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

DOCUMENTAÇÃO BLACKJACK

Paula Mara Ribeiro

Belo Horizonte,
2019

Problema

O sistema é concebido a partir de uma equipe de BlackJack da UFMG. Nele, é necessário cadastrar os integrantes do time e suas relações hierárquicas com os demais alunos. Não é permitido que um aluno comande o seu comandante (formando assim um ciclo hierárquico) e também não é permitido que um aluno não seja comandante nem comandado de nenhum outro no esquema.

A partir da definição da equipe, o sistema oferece opções de funcionalidades para a equipe. É oferecida a opção de encontrar o líder mais novo do aluno especificado, também permite encontrar uma possível ordem de fala nas reuniões seguindo a estrutura hierárquica, e por fim, é possível trocar dois alunos na hierarquia, desde que essa troca não ocasione em um ciclo hierárquico.

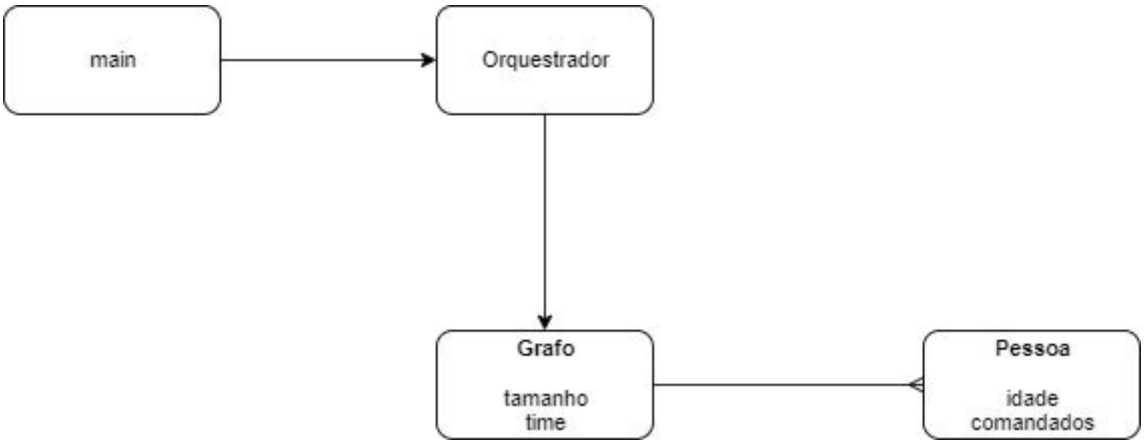
Solução

Para solucionar o problema foram utilizadas como base duas classes: Grafo e Pessoa. A classe Grafo contém o tamanho e os componentes do time, a partir de um vetor de objetos Pessoa. Foi escolhido o vetor simples da linguagem uma vez que o número de elementos é pré-especificado na entrada do programa e permanece constante durante a execução, não necessitando de mais manipulações. Além disso, por ser indexado, nos permite realizar o acesso em O(1).

Já a classe Pessoa é apenas uma estrutura de dados responsável por armazenar a idade do integrante e possui um vetor com o índice dos seus comandados. Nesse caso, foi escolhido o tipo *vector* da linguagem por fornecer funções de adicionar e retirar, que são úteis já que o tamanho do vetor é variável durante a execução e a ordem dos seus elementos não é importante. Assim, foi implementado um grafo direcionado para representar a estrutura hierárquica do time.

A classe Orquestrador é responsável por realizar a lógica das funcionalidades oferecidas, de forma a segregar as responsabilidades do Grafo - operar sobre a estrutura; e das operações - realizar funções lógicas sobre a estrutura do Grafo. Assim, enquanto o grafo é responsável por funções como adicionar e remover adjacências (relacionamentos), verificar se há ciclo, criar a pilha da ordem topológica e criar um grafo invertido de si mesmo, o orquestrador utiliza essas funções para realizar o *swap*, o *commander* e o *meeting.*

Por fim, a classe *main* realiza as operações de entrada, como receber e ler o arquivo, iniciar as pessoas e montar o grafo com os relacionamentos e também controlar qual das funcionalidades deve ser chamada.



