Árvores Geradoras de Peso Mínimo

Prof. Andrei Braga

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

Conteúdo

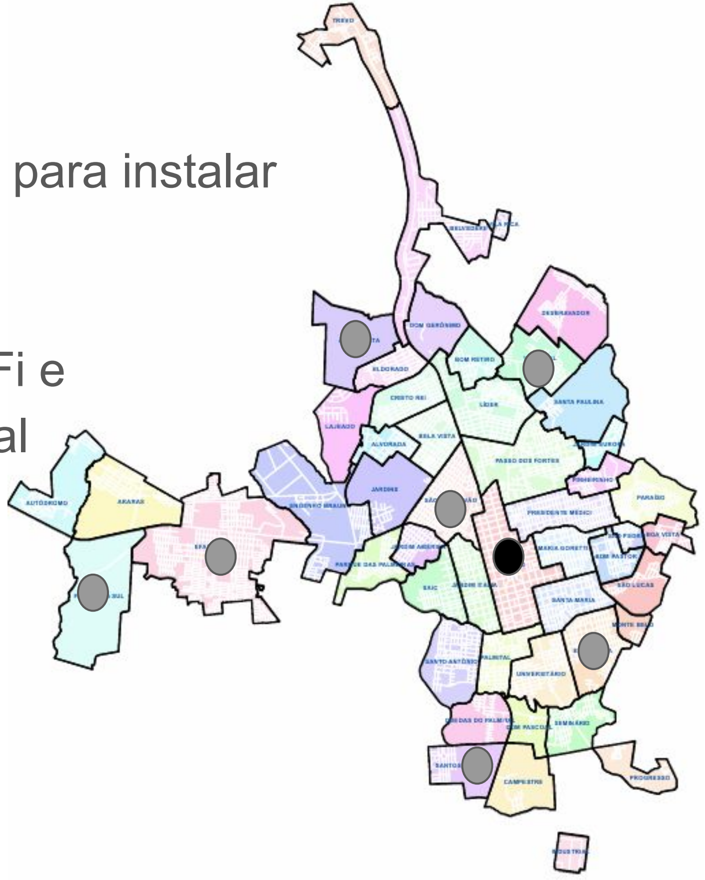
- Problema motivador
- Uma estratégia geral para encontrar árvores geradoras de peso mínimo
- Algoritmo de Prim
- Exercícios
- Referências

Problema

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

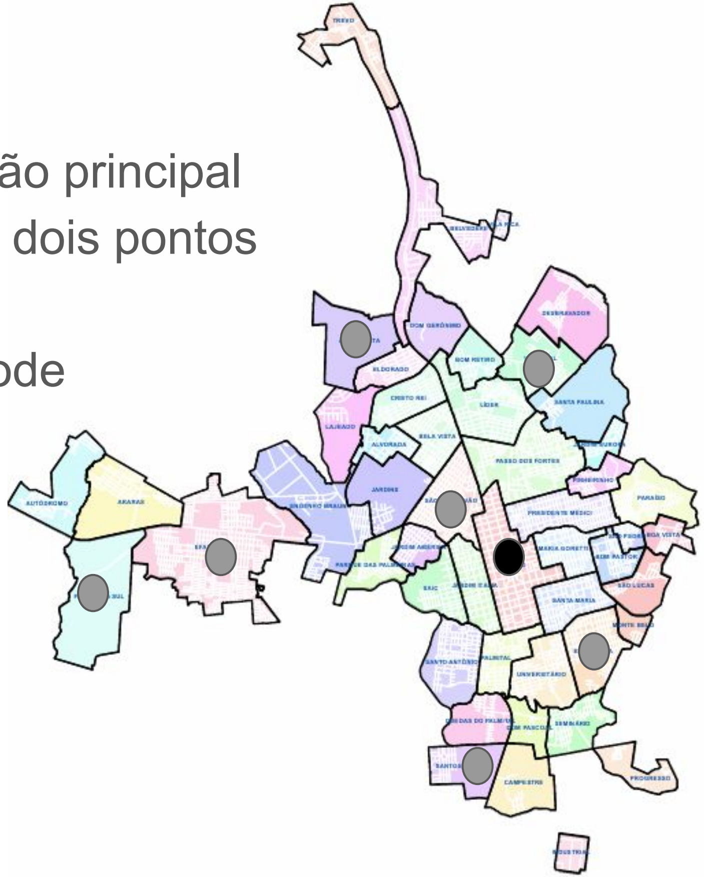
Problema

- 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 e a nossa empresa possui uma estação principal de onde será fornecida a comunicação com a Internet aos pontos de acesso Wi-Fi



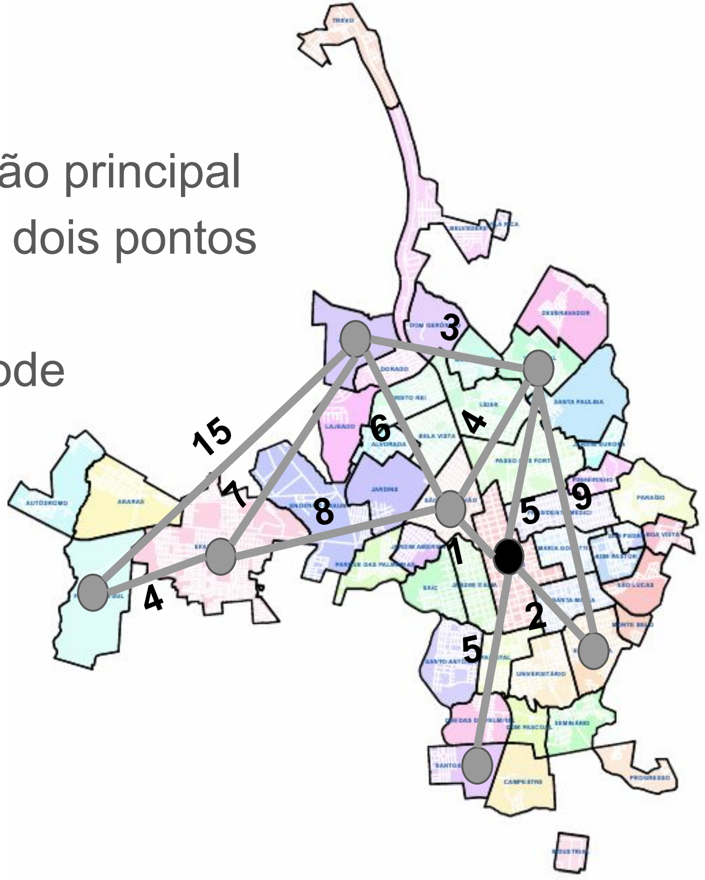
Problema

- Poderemos usar cabos para conectar a estação principal a um ponto de acesso Wi-Fi ou 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



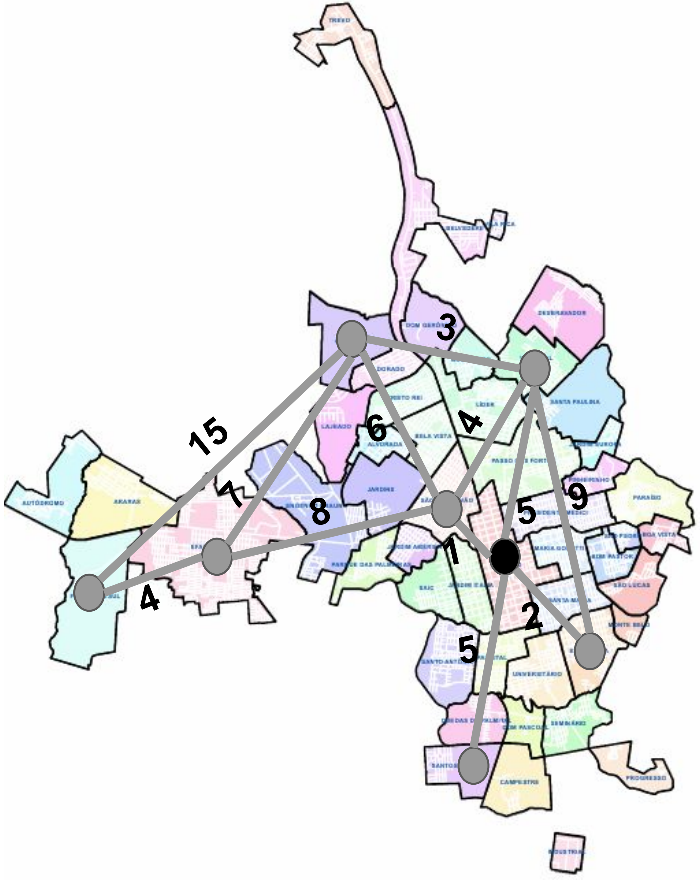
Problema

- Poderemos usar cabos para conectar a estação principal a um ponto de acesso Wi-Fi ou 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



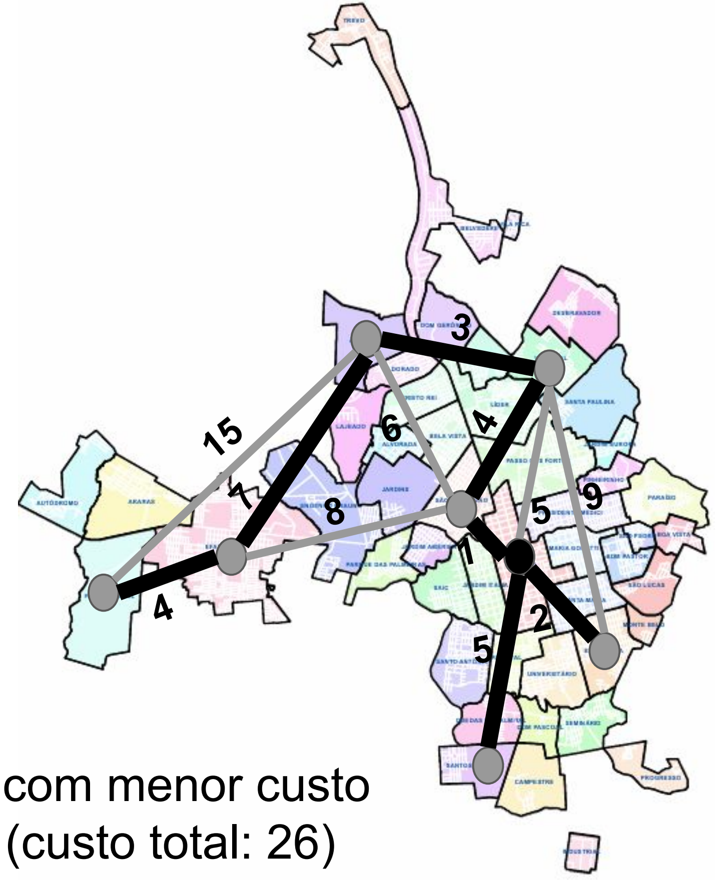
Problema

- **Problema:** Como conectar todos os pontos de acesso Wi-Fi e a estação principal tendo o menor custo possível?



Problema

- **Problema:** Como conectar todos os pontos de acesso Wi-Fi e a estação principal tendo o menor custo possível?



Solução com menor custo possível (custo total: 26)

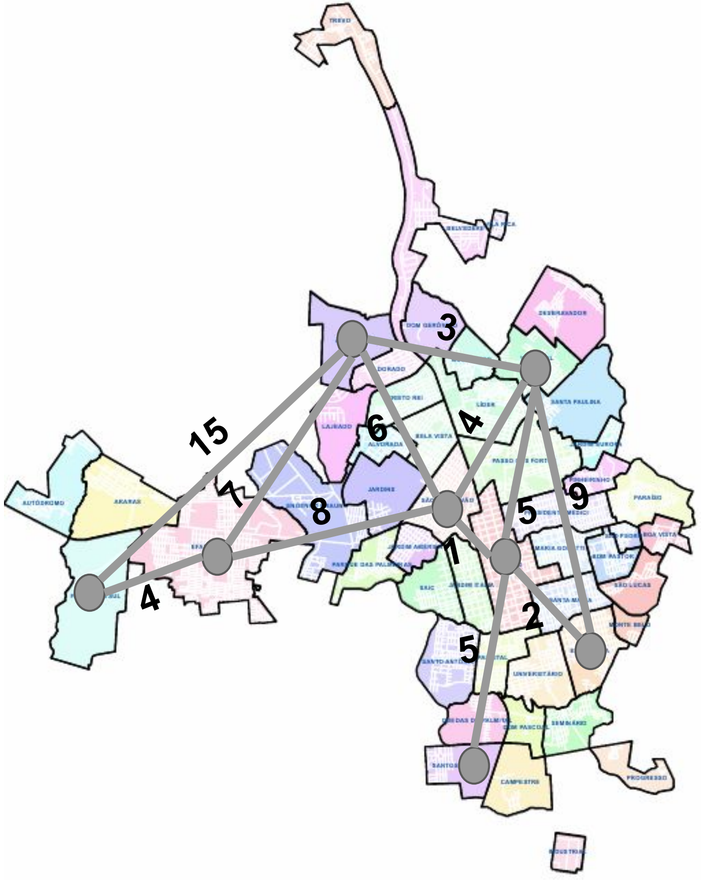
Problema - Modelagem com grafos

- Podemos modelar este problema usando um grafo *G* tal que
  - os vértices de *G* representam os pontos a serem conectados (pontos de acesso Wi-Fi e a estação principal) 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

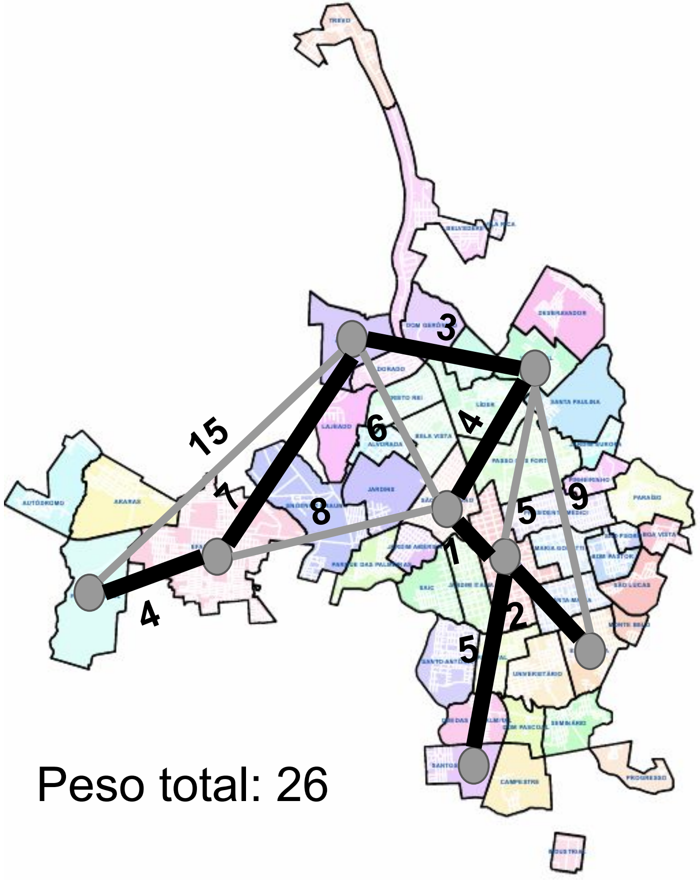
Problema - Modelagem com grafos

- 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

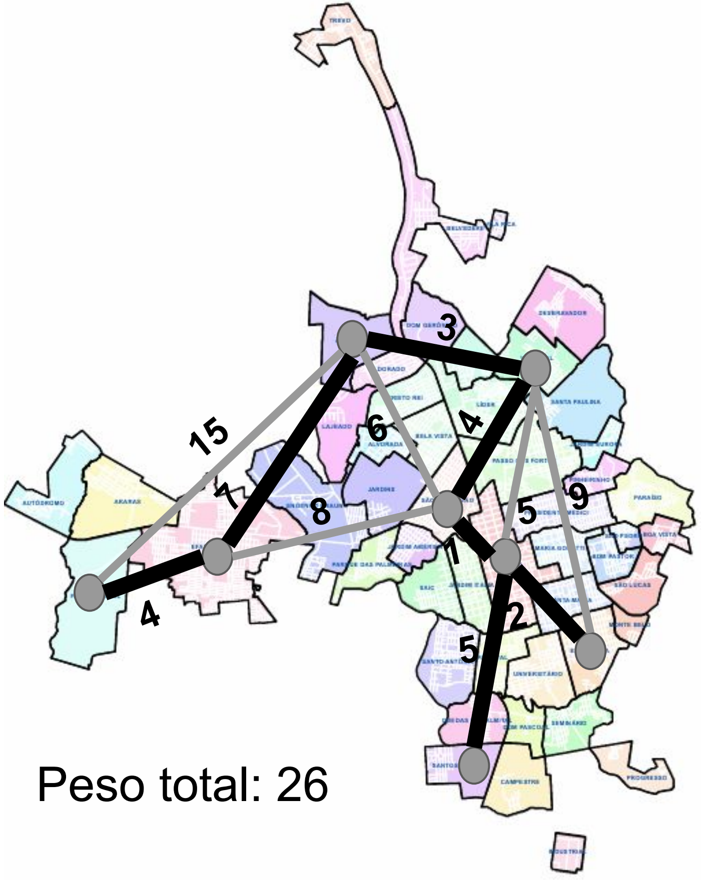


Peso total: 26

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*

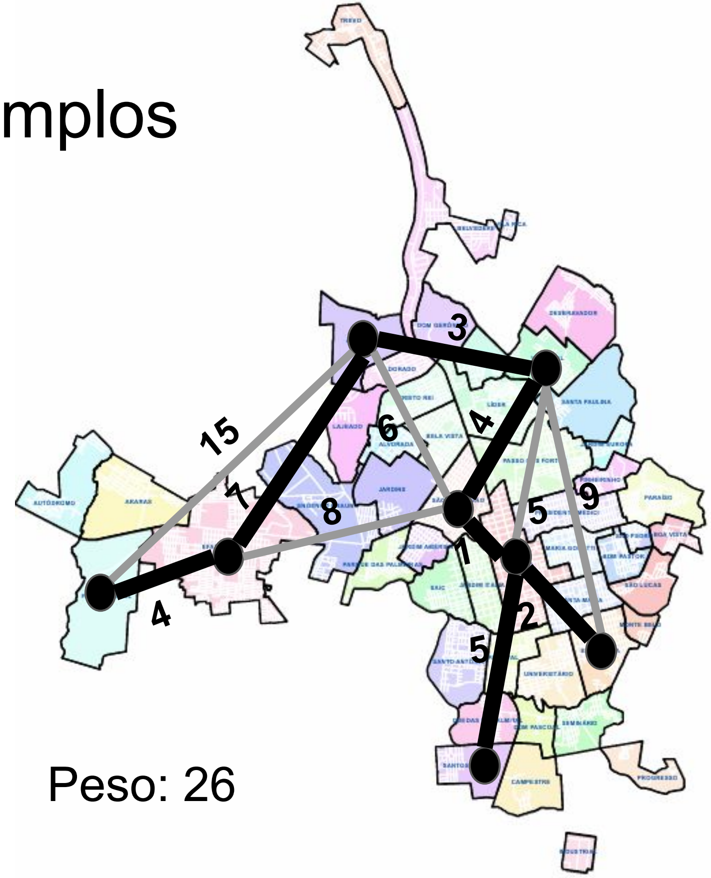
- O nosso objetivo é, então, encontrar uma árvore geradora *T* de *G* que tenha peso mínimo, com o **peso da árvore geradora** *T* sendo igual à soma dos pesos das suas arestas

