Grafos: árvore geradora mínima

Prof. Jefferson T. Oliva
Material da Profa. Luciene de Oliveira Marin

Algoritmos e Estrutura de Dados II (AE23CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

DAINF
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CÂMPUS PATO BRANCO

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Árvore Geradora Mínima (*Minimum Spanning Tree*)

Contextualização

Suponha que queremos resolver o seguinte problema:

> *dado um conjunto de computadores, onde cada par de computadores pode ser ligado usando uma quantidade de fibra ótica, encontrar uma rede interconectando-os que use a menor quantidade de fibra ótica possível.*

Este problema pode ser modelado através de um grafo não orientado ponderado onde os vértices representam os computadores, as arestas representam as conexões que podem ser construídas e o peso/custo de uma aresta representa a quantidade de fibra ótica necessária.

Árvore Geradora Mínima

Abstração do problema

- Nessa modelagem, o problema que queremos resolver é encontrar um **subgrafo gerador** (que contém todos os vértices do grafo original), **conexo** (para garantir a interligação de todas as cidades) e cuja soma dos custos de suas arestas seja a menor possível.
- Obviamente, o problema só tem solução se o **grafo** for **conexo**. Daqui pra frente vamos supor que o grafo de entrada é conexo.
- Além disso, o sugrafo gerador procurado é sempre uma árvore (supondo que os pesos são positivos).

Árvore Geradora Mínima

Problema da árvore geradora mínima

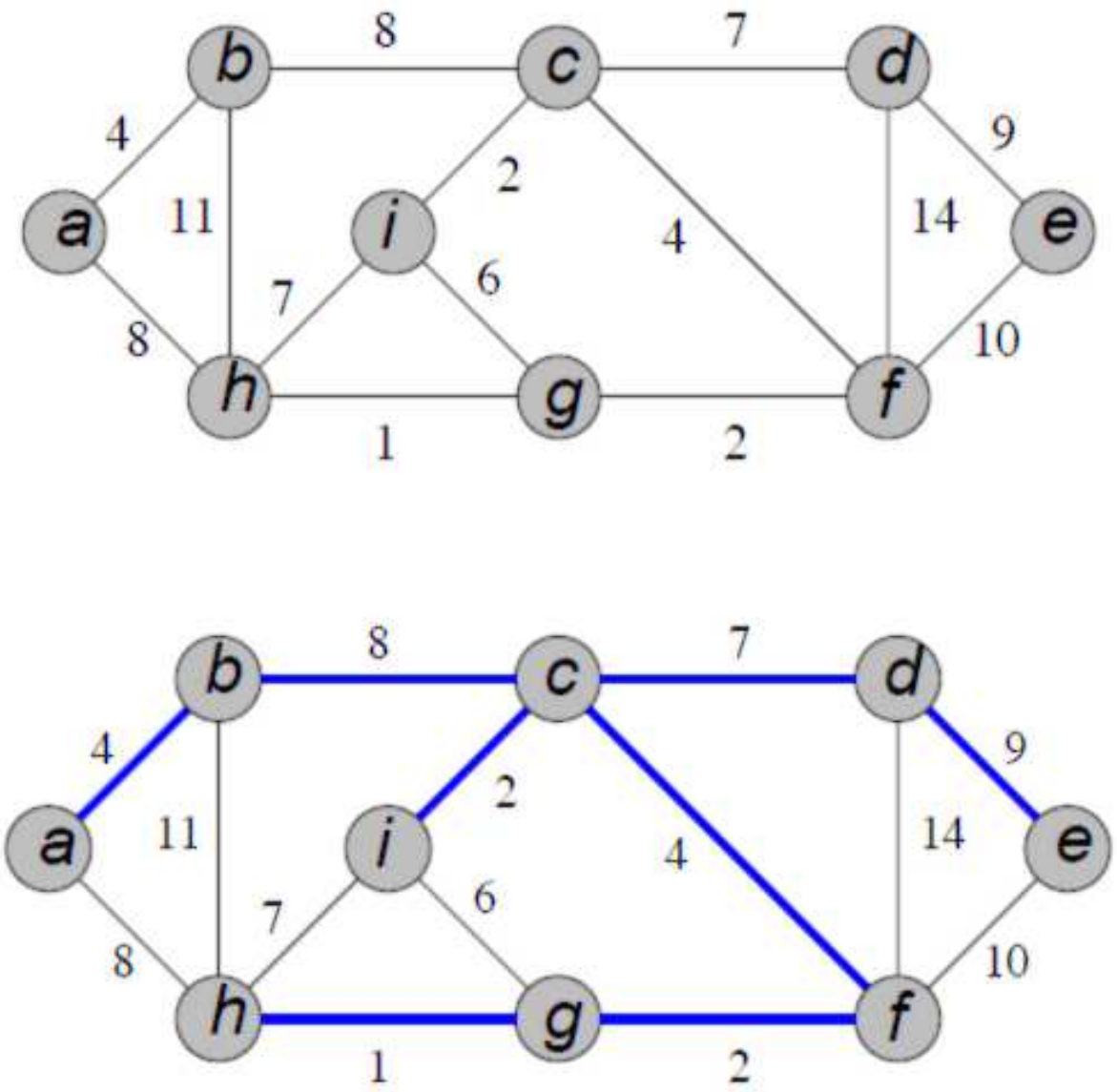
Entrada: grafo conexo $G = (V, E)$ com pesos $w(u, v)$ para cada aresta (u, v).

Saída: subgrafo gerador conexo T de G cujo peso total

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

seja o menor possível.

Árvore Geradora Mínima - exemplo



Árvore Geradora Mínima

Veremos dois algoritmos para resolver o problema:

- **algoritmo de Prim**
- **algoritmo de Kruskal**

Ambos algoritmos usam **estratégia gulosa**. Eles são exemplos clássicos de algoritmos gulosos.

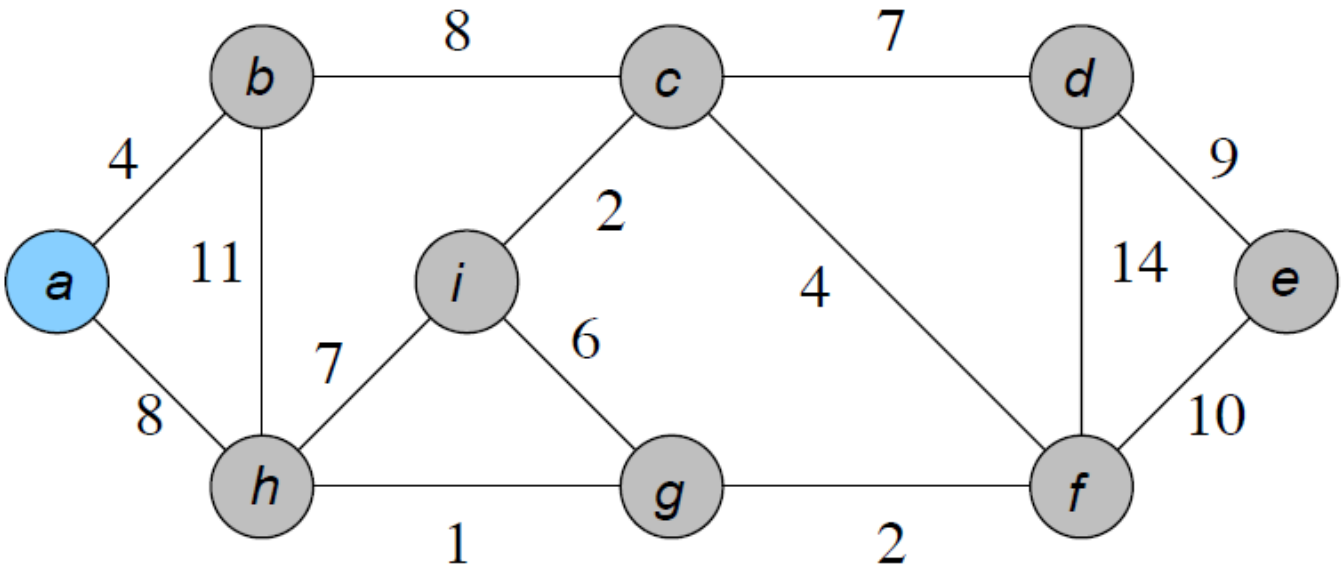
O algoritmo de Prim

O algoritmo de Prim

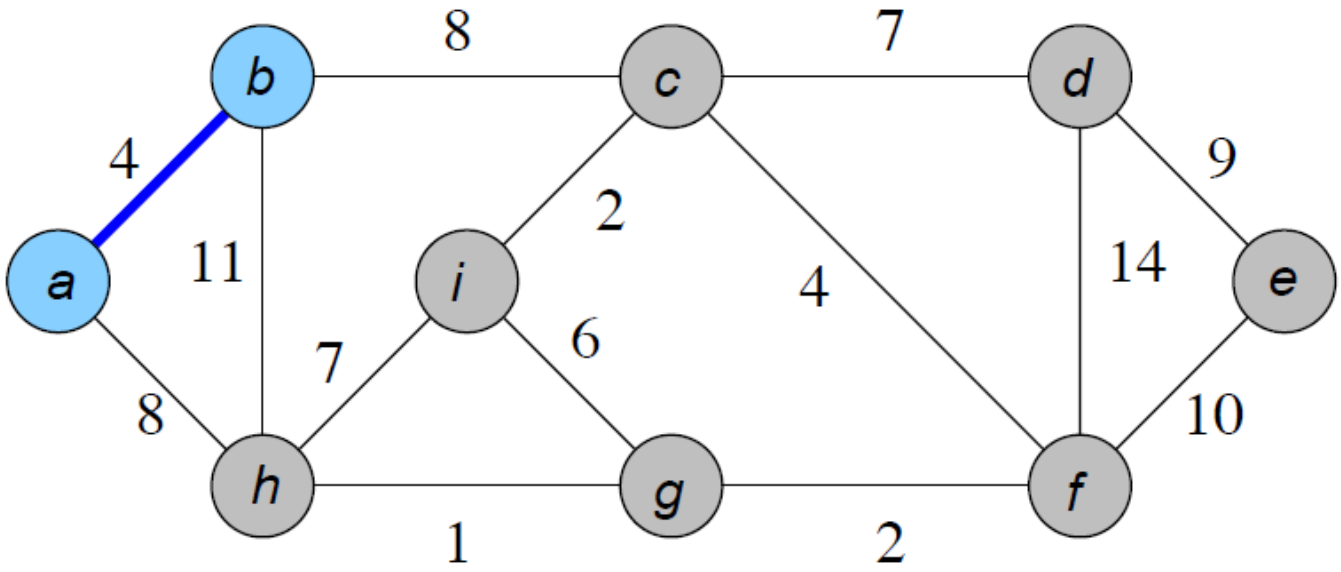
- No algoritmo de Prim, o conjunto A é uma árvore com raiz r (escolhido arbitrariamente no início). Inicialmente, A é vazio.
- Em cada iteração, o algoritmo considera o corte $\delta(C)$ onde C é o conjunto de vértices que são extremos de A.
- Ele encontra uma aresta leve (u, v) neste corte e acrescenta-a ao conjunto A e começa outra iteração até que A seja uma árvore geradora.

Um detalhe de implementação importante é como encontrar eficientemente uma aresta leve no corte.

