Árvore Geradora Mínima Generalizada

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.
- ~aresta_grafo ()

  Destrutor da classe aresta_grafo.

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.

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.

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:

- pedro/include/aresta_grafo.h
- pedro/src/aresta_grafo.cpp

4.2 Referência da Estrutura ArestaInfo

Atributos Públicos

- int origem
- int destino
- int peso

4.2.1 Atributos

4.2.1.1 destino

```
int ArestaInfo::destino
```

4.2.1.2 origem

```
int ArestaInfo::origem
```

4.2.1.3 peso

```
int ArestaInfo::peso
```

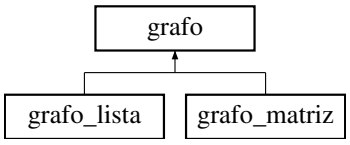
A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- pedro/src/grafo.cpp

4.3 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
- virtual int * agmg_gulosa (int *num_arestas)

  Algoritmo guloso para AGMG (Kruskal modificado).
- virtual int * agmg_randomizada (int *num_arestas, double alpha)

  Algoritmo randomizado para AGMG.
- virtual int * agmg_reativa (int *num_arestas)

  Algoritmo reativo para AGMG.
- int verificar_erro (int *cobertura, int tamanho)
- int get_grau ()

  Retorna o grau do grafo.
- bool eh_completo ()

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

  Retorna 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

4.3.1 Construtores e Destrutores

4.3.1.1 grafo()

```
grafo::grafo ()
00016 {}
```

4.3.1.2 ~grafo()

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

4.3.2 Documentação das funções

4.3.2.1 add_aresta()

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

Implementado por grafo_lista e grafo_matriz.

4.3.2.2 add_no()

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

Implementado por grafo_lista e grafo_matriz.

4.3.2.3 agmg_gulosa()

```
int * grafo::agmg_gulosa (
            int * num_arestas)  [virtual]
```

Algoritmo guloso para AGMG (Kruskal modificado).

Parâmetros

num_arestas	Ponteiro para armazenar o número de arestas na AGMG.

