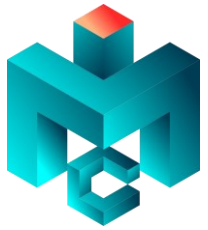
UNIVERSIDADE FEDERAL DE ITAJUBÁ
Instituto de Matemática e Computação



SMAC03 – Grafos

5. Árvores Geradoras

Rafael Frinhani
frinhani@unifei.edu.br

1º Semestre de 2025

OBJETIVOS

Apresentar os conceitos e algoritmos para obtenção de árvores geradores mínimas ou máximas de grafos.

AGENDA

5. Árvores Geradoras

Conceitos, definição de árvore geradora, aplicações.

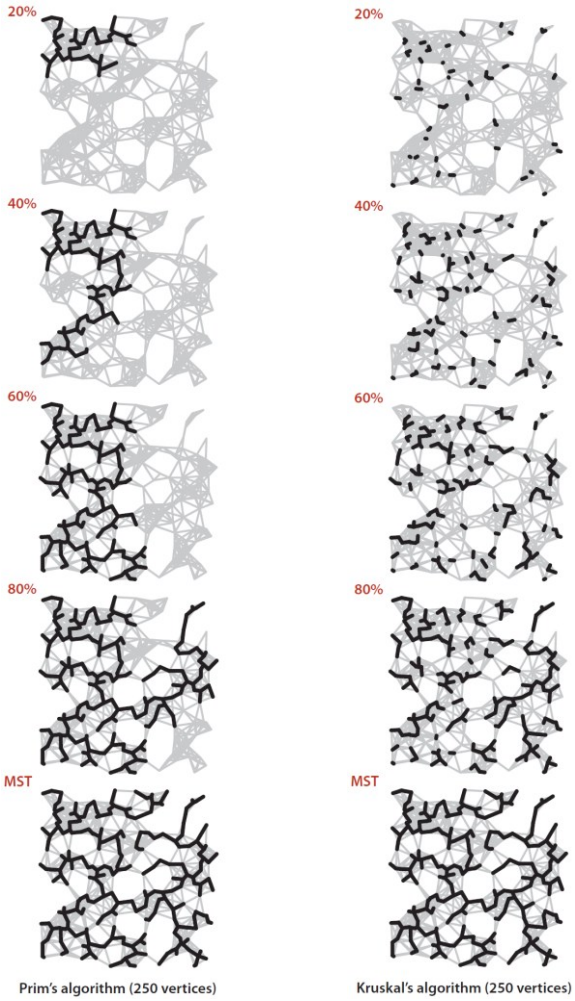
5.1. Algoritmo de Prim

Contexto Histórico, Princípio de funcionamento, Algoritmo, Exemplo.

5.2. Algoritmo de Kruskal

Contexto Histórico, Princípio de funcionamento, Algoritmo, Exemplo.

Comparação dos algoritmos.



Introdução

Em certas situações modeladas em grafos existe o interesse em descobrir uma estrutura de conexão acíclica e com a menor quantidade de arestas, que possibilite alcançar todos os vértices a partir de qualquer vértice.

Árvore: Caso especial de grafo conexo e sem ciclos em que há somente um caminho entre qualquer par de vértices.

Introdução

Em certas situações modeladas em grafos existe o interesse em descobrir uma estrutura de conexão acíclica e com a menor quantidade de arestas, que possibilite alcançar todos os vértices a partir de qualquer vértice.

Árvore: Caso especial de grafo conexo e sem ciclos em que há somente um caminho entre qualquer par de vértices.

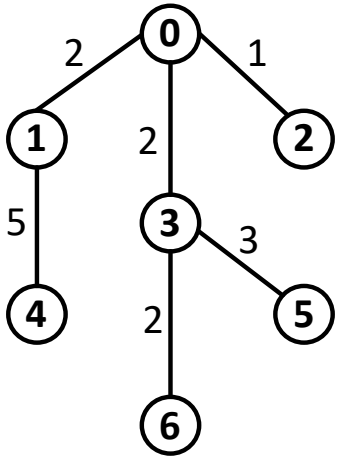
Características: Seja T uma árvore com $|V|$ vértices, então:

- T é conexo e sem ciclos;
- T possui um total de $|V| - 1$ arestas (V é o conjunto de vértices);
- Cada aresta de T é uma ponte (aresta cuja remoção aumenta o número de componentes conexas);
- T é um grafo planar;
- Se $|V| > 1$, então T possui ao menos dois vértices terminais (folhas).

Introdução

Em certas situações modeladas em grafos existe o interesse em descobrir uma estrutura de conexão acíclica e com a menor quantidade de arestas, que possibilite alcançar todos os vértices a partir de qualquer vértice.

Árvore: Caso especial de grafo conexo e sem ciclos em que há somente um caminho entre qualquer par de vértices.



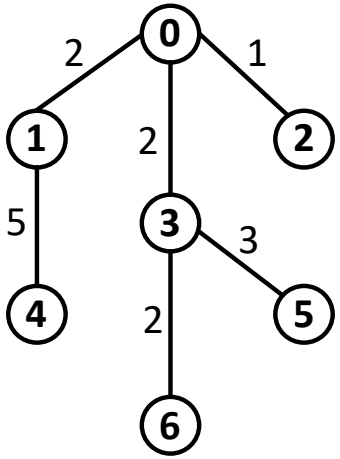
Árvore Ponderada

Introdução

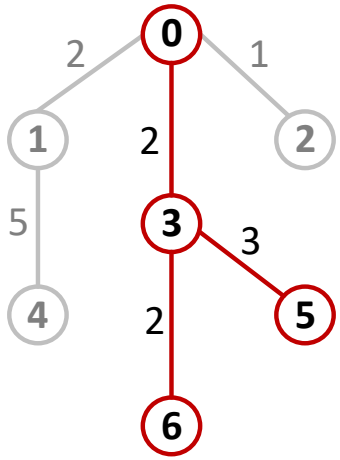
Em certas situações modeladas em grafos existe o interesse em descobrir uma estrutura de conexão acíclica e com a menor quantidade de arestas, que possibilite alcançar todos os vértices a partir de qualquer vértice.

Árvore: Caso especial de grafo conexo e sem ciclos em que há somente um caminho entre qualquer par de vértices.

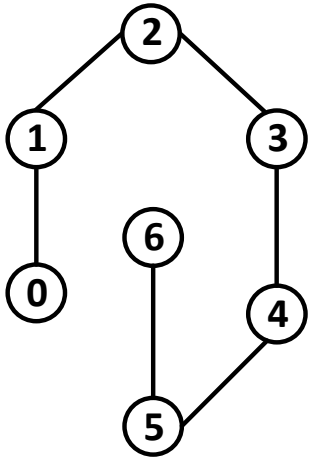
- **Subárvore:** Subgrafo conexo e acíclico de uma árvore.
- **Grafo Linha:** ou grafo caminho, é aquele no qual todos os vértices têm grau 2, existindo apenas dois vértices de grau 1 (extremos).



Árvore Ponderada



Subárvore (em vermelho)

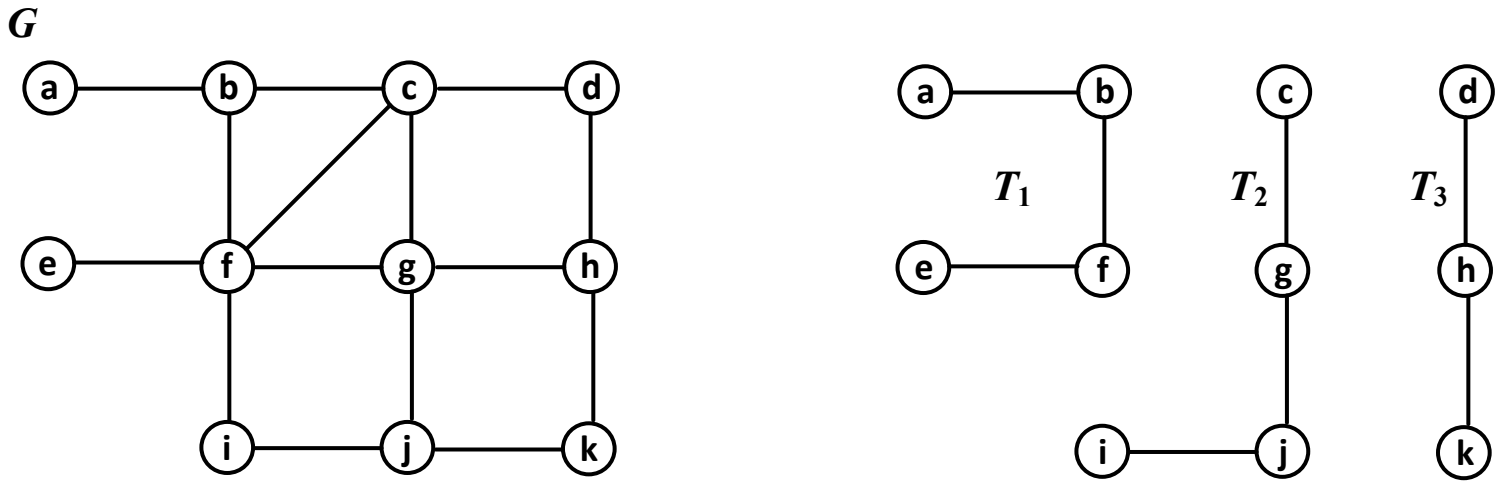


Grafo Linha

Introdução

Em certas situações modeladas em grafos existe o interesse em descobrir uma estrutura de conexão acíclica e com a menor quantidade de arestas, que possibilite alcançar todos os vértices a partir de qualquer vértice.

Floresta: Corresponde aos grafos acíclicos e desconexos. Conjunto de árvores sem vértices em comum, dado pela união disjunta de árvores.



Exemplo de floresta constituída pelas árvores T_1, T_2 e T_3 obtidas a partir do grafo G.

