Trabalho de Estrutura de Dados Avançadas

Matheus da Rocha Delgado
31542 - Regime Pós-laboral

Docente
Luis Gonzaga Martins Ferreira

Ano letivo 2024/2025

Licenciatura em Engenharia de Sistemas Informáticos

Escola Superior de Tecnologia

Instituto Politécnico do Cávado e do Ave

Índice

1 Introdução

1.1 Motivação

A motivação deste projeto reside na necessidade de aplicar, aprofundar e consolidar os conhecimentos teóricos e práticos adquiridos no decorrer da Unidade Curricular de Estruturas de Dados Avançadas. O problema em análise – a gestão de antenas e a determinação dos efeitos nefastos resultantes do seu posicionamento numa matriz – expõe uma situação realista onde a manipulação de dados dinâmicos, a utilização de listas ligadas e a gestão eficiente de memória são cruciais.

Além disso, o projeto surge como resposta à ausência de soluções que integrem, de forma robusta, as diferentes funcionalidades exigidas pela especificação (carregamento de dados, manipulação e preservação em ficheiros, assim como a representação gráfica do cenário). No contexto atual, marcado pelo avanço das tecnologias e pela crescente necessidade de sistemas que otimizem o processamento e a gestão de dados, a implementação de soluções em linguagem C com uma abordagem modular e documentada (através do Doxygen) reveste-se de extrema pertinência. Tais soluções não só melhoram o entendimento dos conceitos fundamentais, mas também contribuem para a formação de uma base sólida para o desenvolvimento de sistemas de dimensão média em ambientes exigentes.

Na segunda fase do projeto, foi implementada uma modelação avançada utilizando grafos, onde cada antena é representada como um vértice e as arestas conectam antenas com a mesma frequência de ressonância. Esta abordagem permitiu a análise da conectividade entre antenas, a procura de caminhos possíveis e a deteção de interseções entre antenas de diferentes frequências. Algoritmos clássicos de busca, como a Busca em Profundidade (DFS) e a Busca em Largura (BFS), foram implementados para explorar o grafo e identificar relações complexas entre as antenas, consolidando assim os objetivos do projeto.

1.2 Problema

O desafio central deste projeto consiste em modelar e gerir, de forma dinâmica, o conjunto de antenas distribuídas por uma matriz que representa um ambiente urbano. Cada antena opera numa frequência específica, identificada por um carácter, e é posicionada mediante coordenadas que determinam a sua localização no mapa. O problema adquire uma dimensão adicional com a necessidade de identificar automaticamente as localizações onde ocorre o chamado "efeito nefasto". Este efeito surge quando duas antenas com a mesma frequência se alinham de modo que uma delas se encontre duas vezes mais distante de um ponto de referência do que a outra, criando zonas de interferência no sinal.

Na segunda fase, o problema foi expandido para incluir a análise da conectividade entre as antenas que partilham a mesma frequência, permitindo a identificação de caminhos possíveis entre elas e a deteção de interseções entre antenas de frequências diferentes. Esta nova dimensão é crucial para uma compreensão mais profunda das relações espaciais e funcionais entre as antenas, possibilitando a otimização da sua disposição.

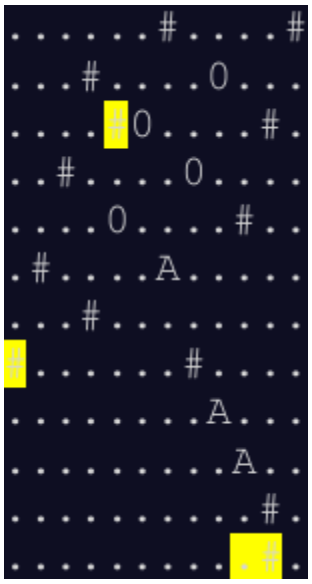

Figura 1: Matriz representativa de um espaço urbanjo Fonte: Enunciado do projeto.

