Cobertura de Vértices

Gerado por Doxygen 1.13.1

1 Índice Hierárquico 1

1.1 Hierarquia de Classes . . . 1

2 Índice dos Componentes 3

2.1 Lista de Classes . . . 3

3 Índice dos Arquivos 5

3.1 Lista de Arquivos . . . 5

4 Classes 7

4.1 Referência da Estrutura Aresta . . . 7

4.1.1 Atributos . . . 7

4.1.1.1 destino . . . 7

4.1.1.2 next . . . 7

4.1.1.3 origem . . . 7

4.2 Referência da Classe aresta_grafo . . . 7

4.2.1 Construtores e Destrutores . . . 8

4.2.1.1 aresta_grafo() . . . 8

4.2.1.2 ∼aresta_grafo() . . . 8

4.2.2 Atributos . . . 8

4.2.2.1 destino . . . 8

4.2.2.2 peso . . . 9

4.2.2.3 proxima . . . 9

4.3 Referência da Classe grafo . . . 9

4.3.1 Construtores e Destrutores . . . 10

4.3.1.1 grafo() . . . 10

4.3.1.2 ∼grafo() . . . 10

4.3.2 Documentação das funções . . . 10

4.3.2.1 add_aresta() . . . 10

4.3.2.2 add_no() . . . 10

4.3.2.3 aresta_ponderada() . . . 10

4.3.2.4 carrega_grafo() . . . 10

4.3.2.5 cobertura_gulosa() . . . 11

4.3.2.6 cobertura_randomizada() . . . 12

4.3.2.7 cobertura_reativa() . . . 14

4.3.2.8 eh_completo() . . . 14

4.3.2.9 eh_direcionado() . . . 15

4.3.2.10 exibe_descricao() . . . 15

4.3.2.11 existe_aresta() . . . 15

4.3.2.12 get_aresta() . . . 15

4.3.2.13 get_grau() . . . 16

4.3.2.14 get_no() . . . 16

4.3.2.15 get_ordem() . . . 16

4.3.2.16 get_vizinhos() . . . 16
4.3.2.17 verificar_erro() . . . 16
4.3.2.18 vertice_ponderado() . . . 17
4.3.3 Atributos . . . 17
4.3.3.1 direcionado . . . 17
4.3.3.2 num_nos . . . 17
4.3.3.3 ponderado_arestas . . . 17
4.3.3.4 ponderado_vertices . . . 18
4.4 Referência da Classe grafo_lista . . . 18
4.4.1 Construtores e Destrutores . . . 19
4.4.1.1 grafo_lista() . . . 19
4.4.1.2 ∼grafo_lista() . . . 20
4.4.2 Documentação das funções . . . 20
4.4.2.1 add_aresta() . . . 20
4.4.2.2 add_no() . . . 20
4.4.2.3 existe_aresta() . . . 21
4.4.2.4 get_aresta() . . . 21
4.4.2.5 get_no() . . . 22
4.4.2.6 get_ordem() . . . 22
4.4.2.7 get_vizinhos() . . . 22
4.5 Referência da Classe grafo_matriz . . . 23
4.5.1 Construtores e Destrutores . . . 24
4.5.1.1 grafo_matriz() . . . 24
4.5.1.2 ∼grafo_matriz() . . . 25
4.5.2 Documentação das funções . . . 25
4.5.2.1 add_aresta() . . . 25
4.5.2.2 add_no() . . . 25
4.5.2.3 existe_aresta() . . . 26
4.5.2.4 get_aresta() . . . 26
4.5.2.5 get_no() . . . 27
4.5.2.6 get_ordem() . . . 27
4.5.2.7 get_vizinhos() . . . 27
4.6 Referência da Classe no_grafo . . . 28
4.6.1 Construtores e Destrutores . . . 28
4.6.1.1 no_grafo() . . . 28
4.6.1.2 ∼no_grafo() . . . 29
4.6.2 Atributos . . . 29
4.6.2.1 id . . . 29
4.6.2.2 peso . . . 29
4.6.2.3 primeira_aresta . . . 29
4.6.2.4 proximo . . . 29