Retorna

Array de inteiros representando as arestas selecionadas (origem, destino, peso).

```
00125                                       {
00126     int n = get_ordem();
00127     int* pesos_vertices = new int[n + 1];
00128     for (int i = 1; i <= n; ++i) {
00129         no_grafo* no = get_no(i);
00130         if (!no) {
00131             delete[] pesos_vertices;
00132             *num_arestas = 0;
00133             return nullptr;
00134         }
00135         pesos_vertices[i] = no->peso;
00136     }
00137
00138     ArestaInfo* arestas = nullptr;
00139     int contador = 0;
00140     int capacidade = 0;
00141
00142     for (int u = 1; u <= n; ++u) {
00143         aresta_grafo* vizinhos = get_vizinhos(u);
00144         aresta_grafo* atual = vizinhos;
00145         while (atual) {
00146             if (!direcionado && u > atual->destino) {
00147                 atual = atual->proxima;
00148                 continue;
00149             }
00150
00151             if (contador >= capacidade) {
00152                 int nova_capacidade = (capacidade == 0) ? 1 : capacidade * 2;
00153                 ArestaInfo* nova_arestas = new ArestaInfo[nova_capacidade];
00154                 for (int i = 0; i < contador; ++i) {
00155                     nova_arestas[i] = arestas[i];
00156                 }
00157                 delete[] arestas;
00158                 arestas = nova_arestas;
00159                 capacidade = nova_capacidade;
00160             }
00161
00162             arestas[contador++] = {u, atual->destino, atual->peso};
00163             atual = atual->proxima;
00164         }
00165         liberar_arestas_temp(vizinhos);
00166     }
00167
00168     std::sort(arestas, arestas + contador, compararArestas);
00169
00170     int* cluster_id = new int[n + 1];
00171     int num_clusters = 0;
00172     int* cluster_map = new int[n + 1]{0};
00173
00174     for (int i = 1; i <= n; ++i) {
00175         bool encontrado = false;
00176         for (int j = 1; j < i; ++j) {
00177             if (pesos_vertices[i] == pesos_vertices[j]) {
00178                 cluster_id[i] = cluster_id[j];
00179                 encontrado = true;
00180                 break;
00181             }
00182         }
00183         if (!encontrado) {
00184             cluster_id[i] = num_clusters++;
00185         }
00186     }
00187
```

```
00188     int* pai = new int[num_clusters];
00189     int* rank = new int[num_clusters];
00190     for (int i = 0; i < num_clusters; ++i) {
00191         pai[i] = i;
00192         rank[i] = 0;
00193     }
00194
00195     auto find = [&](int u) {
00196         while (pai[u] != u) {
00197             pai[u] = pai[pai[u]];
00198             u = pai[u];
00199         }
00200         return u;
00201     };
00202
00203     auto unite = [&](int u, int v) {
00204         u = find(u);
00205         v = find(v);
00206         if (u != v) {
00207             if (rank[u] < rank[v]) std::swap(u, v);
00208             pai[v] = u;
00209             if (rank[u] == rank[v]) rank[u]++;
00210         }
00211     };
00212
00213     int* resultado = new int[3 * (num_clusters - 1)];
00214     *num_arestas = 0;
00215     int idx = 0;
00216
00217     for (int i = 0; i < contador; ++i) {
00218         int u = arestas[i].origem;
00219         int v = arestas[i].destino;
00220         int c_u = cluster_id[u];
00221         int c_v = cluster_id[v];
00222
00223         if (find(c_u) != find(c_v)) {
00224             resultado[idx++] = u;
00225             resultado[idx++] = v;
00226             resultado[idx++] = arestas[i].peso;
00227             (*num_arestas)++;
00228             unite(c_u, c_v);
00229             if (*num_arestas == num_clusters - 1) break;
00230         }
00231     }
00232
00233     delete[] arestas;
00234     delete[] pesos_vertices;
00235     delete[] cluster_id;
00236     delete[] cluster_map;
00237     delete[] pai;
00238     delete[] rank;
00239
00240     return resultado;
00241 }
```

4.3.2.4 agmg_randomizada()

```
int * grafo::agmg_randomizada (
            int * num_arestas,
            double alpha = 0.5)  [virtual]
```

Algoritmo randomizado para AGMG.

Parâmetros

num_arestas	Ponteiro para armazenar o número de arestas na AGMG.
alpha	Parâmetro de aleatoriedade (0-1).

