Engenharia de Computação

Estrutura de Dados 2

Aula 15 – Árvore B – Pesquisa e Inserção

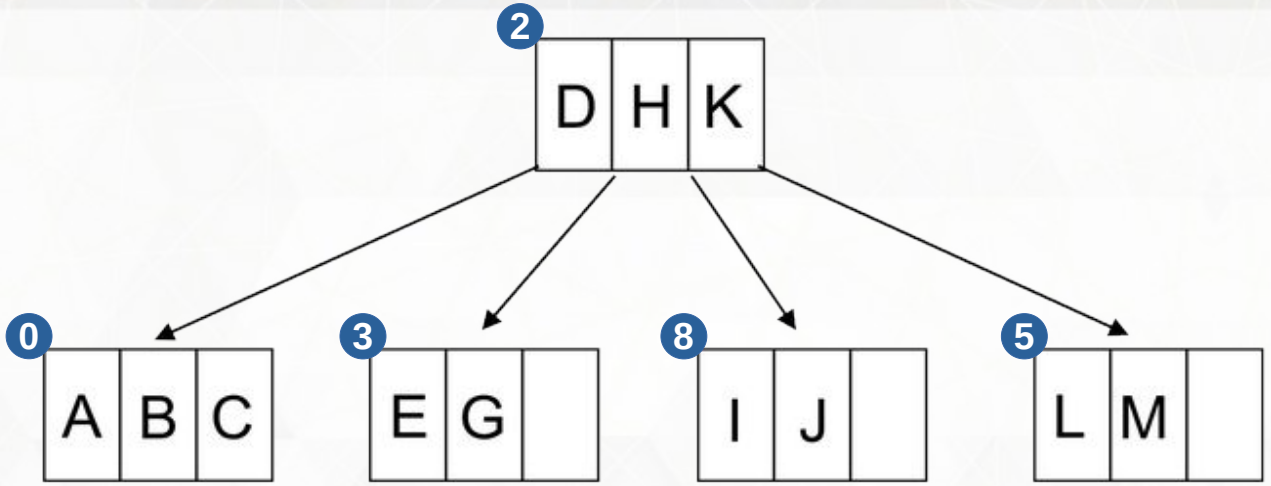
Prof. Muriel de Souza Godoi
muriel@utfpr.edu.br

Árvore B - Implementação

- 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

```
typedef struct {
    int keycount; // número de chaves armazenadas na pagina
     char key[MAXKEYS]; // as chaves atuais
     int child[MAXKEYS+1]; // RRN dos filhos ou -1((NIL)
} Page;//struct
```

Arquivo da Árvore B



Contador de Ocupação
keyCount

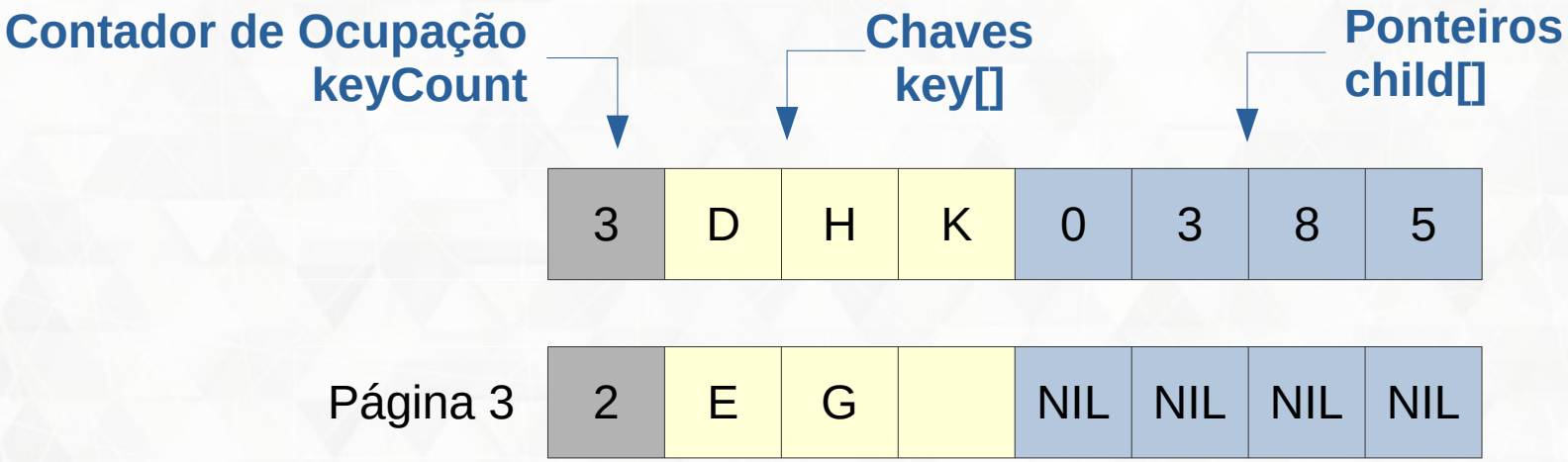
Chaves
key[]

