Curso Estruturas de Dados e Algoritmos Expert

Prof. Nelio Alves

Grafos (parte 5)

https://devsuperior.com.br

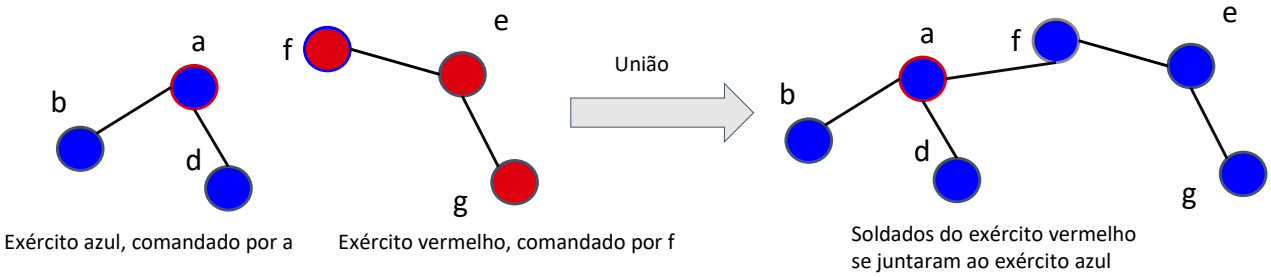
Disjoint Set Union (Union-Find)

https://devsuperior.com.br

Prof. Dr. Nelio Alves

Problema motivador

- Suponha que, em uma região da antiga Idade Média, existam vários lordes, cada um com seu próprio exército pessoal.
- Após estourar um conflito na região, os senhores estão buscando fazer alianças entre si, juntando seus exércitos.
- Porém, após várias alianças, os soldados estão confusos e não sabem mais a que exército pertencem. Como podemos os ajudar a que exército pertencem após todas essas fusões e alianças?



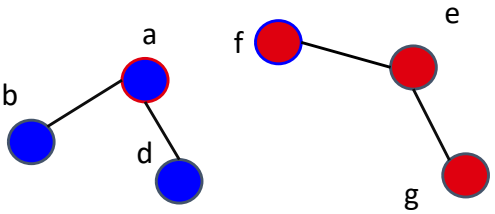
Exército azul, comandado por a

Exército vermelho, comandado por f

Soldados do exército vermelho se juntaram ao exército azul

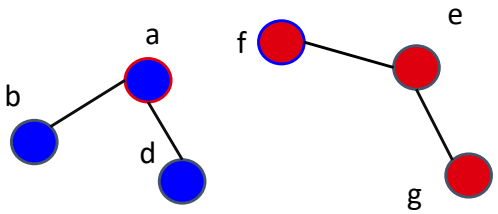
Modelagem do problema

- Soldados são vértices de um grafo
- Soldados conectado por arestas formam um exército
- Exército = componente conexa
- **Questão: a que exército (componente) um soldado pertence?**



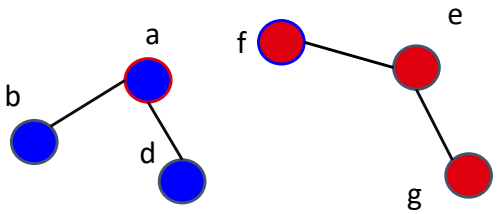
Modelagem do problema

- **Questão: a que exército (componente) um soldado pertence?**
  - Para contar componentes, podemos fazer uso de DFS/BFS
    - Complexidade: O(V + E)
  - Chamamos a busca a partir de cada nó e o marcamos como visitado
    - Descobrimos duas componentes
  - E se adicionarmos uma aresta qualquer entre dois nós?
    - DFS/BFS novamente!



Modelagem do problema

- **Limitações da abordagem com DFS/BFS**
  - Sempre que adicionamos uma nova aresta, precisamos percorrer novamente o grafo para contar as componentes
  - No cenário em que fusões acontecem frequentemente, se torna ineficiente em termos de tempo!
  - Possível solução: **Union-Find**

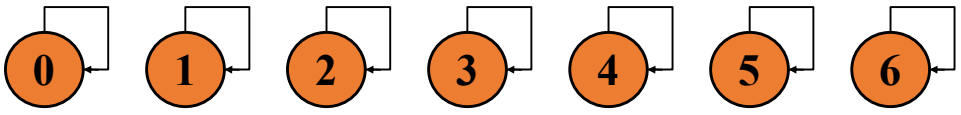


Union-Find

- **Características**
  - **Estrutura de dados para conjuntos disjuntos (sem intersecção)**
    - Representa vários conjuntos onde cada elemento pertence a exatamente um deles
  - **Hierarquia de Pais e Representantes**
    - Cada elemento aponta para um “pai”
    - O representante de um conjunto é aquele que aponta para si mesmo
  - **Operações principais:**
    - Find(u)
      - Acha o representante do conjunto ao qual u pertence
    - Union(u, v)
      - Une os conjuntos que contêm u e v

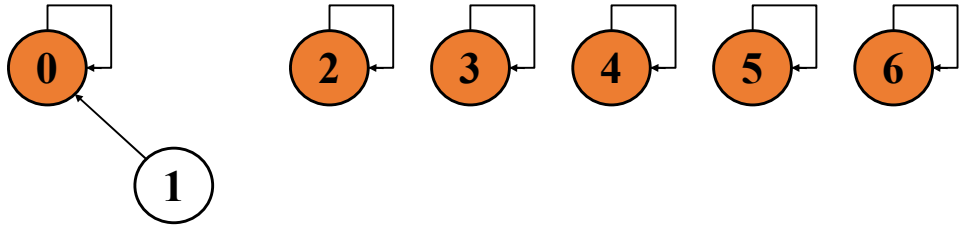
Exemplo

	0	1	2	3	4	5	6
parent	0	1	2	3	4	5	6



Exemplo

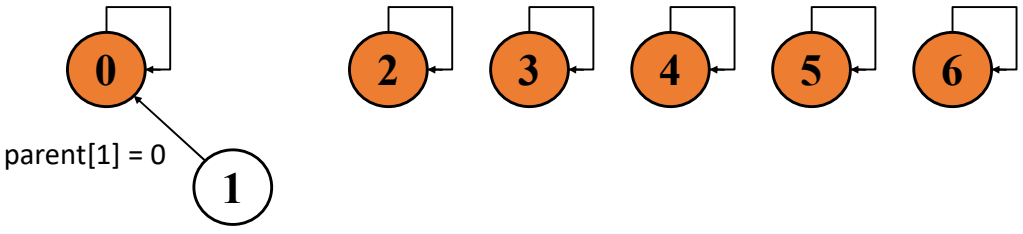
	0	1	2	3	4	5	6
parent	0	**0**	2	3	4	5	6



- union(0, 1)

Exemplo

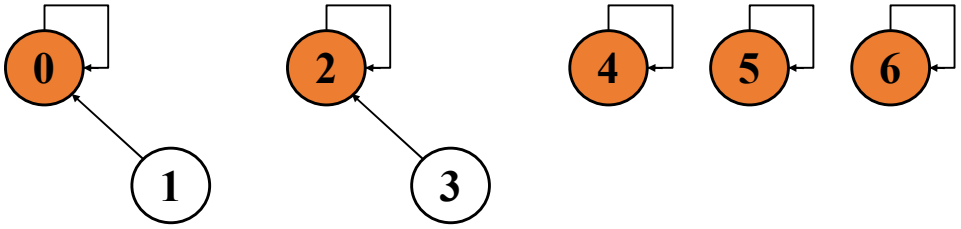
	0	1	2	3	4	5	6
parent	0	**0**	2	3	4	5	6



- find(1) = 0

Exemplo

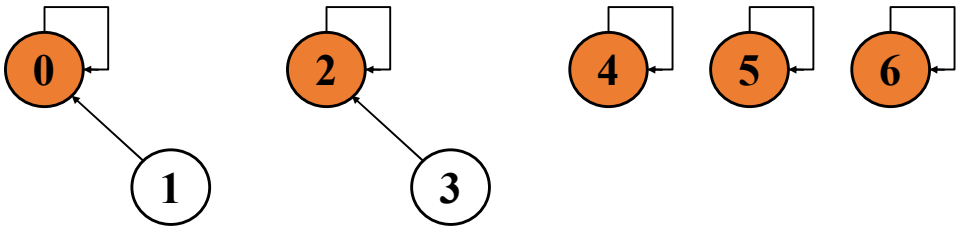
	0	1	2	3	4	5	6
parent	0	0	2	**2**	4	5	6



- union(2, 3)

Exemplo

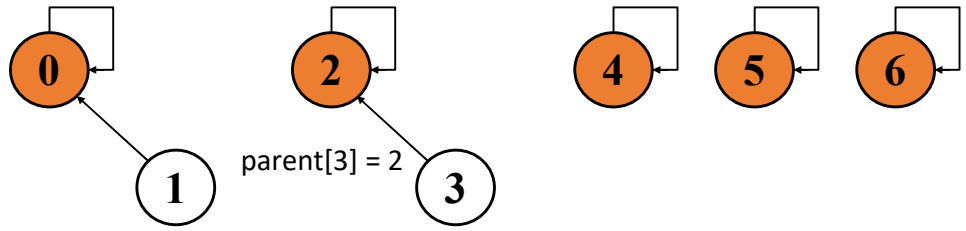
	0	1	2	3	4	5	6
parent	0	0	2	2	4	5	6



- union(3, 4)?

