Teoria dos Grafos

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

`#include <ArestaEncadeada.h>`

Public Member Functions

- ArestaEncadeada (VerticeEncadeado *origem, VerticeEncadeado *destino, float peso)
- VerticeEncadeado * getOrigem () const
- VerticeEncadeado * getDestino () const
- float getPeso () const
- ArestaEncadeada * getProximo () const
- void setProximo (ArestaEncadeada *novoProximo)

Friends

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

4.1.1 Constructor & Destructor Documentation

4.1.1.1 ArestaEncadeada()

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

4.1.2 Member Function Documentation

4.1.2.1 getDestino()

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

4.1.2.2 getOrigem()

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

4.1.2.3 getPeso()

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

4.1.2.4 getProximo()

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

4.1.2.5 setProximo()

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

4.1.3 Friends And Related Symbol Documentation

4.1.3.1 operator<<

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

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

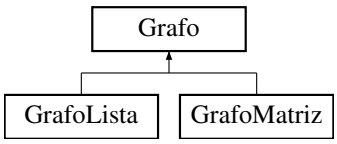
- include/ArestaEncadeada.h
- src/ArestaEncadeada.cpp

4.2 Grafo Class Reference

Classe base para a representação de um grafo.

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

Inheritance diagram for Grafo:



Public Member Functions

- Grafo ()=default

  Construtor padrão.
- virtual ~Grafo ()=default

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

  Método abstrato para obter o peso de uma aresta entre dois vértices.
- virtual int get_vertice (int vertice)=0

  Método abstrato para obter o peso de um vértice.
- virtual int get_vizinhos (int vertice)=0

  Método abstrato para obter o número de vizinhos de um vértice.
- virtual void nova_aresta (int origem, int destino, int peso)=0

  Método abstrato para adicionar uma nova aresta entre dois vértices.
- virtual void set_aresta (int origem, int destino, float peso)=0

  Método abstrato para definir o peso de uma aresta.
- virtual void set_vertice (int id, float peso)=0

  Método abstrato para definir o peso de um vértice.
- int get_ordem ()

  Obtém o número de vértices no grafo (ordem do grafo).
- void set_ordem (int ordem)

  Define o número de vértices do grafo.
- void aumenta_ordem ()

  Aumenta a ordem do grafo em 1.
- bool eh_direcionado ()

  Verifica se o grafo é direcionado.
- void set_eh_direcionado (bool direcionado)

  Define se o grafo é direcionado.
- bool vertice_ponderado ()

  Verifica se os vértices do grafo são ponderados.
- void set_vertice_ponderado (bool verticePonderado)

  Define se os vértices do grafo são ponderados.
- bool aresta_ponderada ()

  Verifica se as arestas do grafo são ponderadas.
- void set_aresta_ponderada (bool arestaPonderada)

  Define se as arestas do grafo são ponderadas.
- void carrega_grafo ()

  Carrega o grafo a partir de um arquivo.
- void carrega_grafo2 ()

  Carrega o grafo a partir de um arquivo com outro formato.
- int get_grau ()

  Obtém o grau (número de vizinhos) do vértice com maior grau.
- bool eh_completo ()

  Verifica se o grafo é completo.
- void dfs (int vertice, bool visitado[ ])

  Realiza uma busca em profundidade (DFS) para explorar todos os vértices conectados a partir de um vértice inicial.
- int n_conexo ()

  Calcula o número de componentes conexas do grafo.
- virtual void inicializa_grafo ()=0

  Método abstrato para inicializar o grafo, implementado pelas classes derivadas.
- void maior_menor_distancia ()

  Encontra a maior menor distância entre os vértices utilizando o algoritmo de Floyd-Warshall.

4.2.1 Detailed Description

Classe base para a representação de um grafo.

Essa classe define as operações gerais que podem ser realizadas em grafos, como manipulação de arestas, vértices, grau, e conexões. As subclasses precisam implementar a inicialização do grafo e operações de manipulação específicas.

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]
```

Destruidor virtual.

4.2.3 Member Function Documentation

4.2.3.1 aresta_ponderada()

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

Verifica se as arestas do grafo são ponderadas.

Returns

Verdadeiro se as arestas forem ponderadas, falso caso contrário.

4.2.3.2 aumenta_ordem()

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

Aumenta a ordem do grafo em 1.

4.2.3.3 carrega_grafo()

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

Carrega o grafo a partir de um arquivo.

4.2.3.4 carrega_grafo2()

```
void Grafo::carrega_grafo2 () [inline]
```

Carrega o grafo a partir de um arquivo com outro formato.

4.2.3.5 dfs()

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

Realiza uma busca em profundidade (DFS) para explorar todos os vértices conectados a partir de um vértice inicial.

Parameters

vertice	Vértice de origem para a busca.
visitado	Array de visitados para marcar os vértices já visitados.

4.2.3.6 eh_completo()

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

Verifica se o grafo é completo.

Returns

Verdadeiro se o grafo for completo (todos os vértices estão conectados entre si), falso caso contrário.

4.2.3.7 eh_direcionado()

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

Verifica se o grafo é direcionado.

Returns

Verdadeiro se o grafo for direcionado, falso caso contrário.

4.2.3.8 get_aresta()

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

Método abstrato para obter o peso de uma aresta entre dois vértices.

Parameters

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

Returns

O peso da aresta.

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.9 get_grau()

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

Obtém o grau (número de vizinhos) do vértice com maior grau.

Returns

O grau máximo.

4.2.3.10 get_ordem()

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

Obtém o número de vértices no grafo (ordem do grafo).

Returns

O número de vértices do grafo.

4.2.3.11 get_vertice()

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

Método abstrato para obter o peso de um vértice.

Parameters

vertice	O identificador do vértice.

Returns

O peso do vértice.

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.12 get_vizinhos()

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

Método abstrato para obter o número de vizinhos de um vértice.

Parameters

vertice	O identificador do vértice.

Returns

O número de vizinhos.

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.13 inicializa_grafo()

