



Árvores-B (Parte I)

SCC-503 Algoritmos e Estruturas de Dados II

Thiago A. S. Pardo

Leandro C. Cintra

M.C.F. de Oliveira



Problema

- Cenário até então
 - 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 100.000 itens podem requerer até 17 acessos

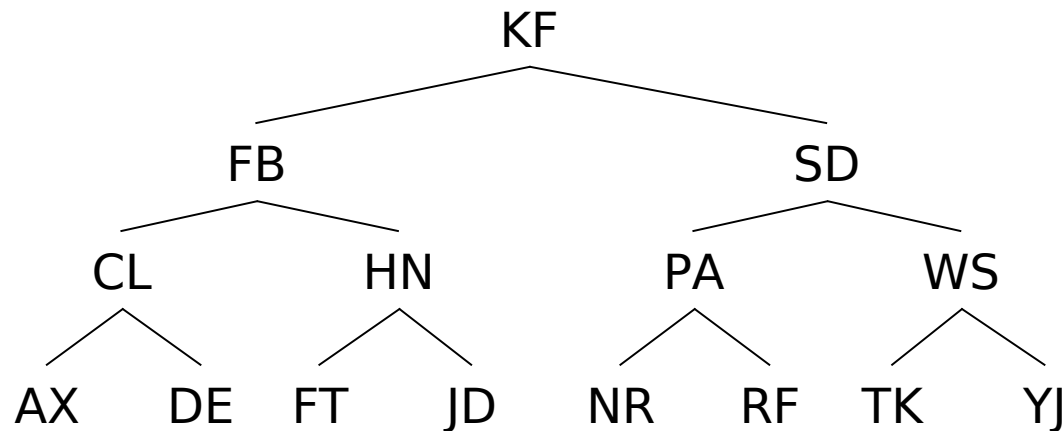


Problema

- Cenário
 - Manter em disco um índice ordenado para busca binária tem custo proibitivo
 - Necessidade de **método** com inserção e eliminação com apenas efeitos locais, isto é, que **não exija a reorganização total do índice**

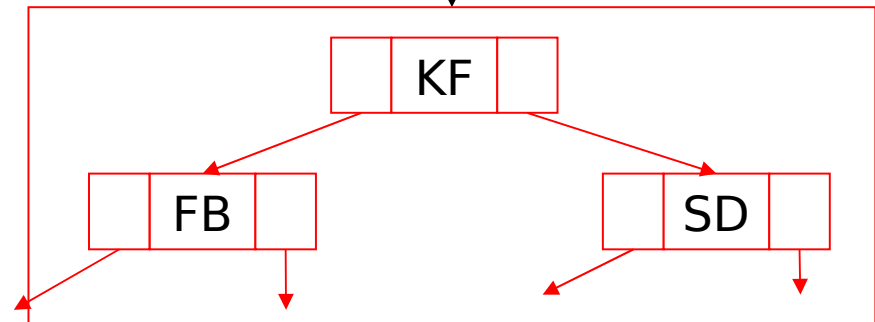
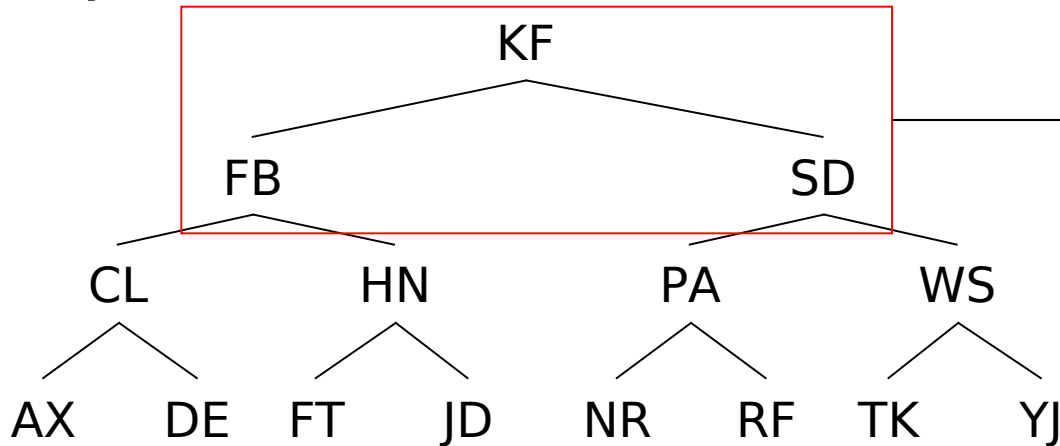
Solução: árvores binárias de busca?

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

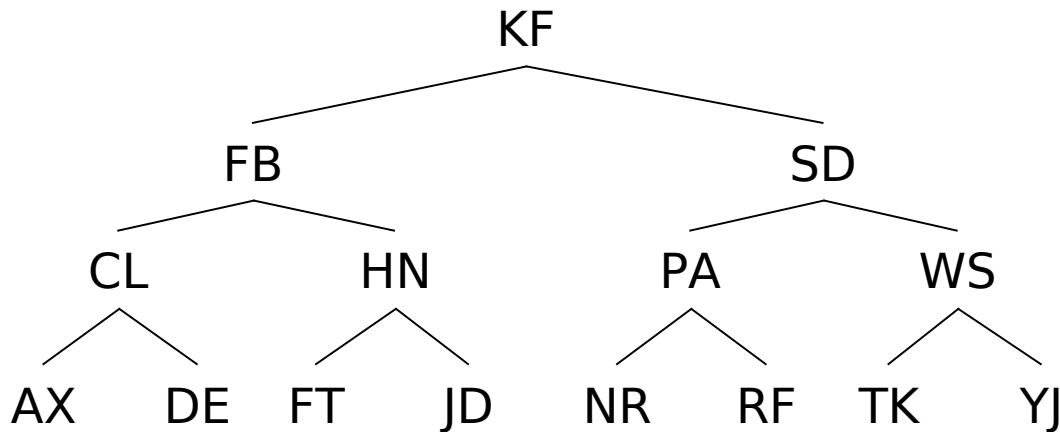


Vetor ordenado e representação por árvore binária

Solução: árvores binárias de busca



Representação da árvore no arquivo



Registros são mantidos em **arquivo**, e **ponteiros** (**esq** e **dir**) indicam onde estão os registros filhos

Chave Filho esq. Filho dir.

	Chave	Filho esq.	Filho dir.
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		
12	DE		
13	WS	14	5
14	TK		

raiz → 9



Vantagens

- Quais são as vantagens de se utilizar ABBs?



