

Árvores B (part 2)

Prof. Dr. Lucas C. Ribas

Disciplina: Estrutura de Dados II

Departamento de Ciências de Computação e Estatística



UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”

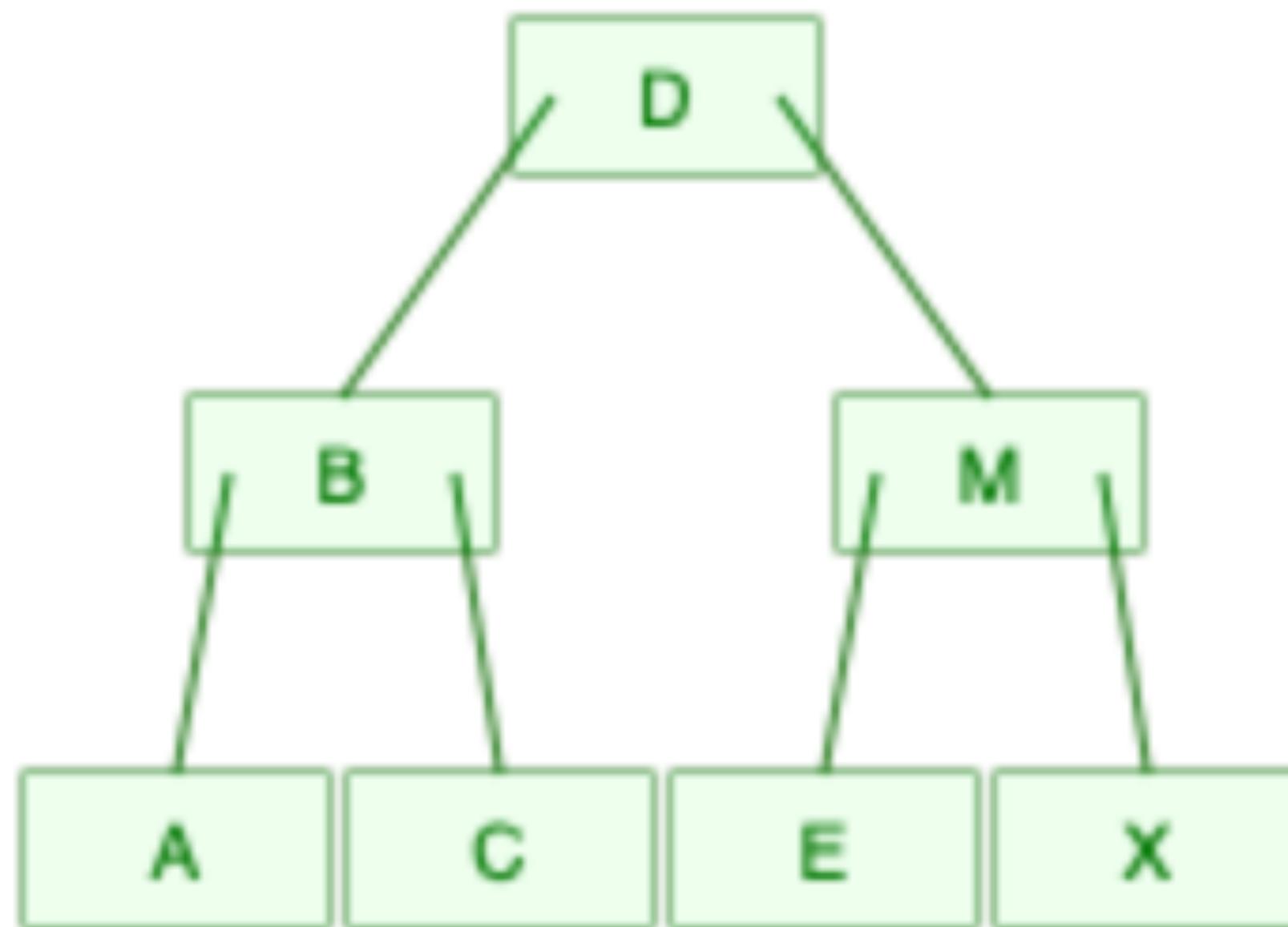


IBILCE / UNESP - CÂMPUS DE SÃO JOSÉ DO RIO PRETO



- Inserindo a sequência **M, A, B, C, X, D, E** em uma árvore-B de ordem 3 (portanto, 2 chaves por nó/página e 3 ponteiros filhos)

- Inserindo a sequência **M, A, B, C, X, D, E** em uma árvore-B de ordem 3 (portanto, 2 chaves por nó/página e 3 ponteiros filhos)





○ Estrutura de dados

- determina cada página de disco
- pode ser implementada de diferentes formas

○ Implementação adotada

- contador de ocupação (número de chaves por página)
- chaves -> caracteres (por simplicidade)
- ponteiros -> campos de referência para as páginas filhas

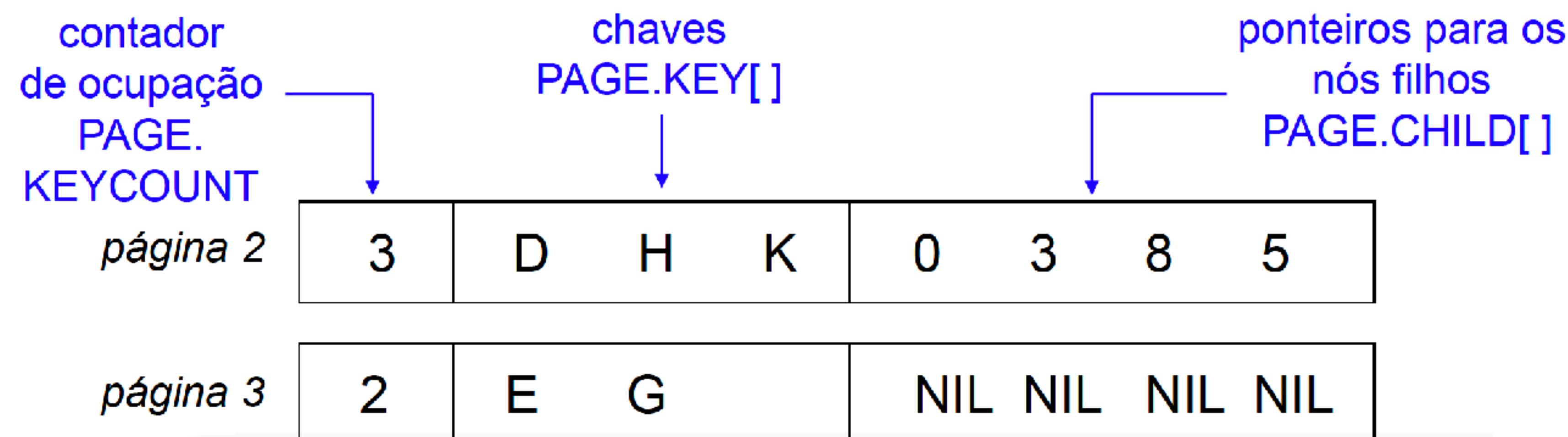
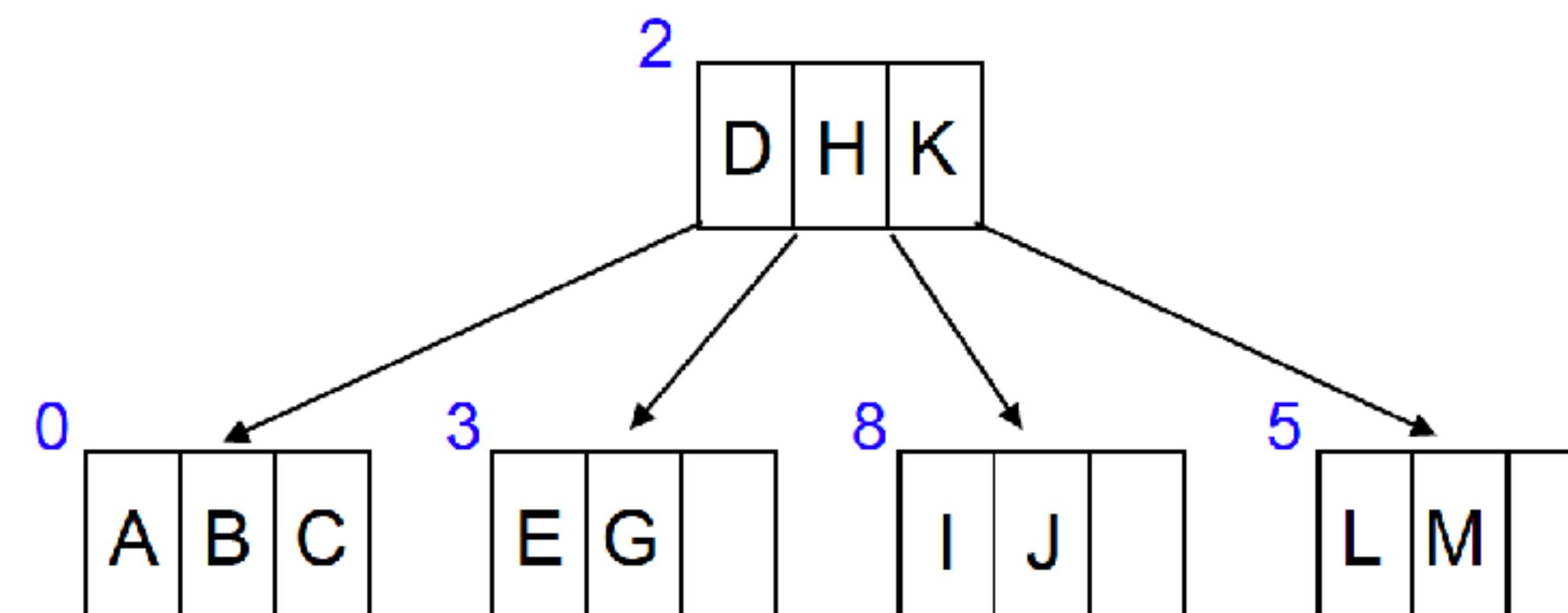
Declaração da Página



```
struct BTPage{  
  
    int keycount; /* número de chaves guardadas na página */  
  
    char key[MAX_KEYS]; /* as chaves atuais na página */  
  
    int child[MAX_KEYS+1]; /* RRNs dos filhos */  
  
} PAGE;
```

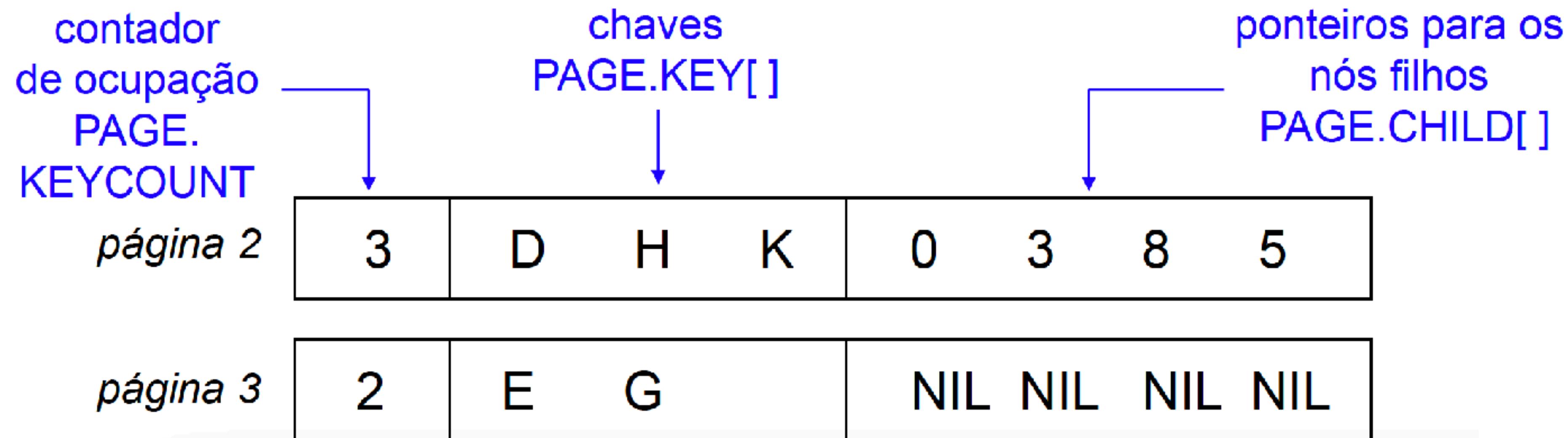
- **MAX_KEYS**: número máximo de chaves por página de disco
- **PAGE.keycount**: determina se a página está cheia ou não
- **PAGE.child[]**: contém os RRN dos nós-filhos ou -1 (ou NIL) se não houver descendentes