```
virtual void Grafo::inicializa_grafo ()  [pure virtual]
```

Método abstrato para inicializar o grafo, implementado pelas classes derivadas.

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.14 maior_menor_distancia()

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

Encontra a maior menor distância entre os vértices utilizando o algoritmo de Floyd-Warshall.

O algoritmo calcula todas as menores distâncias entre pares de vértices e depois encontra a maior distância entre qualquer par de vértices. Caso não haja caminho entre os vértices, o algoritmo considerará um valor de infinito para aquelas distâncias.

4.2.3.15 n_conexo()

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

Calcula o número de componentes conexas do grafo.

Returns

O número de componentes conexas no grafo.

4.2.3.16 nova_aresta()

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

Método abstrato para adicionar uma nova aresta entre dois vértices.

Parameters

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

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.17 set_aresta()

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

Método abstrato para definir o peso de uma aresta.

Parameters

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

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.18 set_aresta_ponderada()

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

Define se as arestas do grafo são ponderadas.

Parameters

arestaPonderada	Valor que define se as arestas serão ponderadas.

4.2.3.19 set_eh_direcionado()

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

Define se o grafo é direcionado.

Parameters

direcionado	Valor que define se o grafo será direcionado.

4.2.3.20 set_ordem()

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

Define o número de vértices do grafo.

Parameters

ordem	O número de vértices a ser definido.

4.2.3.21 set_vertice()

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

Método abstrato para definir o peso de um vértice.

Parameters

id	Identificador do vértice.
peso	Peso do vértice.

Implemented in GrafoLista, and GrafoMatriz.

4.2.3.22 set_vertice_ponderado()

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

Define se os vértices do grafo são ponderados.

Parameters

verticePonderado	Valor que define se os vértices serão ponderados.

4.2.3.23 vertice_ponderado()

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

Verifica se os vértices do grafo são ponderados.

Returns

Verdadeiro se os vértices forem ponderados, falso caso contrário.

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

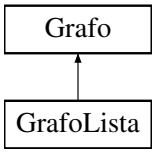
- include/Grafo.h

4.3 GrafoLista Class Reference

A classe GrafoLista é uma implementação de grafo que utiliza listas encadeadas para armazenar os vértices e arestas. Ela herda da classe abstrata Grafo e implementa suas funções virtuais para manipulação de vértices e arestas.

```
#include <GrafoLista.h>
```

Inheritance diagram for GrafoLista:



Public Member Functions

- GrafoLista ()

  Construtor da classe GrafoLista. Inicializa as listas de vértices e arestas.
- int get_vertice (int id) override

  Método para obter o peso de um vértice dado seu id.
- int get_aresta (int idOrigem, int idDestino) override

  Método para obter o peso de uma aresta entre dois vértices dados seus ids.
- void set_vertice (int id, float peso) override

  Método para definir o peso de um vértice dado seu id.
- void set_aresta (int origem, int destino, float peso) override

  Método para definir o peso de uma aresta entre dois vértices dados seus ids.
- void nova_aresta (int origem, int destino, int peso) override

  Método para adicionar uma nova aresta entre dois vértices. Verifica se a aresta já existe e, caso contrário, adiciona a aresta na lista.
- int get_vizinhos (int vertice) override

  Método para obter o número de vizinhos de um vértice.
- void imprimir ()

  Método para imprimir os vértices e as arestas do grafo, além de informações sobre o grau e componentes conexas.
- void inicializa_grafo () override

  Método para inicializar o grafo a partir de um arquivo de entrada. Lê as informações de vértices, arestas, pesos e outras configurações.
- ~GrafoLista ()

  Destruidor da classe GrafoLista. Libera a memória alocada para as listas de vértices e arestas.

Public Member Functions inherited from Grafo

- Grafo ()=default

  Construtor padrão.
- virtual ~Grafo ()=default

  Destruidor virtual.
- int get_ordem ()

  Obtém o número de vértices no grafo (ordem do grafo).
- void set_ordem (int ordem)

  Define o número de vértices do grafo.
- void aumenta_ordem ()

  Aumenta a ordem do grafo em 1.
- bool eh_direcionado ()

  Verifica se o grafo é direcionado.
- void set_eh_direcionado (bool direcionado)

  Define se o grafo é direcionado.
- bool vertice_ponderado ()

  Verifica se os vértices do grafo são ponderados.
- void set_vertice_ponderado (bool verticePonderado)

  Define se os vértices do grafo são ponderados.
- bool aresta_ponderada ()

  Verifica se as arestas do grafo são ponderadas.
- void set_aresta_ponderada (bool arestaPonderada)

  Define se as arestas do grafo são ponderadas.
- void carrega_grafo ()

  Carrega o grafo a partir de um arquivo.

- void carrega_grafo2 ()
  Carrega o grafo a partir de um arquivo com outro formato.
- int get_grau ()
  Obtém o grau (número de vizinhos) do vértice com maior grau.
- bool eh_completo ()
  Verifica se o grafo é completo.
- void dfs (int vertice, bool visitado[])
  Realiza uma busca em profundidade (DFS) para explorar todos os vértices conectados a partir de um vértice inicial.
- int n_conexo ()
  Calcula o número de componentes conexas do grafo.
- void maior_menor_distancia ()
  Encontra a maior menor distância entre os vértices utilizando o algoritmo de Floyd-Warshall.

4.3.1 Detailed Description

A classe GrafoLista é uma implementação de grafo que utiliza listas encadeadas para armazenar os vértices e arestas. Ela herda da classe abstrata Grafo e implementa suas funções virtuais para manipulação de vértices e arestas.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 GrafoLista()

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

Construtor da classe GrafoLista. Inicializa as listas de vértices e arestas.

4.3.2.2 ~GrafoLista()

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

Destruidor da classe GrafoLista. Libera a memória alocada para as listas de vértices e arestas.

4.3.3 Member Function Documentation

4.3.3.1 get_aresta()

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

Método para obter o peso de uma aresta entre dois vértices dados seus ids.

Parameters

idOrigem	O id do vértice de origem.
idDestino	O id do vértice de destino.

Returns

O peso da aresta entre os vértices de origem e destino.

Implements Grafo.

4.3.3.2 get_vertice()

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

Método para obter o peso de um vértice dado seu id.

Parameters

id	O id do vértice.

Returns

O peso do vértice correspondente ao id.

Implements Grafo.

4.3.3.3 get_vizinhos()

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

Método para obter o número de vizinhos de um vértice.

Parameters

vertice	O id do vértice.

Returns

O número de vizinhos do vértice.

Implements Grafo.

4.3.3.4 imprimir()

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

Método para imprimir os vértices e as arestas do grafo, além de informações sobre o grau e componentes conexas.

4.3.3.5 inicializa_grafo()