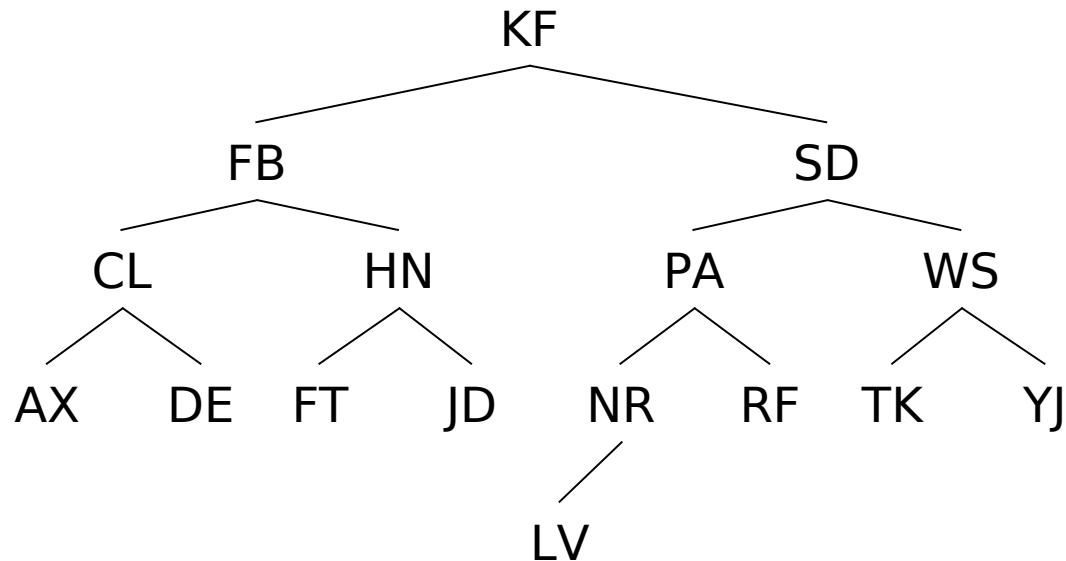
Vantagens

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



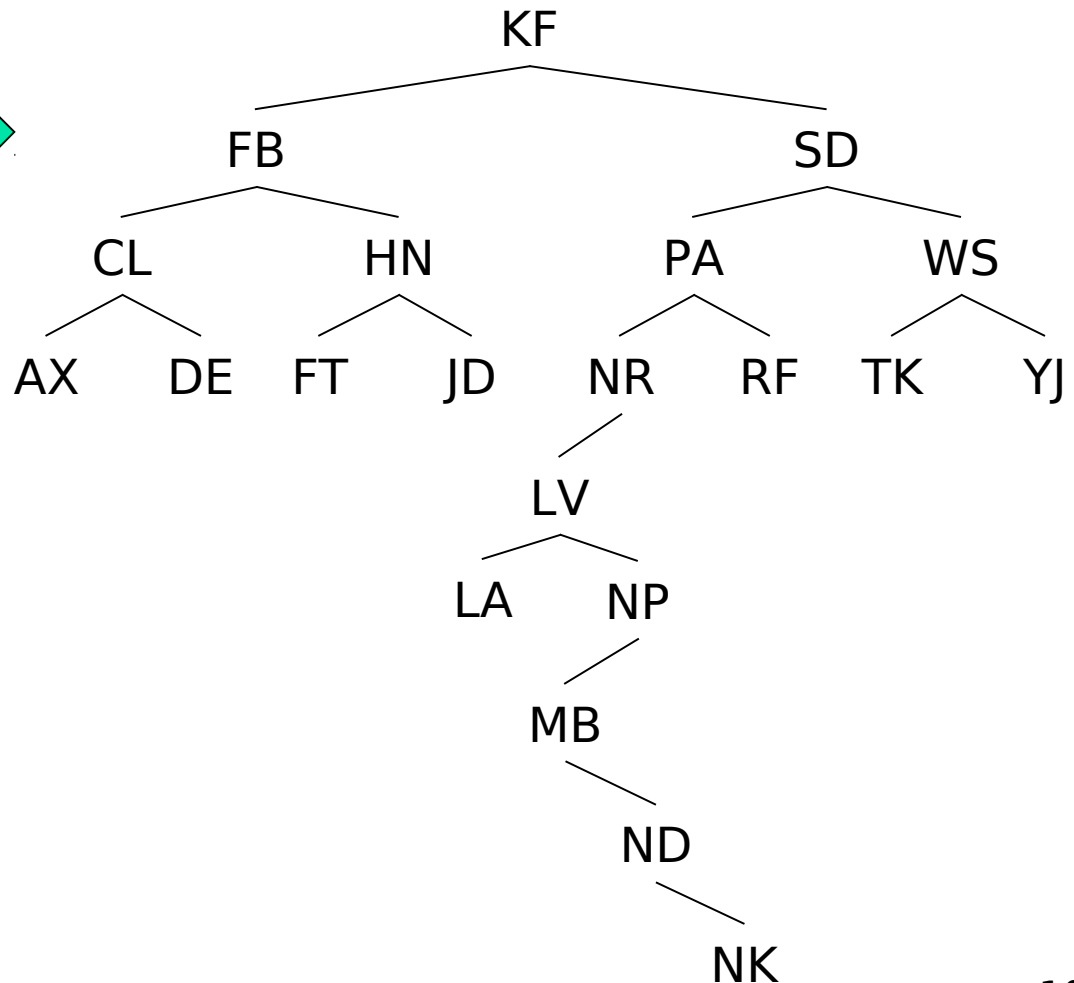
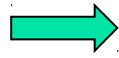
Inserção de chave

Inserção da chave LV



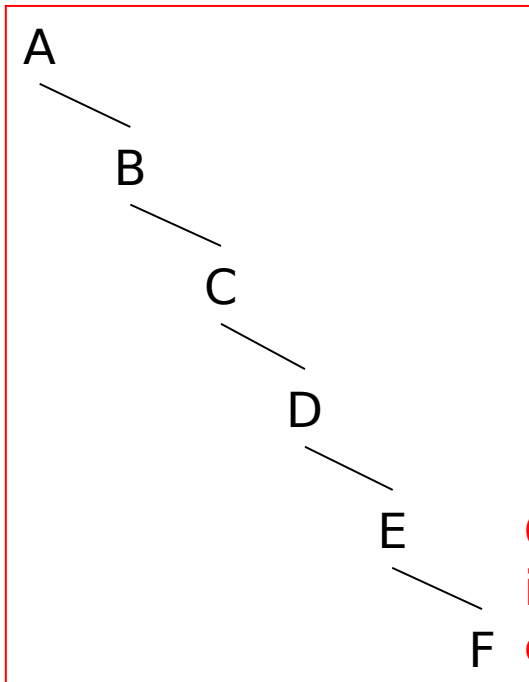
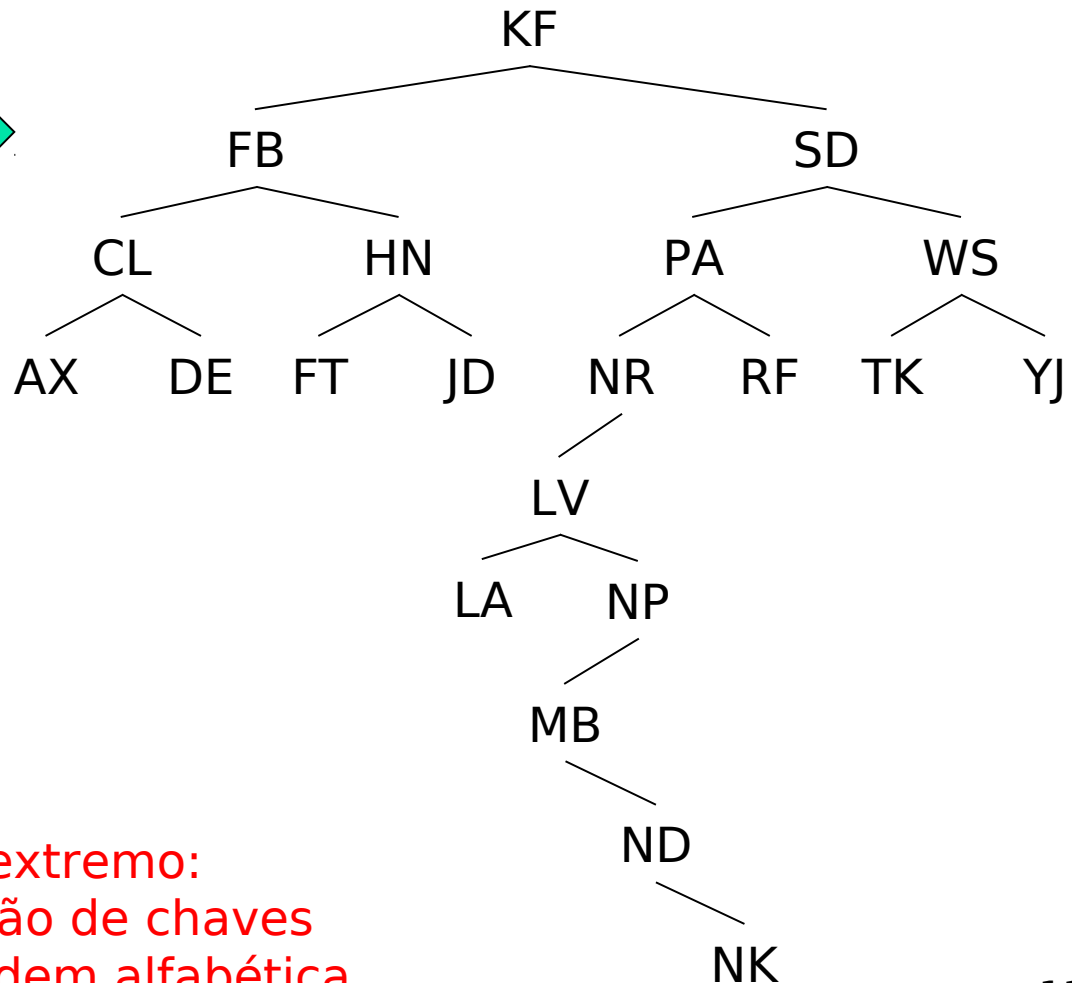
Problema: desbalanceamento

Inserção das chaves
NP, MB, TM, LA, UF,
ND, TS e NK



Problema: desbalanceamento

Inserção das chaves
NP, MB, TM, LA, UF,
ND, TS e NK

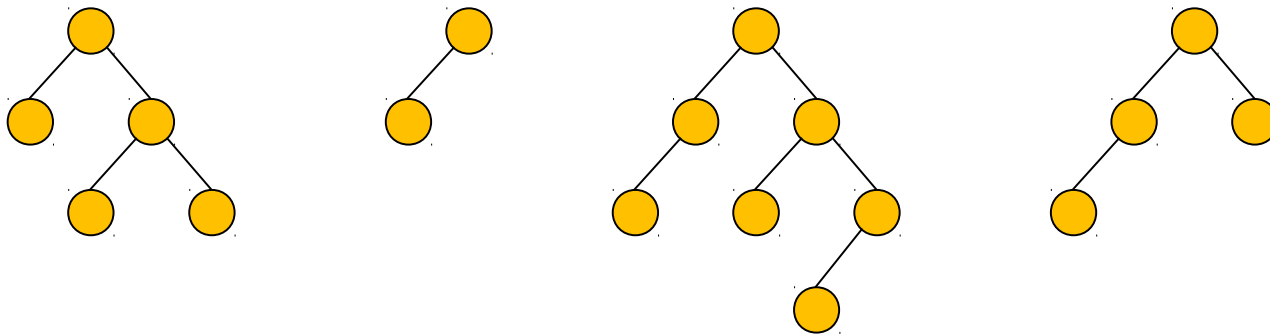


Caso extremo:
inserção de chaves
em ordem alfabética



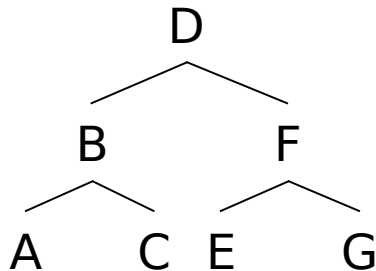
Solução por árvores-AVL

- **Diferença limitada entre níveis**
 - Garante performance aproximada de uma árvore completamente balanceada
 - Tradicionalmente, 1 nível de diferença
 - Procedimentos específicos de inserção e remoção
 - Manutenção feita por 4 tipos de rotação diferentes

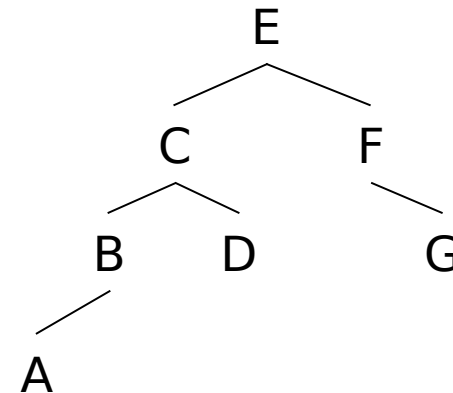


Árvores binárias perfeitamente balanceadas e AVL

Chaves de entrada: B C G E F D A



Árvore perfeitamente balanceada



AVL
→ aceitável



Solução por árvores-AVL

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



Solução por árvores-AVL

- **Problema**

- Se **chaves** em memória secundária, ainda há **muitos acessos!**
 - 20 ou 28 *seeks* ainda é muito para disco

- Até agora...