Análise de Complexidade

Classe Pessoa.cpp

Análise de Tempo

A classe pessoa possui três funções: construtor, destrutor e comanda. O construtor e destrutor são operações lineares, portanto O(1). Já a função comanda itera sobre todos os seus comandados para verificar se determinado aluno está na lista. Sendo assim, essa função é O(grau(v)), sendo *v* a instância de pessoa.

Análise de Espaço

A classe pessoa armazena para cada uma de suas instâncias um atributo de idade e um vetor com grau(v) comandados, sendo também O(grau(v)).

Classe Grafo.cpp

Análise de Tempo

A classe grafo possui três funções lineares: o construtor, destrutor e o adicionarAdjacencia. Este último pode ser justificado pelo fato de que o novo relacionamento é sempre inserido no final do vetor, sem precisar iterar sobre ele.

O método removerAdjacencia é O(grau(u)), sendo *u* aluno que comanda, pois é preciso iterar sobre todos os alunos que ele comanda até encontrar o aluno a ser retirado. No pior caso, esse aluno estará na última posição ou não existirá, tendo que passar por todos os relacionamentos.

O método inverter é O(m+n). Para inverter o grafo, é preciso passar por todos os elementos dele e, para cada elemento, deve passar por cada um dos seus relacionamentos

(grau(u) iterações para cada). Por ser um grafo direcionado, ele terá a informação de cada relacionamento apenas uma vez, totalizando m iterações. Essas m iterações chamam uma função linear (adicionarAdjacencia), por isso o custo é O(m). Como esse método inicializa o grafo invertido, ele tem o custo inicial de n, totalizando O(m+n).

O método verificarCiclo é O(m+n). Para sua implementação, foi utilizado uma variação da pesquisa em profundidade, assim, para cada elemento do grafo realiza-se uma pesquisa em profundidade, para verificar se este em algum momento retorna para o elemento inicial. Porém, como é verificado se aquele aluno já foi processado, apenas é realizada a pesquisa em profundidade para aquele aluno uma vez. Esse método utiliza o método auxiliar verificarRecorrenciaVerticesAdjacentes, que itera sobre todos comandados do aluno na iteração apenas se ele já não tiver sido verificado.

O método inicializarPilhaTopologica também é O(m+n), já que ele passa por cada aluno apenas uma vez, verificando cada um de seus comandados a partir da função inserirPilhaTopologica, e o insere numa pilha auxiliar.

Análise de Espaço

O grafo ocupa um espaço de O(m+n), pois ele registra todos os elementos (tamanho n) e todos os relacionamentos (tamanho m). Adicionalmente, ele registra também o tamanho dele mesmo, que é linear.

Em seus métodos, é comum utilizar estruturas de dados auxiliares do seu tamanho, chegando assim a gastar O(m+3n), porém, assintoticamente seu custo continua como O(m+n).

Classe Orquestrador.cpp

Análise de Tempo

A função swap realiza duas vezes a função comanda da classe Pessoa, sendo O(grau(u) + grau(v)), no pior caso sendo O(m). A função auxiliar realizar troca chama a função removerAdjacencia, que é O(grau(u)) e adicionarAdjacencia, O(1). Ela também chama duas vezes a função inicializarVisitados, que é O(n). A operação mais custosa é a verificarCiclo, que é O(m+n). Assim, assintoticamente seu custo é O(m+n).

A função meeting realiza uma vez a função inicializarVisitados, O(n), uma vez a inicilizarPilhaTopologica, O(m+n) e uma vez a imprimirPilha, O(n), pois itera uma vez sobre cada elemento do grafo. Assim, assintoticamente seu custo também é O(m+n).

A função commander realiza uma vez a função inicializarVisitados e uma vez um vetor auxiliar de pessoas, ambos O(n). Chama também a função de inverterGrafo, O(m+n) e então faz uma pesquisa em largura para encontrar o líder mais novo do aluno especificado, passando uma vez sobre cada aluno, tendo um custo de O(m+n). Assim, assintoticamente seu custo é O(m+n).