```
void GrafoLista::inicializa_grafo ()  [override], [virtual]
```

Método para inicializar o grafo a partir de um arquivo de entrada. Lê as informações de vértices, arestas, pesos e outras configurações.

Implements Grafo.

4.3.3.6 nova_aresta()

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

Método para adicionar uma nova aresta entre dois vértices. Verifica se a aresta já existe e, caso contrário, adiciona a aresta na lista.

Parameters

origem	O id do vértice de origem.
destino	O id do vértice de destino.
peso	O peso da nova aresta.

Implements Grafo.

4.3.3.7 set_aresta()

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

Método para definir o peso de uma aresta entre dois vértices dados seus ids.

Parameters

origem	O id do vértice de origem.
destino	O id do vértice de destino.
peso	O peso a ser atribuído à aresta.

Implements Grafo.

4.3.3.8 set_vertice()

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

Método para definir o peso de um vértice dado seu id.

Parameters

id	O id do vértice.
peso	O peso a ser atribuído ao vértice.

Implements Grafo.

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

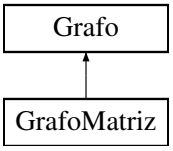
- include/GrafoLista.h
- src/GrafoLista.cpp

4.4 GrafoMatriz Class Reference

A classe GrafoMatriz implementa a interface da classe abstrata Grafo utilizando uma matriz de adjacência. A classe gerencia tanto grafos direcionados quanto não direcionados, além de permitir a manipulação de pesos de vértices e arestas.

`#include <GrafoMatriz.h>`

Inheritance diagram for GrafoMatriz:



Public Member Functions

- GrafoMatriz ()

  Construtor da classe GrafoMatriz. Inicializa as matrizes e vetores necessários.
- virtual ~GrafoMatriz ()

  Destruidor da classe GrafoMatriz. Libera a memória alocada para as matrizes e vetores.
- void redimensionarMatriz ()

  Método para redimensionar a matriz 2D de adjacência.
- void redimensionarMatrizLinear ()

  Método para redimensionar a matriz linear de adjacência.
- void inicializa_grafo ()

  Método para inicializar o grafo a partir de um arquivo de entrada. Lê os vértices, arestas e pesos do arquivo de dados.
- int calcularIndiceLinear (int origem, int destino)

  Calcula o índice linear na matriz comprimida para grafos não direcionados.
- int get_aresta (int origem, int destino) override

  Método para obter o peso de uma aresta entre dois vértices dados seus ids.
- int get_vertice (int vertice) override

  Método para obter o peso de um vértice dado seu id.
- int get_vizinhos (int vertice) override

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

  Método para definir o peso de um vértice dado seu id.
- void set_aresta (int origem, int destino, float peso) override

  Método para definir o peso de uma aresta entre dois vértices dados seus ids.
- void nova_aresta (int origem, int destino, int peso)

  Método para adicionar uma nova aresta entre dois vértices, verificando se a aresta já existe.

Public Member Functions inherited from Grafo

- Grafo ()=default

  Construtor padrão.
- virtual ~Grafo ()=default

  Destruidor virtual.
- int get_ordem ()

  Obtém o número de vértices no grafo (ordem do grafo).
- void set_ordem (int ordem)

  Define o número de vértices do grafo.
- void aumenta_ordem ()

  Aumenta a ordem do grafo em 1.
- bool eh_direcionado ()

  Verifica se o grafo é direcionado.
- void set_eh_direcionado (bool direcionado)

  Define se o grafo é direcionado.
- bool vertice_ponderado ()

  Verifica se os vértices do grafo são ponderados.
- void set_vertice_ponderado (bool verticePonderado)

  Define se os vértices do grafo são ponderados.
- bool aresta_ponderada ()

  Verifica se as arestas do grafo são ponderadas.
- void set_aresta_ponderada (bool arestaPonderada)

  Define se as arestas do grafo são ponderadas.
- void carrega_grafo ()

  Carrega o grafo a partir de um arquivo.
- void carrega_grafo2 ()

  Carrega o grafo a partir de um arquivo com outro formato.
- int get_grau ()

  Obtém o grau (número de vizinhos) do vértice com maior grau.
- bool eh_completo ()

  Verifica se o grafo é completo.
- void dfs (int vertice, bool visitado[ ])

  Realiza uma busca em profundidade (DFS) para explorar todos os vértices conectados a partir de um vértice inicial.
- int n_conexo ()

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

  Encontra a maior menor distância entre os vértices utilizando o algoritmo de Floyd-Warshall.

4.4.1 Detailed Description

A classe GrafoMatriz implementa a interface da classe abstrata Grafo utilizando uma matriz de adjacência. A classe gerencia tanto grafos direcionados quanto não direcionados, além de permitir a manipulação de pesos de vértices e arestas.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 GrafoMatriz()

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

Construtor da classe GrafoMatriz. Inicializa as matrizes e vetores necessários.

4.4.2.2 ~GrafoMatriz()

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

Destruidor da classe GrafoMatriz. Libera a memória alocada para as matrizes e vetores.

4.4.3 Member Function Documentation

4.4.3.1 calcularIndiceLinear()

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

Calcula o índice linear na matriz comprimida para grafos não direcionados.

Parameters

origem	O id do vértice de origem.
destino	O id do vértice de destino.

Returns

O índice linear correspondente aos vértices de origem e destino.

4.4.3.2 get_aresta()

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

Método para obter o peso de uma aresta entre dois vértices dados seus ids.

Parameters

origem	O id do vértice de origem.
destino	O id do vértice de destino.

Returns

O peso da aresta entre os vértices de origem e destino.

Implements Grafo.

4.4.3.3 get_vertice()

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

Método para obter o peso de um vértice dado seu id.

Parameters

vertice	O id do vértice.

Returns

O peso do vértice correspondente ao id.

Implements Grafo.

4.4.3.4 get_vizinhos()

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

Método para obter o número de vizinhos de um vértice.

Parameters

vertice	O id do vértice.

Returns

O número de vizinhos do vértice.

Implements Grafo.

4.4.3.5 inicializa_grafo()