- Árvores binárias de busca **dispensam ordenação dos registros**
- Mas **número excessivo de acessos**



Solução por Árvores Binárias Paginadas (*Paged Binary Trees*)

■ **Paginação**

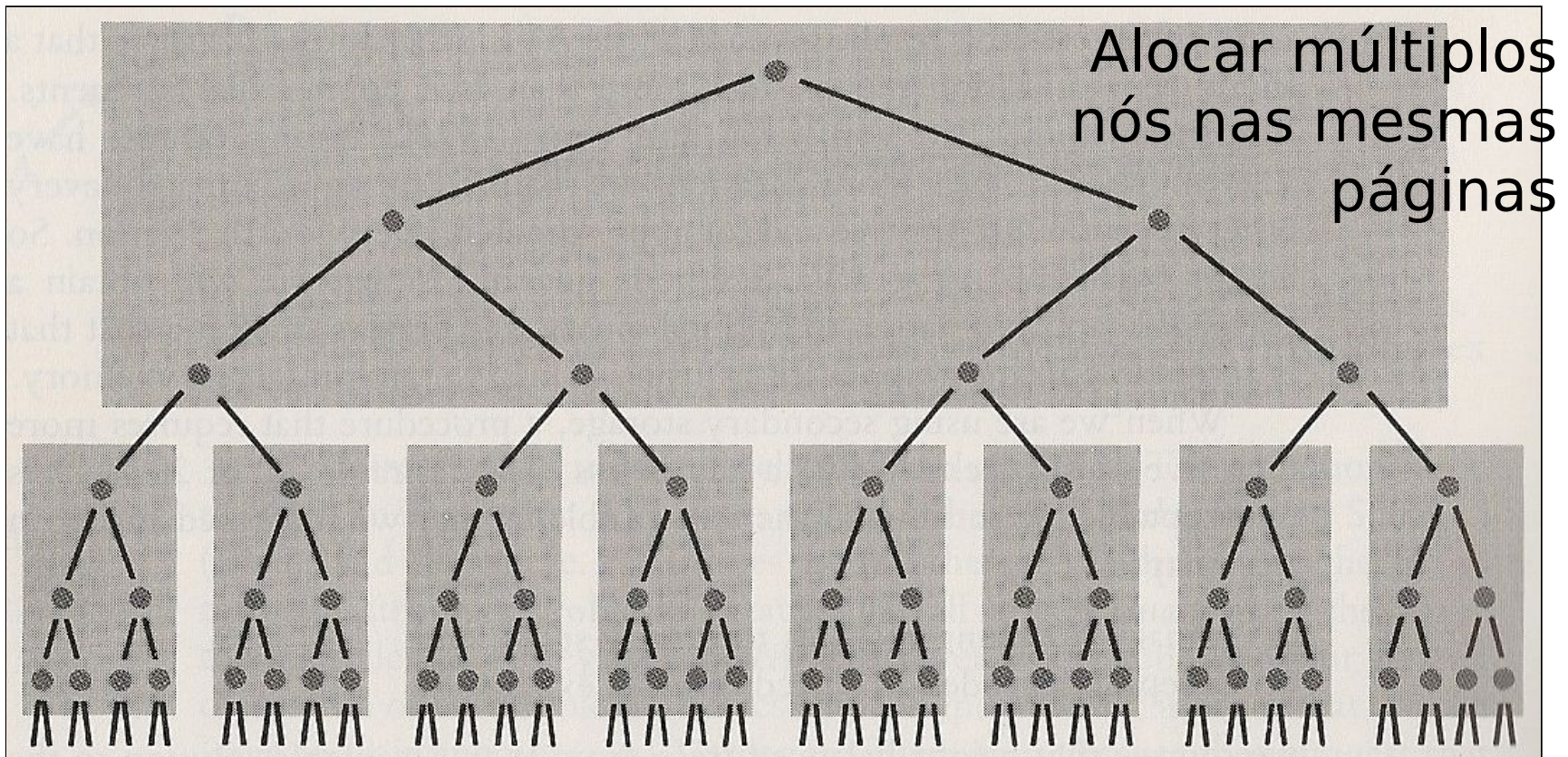
- A busca (*seek*) por uma posição específica do disco é muito lenta
- Mas, uma vez na posição, pode se ler uma grande quantidade de registros seqüencialmente a um custo relativamente pequeno



Solução por Árvores Binárias Paginadas (*Paged Binary Trees*)

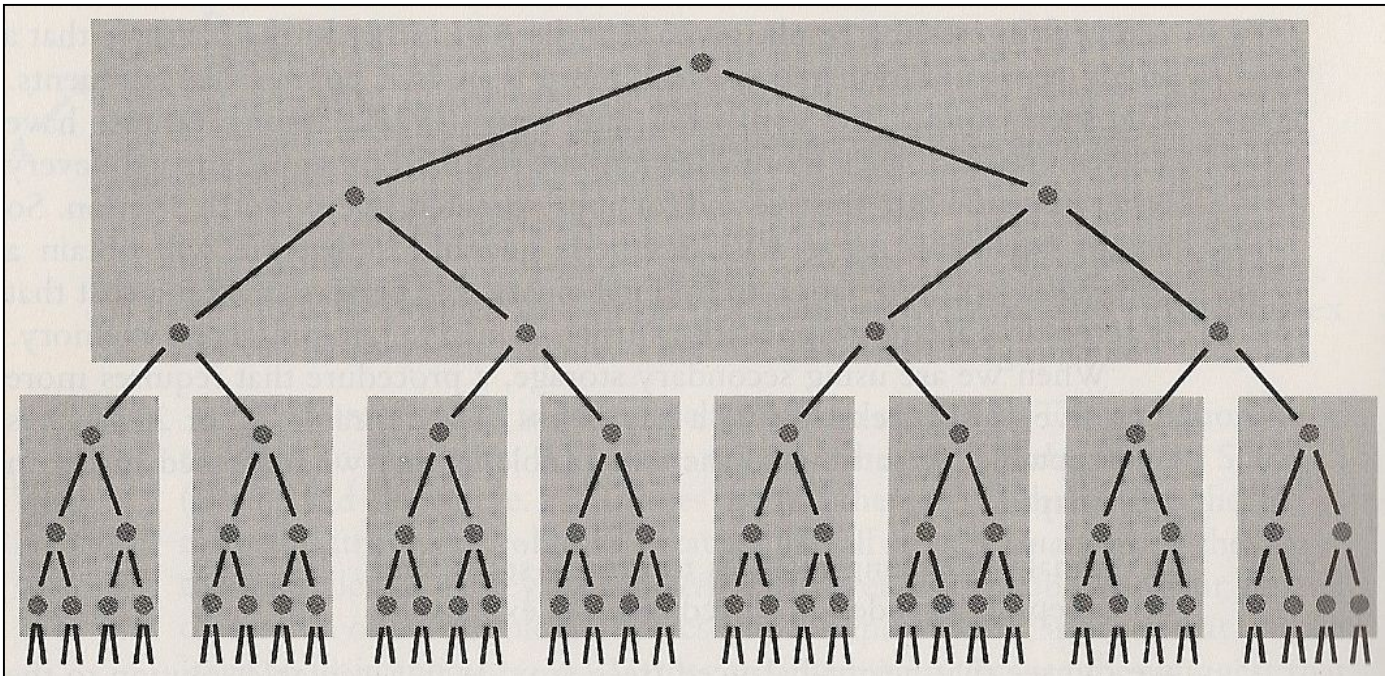
- Noção de **página** em *sistemas paginados*
 - Feito um seek, todos os registros de uma mesma "página" do arquivo (p.ex. 2 KB de um setor) são lidos
 - Esta página pode conter um número grande de registros
 - Se o próximo registro a ser recuperado estiver na mesma página já lida, evita-se novo acesso
 - Em ABB ou AVL quais informações temos ao recuperar um nó, e quais estão disponíveis na "página" lida?

