

**IPCA**



**INSTITUTO POLITÉCNICO  
DO CÁVADO E DO AVE  
ESCOLA SUPERIOR  
DE TECNOLOGIA**

**Instituto Politécnico do Cávado e do Ave**

**Escola Superior de Tecnologia**

**Estruturas de Dados Avançadas**

**Licenciatura em Engenharia de Sistemas Informáticos**

**Trabalho Prático**

Fase 2

Fábio Alexandre Gomes Fernandes – a22996

março de 2025



## Resumo

Este projeto, desenvolvido no âmbito da unidade curricular de Estruturas de Dados Avançadas, consiste na implementação de um sistema de gestão de antenas e deteção de interferências baseado em grafos. Cada antena é representada como um vértice num grafo não dirigido, sendo criadas arestas entre antenas que partilham a mesma frequência de ressonância.

Foram implementadas operações fundamentais sobre grafos, nomeadamente a procura em profundidade (DFT), procura em largura (BFT), a descoberta de todos os caminhos possíveis entre duas antenas, e a deteção de interseções geométricas entre ligações de frequências distintas.

Adicionalmente, o sistema permite o armazenamento e carregamento do grafo em ficheiros binários, garantindo a persistência dos dados entre execuções. Toda a implementação foi realizada em linguagem C, respeitando as restrições propostas, como a não utilização de matrizes nem apontadores duplos, e foi documentada com *Doxygen* para facilitar a manutenção e a leitura do código.

## Índice

|  |    |
|--|----|
| 1. Introdução.....                         | 6  |
| 2. Enquadramento Teórico e Prático .....   | 8  |
| 2.1 Conceitos e Fundamentos .....          | 9  |
| 3. Análise e Especificação .....           | 9  |
| 4. Implementação .....                     | 11 |
| 4.1 Ficheiro “Grafos” .....                | 11 |
| 4.1.1 Código “grafos.h” .....              | 12 |
| 4.1.2 Código “grafos.c” .....              | 15 |
| 4.2 Ficheiro “estrutura.txt” .....         | 21 |
| 4.3 Ficheiro “main.c” .....                | 21 |
| 4.3.1 Função main .....                    | 21 |
| 5. Análise e Discussão de Resultados ..... | 23 |
| 6. Conclusão .....                         | 25 |
| 7. Referências .....                       | 26 |

## Índice de Ilustrações

|  |    |
|--|----|
| Figura 1: Estrutura de Ligacao .....                     | 12 |
| Figura 2: Estrutura de Vertice .....                     | 13 |
| Figura 3: Estrutura de Grafo .....                       | 13 |
| Figura 4: VerticeSimples .....                           | 13 |
| Figura 5: Funções declaradas em “grafos.h” .....         | 15 |
| Figura 6: Função "iniciarGrafo" .....                    | 16 |
| Figura 7: Função "adicionarAntena" .....                 | 16 |
| Figura 8: Função "criarLigacao" .....                    | 16 |
| Figura 9: Função "ligarAntenasIguais" .....              | 17 |
| Figura 10: Função "limparLigacoes" .....                 | 17 |
| Figura 11: Função "mostrarAntenas" .....                 | 17 |
| Figura 12: Função "encontrarAntena" .....                | 18 |
| Figura 13: Funções "visitarDFT" e "dft" .....            | 18 |
| Figura 14: Função "bft" .....                            | 18 |
| Figura 15: Função "mostrarTodosCaminhos" .....           | 19 |
| Figura 16: Função "mostrarIntersecoes" (uma parte) ..... | 19 |
| Figura 17: Função "guardarGrafoBinario" .....            | 20 |
| Figura 18: Função "carregarGrafoBinario" .....           | 20 |
| Figura 19: Ficheiro "estrutura.txt" .....                | 21 |
| Figura 20: Função "Main" (uma parte) .....               | 22 |
| Figura 21: Resultados (parte 1) .....                    | 23 |
| Figura 22: Resultados (parte 2) .....                    | 24 |

## **1. Introdução**

Este relatório documenta a segunda fase do projeto desenvolvido na unidade curricular de Estruturas de Dados Avançadas (EDA), do curso de Licenciatura em Engenharia de Sistemas Informáticos, lecionada no Instituto Politécnico do Cávado e do Ave. Esta fase estende o sistema de gestão de antenas criado anteriormente, através da implementação de uma estrutura de grafo, com o objetivo de representar de forma mais eficaz as relações entre antenas com a mesma frequência.

### **Motivação**

A modelação de redes de comunicação com grafos permite analisar a conectividade entre antenas e detetar interferências de forma mais natural e eficiente. A aplicação de algoritmos como a procura em profundidade (DFT), em largura (BFT) e a verificação de interseções geométricas aproxima a solução de cenários reais, reforçando o domínio sobre estruturas de dados mais complexas.

### **Enquadramento**

Desenvolvido no segundo semestre do primeiro ano, este projeto baseia-se na implementação de um grafo não dirigido que substitui a estrutura anterior baseada em listas ligadas. Inclui a capacidade de realizar percursos, descobrir caminhos entre antenas e identificar interseções, bem como guardar e carregar o grafo de ficheiros binários. Todo o código foi desenvolvido em C e documentado com Doxygen.

