Trabalho Prático 1

Algoritmos 1

Lucas Paulo Martins Mariz

2018055016

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

lucaspaulom@hotmail.com

Abstract: *O trabalho a seguir se trata da aplicação de conceitos e algoritmos relacionados ao estudo de grafos e suas utilidades no mundo real. Além disso, questões como caminhamentos, complexidade e performance são discutidos com o intuito de concretizar e desenvolver uma capacidade analítica nos alunos para com o tema.*

Resumo: *The following work deals with the application of concepts and algorithms related to the study of graphs and their usefulness in the real world. In addition, issues such as walkers, complexity and performance are discussed in order to realize and develop an analytical capacity in students for the subject.*

1. Introdução

Foi proposto pela disciplina de Algoritmos 1 o desenvolvimento de uma estrutura que comportasse a distribuição de alguns alunos na forma de um time, no qual uma noção de hierarquia seria considerada. Para tal, foi utilizado um grafo direcionado e não ponderado *G(V, A)* contendo V alunos (vértices) e A relações (arestas) entre eles.

Uma relação R de X para Y, sendo esses vértices do nosso grafo, indica que X "comanda" Y ou, por uma outra perspectiva, Y é comandado diretamente por X. A noção de simetria não é válida para o problema, ou seja, se X comanda Y, Y não pode comandar X. Por outro lado, a transitividade é válida: se X comanda Y e Y comanda Z, X comanda Z indiretamente.

Ainda seguindo a descrição do problema, foi outorgado um teto de complexidade igual a *O(V + A)* para a execução das seguintes operações sob o grafo:

- **SWAP:** esse comando inverte uma relação entre dois vértices caso esta exista e a sua alteração não ocasione um ciclo no grafo.
- **MEETING**: indica uma ordem de fala dos alunos em uma suposta reunião, respeitando a regra em que o "comandante" fala antes de seus "comandados"
- **COMMANDER:** retorna o "chefe" mais novo de um dado vértice.

Além disso, a padronização das entradas bem como a unicidade das idades de cada aluno constituem fatores de suma importância para a veracidade da saída.

2. Instruções de Compilação e Execução

O programa foi desenvolvido na linguagem C, compilado pelo GCC versão 7.4.0 em um sistema operacional Ubuntu 18.04. Todo o código foi executado em meu notebook cujas especificações principais estão descritas abaixo:

- Intel Core i5-7200U CPU 2.5GHZ x 4
- Nvidia G-Force GTX 840M
- 8GB RAM
- SSD Kingston A400 240GB
- Ubuntu 18.04.2 LTS

Portanto, recomenda-se que a execução do código ocorra em um sistema Ubuntu versão 14 ou superior contendo pelo menos a versão 7 do GNU C Compiler (GCC). Foi utilizado um makefile que se encarrega de todas as operações de compilação dos arquivos '.c'.

A execução do projeto aguarda um argumento (argv) indicando o arquivo contendo os dados de entrada e deve seguir o seguinte padrão:

./tp1 path/to/file.txt

3. Implementação

Para a implementação de uma estrutura que respeitasse o grafo e a ordem de complexidade imposta foi utilizada a ideia de uma lista de adjacências. Sendo assim, foi montada uma estrutura de dados contendo um vetor com as idades de cada aluno, um vetor de listas representando o "comandados" de cada vértice, o número de membros do time (N), o número de relações (M), o número de instruções (I), entre outros dados.

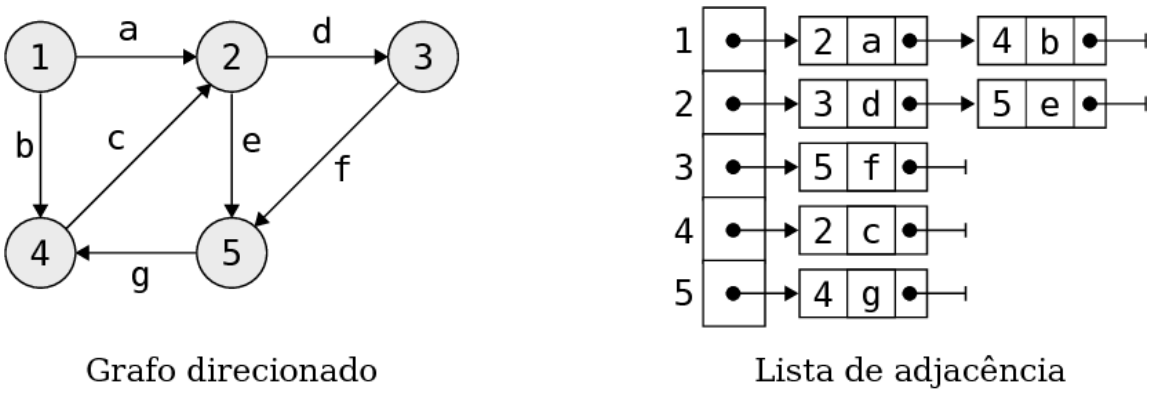

Figura 1: Lista de Adjacências – grafo meramente ilustrativo. Disponível em <https://www.researchgate.net/figure/Figura-A2-Representacao-de-redes-por-meio-de-lista-de-adjacencia-composta-por-n-listas_fig12_279962622>. Acesso em 27 Set 2019.

