UNIVERSIDADE ESTADUAL DE SANTA CRUZ - UESC

LUIZ FELIPE DO ROSÁRIO ALVES SILVA

TEORIA DOS GRAFOS - ALGORITMO DE PRIM PARA ÁRVORES GERADORAS MÍNIMAS

ILHÉUS – BAHIA

2023

LUIZ FELIPE DO ROSÁRIO ALVES SILVA

TEORIA DOS GRAFOS - ALGORITMO DE PRIM PARA ÁRVORES GERADORAS MÍNIMAS

Trabalho apresentado ao Curso de Graduação em Ciência da Computação da Universidade Estadual de Santa Cruz, como um dos pré-requisitos da disciplina Projeto e Análise de Algoritmos.

Orientadora: Prof.ª Marta Magda Dornelles Bertoldi

ILHÉUS – BAHIA

2023

TEORIA DOS GRAFOS - ALGORITMO DE PRIM PARA ÁRVORES GERADORAS MÍNIMAS

RESUMO

Este projeto tem como objetivo fazer uma análise dos principais algoritmos que envolvem a teoria dos grafos. Nele, abordaremos a utilização, eficiência e utilidade destes 5 algoritmos: Prim, Kruskal, Dijkstra, Bellman-Ford e Floyd-Warshall. Mais a frente no projeto, haverá um aprofundamento no algoritmo de Prim, explicando sua história, sua funcionalidade, apresentando exemplos práticos de seu uso, implementação e discutir um pouco sobre os resultados obtidos.

INTRODUÇÃO

- Algoritmo de Prim

Utilizado para retornar um Árvore Geradora Mínima (AGM). Tem como entrada um grafo não direcionado ponderado. Então, a partir de um vértice arbitrário, o coloca dentro da árvore. Após isso, busca de forma gulosa a aresta com menor peso que conecta um vértice de fora da árvore com um de dentro. E, acrescenta na árvore o vértice da ponta de fora. Onde, no final, resultará na AGM.

Sua complexidade é $O(V^2)$ ou $O(V * \log V + A * \log V)$, a depender da forma de implementação.

- Algoritmo de Kruskal

Assim como o de Prim, retorna uma AGM e também tem como entrada o mesmo tipo de grafo. Parte da menor aresta que não crie um ciclo. Então escolhe a menor aresta que não crie um ciclo. De forma que nunca seja escolhida uma aresta que conecta 2 nós que estão na mesma árvore, e então retorna uma árvore geradora mínima.

Sua complexidade é $O(A * \log A)$.

- Algoritmo de Dijkstra:

Retorna **o** menor caminho entre o nó escolhido e todos os outros nós. Cria uma tabela de distância com todos os nós, coloca 0 no nó inicial e infinito em todos os outros. Que significa que não foram visitados. Então, analisamos as arestas saindo do nó inicial e

atualizamos o valor da tabela com os valores das distâncias. Em seguida, pegamos a menor aresta que leva a um nó não analisado. Assim, analisamos este nó, atualizando a tabela caso haja novos nós disponíveis através dele. E, caso já tenha valor neste nó, atualizar se a soma das arestas até ele resultar em um valor menor.

Sua complexidade é O(A + V * log V).

- O algoritmo de Bellman-Ford

É usado para encontrar o caminho mais curto em um grafo ponderado, considerando a possibilidade de existirem arestas com pesos negativos. Ele recebe como entrada um grafo ponderado e um vértice de origem. O algoritmo realiza relaxamentos nas arestas repetidamente, atualizando as distâncias dos vértices adjacentes. Ele também verifica se há ciclos de peso negativo no grafo. Ao final, retorna as distâncias mínimas do vértice de origem para todos os outros vértices.

Sua complexidade é O(V * A).

- O algoritmo de Floyd-Warshall

Encontra os menores caminhos entre todos os pares de vértices em um grafo ponderado, considerando até mesmo arestas com pesos negativos. Ele utiliza uma matriz de distâncias inicializada com os pesos das arestas do grafo. O algoritmo realiza iterações para calcular as distâncias mínimas através de vértices intermediários e atualiza a matriz de distâncias sempre que encontrar um caminho menor. Ele também verifica a existência de ciclos negativos no grafo.

Sua complexidade é O(V^3).