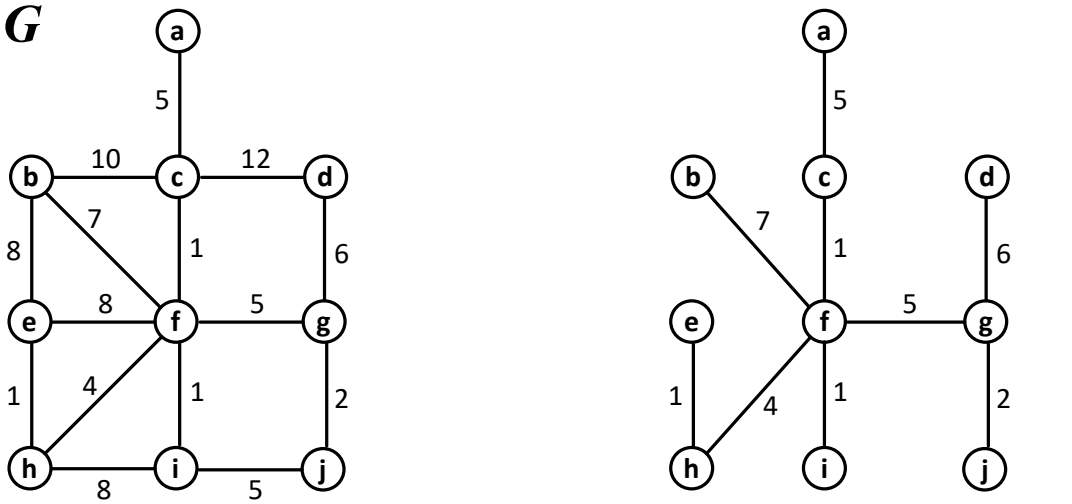
Definição

Uma árvore geradora (*spanning tree*) de um grafo G é um subgrafo conexo e acíclico, que possui todos os vértices de G conectados por um subconjunto de arestas de G. A quantidade total de arestas de uma árvore geradora é $|V| - 1$.

Definição

Uma árvore geradora (*spanning tree*) de um grafo G é um subgrafo conexo e acíclico, que possui todos os vértices de G conectados por um subconjunto de arestas de G. A quantidade total de arestas de uma árvore geradora é $|V| - 1$.

Uma **árvore geradora mínima** (*Minimum Spanning Tree*, MST) é um subconjunto não-direcionado, conexo, ponderado e acíclico das arestas de um grafo, que conectam todos os vértices e cuja soma dos seus pesos é a menor possível.

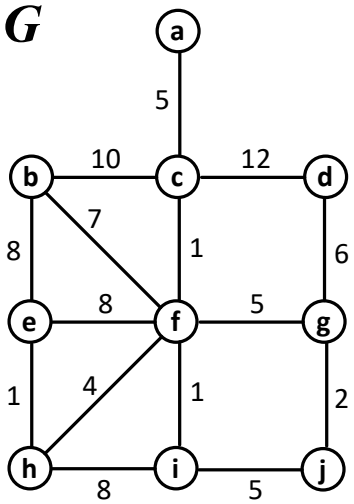


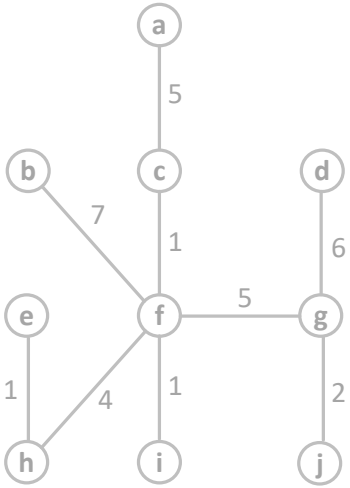
Árvore Geradora Mínima

Definição

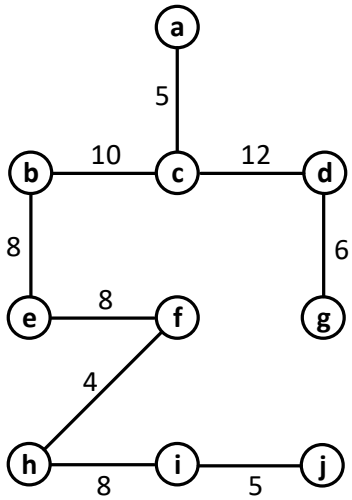
Uma árvore geradora (*spanning tree*) de um grafo G é um subgrafo conexo e acíclico, que possui todos os vértices de G conectados por um subconjunto de arestas de G. A quantidade total de arestas de uma árvore geradora é $|V| - 1$.

Analogamente, **árvore geradora máxima** é a árvore geradora de maior custo dentre todas as possíveis em um grafo G. Pode ser obtida pelos mesmos métodos de árvores geradoras mínimas, mas consideram as arestas de maior peso.





Árvore Geradora Mínima



Árvore Geradora Máxima

Aplicações

- Projeto de redes de comunicação (ex. computadores, celular etc.), métodos para eficiência de operação (*Spanning Tree Protocol*, STP), Redes de Sensores sem Fio;
- Projetos de redes de energia, *design* de circuitos eletrônicos, redes hidráulicas e de esgoto, redes de petróleo ou gás;
- Identificação de caminho crítico em processos de negócio, teoria dos jogos;
- Mapas geográficos (ex. regionalização de áreas sócio geográficas), mapas de altimetria de terrenos, estudos ecotoxicológicos;

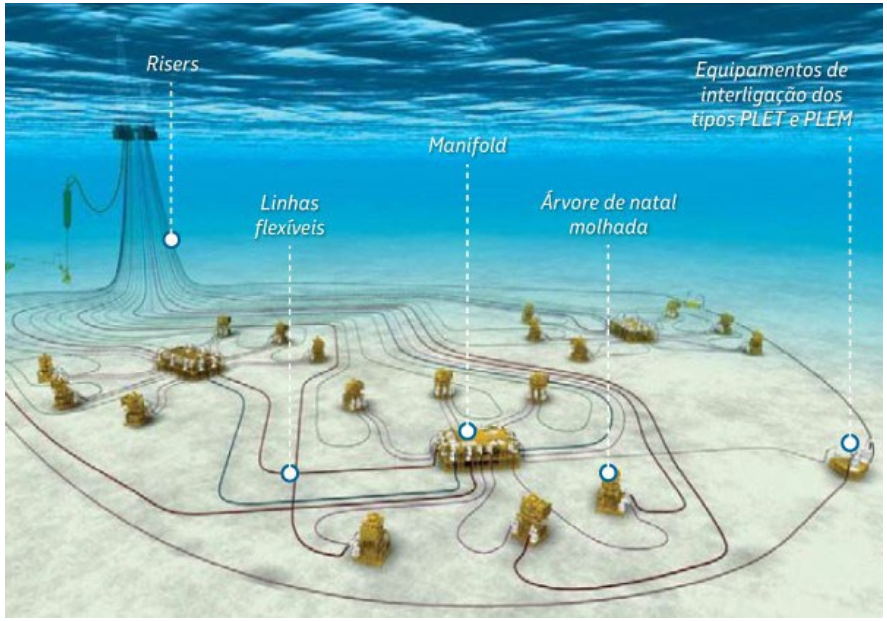
Aplicações

- Projeto de redes de comunicação (ex. computadores, celular etc.), métodos para eficiência de operação (*Spanning Tree Protocol*, STP), Redes de Sensores sem Fio;
- Projetos de redes de energia, *design* de circuitos eletrônicos, redes hidráulicas e de esgoto, redes de petróleo ou gás;
- Identificação de caminho crítico em processos de negócio, teoria dos jogos;
- Mapas geográficos (ex. regionalização de áreas sócio geográficas), mapas de altimetria de terrenos, estudos ecotoxicológicos;
- Análise de agrupamentos de dados, reconhecimento automático de fala;
- Diagnóstico por Imagem (ex. ressonância magnética)
- Análise genética (ex. otimizar mapas genéticos densos, ordenação de marcadores),
- Análise de padrões de distribuição espacial de esporos;
- Astronomia (ex. determinar agrupamento de *quasars*, segregação de massas em estrelas);
- Modelos de interação de partículas em fluxo turbulento de fluidos, simulação de sistemas de escoamento etc.

Aplicações – Exemplos

Sistemas Petrolíferos

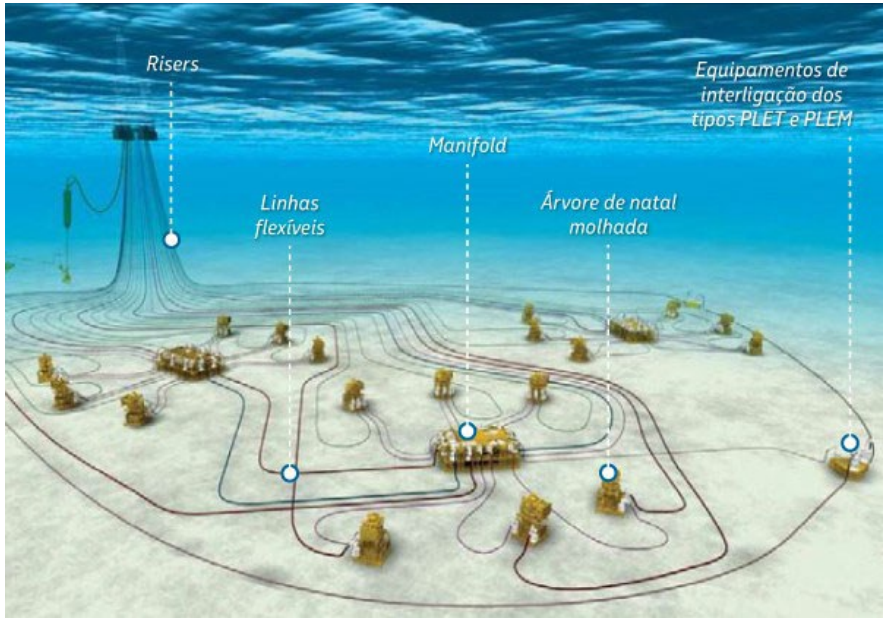
Campos de petróleo coletam e transferem o óleo de poços submarinos até um ponto de transporte. Concentradores (*mainfolds*) reúnem a produção de vários poços para reduzir a quantidade de tubulações que entram na plataforma, regular o fluxo de óleo, entre outros. Definir a melhor localização e a capacidade dos concentradores contribui para redução dos custos de produção.



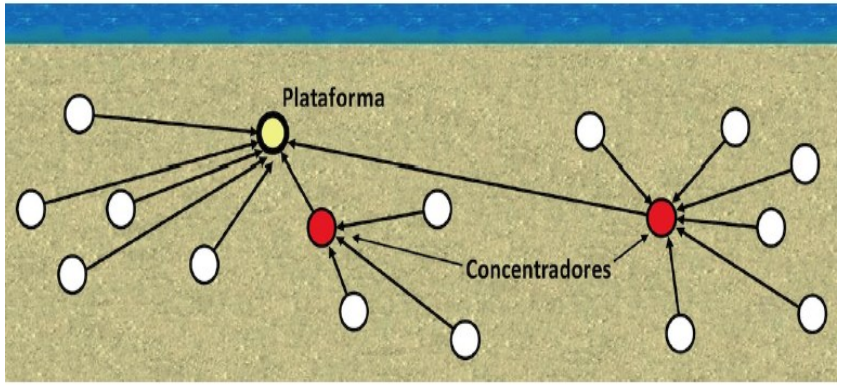
Aplicações – Exemplos

Sistemas Petrolíferos

Campos de petróleo coletam e transferem o óleo de poços submarinos até um ponto de transporte. Concentradores (*mainfolds*) reúnem a produção de vários poços para reduzir a quantidade de tubulações que entram na plataforma, regular o fluxo de óleo, entre outros. Definir a melhor localização e a capacidade dos concentradores contribui para redução dos custos de produção.



