



Projeto de Avaliação Estrutura de Dados Avançadas

João Ricardo Costa Pereira

Nº 31505 – Regime Diurno

Ano letivo 2024/2025

Licenciatura em Engenharia de Sistemas Informáticos

Escola Superior de Tecnologia

Instituto Politécnico do Cávado e do Ave

RESUMO

O projeto visou a implementação de uma solução para a gestão e análise da distribuição de antenas numa cidade, utilizando estruturas de dados dinâmicas. Na primeira fase, foi desenvolvida uma abordagem baseada em listas ligadas para armazenar e manipular informações das antenas, permitindo a identificação e representação das localizações com efeito nefasto.

LISTA DE ABREVIATURAS E/OU SIGLAS

Estruturas de Dados Avançadas (EDA)

Unidade Curricular (UC)

DepthFirst Search (DFS)

Breadth-First Search (BFS)

ÍNDICE

1. INTRODUÇÃO.....	4
1.1. MOTIVAÇÃO	5
1.2. ENQUADRAMENTO	6
1.3. OBJETIVOS	6
1.4. METODOLOGIA DE TRABALHO	7
1.5. PLANO DE TRABALHO	8
1.6. ESTRUTURA DO DOCUMENTO	11
2. ESTADO DA ARTE/ REVISÃO DE LITERATURA/ENQUADRAMENTO TEÓRICO E PRÁTICO	13
2.1. CONCEITOS/FUNDAMENTOS TEÓRICOS/INVESTIGAÇÕES EXISTENTES/TECNOLOGIAS A EXPLORAR	13
2.2. SOLUÇÕES (TÉCNICAS) EXISTENTES	14
3. TRABALHO DESENVOLVIDO	15
3.1. ANÁLISE E ESPECIFICAÇÃO	15
3.1.1. <i>Requisitos.....</i>	<i>15</i>
3.1.2. <i>Arquiteturas Lógica/Funcional</i>	<i>16</i>
3.1.3. <i>Modelação.....</i>	<i>16</i>
3.1.4. <i>Interação.....</i>	<i>18</i>
3.1.5. <i>Serviços/Integração</i>	<i>18</i>
3.2. IMPLEMENTAÇÃO	18
3.2.1. <i>Linguagem e Estrutura</i>	<i>18</i>
3.2.2. <i>Funções Implementadas.....</i>	<i>19</i>
3.2.3. <i>Ficheiros Utilizados</i>	<i>20</i>
3.2.4. <i>Primeira Fase de Implementação - Listas Ligadas.....</i>	<i>21</i>
3.2.5. <i>Segunda Fase de Implementação - Grafos</i>	<i>25</i>
4. ANÁLISE E DISCUSSÃO DE RESULTADOS.....	28
5. CONCLUSÃO.....	30
6. BIBLIOGRAFIA	31

ÍNDICE DE FIGURAS

FIGURA 1-STRUCT ANTENA	16
FIGURA 2 - STRUCT ADJACENTE E VERTICE	17
FIGURA 3 - ESBOÇO DA LISTA LIGADA.....	21
FIGURA 4 - FICHEIRO "ANTENAS.TXT"	22
FIGURA 5 - LISTA DAS COORDENADAS APRESENTADAS NA CONSOLA	22
FIGURA 6 - FUNÇÃO "CARREGARANTENASDEFICHEIRO"	23
FIGURA 7 - ATUALIZAÇÃO DO FICHEIRO "ANTENAS.TXT"	24
FIGURA 8 - FUNÇÃO "CARREGARANTENASDEFICHEIRO"	24
FIGURA 9 - ESBOÇO GRAFO - ESTRUTURA VÉRTICE	25
FIGURA 10 - FUNÇÃO "CRIARGRAFO"	26
FIGURA 11 - FUNÇÃO "LISTARINTERSECOES"	27

1. Introdução

Este projeto de avaliação de realização individual da Unidade Curricular (UC) Estruturas de Dados Avançadas (EDA), integrado no 2º semestre do 1º ano, visa o reforço e a aplicação dos conhecimentos adquiridos ao longo do semestre.

Com este projeto de avaliação pretende-se consolidar os conhecimentos relativos à definição e manipulação de estruturas de dados dinâmicas na linguagem de programação C.

Este documento deve ser visto como uma referência para uma abordagem de desenvolvimento de soluções de *software* para um problema. A implementação das soluções considera estruturas de dados dinâmicas, armazenamento em ficheiro e documentação com *Doxygen*.

Mais concretamente, aborda a distribuição de antenas numa cidade e o impacto das suas frequências, utilizando estruturas de dados dinâmicas.

Inicialmente, foram utilizadas listas ligadas para armazenar e manipular antenas, identificando localizações com efeito nefasto.

Na segunda fase do projeto, o foco passou a ser a modelação do problema através de grafos.

Cada antena foi representada como um vértice de um grafo, e as ligações entre antenas como arestas. Estas ligações foram estabelecidas com base na frequência das antenas. Com esta estrutura foi possível implementar funcionalidades adicionais como:

- Criação e inserção de adjacências (arestas) entre antenas.
- Percursos em profundidade (DFS) e largura (BFS) para explorar as antenas alcançáveis a partir de uma determinada origem.
- Procura de todos os caminhos possíveis entre duas antenas.
- Identificação de intersecções entre antenas de frequências distintas, permitindo analisar interferências ou pontos críticos de sobreposição.

A introdução de grafos permitiu uma visão mais completa da rede de antenas, permitindo também uma analogia com vários problemas do dia a dia. Esta fase exigiu também o reforço da gestão de memória dinâmica e o uso de algoritmos clássicos de grafos adaptados ao contexto do projeto.

