Instituto Politécnico do Cávado e do Ave

Escola Superior de Tecnologia

Estrutura de Dados Avançada

Licenciatura em Engenharia de Sistemas Informáticos

Trabalho Prático

1ºFase

Dani Carvalho da Cruz– a23016

março de 2025

Índice

Índice de Ilustrações

1. Introdução

No âmbito da cadeira de Estrutura de Dados Avançada, lecionada por Luís Ferreira no curso de Engenharia de Sistemas Informáticos, foi nos pedido para desenvolver uma solução para uma cidade, onde existe antenas com determinadas coordenadas (x,y), frequências associadas à mesma e ainda localização nefastas que são originadas por posições específicas de antenas com frequências iguais.

Para o desenvolvimento deste trabalho, a solução, deveria ser sob a forma de uma lista ligada, onde contêm todas as informações necessárias das antenas. Sob as mesmas, têm que ser implementadas operações para inserir e eliminar antenas.

Carregamento de um ficheiro .txt, contendo uma matriz preenchida por posições, antenas e localizações nefastas.

Por fim, implementar também uma listagem tabular na consola com as antenas e localizações nefastas.

2. Perceber o problema

Inicialmente, temos que perceber o nosso problema antes de tentar realizar uma solução. Sabemos que, temos uma Matriz e, portanto, necessitamos de ter linhas e colunas.

Nessa matriz temos antenas associadas que possuem

- Frequência
- Localizações Nefastas
- Coordenadas

2.1 Localizações Nefastas

Aqui surge o primeiro problema que enfrentei. Pela descrição do enunciado uma localização nefasta ocorre, quando:

- Uma localização nefasta entre antenas só ocorre, se tiverem todas a mesma frequência;
- Uma localização não pode ocorrer se a distância entre as antenas for menor que 2.
- Para cada comparação entre antenas de mesma frequência existem um par de localizações, uma para cada antena.
- As coordenadas são dadas pela distância entre antenas.

O problema começa a surgir quando não consigo arranjar forma de calcular as posições, ou seja, onde ficariam as localizações?

Pelas imagens presentes no enunciado, percebemos que o par de localização nefasta, acaba por ser o "simétrico", ou seja, se a para uma distância entre A1 e A2 é 3 níveis acima e 1 nível para a esquerda, por exemplo.

Localização nefasta de A1, seria dada por, desde da posição da mesma subimos 3 níveis e 1 nível para a esquerda.
Para A2, seria o simétrico e portanto 3 níveis para baixo e 1 nível para a direita.

Por fim, devemos ainda ter cuidado que essas coordenadas não poderão também exceder a matriz ou coincidir com outra antena já existente.

2.2 Cálculo das coordenadas das localizações nefastas

Para o cálculo, depois de várias tentativas para arranjar uma solução uniforme que fosse igual às posições do enunciado, cheguei ao seguinte:

- Selecionar apenas antenas da mesma frequência;
- Fazer a subtração utilizando função abs, pois caso a antena atual inserida possua valores menores que a antena anterior, ou vice versa, não poderemos aceitar valores negativos.
- Ou seja, A1(x1,y1) e A2(x2,y2). Então teremos x3 = |x2-x1| e y3 = |y2-y1|.
- Depois para o cálculo das coordenadas das localizações nefastas, faço a seguinte condição: Se x1+x3 = x2 então as posições de x são dadas por, # = x1 – x3. Depois se esta condição for respeitada fazemos o mesmo para o y, logo se y1 + y3 = y2 então # = y1 – y3.
- Esta condição provém, pois se não, as localizações nefastas para as antenas seriam as mesmas.

3. Estrutura Planeada

Agora, com o problema totalmente percebido podemos começar a pensar como iremos a fazer a implementação inicial.

Para isto, recorri à estrutura de pensamento de POO, que apesar de serem programações distintas, pois uma é orientada a objetos, ajuda bastante para implementação da solução. Portanto, em POO temos classes em EDA, temos algo semelhante as structs.

3.1 Matriz

```
typedef struct
{
    int tamMatriz[2];
    struct ListaAntenas* listaAntenas;
}Matriz;
```

Figura 1: Struct *"Matriz"*

Matriz então terá:

- Array de tamanho 2 para inteiros, sendo posição 0 linhas e 1 colunas.
- Apontador do tipo ListaAntena., que apontará para uma estrutura de antenas, onde estas serão inseridas.

