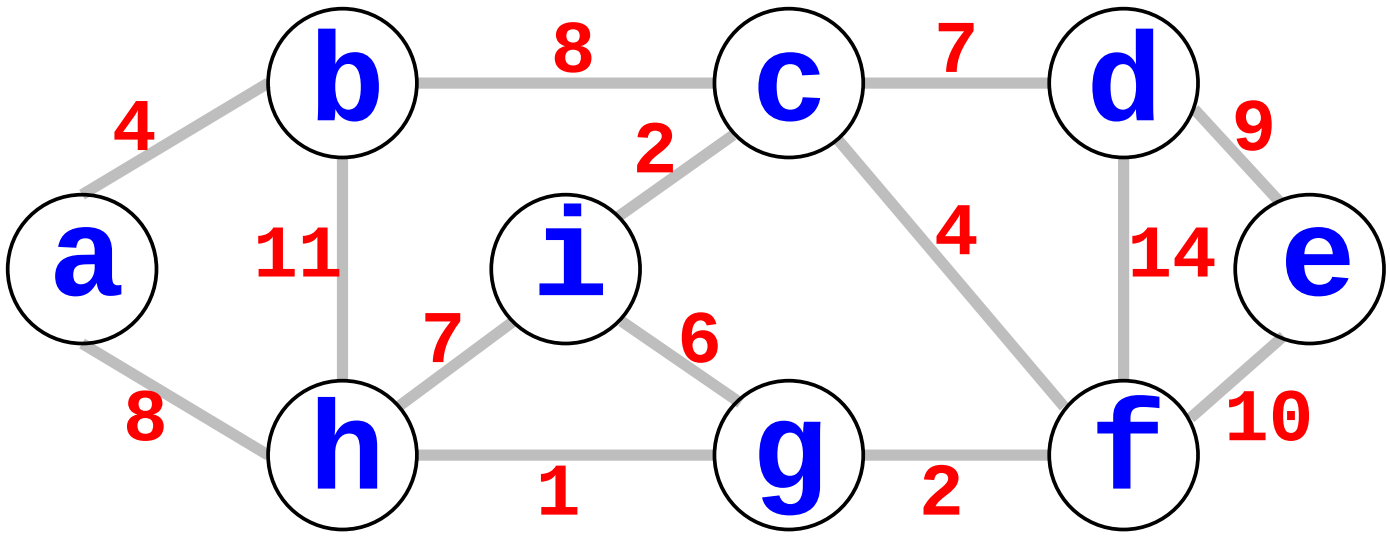
Aresta leve: aresta de menor peso que atravessa um corte



