

Árvores-B: Algoritmos de Pesquisa e de Inserção

Profa. Dra. Cristina Dutra de Aguiar Ciferri

Algoritmos

- Estrutura de dados
 - determina cada página de disco
 - pode ser implementada de diferentes formas
 - Implementação adotada
 - contador de ocupação \Rightarrow número de chaves por página
 - chaves \Rightarrow caracteres
 - ponteiros \Rightarrow campos de referência para cada chave
-

Declaração da Página

```
In C:
struct BTPAGE {
    short KEYCOUNT;          /* number of keys stored in PAGE */
    char  KEY[MAXKEYS];       /* the actual keys                */
    short CHILD[MAXKEYS+1];   /* RRNs of children              */
} PAGE;

In Pascal:
TYPE
    BTPAGE = RECORD
        KEYCOUNT: integer;
        KEY       : array[1..MAXKEYS] of char;
        CHILD     : array[1..MAXCHILDREN] of integer
    END;
VAR
    PAGE : BTPAGE;
```

MAXKEYS: número máximo de chaves por página de disco

MAXCHILDREN: número máximo de ponteiros para páginas de disco

Declaração da Página

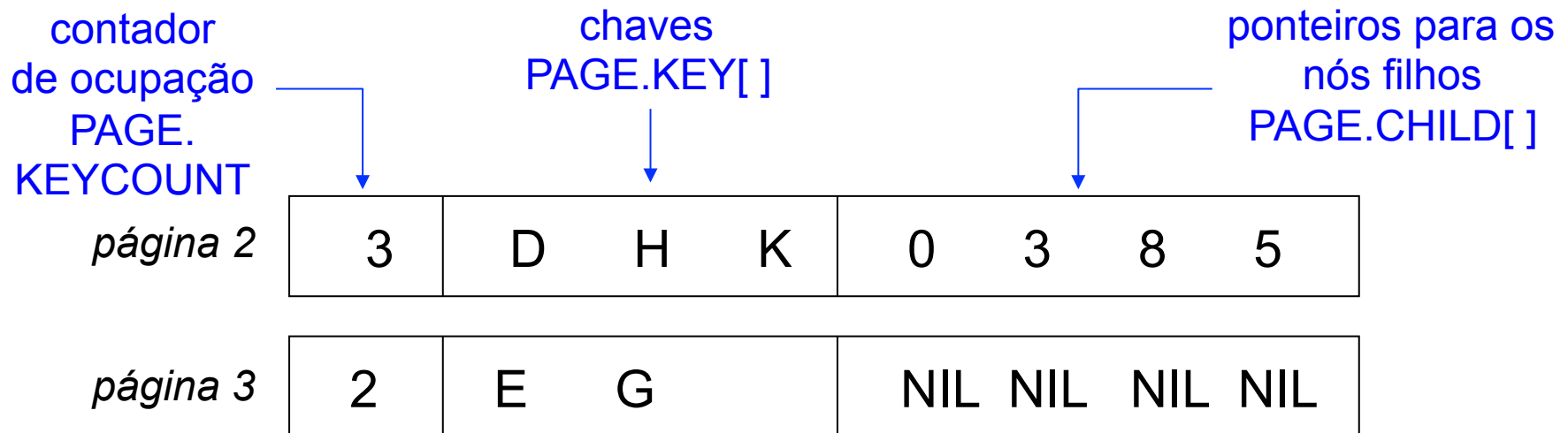
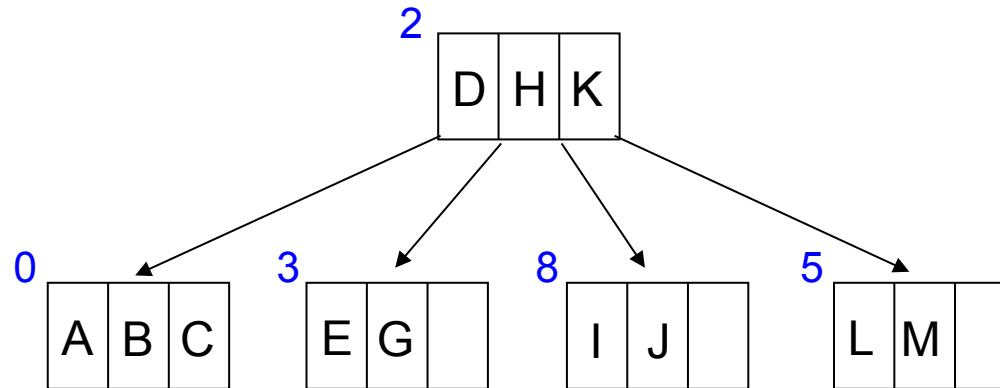
```
In C:
struct BTPAGE {
    short KEYCOUNT; /* number of keys stored in PAGE */
    char KEY[MAXKEYS]; /* the actual keys */
    short CHILD[MAXKEYS+1]; /* RRNs of children */
} PAGE;

In Pascal:
TYPE
    BTPAGE = RECORD
        KEYCOUNT: integer;
        KEY : array[1..MAXKEYS] of char;
        CHILD : array[1..MAXCHILDREN] of integer
    END;
VAR
    PAGE : BTPAGE;
```

