O Algoritmo de Prim

O algoritmo de Prim é um método para encontrar a Árvore Geradora Mínima (MST) em um grafo ponderado e conexo.

O nome "Prim" no **algoritmo de Prim** vem do matemático e cientista da computação **Robert Clay Prim**, que publicou o algoritmo em 1957.

Prof. Patricia Plentz

O que é o Algoritmo de Prim?

Método Fundamental

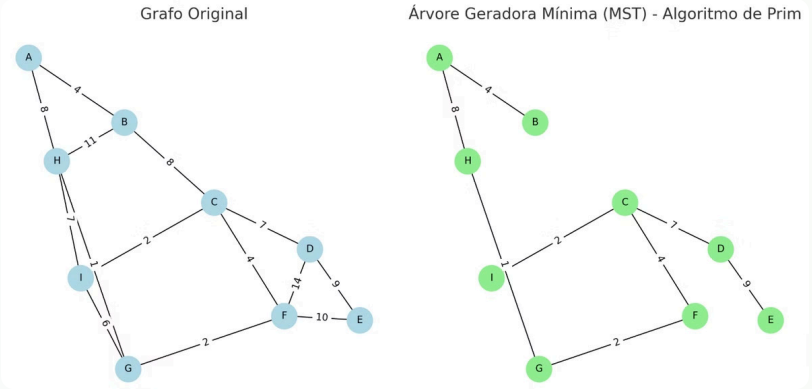
O **algoritmo de Prim** é um método utilizado na **Teoria dos Grafos** para encontrar a **Árvore Geradora Mínima (MST - Minimum Spanning Tree)** de um grafo conectado e com pesos nas arestas.

A MST é uma subárvore que conecta todos os vértices do grafo com o menor custo total possível, ou seja, a soma dos pesos das arestas é minimizada.

Abordagem Gulosa

Baseado na abordagem gulosa (greedy), o algoritmo conecta todos os vértices com o menor custo total possível.

Funcionamento do Algoritmo



1

Escolha Inicial

Escolha um vértice inicial arbitrário.

2

Menor Aresta

Encontre a menor aresta conectando o grafo atual a um novo vértice.

3

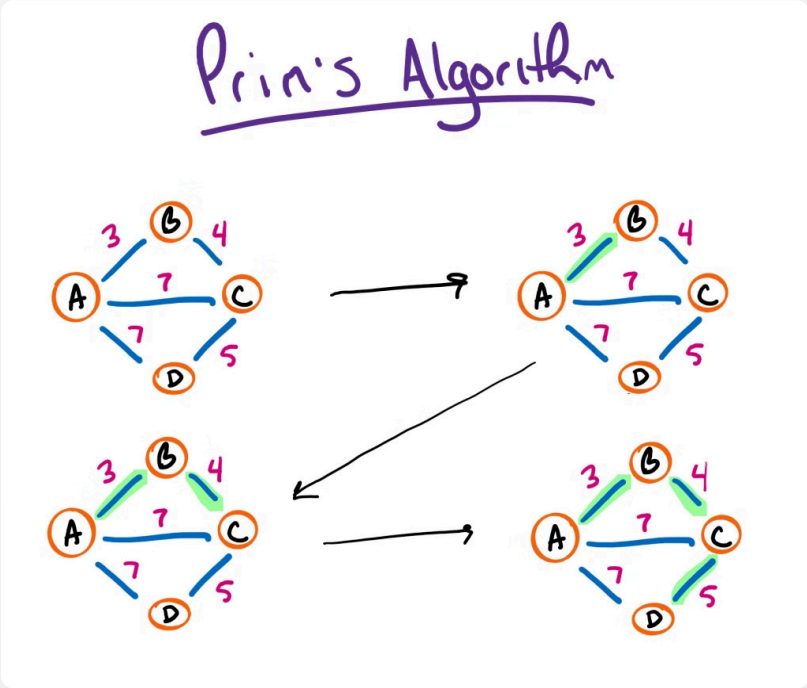
Adição

Adicione essa aresta e o vértice à árvore geradora.

4

Repetição

Repita o processo até que todos os vértices estejam conectados.