Arquivo da Árvore-B





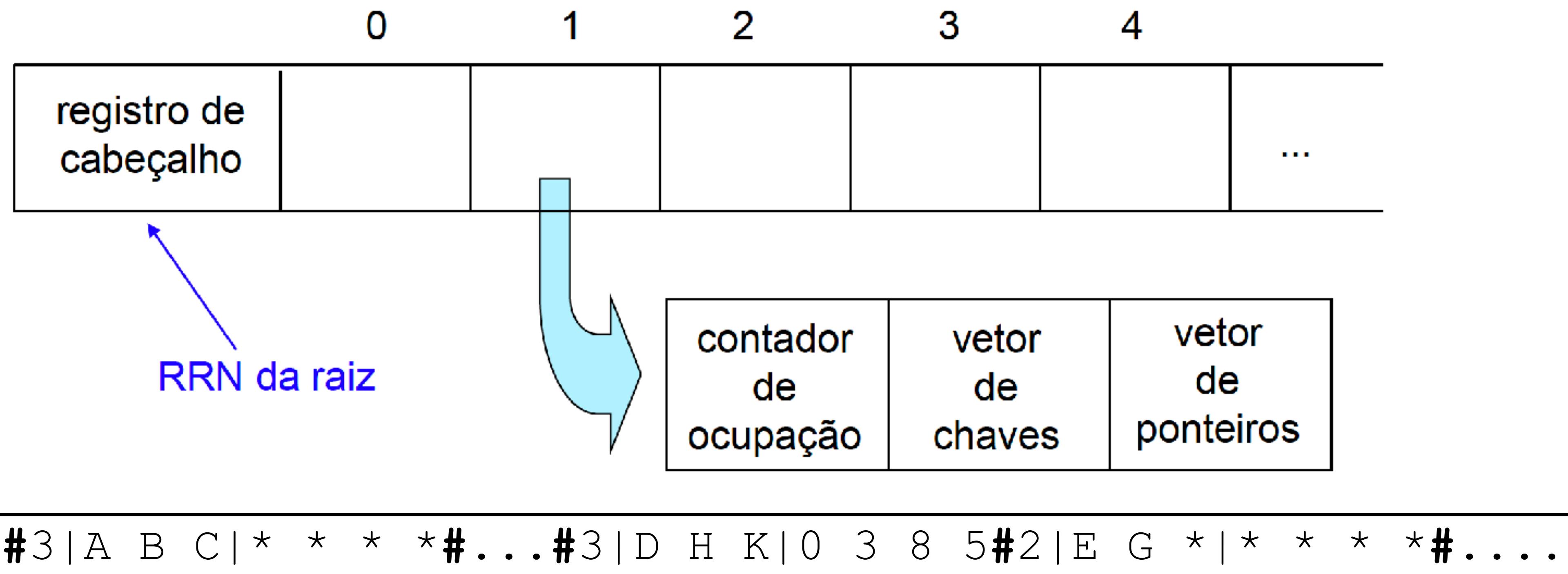
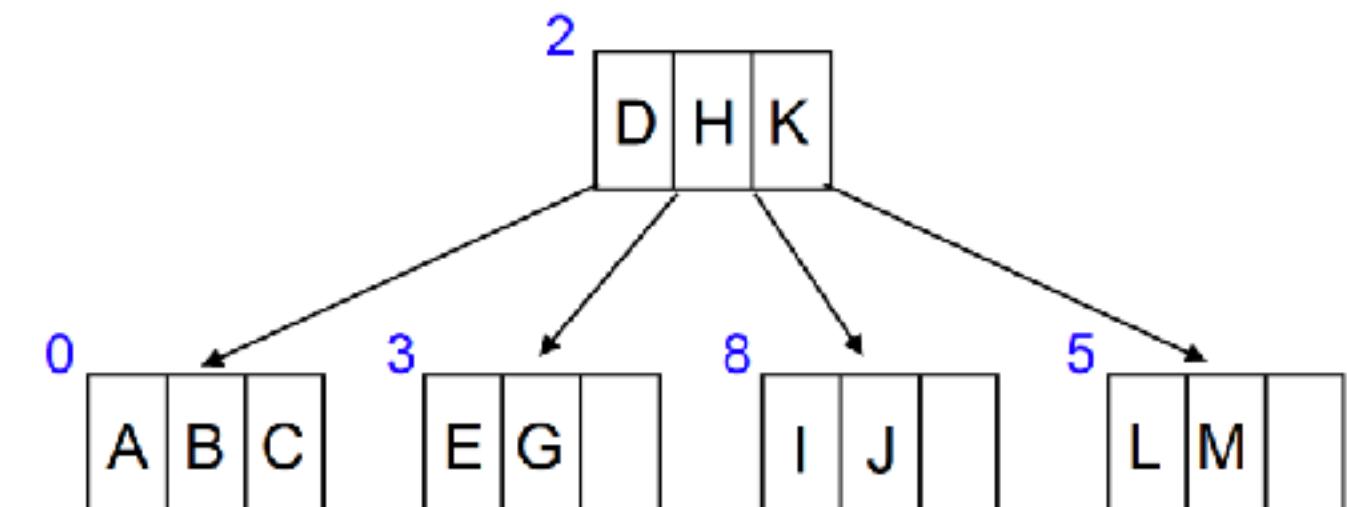
PAGE.KEY[i] : PAGE.CHILD[i] – ponteiro à esquerda
PAGE.CHILD[i+1] – ponteiro à direita



Arquivo da Árvore-B



- Conjunto de registros de tamanho fixo



- Cada registro contém uma página



● Operações básicas

- Pesquisa, inserção e remoção

● Características gerais dos algoritmos

- algoritmos **recursivos**
- **dois estágios** de processamento
 - em páginas inteiras e...
 - dentro das páginas



- Exercício: escreva o algoritmo de busca em árvore-B



- **Dados:** Raiz da árvore + chave procurada
- **Saída:** Achou + RRN da pag. + posição da chave na pag. **ou** Não-Achou

Se árvore vazia **retorne** Não-Achou

Senão

leia página raiz do arquivo;

procure chave na página;

se encontrou **retorne** RRN e posição da chave na pag.

senão pesquise recursivamente na subárvore apropriada da raiz

Refinando o Algoritmo



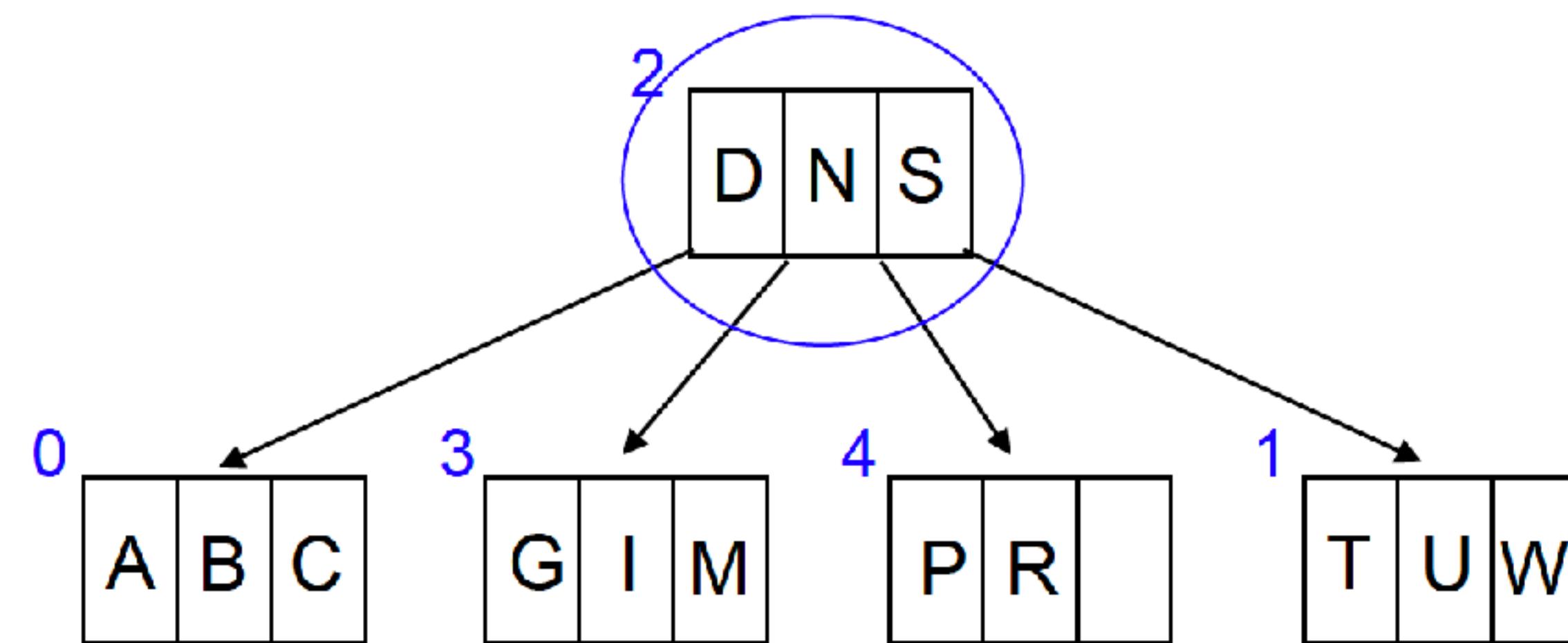
```
FUNCTION: search (RRN,  
                 KEY,  
                 FOUND_RRN,  
                 FOUND_POS)           //página a ser pesquisada  
                           //chave sendo procurada  
                           //página que contém a chave  
                           //posição da chave na página  
  
if RRN == NIL then  
    return NOT FOUND          //chave de busca não encontrada  
  
else  
    leia página identificada por RRN e armazene em PAGE  
    procure KEY em PAGE, fazendo POS igual a posição em que KEY ocorre ou deveria ocorrer  
    if KEY encontrada then  
        FOUND_RRN := RRN         //RRN corrente contém a chave  
        FOUND_POS := POS          //POS contém a posição da chave em PAGE  
        return FOUND  
    else return (search(PAGE.CHILD[POS], KEY, FOUND_RRN, FOUND_POS)) //a chave de busca  
                           //não foi encontrada,  
                           //então se procura a  
                           //chave no nó filho
```



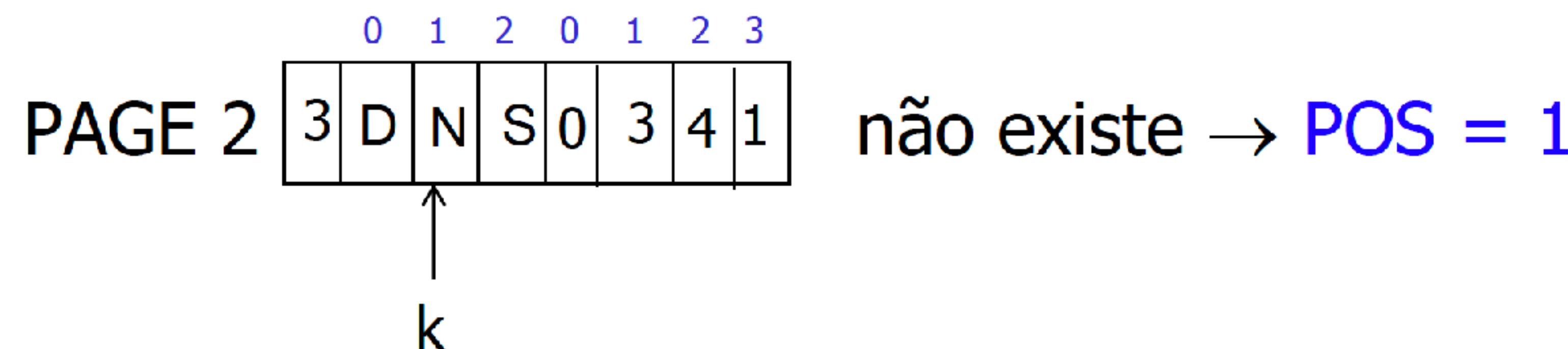
Busca da Chave K



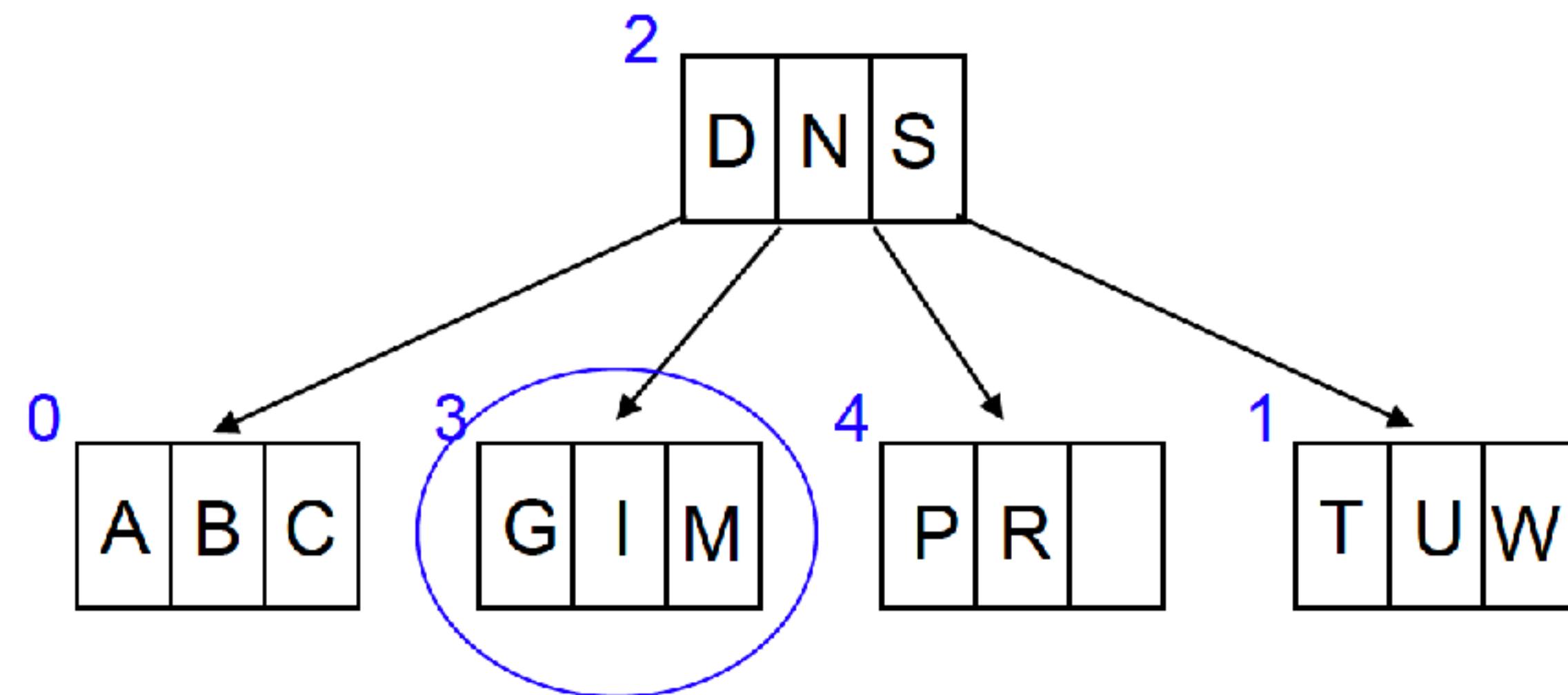
Raiz=2



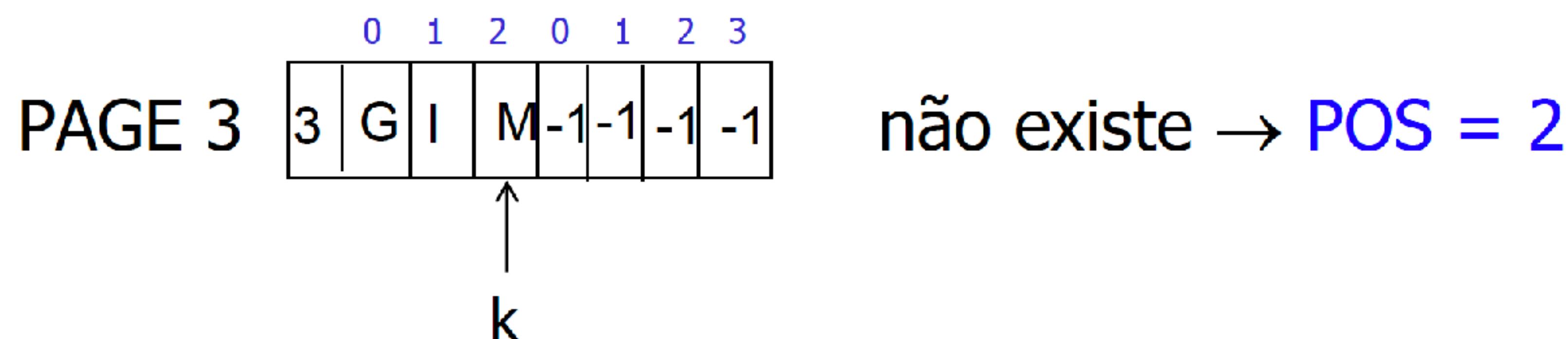
- search (2, K, FOUND_RRN, FOUND_POS)

