Trabalho Prático 1
Algoritmos I

Gabriela Peres Neme (2018054745)

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil
gabrielaperesneme@gmail.com

1. Descrição do problema

O presente trabalho consiste em realizar uma série de operações sobre um grupo de alunos organizados hierarquicamente. Essa hierarquia é definida de forma que se A comanda B, então B não comanda A (direta ou indiretamente).

A fim de modelar o problema, foi utilizado um grafo *G(N, M)* direcionado não ponderado em que os N alunos representam vértices e as M hierarquias diretas entre dois alunos A e B são representadas por uma aresta entre eles.

Dado esse grupo de alunos, as operações a serem feitas no presente trabalho, com ordem de complexidade de tempo de $O(N + M)$, são:

- SWAP: Essa instrução recebe como entrada os alunos A e B, e objetiva trocar a ordem de comando entre esses alunos. Ou seja, se existir uma aresta de A para B, ou de B para A e a inversão na ordem de comando não gerar um ciclo no grafo, a hierarquia entre esses dois alunos deve ser trocada, ou seja, se A comanda B, B comandará A, e vice-versa.
- COMMANDER: Essa instrução recebe como entrada um aluno A e retorna a idade da pessoa mais jovem que comanda A (direta ou indiretamente).
- MEETING: essa instrução identifica uma possível ordem de fala entre os alunos, em uma reunião, respeitando a hierarquia de que, se A comanda B, direta ou indiretamente, A deve falar antes que B.

2. Implementação

2.1. Entrada de dados

A entrada de dados é feita por um arquivo de texto (*.txt) que deve ser passado como parâmetro na chamada do programa, como no exemplo abaixo, em que os dados da equipe e as instruções desejadas estão no arquivo “arquivo_entrada.txt”.

```
./tp1 arquivo_entrada.txt
```

Esse arquivo de entrada deve conter, na primeira linha, três inteiros separados por espaço, N, M e I, indicando respectivamente o número de pessoas no time, o número de

relações de hierarquia direta entre os membros e o número de instruções. Os N alunos são representados através de números de 1 a N.

Na segunda linha do arquivo devem existir N números, separados por espaço, que indicam a idade dos N membros da equipe, em ordem. Nas próximas M linhas existirão dois números inteiros X e Y que indicam membros da equipe. Se X aparece antes de Y, significa que X comanda diretamente Y.

As próximas I linhas contêm, cada uma, uma instrução. As instruções do tipo *Swap* iniciam com um S seguido de dois inteiros, A e B, que representam os alunos que se deseja inverter a hierarquia. As instruções do tipo *Commander* são seguidas de um inteiro, indicando um aluno. Por fim, as instruções do tipo *Meeting* consistem em uma linha contendo apenas a letra M.

2.2. Saída de dados

Os resultados das instruções são impressos na saída padrão, na mesma ordem que elas foram especificadas no arquivo de entrada.

Para cada instrução *Swap* o programa imprime S T caso a troca tenha sido um sucesso e S N caso contrário. Para cada instrução *Commander* o programa imprime C X, em que X é um inteiro correspondente à idade da pessoa mais jovem que comanda o aluno especificado na entrada da instrução. Caso o aluno não seja comandado por ninguém, será impresso C *.

Por fim, para a instrução *Meeting* será impressa a letra M seguida de uma ordem de fala dessa equipe.

2.3. Estrutura de dados

Para representar o grafo que contém a hierarquia da equipe utilizou-se uma lista de adjacência, através de um vetor de listas.

O vetor possui tamanho N (número de membros da equipe). Para cada membro i no vetor, existe uma lista contendo seus comandantes diretos, que também pode ser vazia, conforme exemplo abaixo.

