Trabalho de Grafos parte 2

Generated by Doxygen 1.13.2

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 ArestaEncadeada Class Reference

Representa uma aresta em um grafo utilizando uma estrutura de lista encadeada.

`#include <ArestaEncadeada.h>`

Public Member Functions

- ArestaEncadeada (VerticeEncadeado *origem, VerticeEncadeado *destino, float peso)
  Construtor da classe ArestaEncadeada.
- VerticeEncadeado * getOrigem () const
  Obtém o vértice de origem da aresta.
- VerticeEncadeado * getDestino () const
  Obtém o vértice de destino da aresta.
- float getPeso () const
  Obtém o peso da aresta.
- ArestaEncadeada * getProximo () const
  Obtém a próxima aresta na lista encadeada.
- void setProximo (ArestaEncadeada *novoProximo)
  Define a próxima aresta na lista encadeada.

Friends

- std::ostream & operator<< (std::ostream &os, const ArestaEncadeada &aresta)
  Sobrecarga do operador de saída para imprimir a aresta.

4.1.1 Detailed Description

Representa uma aresta em um grafo utilizando uma estrutura de lista encadeada.

Essa classe é utilizada na implementação do grafo baseado em listas de adjacência. Cada aresta contém referências ao vértice de origem e destino, além de um ponteiro para a próxima aresta na estrutura encadeada.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 ArestaEncadeada()

```
ArestaEncadeada::ArestaEncadeada (
            VerticeEncadeado * origem,
            VerticeEncadeado * destino,
            float peso)
```

Construtor da classe ArestaEncadeada.

Parameters

origem	Ponteiro para o vértice de origem.
destino	Ponteiro para o vértice de destino.
peso	Peso da aresta.

Inicializa uma aresta entre dois vértices com um determinado peso.

Parameters

origem	Ponteiro para o vértice de origem da aresta.
destino	Ponteiro para o vértice de destino da aresta.
peso	Peso da aresta.

4.1.3 Member Function Documentation

4.1.3.1 getDestino()

```
VerticeEncadeado * ArestaEncadeada::getDestino () const
```

Obtém o vértice de destino da aresta.

Returns

Ponteiro para o vértice de destino.

4.1.3.2 getOrigem()

```
VerticeEncadeado * ArestaEncadeada::getOrigem () const
```

Obtém o vértice de origem da aresta.

Returns

Ponteiro para o vértice de origem.

4.1.3.3 getPeso()

```
float ArestaEncadeada::getPeso () const
```

Obtém o peso da aresta.

Returns

Peso da aresta.

4.1.3.4 getProximo()

```
ArestaEncadeada * ArestaEncadeada::getProximo () const
```

Obtém a próxima aresta na lista encadeada.

Obtém o próximo elemento na lista de arestas encadeadas.

Returns

Ponteiro para a próxima aresta.

Ponteiro para a próxima aresta na lista encadeada.

4.1.3.5 setProximo()

```
void ArestaEncadeada::setProximo (
            ArestaEncadeada * novoProximo)
```

Define a próxima aresta na lista encadeada.

Parameters

novoProximo	Ponteiro para a nova aresta seguinte.
novoProximo	Ponteiro para a nova próxima aresta.

4.1.4 Friends And Related Symbol Documentation

4.1.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const ArestaEncadeada & aresta)  [friend]
```

Sobrecarga do operador de saída para imprimir a aresta.

Parameters

os	Stream de saída.
aresta	Aresta a ser impressa.

Returns

Stream de saída atualizado.

Exibe a origem, o destino e o peso da aresta.

Parameters

os	Fluxo de saída.
aresta	Aresta a ser exibida.

Returns

Fluxo de saída atualizado.

The documentation for this class was generated from the following files:

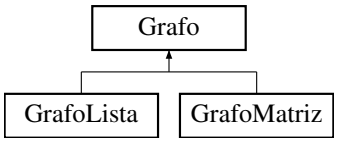
- include/ArestaEncadeada.h
- src/ArestaEncadeada.cpp

4.2 Grafo Class Reference

Classe abstrata que representa um grafo.

`#include <Grafo.h>`

Inheritance diagram for Grafo:



Public Member Functions

- Grafo ()=default

  Construtor padrão.
- virtual ~Grafo ()=default

  Destrutor virtual.
- virtual int get_aresta (int origem, int destino)=0

  Retorna o peso da aresta entre dois vértices.
- virtual int get_vertice (int vertice)=0

  Obtém o peso de um vértice.
- virtual int get_vizinhos (int vertice)=0

  Retorna o número de vizinhos de um vértice.
- virtual void nova_aresta (int origem, int destino, int peso)=0
- virtual void deleta_aresta (int vertice1, int vertice2)=0
- virtual void set_aresta (int origem, int destino, float peso)=0
- virtual void set_vertice (int id, float peso)=0
- virtual void novo_no (int peso)=0
- virtual void deleta_no (int vertice)=0
- int get_ordem ()
- void set_ordem (int ordem)
- void aumenta_ordem ()
- bool eh_direcionado ()
- void set_eh_direcionado (bool direcionado)
- bool vertice_ponderado ()
- void set_vertice_ponderado (bool verticePonderado)
- bool aresta_ponderada ()
- void set_aresta_ponderada (bool arestaPonderada)
- void carrega_grafo ()

  Carrega o grafo a partir de um arquivo de entrada.
- int get_grau ()

  Calcula o grau máximo do grafo.
- bool eh_completo ()
- void dfs (int vertice, bool visitado[ ])
- int n_conexo ()

  Determina o número de componentes conexas do grafo.
- void maior_menor_distancia ()

  Calcula a maior menor distância entre dois vértices utilizando o algoritmo de Floyd-Warshall.

4.2.1 Detailed Description

Classe abstrata que representa um grafo.

Define operações fundamentais para manipulação de grafos, como adição e remoção de arestas e vértices. Essa classe serve como base para diferentes representações de grafos.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Grafo()

```
Grafo::Grafo ()  [default]
```

Construtor padrão.

4.2.2.2 ~Grafo()

```
virtual Grafo::~Grafo ()  [virtual], [default]
```

Destrutor virtual.

4.2.3 Member Function Documentation

4.2.3.1 aresta_ponderada()

```
bool Grafo::aresta_ponderada ()  [inline]
```

4.2.3.2 aumenta_ordem()

```
void Grafo::aumenta_ordem ()  [inline]
```

4.2.3.3 carrega_grafo()

```
void Grafo::carrega_grafo ()  [inline]
```

Carrega o grafo a partir de um arquivo de entrada.

O formato do arquivo deve conter:

- Número de vértices, se é direcionado e se é ponderado.
- Lista de vértices e seus pesos (se for ponderado).
- Lista de arestas com origem, destino e peso (se for ponderado).

4.2.3.4 deleta_aresta()

```
virtual void Grafo::deleta_aresta (
            int vertice1,
            int vertice2)  [pure virtual]
```

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.5 deleta_no()

```
virtual void Grafo::deleta_no (
            int vertice)  [pure virtual]
```

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.6 dfs()

```
void Grafo::dfs (
            int vertice,
            bool visitado[])  [inline]
```

4.2.3.7 eh_completo()

```
bool Grafo::eh_completo ()  [inline]
```

4.2.3.8 eh_direcionado()

