PROJETO GESTÃO DE ANTENAS

ESTRUTURAS DE DADOS AVANÇADAS

Aluno

Ana Rita Ranito (a31511)

Professor

Luís Ferreira

março de 2025

LISTA ABREVIATURAS E/OU SIGLAS

EDA - Estrutura de Dados Avançadas

EST – Escola Superior de Tecnologia

IPCA – Instituto Politécnico do Cávado e do Ave

LESI - Licenciatura em Engenharia de Sistemas Informáticos

UC - Unidade Curricular

INDICE DE FIGURAS

ÍNDICE

INTRODUÇÃO

O presente relatório pretende dar resposta a um objetivo de avaliação da Unidade Curricular (UC): Estrutura de Dados Avançadas (EDA), inserida no plano de estudos da Licenciatura em Engenharia de Sistemas Informáticos (LESI), da Escola Superior de Tecnologia (EST), do Instituto Politécnico do Cávado e do Ave (IPCA).

Este relatório objetiva documentar o processo de elaboração do Projeto Gestão de Antenas. O Projeto consiste no desenvolvimento de um programa em C, através de manipulação de estruturas de dados dinâmicas, armazenamento em ficheiro e modularização de código. Como objetivos pessoais, pretendo desenvolver competências no âmbito da realização de Projetos, desenvolver competências em linguagem C, bem como aprimorar a capacidade de comentar/documentar o código produzido.

No que concerne à documentação do código, fiz uso da ferramenta *Doxygen*. Esta é amplamente utilizada para gerar documentação a partir de comentários em código fonte, permite incluir descrições detalhadas de funções, classes e parâmetros diretamente nos arquivos de código, facilitando a manutenção e a legibilidade do Projeto.

Para o desenvolvimento deste trabalho foram aplicados os conhecimentos lecionados nas aulas de EDA, bem como saberes aprendidos nas UC´s lecionadas no primeiro semestre de LESI.

O relatório encontra-se dividido em quatro partes, das quais: Introdução, Projeto de Gestão de Antenas, Conclusão e Referências Bibliográficas. Na Introdução, o presente capítulo, é feita a apresentação do contexto, objetivos e estrutura do relatório. O capítulo denominado “Projeto Gestão de Antenas”, concretiza o corpo do trabalho, onde se aborda a estrutura do Projeto, definição de estruturas de dados, funcionalidades implementadas, descrição dos ficheiros do Projeto e diagramas ilustrativos elaborados ao longo do processo. Já na Conclusão, temos um resumo dos resultados obtidos e dificuldades sentidas. Por último, as Referências Bibliográficas, onde são apresentadas as bases teóricas utilizadas no desenvolvimento do trabalho.

1. PROJETO GESTÃO DE ANTENAS

1.1 Objetivos do Projeto

O principal objetivo deste Projeto é dar resposta à componente de avaliação da UC EDA. O Projeto consistiu em desenvolver um programa em C, com estruturas organizadas e funcionalidades bem definidas, objetivando manipulação de listas ligadas e ficheiros. No contexto do problema (ANEXO I), pretende-se desenvolver uma solução capaz de inserir uma nova antena na lista ligada, remover uma antena da mesma, deduzir as localizações com efeito nefasto e representá-las, bem como listar o resultado na consola.

1.2 Estrutura do Projeto

O Projeto foi organizado de forma modular, seguindo as boas práticas de programação em C, separando o código por responsabilidade e facilitando a manutenção e expansão do sistema. Deste modo, este Projeto foi subdividido da forma considerada mais adequada, os seguintes subcapítulos representam os ficheiros do mesmo.

1.2.1 estruturas.h

É o ficheiro de cabeçalho onde é definida a estrutura de dados principal, a estrutura da antena. Contem a frequência da antena (A ou O), coordenadas da antena (x e y) e um ponteiro para a próxima antena (formando uma lista ligada). Na Figura 1, podemos ver a estrutura utlizada para representar cada antena como um nó de uma lista ligada.

```
7   typedef struct Antena {
8       char frequencia;     /**< Frequência da antena */
9       int x, y;            /**< Coordenadas da antena (x, y) */
10      struct Antena *proxima; /**< Ponteiro para a próxima antena na lista ligada */
11  } Antena;
```

Figura 1 - Estrutura "Antena"

Além disto, é neste ficheiro que está definida a estrutura AntenaBinaria (Figura 2), que foi criada para representar uma antena de forma simplificada, contendo apenas os dados essenciais que devem ser guardados ou lidos de um ficheiro binário, como a frequência e as coordenadas da mesma na matriz (x e y).

```c
typedef struct {
    char frequencia;  /**< Frequência da antena */
    int x, y;         /**< Coordenadas da antena */
} AntenaBinaria;
```

Figura 2 - Estrutura "AntenaBinaria"

1.2.2 funcoes.h

É o ficheiro de cabeçalho com os protótipos das funções utilizadas em todo o Projeto. Permite ao main.c e outros módulos usarem as funções de manipulação de antenas e de cálculo de interferências.

1.2.3 funcoes.c

Neste documento, temos a implementação da criação, inserção, remoção e cálculo das interferências. Estão implementadas neste ficheiro as funções:

- criarAntena: Cria uma antena com os dados fornecidos e aloca memória.