Análise de Espaço

A classe Orquestrador não possui atributos próprios e compartilha dos atributos da classe Grafo. Assim, seu custo também é de O(m+n).

Classe main.cpp

Análise de Tempo e Espaço

A classe main apenas realiza as operações de entrada e saída e chama as funções conforme necessário. Como todas as funções analisadas convergem para O(m+n), a complexidade da main e, por consequência, do sistema em geral, também é O(m+n).

Avaliação Experimental

Questões Teóricas

• Por que o grafo tem que ser dirigido?

O grafo tem que ser dirigido pois o relacionamento “comanda” não é recíproco. Um aluno A comanda o outro B, porém a recíproca não é verdadeira.

• O grafo pode ter ciclos?

Não, devido a singularidade do relacionamento “comanda”. Conceitualmente, um aluno não pode comandar àquele que o comanda.

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

Sim, o grafo pode ser uma árvore, representando uma estrutura hierárquica rígida, em que uma pessoa é comandada apenas por um superior, mas não necessariamente ele vai ser uma árvore, pois uma pessoa pode ser comandada por mais de um superior, como se fossem de dois “departamentos” diferentes. Até por isso, podem haver duas raízes distintas, ou seja, dois alunos que não são comandados por ninguém porém que comandam outras pessoas.

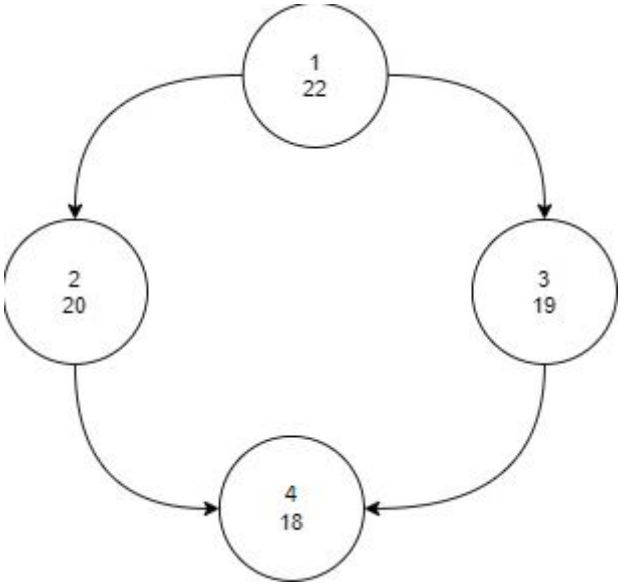
Avaliação

Estão aqui os diagramas e resultados para cada uma das equipes. Em anexo estão também os arquivos geradores utilizados para obter os resultados. Para cada caso de teste, foram executados: quatro commander, dois swaps e três meetings.

Para a medição do tempo, foi utilizado o código fornecido que divide o tempo gasto pela quantidade de clocks por segundo.

Equipe 1

N = 4
M = 4



Comandos e Resultados

C 4	19	O líder mais novo do 4 é o 3
C 3	22	O único líder de 3 é o 1
C 2	22	O único líder de 2 é o 1
C 1	*	1 não possui líder
M	1 3 2 4	Ordem topológica
S 1 3	T	A troca é permitida
M	3 1 2 4	Nova ordem topológica com 3 como raiz
S 4 3	N	Ocasiona em ciclo
M	3 1 2 4	Mesma ordem anterior

