



Instituto Politécnico do Cávado e do Ave

Escola Superior de Tecnologia

Licenciatura em Engenharia de Sistemas Informáticos

Ano Letivo 2024/2025

Trabalho Prático – Fase 1

Estruturas de Dados Avançadas

Professor Luís Ferreira

Realizado por:

Duarte Miguel Fernandes Pereira N.º 27959

30 de março de 2025

Índice

Resumo	3
Introdução	4
Enquadramento	4
Problema a Resolver	4
Trabalho Desenvolvido	5
Análise e Modelação	5
Representação das Antenas: Estruturas de Dados.....	5
Estrutura de Dados para as Localizações com Efeito Nefasto	6
Deteção do Efeito Nefasto	Erro! Marcador não definido.
Implementação.....	7
Criação e Inserção de Antenas	7
Remoção de Antenas	8
Listagem de Antenas	8
Visualização do Mapa de Antenas	8
Destrução da Lista	11
Manipulação de Ficheiros	11
Resumo da Implementação	15
Análise de Resultados	16
Funcionalidades Não Implementadas	16
Conclusão	17
Resumo dos Resultados Alcançados.....	17
O Que Poderia Ser Melhorado.....	17
Possíveis Expansões do Trabalho	18
Conclusão Final	18

Resumo

O presente relatório descreve a implementação de uma solução para o problema de localização e análise de antenas numa cidade, com base na definição e manipulação de listas ligadas. A solução foi desenvolvida no âmbito do projeto de avaliação da Unidade Curricular de Estruturas de Dados Avançadas (EDA), do curso de Engenharia de Sistemas Informáticos do Instituto Politécnico do Cávado e do Ave (IPCA).

A cidade é modelada através de uma matriz onde cada elemento representa uma antena com uma frequência de ressonância específica. O objetivo é identificar e manipular essas antenas utilizando uma lista ligada simples, onde cada elemento da lista armazena a frequência e as coordenadas de uma antena. A solução inclui o carregamento de dados a partir de um ficheiro de texto e a implementação de operações como inserção e remoção de antenas, bem como a dedução automática das localizações com efeito nefasto, resultante da interação entre antenas da mesma frequência.

Foram implementadas também operações para manipulação das listas ligadas, com destaque para a listagem tabular das antenas. O código foi documentado extensivamente, utilizando Doxygen, e foram realizados testes com matrizes de diferentes dimensões, validando a eficiência e a correção da solução proposta.

Repositório no GitHub

O código fonte completo deste projeto está disponível no GitHub, onde pode ser visualizado, clonado ou contribuído. O repositório contém todo o código desenvolvido, bem como a documentação necessária para a execução e compreensão do projeto.

[GitHub - Repositório do Projeto](#)

Introdução

Enquadramento

O estudo e implementação de **estruturas de dados dinâmicas** é essencial para o desenvolvimento de soluções informáticas eficientes. Este trabalho insere-se na área de **Estruturas de Dados Avançadas (EDA)**, focando-se em técnicas de manipulação de dados de forma eficaz, com ênfase em problemas que envolvem **modificação dinâmica** de coleções de dados. A escolha de estruturas como **listas ligadas** é especialmente relevante para problemas que exigem inserção e remoção frequente de dados, já que esta estrutura permite uma manipulação eficiente dos dados sem a necessidade de realocação ou cópias dispendiosas, o que é crucial para o desempenho da solução.

A motivação para a resolução deste problema surge da necessidade de modelar e analisar a interferência entre antenas numa cidade. Este tipo de análise pode ser associado a problemas reais na gestão de **redes de telecomunicações** ou sistemas de **comunicação sem fio**, onde é necessário garantir que as frequências de transmissão das antenas não causem interferências prejudiciais em locais críticos. A aplicação de algoritmos de manipulação de dados e análise de localizações com interferência resulta numa solução que pode ser adaptada a diversas outras áreas de estudo, como sistemas de sensores ou redes de comunicação.

Problema a Resolver

O enunciado do trabalho propõe o desenvolvimento de uma solução para um problema de análise de antenas numa cidade, onde cada antena está sintonizada numa frequência específica, representada por caracteres. O problema consiste em identificar as localizações que apresentam **efeitos nefastos**, causados pela interferência entre antenas da mesma frequência, com base num padrão geométrico de alinhamento entre as antenas.

