

# Árvores B

Prof.: Leonardo Tórtoro Pereira  
leonardop@usp.br

\*Material baseado em aulas dos professores: Elaine Parros Machado de Souza, Gustavo Batista, Robson Cordeiro, Moacir Ponti Jr., Maria Cristina Oliveira e Cristina Ciferri.

# Relembrando...

→ Acesso a disco?

- ◆ Lento

→ Pode-se usar índices para guardar apenas as chaves e RRN em memória principal

- ◆ Precisa ordenar os índices para usar busca binária

- ◆ Podem ser índices primários e/ou secundários

→ Mas índices grandes não cabem na memória principal 😞

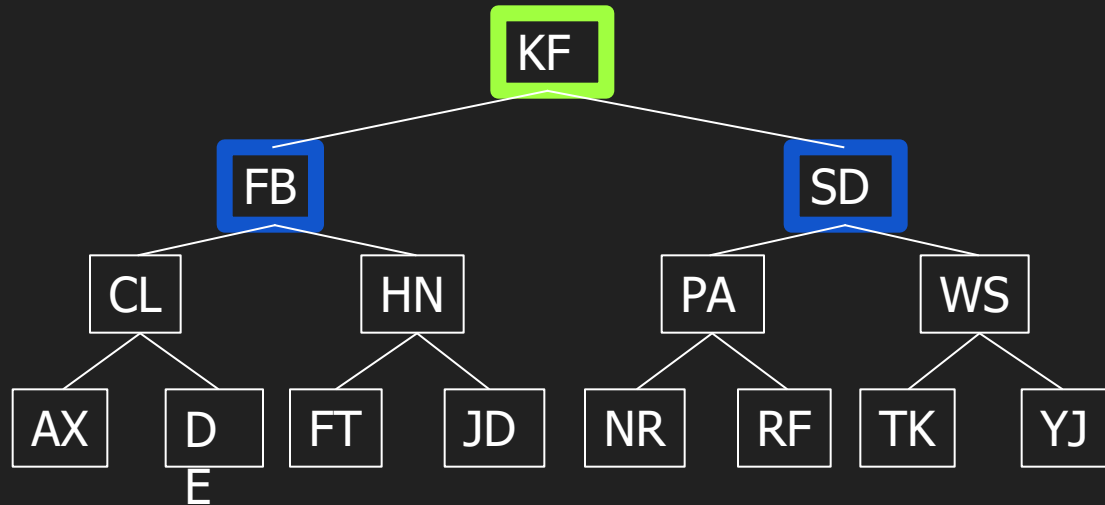
- ◆ Não adianta mais busca binária nem ordenação 😞

## Relembrando...

- Precisamos de um método de inserção e eliminação apenas com efeitos locais
  - ◆ Que não exija reorganização total do índice
- Árvore Binária de busca?

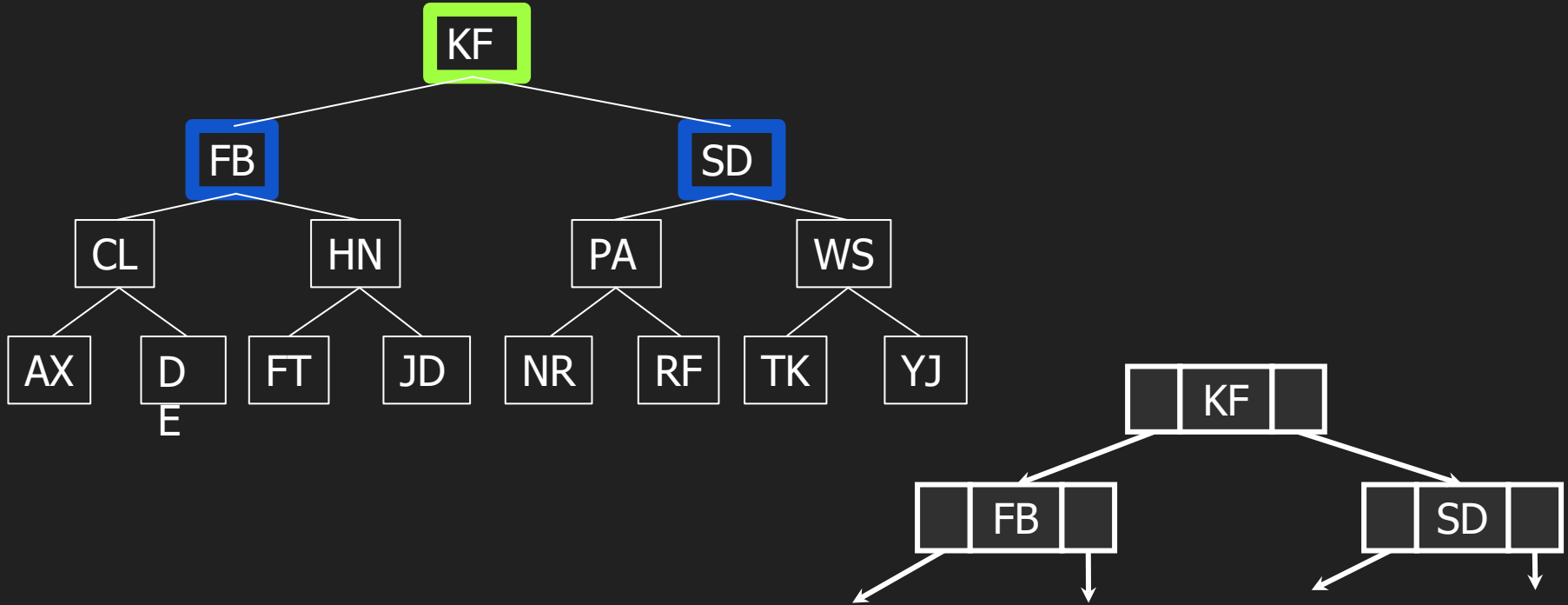
# Árvore Binária de Busca

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



Vetor ordenado e representação por árvore binária

# Árvore Binária de Busca

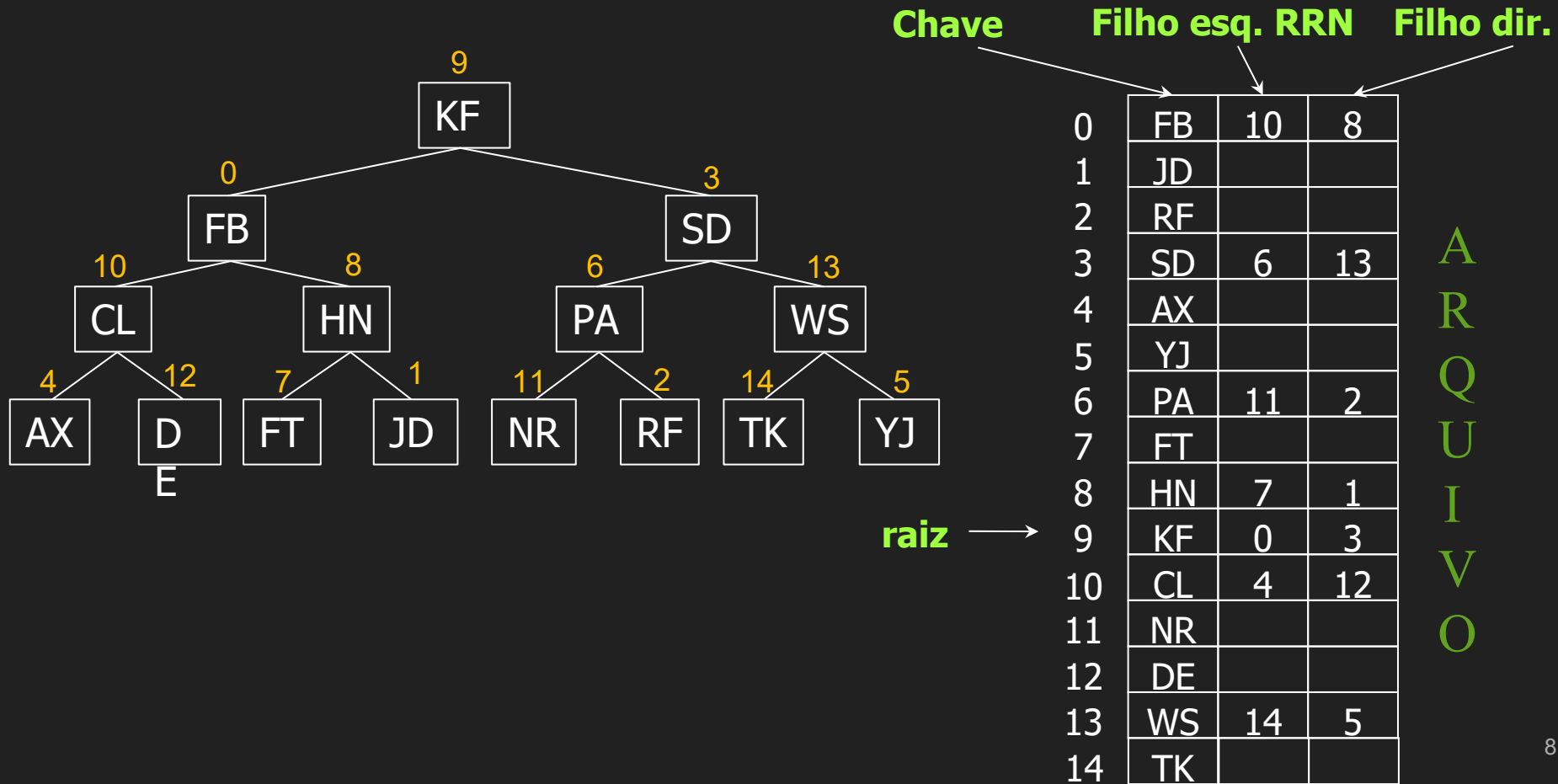


Como organizar uma árvore em disco de maneira eficiente?

