

Árvore B, B* e B+

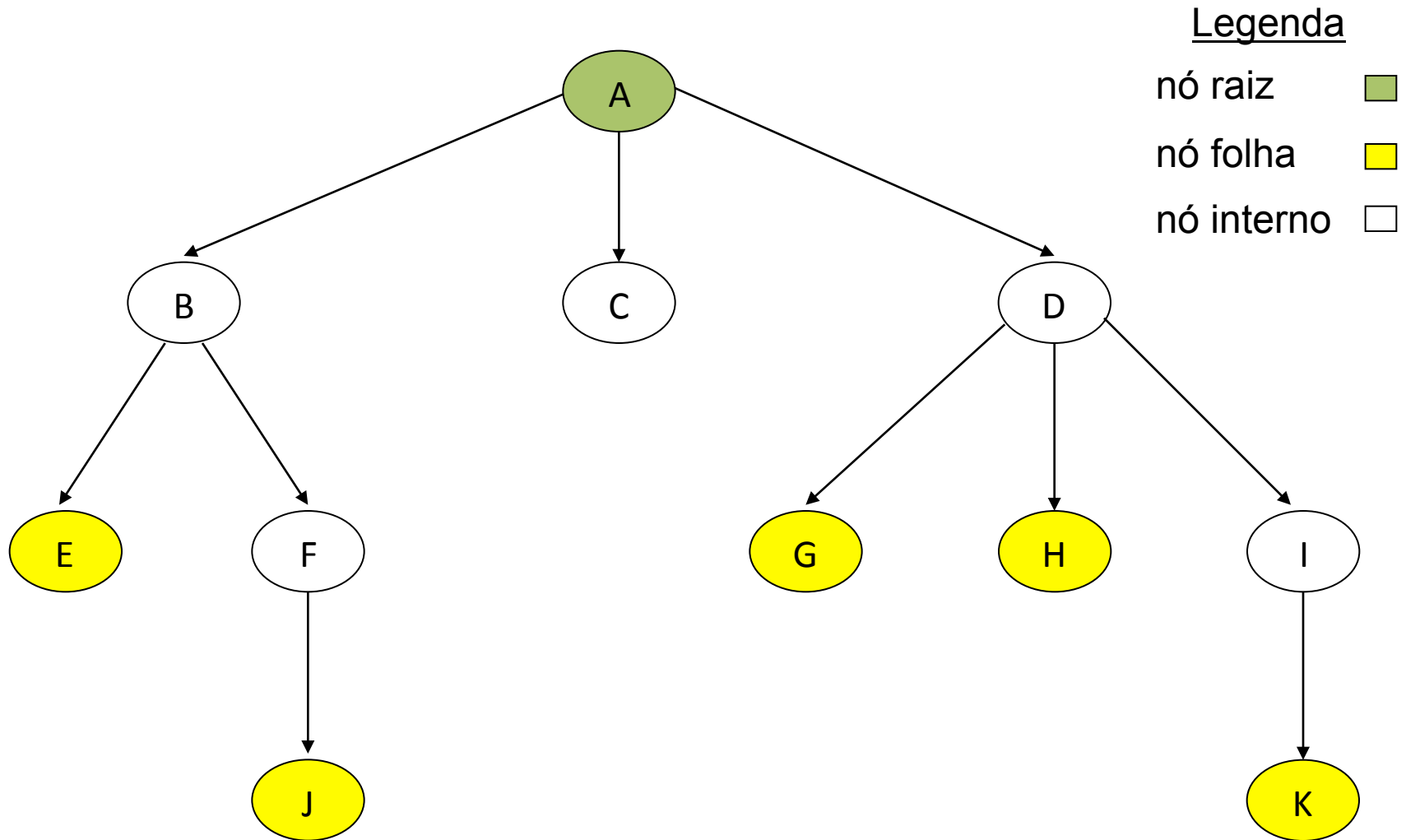
Murilo Gleyson Gazzola

Slides: Profa. Dra. Cristina Dutra de
Aguilar Ciferri

Tópicos

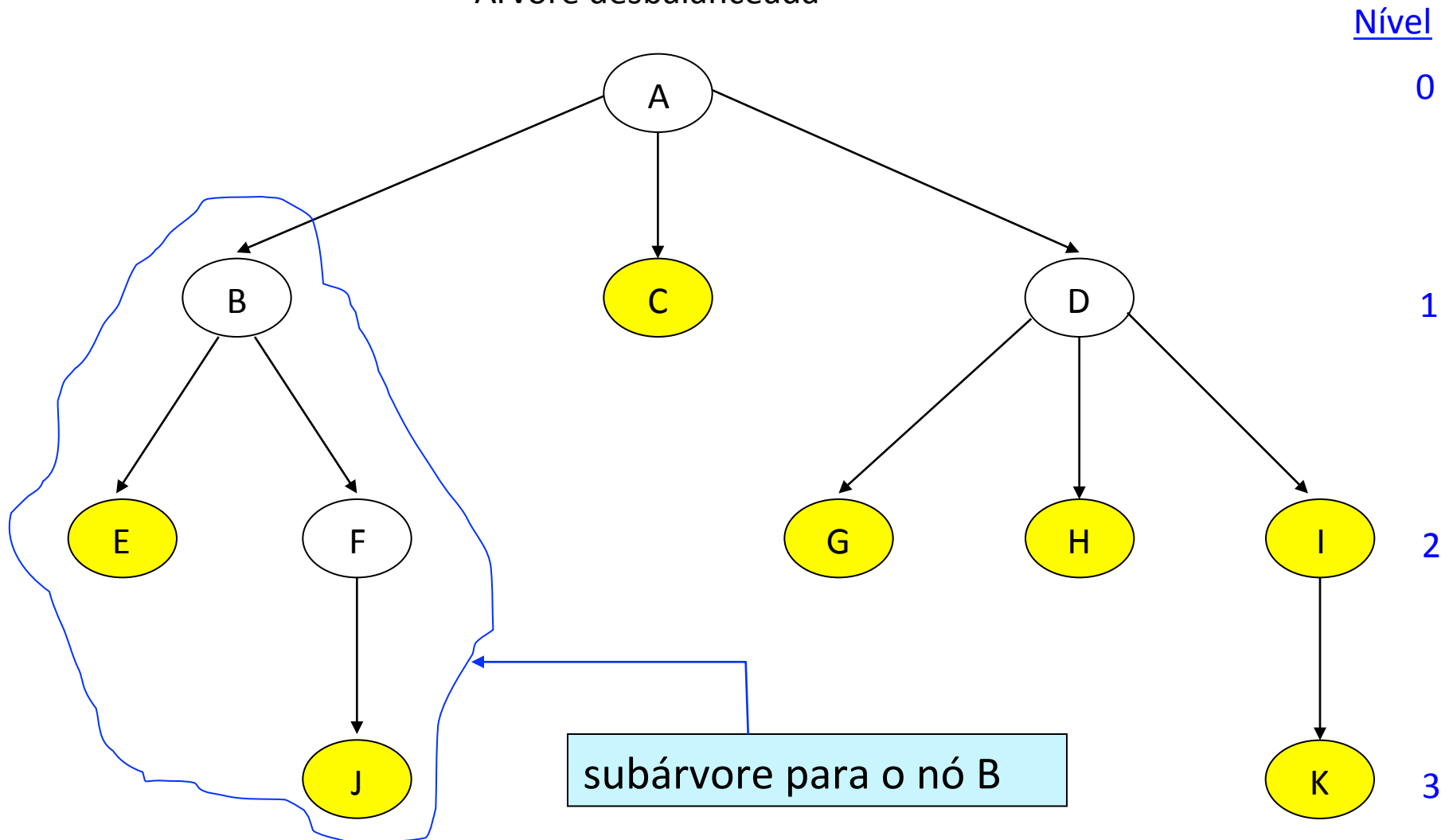
- Árvore de Pesquisa
- Árvore B
- Árvore B*
- Arvore B+

Estrutura de Dados de Árvore



Estrutura de Dados de Árvore

Árvore desbalanceada



Tópicos

- Árvore de Pesquisa
- Árvore B
- Árvore B*
- Árvore B+

Objetivos para balancear uma árvore de pesquisa

- Garantir que os nós sejam distribuídos por igual, de modo que a profundidade da árvore seja minimizada para determinado conjunto de chaves
- Tornar a velocidade de pesquisa uniforme, de modo que o tempo médio para encontrar qualquer chave aleatória seja aproximadamente o mesmo

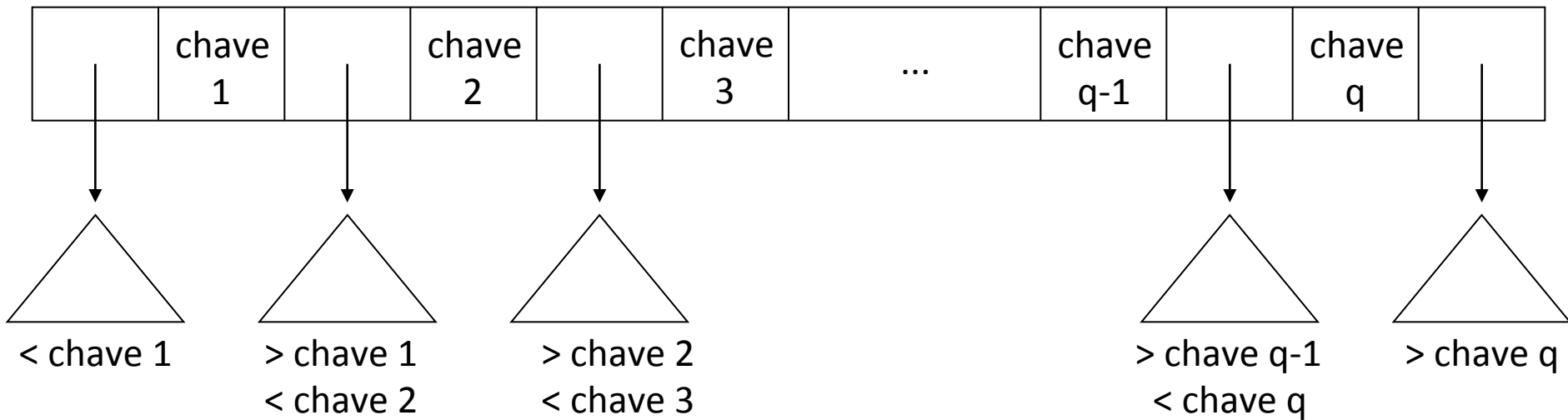
Árvores de Pesquisa

- Uma árvore de pesquisa de ordem p é uma árvore que contém no máximo $p-1$ valores de pesquisa e p ponteiros na ordem $\langle P_1, C_1, P_2, C_2, \dots, P_{q-1}, C_{q-1}, P_q \rangle$

Objetivos para balancear uma árvore de pesquisa

- Garantir que os nós sejam distribuídos por igual, de modo que a profundidade da árvore seja minimizada para determinado conjunto de chaves

Estrutura Lógica de um Nó



Tópicos

- Árvore de Pesquisa
- + **Árvore B**
 - Características
 - Inserção
 - Pesquisa
 - Remoção
 - Análise
- Árvore B*
- Arvore B+

Árvore-B

- São árvores de pesquisa balanceadas projetadas para funcionar bem em discos magnéticos ou outros dispositivos de armazenamento secundário (Cormen, T.)
- Muitos SGBD usam árvores B ou variações de árvores B para armazenar informações

Árvore-B

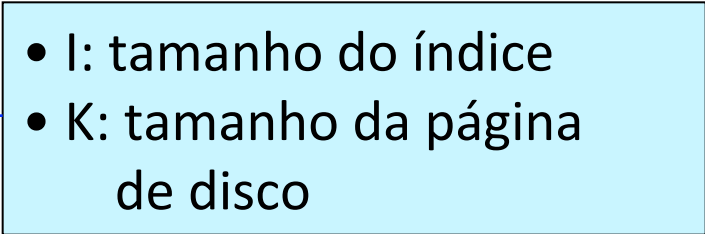
- Método genérico para o armazenamento e a recuperação de dados
 - voltado para arquivos volumosos
 - proporciona rápido acesso aos dados
 - possui custo mínimo de *overhead*
- Referência
 - Bayer, R.; McCreight, E. *Organization and Maintenance of Large Ordered Indexes*.
 - Boing Corporation, 1972.

Árvore-B

- Autores por Bayer e McCreight, 1972
- Trabalho foi desenvolvido na Boeing Scientific Research Labs.
- São árvores de pesquisa balanceadas projetadas para funcionar bem em discos magnéticos ou outros dispositivos de armazenamento secundário (Cormen, T.)
 - voltado para arquivos volumosos
 - proporciona rápido acesso aos dados

Características

- Índice
 - extremamente volumoso
- *Buffer-pool* pequeno
 - apenas uma parcela do índice pode ser carregada em memória principal
 - operações baseadas em disco
- Desempenho
 - proporcional a \log_K^I
ou melhor

- 
- I: tamanho do índice
 - K: tamanho da página de disco

Características

- Normalmente um nó da árvore B é tão grande quanto uma página de disco inteira.
- Desempenho (Goodrich M.)
 - Um árvore B com n itens tem complexidade de E/S
 - $O(\log_B n)$ para operações de pesquisa/atualização e usa $O(n/B)$ blocos.
 - B é o tamanho de um bloco.

Características

- Desempenho
 - O número de acessos ao disco exigidos para a maioria das operações em uma árvore B é proporcional a sua altura (Cormen, T.).
 - Uma árvore B com n itens tem complexidade de E/S
 - $O(\log_B n)$ para operações de pesquisa/atualização e usa $O(n/B)$ blocos.
 - B é o tamanho de um bloco.

Características

- Desempenho
 - O número de acessos ao disco exigidos para a maioria das operações em uma árvore B é proporcional a sua altura (Cormen, T.).

Características

- Nó
 - seqüência ordenada de chaves
 - conjunto de ponteiros
 - número de ponteiros = número de chaves + 1
- Ordem
 - número máximo de ponteiros que pode ser armazenado em um nó
 - exemplo: árvore B de ordem 8

Características

- Nó
 - seqüência ordenada de chaves
 - conjunto de ponteiros
 - número de ponteiros = número de chaves + 1
- 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

Nomenclatura


- Formalização da terminologia
 - especifica precisamente as propriedades que devem estar presentes para uma estrutura de dados ser qualificada como árvore-B
 - direciona a implementação do algoritmo de remoção da árvore-B
- Problema
 - literatura não é uniforme no uso e definição dos termos

Características

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

Ordem

- Bayer and McGreight (1972)
Cormen (1979)
 - número mínimo de chaves que podem estar em uma página da árvore
 - Knuth (1973)
 - número máximo de descendentes que uma página pode ter
 - facilita a determinação de nó cheio

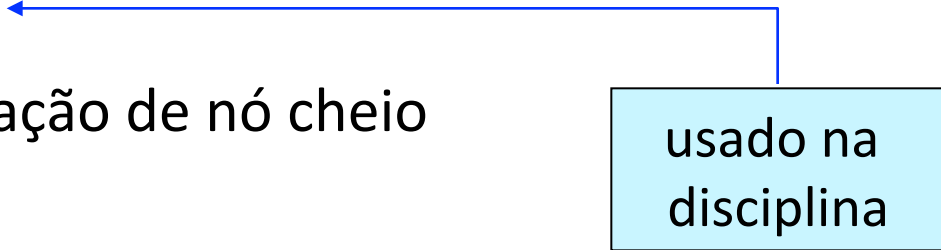
⇒ chaves = ordem – 1 (máximo)
- 
- usado na disciplina

Página ou Nó das Árvores B

- Quando uma árvore de pesquisa possui mais de uma chave por nó, ela deixa de ser binária e passa a ser chamadas n-árias.
- Os nós são mais comumente chamados de páginas ao invés de nó. (Ziviani)

Ordem


- Bayer and McGreight (1972)
Cormen (1979)
 - número mínimo de chaves que podem estar em uma página da árvore
- Knuth (1973)
 - número máximo de descendentes que uma nó pode ter
 - facilita a determinação de nó cheio



usado na disciplina

Ordem

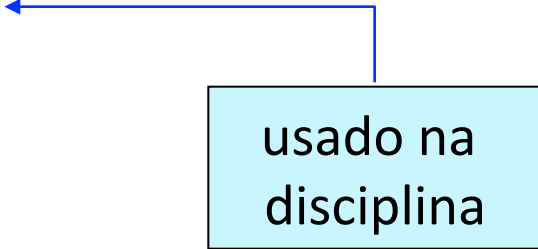
- Bayer and McGreight (1972)
Cormen (1979)
 - número mínimo de chaves que podem estar em uma página da árvore
- Knuth (1973)
 - número máximo de descendentes que uma nó pode ter
 - facilita a determinação de nó cheio



usado na disciplina

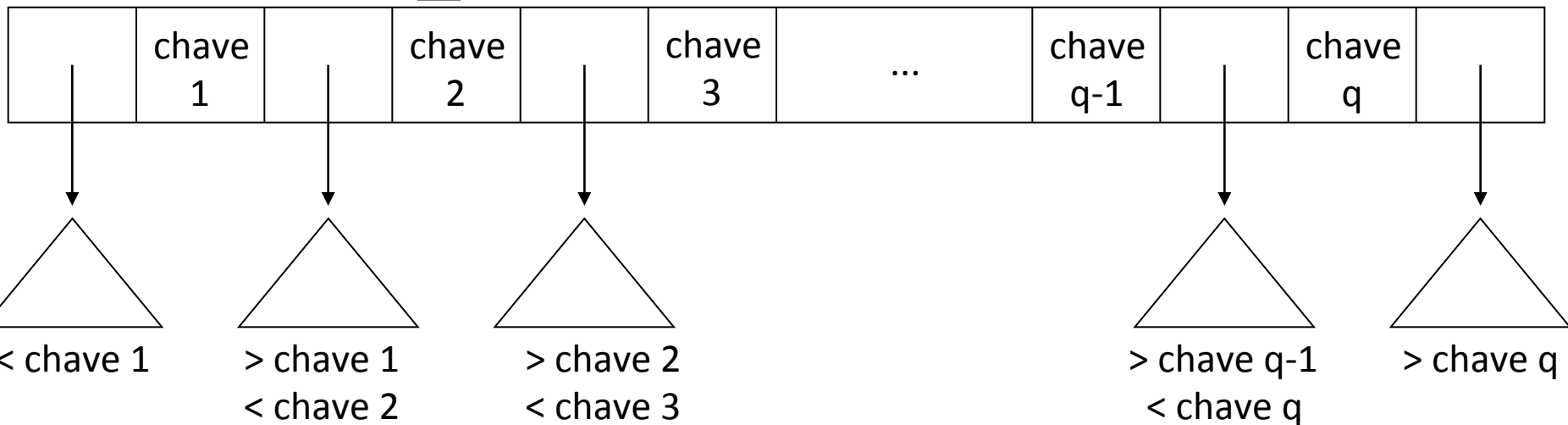
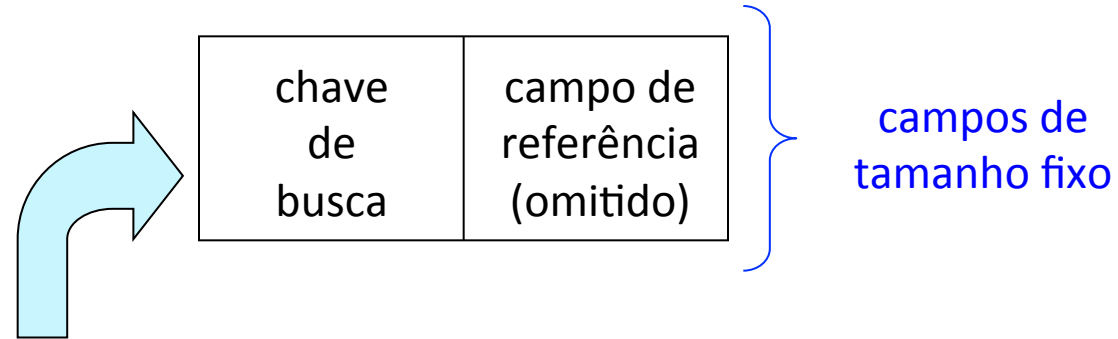
Nó Folha

- Bayer and McGreight (1972)
 - nível mais baixo das chaves
- Knuth (1973)
 - um nível depois do nível mais baixo das chaves
 - ⇒ folhas: registros de dados que podem ser apontados pelo nível mais baixo das chaves

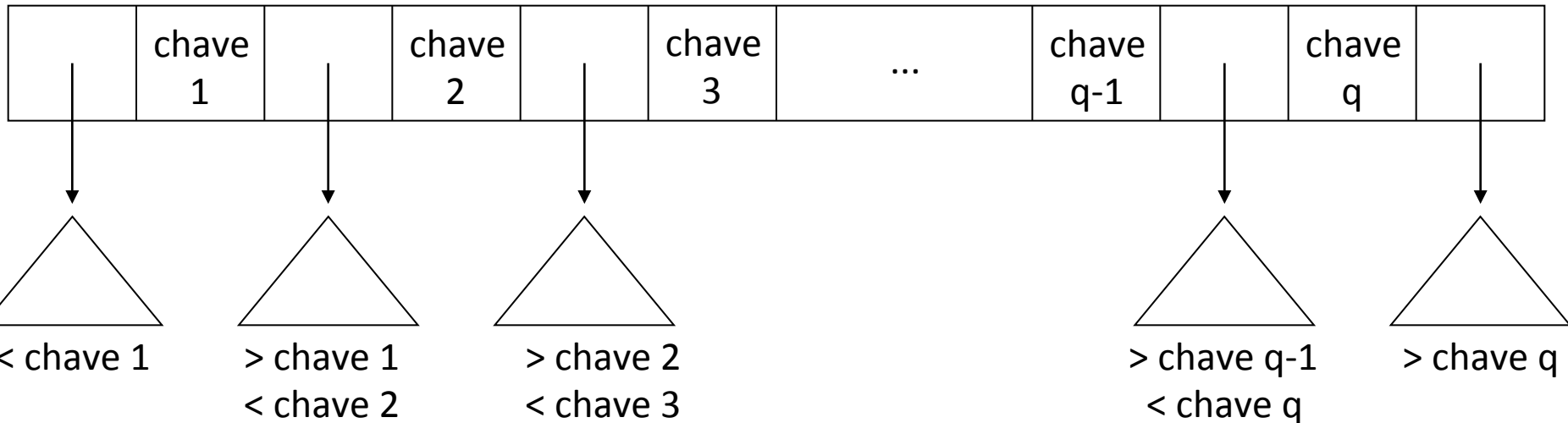
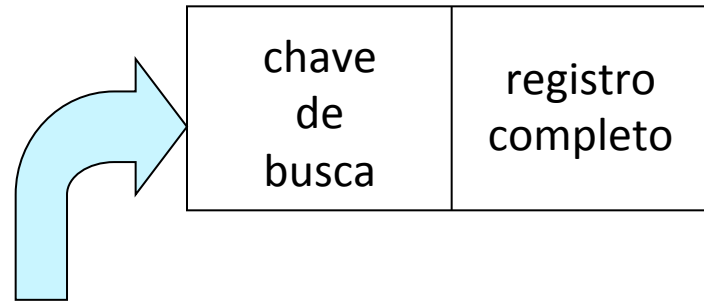


usado na disciplina

Estrutura Lógica de um Nó



Estrutura Lógica de um Nó



Árvore-B de Ordem m

- *Split*
 - os descendentes são divididos o mais uniformemente possível entre as páginas velha e nova
- Cada página, exceto a raiz e os nós folhas
 - $\lceil m/2 \rceil$ descendentes (pelo menos)
 - $\lceil m/2 \rceil - 1$ chaves (no mínimo)

