Trabalho-de-Grafos

Gerado por Doxygen 1.13.1

Capítulo 1

Índice Hierárquico

1.1 Hierarquia de Classes

Esta lista de hierarquias está parcialmente ordenada (ordem alfabética):

Capítulo 2

Índice dos Componentes

2.1 Lista de Classes

Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:

Capítulo 3

Índice dos Arquivos

3.1 Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

Capítulo 4

Classes

4.1 Referência da Classe aresta_grafo

`#include <aresta_grafo.h>`

Membros Públicos

- **aresta_grafo** (int **destino**, int **peso**=0)
  *Construtor da classe **aresta_grafo** (p. **??**).*
- ∼**aresta_grafo** ()
  *Destrutor da classe **aresta_grafo** (p. **??**).*

Atributos Públicos

- int **destino**
- int **peso**
- **aresta_grafo** ∗ **proxima**

4.1.1 Construtores e Destrutores

4.1.1.1 aresta_grafo()

```
aresta_grafo::aresta_grafo (
            int destino,
            int peso = 0)
```

Construtor da classe **aresta_grafo** (p. **??**).

Parâmetros

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

O ponteiro para a próxima aresta é inicializado como nullptr.

```
00014                                                :
00015     destino(destino),
00016     peso(peso),
00017     proxima(nullptr)
00018 {}
```

4.1.1.2 ∼aresta_grafo()

```
aresta_grafo::∼aresta_grafo ()  [default]
```

Destrutor da classe **aresta_grafo** (p. **??**).

4.1.2 Atributos

4.1.2.1 destino

```
int aresta_grafo::destino
```

4.1.2.2 peso

```
int aresta_grafo::peso
```

4.1.2.3 proxima

```
aresta_grafo* aresta_grafo::proxima
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/include/ **aresta_grafo.h**
- C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/src/ **aresta_grafo.cpp**

4.2 Referência da Classe grafo

```
#include <grafo.h>
```

Diagrama de hierarquia da classe grafo:

Membros Públicos

- **grafo** ()
- virtual ∼**grafo** ()=default
- virtual **no_grafo** ∗ **get_no** (int id)=0
- virtual **aresta_grafo** ∗ **get_aresta** (int origem, int destino)=0
- virtual **aresta_grafo** ∗ **get_vizinhos** (int id)=0
- virtual int **get_ordem** ()=0
- virtual bool **existe_aresta** (int origem, int destino)=0
- int **get_grau** ()

  Retorna o grau do grafo.
- bool **eh_completo** ()

  Verifica se o grafo é completo.
- bool **eh_direcionado** () const

  Funções que retornam as flags: direcionado, ponderado_vertices e ponderado_arestas.
- bool **vertice_ponderado** () const
- bool **aresta_ponderada** () const
- void **carrega_grafo** (const std::string &arquivo)

  Constroi o grafo a partir de um arquivo.
- void **exibe_descricao** ()

  Exibe a descrição do grafo.
- virtual void **add_no** (int id, int peso)=0
- virtual void **add_aresta** (int origem, int destino, int peso)=0

Atributos Protegidos

- bool **direcionado**
- bool **ponderado_vertices**
- bool **ponderado_arestas**
- int **num_nos**

4.2.1 Construtores e Destrutores

4.2.1.1 grafo()

```
grafo::grafo ()
00012 {}
```

4.2.1.2 ∼grafo()

```
virtual grafo::∼grafo ()  [virtual], [default]
```

4.2.2 Documentação das funções

4.2.2.1 add_aresta()

```
virtual void grafo::add_aresta (
            int origem,
            int destino,
            int peso)  [pure virtual]
```

Implementado por **grafo_lista** (p. **??**) e **grafo_matriz** (p. **??**).

4.2.2.2 add_no()

```
virtual void grafo::add_no (
            int id,
            int peso)  [pure virtual]
```

Implementado por **grafo_lista** (p. **??**) e **grafo_matriz** (p. **??**).

4.2.2.3 aresta_ponderada()

```
bool grafo::aresta_ponderada () const
00125 { return ponderado_arestas; }
```

4.2.2.4 carrega_grafo()

```
void grafo::carrega_grafo (
            const std::string & arquivo)
```

Constroi o grafo a partir de um arquivo.

Parâmetros

arquivo	O caminho para o arquivo contendo a descrição do grafo.