```
bool Grafo::eh_direcionado ()  [inline]
```

4.2.3.9 get_aresta()

```
virtual int Grafo::get_aresta (
            int origem,
            int destino)  [pure virtual]
```

Retorna o peso da aresta entre dois vértices.

Parameters

origem	Vértice de origem.
destino	Vértice de destino.

Returns

Peso da aresta ou um valor indicando inexistência.

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.10 get_grau()

```
int Grafo::get_grau ()  [inline]
```

Calcula o grau máximo do grafo.

Returns

O maior grau encontrado entre os vértices.

4.2.3.11 get_ordem()

```
int Grafo::get_ordem ()  [inline]
```

4.2.3.12 get_vertice()

```
virtual int Grafo::get_vertice (
            int vertice)  [pure virtual]
```

Obtém o peso de um vértice.

Parameters

vertice	Identificador do vértice.

Returns

Peso do vértice.

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.13 get_vizinhos()

```
virtual int Grafo::get_vizinhos (
            int vertice)  [pure virtual]
```

Retorna o número de vizinhos de um vértice.

Parameters

vertice	Identificador do vértice.

Returns

Número de vizinhos do vértice.

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.14 maior_menor_distancia()

```
void Grafo::maior_menor_distancia ()  [inline]
```

Calcula a maior menor distância entre dois vértices utilizando o algoritmo de Floyd-Warshall.

4.2.3.15 n_conexo()

```
int Grafo::n_conexo ()  [inline]
```

Determina o número de componentes conexas do grafo.

Returns

Número de componentes conexas.

4.2.3.16 nova_aresta()

```
virtual void Grafo::nova_aresta (
            int origem,
            int destino,
            int peso)  [pure virtual]
```

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.17 novo_no()

```
virtual void Grafo::novo_no (
            int peso)  [pure virtual]
```

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.18 set_aresta()

```
virtual void Grafo::set_aresta (
            int origem,
            int destino,
            float peso)  [pure virtual]
```

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.19 set_aresta_ponderada()

```
void Grafo::set_aresta_ponderada (
            bool arestaPonderada)  [inline]
```

4.2.3.20 set_eh_direcionado()

```
void Grafo::set_eh_direcionado (
            bool direcionado)  [inline]
```

4.2.3.21 set_ordem()

```
void Grafo::set_ordem (
            int ordem)  [inline]
```

4.2.3.22 set_vertice()

```
virtual void Grafo::set_vertice (
            int id,
            float peso)  [pure virtual]
```

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.23 set_vertice_ponderado()

```
void Grafo::set_vertice_ponderado (
            bool verticePonderado)  [inline]
```

4.2.3.24 vertice_ponderado()

```
bool Grafo::vertice_ponderado ()  [inline]
```

The documentation for this class was generated from the following file:

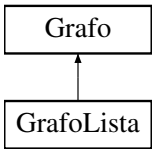
- include/Grafo.h

4.3 GrafoLista Class Reference

Implementação de um grafo usando listas encadeadas.

`#include <GrafoLista.h>`

Inheritance diagram for GrafoLista:



Public Member Functions

- GrafoLista ()
  Construtor da classe GrafoLista.
- int get_vertice (int id) override
  Obtém o peso de um vértice.
- int get_aresta (int idOrigem, int idDestino) override
  Obtém o peso da aresta entre dois vértices.
- void set_vertice (int id, float peso) override
  Define o peso de um vértice.
- void set_aresta (int origem, int destino, float peso) override
  Define o peso de uma aresta existente.
- void nova_aresta (int origem, int destino, int peso) override
  Adiciona uma nova aresta ao grafo.
- void deleta_aresta (int vertice1, int vertice2) override
  Remove uma aresta do grafo.
- int get_vizinhos (int vertice) override
  Obtém o número de vizinhos de um vértice.
- void novo_no (int peso) override
  Adiciona um novo vértice ao grafo.
- void deleta_no (int vertice) override
  Remove um vértice do grafo e todas suas conexões.
- void imprimir ()
  Imprime a representação do grafo.
- ~GrafoLista ()
  Destrutor da classe GrafoLista.

Public Member Functions inherited from Grafo

- Grafo ()=default

  Construtor padrão.
- virtual ~Grafo ()=default

  Destrutor virtual.
- int get_ordem ()
- void set_ordem (int ordem)
- void aumenta_ordem ()
- bool eh_direcionado ()
- void set_eh_direcionado (bool direcionado)
- bool vertice_ponderado ()
- void set_vertice_ponderado (bool verticePonderado)
- bool aresta_ponderada ()
- void set_aresta_ponderada (bool arestaPonderada)
- void carrega_grafo ()

  Carrega o grafo a partir de um arquivo de entrada.
- int get_grau ()

  Calcula o grau máximo do grafo.
- bool eh_completo ()
- void dfs (int vertice, bool visitado[ ])
- int n_conexo ()

  Determina o número de componentes conexas do grafo.
- void maior_menor_distancia ()

  Calcula a maior menor distância entre dois vértices utilizando o algoritmo de Floyd-Warshall.

4.3.1 Detailed Description

Implementação de um grafo usando listas encadeadas.

Esta classe herda da classe abstrata `Grafo` e implementa suas funcionalidades utilizando listas encadeadas para armazenar os vértices e arestas.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 GrafoLista()

```
GrafoLista::GrafoLista ()
```

Construtor da classe GrafoLista.

Inicializa a estrutura do grafo utilizando listas encadeadas.

Inicializa as listas encadeadas de vértices e arestas.

4.3.2.2 ~GrafoLista()

```
GrafoLista::~GrafoLista ()
```

Destrutor da classe GrafoLista.

Libera a memória alocada para as listas encadeadas de vértices e arestas.

4.3.3 Member Function Documentation

4.3.3.1 deleta_aresta()

```
void GrafoLista::deleta_aresta (
            int origem,
            int destino)  [override], [virtual]
```

Remove uma aresta do grafo.

Parameters

vertice1	Identificador do primeiro vértice.
vertice2	Identificador do segundo vértice.
origem	Identificador do vértice de origem.
destino	Identificador do vértice de destino.

Implements Grafo.

4.3.3.2 deleta_no()

```
void GrafoLista::deleta_no (
            int id)  [override], [virtual]
```

Remove um vértice do grafo e todas suas conexões.

Remove um vértice e todas as suas conexões do grafo.

Parameters

vertice	Identificador do vértice a ser removido.
id	Identificador do vértice a ser removido.

Implements Grafo.

4.3.3.3 get_aresta()

```
int GrafoLista::get_aresta (
            int idOrigem,
            int idDestino)  [override], [virtual]
```

Obtém o peso da aresta entre dois vértices.

Obtém o peso de uma aresta entre dois vértices.

Parameters

idOrigem	Identificador do vértice de origem.
idDestino	Identificador do vértice de destino.

Returns

Peso da aresta ou -1 se não existir.

Implements Grafo.

4.3.3.4 get_vertice()

```
int GrafoLista::get_vertice (
            int id)  [override], [virtual]
```

Obtém o peso de um vértice.

Obtém o peso de um vértice pelo seu identificador.

Parameters

id	Identificador do vértice.

Returns

Peso do vértice ou -1 se não existir.

Implements Grafo.

4.3.3.5 get_vizinhos()

