

# Trilha de Estrutura de Dados

*Módulo 06: Estrutura de Dados  
Avançados*

## Instruções para a melhor prática de Estudo

- 1. Leia atentamente todo o conteúdo:** Antes de iniciar qualquer atividade, faça uma leitura detalhada do material fornecido na trilha, compreendendo os conceitos e os exemplos apresentados.
- 2. Não se limite ao material da trilha:** Utilize o material da trilha como base, mas busque outros materiais de apoio, como livros, artigos acadêmicos, vídeos, e blogs especializados. Isso enriquecerá o entendimento sobre o tema.
- 3. Explore a literatura:** Consulte livros e publicações reconhecidas na área, buscando expandir seu conhecimento além do que foi apresentado. A literatura acadêmica oferece uma base sólida para a compreensão de temas complexos.
- 4. Realize todas as atividades propostas:** Conclua cada uma das atividades práticas e teóricas, garantindo que você esteja aplicando o conhecimento adquirido de maneira ativa.
- 5. Evite o uso de Inteligência Artificial para resolução de atividades:** Utilize suas próprias habilidades e conhecimentos para resolver os exercícios. O aprendizado vem do esforço e da prática.
- 6. Participe de debates:** Discuta os conteúdos estudados com professores, colegas e profissionais da área. O debate enriquece o entendimento e permite a troca de diferentes pontos de vista.
- 7. Pratique regularmente:** Não deixe as atividades para a última hora. Pratique diariamente e revise o conteúdo com frequência para consolidar o aprendizado.
- 8. Peça feedback:** Solicite o retorno dos professores sobre suas atividades e participe de discussões sobre os erros e acertos, utilizando o feedback para aprimorar suas habilidades.

Essas instruções são fundamentais para garantir um aprendizado profundo e eficaz ao longo das trilhas.

## Estruturas de Dados Avançadas

### 1. Conjuntos Disjuntos (Union-Find)

**Conceito:** Conjuntos disjuntos, também chamados de **Union-Find**, são uma estrutura de dados que permite gerenciar uma coleção de conjuntos não sobrepostos (disjuntos) e realizar operações eficientes para determinar se dois elementos pertencem ao mesmo conjunto.

As duas operações principais são:

- **Union (União):** Combina dois conjuntos em um único conjunto.
- **Find (Busca):** Determina a qual conjunto um elemento pertence.

#### Exemplo de Aplicação:

Conjuntos disjuntos são amplamente utilizados em algoritmos de grafos, como em **algoritmos de Kruskal** para encontrar a árvore geradora mínima de um grafo.

#### Estrutura:

Cada conjunto é representado como uma árvore, onde cada elemento aponta para um "pai". O elemento raiz de cada árvore é o representante do conjunto.

- **Find:** A operação de busca segue os ponteiros para os pais até encontrar a raiz.
- **Union:** Combina dois conjuntos unindo as árvores de forma eficiente (usando a técnica de "union by rank" ou "union by size").

#### Otimizações:

- **Union by Rank:** Sempre anexa a árvore de menor profundidade à árvore de maior profundidade.
- **Path Compression:** Durante uma operação de busca, todos os nós visitados têm seus pais diretamente ligados à raiz, o que acelera futuras operações de busca.

#### Exemplo em Pseudocódigo:

 No link vemos o exemplo completo referente a: [Union-Find - Estrutura de Dados](#)

---

### 2. Árvores B e B+

**Conceito:** As **árvores B** e **B+** são tipos de árvores balanceadas amplamente utilizadas em sistemas de banco de dados e sistemas de arquivos. Essas árvores foram projetadas para funcionar bem com armazenamento secundário (discos) e podem manter seus nós平衡eados automaticamente, permitindo que as operações de busca, inserção e remoção sejam eficientes.