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

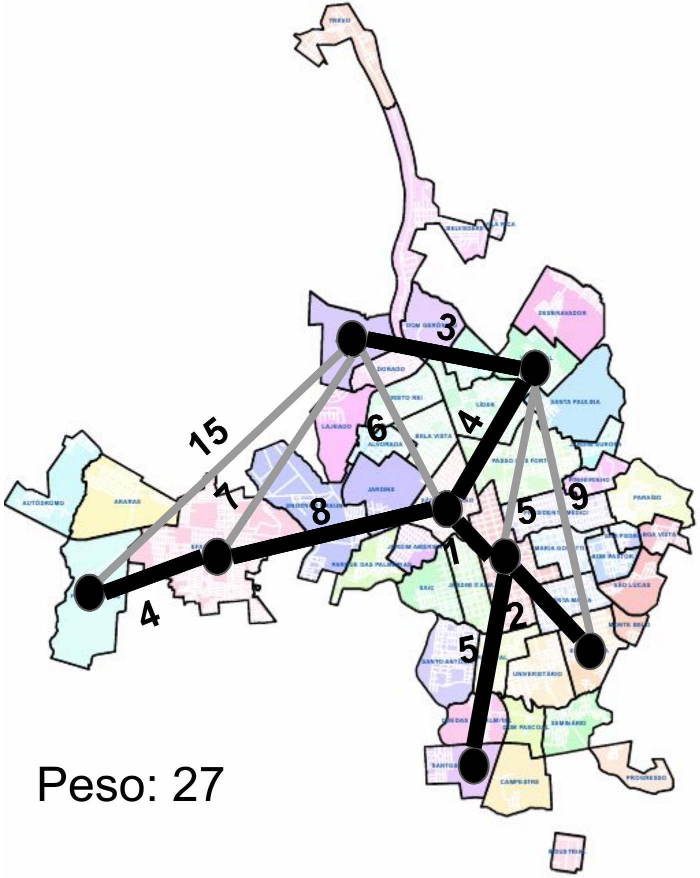
- O nosso objetivo é, então, encontrar uma **árvore geradora de peso mínimo** de *G*

Exemplos



Peso: 26

Árvore geradora de peso mínimo



Peso: 27

Árvore geradora de peso mínimo

Algoritmos

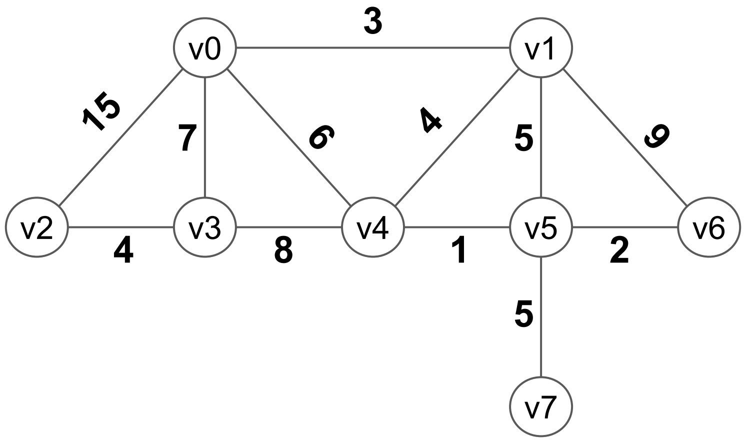
- O problema de encontrar uma árvore geradora de peso mínimo de um grafo com pesos nas arestas tem sido estudado (pelo menos) desde os anos 1920
- Para este problema, vamos considerar que o **grafo** recebido como **entrada** é **conexo** (caso contrário, o problema não admite solução)
- Veremos dois 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

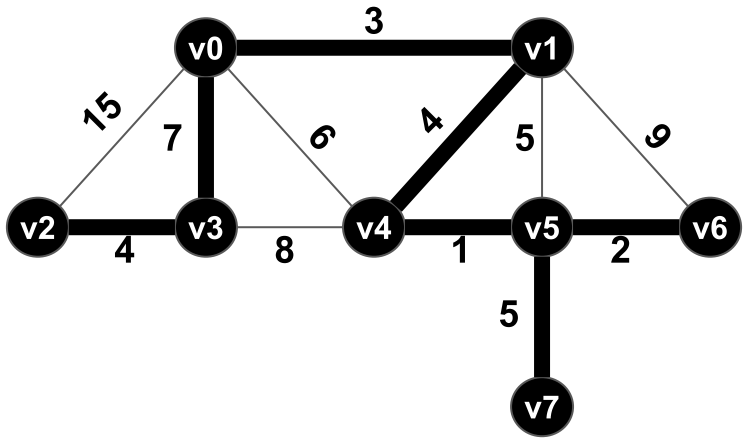
Aresta segura

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

Aresta segura

- Exemplo:



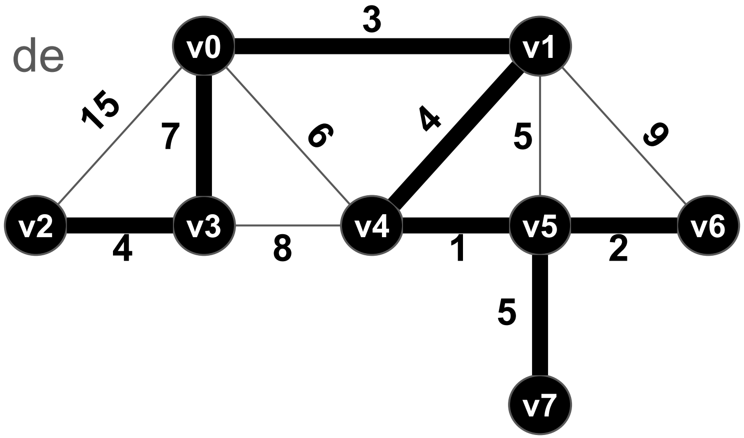


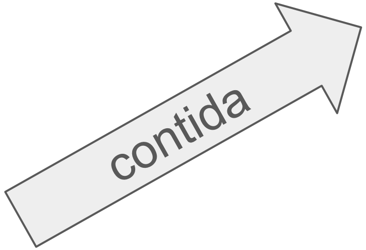
Árvore geradora de peso mínimo

Aresta segura

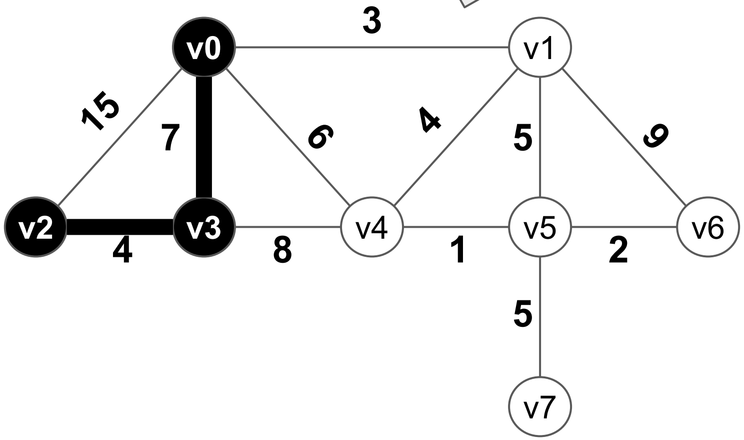
- Exemplo:

Árvore geradora de peso mínimo





T:

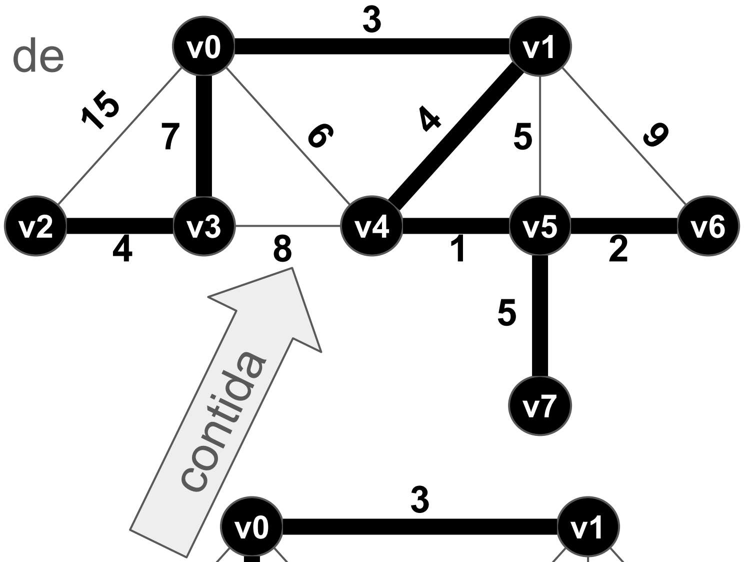


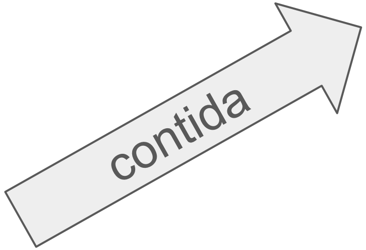
Árvore contida em uma árvore geradora de peso mínimo

Aresta segura

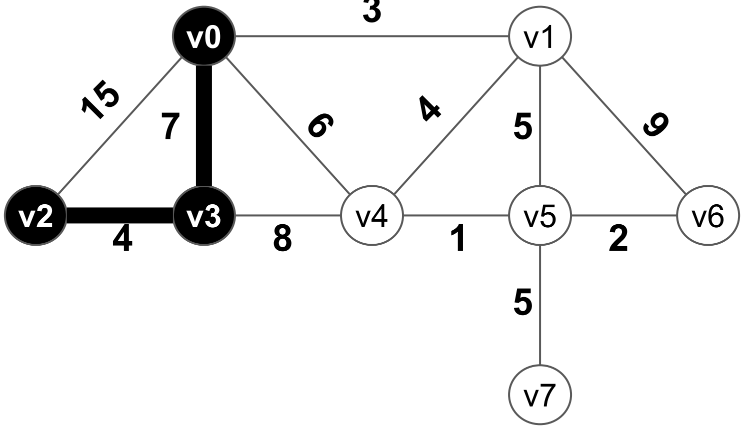
- Exemplo:

Árvore geradora de peso mínimo



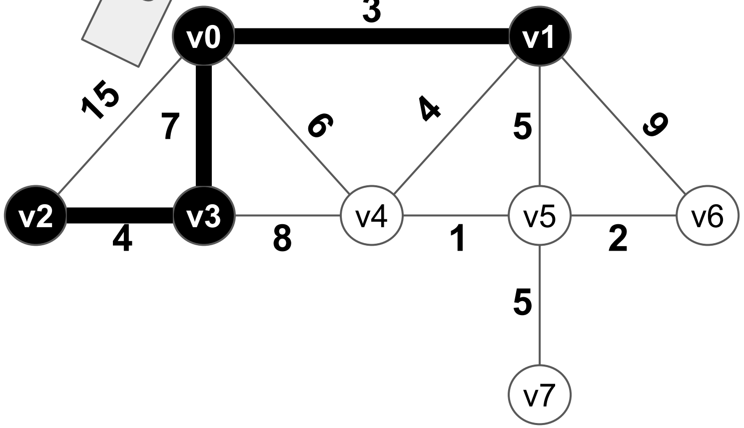


T:



Árvore contida em uma árvore geradora de peso mínimo

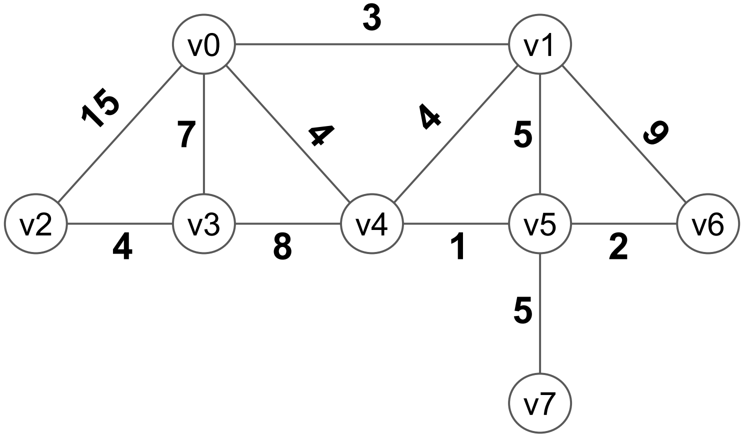
T:

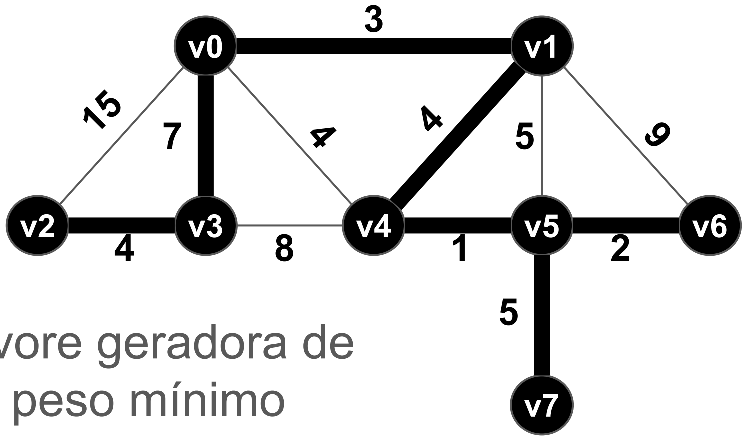


v0v1 é uma aresta segura para *T*

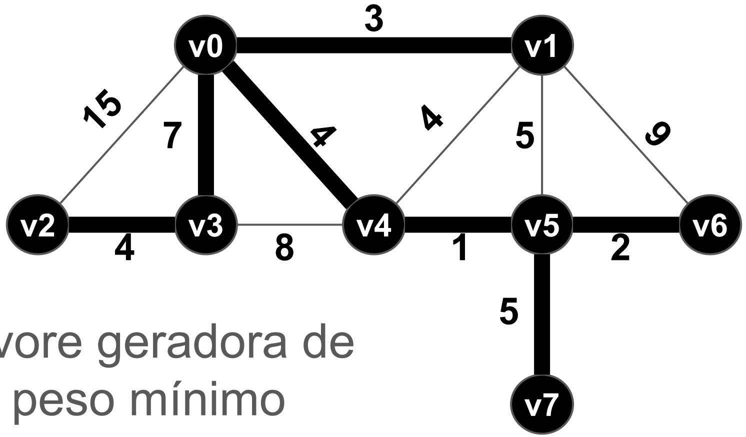
Aresta segura

- Exemplo:





Árvore geradora de peso mínimo

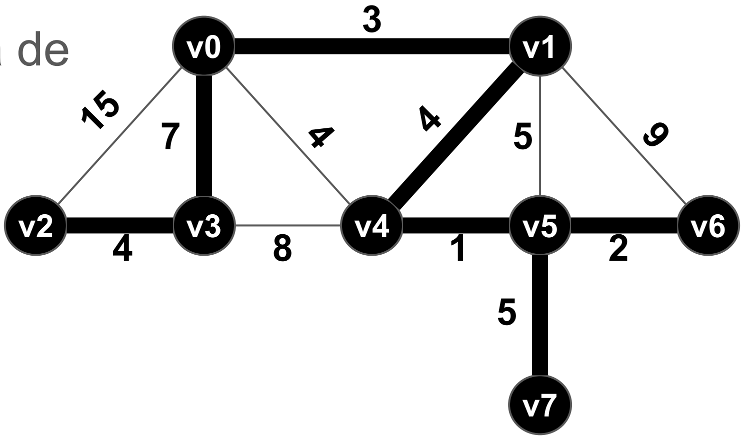


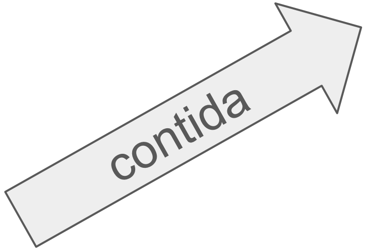
Árvore geradora de peso mínimo

Aresta segura

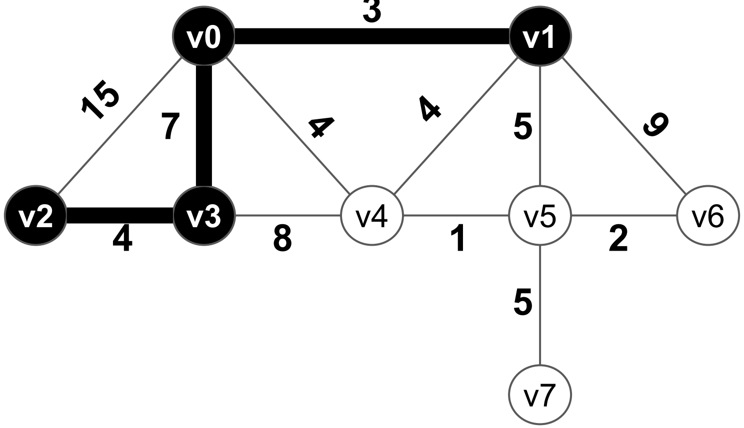
- Exemplo:

Árvore geradora de peso mínimo





T:

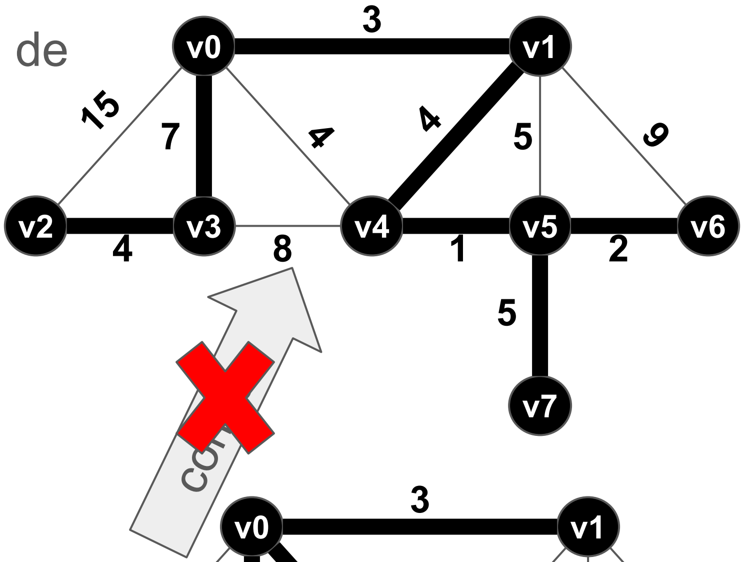


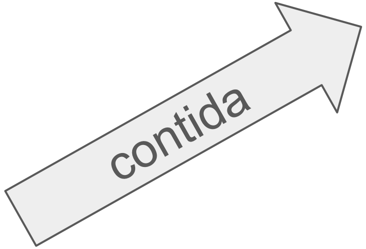
Árvore contida em uma árvore geradora de peso mínimo

