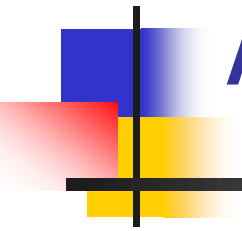
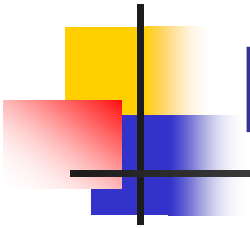


Árvores-B (Parte IIa)

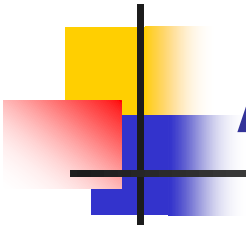
SCC203 – Algoritmos e Estruturas de Dados II

Graça Nunes



Relembrando

- 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)



Algoritmos

- 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 $\Rightarrow$ caracteres (por simplicidade)
  - ponteiros $\Rightarrow$ campos de referência para as páginas filhas

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

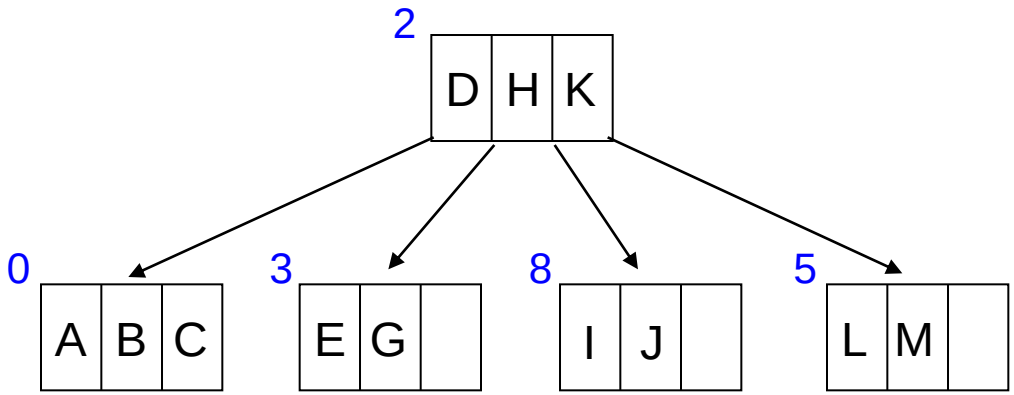
Declaração da Página

```
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



contador de ocupação PAGE. KEYCOUNT

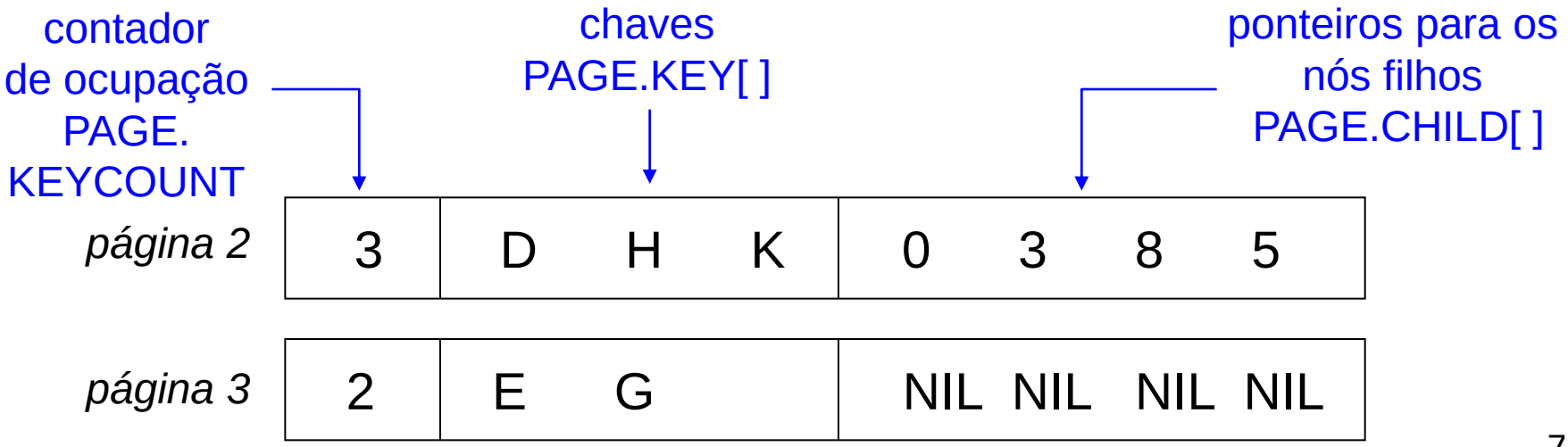
chaves PAGE.KEY[ ]

ponteiros para os nós filhos PAGE.CHILD[ ]

página 2	3	D	H	K	0	3	8	5
página 3	2	E	G		NIL	NIL	NIL	NIL

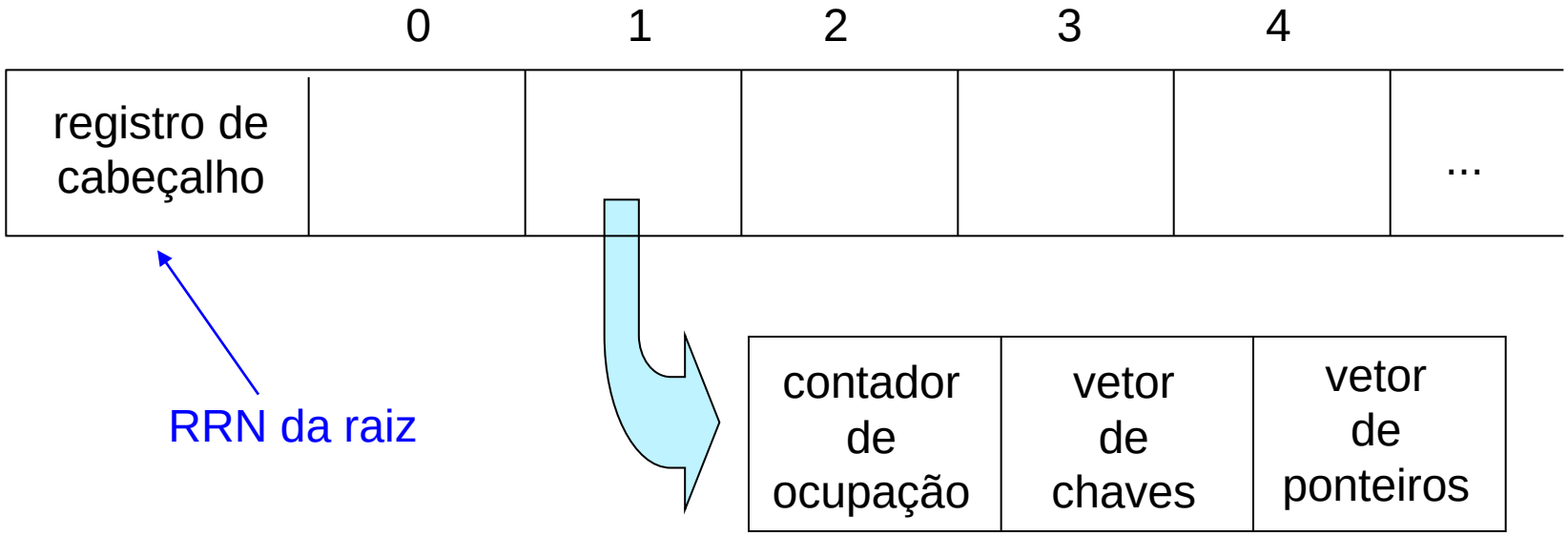
Arquivo da Árvore-B

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