Um modelo em grafos de uma AGM pode representar um sistema ótimo com os vértices vermelhos representando os concentradores, os brancos os poços e a plataforma em amarelo.



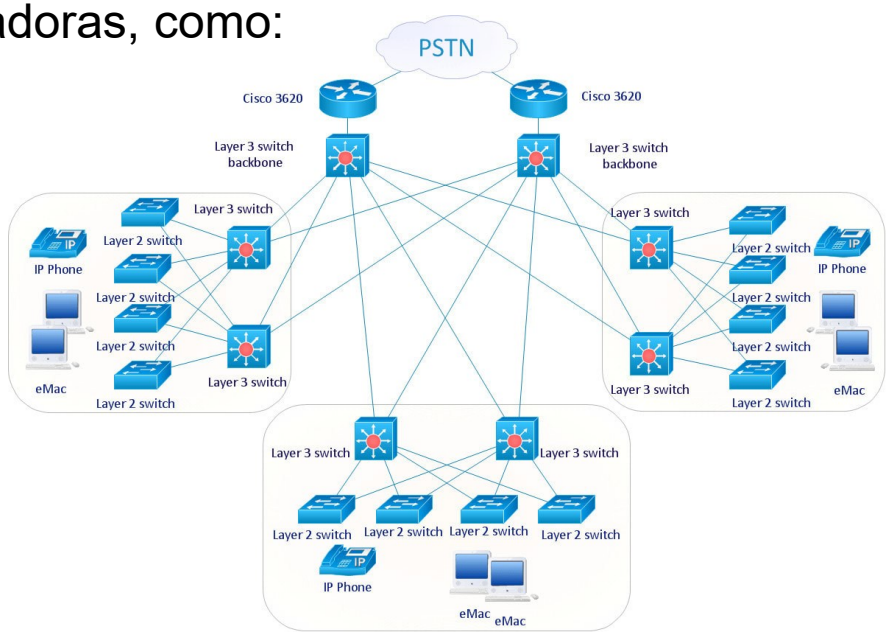
Aplicações – Exemplos

Redes de Comunicação

Em redes de comunicação (ex. Internet, LAN, telefonia etc.) *links* de transmissão conectam nós em uma topologia de malha que tipicamente inclui loops.

Certos *loops* devem removidos para evitar uma redundância no encaminhamento de pacotes de dados e perda de desempenho. Para isso, diversos protocolos de roteamento são baseados em árvores geradoras, como:

- *Spanning Tree Protocol* (STP)
- *Open Shortest Path First*
- *Link-state Routing Protocol*,
- *Augmented Tree-based Routing* etc.

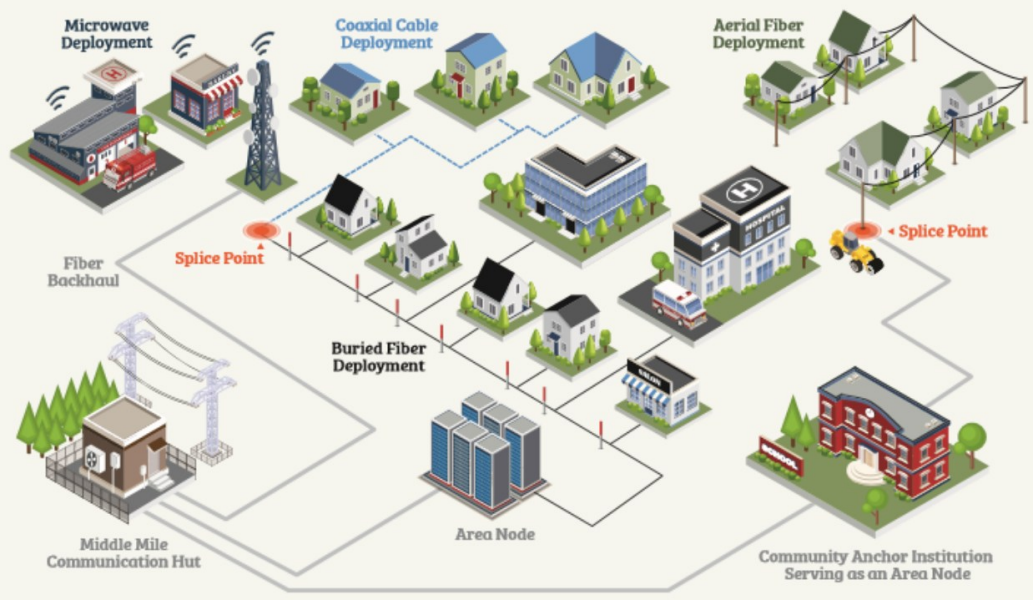


Aplicações – Exemplos

Redes de Comunicação (Fibra Óptica)

Projetos de redes ópticas podem abranger uma vasta área (ex. campus , cidades, metrópoles) envolvendo custos significativos de implantação e de operação.

Em uma rede óptica o sinal é transmitido por canais de fibra, sendo que caixas de emenda são utilizadas para derivações de canais de um mesmo cabo.



Aplicações – Exemplos

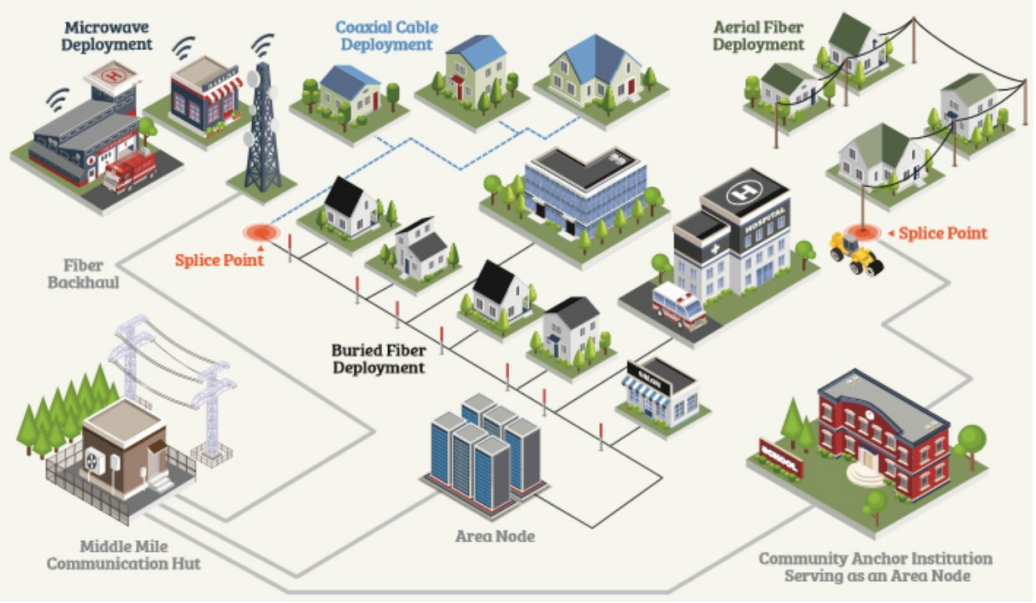
Redes de Comunicação (Fibra Óptica)

Projetos de redes ópticas podem abranger uma vasta área (ex. campus , cidades, metrópoles) envolvendo custos significativos de implantação e de operação.

Em uma rede óptica o sinal é transmitido por canais de fibra, sendo que caixas de emenda são utilizadas para derivações de canais de um mesmo cabo.

Uma árvore geradora mínima pode auxiliar no projeto de uma infraestrutura de rede óptica, que seja otimizada em relação a quantidade de pontos de derivação e ramais.

Também pode auxiliar na identificação de pontos ou canais críticos da rede.



Algoritmo de Prim

Proposto originalmente em 1930 pelo matemático tcheco Vojtech Jarník, posteriormente pelo cientista da computação americano Robert C. Prim (1957) e redescoberto em seguida pelo holandês Edsger Dijkstra em 1959.

Princípio de Funcionamento

- Através de estratégia gulosa, inclui vértice por vértice em uma árvore parcial (subárvore geradora) até obter a árvore geradora mínima completa;
- O algoritmo inicia por qualquer vértice do grafo;
- A cada passo, seleciona um vértice de um conjunto de vértices que ainda não foram analisados, cuja aresta é extremidade de um dos vértices que já foram selecionados para compor a árvore geradora e possui o menor peso entre todas do conjunto.

Algoritmo de Prim

```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6      Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7      S = S ∪ u;
8      N = N \ u;
9      T = T ∪ {v, u}
10 end
11 return T
```

ESTRUTURAS

- V, E : conjunto de vértices e arestas do grafo.
- S : conjunto de vértices já selecionados pelo algoritmo.
- N : conjunto de vértices não selecionados pelo algoritmo.
- T : conjunto de arestas que constituem a AGM.
- w_{vu} : valor do peso da aresta que conecta v e u.

Definir como v qualquer vértice do grafo e adicioná-lo ao conjunto de vértices selecionados (linhas 1 e 2).

Algoritmo de Prim

```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6  |   Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7  |   S = S ∪ u;
8  |   N = N \ u;
9  |   T = T ∪ {v, u}
10 end
11 return T
```

ESTRUTURAS

- V, E : conjunto de vértices e arestas do grafo.
- S : conjunto de vértices já selecionados pelo algoritmo.
- N : conjunto de vértices não selecionados pelo algoritmo.
- T : conjunto de arestas que constituem a AGM.
- w_{vu} : valor do peso da aresta que conecta v e u.

Definir como v qualquer vértice do grafo e adicioná-lo ao conjunto de vértices selecionados (linhas 1 e 2).

Remover v do conjunto de vértices não selecionados e construir o conjunto T que armazenará as arestas da árvore geradora mínima (linhas 3 e 4).

Algoritmo de Prim

```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6  |   Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7  |   S = S ∪ u;
8  |   N = N \ u;
9  |   T = T ∪ {v, u}
10 end
11 return T
```

ESTRUTURAS

- V, E : conjunto de vértices e arestas do grafo.
- S : conjunto de vértices já selecionados pelo algoritmo.
- N : conjunto de vértices não selecionados pelo algoritmo.
- T : conjunto de arestas que constituem a AGM.
- w_{vu} : valor do peso da aresta que conecta v e u.

Definir como v qualquer vértice do grafo e adicioná-lo ao conjunto de vértices selecionados (linhas 1 e 2).