PRIM

O algoritmo de Prim é um algoritmo guloso utilizado para encontrar uma Árvore Geradora Mínima (AGM) em um grafo conexo, não direcional e ponderado. Sua finalidade é encontrar um subgrafo do grafo original no qual a soma total das arestas é a mínima e todos os vértices estão interligados.

Foi desenvolvido pelo matemático Vojtěch Jarník em 1930 e, então, redescoberto pelo cientista da computação Robert Clay Prim em 1957. Ironicamente, também foi redescoberto

por Edsger Dijkstra em 1959. Comparado a outros algoritmos de AGM, como o algoritmo de Kruskal, o algoritmo de Prim requer que o grafo seja conexo.

O algoritmo de Prim encontra uma solução ótima para o problema de encontrar uma AGM em um grafo ponderado e não direcionado. É utilizado em situações em que os vértices representam cidades e as arestas representam estradas que interligam essas cidades. Nesse contexto, o algoritmo de Prim pode ser aplicado para determinar quais estradas devem ser asfaltadas, buscando gastar a menor quantidade de asfalto possível para interligar todas as cidades.

O algoritmo de Prim requer que o grafo seja conexo, ou seja, que todos os vértices estejam interligados. Caso o grafo não seja conexo, o algoritmo de Prim precisaria ser aplicado em cada componente conexo separadamente. E também é aplicável apenas a grafos não direcionados, onde todas as arestas têm pesos atribuídos.

A complexidade dos algoritmos podem variar. O algoritmo de Prim tem uma complexidade de tempo de O(V^2) quando usa uma matriz de Adjacência ou busca, ou, O(A log V), em um heap ou lista de adjacência, onde V é o número de vértices e A é o número de arestas no grafo. Por outro lado, o algoritmo de Kruskal tem uma complexidade de tempo de O(A log A), onde A é o número de arestas no grafo.

FERRAMENTAS E MÉTODOS

A linguagem escolhida foi java. Pelo fato de já ter conhecimento prévio com esta linguagem e também já ter trabalhado com grafos nela.

Como estruturas foram utilizadas matrizes, já que a entrada é em uma matriz de adjacência. E também, vetores do tamanho da quantidade de vértices para os vértices-pais, pesos e para separar os dentro da AGM dos de fora.

Já para a criar o exemplo de grafo foi utilizado o Figma. Enquanto a plotagem do gráfico foi realizada no Canva.

As especificações do computador utilizado para os dados são:

- Processador: Intel Core 3470;
- Placa de Vídeo: GT 710;
- Placa-Mãe: H61 DDR3;

- 8gb RAM;
- Sistema operacional: Windows 10;

O ALGORITMO

Pseudocódigo

Entrada:

Uma matriz de adjacência que representa um grafo não direcionado e conexo G = (V, A) com pesos em suas arestas;

Saída:

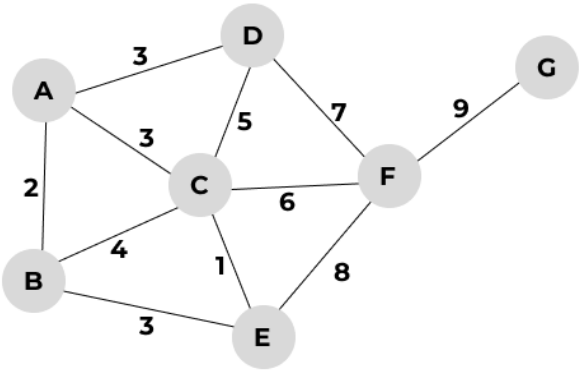
Uma Árvore Geradora Mínima definida por pares de vértices representando arestas e seus pesos;

```
Prim(Adj[][])
   n = tamanho do grafo Adj
   pai[] = novo vetor de tamanho n
   peso[] = novo vetor de tamanho n
   arvoreAGM[] = novo vetor de tamanho n

   para cada vértice v do grafo
      peso[v] = infinito
      arvoreAGM[v] = falso

   peso[0] = 0
   pai[0] = -1

   para i de 0 até n - 1
      u = pesoMin(peso, arvoreAGM)
      arvoreAGM[u] = verdadeiro

      para cada vértice v do grafo
         se Adj[u][v] ≠ 0 e arvoreAGM[v] = falso e Adj[u][v] < peso[v]
            pai[v] = u
            peso[v] = Adj[u][v]

   printAGM(pai, Adj)
```

Exemplo de funcionamento

1- Primeiro, temos nosso grafo e escolhemos um vértice arbitrário:

(Neste caso, o A).



