IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

RELATÓRIO TRABALHO PRÁTICO - 2ª FASE

Diogo Alexandre Alves da Silva nº 31504

UC: Estruturas de Dados Avançadas

Professor: Luís Ferreira

Março, 2025

Índice

Índice de figuras

Glossário e Siglas ordenado por ordem alfabética

EDA – Estruturas de Dados Avançadas

UC**-** Unidade Curricular

txt- Texto

Resumo

Este trabalho foi desenvolvido no âmbito da *UC* de *EDA*, com o objetivo de aplicar e consolidar a matéria de estruturas de dados dinâmicas, em particular dicionários e grafos. O problema proposto consistia na gestão de um mapa onde havia antenas, identificadas por frequências e localização, onde antenas podias estar conectadas umas às outras.

O projeto foi a criação de uma biblioteca modular em C, capaz de representar e manipular dicionários e grafos, bem como realizar operações de leitura e escrita em ficheiros de texto e binário. A abordagem escolhida deu origem a diferentes *structs* para representar antenas e o dicionário de erros. Também para melhorar ainda mais, a modularização do código e a sua documentação com *Doxygen*.

Introdução

O presente capítulo pretende contextualizar o trabalho realizado ao apresentar a motivação para o seu desenvolvimento, o seu enquadramento e os principais objetivos.

Motivação

No contexto deste projeto, foi proposto o desenvolvimento de uma biblioteca capaz de representar e gerir uma cidade, onde diferentes antenas, posicionadas num mapa bidimensional, podem ser conectada, caso tenham a mesma frequência e existam posições no mapa suscetíveis a esse tipo de impacto.

À parte também foi proposto um desafio adicional de fazer um dicionário para armazenar erros em diferentes línguas para representar os erros que o programa produzisse.

A resolução deste problema permitiu aplicar, de forma prática e consolidada, conceitos fundamentais de grafos e dicionários. O desafio não está apenas na construção de uma biblioteca funcional, mas sobretudo na correta manipulação de memória dinâmica, modularização do código e estruturação de dados, aspetos essenciais no curso.

Enquadramento

Este projeto foi desenvolvido no âmbito da unidade curricular de *Estruturas de Dados Avançadas*, integrada no curso de **Licenciatura em Engenharia Informática**, sob orientação do Professor Luís Ferreira. Trata-se de um trabalho prático individual, inserido na 2.ª fase de avaliação da unidade curricular, com o objetivo de aplicar os conhecimentos adquiridos sobre estruturas de dados dinâmicas, especificamente, grafos.

O trabalho baseia-se na implementação de uma biblioteca em linguagem C, que permite a criação, gestão e visualização de um mapa urbano composto por antenas sintonizadas em diferentes frequências, representadas por caracteres. Sempre que duas antenas com a mesma frequência se encontrem posicionadas próximas uma da outra, é gerado automaticamente uma adjacência, devendo o sistema detetar e representar a mesmo. Para além da gestão das listas dinâmicas, o projeto envolveu também a manipulação de ficheiros (texto e binário), modularização do código e documentação técnica utilizando Doxygen.

Estado da Arte

O presente capítulo pretende contextualizar as ferramentas e estruturas utilizadas ao longo do projeto para realizar o trabalho proposto.

Conceitos / Fundamentos Teóricos

C

"C é uma linguagem de programação compilada de propósito geral, estruturada, imperativa, procedural, padronizada pela Organização Internacional para Padronização (ISO), criada em 1972 por Dennis Ritchie na empresa AT&T Bell Labs para desenvolvimento do sistema operacional Unix (originalmente escrito em Assembly)."(Referência: C : Wikipédia, 2025)

Doxygen

"O Doxygen é um gerador de documentação amplamente utilizado ferramenta no desenvolvimento de software. Automatiza a geração de documentação a partir do código-fonte comentários, analisar informação sobre classes, funções e variáveis para produzir saída em formatos como HTML e PDF. Ao simplificar e uniformizar o processo de documentação, O Doxygen melhora a colaboração e a manutenção em diversas linguagens de programação e projetos escalas."
(Referência traduzida com Google Tradutor:Doxygen : Site oficial Doxygen, 2025)

Grafos

"*Trata-se de um modelo para representar conexões entre qualquer tipo de objeto. Percebe-se que há dois tipos de estruturas: o vértice, representado por um círculo, e as arestas, representadas por uma linha.* " (Grafos: mirkoperkusich , 2025)

Dicionários

"A estrutura de dados Dicionário é uma das mais importantes e utilizadas na programação [...] Um dicionário é uma coleção de pares chave-valor, onde cada chave é única e associada a um valor específico. Esse tipo de estrutura permite armazenar e acessar valores de forma rápida e eficiente, com base em suas chaves, ao invés de percorrer a coleção inteira."(Dicionários: luisdev.com.br, 2025)

