Instituto Politécnico do Cavado e Ave

Ano Letivo: 2024/2025

Trabalho Prático

Estruturas de Dados Avançados

Professor:

Luís Ferreira

Nº/Aluno:

31501 - Tomás Afonso Cerqueira Gomes

Data:

Março de 2025

Índice

1. Introdução

1.1 Enquadramento

Este trabalho foi desenvolvido no âmbito da Licenciatura em Sistemas Informáticos, no Instituto Politécnico do Cavado e do Ave, na cadeira de Estruturas de dados Avançados.

1.2 Motivação

Este trabalho tem como objetivo desenvolver uma aplicação em linguagem C que utiliza estruturas de dados avançadas para gerir antenas e analisar os efeitos nefastos causados pela sua interação. O projeto é dividido em duas fases, com diferentes objetivos e complexidade.

Fase 1 – Gestão de antenas e efeitos nefastos

Nesta fase, a aplicação permite carregar antenas a partir de um ficheiro de texto, armazená-las numa lista ligada e calcular os efeitos nefastos entre antenas da mesma frequência que estejam alinhadas e posicionadas a uma distância tal que resulta num ponto de efeito.

Fase 2 – Representação e análise em grafo

A segunda fase do projeto introduz o uso de grafos. As antenas passam a ser vértices ligados entre si através de arestas quando partilham a mesma frequência. O grafo permite realizar percursos como DFS (Depth-First Search) e BFS (Breadth-First Search), bem como guardar toda a estrutura num ficheiro binário.

1.3 Objetivos

O objetivo principal deste trabalho é desenvolver um sistema que calcule os efeitos nefastos causados por antenas, e que represente essa informação num grafo estruturado.

Objetivos da Fase 1:

- Carregar antenas a partir de um ficheiro de texto, com frequência e coordenadas.
- Analisar e calcular os efeitos nefastos com base no alinhamento e na distância entre antenas.
- Armazenar os efeitos nefastos detetados numa lista ligada.
- Listar de forma organizada as antenas e os efeitos no terminal.

Objetivos da Fase 2:

- Construir um grafo a partir das antenas carregadas, utilizando listas ligadas.
- Ligar antenas (vértices) com a mesma frequência através de arestas.
- Implementar e executar os algoritmos DFS e BFS no grafo.
- Armazenar toda a estrutura do grafo num ficheiro binário (grafo.bin).
- Garantir a reutilização do código através de bibliotecas.

1.4 Metodologia de trabalho

Este trabalho foi desenvolvido por etapas, respeitando uma divisão clara entre as duas fases propostas no enunciado.

Fase 1:

Começou-se por analisar o problema e planear a estrutura base do código. Desenvolveram-se as estruturas para representar as antenas e os efeitos, assim como as funções para carregar um ficheiro, processar e listar. A lógica de cálculo dos efeitos foi testada continuamente para garantir o correto alinhamento e distância.

Fase 2:

Após validar a Fase 1, avançou-se para a criação do grafo. Cada antena foi representada como um vértice e foi criada uma ligação (aresta) entre antenas com a mesma frequência. Posteriormente, implementaram-se os algoritmos de percurso (DFS e BFS), bem como uma função para armazenar todo o grafo num ficheiro binário. Também foi introduzido um Makefile mais completo para gerir a compilação de ambas as fases com uso de bibliotecas.

2. Trabalho desenvolvido

2.1 Análise e Especificação

Fase 1:

Nesta fase, o sistema permite:

- Ler as antenas a partir de um ficheiro de texto.
- Calcular os efeitos nefastos através da posição e da frequência de cada antena.
- Armazenar e listar a informação de forma organizada no terminal.

Fase 2:

A aplicação passou a incluir:

- Representação das antenas como vértices de um grafo.
- Criação automática de arestas entre antenas da mesma frequência.
- Realização de percursos no grafo utilizando algoritmos como BFS e DFS.
- Armazenamento da estrutura do grafo num ficheiro binário.
- Reutilização da Fase 1 via biblioteca estática libfase1.a.

2.1.1 Requisitos

Fase 1:

- Gerir antenas e calcular os efeitos nefastos com base em coordenadas e frequência.
- Garantir a não duplicação de efeitos e a sua correta deteção.