1.3 Enquadramento

O enquadramento deste projeto situa-se na interseção entre a teoria de estruturas de dados e a prática da programação em C. Trabalhar com estruturas dinâmicas, nomeadamente listas ligadas, permite abordar problemas que requerem flexibilidade, escalabilidade e eficiência no uso de recursos. A modularização do código e a documentação com Doxygen constituem aspetos fundamentais, permitindo uma melhor manutenção e compreensão do sistema por parte de qualquer leitor, independentemente do seu nível de conhecimento em C ou na área de estruturas de dados.

Na segunda fase, o projeto evoluiu para incluir o uso de grafos, uma estrutura de dados mais avançada que permite modelar redes complexas de antenas interconectadas por frequências comuns. Esta abordagem é complementada pela implementação de algoritmos clássicos de busca, como a Busca em Profundidade (DFS) e a Busca em Largura (BFS), que são essenciais para explorar e analisar a conectividade entre as antenas, identificando componentes conectados e caminhos possíveis.

1.4 Objetivos

Os objetivos do projeto foram definidos de forma clara e sucinta, focando-se em:

- Consolidar conhecimentos teóricos e práticos: Aplicar os conceitos aprendidos sobre apontadores, alocação dinâmica de memória e gestão de estruturas de dados.
- Implementar uma solução robusta em C: Desenvolver funcionalidades que permitam a criação, inserção e remoção de registos em listas ligadas, assegurando a dedução dos pontos de efeito nefasto.
- Garantir modularidade e reutilização: Organizar o código em ficheiros distintos e promover a manutenção futura através de documentação rigorosa.
- Facilitar a preservação e análise dos dados: Utilizar técnicas de leitura e escrita em ficheiros binários e textuais, assegurando uma abordagem prática para a recuperação e visualização dos registos.

Na segunda fase, foram adicionados objetivos específicos para a análise avançada utilizando grafos:

- Modelar as antenas e suas conexões como um grafo, onde vértices representam antenas e arestas representam conexões entre antenas com a mesma frequência.
- Implementar algoritmos de busca em grafos, como DFS e BFS, para analisar a conectividade e identificar componentes conectados.
- Desenvolver funcionalidades para encontrar todos os caminhos possíveis entre duas antenas específicas.
- Detectar interseções entre antenas de diferentes frequências, permitindo uma análise mais aprofundada das relações espaciais.

1.5 Metodologia de Trabalho

A abordagem metodológica adotada para este projeto baseia-se num processo incremental e iterativo. Inicialmente, concentrou-se a implementação dos conceitos fundamentais, como a definição e manipulação de listas ligadas, a leitura e escrita de ficheiros e o cálculo dos efeitos nefastos decorrentes do alinhamento das antenas. Cada módulo foi desenvolvido, testado e integrado progressivamente para garantir a estabilidade e a modularidade do sistema.

Na transição para a segunda fase, foi aproveitado o código modular desenvolvido na Fase 1, adaptando-o para suportar a nova estrutura de grafos. As funcionalidades de grafos, incluindo a criação do grafo a partir da matriz, a implementação de algoritmos de busca e a detecção de interseções, foram desenvolvidas de forma incremental, com testes unitários para cada nova função. A documentação rigorosa, utilizando Doxygen, foi mantida e expandida para incluir as novas estruturas e funções, assegurando que o sistema permaneça compreensível e mantível.

1.6 Plano de Trabalho

No início deste projeto, o foco recaiu na definição e implementação das estruturas de dados, especificamente as listas ligadas, que permitiram a representação dinâmica das antenas e dos pontos onde ocorrem os efeitos nefastos. A primeira etapa envolveu o desenvolvimento e a validação das funções básicas de inserção, remoção e procura dos registos, bem como a criação dos módulos responsáveis pela leitura dos dados a partir de ficheiros de texto e pela preservação dos mesmos em ficheiros binários e em representações gráficas.

