

Engenharia de Computação

Estrutura de Dados 2

Aula 13 – Árvore Paginada

Prof. Muriel de Souza Godoi
muriel@utfpr.edu.br

Cenário Atual

- Acesso a disco é **caro** (lento)
- **Pesquisa binária** é útil em índices ordenados...
 - mas com índice grande que **não cabe em memória principal**, pesquisa binária exige muitos acessos a disco
- Exemplo:
 - 15 itens podem requerer **4** acessos
 - enquanto 1.000 itens podem requerer até **11** acessos

Cenário Atual

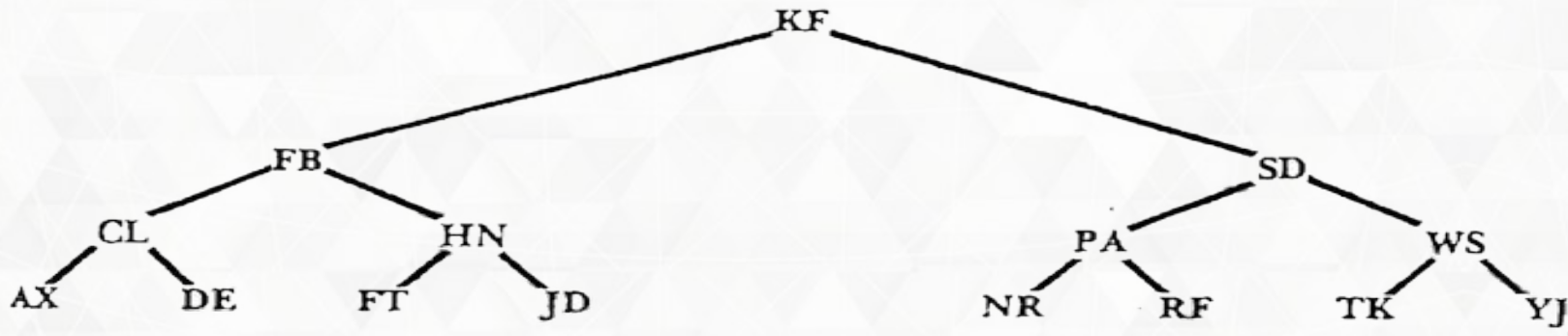
- Manter em disco um índice ordenado para busca binária tem custo proibitivo
 - Inserir ou eliminar, mantendo o arquivo ordenado custa muito caro.
- Necessidade de método com inserção e eliminação com apenas efeitos locais, isto é, que não exija a reorganização total do índice
- Possível solução?
 - Poderíamos usar uma Árvore Binária de Busca?

Árvore Binária de Busca

- Arquivo de **Índice ordenado**:

AX CL DE FB FT HN JD KF NR PA RF SD TK WS YJ

- Índice utilizando **Arvore Binária de Busca**



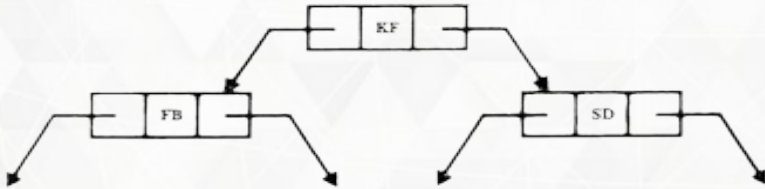
Representação da ABB no arquivo

- **Registros** (tam. fixo) são mantidos em arquivo
- **Ponteiros** (Esq e Dir) indicam onde (RRN) estão os registros filhos.