Exemplo

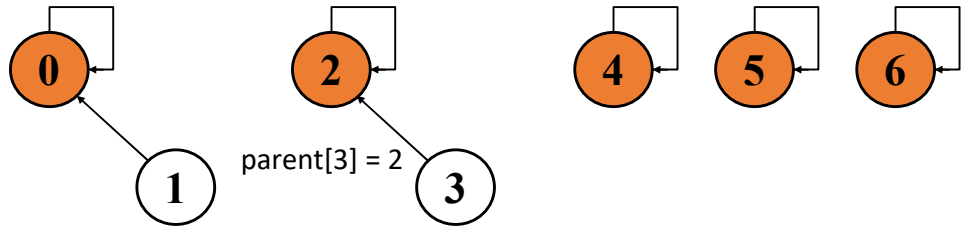
	0	1	2	3	4	5	6
parent	0	0	2	2	4	5	6



- union(3, 4) = union(find(3), find(4))

Exemplo

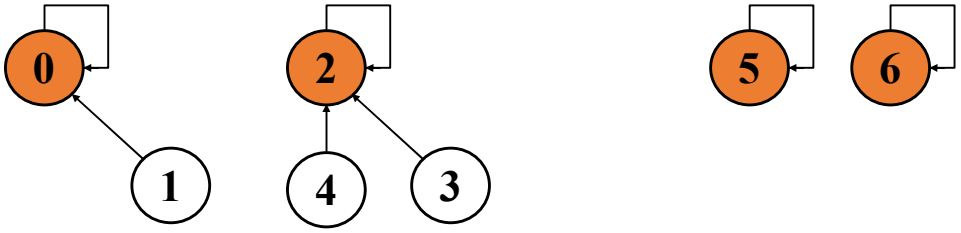
	0	1	2	3	4	5	6
parent	0	0	2	2	4	5	6



- union(3, 4) = union(2, 4)

Exemplo

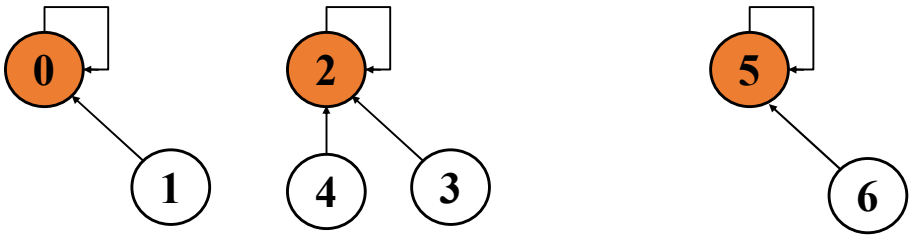
	0	1	2	3	4	5	6
parent	0	0	2	2	**2**	5	6



- union(3, 4) = union(2, 4)

Exemplo

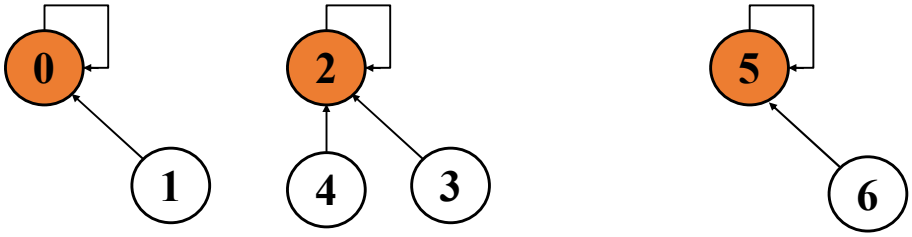
	0	1	2	3	4	5	6
parent	0	0	2	2	2	5	**5**



- union(5, 6)

Exemplo

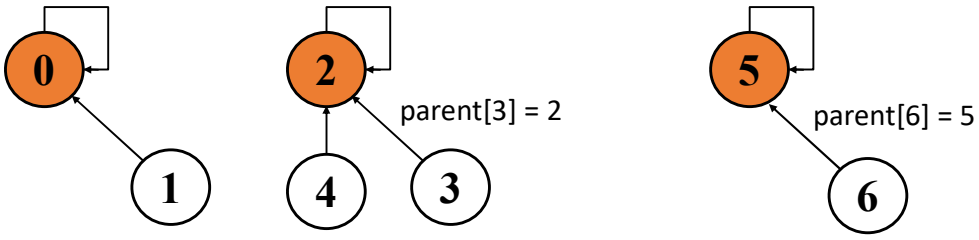
	0	1	2	3	4	5	6
parent	0	0	2	2	2	5	5



- union(3, 6)?

Exemplo

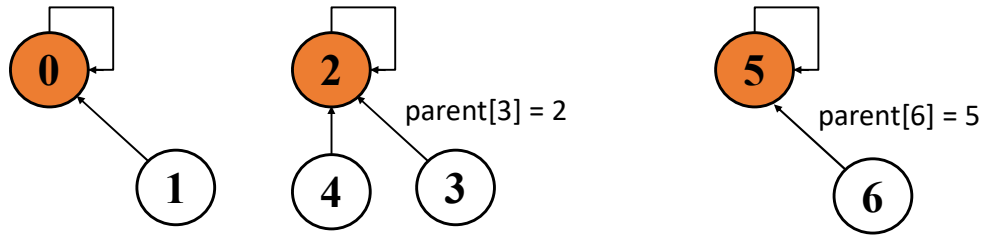
	0	1	2	3	4	5	6
parent	0	0	2	2	2	5	5



- union(3, 6) = union(find(3), find(6))

Exemplo

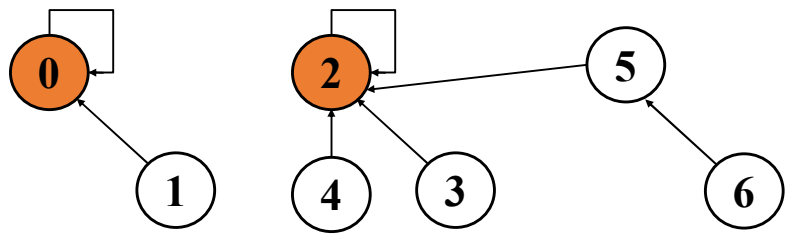
	0	1	2	3	4	5	6
parent	0	0	2	2	2	5	5



- union(3, 6) = union(2, 5)

Exemplo

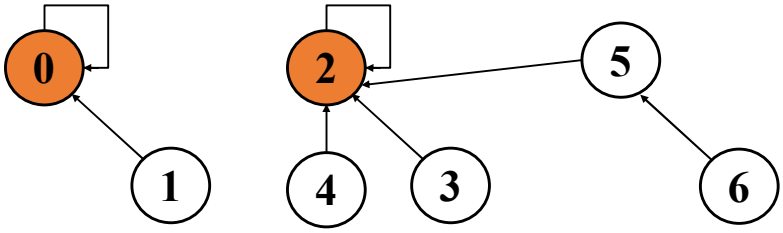
	0	1	2	3	4	5	6
parent	0	0	2	2	2	**2**	5



- union(3, 6) = union(2, 5)

Exemplo

	0	1	2	3	4	5	6
parent	0	0	2	2	2	2	5



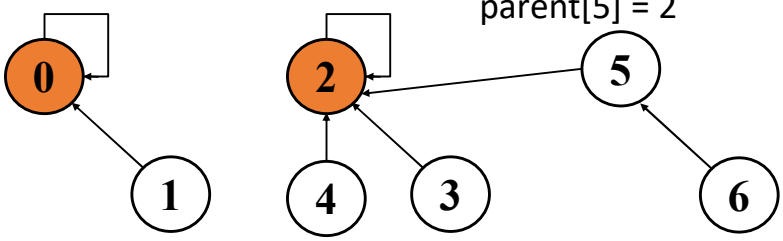
- find(6) = ?

Exemplo

	0	1	2	3	4	5	6
parent	0	0	**2**	2	2	**2**	**5**

parent[2] = 2

parent[5] = 2

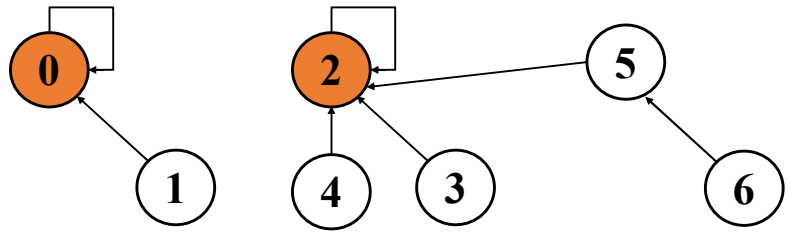


parent[6] = 5

- find(6) = 2

Exemplo

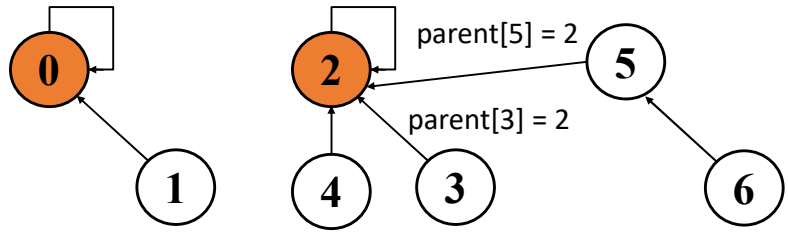
	0	1	2	3	4	5	6
parent	0	0	2	2	2	2	5



- union(5, 3)

Exemplo

	0	1	2	3	4	5	6
parent	0	0	2	2	2	2	5

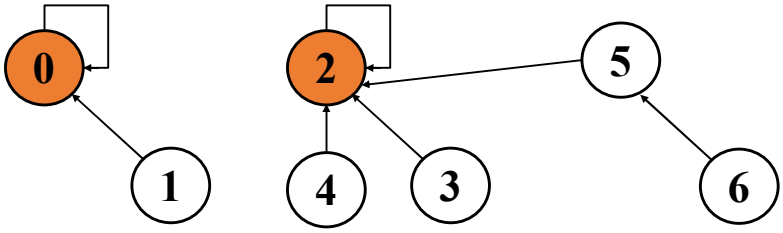


- union(5, 3) = union(find(5), find(3))

Nada acontece, pois já estão unidos!

Exemplo

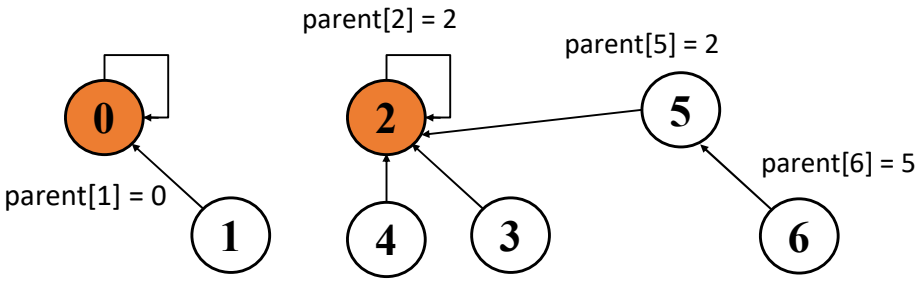
	0	1	2	3	4	5	6
parent	0	0	2	2	2	2	5



- union(6, 1)

Exemplo

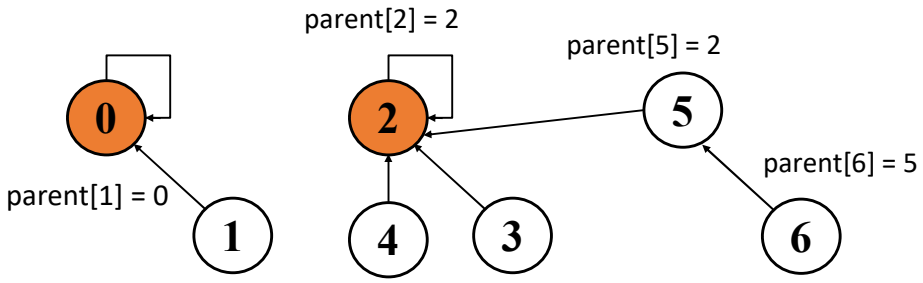
	0	1	2	3	4	5	6
parent	0	0	2	2	2	2	5



- union(6, 1) = union(find(6), find(1))

