Documentação do Código - Trabalho 2 de Grafos

UNIVERSIDADE FEDERAL DE JUIZ DE FORA

Isabela Salvador Romão - MAT 202165065AB

Isabella Mourão dos Santos Dias - MAT 202165066AC

Mateus Alves da Silva - MAT 202076023

Professor: Gabriel Henrique de Souza

Fevereiro 11, 2025

1 - Introdução

Este documento descreve o código desenvolvido para o Trabalho 2 de Grafos, parte da disciplina Teoria dos Grafos (DCC059 - UFJF). O objetivo deste projeto foi aprimorar a implementação de grafos com matriz de adjacência e lista encadeada, adicionando funcionalidades, como:

- Manipulação dinâmica da estrutura do grafo (adição e remoção de nós e arestas).
- Alocação dinâmica da matriz de adjacência, garantindo escalabilidade.
- Recalculo de IDs ao remover nós na matriz, mantendo um grafo isomorfo.
- Cálculo da menor distância entre dois nós.

2 - Estrutura do Código

O projeto é organizado da seguinte forma:

- • /src: Contém a implementação dos algoritmos principais.
- • /tests: Inclui arquivos de teste que verificam o comportamento dos algoritmos.

3 - Descrição das Classes

Nesta seção, apresentamos os algoritmos implementados no projeto

Classe Abstrata Grafo

A classe Grafo é abstrata e define a interface base para a representação dos grafos. Seus principais métodos são:

- eh_bipartido() const: Verifica se o grafo é bipartido.
- n_conexo() const: Retorna o número de componentes conexas do grafo.
- get_grau(int vertice) const: Retorna o grau do vértice especificado.
- possui_ponte() const: Verifica se o grafo possui arestas de ponte.

- novo_grafo(const string& nomeArquivo): Cria um novo grafo a partir de um arquivo.
- carregar_grafo(const string& nomeArquivo) = 0: Carrega o grafo de um arquivo.
- imprimir_grafo(const string& nomeArquivo) const = 0: Imprime a representação do grafo em um arquivo.
- get_ordem() const: Retorna a ordem do grafo (número de vértices).
- eh_direcionado() const: Verifica se o grafo é direcionado.
- vertice_ponderado() const: Verifica se os vértices possuem peso.
- aresta_ponderada() const: Verifica se as arestas possuem peso.
- eh_completo() const = 0: Verifica se o grafo é completo.
- eh_arvore() const = 0: Verifica se o grafo é uma árvore.
- possui_articulacao() const = 0: Verifica se o grafo possui vértices de articulação.

Classe GrafoMatriz

Esta classe utiliza uma matriz de adjacência alocada dinamicamente. Inicialmente, a matriz tem tamanho 10x10 e duplica sua capacidade sempre que necessário. Além disso, ao excluir um nó, os IDs são recalculados para manter um grafo isomorfo.

Métodos implementados:

- novo_no(int id): Redimensiona a matriz caso necessário e adiciona o nó.
- deleta_no(int id): Remove o nó e recalcula os IDs.
- nova_aresta(int origem, int destino, float peso): Adiciona uma aresta.
- deleta_aresta(int origem, int destino): Remove uma aresta.
- menor_distancia(int origem, int destino): Aplica Floyd-Warshall para encontrar o caminho mínimo.

Classe GrafoLista

Nesta implementação, os vértices são armazenados em listas encadeadas, o que permite operações eficientes para adição e remoção de nós e arestas.

Métodos implementados:

- novo_no(int id): Insere um novo nó na lista.
- deleta_no(int id): Remove um nó e ajusta as conexões.
- nova_aresta(int origem, int destino, float peso): Adiciona uma aresta.
- deleta_aresta(int origem, int destino): Remove uma aresta.
- menor_distancia(int origem, int destino): Aplica Dijkstra para encontrar a menor distância.

4 - Detalhes de Implementação

- O código foi projetado para ser modular, garantindo facilidade de manutenção e escalabilidade.
- A alocação dinâmica da matriz evita desperdício de memória e permite grafos maiores.
- O reajuste de IDs na matriz mantém a coerência estrutural ao remover vértices.
- O cálculo da menor distância usa Floyd-Warshall (para matriz) e Dijkstra (para lista), garantindo eficiência e precisão.

5 - Conclusão

Este trabalho ampliou a nossa compreensão sobre estruturas de dados para grafos e algoritmos de busca e caminhos mínimos. A implementação dinâmica da matriz e a reformulação dos IDs foram importantes para nosso aprendizado sobre efieciencia de memória e manipulação de estruturas dinâmicas.