A implementação do projeto envolve a criação de uma **lista ligada simples**, onde cada elemento representa uma antena e contém informações sobre sua **frequência de ressonância** e suas **coordenadas**. A solução proposta exige a **manipulação dinâmica** dessas antenas, permitindo a inserção, remoção e dedução automática das localizações com efeitos nefastos, de forma a otimizar o desempenho da rede. O processo de dedução de interferências entre antenas, combinado com a **representação tabular** dos resultados, permite uma análise mais clara e eficiente do comportamento da cidade em termos de interferências de sinal.

A implementação das funcionalidades propostas será realizada em **linguagem C**, aproveitando a flexibilidade das **estruturas de dados dinâmicas** e garantindo que a solução seja modular e bem documentada. Para tanto, será utilizado **Doxygen** para gerar a documentação do código, facilitando a compreensão e manutenção do sistema.

Trabalho Desenvolvido

Análise e Modelação

A fase 1 deste projeto de avaliação envolve a implementação de uma solução para modelar antenas numa cidade, usando estruturas de dados dinâmicas para representar as antenas e suas propriedades (frequência e coordenadas). Além disso, é necessário calcular e identificar os locais que são afetados pelo efeito nefasto, causado por antenas alinhadas que compartilham a mesma frequência e que estão separadas por uma distância específica.

Representação das Antenas: Estruturas de Dados

O primeiro passo foi analisar como representar as antenas e suas propriedades de forma eficiente. Considerando que as antenas podem ser adicionadas ou removidas de forma dinâmica, a estrutura de dados ideal seria uma **lista ligada simples**. Vamos detalhar a escolha:

- **Lista Ligada Simples:** A lista ligada é uma estrutura de dados onde cada elemento (ou nó) contém:
 - **Valor:** As informações relacionadas à antena, ou seja, a **frequência** da antena (um carácter, como 'A', 'O', etc.) e suas **coordenadas** (x, y) na matriz que representa a cidade.
 - **Ponteiro:** Cada nó possui um ponteiro para o próximo nó na lista, o que facilita a navegação e inserção de novos elementos sem a necessidade de realocar ou reorganizar a memória.

A escolha da lista ligada deve-se à sua flexibilidade para **inserir** ou **remover** elementos de forma eficiente, sem a necessidade de mover outros elementos. Essa estrutura é particularmente útil quando não se conhece de antemão o número de antenas a serem armazenadas.

- **Estrutura da Antena:** A estrutura que modela uma antena é simples, composta por três atributos:
 - **frequência:** O carácter que representa a frequência da antena (por exemplo, 'A', 'O', etc.).
 - **coordenadas:** As coordenadas (x, y) que indicam a posição da antena na matriz da cidade.
 - **ponteiro para a próxima antena:** Este ponteiro permite que a lista ligada seja percorrida de forma sequencial, a partir da primeira antena até a última.

Aqui está a definição da estrutura para a **Antena**:

```
1  typedef struct Antena {  
2      char frequencia; /**< Frequência da antena */  
3      int x, y;        /**< Coordenadas X e Y da antena */  
4      struct Antena* prox; /**< Ponteiro para a próxima antena */  
5  } Antena;
```

Estrutura de Dados para as Localizações com Efeito Nefasto

Além da lista de antenas, é necessário considerar a identificação das localizações com efeito nefasto. O efeito nefasto ocorre quando duas antenas da **mesma frequência** estão **alinhadas** na mesma direção e a distância entre elas é tal que uma está **duas vezes mais distante** que a outra.

Para encontrar essas localizações afetadas, a solução vai precisar de calcular as distâncias relativas entre pares de antenas que têm a mesma frequência. A partir desses cálculos, geraremos uma lista ligada onde cada nó representará uma **localização com efeito nefasto**.

Como a lista ligada já está sendo utilizada para armazenar as antenas, a mesma estrutura pode ser adaptada para armazenar essas localizações, com uma ligeira modificação. Em vez de armazenar coordenadas da antena, armazenaremos as coordenadas das localizações afetadas, ou seja, as **posições em que ocorre o efeito nefasto**.

Implementação

A implementação do projeto envolveu a criação e manipulação de antenas utilizando listas ligadas. A seguir, detalho as principais funções implementadas para lidar com as antenas, a manipulação da lista e a exibição das antenas num mapa.

