Algoritmos e Estruturas de Dados II

Capítulo da Aula 10: Árvores B e B+

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br
CEFET-MG - Campus Timóteo

Fevereiro de 2026

0.1 1. O Abismo da Memória

Por que não usamos AVL ou Árvore Rubro-Negra para bancos de dados em disco (como PostgreSQL ou MySQL)?

- **RAM:** Acesso em nanossegundos (10^{-9} s).
- **Disco (SSD/HDD):** Acesso em milissegundos (10^{-3} s). É **1.000.000x mais lento**.

Se usarmos uma árvore binária, buscar um registro em 1 milhão de itens requer altura $\log_2(10^6) \approx 20$. Isso significa 20 leituras de disco aleatórias. Se cada leitura leva 10ms, a busca leva 0.2 segundos. Inaceitável para bancos de dados de alta performance.

Precisamos de uma árvore **mais baixa e mais gorda**, que aproveite o fato de que, quando lemos do disco, lemos um bloco inteiro (ex: 4KB ou 8KB) de uma vez.

0.2 2. Árvore B (B-Tree)

Inventada por Bayer e McCreight (1972), a Árvore B generaliza a árvore binária. Em vez de 2 filhos, um nó pode ter **milhares** de filhos.

0.2.1 Propriedades da Árvore B de ordem M

1. Cada nó contém no máximo $M - 1$ chaves. 2. Cada nó (exceto raiz e folhas) tem pelo menos $\lceil M/2 \rceil$ filhos. 3. **Todas as folhas estão no mesmo nível.** A árvore é perfeitamente balanceada por definição. 4. As chaves dentro de um nó estão ordenadas.

0.2.2 A Vantagem da Altura

Se $M = 1000$ (ordem mil), um nó raiz com 1000 filhos, e cada filho com 1000 netos, gera:

- Nível 1: 1.000 itens.
- Nível 2: 1.000.000 itens.
- Nível 3: 1.000.000.000 (1 bilhão) de itens.

Com apenas **3 acessos a disco**, encontramos qualquer chave em 1 bilhão.

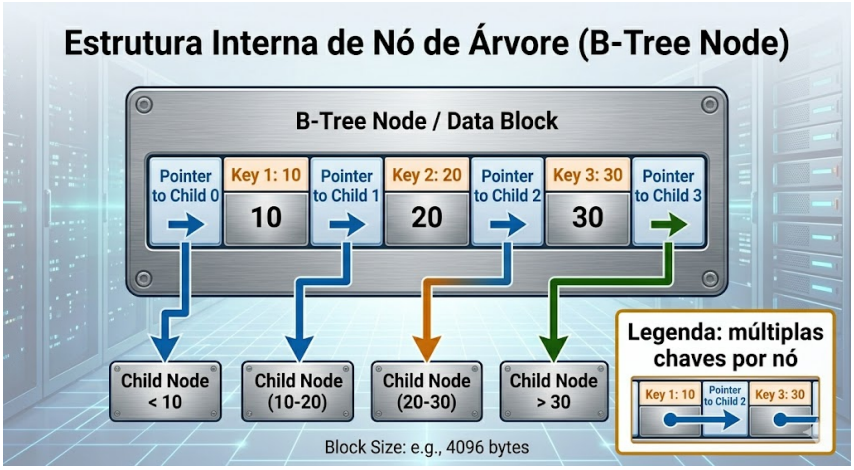

Figure 1: Nó de uma árvore B: várias chaves ordenadas e vários ponteiros para filhos.

0.3 3. Lógica de Inserção (Split)

Diferente de árvores binárias que crescem para baixo (adicionando folhas), a Árvore B **cresce para cima**.

1. Busque a folha onde a chave deve entrar. 2. Se a folha tiver espaço (menos de $M-1$ chaves), insira ordenado. 3. Se a folha estiver **cheia** (Overflow):

- Divida o nó em dois.
- A chave do meio sobe para o pai ("promovida").
- Se o pai encher, divida o pai... (pode propagar até a raiz).
- Se a raiz dividir, cria-se uma nova raiz, aumentando a altura da árvore em 1.