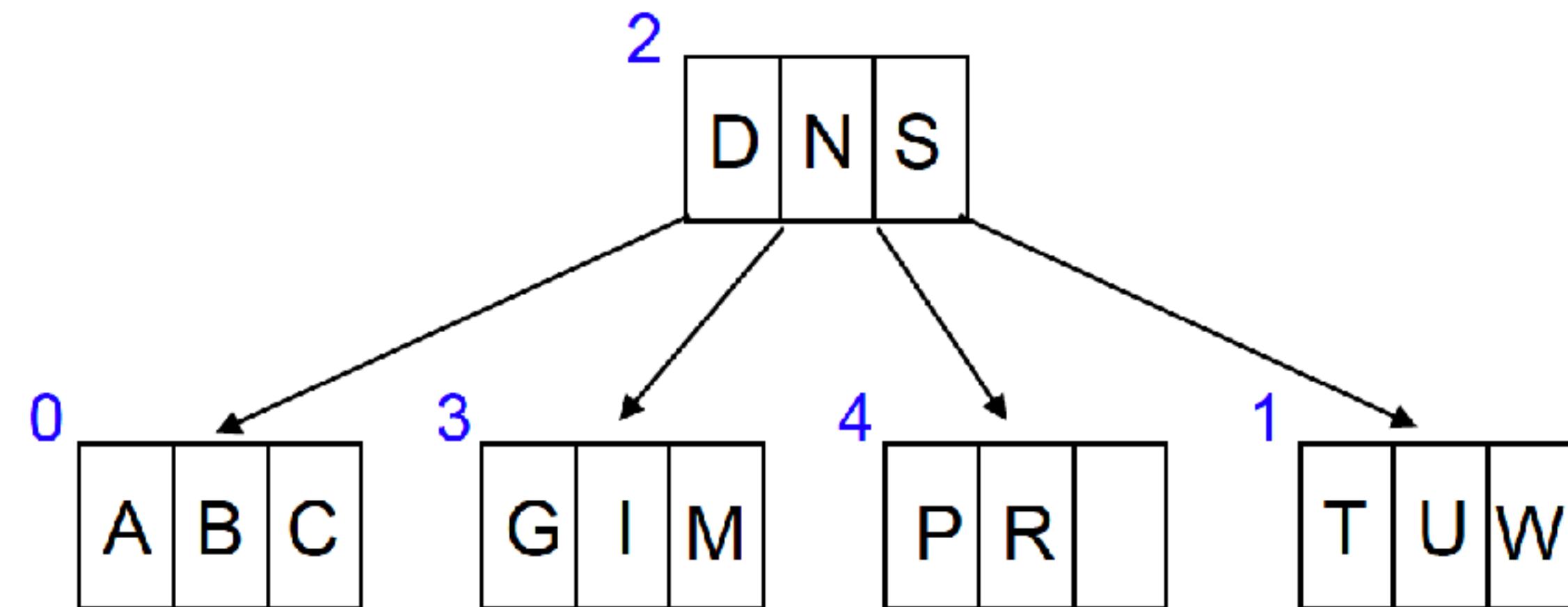


Busca da Chave K



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



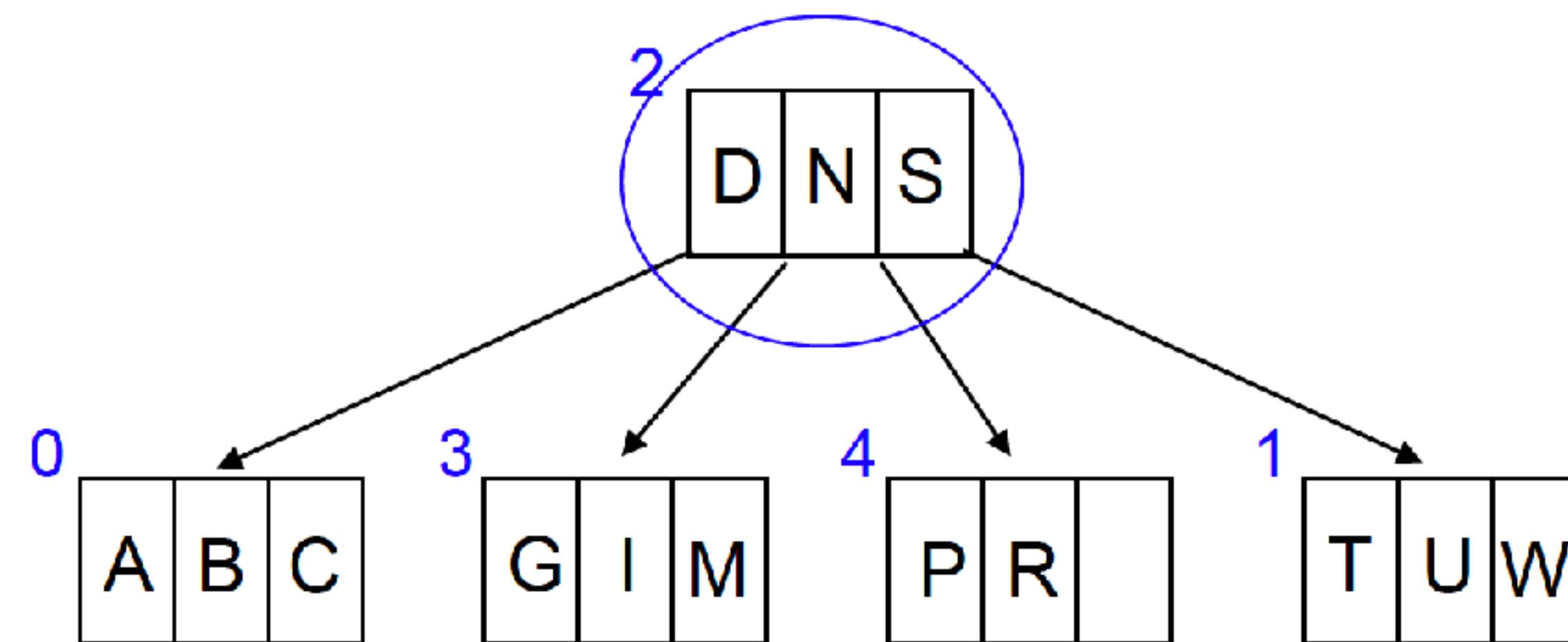


- `search (PAGE.CHILD[2], K, FOUND_RRN, FOUND_POS)`

`PAGE.CHILD[2] = NIL` → chave de busca não encontrada

return NOT FOUND

Busca da Chave M



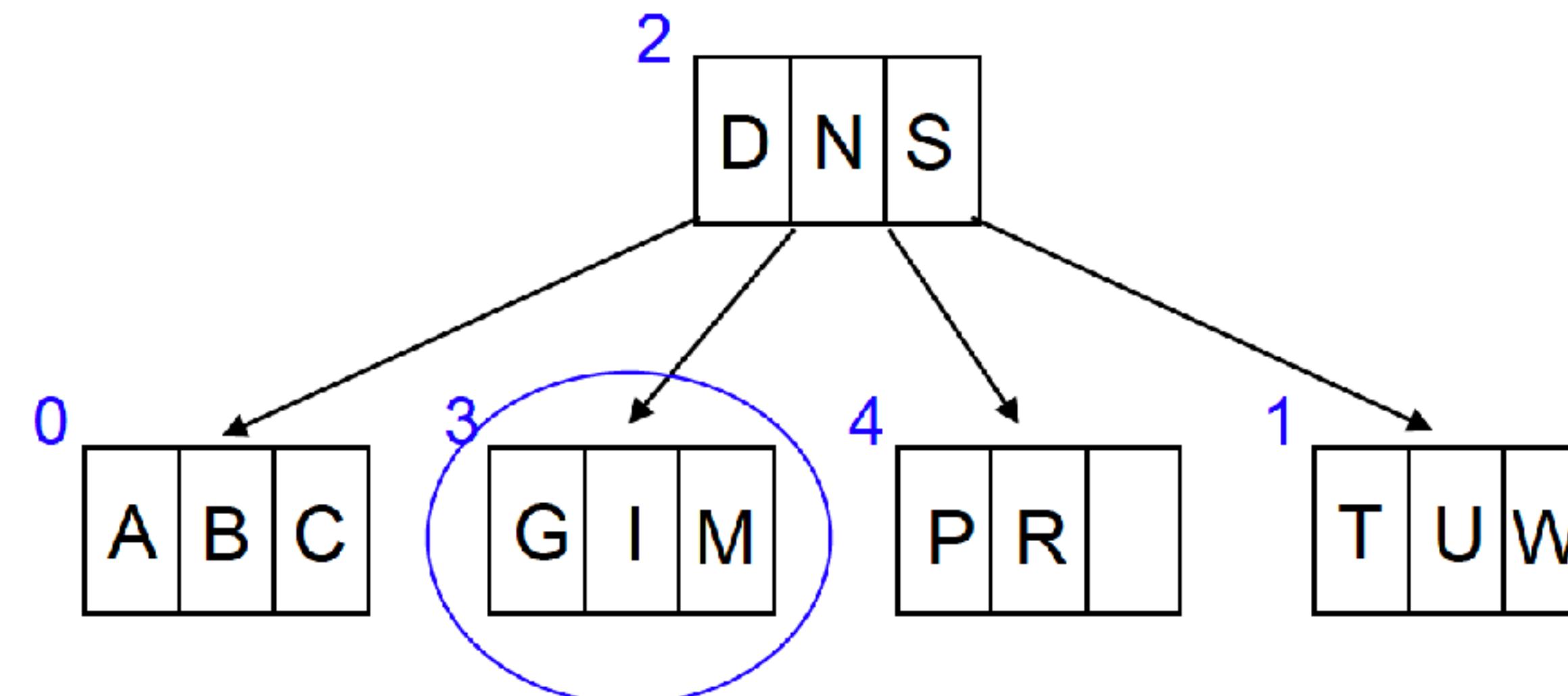
- search (2, M, FOUND_RRN, FOUND_POS)

PAGE =

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

 não existe \rightarrow POS = 1

Busca da Chave M



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

PAGE =

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

chave de busca encontrada

POS = FOUND_POS = 2

FOUND_RRN = 3

return FOUND



● Observações gerais

- inicia-se com uma **pesquisa** que desce até o nível dos nós folhas
- uma vez escolhido o nó folha no qual a nova chave deve ser inserida, os processos de **inserção**, **particionamento** (*split*) e **promoção** (*promotion*) propagam-se em direção à raiz
 - construção *bottom-up*

Algoritmo: Inserção



- **Fases** (procedimento recursivo)

1. busca pela página (*search step*)

- pesquisa da página antes da chamada recursiva

2. chamada recursiva (*recursive call*)

- move a operação para os níveis inferiores da árvore

3. inserção, split e promotion (*insertion and splitting logic*)

- executados após a chamada recursiva
- a propagação destes processos ocorre no retorno da chamada recursiva

caminho inverso
ao da pesquisa



Parâmetros

- *CURRENT_RRN*: RRN da página da árvore-B que está atualmente em uso (inicialmente, a raiz)
- *KEY*: a chave a ser inserida
- *PROMO_KEY*: retorna a chave promovida, caso a inserção resulte no particionamento e na promoção da chave
- *PROMO_R_CHILD*: retorna o ponteiro para o filho direito de *PROMO_KEY*
 - quando ocorre um particionamento, não somente a chave promovida deve ser inserida em um nó de nível mais alto da árvore, mas também deve ser inserido o RRN da nova página criada no particionamento



Valores de retorno

○ PROMOTION

- quando uma inserção é feita e uma chave é promovida -> **condição de nó cheio (i.e., overflow)**

○ NO PROMOTION

- quando uma inserção é feita e nenhuma chave é promovida -> **nó com espaço livre**

○ ERROR

- quando uma chave sendo inserida já existe na árvore-B -> **índice de chave primária**



Variáveis locais

- $PAGE$: página atualmente examinada pela função
- $NEWPAGE$: página nova resultante do particionamento
- POS : posição na página (i.e., $PAGE$) na qual a chave já ocorre ou deveria ocorrer
- P_B_KEY : chave promovida do nível inferior para ser inserida em $PAGE$
- P_B_RRN :
 - RRN promovido do nível inferior para ser inserido em $PAGE$
 - filho à direita de P_B_KEY

Função Insert



FUNCTION: `insert(CURRENT_RRN, KEY, PROMO_R_CHILD, PROMO_KEY)`

```
if CURRENT_RRN=NIL then
    PROMO_KEY:=KEY
    PROMO_R_CHILD:=NIL
    return PROMOTION
else
```

leia página CURRENT_RRN e armazene em PAGE
procure por KEY em PAGE
faça POS igual a posição em que KEY ocorre ou deveria ocorrer

Search step

```
if KEY encontrada then
    produza mensagem de erro indicado que chave já existe
    retorno ERRO
```

`RETURN_VALUE:=insert(PAGE.CHILD[POS], KEY, P_B_RRN, P_B_KEY)`

Recursive call

...

