

Trabalho de Grafos - 2024.3

v1

Generated by Doxygen 1.13.1



<b>1 Projeto de Implementação de Grafos</b>	<b>1</b>
1.1 Funcionalidades	1
1.2 Estrutura de Arquivos	1
1.2.1 Exemplo de <code>grafo.txt</code>	1
1.2.2 Exemplo de <code>descricao.txt</code>	2
1.3 Como Compilar	2
1.4 Saída Esperada	2
1.5 Estrutura do Código	2
1.6 Requisitos	3
<b>2 Hierarchical Index</b>	<b>5</b>
2.1 Class Hierarchy	5
<b>3 Class Index</b>	<b>7</b>
3.1 Class List	7
<b>4 File Index</b>	<b>9</b>
4.1 File List	9
<b>5 Class Documentation</b>	<b>11</b>
5.1 Aresta Class Reference	11
5.1.1 Constructor & Destructor Documentation	11
5.1.1.1 <code>Aresta()</code>	11
5.1.1.2 <code>~Aresta()</code>	12
5.1.2 Member Function Documentation	12
5.1.2.1 <code>getIdDestino()</code>	12
5.1.2.2 <code>getIdOrigem()</code>	12
5.1.2.3 <code>getPeso()</code>	12
5.1.2.4 <code>getProxAresta()</code>	13
5.1.2.5 <code>setNoDestino()</code>	13
5.1.2.6 <code>setNoOrigem()</code>	13
5.1.2.7 <code>setProxAresta()</code>	13
5.2 Grafo Class Reference	14
5.2.1 Constructor & Destructor Documentation	15
5.2.1.1 <code>Grafo()</code>	15
5.2.1.2 <code>~Grafo()</code>	15
5.2.2 Member Function Documentation	16
5.2.2.1 <code>arestaPonderada()</code>	16
5.2.2.2 <code>carregaGrafo()</code>	16
5.2.2.3 <code>ehArvore()</code>	16
5.2.2.4 <code>ehBipartido()</code>	16
5.2.2.5 <code>ehCompleto()</code>	17
5.2.2.6 <code>ehDirecionado()</code>	17
5.2.2.7 <code>getGrau()</code>	17

5.2.2.8	<a href="#">getNoPelold()</a>	17
5.2.2.9	<a href="#">getNumNos()</a>	18
5.2.2.10	<a href="#">getOrdem()</a>	18
5.2.2.11	<a href="#">nConexo()</a>	18
5.2.2.12	<a href="#">novoGrafo()</a>	18
5.2.2.13	<a href="#">possuiArticulacao()</a>	19
5.2.2.14	<a href="#">possuiPonte()</a>	19
5.2.2.15	<a href="#">verticePonderado()</a>	19
5.3	<a href="#">GrafoLista Class Reference</a>	19
5.3.1	<a href="#">Constructor &amp; Destructor Documentation</a>	21
5.3.1.1	<a href="#">GrafoLista()</a>	21
5.3.2	<a href="#">Member Function Documentation</a>	21
5.3.2.1	<a href="#">carregaGrafo()</a>	21
5.3.2.2	<a href="#">ehArvore()</a>	22
5.3.2.3	<a href="#">ehBipartido()</a>	22
5.3.2.4	<a href="#">ehCompleto()</a>	22
5.3.2.5	<a href="#">getGrau()</a>	22
5.3.2.6	<a href="#">nConexo()</a>	23
5.3.2.7	<a href="#">novoGrafo()</a>	23
5.3.2.8	<a href="#">possuiArticulacao()</a>	23
5.3.2.9	<a href="#">possuiPonte()</a>	23
5.4	<a href="#">GrafoMatriz Class Reference</a>	24
5.4.1	<a href="#">Constructor &amp; Destructor Documentation</a>	25
5.4.1.1	<a href="#">GrafoMatriz()</a>	25
5.4.2	<a href="#">Member Function Documentation</a>	25
5.4.2.1	<a href="#">carregaGrafo()</a>	25
5.4.2.2	<a href="#">ehArvore()</a>	26
5.4.2.3	<a href="#">ehBipartido()</a>	26
5.4.2.4	<a href="#">ehCompleto()</a>	26
5.4.2.5	<a href="#">getGrau()</a>	26
5.4.2.6	<a href="#">nConexo()</a>	27
5.4.2.7	<a href="#">novoGrafo()</a>	27
5.4.2.8	<a href="#">possuiArticulacao()</a>	27
5.4.2.9	<a href="#">possuiPonte()</a>	28
5.5	<a href="#">Lista Class Reference</a>	28
5.5.1	<a href="#">Constructor &amp; Destructor Documentation</a>	28
5.5.1.1	<a href="#">Lista()</a>	28
5.5.1.2	<a href="#">~Lista()</a>	29
5.5.2	<a href="#">Member Function Documentation</a>	29
5.5.2.1	<a href="#">adicionar()</a>	29
5.5.2.2	<a href="#">contem()</a>	29
5.5.2.3	<a href="#">getElemento()</a>	29

5.5.2.4	<a href="#">getTamanho()</a>	30
5.5.2.5	<a href="#">remover()</a>	30
5.6	No Class Reference	31
5.6.1	Constructor & Destructor Documentation	31
5.6.1.1	<a href="#">No()</a>	31
5.6.1.2	<a href="#">~No()</a>	32
5.6.2	Member Function Documentation	32
5.6.2.1	<a href="#">adicionaAresta()</a>	32
5.6.2.2	<a href="#">getGrauEntrada()</a>	32
5.6.2.3	<a href="#">getGrauSaida()</a>	32
5.6.2.4	<a href="#">getIdNo()</a>	33
5.6.2.5	<a href="#">getPesoNo()</a>	33
5.6.2.6	<a href="#">getPrimeiraAresta()</a>	33
5.6.2.7	<a href="#">getProxNo()</a>	33
5.6.2.8	<a href="#">incGrauEntrada()</a>	34
5.6.2.9	<a href="#">incGrauSaida()</a>	34
5.6.2.10	<a href="#">setProxNo()</a>	34
<b>6</b>	<b>File Documentation</b>	<b>35</b>
6.1	<a href="#">C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/Aresta.cpp File Reference</a>	35
6.1.1	Detailed Description	35
6.2	<a href="#">C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/Aresta.h</a>	35
6.3	<a href="#">C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/Grafo.cpp File Reference</a>	36
6.3.1	Detailed Description	36
6.4	<a href="#">C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/Grafo.h</a>	36
6.5	<a href="#">C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/GrafoLista.cpp File Reference</a>	36
6.5.1	Detailed Description	37
6.6	<a href="#">C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/GrafoLista.h</a>	37
6.7	<a href="#">C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/GrafoMatriz.cpp File Reference</a>	37
6.7.1	Detailed Description	37
6.8	<a href="#">C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/GrafoMatriz.h</a>	38
6.9	<a href="#">C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ListaEncad.cpp File Reference</a>	38
6.9.1	Detailed Description	38
6.10	<a href="#">C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ListaEncad.h</a>	38
6.11	<a href="#">C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/No.cpp File Reference</a>	39
6.11.1	Detailed Description	39
6.12	<a href="#">C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/No.h</a>	39
<b>Index</b>		<b>41</b>



# Chapter 1

## Projeto de Implementação de Grafos

Este projeto é uma implementação de estruturas e funcionalidades relacionadas a grafos em C++. Ele permite carregar grafos a partir de arquivos, realizar análises como conexidade, verificação de árvores, detecção de arestas ponte, entre outras.