Retorna

Array de inteiros representando as arestas selecionadas.

```
00249                                                                   {
00250     int n = get_ordem();
00251     int* pesos_vertices = new int[n + 1];
00252     for (int i = 1; i <= n; ++i) {
00253         no_grafo* no = get_no(i);
00254         pesos_vertices[i] = no->peso;
00255     }
00256
00257     ArestaInfo* arestas = nullptr;
00258     int contador = 0;
00259     int capacidade = 0;
00260
00261     for (int u = 1; u <= n; ++u) {
00262         aresta_grafo* vizinhos = get_vizinhos(u);
00263         aresta_grafo* atual = vizinhos;
00264         while (atual) {
00265             int v = atual->destino;
00266             if (!direcionado && u > v) {
00267                 atual = atual->proxima;
00268                 continue;
00269             }
00270             if (contador >= capacidade) {
00271                 capacidade = (capacidade == 0) ? 1 : capacidade * 2;
00272                 ArestaInfo* nova = new ArestaInfo[capacidade];
00273                 for (int i = 0; i < contador; ++i) nova[i] = arestas[i];
00274                 delete[] arestas;
00275                 arestas = nova;
00276             }
00277             arestas[contador++] = {u, v, atual->peso};
00278             atual = atual->proxima;
00279         }
00280         liberar_arestas_temp(vizinhos);
00281     }
00282
00283     int k = std::max(1, (int)(contador * alpha));
00284     for (int i = 0; i < k; ++i) {
00285         int r = i + rand() % (contador - i);
00286         std::swap(arestas[i], arestas[r]);
00287     }
00288
00289     int* cluster_id = new int[n + 1];
00290     int num_clusters = 0;
00291     int* cluster_map = new int[n + 1]{0};
00292
00293     for (int i = 1; i <= n; ++i) {
00294         bool encontrado = false;
00295         for (int j = 1; j < i; ++j) {
00296             if (pesos_vertices[i] == pesos_vertices[j]) {
00297                 cluster_id[i] = cluster_id[j];
00298                 encontrado = true;
00299                 break;
00300             }
00301         }
00302         if (!encontrado) {
00303             cluster_id[i] = num_clusters++;
00304         }
00305     }
00306
00307     int* pai = new int[num_clusters];
00308     int* rank = new int[num_clusters];
00309     for (int i = 0; i < num_clusters; ++i) {
00310         pai[i] = i;
00311         rank[i] = 0;
00312     }
00313
00314     auto find = [&](int u) {
00315         while (pai[u] != u) {
00316             pai[u] = pai[pai[u]];
00317             u = pai[u];
00318         }
00319         return u;
00320     };
00321
00322     auto unite = [&](int u, int v) {
00323         u = find(u);
00324         v = find(v);
00325         if (u != v) {
00326             if (rank[u] < rank[v]) std::swap(u, v);
00327             pai[v] = u;
00328             if (rank[u] == rank[v]) rank[u]++;
00329         }
00330     };
00331
```

```
00332     int* resultado = new int[3 * (num_clusters - 1)];
00333     *num_arestas = 0;
00334     int idx = 0;
00335
00336     for (int i = 0; i < contador; ++i) {
00337         int u = arestas[i].origem;
00338         int v = arestas[i].destino;
00339         int c_u = cluster_id[u];
00340         int c_v = cluster_id[v];
00341
00342         if (find(c_u) != find(c_v)) {
00343             resultado[idx++] = u;
00344             resultado[idx++] = v;
00345             resultado[idx++] = arestas[i].peso;
00346             (*num_arestas)++;
00347             unite(c_u, c_v);
00348             if (*num_arestas == num_clusters - 1) break;
00349         }
00350     }
00351
00352     delete[] arestas;
00353     delete[] pesos_vertices;
00354     delete[] cluster_id;
00355     delete[] cluster_map;
00356     delete[] pai;
00357     delete[] rank;
00358
00359     return resultado;
00360 }
```

4.3.2.5 agmg_reativa()

```
int * grafo::agmg_reativa (
            int * num_arestas)  [virtual]
```

Algoritmo reativo para AGMG.

Parâmetros

num_arestas	Ponteiro para armazenar o número de arestas.

Retorna

Array de inteiros com as arestas selecionadas.

```
00367                                       {
00368     const int MAX_ITER = 50;
00369     const double INITIAL_PROB = 0.5;
00370     double prob_guloso = INITIAL_PROB;
00371     int* melhor = nullptr;
00372     int menor_custo = INT_MAX;
00373     int falhas_guloso = 0, falhas_random = 0;
00374
00375     for (int iter = 0; iter < MAX_ITER; ++iter) {
00376         int* solucao;
00377         int temp_size;
00378         bool usar_guloso = ((double)rand() / RAND_MAX < prob_guloso);
00379
00380         if (usar_guloso) {
00381             solucao = agmg_gulosa(&temp_size);
00382         } else {
00383             solucao = agmg_randomizada(&temp_size);
00384         }
00385
00386         int custo = 0;
00387         for (int i = 2; i < temp_size * 3; i += 3) {
00388             custo += solucao[i];
00389         }
00390
00391         if (custo < menor_custo) {
00392             delete[] melhor;
00393             menor_custo = custo;
00394             melhor = new int[temp_size * 3];
00395             for (int i = 0; i < temp_size * 3; ++i) {
00396                 melhor[i] = solucao[i];
```

```
00397             }
00398             *num_arestas = temp_size;
00399         } else {
00400             if (usar_guloso) falhas_guloso++;
00401             else falhas_random++;
00402         }
00403
00404         if (falhas_guloso + falhas_random > 0) {
00405             prob_guloso = 1.0 - ((double)falhas_guloso / (falhas_guloso + falhas_random));
00406         }
00407
00408         delete[] solucao;
00409     }
00410
00411     return melhor;
00412 }
```