5 Arquivos **31**

5.1 Referência do Arquivo trabalho-final-grupo/include/aresta_grafo.h . . . 31

5.1.1 Descrição detalhada . . . 31

5.2 aresta_grafo.h . . . 31

5.3 Referência do Arquivo trabalho-final-grupo/include/grafo.h . . . 32

5.3.1 Descrição detalhada . . . 32

5.4 grafo.h . . . 32

5.5 Referência do Arquivo trabalho-final-grupo/include/grafo_lista.h . . . 33

5.5.1 Descrição detalhada . . . 33

5.6 grafo_lista.h . . . 33

5.7 Referência do Arquivo trabalho-final-grupo/include/grafo_matriz.h . . . 33

5.7.1 Descrição detalhada . . . 34

5.8 grafo_matriz.h . . . 34

5.9 Referência do Arquivo trabalho-final-grupo/include/no_grafo.h . . . 34

5.9.1 Descrição detalhada . . . 34

5.10 no_grafo.h . . . 35

5.11 Referência do Arquivo trabalho-final-grupo/main.cpp . . . 35

5.11.1 Descrição detalhada . . . 35

5.11.2 Funções . . . 35

5.11.2.1 executar_cobertura() . . . 35

5.11.2.2 exibir_uso() . . . 36

5.11.2.3 main() . . . 36

5.11.2.4 validar_argumentos() . . . 37

5.12 Referência do Arquivo trabalho-final-grupo/src/aresta_grafo.cpp . . . 38

5.12.1 Descrição detalhada . . . 38

5.13 Referência do Arquivo trabalho-final-grupo/src/grafo.cpp . . . 38

5.13.1 Descrição detalhada . . . 38

5.13.2 Funções . . . 38

5.13.2.1 liberar_arestas_temp() . . . 38

5.14 Referência do Arquivo trabalho-final-grupo/src/grafo_lista.cpp . . . 39

5.14.1 Descrição detalhada . . . 39

5.15 Referência do Arquivo trabalho-final-grupo/src/grafo_matriz.cpp . . . 39

5.15.1 Descrição detalhada . . . 39

5.16 Referência do Arquivo trabalho-final-grupo/src/no_grafo.cpp . . . 39

5.16.1 Descrição detalhada . . . 39

Índice Remissivo **41**

Capítulo 1

Índice Hierárquico

1.1 Hierarquia de Classes

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

Aresta 7
aresta_grafo 7
grafo 9
    grafo_lista 18
    grafo_matriz 23
no_grafo 28

Capítulo 2

Índice dos Componentes

2.1 Lista de Classes

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

Aresta 7
aresta_grafo 7
grafo 9
grafo_lista 18
grafo_matriz 23
no_grafo 28

Capítulo 3

Índice dos Arquivos

3.1 Lista de Arquivos

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

trabalho-final-grupo/main.cpp
Programa principal 35
trabalho-final-grupo/include/aresta_grafo.h
Classe que representa uma aresta de um grafo 31
trabalho-final-grupo/include/grafo.h
Classe abstrata que define as operações que podem ser realizadas em um grafo 32
trabalho-final-grupo/include/grafo_lista.h
Classe que representa um grafo implementado com listas de adjacência 33
trabalho-final-grupo/include/grafo_matriz.h
Classe que representa um grafo implementado com matriz de adjacência 33
trabalho-final-grupo/include/no_grafo.h
Classe que representa um nó de um grafo 34
trabalho-final-grupo/src/aresta_grafo.cpp
Implementação da classe aresta_grafo 38
trabalho-final-grupo/src/grafo.cpp
Implementação da classe grafo 38
trabalho-final-grupo/src/grafo_lista.cpp
Implementação da classe grafo_lista 39
trabalho-final-grupo/src/grafo_matriz.cpp
Implementação da classe grafo_matriz 39
trabalho-final-grupo/src/no_grafo.cpp
Implementação da classe no_grafo 39

