Universidade Federal De Minas Gerais - UFMG

RELATÓRIO DO TRABALHO PRÁTICO 1

Disciplina Algoritmos 1

Aluna: Júlia Amorim de Araujo
Matrícula: 2016058344

Belo Horizonte, Setembro 2019.

1) Introdução:

O objetivo do programa a ser desenvolvido é auxiliar a equipe de BlackJack da UFMG a ganhar muito dinheiro em uma viagem pelos cassinos de Las Vegas jogando BlackJack e utilizando a estratégia de contagem de cartas sem serem pegos, uma vez que apesar de não ser contra a lei, a prática pode levar a expulsão do cassino.

A contagem de cartas, de forma sucinta, é a estratégia de “adicionar 1 para cartas baixas e subtrair 1 para cartas altas. Quando o registro aumenta (o que significa mais cartas altas do que as cartas baixas no baralho), é hora de começar a fazer apostas mais altas”.

Observando a prática da estratégia pela equipe de BlackJack do MIT e querendo evitar as mesmas dificuldades, foi proposto que a equipe da UFMG se organizasse de forma hierárquica dividindo as responsabilidades para cada membro da equipe e essa mesma estrutura seria remodelada de tempos em tempos para evitar suspeitas.
Os alunos também poderiam discutir o andamento desde que seguindo a ordem da hierarquia para a fala nas reuniões.

Para tornar tudo isso possível e facilitar o gerenciamento, como estudante da computação minha responsabilidade foi de desenvolver um programa modelando um grafo que conecta os alunos participantes da equipe de acordo com a devida hierarquia na complexidade dominante de O(V + A) onde V são os vértices do grafo e A as arestas, e que realiza 3 instruções:

(1) SWAP: essa instrução garante a rotação da hierarquia da equipe pois, modelando o problema no formato de um grafo, através do swap garanto que existe uma relação de hierarquia e que a troca de A→B para A□B não vai gerar um ciclo no grafo;
(2) COMMANDER: Com essa função consigo identificar qual a pessoa mais jovem que comanda direta ou indiretamente um aluno;
(3) MEETING: Essa instrução auxilia a identificar uma possível ordem de fala (considerando a hierarquia) nas reuniões;

2) A solução:

Para solucionar o problema indicado acima, foi modelado um grafo no qual os vértices representam os alunos e as arestas a hierarquia entre eles. O grafo é direcionado e a direção dele é de acordo com qual vértice comando qual, ou seja, os vértices (alunos) comandantes apontam para os vértices (alunos) comandados. O grafo não é ponderado.

Questões a serem respondidas sobre a modelagem do grafo para solução do problema:
• Por que o grafo tem que ser dirigido?
Para ser possível modelar a hierarquia entre os alunos, no grafo direcionado conseguimos utilizar a direção das arestas para representar qual aluno comanda qual.

• O grafo pode ter ciclos?
Não, porque se um aluno X comanda algum aluno Y, o aluno Y não pode comandar diretamente ou indiretamente o aluno X. Se fosse possível esses tipo de hierarquia o grafo teria ciclos.

• O grafo pode ser uma árvore? O grafo necessariamente é uma árvore?
O grafo pode ser uma árvore porém não necessariamente precisa ser uma árvore, isso porque as característica de um árvore são: Um grafo conectado e direcionado que não possui ciclos, possui um vértice como raiz, e onde cada vértice possui um único caminho até ele (um único comandante). Como no problema um mesmo aluno pode ser comandando por outros dois alunos, o grafo é sempre um DAG (grafo direcionado acíclico) mas não **obrigatoriamente** sempre uma árvore.

Para implementação do grafo utilizamos uma lista de adjacência que possui complexidade de memória (ou espaço) O(V+A) onde V é número de vértices do grafo e A o número de arestas. Isso porque na lista de adjacência é armazenado o exato número de vértices e arestas que existem no grafo.

A resolução e implementação de cada uma das instruções citadas se deu da seguinte forma:

1) Instrução **SWAP**:
O ponto principal de uma boa implementação de solução para essa instrução é a verificação de ciclos no grafo.

Primeiro é feita a transposição da direção da aresta indicada no entrada do programa, depois é verificado a existência de um ciclo utilizando o algoritmo de busca em profundidade DFS modificado.

A implementação do DFS foi realizada tal que ele caminha pelo grafo marcando quais arestas já foram visitadas e verifica caso uma aresta marcada (já visitada) seja visitada novamente através do mesmo vértice ‘pai’ retornando nesse caso que existe um ciclo. Se o ciclo existe a aresta é transposta novamente (volta a direção original) e o programa imprime na tela a mensagem “S N” indicando que não foi possível realizar a troca.

2) Instrução **COMMANDER**:
A instrução COMMANDER primeiro transpõe o grafo (inverte a direção de todas as arestas) e logo após roda um algoritmo BFS (algoritmo de busca em largura) percorrendo a partir do vértice indicado no parâmetro da função (especificado no arquivo de entrada) e marca todos os vértices que conseguir alcança a partir desse primeiro comparando as idades para encontrar o comandante com a idade menor.

Um detalhe importante para destacar é que a transposição do grafo é feita para tornar possível percorrer o grafo na direção dos comandantes.

3) Instrução **MEETING**:

Essa instrução em suma faz apenas a ordenação topológica do grafo que nada mais é do que posicionar o nosso grafo direcionado acíclico (DAG) na “horizontal” tal que cada arestas (A_i, A_j) temos $i < j$.