4.3.2.6 aresta_ponderada()

```
bool grafo::aresta_ponderada () const
00432 { return ponderado_arestas; }
```

4.3.2.7 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.

```
00032                                                 {
00033     std::ifstream file(arquivo);
00034     if (!file.is_open()) throw std::runtime_error("Arquivo não encontrado");
00035
00036     int num_nos, dir, pond_vertices, pond_arestas;
00037     file » num_nos » dir » pond_vertices » pond_arestas;
00038
00039     this->direcionado = dir;
00040     this->ponderado_vertices = pond_vertices;
00041     this->ponderado_arestas = pond_arestas;
00042     this->num_nos = num_nos;
00043
00044     if (ponderado_vertices) {
00045         for (int i = 1; i <= num_nos; ++i) {
00046             int peso;
00047             file » peso;
00048             add_no(i, peso);
00049         }
00050     } else {
00051         for (int i = 1; i <= num_nos; ++i) {
00052             add_no(i, 0);
00053         }
00054     }
00055
00056     int origem, destino, peso = 0;
00057     while (file » origem » destino) {
00058         if (ponderado_arestas) file » peso;
00059         add_aresta(origem, destino, peso);
00060     }
00061 }
```

4.3.2.8 eh_completo()

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

Verifica se o grafo é completo.

Retorna

true se o grafo é completo, false caso contrário.

```
00067                           {
00068     int n = get_ordem();
00069     for (int i = 1; i <= n; ++i) {
00070         for (int j = 1; j <= n; ++j) {
00071             if (i != j && !existe_aresta(i, j)) {
00072                 if (direcionado) return false;
00073                 if (!existe_aresta(j, i)) return false;
00074             }
00075         }
00076     }
00077     return true;
00078 }
```

4.3.2.9 eh_direcionado()

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

Retorna as flags: direcionado, ponderado_vertices e ponderado_arestas.

```
00430 { return direcionado; }
```

4.3.2.10 exibe_descricao()

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

Exibe a descrição do grafo.

```
00418                               {
00419     std::cout « "Grau: " « get_grau() « std::endl;
00420     std::cout « "Ordem: " « get_ordem() « std::endl;
00421     std::cout « "Direcionado: " « (eh_direcionado() ? "Sim" : "Nao") « std::endl;
00422     std::cout « "Vertices ponderados: " « (vertice_ponderado() ? "Sim" : "Nao") « std::endl;
00423     std::cout « "Arestas ponderadas: " « (aresta_ponderada() ? "Sim" : "Nao") « std::endl;
00424     std::cout « "Completo: " « (eh_completo() ? "Sim" : "Nao") « std::endl;
00425 }
```

4.3.2.11 existe_aresta()

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

Implementado por grafo_lista e grafo_matriz.

4.3.2.12 get_aresta()

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

Implementado por grafo_lista e grafo_matriz.

4.3.2.13 get_grau()

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

Retorna o grau do grafo.

Retorna

O grau do grafo.

```
00096                     {
00097     int grau_maximo = 0;
00098     int n = get_ordem();
00099     for (int i = 1; i <= n; ++i) {
00100         int grau_atual = 0;
00101         aresta_grafo* vizinhos = get_vizinhos(i);
00102         aresta_grafo* atual = vizinhos;
00103         while (atual) {
00104             grau_atual++;
00105             atual = atual->proxima;
00106         }
00107         liberar_arestas_temp(vizinhos);
00108 
00109         if (direcionado) {
00110             for (int j = 1; j <= n; ++j) {
00111                 if (existe_aresta(j, i)) grau_atual++;
00112             }
00113         }
00114 
00115         if (grau_atual > grau_maximo) grau_maximo = grau_atual;
00116     }
00117     return grau_maximo;
00118 }
```

4.3.2.14 get_no()

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

Implementado por grafo_lista e grafo_matriz.

