

Trabalho Prático de Grafos

Engenharia de Software

Bruno Pontes Duarte, Davi Fernandes Ferreira Silva
Diogo Martins de Assis, Eduardo Augusto Brito, Samuel Marques Sousa Leal

¹Instituto Ciências Exatas e Informática (ICEI)
Pontifícia Universidade Católica de Minas Gerais (PUC MG)
Belo Horizonte – MG – Brasil

Abstract. *This article describes the techniques and the technologies chosen by the group for the programming of the practical exercise in Graphs Theory discipline. On the back-end, Java language with the SpringBoot framework was used, while the front-end has been made using Next.js, along with the Vis.JS library. Finally, the tests were made using Java's library JUnit.*

Resumo. *Este artigo descreve as técnicas e escolhas de tecnologias utilizadas pelo grupo para a realização do trabalho prático da disciplina de Teoria dos Grafos. Para o back-end, foi escolhida a linguagem Java com o SpringBoot framework, e o front-end foi feito utilizando Next.js, juntamente com a biblioteca Vis.js; por fim os testes foram realizados com a biblioteca JUnit, de java.*

1. Introdução

Primeiramente, o projeto para o trabalho prático da disciplina de Teoria dos Grafos foi feito com Java no back-end, juntamente com o framework SpringBoot, visto que ele promove conceitos da programação orientada a objetos e é uma ferramenta que cria APIs para serem consumidas no ambiente do front-end. Ainda nesse viés, no front-end foi utilizado Next JS, um framework para desenvolvimento com React, assim como uma biblioteca chamada Vis JS; focada na exibição de grafos e gráficos de diversos tipos.

As classes em java utilizadas para a estruturação do projeto são "Aresta", "Vértice", "Cidade", "Grafo" e "Haversine". Além destas, há as classes que são responsáveis pelo acesso e estruturação de dados: "CidadeLoader", "GrafoController", "LeitorCsv", "ABB" e "Lista". Por fim, utilizamos as classes "ArestaWrapper", "VerticeWrapper" e "GrafoWrapper" para a estruturação de dados do retorno da API.

2. Desenvolvimento

O desenvolvimento foi realizado nas IDEs Visual Studio Code e IntelliJ, de maneira que cada integrante do grupo utilizou a aplicação que sentia mais familiarizado. O versionamento foi feito utilizando a ferramenta Git, com a separação de 4 branches principais, sendo elas: "master", "api-grafo", "logica-carregamento-cidades" e "leitor-csv"; controlando os merges para que não houvessem erros e substituições indevidas no código.

Após o tratamento e leitura dos 2903 registros do arquivo disponibilizado [Brazil Cities Database, 2023], foi formado um grafo com as cidades do Brasil. Para a demonstração de tal de uma maneira visual, foi utilizada a biblioteca Vis JS [Vis, 2023], em um ambiente feito com NextJS.



Figure 1. Visualização do Grafo lido do arquivo disponibilizado

Além disso, foi desenvolvida uma função que caso a cidade seja clicada, ela é destacada no mapa do grafo, assim como suas arestas. O objetivo é facilitar a visualização de seu grau e de seus respectivos vizinhos.