Por meio da ordenação topológica conseguimos apresentar a ordem da hierarquia do grafo como uma possível ordem de fala nas reuniões. A ordenação topológica utiliza o algoritmo de DFS original.

3) Análise de complexidade e espaço:

Como já citado foi utilizada a implementação do grafo via lista de adjacência que possui complexidade de espaço O(A+V) onde A é o número de arestas e V o número de vértices. A complexidade de busca por uma aresta é de grau(V) onde grau é o número de arestas que sai de um vértice V.

As instruções de SWAP e MEETING ambas utilizam o algoritmo DFS que apesar de modificado na instrução SWAP não foi adicionado nenhuma operação que altere a complexidade de tempo do algoritmo que é de O(V + A) onde A é o número de arestas e V o número de vértices.

A instrução COMMANDER utiliza o algoritmo de busca em largura BFS que também tem complexidade de tempo igual a O(V+A) onde A é o número de arestas e V o número de vértices.

A prova da complexidade dos algoritmos BFS e DFS é igual, apesar do BFS funcionar visitando os vértices e separando-os em camadas e já o DFS visita os vértices a partir um inicial até o último vértice “filho” ambos possuem complexidade de visita ou marcação de um vértice ou arestas igual a O(1) e no pior caso ambos vão visitar todos os vértices e arestas sendo então complexidade de O(V+A) considerando nossa implementação do grafo na lista de adjacência com complexidade de espaço O(V+A) onde A é o número de arestas e V o número de vértices.

Como o programa tem como funções dominantes o BFS e o DFS concluímos que a complexidade do programa como um todo é tal que O(V+A). E a complexidade de espaço também é de O(V+A) já que o nosso armazenamento de dados dominante é a da lista de adjacência.

4) Compilando o programa:

O programa foi escrito na linguagem de programação C++ e conta com um arquivo MakeFile para auxiliar a compilação.

Para rodar o programa desenvolvido basta acessar o diretório com os arquivos do programa descompactados juntamente com o arquivo de texto contendo os dados de entrada através

do terminal de comando e digitar o comando “make” e logo após digitar o comando “./tp1 nomearquivo.txt” para executar o programa com o nomearquivo.txt como entrada.

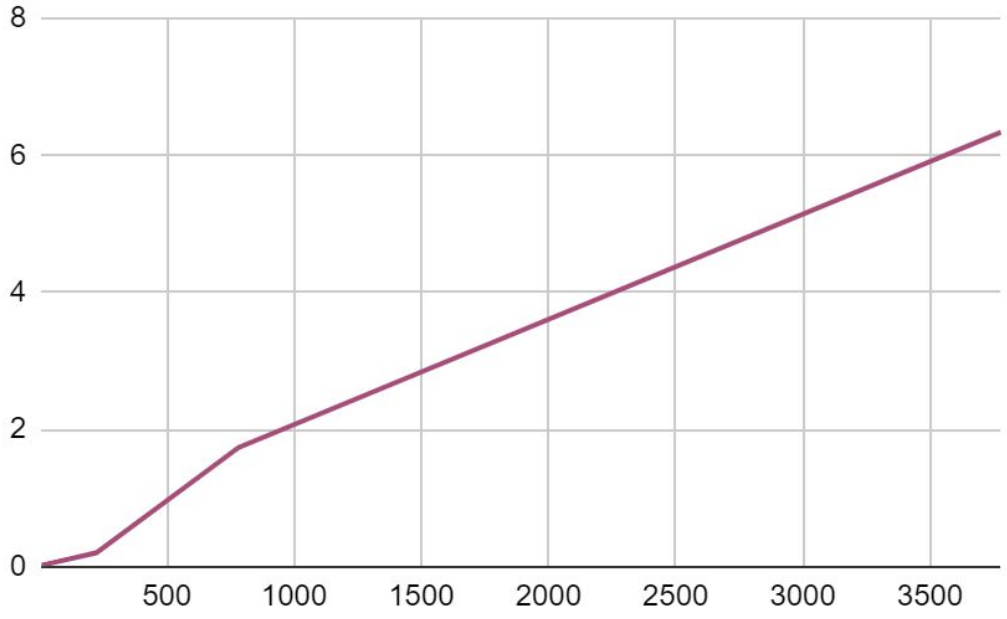
Se necessário para fins de teste, por exemplo, é possível deletar os arquivos .o gerados na etapa da compilação com o comando “make clean” também via terminal comando.

5) Avaliação experimental:

Os valores de média e desvio padrão apresentados foram baseados na execução seguida de 10 vezes com a entrada no tamanho indicado.

Vértices	7	223	782	3780
Comandos	8	136	985	114
Média	0,0135	0,19611	1,73367	6,342
Desvio padrão	0,00393	0,01336	0,04089	0,01643

Número de vértices x média do tempo de execução



Observamos que o tempo de execução aumenta linearmente de acordo com o aumento no tamanho de vértices do grafo de entrada o que já era esperado considerando que o programado desenvolvido respeita a ordem de complexidade linear O(V + A) onde V são vértices e A são arestas.

6) Conclusão:

Concluímos que o objetivo inicial do desenvolvimento do programa foi alcançado dentro da complexidade máxima solicitada de O(V + A). Os alunos agora podem se organizar de forma automatizada para aplicar a estratégia e fazer muito dinheiro jogando BlackJack pelo cassinos de LasVegas.