Ponteiros
child[]

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

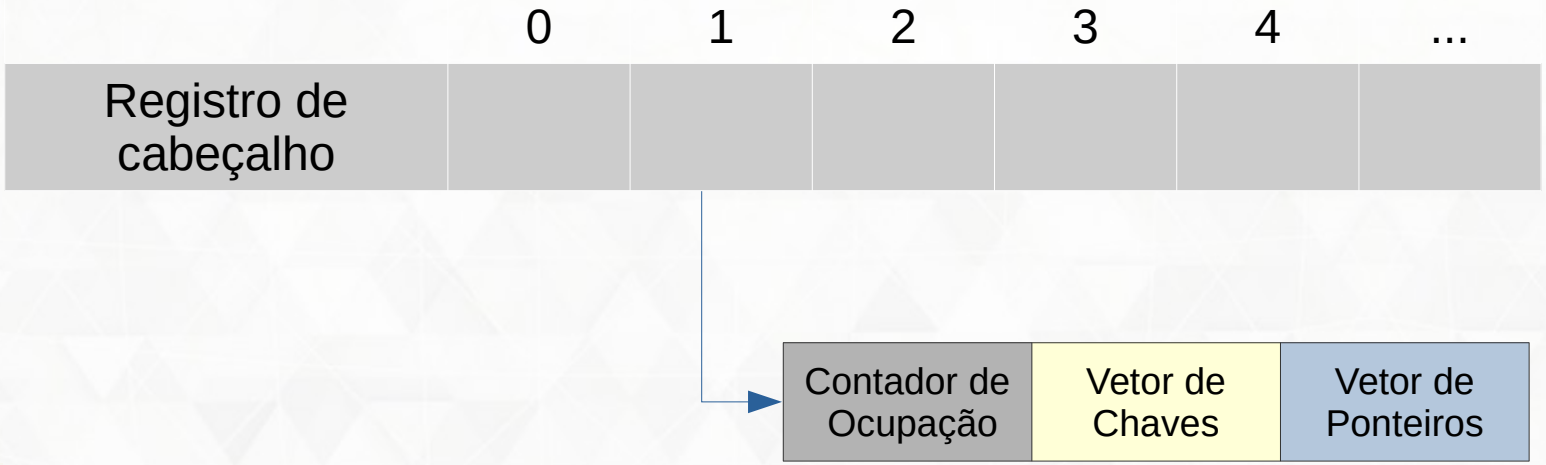
Arquivo da Árvore B

- `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 da árvore (nó)**
e pode ter o tamanho de 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

- **Argumentos**:
  - Raiz da árvore e chave procurada
- **Saída**:
  - Achou → RRN da pag. + posição da chave na pag