PAGE.KEYCOUNT: determina se a página está cheia ou não

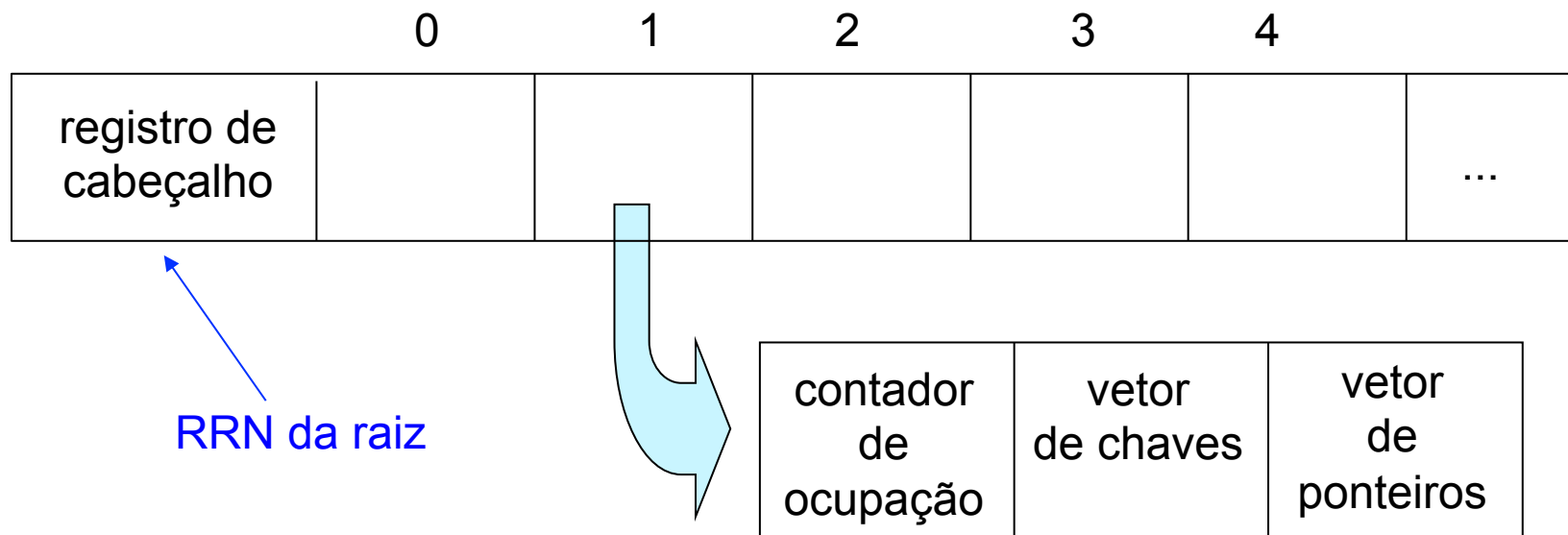
PAGE.CHILD[]: contém os RRN dos nós-filhos ou -1 (ou NIL) se não houver descendentes

Arquivo da Árvore-B



Arquivo da Árvore-B

- Conjunto de registros de tamanho fixo



- Cada registro
 - ocupa uma página de disco
-

Algoritmos

- Operações básicas
 - pesquisa, inserção e remoção
 - Características gerais
 - recursivos
 - dois estágios de processamento
 - em páginas inteiras *e então*
 - dentro das páginas
-