Exemplo

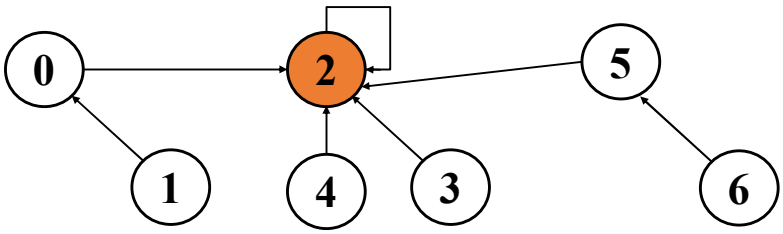
	0	1	2	3	4	5	6
parent	0	0	2	2	2	2	5



- union(6, 1) = union(2, 0)

Exemplo

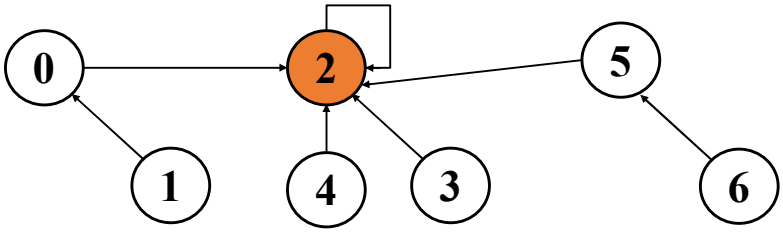
	0	1	2	3	4	5	6
parent	**2**	0	2	2	2	2	5



- union(6, 1) = union(2, 0)

Exemplo

	0	1	2	3	4	5	6
parent	2	0	2	2	2	2	5



Operações do Union-Find

- **Operação Find**
  - Para achar a que componente um elemento pertence, percorre pelos pais até achar um laço (nó que é pai de si mesmo)

- **Operação Union**
  - Para unir dois elementos precisamos achar a raíz de cada componente, e se as raízes forem diferentes, torne uma o pai da outra

Como resolver o problema inicial?

- **Questão: a que exército (componente) um soldado pertence?**
  - Operação Find!

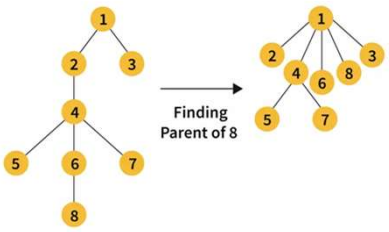
- **Por que é uma solução melhor que BFS/DFS?**
  - **Operações de União e Busca**
    - Quando bem implementadas, sua complexidade é O(α(n)), onde α é a função inversa de Ackermann (cresce muito devagar!)

  - **Adição de Arestas Dinâmicas**
    - Não precisa recalcular toda a conectividade depois de inserir aresta

Heurísticas de otimização

- **Podemos melhorar a estrutura adicionando algumas modificações**

  - Path Compression
    - Aumenta a eficiência do find ao reduzir a altura das árvores que representam os conjuntos



  - Union by Rank
    - Minimizar a altura das árvores ao unir dois conjuntos
      - A árvore de menor altura é anexada à árvore maior

Algoritmos de Árvore Geradora Mínima

https://devsuperior.com.br

Prof. Dr. Nelio Alves

Problema motivador

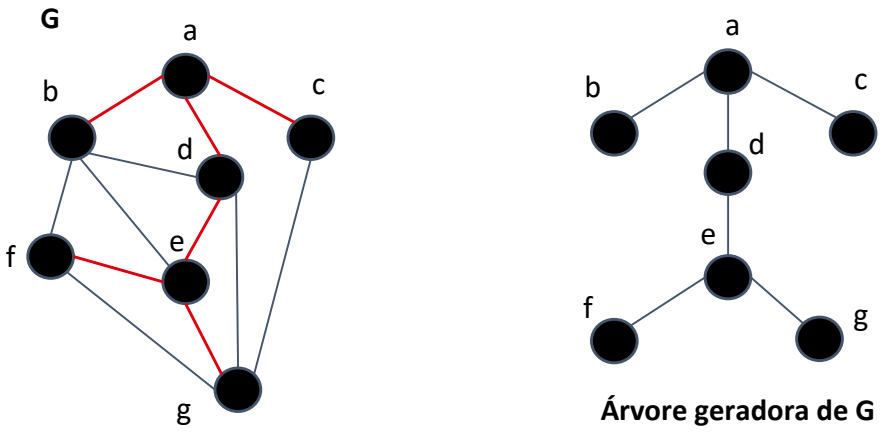
- Suponha que um prefeito precisa realizar obras na cidade e quer construir viadutos conectando diferentes bairros
- Foi feito um levantamento de possíveis viadutos (arestas), com seus respectivos custos de construção, ligando sempre dois bairros (vértices) diferentes
- Como escolher que viadutos construir, de forma que ainda seja possível chegar de um bairro a qualquer outro, e o custo das obras seja mínimo?

A solução envolve encontrar a árvore geradora mínima!

Definição

Árvore geradora: subgrafo gerador que é uma árvore

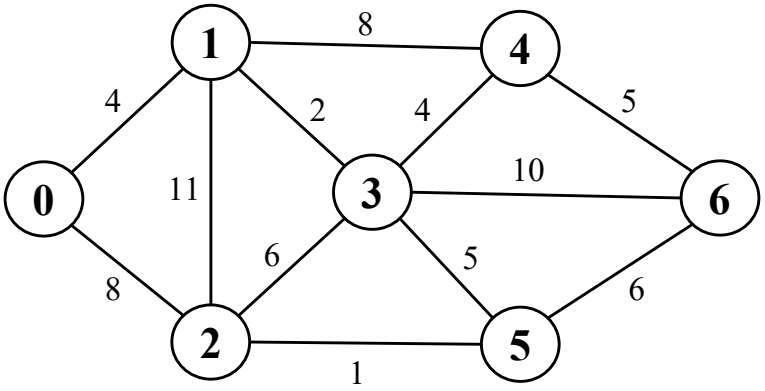
- Ou seja, podemos retirar apenas arestas de G, formando uma árvore



Árvore geradora de G

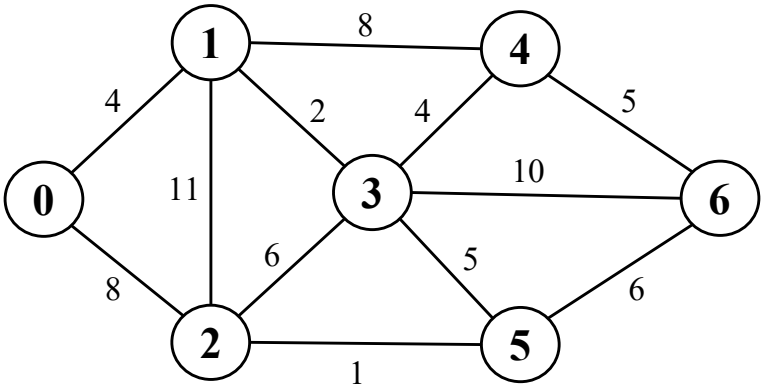
Árvore Geradora Mínima / Minimum Spanning Tree (MST)

- Em um grafo ponderado, é a sub-árvore geradora cuja soma das arestas seja a menor possível



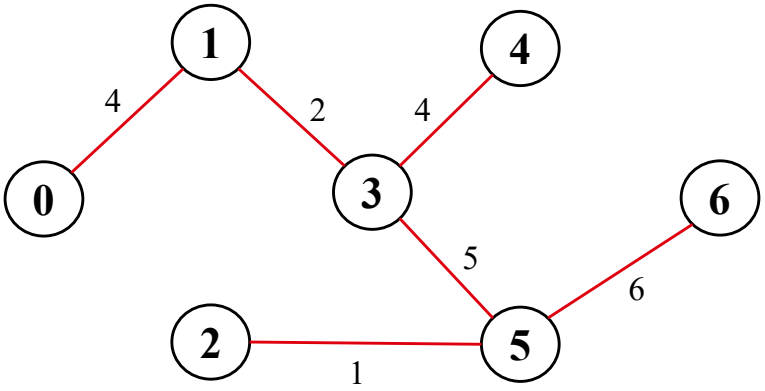
Árvore Geradora Mínima / Minimum Spanning Tree (MST)

- Como retirar arestas, de forma que o grafo permaneça conectado, sem ciclos, e que a soma das arestas seja a menor possível?



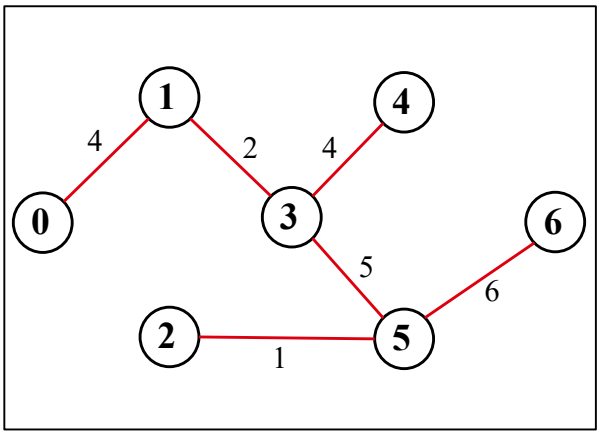
Árvore Geradora Mínima / Minimum Spanning Tree (MST)

Custo da árvore mínima: 22

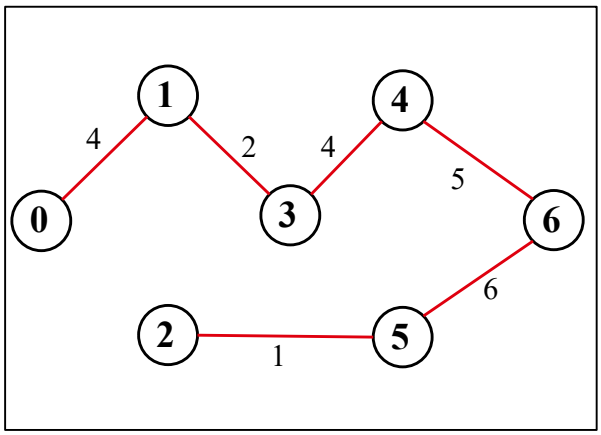


Árvore Geradora Mínima / Minimum Spanning Tree (MST)

Obs: a árvore geradora mínima pode não ser única



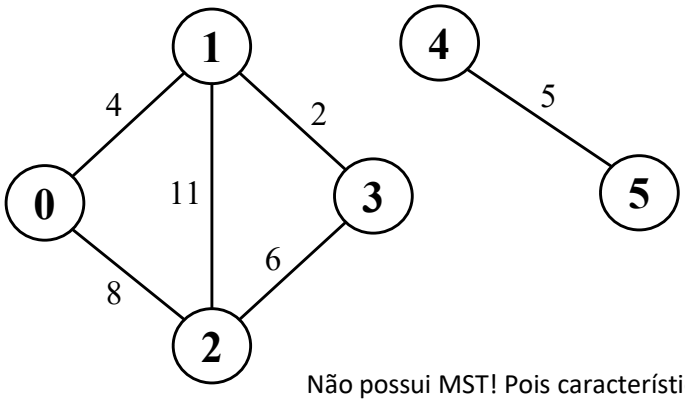
Custo: 22



Custo: 22

Árvore Geradora Mínima / Minimum Spanning Tree (MST)