- inserirAntena: Insere antenas no início da lista ligada.

- removerAntena: Remove uma antena específica.

- carregarDeFicheiro: Lê uma matriz completa a partir de ficheiro e identifica as antenas automaticamente.

- exibirAntena: Percorre e imprimi no terminal todos os elementos (nós) da lista ligada de antenas.

- calcularEdesenharMatriz: Calcula as interferências entre pares de antenas da mesma frequência e imprime a matriz no terminal.

- gravarEmBinario: Percorre a lista ligada de antenas e grava os dados (frequência e coordenadas) num ficheiro binário.

1.2.4 main.c

O ficheiro main.c é o ponto de entrada principal do programa. É responsável por carregar a matriz de antenas a partir do ficheiro de texto, construir dinamicamente uma lista ligada

de antenas com os dados lidos, exibir no terminal as antenas carregadas, calcular e imprimir a matriz e gravar a lista ligada num ficheiro binário.

1.2.5 Makefile

A Makefile (Figura 3), permite automatizar o processo de compilação e execução. Esta:

- Compila os ficheiros .c em ficheiros .o;

- Gera o executável final - programa;

- Permite limpar os ficheiros temporários.

```
# Objetivo principal: compilar o executável
all: programa

# Compila funcoes.c em funcoes.o
funcoes.o: funcoes.c funcoes.h estruturas.h
    gcc -c funcoes.c

# Compila main.c em main.o
main.o: main.c funcoes.h estruturas.h
    gcc -c main.c

# Cria o executável 'programa' a partir de main.o e funcoes.o
programa: main.o funcoes.o
    gcc main.o funcoes.o -o programa

# Para executar o programa
run:
    ./programa

# Para limpar os ficheiros-objeto e o executável
clean:
    rm -f *.o programa
```

Figura 3 – Makefile

1.3 As Antenas

As listas ligadas são estruturas de dados dinâmicas compostas por elementos (nós), onde cada elemento armazena dados e um ponteiro para o próximo nó da sequência. Uma das principais vantagens das listas ligadas é a flexibilidade na alocação de memória, permitindo a inserção e remoção eficiente de elementos sem necessidade de realocar ou reorganizar a estrutura (Cormen et al., 2009).

No contexto deste trabalho, a utilização de listas ligadas permitiu representar de forma eficiente um conjunto dinâmico de antenas lidas a partir de um ficheiro de texto, cujas quantidades e posições não eram previamente conhecidas. Esta abordagem facilitou a gestão da memória e a implementação de operações como inserir, remover e percorrer a lista (Weiss, 2014), além de tornar o código mais modular e adaptável a alterações. Assim, as listas ligadas mostraram-se uma solução eficaz para manipular um conjunto de dados com tamanho variável.

a) Criar a Antena

A função *malloc* (*memory allocation*) é utilizada para alocar dinamicamente memória durante a execução de um programa, esta alocação dinâmica permite reservar apenas a memória necessária quando esta for realmente utilizada, oferecendo maior flexibilidade e melhor gestão de recursos (Damas, 2015).

Na função criarAntena, o *malloc* foi utilizado para alocar memória suficiente para armazenar uma nova estrutura do tipo Antena. Como o número de antenas a serem criadas não é conhecido à partida (depende do conteúdo do ficheiro lido), é essencial que cada antena seja alocada individualmente em tempo de execução. Isto garante que o programa consiga criar tantos nós quantos forem necessários, sem desperdiçar memória ou impor limites fixos. Além disso, esta abordagem é indispensável para trabalhar com listas ligadas, já que cada novo nó precisa de existir independentemente dos outros e ser ligado dinamicamente à lista.

Na figura seguinte (Figura 4), podemos observar a função criarAntena.

```
Antena* criarAntena(char frequencia, int x, int y) {
    Antena* novaAntena = (Antena*)malloc(sizeof(Antena));
    novaAntena->frequencia = frequencia;
    novaAntena->x = x;
    novaAntena->y = y;
    novaAntena->proxima = NULL;
    return novaAntena;
}
```

Figura 4 - Função criarAntena

b) Inserir a Antena

As antenas são guardadas numa lista ligada. A função inserirAntena (Figura 5) insere sempre no início da lista:

```c
void inserirAntena(Antena **cabeca, char frequencia, int x, int y) {
    Antena* novaAntena = criarAntena(frequencia, x, y);
    novaAntena->proxima = *cabeca;
    *cabeca = novaAntena;
}
```

Figura 5 - Função inserirAntena

c) Remover a Antena

```c
void removerAntena(Antena **cabeca, char frequencia, int x, int y) {
    Antena *temp = *cabeca, *anterior = NULL;

    while (temp != NULL && (temp->frequencia != frequencia || temp->x != x || temp->y != y)) {
        anterior = temp;
        temp = temp->proxima;
    }
    if (temp == NULL) return;

    if (anterior == NULL)
        *cabeca = temp->proxima;
    else
        anterior->proxima = temp->proxima;

    free(temp);
}
```