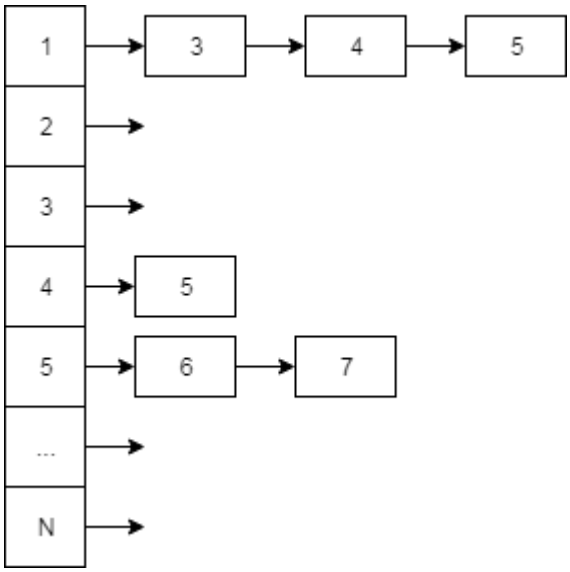


Figura 1 - Estrutura de dados da hierarquia da equipe

Para representar as instruções foi criada uma classe abstrata *Instrucao*, que define métodos que devem estar presentes nas classes que a herdarem.

Para cada um dos três tipos de instrução foi criada uma classe que herda *Instrucao*. Elas se chamam *Swap*, *Commander* e *Meeting*.

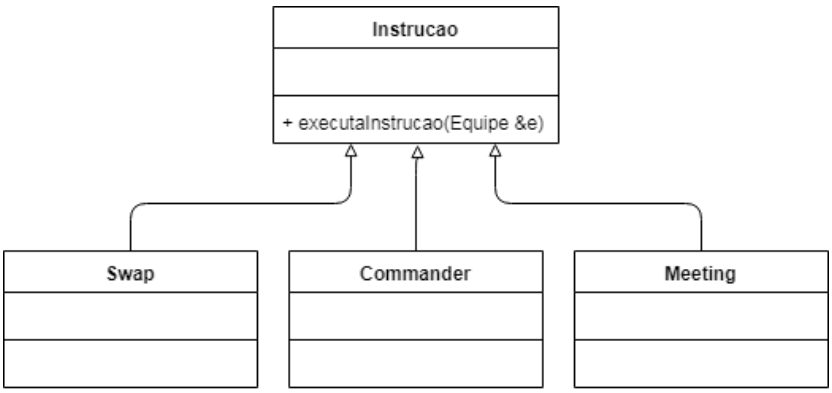


Figura 2 - Diagrama de classes das instruções

Desta forma, a fim de armazenar as instruções à medida que são lidas na entrada de dados, criou-se uma lista de instâncias de *Instrucao*, que se comporta como uma fila e utiliza do polimorfismo desse conjunto de classes para armazenar e, posteriormente, executar as instruções.

Os vetores e listas utilizados foram, respectivamente, as estruturas de dados *vector* e *list*, presentes na biblioteca padrão de C++.

2.4. Algoritmo para resolver *Swap* e complexidade

A fim de resolver o problema *Swap* foi necessário inverter as arestas relativas à hierarquia que se deseja trocar (se existentes) e, então, verificar a existência de um ciclo no grafo.

Para checar se existe aresta entre os alunos A e B, verificou-se a lista de adjacência do aluno A, procurando por B, e a lista de B, procurando A.

Para verificar se o grafo possui um ciclo utilizou-se uma busca em profundidade (DFS) que checa se foi atingido um vértice cuja busca foi iniciada, mas ainda não encerrada, ou seja, um vértice que ainda se está explorando seus descendentes. Esses nós estão marcados como “Visitando”, no presente algoritmo, o que significa o mesmo que vértices “cinza”, conforme nomenclatura utilizada no material da presente disciplina.

Os métodos utilizados para realizar a DFS com checagem de ciclo, na classe Equipe, foram *possuiCiclo* e *procuraCiclo*. O primeiro chama o segundo para cada aluno ainda não visitado, e o segundo funciona recursivamente, chamando a si mesmo para cada comandante do aluno que o método recebeu como parâmetro. Ambos retornam *true* caso um ciclo seja encontrado.

