Estruturas de Dados

Aula 11 — Árvores B e B+

Prof. Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal da Bahia — Campus Feira de Santana

2026

Agenda

Árvores B

Inserção em Árvores B

Árvores B+

Comparação B vs B+

Aplicações

Árvores B (B-Trees)

Bayer e McCreight, 1972

O que é uma Árvore B?

Definição

Árvore de busca **balanceada** projetada para **dispositivos de armazenamento secundário** (HD, SSD). Minimiza os acessos de E/S ao disco. Usada na maioria dos **Sistemas de Arquivos** e **Bancos de Dados**. Criada por **Bayer e McCreight** em 1972.

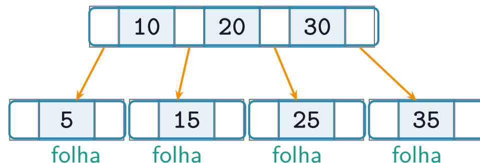
Por que não ABB ou AVL?

Em ABB/AVL, cada nó acessa **1 chave**. Para 10^9 registros, seriam ∼30 acessos a disco. Na Árvore B, cada nó pode ter **milhares de chaves**: apenas **2–4 acessos** ao disco!

Ideia central

- Nó = **página de disco** (ex.: 4 KB)
- Cabe muitas chaves por página
- Árvore **larga e baixa** em vez de alta e estreita

Árvore B de ordem $t = 2$
(max 3 chaves/nó)



Propriedades da Árvore B

Campos de cada nó x

- $n[x]$: número de chaves no nó
- Chaves em **ordem crescente**: $k_1 \leq k_2 \leq \cdots \leq k_{n[x]}$
- $folha[x]$: booleano (true se folha)
- $n[x] + 1$ ponteiros para filhos $c_1, \ldots, c_{n[x]+1}$ (NULL se folha)

Propriedade de ordenação

Se k_i e qualquer chave na subárvore c_i:

$$k_1 \leq \text{chave}_1 \leq k_2 \leq \cdots \leq \text{chave}_{n[x]} \leq k_{n[x]+1}$$

Restrições de tamanho (grau mínimo $t \geq 2$)

Nó	**Chaves**
Raiz (não vazia)	≥ 1
Qualquer nó não raiz	$\geq t - 1$
Qualquer nó	$\leq 2t - 1$

Nó com $2t - 1$ chaves: **completo**.

Profundidade uniforme

Toda folha tem a mesma profundidade h.
Garante custo idêntico para qualquer busca.

Altura da Árvore B

Teorema

Árvore B de grau mínimo $t \geq 2$ com $n \geq 1$ chaves:

$$h \leq \log_t \frac{n+1}{2}$$

(Cormen et al., 2001)

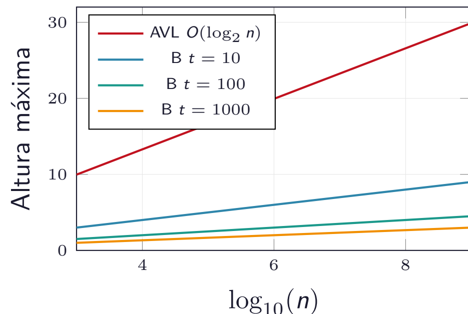
Comparação prática

Para $n = 10^9$ chaves e $t = 1000$:

- ABB: até 10^9 comparações
- AVL: até $\sim$30 acessos a disco
- Árvore B ($t = 1000$): apenas **3 acessos**!

Complexidades

Busca	$O(t \cdot \log_t n)$
Inserção	$O(t \cdot \log_t n)$
Remoção	$O(t \cdot \log_t n)$



