TP01 - Logística de Transporte de vacina

Rita Rezende Borges de Lima - 2020065317

[1]Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

ritaborgesdelima@dcc.ufmg.br

1. Introdução

O problema proposto nesse trabalho consiste em analisar as rotas prévias da distribuição de vacinas de uma cidade. Com a chegada da vacina de imunização do vírus Sars-CoV-2 é necessário verificar quais postos serão contemplados com o estoque pois existe um limite de tempo que a vacina pode passar depois de sair de um dos centros de distribuição. Logo, no trabalho calculamos quantos e quais postos terão acesso ao imunizante dado um incremento de temperatura (fator limitante) a cada posto passado e também verificamos se a rota feita por cada um dos motoristas é a mais eficaz (sem ciclos).

2. Implementação

O código foi desenvolvido na linguagem *c++* e compilado com *g++ -std=c++11* da GNU Compiler Collection

2.1. A Estrutura Grafo

O grafo é um tipo abstrado de dados que consiste em um conjunto finito de objetos, chamados de vértices, que em pares podem possuir algum tipo de relação, suas arestas. O trabalho é modelado de modo que postos de saúde e centros de distribuição são considerados vértices e suas rotas de um para outro, arestas. Para a implementação do grafo foi escolhida uma lista de adjacência. Os locais são indexados na estrutura da seguinte maneira: centros de distribuição possuem índice de 0 até o número de centros menos um e os índices de postos de saúde vão da quantidade de centros de distribuição até a quantidade de centros de distribuição mais a quantidade de postos de saúde menos um.

2.2. A busca em largura

Existem múltiplas formas de percorrer um grafo, uma das maneiras abordada nesse trabalho foi a busca em largura, ou BFS. Esta percorre todos os vertices a uma distância n de um vértice inicial para depois percorrer os de distância n + 1 e assim sucessivamente.

No projeto, todos os nossos centros de distribuição são vértices iniciais e todos os postos de saúde são percorridos em ordem de proximidade ao centro mais próximo, para isso armazenamos as distancias de todos os postos em um vetor e os indices dos postos que são percorridos são adicionados em um set. Quando essa distância ultrapassar 30/(incremento de temperatura), paramos de percorrer o grafo.

2.3. A busca em profundidade

Outra maneira de percorrer um grafo é a busca em profundidade, ou DFS. Esta começa num nó raiz e explora tanto quanto possível cada um dos seus ramos antes de retroceder. No projeto, chamamos uma dfs para cada um de nossos centros de distribuição, de forma que cada ramo pode ser considerada uma rota de distribuição.

Uma modificação feita na DFS do trabalho é a adição de uma verificação dentro da função. Quando estamos olhando para o próximo nó adjacente do vértice atual verificamos se este já foi visitado e se todos os seus nós vizinhos ainda não foram processados por meio de um vetor auxiliar. Caso isso ocorra, significa que encontramos um ciclo e a rota de distribuição não é a melhor possível.

2.4. Classes implementadas

2.4.1. Grafo

A classe grafo representa todos os locais de nosso mapa, suas respectivas rotas e características. Em seus atributos temos a quantidade de vértices, ou seja, postos e centros somados, um vetor com a menor distância entre cada posto e o centro de distribuição mais próximo deste, um booleano que diz se alguma rota possui um ciclo e por fim a lista de adjacência que representa o mapa de nossa distribuição. A classe também possui os seguintes métodos:

- **O construtor** da classe que recebe a quantidade de vértices do grafo e aloca espaço para este e para o vetor de distâncias para cada posto.

- **O destrutor** da classe que apenas libera o espaço ocupado pelo vetor de distâncias.

- **O método** ***set<int> bfs multi source(vector<int> centros, int max)*** que recebe como parâmetro o índice de cada centro de distribuição do grafo e a distância máxima que poderá ser percorrida. Por meio do algoritmo já explicado na sessão 2.2, os postos de saúde são processados considerando o fator limitante da distância recebida como parâmetro e seus índices são adicionados numa estrutura set que é por fim retornada.

- **O método** ***void dfs (vector<int> visitados, int v)*** uma subrotina recursiva que recebe como parâmetro o vetor de visitados e o índice do vértice que irá ser processado nessa iteração, esse último é marcado como visitado no vetor e analisamos cada um de seus nós adjacentes. Caso algum de seus vizinhos não tenho sido visitado chamamos a função para esse vértice, caso contrário se este tiver sido visitado contudo todos os seus adjacentes não, um ciclo tera sido encontrado e podemos marcar a variável booleana da classe grafo, possui ciclo, como verdadeira.

- **O método** ***void procura ciclo(vector<int> centros)*** que chama uma busca em profundidade (a função dfs) para cada um dos centros de distribuição que ainda não foram visitados.

2.5. Entradas e Saída do programa

As entradas e saídas do programa são as padrão do sistema (stdin).

2.5.1. Entradas do programa

Inicialmente recebemos 3 inteiros, C, a quantidade de centros de distribuição do mapa, P, a quantidade de postos de saúde e I o incremento de temperatura por posto em graus celsius. Nas próximas C linhas, recebemos uma quantidade variavel de inteiros. Na linha i, onde i ¡= C, cada um desses números representa o índice de um posto de sáude que tem conexão com o centro i. Nas próximas P linhas, recebemos uma quantidade variavel de inteiros. Na linha i, onde i ¡= P, cada um desses números representa o índice de um posto de sáude que tem conexão com o posto i.