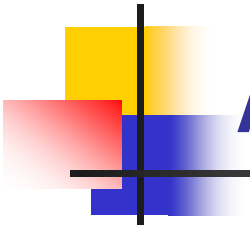


Arquivo da Árvore-B

- Conjunto de registros de tamanho fixo



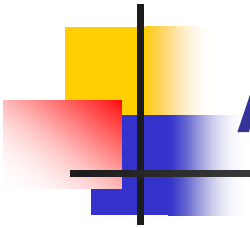
- Cada registro
  - contém uma página de disco



Algoritmos

- Operações básicas
  - Pesquisa, inserção e remoção

- Características gerais dos algoritmos
  - algoritmos recursivos
  - dois estágios de processamento
    - em páginas inteiras e...
    - dentro das páginas



Algoritmo: Pesquisa

- Exercício: escreva o algoritmo de busca em árvore binária

Algoritmo: Pesquisa

- 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

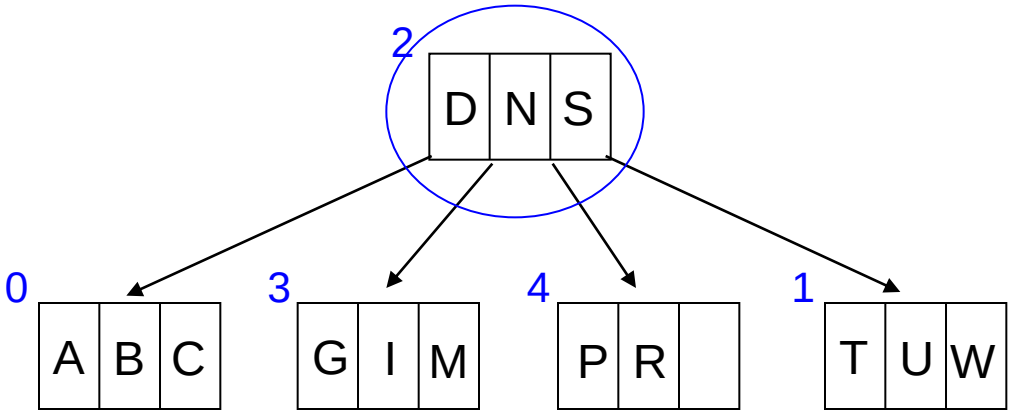
apropriada da raiz

Refinando o Algoritmo

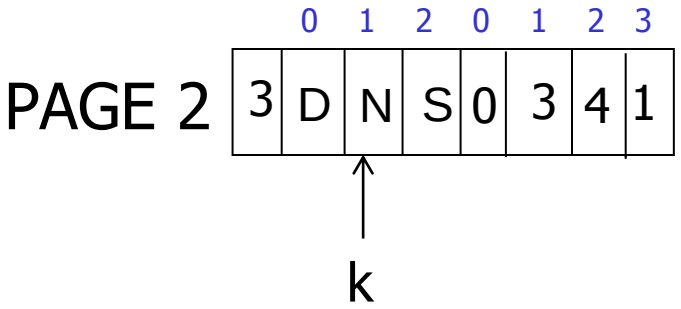
```
FUNCTION: search (RRN,                  //página a ser pesquisada
                  KEY,                  //chave sendo procurada
                  FOUND_RRN,            //página que contém a chave
                  FOUND_POS)            //posição da chave na página