Algoritmo: Pesquisa (1/2)

```
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
                                pesquisa a página procurando a chave de busca
```

Algoritmo: Pesquisa (2/2)

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

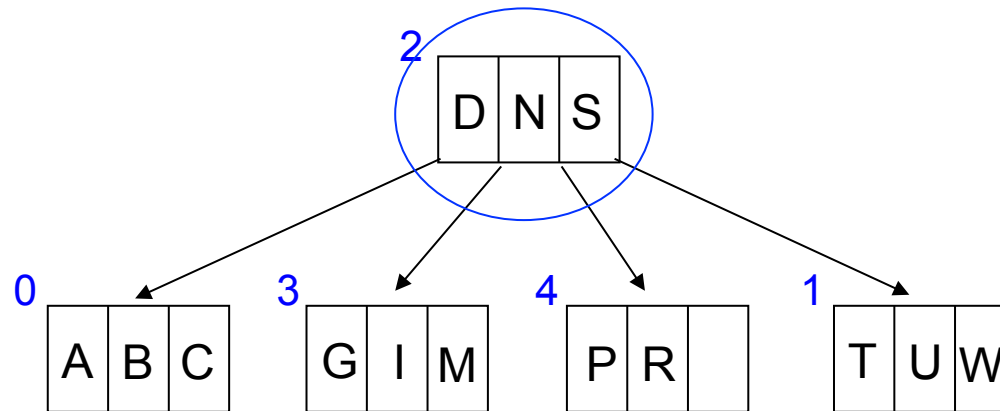
return (search(PAGE.CHILD[POS], KEY, FOUND_RRN,
 FOUND_POS))

endif

endif

end FUNCTION

Busca da Chave K



- search (2, K, FOUND_RRN, FOUND_POS)

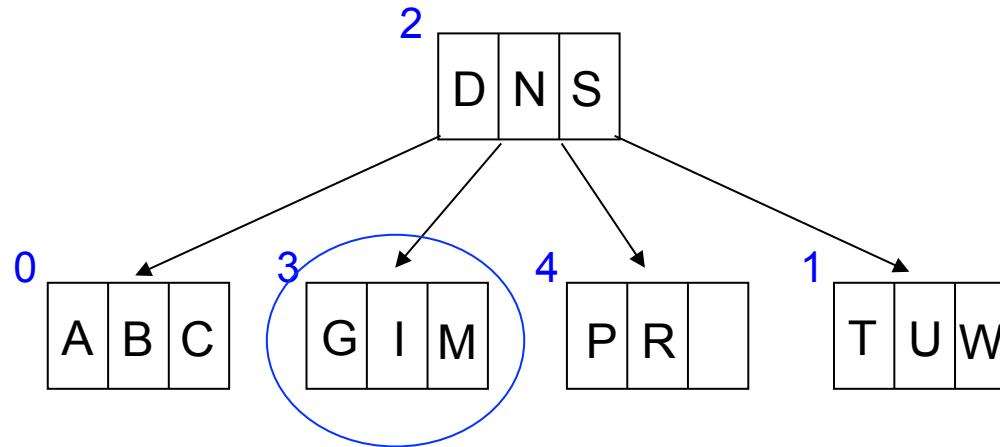
PAGE =

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

não existe → POS = 1

... PAGE.CHILD[1]

Busca da Chave K



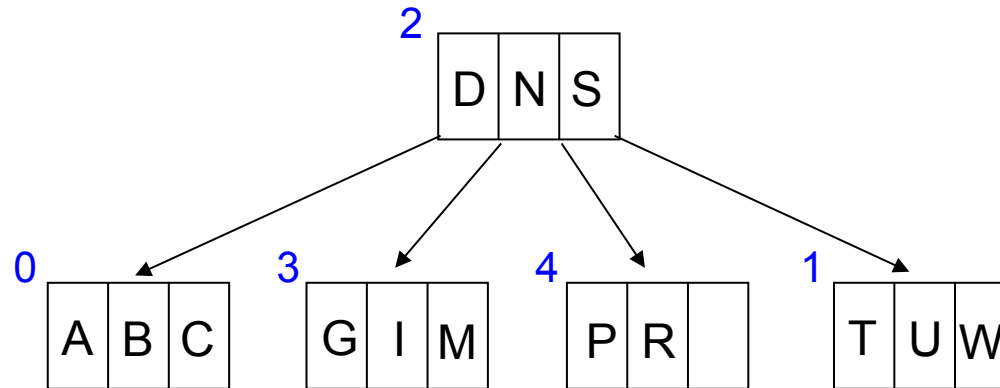
- search (PAGE.CHILD[1], K, FOUND_RRN, FOUND_POS)