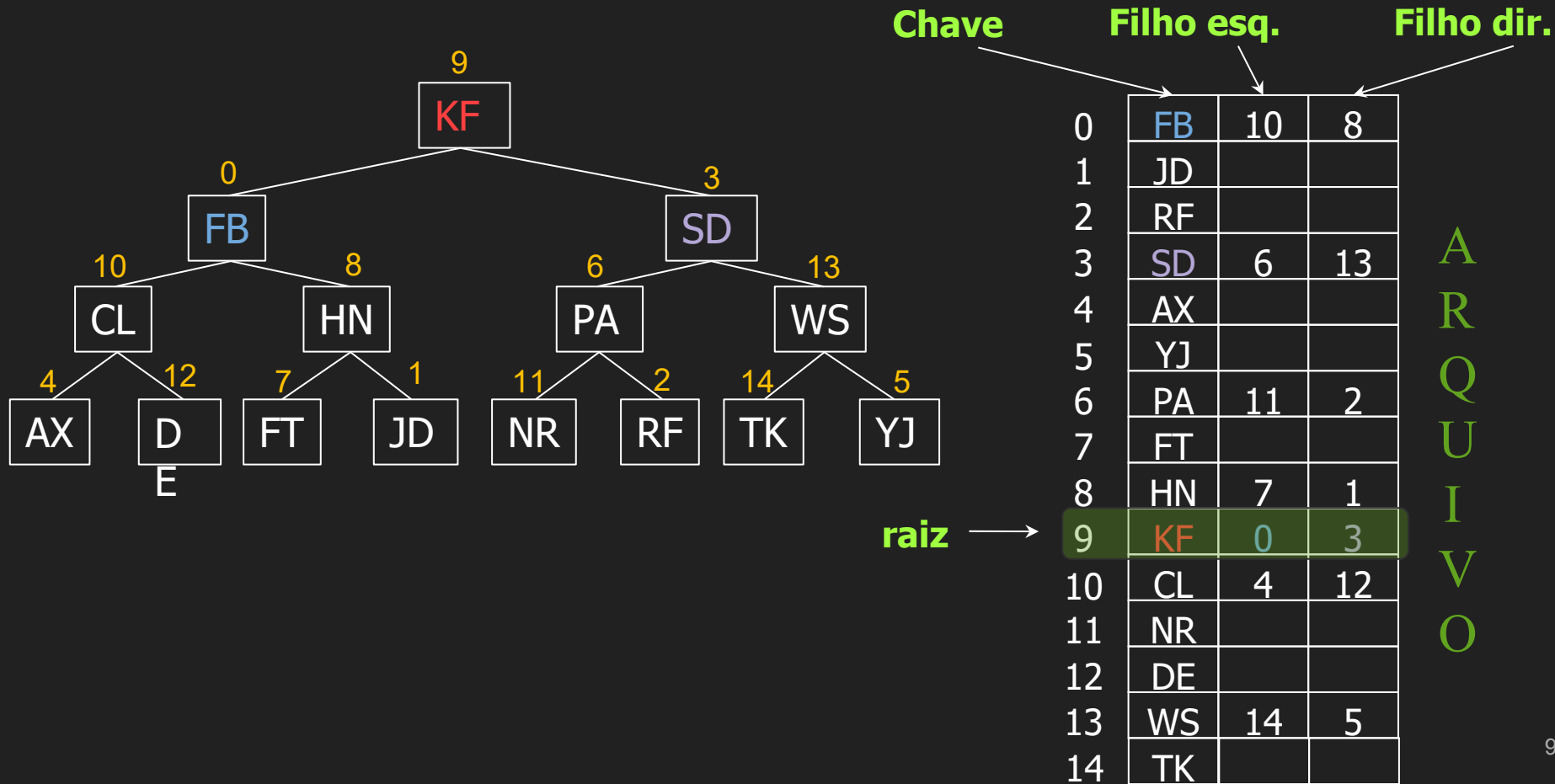
# Representação Lógica da Árvore no Arquivo

- Nós da árvore
  - ◆ Registros em arquivo com 3 campos: chave e 2 “ponteiros” (RRN)
- “Ponteiros” (esq e dir) indicam onde estão os registros dos nós filhos
- Para indexação: um 3o “ponteiro” para o registro correspondente no arquivo de dados
- Posição da raiz no cabeçalho do arquivo