Capítulo 4

Classes

4.1 Referência da Estrutura Aresta

Atributos Públicos

- int origem
- int destino
- Aresta * next

4.1.1 Atributos

4.1.1.1 destino

```
int Aresta::destino
```

4.1.1.2 next

```
Aresta* Aresta::next
```

4.1.1.3 origem

```
int Aresta::origem
```

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

- trabalho-final-grupo/src/grafo.cpp

4.2 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

4.2.1 Construtores e Destrutores

4.2.1.1 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 {}
```

4.2.1.2 ~aresta_grafo()

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

Destrutor da classe aresta_grafo.

4.2.2 Atributos

4.2.2.1 destino

```
int aresta_grafo::destino
```

4.2.2.2 peso

```
int aresta_grafo::peso
```

4.2.2.3 proxima

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

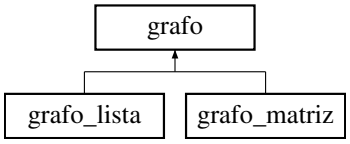
A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- trabalho-final-grupo/include/aresta_grafo.h
- trabalho-final-grupo/src/aresta_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
- int * cobertura_gulosa (int *tamanho)

  Calcula uma cobertura de vértices usando a estratégia gulosa.
- int * cobertura_randomizada (int *tamanho)

  Calcula uma cobertura de vértices usando uma estratégia randomizada.
- int * cobertura_reativa (int *tamanho)

  Calcula uma cobertura reativa de vértices combinando estratégias gulosa e randomizada.
- int verificar_erro (int *cobertura, int tamanho)

  Verifica se a cobertura de vértices encontrada cobre todas as arestas.
- 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 ()
00015 {}
```

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 aresta_ponderada()

```
bool grafo::aresta_ponderada () const
00391 { return ponderado_arestas; }
```

4.3.2.4 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.

```
00027                                                 {
00028     std::ifstream file(arquivo);
00029     if (!file.is_open()) throw std::runtime_error("Arquivo não encontrado");
00030
00031     int num_nos, dir, pond_vertices, pond_arestas;
00032     file » num_nos » dir » pond_vertices » pond_arestas;
00033
00034     this->direcionado = dir;
00035     this->ponderado_vertices = pond_vertices;
00036     this->ponderado_arestas = pond_arestas;
00037     this->num_nos = num_nos;
00038
00039     if (ponderado_vertices) {
00040         for (int i = 1; i <= num_nos; ++i) {
00041             int peso;
00042             file » peso;
00043             add_no(i, peso);
00044         }
00045     } else {
00046         for (int i = 1; i <= num_nos; ++i) {
00047             add_no(i, 0);
00048         }
00049     }
00050
00051     int origem, destino, peso = 0;
00052     while (file » origem » destino) {
00053         if (ponderado_arestas) file » peso;
00054         add_aresta(origem, destino, peso);
00055     }
00056 }
```

4.3.2.5 cobertura_gulosa()

```
int * grafo::cobertura_gulosa (
             int * tamanho)
```

Calcula uma cobertura de vértices usando a estratégia gulosa.

Parâmetros

tamanho	Ponteiro para armazenar o tamanho da cobertura encontrada.