Vantagem em utilizar Dicionários/Grafos perante *arrays*

Dicionários e grafos oferecem várias vantagens importantes em relação aos arrays, principalmente devido à capacidade de crescimento dinâmico e alocação flexível de memória, enquanto os arrays precisam de um bloco contínuo de memória. Isto pode limitar a sua expansão e a sua eficiência em certos cenários, dicionários e grafos (implementados com listas ligadas) alocam memória sob demanda, sem exigir continuidade no armazenamento. Além disso, dicionários permitem mapeamentos diretos entre chaves e valores, enquanto grafos podem representar relações complexas

entre elementos. Arrays, por outro lado, embora tenham acesso rápido a elementos por índice, exigem estruturas adicionais para simular relacionamentos.

Modularização e Separação de Código

Em C, a separação entre ficheiros .h (headers) e .c (implementações) permite organizar o código de forma lógica e reutilizável. Esta abordagem facilita a manutenção e a legibilidade do programa.

Alocação Dinâmica de Memória e Manipulação de Ficheiros em C

A linguagem C fornece mecanismos explícitos para a alocação dinâmica de memória através das funções *malloc*, *calloc*, *realloc* e *free*, da biblioteca <*stdlib.h*> e a manipulação de ficheiros em C é realizada através das funções da biblioteca <*stdio.h*>, como *fopen*, *fscanf*, *fprintf*, *fread* e *fwrite*.

Problema

O problema apresentado foi o seguinte:

Dado um mapa num ficheiro de texto (como demonstrado abaixo), o aluno deveria desenvolver uma biblioteca que fizesse as seguintes operações com o âmbito de aprofundar nas listas ligadas:

- Gestão de Antenas
  - Definição da estrutura, adicionar e remover
- Dedução automática das relações
  - Definição da estrutura, cálculo das posições e inserção
- Leitura e Escritura em ficheiros
  - Ficheiros de texto
  - Ficheiros binários
- Listagem das Antenas e Adjacências

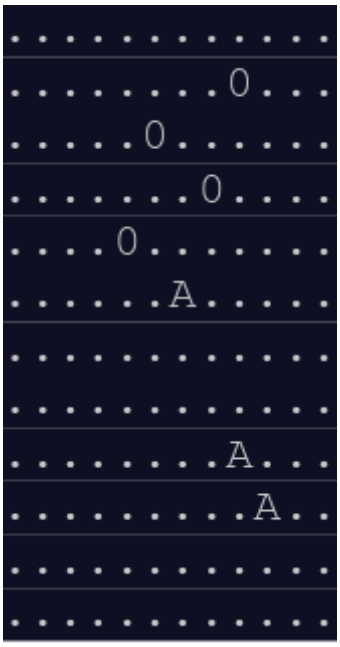

Imagem 1: Exemplo de mapa da cidade

Com estas informações teriam que ser feitas pesquisas, percursos e interseções de antenas

Abordagem

A abordagem escolhida para a resolução do problema foi a implementação de . Essa escolha deve-se à sua eficiência e simplicidade frente às exigências do projeto nas funções para trabalhar com estruturas de dados dinâmicas. O objetivo também foi aprofundar o conhecimento na definição, manipulação e uso prático de dicionários e grafos, explorando conceitos como a alocação dinâmica de memória e algoritmos específicos para percorrer o resto.

