Árvores Geradoras Mínimas: Algoritmos de Prim e Kruskal

Professor: Cleiton Silvano Goulart

09 de maio de 2024

1 Da forma de realização da atividade

- A atividade deve ser realizada em grupos de até 2 alunos.
- A atividade deverá ser entregue em formato digital, por meio do **diário-de-bordo** no AVA até o dia **16 de maio de 2024.**
- O diário-de-bordo deve conter:
  - O código desenvolvido.
  - Os resultados obtidos.
  - Uma breve descrição do que foi aprendido durante a atividade.

1. Fundamentação Teórica

A teoria das árvores geradoras mínimas (MST - Minimum Spanning Tree) é fundamental em diversas áreas da computação e engenharia, especialmente em problemas de otimização de redes, como redes de computadores, redes de transporte, distribuição de energia elétrica e até mesmo em bioinformática. O objetivo principal é conectar todos os pontos (vértices) de um grafo com o menor custo total possível, sem formar ciclos, garantindo assim a eficiência e a economia de recursos.

1.1 Grafo, Árvore e Árvore Geradora

Um **grafo** $G = (V, E)$ é uma estrutura composta por um conjunto de vértices V e um conjunto de arestas E que conectam pares de vértices. Quando cada aresta possui um valor associado (peso ou custo), dizemos que o grafo é **ponderado**. Uma **árvore** é um grafo conexo e acíclico, ou seja, não possui ciclos e existe exatamente um caminho entre quaisquer dois vértices. Uma **árvore geradora** de um grafo conexo é um subgrafo que inclui todos os vértices do grafo original e é uma árvore. Entre todas as árvores geradoras possíveis, a de menor custo total é chamada de **árvore geradora mínima** (MST).

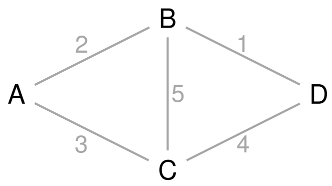
1.2 Aplicações das Árvores Geradoras Mínimas

As MSTs têm aplicações práticas em diversas áreas:

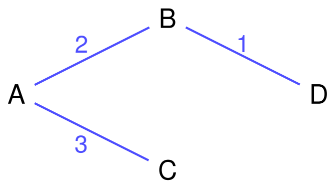
- **Redes de computadores:** Projetar a infraestrutura de cabeamento de uma rede local (LAN) minimizando o custo de instalação.
- **Redes de energia:** Planejar a distribuição de energia elétrica conectando subestações com o menor custo de fios.
- **Redes de transporte:** Construção de estradas, ferrovias ou dutos de forma eficiente.
- **Clusterização:** Em aprendizado de máquina, MSTs podem ser usadas para agrupar dados de acordo com distâncias mínimas.

1.3 Exemplo Ilustrativo de MST

Considere o grafo ponderado abaixo à esquerda. À direita, temos uma árvore geradora mínima (MST) equivalente, conectando todos os vértices com o menor custo total possível.



Grafo original



MST equivalente

Neste exemplo, a MST conecta todos os vértices (A, B, C, D) com custo total $2 + 3 + 1 = 6$, evitando ciclos e minimizando o custo.

1.3 Algoritmo de Prim

O algoritmo de Prim constrói a MST a partir de um vértice inicial, expandindo a árvore a cada passo pela aresta de menor peso que conecta um vértice já incluído a um vértice ainda não incluído. O algoritmo utiliza uma fila de prioridade (heap) para selecionar rapidamente a próxima aresta de menor custo. É eficiente para grafos densos, pois explora as conexões locais de cada vértice.

