Departamento de Ciência da Computação - UFJF
DCC059 - Teoria dos Grafos
Prof. Gabriel Souza

Da Entrega

Deve ser enviado um arquivo zip para o email gabriel.souza@ufjf.br até dia 04/01/2025 com os seguintes arquivos:

- arquivos cpp, c, h ou hpp (hpp só será permitido quando for usado templates)
- link do projeto do github onde o código por desenvolvido
- apresentação uníca no formato pdf com o conteúdo que será apresentado por todos os membros do grupo
- documentação do código no formato pdf ou html/css/js
- nome e matricula dos intengrates do grupo e indicação de quais partes foram feitas por cada um no formato pdf (devendo estar de acordo com os commits presentes no github)

Da apresentação

A apresentação será feita em sala de aula no esquema de elevator pitch onde cada aluno terá exatamente 1 minuto para apresentar as atividades que ele executou no projeto. Após isso, terá um periodo de perguntas por parte do professor.

Não será necessário que todos os alunos assistam todas as apresentações, mas todas serão livres para quem quiser assistir. O horário de apresentação de cada grupo dentro do periodo da aula será disponibilizado posteriormente.

Da avaliação

O trabalho será avaliado em 100 pontos e depois será ponderado conforme seu valor na nota final da disciplina.

O calculo da nota será dado por: $\sqrt{T * \sqrt{A * P}}$ onde A, T e P valem 100 pontos e representam:

- A - Apresentação
- T - Avaliação Geral do trabalho
- P - Participação do aluno no desenvolvimento do código

Departamento de Ciência da Computação - UFJF
DCC059 - Teoria dos Grafos
Prof. Gabriel Souza

Do código

Implementar uma classe abstrada “grafo” em C++ com 2 classes filhas para diferentes estruturas de armazenamento para nós e arestas:

- grafo_matriz - uso de matriz de adjacencia para representar arestas (quando grafo não direcionado deve ser usado a representação linear de matriz triangular). Além disso, os vertices e arestas devem ser estáticos.
- grafo_lista - uso de lista encadeada tanto para vertices quanto arestas usando alocação dinamica.

Funções necessárias para a classe abstrata:

- eh_bipartido - função força bruta que indica se o grafo é bipartido ou não
- n_conexo - função que indica a quantidade de componentes conexas
- get_grau - função que retorna o grau do grafo
- get_ordem - função que retorna a ordem do grafo
- eh_direcionado - função que retorna se o grafo é direcionado ou não
- vertice_ponderado - função que informa se os vertices do grafo tem peso
- aresta_ponderada - função que informa se as arestas do grafo tem peso
- eh_completo - função que diz se um grafo é completo ou não
- eh_arvore - função que diz se o grafo é uma árvore
- possui_articulacao - função que diz se existe ao menos um vertice de articulação
- possui_ponte - função que diz se existe ao menos uma aresta ponte
- carrega_grafo - função que lê um arquivo txt com um grafo e carrega ele
- novo_grafo - função que lê um arquivo txt de configuracao e gera um grafo aleatorio

Observações:

- Outras funções podem (e devem) ser inclusas para melhor estruturação do código seguindo os preceitos de Orientação a Objetos.
- Apenas as funções de acesso as estruturas de vertices e arestas devem estar nas classes filhas. Como por exemplo as funções get_vizinhos, get_arestas...
- Será necessário implementar classes de lista encadeada de forma separada.
- Deve ser respeitado os conceitos de herança e encapsulamento durante o desenvolvimento do código.
- O código precisa estar devidamente documentado.
- A execução do código será feita usando o Valgrind.
- Bibliotecas permitidas: fstream, iostream, iomanip, cmath, cstdlib, cstdarg, ctime, string
- Os grafos não devem aceitar arestas multiplas ou laços
- Não será necessário adição e remoção de nós e arestas nessa parte do trabalho

Departamento de Ciência da Computação - UFJF
DCC059 - Teoria dos Grafos
Prof. Gabriel Souza

Da execução

Após compilado, o código deve ser executado via terminal com as seguintes linhas de comando:

Caso 1: main.out -d -m grafo.txt

- Imprime descrição do grafo apos carregá-lo com matriz de adjacencia como no exemplo abaixo:

grafo.txt

Grau: 3
Ordem: 3
Direcionado: Sim
Componentes conexas: 1
Vertices ponderados: Sim
Arestas ponderadas: Sim
Completo: Sim
Bipartido: Não
Arvore: Não
Aresta Ponte: Não
Vertice de Articulação: Não

Caso 2: main.out -d -l grafo.txt

- Mesma coisa que o caso 1, mas carregando o grafo com lista encadeada.

Caso 3: main.out -c -m descricao.txt grafo.txt

- Recebe um arquivo de descrição, cria um grafo aleatório na estrutura de matriz seguindo as propriedades definidas no arquivo, e salva o grafo no arquivo grafo.txt

Caso 4: main.out -c -l descricao.txt grafo.txt

- Recebe um arquivo de descrição, cria um grafo aleatório na estrutura de lista seguindo as propriedades definidas no arquivo, e salva o grafo no arquivo grafo.txt

Departamento de Ciência da Computação - UFJF
DCC059 - Teoria dos Grafos
Prof. Gabriel Souza

Dos arquivos

Grafo.txt

```
3 1 1 1 // numero de nos, direcionado, ponderado vertices, ponderado arestas
2 3 7 // peso dos nos (apenas se ponderado nos vertices)
1 2 6 // origem, destino, peso (peso apenas se ponderado na aresta)
2 1 4 // origem, destino, peso (peso apenas se ponderado na aresta)
2 3 -5 // origem, destino, peso (peso apenas se ponderado na aresta)
…
```

Obs.: o id dos vertices deve começar em 1, o arquivo não deve conter os comentarios do exemplo acima, eles são apenas descritivos para esse documento

Descricao.txt

```
3 // Grau
3 // Ordem
1 // Direcionado
2 // Componentes conexas
1 // Vertices ponderados
1 // Arestas ponderadas
0 // Completo
1 // Bipartido
0 // Arvore
1 // Aresta Ponte
1 // Vertice de Articulação
```

Obs.: o arquivo não deve conter os comentarios do exemplo acima, eles são apenas descritivos para esse documento