

Licenciatura em Engenharia de Sistemas Informáticos

Estruturas de Dados Avançadas

Relatório Projeto (Fase 2)

Raúl Ribeiro, aluno n.º 22552

18/05/2025

Conteúdo

[1. Introdução 3](#_Toc195632505)

[1.1. Motivação 3](#_Toc195632506)

[1.2. Enquadramento 3](#_Toc195632507)

[1.3. Objetivos 4](#_Toc195632508)

[1.4. Metodologia de Trabalho 4](#_Toc195632509)

[2. Trabalho Desenvolvido 5](#_Toc195632510)

[2.1. Requisitos 5](#_Toc195632511)

[2.1.1. Requisitos Funcionais 5](#_Toc195632512)

[2.1.2. Requisitos Não Funcionais 5](#_Toc195632513)

[2.2. Estrutura do projeto 5](#_Toc195632514)

[2.3. Implementação 6](#_Toc195632515)

[2.3.1. Estruturas de Dados 6](#_Toc195632516)

[2.3.2. Implementação de funções 7](#_Toc195632517)

[2.3.3. Impressão de grafos 11](#_Toc195632518)

[2.4. Organização do código 12](#_Toc195632519)

[3. Resultados 13](#_Toc195632520)

[4. Conclusão 13](#_Toc195632521)

[5. Glossário 14](#_Toc195632522)

[6. Referências 15](#_Toc195632523)

# Introdução

## Motivação

A segunda fase deste projeto, no âmbito da unidade curricular de Estruturas de Dados Avançadas (EDA), tem como principal objetivo aprofundar os conhecimentos adquiridos, explorando conceitos avançados de teoria dos grafos e a sua aplicação prática na linguagem de programação C. Esta abordagem permite uma compreensão mais abrangente dos conceitos teóricos e promove o desenvolvimento de competências fundamentais na modelação e manipulação de estruturas de dados complexas.

Nesta fase, o foco do projeto recai na representação da rede de antenas através de um grafo, em que os vértices representam antenas e as arestas interligam antenas com frequências iguais. Esta modelação permite aplicar algoritmos clássicos de grafos, como a pesquisa em profundidade (DFS) e em largura (BFS), bem como identificar caminhos entre antenas.

Para além de reforçar o pensamento algorítmico e a capacidade de resolução de problemas, o projeto mantém a ênfase na organização, modularização e documentação do código com Doxygen, garantindo a clareza e a eficiência da solução desenvolvida.

## Enquadramento

O presente relatório foi desenvolvido no âmbito da unidade curricular de Estruturas de Dados Avançadas (EDA), lecionada no segundo semestre do 1.º ano da Licenciatura em Engenharia de Sistemas Informáticos da Escola Superior de Tecnologia do Instituto Politécnico do Cávado e do Ave.

Esta segunda fase do projeto tem como finalidade aplicar conceitos avançados de estruturas de dados, com foco especial na teoria dos grafos, utilizando a linguagem de programação C. O trabalho insere-se numa abordagem prática e progressiva de consolidação de conteúdos, permitindo ao aluno transitar de uma solução baseada em listas ligadas para uma solução mais complexa e eficiente baseada em grafos.

## Objetivos

Os principais objetivos desta segunda fase do projeto incluem:

* Aplicar os conhecimentos adquiridos sobre grafos na resolução de um problema prático.
* Desenvolver uma solução modular em linguagem C, utilizando estruturas de grafos.
* Implementar algoritmos de pesquisa em profundidade (DFS) e em largura (BFS) para navegação no grafo.
* Identificar todos os caminhos possíveis entre antenas, consequentemente encontrando o mais curto entre eles.
* Promover a organização, documentação e clareza do código, utilizando a ferramenta Doxygen.
* Testar a solução com diferentes cenários, avaliando a sua eficiência e correção.

## Metodologia de Trabalho

A metodologia seguida para o desenvolvimento desta fase do projeto baseou-se numa abordagem prática e iterativa, permitindo aplicar de forma estruturada os conceitos adquiridos ao longo da unidade curricular.

Inicialmente, foi realizada uma análise detalhada do problema, de forma a compreender a transição do modelo de listas ligadas para grafos. Esta análise permitiu identificar os vértices (antenas) e as arestas (ligações entre antenas com frequências iguais).

