Instituto Politécnico do Cávado e do Ave

Escola Superior de Tecnologia

Licenciatura em

Engenharia de Sistemas Informáticos

Trabalho Prático (Fase 1)

Fábio Rafael Gomes Costa – a22997

Unidade Curricular:

Estruturas de Dados Avançadas

Docente:

Dr. Luís Gonzaga Martins Ferreira

Barcelos, março de 2025

Índice

Índice de Figuras

1. Introdução

1.1 Contextualização

No âmbito da Unidade Curricular (UC) de Estruturas de Dados Avançadas (EDA), inserida no 2º semestre do 1º ano do curso, foi-me atribuída a realização de um projeto prático como instrumento de avaliação, sob a orientação do docente Dr. Luís G. Ferreira. Este projeto tem como principal objetivo a aplicação e consolidação dos conhecimentos adquiridos ao longo do semestre, através da implementação e manipulação de estruturas de dados dinâmicas na linguagem de programação C.

1.2 Motivação/Pretensões e Objetivos

O desenvolvimento deste projeto visa aprofundar a compreensão sobre o uso de estruturas de dados dinâmicas, destacando a importância da sua correta definição, implementação e manipulação. Além disso, pretende-se estimular a capacidade de resolução de problemas computacionais, reforçando boas práticas de programação, como modularização, documentação com Doxygen e armazenamento eficiente de dados em ficheiros.

Na Fase 1, será dada ênfase à construção e manipulação de listas ligadas, abordando operações fundamentais, como inserção e remoção de elementos, bem como a identificação de padrões específicos dentro dos dados armazenados. Esta abordagem permitirá compreender melhor as vantagens e desafios associados a esta estrutura de dados, preparando o terreno para a implementação de soluções mais complexas nas fases subsequentes do projeto.

1.3 Estrutura do Documento

Este relatório foi pensado/estruturado no seguinte formato, primeiro a capa o, depois o índice (índice de tópicos e de imagens e/ou tabelas), apos iniciasse a introdução, de seguida o desenvolvimento, posteriormente a conclusão, e por fim a Webgrafia, onde são listados todos os links consultados na elaboração do trabalho.

2. Problemas

"O seguinte texto e as seguintes imagens foram retiradas do Enunciado do trabalho pratico disponibilizado pelo docente na plataforma académica Moodle, para quem tiver interesse de consultar o documento na integra, esta disponibilizada uma copia no repositório do GitHub do projeto."

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:

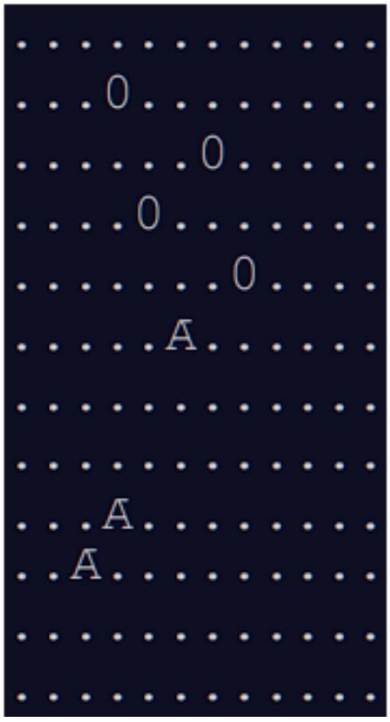

Figura 1 - Matriz Problema I

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 #:

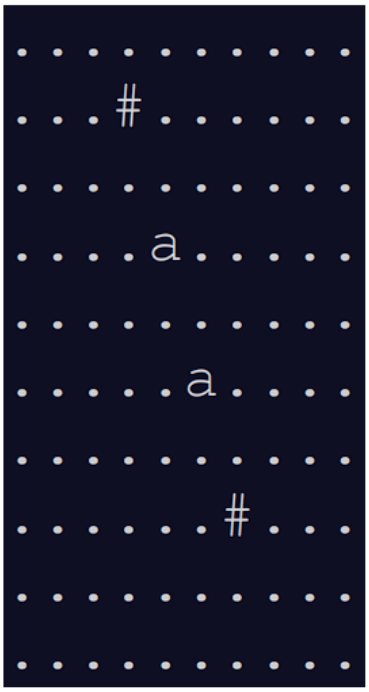

Figura 2 - Matriz Problema II

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

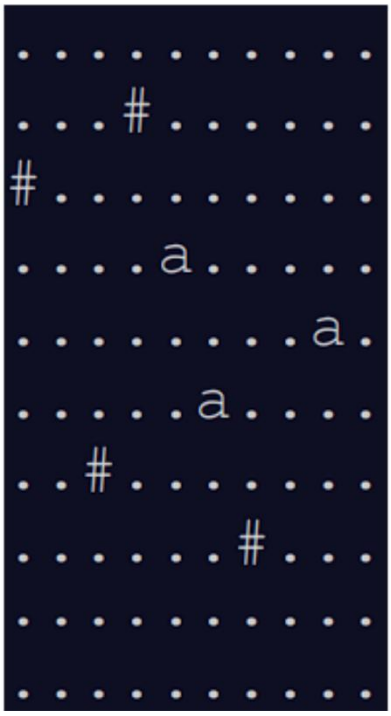

Figura 3 - Matriz Problema III

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:

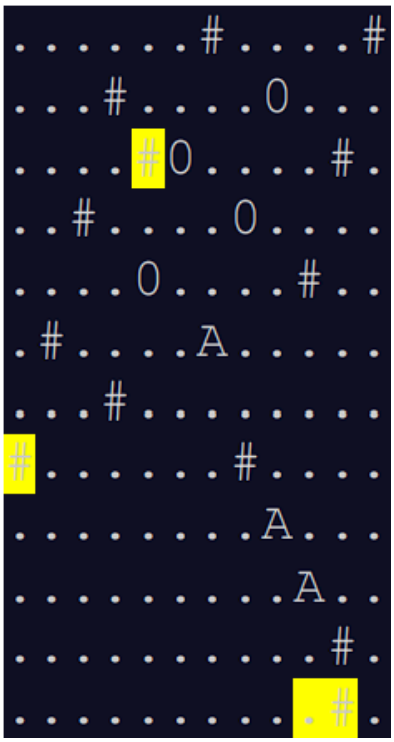

Figura 4 - Matriz Problema IV

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:

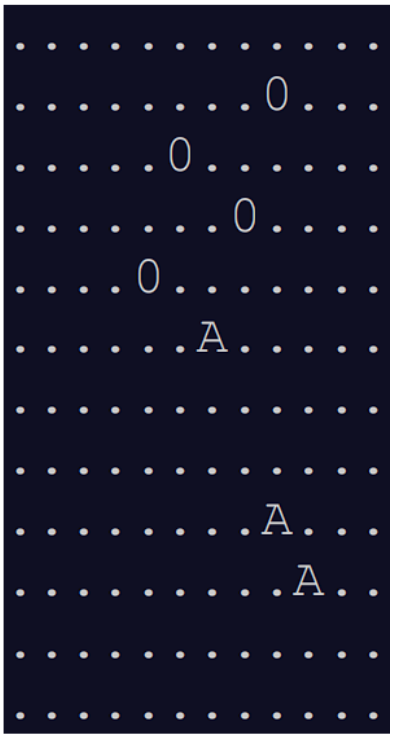

Figura 5 – Matriz Problema V

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.

3. Resolução dos Problemas

3.1 Resolução do Problema 2.

A função LerLista lê os dados de um ficheiro e armazena a informação numa matriz e numa lista ligada de antenas. O ficheiro de entrada contém a representação da matriz, onde cada caractere indica a presença ou ausência de uma antena.

O processo começa com a abertura do ficheiro em modo de leitura. Caso ocorra um erro ao abrir o ficheiro, a função exibe uma mensagem de erro e termina. Em seguida, a função percorre o ficheiro linha a linha, armazenando os dados na matriz e identificando as antenas presentes. Sempre que um caractere diferente de "." é encontrado, a função inserirAntena é chamada para adicionar a antena à lista ligada.