Fase 2:

- Representar a rede de antenas como um grafo.
- Ligar antenas com a mesma frequência.
- Permitir análise da conectividade entre antenas.
- Guardar o grafo num ficheiro binário para consulta posterior.

2.1.2 Arquitetura Lógica/Funcional

Fase 1:

- Carregamento dos dados de antenas a partir de ficheiro.
- Cálculo dos efeitos nefastos entre antenas.
- Remoção e listagem das antenas e efeitos.

Fase 2:

- Criação de uma estrutura de grafo com base nas antenas carregadas.
- Ligação automática entre antenas com frequência comum.
- Execução dos algoritmos DFS e BFS sobre o grafo.
- Armazenamento da rede de antenas num ficheiro binário (grafo.bin).

2.1.3 Interação

Fase 1:

O sistema calcula automaticamente os efeitos após o carregamento das antenas e permite listá-los, bem como limpar e remover entradas da lista ligada.

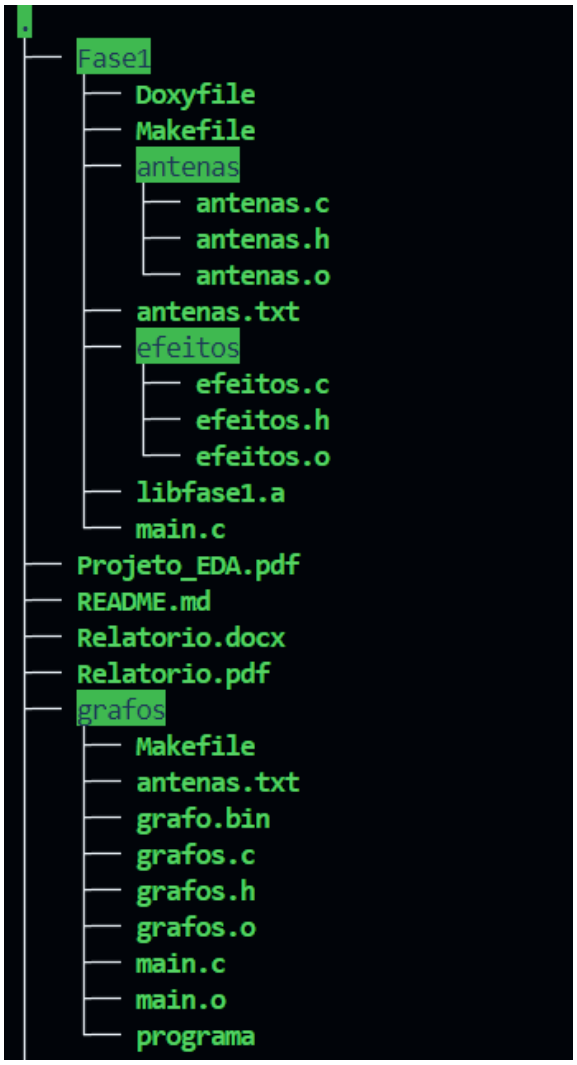
Fase 2:

O utilizador pode visualizar o grafo, realizar percursos DFS ou BFS, e guardar a estrutura final num ficheiro binário. A interação foi mantida simples e com clareza na execução.

2.2 Implementação

2.2.1 Estrutura do Código

O código foi organizado da forma correta e clara, utilizando boas práticas de programação para garantir facilidade de manutenção e leitura. Para isso, dividi o projeto em várias partes e utilizei funções específicas para cada tarefa. A estrutura do código segue a organização por funcionalidades o que facilita a compreensão.



O código foi dividido em diferentes funções e ficheiros, permitindo que cada parte do programa tenha uma função bem definida. Isso facilita a manutenção e a expansão do código no futuro.

2.2.2Antenas

Os ficheiros antenas.c e antenas.h são responsáveis pela gestão das antenas no sistema, incluindo a criação, armazenamento e exibição das antenas. A estrutura do código foi organizada e desenvolvido para utilizar funções específicas para cada tarefa, promovendo a clareza e a reutilização do código.

2.2.3Ficheiro antenas.h

O ficheiro antenas.h contém a estrutura Antena e a declaração das funções que manipulam essas antenas.