```
00018                                                   {
00019
00020     std::ifstream file(arquivo);
00021     if (!file.is_open()) throw std::runtime_error("Arquivo não encontrado");
00022
00023     int num_nos, dir, pond_vertices, pond_arestas;
00024     file » num_nos » dir » pond_vertices » pond_arestas;
00025
00026     this->direcionado = dir;
00027     this->ponderado_vertices = pond_vertices;
00028     this->ponderado_arestas = pond_arestas;
00029     this->num_nos = num_nos;
00030
00031     if (ponderado_vertices) {
00032         for (int i = 1; i <= num_nos; ++i) {
00033             int peso;
00034             file » peso;
00035             add_no(i, peso);
00036         }
00037     } else {
00038         for (int i = 1; i <= num_nos; ++i) {
00039             add_no(i, 0);
00040         }
00041     }
00042
00043     int origem, destino, peso = 0;
00044     while (file » origem » destino) {
00045         if (ponderado_arestas) file » peso;
00046         add_aresta(origem, destino, peso);
00047     }
00048 }
```

4.2.2.5 eh_completo()

```
bool grafo::eh_completo ()
```

Verifica se o grafo é completo.

Retorna

true se o grafo é completo, false caso contrário.

```
00054                         {
00055     int n = get_ordem();
00056     for (int i = 1; i <= n; ++i) {
00057         for (int j = 1; j <= n; ++j) {
00058             if (i != j && !existe_aresta(i, j)) {
00059                 if (direcionado) return false;
00060                 if (!existe_aresta(j, i)) return false;
00061             }
00062         }
00063     }
00064     return true;
00065 }
```

4.2.2.6 eh_direcionado()

```
bool grafo::eh_direcionado () const
```

Funções que retornam as flags: direcionado, ponderado_vertices e ponderado_arestas.

```
00123 { return direcionado; }
```

4.2.2.7 exibe_descricao()

```
void grafo::exibe_descricao ()
```

Exibe a descrição do grafo.

```
00111                             {
00112     std::cout « "Grau: " « get_grau() « std::endl;
00113     std::cout « "Ordem: " « get_ordem() « std::endl;
00114     std::cout « "Direcionado: " « (eh_direcionado() ? "Sim" : "Nao") « std::endl;
00115     std::cout « "Vertices ponderados: " « (vertice_ponderado() ? "Sim" : "Nao") « std::endl;
00116     std::cout « "Arestas ponderadas: " « (aresta_ponderada() ? "Sim" : "Nao") « std::endl;
00117     std::cout « "Completo: " « (eh_completo() ? "Sim" : "Nao") « std::endl;
00118 }
```

4.2.2.8 existe_aresta()

```
virtual bool grafo::existe_aresta (
             int origem,
             int destino)  [pure virtual]
```

Implementado por **grafo_lista** (p. **??**) e **grafo_matriz** (p. **??**).

4.2.2.9 get_aresta()

```
virtual  aresta_grafo * grafo::get_aresta (
             int origem,
             int destino)  [pure virtual]
```

Implementado por **grafo_lista** (p. **??**) e **grafo_matriz** (p. **??**).

4.2.2.10 get_grau()

```
int grafo::get_grau ()
```

Retorna o grau do grafo.

Retorna

O grau do grafo.

```
00083                     {
00084     int grau_maximo = 0;
00085     for (int i = 1; i <= get_ordem(); ++i) {
00086         int grau_atual = 0;
00087 
00088         aresta_grafo* vizinhos = get_vizinhos(i);
00089         aresta_grafo* atual = vizinhos;
00090         while (atual) {
00091             grau_atual++;
00092             atual = atual->proxima;
00093         }
00094 
00095         liberar_arestas_temp(vizinhos);
00096 
00097         if (direcionado) {
00098             for (int j = 1; j <= get_ordem(); ++j) {
00099                 if (existe_aresta(j, i)) grau_atual++;
00100             }
00101         }
00102 
00103         if (grau_atual > grau_maximo) grau_maximo = grau_atual;
00104     }
00105     return grau_maximo;
00106 }
```

4.2.2.11 get_no()

```
virtual  no_grafo * grafo::get_no (
             int id)  [pure virtual]
```

Implementado por **grafo_lista** (p. **??**) e **grafo_matriz** (p. **??**).

4.2.2.12 get_ordem()

```
virtual int grafo::get_ordem ()  [pure virtual]
```

Implementado por **grafo_lista** (p. **??**) e **grafo_matriz** (p. **??**).

4.2.2.13 get_vizinhos()

```
virtual  aresta_grafo * grafo::get_vizinhos (
             int id)  [pure virtual]
```

Implementado por **grafo_lista** (p. **??**) e **grafo_matriz** (p. **??**).

4.2.2.14 vertice_ponderado()

```
bool grafo::vertice_ponderado () const
00124 { return ponderado_vertices; }
```

4.2.3 Atributos

4.2.3.1 direcionado

```
bool grafo::direcionado  [protected]
```

4.2.3.2 num_nos

```
int grafo::num_nos  [protected]
```

4.2.3.3 ponderado_arestas

```
bool grafo::ponderado_arestas  [protected]
```

4.2.3.4 ponderado_vertices

```
bool grafo::ponderado_vertices  [protected]
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/include/ **grafo.h**
- C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/src/ **grafo.cpp**