3.1. Swap

A função de swap, como mencionado anteriormente, inverte a hierarquia estabelecida entre dois vértices, ou seja, caso A seja o chefe de B, após a inversão, B será o chefe de

A. No entanto algumas verificações devem ser realizadas com o intuito de manter / realizar ou não as alterações. São elas

- Caso não exista uma aresta de A para B, e nem de B para A, o swap não deve ser realizado;
- Caso o swap entre A e B ocasione um ciclo as alterações devem ser desfeitas.

Sendo assim, o algoritmo foi desenvolvido da seguinte forma: primeiro verificamos se existe uma aresta de A para B. Caso não exista, realizamos a mesma verificação de B para A. Caso não exista, nada é realizado. Por outro lado, se alguma das verificações anteriores retornar resultados positivos, removemos a aresta do primeiro vértice e adicionamos no segundo. Após isso checamos ciclos no grafo.

Para checar um ciclo foi utilizado a ideia do algoritmo de DFS que colore os vértices de cinza e preto. Em suma, caso estejamos fazendo o caminhamento em profundidade pelo grafo e ocorra a visitação de um vértice pintado de cinza (visitado), o grafo possui um ciclo. Assim sendo, as alterações no grafo são desfeitas ou mantidas caso o mesmo possua um ciclo ou não, respectivamente.

3.2. Commander

A função commander retorna o chefe mais novo de um vértice A. Para tal, foi adotado o seguinte método:

- Inverte-se todas as arestas do grafo;
- Aplica-se a busca em profundidade a partir do vértice A;

Ao aplicar a DFS no vértice, passaremos recursivamente por todos os alunos alcançáveis por A, ou seja, todos os comandantes (que agora são comandados devido a inversão) de A. Dessa forma, basta ir armazenando a menor idade encontrada.

Com o intuito de manter a estabilidade da estrutura original, foi criado um grafo auxiliar para essa tarefa, no qual as arestas inseridas de forma invertida.

3.3. Meeting

O comando de meeting é responsável por apresentar uma das possíveis ordens de fala dos alunos que respeite a hierarquia das relações – os chefes falam antes de seus subordinados.

A solução proposta consiste na montagem da ordem topológica do grafo (DAG), a qual corresponderá na ordem de fala dos alunos. Assim, aplicamos a DFS no grafo e, quando um vértice finaliza, o mesmo é inserido em uma pilha. No final, imprimimos o conteúdo da pilha normalmente.

4. Análise Experimental

Uma massiva quantidade de dados gerados foi considerada na análise que se segue. Foram realizados diversos testes com o intuito de checar a veracidade dos dados. Os dados obtidos com os experimentos realizados estão armazenados no repositório do projeto no github (https://github.com/LucasPMM/Blackjack_Team) e na própria pasta do projeto.

Foram gerados grafos contendo o número máximo de arestas ((n^2 – n) / 2) – em que n é o número de vértices. Para esses, foi atribuída uma instrução de cada tipo (swap, meeting, commander) e para cada número de vértices foram realizadas 100 execuções, de forma a obter um valor satisfatório para a média.

Inicialmente, fiz os testes na minha máquina local, o que resultou no gráfico da figura 2. Analisando esse gráfico e discutindo com alguns colegas, foi constatado que sua aparência se deu pelas constantes trocas de contexto existentes no processador. Ações como navegar na internet ou estar com outros programas abertos na máquina no momento da execução do trabalho implicou em inconstâncias nos resultados e na produção de diversos outliers (pontos fora da curva) na contagem do tempo de execução. Com o intuito de contornar essa situação, foi simulado um ambiente virtual Ubuntu na plataforma Codenvy e os mesmos testes foram refeitos, gerando o gráfico da figura 3. Pode-se perceber a diferença entre os dois gráficos tanto na linearidade quanto na velocidade de execução em cada caso. Portanto, por se apresentar de maneira mais coesa e consistente, será considerado na análise o gráfico da figura 3.

