Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Bacharelado em Sistemas de Informação

TRABALHO PRÁTICO: EQUIPE DE BLACKJACK

Victor Gabriel Ferreira Moraes

Belo Horizonte,

2019

Descrição do Problema

O contexto deste trabalho consiste na criação de um sistema responsável por organizar e gerenciar uma equipe do jogo Blackjack, composta por alunos da Universidade Federal de Minas Gerais (UFMG), no intuito de participar de competições nos casinos de Las Vegas.

Em relação a equipe, essa é organizada por uma estrutura hierárquica, onde existem alunos com cargos de comandante, que possuem prioridade de fala durante as reuniões da equipe e são responsáveis por comandar determinados alunos que são tidos como subordinados. Essa estrutura é dinâmica, de modo que com o tempo, os alunos são reorganizados e as funções de comandante e subordinado são frequentemente trocadas entre si, passando a forma uma nova hierarquia dentro da equipe.

Posto isso, a solução proposta para o problema, consiste em um sistema responsável por fazer a manutenção das trocas de cargo entre os alunos na estrutura de hierarquia das equipes, além de ser capaz de obter informações sobre o relacionamento de determinado aluno com seus comandantes, sendo possível obter a idade dos envolvidos, e por fim, faz-se necessário uma função que informe, de forma hierárquica, uma possível ordem de fala dos alunos nas reuniões.

Estrutura de dados e Algoritmos

O sistema proposto para solucionar o problema apresentado, foi estruturado contendo as seguintes classes: Grafo.cpp, Gerenciador.cpp e main.cpp, de modo que no decorrer de seu desenvolvimento, foram utilizados estruturas de dados auxiliares da biblioteca padrão da linguagem C++, tais como vector, stack e list.

Em relação aos algoritmos utilizados para suprir as tarefas requisitadas pelo problema, esses foram desenvolvidos de modo a respeitar a complexidade limite exigida de O(m + n), onde m representa a quantidade de arestas e n a quantidade de vértices da estrutura de grafos utilizada para representar a equipe de alunos.

Além disso, a parte conceitual de grafos e os algoritmos como DFS e BFS utilizados, foram essenciais no processo de desenvolvimento do sistema.

Classe Grafo

Análise de Complexidade de Tempo

A classe Grafo.cpp corresponde a uma estrutura de dados criada para representar as equipes de alunos do sistema, através da representação de um grafo direcionado por lista de adjacência. Nessa classe, existe duas struct internas, uma para representar os vértices e outra representando as arestas, além disso, há também um vector de vértices, representando a lista de adjacência do grafo.

Em relação aos métodos, existe um construtor default, que possui complexidade constante O(1), um construtor com parâmetros, utilizado para inicializar o grafo, percorrendo, dessa forma, todos os vértices (n) e todas as arestas (m) a fim de preencher a lista de adjacência do grafo, gerando uma complexidade de O(m + n). Além disso, a classe possui os métodos “setListaVertices”, “getListaVertices” e “getQuantidadeVertices”, ambos métodos que ou atribuem ou recuperando o valor dos atributos do grafo, e que portanto realizam essas operações em uma complexidade constante O(1).

Análise de Complexidade de Espaço

Em relação à complexidade de espaço da classe Grafo.cpp, essa consiste basicamente no tamanho ocupado pela lista de adjacência, que no caso, é composta por uma lista de n vértices do grafo, onde cada vértice possui uma lista para representar suas arestas de adjacências, sendo m arestas no total, visto que trata-se de um grafo direcionado. Dessa forma, a complexidade de espaço da classe Grafo.cpp, é da ordem de O(m + n).

Classe Gerenciador

Análise de Complexidade de Tempo

A função destinada para a classe Gerenciador.cpp no sistema, é a de coordenar e executar todas as três operações propostas pelo problema (swap, commander e meeting), por meio da utilização da classe Grafo.cpp como estrutura de dados principal e o uso de demais funções criadas para auxiliar na execução dessas operações.

A função swap, realiza uma chamada a uma função auxiliar “isVerticesAdjacentes”, que verifica se os dois vértices passados são adjacentes por meio da chamada de duas funções find(), que ocorre em tempo linear, sobre a lista de adjacência dos dois vértices, portanto gerando uma complexidade de O(grau(u) + grau(v)), onde o pior caso é O(m). Também é chamada a função “trocaVerticesAlunos”, que realiza a troca entre dois vértices em tempo linear O(m). E por fim, é executado a função “contemCiclo”, que verifica se existe um ciclo no grafo, por meio da realização de uma Deep First Search (DFS) de forma recursiva, possuindo complexidade O(m + n). Dessa forma, a função swap possui de modo assintótico uma complexidade O(m + n).