Árvore B — Struct em C

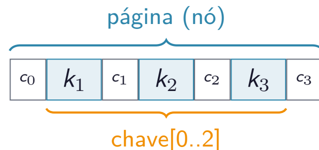
pagina.h — Definição da struct

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define T 2 /* grau mínimo */
#define MAX (2*T - 1)/* max chaves */
#define MIN (T - 1) /* min chaves */
typedef struct pagina{
  int chave[MAX];/*chaves */
  int n; /* qtde chaves */
  bool folha; /* é folha? */
  struct pagina *filho[MAX+1];
}pagina;
pagina* novaPagina(bool folha) {
  pagina *p=(pagina*)malloc(sizeof(
      pagina));
  p->n = 0; p->folha = folha;
  for (int i = 0; i<=MAX;i++)
    p->filho[i] = NULL;
  return p;
}
```

▪ Representação do nó

Cada página armazena até $2t-1$ chaves e $2t$ ponteiros para filhos. Para $t=2$: max **3 chaves** e **4 filhos**.

Layout de uma página ($t=2$)



Árvore B — Busca

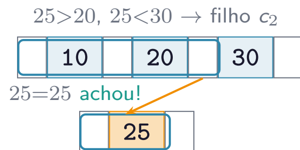
buscar.c

```c
/* Busca 'k' na subarvore com raiz 'x'
   Retorna pagina e indice, ou NULL */
pagina* buscar(pagina *x, int k, int *
        idx){
        int i = 0;
        /* avanca enquanto k > chave[i] */
        while (i < x->n && k > x->chave[i])
                i++;
        if (i < x->n && k == x->chave[i]) {
                *idx = i;
                return x;       /* achou! */
        }
        if (x->folha)
                return NULL;  /* nao existe */
        /* desce para o filho correto */
        return buscar(x->filho[i], k, idx);
}
```

▪ Como funciona

Compara a chave com as chaves do nó da esquerda para a direita. Quando encontra $k \leq \text{chave}[i]$, desce pelo filho c_i. Repete até encontrar ou chegar a uma folha.

Busca por 25



Inserção em Árvores B

Split de nós completos

Ideia geral

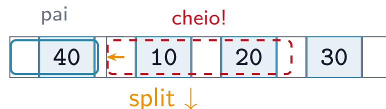
A inserção ocorre sempre nas **folhas**. Se o nó de destino estiver **completo** ($2t-1$ chaves), realiza um **split** antes de inserir. O split pode propagar-se até a raiz.

Split de nó completo

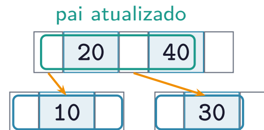
1. Divide o nó em **dois nós** com $t-1$ chaves cada
2. A **chave mediana** sobe para o nó pai
3. O nó pai ganha um novo filho
4. Se o pai também estava cheio, propaga o split

Split de nó completo ($t = 2$, 3 chaves → 2+1+2)

antes: nó cheio

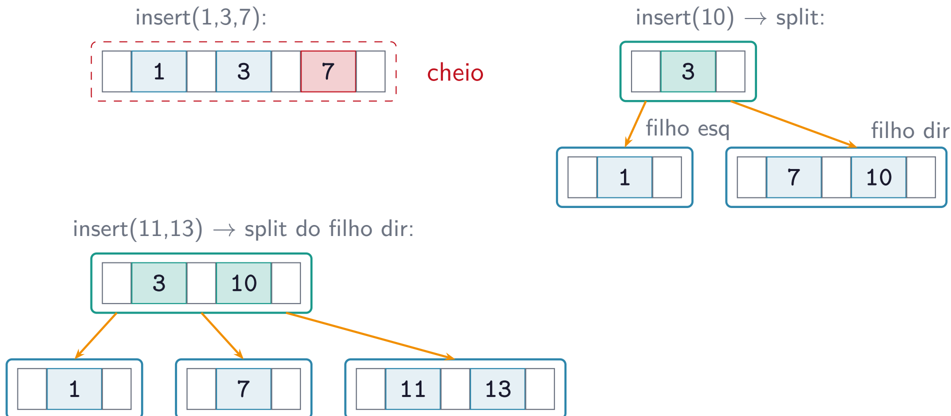


depois: mediana sobe



Inserção — Trace Passo a Passo

Inserindo 1, 3, 7, 10, 11, 13 em árvore B com $t = 2$



Árvore B — Função de Inserção (simplificada)

splitFilho.c