Função Insert



*insertion and
splitting logic*

...

```
if RETURN_VALUE = NO PROMOTION or ERROR then
    return RETURN_VALUE
```

```
elseif há espaço em PAGE para P_B_KEY then
    inserir P_B_KEY e P_B_RRN em PAGE
    return NO PROMOTION
```

```
else
```

```
    split(P_B_KEY, P_B_RRN, PAGE, PROMO_KEY, PROMO_R_CHILD, NEWPAGE)
```

```
    escreva PAGE no arquivo na posição CURRENT_RRN
```

```
    escreva NEWPAGE no arquivo na posição PROMO_R_CHILD
```

```
    return PROMOTION
```

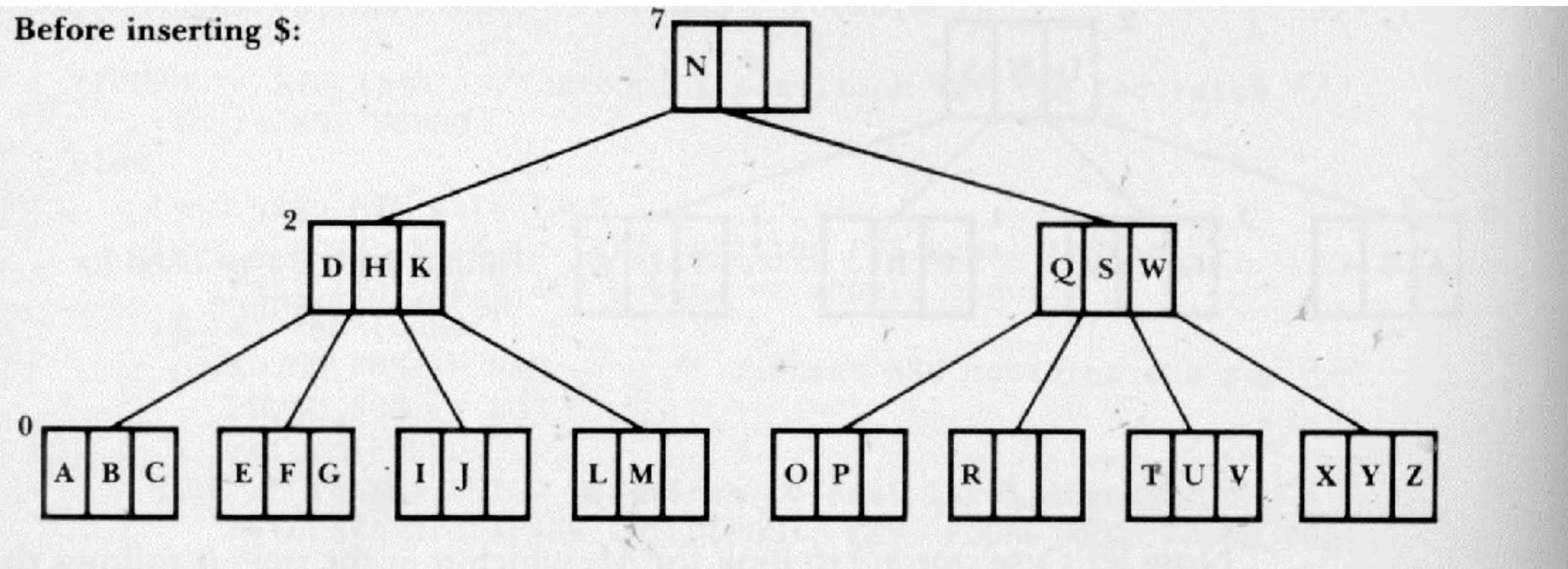
```
end
```



Exemplo: Inserção do \$ em que \$ < A



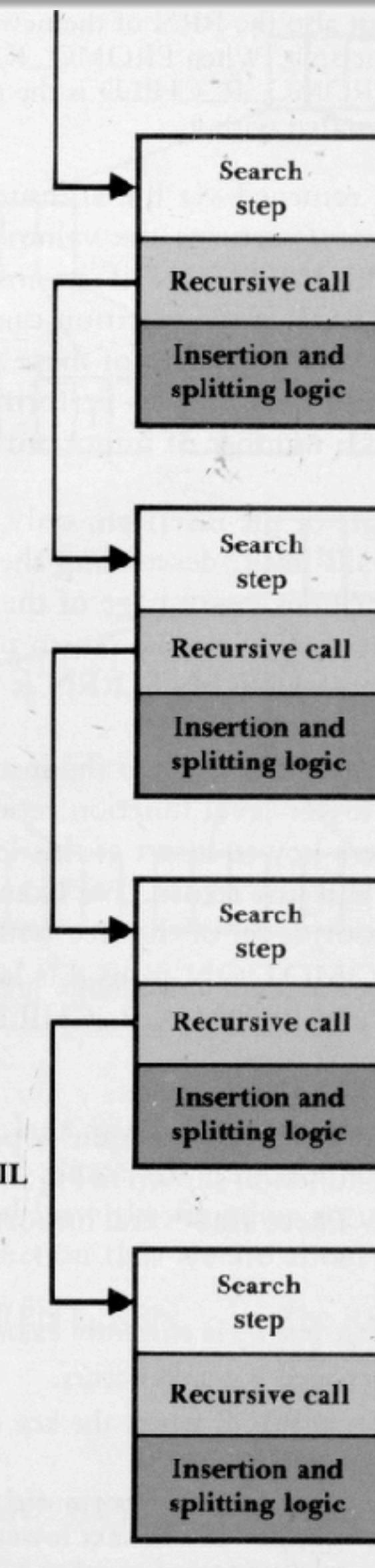
Before inserting \$:



KEY = \$
CURRENT_RRN = 2

KEY = \$
CURRENT_RRN = 0

KEY = \$
CURRENT_RRN = NIL



Exemplo: Inserção do \$ em que \$ < A

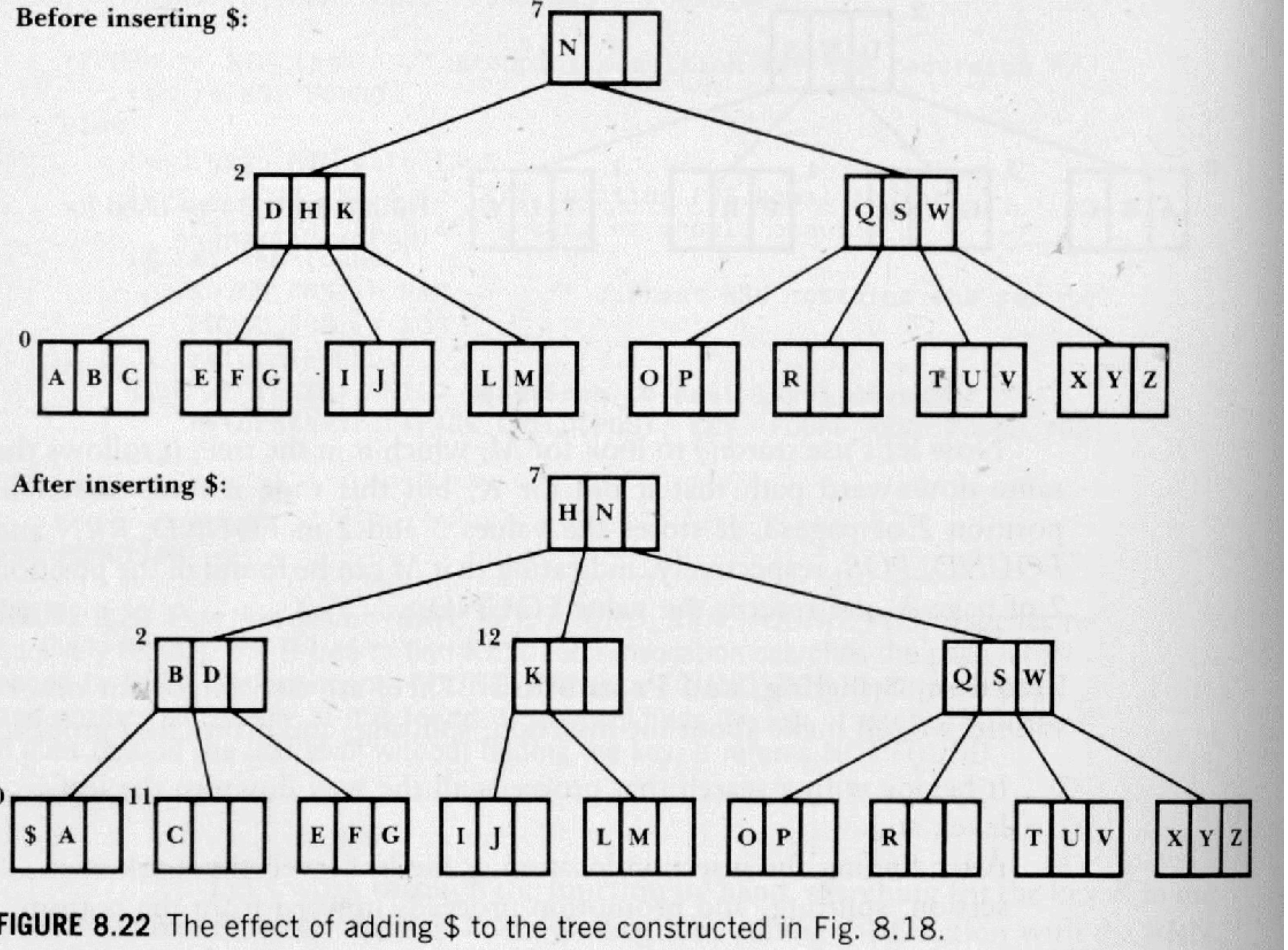


FIGURE 8.22 The effect of adding \$ to the tree constructed in Fig. 8.18.

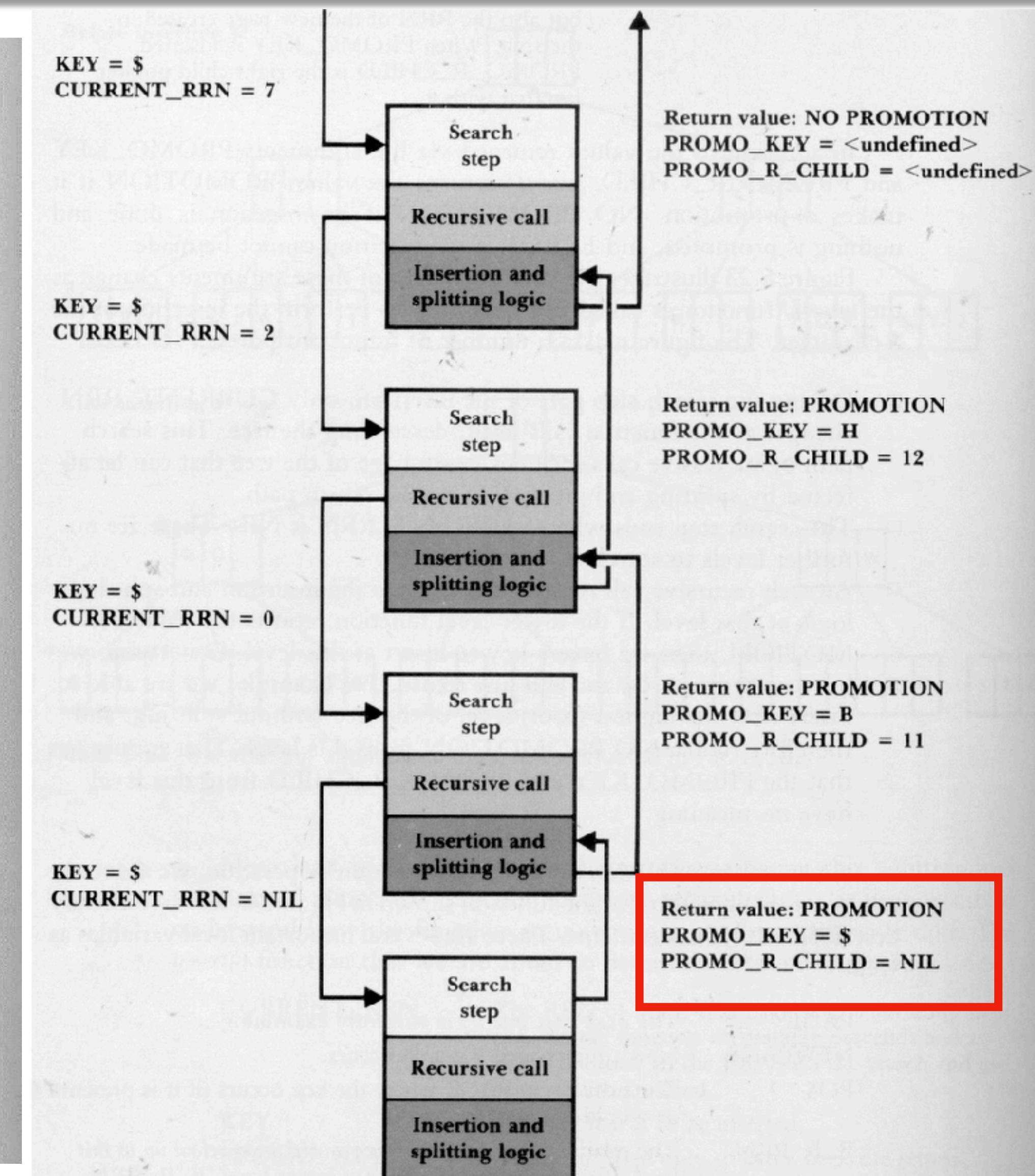
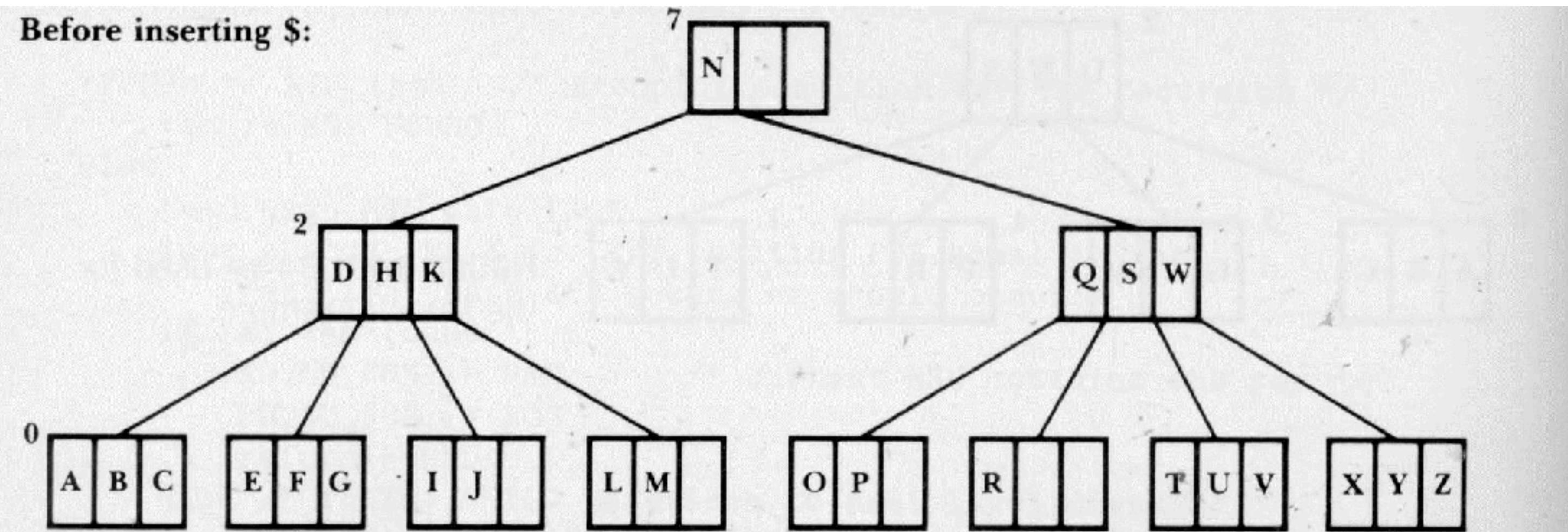


FIGURE 8.23 Pattern of recursive calls to insert \$ into the B-tree as illustrated in Fig. 8.22.