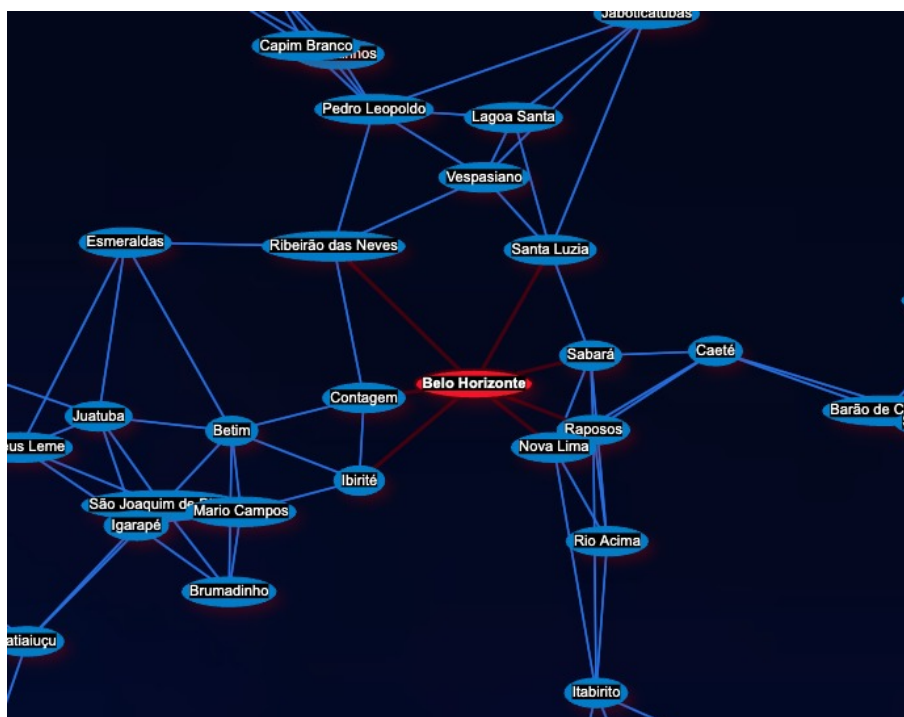


Figure 2. Visualização da cidade de Belo Horizonte

Uma outra função importante no projeto, é a da geração de Árvores Geradoras Mínimas (AGM). Elas são um subconjunto de uma árvore conectada em um grafo, que contém todos os vértices e um conjunto mínimo de arestas necessárias para conectar esses vértices sem formar ciclos. Em outras palavras, é uma árvore que abrange todos os nós do

grafo com o menor peso total possível. A AGM é frequentemente utilizada em algoritmos de otimização para encontrar o caminho mais curto ou a rede de custo mínimo em um grafo ponderado.

Utilizando o algoritmo de Prim para a geração da AGM, e testando a partir de duas cidades com componentes de tamanhos extremamente diferentes (Belo Horizonte e Rio de Janeiro), foi possível chegar nos seguintes resultados após a integração com o ambiente do front-end:

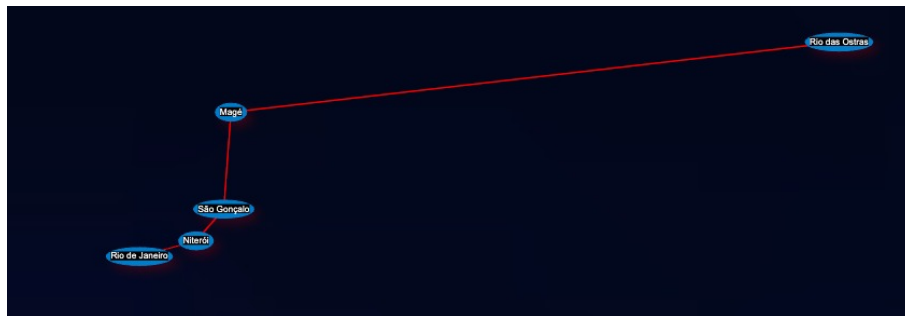
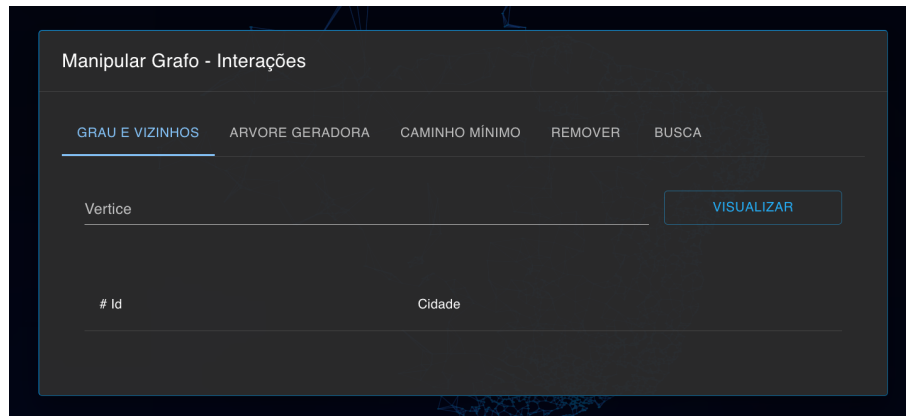


Figure 3. Visualização da AGM da cidade do Rio de Janeiro



Figure 4. Visualização da AGM da cidade de Belo Horizonte

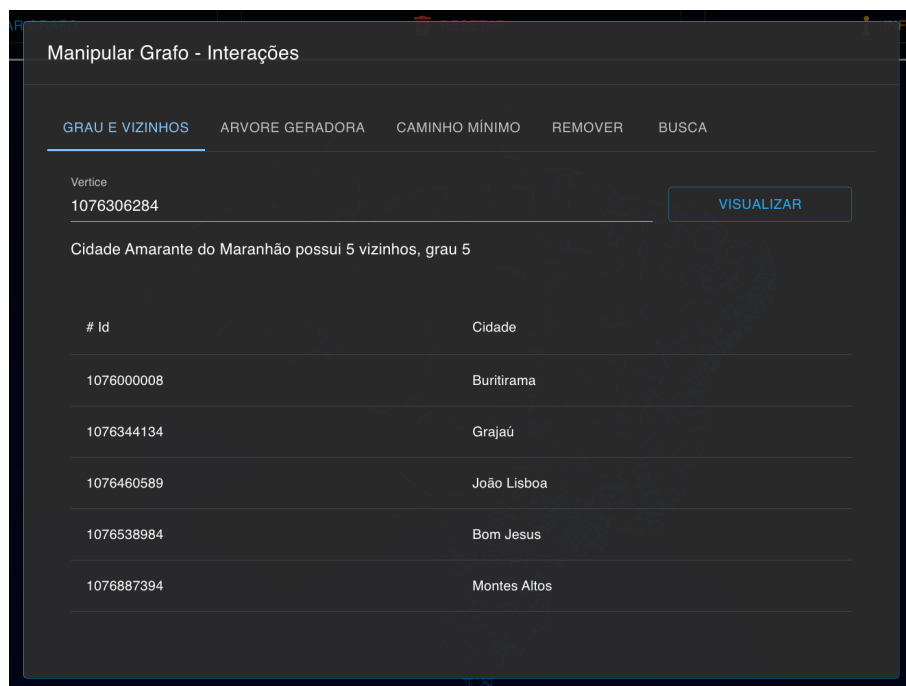
Ao clicar em "Manipular Grafo" na Figura 1, um modal é aberto com diversas opções de edição, para que o usuário consiga especificar qual função quer executar no grafo base.