Figura 6 - Função removerAntena

A função removerAntena (Figura 6) percorre a lista ligada à procura de uma antena com frequência e coordenadas específicas. Se for a primeira (cabeça), atualiza o ponteiro da cabeça. Caso contrário, liga o nó anterior ao seguinte. Por fim, liberta a memória da antena removida.

d) Exibir Antenas

```c
void exibirAntenas(Antena *cabeca) {
    printf("Lista de Antenas:\n");
    while (cabeca) {
        printf("Frequência: %c, Coordenadas: (%d, %d)\n", cabeca->frequencia, cabeca->x, cabeca->y);
        cabeca = cabeca->proxima;
    }
}
```

Figura 7 - Função exibirAntena

A função exibirAntena (Figura 7) percorre a lista ligada imprimindo a frequência e coordenadas de cada antena. É útil para verificar se a leitura do ficheiro e a inserção na lista ocorreram corretamente.

1.4 As Interferências

A parte mais desafiadora deste projeto, prendeu-se pela descoberta da lógica para detetar interferência entre antenas com a mesma frequência. A ideia baseia-se na criação de um vetor entre duas antenas iguais e na marcação dos extremos do vetor com " # ".

1.4.1 Cálculo do número de interferências

Objetivando perceber melhor a dinâmica do exercício proposto no Projeto, foram elaborados diversos esquemas. Na imagem seguinte (Figura 8), é possível observar o raciocínio que despoletou o cálculo do número de interferência, tendo em conta o número de antenas.

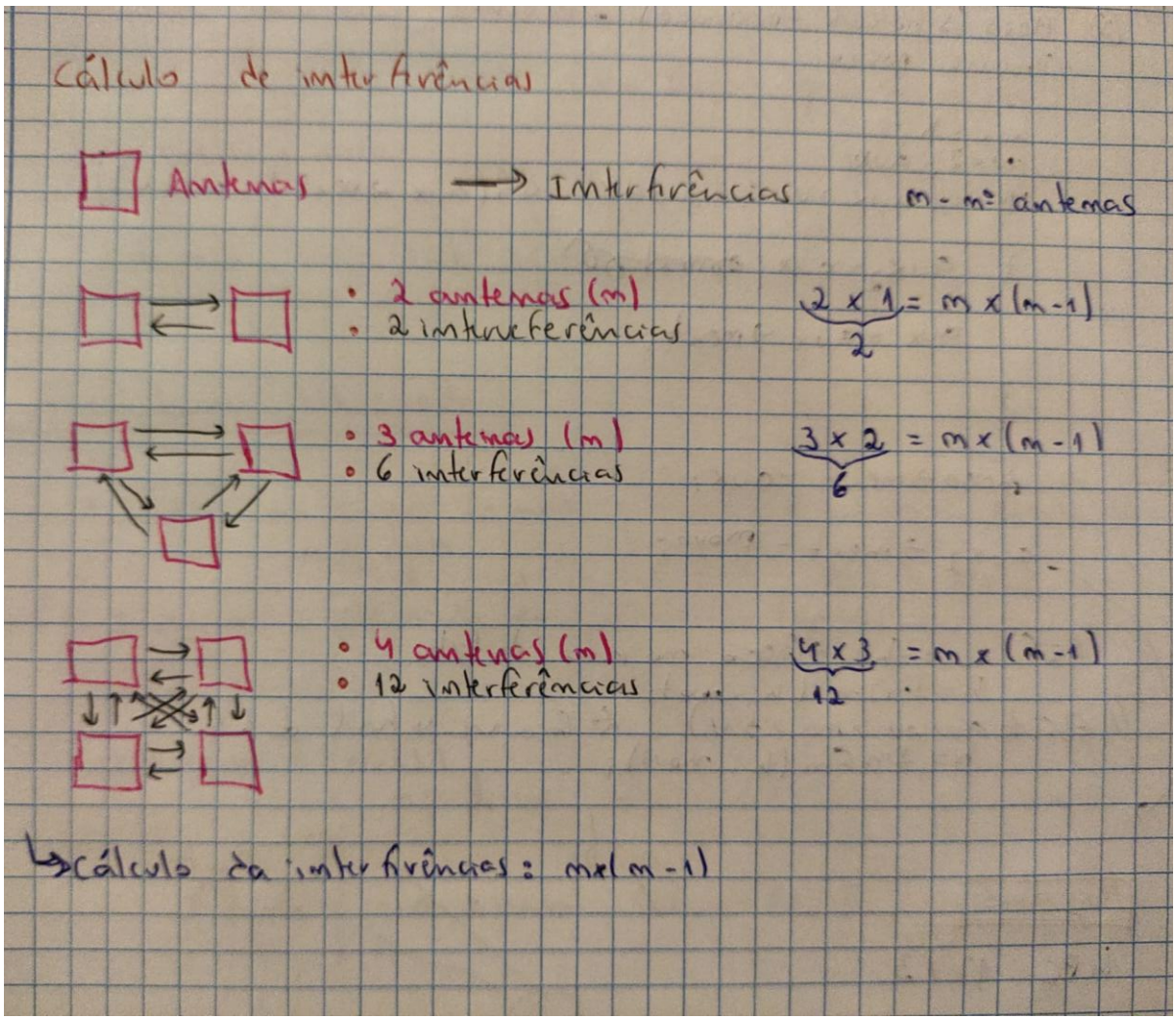


Figura 8 - Esquema de cálculo de interferências

1.4.2 Posicionamento de interferências

O sistema identifica interferências com base na relação geométrica entre pares de antenas que transmitem na mesma frequência. Para cada par de antenas A1(x1, y1) e A2(x2, y2) com a mesma frequência, calcula-se o vetor v = (dx, dy), onde:

- dx = x2 - x1 ➞ deslocamento horizontal.
- dy = y2 - y1 ➞ deslocamento vertical.