PAGE =

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

não existe → POS = 2
... PAGE.CHILD[2]

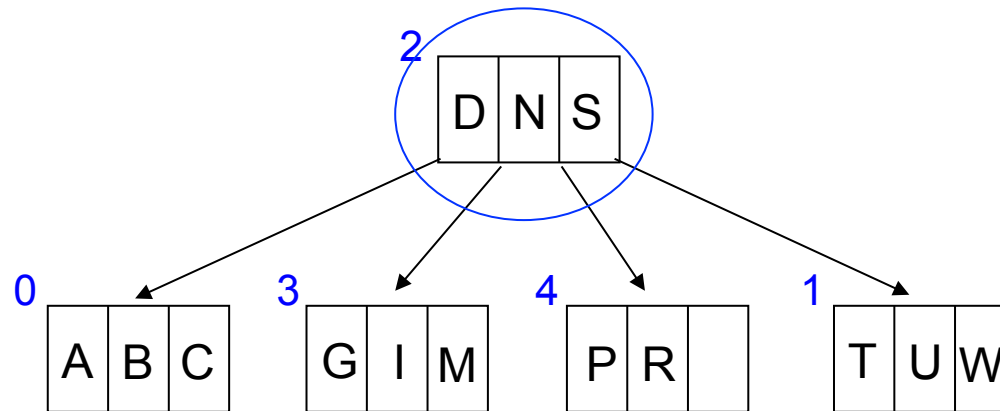
Busca da Chave K



- 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



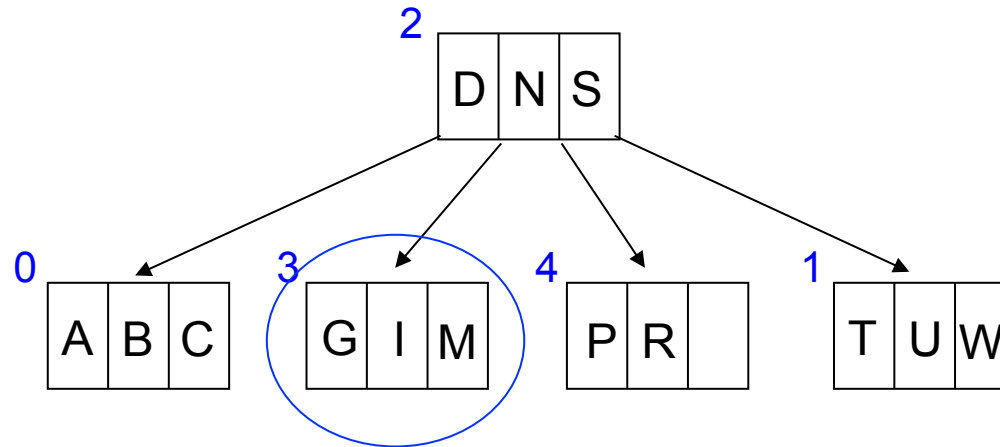
- search (2, M, FOUND_RRN, FOUND_POS)

PAGE =

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

 não existe → POS = 1

Busca da Chave M



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

PAGE =

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

chave de busca encontrada

POS = FOUND_POS = 2

FOUND_RRN = 3

return FOUND

Algoritmos: Inserção

- 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 (i.e., *split*) e promoção (i.e., *promotion*) propagam-se em direção à raiz
 - construção *bottom-up*
-

Algoritmos: Inserção

- Fases (procedimento recursivo)
 - busca pela página
 - pesquisa da página antes da chamada recursiva
 - chamada recursiva
 - move a operação para os níveis inferiores da árvore
 - inserção, *split* e *promotion*
 - executados após a chamada recursiva
 - a propagação destes processos ocorre no retorno da chamada recursiva

caminho inverso
ao da pesquisa

Função Insert

insert (CURRENT_RRN, KEY, PROMO_KEY, PROMO_R_CHILD)

- **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 // **parâmetro de retorno da recursão**

- retorna a chave promovida, caso a inserção resulte no particionamento e na promoção da chave
-