A complexidade de tempo deste algoritmo é $O(N + M)$, em que N representa o número de alunos e M o número de comandos diretos entre eles (arestas do grafo). Pode-se afirmar isso porque a busca pela aresta possui complexidade $O(M)$, pois percorre no máximo todas as arestas do grafo. O algoritmo de checagem de ciclo, por sua vez, é uma DFS, e esse algoritmo percorre todos os vértices uma vez, e também todas as arestas uma vez, pois vértices já visitados são marcados e não serão visitados novamente, o que resulta na complexidade $O(N + M)$.

$$O(N + M) + O(M) = O(N + M)$$

A complexidade de espaço desse algoritmo também é *O(N + M)*, pois o vetor possui tamanho N (número de vértices), armazenando cada um dos N alunos, e as listas de cada um dos vértices conterão um elemento para cada aresta, ou seja, um inteiro e um ponteiro para cada hierarquia direta (M). As idades dos alunos também foram salvas em um vetor de tamanho N. Durante a execução do algoritmo a única alteração de dados seria a inversão de uma aresta, que remove um item da lista encadeada, mas adiciona outro, o que mantém a ordem de complexidade de espaço. Ademais, durante a execução recursiva da DFS, a equipe é passada como referência e, por isso, não será duplicada na memória.

2.5. Algoritmo para resolver *Commander* e complexidade

O algoritmo utilizado para executar a instrução *Commander* consiste em uma DFS executada apenas a partir do aluno X para o qual se deseja descobrir seu chefe mais jovem.

Assim, ao contrário de uma DFS tradicional, que chama a função recursiva para cada um dos n nós ainda não visitados, o presente algoritmo chama a função recursiva apenas para o aluno X.

Nessa DFS, para cada nó visitado, ou seja, para cada aluno em hierarquia superior ao atual, é salva a menor idade da hierarquia superior, comparativamente à idade do aluno atual. Assim, recursivamente é obtida a menor idade de toda a hierarquia superior do aluno X.

Como o presente grafo é composto de arestas que vão de quem é comandado para quem comanda, não é necessário invertê-las ou duplicar a estrutura da equipe para executar o algoritmo acima especificado.

Essas tarefas são desempenhadas no código através dos métodos presentes na classe Equipe, *chefeMaisNovo* e *menorIdade*. O primeiro chama o segundo para o aluno X, e o segundo chama a si mesmo, recursivamente, para todos os comandantes de X, salvando a menor idade encontrada, conforme acima especificado.

A ordem de complexidade de tempo desse algoritmo também é *O(N + M)*, pois é executada uma DFS, em que podem ser percorridos todos os nós (mas não necessariamente), e podem ser percorridas todas as arestas (também não necessariamente). Ademais, o número de comparação de idade feito é igual ao número de arestas percorridas.

Pelos mesmos motivos do tópico anterior, a ordem de complexidade de espaço desse algoritmo é *O(N+M)*. Nesse comando, entretanto, nenhuma mudança é feita na estrutura de dados da equipe.

2.6. Algoritmo para resolver *Meeting* e complexidade

A fim de identificar uma possível ordem de fala dos alunos, respeitando a hierarquia, utilizou-se uma DFS para criar uma ordenação topológica para o grafo. Como se optou por construir o grafo com arestas de quem é comandado para quem comanda, não foi necessário inverter a impressão dos vértices gerados na ordem topológica, pois os vértices com menor tempo de término são os alunos com maior nível hierárquico (pois não possuem chefes), e são os primeiros a serem impressos.

Nesse sentido, executou-se uma DFS tradicional, alterada apenas pela contabilização do tempo de término e consequente adição do vértice recém-terminado a uma fila de

impressão. Em outras palavras, a cada vez que um vértice termina a busca de todos os seus superiores, feita de forma recursiva, ele é adicionado à fila de impressão.

Como elucidado anteriormente, o resultado será a ordem topológica inversa do grafo modelado, que corresponde, assim, à ordem de fala dos estudantes.

Os métodos da classe Equipe utilizados para executar esse algoritmo foram *ordemDeFala* e *momentoFinalizacao*. O primeiro chama o segundo para cada aluno ainda não visitado do grafo. O segundo método, por sua vez, chama a si mesmo para cada um dos comandantes diretos do aluno passado a ele como parâmetro e, ao finalizar o processo recursivo, adiciona esse aluno à fila de ordem de fala (resultando na ordem topológica do grafo).

A complexidade desse algoritmo também é *O(N+M)*, pois trata-se de uma DFS tradicional em que a única diferença é que cada vértice será adicionado à lista de impressão quando tiver percorrido todos os seus comandantes, direta e indiretamente através da recursão. Como a DFS possui complexidade *O(N+M)*, e nada foi alterado em relação à forma de percorrer o grafo, confirma-se a complexidade previamente especificada.