Além disso, a função ajusta as dimensões da matriz, garantindo que todas as colunas não utilizadas sejam preenchidas com "." . No final, o ficheiro é fechado e a função retorna a lista de antenas extraída do ficheiro.

```
ANT* LerLista(const char* nomeFicheiro, char matriz[MAX_LINHAS][MAX_COLUNAS],
int* linhas, int* colunas) {
    FILE* ficheiro = fopen(nomeFicheiro, "r");
    if (ficheiro == NULL) {
        perror("Erro ao abrir o ficheiro");
        return NULL;
    }

    ANT* lista = NULL;
    char linha[MAX_COLUNAS];
    int y = 0;
    int maxColunas = 0;

    while (fgets(linha, sizeof(linha), ficheiro)) {
        int tamLinha = strlen(linha);
        if (linha[tamLinha - 1] == '\n') {
            linha[tamLinha - 1] = '\0'; // Remover quebra de linha
            tamLinha--;
        }

        if (tamLinha > maxColunas) {
            maxColunas = tamLinha;
        }

        for (int x = 0; x < tamLinha; x++) {
            matriz[y][x] = linha[x]; // Armazena na matriz
            if (linha[x] != '.') {
                inserirAntena(&lista, linha[x], x, y);
            }
        }

        // Preencher com '.' os espaços não usados
```

```
        for (int x = tamLinha; x < MAX_COLUNAS; x++) {
            matriz[y][x] = '.';
        }
        y++;
    }

    *linhas = y;
    *colunas = maxColunas;
    fclose(ficheiro);
    return lista;
}
```

3.2 Resolução do Problema 3.1.

A função inserirAntena é responsável por adicionar uma nova antena à lista ligada de antenas. Sempre que uma antena é identificada, esta função é chamada para armazenar a sua informação.

O processo inicia-se com a alocação de memória para um novo nó da lista. Caso a alocação falhe, é exibida uma mensagem de erro e a função termina. Caso contrário, os dados da antena (frequência e coordenadas) são armazenados na nova estrutura.

A nova antena é então inserida no início da lista ligada, garantindo que a lista se mantém atualizada com todas as antenas detetadas.

```
void inserirAntena(ANT** lista, char freq, int x, int y) {
    ANT* novaAntena = (ANT*)malloc(sizeof(ANT));
    if (novaAntena == NULL) {
        perror("Erro ao alocar memória");
        return;
    }
    novaAntena->freqAntena = freq;
    novaAntena->x = x;
    novaAntena->y = y;
    novaAntena->proxAntena = *lista;
    *lista = novaAntena;
}
```

3.3 Resolução do Problema 3.a.

Esta função atualiza a matriz e a lista de antenas ao adicionar uma nova antena. Para isso, percorre a lista de antenas existente e verifica se já existe uma antena na posição especificada. Se existir, a frequência da matriz é atualizada com a nova frequência fornecida. Caso contrário, a função insere a nova antena na lista e atualiza a matriz com a respetiva frequência.

```
void atualizarMatriz(ANT** lista, char freq, int x, int y, char
matriz[MAX_LINHAS][MAX_COLUNAS])
{
    ANT* temp = *lista;
    while (temp != NULL)
    {
        if (temp->x == x && temp->y == y)
        {
            matriz[y][x] = freq;
            return;
        }
        temp = temp->proxAntena;
    }
    inserirAntena(lista, freq, x, y);
    matriz[y][x] = freq;
}
```

3.4 Resolução do Problema 3.b.

A função removerAntena remove uma antena da lista e atualiza a matriz. Para isso, percorre a lista ligada de antenas até encontrar a antena na posição especificada. Se a antena for encontrada, é removida da lista, garantindo a correta ligação dos elementos restantes. Em seguida, a matriz é atualizada, substituindo a posição da antena removida por um caractere padrão ('.'). Por fim, a memória alocada para a antena é libertada para evitar fugas de memória.

```
void removerAntena(ANT** lista, int x, int y, char
matriz[MAX_LINHAS][MAX_COLUNAS])
{
    ANT* temp = *lista;
    ANT* anterior = NULL;
    while (temp != NULL)
    {
        if (temp->x == x && temp->y == y)
        {
            if (anterior == NULL)
            {
                *lista = temp->proxAntena;
            }
            else
            {
                anterior->proxAntena = temp->proxAntena;
            }
            matriz[y][x] = '.';
            free(temp);
            return;
        }
        anterior = temp;
        temp = temp->proxAntena;
    }
}
```

3.5 Resolução do Problema 3.c.

A função matrizNefastos percorre a matriz para identificar frequências repetidas em posições distintas. Quando encontra duas coordenadas com a mesma frequência, calcula o cruzamento entre essas posições para determinar as localizações dos efeitos nefastos.