2.5.2. Saída do programa

A saída do programa consiste de 3 linhas; A primeira é a quantidade de postos de saúde que serão contemplados com o imunizante dado o incremento de temperatura digitado pelo usuário. A próxima linha contém os índices dos postos que terão acesso ao imunizante, caso nenhum posto consiga o acesso será impresso um asterisco, *. Por fim, a última linha consiste de uma saída binária, 1 se alguma rota conter um ciclo e 0 caso contrário.

3. Análise de Complexidade

Agora que entendemos o funcionamento do programa podemos definir a complexidade de suas funções e de forma geral.

3.1. Tempo

Existem três operações que ocorrem no projeto: leitura, busca em largura e a procura de ciclo utilizando a busca em profundidade. Essas operaçãos são afetas pelas seguintes variáveis: quantidade de centros de distribuição, C, quantidade de postos de saúde, P, distância máxima que pode ser percorrida, D, quantidade de vértices do grafo C+P, ou N e a quantidade de arestas do grafo, M.

- **Leitura:** Por meio da entrada padrão do sistema C linhas são lidas, cada uma com as arestas de cada um dos centros de distruição. Depois são processadas P linhas, cada uma destas com as arestas do respectivo posto. Logo sua complexidade é da forma $\theta(C + P + M)$ ou $\theta(N + M)$.

- **Busca em Largura:** A bfs percorre todos os vértices que possuem uma distância para os centros menor que D. Assim, o melhor caso ocorre quando D = 0 e apenas os vértices representando os centros de distribuição serão processados, ocasionando em C vértices processados.
  O pior caso ocorre quando o incremento de temperatura for igual a 1, ocasionando D = 30, de forma que podem ser atingidos vértices de até 30 arestas de distância do centro de distribuição. Como a quantidade de arestas por vértice é algo variável, é difícil descrever um pior caso *preciso* com notação theta usando apenas o número total de arestas e vértices. Contudo, é possível afirmar com certeza que no máximo N vértices e M arestas serão processados. Logo nossa complexidade temporal é de forma: $O(N + M)$

- **Procura de Ciclo em Rota:** A função itera para cada um dos C centros de distribuição, e chama a busca em profundidade para esse vértice específico, contudo como nosso vetor de visitados não é zerado entre interações, cada vértice e arestado grafo é processado no máximo uma vez de modo que nossa complexidade temporal é de forma: $O(N + M)$

Na função main, chamamos cada uma das três operações descritas apenas uma vez, o que ocasiona na complexidade geral: $\theta(N + M) + O(N + M) + O(N + M)$ que é o mesmo que $\theta(N + M)$.

3.2. Espaço

No algoritmo todas as estruturas dependem de duas variáveis, a quantidade de centros de distribuição, C e a quantidade de postos de saúde, P. Vale lembrar que P+C é a quantidade total de vértices do grafo, N.

Usamos quatro estruturas no projeto:

- **Vetor de distancias**: seu tamanho é sempre N, de maneira que sua complexidade espacial é da forma: $\theta(N)$.

- **Vetor de visitados**: também tem como tamanho N, de maneira que sua complexidade espacial é da forma: $\theta(N)$.

- **Um set:** este possui tamanho variável, contudo sempre menor ou igual que N, de maneira que sua complexidade espacial é da forma: $O(N)$.

- **Uma lista de adjacência:** onde a quantidade de linhas é a quantidade de vértices no grafo N, e cada linha tem como tamanho a quantidade de arestas, M, que o vértice respectivo da linha tem, totalizando assim tamanho N + M, logo sua complexidade espacial é: $O(N + M)$.

4. Instruções de compilação e execução

Para a compilação do programa basta a utilização do *Makefile* presente dentro da pasta raiz do projeto executando o comando *make* no diretório. Para a execução: *./tp01*.
O algoritmo foi implementado e testado em um ambiente linux Ubuntu:

> Distributor ID: Ubuntu
> Description: Ubuntu 18.04.5 LTS
> Release: 18.04
> Codename: bionic

Utilizando o compilador *g++ -std=c++11*.

5. Conclusões

A partir da implementação do trabalho foi possível colocar em prática os conhecimentos adquiridos na disciplina a respeito de modelagem de grafos e como percorrer essa estrutura. Por meio de uma busca em largura foi possível analisar locais, ou vértices, até uma certa distância de pontos importantes de nosso mapa e com a busca em profundidade, procurar por ciclos, ou imperfeições na rota de distribuição da vacina. Dessa forma foi possível a percepção de vantagens em cada um dos algoritmos. Na busca em largura, a facilidade em calcular e estipular distâncias, e na busca em profundidade a facilidade de sua implementação e adaptação para procura de ciclos em grafo.

6. Bibliografia

- Ziviani, N. (2006).Projetos de Algoritmos com Implementações em Java e C++ Editora Cengage
- Geeks for Geeks: Detect Cycle in a Graph - Acessado em 10 de janeiro de 2021
- Kleinberg, Jon; Tardos, Eva. Algorithm Design, Pearson Education India, 2006