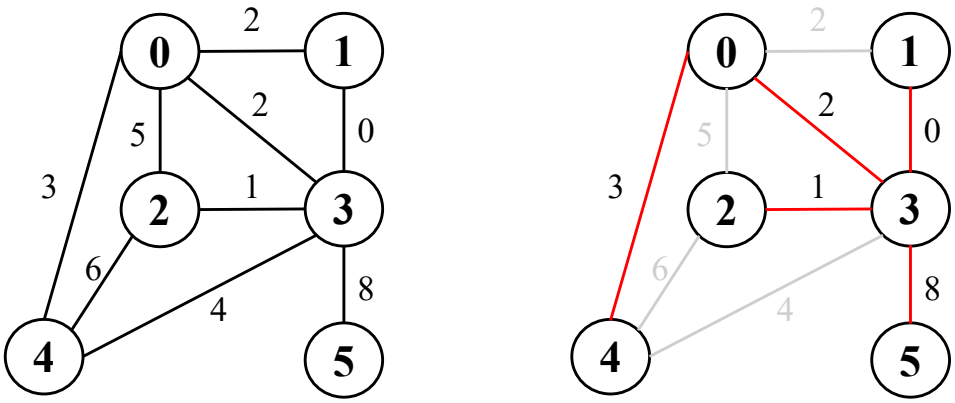
Obs: o grafo original precisa ser conectado para que exista uma MST



Não possui MST! Pois característica da árvore é ser conexa.

Exercício

Ache qualquer Árvore Geradora Mínima



Custo da MST: 3 + 2 + 1 + 0 + 8 = 14

Árvore Geradora Mínima / Minimum Spanning Tree (MST)

- Que algoritmos resolvem esse problemas?
  - **Algoritmo de Prim**
  - **Algoritmo de Kruskal**

- Abordagens diferentes para selecionar “arestas seguras” a se adicionar na solução
  - Prim
    - Fila de prioridade
  - Kruskal
    - Disjoint Set Union (DSU)

Algoritmo de Prim

https://devsuperior.com.br

Prof. Dr. Nelio Alves

Algoritmo de Prim

Características

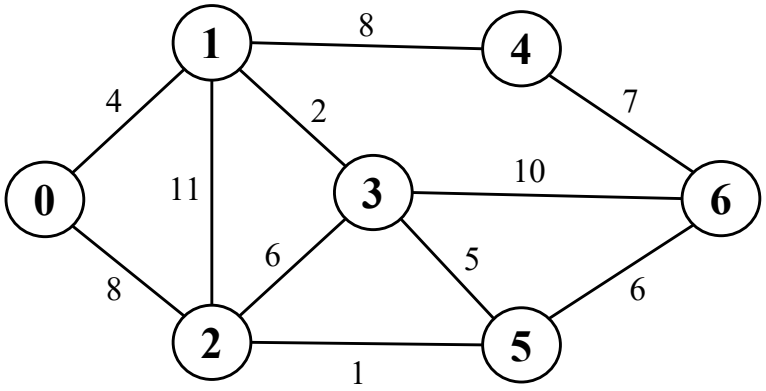
- Procura pela árvore geradora mínima a partir de um vértice de origem
- Utiliza fila de prioridade para determinar que vértice visitar
  - Arestas de menor peso primeiro
    - Algoritmo Guloso!

Estratégia

1. Começa a partir de um nó inicial s e o marca como visitado
2. Itera por todas as arestas de s e as adiciona na fila
3. Extrai a aresta de menor custo e a adiciona na resposta, se for válida
4. Visita vértice dessa aresta, marca como visitado e adiciona arestas que não apontam para ninguém já visitado
5. Volta ao passo 3 até a fila estar vazia

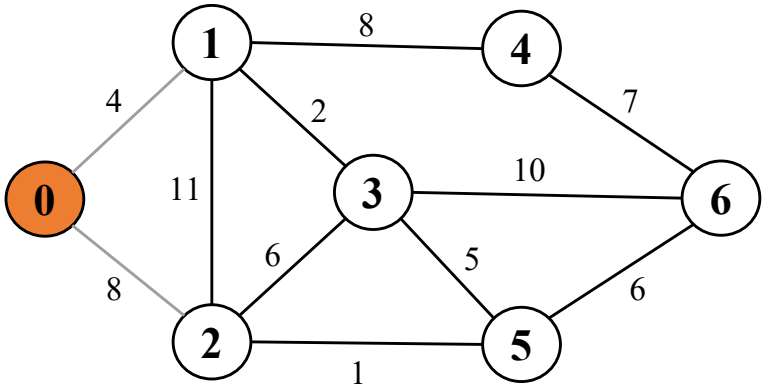
Algoritmo de Prim

- Começando a partir de 0, adicionaremos à resposta sempre a menor aresta acessível a partir dos vértices atuais



Algoritmo de Prim

- Adicionaremos à resposta sempre a menor aresta acessível a partir dos vértices atuais

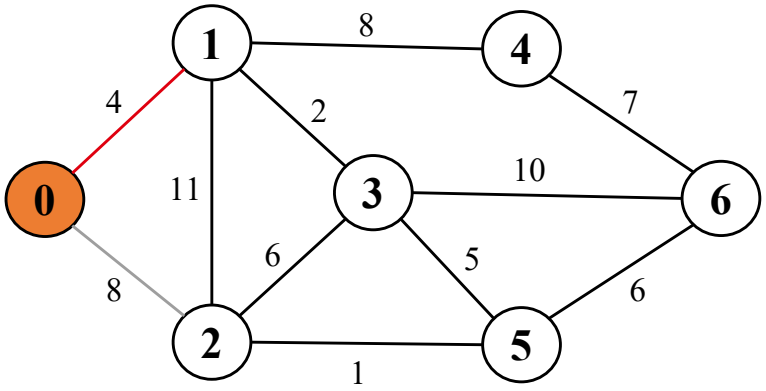


Fila de Prioridade

(0, 2, 8)
(0, 1, 4)

Algoritmo de Prim

- Menor aresta acessível é (0, 1, 4), a adiciona na resposta



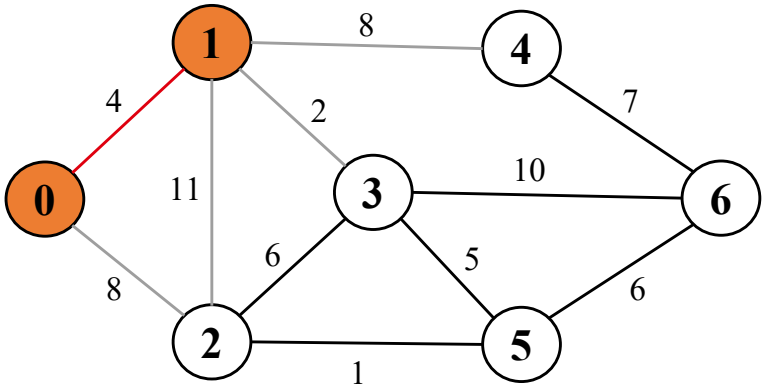
Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~

47

Algoritmo de Prim

- Vai ao vértice 1, o marca e adiciona suas arestas válidas à fila



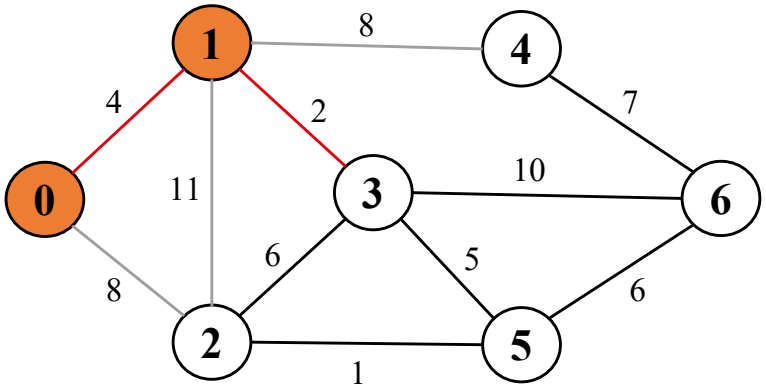
Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
(1, 3, 2)
(1, 2, 11)

48

Algoritmo de Prim

- Menor aresta acessível é (1, 3, 2), a adiciona na resposta

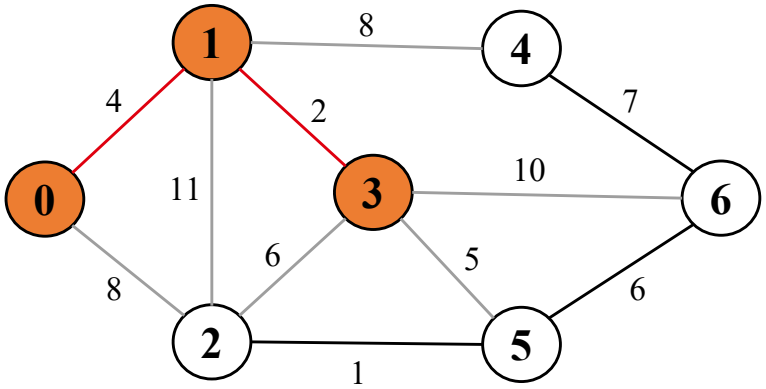


Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
~~(1, 3, 2)~~
(1, 2, 11)

Algoritmo de Prim

- Vai ao vértice 3, o marca e adiciona suas arestas válidas à fila

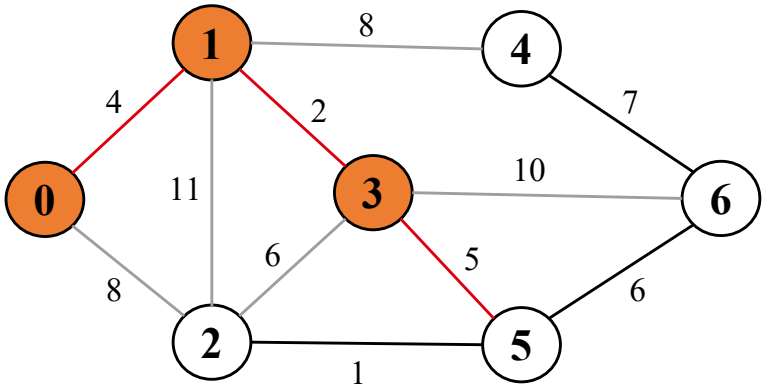


Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
~~(1, 3, 2)~~
(1, 2, 11)
(3, 2, 6)
(3, 5, 5)
(3, 6, 10)