	Key	Esq	Dir
0	FB	10	8
1	JD		
2	RF		
3	SD	6	13
4	AX		
5	YJ		
6	PA	11	2
7	FT		

	Key	Esq	Dir
8	HN	7	1
9	KF	0	3
10	CL	4	12
11	NR		
12	DE		
13	WS	14	5
14	TK		



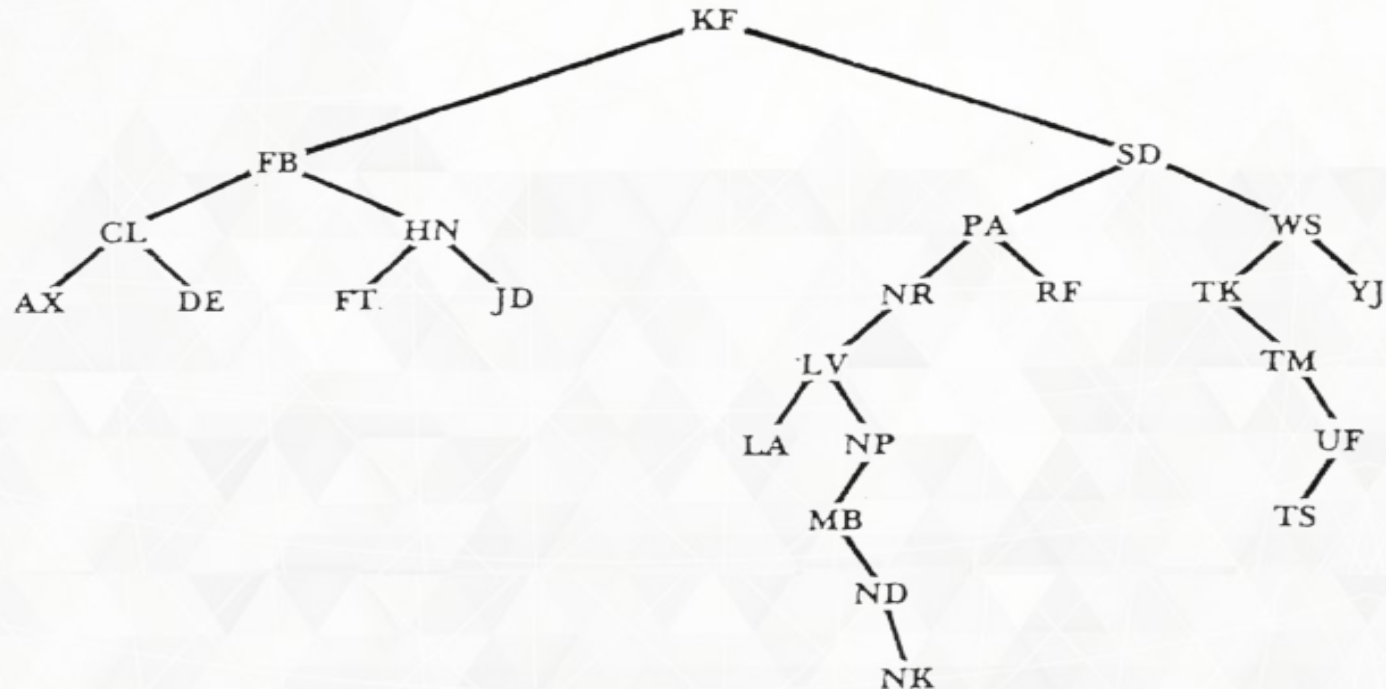
Vantagens da ABB

- Ordem lógica dos registros \neq ordem física no arquivo
 - Ordem lógica: dada por ponteiros esq e dir
 - Registros não precisam estar fisicamente ordenados
- Inserção de uma nova chave no arquivo
 - É necessário saber onde inserir
 - Busca pelo registro é necessária, mas reorganização do arquivo não

Problema da ABB

- Desbalanceamento!

- **Ex.:** Inserção das chaves **NP MB TM LA UF ND TS NK**

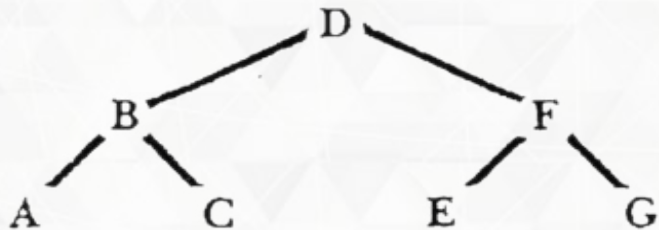


Árvore AVL

- Para todo nó: as alturas de suas duas subárvores diferem de, no máximo, **1**.