Todo o código, bem como os restantes documentos que constituem o projeto encontram-se disponíveis no repositório *GitHub* associado ao projeto, acessível através do seguinte link: https://github.com/ricardopereira05/EDA_TP.git.

1.1. Motivação

O desenvolvimento deste projeto constitui uma oportunidade para aplicar os conhecimentos teóricos adquiridos em sala de aula num cenário com características práticas e concretas.

Esta experiência reforçou a importância da organização do código, da clareza na documentação.

A primeira fase, centrada na manipulação de listas ligadas, permitiu reforçar a compreensão dos conceitos fundamentais da linguagem C, como a alocação dinâmica de memória, a estruturação de dados e o armazenamento através de ficheiros. Já a segunda fase, com a introdução de estruturas de grafos, trouxe uma perspetiva mais avançada abrindo espaço à implementação de algoritmos como a procura em profundidade e largura.

Assim, mais do que cumprir os objetivos definidos, este projeto contribuiu para uma compreensão mais profunda dos desafios associados ao desenvolvimento de *software* estruturado e eficiente, sendo uma etapa importante no percurso académico.

1.2. Enquadramento

Este projeto insere-se na unidade curricular EDA, como o objetivo do desenvolvimento de soluções de *software* para problemas reais, com ênfase no uso de estruturas de dados dinâmicas, persistência de dados em ficheiro e documentação com *Doxygen*.

Inicialmente, foram utilizadas estruturas como listas ligadas para representar e manipular as antenas, as suas coordenadas e as respetivas frequências. Esta abordagem permitiu identificar locais com interferência, simulando por exemplo o impacto negativo da proximidade entre antenas com frequências iguais.

Na segunda fase do projeto, a complexidade aumentou com a introdução de estruturas de grafos. Foram implementadas operações de procura em profundidade (DFS) e procura em largura (BFS) e identificação de caminhos entre duas antenas.

A escolha por uma linguagem como C (ou C++) e a utilização de estruturas de dados dinâmicas justificam-se pela necessidade de simular de forma eficiente um ambiente realista, onde os recursos não são fixos e as alterações são frequentes. Além disso, a abordagem proposta visa ser uma ferramenta de referência e aprendizagem.

1.3. Objetivos

Fase 1:

- Gerir antenas: Criar uma estrutura de dados dinâmica para armazenar as antenas e as suas respetivas características.
- Analisar interferência: Implementar um algoritmo para detetar localizações com efeito nefasto com base na frequência e na distância entre as antenas.
- Otimizar a distribuição: Desenvolver uma estratégia eficiente para a distribuição das antenas.
- Gerir os dados: Implementar funcionalidades para guardar e carregar os dados das antenas em ficheiros.
- Documentar com *Doxygen*: Gerar documentação técnica detalhada através do *Doxygen*.
- Desenvolver habilidades em C: Consolidar e adquirir competências de programação em linguagem C, com foco na utilização de estruturas de dados.
- Avaliar desempenho: Testar e analisar a eficiência do código desenvolvido.

Fase 2:

- Grafos: Expandir a representação do sistema de antenas utilizando a estrutura de dados grafo.
- Criar e gerir adjacências: Implementar a criação e inserção de arestas (ligações) entre vértices (antenas), representando a ligação entre elas.
- Explorar o grafo: Desenvolver algoritmos de procura em profundidade (DFS) e procura em largura (BFS), que permitem percorrer a rede de antenas a partir de uma origem e identificar todas as antenas alcançáveis.
- Analisar caminhos: Identificar e listar todos os caminhos possíveis entre duas antenas específicas.
- Ligação com a Fase 1: Reutilização de funções implementadas na Fase 1.
- Documentar com *Doxygen*: Gerar documentação técnica detalhada através do *Doxygen*.

1.4. Metodologia de trabalho

O desenvolvimento deste projeto foi estruturado em várias fases, seguindo uma abordagem modular.

Numa fase inicial (Fase 1), procedeu-se a uma análise detalhada do problema, com o intuito de identificar os tipos de dados a serem manipulados e as suas inter-relações. Elaborou-se um modelo do sistema, com a definição de listas ligadas para representar as antenas, as suas coordenadas e frequências. Foram também implementadas funcionalidades principais como a inserção, remoção, listagem e identificação de efeitos nefastos entre antenas, bem como o armazenamento e carregamento de dados em ficheiros.

A implementação da Fase 1 foi organizada da seguinte forma:

- main.c: Contém o ponto de entrada e a lógica principal do programa;
- funcoes.c: Implementação das estruturas e funções de manipulação de listas;
- funcoes.h: Declaração das funções e estruturas utilizadas na Fase 1.

Na Fase 2, o projeto evoluiu com a introdução de estruturas de grafos, permitindo modelar arestas entre as antenas.

A estrutura modular dos ficheiros nesta fase foi ajustada para:

- main.c: Ponto de entrada do programa, incluindo chamadas a funcionalidades de grafos;
- grafos.c: Implementação de todas as funções relacionadas com a estrutura de grafos (criação de arestas, buscas em profundidade e largura, deteção de caminhos, etc.);
- grafos.h: Definição das estruturas de grafos e declaração das funções associadas.

Foram ainda implementados algoritmos para:

- Procura em profundidade (DFS) e procura em largura (BFS);
- Identificação de todos os caminhos possíveis entre duas antenas;
- Deteção de intersecções entre antenas com diferentes frequências.

Para garantir uma documentação clara e estruturada, foi utilizada a ferramenta Doxygen, permitindo a geração automática de documentação técnica a partir dos comentários no código.

Por fim, foi feita uma reflexão crítica sobre o trabalho realizado, destacando os ganhos em termos de domínio das estruturas dinâmicas e compreensão prática dos conceitos de grafos aplicados a problemas reais.

