Uma imagem com Tipo de letra, texto, Gráficos, design gráfico

Os conteúdos gerados por IA poderão estar incorretos.

**Sistemas De Antenas**

Hugo Manuel Alves Gomes

Nº 31545 – Regime Pós-laboral

Ano letivo 2024/2025

Docente: Luís Gonzaga Martins Ferreira

Unidade Curricular - Estrutura de Dados Avançados

Licenciatura em Engenharia de Sistemas Informáticos

Escola Superior de Tecnologia

Instituto Politécnico do Cávado e do Ave

**Índice**

[Índice de figuras IV](#_Toc198562558)

[**Índice de Tabelas** IV](#_Toc198562559)

[Resumo V](#_Toc198562560)

[Glossário VI](#_Toc198562561)

[1. Introdução VII](#_Toc198562562)

[1.1 Motivação VII](#_Toc198562563)

[1.2 Enquadramento VIII](#_Toc198562564)

[1.3 Objetivos VIII](#_Toc198562565)

[1.4 Estrutura do documento VIII](#_Toc198562566)

[2. Análise e Especificação IX](#_Toc198562567)

[2.1 Requisitos IX](#_Toc198562568)

[2.2 Listas Ligadas IX](#_Toc198562569)

[2.3 Estruturas Definidas X](#_Toc198562570)

[2.4 Inserção de Antenas XI](#_Toc198562571)

[2.5 Cálculo de Efeito Nefasto XI](#_Toc198562572)

[2.6 Preservação dos ficheiros XIV](#_Toc198562573)

[2.7 Interação XIV](#_Toc198562574)

[3. Implementação XV](#_Toc198562575)

[3.1 Inserir Antenas XV](#_Toc198562576)

[3.2 Cálculo do Efeito Nefasto XVI](#_Toc198562577)

[3.3 Guardar o Estado em Ficheiro Binário XVII](#_Toc198562578)

[4. Análise e Discussão de Resultados XVIII](#_Toc198562579)

[5. Conclusão XIX](#_Toc198562580)

[6. Referências bibliográficas XX](#_Toc198562581)

[Índice de figuras 1](#_Toc198562582)

[**Índice de Tabelas** 1](#_Toc198562583)

[Resumo 2](#_Toc198562584)

[Glossário 3](#_Toc198562585)

[7. Introdução 4](#_Toc198562586)

[1.1 Motivação 4](#_Toc198562587)

[1.2 Enquadramento 5](#_Toc198562588)

[1.3 Objetivos 5](#_Toc198562589)

[1.4 Estrutura do documento 5](#_Toc198562590)

[8. Análise e Especificação 6](#_Toc198562591)

[8.1 Requisitos 6](#_Toc198562592)

[2.2 Listas Ligadas 6](#_Toc198562593)

[2.3 Grafos 7](#_Toc198562594)

[2.3.1 DFS 7](#_Toc198562595)

[2.3.2 BFS 7](#_Toc198562596)

[2.4 Estruturas Definidas 8](#_Toc198562597)

[2.5 Inserção de Antenas Ordenadas 10](#_Toc198562598)

[2.6 Preservação dos ficheiros 11](#_Toc198562599)

[2.7 Interação 11](#_Toc198562600)

[9. Implementação 12](#_Toc198562601)

[9.1 Inserir Antenas 12](#_Toc198562602)

[9.2 Cálculo do BFS e DFS 13](#_Toc198562603)

[9.3 Guardar o Estado em Ficheiro Binário 15](#_Toc198562604)

[10. Análise e Discussão de Resultados 16](#_Toc198562605)

[11. Conclusão 17](#_Toc198562606)

[12. Referências bibliográficas 18](#_Toc198562607)

# Índice de figuras

[Figura 1 Lista Ligada, retirado de (Georgescu, 2006). IX](#_Toc198562608)

[Figura 2 Estrutura Antena retirado de Visual Studio X](#_Toc198562609)

[Figura 3 Estrutura Efeito Nefasto retirado de Visual Studio X](#_Toc198562610)

[Figura 4 Inserção de nova antena XI](#_Toc198562611)

[Figura 5 Lista dos Pontos de igual frequência XII](#_Toc198562612)

[Figura 6 cálculos das coordenadas do Efeito Nefasto XIII](#_Toc198562613)

[Figura 7 Função inserir Antena retirada do Visual Studio Code XV](#_Toc198562614)

[Figura 8 Função calcularPontosEfeitoNefasto retirado de Visual Studio Code XVI](#_Toc198562615)

[Figura 9 Função salvarEstadoBinario retirado de Visual Studio Code XVII](#_Toc198562616)

[Figura 10 BFS e DFS 7](#_Toc198562617)

[Figura 11 Estrutura Antena retirado de Visual Studio 8](#_Toc198562618)

[Figura 12 Estrutura Adjacências retirado de Visual Studio 8](#_Toc198562619)

[Figura 13 Estrutura grafo retirado de Visual Studio 9](#_Toc198562620)

[Figura 14 Função inserir Antena retirada do Visual Studio Code 12](#_Toc198562621)

[Figura 15 DFS 13](#_Toc198562622)

[Figura 16 BFS 14](#_Toc198562623)

[Figura 18 Função SaveGrafoBin retirado de Visual Studio Code 15](#_Toc198562624)

# **Índice de Tabelas**

[Tabela 1 Análise e Discussão de Resultados XVIII](#_Toc198562625)

[Tabela 2 Análise e Discussão de Resultados 16](#_Toc198562626)

# Resumo

Nesta primeira fase do trabalho desenvolveu-se um programa em linguagem C responsável por gerenciar antenas de uma cidade, na qual antenas com a mesma frequência criam interferências entre si.

A solução de software desenvolvida permite identificar a localização em que a interferência ocorre, representando-as numa matriz de caracteres, que contêm as antenas (representadas por um caractere) que operam numa dada frequência.

Para a concretização deste projeto realizou-se a implementação de uma lista ligada simples, um tipo de Estruturas de Dados Dinâmicos, onde cada item está ligado unicamente ao próximo item, e que pode aumentar dinamicamente.

Construi-se um programa capaz de carregar e guardar dados de ficheiros texto e binário, com a possibilidade de inserir/remover antenas de forma dinâmica, que identifica de forma automática as localizações onde ocorrem as interferências e as demonstra através de uma matriz, para uma melhor perceção visual.

Este trabalho contém uma documentação realizada em Doxygen, que consiste numa ferramenta grátis de código aberto, que permite gerar documentação do código desenvolvido de forma automática em vários formatos de saída.

Com este trabalho solidificou-se conhecimentos adquiridos ao longo do semestre e identificou-se dúvidas na qual, sem a prática não seriam identificadas.

# Glossário

**Antenas** – Elementos que emitem ou recebem sinais, neste projeto estão representadas por letras numa matriz.

**binário (ficheiro)** – Ficheiro que armazena dados no formato binário, mais compacto e eficiente que texto.

**Doxygen** – Ferramenta gratuita e de código aberto usada para gerar automaticamente documentação a partir de comentários no código-fonte.