Retorna

Um array com os vértices da cobertura.

```
00120                                           {
00121     *tamanho = 0;
00122     int* cobertura = nullptr;
00123     int capacidade = 0;
00124     Aresta* lista_arestas = nullptr;
00125     int n = get_ordem();
00126
00127     for (int i = 1; i <= n; ++i) {
00128         aresta_grafo* atual = get_vizinhos(i);
00129         aresta_grafo* temp;
00130         while (atual) {
00131             if (!direcionado && i > atual->destino) {
00132                 temp = atual;
00133                 atual = atual->proxima;
00134                 delete temp;
00135                 continue;
00136             }
00137             Aresta* nova = new Aresta;
00138             nova->origem = i;
00139             nova->destino = atual->destino;
00140             nova->next = lista_arestas;
00141             lista_arestas = nova;
```

```
00142             temp = atual;
00143             atual = atual->proxima;
00144             delete temp;
00145         }
00146     }
00147
00148     int* graus = new int[n + 1];
00149
00150     while (lista_arestas) {
00151         for (int i = 0; i <= n; i++)
00152             graus[i] = 0;
00153
00154         Aresta* atual = lista_arestas;
00155         while (atual) {
00156             graus[atual->origem]++;
00157             graus[atual->destino]++;
00158             atual = atual->next;
00159         }
00160
00161         int max_grau = -1;
00162         int escolhido = -1;
00163         for (int i = 1; i <= n; ++i) {
00164             if (graus[i] > max_grau) {
00165                 max_grau = graus[i];
00166                 escolhido = i;
00167             }
00168         }
00169
00170         if (*tamanho >= capacidade) {
00171             capacidade = (capacidade > 0 ? capacidade * 2 : 1);
00172             int* nova = new int[capacidade];
00173             for (int i = 0; i < *tamanho; i++) {
00174                 nova[i] = cobertura[i];
00175             }
00176             delete[] cobertura;
00177             cobertura = nova;
00178         }
00179         cobertura[(*tamanho)++] = escolhido;
00180
00181         Aresta** ptr = &lista_arestas;
00182         while (*ptr) {
00183             if ((*ptr)->origem == escolhido || (*ptr)->destino == escolhido) {
00184                 Aresta* temp = *ptr;
00185                 *ptr = (*ptr)->next;
00186                 delete temp;
00187             } else {
00188                 ptr = &(*ptr)->next;
00189             }
00190         }
00191     }
00192     delete[] graus;
00193     return cobertura;
00194 }
```

4.3.2.6 cobertura_randomizada()

```
int * grafo::cobertura_randomizada (
            int * tamanho)
```

Calcula uma cobertura de vértices usando uma estratégia randomizada.

Parâmetros

tamanho	Ponteiro para armazenar o tamanho da cobertura encontrada.

Retorna

Um array com os vértices da cobertura.

```
00201                                           {
00202     *tamanho = 0;
00203     int* cobertura = nullptr;
00204     int capacidade = 0;
00205     Aresta* lista_arestas = nullptr;
00206     int n = get_ordem();
```

```
00207
00208     srand((unsigned)time(nullptr));
00209
00210     for (int i = 1; i <= n; ++i) {
00211         aresta_grafo* atual = get_vizinhos(i);
00212         aresta_grafo* temp;
00213         while (atual) {
00214             if (!direcionado && i > atual->destino) {
00215                 temp = atual;
00216                 atual = atual->proxima;
00217                 delete temp;
00218                 continue;
00219             }
00220             Aresta* nova = new Aresta;
00221             nova->origem = i;
00222             nova->destino = atual->destino;
00223             nova->next = lista_arestas;
00224             lista_arestas = nova;
00225             temp = atual;
00226             atual = atual->proxima;
00227             delete temp;
00228         }
00229     }
00230
00231     int candidateCapacity = 0;
00232     int* candidatosBuffer = 0;
00233
00234     while (lista_arestas) {
00235         int contador = 0;
00236         Aresta* atual = lista_arestas;
00237         while (atual) {
00238             contador += 2;
00239             atual = atual->next;
00240         }
00241
00242         if (candidateCapacity < contador) {
00243             if (candidatosBuffer != 0) {
00244                 delete[] candidatosBuffer;
00245             }
00246             candidatosBuffer = new int[contador];
00247             candidateCapacity = contador;
00248         }
00249
00250         int pos = 0;
00251         atual = lista_arestas;
00252         while (atual) {
00253             candidatosBuffer[pos++] = atual->origem;
00254             candidatosBuffer[pos++] = atual->destino;
00255             atual = atual->next;
00256         }
00257
00258         int escolhido = candidatosBuffer[rand() % contador];
00259
00260         if (*tamanho >= capacidade) {
00261             capacidade = (capacidade > 0 ? capacidade * 2 : 1);
00262             int* nova = new int[capacidade];
00263             for (int i = 0; i < *tamanho; i++) {
00264                 nova[i] = cobertura[i];
00265             }
00266             delete[] cobertura;
00267             cobertura = nova;
00268         }
00269         cobertura[(*tamanho)++] = escolhido;
00270
00271         Aresta** ptr = &lista_arestas;
00272         while (*ptr) {
00273             if ((*ptr)->origem == escolhido || (*ptr)->destino == escolhido) {
00274                 Aresta* temp = *ptr;
00275                 *ptr = (*ptr)->next;
00276                 delete temp;
00277             } else {
00278                 ptr = &(*ptr)->next;
00279             }
00280         }
00281     }
00282
00283     if (candidatosBuffer != 0) {
00284         delete[] candidatosBuffer;
00285     }
00286     return cobertura;
00287 }
```