Em relação a operação commander, essa inicializa um vetor que indica os vertices ja visitados, tendo portanto complexidade O(n), em seguida chama a função “transporGrafo”, que gera a versão transposta do grafo, inicializando o novo grafo (n) e em seguida passando todos os elementos transpostos para a lista de adjacencia (m + n), gerando uma complexidade de O(m + n). Finalmente, é executado a função “defineComandanteMaisJovem”, que determina o integrante mais jovem que comanda o aluno informado por meio da realização de uma Breadth Search Tree, logo sua complexidade é O(m + n). Portanto, o resultado da operação commander corresponde a um custo assintótico da ordem de O(m + n).

Por último, o comando meeting inicializa o vetor de vértices visitados, com uma complexidade O(n), e realiza uma ordenação topológica dos vértices, colocando-os em uma pilha. Esse processo é feito ao percorrer o grafo por meio da

realização de uma DFS, logo sua complexidade é de O(m + n). Para finalizar, o método imprime os n vértices conforme são desempilhados da pilha, gerando complexidade O(n). Conclui-se então que o método de meeting possui uma complexidade assintótica de O(m + n).

Análise de Complexidade de Espaço

Por sem uma classe destina a coordenação e execução dos métodos do sistema, a classe Gerenciador.cpp não possui atributos e apenas utiliza as estruturas de dados padrão da biblioteca de C++ (vector, stack e list) e a estrutura de grafos da classe Grafo.cpp, dessa forma sua complexidade de espaço é assintoticamente dominada por uma complexidade O(m + n), onde m é o número de arestas e n o número de vértices do grafo.

Classe Main

Análise de Complexidade de Tempo

A classe main é encarregada de realizar a leitura e escrita dos dados, processo esse feito em tempo linear, e em realizar a chamada da execução dos métodos conforme os parâmetros passados. Dessa forma, a complexidade da classe main.cpp é da ordem de O(m + n), que equivale a complexidade a qual todos os comandos do sistema analisados convergem.

Análise de Complexidade de Espaço

Referente à complexidade de espaço da classe main.cpp, sua análise é análoga à da classe Gerenciador.cpp, visto que a mesma apenas faz uso das estruturas de dados padrão da biblioteca e da estrutura de grafos. Portanto, sua complexidade também é da ordem O(m + n).

Avaliação experimental

As avaliações experimentais realizadas para o sistema desenvolvido foram realizadas por meio da execução de 11 comandos, sendo 4 commanders, 4 meetings e 3 swaps, dobrando a quantidade de vértices e arestas do grafo para cada caso de teste, sendo que ao todo foram realizados 3 cenários distintos de teste, rodando 10 vezes cada um, totalizando em 30 execuções.

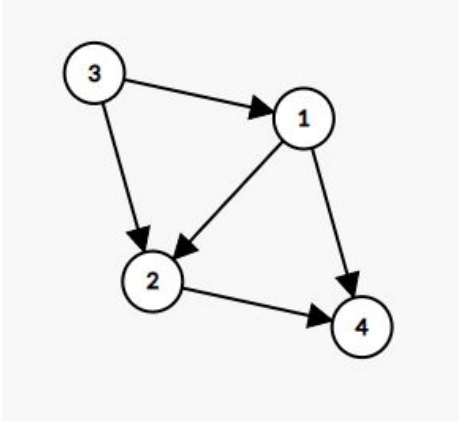
A medição do tempo de execução foi com base no tempo gasto dividido pela quantidade de clocks por segundo, realizada conforme código fornecido e encontra-se disponível na seção de referências deste trabalho.

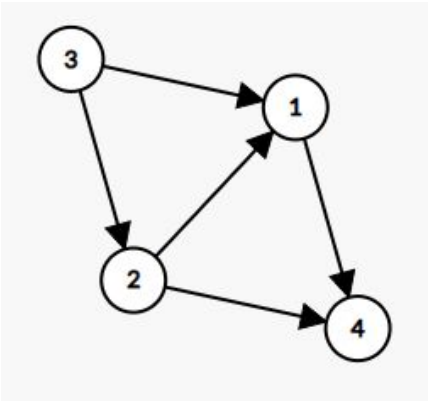
Experimento 1

Grafo com 4 vértices, 5 arestas e 11 comandos (4 C, 4 M, 3 S).