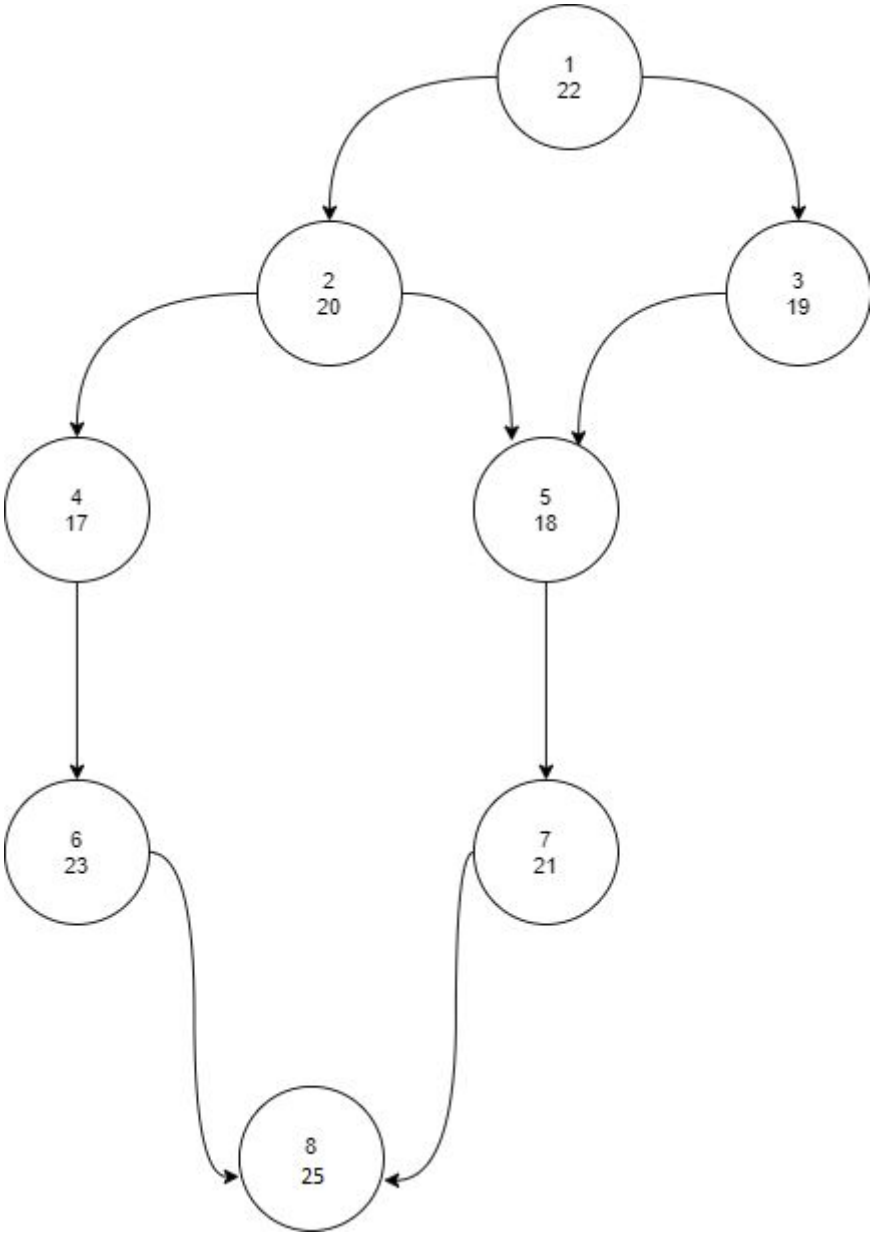
Tempo de Execução

1	2	3	4	5	6	7	8	9	10
5	12	11	11	12	11	12	11	11	10

Média: 10,6
Desvio Padrão: 1,9596

Equipe 2

N = 8
M = 9



Comandos e Resultados

C 8	17	O líder mais novo de 8 é o 4
C 7	18	O líder mais novo de 7 é o 5
C 1	*	1 não tem líder
C 6	17	O líder mais novo de 6 é o 4
M	1 3 2 5 7 4 6 8	Ordem topológica
S 2 5	T	Troca permitida
M	1 3 5 7 2 4 6 8	Nova ordem topológica com 5 como líder do 2
S 2 1	N	Ocasiona em ciclo
M	1 3 5 2 4 6 7 8	Outra ordem topológica possível com 5 como líder do 2