A implementação concentrou-se no desenvolvimento de funcionalidades essenciais, incluindo:

* Carregamento dos dados a partir de ficheiro, criando o grafo correspondente;
* Implementação de algoritmos de pesquisa em profundidade (DFS) e em largura (BFS);
* Identificação de caminhos possíveis entre antenas com a mesma frequência;
* Apresentação dos resultados na consola de forma clara e organizada.  
  Tal como na primeira fase, a modularização do código e a sua documentação com Doxygen foram uma prioridade, garantindo a manutenibilidade e clareza da solução.

# Trabalho Desenvolvido

## Requisitos

### Requisitos Funcionais

* Permitir o carregamento de uma matriz com antenas a partir de um ficheiro de texto.
* Representar cada antena com a sua frequência (carácter) e localização (coordenadas).
* Armazenar as antenas numa estrutura dinâmica (lista ligada).
* Identificar e adicionar as adjacências de cada antena.
* Realizar a pesquisa por profundidade e por largura a partir de uma antena inicial, mostrando todas as antenas que podem ser alcançadas a partir dela.
* Identificar e mostrar todos os caminhos possíveis entre duas antenas (realizando a pesquisa por profundidade), mostrando também a distância percorrida em cada um deles.

### Requisitos Não Funcionais

* O programa deve ser modular, com separação clara entre leitura de dados, lógica de processamento e apresentação.
* O código deve ser documentado com *Doxygen*.
* Deve permitir o manuseamento de matrizes de qualquer dimensão.

## Estrutura do projeto

O projeto foi desenvolvido de forma modular, com a separação de responsabilidades em diferentes ficheiros:

* *Source.c*: Contém a função principal (*main*), responsável por orquestrar a execução do programa. Inclui a leitura dos dados das antenas, a criação das adjacências, a execução dos dois tipos de pesquisa (DFS e BFS) e a apresentação dos caminhos entre duas antenas.
* *Data.h*: Declara as estruturas de dados utilizadas para representar as antenas, grafos e todas as outras estruturas necessárias para a realização de pesquisas.
* *Functions.h*: Declara as funções auxiliares que implementam a lógica do programa, como a leitura de ficheiros, gestão de grafos e operações de pesquisa.

## Implementação

### Estruturas de Dados

As antenas foram representadas utilizando uma lista ligada dinâmica. Cada nó da lista contém um código, uma frequência, uma flag a informar se foi visitada (utilizada para as pesquisas dentro do grafo), as coordenadas, a próxima antena e as adjacências. A estrutura foi definida no ficheiro Data.h como podemos observar na *Figura 1 – Estrutura GR* (abaixo).

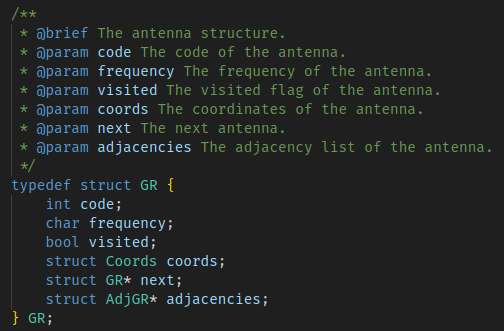


Figura 1 – Estrutura GR

Este ficheiro contém ainda todas as definições dos dados necessários para o bom funcionamento da aplicação:

* *struct Coords:* Estrutura das coordenadas de uma antena
* *struct GR:* Estrutura de uma antena (o nosso grafo)
* *struct AdjGR:* Estrutura das adjacências de uma antena
* *struct GRFile:* Semelhante à struct GR, mas para gravação e leitura de um ficheiro .bin
* *struct AdjGRFile:* Semelhante à struct AdjGR, mas para gravação e leitura de um ficheiro .bin
* *struct QueueNode:* Estrutura de um nó de uma fila utilizado para a pesquisa BFS
* *struct Queue:* Estrutura de uma fila de espera utilizada para a pesquisa BFS

### Implementação de funções

Todas as funções para o funcionamento da aplicação foram implementadas no ficheiro Functions.h. Abaixo está disponível um resumo de cada uma.

#### 2.3.2.1. Gestão de Antenas

A função *CreateAntenna* cria uma antena e retorna um apontador para ela, como é visível na *Figura 2 – Função para criar uma antena* (abaixo).