Árvore-B de Ordem m

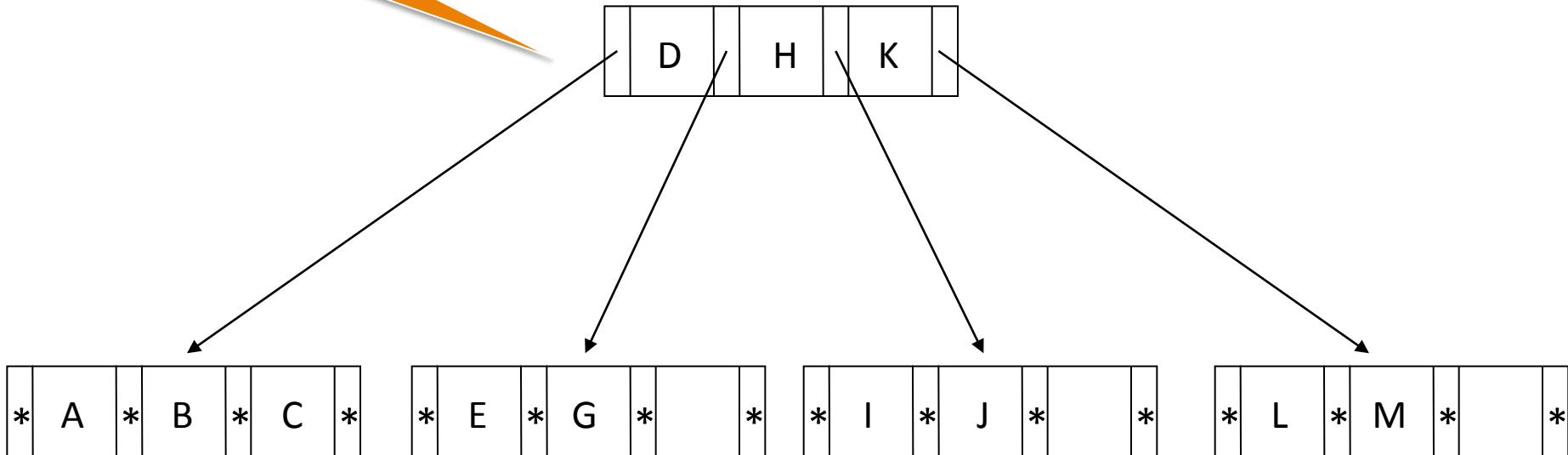
- *Split*
 - os descendentes são divididos o mais uniformemente possível entre as páginas velha e nova
- Cada página, exceto a raiz e os nós folhas
 - $\lceil m/2 \rceil$ descendentes (pelo menos)
 - $\lceil m/2 \rceil - 1$ chaves (no mínimo)

Definição Formal

- **Árvore-B com ordem m**
 - cada página possui um máximo de m descendentes
 - cada página, exceto a raiz e as folhas, possui no mínimo $\lceil m/2 \rceil$ descendentes → taxa de ocupação
 - a raiz possui pelo menos 2 descendentes, a menos que seja um nó folha
 - todas as folhas aparecem no mesmo nível
 - uma página interna com k descendentes contém $k-1$ chaves
 - uma folha possui no mínimo $\lceil m/2 \rceil - 1$ chaves e no máximo $m - 1$ chaves → taxa de ocupação

Exemplo

Ordem 4



Tópicos

- Árvore de Pesquisa
- + **Árvore B**
 - Características
 - **Inserção**
 - Pesquisa
 - Remoção
 - Análise
- Árvore B*
- Arvore B+

Inserção de Dados (Chave)

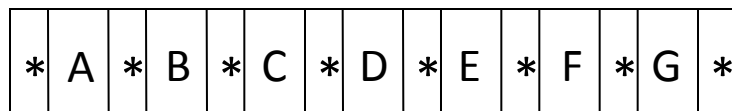
- Característica
 - sempre realizada nos nós folhas
- Situações a serem analisadas
 - árvore vazia
 - *overflow* no nó raiz
 - inserção nos nós folhas

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
 - 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 (NIL)
 - nós internos: RRN do nó filho ou -1
- Nó raiz (= nó folha)



Inserção: Situação Inicial

- Chaves B C G E F D A
- Ponteiros (*)
 - nós folhas: -1 ou fim de lista (NIL)
 - nós internos: RRN do nó filho ou -1
- Nó raiz (= nó folha)

Inseridas
ordenadas ou
desordenadas?
Ocorre overflow?

Inserção: Situação Inicial

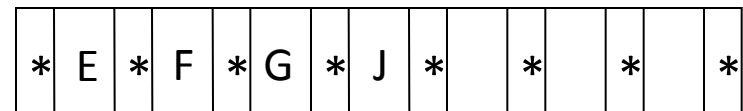
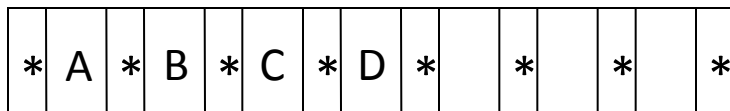
- Chaves B C G E F D A
- Ponteiros (*)
 - nós folhas: -1 ou fim de lista (NIL)
 - nós internos: RRN do nó filho ou -1
- Nó raiz (= nó folha)

Inseridas
ordenadas ou
desordenadas?

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

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
 - chaves do nó original + nova chave
- Exemplo: inserção de J



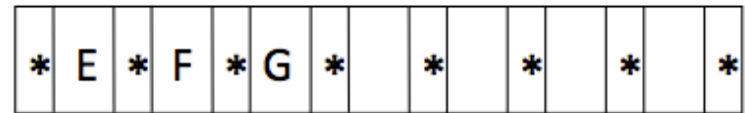
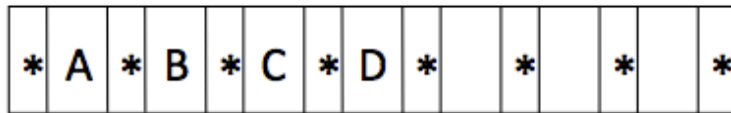
Inserção: *Overflow* Nó Raiz

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

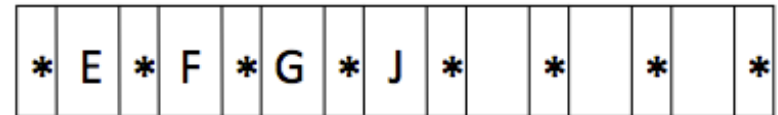
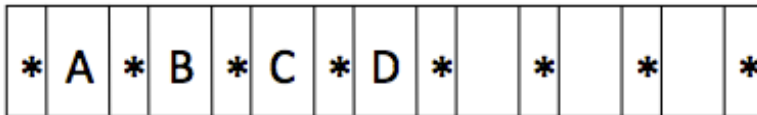
Inserção: *Overflow* Nó Raiz

- Passo 1 - Inserção J

Split

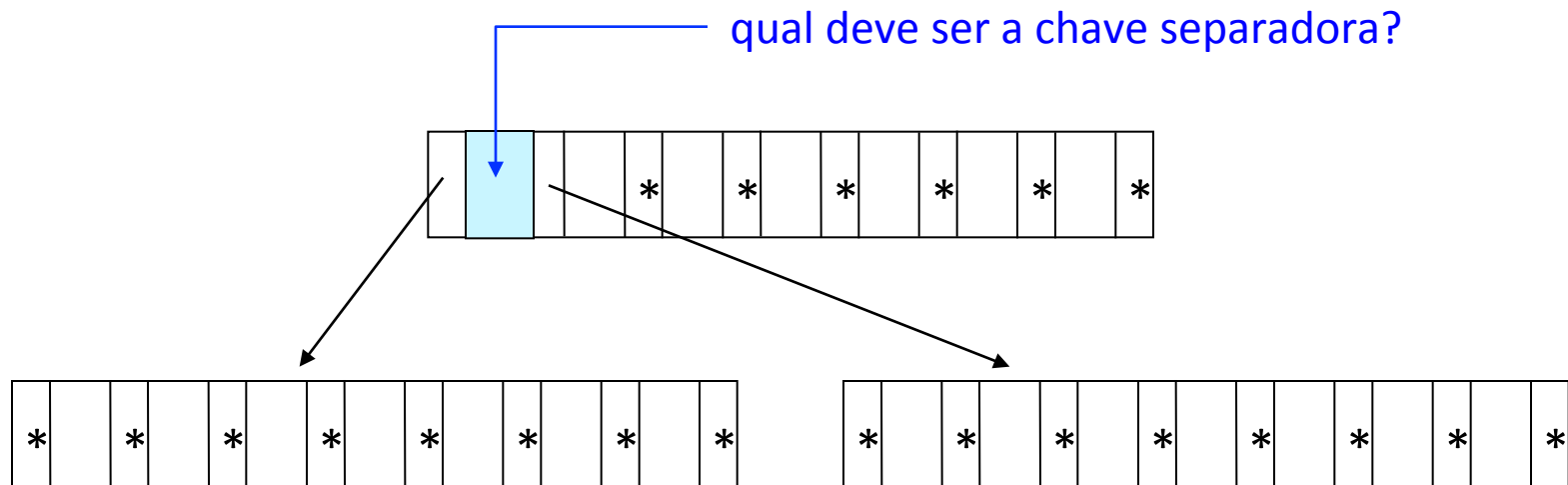


Inserir J



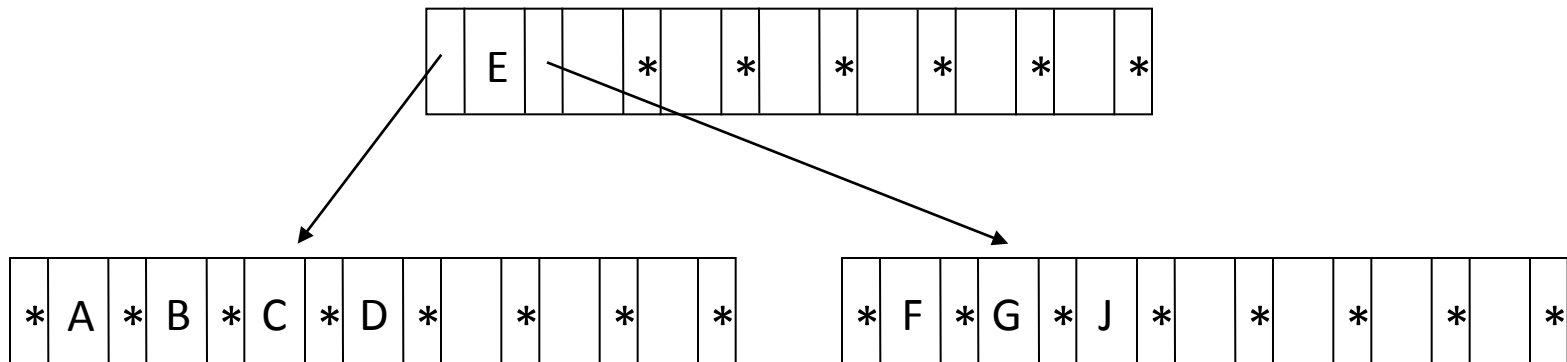
Inserção: *Overflow* Nó Raiz

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



Inserção: *Overflow* Nó Raiz

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



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
 - ajuste do nó pai para apontar para o novo nó
 - propagação de *overflow*

Exemplo

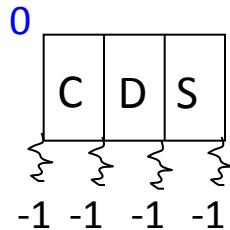
- Insira as seguintes chaves em um índice árvore-B
 - C S D T A M P I B W N G U R K E H O L J Y Q Z F X V
- Ordem da árvore-B: 4
 - em cada nó (página de disco)
 - número de chaves: 3
 - número de ponteiros: 4

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

↑
Próximo

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

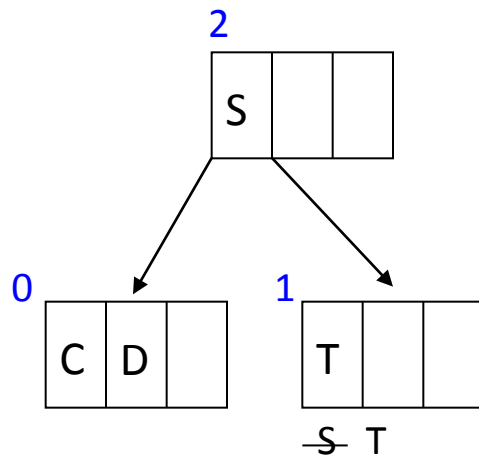
- C
- C S
- C D S



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

↑
Próximo

- Passo 2 – inserção de T
– nó raiz cheio

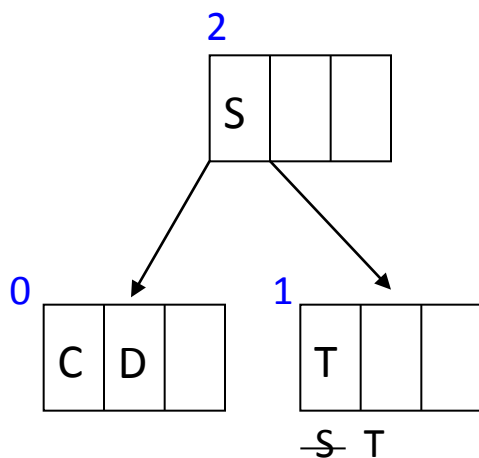


C S D T A M P I B W N G U R K ...

↑
Próximo

- Passo 2 – inserção de T
– nó raiz cheio

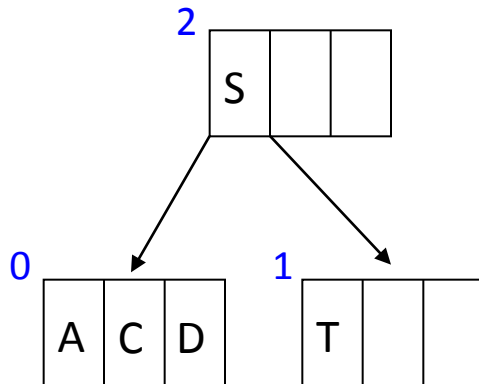
- particionamento do nó
- criação de uma nova raiz
- promoção de S



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

↑
Próximo

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

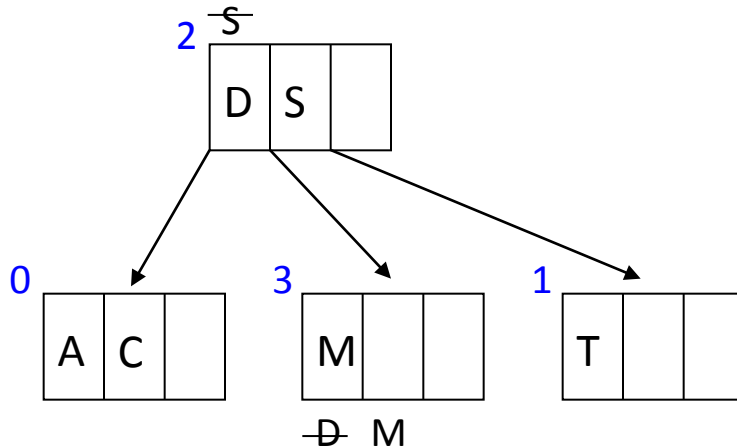


C S D T A M P I B W N G U R K ...

↑
Próximo

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

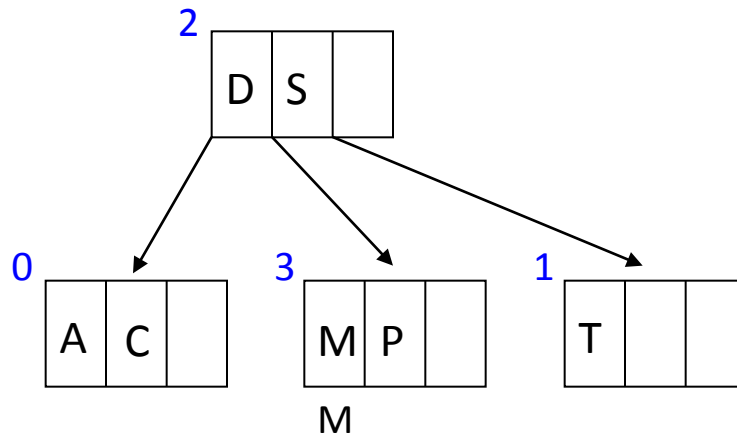
- particionamento do nó
- promoção de D



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

↑
Próximo

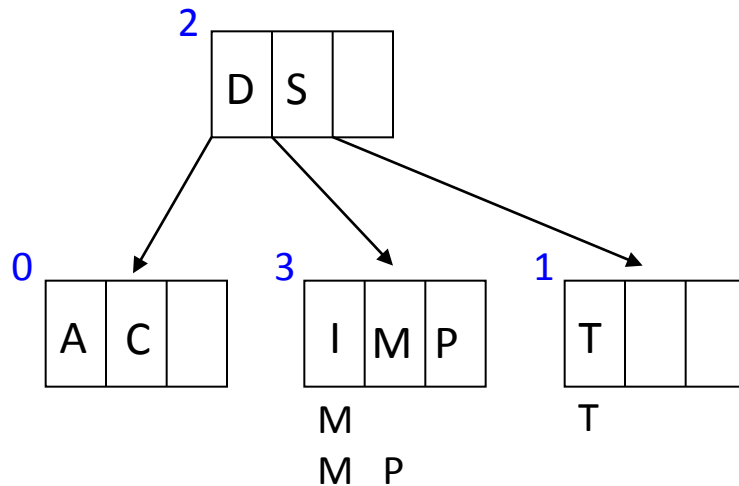
- Passo 5 – inserção de P
– nós folhas com espaço



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

↑
Próximo

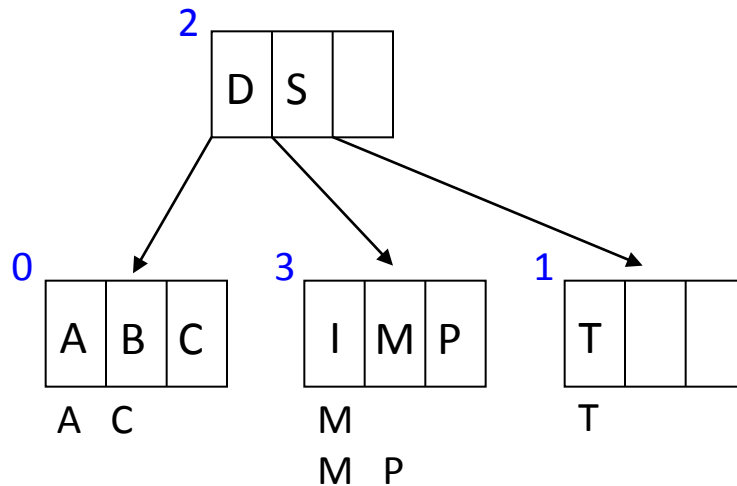
- Passo 5 – inserção de I
– nós folhas com espaço



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

↑
Próximo

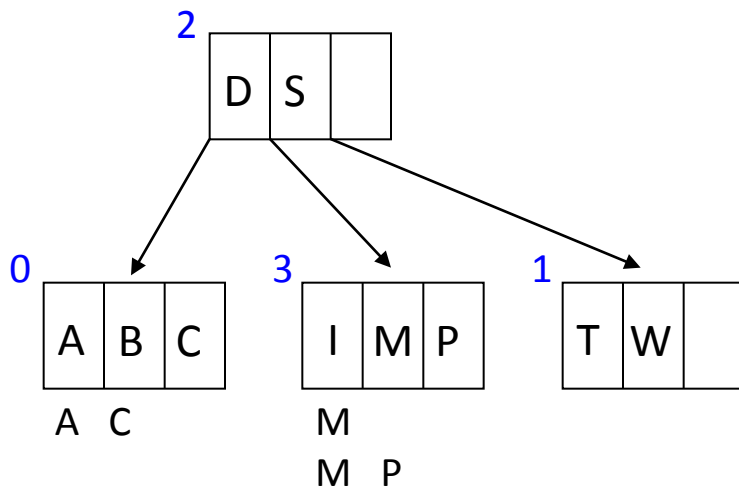
- Passo 5 – inserção de B
– nós folhas com espaço



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

↑
Próximo

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

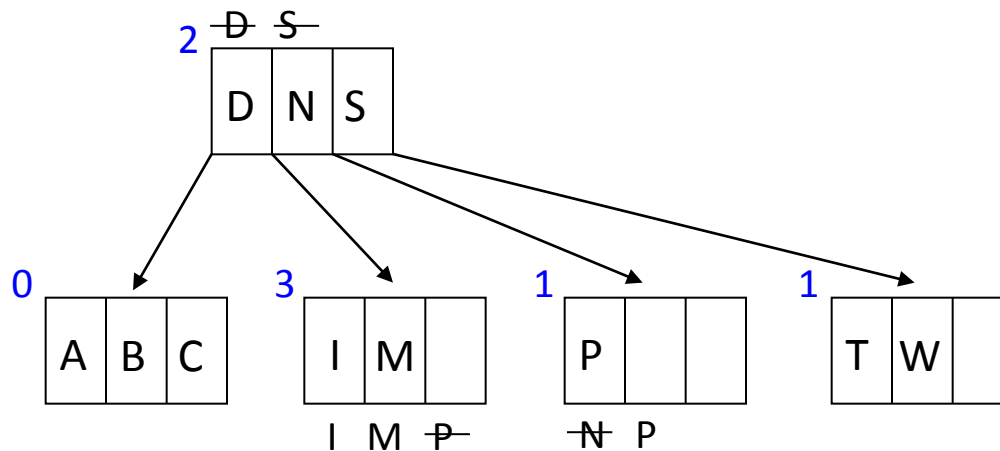


C S D T A M P I B W N G U R K ...