Exemplo: Inserção do \$ em que \$ < A



Before inserting \$:



After inserting \$:

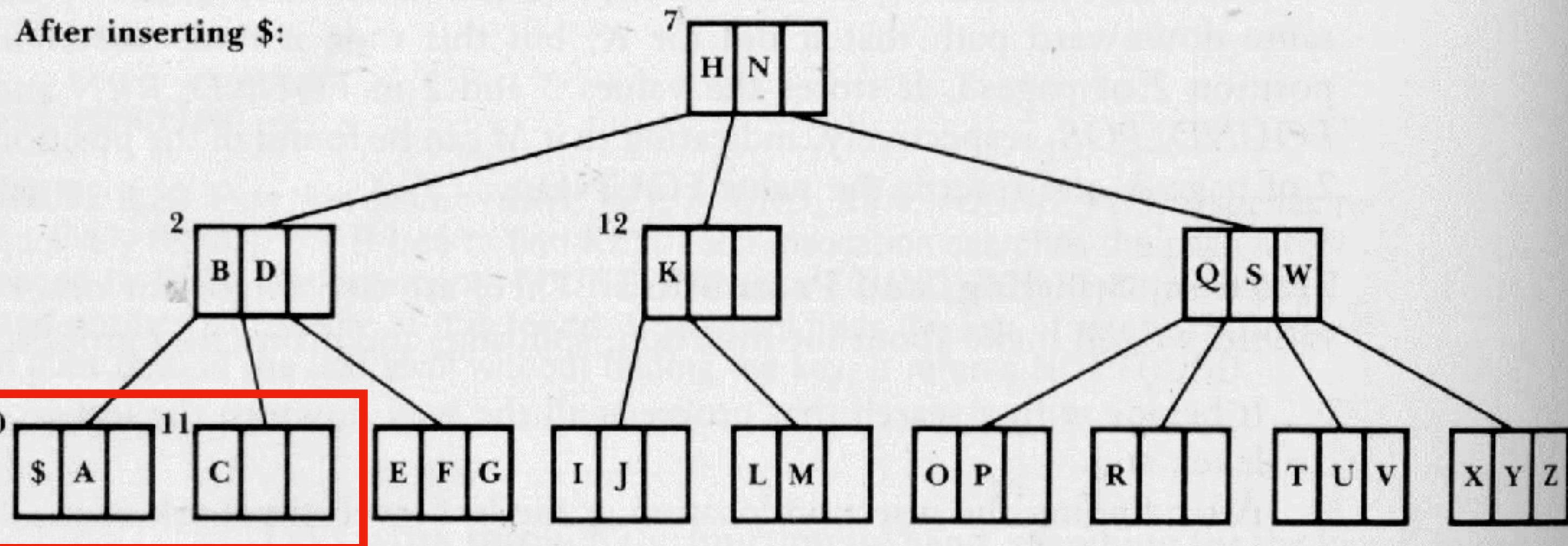


FIGURE 8.22 The effect of adding \$ to the tree constructed in Fig. 8.18.

KEY = \$
CURRENT_RRN = 7

KEY = \$
CURRENT_RRN = 2

KEY = \$
CURRENT_RRN = 0

KEY = \$
CURRENT_RRN = NIL

Return value: NO PROMOTION
PROMO_KEY = <undefined>
PROMO_R_CHILD = <undefined>

Return value: PROMOTION
PROMO_KEY = H
PROMO_R_CHILD = 12

Return value: PROMOTION
PROMO_KEY = B
PROMO_R_CHILD = 11

Return value: PROMOTION
PROMO_KEY = \$
PROMO_R_CHILD = NIL

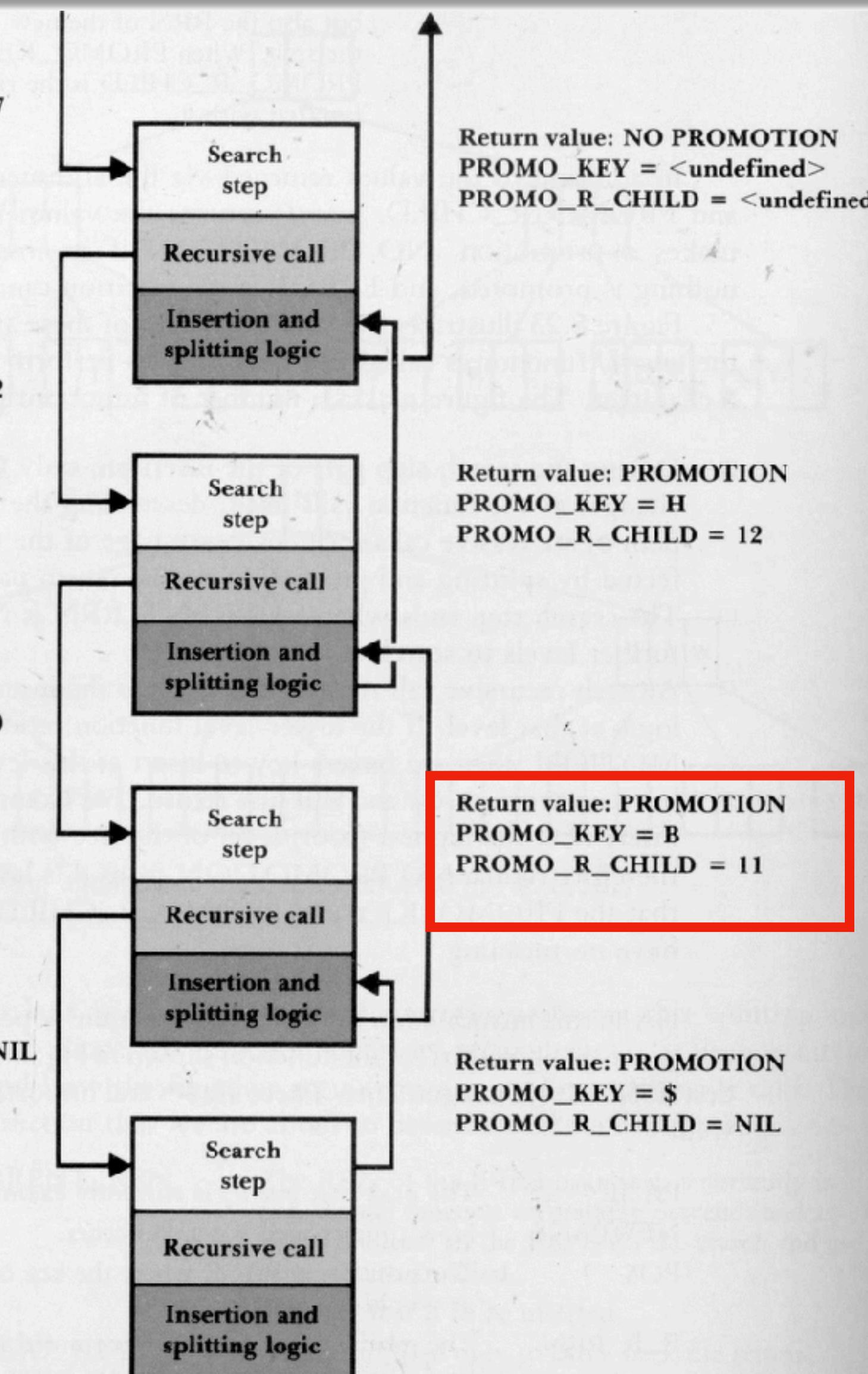
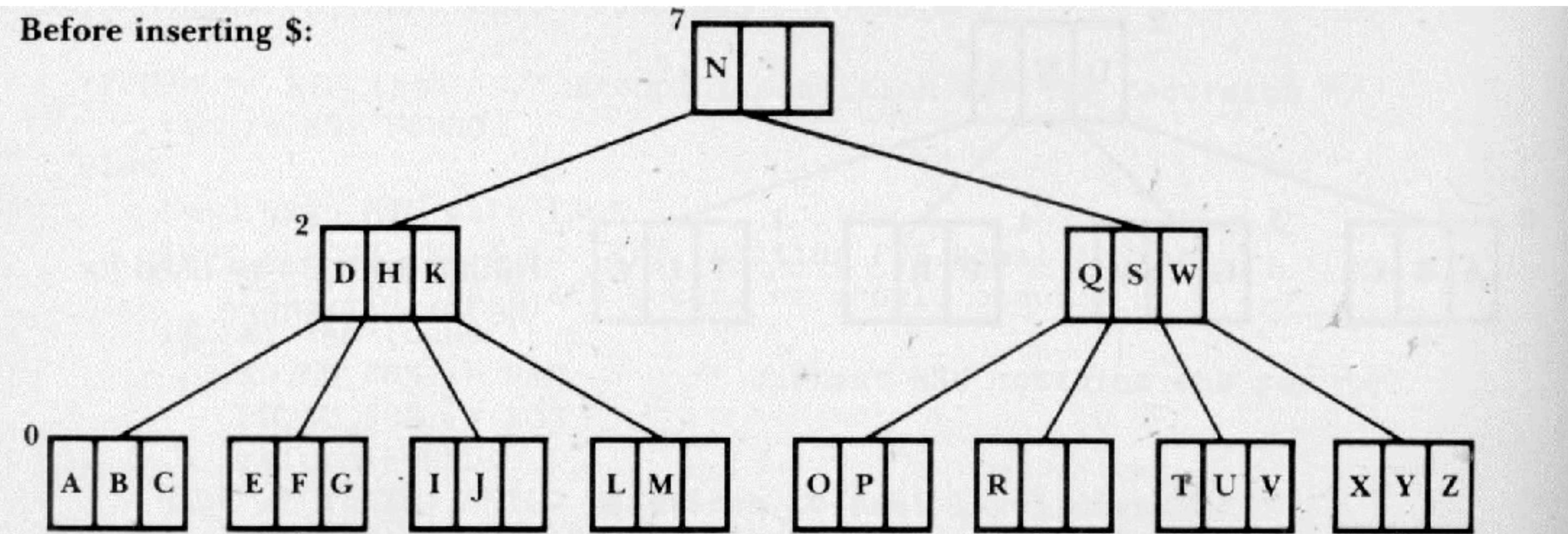


FIGURE 8.23 Pattern of recursive calls to insert \$ into the B-tree as illustrated in Fig. 8.22.

Exemplo: Inserção do \$ em que \$ < A



Before inserting \$:



After inserting \$:

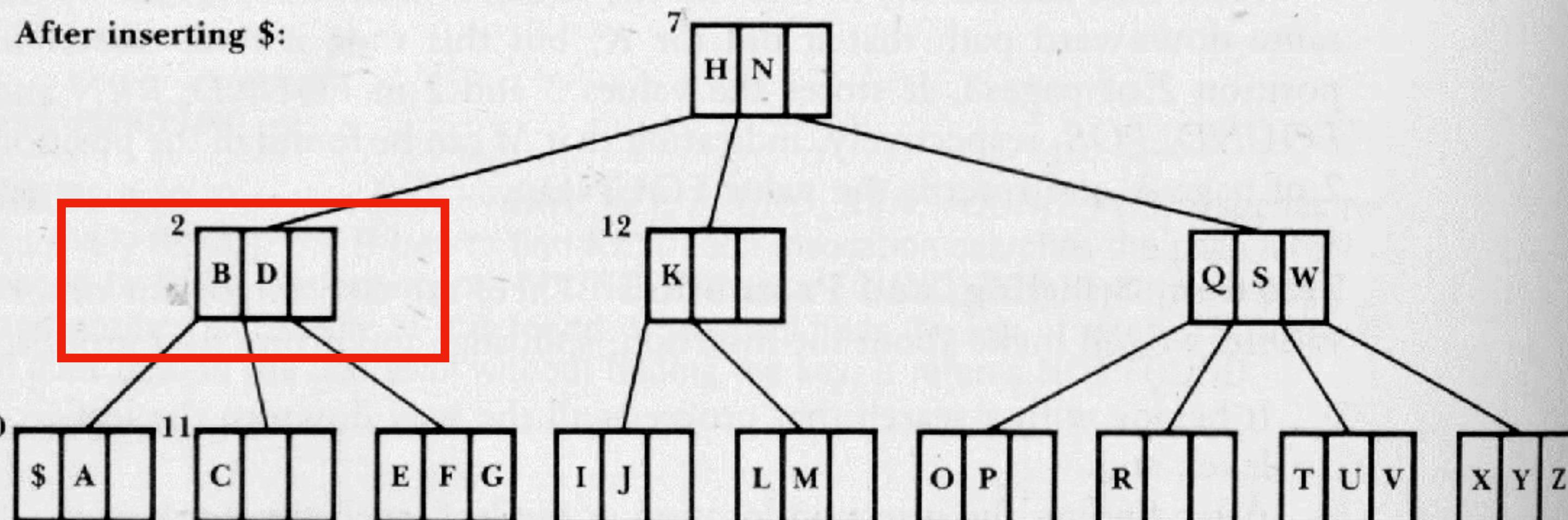


FIGURE 8.22 The effect of adding \$ to the tree constructed in Fig. 8.18.

