Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
DCC059 – Teoria dos Grafos – Turma A
Semestre 2024/1 - Profª Luciana Brugiolo
Data entrega: 04/06

ESPECIFICAÇÃO DO TRABALHO PRÁTICO 1

Funcionalidade básicas em um grafo simples (10 pontos)

Este documento traz a especificação do Trabalho 1, atividade avaliativa do tipo "Trabalho Prático", e serve de base para todas as atividades de implementação cobradas ao longo do curso da disciplina DCC059 - Teoria dos Grafos no semestre letivo 2024-1.

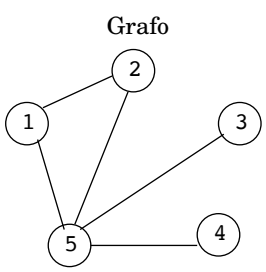
Desenvolver um Tipo Abstrato de Dados - TAD ou uma Classe que represente grafos simples, que podem ser:

- Orientados ou não orientados,
- Ponderados ou não ponderados (nos vértices e/ou nas arestas)

Implemente o conjunto de funcionalidades apresentados a seguir, detalhadas em sala de aula. O desenvolvimento dos algoritmos contribui como base para a resolução das questões das avaliações teóricas.

Orientações:

- Seu TAD ou Classe deve ser capaz de representar grafos utilizando **LISTA DE ADJACÊNCIA**;
- O código deve ser desenvolvido em linguagem C ou C++ e você não deve usar funções que não sejam nativas da linguagem, pois isso pode levar à impossibilidade de compilar seu código no ambiente que será usado para testes;
- Além do atendimento às funcionalidades, alguns dos elementos avaliados são a clareza e a organização do código (nomes de funções e variáveis, comentários que indiquem o propósito das principais funções e procedimentos, inclusive explicando o que são os parâmetros e o retorno, em caso de função);
- O programa principal que usará o TAD ou a Classe Grafo deve ler os dados do grafo de entrada (direcionados ou não direcionados, ponderados ou não ponderados) a partir de arquivo texto. O formato do arquivo dependerá da origem dos dados de entrada. Assim, cabe ao grupo ler o arquivo README que explica a semântica do arquivo de entrada ou, caso não haja este arquivo, o grupo deve ler o detalhamento do mesmo na fonte de dados, implementando conforme o caso. Entenda-se por formato do arquivo a estrutura em que os dados do grafo aparecem no texto. Por exemplo, para algumas instâncias teste, o arquivo pode ser apresentado como segue no exemplo, onde se tem um grafo simples, não ponderado nos vértices e nas arestas, e não direcionado. Neste exemplo, a primeira linha indica o número de vértices e as demais linhas indicam as arestas.



Arquivo de entrada

```
5
1  2
2  5
5  3
4  5
1  5
```

- a informação sobre o tipo de grafo, se direcionado ou não direcionado, deve ser passada ao programa por parâmetro via linha de comando, sendo 0 (zero) para não direcionado e 1 (um) para grafos direcionados.
- a informação sobre arestas ponderadas ou não deve ser passada ao programa por parâmetro via linha de comando, sendo 0 (zero) para não ponderado nas arestas e 1 (um) para grafos com peso nas arestas.
- a informação sobre a existência de pesos nos vértices deve ser passada ao programa por parâmetro via linha de comando, sendo 0 (zero) quando o grafo não é ponderado nos vértices e 1 (um) para grafos com pesos nos vértices.
- o nome do arquivo a ser lido deve ser informado ao programa via teclado para a função main (utilizar `int main (int argc, char **argv)` para passar ao programa todas as informações necessárias ao seu funcionamento;
- cada grupo enviará um único trabalho contendo **APENAS** os arquivos fonte (extensão c, cc, cpp e h) e os arquivos de entrada utilizados (caso estes não tenham sido disponibilizados);
- o padrão para compilação a ser utilizado (ambiente Linux ou IOS) será g++ *.c* -o execGrupoX. Onde "GrupoX" indica a qual grupo o trabalho se refere;
- o padrão para a execução a ser utilizado pelo professor será a linha abaixo, executada em ambiente Linux ou IOS:

```
./execGrupoX <arquivo_entrada> <arquivo_saida> <Op_Direc> <Op_PesoAresta> <Op_PesoNos>
```

Onde:

`<arquivo_entrada>` é o nome do arquivo que contém as informações do grafo,

`<arquivo_saida>` é o arquivo onde será gravado o grafo armazenado na memória ao término da execução do programa;

`<Op_Direc> <Op_PesoAresta> <Op_PesoNos>` são os parâmetros referentes às características do grafo.