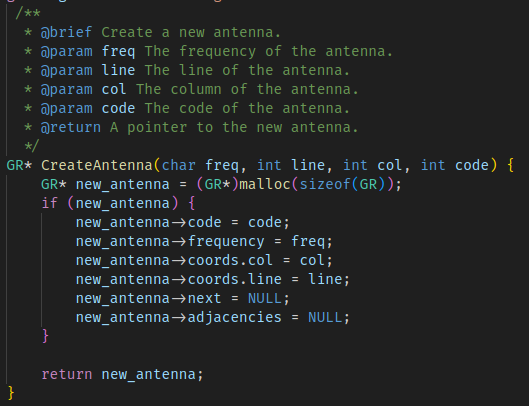


Figura 2 - Função para criar uma antena

Dentro da gestão de antenas existem ainda as seguinte funções:

* *AddAntenna*: Adiciona uma antena de forma organizada ao grafo (com base na sua frequência.
* *AddAntennaToEnd*: Adiciona uma antena no final do grafo, descartando qualquer organização.
* *SearchAntenna*: Procura e devolve uma antena existente no grafo com base no seu código.
* *AddAllAdjacencies*: Deteta e insere todas as adjacências de uma antena com base na sua frequência.

#### 2.3.2.2. Gestão de Grafos

A função *AddAntennaAdjacency* adiciona uma adjacência de uma antena à lista de adjacências da mesma, como é visível na *Figura 3 – Função para adicionar adjacência* (abaixo).

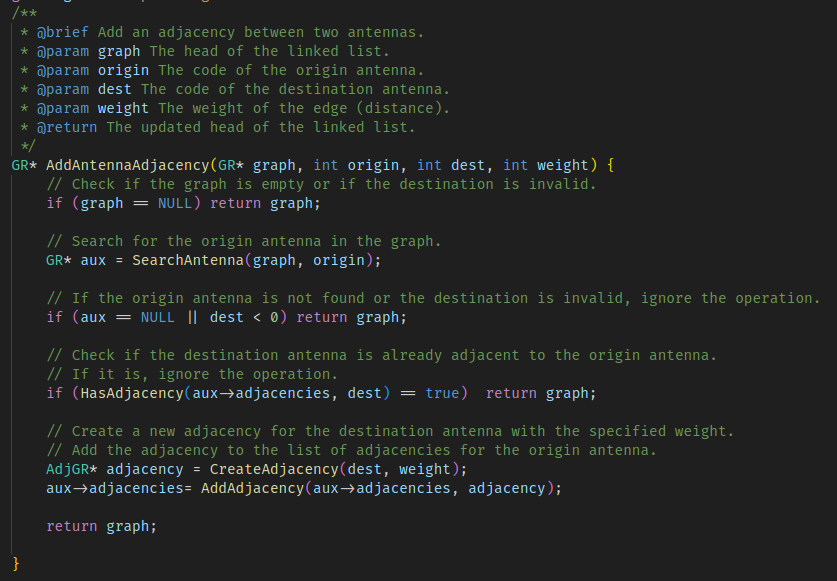


Figura 3 - Função para adicionar adjacência

Dentro da gestão de grafos existem ainda as seguinte funções:

* *HasAdjacency*: Verifica se uma adjacência já existe na lista de maneira a não a adicionar mais que uma vez.
* *CreateAdjacency*: Cria e devolve uma adjacência nova.
* *AddAdjacency*: Adiciona uma adjacência à lista de adjacências.
* *ShowAdjancencies*: Imprime as adjacências de uma antena na consola.
* *ShowGraph*: Imprime as informações das antenas existentes no grafo, incluindo as suas adjacências.
* *ResetVisitedAntennas*: Função simples para reiniciar a flag que indica se uma antena já foi visitada ou não.

#### 2.3.2.3. Operações de Ficheiros

A função *GetAntennasFromTextFile* carrega as antenas iniciais de um ficheiro de texto. Neste momento ainda não existe informação de adjacências, por isso ainda têm de ser criadas posteriormente, como é visível na *Figura 4 – Função para carregar* *antenas de ficheiro de texto* (abaixo).

