Licenciatura
de
Engenharia de Sistemas Informáticos

Pós-Laboral

UC: Estruturas de Dados Avançadas

David da Silva Pinheiro, Nº25444

Docente: Luis Gonzaga Martins Ferreira

Maio de 2025

Resumo

Este projeto foi desenvolvido para aplicar os conhecimentos adquiridos ao longo da disciplina de **Estruturas de Dados Avançadas (EDA)**, utilizando a linguagem de programação **C**. O objetivo foi implementar uma solução para representar e manipular antenas dispostas numa matriz, levando em consideração os efeitos nefastos provocados pela ressonância entre antenas de frequências iguais. O projeto foi dividido em duas fases:

- **Fase 1**: Implementação de uma lista ligada para representar as antenas e suas localizações, além da deteção de pontos de interferência.
- **Fase 2**: Aplicação de conceitos avançados de **teoria dos grafos** para interligar antenas com frequências iguais, utilizando operações de busca em profundidade e largura, bem como a identificação de caminhos entre antenas.

Este documento documenta as decisões de implementação, o código desenvolvido e a eficiência da solução para diferentes casos de teste.

Índice

Introdução

O projeto visa o desenvolvimento de uma solução para representar antenas numa cidade e analisar os efeitos nefastos causados pela ressonância entre antenas com frequências iguais. As antenas estão dispostas em uma matriz com localizações e frequências específicas. A solução foi estruturada em duas fases, inicialmente utilizando listas ligadas para gerenciar as antenas e, posteriormente, utilizando grafos para representar as interações entre antenas de frequências iguais.

A estrutura de listas ligadas foi escolhida para a **Fase 1** devido à sua capacidade de manipulação dinâmica de memória, facilitando a inserção e remoção de antenas.

Na **Fase 2**, a utilização de grafos permite representar as relações entre antenas com frequências iguais, facilitando operações de busca e análise de caminhos entre antenas.

Objetivos do Projeto

Os objetivos principais deste projeto foram:

- **Fase 1**:
  - Implementar uma estrutura de dados dinâmica (lista ligada) para armazenar as antenas e suas frequências.
  - Carregar os dados das antenas a partir de um ficheiro de texto.
  - Implementar operações como inserção, remoção e deteção de pontos de interferência.
- **Fase 2**:
  - Representar as antenas como grafos, com vértices correspondendo a antenas e arestas conectando antenas com a mesma frequência.
  - Implementar operações de busca em profundidade e largura.
  - Identificar caminhos e intersecções entre antenas de frequências diferentes.

Fase 1

Definição da Estrutura de Dados ED

Na Fase 1, foi implementada uma estrutura de dados para representar as antenas utilizando listas ligadas.

- Frequência (um caractere representando a frequência da antena).
- Coordenadas (as coordenadas da antena na matriz).
- Ponteiro para o próximo nó (permitindo a ligação dinâmica entre antenas).

Esta estrutura foi escolhida para otimizar a alocação de memória e facilitar a inserção e remoção de antenas, dado que a quantidade de antenas pode variar ao longo do tempo.

```
typedef struct Antena {
      char frequencia;          /* Frequencia da antena */
      int x;                          /* Coordenada X da antena */
      int y;                          /* Coordenada Y da antena */
      struct Antena* prox;      /* Apontador para a proxima antena na lista
} Antena;
```

Carregamento de Dados de Antenas

Os dados relativos às antenas foram carregados a partir de um ficheiro de texto, que contém uma matriz com as coordenadas e frequências das antenas. A estrutura de dados permite inserir dinamicamente cada antena no sistema, respeitando o formato do ficheiro.

Formato do ficheiro antenas.txt:

```
............
........0...
.....0......
.......0....
....0.......
......A.....
............
............
........A...
.........A..
............
............
```

Operações de Manipulação

1. **Inserção de uma nova antena**:

   A função insere uma nova antena na lista ligada, alocando memória dinamicamente e atribuindo as suas coordenadas e frequência.

Código InserirAntena

2. **Remoção de uma antena:**

   A função permite remover uma antena específica da lista ligada, liberando a memória associada a essa antena.

Código RemoverAntena

3. **Deteção de efeitos nefastos:**

   Baseado nas frequências das antenas, a função determina as localizações com efeito nefasto, representadas como #. Essas localizações são armazenadas em uma lista ligada.

Código EfeitoNefasto

4. **Listagem das antenas e efeitos nefastos:**

   A função exibe as antenas e os pontos de interferência na matriz.

Código ListarAntenas

Fase 2

Definição da Estrutura de Dados GR

Na Fase 2, a solução foi expandida para utilizar grafos para representar a relação entre antenas de frequências iguais. Cada antena é um vértice do grafo, e há uma aresta entre dois vértices se as antenas tiverem a mesma frequência.