Na segunda fase, o plano de trabalho foi expandido para incluir a implementação de grafos. Inicialmente, foi desenvolvida a estrutura de dados para representar o grafo, com vértices correspondendo às antenas e arestas conectando antenas com a mesma frequência. Posteriormente, foram implementados os algoritmos de busca em profundidade (DFS) e busca em largura (BFS) para explorar o grafo. Cada funcionalidade foi testada incrementalmente, utilizando matrizes de diferentes tamanhos e complexidades, garantindo a corretude e eficiência das soluções desenvolvidas.

2 Estado da Arte

2.1 Listas Ligadas

Uma lista ligada é uma estrutura de dados dinâmica que organiza os elementos em "nós", sendo cada nó composto por um campo de dados e um apontador para o nó seguinte. Esta característica permite que a estrutura se expanda ou contraia conforme necessário, sem a necessidade de alocação contínua de memória. Conforme salientam Cormen et al. (2009), "a inserção em uma lista ligada pode ser efetuada em tempo constante", o que evidencia a eficiência desta operação.

2.2 Teoria dos Grafos

A teoria dos grafos é uma área da matemática e da ciência da computação que estuda as propriedades e aplicações de grafos, estruturas compostas por vértices (ou nós) conectados por arestas. No contexto deste projeto, os grafos são utilizados para modelar as antenas como vértices e as conexões entre antenas com a mesma frequência como arestas. Esta modelação permite a aplicação de algoritmos para análise de conectividade, procura de caminhos e detecção de componentes conectados.

2.3 Algoritmos DFS e BFS

A Busca em Profundidade (DFS) e a Busca em Largura (BFS) são algoritmos fundamentais para a travessia e exploração de grafos. O DFS explora o grafo de forma recursiva, visitando os vértices o mais profundamente possível antes de retroceder, enquanto o BFS explora os vértices nível a nível, começando pelo vértice inicial. Ambos os algoritmos são utilizados para identificar componentes conectados e verificar a existência de caminhos entre vértices.

2.4 Representações de Grafos em C

Em C, os grafos podem ser representados utilizando estruturas dinâmicas, como listas ligadas, para garantir flexibilidade e eficiência. Neste projeto, a estrutura do grafo foi implementada utilizando uma lista ligada de vértices, onde cada vértice contém uma lista ligada das arestas que dele partem. Esta abordagem permite a adição e remoção dinâmica de vértices e arestas, adequando-se às necessidades de gestão de antenas em tempo real.

3 Trabalho Desenvolvido na Fase 1 - Matrizes

3.1 Implementação em Linguagem C Estruturas e Definições

Estrutura Antena

A estrutura `Antena` é definida no ficheiro `dados.h` e contém os campos necessários para a identificação e gestão das antenas. O apontador `próximo` é fundamental para criar a ligação entre os elementos, constituindo assim uma lista ligada dinâmica (Figura 2).

- **linha** e **coluna**: Coordenadas da antena na matriz;
- **frequência**: Indica a frequência de operação da antena;
- **id**: Identificador único da antena;
- **próximo**: Apontador para a próxima antena na lista ligada.

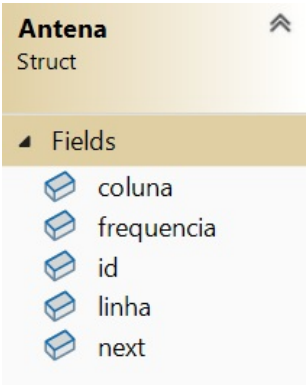

Figura 2: Representação da estrutura `Antena` no Visual Studio. Fonte: ClassDiagram Visual Studio.

Estrutura Nefasto

A estrutura `Nefasto` é utilizada para armazenar informações sobre os efeitos nefastos detetados (Figura 3), incluindo:

- **linha** e **coluna**: Coordenadas do efeito na matriz;
- **idAntena1** e **idAntena2**: Identificadores das antenas responsáveis pelo efeito;
- **próximo**: Apontador para o próximo efeito nefasto na lista ligada.