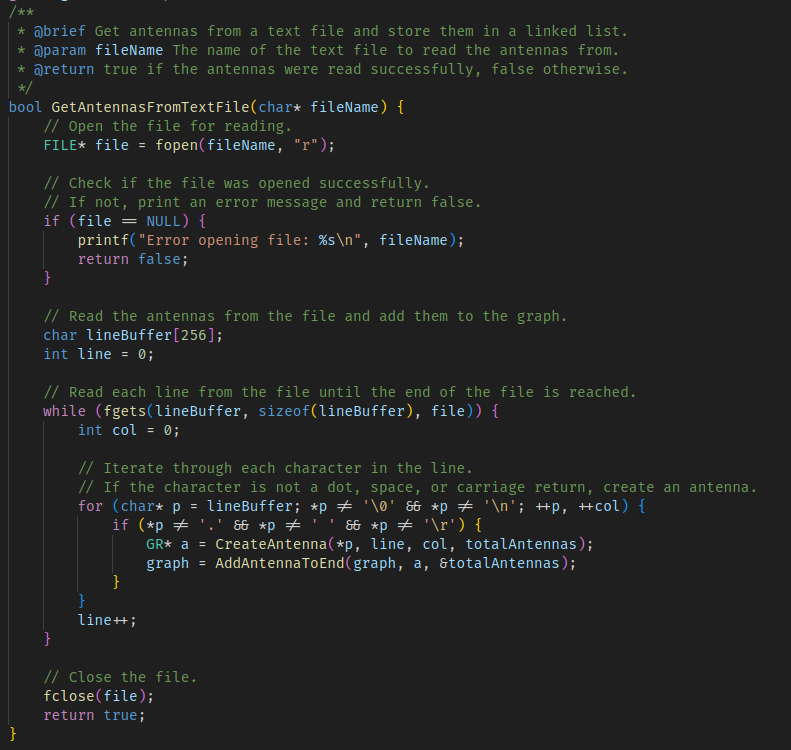


Figura 4 – Função para carregar antenas de ficheiro de texto

Dentro das operações de ficheiro existem ainda as seguinte funções:

* *GetGraphFromBinaryFile*: Carrega o grafo a partir de um ficheiro binário. Como este ficheiro é guardado depois do processamento inicial das antenas, o mesmo já contém as adjacências de cada antena.
* *SaveGraphToBinaryFile*: Após o processamento do grafo pela aplicação guarda os dados do grafo (incluindo as adjacências) dentro de um ficheiro binário.

#### 2.3.2.4. Operações de Pesquisa

Esta secção inclui todas as funções referentes à pesquisa dentro dos grafos, seja em largura (BFS) como em profundidade (DFS).

##### 2.3.2.4.1. Breadth First Search (BFS)

* *enqueue*: Adiciona um novo nó à fila de espera da pesquisa.
* *dequeue*: Remove um nó da fila de espera da pesquisa.
* *isEmpty*: Verifica se a fila de espera está vazia.
* *BreadthFirstSearch*: Executa a pesquisa por largura dentro do grafo a partir de uma antena inicial (BFS), imprimindo todas as antenas por onde passa.

##### 2.3.2.4.2. Depth First Search (DFS)

* *DepthFirstSearch*: Reinicia a flag das antenas visitadas e chama a função para executar a pesquisa em profundidade.
* *DepthFirstSearchUtil*: Executa a pesquisa em profundidade recursivamente, imprimindo pelo caminho todas as antenas por onde passa.
* *DepthFirstSearchPaths*: Semelhante ao DepthFirstSearch, mas inicia o array de caracteres responsável por carregar os caminhos disponíveis de uma antena a outra.
* *DepthFirstSearchPathUtil*: Semelhante ao DepthFirstSearchUtil, mas vez de mostrar as antenas por onde passa, vai armazenando todos os caminhos disponíveis de uma antena de origem a uma de destino, mostrando depois para o utilizador todos os caminhos possíveis entre as duas antenas, bem como a distância percorrida em cada caminho.

### Impressão de grafos

Para a execução do código e subconsequente impressão dos grafos, adjacências, resultados das pesquisas e caminhos, as funções declaradas anteriormente são chamadas dentro da função *main* existente no ficheiro *Source.c.*

*Como é possível ver na Figura 5 – Chamadas às funções criadas* (abaixo)*,* neste caso já existia um ficheiro .bin criado com a informação de um grafo que incluia as adjacências de cada antena, pelo que não era necessário ler o ficheiro de texto, nem voltar a calcular as adjacências, nem voltar a guardar este ficheiro.

