Relatório Atividade 1 Grafos - INE5413

Leonardo Luiz Gambalonga Alves de Oliveira (21201250)
Lucas Gusmão Valduga (21103505)
Ciências da Computação - UFSC

Para realizar esse trabalho escolhemos implementar os algoritmos em Python e utilizamos majoritariamente o material *Anotações para a Disciplina de Grafos (Versão de 27 de março de 2023, Rafael de Santiago)* como base para implementação dos algoritmos.

Representação:

Essa questão consiste em implementar uma estrutura que representa um grafo não dirigido e ponderado.

Realizamos a tarefa por meio de uma estrutura de classe e optamos por utilizar o método de lista de adjacências por ter uma implementação mais simples do que o método de matriz de adjacências. A lista de adjacências implementada consiste em uma estrutura de dicionário aninhado conforme a imagem abaixo.

```
{vertexLabel(str): {'neighborhood': {neighborLabel(str) : weight(float)} , 'index': i(int)}}
```

Os métodos solicitados foram implementados a partir de operações em cima da lista de adjacências além de um atributo adicional na classe, *numEdges*, para facilitar a implementação do método que busca a quantidade de arestas existentes no grafo.

Buscas:

Nessa questão foi implementado um algoritmo de busca em largura a partir de um vértice inicial (s) do grafo.

Foram três estruturas lista (Cv, Dv, Av), conforme o pseudocódigo da apostila, que representam respectivamente os vértices que já foram visitados na busca, à distância para cada vértice visitado (em número de arestas, desconsiderando peso) a partir de (s) e o antecessor de cada vértice no caminho definido com início em (s).

Além disso foi utilizada uma fila (Q) para auxiliar na busca, onde inicia enfileirando (s) e depois entra em um laço que busca todos os vizinhos do primeiro vértice da fila, atualiza suas informações nas três estruturas de lista mencionadas anteriormente e o enfileiram em (Q) caso eles já não tenham sido visitados antes, de acordo com (Cv).

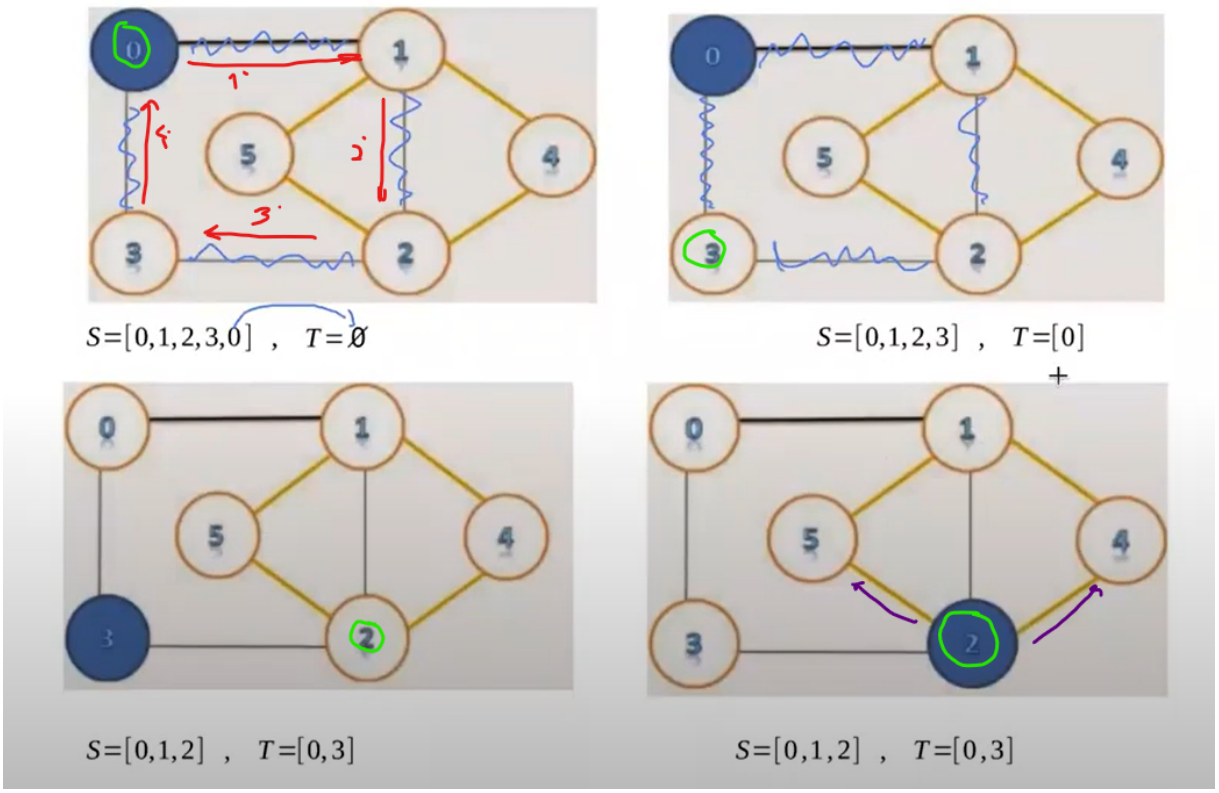
Ciclo Euleriano:

No algoritmo do ciclo euleriano, primeiramente é feita a verificação se o grafo a ser executado é mesmo um ciclo euleriano. Dessa forma, foi verificado se todos os vértices possuem grau par, caso tivesse algum vértice com grau ímpar, já poderia

considerar como não sendo um ciclo euleriano, nessa verificação aproveitou-se também e foi coletado um vértice como origem.

Após essa primeira etapa, é chamada função buscarCicloEuleriano, em que só aceita ciclos eulerianos e nela foi utilizada a estrutura de pilha, uma para a resposta em que conteria todos os vértices em ordem para formar o ciclo (pilha T) e outra para os vértices visitados a partir da origem sem estar na ordem correta (pilha S), além de uma lista para colocar todas as arestas representadas por uma tupla cujo contenha os dois vértices pertencentes a essa aresta. Dito isso, nessa segunda etapa foi feito um laço de repetição que só é finalizado caso essa lista esteja completa, ou seja, foi visto todas as arestas e a pilha S estiver vazia.

Portanto, é criado outro laço de repetição e dessa vez repetindo a quantidade igual ao número de vizinhos que o vértice de origem tem, nessa estrutura é visto cada vizinho e se a aresta com o vizinho ainda não foi visitada é posto dentro da lista anteriormente criada (o vizinho é escolhido de forma arbitrária, nesse código é pego na ordem de vizinhos como está no arquivo de texto), e esse vértice vizinho entra na pilha S e se torna o vértice atual (no lugar do vértice de origem) também. Assim repetindo até completar todas as arestas. Contudo, caso todas as arestas relacionadas ao vértice de origem atual já tenham sido visitadas, é retirado esse vértice da pilha S e colocado na pilha de resultado, como se tivesse voltando ao caminho. Dessa forma é verificado novamente, supondo que já tenha sido visitado novamente então volta mais um vértice, refazendo o caminho.



Para exemplificar, na imagem, é mostrado um grafo que já percorreu 4 vértices e está num vértice repetido (0), no primeiro quadro as setas em vermelho mostra a ordem de arestas que o caminho seguiu, o círculo em verde indica o vértice atual e as setas roxas indica as arestas não visitadas ainda. Desse jeito, é importante notar que retira-se da pilha os vértices até encontrar um que tenha caminhos ainda não visitados, no caso da imagem, o vértice 2.

Por fim, após acabar as arestas a serem visitadas e esvaziada a pilha S, é retornado o caminho representado pela pilha T e impresso na tela.

Algoritmo de Bellman-Ford ou Dijkstra:

Nessa questão optamos por implementar o algoritmo de Bellman-Ford pois ele consegue identificar e lidar com ciclos negativos no grafo, ao contrário do algoritmo de Dijkstra.

Neste algoritmo foram utilizadas duas estruturas de lista (D, A) que representam respectivamente a distância (em peso das arestas) do vértice inicial (s) para cada vértice do grafo e os vértices antecessores a cada vértice do grafo nos caminhos mínimos encontrados.