**Efeito nefasto** – Zona da matriz onde ocorre interferência devido à disposição e frequência das antenas.

**fopen** – Função da linguagem C utilizada para abrir ficheiros para leitura ou escrita.

**fwrite** – Função da linguagem C usada para escrever blocos de dados num ficheiro binário.

**GitHub** – Plataforma online para alojamento e controlo de versões de código, baseada em Git.

**Link** – Termo em programação que pode referir-se à ligação entre estruturas (ex. nós de listas ligadas) ou à ligação de dados na memória.

**Lista ligada** – Estrutura de dados onde cada elemento (nó) aponta para o seguinte, permitindo inserções e remoções dinâmicas.

**malloc** – Função da linguagem C que aloca dinamicamente memória durante a execução do programa.

**struct** – Palavra-chave em C utilizada para definir estruturas compostas por diferentes tipos de dados agrupados.

**Visual Studio Code** – Editor de código-fonte leve e multiplataforma, amplamente utilizado para desenvolvimento em várias linguagens.

# Introdução

No presente capítulo pretende-se contextualizar o trabalho desenvolvido, a motivação da sua origem, objetivos traçados, enquadramento académico e a estrutura geral do documento, de forma a fornecer uma visão clara do percurso realizado ao longo do projeto.

## 1.1 Motivação

A gestão eficiente de redes de telecomunicações em ambientes urbanos representa um desafio crescente, principalmente devido à elevada densidade de dispositivos emissores e recetores de sinal. Um dos principais problemas enfrentados nestes cenários consiste na interferência entre antenas, que pode ocorrer devido a fenómenos de ressonância e má distribuição de frequências.

Este problema propõe o desenvolvimento de uma solução de software que identifique zonas de interferência (localizações com efeito nefasto) numa cidade representada por uma matriz de caracteres, onde cada posição pode conter uma antena que opera numa determinada frequência (indicada por um caractere).

O efeito nefasto ocorre em pontos que estão perfeitamente alinhados com duas antenas da mesma frequência, desde que uma delas esteja ao dobro da distância da outra em relação ao local afetado.

Apesar da crescente necessidade de soluções automatizadas para a deteção e prevenção dessas interferências, ainda se observa alguma escassez de ferramentas capazes de simular e identificar, de forma precisa, os efeitos nefastos causados por determinadas configurações de antenas em cidades. A pertinência deste trabalho insere-se no contexto atual de evolução das tecnologias de comunicação, que exigem redes mais robustas e eficientes.

## 1.2 Enquadramento

Este trabalho foi desenvolvido no âmbito de um projeto de Estrutura de Dados Avançados do curso de Licenciatura de Sistemas Informáticos, com o objetivo de aplicar conhecimentos teóricos adquiridos durante o semestre para a resolução de um problema prático relacionado com a análise e planeamento de redes de telecomunicações.

## 1.3 Objetivos

Os principais objetivos deste projeto são:

* Desenvolver uma solução de software para identificar zonas de interferência entre antenas numa cidade modelada por uma matriz de caracteres;
* Implementar uma estrutura de dados eficiente para armazenar e manipular dinamicamente as posições das antenas;
* Carregar dados de um ficheiro de texto que define a matriz de antenas;
* Apresentar os resultados de forma clara, auxiliando na interpretação e planeamento da rede.

Neste projeto, desenvolveu-se uma estrutura de dados baseada em listas ligadas para armazenar as antenas e coordenadas, bem como algoritmos para:

## 1.4 Estrutura do documento

**Capítulo 1** - Introdução

**Capítulo 2** - Análise e Especificação

**Capítulo 3** - Implementação

**Capítulo 4** - Análise e Discussão de Resultados

**Capítulo 5** - Conclusão e Futuras Melhorias

**Capítulo 6** - Referências Bibliográficas

# Análise e Especificação

Neste capítulo pretende-se demonstrar os requisitos, arquiteturas lógicas/funcionais, modelação e interação necessários para o funcionamento do projeto desenvolvido.

## 2.1 Requisitos

Neste subcapítulo apresentam-se alguns dos requisitos necessários para a realização deste trabalho.

## 2.2 Listas Ligadas

Optou-se por utilizar listas ligadas simples devido à natureza linear do problema, quando á flexibilidade na alocação de memória e a eficiência e à necessidade de operações básicas de inserção, remoção.

“*One of the most common data structures is the singly linked list. A linked list is a collection of items accessible one after another beginning at the head and ending at the tail. There are implemented by each item having a link to the next item. The head is the first item of a list, and the tail is the last item of the list.*”(Georgescu, 2006).

Uma lista ligada entende-se como uma lista encadeada tal como a Figura 1 representa, na qual é um conjunto de itens que podem ser acedidos um após o outro, que começa na cabeça e que termina numa cauda. A cabeça e a cauda são o primeiro e o último item da lista respetivamente. Cada item encontra-se ligado ao próximo item através de um *link.* Tal como referido (Georgescu, 2006).

Uma imagem com texto, Tipo de letra, captura de ecrã, Retângulo

Os conteúdos gerados por IA poderão estar incorretos.

Figura 1 Lista Ligada, retirado de (Georgescu, 2006).

Neste projeto implementou-se várias listas ligadas para as diversas vertentes, como por exemplo para as Antenas e para os Efeitos Nefastos.

## 2.3 Estruturas Definidas

Para este trabalho implementou-se as seguintes estruturas:

Tal como a Figura 2 mostra que a Estrutura Antena contém os seguintes campos:

* Uma imagem com texto, captura de ecrã, Tipo de letra

  Os conteúdos gerados por IA poderão estar incorretos.Coordenadas da Antena (linha, coluna);
* Frequência;
* Apontador para a próxima Antena.

Figura 2 Estrutura Antena retirado de Visual Studio

Estrutura EfeitoNefasto representada na Figura 3 que contém os seguintes campos:

* Coordenadas do efeito nefasto (linha, coluna);
* Uma imagem com texto, captura de ecrã, Tipo de letra

  Os conteúdos gerados por IA poderão estar incorretos.Frequência originária do efeito nefasto (#);
* Apontador para o próximo EfeitoNefasto.

Figura 3 Estrutura Efeito Nefasto retirado de Visual Studio

## 2.4 Inserção de Antenas

As antenas neste software são inseridas sempre no início, e necessita-se de receber os seguintes campos:

* Frequência(caracter);
* Linha (coordenada correspondente na matriz);
* Coluna (coordenada correspondente na matriz).

O processo de inserção está representado na Figura 4.

Uma imagem com texto, escrita à mão, diagrama, file

Os conteúdos gerados por IA poderão estar incorretos.

Figura 4 Inserção de nova antena

## 2.5 Cálculo de Efeito Nefasto

Um dos principais problemas enfrentados nestes cenários consiste na interferência entre antenas, que pode ocorrer devido a fenómenos de ressonância e má distribuição de frequências.

O efeito nefasto ocorre em pontos que estão perfeitamente alinhados com duas antenas da mesma frequência, desde que uma delas esteja ao dobro da distância da outra em relação ao local afetado.

Para calcular-se o local onde ocorre o efeito nefasto representado pelo caracter ‘#’, é necessário saber as coordenadas das duas antenas da mesma frequência provocadoras dessa interferência.