A estrutura Antena é o núcleo principal, pois representa uma antena no sistema. Contém um campo para armazenar a frequência e outros dois para armazenar as coordenadas de localização dessa antena (x e y), por outro lado ainda tem um outro campo que é um apontador para o próximo da lista, permitindo a ligação entre as antenas.

Esse ficheiro têm também declaradas todas as funções para gerir da melhor forma todas as antenas.

2.2.4Ficheiro antenas.c

O ficheiro antenas.c contém a implementação das funções declaradas em antenas.h.

carregarAntenasDeFicheiro

Esta função tem como objetivo carregar as antenas a partir de um ficheiro de texto e armazená-las numa lista ligada. A função lê cada linha do ficheiro e verifica cada caractere. Os outros caracteres são considerados frequências de antenas e são inseridos na lista. As coordenadas (x, y) correspondem à posição na linha e na coluna, respetivamente.

inserirAntena

Esta função cria e insere uma nova antena no início da lista ligada. Ao ser chamada, ela aloca memória para a nova antena, preenche os dados (frequência e coordenadas) e insere esta no início da lista. O ponteiro da lista é redirecionado para apontar para a nova antena, fazendo dela a cabeça da lista.

removerAntena

Esta função remove uma antena da lista ligada com base nas coordenadas fornecidas. Ao ser chamada, percorre a lista ligada à procura de uma antena com as coordenadas especificadas. Se a antena for encontrada, a função ajusta os ponteiros da lista para remover a antena e liberta a memória alocada. Se a antena não for encontrada, a função não faz nada. A função retorna 1 se a antena foi removida com sucesso e 0 caso contrário.

limparLista

Esta função percorre toda a lista de antenas e limpa a memória associada a cada antena. Para evitar lixo na memória, a função garante que todos os nós da lista sejam removidos e a memória alocada seja devidamente limpa.

listarAntenas

A função listarAntenas imprime na consola uma tabela com as frequências e as coordenadas de cada antena armazenada na lista.

2.2.5 Gestão dos efeitos nefastos

A gestão dos efeitos nefastos foi desenvolvido de forma a detetar os efeitos causados pelas antenas que estão alinhadas no plano 2D. Esses efeitos nefastos são identificados quando duas antenas da mesma frequência estão alinhadas (na mesma linha, coluna ou diagonal) e uma delas está ao dobro da distância da outra, o que gera um ponto de efeito.

A gestão dos efeitos nefastos foi organizada nos ficheiros efeitos.c e efeitos.h, onde a lógica de cálculo e armazenamento dos efeitos foi implementada.

2.2.6 Ficheiro efeitos.h

O ficheiro efeitos.h contém a declaração da estrutura Efeito e a declaração das funções relacionadas à deteção e gestão dos efeitos nefastos.

2.2.7 Ficheiro efeitos.c

O ficheiro efeitos.c contém a implementação das funções declaradas em efeitos.h. Trata da lógica principal para a obtenção dos efeitos nefastos entre as antenas e do armazenamento desses efeitos numa lista ligada.

calcularEfeito

A função calcularEfeito tem como objetivo verificar se duas antenas da mesma frequência estão alinhadas e se a distância entre elas é tal que gera um efeito nefasto. O cálculo verifica se as antenas estão alinhadas (na mesma linha, coluna ou diagonal) e se a distância entre elas é o dobro. Caso isso seja verdade, calcula a localização do efeito.

efeitoExiste

A função efeitoExiste percorre a lista de efeitos para verificar se já existe um efeito com as coordenadas fornecidas. Se o efeito já existir, a função retorna 1, caso contrário, retorna 0. Isso é útil para garantir que não haja duplicação de efeitos nefastos na lista.

adicionarEfeito

A função adicionarEfeito tem como objetivo adicionar um novo efeito à lista de efeitos nefastos. Esta realiza a inserção de um novo nó na lista ligada de efeitos, sendo que antes de adicionar o efeito, é feita uma verificação para garantir que não há duplicação de efeitos. Se o efeito já existir na lista, a função não faz nenhuma alteração. Caso contrário, cria-se um novo nó com as coordenadas fornecidas e o insere-se no início da lista.