if  RRN == NIL then
     return NOT FOUND          //chave de busca não encontrada

else
     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

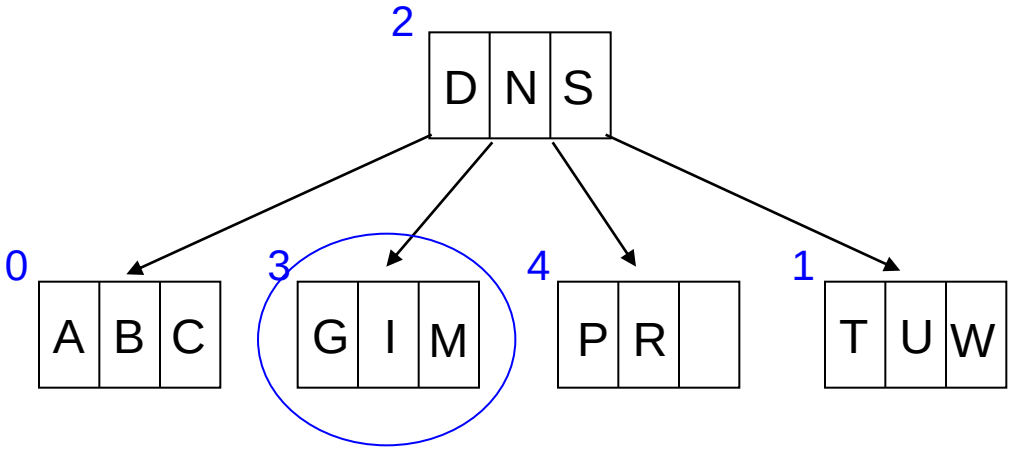


- search (2, K, FOUND_RRN, FOUND_POS)