Aresta segura

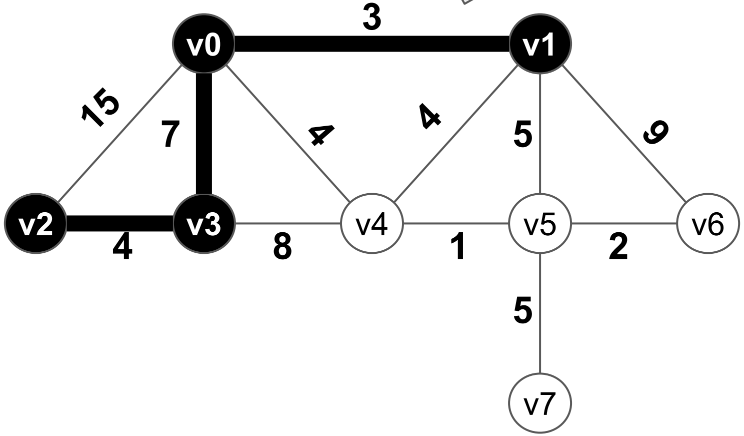
- Exemplo:

Árvore geradora de peso mínimo

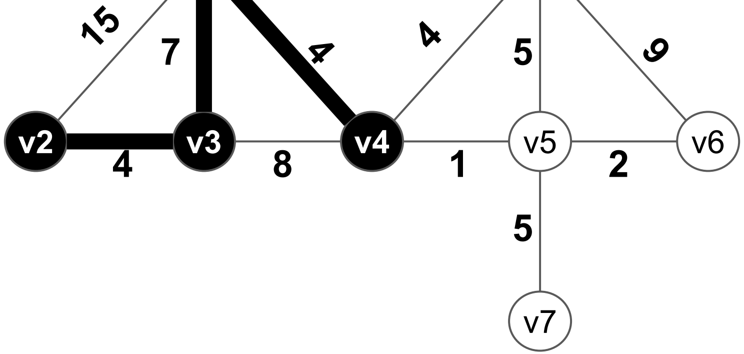




T:



T:

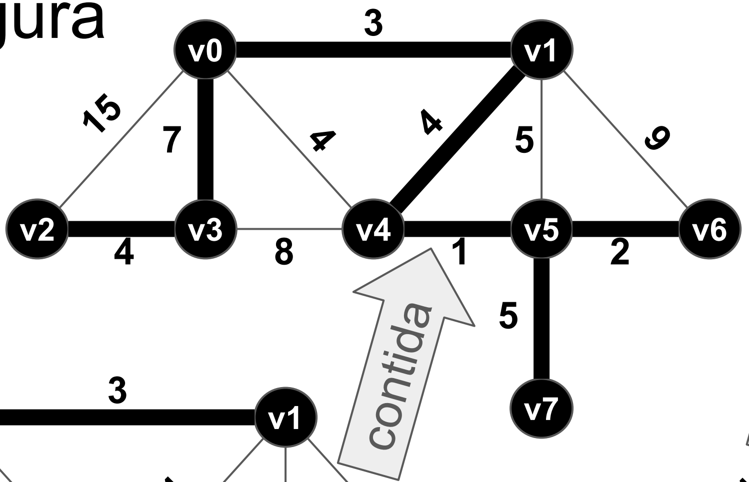


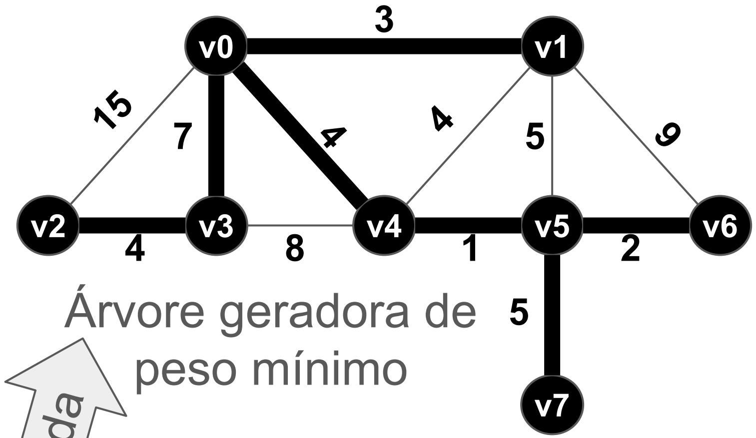
Árvore contida em uma árvore geradora de peso mínimo

Aresta segura

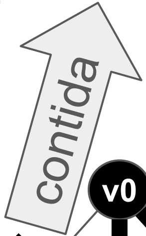
- Exemplo:

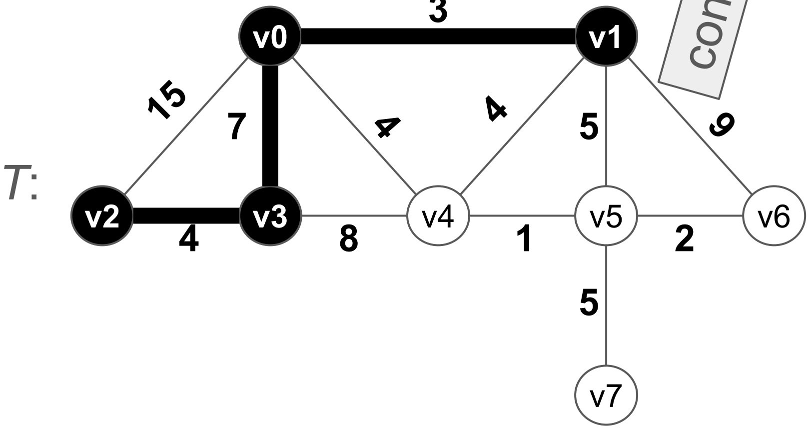
Árvore geradora de
peso mínimo



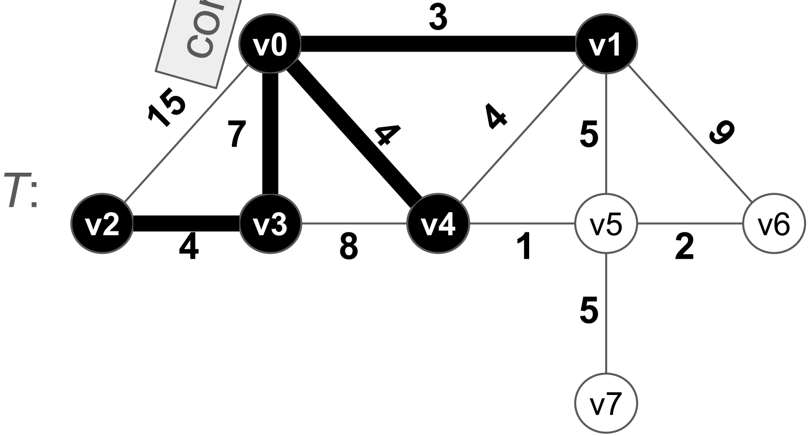


Árvore geradora de
peso mínimo





Árvore contida em uma árvore
geradora de peso mínimo



v0v4 é uma aresta segura para *T*

Como encontrar uma árvore geradora de peso mínimo

EncontraArvGerPesMin(*G* conexo)

1. $T = (\{0\}, \varnothing)$
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*

Inicialmente, *T* é uma árvore que consiste apenas no vértice 0 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 uma árvore geradora de peso mínimo de *G*

Ao fim do laço das linhas 2-4, temos uma árvore *T*
• que contém todos os vértices de *G* e
• que está contida em uma árvore geradora de peso mínimo de *G*

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

Como encontrar uma árvore geradora de peso mínimo

EncontraArvGerPesMin(G conexo)

1. $T = (\{ 0 \}, \varnothing)$
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

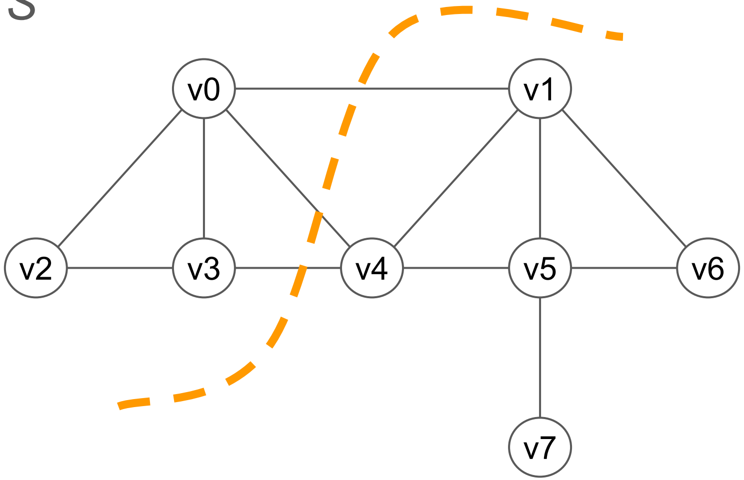
- O Algoritmo de Prim se baseia em um resultado interessante
- Antes de descrever este resultado vamos definir alguns conceitos importantes

Corte

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

- Exemplo:

G:

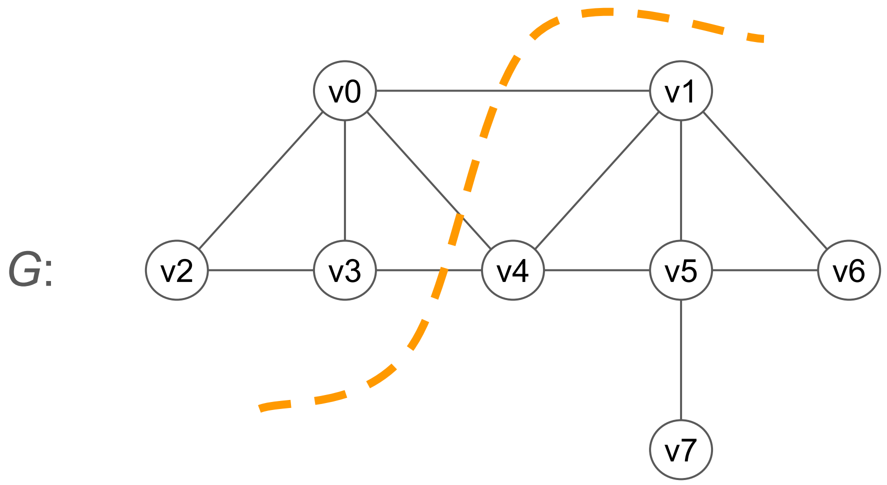


({ v0, v2, v3 }, { v1, v4, v5, v6, v7 }) é um corte de *G*

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:

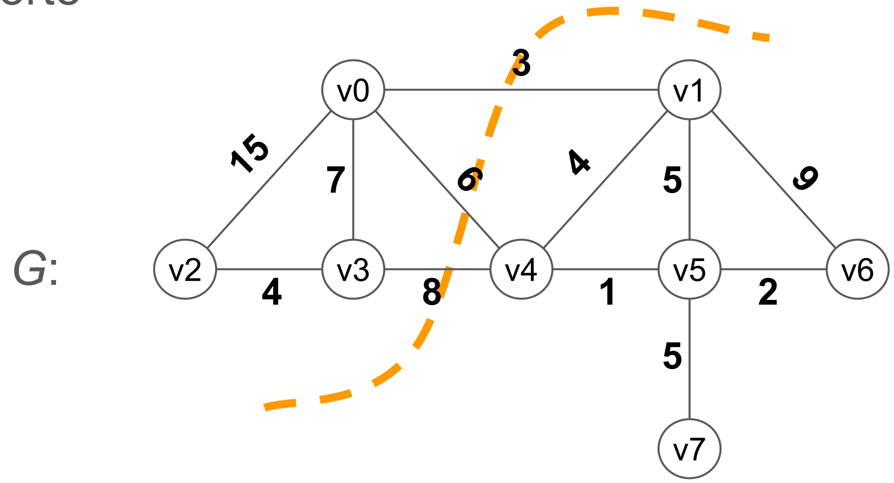


  - v0v4 e v3v4 são arestas do corte
  - v0v2 e v5v6 **não** são arestas do corte

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

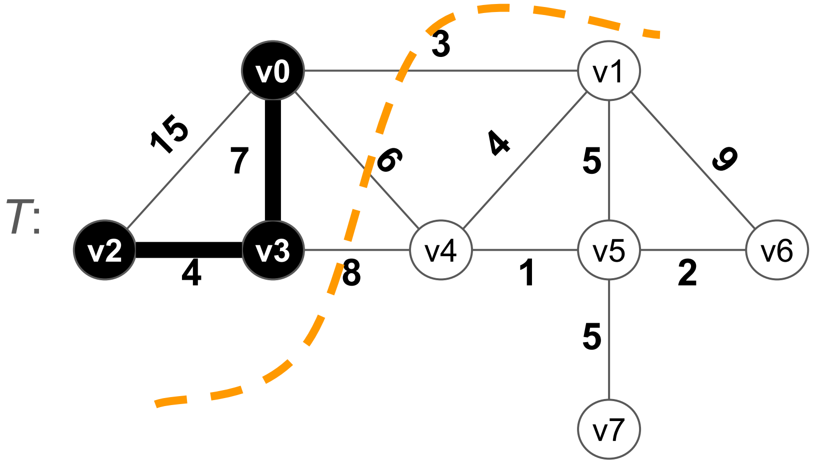
- Exemplo:

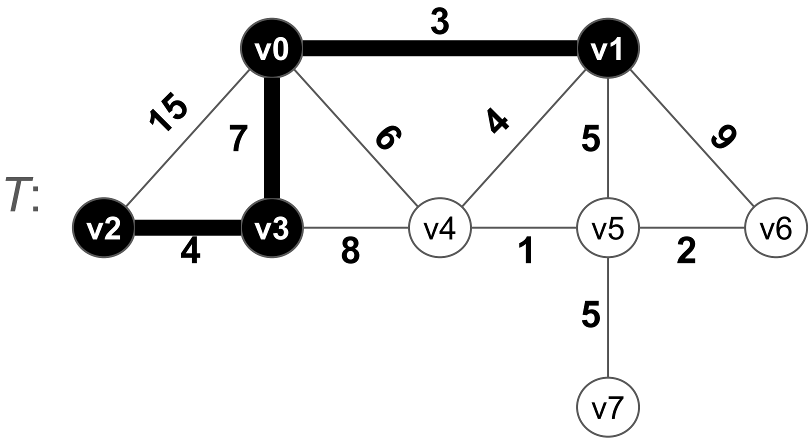


  - v0v1 é uma aresta de peso mínimo do corte
  - v3v4 **não** é uma aresta de peso mínimo do corte

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 uma árvore geradora de peso mínimo de *G*. Se *uv* é uma aresta de peso mínimo do corte (*V*(*T*), *V*(*G*) \ *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*

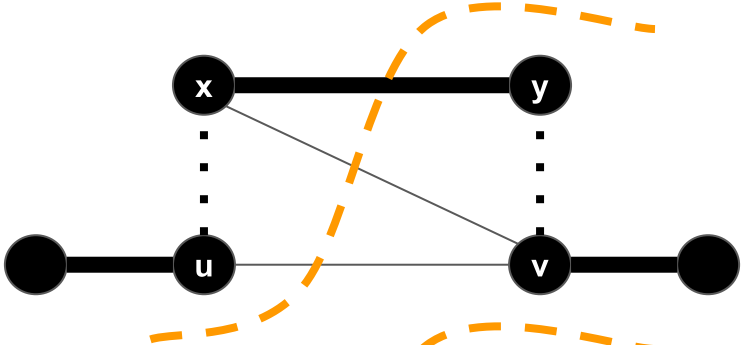
Aresta segura em um corte

- **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*

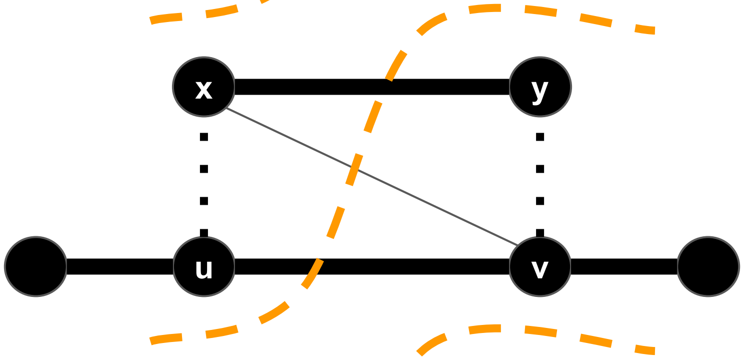
Aresta segura em um corte

- **Prova:**

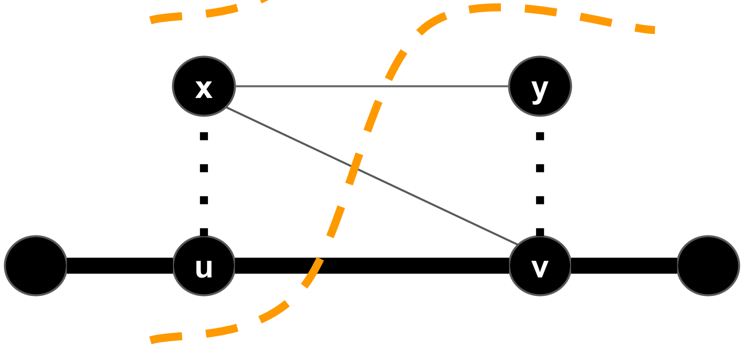
T’:



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* □

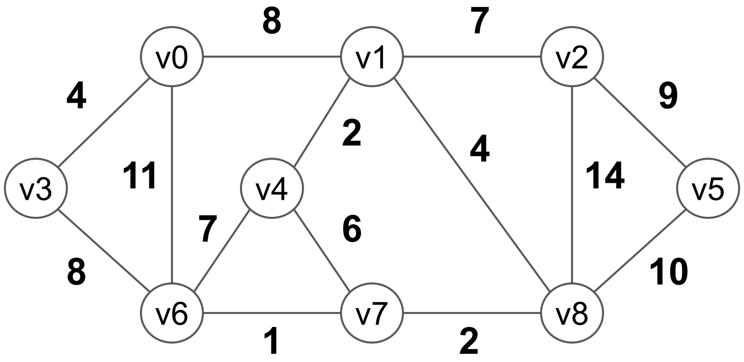
Algoritmo de Prim

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

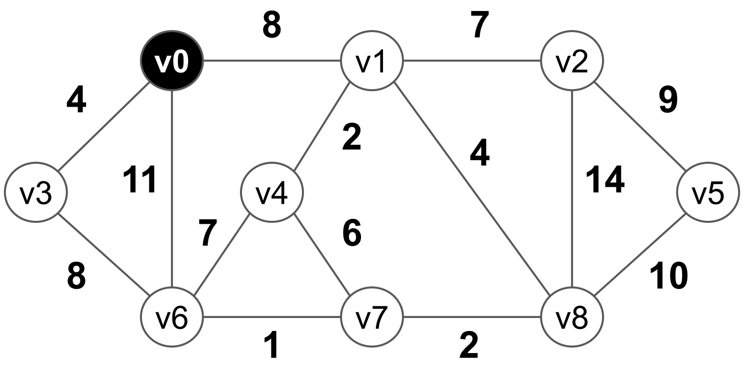
Prim(G conexo)