Implementação

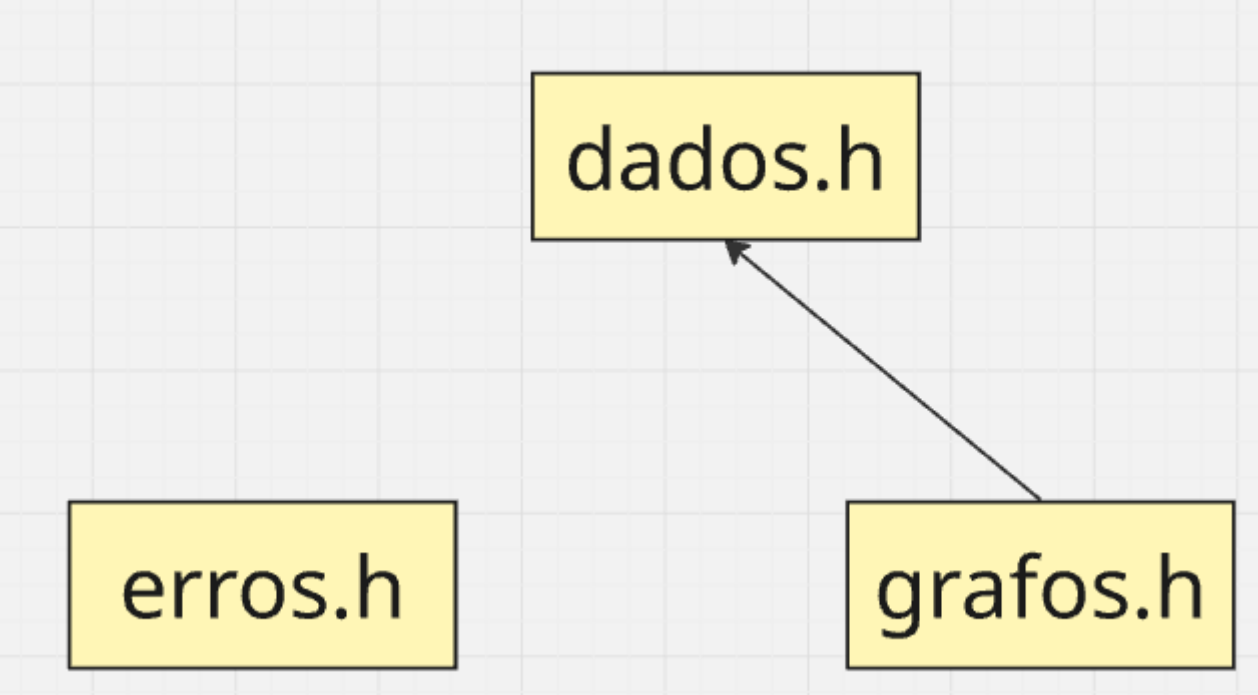

Imagem 2: Estruturas de ficheiros

Desenvolvimento

O presente capítulo pretende mostrar as soluções e o desenvolvimento ao longo do projeto.

Para iniciar este projeto teve de decidir como iria definir as regras/instruções de utilização do programa sendo elas:

- Apenas uma antena em cada posição.
- Apenas uma adjacência para os mesmos dois vértices
- Uma antena está ligada a outra caso se encontre num raio de 2 de outra antena com a mesma frequencia, ou seja, uma antena que esteja em (3,3) tem um alcance de (1,1) a (5,5).

erros.h/erros.c

Nesta parte da biblioteca encontram-se funções, e estruturas para criar um dicionário de erros.

Com as structs utilizaram-se as seguintes:

	Mensagem
int	idMensagem
char*	mensagem
Mensagem*	proxima

	LinguaDicionario
int	id
char[6]	linguaCod
Mensagem*	Mensagens
LinguaDicionario*	prox

A estrutura destes ficheiros podem resumir-se em duas função leFicheiro, ErroPorPais_Cod, limpaMemoriaErros.

leFicheiro

- Recebe:
  - Ficheiro a ser lido
- Devolve
  - Inteiro

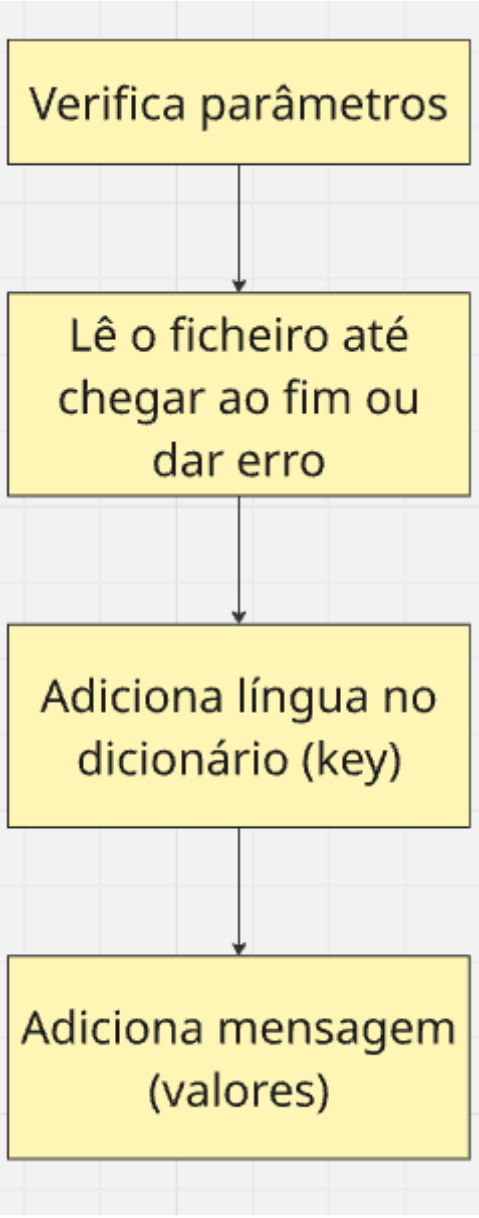

Imagem 3: Funcionamento da função lêFicheiro

Para adicionar as línguas e as mensagens utilizei duas funções (uma para cada tipo) em que utilizei a mesma formúla da 1ª fase em que tinha de se manipular listas ligadas simples e para isso utilizei uma função Cria e uma função Insere para cada tipo de estrutura.