The screenshot shows a modal titled "Manipular Grafo - Interações". It has five tabs: "GRAU E VIZINHOS" (selected), "ARVORE GERADORA", "CAMINHO MÍNIMO", "REMOVER", and "BUSCA". Below the tabs, there is a "Vertice" input field and a "VISUALIZAR" button. At the bottom, there is a table with two columns: "# Id" and "Cidade".

Figure 5. Modal de visualização de funções

Dessa maneira, após preenchimento do ID do vértice e o clique no botão "Visualizar", o site retorna ao usuário uma lista dos Vértices correspondentes, assim como a quantidade de vizinhos diretos e o grau do vértice, representado na imagem seguinte:



The screenshot shows the same modal as Figure 5, but with the "Vertice" field filled with "1076306284" and the "VISUALIZAR" button clicked. Below the input field, a message states: "Cidade Amarante do Maranhão possui 5 vizinhos, grau 5". Below this message is a table with two columns: "# Id" and "Cidade".

# Id	Cidade
1076000008	Buritirama
1076344134	Grajaú
1076460589	João Lisboa
1076538984	Bom Jesus
1076887394	Montes Altos

Figure 6. Retorno das informações pelo modal

3. Testes e resultados

Os testes foram realizados na classe "GrafoTest.java" e fizeram uso da biblioteca JUnit. Dessa forma, por meio de declarações "assertEquals", "assertDoesNotThrow", "assert-

False”, ”assertNull” e ”assertNotNull” foram testados diversos aspectos do sistema. Os testes unitários consistem em:

- testaGrafoCompleto;
- testaAdicionarVertice;
- testaAdicionarAresta;
- testaArestaNaoDeveSerDirecionada;
- testaArestaNaoDeveExistir;
- testaQuandoNaoExisteVertice;
- testaNaoDeveCriarDoisVerticesComIDsIguais;
- testaRetornarOrdemDoGrafo;
- testaRetornarTamanhoDoGrafo;
- testaCarregarDeArquivo;
- testaCriarSubgrafo;
- testaCriarSubGrafoComArestas;
- testaPesquisaEmProfundidade;
- testaPesquisaEmLargura;
- testaRetornaGrauDeUmVertice;
- testaRetornaQuantidadeDeVizinhos;

Como resultado, obtivemos sucesso em todos os casos de testes possíveis em que teorizamos possíveis falhas.

4. Conclusão

Esse trabalho foi feito com o intuito de transformar um mapa convencional em um grafo, ligando a cada cidade com outras quatro mais próximas, tornando as cidades em vértices e as estradas/conexões em arestas. Utilizou-se esse grafo para criar algoritmos com diferentes objetivos, como por exemplo árvore geradora mínima e o caminho mínimo. Recorreu-se a fórmula de Haversine para definir as distâncias entre as cidades e determinar as quatro cidades mais próximas.

Foi necessário realizar uma otimização para definir as cidades mais próximas, já que inicialmente o projeto fazia a comparação com todas as quase 3000 cidades. Para isso, a partir de cada cidade escolhida, foi determinado um raio de pesquisa para definir as outras quatro cidades. Por fim, vale ressaltar que, por abrodar diversas áreas do conhecimento, o trabalho universitário foi de suma importância para o desenvolvimento acadêmico e pessoal de cada um dos integrantes; especialmente no que tange a matéria de Teoria dos Grafos.

5. Referências

Vis Js, 2023. Disponível em: <https://visjs.org/>. Acesso em: 05/06/2023.

Java Oracle Documentation, 2023. Disponível em: <https://docs.oracle.com/en/java/javase/20/>. Acesso em: 03/06/2023.

Brazil Cities Database, 2023. Disponível em: <https://simplemaps.com/data/br-cities>. Acesso em 06/06/2023.