Henrique Soares Assumpção e Silva
Matrícula: 2020006620
henriquesoares@dcc.ufmg.br

Trabalho Prático II
Algoritmos I
Universidade Federal de Minas Gerais

2021-12-18

1 Introdução

Durante o mês de Novembro ocorre uma das datas mais relevantes para as empresas de vendas nacionais, a Black Friday. O projeto implementado no trabalho prévio foi um sucesso, e por esse motivo a empresa contratante decidiu realizar um segundo projeto para trazer melhorias nos processos de delivery de vendas online. Nesse contexto, deve-se desenvolver um algoritmo que seja capaz de, dado um conjunto de lojas, as distâncias entre elas e conjunto de meios de transporte disponíveis, encontrar a melhor combinação de caminhos de modo a minimizar o custo com transporte.
Formalmente, seja um conjunto de lojas $\mathcal{V}$, $|\mathcal{V}| = N$, onde cada $v \in \mathcal{V}$ representa uma loja, que por sua vez está associada à uma geolocalização (x_v, y_v), $-10000 \leq x_v, y_v \leq 10000$. Podemos definir um grafo não-direcionado ponderado $G = (\mathcal{V}, E, \delta)$, onde $\mathcal{V}$ é o supracitado conjunto de lojas, E é um conjunto de arestas que representa os caminhos entre as lojas, e $\delta : E \to \mathcal{F}$ é uma função que mapeia as arestas do grafo à valores reais positivos $\mathcal{F}$, indicando o peso desta aresta, i.e. distâncias dos caminhos. Dado $e = (v, u) \in E$, o valor de $d_e = \delta(e)$ pode ser calculado pela distância euclidiana entre os valores atribuidos às geolocalizações dos nós v e u. Como, de acordo com o enunciado, existem caminhos entre quaisquer pares de lojas, G também é um grafo completo, no caso K_N, onde $|E| = \frac{N(N-1)}{2}$.
No contexto do trabalho, existem três tipos possíveis de transporte para o delivery: Drones, Motocicletas e Caminhões. O número D de drones é limitado por $1 \leq D \leq N$, e o custo associado a esse meio de transporte é nulo, no entanto, cada drone só pode transportar mercadoria entre lojas que também possuam tal tecnologia. O transporte por motocicletas tem uma limitação de autonomia, e portanto define-se K,$1 \leq K \leq 10000$ como sendo o tamanho máximo de um trajeto para que esse possa ser percorrido por uma motocicleta. Qualquer caminho de tamanho maior que K, isto é qualquer $e \in E : \delta(e) > K$, deve ser percorrido por um caminhão. O custo do kilômetro percorrido por caminhões tem valor C e o custo para motocicletas tem valor M, onde $1 \leq M \leq C \leq 10$.
O enunciado descreve que deve-se "encontrar uma combinação de trjetos de forma que, ao percorrer cada trajeto uma única vez ao dia e em uma única direção, o custo geral de deslocamento seja minimizado". Portanto, podemos interpretar o problema como o de encontrar uma árvore geradora mínima para o grafo G definido, e então calcular o custo total com transporte.

2 Modelagem

O programa foi desenvolvido na linguagem C++, compilada pelo compilador G++ da GNU Compiler Collection, no sistema operacional Windows 10. O programa foi inicialmente implementado e compilado em uma máquina com 16GB de RAM em um processador Intel Core I5-9400F.

2.1 Classes e Estruturas

Para a implementação do algoritmo requisitado, foram criadas classes e estruturas para auxiliar no desenvolvimento da lógica do programa. Seguem as descrições de cada uma dessas construções:

- `Struct Point`: Estrutura simples que representa um ponto em duas dimensões. Foi utilizada para armazenar os dados x e y das lojas. Possui apenas métodos construtores e destrutor.
- `Class Store`: Classe que representa uma loja. Armazena um valor inteiro identificador v e os valores da 2-tupla (x_v, y_v), que define a geolocalização da loja. A classe possui apenas métodos `get` e `set` para esses atributos.

2.2 Funções

Além das estruturas descritas na seção anterior, também foi-se desenvolvido funções específicas para o funcionamento do algoritmo. Seguem suas descrições:

- `int min_weight_node(double weights, bool is_visited, int num_stores)`:
  Função que recebe um vetor de pesos `weights`, um vetor de booleanos indicando os nós visitados `is_visited` e o número de lojas `num_stores`, i.e. $|\mathcal{V}|$. Itera sobre os nós(lojas) do grafo e encontra o nó

que ainda não visitado e que possui menor peso, e então retorna seu índice. A utilização dessa função é especificada de forma mais aprofundada na Seção 3.