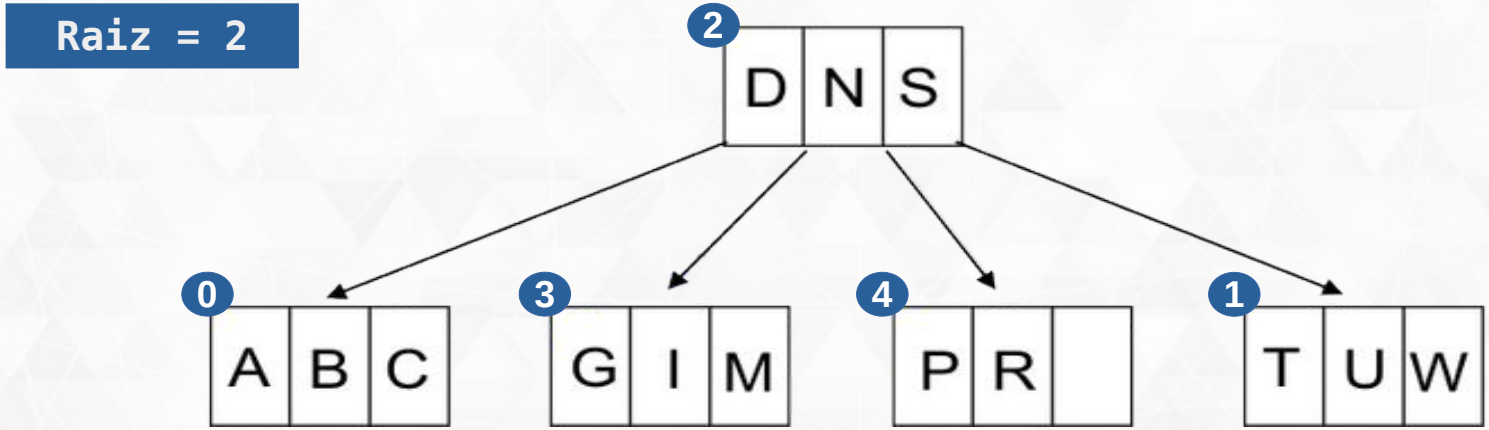
```
Se árvore vazia
  retorne Não-Achou
Senão
  leia página raiz do arquivo;
  procure chave na página;
  Se encontrou
    retorne RRN e posição da chave na pag.
  Senão
    pesquise recursivamente na subárvore apropriada da raiz
```

Algoritmo – Pesquisa – Chave K

- Considerando a seguinte assinatura:

```
bool pesquisa(RRN,          //página a ser pesquisada
              KEY,          //chave sendo procurada
              FOUND_RRN,    //página que contém a chave
              FOUND_POS);//posição da chave na página
```

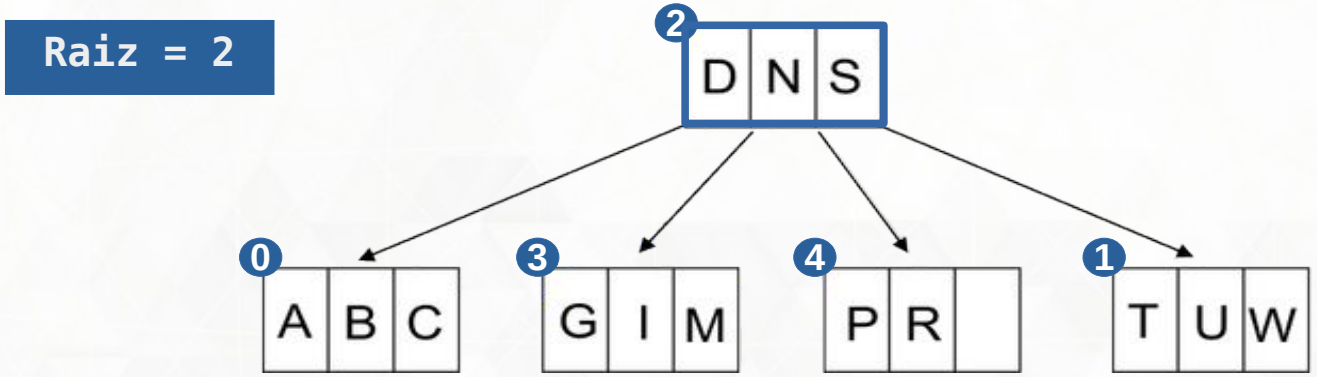
- Pesquisar a chave K na seguinte árvore