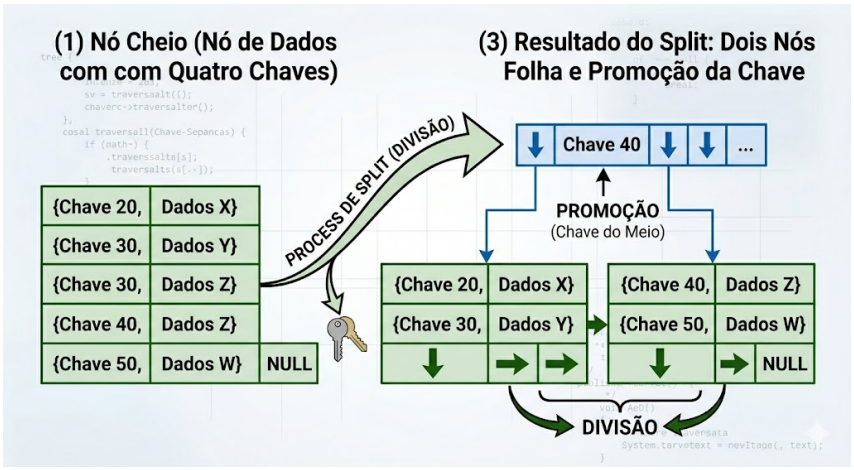

Figure 2: Overflow: dividir nó ao meio; chave do meio sobe para o pai (crescimento para cima).

0.4 4. Árvore B+ (B-Plus Tree)

É a variação usada na prática (sistemas de arquivos NTFS, ext4, bancos SQL).

0.4.1 Diferenças da B-Tree Clássica

1. **Dados apenas nas Folhas:** Os nós internos guardam apenas cópias das chaves para guiar a busca (índices). Os registros reais (ou ponteiros para o arquivo de dados) ficam exclusivamente nos nós folha. 2. **Folhas Encadeadas:** Todas as folhas são ligadas como uma Lista Encadeada.

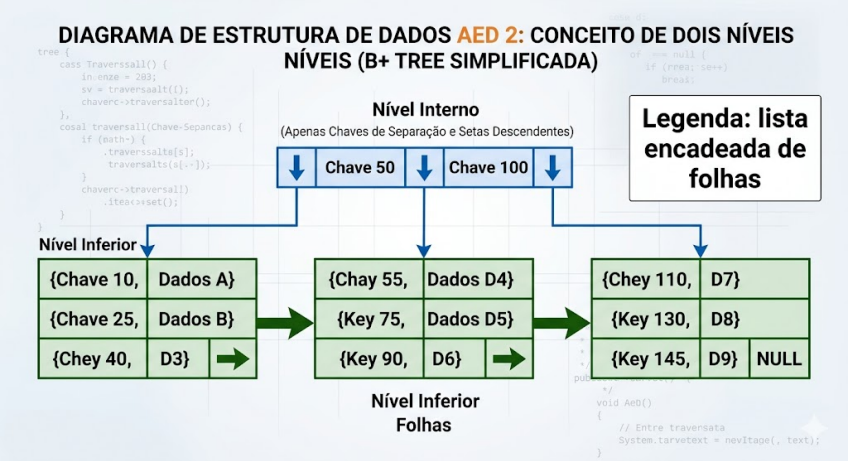

Figure 3: Árvore B+: dados só nas folhas; folhas encadeadas para consultas de faixa.

0.4.2 Por que B+ é melhor?

- **Range Queries (Consultas de Faixa):** Para fazer `SELECT * FROM users WHERE id BETWEEN 100 AND 200`:
- Buscamos o 100 (Custo $\log_M N$).
- Percorremos a lista encadeada das folhas até passar de 200.
- Na Árvore B clássica, teríamos que subir e descer na árvore várias vezes (in-order traversal custoso em disco).
- **Densidade:** Como nós internos não guardam dados pesados, cabem muito mais chaves em um bloco de disco, aumentando o fator de ramificação M e diminuindo a altura.

0.5 5. Exemplo Conceitual (Ordem 4)

Imagine uma Árvore B de ordem 4 (Máximo 3 chaves por nó). Inserindo: `10, 20, 30`
Nó: `[10, 20, 30]`
Inserindo `40`: Nó estoura (4 chaves). Dividimos!

- `20` sobe.

- Raiz: `[20]`
- Filhos: `[10]` e `[30, 40]`

Inserindo `50, 60`: O filho da direita enche `[30, 40, 50, 60]`. Split!