Após o conhecimento das coordenadas dos pontos de igual frequência representada na Figura 5 são realizados os cálculos para determinar a localização dos efeitos nefastos.

Os cálculos tal como representado na Figura 6 são os seguintes:

* **delta\_linha** = coordenada da ***linha 2*** – ***linha 1***
* **delta\_coluna** = coordenadas da ***coluna 2 – coluna 1***

**Determinação dos Pontos de Interferência**:

* **Ponto 1**: Localizado na direção oposta à antena mais próxima

(linha1 - delta\_linha, col1 - delta\_coluna)

* **Ponto 2**: Localizado na extensão além da antena mais distante

(linha2 + delta\_linha, col2 + delta\_coluna)

Uma imagem com escrita à mão, texto, desenho

Os conteúdos gerados por IA poderão estar incorretos.

Figura 5 Lista dos Pontos de igual frequência

Uma imagem com texto, escrita à mão, número, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

Figura 6 cálculos das coordenadas do Efeito Nefasto

## 2.6 Preservação dos ficheiros

Optou-se pela utilização de ficheiros binários para preservação do estado do programa devido às seguintes vantagens fundamentais:

**Eficiência de Armazenamento:**

* Os ficheiros binários ocupam significativamente menos espaço em disco comparado com formato texto;
* Armazenam os dados exatamente como estão representados em memória, sem conversões ou formatação adicional.

**Performance Superior:**

* Operações de leitura/escrita são mais rápidas;
* Não há necessidade de conversão entre texto e tipos binários;
* Redução significativa no tempo de carregamento para conjuntos de dados grandes;
* Garantia de preservação exata do estado da memória;
* Eliminação de erros de arredondamento ou conversão que podem ocorrer em formatos texto.

**Integridade do Sistema:**

* Permite reconstrução perfeita do estado do programa entre execuções

Ideal para operações de backup e recuperação de estado

## 2.7 Interação

Uma das vertentes mais relevantes deste projeto é a interação entre o utilizador e o sistema, concretizada através de uma interface textual intuitiva. É disponibilizado um menu interativo com uma estrutura clara, que permite ao utilizador realizar operações como a inserção e remoção de antenas, bem como a manipulação e armazenamento dinâmico das suas posições.

Cada ação efetuada pelo utilizador é imediatamente acompanhada de uma mensagem informativa, indicando o sucesso ou erro da operação, o que garante um elevado nível de usabilidade. Além disso, os resultados das interações podem ser visualizados através de matrizes atualizadas, que refletem o estado atual do sistema de forma clara e fácil de interpretar, o que promove uma melhor compreensão dos dados e do impacto das ações realizadas.

# Implementação

Neste capítulo serão apresentadas as implementações das funções a nível do código desenvolvido.

## Inserir Antenas

A função de inserção de antenas, representada na **Figura 7**, é responsável por criar uma antena e adicioná-la à lista ligada que armazena todas as antenas existentes no sistema. Esta operação é essencial para garantir que as antenas possam ser manipuladas dinamicamente, de acordo com as ações do utilizador.

A função recebe como parâmetros:

* **cabeca**: ponteiro para o início da lista ligada de antenas;
* **frequencia**: carácter que representa a frequência de operação da antena;
* **linha** e **coluna**: coordenadas da antena na matriz.

Dentro da função, é feita a alocação dinâmica de memória para uma nova antena usando ***malloc***. Caso a alocação falhe, o programa é encerrado com uma mensagem de erro. Em seguida, os campos da estrutura Antena são preenchidos com os valores fornecidos.

Por fim, a nova antena é inserida no início da lista ligada, apontando para a antiga cabeça da lista, e o ponteiro para essa nova antena é retornado.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Os conteúdos gerados por IA poderão estar incorretos. Este método de inserção no início da lista é eficiente em termos de complexidade temporal, pois não requer percorrer a lista, e é apropriado para cenários onde a ordem de inserção não é relevante.

Figura 7 Função inserir Antena retirada do Visual Studio Code

## 3.2 Cálculo do Efeito Nefasto

**Uma imagem com texto, captura de ecrã, software, Software de multimédia

Os conteúdos gerados por IA poderão estar incorretos.** O cálculo das localizações com efeito nefasto é baseado na interferência entre antenas da mesma frequência, seguindo um critério geométrico preciso.

Figura 8 Função calcularPontosEfeitoNefasto retirado de Visual Studio Code

**1ºParâmetros de Entrada (fig.3 linha do código 2 e 3):**

* **linha1**, **col1**: Coordenadas (linha, coluna) da primeira antena;
* **linha2**, **col2**: Coordenadas (linha, coluna) da segunda antena.

**Processamento/Cálculo dos Deltas (linha do código 8 e 11):**

**delta\_linha** = ***linha2*** - ***linha1*** (diferença horizontal entre antenas);

**delta\_coluna** = ***col2*** - ***col1*** (diferença vertical entre antenas).

**2ºDeterminação dos Pontos de Interferência**:

**Ponto 1**: Localizado na direção oposta à antena mais próxima

(linha1 - delta\_linha, col1 - delta\_coluna)

**Ponto 2**: Localizado na extensão além da antena mais distante

(linha2 + delta\_linha, col2 + delta\_coluna)

**3ºEstruturação dos Resultados**:

Os pontos são armazenados numa lista ligada, com o Ponto 2 como cabeça da lista.

## 3.3 Guardar o Estado em Ficheiro Binário

A função **salvarEstadoBinario**, apresentada na Figura 9, tem como objetivo guardar o estado atual da lista de antenas num ficheiro binário, de forma a preservar os dados para uso futuro ou para análise posterior. Para esta realização consultou-se um documento sobre os arquivos binários. Clique ou toque aqui para introduzir texto.

**Funcionamento:**

A função verifica se a lista de antenas está vazia (cabeca == NULL). Se estiver, retorna false, indicando que não há nada a guardar. Abre o ficheiro em modo binário de escrita ("wb”) (linha 4). Se a abertura falhar, retorna novamente false. Conta o número total de antenas na lista, utilizando um ciclo **for**. Escreve esse número no início do ficheiro, para que, no futuro, ao carregar os dados, se saiba quantas antenas devem ser lidas.

Percorre a lista e guarda individualmente os dados de cada antena:

* Frequência (como char);
* Linha (como int);
* Coluna (como int).

Uma imagem com texto, captura de ecrã, software, ecrã

Os conteúdos gerados por IA poderão estar incorretos.Fecha o ficheiro e retorna **true**, indicando sucesso.

Figura 9 Função salvarEstadoBinario retirado de Visual Studio Code

# Análise e Discussão de Resultados

Este capítulo tem como objetivo avaliar os resultados obtidos ao longo do desenvolvimento do projeto, com base nos objetivos inicialmente propostos. Será apresentada uma análise sobre as funcionalidades implementadas, os desafios enfrentados e as limitações encontradas durante o processo de desenvolvimento.

Para facilitar a compreensão, representou-se na Tabela 1 a indicação clara das funcionalidades que foram concretizadas com sucesso, bem como aquelas que, por motivos técnicos ou de tempo, não foram totalmente desenvolvidas ou ficaram em aberto para trabalho futuro.

