

Trabalho de Teoria dos Grafos

Integrantes do grupo:

Matheus Leal Costa - 202176013

Carlos Gustavo Ferreira Rezende – 202065503B

Gabriel De Oliveira Vieira - 202265029AC

Maria Luiza Dornelas Corrêa - 201665194C

1. Introdução

Este trabalho tem como objetivo a implementação de estruturas de dados para representação e manipulação de grafos em C++. O projeto abrange a representação de grafos tanto por listas de adjacência quanto por matrizes de adjacência, permitindo operações como inserção e remoção de nós e arestas, verificação de propriedades do grafo e cálculo de menores distâncias.

2. Estrutura do Projeto

```
TrabalhoGrafosGrupoX.zip
|
| docs/
| | documentacao.pdf
| | apresentacao.pdf
| | contribuidores.pdf    (o que cada membro fez, junto com o link do github)
| | descricao.pdf        (este documento)
|
| include/
| | *.h
| | *.hpp
| | *.tpp
|
| src/
| | *.c
| | *.cpp
|
| entradas/
| | *.txt
|
| main.cpp
|
| README    (explicação simples e considerações sobre a execução do programa)
```

- **docs/**: Contém documentos de apoio, como a documentação técnica, apresentação e contribuições dos membros do grupo.
- **include/**: Arquivos de cabeçalho (.h, .hpp) e templates (.tpp).
- **src/**: Implementação do código-fonte em C++.
- **entradas/**: Arquivos de entrada utilizados para testar o programa.
- **main.cpp**: Arquivo principal de execução do programa.
- **README**: Guia rápido de uso do programa.

3. Descrição das Classes e Funções

3.1. Grafo

Classe base abstrata que define as operações fundamentais para grafos. As funções marcadas como virtuais devem ser implementadas pelas classes derivadas (GrafoMatriz e GrafoLista).

- **virtual void novo_no(int id, float peso):** Adiciona um novo nó.
- **virtual void deleta_no(int id):** Remove um nó.
- **virtual void nova_aresta(int origem, int destino, float peso):** Adiciona uma nova aresta.
- **virtual void deleta_aresta(int origem, int destino):** Remove uma aresta.
- **virtual float getPesoAresta(int origem, int destino):** Retorna o peso de uma aresta.
- **virtual bool existeNo(int id):** Verifica se um nó existe.
- **virtual bool existeAresta(int origem, int destino):** Verifica se uma aresta existe.
- **virtual int getGrau(int id):** Retorna o grau de um nó.
- **virtual int* getVizinhos(int id):** Retorna os vizinhos de um nó.
- **virtual int getNumArestas():** Retorna o número total de arestas do grafo.
- **void carrega_grafo(const std::string& arquivo):** Carrega um grafo a partir de um arquivo.
- **int get_ordem():** Retorna a ordem do grafo (número de nós).
- **bool eh_direcionado():** Verifica se o grafo é direcionado.
- **bool vertice_ponderado():** Verifica se os vértices são ponderados.
- **bool aresta_ponderada():** Verifica se as arestas são ponderadas.
- **int get_grau():** Retorna o grau máximo do grafo.
- **bool eh_completo():** Verifica se o grafo é completo.
- **int n_conexo():** Retorna o número de componentes conexas do grafo.
- **void DFS(int no, bool* visitado):** Realiza uma busca em profundidade.
- **void calculaMenorDistancia():** Calcula as menores distâncias usando o algoritmo de Floyd-Warshall.
- **void deleta_primeira_aresta(int id):** Deleta a primeira aresta de um nó.

- **void imprimeGrafo():** Exibe informações do grafo.
- **virtual void imprimeLista():** Imprime a lista de adjacência do grafo.

3.2. GrafoLista

Implementação do grafo usando listas de adjacência. Esta classe implementa todos os métodos virtuais da classe base Grafo.