```
int GrafoLista::get_vizinhos (
            int id)  [override], [virtual]
```

Obtém o número de vizinhos de um vértice.

Parameters

vertice	Identificador do vértice.

Returns

Número de vizinhos do vértice.

Parameters

id	Identificador do vértice.

Returns

Número de vizinhos do vértice.

Implements Grafo.

4.3.3.6 imprimir()

```
void GrafoLista::imprimir ()
```

Imprime a representação do grafo.

Imprime os vértices e as arestas do grafo.

4.3.3.7 nova_aresta()

```
void GrafoLista::nova_aresta (
            int origem,
            int destino,
            int peso)  [override], [virtual]
```

Adiciona uma nova aresta ao grafo.

Parameters

origem	Identificador do vértice de origem.
destino	Identificador do vértice de destino.
peso	Peso da aresta.

Implements Grafo.

4.3.3.8 novo_no()

```
void GrafoLista::novo_no (
            int peso)  [override], [virtual]
```

Adiciona um novo vértice ao grafo.

Parameters

peso	Peso do novo vértice.

Implements Grafo.

4.3.3.9 set_aresta()

```
void GrafoLista::set_aresta (
            int origem,
            int destino,
            float peso)  [override], [virtual]
```

Define o peso de uma aresta existente.

Adiciona uma nova aresta ao grafo.

Parameters

origem	Identificador do vértice de origem.
destino	Identificador do vértice de destino.
peso	Novo peso da aresta.
origem	Identificador do vértice de origem.
destino	Identificador do vértice de destino.
peso	Peso da aresta.

Implements Grafo.

4.3.3.10 set_vertice()

```
void GrafoLista::set_vertice (
            int id,
            float peso)  [override], [virtual]
```

Define o peso de um vértice.

Adiciona um novo vértice ao grafo.

Parameters

id	Identificador do vértice.
peso	Novo peso do vértice.
id	Identificador do vértice.
peso	Peso do vértice.

Implements Grafo.

The documentation for this class was generated from the following files:

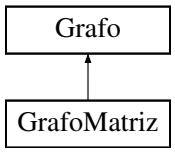
- include/GrafoLista.h
- src/GrafoLista.cpp

4.4 GrafoMatriz Class Reference

Implementação de um grafo utilizando matriz de adjacência.

`#include <GrafoMatriz.h>`

Inheritance diagram for GrafoMatriz:



Public Member Functions

- GrafoMatriz ()

  Construtor da classe GrafoMatriz.
- virtual ~GrafoMatriz ()

  Destrutor da classe GrafoMatriz.
- void redimensionarMatriz ()

  Redimensiona a matriz de adjacência para acomodar novos vértices.
- void redimensionarMatrizLinear ()

  Redimensiona a matriz linear de adjacência para grafos não direcionados.
- int calcularIndiceLinear (int origem, int destino)

  Calcula o índice correspondente na matriz linear para uma aresta.
- int get_aresta (int origem, int destino) override

  Obtém o peso da aresta entre dois vértices.
- int get_vertice (int vertice) override

  Obtém o peso de um vértice.
- int get_vizinhos (int vertice) override

  Retorna o número de vizinhos de um vértice.
- void set_vertice (int id, float peso) override

  Define o peso de um vértice.
- void set_aresta (int origem, int destino, float peso) override

Define o peso de uma aresta existente.

- void nova_aresta (int origem, int destino, int peso)

  Adiciona uma nova aresta ao grafo.
- void novo_no (int peso) override

  Adiciona um novo vértice ao grafo.
- void deleta_no (int vertice) override

  Remove um vértice do grafo e todas suas conexões.
- void deleta_arestas_direcionadas (int vertice)

  Remove todas as arestas direcionadas associadas a um vértice.
- void deleta_arestas_nao_direcionadas (int vertice)

  Remove todas as arestas não direcionadas associadas a um vértice.
- void reorganiza_matriz (int vertice)

  Reorganiza a matriz de adjacência após a remoção de um vértice.
- void reorganiza_vetor_pesos (int vertice)

  Reorganiza o vetor de pesos dos vértices após a remoção de um vértice.
- void deleta_aresta (int vertice1, int vertice2) override

  Remove uma aresta do grafo.

Public Member Functions inherited from Grafo

- Grafo ()=default

  Construtor padrão.
- virtual ~Grafo ()=default

  Destrutor virtual.
- int get_ordem ()
- void set_ordem (int ordem)
- void aumenta_ordem ()
- bool eh_direcionado ()
- void set_eh_direcionado (bool direcionado)
- bool vertice_ponderado ()
- void set_vertice_ponderado (bool verticePonderado)
- bool aresta_ponderada ()
- void set_aresta_ponderada (bool arestaPonderada)
- void carrega_grafo ()

  Carrega o grafo a partir de um arquivo de entrada.
- int get_grau ()

  Calcula o grau máximo do grafo.
- bool eh_completo ()
- void dfs (int vertice, bool visitado[ ])
- int n_conexo ()

  Determina o número de componentes conexas do grafo.
- void maior_menor_distancia ()

  Calcula a maior menor distância entre dois vértices utilizando o algoritmo de Floyd-Warshall.

4.4.1 Detailed Description

Implementação de um grafo utilizando matriz de adjacência.

Esta classe herda da classe abstrata `Grafo` e implementa suas funcionalidades usando uma matriz 2D para armazenar as conexões entre os vértices.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 GrafoMatriz()

```
GrafoMatriz::GrafoMatriz ()
```

Construtor da classe GrafoMatriz.

Inicializa a matriz de adjacência e o vetor de pesos dos vértices.

Inicializa a matriz de adjacência, a matriz linear e o vetor de pesos dos vértices. O grafo começa com um tamanho inicial definido por `TAMANHO_INICIAL`.

4.4.2.2 ~GrafoMatriz()

```
GrafoMatriz::~GrafoMatriz ()  [virtual]
```

Destrutor da classe GrafoMatriz.

Libera a memória alocada para a matriz de adjacência e demais estruturas.

Libera a memória alocada para a matriz de adjacência, a matriz linear e o vetor de pesos dos vértices. Garante que todos os ponteiros sejam corretamente liberados e evita dangling pointers.

4.4.3 Member Function Documentation

4.4.3.1 calcularIndiceLinear()

```
int GrafoMatriz::calcularIndiceLinear (
            int origem,
            int destino)
```

Calcula o índice correspondente na matriz linear para uma aresta.

Parameters

origem	Vértice de origem da aresta.
destino	Vértice de destino da aresta.

Returns

Índice na matriz linear correspondente à aresta.

A matriz linear armazena apenas a metade da matriz de adjacência para economizar espaço. Este método converte um par (origem, destino) em um índice dentro dessa matriz comprimida.

Parameters

origem	Vértice de origem da aresta.
destino	Vértice de destino da aresta.

Returns

Índice correspondente na matriz linear.

4.4.3.2 deleta_aresta()

