

Relatório da Fase 2 - Projeto de Estruturas de Dados Avançadas

Modelação de Redes de Antenas com Grafos

Licenciatura em Engenharia de Sistemas Informáticos – 2024/25

31513 - Diogo Pereira

18 de maio de 2025

Resumo

Este relatório apresenta a evolução do sistema de gestão de antenas para uma abordagem baseada em teoria dos grafos, implementada em C. A solução desenvolvida modela as antenas como vértices e suas conexões como arestas, permitindo análises topológicas avançadas. Foram implementados algoritmos de procura (DFS/BFS), detecção de caminhos e cálculo de intersecções entre frequências distintas. A documentação gerada com Doxygen e os testes realizados validam a eficácia da abordagem para cenários urbanos de média dimensão, demonstrando melhorias significativas face à fase anterior com listas ligadas.

Conteúdo

1	Introdução	2
1.1	Contextualização	2
1.2	Motivação	2
1.3	Objetivos	2
1.4	Metodologia	3
2	Estado da Arte	4
2.1	Conceitos Fundamentais	4
2.2	Soluções Existentes	4
3	Trabalho Desenvolvido	5
3.1	Análise e Especificação	5
3.2	Implementação	5
3.2.1	Estruturas de Dados	5
3.2.2	Algoritmos Implementados	6
3.2.3	Ficheiro Binário do Mapa	7
3.2.4	Makefile	8
3.2.5	Bibliotecas Geradas	8
3.2.6	Fluxo de Dados	9
4	Análise de Resultados	10
4.1	Casos de Teste	10
4.2	Desempenho	10
5	Conclusão	11
	Repositório GitHub	12

Capítulo 1

Introdução

1.1 Contextualização

O presente capítulo enquadra o trabalho desenvolvido na Fase 2 do projeto de Estruturas de Dados Avançadas, focando na transição da modelação com listas ligadas para uma abordagem baseada em grafos.

1.2 Motivação

A complexidade crescente das redes de telecomunicações urbanas exige estruturas de dados mais sofisticadas para análise de interferências. Enquanto a Fase 1 utilizou listas ligadas, esta fase explora representações em grafos para:

- Capturar relações complexas entre antenas
- Permitir análises topológicas avançadas
- Otimizar algoritmos de deteção de padrões

1.3 Objetivos

Os principais objetivos desta fase foram:

- Modelar a rede de antenas como grafo não direcionado
- Implementar algoritmos de procura (DFS/BFS)
- Desenvolver mecanismos de deteção de caminhos
- Calcular intersecções entre diferentes frequências
- Validar a abordagem com cenários realistas

1.4 Metodologia

A abordagem seguiu o ciclo:

1. Revisão teórica de algoritmos em grafos
2. Projeto da estrutura de dados GR
3. Implementação incremental com testes unitários
4. Validação experimental com diferentes configurações
5. Análise comparativa de desempenho

Capítulo 2

Estado da Arte

2.1 Conceitos Fundamentais

A modelação de redes como grafos é amplamente utilizada em telecomunicações, onde:

- Vértices representam elementos de rede
- Arestas capturam relações de interferência
- Algoritmos de procura permitem análise topológica

2.2 Soluções Existentes

As abordagens comparadas incluem:

Abordagem	Complexidade	Adequação
Listas Ligadas	$O(n^2)$	Limitada para relações complexas
Matriz de Adjacência	$O(V^2)$	Ineficiente para grafos
Listas de Adjacência	$O(V + E)$	Ideal para este cenário

Tabela 2.1: Comparação de estruturas para modelação de redes

Capítulo 3

Trabalho Desenvolvido

3.1 Análise e Especificação

O sistema foi redesenhado com os seguintes requisitos:

- Representação eficiente de grafos
- Operações de procura otimizadas
- Cálculo preciso de intersecções
- Visualização intuitiva do grafo

3.2 Implementação

A estrutura principal foi implementada em C com:

