Uma imagem com texto, Tipo de letra, captura de ecrã, Azul elétrico

Descrição gerada automaticamente

**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

**Barcelos, abril de 2025**

****

**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, abril de 2025**

# Ficha de Identificação

|  |  |
| --- | --- |
| **Elaborado por** | Fábio Rafael Gomes Costa |
| **Contato** | a22997@alunos.ipca.pt |
|  |  |
| **Unidade Curricular** | Estruturas de Dados Avançadas |
| **Curso** | Licenciatura em Engenharia de Sistemas Informáticos |
| **Instituição** | Escola Superior de Tecnologia do  Instituto Politécnico do Cávado e do Ave |
| **Professor** | Doutor Luís Gonzaga Martins Ferreira |
| **Gabinete** |  |
| **Contato** | lufer@ipca.pt |
|  |  |
| **Data de início** | 14 de março de 2025 |
| **Data de conclusão** | 11 de abril de 2025 |

# Resumo

O presente trabalho insere-se no âmbito da Unidade Curricular de Estruturas de Dados Avançadas (EDA) e tem como objetivo principal a aplicação prática dos conhecimentos adquiridos, através do desenvolvimento de uma solução que permita representar e manipular dados referentes a antenas e suas localizações numa cidade, utilizando estruturas de dados dinâmicas implementadas em linguagem C.

Na Fase 1 do projeto, foi implementado um sistema baseado em listas ligadas simples, permitindo a leitura de ficheiros de texto ou binários contendo matrizes representativas da distribuição de antenas, a inserção e remoção dinâmica de antenas, e a identificação automática de localizações com efeitos nefastos resultantes da interferência entre antenas com a mesma frequência.

A solução implementada foi posteriormente revista após a defesa da fase, tendo sido integradas sugestões do docente, tais como a eliminação da utilização de matrizes em favor de listas dinâmicas, a padronização das estruturas de dados e a substituição de funções void por funções com retorno. Adicionalmente, foram incorporadas melhorias próprias, como a introdução de uma estrutura agregadora (AntNef), a criação de funções dedicadas à libertação de memória e a atribuição de identificadores únicos às antenas, o que facilitou o rastreio e gestão de interferências.

O projeto revelou-se uma oportunidade de consolidação de conhecimentos sobre listas ligadas, modularização de código, leitura e escrita em ficheiros, bem como a importância de uma abordagem sistemática na deteção e representação de padrões em dados espaciais. A documentação foi realizada com *Doxygen* e o desenvolvimento decorreu num ambiente controlado com recurso ao *Visual Studio* e *GitHub*.

# Índice

[Ficha de Identificação I](#_Toc195209114)

[Resumo II](#_Toc195209115)

[Índice III](#_Toc195209116)

[Índice de Fragmentos de Código V](#_Toc195209117)

[Índice de Figuras VI](#_Toc195209118)

[Lista de Siglas e Acrônimo VI](#_Toc195209119)

[1. Introdução 1](#_Toc195209120)

[1.1 Contextualização 1](#_Toc195209121)

[1.2 Motivação/Pretensões e Objetivos 1](#_Toc195209122)

[1.3 Metodologia de Trabalho 1](#_Toc195209123)

[1.4 Ferramentas e Tecnologias Utilizadas 2](#_Toc195209124)

[1.5 Estrutura do Documento 2](#_Toc195209125)

[2. Enquadramento Teórico e Prático 4](#_Toc195209126)

[2.1 Fundamentos Teóricos 4](#_Toc195209127)

[3. Problemas 5](#_Toc195209128)

[4. Resolução dos Problemas (Antes da Defesa) 9](#_Toc195209129)

[4.1 Resolução do Problema 1. 9](#_Toc195209130)

[4.2 Resolução do Problema 2. 10](#_Toc195209131)

[4.3 Resolução do Problema 3.a. 11](#_Toc195209132)

[4.4 Resolução do Problema 3.b. 12](#_Toc195209133)

[4.5 Resolução do Problema 3.c. 13](#_Toc195209134)

[4.6 Resolução do Problema 3.d. 15](#_Toc195209135)

[5. Resolução dos Problemas (Pós-Defesa) 16](#_Toc195209136)

