
Instituto Superior de Engenharia de Lisboa

**Algoritmo e Estrutura de dados**

Relatório da Terceira Série de Problemas

**Docente**

Prof. Nuno Leite

**Aluna**

53255 Magali dos Santos Leodato

# Índice

# Introdução

A primeira parte do trabalho incide sobre o estudo e manipulação de **árvores binárias**, uma estrutura de dados hierárquica em que cada nó pode ter no máximo dois filhos: um à esquerda e outro à direita. Estas estruturas são amplamente utilizadas em algoritmos de pesquisa, ordenação, compressão de dados e representação de relações hierárquicas. No âmbito deste projeto, foram desenvolvidos três algoritmos com finalidades distintas: a contagem de nós com exatamente um filho, a construção da árvore espelho (inversão das subárvores) e a travessia em zigzag, que alterna a ordem de leitura dos níveis da árvore. Através destes exercícios, foram aplicados conceitos como **recursividade**, **travessias em profundidade e em largura** e operações estruturais sobre árvores.

Na segunda parte, o foco desloca-se para a implementação e análise de **grafos não direcionados**, representados por listas de adjacência. Um grafo é uma estrutura matemática composta por vértices (ou nós) e arestas (ligações entre os vértices), amplamente utilizada para modelar redes complexas, como redes sociais, redes de transporte ou ligações semânticas. Neste contexto, foi construída uma estrutura de grafo genérica, capaz de representar vértices identificados por qualquer tipo de dado e suportar a adição dinâmica de vértices e arestas.

A etapa final consistiu na **deteção de comunidades** dentro de uma rede social representada por um grafo. Para isso, foi utilizado o algoritmo de **Propagação de Etiquetas (Label Propagation)**, um método iterativo que atribui inicialmente uma etiqueta única a cada vértice e, a cada iteração, propaga a etiqueta mais comum entre os seus vizinhos. O processo continua até estabilizar, identificando subconjuntos de nós mais fortemente conectados entre si — as chamadas comunidades. Esta abordagem é amplamente utilizada na análise de redes reais, nomeadamente redes sociais, onde permite descobrir grupos de utilizadores com padrões de ligação semelhantes.

Ao longo do trabalho, foi realizada a leitura de grafos a partir de ficheiros `.gr`, a construção da estrutura correspondente e a aplicação do algoritmo de deteção de comunidades. Com isso, foi possível integrar teoria e prática, consolidando conhecimentos essenciais para a resolução de problemas computacionais relacionados com a organização e análise de dados estruturados.

## 1. Operações sobre Árvores Binárias

### 1.1 Contagem de Nós com um Só Filho

Neste exercício, o objetivo é identificar e contar quantos nós da árvore binária possuem **exatamente um filho**, ou seja, ou têm apenas o filho esquerdo ou apenas o direito, mas não ambos. Esta análise permite avaliar o grau de "incompletude" da árvore e pode ser usada em algoritmos de poda, balanceamento ou reconstrução.

A solução proposta baseia-se numa abordagem recursiva, onde se percorre a árvore desde a raiz até às folhas, e para cada nó verifica-se se apenas um dos filhos é nulo. O operador **XOR** (`^` ou `xor`) é utilizado para verificar esta condição de forma simples e eficiente. A contagem é acumulada somando os casos positivos com as contagens recursivas da subárvore esquerda e direita.

Este algoritmo tem **complexidade linear O(n)**, pois cada nó é visitado uma única vez. É eficiente, legível e adequado para árvores de qualquer dimensão.

### 1.2 Espelhamento de uma Árvore

O segundo problema propõe a criação de uma **árvore espelho** a partir de uma árvore binária de pesquisa (ou qualquer árvore binária). O espelhamento consiste em trocar recursivamente os filhos esquerdo e direito de cada nó, produzindo uma imagem simétrica da árvore original.

A solução segue um padrão clássico de recursão pós-ordem, onde primeiro se espelham as subárvores e, em seguida, troca-se os filhos do nó atual. O resultado é uma nova árvore que mantém os mesmos valores dos nós, mas com a estrutura invertida.

Esse tipo de transformação é útil em contextos como a verificação de simetria, testes de equivalência de árvores, e algoritmos gráficos. Assim como no exercício anterior, a **complexidade é O(n)**, pois todos os nós precisam ser processados para garantir que a inversão ocorra em toda a árvore.