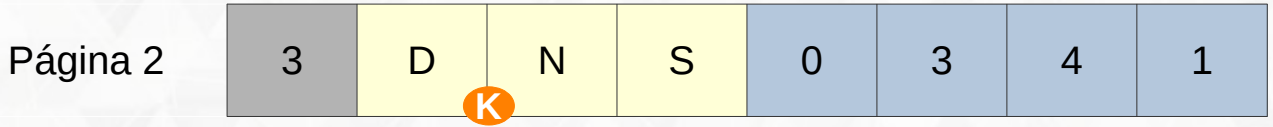
Algoritmo – Pesquisa – Chave K

- **Chamada da função**

```
pesquisa(2, K, FOUND_RRN, FOUND_POS);
```



- **Conteúdo da página**

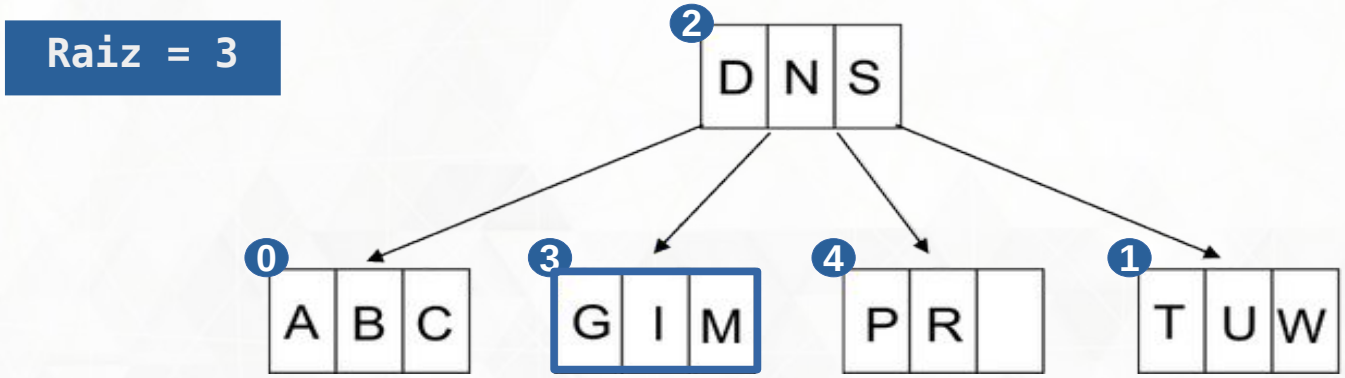


  - Não achou. Pos = 1 (PAGE.child[1])