# Representação Lógica da Árvore no Arquivo

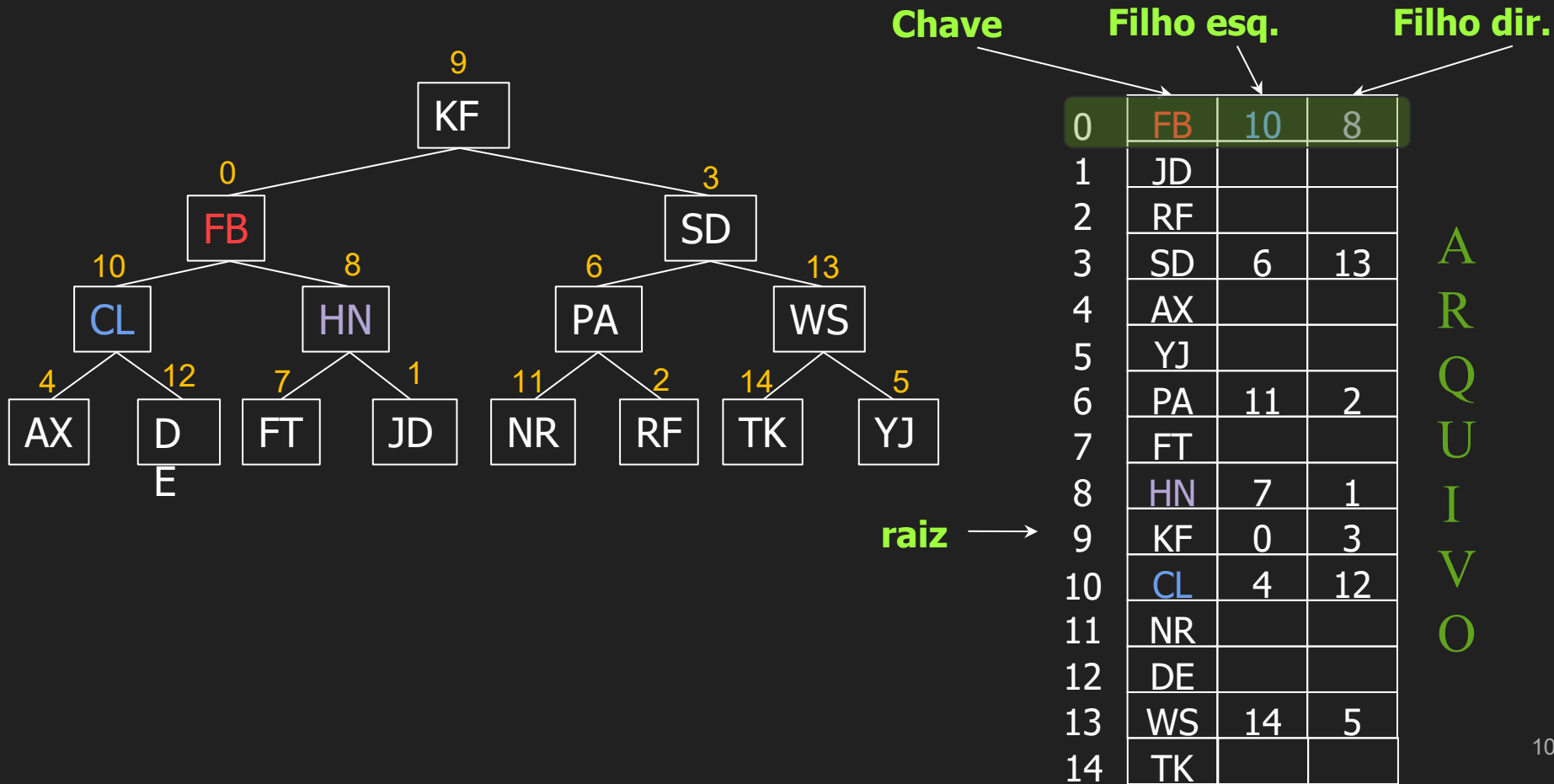


# Representação Lógica da Árvore no Arquivo





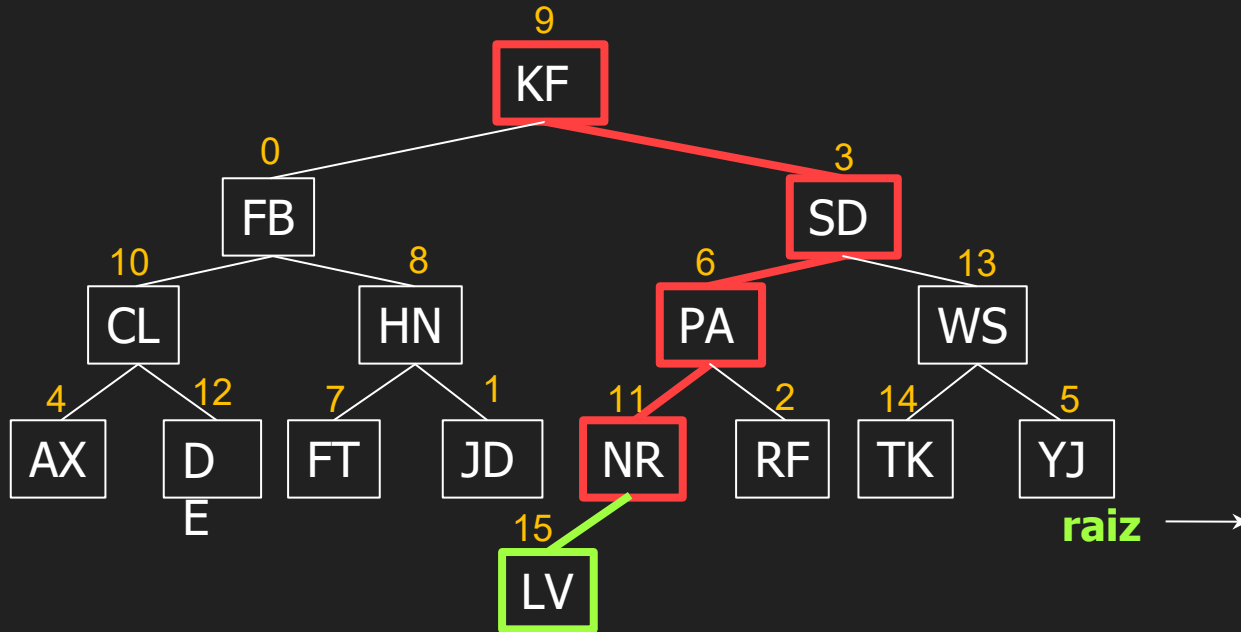
# Representação Lógica da Árvore no Arquivo



# Vantagens

- Ordem lógica dos registros na Estrutura de Dados não está associada à ordem lógica ou física dos registros no arquivo de índice
- Índice não precisa mais ser mantido ordenado
  - ◆ “ponteiros” esq. e dir. permitem recuperar a estrutura lógica da árvore
- Inserção de uma nova chave no arquivo
  - ◆ Necessário saber onde inserir esta chave na árvore
  - ◆ Busca é necessária, mas reorganização do arquivo não

# Inserção da Chave LV



0	FB	10	8
1	JD		
2	RF		
3	SD	6	13
4	AX		
5	YJ		
6	PA	11	2
7	FT		
8	HN	7	1
9	KF	0	3
10	CL	4	12
11	NR	15	
12	DE		
13	WS	14	5
14	TK		
15	LV		

ARQUIVO

# Desempenho

→ Árvores binárias de busca perfeitamente balanceadas

- ◆ Busca no pior caso

  - Altura da árvore

- ◆  $O(\log_2 N)$

→ Ex:

- ◆ Para  $N = 1.000.000$  chaves

- ◆ ABB perfeitamente balanceada

  - Busca em até 20 níveis = 20 seeks

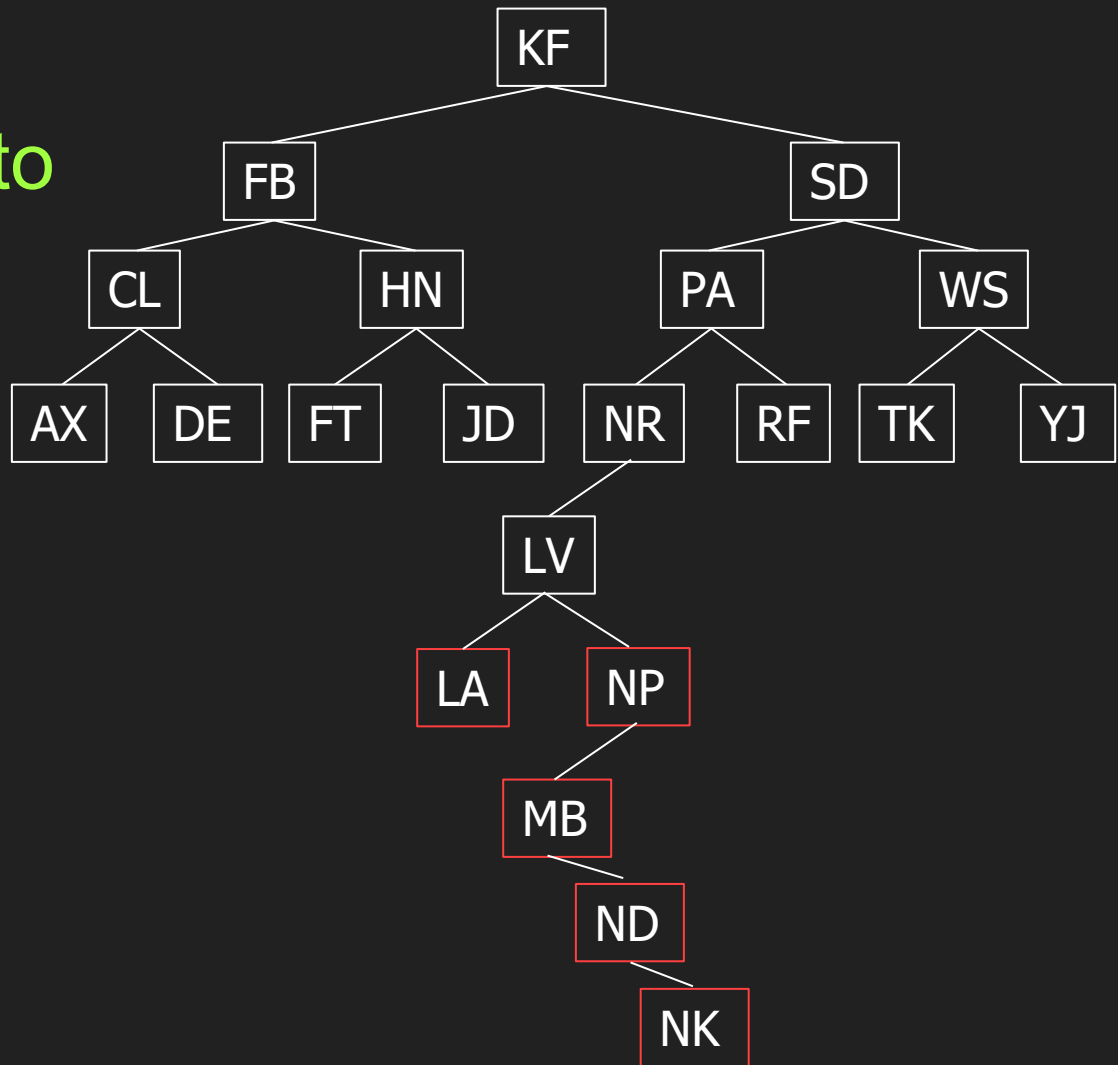
# Problemas?