Remover v do conjunto de vértices não selecionados e construir o conjunto T que armazenará as arestas da árvore geradora mínima (linhas 3 e 4).

Enquanto o tamanho do conjunto T for menor que a quantidade de arestas da AGM (linhas 5 a 10):

Obter uma aresta $\{v, u\}$ pertencente ao conjunto de arestas E do grafo, que tem o menor custo entre todas os demais ainda não selecionadas (linha 6). Obs. v é um vértice de S e u é um vértice de N.

Algoritmo de Prim

```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6      Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7      S = S ∪ u;
8      N = N \ u;
9      T = T ∪ {v, u}
10 end
11 return T
```

ESTRUTURAS

- V, E : conjunto de vértices e arestas do grafo.
- S : conjunto de vértices já selecionados pelo algoritmo.
- N : conjunto de vértices não selecionados pelo algoritmo.
- T : conjunto de arestas que constituem a AGM.
- w_{vu} : valor do peso da aresta que conecta v e u.

Definir como v qualquer vértice do grafo e adicioná-lo ao conjunto de vértices selecionados (linhas 1 e 2).

Remover v do conjunto de vértices não selecionados e construir o conjunto T que armazenará as arestas da árvore geradora mínima (linhas 3 e 4).

Enquanto o tamanho do conjunto T for menor que a quantidade de arestas da AGM (linhas 5 a 10):

Obter uma aresta $\{v, u\}$ pertencente ao conjunto de arestas E do grafo, que tem o menor custo entre todas os demais ainda não selecionadas (linha 6). Obs. v é um vértice de S e u é um vértice de N.

Incluir u no conjunto de vértices selecionados e remover do conjunto dos não selecionados (linhas 7 e 8).

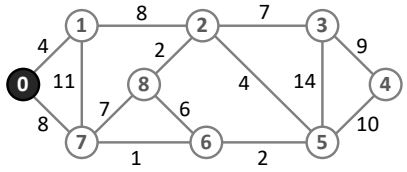
Adicionar a aresta $\{v, u\}$ no conjunto T, que contém as arestas da árvore geradora mínima (linha 9).

Retornar T ao final da execução do algoritmo (linha 11).

Algoritmo de Prim – Exemplo

prim(G)

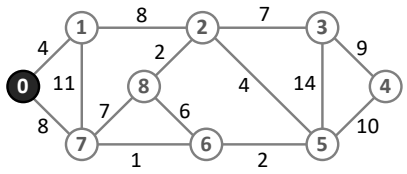
```
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6  |   Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7  |   S = S ∪ u;
8  |   N = N \ u;
9  |   T = T ∪ {v, u}
10 end
11 return T
```

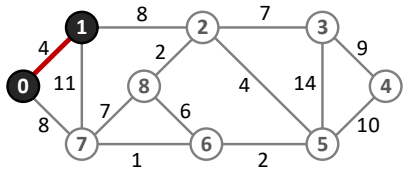


Considerando como v o vértice de índice 0 para o início da construção da árvore.

Algoritmo de Prim – Exemplo

```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6  |   Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7  |   S = S ∪ u;
8  |   N = N \ u;
9  |   T = T ∪ {v, u}
10 end
11 return T
```



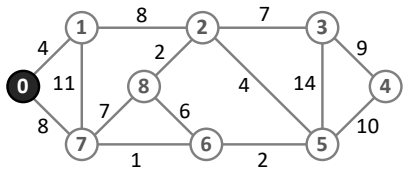


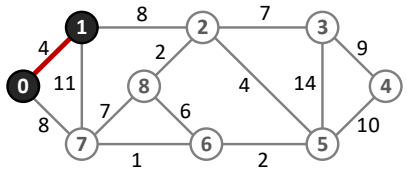
Considerando como v o vértice de índice 0 para o início da construção da árvore.

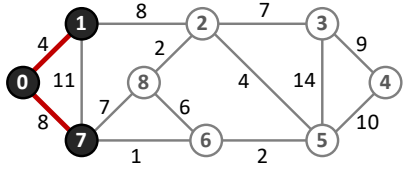
O vértice de id 1 (u) é selecionado como o de menor peso entre os adjacentes a 0 (v) do conjunto N (não selecionados).

Algoritmo de Prim – Exemplo

```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6  |   Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7  |   S = S ∪ u;
8  |   N = N \ u;
9  |   T = T ∪ {v, u}
10 end
11 return T
```







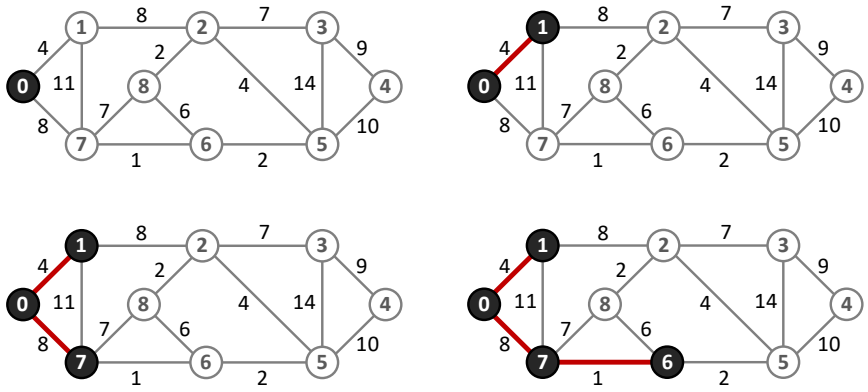
Considerando como v o vértice de índice 0 para o início da construção da árvore.

O vértice de id 1 (u) é selecionado como o de menor peso entre os adjacentes a 0 (v) do conjunto N (não selecionados).

Em seguida o vértice de *id* 7 é selecionado. A aresta tem peso igual ao vértice 2, mas 7 é selecionado pelo índice de seu adjacente (0).

Algoritmo de Prim – Exemplo

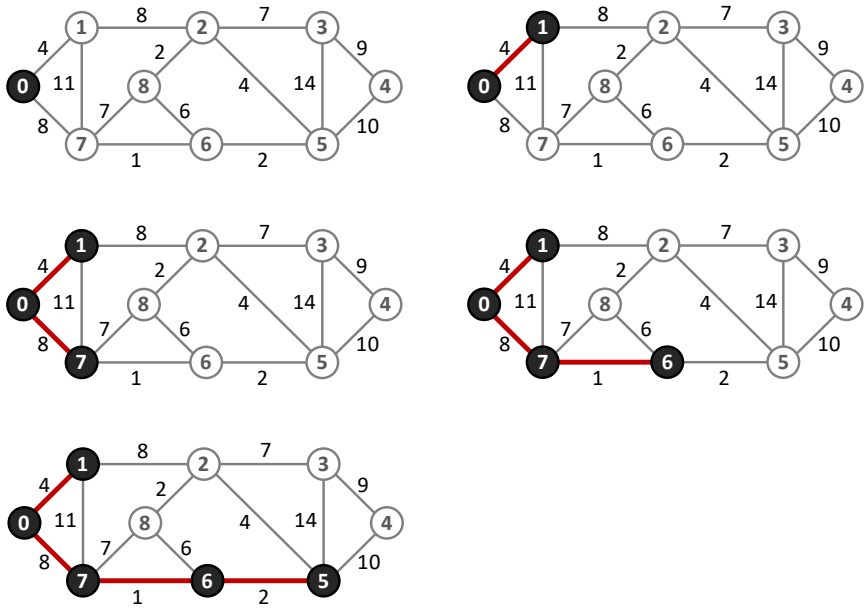
```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6  |   Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7  |   S = S ∪ u;
8  |   N = N \ u;
9  |   T = T ∪ {v, u}
10 end
11 return T
```



O vértice 6 é selecionado, pois sua aresta possui o menor peso em relação as arestas de outros vértices de S {0, 1, 7} e vértices disponíveis de N.

Algoritmo de Prim – Exemplo

```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6  |   Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7  |   S = S ∪ u;
8  |   N = N \ u;
9  |   T = T ∪ {v, u}
10 end
11 return T
```

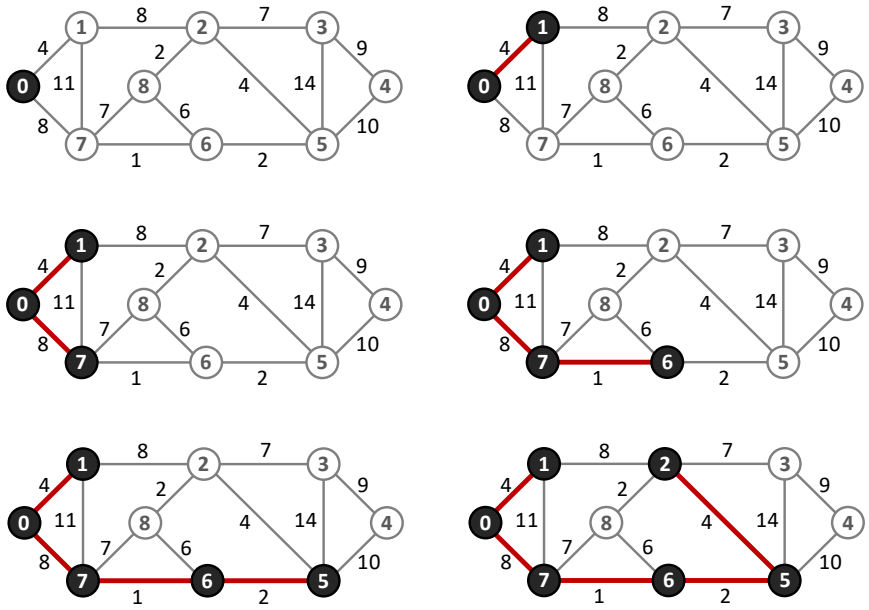


O vértice 8 é selecionado, pois sua aresta possui o menor peso em relação as arestas de outros vértices de S {0, 1, 2} e vértices disponíveis de N.

Em seguida é selecionado vértice 5 (aresta 6–5),

Algoritmo de Prim – Exemplo

```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6  |   Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7  |   S = S ∪ u;
8  |   N = N \ u;
9  |   T = T ∪ {v, u}
10 end
11 return T
```



O vértice 8 é selecionado, pois sua aresta possui o menor peso em relação as arestas de outros vértices de S {0, 1, 2} e vértices disponíveis de N.

