Documentação do Trabalho Grafos Parte 2

Samuel Paiva Bernardes

11 de Fevereiro de 2025

1 Introdução

Este documento descreve a **Parte 2 do Trabalho de Teoria dos Grafos**, que consiste na evolução do código da Parte 1, adicionando novas funcionalidades para manipulação de grafos. As principais adições incluem métodos para adicionar/remover nós e arestas, além de calcular a menor distância entre dois vértices usando o algoritmo de Dijkstra.

A Parte 2 foi desenvolvida em C++ e mantém a estrutura modular e orientada a objetos da Parte 1, com classes derivadas para representação de grafos por matriz de adjacência e lista de adjacência.

2 Estrutura do Projeto

O projeto mantém a mesma estrutura da Parte 1, com as seguintes adições e modificações:

```
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`: Atualizado com novos métodos para adição/remoção de nós e arestas, e cálculo da menor distância.
- `grafo_matriz.h` e `grafo_matriz.cpp`: Implementam as novas funcionalidades para grafos representados por matriz de adjacência.
- `grafo_lista.h` e `grafo_lista.cpp`: Implementam as novas funcionalidades para grafos representados por lista de adjacência.
- `main.cpp`: Atualizado para testar as novas funcionalidades.

3 Funcionalidades Implementadas

3.1 Novos Métodos na Classe Abstrata `Grafo`

A classe `Grafo` foi estendida com os seguintes métodos:

- `novo_no()`: Adiciona um novo nó ao grafo.
- `nova_aresta(int origem, int destino, int peso)`: Adiciona uma nova aresta entre dois vértices.
- `deleta_no(int id)`: Remove um nó do grafo.
- `deleta_aresta(int origem, int destino)`: Remove uma aresta entre dois vértices.
- `menor_distancia(int origem, int destino)`: Calcula a menor distância entre dois vértices usando o algoritmo de Dijkstra.

3.2 Classe `GrafoMatriz`

A classe `GrafoMatriz` implementa as novas funcionalidades para grafos representados por matriz de adjacência:

- **Adição de Nós**: Redimensiona a matriz de adjacência para acomodar novos nós.
- **Adição de Arestas**: Atualiza a matriz de adjacência com o peso da aresta.
- **Remoção de Nós**: Remove um nó e recalcula os IDs dos vértices.
- **Remoção de Arestas**: Remove uma aresta da matriz de adjacência.
- **Menor Distância**: Implementa o algoritmo de Dijkstra para calcular a menor distância entre dois vértices.

3.3 Classe `GrafoLista`

A classe `GrafoLista` implementa as novas funcionalidades para grafos representados por lista de adjacência:

- **Adição de Nós**: Adiciona um novo nó à lista de adjacência.
- **Adição de Arestas**: Adiciona uma nova aresta à lista de adjacência.
- **Remoção de Nós**: Remove um nó e atualiza as listas de adjacência dos vértices restantes.
- **Remoção de Arestas**: Remove uma aresta da lista de adjacência.
- **Menor Distância**: Implementa o algoritmo de Dijkstra para calcular a menor distância entre dois vértices.

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.

5 Explicação dos Métodos

5.1 `novo_no()`

- **O que faz**: Adiciona um novo nó ao grafo.
- **Exemplo de uso**:

```
GrafoMatriz grafo(0, false, false, false);
grafo.novo_no();
```

5.2 `nova_aresta()`

- **O que faz**: Adiciona uma nova aresta entre dois vértices.
- **Exemplo de uso**:

```
grafo.nova_aresta(1, 2, 5); // Adiciona aresta entre 1 e 2 com peso 5
```

5.3 `deleta_no()`

- **O que faz**: Remove um nó do grafo.
- **Exemplo de uso**:

```
grafo.deleta_no(2); // Remove o nó com ID 2
```

5.4 `deleta_aresta()`

- **O que faz**: Remove uma aresta entre dois vértices.
- **Exemplo de uso**:

```
grafo.deleta_aresta(1, 2); // Remove a aresta entre 1 e 2
```

5.5 `menor_distancia()`

- **O que faz**: Calcula a menor distância entre dois vértices.
- **Exemplo de uso**:

```
double distancia = grafo.menor_distancia(1, 2);
std::cout << "Menor distância entre 1 e 2: " << distancia << std::endl;
```

6 Considerações Finais

A Parte 2 do trabalho expande as funcionalidades da Parte 1, permitindo a manipulação dinâmica de grafos e o cálculo de distâncias mínimas. A estrutura modular e orientada a objetos facilita a extensão do código para novas funcionalidades.

Equipe de Desenvolvimento

- Samuel Paiva Bernardes

Informações Adicionais

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