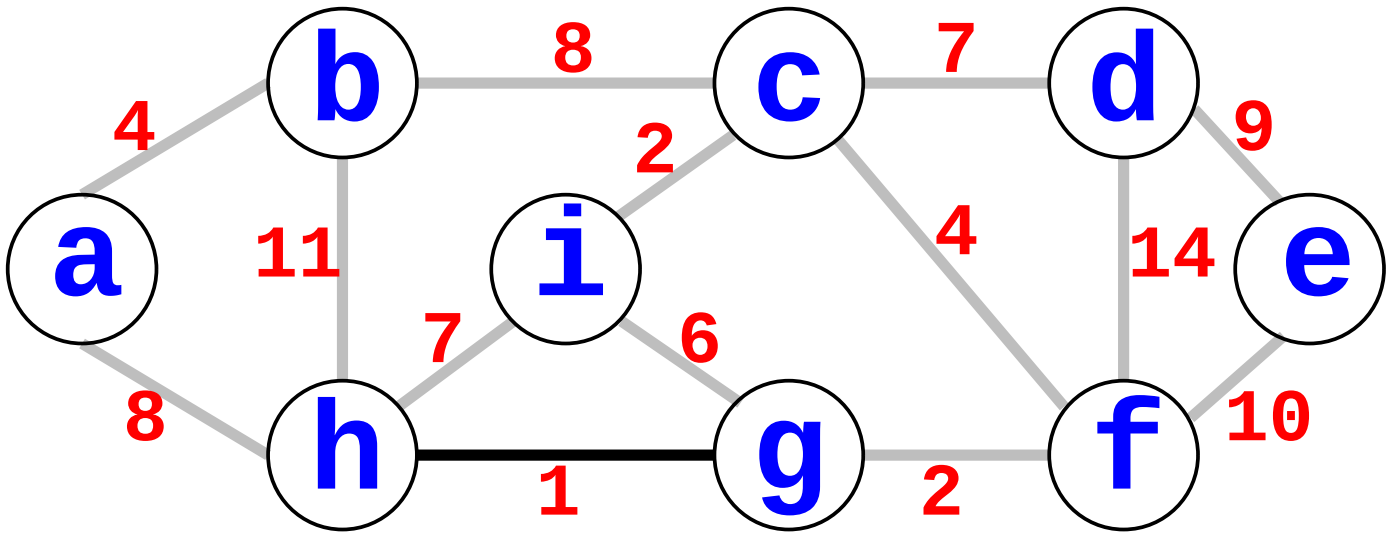
Árvore geradora mínima: exemplo



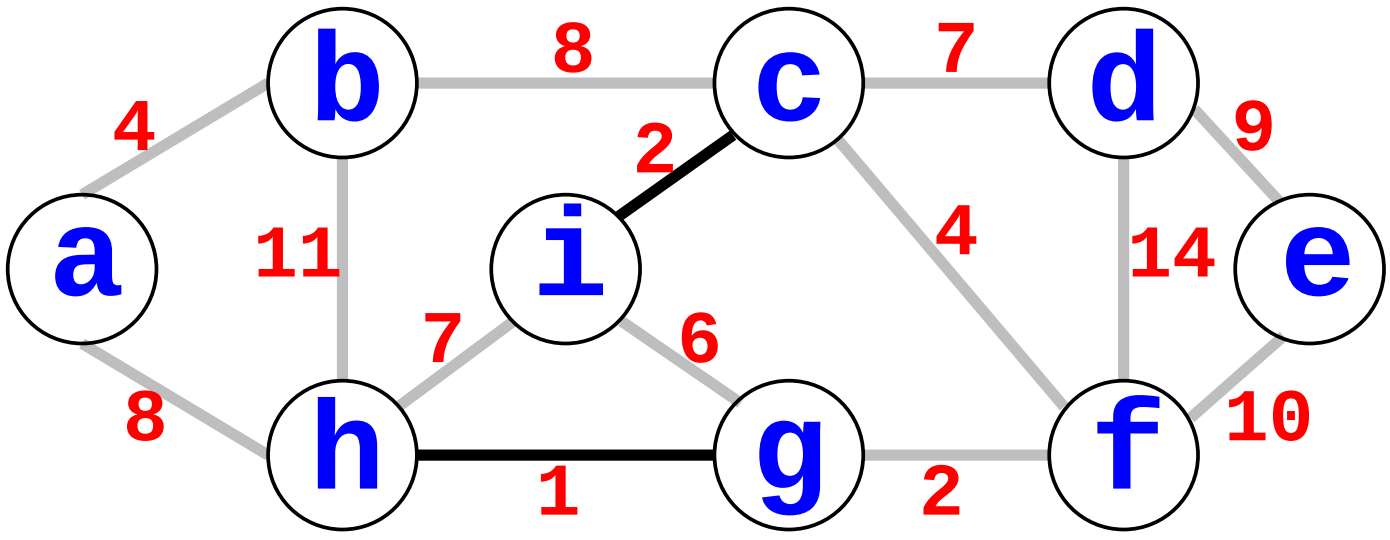
Árvore geradora mínima: exemplo



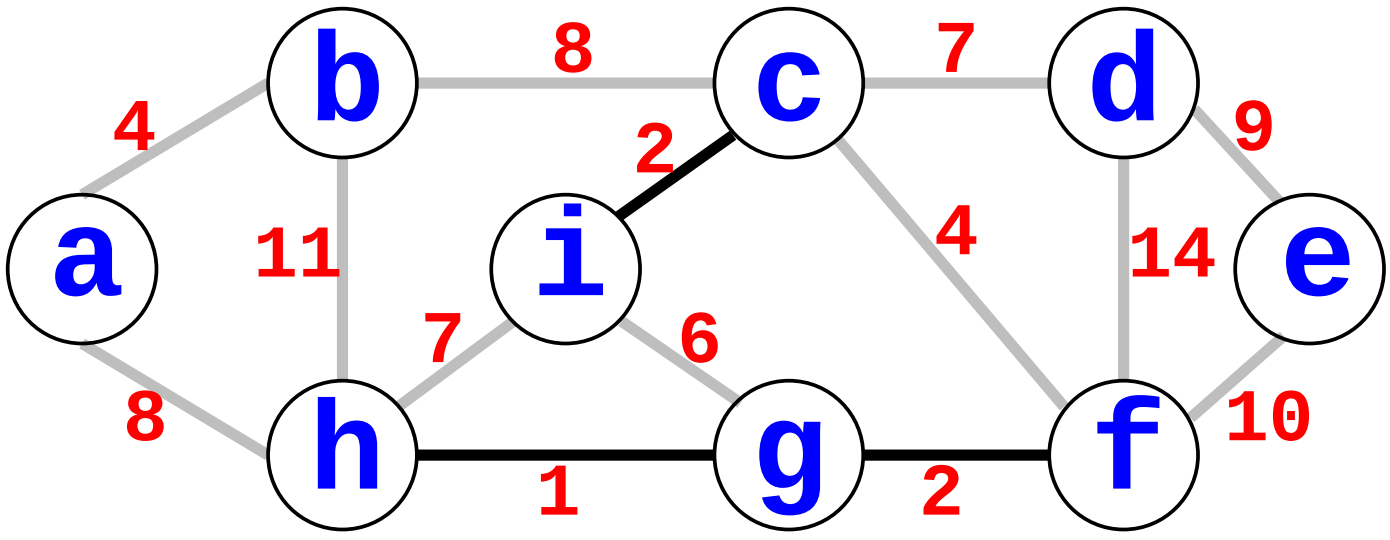
Árvore geradora mínima: exemplo



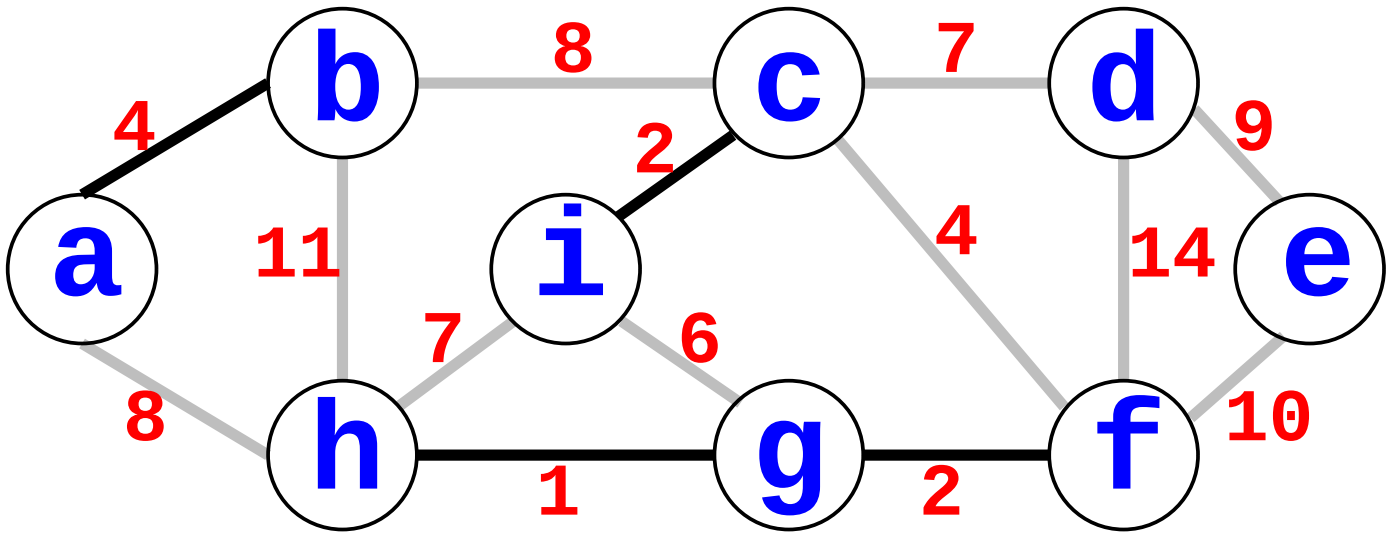
Árvore geradora mínima: exemplo



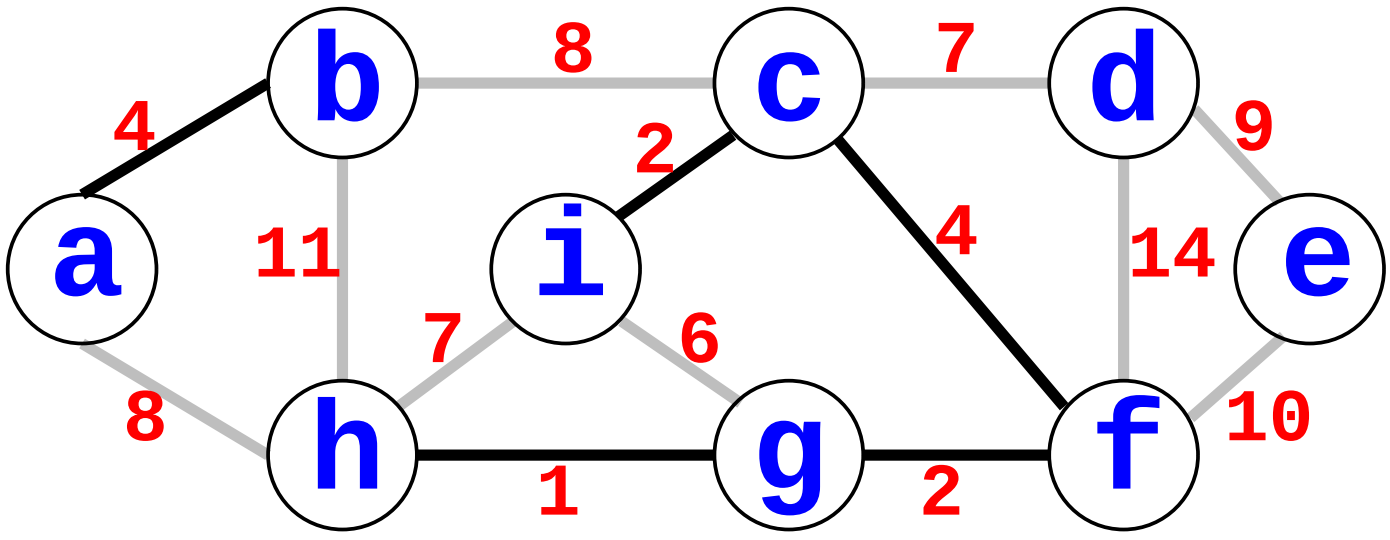