KEY = \$
CURRENT_RRN = 7

KEY = \$
CURRENT_RRN = 2

KEY = \$
CURRENT_RRN = 0

KEY = \$
CURRENT_RRN = NIL

Return value: NO PROMOTION
PROMO_KEY = <undefined>
PROMO_R_CHILD = <undefined>

Return value: PROMOTION
PROMO_KEY = H
PROMO_R_CHILD = 12

Return value: PROMOTION
PROMO_KEY = B
PROMO_R_CHILD = 11

Return value: PROMOTION
PROMO_KEY = \$
PROMO_R_CHILD = NIL

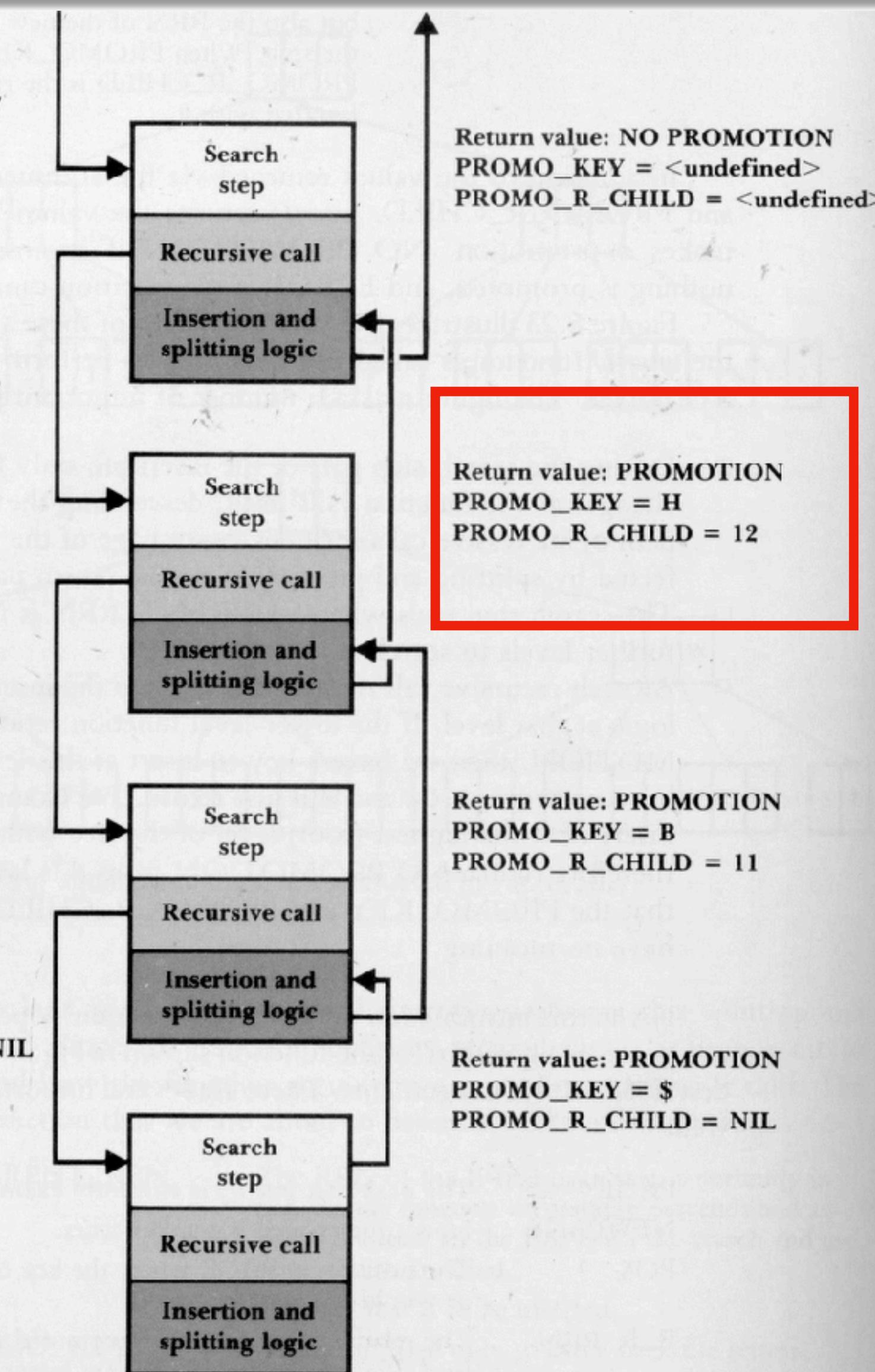


FIGURE 8.23 Pattern of recursive calls to insert \$ into the B-tree as illustrated in Fig. 8.22.

Exemplo: Inserção do \$ em que \$ < A

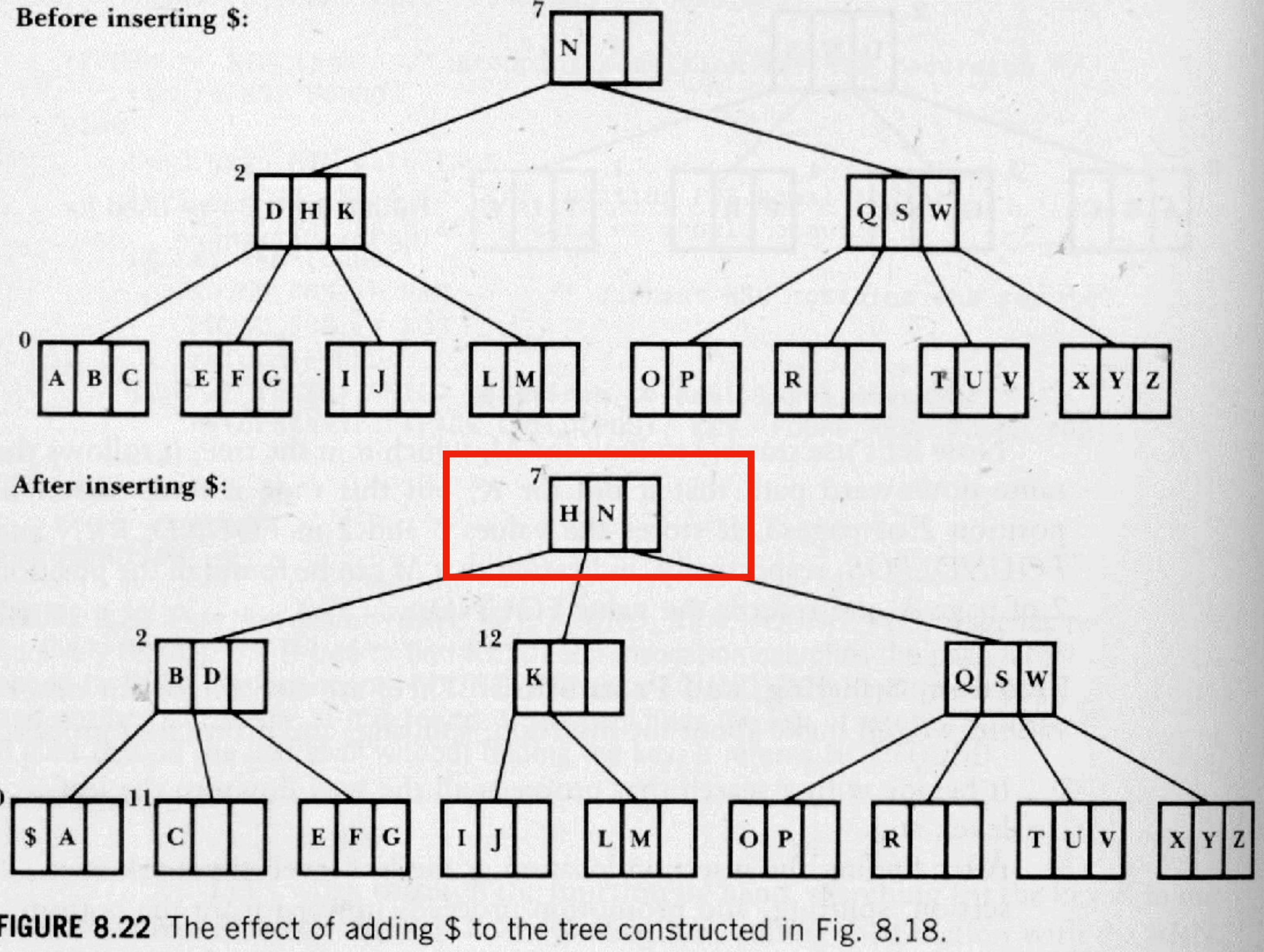


FIGURE 8.22 The effect of adding \$ to the tree constructed in Fig. 8.18.

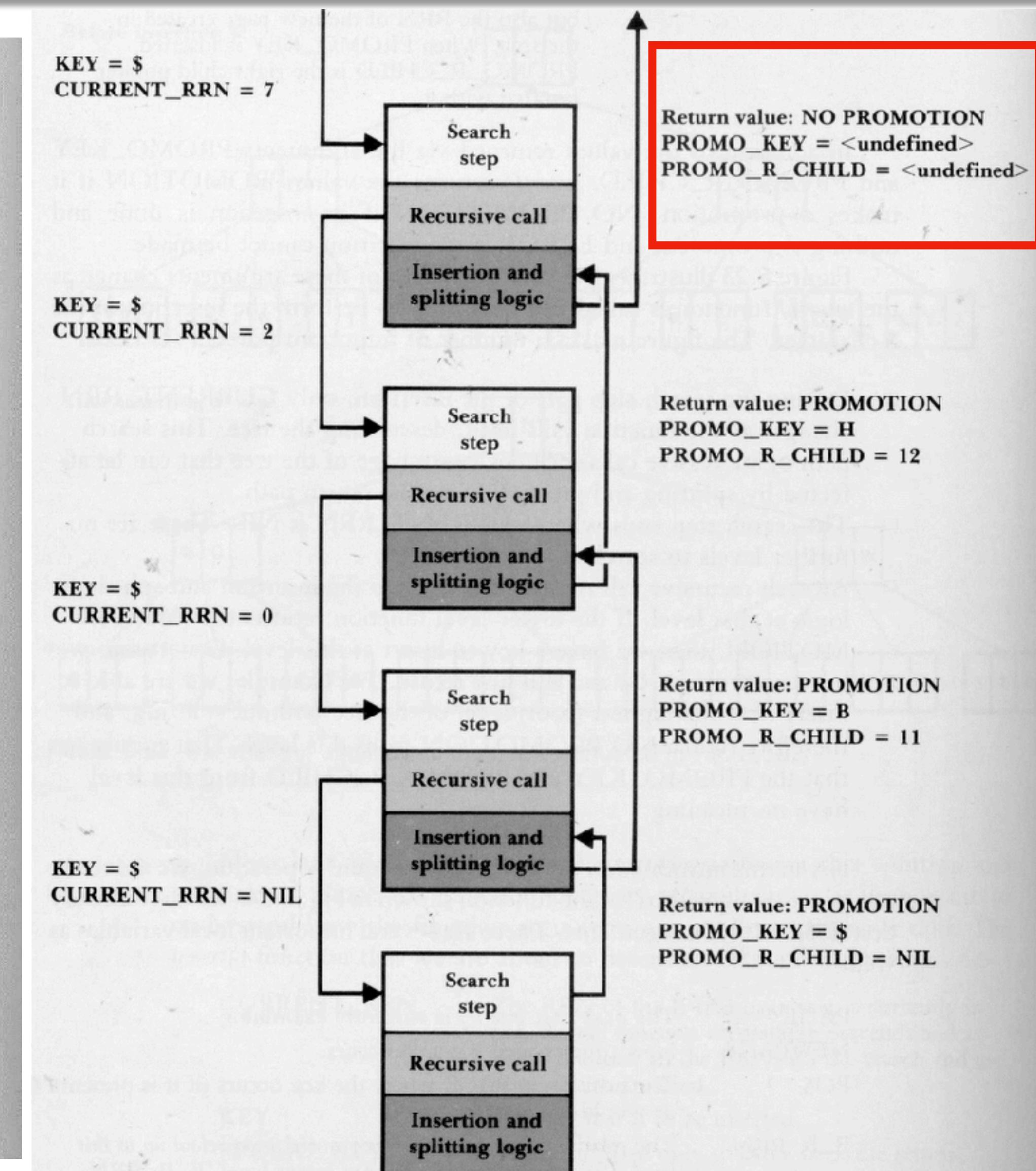


FIGURE 8.23 Pattern of recursive calls to insert \$ into the B-tree as illustrated in Fig. 8.22.

A Função Split



- **Split(P_B_KEY, P_B_RRN, PAGE, PROMO_KEY, PROMO_R_CHILD, NEWPAGE)**
- Parâmetros
- P_B_KEY, P_B_RRN
 - nova chave a ser inserida
- $PAGE$
 - página atual
- $PROMO_KEY, PROMO_R_CHILD, NEWPAGE$
 - parâmetros de retorno: chave promovida; RRN de sua subárvore a direita; Registro da página a ser gravada no arquivo



- **Tratamento do overflow causado pela inserção de uma chave**
 - cria uma nova página (i.e., NEWPAGE)
 - distribui as chaves o mais uniformemente possível entre PAGE e NEWPAGE
 - determina qual chave e qual RRN serão promovidos
 - PROMO_KEY
 - PROMO_R_CHILD

A Função Split



Function split(I_KEY, I_RRN, PAGE, PROMO_KEY, PROMO_R_CHILD, NEWPAGE)

copie todas as chaves e ponteiros de PAGE para uma página temporária estendida com espaço extra para uma nova chave e um novo ponteiro