Solução por Árvores Binárias Paginadas (*Paged Binary Trees*)



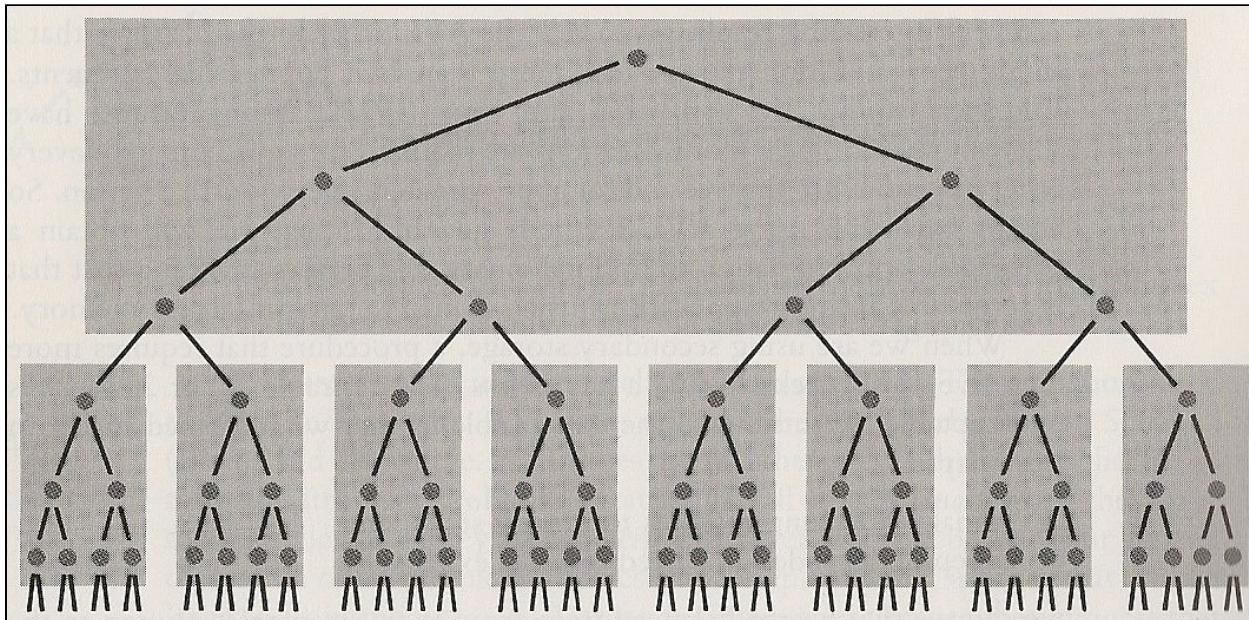
Solução por Árvores Binárias Paginadas (*Paged Binary Trees*)

- Cada página aloca 7 nós e permite acesso a 8 páginas
- Assim, qualquer um dos 63 registros (9x7 nós) pode ser acessado em, no máximo, 2 acessos



Solução por Árvores Binárias Paginadas (*Paged Binary Trees*)

- 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 (contra 9 de uma AVL)





Eficiência da árvore paginada

- Supondo que
 - Cada página de uma árvore ocupa **4KB** e armazena **511 pares chave/referencia**
 - Cada página contém uma árvore completa perfeitamente balanceada
 - Uma árvore de 3 níveis pode armazenar **134.217.727 chaves**
 - Encontra-se qualquer uma das chaves com no máximo 3 seeks



Eficiência da árvore

- **Pior caso da busca**

- **ABB completa, perfeitamente balanceada:** $\log_2 (N+1)$
- **Versão paginada:** $\log_{k+1} (N+1)$

onde **N** é o número total de chaves, e **k** é o número de chaves armazenadas em uma página

- Note que, na ABB tradicional, base do \log_2 nada mais é do que 1 chave por página + 1
- **Exemplo**
 - ABB: $\log_2 (134.217.727) = 27$ acessos
 - Versão paginada: $\log_{511+1} (134.217.727) = 3$ acessos



Solução por Árvores Binárias Paginadas (*Paged Binary Trees*)

- Preços a pagar?



Solução por Árvores Binárias Paginadas (*Paged Binary Trees*)

- **Preços a pagar**

- Maior **tempo na transmissão** de dados
- Necessário **manter a organização da árvore**



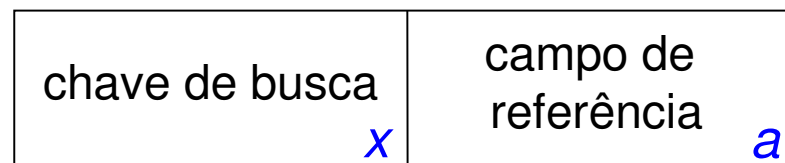
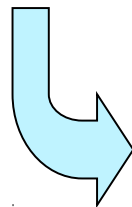
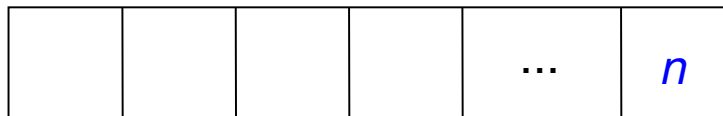
A invenção das árvores-B

- **Árvores-B: generalização** de uma ABB paginada
 - Não binárias, com conteúdo de uma página não mantido como árvore
- 1960s: competição entre fabricantes e pesquisadores
- **1972**: Bayer and McGreight (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
 - De onde vem o “B” do nome?



Características Gerais

- Organizar e manter um índice para um arquivo de acesso aleatório altamente dinâmico
- Índice
 - n elementos (x,a) de tamanho fixo





Características Gerais

- Índice
 - extremamente volumoso
- *Pool de buffers é pequeno*
 - apenas uma parcela do índice pode ser carregada em memória principal
 - operações baseadas em disco

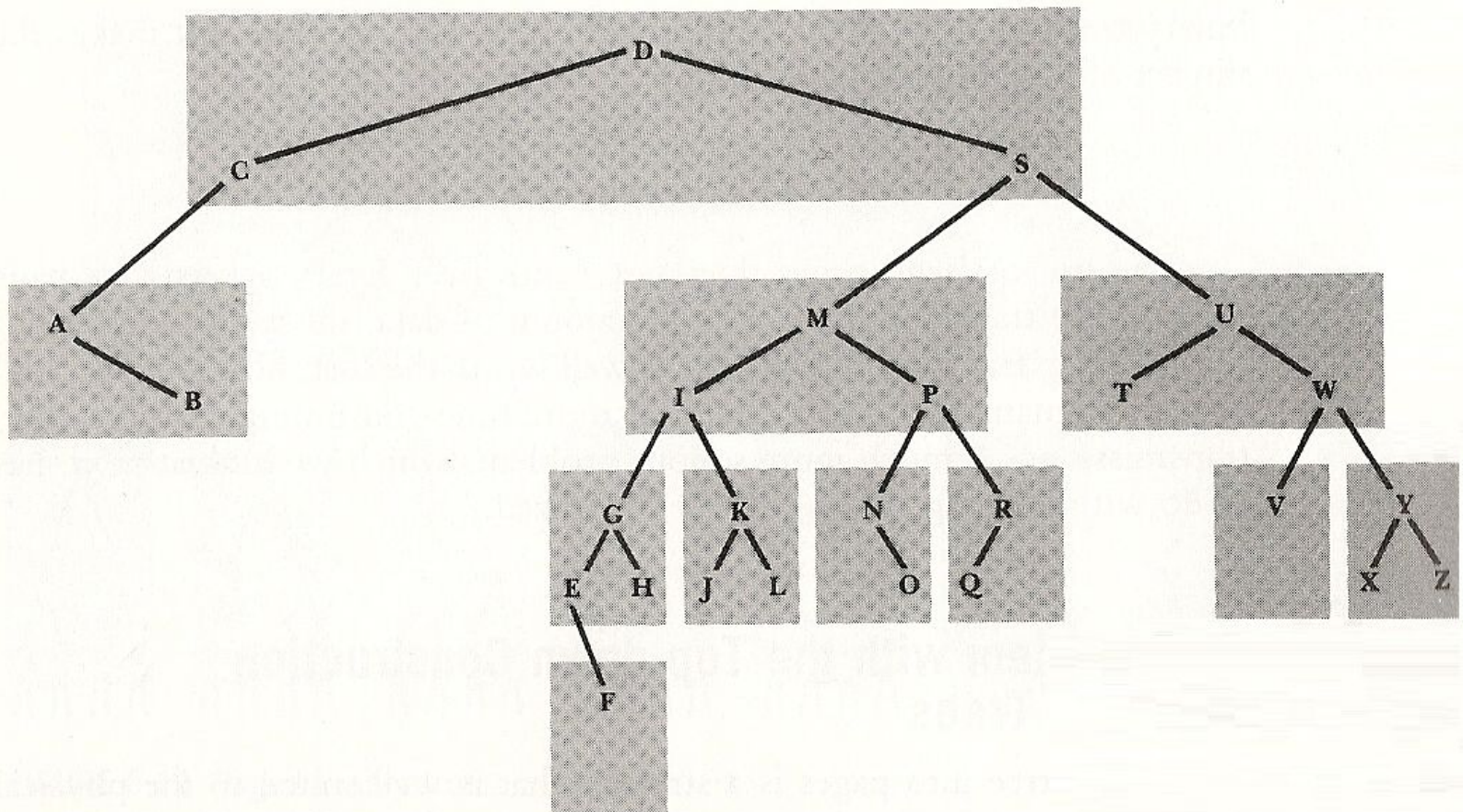


Construção *Top-Down* de árvores paginadas

- Se conjunto de chaves é conhecido, construção da árvore é simples
 - Inicia-se pela **chave do meio** para obter uma árvore balanceada
- Porém, é **complicado** se as chaves são recebidas em uma **seqüência aleatória**

Construção *Top-Down* de árvores paginadas

- Ordem: 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





Construção *Top-Down* de árvores paginadas

- Figura anterior: a construção foi feita *top-down*, a partir da raiz
 - Quando uma chave é inserida, a árvore dentro da página pode sofrer *rotações* para manter o balanceamento
 - Construção a partir da raiz implica em que as *chaves iniciais tendem a ficar na raiz*
 - **C** e **D** não deveriam estar no topo, pois acabam desbalanceando a árvore de forma definitiva



Construção *Top-Down* de árvores paginadas

■ Questões

- Como garantir que as **chaves** na página raiz são **boas separadoras**, i.e., dividem o conjunto de chaves de maneira balanceada?
- Como **impedir o agrupamento de chaves** que não deveriam estar na mesma página (como C, D e S, por exemplo)?
- Como garantir que cada página contenha um **número mínimo de chaves**?