Diagrama e Comandos

Entrada: **Saída:**





Idades respectivas: 23 61 57 35

Comandos:

Entrada	Saída	Descrição
C 1	C 57	Idade aluno mais novo que comanda 1.
C 2	C 23	Idade aluno mais novo que comanda 2.

C 3	C *	Nenhum aluno comanda 3.
C 4	C 23	Idade aluno mais novo que comanda 3.
M	M 3 1 2 4	Possível ordem de fala dos alunos.
S 1 2	S T	Troca realizada entre hierarquia de 1 e 2.
M	M 3 2 1 4	Possível ordem de fala dos alunos após troca.
S 3 4	S N	Não realiza troca pois vértices não são adjacentes.
M	M 3 2 1 4	Ordem de fala permanece igual.
S 1 3	S N	Não realiza troca pois gera ciclo no grafo.
M	M 3 2 1 4	Ordem de fala permanece igual.

Resultado

Número	1	2	3	4	5	6	7	8	9	10
Tempo de execução	8	6	13	15	8	7	9	8	14	10

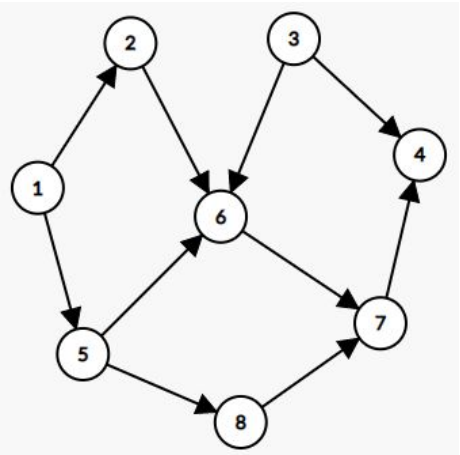
Média	**Desvio Padrão**
9,8	3,120

Experimento 2

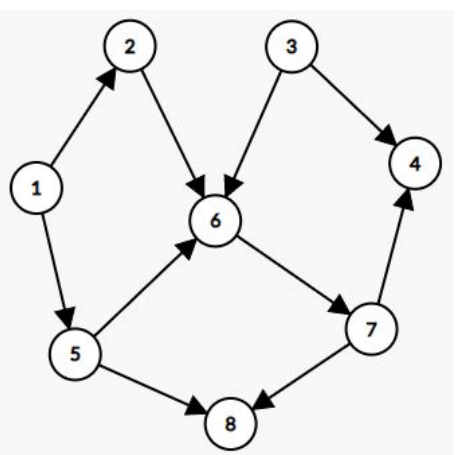
Grafo com 8 vértices, 10 arestas e 11 comandos (4 C, 4 M, 3 S).

Diagrama e Comandos

Entrada:



Saída:



Idades respectivas: 27 48 33 16 41 21 39 24

Comandos:

Entrada	Saída	Descrição
C 1	C *	Nenhum aluno comanda 1.
C 5	C 27	Idade aluno mais novo que comanda 5.
C 7	C 21	Idade aluno mais novo que comanda 7.
C 8	C 21	Idade aluno mais novo que comanda 8.
M	M 3 1 5 2 6 7 8 4	Possível ordem de fala dos alunos.
S 7 8	S T	Troca realizada entre hierarquia de 7 e 8.
M	M 3 1 5 8 2 6 7 4	Possível ordem de fala dos alunos após troca.
S 3 7	S N	Não realiza troca pois vértices não são adjacentes.
M	M 3 1 5 8 2 6 7 4	Ordem de fala permanece igual.
S 3 4	S N	Não realiza troca pois gera ciclo no grafo.
M	M 3 1 5 8 2 6 7 4	Ordem de fala permanece igual.