```
void GrafoMatriz::deleta_aresta (
            int vertice1,
            int vertice2)  [override], [virtual]
```

Remove uma aresta do grafo.

Parameters

vertice1	Identificador do primeiro vértice.
vertice2	Identificador do segundo vértice.

Para grafos direcionados, apenas a entrada correspondente na matriz de adjacência é apagada. Para grafos não direcionados, a aresta é removida da matriz linear comprimida.

Parameters

vertice1	Identificador do primeiro vértice.
vertice2	Identificador do segundo vértice.

Implements Grafo.

4.4.3.3 deleta_arestas_direcionadas()

```
void GrafoMatriz::deleta_arestas_direcionadas (
            int vertice)
```

Remove todas as arestas direcionadas associadas a um vértice.

Parameters

vertice	Identificador do vértice.

Remove todas as conexões de saída e entrada do vértice na matriz de adjacência.

Parameters

vertice	Identificador do vértice.

4.4.3.4 deleta_arestas_nao_direcionadas()

```
void GrafoMatriz::deleta_arestas_nao_direcionadas (
            int vertice)
```

Remove todas as arestas não direcionadas associadas a um vértice.

Parameters

vertice	Identificador do vértice.

Remove as conexões do vértice na matriz linear de adjacência.

Parameters

vertice	Identificador do vértice.

4.4.3.5 deleta_no()

```
void GrafoMatriz::deleta_no (
            int vertice)  [override], [virtual]
```

Remove um vértice do grafo e todas suas conexões.

Remove um vértice do grafo e reorganiza suas estruturas.

Parameters

vertice	Identificador do vértice a ser removido.

Remove todas as arestas associadas ao vértice e reorganiza a matriz de adjacência e o vetor de pesos. Reduz a ordem do grafo.

Parameters

vertice	Identificador do vértice a ser removido.

Implements Grafo.

4.4.3.6 get_aresta()

```
int GrafoMatriz::get_aresta (
            int origem,
            int destino)  [override], [virtual]
```

Obtém o peso da aresta entre dois vértices.

Parameters

origem	Vértice de origem.
destino	Vértice de destino.

Returns

Peso da aresta ou -1 se não existir.

Retorna o valor armazenado na matriz de adjacência ou na matriz linear, dependendo se o grafo é direcionado ou não.

Parameters

origem	Identificador do vértice de origem.
destino	Identificador do vértice de destino.

Returns

Peso da aresta ou -1 se não existir.

Implements Grafo.

4.4.3.7 get_vertice()

```
int GrafoMatriz::get_vertice (
            int origem)  [override], [virtual]
```

Obtém o peso de um vértice.

Parameters

vertice	Identificador do vértice.

Returns

Peso do vértice.

Se o grafo for ponderado nos vértices, retorna o peso armazenado. Caso contrário, retorna 1 (peso padrão para vértices não ponderados).

Parameters

origem	Identificador do vértice.

Returns

Peso do vértice.

Implements Grafo.

4.4.3.8 get_vizinhos()

```
int GrafoMatriz::get_vizinhos (
            int vertice)  [override], [virtual]
```

Retorna o número de vizinhos de um vértice.

Obtém o número de vizinhos de um vértice.

Parameters

vertice	Identificador do vértice.

Returns

Número de vizinhos do vértice.

Para grafos direcionados, conta o número de arestas de saída do vértice. Para grafos não direcionados, verifica a matriz linear de adjacência, evitando contagens duplicadas.

Parameters

vertice	Identificador do vértice.

Returns

Número de vizinhos do vértice ou -1 se o vértice for inválido.

Implements Grafo.

4.4.3.9 nova_aresta()

```
void GrafoMatriz::nova_aresta (
            int origem,
            int destino,
            int peso)  [virtual]
```

Adiciona uma nova aresta ao grafo.

Parameters

origem	Vértice de origem.
destino	Vértice de destino.
peso	Peso da aresta.

A função verifica se a aresta já existe antes de adicioná-la. Para grafos direcionados, a aresta é armazenada na matriz de adjacência. Para grafos não direcionados, a aresta é armazenada na matriz linear comprimida.

Parameters

origem	Identificador do vértice de origem.
destino	Identificador do vértice de destino.
peso	Peso da aresta.

Adiciona uma nova aresta ao grafo.

A função verifica se a aresta já existe antes de adicioná-la. Para grafos direcionados, a aresta é armazenada na matriz de adjacência. Para grafos não direcionados, a aresta é armazenada na matriz linear comprimida.

Parameters

origem	Identificador do vértice de origem.
destino	Identificador do vértice de destino.
peso	Peso da aresta.

Implements Grafo.

4.4.3.10 novo_no()

```
void GrafoMatriz::novo_no (
            int peso)  [override], [virtual]
```

Adiciona um novo vértice ao grafo.

Parameters

peso	Peso do novo vértice.

Aumenta a ordem do grafo e redimensiona a matriz de adjacência ou a matriz linear se necessário. Inicializa a nova linha e a nova coluna da matriz com zeros.

Parameters

peso	Peso do novo vértice.

Implements Grafo.

4.4.3.11 redimensionarMatriz()

```
void GrafoMatriz::redimensionarMatriz ()
```

Redimensiona a matriz de adjacência para acomodar novos vértices.

A matriz é duplicada em tamanho para evitar realocações frequentes. Os valores da matriz original são copiados para a nova matriz, e as novas posições são inicializadas com zero. Após a cópia, a matriz original é desalocada.

4.4.3.12 redimensionarMatrizLinear()

```
void GrafoMatriz::redimensionarMatrizLinear ()
```

Redimensiona a matriz linear de adjacência para grafos não direcionados.

Redimensiona a matriz linear de adjacência para acomodar novos vértices.

A matriz linear é usada para representar grafos não direcionados de forma eficiente. Essa função realoca a matriz linear, copiando os valores existentes e inicializando novas posições com zero. Após a cópia, a matriz original é desalocada.

4.4.3.13 reorganiza_matriz()

```
void GrafoMatriz::reorganiza_matriz (
            int vertice)
```

Reorganiza a matriz de adjacência após a remoção de um vértice.

Parameters

vertice	Identificador do vértice removido.

Cria uma nova matriz sem a linha e a coluna do vértice removido, copiando os dados restantes.

Parameters

vertice	Identificador do vértice removido.

4.4.3.14 reorganiza_vetor_pesos()

```
void GrafoMatriz::reorganiza_vetor_pesos (
            int vertice)
```

Reorganiza o vetor de pesos dos vértices após a remoção de um vértice.

Parameters

vertice	Identificador do vértice removido.

Cria um novo vetor de pesos sem o valor correspondente ao vértice removido.

Parameters

vertice	Identificador do vértice removido.

4.4.3.15 set_aresta()

```
void GrafoMatriz::set_aresta (
            int origem,
            int destino,
            float peso)  [override], [virtual]
```

Define o peso de uma aresta existente.

Parameters

origem	Vértice de origem.
destino	Vértice de destino.
peso	Novo peso da aresta.

Se o grafo não for ponderado nas arestas, o peso será forçado para 0. O método verifica se o grafo é direcionado e armazena o peso na estrutura correspondente.

Parameters

origem	Identificador do vértice de origem.
destino	Identificador do vértice de destino.
peso	Novo peso da aresta.

Implements Grafo.

4.4.3.16 set_vertice()

