

Árvores Geradoras de Peso Mínimo

Prof. Andrei Braga



Conteúdo

- Problema motivador
- Uma estratégia geral para encontrar árvores geradoras de peso mínimo
- Algoritmo de Prim
- Exercícios
- Referências

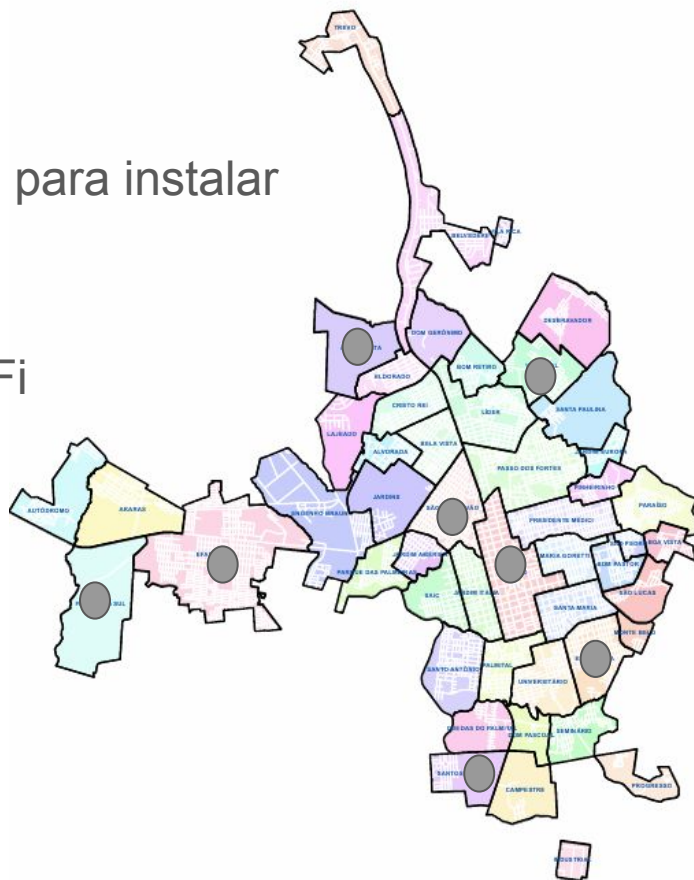
Problema

- Suponha que a nossa empresa foi contratada para instalar pontos de acesso Wi-Fi em Chapecó




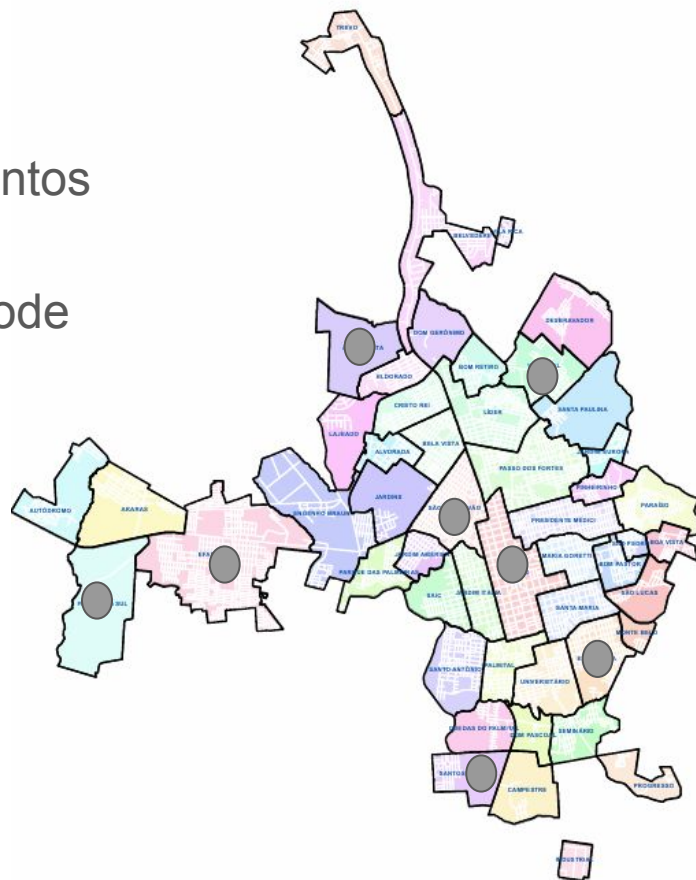
Problema

- Suponha que a nossa empresa foi contratada para instalar pontos de acesso Wi-Fi em Chapecó
- Foram selecionados bairros da cidade onde deverão ser instalados pontos de acesso Wi-Fi
- Junto a um destes pontos, a nossa empresa terá uma estação principal de onde será fornecida a comunicação com a Internet aos pontos de acesso Wi-Fi (neste exemplo, não vamos nos preocupar com onde estará localizada a estação principal)



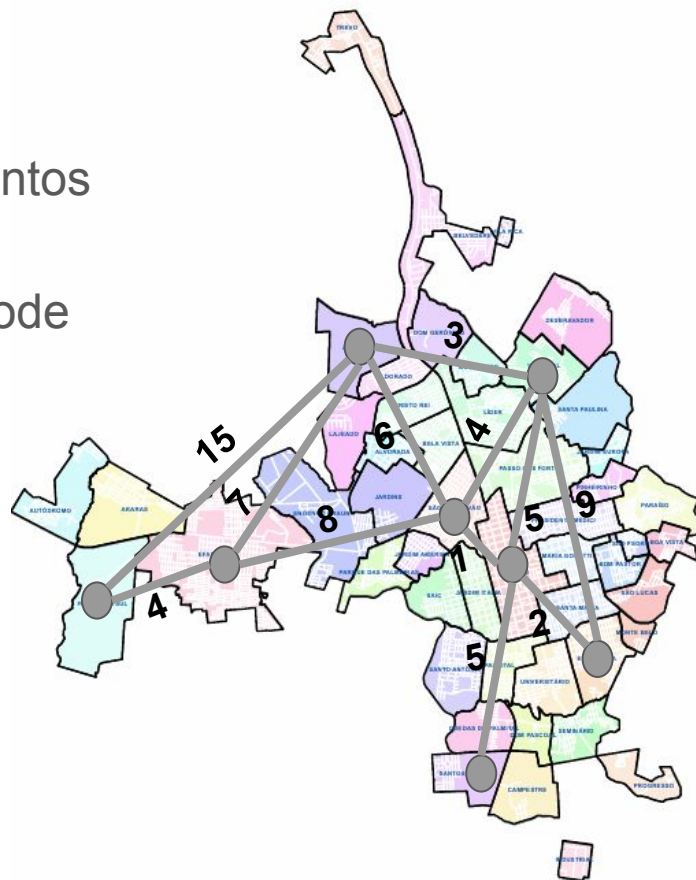
Problema

- Poderemos usar cabos para conectar dois pontos de acesso Wi-Fi
 - Cada conexão por cabo tem um custo, que pode depender da distância entre os pontos que serão conectados e de outros fatores
 - Podemos não considerar conexões entre alguns pares de pontos, por estas conexões serem inviáveis ou por algum outro motivo
- 



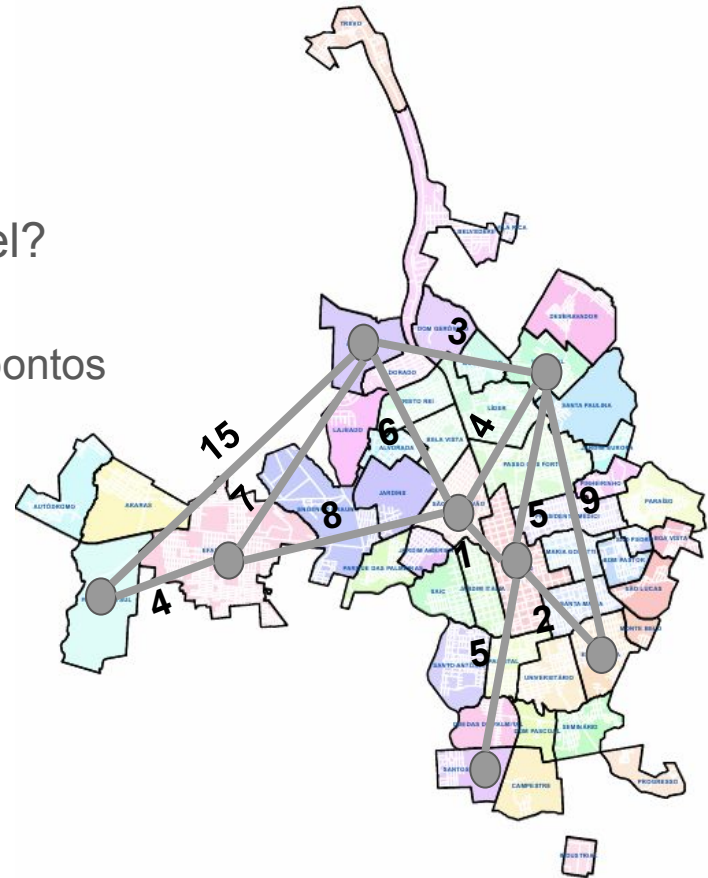
Problema

- Poderemos usar cabos para conectar dois pontos de acesso Wi-Fi
- Cada conexão por cabo tem um custo, que pode depender da distância entre os pontos que serão conectados e de outros fatores
- Podemos não considerar conexões entre alguns pares de pontos, por estas conexões serem inviáveis ou por algum outro motivo



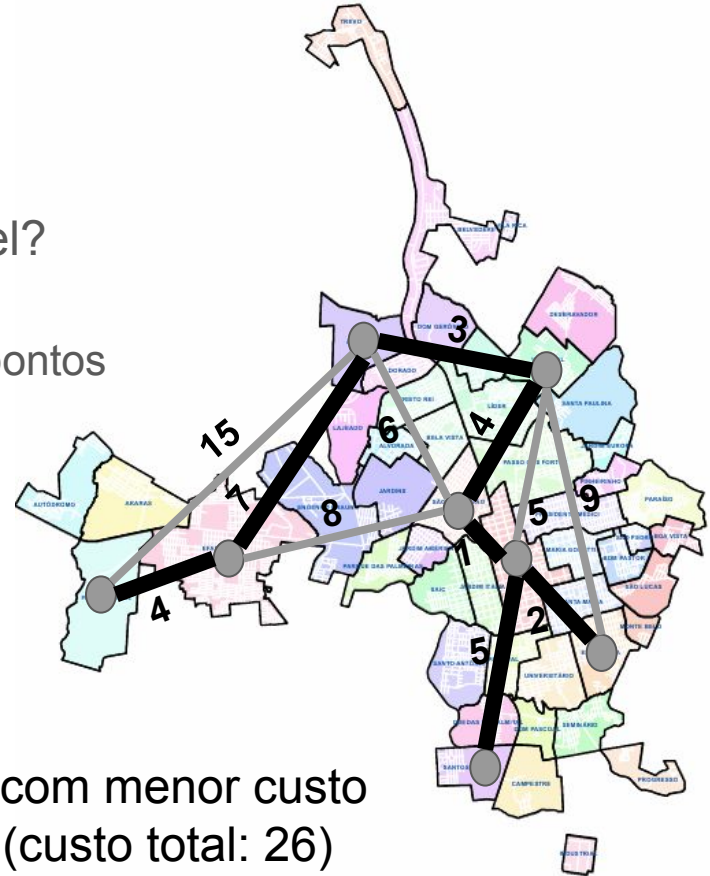
Problema

- **Problema:** Como conectar todos os pontos de acesso Wi-Fi tendo o menor custo possível?
 - **Conectar todos os pontos** de acesso Wi-Fi significa fazer com que, para quaisquer dois pontos de acesso Wi-Fi, exista uma sequência de cabos que conecta estes pontos



Problema

- **Problema:** Como conectar todos os pontos de acesso Wi-Fi tendo o menor custo possível?
 - **Conectar todos os pontos** de acesso Wi-Fi significa fazer com que, para quaisquer dois pontos de acesso Wi-Fi, exista uma sequência de cabos que conecta estes pontos



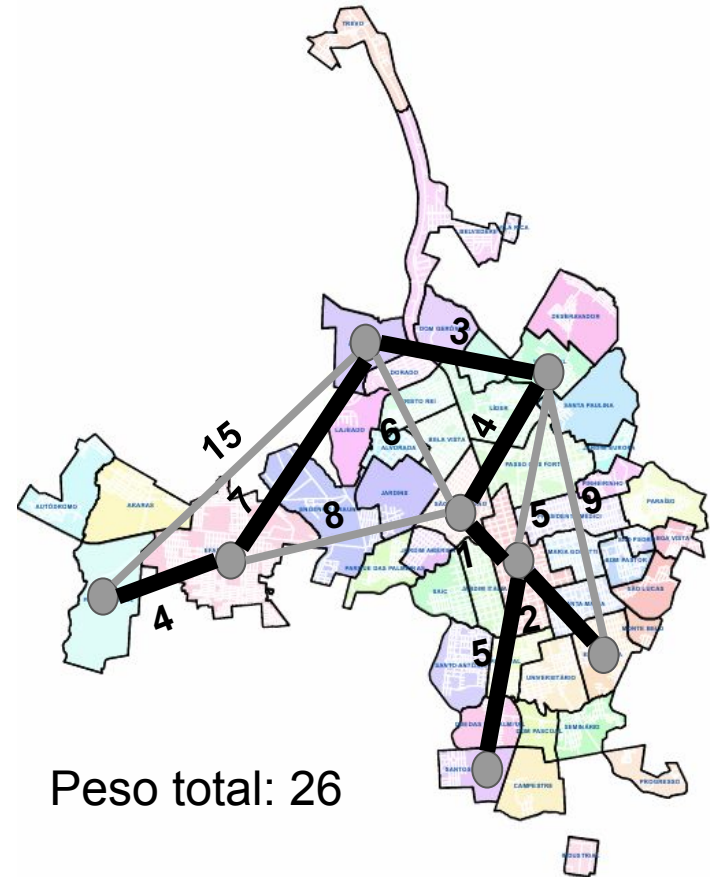
Solução com menor custo possível (custo total: 26)

Problema - Modelagem com grafos

- Podemos modelar este problema usando um grafo G tal que
 - os vértices de G representam os pontos de acesso Wi-Fi e
 - as arestas de G representam as conexões por cabo que podemos realizar,
 - com cada aresta de G tendo um **peso**, que representa o **custo** da conexão correspondente
- O nosso **objetivo** é, então, encontrar um subconjunto de arestas de G que
 - conecte todos os vértices de G ,
 - não contenha ciclos e
 - tenha peso total mínimo

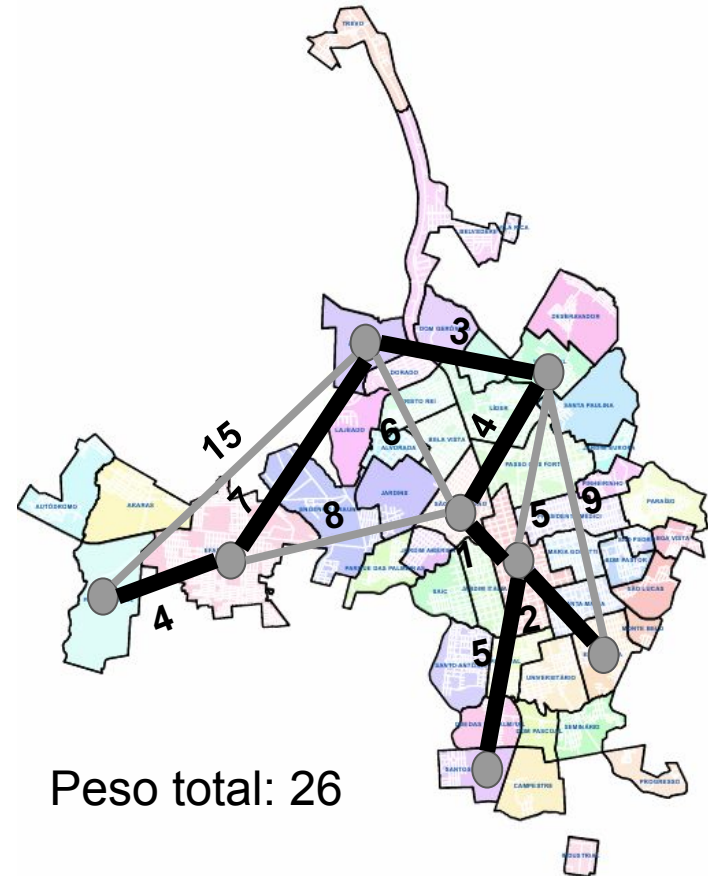
Problema - Modelagem com grafos

- O nosso objetivo é, então, encontrar um subconjunto de arestas de G que
 - conecte todos os vértices de G ,
 - não contenha ciclos e
 - tenha peso total mínimo

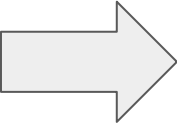


Problema - Modelagem com grafos

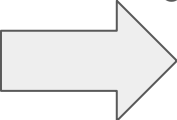
- O nosso objetivo é, então, encontrar um subconjunto de arestas de G que
 - conecte todos os vértices de G ,
 - não contenha ciclos e
 - tenha peso total mínimo
- Podemos definir este objetivo de outra maneira



Problema - Modelagem com grafos

- O nosso objetivo é, então, encontrar um subconjunto de arestas de G que
 - conecte todos os vértices de G ,
 - não contenha ciclos e
 - tenha peso total mínimo
- 
- O nosso objetivo é, então, encontrar um subgrafo T de G que
 - seja gerador (contenha todos os vértices de G) e conexo,
 - seja acíclico e
 - tenha peso mínimo, com o **peso do subgrafo T** sendo igual à soma dos pesos das suas arestas

Problema - Modelagem com grafos

- O nosso objetivo é, então, encontrar um subgrafo T de G que
 - seja gerador (contenha todos os vértices de G) e conexo,
 - seja acíclico e
 - tenha peso mínimo, com o peso do subgrafo T sendo igual à soma dos pesos das suas arestas
 - Um subgrafo de G que é gerador, conexo e acíclico é denominado uma **árvore geradora** de G
 - O nosso objetivo é, então, encontrar uma árvore geradora T de G que tenha peso mínimo, com o **peso da árvore geradora T** sendo igual à soma dos pesos das suas arestas
- 

Problema - Modelagem com grafos

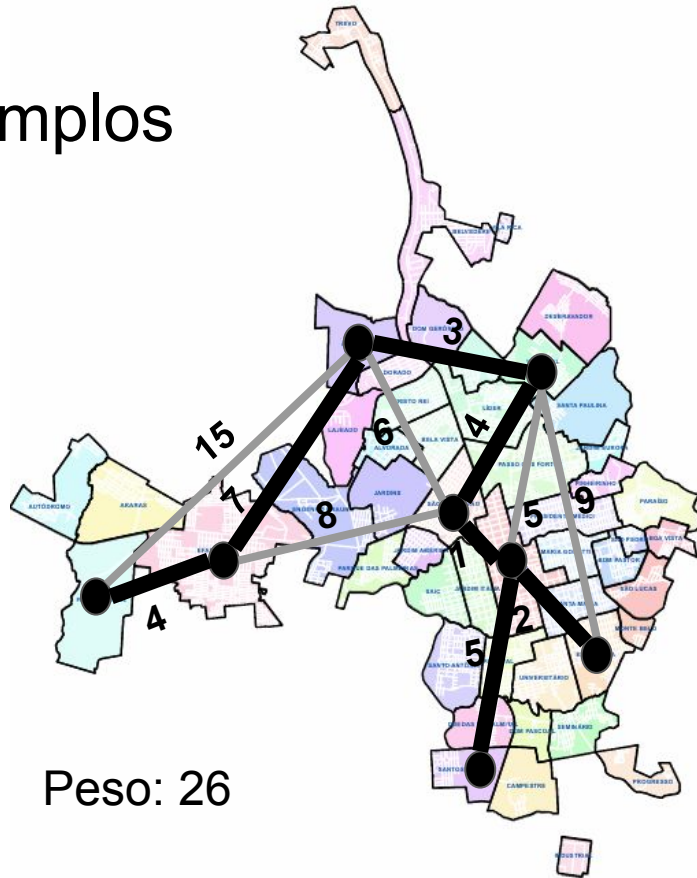
- O nosso objetivo é, então, encontrar uma árvore geradora T de G que tenha peso mínimo, com o peso da árvore geradora T sendo igual à soma dos pesos das suas arestas



- O nosso objetivo é, então, encontrar uma **árvore geradora de peso mínimo** de G