Árvore-B

- Características

- balanceada
- *bottom-up* para a criação (em disco)
 - nós folhas → nó raiz

- Inovação

- Não é necessário construir a árvore a partir do nó raiz, como é feito para árvores em memória principal e para as árvores anteriores



Construção *Bottom-Up*

- Conseqüências

- Chaves indevidas não são mais alocadas na raiz
 - Elimina as questões em aberto de *chaves separadoras* e de *chaves extremas*
- Não é necessário tratar o problema de desbalanceamento

na árvore-B, as chaves na raiz da árvore emergem naturalmente



Características

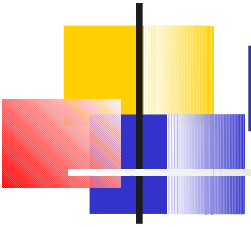
- Conteúdo de cada nó (página de disco)
 - Seqüência ordenada de chaves
 - Conjunto de ponteiros
 - Número de ponteiros = número de chaves + 1

Não há uma árvore explícita dentro de uma página

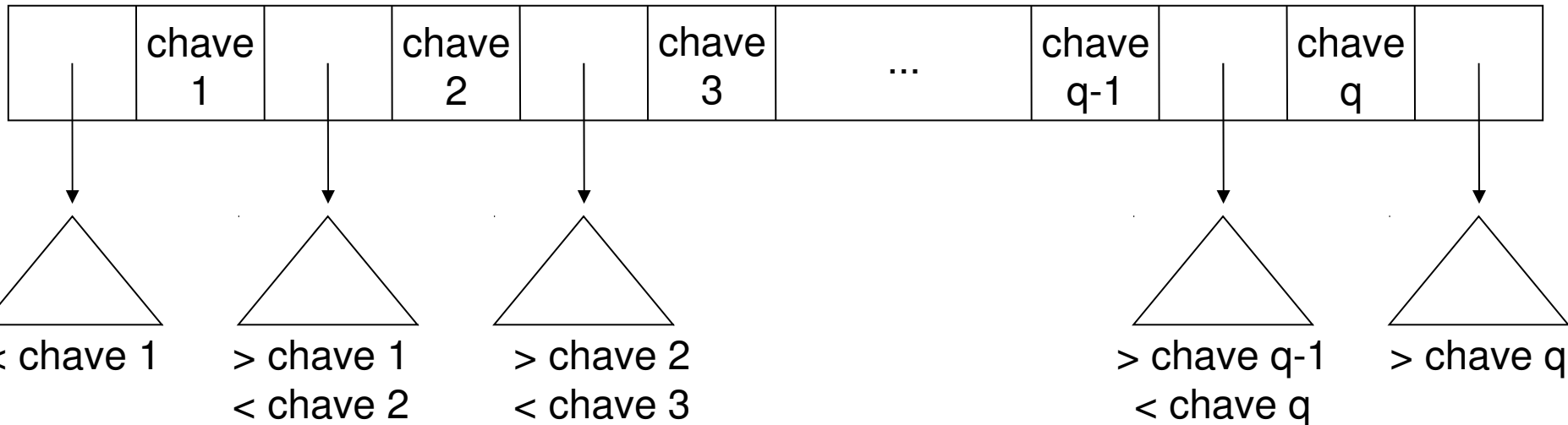


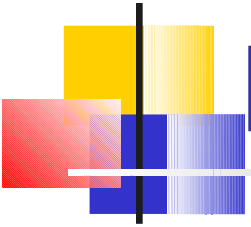
Propriedades das árvores-B

- Uma árvore-B é n -ária pois possui **mais** de 2 descendentes por nó (**página**). Numa árvore-B de ordem **m**
 1. Cada página tem:
 - no máximo, **m** descendentes e **m-1** registros
 - no mínimo $\lceil m/2 \rceil$ descendentes (exceto raiz e folhas)
 2. A raiz tem, no mínimo, dois descendentes
 - a menos que seja uma folha
 3. Todas as folhas estão no mesmo nível
 4. Uma página não folha com **k** descendentes contém **k-1** chaves
 5. Uma página folha contém, no mínimo $\lceil m/2 \rceil - 1$ e, no máximo, **m-1** chaves

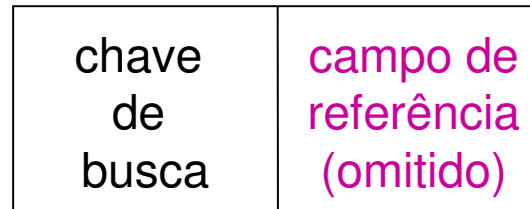
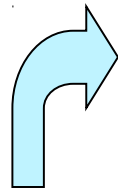


Estrutura Lógica de um Nó

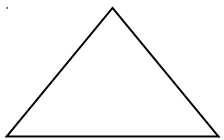
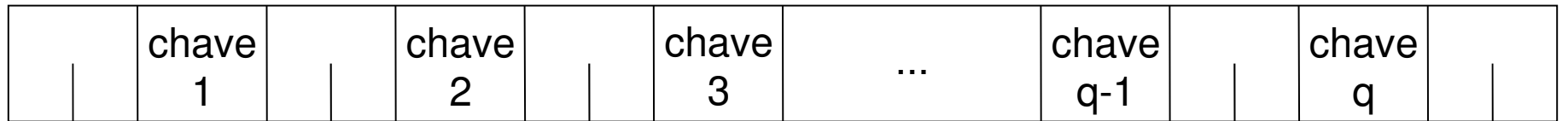




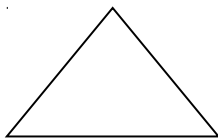
Estrutura Lógica de um Nó



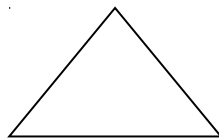
campos de tamanho fixo, em princípio



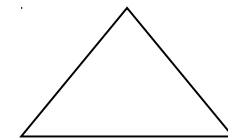
< chave 1



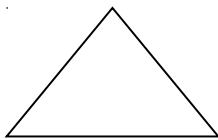
> chave 1
< chave 2



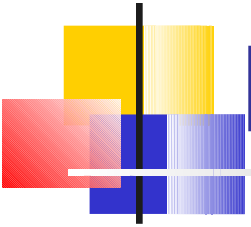
> chave 2
< chave 3



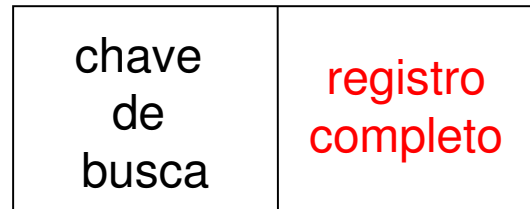
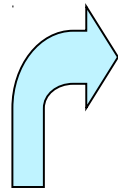
> chave q-1
< chave q



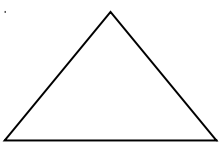
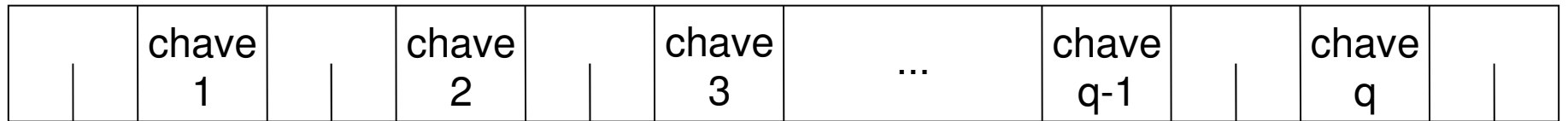
> chave q



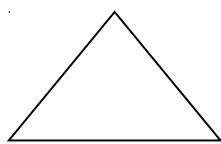
Estrutura Lógica de um Nó



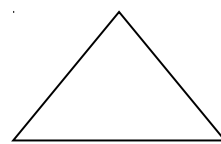
campos de tamanho fixo, em princípio



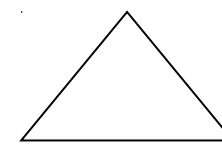
< chave 1



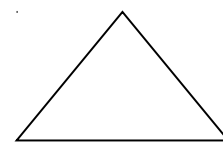
> chave 1
< chave 2



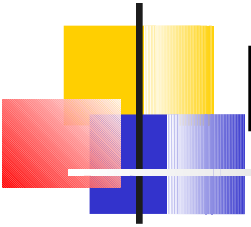
> chave 2
< chave 3



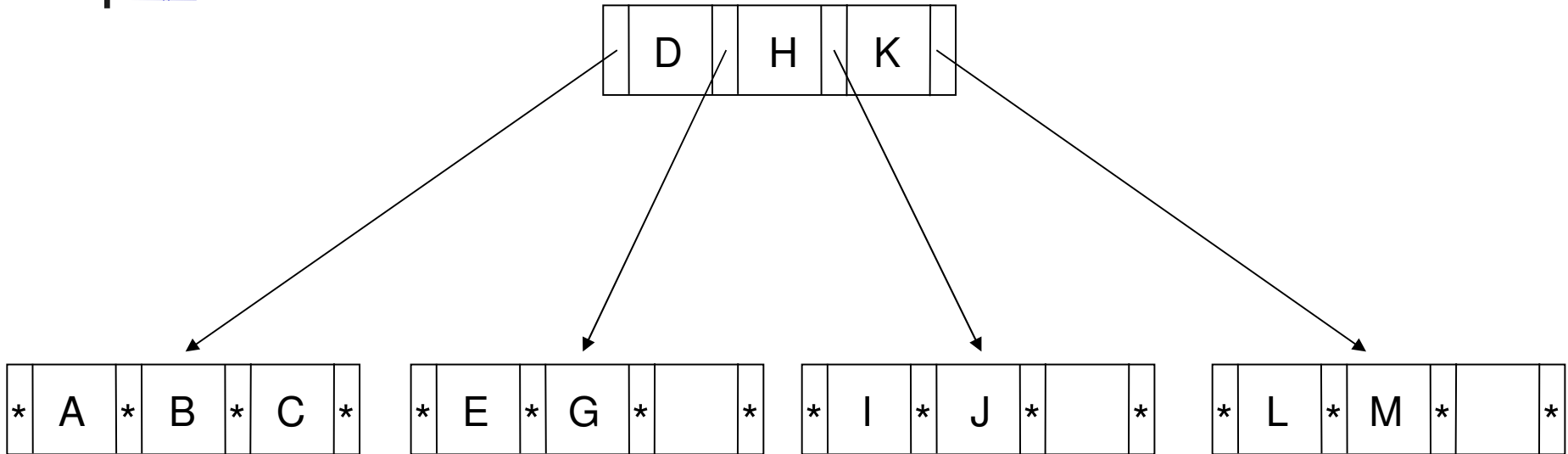
> chave q-1
< chave q



> chave q



Exemplo





Características

■ Ordem

- Número máximo de ponteiros que pode ser armazenado em um nó
- Exemplo: árvore-B de ordem 8
 - máximo de 7 chaves e 8 ponteiros