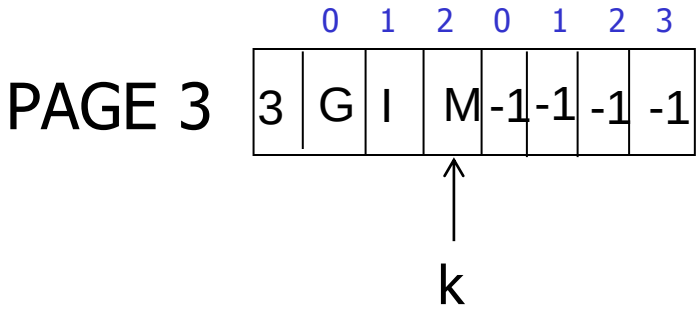


não existe → POS = 1

Busca da Chave K

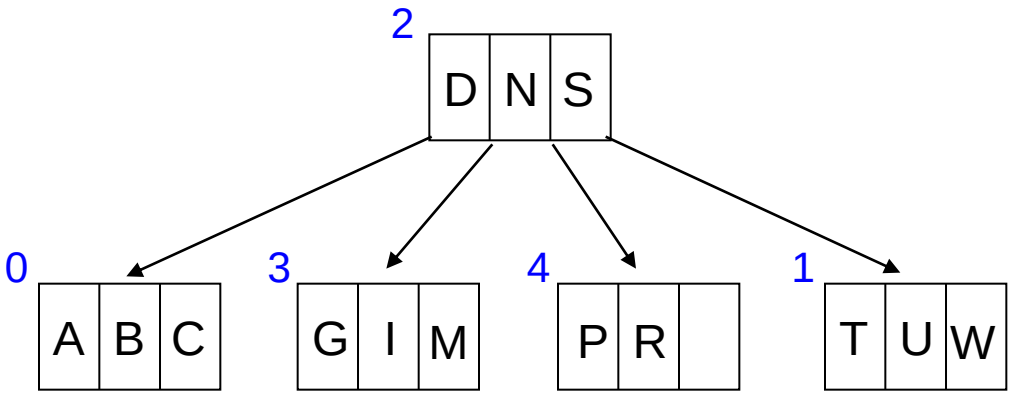


- search (PAGE.CHILD[1], K, FOUND_RRN, FOUND_POS)



não existe → POS = 2

Busca da Chave K

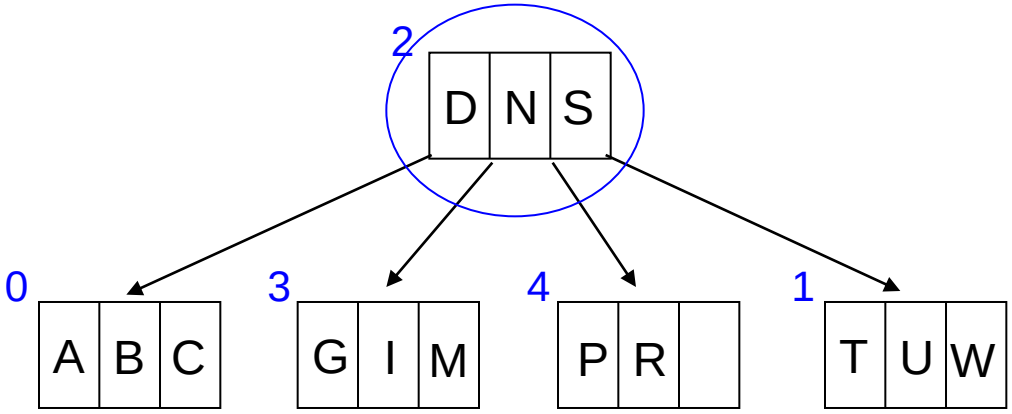


- search (PAGE.CHILD[2], K, FOUND_RRN, FOUND_POS)

