Fase 2 - Grafos com Listas

Gerado por Doxygen 1.13.2

Capítulo 1

Índice das estruturas de dados

1.1 Estruturas de dados

Lista das estruturas de dados com uma breve descrição:

Capítulo 2

Índice dos ficheiros

2.1 Lista de ficheiros

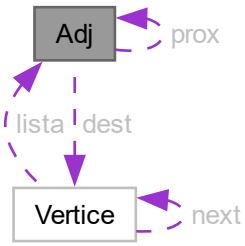
Lista de todos os ficheiros documentados com uma breve descrição:

Capítulo 3

Documentação da estruturas de dados

3.1 Referência à estrutura Adj

Diagrama de colaboração para Adj:



Campos de Dados

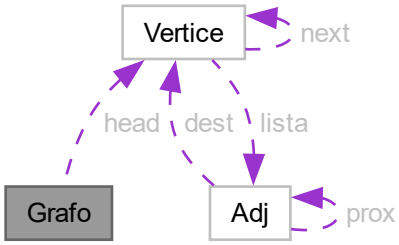
- Vertice ∗ **dest**
- Adj ∗ **prox**

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

- grafo.h

3.2 Referência à estrutura Grafo

Diagrama de colaboração para Grafo:



Campos de Dados

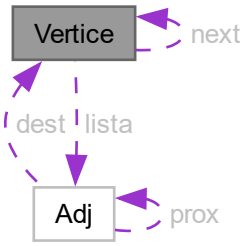
- Vertice ∗ **head**
- int **n_vertices**

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

- grafo.h

3.3 Referência à estrutura Vertice

Diagrama de colaboração para Vertice:



Campos de Dados

- int **id**
- int **x**
- int **y**
- char **freq**
- Adj ∗ **lista**
- Vertice ∗ **next**
- bool **visited**

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

- grafo.h

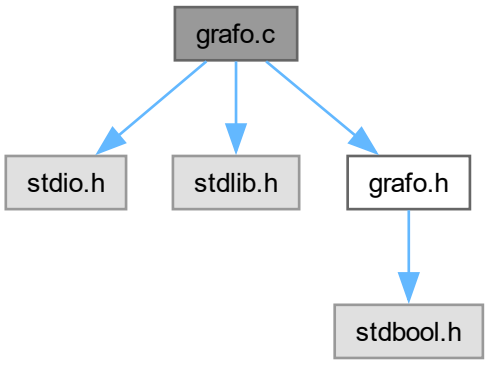
Capítulo 4

Documentação do ficheiro

4.1 Referência ao ficheiro grafo.c

```
#include <stdio.h>
#include <stdlib.h>
#include "grafo.h"
```

Diagrama de dependências de inclusão para grafo.c:



Funções

- void inicializaGrafo (Grafo *g)

  Inicializa o grafo como vazio.
- int adicionaVertice (Grafo *g, int x, int y, char freq)

  Adiciona um novo vértice/antena ao grafo.
- bool adicionaAresta (Grafo *g, int orig_id, int dest_id)

  Adiciona uma aresta entre dois vértices.
- void imprimeGrafo (Grafo *g)

  Imprime o grafo em formato legível.

- bool dfs (Grafo *g, int start_id)

  Executa Depth-First Search (DFS) a partir de um vértice dado.
- bool bfs (Grafo *g, int start_id)

  Executa Breadth-First Search (BFS) a partir de um vértice dado.
- bool listarCaminhos (Grafo *g, int origem, int destino)

  Lista todos os caminhos simples entre dois vértices de mesma frequência.
- void listarIntersecoes (Grafo *g, char freq1, char freq2)

  Lista todas as combinações de pares de antenas de frequências distintas.
- bool removerVertice (Grafo *g, int id)

  Remove um vértice do grafo, junto com todas as arestas incidentes.

4.1.1 Descrição detalhada

Autor

Daniel Vilaça (a16939@alunos.ipca.pt)

Versão

1.0

Data

2025-05-31

Copyright

Copyright (c) 2025

4.1.2 Documentação das funções

4.1.2.1 adicionaAresta()

```
bool adicionaAresta (
            Grafo * g,
            int orig_id,
            int dest_id)
```

Adiciona uma aresta entre dois vértices.

Adiciona uma aresta não dirigida entre dois vértices do grafo.

Parâmetros

	g	Apontador para o grafo que contém os vértices.
	orig_id	ID do vértice de origem.
	dest_id	ID do vértice de destino.

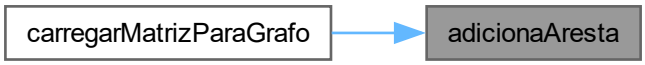
Retorna

true Se a aresta for criada com sucesso ou já existir.

false Se "orig_id" ou "dest_id" forem inválidos, frequências diferentes, mesmo vértice, ou falha de memória.

```
00128                                                           {
00129     Vertice *v1 = getVerticePorID(g, orig_id);
00130     Vertice *v2 = getVerticePorID(g, dest_id);
00131     if (!v1 || !v2) return false;
00132     if (v1->freq != v2->freq) return false;
00133     if (v1 == v2) return false;
00134     if (arestaExiste(v1, v2)) return true;
00135
00136
00137     Adj *a1 = malloc(sizeof *a1);
00138     if (!a1) return false;
00139     a1->dest = v2;
00140     a1->prox = v1->lista;
00141     v1->lista = a1;
00142
00143
00144     Adj *a2 = malloc(sizeof *a2);
00145     if (!a2) {
00146
00147         Adj *undo = v1->lista;
00148         v1->lista = undo->prox;
00149         free(undo);
00150         return false;
00151     }
00152     a2->dest = v1;
00153     a2->prox = v2->lista;
00154     v2->lista = a2;
00155
00156     return true;
00157 }
```

Este é o diagrama das funções que utilizam esta função:



4.1.2.2 adicionaVertice()

```
int adicionaVertice (
            Grafo * g,
            int x,
            int y,
            char freq)
```

Adiciona um novo vértice/antena ao grafo.