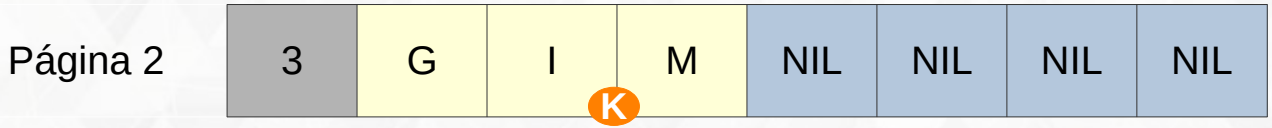
Algoritmo – Pesquisa – Chave K

- **Chamada da função**

```
pesquisa(3, K, FOUND_RRN, FOUND_POS);
```



- **Conteúdo da página**



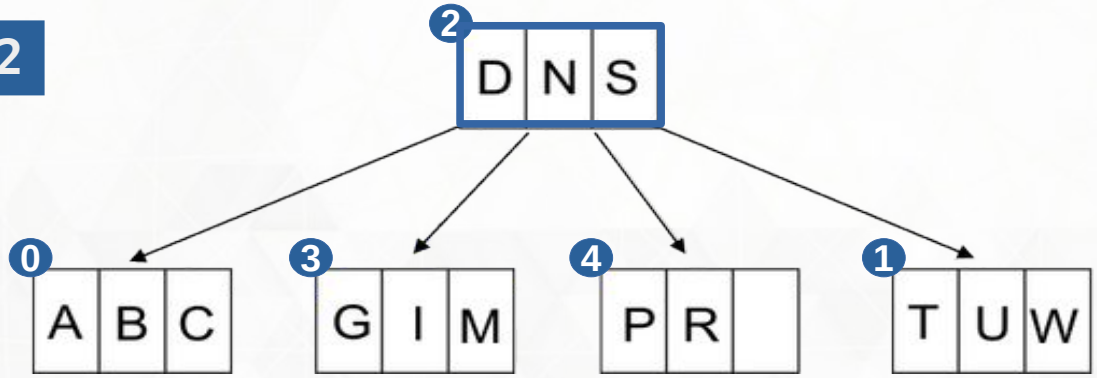
  - Não achou - Pos = 2 (PAGE.child[2])

Algoritmo – Pesquisa – Chave M

- **Nova pesquisa - Chamada da função**

```
pesquisa(2, M, FOUND_RRN, FOUND_POS);
```

Raiz = 2



- **Conteúdo da página**

Página 2

3	D	N	S	0	3	4	1

K

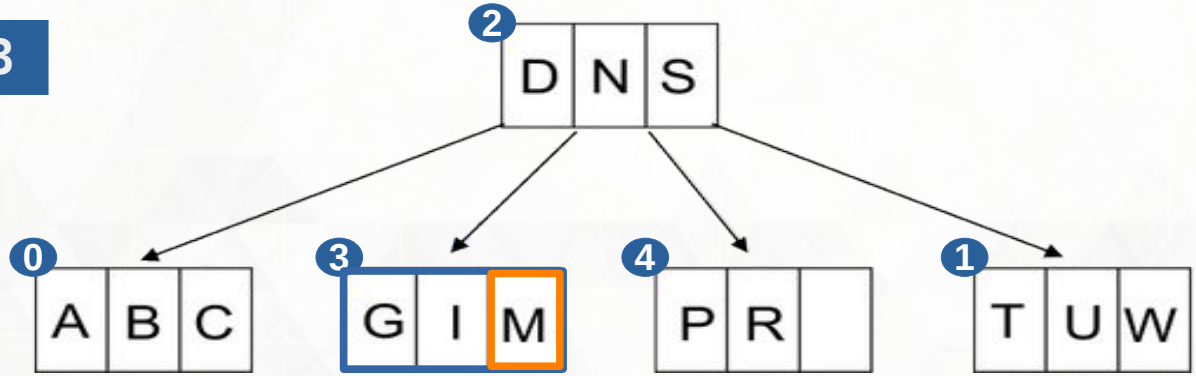
  - Não achou. Pos = 1 (PAGE.child[1])

Algoritmo – Pesquisa – Chave M

- **Chamada da função**

```
pesquisa(3, M, FOUND_RRN, FOUND_POS);
```

Raiz = 3



- **Retorna Achou**
  - FOUND_RRN = 3
  - FOUND_POS = 2

Algoritmo – Inserção

3 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, divisão e promoção** (i*nsert, split and promotion*)
  - executados após a chamada recursiva
  - a propagação destes processos ocorre no retorno da chamada recursiva

Algoritmo – Inserção

- **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 -** chave promovida, caso a inserção resulte no particionamento e na promoção da chave
  - **PROMO_R_CHILD** - RRN para o filho direito de PROMO_KEY

Quando ocorre um particionamento, deve ser retornado a **chave promovida** e o **RRN da nova página** criada