inseria I_KEY e I_RRN no lugar apropriado na página temporária

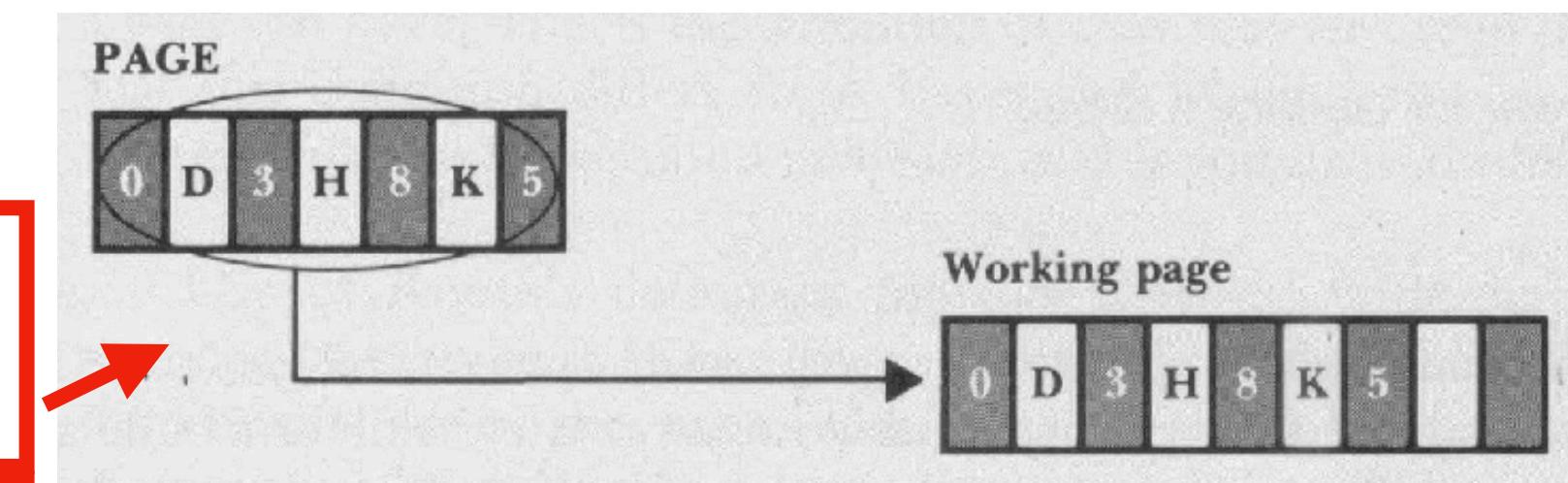
crie a nova página NEWPAGE

faça PROMO_KEY igual à chave do meio da página temporária

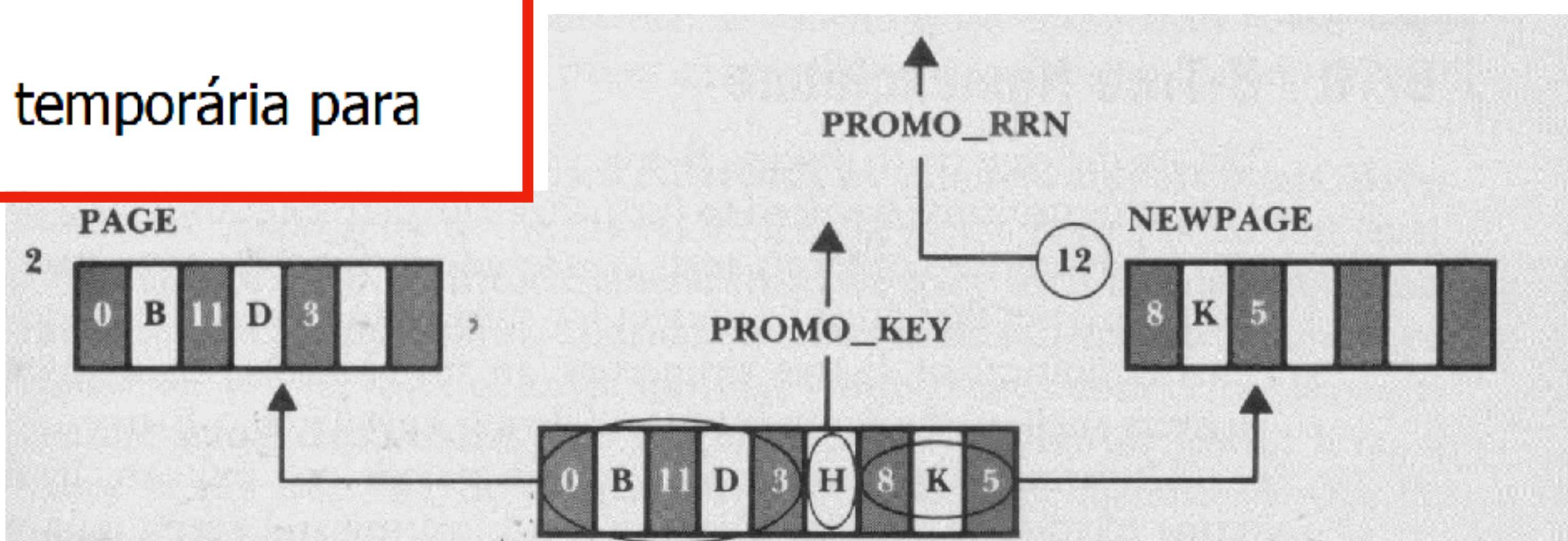
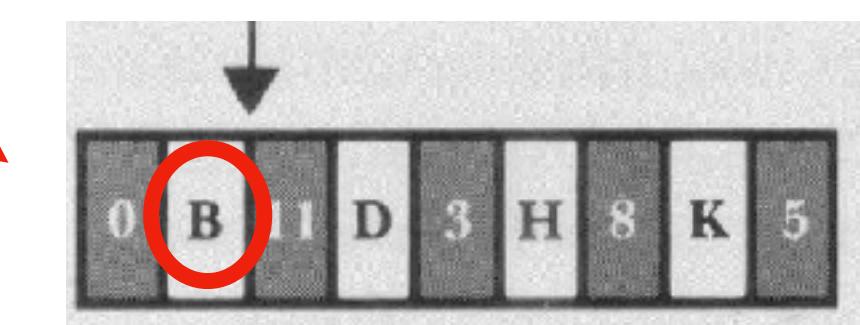
faça PROMO_R_CHILD igual ao RRN da nova página

copie as chaves e os ponteiros que precedem PROMO_KEY da página temporária para PAGE

copie as chaves e os ponteiros que seguem PROMO_KEY da página temporária para NEWPAGE



Inserir B (11)

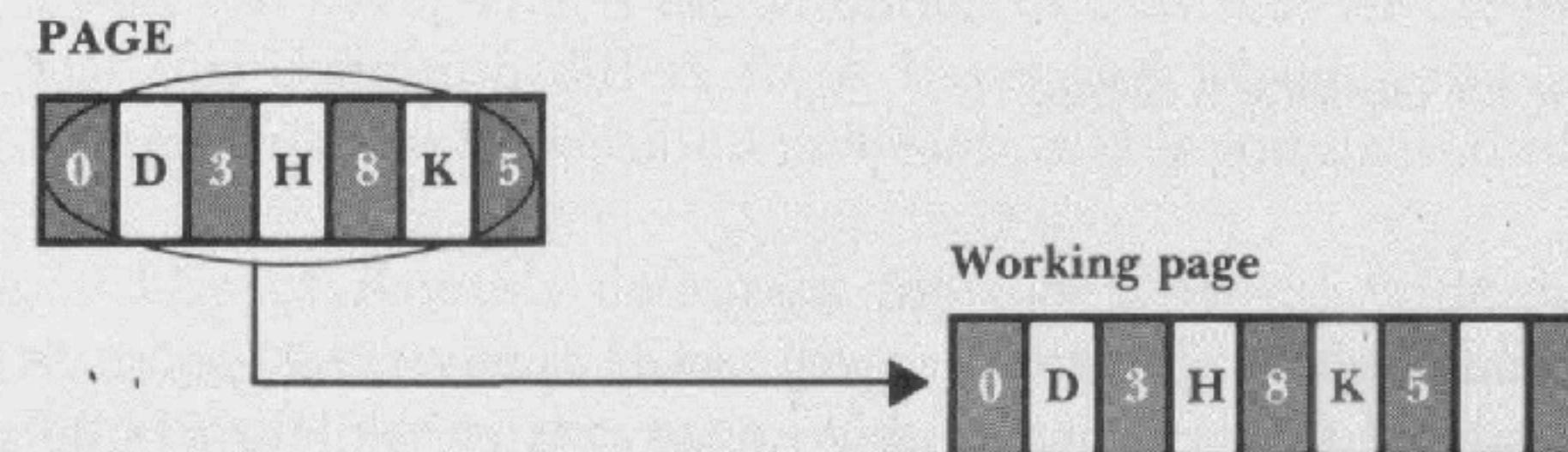


A Função Split

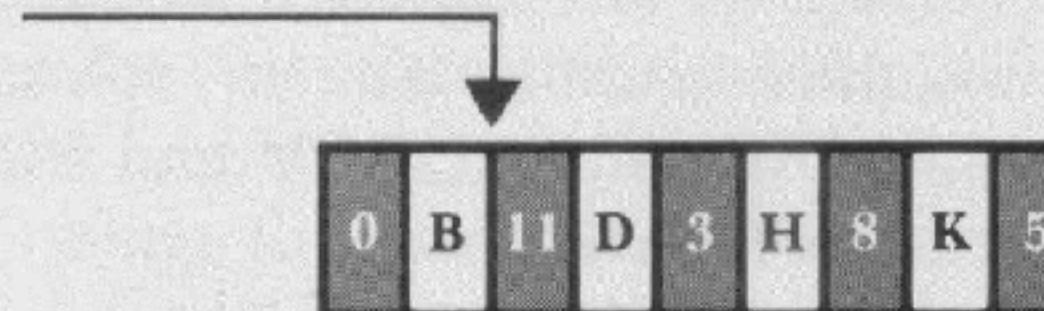


FIGURE 8.26 The movement of data in *split()*.

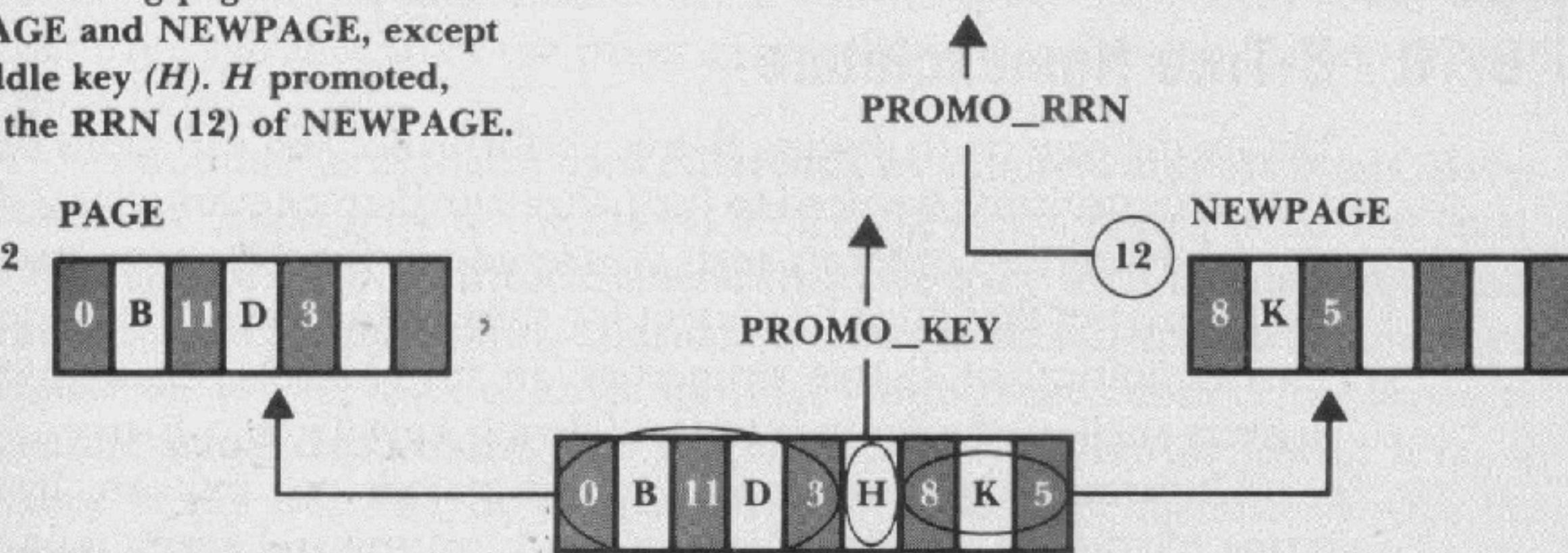
Contents of PAGE are copied to the working page.



I_KEY (B) and I_RRN (11) are inserted into working page.



Contents of working page are divided between PAGE and NEWPAGE, except for the middle key (H). H promoted, along with the RRN (12) of NEWPAGE.





○ Observações

- somente uma chave é promovida e sai da página de trabalho atual
- todos os RRN dos nós filhos são transferidos de volta para PAGE e NEWPAGE
- o RRN promovido é o de NEWPAGE
 - NEWPAGE é a descendente direita da chave promovida

Note que a função *split* move os dados!



● Rotina inicializadora e de tratamento da raiz

- abre ou cria o arquivo de índice (árvore-B)
- identifica ou cria a [página da raiz](#)
- lê [chaves](#) para serem armazenadas na árvore-B e chama insert de forma apropriada
- [cria uma nova raiz](#) quando insert particionar a raiz corrente

Algoritmo: Driver



MAIN PROCEDURE: driver

```
if arquivo com árvore-B existe then
    abra arquivo
else crie arquivo e coloque a primeira chave na raiz
recupere RRN da página raiz e armazene em ROOT
leia uma chave e armazene em KEY

while KEY existe do
    if (insert(ROOT, KEY, PROMO_R_CHILD, PROMO_KEY)=PROMOTION) then
        crie nova página raiz com key:=PROMO_KEY, l_child:=ROOT, r_child:=PROMO_R_CHILD
        faça ROOT igual ao RRN da nova página raiz
    leia próxima chave e armazene em KEY
escreva no arquivo o RRN armazenado em ROOT

feche arquivo
```

Definição e Propriedades de Árvores-B



- **Ordem** de uma árvore-B
 - Número máximo de descendentes que uma página, ou nó, pode possuir
- Em uma árvore-B de ordem **m**, o número máximo de chaves em uma página é **m-1**
 - Exemplo:
 - Uma árvore-B de ordem 8 tem, no máximo, 7 chaves por página



● Número mínimo de chaves por página

- Quando uma página é particionada na inserção, as chaves são divididas (quase) igualmente entre as páginas velha e nova
- o número mínimo de chaves em um nó é dado por $\lceil m/2 \rceil - 1$ (exceto para a raiz principal, que pode ter menos da metade e, no mínimo, 1 chave)
- **Exemplo:** árvore B de ordem 8, armazena no máximo 7 chaves por página e tem, no mínimo, 3 chaves por página

Propriedades das Árvores-B



- Para uma árvore-B de ordem m

- 1.cada página tem, no máximo, m descendentes
- 2.cada página, exceto a raiz e as folhas, tem no mínimo $\lceil m/2 \rceil$ descendentes
- 3.a raiz tem, no mínimo, dois descendentes - a menos que seja uma folha
- 4.todas as folhas estão no mesmo nível
- 5.uma página não folha que possui k descendentes contém $k-1$ chaves
- 6.uma página folha contém, no mínimo $\lceil m/2 \rceil - 1$ e, no máximo, $m-1$ chaves

Propriedades das Árvores-B



● Profundidade da busca no pior caso

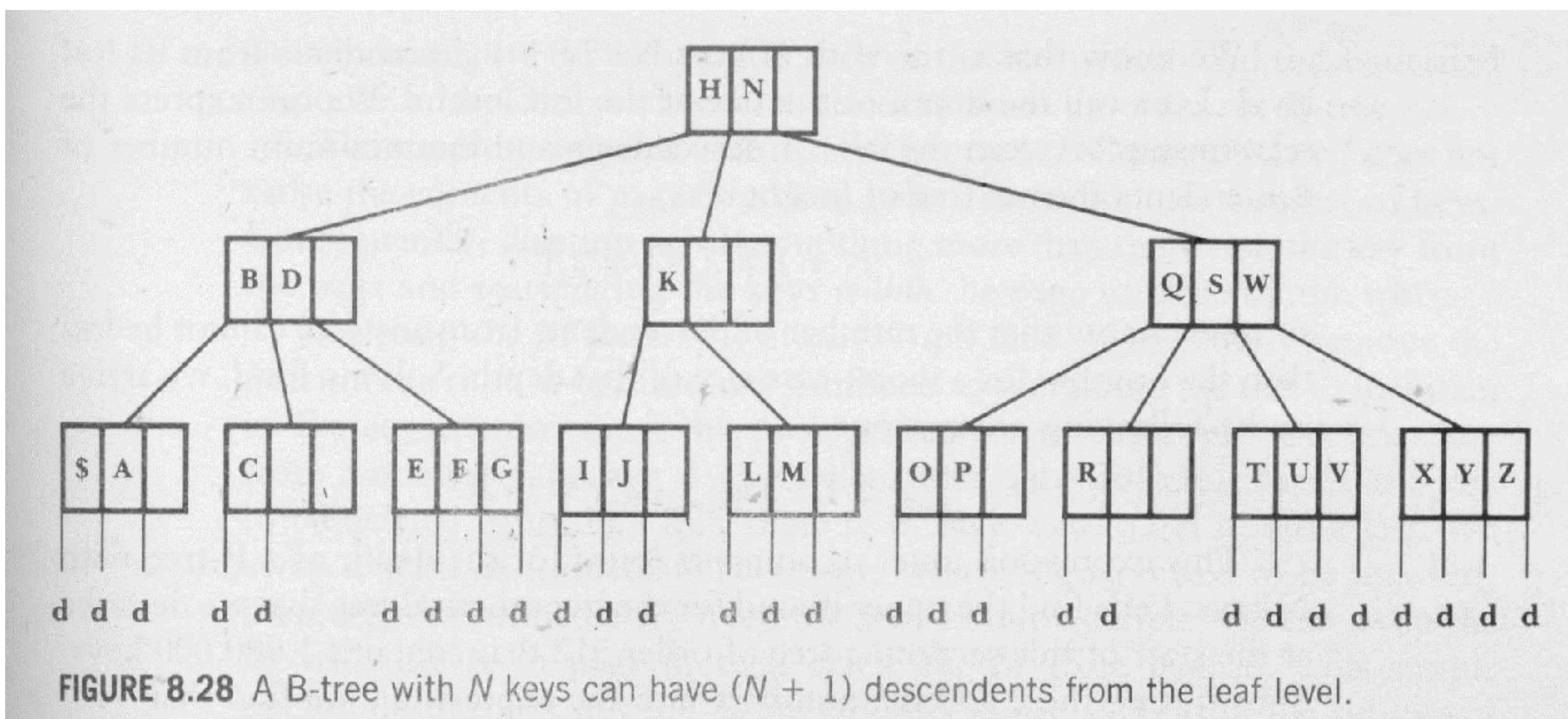
- Tendo N chaves na árvore-B de ordem m , qual o número de acessos a disco necessário?
- Ou, mais apropriado, “**qual a altura da árvore**”?