Pseudo-código

1. Escolha um vértice inicial arbitrário.
2. Inicialize a MST como um conjunto vazio.
3. Enquanto a MST não conectar todos os vértices:
   a. Escolha a menor aresta conectando a árvore a um novo vértice.
   b. Adicione a aresta e o vértice à árvore.
4. Retorne a MST.

Complexidade:

- O(E log V) com lista de adjacência e heap.
- O(V^2) com matriz de adjacência.

Empates no Algoritmo

Critérios de Desempate:

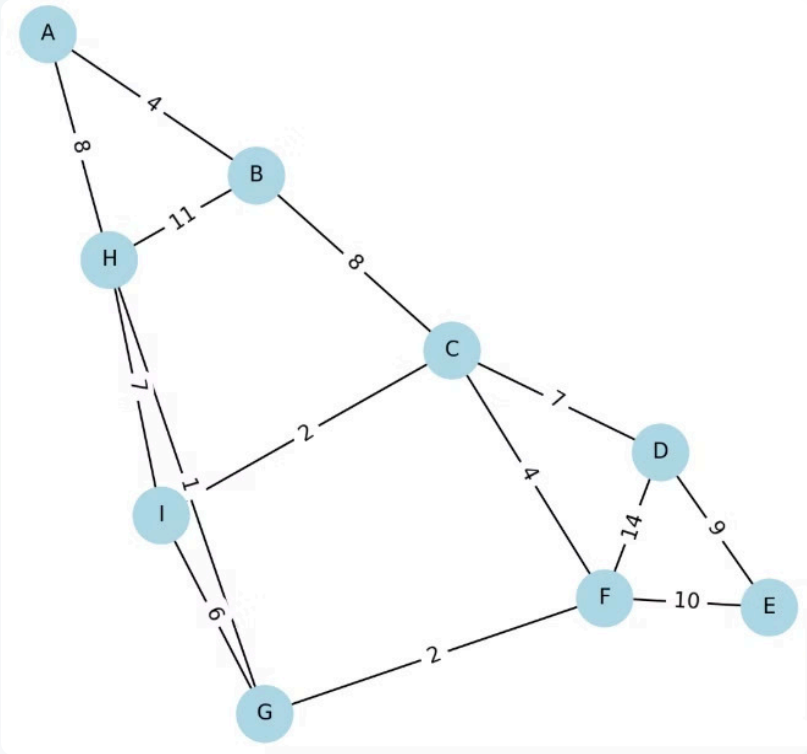
Ordem das arestas no grafo

Menor identificador do vértice

Escolha arbitrária entre as arestas com mesmo peso

Independente do critério usado:

O custo total da MST continuará sendo o mesmo.