■ Observações

- Número máximo de ponteiros é igual ao número máximo de descendentes de um nó
- Nós folhas não possuem filhos, e seus ponteiros são nulos



Estrutura típica de um nó

```
const m = 2;    // ordem da arvore-B
typedef struct node_Btree Btree;
struct node_Btree {
    int num_keys;    // numero de chaves armazenadas
    char keys[2*m-1]; // vetor de chaves
    Btree *desc[2*m]; // ponteiros para os descendentes
    bool leaf;    // flag folha da arvore
};
```



Inserção de Dados (Chave)

- Característica
 - Sempre realizada nos **nós folha**
- Situações a serem analisadas
 1. árvore vazia
 2. *overflow* no nó raiz
 3. inserção em nós folha



Inserção em árvore vazia



Inserção: situação inicial

- Criação e preenchimento do nó
 - primeira chave: criação do nó raiz
 - demais chaves: inserção até a capacidade limite do nó

- Exemplo
 - nó com capacidade para 7 chaves → ordem 8
 - chaves: letras do alfabeto
 - situação inicial: árvore vazia



Inserção: situação inicial

- Chaves B C G E F D A
 - inseridas desordenadamente
 - mantidas ordenadas no nó
- Ponteiros (*)
 - nós folhas: -1 ou fim de lista (NULL)
 - nós internos: referência para o nó filho ou -1
- Nó raiz (= nó folha nesse momento)

*	A	*	B	*	C	*	D	*	E	*	F	*	G	*
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Overflow no nó raiz



Inserção: *overflow* nó raiz

- **Passo 1** – particionamento do nó (*split*)
 - nó original → nó original + novo nó
 - *split* 1-to-2
 - as chaves são distribuídas uniformemente nos dois nós
 - considerando chaves do nó original + nova chave

- Exemplo: inserção de **J** em
– particionamento da página

*	A	*	B	*	C	*	D	*	E	*	F	*	G	*
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

*	A	*	B	*	C	*	D	*		*		*		*
---	---	---	---	---	---	---	---	---	--	---	--	---	--	---

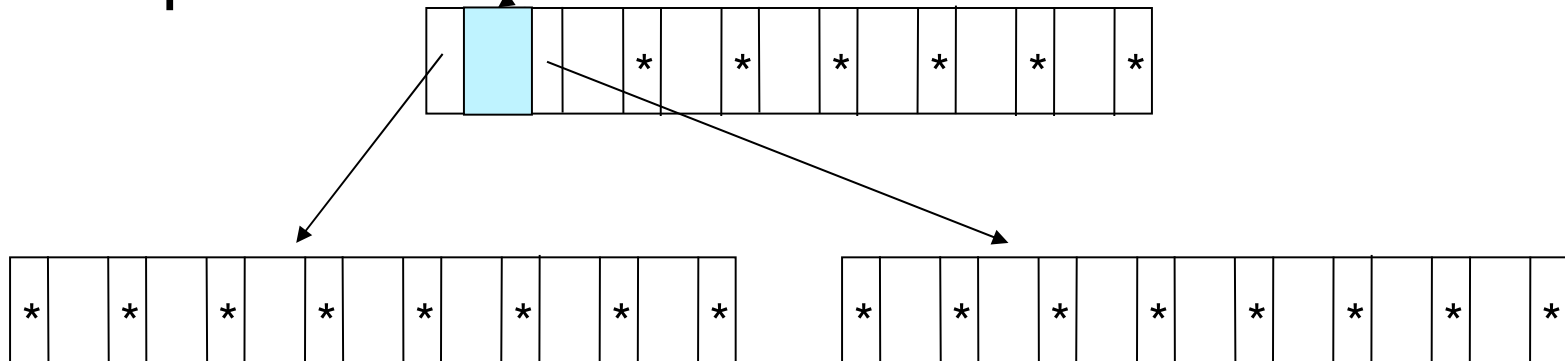
*	E	*	F	*	G	*	J	*		*		*		*
---	---	---	---	---	---	---	---	---	--	---	--	---	--	---

Inserção: *overflow* nó raiz

- **Passo 2** - criação de uma **nova raiz**
 - a existência de um nível mais alto na árvore permite a escolha das folhas durante a pesquisa

nova raiz será construída com 1 elemento
- qual elemento deve ser incluído?
- esse será a chave separadora.

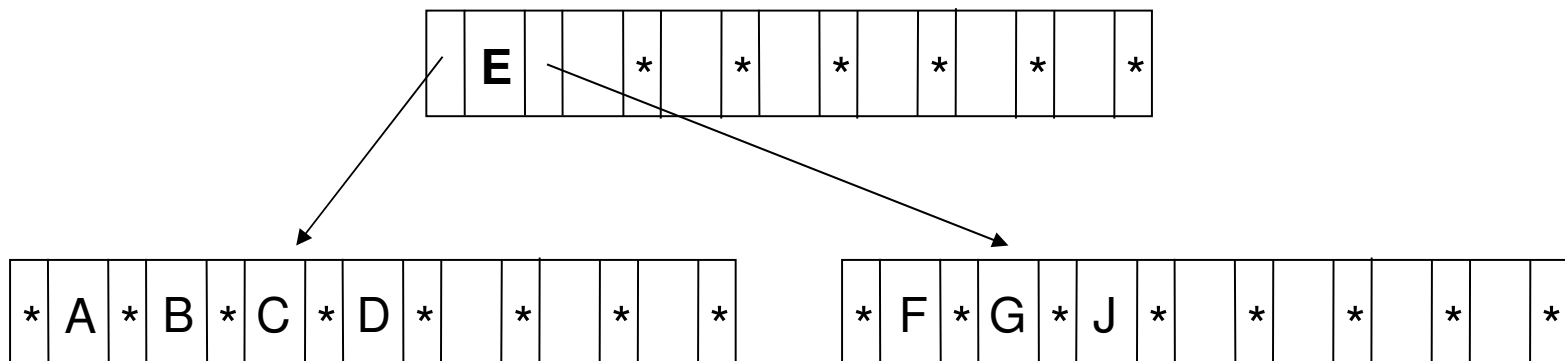
■ Esquema

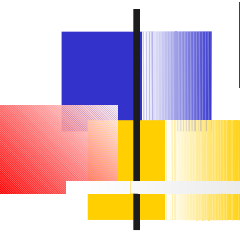


Inserção: *overflow* nó raiz

- **Passo 3** – promoção de chave (*promotion*)
 - a primeira chave do novo nó após particionamento é promovida para o nó raiz

■ Exemplo





Inserção em nós folha



Inserção: nós folhas

- **Passo 1** – pesquisa
 - a árvore é percorrida até encontrar o nó folha no qual a nova chave será inserida
- **Passo 2** – inserção em nó com espaço
 - ordenação da chave após a inserção
 - alteração dos valores dos campos de referência

nó folha em
memória principal



Inserção: nós folhas

- **Passo 2** – inserção em nó cheio
 - particionamento
 - criação de um novo nó
(nó original → nó original + novo nó)
 - distribuição uniforme das chaves nos dois nós
 - promoção
 - escolha da primeira chave do novo nó como chave separadora no nó pai (nó por onde a pesquisa passou antes)
 - ajuste do nó pai para apontar para o novo nó
 - propagação de *overflow*




Exemplo

- Insira as seguintes chaves em um índice árvore-B
 - C S D T A M P I B W N G U K
- Ordem da árvore-B: 4
 - em cada nó (página de disco)
 - número de chaves: 3
 - número de ponteiros: 4

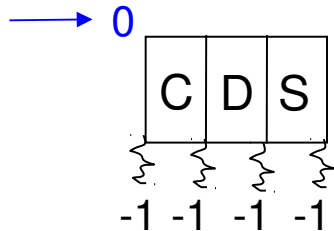


C S D T A M P I B W N G U

- Passo 1 – inserção de C, S, D
 - criação do nó raiz

- 
- C
 - C S
 - C D S

RRN da
página

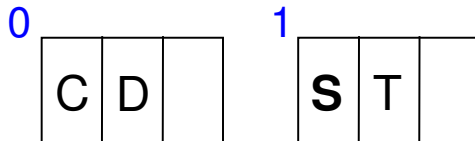




C S D T A M P I B W N G U

- Passo 2 – inserção de T
 - nó raiz cheio

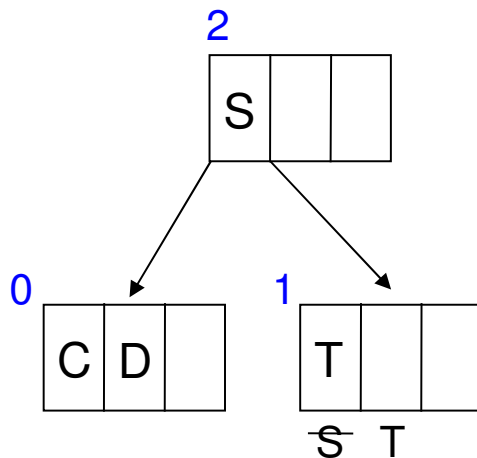
- particionamento do nó
- criação de uma nova raiz
- promoção de S