deduzirEfeitosNefastos

A função deduzirEfeitosNefastos percorre todas as antenas da lista, comparando cada par de antenas. Para cada par, a função chama calcularEfeito para verificar se há um efeito nefasto entre elas. Quando um efeito é detetado, este é adicionado à lista de efeitos com a função adicionarEfeito.

listarEfeitos

A função listarEfeitos imprime no terminal a lista de efeitos nefastos. Exibe as coordenadas (x, y) de cada efeito encontrado, mostrando-os de forma tabular, facilitando a visualização dos resultados.

limparEfeitos

Esta função é responsável por limpar a memória alocada na lista de efeitos. Percorre a lista de efeitos e, para cada nó, utiliza o “free” para esvaziar a memória.

2.2.8 Gestão de Grafos

A gestão dos grafos foi desenvolvida com o objetivo de representar as ligações entre antenas que partilham a mesma frequência. Cada antena é tratada como um vértice e cada ligação entre duas antenas da mesma frequência é representada como uma aresta. Esta funcionalidade permite uma melhor análise da estrutura das antenas.

A estrutura do grafo está organizada nos ficheiros grafos.c e grafos.h, onde se encontra a definição das estruturas Vertice, Aresta e GR, bem como a implementação das funções necessárias para o desenvolvimento da estrutura do grafo.

2.2.9 Ficheiro grafos.h

O ficheiro grafos.h contém as definições das estruturas utilizadas na construção do grafo:

- Vertice: representa um nó do grafo, contendo a informação da antena, uma lista de adjacências (arestas), um apontador para o próximo vértice e uma flag de visita.
- Aresta: representa uma ligação entre dois vértices (antenas) com um valor de distância e um apontador para o destino.
- GR: estrutura global que representa o grafo completo, com um apontador para o primeiro vértice e um campo para contar o número total de vértices.

Além das estruturas, o ficheiro também declara todas as funções responsáveis pela manipulação do grafo, como a construção, inserção de arestas, percursos, libertação de memória e armazenamento em ficheiro.

2.2.10 Ficheiro grafos.c

O ficheiro grafos.c contém a implementação das funções declaradas em grafos.h, incluindo a lógica principal de construção do grafo a partir das antenas carregadas e a ligação entre vértices com a mesma frequência.

construirGrafo

A função construirGrafo percorre a lista de antenas e cria um vértice para cada uma delas. Posteriormente, compara todas as antenas entre si e, se duas antenas tiverem a mesma frequência, cria uma aresta entre elas com a distância calculada com base nas suas coordenadas. O resultado é um grafo não-direcionado representado por listas ligadas.

adicionarAresta

A função adicionarAresta cria uma nova aresta entre dois vértices, preenchendo a estrutura Aresta com a distância entre as antenas de origem e destino. Esta nova aresta é inserida no início da lista de adjacências do vértice de origem.

mostrarGrafo

A função mostrarGrafo percorre todos os vértices do grafo e imprime, para cada um, todas as suas adjacências (arestas). É útil para visualizar de forma clara a topologia da rede de antenas.

dfs (Depth-First Search)

A função dfs implementa o algoritmo de pesquisa em profundidade, marcando os vértices visitados a partir de uma antena inicial. Este tipo de percurso permite analisar a conectividade do grafo, explorando um caminho até ao fim antes de retroceder.

bfs (Breadth-First Search)

A função bfs executa uma pesquisa em largura, percorrendo os vértices vizinhos antes de avançar para o próximo nível. Esta abordagem é útil para determinar distâncias mínimas ou verificar camadas de ligação entre antenas.

limparVisitados

Esta função percorre todos os vértices e redefine o campo visitado para 0, permitindo reutilizar o grafo em múltiplas execuções de DFS ou BFS sem interferência.

libertarGrafo

A função libertarGrafo percorre o grafo libertando todas as estruturas dinâmicas alocadas para as listas de vértices e arestas. Garante uma gestão adequada da memória no final do programa.

guardarGrafoBinario

A função guardarGrafoBinario permite guardar toda a estrutura do grafo num ficheiro binário. Para cada vértice, grava-se a antena, o número de arestas e, para cada aresta, a distância e a antena de destino. Esta funcionalidade permite persistir o grafo e reutilizá-lo em futuras execuções ou análises.