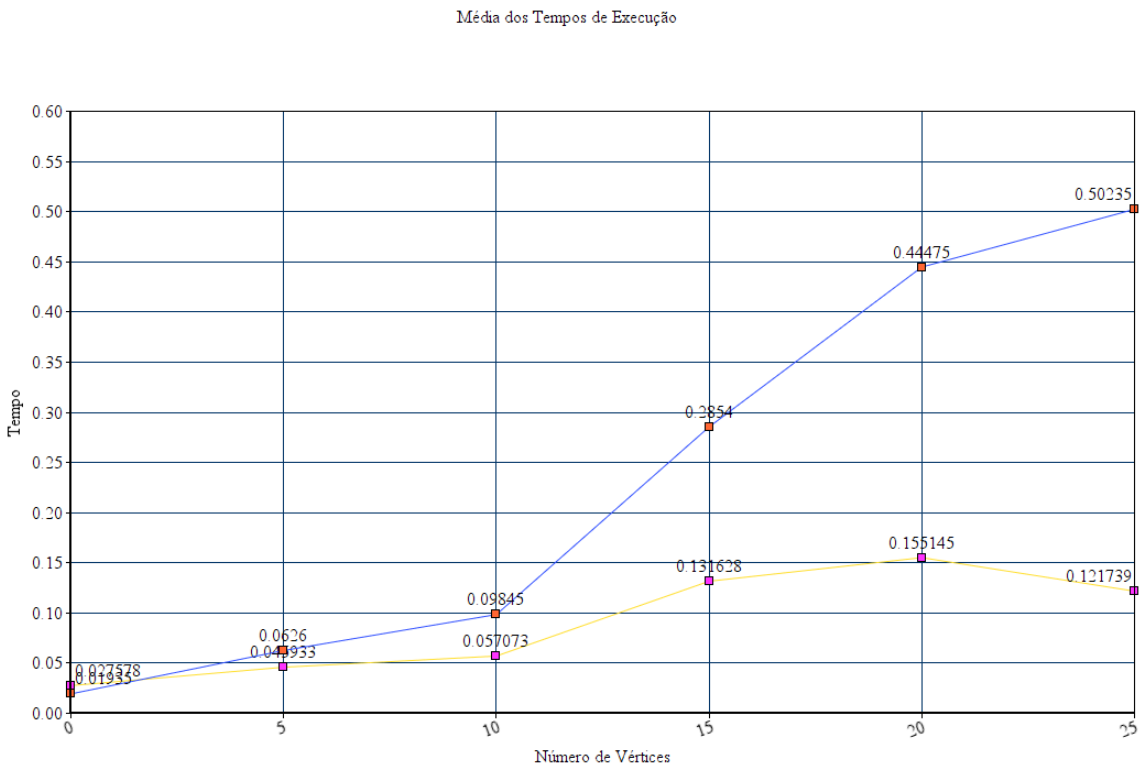

Figura 2: Média dos tempos de execução X Número de vértices (gerado no ambiente local)

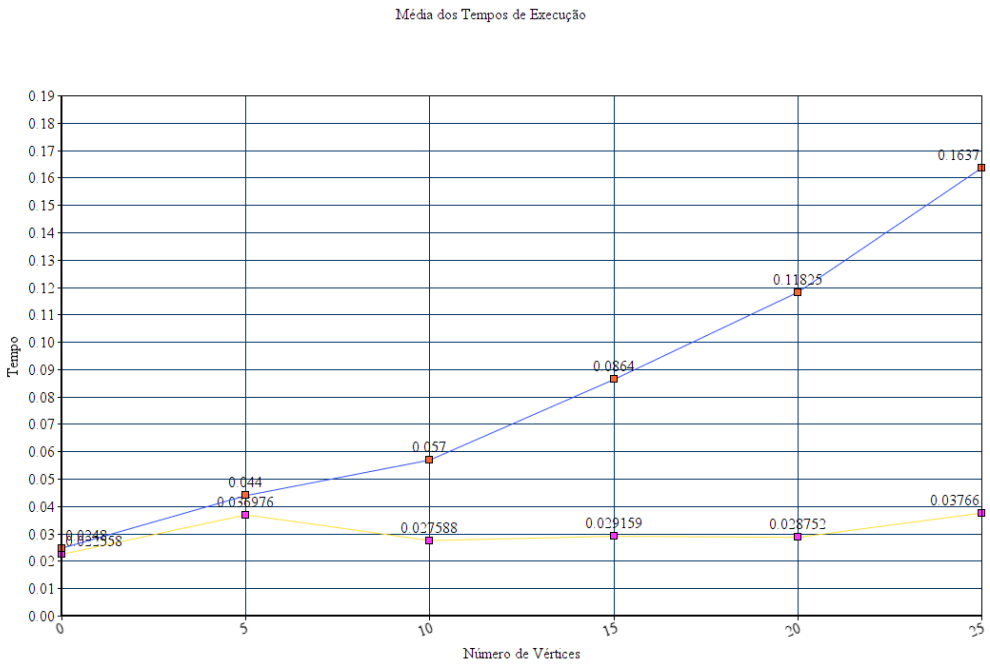

Figura 3: Média dos tempos de execução X Número de vértices (gerado no ambiente virtual / nuvem)

Seja V o número de vértices e A o número de arestas, vamos analisar alguns pontos do código a fim de determinar a ordem de complexidade do mesmo.