4.3 Referência da Classe grafo_lista

```
#include <grafo_lista.h>
```

Diagrama de hierarquia da classe grafo_lista:

Membros Públicos

- **grafo_lista** ()
  *Construtor da classe **grafo_lista** (*p. **??**).*
- ∼**grafo_lista** () override
  *Destrutor da classe **grafo_lista** (*p. **??**).*
- **no_grafo** ∗ **get_no** (int id) override
  Retorna um nó do grafo.
- **aresta_grafo** ∗ **get_aresta** (int origem, int destino) override
  Retorna uma aresta do grafo.
- **aresta_grafo** ∗ **get_vizinhos** (int id) override
  Retorna as arestas que saem de um nó.
- int **get_ordem** () override
  Retorna a ordem do grafo.
- bool **existe_aresta** (int origem, int destino) override
  Verifica se uma aresta existe no grafo.
- void **add_no** (int id, int peso) override
  Adiciona um nó ao grafo.
- void **add_aresta** (int origem, int destino, int peso) override
  Adiciona uma aresta ao grafo.

Membros Públicos herdados de grafo

- **grafo** ()
- virtual ~**grafo** ()=default
- int **get_grau** ()

  Retorna o grau do grafo.
- bool **eh_completo** ()

  Verifica se o grafo é completo.
- bool **eh_direcionado** () const

  Funções que retornam as flags: direcionado, ponderado_vertices e ponderado_arestas.
- bool **vertice_ponderado** () const
- bool **aresta_ponderada** () const
- void **carrega_grafo** (const std::string &arquivo)

  Constroi o grafo a partir de um arquivo.
- void **exibe_descricao** ()

  Exibe a descrição do grafo.

Outros membros herdados

Atributos Protegidos herdados de grafo

- bool **direcionado**
- bool **ponderado_vertices**
- bool **ponderado_arestas**
- int **num_nos**

4.3.1 Construtores e Destrutores

4.3.1.1 grafo_lista()

```
grafo_lista::grafo_lista ()
```

Construtor da classe **grafo_lista** (p. **??**).

O ponteiro para o primeiro nó é inicializado como nullptr.

```
00012 : primeiro_no(nullptr) {}
```

4.3.1.2 ~grafo_lista()

```
grafo_lista::~grafo_lista ()  [override]
```

Destrutor da classe **grafo_lista** (p. **??**).

Deleta todos os nós e arestas do grafo.

```
00018                           {
00019     no_grafo* atual = primeiro_no;
00020     while (atual) {
00021         no_grafo* proximo = atual->proximo;
00022         delete atual;
00023         atual = proximo;
00024     }
00025 }
```

4.3.2 Documentação das funções

4.3.2.1 add_aresta()

```
void grafo_lista::add_aresta (
            int origem,
            int destino,
            int peso)  [override], [virtual]
```

Adiciona uma aresta ao grafo.

Parâmetros

origem	O id do nó de origem da aresta.
destino	O id do nó de destino da aresta.
peso	O peso da aresta.

Implementa **grafo** (p. **??**).

```
00133                                                                {
00134     if (origem == destino) return;
00135
00136     no_grafo* no_origem = get_no(origem);
00137     no_grafo* no_destino = get_no(destino);
00138
00139     if (!no_origem || !no_destino) return;
00140
00141     if (existe_aresta(origem, destino)) return;
00142
00143     aresta_grafo* nova_aresta = new aresta_grafo(destino, peso);
00144     nova_aresta->proxima = no_origem->primeira_aresta;
00145     no_origem->primeira_aresta = nova_aresta;
00146
00147     if (!direcionado) {
00148         aresta_grafo* aresta_inversa = new aresta_grafo(origem, peso);
00149         aresta_inversa->proxima = no_destino->primeira_aresta;
00150         no_destino->primeira_aresta = aresta_inversa;
00151     }
00152 }
```

4.3.2.2 add_no()

```
void grafo_lista::add_no (
            int id,
            int peso)  [override], [virtual]
```

Adiciona um nó ao grafo.

Parâmetros

id	O id do nó.
peso	O peso do nó.

Implementa **grafo** (p. **??**).

```
00119                                       {
00120     if (get_no(id)) return;
00121
00122     no_grafo* novo_no = new no_grafo(id, peso);
00123     novo_no->proximo = primeiro_no;
00124     primeiro_no = novo_no;
00125 }
```

4.3.2.3 existe_aresta()

```
bool grafo_lista::existe_aresta (
            int origem,
            int destino)  [override], [virtual]
```

Verifica se uma aresta existe no grafo.

Parâmetros

origem	O id do nó de origem da aresta.
destino	O id do nó de destino da aresta.

Retorna

true se a aresta existe, false caso contrário.