Árvore geradora mínima: exemplo



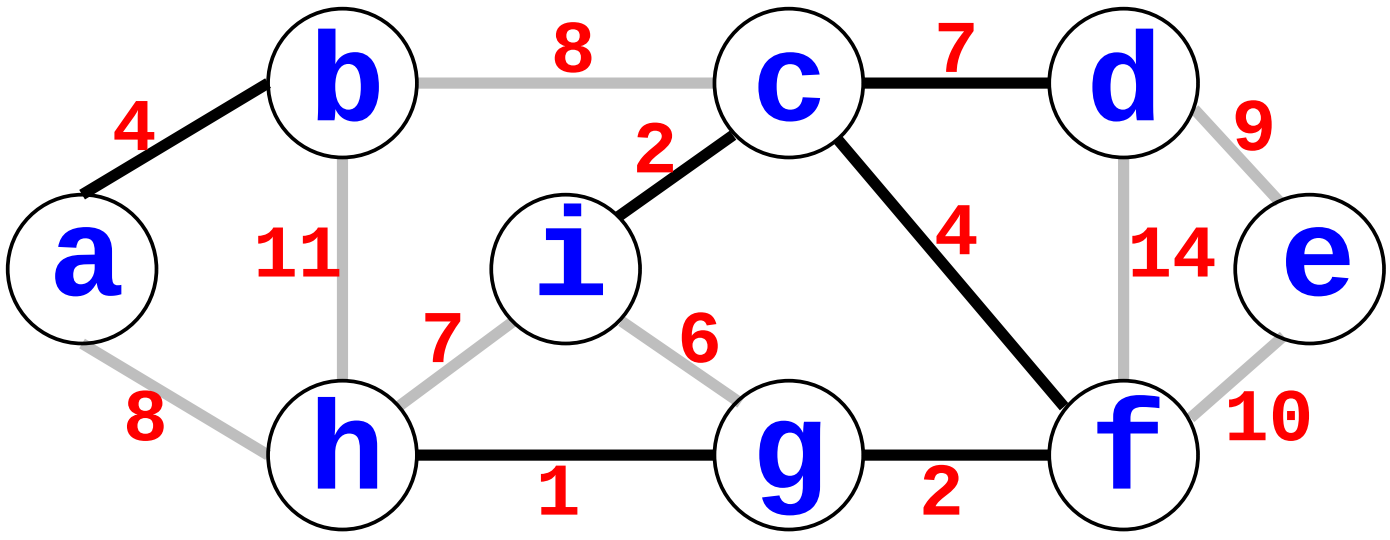
Árvore geradora mínima: exemplo



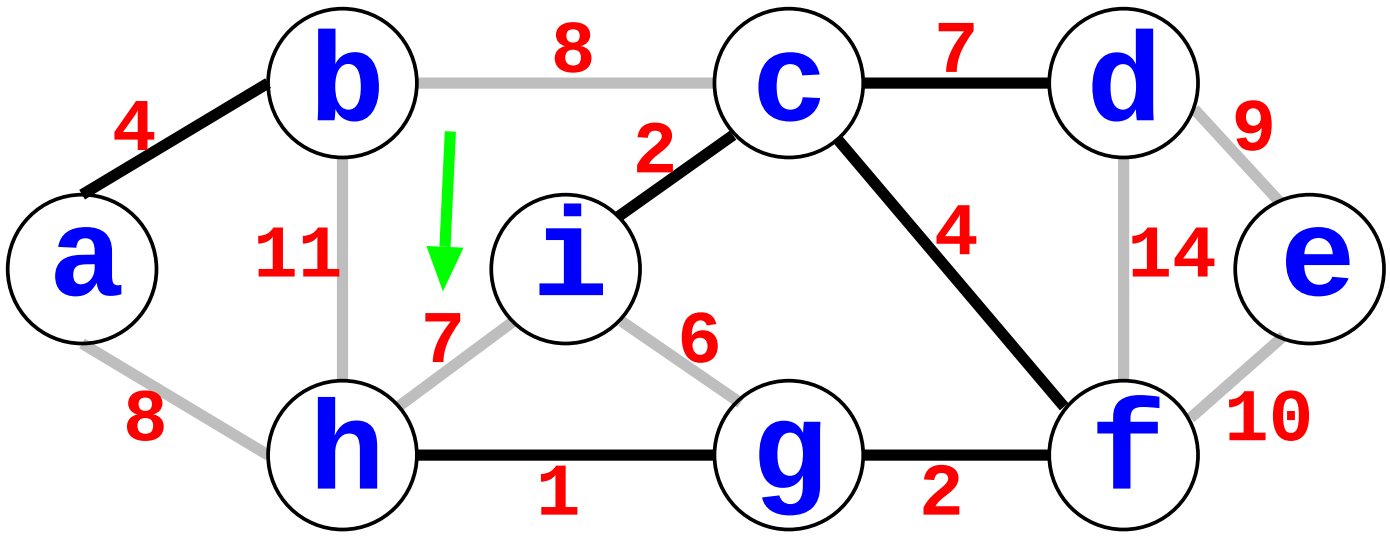
Árvore geradora mínima: exemplo



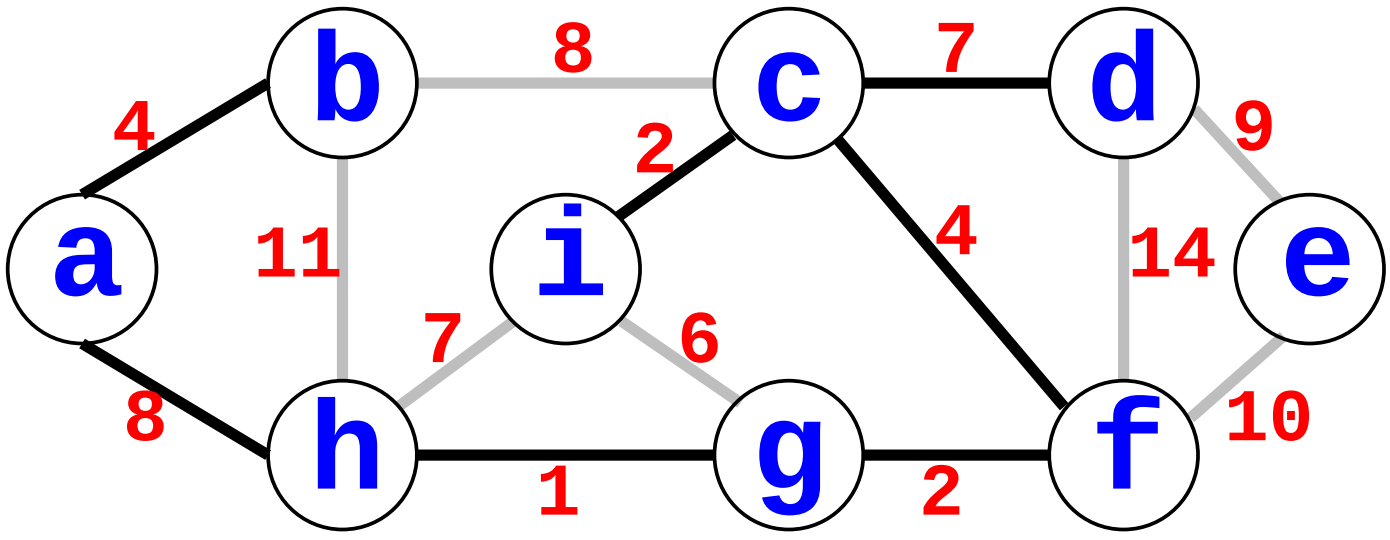
Árvore geradora mínima: exemplo



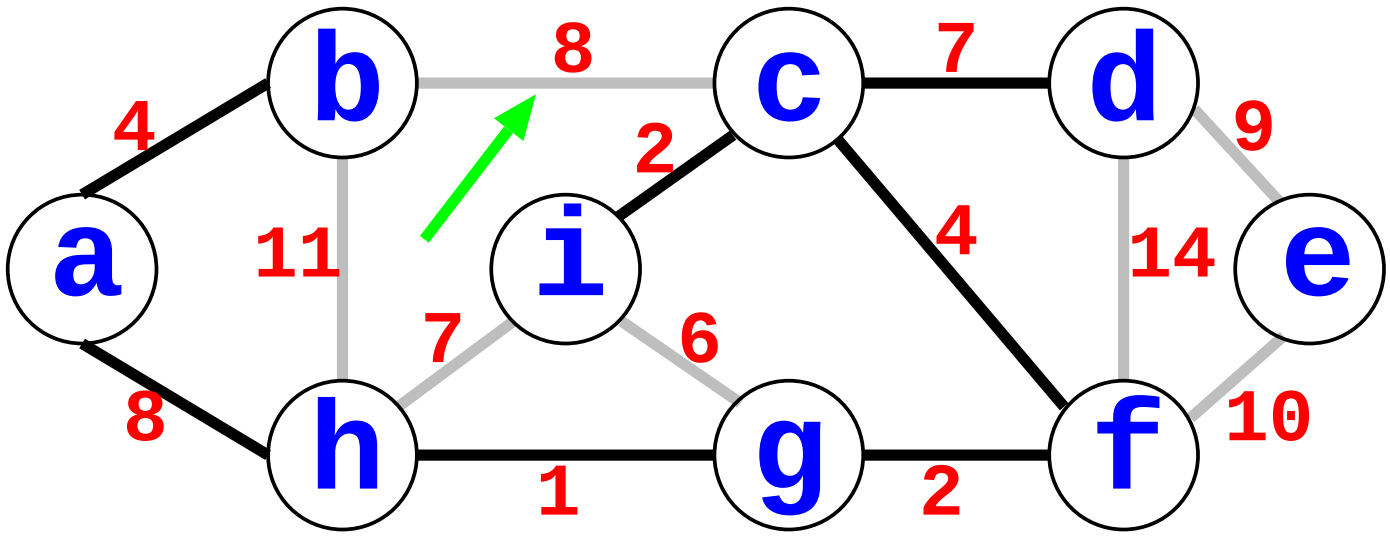