- Alto número de acesso a discos
- Desbalanceamento

# Desbalanceamento

→ Inserção das  
chaves

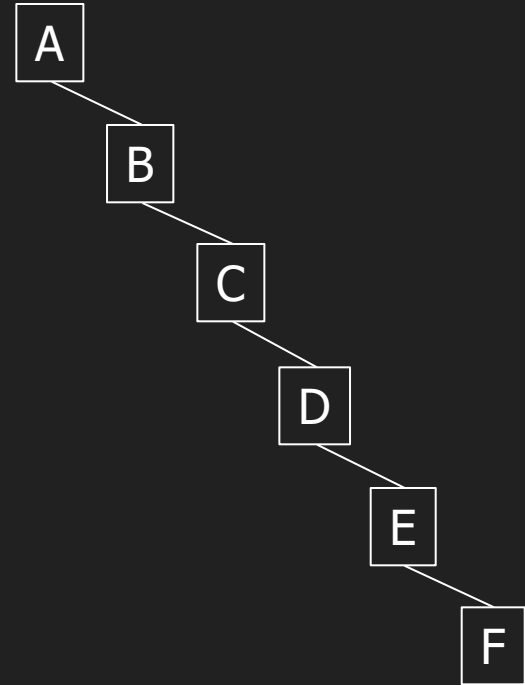
→ NP, MB, LA, ND e  
NK



# Desbalanceamento

→ Caso extremo:

◆ Inserção em ordem alfabética

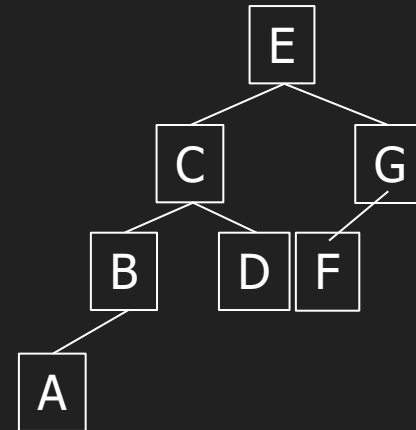
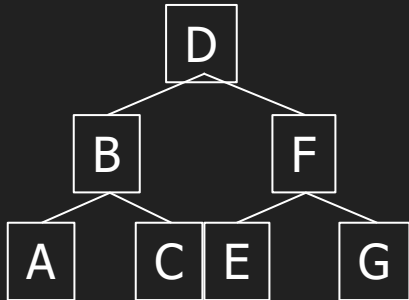


# Possíveis Soluções

→ Árvore AVL - **Reduz Desbalanceamento**

- ◆ 1 nível de diferença entre alturas de 2 subárvores de mesma raiz
- ◆ Garante performance aproximada de uma árvore perfeitamente balanceada

Árvore  
perfeitamente  
balanceada

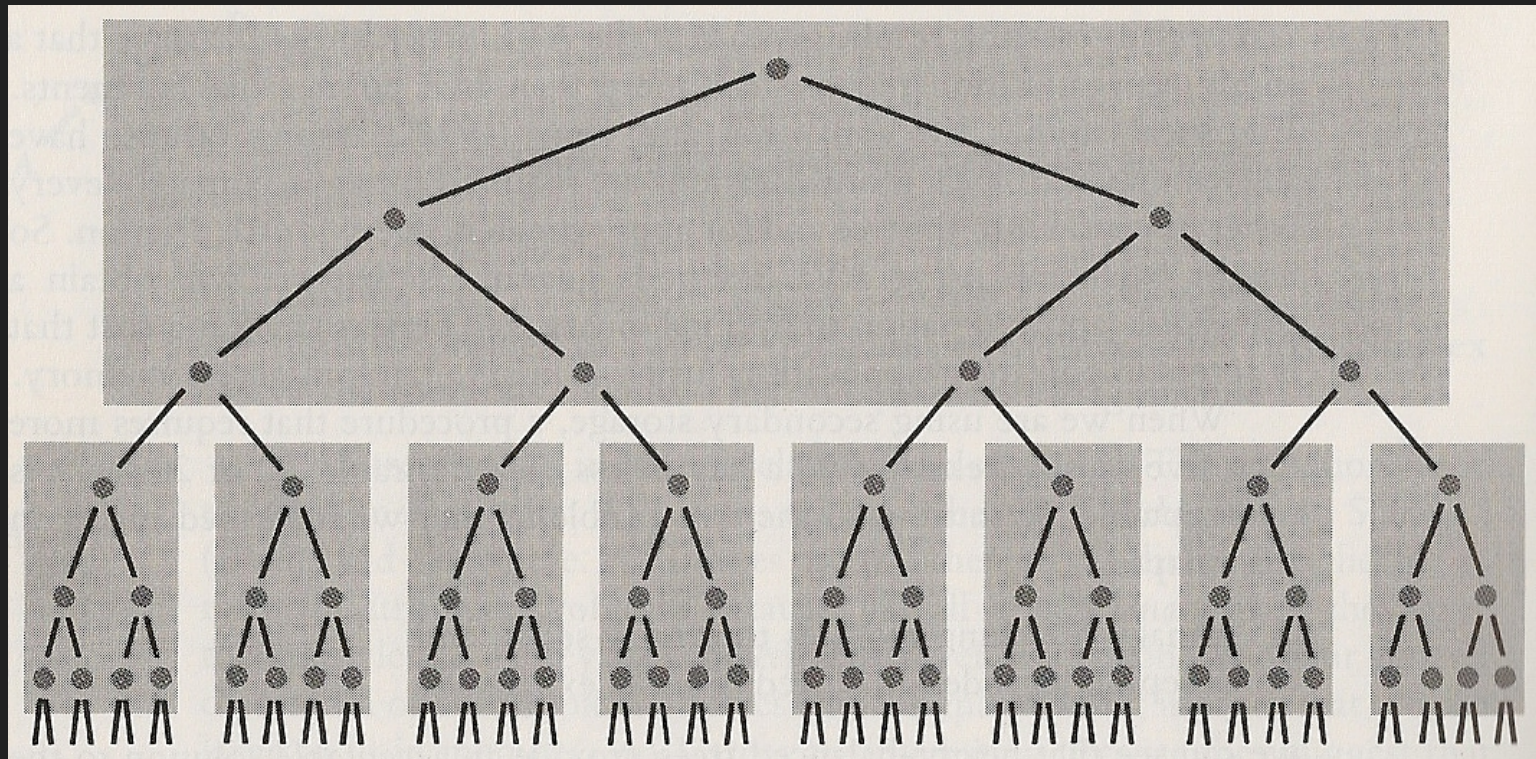


AVL



# Possíveis Soluções

- Paginação - **Reduz número de acessos a disco**
  - ◆ ABB (ou AVL) paginada
  - ◆ Leitura de vários registros num único acesso (seek) à página (bloco) de disco



Alocar múltiplos nós na mesma página

# Problemas?

→ ABB ou AVL paginadas

- ◆ O problema do Desbalanceamento (ABB) ainda existe
- ◆ Se conjunto de chaves é conhecido
  - Construção da árvore é simples
    - Ordena-se o conjunto de chaves
    - Inicia-se a construção pela chave do meio para obter uma árvore balanceada

# Problemas?

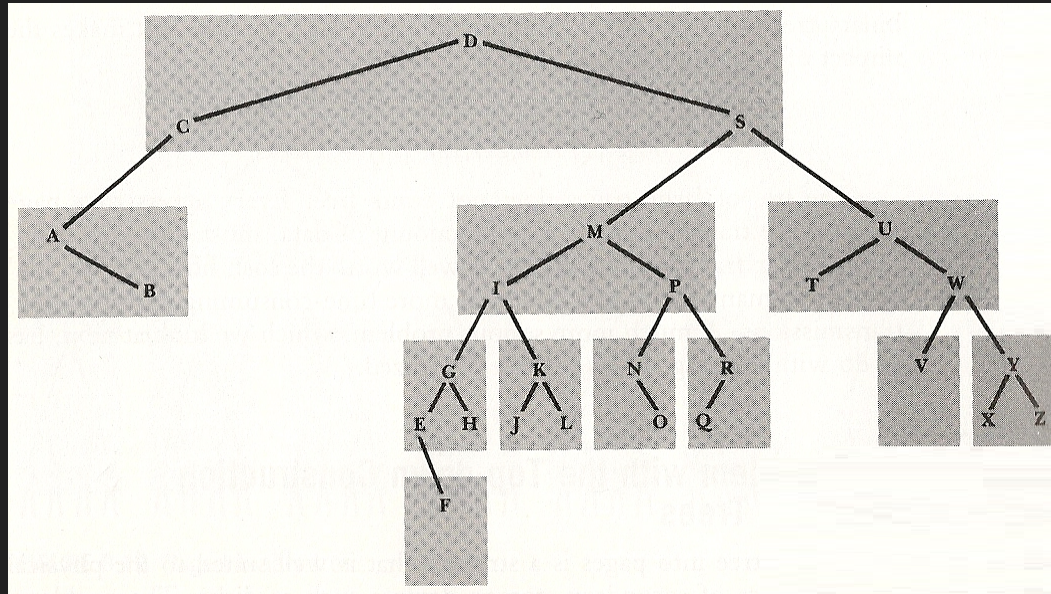
→ ABB ou AVL paginadas

- ◆ Se as chaves são recebidas em uma sequência aleatória
  - Inserção pode levar a desbalanceamento