O algoritmo analisa a diferença entre as coordenadas das frequências iguais e projeta os pontos de interferência. Se esses pontos estiverem dentro dos limites da matriz e não contiverem uma antena, são marcados com o símbolo #, representando um efeito nefasto. Esses pontos são então inseridos numa lista ligada, que é devolvida no final da execução da função.

```
NEF* matrizNefastos(char matriz[MAX_LINHAS][MAX_COLUNAS], int linhas, int
colunas) {
    int menorx, menory, maiorx, maiory, difx, dify;
    NEF* lista = NULL;
    for (int y = 0; y < linhas; y++) {
        for (int x = 0; x < colunas; x++) {
            if (matriz[y][x] != '.' && matriz[y][x] != '#') {
                for (int y2 = 0; y2 < linhas; y2++) {
                    for (int x2 = 0; x2 < colunas; x2++) {
                        if (matriz[y][x] == matriz[y2][x2] && (y != y2 || x !=
x2)) {

                            if (x > x2)
                            {
                                difx = x - x2;
                                menorx = x2 - difx;
                                maiorx = x + difx;
                            }
                            else
                            {
                                difx = x2 - x;
                                menorx = x - difx;
                                maiorx = x2 + difx;
                            }
                            if (y > y2)
                            {
                                dify = y - y2;
                                menory = y2 - dify;
                                maiory = y + dify;
                            }
                            else
                            {
                                dify = y2 - y;
                                menory = y - dify;
                                maiory = y2 + dify;
                            }
                            if (x > x2 && y > y2 || x < x2 && y < y2)
                            {
                                if (matriz[menory][menorx] == '.' && menorx >=
0 && menory >= 0)
                                {
```

```
                                matriz[menory][menorx] = '#';
                                inserirNefasto(&lista, menorx, menory);
                            }
                            if (matriz[maiory][maiorx] == '.' && maiorx <=
colunas && maiory <= linhas)
                            {
                                matriz[maiory][maiorx] = '#';
                                inserirNefasto(&lista, maiorx, maiory);
                            }
                        }
                        else if (x > x2 && y < y2 || x<x2 && y>y2)
                        {
                            if (matriz[menory][maiorx] == '.' && menorx >=
0 && menory >= 0)
                            {
                                matriz[menory][maiorx] = '#';
                                inserirNefasto(&lista, maiorx, menory);
                            }
                            if (matriz[maiory][menorx] == '.' && maiorx <=
colunas && maiory <= linhas)
                            {
                                matriz[maiory][menorx] = '#';
                                inserirNefasto(&lista, menorx, maiory);
                            }
                        }
                    }
                }
            }
        }
    }
    return lista;
}
```

A função inserirNefasto é responsável por armazenar na lista ligada os pontos identificados como efeitos nefastos durante a execução da função matrizNefastos. Sempre que é encontrada uma localização nefasta, esta função é chamada para inserir as coordenadas na lista.

O processo inicia-se com a alocação de memória para um novo elemento da lista. Se a alocação falhar, é exibida uma mensagem de erro e a função termina. Caso contrário, os valores das coordenadas são atribuídos ao novo elemento, que é inserido no início da lista ligada. Assim, a lista de efeitos nefastos é continuamente atualizada com novas ocorrências identificadas na matriz.

```
void inserirNefasto(NEF** lista, int x, int y) {
    NEF* novoNefasto = (NEF*)malloc(sizeof(NEF));
    if (novoNefasto == NULL) {
```

```
        perror("Erro ao alocar memória");
        return;
    }
    novoNefasto->x = x;
    novoNefasto->y = y;
    novoNefasto->proxNef = *lista;
    *lista = novoNefasto;
}
```

