Universidade Federal
de Juiz de Fora

TRABALHO DE GRAFOS - DOCUMENTAÇÃO

Álvaro Davi Santos - 202176037
João Vitor Pereira - 202176010

Introdução

Este trabalho tem como objetivo a implementação de uma classe abstrata de grafo em C++ com duas classes derivadas para representação de grafos em estruturas de lista encadeada e matriz de adjacência. O projeto também inclui funções para análise de propriedades do grafo, como bipartição, conexidade e completude.

Estrutura do Código

O código foi estruturado com base nos princípios de programação orientada a objetos (POO), garantindo modularidade, reutilização e encapsulamento. Ele consiste em uma classe base abstrata chamada Grafo e duas classes derivadas, Grafo_matrix e Grafo_lista, que representam grafos em diferentes estruturas de armazenamento.

A classe abstrata Grafo define a interface comum e implementa funcionalidades gerais aplicáveis a todos os grafos.
Já a classe Matrix utiliza uma matriz de adjacência para armazenar as arestas do grafo e a classe List utiliza uma lista encadeada para armazenar os nós e suas respectivas arestas.

Funções Implementadas

- 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 vértices 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 vértice 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 configuração e gera um grafo aleatório
- getMaiorCaminhoMinimo - função que retorna o maior dos menores caminhos entre os pares de vértices do grafo, utilizando o algoritmo de Floyd-Warshall.
- floydA0 - função que inicializa a matriz de distâncias para o algoritmo de Floyd-Warshall, preenchendo-a com os pesos das arestas do grafo.
- Floyd - função que implementa o algoritmo de Floyd-Warshall para encontrar os menores caminhos entre todos os pares de vértices.
- generateRandomGraph - função que gera um grafo aleatório com base nas configurações definidas, podendo incluir características como bipartição, conectividade, completude e presença de articulações ou pontes.

- existeAresta - função que verifica se há uma aresta entre dois vértices no grafo.
- ehPonte - função que verifica se a remoção de uma aresta desconectaria o grafo, indicando que ela é uma ponte.
- ehArticulacao - função que verifica se a remoção de um vértice altera a conectividade do grafo, indicando que ele é um vértice de articulação.
- verificaConexidade - função que verifica se o grafo é conexo, comparando o número de componentes conexas.
- auxGeraArvore - função auxiliar que gera uma árvore dentro do grafo, garantindo que ele atenda a restrições de grau e conectividade.

Execução

Exemplo de Entrada: Arquivo “grafo.txt”

Este arquivo contém a definição de um grafo, incluindo o número de nós, se o grafo é direcionado, se os vértices ou arestas são ponderados, bem como as arestas com seus respectivos pesos.
Exemplo:

```
3 1 1 1 // número de nós, direcionado, ponderado nos vértices, ponderado nas arestas
2 3 7 // pesos dos nós (se ponderado nos vértices)
1 2 6 // origem, destino, peso (se ponderado nas arestas)
2 1 42 3 -5
```

Exemplo de Entrada: Arquivo “descricao.txt”

Este arquivo descreve as propriedades do grafo que será gerado.
Exemplo:

```
3  // Grau
3  // Ordem
1  // Direcionado
2  // Componentes conexas
1  // Vértices ponderados
1  // Arestas ponderadas
0  // Completo
1  // Bipartido
0  // Árvore
1  // Aresta Ponte
1  // Vértice de Articulação
```

Os arquivos devem estar sem os comentários!

Exemplo de Saída
Quando o grafo é carregado e analisado, as propriedades calculadas são exibidas.
Exemplo de saída após execução:

```
Grau: 3
Ordem: 3
Direcionado: Sim
Componentes conexas: 1
Vértices ponderados: Sim
Arestas ponderadas: Sim
Completo: Sim
Bipartido: Não
Árvore: Não
Aresta Ponte: Não
Vértice de Articulação: Não
```

Comandos de Execução

Para carregar um grafo de matriz de adjacência:

```
main.out -d -m grafo.txt
```

Para carregar um grafo de lista encadeada:

```
main.out -d -l grafo.txt
```

Para gerar um grafo aleatório a partir de um arquivo de descrição:

```
main.out -c -m descricao.txt grafo.txt
```

ou

```
main.out -c -l descricao.txt grafo.txt
```

Os arquivos de entrada e saída devem estar nas pastas input e output respectivamente.