Universidade Federal de Minas Gerais

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Trabalho Prático

Programação e Desenvolvimento de Software II (DCC204)

Estevão de Almeida Vilela

Wagner Abreu

Belo Horizonte, Novembro de 2019

1. Introdução

Este Trabalho Prático consiste no desenvolvimento de um sistema em linguagem C++ baseado no paradigma de Orientação à Objetos. Para modelar um sistema de alocação de demanda de transporte utilizamos de classes abstratas e conceitos de OO como modularização, polimorfismo, testes de unidade, encapsulamento, contrato e boas práticas de programação em geral como refatoração, código comentado e versionamento de código.

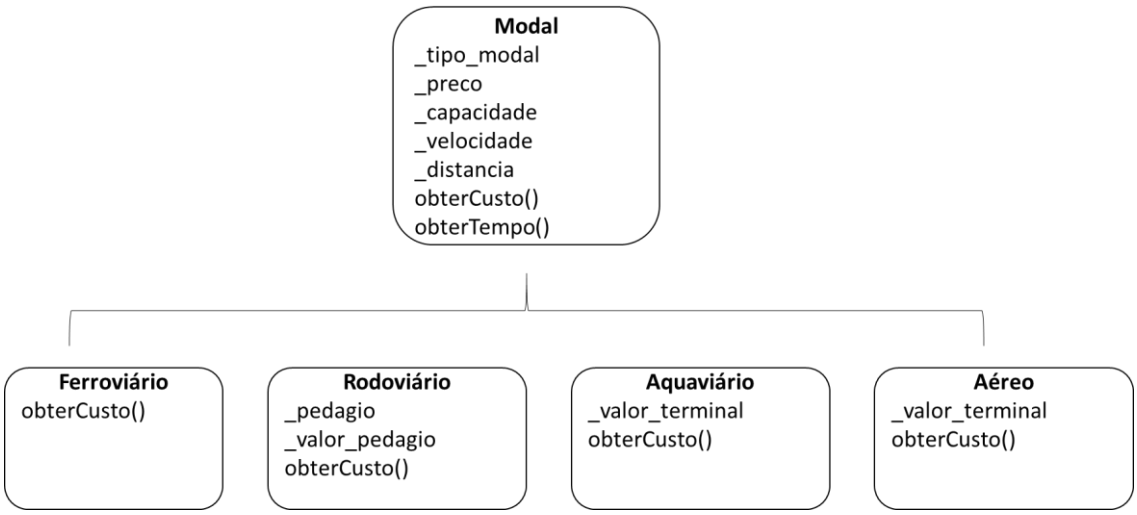
Desenvolvemos, portanto, um sistema que, dada uma demanda de transporte de uma certa quantidade de determinado produto, para alguma localidade do Brasil ou Exterior, encontra o caminho de menor custo. O custo a ser considerado pode ser monetário ou menor distância ou tempo de percurso, considerando os preços dos serviços, tempo de transporte e capacidade dos diversos modais disponíveis (rodoviário, ferroviário, aéreo e aquaviário).

2. Decisões de implementação, Classes e conceitos de OO

O sistema de alocação de demanda recebe do usuário três *inputs*: a quantidade de carga a ser transportada em toneladas, a localidade de origem e a localidade de destino. Dada essas informações um operador de transporte de cargas realiza o cálculo para verificar qual a melhor rota entre as duas localidades.
A conexão entre todas as localidades se dá por meio de um modal de transporte, ou seja, a carga pode ser transportada por meio de um dos seguintes tipos de transporte: aéreo, aquaviário, ferroviário ou rodoviário.
Portanto, a implementação do sistema de alocação de demanda de transporte constitui principalmente do desenvolvimento de três classes principais:
1) Localidade:
Utilizamos dados reais de capitais brasileiras e capitais de alguns países estrangeiros. A classe Localidade possui os seguintes atributos: código da localidade, município, latitude, longitude, estado e país.
Para essa classe utilizamos uma base de dados em formato .txt.

2) Modal
Cada conexão entre duas localidades constitui um tipo de transporte que possibilita o tráfego entre as cidades: aéreo, aquaviário, ferroviário ou rodoviário. Para implementar essa ideia abstraímos o conceito em uma classe Modal que é a superclasse em que cada

tipo de transporte constitui uma classe derivada. A estrutura a seguir é elucidativa:



A classe Modal possui atributos com nível de proteção *protected*:

- O atributo tipo modal para definir os tipos de transporte;
- O atributo preço indica o preço do serviço de transporte em R$/Km;
- O atributo capacidade indica a capacidade do veículo de carga;
- O atributo velocidade indica a velocidade do veículo de transporte em Km/h;
- O atributo distância indica a distância entre duas arestas.

E métodos públicos – para além dos *setters* e *getters*:

- O método obter custo calcula o custo associado a cada conexão entre duas localidades dado o tipo de transporte que as conecta. Usamos aqui o conceito de **polimorfismo**, pois cada modal terá o custo calculado de forma diferente: o tipo de transporte rodoviário acrescente pedágio ao custo, enquanto os tipos de transporte aquaviário e aéreo acrescentam o valor do terminal ao custo.
- O método obter tempo calcula o tempo necessário para uma viagem de ida entre duas localidades, dividindo a distância pela velocidade do tipo de transporte que conecta as localidades.

Para essa classe utilizamos uma base de dados em formato .txt.

3) Operador

A classe Operador é responsável por realizar o cálculo que otimiza o custo ou o tempo gasto para transportar uma determinada quantidade de carga entre duas localidades. Essa classe foi implementada utilizando do conceito de **composição**: um dos atributos é o tipo

de abstrato de dados grafo implementado como matriz de adjacências em que cada aresta é um objeto da classe Modal; além disso, outro atributo é um *vector* que armazena as solicitações dos usuários.
Para calcular a rota que minimiza o custo ou o tempo entre duas localidades utilizamos o algoritmo *Dijkstra's Shortests Path* e para detectar se há uma conexão (caminho) entre duas localidades utilizamos o algoritmo *Breadth First Search*. A implementação dos algoritmos utilizada foi a disponível no website *GeeksForGeeks*.

3. Conclusão

Este trabalho procurou exercitar os conceitos de Programação Orientada à Objetos aprendidos durante a disciplina de Programação e Desenvolvimento de Software II na implementação de um sistema de pequeno porte.

Utilizamos a linguagem C++ para implementar esse sistema e utilizamos como contexto um sistema de alocação de demanda de transporte que retorna ao usuário a rota que minimiza o custo de transporte entre duas localidades.

Diversos conceitos de OO foram utilizados durante este trabalho prático, como: modularização, polimorfismo, testes de unidade, encapsulamento e contrato. Utilizamos o Git como ferramenta de versionamento de controle, seguimos boas práticas de programação e todo o código reproduzido pode ser acessado na plataforma GitHub:

https://github.com/pds2/20192-team-12

Referências:

Dijkstra's Shortests Path

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

https://www.geeksforgeeks.org/printing-paths-dijkstras-shortest-path-algorithm/

Breadth First Search

https://www.geeksforgeeks.org/implementation-of-bfs-using-adjacency-matrix/

https://www.geeksforgeeks.org/find-if-there-is-a-path-between-two-vertices-in-a-given-graph/