4.3.2.7 cobertura_reativa()

```
int * grafo::cobertura_reativa (
            int * tamanho)
```

Calcula uma cobertura reativa de vértices combinando estratégias gulosa e randomizada.

Parâmetros

tamanho	Ponteiro para armazenar o tamanho da melhor cobertura encontrada.

Retorna

Um array com os vértices da melhor cobertura.

```
00329                                           {
00330     const int MAX_ITER = 50;
00331     const double INITIAL_PROB = 0.5;
00332     double prob_guloso = INITIAL_PROB;
00333     int* melhor = 0;
00334     int menor = INT_MAX;
00335     int falhas_guloso = 0, falhas_random = 0;
00336 
00337     for (int iter = 0; iter < MAX_ITER; ++iter) {
00338         int* cobertura;
00339         int temp_size;
00340         bool estrategia_gulosa = ((double)rand() / RAND_MAX < prob_guloso);
00341 
00342         if (estrategia_gulosa) {
00343             cobertura = cobertura_gulosa(&temp_size);
00344         } else {
00345             cobertura = cobertura_randomizada(&temp_size);
00346         }
00347 
00348         int erros = verificar_erro(cobertura, temp_size);
00349         if (erros == 0) {
00350             if (temp_size < menor) {
00351                 delete[] melhor;
00352                 menor = temp_size;
00353                 melhor = new int[menor];
00354                 for (int i = 0; i < menor; i++) {
00355                     melhor[i] = cobertura[i];
00356                 }
00357                 *tamanho = menor;
00358             }
00359         } else {
00360             if (estrategia_gulosa) falhas_guloso++;
00361             else falhas_random++;
00362         }
00363 
00364         if (falhas_guloso + falhas_random > 0) {
00365             prob_guloso = 1.0 - ((double) falhas_guloso / (falhas_guloso + falhas_random));
00366         }
00367 
00368         delete[] cobertura;
00369     }
00370 
00371     return melhor;
00372 }
```

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.

```
00062                         {
00063     int n = get_ordem();
00064     for (int i = 1; i <= n; ++i) {
00065         for (int j = 1; j <= n; ++j) {
00066             if (i != j && !existe_aresta(i, j)) {
00067                 if (direcionado) return false;
00068                 if (!existe_aresta(j, i)) return false;
00069             }
00070         }
00071     }
00072     return true;
00073 }
```

4.3.2.9 eh_direcionado()

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

Retorna as flags: direcionado, ponderado_vertices e ponderado_arestas.

```
00389 { return direcionado; }
```

4.3.2.10 exibe_descricao()