Algoritmo – Inserção

- 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

Algoritmo – Inserção

```
insere(CURRENT_RRN, KEY, PROMO_R_CHILD, PROMO_KEY);
```

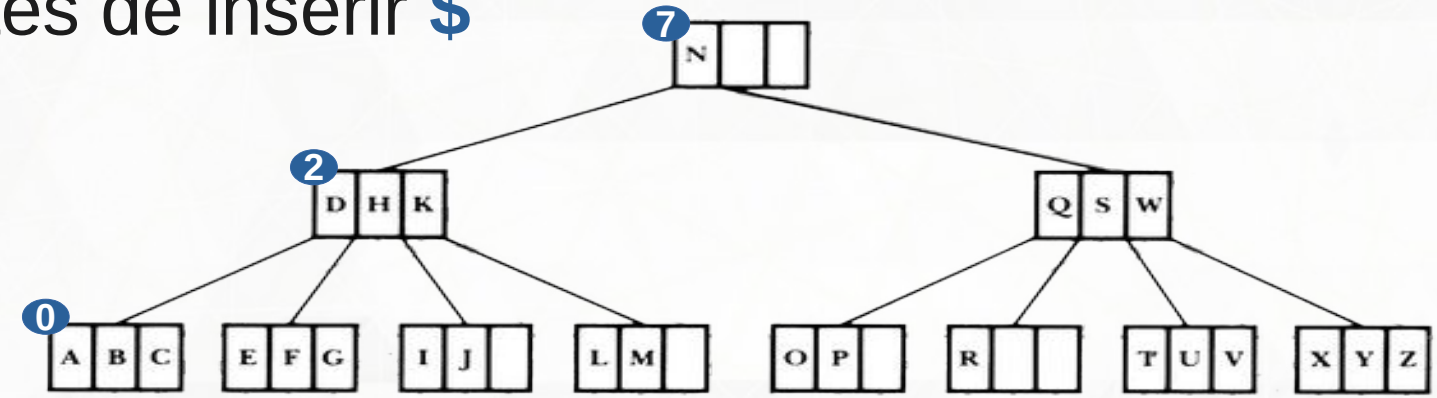
```
Se CURRENT_RRN = NIL então
  PROMO_KEY = KEY
  PROMO_R_CHILD = NIL
  retorne PROMOTION
Senão
  leia página CURRENT_RRN e armazene em PAGE procure por KEY em PAGE
  faça POS igual a posição em que KEY ocorre ou deveria ocorrer
Se KEY encontrada então
  produza mensagem de erro indicado que chave já existe
  retorne ERRO

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

Algoritmo – Inserção

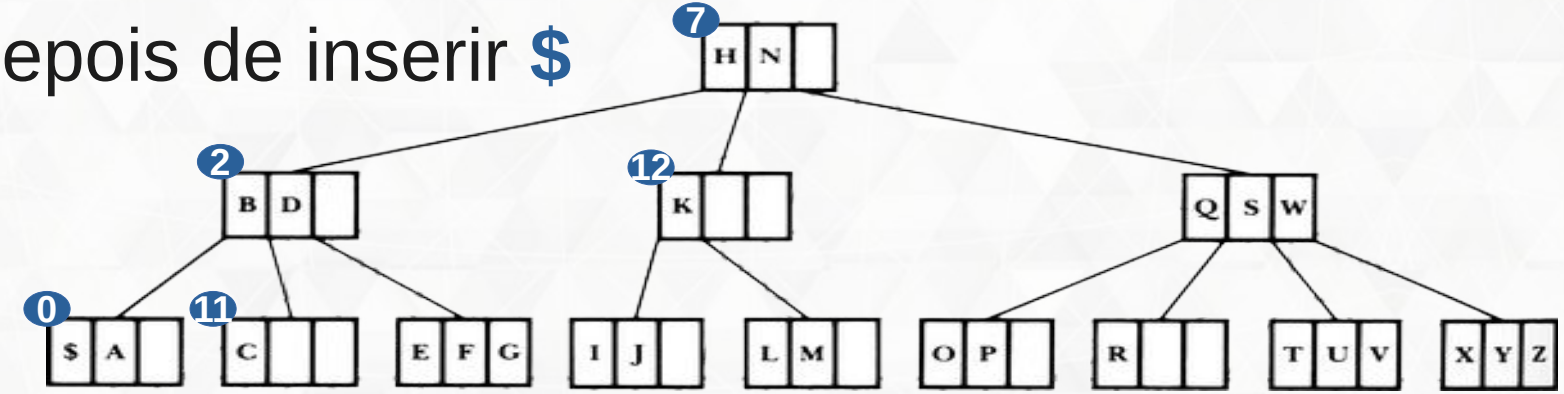
```
...