Esta função retorna um inteiro que significa o sucesso da operação.

O ficheiro lido é um ficheiro TXT em que são colocados os erros da seguinte forma:

- Código da língua;Código do erro;Mensagem do Erro;

ErroPorPais_Cod

Esta função serve para alguém que queira utilizar o dicionário só precisa de utilizar a função colocando os códigos (do país e do erro) que a função devolve automaticamente a mensagem.

- Recebe
  - Código do país
  - Código do erro
- Devolve
  - Mensagem de erro

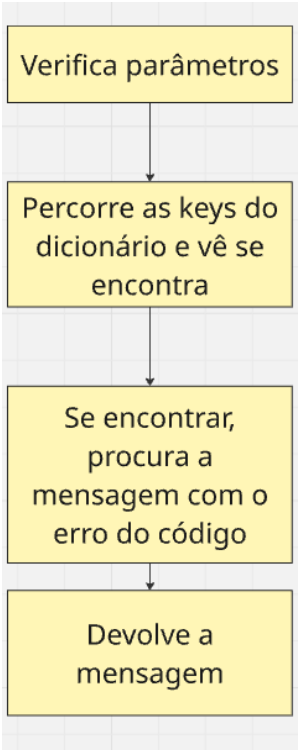

Imagem 4: Funcionamento, função ErroPorPais_Cod

limpaMemoriaErros

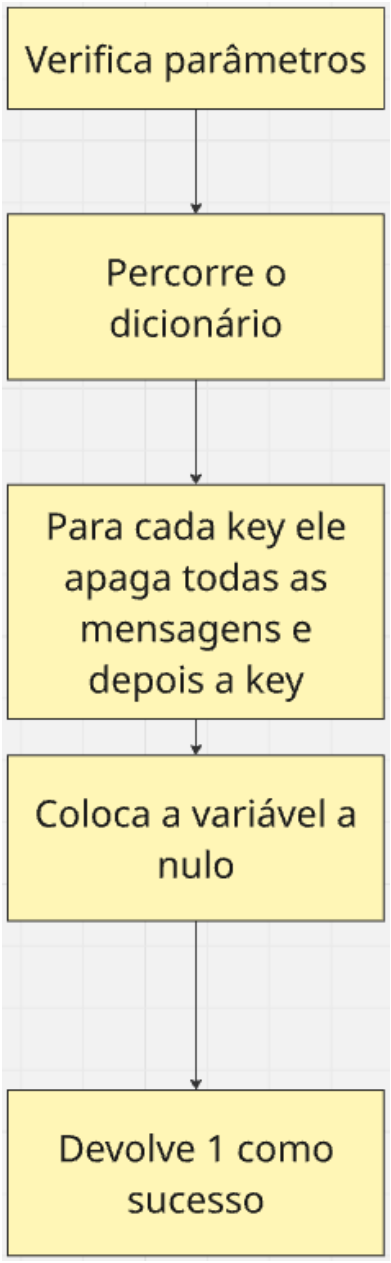

Imagem 5: Funcionamento da limpaMemoriaErros

dados.h

Deste header foi utilizada a estrutura *CasaF* que serviu para armazenar antenas:

	CasaF
char	Frequência
int	linha
int	coluna

grafos.h/grafos.c

Nestes ficheiros encontram-se implementadas as principais funções responsáveis pela resolução do problema dos grafos. Estas funções tratam da criação, gestão e análise da estrutura de grafos, permitindo, por exemplo, a inserção de vértices e arestas, bem como a aplicação de algoritmos para encontrar caminhos, calcular distâncias mínimas ou verificar conexões entre os vértices.

Para utilizar os grafos foram feitas três estruturas:

	Vertice
CasaF*	antena
Adj*	adjacências
Vertice*	prox
int	visitado

	Adj
Vertice*	v1
Vertice*	v2
Adj*	prox

	Grafo
Vertice*	h

E posteriormente umas estruturas para algoritmos posteriores:

	Caminho
Vertice*	vertices[100]
int	tamanho

	ListaDeCaminhos
Caminho	caminhos[100]
int	total

(100 foi um valor max estipulado que pode ser mudado posteriormente, foi feito com variaveis globais para maior flexibilidade).