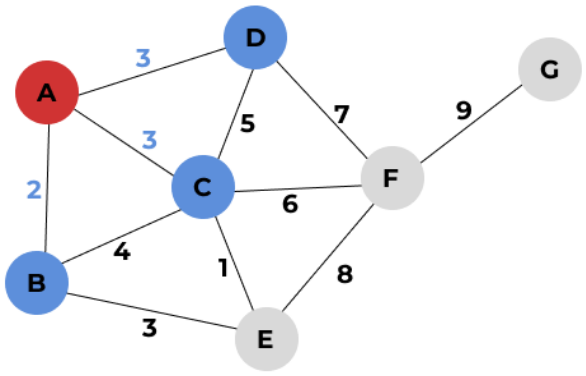
2- Logo em seguida, temos na nossa árvore AGM o A.

Aqui representaremos os vértices True no código como dentro da árvore.

Árvore AGM: {A} (Em vermelho)

pai: {-1, , , , , }

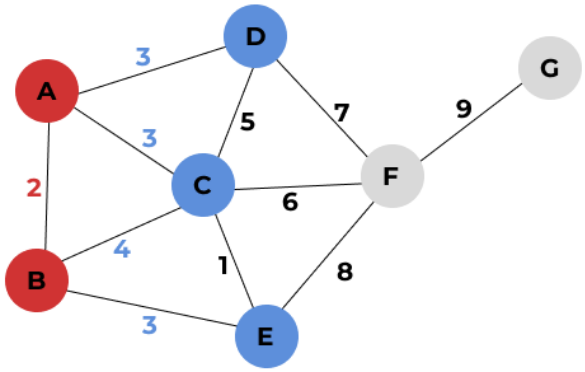
E também temos os vértices disponíveis. (Em azul)



3- Então, buscaremos a aresta de valor mínimo que leva um vértice de dentro da árvore a um de fora. Neste caso, AB

Árvore AGM: {A, B}

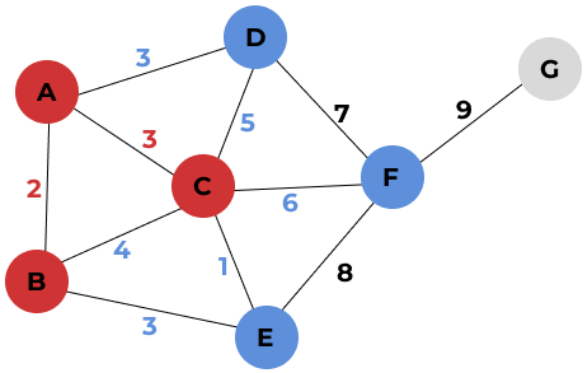
pai: {-1, 0, , , , }



4- Temos valores iguais, podemos escolher qualquer um deles, escolhemos o AC.

Árvore AGM: {A, B, C}

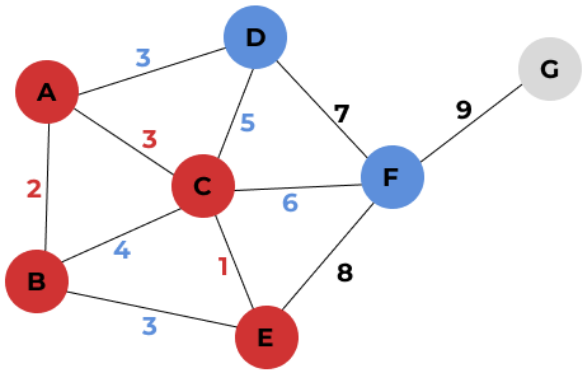
pai: {-1, 0, 0, , , }



5- Então CE.

