Trabalho-Pratico_EDA

Gerado por Doxygen 1.10.0

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 Edge

Representa uma aresta (ligação) entre dois vértices no grafo.

`#include <func.h>`

Campos de Dados

- float **weight**
- struct Vertex * **dest**
- struct Edge * **next**

3.1.1 Descrição detalhada

Representa uma aresta (ligação) entre dois vértices no grafo.

Cada aresta possui um peso (distância ou custo), um ponteiro para o vértice de destino e um ponteiro para a próxima aresta na lista de adjacência.

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

- src/func.h

3.2 Referência à estrutura Element

Representa um elemento de uma lista auxiliar que contém vértices.

`#include <func.h>`

Campos de Dados

- Vertex * **item**
- struct Element * **next**

3.2.1 Descrição detalhada

Representa um elemento de uma lista auxiliar que contém vértices.

Utilizado para armazenar e organizar vértices durante percursos ou agrupamentos.

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

- src/func.h

3.3 Referência à estrutura Graph

Representa a estrutura principal de um grafo.

`#include <func.h>`

Campos de Dados

- int **count**
- Vertex * **vertices**

3.3.1 Descrição detalhada

Representa a estrutura principal de um grafo.

Contém o número total de vértices e um ponteiro para a lista de vértices.

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

- src/func.h

3.4 Referência à estrutura Node

Representa um nó de uma lista ligada de elementos na matriz.

`#include <func.h>`

Campos de Dados

- Vector2 **pos**

  Posição do nó na matriz.
- char **value**

  Valor do nó (ex: antena tipo A, efeito nefasto '#')
- struct Node * **next**

  Ponteiro para o próximo nó na lista.

3.4.1 Descrição detalhada

Representa um nó de uma lista ligada de elementos na matriz.

Cada nó contém um valor (por exemplo, um tipo de antena ou um efeito nefasto), uma posição na matriz representada por um Vector2, e um ponteiro para o próximo nó da lista.

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

- src/func.h

3.5 Referência à estrutura Path

Estrutura auxiliar utilizada para armazenar percursos ou listas ordenadas de vértices.

```
#include <func.h>
```

Campos de Dados

- Element * **first**
- int **max**

3.5.1 Descrição detalhada

Estrutura auxiliar utilizada para armazenar percursos ou listas ordenadas de vértices.

Guarda a lista de elementos visitados e um contador máximo de visitas, utilizado para BFS/DFS.

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

- src/func.h

3.6 Referência à estrutura Vector2

Representa um vetor bidimensional com coordenadas inteiras.

```
#include <func.h>
```

Campos de Dados

- int **x**

  Coordenada X.
- int **y**

  Coordenada Y.

3.6.1 Descrição detalhada

Representa um vetor bidimensional com coordenadas inteiras.

A estrutura Vector2 é usada para armazenar posições em uma matriz, com valores inteiros para coordenadas X e Y.

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

- src/func.h

3.7 Referência à estrutura Vertex

Representa um vértice (nó) no grafo.

```
#include <func.h>
```

Campos de Dados

- int **id**
- Vector2 **pos**
- char **value**
- Edge ∗ **edges**
- int **seen**
- struct Vertex ∗ **next**

3.7.1 Descrição detalhada

Representa um vértice (nó) no grafo.

Cada vértice possui um identificador único, uma posição na matriz (Vector2), um valor (carácter representando o tipo, como 'A' ou 'B'), uma lista de arestas (ligações), uma flag de controlo de visitação (seen) e um ponteiro para o próximo vértice.

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

- src/func.h

Capítulo 4

Documentação do ficheiro

4.1 Referência ao ficheiro src/func.c

Implementação de funções para manipulação de uma lista de elementos representados em matriz.

```
#include <stdio.h>
#include <stdlib.h>
#include "func.h"
#include "interface.h"
#include <malloc.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>
```

Funções

- Node ∗ MakeNode (char value, Vector2 position)

  Cria um novo elemento (nó) para a lista.
- bool NodeInBounds (Vector2 pos)

  Verifica se a posição está dentro dos limites da matriz.
- Node ∗ InsertNode (Node ∗dnew, Node ∗st)

  Insere um novo nó na lista, verificando se a posição está disponível.
- Node ∗ RemoveNode (Node ∗rm, Node ∗st)

  Remove um nó da lista.
- Node ∗ ClearNoise (Node ∗st)

  Limpa elementos de ruído ('#') da lista.
- bool ValidNodePos (Node ∗dnew, Node ∗st)

  Verifica se a posição de um novo nó é válida (sem sobreposição).
- Node ∗ NoiseCheck (Node ∗st)

  Verifica e aplica a regra de efeito nefasto aos elementos na lista.
- Node ∗ NoiseCheckAlt (Node ∗st)

  Versão alternativa da verificação de ruído que compara todos os elementos entre si.
- Node ∗ FindNodePos (Node ∗st, Vector2 npos)

  Procura um nó na lista pela sua posição.
- Node ∗ FreeNodes (Node ∗st)

Procura um nó em uma posição específica.

- Graph * InitGraph ()

  Inicializa uma estrutura de grafo vazia.

- Vertex * MakeVertex (char value, Vector2 position, int *id)

  Cria um novo vértice com valor e posição especificados.

- bool InsertEdge (Vertex *from, Vertex *dest)

  Cria uma nova aresta entre dois vértices.

- bool IsNewEdge (Vertex *from, Vertex *dest)

  Verifica se uma ligação entre dois vértices já existe.

- Vertex * InsertVertex (Vertex *dnew, Vertex *st, bool edge)

  Insere um vértice na lista de vértices.

- Vertex * RemoveVertex (Vertex *rm, Vertex *st)

  Remove um vértice e as suas arestas da lista.

- bool ClearEdges (Vertex *old)

  Liberta todas as arestas de um vértice.

- bool RemoveEdge (Vertex *from, Vertex *dest)

  Remove uma aresta entre dois vértices.

- bool EdgeFindDest (Edge **current, Edge **previous, Vertex *pick)

  Encontra uma aresta de destino e os ponteiros anteriores na lista.

- bool AddEdges (Vertex *new, Vertex *st)

  Adiciona arestas entre um novo vértice e outros na lista, se forem compatíveis.

- Vertex * FindVertexAt (Vertex *st, Vector2 npos)

  Procura um vértice no grafo a partir de uma posição.

- Vertex * FindVertexById (Vertex *vertices, int id)

  Procura um vértice no grafo com base no seu identificador (ID).

- bool FreeGraph (Graph *gr)

  Liberta todos os vértices e arestas de um grafo.

- bool ClearSeen (Vertex *st)

  Limpa o estado de "visitado" (seen) de todos os vértices.

- Element * GraphDFS (Graph *gr, Vertex *start)

  Realiza uma pesquisa em profundidade (DFS) a partir de um vértice inicial.

- void DFSMark (Vertex *current, int *count)

  Marca recursivamente os vértices visitados em profundidade (DFS).

- Element * GraphBFS (Graph *gr, Vertex *start)

  Realiza uma pesquisa em largura (BFS) a partir de um vértice inicial.

- Element * MakeSeenList (Graph *gr, int max)

  Cria uma lista ordenada de elementos com base na ordem de visita (seen).

- Element * GraphPaths (Graph *gr, Vertex *start, Vertex *end)

  Calcula todos os caminhos possíveis entre dois vértices.

- void Pathing (Vertex *current, Vertex *end, Path *path, Graph *gr)

  Caminha recursivamente no grafo entre dois vértices, acumulando o caminho atual.

- void AddPath (Path *path, Graph *gr, int max)

  Adiciona um caminho encontrado à lista principal.

- Element * FindPairs (Graph *gr, char a, char b)

  Encontra e retorna uma lista de intersecções entre pares de antenas com ressonância A e B.

- Element * MakeElement (Vertex *value)

  Cria um novo elemento com base num vértice.

- Element * InsertElement (Element *dnew, Element *st)

  Insere um elemento no início de uma lista.

- Element * InsertElementAtEnd (Element *new, Element *st)

  Insere um elemento no final da lista.

- Element * RemoveElement (Element *rm, Element *st)

  Remove um elemento específico da lista.
- Element * FindElement (void *item, Element *st)

  Procura um elemento na lista.
- Element * FreeElements (Element *path)

  Liberta toda a memória alocada por uma lista de elementos.
- bool Vector2Compare (Vector2 a, Vector2 b)

  Compara se dois vetores possuem as mesmas coordenadas.
- Vector2 Vector2Subtract (Vector2 a, Vector2 b)

  Subtrai um vetor do outro.
- Vector2 Vector2Add (Vector2 a, Vector2 b)

  Soma dois vetores.
- bool ReadGraphFile (const char *filename, Graph *gr)

  Lê um grafo a partir de um ficheiro binário, incluindo os vértices e as arestas com base nas posições.
- bool SaveGraphFile (const char *filename, Graph *gr)

  Guarda um grafo num ficheiro binário, incluindo vértices e arestas por posição (Vector2).
- Node * ReadListFile (const char *filename, Node *st)

  Lê uma lista de elementos a partir de um ficheiro.
- bool SaveList (const char *filename, Node *st)

  Guarda a lista de elementos num ficheiro.
- bool CopyListToGraph (Node *st, Graph *gr)

  Copia os nós de uma lista para um grafo.
- Node * CopyGraphToList (Graph *gr, Node *st)

  Copia os vértices de um grafo para uma lista de nós.