Carregamento dos Dados e Criação do Grafo

A estrutura de grafo foi implementada para armazenar antenas e suas interações. O grafo é construído dinamicamente a partir dos dados do ficheiro, criando vértices para cada antena e conectando-os por arestas se as antenas tiverem a mesma frequência.

Operações de Manipulação do Grafo

1. **Procura em profundidade**

A função realiza uma busca em profundidade a partir de uma antena específica, listando as antenas alcançadas.

Código Profundidade

2. **Procura em largura**

A função realiza uma busca em largura, também a partir de uma antena específica, listando as antenas alcançadas.

Código Largura

3. **Identificação de caminhos entre duas antenas**

A função encontra todos os caminhos possíveis entre duas antenas, listando a sequência de arestas (ou antenas) que ligam ambas.

Código CaminhosentreAntenas

4. Intersecções de antenas com frequências distintas

Dadas duas frequências, a função encontra todas as intersecções de antenas de diferentes frequências e lista suas coordenadas.

Código ListarIntersecoes

Conclusão

O projeto foi desenvolvido com sucesso, utilizando as estruturas de dados adequadas para resolver o problema de gestão e análise de antenas. Na **Fase 1**, a implementação de listas ligadas permitiu uma eficiente manipulação das antenas e a deteção dos efeitos nefastos. Na **Fase 2**, o uso de grafos possibilitou a representação das relações entre antenas de frequências iguais, facilitando operações avançadas como busca em profundidade, busca em largura e identificação de caminhos.

A solução foi testada com diferentes dimensões e complexidades de matrizes, mostrando-se eficiente tanto em termos de tempo de execução quanto de uso de memória.

Código Fonte

Apresenta-se abaixo o código correspondente a cada uma das funções descritas neste relatório.

1. Função InserirAntena

```
int InserirAntena(ListaLigada* lista, char frequencia, int x, int y) {
    Antena* nova = (Antena*)malloc(sizeof(Antena));
    nova->frequencia = frequencia;
    nova->x = x;
    nova->y = y;
    nova->prox = lista->h;
    lista->h = nova;
    return 1;
}
```

2. Função RemoverAntena

```
int RemoverAntena(ListaLigada* lista, int x, int y) {
    Antena* atual = lista->h;
    Antena* anterior = NULL;
    while (atual) {
        if (atual->x == x && atual->y == y) {
            if (anterior) anterior->prox = atual->prox;
            else lista->h = atual->prox;
            free(atual);
            return 1;
        }

        anterior = atual;
        atual = atual->prox;
    }

    return 0;
}
```

3. Função EfeitoNefasto

```
ListaLigada* EfeitoNefasto(ListaLigada* lista) {
    ListaLigada* locais_nefastos = CriarLista();
    Antena* a = lista->h;
    while (a) {
        Antena* b = a->prox;
        while (b) {
            if (a->frequencia == b->frequencia) {
                int dx = b->x - a->x;
                int dy = b->y - a->y;
                if (dx != 0 || dy != 0) {
                    InserirAntena(locais_nefastos, '#', a->x - dx, a-
>y - dy);
                    InserirAntena(locais_nefastos, '#', b->x + dx, b-
>y + dy);
                }
            }
            b = b->prox;
        }
        a = a->prox;
    }
    return locais_nefastos;
}
```

4. Função ListarAntenas

```
int ListarAntenas(char matriz[12][12], ListaLigada* lista) {
    printf("\n------ Listar as Antenas ------\n");
    printf("+------------+----+----+\n");
    printf("| Frequencia |  X |  Y |\n");
    printf("+------------+----+----+\n");

    for (int i = 0; i < 12; i++) {
        for (int j = 0; j < 12; j++) {
            if (matriz[i][j] != '.' && matriz[i][j] != ' ' &&
matriz[i][j] != '#') {
                printf("|     %c      | %2d | %2d |\n", matriz[i][j],
j, i);

            }
        }
    }
    printf("+------------+----+----+\n");
    printf("\n------ Locais com Efeito Nefasto ------\n");
    printf("+----+----+\n");
    printf("|  X |  Y |\n");
    printf("+----+----+\n");
    for (int i = 0; i < 12; i++) {
        for (int j = 0; j < 12; j++) {
            if (matriz[i][j] == '#') {
                printf("| %2d | %2d |\n", j, i);

            }
        }
    }
    return 1;
}
```