- Árvore **Completamente Balanceada** X **AVL Construída**



Árvore AVL – Análise da Solução

- Árvores binárias de busca balanceadas garantem eficiência
 - AVLs
- Busca no pior caso
 - Árvore binária perfeitamente balanceada:
 - Altura da árvore, ou seja, $\log_2 (N+1)$
 - AVL
 - $1.44 * \log_2 (N+2)$ – **44% mais alta no máximo**
- **Exemplo:** com 1.000.000 chaves
 - Árvore binária perfeitamente balanceada:
 - busca em até 20 níveis
 - AVL:
 - busca em até 28 níveis

Árvore AVL – Análise da Solução

- Problema
 - Se as chaves estão em memória secundária, ainda há muitos acessos!
 - 20 ou 28 seeks são **inaceitáveis!**
- Até agora...
- Árvores binárias de busca
 - Dispensam ordenação dos registros 
 - Mas têm número excessivo de acessos 
- Nova possível solução?
 - **Árvores Binárias Paginadas**

Árvores Binárias Paginadas

- **Paginação**

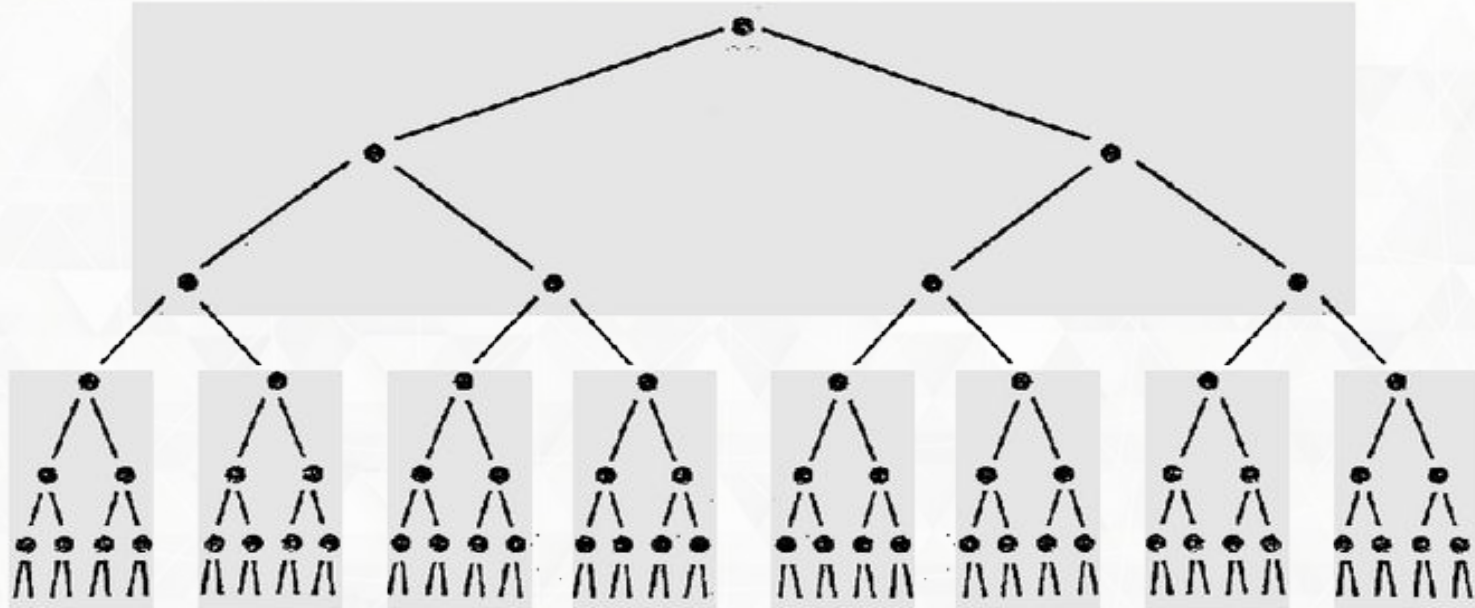
- A busca (seek) por uma posição específica do disco é muito lenta
- Mas, uma vez na posição, pode se ler uma grande quantidade de registros sequencialmente a um custo relativamente pequeno

- Noção de **página** em sistemas paginados

- Feito o seek, todos os registros de uma mesma "página" do arquivo são lidos
- Esta página pode conter um número grande de registros
- Se o próximo registro a ser recuperado estiver na mesma página já lida, evita-se um novo acesso ao disco