Esse vetor é aplicado simetricamente para gerar duas interferências:

- Uma a partir de A1 na direção inversa:

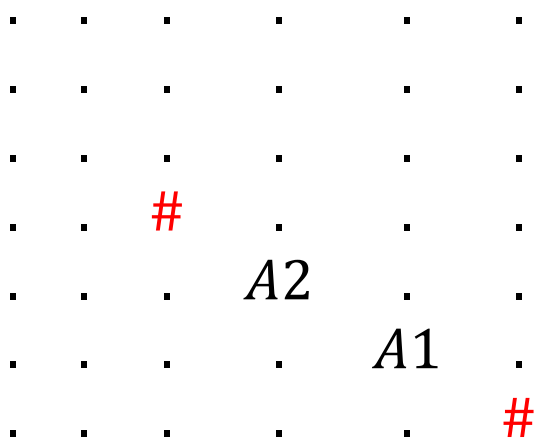
- Interferência 1 = (x1 - dx, y1 - dy) = A1 - v.

- Outra a partir de A2 na direção direta:

- Interferência 2 = (x2 + dx, y2 + dy) = A2 + v.

Os seguintes subcapítulos demonstram alguns dos esboços elaborados, para compreender melhor o cálculo vetorial por de trás deste Projeto.

1.4.2.1 Duas antenas



A1 em (5, 4)

A2 em (6, 5)

A1(5, 4) e A2(6, 5)

dx=6−5=1

dy=5−4=1

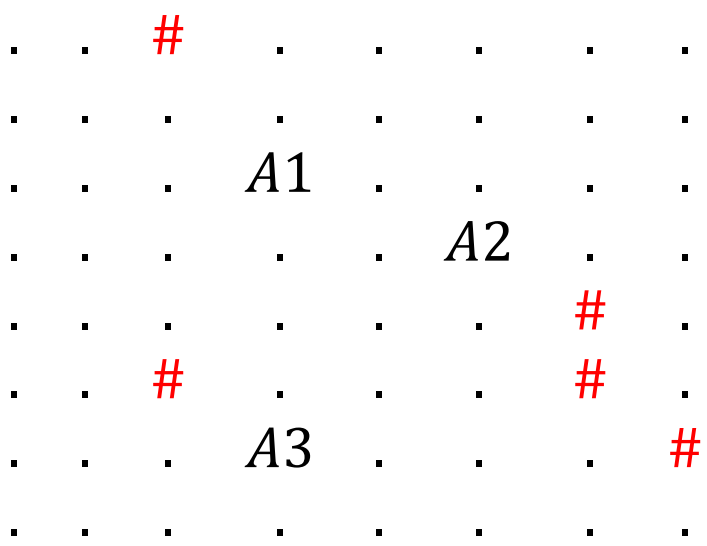
-Vetor: v = (1,1)

- Interferências

A1 – v = (5−1, 4−1) = (4,3)

A2 + v = (6+1,5+1) = (7,6)

1.4.2.2 Três antenas (ou mais)



A1 em (2,3)

A2 em (3,5)

A3 em (6,3)

A1(2,3) e A2(3,5)	A1(2,3) e A3(6,3)
dx = 5 - 3 = 2 dy = 3 - 2 = 1 - Vetor v = (2,1) - Interferências: A1 - v = (2 - 2, 3 - 1) = (0,2) A2 + v = (3 + 2, 5 + 1) = (5,6)	dx = 3 - 3 = 0 dy = 6 - 2 = 4 - Vetor v = (0,4) - Interferências: A1 - v = (2 - 0, 3 - 4) = (2, -1) (fora da matriz) A3 + v = (6 + 0, 3 + 4) = (6,7)
A2(3,5) e A3(6,3)	
dx = 3 - 5 = -2 dy = 6 - 3 = 3 - Vetor v = (-2,3) - Interferências: A2 - v = (3 + 2, 5 - 3) = (5,2) A3 + v = (6 - 2, 3 + 3) = (4,6)	

1.4.3 Concretização

```
for (Antena *a1 = cabeca; a1 != NULL; a1 = a1->proxima) {
    for (Antena *a2 = a1->proxima; a2 != NULL; a2 = a2->proxima) {
        if (a1->frequencia == a2->frequencia) {
            int dx = a2->x - a1->x;
            int dy = a2->y - a1->y;

            int x1 = a1->x - dx;
            int y1 = a1->y - dy;
            if (x1 >= 0 && x1 < colunas && y1 >= 0 && y1 < linhas && matriz[y1][x1] == '.')
                matriz[y1][x1] = '#';

            int x2 = a2->x + dx;
            int y2 = a2->y + dy;
            if (x2 >= 0 && x2 < colunas && y2 >= 0 && y2 < linhas && matriz[y2][x2] == '.')
                matriz[y2][x2] = '#';
        }
    }
}
```