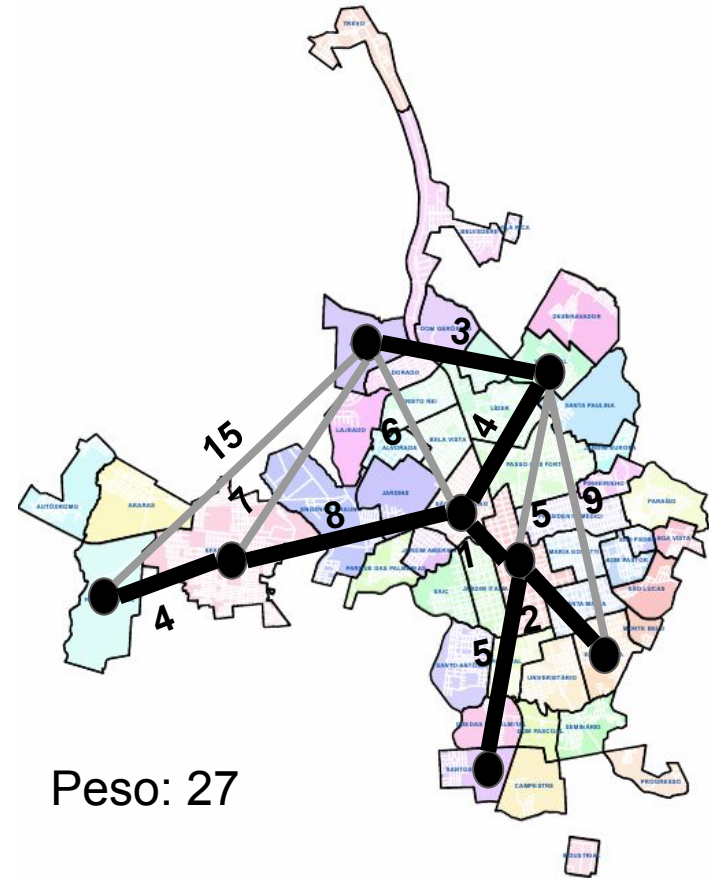


Exemplos



Peso: 26

Árvore geradora de
peso mínimo



Peso: 27

Árvore geradora de
peso mínimo

Algoritmos

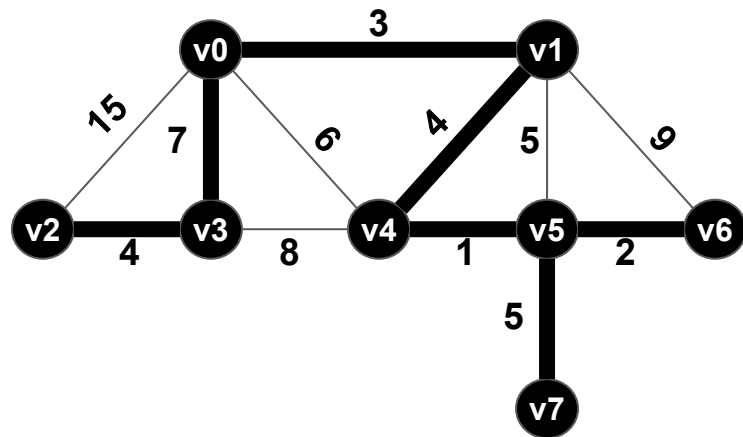
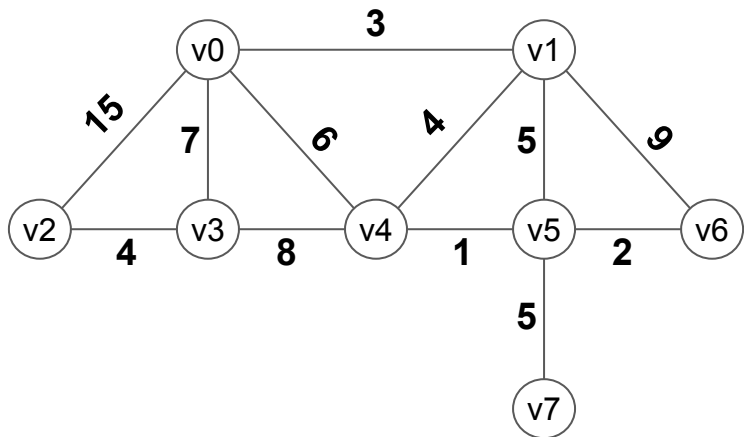
- O problema de encontrar uma árvore geradora de peso mínimo de um grafo com pesos nas arestas tem sido estudado (pelo menos) desde os anos 1920
- Para este problema, vamos considerar que o **grafo** recebido como **entrada** é **conexo** (caso contrário, o problema não admite solução)
- Veremos dois algoritmos para resolver o problema: o **Algoritmo de Prim** e o **Algoritmo de Kruskal**
- Antes disso, vamos examinar uma estratégia geral para encontrar uma árvore geradora de peso mínimo de um grafo

Aresta segura

- Seja G um grafo conexo com pesos nas arestas e T uma árvore com a seguinte propriedade:
 T está contida em alguma árvore geradora de peso mínimo de G .
Uma aresta uv de G é **segura para T** se T continua sendo uma árvore com a mesma propriedade depois de uv ser adicionada a T

Aresta segura

- Exemplo:

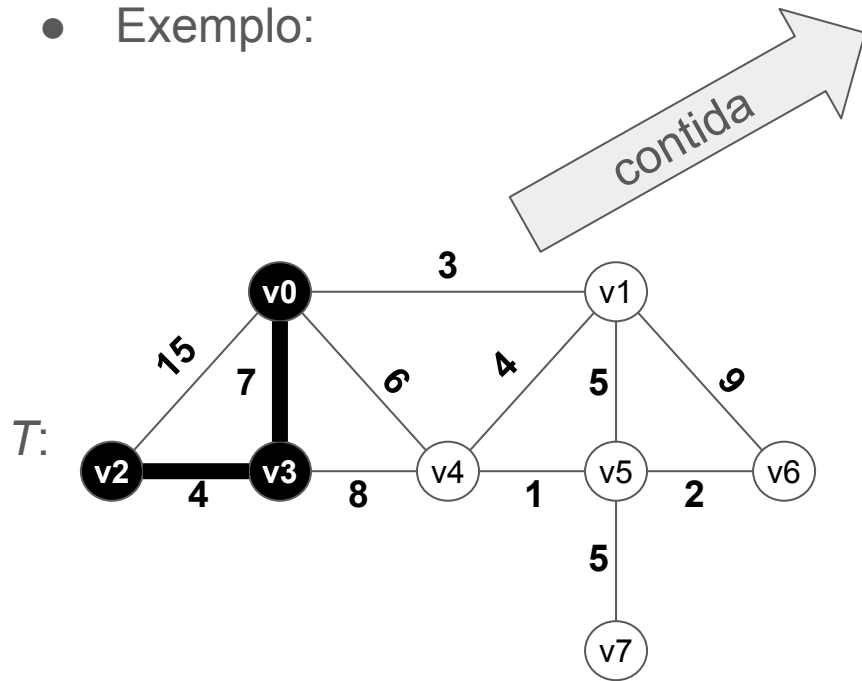
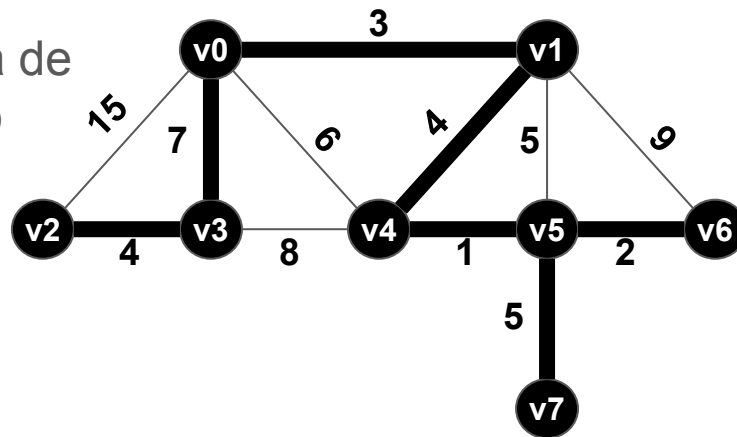


Árvore geradora de
peso mínimo

Aresta segura

- Exemplo:

Árvore geradora de peso mínimo

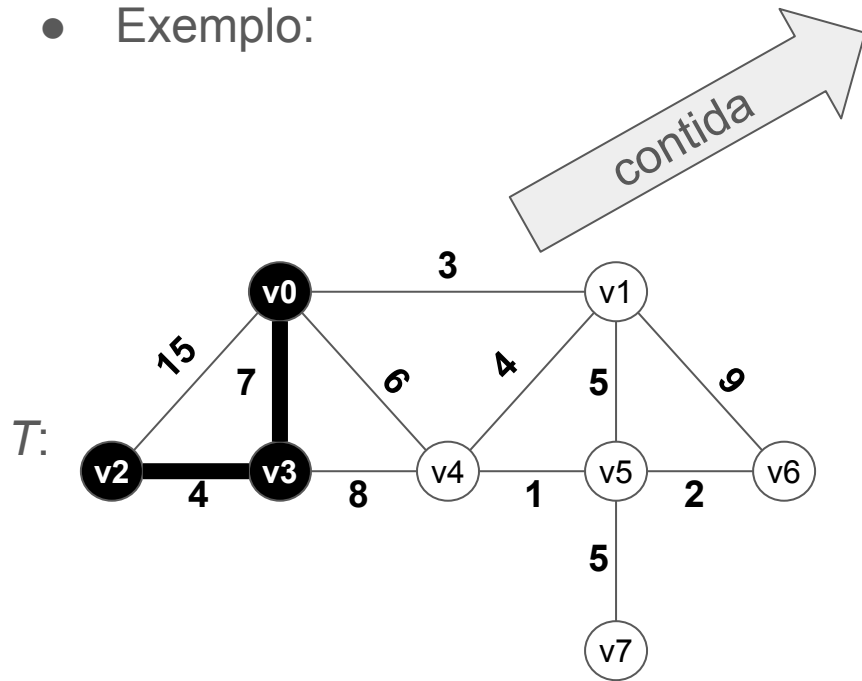


Árvore contida em uma árvore geradora de peso mínimo

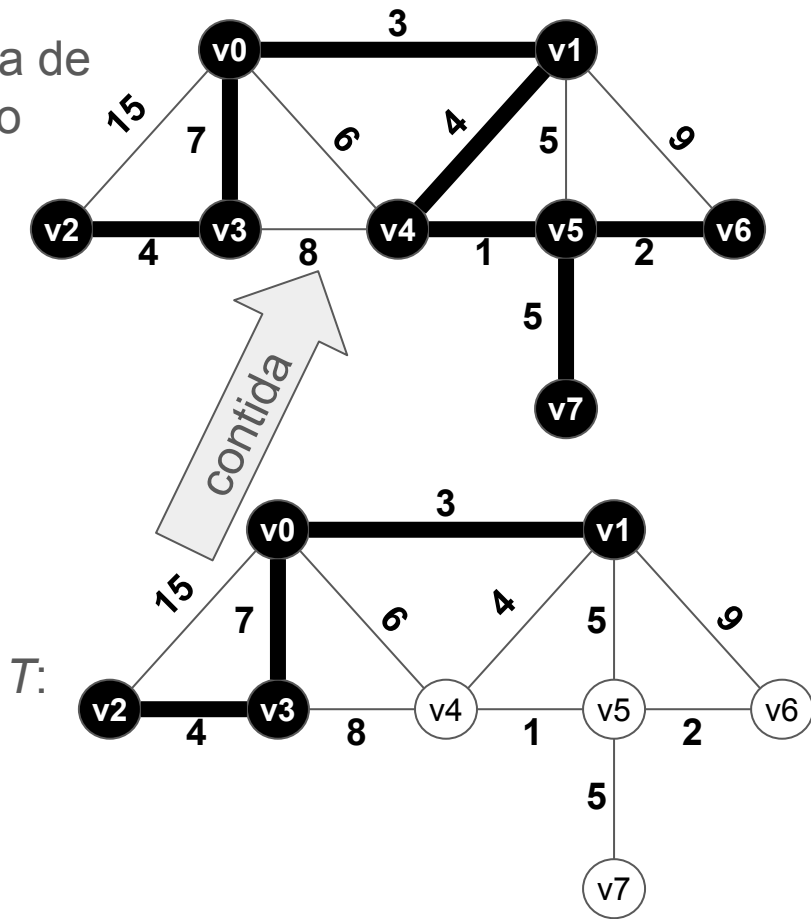
Aresta segura

- Exemplo:

Árvore geradora de peso mínimo



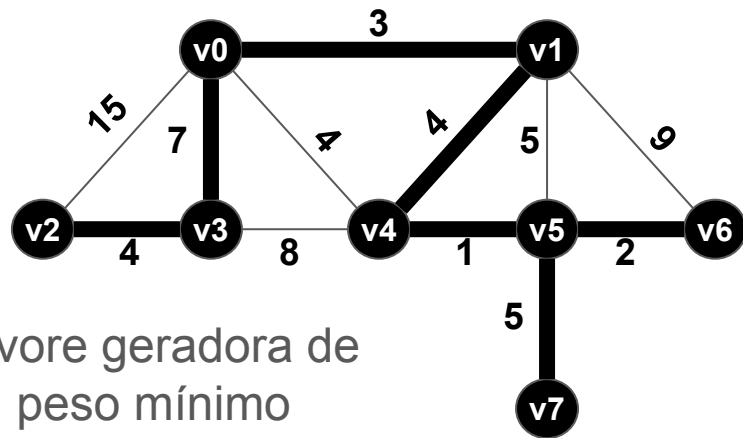
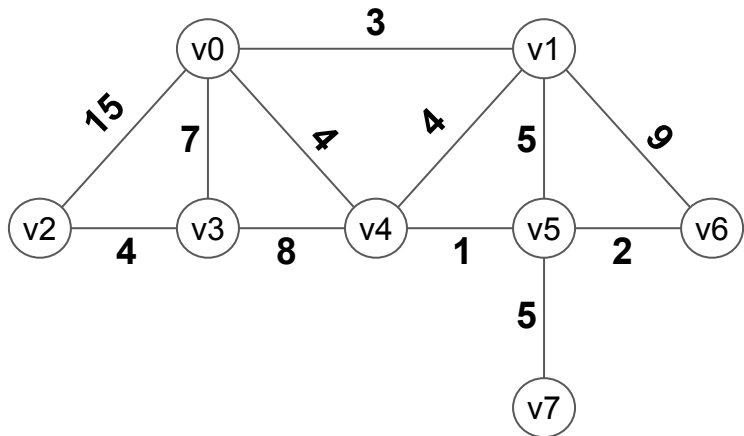
Árvore contida em uma árvore geradora de peso mínimo



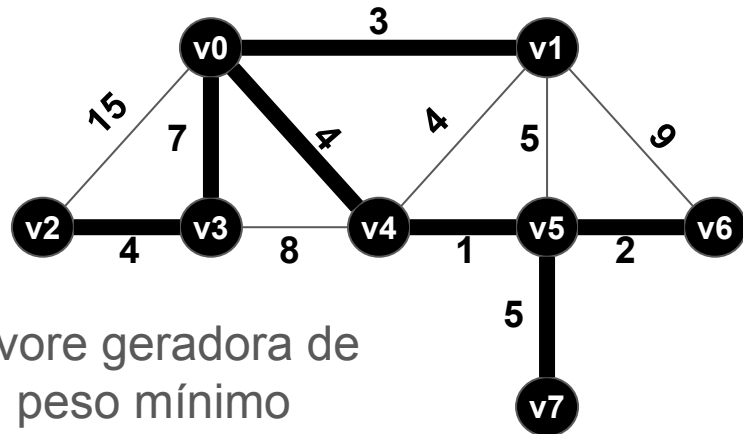
v_0v_1 é uma aresta segura para T

Aresta segura

- Exemplo:



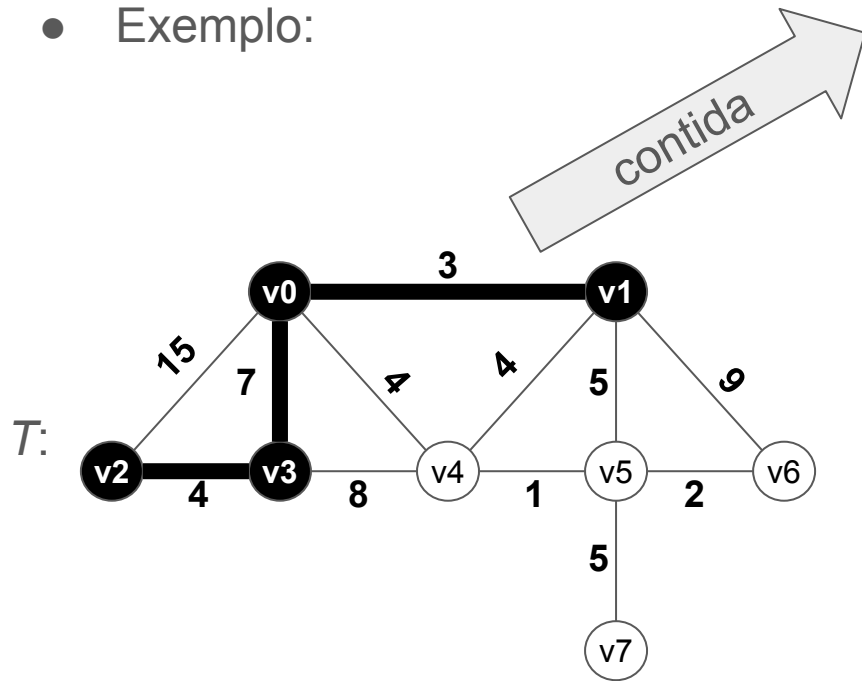
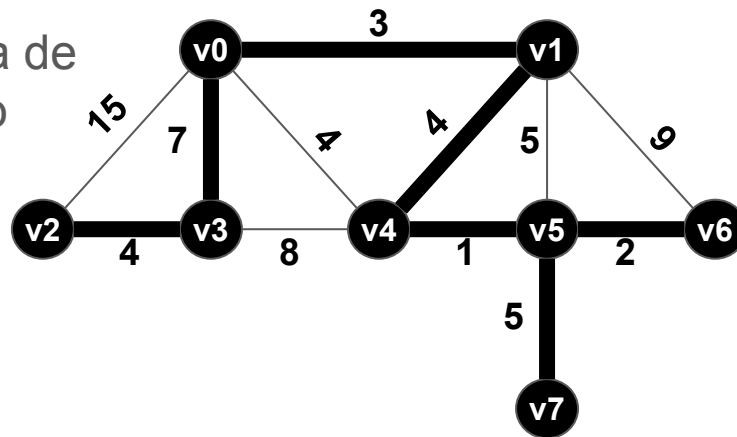
Árvore geradora de peso mínimo



Aresta segura

- Exemplo:

Árvore geradora de peso mínimo

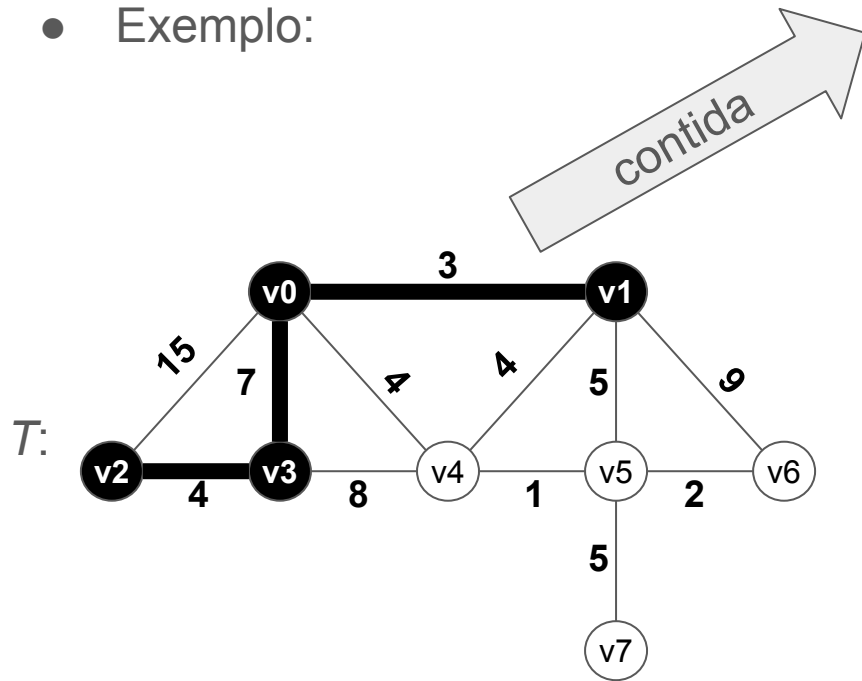


Árvore contida em uma árvore geradora de peso mínimo

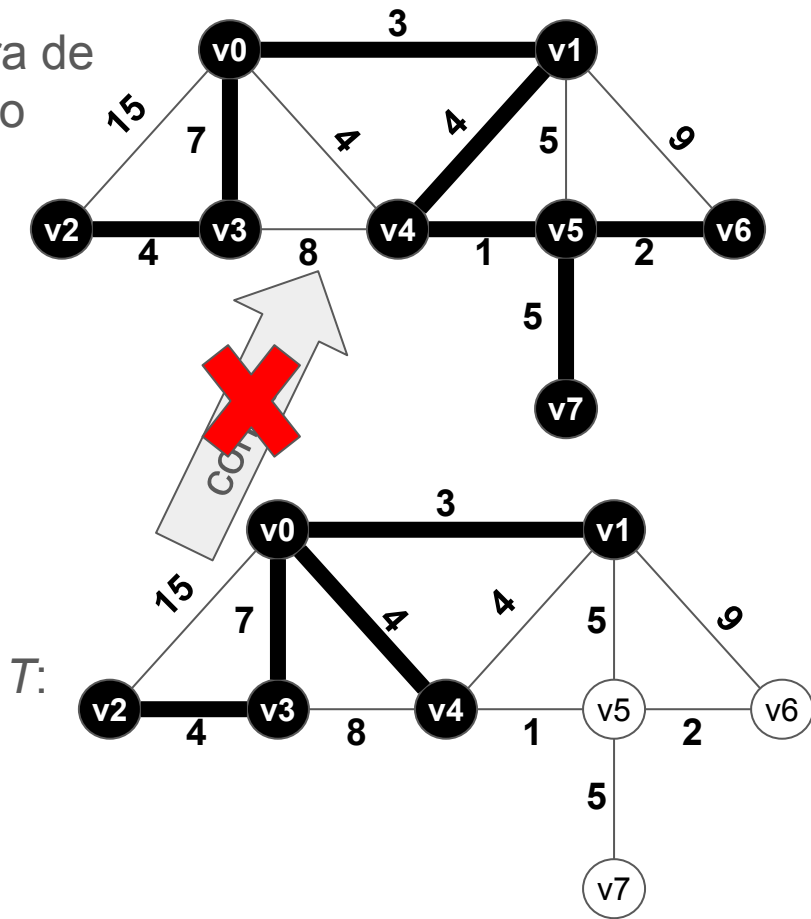
Aresta segura

- Exemplo:

Árvore geradora de peso mínimo



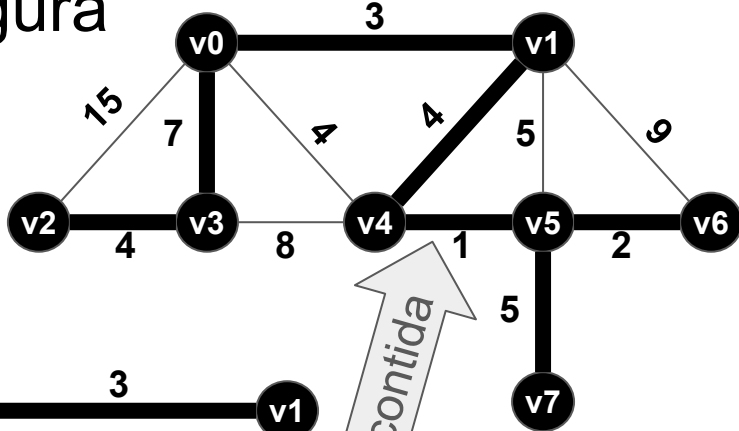
Árvore contida em uma árvore geradora de peso mínimo



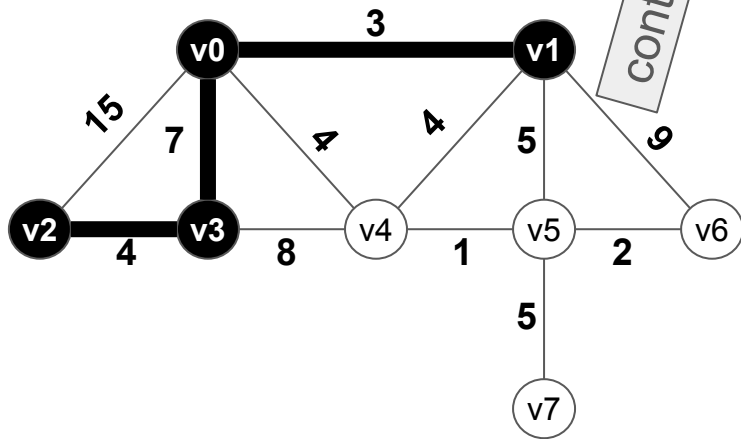
Aresta segura

- Exemplo:

Árvore geradora de peso mínimo

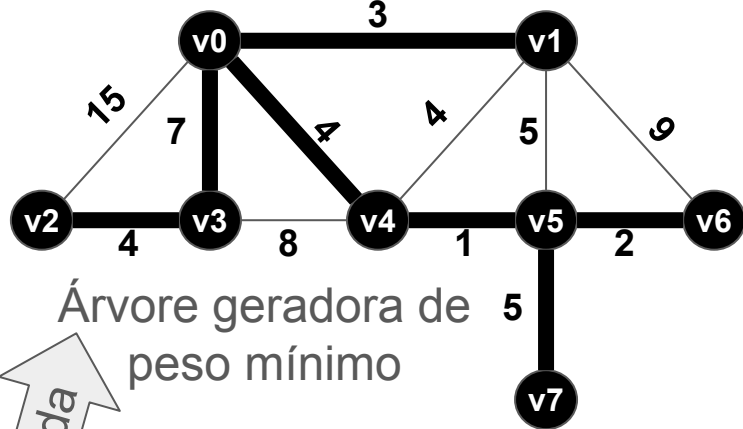


T :



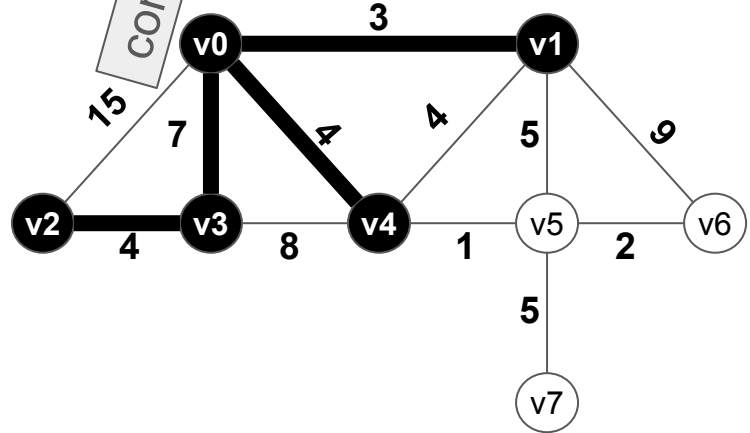
Árvore contida em uma árvore geradora de peso mínimo

Árvore geradora de peso mínimo



contida

T :



v_0v_4 é uma aresta segura para T

Como encontrar uma árvore geradora de peso mínimo

EncontraArvGerPesMin(G conexo)

Inicialmente, T é uma árvore que consiste apenas no vértice 0 de G

1. $T = (\{0\}, \emptyset)$ ←
2. Enquanto T não é uma árvore geradora de G :
3. Encontre uma aresta uv de G que é segura para T
4. Adicione uv a T ←
5. Retorne T

Ao fim do laço das linhas 2-4, temos uma árvore T

- que contém todos os vértices de G e
- que está contida em alguma árvore geradora de peso mínimo de G

Então, temos uma árvore geradora de peso mínimo de G !

Ao fim de cada iteração do laço das linhas 2-4, temos uma árvore T

- que tem 1 vértice a mais e
- que está contida em alguma árvore geradora de peso mínimo de G

Como encontrar uma árvore geradora de peso mínimo

EncontraArvGerPesMin(G conexo)

1. $T = (\{0\}, \emptyset)$
2. Enquanto T não é uma árvore geradora de G :
3. **Encontre** uma **aresta** uv de G que é **segura para T**
4. Adicione uv a T
5. Retorne T

Como podemos fazer isso?

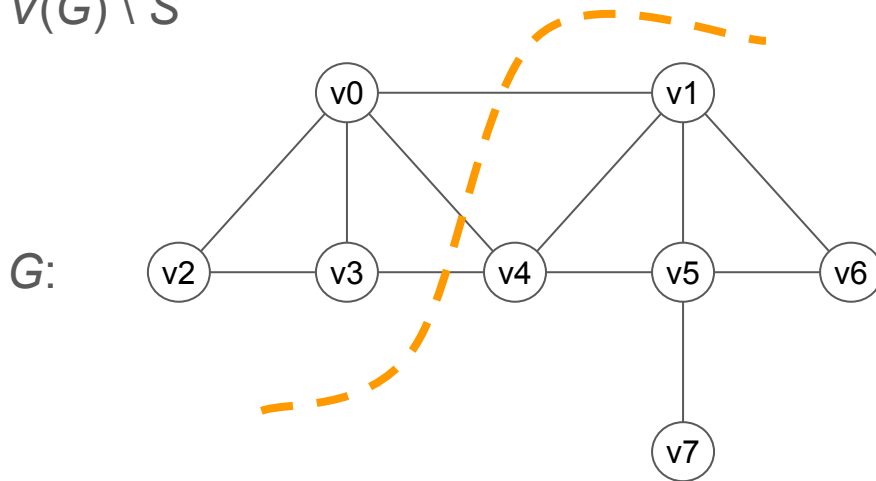
Algoritmo de Prim

- O Algoritmo de Prim se baseia em um resultado interessante
- Antes de descrever este resultado vamos definir alguns conceitos importantes

Corte

- Um **corte** em um grafo G é uma partição $(S, V(G) \setminus S)$ do conjunto de vértices de G em dois subconjuntos disjuntos (e não vazios): o subconjunto S e o subconjunto $V(G) \setminus S$

- Exemplo:

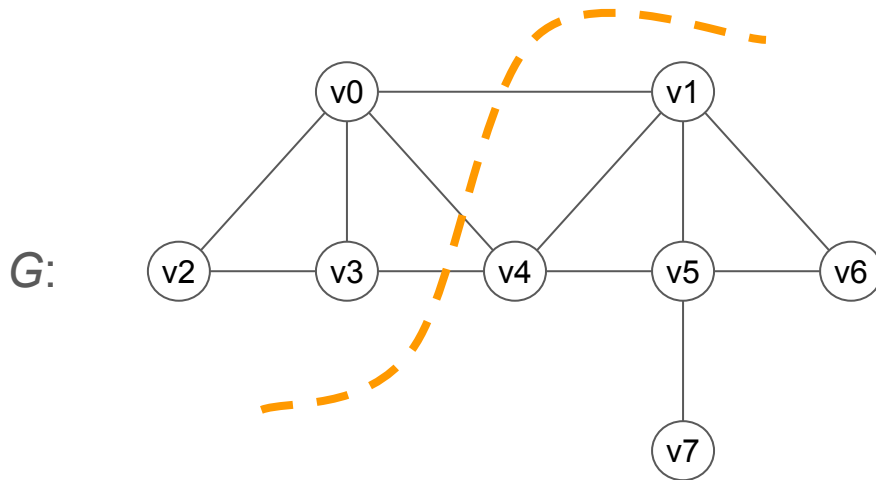


$(\{v_0, v_2, v_3\}, \{v_1, v_4, v_5, v_6, v_7\})$ é um corte de G

Corte

- Dado um corte $(S, V(G) \setminus S)$ em um grafo G , uma **aresta do corte** é uma aresta que tem um extremo em S e um extremo em $V(G) \setminus S$

- Exemplo:

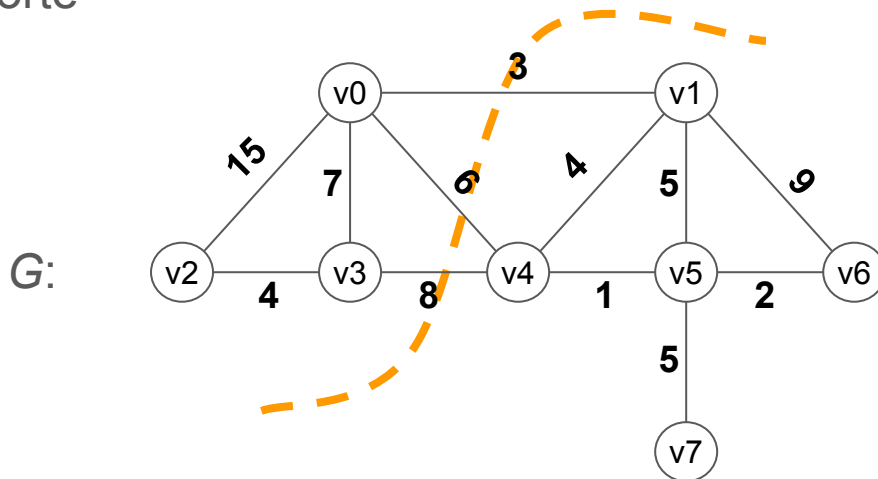


- v_0v_4 e v_3v_4 são arestas do corte
- v_0v_2 e v_5v_6 **não** são arestas do corte

Corte

- Dado um corte $(S, V(G) \setminus S)$ em um grafo G com pesos nas arestas, uma **aresta de peso mínimo do corte** é uma aresta de peso mínimo dentre as arestas do corte

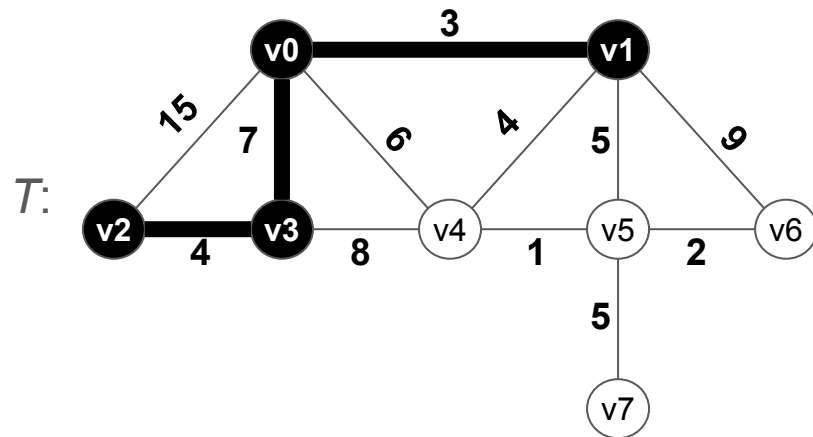
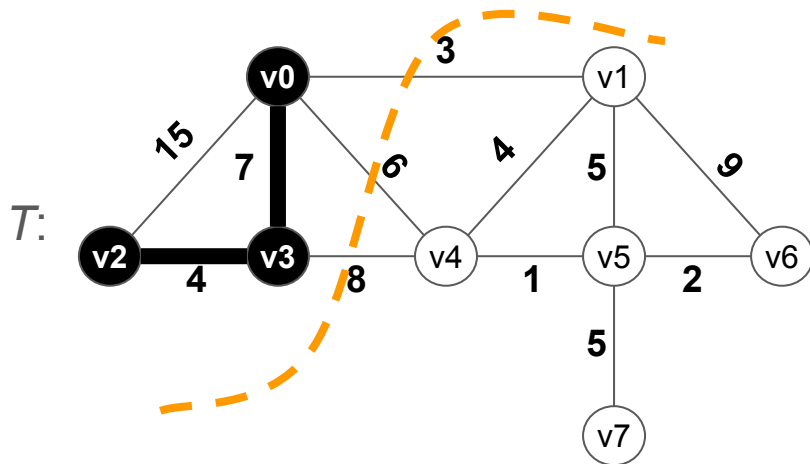
- Exemplo:



- v_0v_1 é uma aresta de peso mínimo do corte
- v_3v_4 **não** é uma aresta de peso mínimo do corte

Aresta segura em um corte

- Teorema:** Seja G um grafo conexo com pesos nas arestas e T uma árvore com a seguinte propriedade: T está contida em alguma árvore geradora de peso mínimo de G . Se uv é uma aresta de peso mínimo do corte $(V(T), V(G) \setminus V(T))$, então uv é uma aresta segura para T



Aresta segura em um corte

- **Prova:**

- Seja T' uma árvore geradora de peso mínimo de G que contém T
- Vamos construir uma árvore geradora T'' de peso mínimo de G que contém T adicionada de uv (relembre o exemplo do [Slide 25](#))
- Com isso, o teorema estará provado

- Se T' contém uv , então fazemos simplesmente $T'' = T'$
- Suponha, então, que T' não contém uv

Aresta segura em um corte

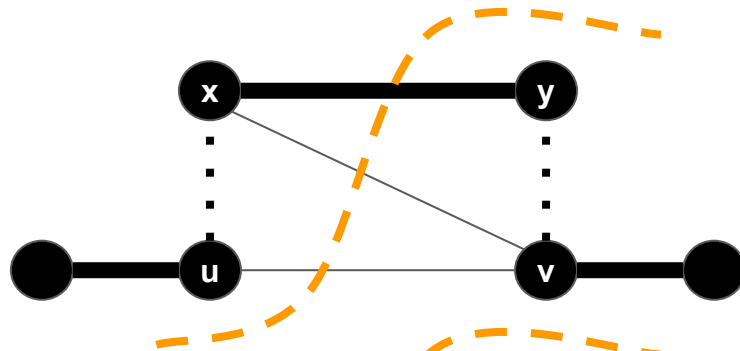
- **Prova:**

- Como T' é uma árvore geradora de G , existe um único caminho P entre u e v em T'
- Defina T'' como T' adicionada de uv . T'' contém um único ciclo, que é formado pelo caminho P adicionado de uv
- Já que u e v estão em subconjuntos diferentes do corte $(V(T), V(G) \setminus V(T))$, existe, no caminho P , uma aresta xy do corte
- Note que a aresta xy não está contida em T , pois nenhuma aresta do corte $(V(T), V(G) \setminus V(T))$ está contida em T
- Remova xy de T'' . Agora, T'' é uma árvore geradora de G que contém T adicionada de uv

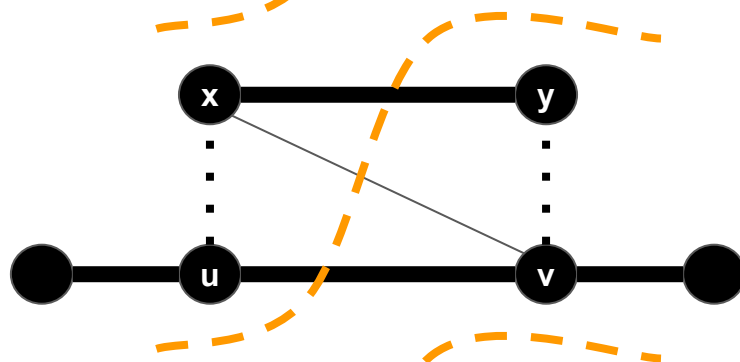
Aresta segura em um corte

- Prova:

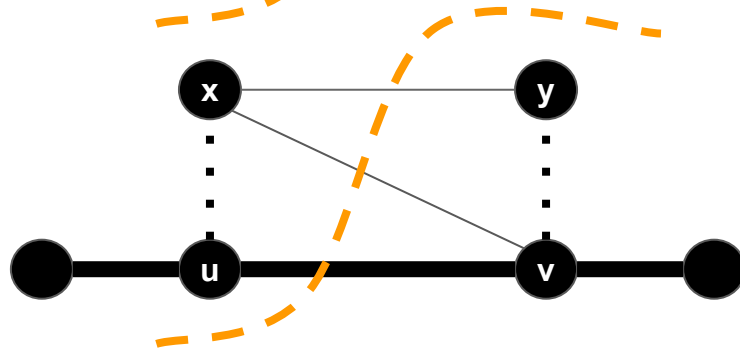
T' :



$T'' = T'$ adicionada de uv :



$T'' = T''$ com a remoção de xy :



Aresta segura em um corte

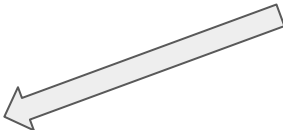
- **Prova:**

- Sendo uv uma aresta de peso mínimo do corte $(V(T), V(G) \setminus V(T))$, o peso de uv é menor ou igual ao peso de xy
- O peso de T'' é igual a
o peso de $T' +$ o peso de $uv -$ o peso de xy
- Então, o peso de T'' é menor ou igual ao peso de T'
- Como T' é uma árvore geradora de peso mínimo de G , T'' também é uma árvore geradora de peso mínimo de G
- Portanto, T'' é uma árvore geradora de peso mínimo de G que contém T adicionada de uv \square

Algoritmo de Prim

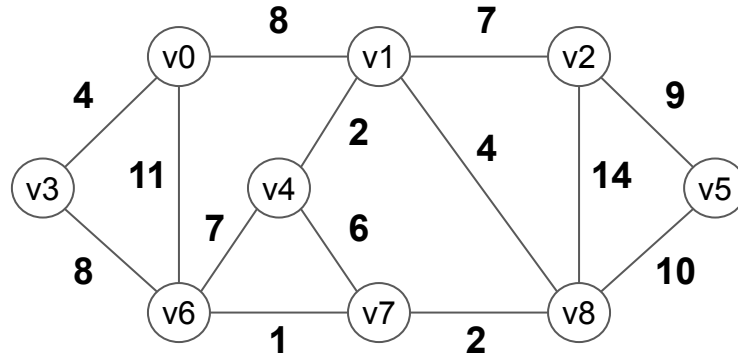
Prim(G conexo)

Inicialmente, T é uma árvore que consiste apenas no vértice 0 de G

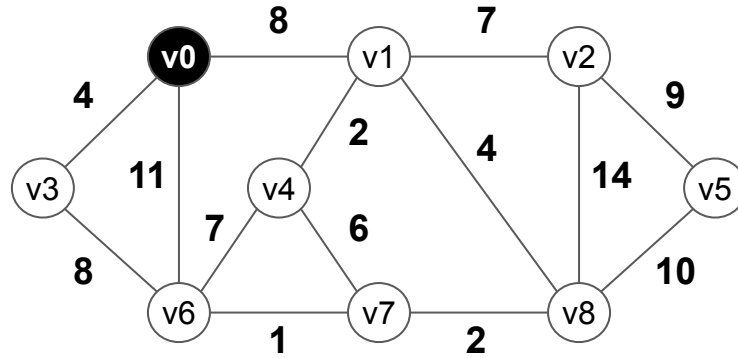


1. $T = (\{0\}, \emptyset)$
2. Enquanto T não é uma árvore geradora de G :
3. Encontre uma aresta uv de peso mínimo do corte $(V(T), V(G) \setminus V(T))$
4. Adicione uv a T
5. Retorne T