Resultado

Número	1	2	3	4	5	6	7	8	9	10
Tempo de execução	9	10	10	16	11	17	8	20	16	12

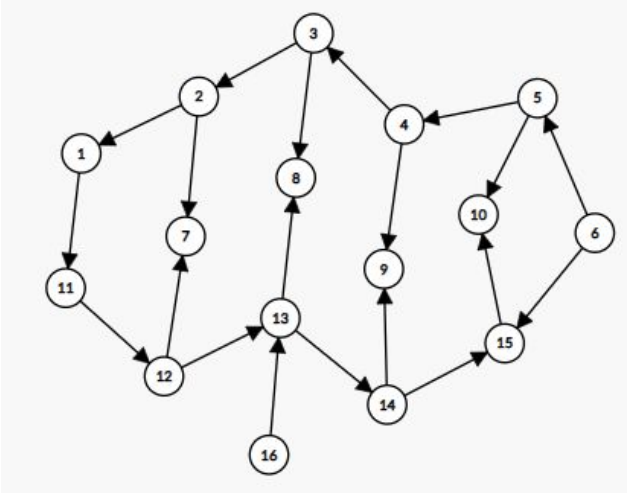
Média	**Desvio Padrão**
12,9	4,040

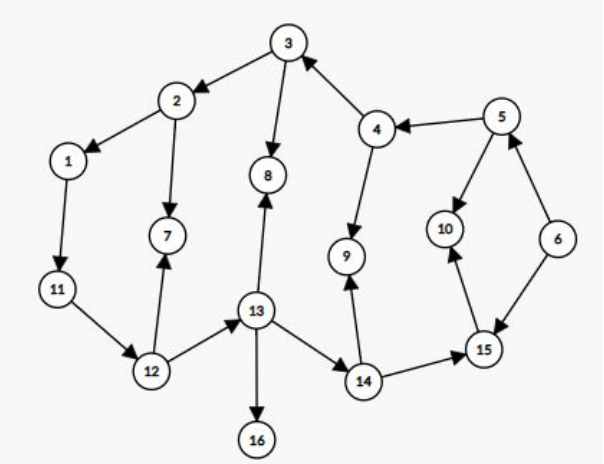
Experimento 3

Grafo com 16 vértices, 20 arestas e 11 comandos (4 C, 4 M, 3 S).

Diagrama e Comandos

Entrada: **Saída:**





Idades respectivas: 35 24 49 21 48 23 15 14 32 20 43 30 34 31 10 33

Comandos:

Entrada	Saída	Descrição
C 3	C 21	Idade aluno mais novo que comanda 3.
C 6	C *	Nenhum aluno comanda 6.
C 10	C 10	Idade aluno mais novo que comanda 10.
C 15	C 21	Idade aluno mais novo que comanda 15.
M	M 6 5 4 3 2 1 11 12 13 16 14 15 10 9 8 7	Possível ordem de fala dos alunos.
S 13 16	S T	Troca realizada entre hierarquia de 13 e 16.
M	M 16 6 5 4 3 2 1 11 12 13 14 15 10 9 8 7	Possível ordem de fala dos alunos após troca.
S 7 8	S N	Não realiza troca pois vértices não são adjacentes.
M	M 16 6 5 4 3 2 1 11 12 13 14 15 10 9 8 7	Ordem de fala permanece igual.

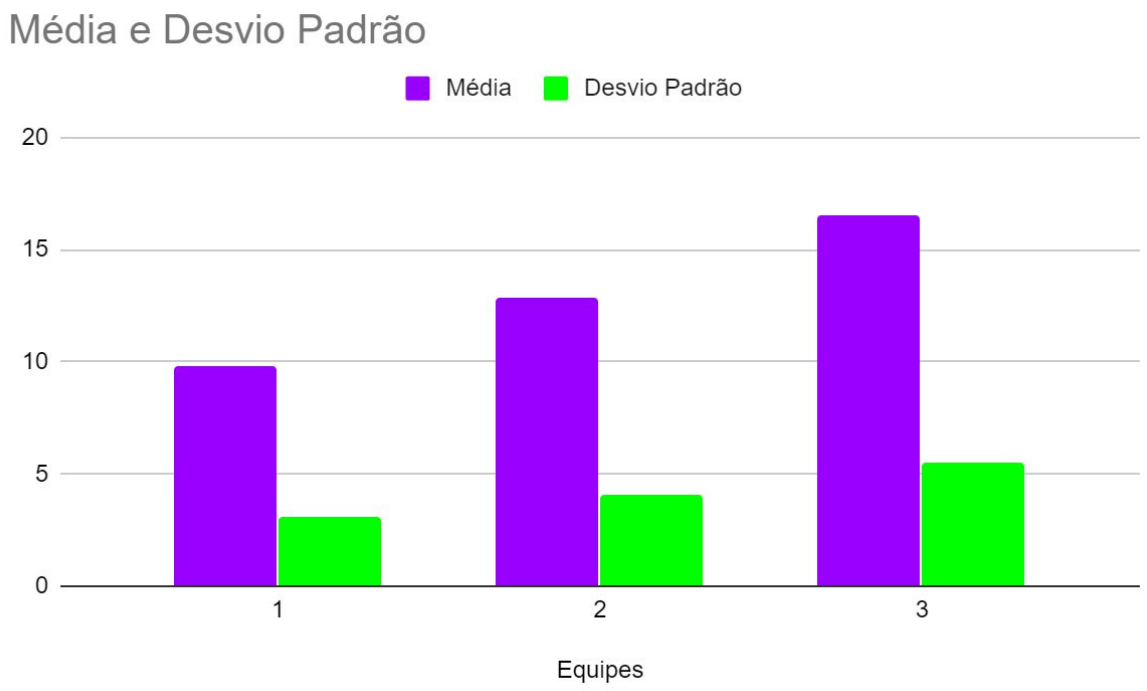
S 2 7	S N	Não realiza troca pois gera ciclo no grafo.
M	M 16 6 5 4 3 2 1 11 12 13 14 15 10 9 8 7	Ordem de fala permanece igual.