Esta análise permite não só validar o grau de cumprimento dos objetivos definidos, como também refletir sobre possíveis melhorias e evoluções da solução apresentada.

|  |  |  |
| --- | --- | --- |
| Funções | SIM | NÃO |
| Inserir No Inicio | X |  |
| Inserir De Forma Ordenada |  | X |
| Remove Antena | X |  |
| Calcular Efeito Nefasto | X |  |
| Mostrar Matriz de Antenas | X |  |
| Mostrar Matriz de Antenas e Nefasto | X |  |
| Tabela Efeito Nefasto | X |  |
| Carregar Ficheiro .TXT | X |  |
| Exportar Ficheiro .TXT | X |  |
| Gravar Ficheiro Binário | X |  |
| Carregar Ficheiro Binário | X |  |
| Documentação em Doxygen  (Doxygen, n.d.) | X |  |
| Exportação | X |  |

Tabela 1 Análise e Discussão de Resultados

# Conclusão

O desenvolvimento deste projeto permitiu a criação de uma solução eficaz para a identificação de zonas de interferência em redes de telecomunicações, representadas numa matriz. Ao longo do trabalho, foram aplicados conhecimentos essenciais sobre estruturas de dados, nomeadamente listas ligadas, bem como técnicas de leitura e escrita de ficheiros, gestão dinâmica de memória e interação com o utilizador através de uma interface textual simples e funcional.

As funcionalidades principais como a inserção e remoção de antenas, a deteção automática de interferências e o armazenamento do estado do sistema em ficheiros binários foram implementadas com sucesso. O menu interativo revelou-se eficaz, ao entregar uma utilização fluída e intuitiva para o utilizador.

Apesar dos objetivos alcançados, identificaram-se possibilidades de melhoria, tais como:

* Reorganização da estrutura de saída: transferir todas as instruções ***printf*** para a função **main()**, de forma a centralizar a apresentação de mensagens e a tornar o código mais modular e legível;
* Inserção ordenada das antenas: alterar a lógica de inserção de antenas na lista ligada, deixando de inserir no início da lista para passar a inseri-las de forma ordenada por coordenadas, o que irá facilitar as futuras pesquisas e otimizações no processamento dos dados;
* Melhoria na escrita binária: na função de gravação em ficheiro binário, deixar de passar apenas o endereço do campo frequência (&a->frequência) e passar diretamente o ponteiro para a antena (a), o que resultará num código mais limpo, genérico e fácil de manter.

Em suma, este projeto constituiu uma experiência prática enriquecedora, que permitiu consolidar os conhecimentos adquiridos ao longo do semestre, enquanto abre portas para evoluções futuras da solução aqui apresentada.

GitHub Pessoal:

O trabalho realizado encontra-se disponível no ***Github*** e pode ser acedido através do seguinte link:

* https://github.com/hugo31545/-Projeto-EDA-LESI2025

# Referências bibliográficas

Neste capítulo apresentam-se as fontes de informação consultadas e utilizadas ao longo do desenvolvimento do presente trabalho. As referências incluem materiais universitários, documentação técnica e conteúdos online que serviram de apoio à implementação, compreensão e fundamentação teórica do projeto. Parte superior do formulário

Parte inferior do formulário

*Arquivos binários*. (n.d.).

Doxygen. (n.d.). *Doxygen Documentation: Special Commands and Formatting*. Retrieved March 27, 2025, from https://www.doxygen.nl/manual/docblocks.html

Georgescu, I. (2006). Rationality of Fuzzy Choice Functions Through Indicators Article in. In *International Journal of Computers*. https://www.researchgate.net/publication/31597536

Uma imagem com Tipo de letra, texto, Gráficos, design gráfico

Os conteúdos gerados por IA poderão estar incorretos.

**Sistemas De Antenas Grafos – Fase 2**

Hugo Manuel Alves Gomes

Nº 31545– Regime Pós-laboral

Ano letivo 2024/2025

Docente: Luís Gonzaga Martins Ferreira

Unidade Curricular - Estrutura de Dados Avançados

Licenciatura em Engenharia de Sistemas Informáticos

Escola Superior de Tecnologia

Instituto Politécnico do Cávado e do Ave

**Índice**

[Índice de figuras IV](#_Toc198562627)

[**Índice de Tabelas** IV](#_Toc198562628)

[Resumo V](#_Toc198562629)

[Glossário VI](#_Toc198562630)

[1. Introdução VII](#_Toc198562631)

[1.1 Motivação VII](#_Toc198562632)

[1.2 Enquadramento VIII](#_Toc198562633)

[1.3 Objetivos VIII](#_Toc198562634)

[1.4 Estrutura do documento VIII](#_Toc198562635)

[2. Análise e Especificação IX](#_Toc198562636)

[2.1 Requisitos IX](#_Toc198562637)

[2.2 Listas Ligadas IX](#_Toc198562638)

[2.3 Estruturas Definidas X](#_Toc198562639)

[2.4 Inserção de Antenas XI](#_Toc198562640)

[2.5 Cálculo de Efeito Nefasto XI](#_Toc198562641)

[2.6 Preservação dos ficheiros XIV](#_Toc198562642)

[2.7 Interação XIV](#_Toc198562643)

[3. Implementação XV](#_Toc198562644)

[3.1 Inserir Antenas XV](#_Toc198562645)

[3.2 Cálculo do Efeito Nefasto XVI](#_Toc198562646)

[3.3 Guardar o Estado em Ficheiro Binário XVII](#_Toc198562647)

[4. Análise e Discussão de Resultados XVIII](#_Toc198562648)

[5. Conclusão XIX](#_Toc198562649)

[6. Referências bibliográficas XX](#_Toc198562650)

[Índice de figuras 1](#_Toc198562651)

[**Índice de Tabelas** 1](#_Toc198562652)

[Resumo 2](#_Toc198562653)

[Glossário 3](#_Toc198562654)

[7. Introdução 4](#_Toc198562655)

[1.1 Motivação 4](#_Toc198562656)

[1.2 Enquadramento 5](#_Toc198562657)

[1.3 Objetivos 5](#_Toc198562658)

[1.4 Estrutura do documento 5](#_Toc198562659)

[8. Análise e Especificação 6](#_Toc198562660)

[8.1 Requisitos 6](#_Toc198562661)

[2.2 Listas Ligadas 6](#_Toc198562662)

[2.3 Grafos 7](#_Toc198562663)

[2.3.1 DFS 7](#_Toc198562664)

[2.3.2 BFS 7](#_Toc198562665)

[2.4 Estruturas Definidas 8](#_Toc198562666)

[2.5 Inserção de Antenas Ordenadas 10](#_Toc198562667)

[2.6 Preservação dos ficheiros 11](#_Toc198562668)

[2.7 Interação 11](#_Toc198562669)

[9. Implementação 12](#_Toc198562670)

[9.1 Inserir Antenas 12](#_Toc198562671)

[9.2 Cálculo do BFS e DFS 13](#_Toc198562672)

[9.3 Guardar o Estado em Ficheiro Binário 15](#_Toc198562673)

[10. Análise e Discussão de Resultados 16](#_Toc198562674)