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

Exibe a descrição do grafo.

```
00377                             {
00378     std::cout « "Grau: " « get_grau() « std::endl;
00379     std::cout « "Ordem: " « get_ordem() « std::endl;
00380     std::cout « "Direcionado: " « (eh_direcionado() ? "Sim" : "Nao") « std::endl;
00381     std::cout « "Vertices ponderados: " « (vertice_ponderado() ? "Sim" : "Nao") « std::endl;
00382     std::cout « "Arestas ponderadas: " « (aresta_ponderada() ? "Sim" : "Nao") « std::endl;
00383     std::cout « "Completo: " « (eh_completo() ? "Sim" : "Nao") « std::endl;
00384 }
```

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.

```
00091                     {
00092     int grau_maximo = 0;
00093     int n = get_ordem();
00094     for (int i = 1; i <= n; ++i) {
00095         int grau_atual = 0;
00096         aresta_grafo* vizinhos = get_vizinhos(i);
00097         aresta_grafo* atual = vizinhos;
00098         while (atual) {
00099             grau_atual++;
00100             atual = atual->proxima;
00101         }
00102         liberar_arestas_temp(vizinhos);
00103 
00104         if (direcionado) {
00105             for (int j = 1; j <= n; ++j) {
00106                 if (existe_aresta(j, i)) grau_atual++;
00107             }
00108         }
00109 
00110         if (grau_atual > grau_maximo) grau_maximo = grau_atual;
00111     }
00112     return grau_maximo;
00113 }
```

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)
```

Verifica se a cobertura de vértices encontrada cobre todas as arestas.

Parâmetros

cobertura	Array com os vértices da cobertura.
tamanho	Tamanho do array.

Retorna

Número de arestas não cobertas.

```
00295                                                   {
00296     int erros = 0;
00297     int n = get_ordem();
00298
00299     for (int i = 1; i <= n; ++i) { // IDs dos nós começam em 1
00300         aresta_grafo* vizinhos = get_vizinhos(i);
00301         aresta_grafo* atual = vizinhos;
00302         while (atual) {
00303             int u = i;
00304             int v = atual->destino;
00305             bool coberta = false;
00306
00307             for (int j = 0; j < tamanho; ++j) {
00308                 if (cobertura[j] == u || cobertura[j] == v) {
00309                     coberta = true;
00310                     break;
00311                 }
00312             }
00313
00314             if (!coberta) erros++;
00315
00316             aresta_grafo* temp = atual;
00317             atual = atual->proxima;
00318             delete temp;
00319         }
00320     }
00321     return (direcionado) ? erros : erros / 2;
00322 }
```

4.3.2.18 vertice_ponderado()

```
bool grafo::vertice_ponderado () const
00390 { 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:

- trabalho-final-grupo/include/grafo.h
- trabalho-final-grupo/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
- int * cobertura_gulosa (int *tamanho)

  Calcula uma cobertura de vértices usando a estratégia gulosa.
- int * cobertura_randomizada (int *tamanho)

  Calcula uma cobertura de vértices usando uma estratégia randomizada.
- int * cobertura_reativa (int *tamanho)

  Calcula uma cobertura reativa de vértices combinando estratégias gulosa e randomizada.
- int verificar_erro (int *cobertura, int tamanho)

  Verifica se a cobertura de vértices encontrada cobre todas as arestas.
- 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 }
```

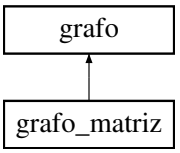
A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- trabalho-final-grupo/include/grafo_lista.h
- trabalho-final-grupo/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

  Destrutor da classe grafo_matriz.
- 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 * cobertura_gulosa (int *tamanho)

  Calcula uma cobertura de vértices usando a estratégia gulosa.
- int * cobertura_randomizada (int *tamanho)

  Calcula uma cobertura de vértices usando uma estratégia randomizada.
- int * cobertura_reativa (int *tamanho)

  Calcula uma cobertura reativa de vértices combinando estratégias gulosa e randomizada.
- int verificar_erro (int *cobertura, int tamanho)

  Verifica se a cobertura de vértices encontrada cobre todas as arestas.
- 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) {
00014     num_nos = 0;
00015 }
```

4.5.1.2 ~grafo_matriz()

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

Destrutor da classe grafo_matriz.