Árvore AGM: {A, B, C, E}

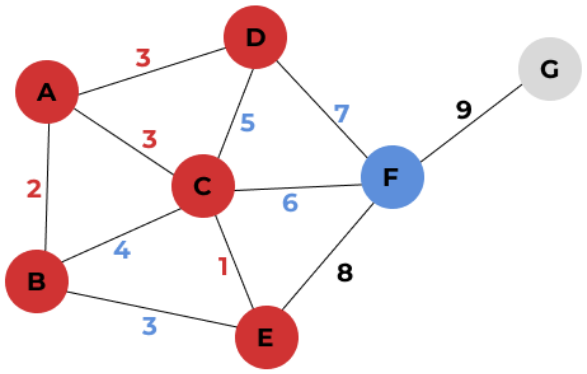
pai: {-1, 0, 0, , 2, }



6- Então AD

Árvore AGM: {A, B, C, E, D}

pai: {-1, 0, 0, 0, 2, }

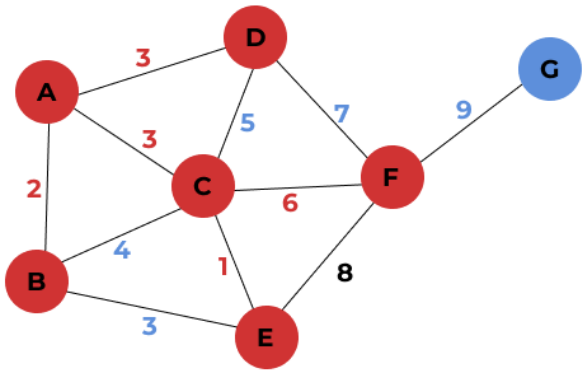


7- Nota-se que há arestas que possuem valor menor, porém, ligam vértices que já estão na árvore. O que não buscamos.

Logo, acrescentamos CF

Árvore AGM: {A, B, C, E, D, F}

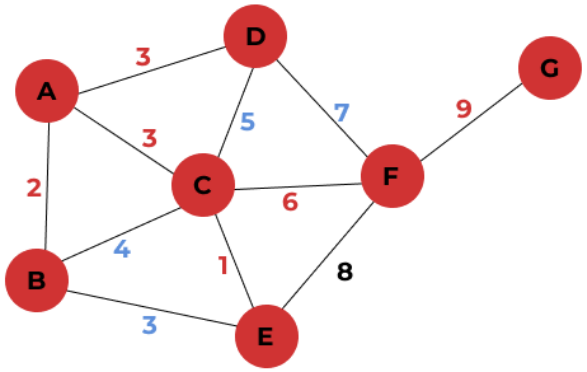
pai: {-1, 0, 0, 0, 2, 3, }



8- Finalizando com FG

Árvore AGM: {A, B, C, E, D, F, G}

pai: {-1, 0, 0, 0, 2, 3, 5}



9 - Na saída teremos

pai: {-1, 0, 0, 0, 2, 3, 5} onde temos a relação vértices e pais, ou predecessores.

Adj [i][pai[i]] e aqui onde buscaremos o valor de cada aresta.