[11. Conclusão 17](#_Toc198562675)

[12. Referências bibliográficas 18](#_Toc198562676)

# Índice de figuras

[Figura 1 Lista Ligada, retirado de (Georgescu, 2006). IX](#_Toc198562677)

[Figura 2 Estrutura Antena retirado de Visual Studio X](#_Toc198562678)

[Figura 3 Estrutura Efeito Nefasto retirado de Visual Studio X](#_Toc198562679)

[Figura 4 Inserção de nova antena XI](#_Toc198562680)

[Figura 5 Lista dos Pontos de igual frequência XII](#_Toc198562681)

[Figura 6 cálculos das coordenadas do Efeito Nefasto XIII](#_Toc198562682)

[Figura 7 Função inserir Antena retirada do Visual Studio Code XV](#_Toc198562683)

[Figura 8 Função calcularPontosEfeitoNefasto retirado de Visual Studio Code XVI](#_Toc198562684)

[Figura 9 Função salvarEstadoBinario retirado de Visual Studio Code XVII](#_Toc198562685)

[Figura 10 BFS e DFS 7](#_Toc198562686)

[Figura 11 Estrutura Antena retirado de Visual Studio 8](#_Toc198562687)

[Figura 12 Estrutura Adjacências retirado de Visual Studio 8](#_Toc198562688)

[Figura 13 Estrutura grafo retirado de Visual Studio 9](#_Toc198562689)

[Figura 14 Função inserir Antena retirada do Visual Studio Code 12](#_Toc198562690)

[Figura 15 DFS 13](#_Toc198562691)

[Figura 16 BFS 14](#_Toc198562692)

[Figura 18 Função SaveGrafoBin retirado de Visual Studio Code 15](#_Toc198562693)

# **Índice de Tabelas**

[Tabela 1 Análise e Discussão de Resultados XVIII](#_Toc198562694)

[Tabela 2 Análise e Discussão de Resultados 16](#_Toc198562695)

# Resumo

Na 2º fase do projeto, aplicaram-se conceitos de teoria de grafos e programação em C para resolver o problema da interligação entre antenas. As principais funcionalidades implementadas foram:

* **Estrutura de Dados GR -** Definição de um tipo de dados GR para representar grafos com um número arbitrário de vértices. Cada vértice corresponde a uma antena, com posição e frequência de ressonância. As arestas ligam apenas antenas com a mesma frequência.
* **Leitura de Dados a partir de Ficheiro -** Implementação da funcionalidade de carregamento de dados de antenas a partir de um ficheiro de texto com um formato pré-definido, que cria automaticamente o grafo correspondente.
* **Operações sobre o Grafo:**
  + **Procura em Profundidade (DFS):** Listagem das antenas alcançáveis a partir de uma antena inicial.
  + **Procura em Largura (BFS):** Listagem das antenas alcançáveis em largura a partir de uma antena inicial.
  + **Caminhos Possíveis:** Identificação de todos os caminhos possíveis entre duas antenas específicas.
  + **Interseções entre Frequências:** Listagem das coordenadas de antenas com frequências diferentes (A e B) que se encontram na mesma posição.

# Glossário

**Antenas** – Elementos que emitem ou recebem sinais, neste projeto estão representadas por letras numa matriz.

**binário (ficheiro)** – Ficheiro que armazena dados no formato binário, mais compacto e eficiente que texto.

**Doxygen** – Ferramenta gratuita e de código aberto usada para gerar automaticamente documentação a partir de comentários no código-fonte.

**fopen** – Função da linguagem C utilizada para abrir ficheiros para leitura ou escrita.

**fwrite** – Função da linguagem C usada para escrever blocos de dados num ficheiro binário.

**Lista ligada** – Estrutura de dados onde cada elemento (nó) aponta para o seguinte, permitindo inserções e remoções dinâmicas.

**malloc** – Função da linguagem C que aloca dinamicamente memória durante a execução do programa.

**struct** – Palavra-chave em C utilizada para definir estruturas compostas por diferentes tipos de dados agrupados.

**Visual Studio Code** – Editor de código-fonte leve e multiplataforma, amplamente utilizado para desenvolvimento em várias linguagens.

**Grafo** – conjunto de estruturas conectadas entre si.

# Introdução

No presente capítulo pretende-se contextualizar o trabalho desenvolvido, a motivação da sua origem, objetivos traçados, enquadramento académico e a estrutura geral do documento, de forma a fornecer uma visão clara do percurso realizado ao longo do projeto.

## 1.1 Motivação

Num mundo cada vez mais interligado e dependente da comunicação em tempo real, a gestão eficiente de redes de transmissão de dados torna-se essencial. As antenas de comunicação, enquanto elementos centrais dessas redes, desempenham um papel fundamental na cobertura e estabilidade dos sistemas de telecomunicações.

Este projeto surge com o objetivo de aprofundar o conhecimento sobre estruturas de dados avançados à representação e manipulação de redes de antenas, através da construção de um programa capaz de carregar, representar graficamente e analisar conexões entre antenas. A implementação de funcionalidades como a leitura de ficheiros, a criação de grafos, a gestão de adjacências e a procura de caminhos entre nós permite consolidar competências em programação em linguagem C, bem como compreender melhor os desafios associados à modelação de sistemas reais.

Para além disso, esta aplicação pode servir de base para projetos futuros que envolvam a otimização de redes ou a simulação de cobertura em sistemas de comunicações.

## 1.2 Enquadramento

Este trabalho foi desenvolvido no âmbito de um projeto de Estrutura de Dados Avançados do curso de Licenciatura de Engenharia de Sistemas Informáticos, com o objetivo de aplicar conhecimentos teóricos adquiridos durante o semestre para a resolução de um problema prático relacionado com a análise e planeamento de grafos.

## 1.3 Objetivos

Os principais objetivos deste projeto são:

* **Procura em Profundidade (DFS):** Listagem das antenas alcançáveis a partir de uma antena inicial;
* **Procura em Largura (BFS):** Listagem das antenas alcançáveis em largura a partir de uma antena inicial;
* **Caminhos Possíveis:** Identificação de todos os caminhos possíveis entre duas antenas específicas;
* **Intersecções entre Frequências:** Listagem das coordenadas de antenas com frequências diferentes (A e B) que se encontram na mesma posição.

Neste projeto, desenvolveu-se uma estrutura de dados baseada em listas ligadas.

## 1.4 Estrutura do documento

**Capítulo 7** - Introdução

**Capítulo 8** - Análise e Especificação

**Capítulo 9** - Implementação

**Capítulo 10** - Análise e Discussão de Resultados

**Capítulo 11** - Conclusão e Futuras Melhorias

**Capítulo 12** - Referências Bibliográficas

# Análise e Especificação

Neste capítulo pretende-se demonstrar os requisitos, arquiteturas lógicas/funcionais, modelação e interação necessários para o funcionamento do projeto desenvolvido.

## 8.1 Requisitos

Neste subcapítulo apresentam-se alguns dos requisitos necessários para a realização deste trabalho.

## 2.2 Listas Ligadas

Optou-se por utilizar listas ligadas simples devido à natureza linear do problema, quando á flexibilidade na alocação de memória e a eficiência e à necessidade de operações básicas de inserção, remoção.