4.3.2.15 get_ordem()

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

Implementado por grafo_lista e grafo_matriz.

4.3.2.16 get_vizinhos()

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

Implementado por grafo_lista e grafo_matriz.

4.3.2.17 verificar_erro()

```
int grafo::verificar_erro (
            int * cobertura,
            int tamanho)
```

4.3.2.18 vertice_ponderado()

```
bool grafo::vertice_ponderado () const
00431 { return ponderado_vertices; }
```

4.3.3 Atributos

4.3.3.1 direcionado

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

4.3.3.2 num_nos

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

4.3.3.3 ponderado_arestas

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

4.3.3.4 ponderado_vertices

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

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

- pedro/include/grafo.h
- pedro/src/grafo.cpp

4.4 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.
- ~grafo_lista () override

  Destrutor da classe grafo_lista.
- 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
- virtual int * agmg_gulosa (int *num_arestas)

  Algoritmo guloso para AGMG (Kruskal modificado).
- virtual int * agmg_randomizada (int *num_arestas, double alpha)

  Algoritmo randomizado para AGMG.
- virtual int * agmg_reativa (int *num_arestas)

  Algoritmo reativo para AGMG.
- int verificar_erro (int *cobertura, int tamanho)
- int get_grau ()

  Retorna o grau do grafo.
- bool eh_completo ()

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

  Retorna 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_lista()

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

Construtor da classe grafo_lista.

O ponteiro para o primeiro nó é inicializado como nullptr.

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

4.4.1.2 ~grafo_lista()

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

Destrutor da classe grafo_lista.

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 }
```

4.4.2 Documentação das funções

4.4.2.1 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.

```
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 }
```

4.4.2.2 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.

```
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 }
```

4.4.2.3 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.

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

4.4.2.4 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.

```
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 }
```

4.4.2.5 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.

```
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 }
```

4.4.2.6 get_ordem()

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

Retorna a ordem do grafo.

Retorna

O número de nós do grafo.

Implementa grafo.

```
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 }
```

4.4.2.7 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.

```
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 }
```

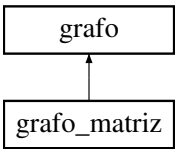
A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- pedro/include/grafo_lista.h
- pedro/src/grafo_lista.cpp

4.5 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.
- ~grafo_matriz () override
- 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
- virtual int * agmg_gulosa (int *num_arestas)

  Algoritmo guloso para AGMG (Kruskal modificado).
- virtual int * agmg_randomizada (int *num_arestas, double alpha)

  Algoritmo randomizado para AGMG.
- virtual int * agmg_reativa (int *num_arestas)

  Algoritmo reativo para AGMG.
- int verificar_erro (int *cobertura, int tamanho)
- int get_grau ()

  Retorna o grau do grafo.
- bool eh_completo ()

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

  Retorna 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

4.5.1 Construtores e Destrutores

4.5.1.1 grafo_matriz()

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

Construtor da classe grafo_matriz.

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

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

4.5.1.2 ~grafo_matriz()

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

```
00017                             {
00018     if (matriz_inicializada) {
00019         for (int i = 0; i < num_nos; ++i) {
00020             for (int j = 0; j < num_nos; ++j) {
00021                 delete matriz[i][j];
00022             }
00023             delete[] matriz[i];
00024         }
00025         delete[] matriz;
00026     }
00027
00028     if (nos) {
00029         for (int i = 0; i < num_nos; ++i) {
00030             delete nos[i];
00031         }
00032         delete[] nos;
00033     }
00034 }
```

4.5.2 Documentação das funções

4.5.2.1 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.