↑
Próximo

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

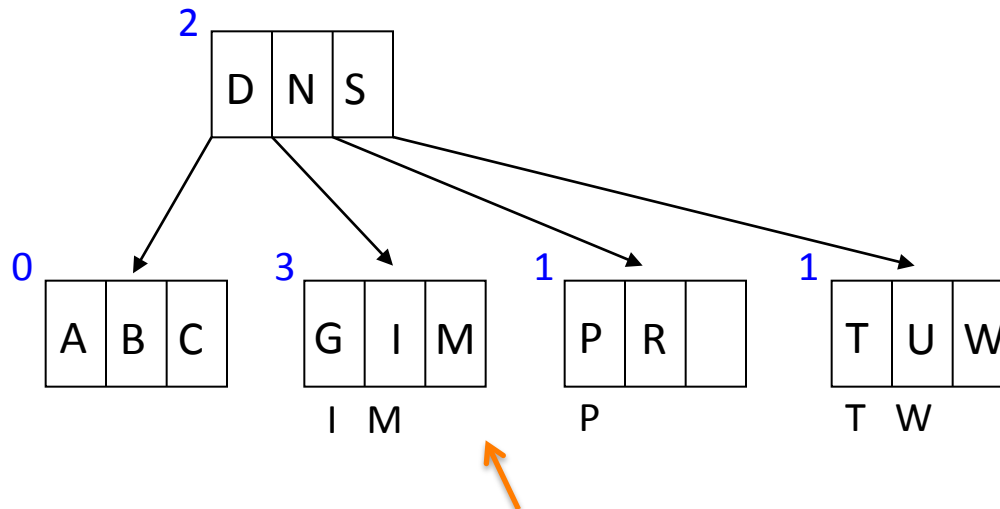
- particionamento do nó
- promoção de N



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

↑
Próximo

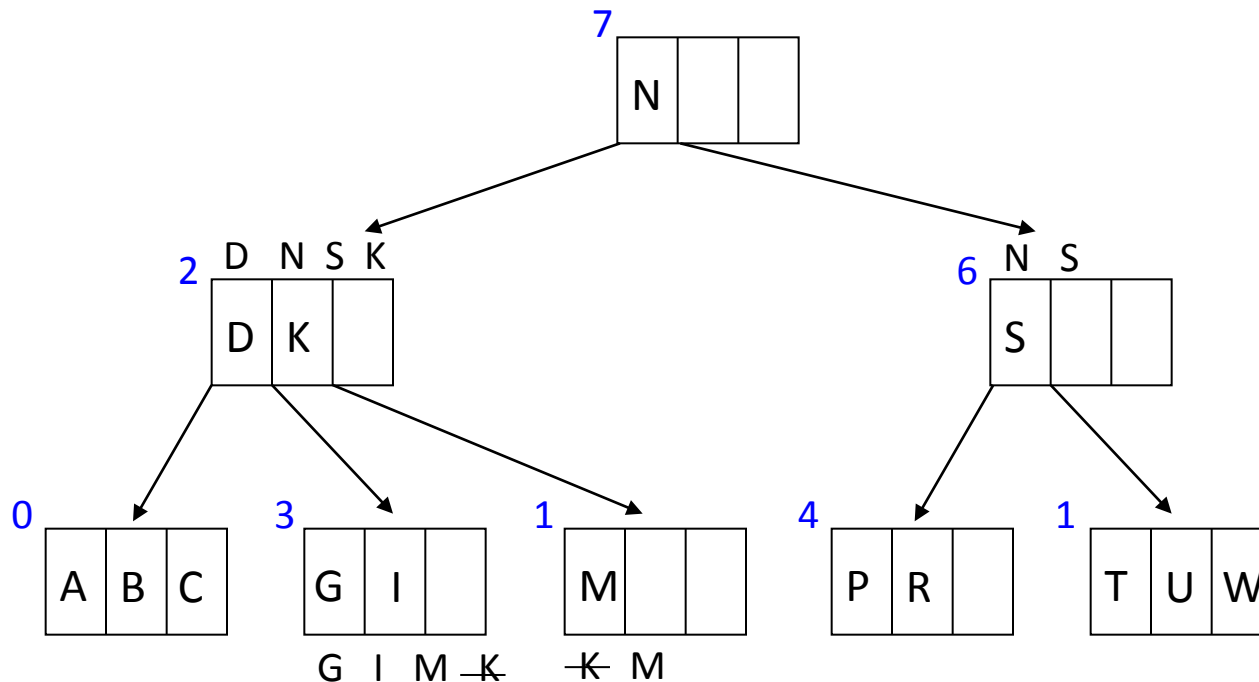
- Passo 7 – inserção de G, U, R
– nós folhas com espaço



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

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

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



... E H O L J Y Q Z F X V

- Finalizar a construção da árvore

Exercícios

- Na árvore-B do exemplo anterior, insira a chave \$, sendo que $\$ < A$.
- 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 J Y Q Z F X V
 - diferentemente do exemplo anterior, escolha o último elemento do primeiro nó para promoção durante o particionamento do nó.

Algoritmo

- Estrutura de dados
 - para cada bloco de disco
 - diferentes formas de implementação
 - contador de ocupação
 - chaves \Rightarrow caracteres
 - ponteiros \Rightarrow campos de referência para cada chave

Declaração (Pascal)

```
type BTPAGE = record
  KEYCOUNT: integer;
    // MAXKEYS: número máximo de chaves por bloco
  KEY: array [ 1..MAXKEYS] of char;
    // MAXCHILDREN: número máximo de ponteiros por bloco
  CHILD: array [ 1..MAXCHILDREN] of integer;
end;
```

```
var PAGE: BTPAGE;
```

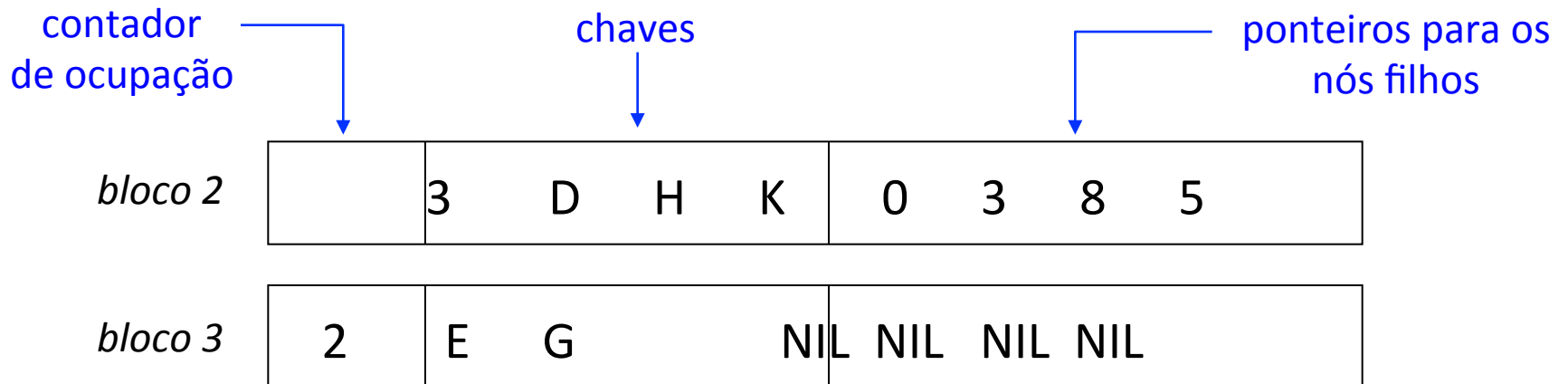
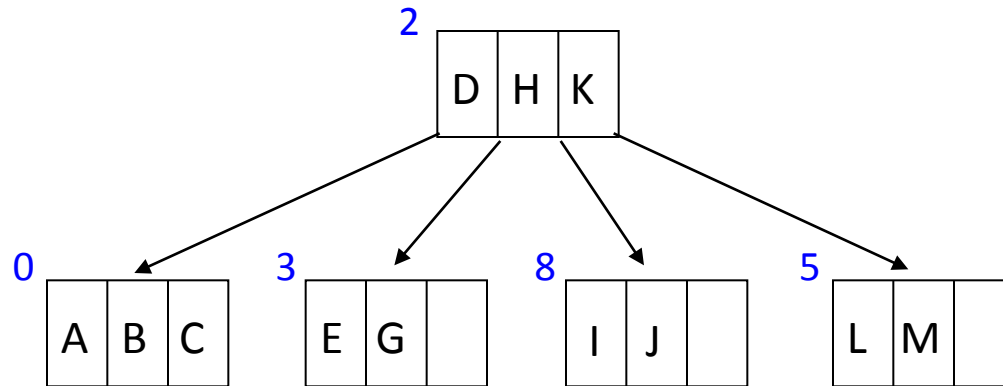
- PAGE.KEYCOUNT
 - útil para determinar se o bloco está cheio

Declaração em C - Estrutura

```
#define ORDEM 5
```

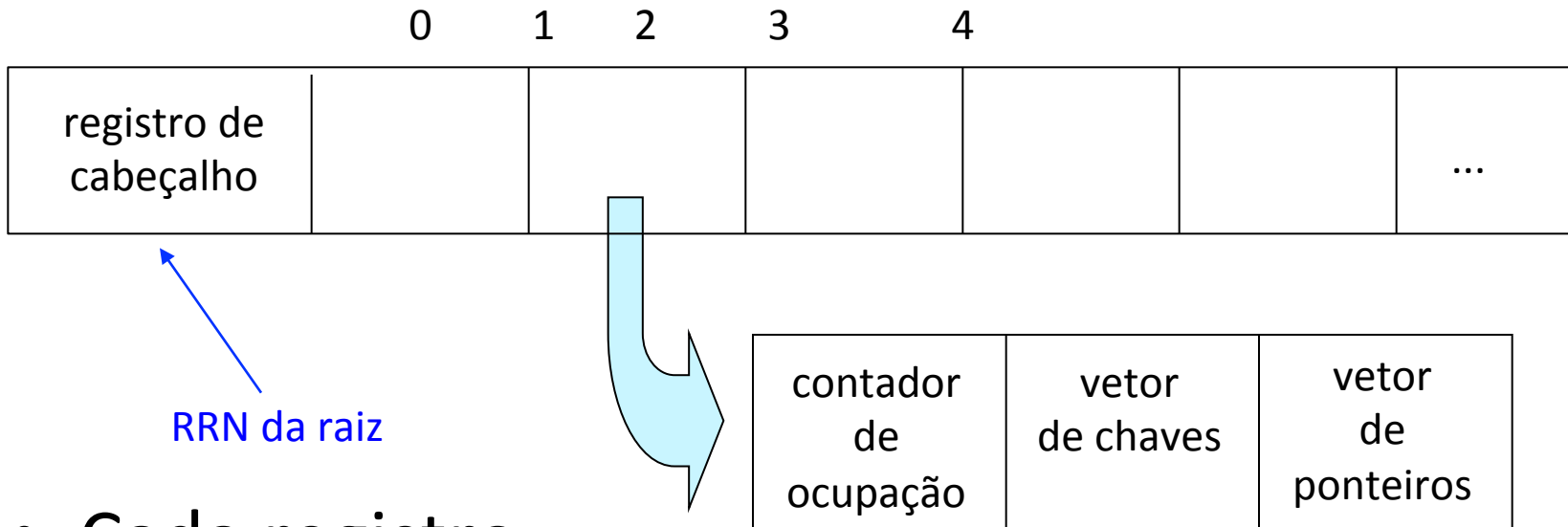
```
typedef struct pagina {  
    short n;  
    int chave[ORDEM-1];  
    struct pagina* filho[ORDEM];  
} tpag;
```

Exemplo



Arquivo da Árvore B

- Conjunto de registros de tamanho fixo



- Cada registro
 - contém um bloco de disco

Tópicos

- Árvore de Pesquisa
- + **Árvore B**
 - Características
 - Inserção
 - **Pesquisa**
 - Remoção
 - Análise
- Árvore B*
- Arvore B+

Algoritmo: Pesquisa

```
bool busca(arvoreB *raiz, int info)
{
    arvoreB *no;
    int pos; //posição retornada pelo busca binária.

    no = raiz;
    while (no != NULL)
    {
        pos = busca_binaria(no, info);
        if (pos < no->num_chaves && no->chaves[pos] == info)
            return(true);
        else no = no->filhos[pos];
    }
    return(false);
}
```

É muito semelhante, porém ao invés de tomar uma decisão binária em cada nó, tomamos uma decisão de ramificação de várias vias.

Algoritmo: Pesquisa

```
int busca_binaria(arvoreB *no, int info)
{
    int meio, i, f;
    i = 0;
    f = no->num_chaves-1;

    while (i <= f)
    {
        meio = (i + f)/2;
        if (no->chaves[meio] == info)
            return(meio); //Encontrou. Retorna a posição em que a chave está.
        else if (no->chaves[meio] > info)
            f = meio - 1;
        else i = meio + 1;
    }
    return(i); //Não encontrou. Retorna a posição do ponteiro para o filho.
}
```

Algoritmo: Pesquisa

if KEY was found then

FOUND_RRN := RRN RRN corrente contém a chave

FOUND_POS := POS

return FOUND chave de busca encontrada

else a chave de busca não foi encontrada, portanto
 procura a chave de busca no nó filho

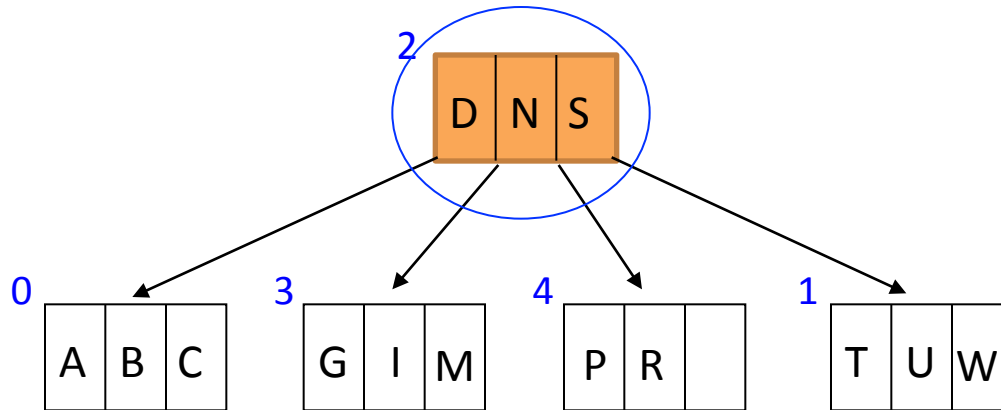
return (search(PAGE.CHILD[POS], KEY, FOUND_RRN,
 FOUND_POS))

endif

endif

end FUNCTION

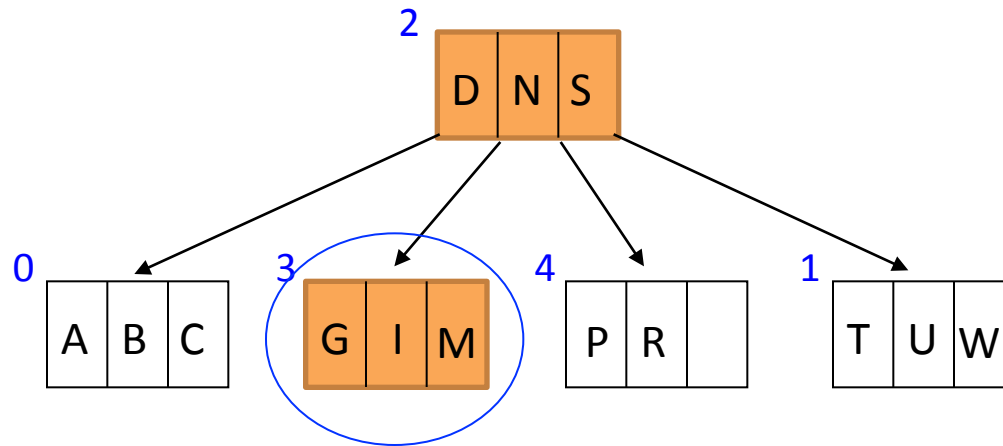
Busca da Chave "K"



PAGE =

D	N	S
---	---	---

Busca da Chave "K"

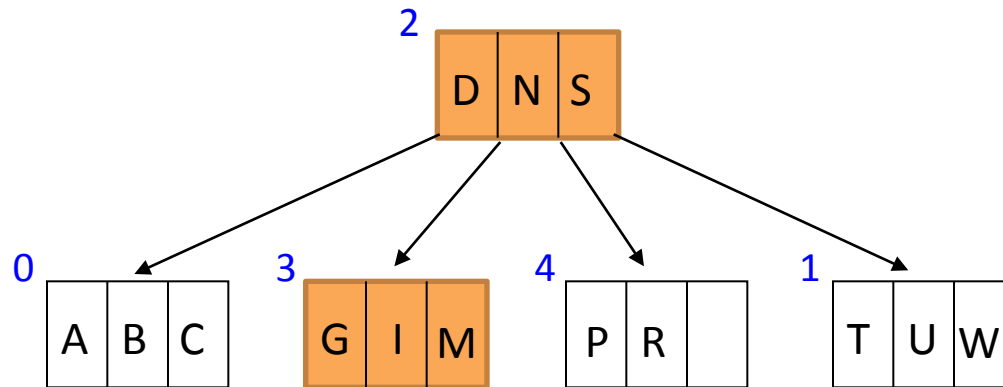


PAGE =

G	I	M
---	---	---

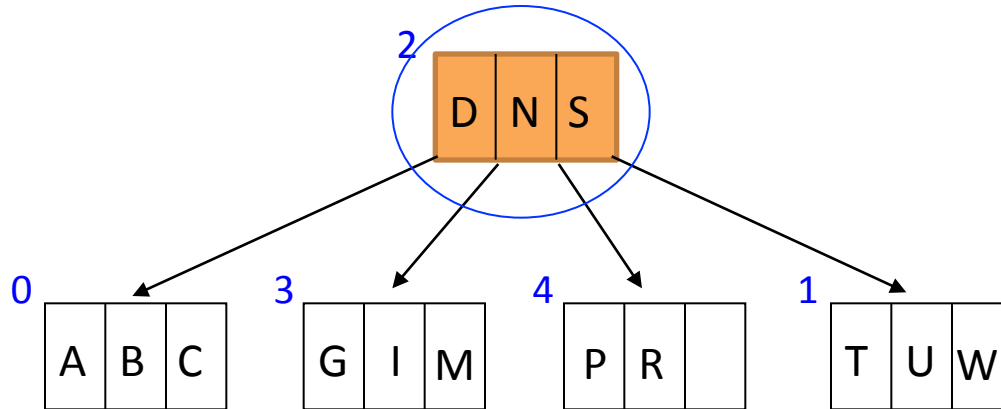
 não encontrado

Busca da Chave “K”



PAGE.CHILD[2] = NIL → chave de busca não encontrada
return NOT FOUND

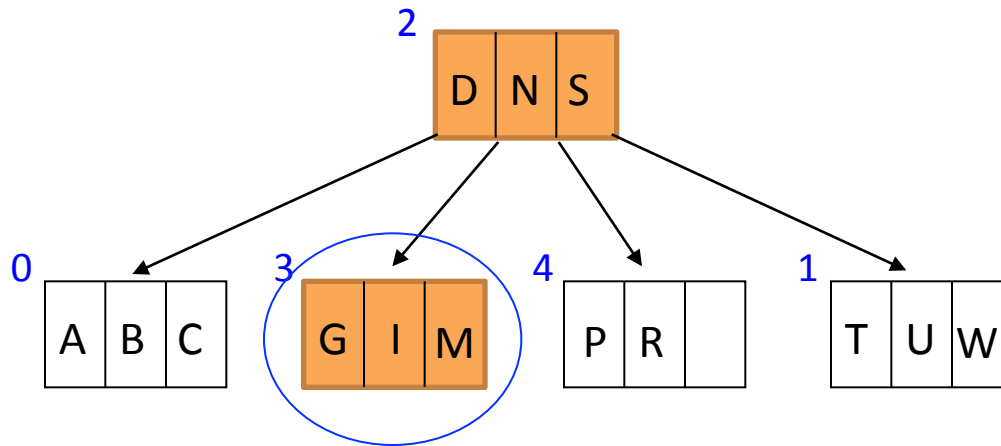
Busca da Chave "M"



PAGE =

D	N	S
---	---	---

Busca da Chave M



Ponteiro 2->3

POS = 2

Algoritmo: Pesquisa

```
FUNCTION: search (RRN,   página a ser pesquisada
                 KEY,   chave sendo procurada
                 FOUND_RRN, página que contém a chave
                 FOUND_POS) posição da chave na página
if RRN == NIL then
    return NOT FOUND   chave de busca não encontrada
else
    read page RRN into PAGE   leia o bloco apontado por RRN na
                             variável PAGE

    look through PAGE for KEY, setting POS equal to the position
    where KEY occurs or should occur
```

Algoritmo: Pesquisa

if KEY was found then

FOUND_RRN := RRN RRN corrente contém a chave

FOUND_POS := POS

return FOUND chave de busca encontrada

else a chave de busca não foi encontrada, portanto
 procura a chave de busca no nó filho