```
void GrafoMatriz::inicializa_grafo ()  [virtual]
```

Método para inicializar o grafo a partir de um arquivo de entrada. Lê os vértices, arestas e pesos do arquivo de dados.

Implements Grafo.

4.4.3.6 nova_aresta()

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

Método para adicionar uma nova aresta entre dois vértices, verificando se a aresta já existe.

Parameters

origem	O id do vértice de origem.
destino	O id do vértice de destino.
peso	O peso da nova aresta.

Implements Grafo.

4.4.3.7 redimensionarMatriz()

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

Método para redimensionar a matriz 2D de adjacência.

4.4.3.8 redimensionarMatrizLinear()

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

Método para redimensionar a matriz linear de adjacência.

4.4.3.9 set_aresta()

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

Método para definir o peso de uma aresta entre dois vértices dados seus ids.

Parameters

origem	O id do vértice de origem.
destino	O id do vértice de destino.
peso	O peso a ser atribuído à aresta.

Implements Grafo.

4.4.3.10 set_vertice()

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

Método para definir o peso de um vértice dado seu id.

Parameters

id	O id do vértice.
peso	O peso a ser atribuído ao 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

A classe ListaEncadeada é uma implementação genérica de uma lista encadeada, capaz de armazenar elementos de qualquer tipo. Esta classe fornece métodos para adicionar, remover e imprimir elementos da lista, além de acessar o primeiro e o último elemento.