Pseudocódigo do Algoritmo de Prim:

PRIM-MST(G, w, r)
  para cada $u \in V[G]$ faça
    $key[u] \leftarrow \infty$
    $pai[u] \leftarrow$ NIL
  $key[r] \leftarrow 0$
  $Q \leftarrow V[G]$
  enquanto $Q \neq \emptyset$ faça
    $u \leftarrow$ EXTRACT-MIN(Q)
    para cada $v \in Adj[u]$ faça
      se $v \in Q$ e $w(u, v) < key[v]$ então
        $pai[v] \leftarrow u$
        $key[v] \leftarrow w(u, v)$

Notação Utilizada e Parâmetros:

- `G`: grafo ponderado de entrada.
- `w`: função que retorna o peso de uma aresta (u, v).
- `r`: vértice inicial para a construção da MST.
- `u, v`: vértices do grafo.
- `V[G]`: conjunto de vértices de G.
- `Adj[u]`: conjunto de vértices adjacentes a u (vizinhos de u).
- `key[v]`: menor peso de aresta que conecta v à árvore parcial.
- `pai[v]`: vértice anterior a v na MST (pai de v).
- `NIL`: valor nulo, indicando ausência de pai.
- `Q`: conjunto (ou fila de prioridade) dos vértices ainda não incluídos na MST.
- `EXTRACT-MIN(Q)`: remove e retorna o vértice de menor chave em `Q`.

O algoritmo de Prim garante que, a cada iteração, a árvore parcial construída é sempre uma subárvore de uma MST do grafo original.

1.4 Algoritmo de Kruskal

O algoritmo de Kruskal constrói a MST selecionando as arestas de menor peso, em ordem crescente, e adicionando-as à árvore se não formarem ciclos. Para detectar ciclos de forma eficiente, utiliza-se a estrutura de dados Union-Find (ou Disjoint Set). Kruskal é especialmente eficiente para grafos esparsos, pois trabalha diretamente sobre as arestas.

Pseudocódigo do Algoritmo de Kruskal:

KRUSKAL-MST(G, w)
  $A \leftarrow \emptyset$
  para cada vértice $v \in V[G]$ faça
    MAKE-SET(v)
  ordene as arestas de E por peso w
  para cada aresta $(u, v) \in E$ (em ordem) faça
    se FIND-SET(u) $\neq$ FIND-SET(v) então
      $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
  retorne A

Notação Utilizada e Parâmetros:

- `G`: grafo ponderado de entrada.
- `w`: função que retorna o peso de uma aresta (u, v).
- `A`: conjunto de arestas da árvore geradora mínima.
- `u, v`: vértices do grafo.
- `V[G]`: conjunto de vértices de G.
- `E`: conjunto de arestas de G.
- `MAKE-SET(v)`: cria um novo conjunto contendo apenas o vértice v (Union-Find).
- `FIND-SET(u)`: retorna o representante do conjunto ao qual u pertence (Union-Find).
- `UNION(u,v)`: une os conjuntos de u e v (Union-Find).

O algoritmo de Kruskal pode ser implementado de forma eficiente usando Union-Find com compressão de caminho, tornando-o adequado para grafos com muitas arestas dispersas.

1.5 Comparação entre Prim e Kruskal

Critério	**Prim**	**Kruskal**
Estrutura de Dados	Heap/Fila de Prioridade	Union-Find
Complexidade	$O(E \log V)$	$O(E \log E)$
Melhor para	Grafos densos	Grafos esparsos
Implementação	Mais simples	Requer estrutura Union-Find
Memória	$O(V)$	$O(E)$

Resumo: Ambos os algoritmos garantem a obtenção de uma árvore geradora mínima, mas a escolha entre eles depende das características do grafo e dos requisitos de implementação. Prim é preferido para grafos densos e quando se deseja construir a MST a partir de um vértice específico. Kruskal é mais eficiente para grafos esparsos e quando se deseja flexibilidade na escolha das arestas.

2. Situação-Problema

Imagine que você é um engenheiro de software responsável pelo desenvolvimento de um sistema de gerenciamento de infraestrutura de rede para uma empresa de telecomunicações. A empresa precisa otimizar o custo de manutenção de sua rede de fibra óptica, que conecta diversos pontos de presença (PoPs) em uma região metropolitana.

Cada PoP representa um vértice no grafo, e as possíveis conexões de fibra óptica entre eles são as arestas, com pesos representando o custo de instalação e manutenção anual de cada conexão.