### **Objetivos**

- Estruturar a rede de antenas sob a forma de um grafo não dirigido;
- Carregar e representar dados dinamicamente a partir de ficheiros;
- Implementar algoritmos de percurso (DFT e BFT);

- Descobrir todos os caminhos possíveis entre duas antenas;
- Detetar interseções geométricas entre ligações;
- Guardar e carregar o estado do grafo em ficheiros binários;
- Garantir modularidade, organização e clareza do código.

## Metodologia de Trabalho

A estrutura do grafo foi implementada com listas ligadas simples, representando antenas como vértices e as suas ligações como arestas. As antenas foram lidas de um ficheiro de texto e posicionadas com base nas suas coordenadas. Posteriormente, foram implementadas funcionalidades para realizar percursos (DFT e BFT), listar caminhos e verificar interseções entre ligações. O sistema foi documentado com Doxygen e controlado com Git.

**Repositório GitHub:** [https://github.com/fabiofernandes6/TP\\_EDA\\_A22996.git](https://github.com/fabiofernandes6/TP_EDA_A22996.git)

## Plano de Trabalho

O plano de trabalho para esta segunda fase foi organizado em três etapas principais:

- **Etapas 1:** Definição da estrutura de grafo e implementação das funções de criação de vértices e ligações entre antenas da mesma frequência;
- **Etapas 2:** Desenvolvimento das funcionalidades de percursos (DFT e BFT), descoberta de caminhos e deteção de interseções geométricas entre ligações;
- **Etapas 3:** Implementação da funcionalidade de armazenamento e carregamento binário, validação do sistema e documentação técnica com *Doxygen*.

## Estrutura do Documento

O presente relatório encontra-se organizado da seguinte forma:

- **Capítulo 2:** Apresenta os conceitos e fundamentos teóricos relacionados com estruturas de grafos, percursos em profundidade e largura, e interseções geométricas;

- **Capítulo 3:** Descreve a análise e especificação do sistema, incluindo os requisitos funcionais, a arquitetura do grafo e a modelação das operações sobre antenas;
- **Capítulo 4:** Detalha a implementação das várias componentes do projeto, com especial enfoque na estrutura do grafo, percursos, caminhos, interseções e armazenamento binário;
- **Capítulo 5:** Apresenta os testes realizados, os resultados obtidos e a respetiva análise crítica;
- **Capítulo 6:** Apresenta as conclusões finais sobre o desenvolvimento, os objetivos alcançados e as aprendizagens obtidas.

## 2. Enquadramento Teórico e Prático

Nesta fase do projeto, o foco recai sobre a aplicação prática dos conceitos de estruturas de dados baseadas em grafos, utilizados para representar de forma eficaz redes de comunicação entre antenas. Este capítulo apresenta os fundamentos teóricos que sustentam a solução implementada, bem como a sua relevância no contexto da deteção de interferências e gestão de conectividade.

A representação de cada antena como um vértice e das suas ligações como arestas permite explorar a estrutura da rede de forma eficiente, facilitando a análise de conectividade entre dispositivos com a mesma frequência. Esta abordagem é comum em problemas de redes, telecomunicações e sistemas distribuídos.

A adoção de um grafo não dirigido é justificada pela natureza bidirecional das ligações entre antenas da mesma frequência. Com base nesta estrutura, foi possível implementar algoritmos clássicos como a procura em profundidade (DFT) e em largura (BFT), que permitem percorrer a rede e identificar grupos de antenas interligadas.

Além disso, foi desenvolvida uma funcionalidade de deteção de interseções geométricas entre ligações de diferentes frequências, com base em análise vetorial, permitindo simular potenciais interferências no espaço. Estes conceitos aproximam o projeto de cenários reais de planeamento de redes e reforçam a importância da modelação correta dos dados.



A utilização da linguagem C obrigou ainda a uma gestão manual da memória e ao cumprimento de restrições técnicas como a não utilização de matrizes nem apontadores duplos, o que promoveu uma maior compreensão do funcionamento interno das estruturas de dados.

## **2.1 Conceitos e Fundamentos**

A gestão de redes de comunicação através de antenas com frequências de ressonância pode ser modelada eficientemente através de estruturas de dados baseadas em grafos. A representação de cada antena como um vértice e das suas ligações como arestas permite analisar a conectividade entre antenas com a mesma frequência, identificar caminhos possíveis e detetar situações de interseção entre ligações de diferentes grupos.

Os grafos não dirigidos são particularmente adequados neste contexto, uma vez que a ligação entre duas antenas é bidirecional e depende exclusivamente da coincidência de frequência. Operações clássicas como a procura em profundidade (DFT) e em largura (BFT) permitem explorar os componentes conexos do grafo, enquanto a análise geométrica das ligações possibilita a deteção de interseções espaciais entre segmentos, representando possíveis conflitos de sinal.