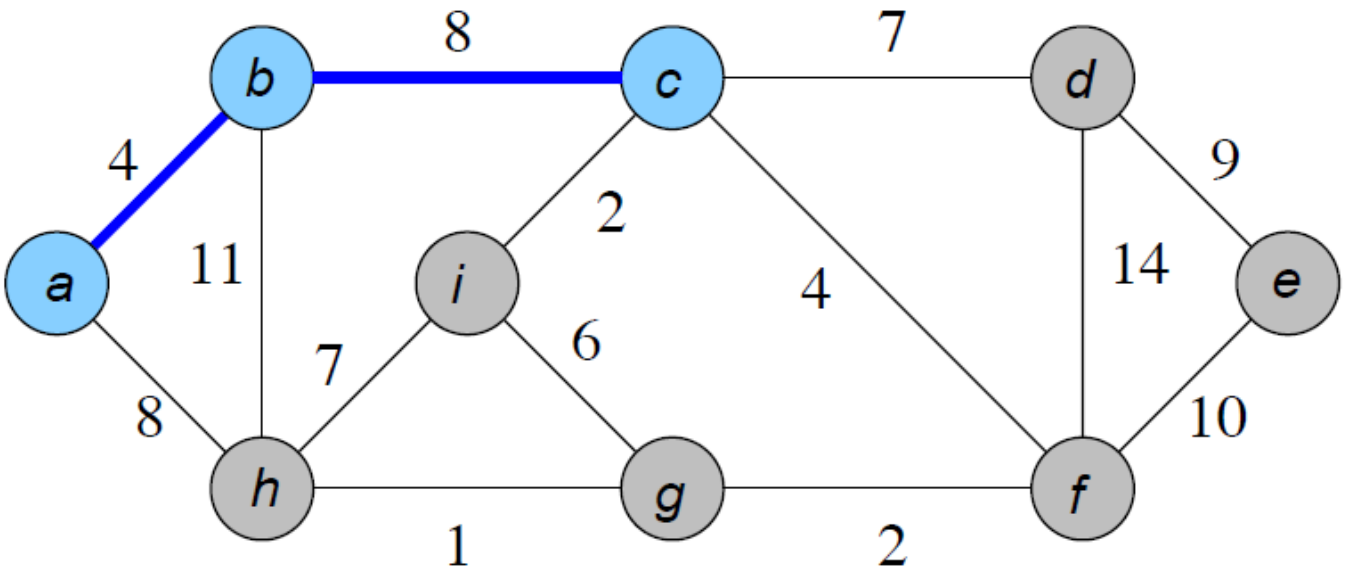
O algoritmo de Prim



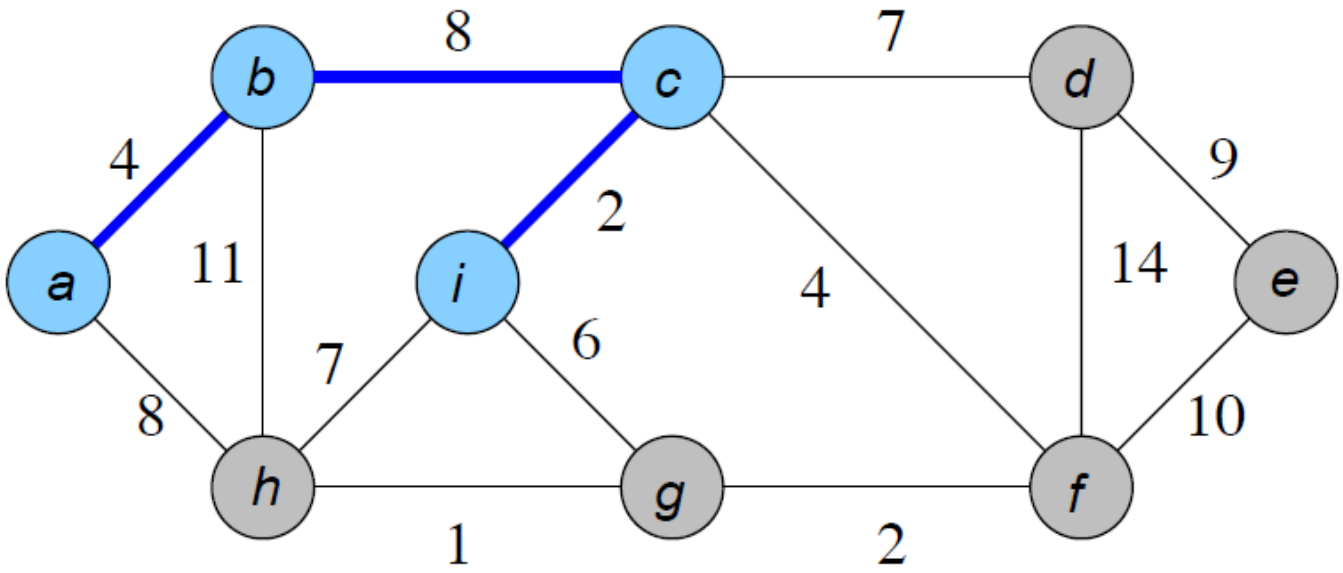
O algoritmo de Prim



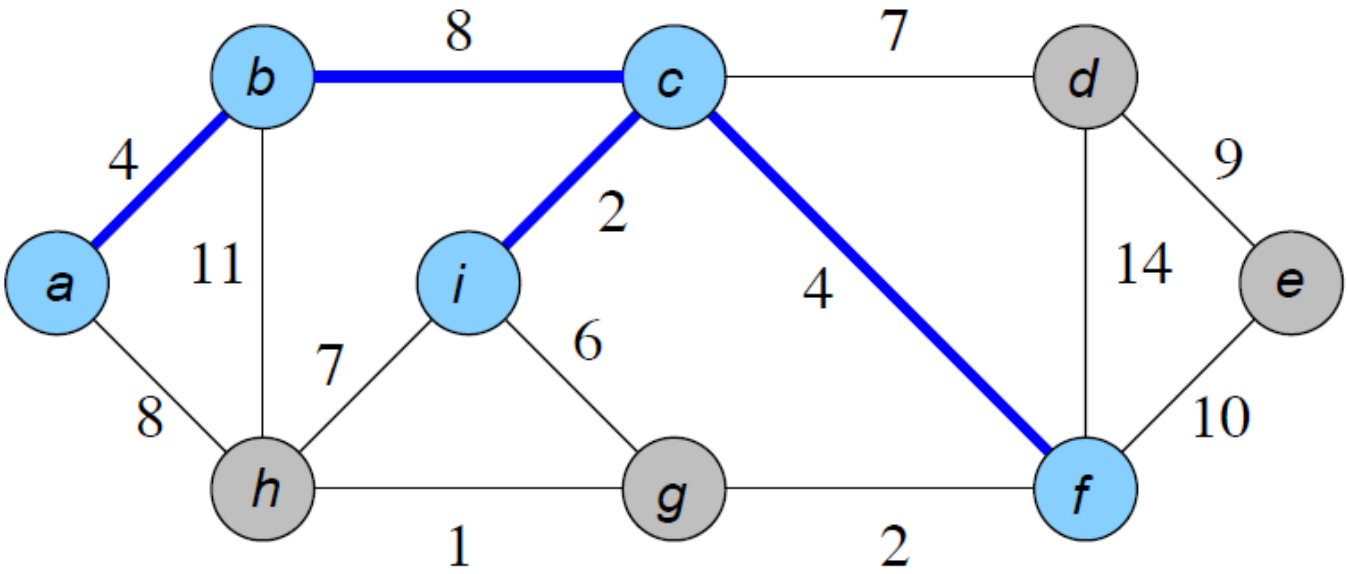
O algoritmo de Prim



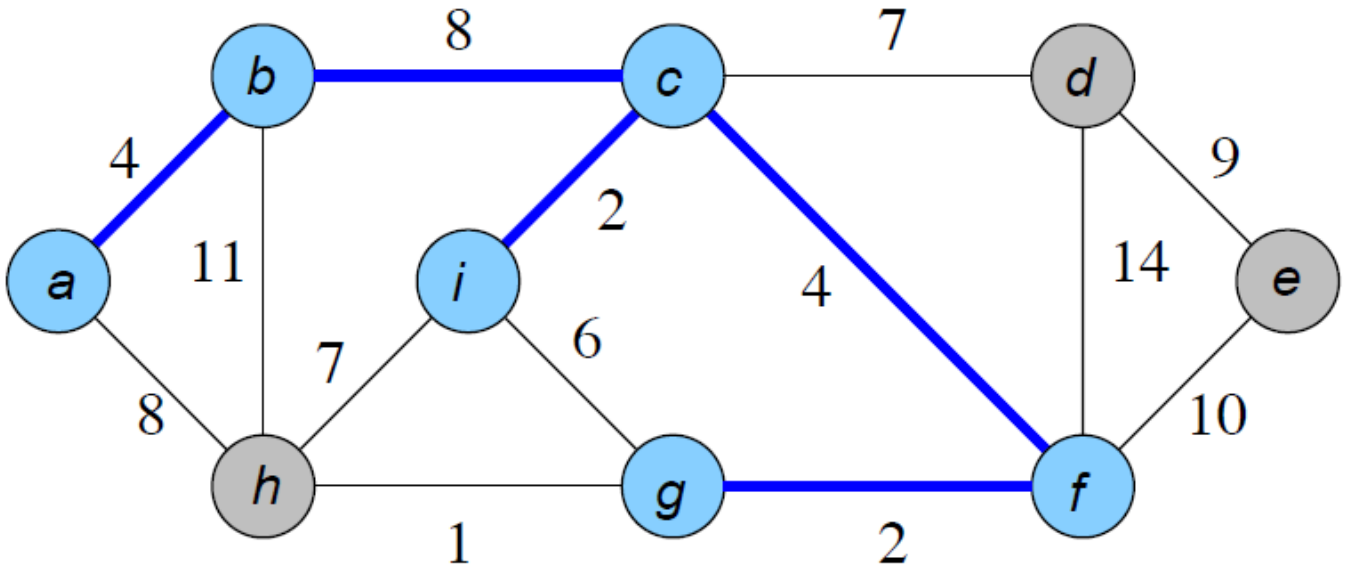
O algoritmo de Prim



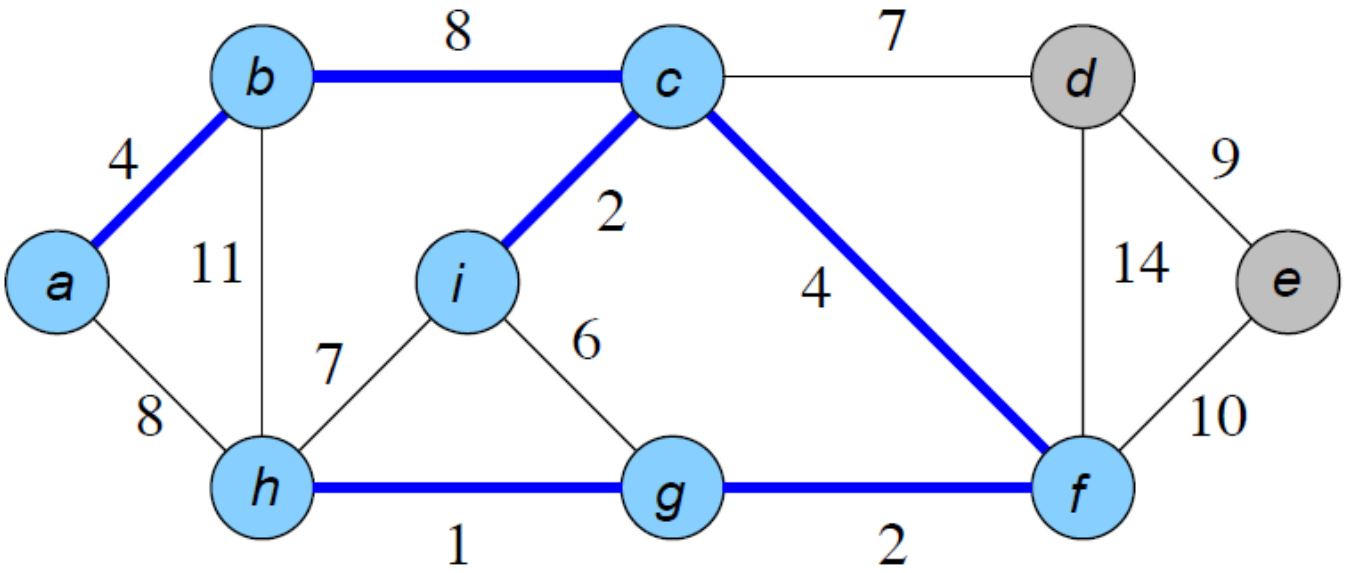
O algoritmo de Prim



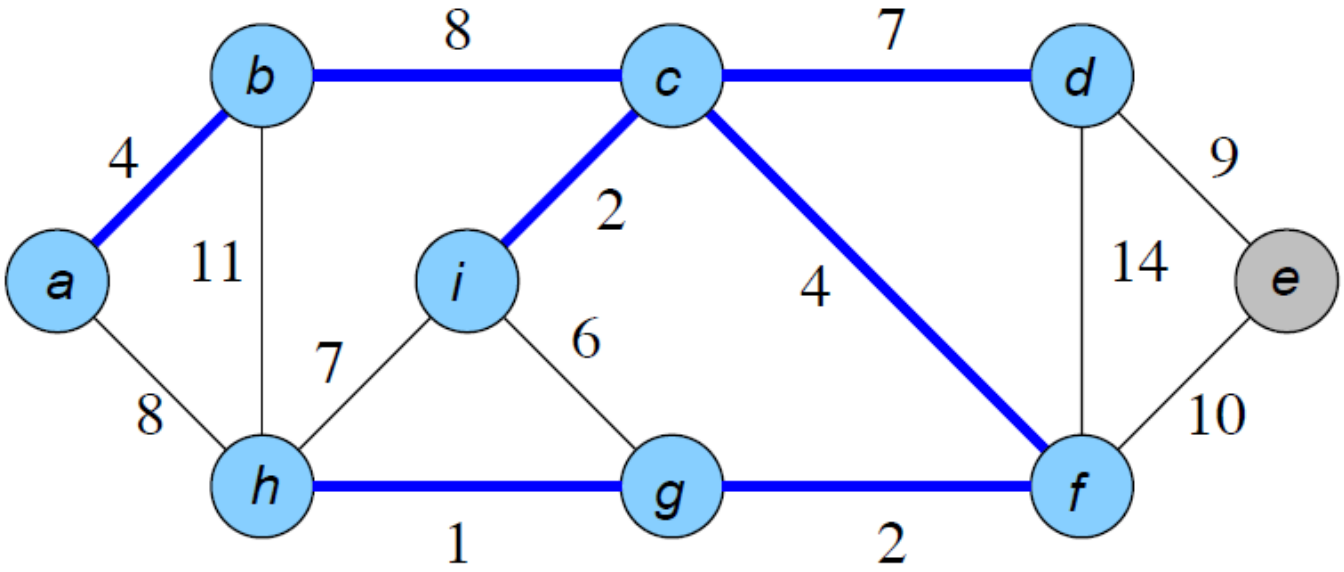
O algoritmo de Prim



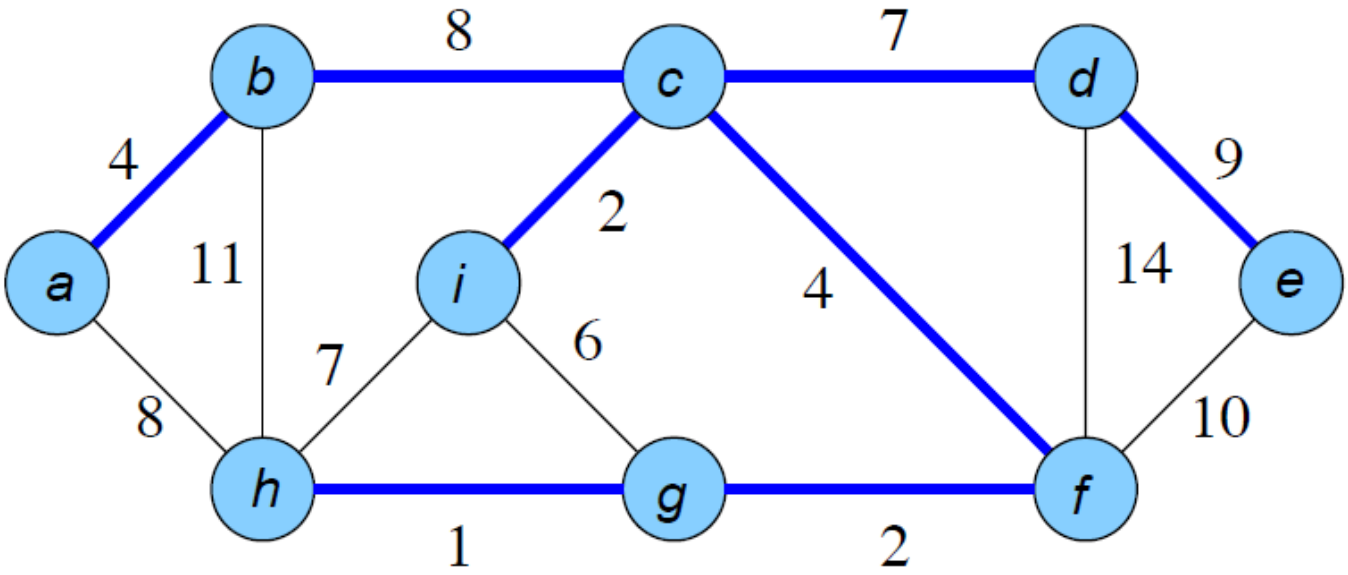
O algoritmo de Prim



O algoritmo de Prim



O algoritmo de Prim



O algoritmo de Prim

O algoritmo mantém durante sua execução as seguintes informações:

- Todos os vértices que **não** estão na árvore estão em uma fila de prioridade (de mínimo) Q.
- Cada vértice v em Q tem uma chave $key[v]$ que indica o menor peso de qualquer aresta ligando v a algum vértice da árvore. Se não existir nenhuma aresta, então $key[v] = \infty$.
- A variável $\pi[u]$ indica o pai de u na árvore. Então

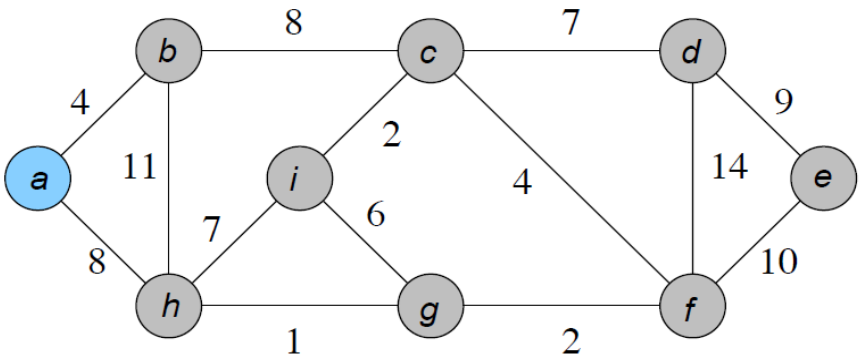
$$A = \{(u, \pi[u]) : u \in V - \{r\} - Q\}.$$

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1   para cada u ∈ V[G]
 2       faça key[u] ← ∞
 3            π[u] ← NIL
 4   key[r] ← 0
 5   Q ← V[G]
 6   enquanto Q ≠ ∅ faça
 7       u ← EXTRACT-MIN(Q) //o de menor key
 8       para cada v ∈ Adj[u]
 9           se v ∈ Q e w(u, v) < key[v]
10               então π[v] ← u
11                    key[v] ← w(u, v)
```

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2       faça key[u] ← ∞
 3            π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7       u ← EXTRACT-MIN(Q)
 8       para cada v ∈ Adj[u]
 9            se v ∈ Q e w(u, v) < key[v]
10                 então π[v] ← u
11                      key[v] ← w(u, v)
```



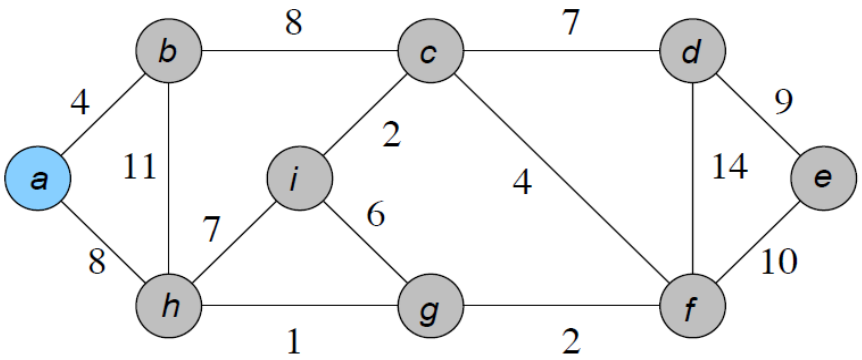
	a	b	c	d	e	f	g	h	i
key	0	∞	∞	∞	∞	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL

r = a

Q: a b c d e f g h i

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```



	a	b	c	d	e	f	g	h	i
key	0	∞	∞	∞	∞	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL

Q: ~~a~~ b c d e f g h i

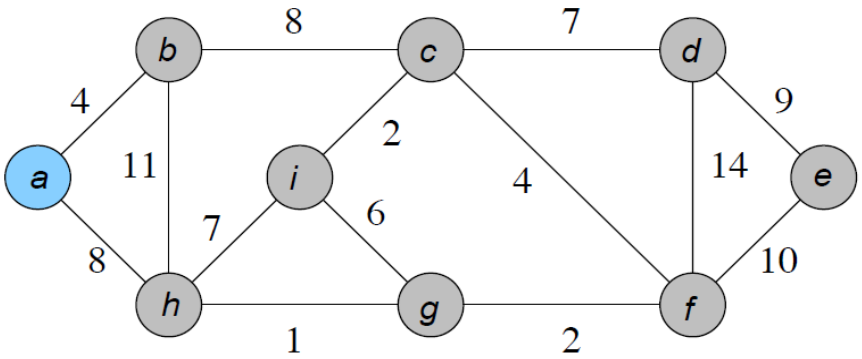
u = a

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	∞	∞	∞	∞	∞	∞	∞	∞
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL