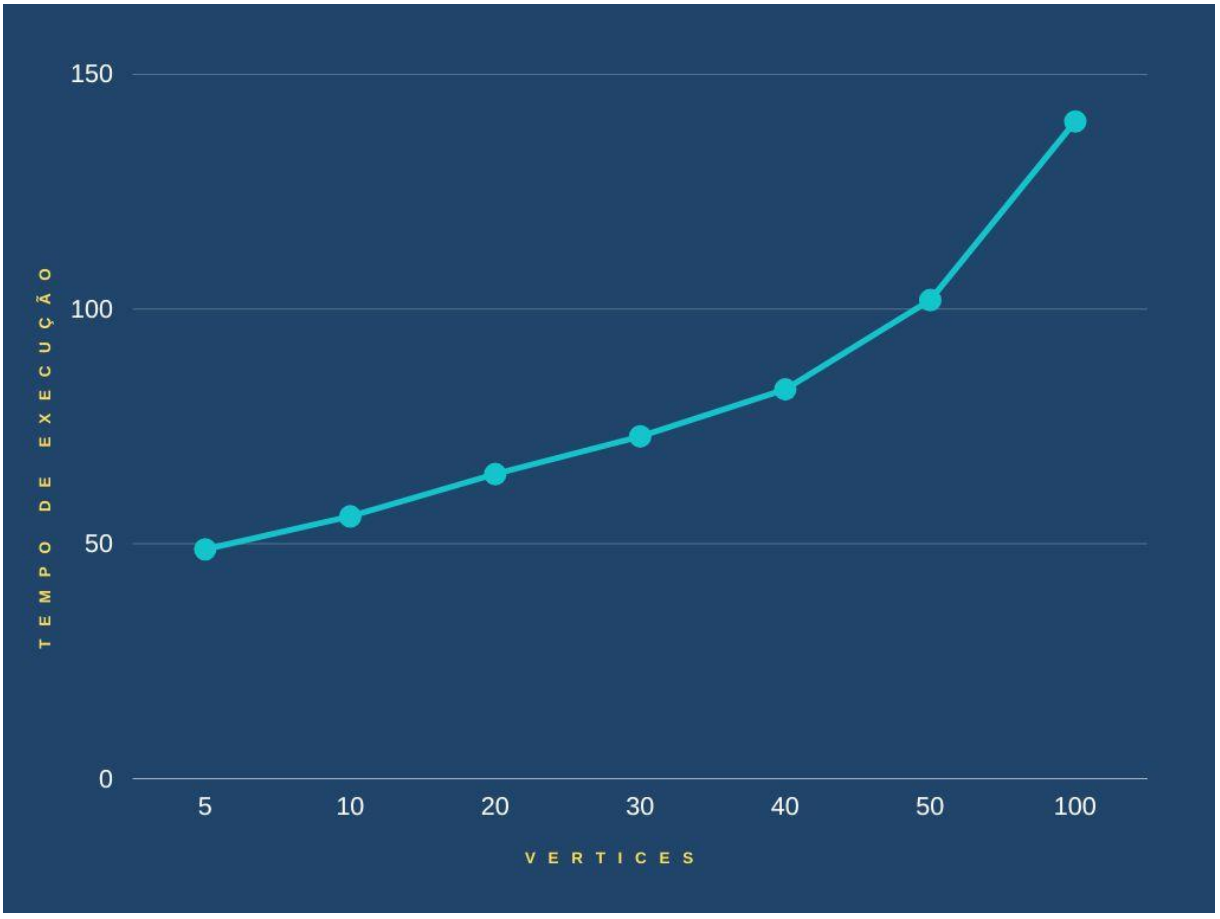
Arestas	Peso
0 - 1	2
0 - 2	3
0 - 3	3
2 - 4	1
2 - 5	6
5 - 6	9
Total:	24

COMPLEXIDADE DE TEMPO DO ALGORITMO

Para complexidade, já pela leitura da matriz de adjacência, temos um $O(n^2)$, sendo n o número de vértices. Na inicialização dos vetores auxiliares pai[], peso[] e arvoreAGM[] temos O(3n). E em acréscimo, temos um loop externo O(n) e um interno O(n), resultando em $O(n^2)$. No total, temos $O(2n^2 + 3n + C)$, sendo C as constantes do código que serão desprezadas no final. Podemos então dizer que a complexidade do algoritmo é $O(n^2)$.

TESTES E RESULTADOS

Vértices	Tempo de Execução (milisegundos)
5	47
10	54
20	68
30	79
40	87
50	102
100	139



- Gráfico Tempo de execução (ms) x Vértices;

DISCUSSÃO

O tempo de execução seguiu bem o que estava planejado. Houveram mais que uma amostragem e tirou-se uma média. O fato é que existiram amostragens que representam um

ponto fora da curva, porém, após diversas execuções notei que não fazia sentido utilizar aquela única amostragem.

Pelo fato de termos apenas 7 grafos, a demonstração de uma progressão quadrática é muito suave. Mas, ainda assim é notória no gráfico.

REFERÊNCIAS BIBLIOGRÁFICAS

Wikipedia. **Algoritmo de Prim.** Disponível em: <https://pt.wikipedia.org/wiki/Algoritmo_de_Prim>. Acesso em: 25 de junho de 2023.

GeeksforGeeks. **Prim's Minimum Spanning Tree (MST) - Greedy Algo-5.** Disponível em: <https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/>. Acesso em: 25 de Junho de 2023.

Sambol, Michael. **Playlist de vídeos no YouTube sobre "Algoritmo de Prim"**. Disponível em: <https://www.youtube.com/playlist?list=PL9xmBV_5YoZPKwb4XPB1sG7S6kNpN9JJo>. Acesso em: 28 de junho.

Almeida, Lauro. **Um estudo comparativo de algoritmos de árvore geradora mínima.** TCC (Trabalho de Conclusão de Curso) - Instituto Nacional de Matemática Pura e Aplicada (IMPA), 2018. Disponível em: <https://impa.br/wp-content/uploads/2018/03/TCC_2018_Lauro-e-Almeida.pdf>. Acesso em: 28 de Junho de 2023.