Função Insert

insert (CURRENT_RRN, KEY, PROMO_KEY, PROMO_R_CHILD)

- **Parâmetros**

- **PROMO_R_CHILD** // parâmetro de retorno da recursão

- 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

Função Insert

insert (CURRENT_RRN, KEY, PROMO_KEY, PROMO_R_CHILD)

- **Valores de retorno**

- PROMOTION

- quando uma inserção é feita e uma chave é promovida ⇒ **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**
-

Função Insert

insert (CURRENT_RRN, KEY, PROMO_KEY, PROMO_R_CHILD)

- **Variáveis locais**

- PAGE

- página de disco correntemente examinada pela função

- NEWPAGE

- página de disco nova resultante do particionamento

- POS

- posição na página (i.e., PAGE) na qual a chave ocorre ou deveria ocorrer
-

Função Insert

insert (CURRENT_RRN, KEY, PROMO_KEY, PROMO_R_CHILD)

- Variáveis locais

- 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



Algoritmo: Inserção (1/3)

```
FUNCTION: insert (CURRENT_RRN, página a ser pesquisada
                KEY, chave a ser inserida
                PROMO_R_CHILD, RRN filho direito PROMO_KEY
                PROMO_KEY) chave promovida
if CURRENT_RRN == NIL then construção a partir das folhas (bottom)
    PROMO_KEY = KEY
    PROMO_R_CHILD = NIL
    return PROMOTION
else ....
```

se a página não é um nó folha, a função é chamada recursivamente até que ela encontre uma KEY ou chegue o nó folha

Algoritmo: Inserção (2/3)

read page at CURRENT_RRN into PAGE

search for KEY in PAGE, setting POS to be equal to the position
where KEY occurs or should occur

pesquisa a página procurando a chave de busca

if KEY was found then

issue error message indicating duplicate key

return ERRO chave de busca já existe

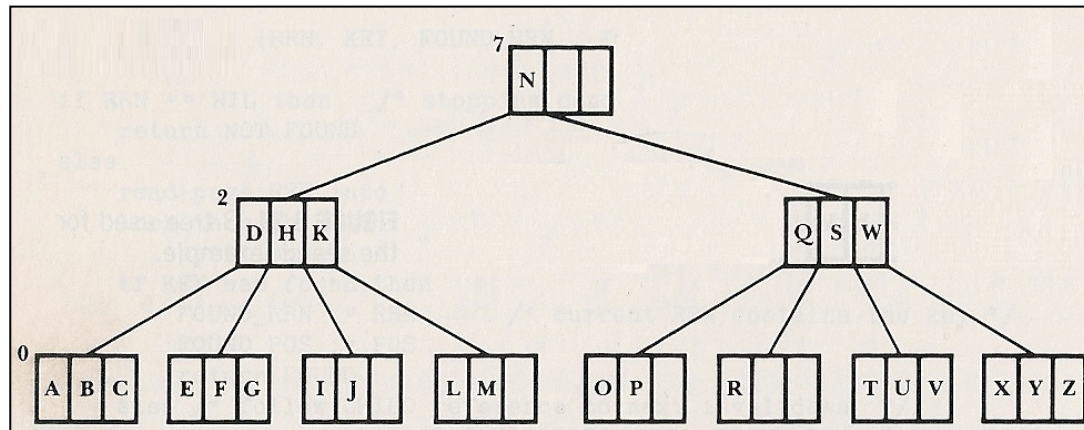
a chave de busca não foi encontrada, portanto
procura a chave de busca no nó filho