### 1.1 Funcionalidades

- **Representação de grafos:**
  - Matriz de adjacência
  - [Lista](#) de adjacência
- **Carregamento de grafos:**
  - A partir de arquivos de entrada (`grafo.txt` e `descricao.txt`)
- **Análises:**
  - Componentes conexas
  - Verificação de completude
  - Detecção de bipartidos
  - Identificação de arestas ponte
  - Identificação de vértices de articulação

### 1.2 Estrutura de Arquivos

#### 1.2.1 Exemplo de `grafo.txt`

```
3 1 1 1
2 3 7
1 2 6
2 1 4
2 3 -5
```

- Linha 1: `nós direcionado ponderado_vertices ponderado_arestas`
- Linha 2: Pesos dos vértices (opcional, dependendo do grafo)
- Linhas seguintes: `origem destino peso`

## 1.2.2 Exemplo de descricao.txt

```
matriz
3
3
1
2
1
1
0
1
0
1
1
```

- Linha 1: Tipo da representação (`matriz` ou `lista`)
- Demais linhas: Informam propriedades do grafo (grau, ordem, conexidade etc.)

## 1.3 Como Compilar

Certifique-se de que você tenha um compilador C++ instalado, como o **g++**.

1. Compile o projeto:

```
g++ -std=c++14 -Wall -o grafo_exec Grafo.cpp GrafoLista.cpp GrafoMatriz.cpp Aresta.cpp No.cpp main.cpp
```

2. Execute o programa:

- Para carregar um grafo como **matriz**:  
`.\grafo_exec.exe -m grafo.txt`
- Para carregar um grafo como **lista**:  
`.\grafo_exec.exe -l grafo.txt`
- Para carregar um grafo a partir de um arquivo de configuração: `bash .\grafo_exec.exe -c descricao.txt`

## 1.4 Saída Esperada

Exemplo de saída:

```
===== Descrição do Grafo =====
Ordem: 3
Direcionado: Sim
Vertices ponderados: Sim
Arestas ponderadas: Sim
Grau do vértice 0: 1
Componentes conexas: 1
Completo: Não
Bipartido: Sim
Árvore: Não
Aresta Ponte: Sim
Vértice de Articulação: Sim
=====
```

## 1.5 Estrutura do Código

- `main.cpp`: Ponto de entrada do programa.
- `Grafo.h` e `Grafo.cpp`: Classe base para grafos.
- `GrafoMatriz.h` e `GrafoMatriz.cpp`: Implementação da representação por matriz de adjacência.
- `GrafoLista.h` e `GrafoLista.cpp`: Implementação da representação por lista de adjacência.
- `Aresta.h` e `Aresta.cpp`: Classe para representação de arestas.
- `No.h` e `No.cpp`: Classe para representação de nós.



## 1.6 Requisitos

- **Compilador C++** (suporta padrão C++14 ou superior).
- Sistema operacional Windows, Linux ou macOS.



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

Aresta . . . . .	11
Grafo . . . . .	14
GrafoLista . . . . .	19
GrafoMatriz . . . . .	24
Lista . . . . .	28
No . . . . .	31



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Aresta</a> . . . . .	11
<a href="#">Grafo</a> . . . . .	14
<a href="#">GrafoLista</a> . . . . .	19
<a href="#">GrafoMatriz</a> . . . . .	24
<a href="#">Lista</a> . . . . .	28
<a href="#">No</a> . . . . .	31



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all documented files with brief descriptions:

C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ <a href="#">Aresta.cpp</a>	
Implementação das funções da classe <a href="#">Aresta</a> . . . . .	35
C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ <a href="#">Aresta.h</a> . . . . .	35
C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ <a href="#">Grafo.cpp</a>	
Implementação das funções da classe abstrata <a href="#">Grafo</a> . . . . .	36
C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ <a href="#">Grafo.h</a> . . . . .	36
C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ <a href="#">GrafoLista.cpp</a>	
Implementação das funções da classe <a href="#">GrafoLista</a> . . . . .	36
C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ <a href="#">GrafoLista.h</a> . . . . .	37
C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ <a href="#">GrafoMatriz.cpp</a>	
Implementação das funções da classe <a href="#">GrafoMatriz</a> . . . . .	37
C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ <a href="#">GrafoMatriz.h</a> . . . . .	38
C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ <a href="#">ListaEncad.cpp</a>	
Implementação das funções da classe <a href="#">Lista</a> . . . . .	38
C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ <a href="#">ListaEncad.h</a> . . . . .	38
C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ <a href="#">No.cpp</a>	
Implementação das funções da classe <a href="#">No</a> . . . . .	39
C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ <a href="#">No.h</a> . . . . .	39





## Chapter 5

# Class Documentation

### 5.1 Aresta Class Reference

#### Public Member Functions

- [Aresta](#) ([No](#) \*origem, [No](#) \*destino, float peso=1.0)  
*Construtor da classe [Aresta](#).*
- [~Aresta](#) ()  
*Destrutor da classe [Aresta](#).*
- int [getIdDestino](#) ()  
*Obtém o ID do nó de destino da aresta.*
- int [getIdOrigem](#) ()  
*Obtém o ID do nó de origem da aresta.*
- void [setNoOrigem](#) ([No](#) \*no)  
*Define o nó de origem da aresta.*
- void [setNoDestino](#) ([No](#) \*no)  
*Define o nó de destino da aresta.*
- float [getPeso](#) ()  
*Obtém o peso da aresta.*
- [Aresta](#) \* [getProxAresta](#) ()  
*Obtém a próxima aresta.*
- void [setProxAresta](#) ([Aresta](#) \*prox)  
*Define a próxima aresta.*

#### 5.1.1 Constructor & Destructor Documentation

##### 5.1.1.1 Aresta()

```
Aresta::Aresta (  
    No * origem,  
    No * destino,  
    float peso = 1.0)
```

Construtor da classe [Aresta](#).

Este construtor inicializa uma aresta com os nós de origem e destino, o peso da aresta e o ponteiro para a próxima aresta.

**Parameters**

<i>origem</i>	Ponteiro para o nó de origem da aresta.
<i>destino</i>	Ponteiro para o nó de destino da aresta.
<i>peso</i>	O peso da aresta.

**5.1.1.2 ~Aresta()**

```
Aresta::~~Aresta ()
```

Destrutor da classe [Aresta](#).

O destrutor da classe [Aresta](#) é vazio, pois a liberação de memória dos nós (origem e destino) deve ser tratada em outra parte do código.

**5.1.2 Member Function Documentation****5.1.2.1 getIdDestino()**

```
int Aresta::getIdDestino ()
```

Obtém o ID do nó de destino da aresta.

Este método retorna o ID do nó de destino da aresta.

**Returns**

O ID do nó de destino.

**5.1.2.2 getIdOrigem()**

```
int Aresta::getIdOrigem ()
```

Obtém o ID do nó de origem da aresta.

Este método retorna o ID do nó de origem da aresta.

**Returns**

O ID do nó de origem.

**5.1.2.3 getPeso()**

```
float Aresta::getPeso ()
```

Obtém o peso da aresta.

Este método retorna o peso atribuído à aresta.

**Returns**

O peso da aresta.

#### 5.1.2.4 getProxAresta()

```
Aresta * Aresta::getProxAresta ()
```

Obtém a próxima aresta.