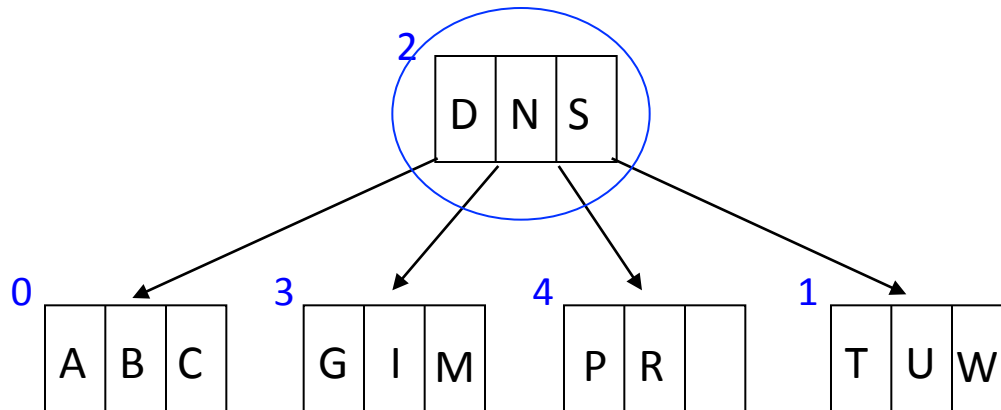
return (search(PAGE.CHILD[POS], KEY, FOUND_RRN,
 FOUND_POS))

endif

endif

end FUNCTION

Busca da Chave K



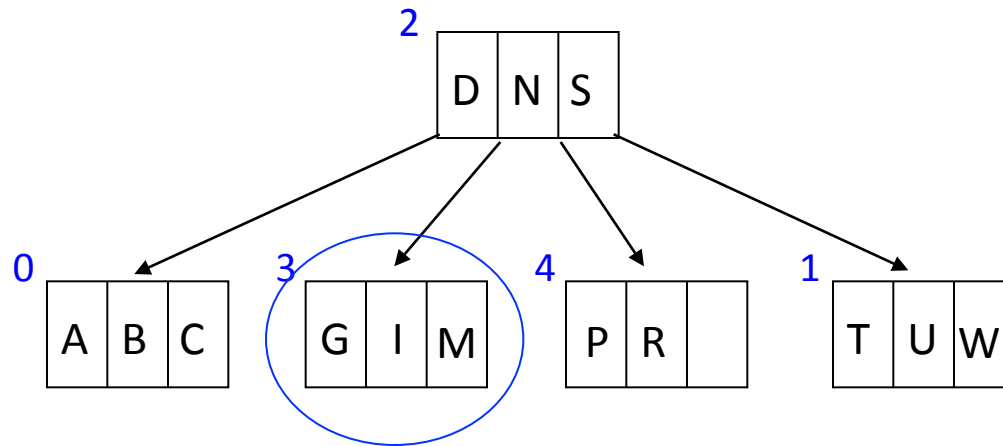
- search (2, K, FOUND_RRN, FOUND_POS)

PAGE =

D	N	S
---	---	---

 não existe → POS = 1

Busca da Chave K



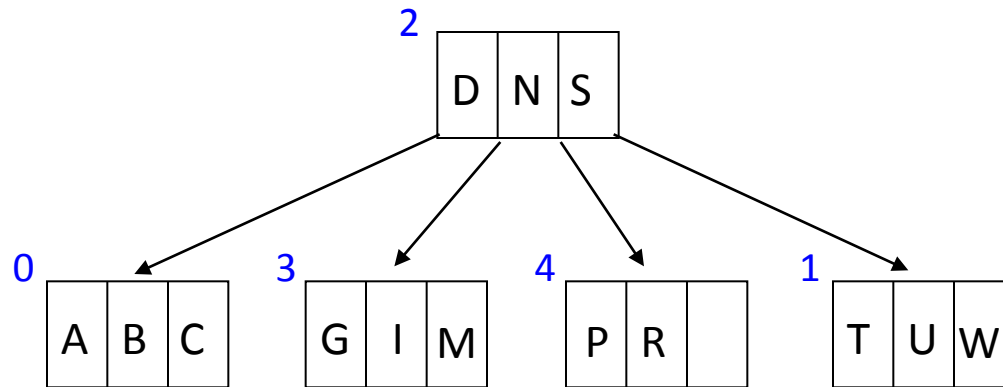
- search (PAGE.CHILD[1], K, FOUND_RRN, FOUND_POS)

PAGE =

G	I	M
---	---	---

 não existe → POS = 2

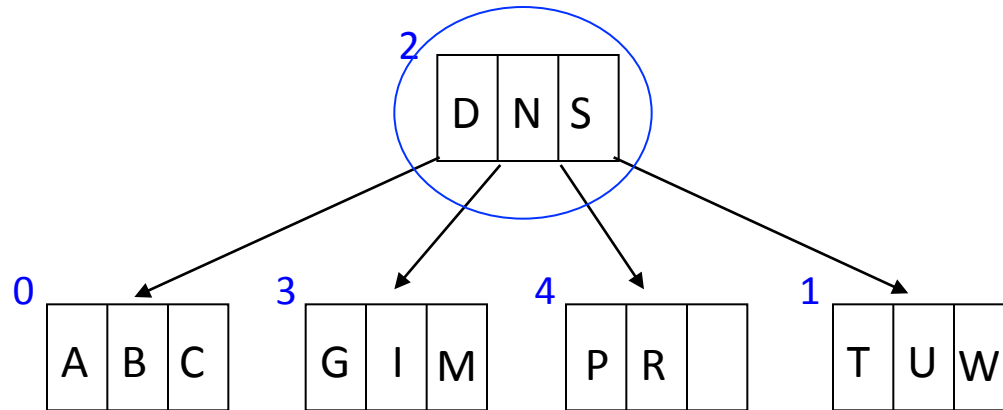
Busca da Chave K



- search (PAGE.CHILD[2], K, FOUND_RRN, FOUND_POS)

PAGE.CHILD[2] = NIL → chave de busca não encontrada
return NOT FOUND

Busca da Chave M



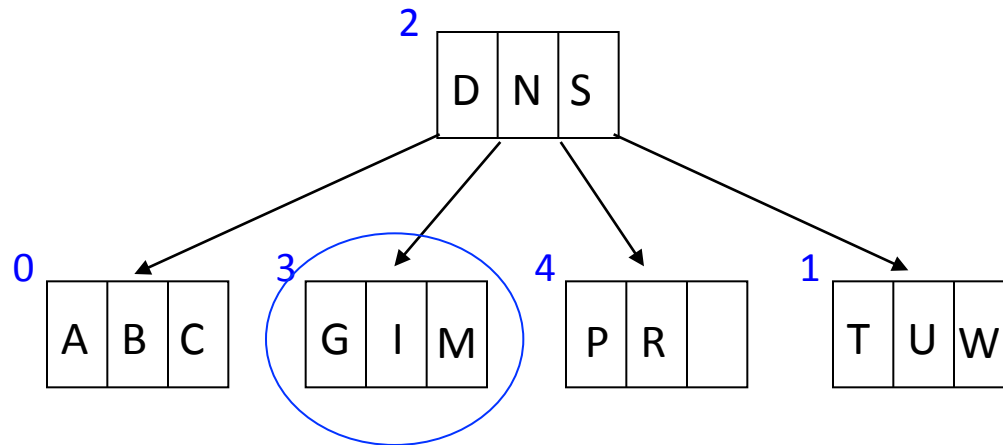
- search (2, M, FOUND_RRN, FOUND_POS)

PAGE =

D	N	S
---	---	---

 não existe → POS = 1

Busca da Chave M



- search (PAGE.CHILD[1], M, FOUND_RRN, FOUND_POS)

chave de busca encontrada

PAGE =

G	I	M
---	---	---

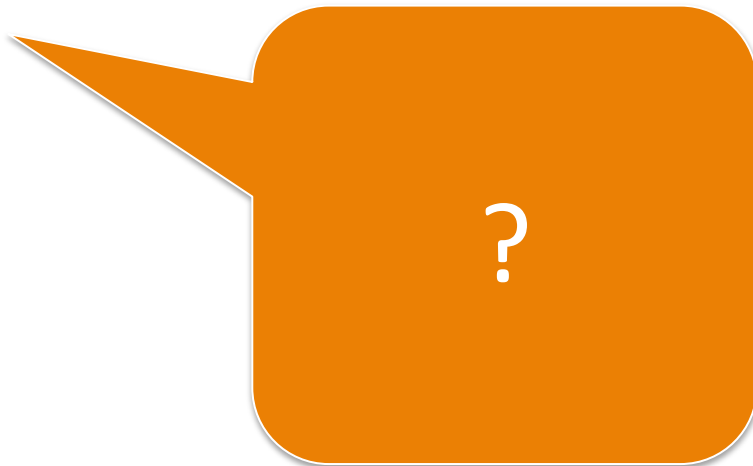
 POS = FOUND_POS = 2
FOUND_RRN = 3
return FOUND

Tópicos

- Árvore de Pesquisa
- + **Árvore B**
 - Características
 - Inserção
 - Pesquisa
 - **Remoção**
 - Análise
- Árvore B*
- Árvore B+

Remoção: Caso 1

- Remoção de uma chave em um nó folha, sem causar *underflow*
 - situação mais simples possível
- Solução

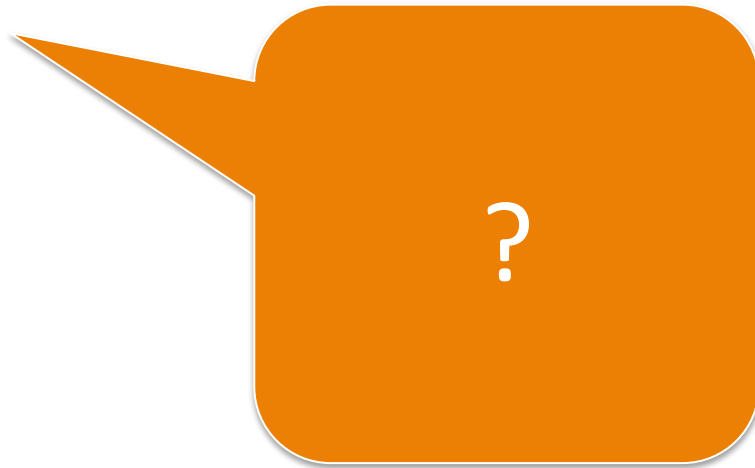


Remoção: Caso 1

- Remoção de uma chave em um nó folha, sem causar *underflow*
 - situação mais simples possível
- Solução
 - eliminar a chave da página
 - rearranjar as chaves remanescentes dentro da página para fechar o espaço liberado

Remoção: Caso 2

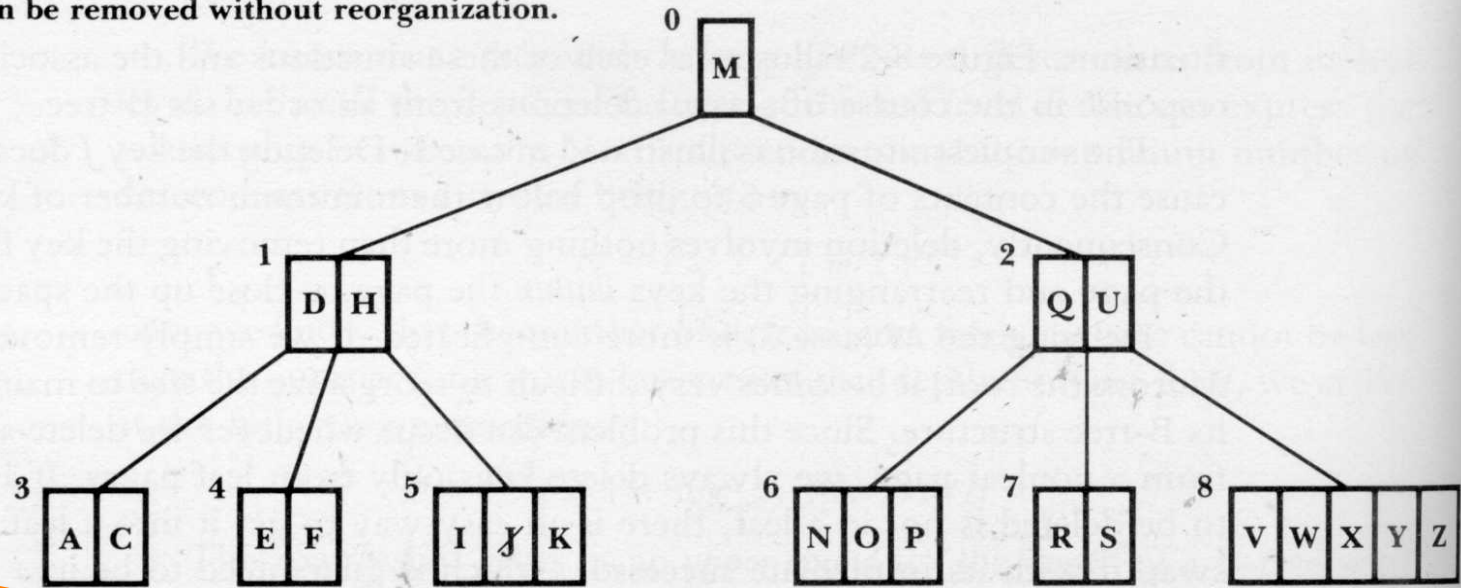
- Remoção de uma chave em um nó não folha
- Solução



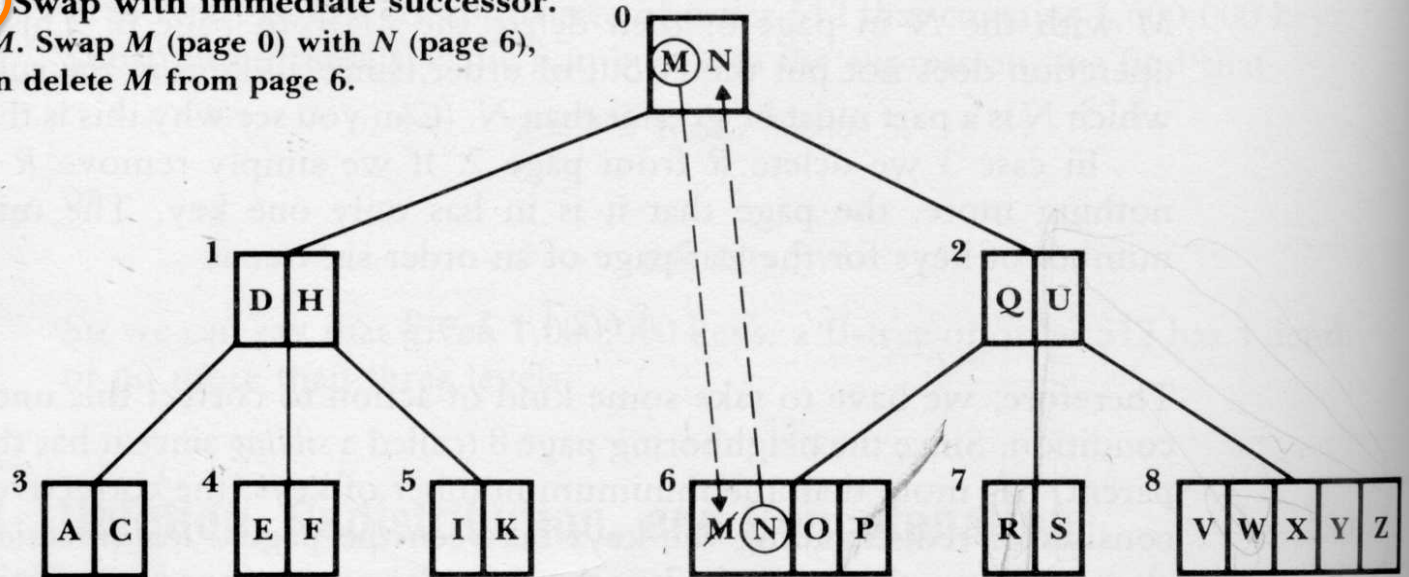
Remoção: Caso 2

- Remoção de uma chave em um nó não folha
- Solução
 - sempre remover chaves somente nas folhas
- Passos
 - trocar a chave a ser removida com a sua chave sucessora imediata (que está em um nó folha)
 - remover a chave diretamente do nó folha

Case 1: NO action.
Delete *J* from page 5. Since page 5 has more than the minimum number of keys, *J* can be removed without reorganization.

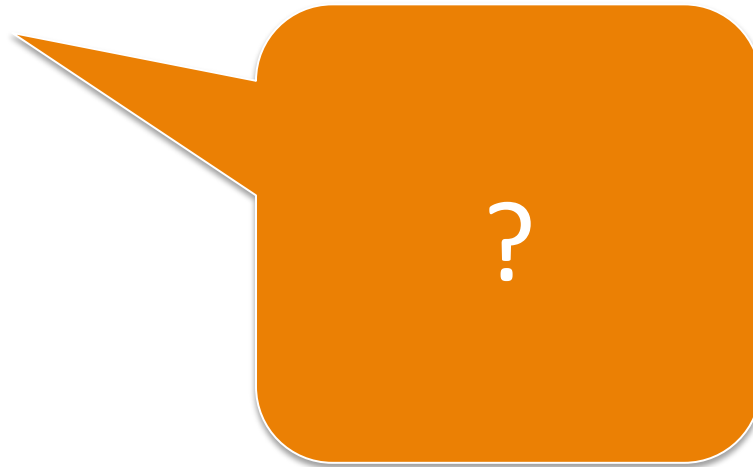


Case 2: Swap with immediate successor.
Delete *M*. Swap *M* (page 0) with *N* (page 6), and then delete *M* from page 6.



Remoção: Caso 3

- Remoção de uma chave em um nó, causando *underflow*
- Solução:



Remoção: Caso 3

- Remoção de uma chave em um nó, causando *underflow*
- Solução: Redistribuição
 - procurar uma página irmã (i.e., que possui o mesmo pai) adjacente que contenha mais chaves do que o mínimo
 - se encontrou
 - redistribuir as chaves entre as páginas
 - reacomodar a chave separadora, modificando o conteúdo do nó pai

Quando o nó possui menos que a metade cheio.

Case 3: Redistribution.

Delete *R*. Underflow occurs. Redistribute keys among pages 2, 7, and 8 to restore balance between leaves.

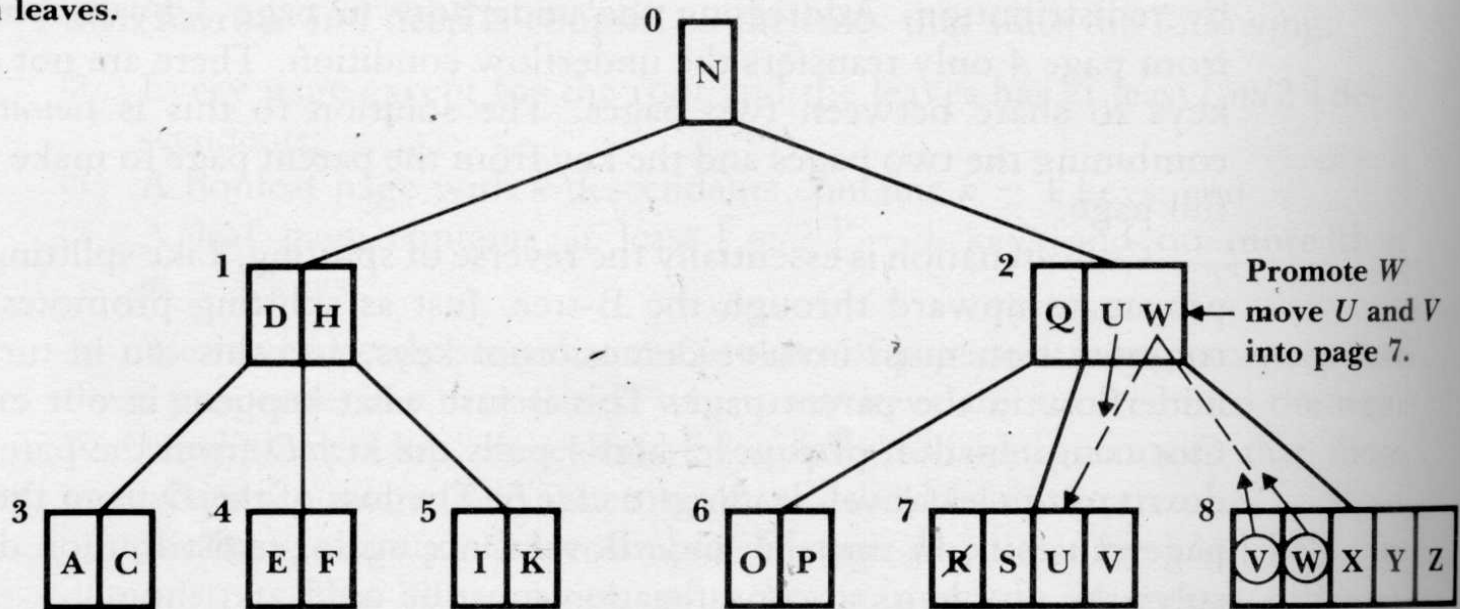
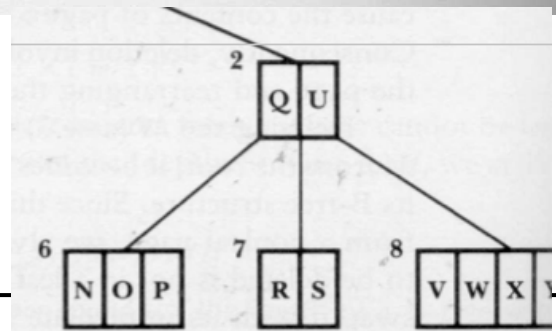


FIGURE 8.29 Six situations that can occur during deletions.