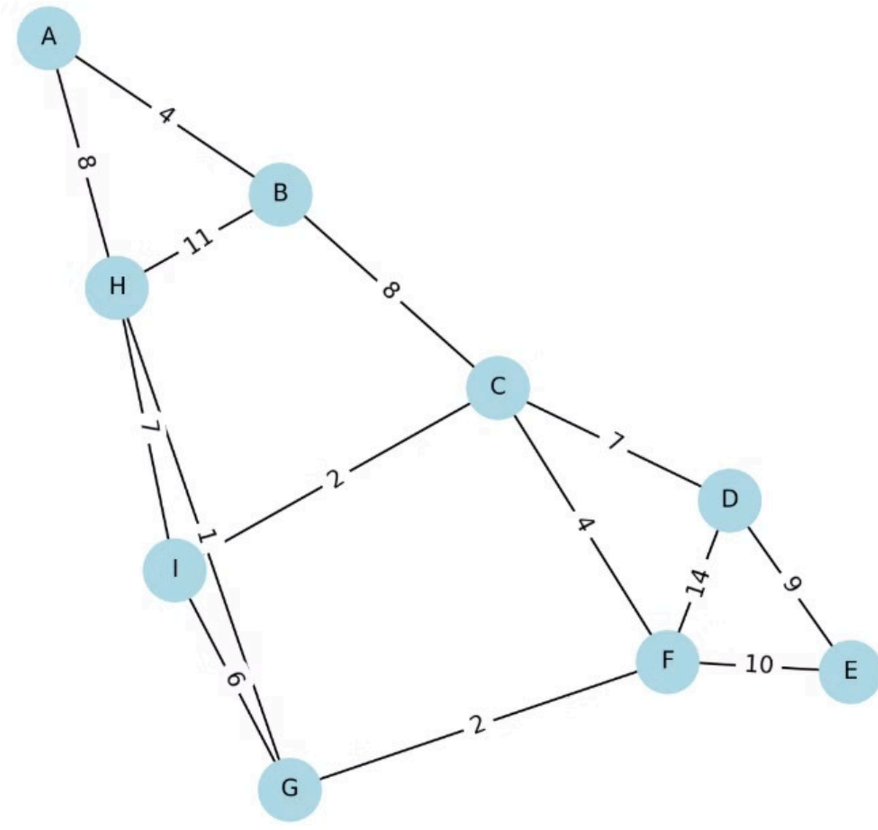


Exemplo:

O grafo possui os seguintes vértices e arestas com pesos:

A-B (4), A-H (8), B-H (11), B-C (8), C-D (7), C-I (2), C-F (4), D-E (9), D-F (14), E-F (10), F-G (2), G-I (6), G-H (1), H-I (7)

Nosso objetivo é encontrar a **Árvore Geradora Mínima (MST)** usando o algoritmo de Prim.



Passo-a-Passo do Algoritmo:

1

2

Passo 1: Escolher um vértice inicial

Escolhemos arbitrariamente o vértice **A**.

- Vértices na MST: **{A}**
- Arestas disponíveis a partir de A:
  - A-B (4), A-H (8)
- Menor aresta: **A-B (4)**

Passo 2: Adicionar a aresta de menor peso

Adicionamos a aresta **A-B** e incluímos o vértice **B**.

- Vértices na MST: **{A, B}**
- Arestas disponíveis:
  - B-C (8), B-H (11), A-H (8)
- Menor aresta: **A-H (8)**

Passo 3: Repetir o processo

3	4	5

Adicionamos a aresta **A-H** e incluímos o vértice **H**.
• Vértices na MST: **{A, B, H}**
• Arestas disponíveis:
  ◦ B-C (8), B-H (11), H-I (7), G-H (1)
• Menor aresta: **G-H (1)**
	Adicionamos a aresta **G-H** e incluímos o vértice **G**.
	• Vértices na MST: **{A, B, H, G}**
	• Arestas disponíveis:
	  ◦ B-C (8), H-I (7), G-I (6), F-G (2)
	• Menor aresta: **F-G (2)**
		Adicionamos a aresta **F-G** e incluímos o vértice **F**.
		• Vértices na MST: **{A, B, H, G, F}**
		• Arestas disponíveis:
		  ◦ B-C (8), H-I (7), G-I (6), C-F (4), D-F (14)
		• Menor aresta: **C-F (4)**

Passo 3: Repetir o processo

6

Adicionamos a aresta **C-F** e incluímos o vértice **C**.

- Vértices na MST: **{A, B, H, G, F, C}**
- Arestas disponíveis:
  - H-I (7), G-I (6), C-I (2), C-D (7)
- Menor aresta: **C-I (2)**

7

Adicionamos a aresta **C-I** e incluímos o vértice **I**.

- Vértices na MST: **{A, B, H, G, F, C, I}**
- Arestas disponíveis:
  - C-D (7), H-I (7)
- Menor aresta: **C-D (7)**

8

Adicionamos a aresta **C-D** e incluímos o vértice **D**.

- Vértices na MST: **{A, B, H, G, F, C, I, D}**
- Arestas disponíveis:
  - D-E (9)
- Menor aresta: **D-E (9)**

Adicionamos a aresta **D-E** e incluímos o vértice **E**.