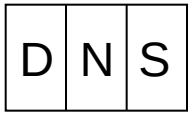
PAGE.CHILD[2] = NIL $\rightarrow$ chave de busca não encontrada

return NOT FOUND

Busca da Chave M

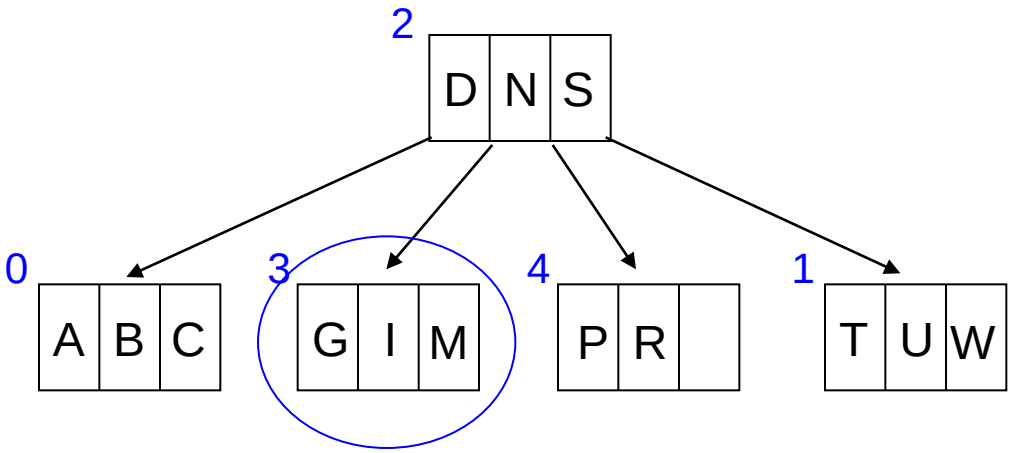


- search (2, M, FOUND_RRN, FOUND_POS)

PAGE = 

não existe → POS = 1

Busca da Chave M



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

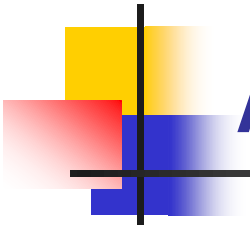
PAGE = 

chave de busca encontrada

POS = FOUND_POS = 2

FOUND_RRN = 3

return FOUND



Algoritmo: Inserção

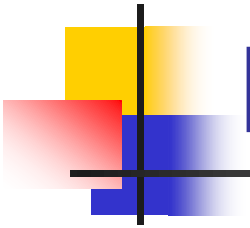
- Observações gerais
  - inicia-se com uma pesquisa que desce até o nível dos nós folhas
  - uma vez escolhido o nó folha no qual a nova chave deve ser inserida, os processos de **inserção**, **particionamento** (*split*) e **promoção** (*promotion*) propagam-se em direção à raiz
    - construção *bottom-up*

Algoritmo: Inserção

- Fases (procedimento recursivo)

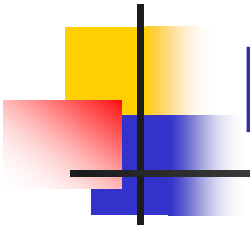
1. busca pela página *(search step)*
   - pesquisa da página antes da chamada recursiva
2. chamada recursiva *(recursive call)*
   - move a operação para os níveis inferiores da árvore
3. inserção, *split* e *promotion* *(insertion and splitting logic)*
   - executados após a chamada recursiva
   - a propagação destes processos ocorre no retorno da chamada recursiva