Variáveis

- Vector2 **sSize** = {SWIDTH, SHEIGHT}
- int **noiseRange** = NOISERANGE

4.1.1 Descrição detalhada

Implementação de funções para manipulação de uma lista de elementos representados em matriz.

Autor

Vitor Rezende (a31521@alunos.ipca.pt)

Versão

0.10

Data

2025-03-18

Copyright

4.1.2 Documentação das funções

4.1.2.1 AddEdges()

```
bool AddEdges (
            Vertex * new,
            Vertex * st )
```

Adiciona arestas entre um novo vértice e outros na lista, se forem compatíveis.

Compatibilidade geralmente depende do valor/resonância da antena.

Parâmetros

new	Novo vértice.
st	Lista de vértices existentes.

Retorna

true Se pelo menos uma aresta foi adicionada.

4.1.2.2 AddPath()

```
void AddPath (
            Path * head,
            Graph * gr,
            int max )
```

Adiciona um caminho encontrado à lista principal.

Parâmetros

head	Lista principal de caminhos.
gr	Grafo de onde os caminhos provêm.
max	Número máximo de visitas.

4.1.2.3 ClearEdges()

```
bool ClearEdges (
            Vertex * old )
```

Liberta todas as arestas de um vértice.

Parâmetros

old	Vértice cujas arestas devem ser removidas.

Retorna

true Se as arestas foram removidas com sucesso.

4.1.2.4 ClearNoise()

```
Node * ClearNoise (
            Node * st )
```

Limpa elementos de ruído ('#') da lista.

Parâmetros

st	Ponteiro para a lista.

Retorna

Ponteiro para a lista sem elementos de ruído.

4.1.2.5 ClearSeen()

```
bool ClearSeen (
            Vertex * st )
```

Limpa o estado de "visitado" (seen) de todos os vértices.

Define seen = 0 para todos os vértices, preparando para uma nova pesquisa.

Parâmetros

st	Lista de vértices.

Retorna

true Se os valores foram reiniciados com sucesso.

4.1.2.6 CopyGraphToList()

```
Node * CopyGraphToList (
            Graph * gr,
            Node * st )
```

Copia os vértices de um grafo para uma lista de nós.

Cria uma nova lista de elementos baseada na posição e tipo dos vértices do grafo.

Parâmetros

gr	Grafo de origem.
st	Lista de destino, será atualizada.

Retorna

Node∗ Ponteiro para a nova lista resultante.

4.1.2.7 CopyListToGraph()

```
bool CopyListToGraph (
            Node * st,
            Graph * gr )
```

Copia os nós de uma lista para um grafo.

Converte cada nó da lista de elementos (Node) para um vértice no grafo, mantendo as posições e valores. Atribui automaticamente arestas entre os vértices conforme necessário.

Parâmetros

st	Lista de nós a converter.
gr	Grafo onde os vértices serão inseridos.

Retorna

true Se a operação foi concluída com sucesso.

4.1.2.8 DFSMark()

```
void DFSMark (
            Vertex * current,
            int * count )
```

Marca recursivamente os vértices visitados em profundidade (DFS).

Utilizado para contagem ou verificação de conectividade.

Parâmetros

current	Vértice atual.
count	Ponteiro para o contador de visitas.

4.1.2.9 EdgeFindDest()

```
bool EdgeFindDest (
            Edge ** current,
```

```
            Edge ** previous,
            Vertex * pick )
```

Encontra uma aresta de destino e os ponteiros anteriores na lista.

Auxiliar para remoção de arestas.

Parâmetros

current	Ponteiro para a aresta atual.
previous	Ponteiro para a aresta anterior.
pick	Vértice de destino a encontrar.

Retorna

true Se a aresta foi encontrada.

4.1.2.10 FindElement()

```
Element * FindElement (
            void * item,
            Element * st )
```

Procura um elemento na lista.

Parâmetros

item	Ponteiro para o item a procurar.
st	Lista onde será feita a procura.

Retorna

Element∗ Ponteiro para o elemento encontrado ou NULL.

4.1.2.11 FindNodePos()

```
Node * FindNodePos (
            Node * st,
            Vector2 npos )
```

Procura um nó na lista pela sua posição.

Procura um nó numa posição específica da matriz.

Parâmetros

st	Ponteiro para a lista.
npos	Posição do nó a ser encontrado.

Retorna

Ponteiro para o nó encontrado, ou NULL se não existir.

4.1.2.12 FindPairs()

```
Element * FindPairs (
            Graph * gr,
            char a,
            char b )
```

Encontra e retorna uma lista de intersecções entre pares de antenas com ressonância A e B.

Percorre o grafo e encontra todos os pares de antenas (vértices) em que uma possui valor/resonância A e a outra B. Apenas regista intersecções diretas (arestas existentes). A lista resultante contém os pares encontrados.

Parâmetros

gr	Ponteiro para o grafo.
a	Carácter correspondente à primeira frequência (ex: 'A').
b	Carácter correspondente à segunda frequência (ex: 'B').

Retorna

Element∗ Lista de elementos contendo os pares encontrados.

4.1.2.13 FindVertexAt()

```
Vertex * FindVertexAt (
            Vertex * st,
            Vector2 npos )
```

Procura um vértice no grafo a partir de uma posição.

Parâmetros

st	Lista de vértices do grafo.
npos	Posição procurada.

Retorna

Vertex∗ Ponteiro para o vértice encontrado, ou NULL se não existir.

4.1.2.14 FindVertexById()

```
Vertex * FindVertexById (
            Vertex * vertices,
            int id )
```

Procura um vértice no grafo com base no seu identificador (ID).

Percorre a lista de vértices e retorna o ponteiro correspondente ao ID.

Parâmetros

vertices	Lista de vértices do grafo.
id	Identificador único do vértice a encontrar.

Retorna

Vertex∗ Ponteiro para o vértice correspondente, ou NULL se não encontrado.

4.1.2.15 FreeElements()

```
Element * FreeElements (
            Element * head )
```

Liberta toda a memória alocada por uma lista de elementos.

Parâmetros

head	Início da lista.

Retorna

Element∗ NULL, lista vazia.

4.1.2.16 FreeGraph()

```
bool FreeGraph (
            Graph * gr )
```

Liberta todos os vértices e arestas de um grafo.

Remove completamente todos os dados da estrutura de grafo e reinicia os contadores.

Parâmetros

gr	Ponteiro para o grafo a ser limpo.

Retorna

true Se a operação foi concluída com sucesso.

4.1.2.17 FreeNodes()

```
Node * FreeNodes (
```

```
            Node * head )
```

Procura um nó em uma posição específica.

Parâmetros

st	Lista de nós.
npos	Posição a ser pesquisada.

Retorna

Node∗ Ponteiro para o nó encontrado ou NULL se não existir.

4.1.2.18 GraphBFS()

```
Element * GraphBFS (
            Graph * gr,
            Vertex * start )
```

Realiza uma pesquisa em largura (BFS) a partir de um vértice inicial.

Visita todos os vértices acessíveis por níveis, e armazena os visitados numa lista.

Parâmetros

gr	Grafo onde a pesquisa será realizada.
start	Vértice inicial da pesquisa.

Retorna

Element∗ Lista de elementos visitados pela BFS.

4.1.2.19 GraphDFS()

```
Element * GraphDFS (
            Graph * gr,
            Vertex * start )
```

Realiza uma pesquisa em profundidade (DFS) a partir de um vértice inicial.

Visita todos os vértices conectados recursivamente e guarda a ordem numa lista.

Parâmetros

gr	Grafo onde a pesquisa será realizada.
start	Vértice inicial da pesquisa.

Retorna

Element* Lista de elementos visitados pela DFS.

4.1.2.20 GraphPaths()

```
Element * GraphPaths (
            Graph * gr,
            Vertex * start,
            Vertex * end )
```

Calcula todos os caminhos possíveis entre dois vértices.

Usa backtracking para encontrar todas as combinações possíveis.

Parâmetros

gr	Ponteiro para o grafo.
start	Vértice de início.
end	Vértice de destino.

Retorna

Element* Lista de caminhos encontrados.

4.1.2.21 InitGraph()

```
Graph * InitGraph ( )
```

Inicializa uma estrutura de grafo vazia.

Reserva memória para um novo grafo e inicializa o contador e a lista de vértices.

Retorna

Graph* Ponteiro para o novo grafo.