C S D T A M P I B W N G U

- Passo 2 – inserção de T
 - nó raiz cheio

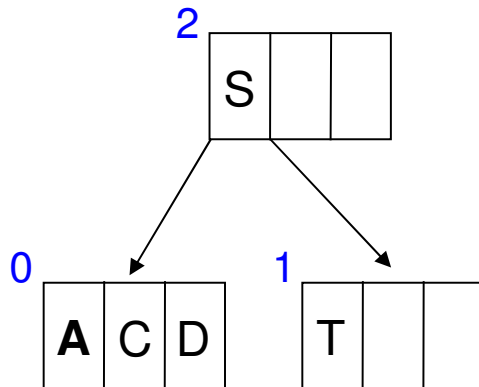


- particionamento do nó
- criação de uma nova raiz
- promoção de S



C S D T A M P I B W N G U

- Passo 3 – inserção de A
 - nó folha com espaço

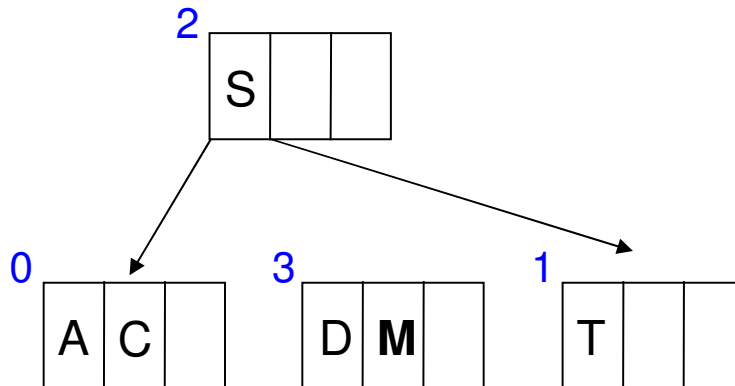




C S D T A M P I B W N G U

- Passo 4 – inserção de M
 - nó folha 0 cheio

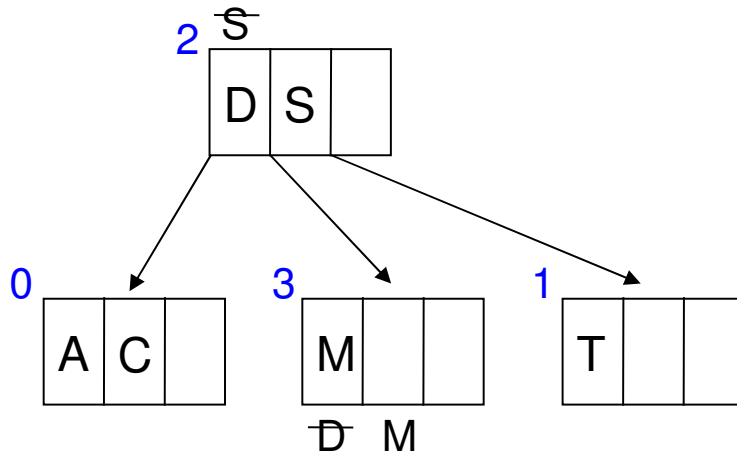
• particionamento do nó



C S D T A M P I B W N G U

- Passo 4 – inserção de M
 - nó folha 0 cheio

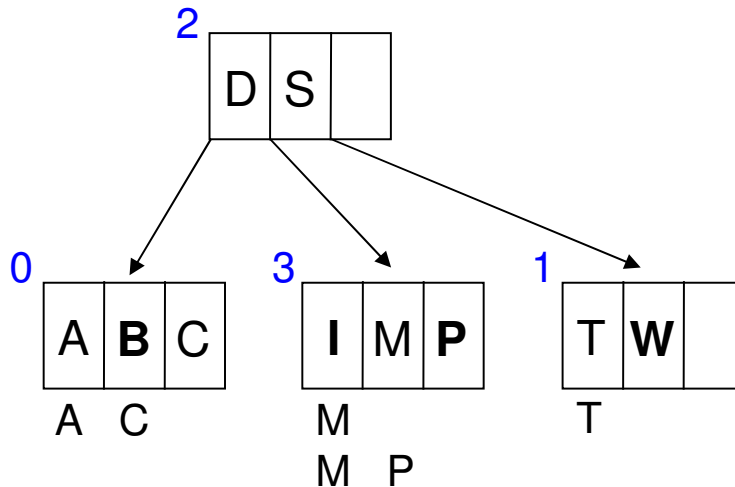
- particionamento do nó
- promoção de D





C S D T A M P I B W N G U

- Passo 5 – inserção de P, I, B, W
 - nós folhas com espaço

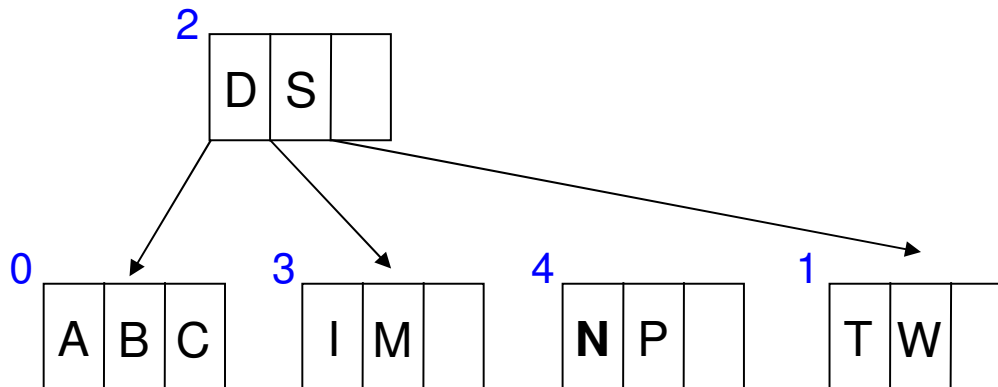




C S D T A M P I B W N G U

- Passo 6 – inserção de N
 - nó folha 3 cheio

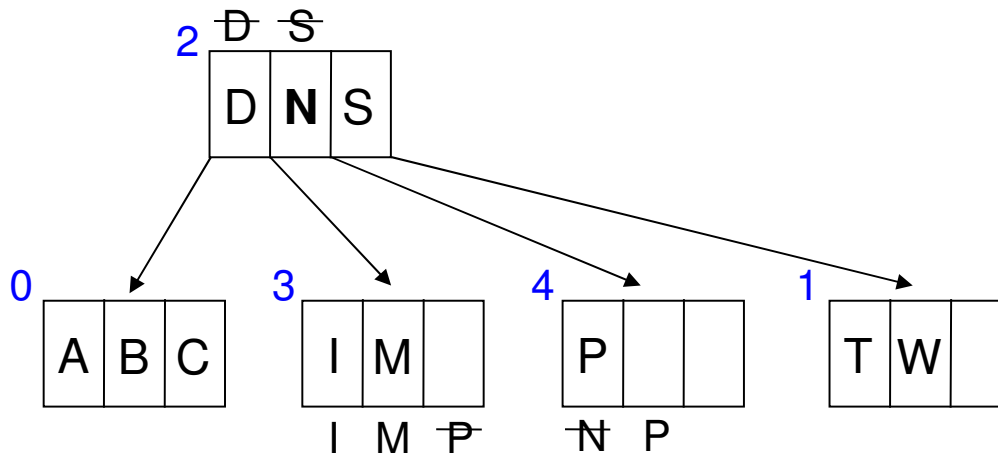
• particionamento do nó



C S D T A M P I B W N G U

- Passo 6 – inserção de N
 - nó folha 3 cheio

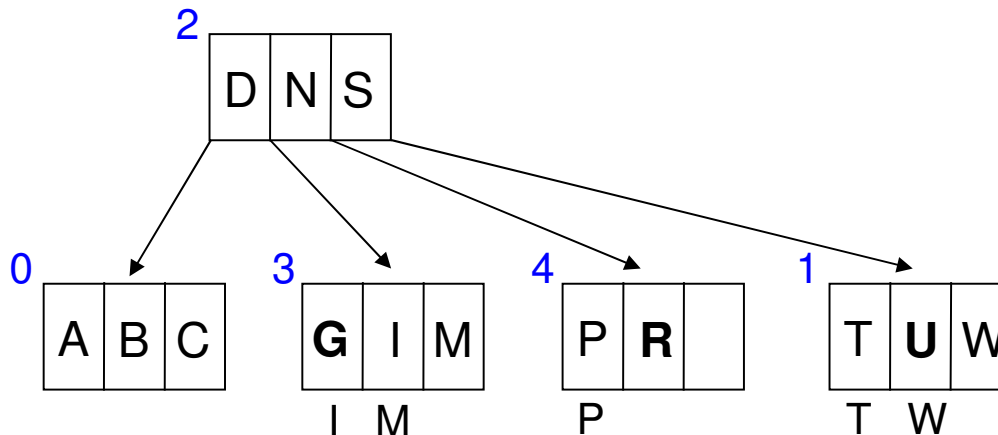
- particionamento do nó
- promoção de N



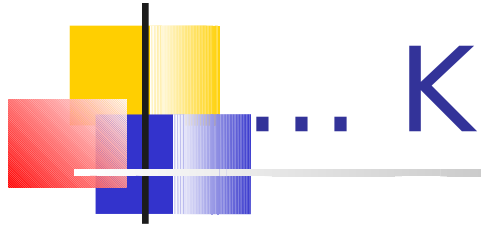


C S D T A M P I B W N G U