Adiciona um novo vértice (antena) ao grafo.

Parâmetros

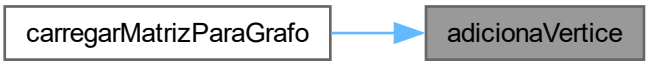
	g	Apontador para o grafo onde o vértice será inserido.
	x	Coordenada x (coluna) da antena.
	y	Coordenada y (linha) da antena.
	freq	Char que representa a frequência/identificador da antena.

Retorna

int ID atribuído ao novo vértice, ou -1 em caso de erro de alocação.

```
00099                                                       {
00100     Vertice *novo = malloc(sizeof *novo);
00101     if (!novo) return -1;
00102
00103     novo->id       = g->n_vertices;
00104     novo->x        = x;
00105     novo->y        = y;
00106     novo->freq     = freq;
00107     novo->lista    = NULL;
00108     novo->visited  = false;
00109
00110     // Cabeça da lista
00111     novo->next     = g->head;
00112     g->head        = novo;
00113     g->n_vertices++;
00114
00115     return novo->id;
00116 }
```

Este é o diagrama das funções que utilizam esta função:



4.1.2.3 bfs()

```
bool bfs (
              Grafo * g,
              int start_id)
```

Executa Breadth-First Search (BFS) a partir de um vértice dado.

Parâmetros

	g	Apontador para o grafo onde será feita a BFS.
	start_id	Identificador do vértice inicial da procura.

Retorna

true Se a procura for executada (mesmo que visite apenas um vértice).

false Se `start_id` não corresponder a nenhum vértice ou falhar alocação.

```
00226                                   {
00227     Vertice *start = getVerticePorID(g, start_id);
00228     if (!start) {
00229         printf("ID invalido.\n");
00230         return false;
00231     }
00232     clearVisited(g);
00233
00234
00235     Vertice **fila = malloc(sizeof(Vertice *) * g->n_vertices);
00236     if (!fila) return false;
00237
```

```
00238     int ini = 0, fim = 0;
00239     fila[fim++] = start;
00240     start->visited = true;
00241     printf("BFS a partir da antena %d\n", start_id);
00242
00243     while (ini < fim) {
00244         Vertice *v = fila[ini++];
00245         printf("Visitado BFS: Antena %d [%c] em (%d,%d)\n",
00246                v->id, v->freq, v->x, v->y);
00247         for (Adj *a = v->lista; a; a = a->prox) {
00248             if (!a->dest->visited) {
00249                 a->dest->visited = true;
00250                 fila[fim++] = a->dest;
00251             }
00252         }
00253     }
00254
00255     free(fila);
00256     return true;
00257 }
```

4.1.2.4 dfs()

```
bool dfs (
              Grafo * g,
              int start_id)
```

Executa Depth-First Search (DFS) a partir de um vértice dado.

Parâmetros

	g	Apontador para o grafo onde será feita a DFS.
	start_id	ID do vértice inicial da procura.

Retorna

true Se a procura for executada (mesmo que encontre apenas um vértice).

false Se "start_id" não corresponder a nenhum vértice existente.

```
00205                                 {
00206     Vertice *start = getVerticePorID(g, start_id);
00207     if (!start) {
00208         printf("ID invalido.\n");
00209         return false;
00210     }
00211     clearVisited(g);
00212     printf("DFS a partir da antena %d\n", start_id);
00213     dfs_rec(start);
00214     return true;
00215 }
```

4.1.2.5 imprimeGrafo()

```
void imprimeGrafo (
              Grafo * g)
```

Imprime o grafo em formato legível.

Imprime o grafo de forma legível, listando cada vértice e as suas adjacências.

Parâmetros

	g	Apontador para o grafo a ser impresso.

```
00165                               {
00166
00167     for (int i = 0; i < g->n_vertices; i++) {
00168         Vertice *v = getVerticePorID(g, i);
00169         if (!v) continue;
00170         printf("Antena %d [%c] em (%d,%d) -> ",
00171                v->id, v->freq, v->x, v->y);
00172         for (Adj *a = v->lista; a; a = a->prox) {
00173             printf("%d ", a->dest->id);
00174         }
00175         printf("\n");
00176     }
00177 }
```

4.1.2.6 inicializaGrafo()

```
void inicializaGrafo (
             Grafo * g)
```

Inicializa o grafo como vazio.

Inicializa um grafo vazio.

Parâmetros

	g	Apontador para o grafo a ser inicializado.

```
00084                                 {
00085     g->head = NULL;
00086     g->n_vertices = 0;
00087 }
```

4.1.2.7 listarCaminhos()

```
bool listarCaminhos (
             Grafo * g,
             int origem,
             int destino)
```

Lista todos os caminhos simples entre dois vértices de mesma frequência.

Parâmetros

	g	Apontador para o grafo onde serão procurados os caminhos.
	origem	ID do vértice de origem.
	destino	ID do vértice de destino.