1. T = ({ 0 }, ∅)
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

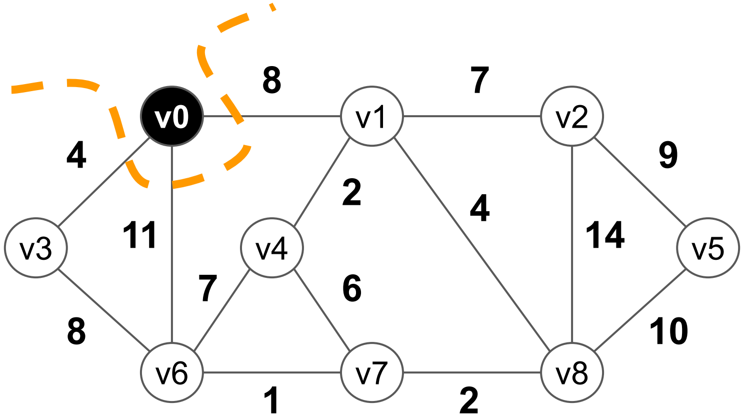
Algoritmo de Prim



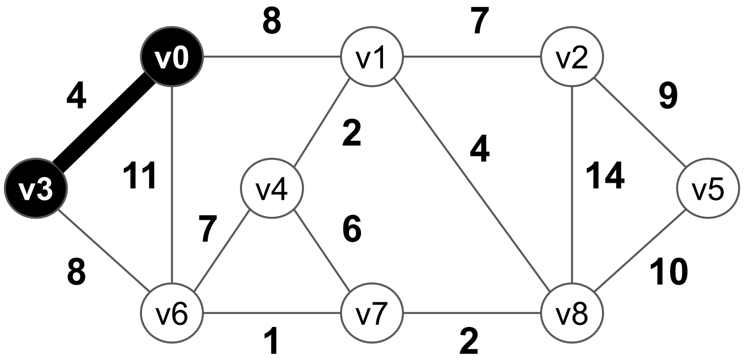
Algoritmo de Prim



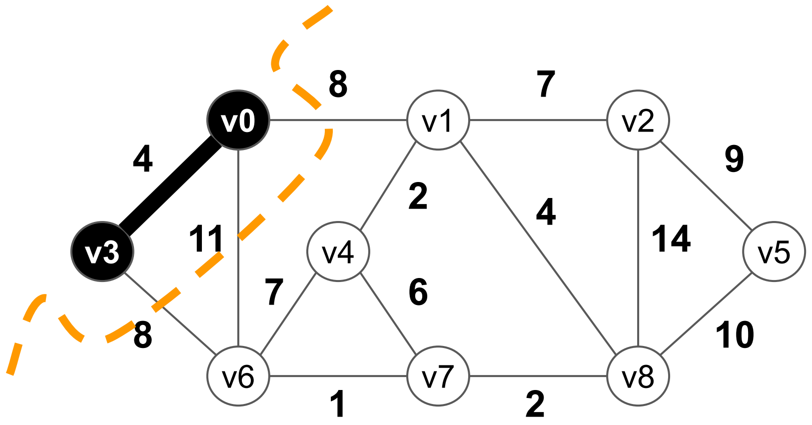
Algoritmo de Prim



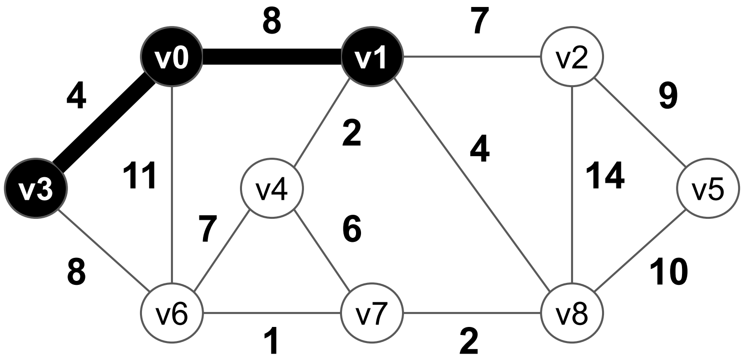
Algoritmo de Prim



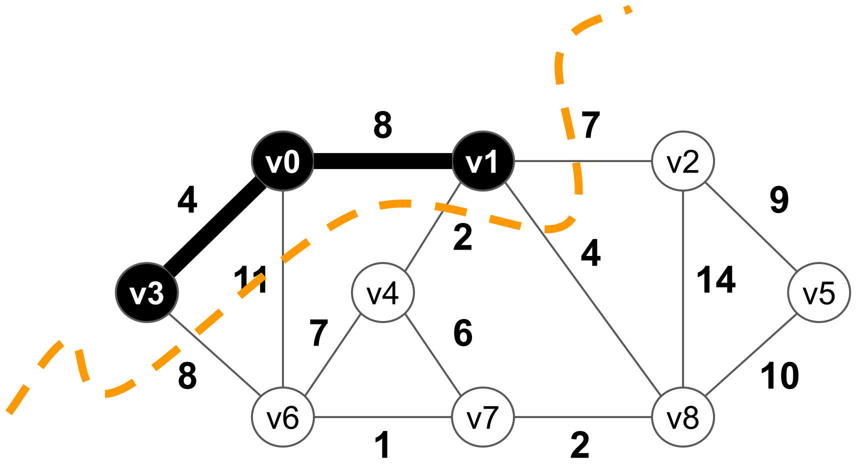
Algoritmo de Prim



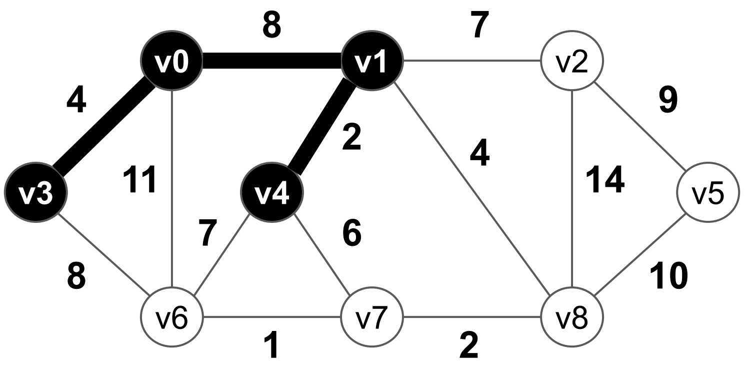
Algoritmo de Prim



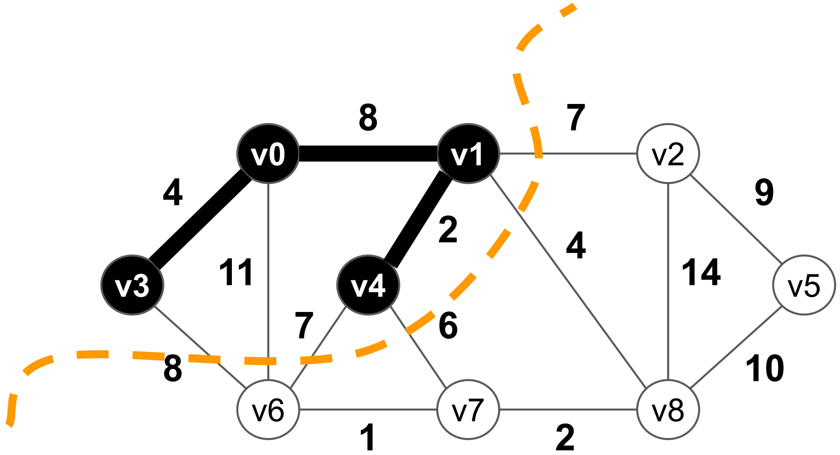
Algoritmo de Prim



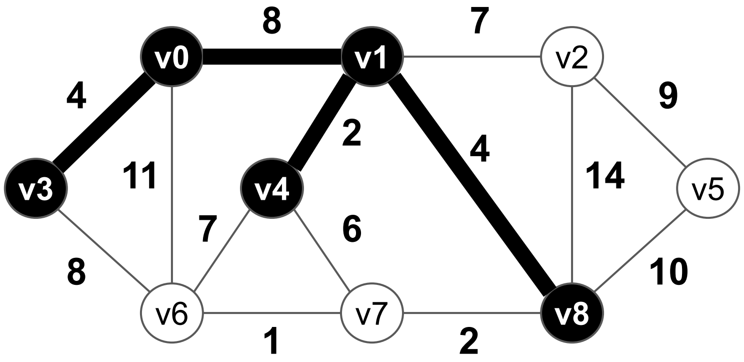
Algoritmo de Prim



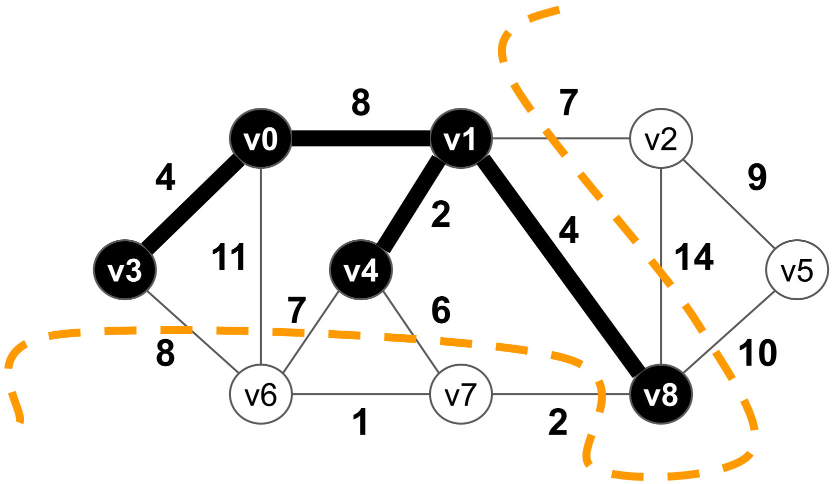
Algoritmo de Prim



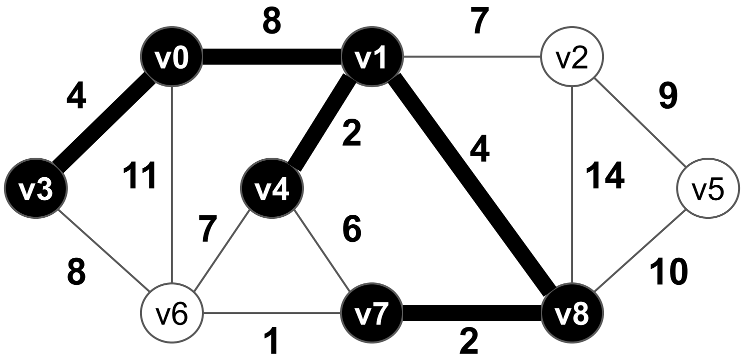
Algoritmo de Prim



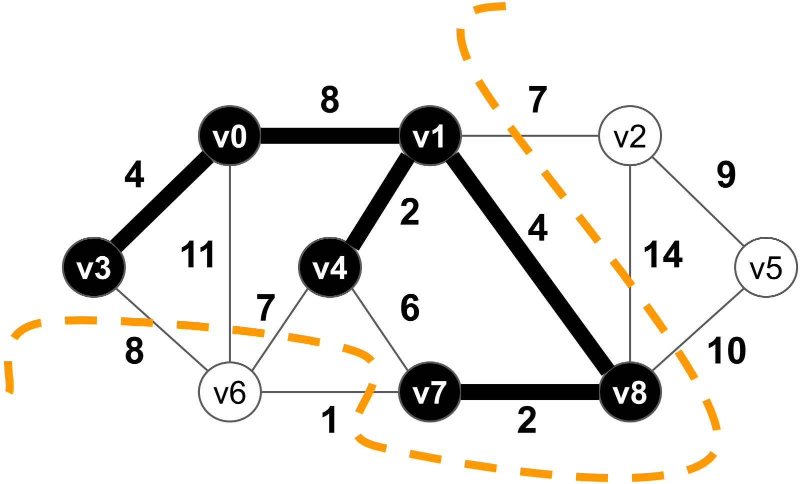
Algoritmo de Prim



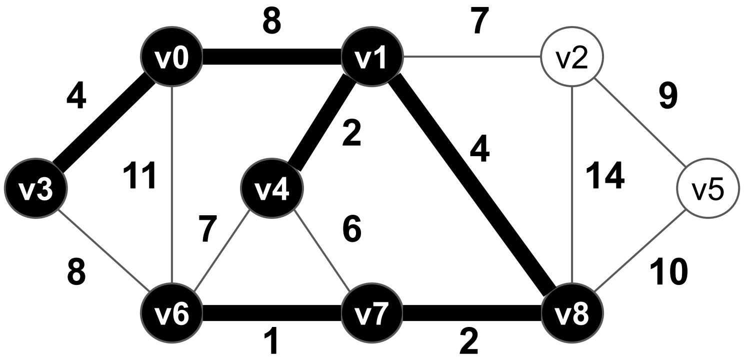
Algoritmo de Prim



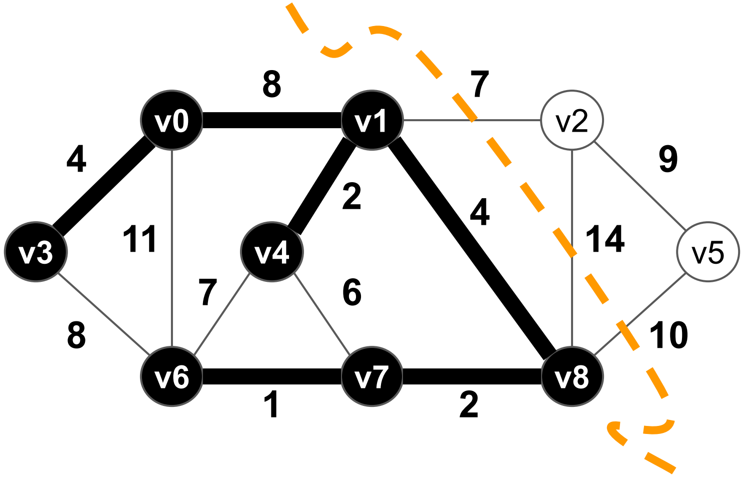
Algoritmo de Prim



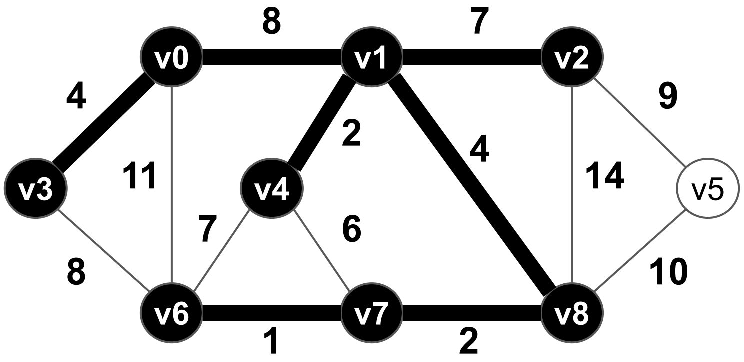
Algoritmo de Prim



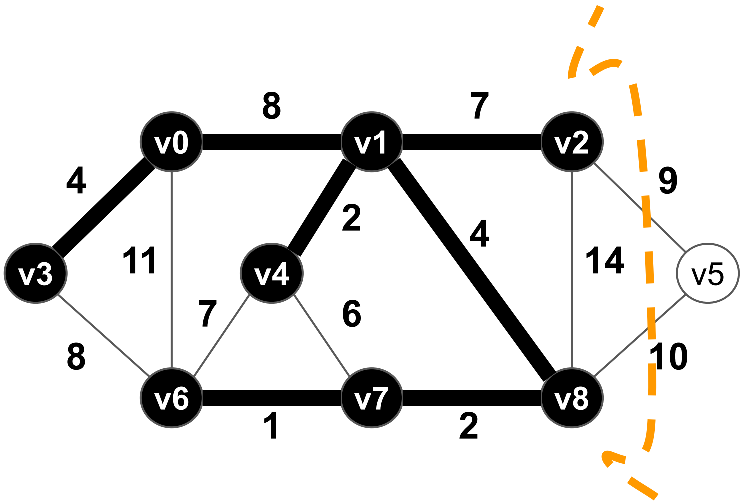
Algoritmo de Prim



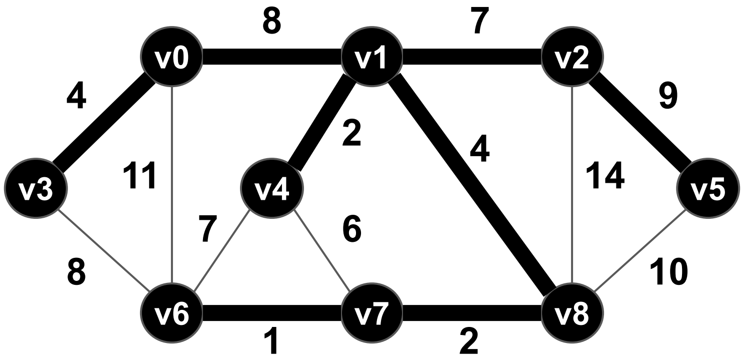
Algoritmo de Prim



Algoritmo de Prim



Algoritmo de Prim



Algoritmo de Prim

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

Prim(G conexo)

1. T = ({ 0 }, ∅)
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

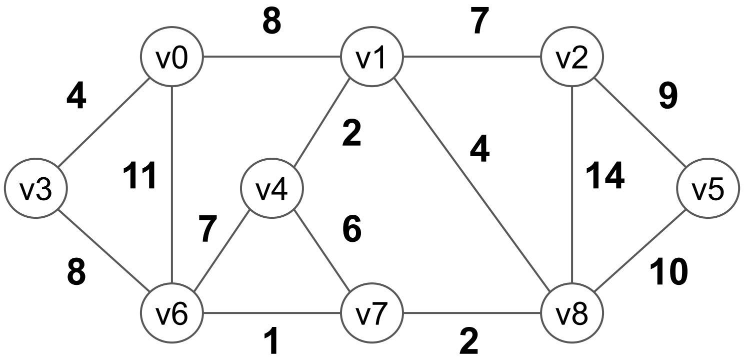
Algoritmo de Prim - Implementação 1 (ineficiente)

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

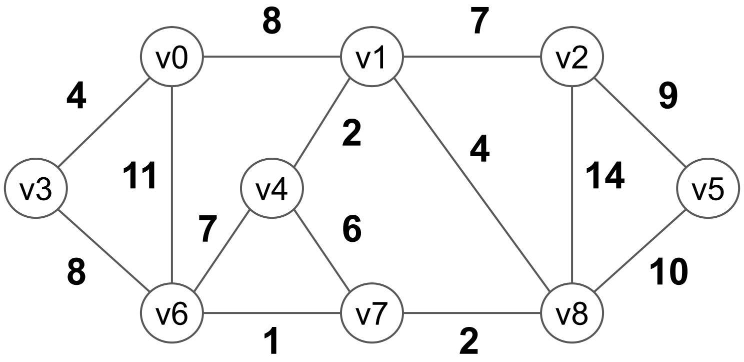
Prim(*G* conexo)

```
T = ({ 0 }, ∅)
Enquanto T não é uma árvore geradora de G:
    Inicialize o peso de uv como infinito
    Para cada aresta xy de G:
        Se x ∈ V(T) e y ∉ V(T) ou y ∈ V(T) e x ∉ V(T):
            Se o peso de xy é menor que o peso de uv:
                uv = xy
    Adicione uv a T
Retorne T
```