Árvore geradora mínima: exemplo



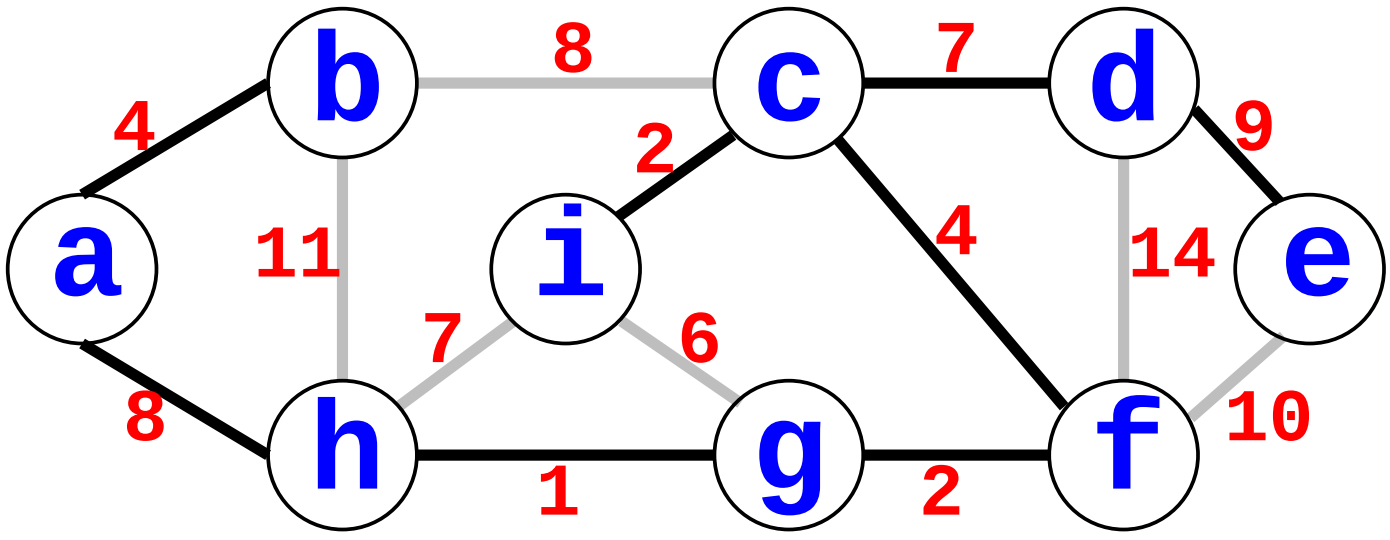
Árvore geradora mínima: exemplo



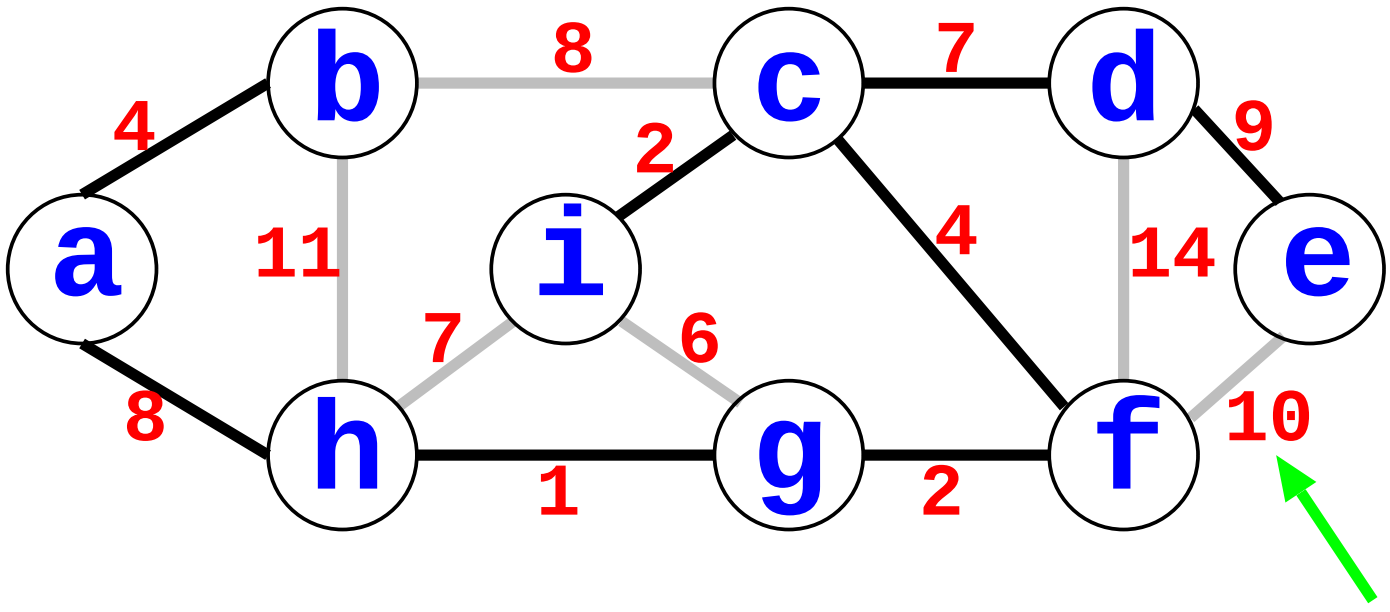
Árvore geradora mínima: exemplo



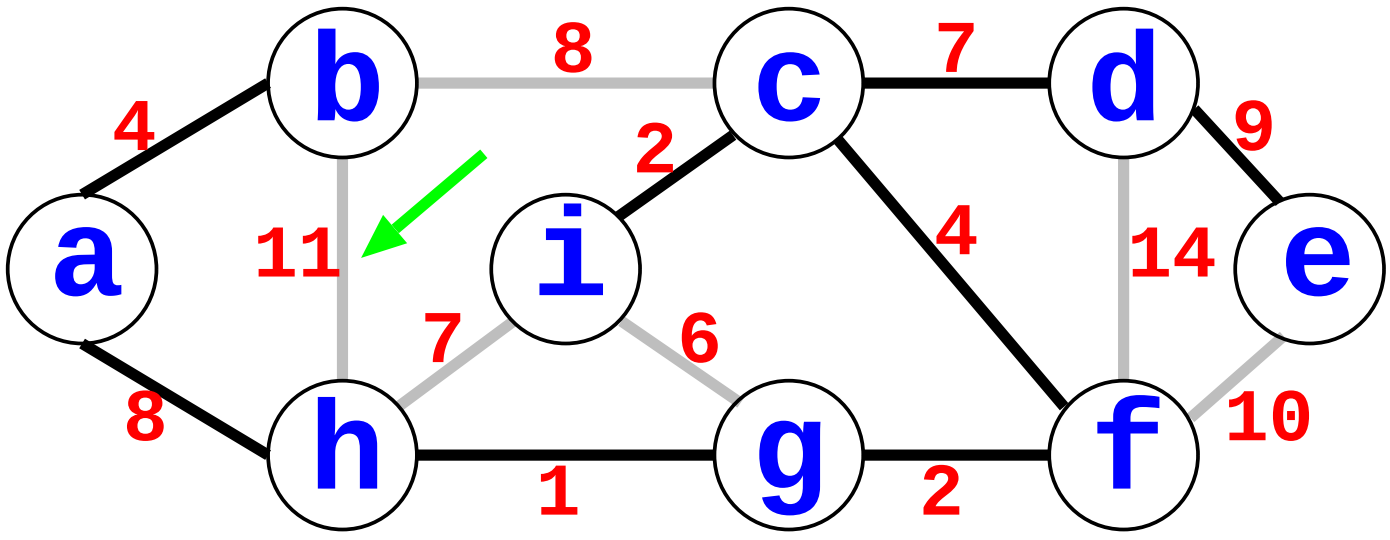
Árvore geradora mínima: exemplo



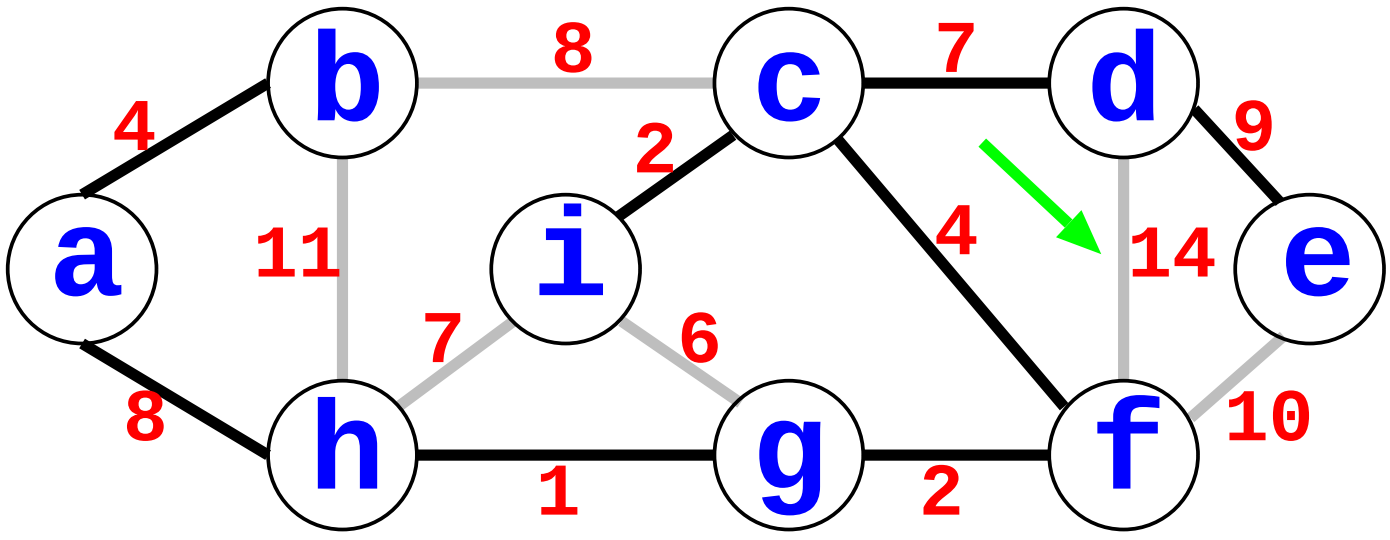
Árvore geradora mínima: exemplo