# Problemas?

→ ABB ou AVL paginadas

→ 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



# Problemas?

→ ABB ou AVL paginadas

◆ Na inserção de uma chave

- A subárvore dentro da página pode sofrer rotações para manter o balanceamento, mas não é possível rotacionar as páginas

# Problemas?

→ ABB ou AVL paginadas

- ◆ Construção top-down, a partir da raiz
  - As chaves iniciais tendem a ficar na raiz
  - No exemplo, C e D não deveriam estar no topo
    - Acabam desbalanceando a árvore

# Problemas?

- Como garantir que as chaves na página raiz são boas separadoras
  - ◆ Dividem o conjunto de chaves de maneira balanceada?
- Como impedir o agrupamento de chaves que não deveriam estar na mesma página?
  - ◆ Ex: C, D e S
- Como garantir que cada página contenha um número mínimo de chaves?



# Árvore B

# Árvore B

- Generalização de uma ABB paginada
  - ◆ Não é binária
  - ◆ Conteúdo de uma página não é mantido como árvore
- 1960s: competição entre fabricantes e pesquisadores
  - ◆ Descobrir um método eficiente para armazenamento e recuperação em sistemas com grandes arquivos de dados

# Árvore B

- 1972: Bayer and McCreight (trabalhando pela Boeing) publicam o artigo Organization and Maintenance of Large Ordered Indexes
- 1979: árvores-B viram padrão em sistemas de arquivos de propósito geral
- Atualmente: árvores-B são o padrão em SGBDs (Sistemas de Gerenciamento de Banco de Dados)
  - ◆ Na verdade, suas variantes, como B\*, B+, B-link, etc.

## Modern B-Tree Techniques:

<https://w6113.github.io/files/papers/btreesurvey-graefe.pdf>

# Modern B-Tree Techniques

“In summary, the core design of B-trees has remained unchanged in 40 years: balanced trees, pages or other units of I/O as nodes, efficient root-to-leaf search, splitting and merging nodes, etc. On the other hand, an enormous amount of research and development has improved every aspect of B-trees including data contents such as multi-dimensional data, access algorithms such as multi-dimensional queries, data organization within each node such as compression and cache optimization, concurrency control such as separation of latching and locking, recovery such as multi-level recovery, etc.”

- Goetz Graefe. 2011. Modern B-Tree Techniques. Found. Trends databases 3, 4 (April 2011), 203–402. DOI:<https://doi.org/10.1561/1900000028>

Evolution of tree data structures for indexing: more exciting than it sounds

# Variantes de B-Tree

B-tree	B <sup>+</sup> -tree	B <sub>link</sub> -tree	DPTree
wB <sup>+</sup> -tree	NV Tree	FPTree	FASTFAIR
HiKV	Masstree	Skip List	ART
WORT	CDDS Tree	Bw Tree	HOT
KISS Tree	VAST Tree	FAST	HV Tree
UB Tree	LHAM	MDAM	Hybrid B <sup>+</sup> Tree

# Características Gerais

- Método genérico para armazenamento e recuperação de dados
- Propósito principal:
  - ◆ Organizar e manter um índice para um arquivo de acesso aleatório altamente dinâmico



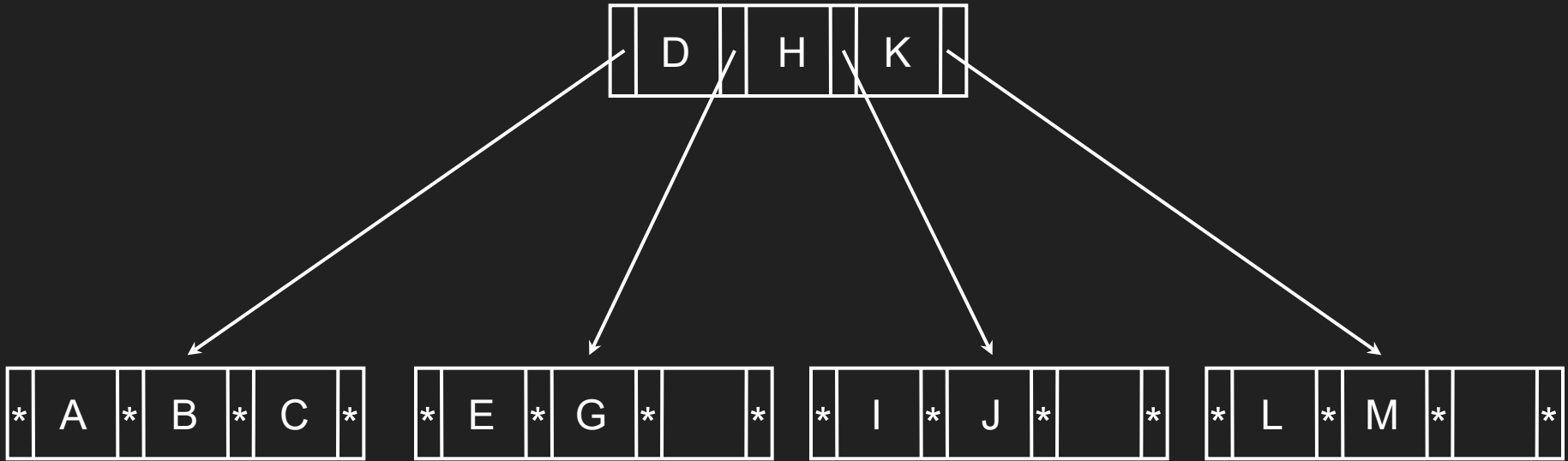
# Características Gerais

- Voltada para índices extremamente volumosos
  - ◆ Com Pool de buffers pequeno
    - Apenas uma parcela do índice pode ser carregada em memória principal
    - Operações baseadas em disco
- Balanceada
- Paginada
- Construção bottom-up (em disco)
  - ◆ Nós folhas --> nó raiz

## Construção Bottom -up

- Chaves “indevidas” não são alocadas na raiz
  - ◆ Chaves adequadas emergem para a raiz naturalmente
- Elimina os problemas de chaves separadoras inadequadas e de chaves extremas
- Não é necessário tratar o problema de desbalanceamento
  - ◆ Balanceamento ocorre naturalmente

# Exemplo

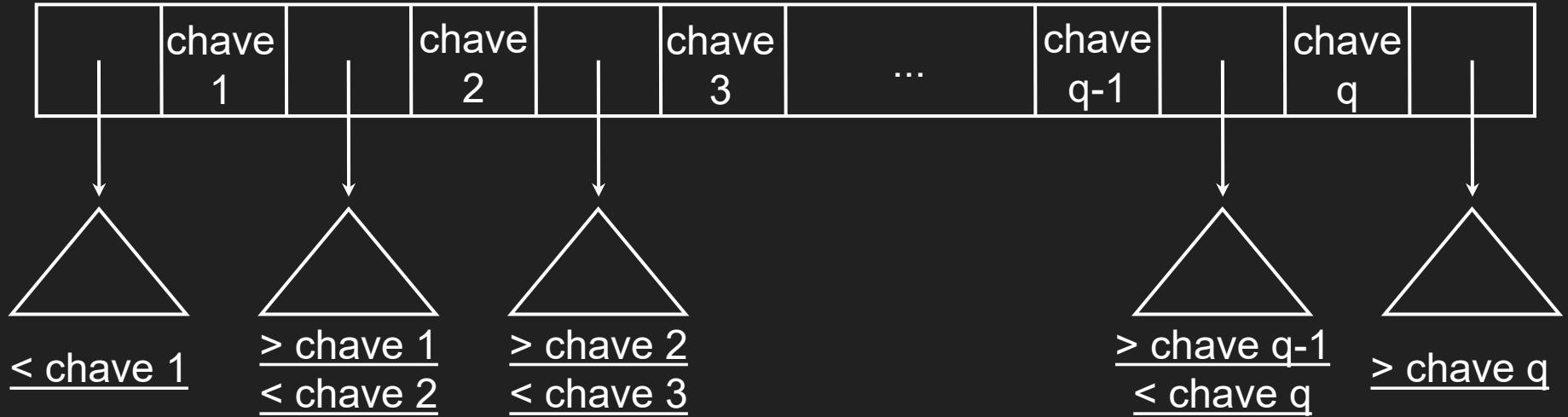


# Características - Nó

→ Cada nó

- ◆ Página de disco
- ◆ Sequência ordenada de chaves
- ◆ Conjunto de “ponteiros” para subárvores (descendentes)
  - Número de ponteiros = número de chaves + 1