Este método retorna o ponteiro para a próxima aresta na lista.

##### Returns

Ponteiro para a próxima aresta.

#### 5.1.2.5 setNoDestino()

```
void Aresta::setNoDestino (  
    No * valor)
```

Define o nó de destino da aresta.

Este método define o nó de destino da aresta.

##### Parameters

<i>valor</i>	Ponteiro para o nó de destino.
--------------	--------------------------------

#### 5.1.2.6 setNoOrigem()

```
void Aresta::setNoOrigem (  
    No * valor)
```

Define o nó de origem da aresta.

Este método define o nó de origem da aresta.

##### Parameters

<i>valor</i>	Ponteiro para o nó de origem.
--------------	-------------------------------

#### 5.1.2.7 setProxAresta()

```
void Aresta::setProxAresta (  
    Aresta * prox)
```

Define a próxima aresta.

Este método define a próxima aresta na lista de arestas.

## Parameters

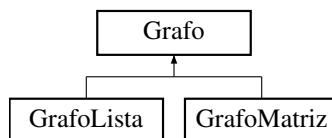
<i>prox</i>	Ponteiro para a próxima aresta a ser associada.
-------------	---

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

- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/Aresta.h
- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/[Aresta.cpp](#)

## 5.2 Grafo Class Reference

Inheritance diagram for Grafo:



### Public Member Functions

- [Grafo](#) (int ordem, bool direcionado, bool ponderadoVertices, bool ponderadoArestas)  
*Construtor da classe [Grafo](#).*
- virtual [~Grafo](#) ()  
*Destrutor da classe [Grafo](#).*
- virtual int [getGrau](#) (int vertice)  
*Obtém o grau de um vértice.*
- virtual int [getOrdem](#) ()  
*Obtém a ordem do grafo.*
- virtual int [getNumNos](#) ()  
*Obtém o número de nós do grafo.*
- virtual bool [ehDirecionado](#) ()  
*Verifica se o grafo é direcionado.*
- virtual bool [verticePonderado](#) ()  
*Verifica se o grafo possui vértices ponderados.*
- virtual bool [arestaPonderada](#) ()  
*Verifica se o grafo possui arestas ponderadas.*
- virtual bool [ehCompleto](#) ()  
*Verifica se o grafo é completo. Um grafo completo possui todas as combinações possíveis de arestas entre seus vértices.*
- virtual bool [ehBipartido](#) ()  
*Verifica se o grafo é bipartido. Um grafo bipartido pode ser dividido em dois subconjuntos, onde não existem arestas entre vértices do mesmo subconjunto.*
- [No](#) \* [getNoPeloid](#) (int id)  
*Obtém o nó pelo seu ID.*
- virtual int [nConexo](#) ()  
*Determina o número de componentes conexos no grafo. Um componente conexo é um subconjunto de vértices onde existe pelo menos um caminho entre cada par de vértices.*

- virtual bool [ehArvore](#) ()  
*Verifica se o grafo é uma árvore. Uma árvore é um grafo conexo sem ciclos.*
- virtual bool [possuiPonte](#) ()  
*Verifica se o grafo possui pontes. Uma ponte é uma aresta cuja remoção aumenta o número de componentes conexos do grafo.*
- virtual bool [possuiArticulacao](#) ()  
*Verifica se o grafo possui vértices de articulação. Um vértice de articulação é aquele cuja remoção aumenta o número de componentes conexos do grafo.*
- virtual void [carregaGrafo](#) (const std::string &arquivo)
- virtual void [novoGrafo](#) (const std::string &arquivoConfig)

### Protected Attributes

- int **ordem**
- int **numNos**
- bool **direcionado**
- bool **ponderadoVertices**
- bool **ponderadoArestas**
- No \* **primeiroNo**
- No \* **ultimoNo**

## 5.2.1 Constructor & Destructor Documentation

### 5.2.1.1 Grafo()

```
Grafo::Grafo (
    int ordem,
    bool direcionado,
    bool ponderadoVertices,
    bool ponderadoArestas)
```

Construtor da classe [Grafo](#).

Este construtor inicializa um grafo com base na ordem (número de vértices), se o grafo é direcionado ou não, se os vértices são ponderados e se as arestas são ponderadas.

#### Parameters

<i>ordem</i>	Número de vértices do grafo.
<i>direcionado</i>	Indica se o grafo é direcionado (true) ou não (false).
<i>ponderadoVertices</i>	Indica se os vértices são ponderados (true) ou não (false).
<i>ponderadoArestas</i>	Indica se as arestas são ponderadas (true) ou não (false).

### 5.2.1.2 ~Grafo()

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

Destrutor da classe [Grafo](#).

Este destrutor é responsável por liberar os recursos alocados pelo grafo.

## 5.2.2 Member Function Documentation

### 5.2.2.1 arestaPonderada()

```
bool Grafo::arestaPonderada () [virtual]
```

Verifica se o grafo possui arestas ponderadas.

#### Returns

true se ao menos uma aresta for ponderada, false caso contrário.

### 5.2.2.2 carregaGrafo()

```
virtual void Grafo::carregaGrafo (  
    const std::string & arquivo) [virtual]
```

Reimplemented in [GrafoLista](#), and [GrafoMatriz](#).

### 5.2.2.3 ehArvore()

```
bool Grafo::ehArvore () [virtual]
```

Verifica se o grafo é uma árvore. Uma árvore é um grafo conexo sem ciclos.

#### Returns

true se o grafo é uma árvore; caso contrário, false.

Reimplemented in [GrafoLista](#), and [GrafoMatriz](#).

### 5.2.2.4 ehBipartido()

```
bool Grafo::ehBipartido () [virtual]
```

Verifica se o grafo é bipartido. Um grafo bipartido pode ser dividido em dois subconjuntos, onde não existem arestas entre vértices do mesmo subconjunto.

#### Returns

true se o grafo é bipartido; caso contrário, false.

Reimplemented in [GrafoLista](#), and [GrafoMatriz](#).

### 5.2.2.5 ehCompleto()

```
bool Grafo::ehCompleto () [virtual]
```

Verifica se o grafo é completo. Um grafo completo possui todas as combinações possíveis de arestas entre seus vértices.

#### Returns

true se o grafo é completo; caso contrário, false.

Reimplemented in [GrafoLista](#), and [GrafoMatriz](#).

### 5.2.2.6 ehDirecionado()

```
bool Grafo::ehDirecionado () [virtual]
```

Verifica se o grafo é direcionado.

#### Returns

true se o grafo for direcionado, false caso contrário.

### 5.2.2.7 getGrau()

```
int Grafo::getGrau (
    int vertice) [virtual]
```

Obtém o grau de um vértice.

#### Parameters

<i>vertice</i>	Índice do vértice.
----------------	--------------------

#### Returns

Número de arestas conectadas ao vértice especificado.

Reimplemented in [GrafoLista](#), and [GrafoMatriz](#).

### 5.2.2.8 getNoPeloid()

```
No * Grafo::getNoPeloId (
    int id)
```

Obtém o nó pelo seu ID.

**Parameters**

<i>id</i>	O ID do nó a ser buscado.
-----------	---------------------------

**Returns**

O ponteiro para o nó com o ID especificado ou nullptr caso o nó não seja encontrado.

**5.2.2.9 getNumNos()**

```
int Grafo::getNumNos () [virtual]
```

Obtém o número de nós do grafo.

**Returns**

O número de nós do grafo.