4.1.2.22 InsertEdge()

```
bool InsertEdge (
            Vertex * from,
            Vertex * dest )
```

Cria uma nova aresta entre dois vértices.

Apenas adiciona se a ligação ainda não existir.

Parâmetros

from	Vértice de origem.
dest	Vértice de destino.

Retorna

true Se a aresta foi criada com sucesso.

false Se a ligação já existia.

4.1.2.23 InsertElement()

```
Element * InsertElement (
            Element * dnew,
            Element * st )
```

Insere um elemento no início de uma lista.

Parâmetros

dnew	Novo elemento.
st	Lista existente.

Retorna

Element∗ Lista atualizada.

4.1.2.24 InsertElementAtEnd()

```
Element * InsertElementAtEnd (
            Element * new,
            Element * st )
```

Insere um elemento no final da lista.

Parâmetros

new	Novo elemento.
st	Lista atual.

Retorna

Element∗ Lista atualizada.

4.1.2.25 InsertNode()

```
Node * InsertNode (
```

```
            Node * dnew,
            Node * st )
```

Insere um novo nó na lista, verificando se a posição está disponível.

Parâmetros

dnew	Novo nó a ser inserido.
st	Ponteiro para a lista.

Retorna

Ponteiro para a lista atualizada.

4.1.2.26 InsertVertex()

```
Vertex * InsertVertex (
            Vertex * dnew,
            Vertex * st,
            bool edge )
```

Insere um vértice na lista de vértices.

Parâmetros

dnew	Novo vértice.
st	Lista existente de vértices.
edge	Define se deve adicionar arestas automaticamente para vértices compatíveis.

Retorna

Vertex∗ Lista atualizada.

4.1.2.27 IsNewEdge()

```
bool IsNewEdge (
            Vertex * from,
            Vertex * dest )
```

Verifica se uma ligação entre dois vértices já existe.

Parâmetros

from	Vértice de origem.
dest	Vértice de destino.

Retorna

true Se já existe uma aresta entre os vértices.

false Caso contrário.

4.1.2.28 MakeElement()

```
Element * MakeElement (
            Vertex * value )
```

Cria um novo elemento com base num vértice.

Parâmetros

value	Ponteiro para o vértice.

Retorna

Element∗ Novo elemento criado.

4.1.2.29 MakeNode()

```
Node * MakeNode (
            char value,
            Vector2 position )
```

Cria um novo elemento (nó) para a lista.

Parâmetros

value	Valor do elemento (ex: antena tipo A, efeito nefasto '#').
position	Posição na matriz.

Retorna

Ponteiro para o novo nó criado.

4.1.2.30 MakeSeenList()

```
Element * MakeSeenList (
            Graph * gr,
            int max )
```

Cria uma lista ordenada de elementos com base na ordem de visita (seen).

Parâmetros

gr	Grafo com os vértices já marcados.
max	Valor máximo da marcação seen.

Retorna

Element* Lista ordenada de vértices pela ordem de visita.

4.1.2.31 MakeVertex()

```
Vertex * MakeVertex (
            char value,
            Vector2 position,
            int * id )
```

Cria um novo vértice com valor e posição especificados.

Gera um identificador único com base no ponteiro de contador fornecido.

Parâmetros

value	Valor (tipo) do vértice (ex: 'A', 'B', '#').
position	Posição no espaço bidimensional.
id	Ponteiro para o contador global de identificadores.

Retorna

Vertex* Novo vértice criado.

4.1.2.32 NodeInBounds()

```
bool NodeInBounds (
            Vector2 pos )
```

Verifica se a posição está dentro dos limites da matriz.

Parâmetros

pos	Posição a ser verificada.

Retorna

true se estiver dentro dos limites, false caso contrário.

4.1.2.33 NoiseCheck()

```
Node * NoiseCheck (
            Node * st )
```

Verifica e aplica a regra de efeito nefasto aos elementos na lista.

Parâmetros

st	Lista de nós.

Retorna

Node∗ Lista atualizada após a verificação do efeito nefasto.

4.1.2.34 NoiseCheckAlt()

```
Node * NoiseCheckAlt (
            Node * st )
```

Versão alternativa da verificação de ruído que compara todos os elementos entre si.

Parâmetros

st	Lista de nós.

Retorna

Node∗ Lista atualizada após a verificação do efeito nefasto.

4.1.2.35 Pathing()

```
void Pathing (
            Vertex * current,
            Vertex * end,
            Path * head,
            Graph * gr )
```

Caminha recursivamente no grafo entre dois vértices, acumulando o caminho atual.

Parâmetros

current	Vértice atual.
end	Vértice de destino.
head	Caminho acumulado.
gr	Ponteiro para o grafo.

4.1.2.36 ReadGraphFile()

```
bool ReadGraphFile (
            const char * filename,
            Graph * gr )
```

Lê um grafo a partir de um ficheiro binário, incluindo os vértices e as arestas com base nas posições.

Os vértices são lidos primeiro, seguidos de um marcador (-1), e depois as arestas que ligam posições (Vector2).

Parâmetros

filename	Nome do ficheiro a ler.
gr	Ponteiro para o grafo a ser preenchido.

Retorna

true Se a leitura for bem-sucedida.

false Em caso de erro na leitura do ficheiro.

4.1.2.37 ReadListFile()

```
Node * ReadListFile (
            const char * filename,
            Node * st )
```

Lê uma lista de elementos a partir de um ficheiro.

Parâmetros

filename	Nome do ficheiro.

Retorna

Node∗ Lista carregada do ficheiro.

4.1.2.38 RemoveEdge()

```
bool RemoveEdge (
            Vertex * from,
            Vertex * dest )
```

Remove uma aresta entre dois vértices.

Parâmetros

from	Vértice de origem.
dest	Vértice de destino.

Retorna

true Se a aresta foi removida com sucesso.

4.1.2.39 RemoveElement()

```
Element * RemoveElement (
            Element * rm,
            Element * st )
```

Remove um elemento específico da lista.

Parâmetros

rm	Elemento a remover.
st	Lista original.

Retorna

Element∗ Lista atualizada.

4.1.2.40 RemoveNode()

```
Node * RemoveNode (
            Node * rm,
            Node * st )
```

Remove um nó da lista.

Parâmetros

rm	Ponteiro para o nó a ser removido.
st	Ponteiro para a lista.

Retorna

Ponteiro para a lista atualizada.

4.1.2.41 RemoveVertex()

```
Vertex * RemoveVertex (
            Vertex * rm,
            Vertex * st )
```

Remove um vértice e as suas arestas da lista.

Parâmetros

rm	Vértice a remover.
st	Lista de vértices.

Retorna

Vertex∗ Lista atualizada.

4.1.2.42 SaveGraphFile()

```
bool SaveGraphFile (
```

```
            const char * filename,
            Graph * gr )
```

Guarda um grafo num ficheiro binário, incluindo vértices e arestas por posição (Vector2).

Os vértices são escritos primeiro, seguidos de um marcador (-1), e depois as arestas como pares de posições com peso.

Parâmetros

filename	Nome do ficheiro onde será guardado.
gr	Ponteiro para o grafo a ser guardado.

Retorna

true Se a gravação for bem-sucedida.

false Em caso de erro ao abrir ou escrever no ficheiro.

4.1.2.43 SaveList()

```
bool SaveList (
            const char * filename,
            Node * st )
```

Guarda a lista de elementos num ficheiro.

Parâmetros

filename	Nome do ficheiro onde os dados serão armazenados.
st	Lista de nós a ser salva.

4.1.2.44 ValidNodePos()

```
bool ValidNodePos (
            Node * dnew,
            Node * st )
```

Verifica se a posição de um novo nó é válida (sem sobreposição).

Parâmetros

dnew	Novo nó.
st	Ponteiro para a lista.

Retorna

true se a posição for válida, false caso contrário.

4.1.2.45 Vector2Add()

```
Vector2 Vector2Add (
            Vector2 a,
            Vector2 b )
```

Soma dois vetores.

Parâmetros

a	Primeiro vetor.
b	Segundo vetor.

Retorna

Vector2 Resultado da soma.

4.1.2.46 Vector2Compare()

```
bool Vector2Compare (
            Vector2 a,
            Vector2 b )
```

Compara se dois vetores possuem as mesmas coordenadas.

Parâmetros

a	Primeiro vetor.
b	Segundo vetor.

Retorna

true Se forem iguais.

false Se forem diferentes.

4.1.2.47 Vector2Subtract()

```
Vector2 Vector2Subtract (
            Vector2 a,
            Vector2 b )
```

Subtrai um vetor do outro.

Parâmetros

a	Primeiro vetor.
b	Segundo vetor.

Retorna

Vector2 Resultado da subtração.