Algoritmo de Prim

- Menor aresta acessível é (3, 5, 5), a adiciona na resposta

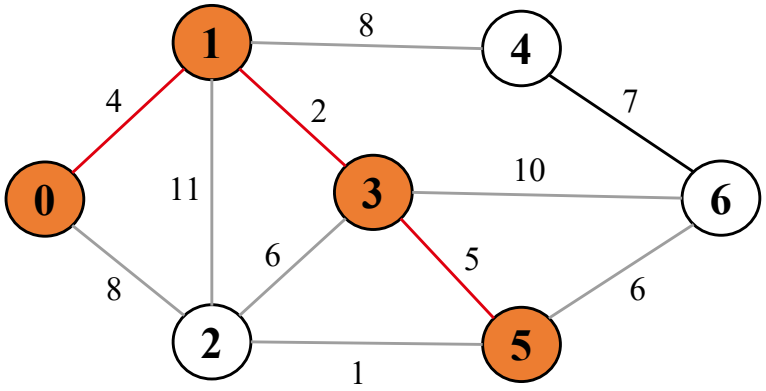


Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
~~(1, 3, 2)~~
(1, 2, 11)
(3, 2, 6)
~~(3, 5, 5)~~
(3, 6, 10)

Algoritmo de Prim

- Vai ao vértice 5, o marca e adiciona suas arestas válidas à fila

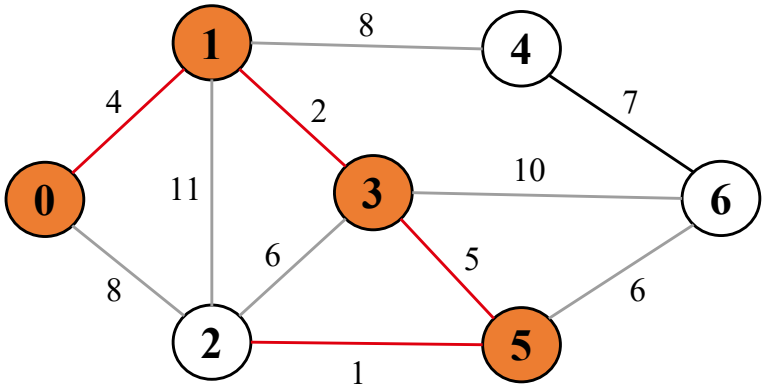


Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
~~(1, 3, 2)~~
(1, 2, 11)
(3, 2, 6)
~~(3, 5, 5)~~
(3, 6, 10)
(5, 2, 1)
(5, 6, 6)

Algoritmo de Prim

- Menor aresta acessível é (5, 2, 1), a adiciona na resposta



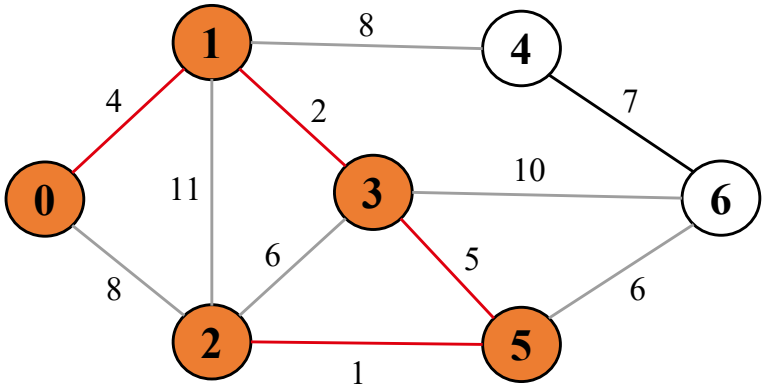
Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
~~(1, 3, 2)~~
(1, 2, 11)
(3, 2, 6)
~~(3, 5, 5)~~
(3, 6, 10)
~~(5, 2, 1)~~
(5, 6, 6)

53

Algoritmo de Prim

- Vai ao vértice 2, todas suas arestas levam a nós já visitados, não adicionamos ninguém



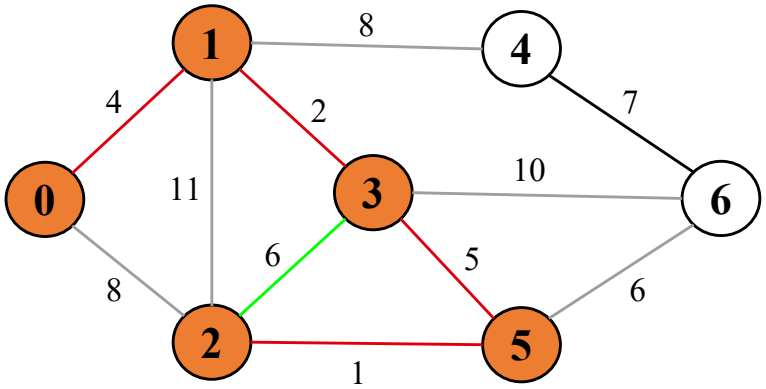
Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
~~(1, 3, 2)~~
(1, 2, 11)
(3, 2, 6)
~~(3, 5, 5)~~
(3, 6, 10)
~~(5, 2, 1)~~
(5, 6, 6)

54

Algoritmo de Prim

- Aresta (3, 2, 6) é umas de custo mais baixo, mas adiciona ciclo ao grafo, então apenas a retiramos a fila



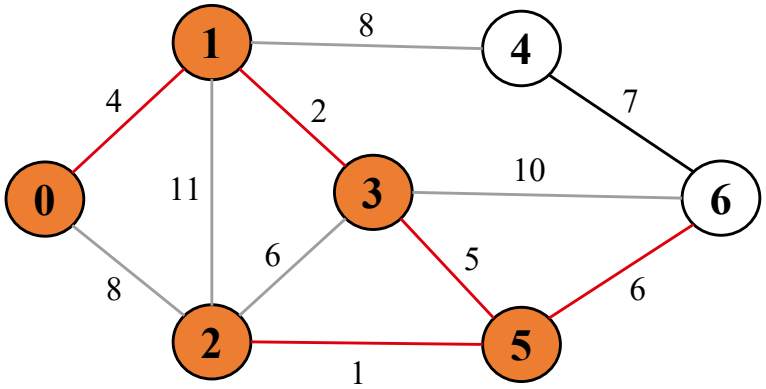
Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
~~(1, 3, 2)~~
(1, 2, 11)
~~(3, 2, 6)~~
~~(3, 5, 5)~~
(3, 6, 10)
~~(5, 2, 1)~~
(5, 6, 6)

55

Algoritmo de Prim

- Menor aresta acessível é (5, 6, 6), a adiciona na resposta



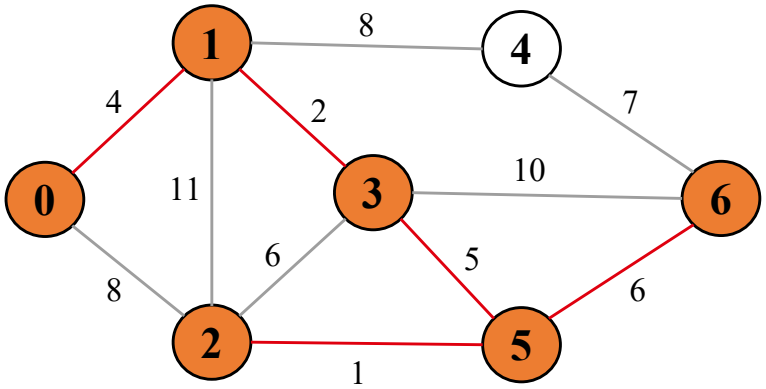
Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
~~(1, 3, 2)~~
(1, 2, 11)
~~(3, 2, 6)~~
~~(3, 5, 5)~~
(3, 6, 10)
~~(5, 2, 1)~~
~~(5, 6, 6)~~

56

Algoritmo de Prim

- Vai ao vértice 6, o marca e adiciona suas arestas válidas à fila



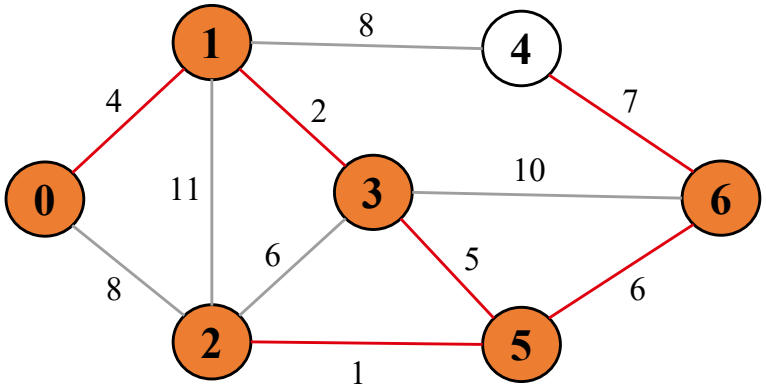
Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
~~(1, 3, 2)~~
(1, 2, 11)
~~(3, 2, 6)~~
~~(3, 5, 5)~~
(3, 6, 10)
~~(5, 2, 1)~~
~~(5, 6, 6)~~
(6, 4, 7)

57

Algoritmo de Prim

- Menor aresta acessível é (6, 4, 7), a adiciona na resposta



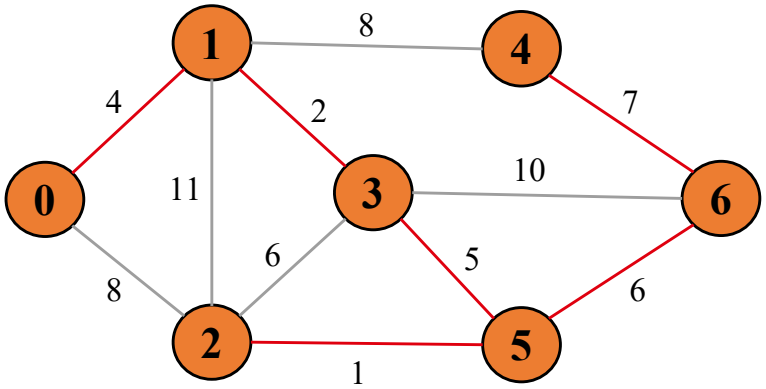
Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
~~(1, 3, 2)~~
(1, 2, 11)
~~(3, 2, 6)~~
~~(3, 5, 5)~~
(3, 6, 10)
~~(5, 2, 1)~~
~~(5, 6, 6)~~
(6, 4, 7)

58

Algoritmo de Prim

- Marcamos o vértice 4, não tem arestas válidas para adicionar na fila

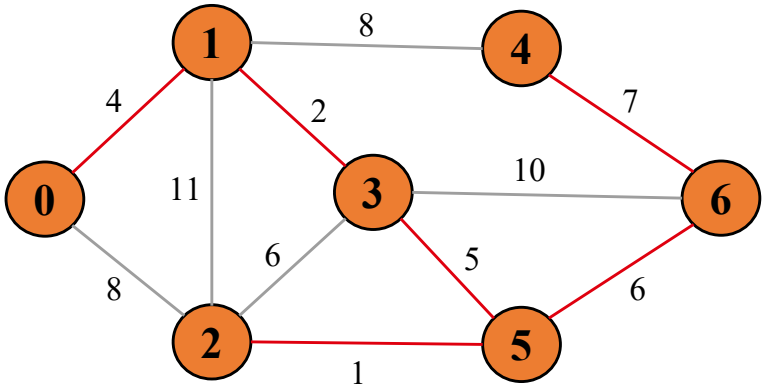


Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
~~(1, 3, 2)~~
(1, 2, 11)
~~(3, 2, 6)~~
~~(3, 5, 5)~~
(3, 6, 10)
~~(5, 2, 1)~~
~~(5, 6, 6)~~
(6, 4, 7)