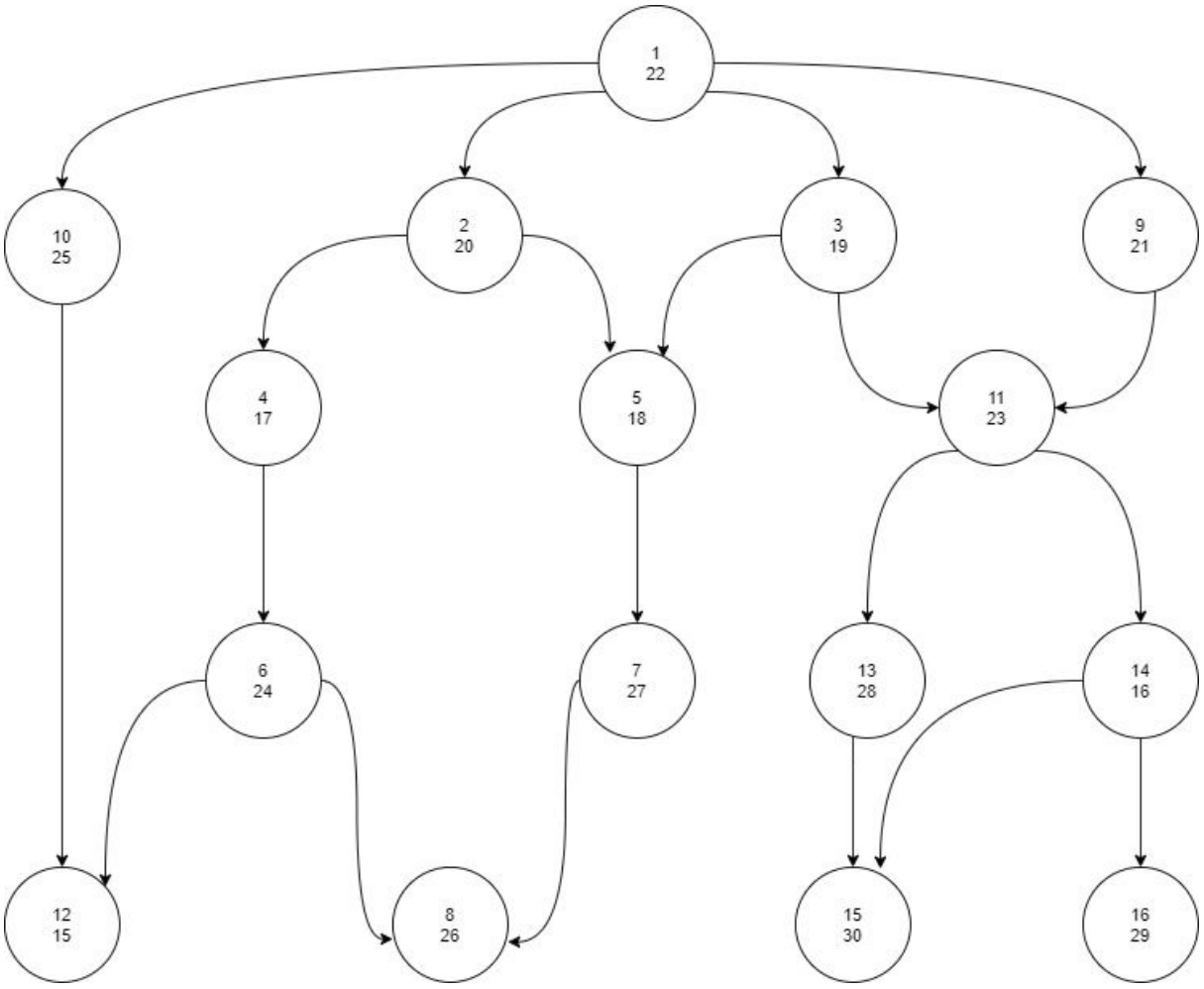
Tempo de Execução

1	2	3	4	5	6	7	8	9	10
12	15	14	17	17	17	14	17	16	11

Média: 15
Desvio Padrão: 2,0976

Equipe 3

N = 16
M = 20



Comandos e Resultados

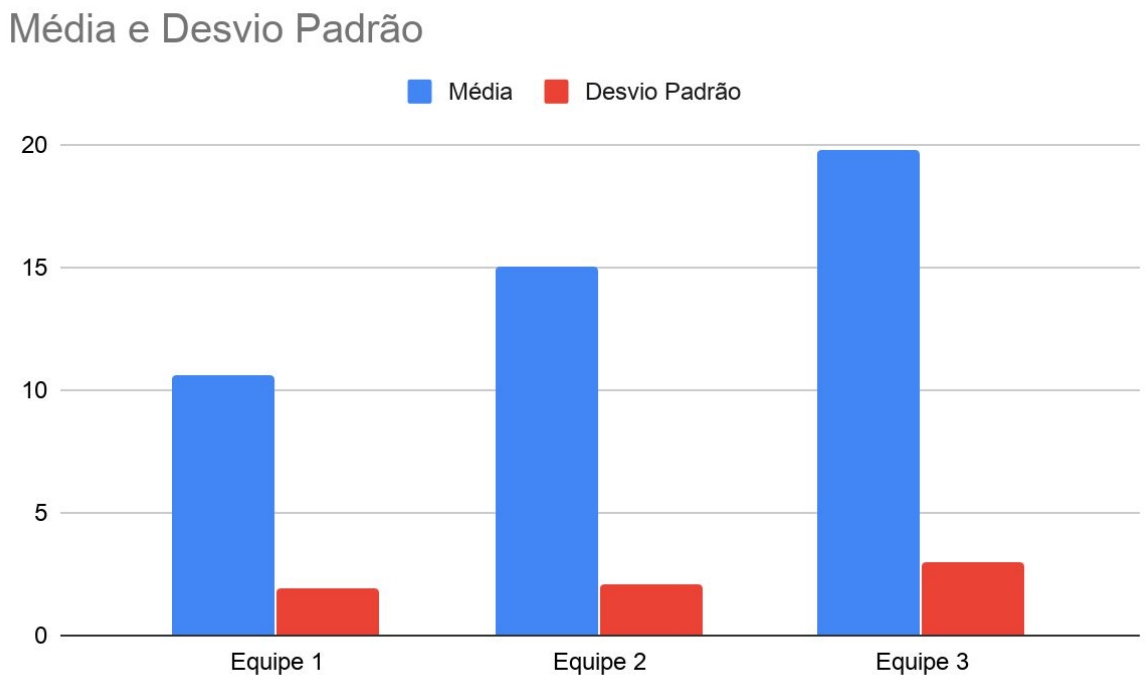
C 15	16	O líder mais novo do 15 é o 14
C 16	16	O líder mais novo do 16 é o 14
C 7	18	O líder mais novo do 7 é o 5
M	1 9 10 3 11 14 16 13 15 2 5 7 4 6 8 12	Ordem topológica
S 16 14	T	Troca permitida
C 16	*	16 passa a não ter nenhum líder
M	16 1 9 10 3 11 14 13 15 2 5 7 4 6 8 12	Nova ordem topológica com 16 como líder do 14
S 11 14	T	Troca permitida
M	16 14 1 9 10 3 11 13 15 2 5 7 4 6 8 12	Nova ordem topológica com o 14 como líder do 11

Tempo de Execução

1	2	3	4	5	6	7	8	9	10
21	18	28	18	19	18	19	20	20	17

Média: 19,8
Desvio Padrão: 2,9597

Análise



Para cada um dos casos de teste, o número de participantes foi dobrado, assim como o número de relacionamentos. Porém, a média do tempo de execução não dobrou junto com as entradas, demonstrando sua propriedade linear de complexidade. Podemos também perceber que, de forma assintótica, as execuções seguiram o tempo O(m+n).

Conclusão

Podemos concluir que a implementação proposta está seguindo a análise de complexidade feita como O(m+n). O conhecimento de técnicas de implementação em grafos como a busca em largura e a busca em profundidade favoreceu o desenvolvimento do trabalho.

Referências

Breadth First Search or BFS for a Graph - GeeksforGeeks. GeeksforGeeks. Disponível em: <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>. Acesso em: 17 set. 2019.

Código de Medição de Tempo. Pastebin. Disponível em: <https://pastebin.com/YwNNVxZ8>. Acesso em: 22 set. 2019.

Depth First Search or DFS for a Graph - GeeksforGeeks. GeeksforGeeks. Disponível em: <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>. Acesso em: 15 set. 2019.