Árvores B (Parte III)

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
  - chaves ⇒ caracteres
  - ponteiros ⇒ 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

MAXCHILDREN: número máximo de ponteiros para páginas de disco

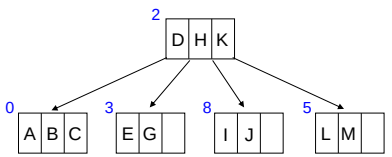
Declaração da Página

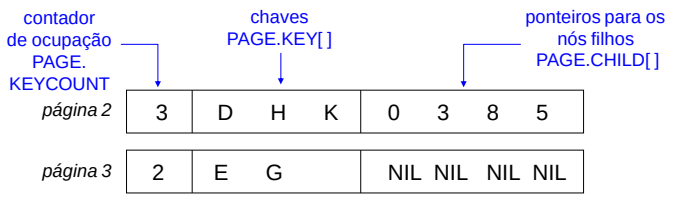
```
In C:
struct BTPAGE {
    short    KEYCOUNT;          /* number of keys stored in PAGE */
    char     KEY[MAXKEYS];      /* the actual keys               */
    short    CHILD[MAXKEYS+1]; /* RRNs of children              */
} PAGE;
In Pascal:
TYPE
    BTPAGE = RECORD
               KEYCOUNT: integer;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

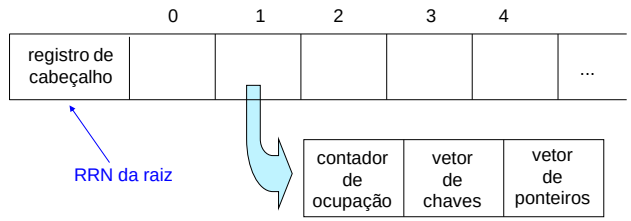
Arquivo da Árvore-B





Arquivo da Árvore-B

- Conjunto de registros de tamanho fixo



- Cada registro
  - contém uma página de disco

Algoritmos

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

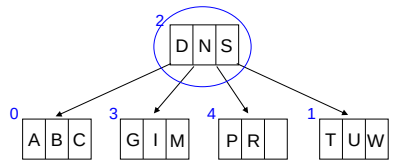
Algoritmo: Pesquisa