[5.1 Sugestões e Pedidos de Alteração da Defesa 16](#_Toc195209137)

[5.2 Alterações não sugeridas 17](#_Toc195209138)

[5.3 Resolução dos problemas 18](#_Toc195209139)

[5.3.1 Resolução do Problema 1. 19](#_Toc195209140)

[5.3.2 Resolução do Problema 2. 20](#_Toc195209141)

[5.3.3 Resolução do Problema 3.a. 22](#_Toc195209142)

[5.3.4 Resolução do Problema 3.b. 23](#_Toc195209143)

[5.3.5 Resolução do Problema 3.c. 24](#_Toc195209144)

[5.3.6 Resolução do Problema 3.d. 27](#_Toc195209145)

[5.3.7 Outras Funções 29](#_Toc195209146)

[6. Repositório GitHub 30](#_Toc195209147)

[7. Considerações Finais 31](#_Toc195209148)

[8. Referencias bibliográfica 32](#_Toc195209149)

# Índice de Fragmentos de Código

[Código 1 - Função Inserir Antena 9](#_Toc195209150)

[Código 2 - Função LerLista 10](#_Toc195209151)

[Código 3 - Função atualizarMatriz 11](#_Toc195209152)

[Código 4 - Função removerAntena 12](#_Toc195209153)

[Código 5 - Função matrizNefastos 14](#_Toc195209154)

[Código 6 - Função inserirNefasto 14](#_Toc195209155)

[Código 7 - Função apresentarLista 15](#_Toc195209156)

[Código 8 - Função apresentarListaNef 15](#_Toc195209157)

[Código 9 - Função apresentarMatriz 15](#_Toc195209158)

[Código 10 - struct Ant 18](#_Toc195209159)

[Código 11 - struct Nef 18](#_Toc195209160)

[Código 12 - struct AntNef 18](#_Toc195209161)

[Código 13 - Função InserirAntena 19](#_Toc195209162)

[Código 14 - Função LerLista 21](#_Toc195209163)

[Código 15 - Função AdicionarAntena 22](#_Toc195209164)

[Código 16 - Função RemoverAntena 23](#_Toc195209165)

[Código 17 - Função EncontrarNefastos 25](#_Toc195209166)

[Código 18 - Função InserirNefasto 26](#_Toc195209167)

[Código 19 – ApresentarLista 27](#_Toc195209168)

[Código 20 - Função ApresentarMatrizLista 27](#_Toc195209169)

[Código 21 - Função ApresentarListaNef 28](#_Toc195209170)

[Código 22 - Função ApresentarMatrizListaNef 28](#_Toc195209171)

[Código 23 - Função FreeLista 29](#_Toc195209172)

[Código 24 - Função FreeListaNef 29](#_Toc195209173)

[Código 25 - Função FreeAntNef 29](#_Toc195209174)

# Índice de Figuras

[Figura 1 - Matriz Problema I 5](#_Toc195209175)

[Figura 2 - Matriz Problema II 6](#_Toc195209176)

[Figura 3 - Matriz Problema III 6](#_Toc195209177)

[Figura 4 - Matriz Problema IV 7](#_Toc195209178)

[Figura 5 - Matriz Problema V 8](#_Toc195209179)

# Lista de Siglas e Acrônimo

EDA – Estruturas de Dados Avançadas

ID – Identificador (Identificador Unico)

IPCA – Instituto Politécnico do Cavado e do Ave

UC – Unidade Curricular

VS – Visual Studio 2022

# Introdução

## 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.

## 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.

## Metodologia de Trabalho

Para a concretização deste projeto, foi adotada uma abordagem faseada, que permitiu uma evolução gradual e sustentada do desenvolvimento. Inicialmente, foi feita uma análise cuidada dos requisitos propostos, de forma a planear as funcionalidades a implementar e a definir a estrutura geral do código. Seguidamente, iniciou-se a fase de codificação, onde cada forma foi desenvolvida de forma modular, facilitando tanto a leitura como a manutenção do código. Ao longo do processo, foram realizados testes frequentes com o objetivo de garantir a correção das operações e a integridade dos dados. A documentação foi também integrada desde cedo, utilizando o *Doxygen* para manter a organização e clareza do código.