1.5. Plano de Trabalho

Objetivo 1 – Estabelecer bases do projeto:

- Gestão de antenas e análise de interferências.
- Levantamento dos requisitos técnicos.
- Planeamento do uso das ferramentas: linguagem C e Doxygen para documentação.

Objetivo 2 – Desenvolver a estrutura de dados dinâmica (Fase 1):

- Criar a estrutura de dados dinâmica baseada em listas ligadas.
- Implementar funções de manipulação de dados (adicionar, remover, listar antenas).
- Definir as características das antenas (coordenadas, frequência).
- Criar a função para calcular a distância entre antenas e determinar a existência de efeito nefasto com base na proximidade e na frequência.

Objetivo 3 – Ficheiros:

- Desenvolver funções para guardar antenas em ficheiros.
- Implementar funções para carregar os dados de antenas a partir de ficheiros.

Objetivo 4 – Documentação do Código com Doxygen:

- Inserir comentários Doxygen nas funções e estruturas de dados, explicando os seus objetivos.
- Gerar documentação técnica automática e atualizada a partir do código.

Objetivo 5 – Elaboração da primeira fase do relatório:

- Registrar as decisões de implementação, estrutura do código e imagens ilustrativas de erros encontrados.
- Descrever os objetivos, a metodologia e os resultados obtidos na Fase 1.

Objetivo 6 – Implementar estrutura de grafos (Fase 2):

- Criar e implementar novas estruturas para representar grafos (arestas e vértices).
- Adaptar as funções utilizadas na Fase 1 do projeto.
- Desenvolver funcionalidades de procura em profundidade (DFS) e largura (BFS).
- Criar algoritmo para identificar todos os caminhos possíveis entre duas antenas.
- Listar intersecções de antenas com frequências distintas, simulando conflitos ou sobreposições.

- Permitir ligação entre antenas através de criação de arestas.

Objetivo 7 – Ajustes finais:

- Realizar testes.
- Corrigir erros e melhorar a estrutura visual do código.

Objetivo 8 – Preparação do relatório final:

- Consolidar o relatório com todas as fases do projeto.
- Rever e organizar o conteúdo para garantir coerência, clareza e correspondência com o código desenvolvido.

1.6. Estrutura do documento

Este documento encontra-se organizado por capítulos e secções que descrevem as etapas do projeto desenvolvido. A estrutura adotada permite uma compreensão clara do problema, da abordagem seguida e das soluções implementadas.

Capítulo 1 – Introdução

Apresenta o contexto geral do projeto, incluindo a motivação, o enquadramento teórico, os objetivos definidos, a metodologia adotada e o plano de trabalho a seguir.

Capítulo 2 – Estado da Arte / Revisão de Literatura / Enquadramento Teórico e Prático

Reúne os principais conceitos teóricos e técnicos relevantes para o projeto, o uso de listas ligadas em C.

Capítulo 3 – Trabalho Desenvolvido

Descreve detalhadamente o processo de desenvolvimento do projeto. Inclui a análise e especificação do sistema, bem como a implementação das funcionalidades previstas.

3.1 Análise e Especificação: Define os requisitos funcionais e a estrutura lógica do sistema.

3.2 Implementação: Apresenta a linguagem utilizada, a estrutura do código, as funções implementadas e os ficheiros envolvidos.

3.2.1 Primeira Fase de Implementação – Listas Ligadas: Detalha a fase inicial do projeto, centrada na utilização de listas ligadas para representar e manipular antenas, incluindo os objetivos, a estrutura utilizada e os principais desafios enfrentados.

3.2.2 Segunda Fase de Implementação – Grafos: Apresenta a evolução para o uso de grafos como estrutura de dados, descrevendo como as antenas passaram a estar interligadas por adjacências. Inclui também os problemas detetados, como a ligação indevida entre frequências distintas.

Capítulo 4 – Análise e Discussão de Resultados

Avalia criticamente os resultados obtidos com base nos testes realizados, relacionando-os com os objetivos definidos.

Capítulo 5 – Conclusão

Resume os principais contributos do trabalho realizado, reflete sobre o processo de desenvolvimento e propõe possíveis evoluções futuras para o projeto.

Capítulo 6 – Bibliografia

Apresenta as fontes bibliográficas e técnicas consultadas durante a realização do projeto.

2. Estado da Arte/ Revisão de literatura/Enquadramento Teórico e Prático

2.1. Conceitos/Fundamentos Teóricos/Investigações existentes/Tecnologias a explorar

As listas ligadas são uma estrutura de dados dinâmica amplamente utilizada na área da programação, permitindo o armazenamento e a gestão eficiente de conjuntos de dados, cujo tamanho pode variar durante a execução de um programa.

Ao contrário dos *arrays*, que possuem um tamanho fixo, as listas ligadas possibilitam a inserção e remoção de elementos sem a necessidade de realizar alterações na alocação de memória.

Cada nó de uma lista ligada contém os dados relevantes e um apontador que referencia o próximo nó da lista. Esta abordagem facilita a inserção ordenada de dados, como no caso da organização de antenas por coordenadas, e permite percorrer todos os elementos de forma sequencial.

Neste projeto, as listas ligadas são utilizadas para representar a rede de antenas, armazenando informação como a frequência e as coordenadas de cada antena, e permitindo as diferentes operações.

"Uma lista encadeada (ou lista ligada) é uma representação de uma sequência de objetos na memória do computador. Cada elemento é armazenado em uma célula que contém o valor e um ponteiro para o próximo elemento da lista" (Coelho, s.d.).

Um grafo é uma forma de representar objetos e as ligações entre eles. Os objetos são chamados de vértices e as ligações são chamadas de arestas.

Por exemplo, se pensarmos nas antenas como pontos no espaço, um grafo permite mostrar quais estão ligadas entre si. É como desenhar um mapa onde os pontos representam antenas e as linhas entre eles mostram ligações.