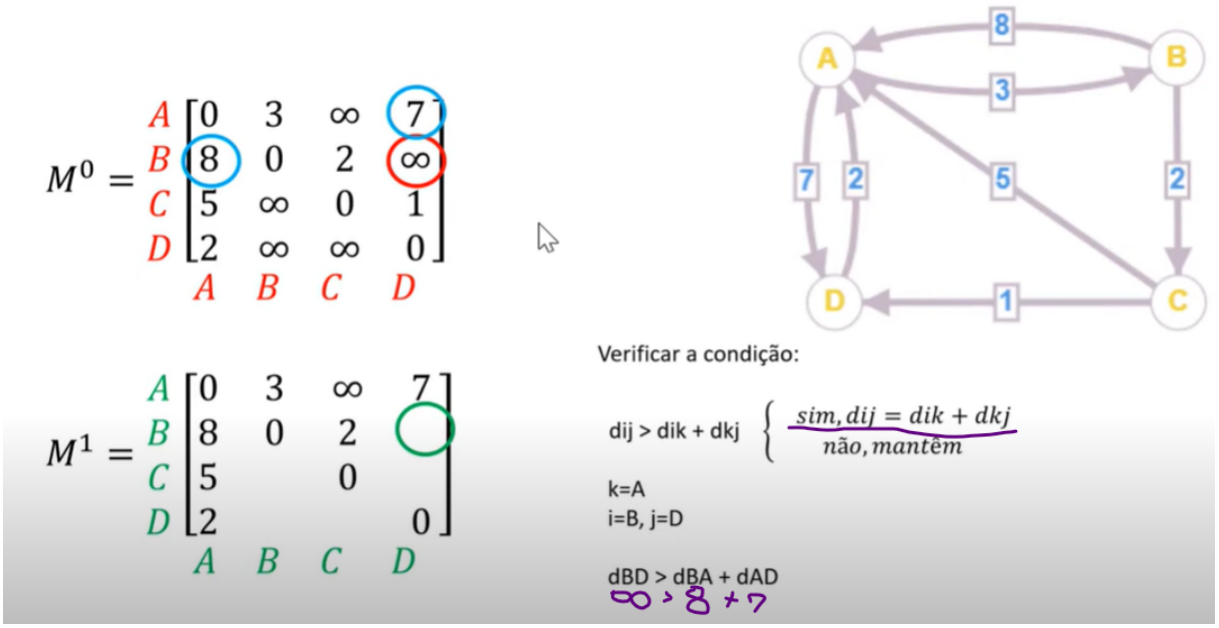
Utilizando a estrutura D que foi inicializado com infinito para cada vértice do grafo, o algoritmo faz o processo de relaxamento, onde, pelo nosso entendimento, será verificado, para cada vizinho de cada vértice, se a distância até o respectivo vizinho é maior que a distância até o respectivo vértice somado ao peso da aresta que leva até o respectivo vizinho. Esse processo é realizado (|V| -1) vezes, seguido da verificação dos ciclos negativos que consiste em fazer uma verificação adicional do processo de relaxamento, caso essa verificação seja verdadeira para algum vértice, significa que há um ciclo negativo.

Algoritmo de Floyd-Warshall:

Primeiramente, no algoritmo de Floyd-Warshall é feita uma matriz cujo o tamanho de linhas e colunas é igual ao número de vértices, e em cada posição que não tenha índices iguais, no caso a distância de um vértice para ele mesmo, que é 0, é colocada a distância entre o vértice indicado pelo índice da linha com o vértice indicado pelo índice da coluna, porém se não tiver ligação nesse primeiro momento é colocado o valor de infinito na posição.

Com essa matriz pronta é passada para uma outra função que vai ser chamada o número de vezes igual ao número de vértices. Nessa outra função, é feita outra matriz nos mesmo moldes da matriz feita anteriormente, contudo, mantendo os valores relacionados ao vértice indicado pelo índice de repetição dessa função, ou seja, mantém se os valores da linha e da coluna que tem índice igual ao da repetição da função. Contudo, para os outros valores da matriz, além da diagonal principal que é 0, é verificada a menor distância entre o valor direto do vértice representado pelo índice coluna com o vértice representado pelo índice da linha com

a soma da distância entre o vértice representado pelo índice da coluna com o vértice representado pelo índice da repetição da função com a distância desse mesmo vértice com o vértice representado pelo índice da linha. Caso não for, então é pego o valor da matriz anterior e repassado para essa nova matriz. Por fim é retornado a nova matriz que vai se tornar a matriz de referência para a próxima iteração. Na imagem abaixo está um exemplificação da lógica utilizada, M1 representa a nova matriz e M0 a matriz de referência e k o número de iterações.



Finalmente, ao final de todas as iterações dessa nova função é retornada a matriz resultante e a quantidade de vértices que vai ser utilizada para a impressão dos valores.