Instituto Politécnico do Cávado e do Ave

Escola Superior de Tecnologia

Licenciatura

em

Engenharia de Sistemas Informáticos

Projeto EDA

Luís Pedro Pereira Freitas – a23008

Barcelos, março de 2025

Índice

Índice de Figuras

1. Introdução

1.1 Motivação

O problema abordado neste projeto está relacionado com a **distribuição de antenas numa cidade** e os impactos negativos resultantes da **interferência entre antenas com a mesma frequência de ressonância**. Esta interferência pode gerar efeitos nefastos em determinadas localizações, comprometendo a qualidade das comunicações e provocando potenciais falhas no sistema.

A motivação para a realização deste projeto assenta na necessidade de desenvolver uma solução computacional eficiente que **permita identificar, analisar e minimizar tais interferências**, reforçando simultaneamente os conhecimentos adquiridos na unidade curricular de Estruturas de Dados Avançadas (EDA). Pretende-se assim criar uma ferramenta prática que simule um cenário realista de telecomunicações, facilitando a gestão das infraestruturas e promovendo boas práticas de programação, nomeadamente através da utilização de estruturas de dados dinâmicas, modularização do código e documentação com Doxygen.

.

1.2 Enquadramento

A cidade em estudo é representada por uma matriz bidimensional, onde cada célula pode conter uma antena com uma determinada frequência (representada por um caractere) ou estar vazia. Antenas com a mesma frequência podem provocar efeitos nefastos em localizações perfeitamente alinhadas e equidistantes entre si, quando uma das antenas se encontra a dobro da distância da outra. Estes pontos críticos são identificados de acordo com uma lógica geométrica rigorosa e devem ser representados e analisados no sistema.

Este projeto é desenvolvido em duas fases. Na Fase 1, recorre-se a listas ligadas para a representação e manipulação dos dados das antenas e efeitos nefastos. Na Fase 2, a abordagem evolui para a utilização de estruturas de grafos, permitindo explorar relações complexas entre antenas e aplicar algoritmos de procura e análise de conexões.

1.3 Objetivos

O principal objetivo deste projeto é a criação de um sistema funcional que permita:

- **Identificar e registar** a localização das antenas existentes;
- **Calcular automaticamente os efeitos nefastos** causados por antenas com a mesma frequência;
- **Apresentar a informação de forma clara e acessível** ao utilizador final, quer em formato tabular, quer gráfico;
- **Sugerir melhorias e otimizações na distribuição das antenas**, com vista à redução dos impactos negativos;
- **Aplicar e consolidar conhecimentos práticos** sobre estruturas de dados dinâmicas e grafos, reforçando competências em programação na linguagem C;
- **Garantir a organização e manutenção do código**, através de uma documentação clara e detalhada com Doxygen.

1.4 Metodologia de Trabalho

A metodologia aplicada neste projeto assenta numa abordagem **iterativa e modular**, com divisão em duas fases principais:

- **Fase 1 – Listas Ligadas**: Implementação de uma estrutura de lista ligada simples para armazenar dados das antenas e dos efeitos nefastos. Serão desenvolvidas operações para inserção, remoção, listagem e cálculo de localizações críticas com base nas distâncias e frequências das antenas.
- **Fase 2 – Grafos**: Construção de um grafo orientado, onde cada antena representa um vértice, e as arestas ligam apenas antenas com a mesma frequência.

Ao longo do desenvolvimento, serão aplicadas técnicas de verificação e validação, garantindo a robustez do sistema e a precisão dos resultados obtidos.

- **1.5 Plano de Trabalho**

O projeto segue o seguinte plano de execução:

- **Semana 1-2**: Compreensão do problema e definição das estruturas de dados para a Fase 1;
- **Semana 3-4**: Implementação das listas ligadas, leitura de ficheiros e cálculo dos efeitos nefastos;
- **Semana 5**: Testes e entrega da Fase 1 (até 30 de março de 2025);
- **Semana 6-7**: Estudo teórico de grafos e estruturação da Fase 2;
- **Semana 8-9**: Programação das operações com grafos.

1.5 Plano de Trabalho

O plano de trabalho encontra-se estruturado de acordo com as fases de desenvolvimento: Analise do problema.