## **3. Análise e Especificação**

O sistema desenvolvido nesta fase tem como objetivo a gestão de antenas com base na respetiva localização e frequência de ressonância, recorrendo a uma representação baseada em grafos. Pretende-se, adicionalmente, permitir a execução de percursos sobre o grafo, a descoberta de caminhos entre antenas e a deteção de interseções geométricas entre ligações de diferentes frequências.

Esta secção apresenta os requisitos funcionais do sistema, a arquitetura lógica adotada e a modelação dos dados implementada, incluindo a estrutura do grafo, os vértices (antenas), as arestas (ligações), e os algoritmos associados à análise de conectividade e conflito de sinais.

- **Requisitos Funcionais**

Os principais objetivos desta fase foram:

- Carregamento automático de antenas a partir de ficheiro de texto;
- Representação de antenas como vértices num grafo não dirigido;
- Criação dinâmica de arestas entre antenas com a mesma frequência;
- Percursos automáticos por profundidade (DFT) e largura (BFT);
- Listagem de todos os caminhos possíveis entre duas antenas;
- Detecção de interseções geométricas entre ligações de diferentes frequências;
- Inserção e remoção de antenas com atualização automática do grafo;
- Persistência dos dados através de ficheiros binários;
- Apresentação dos resultados no terminal de forma clara e organizada.

- **Arquitetura Funcional**

O sistema foi desenvolvido com uma abordagem modular, organizado em três componentes principais:

- **Grafos:** módulo responsável pela representação do grafo de antenas, incluindo a criação de vértices e arestas, percursos (DFT e BFT), descoberta de caminhos, deteção de interseções e armazenamento binário;
- **main.c:** ficheiro principal que coordena o carregamento de dados, a execução das operações sobre o grafo e a visualização dos resultados no terminal;
- **estrutura.txt:** ficheiro externo utilizado como base de dados inicial para carregamento das antenas e respetivas posições no espaço.

Esta separação funcional permite uma organização clara do projeto, facilita a manutenção e promove a reutilização de código em contextos semelhantes.

- **Modelação dos Dados**

Os dados são representados através de um grafo não dirigido, onde cada antena corresponde a um vértice. Cada vértice armazena a frequência, as coordenadas e uma lista ligada de ligações para outros vértices com a mesma frequência. Esta abordagem permite

representar dinamicamente a conectividade entre antenas, possibilitando a execução de percursos, análise de caminhos e deteção de interseções.

A escolha por uma representação baseada em listas ligadas simples garante uma estrutura leve e eficiente, facilitando ainda a gestão manual da memória em linguagem C.

## 4. Implementação

Após a definição dos requisitos e da arquitetura funcional do sistema, procedeu-se à implementação das diferentes componentes da segunda fase. O desenvolvimento foi realizado em linguagem C, seguindo uma abordagem modular, com foco na reutilização e clareza do código. A estrutura do projeto foi organizada em múltiplos ficheiros, cada um dedicado a uma funcionalidade específica, facilitando a manutenção e a expansão do sistema.

A implementação contempla três blocos principais: o módulo responsável pela estrutura e manipulação do grafo de antenas, o ficheiro principal que executa as operações sobre o grafo e apresenta os resultados no terminal, e os ficheiros de suporte para carregamento de dados e persistência em formato binário. Cada componente foi desenvolvido de forma independente, garantindo a separação de responsabilidades e a consistência da estrutura de dados, respeitando as restrições impostas quanto à utilização exclusiva de listas ligadas e ausência de matrizes.

### 4.1 Ficheiro “Grafos”

Este ficheiro é responsável pela implementação da estrutura de dados que representa o grafo de antenas, assim como pelas funções associadas à sua manipulação. Cada antena é representada como um vértice, contendo as suas coordenadas, frequência e uma lista ligada de ligações para outras antenas com a mesma frequência.

Foram desenvolvidas funcionalidades para inicializar o grafo, adicionar antenas, criar ligações automáticas entre antenas compatíveis, e realizar operações como procura em profundidade (DFT), procura em largura (BFT), listagem de todos os caminhos entre dois

vértices e deteção de interseções geométricas entre ligações de diferentes frequências. O módulo inclui ainda funções para guardar e carregar o grafo em ficheiro binário.

#### 4.1.1 Código “grafos.h”

O ficheiro “**grafos.h**” é o cabeçalho do módulo responsável pela gestão do grafo de antenas. Nele são definidas as estruturas de dados que representam uma antena como vértice (Vertice), as ligações entre antenas (Ligacao), e o grafo em si (Grafo).

Este ficheiro contém ainda as declarações das funções necessárias para a criação e manipulação do grafo, incluindo operações de inserção de antenas, criação de ligações, percursos (DFT e BFT), descoberta de caminhos, deteção de interseções entre ligações e funcionalidades para guardar e carregar o grafo em ficheiros binários.

- **Estrutura de Dados**

A modelação do sistema na segunda fase recorre a estruturas baseadas em grafos para representar as antenas e as suas ligações. As estruturas de dados foram definidas da seguinte forma:

##### **Ligacao**

Representa uma ligação (aresta) entre dois vértices do grafo. Utiliza uma lista ligada simples para armazenar múltiplas ligações por vértice.