#### Árvore B:

- Uma **árvore B** é uma árvore de busca balanceada, onde cada nó pode ter múltiplos filhos e múltiplos valores.
- A árvore B mantém as propriedades de ordenação, com todos os valores à esquerda de um nó menores que o valor do nó, e todos à direita maiores.
- Cada nó pode ter até um número máximo de chaves definido, e, quando esse número é excedido, o nó é dividido.
- A altura da árvore é mantida pequena, o que garante eficiência mesmo para grandes volumes de dados.

### Árvore B+:

- A **árvore B+** é uma variação da árvore B, com a diferença de que as chaves e valores só são armazenados nas folhas, enquanto os nós intermediários servem apenas como índices.
- As folhas estão conectadas por meio de uma lista encadeada, o que facilita a busca sequencial.

### Exemplo de Estrutura de Árvore B+:

```
Nó Interno: [ 50 | 100 ]
      /   |   \
[ 10, 20, 30 ] → [ 60, 70 ] → [ 110, 120, 130 ] (Nós folha)
```

### Aplicações:

- As árvores B e B+ são amplamente utilizadas em **sistemas de banco de dados** para armazenar índices, já que elas permitem a realização de operações de inserção, busca e remoção de maneira eficiente, mesmo com grandes volumes de dados.
- O **sistema de arquivos NTFS** utiliza árvores B+ para organizar e gerenciar arquivos no disco.

### Operações:

- **Busca:** A busca em uma árvore B ou B+ segue a mesma lógica de uma árvore de busca binária, mas verifica vários filhos em vez de apenas dois.
- **Inserção:** Quando um nó excede a capacidade de armazenamento de chaves, ele é dividido e a chave do meio é promovida ao nó pai.
- **Remoção:** Quando um nó perde chaves abaixo da capacidade mínima, ele pode ser fundido com um nó irmão ou "pegar" uma chave emprestada.

### Nota

As estruturas de dados avançadas, como Conjuntos Disjuntos e Árvores B/B+, desempenham um papel fundamental na otimização de operações em algoritmos de grafos e em sistemas de banco de dados. A implementação dessas estruturas é essencial para

lidar com grandes volumes de dados e operações eficientes, especialmente quando se trata de armazenamento e organização de informações em tempo real.

---

## **Lista de Exercícios de Fixação**

### **1. Implementação de Union-Find:**

- Implemente uma estrutura de conjuntos disjuntos utilizando path compression e union by rank.
- Teste a estrutura resolvendo o problema de identificar componentes conectados em um grafo não direcionado.

### **2. Aplicação de Union-Find em Algoritmos de Grafos:**

- Utilizando sua implementação de Union-Find, implemente o **Algoritmo de Kruskal** para encontrar a árvore geradora mínima de um grafo ponderado.
- Dado um conjunto de arestas e vértices, determine se existe um ciclo no grafo.

### **3. Árvore B:**

- Implemente uma árvore B com um grau mínimo de 3 e insira os seguintes valores: 10, 20, 5, 6, 12, 30, 7, 17. Mostre a estrutura da árvore após cada inserção.
- Adicione a funcionalidade de remoção e demonstre a remoção dos valores 6 e 17.

### **4. Árvore B+:**

- Implemente uma árvore B+ com grau mínimo de 2 e insira os valores: 15, 5, 25, 10, 20, 30, 35.
- Mostre a estrutura da árvore após cada inserção, destacando a organização dos nós internos e folhas.

### **5. Análise de Desempenho:**

- Compare o desempenho de inserção e busca entre uma árvore B e uma árvore AVL com o mesmo conjunto de dados. Qual delas oferece melhor performance quando o conjunto de dados é muito grande?
- Com base no comportamento da árvore B, explique por que ela é preferida para sistemas de banco de dados em vez de árvores AVL ou Red-Black.

### **6. Aplicação de Árvores B em Banco de Dados:**

- Explique como uma árvore B pode ser usada para implementar um índice de banco de dados.
- Dado um cenário de banco de dados com milhões de registros, simule a inserção e busca de registros usando uma árvore B e explique os benefícios em termos de eficiência.