Figura 9 - Cálculo "vetorial" das interferências

Percorre pares de antenas, compara cada antena (a1) com todas as antenas seguintes (a2), evitando repetições.

```
if (a1->frequencia == a2->frequencia)
```

Só interessa calcular interferência se ambas têm a mesma frequência.

```
int dx = a2->x - a1->x;
int dy = a2->y - a1->y;
```

Cálculo do vetor entre elas.

```
int x1 = a1->x - dx;
int y1 = a1->y - dy;
```

Estende o vetor inversamente.

```
int x2 = a2->x + dx;
int y2 = a2->y + dy;
```

Estende o vetor diretamente.

```
if (x1 >= 0 && x1 < colunas && y1 >= 0 && y1 < linhas && matriz[y1][x1] == '.')
    matriz[y1][x1] = '#';
if (x2 >= 0 && x2 < colunas && y2 >= 0 && y2 < linhas && matriz[y2][x2] == '.')
    matriz[y2][x2] = '#';
```

Verifica se:

- A posição está dentro dos limites da matriz;
- O local está vazio “ . “.

1.5 Manipulação de ficheiros

1.5.1 .txt

Os ficheiros de texto armazenam dados em formato “legível para humanos”.

Neste Projeto os ficheiros .txt permitiram-me representar graficamente uma matriz de antenas, onde cada caractere diferente de “ . ” representa uma antena. A função

carregarDeFicheiro lê o ficheiro linha a linha e cria nós da lista com base nesses caracteres.

```
Antena* carregarDeFicheiro(const char* nomeFicheiro, int *linhas, int *colunas);
```

Figura 10 - Função carregarDeFicheiro

Esta função, representada na Figura 10, permite-nos abrir o ficheiro em modo leitura "r" e ler o ficheiro linha por linha com "*fgets*". Em cada linha percorre cada caractere e se este for diferente de " . " assume que é uma antena e cria um nó com as coordenadas (x, y). Posteriormente, é usada a função inserirAntena que adiciona a antena à lista ligada. É feita a contagem do número de linhas e colunas para saber o tamanho da matriz, fecha-se o ficheiro e retorna-se o ponteiro para a lista ligada criada.

1.5.2 .bin

Por outro lado, os ficheiros binários armazenam dados no seu "formato bruto", ou seja, tal como estão representados na memória (em bytes). Não são legíveis por humanos.

A função gravarEmBinario grava cada antena (frequência e coordenadas) diretamente num ficheiro .bin como bytes. O que permite guardar a lista de antenas para ser carregada mais tarde, evitando a reconstrução a partir de texto.

```
void gravarEmBinario(Antena *cabeca, const char *nomeFicheiro);
```

Figura 11 - Função gravarEmBinario

A função gravarEmBinario (Figura 11), abre o ficheiro em modo binário "wb" e percorre a lista ligada. Para cada antena cria uma estrutura auxiliar (AntenaBinaria) com os campos frequência e coordenadas (x e y). Usa *fwrite* para escrever essa estrutura diretamente no ficheiro. Por fim, fecha o ficheiro após guardar todas as antenas.

Neste sentido, fazendo uso das vantagens destes dois tipos de ficheiros, utilizei o ficheiro .txt, que é ideal para entrada inicial, por ser fácil de criar, visualizar e alterar; e o ficheiro .bin, que se revela perfeito para guardar resultados ou estados internos do programa, sendo mais rápido e eficiente para reutilização.

1.6 Resultados obtidos

Para validar o correto funcionamento do programa, foram realizados alguns testes com ficheiros de entrada .txt, contendo diferentes distribuições de antenas. O objetivo foi verificar se o programa conseguia detetar as antenas corretamente, calcular as posições de interferência e imprimir a matriz de forma coerente.