Os grafos ajudam para fazer pesquisas, como descobrir todas as antenas ligadas a uma determinada antena ou encontrar caminhos entre duas.

“Um grafo é descrito por um par (V,E) , onde V é o conjunto de vértices e E o conjunto das arestas que ligam pares de vértices. Um grafo pode ser dirigido ou direcionado, se as arestas tiverem uma direção, ou não dirigido, se as arestas não tiverem direção”. (WikiCiências, s.d.)

2.2. Soluções (técnicas) existentes

Diversos projetos utilizam estruturas de dados dinâmicas, como listas ligadas, para representar sistemas que exigem flexibilidade na inserção e remoção de elementos, tal como acontece em redes de antenas. Nos domínios como a gestão de redes de comunicação, é comum utilizar listas ligadas para armazenar dispositivos (nós) de forma dinâmica.

A utilização de grafos com listas de adjacências permite representar de forma eficiente ligações entre antenas, permitindo encontrar os caminhos possíveis entre os vértices, e identificar situações de interferência entre frequências. Através de pesquisas sobre grafos conclui que esta abordagem é bastante comum em várias situações.

3. Trabalho Desenvolvido

3.1. Análise e Especificação

3.1.1. Requisitos

Fase 1 – Lista de Antenas:

- O programa deve permitir o carregamento de antenas a partir de um ficheiro.
- Deve ser possível inserir manualmente novas antenas, garantindo que não haja duplicação de coordenadas.
- Deve ser possível remover uma antena com base nas suas coordenadas.
- O sistema deve identificar efeitos nefastos entre antenas da mesma frequência (com base na proximidade).
- Deve ser possível listar todas as antenas presentes na estrutura.
- A informação das antenas deve ser guardada e atualizada num ficheiro binário.
- O sistema deve libertar toda a memória utilizada antes de terminar a execução.

Fase 2 – Representação em Grafo:

- O programa deve permitir criar arestas entre antenas, representando as antenas como um grafo com lista de adjacências.
- Deve ser possível realizar uma busca em profundidade (DFS) a partir de uma antena, listando as antenas alcançadas.
- Deve ser possível realizar uma busca em largura (BFS) para explorar as antenas em níveis de ligação.
- O sistema deve ser capaz de identificar todos os caminhos possíveis entre duas antenas específicas, listando as sequências de arestas.
- Deve ainda ser possível, dada duas frequências diferentes (ex: A e B), listar as interseções de pares de antenas que estão ligadas diretamente e que apresentam essas frequências.

3.1.2. Arquiteturas Lógica/Funcional

- Estrutura principal:
 - Fase 1: Lista ligada para armazenar dinamicamente as antenas e as suas características (frequência, coordenadas).
 - Fase 2: Estrutura em grafo com lista de adjacências, onde cada antena (vértice) tem arestas para outras antenas.
- Camadas funcionais:
 - Input/Output:
 - ❖ Leitura das antenas a partir de ficheiro binário (antenas.bin) e texto (antenas.txt).
 - ❖ Escrita e atualização da estrutura de dados no ficheiro binário.
 - Gestão de dados:
 - ❖ Inserção e remoção de antenas na lista ligada.
 - ❖ Identificação de zonas com efeito nefasto com base na frequência e proximidade.
 - ❖ Apresentação da lista das antenas armazenadas.
 - Módulo de grafos:
 - ❖ Criação de arestas entre antenas (ligação entre vértices).
 - ❖ Algoritmos de busca em profundidade (DFS) e busca em largura (BFS) para explorar o grafo.
 - ❖ Função para listar todos os caminhos possíveis entre duas antenas.
 - ❖ Identificação de pares de antenas interligadas com frequências distintas (ex: A e B), listando as respetivas coordenadas.

3.1.3. Modelação

- struct Antena: representa cada antena com frequência, coordenadas e apontador para o próximo elemento.

```
typedef struct Antena {  
    char frequencia; //Frequência da antena (exemplos utilizados: '0', 'A', 'B', 'C')  
    int x,y;         // Coordenadas da antena no plano (x, y)  
    struct Antena *prox; //Ponteiro para a próxima antena da lista  
} Antena;
```

Figura 1-Struct Antena

```
typedef struct adjacente {
    struct vertice* origem;
    struct vertice* destino;
    struct adjacente* next;
} AdjD;

typedef struct vertice {
    char frequencia;
    int x, y;
    int visitado;
    struct vertice* prox;
    AdjD* adjacencias;
} Vertice;
```

Figura 2 - Struct adjacente e vertice

As estruturas apresentadas modelam um grafo onde cada vértice representa uma antena, e as arestas representam ligações entre antenas

➤ Vertice

Esta estrutura representa uma antena. As características são:

- char frequencia - Armazena a frequência da antena (por exemplo, 'A', 'B', 'C'). Este campo é útil para distinguir grupos de antenas por frequência.
- int x, y - Coordenadas que representam a posição da antena.
- int visitado - Um marcador auxiliar usado em algoritmos de travessia do grafo. Evita visitar o mesmo vértice múltiplas vezes.
- struct vertice* prox - Aponta para o próximo vértice (antena) na lista ligada de todas as antenas. Isto permite armazenar todos os vértices do grafo numa lista.
- AdjD* adjacencias - Aponta para a lista de adjacências (arestas) que partem deste vértice. Cada item desta lista representa uma ligação a outra antena (vértice).

➤ Adjacente

Esta estrutura representa uma ligação entre duas antenas (vértices). Cada aresta liga um vértice de origem a um vértice de destino:

- struct vertice* origem - Aponta para o vértice de origem da ligação.
- struct vertice* destino - Aponta para o vértice de destino da ligação. Ou seja, indica qual antena está ligada a partir do vértice de origem.
- struct adjacente* next - Aponta para a próxima adjacência (aresta) na lista ligada. Permite armazenar múltiplas ligações a partir de um mesmo vértice.