Algoritmo de Prim - Implementação 2



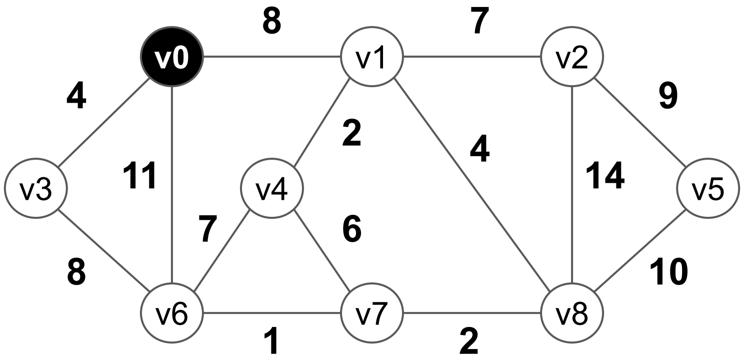
Algoritmo de Prim - Implementação 2



aresta_menor_peso_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

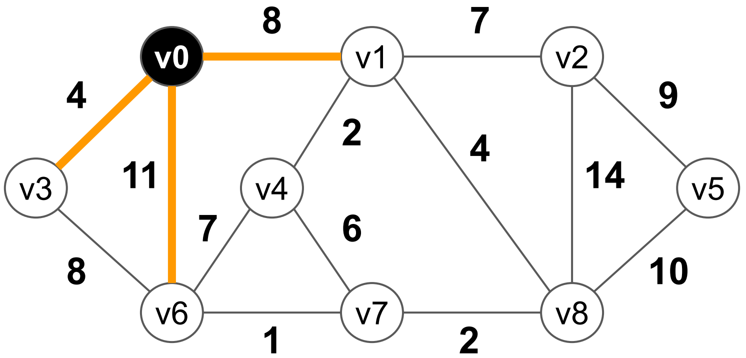
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

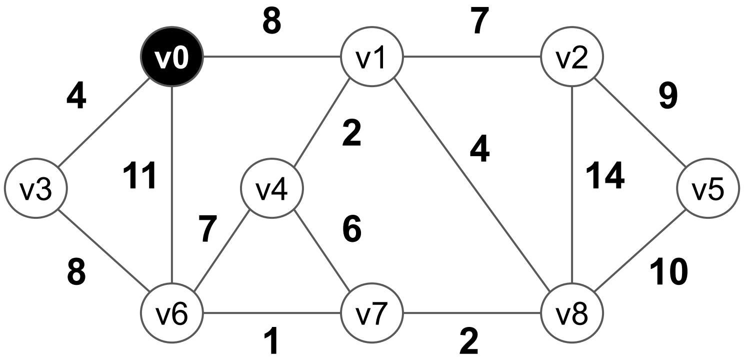
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

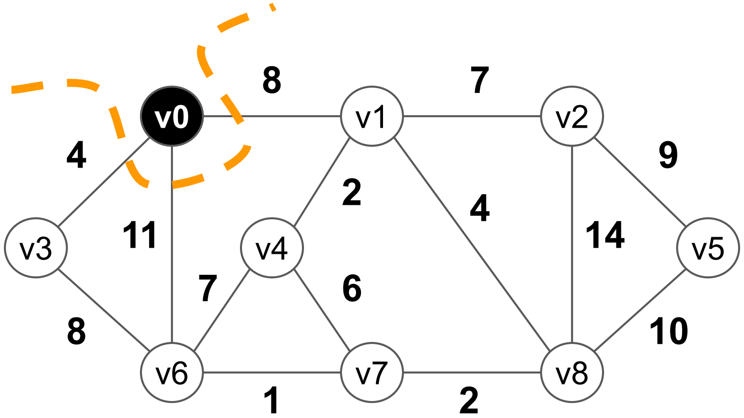
0	1	2	3	4	5	6	7	8
-1 -1 ∞	0 1 8	-1 -1 ∞	0 3 4	-1 -1 ∞	-1 -1 ∞	0 6 11	-1 -1 ∞	-1 -1 ∞

Algoritmo de Prim - Implementação 2



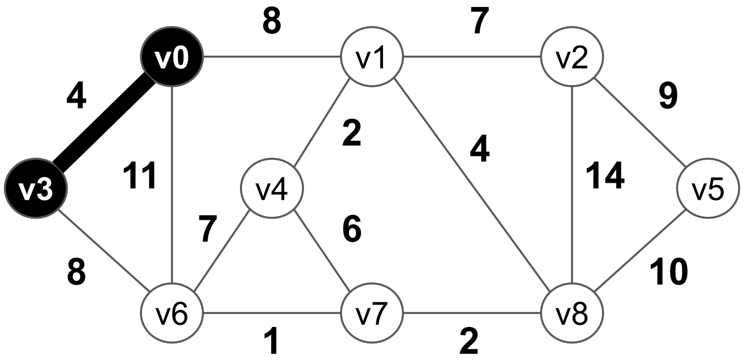
aresta_menor_peso_para_arvore:	-1 -1 ∞	0 1 8	-1 -1 ∞	0 3 4	-1 -1 ∞	-1 -1 ∞	0 6 11	-1 -1 ∞	-1 -1 ∞
	0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 2



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

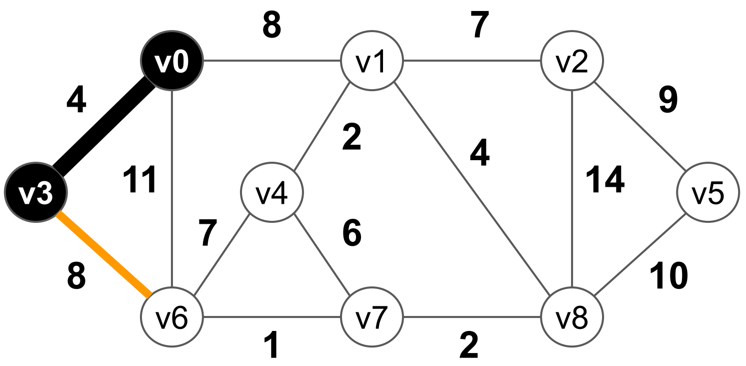
Algoritmo de Prim - Implementação 2



aresta_menor_peso_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

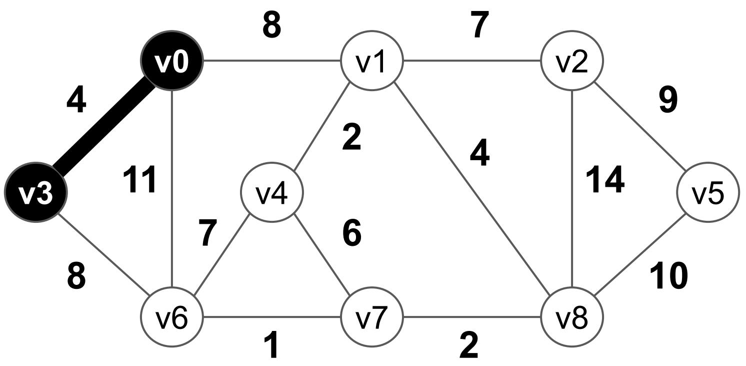
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

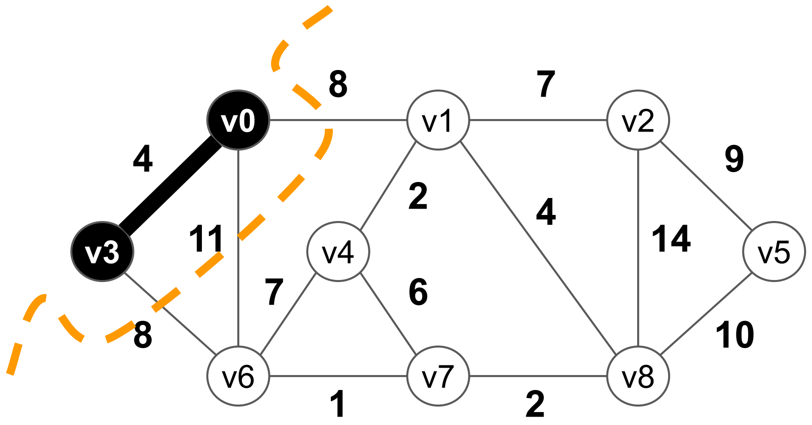
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

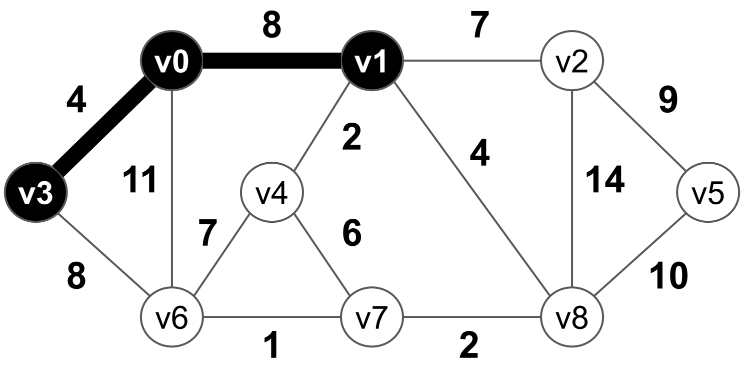
Algoritmo de Prim - Implementação 2



aresta_menor_peso_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

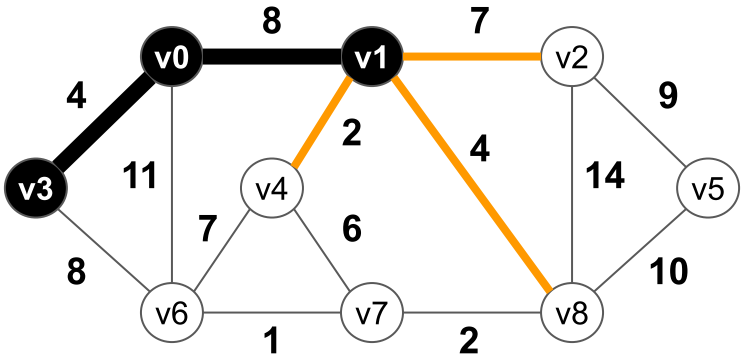
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

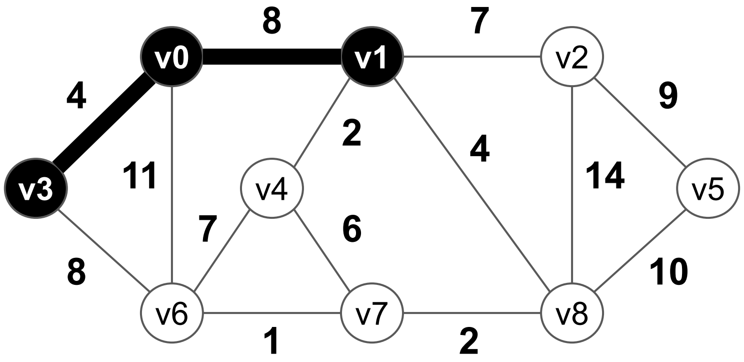
0	1	2	3	4	5	6	7	8
-1 -1 ∞	0 1 8	-1 -1 ∞	0 3 4	-1 -1 ∞	-1 -1 ∞	3 6 8	-1 -1 ∞	-1 -1 ∞

Algoritmo de Prim - Implementação 2



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

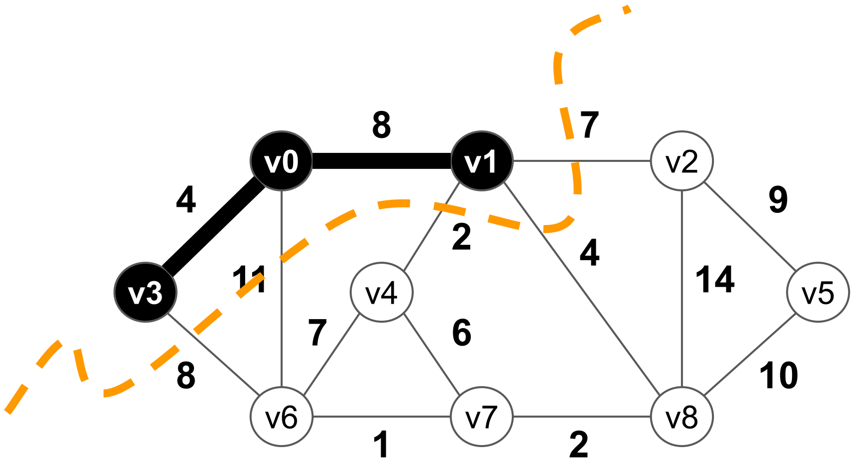
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

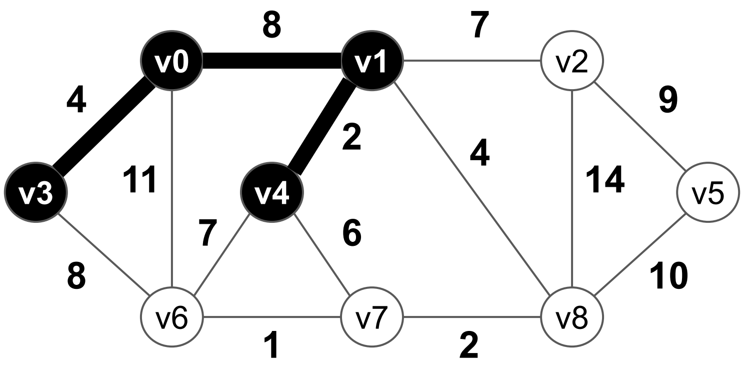
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

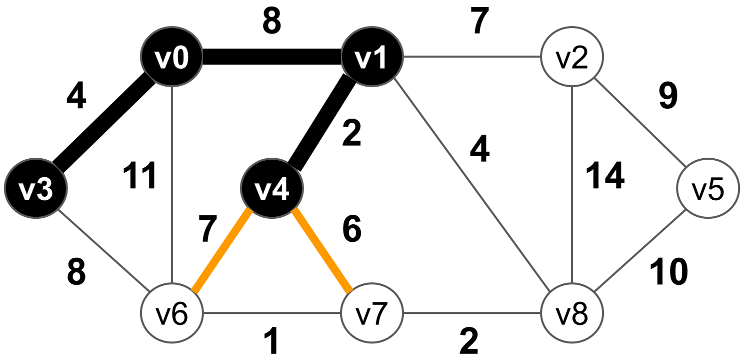
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

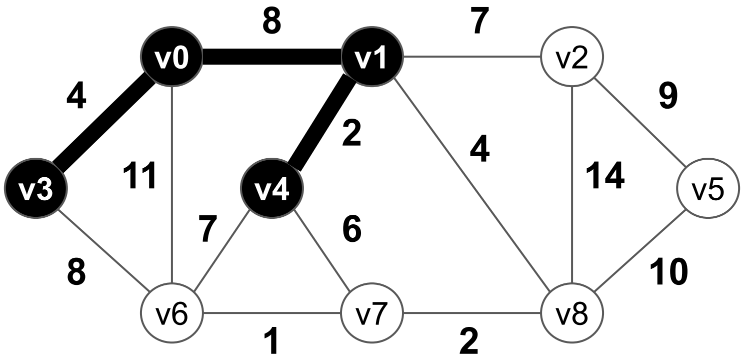
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

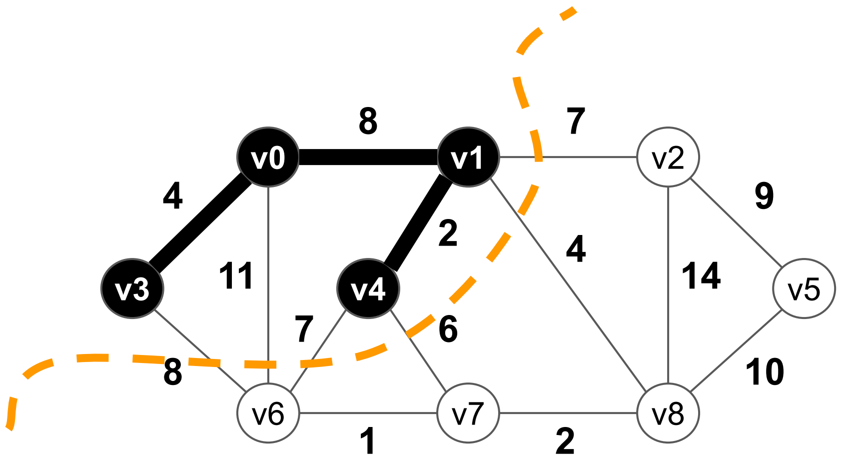
Algoritmo de Prim - Implementação 2



aresta_menor_peso_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

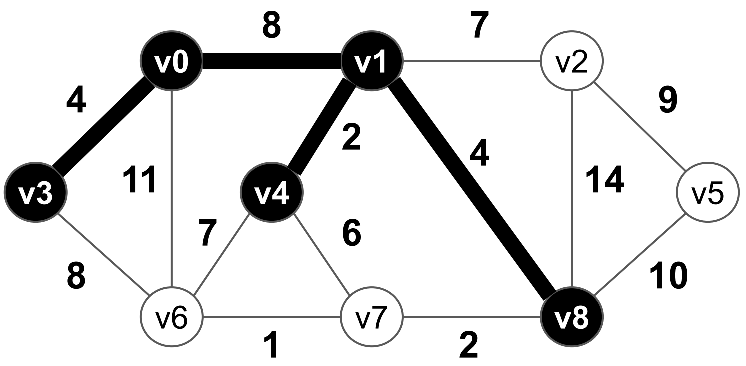
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

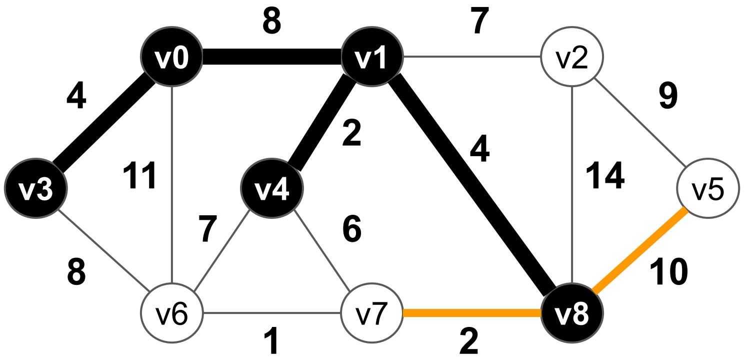
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

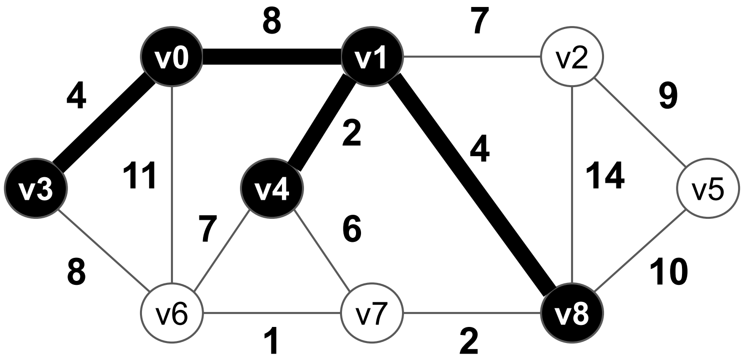
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