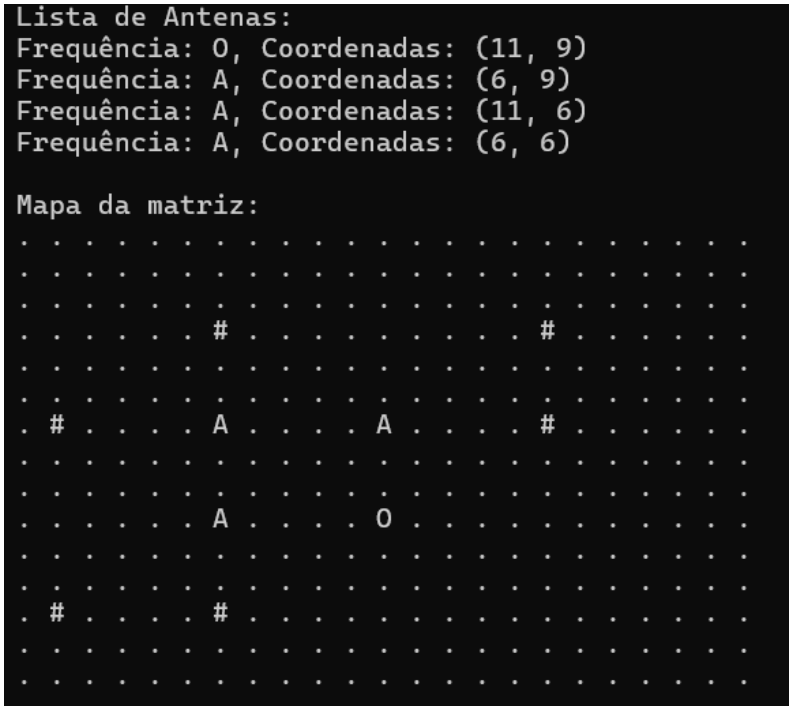


Figura 12 - Resultado 1

Num primeiro teste (Figura 12), após compilação e execução do programa, usando o documento .txt, foi possível observar o correto carregamento da lista ligada com quatro antenas, identificadas pelas suas frequências e coordenadas. Analisando o resultado obtido verificamos que três dessas antenas têm frequência "A" e estão posicionadas nas coordenadas (6,9), (11,6) e (6,6), enquanto uma antena adicional com frequência "O" encontra-se em (11,9).

No mapa da matriz gerado (Figura 12) é visível a representação dessas antenas nas respetivas posições, bem como as interferências geradas. Estas interferências são indicadas com o caractere "#" e surgem como resultado da análise vetorial entre pares de antenas "A". Como só temos uma antena de frequência "O", verificamos que foi corretamente ignorada para efeitos de interferência.

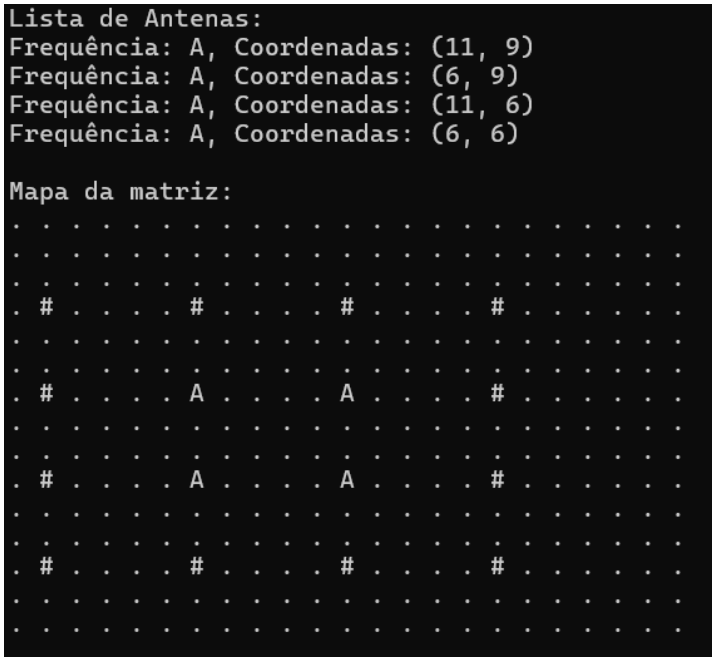


Figura 13 - Resultado 2

No segundo teste (Figura 13), o ficheiro de entrada foi alterado para conter quatro antenas com a mesma frequência "A". Podemos verificar que as coordenadas destas antenas são: (11,9), (6,9), (11,6) e (6,6). No que concerne ao cálculo de interferências, observamos que estão corretamente posicionadas e assinaladas por "#", bem como em número adequado [n x (n-1) = 4 x 3 = 12]. A saída gráfica confirmou que o algoritmo lida corretamente com múltiplos pares de antenas da mesma frequência.

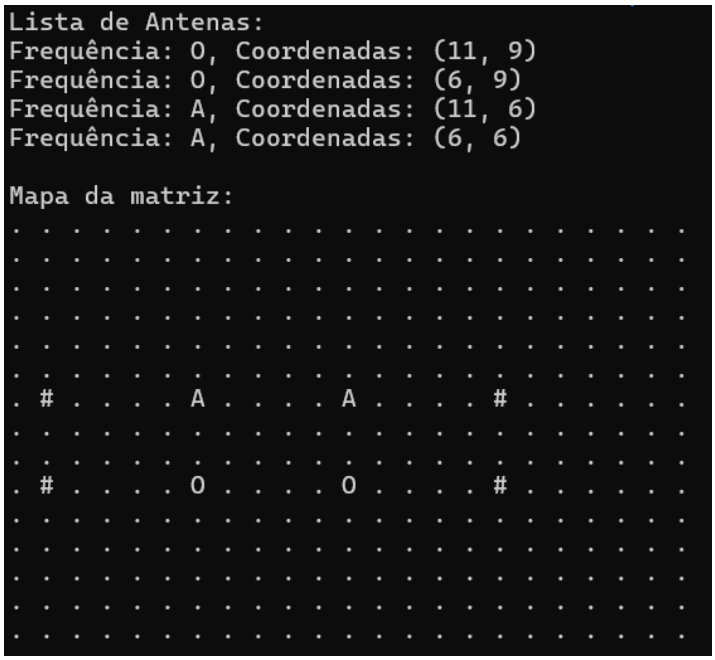


Figura 14 - Resultado 3

Para continuar a validação do programa, foi realizado um terceiro teste (Figura 14), modificando novamente o ficheiro de entrada de modo a alterar a distribuição e frequência das antenas. Neste novo cenário, foram colocadas duas antenas com frequência "O" (de coordenadas (11,9 e (6,9)) e duas com frequência "A" (de coordenadas (11,6) e (6,6)), totalizando quatro antenas. O objetivo foi verificar se o algoritmo era capaz de lidar com múltiplas frequências diferentes em simultâneo, mantendo a lógica de interferência apenas entre antenas com a mesma frequência.