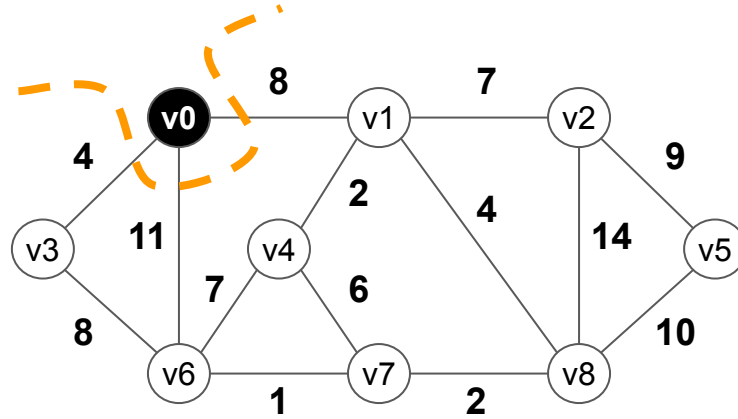
Algoritmo de Prim



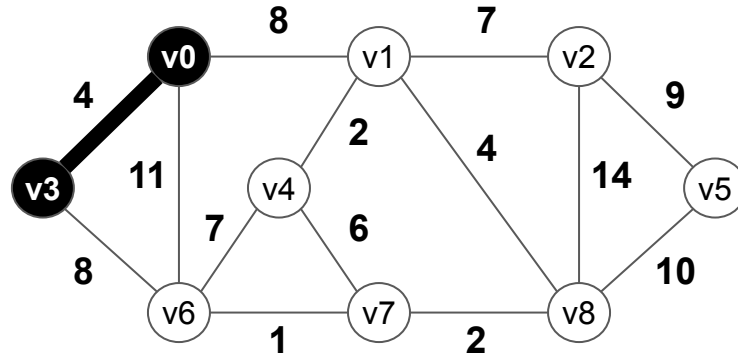
Algoritmo de Prim



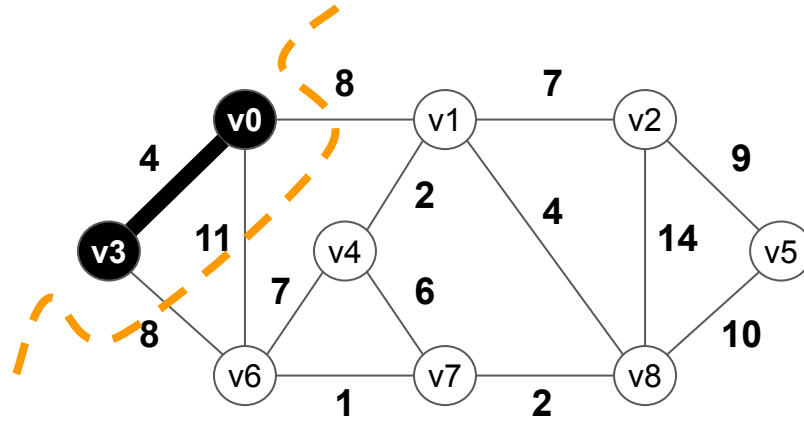
Algoritmo de Prim



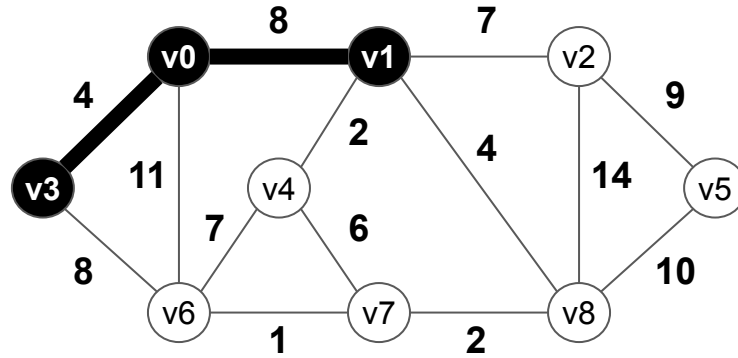
Algoritmo de Prim



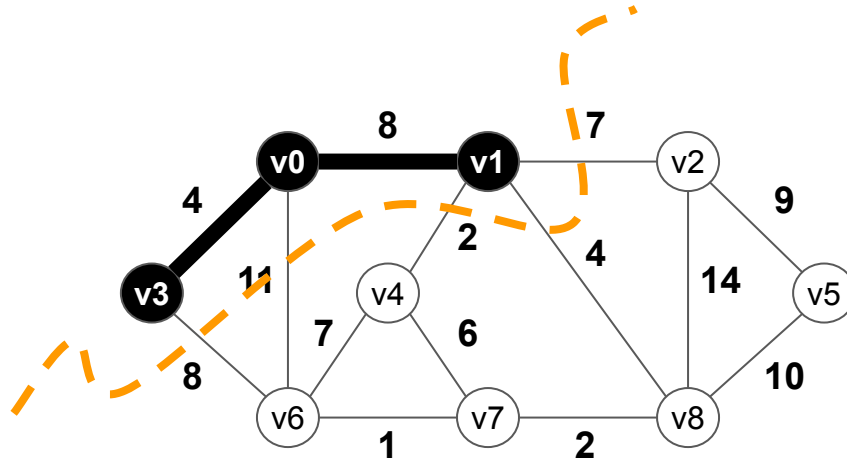
Algoritmo de Prim



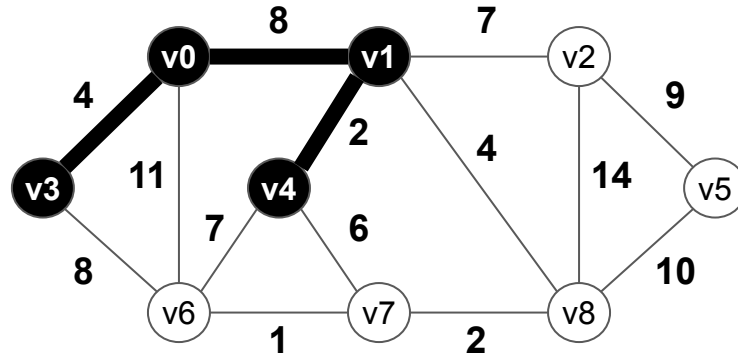
Algoritmo de Prim



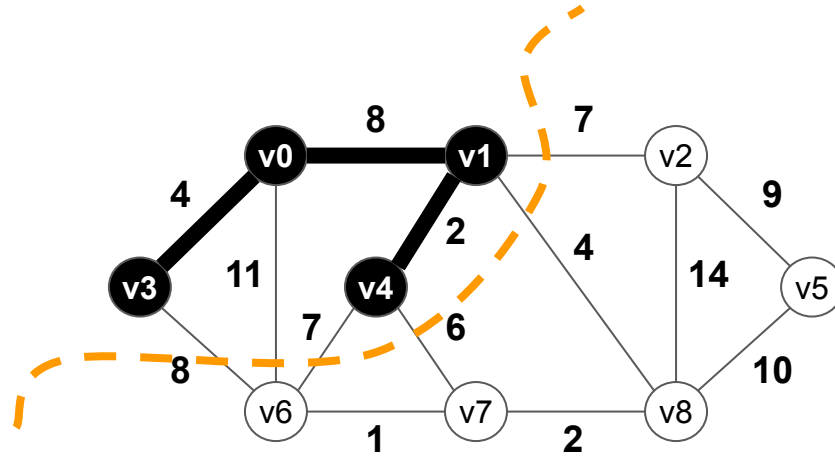
Algoritmo de Prim



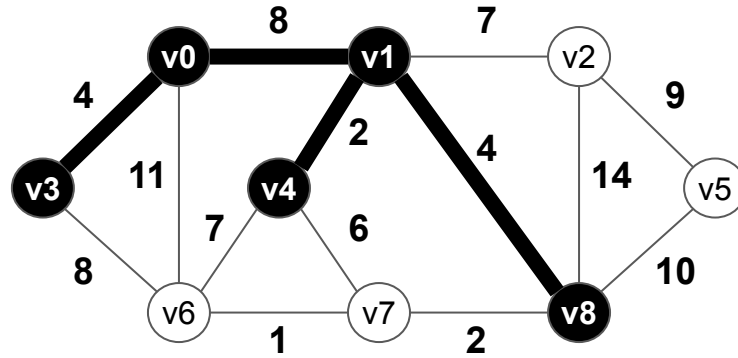
Algoritmo de Prim



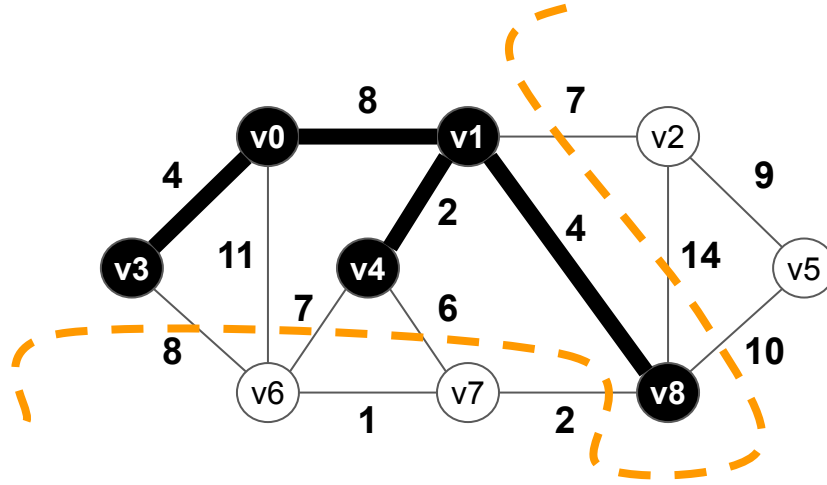
Algoritmo de Prim



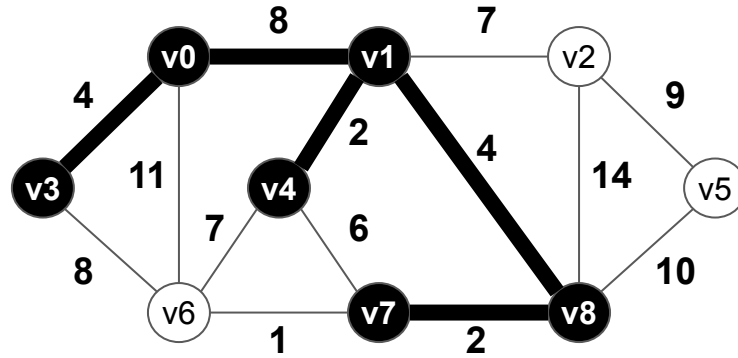
Algoritmo de Prim



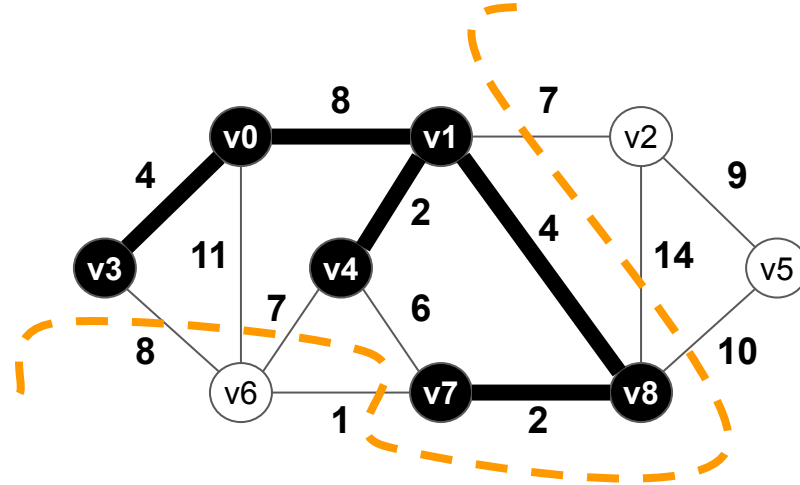
Algoritmo de Prim



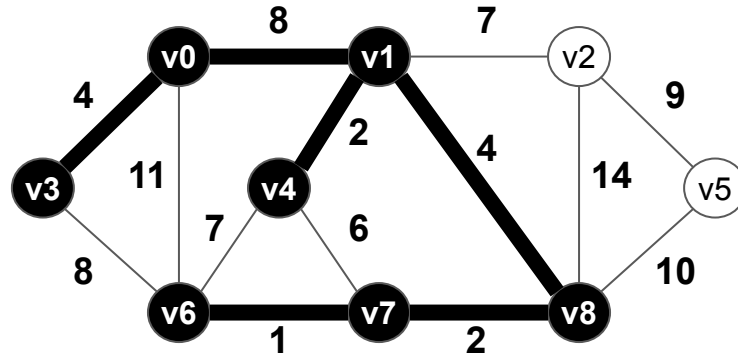
Algoritmo de Prim



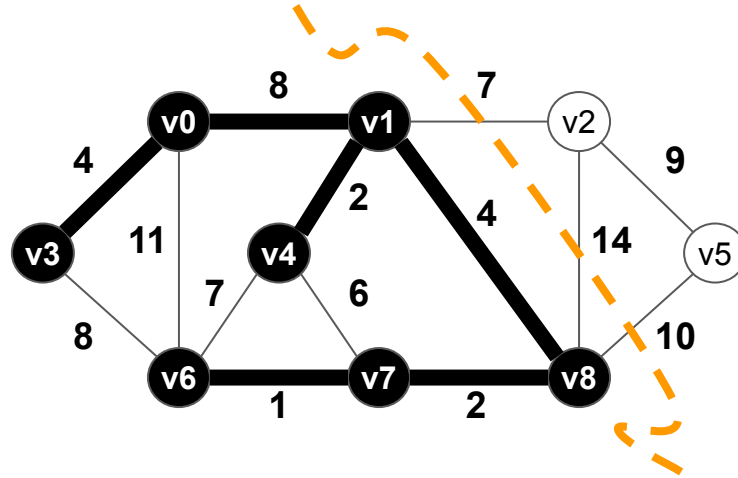
Algoritmo de Prim



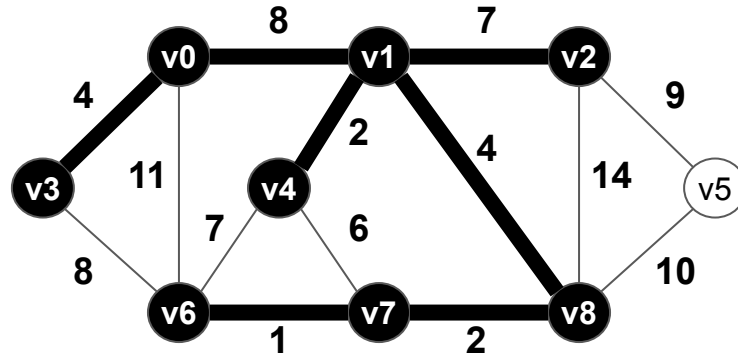
Algoritmo de Prim



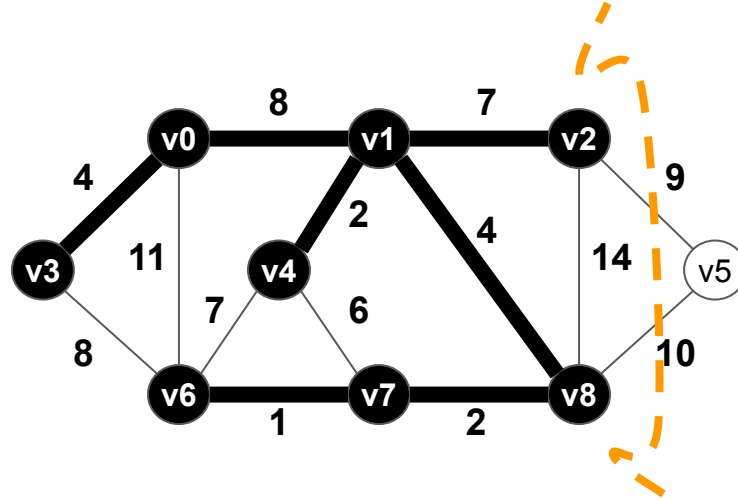
Algoritmo de Prim



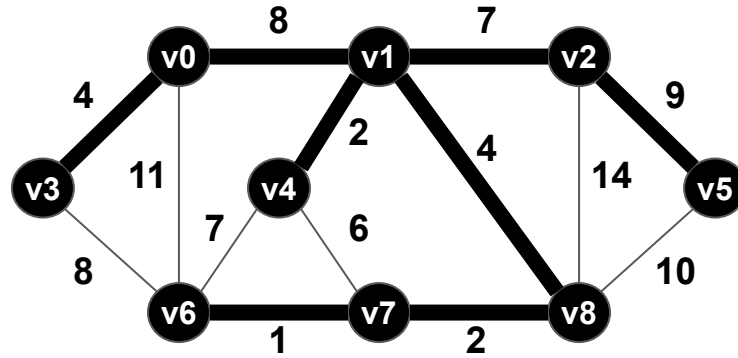
Algoritmo de Prim



Algoritmo de Prim



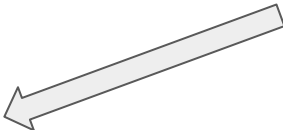
Algoritmo de Prim



Algoritmo de Prim

Prim(G conexo)

Inicialmente, T é uma árvore que consiste apenas no vértice 0 de G




1. $T = (\{0\}, \emptyset)$
2. Enquanto T não é uma árvore geradora de G :
3. Encontre uma aresta uv de peso mínimo do corte $(V(T), V(G) \setminus V(T))$
4. Adicione uv a T
5. Retorne T

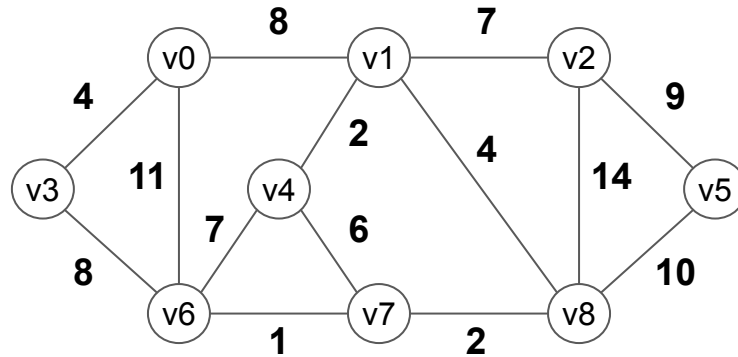
Algoritmo de Prim - Implementação 1 (ineficiente)

Prim(G conexo)

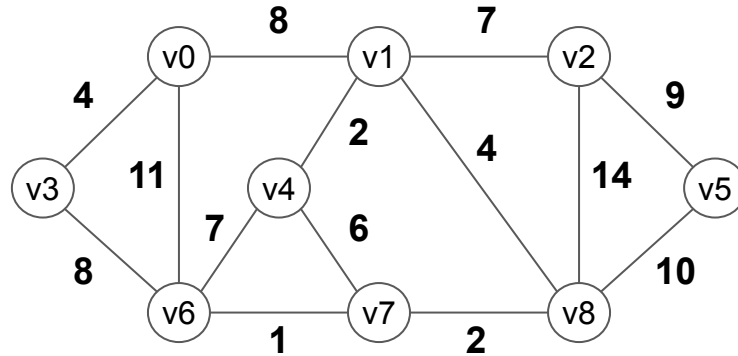
Inicialmente, T é uma árvore que consiste apenas no vértice 0 de G

1. $T = (\{0\}, \emptyset)$ 
2. Enquanto T não é uma árvore geradora de G :
3. Inicialize o peso de uv como infinito
4. Para cada aresta xy de G :
5. Se $x \in V(T)$ e $y \notin V(T)$ ou $y \in V(T)$ e $x \notin V(T)$:
6. Se o peso de xy é menor que o peso de uv :
7. $uv = xy$
8. Adicione uv a T
9. Retorne T

Algoritmo de Prim - Implementação 2



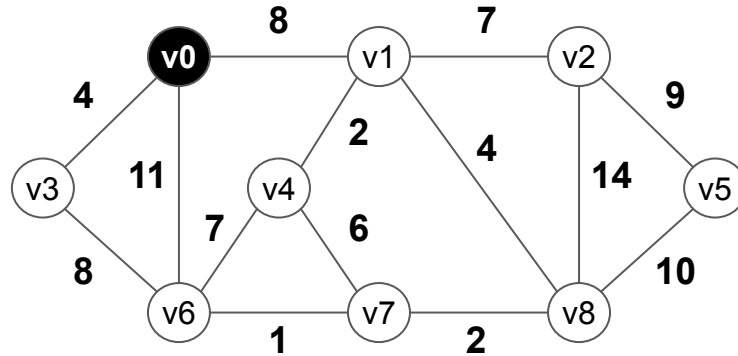
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1 ∞	-1 -1 ∞	-1 -1 ∞	-1 -1 ∞	-1 -1 ∞	-1 -1 ∞	-1 -1 ∞	-1 -1 ∞	-1 -1 ∞
0	1	2	3	4	5	6	7	8