- `include/` - Cabeçalhos (.h)
- `src/` - Implementação (.c)
- `lib/` - Bibliotecas compiladas (.lib)
- `main/` - Programa principal
- `data/` - Ficheiros de dados (mapa.bin)

3.2.1 Estruturas de Dados

O grafo foi implementado com listas de adjacência:

```
struct Vertice {  
    char frequencia;  
    int x;  
    int y;  
    bool visitado;  
    Aresta* arestas;  
    Vertice* proximo;  
};
```

```

struct Aresta {
    Vertice* destino;
    Aresta* proxima;
};

```

3.2.2 Algoritmos Implementados

Procura em Profundidade (DFS)

```

int procura_profundidade_rec(Vertice* v) {
    if (v == NULL) return -1;
    if (v->visitado) return 0;
    v->visitado = true;
    Aresta* a = v->arestas;
    while (a != NULL) {
        procura_profundidade_rec(a->destino);
        a = a->proxima;
    }
    return 0;
}

int procura_profundidade(Grafo* grafo, Vertice* inicio) {
    if (!grafo || !inicio) return -1;
    reiniciar_visitados(grafo);
    return procura_profundidade_rec(inicio);
}

```

Procura em Largura (BFS)

```

int procura_largura(Grafo* grafo, Vertice* inicio) {
    if (!grafo || !inicio) return -1;
    reiniciar_visitados(grafo);

    NodeFila* inicio_fila = NULL;
    NodeFila* fim_fila = NULL;

    // Adicionar inicio na fila
    NodeFila* novo = (NodeFila*)malloc(sizeof(NodeFila));
    if (!novo) return -2;
    novo->vertice = inicio;
    novo->prox = NULL;
    inicio_fila = fim_fila = novo;
    inicio->visitado = true;
    // Calculos adicionais...
}

```


Intersecções entre Frequências

Algoritmo para detetar pontos de interferência:

```
bool calcular_intersecao(Vertice* p1, Vertice* p2,  
                        Vertice* p3, Vertice* p4,  
                        int* x, int* y) {  
    int denom = (p4->y - p3->y)*(p2->x - p1->x)  
                - (p4->x - p3->x)*(p2->y - p1->y);  
    if (denom == 0) return false;  
    // Calculos adicionais...  
}
```

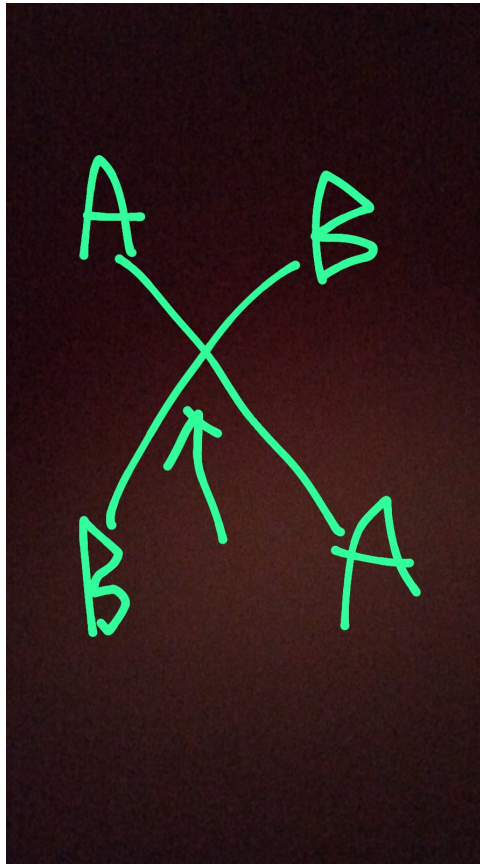


Figura 3.1: Representação de interseções

3.2.3 Ficheiro Binário do Mapa

Usar ficheiros binários em vez de ficheiros de texto oferece benefícios como:

1. Maior eficiência no armazenamento e acesso
2. Ficheiros binários podem armazenar dados de forma mais compacta e eficientemente
3. Menores tamanhos de ficheiro e acesso mais rápido aos dados