- Implementação das funcionalidades de listas ligadas.
- Validação e entrega da fase 1.
- Estudo de grafos.
- Implementação das funcionalidades de grafos
- Validação e entrega da fase 2.

1.6 Link Repositório GitHub

https://github.com/Luisfreitas135/LESI-EDA.git

2. Trabalho Desenvolvido

- ## Fase 1 – Listas Ligadas

Nesta fase, foi implementado um sistema de gestão de antenas recorrendo a listas ligadas simples. Cada antena é caracterizada por uma frequência (representada por um caractere) e pelas suas coordenadas (x, y) numa matriz que representa o mapa da cidade.

O programa permite:

- Carregar automaticamente as antenas a partir de um ficheiro de texto;
- Inserir e remover antenas da lista;
- Calcular automaticamente os efeitos nefastos, que ocorrem quando duas antenas da mesma frequência estão alinhadas e a uma distância proporcional (uma ao dobro da outra);
- Representar os efeitos nefastos como uma nova lista ligada;
- Listar na consola, de forma tabular, todas as antenas e as localizações com efeito nefasto.

Este módulo foi essencial para consolidar os conhecimentos sobre manipulação de memória dinâmica e estruturas sequenciais não-contíguas.

- ## Fase 2 – Grafos

A segunda fase consistiu na transição da estrutura de dados baseada em listas ligadas para um modelo baseado em grafos. Cada antena foi modelada como um vértice e as conexões (arestas) foram estabelecidas apenas entre antenas com a mesma frequência de ressonância.

As principais funcionalidades implementadas incluem:

- Construção automática do grafo a partir do mesmo ficheiro de texto da Fase 1;
- Armazenamento e leitura do grafo num ficheiro binário;
- Visualização das antenas carregadas;
- Realização de procura em profundidade (DFS) a partir de uma antena selecionada;

- Determinação de todos os caminhos possíveis entre duas antenas;
- Identificação de interseções entre antenas com frequências diferentes (isto é, coordenadas ocupadas simultaneamente por duas frequências distintas).

Esta fase permitiu aplicar conceitos avançados da teoria dos grafos e estruturar o código de forma modular e reutilizável.

3. Implementação – Fase 1

- criarAntena – aloca uma nova antena

```c
Antena* criarAntena(char freq, int x, int y) {
    Antena *nova = (Antena*)malloc(sizeof(Antena));
    if (!nova) return NULL;
    nova->frequencia = freq;
    nova->x = x;
    nova->y = y;
    nova->prox = NULL;
    return nova;
}
```

Figura 1 - Método criarAntena

- inserirAntena – insere uma antena na lista

```c
void inserirAntena(Antena **lista, char freq, int x, int y) {
    Antena *nova = criarAntena(freq, x, y);
    if (!nova) return;
    nova->prox = *lista;
    *lista = nova;
}
```

Figura 2 - Método inserirAntena

- removerAntena – remove uma antena por coordenadas

```c
void removerAntena(Antena **lista, int x, int y) {
    Antena *atual = *lista, *anterior = NULL;
    while (atual) {
        if (atual->x == x && atual->y == y) {
            if (anterior)
                anterior->prox = atual->prox;
            else
                *lista = atual->prox;
            free(atual);
            return;
        }
        anterior = atual;
        atual = atual->prox;
    }
}
```

Figura 3- Método removerAntena

- carregarDeFicheiro – lê a grelha do ficheiro antenas.txt

```c
Antena* carregarDeFicheiro(const char *nomeFicheiro, int *linhas, int *colunas) {
    FILE *file = fopen(nomeFicheiro, "r");
    if (!file) return NULL;
    Antena *lista = NULL;
    char linha[MAX_COLUNAS];
    int y = 0;
    while (fgets(linha, sizeof(linha), file)) {
        linha[strcspn(linha, "\n")] = 0;
        *colunas = strlen(linha);
        for (int x = 0; x < *colunas; x++) {
            if (linha[x] != '.') {
                inserirAntena(&lista, linha[x], x, y);
            }
        }
        y++;
    }
    *linhas = y;
    fclose(file);
    return lista;
}
```