Implementa **grafo** (p. **??**).

```
00110                                                   {
00111     return get_aresta(origem, destino) != nullptr;
00112 }
```

4.3.2.4 get_aresta()

```
aresta_grafo * grafo_lista::get_aresta (
            int origem,
            int destino)  [override], [virtual]
```

Retorna uma aresta do grafo.

Parâmetros

origem	O id do nó de origem da aresta.
destino	O id do nó de destino da aresta.

Retorna

A aresta que vai do nó de origem para o nó de destino, ou nullptr se ela não existir.

Implementa **grafo** (p. **??**).

```
00047                                                                   {
00048     no_grafo* no_origem = get_no(origem);
00049     if (!no_origem) return nullptr;
00050
00051     aresta_grafo* atual = no_origem->primeira_aresta;
00052     while (atual) {
00053         if (atual->destino == destino) return atual;
00054         atual = atual->proxima;
00055     }
00056     return nullptr;
00057 }
```

4.3.2.5 get_no()

```
no_grafo * grafo_lista::get_no (
            int id)  [override], [virtual]
```

Retorna um nó do grafo.

Parâmetros

id	O id do nó a ser retornado.

Retorna

O nó com o id especificado, ou nullptr se ele não existir.

Implementa **grafo** (p. **??**).

```
00032                                       {
00033     no_grafo* atual = primeiro_no;
00034     while (atual) {
00035         if (atual->id == id) return atual;
00036         atual = atual->proximo;
00037     }
00038     return nullptr;
00039 }
```

4.3.2.6 get_ordem()

```
int grafo_lista::get_ordem ()  [override], [virtual]
```

Retorna a ordem do grafo.

Retorna

O número de nós do grafo.

Implementa **grafo** (p. **??**).

```
00094                               {
00095     int count = 0;
00096     no_grafo* atual = primeiro_no;
00097     while (atual) {
00098         count++;
00099         atual = atual->proximo;
00100     }
00101     return count;
00102 }
```

4.3.2.7 get_vizinhos()

```
aresta_grafo * grafo_lista::get_vizinhos (
            int id)  [override], [virtual]
```

Retorna as arestas que saem de um nó.

Parâmetros

id	O id do nó.

Retorna

Um ponteiro para a primeira aresta que sai do nó, ou nullptr se ele não existir.

Implementa **grafo** (p. **??**).

```
00064                                                {
00065     no_grafo* no = get_no(id);
00066     if (!no) return nullptr;
00067
00068     aresta_grafo* cabeca = nullptr;
00069     aresta_grafo* atual = nullptr;
00070
00071     aresta_grafo* aresta_original = no->primeira_aresta;
00072     while (aresta_original) {
00073         // Cria uma cópia da aresta original
00074         aresta_grafo* copia = new aresta_grafo(aresta_original->destino, aresta_original->peso);
00075
00076         if (!cabeca) {
00077             cabeca = copia;
00078             atual = cabeca;
00079         } else {
00080             atual->proxima = copia;
00081             atual = atual->proxima;
00082         }
00083
00084         aresta_original = aresta_original->proxima;
00085     }
00086
00087     return cabeca;
00088 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/include/ **grafo_lista.h**
- C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/src/ **grafo_lista.cpp**

4.4 Referência da Classe grafo_matriz

`#include <grafo_matriz.h>`

Diagrama de hierarquia da classe grafo_matriz:

Membros Públicos

- **grafo_matriz** ()

  *Construtor da classe **grafo_matriz** (*p. **??**).
- ∼**grafo_matriz** () override

  *Destrutor da classe **grafo_matriz** (*p. **??**).
- **no_grafo** ∗ **get_no** (int id) override

  Retorna um nó do grafo.
- **aresta_grafo** ∗ **get_aresta** (int origem, int destino) override

  Retorna uma aresta do grafo.
- **aresta_grafo** ∗ **get_vizinhos** (int id) override

  Retorna as arestas que saem de um nó.

- int **get_ordem** () override

  Retorna a ordem do grafo.
- bool **existe_aresta** (int origem, int destino) override

  Verifica se uma aresta existe no grafo.
- void **add_no** (int id, int peso) override

  Adiciona um nó ao grafo.
- void **add_aresta** (int origem, int destino, int peso) override

  Adiciona uma aresta ao grafo.

Membros Públicos herdados de grafo

- **grafo** ()
- virtual ~**grafo** ()=default
- int **get_grau** ()

  Retorna o grau do grafo.
- bool **eh_completo** ()

  Verifica se o grafo é completo.
- bool **eh_direcionado** () const

  Funções que retornam as flags: direcionado, ponderado_vertices e ponderado_arestas.
- bool **vertice_ponderado** () const
- bool **aresta_ponderada** () const
- void **carrega_grafo** (const std::string &arquivo)

  Constroi o grafo a partir de um arquivo.
- void **exibe_descricao** ()

  Exibe a descrição do grafo.

Outros membros herdados

Atributos Protegidos herdados de grafo

- bool **direcionado**
- bool **ponderado_vertices**
- bool **ponderado_arestas**
- int **num_nos**

4.4.1 Construtores e Destrutores

4.4.1.1 grafo_matriz()

```
grafo_matriz::grafo_matriz ()
```

Construtor da classe **grafo_matriz** (p. **??**).

Inicializa a matriz de adjacência como nullptr e a flag de inicialização como false.