- **destino:** Índice do vértice de destino da ligação;
- **seguinte:** Apontador para a próxima ligação.

```
/**
 * @brief Representa uma ligação (aresta) entre dois vértices do grafo.
 */
typedef struct Ligacao {
    int destino;           ///< Índice do vértice de destino
    struct Ligacao* seguinte; ///< Próxima ligação
} Ligacao;
```

*Figura 1: Estrutura de Ligacao*

## Vertice

Representa uma antena individual no grafo, armazenando as suas características e as ligações associadas.

- **freq:** Frequência da antena;
- **x, y:** Coordenadas da antena no espaço;
- **ligacoes:** Lista ligada de ligações para outros vértices com a mesma frequência.

```
/**
 * @brief Representa um vértice (antena) no grafo.
 */
typedef struct Vertice {
    char freq;          ///< Frequência da antena
    int x, y;           ///< Coordenadas da antena
    Ligacao* ligacoes; ///< Lista de ligações para outras antenas
} Vertice;
```

Figura 2: Estrutura de Vertice

## Grafo

Contém o conjunto total de antenas registadas.

- **vertices:** Vetor de vértices com dimensão fixa (MAX\_VERTICES);
- **tamanho:** Número atual de vértices inseridos no grafo.

```
/**
 * @brief Representa um grafo com lista fixa de vértices.
 */
typedef struct Grafo {
    Vertice vertices[MAX_VERTICES]; ///< Lista de antenas (vértices)
    int tamanho;                   ///< Número atual de vértices
} Grafo;
```

Figura 3: Estrutura de Grafo

## VerticeSimples

Estrutura auxiliar utilizada para guardar ou carregar vértices em ficheiros binários.

- **freq:** Frequência da antena;
- **x, y:** Coordenadas da antena.

```
/**
 * @brief Estrutura simples usada para guardar e carregar vértices em ficheiros binários.
 */
typedef struct VerticeSimples {
    char freq;          ///< Frequência
    int x;              ///< Coordenada X
    int y;              ///< Coordenada Y
} VerticeSimples;
```

Figura 4: VerticeSimples

- **Funções Declaradas**

As funções declaradas em “grafos.h” permitem criar e manipular um grafo de antenas, incluindo operações sobre vértices (antenas), ligações, percursos, caminhos, interseções e persistência em ficheiro:

1. **int iniciarGrafo(Grafo\* g):** Inicializa o grafo, definindo o número de vértices como zero e todas as listas de ligações como nulas.
2. **int adicionarAntena(Grafo\* g, char freq, int x, int y):** Adiciona uma nova antena ao grafo, atribuindo-lhe frequência e coordenadas.
3. **int ligarAntenasIguais(Grafo\* g):** Cria ligações (arestas) entre todas as antenas que partilham a mesma frequência.
4. **int limparLigacoes(Grafo\* g):** Liberta a memória de todas as ligações do grafo, removendo as arestas entre vértices.
5. **int mostrarAntenas(Grafo\* g):** Imprime no terminal a lista de antenas (vértices), mostrando frequência e coordenadas.
6. **int encontrarAntena(Grafo\* g, int x, int y):** Devolve o índice de uma antena com as coordenadas indicadas, ou -1 se não existir.
7. **int dft(Grafo\* g, int inicio):** Executa um percurso em profundidade (Depth-First Traversal) a partir da antena indicada.
8. **int bft(Grafo\* g, int inicio):** Executa um percurso em largura (Breadth-First Traversal) a partir da antena indicada.
9. **int mostrarTodosCaminhos(Grafo\* g, int origem, int destino, int\* caminho, int comprimento, int\* visitado):** Lista todos os caminhos possíveis entre duas antenas especificadas, através de recursão.
10. **int mostrarIntersecoes(Grafo\* g, char freqA, char freqB):** Deteta e imprime interseções geométricas entre ligações de duas frequências distintas.
11. **int guardarGrafoBinario(Grafo\* g, const char\* nome):** Guarda o estado atual do grafo num ficheiro binário.
12. **int carregarGrafoBinario(Grafo\* g, const char\* nome):** Carrega os dados de um grafo a partir de um ficheiro binário previamente guardado.

```
> /** ...  
int iniciarGrafo(Grafo* g);  
  
> /** ...  
int adicionarAntena(Grafo* g, char freq, int x, int y);  
  
> /** ...  
int ligarAntenasIguais(Grafo* g);  
  
> /** ...  
int limparLigacoes(Grafo* g);  
  
> /** ...  
int mostrarAntenas(Grafo* g);  
  
> /** ...  
int encontrarAntena(Grafo* g, int x, int y);  
  
> /** ...  
int dft(Grafo* g, int inicio);  
  
> /** ...  
int bft(Grafo* g, int inicio);  
  
> /** ...  
int mostrarTodosCaminhos(Grafo* g, int origem, int destino, int* caminho, int comprimento, int* visitado);  
  
> /**|...  
int mostrarIntersecoes(Grafo* g, char freqA, char freqB);  
  
> /** ...  
int guardarGrafoBinario(Grafo* g, const char* nome);  
  
> /** ...  
int carregarGrafoBinario(Grafo* g, const char* nome);
```

Figura 5: Funções declaradas em “grafos.h”

#### 4.1.2 Código “grafos.c”

O ficheiro “**grafos.c**” implementa as funções necessárias para a gestão de um grafo de antenas. Estas funções permitem adicionar antenas como vértices, criar ligações entre antenas com a mesma frequência, realizar percursos em profundidade (DFT) e em largura (BFT), descobrir todos os caminhos possíveis entre duas antenas e detetar interseções geométricas entre ligações de diferentes frequências.