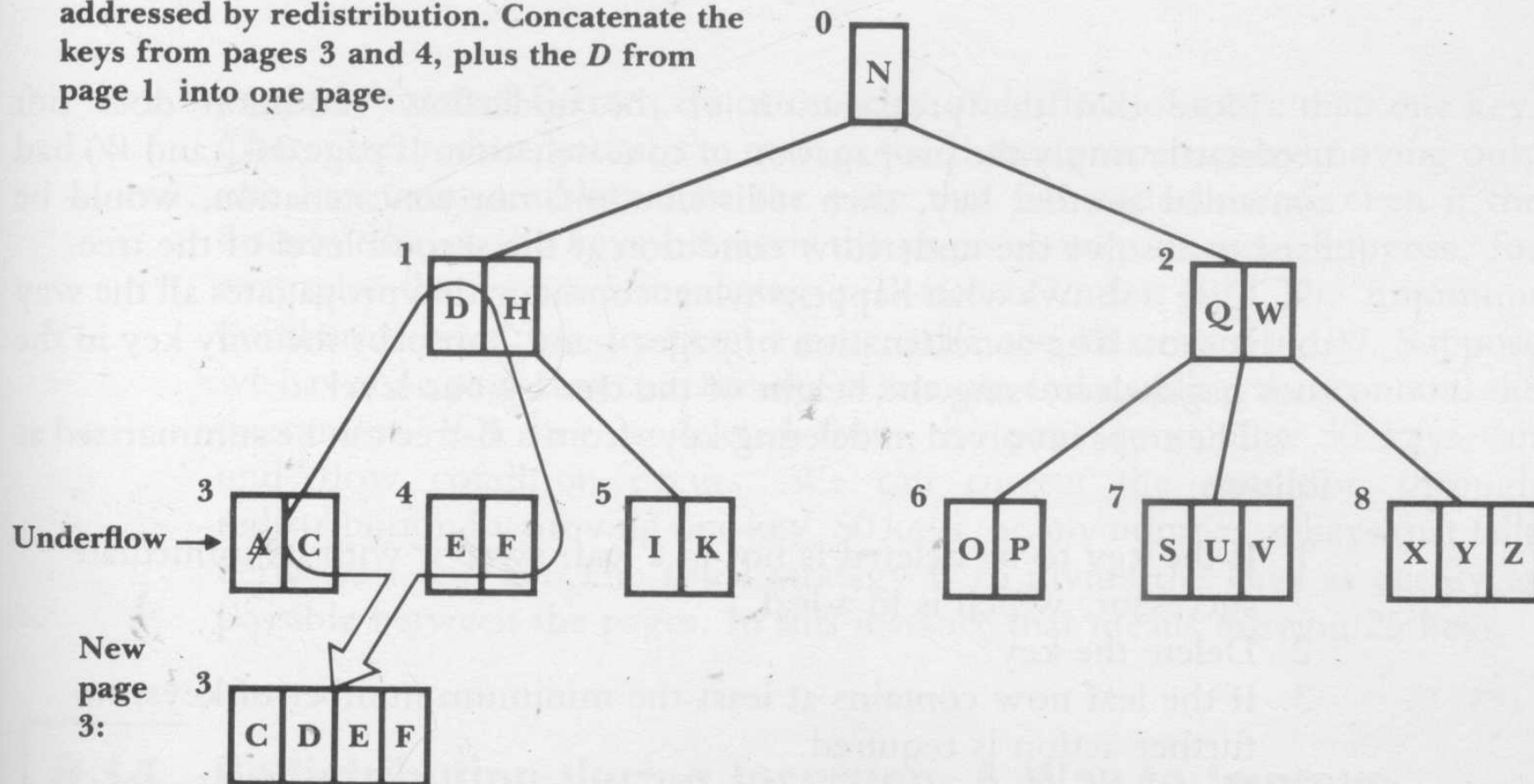


Remoção: Caso 4

- Remoção de uma chave em um nó, causando *underflow* e a **redistribuição não** pode ser aplicada
- Solução: Concatenação
 - combinar para formar uma nova página
 - o conteúdo do nó que sofreu *underflow*
 - o conteúdo de um nó irmão adjacente
 - a chave separadora no nó pai
 - tratar o *underflow* no nó pai, caso necessário

Case 4: Concatenation.

Delete A. Underflow occurs, but it cannot be addressed by redistribution. Concatenate the keys from pages 3 and 4, plus the D from page 1 into one page.



Concatenação

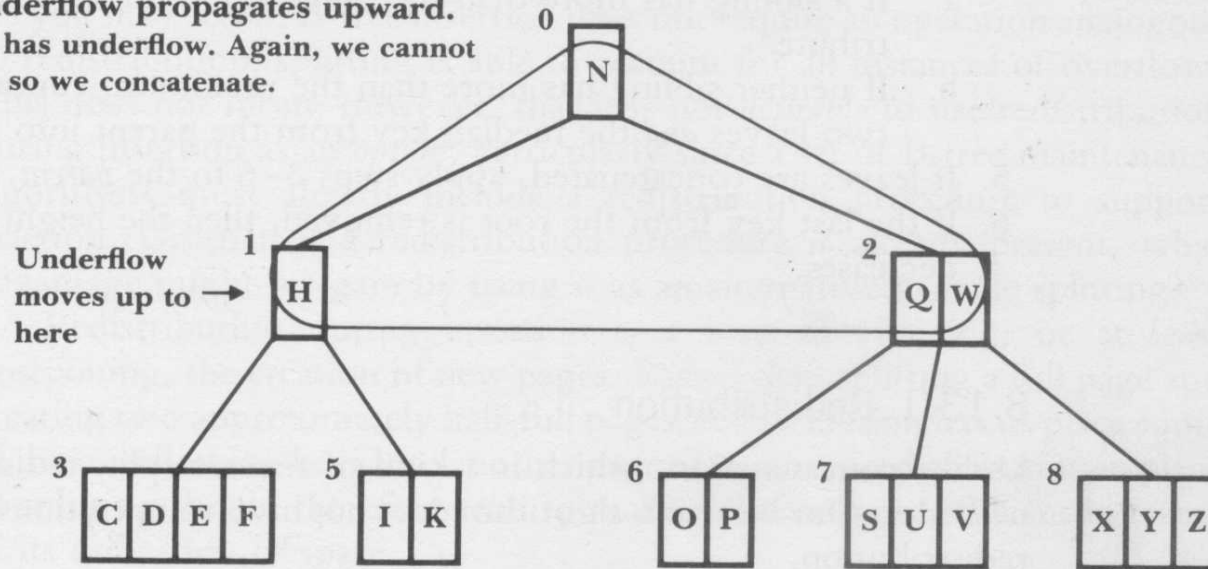
- Processo inverso do *split*
 - Características
 - reverte a promoção de uma chave
 - pode causar *underflow* no nó pai
- ⇒ concatenação pode ser propagada em direção ao nó raiz

ocorre a redução no número total de nós da árvore

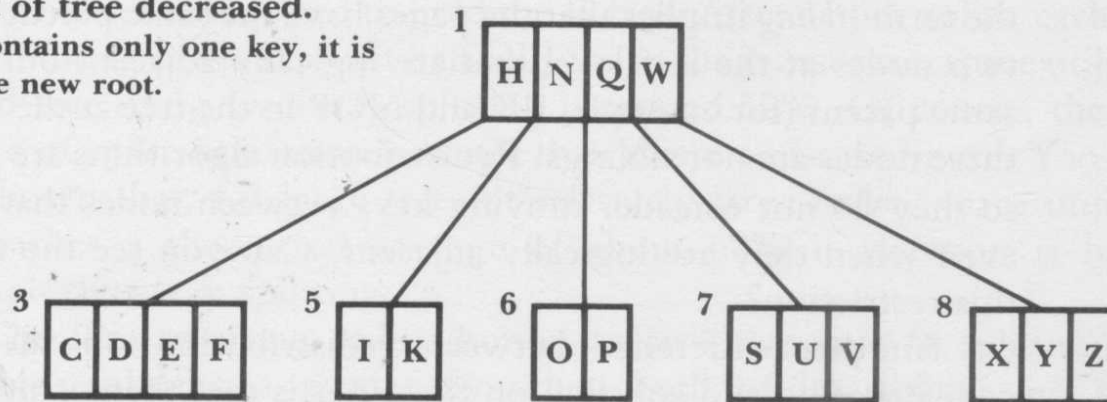
Remoção: Caso 5

- *Underflow* no **nó pai** causado pela remoção de uma chave em um nó filho
- Solução
 - utilizar redistribuição ou concatenação, dependendo da quantidade de chaves que a página irmã adjacente contém

Case 5: Underflow propagates upward.
 Now page 1 has underflow. Again, we cannot redistribute, so we concatenate.



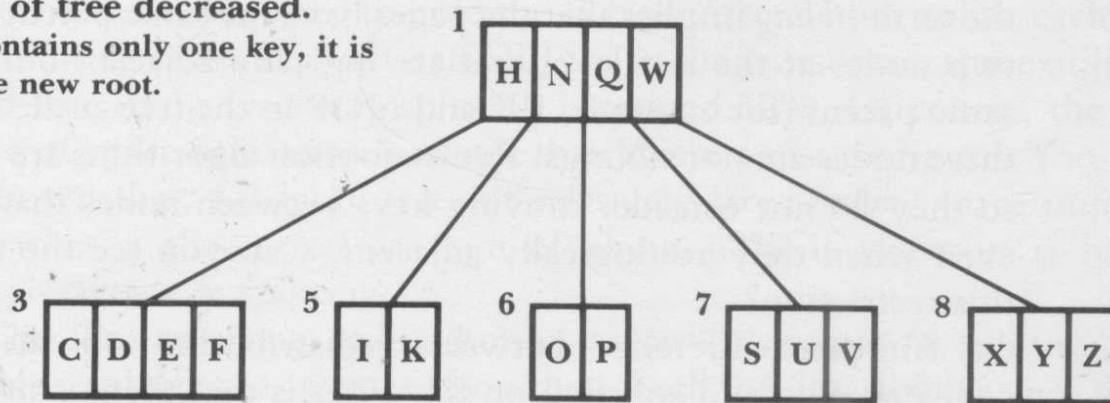
Case 6: Height of tree decreased.
 Since the root contains only one key, it is absorbed into the new root.



Remoção: Caso 6

- Diminuição da altura da árvore
- Característica
 - o nó raiz possui uma única chave
 - a chave é absorvida pela concatenação de seus nós filhos
- Solução
 - eliminar a raiz antiga
 - tornar no nó resultante da concatenação dos nós filhos a nova raiz da árvore

Case 6: Height of tree decreased.
Since the root contains only one key, it is absorbed into the new root.



Remoção em Árvore-B

1. se a chave a ser removida não estiver em um nó folha, troque-a com sua sucessora imediata, que está em um nó folha
2. remova a chave
3. após a remoção, se o nó satisfaz o número mínimo de chaves, nenhuma ação adicional é requerida

Remoção em Árvore-B

4. após a remoção, caso ocorra *underflow*, verifique o número de chaves nos nós irmãos adjacentes à esquerda e à direita
 - a. se algum nó irmão adjacente possui mais do que o número mínimo de chaves, aplique a redistribuição
 - b. se nenhum nó irmão adjacente possui mais do que o número mínimo de chaves, aplique a concatenação

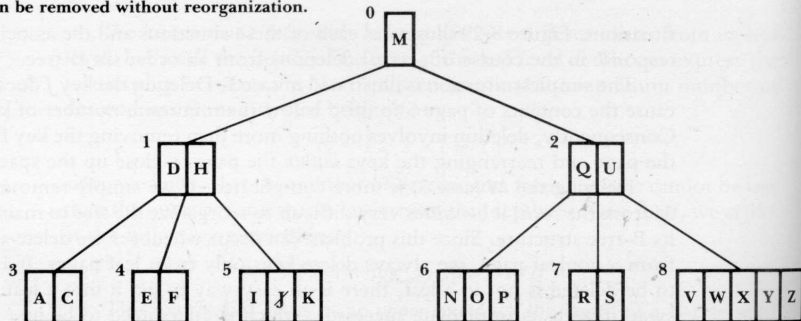
Remoção em Árvore-B

5. se ocorreu concatenação, repita os passos 3 a 5 para o nó pai
6. se a última chave da raiz for removida, a altura da árvore é diminuída

Remoção: Diferentes Casos

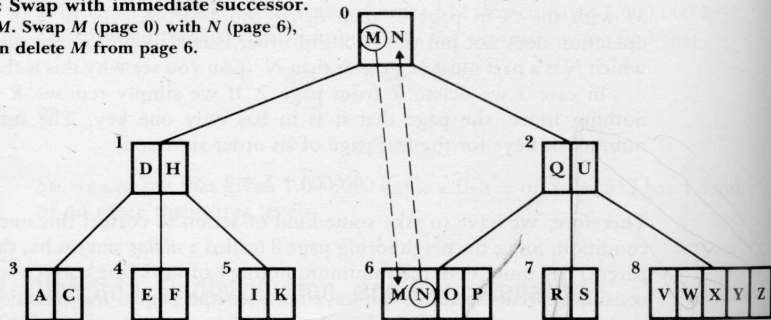
Case 1: No action.

Delete *J* from page 5. Since page 5 has more than the minimum number of keys, *J* can be removed without reorganization.



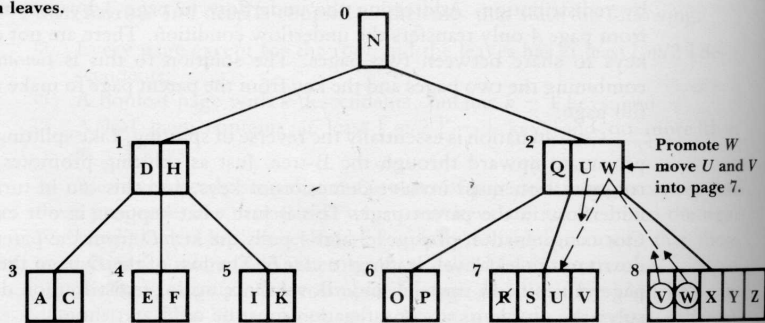
Case 2: Swap with immediate successor.

Delete *M*. Swap *M* (page 0) with *N* (page 6), and then delete *M* from page 6.



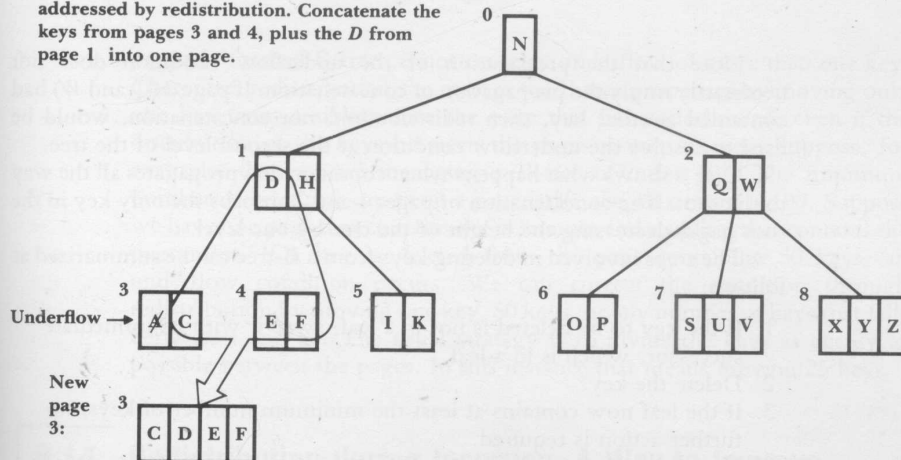
Case 3: Redistribution.

Delete *R*. Underflow occurs. Redistribute keys among pages 2, 7, and 8 to restore balance between leaves.



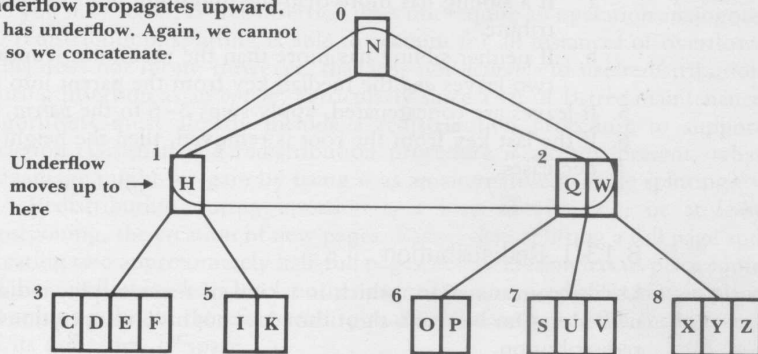
Case 4: Concatenation.

Delete *A*. Underflow occurs, but it cannot be addressed by redistribution. Concatenate the keys from pages 3 and 4, plus the *D* from page 1 into one page.



Case 5: Underflow propagates upward.

Now page 1 has underflow. Again, we cannot redistribute, so we concatenate.



Case 6: Height of tree decreased.

Since the root contains only one key, it is absorbed into the new root.

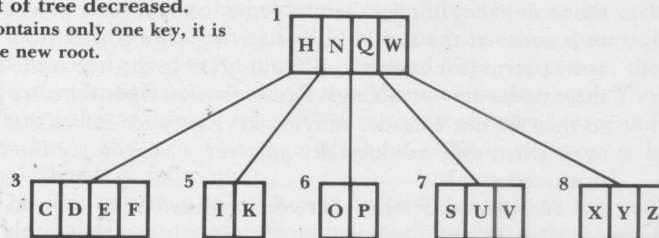


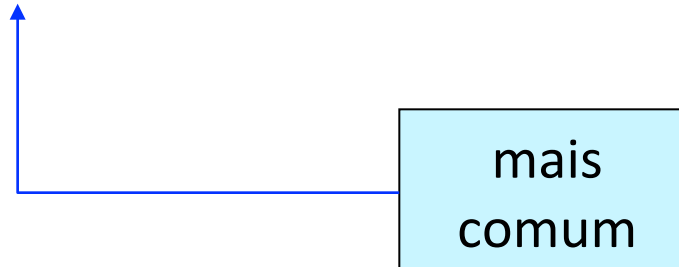
FIGURE 8.29 Six situations that can occur during deletions.

Redistribuição

- Representa uma idéia inovadora
 - diferente do *split* ou da concatenação
- Não se propaga para os nós superiores
 - apenas efeito local na árvore
- Baseada no conceito de nós irmãos adjacentes
 - dois nós logicamente adjacentes, mas com pais diferentes não são irmãos

Redistribuição

- Não fixa a forma na qual as chaves devem ser redistribuídas
 - possibilidade 1: mover somente uma chave, mesmo que a distribuição das chaves entre as páginas não seja uniforme
 - possibilidade 2: mover k chaves
 - possibilidade 3: distribuição uniforme das chaves entre os nós



Redistribuição durante Inserção

- Funcionalidade
 - permite melhorar a taxa de utilização do espaço alocado para a árvore
- *split*
 - divide uma página com *overflow* (i.e., working page) em duas páginas semi-vazias (i.e., page e newpage)
- redistribuição
 - a chave que causou *overflow* (além de outras chaves) pode ser colocada em outra página

X

Redistribuição durante Inserção

- Opção interessante
 - a rotina de redistribuição já está codificada para prover suporte à remoção
 - a redistribuição evita, ou pelo menos adia, a criação de novas páginas
 - tende a tornar a árvore-B mais eficiente em termos de utilização do espaço em disco
 - garante um melhor desempenho na busca, desde que um número menor de nós pode reduzir a altura da árvore, por exemplo

Split x Redistribuição

- Somente *split* na inserção
 - no pior caso, a utilização do espaço é de cerca de 50%
 - em média, para árvores grandes, o índice de ocupação é de ~69%
- Com redistribuição na inserção
 - em média, para árvores grandes, o índice de ocupação é de ~86%

Tópicos

- Árvore de Pesquisa

+ **Árvore B**

– Características

– Inserção

– Pesquisa

– Remoção

– **Análise**

- Árvore B*

- Arvore B+

Complexidade

- Profundidade do caminho de busca
 - número máximo de acessos a disco
 - Relacionamento
 - tamanho da página de disco
 - ex: árvore-B de ordem 512 → 511 chaves/página
 - número de chaves
 - ex: 1.000.000 de chaves
- ⇒ número de níveis que pode ser atingido?

pior caso

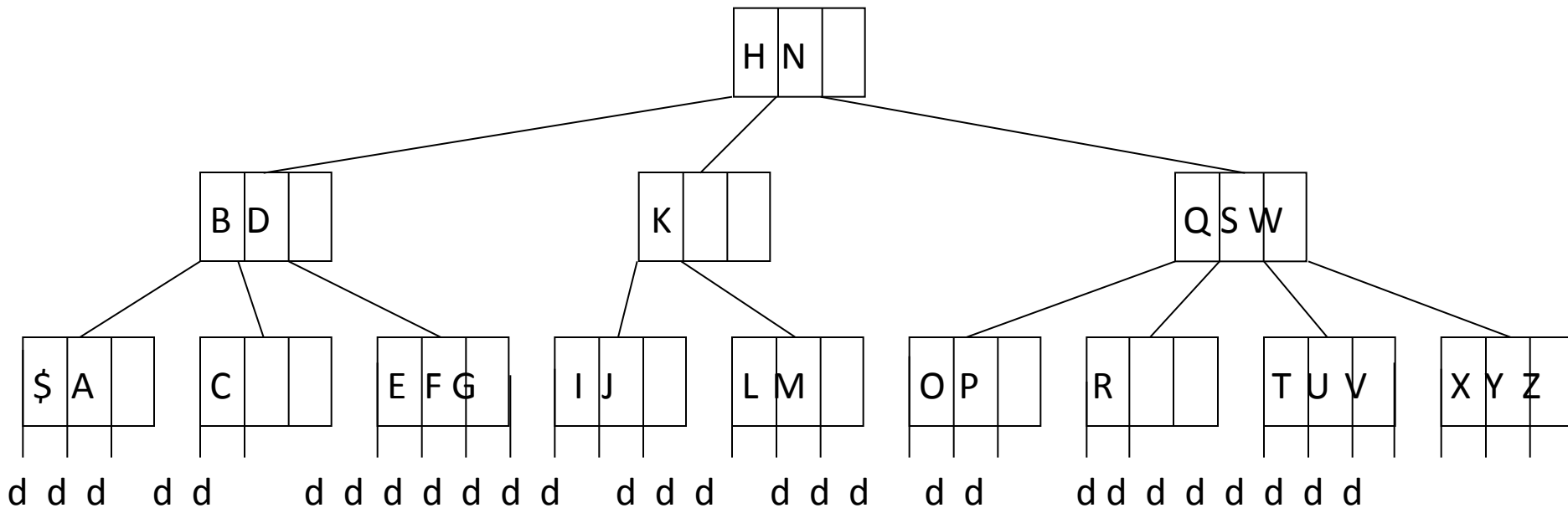
Complexidade

- Profundidade do caminho de busca
 - número máximo de acessos a disco
- Relacionamento
 - tamanho da página de disco
 - ex: árvore-B de ordem 512 → 511 chaves/página

pior caso

Observação 1

número de descendentes de um nível da árvore-B = número de chaves contidas no nível em questão e em todos os níveis acima + 1



árvore-B com 27 chaves e 28 descendentes

Observação 2

- Propriedades da árvore-B de ordem m
 - cálculo do número mínimo de descendentes de um nível

nível	número mínimo de descendentes
1	2
2	$2 \times \lceil m/2 \rceil$
3	$2 \times \lceil m/2 \rceil \times \lceil m/2 \rceil = 2 \times \lceil m/2 \rceil^2$
4	$2 \times \lceil m/2 \rceil \times \lceil m/2 \rceil \times \lceil m/2 \rceil = 2 \times \lceil m/2 \rceil^3$
...	...
d	$2 \times \lceil m/2 \rceil^{d-1}$

para qualquer nível d , com exceção da raiz (nível 1)

Complexidade

- Número de chaves (N)
 - $N + 1$ descendentes no nível das folhas
- Profundidade da árvore-B no nível das folhas
 - d
- Relacionamento
 - $N + 1$ descendentes e
 - número mínimo de descendentes da árvore-B com profundidade d

Complexidade

$$N + 1 \geq 2 \times \lceil m/2 \rceil^{d-1}$$
$$d \leq 1 + \log_{\lceil m/2 \rceil} \left((N + 1)/2 \right)$$

- Exemplo

- $m = 512$

- $N = 1.000.000$

- $d \leq 1 + \log_{256} (500.000,50) \Rightarrow d \leq 3,37$

- acesso a disco adicional: arquivo de dados

a árvore possui não mais do que 3 níveis de altura



Tópicos

- Árvore de Pesquisa
- Árvore B
- + Árvore B*
 - Características
 - Diferenças
- Arvore B+

Árvore-B*

- Proposta por Knuth em 1973
 - variação de árvore-B
- Característica
 - cada nó contém, no mínimo, $2/3$ do número máximo de chaves
- Posterga o *split*
 - estende a noção de redistribuição durante a inserção para incluir novas regras para o particionamento de nós



Árvore-B*

- Proposta por Knuth em 1973
 - variação de árvore-B
- Mínimo de chave
- Posterga o *split*



Efeito disso ?