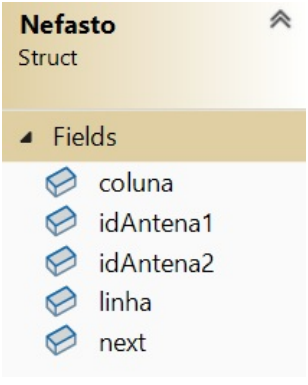

Figura 3: Representação da estrutura `Nefasto` no Visual Studio. Fonte: ClassDiagram Visual Studio.

3.2 Funcionalidades e Requisitos

3.2.1 Funcionalidades Implementadas

O sistema implementa várias funcionalidades que abrangem:

- Gestão de antenas: Criação, inserção ordenada, remoção, alteração e procura de antenas na lista;
- Deteção dos efeitos nefastos: Cálculo dos pontos onde ocorre o efeito nefasto para cada par de antenas com a mesma frequência, utilizando fórmulas que consideram a posição relativa das antenas;
- Preservação e recuperação dos dados: Gravação dos registos das antenas e dos efeitos nefastos em ficheiros binários e geração de uma representação gráfica em ficheiros de texto;

3.2.2 Especificação dos Dados de Entrada e Saída

- Dados de entrada: Ficheiro de texto contendo a representação matricial do mapa urbano, onde cada célula da matriz indica a presença ou ausência de uma antena, bem como a respetiva frequência;
- Dados a preservar: Estruturas dinâmicas (listas ligadas) que armazenam os registos das antenas e dos efeitos nefastos;
- Formato de saída:
  - Ficheiro binário que contém os dados das antenas e dos efeitos, permitindo a reconstrução da lista em execuções futuras;
  - Ficheiro de texto com a representação gráfica do cenário, onde as antenas são identificadas pela sua frequência e os efeitos nefastos são marcados com um símbolo;

3.3 Organização do Código

O código encontra-se organizado em três ficheiros principais, de forma a separar a implementação dos conceitos de interface e de dados:

- funcoes.c: Contém a implementação de todas as funções, desde a criação e manipulação das listas de antenas e efeitos, até à gestão dos ficheiros. Este ficheiro é rico em comentários no formato Doxygen, facilitando a compreensão do seu funcionamento;

- funcoes.h: Reúne as assinaturas das funções implementadas em funcoes.c, servindo de interface para o resto do programa, nomeadamente a função main;
- dados.h: Define as estruturas de dados (Antena e Nefasto) e inclui as bibliotecas necessárias, garantindo que os registos são definidos de forma única através do pragma once. A diretiva CRT-SECURE-NO-WARNINGS é também aplicada para permitir o uso de funções de I/O (neste caso fopen) sem restrições;

3.4 Implementação das Funções

3.4.1 Funções de Gestão de Antenas

As funções dedicadas à gestão das antenas incluem:

- criaAntena: Aloca memória e inicializa os campos de um novo registo de antena;
- inserirOrdenado: Insere o registo da antena na lista de forma ordenada, recorrendo às coordenadas (linha e coluna) para definir a posição correta;
- removeAntena: Remove um registo da lista com base nas coordenadas fornecidas, assegurando a correta atualização dos apontadores;
- alteraAntena: Permite a modificação dos dados de uma antena, identificada através do seu id, mantendo a consistência da lista;
- ProcuraAntena: Procura e retorna um apontador para uma antena com o id indicado, facilitando a sua manipulação;

3.5 Funções para os Efeitos Nefastos

No que toca à gestão dos efeitos nefastos, destacam-se as seguintes funções:

- inserirEfeito: Cria um novo registo para um efeito nefasto e insere-o na lista, mantendo a ordem baseada nas coordenadas. O apontador next é utilizado para ligar cada registo, criando assim uma lista ligada;
- atualizaEfeito: Percorre a lista de antenas e, para cada par de antenas com a mesma frequência, calcula os pontos de efeito utilizando fórmulas que determinam as coordenadas dos efeitos. Para cada par, dois registos são gerados e inseridos na lista de efeitos, garantindo que todos os pontos relevantes são registados;

- mostraLista: Apresenta, de forma tabular, os registos das listas de antenas e efeitos, permitindo a verificação visual e a depuração do sistema;

3.6 Gestão de Ficheiros

Os dados das antenas e dos efeitos nefastos são lidos a partir de ficheiros de texto e armazenados em ficheiros binários, permitindo uma recuperação eficiente.

A preservação e recuperação dos dados são alcançadas através de duas abordagens complementares:

- Gravação em ficheiro binário:
  - A função gravarFicheiroB grava sequencialmente os dados das antenas e dos efeitos num ficheiro binário.
  - Este método permite armazenar os dados de forma compacta e eficiente, sem a necessidade de conversão para formato textual.
- Geração de ficheiro de texto:
  - A função gravarMatrizTxt gera uma representação gráfica do cenário, onde cada antena é representada pelo seu identificador de frequência e os efeitos nefastos são marcados com um símbolo
  - Esta abordagem facilita a análise visual e a verificação da disposição das antenas e dos pontos de efeito no mapa.
- Leitura de ficheiro binário:
  - A função lerFicheirobinario permite reconstituir a lista de antenas a partir dos dados armazenados, possibilitando a recuperação e continuidade do trabalho em execuções subsequentes.

3.7 Cálculos e Lógica dos Efeitos Nefastos

A lógica para determinar os efeitos nefastos é implementada na função atualizaEfeito. Para cada par de antenas com a mesma frequência, os seguintes cálculos são efetuados:

- Cálculo do primeiro ponto de efeito:
  - Fórmula: efeitoX1 = 2 * linha(antena 1) - linha(antena 2);
  - Fórmula: efeito Y1 = 2 * coluna(antena 1) - coluna(antena 2)

- Cálculo do segundo ponto de efeito:
  - Fórmula: efeitoX2 = 2 * linha(antena 2) - linha(antena 1)
  - Fórmula: efeito Y2 = 2 * coluna(antena 2) - coluna(antena 1)

Estes pontos são posteriormente inseridos na lista de efeitos através da função inserirEfeito, que garante a ordenação e a preservação dos registos.

4 Trabalho Desenvolvido na Fase 2 – Grafos

4.1 Representação e Estruturas de Dados

Para a modelação do grafo, foram definidas novas estruturas em `C` no ficheiro `struct.h`, que incluem:

- **Grafo**: contém o número total de vértices e um apontador para o primeiro vértice.

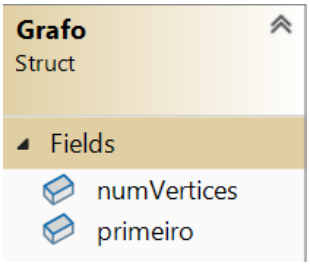

Figura 4: Representação da estrutura `Grafo` no Visual Studio. Fonte: ClassDiagram Visual Studio.

- **NoVertice**: representa uma antena, contendo os seus dados e apontadores para as arestas (ligações) e próximo vértice.

Figura 5: Representação da estrutura `Nó Vértice` no Visual Studio. Fonte: ClassDiagram Visual Studio.

- **Aresta**: representa a ligação de uma antena para outra (grafo orientado).

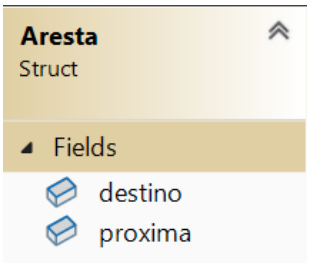

Figura 6: Representação da estrutura `Aresta` no Visual Studio. Fonte: ClassDiagram Visual Studio.