4.2 Referência ao ficheiro src/func.h

Implementação de funções para manipulação de uma lista de elementos representados em matriz.

```
#include <stdbool.h>
```

Estruturas de Dados

- struct Vector2

  Representa um vetor bidimensional com coordenadas inteiras.
- struct Node

  Representa um nó de uma lista ligada de elementos na matriz.
- struct Edge

  Representa uma aresta (ligação) entre dois vértices no grafo.
- struct Vertex

  Representa um vértice (nó) no grafo.
- struct Graph

  Representa a estrutura principal de um grafo.
- struct Element

  Representa um elemento de uma lista auxiliar que contém vértices.
- struct Path

  Estrutura auxiliar utilizada para armazenar percursos ou listas ordenadas de vértices.

Macros

- #define **SWIDTH** 12
- #define **SHEIGHT** 12
- #define **NOISERANGE** 4
- #define **MAXINPUT** 100

Definições de tipos

- typedef struct Vector2 Vector2

  Representa um vetor bidimensional com coordenadas inteiras.
- typedef struct Node Node

  Representa um nó de uma lista ligada de elementos na matriz.
- typedef struct Edge Edge

  Representa uma aresta (ligação) entre dois vértices no grafo.
- typedef struct Vertex Vertex

  Representa um vértice (nó) no grafo.
- typedef struct Graph Graph

  Representa a estrutura principal de um grafo.
- typedef struct Element Element

  Representa um elemento de uma lista auxiliar que contém vértices.
- typedef struct Path Path

  Estrutura auxiliar utilizada para armazenar percursos ou listas ordenadas de vértices.

Funções

- Node ∗ MakeNode (char value, Vector2 position)

  Cria um novo elemento (nó) para a lista.
- bool NodeInBounds (Vector2 pos)

  Verifica se a posição está dentro dos limites da matriz.
- Node ∗ InsertNode (Node ∗st, Node ∗dnew)

  Insere um novo nó na lista, verificando se a posição está disponível.
- Node ∗ RemoveNode (Node ∗rm, Node ∗st)

  Remove um nó da lista.
- Node ∗ ClearNoise (Node ∗st)

  Limpa elementos de ruído ('#') da lista.
- bool ValidNodePos (Node ∗dnew, Node ∗st)

  Verifica se a posição de um novo nó é válida (sem sobreposição).
- Node ∗ NoiseCheck (Node ∗st)

  Verifica e aplica a regra de efeito nefasto aos elementos na lista.
- Node ∗ NoiseCheckAlt (Node ∗st)

  Versão alternativa da verificação de ruído que compara todos os elementos entre si.
- Node ∗ FreeNodes (Node ∗head)

  Procura um nó em uma posição específica.
- Node ∗ FindNodePos (Node ∗st, Vector2 npos)

  Procura um nó numa posição específica da matriz.
- Graph ∗ InitGraph ()

  Inicializa uma estrutura de grafo vazia.
- Vertex ∗ MakeVertex (char value, Vector2 position, int ∗id)

  Cria um novo vértice com valor e posição especificados.
- bool InsertEdge (Vertex ∗from, Vertex ∗dest)

  Cria uma nova aresta entre dois vértices.
- bool IsNewEdge (Vertex ∗from, Vertex ∗dest)

  Verifica se uma ligação entre dois vértices já existe.
- Vertex ∗ InsertVertex (Vertex ∗dnew, Vertex ∗st, bool edge)

  Insere um vértice na lista de vértices.
- Vertex ∗ RemoveVertex (Vertex ∗rm, Vertex ∗st)

  Remove um vértice e as suas arestas da lista.
- bool ClearEdges (Vertex ∗old)

  Liberta todas as arestas de um vértice.
- bool RemoveEdge (Vertex ∗from, Vertex ∗dest)

  Remove uma aresta entre dois vértices.
- bool EdgeFindDest (Edge ∗∗current, Edge ∗∗previous, Vertex ∗pick)

  Encontra uma aresta de destino e os ponteiros anteriores na lista.
- bool AddEdges (Vertex ∗new, Vertex ∗st)

  Adiciona arestas entre um novo vértice e outros na lista, se forem compatíveis.
- Vertex ∗ FindVertexAt (Vertex ∗st, Vector2 npos)

  Procura um vértice no grafo a partir de uma posição.
- Vertex ∗ FindVertexById (Vertex ∗vertices, int id)

  Procura um vértice no grafo com base no seu identificador (ID).
- bool FreeGraph (Graph ∗gr)

  Liberta todos os vértices e arestas de um grafo.
- bool ClearSeen (Vertex ∗st)

  Limpa o estado de "visitado" (seen) de todos os vértices.
- Element ∗ GraphDFS (Graph ∗gr, Vertex ∗start)

Realiza uma pesquisa em profundidade (DFS) a partir de um vértice inicial.

- void DFSMark (Vertex ∗current, int ∗count)

  Marca recursivamente os vértices visitados em profundidade (DFS).
- Element ∗ GraphBFS (Graph ∗gr, Vertex ∗start)

  Realiza uma pesquisa em largura (BFS) a partir de um vértice inicial.
- Element ∗ MakeSeenList (Graph ∗gr, int max)

  Cria uma lista ordenada de elementos com base na ordem de visita (seen).
- Element ∗ GraphPaths (Graph ∗gr, Vertex ∗start, Vertex ∗end)

  Calcula todos os caminhos possíveis entre dois vértices.
- void Pathing (Vertex ∗current, Vertex ∗end, Path ∗head, Graph ∗gr)

  Caminha recursivamente no grafo entre dois vértices, acumulando o caminho atual.
- void AddPath (Path ∗head, Graph ∗gr, int max)

  Adiciona um caminho encontrado à lista principal.
- Element ∗ FindPairs (Graph ∗gr, char a, char b)

  Encontra e retorna uma lista de intersecções entre pares de antenas com ressonância A e B.
- Element ∗ MakeElement (Vertex ∗value)

  Cria um novo elemento com base num vértice.
- Element ∗ InsertElement (Element ∗dnew, Element ∗st)

  Insere um elemento no início de uma lista.
- Element ∗ InsertElementAtEnd (Element ∗new, Element ∗st)

  Insere um elemento no final da lista.
- Element ∗ RemoveElement (Element ∗rm, Element ∗st)

  Remove um elemento específico da lista.
- Element ∗ FindElement (void ∗item, Element ∗st)

  Procura um elemento na lista.
- Element ∗ FreeElements (Element ∗head)

  Liberta toda a memória alocada por uma lista de elementos.
- bool Vector2Compare (Vector2 a, Vector2 b)

  Compara se dois vetores possuem as mesmas coordenadas.
- Vector2 Vector2Subtract (Vector2 a, Vector2 b)

  Subtrai um vetor do outro.
- Vector2 Vector2Add (Vector2 a, Vector2 b)

  Soma dois vetores.
- bool ReadGraphFile (const char ∗filename, Graph ∗gr)

  Lê um grafo a partir de um ficheiro binário, incluindo os vértices e as arestas com base nas posições.
- bool SaveGraphFile (const char ∗filename, Graph ∗gr)

  Guarda um grafo num ficheiro binário, incluindo vértices e arestas por posição (Vector2).
- Node ∗ ReadListFile (const char ∗filename, Node ∗st)

  Lê uma lista de elementos a partir de um ficheiro.
- bool SaveList (const char ∗filename, Node ∗st)

  Guarda a lista de elementos num ficheiro.
- bool CopyListToGraph (Node ∗st, Graph ∗gr)

  Copia os nós de uma lista para um grafo.
- Node ∗ CopyGraphToList (Graph ∗gr, Node ∗st)

  Copia os vértices de um grafo para uma lista de nós.

Variáveis

- Vector2 **sSize**
- int **noiseRange**

4.2.1 Descrição detalhada

Implementação de funções para manipulação de uma lista de elementos representados em matriz.

Autor

Vitor Rezende (a31521@alunos.ipca.pt)

Versão

0.10

Data

2025-03-18

Copyright

4.2.2 Documentação dos tipos

4.2.2.1 Edge

```
typedef struct Edge Edge
```

Representa uma aresta (ligação) entre dois vértices no grafo.

Cada aresta possui um peso (distância ou custo), um ponteiro para o vértice de destino e um ponteiro para a próxima aresta na lista de adjacência.

4.2.2.2 Element

```
typedef struct Element Element
```

Representa um elemento de uma lista auxiliar que contém vértices.

Utilizado para armazenar e organizar vértices durante percursos ou agrupamentos.

4.2.2.3 Graph

```
typedef struct Graph Graph
```

Representa a estrutura principal de um grafo.

Contém o número total de vértices e um ponteiro para a lista de vértices.

4.2.2.4 Node

```
typedef struct Node Node
```

Representa um nó de uma lista ligada de elementos na matriz.

Cada nó contém um valor (por exemplo, um tipo de antena ou um efeito nefasto), uma posição na matriz representada por um Vector2, e um ponteiro para o próximo nó da lista.

4.2.2.5 Path

```
typedef struct Path Path
```

Estrutura auxiliar utilizada para armazenar percursos ou listas ordenadas de vértices.