Sua tarefa é desenvolver um sistema que:

1. Permita a entrada da topologia da rede (grafo) com seus respectivos custos;
2. Implemente os algoritmos de Prim e Kruskal para encontrar a árvore geradora mínima;
3. Compare os resultados obtidos pelos dois algoritmos;
4. Sugira a melhor solução considerando:
   - Custo total de implementação
   - Resiliência da rede
   - Possibilidade de expansão futura

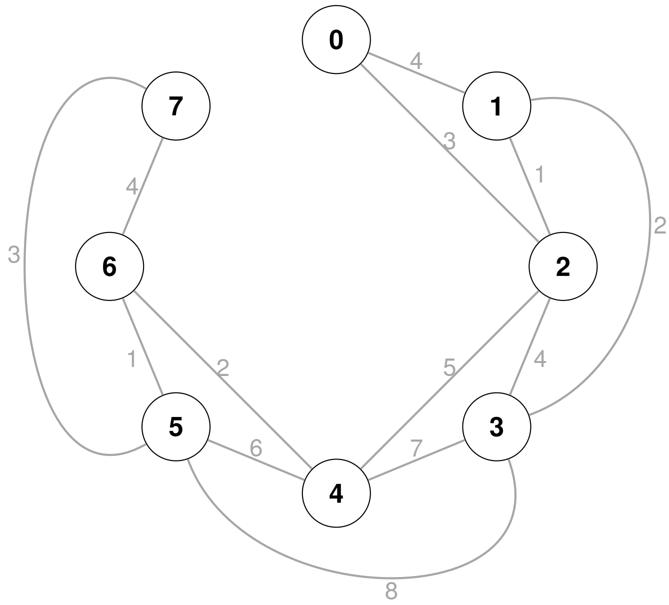
Formato da entrada de dados:

- A primeira linha contém dois inteiros: n (número de PoPs) e m (número de possíveis conexões);
- As próximas m linhas contêm três inteiros u, v e w por linha, representando uma possível conexão entre os PoPs u e v com custo w.

Exemplo de entrada:

```
8 13
0 1 4
0 2 3
1 2 1
1 3 2
2 3 4
2 4 5
3 4 7
3 5 8
4 5 6
4 6 2
5 6 1
5 7 3
6 7 4
```

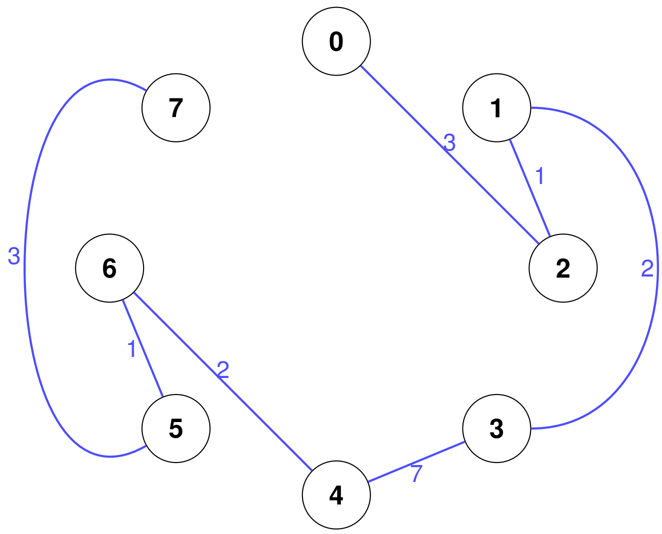
Grafo correspondente:



Entrada de referência:

```
8 13
0 1 4
0 2 3
1 2 1
1 3 2
2 3 4
2 4 5
3 4 7
3 5 8
4 5 6
4 6 2
5 6 1
5 7 3
6 7 4
```

Árvore Geradora Mínima (MST):



Peso total da MST (Prim/Kruskal): 19

Saída esperada (Prim ou Kruskal):

```
Aresta 1-2 com peso 1
Aresta 5-6 com peso 1
Aresta 1-3 com peso 2
Aresta 4-6 com peso 2
Aresta 0-2 com peso 3
Aresta 5-7 com peso 3
Aresta 3-4 com peso 7

Peso total da MST: 19
```