Funções de gestão de grafos:

- *Grafo* InicializaGrafo*
- *Vertice* CriarVertice*
- *CasaF* CriarAntena*
- *Adj* CriarAdj*
- *Vertice* ProcuraVertice*
- *Vertice* ProcuraVerticeAntena*
- *int PosicaoVertice é*
- *Grafo* InsereVertice*
- *Grafo* insereAdjacencias*
- *Vertice* insereAdjacenciaVertice*
- *int limpaAdj*
- *Grafo* limpaV*
- *int limpaConteudoGrafo*
- *int limpaGrafo*

Estas funções são funções que são parecidas com as funções da primeira fase do trabalho (listas ligadas simples) pois o grafo são 1 lista ligada (Vértices) que está ligada a várias listas ligadas (Adjacências). A diferença principal está nas funções de inserir (pois o processo para encontrar o caminho para inserir é diferente).

Como por exemplo

Vertice* insereAdjacencias

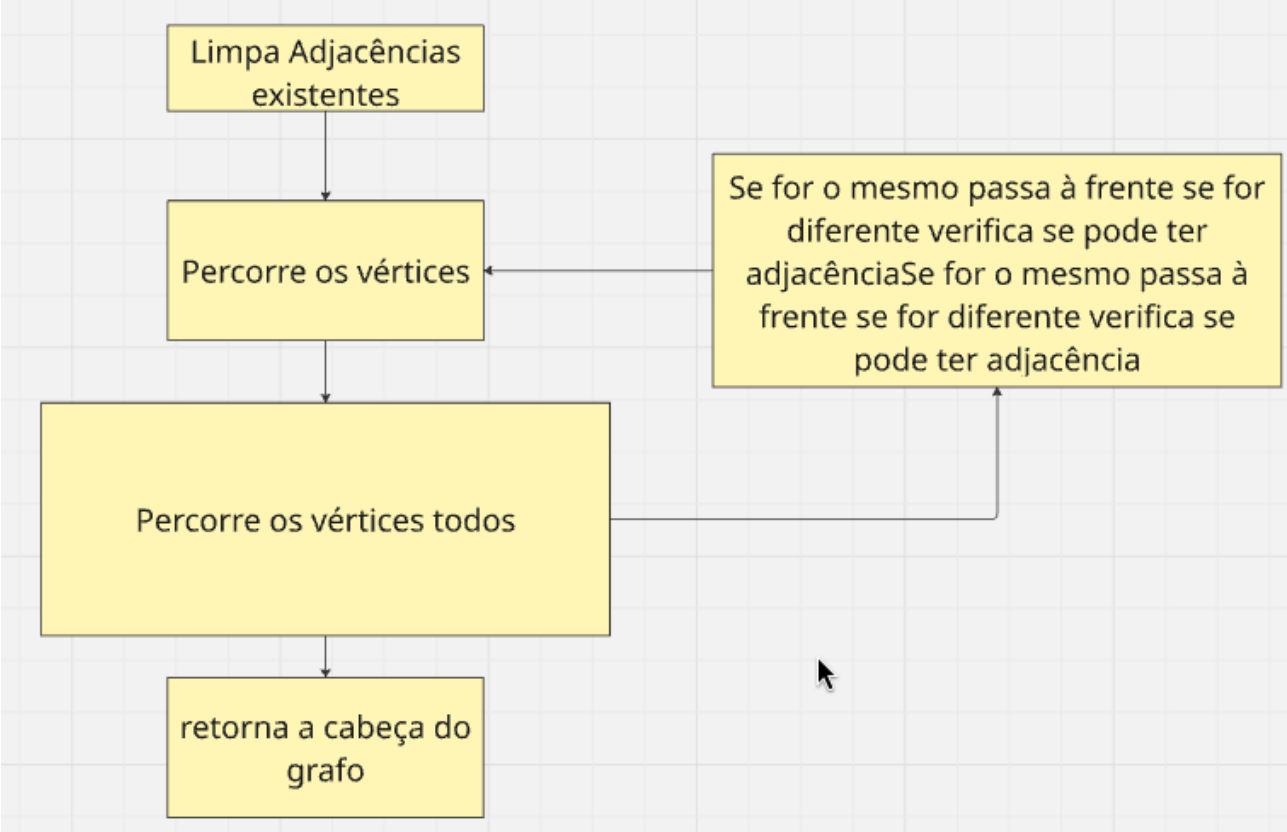

Imagem 6: Funcionamento da Função insereAdjacencias()

A função *insereAdjacencias* é um bom exemplo de como percorrer o grafo deste trabalho, nesta função ao percorrer os vértices e ver que pode adicionar ele chama a função *insereAdjacenciaVertice()* que percorre as adjacências do vértice e insere no fim a adjacência.