Guarda a lista de elementos visitados e um contador máximo de visitas, utilizado para BFS/DFS.

4.2.2.6 Vector2

```
typedef struct Vector2 Vector2
```

Representa um vetor bidimensional com coordenadas inteiras.

A estrutura Vector2 é usada para armazenar posições em uma matriz, com valores inteiros para coordenadas X e Y.

4.2.2.7 Vertex

```
typedef struct Vertex Vertex
```

Representa um vértice (nó) no grafo.

Cada vértice possui um identificador único, uma posição na matriz (Vector2), um valor (carácter representando o tipo, como 'A' ou 'B'), uma lista de arestas (ligações), uma flag de controlo de visitação (seen) e um ponteiro para o próximo vértice.

4.2.3 Documentação das funções

4.2.3.1 AddEdges()

```
bool AddEdges (
            Vertex * new,
            Vertex * st )
```

Adiciona arestas entre um novo vértice e outros na lista, se forem compatíveis.

Compatibilidade geralmente depende do valor/resonância da antena.

Parâmetros

new	Novo vértice.
st	Lista de vértices existentes.

Retorna

true Se pelo menos uma aresta foi adicionada.

4.2.3.2 AddPath()

```
void AddPath (
            Path * head,
            Graph * gr,
            int max )
```

Adiciona um caminho encontrado à lista principal.

Parâmetros

head	Lista principal de caminhos.
gr	Grafo de onde os caminhos provêm.
max	Número máximo de visitas.

4.2.3.3 ClearEdges()

```
bool ClearEdges (
            Vertex * old )
```

Liberta todas as arestas de um vértice.

Parâmetros

old	Vértice cujas arestas devem ser removidas.

Retorna

true Se as arestas foram removidas com sucesso.

4.2.3.4 ClearNoise()

```
Node * ClearNoise (
            Node * st )
```

Limpa elementos de ruído ('#') da lista.

Parâmetros

st	Ponteiro para a lista.

Retorna

Ponteiro para a lista sem elementos de ruído.

4.2.3.5 ClearSeen()

```
bool ClearSeen (
            Vertex * st )
```

Limpa o estado de "visitado" (seen) de todos os vértices.

Define seen = 0 para todos os vértices, preparando para uma nova pesquisa.

Parâmetros

st	Lista de vértices.

Retorna

true Se os valores foram reiniciados com sucesso.

4.2.3.6 CopyGraphToList()

```
Node * CopyGraphToList (
            Graph * gr,
            Node * st )
```

Copia os vértices de um grafo para uma lista de nós.

Cria uma nova lista de elementos baseada na posição e tipo dos vértices do grafo.

Parâmetros

gr	Grafo de origem.
st	Lista de destino, será atualizada.

Retorna

Node∗ Ponteiro para a nova lista resultante.

4.2.3.7 CopyListToGraph()

```
bool CopyListToGraph (
```

```
            Node * st,
            Graph * gr )
```

Copia os nós de uma lista para um grafo.

Converte cada nó da lista de elementos (Node) para um vértice no grafo, mantendo as posições e valores. Atribui automaticamente arestas entre os vértices conforme necessário.

Parâmetros

st	Lista de nós a converter.
gr	Grafo onde os vértices serão inseridos.

Retorna

true Se a operação foi concluída com sucesso.

4.2.3.8 DFSMark()

```
void DFSMark (
            Vertex * current,
            int * count )
```

Marca recursivamente os vértices visitados em profundidade (DFS).

Utilizado para contagem ou verificação de conectividade.

Parâmetros

current	Vértice atual.
count	Ponteiro para o contador de visitas.

4.2.3.9 EdgeFindDest()

```
bool EdgeFindDest (
            Edge ** current,
            Edge ** previous,
            Vertex * pick )
```

Encontra uma aresta de destino e os ponteiros anteriores na lista.

Auxiliar para remoção de arestas.

Parâmetros

current	Ponteiro para a aresta atual.
previous	Ponteiro para a aresta anterior.
pick	Vértice de destino a encontrar.

Retorna

true Se a aresta foi encontrada.

4.2.3.10 FindElement()

```
Element * FindElement (
            void * item,
            Element * st )
```

Procura um elemento na lista.

Parâmetros

item	Ponteiro para o item a procurar.
st	Lista onde será feita a procura.

Retorna

Element∗ Ponteiro para o elemento encontrado ou NULL.

4.2.3.11 FindNodePos()

```
Node * FindNodePos (
            Node * st,
            Vector2 npos )
```

Procura um nó numa posição específica da matriz.

Percorre a lista de nós para encontrar um nó cuja posição coincida com a posição fornecida.

Parâmetros

st	Lista de nós a pesquisar.
npos	Posição (Vector2) a procurar.

Retorna

Node∗ Ponteiro para o nó encontrado ou NULL se não existir nenhum nó nessa posição.

Procura um nó numa posição específica da matriz.

Parâmetros

st	Ponteiro para a lista.
npos	Posição do nó a ser encontrado.

Retorna

Ponteiro para o nó encontrado, ou NULL se não existir.

4.2.3.12 FindPairs()

```
Element * FindPairs (
            Graph * gr,
            char a,
            char b )
```

Encontra e retorna uma lista de intersecções entre pares de antenas com ressonância A e B.

Percorre o grafo e encontra todos os pares de antenas (vértices) em que uma possui valor/resonância A e a outra B. Apenas regista intersecções diretas (arestas existentes). A lista resultante contém os pares encontrados.

Parâmetros

gr	Ponteiro para o grafo.
a	Carácter correspondente à primeira frequência (ex: 'A').
b	Carácter correspondente à segunda frequência (ex: 'B').

Retorna

Element∗ Lista de elementos contendo os pares encontrados.

4.2.3.13 FindVertexAt()

```
Vertex * FindVertexAt (
            Vertex * st,
            Vector2 npos )
```

Procura um vértice no grafo a partir de uma posição.

Parâmetros

st	Lista de vértices do grafo.
npos	Posição procurada.

Retorna

Vertex∗ Ponteiro para o vértice encontrado, ou NULL se não existir.

4.2.3.14 FindVertexById()

```
Vertex * FindVertexById (
            Vertex * vertices,
            int id )
```

Procura um vértice no grafo com base no seu identificador (ID).

Percorre a lista de vértices e retorna o ponteiro correspondente ao ID.

Parâmetros

vertices	Lista de vértices do grafo.
id	Identificador único do vértice a encontrar.

Retorna

Vertex∗ Ponteiro para o vértice correspondente, ou NULL se não encontrado.

4.2.3.15 FreeElements()

```
Element * FreeElements (
            Element * head )
```

Liberta toda a memória alocada por uma lista de elementos.

Parâmetros

head	Início da lista.

Retorna

Element∗ NULL, lista vazia.

4.2.3.16 FreeGraph()

```
bool FreeGraph (
            Graph * gr )
```

Liberta todos os vértices e arestas de um grafo.

Remove completamente todos os dados da estrutura de grafo e reinicia os contadores.

Parâmetros

gr	Ponteiro para o grafo a ser limpo.

Retorna

true Se a operação foi concluída com sucesso.

4.2.3.17 FreeNodes()

```
Node * FreeNodes (
```

```
            Node * head )
```

Procura um nó em uma posição específica.

Parâmetros

st	Lista de nós.
npos	Posição a ser pesquisada.

Retorna

Node∗ Ponteiro para o nó encontrado ou NULL se não existir.

4.2.3.18 GraphBFS()

```
Element * GraphBFS (
            Graph * gr,
            Vertex * start )
```

Realiza uma pesquisa em largura (BFS) a partir de um vértice inicial.

Visita todos os vértices acessíveis por níveis, e armazena os visitados numa lista.

Parâmetros

gr	Grafo onde a pesquisa será realizada.
start	Vértice inicial da pesquisa.

Retorna

Element∗ Lista de elementos visitados pela BFS.

4.2.3.19 GraphDFS()

```
Element * GraphDFS (
            Graph * gr,
            Vertex * start )
```

Realiza uma pesquisa em profundidade (DFS) a partir de um vértice inicial.

Visita todos os vértices conectados recursivamente e guarda a ordem numa lista.

Parâmetros

gr	Grafo onde a pesquisa será realizada.
start	Vértice inicial da pesquisa.

Retorna

Element* Lista de elementos visitados pela DFS.

4.2.3.20 GraphPaths()

```
Element * GraphPaths (
            Graph * gr,
            Vertex * start,
            Vertex * end )
```

Calcula todos os caminhos possíveis entre dois vértices.

Usa backtracking para encontrar todas as combinações possíveis.

Parâmetros

gr	Ponteiro para o grafo.
start	Vértice de início.
end	Vértice de destino.

Retorna

Element* Lista de caminhos encontrados.

4.2.3.21 InitGraph()

```
Graph * InitGraph ( )
```

Inicializa uma estrutura de grafo vazia.

Reserva memória para um novo grafo e inicializa o contador e a lista de vértices.