## Ferramentas e Tecnologias Utilizadas

No desenvolvimento do projeto, utilizei a linguagem de programação C, utilizada no contexto de sala de aula. O *Visual Studio* foi o ambiente escolhido pela sua interface intuitiva e funcionalidades de depuração. Para o controlo de versões, recorreu-se ao *GitHub*, garantindo o acompanhamento e preservação do código. A documentação foi gerada automaticamente com o *Doxygen*, a partir de comentários no código. Finalmente, o *Microsoft Word* foi utilizado para a redação do relatório.

## Estrutura do Documento

Este relatório foi organizado de forma a apresentar o trabalho desenvolvido de maneira clara e sequencial. Inicia-se com a capa, subcapa e Ficha de Identificação, seguida do Resumo e do Índice, que inclui tanto os tópicos principais como o Índice de Figuras, o Índice de Fragmentos de Código e a Lista de Siglas e Acrónimos.

A Introdução apresenta o enquadramento geral do projeto, incluindo a contextualização, os objetivos, a metodologia e as ferramentas utilizadas.

Seguindo a introdução, encontra-se o Enquadramento Teórico e Prático, onde são discutidos os fundamentos teóricos.

Na secção seguinte, Problemas, são identificados e descritos os problemas enfrentados durante o desenvolvimento do projeto.

O Desenvolvimento do projeto está dividido em duas fases: a resolução dos problemas antes da defesa, detalhada na Resolução dos Problemas (Antes da Defesa), e as modificações feitas após a defesa, descritas em Resolução dos Problemas (Pós-Defesa). Cada problema é analisado individualmente, com as respetivas soluções apresentadas nas subsecções de Resolução do Problema 1, Resolução do Problema 2, e assim por diante, até o Problema 3.d.

A seguir, encontra-se o Repositório GitHub, onde é apresentado o endereço do repositório do projeto.

O relatório é concluído com as Considerações Finais, onde são apresentadas as reflexões sobre o trabalho realizado, as aprendizagens obtidas e sugestões de melhoria.

Por fim, são apresentadas as Referências Bibliográficas, onde se encontram listados todos os links consultados durante o desenvolvimento do projeto.

# Enquadramento Teórico e Prático

Neste capítulo, abordamos os conceitos teóricos e as opções práticas que orientam o desenvolvimento do projeto. São exploradas as estruturas de dados utilizadas, a lógica de deteção de interferências entre antenas e as principais decisões de implementação em linguagem C.

## Fundamentos Teóricos

Este projeto insere-se no domínio das estruturas de dados e processamento de informação espacial. A sua base assenta na utilização de listas ligadas simples para representar duas entidades distintas: as antenas e as localizações nefastas (zonas onde ocorre interferência entre antenas).

Cada antena é definida por um carácter que representa a sua frequência, e pelas coordenadas bidimensionais (x, y) na grelha. As listas permitem armazenar estas antenas de forma dinâmica, possibilitando a inserção, remoção e atualização eficiente de dados sem necessidade de redimensionar estruturas de memória.

Em termos de abordagem algorítmica, o projeto implementa:

* Leitura e conversão de ficheiros (.txt ou binários) para uma estrutura de lista ligada de antenas.
* Cálculo de posições nefastas baseado em simetria entre pares de antenas com igual frequência.
* Representação visual das antenas e das interferências num modelo matricial.
* Operações de atualização dinâmica (inserção, remoção).

Além disso, são aplicados princípios fundamentais da programação em C, como modularização (separação entre ficheiros .c e .h), encapsulamento de dados e gestão explícita da memória. A documentação das funções segue o padrão *Doxygen*, facilitando a manutenção e o entendimento do sistema.

# Problemas

O presente capítulo serve para apresentar ao leitor os problemas utilizados como base para a resolução do projeto.