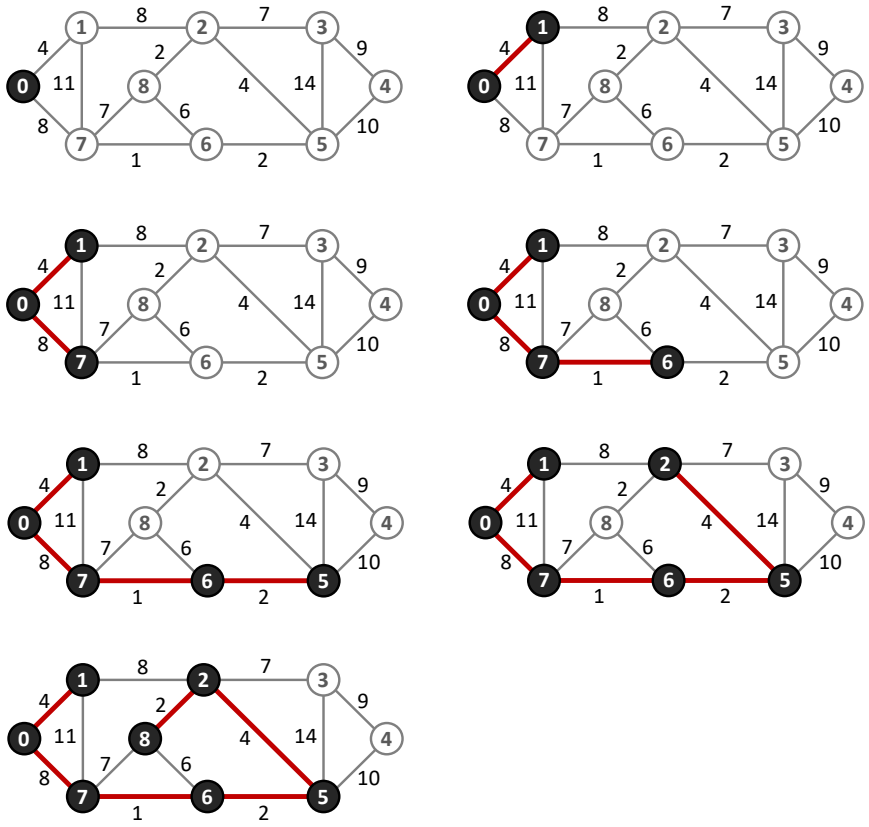
Em seguida é selecionado vértice 5 (aresta 6–5), vértice 2

Algoritmo de Prim – Exemplo

```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6  |   Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7  |   S = S ∪ u;
8  |   N = N \ u;
9  |   T = T ∪ {v, u}
10 end
11 return T
```

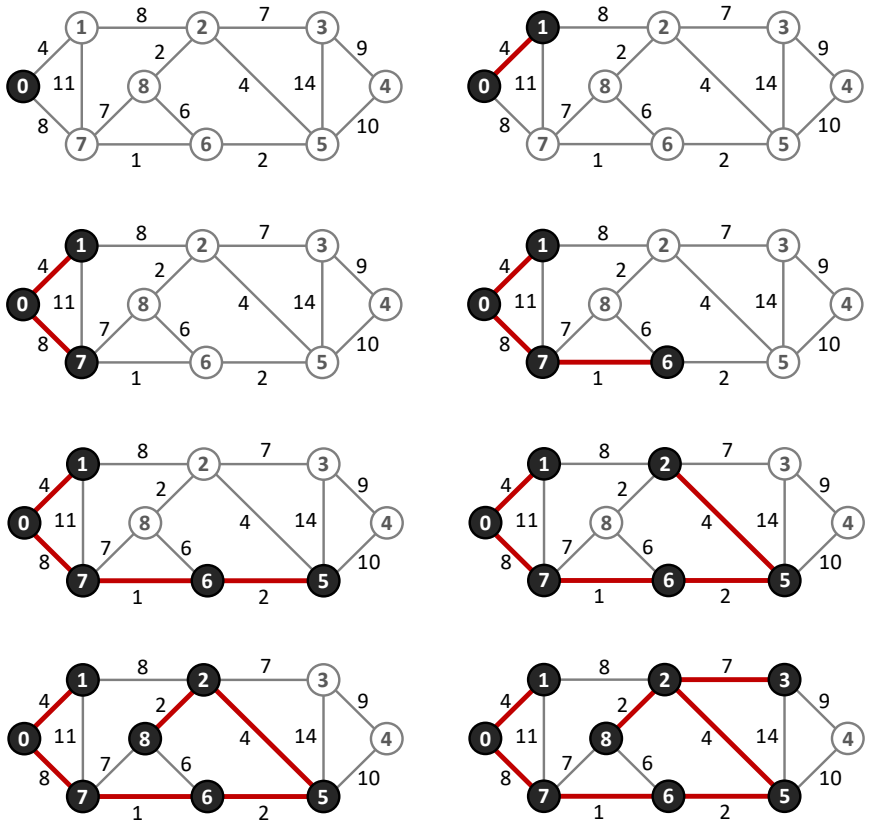
O vértice 8 é selecionado, pois sua aresta possui o menor peso em relação as arestas de outros vértices de S {0, 1, 2} e vértices disponíveis de N.

Em seguida é selecionado vértice 5 (aresta 6–5), vértice 2 e vértice 8.



Algoritmo de Prim – Exemplo

```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6  |   Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7  |   S = S ∪ u;
8  |   N = N \ u;
9  |   T = T ∪ {v, u}
10 end
11 return T
```



O vértice 8 é selecionado, pois sua aresta possui o menor peso em relação as arestas de outros vértices de S {0, 1, 2} e vértices disponíveis de N.

Em seguida é selecionado vértice 5 (aresta 6–5), vértice 2 e vértice 8.

A construção da árvore continua no vértice de id 3

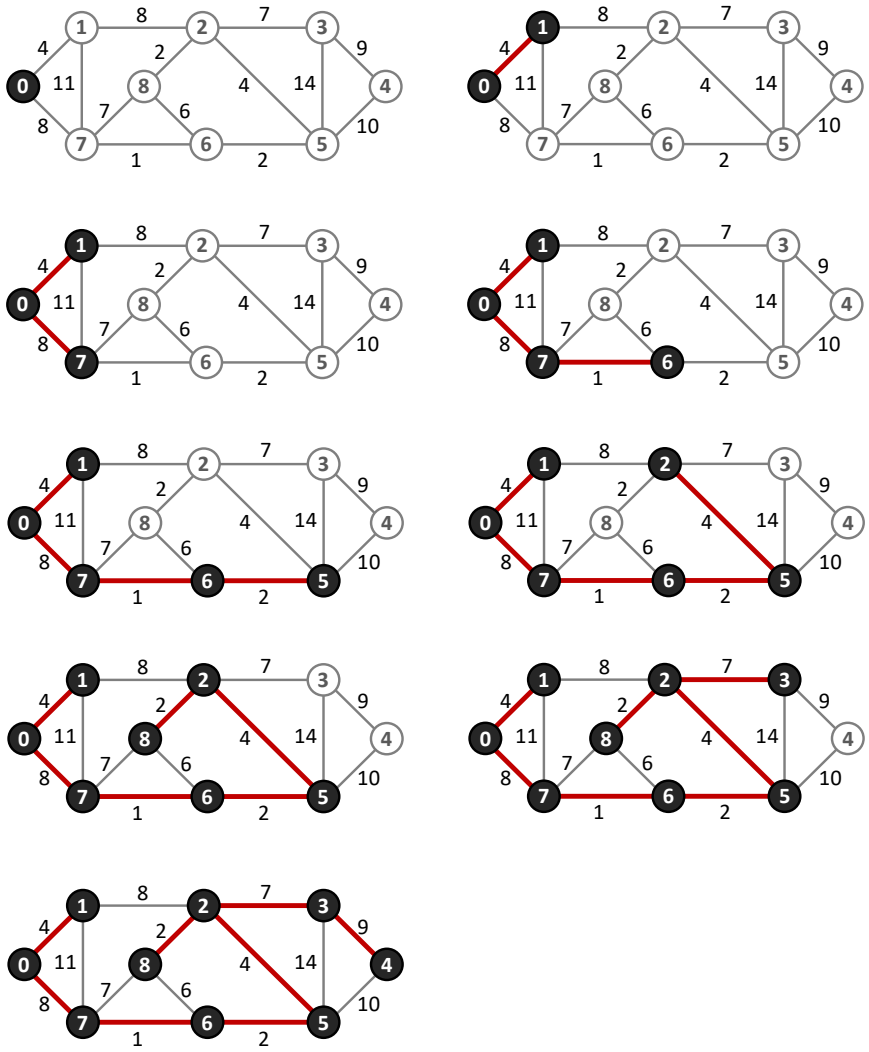
Algoritmo de Prim – Exemplo

```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6  |   Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7  |   S = S ∪ u;
8  |   N = N \ u;
9  |   T = T ∪ {v, u}
10 end
11 return T
```

O vértice 8 é selecionado, pois sua aresta possui o menor peso em relação as arestas de outros vértices de S {0, 1, 2} e vértices disponíveis de N.

Em seguida é selecionado vértice 5 (aresta 6–5), vértice 2 e vértice 8.

A construção da árvore continua no vértice de id 3, finalizando com a inclusão do vértice 4.