Q: b c d e f g h i

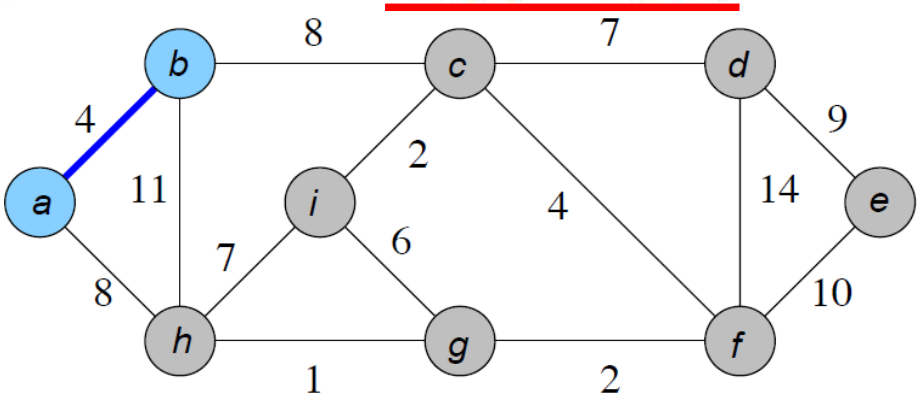


u = a

v = b

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                    key[v] ← w(u, v)
```



	a	b	c	d	e	f	g	h	i
key	0	4	∞	∞	∞	∞	∞	∞	∞
π	NIL	a	NIL	NIL	NIL	NIL	NIL	NIL	NIL

Q: b c d e f g h i

u = a

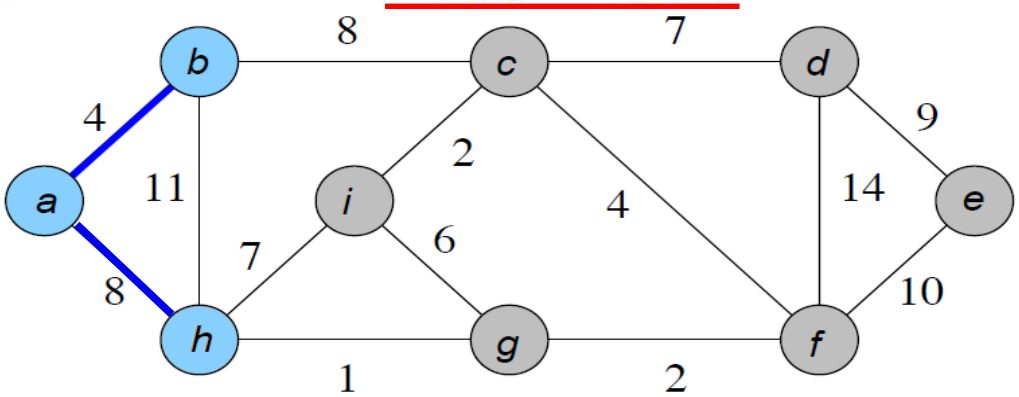
v = b

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                    key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	∞	∞	∞	∞	∞	8	∞
π	NIL	a	NIL	NIL	NIL	NIL	NIL	a	NIL

Q: b c d e f g h i



u = a

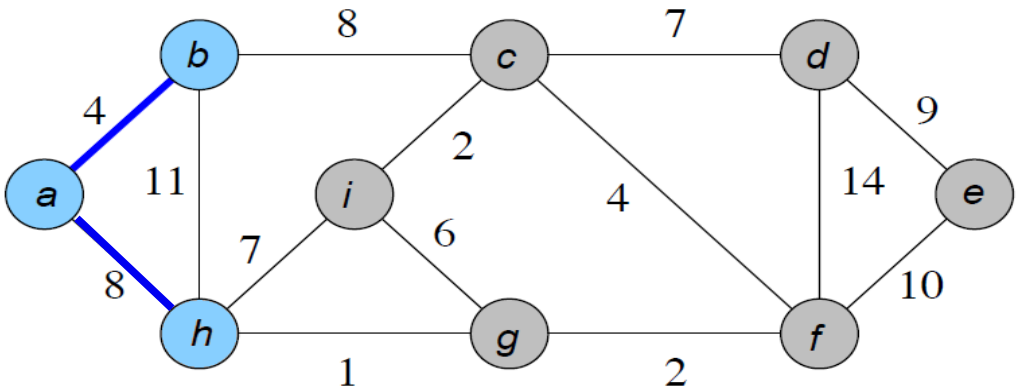
v = h

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	∞	∞	∞	∞	∞	8	∞
π	NIL	a	NIL	NIL	NIL	NIL	NIL	a	NIL

Q: b c d e f g h i



u = a

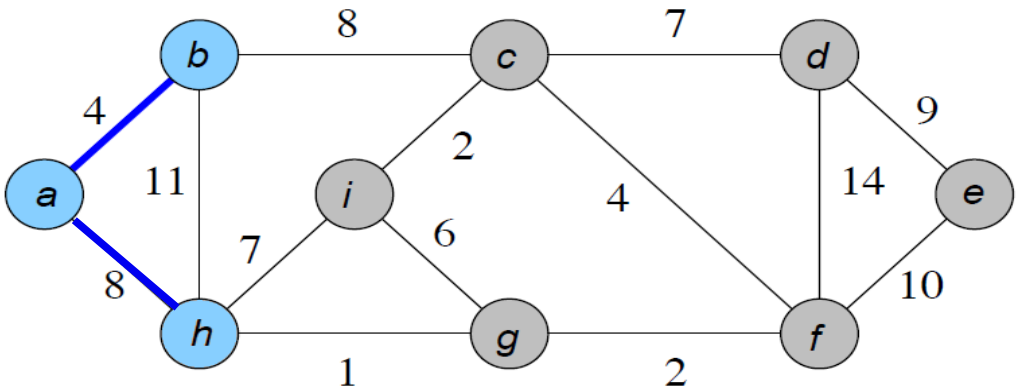
v = b

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	∞	∞	∞	∞	∞	∞	∞
π	NIL	a	NIL	NIL	NIL	NIL	NIL	NIL	NIL

Q: ~~b~~ c d e f g h i



u = b

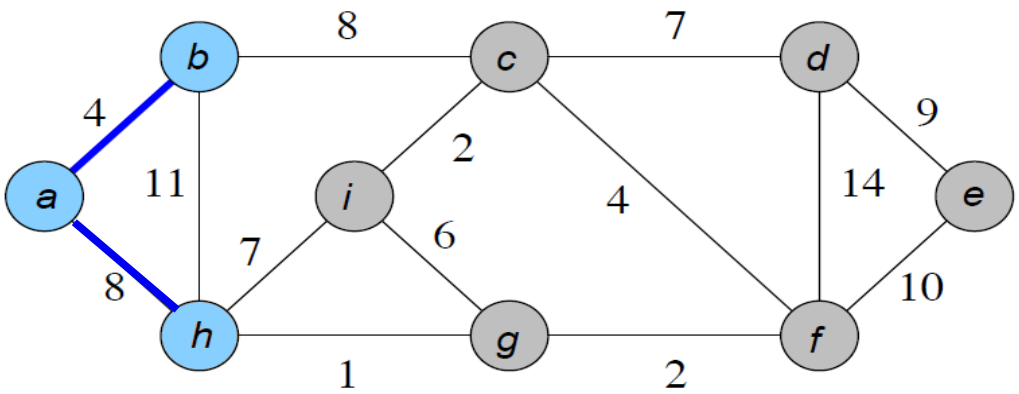
v =

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	∞	∞	∞	∞	∞	8	∞
π	NIL	a	NIL	NIL	NIL	NIL	NIL	a	NIL

Q: c d e f g h i



u = b

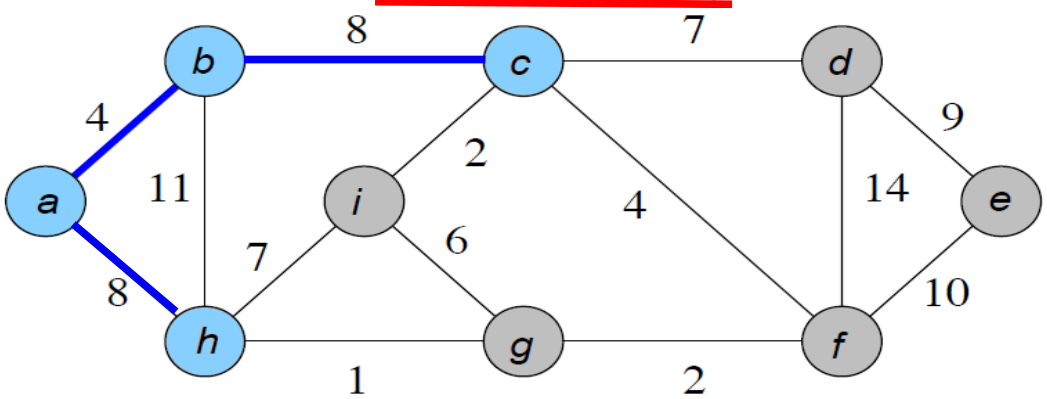
v = c

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	∞	∞	∞	∞	8	∞
π	NIL	a	b	NIL	NIL	NIL	NIL	a	NIL

Q: c d e f g h i



u = b

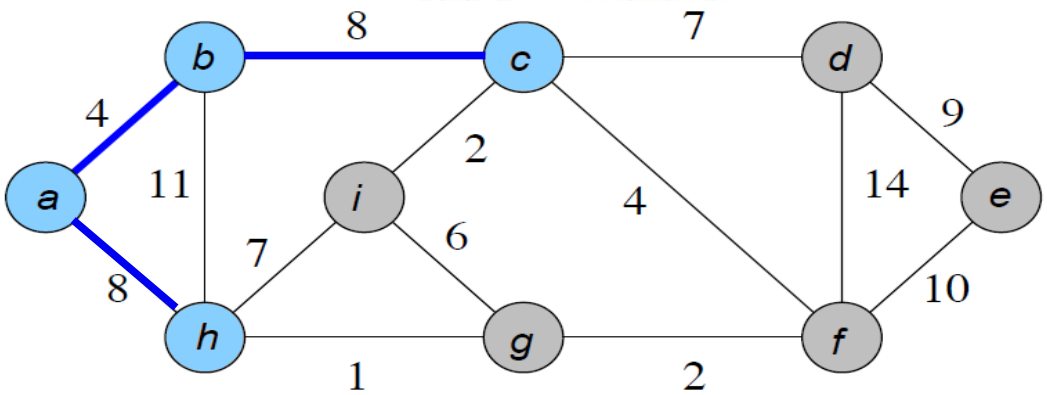
v = c

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	∞	∞	∞	∞	8	∞
π	NIL	a	b	NIL	NIL	NIL	NIL	a	NIL

Q: c d e f g h i



u = b

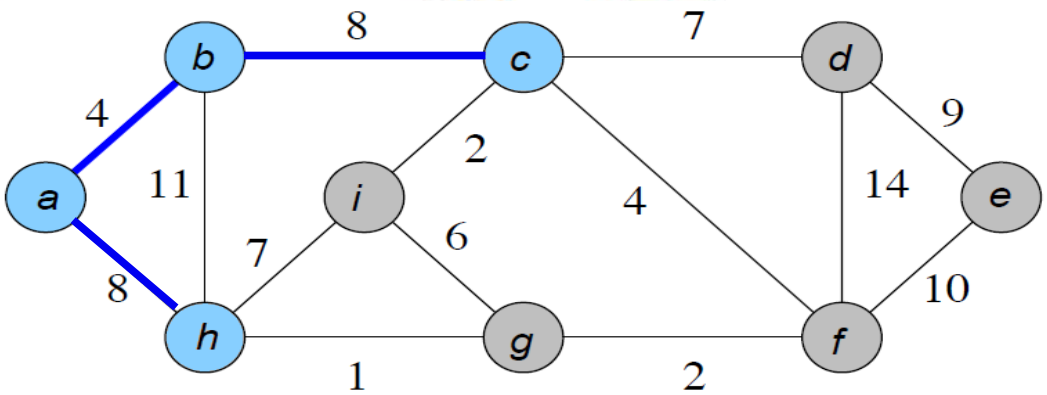
v = h

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2       faça key[u] ← ∞
 3            π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7       u ← EXTRACT-MIN(Q)
 8       para cada v ∈ Adj[u]
 9            se v ∈ Q e w(u, v) < key[v]
10                 então π[v] ← u
11                      key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	∞	∞	∞	∞	8	∞
π	NIL	a	b	NIL	NIL	NIL	NIL	a	NIL

Q: ~~c~~ d e f g h i



u = c

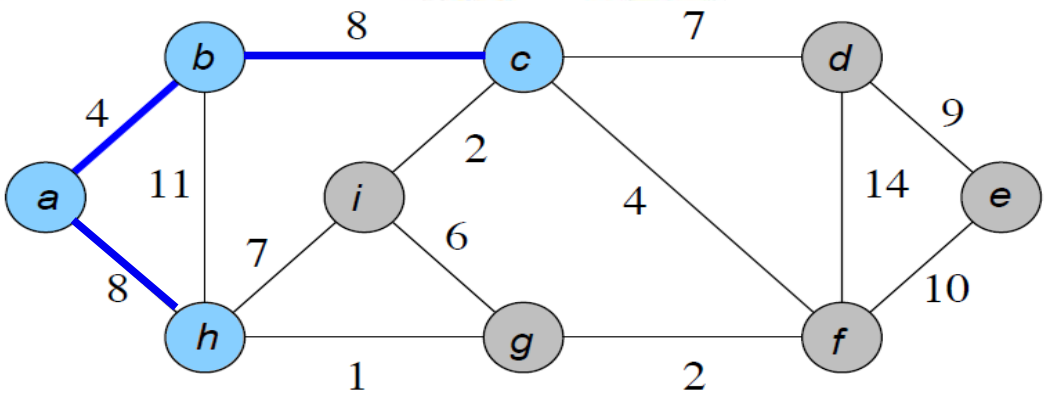
v =

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	∞	∞	∞	∞	8	∞
π	NIL	a	b	NIL	NIL	NIL	NIL	a	NIL

Q: d e f g h i



u = c

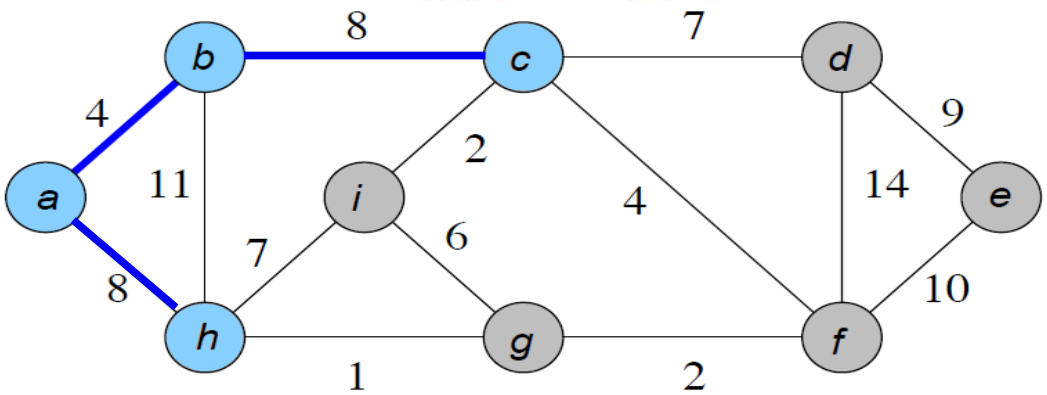
v = i

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	∞	∞	∞	∞	8	∞
π	NIL	a	b	NIL	NIL	NIL	NIL	a	NIL

Q: d e f g h i



u = c

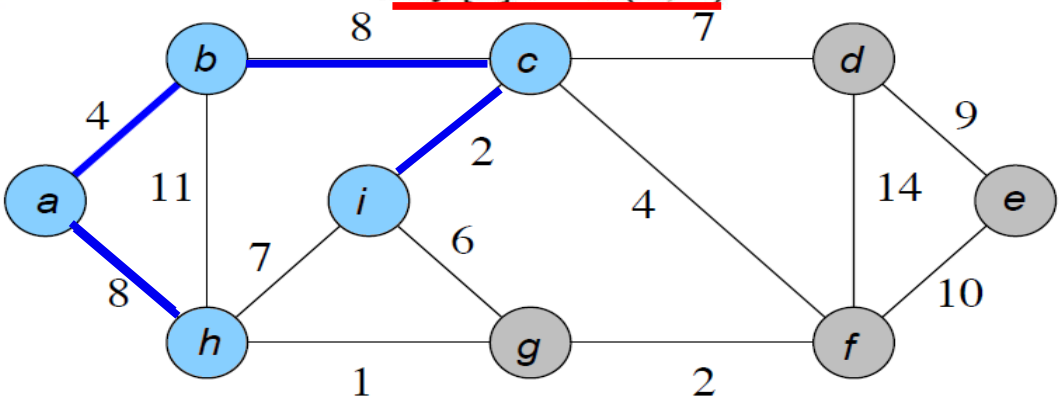
v = i

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	∞	∞	∞	∞	8	2
π	NIL	a	b	NIL	NIL	NIL	NIL	a	c

Q: d e f g h i



u = c

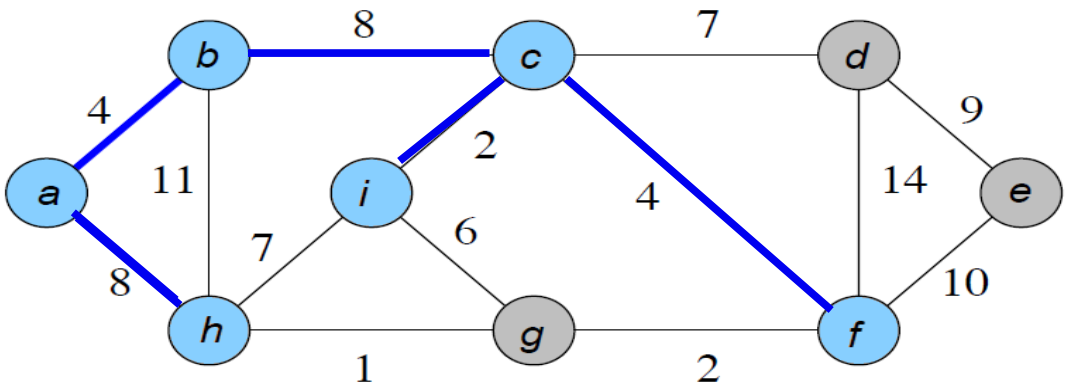
v = i