```
void GrafoMatriz::set_vertice (
            int id,
            float peso)  [override], [virtual]
```

Define o peso de um vértice.

Parameters

id	Identificador do vértice.
peso	Novo peso do vértice.

Se o grafo não for ponderado nos vértices, o peso será forçado para 0.

Parameters

id	Identificador do vértice.
peso	Novo peso do vértice.

Implements Grafo.

The documentation for this class was generated from the following files:

- include/GrafoMatriz.h
- src/GrafoMatriz.cpp

4.5 ListaEncadeada< T > Class Template Reference

Implementação genérica de uma lista encadeada.

`#include <ListaEncadeada.h>`

Public Member Functions

- ListaEncadeada ()
  Construtor da lista encadeada.
- T * getInicio () const
  Obtém o primeiro nó da lista.
- void setInicio (T *novoInicio)
  Define um novo primeiro nó na lista.
- void adicionar (T *novoNo)
  Adiciona um novo nó ao final da lista.
- void imprimir () const
  Imprime todos os elementos da lista encadeada.
- void remover (T *noParaRemover)
  Remove um nó específico da lista encadeada.
- int get_tamanho ()
  Obtém o tamanho atual da lista encadeada.
- ~ListaEncadeada ()
  Destrutor da lista encadeada.

4.5.1 Detailed Description

template<typename T>
class ListaEncadeada< T >

Implementação genérica de uma lista encadeada.

Esta classe gerencia uma lista encadeada de elementos do tipo `T`. Suporta operações básicas como adição, remoção e impressão dos elementos.

Template Parameters

T	Tipo dos elementos armazenados na lista.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 ListaEncadeada()

```
template<typename T>
ListaEncadeada< T >::ListaEncadeada () [inline]
```

Construtor da lista encadeada.

Inicializa a lista como vazia, com `primeiro` e `ultimo` apontando para nullptr.

4.5.2.2 ~ListaEncadeada()

```
template<typename T>
ListaEncadeada< T >::~ListaEncadeada ()  [inline]
```

Destrutor da lista encadeada.

Libera a memória de todos os nós armazenados na lista.

4.5.3 Member Function Documentation

4.5.3.1 adicionar()

```
template<typename T>
void ListaEncadeada< T >::adicionar (
            T * novoNo)  [inline]
```

Adiciona um novo nó ao final da lista.

Parameters

novoNo	Ponteiro para o novo nó a ser adicionado.

4.5.3.2 get_tamanho()

```
template<typename T>
int ListaEncadeada< T >::get_tamanho ()  [inline]
```

Obtém o tamanho atual da lista encadeada.

Returns

Número de elementos na lista.

4.5.3.3 getInicio()

```
template<typename T>
T * ListaEncadeada< T >::getInicio () const  [inline]
```

Obtém o primeiro nó da lista.

Returns

Ponteiro para o primeiro nó da lista.

4.5.3.4 imprimir()

```
template<typename T>
void ListaEncadeada< T >::imprimir () const  [inline]
```

Imprime todos os elementos da lista encadeada.

4.5.3.5 remover()

```
template<typename T>
void ListaEncadeada< T >::remover (
            T * noParaRemover)  [inline]
```

Remove um nó específico da lista encadeada.

Parameters

noParaRemover	Ponteiro para o nó que será removido.

4.5.3.6 setInicio()

```
template<typename T>
void ListaEncadeada< T >::setInicio (
            T * novoInicio)  [inline]
```

Define um novo primeiro nó na lista.

Parameters

novoInicio	Ponteiro para o novo primeiro nó.

The documentation for this class was generated from the following file:

- include/ListaEncadeada.h

4.6 VerticeEncadeado Class Reference

Representa um vértice em uma estrutura de grafo utilizando listas encadeadas.

`#include <VerticeEncadeado.h>`

Public Member Functions

- VerticeEncadeado (int id, int peso)
  Construtor da classe VerticeEncadeado.
- int getId () const
  Obtém o identificador do vértice.
- int getPeso () const
  Obtém o peso do vértice.
- int getGrau () const
  Obtém o grau do vértice (número de conexões).
- VerticeEncadeado * getProximo () const
  Obtém o próximo vértice na lista encadeada.
- void setProximo (VerticeEncadeado *novoProximo)
  Define o próximo vértice na lista encadeada.
- void setConexao (VerticeEncadeado *verticeDestino, int pesoAresta)
  Cria uma conexão entre este vértice e outro vértice destino.
- ArestaEncadeada * getPrimeiraConexao ()
  Obtém a primeira conexão (aresta) do vértice.
- ListaEncadeada< ArestaEncadeada > * getConexoes ()
  Obtém a lista de conexões (arestas) do vértice.
- void setConexoes (ListaEncadeada< ArestaEncadeada > *novasConexoes)
  Define a lista de conexões (arestas) do vértice.
- int removeConexao (VerticeEncadeado *destino)
  Remove uma conexão específica deste vértice para outro vértice.
- ArestaEncadeada * getConexao (int origem, int destino)
  Obtém uma conexão específica entre dois vértices.

Friends

- std::ostream & operator<< (std::ostream &os, const VerticeEncadeado &vertice)

  Sobrecarga do operador de saída para exibir informações do vértice.

4.6.1 Detailed Description

Representa um vértice em uma estrutura de grafo utilizando listas encadeadas.

Cada vértice possui um identificador único, um peso, um grau (número de conexões) e uma lista encadeada de conexões (arestas).

4.6.2 Constructor & Destructor Documentation

4.6.2.1 VerticeEncadeado()

```
VerticeEncadeado::VerticeEncadeado (
            int id,
            int peso)
```

Construtor da classe VerticeEncadeado.

Parameters

id	Identificador do vértice.
peso	Peso do vértice.

Inicializa um vértice com um identificador e peso fornecidos. O grau é inicialmente 0, e a lista de conexões é alocada dinamicamente.

Parameters

id	Identificador único do vértice.
peso	Peso do vértice.

4.6.3 Member Function Documentation

4.6.3.1 getConexao()

```
ArestaEncadeada * VerticeEncadeado::getConexao (
            int origem,
            int destino)
```

Obtém uma conexão específica entre dois vértices.

Parameters

origem	Identificador do vértice de origem.
destino	Identificador do vértice de destino.

Returns

Ponteiro para a aresta correspondente ou nullptr se não existir.

Percorre a lista de conexões para encontrar uma aresta que conecte o vértice de origem ao de destino.

Parameters

origem	Identificador do vértice de origem.
destino	Identificador do vértice de destino.

Returns

Ponteiro para a aresta correspondente ou nullptr se não existir.

4.6.3.2 getConexoes()

```
ListaEncadeada< ArestaEncadeada > * VerticeEncadeado::getConexoes ()
```

Obtém a lista de conexões (arestas) do vértice.

Returns

Ponteiro para a lista encadeada de arestas.

4.6.3.3 getGrau()

```
int VerticeEncadeado::getGrau () const
```

Obtém o grau do vértice (número de conexões).

Returns

Grau do vértice.

4.6.3.4 getId()

```
int VerticeEncadeado::getId () const
```

Obtém o identificador do vértice.

Returns

Identificador do vértice.

O identificador do vértice.

4.6.3.5 getPeso()

```
int VerticeEncadeado::getPeso () const
```

Obtém o peso do vértice.

Returns

Peso do vértice.

O peso do vértice.

4.6.3.6 getPrimeiraConexao()