Adicionalmente, estão incluídas funcionalidades para guardar e carregar o grafo em ficheiros binários, bem como para limpar as ligações entre vértices, garantindo uma gestão dinâmica, modular e eficiente da estrutura.

### 1. Inicialização do grafo (iniciarGrafo)

Inicializa um grafo vazio, definindo o número de vértices como zero e todas as listas de ligações a NULL.

```
> /** ...  
int iniciarGrafo(Grafo* g) {  
    g->tamanho = 0;  
    for (int i = 0; i < MAX_VERTICES; i++) {  
        g->vertices[i].ligacoes = NULL;  
    }  
    return 1;  
}
```

Figura 6: Função "iniciarGrafo"

### 2. Criação de uma antena no grafo (adicionarAntena)

Adiciona uma nova antena ao grafo como vértice, atribuindo-lhe frequência e coordenadas. Retorna o índice da antena adicionada ou -1 se o grafo estiver cheio.

```
> /** ...  
int adicionarAntena(Grafo* g, char freq, int x, int y) {  
    if (g->tamanho >= MAX_VERTICES) return -1;  
    g->vertices[g->tamanho].freq = freq;  
    g->vertices[g->tamanho].x = x;  
    g->vertices[g->tamanho].y = y;  
    g->vertices[g->tamanho].ligacoes = NULL;  
    g->tamanho++;  
    return g->tamanho - 1;  
}
```

Figura 7: Função "adicionarAntena"

### 3. Criação de uma ligação entre antenas (criarLigacao)

Cria uma aresta entre dois vértices. Aloca dinamicamente uma nova ligação e insere-a no início da lista ligada de ligações do vértice de origem.

```
> /** ...  
int criarLigacao(Grafo* g, int origem, int destino) {  
    Ligacao* nova = (Ligacao*)malloc(sizeof(Ligacao));  
    if (nova == NULL) return 0;  
    nova->destino = destino;  
    nova->seguinte = g->vertices[origem].ligacoes;  
    g->vertices[origem].ligacoes = nova;  
    return 1;  
}
```

Figura 8: Função "criarLigacao"



#### 4. Ligação automática de antenas iguais (ligarAntenasIguais)

Cria ligações bidirecionais entre todas as antenas que partilham a mesma frequência, evitando ligações duplicadas.

```
> /** ...
int ligarAntenasIguais(Grafo* g) {
    for (int i = 0; i < g->tamanho; i++) {
        for (int j = i + 1; j < g->tamanho; j++) {
            if (g->vertices[i].freq == g->vertices[j].freq) {
                criarLigacao(g, i, j);
                criarLigacao(g, j, i);
            }
        }
    }
    return 1;
}
```

Figura 9: Função "ligarAntenasIguais"

#### 5. Limpeza das ligações (limparLigacoes)

Liberta a memória de todas as listas de ligações associadas aos vértices do grafo. Esta função é útil para atualizar as ligações sem duplicação.

```
> /** ...
int limparLigacoes(Grafo* g) {
    for (int i = 0; i < g->tamanho; i++) {
        Ligacao* atual = g->vertices[i].ligacoes;
        while (atual != NULL) {
            Ligacao* temp = atual;
            atual = atual->seguinte;
            free(temp);
        }
        g->vertices[i].ligacoes = NULL;
    }
    return 1;
}
```

Figura 10: Função "limparLigacoes"

#### 6. Mostrar a lista de antenas (mostrarAntenas)

Percorre o vetor de vértices do grafo e imprime, no terminal, as coordenadas e frequência de cada antena registada.

```
> /** ...
int mostrarAntenas(Grafo* g) {
    for (int i = 0; i < g->tamanho; i++) {
        printf("\tAntena %c em (%d,%d)\n", g->vertices[i].freq, g->vertices[i].x, g->vertices[i].y);
    }
    return 1;
}
```

Figura 11: Função "mostrarAntenas"

## 7. Procura de antena por coordenadas (encontrarAntena)

Devolve o índice de uma antena com as coordenadas indicadas. Retorna -1 se a antena não existir no grafo.

```
> /** ...
int encontrarAntena(Grafo* g, int x, int y) {
    for (int i = 0; i < g->tamanho; i++) {
        if (g->vertices[i].x == x && g->vertices[i].y == y) return i;
    }
    return -1;
}
```

Figura 12: Função "encontrarAntena"

