Documentação do Trabalho de Grafos 02

DCC059

12 de fevereiro de 2025

Sumário

1 Introdução

Neste trabalho expandimos o projeto de Grafos (Trabalho 1) adicionando funcionalidades dinâmicas que permitem:

- Inserir novos nós e arestas durante a execução.
- Remover nós e arestas, com reindexação automática dos nós remanescentes de forma que o grafo resultante seja isomorfo ao original.
- Calcular a menor distância entre quaisquer dois nós (utilizando, por exemplo, o algoritmo de Floyd–Warshall).

O projeto foi implementado em duas versões de armazenamento:

1. **GrafoMatriz**: Utiliza uma matriz de adjacência alocada dinamicamente (capacidade inicial de 10 e realocada com o dobro do tamanho, se necessário).
2. **GrafoLista**: Utiliza uma lista encadeada para armazenar nós e suas arestas.

2 Objetivos

Os principais objetivos deste trabalho são:

- Adicionar as funções dinâmicas:
  - **novo_no**: Insere um novo nó na estrutura (matriz ou lista).
  - **nova_aresta**: Insere uma nova aresta.
  - **deleta_no**: Remove um nó e todas as arestas incidentes a ele, realizando a reindexação dos nós remanescentes.
  - **deleta_aresta**: Remove uma aresta específica.
- Na implementação por **matriz**:
  - A matriz de adjacência é alocada dinamicamente com capacidade inicial de 10 e, quando necessário, é realocada com o dobro do tamanho.
  - Ao remover um nó, a matriz é reconstruída excluindo a linha e a coluna correspondentes, e os IDs dos nós são recalculados (os nós remanescentes serão renumerados de 1 a N, onde N é o novo número de nós).
- Implementar uma função genérica, na classe base, para calcular a menor distância (ou, mais precisamente, o maior dos menores caminhos entre dois nós) utilizando os métodos virtuais que cada estrutura deve implementar.

3 Estrutura do Projeto

A estrutura dos arquivos está organizada da seguinte forma:

```
| include/
|    Grafo.hpp
|    GrafoMatriz.hpp
|    GrafoLista.hpp
|    IntList.hpp
|    ListaEncadeada.hpp
|    ListaEncadeada.tpp
|
| src/
|    Grafo.cpp
|    GrafoMatriz.cpp
|    GrafoLista.cpp
|    IntList.cpp
|
| entradas/
|    grafo.txt
|
| main.cpp
```

4 Descrição das Principais Classes

4.1 Classe `Grafo` (Base)

A classe abstrata `Grafo` define os atributos comuns e declara métodos virtuais que cada implementação deve fornecer:

- Atributos: `ordem`, `direcionado`, `ponderadoVertices`, `ponderadoArestas`.
- Funções de acesso: `get_ordem()`, `eh_direcionado()`, etc.
- Método `carrega_grafo`: Lê o arquivo de entrada e monta o grafo.
- Métodos virtuais puros:
  - `inserir_vertice(int id, int peso = 0)`
  - `inserir_aresta(int origem, int destino, int peso = 0)`
  - `getPesoAresta(int origem, int destino) const`
  - `get_vizinhos(int vertice) const`
- Função `calculaMaiorMenorDistancia`: Implementada de forma genérica (usando, por exemplo, o algoritmo de Floyd–Warshall) e que utiliza os métodos virtuais para obter os dados do grafo.

4.2 Classe `GrafoMatriz`

Nesta implementação o grafo é representado por:

- Uma matriz de adjacência dinâmica: alocada inicialmente com tamanho 10, e realocada com o dobro da capacidade quando necessário.

- Um vetor de pesos para os nós.
- Funções dinâmicas:
  - `novo_no`: Insere um novo nó na matriz.
  - `nova_aresta`: Insere uma nova aresta (atualizando a posição [i][j] e, para não direcionados, [j][i]).
  - `deleta_no`: Remove um nó – recria a matriz sem a linha e a coluna do nó removido e atualiza os contadores (nNos e ordem).
  - `deleta_aresta`: Zera o valor da aresta na matriz.

4.3 Classe `GrafoLista`

Nesta implementação o grafo é representado por:

- Uma lista encadeada de vértices, onde cada vértice possui uma lista encadeada de suas arestas.
- Para preservar a ordem de leitura (e facilitar a reindexação), os vértices são inseridos no final da lista (usando um método `append`).
- Funções dinâmicas:
  - `novo_no`: Insere um novo nó na lista.
  - `nova_aresta`: Insere uma nova aresta na lista de arestas de um vértice.
  - `deleta_no`: Remove o nó com o id especificado, remove as arestas incidentes em todos os nós, reatribui novos IDs sequencialmente e atualiza as referências nas arestas.
  - `deleta_aresta`: Remove uma aresta específica da lista de arestas.

5 Fluxo de Execução e Operações Dinâmicas

A execução do programa segue os seguintes passos:

1. **Carregamento do Grafo:** O método `carrega_grafo` lê o arquivo de entrada e monta o grafo usando `inserir_vertice` e `inserir_aresta`. Por exemplo, para o arquivo:

```
5 0 0 0
1 2
2 5
5 4
4 3
3 1
1 4
3 2
1 5
```

o grafo inicial deve ter ordem 5, com as arestas conforme especificado.

2. **Remoção Dinâmica:**
   - `deleta_no(1)`: Remove o nó com id 1, eliminando suas arestas e reindexando os nós remanescentes.
   - `deleta_aresta(2, X)`: Remove a "primeira aresta" do nó com id 2 (obtida via `get_vizinhos`).
3. **Reimpressão e Cálculos:** Após as remoções, o grafo é reimpresso e são calculadas as propriedades (grau, ordem, etc.) e a função `calculaMaiorMenorDistancia` é chamada para determinar o par de nós com a maior distância mínima.

6 Cálculo da Maior Menor Distância

A função `calculaMaiorMenorDistancia` calcula, para o grafo atual, o menor caminho entre todos os pares de nós conectados utilizando o algoritmo de Floyd–Warshall. Após computar todos os menores caminhos, a função identifica o par de nós cujo caminho mínimo é o maior dentre todos os pares (isto é, o diâmetro do grafo ou a "maior menor distância"). Caso não exista caminho entre um par de nós, esse par é ignorado (ou recebe um valor representado como infinito). O resultado é retornado em uma estrutura que contém os IDs dos dois nós que possuem a maior menor distância, bem como o valor dessa distância.. Essa função depende de:

- `getPesoAresta(int origem, int destino) const`: Deve retornar o peso da aresta se existir ou um valor grande (por exemplo, INF) se não houver conexão.

Dessa forma, tanto a versão **GrafoMatriz** quanto a **GrafoLista** herdarão essa funcionalidade e produzirão o mesmo resultado, desde que os métodos virtuais sejam implementados de maneira equivalente.

7 Depuração

Para facilitar a depuração:

- Durante as operações de remoção, mensagens de depuração podem ser inseridas para verificar a reindexação dos nós e a atualização das arestas.

8 Compilação e Execução

Para compilar o projeto, use:

```
clang++ -o main.out main.cpp src/*.cpp
```

E para executar:

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

9 Considerações Finais

Este trabalho amplia o projeto anterior, permitindo modificações dinâmicas no grafo (inserção e remoção de nós e arestas) e o cálculo de propriedades que dependem da estrutura atual. É fundamental que as operações de remoção e reindexação sejam implementadas de forma consistente em ambas as versões (matriz e lista) para que os resultados (por exemplo, o cálculo da maior menor distância) sejam os mesmos.