```
00144                                                                                {
00145     if (origem == destino) return;
00146     if (!get_no(origem) || !get_no(destino)) return;
00147
00148     int i = origem - 1;
00149     int j = destino - 1;
00150
00151     if (i >= 0 && i < num_nos && j >= 0 && j < num_nos && !matriz[i][j]) {
00152         matriz[i][j] = new aresta_grafo(destino, peso);
00153
00154         if (!direcionado && origem != destino) {
00155             matriz[j][i] = new aresta_grafo(origem, peso);
00156         }
00157     }
00158 }
```

4.5.2.2 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.

```
00112                                           {
00113     if (id < 1 || id > num_nos) return;
00114
00115     if (!nos) {
00116         nos = new no_grafo*[num_nos];
00117         for (int i = 0; i < num_nos; ++i) {
00118             nos[i] = nullptr;
00119         }
00120     }
00121
00122     if (!nos[id - 1]) {
00123         nos[id - 1] = new no_grafo(id, peso);
00124     }
00125
00126     if (!matriz_inicializada && num_nos > 0) {
00127         matriz = new aresta_grafo**[num_nos];
00128         for (int i = 0; i < num_nos; ++i) {
00129             matriz[i] = new aresta_grafo*[num_nos];
00130             for (int j = 0; j < num_nos; ++j) {
00131                 matriz[i][j] = nullptr;
00132             }
00133         }
00134         matriz_inicializada = true;
00135     }
00136 }
```

4.5.2.3 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.

```
00103                                                       {
00104     return get_aresta(origem, destino) != nullptr;
00105 }
```

4.5.2.4 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.

```
00052                                                                   {
00053     if (origem < 1 || origem > num_nos || destino < 1 || destino > num_nos)
00054         return nullptr;
00055
00056     if (!direcionado && origem > destino)
00057         std::swap(origem, destino);
00058
00059     return matriz[origem-1][destino-1];
00060 }
```

4.5.2.5 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.

```
00041                                       {
00042     if (id < 1 || id > num_nos || !nos) return nullptr;
00043     return nos[id - 1];
00044 }
```

4.5.2.6 get_ordem()

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

Retorna a ordem do grafo.

Retorna

O número de nós do grafo.

Implementa grafo.

```
00093                               {
00094     return num_nos;
00095 }
```

4.5.2.7 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.

```
00067                                                {
00068     if (id < 1 || id > num_nos) return nullptr;
00069
00070     aresta_grafo* cabeca = nullptr;
00071     aresta_grafo* atual = nullptr;
00072
00073     for (int j = 0; j < num_nos; ++j) {
00074         if (matriz[id-1][j] != nullptr) {
00075             aresta_grafo* nova_aresta = new aresta_grafo(matriz[id-1][j]->destino,
      matriz[id-1][j]->peso);
00076
00077             if (!cabeca) {
00078                 cabeca = nova_aresta;
00079                 atual = cabeca;
00080             } else {
00081                 atual->proxima = nova_aresta;
00082                 atual = atual->proxima;
00083             }
00084         }
00085     }
00086     return cabeca;
00087 }
```

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

- pedro/include/grafo_matriz.h
- pedro/src/grafo_matriz.cpp

4.6 Referência da Classe no_grafo

`#include <no_grafo.h>`

Membros Públicos

- no_grafo (int id, int peso=0)

  Construtor da classe no_grafo.
- ~no_grafo ()

  Destrutor da classe no_grafo.

Atributos Públicos

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

4.6.1 Construtores e Destrutores

4.6.1.1 no_grafo()

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

Construtor da classe no_grafo.

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 {}
```

4.6.1.2 ~no_grafo()

```
no_grafo::~no_grafo ()
```

Destrutor da classe no_grafo.