2.2.11 Makefile

O Makefile é uma ferramenta essencial para automatizar o processo de compilação e execução do projeto. Ao utilizar o make, conseguimos definir um conjunto de regras para garantir que o código seja compilado corretamente e de forma eficiente, sem a necessidade de realizar manualmente cada etapa da compilação.

Na Fase 1, o Makefile compilava diretamente os ficheiros antenas.c, efeitos.c e main.c. Com a evolução do projeto, foi criada uma biblioteca estática (libfase1.a) contendo todas as funcionalidades da Fase 1, como leitura de antenas e cálculo de efeitos nefastos. Esta biblioteca passou a ser utilizada na Fase 2, facilitando a organização modular e a reutilização de código.

Quando escrito make no terminal, o programa irá ser compilado, por sua vez se fizermos make run, o terminal irá executar o programa, ainda com mais algumas funcionalidades embutidas quando escrito make clean os ficheiros gerados quando ouve a compilação iram desaparecer. Para gerar a documentação automática do código basta utilizar o comando make doc que isso irá acontecer.

3. Analise e discussão dos resultados

O desenvolvimento deste projeto foi acompanhado por constantes testes, o que permitiu detetar e resolver erros à medida que iam surgindo.

Fase 1 – Gestão de Antenas e Efeitos Nefastos

Na primeira fase, o foco esteve na leitura de antenas a partir de ficheiro e na identificação de efeitos nefastos causados pela sua disposição no plano. A maior dificuldade prendeu-se com a lógica de cálculo dos efeitos, em especial a verificação do alinhamento entre antenas e a verificação da distância adequada.

Foi necessário compreender com detalhe o conceito de ponto médio e garantir que o efeito detetado não coincidisse com a posição das antenas de origem. A função calcularEfeito foi ajustada várias vezes para cobrir todos os casos, e funções auxiliares como efeitoExiste evitaram a duplicação de dados na lista de efeitos.

Fase 2 – Construção e Análise de Grafos

A segunda fase trouxe um novo nível de complexidade ao projeto. A conversão das antenas em vértices e a ligação entre antenas com a mesma frequência por arestas exigiram uma nova abordagem estrutural baseada em grafos.

Foi necessário criar novas estruturas (Vertice, Aresta, GR) e garantir a correta gestão para a construção do grafo.

Além disso, a implementação dos algoritmos de pesquisa em profundidade (DFS) e em largura (BFS) demonstrou ser um exercício eficaz de aplicação prática dos conceitos de grafos.

Por fim, a funcionalidade de exportação do grafo para um ficheiro binário foi útil para consolidar a persistência dos dados e demonstrar a capacidade de extensão do sistema.

Avaliação Global

No geral, o sistema demonstrou-se robusto e funcional. Foi possível representar de forma clara tanto os efeitos nefastos quanto as relações entre antenas, com uma separação clara entre lógica de dados, processamento e apresentação. A criação de uma biblioteca estática na Fase 1 também contribuiu para a organização e reutilização eficiente do código na Fase 2.

4. Conclusão

Apesar dessas dificuldades, o projeto foi concluído com sucesso, e as soluções desenvolvidas respondem em conformidade aos problemas propostos.

Toda a lógica para calcular os efeitos nefastos entre antenas apresentou desafios, principalmente para garantir que a distância entre elas fosse corretamente calculada e que as antenas estivessem realmente alinhadas.

Na segunda fase, foi possível aprofundar o domínio sobre estruturas de dados mais avançadas, nomeadamente a construção e manipulação de grafos.

Este projeto revelou-se uma excelente oportunidade para reforçar competências de programação, organização do código, utilização de bibliotecas próprias (.lib), e integração com ferramentas como Makefile e Doxygen. Ensinou também a importância da validação contínua e da documentação durante o desenvolvimento do programa.

5. Referências

Link do repositório do github: https://github.com/tomascerqueiraa/TP---EDA.git

ChatGPT– https://chatgpt.com/

Gemini - https://gemini.google.com/app