```c
void splitFilho(pagina *x,int i,
    pagina *y) {
  pagina *z = novaPagina(y->folha);
  z->n = T-1;
//copia metade direita de y
  for (int j=0; j<T-1; j++)
    z->chave[j] =y->chave[j+T];
  if (!y->folha)
    for (int j = 0; j < T; j++)
      z->filho[j] =y->filho[j+T];
  y->n = T-1;
//abre espaço no pai x
  for (int j=x->n; j>=i+1; j--)
    x->filho[j+1] =x->filho[j];
  x->filho[i+1] = z;
  for (int j=x->n-1; j>=i; j--)
    x->chave[j+1] = x->chave[j];
/* mediana de y sobe para x */
  x->chave[i] = y->chave[T-1];
  x->n++;
}
```

inserirNaoCheio.c

```c
void inserirNaoCheio(pagina *x,int k){
  int i = x->n - 1;
  if (x->folha){
//desloca chaves para abrir espaco
    while (i >= 0 && k < x->chave[i]) {
      x->chave[i+1] =x->chave[i]; i--; }
    x->chave[i+1] = k;
    x->n++;
  }else{//encontra filho correto
    while (i >= 0 && k<x->chave[i]) i--;
    i++;
    if (x->filho[i]->n == MAX) {
      splitFilho(x, i, x->filho[i]);
      if(k>x->chave[i]) i++;
    }
    inserirNaoCheio(x->filho[i], k);
  }
}
```

Árvores B+ (B+-Trees)

Dados apenas nas folhas

Árvore B+ — Definição e Diferenças

O que é a Árvore B+?

Variação da Árvore B onde os **dados reais** (registros completos) ficam **apenas nas folhas**. Os nós internos contêm apenas **chaves de roteamento**. As folhas são ligadas em uma **lista encadeada ordenada**, permitindo varredura sequencial eficiente.

Vantagens sobre a B simples

- Nós internos cabem **mais chaves** (sem dados) → fator de ramificação maior → árvore mais **baixa**
- Varredura sequencial em $O(n)$ pelas folhas
- **Consultas de intervalo** muito eficientes
- Busca sempre percorre até a folha: **custo uniforme**

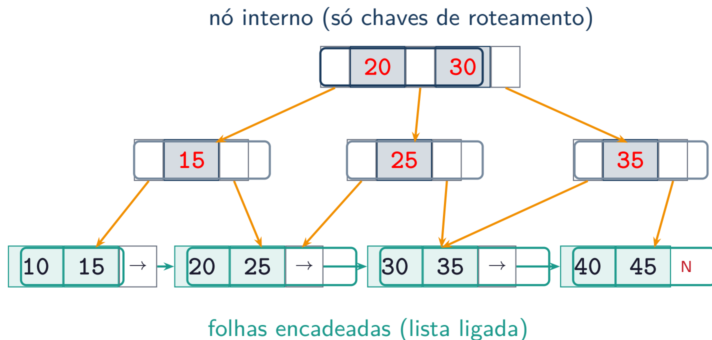
Uso predominante

A maioria dos SGBDs modernos usa **B+**, não B simples: Oracle, PostgreSQL, MySQL/InnoDB, SQL Server, SQLite.

Diferença estrutural

	B	**B+**
Dados em nós internos	Sim	Não
Dados nas folhas	Sim	Sim
Folhas encadeadas	Não	Sim

Árvore B+ com chaves {10, 15, 20, 25, 30, 35, 40, 45}



Árvore B+ — Busca e Varredura Sequencial

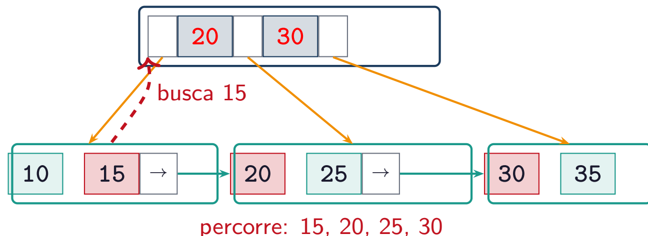
Busca pontual

Semelhante à Árvore B: percorre nós internos usando as chaves de roteamento. A diferença é que a busca **sempre** continua até uma folha, mesmo que a chave seja encontrada em um nó interno.

Busca de intervalo

Para buscar todas as chaves com $k_1 \leq k \leq k_2$:

1. Busca pontual por $k_1 \rightarrow$ chega a folha
2. Percorre a lista encadeada até $k > k_2$
3. Custo: $O(\log_t n + m)$ onde m é o nº de resultados



Comparação

Árvore B vs. Árvore B+

Árvore B vs. Árvore B+ — Comparação

Característica	**Árvore B**	**Árvore B+**
Dados (registros)	Em **todos** os nós	Apenas nas **folhas**
Nós internos	Chaves + dados + ponteiros	Apenas chaves + ponteiros