“*One of the most common data structures is the singly linked list. A linked list is a collection of items accessible one after another beginning at the head and ending at the tail. There are implemented by each item having a link to the next item. The head is the first item of a list, and the tail is the last item of the list.*”(Georgescu, 2006).

Uma lista ligada entende-se como uma lista encadeada tal como a Figura representa, na qual é um conjunto de itens que podem ser acedidos um após o outro, que começa na cabeça e que termina numa cauda. A cabeça e a cauda são o primeiro e o último item da lista respetivamente. Cada item encontra-se ligado ao próximo item através de um *link.* Tal como referido (Georgescu, 2006).

Uma imagem com texto, Tipo de letra, captura de ecrã, Retângulo

Os conteúdos gerados por IA poderão estar incorretos.

Figura 1 Lista Ligada, retirado de (Georgescu, 2006).

Neste projeto implementou-se várias listas ligadas para as diversas vertentes, como por exemplo para as Antenas de diferentes frequências.

## 2.3 Grafos

Grafos são estruturas de dados que representam conjuntos de elementos chamados vértices (ou nós) e as conexões entre eles, chamam-se de arestas. Eles podem ser direcionados (com setas indicando a direção da conexão) ou não direcionados (grafo que se implementou nesta fase), e também podem ter pesos associados às arestas, como distâncias ou custos.

Grafos são usados para modelar situações como redes sociais, mapas de estradas, redes de computadores, entre outros. Alguns exemplos de algoritmos que trabalham com grafos incluem a procura em profundidade (DFS), procura em largura (BFS) (Figura 10) e o algoritmo de Dijkstra.

### 2.3.1 DFS

**DFS (Depth-First Search)** – procura em Profundidade:  
Vai ao caminha mais longo antes de voltar e explorar outros.  
Usa recursão para controlar a visita.  
É útil para verificar conexões, ciclos e componentes em grafos.

### 2.3.2 BFS

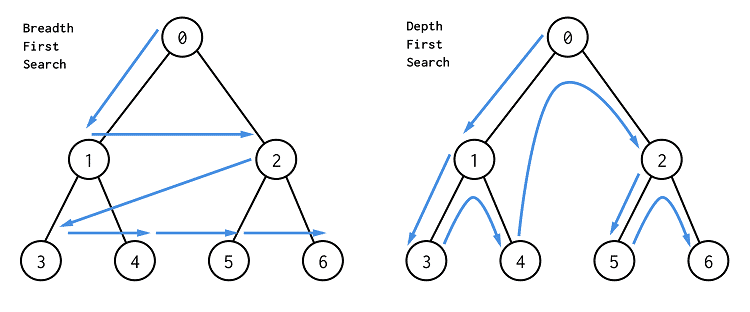
 **BFS (Breadth-First Search)** – procura em Largura:  
Explora o grafo em camadas, visita primeiro todos os vizinhos de um vértice antes de ir para os vizinhos dos vizinhos.  
É ideal para encontrar o caminho mais curto em grafos não ponderados.

Figura 10 BFS e DFS

## 2.4 Estruturas Definidas

**Estrutura Antena**

Na Figura 11 está representada a Estrutura Antena e contém os seguintes campos:

* Coordenadas da Antena (linha, coluna);
* Frequência;
* Campo para controlo de algoritmos de travessia.
* Lista de adjacências com outras antenas da **mesma frequência**.

Uma imagem com texto, Tipo de letra, captura de ecrã, número

Os conteúdos gerados por IA poderão estar incorretos.

Figura 11 Estrutura Antena retirado de Visual Studio

**Importância:**

É a estrutura central do sistema, pois cada antena é um vértice no grafo, e seu comportamento determina a conectividade e análise de frequência.

**Estrutura das Adjacências**

Esta estrutura (Figura 12) define a relação de adjacência entre antenas **da mesma frequência**, representando as arestas do grafo.

**Componentes:**

* destino: Aponta para uma antena que é adjacente à antena origem.
* proximo: Permite a criação de uma lista ligada de adjacências.

Uma imagem com texto, Tipo de letra, captura de ecrã

Os conteúdos gerados por IA poderão estar incorretos.

Figura 12 Estrutura Adjacências retirado de Visual Studio

**Estrutura do Grafo**

Esta estrutura (Figura 13Figura 12) representa o grafo como um todo, tendo como entrada a cabeça da lista ligada de antenas.

Uma imagem com texto, Tipo de letra, captura de ecrã, tipografia

Os conteúdos gerados por IA poderão estar incorretos.

Figura 13 Estrutura grafo retirado de Visual Studio

**Componentes:**

* cabeca: Aponta para o início da lista de antenas.

**Importância:**

Serve como ponto de partida para percorrer todas as antenas do sistema, essencial para inicializações e algoritmos de busca.

## 2.5 Inserção de Antenas Ordenadas

A inserção das antenas é feita de forma ordenada, organizando as antenas na lista conforme suas coordenadas para facilitar o gerenciamento. Cada antena conecta-se somente a outras antenas que possuem a mesma frequência e tipo.

Diferentemente de modelos orientados, as ligações aqui são não orientadas, ou seja, as conexões entre antenas são bidirecionais. Por exemplo, se a antena A1 está ligada à antena A2, então A2 também está ligada a A1, garantindo comunicação em ambos os sentidos.

Para cada antena inserida, são estabelecidas conexões com antenas do mesmo tipo e frequência dentro de um alcance aceitável, armazenadas numa lista de adjacência. Essa lista é organizada por tipo de antena, mantendo grupos separados para evitar interferências entre diferentes categorias.

Este modelo representa situações práticas em que dispositivos que compartilham a mesma frequência podem se comunicar diretamente, enquanto aqueles de frequências distintas permanecem isolados. A estrutura bidirecional permite o uso eficiente de algoritmos de busca e análise, como BFS e DFS, para explorar as conexões da rede.

## 2.6 Preservação dos ficheiros

Optou-se pela utilização de ficheiros binários para preservação do estado do programa devido às seguintes vantagens fundamentais:

**Eficiência de Armazenamento:**

* Os ficheiros binários ocupam significativamente menos espaço em disco comparado com formato texto;
* Armazenam os dados exatamente como estão representados em memória, sem conversões ou formatação adicional.

**Performance Superior:**

* Operações de leitura/escrita são mais rápidas;
* Não há necessidade de conversão entre texto e tipos binários;
* Redução significativa no tempo de carregamento para conjuntos de dados grandes;
* Garantia de preservação exata do estado da memória;
* Eliminação de erros de arredondamento ou conversão que podem ocorrer em formatos texto.

**Integridade do Sistema:**

* Permite reconstrução perfeita do estado do programa entre execuções

Ideal para operações de backup e recuperação de estado

## 2.7 Interação

Uma das vertentes mais relevantes deste projeto é a interação entre o utilizador e o sistema, concretizada através de uma interface textual intuitiva. É disponibilizado um menu interativo com uma estrutura clara, que permite ao utilizador realizar operações como a inserção e remoção de antenas, bem como a manipulação e armazenamento dinâmico das suas posições.