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 }
```

4.5.2 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.

```
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 }
```

4.5.2.2 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.

```
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 }
```

4.5.2.3 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.

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

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.

```
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 }
```

4.5.2.5 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.

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

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.

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

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.

```
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:

- trabalho-final-grupo/include/grafo_matriz.h
- trabalho-final-grupo/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:

- trabalho-final-grupo/include/no_grafo.h
- trabalho-final-grupo/src/no_grafo.cpp

Capítulo 5

Arquivos

5.1 Referência do Arquivo trabalho-final-grupo/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 trabalho-final-grupo/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     int* cobertura_gulosa(int* tamanho);
00029     int* cobertura_randomizada(int* tamanho);
00030     int* cobertura_reativa(int* tamanho);
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 trabalho-final-grupo/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 trabalho-final-grupo/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 trabalho-final-grupo/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 trabalho-final-grupo/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 ()
- bool validar_argumentos (int argc, char ∗argv[ ])

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

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

  Função principal.

5.11.1 Descrição detalhada

Programa principal.

5.11.2 Funções

5.11.2.1 executar_cobertura()

```
void executar_cobertura (
            grafo * g)
```

Executa o algoritmo de cobertura selecionado.

Parâmetros

g	Ponteiro para o 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 tamanho;
00071     int* cobertura = nullptr;
00072     clock_t inicio = clock();
00073 
00074     switch(escolha) {
00075         case 1:
00076             cobertura = g->cobertura_gulosa(&tamanho);
00077         break;
00078         case 2:
00079             cobertura = g->cobertura_randomizada(&tamanho);
00080         break;
00081         case 3:
00082             cobertura = g->cobertura_reativa(&tamanho);
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 « "\nCobertura encontrada\n";
00092     for(int i = 0; i < tamanho; i++) {
00093         cout « cobertura[i] « " ";
00094     }
00095     cout « "\n\nTamanho da cobertura: " « tamanho;
00096     cout « "\nTempo de execução: " « tempo « "s\n\n";
00097 
00098     delete[] cobertura;
00099 }
```

5.11.2.2 exibir_uso()

```
void exibir_uso ()
```

```
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.

```
00107                                   {
00108     if (!validar_argumentos(argc, argv)) {
00109         return 1;
00110     }
00111
00112     const string modo = argv[1];
00113     const string estrutura = argv[2];
00114     const string arquivo = argv[3];
00115
00116     grafo* g = nullptr;
00117
00118     try {
00119         if(estrutura == "-m") {
00120             g = new grafo_matriz();
00121         } else {
00122             g = new grafo_lista();
00123         }
00124
00125         g->carrega_grafo(arquivo);
00126
00127         if(modo == "-d") {
00128             g->exibe_descricao();
00129         } else {
00130             executar_cobertura(g);
00131         }
00132
00133         delete g;
00134     }
00135     catch(const exception& e) {
00136         cerr « "Erro: " « e.what() « endl;
00137         if(g) delete g;
00138         return 1;
00139     }
00140
00141     return 0;
00142 }
```

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 trabalho-final-grupo/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 trabalho-final-grupo/src/grafo.cpp

Implementação da classe grafo.

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

Componentes

- struct Aresta

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.

```
00079                                                {
00080     while (cabeca) {
00081         aresta_grafo* temp = cabeca;
00082         cabeca = cabeca->proxima;
00083         delete temp;
00084     }
00085 }
```

5.14 Referência do Arquivo trabalho-final-grupo/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 trabalho-final-grupo/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 trabalho-final-grupo/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