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 }
```

4.6.2 Atributos

4.6.2.1 id

```
int no_grafo::id
```

4.6.2.2 peso

```
int no_grafo::peso
```

4.6.2.3 primeira_aresta

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

4.6.2.4 proximo

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

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

- pedro/include/no_grafo.h
- pedro/src/no_grafo.cpp

Capítulo 5

Arquivos

5.1 Referência do Arquivo pedro/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 pedro/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 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     virtual int* agmg_gulosa(int* num_arestas);
00029     virtual int* agmg_randomizada(int* num_arestas, double alpha);
00030     virtual int* agmg_reativa(int* num_arestas);
00031     int verificar_erro(int* cobertura, int tamanho);
00032
00033     int get_grau();
00034     bool eh_completo();
00035     bool eh_direcionado() const;
00036     bool vertice_ponderado() const;
00037     bool aresta_ponderada() const;
00038     void carrega_grafo(const std::string& arquivo);
00039
00040     void exibe_descricao();
00041
00042     virtual void add_no(int id, int peso) = 0;
00043     virtual void add_aresta(int origem, int destino, int peso) = 0;
00044 };
00045
00046 #endif //GRAFO_H
```

5.5 Referência do Arquivo pedro/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 pedro/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     no_grafo** nos;
00016
00017 public:
00018     grafo_matriz();
00019     ~grafo_matriz() override;
00020
00021     no_grafo* get_no(int id) override;
00022     aresta_grafo* get_aresta(int origem, int destino) override;
00023     aresta_grafo* get_vizinhos(int id) override;
00024     int get_ordem() override;
00025     bool existe_aresta(int origem, int destino) override;
00026
00027     void add_no(int id, int peso) override;
00028     void add_aresta(int origem, int destino, int peso) override;
00029 };
00030
00031 #endif // GRAFO_MATRIZ_H
```

5.9 Referência do Arquivo pedro/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 pedro/main.cpp

Programa principal.

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

Funções

- void exibir_uso ()

  Exibe mensagem de uso do programa.
- bool validar_argumentos (int argc, char ∗argv[ ])

  Valida os argumentos passados para o programa.
- void executar_agmg (grafo ∗g)

  Executa o algoritmo AGMG.
- int main (int argc, char ∗argv[ ])

  Função principal.

5.11.1 Descrição detalhada

Programa principal.

5.11.2 Funções

5.11.2.1 executar_agmg()

```
void executar_agmg (
            grafo * g)
```

Executa o algoritmo AGMG.

Parâmetros

g	Grafo.

```
00061                               {
00062     int escolha;
00063     cout « "\nSelecione o algoritmo:\n";
00064     cout « "1 - Guloso\n";
00065     cout « "2 - Randomizado\n";
00066     cout « "3 - Reativo\n";
00067     cout « "Digite sua escolha (1-3): ";
00068     cin » escolha;
00069 
00070     int num_arestas;
00071     int* agmg = nullptr;
00072     clock_t inicio = clock();
00073 
00074     switch(escolha) {
00075         case 1:
00076             agmg = g->agmg_gulosa(&num_arestas);
00077             break;
00078         case 2:
00079             agmg = g->agmg_randomizada(&num_arestas, 0.5);
00080             break;
00081         case 3:
00082             agmg = g->agmg_reativa(&num_arestas);
00083             break;
00084         default:
00085             cout « "Opção inválida!\n";
00086             return;
00087     }
00088 
00089     double tempo = double(clock() - inicio) / CLOCKS_PER_SEC;
00090 
00091     cout « "\nAGMG (" « num_arestas « " arestas):\n";
00092     int custo_total = 0;
00093     for(int i = 0; i < num_arestas * 3; i += 3) {
00094         cout « agmg[i] « "-" « agmg[i+1] « " (" « agmg[i+2] « ")\n";
00095         custo_total += agmg[i+2];
00096     }
00097     cout « "Custo total: " « custo_total « "\n";
00098     cout « "Tempo de execução: " « tempo « "s\n\n";
00099 
00100     delete[] agmg;
00101 }
```

5.11.2.2 exibir_uso()

```
void exibir_uso ()
```

Exibe mensagem de uso do programa.