# Estrutura Lógica do Nó



# Características - Ordem

→ Ordem

- ◆ Número máximo de ponteiros para descendentes que podem ser armazenados em um nó

→ Ex: árvore-B de ordem 8

- ◆ Máximo de 7 chaves e 8 ponteiros

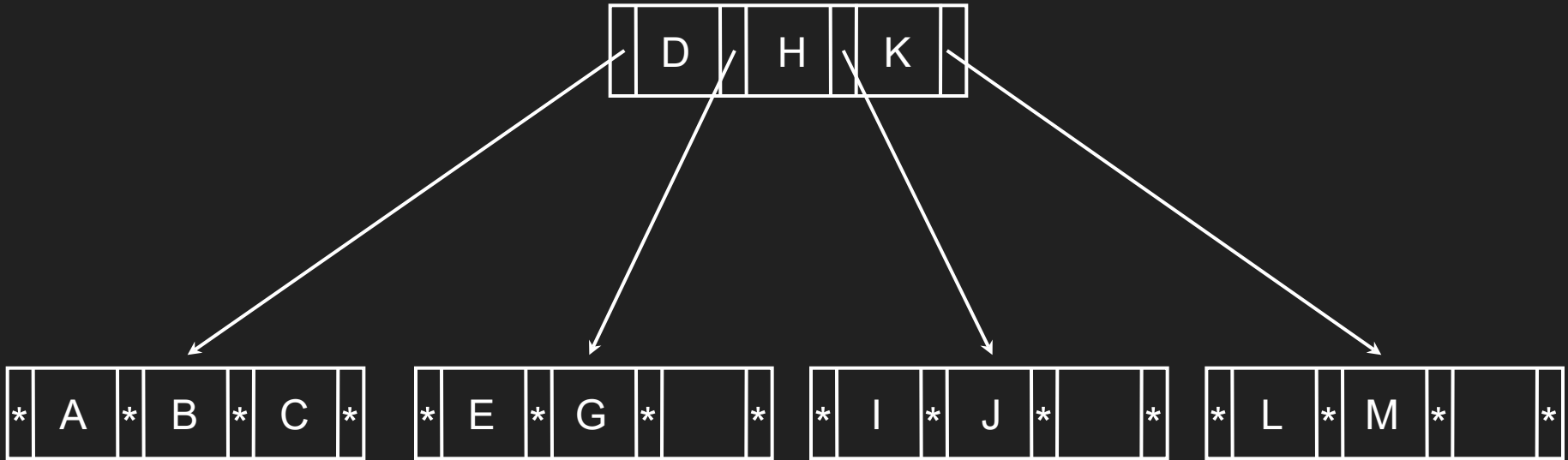
# Características - Ordem

## → Observações

- ◆ Número máximo de ponteiros é igual ao número máximo de descendentes de um nó
- ◆ Nós folhas não possuem filhos
  - Seus ponteiros são nulos

# Exemplo

Ordem = 4





# Estrutura Lógica do Nó

→ Nó com registro de índice

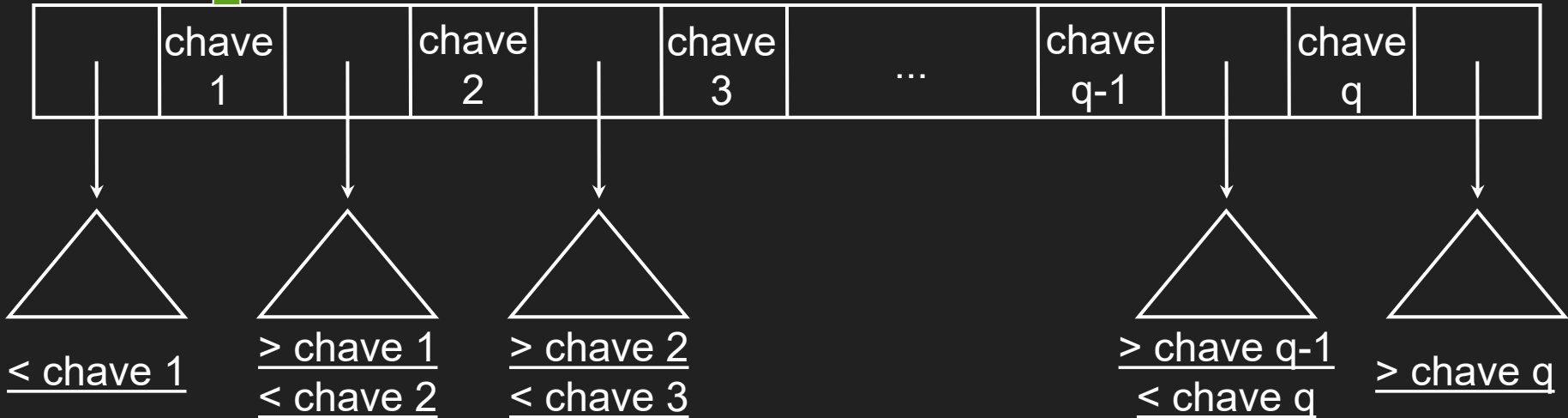
- ◆ Árvore-B utilizada como estrutura de indexação
- ◆ mais comum...(utilizada nas discussões posteriores)

# Estrutura do Nó



campos de tamanho fixo

q é fixo



# Estrutura Lógica do Nó

→ Nó com registro de dados

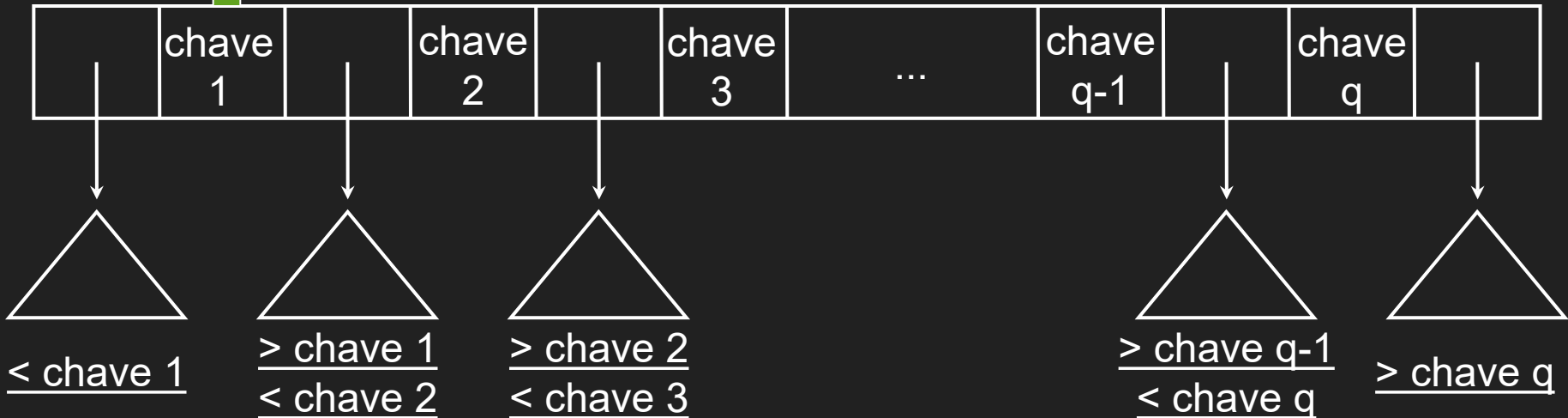
- ◆ Árvore-B utilizada como estrutura de organização de registro de dados