Retorna

Graph* Ponteiro para o novo grafo.

4.2.3.22 InsertEdge()

```
bool InsertEdge (
            Vertex * from,
            Vertex * dest )
```

Cria uma nova aresta entre dois vértices.

Apenas adiciona se a ligação ainda não existir.

Parâmetros

from	Vértice de origem.
dest	Vértice de destino.

Retorna

true Se a aresta foi criada com sucesso.

false Se a ligação já existia.

4.2.3.23 InsertElement()

```
Element * InsertElement (
            Element * dnew,
            Element * st )
```

Insere um elemento no início de uma lista.

Parâmetros

dnew	Novo elemento.
st	Lista existente.

Retorna

Element∗ Lista atualizada.

4.2.3.24 InsertElementAtEnd()

```
Element * InsertElementAtEnd (
            Element * new,
            Element * st )
```

Insere um elemento no final da lista.

Parâmetros

new	Novo elemento.
st	Lista atual.

Retorna

Element∗ Lista atualizada.

4.2.3.25 InsertNode()

```
Node * InsertNode (
```

```
            Node * dnew,
            Node * st )
```

Insere um novo nó na lista, verificando se a posição está disponível.

Parâmetros

dnew	Novo nó a ser inserido.
st	Ponteiro para a lista.

Retorna

Ponteiro para a lista atualizada.

4.2.3.26 InsertVertex()

```
Vertex * InsertVertex (
            Vertex * dnew,
            Vertex * st,
            bool edge )
```

Insere um vértice na lista de vértices.

Parâmetros

dnew	Novo vértice.
st	Lista existente de vértices.
edge	Define se deve adicionar arestas automaticamente para vértices compatíveis.

Retorna

Vertex∗ Lista atualizada.

4.2.3.27 IsNewEdge()

```
bool IsNewEdge (
            Vertex * from,
            Vertex * dest )
```

Verifica se uma ligação entre dois vértices já existe.

Parâmetros

from	Vértice de origem.
dest	Vértice de destino.

Retorna

true Se já existe uma aresta entre os vértices.

false Caso contrário.

4.2.3.28 MakeElement()

```
Element * MakeElement (
            Vertex * value )
```

Cria um novo elemento com base num vértice.

Parâmetros

value	Ponteiro para o vértice.

Retorna

Element∗ Novo elemento criado.

4.2.3.29 MakeNode()

```
Node * MakeNode (
            char value,
            Vector2 position )
```

Cria um novo elemento (nó) para a lista.

Parâmetros

value	Valor do elemento (ex: antena tipo A, efeito nefasto '#').
position	Posição na matriz.

Retorna

Ponteiro para o novo nó criado.

4.2.3.30 MakeSeenList()

```
Element * MakeSeenList (
            Graph * gr,
            int max )
```

Cria uma lista ordenada de elementos com base na ordem de visita (seen).

Parâmetros

gr	Grafo com os vértices já marcados.
max	Valor máximo da marcação seen.

Retorna

Element* Lista ordenada de vértices pela ordem de visita.

4.2.3.31 MakeVertex()

```
Vertex * MakeVertex (
            char value,
            Vector2 position,
            int * id )
```

Cria um novo vértice com valor e posição especificados.

Gera um identificador único com base no ponteiro de contador fornecido.

Parâmetros

value	Valor (tipo) do vértice (ex: 'A', 'B', '#').
position	Posição no espaço bidimensional.
id	Ponteiro para o contador global de identificadores.

Retorna

Vertex* Novo vértice criado.

4.2.3.32 NodeInBounds()

```
bool NodeInBounds (
            Vector2 pos )
```

Verifica se a posição está dentro dos limites da matriz.

Parâmetros

pos	Posição a ser verificada.

Retorna

true se estiver dentro dos limites, false caso contrário.

4.2.3.33 NoiseCheck()

```
Node * NoiseCheck (
            Node * st )
```

Verifica e aplica a regra de efeito nefasto aos elementos na lista.

Parâmetros

st	Lista de nós.

Retorna

Node∗ Lista atualizada após a verificação do efeito nefasto.

4.2.3.34 NoiseCheckAlt()

```
Node * NoiseCheckAlt (
            Node * st )
```

Versão alternativa da verificação de ruído que compara todos os elementos entre si.

Parâmetros

st	Lista de nós.

Retorna

Node∗ Lista atualizada após a verificação do efeito nefasto.

4.2.3.35 Pathing()

```
void Pathing (
            Vertex * current,
            Vertex * end,
            Path * head,
            Graph * gr )
```

Caminha recursivamente no grafo entre dois vértices, acumulando o caminho atual.

Parâmetros

current	Vértice atual.
end	Vértice de destino.
head	Caminho acumulado.
gr	Ponteiro para o grafo.

4.2.3.36 ReadGraphFile()

```
bool ReadGraphFile (
            const char * filename,
            Graph * gr )
```

Lê um grafo a partir de um ficheiro binário, incluindo os vértices e as arestas com base nas posições.

Os vértices são lidos primeiro, seguidos de um marcador (-1), e depois as arestas que ligam posições (Vector2).

Parâmetros

filename	Nome do ficheiro a ler.
gr	Ponteiro para o grafo a ser preenchido.

Retorna

true Se a leitura for bem-sucedida.

false Em caso de erro na leitura do ficheiro.

4.2.3.37 ReadListFile()

```
Node * ReadListFile (
            const char * filename,
            Node * st )
```

Lê uma lista de elementos a partir de um ficheiro.

Parâmetros

filename	Nome do ficheiro.

Retorna

Node∗ Lista carregada do ficheiro.

4.2.3.38 RemoveEdge()

```
bool RemoveEdge (
            Vertex * from,
            Vertex * dest )
```

Remove uma aresta entre dois vértices.

Parâmetros

from	Vértice de origem.
dest	Vértice de destino.

Retorna

true Se a aresta foi removida com sucesso.

4.2.3.39 RemoveElement()

```
Element * RemoveElement (
            Element * rm,
            Element * st )
```

Remove um elemento específico da lista.

Parâmetros

rm	Elemento a remover.
st	Lista original.

Retorna

Element∗ Lista atualizada.

4.2.3.40 RemoveNode()

```
Node * RemoveNode (
            Node * rm,
            Node * st )
```

Remove um nó da lista.

Parâmetros

rm	Ponteiro para o nó a ser removido.
st	Ponteiro para a lista.

Retorna

Ponteiro para a lista atualizada.

4.2.3.41 RemoveVertex()

```
Vertex * RemoveVertex (
            Vertex * rm,
            Vertex * st )
```

Remove um vértice e as suas arestas da lista.

Parâmetros

rm	Vértice a remover.
st	Lista de vértices.

Retorna

Vertex∗ Lista atualizada.

4.2.3.42 SaveGraphFile()

```
bool SaveGraphFile (
```

```
            const char * filename,
            Graph * gr )
```

Guarda um grafo num ficheiro binário, incluindo vértices e arestas por posição (Vector2).

Os vértices são escritos primeiro, seguidos de um marcador (-1), e depois as arestas como pares de posições com peso.

Parâmetros

filename	Nome do ficheiro onde será guardado.
gr	Ponteiro para o grafo a ser guardado.

Retorna

true Se a gravação for bem-sucedida.

false Em caso de erro ao abrir ou escrever no ficheiro.

4.2.3.43 SaveList()

```
bool SaveList (
            const char * filename,
            Node * st )
```

Guarda a lista de elementos num ficheiro.

Parâmetros

filename	Nome do ficheiro onde os dados serão armazenados.
st	Lista de nós a ser salva.

4.2.3.44 ValidNodePos()

```
bool ValidNodePos (
            Node * dnew,
            Node * st )
```

Verifica se a posição de um novo nó é válida (sem sobreposição).

Parâmetros

dnew	Novo nó.
st	Ponteiro para a lista.

Retorna

true se a posição for válida, false caso contrário.

4.2.3.45 Vector2Add()

```
Vector2 Vector2Add (
            Vector2 a,
            Vector2 b )
```

Soma dois vetores.

Parâmetros

a	Primeiro vetor.
b	Segundo vetor.

Retorna

Vector2 Resultado da soma.

4.2.3.46 Vector2Compare()

```
bool Vector2Compare (
            Vector2 a,
            Vector2 b )
```

Compara se dois vetores possuem as mesmas coordenadas.

Parâmetros

a	Primeiro vetor.
b	Segundo vetor.

Retorna

true Se forem iguais.

false Se forem diferentes.

4.2.3.47 Vector2Subtract()

```
Vector2 Vector2Subtract (
            Vector2 a,
            Vector2 b )
```

Subtrai um vetor do outro.

Parâmetros

a	Primeiro vetor.
b	Segundo vetor.

Retorna

Vector2 Resultado da subtração.