3.1.4. Interação

- O ficheiro main.c centraliza a execução do programa, manipulando diretamente as funções de gestão de antenas e, numa segunda fase, as funções de criação de ligações (arestas) e operações no grafo.
- O programa não possui menu interativo, apenas chamadas “secas” das funções. Toda a interação é feita com inserções manuais de antenas, remoções, criação de ligações e chamadas a funções de procura (DFS, BFS) ou listagem de caminhos possíveis.

3.1.5. Serviços/Integração

- Leitura de antenas a partir de ficheiro: O programa inclui uma funcionalidade que permite carregar antenas guardadas num ficheiro, facilitando a recuperação de dados entre execuções. Na Fase 1, este ficheiro é designado por antenas.txt.
- Escrita automática em ficheiro: Após a inserção, remoção ou atualização de antenas, os dados são guardados num novo ficheiro, garantindo persistência da informação.

3.2. Implementação

3.2.1. Linguagem e Estrutura

- Linguagem utilizada: Toda a implementação foi realizada na linguagem C permitindo uma gestão precisa das estruturas dinâmicas envolvidas.

- Organização modular do projeto:

Fase 1:

- main.c: Ponto de entrada do programa. Contém a lógica geral, chamadas às funções de manipulação de antenas e testes.
- funcoes.h: Ficheiro com as declarações das funções e estruturas utilizadas, além de comentários Doxygen para documentação automática.
- funcoes.c: Implementação das funcionalidades relativas à gestão de antenas (inserção, remoção, listagem, cálculo de efeitos nefastos, leitura/escrita de ficheiros).

Fase 2:

- main.c: Mantido como ponto de entrada, agora com chamadas às funções relacionadas com grafos.
- grafo.h: Declarações das estruturas de dados e funções específicas para a modelação do grafo, incluindo busca em largura, caminhos e intersecções de frequências.
- grafo.c: Implementação das funcionalidades relacionadas com grafos: criação de arestas, inserção de adjacências, criação de uma nova antena, inserção, cálculo de efeitos nefastos, percursos em largura (BFS), identificação de todos os caminhos possíveis entre duas antenas e listagem de ligações entre antenas.

Esta organização modular permitiu dividir responsabilidades e facilitar a nível estrutural, evitando longas procuras por funções, por exemplo, na procura no código.

3.2.2. Funções Implementadas

Fase 1:

- inserirAntena: insere antenas na lista de forma ordenada e evita duplicados.
- removerAntena: remove uma antena e atualiza o ficheiro.
- listarAntenas: imprime todas as antenas existentes.
- calcularEfeitosNefastos: deteta e mostra os locais com possíveis interferências.
- carregarAntenasDeFicheiro: lê uma matriz de antenas de um ficheiro.
- libertarMemoria: liberta todos os nós da lista.

Fase 2:

- Operações com Grafos

- criarAdjacencia: Cria uma aresta entre duas antenas (vértices), representando uma ligação no grafo.
- inserirAdjacencia: Insere uma aresta (adjacência) na lista de ligações de uma antena.
- removerAdjacencia: Remove uma adjacência da lista das ligações.
- insereAntena: insere antenas na lista de forma ordenada e evita duplicados.
- removeAntena: remove uma antena.
- dfs: Implementa a procura em profundidade (DFS) a partir de uma antena, listando na consola as coordenadas de todas as antenas alcançadas.
- bfs: Implementa a procura em largura (BFS) a partir de uma antena, listando todas as antenas alcançadas.
- encontrarCaminhos: Lista todos os caminhos possíveis entre duas antenas, apresentando a sequência das antenas que foram visitadas.
- imprimirCaminho: Imprime o caminho da antena de origem até á de destino.
- listarIntersecoes: Dadas duas frequências distintas na mesma localização imprime as diferentes antenas.
- efeitoExiste: Verifica se o Efeito Nefasto já foi registado.
- calcularEfeitosNefastos: deteta e mostra os locais com possíveis interferências.
- libertarMemoria: Liberta toda a memória alocada para a lista de antenas
- listarAntenas: Lista todas as antenas da estrutura ligada de vértices.
- carregarAntenasDeFicheiro: lê uma matriz de antenas de um ficheiro.
- guardarAntenasEmFicheiroBinario: * Esta função percorre a lista ligada de antenas e escreve cada vértice (antena)
- * no ficheiro binário "antenas.bin". Cada vértice é guardado com todos os atributos.
- limparVisitados: Marca todos os vértices da lista como não visitados.

3.2.3. Ficheiros Utilizados

Entrada: antenas.txt (com matriz de antenas).

Saída: antenas2.txt (com as antenas atualmente listadas)

antenas.bin (com as antenas guardades num ficheiro binário)

3.2.4. Primeira Fase de Implementação - Listas Ligadas

O principal objetivo da primeira fase do trabalho foi desenvolver uma solução para representar e manipular a distribuição de antenas numa cidade, utilizando estruturas de dados dinâmicas.

Pretendeu-se:

- Implementar uma lista ligada para armazenar as antenas e as suas características como frequência e coordenadas;
- Carregar os dados das antenas a partir de um ficheiro de texto;
- Implementar operações para inserir, remover e listar antenas na estrutura de dados;
- Identificar e representar localizações com efeito nefasto, com base nas frequências e coordenadas das antenas;
- Exibir os resultados na consola.

Este trabalho permitiu consolidar conhecimentos sobre estruturas de dados dinâmicas em C.