```
00013                           : matriz(nullptr), matriz_inicializada(false) {
00014     num_nos = 0;
00015 }
```

4.4.1.2 ~grafo_matriz()

```
grafo_matriz::~grafo_matriz ()  [override]
```

Destrutor da classe **grafo_matriz** (p. **??**).

Deleta a matriz de adjacência e todas as arestas.

```
00021                                {
00022     if (matriz_inicializada) {
00023         for (int i = 0; i < num_nos; ++i) {
00024             for (int j = 0; j < num_nos; ++j) {
00025                 delete matriz[i][j];
00026             }
00027             delete[] matriz[i];
00028         }
00029         delete[] matriz;
00030         matriz_inicializada = false;
00031     }
00032 }
```

4.4.2 Documentação das funções

4.4.2.1 add_aresta()

```
void grafo_matriz::add_aresta (
             int origem,
             int destino,
             int peso)  [override], [virtual]
```

Adiciona uma aresta ao grafo.

Parâmetros

origem	O id do nó de origem da aresta.
destino	O id do nó de destino da aresta.
peso	O peso da aresta.

Implementa **grafo** (p. **??**).

```
00128                                                                {
00129     if (origem == destino) return;
00130
00131     int i = origem - 1;
00132     int j = destino - 1;
00133
00134     if (i >= 0 && i < num_nos && j >= 0 && j < num_nos && !matriz[i][j]) {
00135         matriz[i][j] = new aresta_grafo(destino, peso);
00136
00137         if (!direcionado && origem != destino) {
00138             matriz[j][i] = new aresta_grafo(origem, peso);
00139         }
00140     }
00141 }
```

4.4.2.2 add_no()

```
void grafo_matriz::add_no (
             int id,
             int peso)  [override], [virtual]
```

Adiciona um nó ao grafo.

Parâmetros

id	O id do nó a ser adicionado.
peso	O peso do nó a ser adicionado.

Implementa **grafo** (p. **??**).

```
00109                                       {
00110     if (!matriz_inicializada && num_nos > 0) {
00111         matriz = new aresta_grafo**[num_nos];
00112         for (int i = 0; i < num_nos; ++i) {
00113             matriz[i] = new aresta_grafo*[num_nos];
00114             for (int j = 0; j < num_nos; ++j) {
00115                 matriz[i][j] = nullptr;
00116             }
00117         }
00118         matriz_inicializada = true;
00119     }
00120 }
```

4.4.2.3 existe_aresta()

```
bool grafo_matriz::existe_aresta (
            int origem,
            int destino)  [override], [virtual]
```

Verifica se uma aresta existe no grafo.

Parâmetros

origem	O id do nó de origem da aresta.
destino	O id do nó de destino da aresta.

Retorna

true se a aresta existe, false caso contrário.

Implementa **grafo** (p. **??**).

```
00100                                                   {
00101     return get_aresta(origem, destino) != nullptr;
00102 }
```

4.4.2.4 get_aresta()

```
aresta_grafo * grafo_matriz::get_aresta (
            int origem,
            int destino)  [override], [virtual]
```

Retorna uma aresta do grafo.

Parâmetros

origem	O id do nó de origem da aresta.
destino	O id do nó de destino da aresta.

Retorna

A aresta que vai do nó de origem para o nó de destino, ou nullptr se ela não existir.

Implementa **grafo** (p. **??**).

```
00049                                                                   {
00050     if (origem < 1 || origem > num_nos || destino < 1 || destino > num_nos)
00051         return nullptr;
00052
00053     if (!direcionado && origem > destino)
00054         std::swap(origem, destino);
00055
00056     return matriz[origem-1][destino-1];
00057 }
```

4.4.2.5 get_no()

```
no_grafo * grafo_matriz::get_no (
            int id)  [override], [virtual]
```

Retorna um nó do grafo.

Parâmetros

id	O id do nó a ser retornado.

Retorna

O nó com o id especificado, ou nullptr se ele não existir.

Implementa **grafo** (p. **??**).

```
00039                                       {
00040     return nullptr;
00041 }
```

4.4.2.6 get_ordem()

```
int grafo_matriz::get_ordem ()  [override], [virtual]
```

Retorna a ordem do grafo.

Retorna

O número de nós do grafo.

Implementa **grafo** (p. **??**).