- `double euclidean_distance(Point p1, Point p2)`: Função que recebe dois objetos do tipo Ponto e retorna a distância euclidiana entre eles.

Uma função `read_data` também foi implementada para auxiliar na leitura dos dados de entrada, mas como sua assinatura é extensa e seu comportamento é trivial, optou-se por omitir sua descrição detalhada nesta documentação.

2.3 Programa principal

Esta seção ia descrever o funcionamento da função principal `main` do programa desenvolvido. Os passos do programa principal são descritos a seguir:

1. Lê-se os dados do arquivo de entrada e então se obtem os valores necessários para o cálculo dos custos totais c_m e c_c associados à motocicletas e caminhões, respectivamente. Então se calcula a distância euclidiana entre todos os pares de geolocalizações de lojas presentes, i.e. calcula-se o peso d_e de cada aresta e do grafo G;

2. Após isso, entra-se na sequência que compoe o algoritmo principal do programa. O objetivo é encontrar uma árvore geradora mínima de G, e para isso, além da supracitada matriz de adjacências, inicializa-se um vetor de pesos para todos os nós do grafo, um vetor booleano indicando quais nós já foram visitados pelo algoritmo e um vetor de inteiros V_m, onde o valor da i-ésima posição desse vetor indica o nó que esta conectado com o nó de índice i;

3. Escolhe-se um nó qualquer, de forma aleatória, e então inicializamos os pesos de todos os outros nós como infinito, e o deste nó como 0. Isso é feito para que esse nó seja escolhido na primeira iteração do algoritmo;

4. Inicia-se o loop principal do algoritmo para encontrar a árvore geradora mínima, que no caso é o algoritmo de Prim [2]. Esse algoritmo garante que os nós armazenados em V_m irão compor uma árvore geradora mínima. Seu funcionamento detalhado em pseudo-código é fornecido na Seção 3;

5. Após a obtenção da árvore geradora mínima em V_m, obtem-se os pesos de todas as arestas da árvore e então os ordena em um vetor. Após isso, remove-se as $D-1$ maiores pesos desse vetor, que representam caminhos que poderão ser percorridos com custo zero por drones, e então calcula-se o custo do restante dos caminhos, finalmente obtendo os valores c_m e c_c desejados, que são impressos na tela(stdout).

3 Solução

O algoritmo 1 descreve o pseudo-código do algoritmo de Prim adaptado para o problema em questão. Esse algoritmo garante que os nós de S e as arestas de $\mathcal{E}$ formem uma árvore geradora mínima de G, que então pode ser utilizada para calcular os valores c_m e c_c descritos na Seção 2.3. O enunciado do problema informa que cada drone só pode percorrer caminhos entre pares de lojas que utilizem essa tecnologia, além de que cada drone só consegue atender um trajeto por dia. Assim sendo, é fácil perceber que, para obter um custo mínimo, basta escolhermos as lojas que pertencem aos $D-1$ maiores caminhos no grafo, de modo a garantir que todos os drones sejam utilizados e que o caminho final possui custo de transporte mínimo.
Para realizar o cálculo de tais valores, considere o conjunto $\mathcal{E} \subseteq E$ de arestas da árvore geradora mínima encontrada para o grafo G. Considere o conjunto $\mathcal{D} \subseteq \mathcal{E}, |\mathcal{D}| = D-1$ das $D-1$ arestas de maior peso de $\mathcal{E}$, i.e. $\mathcal{D} = \{e_1, e_2, ..., e_{D-1}\}$, onde $\forall e \in \mathcal{D}, \forall h \in \mathcal{E} - \mathcal{D} : \delta(e) \geq \delta(h)$, que representam o conjunto de caminhos no grafo que serão percorridos por drones, e portanto tem custo zero. Assim sendo, podemos calcular o valor de c_m da seguinte forma:

$$c_m = M \cdot \sum_{H} d_k \tag{1}$$

onde $H = \{k \in \mathcal{E} - \mathcal{D} | \delta(e) \leq K\}$. Também podemos obter o valor de c_c de forma semelhante:

$$c_c = C \cdot \sum_{\widetilde{H}} d_k \tag{2}$$

onde $\widetilde{H} = \{k \in \mathcal{E} - \mathcal{D} | \delta(e) > K\}$.