Esboço da Lista Ligada

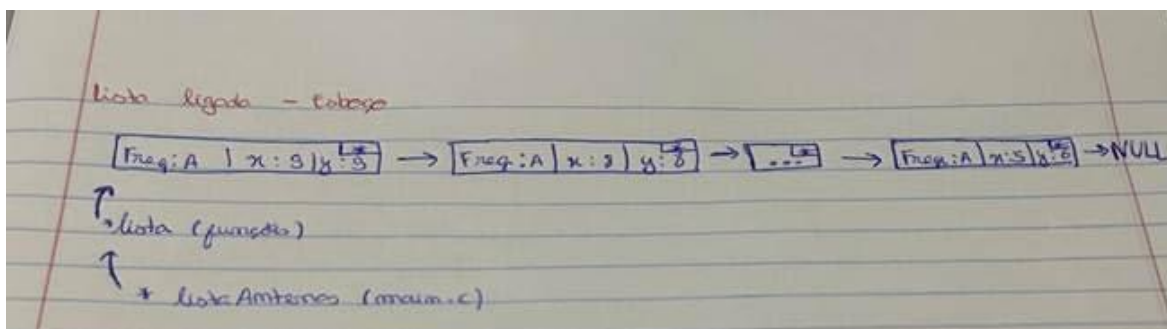


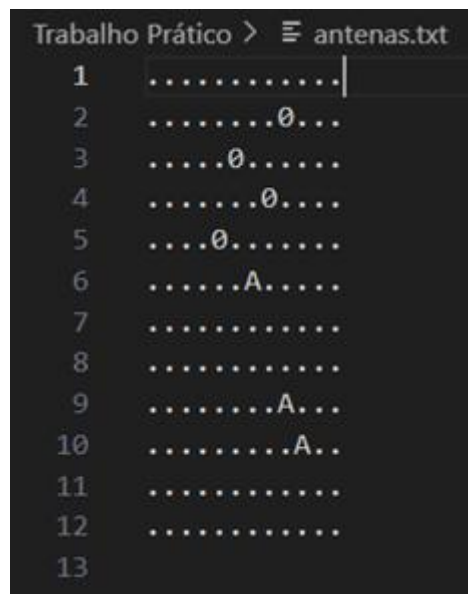
Figura 3 - Esboço da Lista Ligada

- listaAntenas: É a variável principal na main que armazena o apontador para o primeiro nó da lista;

- lista: É um apontador passado como argumento para as funções. Dentro da função, lista recebe o valor de listaAntenas, permitindo manipular a lista;
- “*”: Representa os apontadores prox que ligam os nós entre si.

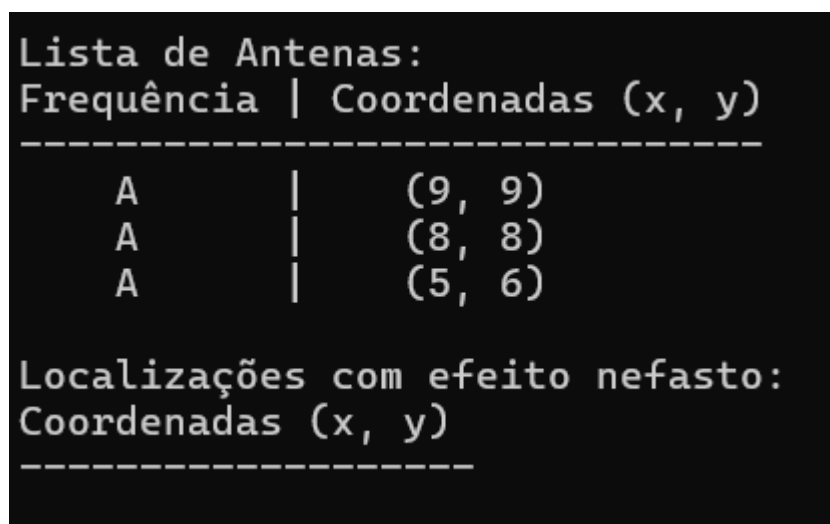
Problemas encontrados:

- Erro na listagem das coordenadas das antenas



```
Trabalho Prático > antenas.txt
1 .....|
2 .....0...
3 .....0....
4 .....0....
5 ....0.....
6 .....A....
7 .....
8 .....
9 .....A...
10 .....A..
11 .....
12 .....
13 .....
```

Figura 4 - Ficheiro "antenas.txt"



```
Lista de Antenas:
Frequência | Coordenadas (x, y)
-----
A          | (9, 9)
A          | (8, 8)
A          | (5, 6)

Localizações com efeito nefasto:
Coordenadas (x, y)
-----
```

Figura 5 - Lista das coordenadas apresentadas na consola


```

// Carregar antenas a partir de um ficheiro
Antena* carregarAntenasDeFicheiro(const char *nomeFicheiro) {
    FILE *file = fopen(nomeFicheiro, "r");
    if (!file) {
        printf("Erro ao abrir o ficheiro!\n");
        return NULL;
    }

    Antena *lista = NULL;
    int linha = 0;
    char buffer[256];

    while (fgets(buffer, sizeof(buffer), file)) {
        for (int coluna = 0; buffer[coluna] != '\n' && buffer[coluna] != '\0'; coluna++) {
            if (buffer[coluna] >= 'A' && buffer[coluna] <= 'Z') {
                printf("Lendo antena %c na posição (x=%d, y=%d)\n", buffer[coluna], coluna, linha);
                lista = inserirAntena(lista, buffer[coluna], coluna, linha);
            }
        }
        linha++;
    }
}

```

Figura 6 - Função "carregarAntenasDeFicheiro"

As coordenadas das antenas foram ajustadas para começar em 1, em vez de 0, tornando a representação mais intuitiva.

```
lista = inserirAntena(lista, buffer[coluna], coluna + 1, linha + 1);
```