~aresta_grafo
  aresta_grafo, 8
~grafo
  grafo, 10
~grafo_lista
  grafo_lista, 19
~grafo_matriz
  grafo_matriz, 24
~no_grafo
  no_grafo, 29

add_aresta
  grafo, 10
  grafo_lista, 20
  grafo_matriz, 25
add_no
  grafo, 10
  grafo_lista, 20
  grafo_matriz, 25
Aresta, 7
  destino, 7
  next, 7
  origem, 7
aresta_grafo, 7
  ~aresta_grafo, 8
  aresta_grafo, 8
  destino, 8
  peso, 8
  proxima, 9
aresta_ponderada
  grafo, 10

carrega_grafo
  grafo, 10
cobertura_gulosa
  grafo, 11
cobertura_randomizada
  grafo, 12
cobertura_reativa
  grafo, 13

destino
  Aresta, 7
  aresta_grafo, 8
direcionado
  grafo, 17

eh_completo
  grafo, 14
eh_direcionado
  grafo, 15
executar_cobertura
  main.cpp, 35
exibe_descricao
  grafo, 15
exibir_uso
  main.cpp, 36
existe_aresta
  grafo, 15
  grafo_lista, 21
  grafo_matriz, 26

get_aresta
  grafo, 15
  grafo_lista, 21
  grafo_matriz, 26
get_grau
  grafo, 15
get_no
  grafo, 16
  grafo_lista, 21
  grafo_matriz, 27
get_ordem
  grafo, 16
  grafo_lista, 22
  grafo_matriz, 27
get_vizinhos
  grafo, 16
  grafo_lista, 22
  grafo_matriz, 27
grafo, 9
  ~grafo, 10
  add_aresta, 10
  add_no, 10
  aresta_ponderada, 10
  carrega_grafo, 10
  cobertura_gulosa, 11
  cobertura_randomizada, 12
  cobertura_reativa, 13
  direcionado, 17
  eh_completo, 14
  eh_direcionado, 15
  exibe_descricao, 15
  existe_aresta, 15
  get_aresta, 15
  get_grau, 15
  get_no, 16
  get_ordem, 16
  get_vizinhos, 16

grafo, 10
num_nos, 17
ponderado_arestas, 17
ponderado_vertices, 17
verificar_erro, 16
vertice_ponderado, 17
grafo.cpp
liberar_arestas_temp, 38
grafo_lista, 18
~grafo_lista, 19
add_aresta, 20
add_no, 20
existe_aresta, 21
get_aresta, 21
get_no, 21
get_ordem, 22
get_vizinhos, 22
grafo_lista, 19
grafo_matriz, 23
~grafo_matriz, 24
add_aresta, 25
add_no, 25
existe_aresta, 26
get_aresta, 26
get_no, 27
get_ordem, 27
get_vizinhos, 27
grafo_matriz, 24

id
no_grafo, 29

liberar_arestas_temp
grafo.cpp, 38

main
main.cpp, 36
main.cpp
executar_cobertura, 35
exibir_uso, 36
main, 36
validar_argumentos, 37

next
Aresta, 7
no_grafo, 28
~no_grafo, 29
id, 29
no_grafo, 28
peso, 29
primeira_aresta, 29
proximo, 29
num_nos
grafo, 17

origem
Aresta, 7

peso
aresta_grafo, 8
no_grafo, 29
ponderado_arestas
grafo, 17
ponderado_vertices
grafo, 17
primeira_aresta
no_grafo, 29
proxima
aresta_grafo, 9
proximo
no_grafo, 29

trabalho-final-grupo/include/aresta_grafo.h, 31
trabalho-final-grupo/include/grafo.h, 32
trabalho-final-grupo/include/grafo_lista.h, 33
trabalho-final-grupo/include/grafo_matriz.h, 33, 34
trabalho-final-grupo/include/no_grafo.h, 34, 35
trabalho-final-grupo/main.cpp, 35
trabalho-final-grupo/src/aresta_grafo.cpp, 38
trabalho-final-grupo/src/grafo.cpp, 38
trabalho-final-grupo/src/grafo_lista.cpp, 39
trabalho-final-grupo/src/grafo_matriz.cpp, 39
trabalho-final-grupo/src/no_grafo.cpp, 39

validar_argumentos
main.cpp, 37
verificar_erro
grafo, 16
vertice_ponderado
grafo, 17