Criação e Inserção de Antenas

A primeira parte da implementação envolveu a criação de antenas. Para isso, foi desenvolvida uma função para alocar dinamicamente memória para uma nova antena e inicializar os seus campos com valores fornecidos. Cada antena tem três atributos principais: a frequência (um carácter que a identifica), as coordenadas **x** e **y** (que representam a sua localização no mapa), e um ponteiro **prox** para a próxima antena na lista.

Além da criação individual de antenas, foi necessário criar funções para inserir novas antenas na lista. A inserção pode ocorrer de duas formas:

1. **Inserção no início da lista:** Aqui, criamos uma antena através da função **CriarAntena** e a colocamos no início da lista, ajustando os ponteiros para garantir que o novo elemento seja corretamente ligado ao restante da lista. Este processo garante que as antenas mais recentes fiquem sempre no topo da lista.
2. **Inserção no fim da lista:** Para inserir uma antena no final, percorremos toda a lista até encontrar o último elemento, e aí ajustamos o ponteiro **prox** do último para apontar para a nova antena. Esse tipo de inserção é útil quando queremos garantir que as antenas sejam processadas na ordem em que foram adicionadas. Aqui está como foi implementada a função **InserirAntenaFim**:

```
1 Antena* InserirAntenaFim(Antena* lista, char freq, int x, int y)
2 {
3     Antena* novaAntena = CriarAntena(freq, x, y);
4     if (novaAntena == NULL) return lista;
5
6     if (lista == NULL) return novaAntena; // Se a lista estiver vazia, a nova antena se torna a lista
7
8     Antena* atual = lista;
9     while (atual->prox != NULL) { // Percorre a lista até o último elemento
10        atual = atual->prox;
11    }
12    atual->prox = novaAntena; // Liga a nova antena ao final da lista
13    return lista;
14 }
```

Remoção de Antenas

A remoção de antenas é um aspeto importante para gerir a lista de antenas dinamicamente. Implementamos uma função que permite remover uma antena com base nas suas coordenadas (**x**, **y**). A função percorre a lista procurando pela antena com essas coordenadas. Quando encontra a antena desejada, a memória alocada para ela é liberada e a lista é ajustada para que o ponteiro **prox** do elemento anterior aponte para o elemento seguinte, efetivamente removendo o nó da lista.

Além disso, a função consegue lidar com casos especiais, como a remoção de antenas no início da lista, onde o ponteiro principal da lista precisa ser atualizado para o próximo elemento

Listagem de Antenas

Para facilitar a visualização e verificação das antenas na lista, foi criada uma função para listar as antenas presentes. Esta função percorre a lista e imprime a frequência e as coordenadas de cada antena, permitindo que o utilizador ou desenvolvedor visualize rapidamente as antenas registadas.

Visualização do Mapa de Antenas

Outro aspeto importante foi a criação de uma função para exibir as antenas num mapa, representando cada célula do mapa com um ponto (.), e posicionando as antenas de acordo com as suas coordenadas. A frequência de cada antena é representada pela letra correspondente, permitindo identificar facilmente as antenas no mapa. O mapa é impresso na consola, o que facilita a visualização da distribuição das antenas.

Como Funciona a Função `MostrarMapaAntenas`

A função `MostrarMapaAntenas` recebe dois parâmetros principais:

1. **lista**: Um ponteiro para a lista de antenas, onde cada antena possui informações sobre a sua frequência e coordenadas (x, y).
2. **numLinhas e numColunas**: Especificam as dimensões do mapa (ou seja, quantas linhas e colunas o mapa terá).

1. Alocação de Memória para o Mapa

O primeiro passo da função é alocar dinamicamente memória para o mapa. O mapa será representado como um vetor linear de caracteres, onde cada célula será representada por um ponto (.), que serve como um espaço vazio. A alocação de memória é feita da seguinte forma:

```
1 // Aloca memória para o mapa de uma vez só (um único bloco contínuo)
2 char* mapa = (char*)malloc(numLinhas * numColunas * sizeof(char));
3
4 if (mapa == NULL) {
5     printf("Erro ao alocar memória para o mapa.\n");
6     return; // Se falhar a alocação, sai da função
7 }
```

Aqui, alocamos um vetor com tamanho suficiente para armazenar todos os pontos do mapa (o número de células é `numLinhas * numColunas`). Caso a alocação falhe, a função imprime um erro e termina.