Por fim, a complexidade de espaço também é *O(N+M)*, conforme explicado no item 2.4. Complementa-se que, nesse algoritmo, os dados do grafo não são alterados, mas cria-se um novo array de tamanho N, cuja complexidade de espaço é *O(N)*. Desta forma, a complexidade de *O(N+M)* é mantida.

3. Avaliação experimental

3.1. Parâmetros

Para a avaliação experimental foram utilizadas equipes de 4, 8, 12, 16, 20, 30, 40, 50, 60, 70, 80, 90 e 100 alunos. Para cada equipe definiu-se, arbitrariamente, N*(N-2)/4 arestas, ou seja, N*(N-2)/4 chefias diretas.

Além disso, os arquivos de entrada possuem uma instrução de cada tipo, independentemente do tamanho da equipe, resultando em uma instrução *Swap*, uma instrução *Commander* e uma instrução *Meeting*, nessa ordem.

Assegurou-se que nenhum dos grafos da entrada possuíssem ciclos e, por fim, foram realizadas 100 execuções do algoritmo para cada uma das equipes, e registrou-se todos os tempos encontrados.

3.2. Resultados

Por terem sido realizadas 1300 medições de tempo (13 equipes, e 100 execuções cada), optou-se por não explicitar todos os valores encontrados, individualmente. Para eventuais consultas, esses foram apresentados na pasta *resultados*.

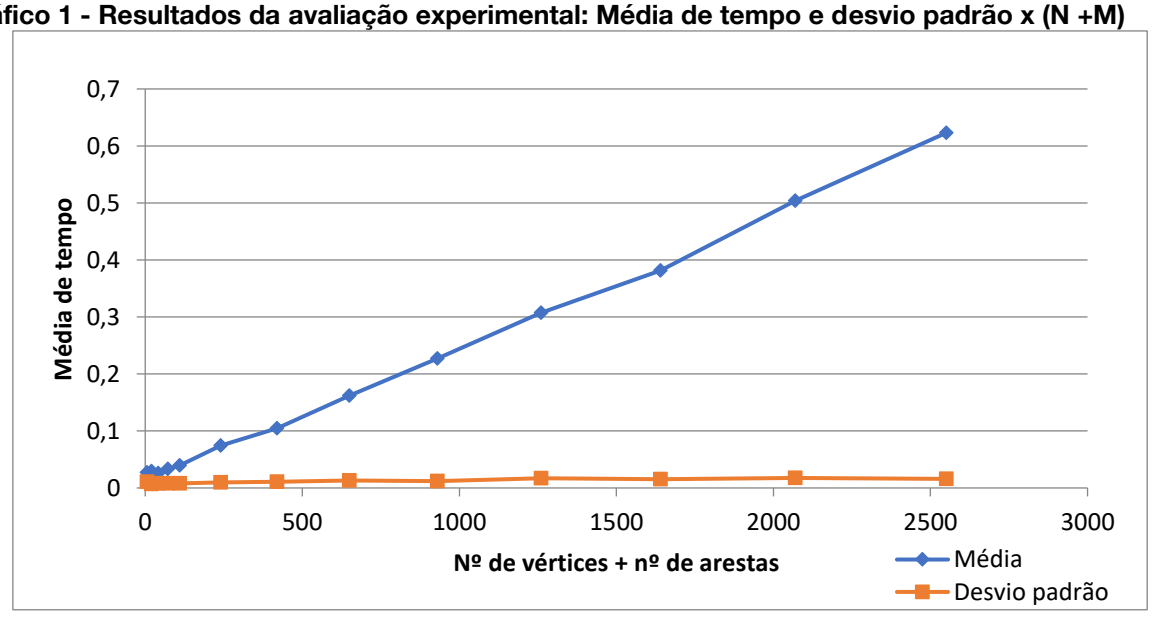
À vista disso, a tabela abaixo mostra a média encontrada e o desvio padrão do tempo de execução para cada uma das equipes.