Figura 4 - Método carregarDeFicheiro

- calcularEfeitosNefastos – identifica os efeitos nefastos com base nas posições das antenas

```c
void calcularEfeitosNefastos(Antena *lista, EfeitoNefasto **efeitos) {
    for (Antena *a1 = lista; a1 != NULL; a1 = a1->prox) {
        for (Antena *a2 = lista; a2 != NULL; a2 = a2->prox) {
            if (a1 != a2 && a1->frequencia == a2->frequencia) {
                int dx = a2->x - a1->x;
                int dy = a2->y - a1->y;

                // Verifica se estão alinhadas (linha, coluna ou diagonal)
                if ((dx == 0 || dy == 0 || abs(dx) == abs(dy))) {
                    // Verifica se a2 está ao dobro da distância de a1 (ou vice-versa)
                    if ((dx % 2 == 0) && (dy % 2 == 0)) {
                        int mx = (a1->x + a2->x) / 2;
                        int my = (a1->y + a2->y) / 2;

                        // Calcula o vetor de deslocamento do ponto médio para gerar efeitos nefastos
                        int ex = mx - dx / 2;
                        int ey = my - dy / 2;
                        int ex2 = mx + dx / 2;
                        int ey2 = my + dy / 2;

                        // Adiciona os dois pontos espelhados
                        if (!efeitoExiste(*efeitos, ex, ey))
                            adicionarEfeitoNefasto(efeitos, ex, ey);
                        if (!efeitoExiste(*efeitos, ex2, ey2))
                            adicionarEfeitoNefasto(efeitos, ex2, ey2);
                    }
                }
            }
        }
    }
}
```

Figura 5 - Método calcularEfeitosNefastos

- efeitoExiste – verifica se um efeito já está registado

```c
int efeitoExiste(EfeitoNefasto *lista, int x, int y) {
    while (lista) {
        if (lista->x == x && lista->y == y)
            return 1;
        lista = lista->prox;
    }
    return 0;
}
```

Figura 6 - Método efeitoExiste

- adicionarEfeitoNefasto - adiciona um efeito nefasto à lista

```c
void adicionarEfeitoNefasto(EfeitoNefasto **lista, int x, int y) {
    if (efeitoExiste(*lista, x, y)) return;
    EfeitoNefasto *novo = (EfeitoNefasto*)malloc(sizeof(EfeitoNefasto));
    if (!novo) return;
    novo->x = x;
    novo->y = y;
    novo->prox = *lista;
    *lista = novo;
}
```

Figura 7 - Método adicionarEfeitoNefasto

- listarAntenas – imprimem as listas

```c
void listarAntenas(Antena *lista) {
    printf("Lista de antenas:\n");
    while (lista) {
        printf("Antena %c em (%d, %d)\n", lista->frequencia, lista->x, lista->y);
        lista = lista->prox;
    }
}
```

Figura 8 - Método listarAntenas

- listarEfeitosNefastos – imprimem as listas

```c
void listarEfeitosNefastos(EfeitoNefasto *lista) {
    printf("\nLocais com efeito nefasto:\n");
    while (lista) {
        printf("Efeito nefasto em (%d, %d)\n", lista->x, lista->y);
        lista = lista->prox;
    }
}
```

Figura 9 - Método listarEfeitosNefastos

4. Implementação – Fase 2

- criarGrafo – cria e inicializa uma estrutura de grafo vazia

```
Grafo* criarGrafo() {
    Grafo *g = malloc(sizeof(Grafo));
    g->vertices = NULL;
    return g;
}
```

Figura 10 - criarGrafo

- adicionarVertice – adiciona um novo vértice ao grafo, com frequência e coordenadas dadas.

```
Vertice* adicionarVertice(Grafo *g, char freq, int x, int y) {
    Vertice *novo = malloc(sizeof(Vertice));
    novo->frequencia = freq;
    novo->x = x;
    novo->y = y;
    novo->arestas = NULL;
    novo->prox = g->vertices;
    g->vertices = novo;
    return novo;
}
```

Figura 11 - adicionarVertice

- adicionarAresta – cria uma ligação (aresta) entre dois vértices.

```
void adicionarAresta(Vertice *v1, Vertice *v2) {
    Aresta *a = malloc(sizeof(Aresta));
    a->destino = v2;
    a->prox = v1->arestas;
    v1->arestas = a;
}
```