# Estrutura do Nó



campos podem ser de tamanho variável

q pode ser variável

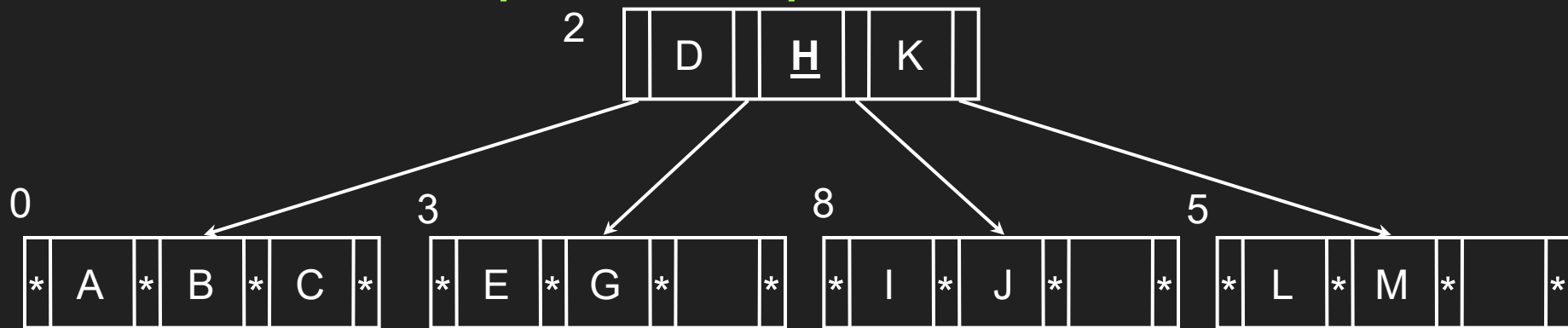


# Exemplo de implementação de nó

```
#define ordem X      /* X é calculado considerando o tamanho de  
página de disco do sistema, o tamanho da chave e dos itens de  
armazenados no nó */
```

```
typedef struct pagina {  
    int contador;      //nro de chaves armazenadas  
    char chaves[ordem - 1]; //assumindo chaves char  
    int filhos[ordem]; // armazena o RRN dos filhos  
    bool folha;  
} PAGINA;
```

# Exemplo de arquivo na árvore B



`pagina.contador`

`pagina.chaves[ ]`

`pagina.filhos[ ]`

*página 2*

3	D	<u>H</u>	K	0	<u>3</u>	<u>8</u>	5
---	---	----------	---	---	----------	----------	---

*página 3*

2	E	G		-1	-1	-1	-1
---	---	---	--	----	----	----	----

# Propriedades da Árvore B

→ Uma árvore-B é  $n$ -ária

◆ Possui mais de 2 descendentes por nó (página)

# Propriedades da Árvore B

→ Numa árvore-B de ordem  $m$

◆ Cada página tem:

- No máximo  $m$  descendentes e  $m-1$  chaves
- No mínimo  $\lceil m/2 \rceil$  descendentes (exceto raiz e folhas)
  - Taxa de ocupação
- A raiz tem, no mínimo, 2 descendentes
  - A menos que seja uma folha



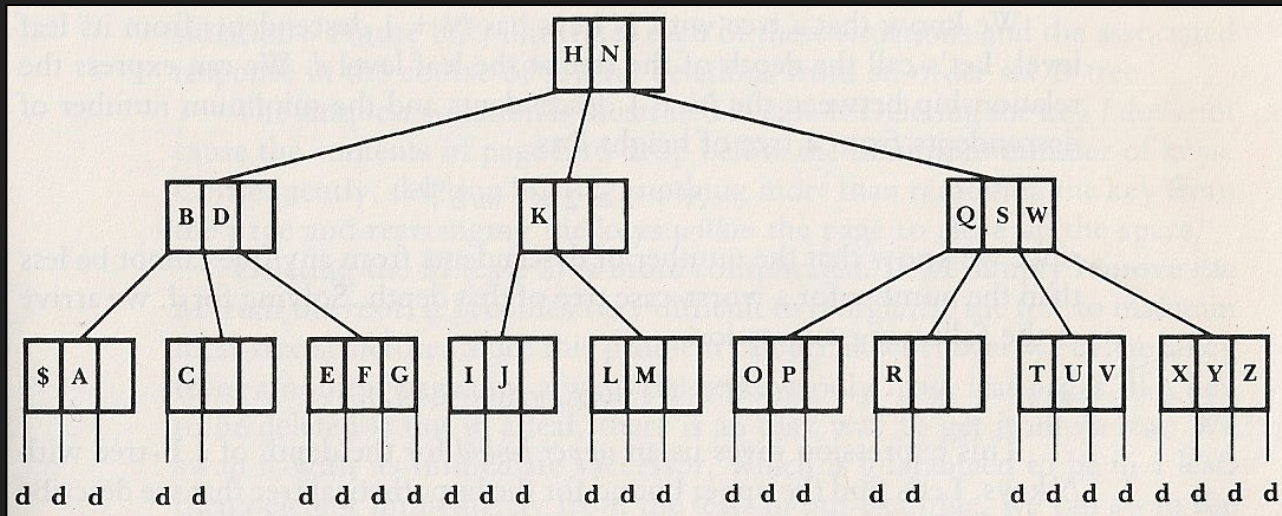
# Propriedades da Árvore B

- Todas as folhas estão no mesmo nível
- Uma página não folha com  $k$  descendentes contém  $k-1$  chaves
  - ◆  $\lceil m/2 \rceil \leq k \leq m$
- Uma página folha contém:
  - ◆ No mínimo  $\lceil m/2 \rceil - 1$  chaves
    - Taxa de ocupação
  - ◆ No máximo  $m-1$  chaves