Algoritmo de Prim

- Percorrendo todas as arestas restantes na fila, chegaremos apenas em vértices marcados…

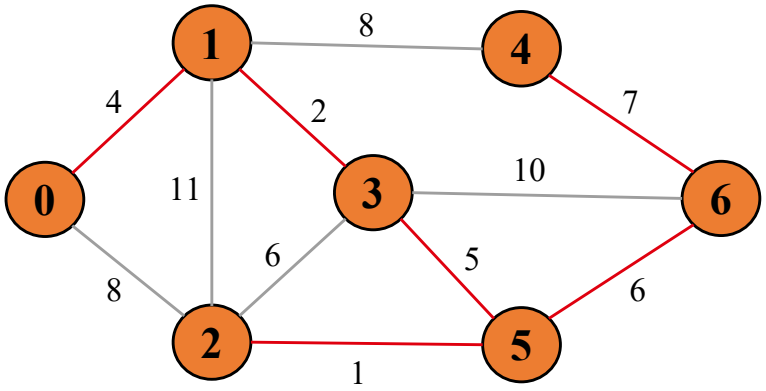


Fila de Prioridade

(0, 2, 8)
~~(0, 1, 4)~~
(1, 4, 8)
~~(1, 3, 2)~~
(1, 2, 11)
~~(3, 2, 6)~~
~~(3, 5, 5)~~
(3, 6, 10)
~~(5, 2, 1)~~
~~(5, 6, 6)~~
(6, 4, 7)

Algoritmo de Prim

- Percorrendo todas as arestas restantes na fila, chegaremos apenas em vértices marcados…

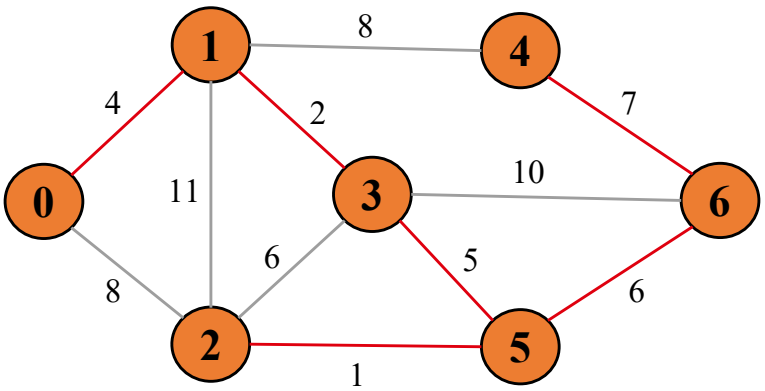


Fila de Prioridade

~~(0, 2, 8)~~
~~(0, 1, 4)~~
~~(1, 4, 8)~~
~~(1, 3, 2)~~
~~(1, 2, 11)~~
~~(3, 2, 6)~~
~~(3, 5, 5)~~
~~(3, 6, 10)~~
~~(5, 2, 1)~~
~~(5, 6, 6)~~
~~(6, 4, 7)~~

Algoritmo de Prim

- Percorrendo todas as arestas restantes na fila, chegaremos apenas em vértices marcados…



Custo da MST: 4 + 2 + 5 + 1 + 6 + 7 = 25

Algoritmo de Prim - Implementação

- Usaremos uma fila de prioridade (heap) para processar os vértices
- Guardaremos as arestas utilizadas na resposta até agora
- Marcaremos vértices para não os visitarmos novamente

1. Começa a partir de um nó inicial s e o marca como visitado.
2. Itera por todas as arestas de s e as adiciona na fila de prioridade (heap).
3. Extrai a aresta de menor custo e a adiciona na resposta, se for válida.
4. Visita o vértice dessa aresta, marca como visitado e adiciona as arestas que não apontam para um vértice já visitado.
5. Volta ao passo 3 até que a fila esteja vazia.

Algoritmo de Prim - Complexidade

- As operações envolvidas são:
  - Extrair o mínimo da heap: O(log V)
  - Marcar o vértice: O(1)
  - Adicionar arestas do vértice atual não visitado na heap: O(E log V)
    - Pois a cada inserção tem custo O(log V)

- Logo, a complexidade de tempo do algoritmo é: O(E log V)

Algoritmo de Kruskal

https://devsuperior.com.br

Prof. Dr. Nelio Alves

Algoritmo de Kruskal

Características

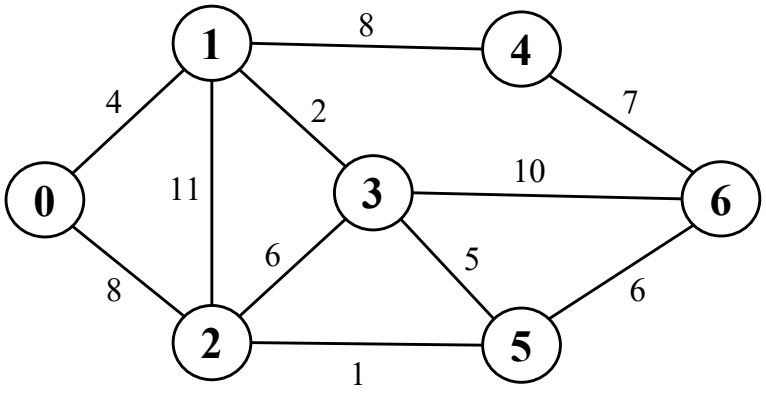
- Procura pela árvore geradora mínima a partir da lista de arestas ordenada por menor custo
- Utiliza a estrutura Disjoint Set Union para formar árvores e as juntar
  - Arestas de menor peso primeiro
    - Algoritmo Guloso!

Estratégia

1. Ordenar a lista de arestas do grafo por menor custo
2. Percorrer cada aresta de menor custo e a adicionar na solução se ela não ligar vértices da mesma componente

Algoritmo de Kruskal

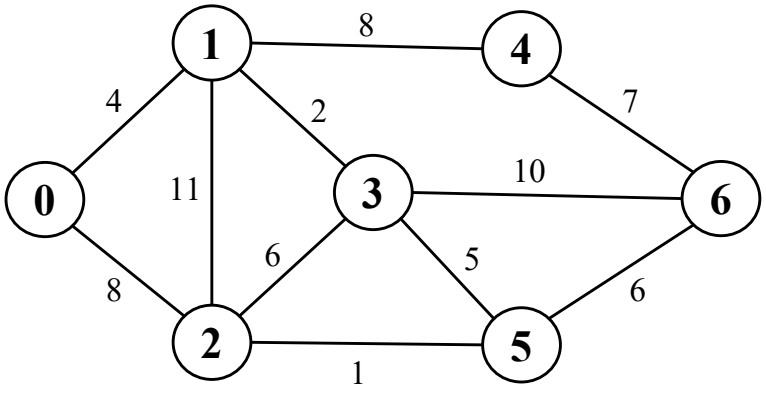
- Grafo inicial com sua lista de arestas



Lista de Arestas
(0, 1, 4)
(0, 2, 8)
(1, 2, 11)
(1, 3, 2)
(1, 4, 8)
(2, 5, 1)
(2, 3, 6)
(3, 6, 10)
(3, 5, 5)
(4, 6, 7)
(5, 6, 6)

Algoritmo de Kruskal

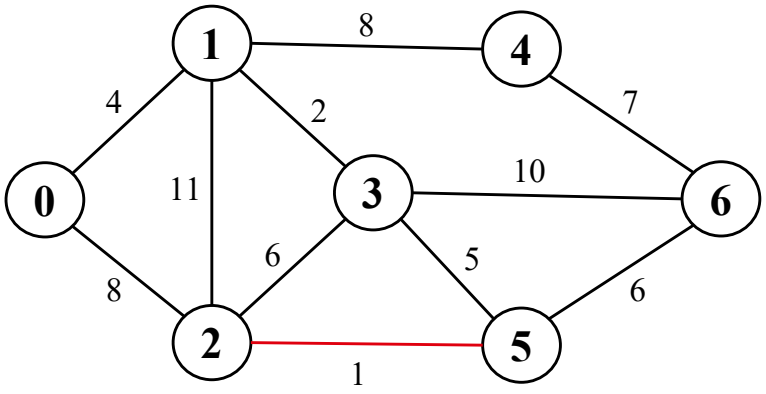
- Ordenar lista de arestas por peso



Lista de Arestas
(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
(2, 3, 6)
(5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Começar pelas arestas menores e tentar ligar vértices sem formar ciclos

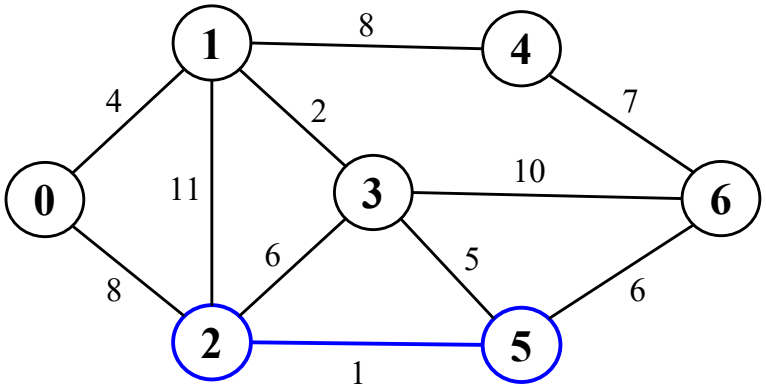


Lista de Arestas

→ (2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
(2, 3, 6)
(5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Podemos formar uma componente azul juntando os vértices 2 e 5

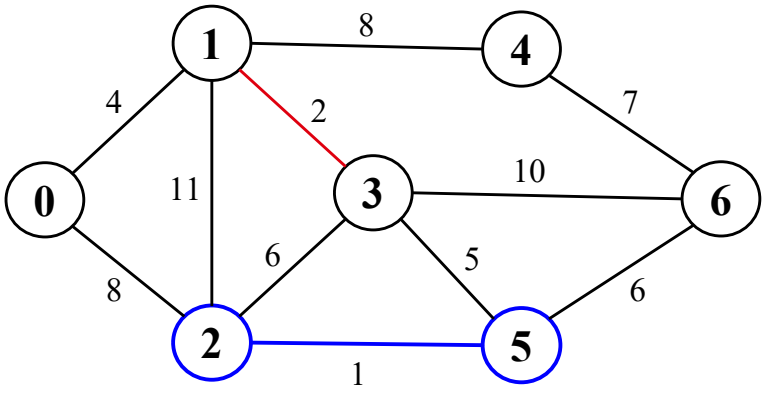


Lista de Arestas

→ **(2, 5, 1)**
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
(2, 3, 6)
(5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Consideramos aresta (1, 3, 2)