```
00090                               {
00091     return num_nos;
00092 }
```

4.4.2.7 get_vizinhos()

```
aresta_grafo * grafo_matriz::get_vizinhos (
            int id)  [override], [virtual]
```

Retorna as arestas que saem de um nó.

Parâmetros

id	O id do nó.

Retorna

Um ponteiro para a primeira aresta que sai do nó, ou nullptr se ele não existir.

Implementa **grafo** (p. **??**).

```
00064                                                {
00065     if (id < 1 || id > num_nos) return nullptr;
00066
00067     aresta_grafo* cabeca = nullptr;
00068     aresta_grafo* atual = nullptr;
00069
00070     for (int j = 0; j < num_nos; ++j) {
00071         if (matriz[id-1][j] != nullptr) {
00072             aresta_grafo* nova_aresta = new aresta_grafo(matriz[id-1][j]->destino,
      matriz[id-1][j]->peso);
00073
00074             if (!cabeca) {
00075                 cabeca = nova_aresta;
00076                 atual = cabeca;
00077             } else {
00078                 atual->proxima = nova_aresta;
00079                 atual = atual->proxima;
00080             }
00081         }
00082     }
00083     return cabeca;
00084 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/include/ **grafo_matriz.h**
- C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/src/ **grafo_matriz.cpp**

4.5 Referência da Classe no_grafo

`#include <no_grafo.h>`

Membros Públicos

- **no_grafo** (int **id**, int **peso**=0)
  *Construtor da classe **no_grafo** (p. **??**).*
- ∼**no_grafo** ()
  *Destrutor da classe **no_grafo** (p. **??**).*

Atributos Públicos

- int **id**
- int **peso**
- **aresta_grafo** ∗ **primeira_aresta**
- **no_grafo** ∗ **proximo**

4.5.1 Construtores e Destrutores

4.5.1.1 no_grafo()

```
no_grafo::no_grafo (
            int id,
            int peso = 0)
```

Construtor da classe **no_grafo** (p. **??**).

Parâmetros

id	O id do nó.
peso	O peso do nó.

O ponteiro para a primeira aresta é inicializado como nullptr.

```
00015                                   :
00016     id(id),
00017     peso(peso),
00018     primeira_aresta(nullptr),
00019     proximo(nullptr)
00020 {}
```

4.5.1.2 ∼no_grafo()

```
no_grafo::∼no_grafo ()
```

Destrutor da classe **no_grafo** (p. **??**).

Deleta todas as arestas do nó.

```
00026                     {
00027     aresta_grafo* atual = primeira_aresta;
00028     while (atual) {
00029         aresta_grafo* temp = atual;
00030         atual = atual->proxima;
00031         delete temp;
00032     }
00033 }
```

4.5.2 Atributos

4.5.2.1 id

```
int no_grafo::id
```

4.5.2.2 peso

```
int no_grafo::peso
```

4.5.2.3 primeira_aresta

```
aresta_grafo* no_grafo::primeira_aresta
```

4.5.2.4 proximo

```
no_grafo* no_grafo::proximo
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/include/ **no_grafo.h**
- C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/src/ **no_grafo.cpp**

Capítulo 5

Arquivos

5.1 Referência do Arquivo C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/include/aresta_grafo.h

Classe que representa uma aresta de um grafo.

Componentes

- class **aresta_grafo**

5.1.1 Descrição detalhada

Classe que representa uma aresta de um grafo.

Cada aresta possui um destino, que é o vértice para o qual ela aponta, um peso, que é o custo para se chegar ao vértice de destino, e um ponteiro para a próxima aresta.

5.2 aresta_grafo.h

Ir para a documentação desse arquivo.

```
00001 #ifndef ARESTA_GRAFO_H
00002 #define ARESTA_GRAFO_H
00003
00009 class aresta_grafo {
00010 public:
00011     int destino;
00012     int peso;
00013     aresta_grafo* proxima;
00014
00015     aresta_grafo(int destino, int peso = 0);
00016
00017     ~aresta_grafo();
00018 };
00019
00020 #endif // ARESTA_GRAFO_H
```

5.3 Referência do Arquivo C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/include/grafo.h

Classe abstrata que define as operações que podem ser realizadas em um grafo.

```
#include <string>
#include "no_grafo.h"
#include "aresta_grafo.h"
```

Componentes

- class **grafo**

5.3.1 Descrição detalhada

Classe abstrata que define as operações que podem ser realizadas em um grafo.

Essa classe possui duas filhas: **grafo_matriz** (p. **??**) e grafos_lista, que implementam as operações definidas aqui.

5.4 grafo.h

Ir para a documentação desse arquivo.