Retorna

true Se os caminhos foram listados (ou se pelo menos tentativa foi válida).

false Se IDs inválidos ou frequências diferentes.

```
00305                                                   {
00306     Vertice *src = getVerticePorID(g, origem);
00307     Vertice *dst = getVerticePorID(g, destino);
00308     if (!src || !dst) {
00309         printf("ID invalido.\n");
00310         return false;
00311     }
00312     if (src->freq != dst->freq) {
00313         printf("Antenas com frequencias diferentes nao estao ligadas.\n");
00314         return false;
00315     }
00316
00317     clearVisited(g);
00318     Vertice **stack = malloc(sizeof(Vertice *) * g->n_vertices);
00319     if (!stack) return false;
00320
00321     printf("Caminhos entre %d e %d:\n", origem, destino);
00322     listarCaminhosRec(src, dst, stack, 0);
00323     free(stack);
00324     return true;
00325 }
```

4.1.2.8 listarIntersecoes()

```
void listarIntersecoes (
            Grafo * g,
            char freq1,
            char freq2)
```

Lista todas as combinações de pares de antenas de frequências distintas.

Parâmetros

	g	Apontador para o grafo onde se procura as interseções.
	freq1	Char da primeira frequência.
	freq2	Char da segunda frequência.

```
00335                                                       {
00336     if (freq1 == freq2) {
00337         printf("As frequencias devem ser diferentes.\n");
00338         return;
00339     }
00340     printf("Intersecoes entre antenas com frequencias '%c' e '%c':\n",
00341            freq1, freq2);
00342
00343     for (Vertice *v = g->head; v; v = v->next) {
00344         if (v->freq == freq1) {
00345             for (Vertice *w = g->head; w; w = w->next) {
00346                 if (w->freq == freq2) {
00347                     printf("Antena [%c] em (%d,%d)  <->  Antena [%c] em (%d,%d)\n",
00348                            freq1, v->x, v->y,
00349                            freq2, w->x, w->y);
00350                 }
00351             }
00352         }
00353     }
00354 }
```

4.1.2.9 removerVertice()

```
bool removerVertice (
            Grafo * g,
            int id)
```

Remove um vértice do grafo, junto com todas as arestas incidentes.

Remove um vértice do grafo, juntamente com todas as arestas incidentes.

Parâmetros

	g	Apontador para o grafo onde está o vértice.
	id	Identificador do vértice a ser removido.

Retorna

true Se o vértice existia e foi removido com sucesso.

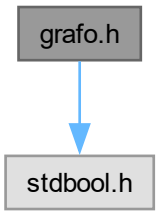
false Se não houver vértice com esse ID.

```
00365                                       {
00366     Vertice *alvo = getVerticePorID(g, id);
00367     if (!alvo) return false;
00368
00369     for (Vertice *v = g->head; v; v = v->next) {
00370         if (v != alvo) {
00371             removerArestaDireta(v, alvo);
00372         }
00373     }
00374
00375     Adj *a = alvo->lista;
00376     while (a) {
00377         Adj *aux = a->prox;
00378         free(a);
00379         a = aux;
00380     }
00381
00382     Vertice *prev = NULL, *cur = g->head;
00383     while (cur) {
00384         if (cur == alvo) {
00385             if (prev) prev->next = cur->next;
00386             else      g->head = cur->next;
00387             free(cur);
00388             break;
00389         }
00390         prev = cur;
00391         cur  = cur->next;
00392     }
00393
00394     g->n_vertices--;
00395
00396     int contador = 0;
00397     for (Vertice *v = g->head; v; v = v->next) {
00398         v->id = contador++;
00399     }
00400
00401     return true;
00402 }
```

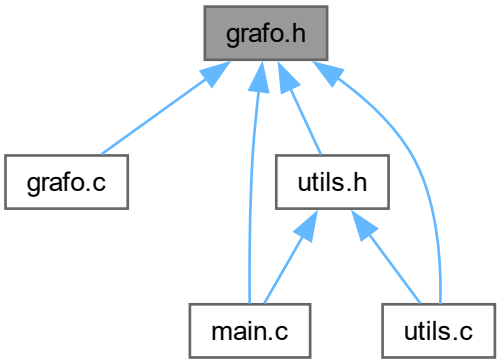
4.2 Referência ao ficheiro grafo.h

`#include <stdbool.h>`

Diagrama de dependências de inclusão para grafo.h:



Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Estruturas de Dados

- struct Adj
- struct Vertice
- struct Grafo

Definições de tipos

- typedef struct Adj **Adj**
- typedef struct Vertice **Vertice**

Funções

- void inicializaGrafo (Grafo ∗g)

  Inicializa um grafo vazio.
- int adicionaVertice (Grafo ∗g, int x, int y, char freq)

  Adiciona um novo vértice (antena) ao grafo.
- bool adicionaAresta (Grafo ∗g, int orig_id, int dest_id)

  Adiciona uma aresta não dirigida entre dois vértices do grafo.
- void imprimeGrafo (Grafo ∗g)

  Imprime o grafo de forma legível, listando cada vértice e as suas adjacências.
- bool dfs (Grafo ∗g, int start_id)

  Executa Depth-First Search (DFS) a partir de um vértice dado.
- bool bfs (Grafo ∗g, int start_id)

  Executa Breadth-First Search (BFS) a partir de um vértice dado.
- bool listarCaminhos (Grafo ∗g, int origem, int destino)

  Lista todos os caminhos simples entre dois vértices de mesma frequência.
- void listarIntersecoes (Grafo ∗g, char freq1, char freq2)

  Lista todas as combinações de pares de antenas de frequências distintas.
- bool removerVertice (Grafo ∗g, int id)

  Remove um vértice do grafo, juntamente com todas as arestas incidentes.