2. Inicialização do Mapa com Pontos (.)

Após alocar a memória, o mapa é inicializado com o valor . (ponto), que representará uma célula vazia no mapa. A função faz isso com um simples loop:

```
1 // Inicializa o mapa com pontos (.)
2 for (int i = 0; i < numLinhas * numColunas; i++) {
3     mapa[i] = '.';
4 }
```

3. Posicionamento das Antenas no Mapa

A função percorre a lista de antenas e posiciona cada uma delas no mapa. Cada antena tem coordenadas x e y, e a sua frequência deve ser colocada na célula correspondente no mapa.

- Como o vetor é linear, o mapa é acessado usando uma fórmula para converter as coordenadas 2D (linha e coluna) num índice único.
- O código ajusta as coordenadas das antenas (subtraindo 1) para que o índice comece de 0 (ou seja, converte coordenadas de 1-based para 0-based).

O bloco do código que faz isso é o seguinte:

```
1  Antena* aux = lista;
2  while (aux != NULL) { // Posiciona as antenas no mapa
3      int x = aux->x - 1; // Ajusta para o índice zero
4      int y = aux->y - 1; // Ajusta para o índice zero
5      mapa[y * numColunas + x] = aux->frequencia; // Calcula a posição no vetor linear
6      aux = aux->prox;
7  }
```

Aqui, percorremos a lista de antenas, ajustamos as suas coordenadas (subtraindo 1) e calculamos a posição correta no vetor linear, que será a célula onde a antena deve ser posicionada. A frequência da antena é colocada nessa posição.

4. Impressão do Mapa

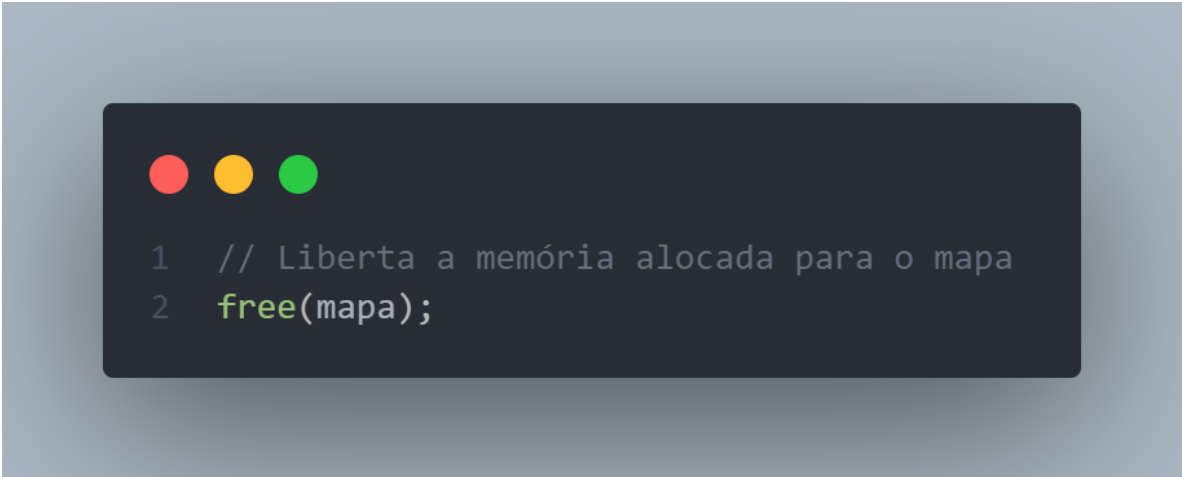
Uma vez que todas as antenas foram posicionadas, o mapa é impresso linha por linha para representar a visualização do espaço com as antenas.

```
1  // Imprime o mapa
2  for (int i = 0; i < numLinhas; i++) {
3      for (int j = 0; j < numColunas; j++) {
4          printf("%c", mapa[i * numColunas + j]); // Acessa os elementos do mapa
5      }
6      printf("\n");
7  }
```

Aqui, percorremos o mapa linha por linha e coluna por coluna, acessando os elementos do vetor linear de acordo com a fórmula $\text{mapa}[i * \text{numColunas} + j]$. A função imprime o caractere correspondente em cada posição do mapa, representando uma célula com o valor da frequência da antena ou com o ponto (.) caso a célula esteja vazia.