- **void novo_no(int id, float peso) override:** Adiciona um novo nó.
- **void deleta_no(int id) override:** Remove um nó.
- **void nova_aresta(int origem, int destino, float peso) override:** Adiciona uma nova aresta.
- **void deleta_aresta(int origem, int destino) override:** Remove uma aresta.
- **bool existeNo(int id) override:** Verifica se o nó existe.
- **bool existeAresta(int origem, int destino) override:** Verifica se a aresta existe.
- **int getGrau(int id) override:** Retorna o grau de um vértice.
- **int* getVizinhos(int id) override:** Retorna os vizinhos de um nó.
- **int getNumArestas() override:** Retorna o número total de arestas.
- **float getPesoAresta(int origem, int destino) override:** Retorna o peso da aresta.
- **void imprimeLista() override:** Imprime a lista de adjacência do grafo.

3.3. GrafoMatriz

Implementação do grafo usando matrizes de adjacência. Também implementa todos os métodos virtuais da classe base Grafo.

- **void novo_no(int id, float peso) override:** Adiciona um novo nó.
- **void deleta_no(int id) override:** Remove um nó.
- **void nova_aresta(int origem, int destino, float peso) override:** Adiciona uma nova aresta.
- **void deleta_aresta(int origem, int destino) override:** Remove uma aresta.
- **bool existeAresta(int origem, int destino) override:** Verifica se uma aresta existe.
- **float getPesoAresta(int origem, int destino) override:** Retorna o peso da aresta.
- **int getGrau(int id) override:** Retorna o grau de um nó.

- **int* getVizinhos(int id) override:** Retorna os vizinhos de um nó.
- **int getNumArestas() override:** Retorna o número total de arestas.
- **void imprimeMatriz() override:** Imprime a matriz de adjacência do grafo.

3.4. ListaEncadeada

Suporte para a representação de listas de nós e arestas:

- **No* getNo(int id):** Retorna o nó correspondente.
- **Aresta* getAresta(int origem, int destino):** Retorna a aresta entre dois nós.
- **void setCabeca(No* novoCabeca):** Define o nó cabeça da lista.
- **No* getCabeca():** Retorna o nó cabeça da lista.
- **void imprimeLista():** Imprime a lista de adjacência do grafo.

4. Compilação e Execução

4.1. Compilação

Para compilar o programa, utilize o seguinte comando:

```
g++ -std=c++11 -I./include -o main main.cpp ./src/grafo.cpp
./src/lista_encadeada.cpp ./src/grafo_lista.cpp ./src/grafo_matriz.cpp
```

4.2. Execução

O programa pode ser executado com o seguinte comando:

```
./main -d -m|-l ./entradas/arquivo_entrada.txt
```

- **-d:** Comando para executar o modo de análise do grafo.
- **-m ou -l:** Define se o grafo será representado por matriz (-m) ou lista (-l).
- **arquivo_entrada.txt:** Arquivo de entrada contendo a descrição do grafo.

5. Exemplos de Entrada/Saída

Exemplo de Arquivo de Entrada (entradas/exemplo.txt):

```
5 0 1 1
10 20 30 40 50
1 2 5
1 3 3
2 3 4
```

2 5 9.6

2 4 2.588

3 5 7.32

3 4 9.1

4 5 1.5

Saída Esperada:

INFORMAÇÕES DO GRAFO:

Grau: 3

Ordem: 4

Direcionado: Não

Componentes conexas: 1

Vértices Ponderados: Sim

Arestas Ponderadas: Sim

Completo: Não

Maior menor distância: (1, 2) - 11.408

6. Conclusão

O trabalho permitiu a compreensão prática de conceitos de Teoria dos Grafos, como a representação de grafos por listas e matrizes de adjacência, além de algoritmos de busca, e, com as expansões feitas durante o trabalho 2, pudemos adicionar o cálculo de distâncias, e fazer a adição e remoção de nós e arestas.