- **Fila**: usada na implementação da pesquisa em largura (BFS).

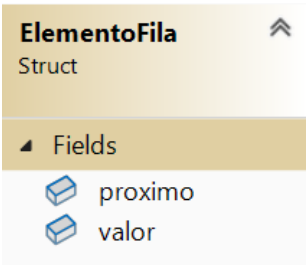

Figura 7: Representação da estrutura `Fila` no Visual Studio. Fonte: ClassDiagram Visual Studio.

- **ElementoCaminho** e **ListaCaminho**: usadas para armazenar caminhos múltiplos entre duas antenas.

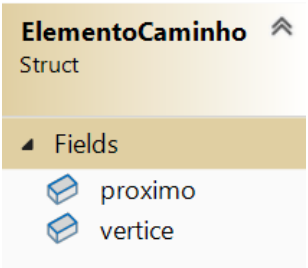

Figura 8: Representação da estrutura `Caminho` no Visual Studio. Fonte: ClassDiagram Visual Studio.

- **Intersecao**: representa uma ligação possível entre antenas de frequências distintas.

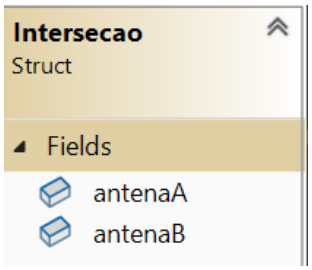

Figura 9: Representação da estrutura `Interseção` no Visual Studio. Fonte: ClassDiagram Visual Studio.

4.2 Construção do Grafo

O grafo é criado a partir da leitura de um ficheiro de texto que representa a matriz urbana. Cada antena identificada é transformada num vértice. As arestas são adicionadas automaticamente entre pares de antenas com a mesma frequência. A função responsável por esta operação é `carregarDadosGrafo`.

4.3 Operações sobre o Grafo

Foram implementadas diversas operações sobre o grafo, com destaque para:

- **Busca em Profundidade (DFS)**: Utilizada para explorar os vértices de forma recursiva a partir de uma antena.

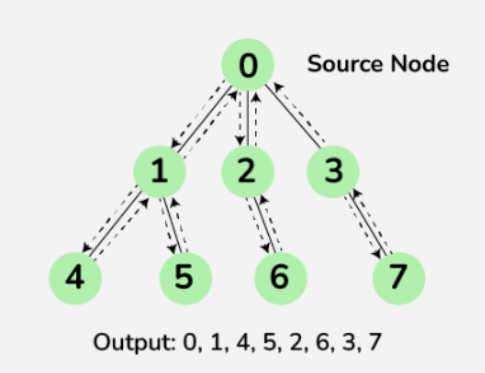

Figura 10: Representação grafica da busca em profundidade. Fonte:geeksforgeeks - Difference between BFS and DFS.

- **Busca em Largura (BFS)**: Implementada com o auxílio de uma fila, útil para encontrar o menor caminho em termos de saltos.

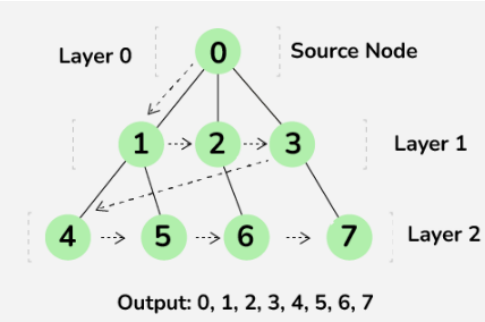

Figura 11: Representação grafica da busca em largura. Fonte:geeksforgeeks - Difference between BFS and DFS.

- **Procura de Caminhos**: A função `encontrarCaminhos` encontra todos os caminhos possíveis entre duas antenas.
- **Interseções**: A função `encontrarIntersecoes` identifica pares de antenas de diferentes frequências e mapeia as respetivas localizações.