caminho inverso ao da pesquisa



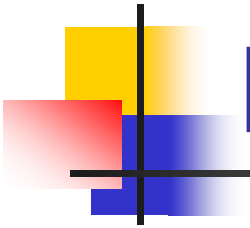
Função Insert

- 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



Função Insert

- 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



Função Insert

- 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

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

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

...
```

Search step

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 $

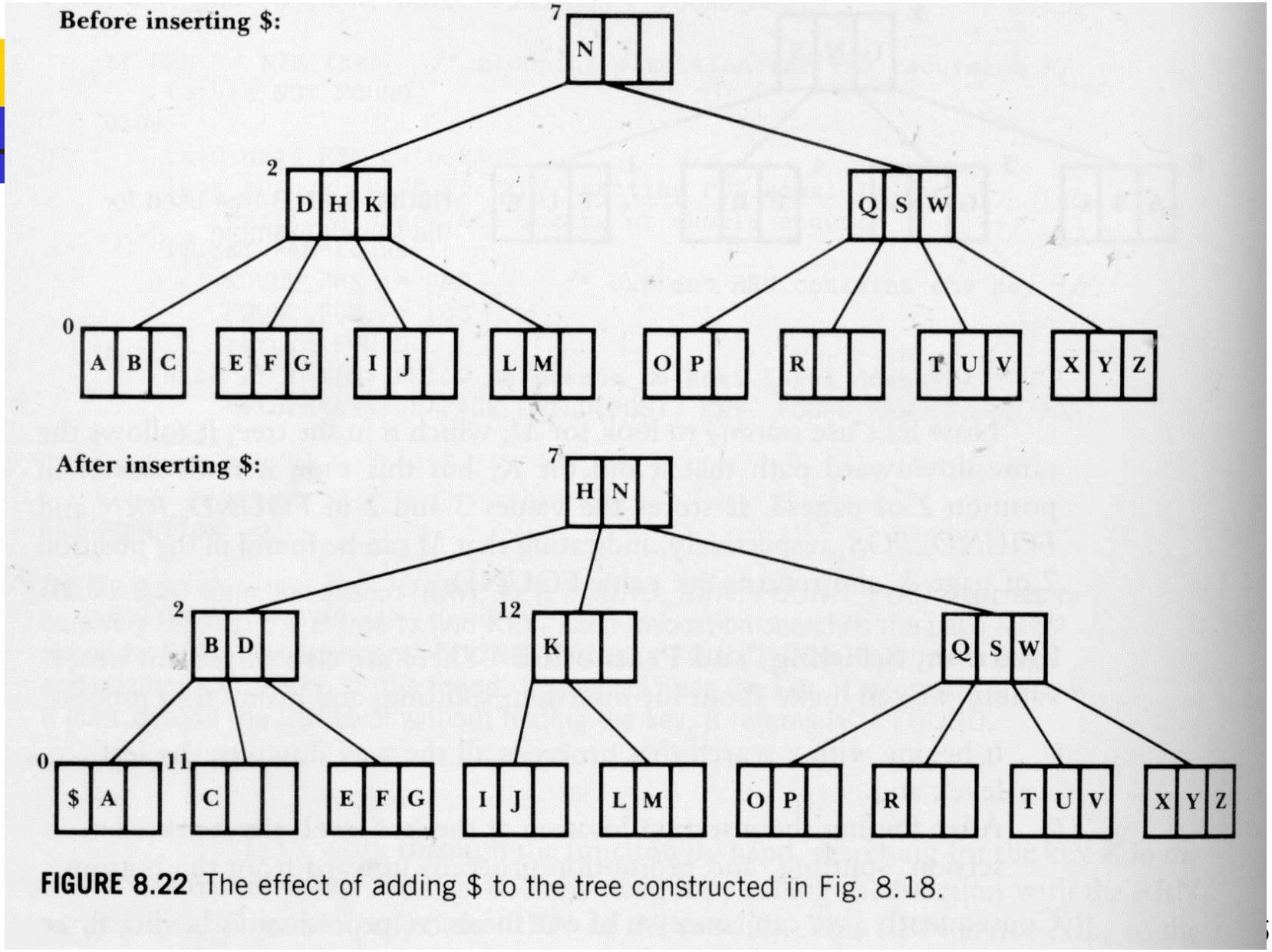


FIGURE 8.22 The effect of adding $ to the tree constructed in Fig. 8.18.

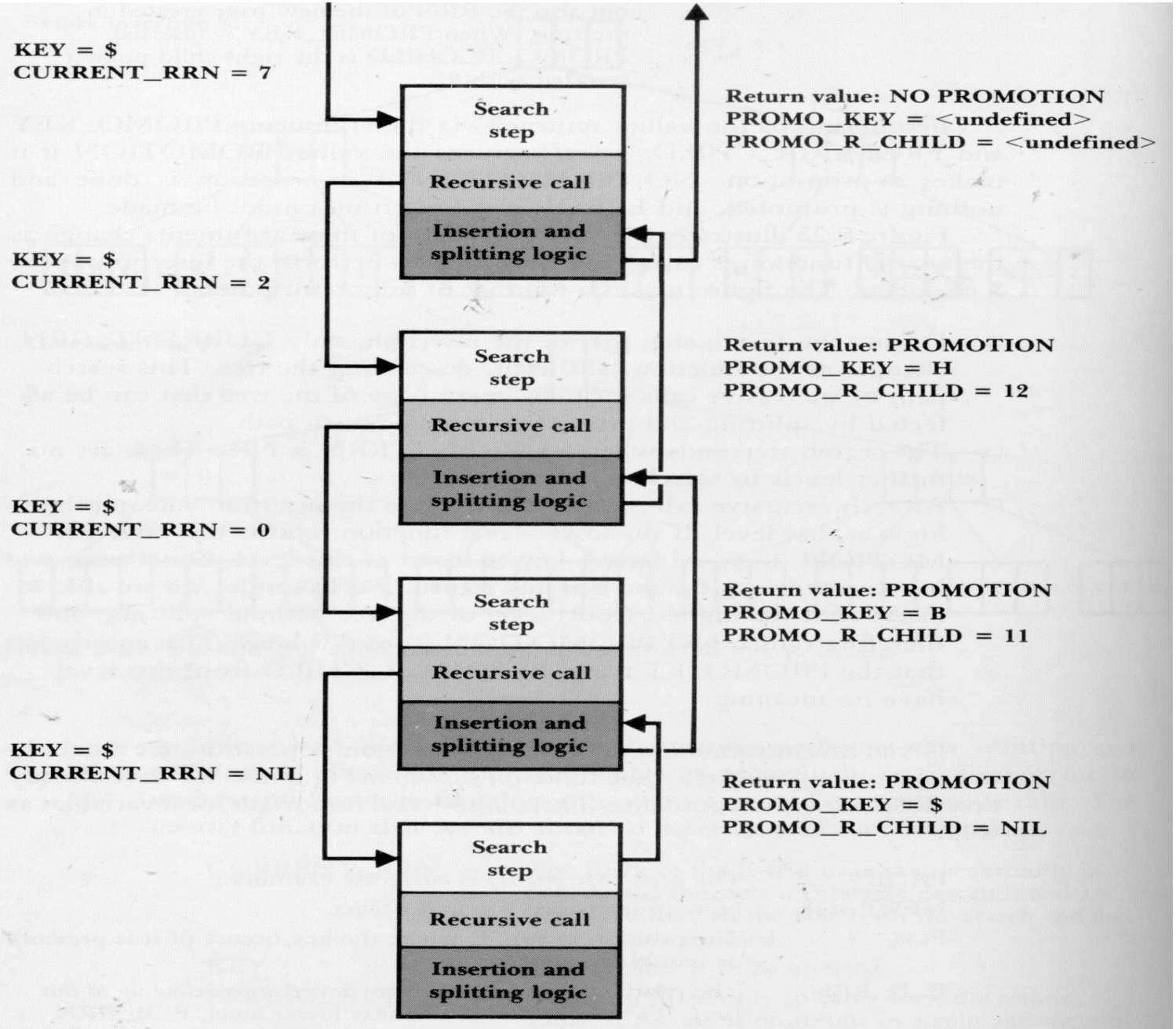