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

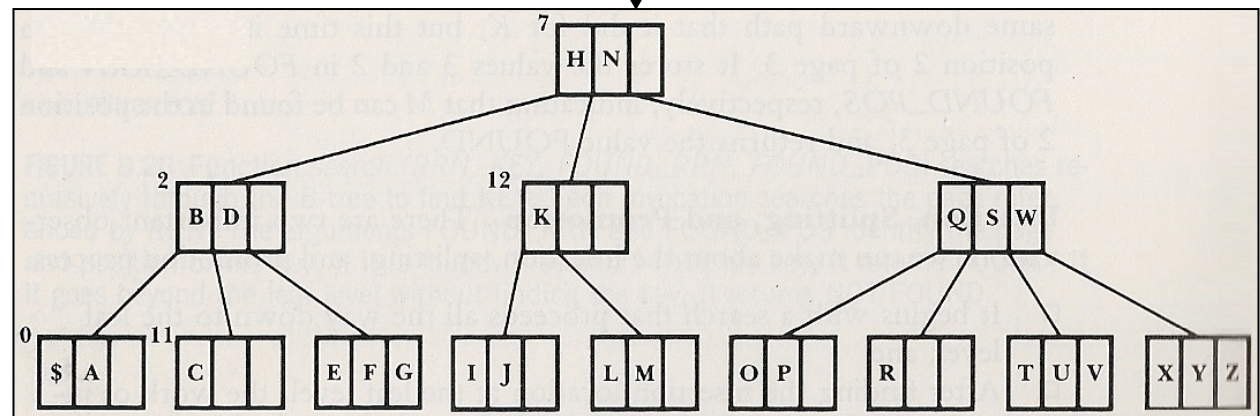
Algoritmo: Inserção (3/3)

```
if RETURN_VALUE == NO PROMOTION or ERROR then
    return RETURN_VALUE
elseif there is space in PAGE for P_B_KEY then
    insert P_B_KEY and P_B_RRN in PAGE
    return NO PROMOTION    inserção sem particionamento
else    inserção sem particionamento, indicando chave promovida
    split (P_B_KEY, P_B_RRN, PAGE, PROMO_KEY,
          PROMO_R_CHILD, NEWPAGE)
    write PAGE to file at CURRENT_RRN
    write NEWPAGE to file at RRN PROMO_R_CHILD
    return PROMOTION
endif
end FUNCTION
```

Exemplo: Inserção do \$



inserindo \$,
sendo que $\$ < A$



fase de pesquisa

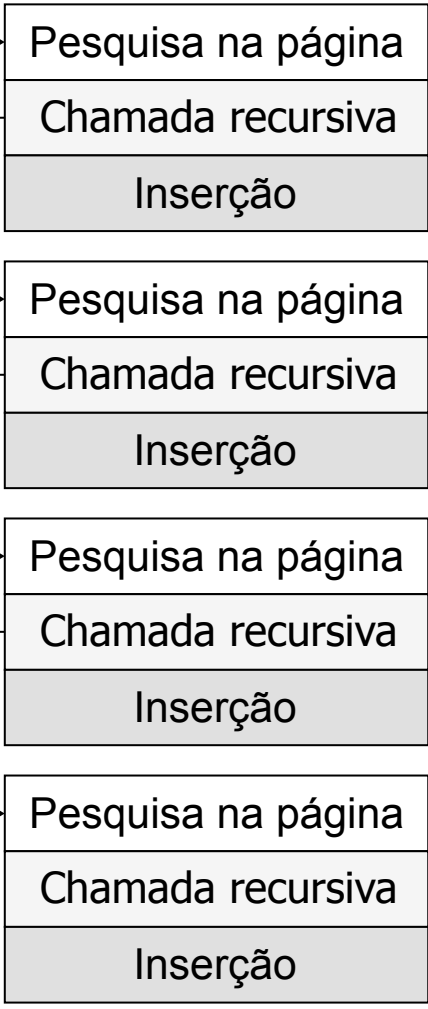
Exemplo: Inserção do \$

KEY=\$
CURRENT_RRN=7

KEY=\$
CURRENT_RRN=2
POS=0
PAGE.CHILD[0]

KEY=\$
CURRENT_RRN=0
POS=0
PAGE.CHILD[0]

KEY=\$
CURRENT_RRN=NIL



PROMO_R_CHILD=undefined
PROMO_R_CHILD=undefined
return value: NO PROMOTION

PROMO_R_CHILD=H
PROMO_R_CHILD=12
return value: PROMOTION

PROMO_KEY=B
PROMO_R_CHILD=11
return value: PROMOTION

PROMO_KEY=\$
PROMO_R_CHILD=NIL
return value: PROMOTION

fase da volta da recursão

Observações

- Fase de pesquisa
 - somente CURRENT_RRN é modificado à medida que ocorre a recursão no caminho de busca da árvore
 - termina quando CURRENT_RRN = NIL
 - Fase da volta da recursão
 - executa a lógica da inserção e do *split*
 - *se o valor retornado é PROMOTION, insere-se uma chave no nível corrente*
 - *caso contrário, apenas retorna para o nível superior*
-