**5.2.2.10 getOrdem()**

```
int Grafo::getOrdem () [virtual]
```

Obtém a ordem do grafo.

**Returns**

A ordem (número de vértices) do grafo.

**5.2.2.11 nConexo()**

```
int Grafo::nConexo () [virtual]
```

Determina o número de componentes conexos no grafo. Um componente conexo é um subconjunto de vértices onde existe pelo menos um caminho entre cada par de vértices.

**Returns**

Número de componentes conexos no grafo.

Reimplemented in [GrafoLista](#), and [GrafoMatriz](#).

**5.2.2.12 novoGrafo()**

```
virtual void Grafo::novoGrafo (  
    const std::string & arquivoConfig) [virtual]
```

Reimplemented in [GrafoLista](#), and [GrafoMatriz](#).



#### 5.2.2.13 possuiArticulacao()

```
bool Grafo::possuiArticulacao () [virtual]
```

Verifica se o grafo possui vértices de articulação. Um vértice de articulação é aquele cuja remoção aumenta o número de componentes conexos do grafo.

##### Returns

true se existe pelo menos um vértice de articulação; caso contrário, false.

Reimplemented in [GrafoLista](#), and [GrafoMatriz](#).

#### 5.2.2.14 possuiPonte()

```
bool Grafo::possuiPonte () [virtual]
```

Verifica se o grafo possui pontes. Uma ponte é uma aresta cuja remoção aumenta o número de componentes conexos do grafo.

##### Returns

true se existe pelo menos uma ponte; caso contrário, false.

Reimplemented in [GrafoLista](#), and [GrafoMatriz](#).

#### 5.2.2.15 verticePonderado()

```
bool Grafo::verticePonderado () [virtual]
```

Verifica se o grafo possui vértices ponderados.

##### Returns

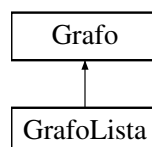
true se ao menos um vértice for ponderado, false caso contrário.

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

- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/Grafo.h
- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/[Grafo.cpp](#)

## 5.3 GrafoLista Class Reference

Inheritance diagram for GrafoLista:



## Public Member Functions

- [GrafoLista](#) (int ordem, bool direcionado, bool ponderadoVertices, bool ponderadoArestas)  
*Construtor da classe [GrafoLista](#).*
- [~GrafoLista](#) ()  
*Destrutor da classe [GrafoLista](#). Libera a memória alocada para a lista de adjacência.*
- int [getGrau](#) (int vertice) override  
*Obtém o grau de um vértice.*
- bool [ehCompleto](#) () override  
*Verifica se o grafo é completo. Um grafo completo possui todas as combinações possíveis de arestas entre seus vértices.*
- bool [ehBipartido](#) () override  
*Verifica se o grafo é bipartido. Um grafo bipartido pode ser dividido em dois subconjuntos, onde não existem arestas entre vértices do mesmo subconjunto.*
- int [nConexo](#) () override  
*Determina o número de componentes conexos no grafo. Um componente conexo é um subconjunto de vértices onde existe pelo menos um caminho entre cada par de vértices.*
- bool [ehArvore](#) () override  
*Verifica se o grafo é uma árvore. Uma árvore é um grafo conexo sem ciclos.*
- bool [possuiPonte](#) () override  
*Verifica se o grafo possui pontes. Uma ponte é uma aresta cuja remoção aumenta o número de componentes conexos do grafo.*
- bool [possuiArticulacao](#) () override  
*Verifica se o grafo possui vértices de articulação. Um vértice de articulação é aquele cuja remoção aumenta o número de componentes conexos do grafo.*
- void [carregaGrafo](#) (const std::string &arquivo) override  
*Carrega o grafo a partir de um arquivo. O arquivo deve conter os dados do grafo, como número de nós, tipo de grafo e arestas.*
- void [novoGrafo](#) (const std::string &arquivoConfig) override  
*Cria um novo grafo a partir de um arquivo de configuração. O arquivo deve conter as informações do grafo, como tipo de estrutura, número de nós e arestas.*

## Public Member Functions inherited from [Grafo](#)

- [Grafo](#) (int ordem, bool direcionado, bool ponderadoVertices, bool ponderadoArestas)  
*Construtor da classe [Grafo](#).*
- virtual [~Grafo](#) ()  
*Destrutor da classe [Grafo](#).*
- virtual int [getOrdem](#) ()  
*Obtém a ordem do grafo.*
- virtual int [getNumNos](#) ()  
*Obtém o número de nós do grafo.*
- virtual bool [ehDirecionado](#) ()  
*Verifica se o grafo é direcionado.*
- virtual bool [verticePonderado](#) ()  
*Verifica se o grafo possui vértices ponderados.*
- virtual bool [arestaPonderada](#) ()  
*Verifica se o grafo possui arestas ponderadas.*
- [No](#) \* [getNoPeloid](#) (int id)  
*Obtém o nó pelo seu ID.*

## Additional Inherited Members

### Protected Attributes inherited from [Grafo](#)

- int **ordem**
- int **numNos**
- bool **direcionado**
- bool **ponderadoVertices**
- bool **ponderadoArestas**
- [No](#) \* **primeiroNo**
- [No](#) \* **ultimoNo**

## 5.3.1 Constructor & Destructor Documentation

### 5.3.1.1 GrafoLista()

```
GrafoLista::GrafoLista (  
    int ordem,  
    bool direcionado,  
    bool ponderadoVertices,  
    bool ponderadoArestas)
```

Construtor da classe [GrafoLista](#).

#### Parameters

<i>ordem</i>	Número de vértices do grafo.
<i>direcionado</i>	Indica se o grafo é direcionado (true) ou não (false).
<i>ponderadoVertices</i>	Indica se os vértices possuem pesos (true) ou não (false).
<i>ponderadoArestas</i>	Indica se as arestas possuem pesos (true) ou não (false).

## 5.3.2 Member Function Documentation

### 5.3.2.1 carregaGrafo()

```
void GrafoLista::carregaGrafo (  
    const std::string & arquivo) [override], [virtual]
```

Carrega o grafo a partir de um arquivo. O arquivo deve conter os dados do grafo, como número de nós, tipo de grafo e arestas.

#### Parameters

<i>arquivo</i>	Caminho para o arquivo de entrada.
----------------	------------------------------------

Reimplemented from [Grafo](#).

### 5.3.2.2 ehArvore()

```
bool GrafoLista::ehArvore () [override], [virtual]
```

Verifica se o grafo é uma árvore. Uma árvore é um grafo conexo sem ciclos.

#### Returns

true se o grafo é uma árvore; caso contrário, false.

Reimplemented from [Grafo](#).

### 5.3.2.3 ehBipartido()

```
bool GrafoLista::ehBipartido () [override], [virtual]
```

Verifica se o grafo é bipartido. Um grafo bipartido pode ser dividido em dois subconjuntos, onde não existem arestas entre vértices do mesmo subconjunto.

#### Returns

true se o grafo é bipartido; caso contrário, false.

Reimplemented from [Grafo](#).

### 5.3.2.4 ehCompleto()

```
bool GrafoLista::ehCompleto () [override], [virtual]
```

Verifica se o grafo é completo. Um grafo completo possui todas as combinações possíveis de arestas entre seus vértices.

#### Returns

true se o grafo é completo; caso contrário, false.

Reimplemented from [Grafo](#).

### 5.3.2.5 getGrau()

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

Obtém o grau de um vértice.