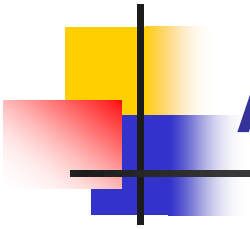
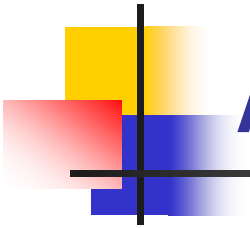


FIGURE 8.23 Pattern of recursive calls to insert $ into the B-tree as illustrated in Fig. 8.22.



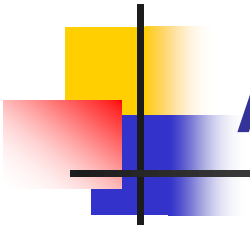
A Função Split

- Split(I_KEY, I_RRN, PAGE, PROMO_KEY, PROMO_R_CHILD, NEWPAGE)
- Parâmetros
  - I_KEY, I_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



A Função Split

- 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

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

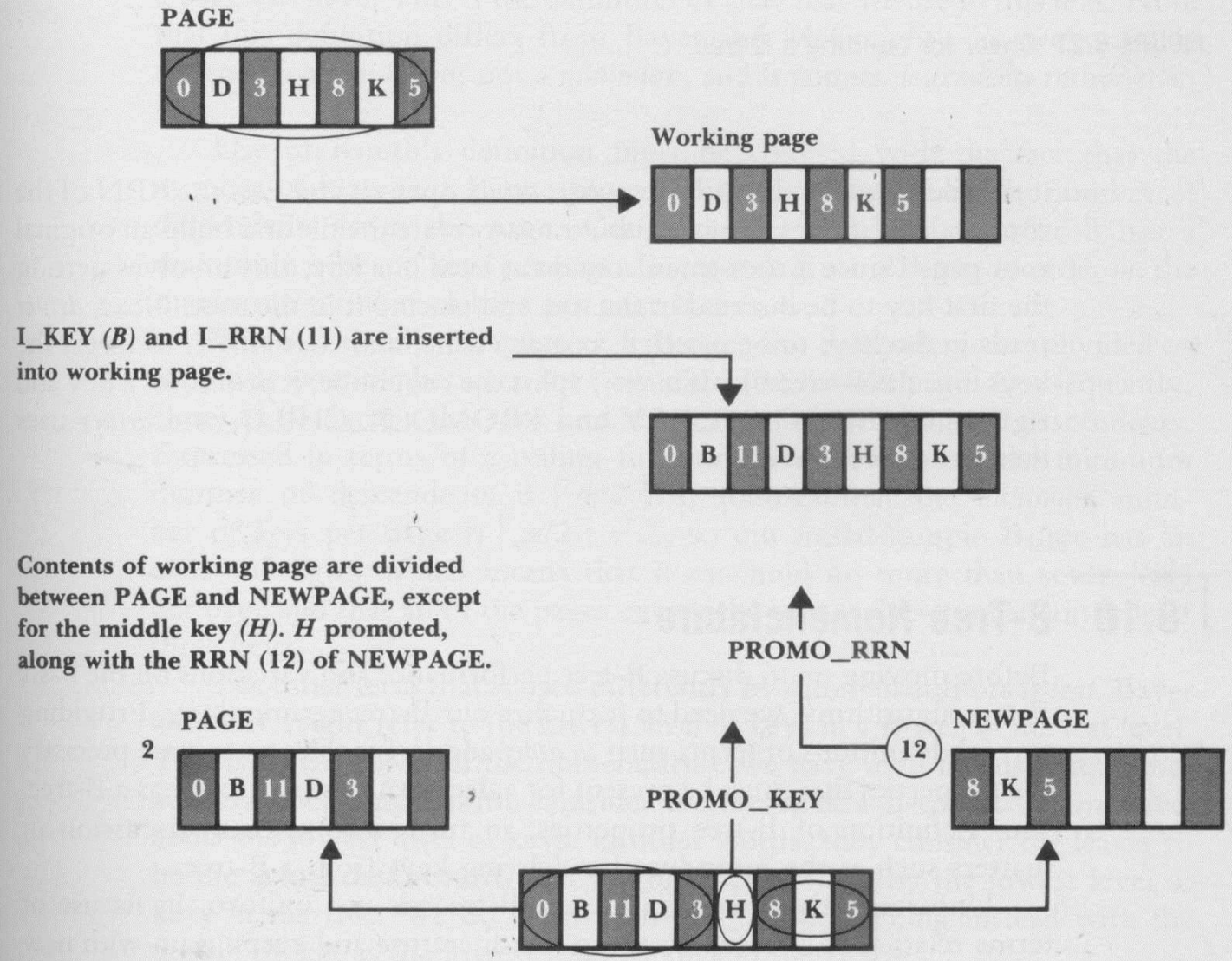
faça PROMO_KEY igual à chave do meio da página temporária

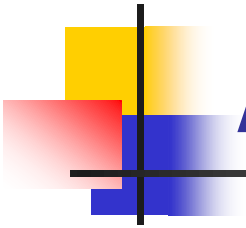
faça PROMO_R_CHILD igual ao RRN da nova página

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

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

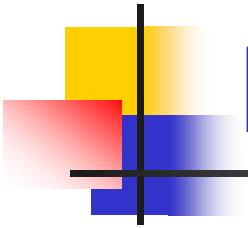




A Função Split

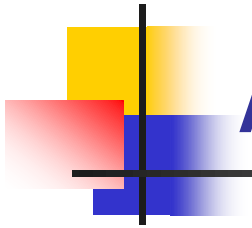
- Observações
  - somente uma chave é promovida e sai da página de trabalho atual
  - todos os RRN dos nós filhos são transferidos de volta para PAGE e NEWPAGE
  - o RRN promovido é o de NEWPAGE
    - NEWPAGE é a descendente direita da chave promovida

Note que a função *split* move os dados!



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 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
```