Função Split

insert (I_KEY, I_RRN, PAGE, PROMO_KEY, PROMO_R_CHILD, NEWPAGE)

- 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
-

Algoritmo: Split (1/2)

PROCEDURE: split (I_KEY, nova chave a ser inserida
I_RRN, filho a direita da nova chave a ser inserida
PAGE, página de disco corrente
PROMO_KEY, chave promovida
PROMO_R_CHILD, filho a direita da chave promovida
NEWPAGE) nova página de disco

copy all keys and pointers from PAGE into a working page that can hold one extra key and child

insert I_KEY and I_RRN into their proper place in the working page

allocate and initialize a new page in the B-tree file to hold NEWPAGE

Algoritmo: Split (2/2)

set PROMO_KEY to the value of middle key, which will be promoted after the split

set PROMO_R_CHILD to RRN of NEWPAGE

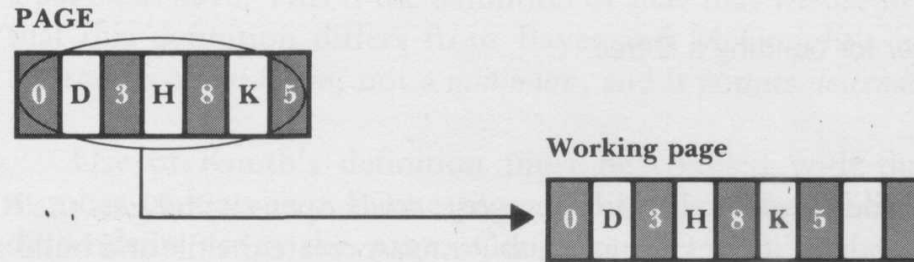
copy keys and child pointers preceding PROMO_KEY from the working page to PAGE

copy keys and child pointers following PROMO_KEY from the working page to NEWPAGE

end PROCEDURE

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

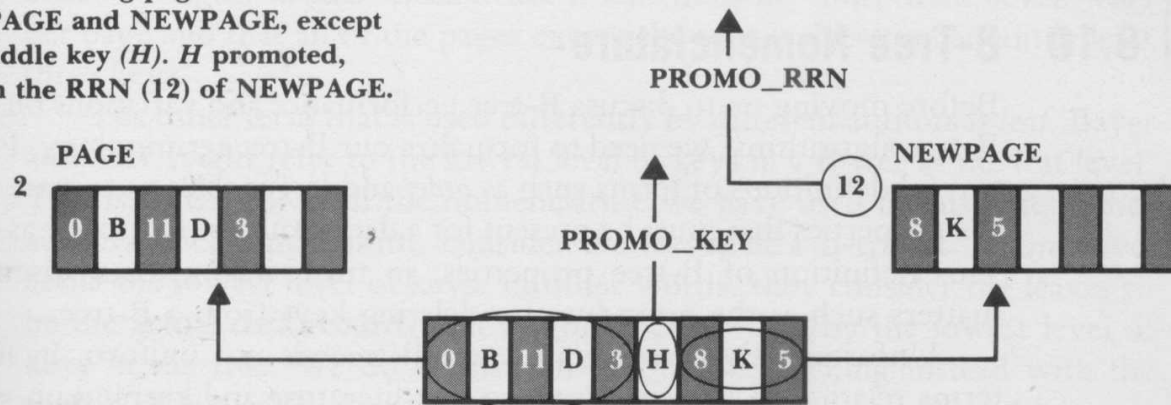
Contents of **PAGE** are copied to the working page.



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
 - essa chave sai da página de trabalho corrente
- Todos os RRN dos nós filhos
 - transferidos de volta entre 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!

Procedimento Driver

- 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 the B-tree file exists then

 open B-tree file

else create a B-tree file and place the first key in the root

get RRN of root page from file and store it in ROOT

get a key and store it in KEY

while keys exist

 if (insert (ROOT, KEY, PROMO_R_CHILD, PROMO_KEY) == PROMOTION) then

 create a new root page with key := PROMO_KEY, left child := ROOT and
 right child := PROMO_R_CHILD

 set ROOT to RRN of new root page

 get next key and store it in KEY

endwhile

write RRN stored in ROOT back to B-tree file

close B-tree file

end MAIN PROCEDURE