```
00020                     {
00021     cout « "Uso:\n";
00022     cout « "Descrição do grafo:\n";
00023     cout « "  main.out -d -m grafo.txt\n";
00024     cout « "  main.out -d -l grafo.txt\n";
00025     cout « "Resolver problema de cobertura:\n";
00026     cout « "  main.out -p -m grafo.txt\n";
00027     cout « "  main.out -p -l grafo.txt\n";
00028 }
```

5.11.2.3 main()

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

Função principal.

Parâmetros

argc	Número de argumentos.
argv	Array de argumentos.

Retorna

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

```
00109                                   {
00110     if (!validar_argumentos(argc, argv)) {
00111         return 1;
00112     }
00113
00114     const string modo = argv[1];
00115     const string estrutura = argv[2];
00116     const string arquivo = argv[3];
00117
00118     grafo* g = nullptr;
00119
00120     try {
00121         if(estrutura == "-m") {
00122             g = new grafo_matriz();
00123         } else {
00124             g = new grafo_lista();
00125         }
00126
00127         g->carrega_grafo(arquivo);
00128
00129         if(modo == "-d") {
00130             g->exibe_descricao();
00131         } else {
00132             executar_agmg(g);
00133         }
00134
00135         delete g;
00136     }
00137     catch(const exception& e) {
00138         cerr « "Erro: " « e.what() « endl;
00139         if(g) delete g;
00140         return 1;
00141     }
00142
00143     return 0;
00144 }
```

5.11.2.4 validar_argumentos()

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

Valida os argumentos passados para o programa.

Parâmetros

argc	Número de argumentos.
argv	Array de argumentos.

Retorna

true se os argumentos são válidos, false caso contrário.

```
00036                                                {
00037     if (argc != 4) {
00038         exibir_uso();
00039         return false;
00040     }
00041
00042     const string modo = argv[1];
00043     if (modo != "-d" && modo != "-p") {
00044         exibir_uso();
00045         return false;
00046     }
00047
00048     const string estrutura = argv[2];
00049     if (estrutura != "-m" && estrutura != "-l") {
00050         exibir_uso();
00051         return false;
00052     }
00053
00054     return true;
00055 }
```

5.12 Referência do Arquivo pedro/src/aresta_grafo.cpp

Implementação da classe aresta_grafo.

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

5.12.1 Descrição detalhada

Implementação da classe aresta_grafo.

5.13 Referência do Arquivo pedro/src/grafo.cpp

Implementação da classe grafo.

```
#include "../include/grafo.h"
#include <fstream>
#include <iostream>
#include <stdexcept>
#include <climits>
#include <cstdlib>
#include <ctime>
#include <algorithm>
```

Componentes

- struct ArestaInfo

Funções

- bool compararArestas (const ArestaInfo &a, const ArestaInfo &b)
- 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 compararArestas()

```
bool compararArestas (
            const ArestaInfo & a,
            const ArestaInfo & b)
00024                                                                {
00025     return a.peso < b.peso;
00026 }
```

5.13.2.2 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.

```
00084                                                {
00085     while (cabeca) {
00086         aresta_grafo* temp = cabeca;
00087         cabeca = cabeca->proxima;
00088         delete temp;
00089     }
00090 }
```

5.14 Referência do Arquivo pedro/src/grafo_lista.cpp

Implementação da classe grafo_lista.

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

5.14.1 Descrição detalhada

Implementação da classe grafo_lista.

5.15 Referência do Arquivo pedro/src/grafo_matriz.cpp

Implementação da classe grafo_matriz.

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

5.15.1 Descrição detalhada

Implementação da classe grafo_matriz.

5.16 Referência do Arquivo pedro/src/no_grafo.cpp

Implementação da classe no_grafo.

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

5.16.1 Descrição detalhada

Implementação da classe no_grafo.

Índice Remissivo