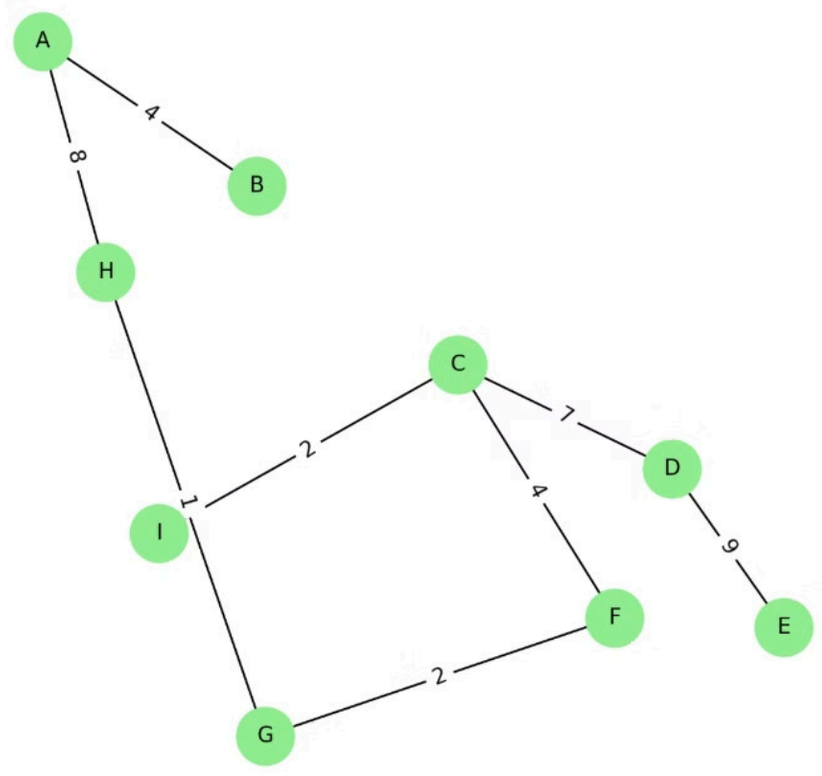
Passo Final: MST completa

Agora todos os vértices foram incluídos: **{A, B, C, D, E, F, G, H, I}**.

As arestas selecionadas para a MST foram: **A-B (4), A-H (8), G-H (1), F-G (2), C-F (4), C-I (2), C-D (7), D-E (9)**

Custo total da MST:

Soma dos pesos das arestas: **4 + 8 + 1 + 2 + 4 + 2 + 7 + 9 = 37**



Diferenças Fundamentais entre Dijkstra, Kruskal e Prim

Objetivos

Dijkstra:

- Encontra o **caminho mais curto de um vértice para todos os outros** em grafos direcionados ou não.
- A solução não é uma árvore, mas uma coleção de caminhos.

Kruskal:

- Encontra a **Árvore Geradora Mínima (MST)**.
- Conecta todos os vértices com o menor custo total, mas por meio de um método diferente[1].

Prim:

- Também encontra a **Árvore Geradora Mínima (MST)**.
- Conecta todos os vértices do grafo com o menor custo total.

Abordagem

Dijkstra:

- Expande caminhos partindo de um vértice inicial.
- Escolhe o próximo vértice com o menor custo acumulado (distância mínima).

Kruskal:

- Ordena todas as arestas do grafo por peso.
- Adiciona as menores arestas uma a uma, desde que não formem ciclos (usando conjuntos disjuntos ou *union-find*).

Prim:

- Expande uma árvore a partir de um único vértice inicial.
- Sempre escolhe a menor aresta conectando um vértice já na árvore com um fora dela.

Complexidade

Dijkstra:

- **O(E log V)** com fila de prioridade (simples com listas: O(V^2).
- Melhor para grafos **não negativos**, direcionados ou não.

Kruskal:

- **O(E log E)** devido à ordenação inicial das arestas.
- Melhor para **grafos esparsos**.

Prim:

- **O(E log V)** usando listas de adjacência e uma fila de prioridade.
- Melhor para **grafos densos**.

Estrutura de Resolução

Dijkstra:

- Constrói caminhos a partir do vértice inicial para todos os outros.

Kruskal:

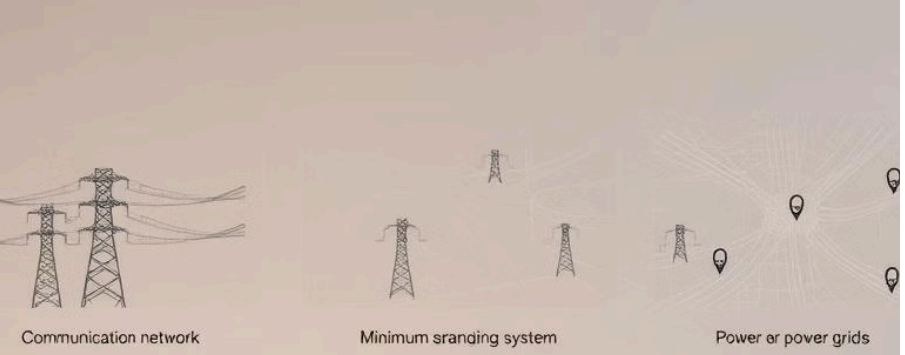
- Não depende de um vértice inicial.
- A construção da árvore se baseia na seleção global de arestas ordenadas.