Algorithm 1 Prim
Entrada: Grafo não-direcionado ponderado $G = (\mathcal{V}, E, \delta)$
Saída: Árvore Geradora Mínima de G

Require: $|\mathcal{V}| > 0$
  $S \leftarrow \emptyset$ ▷ Conjunto de nós da árvore geradora mínima
  $\mathcal{L} \leftarrow \emptyset$ ▷ Conjunto de arestas da árvore geradora mínima
  $\pi[u] \leftarrow \infty, \forall u \in \mathcal{V}$ ▷ Vetor que armazen os pesos de cada nó
  $v \leftarrow$ loja aleatória em $\mathcal{V}$
  $\pi[v] \leftarrow 0$
  $i \leftarrow 0$ ▷ Contador auxiliar
  while $i < |V| - 1$ **do**
    $w \leftarrow$MIN_WEIGHT_NODE$(\mathcal{S}, \pi)$ ▷ w é o nó não visitado de menor peso π
    $S \leftarrow S \cup \{w\}$
    for each $u \in \mathcal{V}$ **do**
      $e \leftarrow (w, u)$
      if $e \in E \wedge u \notin S \wedge d_e < \pi[u]$ **then**
        $S \leftarrow S \cup \{u\}$
        $\mathcal{L} \leftarrow \mathcal{L} \cup \{e\}$
        $\pi[u] \leftarrow d_e$
      end if
    end for
    $i \leftarrow i + 1$
  end while

4 Análise de Complexidade

Nesta seção, será realizada a análise de complexidade de tempo e espaço do algoritmo proposto. As complexidades descritas serão expressas em termos do número de lojas N, que também representa o número de nós do grafo.

4.1 Análise de espaço

Optou-se por implementar o grafo G utilizando uma matriz de adjacência, i.e. uma matriz quadrada e simétrica, pois o grafo é não-direcionado, de tamanho $N \times N$. Além dessa estrutura, implementa-se também um conjunto de vetores também de tamanho N. Todas as outras estruturas implementadas possuem complexidade de espaço constante, e portanto é fácil obter o resultado de que a ordem de complexidade de espaço do programa implementado é $\Theta(N^2)$. O limite fornecido é tanto superior quanto inferior, pois como a implementação feita sempre depende de uma matriz de adjacência de custo quadrático, qualquer caso do algoritmo possuirá complexidade quadrática, e portanto é possível obter o supracitado limite forte para a sua complexidade de espaço.

4.2 Análise de tempo

Como a implementação do grafo em questão foi realizada utilizando-se uma matriz de adjacência ponderada, i.e. valores não nulos da matriz indicam que existe uma aresta entre um dado par de nós e também informam o peso de tal aresta, o preenchimento da mesma possui custo quadrático em qualquer cenário. Após o preenchimento da matriz, inicia-se o loop principal do algoritmo de Prim, cuja implementação escolhida também possui custo de pior caso quadrático, pois ela realiza $N - 1$ chamadas à função `min_weight_node`, que possui custo linear porque percorre todos os nós do grafo. O loop principal também realiza $N - 1$ loops sobre todos os nós do grafo. Após isso, ordena-se um vetor de tamanho N que contém os pesos das arestas da árvore geradora mínima encontrada, sendo que tal ordenação possui complexidade de tempo de pior caso de $\mathcal{O}(N \cdot \log(N))$ pois é feita utilizando uma variação do algoritmo QuickSort [2] fornecida pelo módulo de algoritmos da linguagem `C++`. O restante das operações realizadas no programa possuem custo constante ou linear, e portanto obtemos a seguinte expressão para a ordem de complexidade de tempo de pior caso para o programa:

$$2 \cdot \mathcal{O}(N^2) + \mathcal{O}(N \cdot \log(N)) + \alpha \cdot \mathcal{O}(N) + \beta \cdot \mathcal{O}(1) = \mathcal{O}(N^2) \tag{3}$$

onde α, β são constantes inteiras positivas independentes de N. Nota-se também que, como em qualquer cenário o preenchimento da matriz de adjacência é quadrático, o melhor caso do programa também possui ordem de complexidade de tempo quadrática, ou seja, é possível obter um limite forte de $\Theta(N^2)$ para a complexidade de tempo do programa.

References

[1] Almeida, J. (2021). Slides virtuais da disciplina de Algoritmos 1. Disponibilizado via moodle. Departamento de Ciencia da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

[2] Cormen, Thomas H. Algoritmos: Teoria e Prática, Editora Campus, v. 2 p. 296, 2002

A Instruções para compilação e execução

Os arquivos foram estruturados de acordo com o exigido pelo enunciado, ou seja, para executar o programa principal basta seguir os seguintes passos:

- Utilizando um terminal, execute o arquivo `Makefile`, através do comando `make`
- Após isso, execute os seguintes passos de acordo com o sistema operacional utilizado:
- No terminal Linux, execute o comando:
  `./tp02.out <TARGET_FILE>`
  Onde `<TARGET_FILE>` indica o nome do arquivo com os dados.
- Por fim, execute o comando `make clean` para remover os arquivos `.o` e `.out` gerados.