5. Libertação de Memória

Por fim, após o mapa ser impresso, a memória alocada para o vetor do mapa é liberada para evitar vazamentos de memória.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains two lines of code:

```
1 // Liberta a memória alocada para o mapa
2 free(mapa);
```

Destruição da Lista

Por fim, foi desenvolvida uma função para destruir a lista de antenas. Esta função percorre toda a lista e libera a memória alocada para cada antena, garantindo que não haja vazamentos de memória. Quando todas as antenas são removidas e a memória é liberada, a lista é efetivamente destruída.

Manipulação de Ficheiros

Agora vou explicar a parte da manipulação de ficheiros no meu código. O objetivo dessa função é ler um ficheiro de texto que contém dados sobre antenas e suas localizações, e depois armazenar essas informações numa lista ligada.

A função **LerFicheiro** tem como propósito abrir um ficheiro que contém as localizações das antenas no mapa e armazenar as antenas numa lista ligada. O ficheiro de entrada contém um mapa representado por caracteres, onde cada antena é representada por um caractere (por exemplo, **A**, **O**, etc.), enquanto as células vazias são representadas por um ponto (.).

Além de ler os dados das antenas, a função também calcula o número de linhas e colunas do mapa, o que é útil para visualizar ou manipular a representação do mapa mais tarde.

Como Funciona a Função LerFicheiro

A função **LerFicheiro** recebe os seguintes parâmetros:

1. **nomeFicheiro**: O nome do ficheiro que será lido.
2. **lista**: A lista de antenas onde as antenas extraídas do ficheiro serão armazenadas.
3. **numLinhas e numColunas**: Ponteiros para as variáveis onde os valores das dimensões do mapa (número de linhas e colunas) serão armazenados.

Passo a Passo da Função

1. Abertura do Ficheiro

O primeiro passo da função é tentar abrir o ficheiro utilizando a função **fopen**. Se o ficheiro não puder ser aberto, a função retorna a lista de antenas já existente, sem fazer alterações:

```
1 FILE* ficheiro = fopen(nomeFicheiro, "r"); // Abre o ficheiro para leitura
2 if (ficheiro == NULL) return lista; // Se não conseguir abrir, devolve a lista como está
```

2. Leitura do Ficheiro:

Depois de abrir o ficheiro, começo a ler o conteúdo caracter por caracter utilizando a função **fgetc(ficheiro)**. Para cada linha lida, a coordenada **y** é incrementada, e para cada caracter de cada linha, a coordenada **x** é incrementada.

- Quando a função encontra um caractere de nova linha (**\n**), isso significa que uma nova linha do mapa foi lida. Portanto, incremento a coordenada **y** e reseto a coordenada **x** para 1.

```
1 // Lê o ficheiro linha por linha
2 while ((c = fgetc(ficheiro)) != EOF) {
3     if (c == '\n') { // Nova linha, avançar para a próxima linha (aumenta a coordenada y)
4         y++;
5         x = 1; // Reseta a coordenada x ao passar para a próxima linha
6     }
```

3. Processamento das Antenas

Durante a leitura de cada caractere, verifico se ele não é um ponto (.), porque o ponto representa uma célula vazia do mapa. Caso o caractere seja diferente de ponto, isso significa que é uma antena, então chamo a função **InserirAntenaFim** para adicionar essa antena à lista. As coordenadas **x** e **y** são passadas para a função para garantir que a antena seja posicionada corretamente.

```
1 if (c != '.') { // Ignorar células vazias
2     // Adicionar antena à lista existente
3     Lista = InserirAntenaFim(Lista, c, x, y);
4 }
5 x++; // Avança a coordenada x
```

4. Determinação do Número de Colunas e Linhas

Enquanto percorre o ficheiro, também é preciso calcular o número máximo de colunas (em cada linha) para saber a largura do mapa. Se o número de colunas de uma linha for maior do que o número máximo encontrado até então, atualizo o valor de **maxColunas**.

```
1 // Atualizar o número de colunas, se for a linha com mais colunas
2 if (x > maxColunas) {
3     maxColunas = x - 1; // x foi incrementado após o uso, então subtraímos 1
4 }
```

5. Finalização da Leitura

Depois de ler todo o ficheiro, defino os valores de **numLinhas** e **numColunas** com base nas variáveis **y** e **maxColunas**, que indicam as dimensões do mapa. Em seguida, fecho o ficheiro com a função **fclose**.

```
1 // Definir as variáveis numLinhas e numColunas
2 *numLinhas = y; // O número de linhas será a última linha lida
3 *numColunas = maxColunas; // O número máximo de colunas encontrado
4
5 fclose(ficheiro);
```

Com esta função, consigo importar os dados do ficheiro de texto e construir a lista de antenas de forma eficiente. O cálculo das dimensões do mapa também me permite adaptar a visualização e manipulação dos dados conforme necessário, facilitando a gestão das antenas no sistema.

