Documentação Trabalho Prático 1 da Disciplina Algoritmos 1 - TW

Samuel Assis Vieira (2018109736)

Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG - Brasil

samuelassis@dcc.ufmg.br

1. Introdução

O trabalho prático consistia em criar um algoritmo que resolvesse o seguinte problema: quantos postos de vacinação são alcançáveis a partir dos centros de distribuições antes que a temperatura máxima possível seja atingida, dado que a cada deslocamento entre postos ou entre um posto e um centro de distribuição a temperatura das vacinas aumenta um valor x dado pelo usuário. O usuário também entra com a quantidade de locais, sendo centros de distribuições e postos, e a ligação entre eles. O programa deveria retornar quantos postos podem ser alcançados dado os parâmetros, quais são esses postos e se havia algum caminho cíclico.

2. Implementação

Para solucionar esse problema o modelei como um grafo direcionado, onde os os vértices são os postos e centros de distribuições, já as arestas são as ligações entre eles.

No caminhamento do grafo utilizei um algoritmo de busca em profundidade levemente modificado a partir de cada um dos centros de distribuição, descobrindo assim todos os postos que eram alcançáveis antes da temperatura bater o limite. Utilizei uma variação deste algoritmo de busca para identificar se o grafo era cíclico. Para representar este grafo criei uma classe Graph.

2.1. Graph

A classe representa o grafo por meio de uma lista de adjacência. Para tal utilizei a estrutura de dados *vector* da biblioteca padrão de C++. A classe conta com 6 procedimentos e funções, são eles:

- **Construtor(int,int):** recebe a quantidades de nós, centros e postos, e aloca o *vector* com esse tamanho.
- **void add_edge(int,int):** recebe um vértice de origem e um de destino e adiciona essa ligação(aresta) na lista de adjacência.
- **void route_finder(int):** recebe a distancia máxima que pode ser percorrida. Este procedimento chama uma busca em profundidade partindo de todos os centros de distribuição, imprime na tela a quantidade de postos alcançados, quais são eles e se dentro das rotas possíveis existe algum ciclo. Para retornar os postos é utilizada a estrutura de dados *set* da biblioteca padrão de C++.
- **void searcher(int, bool *, int, int, set<int>&):** procedimento que executa a busca em profundidade recursivamente a partir de um dado vértice contando os passos a cada vez que se "desce" na arvore de busca. Quando o limite de passos é batido, o procedimento para de visitar os nós e de os adicionar no conjunto dos postos possíveis.
- **int cycle_finder(int, bool*):** função com funcionamento similar com route_finder(), a função realiza a chamada de uma busca em profundidade partindo de todos os centros de distribuição buscando se, dentro do limite de passos, existe algum ciclo. Retornando 1 ou 0 para caso tenha ou não tenha ciclo, respectivamente.
- **int has_cycle(int, int, bool*, bool*, int):** função que performa a busca em profundidade a partir de um vértice dado e verifica se existe um ciclo no grafo. Ela faz essa busca verificando se algum vértice aponta para seu pai na arvore de busca. Procura arestas de retorno. Caso estas existam, então existe um ciclo e a função retorna 1. No caso contrario retorna 0.

2.2. Main:

A função main faz organização da chamada das funções de Graph e também processa as entradas dos usuários.

3. Instruções de Execução e Compilação

O programa foi todo desenvolvido em C++11. Para facilitar a compilação foi feito um Makefile. O código foi desenvolvido e testado numa maquina com processador Intel Core 2 Duo, 4GB RAM e SO Linux Mint 20 Cinnamon.

Para realizar a execução do makefile e compilação do programa basta baixar o aquivo 2018109736_SAMUELASSIS.zip, descompactar o mesmo e o acessar pelo terminal. Então digite o comando “make”, o programa sera compilado e o executável terá o nome tp01, para executa-lo basta digitar no terminal ‘./tp01’.

4. Analise de Complexidade

4.1. Complexidade de Espaço

- **Graph():** Sendo N a quantidade de vértices e M a quantidade de arestas, o construtor da classe aloca um array de *vector* de inteiros com N posições. Depois são adicionadas as arestas nesses *vector*’s. Sendo assim a complexidade de espaço pode ser dada pela seguinte expressão: (N+M)*4bytes. Uma vez que uma instância de vector gasta 4 bytes para ser armazenada.

4.2. Complexidade de Tempo

As funções **route_finder()** e **cycle_finder()** tem funcionamento muito parecida, ambas executam uma busca em profundidade no grafo a partir de cada um dos centros de distribuição, sabemos que uma busca em profundidade é O(V+A), sendo V a quantidade de vértices e A a quantidade de arestas do grafo. Sendo assim podemos afirmar que a complexidade do programa é O(C*(V+A)), sendo C a quantidade de centros de distribuição, V numero de vértices e A o numero de arestas.

5. Bibliografia

LEMIRE, Daniel. Daniel Lemire‘s blog. *In*: **The memory usage of STL containers can be surprising**. [*S. l.*], 2016. Disponível em: https://lemire.me/blog/2016/09/15/the-memory-usage-of-stl-containers-can-be-surprising/. Acesso em: 20 jan. 2021.

GEEKSFORGEEKS. Geeks for Geeks. *In*: **Depth First Search or DFS for a Graph**. [*S. l.*], 2020. Disponível em: https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/. Acesso em: 20 jan. 2021.

GEEKSFORGEEKS. Geeks for Geeks. *In*: **Detect Cycle in a Directed Graph**. [*S. l.*], 2020. Disponível em: https://www.geeksforgeeks.org/detect-cycle-in-a-graph/. Acesso em: 20 jan. 2021.

CPLUSPLUS.COM. C++ Reference. *In*: **Standard C++ Library reference**. [*S. l.*], 2020. Disponível em: https://www.cplusplus.com/reference/. Acesso em: 20 jan. 2021.