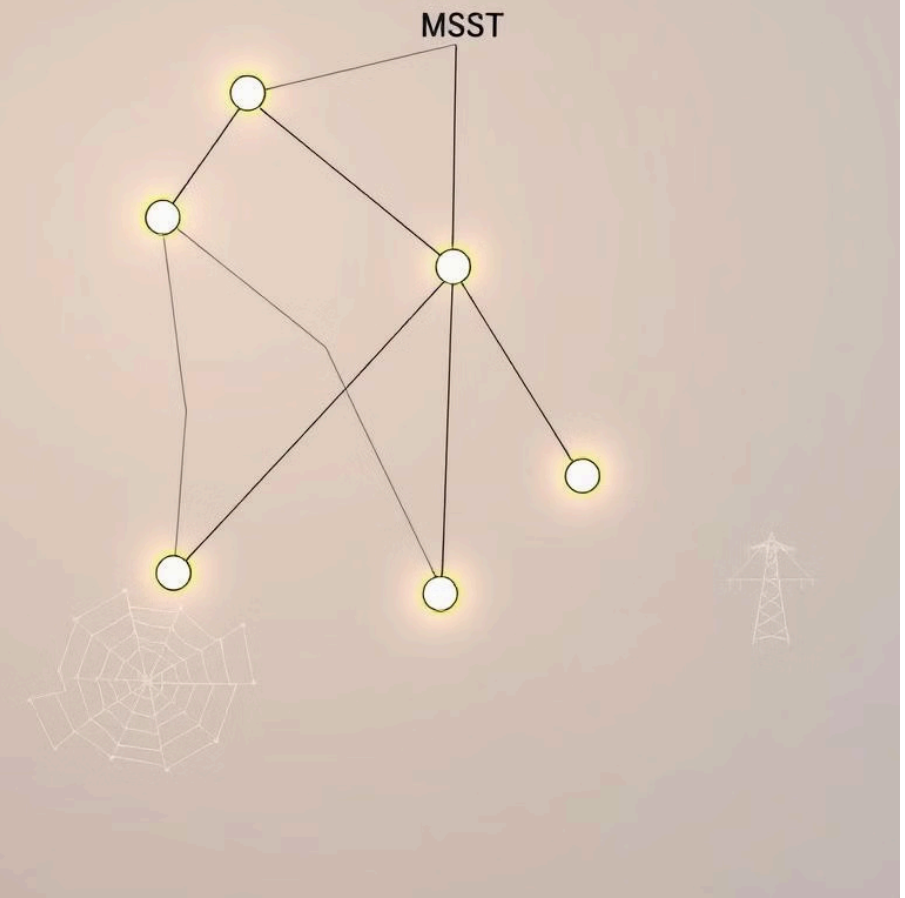
Prim:

- Funciona como uma construção incremental de uma árvore.
- Similar ao Dijkstra em termos de seleção de vértices próximos, mas com foco na árvore.

Comparação Resumida

Característica	Dijkstra	Kruskal	Prim
Objetivo	Caminho mais curto	Árvore Geradora Mínima	Árvore Geradora Mínima
Base de Seleção	Menor caminho acumulado	Menor aresta global	Menor aresta adjacente
Estrutura	Expansão de caminhos	Adição de arestas	Expansão de árvore
Entrada	Grafo com pesos não negativos	Grafo ponderado, conexo	Grafo ponderado, conexo
Complexidade	O(E log V)	O(E log E)	O(E log V)





Conclusão

Eficiência

O algoritmo de Prim é eficiente para encontrar MSTs em grafos ponderados e conexos.

Solução Válida

O algoritmo produz uma solução válida para o problema da Árvore Geradora Mínima.

Implementação

Todas as linguagens de programação oferecem estruturas e comandos para implementar o algoritmo.

Adaptação

Adapta-se bem a grafos densos, onde o número de arestas é alto em relação ao número de vértices.