Relatório

Marcos Vinicius Mendes Faria

May 30, 2023

Abstract

Relatório da primeira etapa do trabalho prático de grafos para a disciplina de Técnicas de Programação Avançada.

1 Estrutura utilizada

Para representar o grafo de forma computacional, escolhi utilizar a lista de arestas. Em geral, essa abordagem foi a mais adequada, pois não há a necessidade de lidar com grafos de grande quantidade de vértices. A lista de arestas proporciona uma visualização mais clara do funcionamento do grafo, permitindo uma compreensão mais intuitiva de sua estrutura e conexões.

1.1 Leitura da matriz de adjacências

Uma observação relevante a ser mencionada é que, durante a leitura da matriz, surgiu uma dúvida sobre se deveria lê-la completamente, levando em conta que isso resultaria em duplicação de arestas. Essa duplicação ocorre devido aos pesos das arestas duplicadas que estão separados pela diagonal principal. No entanto, ao analisar os códigos fornecidos, percebi que não faria sentido ler parcialmente a matriz. Essa abordagem limitaria a capacidade de determinar todos os possíveis destinos a partir de um vértice, por exemplo. Adiante comentarei mais sobre isso.

$$A_{3\times3} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Para exemplificar, vamos utilizar os seguintes elementos da matriz: a_{12} e a_{21}. Ambos os elementos teriam o mesmo peso, caso essa matriz estivesse preenchida. No entanto, podemos associar a linha ao vértice de origem e a coluna ao vértice de destino. Assim, quando esses elementos forem salvos, a_{12} terá como origem o vértice 0 e destino o vértice 1, enquanto a_{21} terá como origem o vértice 1 e destino o vértice 0. Dessa forma, estamos representando para o grafo que essa aresta possui uma navegabilidade de um vértice para o outro. Com isso, percebi que não seria vantajoso ler a matriz parcialmente, a menos que estejamos lidando com um grafo direcionado.

1.1.1 Código utilizado para leitura da matriz.

```
for (int origem = 0; origem < numeroDeVertices; origem++) {
    for (int destino = 0; destino < numeroDeVertices; destino++) {
        //Verifica se a dist ncia entre as cidades    nula.
        if (matriz[origem][destino] != 0) {
            //Adiciona aresta.
            grafo.adicionaAresta(vertices.get(origem).getValor(),
                vertices.get(destino).getValor(), matriz[origem][destino])
                ;
        }
    }
}
```

Como mencionado anteriormente, uma vez estabelecida a correspondência entre linhas/colunas e origem/destino, a leitura da matriz se torna simples. No entanto, é importante verificar se o peso associado não é igual a zero, pois nesse caso a aresta é considerada inexistente.

2 Estratégias utilizadas

A seguir vamos entender quais estratégias foram utilizadas para atender aos requisitos estabelecidos nessa primeira parte do trabalho.

2.1 Vértices vizinhos

Aqui o objetivo era obter todos os vizinhos de uma cidade específica. Para alcançar esse objetivo, era necessário, primeiramente, usar o código fornecido para identificar a cidade desejada. Uma vez identificada, o método "obterDestinos()" era utilizado para encontrar as arestas que levam aos vértices adjacentes. Ao obter essa lista de arestas, era possível percorrê-la e exibir os valores desejados. Vale ressaltar que foi nessa etapa que ficou claro que a leitura parcial da matriz não seria viável.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 107.92 & 0 & 0 & 0 & 0 \\ 66.37 & 48.36 & 0 & 0 & 0 \\ 42.77 & 27.41 & 0 & 0 & 0 \\ 57.59 & 76.63 & 76.03 & 73.04 & 0 \end{bmatrix}$$

Para esclarecer, vamos usar o exemplo da matriz mencionada anteriormente. Podemos afirmar que a leitura dessa matriz é equivalente a ler a parte inferior da diagonal principal de uma matriz completa gerada pelo gerador de arquivo disponibilizado. No início, eu estava seguindo essa abordagem, mas quando chamava o método para obter os destinos da cidade 1, por exemplo, não encontrava nenhuma aresta adjacente. No entanto, o correto seria ter as arestas com os pesos: 107.92, 66.37, 57.59.

2.2 Caminhos possíveis

Nessa etapa, foi necessário realizar uma busca em largura no grafo para determinar os vértices que estavam no caminho. Essa abordagem nos permitiu identificar para quais vértices era possível chegar a partir de um vértice de entrada específico. O funcionamento detalhado do código pode ser encontrado nos arquivos fonte, onde estão devidamente comentados.