```
#include <ListaEncadeada.h>
```

Public Member Functions

- ListaEncadeada ()

  Construtor padrão da lista encadeada. Inicializa a lista com primeiro e último ponteiro nulos.
- T * getInicio () const

  Obtém o primeiro nó da lista.
- void adicionar (T *novoNo)

  Adiciona um novo nó ao final da lista.
- void imprimir () const

  Imprime todos os elementos da lista.
- void remover (T *noParaRemover)

  Remove um nó específico da lista.
- int get_tamanho ()

  Retorna o tamanho atual da lista (número de elementos armazenados).
- ~ListaEncadeada ()

  Destruidor da lista encadeada. Libera toda a memória alocada para os nós da lista.

4.5.1 Detailed Description

template<typename T>
class ListaEncadeada< T >

A classe ListaEncadeada é uma implementação genérica de uma lista encadeada, capaz de armazenar elementos de qualquer tipo. Esta classe fornece métodos para adicionar, remover e imprimir elementos da lista, além de acessar o primeiro e o último elemento.

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 padrão da lista encadeada. Inicializa a lista com primeiro e último ponteiro nulos.

4.5.2.2 ~ListaEncadeada()

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

Destruidor da lista encadeada. Libera toda a memória alocada para os nós da 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 nó a ser adicionado.

4.5.3.2 get_tamanho()

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

Retorna o tamanho atual da lista (número de elementos armazenados).

Returns

O 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

O ponteiro para o primeiro nó.

4.5.3.4 imprimir()

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

Imprime todos os elementos da lista.

Note

A impressão é feita chamando o operador << do tipo T para cada elemento da lista.

4.5.3.5 remover()

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

Remove um nó específico da lista.

Parameters

noParaRemover	Ponteiro para o nó a ser removido.

Note

Se o nó não for encontrado ou a lista estiver vazia, a operação não será realizada.

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

- include/ListaEncadeada.h

4.6 VerticeEncadeado Class Reference

A classe VerticeEncadeado representa um vértice em um grafo implementado usando uma lista encadeada. Ela armazena informações sobre o vértice, como seu identificador, peso e grau, e as conexões (arestas) com outros vértices.

`#include <VerticeEncadeado.h>`

Public Member Functions

- VerticeEncadeado (int id, int peso)
  Construtor que cria um vértice com um identificador e peso especificados.
- int getId () const
  Obtém o identificador do vértice.
- int getPeso () const
  Obtém o peso do vértice.
- int getGrau () const
  Obtém o grau do vértice, que é o número de conexões (arestas) que ele possui.
- 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)
  Estabelece uma conexão entre este vértice e outro vértice, criando uma aresta.
- ArestaEncadeada * getPrimeiraConexao ()
  Obtém a primeira conexão (aresta) do vértice.
- ListaEncadeada< ArestaEncadeada > * getConexoes ()
  Obtém a lista de todas as conexões (arestas) do vértice.
- void setConexoes (ListaEncadeada< ArestaEncadeada > *novasConexoes)
  Define as conexões (arestas) do vértice.
- int removeConexao (VerticeEncadeado *destino)
  Remove uma conexão (aresta) entre este vértice e outro vértice de destino.
- ArestaEncadeada * getConexao (int origem, int destino)
  Obtém uma conexão (aresta) específica entre dois vértices.

Friends

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

  Sobrecarga do operador de saída, permitindo a impressão do vértice no formato desejado.

4.6.1 Detailed Description

A classe VerticeEncadeado representa um vértice em um grafo implementado usando uma lista encadeada. Ela armazena informações sobre o vértice, como seu identificador, peso e grau, e as conexões (arestas) com outros vértices.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 VerticeEncadeado()

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

Construtor que cria um vértice com um identificador e peso especificados.

Parameters

id	Identificador 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 (aresta) específica entre dois vértices.

Parameters

origem	O vértice de origem da aresta.
destino	O vértice de destino da aresta.

Returns

A aresta encontrada entre os dois vértices.

4.6.3.2 getConexoes()

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

Obtém a lista de todas as conexões (arestas) do vértice.

Returns

Ponteiro para a lista de arestas.

4.6.3.3 getGrau()

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

Obtém o grau do vértice, que é o número de conexões (arestas) que ele possui.

Returns

O grau do vértice.

4.6.3.4 getId()

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

Obtém o identificador do vértice.

Returns

O identificador do vértice.

4.6.3.5 getPeso()

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

Obtém o peso do vértice.

Returns

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.

4.6.3.7 getProximo()

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

Obtém o próximo vértice na lista encadeada.

Returns

O próximo vértice.

4.6.3.8 removeConexao()

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

Remove uma conexão (aresta) entre este vértice e outro vértice de destino.

Parameters

destino	O vértice de destino da aresta a ser removida.

Returns

O peso da aresta removida.

4.6.3.9 setConexao()

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

Estabelece uma conexão entre este vértice e outro vértice, criando uma aresta.

Parameters

verticeDestino	O vértice de destino da aresta.
pesoAresta	O peso da aresta.

4.6.3.10 setConexoes()

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

Define as conexões (arestas) do vértice.

Parameters

novasConexoes	Ponteiro para a nova lista de arestas.

4.6.3.11 setProximo()

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

Define o próximo vértice na lista encadeada.

Parameters

novoProximo	Ponteiro para o 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, permitindo a impressão do vértice no formato desejado.

Parameters

os	Fluxo de saída.
vertice	O vértice a ser impresso.

Returns

O fluxo de saída.

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

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 // A classe ArestaEncadeada representa uma aresta em um grafo, que conecta dois vértices (origem e
      destino)
00007 // e pode ter um peso associado. Esta classe é usada para representar arestas encadeadas em um grafo
      baseado
00008 // em listas encadeadas.
00009 class VerticeEncadeado;  // Declaração antecipada da classe VerticeEncadeado (classe de vértices no
      grafo)
00010
00011 class ArestaEncadeada {
00012 private:
00013     VerticeEncadeado* origem;   // Ponteiro para o vértice de origem da aresta
00014     VerticeEncadeado* destino;  // Ponteiro para o vértice de destino da aresta
00015     float peso;                 // Peso da aresta (pode ser 0 para arestas não ponderadas)
00016     ArestaEncadeada* proximo;  // Ponteiro para a próxima aresta, caso haja uma lista de arestas
      encadeadas
00017
00018 public:
00019     // Construtor que inicializa os valores da aresta (origem, destino e peso).
00020     // O próximo ponteiro é inicializado como nullptr (sem aresta subsequente).
00021     ArestaEncadeada(VerticeEncadeado* origem, VerticeEncadeado* destino, float peso);
00022
00023     // Métodos de acesso (getters) para os membros privados da classe.
00024
00025     // Retorna o vértice de origem da aresta.
00026     VerticeEncadeado* getOrigem() const;
00027
00028     // Retorna o vértice de destino da aresta.
00029     VerticeEncadeado* getDestino() const;
00030
00031     // Retorna o peso da aresta.
```

```
00032     float getPeso() const;
00033
00034     // Retorna o ponteiro para a próxima aresta na lista de arestas encadeadas.
00035     ArestaEncadeada* getProximo() const;
00036
00037     // Define o ponteiro da próxima aresta na lista encadeada de arestas.
00038     void setProximo(ArestaEncadeada* novoProximo);
00039
00040     // Sobrecarga do operador de fluxo («) para permitir a impressão das arestas
00041     // A impressão inclui as informações de origem, destino e peso da aresta.
00042     friend std::ostream& operator«(std::ostream& os, const ArestaEncadeada& aresta);
00043 };
00044
00045 #endif // ARESTAENCADEADA_H_INCLUDED
```