Figura 12 - adicionarAresta

- construirGrafosDeFicheiro – lê um ficheiro de texto com o mapa de antenas e constrói o grafo, ligando automaticamente antenas com a mesma frequência.

```c
int construirGrafoDeFicheiro(Grafo *g, const char *nomeFicheiro) {
    FILE *f = fopen(nomeFicheiro, "r");
    if (!f) return 0;

    char linha[100];
    int y = 0;
    Vertice *todos[500];
    int count = 0;

    while (fgets(linha, sizeof(linha), f)) {
        linha[strcspn(linha, "\n")] = 0;
        int col = strlen(linha);
        for (int x = 0; x < col; x++) {
            if (linha[x] != '.') {
                todos[count++] = adicionarVertice(g, linha[x], x, y);
            }
        }
        y++;
    }
    fclose(f);

    for (int i = 0; i < count; i++) {
        for (int j = i + 1; j < count; j++) {
            if (todos[i]->frequencia == todos[j]->frequencia) {
                adicionarAresta(todos[i], todos[j]);
                adicionarAresta(todos[j], todos[i]);
            }
        }
    }
    return 1;
}
```

Figura 13 - construirGrafosDeFicheiro

- guardarGrafosBinário – guarda os vértices do grafo num ficheiro binário.

```c
int guardarGrafoBinario(Grafo *g, const char *ficheiro) {
    FILE *f = fopen(ficheiro, "wb");
    if (!f) return 0;
    for (Vertice *v = g->vertices; v; v = v->prox) {
        fwrite(&v->frequencia, sizeof(char), 1, f);
        fwrite(&v->x, sizeof(int), 1, f);
        fwrite(&v->y, sizeof(int), 1, f);
    }
    fclose(f);
    return 1;
}
```

Figura 14 - guardarGrafosBinário

- carregarGrafoBinario – carrega os vértices do grafo a partir de um ficheiro binário.

```c
int carregarGrafoBinario(Grafo *g, const char *ficheiro) {
    FILE *f = fopen(ficheiro, "rb");
    if (!f) return 0;

    char freq;
    int x, y;
    while (fread(&freq, sizeof(char), 1, f) &&
           fread(&x, sizeof(int), 1, f) &&
           fread(&y, sizeof(int), 1, f)) {
        adicionarVertice(g, freq, x, y);
    }
    fclose(f);
    return 1;
}
```

Figura 15 - carregarGrafoBinario

- conectarVerticesMesmaFrequencia - percorre todos os vértices e cria arestas entre os que têm a mesma frequência.

```c
void conectarVerticesMesmaFrequencia(Grafo *g) {
    for (Vertice *v1 = g->vertices; v1; v1 = v1->prox) {
        for (Vertice *v2 = v1->prox; v2; v2 = v2->prox) {
            if (v1->frequencia == v2->frequencia) {
                adicionarAresta(v1, v2);
                adicionarAresta(v2, v1);
            }
        }
    }
}
```

Figura 16 - conectarVerticesMesmaFrequencia

- listarVertices - lista na consola todas as antenas presentes no grafo com as respetivas frequências e coordenadas.

```c
void listarVertices(Grafo *g) {
    printf("Antenas no grafo:\n");
    for (Vertice *v = g->vertices; v; v = v->prox)
        printf("Frequência %c em (%d, %d)\n", v->frequencia, v->x, v->y);
}
```

Figura 17 - listarVertices

- dfs - executa procura em profundidade a partir de um vértice. Visita recursivamente todos os vértices ligados.

```c
void dfs(Vertice *inicio) {
    int visitado[100][100] = {{0}};
    printf("DFS:\n");
    dfs_rec(inicio, visitado);
}
```