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                    key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	∞	∞	4	∞	8	2
π	NIL	a	b	NIL	NIL	c	NIL	a	c

Q: d e f g h i



u = c

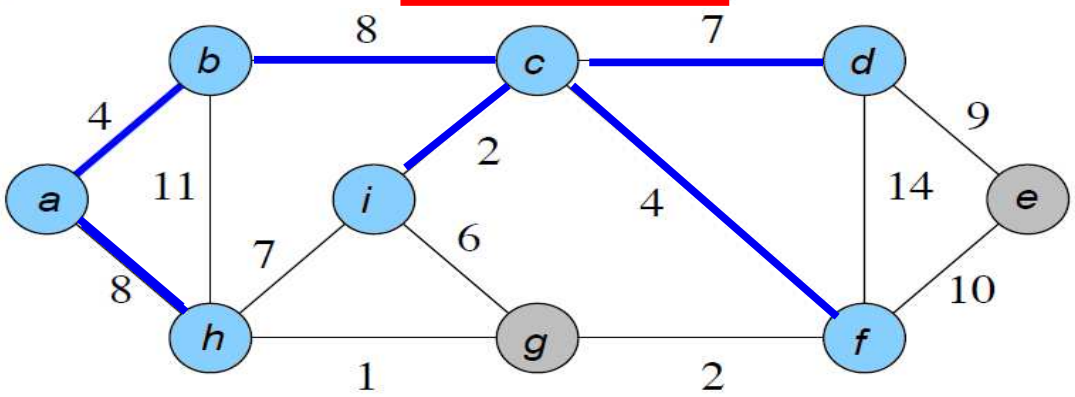
v = f

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	∞	8	2
π	NIL	a	b	c	NIL	c	NIL	a	c

Q: d e f g h i



u = c

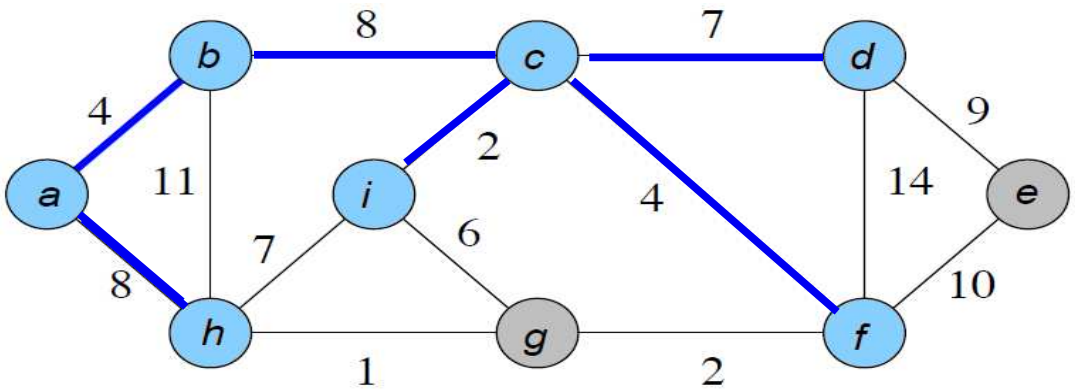
v = d

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	∞	8	2
π	NIL	a	b	c	NIL	c	NIL	a	c

Q: d e f g h ~~i~~



u = i

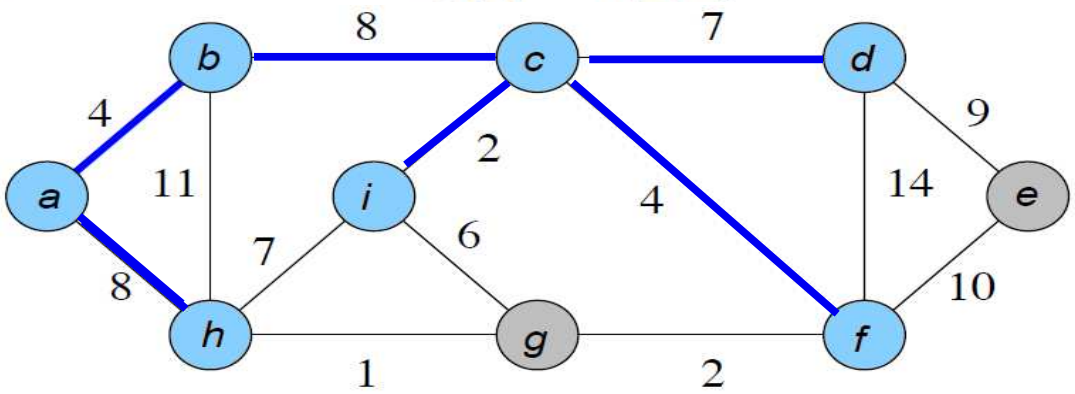
v =

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	∞	8	2
π	NIL	a	b	c	NIL	c	NIL	a	c

Q: d e f g h



u = i

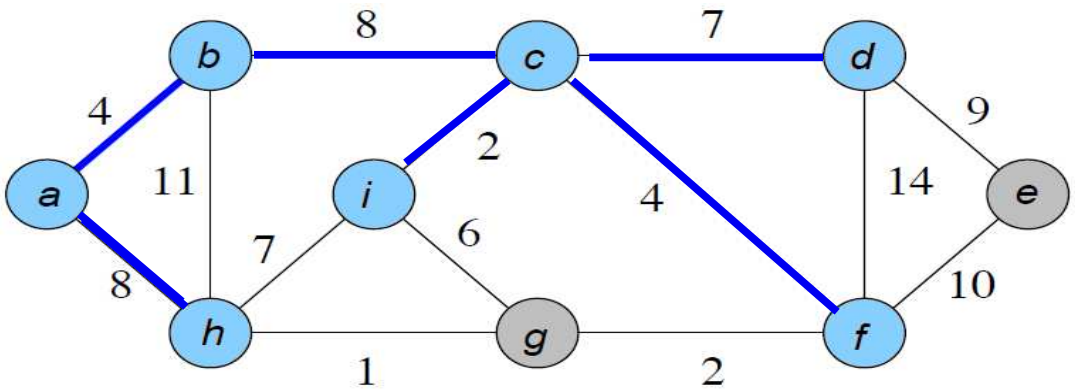
v = h

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	∞	8	2
π	NIL	a	b	c	NIL	c	NIL	a	c

Q: d e f g h



u = i

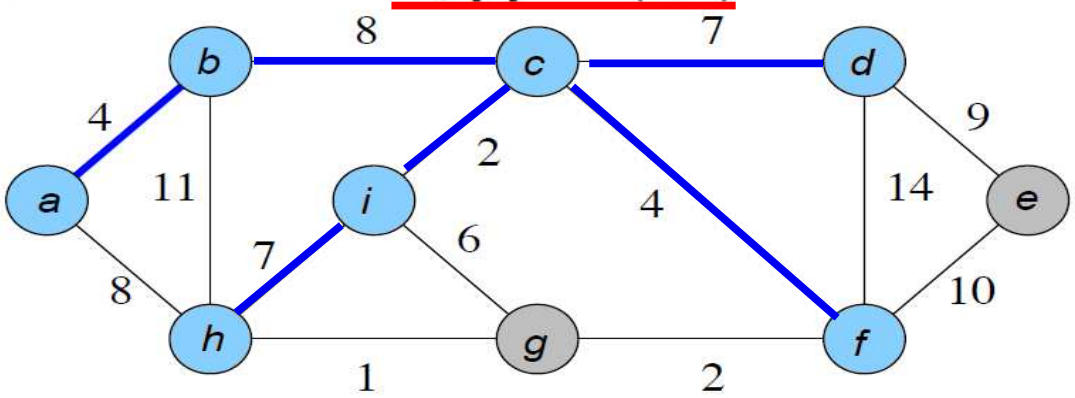
v = h

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	∞	7	2
π	NIL	a	b	c	NIL	c	NIL	i	c

Q: d e f g h



u = i

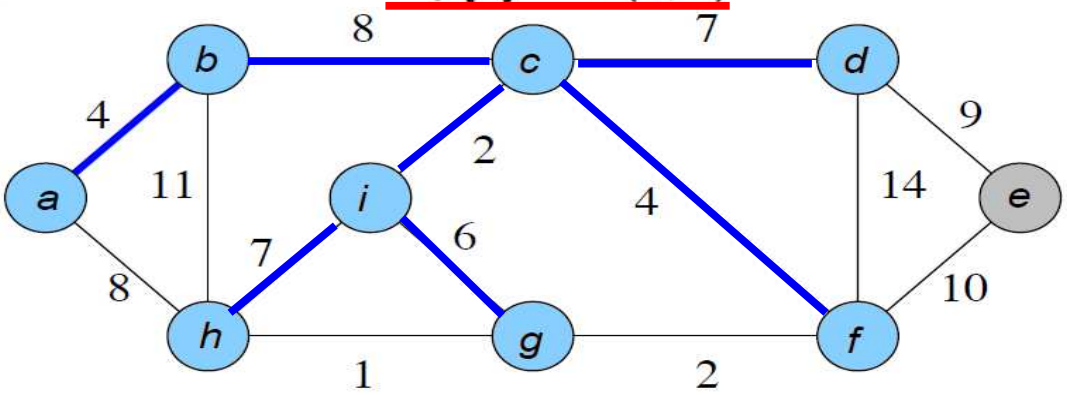
v = h

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	6	7	2
π	NIL	a	b	c	NIL	c	i	i	c

Q: d e f g h



u = i

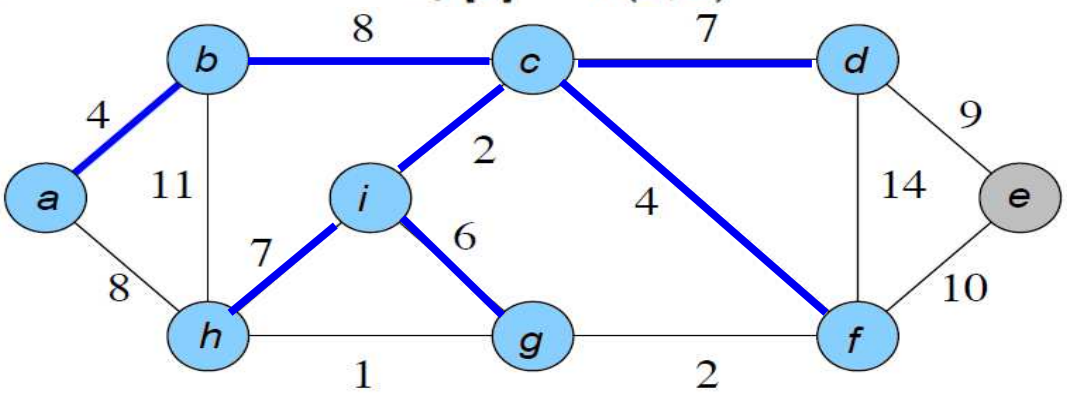
v = g

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	6	7	2
π	NIL	a	b	c	NIL	c	i	i	c

Q: d e f g h



u = i

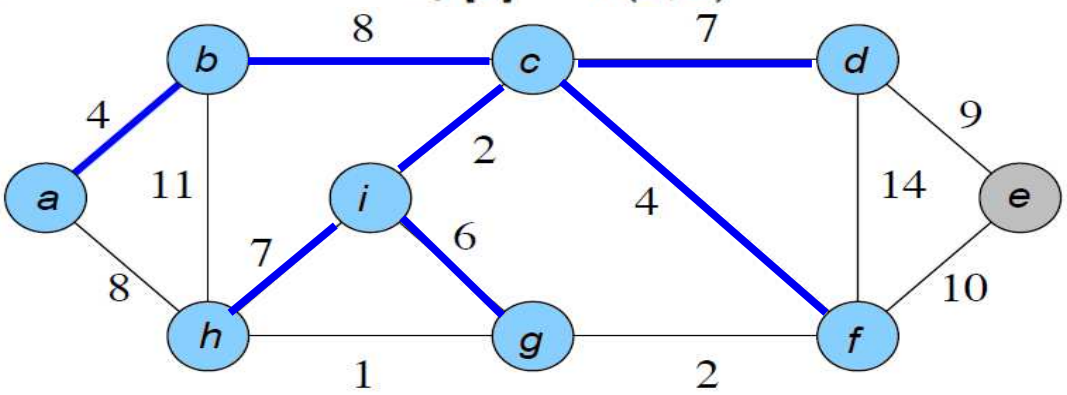
v = c

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	6	7	2
π	NIL	a	b	c	NIL	c	i	i	c

Q: d e ~~f~~ g h



u = f

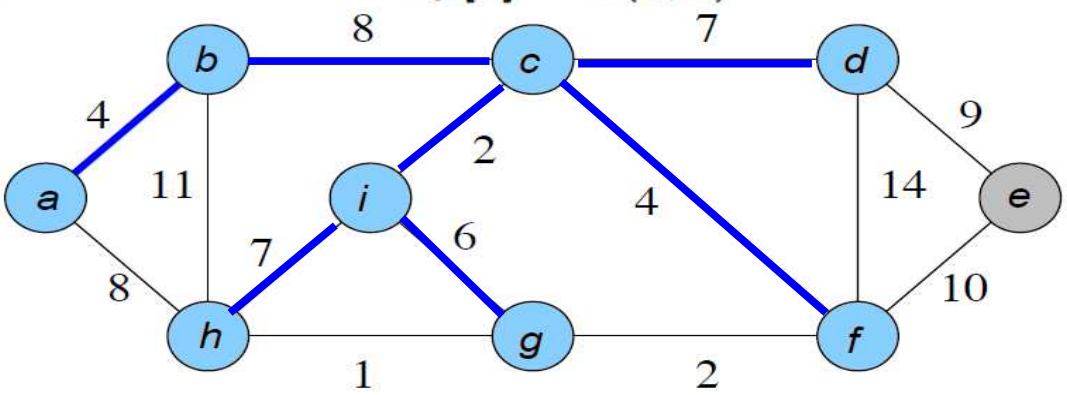
v =

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	6	7	2
π	NIL	a	b	c	NIL	c	i	i	c

Q: d e g h



u = f

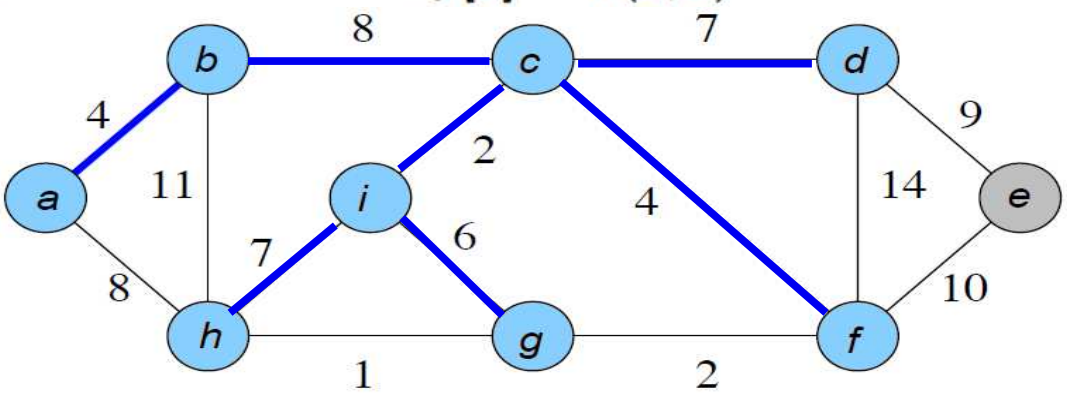
v = g

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	6	7	2
π	NIL	a	b	c	NIL	c	i	i	c

Q: d e g h



u = f

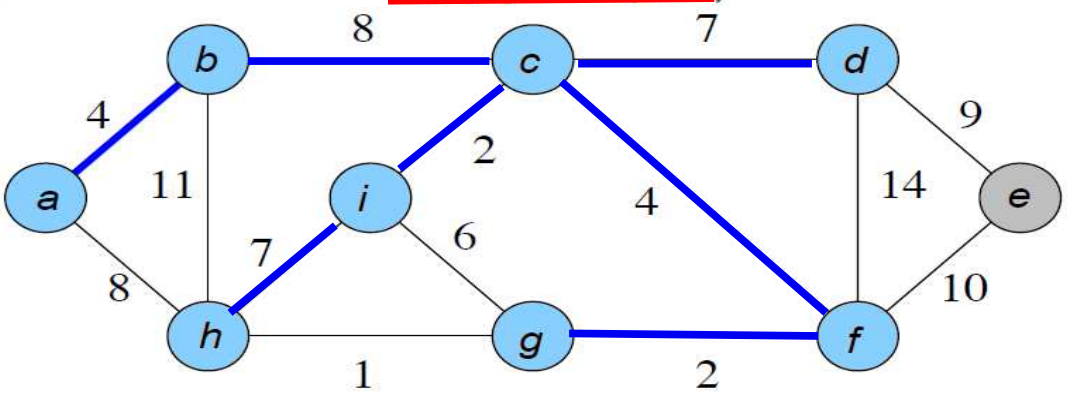
v = g

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	2	7	2
π	NIL	a	b	c	NIL	c	f	i	c

Q: d e g h



u = f

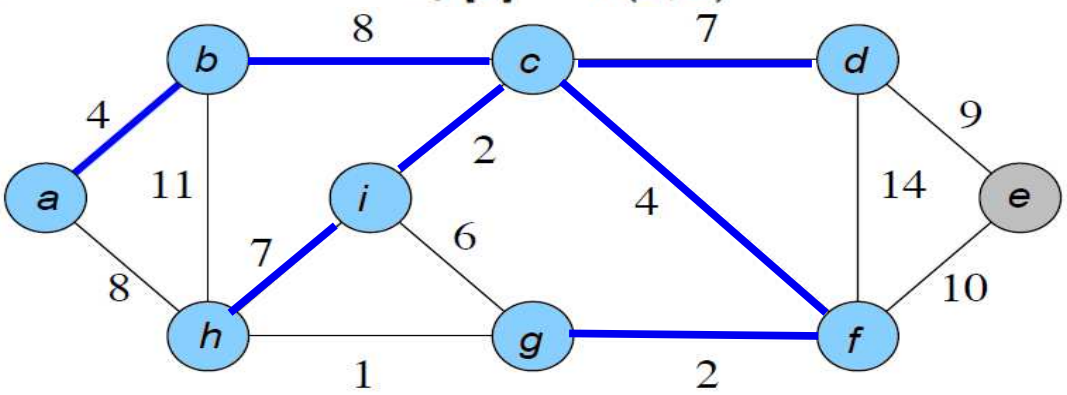
v = g

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	2	7	2
π	NIL	a	b	c	NIL	c	f	i	c

Q: d e g h



u = f

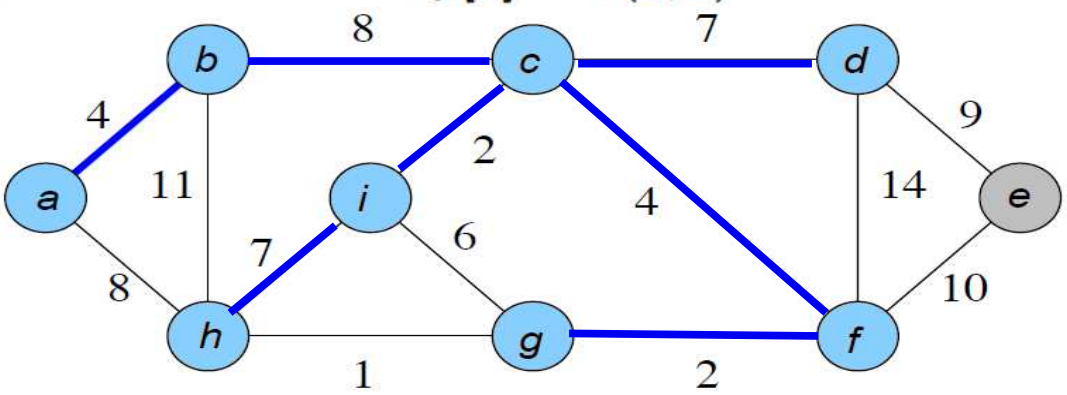
v = c

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	2	7	2
π	NIL	a	b	c	NIL	c	f	i	c

Q: d e g h



u = f

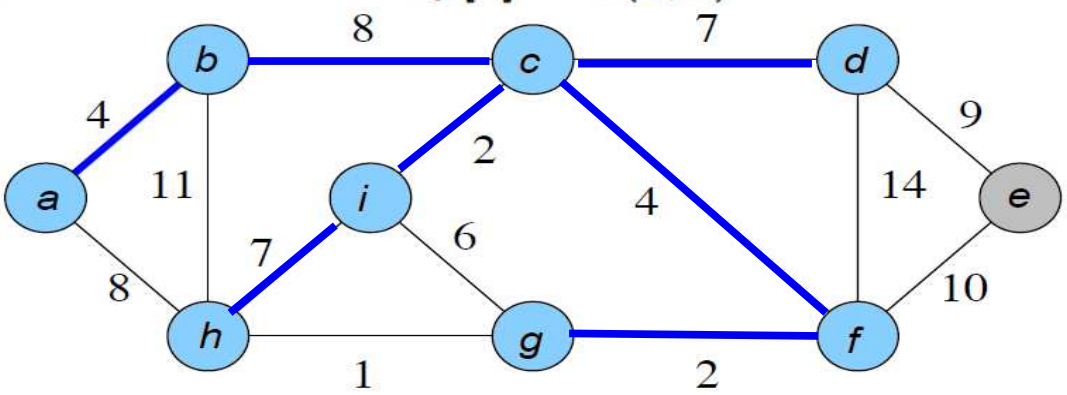
v = d

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	∞	4	2	7	2
π	NIL	a	b	c	NIL	c	f	i	c

Q: d e g h



u = f

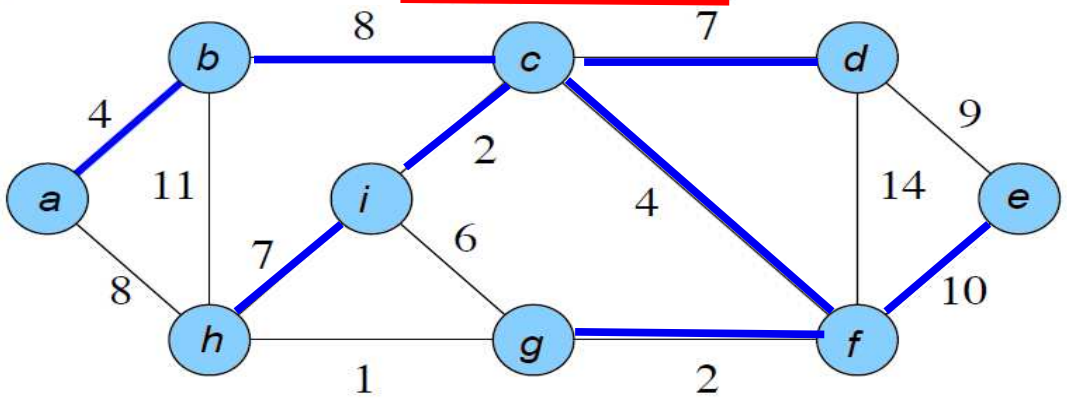
v = e

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	10	4	2	7	2
π	NIL	a	b	c	f	c	f	i	c

Q: d e g h



u = f

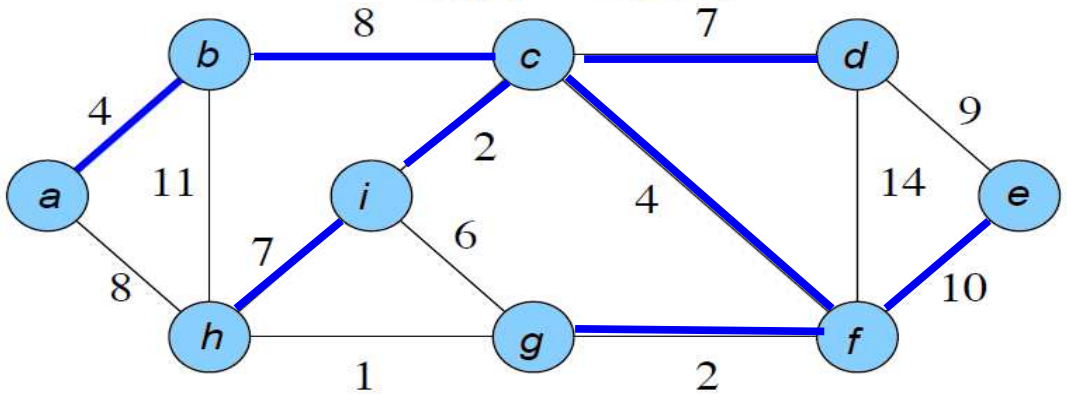
v = e