Propriedades das Árvores-B



- Antes de mais nada, é importante observar

- número de descendentes possíveis de um nível da árvore = número de chaves até o nível atual + 1

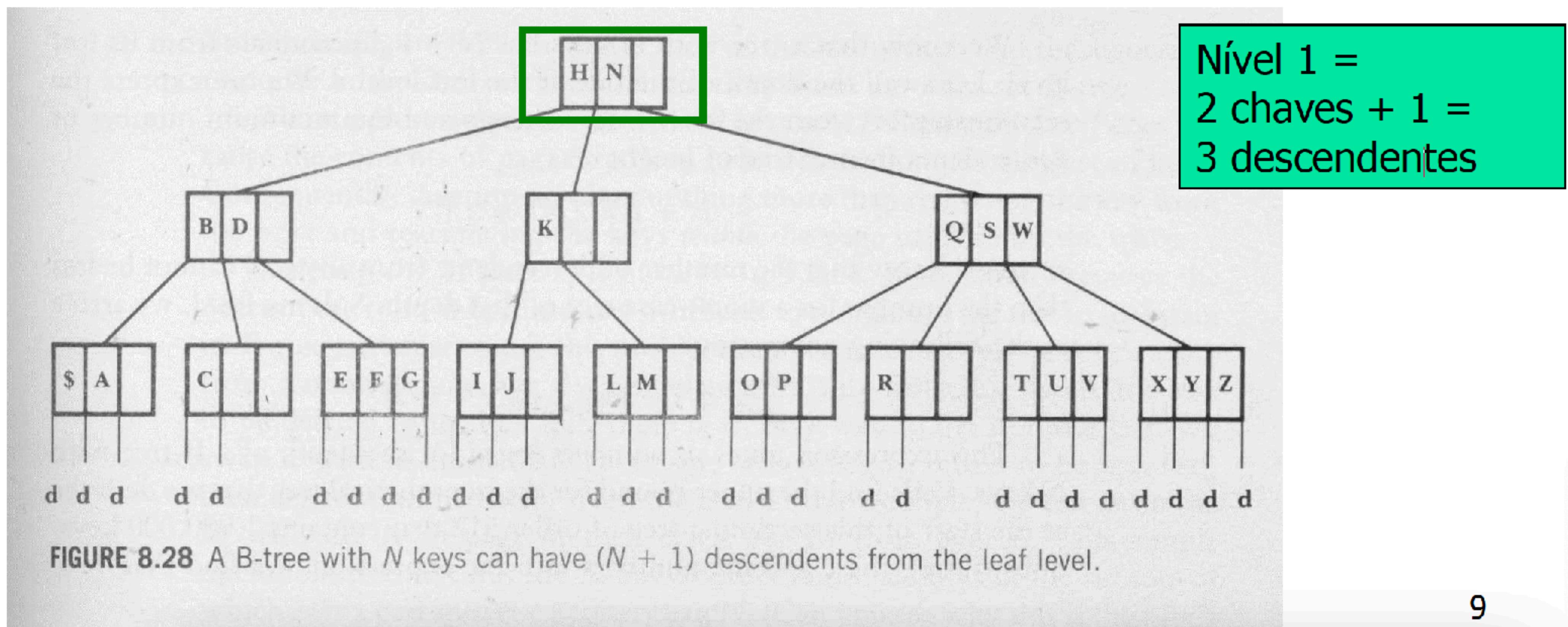


Propriedades das Árvores-B



- Antes de mais nada, é importante observar

- número de descendentes possíveis de um nível da árvore = número de chaves até o nível atual + 1

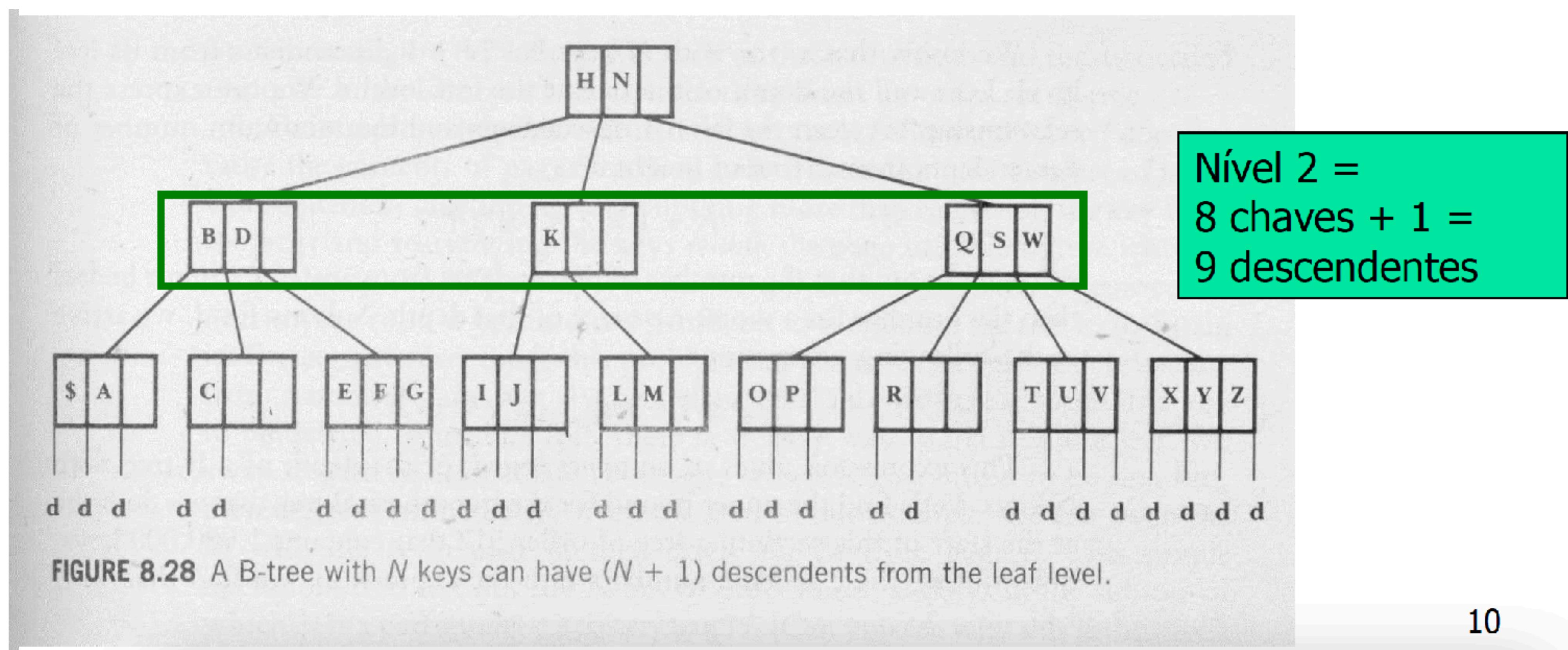


Propriedades das Árvores-B



- Antes de mais nada, é importante observar

- número de descendentes possíveis de um nível da árvore = número de chaves até o nível atual + 1

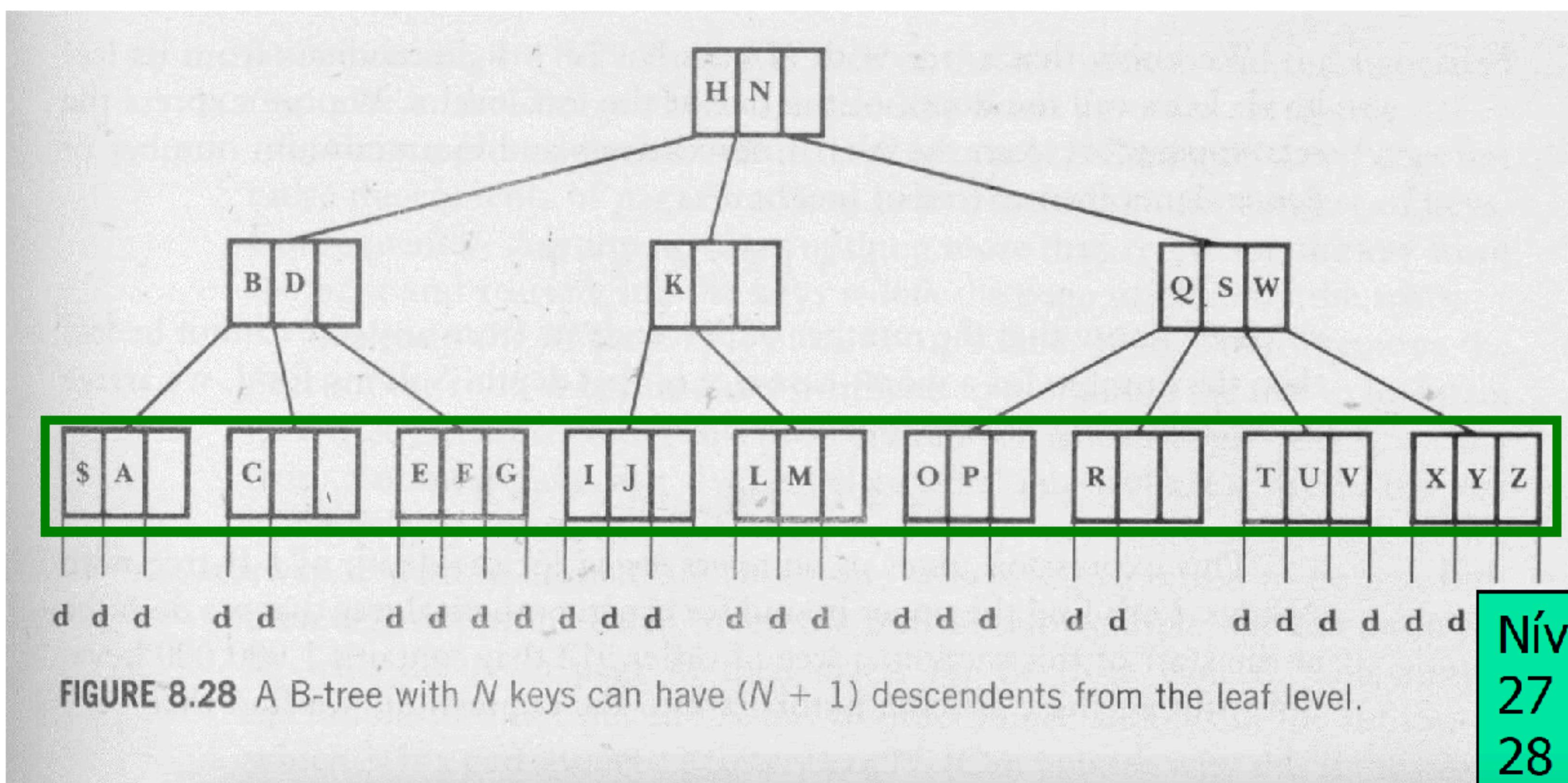


Propriedades das Árvores-B



- Antes de mais nada, é importante observar

- número de descendentes possíveis de um nível da árvore = número de chaves até o nível atual + 1





- No pior caso, cada nó da árvore terá o número mínimo possível de descendentes
 - A árvore terá sua maior altura e menor largura
 - Exceto raiz e folhas, há no mínimo $\lceil m/2 \rceil$ descendentes para cada nó
- Para uma árvore de **ordem m**
 - A raiz (primeiro nível) terá no mínimo 2 descendentes
 - O segundo nível terá somente 2 páginas, tendo cada uma $\lceil m/2 \rceil$ descendentes, ou seja, há $2 \times \lceil m/2 \rceil$ descendentes para o segundo nível
 - O terceiro nível contém $2 \times \lceil m/2 \rceil$ nós x $\lceil m/2 \rceil$ descendentes para cada nó, ou seja $2 \times \lceil m/2 \rceil^2$
 - O nível d terá $2 \times \lceil m/2 \rceil^{d-1}$

Propriedades das Árvores-B



- Ou seja, o número mínimo de descendentes para um nível d da árvore é $2 \times \lceil m/2 \rceil^{d-1}$
- Sabe-se que, no máximo, há $N+1$ descendentes em um nível da árvore com N chaves até então
- Então, podemos calcular o limite superior da profundidade da árvore (=número máximo de acessos a disco)

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

- Exemplo
 - Considerando que temos ($N=$)1,000,000 chaves e uma árvore de ordem ($m=$)512, temos que $d \leq 1 + \log_{\lceil 256 \rceil}(500,000.5)$, ou seja, $d \leq 3.37$
 - Podemos esperar, portanto, não mais do que 3 acessos a disco para acessar qualquer uma das chaves



- FOLK, M.J. File Structures, Addison-Wesley, 1992.
- File Structures: Theory and Practice”, P. E. Livadas, Prentice-Hall, 1990;
- Contém material extraído e adaptado das notas de aula dos professores Moacir Ponti, Thiago Pardo, Leandro Cintra, Thelma Cecília Chiossi e Maria Cristina de Oliveira.