- Erro na atualização do ficheiro “antenas.txt”

Figura 7 - Atualização do Ficheiro "antenas.txt"

Para garantir a atualização correta do ficheiro, optei por gerar um novo ficheiro, "antenas2.txt", onde são registadas todas as coordenadas das antenas no final da execução do programa.

```
Trabalho Prático > C funcoes.c > ...
164 Antena* carregarAntenasDeFicheiro(const char *nomeFicheiro) {
173     Antena *lista = NULL;
174     int linha = 0;
175     char buffer[256];
176
177     while (fgets(buffer, sizeof(buffer), file)) {
178         for (int coluna = 0; buffer[coluna] != '\n' && buffer[coluna] != '\0'; coluna++) {
179             //if (buffer[coluna] >= 'A' && buffer[coluna] <= 'Z') {
180                 if (buffer[coluna] != '.' && buffer[coluna] != '\n') {
```

Figura 8 - Função "carregarAntenasDeFicheiro"

A função “carregarAntenasDeFicheiro” inicialmente listava apenas antenas com frequências entre 'A' e 'Z' devido a uma restrição na linha 179. Para garantir que também fossem lidas as coordenadas com valor 0 foi ajustado o código.

3.2.5. Segunda Fase de Implementação - Grafos

O principal objetivo da segunda fase do trabalho foi através de uma estrutura do tipo grafo representar as relações de ligação entre antenas.

Pretendeu-se:

- Implementar um grafo com listas de adjacências, onde cada antena representa um vértice e as ligações entre elas representam arestas;
- Adicionar funcionalidades de procura em profundidade (DFS) e procura em largura (BFS) a partir de uma antena de origem;
- Determinar todos os caminhos possíveis entre duas antenas distintas;
- Listar interseções entre pares de antenas com diferentes frequências;
- Adaptar a leitura e gravação dos dados para ficheiros binários, garantindo persistência da estrutura de grafo;
- Consolidar conhecimentos sobre grafos.

Esboço da Estrutura Grafo

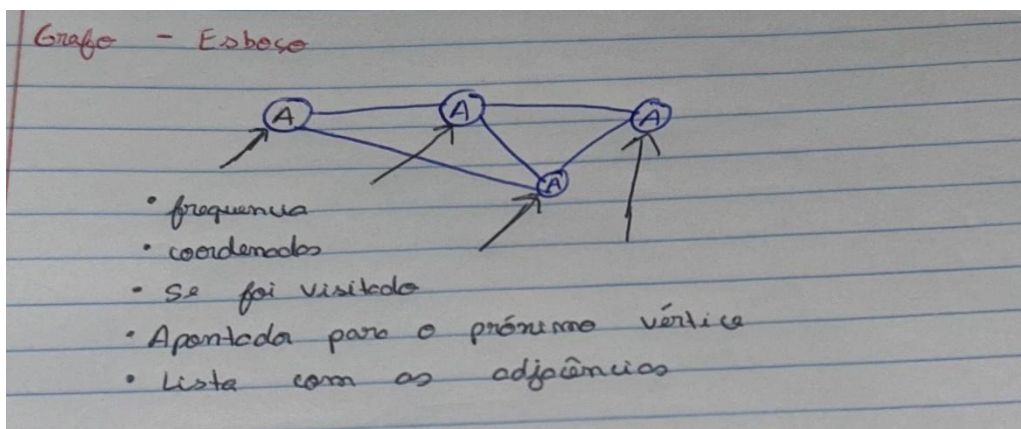


Figura 9 - Esboço Grafo - Estrutura Vértice

Problemas encontrados:

Um dos principais problemas enfrentados foi que as ligações (arestas) estavam a ser criadas entre antenas de frequências diferentes, por exemplo entre antenas da frequência A e da frequência B, o que revelava uma má eficiência da criação do grafo.

```
Vertice* CriarGrafo(Vertice* lista) {  
    if (lista == NULL) {  
        // Se a lista de vértices estiver vazia, não há nada a fazer.  
        return NULL;  
    }  
    Vertice* aux = lista; // aux percorre cada vértice na lista.  
    while (aux != NULL) {  
        Vertice* comparar = lista; // comparar percorre novamente toda a lista para comparar com 'aux'  
        while (comparar != NULL) {  
            // verifica se não está a comparar a antena consigo mesma e se têm a mesma frequência  
            if (comparar != aux && comparar->frequencia == aux->frequencia) {  
                // Cria uma adjacência de aux para comparar  
                inserirAdjacencia(aux, comparar);  
            }  
            comparar = comparar->prox;  
        }  
        aux = aux->prox;  
    }  
    return lista;  
}
```

Figura 10 - Função "CriarGrafo"

Com a implementação da função CriarGrafo na sua forma atual, foi possível garantir uma gestão eficiente do grafo, assegurando que as ligações (arestas) só são estabelecidas entre antenas que partilham a mesma frequência. Assim, a função CriarGrafo contribui para o correto funcionamento do sistema, assegurando que o grafo represente as relações entre antenas, facilitando operações posteriores.

```

int listarIntersecoes(Vertex* lista, char f1, char f2) {
    if (!lista) return 0;
    printf("Interseções entre antenas %c e %c:\n", f1, f2);
    int encontrou = 0;
    for (Vertex* aux1 = lista; aux1 != NULL; aux1 = aux1->prox) {
        for (Vertex* aux2 = aux1->prox; aux2 != NULL; aux2 = aux2->prox) {
            if (((aux1->frequencia == f1 && aux2->frequencia == f2) ||
                (aux1->frequencia == f2 && aux2->frequencia == f1)) &&
                aux1->x == aux2->x && aux1->y == aux2->y) {
                printf("Interseção em (%d, %d)\n", aux1->x, aux1->y);
                encontrou = 1;
            }
        }
    }
    return 1;
}

```

Figura 11 - Função "listarIntersecoes"

Inicialmente, a função `listarIntersecoes` foi desenvolvida com o objetivo de identificar e listar coordenadas onde existiam interseções entre antenas de frequências diferentes, isto é, antenas com a mesma posição (x, y) mas com frequências distintas (por exemplo, frequência A e frequência B). Esta funcionalidade fazia sentido numa fase inicial do projeto, em que o grafo ainda permitia, de forma incorreta a criação de arestas entre antenas de diferentes frequências.