Resumo da Implementação

As operações principais de inserção, remoção, e listagem de antenas foram implementadas utilizando uma lista ligada. Essa estrutura de dados foi escolhida por ser eficiente em termos de inserção e remoção de elementos, além de permitir uma fácil gestão dinâmica da memória. A implementação foi projetada para ser modular e flexível, permitindo adicionar, remover e visualizar antenas de forma simples e eficiente.

A parte da visualização, com o mapa das antenas, foi uma das funcionalidades principais do projeto, permitindo representar graficamente a distribuição das antenas num formato acessível e intuitivo. Com isso, foi possível não apenas manipular as antenas, mas também visualizar as suas localizações de uma forma prática e compreensível.

Análise de Resultados

Durante o desenvolvimento do projeto, houve diversas decisões técnicas a tomar e desafios a ultrapassar. Esta secção serve para refletir sobre as dificuldades encontradas, abordagens alternativas consideradas e eventuais melhorias para trabalhos futuros.

Funcionalidades Não Implementadas

Um dos principais objetivos era analisar e representar o impacto das antenas, incluindo possíveis efeitos nefastos da sua proximidade em determinadas zonas. No entanto, não foi possível implementar essa funcionalidade de forma satisfatória. A dificuldade principal esteve na definição de um modelo matemático ou algoritmo adequado para deduzir a localização de efeitos nefastos das antenas.

Conclusão

Resumo dos Resultados Alcançados

O projeto permitiu a implementação de um sistema funcional para a leitura, armazenamento e visualização de antenas num mapa, utilizando estruturas de dados dinâmicas para facilitar a manipulação das informações. As principais funcionalidades desenvolvidas incluem:

- Leitura de um ficheiro contendo as localizações das antenas e armazenamento dos dados em listas ligadas.
- Inserção, remoção e listagem de antenas de forma eficiente.
- Visualização gráfica simplificada do mapa, facilitando a interpretação da distribuição das antenas.

Estas funcionalidades foram implementadas com sucesso, garantindo um sistema modular e flexível que pode ser facilmente expandido no futuro.

O Que Poderia Ser Melhorado

Embora o projeto tenha cumprido os objetivos iniciais, há algumas áreas que poderiam ser otimizadas:

- **Cálculo do impacto das antenas:** Como mencionado na secção anterior, não foi possível determinar os efeitos nefastos das antenas devido à falta de um modelo adequado. No futuro, seria interessante estudar formas de medir a interferência entre antenas e representar zonas de possível sobrecarga.
- **Melhoria no desempenho:** Embora as listas ligadas sejam adequadas para manipulação dinâmica, para um grande número de antenas, outras estruturas poderiam melhorar a eficiência da pesquisa e modificação dos dados.
- **Armazenamento mais eficiente:** Atualmente, o sistema apenas lê e manipula dados a partir de um ficheiro de texto. Um formato binário poderia melhorar o desempenho na leitura e escrita, reduzindo o tempo de acesso aos dados.

Possíveis Expansões do Trabalho

Para evoluir este projeto, algumas melhorias e novas funcionalidades poderiam ser implementadas:

1. Implementação do cálculo do impacto das antenas

- Desenvolver um modelo que avalie a interferência entre antenas próximas.

2. Leitura e escrita em ficheiro binário

- Criar um sistema que permita armazenar e carregar as antenas num formato binário, tornando o acesso mais eficiente.
- Permitir alternar entre leitura/escrita em ficheiro de texto e em ficheiro binário conforme necessário.

3. Alteração dos dados e preservação das modificações

- Permitir que o utilizador faça modificações no mapa e guarde essas alterações num novo ficheiro.
- Criar opções para salvar apenas as antenas modificadas, reduzindo o tamanho do ficheiro de saída.

Conclusão Final

O trabalho realizado demonstrou a viabilidade de manipular e visualizar dados de antenas de forma eficiente. Apesar de algumas limitações, o sistema desenvolvido pode servir como base para futuras melhorias, tornando-se um projeto mais robusto e capaz de lidar com análises mais complexas sobre a distribuição e impacto das antenas.