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	10	4	2	7	2
π	NIL	a	b	c	f	c	f	i	c

Q: d e ~~g~~ h



u = g

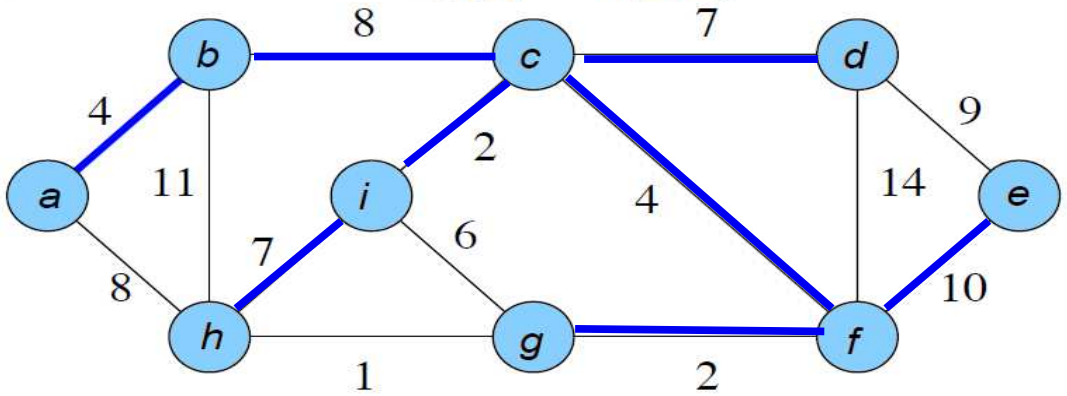
v =

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	10	4	2	7	2
π	NIL	a	b	c	f	c	f	i	c

Q: d e h



u = g

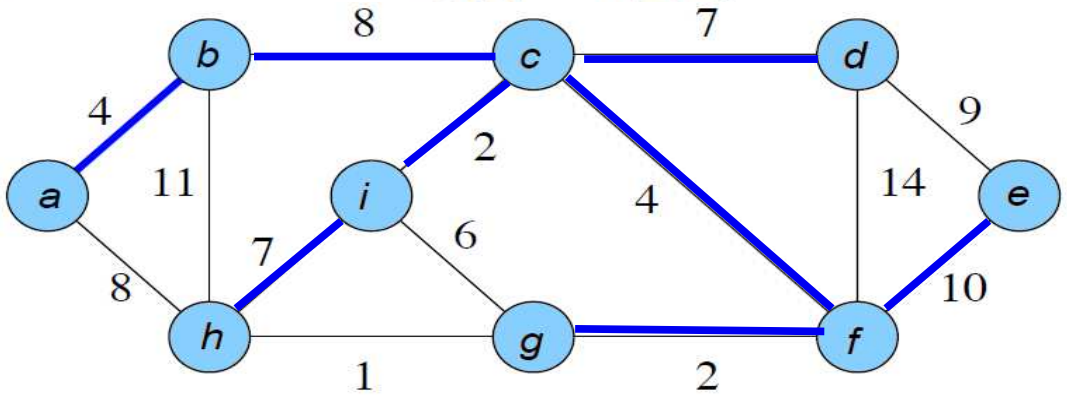
v = h

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	10	4	2	7	2
π	NIL	a	b	c	f	c	f	i	c

Q: d e h



u = g

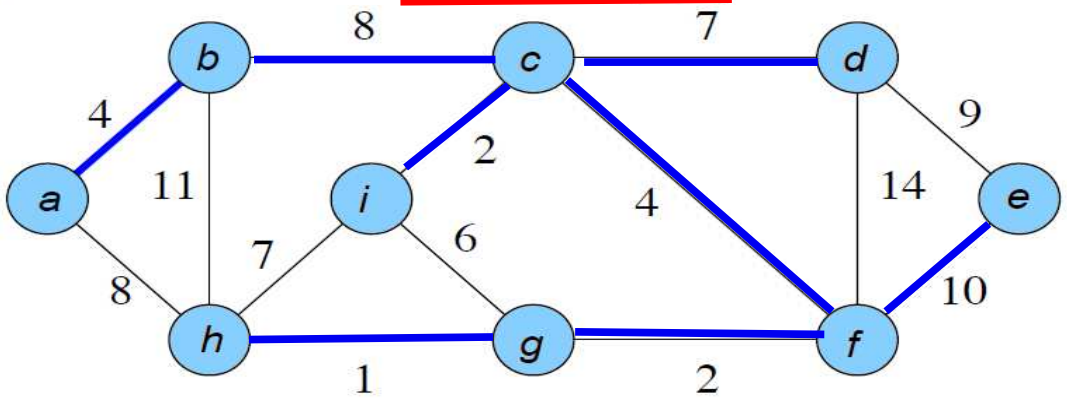
v = h

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	10	4	2	1	2
π	NIL	a	b	c	f	c	f	g	c

Q: d e h



u = g

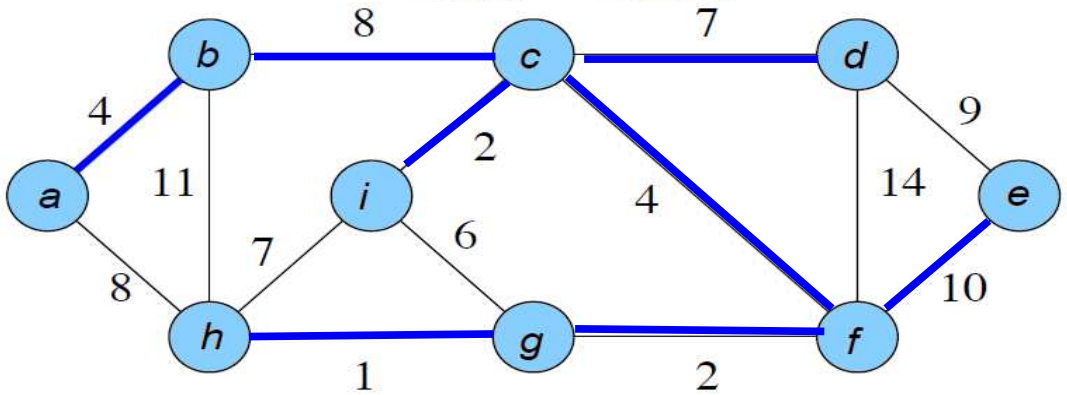
v = h

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2       faça key[u] ← ∞
 3            π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7       u ← EXTRACT-MIN(Q)
 8       para cada v ∈ Adj[u]
 9            se v ∈ Q e w(u, v) < key[v]
10                 então π[v] ← u
11                      key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	10	4	2	1	2
π	NIL	a	b	c	f	c	f	g	c

Q: d e ~~h~~



u = h

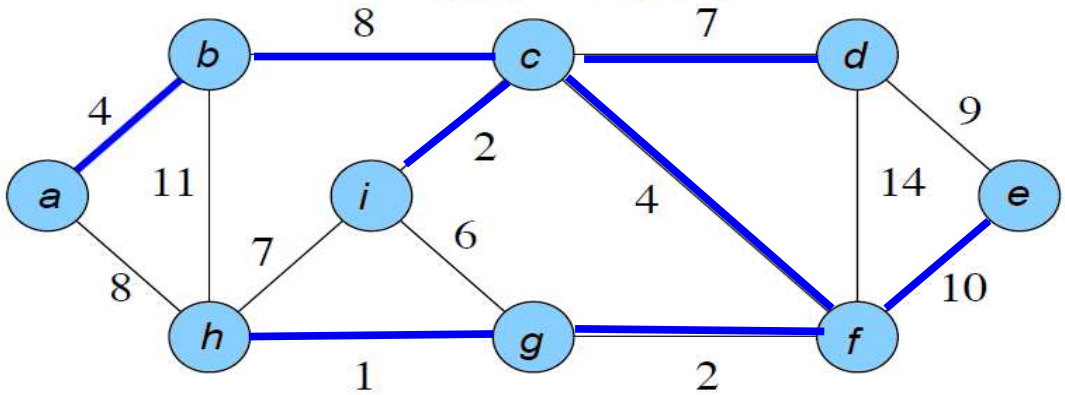
v =

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	10	4	2	1	2
π	NIL	a	b	c	f	c	f	g	c

Q: d e



u = h

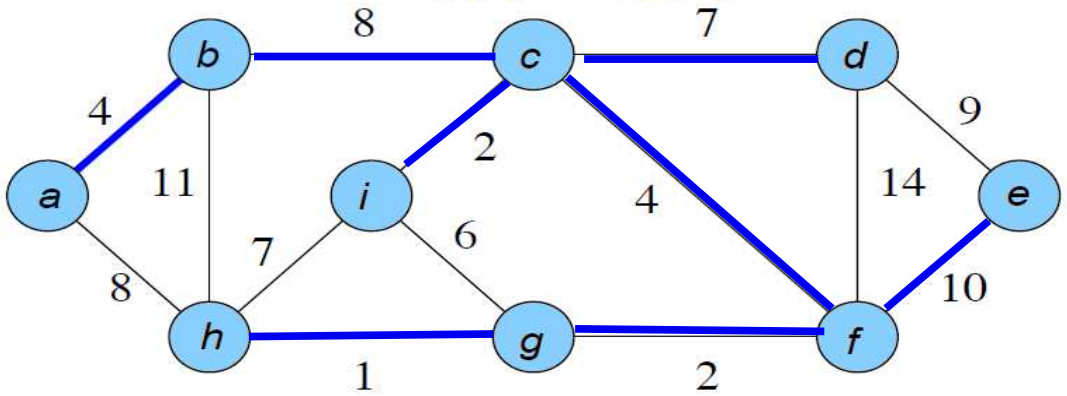
v = a / b / i / g

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	10	4	2	1	2
π	NIL	a	b	c	f	c	f	g	c

Q: ~~d~~ e



u = d

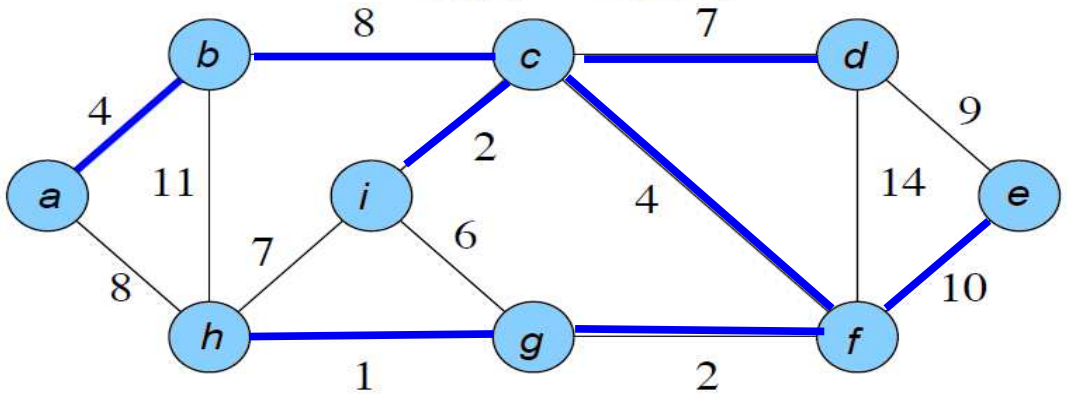
v =

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2       faça key[u] ← ∞
 3            π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7       u ← EXTRACT-MIN(Q)
 8       para cada v ∈ Adj[u]
 9            se v ∈ Q e w(u, v) < key[v]
10                 então π[v] ← u
11                      key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	10	4	2	1	2
π	NIL	a	b	c	f	c	f	g	c

Q: e



u = d

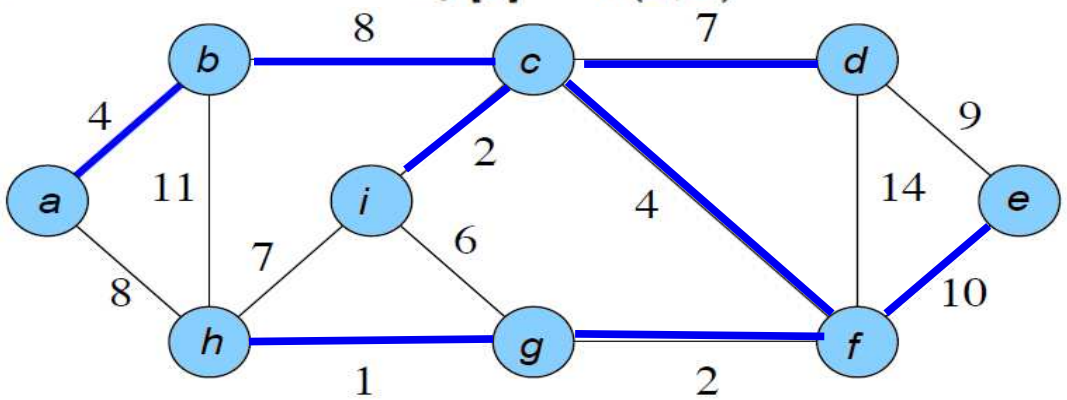
v = e

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

	a	b	c	d	e	f	g	h	i
key	0	4	8	7	10	4	2	1	2
π	NIL	a	b	c	f	c	f	g	c

Q: e



u = d

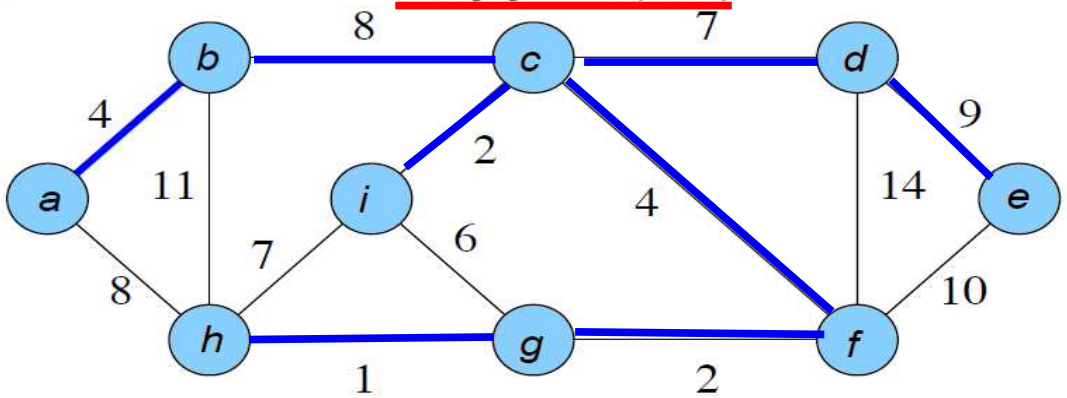
v = e

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

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

Q: e



u = d

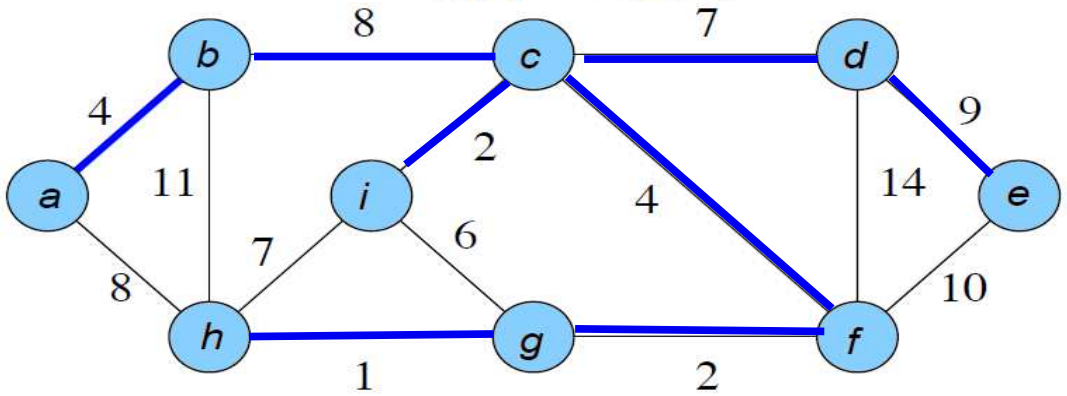
v = e

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2      faça key[u] ← ∞
 3           π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7      u ← EXTRACT-MIN(Q)
 8      para cada v ∈ Adj[u]
 9          se v ∈ Q e w(u, v) < key[v]
10              então π[v] ← u
11                   key[v] ← w(u, v)
```