```
00001 #ifndef GRAFO_H
00002 #define GRAFO_H
00003 #include <string>
00004 #include "no_grafo.h"
00005 #include "aresta_grafo.h"
00006
00012 class grafo {
00013 protected:
00014     bool direcionado;
00015     bool ponderado_vertices;
00016     bool ponderado_arestas;
00017     int num_nos;
00018
00019 public:
00020     grafo();
00021     virtual ~grafo() = default;
00022
00023     virtual no_grafo* get_no(int id) = 0;
00024     virtual aresta_grafo* get_aresta(int origem, int destino) = 0;
00025     virtual aresta_grafo* get_vizinhos(int id) = 0;
00026     virtual int get_ordem() = 0;
00027     virtual bool existe_aresta(int origem, int destino) = 0;
00028
00029     int get_grau();
00030     bool eh_completo();
00031     bool eh_direcionado() const;
00032     bool vertice_ponderado() const;
00033     bool aresta_ponderada() const;
00034     void carrega_grafo(const std::string& arquivo);
00035
00036     void exibe_descricao();
00037
00038     virtual void add_no(int id, int peso) = 0;
00039     virtual void add_aresta(int origem, int destino, int peso) = 0;
00040 };
00041
00042 #endif //GRAFO_H
```

5.5 Referência do Arquivo C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/include/grafo_lista.h

Classe que representa um grafo implementado com listas de adjacência.

```
#include "grafo.h"
#include "no_grafo.h"
```

Componentes

- class **grafo_lista**

5.5.1 Descrição detalhada

Classe que representa um grafo implementado com listas de adjacência.

Cada nó do grafo possui um id e um peso, e cada aresta possui um destino, um peso e um ponteiro para a próxima aresta.

5.6 grafo_lista.h

Ir para a documentação desse arquivo.

```
00001 #ifndef GRAFO_LISTA_H
00002 #define GRAFO_LISTA_H
00003 #include "grafo.h"
00004 #include "no_grafo.h"
00005
00011 class grafo_lista : public grafo {
00012 private:
00013     no_grafo* primeiro_no;
00014
00015 public:
00016     grafo_lista();
00017     ~grafo_lista() override;
00018
00019     no_grafo* get_no(int id) override;
00020     aresta_grafo* get_aresta(int origem, int destino) override;
00021     aresta_grafo* get_vizinhos(int id) override;
00022     int get_ordem() override;
00023     bool existe_aresta(int origem, int destino) override;
00024
00025
00026     void add_no(int id, int peso) override;
00027     void add_aresta(int origem, int destino, int peso) override;
00028 };
00029
00030 #endif // GRAFO_LISTA_H
```

5.7 Referência do Arquivo C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/include/grafo_matriz.h

Classe que representa um grafo implementado com matriz de adjacência.

```
#include "grafo.h"
```

Componentes

- class **grafo_matriz**

5.7.1 Descrição detalhada

Classe que representa um grafo implementado com matriz de adjacência.

Cada nó do grafo possui um id e um peso, e cada aresta possui um destino, um peso e um ponteiro para a próxima aresta.

5.8 grafo_matriz.h

Ir para a documentação desse arquivo.

```
00001 #ifndef GRAFO_MATRIZ_H
00002 #define GRAFO_MATRIZ_H
00003
00004 #include "grafo.h"
00005
00011 class grafo_matriz : public grafo {
00012 private:
00013     aresta_grafo*** matriz;
00014     bool matriz_inicializada;
00015
00016 public:
00017     grafo_matriz();
00018     ~grafo_matriz() override;
00019
00020     no_grafo* get_no(int id) override;
00021     aresta_grafo* get_aresta(int origem, int destino) override;
00022     aresta_grafo* get_vizinhos(int id) override;
00023     int get_ordem() override;
00024     bool existe_aresta(int origem, int destino) override;
00025
00026     void add_no(int id, int peso) override;
00027     void add_aresta(int origem, int destino, int peso) override;
00028 };
00029
00030 #endif // GRAFO_MATRIZ_H
```

5.9 Referência do Arquivo C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/include/no_grafo.h

Classe que representa um nó de um grafo.

```
#include "aresta_grafo.h"
```

Componentes

- class **no_grafo**

5.9.1 Descrição detalhada

Classe que representa um nó de um grafo.

Cada nó possui um id, um peso e um ponteiro para a primeira aresta que parte dele.

5.10 no_grafo.h

Ir para a documentação desse arquivo.

```
00001 #ifndef NO_GRAFO_H
00002 #define NO_GRAFO_H
00003
00004 #include "aresta_grafo.h"
00005
00011 class no_grafo {
00012 public:
00013     int id;
00014     int peso;
00015     aresta_grafo* primeira_aresta;
00016     no_grafo* proximo;
00017
00018     no_grafo(int id, int peso = 0);
00019     ~no_grafo();
00020 };
00021
00022 #endif // NO_GRAFO_H
```

5.11 Referência do Arquivo C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/main.cpp

Arquivo principal do programa.