Algoritmo de Prim – Exemplo

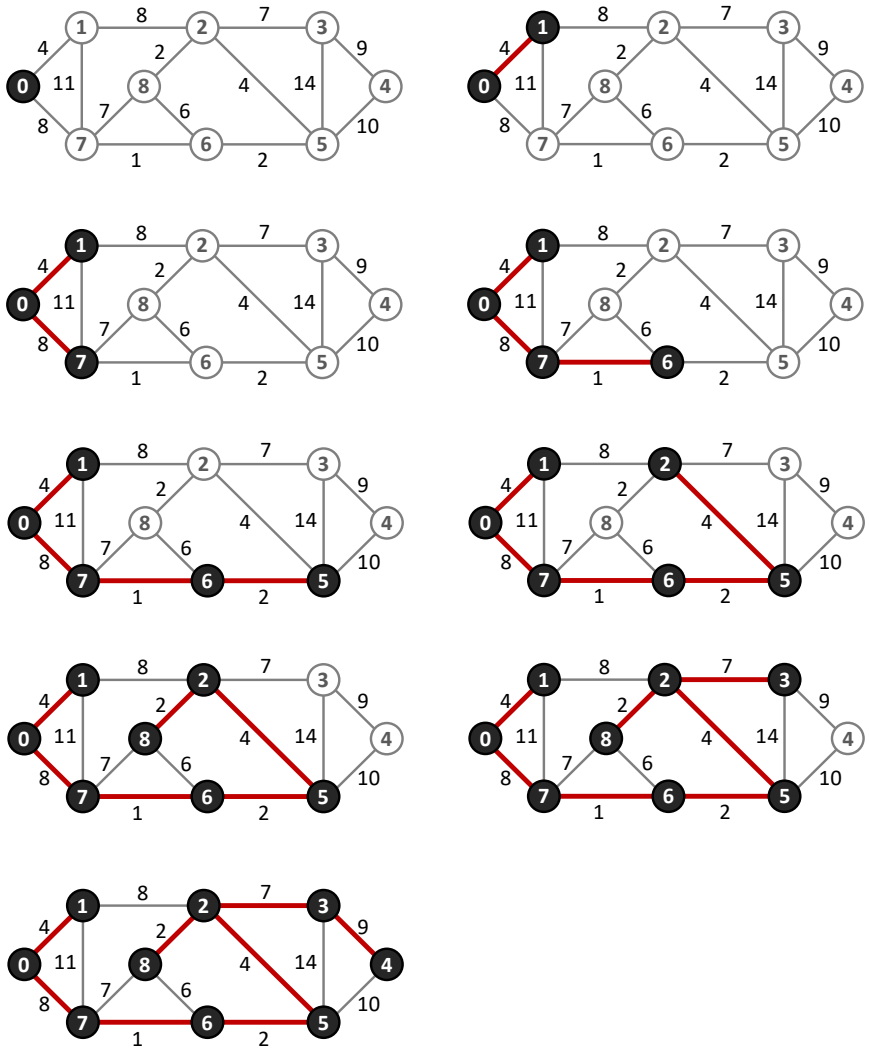
```
prim(G)
1  v = qualquer vértice ∈ V;
2  S = [v];
3  N = V \ v;
4  T = [];
5  while |T| < |V|-1 do
6  |   Obter aresta {v, u} ∈ E com min(w_vu), v ∈ S, u ∈ N;
7  |   S = S ∪ u;
8  |   N = N \ u;
9  |   T = T ∪ {v, u}
10 end
11 return T
```

Resultado:

T = {0, 1}, {0, 7}, {7, 6}, {6, 5}, {5, 2}, {2, 8}, {2, 3}, {3, 4}

$custo(T)$ = 37

Obs. É importante respeitar a ordem dos índices dos vértices origem (conjunto S) e destino (conjunto N), de modo a refletir o comportamento da busca.



Algoritmo de Kruskal

Proposto em 1956 por Joseph Bernard Kruskal Jr., estatístico, matemático, cientista da computação e psicometrista americano.

Princípio de Funcionamento:

- A cada iteração o algoritmo inclui em uma árvore geradora parcial (subárvore) a aresta de menor peso que não formar um ciclo, caso seja adicionada à subárvore.
- Realiza $|V| - 1$ iterações, que correspondem a quantidade de arestas da árvore geradora mínima.
- A estratégia do Algoritmo de Kruskal tem as arestas como perspectiva, sendo que a construção da árvore ocorre com a inclusão de arestas de menor custo. Diferente do algoritmo de Prim que se baseia nos vértices.

Algoritmo de Kruskal

```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v,u} ∈ H do
5  |   |   if T ∪ {v,u} ≠ ciclo then
6  |   |   |   T = T ∪ {v,u};
7  |   |   end
8  |   end
9  end
10 return T
```

"Heurística da Compressão de Caminhos" (função FIND-SET Cormen 2ª Edição, pág. 405)

ESTRUTURAS

- H : conjunto de arestas em ordem crescente de peso.
- T : conjunto de arestas que constituem a AGM.
- w_{vu} : valor do peso da aresta que conecta v e u.

Obter o conjunto de arestas em ordem crescente de peso (H) e criar o vetor que armazenará as arestas da árvore geradora mínima (linhas 1 e 2).

Vídeos:

- **Union Find in 5 minutes.**
  https://www.youtube.com/watch?v=ayW5B2W9hfo
- **Union-Find Algorithm (Detect Cycle)**
  https://www.youtube.com/watch?v=wQqFQeucFDc

Algoritmo de Kruskal

```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```

ESTRUTURAS

- H : conjunto de arestas em ordem crescente de peso.
- T : conjunto de arestas que constituem a AGM.
- w_{vu} : valor do peso da aresta que conecta v e u.

Obter o conjunto de arestas em ordem crescente de peso (H) e criar o vetor que armazenará as arestas da árvore geradora mínima (linhas 1 e 2).

Enquanto o tamanho do conjunto T for menor que a quantidade arestas da árvore geradora mínima, ou seja $|V| - 1$, (linha3):

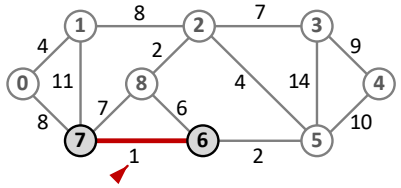
Para cada aresta $\{v, u\}$ pertencente ao conjunto ordenado de arestas (linha 4):

Se a adição da aresta $\{v, u\}$ não formar um ciclo na árvore mínima parcial T, então adicione a aresta $\{v, u\}$ no conjunto T (linhas 5 e 6).

Retornar T ao final da execução do algoritmo (linha10).

Algoritmo de Kruskal – Exemplo

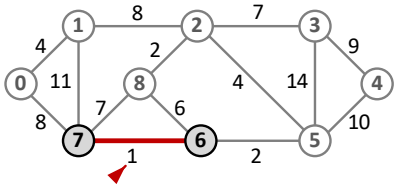
```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```

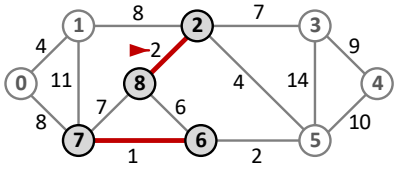


Após a ordenação das arestas em ordem crescente do seu peso, inicia pela aresta {6, 7} que possui o menor peso entre todas.

Algoritmo de Kruskal – Exemplo

```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```



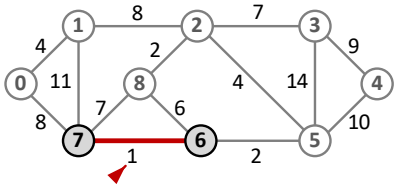


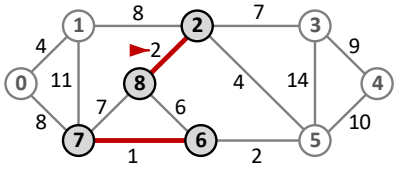
Após a ordenação das arestas em ordem crescente do seu peso, inicia pela aresta {6, 7} que possui o menor peso entre todas.

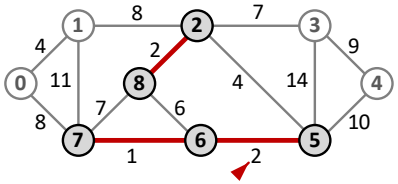
Depois é selecionada a aresta {2, 8}

Algoritmo de Kruskal – Exemplo

```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```





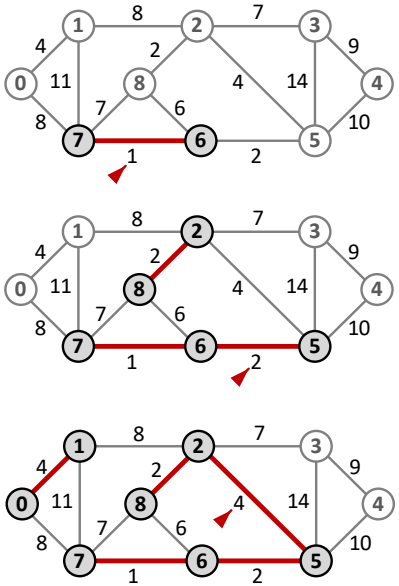


Após a ordenação das arestas em ordem crescente do seu peso, inicia pela aresta {6, 7} que possui o menor peso entre todas.

Depois é selecionada a aresta {2, 8}, seguida da aresta {5, 6}. Ambas têm o mesmo peso, sendo adicionadas a T pela ordem crescente do índice.

Algoritmo de Kruskal – Exemplo

```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```



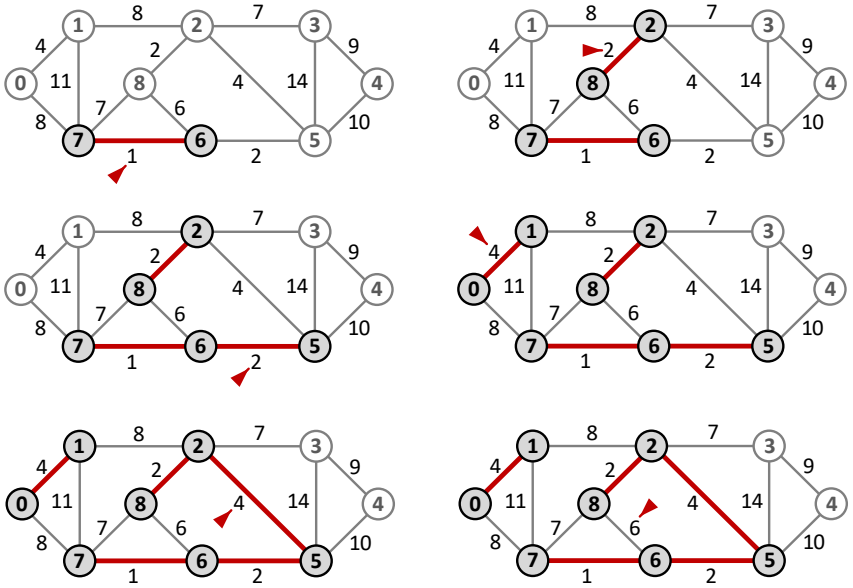
Após a ordenação das arestas em ordem crescente do seu peso, inicia pela aresta {6, 7} que possui o menor peso entre todas.

Depois é selecionada a aresta {2, 8}, seguida da aresta {5, 6}. Ambas têm o mesmo peso, sendo adicionadas a T pela ordem crescente do índice.

O método continua com a inclusão das arestas {0, 1} e {2, 5}, que têm o mesmo peso.

Algoritmo de Kruskal – Exemplo

```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```



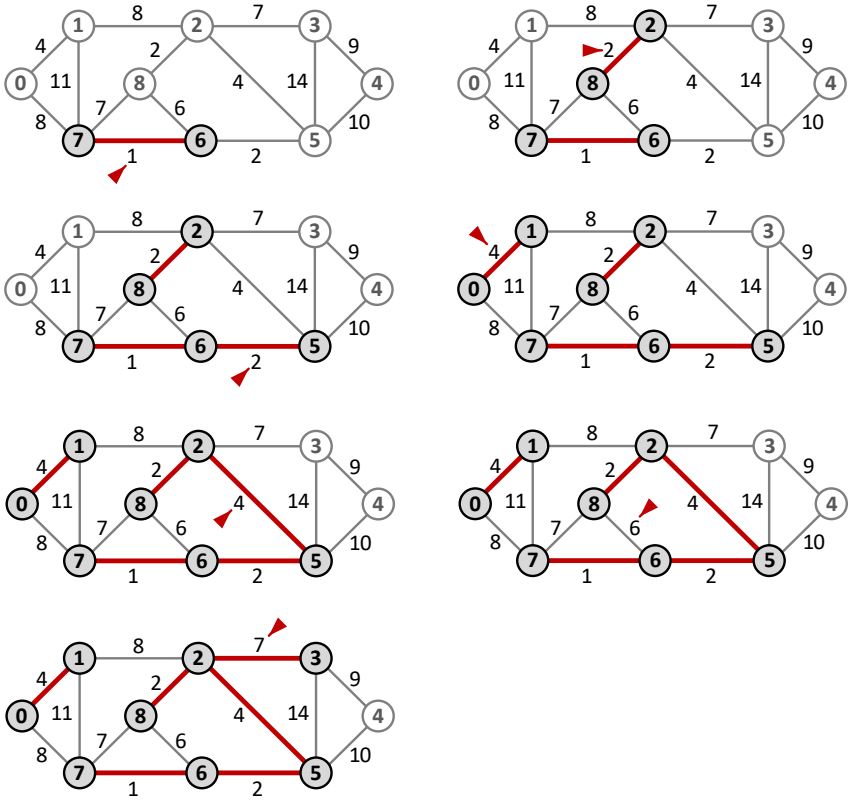
A aresta {6, 8} é analisada, mas não é escolhida pois sua inserção na árvore parcial geraria um ciclo.