Grafo* leTxtGrafos

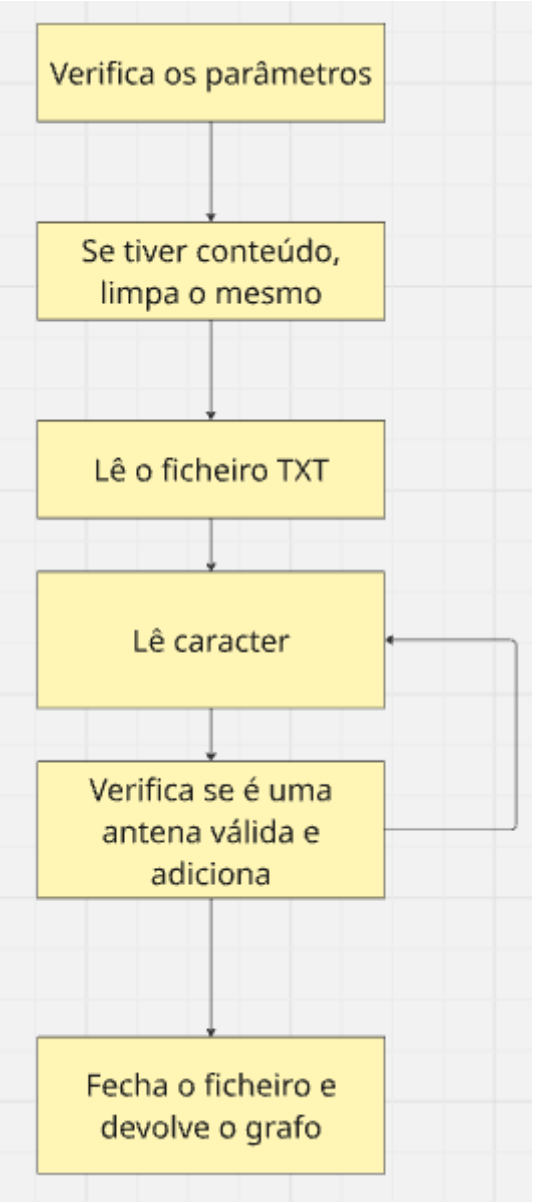

Imagem 7Funcionamento da função leTxtGrafos()