Árvore-B*

- Geração
 - processo de subdivisão
- Características
 - a subdivisão é adiada até que duas páginas irmãs estejam cheias
 - na seqüência, a divisão do conteúdo das duas páginas em três páginas (*two-to-three split*) é realizada

Tópicos

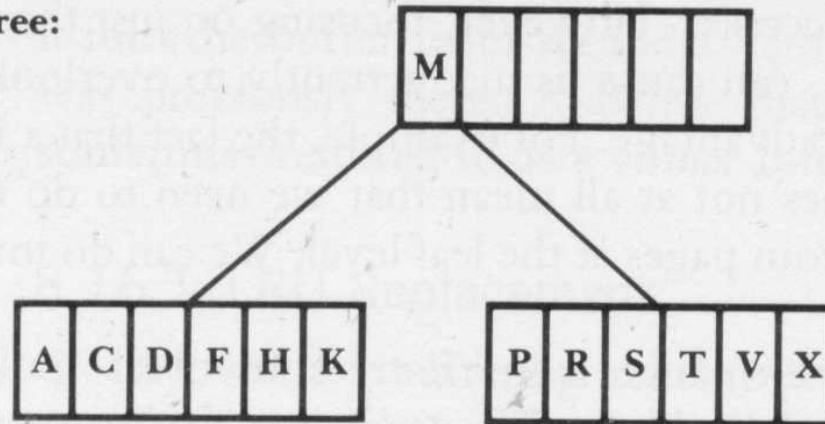
- Árvore de Pesquisa
- Árvore B
- + Árvore B*
 - Características
 - Diferenças
- Arvore B+

Uso da Redistribuição

- Situações (árvore-B e árvore-B*)
 - diferem das árvores B em relação ao particionamento de suas páginas
- árvore-B
 - *split* 1-to-2
- árvore-B*
 - *split* 2-to-3
 - pelo menos um nó irmão está cheio

Split 2-to-3

Original tree:



Two-to-three-split:
After the insertion of the
key *B*.

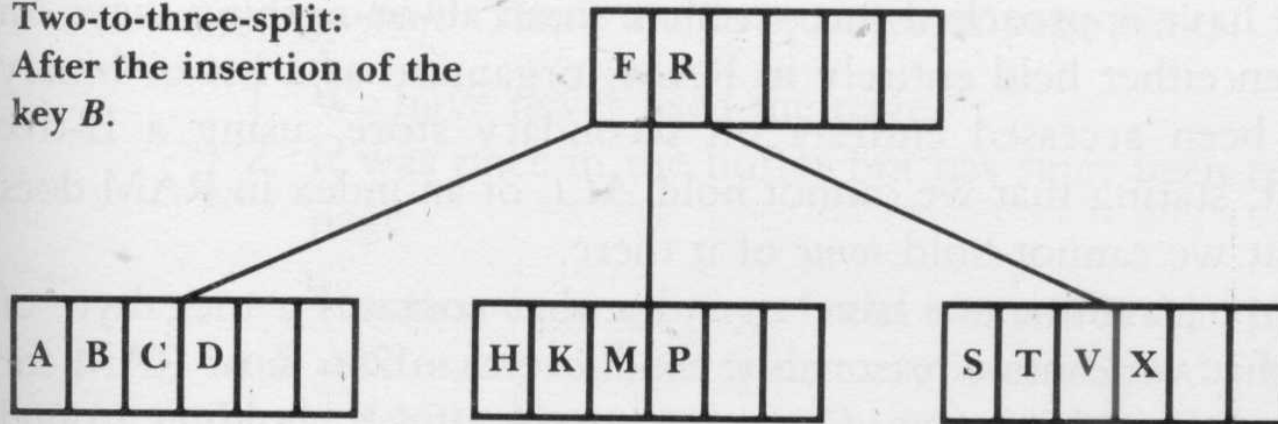


FIGURE 8.30 A two-to-three split.

Definição Formal

- Propriedades de uma Árvore-B*
 - cada página possui um máximo de m descendentes
 - cada página, exceto a raiz e as folhas, possui no mínimo $(2m-1)/3$ descendentes → taxa de ocupação
 - a raiz possui pelo menos 2 descendentes, a menos que seja um nó folha
 - todas as folhas aparecem no mesmo nível
 - uma página interna com k descendentes contém $k-1$ chaves
 - uma folha possui no mínimo $\lfloor (2m-1)/3 \rfloor$ chaves e no máximo $m - 1$ chaves → taxa de ocupação

Observações


- **Mudança na taxa de ocupação**
 - afeta as rotinas de remoção e redistribuição
- **Particionamento da raiz**
 - problema
 - raiz não possui nó irmão
 - soluções
 - dividir a raiz usando a divisão convencional (*1-to-2 split*); ou
 - permitir que a raiz seja maior

Tópicos

- Árvore de Pesquisa
- Árvore B
- Árvore B*
- + Árvore B+
 - Inserção
 - Remoção

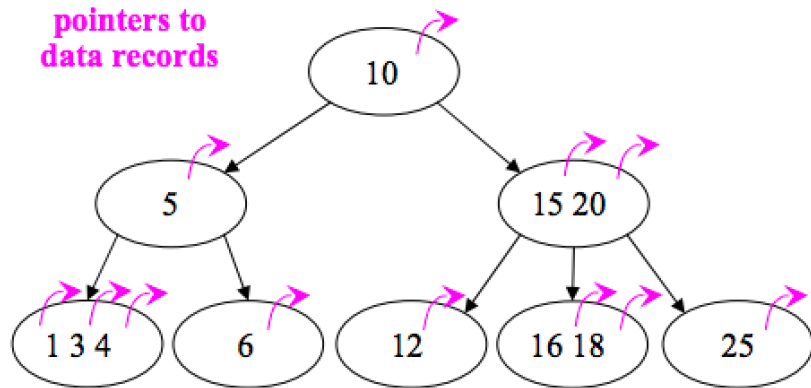
Sequence Sets

- Problema
 - manter os registros ordenados fisicamente pela chave (*sequence set*)
- Solução
 - organizar registros em blocos

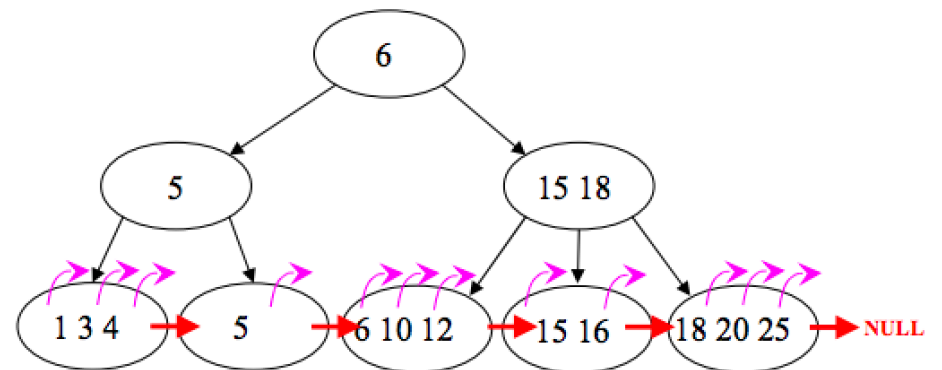


um bloco consiste na unidade básica de entrada e saída e deve ter seu tamanho determinado pelo tamanho do *buffer-pool*

B-tree of order 4



B⁺-tree of order 4



Sequence Sets

- Problema
 - manter os registros ordenados fisicamente pela chave (*sequence set*)
- Solução
 - organizar registros em blocos

Uso de Blocos

- Características
 - o conteúdo de cada bloco está ordenado, e pode ser recuperado em um acesso
 - cada bloco mantém um ‘ponteiro’ para o bloco antecessor e um ‘ponteiro’ para o bloco sucessor
 - blocos logicamente adjacentes não estão (necessariamente) fisicamente adjacentes
- Garante acesso sequencial ao arquivo

Problema 1

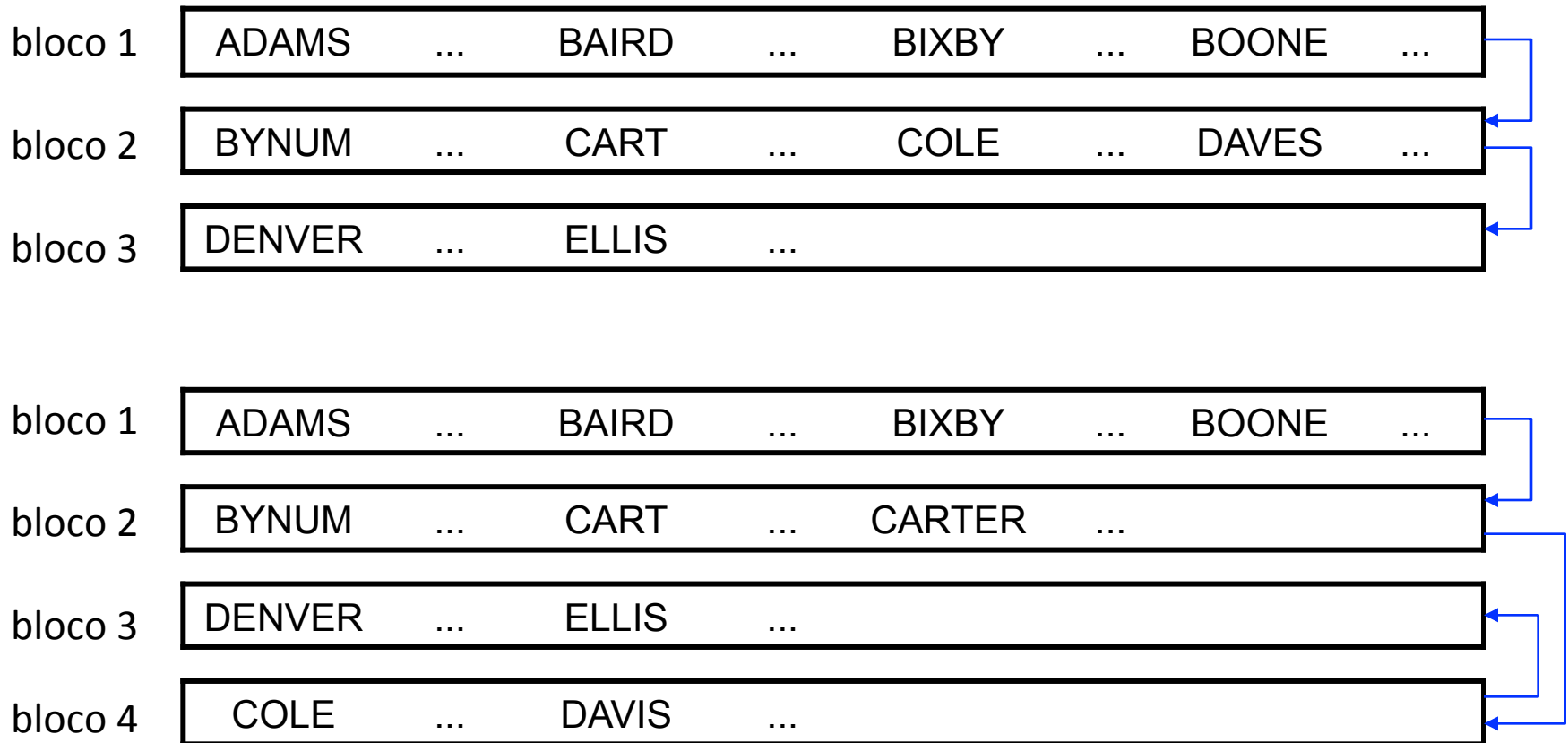
- Inserção de registros pode provocar *overflow* em um bloco
- Solução
 - dividir o bloco, em um processo análogo ao realizado em árvores-B
 - passos
 - divide os registros entre os dois blocos
 - rearranja os ponteiros

não existe
promoção !

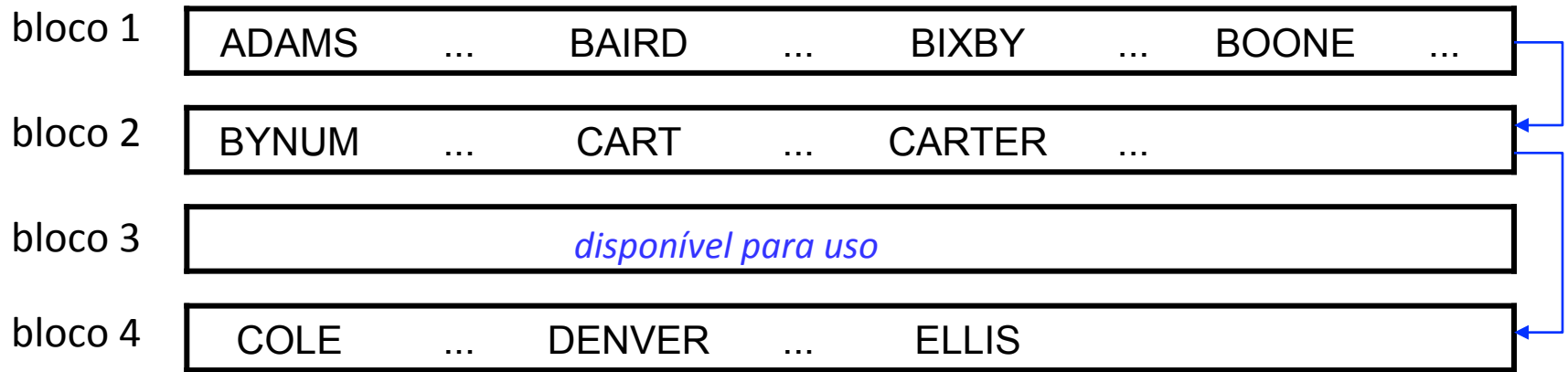
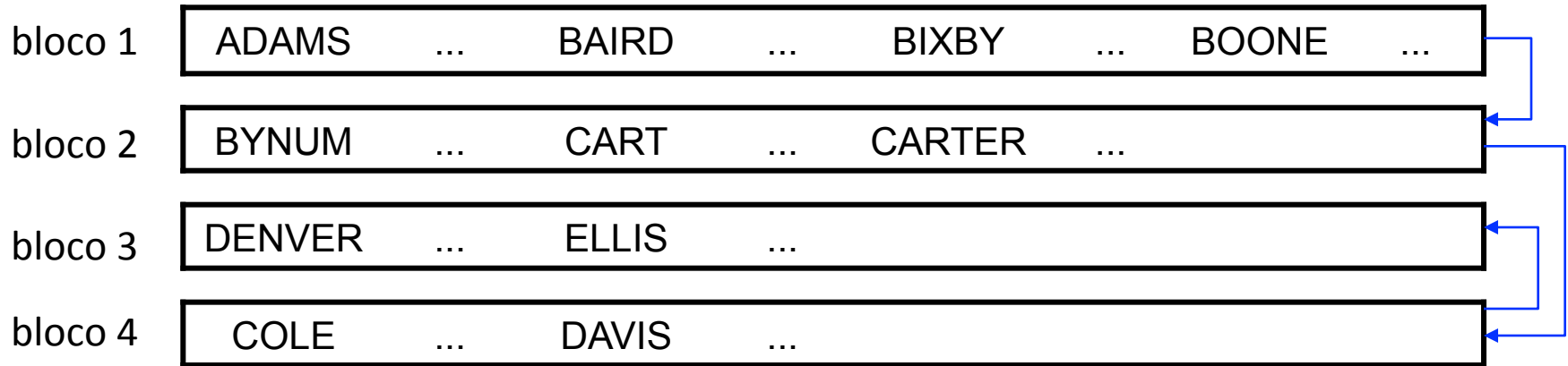
Problema 2

- Remoção de registros pode provocar *underflow* em um bloco
- Solução
 - concatenar o bloco com o seu antecessor ou sucessor na seqüência lógica
 - redistribuir os registros, movendo-os entre blocos logicamente adjacentes

Exemplo: Inserção de CARTER



Exemplo: Remoção de DAVIS



Indexação

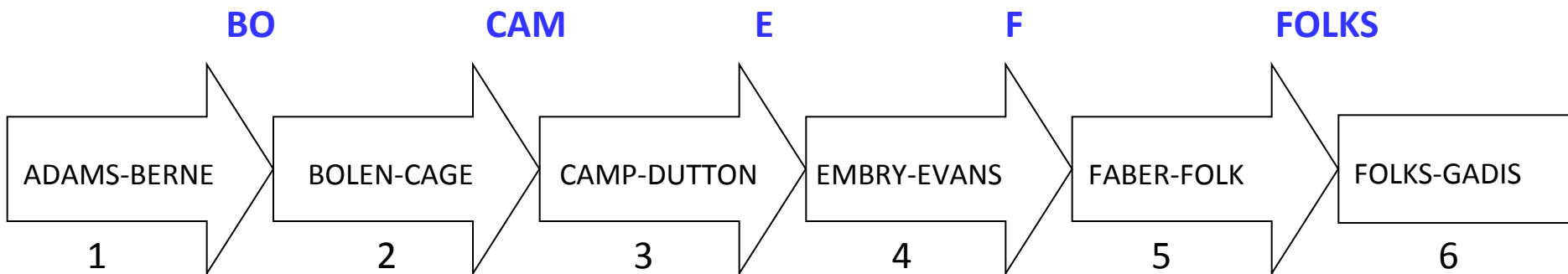
- Característica
 - os registros podem ser acessados em ordem, sequencialmente, pela chave
- Problema
 - localizar eficientemente um bloco com um registro particular, dado a chave do registro
- Uma solução (contexto da disciplina)
 - árvore-B+ (pré-fixada)

Árvore-B+ Pré-Fixada

- Estrutura híbrida
 - chaves
 - organizadas como árvore-B (i.e., *index set*)
 - nós folhas
 - consistem em blocos de *sequence set*
- Pré-fixada simples
 - armazena na árvore as cadeias separadoras mínimas entre cada par de blocos

Separadores

- Características
 - são mantidos no índice, ao invés das chaves de busca
 - possuem tamanho variável
- Exemplo



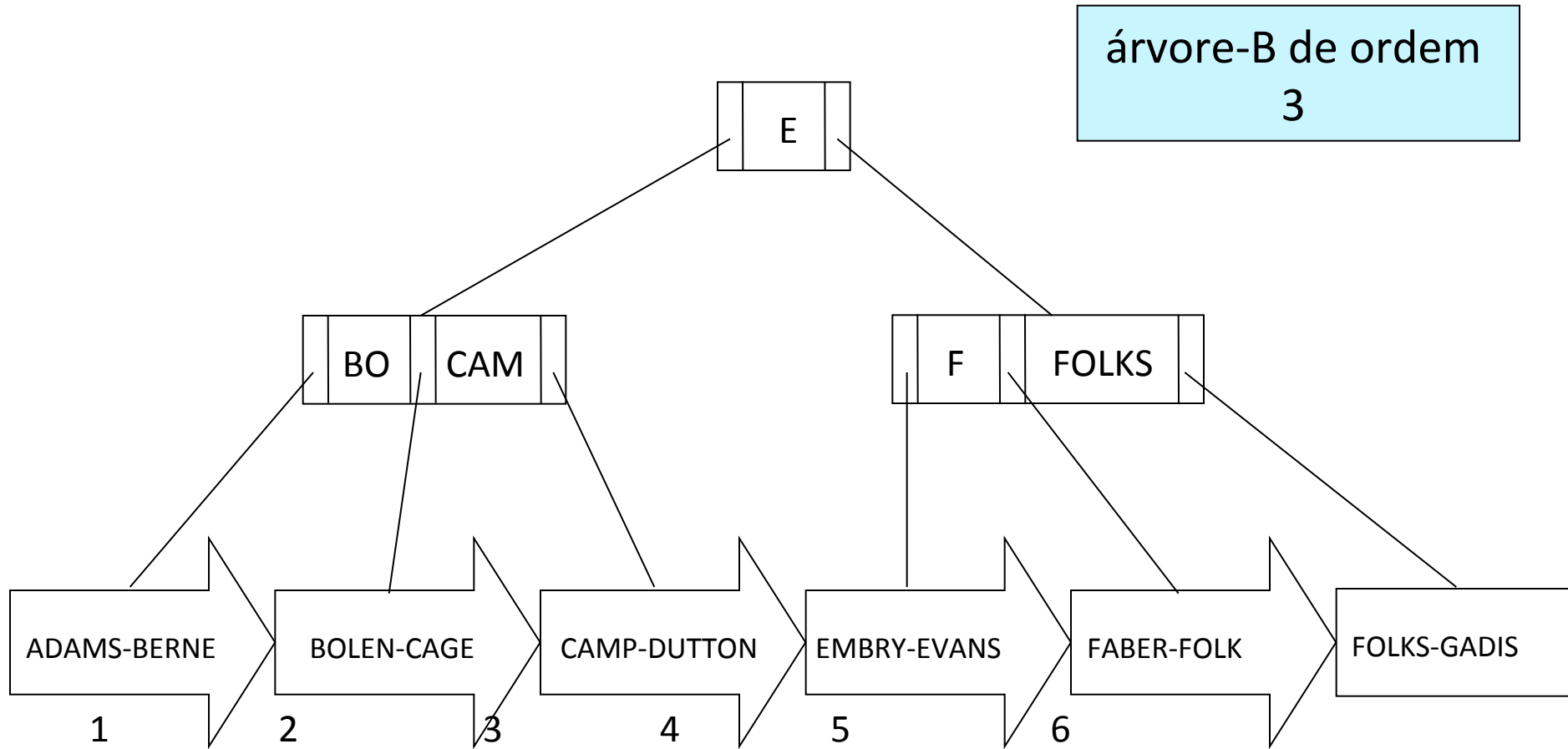
Separadores

- Desafio
 - escolher o menor separador para utilizar no índice
- Tabela de decisão