```
ArestaEncadeada * VerticeEncadeado::getPrimeiraConexao ()
```

Obtém a primeira conexão (aresta) do vértice.

Returns

Ponteiro para a primeira aresta conectada a este vértice.

Ponteiro para a primeira aresta conectada ao vértice.

4.6.3.7 getProximo()

```
VerticeEncadeado * VerticeEncadeado::getProximo () const
```

Obtém o próximo vértice na lista encadeada.

Obtém o próximo vértice encadeado.

Returns

Ponteiro para o próximo vértice.

Ponteiro para o próximo vértice na lista encadeada.

4.6.3.8 removeConexao()

```
int VerticeEncadeado::removeConexao (
            VerticeEncadeado * destino)
```

Remove uma conexão específica deste vértice para outro vértice.

Parameters

destino	Ponteiro para o vértice de destino.

Returns

1 se a conexão foi removida com sucesso, 0 caso contrário.

Encontra a aresta que conecta este vértice ao vértice de destino e a remove da lista de conexões. Reduz o grau do vértice após a remoção.

Parameters

destino	Ponteiro para o vértice de destino da conexão a ser removida.

Returns

Peso da aresta removida ou 0 se a conexão não existir.

4.6.3.9 setConexao()

```
void VerticeEncadeado::setConexao (
            VerticeEncadeado * verticeDestino,
            int pesoAresta)
```

Cria uma conexão entre este vértice e outro vértice destino.

Adiciona uma conexão entre este vértice e outro vértice.

Parameters

verticeDestino	Ponteiro para o vértice de destino.
pesoAresta	Peso da aresta entre os vértices.

Cria uma nova aresta conectando este vértice ao vértice de destino e a adiciona à lista de conexões.

Parameters

verticeDestino	Ponteiro para o vértice de destino.
pesoAresta	Peso da aresta que conecta os dois vértices.

4.6.3.10 setConexoes()

```
void VerticeEncadeado::setConexoes (
            ListaEncadeada< ArestaEncadeada > * novasConexoes)
```

Define a lista de conexões (arestas) do vértice.

Define a lista de conexões do vértice.

Parameters

novasConexoes	Ponteiro para a nova lista de conexões.

4.6.3.11 setProximo()

```
void VerticeEncadeado::setProximo (
            VerticeEncadeado * novoProximo)
```

Define o próximo vértice na lista encadeada.

Parameters

novoProximo	Ponteiro para o novo próximo vértice.

4.6.4 Friends And Related Symbol Documentation

4.6.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const VerticeEncadeado & vertice)  [friend]
```

Sobrecarga do operador de saída para exibir informações do vértice.

Parameters

os	Fluxo de saída.
vertice	Vértice a ser exibido.

Returns

Fluxo de saída atualizado.

Exibe o identificador, peso e grau do vértice, bem como suas conexões.

Parameters

os	Fluxo de saída.
vertice	Vértice a ser exibido.

Returns

Fluxo de saída atualizado.

The documentation for this class was generated from the following files:

- include/VerticeEncadeado.h
- src/VerticeEncadeado.cpp

Chapter 5

File Documentation

5.1 include/ArestaEncadeada.h File Reference

```
#include <iostream>
```

Classes

- class ArestaEncadeada

  Representa uma aresta em um grafo utilizando uma estrutura de lista encadeada.

5.2 ArestaEncadeada.h

Go to the documentation of this file.

```
00001 #ifndef ARESTAENCADEADA_H_INCLUDED
00002 #define ARESTAENCADEADA_H_INCLUDED
00003
00004 #include <iostream>
00005
00006 class VerticeEncadeado;
00007
00016 class ArestaEncadeada {
00017 private:
00018     VerticeEncadeado* origem;
00019     VerticeEncadeado* destino;
00020     float peso;
00021     ArestaEncadeada* proximo;
00022
00023 public:
00030     ArestaEncadeada(VerticeEncadeado* origem, VerticeEncadeado* destino, float peso);
00031
00036     VerticeEncadeado* getOrigem() const;
00037
00042     VerticeEncadeado* getDestino() const;
00043
00048     float getPeso() const;
00049
00054     ArestaEncadeada* getProximo() const;
00055
00060     void setProximo(ArestaEncadeada* novoProximo);
00061
00068     friend std::ostream& operator<<(std::ostream& os, const ArestaEncadeada& aresta);
00069 };
00070
00071 #endif // ARESTAENCADEADA_H_INCLUDED
```

5.3 include/Grafo.h File Reference

```
#include <iostream>
#include <fstream>
```

Classes

- class Grafo

  Classe abstrata que representa um grafo.