**“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:*

*Uma imagem com captura de ecrã, quadrado, Retângulo, tipografia

Os conteúdos gerados por IA poderão estar incorretos.*

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 #:*

*Uma imagem com captura de ecrã, Retângulo, padrão, quadrado

Os conteúdos gerados por IA poderão estar incorretos.*

Figura 2 - Matriz Problema II

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

*Uma imagem com captura de ecrã, padrão, Retângulo, quadrado

Os conteúdos gerados por IA poderão estar incorretos.*

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:*

*Uma imagem com captura de ecrã, texto, quadrado, padrão

Os conteúdos gerados por IA poderão estar incorretos.*

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:*

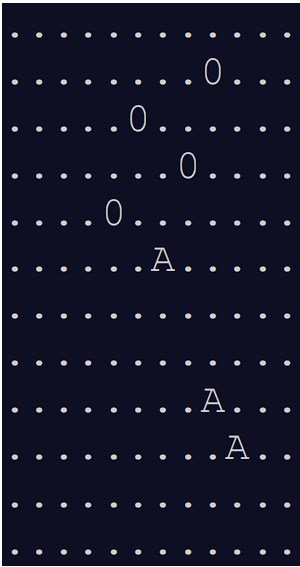
**

Figura 5 - Matriz Problema V

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

# Resolução dos Problemas (Antes da Defesa)

Neste capítulo são apresentadas as soluções desenvolvidas para os problemas propostos no enunciado, com vista à preparação para a defesa do projeto. Cada secção descreve a resolução de um problema específico, através da explicação das funções implementadas, que permitem manipular a matriz, gerir a lista de antenas e identificar os efeitos nefastos.

## Resolução do Problema 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;

}

Código 1 - Função Inserir Antena

## 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, 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;

}

Código 2 - Função LerLista

## 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;

}

Código 3 - Função atualizarMatriz

## 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;

}

}

Código 4 - Função removerAntena

## 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;

}

Código 5 - Função matrizNefastos

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;

}

Código 6 - Função inserirNefasto

## 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;

}

}

Código 7 - Função apresentarLista

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;

}

}

Código 8 - Função apresentarListaNef

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.

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");

}

}

Código 9 - Função apresentarMatriz

# Resolução dos Problemas (Pós-Defesa)

No presente capítulo será apresentada a resolução do projeto posterior a defesa do mesmo, aplicando as recomendações, sugestões e pedidos de mudanças sugeridas pelo docente.

## Sugestões e Pedidos de Alteração da Defesa

Após a apresentação do projeto, foram apontadas diversas recomendações por parte do docente, com o objetivo de melhorar a estrutura, legibilidade e eficiência da implementação. Estas sugestões foram devidamente analisadas e incorporadas na versão final do projeto. As principais alterações solicitadas foram as seguintes:

* **Eliminação da utilização de matrizes**: Foi recomendado que toda a manipulação de dados fosse efetuada exclusivamente através de listas ligadas, promovendo uma abordagem dinâmica e escalável à gestão da informação.
* **Revisão e correção da resolução do problema 3.c**: A implementação original destas funcionalidades foi revista para garantir o correto funcionamento da deteção automática das posições nefastas.
* **Padronização das estruturas de dados**: As estruturas ANT e NEF foram renomeadas para Ant e Nef, respetivamente, em conformidade com as convenções de nomenclatura em C, promovendo maior clareza e consistência.
* **Evitar o uso de funções *void***: Todas as funções passaram a devolver valores explícitos, seja através de ponteiros para estruturas atualizadas ou códigos de retorno, assegurando maior controlo sobre o fluxo de execução.
* **Suporte à leitura de ficheiros de texto e binários**: Foi acrescentada a capacidade de ler tanto ficheiros .txt como ficheiros binários, permitindo maior versatilidade na entrada de dados.
* **Remoção de duplos apontadores**: Foi eliminada a necessidade de utilização de duplos apontadores, substituindo-se por estratégias mais simples e robustas que mantêm a clareza do código.
* **Unificação do estilo de programação**: Foi adotado um estilo consistente para a nomeação de variáveis, funções e estruturas, contribuindo para a legibilidade e manutenibilidade do código.