Grafo* DFS_CaminhoGrafo e Grafo* DFSUtil

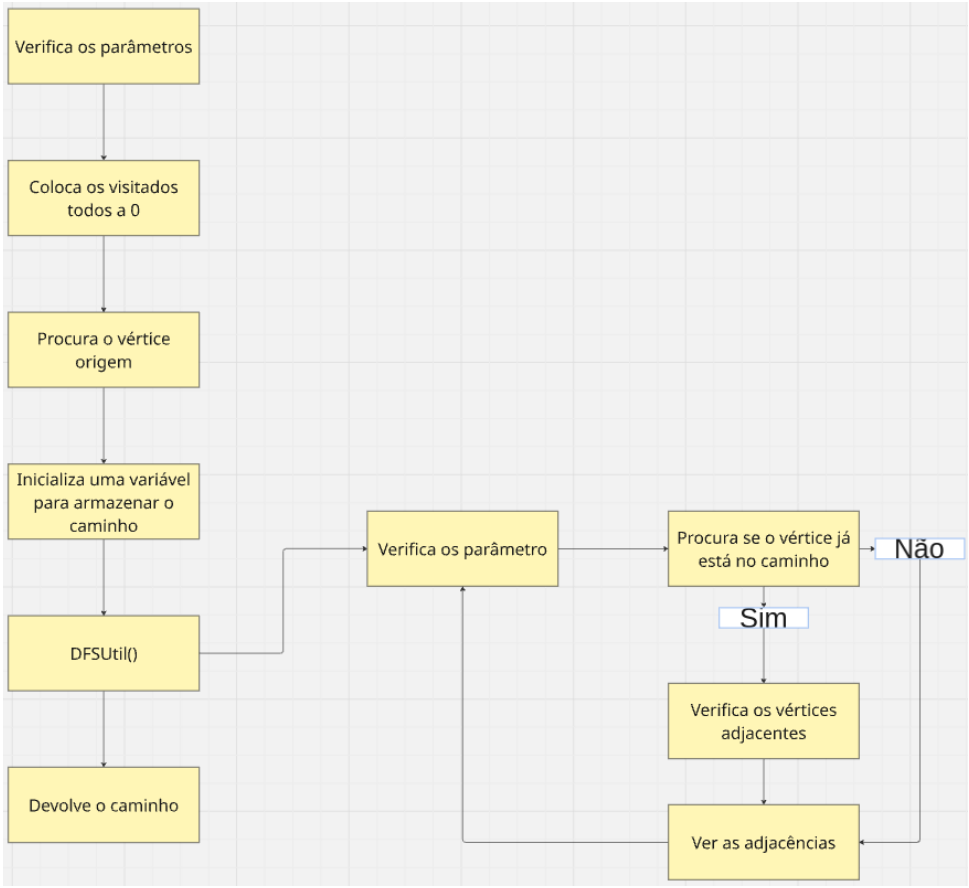

Imagem 8Funcionamento das Funcões DFS_CaminhoGrafo e DFSUtil

Grafo* escreveFicheiroBinario e Grafo* leFicheiroBinario

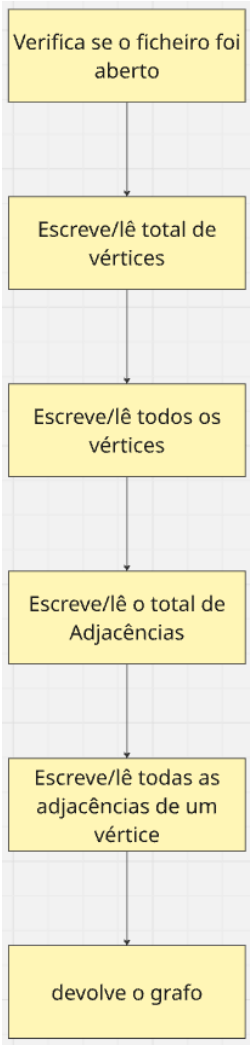

Imagem 9Funcionamento do escrever/ler ficheiros em binário para manipular o gráfico