Algoritmo de Kruskal – Exemplo

```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```

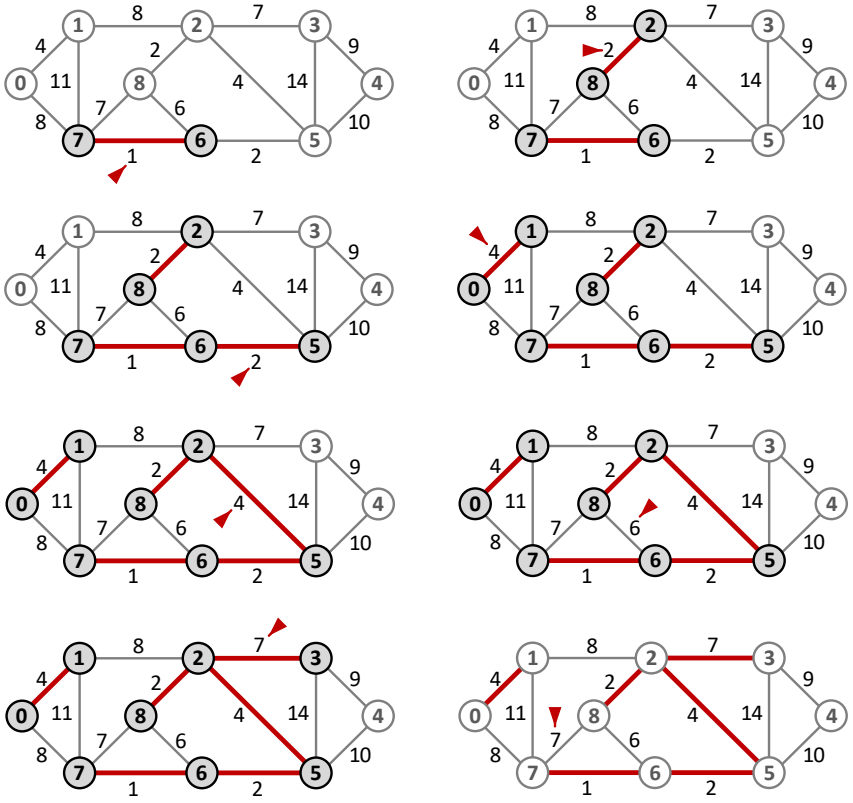
A aresta {6, 8} é analisada, mas não é escolhida pois sua inserção na árvore parcial geraria um ciclo.

A aresta {2, 3} é a próxima de menor peso que é viável de ser adicionada.



Algoritmo de Kruskal – Exemplo

```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```



A aresta {6, 8} é analisada, mas não é escolhida pois sua inserção na árvore parcial geraria um ciclo.

A aresta {2, 3} é a próxima de menor peso que é viável de ser adicionada.

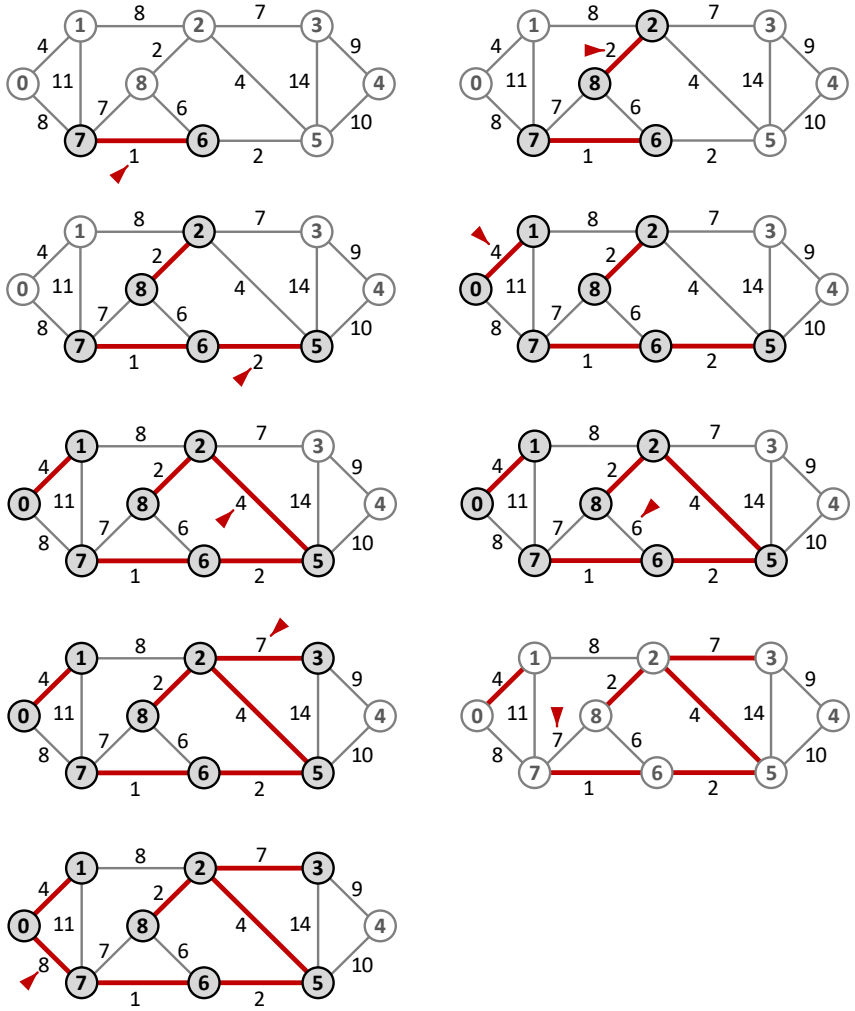
A aresta {7, 8} é analisada mas é descartada, pois sua inserção na árvore parcial ocasionaria em um ciclo.

Algoritmo de Kruskal – Exemplo

```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```

As arestas {0, 7} e {1, 2} possuem o mesmo peso, mas {0, 7} é escolhida pois é analisada primeiro devido o índice do vértice v.

A aresta {1, 2} é descartada pois sua inserção constituiria um ciclo na subárvore.

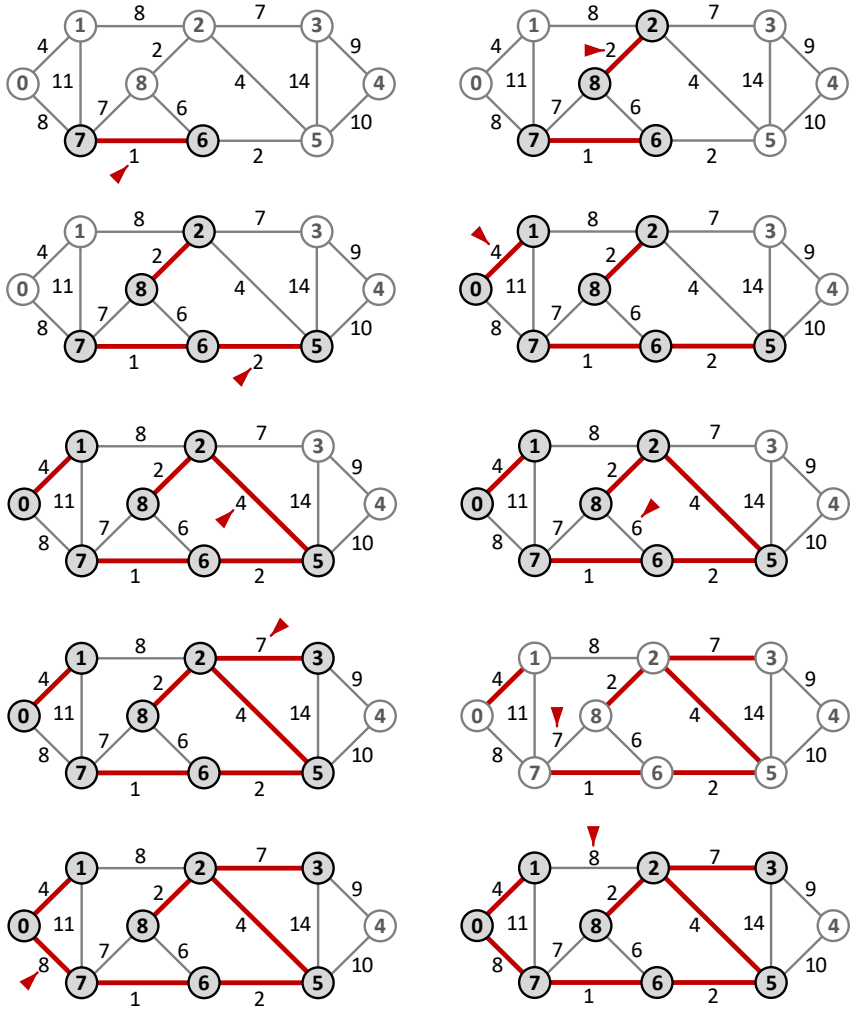


Algoritmo de Kruskal – Exemplo

```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```

As arestas {0, 7} e {1, 2} possuem o mesmo peso, mas {0, 7} é escolhida pois é analisada primeiro devido o índice do vértice v.