4.2.1 Descrição detalhada

Autor

Daniel Vilaça (a16939@alunos.ipca.pt)

Versão

1.0

Data

2025-05-31

Copyright

Copyright (c) 2025

4.2.2 Documentação das funções

4.2.2.1 adicionaAresta()

```
bool adicionaAresta (
            Grafo * g,
            int orig_id,
            int dest_id)
```

Adiciona uma aresta não dirigida entre dois vértices do grafo.

Parâmetros

	g	Apontador para o grafo onde se quer criar a aresta.
	orig_id	ID do vértice de origem.
	dest_id	ID do vértice de destino.

Retorna

true Se a aresta for criada com sucesso ou já existir.

false Se algum ID for inválido, se as frequências forem diferentes, se tentar ligar o vértice a si próprio, ou se falhar alocação.

Adiciona uma aresta não dirigida entre dois vértices do grafo.

Parâmetros

	g	Apontador para o grafo que contém os vértices.
	orig_id	ID do vértice de origem.
	dest_id	ID do vértice de destino.

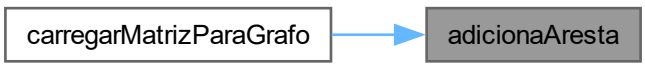
Retorna

true Se a aresta for criada com sucesso ou já existir.

false Se "orig_id" ou "dest_id" forem inválidos, frequências diferentes, mesmo vértice, ou falha de memória.

```
00128                                                                {
00129     Vertice *v1 = getVerticePorID(g, orig_id);
00130     Vertice *v2 = getVerticePorID(g, dest_id);
00131     if (!v1 || !v2) return false;
00132     if (v1->freq != v2->freq) return false;
00133     if (v1 == v2) return false;
00134     if (arestaExiste(v1, v2)) return true;
00135
00136
00137     Adj *a1 = malloc(sizeof *a1);
00138     if (!a1) return false;
00139     a1->dest = v2;
00140     a1->prox = v1->lista;
00141     v1->lista = a1;
00142
00143
00144     Adj *a2 = malloc(sizeof *a2);
00145     if (!a2) {
00146
00147         Adj *undo = v1->lista;
00148         v1->lista = undo->prox;
00149         free(undo);
00150         return false;
00151     }
00152     a2->dest = v1;
00153     a2->prox = v2->lista;
00154     v2->lista = a2;
00155
00156     return true;
00157 }
```

Este é o diagrama das funções que utilizam esta função:



4.2.2.2 adicionaVertice()

```
int adicionaVertice (
            Grafo * g,
            int x,
            int y,
            char freq)
```

Adiciona um novo vértice (antena) ao grafo.

Parâmetros

	g	Apontador para o grafo onde o vértice será inserido.
	x	Coordenada x (coluna) da antena.
	y	Coordenada y (linha) da antena.
	freq	Char que representa a frequência/identificador da antena.

Retorna

int ID atribuído ao novo vértice se a inserção for bem-sucedida, -1 se ocorrer falha de alocação.

Adiciona um novo vértice (antena) ao grafo.

Parâmetros

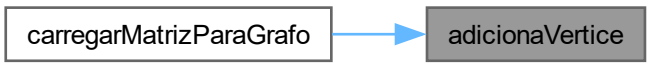
	g	Apontador para o grafo onde o vértice será inserido.
	x	Coordenada x (coluna) da antena.
	y	Coordenada y (linha) da antena.
	freq	Char que representa a frequência/identificador da antena.

Retorna

int ID atribuído ao novo vértice, ou -1 em caso de erro de alocação.

```
00099                                                       {
00100     Vertice *novo = malloc(sizeof *novo);
00101     if (!novo) return -1;
00102
00103     novo->id       = g->n_vertices;
00104     novo->x        = x;
00105     novo->y        = y;
00106     novo->freq     = freq;
00107     novo->lista    = NULL;
00108     novo->visited  = false;
00109
00110     // Cabeça da lista
00111     novo->next     = g->head;
00112     g->head        = novo;
00113     g->n_vertices++;
00114
00115     return novo->id;
00116 }
```

Este é o diagrama das funções que utilizam esta função:



4.2.2.3 bfs()

```
bool bfs (
            Grafo * g,
            int start_id)
```

Executa Breadth-First Search (BFS) a partir de um vértice dado.

Parâmetros

	g	Apontador para o grafo onde será feita a BFS.
	start_id	ID do vértice inicial da procura.

Retorna

true Se a procura foi executada (mesmo que só visite um vértice).

false Se `start_id` não corresponder a nenhum vértice ou falhar alocação.

Parâmetros

	g	Apontador para o grafo onde será feita a BFS.
	start_id	Identificador do vértice inicial da procura.

Retorna

true Se a procura for executada (mesmo que visite apenas um vértice).

false Se `start_id` não corresponder a nenhum vértice ou falhar alocação.

```
00226                                   {
00227     Vertice *start = getVerticePorID(g, start_id);
00228     if (!start) {
00229         printf("ID invalido.\n");
00230         return false;
00231     }
00232     clearVisited(g);
00233
00234
00235     Vertice **fila = malloc(sizeof(Vertice *) * g->n_vertices);
00236     if (!fila) return false;
00237
00238     int ini = 0, fim = 0;
00239     fila[fim++] = start;
00240     start->visited = true;
00241     printf("BFS a partir da antena %d\n", start_id);
00242
00243     while (ini < fim) {
00244         Vertice *v = fila[ini++];
00245         printf("Visitado BFS: Antena %d [%c] em (%d,%d)\n",
00246                v->id, v->freq, v->x, v->y);
00247         for (Adj *a = v->lista; a; a = a->prox) {
00248             if (!a->dest->visited) {
00249                 a->dest->visited = true;
00250                 fila[fim++] = a->dest;
00251             }
00252         }
00253     }
00254
00255     free(fila);
00256     return true;
00257 }
```

4.2.2.4 dfs()

```
bool dfs (
              Grafo * g,
              int start_id)
```

Executa Depth-First Search (DFS) a partir de um vértice dado.