4.4 Pseudocódigo da Busca em Largura e Busca em Profundidade

Algorithm 1 Busca em Largura (BFS)

```
1:  procedure BFS(grafo, verticeInicial)
2:      Inicializar fila vazia
3:      Marcar verticeInicial como visitado
4:      Enfileirar verticeInicial
5:      while fila não vazia do
6:          v ← desenfileirar(fila)
7:          for all vizinhos de v do
8:              if vizinho não visitado then
9:                  Marcar vizinho como visitado
10:                 Enfileirar vizinho
11:             end if
12:         end for
13:     end while
14: end procedure
```

Algorithm 2 Busca em Profundidade (DFS)

```
1: procedure DFS(grafo, verticeInicial, visitados, resultado, tamanhoResultado)
2:     if grafo = NULL verticeInicial = NULL visitados = NULL resultado = NULL then
3:         return false
4:     end if
5:     if resultado = NULL then
6:         Alocar resultado com tamanho grafo.numVertices
7:         tamanhoResultado ← 0
8:     end if
9:     indice ← EncontrarIndice(grafo, verticeInicial)
10:    if indice ≥ grafo.numVertices then
11:        return false
12:    end if
13:    if not visitados[indice] then
14:        visitados[indice] ← true
15:        resultado[tamanhoResultado] ← verticeInicial
16:        tamanhoResultado ← tamanhoResultado + 1
17:        for cada aresta em verticeInicial.primeiraAresta do
18:            DFS(grafo, aresta.destino, visitados, resultado, tamanhoResultado)
19:        end for
20:    end if
21:    return true
22: end procedure
```

4.5 Vantagens e Observações

A utilização de grafos trouxe várias vantagens face à abordagem inicial:

- Representação natural das ligações entre antenas com a mesma frequência;
- Facilidade na análise de conectividade, identificação de clusters e deteção de redundância;
- Estrutura modular, que permite extensões para algoritmos mais complexos (ex.: Dijkstra ou Kruskal).

Contudo, a complexidade na gestão de memória aumenta, requerendo especial atenção à libertação de recursos para evitar fugas.

5 Conclusão

O presente projeto demonstrou a implementação de um sistema robusto para a gestão de antenas e dos respetivos efeitos nefastos, consolidando os conhecimentos adquiridos na disciplina de Estruturas de Dados Avançadas. A utilização de estruturas de dados dinâmicas, notadamente listas ligadas, permitiu uma gestão eficiente e flexível dos registos, tanto para as antenas como para os efeitos. A modularização do código e a clara separação entre interface e implementação, juntamente com a documentação produzida com Doxygen, contribuem para a manutenção e compreensão do sistema, independentemente do nível de conhecimento do utilizador em linguagem C.

Na segunda fase, a incorporação de grafos como estrutura de dados permitiu modelar e analisar as relações complexas entre as antenas de forma mais sofisticada. A implementação de algoritmos como DFS e BFS, juntamente com funcionalidades para encontrar caminhos e detetar interseções, expandiu significativamente as capacidades do sistema, permitindo análises mais profundas e detalhadas da conectividade e interferência entre antenas. Esta evolução demonstra a importância dos grafos na resolução de problemas com múltiplas relações dinâmicas e prepara o sistema para futuras extensões e otimizações.

6 Referências

- Leitura e escrita de estruturas em ficheiros em C - GeeksforGeeks
- Função fwrite() em C - GeeksforGeeks
- Função fread() em C - GeeksforGeeks
- Listas Ligadas em C - GeeksforGeeks
- Alocação Dinâmica de Memória em C - GeeksforGeeks
- Função fopen() em C - GeeksforGeeks
- Graph and its representations - GeeksforGeeks
- Breadth First Search or BFS for a Graph - GeeksforGeeks
- Depth First Search or DFS for a Graph - GeeksforGeeks
- Difference between BFS and DFS - GeeksforGeeks
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.

7 Repositório GitHub

GitHub TrabalhoEDA