#### Parameters

<i>vertice</i>	Índice do vértice.
----------------	--------------------

#### Returns

Número de arestas conectadas ao vértice especificado.

Reimplemented from [Grafo](#).

#### 5.3.2.6 nConexo()

```
int GrafoLista::nConexo () [override], [virtual]
```

Determina o número de componentes conexos no grafo. Um componente conexo é um subconjunto de vértices onde existe pelo menos um caminho entre cada par de vértices.

##### Returns

Número de componentes conexos no grafo.

Reimplemented from [Grafo](#).

#### 5.3.2.7 novoGrafo()

```
void GrafoLista::novoGrafo (
    const std::string & arquivoConfig) [override], [virtual]
```

Cria um novo grafo a partir de um arquivo de configuração. O arquivo deve conter as informações do grafo, como tipo de estrutura, número de nós e arestas.

##### Parameters

<i>arquivoConfig</i>	Caminho para o arquivo de configuração.
----------------------	---

Reimplemented from [Grafo](#).

#### 5.3.2.8 possuiArticulacao()

```
bool GrafoLista::possuiArticulacao () [override], [virtual]
```

Verifica se o grafo possui vértices de articulação. Um vértice de articulação é aquele cuja remoção aumenta o número de componentes conexos do grafo.

##### Returns

true se existe pelo menos um vértice de articulação; caso contrário, false.

Reimplemented from [Grafo](#).

#### 5.3.2.9 possuiPonte()

```
bool GrafoLista::possuiPonte () [override], [virtual]
```

Verifica se o grafo possui pontes. Uma ponte é uma aresta cuja remoção aumenta o número de componentes conexos do grafo.

##### Returns

true se existe pelo menos uma ponte; caso contrário, false.

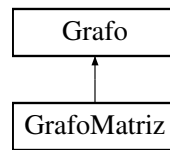
Reimplemented from [Grafo](#).

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

- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/GrafoLista.h
- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/[GrafoLista.cpp](#)

## 5.4 GrafoMatriz Class Reference

Inheritance diagram for GrafoMatriz:



### Public Member Functions

- [GrafoMatriz](#) (int ordem, bool direcionado, bool ponderadoVertices, bool ponderadoArestas)  
*Construtor da classe [GrafoMatriz](#).*
- [~GrafoMatriz](#) ()  
*Destrutor da classe [GrafoMatriz](#). Libera a memória alocada para o grafo de matriz de adjacência.*
- int [getGrau](#) (int vertice) override  
*Calcula o grau de um vértice no grafo. O grau de um vértice é a soma dos pesos das arestas conectadas a ele.*
- bool [ehCompleto](#) () override  
*Verifica se o grafo é completo. Um grafo completo possui todas as combinações possíveis de arestas entre seus vértices.*
- bool [ehBipartido](#) () override  
*Verifica se o grafo é bipartido. Um grafo bipartido pode ser dividido em dois subconjuntos, onde não existem arestas entre vértices do mesmo subconjunto.*
- int [nConexo](#) () override  
*Determina o número de componentes conexos no grafo. Um componente conexo é um subconjunto de vértices onde existe pelo menos um caminho entre cada par de vértices.*
- bool [ehArvore](#) () override  
*Verifica se o grafo é uma árvore. Uma árvore é um grafo conexo sem ciclos.*
- bool [possuiPonte](#) () override  
*Verifica se o grafo possui pontes. Uma ponte é uma aresta cuja remoção aumenta o número de componentes conexos do grafo.*
- bool [possuiArticulacao](#) () override  
*Verifica se o grafo possui vértices de articulação. Um vértice de articulação é aquele cuja remoção aumenta o número de componentes conexos do grafo.*
- void [carregaGrafo](#) (const std::string &arquivo) override  
*Carrega o grafo a partir de um arquivo. O arquivo deve conter os dados do grafo, como número de nós, tipo de grafo e arestas.*
- void [novoGrafo](#) (const std::string &arquivoConfig) override  
*Cria um novo grafo a partir de um arquivo de configuração. O arquivo deve conter as informações do grafo, como tipo de estrutura, número de nós e arestas.*

### Public Member Functions inherited from [Grafo](#)

- [Grafo](#) (int ordem, bool direcionado, bool ponderadoVertices, bool ponderadoArestas)  
*Construtor da classe [Grafo](#).*
- virtual [~Grafo](#) ()  
*Destrutor da classe [Grafo](#).*
- virtual int [getOrdem](#) ()  
*Obtém a ordem do grafo.*

- virtual int [getNumNos](#) ()  
*Obtém o número de nós do grafo.*
- virtual bool [ehDirecionado](#) ()  
*Verifica se o grafo é direcionado.*
- virtual bool [verticePonderado](#) ()  
*Verifica se o grafo possui vértices ponderados.*
- virtual bool [arestaPonderada](#) ()  
*Verifica se o grafo possui arestas ponderadas.*
- [No](#) \* [getNoPeloid](#) (int id)  
*Obtém o nó pelo seu ID.*

### Additional Inherited Members

### Protected Attributes inherited from [Grafo](#)

- int **ordem**
- int **numNos**
- bool **direcionado**
- bool **ponderadoVertices**
- bool **ponderadoArestas**
- [No](#) \* **primeiroNo**
- [No](#) \* **ultimoNo**

## 5.4.1 Constructor & Destructor Documentation

### 5.4.1.1 GrafoMatriz()

```
GrafoMatriz::GrafoMatriz (
    int ordem,
    bool direcionado,
    bool ponderadoVertices,
    bool ponderadoArestas)
```

Construtor da classe [GrafoMatriz](#).

#### Parameters

<i>ordem</i>	Número de vértices do grafo.
<i>direcionado</i>	Indica se o grafo é direcionado (true) ou não (false).
<i>ponderadoVertices</i>	Indica se os vértices possuem pesos (true) ou não (false).
<i>ponderadoArestas</i>	Indica se as arestas possuem pesos (true) ou não (false).

## 5.4.2 Member Function Documentation

### 5.4.2.1 carregaGrafo()

```
void GrafoMatriz::carregaGrafo (
    const std::string & arquivo) [override], [virtual]
```

Carrega o grafo a partir de um arquivo. O arquivo deve conter os dados do grafo, como número de nós, tipo de grafo e arestas.

#### Parameters

<i>arquivo</i>	Caminho para o arquivo de entrada.
----------------	------------------------------------

Reimplemented from [Grafo](#).

#### 5.4.2.2 ehArvore()

```
bool GrafoMatriz::ehArvore () [override], [virtual]
```

Verifica se o grafo é uma árvore. Uma árvore é um grafo conexo sem ciclos.

#### Returns

true se o grafo é uma árvore; caso contrário, false.

Reimplemented from [Grafo](#).

#### 5.4.2.3 ehBipartido()

```
bool GrafoMatriz::ehBipartido () [override], [virtual]
```

Verifica se o grafo é bipartido. Um grafo bipartido pode ser dividido em dois subconjuntos, onde não existem arestas entre vértices do mesmo subconjunto.

#### Returns

true se o grafo é bipartido; caso contrário, false.

Reimplemented from [Grafo](#).

#### 5.4.2.4 ehCompleto()

```
bool GrafoMatriz::ehCompleto () [override], [virtual]
```

Verifica se o grafo é completo. Um grafo completo possui todas as combinações possíveis de arestas entre seus vértices.