Parâmetros

	g	Apontador para o grafo onde será feita a DFS.
	start_id	ID do vértice inicial da procura.

Retorna

true Se a procura foi executada (mesmo que só visite um vértice).

false Se `start_id` não corresponder a nenhum vértice.

Parâmetros

	g	Apontador para o grafo onde será feita a DFS.
	start_id	ID do vértice inicial da procura.

Retorna

true Se a procura for executada (mesmo que encontre apenas um vértice).

false Se "start_id" não corresponder a nenhum vértice existente.

```
00205                                   {
00206     Vertice *start = getVerticePorID(g, start_id);
00207     if (!start) {
00208         printf("ID invalido.\n");
00209         return false;
00210     }
00211     clearVisited(g);
00212     printf("DFS a partir da antena %d\n", start_id);
00213     dfs_rec(start);
00214     return true;
00215 }
```

4.2.2.5 imprimeGrafo()

```
void imprimeGrafo (
            Grafo * g)
```

Imprime o grafo de forma legível, listando cada vértice e as suas adjacências.

Parâmetros

	g	Apontador para o grafo a ser impresso.

Imprime o grafo de forma legível, listando cada vértice e as suas adjacências.

Parâmetros

	g	Apontador para o grafo a ser impresso.

```
00165                               {
00166
00167     for (int i = 0; i < g->n_vertices; i++) {
00168         Vertice *v = getVerticePorID(g, i);
00169         if (!v) continue;
00170         printf("Antena %d [%c] em (%d,%d) -> ",
00171                v->id, v->freq, v->x, v->y);
00172         for (Adj *a = v->lista; a; a = a->prox) {
00173             printf("%d ", a->dest->id);
00174         }
00175         printf("\n");
00176     }
00177 }
```

4.2.2.6 inicializaGrafo()

```
void inicializaGrafo (
            Grafo * g)
```

Inicializa um grafo vazio.

Parâmetros

	g	Apontador para o grafo a inicializar.

Inicializa um grafo vazio.

Parâmetros

	g	Apontador para o grafo a ser inicializado.

```
00084                                {
00085     g->head = NULL;
00086     g->n_vertices = 0;
00087 }
```

4.2.2.7 listarCaminhos()

```
bool listarCaminhos (
             Grafo * g,
             int origem,
             int destino)
```

Lista todos os caminhos simples entre dois vértices de mesma frequência.

Parâmetros

	g	Apontador para o grafo onde serão procurados os caminhos.
	origem	ID do vértice de origem.
	destino	ID do vértice de destino.

Retorna

true Se os caminhos foram listados (ou se a invocação foi válida).

false Se algum ID for inválido ou frequências diferentes.

Parâmetros

	g	Apontador para o grafo onde serão procurados os caminhos.
	origem	ID do vértice de origem.
	destino	ID do vértice de destino.

Retorna

true Se os caminhos foram listados (ou se pelo menos tentativa foi válida).

false Se IDs inválidos ou frequências diferentes.

```
00305                                                    {
00306     Vertice *src = getVerticePorID(g, origem);
00307     Vertice *dst = getVerticePorID(g, destino);
00308     if (!src || !dst) {
00309         printf("ID invalido.\n");
00310         return false;
00311     }
00312     if (src->freq != dst->freq) {
00313         printf("Antenas com frequencias diferentes nao estao ligadas.\n");
00314         return false;
00315     }
00316
00317     clearVisited(g);
00318     Vertice **stack = malloc(sizeof(Vertice *) * g->n_vertices);
00319     if (!stack) return false;
00320
00321     printf("Caminhos entre %d e %d:\n", origem, destino);
00322     listarCaminhosRec(src, dst, stack, 0);
00323     free(stack);
00324     return true;
00325 }
```

4.2.2.8 listarIntersecoes()

```
void listarIntersecoes (
            Grafo * g,
            char freq1,
            char freq2)
```

Lista todas as combinações de pares de antenas de frequências distintas.

Parâmetros

	g	Apontador para o grafo onde se procura as interseções.
	freq1	Char representando a primeira frequência.
	freq2	Char representando a segunda frequência.
	g	Apontador para o grafo onde se procura as interseções.
	freq1	Char da primeira frequência.
	freq2	Char da segunda frequência.

```
00335                                                   {
00336     if (freq1 == freq2) {
00337         printf("As frequencias devem ser diferentes.\n");
00338         return;
00339     }
00340     printf("Intersecoes entre antenas com frequencias '%c' e '%c':\n",
00341            freq1, freq2);
00342
00343     for (Vertice *v = g->head; v; v = v->next) {
00344         if (v->freq == freq1) {
00345             for (Vertice *w = g->head; w; w = w->next) {
00346                 if (w->freq == freq2) {
00347                     printf("Antena [%c] em (%d,%d)  <->  Antena [%c] em (%d,%d)\n",
00348                            freq1, v->x, v->y,
00349                            freq2, w->x, w->y);
00350                 }
00351             }
00352         }
00353     }
00354 }
```

4.2.2.9 removerVertice()

```
bool removerVertice (
            Grafo * g,
            int id)
```

Remove um vértice do grafo, juntamente com todas as arestas incidentes.

Parâmetros

	g	Apontador para o grafo onde o vértice será removido.
	id	Identificador do vértice a ser removido.

Retorna

true Se o vértice existia e foi removido com sucesso.

false Se não existir vértice com esse ID.