3.2 Lista Antena

Lista Antena então terá:

- Array de tamanho 2 para inteiros, sendo posição 0 linhas e 1 colunas.
- Apontador do tipo Antena, que apontará para a primeira antena da Lista Antenas.

```
typedef struct ListaAntenas
{
    int tamMatriz[2];
    struct Antena* primAntena;
}ListaAntenas;
```

Figura 2: Struct *"ListaAntena"*

3.3 Antena

```
typedef struct
{
    int posAntena[2];
    int frequencia;
    struct Antena* proxAntena;
    struct ListaNefastas* listaNefasta;
}Antena;
```

Figura 3: Struct *"Antena"*

Antena então terá:

- Array de tamanho 2 para inteiros, sendo posição 0 linhas e 1 colunas.
- Variável inteira para receber uma frequência.
- Apontador do tipo Antena, que ficará "encadeado" com primAntena, ou seja é a próxima antena.
- Apontador do tipo listaNefasta, que será a lista onde iremos armazenar todas as localizações de uma determinada antena

3.4 Lista Nefasta

Listas Nefastas então terá:

- Apontador do tipo LocalizaçãoNefasta, que apontará para a primeira localização da lista.

```
typedef struct ListaNefastas
{
    struct LocalizaoNefasta* locNefasta;
}ListaNefastas;
```

Figura 4: Struct *"ListaNefasta"*

3.5 Localização Nefasta

```
typedef struct LocalizaoNefasta
{
    int posNefasta[2];
    struct LocalizaoNefasta* proxNefasta;
}LocalizaoNefasta;
```

Figura 5: Struct *"LocalizaçãoNefasta"*

LocalizaçãoNefasta então terá:

- Apontador do tipo LocalizaçãoNefasta, que apontará para a próxima localização da lista.
- Array de tamanho 2 para inteiros, sendo posição 0 linhas e 1 colunas.

4. Principais métodos

Para a definição de métodos para a manipulação da minha matriz, estes são aqules que considero como principais:

4.1 Método CriarMatriz

```
Matriz* CriarMatriz(int linhas, int colunas)
{
    Matriz* m;
    m = (Matriz*)malloc(sizeof(Matriz));
    m->tamMatriz[0] = linhas;
    m->tamMatriz[1] = colunas;
    m->listaAntenas = (ListaAntenas*)malloc(sizeof(ListaAntenas));
    m->listaAntenas->tamMatriz[0] = linhas;
    m->listaAntenas->tamMatriz[1] = colunas;
    m->listaAntenas->primAntena = NULL;
    return m;
}
```

Figura 6: Método *"CriarMatriz"*

Este é o método inicial onde crio a matriz através de inserção de valores que irão definir o tamMatriz, anteriormente referido.

Criamos assim um apontador, onde criamos também espaço para a minha lista de antenas e por uma boa prática, defino o primAntena a NULL.

4.2 Método CriarAntena

```
Antena* CriarAntena(Matriz* m,int linha, int coluna, double frequencia)
{
    Antena* auxAntena;
    auxAntena = AdicionarAntena(linha, coluna, frequencia);
    AdiconarAntenaMatriz(m,auxAntena);
    VerificarAntenaAnterior(m->listaAntenas->primAntena, auxAntena, m->listaAntenas->tamMatriz[0], m->listaAntenas->tamMatriz[1]);

    return auxAntena;
}
```

Figura 7: Método *"CriarAntena"*

Este método foi criado em Matriz.c para não haver, complicações no com os "include" utilizados nos ficheiros .h

Pois mesmo que utilizemos, pragam once, se contivermos include, por exemplo Antena em diferentes .h o programa irá entrar numa espécie de loop.

A partir deste método ocorre tudo, inserção da antena na matriz e cálculo de localizações nefastas.

4.3 Método AdicionarAntenaMatriz

```
void AdiconarAntenaMatriz(Matriz* m, Antena* antena)
{
    if (m->listaAntenas->primAntena != NULL)
    {
        Antena* auxAntena = m->listaAntenas->primAntena;
        while (auxAntena->proxAntena != NULL)
        {
            auxAntena = auxAntena->proxAntena;
        }
        auxAntena->proxAntena = antena;
        antena->proxAntena = NULL;
    }
    else
    {
        m->listaAntenas->primAntena = antena;
        antena->proxAntena = NULL;
    }
}
```

Figura 8: Método *"AdicionarAntenaMatriz"*

Neste método, depois de Antena ser criado associamos à nossa matriz. Para isso criamos um apontador auxiliar percorrera a lista de Antenas que se encontra na nossa matriz criada.

Caso a primeira antena seja NULL, será lá que irá inserir a mesma, se não, percorre a próxima antena até esta ser diferente de NULL e por fim adicionar.