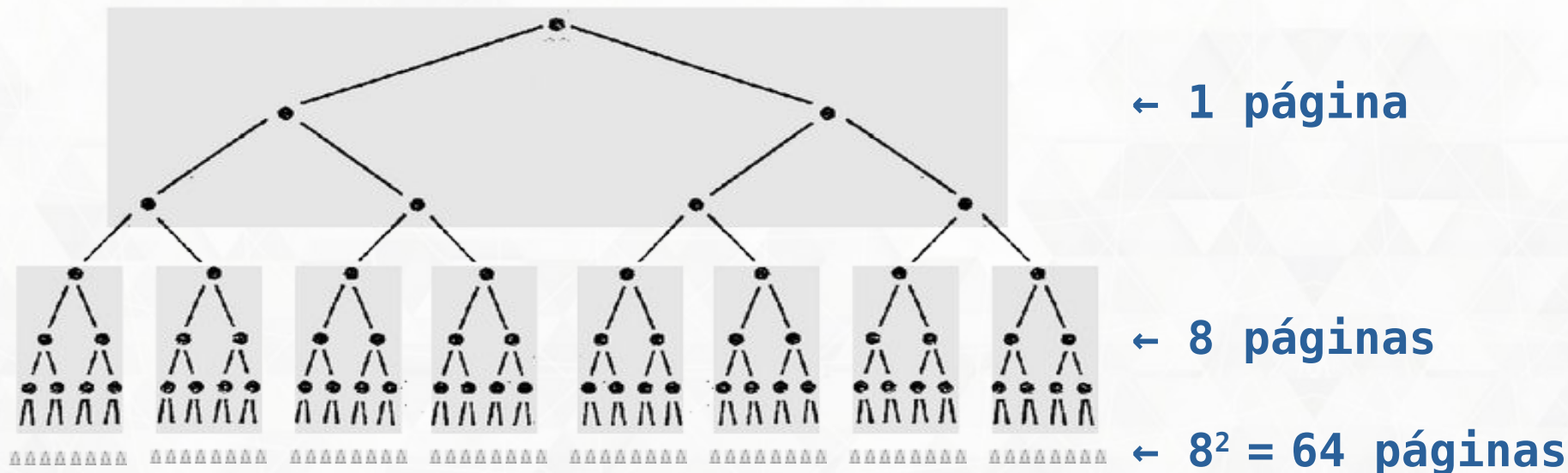
Árvores Binárias Paginadas

- 7 registros por página (por seek);
 - Árvore de altura 2 e ordem 8
 - Qualquer um dos 63 registros pode ser acessado em, no máximo, 2 acessos



Árvores Binárias Paginadas


- Se a árvore é estendida com um nível de paginação adicional, adicionamos **64 novas páginas**
 - Podemos encontrar qualquer uma das 511 ($64 \times 7 + 63$) chaves fazendo apenas **3 seeks**



Eficiência da Árvore Paginada

- Mais realisticamente....supondo que
 - Cada página de uma árvore ocupa 8KB e armazena 511 chaves
 - Cada página contém uma árvore completa perfeitamente balanceada de altura 9 ($=\log_2 512$)
 - A árvore de altura 3 pode armazenar 134.217.727 chaves
 - $= (511 + (512 * 511) + (512 * 512 * 511))$

Eficiência da Árvore Paginada

- ABB completa
 - perfeitamente balanceada: $\log_2 (N+1)$
 - Versão paginada: $\log_{k+1} (N+1)$
- Onde:
 - N é o número total de chaves,
 - k é o número de chaves armazenadas em uma página
- ABB (perfeitamente balanceada):
 - $\log_2 (134.217.727) = \mathbf{27 \text{ acessos}}$
- Versão paginada :
 - $\log_{511+1} (134.217.727) = \mathbf{3 \text{ acessos}}$ 

Eficiência da Árvore Paginada

- **Preços a pagar**

- Menos seeks, mas maior tempo na transmissão de dados
- Necessário manter a organização da árvore

- **Pense como seria a construção da árvore**

- Se tivermos todas as chaves a priori:

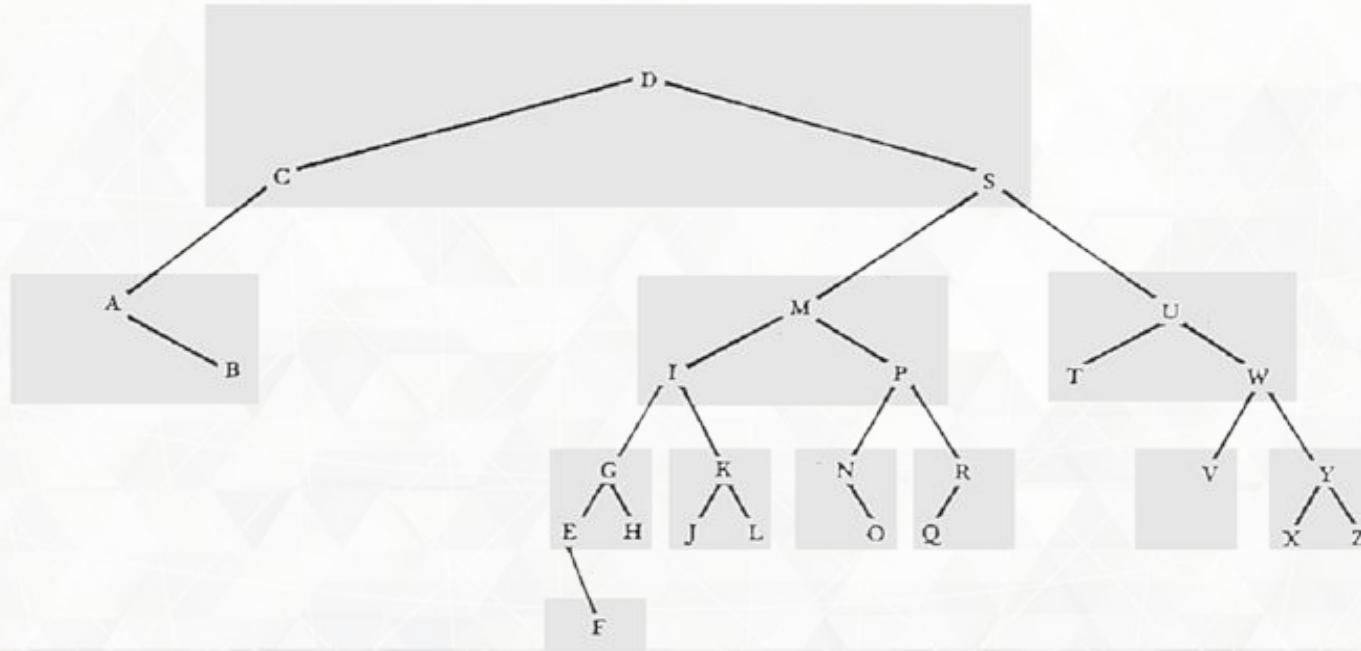
- Ordenar as chaves e inserir de forma a manter balanceada (mediana será a raiz, etc.)

- Se a construção for dinâmica

- Chaves são inseridas em ordem aleatória
- Conforme inserimos, podemos manter a ABB de cada página balanceada (ou seja, uma AVL), rotacionando-a conforme necessário

Exemplo:

- Uma **ABB paginada**
 - Páginas de até 3 chaves por página
 - Sequência de chaves a inserir:
 - C S D T A M P I B W N G U R K E H O L J Y Q Z F X V



Eficiência da Árvore Paginada

- Observe:
 - A construção **top-down** faz com que as primeiras chaves fiquem na página-raiz.
 - No exemplo, C e D (consecutivas e no início do alfabeto) não são boas escolhas para a raiz, pois fatalmente fazem a árvore tender à direita.
- E se rotacionássemos as páginas (como fazemos como os nós)?
- Tente formular um algoritmo para isso...
 - **Muito complexo!**



Problemas a resolver...

- **1** - Como garantir que as chaves da raiz sejam boas separadoras, tal que dividam o conjunto mais ou menos ao meio?
- **2** - Como evitar agrupamento de certas chaves (como C, D e S) na página raiz?
- **3** - Como garantir que cada página contenha um certo número mínimo de chaves?