Árvore geradora mínima: exemplo



Árvore geradora mínima: exemplo



Árvore geradora mínima: algoritmo de Kruskal

Árvore geradora mínima: algoritmo de Kruskal

Implementação usando union-find:

Árvore geradora mínima: algoritmo de Kruskal

Implementação usando union-find:

```
1  kruskal(G):
2      A ← ∅
3      para cada vértice v ∈ V(G) faça
4          makeSet(v)
5      ordene arestas de E por peso w crescente
6      para cada aresta (u, v) ∈ E(G) em ordem crescente faça
7          se findSet(u) ≠ findSet(v) então
8              union(u, v)
9      retorne A
```

Exercícios

1. Escreva o pseudocódigo para **makeSet**(**x**), **union**(**x**, **y**) e **findSet**(**x**) usando a representação de lista ligada e a heurística de união ponderada Suponha que cada objeto *x* tenha um atributo *rep* apontando para o representante do conjunto que contém *x*, e que cada conjunto *S* tem atributos *inicio*, *fim* e *tamanho* (que é igual ao comprimento da lista).
2. A árvore fornecida pelo algoritmo de Kruskal é única, ou seja, o algoritmo sempre fornecerá a mesma árvore geradora mínima?

Bibliografia

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. e STEIN, C. *Introduction to Algorithms*, 3ª edição, MIT Press, 2009.

ZIVIANI, N. Projeto de Algoritmos: com implementações em Java e C++, 1a edição, Cengage Learning, 2009.