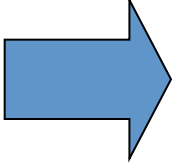
chave de busca x separador	decisão
chave < separador	procure à esquerda
chave = separador	procure à direita
chave > separador	procure à direita

Árvore-B+ Pré-Fixada

árvore-B de ordem
3



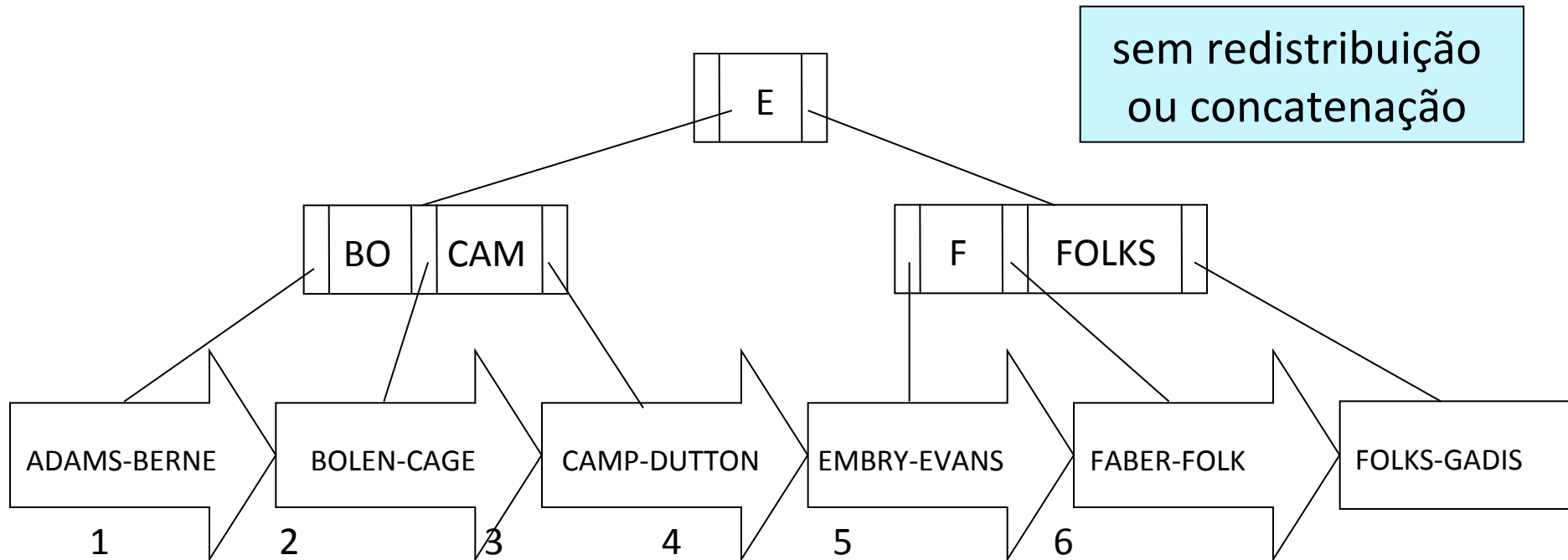
Manutenção

- Cenários
 - inserção
 - remoção
 - *overflow*
 - *underflow*
- 
- Efeitos colaterais
 - *sequence set*
 - árvore-B+

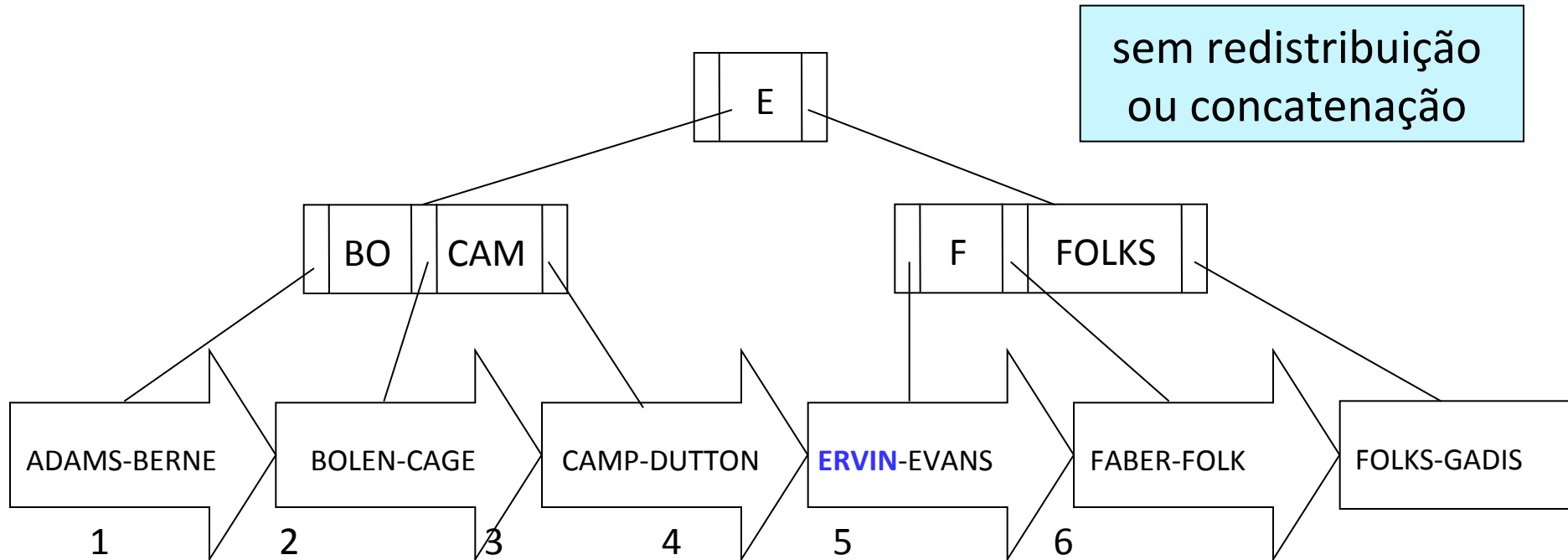
Tópicos

- Árvore de Pesquisa
- Árvore B
- Árvore B*
- + Árvore B+
 - Inserção
 - Remoção

Remoção de EMBRY

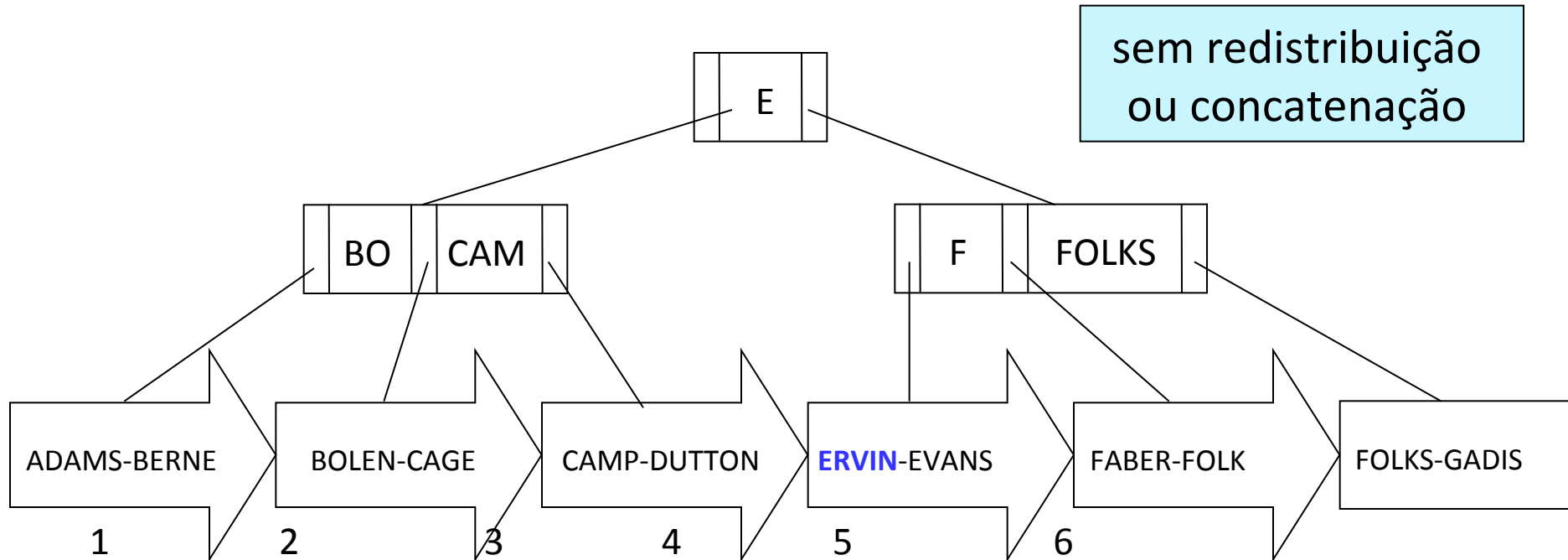


Remoção de EMBRY



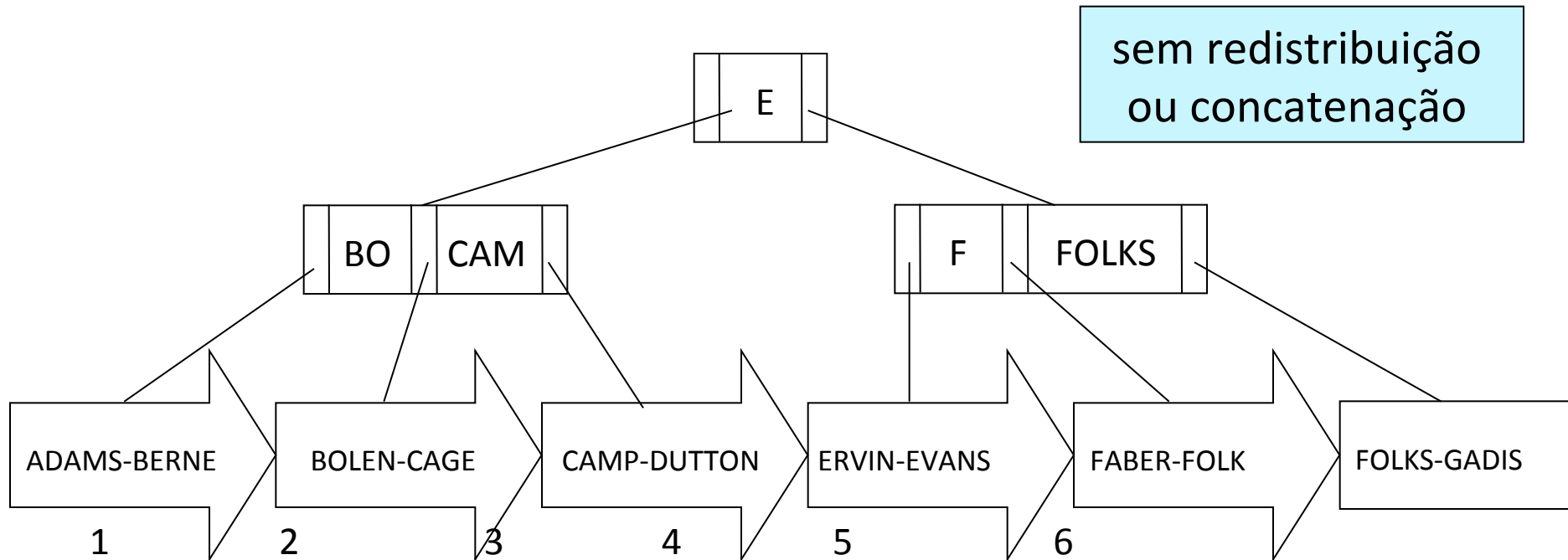
- Efeito no *sequence set*
 - limitado a alterações no bloco 4

Remoção de EMBRY

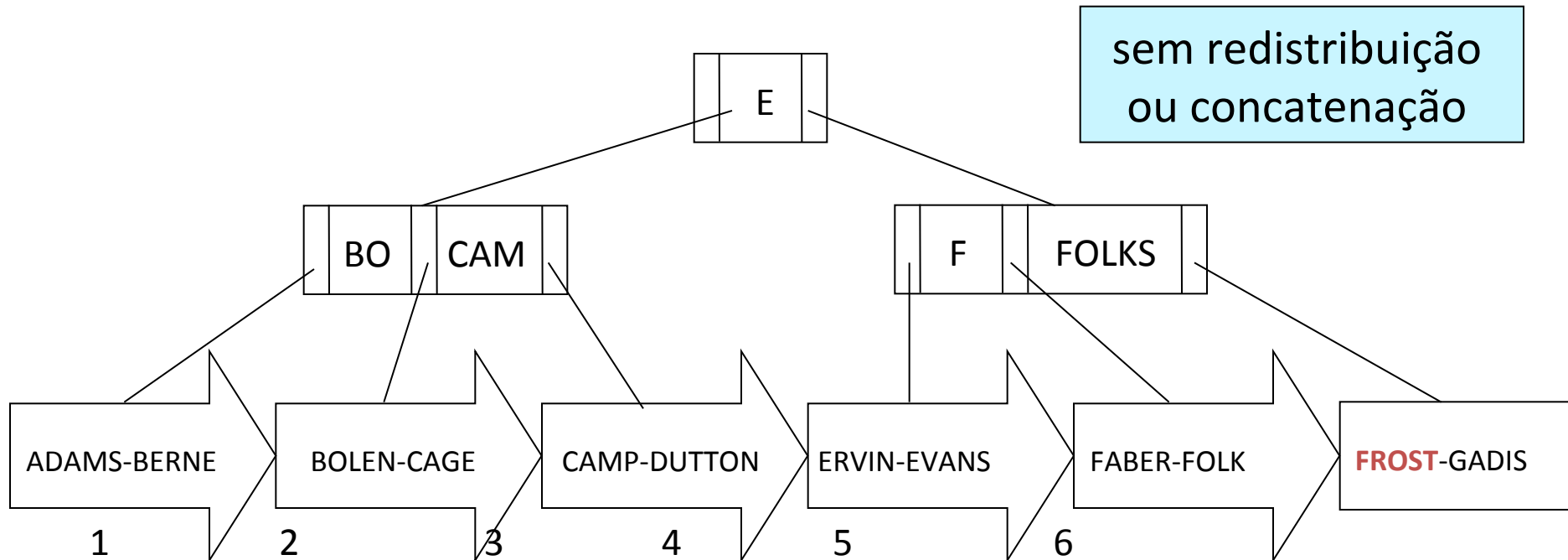


- Efeito na árvore-B+
 - nenhum: E é uma boa chave separadora

Remoção de FOLKS

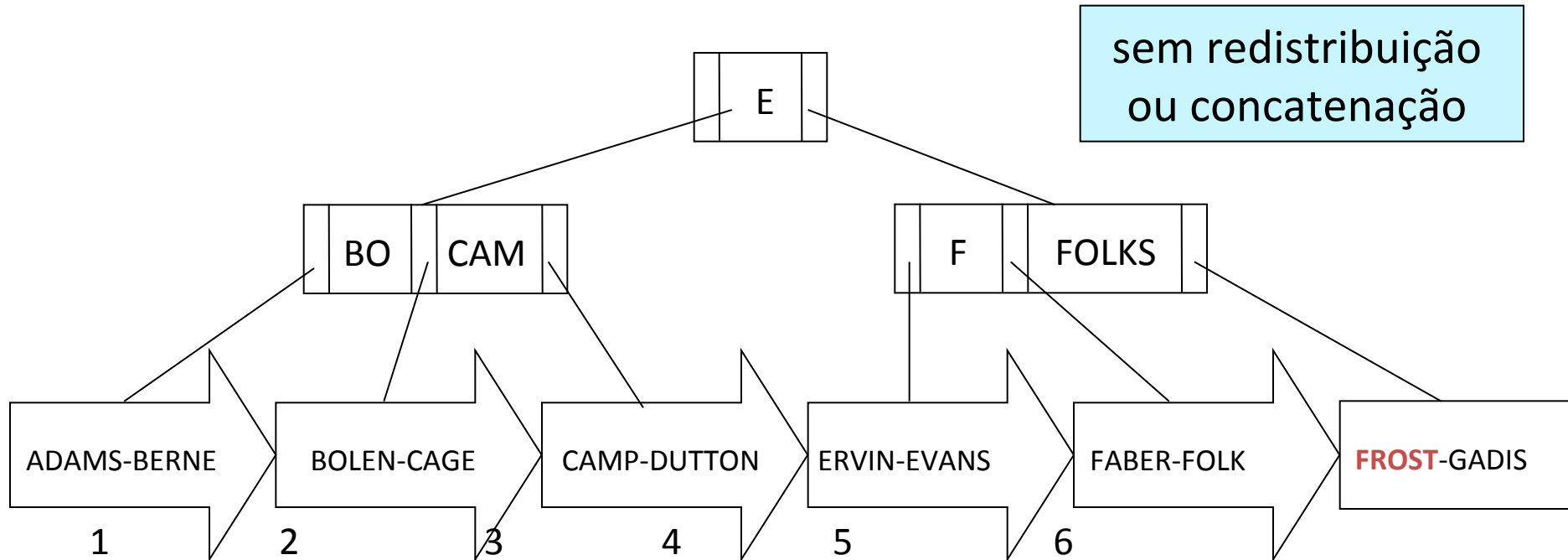


Remoção de FOLKS



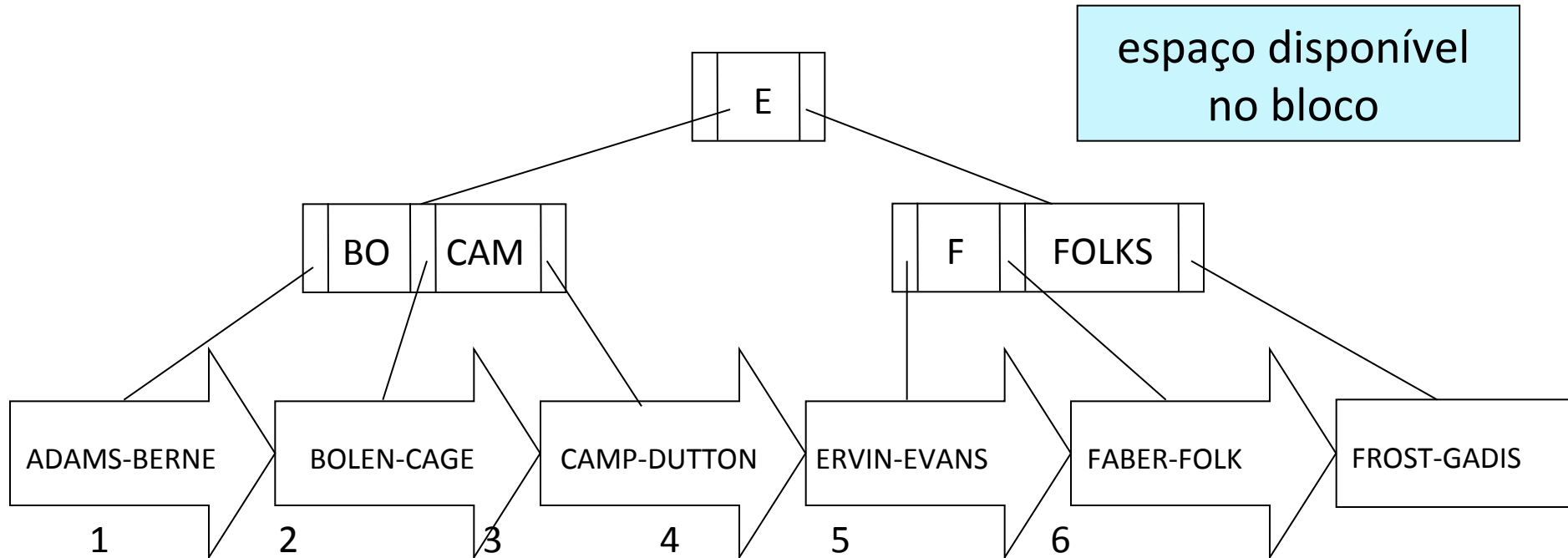
- Efeito no *sequence set*
 - limitado a alterações no bloco 6