Figura 18 - dfs

- caminhosEntre - encontra e imprime **todos os caminhos possíveis** entre dois vértices, usando recursividade.

```c
void caminhosEntreRec(Vertice *atual, Vertice *fim, int visitado[100][100], char *caminho, int
    if (atual == fim) {
        caminho[nivel] = '\0';
        printf("%s(%d,%d)\n", caminho, fim->x, fim->y);
        return;
    }

    visitado[atual->x][atual->y] = 1;
    int len = sprintf(&caminho[nivel], "(%d,%d)->", atual->x, atual->y);
    nivel += len;

    for (Aresta *a = atual->arestas; a; a = a->prox) {
        if (!visitado[a->destino->x][a->destino->y])
            caminhosEntreRec(a->destino, fim, visitado, caminho, nivel);
    }

    visitado[atual->x][atual->y] = 0;
}

void caminhosEntre(Vertice *inicio, Vertice *fim) {
    char caminho[1000];
    int visitado[100][100] = {{0}};
    printf("Caminhos entre (%d,%d) e (%d,%d):\n", inicio->x, inicio->y, fim->x, fim->y);
    caminhosEntreRec(inicio, fim, visitado, caminho, 0);
}
```

Figura 19 - caminhosEntre | caminhosEntreRec

- intersecoesFrequencias - verifica se há vértices de frequências diferentes que ocupam as mesmas coordenadas.

```c
void intersecoesFrequencias(Grafo *g, char f1, char f2) {
    printf("Interseções entre frequências %c e %c:\n", f1, f2);
    for (Vertice *v1 = g->vertices; v1; v1 = v1->prox) {
        if (v1->frequencia != f1) continue;
        for (Vertice *v2 = g->vertices; v2; v2 = v2->prox) {
            if (v2->frequencia == f2 &&
                v1->x == v2->x && v1->y == v2->y) {
                printf("Interseção em (%d, %d)\n", v1->x, v1->y);
            }
        }
    }
}
```

Figura 20 - intersecoesFrequencias

5. Análise e Discussão de Resultados

5.1 Fase 1

Na Fase 1, o programa conseguiu carregar corretamente as antenas a partir do ficheiro de texto e guardá-las numa lista ligada. Foi possível adicionar e remover antenas da lista sem erros.

O cálculo dos efeitos nefastos funcionou bem: os pontos nefastos foram identificados conforme as regras (antenas alinhadas e com uma ao dobro da distância da outra). A listagem no terminal mostrou as antenas e os efeitos de forma clara, permitindo verificar que a lógica estava correta.

5.2 Fase 2

Na Fase 2, o grafo foi construído com sucesso a partir do mesmo ficheiro. As antenas com a mesma frequência foram ligadas automaticamente com arestas.

A procura em profundidade (DFS) mostrou todos os vértices ligados, confirmando que as conexões estavam bem feitas.

Também foi possível encontrar todos os caminhos entre duas antenas, o que permite analisar como estão ligadas.

A função de interseção entre frequências diferentes funcionou, mostrando as coordenadas onde duas antenas (com frequências diferentes) ocupavam o mesmo local.

5.3 Pontos a melhorar

Falta de procura em largura (BFS) na Fase 2: esta funcionalidade está descrita no enunciado, mas não foi implementada.

6. Conclusão

O desenvolvimento deste projeto permitiu aplicar, de forma prática, conceitos fundamentais de estruturas de dados em C. Na Fase 1, foi implementado um sistema com listas ligadas para representar antenas e detetar efeitos nefastos com base em critérios de alinhamento e frequência. As funcionalidades principais foram concluídas com sucesso, incluindo a leitura de ficheiros, inserção, remoção e cálculo automático dos efeitos.

Na Fase 2, o uso de grafos trouxe uma abordagem mais avançada, permitindo representar relações entre antenas com maior flexibilidade. Foi possível construir o grafo a partir do ficheiro, realizar procura em profundidade, identificar caminhos entre antenas e detetar interseções de frequências.

Ambas as fases contribuíram para reforçar os conhecimentos em estruturas dinâmicas, modularização e resolução de problemas com algoritmos eficientes.

7. Referências

Informática, S. (s.d.). *Lista Ligadas C*. Obtido de seguranca-informatica: https://seguranca-informatica.pt/lista-ligada-c/

Progressivo, C. (s.d.). *cprogressivo.net*. Obtido de cprogressivo.net: https://www.cprogressivo.net/2013/10/Como-fazer-uma-lista-em-C.html

USP, P. F. (s.d.). *Listas encadeadas*. Obtido de ime.usp.br: https://www.ime.usp.br/~pf/algoritmos/aulas/lista.html