Se RETURN_VALUE = NO PROMOTION or ERROR então
  retorne RETURN_VALUE

Senão se há espaço em PAGE para P_B_KEY então
  inserir P_B_KEY e P_B_RRN em PAGE
  retorne NO PROMOTION

Senão
  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
  retorne PROMOTION
fim
```

Algoritmo – Inserção

- Antes de inserir **$**



- Depois de inserir **$**



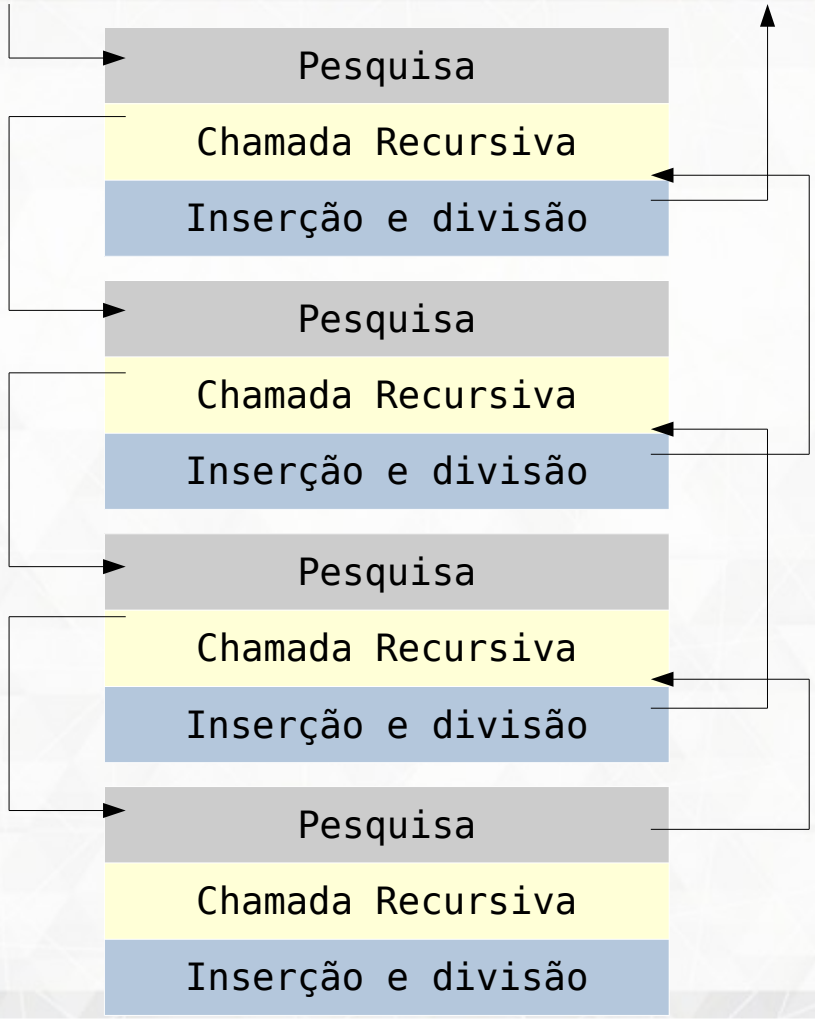
Algoritmo – Inserção

KEY = $
CURRENT_RRN = 7

KEY = $
CURRENT_RRN = 2

KEY = $
CURRENT_RRN = 0

KEY = $
CURRENT_RRN = NIL



RetuRn Value = **NO PROMOTION**
PROMO_KEY = **UNDEF**
PROMO_R_CHILD = **UNDEF**

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**

Algoritmo – Divide (Split)