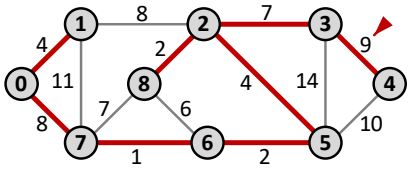
A aresta {1, 2} é descartada pois sua inserção constituiria um ciclo na subárvore.

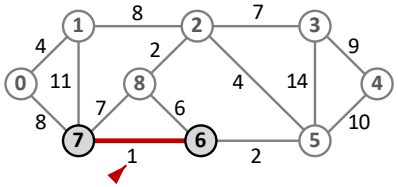


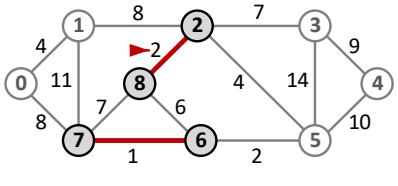
Algoritmo de Kruskal – Exemplo

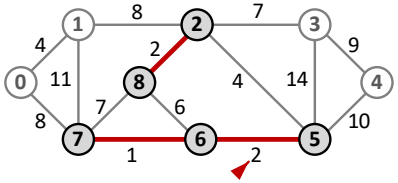
```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```

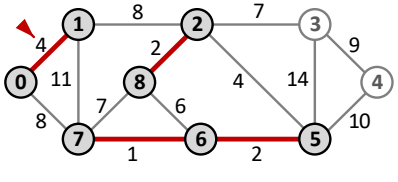
A aresta {3, 4} é adicionada a árvore.

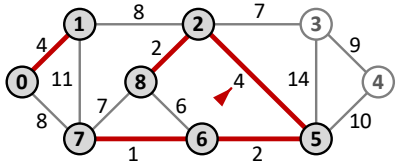


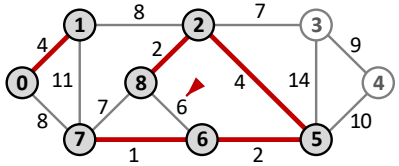


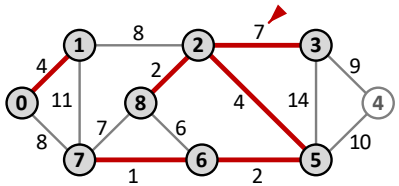


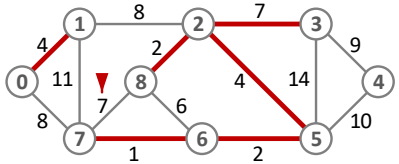


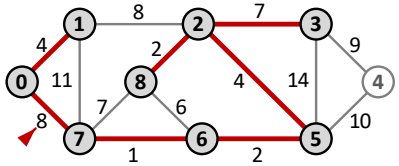


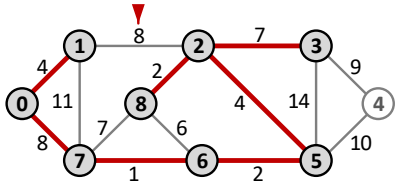










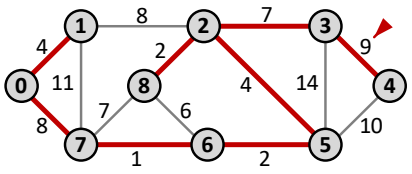


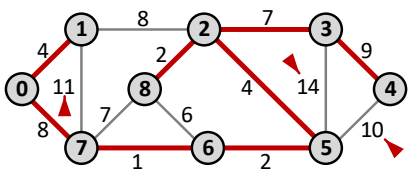
Algoritmo de Kruskal – Exemplo

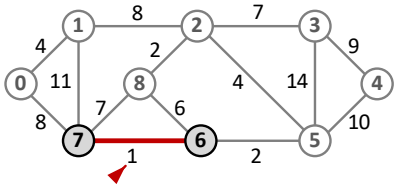
```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```

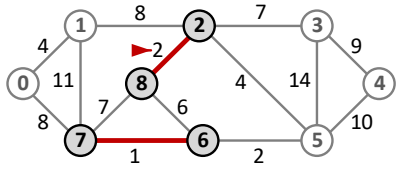
A aresta {3, 4} é adicionada a árvore.

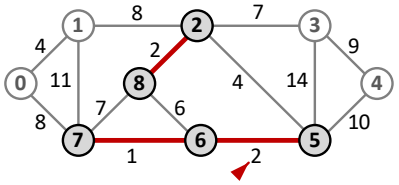
As arestas {4, 5}, {1, 7} e {3, 5} são analisadas nesta ordem, mas são descartadas por constituírem um ciclo caso inseridas na árvore.

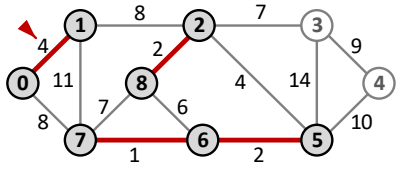


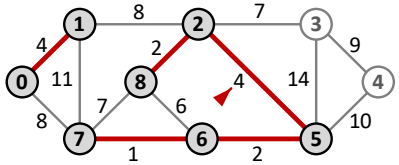


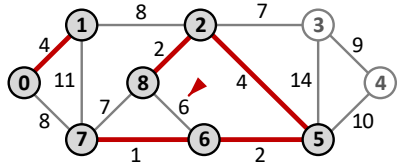


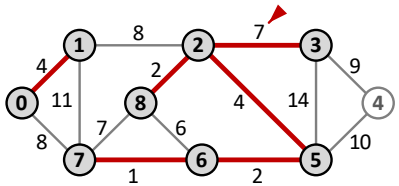


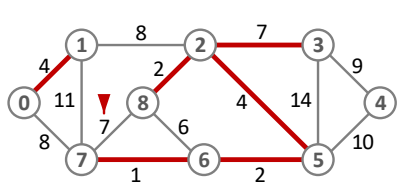


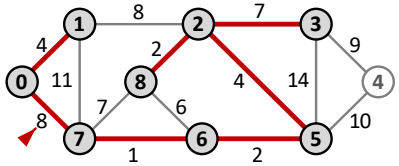


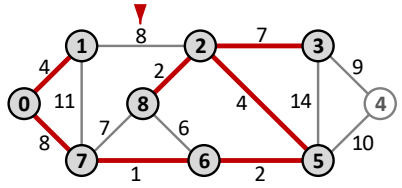






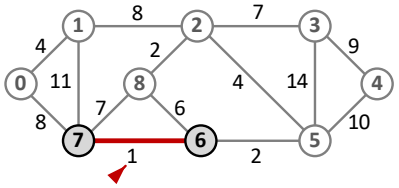


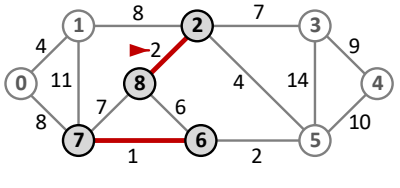


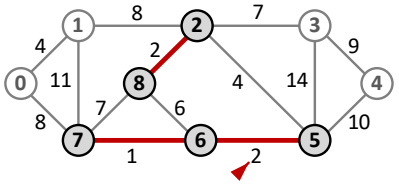


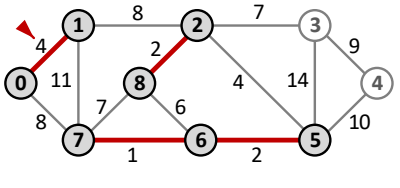
Algoritmo de Kruskal – Exemplo

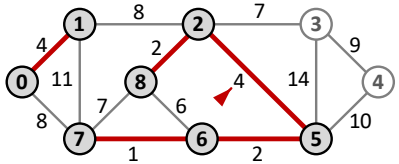
```
kruskal(G)
1  H = arestas ∈ E em ordem crescente de w_vu;
2  T = ∅;
3  while |T| < |V|-1 do
4  |   for {v, u} ∈ H do
5  |   |   if T ∪ {v, u} ≠ ciclo then
6  |   |   |   T = T ∪ {v, u};
7  |   |   end
8  |   end
9  end
10 return T
```

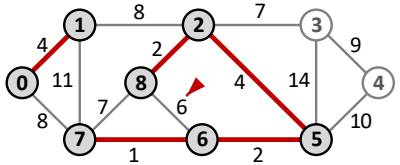


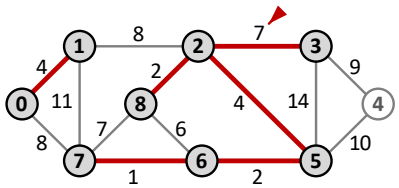


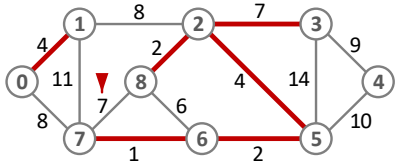


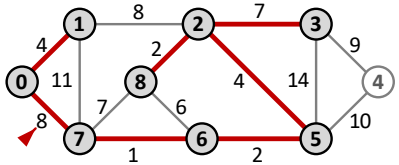


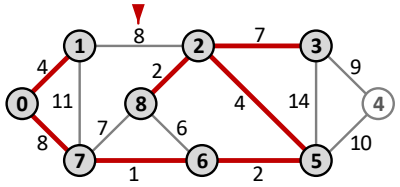










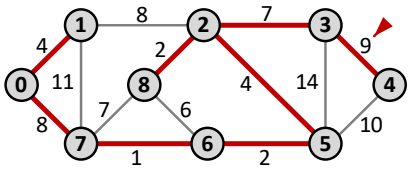


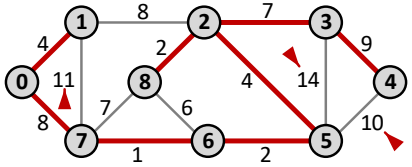
Resultado:

T = {6, 7}, {2, 8}, {5, 6}, {0, 1}, {2, 5}, {2, 3}, {0, 7}, {3, 4}

$custo(T)$ = 37

Obs. É importante respeitar a ordem dos índices, neste caso ficam na ordem crescente.





Comparação entre os Algoritmos

A implementação do algoritmo de Prim utilizando matriz de adjacências e uma busca linear na mesma tem complexidade $O(V^2)$. Melhorias:

- *Heaps* binárias - complexidade $O(E \log V)$
- *Heaps* de Fibonacci - complexidade $O(V \log V + E)$.

O algoritmo de Kruskal tem complexidade $O(E \log E)$.

Os algoritmos de Prim e Kruskal são variações da estratégia de Borůvka. O algoritmo de Borůvka tem complexidade $O(E \log E)$.

A estratégia gulosa é eficiente para o problema obtenção de árvores geradoras mínimas ou máximas, sendo que os três algoritmos possuem complexidade de tempo de ordem polinomial.

Perguntas? Sugestões?