## 8. Percurso em profundidade (dft e visitarDFT)

Executa um percurso em profundidade (Depth-First Traversal) a partir de uma antena. A visita é feita recursivamente através da função auxiliar "visitarDFT".

```
> /** ...
int visitarDFT(Grafo* g, int atual, int* visitado) {
    visitado[atual] = 1;
    printf("\tdft: (%d,%d)\n", g->vertices[atual].x, g->vertices[atual].y);
    Ligacao* l = g->vertices[atual].ligacoes;
    while (l != NULL) {
        if (!visitado[l->destino]) {
            visitarDFT(g, l->destino, visitado);
        }
        l = l->seguinte;
    }
    return 1;
}

> /** ...
int dft(Grafo* g, int inicio) {
    int visitado[MAX_VERTICES] = {0};
    return visitarDFT(g, inicio, visitado);
}
```

Figura 13: Funções "visitarDFT" e "dft"

## 9. Percurso em largura (bft)

Executa um percurso em largura (Breadth-First Traversal), utilizando uma fila e um vetor auxiliar de visitados para controlar a ordem da exploração.

```
int bft(Grafo* g, int inicio) {
    int visitado[MAX_VERTICES] = {0}, fila[MAX_VERTICES];
    int frente = 0, tras = 0;

    fila[tras++] = inicio;
    visitado[inicio] = 1;

    while (frente < tras) {
        int atual = fila[frente++];
        printf("\tbft: (%d,%d)\n", g->vertices[atual].x, g->vertices[atual].y);

        Ligacao* l = g->vertices[atual].ligacoes;
        while (l != NULL) {
            if (!visitado[l->destino]) {
                fila[tras++] = l->destino;
                visitado[l->destino] = 1;
            }
            l = l->seguinte;
        }
    }
}
```

Figura 14: Função "bft"

## 10. Descoberta de todos os caminhos (mostrarTodosCaminhos)

Percorre recursivamente o grafo e imprime todos os caminhos possíveis entre duas antenas especificadas, sem repetir vértices já visitados.

```
> /** ...
int mostrarTodosCaminhos(Grafo* g, int origem, int destino, int* caminho, int comprimento, int* visitado) {
    visitado[origem] = 1;
    caminho[comprimento++] = origem;

    if (origem == destino) {
        for (int i = 0; i < comprimento; i++) {
            printf("\t(%d,%d)", g->vertices[caminho[i]].x, g->vertices[caminho[i]].y);
            if (i < comprimento - 1) printf(" -> ");
        }
        printf("\n");
    } else {
        Ligacao* l = g->vertices[origem].ligacoes;
        while (l != NULL) {
            if (!visitado[l->destino]) {
                mostrarTodosCaminhos(g, l->destino, destino, caminho, comprimento, visitado);
            }
            l = l->seguinte;
        }
    }

    visitado[origem] = 0;
    return 1;
}
```

Figura 15: Função "mostrarTodosCaminhos"

## 11. Detecção de interseções (mostrarIntersecoes)

Verifica se existem interseções geométricas entre ligações de antenas com frequências diferentes. Se detetadas, imprime as ligações que se cruzam.

```
> /** ...
int mostrarIntersecoes(Grafo* g, char freqA, char freqB) {
    int encontrou = 0;

    for (int i = 0; i < g->tamanho; i++) {
        if (g->vertices[i].freq != freqA) continue;

        Ligacao* la = g->vertices[i].ligacoes;
        while (la != NULL) {
            if (g->vertices[la->destino].freq != freqA || i > la->destino) {
                la = la->seguinte;
                continue;
            }

            int ax1 = g->vertices[i].x, ay1 = g->vertices[i].y;
            int ax2 = g->vertices[la->destino].x, ay2 = g->vertices[la->destino].y;

            for (int j = 0; j < g->tamanho; j++) {
                if (g->vertices[j].freq != freqB) continue;

                Ligacao* lb = g->vertices[j].ligacoes;
                while (lb != NULL) {
                    if (g->vertices[lb->destino].freq != freqB || j > lb->destino) {
                        lb = lb->seguinte;
                        continue;
                    }

                    int bx1 = g->vertices[j].x, by1 = g->vertices[j].y;
                    int bx2 = g->vertices[lb->destino].x, by2 = g->vertices[lb->destino].y;
                }
            }
        }
    }
}
```

Figura 16: Função "mostrarIntersecoes" (uma parte)

## 12. Guardar grafo em ficheiro binário (guardarGrafoBinario)

Escreve a quantidade de vértices e os dados básicos de cada antena (frequência e coordenadas) num ficheiro binário, para persistência do grafo.

```
> /** ...  
int guardarGrafoBinario(Grafo* g, const char* nome) {  
    FILE* f = fopen(nome, "wb");  
    if (!f) return 0;  
  
    fwrite(&g->tamanho, sizeof(int), 1, f);  
    for (int i = 0; i < g->tamanho; i++) {  
        VerticeSimples v = {  
            g->vertices[i].freq,  
            g->vertices[i].x,  
            g->vertices[i].y  
        };  
        fwrite(&v, sizeof(VerticeSimples), 1, f);  
    }  
  
    fclose(f);  
    return 1;  
}
```

Figura 17: Função "guardarGrafoBinario"