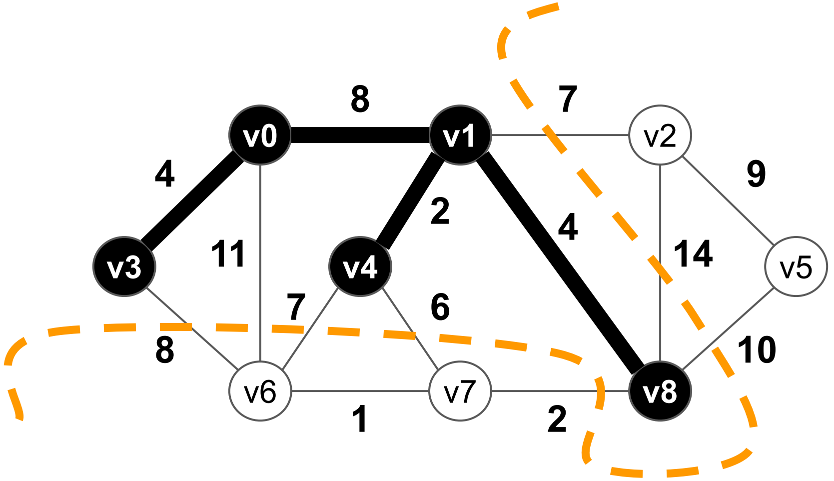
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

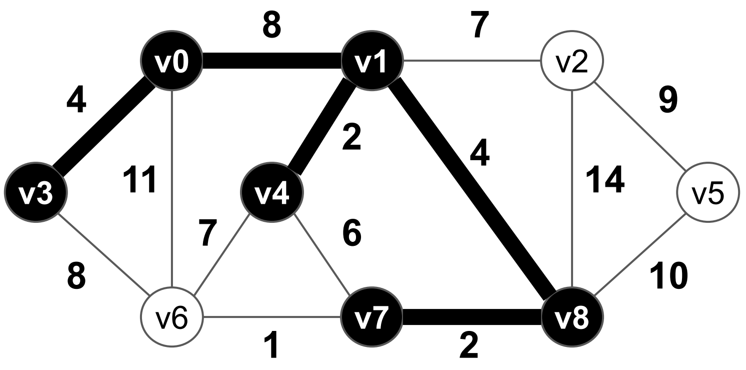
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

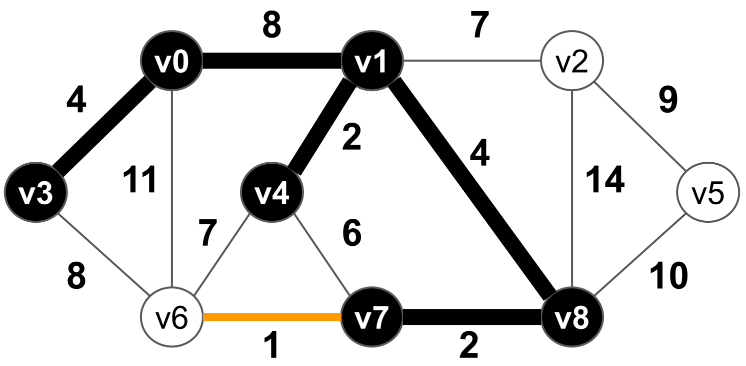
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

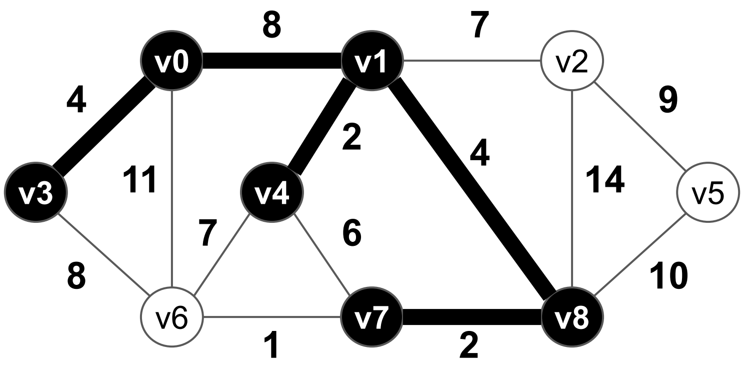
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

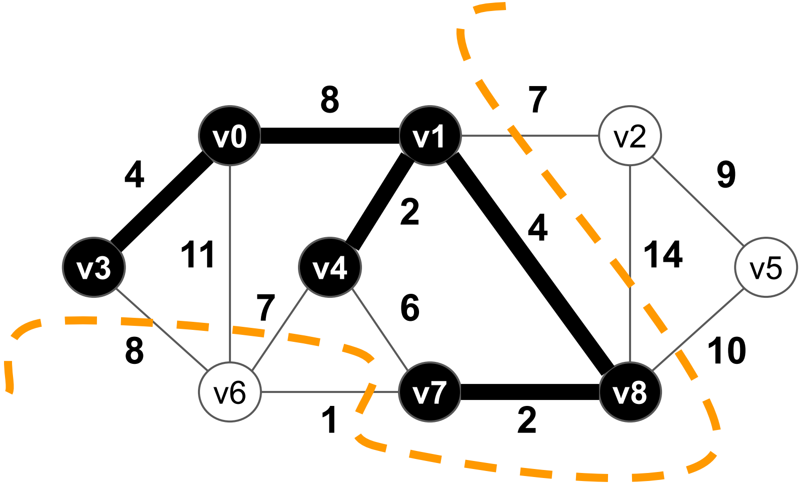
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

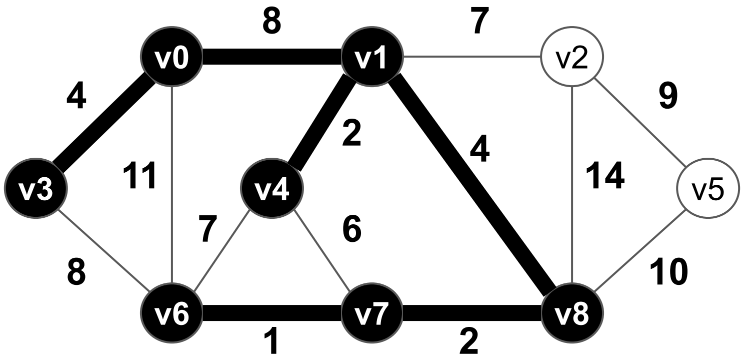
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

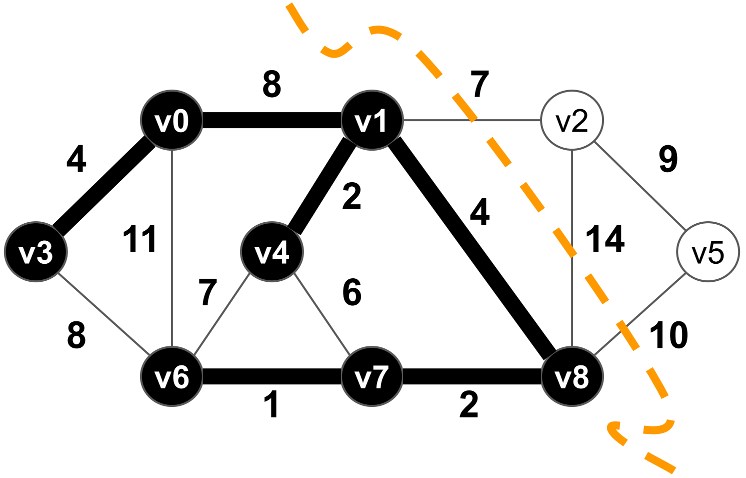
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

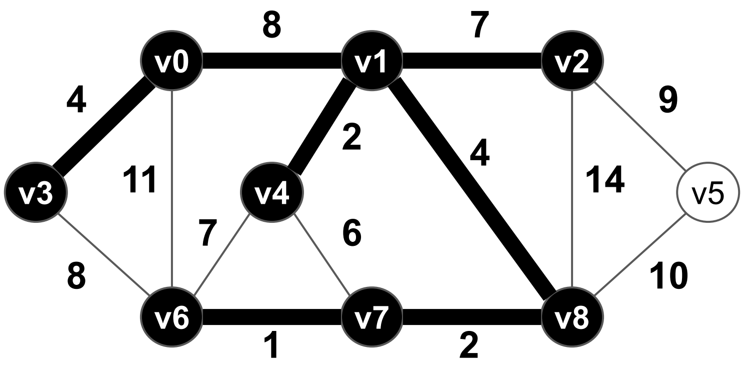
-1 -1 ∞	0 1 8	1 2 7	0 3 4	1 4 2	8 5 10	7 6 1	8 7 2	1 8 4
0	1	2	3	4	5	6	7	8

Algoritmo de Prim - Implementação 2



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

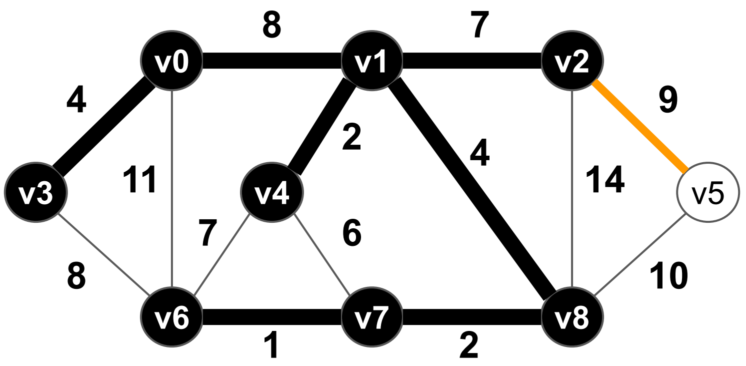
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

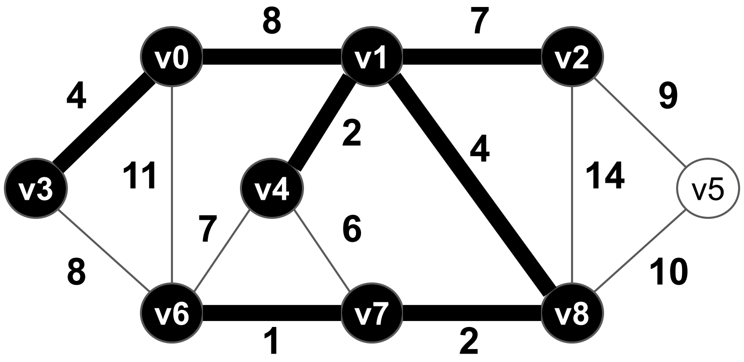
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

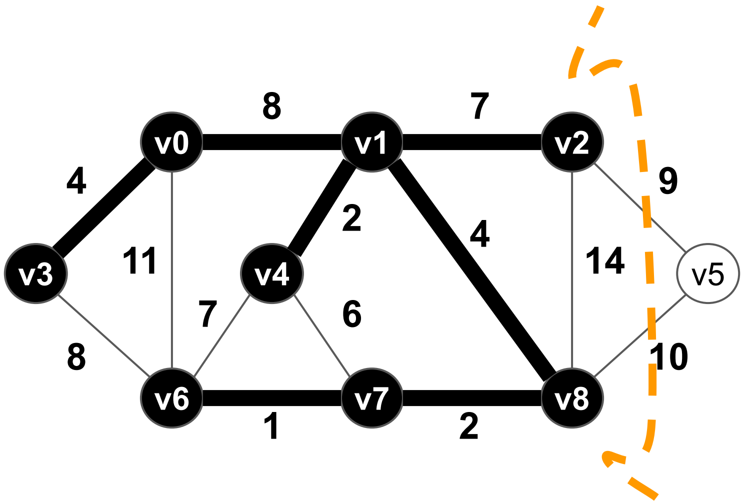
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

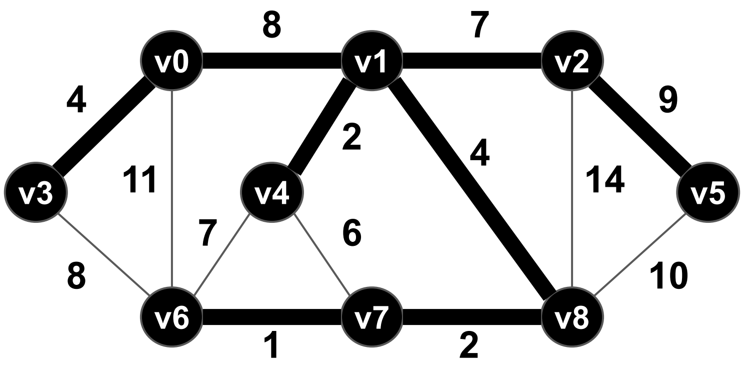
Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

Algoritmo de Prim - Implementação 2



aresta_menor_peso_para_arvore:

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

Algoritmo de Prim - Implementação 2 (é possível melhorar)

Prim(G conexo)

```
Para cada vértice w de G:
    aresta_menor_peso_para_arvore[w] = aresta artificial que tem peso ∞
T = ({ 0 }, ∅)
v = 0
Enquanto T não é uma árvore geradora de G:
    Para cada vizinho w de v que não está em T:
        Se o peso da aresta vw é menor que o peso da aresta_menor_peso_para_arvore[w]:
            aresta_menor_peso_para_arvore[w] = vw
    Inicialize o peso de uv como infinito
    Para cada vértice w de G que não está em T:
        Se o peso da aresta_menor_peso_para_arvore[w] é menor que o peso de uv:
            uv = aresta_menor_peso_para_arvore[w]
    Adicione uv a T
Retorne T
```

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

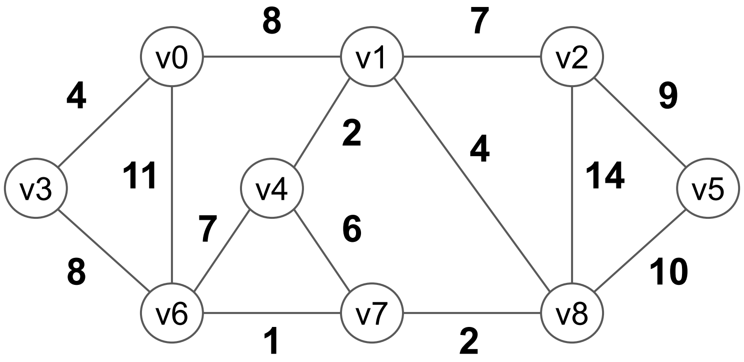
Algoritmo de Prim - Implementação 3

- Nesta implementação, vamos usar uma fila de prioridade 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*

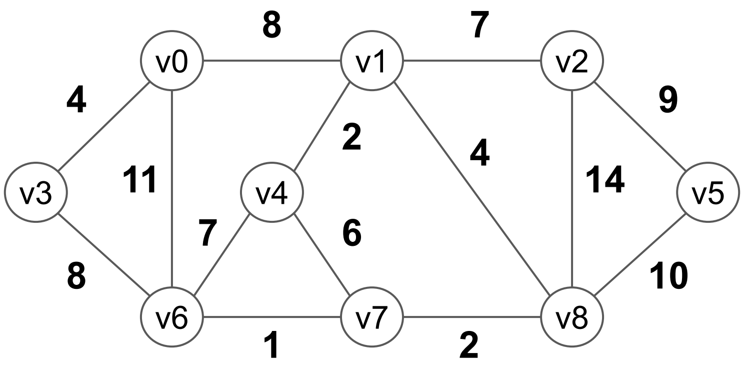
Algoritmo de Prim - Implementação 3

- 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 menor peso 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

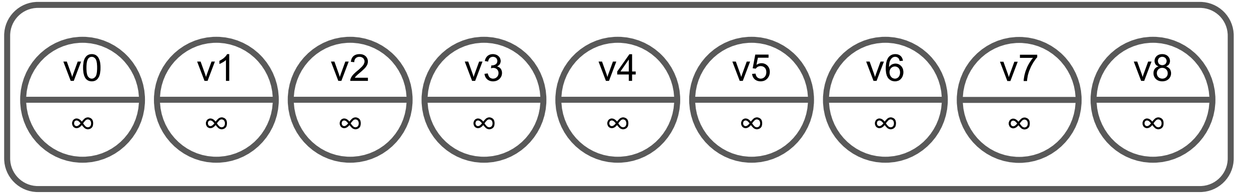
Algoritmo de Prim - Implementação 3



Algoritmo de Prim - Implementação 3



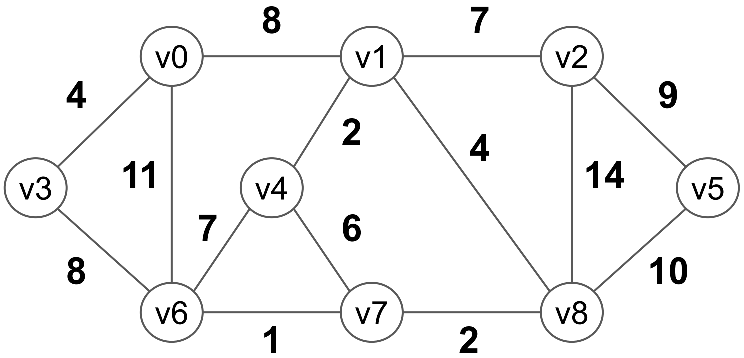
Fila de prioridade:



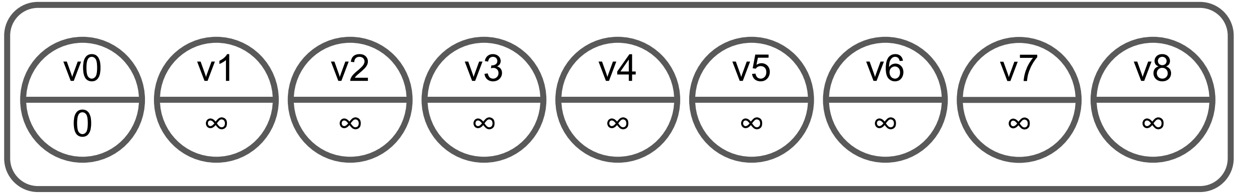
pai:

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

Algoritmo de Prim - Implementação 3



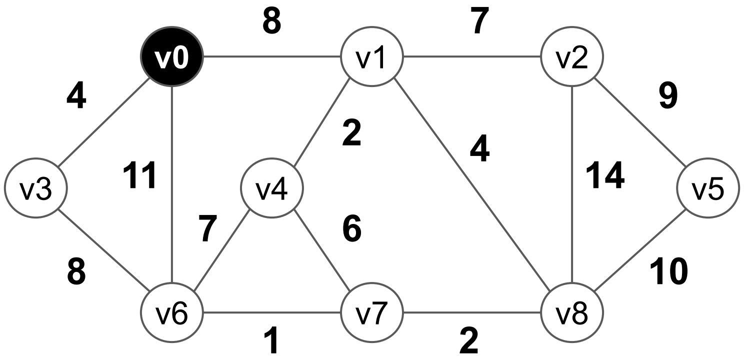
Fila de prioridade:



pai:

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

Algoritmo de Prim - Implementação 3



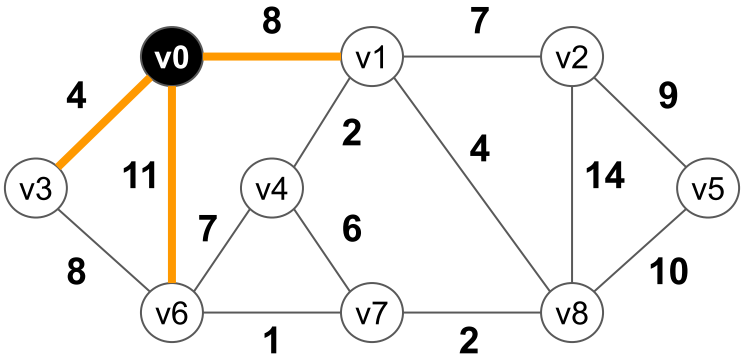
Fila de prioridade:

v1	v2	v3	v4	v5	v6	v7	v8
∞	∞	∞	∞	∞	∞	∞	∞

pai:

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

Algoritmo de Prim - Implementação 3



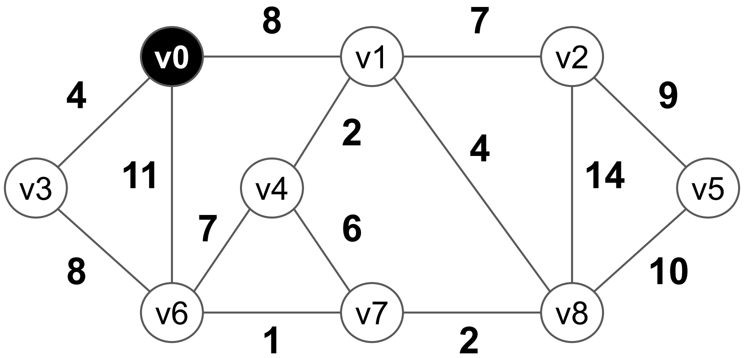
Fila de prioridade:

v1	v2	v3	v4	v5	v6	v7	v8
8	∞	4	∞	∞	11	∞	∞

pai:

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

Algoritmo de Prim - Implementação 3



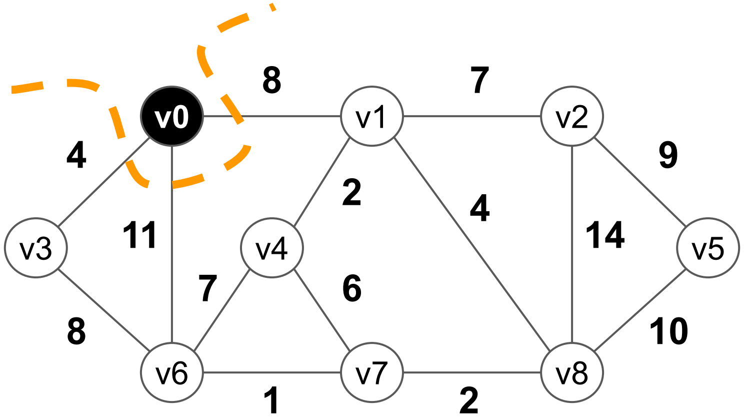
Fila de prioridade:

v1	v2	v3	v4	v5	v6	v7	v8
8	∞	4	∞	∞	11	∞	∞

pai:

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

Algoritmo de Prim - Implementação 3



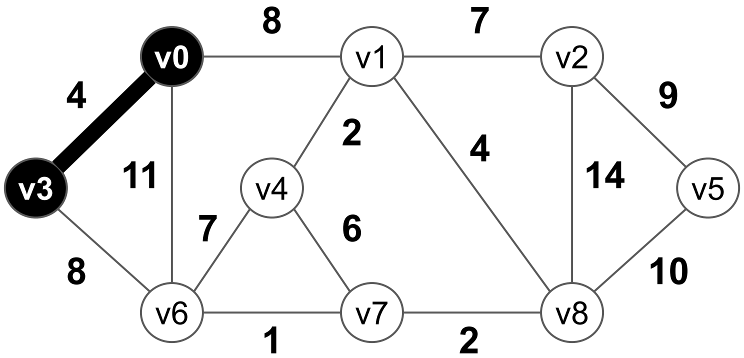
Fila de prioridade:

v1	v2	v3	v4	v5	v6	v7	v8
8	∞	4	∞	∞	11	∞	∞

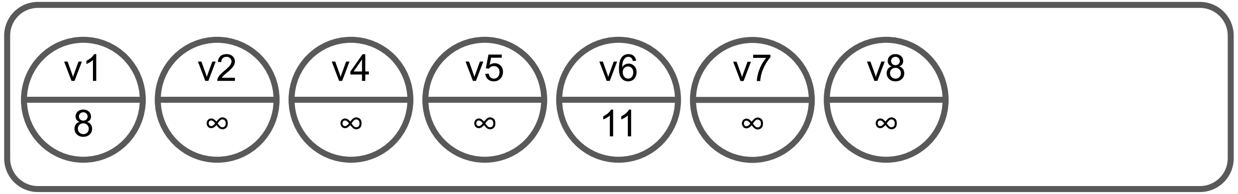
pai:

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

Algoritmo de Prim - Implementação 3



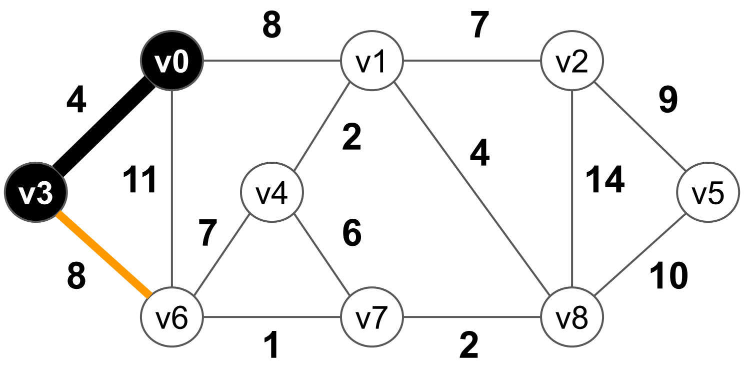
Fila de prioridade:



pai:

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

Algoritmo de Prim - Implementação 3



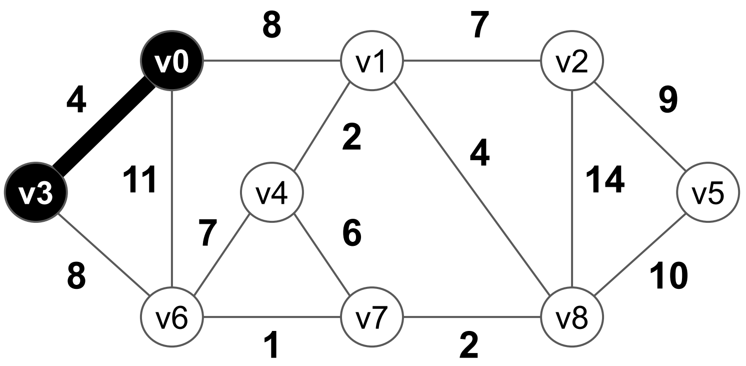
Fila de prioridade:

v1	v2	v4	v5	v6	v7	v8
8	∞	∞	∞	8	∞	∞

pai:

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

Algoritmo de Prim - Implementação 3



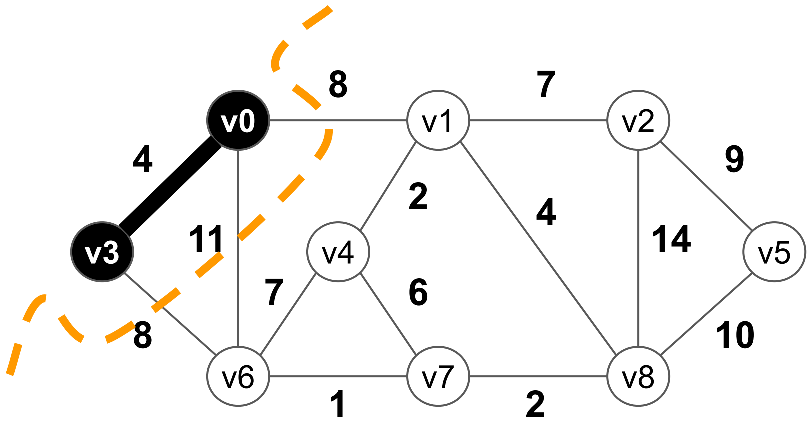
Fila de prioridade:

v1	v2	v4	v5	v6	v7	v8
8	∞	∞	∞	8	∞	∞

pai:

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

Algoritmo de Prim - Implementação 3



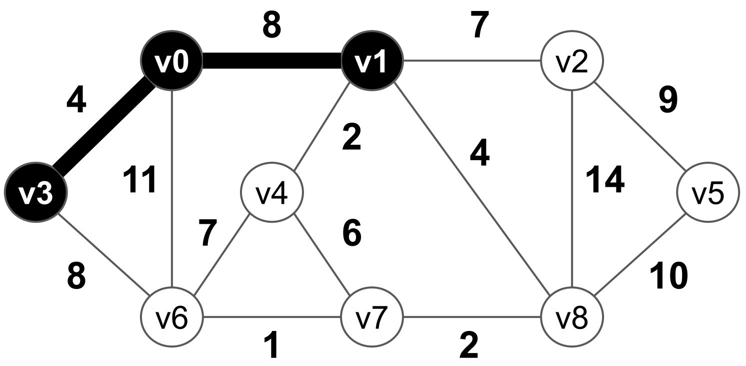
Fila de prioridade:

v1	v2	v4	v5	v6	v7	v8
8	∞	∞	∞	8	∞	∞

pai:

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

Algoritmo de Prim - Implementação 3



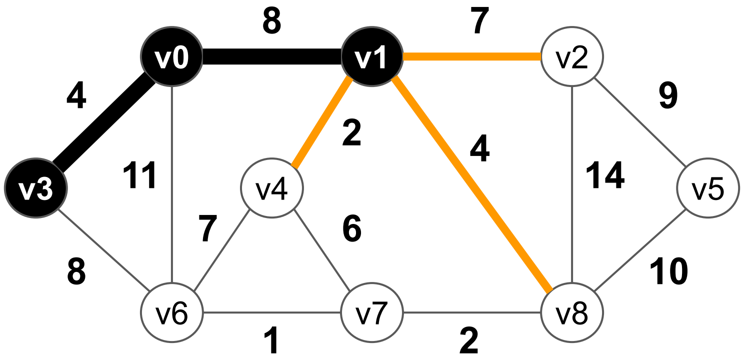
Fila de prioridade:

v2	v4	v5	v6	v7	v8
∞	∞	∞	8	∞	∞

pai:

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

Algoritmo de Prim - Implementação 3



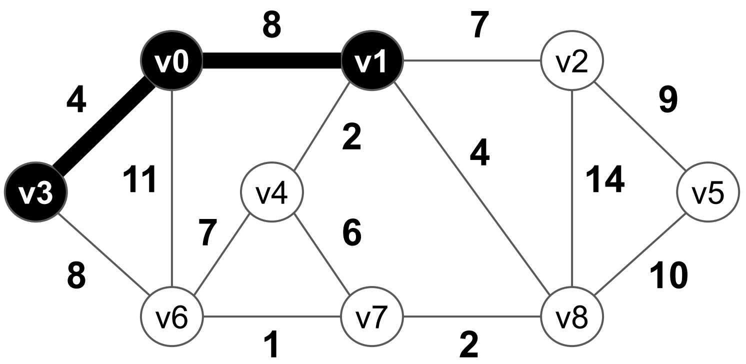
Fila de prioridade:

v2	v4	v5	v6	v7	v8
7	2	∞	8	∞	4

pai:

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

Algoritmo de Prim - Implementação 3



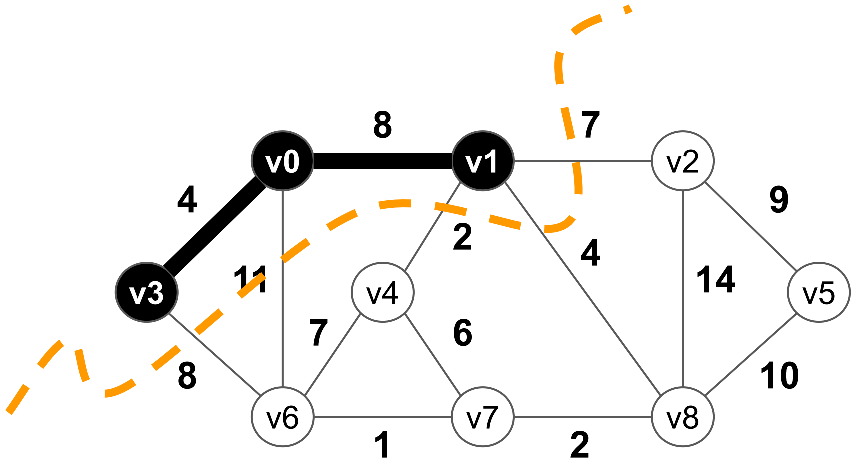
Fila de prioridade:

v2	v4	v5	v6	v7	v8
7	2	∞	8	∞	4

pai:

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

Algoritmo de Prim - Implementação 3



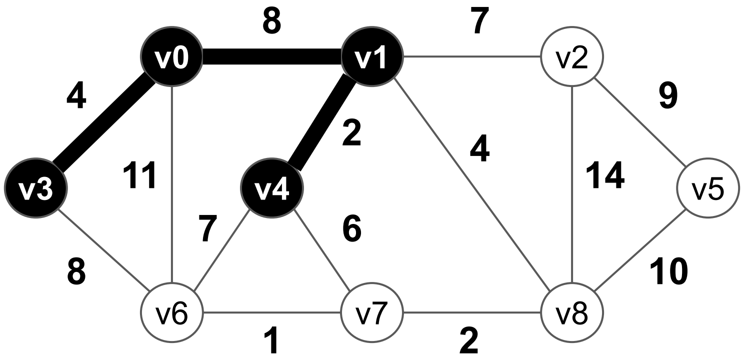
Fila de prioridade:

v2	v4	v5	v6	v7	v8
7	2	∞	8	∞	4

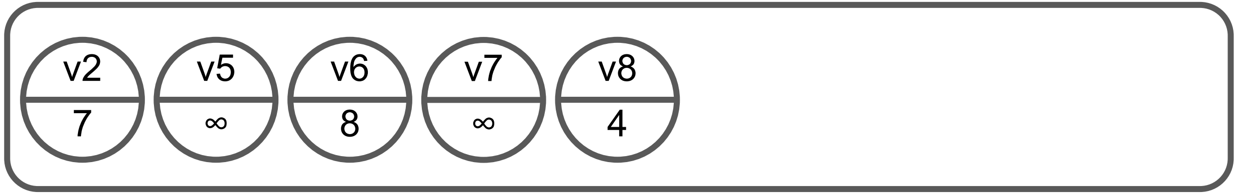
pai:

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

Algoritmo de Prim - Implementação 3



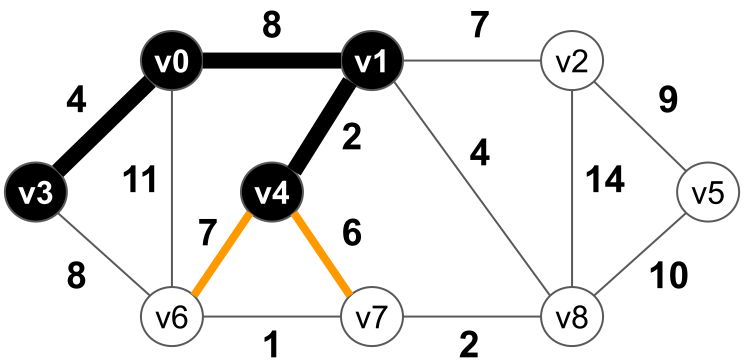
Fila de prioridade:



pai:

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

Algoritmo de Prim - Implementação 3



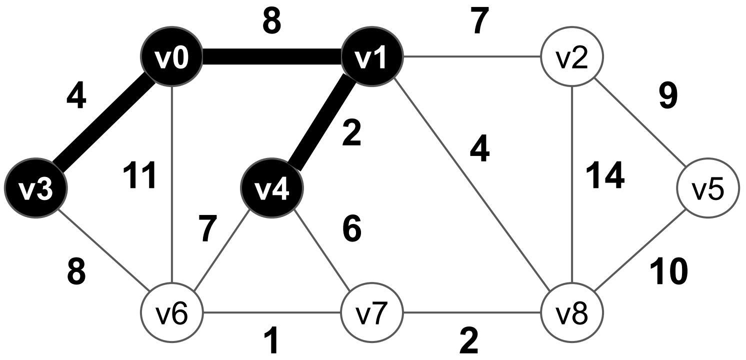
Fila de prioridade:

v2	v5	v6	v7	v8
7	∞	7	6	4

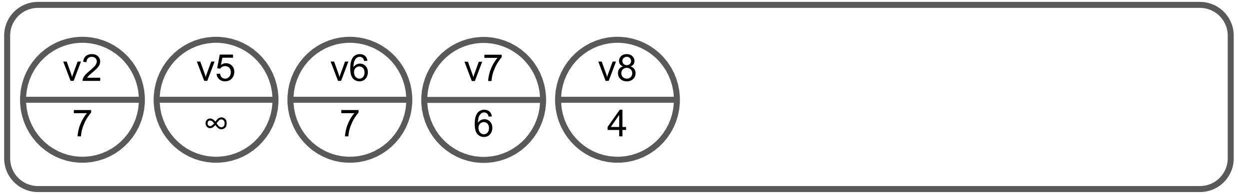
pai:

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

Algoritmo de Prim - Implementação 3



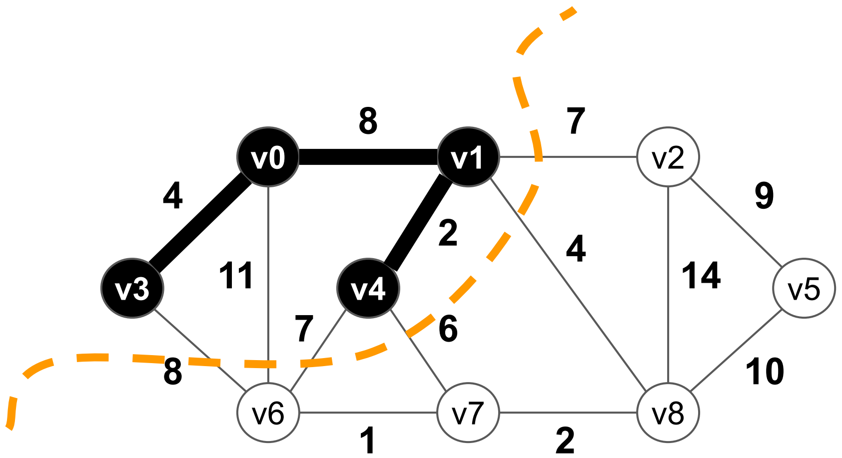
Fila de prioridade:



pai:

