Documentação do Trabalho Grafos Parte 1
Versão 2

Samuel Paiva Bernardes

11 de Fevereiro de 2025

1 Introdução

Este documento descreve a **Parte 1 do Trabalho de Teoria dos Grafos**, que consiste na implementação de uma classe abstrata `Grafo` e duas classes derivadas (`GrafoMatriz` e `GrafoLista`) para representar grafos usando matriz de adjacência e lista de adjacência, respectivamente. O objetivo é fornecer uma estrutura modular e eficiente para manipulação de grafos, seguindo os princípios de orientação a objetos.

A Parte 1 do trabalho foi desenvolvida em C++ e inclui funcionalidades básicas para carregar grafos a partir de arquivos, verificar propriedades como conectividade, bipartição, completude, e realizar operações como busca em profundidade e cálculo de componentes conexos.

2 Estrutura do Projeto

O projeto está organizado da seguinte forma:

```
TrabalhoGrafosGrupoX/
 include/
    grafo.h
    grafo_matriz.h
    grafo_lista.h
    util.h
 src/
    grafo.cpp
    grafo_matriz.cpp
    grafo_lista.cpp
    util.cpp
    main.cpp
 entradas/
    grafo.txt
 README.md
```

2.1 Descrição dos Arquivos

- `grafo.h`: Define a classe abstrata `Grafo`, que contém métodos comuns a todos os tipos de grafos.
- `grafo_matriz.h` e `grafo_matriz.cpp`: Implementam a classe `GrafoMatriz`, que representa grafos usando matriz de adjacência.
- `grafo_lista.h` e `grafo_lista.cpp`: Implementam a classe `GrafoLista`, que representa grafos usando lista de adjacência.
- `util.h` e `util.cpp`: Contêm funções utilitárias, como leitura de arquivos e geração de grafos aleatórios.
- `main.cpp`: Contém a função principal que testa as funcionalidades implementadas.
- `grafo.txt`: Exemplo de arquivo de entrada para carregar um grafo.

3 Funcionalidades Implementadas

3.1 Classe Abstrata `Grafo`

A classe `Grafo` define métodos comuns a todos os tipos de grafos, como:

- `get_ordem()`: Retorna o número de vértices no grafo.
- `eh_direcionado()`: Verifica se o grafo é direcionado.
- `vertice_ponderado()`: Verifica se os vértices possuem pesos.
- `aresta_ponderada()`: Verifica se as arestas possuem pesos.
- `n_conexo()`: Retorna o número de componentes conexas.
- `eh_completo()`: Verifica se o grafo é completo.
- `eh_bipartido()`: Verifica se o grafo é bipartido.
- `eh_arvore()`: Verifica se o grafo é uma árvore.
- `possui_articulacao()`: Verifica se o grafo possui vértices de articulação.
- `possui_ponte()`: Verifica se o grafo possui arestas ponte.
- `carrega_grafo()`: Carrega um grafo a partir de um arquivo.

3.2 Classe `GrafoMatriz`

A classe `GrafoMatriz` implementa um grafo usando **matriz de adjacência**. As principais funcionalidades incluem:

- **Busca em Profundidade (DFS)**: Usada para verificar conectividade e componentes conexos.
- **Verificação de Bipartição**: Usa uma abordagem de coloração para verificar se o grafo é bipartido.
- **Verificação de Árvore**: Verifica se o grafo é uma árvore (conexo e sem ciclos).
- **Vértices de Articulação e Arestas Ponte**: Identifica pontos críticos no grafo.

3.3 Classe `GrafoLista`

A classe `GrafoLista` implementa um grafo usando **lista de adjacência**. As funcionalidades são semelhantes às da classe `GrafoMatriz`, mas com uma estrutura de dados diferente, mais eficiente para grafos esparsos.

3.4 Funções Utilitárias

As funções utilitárias incluem:

- `ler_arquivo()`: Lê um arquivo de texto e retorna suas linhas.
- `verificar_biparticao()`: Verifica se um grafo é bipartido.
- `gerar_grafo_aleatorio()`: Gera um grafo aleatório e o salva em um arquivo.

4 Como Rodar o Projeto

4.1 Compilação

Para compilar o projeto, use o seguinte comando no terminal:

```
g++ -o main src/main.cpp src/grafo.cpp src/grafo_matriz.cpp src/grafo_lista.cpp src/util.cpp
```

Isso gerará um executável chamado `main`.

4.2 Execução

O programa pode ser executado com os seguintes comandos:

4.2.1 Caso 1: Usando Matriz de Adjacência

```
./main -d -m entradas/grafo.txt
```

4.2.2 Caso 2: Usando Lista de Adjacência

```
./main -d -l entradas/grafo.txt
```

4.2.3 Explicação dos Argumentos

- `-d`: Indica que o grafo deve ser carregado a partir de um arquivo.
- `-m`: Usa a matriz de adjacência para representar o grafo.
- `-l`: Usa a lista de adjacência para representar o grafo.
- `entradas/grafo.txt`: Caminho do arquivo de entrada que contém o grafo.

4.3 Formato do Arquivo de Entrada

O arquivo de entrada (`grafo.txt`) deve seguir o seguinte formato:

```
3 1 1 1 // número de nós, direcionado, ponderado vértices, ponderado arestas
2 3 7   // peso dos nós (apenas se ponderado nos vértices)
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)
```

5 Explicação dos Métodos

5.1 `carrega_grafo()`

- **O que faz**: Carrega um grafo a partir de um arquivo de texto.
- **Exemplo de uso**:

```
GrafoMatriz grafo(0, false, false, false);
grafo.carrega_grafo("entradas/grafo.txt");
```

5.2 `n_conexo()`

- **O que faz**: Retorna o número de componentes conexas do grafo.
- **Exemplo de uso**:

```
int componentes = grafo.n_conexo();
std::cout << "Número de componentes conexas: " << componentes << std::endl;
```

5.3 `eh_bipartido()`

- **O que faz**: Verifica se o grafo é bipartido.
- **Exemplo de uso**:

```
if (grafo.eh_bipartido()) {
    std::cout << "O grafo é bipartido." << std::endl;
} else {
    std::cout << "O grafo não é bipartido." << std::endl;
}
```

5.4 `eh_arvore()`

- **O que faz**: Verifica se o grafo é uma árvore.
- **Exemplo de uso**:

```
if (grafo.eh_arvore()) {
    std::cout << "O grafo é uma árvore." << std::endl;
} else {
    std::cout << "O grafo não é uma árvore." << std::endl;
}
```

6 Considerações Finais

Este trabalho implementa uma estrutura robusta para manipulação de grafos, seguindo os princípios de orientação a objetos. A Parte 1 foi projetada para ser modular e extensível, permitindo a adição de novas funcionalidades na Parte 2.

Equipe de Desenvolvimento

- Samuel Paiva Bernardes

Informações Adicionais

- **Professor**: Gabriel Souza
- **Universidade**: UFJF