## Alterações não sugeridas

Além das recomendações feitas durante a defesa, foram ainda realizadas diversas melhorias adicionais por iniciativa própria, com o intuito de robustecer e otimizar o projeto. As principais alterações não sugeridas, mas implementadas, foram:

* **Libertação de memória a partir de funções dedicadas:** Na versão inicial do projeto, a libertação de memória era efetuada diretamente no *main*, o que resultava numa maior dispersão da lógica e dificuldade na manutenção do código. Com a reformulação, foram criadas funções específicas responsáveis por tratar, de forma modular e segura, a desalocação das estruturas de dados dinâmicas.
* **Introdução da estrutura AntNef**: Foi criada uma estrutura auxiliar responsável por armazenar os ponteiros para as listas de antenas e de posições nefastas, facilitando o encapsulamento e manuseamento conjunto destas entidades.
* **Atribuição de um identificador único a cada antena**: A cada antena foi atribuído um campo id, assegurando a identificação inequívoca das antenas, o que se revelou particularmente útil na gestão das posições nefastas, onde se associam pares de antenas com base nos seus identificadores.

## Resolução dos problemas

Nesta secção é apresentada a implementação das funcionalidades solicitadas, distribuídas em diferentes componentes modulares do projeto. A arquitetura adotada divide o código-fonte em três ficheiros principais: *main.c, funcao.h e funcao.c.* O ficheiro *funcao.h* contém a definição das estruturas de dados e os protótipos das funções. O ficheiro *funcao.c* implementa a lógica de todas as funcionalidades relacionadas com antenas e posições nefastas. O *main.c* serve como programa de teste e demonstração das funcionalidades desenvolvidas.

As estruturas de dados centrais encontram-se declaradas no ficheiro *funcao.h,* e são:

* Ant: Representa uma antena, contendo a sua frequência (carácter), coordenadas (x, y) e um identificador único (ID).
* Nef: Representa uma posição nefasta na matriz, armazenando as suas coordenadas e os IDs das antenas que originam essa interferência.
* AntNef: Estrutura agregadora que contém os ponteiros para as listas de antenas e de posições nefastas.

typedef struct Ant {

char freqAntena; /\*\*< Frequência da antena (carácter representativo).\*/

int id; /\*\*< Identificador único da antena.\*/

int x; /\*\*< Coordenada horizontal (coluna) da antena.\*/

int y; /\*\*< Coordenada vertical (linha) da antena.\*/

struct Ant\* proxAntena; /\*\*< Ponteiro para a próxima antena na lista ligada.\*/

} Ant;

Código 10 - struct Ant

typedef struct Nef {

int x; /\*\*< Coordenada horizontal da posição nefasta. \*/

int y; /\*\*< Coordenada vertical da posição nefasta. \*/

int antena1; /\*\*< ID da primeira antena associada ao nefasto. \*/

int antena2; /\*\*< ID da segunda antena associada ao nefasto. \*/

struct Nef\* proxNef; /\*\*< Ponteiro para o próximo nefasto na lista ligada. \*/

} Nef;

Código 11 - struct Nef

typedef struct AntNef {

struct Ant\* lista; /\*\*< Ponteiro para a lista de antenas. \*/

struct Nef\* listaNef; /\*\*< Ponteiro para a lista de posições nefastas. \*/

} AntNef;

Código 12 - struct AntNef

### Resolução do Problema 1.

A função InserirAntena permite criar uma antena e inseri-la na lista ligada. Esta função aloca dinamicamente memória para a nova estrutura, atribui os valores recebidos e liga-a ao início da lista existente. O *ID* é gerado externamente e passado como argumento.

Ant\* InserirAntena(Ant\* lista, char freq, int x, int y, int id) {

Ant\* novaAntena = (Ant\*)malloc(sizeof(Ant));

if (novaAntena == NULL) {

perror("Error allocating memory");

return;

}

novaAntena->freqAntena = freq;

novaAntena->x = x;

novaAntena->y = y;

novaAntena->id = id; // Atribui o ID à nova antena

novaAntena->proxAntena = lista;

lista = novaAntena;

return lista;

}

Código 13 - Função InserirAntena

### Resolução do Problema 2.

A função LerLista lê um ficheiro de texto com uma matriz de caracteres representando as antenas. Cada carácter diferente de ponto (.) é considerado uma antena, sendo criada uma estrutura correspondente. A função deteta dinamicamente o número de linhas e colunas do ficheiro. Após carregar as antenas, são automaticamente calculadas as posições nefastas através da função EncontrarNefastos, e é construída a estrutura AntNef.