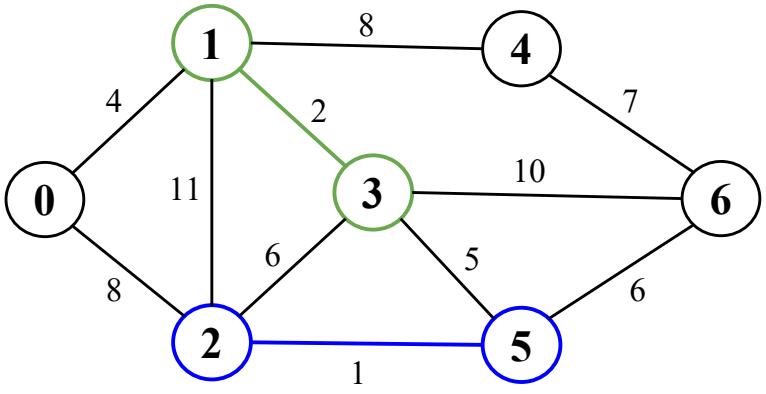


Lista de Arestas

(2, 5, 1)
→ (1, 3, 2)
(0, 1, 4)
(3, 5, 5)
(2, 3, 6)
(5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Podemos formar uma componente verde juntando os vértices 1 e 3

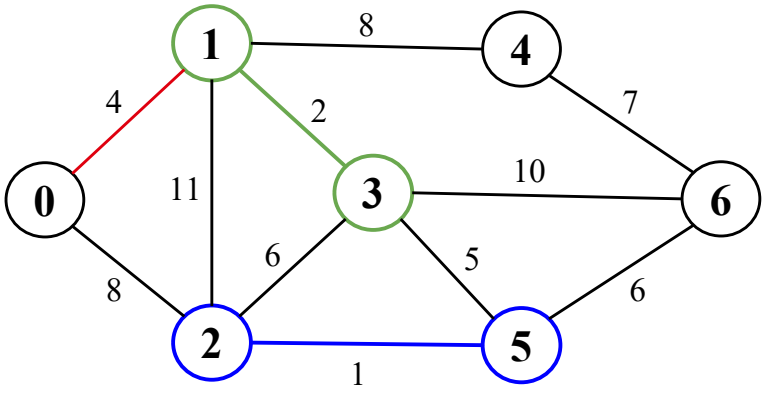


Lista de Arestas

(2, 5, 1)
→ **(1, 3, 2)**
(0, 1, 4)
(3, 5, 5)
(2, 3, 6)
(5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Consideramos aresta (0, 1, 4)



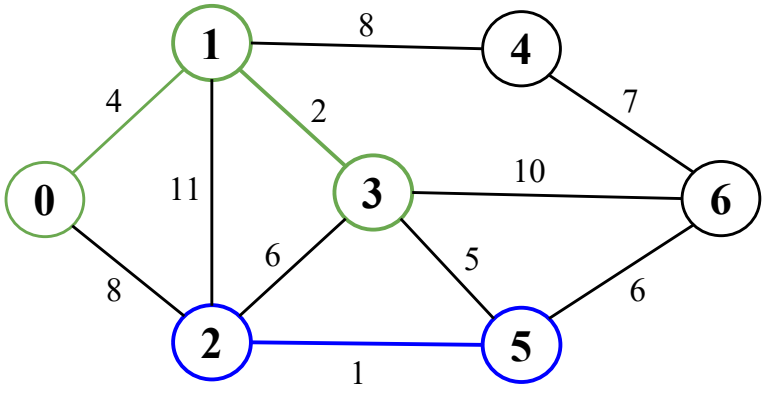
Lista de Arestas

(2, 5, 1)
(1, 3, 2)
→ (0, 1, 4)
(3, 5, 5)
(2, 3, 6)
(5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

73

Algoritmo de Kruskal

- Juntamos 0 à componente verde pois não forma ciclo



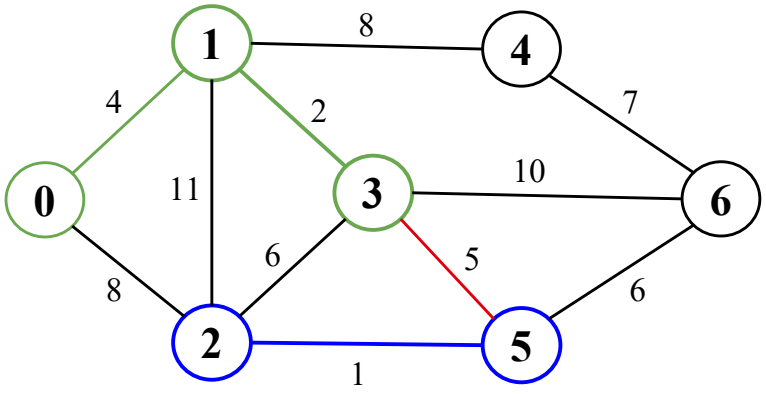
Lista de Arestas

(2, 5, 1)
(1, 3, 2)
→ **(0, 1, 4)**
(3, 5, 5)
(2, 3, 6)
(5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

74

Algoritmo de Kruskal

- Consideramos aresta (3, 5, 5)

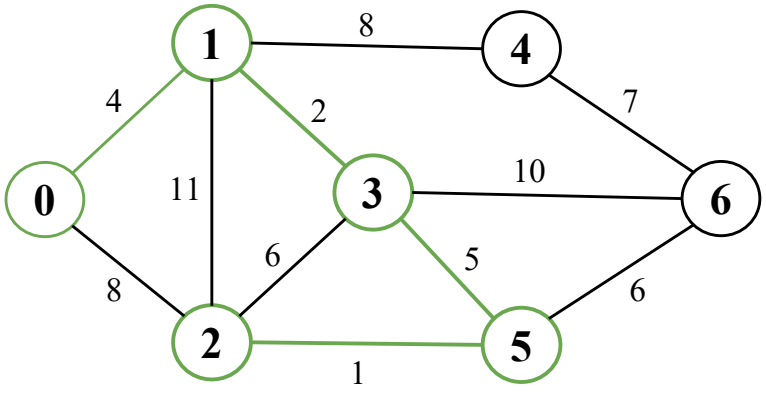


Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
→ (3, 5, 5)
(2, 3, 6)
(5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Aresta (3, 5, 5) liga duas componentes de cores diferentes, podemos fazer a união entre as duas!

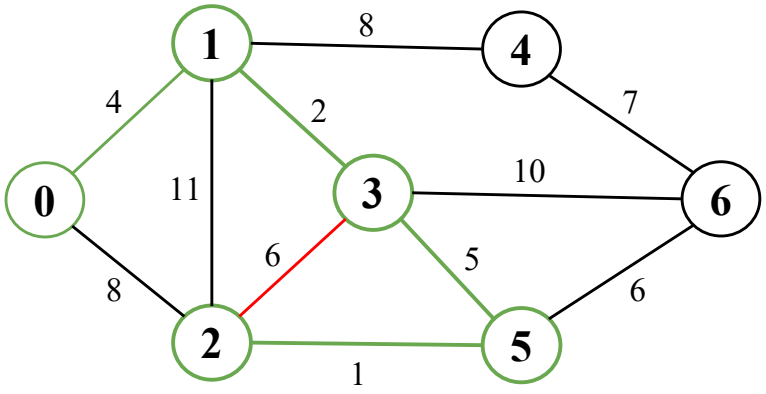


Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
→ **(3, 5, 5)**
(2, 3, 6)
(5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Consideramos aresta (2, 3, 6)

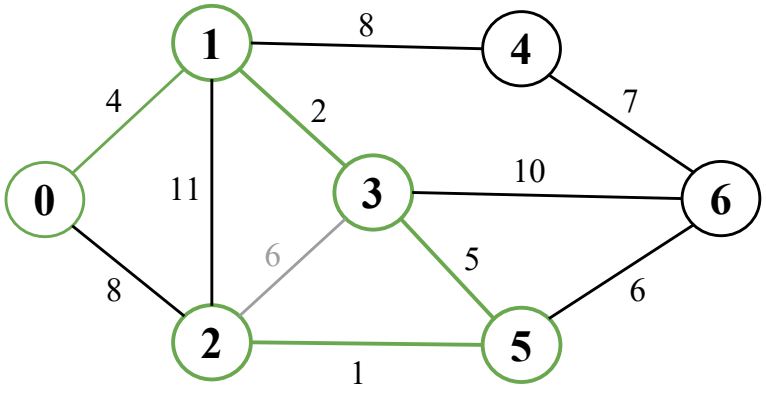


Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
→ (2, 3, 6)
(5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Aresta (2, 3, 6) liga dois vértices verdes, formando ciclo, ignoramos ela.

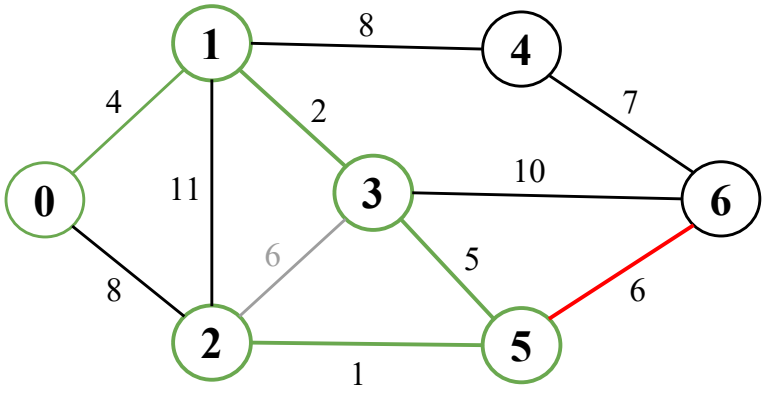


Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
→ ~~(2, 3, 6)~~
(5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Consideramos aresta (5, 6, 6)



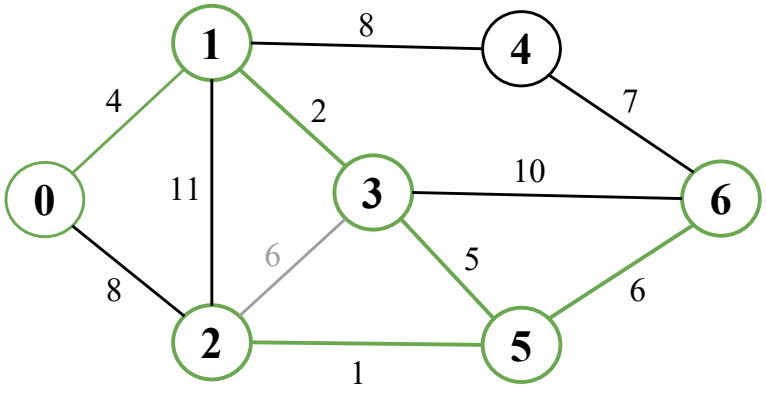
Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
~~(2, 3, 6)~~
→ (5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