3.6 Resolução do Problema 3.d.

A função apresentarLista percorre a lista ligada de antenas e exibe no ecrã a frequência e as coordenadas de cada uma. Caso a lista esteja vazia, é apresentada uma mensagem informativa.

```
void apresentarLista(ANT* lista) {
    if (lista == NULL) {
        printf("Lista vazia\n");
        return;
    }
    printf("\nLista de Antenas:\n");
    while (lista != NULL) {
        printf("Antena: %c | Coordenadas: (%d, %d)\n", lista->freqAntena,
lista->x, lista->y);
        lista = lista->proxAntena;
    }
}
```

A função apresentarListaNef percorre a lista de efeitos nefastos e exibe as suas coordenadas. Tal como na função anterior, se a lista estiver vazia, é apresentada uma mensagem a indicar esse estado.

```
void apresentarListaNef(NEF* lista) {

    if (lista == NULL) {
        printf("Lista vazia\n");
        return;
    }

    printf("\nLista de Nefastos:\n");
    while (lista != NULL) {
        printf("Nefastos Coordenadas: (%d, %d)\n", lista->x, lista->y);
        lista = lista->proxNef;
    }
}
```

A função apresentarMatriz imprime no ecrã o estado atual da matriz, representando a distribuição das antenas e dos efeitos nefastos no espaço definido. Cada posição da matriz é impressa de forma estruturada para facilitar a leitura e análise visual.

```c
void apresentarMatriz(char matriz[MAX_LINHAS][MAX_COLUNAS], int linhas, int
colunas) {
    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            printf("%c ", matriz[i][j]);
        }
        printf("\n");
    }
}
```

4. Repositório GitHub

O repositório GitHub com o relatório da Fase 1 do projeto de Estruturas de Dados e Algoritmos (EDA) pode ser acedido através do seguinte link:

https://github.com/fabiocosta191/Projeto_EDA

Neste repositório, poderão ser consultados todos os detalhes e documentação relacionados com a primeira fase do projeto, incluindo as análises, implementações e conclusões até ao momento.

5. Considerações Finais

Ao longo da realização deste trabalho prático, aprofundei a minha compreensão sobre a implementação e manipulação de estruturas de dados dinâmicas, em particular listas ligadas, na linguagem de programação C. Enfrentei desafios relacionados com a gestão eficiente da memória e a organização dos dados, garantindo um desempenho otimizado e uma estrutura modular no desenvolvimento do código.

A implementação das diferentes operações, como a inserção, remoção e listagem de antenas, permitiu-me consolidar conhecimentos sobre a estruturação de dados e perceber como a escolha da estrutura adequada pode influenciar diretamente a eficiência da solução. Além disso, a análise das localizações com efeito nefasto reforçou a importância da correta modelação do problema, evidenciando a necessidade de um raciocínio lógico e estruturado.

Este projeto foi fundamental para fortalecer as minhas competências no desenvolvimento de soluções robustas e escaláveis, preparando-me para desafios mais complexos no campo da programação e da otimização de algoritmos. A experiência adquirida será, sem dúvida, uma mais-valia para a minha evolução académica e profissional, proporcionando-me uma base sólida para aplicar conceitos avançados em programação.

Em suma, considero que este trabalho foi uma excelente oportunidade para consolidar e aplicar os conhecimentos adquiridos na unidade curricular de Estruturas de Dados Avançadas, cumprindo os objetivos propostos e contribuindo significativamente para o meu desenvolvimento enquanto programador.

6. Referencias bibliográfica

(43) Linguagem C - Português - YouTube. (n.d.). Retrieved March 29, 2025, from https://www.youtube.com/@linguagemc-portugues1473/videos

O manual do iniciante em C: aprenda o básico sobre a linguagem de programação C em apenas algumas horas. (n.d.). Retrieved March 29, 2025, from https://www.freecodecamp.org/portuguese/news/o-manual-do-iniciante-em-c-aprenda-o-basico-sobre-a-linguagem-de-programacao-c-em-apenas-algumas-horas/

Para que serve o struct do C? O que significa a->b? (n.d.). Retrieved March 29, 2025, from https://www.ime.usp.br/~pf/algoritmos/aulas/stru.html

Programação C - Structs. (n.d.). Retrieved March 29, 2025, from https://www.inf.pucrs.br/~pinho/LaproI/Structs/Structs.htm

Programação com apontadores. (n.d.). Retrieved March 29, 2025, from https://www.dcc.fc.up.pt/~pbv/aulas/progimp/teoricas/teorica23.html