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

Q: e



u = d

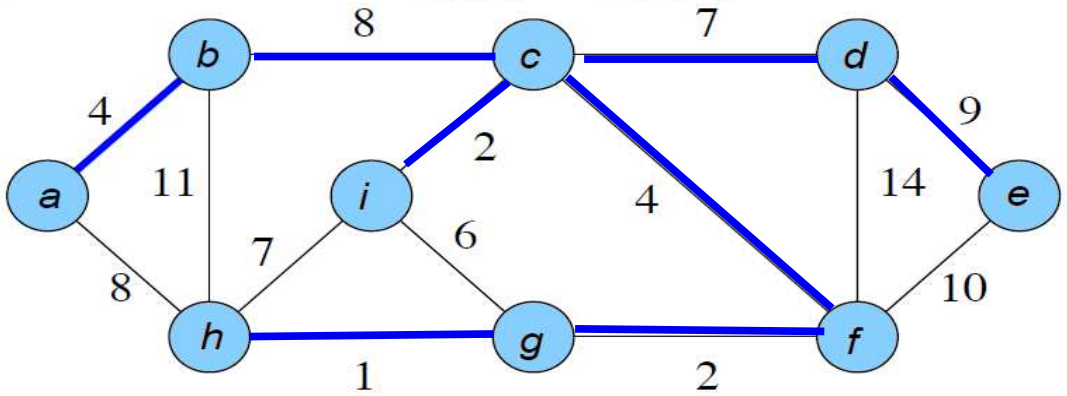
v = f

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2       faça key[u] ← ∞
 3            π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7       u ← EXTRACT-MIN(Q)
 8       para cada v ∈ Adj[u]
 9            se v ∈ Q e w(u, v) < key[v]
10                 então π[v] ← u
11                      key[v] ← w(u, v)
```

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

Q: ~~e~~



u = e

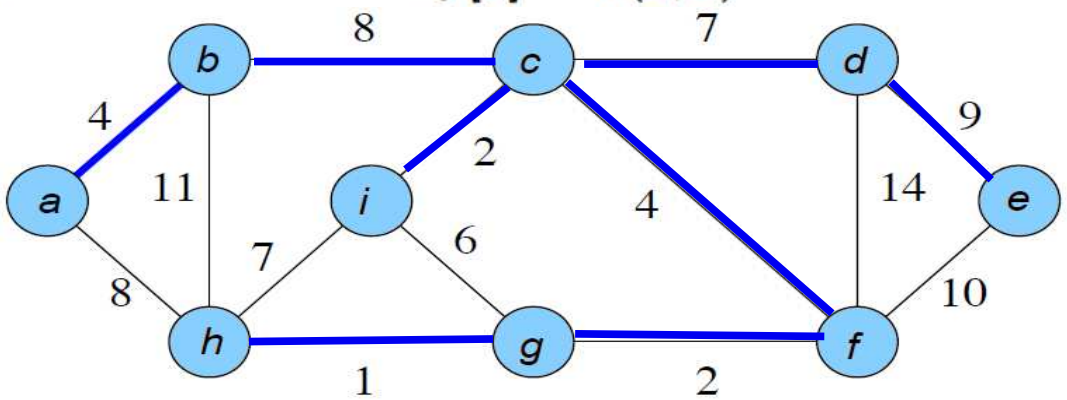
v =

O algoritmo de Prim