5.4 Grafo.h

Go to the documentation of this file.

```
00001 #ifndef GRAFO_H_INCLUDED
00002 #define GRAFO_H_INCLUDED
00003 #include <iostream>
00004 #include <fstream>
00005
00006 using namespace std;
00007
00015 class Grafo
00016 {
00017 private:
00018     bool direcionado;
00019     bool vtp;
00020     bool atp;
00021     int ordem;
00022     int origem, destino, peso;
00023
00024 public:
00025     Grafo() = default;
00026     virtual ~Grafo() = default;
00027
00034     virtual int get_aresta(int origem, int destino) = 0;
00035
00041     virtual int get_vertice(int vertice) = 0;
00042
00048     virtual int get_vizinhos(int vertice) = 0;
00049
00050     virtual void nova_aresta(int origem, int destino, int peso) = 0;
00051     virtual void deleta_aresta(int vertice1, int vertice2) = 0;
00052
00053     virtual void set_aresta(int origem, int destino, float peso) = 0;
00054     virtual void set_vertice(int id, float peso) = 0;
00055
00056     virtual void novo_no(int peso) = 0;
00057     virtual void deleta_no(int vertice) = 0;
00058
00059     int get_ordem()
00060     {
00061         return ordem;
00062     };
00063
00064     void set_ordem(int ordem)
00065     {
00066         this->ordem = ordem;
00067     };
00068
00069     void aumenta_ordem()
00070     {
00071         this->ordem++;
00072     };
00073
00074     bool eh_direcionado()
00075     {
00076         return direcionado;
00077     }
00078
00079     void set_eh_direcionado(bool direcionado)
00080     {
00081         this->direcionado = direcionado;
00082     };
```

```
00083
00084     bool vertice_ponderado()
00085     {
00086         return vtp;
00087     }
00088
00089     void set_vertice_ponderado(bool verticePonderado)
00090     {
00091         this->vtp = verticePonderado;
00092     };
00093
00094     bool aresta_ponderada()
00095     {
00096         return atp;
00097     }
00098
00099     void set_aresta_ponderada(bool arestaPonderada)
00100     {
00101         this->atp = arestaPonderada;
00102     };
00103
00112     void carrega_grafo()
00113     {
00114         ifstream arquivo("./entradas/Grafo.txt");
00115         if (!arquivo.is_open())
00116         {
00117             cerr « "Erro ao abrir o arquivo Grafo.txt" « endl;
00118             return;
00119         }
00120
00121         arquivo » ordem » direcionado » vtp » atp;
00122         set_ordem(ordem);
00123         set_eh_direcionado(direcionado);
00124         set_vertice_ponderado(vtp);
00125         set_aresta_ponderada(atp);
00126
00127         for (int i = 1; i <= ordem; i++)
00128         {
00129             int peso_vertice;
00130             arquivo » peso_vertice;
00131
00132             if (vertice_ponderado())
00133                 set_vertice(i, peso_vertice);
00134             else
00135                 set_vertice(i, 1);
00136         }
00137
00138         int origem, destino = 1;
00139         float peso = 0;
00140
00141         while (arquivo » origem » destino » peso)
00142         {
00143             if (!aresta_ponderada())
00144                 peso = 0;
00145             set_aresta(origem, destino, peso);
00146         }
00147     }
00148
00153     int get_grau()
00154     {
00155         if (!eh_direcionado())
00156         {
00157             int grauMaximo = 0;
00158             for (int i = 1; i <= ordem; i++)
00159             {
00160                 int numVizinhos = get_vizinhos(i);
00161                 if (numVizinhos > grauMaximo)
00162                 {
00163                     grauMaximo = numVizinhos;
00164                 }
00165             }
00166             return grauMaximo;
00167         }
00168         else
00169         {
00170             int maxGrauSaida = 0;
00171             for (int i = 1; i <= ordem; i++)
00172             {
00173                 int grauSaida = get_vizinhos(i);
00174                 if (grauSaida > maxGrauSaida)
00175                 {
00176                     maxGrauSaida = grauSaida;
00177                 }
00178             }
00179             return maxGrauSaida;
00180         }
00181     }
```

```
00182
00183     bool eh_completo()
00184     {
00185         for (int i = 1; i <= get_ordem(); i++)
00186         {
00187             if (get_vizinhos(i) < get_ordem() - 1)
00188                 return false;
00189         }
00190         return true;
00191     }
00192
00193     void dfs(int vertice, bool visitado[]) {
00194         visitado[vertice] = true;
00195         for (int i = 1; i <= ordem; i++) {
00196             if (get_aresta(vertice, i) != -1 && !visitado[i]) {
00197                 dfs(i, visitado);
00198             }
00199         }
00200     }
00201
00206     int n_conexo() {
00207         bool* visitado = new bool[ordem + 1];
00208         for (int i = 1; i <= ordem; i++) {
00209             visitado[i] = false;
00210         }
00211
00212         int componentes = 0;
00213         for (int i = 1; i <= ordem; i++) {
00214             if (!visitado[i]) {
00215                 dfs(i, visitado);
00216                 componentes++;
00217             }
00218         }
00219
00220         delete[] visitado;
00221         return componentes;
00222     }
00223
00227     void maior_menor_distancia()
00228     {
00229         int n = get_ordem();
00230         if (n == 0)
00231         {
00232             cout « "O grafo está vazio." « endl;
00233             return;
00234         }
00235
00236         int dist[n + 1][n + 1];
00237
00238         for (int i = 1; i <= n; i++)
00239         {
00240             for (int j = 1; j <= n; j++)
00241             {
00242                 if (i == j)
00243                     dist[i][j] = 0;
00244                 else
00245                 {
00246                     int peso = get_aresta(i, j);
00247                     dist[i][j] = (peso > 0) ? peso : 999999;
00248                 }
00249             }
00250         }
00251
00252         for (int k = 1; k <= n; k++)
00253         {
00254             for (int i = 1; i <= n; i++)
00255             {
00256                 for (int j = 1; j <= n; j++)
00257                 {
00258                     if (dist[i][k] != 999999 && dist[k][j] != 999999)
00259                     {
00260                         if (dist[i][j] > dist[i][k] + dist[k][j])
00261                             dist[i][j] = dist[i][k] + dist[k][j];
00262                     }
00263                 }
00264             }
00265         }
00266     }
00267 };
00268
00269 #endif // GRAFO_H_INCLUDED
```

5.5 include/GrafoLista.h File Reference

```
#include "Grafo.h"
#include "ListaEncadeada.h"
#include "VerticeEncadeado.h"
#include "ArestaEncadeada.h"
#include <iostream>
```

Classes

- class GrafoLista

  Implementação de um grafo usando listas encadeadas.

5.6 GrafoLista.h

Go to the documentation of this file.

```
00001 #ifndef GRAFOLISTA_H_INCLUDED
00002 #define GRAFOLISTA_H_INCLUDED
00003
00004 #include "Grafo.h"
00005 #include "ListaEncadeada.h"
00006 #include "VerticeEncadeado.h"
00007 #include "ArestaEncadeada.h"
00008
00009 #include <iostream>
00010
00011 using namespace std;
00012
00020 class GrafoLista : public Grafo
00021 {
00022 private:
00023     ListaEncadeada<VerticeEncadeado> *vertices;
00024     ListaEncadeada<ArestaEncadeada> *arestas;
00025
00031     VerticeEncadeado *get_vertice_encadeado(int id);
00032
00038     void buscaEmProfundidade(VerticeEncadeado *vertice, bool *visitados);
00039
00040 public:
00046     GrafoLista();
00047
00053     int get_vertice(int id) override;
00054
00061     int get_aresta(int idOrigem, int idDestino) override;
00062
00068     void set_vertice(int id, float peso) override;
00069
00076     void set_aresta(int origem, int destino, float peso) override;
00077
00084     void nova_aresta(int origem, int destino, int peso) override;
00085
00091     void deleta_aresta(int vertice1, int vertice2) override;
00092
00098     int get_vizinhos(int vertice) override;
00099
00104     void novo_no(int peso) override;
00105
00110     void deleta_no(int vertice) override;
00111
00115     void imprimir();
00116
00122     ~GrafoLista();
00123 };
00124
00125 #endif // GRAFOLISTA_H_INCLUDED
```

5.7 include/GrafoMatriz.h File Reference

```
#include "Grafo.h"
#include <string>
```

Classes

- class GrafoMatriz

  Implementação de um grafo utilizando matriz de adjacência.

Variables

- const int TAMANHO_INICIAL = 10

  Define o tamanho inicial da matriz de adjacência.

5.7.1 Variable Documentation

5.7.1.1 TAMANHO_INICIAL

```
const int TAMANHO_INICIAL = 10
```

Define o tamanho inicial da matriz de adjacência.

5.8 GrafoMatriz.h

Go to the documentation of this file.

```
00001 #ifndef GRAFO_MATRIZ_H_INCLUDED
00002 #define GRAFO_MATRIZ_H_INCLUDED
00003
00004 #include "Grafo.h"
00005 #include <string>
00006
00007 using namespace std;
00008
00009 const int TAMANHO_INICIAL = 10;
00010
00018 class GrafoMatriz : public Grafo {
00019 private:
00020     int** Matriz;
00021     int* MatrizLinear;
00022     int* VetorPesosVertices;
00023     int tamanhoAtual;
00024     int tamanhoAtualLinear;
00025
00026 public:
00032     GrafoMatriz();
00033
00039     virtual ~GrafoMatriz();
00040
00044     void redimensionarMatriz();
00045
00049     void redimensionarMatrizLinear();
00050
00057     int calcularIndiceLinear(int origem, int destino);
00058
00065     int get_aresta(int origem, int destino) override;
00066
00072     int get_vertice(int vertice) override;
00073
```

```
00079     int get_vizinhos(int vertice) override;
00080
00086     void set_vertice(int id, float peso) override;
00087
00094     void set_aresta(int origem, int destino, float peso) override;
00095
00102     void nova_aresta(int origem, int destino, int peso);
00103
00108     void novo_no(int peso) override;
00109
00114     void deleta_no(int vertice) override;
00115
00120     void deleta_arestas_direcionadas(int vertice);
00121
00126     void deleta_arestas_nao_direcionadas(int vertice);
00127
00132     void reorganiza_matriz(int vertice);
00133
00138     void reorganiza_vetor_pesos(int vertice);
00139
00145     void deleta_aresta(int vertice1, int vertice2) override;
00146
00147 };
00148
00149 #endif  // GRAFO_MATRIZ_H_INCLUDED
```

5.9 include/ListaEncadeada.h File Reference

`#include <iostream>`

Classes

- class ListaEncadeada< T >

  Implementação genérica de uma lista encadeada.