Cada ação efetuada pelo utilizador é imediatamente acompanhada de uma mensagem informativa, indicando o sucesso ou erro da operação, o que garante um elevado nível de usabilidade. Além disso, os resultados das interações podem ser visualizados através de matrizes atualizadas, que refletem o estado atual do sistema de forma clara e fácil de interpretar, o que promove uma melhor compreensão dos dados e do impacto das ações realizadas.

# Implementação

Neste capítulo serão apresentadas as implementações das funções a nível do código desenvolvido.

## Inserir Antenas

A função de inserção de antenas, representada na Figura 14, é responsável por criar uma antena e adicioná-la à lista ligada que armazena todas as antenas existentes no sistema. Esta operação é essencial para garantir que as antenas possam ser manipuladas dinamicamente, de acordo com as ações do utilizador.

A função recebe como parâmetros:

* **cabeca**: ponteiro para o início da lista ligada de antenas;
* **frequencia**: carácter que representa a frequência de operação da antena;
* **linha** e **coluna**: coordenadas da antena na matriz.

A alocação dinâmica de memória para uma nova antena usando ***malloc*** é realizada na função *criar\_Antena*. Caso a alocação falhe, o programa é encerrado com uma mensagem de erro. Em seguida, os campos da estrutura Antena são preenchidos com os valores fornecidos.

Uma imagem com texto, captura de ecrã, software

Os conteúdos gerados por IA poderão estar incorretos.Uma imagem com texto, captura de ecrã, Tipo de letra

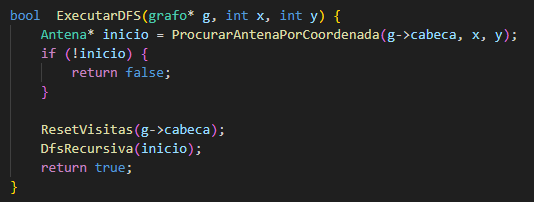
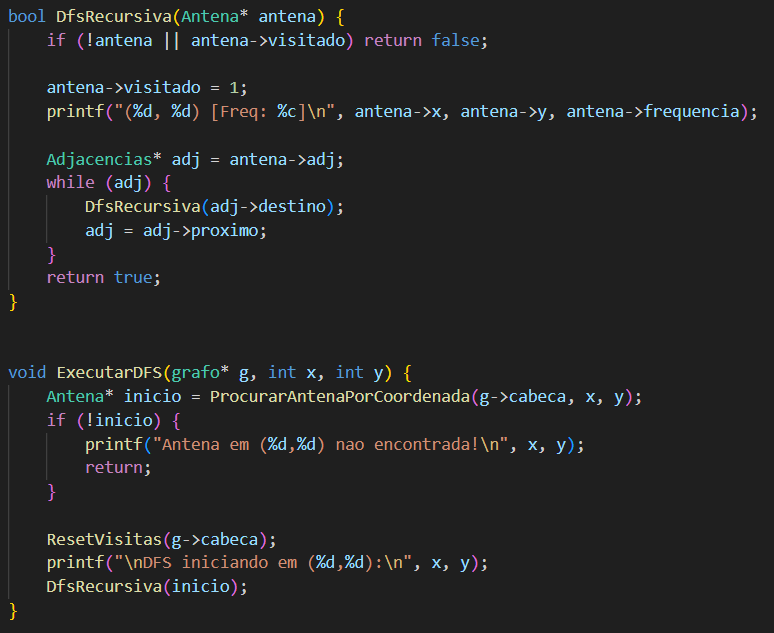
Os conteúdos gerados por IA poderão estar incorretos.

Figura 14 Função inserir Antena retirada do Visual Studio Code

## Cálculo do BFS e DFS

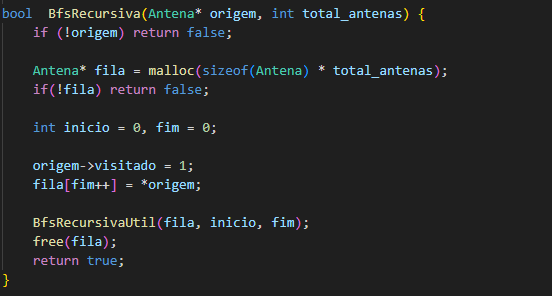
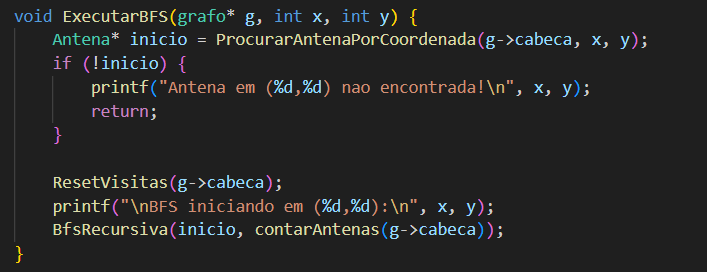
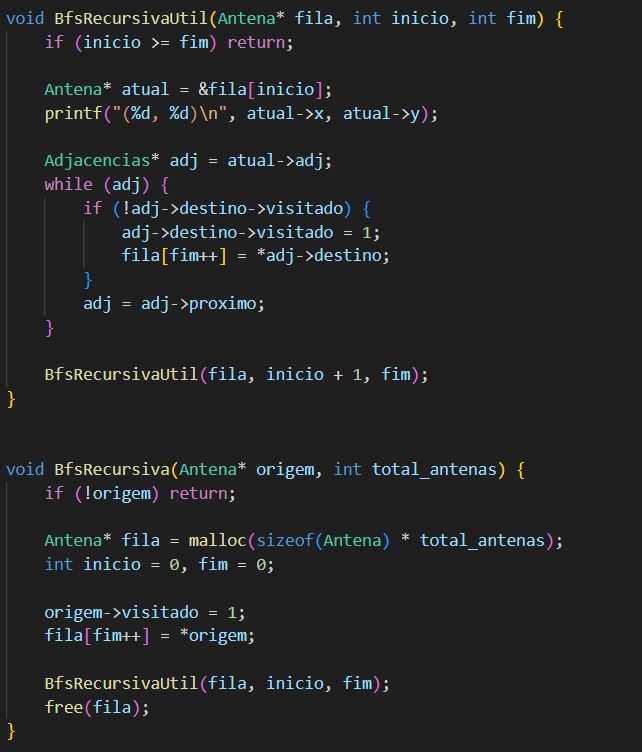
DFS (procura em Profundidade): Explora os caminhos o mais fundo possível, indo para os vizinhos recursivamente. Marca como visitado e volta quando não há mais vizinhos. O código desenvolvido para essa função encontra-se na Figura 15.

Figura 15 DFS



BFS (procura em Largura): Explora os vizinhos primeiro, através de uma fila para visitar todos em "camadas". São adicionados os próximos vizinhos à fila e visitando em ordem. O código desenvolvido para essa função encontra-se na Figura 16.

Figura 16 BFS



## Guardar o Estado em Ficheiro Binário

**Objetivo**: Salvar um grafo (com antenas e as suas conexões) em um arquivo binário.

**Verificações iniciais:**

Confirma se o nome do arquivo e o grafo são válidos. Em seguida, tenta abrir o arquivo em modo binário de escrita ("wb").