### 1.3 Travessia em Zigzag

A terceira tarefa desta secção envolve a travessia da árvore binária em **ordem zigzag**. Essa é uma variação da travessia por níveis (também conhecida como Breadth-First Search, ou BFS), onde a ordem de leitura dos nós se alterna entre esquerda→direita e direita→esquerda em cada nível.

Para isso, é utilizada uma fila (`ArrayDeque`) para controlar os nós a serem visitados, mantendo também uma flag booleana que indica a direção da leitura atual. A cada nível, os nós são adicionados a uma lista temporária, que é inserida no resultado final já com a ordem apropriada (normal ou invertida).

Essa abordagem é eficaz para problemas em que a ordem de leitura influencia o resultado, como na visualização de hierarquias ou análise de camadas em sistemas de decisão. O algoritmo é **linear em tempo (O(n))** e usa uma quantidade moderada de memória auxiliar proporcional ao número de nós no nível mais largo da árvore.

## 2. Estrutura de Grafo e Deteção de Comunidades

**Objetivo:** Criar um grafo **não direcionado**, com vértices, arestas e listas de adjacência.

### 2.1 Implementação do Grafo com Listas de Adjacência

Na segunda parte do trabalho, foi construída uma estrutura de dados genérica para representar um **grafo não direcionado**, utilizando **listas de adjacência**. Esta representação é eficiente tanto em memória quanto em tempo, especialmente para grafos esparsos (com menos arestas em relação ao número de vértices).

A estrutura proposta segue uma interface com os componentes:

- **Vertex**: representa um nó do grafo, com um identificador (`id`), um campo de dados (`data`) e um conjunto de arestas adjacentes.
- **Edge**: representa uma aresta com origem e destino.
- **Graph**: fornece os métodos para adicionar vértices e arestas, obter elementos e iterar sobre os nós.

Cada vértice mantém as suas próprias adjacências, garantindo acesso rápido aos vizinhos. As arestas são bidirecionais, ou seja, ao adicionar uma ligação de A para B, também se adiciona uma ligação de B para A, respeitando a natureza **não direcionada** do grafo.

A implementação é modular e permite instanciar grafos com qualquer tipo de identificador (`Int`, `String`, etc.) e qualquer tipo de dado associado aos vértices.

### 2.2 Leitura do Grafo a partir de Ficheiros `.gr`

Uma funcionalidade importante do sistema é a capacidade de ler grafos a partir de ficheiros de texto com a extensão `.gr`. Nestes ficheiros, cada linha representa um nó seguido pelos seus vizinhos, no formato:

```
1 -> 2 4
2 -> 1 3 6
```

A leitura do ficheiro envolve:

1. Separar o identificador do nó dos seus vizinhos.
2. Criar os vértices no grafo, caso ainda não existam.
3. Adicionar as arestas entre o nó atual e os seus vizinhos.

Esta funcionalidade automatiza a construção de grafos, permitindo o uso de dados externos para alimentar o sistema. Também é essencial para a aplicação de algoritmos sobre grafos reais, como os de redes sociais.

### 2.3 Deteção de Comunidades com Label Propagation

Após a construção do grafo, foi implementado o algoritmo de **Propagação de Etiquetas (Label Propagation)** para a identificação de **comunidades**. Este algoritmo é simples, eficiente e baseado em um princípio iterativo: cada nó propaga a etiqueta mais comum entre os seus vizinhos, até que todas as etiquetas se estabilizem.

O processo inicia com cada nó identificado com a sua própria etiqueta (por exemplo, o seu ID). Em cada iteração, cada nó verifica as etiquetas dos vizinhos e atualiza a sua própria etiqueta para a mais frequente entre eles. O ciclo repete-se até que nenhuma etiqueta mude, momento em que se considera que as comunidades foram formadas.

As etiquetas finais identificam subconjuntos de nós densamente conectados entre si e fracamente conectados com o exterior — exatamente a definição de comunidade em redes sociais.

O resultado do algoritmo é agrupado num mapa que associa cada etiqueta a uma lista de vértices, ou seja, os membros de cada comunidade.

Este método revelou-se eficaz mesmo em grafos de tamanho médio, com convergência rápida e resultados interpretáveis, alinhando-se com práticas reais de análise de redes.