Contudo, com a reestruturação da função `CriarGrafo`, passou a ser garantido que as ligações são estabelecidas apenas entre antenas da mesma frequência, eliminando assim o cenário de interligações inválidas. No entanto, não foi possível atingir os objetivos inicialmente definidos para esta função.

4. Análise e Discussão de Resultados

Durante a execução do programa, foram realizados testes com diferentes disposições de antenas no ficheiro antenas.txt, bem como inserções manuais no código para validar o correto funcionamento das funcionalidades implementadas.

A funcionalidade de carregamento automático do ficheiro funcionou corretamente. Durante o desenvolvimento, inicialmente as coordenadas foram atribuídas a partir da coordenada (0,0), mas posteriormente foi feita uma correção no código acrescentando +1 ao índice de linha e coluna para que as coordenadas passassem a começar na coordenada (1,1), de forma a tornar o sistema de coordenadas mais intuitivo. Os caracteres '.' foram corretamente ignorados, e os restantes foram transformados em objetos do tipo Antena com os dados de coordenadas e frequência devidamente atribuídos.

A remoção de uma antena específica também se comportou como esperado, ao nível da estrutura em memória e na atualização do ficheiro antenas2.txt.

Na verificação dos efeitos nefastos, o programa identifica os pares de antenas com a mesma frequência que estão alinhadas na vertical ou horizontal e cuja distância entre elas é exatamente dois. Quando essa condição é satisfeita, o programa calcula o ponto intermédio entre as antenas e imprime as respetivas coordenadas de interferência na consola. A listagem final das antenas apresenta a informação de forma clara e ordenada, facilitando a visualização dos dados.

Conclui-se que os objetivos definidos para a primeira fase foram cumpridos de forma satisfatória, evidenciando uma implementação alinhada com os requisitos estabelecidos.

Na segunda fase do projeto, foram desenvolvidas e testadas várias funcionalidades adicionais que permitiram a criação e gestão de um grafo, com o objetivo de representar e analisar as ligações possíveis entre antenas da mesma frequência. Para isso, foram criadas estruturas adequadas e funções específicas como CriarGrafo, inserirAdjacencia, removerAdjacencia, dfs entre outras.

Durante os testes, confirmou-se que a função CriarGrafo garantiu uma construção eficiente do grafo, respeitando o critério de ligação apenas entre antenas da mesma frequência. Esta abordagem corrigiu um dos problemas verificados na versão anterior do sistema, em que podiam ser criadas arestas entre antenas de frequências distintas, resultando numa representação incorreta e confusa.

Inicialmente, a função `listarIntersecoes` foi desenvolvida com o objetivo de identificar coordenadas onde existiam interseções entre antenas de frequências diferentes, ou seja, antenas localizadas na mesma posição, mas com frequências distintas. A atualização da função `CriarGrafo`, passou a ser assegurado que as conexões se estabelecem apenas entre antenas da mesma frequência, eliminando assim a ocorrência destas interligações inválidas. Por esse motivo, a função `listarIntersecoes` perdeu a sua utilidade na forma como foi implementada e pensada, não tendo sido possível atingir os objetivos propostos para esta funcionalidade.

De modo geral, as restantes funcionalidades desenvolvidas demonstraram comportar-se conforme o esperado.

Assim, conclui-se que os objetivos definidos para a segunda fase foram, em boa medida atingidos com sucesso.

5. Conclusão

A primeira fase do projeto constituiu um passo fundamental, tendo-se centrado na implementação de uma estrutura dinâmica capaz de representar a distribuição de antenas numa cidade. Através da utilização de listas ligadas, foi possível armazenar e manipular eficientemente os dados relevantes, bem como implementar funcionalidades de inserção, remoção e listagem. A utilização do *Doxygen* contribuiu significativamente para a clareza e manutenção do código, promovendo uma documentação acessível. Apesar de alguns desafios iniciais, como a correta leitura e atualização de ficheiros, estes foram ultrapassados com sucesso, conduzindo a uma solução funcional. Esta fase revelou-se essencial para consolidar a base do projeto e avançar para a segunda fase do mesmo.

Na segunda fase, as antenas passaram a ser representadas como vértices, com ligações (arestas) entre elas. Foram implementadas operações fundamentais de grafos, como procura em profundidade (DFS), procura em largura (BFS), caminhos possíveis entre antenas e interseções entre frequências distintas. O cálculo de efeitos nefastos foi também elaborado tendo em conta a base do código da primeira fase.

Com estas melhorias, o projeto permitiu consolidar na prática os conhecimentos aprendidos sobre estruturas de dados. Foi uma boa forma de treinar programação, organização do código e resolução de problemas. O trabalho foi feito por partes, o que ajudou a manter tudo mais claro e organizado.

6. Bibliografia

Coelho, P. R. (s.d.). *Linguagem C: Listas Encadeadas*. Obtido de https://www.facom.ufu.br/~anilton/EQQ09_PD_EngQuimica/lista_encadeada.pdf

WikiCiências. (s.d.). Obtido de WikiCiências-Casa das Ciências: https://wikiciencias.casadasciencias.org/wiki/index.php/P%C3%A1gina_principal