**Cabeçalho**: Escreve uma sequência fixa de 4 caracteres ("ANTG") no início do arquivo como identificador.

**Quantidade de antenas**: Conta o número total de antenas no grafo e grava esse valor como um inteiro.

**Dados das antenas**: Para cada antena, grava as coordenadas (x, y) e a frequência associada (caractere).

**Conexões (adjacências):**

**1.Para cada antena:**

* Conta quantas conexões (adjacências) ela possui.
* Grava essa quantidade.
* Percorre cada conexão e grava o índice da antena de destino, baseado na posição dela na lista principal.

**Finalização**: Fecha o arquivo e retorna *true* se tudo for salvo corretamente, ou *false* em caso de erro.

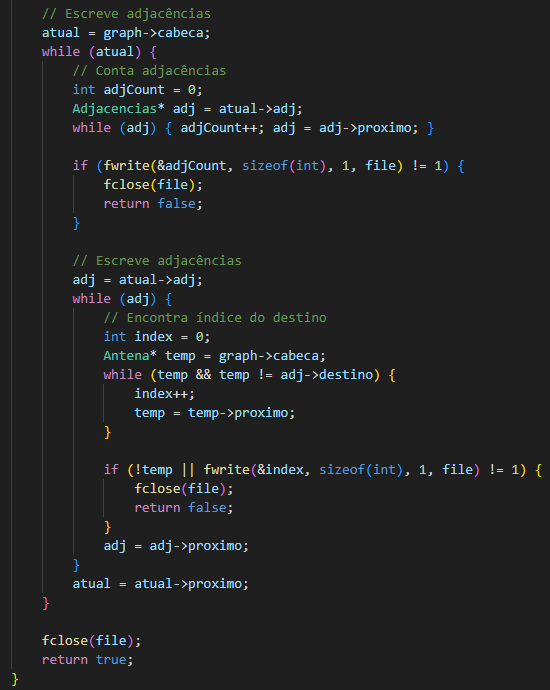
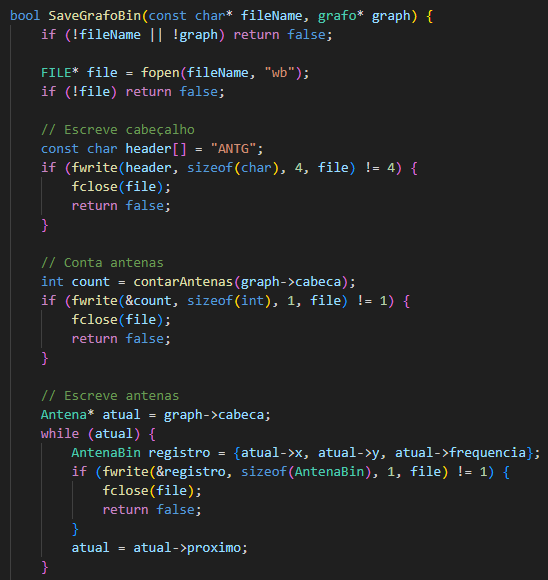


Figura 18 Função SaveGrafoBin retirado de Visual Studio Code

# Análise e Discussão de Resultados

Este capítulo tem como objetivo avaliar os resultados obtidos ao longo do desenvolvimento do projeto, com base nos objetivos inicialmente propostos. Será apresentada uma análise sobre as funcionalidades implementadas, os desafios enfrentados e as limitações encontradas durante o processo de desenvolvimento.

Para facilitar a compreensão, representou-se na Tabela 1 a indicação clara das funcionalidades que foram concretizadas com sucesso, bem como aquelas que, por motivos técnicos ou de tempo, não foram totalmente desenvolvidas ou ficaram em aberto para trabalho futuro.

Esta análise permite não só validar o grau de cumprimento dos objetivos definidos, como também refletir sobre possíveis melhorias e evoluções da solução apresentada.

|  |  |  |
| --- | --- | --- |
| Funções | SIM | NÃO |
| Carregar arquivo de antenas | X |  |
| Criar adjacências | x |  |
| DFS (procura em Profundidade) | X |  |
| BFS (procura em Largura) | X |  |
| Interseções entre frequências | X |  |
| Todos os caminhos entre duas antenas | X |  |
| Criar antena manualmente | X |  |
| Remover antena | X |  |
| Listar grafo | X |  |
| Guardar arquivo binário | X |  |
| Guardar arquivo texto | X |  |
| Carregar arquivo binário |  | x |

Tabela 2 Análise e Discussão de Resultados

# Conclusão

O projeto apresentado teve como objetivo a construção de uma estrutura de dados eficiente para representar antenas com diferentes frequências, permitindo a execução de algoritmos de procura, bem como a leitura e escrita em arquivos texto e binários. A base do sistema é estruturada a partir de uma lista encadeada de antenas, cada uma contendo sua posição, frequência e lista de adjacências, representando conexões com antenas vizinhas da mesma frequência.

Foram desenvolvidas funções essenciais para a criação, inserção ordenada, procura e remoção de antenas, além da construção automática das adjacências entre antenas com base na proximidade e compatibilidade de frequência. A manipulação de arquivos foi implementada tanto em formato texto quanto binário, permitindo a persistência dos dados. O formato binário desenvolvido inclui um cabeçalho identificador, os dados das antenas e suas respetivas conexões, promovendo leitura e gravação otimizadas.

Além disso, foram implementados algoritmos de procura em profundidade (DFS) e procura em largura (BFS), adaptados para percorrer o grafo considerando apenas conexões válidas (frequência igual). Esses algoritmos são fundamentais para verificar conectividade, detetar agrupamentos e explorar caminhos possíveis entre antenas. Também foi proposta uma função para detetar todos os caminhos possíveis entre dois nós, útil para análises mais complexas do grafo.

Por fim, a modularidade do código e a separação clara das responsabilidades entre criação, manipulação, persistência e algoritmos de procura tornam o sistema escalável e de fácil manutenção.

**GitHub Pessoal:**

O trabalho realizado encontra-se disponível no ***Github*** e pode ser acedido através do seguinte link:

# Referências bibliográficas

Neste capítulo apresentam-se as fontes de informação consultadas e utilizadas ao longo do desenvolvimento do presente trabalho. As referências incluem materiais universitários, documentação técnica e conteúdos online que serviram de apoio à implementação, compreensão e fundamentação teórica do projeto. Parte superior do formulário

Parte inferior do formulário

*Arquivos binários*. (n.d.).

Doxygen. (n.d.). *Doxygen Documentation: Special Commands and Formatting*. Retrieved March 27, 2025, from <https://www.doxygen.nl/manual/docblocks.html>

Georgescu, I. (2006). Rationality of Fuzzy Choice Functions Through Indicators Article in. In *International Journal of Computers*. <https://www.researchgate.net/publication/31597536>

Linkdin: <https://www.linkedin.com/pulse/o-que-%C3%A9-e-quais-diferen%C3%A7as-de-dfs-bfs-em-grafos-jo%C3%A3o-vitor-proen%C3%A7a-ewfof/>

geeksforgeeks: <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

geeksforgeeks: <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

hackerearth: <https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/>