```
#include <iostream>
#include <string>
#include <fstream>
#include "include/grafo_lista.h"
#include "include/grafo_matriz.h"
```

Funções

- void **exibir_uso** ()

  Função para exibir como utilizar o programa.
- bool **validar_argumentos** (int argc, char ∗argv[ ])

  Função para validar os argumentos fornecidos pelo usuário.
- int **main** (int argc, char ∗argv[ ])

  Função principal do programa.

5.11.1 Descrição detalhada

Arquivo principal do programa.

5.11.2 Funções

5.11.2.1 exibir_uso()

```
void exibir_uso ()
```

Função para exibir como utilizar o programa.

Caso o usuário não forneça argumentos válidos, exibe como utilizar o programa

```
00016                 {
00017     std::cout « "Uso:\n";
00018     std::cout « "Caso 1: main.out -d -m grafo.txt\n";
00019     std::cout « "Caso 2: main.out -d -l grafo.txt\n";
00020 }
```

5.11.2.2 main()

```
int main (
              int argc,
              char * argv[])
```

Função principal do programa.

Parâmetros

argc	Número de argumentos
argv	Vetor de argumentos

Retorna

0 se o programa foi executado com sucesso, 1 caso contrário

```
00048                                   {
00049     if (!validar_argumentos(argc, argv)) {
00050         return 1;
00051     }
00052
00053     const std::string modo = argv[1];
00054     const std::string estrutura = argv[2];
00055
00056     if (modo == "-d") {
00057         const std::string arquivo = argv[3];
00058         if (estrutura == "-m") {
00059             grafo_matriz grafo;
00060             grafo.carrega_grafo(arquivo);
00061             grafo.exibe_descricao();
00062         } else if (estrutura == "-l") {
00063             grafo_lista grafo;
00064             grafo.carrega_grafo(arquivo);
00065             grafo.exibe_descricao();
00066         } else {
00067             exibir_uso();
00068             return 1;
00069         }
00070     }
00071     else {
00072         exibir_uso();
00073         return 1;
00074     }
00075     return 0;
00076 }
```

5.11.2.3 validar_argumentos()

```
bool validar_argumentos (
              int argc,
              char * argv[])
```

Função para validar os argumentos fornecidos pelo usuário.

Parâmetros

argc	Número de argumentos
argv	Vetor de argumentos

Retorna

true: se os argumentos são válidos, false: caso contrário

```
00028                                                {
00029     if (argc < 4) {
00030         exibir_uso();
00031         return false;
00032     }
00033
00034     if (const std::string modo = argv[1];modo == "-c" && argc < 5) {
00035         exibir_uso();
00036         return false;
00037     }
00038
00039     return true;
00040 }
```

5.12 Referência do Arquivo C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/src/aresta_grafo.cpp

Implementação da classe **aresta_grafo** (p. **??**).

```
#include "../include/aresta_grafo.h"
```

5.12.1 Descrição detalhada

Implementação da classe **aresta_grafo** (p. **??**).

5.13 Referência do Arquivo C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/src/grafo.cpp

Implementação da classe grafo.

```
#include "../include/grafo.h"
#include <fstream>
#include <iostream>
#include <stdexcept>
```

Funções

- void **liberar_arestas_temp** (**aresta_grafo** ∗cabeca)
  Libera a memória alocada para as arestas temporárias.

5.13.1 Descrição detalhada

Implementação da classe grafo.

5.13.2 Funções

5.13.2.1 liberar_arestas_temp()

```
void liberar_arestas_temp (
              aresta_grafo * cabeca)
```

Libera a memória alocada para as arestas temporárias.

Parâmetros

cabeca	O ponteiro para a primeira aresta.

```
00071                                                {
00072     while (cabeca) {
00073         aresta_grafo* temp = cabeca;
00074         cabeca = cabeca->proxima;
00075         delete temp; // Libera apenas a cópia
00076     }
00077 }
```

5.14 Referência do Arquivo C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/src/grafo_lista.cpp

Implementação da classe **grafo_lista** (p. **??**).

```
#include "../include/grafo_lista.h"
```

5.14.1 Descrição detalhada

Implementação da classe **grafo_lista** (p. **??**).

5.15 Referência do Arquivo C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/src/grafo_matriz.cpp

Implementação da classe **grafo_matriz** (p. **??**).

```
#include "../include/grafo_matriz.h"
#include <iostream>
```

5.15.1 Descrição detalhada

Implementação da classe **grafo_matriz** (p. **??**).

5.16 Referência do Arquivo C:/ProjetoGrafos/Trabalho-de-Grafos/trabalho-de-grafos/src/no_grafo.cpp

Implementação da classe **no_grafo** (p. **??**).

```
#include "../include/no_grafo.h"
#include "../include/aresta_grafo.h"
```

5.16.1 Descrição detalhada

Implementação da classe **no_grafo** (p. **??**).