#### Returns

true se o grafo é completo; caso contrário, false.

Reimplemented from [Grafo](#).

#### 5.4.2.5 getGrau()

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

Calcula o grau de um vértice no grafo. O grau de um vértice é a soma dos pesos das arestas conectadas a ele.



## Parameters

<i>vertice</i>	O índice do vértice cujo grau será calculado.
----------------	---

## Returns

O grau do vértice especificado.

## Warning

Certifique-se de que o índice do vértice está dentro do intervalo [0, ordem-1].

Reimplemented from [Grafo](#).

**5.4.2.6 nConexo()**

```
int GrafoMatriz::nConexo () [override], [virtual]
```

Determina o número de componentes conexos no grafo. Um componente conexo é um subconjunto de vértices onde existe pelo menos um caminho entre cada par de vértices.

## Returns

Número de componentes conexos no grafo.

Reimplemented from [Grafo](#).

**5.4.2.7 novoGrafo()**

```
void GrafoMatriz::novoGrafo (  
    const std::string & arquivoConfig) [override], [virtual]
```

Cria um novo grafo a partir de um arquivo de configuração. O arquivo deve conter as informações do grafo, como tipo de estrutura, número de nós e arestas.

## Parameters

<i>arquivoConfig</i>	Caminho para o arquivo de configuração.
----------------------	---

Reimplemented from [Grafo](#).

**5.4.2.8 possuiArticulacao()**

```
bool GrafoMatriz::possuiArticulacao () [override], [virtual]
```

Verifica se o grafo possui vértices de articulação. Um vértice de articulação é aquele cuja remoção aumenta o número de componentes conexos do grafo.

## Returns

true se existe pelo menos um vértice de articulação; caso contrário, false.

Reimplemented from [Grafo](#).

#### 5.4.2.9 possuiPonte()

```
bool GrafoMatriz::possuiPonte () [override], [virtual]
```

Verifica se o grafo possui pontes. Uma ponte é uma aresta cuja remoção aumenta o número de componentes conexos do grafo.

##### Returns

true se existe pelo menos uma ponte; caso contrário, false.

Reimplemented from [Grafo](#).

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

- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/GrafoMatriz.h
- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/[GrafoMatriz.cpp](#)

## 5.5 Lista Class Reference

### Public Member Functions

- [Lista](#) ()  
*Construtor da classe [Lista](#).*
- [~Lista](#) ()  
*Destrutor da classe [Lista](#).*
- void [adicionar](#) (int idNo, float pesoNo=0.0)  
*Adiciona um novo nó na lista.*
- void [remover](#) (int idNo)  
*Remove um nó da lista pelo seu ID.*
- bool [contem](#) (int idNo)  
*Verifica se a lista contém um nó com o ID fornecido.*
- int [getTamanho](#) ()  
*Obtém o tamanho da lista.*
- [No](#) \* [getElemento](#) (int indice)  
*Obtém o elemento na posição especificada da lista.*

### 5.5.1 Constructor & Destructor Documentation

#### 5.5.1.1 Lista()

```
Lista::Lista ()
```

Construtor da classe [Lista](#).

Este construtor inicializa a lista encadeada, configurando a cabeça como nullptr e o tamanho como 0.

### 5.5.1.2 ~Lista()

```
Lista::~~Lista ()
```

Destrutor da classe [Lista](#).

O destrutor percorre a lista encadeada e deleta todos os nós da lista, liberando a memória alocada.

## 5.5.2 Member Function Documentation

### 5.5.2.1 adicionar()

```
void Lista::adicionar (  
    int idNo,  
    float pesoNo = 0.0)
```

Adiciona um novo nó na lista.

Este método adiciona um novo nó no início da lista, com o ID e peso fornecidos.

#### Parameters

<i>idNo</i>	O ID do nó a ser adicionado.
<i>pesoNo</i>	O peso do nó a ser adicionado.

### 5.5.2.2 contem()

```
bool Lista::contem (  
    int idNo)
```

Verifica se a lista contém um nó com o ID fornecido.

Este método percorre a lista para verificar se há um nó com o ID especificado.

#### Parameters

<i>idNo</i>	O ID do nó a ser buscado.
-------------	---------------------------

#### Returns

true Se o nó com o ID fornecido for encontrado, false caso contrário.

### 5.5.2.3 getElemento()

```
No * Lista::getElemento (  
    int indice)
```

Obtém o elemento na posição especificada da lista.

Este método retorna o nó na posição indicada pelo índice. Se o índice for inválido (fora dos limites), retorna nullptr.

**Parameters**

<i>índice</i>	O índice do elemento desejado.
---------------	--------------------------------

**Returns**

O ponteiro para o nó na posição especificada, ou nullptr caso o índice esteja fora dos limites.

**5.5.2.4 getTamanho()**

```
int Lista::getTamanho ()
```

Obtém o tamanho da lista.

Este método retorna o número de nós presentes na lista.

**Returns**

O número de elementos (tamanho) na lista.

**5.5.2.5 remover()**

```
void Lista::remover (  
    int idNo)
```

Remove um nó da lista pelo seu ID.

Este método remove o nó com o ID especificado da lista. Se o nó for encontrado, ele será removido e a memória alocada será liberada.

**Parameters**

<i>idNo</i>	O ID do nó a ser removido.
-------------	----------------------------

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

- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ListaEncad.h
- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/[ListaEncad.cpp](#)

## 5.6 No Class Reference

### Public Member Functions

- `No` (int idNo, float pesoNo=0.0)  
*Construtor da classe `No`.*
- `~No` ()  
*Destrutor da classe `No`.*
- int `getIdNo` ()  
*Obtém o ID do nó.*
- float `getPesoNo` ()  
*Obtém o peso do nó.*
- unsigned int `getGrauEntrada` ()  
*Obtém o grau de entrada do nó.*
- unsigned int `getGrauSaida` ()  
*Obtém o grau de saída do nó.*
- `Aresta *` `getPrimeiraAresta` ()  
*Obtém a primeira aresta associada ao nó.*
- `No *` `getProxNo` ()  
*Obtém o próximo nó na lista de nós.*
- void `setProxNo` (`No *`proxNo)  
*Define o próximo nó na lista de nós.*
- void `adicionaAresta` (int destino, float peso=1.0, bool direcionado=false)  
*Adiciona uma aresta ao nó.*
- void `incGrauEntrada` ()  
*Incrementa o grau de entrada do nó.*
- void `incGrauSaida` ()  
*Incrementa o grau de saída do nó.*

### 5.6.1 Constructor & Destructor Documentation

#### 5.6.1.1 No()

```
No::No (
    int idNo,
    float pesoNo = 0.0)
```

Construtor da classe `No`.

Este construtor inicializa um nó com o ID, peso, e valores iniciais para os graus de entrada e saída, além de inicializar os ponteiros para as arestas e o próximo nó.

#### Parameters

<i>idNo</i>	O ID do nó.
<i>pesoNo</i>	O peso do nó.

### 5.6.1.2 ~No()

```
No::~~No ()
```

Destrutor da classe [No](#).

O destrutor percorre a lista de arestas associadas ao nó e as libera corretamente.

## 5.6.2 Member Function Documentation

### 5.6.2.1 adicionaAresta()

```
void No::adicionaAresta (  
    int destino,  
    float peso = 1.0,  
    bool direcionado = false)
```

Adiciona uma aresta ao nó.