5.10 ListaEncadeada.h

Go to the documentation of this file.

```
00001 #ifndef LISTAENCADEADA_H_INCLUDED
00002 #define LISTAENCADEADA_H_INCLUDED
00003
00004 #include <iostream>
00005
00006 using namespace std;
00007
00017 template <typename T>
00018 class ListaEncadeada {
00019 private:
00020     T* primeiro;
00021     T* ultimo;
00022     int tamanho;
00023
00024 public:
00030     ListaEncadeada() : primeiro(nullptr), ultimo(nullptr), tamanho(0) {}
00031
00036     T* getInicio() const {
00037         return primeiro;
00038     }
00039
00044     void setInicio(T* novoInicio) {
00045         primeiro = novoInicio;
00046         if (novoInicio == nullptr) {
00047             ultimo = nullptr;
00048         }
00049     }
00050
00055     void adicionar(T* novoNo) {
00056         if (primeiro == nullptr) {
```

```
00057             primeiro = novoNo;
00058             ultimo = novoNo;
00059         } else {
00060             ultimo->setProximo(novoNo);
00061             ultimo = novoNo;
00062         }
00063         tamanho++;
00064     }
00065
00069     void imprimir() const {
00070         T* atual = primeiro;
00071         while (atual != nullptr) {
00072             cout << *atual << endl;
00073             atual = atual->getProximo();
00074         }
00075     }
00076
00081     void remover(T* noParaRemover) {
00082         if (!primeiro || !noParaRemover) {
00083             return;
00084         }
00085         if (primeiro == noParaRemover) {
00086             primeiro = primeiro->getProximo();
00087             if (!primeiro) {
00088                 ultimo = nullptr;
00089             }
00090             tamanho--;
00091             delete noParaRemover;
00092             return;
00093         }
00094
00095         T* atual = primeiro;
00096         while (atual->getProximo() && atual->getProximo() != noParaRemover) {
00097             atual = atual->getProximo();
00098         }
00099
00100         if (atual->getProximo() == noParaRemover) {
00101             atual->setProximo(noParaRemover->getProximo());
00102
00103             if (noParaRemover == ultimo) {
00104                 ultimo = atual;
00105             }
00106             tamanho--;
00107             delete noParaRemover;
00108         }
00109     }
00110
00115     int get_tamanho() {
00116         return tamanho;
00117     }
00118
00124     ~ListaEncadeada() {
00125         T* atual = primeiro;
00126         while (atual != nullptr) {
00127             T* proximo = atual->getProximo();
00128             delete atual;
00129             atual = proximo;
00130         }
00131     }
00132 };
00133
00134 #endif // LISTAENCADEADA_H_INCLUDED
```

5.11 include/VerticeEncadeado.h File Reference

```
#include <iostream>
#include "ListaEncadeada.h"
#include "ArestaEncadeada.h"
```

Classes

- class VerticeEncadeado

  Representa um vértice em uma estrutura de grafo utilizando listas encadeadas.

5.12 VerticeEncadeado.h

Go to the documentation of this file.

```
00001 #ifndef VERTICEENCADEADO_H_INCLUDED
00002 #define VERTICEENCADEADO_H_INCLUDED
00003
00004 #include <iostream>
00005 #include "ListaEncadeada.h"
00006 #include "ArestaEncadeada.h"
00007
00015 class VerticeEncadeado {
00016 private:
00017     int id;
00018     int peso;
00019     int grau;
00020     VerticeEncadeado* proximo;
00021     ListaEncadeada<ArestaEncadeada>* conexoes;
00022
00023 public:
00029     VerticeEncadeado(int id, int peso);
00030
00035     int getId() const;
00036
00041     int getPeso() const;
00042
00047     int getGrau() const;
00048
00053     VerticeEncadeado* getProximo() const;
00054
00059     void setProximo(VerticeEncadeado* novoProximo);
00060
00066     void setConexao(VerticeEncadeado* verticeDestino, int pesoAresta);
00067
00072     ArestaEncadeada* getPrimeiraConexao();
00073
00078     ListaEncadeada<ArestaEncadeada>* getConexoes();
00079
00084     void setConexoes(ListaEncadeada<ArestaEncadeada>* novasConexoes);
00085
00092     friend std::ostream& operator<<(std::ostream& os, const VerticeEncadeado& vertice);
00093
00099     int removeConexao(VerticeEncadeado* destino);
00100
00107     ArestaEncadeada* getConexao(int origem, int destino);
00108 };
00109
00110 #endif // VERTICEENCADEADO_H_INCLUDED
```

5.13 src/ArestaEncadeada.cpp File Reference

```
#include "../include/ArestaEncadeada.h"
#include "../include/VerticeEncadeado.h"
#include <iostream>
```

Functions

- std::ostream & operator<< (std::ostream &os, const ArestaEncadeada &aresta)

  Sobrecarga do operador de saída para exibir informações da aresta.

5.13.1 Function Documentation

5.13.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const ArestaEncadeada & aresta)
```

Sobrecarga do operador de saída para exibir informações da aresta.

Sobrecarga do operador de saída para imprimir a aresta.

Exibe a origem, o destino e o peso da aresta.

Parameters

os	Fluxo de saída.
aresta	Aresta a ser exibida.

Returns

Fluxo de saída atualizado.

5.14 src/GrafoLista.cpp File Reference

```
#include <iostream>
#include <fstream>
#include "../include/GrafoLista.h"
```

5.15 src/GrafoMatriz.cpp File Reference

```
#include "../include/GrafoMatriz.h"
#include "../include/Grafo.h"
#include <iostream>
#include <fstream>
#include <cmath>
#include <cstdlib>
#include <ctime>
```

5.16 src/VerticeEncadeado.cpp File Reference

```
#include "../include/VerticeEncadeado.h"
```

Functions

- std::ostream & operator<< (std::ostream &os, const VerticeEncadeado &vertice)

  Sobrecarga do operador de saída para exibir informações do vértice.

5.16.1 Function Documentation

5.16.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const VerticeEncadeado & vertice)
```

Sobrecarga do operador de saída para exibir informações do vértice.

Exibe o identificador, peso e grau do vértice, bem como suas conexões.

Parameters

os	Fluxo de saída.
vertice	Vértice a ser exibido.

Returns

Fluxo de saída atualizado.