Remove um vértice do grafo, juntamente com todas as arestas incidentes.

Parâmetros

	g	Apontador para o grafo onde está o vértice.
	id	Identificador do vértice a ser removido.

Retorna

true Se o vértice existia e foi removido com sucesso.

false Se não houver vértice com esse ID.

```
00365                                       {
00366     Vertice *alvo = getVerticePorID(g, id);
00367     if (!alvo) return false;
00368 
00369     for (Vertice *v = g->head; v; v = v->next) {
00370         if (v != alvo) {
00371             removerArestaDireta(v, alvo);
00372         }
00373     }
00374 
00375     Adj *a = alvo->lista;
00376     while (a) {
00377         Adj *aux = a->prox;
00378         free(a);
00379         a = aux;
00380     }
00381 
00382     Vertice *prev = NULL, *cur = g->head;
00383     while (cur) {
00384         if (cur == alvo) {
00385             if (prev) prev->next = cur->next;
00386             else      g->head = cur->next;
00387             free(cur);
00388             break;
00389         }
00390         prev = cur;
00391         cur  = cur->next;
00392     }
00393 
00394     g->n_vertices--;
00395 
00396     int contador = 0;
00397     for (Vertice *v = g->head; v; v = v->next) {
00398         v->id = contador++;
00399     }
00400 
00401     return true;
00402 }
```

4.3 grafo.h

Ir para a documentação deste ficheiro.

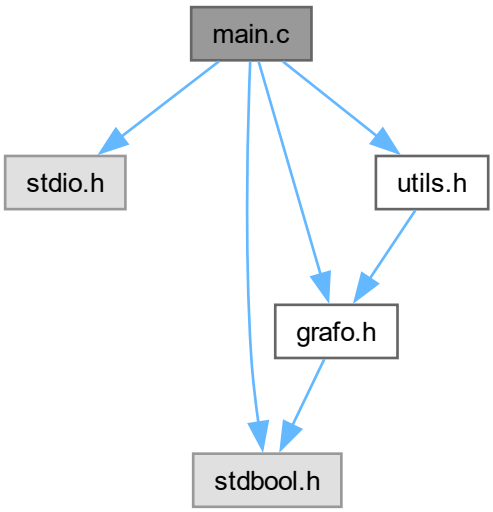
```
00001 
00011 
00012 #ifndef GRAFO_H
00013 #define GRAFO_H
00014 
00015 #include <stdbool.h>
00016 
00017 
00018 
00019 typedef struct Adj {
00020     Vertice *dest; // Apontador para o vértice de destino da aresta
00021     Adj *prox; // Apontador para o próximo nó na lista de adjacências
00022 }Adj;
00023 
00024 
00025 
00026 typedef struct Vertice {
00027     int id; // Identificador único do vértice
00028     int x, y; // Coordenadas da antena na matriz
00029     char freq; // Frequência/identificador da antena
00030     Adj *lista; // Cabeça da lista de adjacências (arestas)
00031     Vertice *next; // Apontador para o próximo vértice na lista de todos os vértices
```

```
00032
00033     bool visited; // Flag auxiliar para marcar vértices visitados em procuras (DFS/BFS)
00034 }Vertice;
00035
00036
00037 typedef struct {
00038     Vertice *head; // Apontador para o primeiro vértice da lista encadeada de vértices
00039     int n_vertices; // Contador do número de vértices atualmente no grafo
00040 } Grafo;
00041
00042
00048 void inicializaGrafo(Grafo *g);
00049
00059 int adicionaVertice(Grafo *g, int x, int y, char freq);
00060
00070 bool adicionaAresta(Grafo *g, int orig_id, int dest_id);
00071
00077 void imprimeGrafo(Grafo *g);
00078
00087 bool dfs(Grafo *g, int start_id);
00088
00097 bool bfs(Grafo *g, int start_id);
00098
00108 bool listarCaminhos(Grafo *g, int origem, int destino);
00109
00117 void listarIntersecoes(Grafo *g, char freq1, char freq2);
00118
00127 bool removerVertice(Grafo *g, int id);
00128
00129 #endif
```

4.4 Referência ao ficheiro main.c

```
#include <stdio.h>
#include <stdbool.h>
#include "grafo.h"
#include "utils.h"
```

Diagrama de dependências de inclusão para main.c:



Funções

- int main (void)

4.4.1 Descrição detalhada

Autor

Daniel Vilaça (a16939@alunos.ipca.pt)

Versão

1.0

Data

2025-05-31

Copyright

Copyright (c) 2025

4.4.2 Documentação das funções

4.4.2.1 main()

```
int main (
             void )
00045                {
00046     Grafo g;
00047     inicializaGrafo(&g);
00048
00049     int opcao;
00050     char filename[256];
00051
00052     do {
00053         menu();
00054         if (scanf("%d", &opcao) != 1) {
00055             printf("Leitura de opcao falhou.\n");
00056             return 1;
00057         }
00058
00059         switch (opcao) {
00060             case 1: {
00061                 printf("Nome do ficheiro (ex: matriz.txt): ");
00062                 scanf("%255s", filename);
00063                 inicializaGrafo(&g);
00064                 if (!carregarMatrizParaGrafo(filename, &g)) {
00065                     printf("Erro ao carregar matriz do ficheiro.\n");
00066                 } else {
00067                     printf("Matriz carregada com %d antenas.\n", g.n_vertices);
00068                 }
00069                 break;
00070             }
00071             case 2:
00072                 if (g.n_vertices == 0) {
00073                     printf("Grafo vazio.\n");
00074                 } else {
00075                     imprimeGrafo(&g);
00076                 }
00077                 break;
00078
00079             case 3: {
00080                 int id;
00081                 if (g.n_vertices == 0) {
00082                     printf("Grafo vazio.\n");
00083                     break;
00084                 }
00085                 printf("ID da antena de partida (0 a %d): ", g.n_vertices - 1);
00086                 scanf("%d", &id);
00087                 clearFlags(&g);
00088                 if (!dfs(&g, id)) {
```