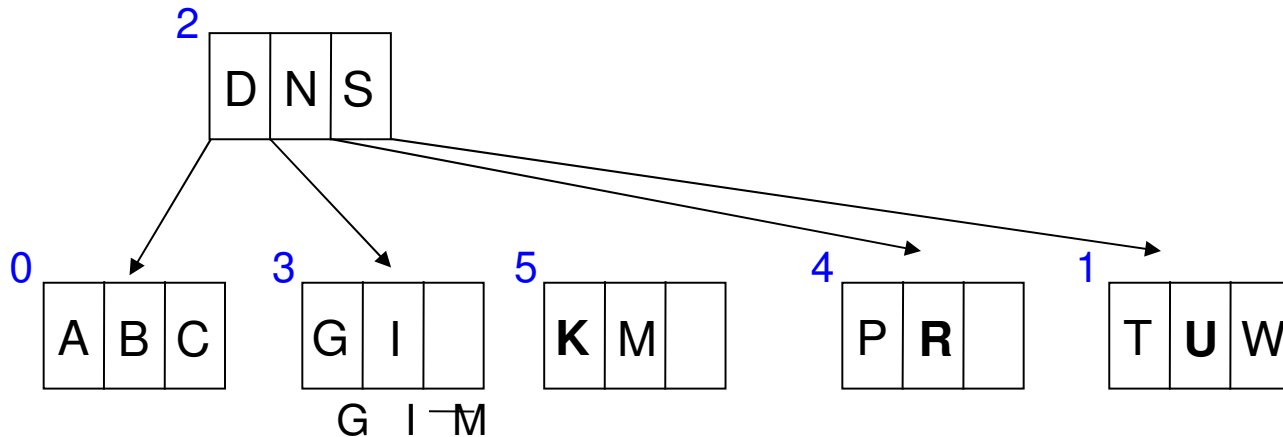
- Passo 7 – inserção de G, U, R
 - nós folhas com espaço

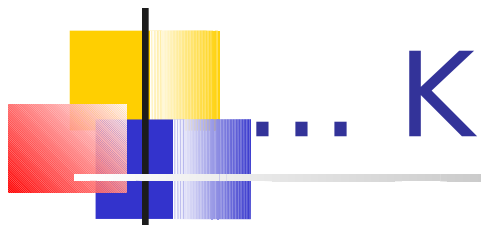


- particionamento do nó 3



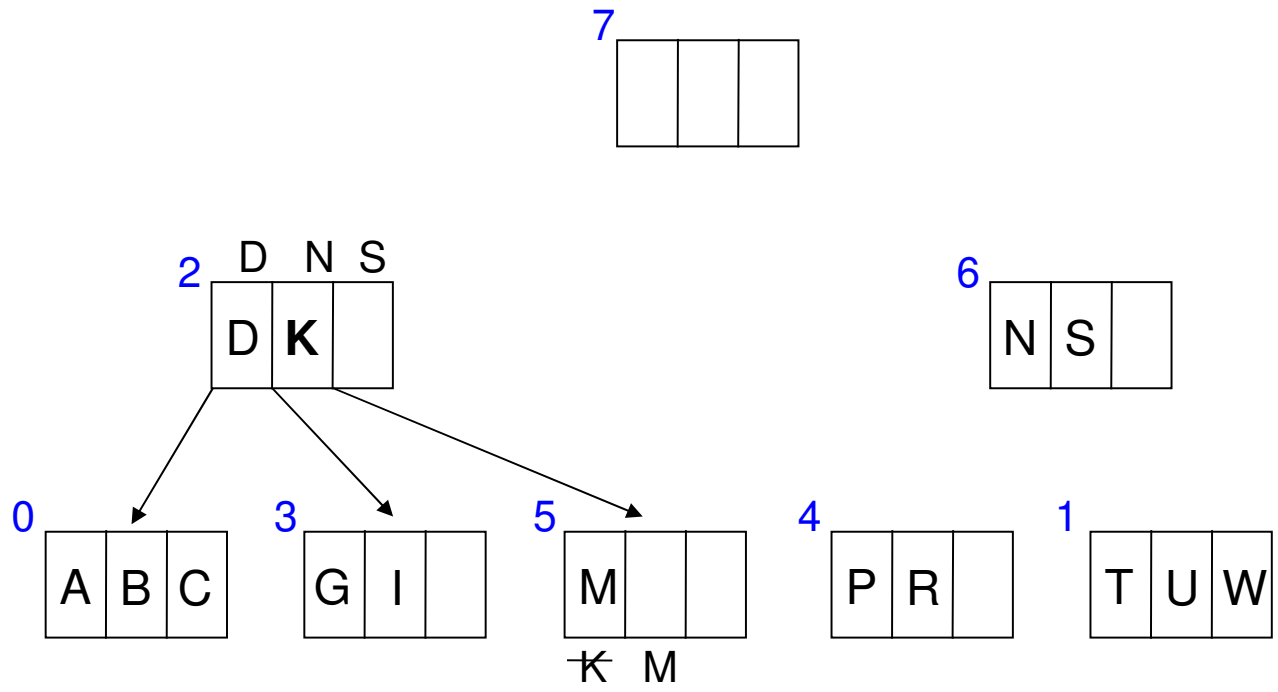
- Passo 8 – inserção de K
 - nó folha 3 cheio

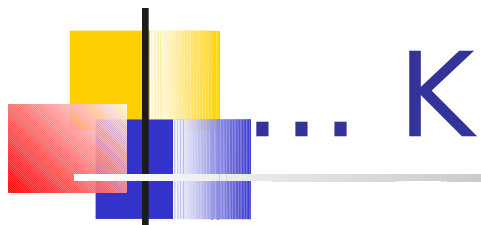




- particionamento do nó 3
- promoção de K
- particionamento do nó 2 e criação de nova raiz

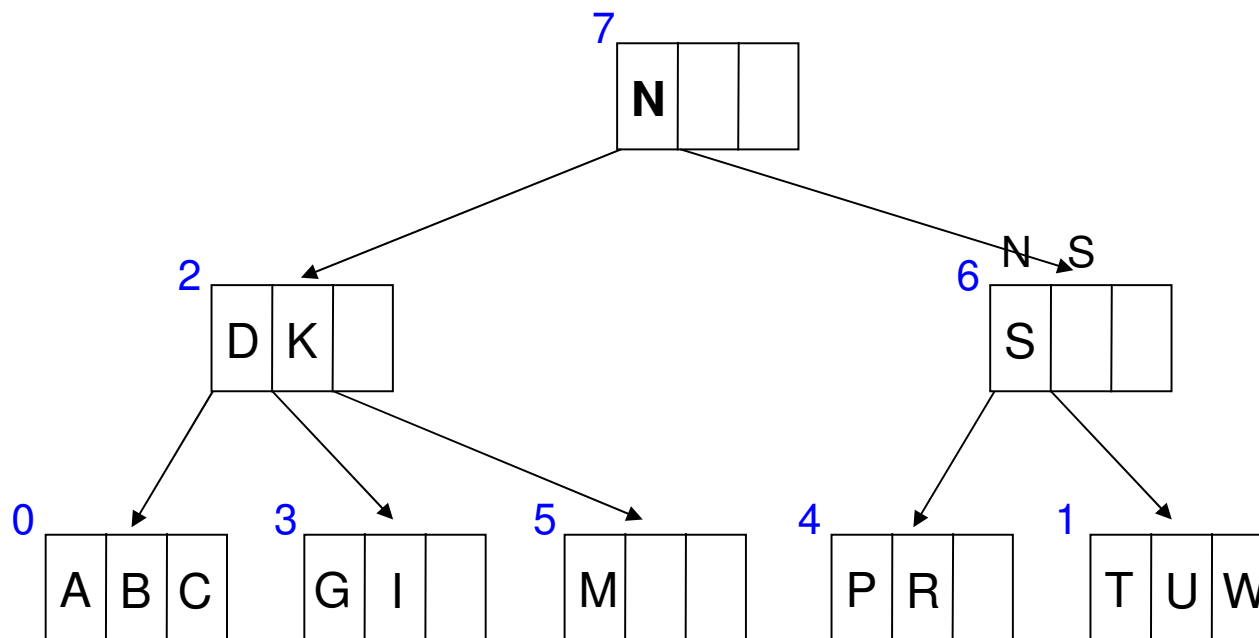
Passo 8 – inserção de K





- particionamento do nó 3
- promoção de K
- particionamento do nó 2
- promoção de N

Passo 8 – inserção de K





Exercício

- Na árvore-B do exemplo anterior, insira a chave \$, sendo que $\$ < A$



Exercício

- Insira as seguintes chaves em um índice árvore-B
 - C S D T A M P I B W N G U R K E H O L
 - diferentemente do exemplo anterior, escolha o último elemento do primeiro nó para promoção durante o particionamento do nó.



Exercício

- Construa uma árvore-B de ordem 3 pela inserção das chaves A, B, C, D, E, F, G, H e I, nessa ordem
- Qual o efeito da inserção das chaves em ordem alfabética? A árvore degenerou?