4.3 func.h

Ir para a documentação deste ficheiro.

```
00001
00011 #ifndef FUNC_H
00012 #define FUNC_H
00013
00014 #define SWIDTH 12
00015 #define SHEIGHT 12
00016 #define NOISERANGE 4
00017 #define MAXINPUT 100
00018
00019 #include <stdbool.h>
00020
00027 typedef struct Vector2
00028 {
00029     int x;
00030     int y;
00031 } Vector2;
00032
00033 extern Vector2 sSize;
00034 extern int noiseRange;
00035
00042 typedef struct Node
00043 {
00044     Vector2 pos;
00045     char value;
00046     struct Node *next;
00047 } Node;
00048
00055 typedef struct Edge
00056 {
00057     float weight;
00058     struct Vertex *dest;
00059     struct Edge *next;
00060 } Edge;
00061
00069 typedef struct Vertex
00070 {
00071     int id;
00072     Vector2 pos;
00073     char value;
00074     Edge *edges;
00075     int seen;
00076     struct Vertex *next;
00077 } Vertex;
00078
00084 typedef struct Graph
00085 {
00086     int count;
00087     Vertex *vertices;
00088 } Graph;
00089
00095 typedef struct Element
00096 {
00097     Vertex *item;
00098     struct Element *next;
00099 } Element;
00100
00106 typedef struct Path
00107 {
00108     Element *first;
00109     int max;
00110 } Path;
00111
00119 Node *MakeNode(char value, Vector2 position);
00120
00127 bool NodeInBounds(Vector2 pos);
00128
00136 Node *InsertNode(Node *st, Node *dnew);
00137
00145 Node *RemoveNode(Node *rm, Node *st);
00146
00153 Node *ClearNoise(Node *st);
00154
00162 bool ValidNodePos(Node *dnew, Node *st);
00163
```

```
00170 Node *NoiseCheck(Node *st);
00171
00178 Node *NoiseCheckAlt(Node *st);
00179
00187 Node *FreeNodes(Node *head);
00188
00198 Node *FindNodePos(Node *st, Vector2 npos);
00199
00207 Graph *InitGraph();
00208
00219 Vertex *MakeVertex(char value, Vector2 position, int *id);
00220
00231 bool InsertEdge(Vertex *from, Vertex *dest);
00232
00241 bool IsNewEdge(Vertex *from, Vertex *dest);
00242
00251 Vertex *InsertVertex(Vertex *dnew, Vertex *st, bool edge);
00252
00260 Vertex *RemoveVertex(Vertex *rm, Vertex *st);
00261
00268 bool ClearEdges(Vertex *old);
00269
00277 bool RemoveEdge(Vertex *from, Vertex *dest);
00278
00289 bool EdgeFindDest(Edge **current, Edge **previous, Vertex *pick);
00290
00300 bool AddEdges(Vertex *new, Vertex *st);
00301
00309 Vertex *FindVertexAt(Vertex *st, Vector2 npos);
00310
00320 Vertex *FindVertexById(Vertex *vertices, int id);
00321
00330 bool FreeGraph(Graph *gr);
00331
00340 bool ClearSeen(Vertex *st);
00341
00351 Element *GraphDFS(Graph *gr, Vertex *start);
00352
00361 void DFSMark(Vertex *current, int *count);
00362
00372 Element *GraphBFS(Graph *gr, Vertex *start);
00373
00381 Element *MakeSeenList(Graph *gr, int max);
00382
00393 Element *GraphPaths(Graph *gr, Vertex *start, Vertex *end);
00394
00403 void Pathing(Vertex *current, Vertex *end, Path *head, Graph *gr);
00404
00412 void AddPath(Path *head, Graph *gr, int max);
00413
00425 Element *FindPairs(Graph *gr, char a, char b);
00426
00433 Element *MakeElement(Vertex *value);
00434
00442 Element *InsertElement(Element *dnew, Element *st);
00443
00451 Element *InsertElementAtEnd(Element *new, Element *st);
00452
00460 Element *RemoveElement(Element *rm, Element *st);
00461
00469 Element *FindElement(void *item, Element *st);
00470
00477 Element *FreeElements(Element *head);
00478
00487 bool Vector2Compare(Vector2 a, Vector2 b);
00488
00496 Vector2 Vector2Subtract(Vector2 a, Vector2 b);
00497
00505 Vector2 Vector2Add(Vector2 a, Vector2 b);
00506
00517 bool ReadGraphFile(const char *filename, Graph *gr);
00518
00529 bool SaveGraphFile(const char *filename, Graph *gr);
00530
00537 Node *ReadListFile(const char *filename, Node *st);
00538
00545 bool SaveList(const char *filename, Node *st);
00546
00557 bool CopyListToGraph(Node *st, Graph *gr);
00558
00568 Node *CopyGraphToList(Graph *gr, Node *st);
00569
00570 #endif
```

4.4 Referência ao ficheiro src/interface.c

Interface do utilizador e visualização de estruturas.

```
#include <stdio.h>
#include <stdlib.h>
#include "func.h"
#include "interface.h"
#include <stdbool.h>
#include <string.h>
#include <stdarg.h>
#include <time.h>
```

Funções

- void **DrawMenu** ()

  Desenha o menu de opções no terminal.
- void Menu (Node ∗st, Graph ∗gr)

  Gerencia a interface do utilizador e manipulação da lista de nós.
- void **DrawCommands** ()

  Exibe os comandos disponíveis para manipulação da lista.
- void CommandIO (Node ∗st, Graph ∗gr)

  Interface baseada em comandos para manipulação da lista de elementos.
- bool HasBinExtension (const char ∗filename)

  Verifica se o nome de ficheiro termina com a extensão '.bin'.
- bool AskReplace (char v, Vector2 pos)

  Solicita ao utilizador confirmação para substituir um elemento existente na mesma posição.
- void DrawMatrix (Node ∗st)

  Desenha a matriz com os elementos presentes na lista.
- void ShowList (Node ∗st, char filter)

  Exibe a lista de nós, com um filtro opcional.
- void ShowGraph (Graph ∗gr, char filter)

  Exibe todos os vértices do grafo no terminal, com filtro por tipo de antena.
- bool ShowPath (Element ∗st, Vertex ∗start, Vertex ∗end)

  Exibe um percurso (caminho) entre dois vértices, se existir.
- bool ShowTraversal (Element ∗st, const char ∗label)

  Exibe uma travessia (BFS ou DFS) completa a partir de um vértice.
- void ShowResonancePairs (Element ∗list)

  Exibe os pares de antenas com ressonância A-B ou B-A encontrados na travessia.
- void Pause ()

  Pausa a execução do programa até que o usuário pressione uma tecla.
- void **InitLog** ()

  Inicializa o sistema de registo (log), criando ou recriando o ficheiro de log.
- void Log (const char ∗format,...)

  Regista mensagens formatadas no ficheiro de log.

4.4.1 Descrição detalhada

Interface do utilizador e visualização de estruturas.

Autor

Vitor Rezende (a31521@alunos.ipca.pt)

Versão

0.10

Data

2025-05-15

Copyright

Copyright (c) 2025

4.4.2 Documentação das funções

4.4.2.1 AskReplace()

```
bool AskReplace (
            char v,
            Vector2 pos )
```

Solicita ao utilizador confirmação para substituir um elemento existente na mesma posição.

Parâmetros

v	Valor do novo elemento.
pos	Posição onde o novo elemento será inserido.

Retorna

true Se o utilizador confirmar a substituição.

false Caso contrário.

4.4.2.2 CommandIO()

```
void CommandIO (
            Node * st,
            Graph * gr )
```

Interface baseada em comandos para manipulação da lista de elementos.

Interface de comandos para o utilizador controlar as operações da lista e do grafo.

Parâmetros

st	Lista de nós.

4.4.2.3 DrawMatrix()

```
void DrawMatrix (
            Node * st )
```

Desenha a matriz com os elementos presentes na lista.

Parâmetros

st	Lista de nós.

4.4.2.4 HasBinExtension()

```
bool HasBinExtension (
            const char * filename )
```

Verifica se o nome de ficheiro termina com a extensão '.bin'.

Parâmetros

filename	Nome do ficheiro.

Retorna

true Se a extensão for '.bin'.

false Caso contrário.

4.4.2.5 Log()

```
void Log (
            const char * format,
             ...  )
```

Regista mensagens formatadas no ficheiro de log.

Parâmetros

format	Texto formatado (igual ao printf).
...	Argumentos adicionais.

4.4.2.6 Menu()

```
void Menu (
            Node * st,
            Graph * gr )
```

Gerencia a interface do utilizador e manipulação da lista de nós.

Gerencia o menu interativo de manipulação da lista de nós e do grafo.

Parâmetros

st	Lista de nós.

4.4.2.7 Pause()

```
void Pause ( )
```

Pausa a execução do programa até que o usuário pressione uma tecla.

Pausa a execução do programa até que o utilizador pressione ENTER.

4.4.2.8 ShowGraph()

```
void ShowGraph (
            Graph * gr,
            char filter )
```

Exibe todos os vértices do grafo no terminal, com filtro por tipo de antena.