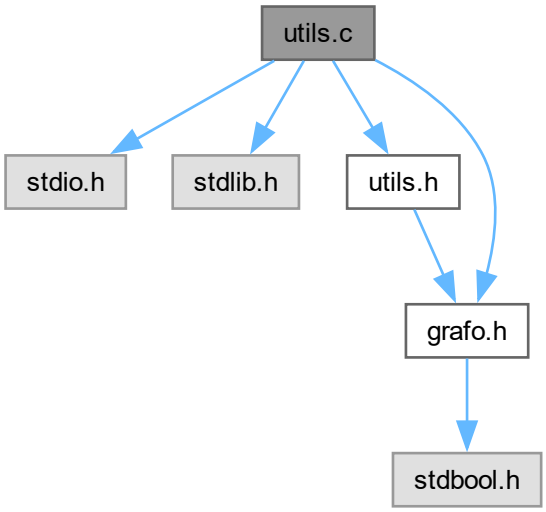
```
00089                     printf("Erro ao executar DFS. Verifique o ID introduzido.\n");
00090                 }
00091                 break;
00092             }
00093
00094             case 4: {
00095                 int id;
00096                 if (g.n_vertices == 0) {
00097                     printf("Grafo vazio.\n");
00098                     break;
00099                 }
00100                 printf("ID da antena de partida (0 a %d): ", g.n_vertices - 1);
00101                 scanf("%d", &id);
00102                 clearFlags(&g);
00103                 if (!bfs(&g, id)) {
00104                     printf("Erro ao executar BFS. Verifique o ID introduzido.\n");
00105                 }
00106                 break;
00107             }
00108
00109             case 5: {
00110                 int src, dst;
00111                 if (g.n_vertices < 2) {
00112                     printf("Nao ha vertices suficientes.\n");
00113                     break;
00114                 }
00115                 printf("ID da antena origem (0 a %d): ", g.n_vertices - 1);
00116                 scanf("%d", &src);
00117                 printf("ID da antena destino (0 a %d): ", g.n_vertices - 1);
00118                 scanf("%d", &dst);
00119                 clearFlags(&g);
00120                 if (!listarCaminhos(&g, src, dst)) {
00121                     printf("Nao foi possivel listar caminhos. IDs invalidos ou frequencias
    diferentes.\n");
00122                 }
00123                 break;
00124             }
00125
00126             case 6: {
00127                 char f1, f2;
00128                 if (g.n_vertices < 2) {
00129                     printf("Nao ha vertices suficientes.\n");
00130                     break;
00131                 }
00132                 printf("Introduza a primeira frequencia: ");
00133                 scanf(" %c", &f1);
00134                 printf("Introduza a segunda frequencia: ");
00135                 scanf(" %c", &f2);
00136                 clearFlags(&g);
00137                 listarIntersecoes(&g, f1, f2);
00138                 break;
00139             }
00140
00141             case 7: {
00142                 int x, y;
00143                 char freq;
00144                 printf("Introduza a frequencia (caracter): ");
00145                 scanf(" %c", &freq);
00146                 printf("Coordenadas (x y): ");
00147                 scanf("%d %d", &x, &y);
00148
00149                 int novo_id = adicionaVertice(&g, x, y, freq);
00150                 if (novo_id < 0) {
00151                     printf("Falha ao inserir antena.\n");
00152                 } else {
00153
00154                     for (Vertice *v = g.head; v; v = v->next) {
00155                         if (v->id != novo_id && v->freq == freq) {
00156                             adicionaAresta(&g, novo_id, v->id);
00157                         }
00158                     }
00159                     printf("Antena inserida com ID %d.\n", novo_id);
00160                 }
00161                 break;
00162             }
00163
00164             case 8: {
00165                 int id;
00166                 if (g.n_vertices == 0) {
00167                     printf("Grafo vazio.\n");
00168                     break;
00169                 }
00170                 printf("ID da antena a remover (0 a %d): ", g.n_vertices - 1);
00171                 scanf("%d", &id);
00172                 if (!removerVertice(&g, id)) {
00173                     printf("Antena não encontrada.\n");
00174                 } else {
```

```
00175                     printf("Antena removida.\n");
00176                 }
00177                 break;
00178             }
00179
00180             case 0:
00181                 printf("A sair...\n");
00182                 break;
00183
00184             default:
00185                 printf("Opcao invalida.\n");
00186         }
00187
00188     } while (opcao != 0);
00189
00190     return 0;
00191 }
```

4.5 Referência ao ficheiro utils.c

```
#include <stdio.h>
#include <stdlib.h>
#include "utils.h"
#include "grafo.h"
```

Diagrama de dependências de inclusão para utils.c:



Funções

- bool carregarMatrizParaGrafo (const char *filename, Grafo *g)

  Carrega uma "matriz" de antenas de um ficheiro e constrói o grafo.

4.5.1 Descrição detalhada

Autor

Daniel Vilaça (a16939@alunos.ipca.pt)

Versão

1.0

Data

2025-05-31

Copyright

Copyright (c) 2025

4.5.2 Documentação das funções

4.5.2.1 carregarMatrizParaGrafo()

```
bool carregarMatrizParaGrafo (
            const char * filename,
            Grafo * g)
```

Carrega uma “matriz” de antenas de um ficheiro e constrói o grafo.