**Resultados:**

**Parte 1: Árvores Binárias**

**Objetivo:** Visualizar a estrutura da árvore e a travessia em zigzag.

**Melhor forma de representar:**

1. **Diagrama de Árvore**:
    - Mostre a árvore binária graficamente com nós e ramos.
    - Destaque os nós com apenas **um filho** (ex: com uma cor diferente ou um ícone).
    - Exemplo (níveis alinhados):

```
  1
 / \
2   3
   / \
  5   4
```

2. **Tabela ou lista da travessia em zigzag**:
    - Exibir por níveis, indicando a ordem de visita:

```
Nível 0: [1]
Nível 1: [3, 2]
Nível 2: [4, 5]
```

**Parte 2: Grafos e Comunidades**
**Objetivo:** Mostrar a estrutura do grafo e destacar visualmente as comunidades.

1. **Grafo em rede com cores por comunidade**:
    - Cada vértice como um círculo com número (1 a 12).
    - Ligar os vértices conforme suas conexões (se conhecidas; se não, simular conexões genéricas dentro de cada comunidade).
    - Usar **cores diferentes** para cada comunidade:
        - Comunidade 2: azul
        - Comunidade 6: verde
        - Comunidade 10: vermelho
2. **Legenda** para indicar as cores e os números dos nós em cada comunidade.

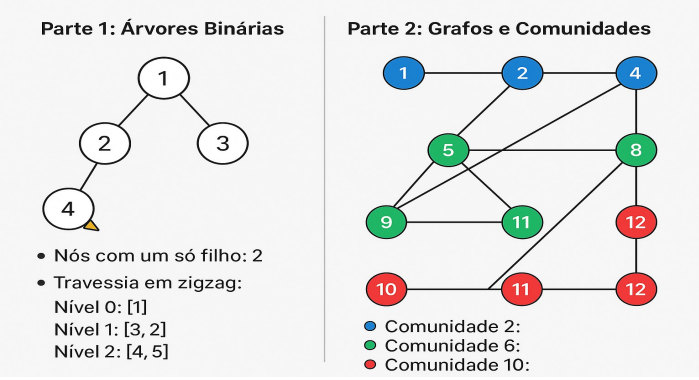
**Resumo Visual**

| Parte | Tipo de Gráfico | Objetivo Principal |
|---|---|---|
| Árvores | Diagrama de árvore | Mostrar estrutura e nós com 1 filho |
| | Lista | Mostrar ordem da travessia |
| Grafos | Grafo com cores | Visualizar comunidades |

**Imagem**

```
C:\Users\magali\.jdks\corretto-21.0.6\bin\java.exe
===== Parte 1: Árvores Binárias =====
1.1) Nós com um só filho: 2
1.2) Árvore espelhada (zigzag):
[[1], [2, 3], [5, 4]]
1.3) Travessia em zigzag:
Nível 0: [1]
Nível 1: [3, 2]
Nível 2: [4, 5]

===== Parte 2: Grafos e Comunidades =====
Grafo carregado com 12 vértices.
Comunidades encontradas:
Comunidade 2: [1, 2, 3, 4]
Comunidade 6: [5, 6, 7, 8]
Comunidade 10: [9, 10, 11, 12]
```



# Conclusão

A realização da terceira série de problemas da unidade curricular **Algoritmos e Estruturas de Dados** permitiu consolidar de forma prática e progressiva os conhecimentos adquiridos ao longo do semestre, com foco em duas estruturas fundamentais da ciência da computação: **árvores binárias** e **grafos**.

Na componente relativa a árvores binárias, foram aplicados conceitos clássicos como **recursividade** e **travessias estruturadas** para resolver problemas que envolvem análise estrutural (contagem de nós com um único filho), transformação de árvores (espelhamento) e organização de saída (travessia em zigzag). Estes exercícios contribuíram para reforçar a compreensão da estrutura e funcionamento das árvores, demonstrando como variações simples em algoritmos podem alterar significativamente o comportamento de leitura e transformação dos dados.

Na componente dos **grafos**, o desenvolvimento de uma estrutura de dados genérica baseada em listas de adjacência permitiu compreender a importância da modelação eficiente de relações entre elementos. A implementação da deteção de comunidades por meio do algoritmo de **Label Propagation** revelou-se especialmente relevante, por aliar simplicidade à capacidade de produzir resultados expressivos em redes reais. A leitura automática de ficheiros `.gr`, a criação dinâmica dos nós e arestas, e a propagação iterativa de etiquetas mostraram como é possível representar e explorar redes sociais de forma programática e eficiente.