## 13. Carregar grafo de ficheiro binário (carregarGrafoBinario)

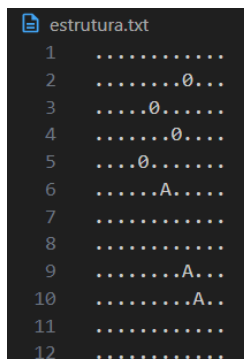
Lê os dados de um ficheiro binário, recria os vértices do grafo e restabelece automaticamente as ligações com base nas frequências.

```
> /** ...  
int carregarGrafoBinario(Grafo* g, const char* nome) {  
    FILE* f = fopen(nome, "rb");  
    if (!f) return 0;  
  
    int tamanhoLido = 0;  
    fread(&tamanhoLido, sizeof(int), 1, f);  
  
    iniciarGrafo(g);  
    for (int i = 0; i < tamanhoLido; i++) {  
        VerticeSimples v;  
        fread(&v, sizeof(VerticeSimples), 1, f);  
        adicionarAntena(g, v.freq, v.x, v.y);  
    }  
  
    fclose(f);  
    ligarAntenasIguais(g);  
    return 1;  
}
```

Figura 18: Função "carregarGrafoBinario"

## 4.2 Ficheiro “estrutura.txt”

O ficheiro “**estrutura.txt**” contém a representação da disposição das antenas num espaço bidimensional. Este ficheiro é utilizado pelo programa para carregar a configuração inicial das antenas e determinar a sua distribuição na matriz.



```
estrutura.txt
1 .....
2 .....0...
3 .....0...
4 .....0...
5 ....0.....
6 .....A....
7 .....
8 .....
9 .....A...
10 .....A..
11 .....
12 .....
```

Figura 19: Ficheiro "estrutura.txt"

## 4.3 Ficheiro “main.c”

O ficheiro “**main.c**” contém o programa principal responsável pela construção e análise do grafo de antenas. É neste ficheiro que se coordenam as operações de carregamento de dados, criação de ligações, execução de percursos, deteção de interseções e armazenamento em binário. A interação com o utilizador ocorre através do terminal, onde são apresentados os resultados obtidos.

### 4.3.1 Função main

A função principal organiza e executa as seguintes ações:

#### 1. Carregamento das antenas:

- O programa lê o ficheiro “estrutura.txt” e adiciona as antenas ao grafo com base na sua posição e frequência;
- Após o carregamento, são automaticamente criadas ligações entre antenas com a mesma frequência;
- A lista de antenas carregadas é apresentada no terminal.

## 2. Procura no grafo:

- A partir da antena na posição (6,5) são executados percursos em profundidade (DFT) e em largura (BFT), apresentando a ordem das antenas visitadas;
- É realizada também a listagem de todos os caminhos possíveis entre duas antenas específicas (ex: de (6,5) para (9,9)).

## 3. Verificação de interseções:

- O programa analisa interseções entre ligações de frequências diferentes, iniciando com a combinação A–O;
- Se não forem encontradas interseções, apresenta essa informação no terminal.

## 4. Exemplo de interseção geométrica:

- São inseridas duas novas antenas da frequência B em (6,8) e (8,5) com o objetivo de criar ligações que se cruzem com as ligações da frequência A;
- Após limpar e recriar as ligações, são detetadas e exibidas as interseções geométricas entre as ligações A–A e B–B.

## 5. Armazenamento em binário:

- O grafo é guardado num ficheiro binário, permitindo preservar a estrutura criada;
- Posteriormente, o grafo é carregado para uma nova variável e as antenas são listadas novamente, confirmando a integridade dos dados.

```

57 // Interseções entre frequências A e O
58 printf("\n>> Intersecoes entre frequencias A e O:\n");
59 mostrarIntersecoes(&g, 'A', 'O');
60
61 // Exemplo: adicionar antenas B para testar interseção
62 printf("\n(Exemplo) Inserir antenas B em (6,8) e (8,5) para demonstrar intersecao com A:\n");
63 adicionarAntena(&g, 'B', 6, 8);
64 adicionarAntena(&g, 'B', 8, 5);
65
66 limparLigacoes(&g); // Remover ligações antigas
67 ligarAntenasIguais(&g); // Atualizar com as novas ligações
68
69 // Interseções entre frequências A e B
70 printf("\n\tIntersecoes entre frequencias A e B:\n");
71 mostrarIntersecoes(&g, 'A', 'B');
72
73 // Guardar grafo em ficheiro binário
74 printf("\nA guardar grafo em binario...\n");
75 guardarGrafoBinario(&g, "grafo_guardado.dat");
76
77 // Carregar grafo do ficheiro binário
78 Grafo g2;
79 iniciarGrafo(&g2);
80 carregarGrafoBinario(&g2, "grafo_guardado.dat");
81
82 printf("\n>> Antenas carregadas do binario:\n");
83 mostrarAntenas(&g2);

```

Figura 20: Função "Main" (uma parte)

## 5. Análise e Discussão de Resultados

Neste capítulo são apresentados e analisados os principais resultados obtidos com a implementação do sistema na Fase 2, centrada na representação e análise de antenas através de grafos. A validação incide sobre a correta execução das funcionalidades exigidas: carregamento das antenas, percursos no grafo, descoberta de caminhos e deteção de interseções.