BFS

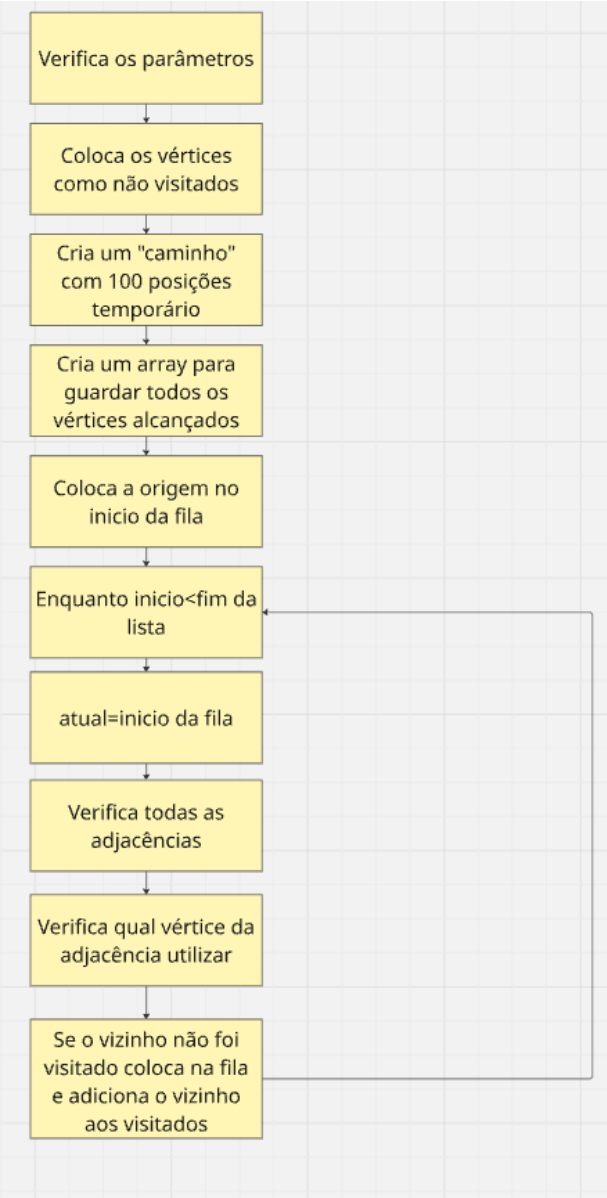

Imagem 10Funcionamento do algoritmo BFS

CalcularCaminhosSimples e caminhosEntreDoisVertices

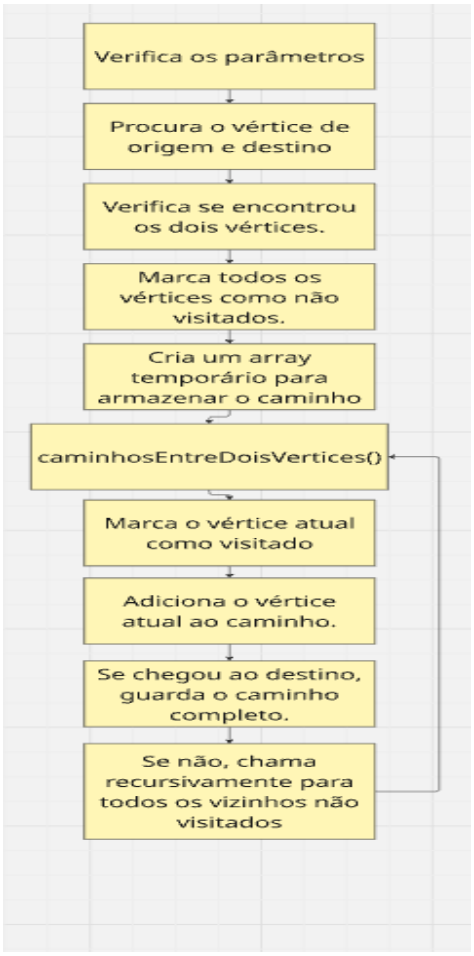

Imagem 11Funcionamento para encontrar todos os caminhos

E no fim ele retorna todos os caminhos possíveis até ao destino.

int detectarCruzamentoGeral

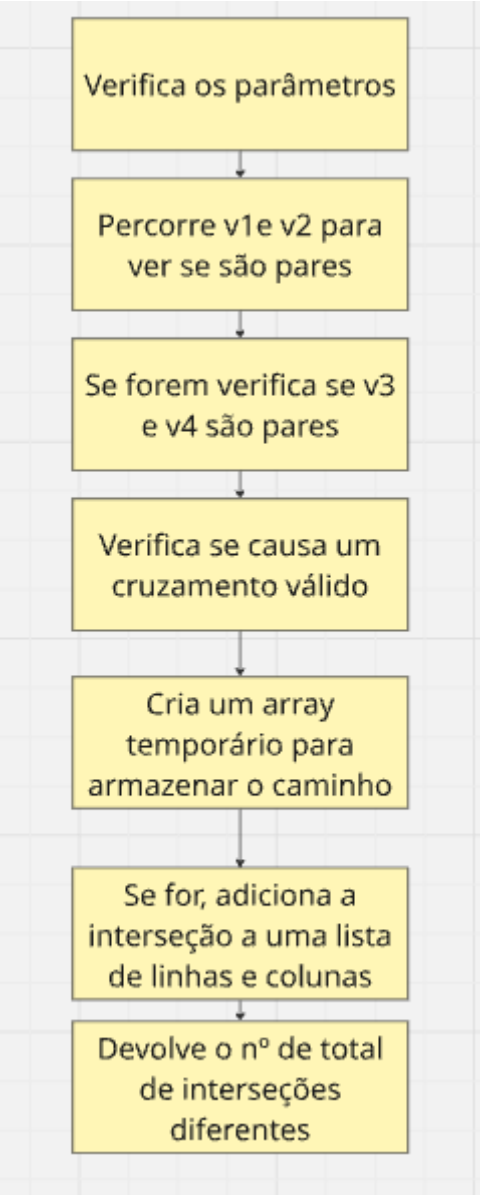

Imagem 12Funcionamento da função detectarCruzamentoGeral

Funções de listar

- int mostraVerticesCaminho
- int mostraVertices
- int mostraVisitados

- int imprimirCaminhosSimples
- int imprimirCruzamentosGerais

Repositório GitHub:

Repositório GitHub

Conclusão

Nesta segunda fase do projeto, foi implementada uma solução baseada em grafos para representar e analisar as antenas de uma cidade e suas interligações. Cada antena foi representada como um vértice e as arestas foram estabelecidas entre antenas com frequências de ressonância iguais, permitindo modelar a rede de forma eficiente.

Foram desenvolvidas as seguintes funcionalidades:

- Representação de grafos com vértices dinâmicos;
- Dicionário de erros
- Algoritmos de procura em profundidade (DFS) e largura (BFS);
- Cálculo de todos os caminhos simples entre duas antenas;
- Identificação de intersecções entre pares de antenas com frequências distintas.

Ao longo do trabalho, aplicaram-se técnicas como recursão, backtracking e manipulação de estruturas de dados dinâmicas, o que reforçou conhecimento prático sobre teoria de grafos e algoritmos clássicos em C, com isto melhorando também o conhecimento técnico.

No fim deste projeto, o mesmo mostrou-se uma forma interessante de aprofundar os conhecimentos dados ao longo das aulas.

Referências

C : Wikipédia, 2025

Doxygen : Site oficial Doxygen, 2025

Grafos: mirkoperkusich , 2025

Dicionários: luisdev.com.br, 2025