Este método cria uma nova aresta com o nó de origem e destino, e o peso atribuído. A aresta é adicionada à lista de arestas do nó. Caso o grafo não seja direcionado, o grau de entrada também é atualizado.

#### Parameters

<i>destino</i>	O ID do nó de destino da aresta.
<i>peso</i>	O peso da aresta.
<i>direcionado</i>	Se a aresta é direcionada ou não.

### 5.6.2.2 getGrauEntrada()

```
unsigned int No::getGrauEntrada ()
```

Obtém o grau de entrada do nó.

Este método retorna o número de arestas que entram no nó.

#### Returns

O grau de entrada do nó.

### 5.6.2.3 getGrauSaida()

```
unsigned int No::getGrauSaida ()
```

Obtém o grau de saída do nó.

Este método retorna o número de arestas que saem do nó.

#### Returns

O grau de saída do nó.

#### 5.6.2.4 getIdNo()

```
int No::getIdNo ()
```

Obtém o ID do nó.

Este método retorna o ID do nó.

##### Returns

O ID do nó.

#### 5.6.2.5 getPesoNo()

```
float No::getPesoNo ()
```

Obtém o peso do nó.

Este método retorna o peso associado ao nó.

##### Returns

O peso do nó.

#### 5.6.2.6 getPrimeiraAresta()

```
Aresta * No::getPrimeiraAresta ()
```

Obtém a primeira aresta associada ao nó.

Este método retorna o ponteiro para a primeira aresta do nó.

##### Returns

Ponteiro para a primeira aresta do nó.

#### 5.6.2.7 getProxNo()

```
No * No::getProxNo ()
```

Obtém o próximo nó na lista de nós.

Este método retorna o ponteiro para o próximo nó.

##### Returns

Ponteiro para o próximo nó.

#### 5.6.2.8 incGrauEntrada()

```
void No::incGrauEntrada ()
```

Incrementa o grau de entrada do nó.

Este método incrementa o grau de entrada do nó em 1.

#### 5.6.2.9 incGrauSaida()

```
void No::incGrauSaida ()
```

Incrementa o grau de saída do nó.

Este método incrementa o grau de saída do nó em 1.

#### 5.6.2.10 setProxNo()

```
void No::setProxNo (  
    No * proxNo)
```

Define o próximo nó na lista de nós.

Este método define o próximo nó ao qual o nó atual está apontando.

##### Parameters

<i>proxNo</i>	Ponteiro para o próximo nó.
---------------	-----------------------------

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

- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/No.h
- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/[No.cpp](#)



## Chapter 6

# File Documentation

### 6.1 C:/Users/Felipe

#### Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/Aresta.cpp File Reference

Implementação das funções da classe [Aresta](#).

```
#include "Aresta.h"
#include "No.h"
```

##### 6.1.1 Detailed Description

Implementação das funções da classe [Aresta](#).

### 6.2 C:/Users/Felipe

#### Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/Aresta.h

```
00001 #ifndef ARESTA_H
00002 #define ARESTA_H
00003
00004 class Aresta {
00005 private:
00006     No* origem;
00007     No* destino;
00008     float peso;
00009     Aresta* proxAresta;
00010
00011 public:
00012     Aresta(No* origem, No* destino, float peso = 1.0);
00013     ~Aresta();
00014
00015     int getIdDestino();
00016     int getIdOrigem();
00017     void setNoOrigem(No* no);
00018     void setNoDestino(No* no);
00019     float getPeso();
00020     Aresta* getProxAresta();
00021     void setProxAresta(Aresta* prox);
00022 };
00023
00024 #endif // ARESTA_H
```

## 6.3 C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/Grafo.cpp File Reference

Implementação das funções da classe abstrata [Grafo](#).

```
#include "Grafo.h"
#include "No.h"
#include "Aresta.h"
#include <iostream>
```

### 6.3.1 Detailed Description

Implementação das funções da classe abstrata [Grafo](#).

## 6.4 C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/Grafo.h

```
00001 #ifndef GRAFO_H
00002 #define GRAFO_H
00003
00004 #include <string>
00005
00006 class Grafo {
00007 protected:
00008     int ordem;
00009     int numNos;
00010     bool direcionado;
00011     bool ponderadoVertices;
00012     bool ponderadoArestas;
00013
00014     No *primeiroNo;
00015     No *ultimoNo;
00016
00017 public:
00018     Grafo(int ordem, bool direcionado, bool ponderadoVertices, bool ponderadoArestas);
00019     virtual ~Grafo();
00020
00021     virtual int getGrau(int vertice);
00022     virtual int getOrdem();
00023     virtual int getNumNos();
00024     virtual bool ehDirecionado();
00025     virtual bool verticePonderado();
00026     virtual bool arestaPonderada();
00027     virtual bool ehCompleto();
00028     virtual bool ehBipartido();
00029     No *getNoPeloId(int id);
00030     virtual int nConexo();
00031     virtual bool ehArvore();
00032     virtual bool possuiPonte();
00033     virtual bool possuiArticulacao();
00034     virtual void carregaGrafo(const std::string& arquivo);
00035     virtual void novoGrafo(const std::string& arquivoConfig);
00036 };
00037
00038 #endif // GRAFO_H
```

## 6.5 C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/GrafoLista.cpp File Reference

Implementação das funções da classe [GrafoLista](#).

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

### 6.5.1 Detailed Description

Implementação das funções da classe [GrafoLista](#).

## 6.6 C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/GrafoLista.h

```
00001 #ifndef GRAFOLISTA_H
00002 #define GRAFOLISTA_H
00003
00004 #include "Grafo.h"
00005 #include "ListaEncad.h"
00006 #include <string>
00007
00008 class GrafoLista : public Grafo {
00009 private:
00010     Lista* listaAdj;
00011     int numVertices;
00012
00013 public:
00014     GrafoLista(int ordem, bool direcionado, bool ponderadoVertices, bool ponderadoArestas);
00015     ~GrafoLista();
00016
00017     int getGrau(int vertice) override;
00018     bool ehCompleto() override;
00019     bool ehBipartido() override;
00020     int nConexo() override;
00021     bool ehArvore() override;
00022     bool possuiPonte() override;
00023     bool possuiArticulacao() override;
00024
00025     void carregaGrafo(const std::string& arquivo) override;
00026     void novoGrafo(const std::string& arquivoConfig) override;
00027 };
00028
00029 #endif // GRAFOLISTA_H
```

## 6.7 C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/GrafoMatriz.cpp File Reference

Implementação das funções da classe [GrafoMatriz](#).

```
#include "GrafoMatriz.h"
#include <fstream>
#include <iostream>
```

### 6.7.1 Detailed Description

Implementação das funções da classe [GrafoMatriz](#).

## 6.8 C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/GrafoMatriz.h

```

00001 #ifndef GRAFOMATRIZ_H
00002 #define GRAFOMATRIZ_H
00003
00004 #include "Grafo.h"
00005
00006
00007 class GrafoMatriz : public Grafo {
00008 private:
00009     int** matrizAdj; // Matriz de adjacência
00010     int numVertices;
00011
00012 public:
00013     GrafoMatriz(int ordem, bool direcionado, bool ponderadoVertices, bool ponderadoArestas);
00014     ~GrafoMatriz();
00015
00016     int getGrau(int vertice) override;
00017     bool ehCompleto() override;
00018     bool ehBipartido() override;
00019     int nConexo() override;
00020     bool ehArvore() override;
00021     bool possuiPonte() override;
00022     bool possuiArticulacao() override;
00023
00024     void carregaGrafo(const std::string& arquivo) override;
00025     void novoGrafo(const std::string& arquivoConfig) override;
00026 };
00027
00028 #endif // GRAFOMATRIZ_H

```

## 6.9 C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ListaEncad.cpp File Reference

Implementação das funções da classe [Lista](#).

```

#include "ListaEncad.h"
#include <iostream>

```

### 6.9.1 Detailed Description

Implementação das funções da classe [Lista](#).