Ant\* LerLista(const char\* nomeFicheiro, const char\* tipoFicheiro, int\* linhas, int\* colunas) {

FILE\* ficheiro;

char nomeCompleto[256]; // Buffer para o nome do arquivo

snprintf(nomeCompleto, sizeof(nomeCompleto), "%s%s", nomeFicheiro, tipoFicheiro);

if (strcmp(tipoFicheiro, ".txt") == 0) {

errno\_t err = fopen\_s(&ficheiro, nomeCompleto, "r");

if (err != 0) {

perror("Erro ao abrir o ficheiro");

return NULL;

}

}

else {

errno\_t err = fopen\_s(&ficheiro, nomeCompleto, "rb");

if (err != 0) {

perror("Erro ao abrir o ficheiro");

return NULL;

}

}

Ant\* lista = NULL;

char linha[MAX\_COLUNAS];//100

int y = 0;

int maxColunas = 0;

int id = 0; // Inicializa o ID da antena

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++) {

if (linha[x] != '.') {

lista = InserirAntena(lista, linha[x], x, y, id);

id++; // Incrementa o ID da antena

}

}

y++;

}

\*linhas = y;

\*colunas = maxColunas;

Nef\* listaNef = EncontrarNefastos(lista, y, maxColunas);

AntNef\* listaAntNef = (AntNef\*)malloc(sizeof(AntNef));

if (listaAntNef == NULL) {

perror("Erro ao alocar memória para AntNef");

FreeLista(lista);

FreeListaNef(listaNef);

return 1;

}

listaAntNef->lista = lista;

listaAntNef->listaNef = listaNef;

fclose(ficheiro);

return listaAntNef;

}

Código 14 - Função LerLista

### Resolução do Problema 3.a.

A função AdicionarAntena permite adicionar uma nova antena ou atualizar a frequência de uma antena já existente na mesma posição. Em qualquer dos casos, as posições nefastas são recalculadas de forma automática após a alteração.

AntNef\* AdicionarAntena(AntNef\* listaAntNef, char freq, int x, int y, int linhas, int colunas) {

Ant\* atual = listaAntNef->lista;

int id = atual->id + 1;

int result;

while (atual != NULL) {

if (atual->x == x && atual->y == y) {

atual->freqAntena = freq;

result = FreeListaNef(listaAntNef->listaNef);

listaAntNef->listaNef = EncontrarNefastos(listaAntNef->lista, linhas, colunas);

return listaAntNef;

}

atual = atual->proxAntena;

}

listaAntNef->lista = InserirAntena(listaAntNef->lista, freq, x, y, id);

result = FreeListaNef(listaAntNef->listaNef);

listaAntNef->listaNef = EncontrarNefastos(listaAntNef->lista, linhas, colunas);

return listaAntNef;

}

Código 15 - Função AdicionarAntena

### Resolução do Problema 3.b.

A função RemoverAntena percorre a lista de antenas e remove a antena cujas coordenadas coincidam com as especificadas. Após a remoção, são igualmente eliminadas todas as posições nefastas associadas à antena removida, garantindo a integridade da estrutura.

AntNef\* RemoverAntena(AntNef\* listaAntNef, int x, int y) {

Ant\* atual = listaAntNef->lista;

Ant\* anterior = NULL;

int id = 0;

while (atual != NULL) {

if (atual->x == x && atual->y == y) {

if (anterior == NULL) {

id = atual->id;

listaAntNef->lista = atual->proxAntena;

}

else {

id = atual->id;

anterior->proxAntena = atual->proxAntena;

}

Nef\* atualNef = listaAntNef->listaNef;

Nef\* anteriorNef = NULL;

while (atualNef != NULL) {

if (atualNef->antena1 == id || atualNef->antena2 == id) {

if (anteriorNef == NULL) {

listaAntNef->listaNef = atualNef->proxNef;

}

else {

anteriorNef->proxNef = atualNef->proxNef;

}

}

anteriorNef = atualNef;

atualNef = atualNef->proxNef;

}

free(atualNef);

free(atual);

return listaAntNef;

}

anterior = atual;

atual = atual->proxAntena;

}

}

Código 16 - Função RemoverAntena

### Resolução do Problema 3.c.

A função EncontrarNefastos verifica todos os pares de antenas com a mesma frequência. Se estas estiverem alinhadas em uma posição tipo L (x da antena 1 é diferente do x da antena 2, e o mesmo para os y’s), são identificadas duas posições nefastas — simétricas em relação às antenas. As posições são adicionadas à lista ligada através da função InserirNefasto, a qual verifica previamente se o local já está ocupado por uma antena ou se já existe na lista.