```
AGM-PRIM(G, w, r)
 1  para cada u ∈ V[G]
 2       faça key[u] ← ∞
 3            π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  enquanto Q ≠ ∅ faça
 7       u ← EXTRACT-MIN(Q)
 8       para cada v ∈ Adj[u]
 9            se v ∈ Q e w(u, v) < key[v]
10                 então π[v] ← u
11                      key[v] ← w(u, v)
```

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

Q: $\emptyset$



u = e

v = d / f

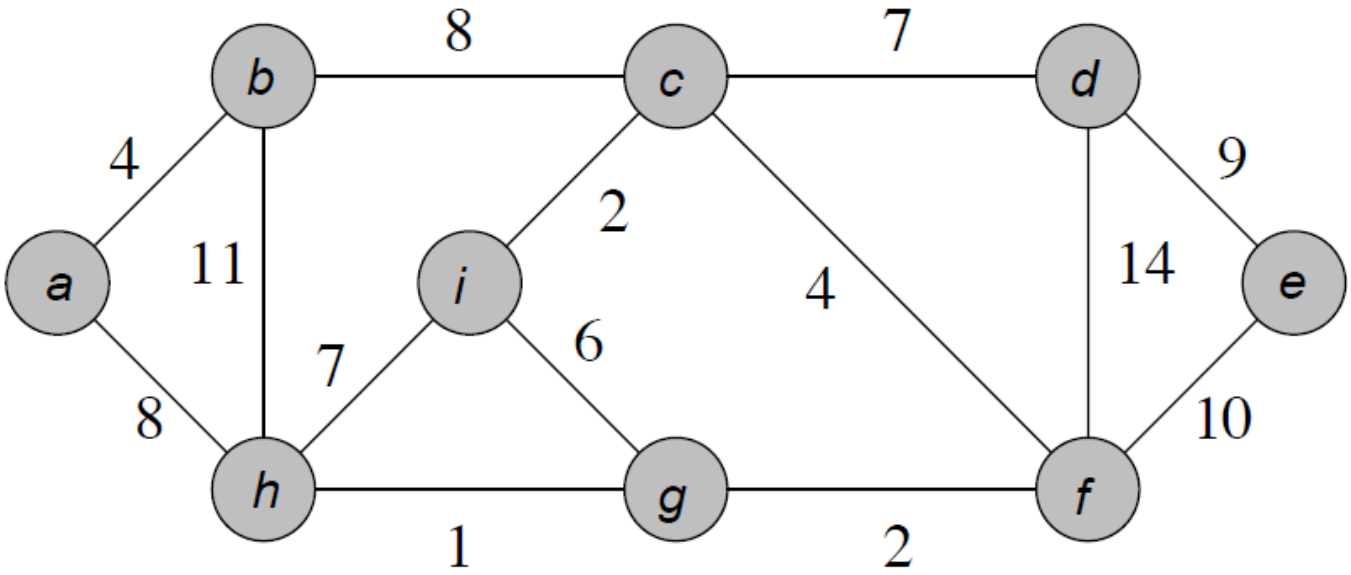
O algoritmo de Kruskal

O algoritmo de Kruskal

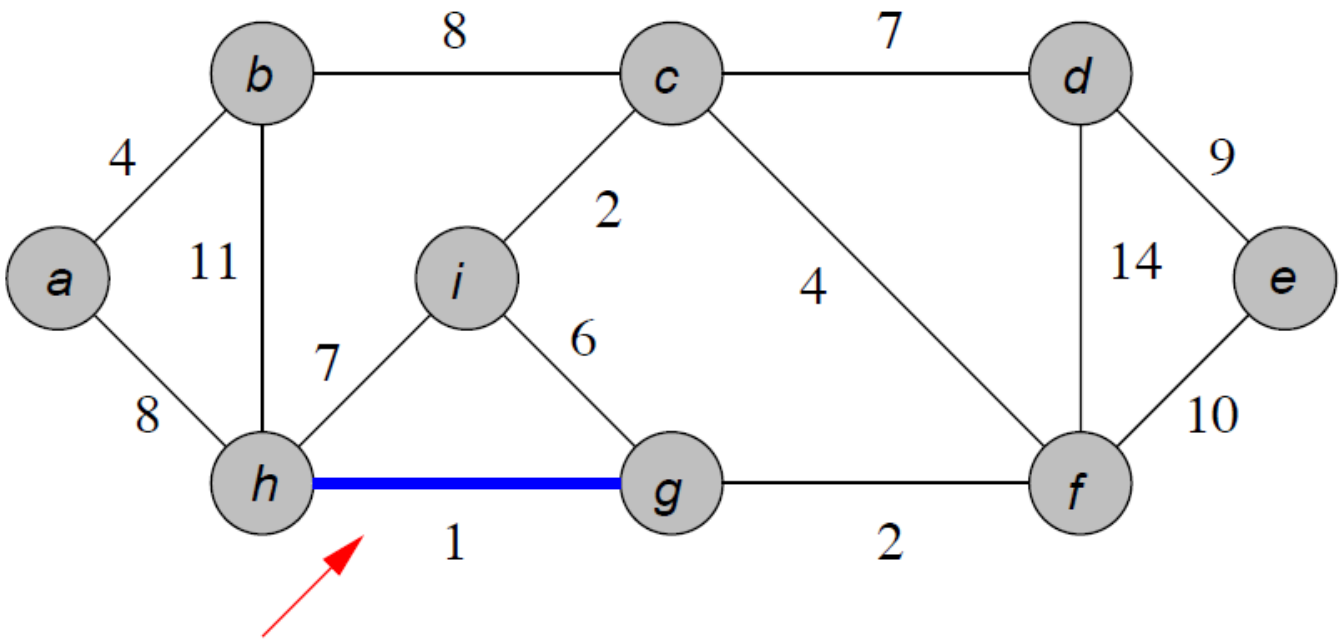
- No algoritmo de Kruskal o subgrafo $F = (V, A)$ é uma floresta. Inicialmente, A é vazio.
- Em cada iteração, o algoritmo escolhe uma aresta (u, v) de menor peso que liga vértices de componentes (árvores) distintos C e C' de $F = (V, A)$.
  Note que (u, v) é uma aresta leve do corte $\delta(C)$.
- Ele acrescenta (u, v) ao conjunto A e começa outra iteração até que A seja uma árvore geradora.

Um detalhe de implementação importante é como encontrar a aresta de menor peso ligando componentes distintos.

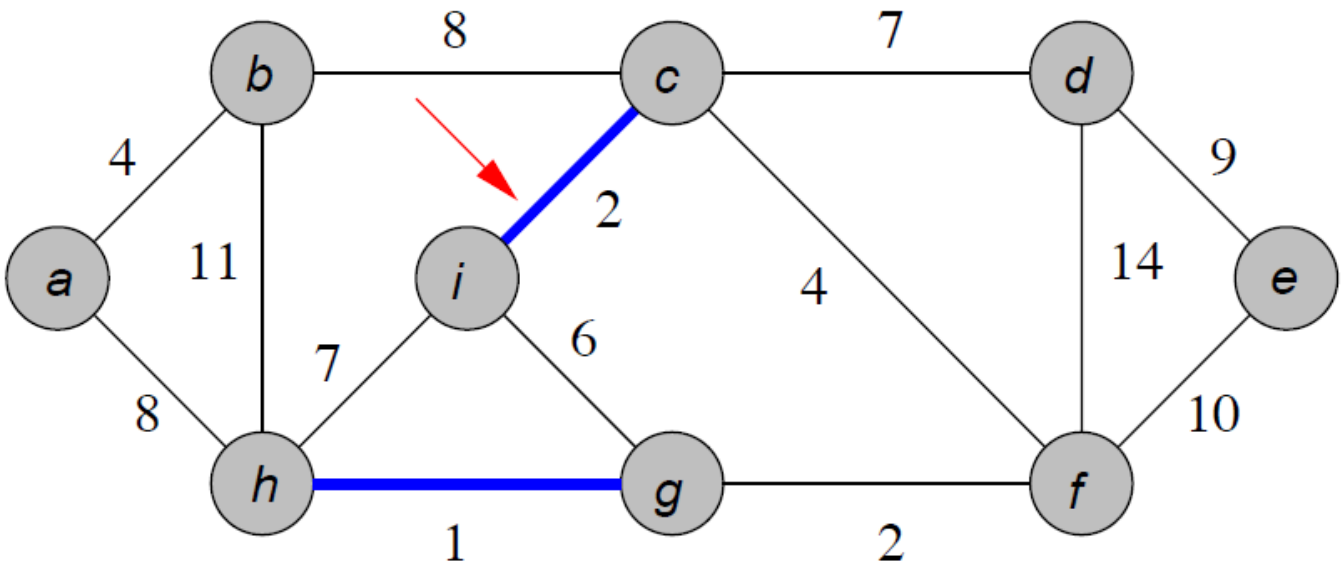
O algoritmo de Kruskal



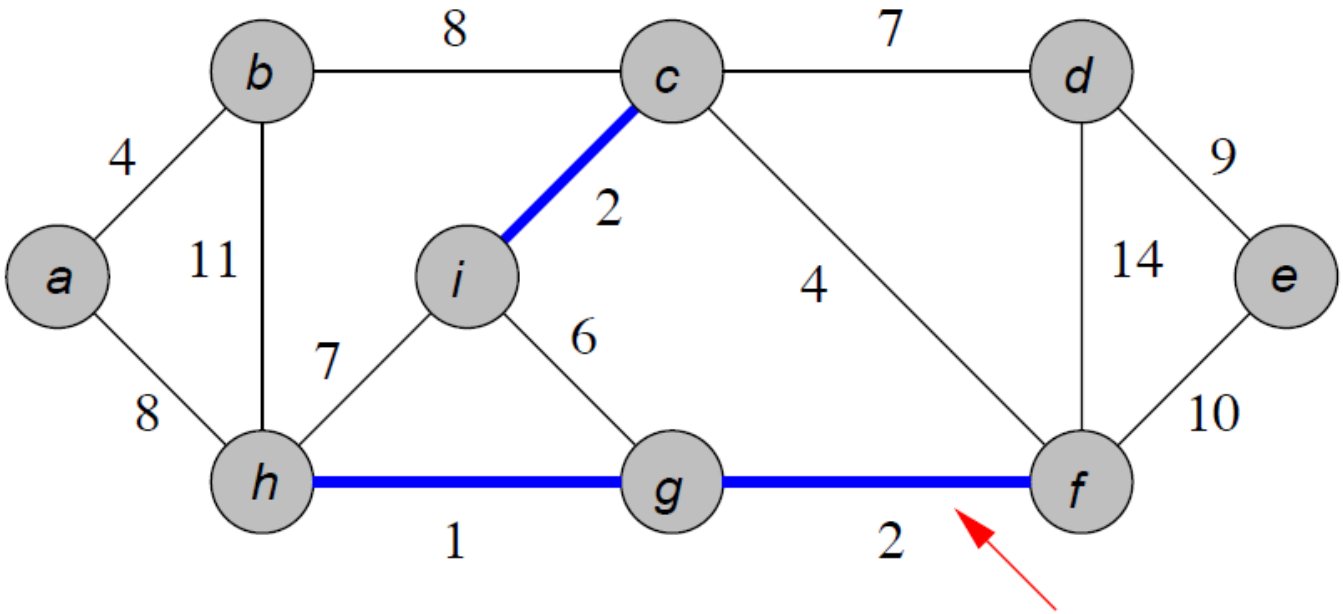
O algoritmo de Kruskal



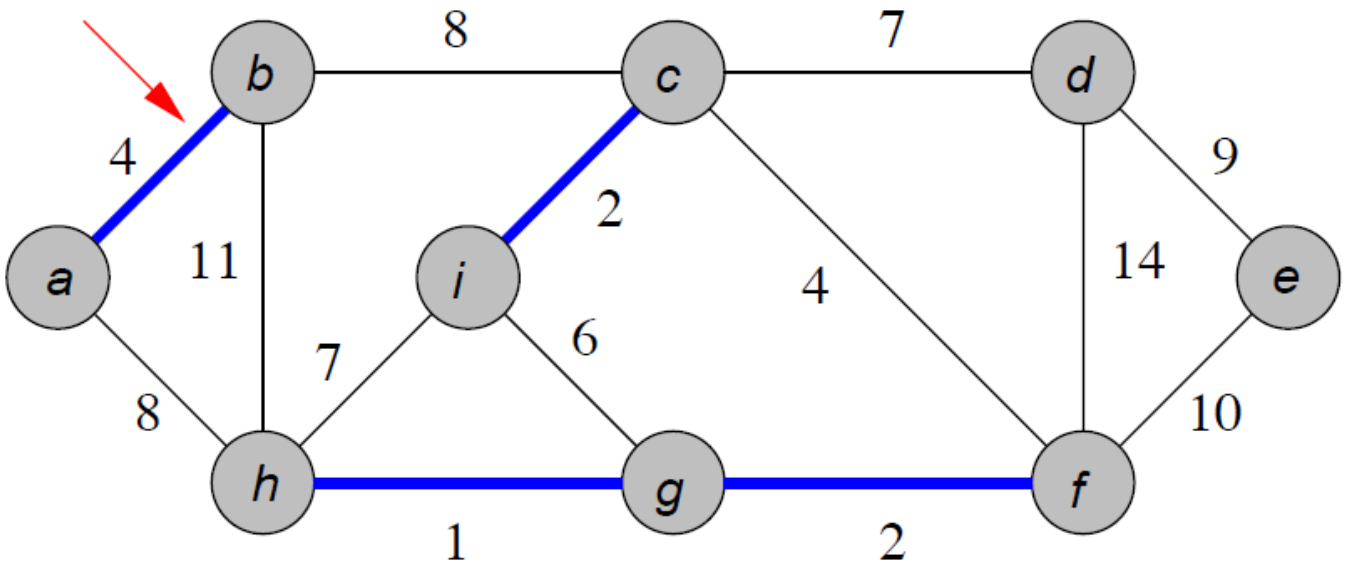
O algoritmo de Kruskal



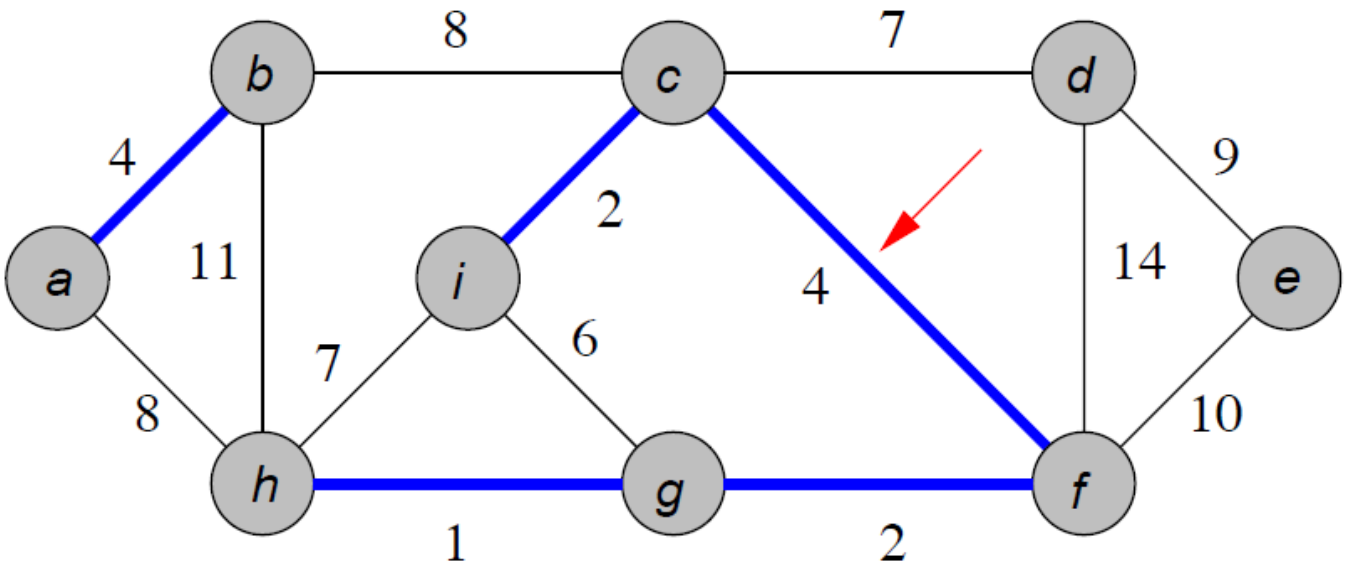
O algoritmo de Kruskal



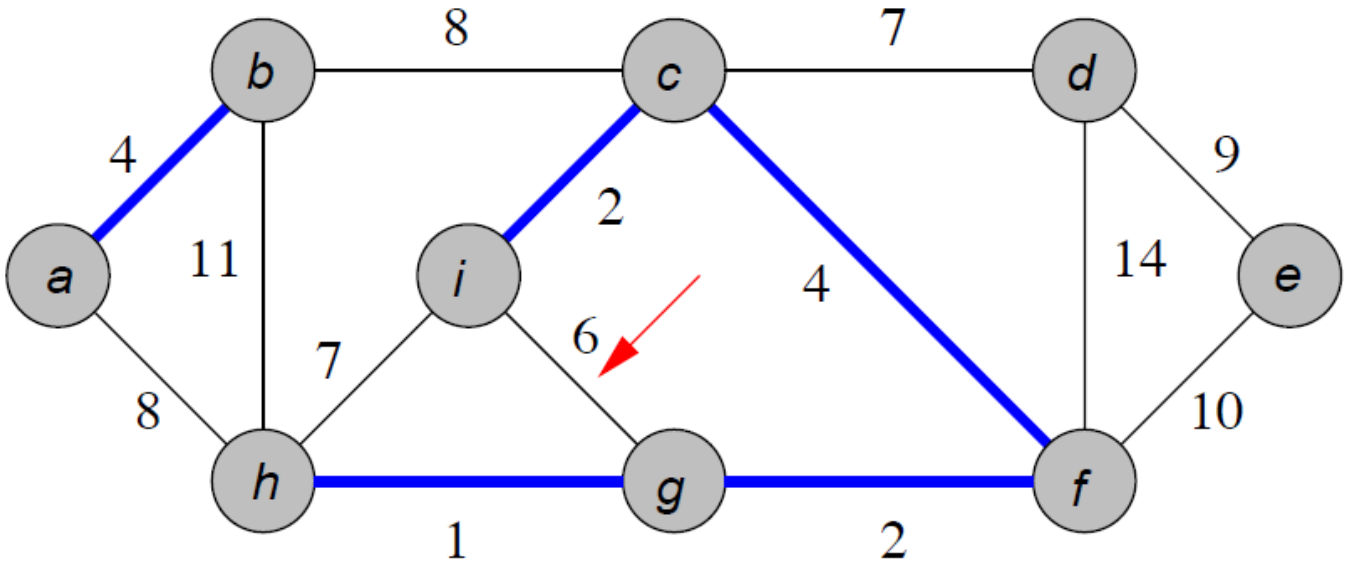
O algoritmo de Kruskal



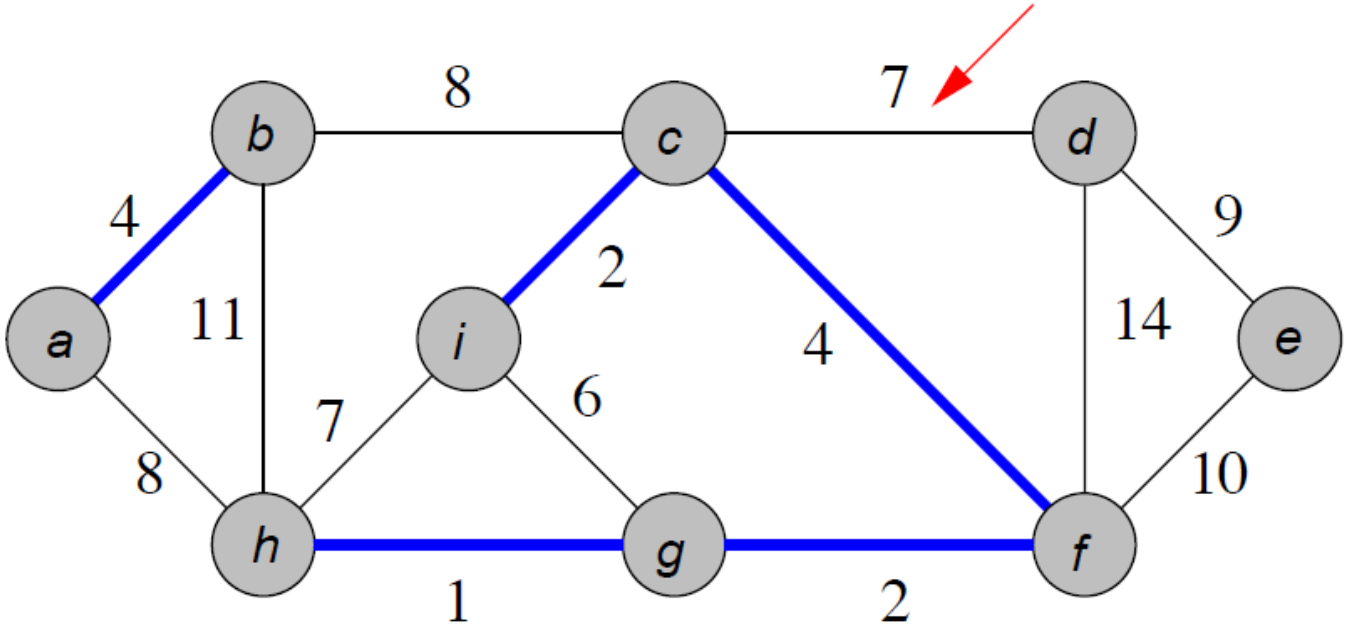
O algoritmo de Kruskal



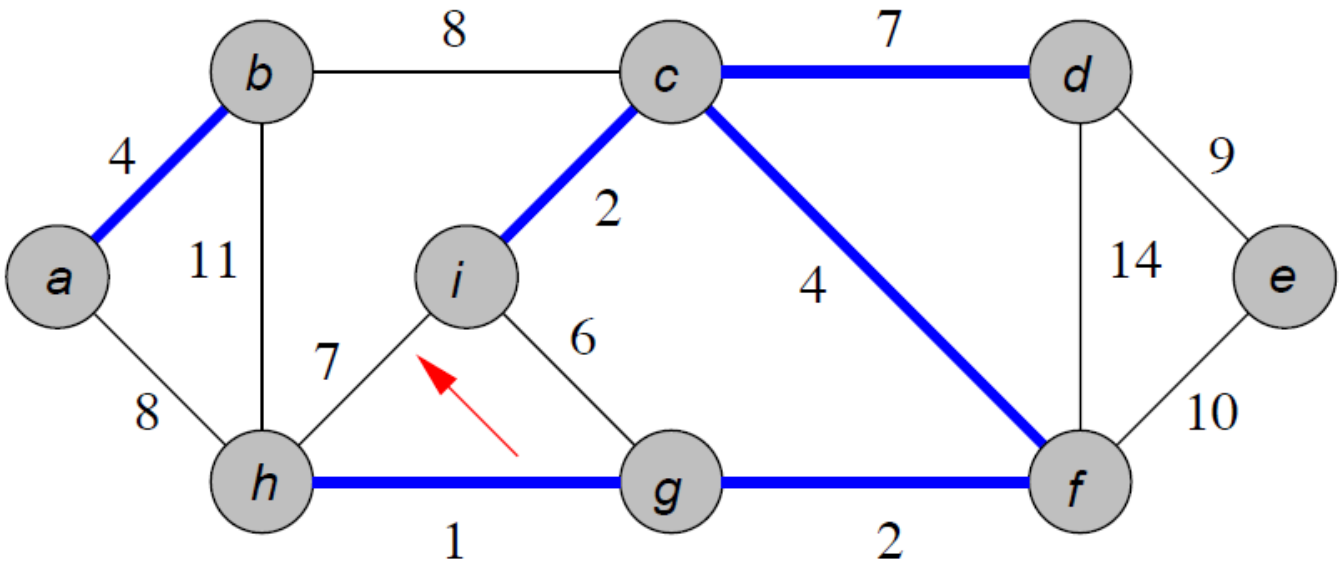
O algoritmo de Kruskal



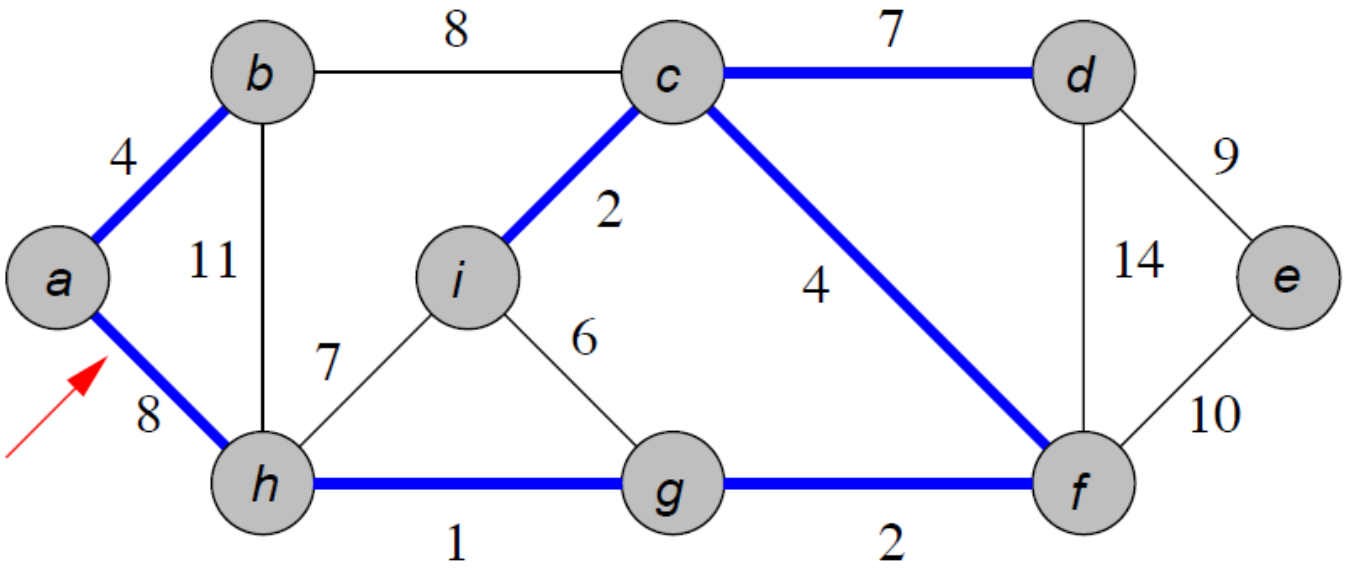
O algoritmo de Kruskal



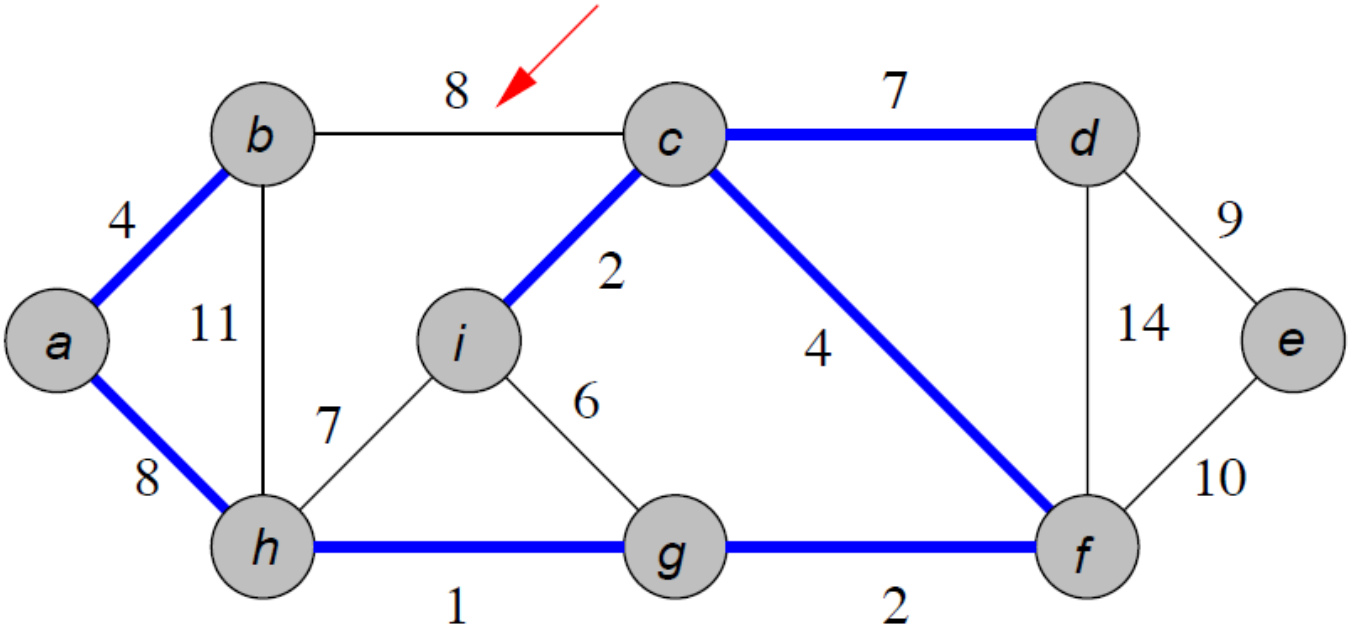
O algoritmo de Kruskal



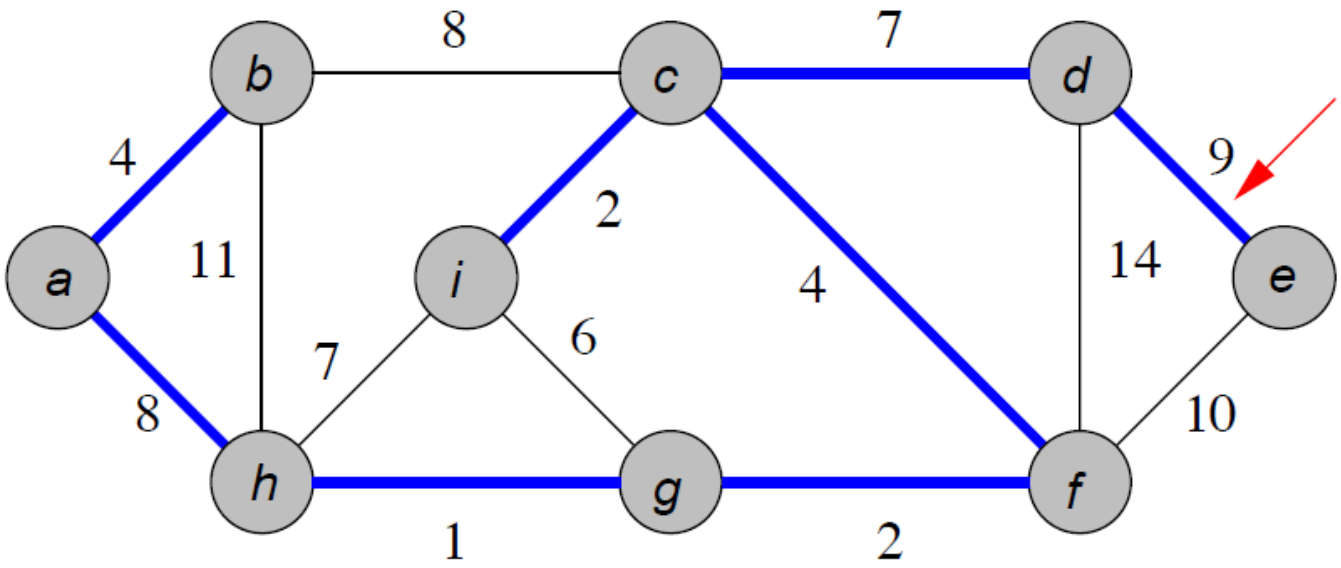
O algoritmo de Kruskal



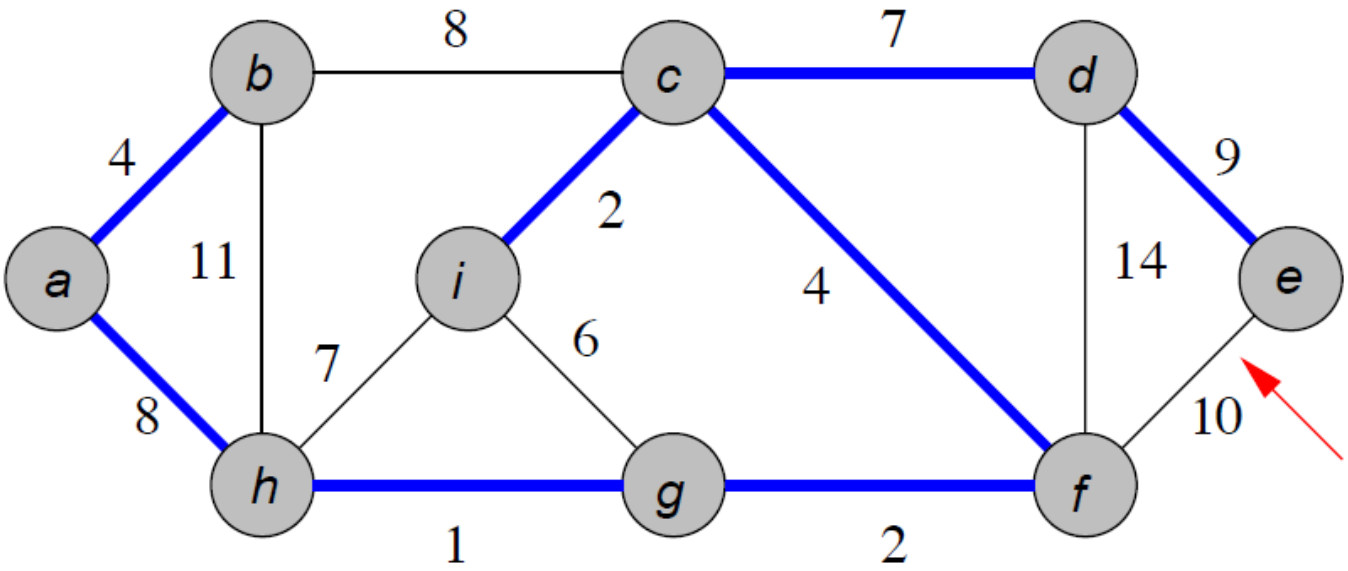
O algoritmo de Kruskal



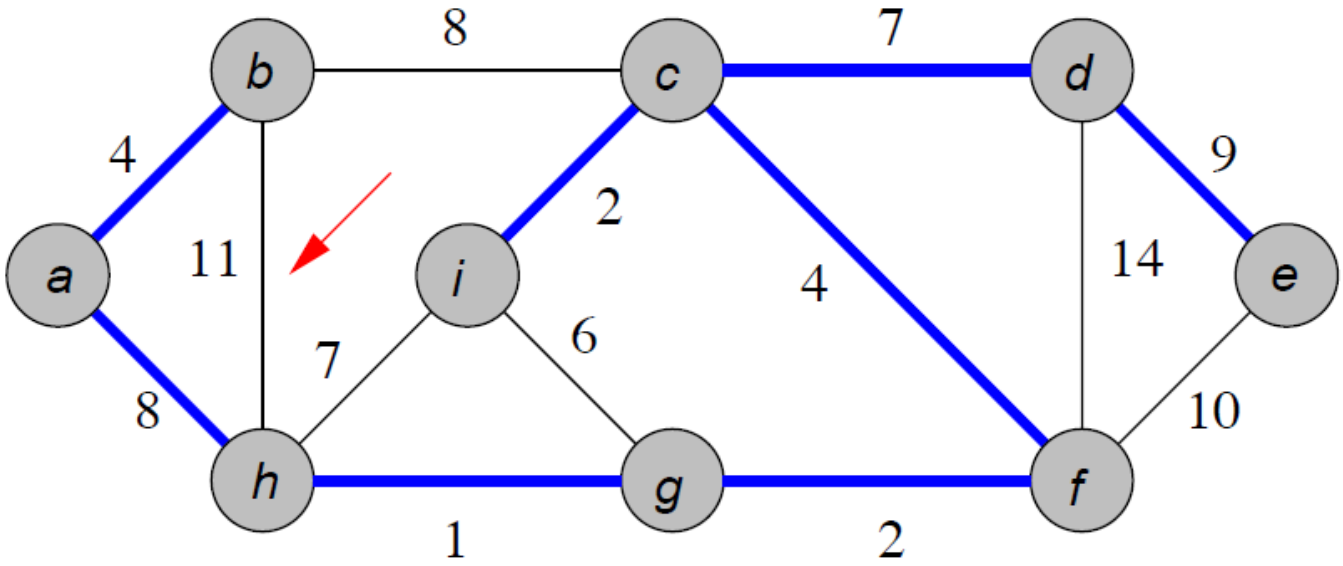
O algoritmo de Kruskal



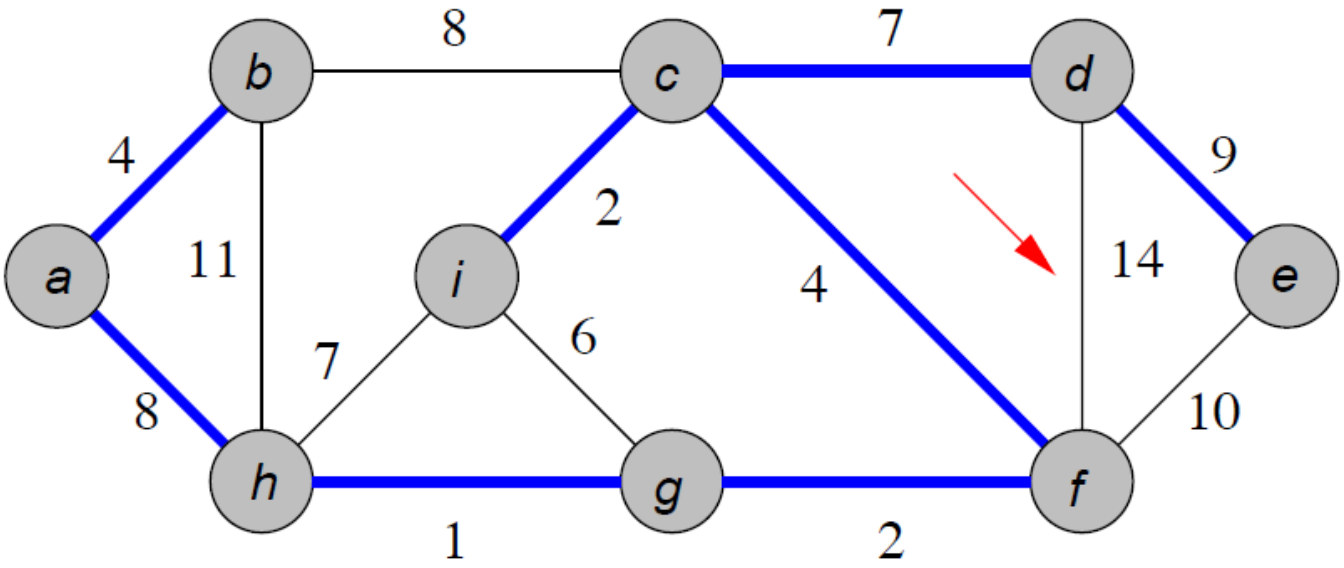
O algoritmo de Kruskal



O algoritmo de Kruskal



O algoritmo de Kruskal



O algoritmo de Kruskal

Eis uma versão 0.0001 do algoritmo de Kruskal.