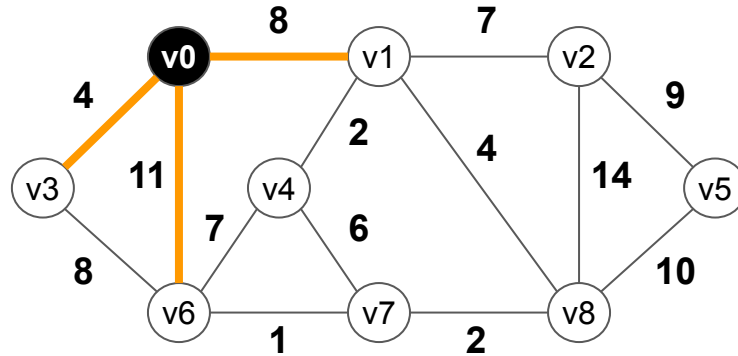
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1 ∞	-1 -1 ∞	-1 -1 ∞	-1 -1 ∞	-1 -1 ∞	-1 -1 ∞	-1 -1 ∞	-1 -1 ∞	-1 -1 ∞
0	1	2	3	4	5	6	7	8

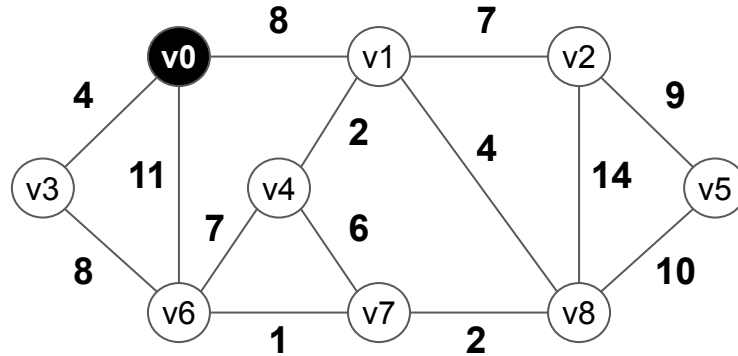
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	-1 -1	0 3	-1 -1	-1 -1	0 6	-1 -1	-1 -1
∞	8	∞	4	∞	∞	11	∞	∞
0	1	2	3	4	5	6	7	8

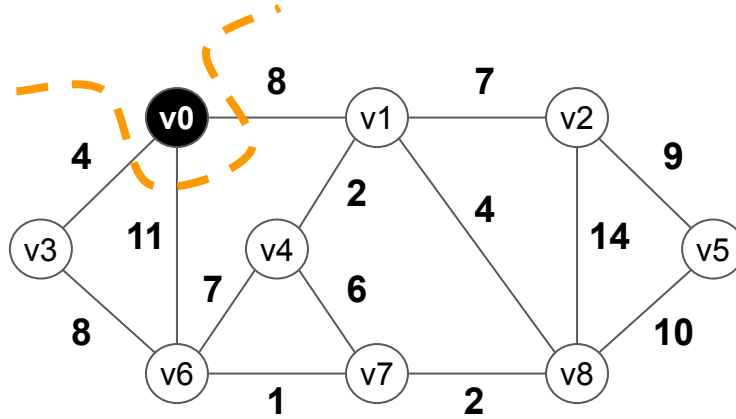
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1 ∞	0 1 8	-1 -1 ∞	0 3 4	-1 -1 ∞	-1 -1 ∞	0 6 11	-1 -1 ∞	-1 -1 ∞
0	1	2	3	4	5	6	7	8

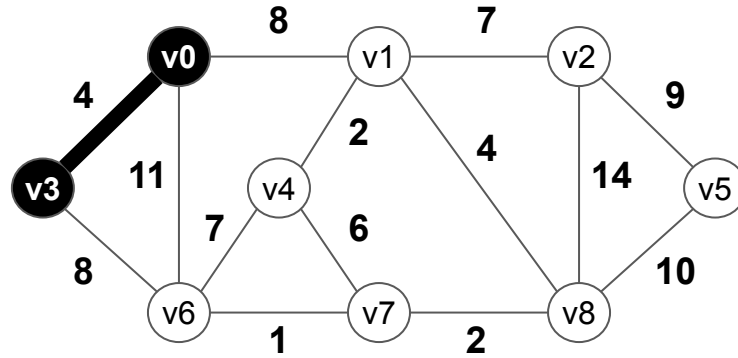
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	-1 -1	0 3	-1 -1	-1 -1	0 6	-1 -1	-1 -1
∞	8	∞	4	∞	∞	11	∞	∞
0	1	2	3	4	5	6	7	8

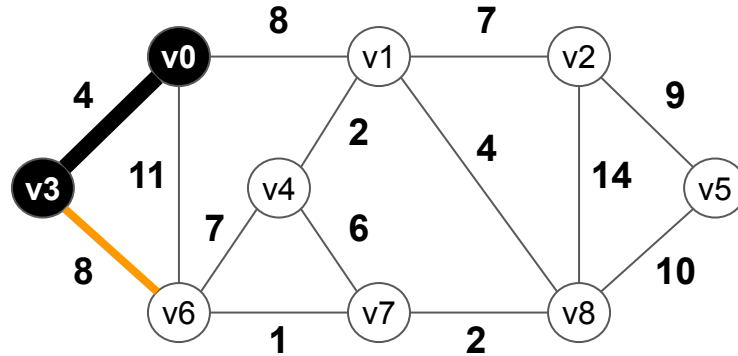
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1 ∞	0 1 8	-1 -1 ∞	0 3 4	-1 -1 ∞	-1 -1 ∞	0 6 11	-1 -1 ∞	-1 -1 ∞
0	1	2	3	4	5	6	7	8

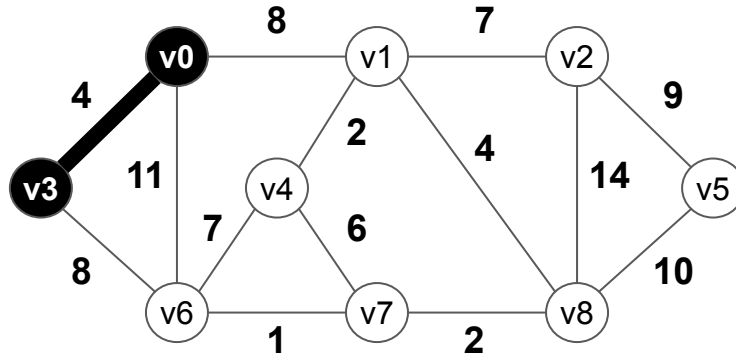
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	-1 -1	0 3	-1 -1	-1 -1	3 6	-1 -1	-1 -1
∞	8	∞	4	∞	∞	8	∞	∞
0	1	2	3	4	5	6	7	8

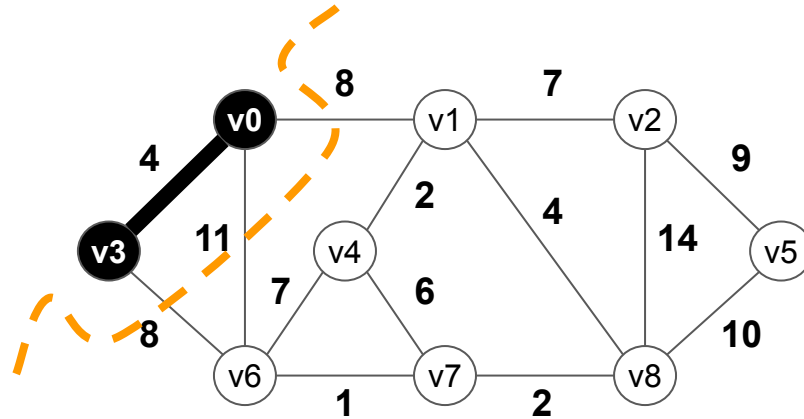
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1 ∞	0 1 8	-1 -1 ∞	0 3 4	-1 -1 ∞	-1 -1 ∞	3 6 8	-1 -1 ∞	-1 -1 ∞
0	1	2	3	4	5	6	7	8

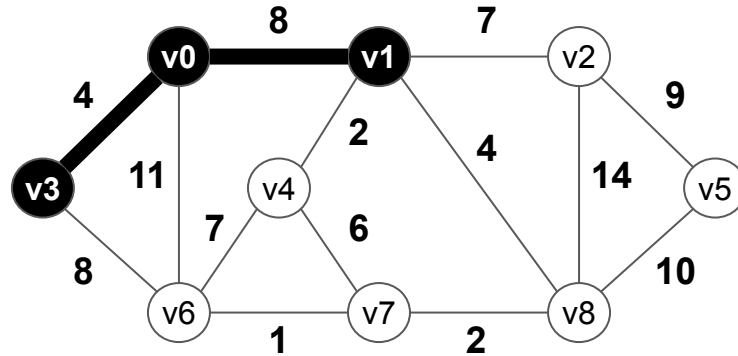
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	-1 -1	0 3	-1 -1	-1 -1	3 6	-1 -1	-1 -1
∞	8	∞	4	∞	∞	8	∞	∞
0	1	2	3	4	5	6	7	8

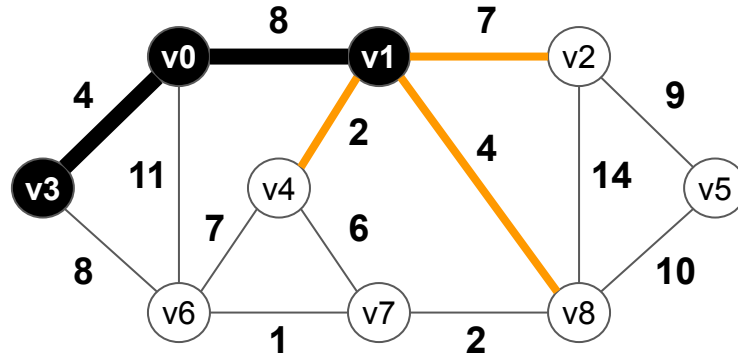
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1 ∞	0 1 8	-1 -1 ∞	0 3 4	-1 -1 ∞	-1 -1 ∞	3 6 8	-1 -1 ∞	-1 -1 ∞
0	1	2	3	4	5	6	7	8

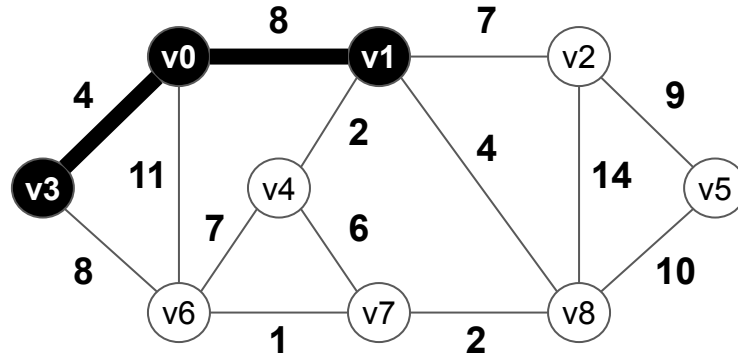
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	-1 -1	3 6	-1 -1	1 8
∞	8	7	4	2	∞	8	∞	4
0	1	2	3	4	5	6	7	8

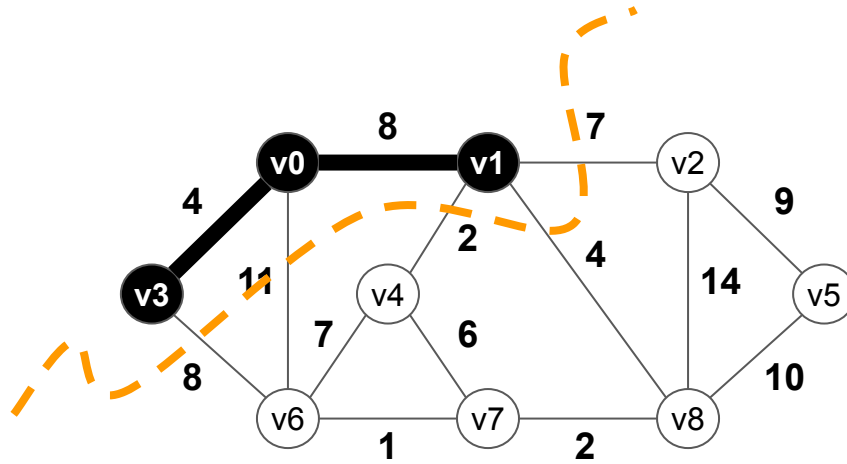
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	-1 -1	3 6	-1 -1	1 8
∞	8	7	4	2	∞	8	∞	4
0	1	2	3	4	5	6	7	8

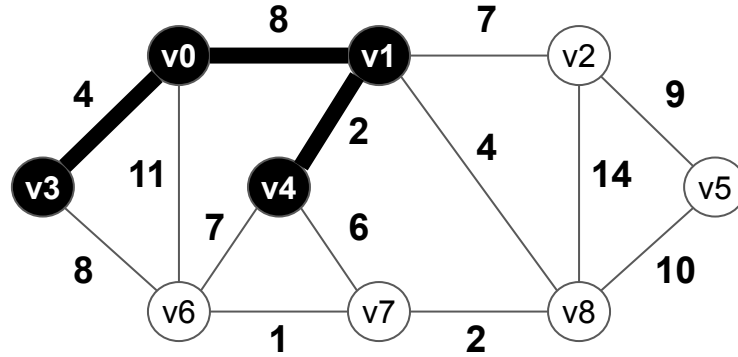
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	-1 -1	3 6	-1 -1	1 8
∞	8	7	4	2	∞	8	∞	4
0	1	2	3	4	5	6	7	8

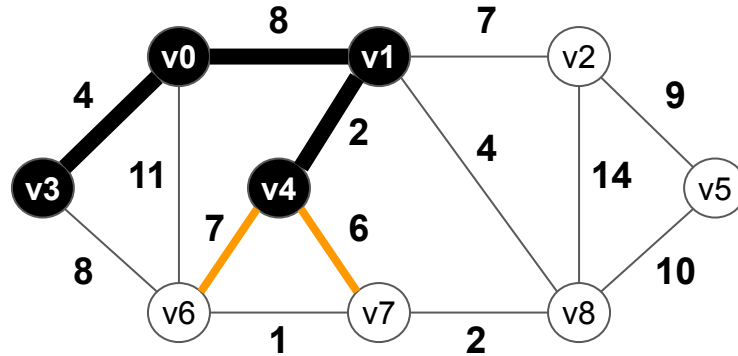
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	-1 -1	3 6	-1 -1	1 8
∞	8	7	4	2	∞	8	∞	4
0	1	2	3	4	5	6	7	8

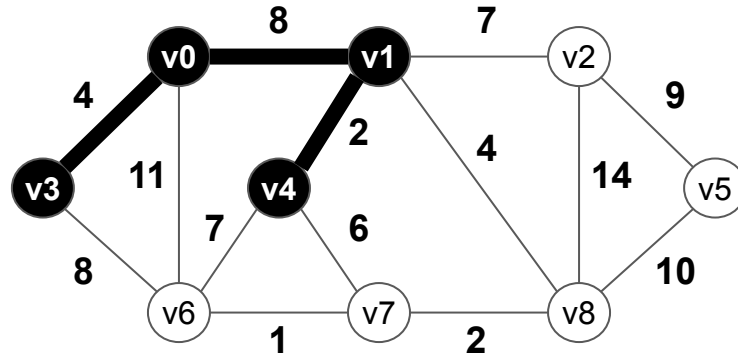
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	-1 -1	4 6	4 7	1 8
∞	8	7	4	2	∞	7	6	4
0	1	2	3	4	5	6	7	8

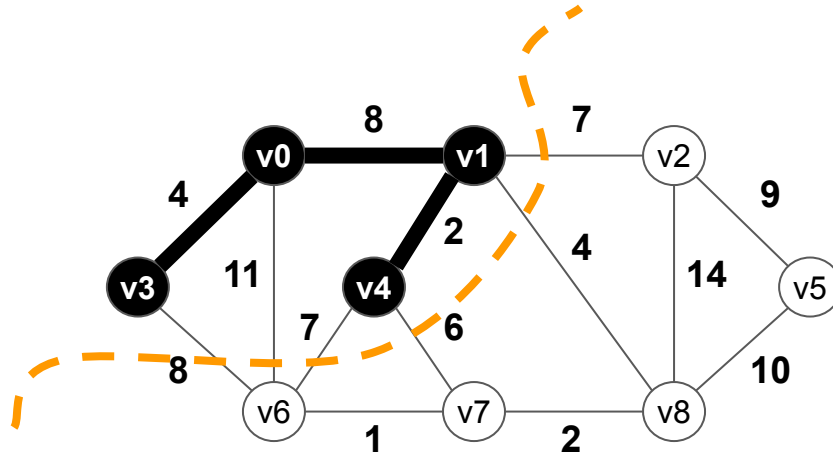
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	-1 -1	4 6	4 7	1 8
∞	8	7	4	2	∞	7	6	4
0	1	2	3	4	5	6	7	8

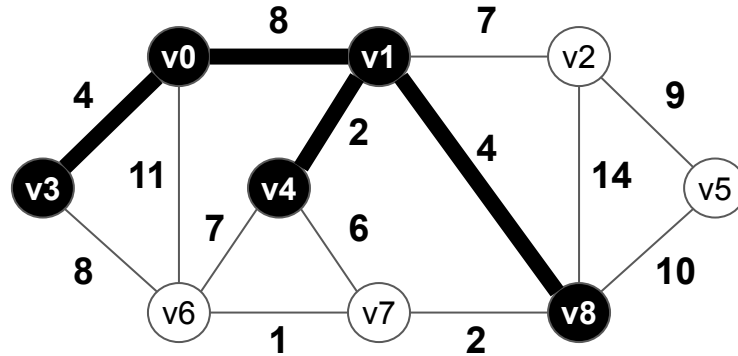
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	-1 -1	4 6	4 7	1 8
∞	8	7	4	2	∞	7	6	4
0	1	2	3	4	5	6	7	8

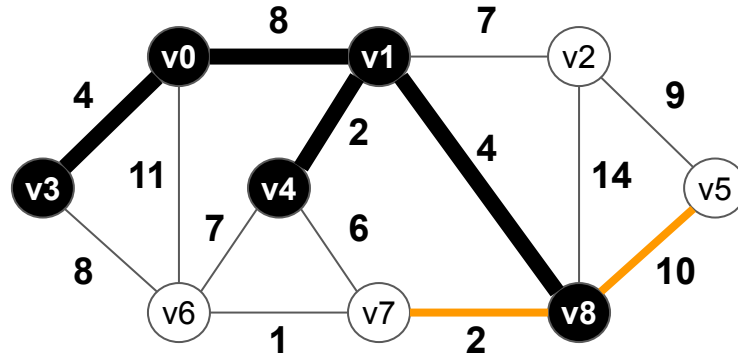
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	-1 -1	4 6	4 7	1 8
∞	8	7	4	2	∞	7	6	4
0	1	2	3	4	5	6	7	8

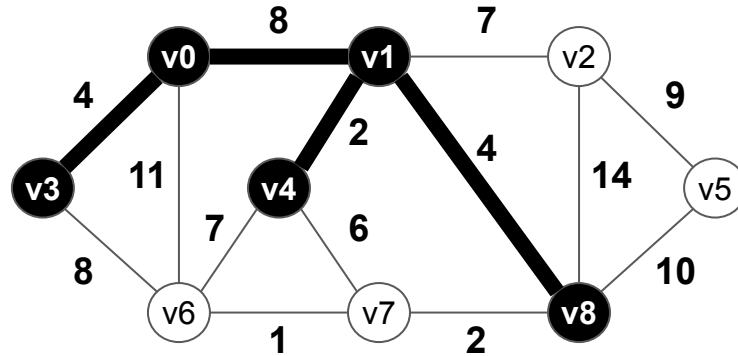
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	8 5	4 6	8 7	1 8
∞	8	7	4	2	10	7	2	4
0	1	2	3	4	5	6	7	8

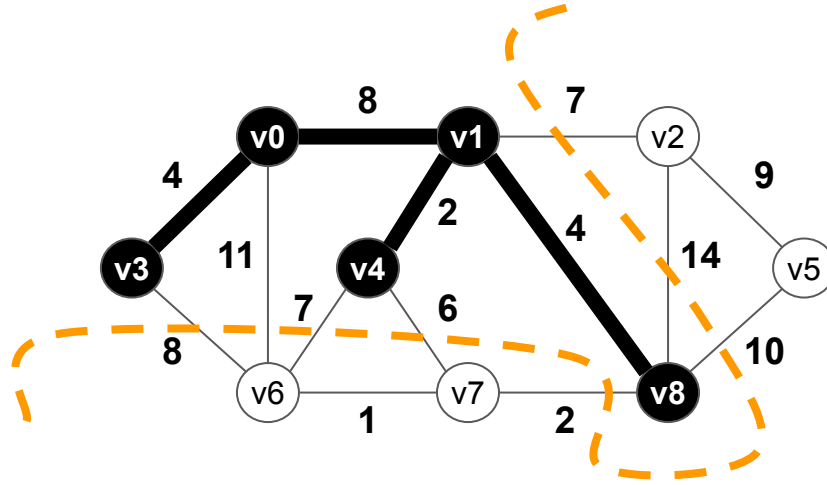
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	8 5	4 6	8 7	1 8
∞	8	7	4	2	10	7	2	4
0	1	2	3	4	5	6	7	8