79

Algoritmo de Kruskal

- Podemos juntar vértice 6 à componente sem formar ciclos



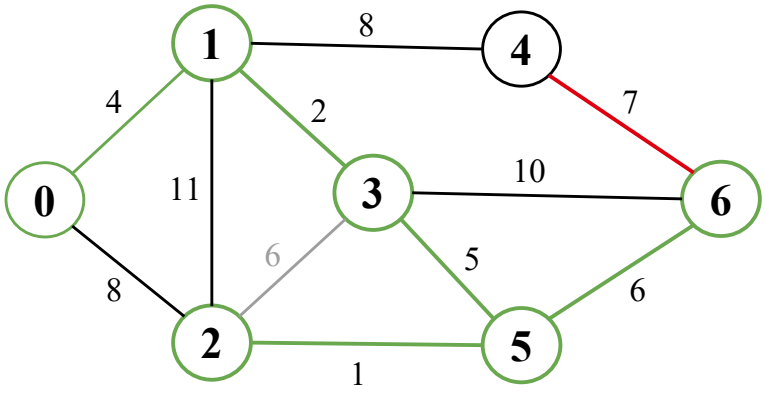
Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
~~(2, 3, 6)~~
→ (5, 6, 6)
(4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

80

Algoritmo de Kruskal

- Consideramos aresta (4, 6, 7)

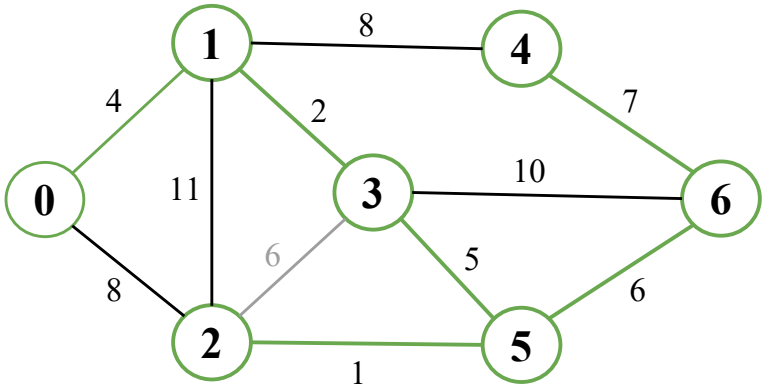


Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
~~(2, 3, 6)~~
(5, 6, 6)
→ (4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Podemos juntar vértice 4 à componente sem formar ciclos

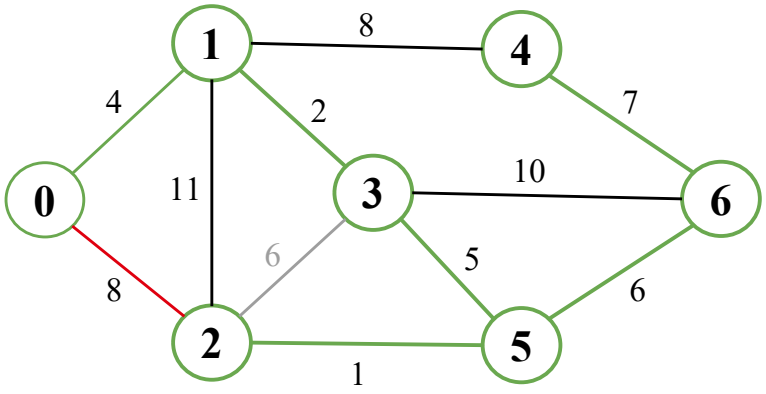


Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
~~(2, 3, 6)~~
(5, 6, 6)
→ (4, 6, 7)
(0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Consideramos aresta (0, 2, 8), a ignoramos pois liga dois vértices verdes

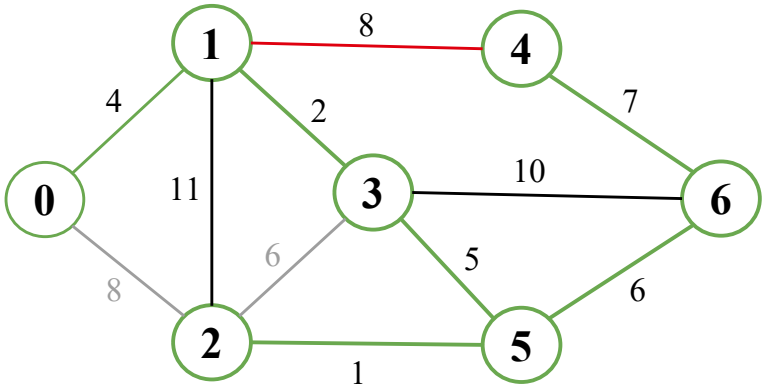


Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
~~(2, 3, 6)~~
(5, 6, 6)
(4, 6, 7)
→ (0, 2, 8)
(1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Consideramos aresta (1, 4, 8), a ignoramos pois liga dois vértices verdes

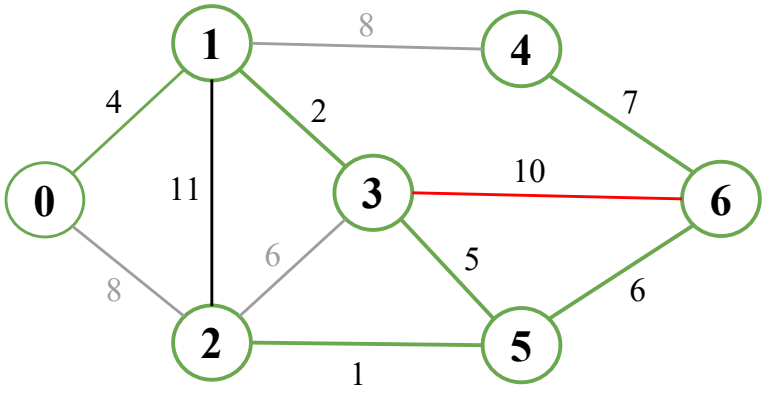


Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
~~(2, 3, 6)~~
(5, 6, 6)
(4, 6, 7)
~~(0, 2, 8)~~
→ (1, 4, 8)
(3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Consideramos aresta (3, 6, 10), a ignoramos pois liga dois vértices verdes

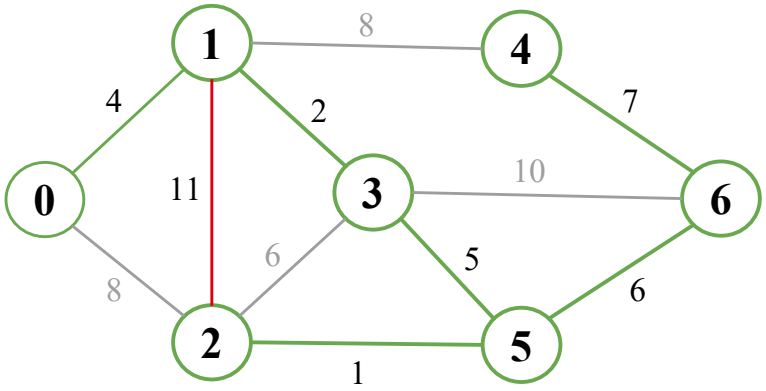


Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
~~(2, 3, 6)~~
(5, 6, 6)
(4, 6, 7)
~~(0, 2, 8)~~
~~(1, 4, 8)~~
→ (3, 6, 10)
(1, 2, 11)

Algoritmo de Kruskal

- Consideramos aresta (1, 2, 11), a ignoramos pois liga dois vértices verdes

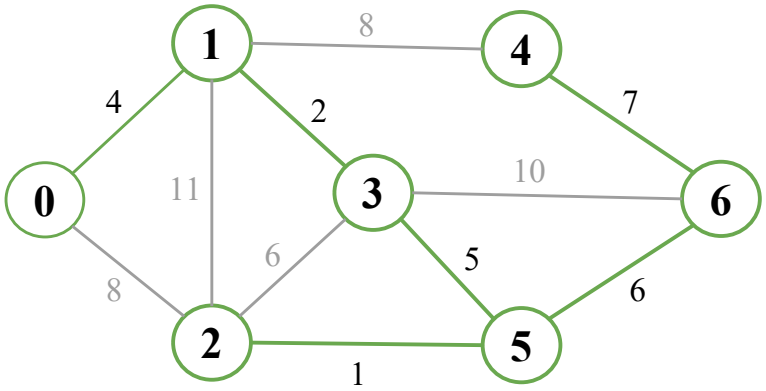


Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
~~(2, 3, 6)~~
(5, 6, 6)
(4, 6, 7)
~~(0, 2, 8)~~
~~(1, 4, 8)~~
~~(3, 6, 10)~~
→ (1, 2, 11)

Algoritmo de Kruskal

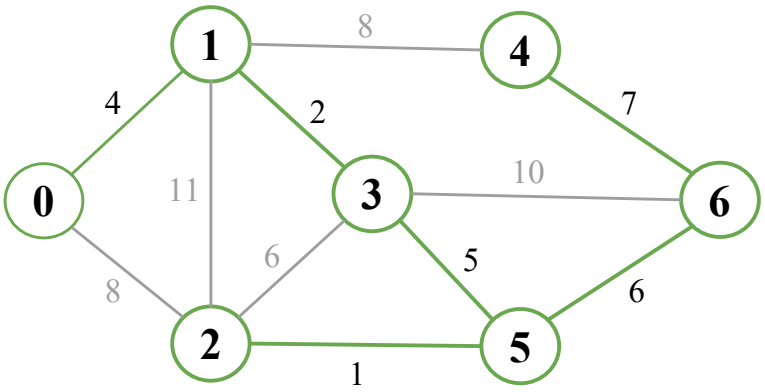
- Finaliza execução, árvore geradora mínima encontrada.



Lista de Arestas

(2, 5, 1)
(1, 3, 2)
(0, 1, 4)
(3, 5, 5)
~~(2, 3, 6)~~
(5, 6, 6)
(4, 6, 7)
~~(0, 2, 8)~~
~~(1, 4, 8)~~
~~(3, 6, 10)~~
~~(1, 2, 11)~~

Algoritmo de Kruskal



Custo da MST: 4 + 2 + 5 + 1 + 6 + 7 = 25

Algoritmo de Kruskal - Implementação

- É simples representar o grafo pela lista de arestas pois precisamos a percorrer depois de ordenar pelos pesos
- Faz uso das estrutura do Union-Find para unir componentes

1. Ordenar arestas por peso
2. Percorrer arestas e verificar se ligam vértices em componentes diferentes, se sim, adicionar à árvore mínima

Complexidade de tempo: O(E log V)

onde,

- E = número de arestas
- V = número de vértices