Documentação Trabalho Prático 2

Algoritmos 1

Misael Saraiva de Rezende

Dezembro 2021

1 Introdução

Uma grande empresa varejista precisa solucionar um problema no processo de entregas a domicílio das suas vendas online. A empresa entende que a resolução do problema é imprescindível para que haja um aumento da participação da empresa no mercado, assim como de fidelização dos clientes.

A empresa possui várias lojas na cidade, e espera que seja criado um processo que selecione o melhor caminho entre as suas lojas, de forma que o custo com o transporte de mercadorias entre essas lojas seja o menor possível. É importante notar, que as lojas possuem distâncias diferentes entre si, e que também a empresa espera que todas lojas sejam atendidas por um meio de transporte.

A empresa tem três tipos de transporte disponíveis para realizar os deslocamentos: drone, motocicleta e caminhão. Os drones têm baixíssimo custo de transporte entre lojas, mas alto custo de aquisição e nem sempre há drones à disposição para atender todas as lojas. Já as motocicletas e caminhões possuem custos variados, sendo que as motocicletas, em especial, têm limite de quilometragem reduzido, e nem sempre podem atender a todos os trajetos. Contudo, as motocicletas e caminhões não tem limite de quantidade disponível para uso

Para pôr a melhoria em prática, é fundamental que o processo de deslocamento de produtos entre as várias lojas da organização seja otimizado. Para isso, foi projetado um algoritmo que encontra os melhores percursos entre as lojas, indica o meio de transporte a ser utilizado em cada rota, e minimiza o custo total desses deslocamentos. Outro fator importante é que o algoritmo garante que todas as lojas da empresa na cidade serão atendidas por um meio de transporte.

Assim sendo, a empresa espera obter a melhoria nos processos, reduzir os custos e alcançar os objetivos almejados de crescimento da rede e maior fidelização de clientes.

2 Implementação

O programa foi desenvolvido na linguagem de programação *C++* e compilado usando o compilador *g++*, na versão *C++11*.

2.1 Solução proposta

O problema apresentado pela empresa pode ser modelado com o auxílio de grafos. Assim, cada loja é um vértice e o caminho entre as lojas é uma aresta. Como pode haver deslocamento de qualquer loja para qualquer outra loja, o problema pode ser modelado como um grafo completo - que é um tipo de grafo conectado - na qual todas as lojas têm arestas para as outras lojas.

O problema de encontrar as melhoras rotas entre as lojas, assinalar o meio de transporte a ser empregado em cada trajeto e minimizar o custo total dessas viagens pode ser modelado em duas partes.

1. Resolver o problema de encontrar o caminho de menor custo que percorre todas lojas.

2. Selecionar quais meios de transporte serão utilizados em cada percurso.

O problema de encontrar o caminho de menor custo que passe por todas as lojas, sem repetir um rota já usada, pode ser modelado com o uso de uma árvore geradora mínima (AGM). Uma AGM é uma árvore geradora de um grafo não direcionado, que tem arestas com pesos definidos, que percorre todos os vértices e que a soma das arestas escolhidas seja a menor possível. O algoritmo de Prim é um dos algoritmos mais utilizados para resolver esse problema. Esse algoritmo encontra uma AGM em um grafo não direcionado com pesos.

Depois de encontrar o caminho de menor custo entre as lojas, é necessário selecionar o tipo de transporte para cada trajeto. Para isso, uma solução proposta pode ser definida da seguinte maneira:

- Quando disponível, aloque os drones para os caminhos mais distantes.

- Selecione a motocicleta ou caminhão para as rotas restantes, observando as seguintes condições: limite de quilometragem da moto e custo por quilômetro de viagem de moto contra custo por quilômetro de viagem de caminhão

2.2 Estrutura de dados

O programa foi implementado usando a estrutura de dados struct do tipo `Vertice` para armazenar os dados de cada vértice: *id*, *x*, *y*;

A classe `Grafo()` contém a definição para gerar e armazenar o grafo com os vértices e arestas. O atributo *_grafo_com_pesos*, é um ponteiro de ponteiros para float, que armazena o grafo completo gerado, com as distâncias entre cada vértice.

A classe `MinhaBiblioteca()` contém algumas estruturas como o grafo completo *_grafo*, os vértices raiz da AGM *_vertices_raiz_agm* e a AGM ordenada *_agm_ordenada*.

2.3 Modularização

O programa foi separado em três módulos, `main`, `MinhaBiblioteca` e `Grafo`.

O módulo `main` contém a implementação da leitura dos dados de entrada: número de lojas, limite de quilometragem (km) das motos, número de drones, custo por km das motos, custo por km dos caminhões e as localizações de cada loja. Nesse módulo ocorre a instanciação do grafo e da biblioteca auxiliar para resolver o problema proposto.

O módulo `Grafo` contém a classe `Grafo()` e implementa o grafo e o método *CalcularDistancias()*. Esse método calcula a distância euclidiana de cada loja para todas as outras lojas e armazena o resultado no grafo completo *_grafo_com_pesos*.