5. Função Profundidade

```
int Profundidade(GR* grafo, int x, int y) {
    if (!grafo) return 0;

    int indiceInicio = EncontrarIndicePorCoordenadas(grafo, x, y);
    if (indiceInicio == -1) {
        return 0;
    }

    int* list = calloc(grafo->numVertices, sizeof(int));
    int* pilha = malloc(sizeof(int) * grafo->numVertices);
    int topo = 0;
    pilha[topo++] = indiceInicio;

    printf("\n------ Procurar Profundidade a partir de (%d,%d) ------
\n", x, y);

    while (topo > 0) {
        int atual = pilha[--topo];
        if (!list[atual]) {
            list[atual] = 1;
            printf("Encontrado: %c(%d,%d)\n",
                grafo->vertices[atual].antena->frequencia,
                grafo->vertices[atual].antena->x,
                grafo->vertices[atual].antena->y);

            Adjacente* adj = grafo->vertices[atual].adjacentes;
            while (adj) {
                if (!list[adj->indice]) {
                    pilha[topo++] = adj->indice;
                }
                adj = adj->prox;
            }
        }
    }

    free(pilha);
    free(list);
    return 1;
}
```

6. Função Largura

```
int Largura(GR* grafo) {
    if (!grafo) return 0;

    int x, y;
    printf("Introduza as coordenadas da antena para procura em
largura\nCoordenada X: ");
    scanf_s("%d", &x);

    printf("Coordenada Y: ");
    scanf_s("%d", &y);

    int indiceInicio = EncontrarIndicePorCoordenadas(grafo, x, y);
    if (indiceInicio == -1) {
        return 0;
    }

    int* list = calloc(grafo->numVertices, sizeof(int));
    int* fila = malloc(sizeof(int) * grafo->numVertices);
```

```
    int inicio = 0, fim = 0;

    fila[fim++] = indiceInicio;
    list[indiceInicio] = 1;

    printf("\n------ Procurar Largura a partir de (%d,%d) ------\n",
x, y);

    while (inicio < fim) {
        int atual = fila[inicio++];
        printf("Encontrado: %c(%d,%d)\n",
            grafo->vertices[atual].antena->frequencia,
            grafo->vertices[atual].antena->x,
            grafo->vertices[atual].antena->y);

        Adjacente* adj = grafo->vertices[atual].adjacentes;
        while (adj) {
            if (!list[adj->indice]) {
                fila[fim++] = adj->indice;
                list[adj->indice] = 1;
            }
            adj = adj->prox;
        }
    }

    free(fila);
    free(list);
    return 1;
}
```

7. Função CaminhosentreAntenas

```
int CaminhosEntreAntenas(GR* grafo, char nomeFicheiro[]) {
    char ant, ant2;
    Antena C1, C2;
    char matriz[12][12]; // assume tamanho definido globalmente ou
como macro

    LerFicheiroParaMatriz(nomeFicheiro, matriz);

    printf("Escolha a Primeira Antena: ");
    scanf_s(" %c", &ant);
    C1 = ListarAntenasPorTipo(ant, matriz);

    printf("Escolha a Segunda Antena: ");
    scanf_s(" %c", &ant2);
    C2 = ListarAntenasPorTipo(ant2, matriz);

    // Impede caminhos entre antenas de frequências diferentes
    if (ant != ant2) {
        return 0;
    }

    // Obtem os índices das antenas no grafo
    int indiceInicio = EncontrarIndicePorCoordenadas(grafo, C1.x,
C1.y);
    int indiceFim = EncontrarIndicePorCoordenadas(grafo, C2.x, C2.y);

    if (indiceInicio != -1 && indiceFim != -1) {
        ListarCaminhos(grafo, indiceInicio, indiceFim);
    }
    else {
        return 0;
```

```
    }

    return 1;
}
```

8. Função ListarIntersecoes

```
int ListarIntersecoes(GR* grafo) {
    if (!grafo) return 0;

    char freqA, freqB;
    printf("Frequencia A: ");
    scanf_s(" %c", &freqA, 1);

    printf("Frequencia B: ");
    scanf_s(" %c", &freqB, 1);

    printf("\n------ Intersecoes entre frequencias %c e %c ------\n",
freqA, freqB);

    for (int i = 0; i < grafo->numVertices; i++) {
        if (grafo->vertices[i].antena->frequencia == freqA) {
            for (int j = 0; j < grafo->numVertices; j++) {
                if (grafo->vertices[j].antena->frequencia == freqB) {
                    printf("Intersecao: %c(%d,%d) <-> %c(%d,%d)\n",
                        grafo->vertices[i].antena->frequencia,
                        grafo->vertices[i].antena->x,
                        grafo->vertices[i].antena->y,
                        grafo->vertices[j].antena->frequencia,
                        grafo->vertices[j].antena->x,
                        grafo->vertices[j].antena->y);

                }
            }
        }
    }

    return 1;
}
```

O Código completo encontra-se no GitHub:

https://github.com/DavidPinheiro55/Projeto_EDA_Fase2_25444.git