Na saída obtida, foi possível confirmar que as antenas foram corretamente posicionadas na matriz e que as interferências foram calculadas de forma isolada para cada grupo de frequência. As antenas "A" produziram pontos de interferência marcados com "#", tal como esperado, enquanto as antenas "O" geraram as suas próprias também representadas com "#", mas sem se afetarem umas as outras.

Comparando com o teste anterior (Figura 13), onde todas as antenas tinham a mesma frequência "A", este novo teste demonstra claramente a capacidade do programa de tratar corretamente diferentes grupos de antenas independentemente, respeitando a lógica de interferência por frequência. Assim, este resultado reforça a robustez e a flexibilidade do algoritmo, que se adapta a diferentes ficheiros de entrada e respeita as restrições lógicas impostas pela frequência de cada antena, garantindo uma representação visual coerente da matriz e das interferências.

Comparando os resultados anteriores (Figura 12, Figura 13 e Figura 14), verifica-se que o programa responde corretamente a diferentes configurações e distribuições de antenas, sendo capaz de identificar e representar com precisão as situações de interferência. Este comportamento confirma a robustez da lógica de cálculo vetorial e a integridade do processamento da lista ligada.

Nos resultados mostrados anteriormente, a matriz tinha o mesmo tamanho para efeito de comparação, mas será que o programa é capaz de receber matrizes de tamanho variável? Para efeito de teste foram testadas matrizes de diferentes tamanhos, os resultados são apresentados a seguir.

Figura 15 - Resultado 4

Na figura 15, vemos vários exemplos de matrizes de tamanho variável, mudando tanto o número de linhas como de colunas. Estes testes demonstraram que o programa é capaz de adaptar-se automaticamente ao tamanho da matriz lida a partir do ficheiro .txt, identificando corretamente a sua largura e altura com base no conteúdo. A estrutura dinâmica da lista ligada, permitiu uma execução consistente independentemente da dimensão ou do número de antenas presentes.

As antenas foram corretamente interpretadas e posicionadas, e os cálculos de interferência mantiveram-se exatos mesmo em matrizes maiores ou com configurações assimétricas. Este comportamento confirma que a implementação é escalável e robusta, capaz de lidar tanto com exemplos simples como com cenários mais complexos e extensos, sem necessidade de alterar o código ou definir previamente limites fixos.

Em suma, os resultados obtidos ao longo dos testes comprovam que o programa cumpre os seus objetivos principais: leitura correta de antenas, cálculo coerente de interferências, visualização clara da matriz e suporte para tamanhos de entrada variáveis, encerrando assim este capítulo com uma validação completa da funcionalidade e fiabilidade da aplicação desenvolvida.

CONCLUSÃO

O Projeto permitiu-me consolidar conceitos de estruturas dinâmicas em C e manipulação de ficheiros. Os resultados obtidos com a execução do programa demonstram o correto exercício das funcionalidades desenvolvidas. Verificou-se a leitura correta das antenas a partir do ficheiro de texto, com a criação dinâmica de uma lista ligada contendo todos os elementos esperados. A função de cálculo e desenho da matriz revelou-se eficaz, permitindo a marcação correta das interferências entre antenas da mesma frequência, representadas através do caractere "#". Por fim, a impressão da matriz no terminal correspondeu ao comportamento esperado, com uma visualização clara e precisa da distribuição espacial das antenas e das respetivas interferências, confirmando a validade da abordagem implementada. Além disto, a estrutura modular adotada permitiu um desenvolvimento progressivo, facilitando a identificação e resolução de problemas. Deste modo, penso ter atingido todos os objetivos proposto no início deste Projeto.

Durante o desenvolvimento do Projeto surgiram algumas dificuldades relevantes que exigiram uma análise cuidadosa e ajustes constantes do código. Uma das principais foi a própria interpretação do enunciado do trabalho. Além disto, o cálculo das interferências e a gestão correta dos limites da matriz revelaram-se desafiantes, pois era essencial garantir que os vetores não os ultrapassassem. Estas dificuldades foram combatidas em grupo, com discussão e partilha de ideias entre colegas. Por outro lado, senti dificuldade na gestão do tempo para a realização deste Projeto, penso que a complexidade do mesmo exigia mais tempo disponível, o que por razões pessoais e laborais, não me foi possível.

Todo o Projeto desenvolvido encontra-se disponível em: https://github.com/AnaRanito/Trabalho-EDA.git.