5.3 include/Grafo.h File Reference

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

Classes

- class Grafo

  Classe base para a representação de 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;
00023     int destino;
00024     int peso;
00025
00026 public:
00030     Grafo() = default;
00031
00035     virtual ~Grafo() = default;
00036
00044     virtual int get_aresta(int origem, int destino) = 0;
00045
00052     virtual int get_vertice(int vertice) = 0;
00053
00060     virtual int get_vizinhos(int vertice) = 0;
00061
00069     virtual void nova_aresta(int origem, int destino, int peso) = 0;
00070
00078     virtual void set_aresta(int origem, int destino, float peso) = 0;
00079
00086     virtual void set_vertice(int id, float peso) = 0;
00087
00093     int get_ordem()
00094     {
00095         return ordem;
00096     };
00097
```

```
00103     void set_ordem(int ordem)
00104     {
00105         this->ordem = ordem;
00106     };
00107
00111     void aumenta_ordem()
00112     {
00113         this->ordem++;
00114     };
00115
00121     bool eh_direcionado()
00122     {
00123         return direcionado;
00124     }
00125
00131     void set_eh_direcionado(bool direcionado)
00132     {
00133         this->direcionado = direcionado;
00134     };
00135
00141     bool vertice_ponderado()
00142     {
00143         return vtp;
00144     }
00145
00151     void set_vertice_ponderado(bool verticePonderado)
00152     {
00153         this->vtp = verticePonderado;
00154     };
00155
00161     bool aresta_ponderada()
00162     {
00163         return atp;
00164     }
00165
00171     void set_aresta_ponderada(bool arestaPonderada)
00172     {
00173         this->atp = arestaPonderada;
00174     };
00175
00179     void carrega_grafo()
00180     {
00181         ifstream arquivo("./entradas/Grafo.txt");
00182         if (!arquivo.is_open())
00183         {
00184             cerr « "Erro ao abrir o arquivo Grafo.txt" « endl;
00185             return;
00186         }
00187
00188         arquivo » ordem » direcionado » vtp » atp;
00189         set_ordem(ordem);
00190         set_eh_direcionado(direcionado);
00191         set_vertice_ponderado(vtp);
00192         set_aresta_ponderada(atp);
00193
00194         inicializa_grafo();
00195     }
00196
00200     void carrega_grafo2()
00201     {
00202         ifstream arquivo("./entradas/Grafo.txt");
00203         if (!arquivo.is_open())
00204         {
00205             cerr « "Erro ao abrir o arquivo Grafo.txt" « endl;
00206             return;
00207         }
00208
00209         arquivo » ordem » direcionado » vtp » atp;
00210         set_ordem(ordem);
00211         set_eh_direcionado(direcionado);
00212         set_vertice_ponderado(vtp);
00213         set_aresta_ponderada(atp);
00214
00215         for (int i = 1; i <= ordem; i++)
00216         {
00217             int peso_vertice;
00218             arquivo » peso_vertice;
00219
00220             if (vertice_ponderado())
00221                 set_vertice(i, peso_vertice);
00222             else
00223                 set_vertice(i, 1);
00224         }
00225
00226         int origem, destino = 1;
00227         float peso = 0;
00228
```