Fator ramificação	Menor (nós maiores)	**Maior** (nós internos enxutos)
Altura	Ligeiramente maior	**Menor**
Busca pontual	Pode terminar em nó interno	Sempre vai até a folha
Varredura sequencial	Percurso na árvore $O(n \log n)$	Lista encadeada $O(n)$
Consulta intervalo	Menos eficiente	**Muito eficiente**
Uso típico	Sistemas de arquivos	**SGBDs** (Oracle, MySQL...)

Árvore B vs. Árvore B+ — Regra Prática

▪ Regra prática

Se precisa de **consultas de intervalo** (ex.: `BETWEEN` em SQL) ou **varredura sequencial**, use B+. Se a maioria das operações são buscas pontuais isoladas, B simples também serve.

Aplicações

Onde B e B+ são usadas na prática

Aplicações de Árvores B e B+

Sistemas de Arquivos

- **NTFS** — Windows (MFT indexada com B+)
- **HFS+ / APFS** — macOS
- **ext4 / JFS** — Linux
- **Btrfs** — Linux moderno
- **ReFS** — Windows Server

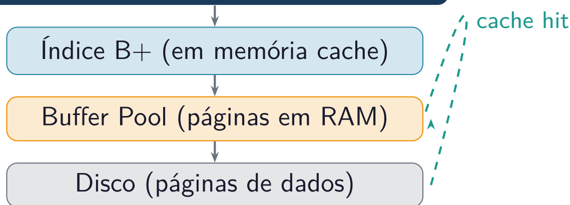
Localizam arquivos entre bilhões com poucos acessos ao disco.

Bancos de Dados

- **Oracle**, **IBM DB2**
- **PostgreSQL**, **MySQL/InnoDB**
- **SQL Server**, **SQLite**
- **MongoDB** (WiredTiger)

Índices B+ permitem buscas em $O(\log_t n)$ com terabytes.

Pilha de software com índice B+



Custo prático

Tabela com 10^9 registros e $t = 200$:

Altura da B+ $\leq \log_{200}\left(\frac{10^9+1}{2}\right) \approx 4$

$\rightarrow$ **4 leituras de página** para qualquer busca.

Exercícios

Exercício 1 — Construção

Construa uma Árvore B com $t = 2$ inserindo as chaves na ordem:

5, 10, 15, 20, 25, 30, 35

Mostre o estado da árvore após cada split.

Exercício 2 — Altura

Para $n = 10^6$ chaves, calcule a altura máxima de:

- AVL $(t = 2)$
- Árvore B com $t = 10$
- Árvore B com $t = 100$

Use $h \leq \log_t \frac{n+1}{2}$.

Exercício 3 — Intervalo

Explique por que, em uma Árvore B+, a busca por um intervalo $[k_1, k_2]$ é mais eficiente do que em uma Árvore B simples. Qual a complexidade de cada uma?

Exercício 4 — Desafio

Modifique a struct `pagina` para incluir um ponteiro `*prox` (para a folha seguinte). Implemente a função de varredura sequencial de todas as chaves a partir de uma chave inicial `k`.

Fim da Aula 11

Dúvidas e próximos passos

Próxima Aula e Referências

Próxima aula — Tabela Hash

- Função de **hash**: mapear chave $\rightarrow$ índice
- Tratamento de **colisões**: encadeamento e endereçamento aberto
- Fator de carga α e **rehashing**
- Busca, inserção e remoção em $O(1)$ amortizado

Resumo da Aula 11

- B-Tree: árvore balanceada para disco
- Folhas na mesma profundidade sempre
- B+: dados só nas folhas, folhas encadeadas
- Intervalo e varredura: B+ em $O(\log_t n + m)$

Referências

- CORMEN et al. *Introduction to Algorithms*, 4ª ed. Cap. 18.
- SILBERSCHATZ et al. *Operating System Concepts*, 10ª ed.

Dúvidas?

carolsoko@ifba.edu.br

IFBA – Campus Feira de Santana