Algoritmos e Estruturas de Dados II

Trabalho 2 (Mapa dos Fenícios)

Lara Alves Kunrath Padilha, Leonardo Garcia Forquim Bertinatto

PUCRS

# 1. Introdução

# Os Fenícios estão voltando a realizar viagens por todos os mares, e querem utilizar das últimas tecnologias disponíveis, tendo como objetivo diminuir o tempo das viagens e reduzir a pegada de carbono, utilizando o mínimo de combustível possível para realizar todos os percursos previstos nos mapas. Eles declararam que, as vezes, é possível que algum mapa esteja errado, colocado um porto em um local que não é possível chegar de barco, então eles gostariam que esse porto fosse removido da rota de viagem, evitando assim dias de viagem e gasto de combustível desnecessários.

# 2. O problema

A partir dos pedidos dos Fenícios nossos problemas eram os seguintes:

- Ler diferentes mapas e processá-los da mesma maneira;

- Passar os mapas para Grafos;

- Percorrer os portos pelo menor caminho possível;

- Calcular o mínimo de combustível para as viagens.

# 3. Processo de solução

Nosso processo de solução se iniciou com a escolha do tipo de grafo a ser utilizado e como faríamos para percorrer o mapa, foram escolhidos o grafo com lista adjacente e o caminhamento com algoritmo de Dijkstra.

A leitura do arquivo é feita no método *lerArquivo(String caminhoArquivo)*, o qual recebe o caminho do arquivo como parâmetro na classe principal do projeto, a App, e utiliza um objeto Scanner para ler as informações do arquivo. Após a leitura do arquivo, a estrutura de grafo é carregada no método *carregarGrafo()*. Nesse método, é criado um objeto Grafo com o número correto de vértices e um *array* do tipo *Vertice* para armazenar as informações de cada vértice. Em seguida, o método *conectarArestas()* é chamado para realizar a conexão das arestas do grafo. Ele percorre a matriz do mapa e conecta os vértices vizinhos que não sejam marcados com o caractere '\*' usando o método *adicionarAresta()* do objeto Grafo.

O método *encontrarPortoInicial()* foi criado com o intuito de encontrar automaticamente o porto de número 1 do mapa, o qual é o ponto de partida para o caminhamento em qualquer mapa.

Já a classe Dijkstra implementa o algoritmo de Dijkstra, responsável por encontrar o caminho mínimo em um grafo. No construtor da classe, o grafo e o vértice de origem são passados como parâmetros, e então são inicializados os arrays *antecessor*, *distancia* e *percorrido*, que armazenam informações sobre o caminho mínimo encontrado. Um objeto do tipo *FilaPrioridadeMinima* é criado para auxiliar na seleção do próximo vértice a ser processado. A função *imprimirResultado()* exibe informações sobre cada vértice: índice, distância, antecessor e se o mesmo foi percorrido.

# 4. Evidências de que o problema foi resolvido

Os resultados esperados eram os seguintes:

5 x 10 | o custo total da viagem é de 304 unidades de combustível

15 x 80 | o custo total da viagem é de 304 unidades de combustível

30 x 80 | o custo total da viagem é de 432 unidades de combustível

50 x 100 | o custo total da viagem é de 484 unidades de combustível

60 x 500 | o custo total da viagem é de 986 unidades de combustível

500 x 1000 | o custo total da viagem é de 3426 unidades de combustível

Os resultados obtidos foram os seguintes:

**Caso de teste 1**

Manualmente indicamos que o porto de partida estava presente no vértice de número 2, então o método imprimirResultado() da classe Dijkstra retornou todos os vértices e as respectivas distâncias em relação ao vértice indicado como origem. Então foi possível descobrir que para os fenícios realizarem a navegação até o porto 2 e voltar seriam necessárias no mínimo 20 unidades de combustível.

Infelizmente a dupla não conseguiu cumprir os requisitos de entrega, conseguindo apenas chegar a etapa de caminhamento do grafo, isso apenas para o arquivo “mapa0”, pois ao testar nos outros arquivos a aplicação começou a quebrar.