```
00229         while (arquivo » origem » destino » peso)
00230         {
00231             if(!aresta_ponderada())
00232                 peso = 0;
00233             set_aresta(origem, destino, peso);
00234         }
00235     }
00236
00242     int get_grau()
00243     {
00244         if (!eh_direcionado())
00245         {
00246             int grauMaximo = 0;
00247             for (int i = 1; i <= ordem; i++)
00248             {
00249                 int numVizinhos = get_vizinhos(i);
00250
00251                 if (numVizinhos > grauMaximo)
00252                 {
00253                     grauMaximo = numVizinhos;
00254                 }
00255             }
00256             return grauMaximo;
00257         }
00258         else
00259         {
00260             int maxGrauSaida = 0;
00261
00262             for (int i = 1; i <= ordem; i++)
00263             {
00264                 int grauSaida = 0;
00265
00266                 // Calcula grau de saída
00267                 int numVizinhos = get_vizinhos(i);
00268                 grauSaida = numVizinhos;
00269
00270                 if (grauSaida > maxGrauSaida)
00271                 {
00272                     maxGrauSaida = grauSaida;
00273                 }
00274             }
00275             return maxGrauSaida;
00276         }
00277     }
00278
00284     bool eh_completo()
00285     {
00286         for (int i = 1; i <= get_ordem(); i++)
00287         {
00288             if (get_vizinhos(i) < get_ordem() - 1)
00289                 return false;
00290         }
00291         return true;
00292     }
00293
00300     void dfs(int vertice, bool visitado[]) {
00301         visitado[vertice] = true;
00302         for (int i = 1; i <= ordem; i++) {
00303             if (get_aresta(vertice, i) && !visitado[i]) {
00304                 dfs(i, visitado);
00305             }
00306         }
00307     }
00308
00314     int n_conexo() {
00315         bool* visitado = new bool[ordem + 1]; // Usa alocação dinâmica para evitar problemas de
      tamanho
00316         for (int i = 1; i <= ordem; i++) { // Se os vértices começam em 1
00317             visitado[i] = false; // Inicializa corretamente
00318         }
00319
00320         int componentes = 0;
00321
00322         for (int i = 1; i <= ordem; i++) { // Se os vértices começam em 1
00323             if (!visitado[i]) { // Usa índice corretamente
00324                 dfs(i, visitado); // Chama a DFS
00325                 componentes++;
00326             }
00327         }
00328
00329         delete[] visitado; // Libera memória alocada dinamicamente
00330         return componentes;
00331     }
00332
00336     virtual void inicializa_grafo() = 0;
00337
00345     void maior_menor_distancia() {
```

```
00346         int n = get_ordem();
00347
00348         if (n == 0) {
00349             cout « "O grafo está vazio." « endl;
00350             return;
00351         }
00352
00353         // Matriz de distâncias
00354         int dist[n + 1][n + 1];
00355
00356         // Inicializa a matriz de distâncias
00357         for (int i = 1; i <= n; i++) {
00358             for (int j = 1; j <= n; j++) {
00359                 if (i == j) {
00360                     dist[i][j] = 0; // Distância de um nó para ele mesmo
00361                 } else {
00362                     int peso = get_aresta(i, j);
00363                     dist[i][j] = (peso > 0) ? peso : 999999; // Se não há aresta, assume um valor alto
      (infinito)
00364                 }
00365             }
00366         }
00367
00368         // Algoritmo de Floyd-Warshall
00369         for (int k = 1; k <= n; k++) {
00370             for (int i = 1; i <= n; i++) {
00371                 for (int j = 1; j <= n; j++) {
00372                     if (dist[i][k] != 999999 && dist[k][j] != 999999) {
00373                         if (dist[i][j] > dist[i][k] + dist[k][j]) {
00374                             dist[i][j] = dist[i][k] + dist[k][j];
00375                         }
00376                     }
00377                 }
00378             }
00379         }
00380
00381         // Encontrar os nós mais distantes
00382         int maxDist = -1;
00383         int no1 = -1, no2 = -1;
00384
00385         for (int i = 1; i <= n; i++) {
00386             for (int j = i + 1; j <= n; j++) {
00387                 if (dist[i][j] != 999999 && dist[i][j] > maxDist) {
00388                     maxDist = dist[i][j];
00389                     no1 = i;
00390                     no2 = j;
00391                 }
00392             }
00393         }
00394
00395         // Exibir resultado
00396         if (no1 != -1 && no2 != -1) {
00397             cout « "Maior menor distância: (" « no1 « "-" « no2 « ") " « maxDist « endl;
00398         } else {
00399             cout « "Não há caminho entre os nós." « endl;
00400         }
00401     }
00402 };
00403
00404 #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

  A classe GrafoLista é uma implementação de grafo que utiliza listas encadeadas para armazenar os vértices e arestas. Ela herda da classe abstrata Grafo e implementa suas funções virtuais para manipulação de vértices e arestas.

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
00017 class GrafoLista : public Grafo
00018 {
00019 private:
00023     ListaEncadeada<VerticeEncadeado> *vertices;
00024
00028     ListaEncadeada<ArestaEncadeada> *arestas;
00029
00036     VerticeEncadeado *get_vertice_encadeado(int id);
00037
00044     void buscaEmProfundidade(VerticeEncadeado *vertice, bool *visitados);
00045
00046 public:
00050     GrafoLista();
00051
00058     int get_vertice(int id) override;
00059
00067     int get_aresta(int idOrigem, int idDestino) override;
00068
00075     void set_vertice(int id, float peso) override;
00076
00084     void set_aresta(int origem, int destino, float peso) override;
00085
00094     void nova_aresta(int origem, int destino, int peso) override;
00095
00102     int get_vizinhos(int vertice) override;
00103
00107     void imprimir();
00108
00113     void inicializa_grafo() override;
00114
00118     ~GrafoLista();
00119 };
00120
00121 #endif // GRAFOLISTA_H_INCLUDED
```

5.7 include/GrafoMatriz.h File Reference

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

Classes

- class GrafoMatriz

  A classe GrafoMatriz implementa a interface da classe abstrata Grafo utilizando uma matriz de adjacência. A classe gerencia tanto grafos direcionados quanto não direcionados, além de permitir a manipulação de pesos de vértices e arestas.

Variables

- const int TAMANHO_INICIAL = 10

5.7.1 Variable Documentation

5.7.1.1 TAMANHO_INICIAL