Remoção de FOLKS

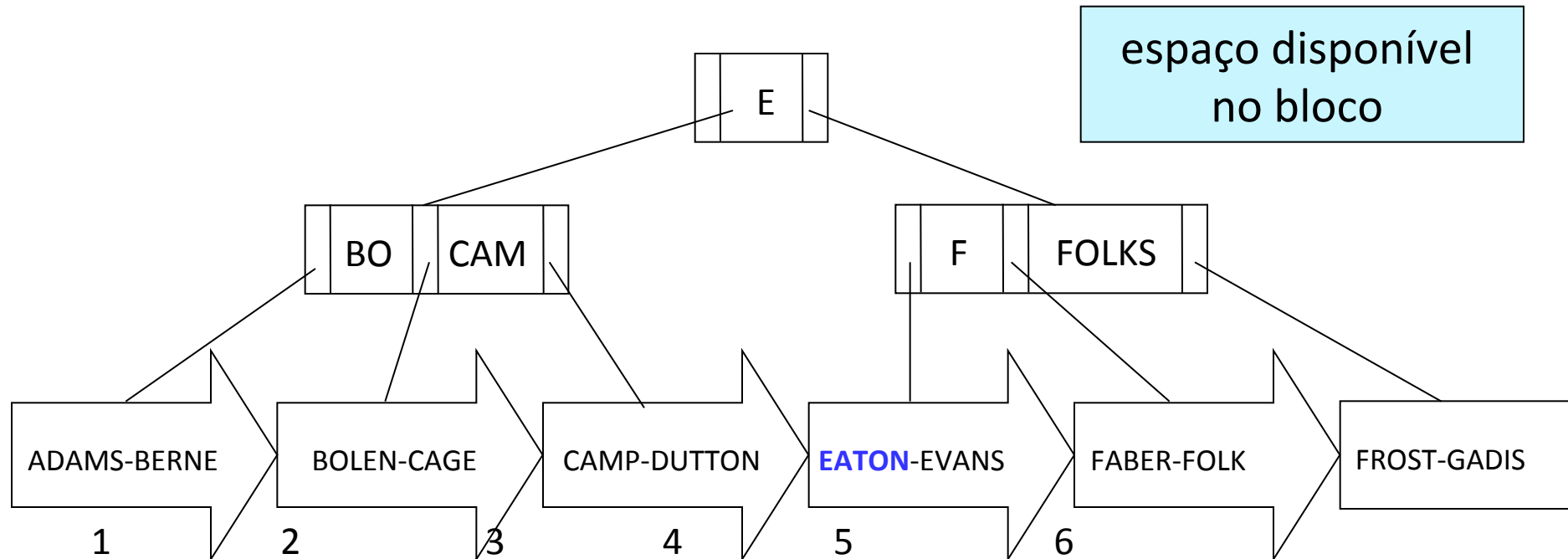


- Efeito na árvore-B+
 - nenhum: custos elevados

Inserção de EATON

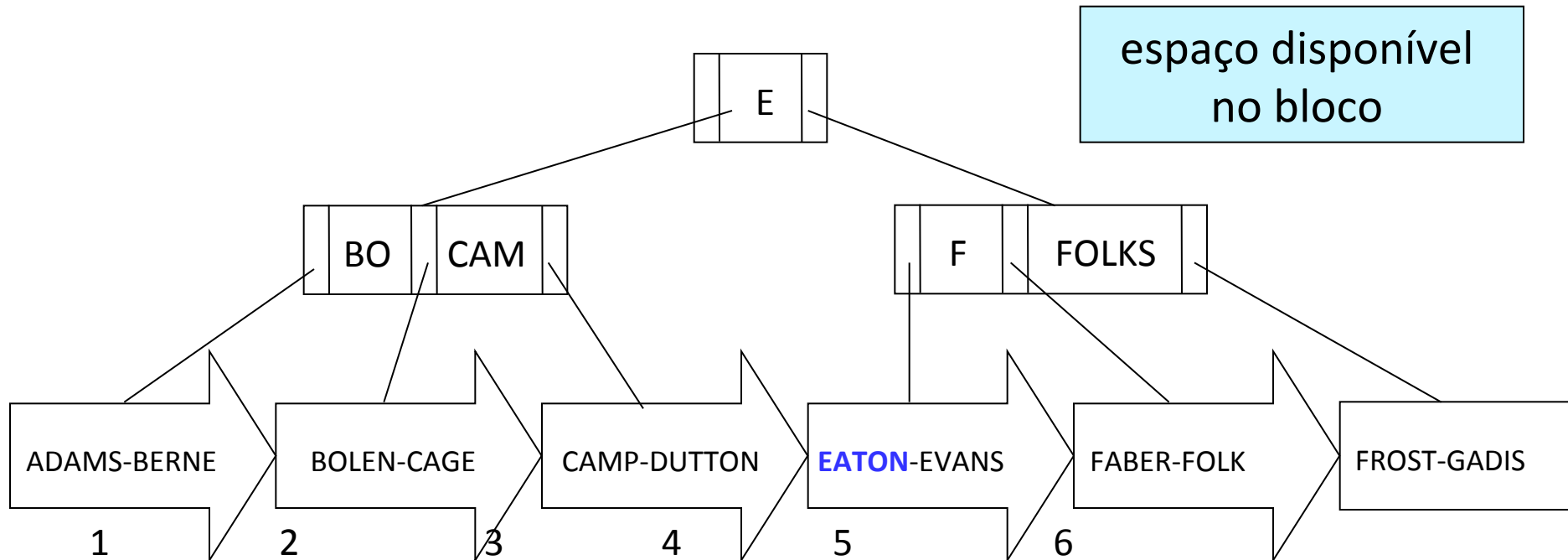


Inserção de EATON



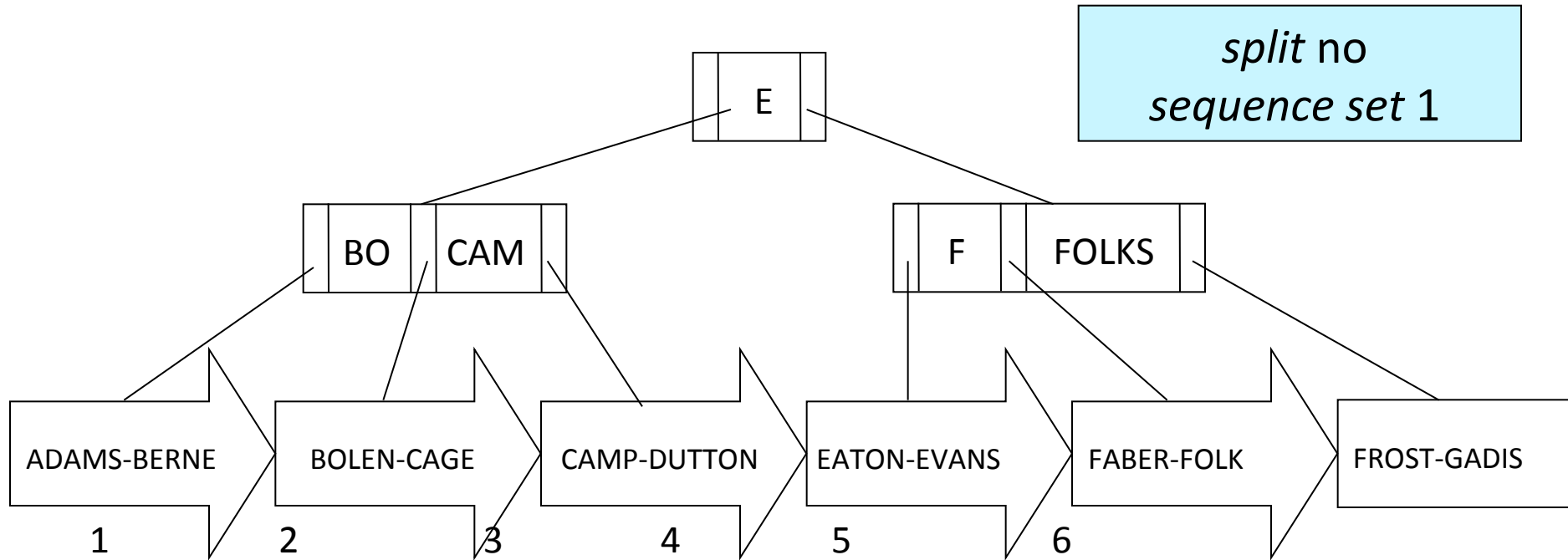
- Efeito no *sequence set*
 - limitado a alterações no bloco 4

Inserção de EATON

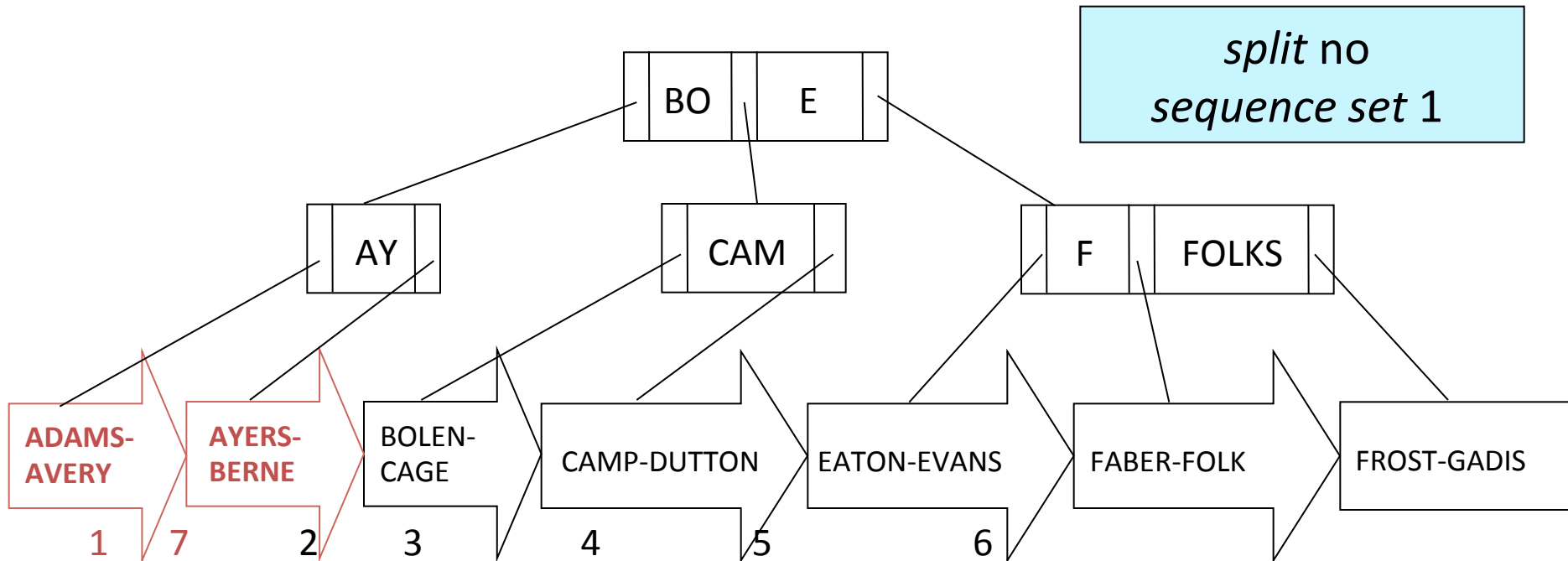


- Efeito na árvore-B+
 - nenhum: E é uma boa chave separadora

Inserção de AVERY



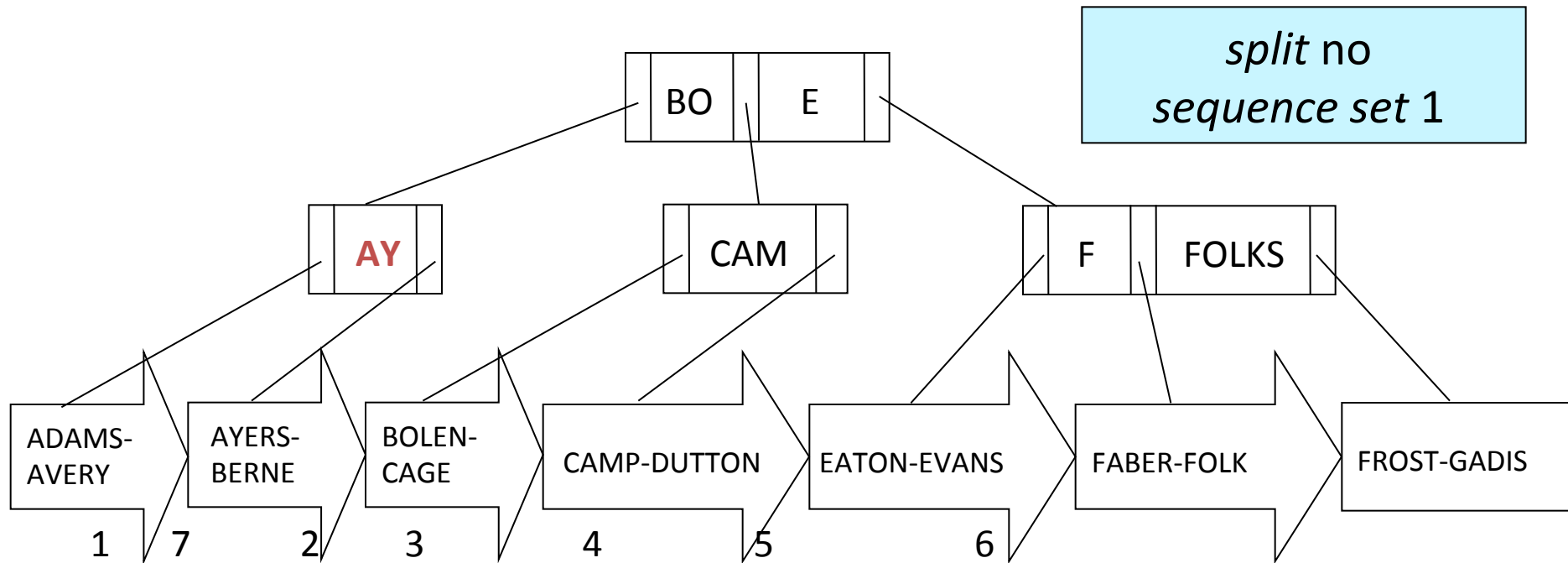
Inserção de AVERY



- Efeito no *sequence set*

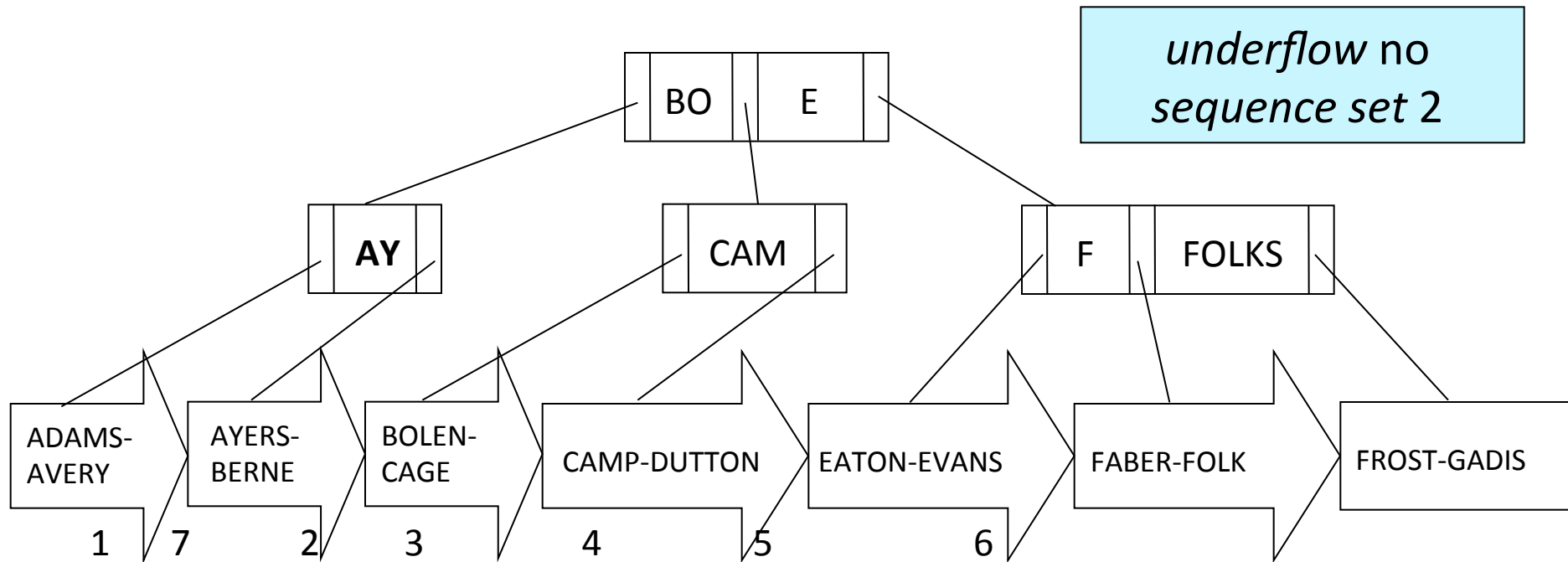
- dados do bloco 1 + AVERY distribuídos entre os blocos 1 e 7

Inserção de AVERY

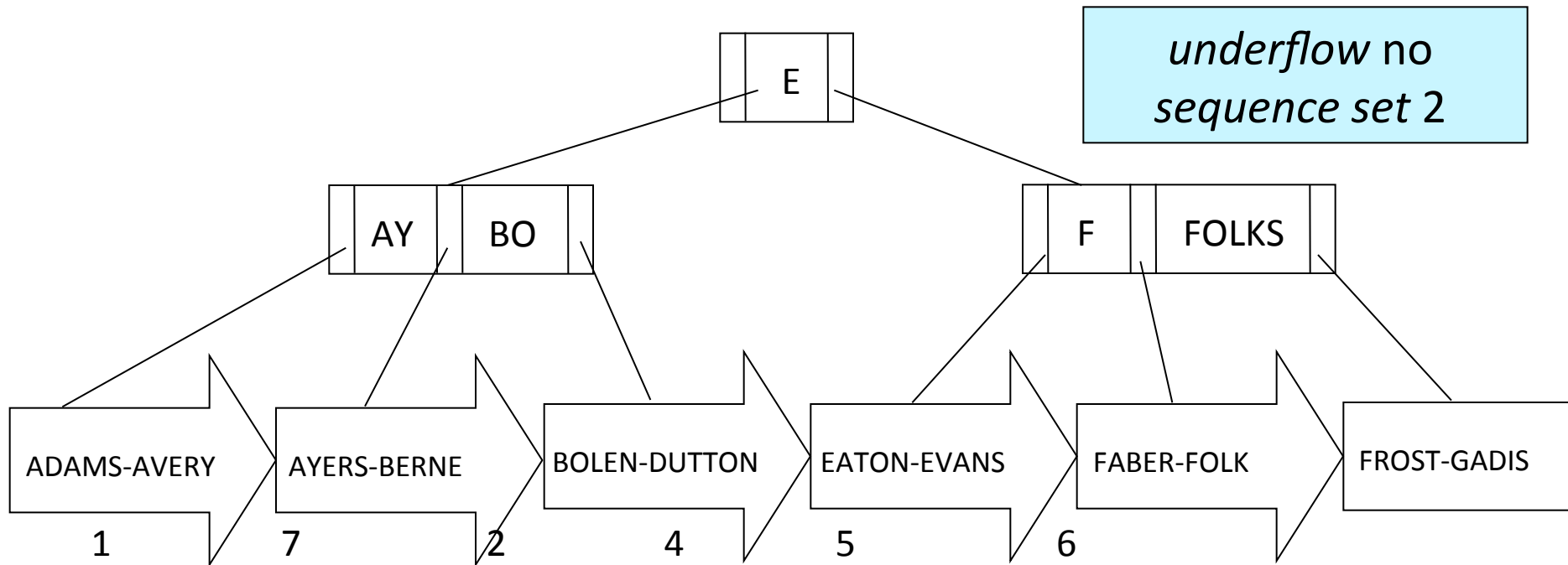


- Efeito na árvore-B+
 - separador adicional AY

Remoção de CAEL

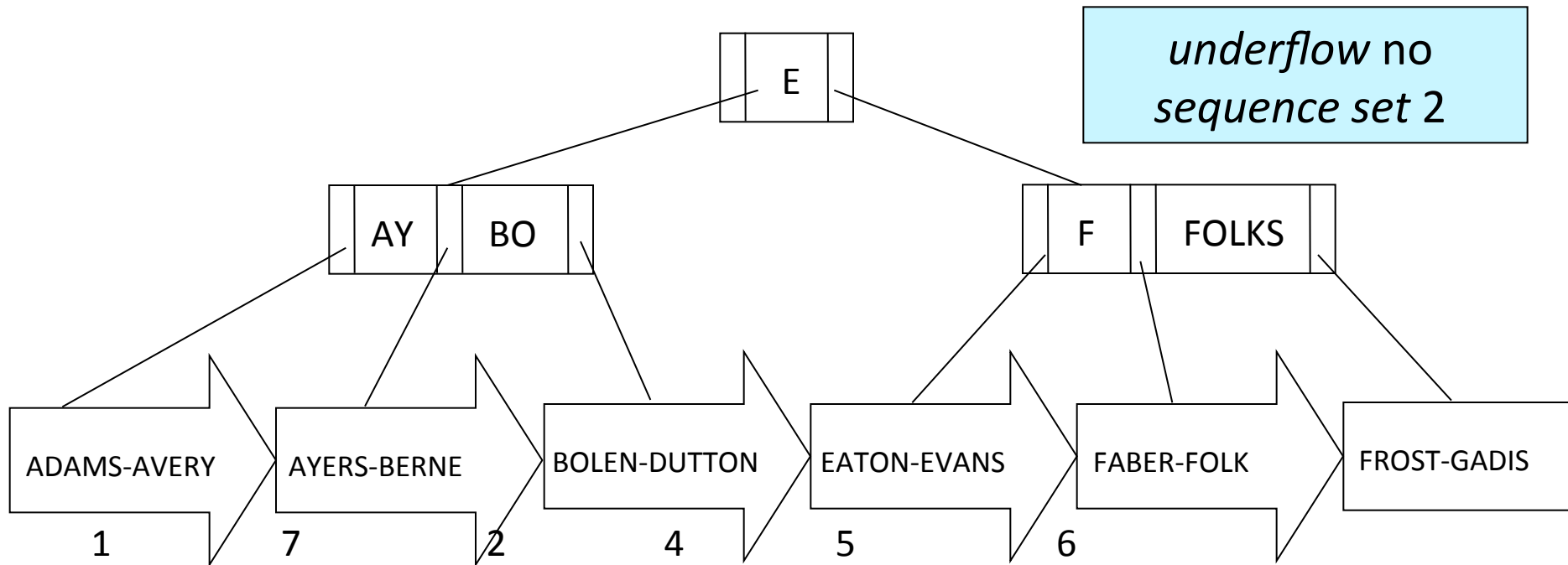


Remoção de CAEL



- Efeito no *sequence set*
 - concatenação dos blocos 2 e 3

Remoção de CAEL



- Efeito na árvore-B+
 - remoção de CAMP e concatenação de nós

Inserção e Remoção

- Primeiro passo: *Sequence Set*
 - inserir ou remover o dado
 - tratar, caso necessário
 - *split*
 - concatenação
 - redistribuição

alterações são
sempre realizadas a
partir do arquivo de
dados

Inserção e Remoção

- Segundo passo: *Árvore-B⁺*
 - se *split* no *sequence set*
 - inserir um novo separador no índice
 - se *concatenação* no *sequence set*
 - remover um separador do índice
 - se *distribuição* no *sequence set*
 - alterar o valor do separador no índice

Exercícios

1) Considere-se uma árvore B + em que $n = 4$, (o número máximo de chaves num nó).

Suponhamos que a árvore B + inicialmente consiste de um único nó, que é ao mesmo tempo a raiz e a única folha, que tem o número 1. Qual é o número mínimo de chaves que podem aparecer em um nó de folha não raiz?

Exercícios

2) Construa uma Árvore-B com $t=2$, para as letras inseridas nesta ordem: F, S, Q, K, C, L, H, T, V, W, R, N, P, A, B, X, Y, D, Z, E

3) Considere o conjunto de chaves 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13. Faça a inserção numa Árvore-B+ de modo que a árvore resultante tenha três níveis.