UFMG

Trabalho Prático I

Rede de Vacinação COVID-19

Eduardo Capanema

Matrícula 2020041515

Aluno de Graduação

Matemática Computacional

Universidade Federal de Minas Gerais

Janeiro, 2021

Submetido para Avaliação: Disciplina de Algoritmos I (Semestre 2020/2).

1

Documentação

O trabalho prático I da disciplina de Algoritmos I, do segundo semestre letivo do ano de 2020, nos solicitou que implementássemos, utilizando a linguagem de programação c ou c++, uma solução para um problema de ordem prática. Resumidamente, o problema consiste em determinar uma estrutura logística base para a distribuição de vacinas contra a COVID-19, a partir de Centros de Distribuição para diversos Postos de Vacinacão, em diferentes cenários fornecidos por meio de entradas padronizadas.

1.1 | Modelagem Computacional

A modelagem computacional que julgamos adequada para a solução do problema consistiu em utilizar Grafos Direcionados, sem peso (uma vez que as arestas não possuem características que justifiquem o uso de pesos).

A partir do padrão de entrada dos dados, criamos uma classe *Graph* responsável por fornecer um nível de abstração que aumenta a facilidade de leitura e compreensão do código proposto. A medida que lemos a entrada padrão, populamos a classe com os vértices determinados.

1.2 | Estruturas de Dados

Em nossa implementação, utilizamos algumas estruturas de dados bastante comuns. Além dos grafos direcionados sem peso, que já mencionamos acima, nosso código utilizou listas de adjacências para ordenamento dos vértices dos

grafos, vetores (*vector*) para manipulação de entradas (e.g. *sort*, *push_back*, *pop*, dentre outras).

1.2.1 | Lista de Adjacências

Utilizamos, conforme destacado acima, em nossa implementação, uma lista de adjacências, para ordenar a representação de nossos Postos de Vacinação presentes no grafo utilizado o que se mostrou extremamente útil no problema em questão.

"Em teoria dos grafos, uma lista de adjacência, estrutura de adjacência ou dicionário é a representação de todas arestas ou arcos de um grafo em uma lista."(Wikipedia - verbete 'Lista de Adjacência')

1.3 | Algoritmos

O principal algoritmo utilizado em nossa implementação foi o de Busca em Profundidade (*Depth First Search*), que é útil para realizar encaminhamentos completos ao longo de grafos direcionados, fornecendo rotas, conforme o trabalho requeria.

O algoritmo de Busca em Profundidade é largamente utilizado para diversos fins. Ressaltamos alguns ilustrativamente:

- Encontrar componentes conectados
- Encontrar componentes fortemente conectados
- Realizar ordenações topológicas
- Resolver problemas em labirintos
- Dentre outros

1.3.1 | Pseudocodigo do DFS

Nossa implementação do algoritmo de Busca em Profundidade realizou-se da seguinte forma:

```
DFS( G, u )
    u.visited = true
    for each v in G.Adj[u]
        if v.visited == false
            DFS( G, v )

init() {
    For each u in G
        u.visited = false
    For each u in G
       DFS( G, u )
}
```

É importante destacar que a implementação da solução apresentada neste trabalho não se resume tão-somente à implementação pura da Busca em Profundidade visto que existe uma condição própria do problema, qual seja, a necessidade de regularmos o encaminhamento no grafo, através da Busca em Profundidade, sob a tutela de um limitador. Este limitador é representado por uma condição referente ao decréscimo na temperatura da caixa que contém a vacina, por um fator constante que denominou-se por X e que atua cada vez que um novo Posto de Vacinação é alcançado. Em nossa implementação, utilizamos o termo *fuel*, do inglês, combustível, como sendo a quantidade de transições possíveis entre vértices do grafo, dado o X fornecido pela entrada padrão.

O limitador, opera conforme o pseudocódigo destacado abaixo:

```
for( [ todos vertices ] ) {
    nFuel = f-1; // retira etapa
        if( !visitado ) {
        if( nFuel >= 0 ){ [ RECURSIVA ] } // chamada recursiva
    }
}
```

1.4 | Complexidade

Ao final de nossa implementação, realizamos a análise da complexidade assintótica, tanto do tempo quanto do espaço (uma vez que, conforme já mencionado, utilizamos um método recursivo na implementação da Busca em Profundidade - DFS).

Nossa implementação mostrou-se prática uma vez que sua execução, em tempo e espaço, é linear, i.e. $O(V + E)$, onde V representa o número de vértices e E o número de arestas.

É importante destacar que a Busca em Profundidade (DFS) possui este caráter linear o que o torna um algoritmo considerado bastante prático e pouco custoso do ponto de vista de seu custo de tempo. Contudo, este algoritmo utiliza-se largamente de uma função recursiva em sua implementação, podendo se tornar custoso no que se refere ao uso de memória nos sistemas que o executam.

1.5 | Conclusão

Nosso trabalho foi capaz de solucionar, para as entradas padrão fornecidas e outras que fomos capazes de testar, o encaminhamento adequado ao longo do Grafo Direcionado sem peso que modelamos. Sua execução é linear, permitindo que afirmemos tratar-se de uma implementação prática e segura. As estruturas de dados são simples e comuns, tornando sua compilação descomplicada e acessível. O algoritmo utilizado em sua implementação, i.e. a Busca em Profundidade (DFS) condiz com o objeto de estudo da disciplina nesta fase do aprendizado e se presta excelentemente bem à missiva. Finalmente, gostaríamos de ressaltar a simplicidade e concisão do código proposto que toma algo em torno de 100 linhas e que apresenta boa lógica de abstração (facilitando sua leitura e compreensão), além de estar devidamente comentado e organizado.