```
AGM-KRUSKAL(G, w)
1  A ← ∅
2  Ordene as arestas em ordem não-decrescente de peso
3  para cada (u, v) ∈ E nessa ordem faça
4      se u e v estão em componentes distintos de (V, A)
5          então A ← A ∪ {(u, v)}
6  devolva A
```

Problema: Como verificar eficientemente se u e v estão no mesmo componente da floresta $G_A = (V, A)$?

O algoritmo de Kruskal

Inicialmente $G_A = (V, \emptyset)$, ou seja, G_A corresponde à floresta onde cada componente é um vértice isolado.
Ao longo do algoritmo, esses componentes são modificados pela inclusão de arestas em A.
Uma estrutura de dados para representar $G_A = (V, A)$ deve ser capaz de executar eficientemente as seguintes operações:

- Dado um vértice u, determinar o componente de G_A que contém u e
- dados dois vértices u e v em componentes distintos C e C', fazer a união desses em um novo componente.

ED para conjuntos disjuntos

Uma estrutura de dados para conjuntos disjuntos mantém uma coleção $\{S_1, S_2, ..., S_k\}$ de conjuntos disjuntos dinâmicos (isto é, eles mudam ao longo do tempo).

- Cada conjunto é identificado por um representante que é um elemento do conjunto.

  Quem é o representante é irrelevante, mas se o conjunto não for modificado, então o representante não pode mudar.

ED para conjuntos disjuntos

Uma estrutura de dados para conjuntos disjuntos deve ser capaz de executar as seguintes operações:

- MAKE-SET(x): cria um novo conjunto $\{x\}$.
- UNION(x, y): une os conjuntos (disjuntos) que contém x e y, digamos S_x e S_y , em um novo conjunto $S_x \cup S_y$.
  Os conjuntos S_x e S_y são descartados da coleção.
- FIND-SET(x) devolve um apontador para o representante do (único) conjunto que contém x.

O algoritmo de Kruskal

Componentes conexos:

```
CONNECTED-COMPONENTS(G)
1  para cada vértice v ∈ V[G] faça
2      MAKE-SET(v)
3  para cada aresta (u, v) ∈ E[G] faça
4      se FIND-SET(u) ≠ FIND-SET(v)
5          então UNION(u, v)

SAME-COMPONENT(u, v)
1  se FIND-SET(u) = FIND-SET(v)
2      então devolva SIM
3      senão devolva NÃO
```

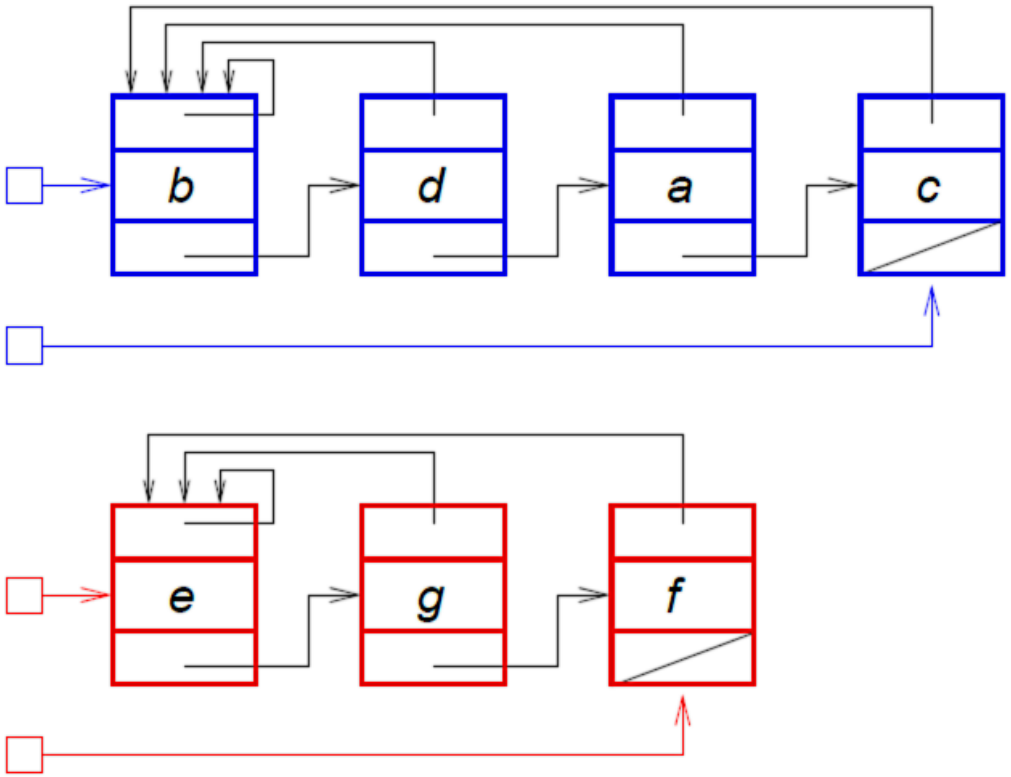
O algoritmo de Kruskal

Eis a versão completa:

```
AGM-KRUSKAL(G, w)
1  A ← ∅
2  para cada v ∈ V[G] faça
3      MAKE-SET(v)
4  Ordene as arestas em ordem não-decrescente de peso
5  para cada (u, v) ∈ E nessa ordem faça
6      se FIND-SET(u) ≠ FIND-SET(v)
7          então A ← A ∪ {(u, v)}
8                UNION(u, v)
9  devolva A
```

O algoritmo de Kruskal

Representação por listas ligadas

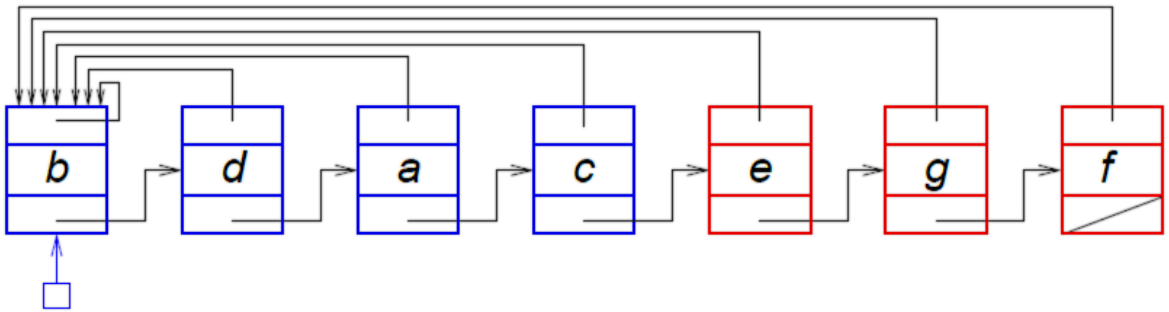


- Cada conjunto tem um representante (início da lista)
- Cada nó tem um campo que aponta para o representante
- Guarda-se um apontador para o fim da lista

O algoritmo de Kruskal

Representação por listas ligadas

- MAKE-SET(x) - $O(1)$
- FIND-SET(x) - $O(1)$
- UNION(x, y) - concatena a lista de x no final da lista de y



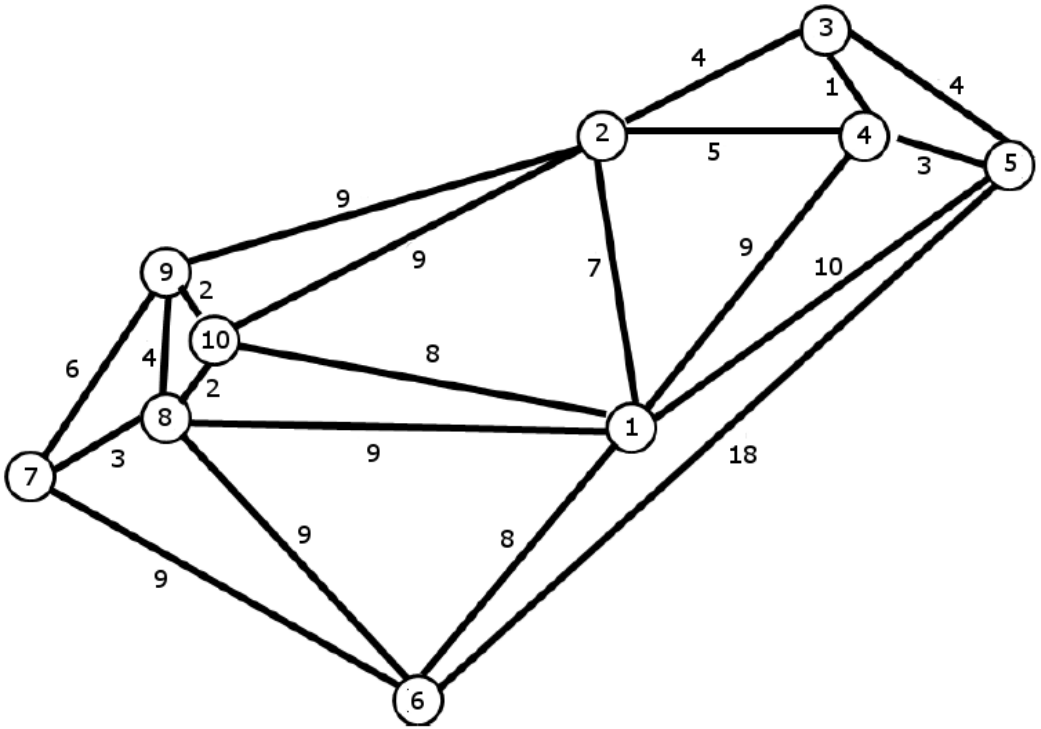
É preciso atualizar os apontadores para o representante.

Referências

- Cormen et. al. *Algoritmos - Teoria e Prática*.
- Nivio Ziviani. *Projeto de Algoritmos com Implementações em Pascal e C*.

Exercício

Considere o grafo abaixo, em seguida faça:



- Execute o algoritmo de Prim e encontre a árvore geradora mínima do grafo acima, considerando o nó 1 como origem.
- Implemente o algoritmo de Prim em linguagem C.