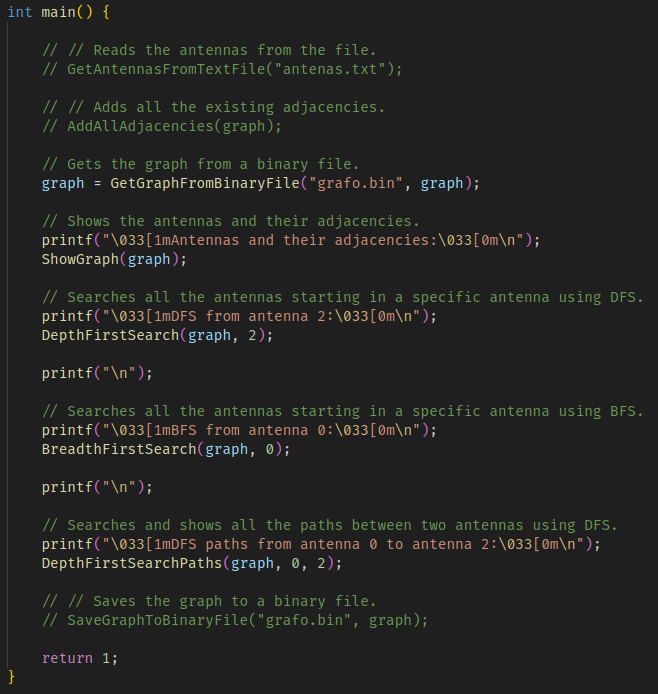


Figura 5 – Chamadas às funções criadas

## Organização do código

O código deste projeto encontra-se estruturado e organizado de maneira a facilitar significativamente a sua leitura, manutenção e escalabilidade. A separação dos ficheiros em "Arquivos de Cabeçalho" (.h) e "Arquivos de Origem" (.c) reflete boas práticas de programação em C, promovendo a modularidade e a reutilização de código.

Cada funcionalidade principal do sistema está devidamente agrupada e documentada e as declarações de funções nos ficheiros .h seguem uma organização lógica.

Um destaque importante é a utilização de #pragma region e #pragma endregion (como demonstrado na *Figura 6 – Exemplo da organização do código* (abaixo)), que permite agrupar logicamente blocos de código relacionados.

Estas diretivas não afetam a compilação, mas são extremamente úteis em editores de texto que as suportam, como o Visual Studio, pois permitem expandir e recolher blocos de código, o que contribui para uma melhor legibilidade e navegação no projeto.

Além disso, os comentários em formato de *docstring* (com @brief, @param, e @return) explicam claramente o propósito de cada função, os seus parâmetros e o valor de retorno, promovendo uma documentação embutida e fácil de entender.

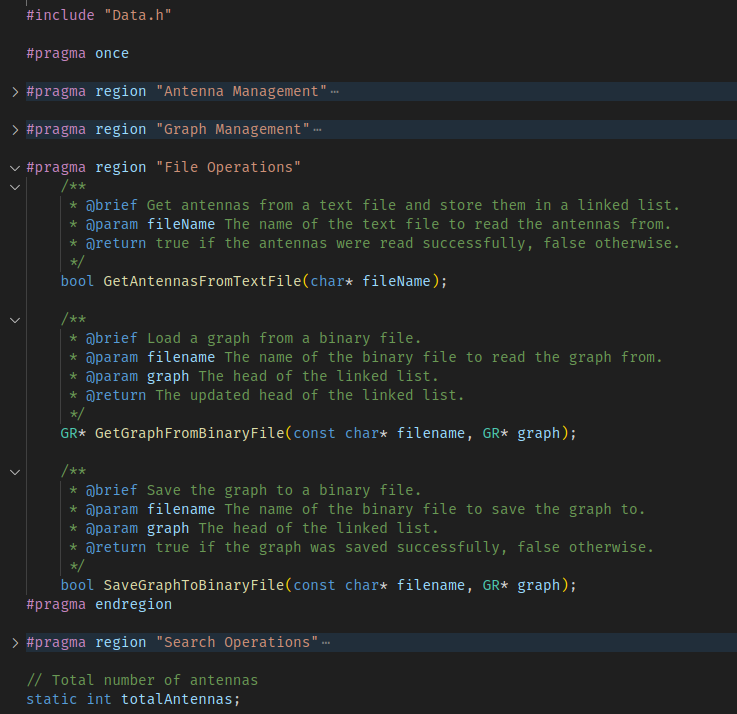


Figura 6 - Exemplo da organização do código

# Resultados

Durante o desenvolvimento da segunda fase do projeto, o programa foi testado com diferentes matrizes de antenas, incluindo cenários de pequena e grande dimensão, e com diversas combinações de frequências.  
Os testes demonstraram que o programa foi capaz de:

* Carregar corretamente um grafo de antenas a partir do ficheiro.
* Guardar um ficheiro com a informação de um grafo.
* Representar as antenas como vértices de um grafo, criando automaticamente as arestas para antenas com frequências iguais.
* Executar os algoritmos de pesquisa em profundidade (DFS) e em largura (BFS) de forma eficiente, listando corretamente os vértices alcançados.
* Identificar e listar todos os caminhos possíveis entre duas antenas selecionadas, garantindo a precisão dos resultados.

A modularidade do código e a utilização de Doxygen para documentação facilitaram a compreensão e manutenção da solução.

# 4. Documentação detalhada

Conforme mencionado no decorrer deste relatório, foi utilizado o Doxygen para a documentação detalhada de cada função e estrutura do projeto.

Isto foi possível graças aos comentários detalhados durante o desenvolvimento, que explicam cada parte do processo. Será ainda possível verificar que cada ação de uma função está devidamente comentada para quem pegar no projeto no futuro poder saber o que cada zona faz, bem como prosseguir caso seja necessário efetuar alguma alteração.

# 5. Conclusão

A realização deste projeto, especialmente na sua segunda fase, permitiu uma compreensão aprofundada dos conceitos de grafos e da sua aplicação prática em C. A transição de listas ligadas para grafos não só ampliou o conhecimento técnico, como também destacou a importância da escolha correta de estruturas de dados na resolução de problemas complexos.

A implementação de algoritmos de pesquisa, como DFS e BFS, reforçou o entendimento de estratégias de navegação em grafos, enquanto o cálculo de caminhos entre antenas demonstrou o poder da modelação com grafos na análise de conectividade. A abordagem modular adotada, juntamente com a documentação clara via Doxygen, garantiu uma solução organizada, fácil de manter e compreender.

A experiência adquirida ao longo deste projeto contribuirá significativamente para o desenvolvimento de futuras competências em programação e resolução de problemas complexos, ficando, assim, melhor preparado para desafios mais avançados na área da engenharia de software.

# 6. Glossário

* **Antena**: Dispositivo que transmite ou recebe sinais de rádio. No contexto do projeto, cada antena é representada por um caracter (frequência) e coordenadas (posição na matriz).
* **Grafo**: Estrutura de dados que interliga antenas.
* **DFS (Depth-First Search)**: Algoritmo de pesquisa em profundidade.
* **BFS (Breadth-First Search)**: Algoritmo de pesquisa em largura.
* **Doxygen**: Ferramenta que gera documentação automaticamente a partir de comentários no código-fonte. A documentação é organizada de forma clara, permitindo que outros programadores compreendam o funcionamento do código.
* **Estruturas de Dados Dinâmicas**: Estruturas de dados cujo tamanho pode variar durante a execução do programa, permitindo alocar memória de forma eficiente. Exemplo: listas ligadas.
* **Lista Ligada**: Estrutura de dados em que cada elemento (ou nó) contém um valor e um apontador para o próximo elemento na lista. Utilizada para armazenar as antenas e localizações afetadas no projeto.
* **Matriz**: Estrutura bidimensional que representa a disposição das antenas na cidade, com cada célula contendo um caracter ou um valor que representa uma antena ou uma localização vazia.
* **Apontador**: Variável que armazena o endereço de memória de outro valor. É amplamente utilizado em linguagens como C para manipular estruturas de dados dinâmicas, como as listas ligadas.

# 7. Referências

* Enunciado do projeto fornecido pela UC Estruturas de Dados Avançadas: <https://elearning.ipca.pt/2425/pluginfile.php/831555/mod_resource/content/1/Projeto_EDA.pdf>
* Código desenvolvido nas aulas: <https://elearning.ipca.pt/2425/course/view.php?id=34124>
* Github do professor Lufer: <https://github.com/luferIPCA/EDA-LESI-2024-2025>
* Repositório Github do projeto: <https://github.com/RJPR95/LESI-eda/tree/main/Projeto2>
* W3Schools – C: <https://www.w3schools.com/c/>
* Doyxgen Wizard: <https://doxygen.nl/manual/doxywizard_usage.html>