Na Figura 21, é apresentada a lista de antenas carregadas inicialmente a partir do ficheiro estrutura.txt, confirmando a correta leitura e inserção das antenas com frequências 0 e A. A representação textual no terminal valida a construção do grafo com base nestes dados.

Em seguida, são realizados dois percursos a partir da antena situada em (6,5), um em profundidade (DFT) e outro em largura (BFT), como mostrado na Figura 19. Ambos os percursos percorrem corretamente os vértices ligados pela mesma frequência, confirmando o funcionamento esperado dos algoritmos.

```
--- Sistema de Grafos ---

>> Antenas carregadas:
    Antena 0 em (8,1)
    Antena 0 em (5,2)
    Antena 0 em (7,3)
    Antena 0 em (4,4)
    Antena A em (6,5)
    Antena A em (8,8)
    Antena A em (9,9)

>> Procura em profundidade (DFT) a partir da antena (6,5):
    DFT: (6,5)
    DFT: (9,9)
    DFT: (8,8)

>> Procura em largura (BFT) a partir da antena (6,5):
    BFT: (6,5)
    BFT: (9,9)
    BFT: (8,8)
```

Figura 21: Resultados (parte 1)

Na Figura 22, observam-se todos os caminhos possíveis entre as antenas localizadas em (6,5) e (9,9). O sistema deteta corretamente dois caminhos distintos, validando a implementação recursiva da funcionalidade de descoberta de percursos.

Também na Figura 22, é realizada a verificação de interseções entre ligações das frequências A e O. Como esperado, o sistema informa que não existem interseções entre as ligações atuais destes grupos.

Posteriormente, são inseridas duas antenas com a frequência B nas coordenadas (6,8) e (8,5), conforme apresentado na mesma figura. Esta adição visa provocar interseções geométricas com as ligações das antenas A. Após recriação das ligações, o sistema deteta corretamente duas interseções entre as ligações A(6,5)-(9,9) e A(6,5)-(8,8) com a nova ligação B(6,8)-(8,5).

Por fim, a figura mostra o resultado do armazenamento e posterior carregamento do grafo a partir de um ficheiro binário. A listagem das antenas carregadas confirma a preservação total do estado do grafo, incluindo os novos vértices inseridos, garantindo o correto funcionamento do sistema de persistência.

```
>> Todos os caminhos entre (6,5) e (9,9):
      (6,5) ->      (9,9)
      (6,5) ->      (8,8) ->      (9,9)

>> Intersecoes entre frequencias A e O:
Nenhuma intersecao entre ligacoes de A e O.

(Exemplo) Inserir antenas B em (6,8) e (8,5) para demonstrar intersecao com A:

      Intersecoes entre frequencias A e B:
      Intersecao entre ligacao A(6,5)-(9,9) e B(6,8)-(8,5)
      Intersecao entre ligacao A(6,5)-(8,8) e B(6,8)-(8,5)

A guardar grafo em binario...

>> Antenas carregadas do binario:
      Antena 0 em (8,1)
      Antena 0 em (5,2)
      Antena 0 em (7,3)
      Antena 0 em (4,4)
      Antena A em (6,5)
      Antena A em (8,8)
      Antena A em (9,9)
      Antena B em (6,8)
      Antena B em (8,5)
```

Figura 22: Resultados (parte 2)



## 6. Conclusão

A segunda fase do projeto permitiu consolidar conhecimentos sobre estruturas de dados avançadas através da implementação de um sistema de gestão e análise de antenas com recurso a grafos. A transição de listas ligadas simples para uma estrutura de grafo não dirigido revelou-se adequada para representar as ligações entre antenas com a mesma frequência e permitiu uma exploração mais profunda da conectividade entre estas.

Foram alcançados todos os objetivos propostos, com destaque para a correta execução dos percursos em profundidade (DFT) e em largura (BFT), a identificação de todos os caminhos possíveis entre duas antenas, e a deteção de interseções geométricas entre ligações de diferentes frequências. A funcionalidade de persistência em ficheiro binário foi também devidamente implementada e validada, assegurando a integridade dos dados entre sessões.

O código foi desenvolvido em linguagem C, com uma organização modular clara e documentação gerada com *Doxygen*, promovendo a legibilidade e a manutenibilidade do projeto. Adicionalmente, foi respeitada a restrição de não utilização de matrizes nem apontadores duplos, o que reforça a eficiência e domínio sobre a alocação dinâmica de memória.

Esta fase evidenciou a importância da estruturação de dados na resolução de problemas reais, nomeadamente na simulação e análise de redes de comunicação. O trabalho realizado contribuiu significativamente para o desenvolvimento de competências técnicas e de pensamento algorítmico, essenciais na formação em Engenharia de Sistemas Informáticos.

## 7. Referências

- [1] “Doxygen download.” Accessed: Mar. 30, 2025. [Online]. Available: <https://www.doxygen.nl/download.html>
- [2] “latex (2) - Online LaTeX Editor Overleaf.” Accessed: Mar. 30, 2025. [Online]. Available: <https://www.overleaf.com/project/67e987ef9fecaabac97c54fc>