## 6.10 C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ListaEncad.h

```

00001 #ifndef LISTA_H
00002 #define LISTA_H
00003
00004 #include "No.h"
00005
00006 class Lista {
00007 private:
00008     No* cabeca;
00009     int tamanho;
00010
00011 public:
00012     Lista();
00013     ~Lista();
00014
00015     void adicionar(int idNo, float pesoNo = 0.0);

```

```

00016     void remover(int idNo);
00017     bool contem(int idNo);
00018     int getTamanho();
00019     No* getElemento(int indice);
00020 };
00021
00022 #endif // LISTA_H

```

## 6.11 C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/No.cpp File Reference

Implementação das funções da classe `No`.

```
#include "No.h"
```

### 6.11.1 Detailed Description

Implementação das funções da classe `No`.

## 6.12 C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/No.h

```

00001 #ifndef NO_H
00002 #define NO_H
00003
00004 #include "Aresta.h"
00005
00006 class No {
00007 private:
00008     int idNo;
00009     float pesoNo;
00010     unsigned int grauEntrada;
00011     unsigned int grauSaida;
00012     Aresta* primeiraAresta;
00013     Aresta* ultimaAresta;
00014     No* proxNo;
00015
00016 public:
00017     No(int idNo, float pesoNo = 0.0);
00018     ~No();
00019
00020     int getIdNo();
00021     float getPesoNo();
00022     unsigned int getGrauEntrada();
00023     unsigned int getGrauSaida();
00024     Aresta* getPrimeiraAresta();
00025     No* getProxNo();
00026
00027     void setProxNo(No* proxNo);
00028     void adicionaAresta(int destino, float peso = 1.0, bool direcionado = false);
00029     void incGrauEntrada();
00030     void incGrauSaida();
00031 };
00032
00033 #endif // NO_H

```



# Index

- ~Aresta
  - Aresta, [12](#)
- ~Grafo
  - Grafo, [15](#)
- ~Lista
  - Lista, [28](#)
- ~No
  - No, [31](#)
- adicionaAresta
  - No, [32](#)
- adicionar
  - Lista, [29](#)
- Aresta, [11](#)
  - ~Aresta, [12](#)
  - Aresta, [11](#)
  - getIdDestino, [12](#)
  - getIdOrigem, [12](#)
  - getPeso, [12](#)
  - getProxAresta, [12](#)
  - setNoDestino, [13](#)
  - setNoOrigem, [13](#)
  - setProxAresta, [13](#)
- arestaPonderada
  - Grafo, [16](#)
- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/Aresta.cpp, [35](#)
- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/Grafo.cpp, [36](#)
- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/GrafoLista.cpp, [36](#)
- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/GrafoMatriz.cpp, [37](#)
- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/ListaEncad.cpp, [38](#)
- C:/Users/Felipe Sant'Anna/OneDrive/Desktop/Teoria-dos-Grafos/No.cpp, [39](#)
- carregaGrafo
  - Grafo, [16](#)
  - GrafoLista, [21](#)
  - GrafoMatriz, [25](#)
- contem
  - Lista, [29](#)
- ehArvore
  - Grafo, [16](#)
  - GrafoLista, [21](#)
  - GrafoMatriz, [26](#)
- ehBipartido
  - Grafo, [16](#)
  - GrafoLista, [22](#)
  - GrafoMatriz, [26](#)
- ehCompleto
  - Grafo, [16](#)
  - GrafoLista, [22](#)
  - GrafoMatriz, [26](#)
- ehDirecionado
  - Grafo, [17](#)
- getElemento
  - Lista, [29](#)
- getGrau
  - Grafo, [17](#)
  - GrafoLista, [22](#)
  - GrafoMatriz, [26](#)
- getGrauEntrada
  - No, [32](#)
- getGrauSaida
  - No, [32](#)
- getIdDestino
  - Aresta, [12](#)
- getIdNo
  - No, [32](#)
- getIdOrigem
  - Aresta, [12](#)
- getNoPeloid
  - Grafo, [17](#)
- getNumNos
  - Grafo, [18](#)
- getOrdem
  - Grafo, [18](#)
- getPeso
  - Aresta, [12](#)
- getPesoNo
  - No, [33](#)
- getPrimeiraAresta
  - No, [33](#)
- getProxAresta
  - Aresta, [12](#)
- getProxNo
  - No, [33](#)
- getTamanho
  - Lista, [30](#)
- Grafo, [14](#)
  - ~Grafo, [15](#)
  - arestaPonderada, [16](#)
  - carregaGrafo, [16](#)
  - ehArvore, [16](#)
  - ehBipartido, [16](#)

- ehCompleto, 16
- ehDirecionado, 17
- getGrau, 17
- getNoPelold, 17
- getNumNos, 18
- getOrdem, 18
- Grafo, 15
- nConexo, 18
- novoGrafo, 18
- possuiArticulacao, 18
- possuiPonte, 19
- verticePonderado, 19
- GrafoLista, 19
  - carregaGrafo, 21
  - ehArvore, 21
  - ehBipartido, 22
  - ehCompleto, 22
  - getGrau, 22
  - GrafoLista, 21
  - nConexo, 22
  - novoGrafo, 23
  - possuiArticulacao, 23
  - possuiPonte, 23
- GrafoMatriz, 24
  - carregaGrafo, 25
  - ehArvore, 26
  - ehBipartido, 26
  - ehCompleto, 26
  - getGrau, 26
  - GrafoMatriz, 25
  - nConexo, 27
  - novoGrafo, 27
  - possuiArticulacao, 27
  - possuiPonte, 27
- incGrauEntrada
  - No, 33
- incGrauSaida
  - No, 34
- Lista, 28
  - ~Lista, 28
  - adicionar, 29
  - contem, 29
  - getElemento, 29
  - getTamanho, 30
  - Lista, 28
  - remover, 30
- nConexo
  - Grafo, 18
  - GrafoLista, 22
  - GrafoMatriz, 27
- No, 31
  - ~No, 31
  - adicionaAresta, 32
  - getGrauEntrada, 32
  - getGrauSaida, 32
  - getIdNo, 32
  - getPesoNo, 33
  - getPrimeiraAresta, 33
  - getProxNo, 33
  - incGrauEntrada, 33
  - incGrauSaida, 34
  - No, 31
  - setProxNo, 34
- novoGrafo
  - Grafo, 18
  - GrafoLista, 23
  - GrafoMatriz, 27
- possuiArticulacao
  - Grafo, 18
  - GrafoLista, 23
  - GrafoMatriz, 27
- possuiPonte
  - Grafo, 19
  - GrafoLista, 23
  - GrafoMatriz, 27
- Projeto de Implementação de Grafos, 1
- remover
  - Lista, 30
- setNoDestino
  - Aresta, 13
- setNoOrigem
  - Aresta, 13
- setProxAresta
  - Aresta, 13
- setProxNo
  - No, 34
- verticePonderado
  - Grafo, 19