# Profundidade da Árvore B

→ Número de descendentes possíveis de um nível  $d$  da árvore

→ = (número de chaves até o nível  $d$ ) + 1

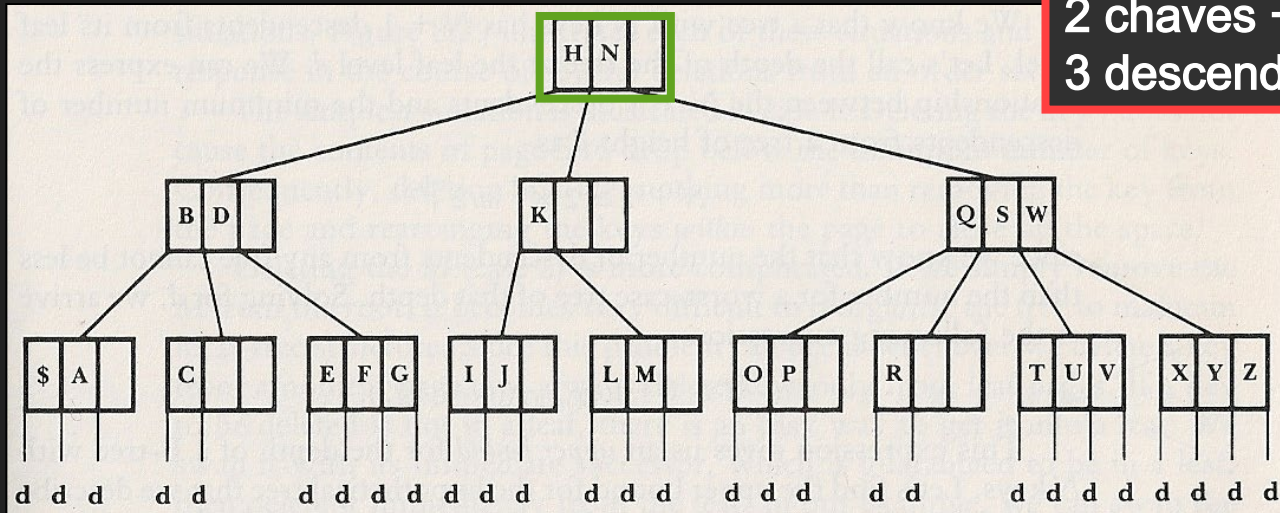


# Profundidade da Árvore B

→ Número de descendentes possíveis de um nível  $d$  da árvore

→ = (número de chaves até o nível  $d$ ) + 1

**Nível 1 ( $d = 1$ ):**  
2 chaves + 1 =  
3 descendentes

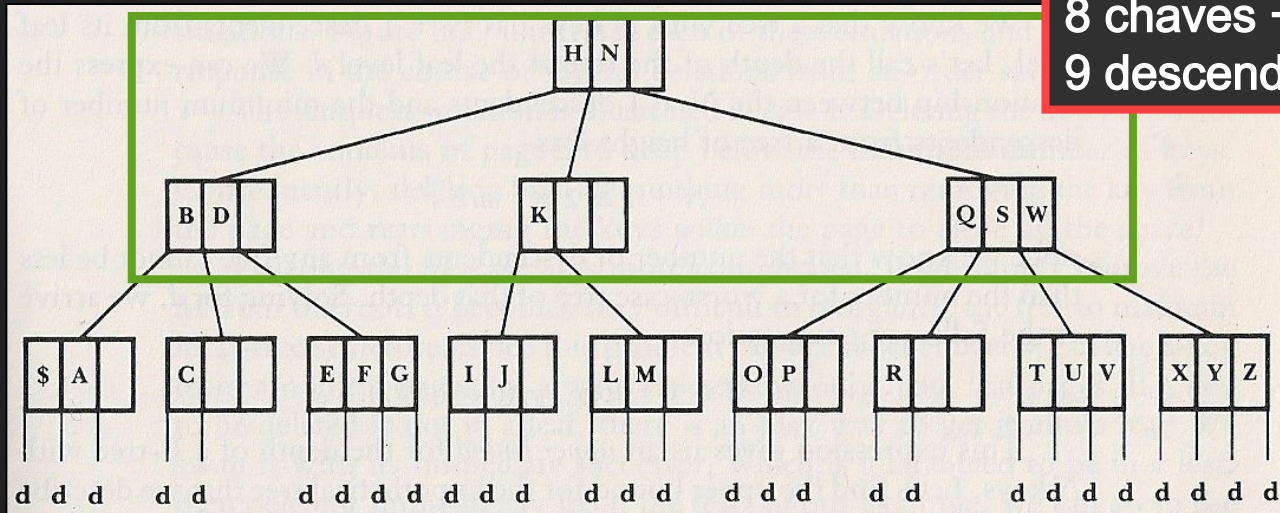


# Profundidade da Árvore B

→ Número de descendentes possíveis de um nível  $d$  da árvore

◆ = (número de chaves até o nível  $d$ ) + 1

Nível 2 ( $d = 2$ ):  
8 chaves + 1 =  
9 descendentes

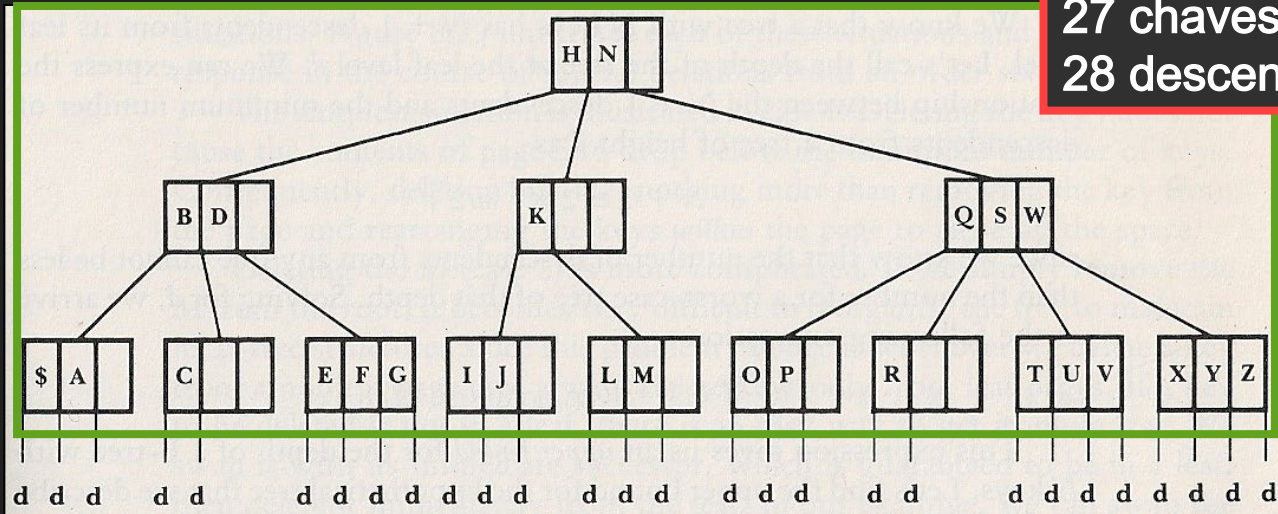


# Profundidade da Árvore B

→ Número de descendentes possíveis de um nível  $d$  da árvore

◆ = (número de chaves até o nível  $d$ ) + 1

Nível 3 ( $d = 3$ ):  
27 chaves + 1 =  
28 descendentes



# Profundidade da Árvore B

→ Consequência:

→ seja  $d$  o nível dos nós folha:  $d$  é a altura da árvore

→ em uma árvore com  $N$  chaves, vimos que o número de descendentes no nível  $d$  é igual a  $N+1$

→ Perguntas:

→ se uma árvore de ordem  $m$  tem  $N$  chaves, qual a altura dela?

→ e porque isso é relevante?

# Profundidade da Árvore B

- **Melhor caso** : todas as páginas da árvore estão com ocupação máxima ( $m - 1$  chaves,  $m$  descendentes), portanto ela tem a altura mínima necessária para acomodar  $N$  chaves
- **Pior caso** : todas as páginas da árvore estão com ocupação mínima ( $\lceil m/2 \rceil - 1$  chaves,  $\lceil m/2 \rceil$  descendentes), portanto ela tem a altura máxima necessária para acomodar  $N$  chaves

# Profundidade da Árvore B

→ Melhor caso:

- ◆ nro. descendentes nível 1 (raiz):  $m$
- ◆ nro. descendentes nível 2:  $m * m = m^2$
- ◆ nro. descendentes nível 3:  $m * m^2 = m^3$
- ◆ ...
- ◆ nro. descendentes nível  $d$ :  $m * m^{(d-1)} = m^d = N+1$



# Profundidade da Árvore B

→ Melhor caso:

→  $d$  = nível dos nós folha = altura da árvore

◆ Vimos que em uma árvore com  $N$  chaves, o número de descendentes no nível dos nós folha é igual a  $N+1$

→ Portanto,  $N + 1 = m^d$ , e  $d = \log_m(N+1)$

→ Essa é a altura da árvore no **melhor caso** (é a árvore com menor altura possível!)

# Profundidade da Árvore B

→ Mas, e o pior caso?

- ◆ Esse é o cenário quando todas as páginas estão com ocupação mínima
- ◆ Cada nó terá o mínimo de chaves e de descendentes
- ◆ Árvore terá a maior altura possível para esse número de chaves!

# Profundidade da Árvore B

→ Pior Caso

- ◆ Cada nó terá o mínimo de descendentes
- ◆ Árvore terá sua maior altura e menor largura
- ◆ Ocupação mínima por página:  $\lceil m/2 \rceil$  descendentes (exceto raiz e folhas)

# Profundidade da Árvore B

→ Para uma árvore de ordem  $m$

◆ Raiz (nível 1)

- 2 descendentes

◆ Nível 2

- 2 nós, tendo cada um  $\lceil m/2 \rceil$  descendentes
- $2 \times \lceil m/2 \rceil$  descendentes no segundo nível

# Profundidade da Árvore B

→ Para uma árvore de ordem  $m$

◆ Nível 3

- $2 \times \lceil m/2 \rceil_{\text{nós}} \times \lceil m/2 \rceil_{\text{descendentes}}$  para cada nó
- $2 \times \lceil m/2 \rceil^2$  descendentes

◆ O nível  $d$  terá ?

- $2 \times \lceil m/2 \rceil^{d-1}$  descendentes

# Profundidade da Árvore B

→ Número mínimo de descendentes para um nível  $d$  da árvore:

$$2 \times \lceil m/2 \rceil^{d-1}$$

→ Propriedade

◆ Número de descendentes de um nível  $d$  da árvore com  $N$  chaves até  $d$ :

$$N+1$$

# Profundidade da Árvore B

→ Considerando...

◆  $d$  = nível das folhas (= altura da árvore)

◆  $m$  = ordem da árvore

◆  $N$  = número de chaves até o nível das folhas (N é o total de chaves armazenadas)

→ Calcula-se a profundidade da árvore no **pio**r caso :

$$(N + 1) = 2 \times \lceil m/2 \rceil^{d-1} \Rightarrow d = 1 + \log_{\lceil m/2 \rceil} ((N+1)/2)$$

$$\log_m(N+1) \leq d \leq 1 + \log_{m/2}((N+1)/2)$$

a altura da árvore varia entre esses 2 limites

ela define o número máximo de acessos a disco necessários para localizar qualquer chave!



## Próximas Aulas...

### → Principais operações

- ◆ Inserção de chaves
- ◆ Remoção de chaves
- ◆ Pesquisa (Busca)

### → Principais variações

- ◆ Árvore-B\*
- ◆ Árvore-B+



# Referências

- M. J. Folk and B. Zoellick, File Structures: A Conceptual Toolkit, Addison Wesley, 1987.