```
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
```

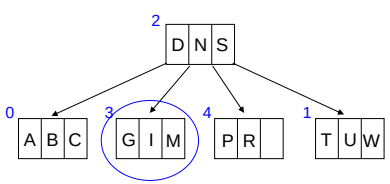

Busca da Chave K



- search (2, K, FOUND_RRN, FOUND_POS)

PAGE = D N S não existe → POS = 1

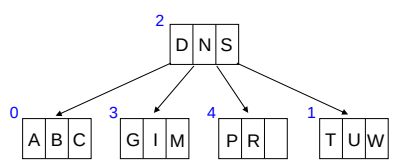
Busca da Chave K



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

PAGE = G I M não existe → POS = 2

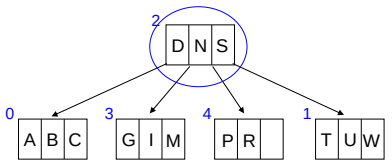
Busca da Chave K



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

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

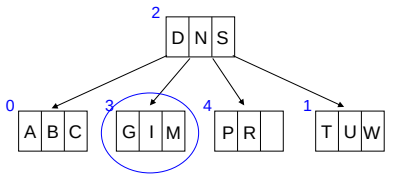
Busca da Chave M



- 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)

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

- 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

Função Insert

- Parâmetros
  - 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

- Valor 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

- 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 já ocorre ou deveria ocorrer

Função Insert

- 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

Exemplo: Inserção do $

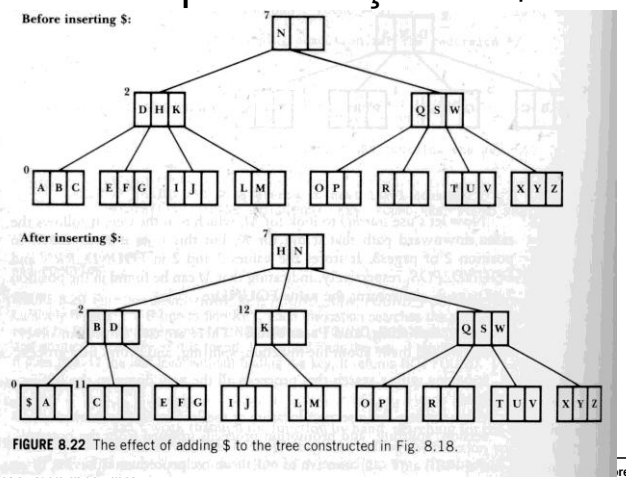


FIGURE 8.22 The effect of adding $ to the tree constructed in Fig. 8.18.

Recursão: Inserção do $

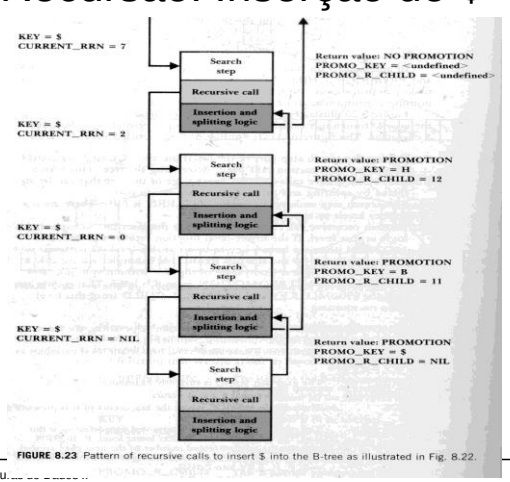
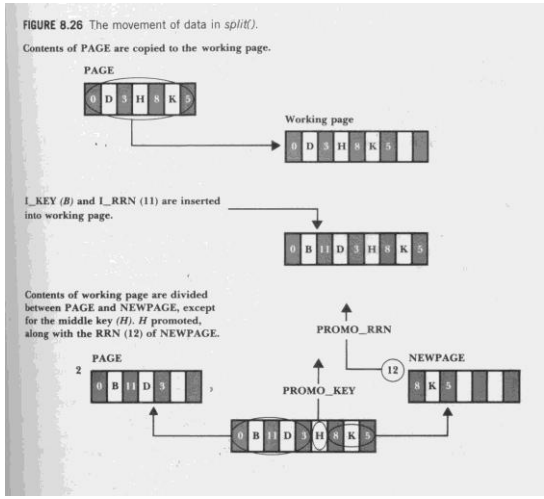


FIGURE 8.23 Pattern of recursive calls to insert $ into the B-tree as illustrated in Fig. 8.22.


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

    if CURRENT_RRN = NIL then      /* past bottom of tree */
        PROMO_KEY := KEY
        PROMO_R_CHILD := NIL
        return PROMOTION       /* promote original key and NIL */
    else
        read page at CURRENT_RRN into PAGE
        search for KEY in PAGE
        let POS := the position where KEY occurs or should occur

    if KEY found then
        issue error message indicating duplicate key
        return ERROR

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

    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 (promoted from below) in PAGE
        return NO PROMOTION
    else
        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   /* promoting PROMO_KEY and PROMO_R_CHILD */
    endif

end FUNCTION
```

FIGURE 8.24 Function *insert (CURRENT_RRN, KEY, PROMO_R_CHILD, PROMO_KEY)* inserts a KEY in a B-tree. The insertion attempt starts at the page with relative record number CURRENT_RRN. If this page is not a leaf page, the function calls itself recursively until it finds KEY in a page or reaches a leaf. If it finds KEY, it issues an error message and quits, returning ERROR. If there is space for KEY in PAGE, KEY is inserted. Otherwise, PAGE is split. A split assigns the value of the middle key to PROMO_KEY and the relative record number of the newly created page to PROMO_R-_CHILD so insertion can continue on the recursive ascent back up the tree. If a promotion does occur, *insert()* indicates this by returning PROMOTION. Otherwise, it returns NO PROMOTION.



A Função Split

- Parâmetros
  - I_KEY, I_RRN
    - nova chave a ser inserida
  - PAGE
    - página de disco corrente
  - PROMO_R_CHILD, NEWPAGE
    - parâmetros de retorno

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

```
PROCEDURE: split (I_KEY, I_RRN, PAGE, PROMO_KEY, PROMO_R_CHILD, NEWPAGE)

    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 places in the working page.

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

    set PROMO_KEY to 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 8.25 *Split (I_KEY, I_RRN, PAGE, PROMO_KEY, PROMO_R_CHILD, NEWPAGE)*, a procedure that inserts I_KEY and I_RRN, causing overflow, creates a new page called NEWPAGE, distributes the keys between the original PAGE and NEWPAGE, and determines which key and RRN to promote. The promoted key and RRN are returned via the arguments PROMO_KEY and PROMO_R_CHILD.

A Função Split

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

Note que *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
```