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

Algoritmo de Prim - Implementação 3



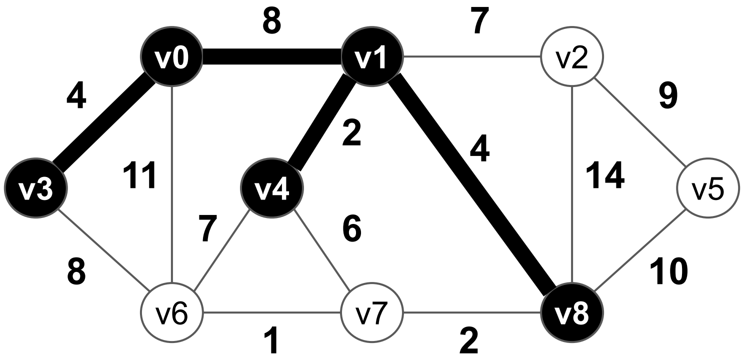
Fila de prioridade:

v2	v5	v6	v7	v8
7	∞	7	6	4

pai:

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

Algoritmo de Prim - Implementação 3



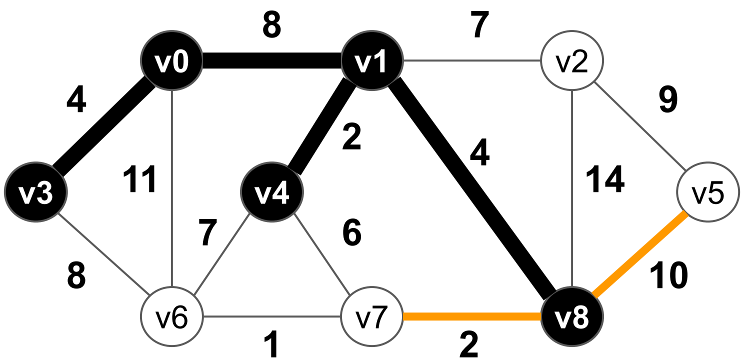
Fila de prioridade:

v2	v5	v6	v7
7	∞	7	6

pai:

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

Algoritmo de Prim - Implementação 3



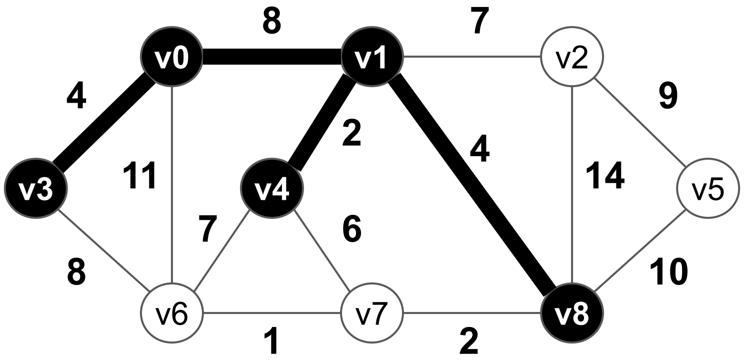
Fila de prioridade:

v2	v5	v6	v7
7	10	7	2

pai:

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

Algoritmo de Prim - Implementação 3



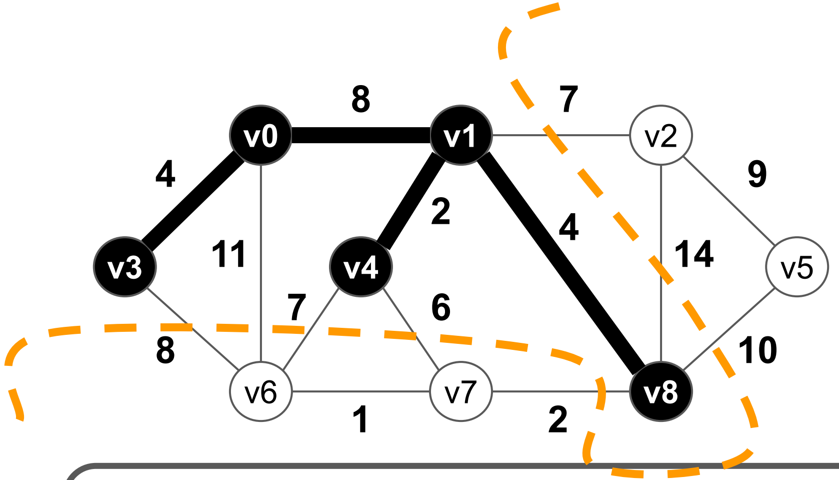
Fila de prioridade:

v2	v5	v6	v7
7	10	7	2

pai:

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

Algoritmo de Prim - Implementação 3



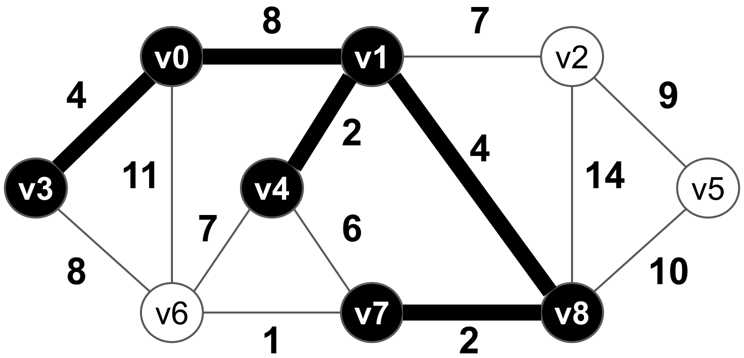
Fila de prioridade:

v2	v5	v6	v7
7	10	7	2

pai:

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

Algoritmo de Prim - Implementação 3



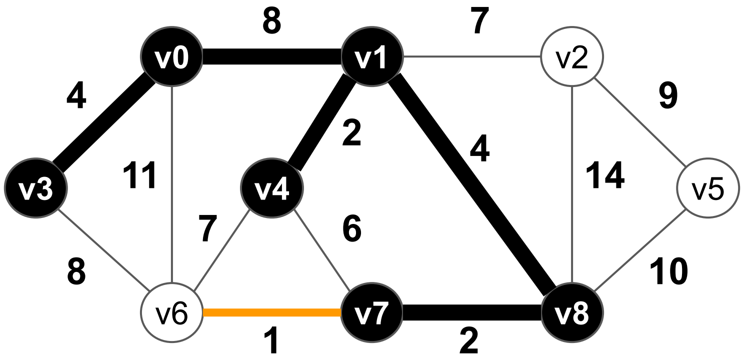
Fila de prioridade:

v2	v5	v6
7	10	7

pai:

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

Algoritmo de Prim - Implementação 3

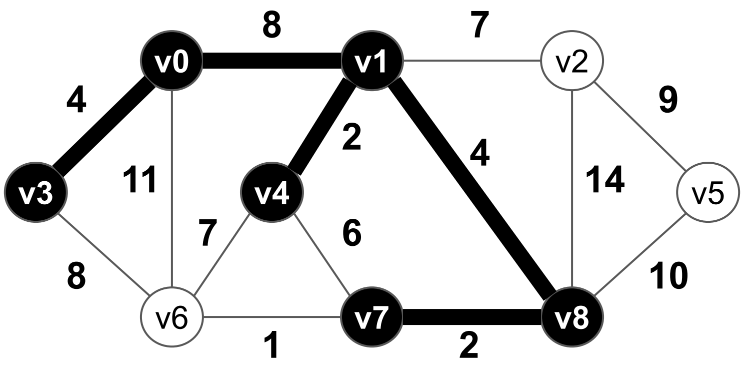


Fila de prioridade:

pai:

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

Algoritmo de Prim - Implementação 3



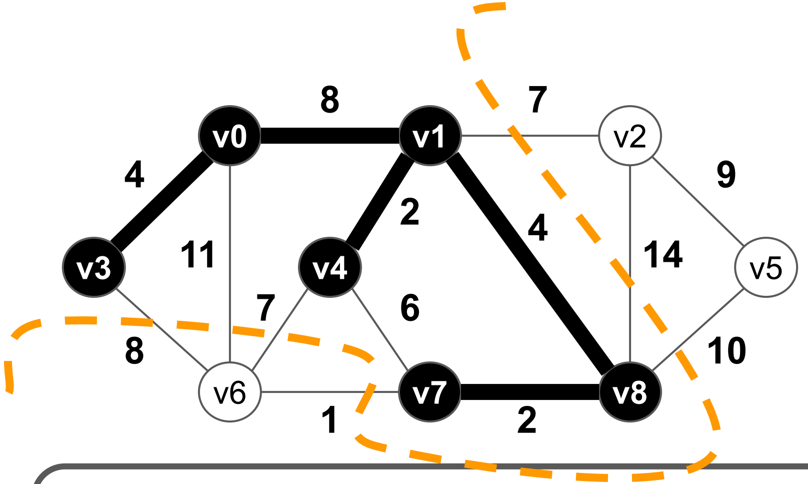
Fila de prioridade:

v2	v5	v6
7	10	1

pai:

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

Algoritmo de Prim - Implementação 3



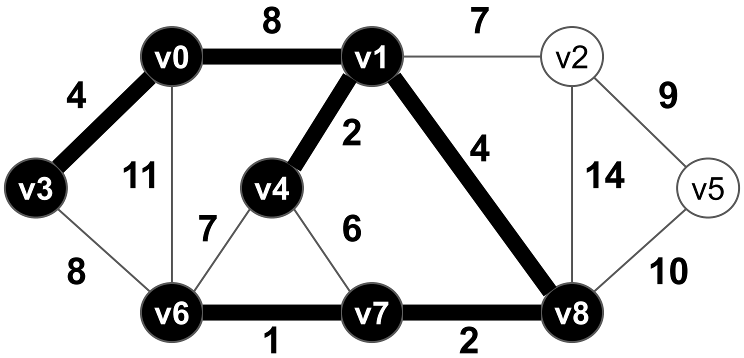
Fila de prioridade:

v2	v5	v6
7	10	1

pai:

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

Algoritmo de Prim - Implementação 3

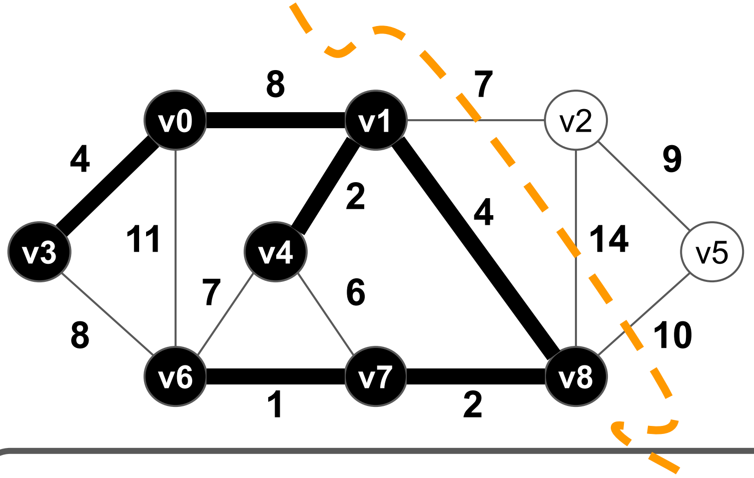


Fila de prioridade:

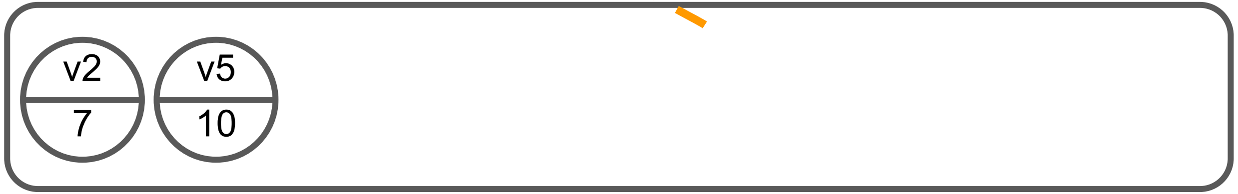
pai:

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

Algoritmo de Prim - Implementação 3



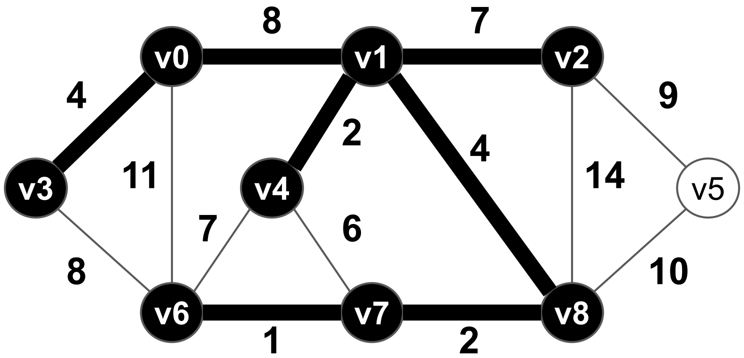
Fila de prioridade:



pai:

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

Algoritmo de Prim - Implementação 3

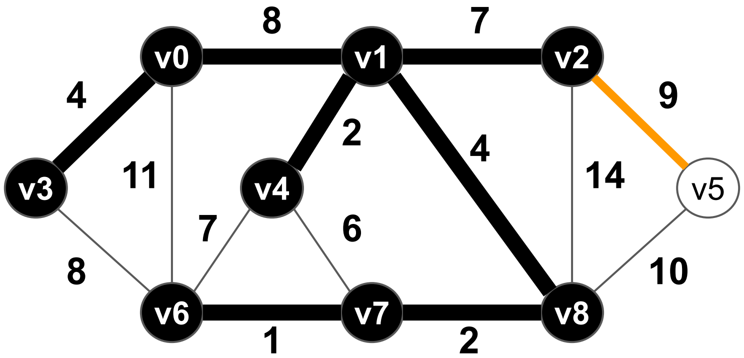


Fila de prioridade:

pai:

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

Algoritmo de Prim - Implementação 3

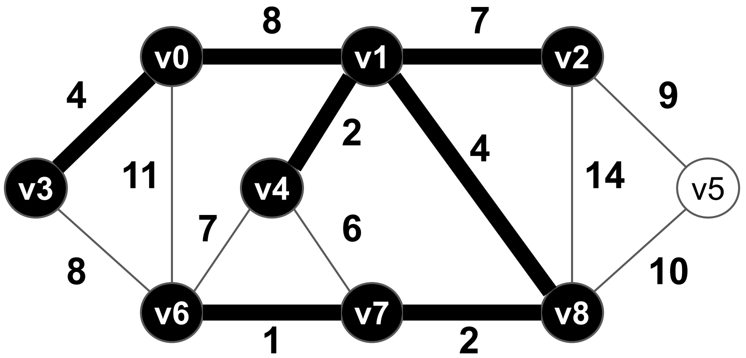


Fila de prioridade:

pai:

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

Algoritmo de Prim - Implementação 3

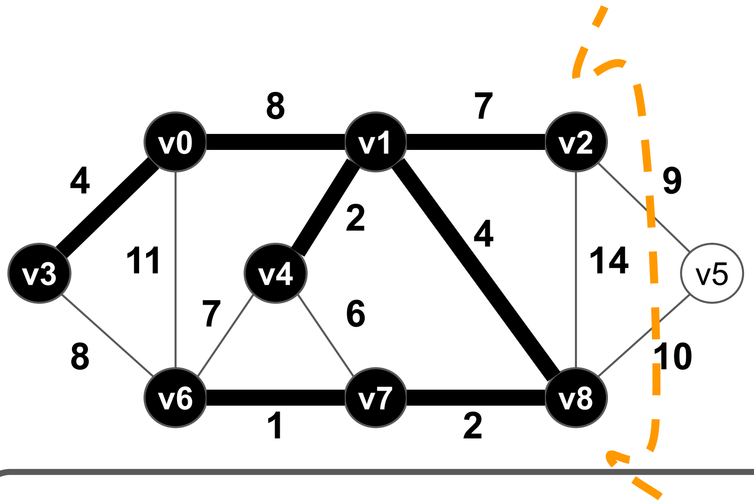


Fila de prioridade:

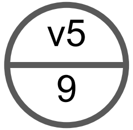
pai:

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

Algoritmo de Prim - Implementação 3



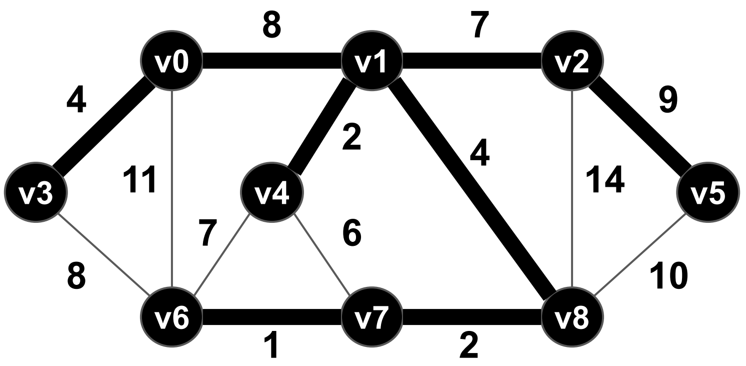
Fila de prioridade:



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:

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

Algoritmo de Prim - Implementação 3

Prim(*G* conexo, *pai*)

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

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

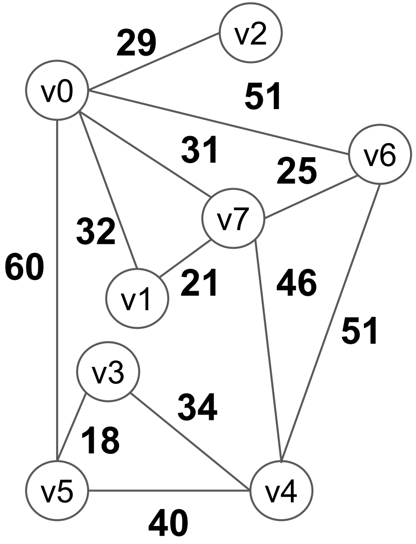
Algoritmo de Prim - Implementação 3

Prim(*G* conexo, *pai*)

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

Exercícios

1. Indique a árvore geradora de peso mínimo retornada pelo Algoritmo de Prim para o grafo abaixo.



Exercícios

2. Considere um grafo conexo *G* com pesos nas arestas. Seja *uv* uma aresta de peso mínimo de G. Prove que existe alguma árvore geradora de peso mínimo de *G* que contém *uv*.

Exercícios

3. Considere o seguinte algoritmo que recebe como entrada um grafo conexo *G*:
   Comece com uma árvore *T* que contém apenas um único vértice de *G*.
   Repita o seguinte passo $V(G)$ - 1 vezes: entre as arestas de *G* incidentes no último vértice adicionado a *T*, encontre a aresta de peso mínimo e adicione esta aresta a *T*. Retorne *T*.

   Este é um algoritmo correto para encontrar uma árvore geradora de peso mínimo de *G*? Se sim, explique por quê. Senão, mostre um exemplo em que o algoritmo não retorna uma árvore geradora de peso mínimo de *G*.

Exercícios

4. Relembre o funcionamento da estrutura de dados fila de prioridade (heap) cuja operação de remoção sempre retorna o item de menor prioridade. Em seguida, implemente o Algoritmo de Prim.

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.