A função de **swap** busca pela aresta e vai percorrer, no máximo, todas as arestas do grafo, sendo limitado superiormente por *O(V)*. Já o processo de checagem de ciclo implementa uma DFS, a qual possui uma complexidade de *O(V+A)*. Sendo assim, a função de swap é limitada superiormente por *O(V+A)* quando estamos analisando o tempo. Em relação ao espaço pode-se afirmar que a complexidade é a mesma que a do tempo pois a estrutura de dados seguiu um padrão de listas de adjacências, e, para fazer o swap, removemos um item de uma aresta e adicionamos em outra, mantendo constante o tamanho do grafo.

A função **commander** busca pelo chefe mais jovem de um dado vértice e, para tal, é implantada uma DFS com uma complexidade igual a *O(V+A)*. Já em relação ao espaço, temos a criação de um grafo auxiliar para a manipulação de arestas, contudo, esse fato não impacta na ordem de grandeza relacionada a complexidade, a qual se mantem *O(V+A)*.

A função **meeting** apresenta uma possível ordem de fala em uma reunião dos membros do time de forma que a hierarquia seja respeitada. Para montar a ordem topológica utilizamos uma DFS, resultando em uma complexidade de tempo igual a *O(V+A)*. Em relação ao espaço, é criada uma nova pilha para armazenar os vértices, sendo *O(V)*. No entanto, ainda continua sendo limitada superiormente por *O(V+A)* levando em consideração a implementação das listas de adjacências.

Sendo assim, podemos concluir que todo o projeto é limitado superiormente por uma complexidade igual *O(V+A)* e esse fato é refletido no gráfico da figura 3 gerado a partir dos dados da execução dos testes. Pode-se notar que o crescimento da média do tempo de execução é, aproximadamente, linear e está diretamente relacionado a quantidade de vértices e arestas contidos no grafo.

Com todos esses dados, podemos responder as perguntas propostas:

- Por que o grafo tem que ser dirigido?

  Toda a noção de hierarquia estabelecida pelo tema envolve os “chefes” e os “comandados” e essa distinção só pode ser feita através da direção da relação. Dessa forma, se existe uma relação de A para B temos a certeza de que A comanda B e não ao contrário, implicando na necessidade de o grafo ser dirigido.

- O grafo pode ter ciclos?

  O grafo não pode ter ciclos por dois motivos:

  1. Caso ocorra um ciclo estaríamos quebrando a regra de comandado ser chefe de seu chefe direta ou indiretamente, pois a noção de transitividade está presente nas relações.
  2. Não seria possível encontrar a ordem topológica do grafo caso ele tenha ciclo, afetando assim o desenvolvimento do comando de meeting.

- O grafo pode ser uma árvore? O grafo necessariamente é uma árvore?

  Removendo a noção das arestas dirigidas, o grafo poderia ser uma árvore levando em consideração que o mesmo não possui ciclos. Por outro lado, ele não é necessariamente uma árvore, pois teríamos nós com mais de um pai.

5. Conclusão

Com a realização dos testes e análise dos dados, é notável que diversos problemas reais podem ser mapeados utilizando grafos e manipulados com os mais diversos algoritmos de caminhamento, busca, entre outros, dependendo do objetivo buscado. Além disso, é necessário destacar que alguns erros de medida podem ter ocorrido devido a troca de contexto existente no processador do computador, fazendo com que processos percam ou ganhem prioridade na execução, oscilando o tempo final de execução do trabalho. No entanto, essa situação foi contornada ao executar o trabalho em um sistema remoto livre, em grande parte, das trocas de contexto locais. Os resultados gerados por ambas os casos têm suas importâncias, pois através deles podemos entender mais como a máquina funciona e os agentes internos e externos que à afetam.

Sendo assim, a análise qualitativa de um problema é de fundamental importância tendo em vista a procura pela melhor solução em termos de complexidade e velocidade da execução. Pode-se afirmar que o trabalho teve grande importância no processo pedagógico da disciplina, colocando em pratica e provando conceitos vistos em sala, bem como pode-se dizer que os objetivos foram cumpridos e as dificuldades superadas uma vez que as metas foram alcançadas.

6. Bibliografia

Chaimowicz, L. e Prates, R. “Ordenação: Quicksort”, pdf disponível no Moodle da turma de Estruturas de Dados 2019/1 da UFMG. Acesso em: 5 jun 2019.

Ziviani, Nívio. (2011), Projeto de Algoritmos com Implementações em Pascal e C, Cengage Learning, 3° Ed.