- O grupo deve enviar um **ÚNICO** arquivo compactado cujo nome deve ser Trabalho_GrupoX.**zip**.

Nota: no arquivo zip deve ter incluso **APENAS** os arquivos com extensão c, cc, cpp ou h e os arquivos das instâncias usadas no experimento.

Funcionalidades: O programa deve apresentar em **tela** a saída para as seguintes:

a) **Parâmetro**: um Id de um vértice de um **grafo direcionado**;
   Saída: o fecho transitivo direto deste vértice.

b) **Parâmetro**: um Id de um vértice de um **grafo direcionado**;
   Saída: o fecho transitivo indireto deste vértice.

c) **Parâmetro**: dois IDs de vértices do grafo;
Saída: o caminho mínimo entre estes dois vértices usando algoritmo de Djkstra;

d) **Parâmetro**: dois IDs de vértices do grafo;
Saída: o caminho mínimo entre estes dois vértices usando algoritmo de Floyd;

e) **Parâmetro**: um subconjunto X de vértices do grafo;
Saída: uma Árvore Geradora Mínima sobre o subgrafo vértice-induzido por X usando o algoritmo de Prim;

f) **Parâmetro**: um subconjunto X de vértices do grafo;
Saída: uma Árvore Geradora Mínima sobre o subgrafo vértice-induzido por X usando o algoritmo de Kruskal;

g) **Parâmetro**: um ID de vértice;
Saída: a árvore dada pela ordem de caminhamento em profundidade a partir de nó dado parâmetro, destacando as arestas de retorno;

h) **Parâmetro**: o grafo (direcionado ou não direcionado) ponderado nas arestas
Saída: O raio, o diâmetro, o centro e a periferia do grafo.

i) **Parâmetro**: o grafo não direcionado
Saída: O conjunto de vértices de articulação.

- Cada uma das saídas das funcionalidades acima deve ser apresentada na tela. Ao final da sua execução e, em seguida, deve-se perguntar ao usuário se o mesmo deseja salvar a saída em arquivo.
- O programa deve apresentar as funcionalidades na forma de um menu de opções que se repete até que o usuário escolha sair do programa.

Perguntas Frequentes

1. *Quantos membros um grupo pode ter?*
   O trabalho pode ter no **máximo 4 pessoas.** Mas, caso haja interesse de fazer o trabalho individualmente, você deve estar ciente de que, uma vez informado ao professor, não poderá integrar um outro grupo.
2. *O projeto poderá ter mais de um arquivo fonte (c, cc, cpp e h)?*
   Pode (e, para boa organização do código, deve). Como usual, a especificação do trabalho descreve somente a interface a ser implementada. A organização do projeto é livre.
3. *O que será levado em conta na correção?*
   Na correção do código serão levados em conta (entre outros) os seguintes elementos.
   1. Conformidade com a especificação.
   2. Correção da implementação.
   3. **Eficiência** da implementação.
   4. Interação com o professor/tutor;
   5. Organização e clareza do código (nomes de funções e variáveis, comentários etc).

4. *Por que a especificação de como o programa será executado é importante?*
   Porque o trabalho entregue será processado por um *script* que considera que a especificação de entrega foi corretamente observada.
5. *O que acontece se a especificação de execução do programa não for corretamente observada?*
   Seu trabalho só será corrigido quando houver tempo de fazer manualmente. **Por isso, haverá desconto na sua nota, proporcional ao trabalho de processamento que tenha que ser feito manualmente**.
6. *Meu trabalho tem um bug. O que vai acontecer com minha nota?*
   Você deve informar isso no relatório e haverá algum desconto na nota, dependendo da gravidade que o *bug* implicar no funcionamento. Se o problema afetar alguma das funcionalidades requeridas, o desconto será proporcional ao que estiver faltando.
7. *Meu código não compila. Posso enviar assim mesmo?*
   Não serão avaliados trabalhos com erros de compilação. Por isso a importância de se usar apenas funções do padrão Ansi.
8. *Tenho outra pergunta/dúvida a respeito do trabalho.*
   Procure o professor ou o tutor para tirar suas dúvidas no horário de atendimento ou durante as aulas.