4.4 Método VerificarAntenaAnterior

```
int VerificarAntenaAnterior(Antena* listaAntena, Antena* antenaAtual, int linhas, int colunas)
{
    Antena* auxAntenaAntes = listaAntena;

    while (auxAntenaAntes != antenaAtual)
    {
        int auxAntenaAnterior[2] = { auxAntenaAntes->posAntena[0], auxAntenaAntes->posAntena[1] };
        int auxAntenaAtual[2] = { antenaAtual->posAntena[0], antenaAtual->posAntena[1] };
        if (auxAntenaAntes->frequencia == antenaAtual->frequencia)
        {
            VerificarNefasta(auxAntenaAnterior, auxAntenaAtual, linhas, colunas);
            NefastaAntena(auxAntenaAntes, antenaAtual, auxAntenaAnterior, auxAntenaAtual);
        }
        auxAntenaAntes = auxAntenaAntes->proxAntena;
    }
    return 0;
}
```

Figura 9: Método *"VerificarAntenaAnterior"*

Aqui, como por em parâmetro entra o apontador presente na struct matriz conseguimos ter acesso completo à lista utilizando um ciclo while, tal como na figura anterior. Depois entra em um método secundário "VerificarNefasta" presente em listaNefastas que possui o cálculo descrito anteriormente. Depois disso temos também "Nefasta antena" que define as posições calculadas a -1, caso as localizações nefastas calculadas forem de iguais coordenadas a alguma antena. Ainda desse, entra por fim em:

4.5 Método InserirNefasta

```
void InserirNefasta(int* auxAntenaAnterior, Antena* auxAntenaAntes, int* auxAntenaAtual, Antena* AntenaAtual)
{
    if (auxAntenaAntes != NULL && auxAntenaAnterior[0] != -1 && auxAntenaAnterior[0] != 0 && auxAntenaAnterior[1] != 0)
    {
        LocalizaoNefasta* novaPosicao = (LocalizaoNefasta*)malloc(sizeof(LocalizaoNefasta));
        novaPosicao->posNefasta[0] = auxAntenaAnterior[0];
        novaPosicao->posNefasta[1] = auxAntenaAnterior[1];
        novaPosicao->proxNefasta = NULL;
        if (auxAntenaAntes->listaNefasta == NULL)
        {
            auxAntenaAntes->listaNefasta = (LocalizaoNefasta*)malloc(sizeof(LocalizaoNefasta));
            auxAntenaAntes->listaNefasta->locNefasta = novaPosicao;
        }
        else
        {
            LocalizaoNefasta* temp = auxAntenaAntes->listaNefasta->locNefasta;
            while (temp->proxNefasta != NULL)
            {
                temp = temp->proxNefasta;
            }
            temp->proxNefasta = novaPosicao;
        }
    }
```

Figura 10: Método *"InserirNefasta"*

Aqui, entra apenas a antenaAnterior, ou seja, a iterada no método anterior e entra também a antena atual, tal como apontador para o array de posições calculadas.

Funcionamento é parecido com Antena, logo, alocamos espaço para guardar a localização nefasta na lista e criamos um apontador que recebe os valores das posições.

Está dividida em dois grandes ciclos um que define as coordenadas para a respetica antena, neste caso da imagem, a antena anterior, mas o mesmo acontece para a antena atual.

5. Conclusão

No âmbito da disciplina de Estrutura de Dados Avançada, este trabalho prático permitiu desenvolver uma solução para gerir antenas e localizações nefastas numa matriz, utilizando estruturas de dados como listas ligadas e métodos eficientes para manipulação e cálculo de coordenadas.

O problema exigiu uma abordagem cuidadosa para garantir que as localizações nefastas fossem calculadas corretamente, respeitando as condições de frequência e distância entre antenas, bem como a verificação de limites da matriz e sobreposição com outras antenas.

A implementação das estruturas, como Matriz, ListaAntena, Antena, ListaNefasta e LocalizaçãoNefasta, foi essencial para organizar os dados e facilitar as operações de inserção, remoção e verificação. Os métodos desenvolvidos, como CriarMatriz, CriarAntena, AdicionarAntenaMatriz, VerificarAntenaAnterior e InserirNefasta, demonstraram-se eficazes na resolução do problema proposto, garantindo a correta atualização das localizações nefastas sempre que uma nova antena era adicionada.

As principais dificuldades enfrentadas relacionaram-se com o cálculo preciso das coordenadas das localizações nefastas e a garantia de que estas não ultrapassassem os limites da matriz ou coincidissem com outras antenas. No entanto, através de uma abordagem sistemática e da utilização de funções como abs para evitar valores negativos, foi possível superar estes desafios.