REFERÊNCIAS BIBLIOGRÁFICAS

Aulas EDA do Professor Luís Ferreira, disponíveis em: https://github.com/luferIPCA/EDA-LESI-2024-2025.git

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Damas, L. (2015). *Linguagem C* (24th ed.). FCA.

Weiss, M. A. (2014). *Data Structures and Algorithm Analysis in C* (4th ed.). Pearson.

ANEXOS

ANEXO I – ENUNCIADO DO TRABALHO PRÁTICO (FASE 1)

Projeto de Avaliação

Estruturas de Dados Avançadas (EDA)

EST-IPCA

Barcelos

5 de março de 2025

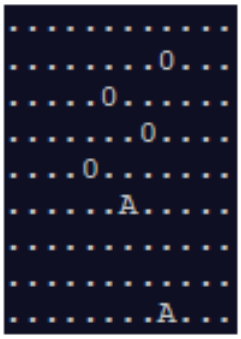
Motivação

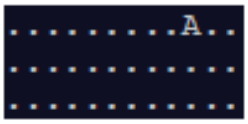
Este projeto de avaliação de realização individual da Unidade Curricular (UC) *Estruturas de Dados Avançadas* (EDA), integrada no 2° semestre do 1° ano, visa o reforço e a aplicação dos conhecimentos adquiridos ao longo do semestre.

Com este projeto de avaliação pretende-se sedimentar 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 clássica de desenvolvimento de soluções de software para um problema de dimensão média. A implementação das soluções deverá considerar estruturas de dados dinâmicas, armazenamento em ficheiro, modularização e apresentar uma estruturação e documentação com Doxygen.

Contextualização

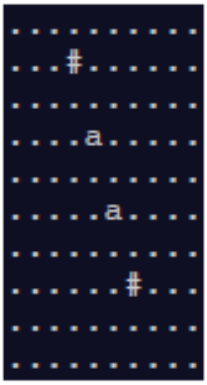
Pretende-se considerar uma cidade com várias antenas. Cada antena é sintonizada numa frequência específica indicada por um caracter. O mapa das antenas com as suas localizações (coordenadas na matriz) e frequências é representado através de uma matriz. Por exemplo:



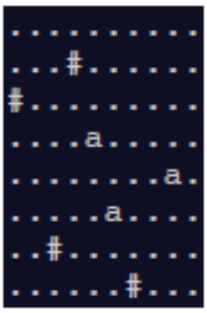


Considere que o sinal de cada antena aplica um efeito nefasto em localizações específicas L com base nas frequências de ressonância das antenas. Em particular, o efeito nefasto ocorre em qualquer localização L que esteja perfeitamente alinhada com duas antenas da mesma frequência - mas apenas quando uma das antenas está duas vezes mais distante que a outra. Isso significa que para qualquer par de antenas com a mesma frequência, existem duas localizações, uma de cada lado das antenas.

A título de exemplo, para duas antenas com frequência a, as localizações com efeito nefasto encontram-se representadas com #:

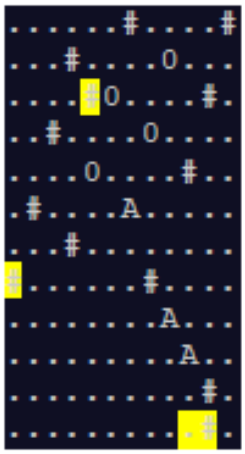


Adicionar uma terceira antena com a mesma frequência cria várias localizações adicionais com efeito nefasto:



Antenas com frequências diferentes (caracteres diferentes) não criam localizações com efeito nefasto. Localizações com efeito nefasto podem ocorrer em locais que contêm antenas. Uma localização com efeito nefasto pode surgir na sequência das várias combinações de antenas em simultâneo.

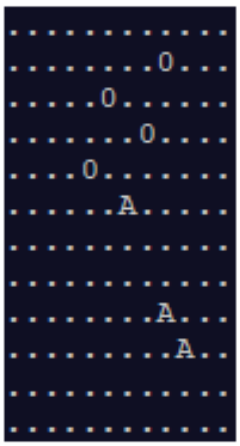
O primeiro exemplo tem antenas com duas frequências diferentes (A e O), dando origem as localizações com efeito nefasto seguintes:



Fase 1 - Listas ligadas

Considerando a contextualização supra-mencionada, procure implementar as funcionalidades seguintes:

1. Definição de uma estrutura de dados *ED*, para a representação das antenas, sob a forma de uma lista ligada simples. Cada registo da lista ligada deverá conter a frequência de ressonância de uma antena e suas coordenadas;
2. Carregamento para uma estrutura de dados *ED* dos dados das antenas constantes num ficheiro de texto. A operação deverá considerar matrizes de caracteres com qualquer dimensão. A título de exemplo, o ficheiro de texto deverá respeitar o formato seguinte:



3. Implementar operações de manipulação da lista ligada do tipo *ED*, incluindo:
    a. Inserção de uma nova antena na lista ligada;
    b. Remoção de uma antena constante na lista ligada;
    c. Dedução automática das localizações com efeito nefasto e respetiva representação sob a forma de uma lista ligada;
    d. Listagem de forma tabular na consola das antenas e localizações com efeito nefasto.