- `50` sobe para a raiz.
- Raiz: `[20, 50]`
- Filhos: `[10]`, `[30, 40]`, `[60]`

0.6 6. Implementação

Implementar uma B-Tree do zero é um projeto complexo (centenas de linhas para gerenciar leitura/escrita de arquivos). Em Java, não há B-Tree na biblioteca padrão (pois é focada em memória RAM), mas bibliotecas como **H2 Database** ou **Apache Derby** implementam B+ Trees em Java puro.

Para fins educacionais em AED2, focamos no entendimento conceitual:

- Calcular altura mínima/máxima.
- Simular inserções e splits no papel.
- Entender o custo de I/O.

0.6.1 Complexidade

- Tempo: $O(\log_M N)$ acessos a disco.
- Espaço: $O(N)$.

• Teoria

Para uma árvore B de ordem M com N chaves, a altura h satisfaz: $\log_M(N+1) \leq h \leq \log_{\lceil M/2 \rceil}((N+1)/2) + 1$. Com M grande (ex: 1000), isso resulta em apenas 2-3 níveis mesmo para bilhões de registros.

0.7 7. Aplicações em Engenharia de Computação

Árvores B e B+ são fundamentais em:

- **Sistemas de Gerenciamento de Banco de Dados (SGBD):** PostgreSQL, MySQL, Oracle usam B+ Trees para índices primários e secundários.
- **Sistemas de Arquivos:** NTFS (Windows), ext4 (Linux), HFS+ (macOS) usam variações de B-Trees para gerenciar metadados e alocação.
- **Bancos de Dados NoSQL:** MongoDB, CouchDB usam B-Trees para índices e consultas eficientes.

- **Sistemas de Cache Distribuído:** Estruturas hierárquicas para gerenciar grandes volumes de dados em memória secundária.

▷ Exemplo

Em um banco de dados com 1 bilhão de registros e uma árvore B+ de ordem 1000, uma busca requer apenas 3 acessos a disco (raiz, nível intermediário, folha). Se cada acesso leva 10ms, a busca completa leva apenas 30ms, comparado a minutos se usássemos busca sequencial.

0.8 8. Exercícios

8.1. Exercícios Conceituais

1 Explique por que árvores B são mais adequadas que árvores binárias para armazenamento em disco. Qual é o impacto do tamanho do bloco de disco no fator de ramificação M?

2 Compare árvore B clássica com árvore B+ em termos de:

- Localização dos dados (nós internos vs folhas).
- Eficiência em consultas de faixa (range queries).
- Uso de espaço em disco.

3 Discuta o processo de split em árvores B. Por que a árvore cresce para cima (aumentando altura) apenas quando a raiz precisa ser dividida?

8.2. Exercícios Analíticos

1 Para uma árvore B de ordem $M = 5$ (máximo 4 chaves por nó), simule a inserção sequencial das chaves: `10, 20, 30, 40, 50, 60, 70, 80, 90`. Mostre a estrutura após cada split.

2 Calcule a altura mínima e máxima de uma árvore B de ordem $M = 100$ com $N = 10^6$ chaves. Quantos acessos a disco são necessários no melhor e pior caso?

3 Para uma árvore B+ usada em um banco de dados:

- Estime o número de chaves que cabem em um nó considerando que cada chave tem 8 bytes e cada ponteiro tem 8 bytes, e o tamanho do bloco é 4KB.
- Calcule o fator de ramificação M resultante.
- Estime a altura para 10^9 registros.

8.3. Exercícios de Programação

1 Implemente uma simulação simplificada de uma árvore B de ordem pequena (ex: $M = 3$ ou $M = 4$) em memória RAM, suportando inserção e busca. Inclua visualização da estrutura da árvore.

2 Escreva um programa que compare o número de acessos a disco teóricos entre:

- Uma árvore binária balanceada (AVL).
- Uma árvore B de ordem $M = 100$.
- Uma árvore B de ordem $M = 1000$.

Para diferentes valores de N (de 10^3 a 10^9).

3 Pesquise e documente como um SGBD real (PostgreSQL, MySQL) implementa índices B+ Tree. Escreva um relatório descrevendo:

- Como os dados são organizados em páginas/blocos.
- Como o split é implementado.
- Como consultas de faixa são otimizadas usando a lista encadeada de folhas.