O módulo `MinhaBiblioteca`, composta pelos arquivos `MinhaBiblioteca.hpp` e `MinhaBiblioteca.cpp` contém a classe `MinhaBiblioteca()` e a implementação de métodos auxiliares. Os três principais métodos nesse módulo são `AlgoritmoPrim()`, `ObterMaioresCustos()` e `MinimizarCustoTrajeto()`.

O método `AlgoritmoPrim()` implementa o algoritmo de Prim. O algoritmo pega um grafo de entrada, e encontra a AGM. A AGM resultante é armazenada no atributo *_vertices_raiz_agm*. A implementação segue o pseudocódigo dado nas aulas, com a diferença de que não é usado a estrutura de dados fila.

O método `ObterMaioresCustos()` ordena a AGM criada em ordem decrescente, ou seja, de acordo com o valor das distâncias dos caminhos da árvore geradora mínima (AGM). O resultado final é armazenado no atributo *_agm_ordenada*, que é um *vector* de *pair*. Nele, é armazenado as lojas escolhidas, a distância entre cada uma delas e se já tem ou não um veículo alocado para transporte em cada uma.

O método `MinimizarCustoTrajeto()` executa a última etapa necessária para resolver o problema de otimização. Esse método usa a AGM obtida pelo método *ObterMaioresCustos()*. O pseudocódigo com a solução implementada nesse método é apresentado na figura 1.

2.4 Entrada e saída de dados

Para executar o programa em uma máquina com sistema operacional Linux Ubuntu ou outras variantes do Linux, o usuário deve digitar os comandos sugeridos pela especificação do trabalho. O exemplo 4.1 demonstra isso.

A saída de dados consiste em resultados da execução do algoritmo de acordo com os dados lidos do arquivo de teste.

3 Análise de complexidade de tempo assintótica

As principais funções implementadas, com maior custo de complexidade de tempo, são as seguintes:

- *CalcularDistancias()* = $O(n^2)$
- *AlgoritmoPrim()* = $O(n^2)$

```
Seja N o número de lojas
Seja K o limite de km das motos
Seja D o número de drones disponíveis
Seja M o custo por km das motos
Seja C o custo por km dos caminhões
Seja L = N, o número de lojas desconectadas
Seja Trajeto a distância entre duas lojas

if D == N-1:
    Imprima é 0.000 0.000
else:
    if D >= 2:
        While D > 0:
            Aloque os drones nos caminhos mais longos
            Decremente D
            Decremente L
        Se L > 0:
            While L > 0:
                if M <= C and Trajeto <= K:
                    Aloque motos aos caminhos
                    Decremente L
                else:
                    Aloque caminhões aos caminhos
                    Decremente L
    Senão:
        While L > 0:
            if M <= C and Trajeto <= K:
                Aloque motos aos caminhos
                Decremente L
            else:
                Aloque caminhões aos caminhos
                Decremente L


imprima o custo das motos e custo dos caminhões
```

Figura 1: Pseudocódigo do algoritmo proposto

- $MinimizarCustoTrajeto() = O(n)$

4 Instruções de compilação e execução

Para compilar o algoritmo, é disponibilizado um arquivo de configuração *MakeFile*, que facilita a compilação do programa principal e suas dependências. O usuário pode usar o comando *make* para compilar o algoritmo. Para limpar os arquivos de compilação gerados, pode-se usar o comando *make clean*.

O arquivo de entrada é passado ao programa pela entrada padrão. Para a execução do programa no ambiente Linux Ubuntu, o modelo a executar é:

```
./tp02 < nome_do_arquivo
```

sendo `nome_do_arquivo` o nome do arquivo de teste.

4.1 Exemplo

Digitando o comando

```
./tp02 < test1.txt
```

vai executar o algoritmo lendo os dados de entrada do arquivo *test1.txt* e gerar, na saída padrão, o resultado apresentado abaixo:

3.000 14.000

5 Conclusão

Foi criado um algoritmo para ajudar a rede de lojas a otimizar o processo de entregas a domicílio das vendas online. Com isso, espera-se que a solução apresentada ajude a empresa a encontrar os percursos mais viáveis entre as várias lojas, os melhores meios de transporte e a minimização do custo total da operação. O resultado dessa solução proposta ajudará a empresa a alcançar um crescimento da participação dela no mercado, assim como de fidelização dos clientes.

6 Referências

COMPLETE GRAPH. In: WIKIPEDIA, the free encyclopedia. Flórida: Wikimedia Foundation, 2021. Disponível em:
https://en.wikipedia.org/wiki/Complete_graph .*Acessoem* : 05*dez*.2021.
PRIM'S ALGORITHM. In: WIKIPEDIA, the free encyclopedia. Flórida: Wikimedia Foundation, 2021. Disponível em:
https://en.wikipedia.org/wiki/Prim's_algorithm .*Acessoem* : 12*dez*.2021.