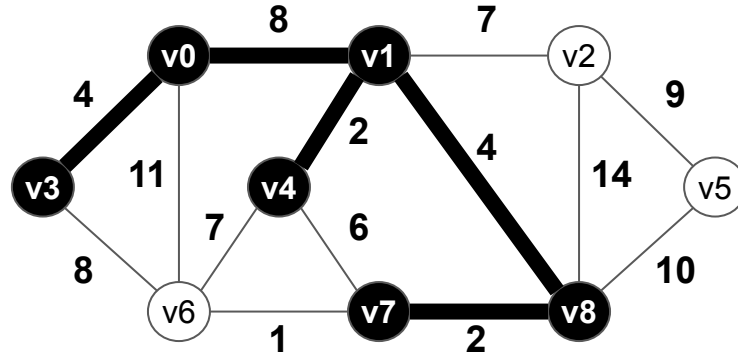
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	8 5	4 6	8 7	1 8
∞	8	7	4	2	10	7	2	4
0	1	2	3	4	5	6	7	8

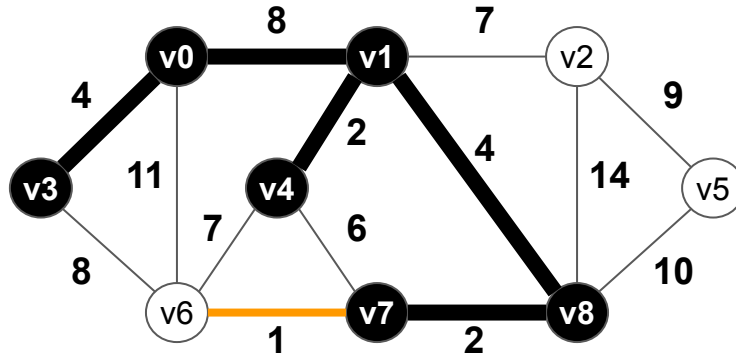
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	8 5	4 6	8 7	1 8
∞	8	7	4	2	10	7	2	4
0	1	2	3	4	5	6	7	8

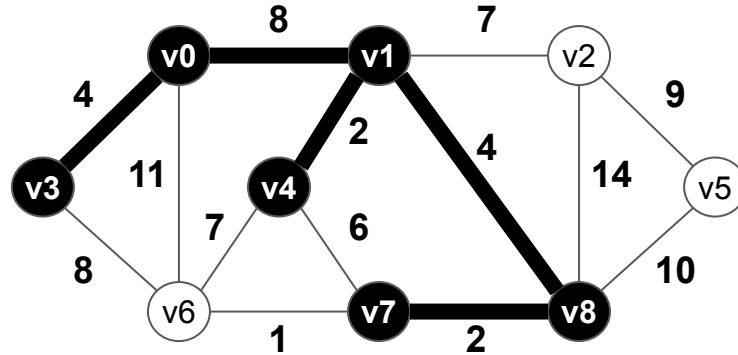
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	8 5	7 6	8 7	1 8
∞	8	7	4	2	10	1	2	4
0	1	2	3	4	5	6	7	8

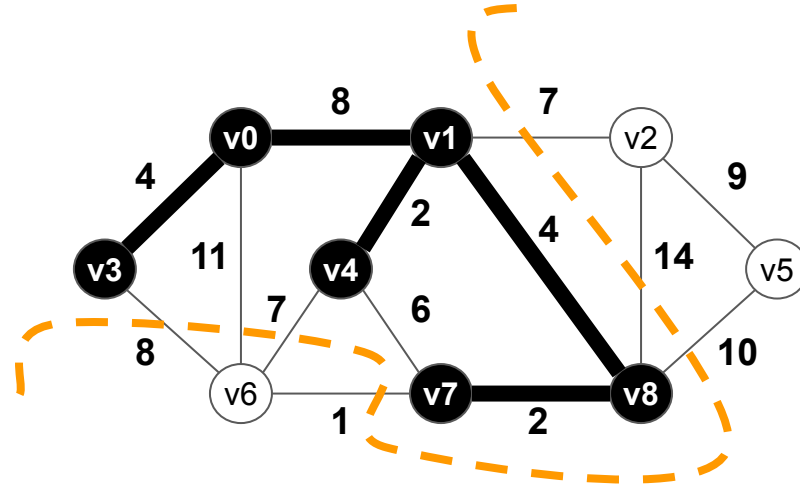
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	8 5	7 6	8 7	1 8
∞	8	7	4	2	10	1	2	4
0	1	2	3	4	5	6	7	8

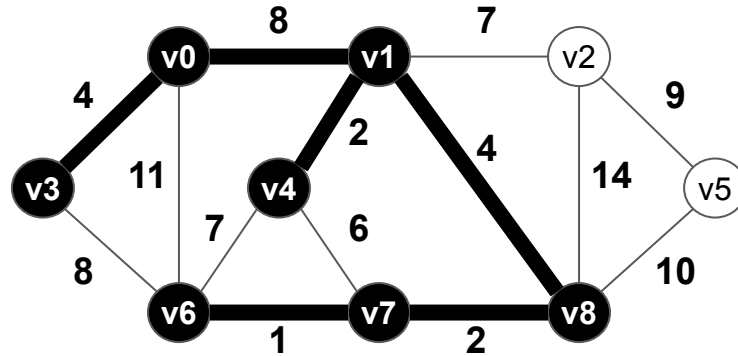
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	8 5	7 6	8 7	1 8
∞	8	7	4	2	10	1	2	4
0	1	2	3	4	5	6	7	8

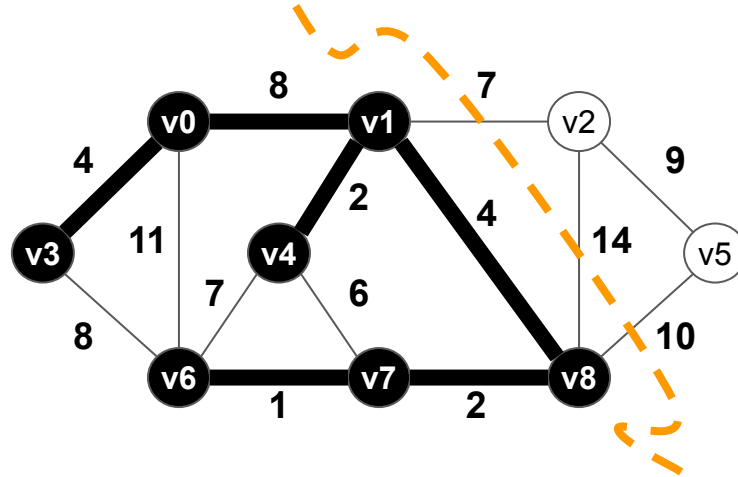
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	8 5	7 6	8 7	1 8
∞	8	7	4	2	10	1	2	4
0	1	2	3	4	5	6	7	8

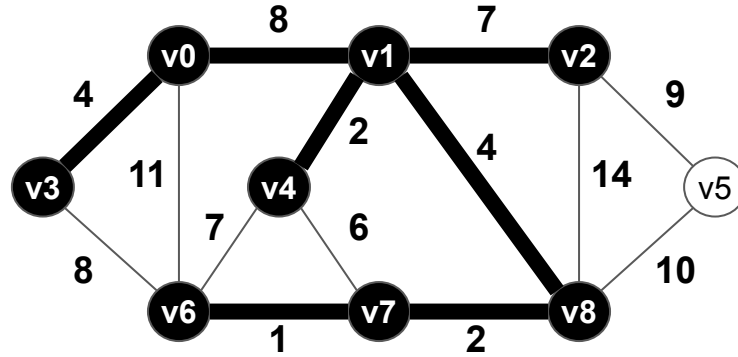
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	8 5	7 6	8 7	1 8
∞	8	7	4	2	10	1	2	4
0	1	2	3	4	5	6	7	8

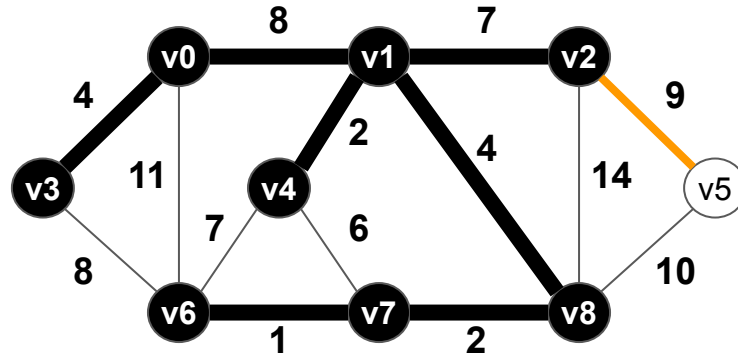
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	8 5	7 6	8 7	1 8
∞	8	7	4	2	10	1	2	4
0	1	2	3	4	5	6	7	8

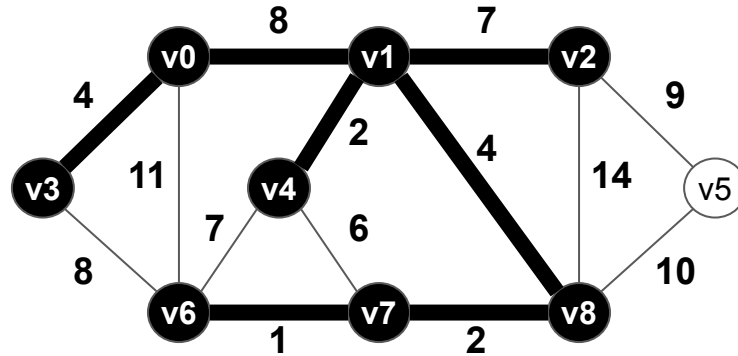
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	2 5	7 6	8 7	1 8
∞	8	7	4	2	9	1	2	4
0	1	2	3	4	5	6	7	8

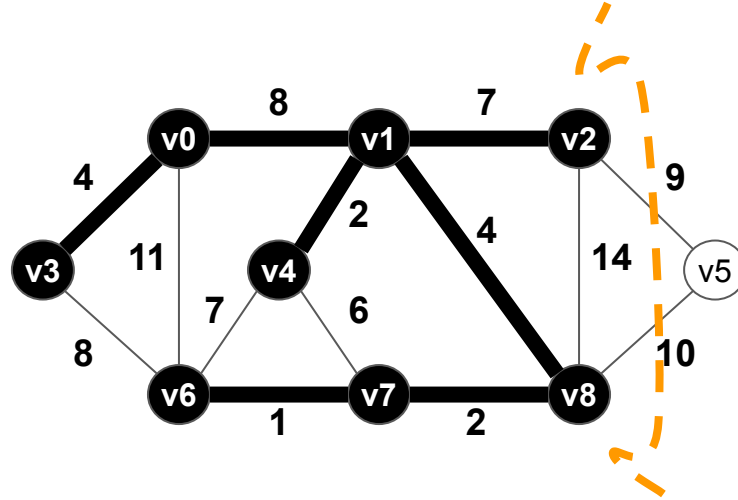
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	2 5	7 6	8 7	1 8
∞	8	7	4	2	9	1	2	4
0	1	2	3	4	5	6	7	8

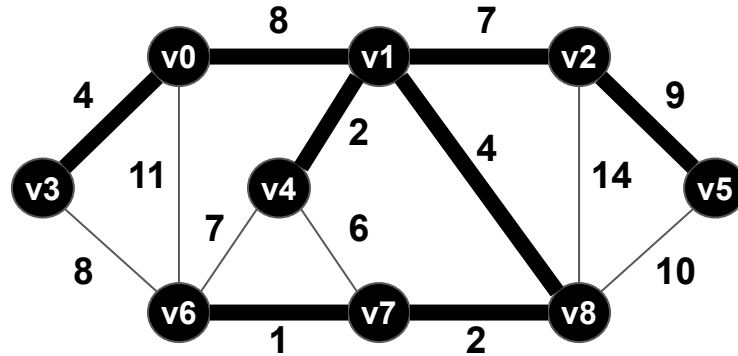
Algoritmo de Prim - Implementação 2



aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	2 5	7 6	8 7	1 8
∞	8	7	4	2	9	1	2	4
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 2




aresta_peso_min_para_arvore:

-1 -1	0 1	1 2	0 3	1 4	2 5	7 6	8 7	1 8
∞	8	7	4	2	9	1	2	4
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 2 (é possível melhorar)

Prim(G conexo)

Inicialmente, T é uma árvore que consiste apenas no vértice 0 de G

1. $T = (\{0\}, \emptyset)$ 
2. Enquanto T não é uma árvore geradora de G :
3. Encontre uma aresta uv de peso mínimo do corte $(V(T), V(G) \setminus V(T))$

Implementar o Passo 3 percorrendo um vetor onde é possível acessar, para cada vértice v que não está em T , uma aresta de peso mínimo entre v e um vértice de T

4. Adicione uv a T
5. Retorne T

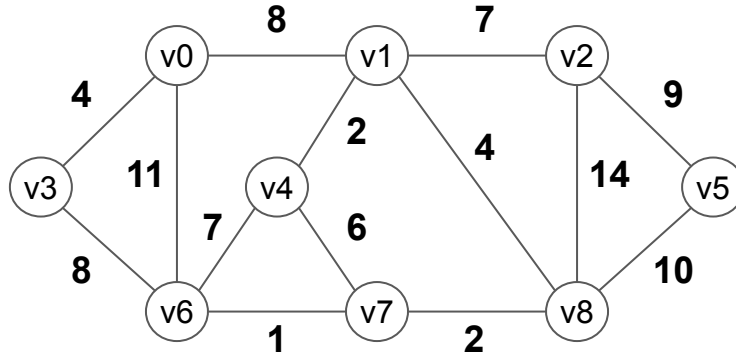
Algoritmo de Prim - Implementação 3

- Nesta implementação, vamos usar uma fila de prioridade (heap) para implementar de maneira eficiente o passo de determinar a próxima aresta a ser adicionada à árvore que estamos construindo
- Além disso, vamos representar a árvore construída da mesma forma que fizemos em algoritmos vistos anteriormente: através de um vetor *pai*

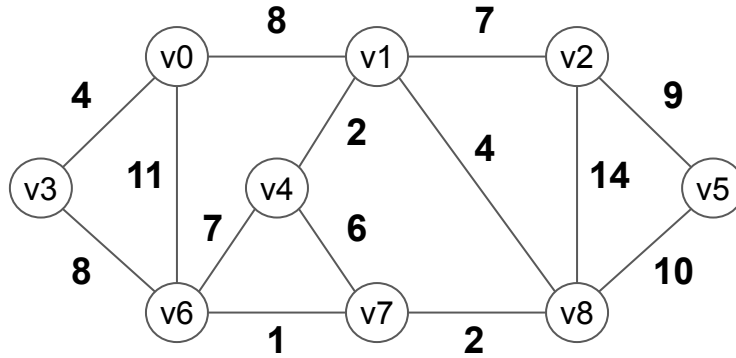
Algoritmo de Prim - Implementação 3

- Seja T a árvore que está sendo construída no algoritmo
- O vetor pai vai representar o seguinte:
 - Para um vértice w que ainda não esteja em T ,
 - $pai[w]$ contém v tal que vw é uma aresta de peso mínimo entre w e um vértice de T
 - Para um vértice w que já esteja em T ,
 - $pai[w]$ contém o pai de w em T

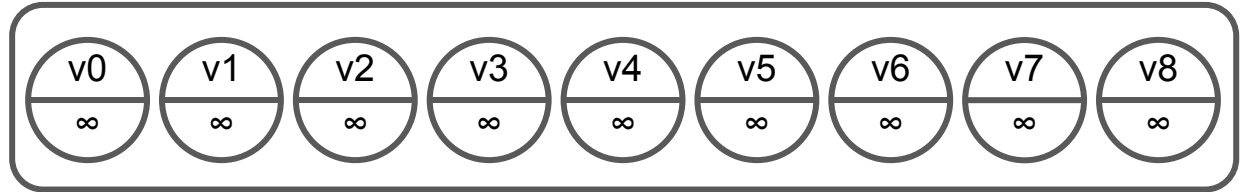
Algoritmo de Prim - Implementação 3



Algoritmo de Prim - Implementação 3



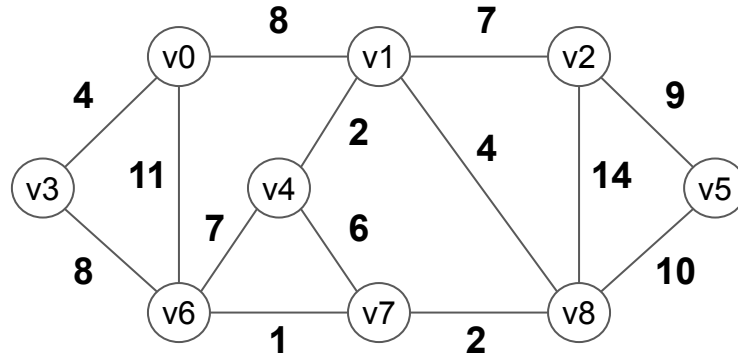
Fila de prioridade:



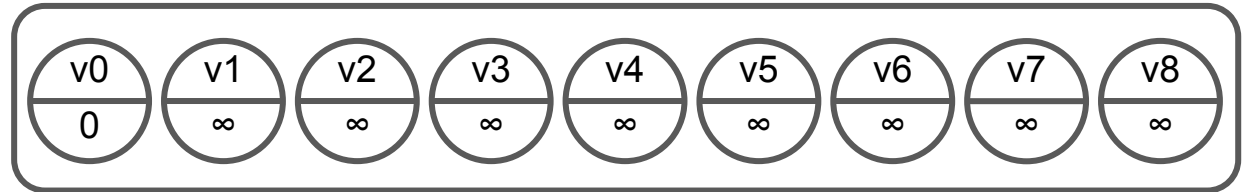
pai:

-1	-1	-1	-1	-1	-1	-1	-1	-1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



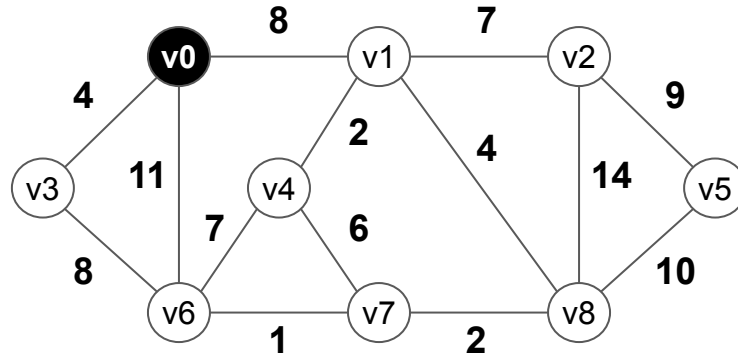
Fila de prioridade:



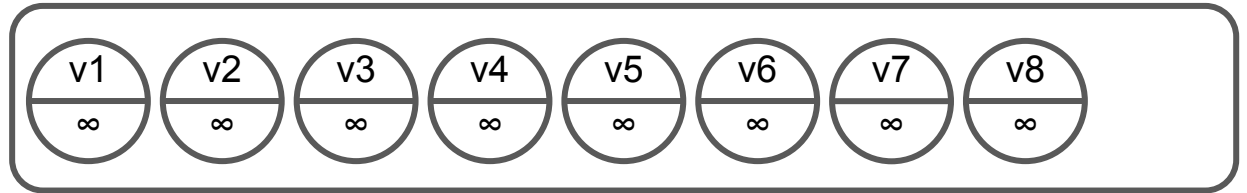
pai:

-1	-1	-1	-1	-1	-1	-1	-1	-1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



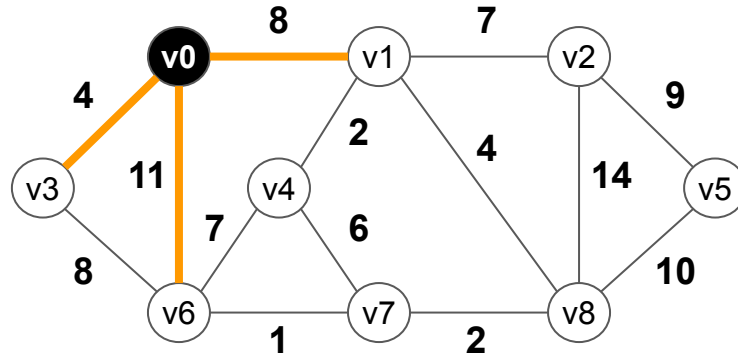
Fila de prioridade:



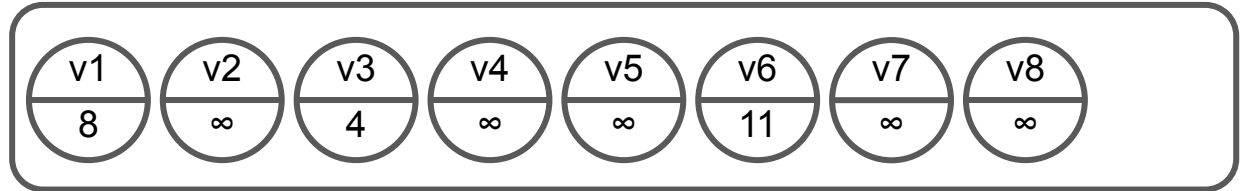
pai:

-1	-1	-1	-1	-1	-1	-1	-1	-1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



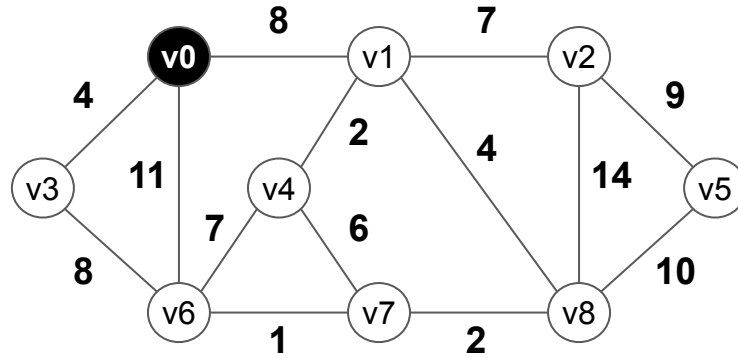
Fila de prioridade:



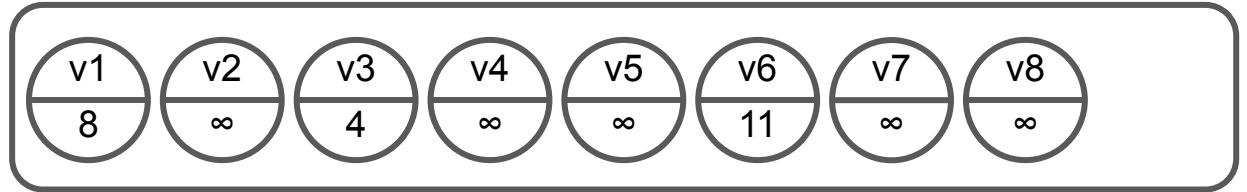
pai:

-1	0	-1	0	-1	-1	0	-1	-1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



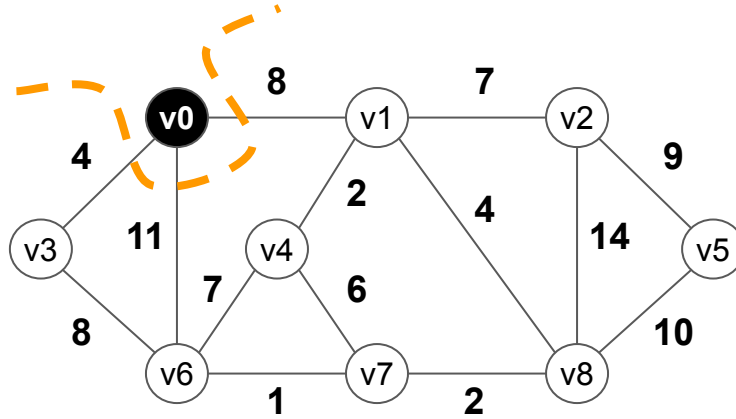
Fila de prioridade:



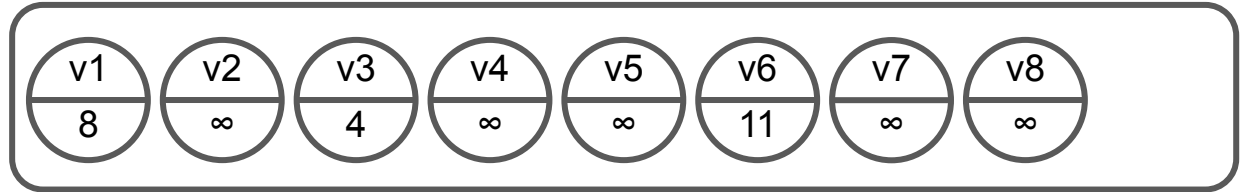
pai:

-1	0	-1	0	-1	-1	0	-1	-1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



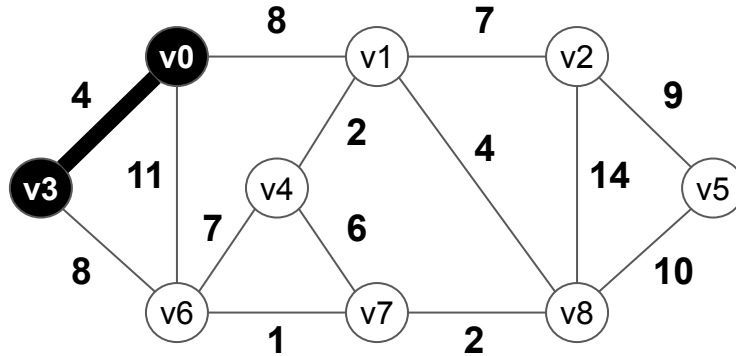
Fila de prioridade:



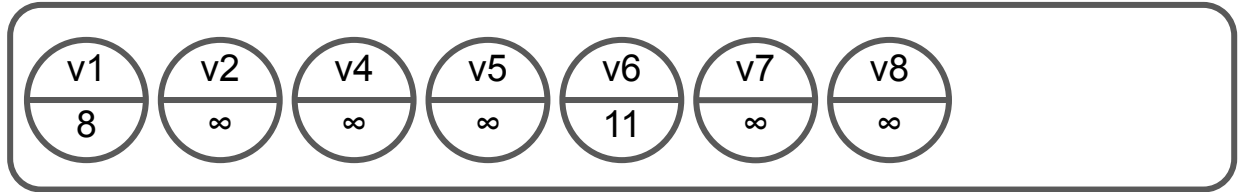
pai:

-1	0	-1	0	-1	-1	0	-1	-1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



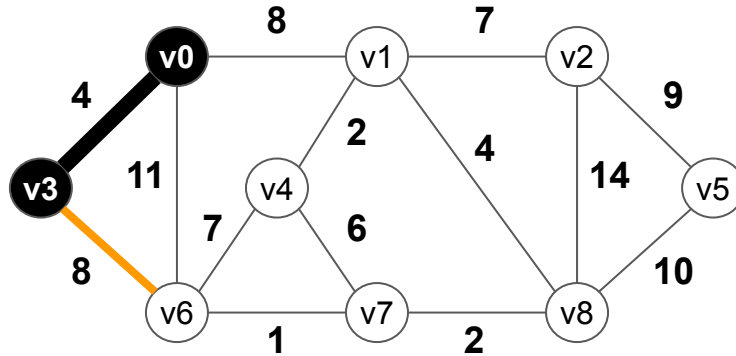
Fila de prioridade:



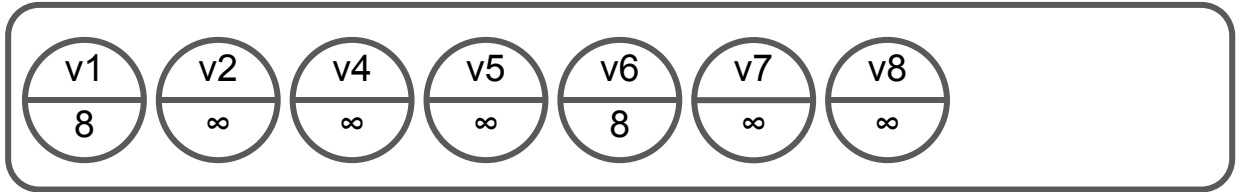
pai:

-1	0	-1	0	-1	-1	0	-1	-1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



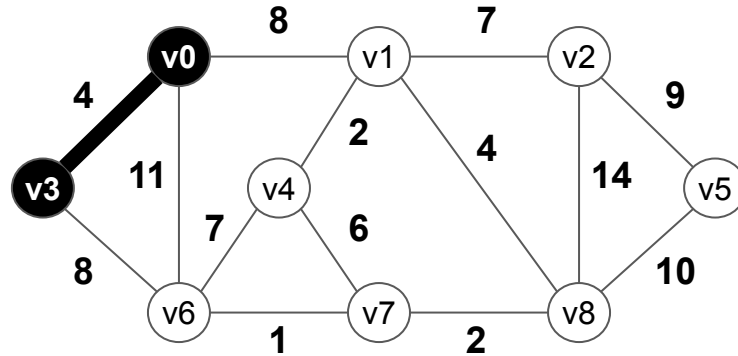
Fila de prioridade:



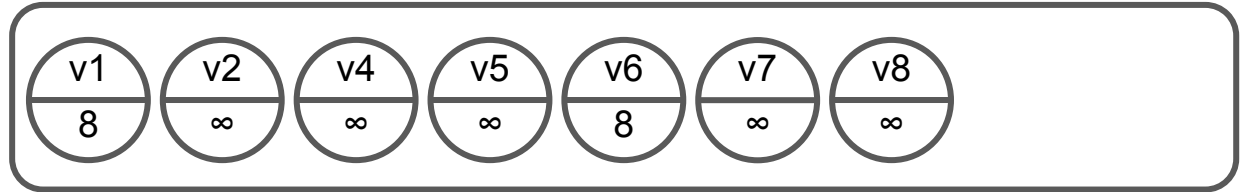
pai:

-1	0	-1	0	-1	-1	3	-1	-1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



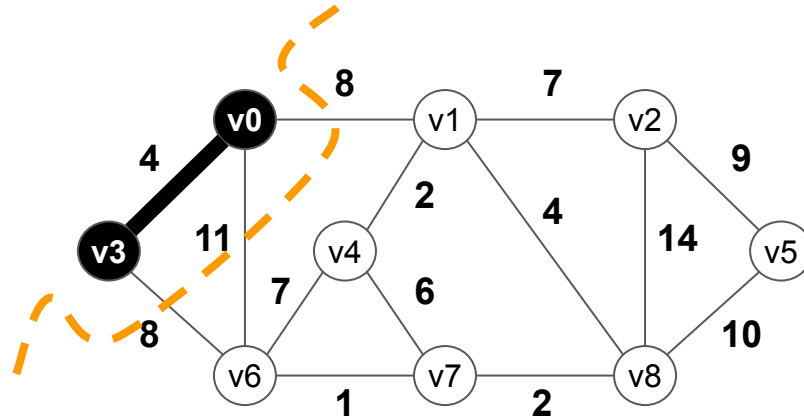
Fila de prioridade:



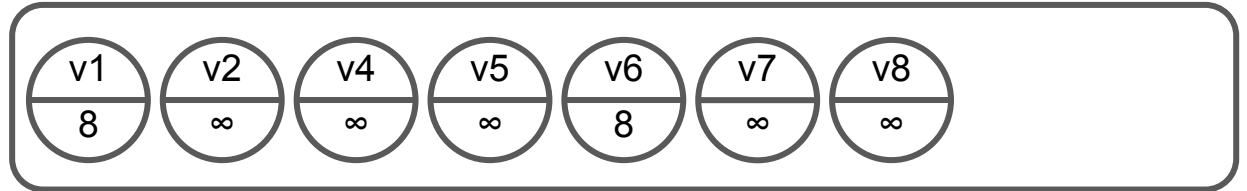
pai:

-1	0	-1	0	-1	-1	3	-1	-1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



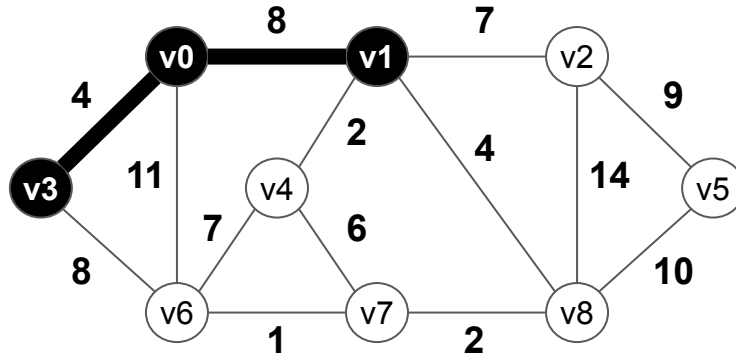
Fila de prioridade:



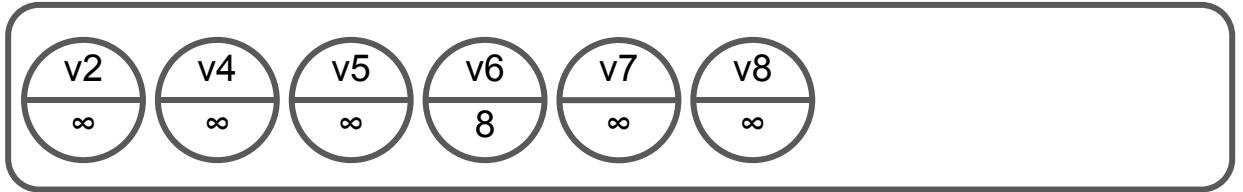
pai:

-1	0	-1	0	-1	-1	3	-1	-1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



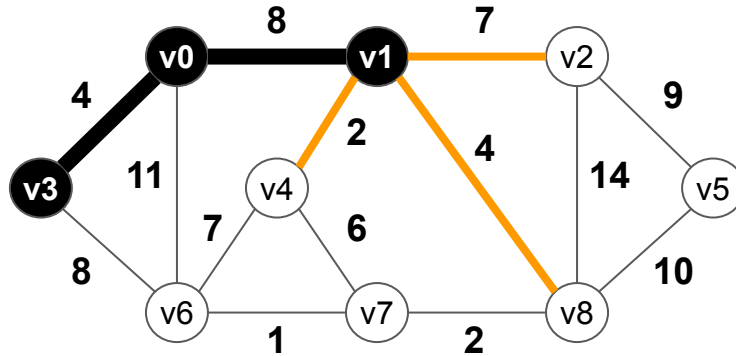
Fila de prioridade:



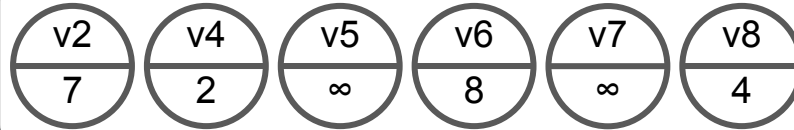
pai:

-1	0	-1	0	-1	-1	3	-1	-1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



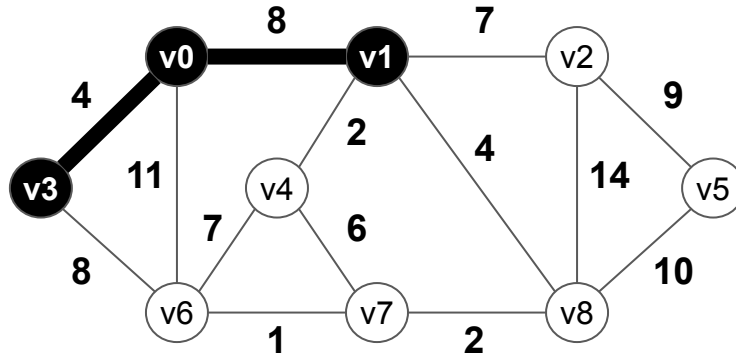
Fila de prioridade:



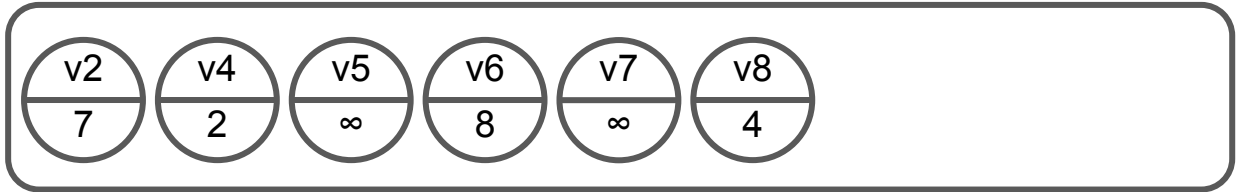
pai:

-1	0	1	0	1	-1	3	-1	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



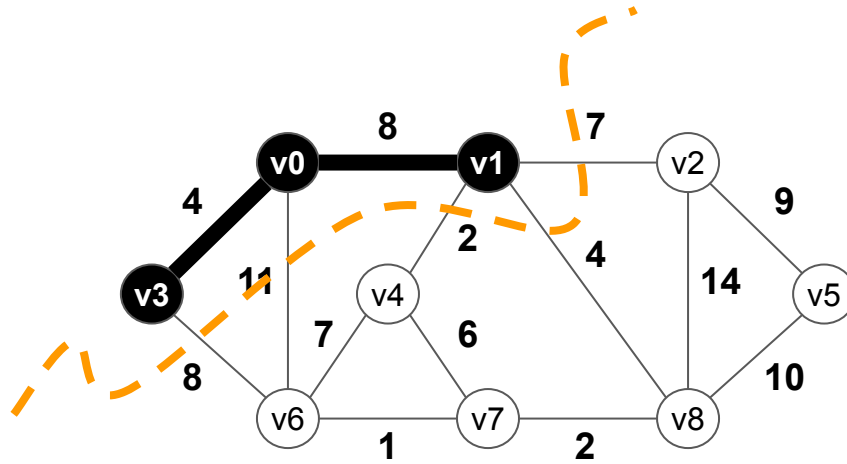
Fila de prioridade:



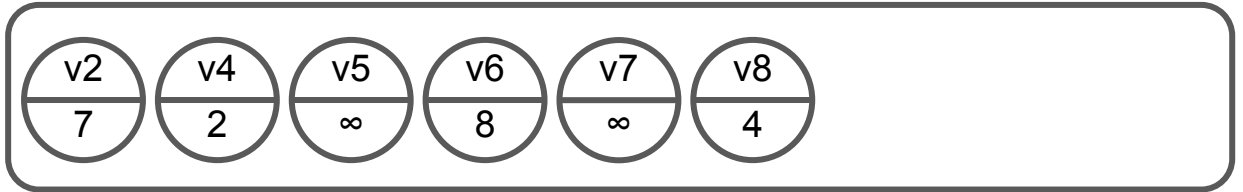
pai:

-1	0	1	0	1	-1	3	-1	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



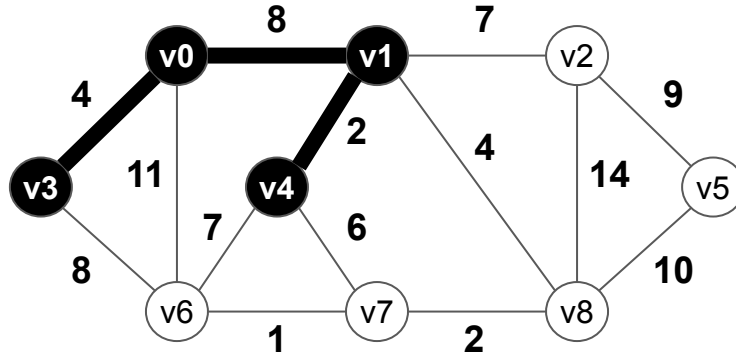
Fila de prioridade:



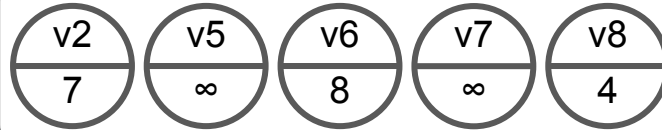
pai:

-1	0	1	0	1	-1	3	-1	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



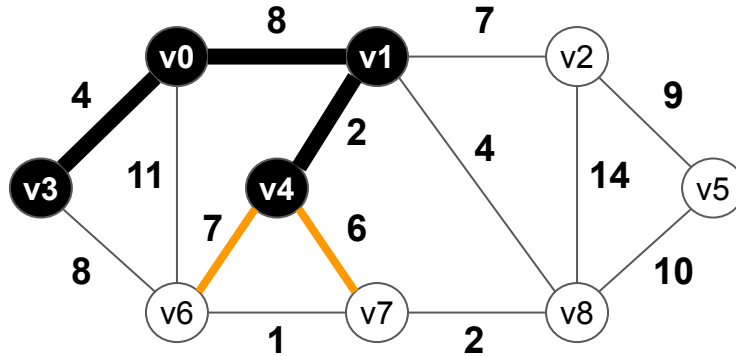
Fila de prioridade:



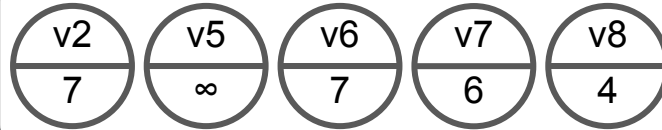
pai:

-1	0	1	0	1	-1	3	-1	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



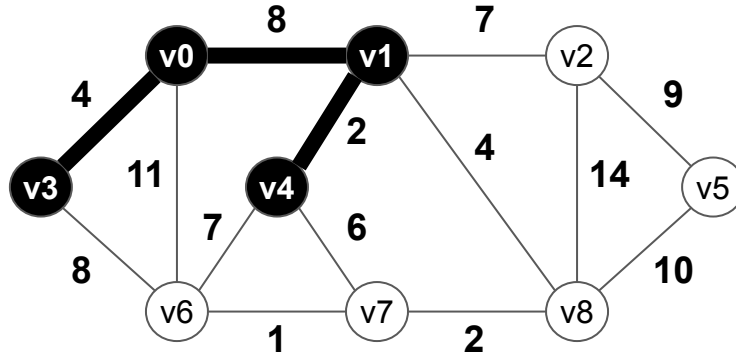
Fila de prioridade:



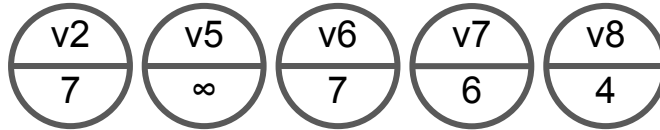
pai:

-1	0	1	0	1	-1	4	4	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



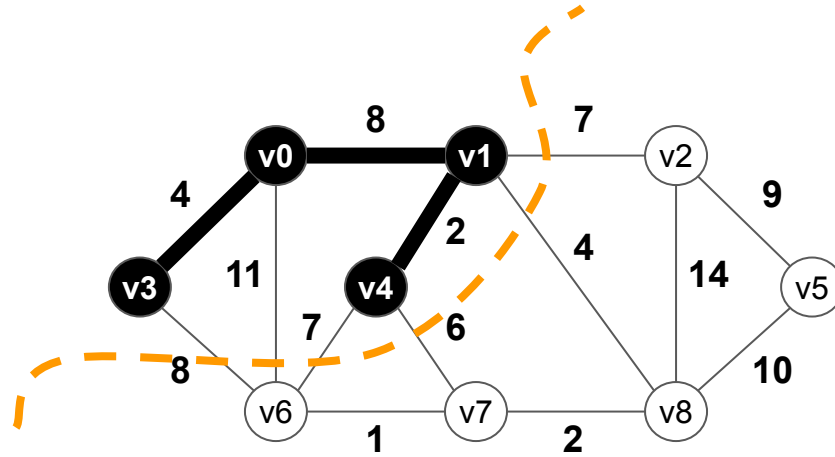
Fila de prioridade:



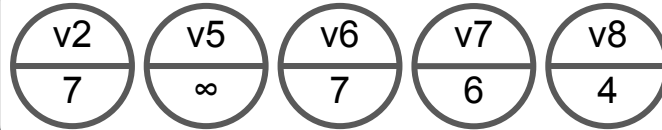
pai:

-1	0	1	0	1	-1	4	4	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



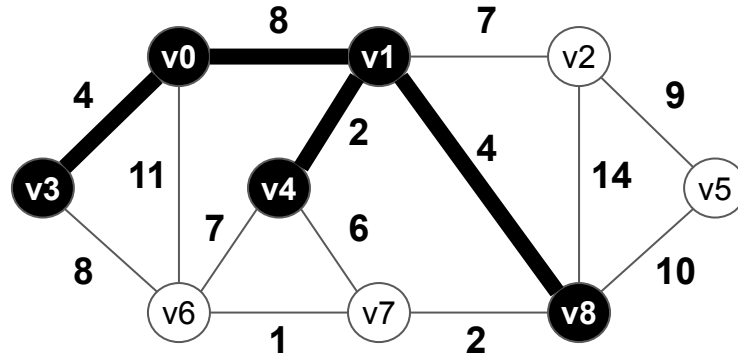
Fila de prioridade:



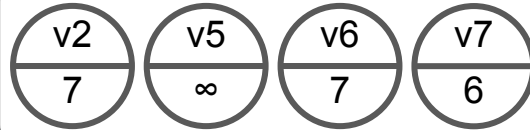
pai:

-1	0	1	0	1	-1	4	4	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



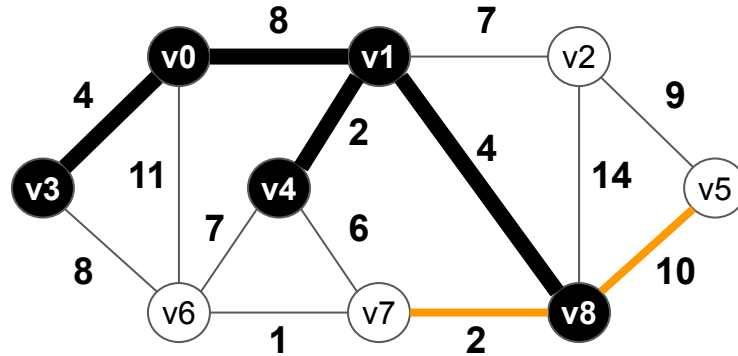
Fila de prioridade:



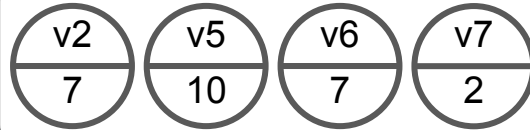
pai:

-1	0	1	0	1	-1	4	4	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



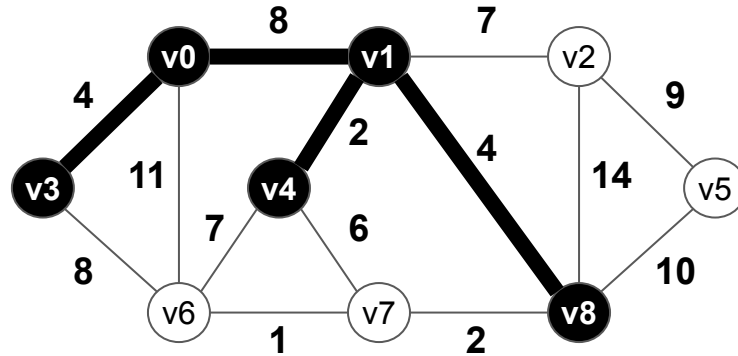
Fila de prioridade:



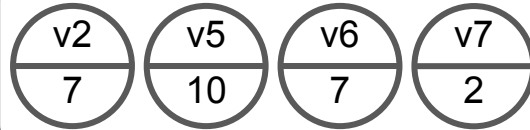
pai:

-1	0	1	0	1	8	4	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



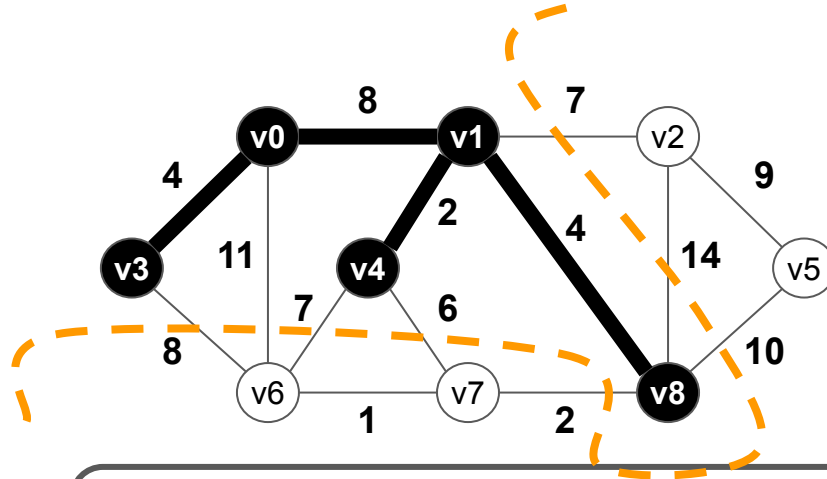
Fila de prioridade:



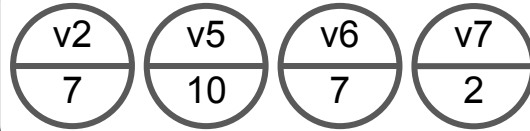
pai:

-1	0	1	0	1	8	4	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



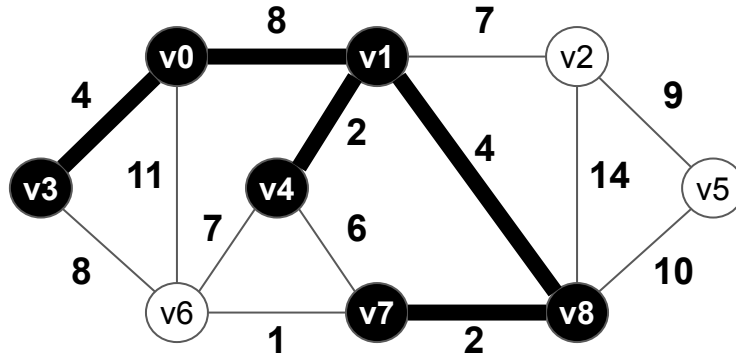
Fila de prioridade:



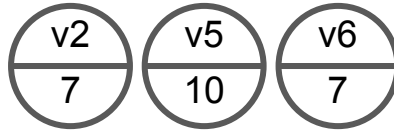
pai:

-1	0	1	0	1	8	4	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



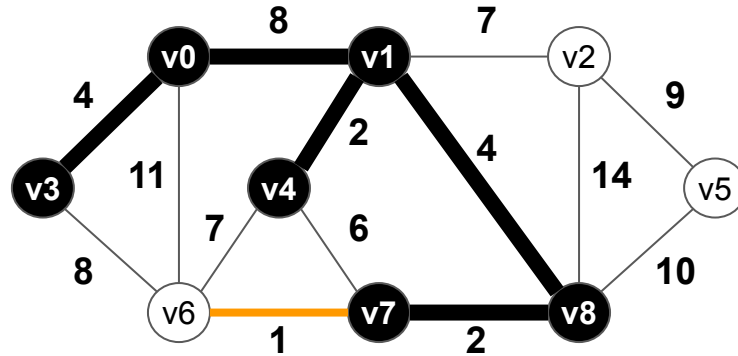
Fila de prioridade:



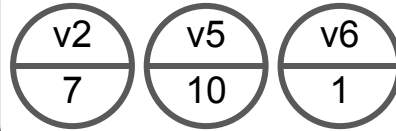
pai:

-1	0	1	0	1	8	4	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



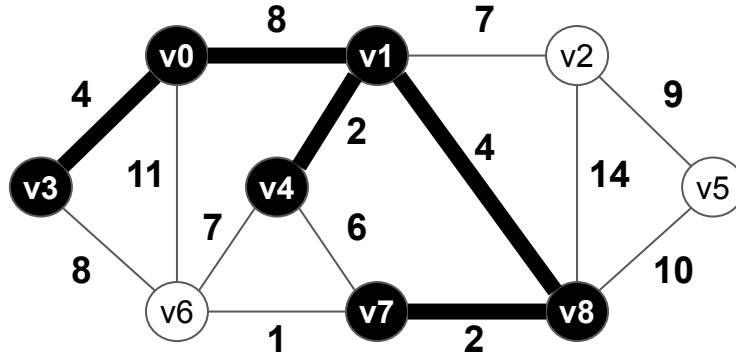
Fila de prioridade:



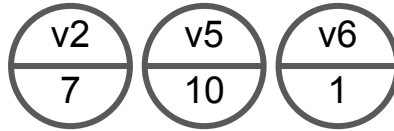
pai:

-1	0	1	0	1	8	7	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



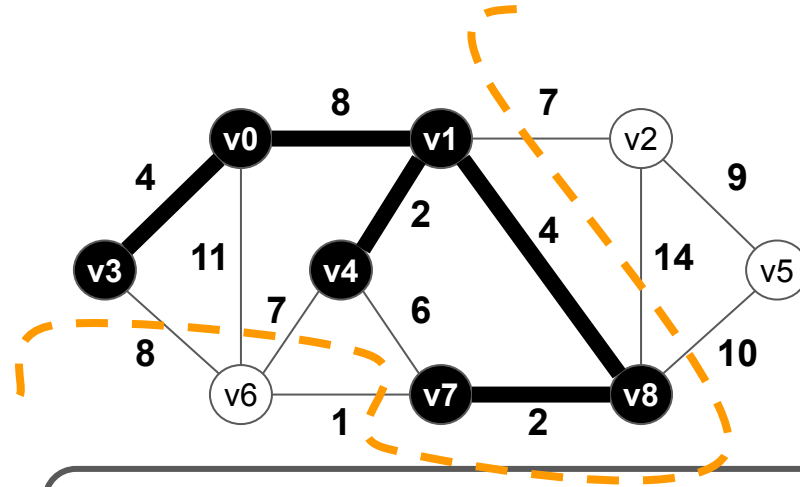
Fila de prioridade:



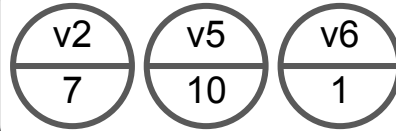
pai:

-1	0	1	0	1	8	7	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



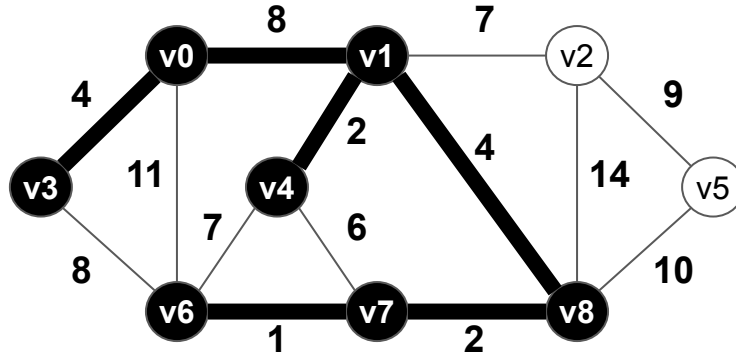
Fila de prioridade:



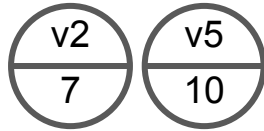
pai:

-1	0	1	0	1	8	7	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



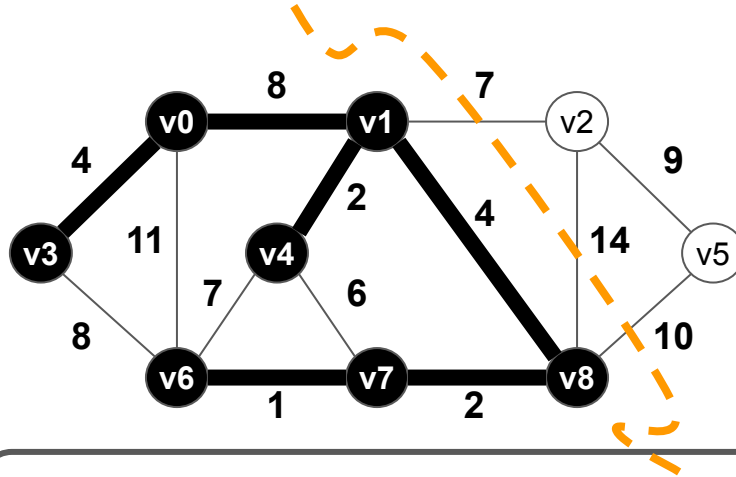
Fila de prioridade:



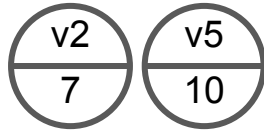
pai:

-1	0	1	0	1	8	7	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



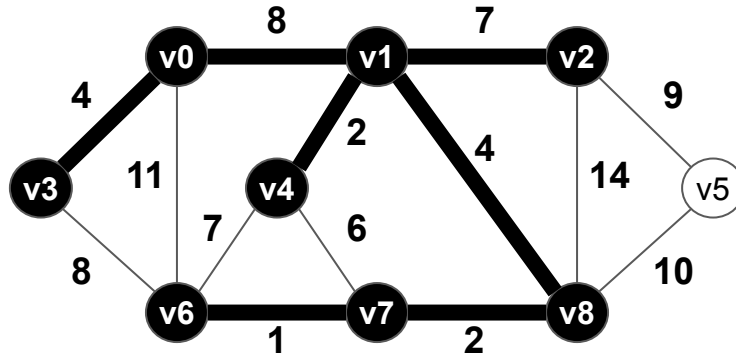
Fila de prioridade:



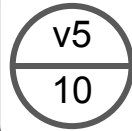
pai:

-1	0	1	0	1	8	7	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



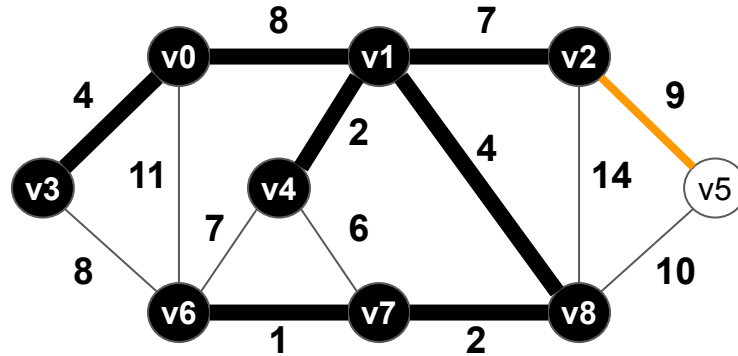
Fila de prioridade:



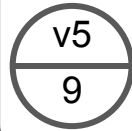
pai:

-1	0	1	0	1	8	7	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



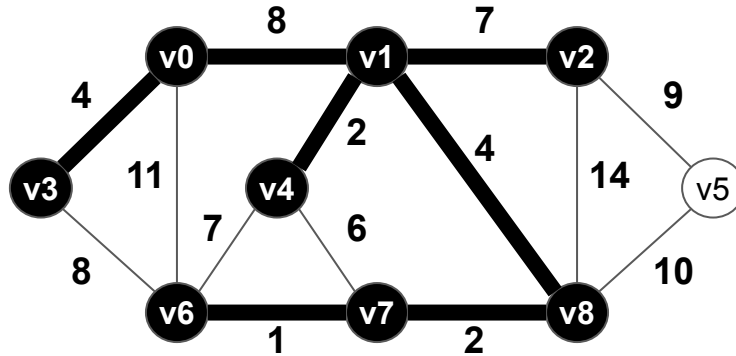
Fila de prioridade:



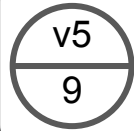
pai:

-1	0	1	0	1	2	7	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



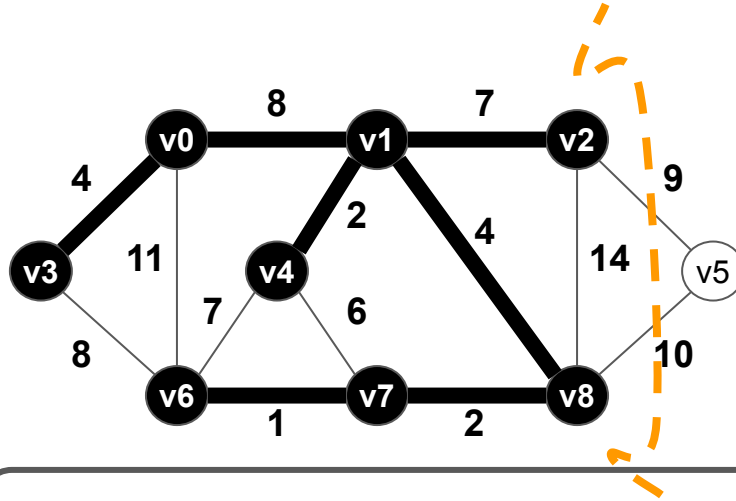
Fila de prioridade:



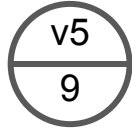
pai:

-1	0	1	0	1	2	7	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



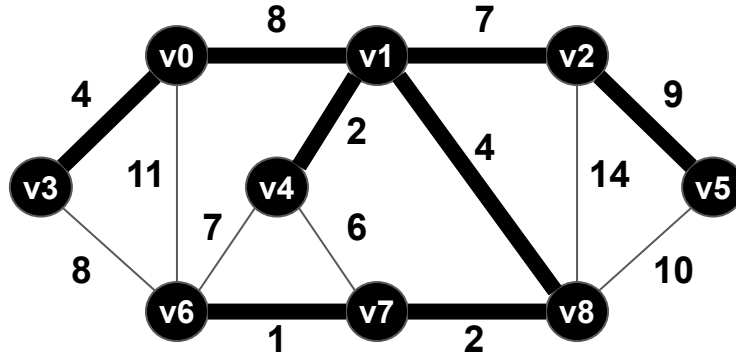
Fila de prioridade:



pai:

-1	0	1	0	1	2	7	8	1
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3



Fila de prioridade:

pai:	-1	0	1	0	1	2	7	8	1
	0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 3

Prim(G conexo, pai)

Inicialmente, a árvore que estamos construindo consiste apenas no vértice 0 de G

1. Para cada vértice w de G :
2. $pai[w] = -1$
3. Crie uma fila de prioridade Q com todos os vértices de G tendo prioridade ∞ (inf.)
4. Altere a prioridade do vértice 0 em Q para 0
5. Remova o item de menor prioridade de Q ; seja v o item removido // $v == 0$
6. Enquanto Q não está vazia:
 7. Para cada vizinho w de v em G :
 8. Se w está em Q e o peso da aresta vw é menor que a prioridade de w em Q :
 9. Altere a prioridade de w em Q para o peso da aresta vw
 10. $pai[w] = v$
 11. Remova o item de menor prioridade de Q ; seja v o item removido

A árvore que estamos construindo consiste nos vértices que já saíram da fila de prioridade Q

Algoritmo de Prim - Implementação 3

Prim(G conexo, pai)

1. Para cada vértice w de G :
2. $pai[w] = -1$
3. Crie uma fila de prioridade Q com todos os vértices de G tendo prioridade ∞ (inf.)
4. Altere a prioridade do vértice 0 em Q para 0
5. Enquanto Q não está vazia:
6. Remova o item de menor prioridade de Q ; seja v o item removido
7. Para cada vizinho w de v em G :
8. Se w está em Q e o peso da aresta vw é menor que a prioridade de w em Q :
9. Altere a prioridade de w em Q para o peso da aresta vw
10. $pai[w] = v$

Simplificação dos Passos 5 e 11 do slide anterior

Exercícios

- Exercício 1 da Lista de Exercícios “Árvores Geradoras de Peso Mínimo”.

Exercícios

- Exercício 2 da Lista de Exercícios “Árvores Geradoras de Peso Mínimo”.

Exercícios

- Exercício 3 da Lista de Exercícios “Árvores Geradoras de Peso Mínimo”.

Exercícios

- Exercício 4 da Lista de Exercícios “Árvores Geradoras de Peso Mínimo”.

Referências

- Esta apresentação é baseada nos seguintes materiais:
 1. Capítulo 23 do livro
Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms. 3rd. ed. MIT Press, 2009.
 2. Capítulo 20 do livro
Sedgewick, R. Algorithms in C++ – Part 5. Graph Algorithms. 3rd. ed. Addison-Wesley, 2002.