Tabela 1 - Resultados da análise experimental

N	M = N*(N-2)/4	N+M	Média de tempo	Desvio padrão de tempo
4	2	6	0,02707	0,010732
8	12	20	0,02932	0,007126
12	30	42	0,02612	0,007984
16	56	72	0,03346	0,008302
20	90	110	0,03961	0,007959
30	210	240	0,07417	0,00987
40	380	420	0,10452	0,010762
50	600	650	0,16165	0,012983
60	870	930	0,22683	0,011969
70	1190	1260	0,30759	0,017238
80	1560	1640	0,38147	0,015104
90	1980	2070	0,50412	0,017507
100	2450	2550	0,62298	0,015839

A fim de facilitar a visualização dos valores, elaborou-se o gráfico seguinte, que representa o tempo médio e o desvio padrão em função do número de vértices somado do número de arestas.

Gráfico 1 - Resultados da avaliação experimental: Média de tempo e desvio padrão x (N +M)



Como pode ser observado no Gráfico 1, o crescimento da média do tempo de execução é linear em relação à soma N+M, o que é esperado a partir das análises de complexidade acima feitas, já que as três instruções têm ordem de complexidade de tempo de O(N + M).

Em relação ao desvio padrão, foi possível observar que ele é praticamente constante, e aumenta muito pouco à medida que o número de alunos na equipe cresce. Isso significa que grupos menores de alunos são mais influenciados pela variação do tempo, o que explica a menor linearidade no início do gráfico, para (N+M) < 100.

Desta forma, foi possível corroborar, com dados experimentais, o comportamento assintótico previsto para os algoritmos apresentados, igual a O(N + M).

3.3. Respostas às perguntas

3.3.1. Por que o grafo tem que ser dirigido?

O grafo deve ser dirigido porque a relação de hierarquia não é simétrica, ou seja, A ser chefe de B não é o mesmo que B ser chefe de A. Por isso, a direção da hierarquia deve ser informada através da direção das arestas que compõem, assim, um grafo dirigido.

3.3.2. O grafo pode ter ciclos?

O grafo em questão não pode ter ciclos, pois uma exigência da especificação do problema é que, para quaisquer alunos A e B do grafo, se A for chefe de B, então B não pode ser chefe de A, direta ou indiretamente. Caso houvesse qualquer ciclo no grafo essa regra seria quebrada e, por isso, não podem haver ciclos.

Além disso, a existência de uma ordem de fala conforme as regras estabelecidas exige a existência de uma ordem topológica, o que é mutuamente excludente com a existência de um ciclo.

3.3.3. O grafo pode ser uma árvore? O grafo necessariamente é uma árvore?

O grafo pode ser uma árvore, mas não necessariamente é uma árvore. Ele pode ser uma floresta ou pode conter nós que tenham mais de um pai (diretamente). Em ambos os casos, o grafo não seria uma árvore.

4. Conclusão

O presente trabalho teve como objetivo aplicar uma série de instruções a um grafo dirigido não ponderado acíclico, que representa a hierarquia de uma equipe de alunos. As instruções consistem em: *Swap*, que inverte a hierarquia de dois alunos; *Commander*, que retorna a idade da pessoa mais jovem que comanda determinado aluno; e *Meeting*, que identifica uma ordem de fala todos os alunos, de forma que se um aluno A comanda um aluno B, A deve falar antes de B.

Os algoritmos desenvolvidos para executar tais instruções possuem ordem de complexidade de tempo de $O(N + M)$, em que N é o número de alunos na equipe e M é o número de relações diretas de hierarquia na equipe (arestas do grafo). Tal complexidade foi corroborada pela avaliação experimental, que demonstrou tendência de crescimento linear do tempo médio de execução em função de N + M.

A complexidade de espaço obtida também é *O(N + M)*, pois a estrutura de dados escolhida para representar o grafo foi uma lista de adjacência, composta por um vetor de listas.

Por fim, os objetivos do trabalho foram atingidos com sucesso, uma vez que todos os comandos funcionaram na complexidade requerida.

5. Referências

Almeida, J., (2019), Graph Transversal, In Algoritmos I, Departamento de Ciência da Computação, UFMG.

Kleinberg J., Tardos E., (2005), Algorithm design, 1ª edição, Pearson Education, Inc.

Rosen K., (2009), Matemática Discreta e Suas Aplicações, 6ª edição, AMGH Editora Ltda.

Ziviani, N. (2011), Projeto de Algoritmos com Implementações em Pascal e C, Cengage Learning, 3ª Edição.