Nef\* EncontrarNefastos(Ant\* lista, int linhas, int colunas) {

int menorx, menory, maiorx, maiory, difx, dify, idant1, idant2;

Nef\* listaNef = NULL;

Ant\* listaAnt1 = lista;

while (listaAnt1 != NULL) {

Ant\* listaAnt2 = lista;

while (listaAnt2 != NULL) {

if (listaAnt1->freqAntena == listaAnt2->freqAntena && (listaAnt1->y != listaAnt2->y || listaAnt1->x != listaAnt2->x)) {

if (listaAnt1->x > listaAnt2->x)

{

difx = listaAnt1->x - listaAnt2->x;

menorx = listaAnt2->x - difx;

maiorx = listaAnt1->x + difx;

}

else

{

difx = listaAnt2->x - listaAnt1->x;

menorx = listaAnt1->x - difx;

maiorx = listaAnt2->x + difx;

}

if (listaAnt1->y > listaAnt2->y)

{

dify = listaAnt1->y - listaAnt2->y;

menory = listaAnt2->y - dify;

maiory = listaAnt1->y + dify;

}

else

{

dify = listaAnt2->y - listaAnt1->y;

menory = listaAnt1->y - dify;

maiory = listaAnt2->y + dify;

}

idant1 = listaAnt1->id;

idant2 = listaAnt2->id;

if (listaAnt1->x > listaAnt2->x && listaAnt1->y > listaAnt2->y || listaAnt1->x < listaAnt2->x && listaAnt1->y < listaAnt2->y)

{

if (menorx >= 0 && menory >= 0 && menorx<=linhas && menory<=colunas)

{

listaNef = InserirNefasto(listaNef, menorx, menory, idant1, idant2, lista);

}

if (maiorx <= colunas && maiory <= linhas && maiorx <= linhas && maiory <= colunas)

{

listaNef = InserirNefasto(listaNef, maiorx, maiory, idant1, idant2, lista);

}

}

else if (listaAnt1->x > listaAnt2->x && listaAnt1->y < listaAnt2->y || listaAnt1->x < listaAnt2->x && listaAnt1->y > listaAnt2->y)

{

if (maiorx >= 0 && menory >= 0 && maiorx <= linhas && menory <= colunas)

{

listaNef = InserirNefasto(listaNef, maiorx, menory, idant1, idant2, lista);

}

if (menorx <= colunas && maiory <= linhas && menorx <= linhas && maiory <= colunas)

{

listaNef = InserirNefasto(listaNef, menorx, maiory, idant1, idant2, lista);

}

}

}

listaAnt2 = listaAnt2->proxAntena;

}

listaAnt1 = listaAnt1->proxAntena;

}

return listaNef;

}

Código 17 - Função EncontrarNefastos

Nef\* InserirNefasto(Nef\* listaNef, int x, int y, int ant1, int ant2, Ant\* lista) {

Ant\* atual = lista;

while (atual != NULL) {

if (atual->x == x && atual->y == y) {

return listaNef;

}

atual = atual->proxAntena;

}

Nef\* temp = listaNef;

while (temp != NULL)

{

if (temp->x == x && temp->y == y && temp->antena1 == ant2 && temp->antena2 == ant1)

{

return listaNef;

}

temp = temp->proxNef;

}

Nef\* novoNefasto = (Nef\*)malloc(sizeof(Nef));

if (novoNefasto == NULL) {

perror("Error allocating memory");

return;

}

novoNefasto->x = x;

novoNefasto->y = y;

novoNefasto->antena1 = ant1;

novoNefasto->antena2 = ant2;

novoNefasto->proxNef = listaNef;

listaNef = novoNefasto;

return listaNef;

}

Código 18 - Função InserirNefasto

### Resolução do Problema 3.d.

Para visualização dos dados na consola, foram implementadas as seguintes funções:

* ApresentarLista: Lista todas as antenas com as suas coordenadas, frequência e ID.
* ApresentarMatrizLista: Apresenta uma representação gráfica das antenas numa matriz, utilizando o carácter da frequência nas respetivas posições e pontos (.) nos restantes locais.
* ApresentarListaNef: Lista todas as posições nefastas com as coordenadas e os IDs das antenas associadas.
* ApresentarMatrizListaNef: Combina a informação das antenas e das posições nefastas numa única matriz. As antenas são representadas pelos seus caracteres, as posições nefastas por “#”, e os espaços vazios por “.”.

int ApresentarLista(Ant\* lista) {

Ant\* atual = lista;

while (atual != NULL) {

printf("Antena: %c, Coordenadas: (%d, %d) | Numero:%d\n", atual->freqAntena, atual->x, atual->y, atual->id);

atual = atual->proxAntena;

}

return 1;

}

Código 19 – ApresentarLista

int ApresentarMatrizLista(Ant\* lista, int linhas, int colunas) {

for (int y = 0; y < linhas; y++) {

for (int x = 0; x < colunas; x++) {

int encontrou = 0;

Ant\* atual = lista;

while (atual != NULL) {

if (atual->x == x && atual->y == y) {

printf("%c ", atual->freqAntena);

encontrou = 1;

break;

}

atual = atual->proxAntena;

}

if (encontrou == 0) printf(". ");

}

printf("\n");

}

return 1;

}

Código 20 - Função ApresentarMatrizLista

int ApresentarListaNef(Nef\* listaNef) {

Nef\* atual = listaNef;

while (atual != NULL) {

printf("Nefasto: (%d, %d) - Antenas: %d e %d\n", atual->x, atual->y, atual->antena1, atual->antena2);

atual = atual->proxNef;

}

return 1;

}

Código 21 - Função ApresentarListaNef

int ApresentarMatrizListaNef(Ant\* lista, int linhas, int colunas, Nef\* listaNef) {

for (int y = 0; y < linhas; y++) {

for (int x = 0; x < colunas; x++) {

int encontrou = 0;

Ant\* atual = lista;

while (atual != NULL) {

if (atual->x == x && atual->y == y) {

printf("%c ", atual->freqAntena);

encontrou = 1;

break;

}

atual = atual->proxAntena;

}

if (encontrou == 0) {

Nef\* atualNef = listaNef;

while (atualNef != NULL) {

if (atualNef->x == x && atualNef->y == y) {

printf("# ");

encontrou = 1;

break;

}

atualNef = atualNef->proxNef;

}

}

if (encontrou == 0) printf(". ");

}

printf("\n");

}

return 1;

}

Código 22 - Função ApresentarMatrizListaNef

### Outras Funções

Para assegurar uma gestão eficiente da memória dinâmica alocada durante a execução do programa, foram desenvolvidas funções auxiliares para libertação das estruturas de dados utilizadas. Estas funções garantem que não existam fugas de memória ao encerrar a aplicação ou após operações de atualização das listas.

* FreeLista: Liberta a memória ocupada pela lista ligada de antenas. A função percorre todos os elementos da lista, desalocando cada nó individualmente através da função free.
* FreeListaNef: Liberta a memória alocada para a lista de posições nefastas. O funcionamento é análogo à função anterior, iterando pelos elementos da lista ligada e libertando cada posição nefasta.
* FreeAntNef: Esta função permite libertar a estrutura agregadora AntNef, chamando internamente as funções de limpeza das listas de antenas e de posições nefastas, e por fim libertando a própria estrutura.

int FreeLista(Ant\* lista) {

Ant\* atual = lista;

while (atual != NULL) {

Ant\* temp = atual;

atual = atual->proxAntena;

free(temp);

}return 1;

}

Código 23 - Função FreeLista

int FreeListaNef(Nef\* listaNef) {

Nef\* atual = listaNef;

while (atual != NULL) {

Nef\* temp = atual;

atual = atual->proxNef;

free(temp);

}return 1;

}

Código 24 - Função FreeListaNef

int FreeAntNef(AntNef\* listaAntNef) {

FreeLista(listaAntNef->lista);

FreeListaNef(listaAntNef->listaNef);

free(listaAntNef);

return 1;

}

Código 25 - Função FreeAntNef

# 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.

# 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.

# 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