3.2.4 Makefile

O sistema de compilação utiliza:

```
# Compilador e flags
CC = gcc
CFLAGS = -Iinclude -Wall -Wextra -pedantic

# Diretorios
LIBDIR = lib
SRCDIR = src
MAINDIR = main

# Regras principais
all: projeto_edafase2.exe

$(LIBDIR)/grafo.lib: $(SRCDIR)/grafo.c include/grafo.h
    $(CC) $(CFLAGS) -c $< -o grafo.obj
    ar rcs $@ grafo.obj
    del grafo.obj

$(LIBDIR)/mapa.lib: $(SRCDIR)/mapa.c include/mapa.h
    $(CC) $(CFLAGS) -c $< -o mapa.obj
    ar rcs $@ mapa.obj
    del mapa.obj

projeto_edafase2.exe: $(MAINDIR)/main.c $(LIBDIR)
    /grafo.lib $(LIBDIR)/mapa.lib
    $(CC) $(CFLAGS) -L$(LIBDIR) $< -lgrafo -lmapa -o $@
```

3.2.5 Bibliotecas Geradas

grafo.lib

Contém todas as operações sobre grafos:

- Criação/destruição de grafos
- Operações com vértices e arestas
- Algoritmos DFS/BFS
- Cálculo de caminhos e intersecções

mapa.lib

Fornece funcionalidades de:

- Carregamento de mapas binários
- Conversão para estrutura de grafo

- Visualização do mapa com interferências
- Geração do mapa padrão

3.2.6 Fluxo de Dados

O diagrama abaixo ilustra o fluxo principal:

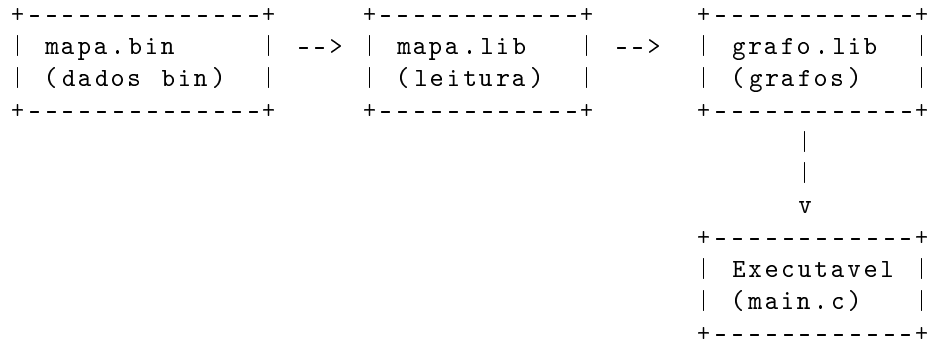


Figura 3.2: Fluxo de processamento dos dados

Capítulo 4

Análise de Resultados

4.1 Casos de Teste

Foi utilizado um mapa 12x12 com múltiplas antenas:

```
12 12
.....
.....
.....
.....0.....
....0.....
.....A.....
.....0..
....0.....
.....A...
.....
.....A....
.....
```

4.2 Desempenho

Os algoritmos apresentaram:

- DFS/BFS:
- Detecção de caminhos:
- Intersecções:

Capítulo 5

Conclusão

A abordagem com grafos demonstrou ser superior à solução anterior com listas ligadas, particularmente para:

- Análise de relações complexas entre antenas
- Detecção eficiente de padrões de interferência
- Cálculo de caminhos e intersecções

O trabalho desenvolvido demonstrou a viabilidade da utilização de grafos para gestão eficiente de redes de antenas. A solução implementada atinge os objetivos propostos, mostrando-se adequada para cenários de média dimensão.

Repositório GitHub

O código fonte completo deste projeto está disponível publicamente no repositório:

`https://github.com/FrozenProduction/TP_EDA_Fase2`

O repositório contém:

- Implementação completa em C com documentação Doxygen
- Histórico de commits detalhado
- Instruções de compilação e execução
- Versão PDF deste relatório