```
const int TAMANHO_INICIAL = 10
```

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; // Começa com 10 vértices
00010
00015 class GrafoMatriz : public Grafo {
00016 private:
00020     int** Matriz;
00021
00025     int* MatrizLinear;
00026
00030     int* VetorPesosVertices;
00031
00035     int tamanhoAtual;
00036
00040     int tamanhoAtualLinear;
00041
00042 public:
00046     GrafoMatriz();
00047
00051     virtual ~GrafoMatriz();
00052
00056     void redimensionarMatriz();
00057
00061     void redimensionarMatrizLinear();
00062
00067     void inicializa_grafo();
00068
00076     int calcularIndiceLinear(int origem, int destino);
00077
00085     int get_aresta(int origem, int destino) override;
00086
00093     int get_vertice(int vertice) override;
00094
00101     int get_vizinhos(int vertice) override;
00102
00109     void set_vertice(int id, float peso) override;
00110
00118     void set_aresta(int origem, int destino, float peso) override;
00119
00127     void nova_aresta(int origem, int destino, int peso);
00128 };
00129
00130 #endif  // GRAFO_MATRIZ_H_INCLUDED
```

5.9 include/ListaEncadeada.h File Reference

```
#include <iostream>
```

Classes

- class ListaEncadeada< T >

  A classe ListaEncadeada é uma implementação genérica de uma lista encadeada, capaz de armazenar elementos de qualquer tipo. Esta classe fornece métodos para adicionar, remover e imprimir elementos da lista, além de acessar o primeiro e o último elemento.

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
00014 template <typename T>
00015
00016 class ListaEncadeada {
00017 private:
00021     T* primeiro;
00022
00026     T* ultimo;
00027
00031     int tamanho;
00032
00033 public:
00037     ListaEncadeada() : primeiro(nullptr), ultimo(nullptr), tamanho(0) {}
00038
00044     T* getInicio() const {
00045         return primeiro;
00046     }
00047
00053     void adicionar(T* novoNo) {
00054         if (primeiro == nullptr) {
00055             primeiro = novoNo;
00056             ultimo = novoNo;
00057         } else {
00058             ultimo->setProximo(novoNo);
00059             ultimo = novoNo;
00060         }
00061         tamanho++;
00062     }
00063
00069     void imprimir() const {
00070         T* atual = primeiro;
00071         while (atual != nullptr) {
00072             cout << *atual << endl;
00073             atual = atual->getProximo();
00074         }
00075     }
00076
00083     void remover(T* noParaRemover) {
00084         if(!primeiro || !noParaRemover) {
00085             return;
00086         }
00087         if(primeiro == noParaRemover) {
00088             primeiro = primeiro->getProximo();
00089             if (!primeiro) {
00090                 ultimo = nullptr;
00091             }
00092             tamanho--;
00093             delete noParaRemover;
00094             return;
00095         }
00096
00097         T* atual = primeiro;
00098         while(atual->getProximo() && atual->getProximo() != noParaRemover) {
00099             atual = atual->getProximo();
00100         }
00101
00102         if(atual->getProximo() == noParaRemover) {
00103             atual->setProximo(noParaRemover->getProximo());
00104
00105             if (noParaRemover == ultimo) {
00106                 ultimo = atual;
00107             }
00108             tamanho--;
00109             delete noParaRemover;
00110         }
00111     }
00112
00118     int get_tamanho() {
00119         return tamanho;
00120     }
00121
00125     ~ListaEncadeada() {
00126         T* atual = primeiro;
00127         while (atual != nullptr) {
00128             T* proximo = atual->getProximo();
00129             delete atual;
```

```
00130             atual = proximo;
00131         }
00132     }
00133 };
00134
00135 #endif // LISTAENCADEADA_H_INCLUDED
```

5.11 include/VerticeEncadeado.h File Reference

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

Classes

- class VerticeEncadeado

  A classe VerticeEncadeado representa um vértice em um grafo implementado usando uma lista encadeada. Ela armazena informações sobre o vértice, como seu identificador, peso e grau, e as conexões (arestas) com outros vértices.

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
00012 class VerticeEncadeado {
00013 private:
00017     int id;
00018
00022     int peso;
00023
00027     int grau;
00028
00032     VerticeEncadeado* proximo;
00033
00037     ListaEncadeada<ArestaEncadeada>* conexoes;
00038
00039 public:
00046     VerticeEncadeado(int id, int peso);
00047
00053     int getId() const;
00054
00060     int getPeso() const;
00061
00067     int getGrau() const;
00068
00074     VerticeEncadeado* getProximo() const;
00075
00081     void setProximo(VerticeEncadeado* novoProximo);
00082
00089     void setConexao(VerticeEncadeado* verticeDestino, int pesoAresta);
00090
00096     ArestaEncadeada* getPrimeiraConexao();
00097
00103     ListaEncadeada<ArestaEncadeada>* getConexoes();
00104
00110     void setConexoes(ListaEncadeada<ArestaEncadeada>* novasConexoes);
00111
00120     friend std::ostream& operator<<(std::ostream& os, const VerticeEncadeado& vertice);
00121
00129     int removeConexao(VerticeEncadeado* destino);
00130
00139     ArestaEncadeada* getConexao(int origem, int destino);
00140 };
00141
00142 #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)

5.13.1 Function Documentation

5.13.1.1 operator<<()

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

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)

5.16.1 Function Documentation

5.16.1.1 operator<<()

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

Parameters

os	Fluxo de saída.
vertice	O vértice a ser impresso.

Returns

O fluxo de saída.