Lê um ficheiro-texto contendo uma matriz de antenas e preenche o grafo.

Parâmetros

filename		Nome do ficheiro (.txt) que contém a matriz.
	g	Apontador para o grafo que será preenchido com vértices e arestas.

Retorna

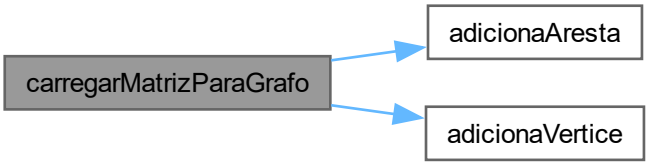
true Se o ficheiro foi aberto corretamente e o grafo foi construído.

false Se ocorrer erro ao abrir o ficheiro ou falha de alocação ao adicionar vértice.

```
00027                                                   {
00028     FILE *fp = fopen(filename, "r");
00029     if (!fp) {
00030         printf("Erro ao abrir o ficheiro.\n");
00031         return false;
00032     }
00033
00034     char linha[512];
00035     int y = 0;
00036
00037     while (fgets(linha, sizeof linha, fp)) {
00038         for (int x = 0; linha[x] != '\0' && linha[x] != '\n'; x++) {
00039             if (linha[x] != '.') {
00040                 if (adicionaVertice(g, x, y, linha[x]) < 0) {
00041                     fclose(fp);
00042                     return false;
00043                 }
00044             }
00045         }
```

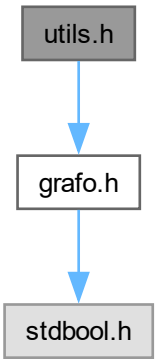
```
00046         y++;
00047     }
00048     fclose(fp);
00049
00050
00051     for (Vertice *v = g->head; v; v = v->next) {
00052         for (Vertice *w = v->next; w; w = w->next) {
00053             if (v->freq == w->freq) {
00054                 adicionaAresta(g, v->id, w->id);
00055             }
00056         }
00057     }
00058
00059     return true;
00060 }
```

Grafo de chamadas desta função:

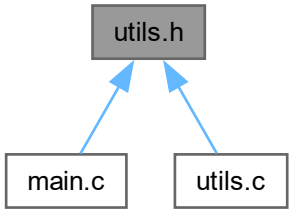


4.6 Referência ao ficheiro utils.h

`#include "grafo.h"`
Diagrama de dependências de inclusão para utils.h:



Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Funções

- bool carregarMatrizParaGrafo (const char ∗filename, Grafo ∗g)

  Lê um ficheiro-texto contendo uma matriz de antenas e preenche o grafo.

4.6.1 Descrição detalhada

Autor

Daniel Vilaça (a16939@alunos.ipca.pt)

Versão

1.0

Data

2025-05-31

Copyright

Copyright (c) 2025

4.6.2 Documentação das funções

4.6.2.1 carregarMatrizParaGrafo()

```
bool carregarMatrizParaGrafo (
            const char * filename,
            Grafo * g)
```

Lê um ficheiro-texto contendo uma matriz de antenas e preenche o grafo.

Parâmetros

filename		Nome do ficheiro (.txt) que contém a matriz de antenas.
	g	Apontador para o grafo que será preenchido com vértices e arestas.

Retorna

true Se o ficheiro for aberto e processado com sucesso.

false Se ocorrer erro ao abrir o ficheiro ou falha de alocação ao adicionar vértice.

Lê um ficheiro-texto contendo uma matriz de antenas e preenche o grafo.

Parâmetros

filename		Nome do ficheiro (.txt) que contém a matriz.
	g	Apontador para o grafo que será preenchido com vértices e arestas.

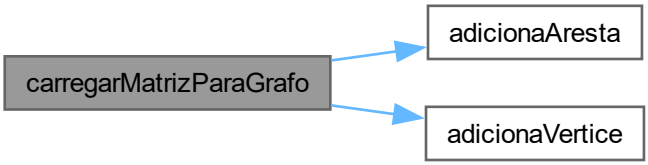
Retorna

true Se o ficheiro foi aberto corretamente e o grafo foi construído.

false Se ocorrer erro ao abrir o ficheiro ou falha de alocação ao adicionar vértice.

```
00027                                                                   {
00028     FILE *fp = fopen(filename, "r");
00029     if (!fp) {
00030         printf("Erro ao abrir o ficheiro.\n");
00031         return false;
00032     }
00033
00034     char linha[512];
00035     int y = 0;
00036
00037     while (fgets(linha, sizeof linha, fp)) {
00038         for (int x = 0; linha[x] != '\0' && linha[x] != '\n'; x++) {
00039             if (linha[x] != '.') {
00040                 if (adicionaVertice(g, x, y, linha[x]) < 0) {
00041                     fclose(fp);
00042                     return false;
00043                 }
00044             }
00045         }
00046         y++;
00047     }
00048     fclose(fp);
00049
00050
00051     for (Vertice *v = g->head; v; v = v->next) {
00052         for (Vertice *w = v->next; w; w = w->next) {
00053             if (v->freq == w->freq) {
00054                 adicionaAresta(g, v->id, w->id);
00055             }
00056         }
00057     }
00058
00059     return true;
00060 }
```

Grafo de chamadas desta função:



4.7 utils.h

Ir para a documentação deste ficheiro.

```
00001
00011
00012
00013 #ifndef UTILS_H
00014 #define UTILS_H
00015
00016 #include "grafo.h"
00017
00018
00027 bool carregarMatrizParaGrafo(const char *filename, Grafo *g);
00028
00029 #endif
```