Resultado

Número	1	2	3	4	5	6	7	8	9	10
Tempo de execução	14	14	16	25	10	15	10	24	14	23

Média	**Desvio Padrão**
16,5	5,543

Análise gráfica



De acordo com os dados dispostos no gráfico, é possível perceber que mesmo após executar diferentes cenários de teste, onde o número de arestas (m) e o número de vértices (n) foi dobrado em cada caso, o tempo de execução do programa não acompanhou o crescimento da variação das entradas, mantendo

assim uma tendência linear de crescimento. Portanto, pôde-se observar que o sistema desenvolvido se comportou assintoticamente de forma linear com complexidade da ordem de O(m + n).

Questões

- **Por que o grafo tem que ser dirigido?**

  O grafo precisa ser dirigido pois a relação hierárquica de comando estabelecida entre os estudantes não se dá por meio de duplo sentido, ou seja, um comandante de uma equipe não pode ser comandado por seu subordinado. Dessa forma, é necessário que o grafo seja dirigido para indicar qual aluno comanda quem.

- **O grafo pode ter ciclos?**

  Não, o grafo não pode conter ciclos, pois dessa forma seria possível gerar uma relação onde um subordinado poderia comandar seu comandante, o que, como já mencionado, não se trata de uma configuração válida na hierarquia das equipes.

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

  O grafo do problema em questão pode sim ser uma árvore, porém, ele não necessariamente corresponde a uma, visto que, por ser dirigido, é possível que o grafo possua um número de arestas superior a (n - 1), ao mesmo tempo que se mantém acíclico. Nesse caso, o grafo deixa de atender a uma das propriedades de uma árvore, e portanto não é considerado como tal.

Conclusão

Conclui-se então que, foi possível alcançar o objetivo inicial do trabalho, ao realizar o desenvolvimento de um sistema que suprisse o problema de equipes de alunos, atendendo a complexidade de O(m + n).

Além disso, os conceitos aprendidos durante a parte teórica da disciplina, tais como os conhecimentos relacionados à grafos, métodos de busca (BFS e DFS) e

ordenação topológica, foram de extrema importância e serviram como auxílio na construção do trabalho.

Dessa forma, foi possível aliar a teoria à prática, a fim de reforçar e consolidar os conhecimentos referentes a disciplina de Algoritmos 1.

Referências

- Techiedeligh. Graph Implementation in C++ (without using STL). Disponível em:<https://www.techiedelight.com/graph-implementation-c-without-using-stl/>. Acesso em: 24 de Setembro de 2019.
- GeeksForGeeks. Transpose graph. Disponível em: <https://www.geeksforgeeks.org/transpose-graph/>. Acesso em: 24 de Setembro de 2019.
- GeeksForGeeks. Depth First Search or DFS for a Graph. Disponível em: <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>. Acesso em: 24 de Setembro de 2019.
- GeeksForGeeks. Breadth First Search or BFS for a Graph. Disponível em: <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>. Acesso em: 24 de Setembro de 2019.
- Pastebin. Código de Tempo de Execução de um programa. Disponível em: <https://pastebin.com/YwNNVxZ8/>. Acesso em: 24 de Setembro de 2019.