```
divide(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

Algoritmo – Divide (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 – Divide (Split)

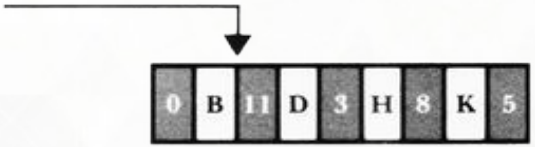
```
divide(I_KEY,I_RRN,PAGE,PROMO_KEY,PROMO_R_CHILD,NEWPAGE);
```

- copie todas as chaves e ponteiros de PAGE para uma página temporária estendida com espaço extra para uma nova chave e um novo ponteiro
- insira I_KEY e I_RRN no lugar apropriado na página temporária
- crie a nova página NEWPAGE
- PROMO_KEY = chave do meio da página temporária
- PROMO_R_CHILD = 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

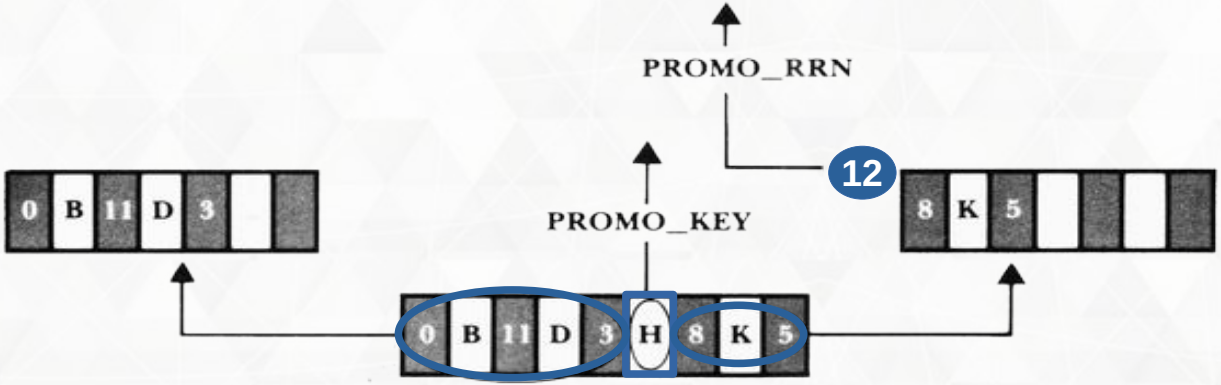
Algoritmo – Divide (Split)

1º Copia **PAGE** para **WORKING_PAGE**

2º Insere **I_KEY(B)** e **I_RRN (11)** na **WORKING_PAGE**



3º Divide o conteúdo entre **PAGE** e **NEWPAGE,** exceto a **chave H** e o **RRN 12** que são promovidos



Algoritmo – Divide (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!

Algoritmo - 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

```
Se arquivo com árvore-B existe então
  abra arquivo
Senão
  crie arquivo, leia e coloque a primeira chave na raiz
recupere RRN da página raiz e armazene em ROOT
leia uma chave e armazene em KEY
Enquanto KEY existe faça
  Se (insert(ROOT, KEY, PROMO_R_CHILD, PROMO_KEY)=PROMOTION) então
    crie nova página raiz com
      Key = PROMO_KEY, l_child = ROOT,  r_child = PROMO_R_CHILD
    ROOT = RRN da nova página raiz
  leia próxima chave e armazene em KEY
escreva no arquivo o RRN armazenado em ROOT
feche arquivo
```

Exercício

- Vamos começar a implementação da nossa Árvore B?