Parâmetros

gr	Grafo contendo os vértices.
filter	Carácter para filtrar (ex: 'A', 'B'). Use '*' para mostrar todos.

4.4.2.9 ShowList()

```
void ShowList (
            Node * st,
            char filter )
```

Exibe a lista de nós, com um filtro opcional.

Parâmetros

st	Lista de nós.
filter	Caracter usado para filtrar os elementos exibidos.

4.4.2.10 ShowPath()

```
bool ShowPath (
            Element * st,
            Vertex * start,
            Vertex * end )
```

Exibe um percurso (caminho) entre dois vértices, se existir.

Parâmetros

st	Lista de elementos contendo o percurso.
start	Vértice de início.
end	Vértice de fim.

Retorna

true Se o percurso for exibido com sucesso.

false Se o percurso for inválido ou vazio.

4.4.2.11 ShowResonancePairs()

```
void ShowResonancePairs (
            Element * list )
```

Exibe os pares de antenas com ressonância A-B ou B-A encontrados na travessia.

Parâmetros

list	Lista de elementos resultante de uma travessia.

4.4.2.12 ShowTraversal()

```
bool ShowTraversal (
            Element * st,
            const char * label )
```

Exibe uma travessia (BFS ou DFS) completa a partir de um vértice.

Parâmetros

st	Lista de elementos visitados.
label	Título da travessia (ex: "DFS", "BFS").

Retorna

true Se a travessia for válida e exibida.

false Se a travessia estiver vazia.

4.5 Referência ao ficheiro src/interface.h

Interface do utilizador e visualização de estruturas.

Macros

- #define **LOG** "log.txt"
- #define **LOG_OLD** "log_old.txt"
- #define **SFILE** "file.txt"

Funções

- void **DrawMenu** ()

  Desenha o menu de opções no terminal.
- void Menu (Node ∗st, Graph ∗gr)

  Gerencia o menu interativo de manipulação da lista de nós e do grafo.
- void **DrawCommands** ()

  Exibe os comandos disponíveis para manipulação da lista.
- void CommandIO (Node ∗st, Graph ∗gr)

  Interface de comandos para o utilizador controlar as operações da lista e do grafo.
- bool HasBinExtension (const char ∗filename)

  Verifica se o nome de ficheiro termina com a extensão '.bin'.
- bool AskReplace (char v, Vector2 pos)

  Solicita ao utilizador confirmação para substituir um elemento existente na mesma posição.
- void DrawMatrix (Node ∗st)

  Desenha a matriz com os elementos presentes na lista.
- void ShowList (Node ∗st, char filter)

  Exibe a lista de nós, com um filtro opcional.
- void ShowGraph (Graph ∗gr, char filter)

  Exibe todos os vértices do grafo no terminal, com filtro por tipo de antena.
- bool ShowPath (Element ∗st, Vertex ∗start, Vertex ∗end)

  Exibe um percurso (caminho) entre dois vértices, se existir.
- bool ShowTraversal (Element ∗st, const char ∗label)

  Exibe uma travessia (BFS ou DFS) completa a partir de um vértice.
- void ShowResonancePairs (Element ∗list)

  Exibe os pares de antenas com ressonância A-B ou B-A encontrados na travessia.
- void Pause ()

  Pausa a execução do programa até que o utilizador pressione ENTER.
- void **InitLog** ()

  Inicializa o sistema de registo (log), criando ou recriando o ficheiro de log.
- void Log (const char ∗format,...)

  Regista mensagens formatadas no ficheiro de log.

4.5.1 Descrição detalhada

Interface do utilizador e visualização de estruturas.

Autor

Vitor Rezende (a31521@alunos.ipca.pt)

Versão

0.10

Data

2025-05-15

Copyright

Copyright (c) 2025

4.5.2 Documentação das funções

4.5.2.1 AskReplace()

```
bool AskReplace (
            char v,
            Vector2 pos )
```

Solicita ao utilizador confirmação para substituir um elemento existente na mesma posição.

Parâmetros

v	Valor do novo elemento.
pos	Posição onde o novo elemento será inserido.

Retorna

true Se o utilizador confirmar a substituição.

false Caso contrário.

4.5.2.2 CommandIO()

```
void CommandIO (
            Node * st,
            Graph * gr )
```

Interface de comandos para o utilizador controlar as operações da lista e do grafo.

Parâmetros

st	Lista de nós.
gr	Estrutura de grafo.

Interface de comandos para o utilizador controlar as operações da lista e do grafo.

Parâmetros

st	Lista de nós.

4.5.2.3 DrawMatrix()

```
void DrawMatrix (
            Node * st )
```

Desenha a matriz com os elementos presentes na lista.

Parâmetros

st	Lista de nós.

4.5.2.4 HasBinExtension()

```
bool HasBinExtension (
            const char * filename )
```

Verifica se o nome de ficheiro termina com a extensão '.bin'.

Parâmetros

filename	Nome do ficheiro.

Retorna

true Se a extensão for '.bin'.

false Caso contrário.

4.5.2.5 Log()

```
void Log (
            const char * format,
             ...  )
```

Regista mensagens formatadas no ficheiro de log.

Parâmetros

format	Texto formatado (igual ao printf).
...	Argumentos adicionais.

4.5.2.6 Menu()

```
void Menu (
            Node * st,
            Graph * gr )
```

Gerencia o menu interativo de manipulação da lista de nós e do grafo.

Parâmetros

st	Lista de nós.
gr	Estrutura de grafo.

Gerencia o menu interativo de manipulação da lista de nós e do grafo.

Parâmetros

st	Lista de nós.

4.5.2.7 Pause()

```
void Pause ( )
```

Pausa a execução do programa até que o utilizador pressione ENTER.

Pausa a execução do programa até que o utilizador pressione ENTER.

4.5.2.8 ShowGraph()

```
void ShowGraph (
            Graph * gr,
            char filter )
```

Exibe todos os vértices do grafo no terminal, com filtro por tipo de antena.

Parâmetros

gr	Grafo contendo os vértices.
filter	Carácter para filtrar (ex: 'A', 'B'). Use '∗' para mostrar todos.

4.5.2.9 ShowList()

```
void ShowList (
            Node * st,
            char filter )
```

Exibe a lista de nós, com um filtro opcional.

Parâmetros

st	Lista de nós.
filter	Caracter usado para filtrar os elementos exibidos.

4.5.2.10 ShowPath()

```
bool ShowPath (
            Element * st,
            Vertex * start,
            Vertex * end )
```

Exibe um percurso (caminho) entre dois vértices, se existir.

Parâmetros

st	Lista de elementos contendo o percurso.
start	Vértice de início.
end	Vértice de fim.

Retorna

true Se o percurso for exibido com sucesso.

false Se o percurso for inválido ou vazio.

4.5.2.11 ShowResonancePairs()

```
void ShowResonancePairs (
            Element * list )
```

Exibe os pares de antenas com ressonância A-B ou B-A encontrados na travessia.

Parâmetros

list	Lista de elementos resultante de uma travessia.

4.5.2.12 ShowTraversal()

```
bool ShowTraversal (
```

```
        Element * st,
        const char * label )
```

Exibe uma travessia (BFS ou DFS) completa a partir de um vértice.

Parâmetros

st	Lista de elementos visitados.
label	Título da travessia (ex: "DFS", "BFS").

Retorna

true Se a travessia for válida e exibida.

false Se a travessia estiver vazia.

4.6 interface.h

Ir para a documentação deste ficheiro.

```
00001
00012 #ifndef INTERFACE_H
00013 #define INTERFACE_H
00014
00015 #define LOG "log.txt"
00016 #define LOG_OLD "log_old.txt"
00017 #define SFILE "file.txt"
00018
00022 void DrawMenu();
00023
00030 void Menu(Node* st, Graph* gr);
00031
00035 void DrawCommands();
00036
00043 void CommandIO(Node* st, Graph* gr);
00044
00052 bool HasBinExtension(const char *filename);
00053
00062 bool AskReplace(char v, Vector2 pos);
00063
00069 void DrawMatrix(Node *st);
00070
00077 void ShowList(Node *st, char filter);
00078
00085 void ShowGraph(Graph * gr, char filter);
00086
00096 bool ShowPath(Element *st, Vertex* start, Vertex* end);
00097
00106 bool ShowTraversal(Element *st, const char* label);
00107
00113 void ShowResonancePairs(Element* list);
00114
00118 void Pause();
00119
00123 void InitLog();
00124
00131 void Log(const char *format, ...);
00132
00133 #endif
```

4.7 Referência ao ficheiro src/main.c

```
#include <stdio.h>
#include "func.h"
#include "interface.h"
```

Funções

- int **main** ()

4.7.1 Descrição detalhada

Autor

Vitor Rezende (a31521@alunos.ipca.pt)

Versão

0.10

Data

2025-03-18

Copyright

Índice