

Árvore-B *Busca e Inserção*

Organização e Recuperação de Dados
Profa. Valéria

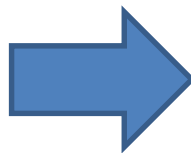
UEM – CTC – DIN

Busca e inserção em árvore-B

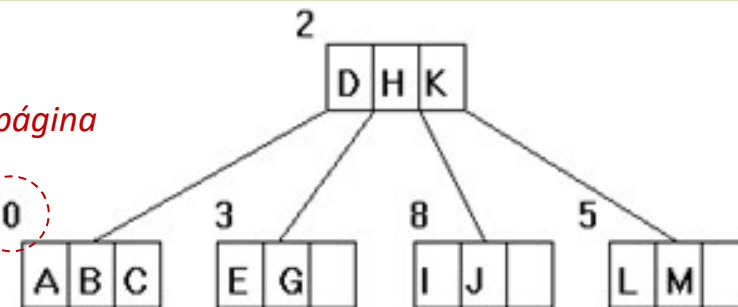
❑ Possível estrutura de página de árvore-B em C

```
typedef struct {  
    int CONTACHAVES;           /* número de chaves na página */  
    char CHAVES[MAXCHAVES];    /* vetor que armazena as chaves */  
    int FILHOS[MAXCHAVES+1];    /* RRNs dos filhos */  
} PAGINA;
```

❑ Parte de uma árvore-B de ordem 4



RRN da página



Conteúdo das páginas 2 e 3:



❑ O arquivo da árvore-B:

- Registros de tamanho fixo
- Cada registro armazena uma página da árvore-B

Pesquisa em árvore-B

□ Algoritmo de busca na árvore-B

- Recursivo
 - Descida na árvore
- Trabalha em 2 etapas:
 - Alterna entre busca de páginas e busca “dentro” da página

Pesquisa em árvore-B

→ Parâmetros de entrada

```
FUNÇÃO busca (RRN, CHAVE, RRN_ENCONTRADO, POS_ENCONTRADA)  
  se RRN == NULO então      /* condição de parada */  
    retorne NAO_ENCONTRADO  
  senão  
    leia a página armazenada no RRN para PAG  
    ENCONTRADA = busca_na_pagina(CHAVE, PAG, POS)  
    /* POS recebe a posição em que CHAVE ocorre  
       ou deveria ocorrer se estivesse em PAG */  
    se ENCONTRADA então  
      RRN_ENCONTRADO := RRN      /* RRN da página que contém a chave */  
      POS_ENCONTRADO := POS      /* posição da chave na página*/  
      retorne ENCONTRADO  
    senão      /* siga o ponteiro para a próxima página da busca */  
      retorne(busca(PAG.FILHOS[POS], CHAVE, RRN_ENCONTRADO, POS_ENCONTRADA))  
  fim se  
fim se  
fim FUNÇÃO
```

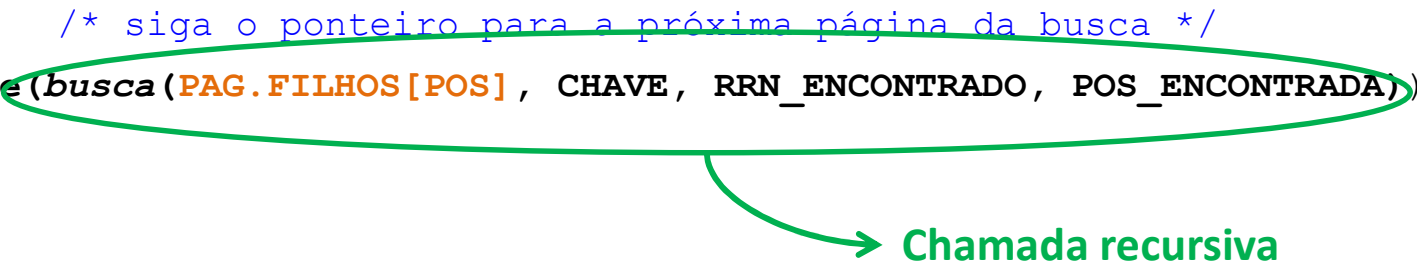
Pesquisa em árvore-B

```
FUNÇÃO busca (RRN, CHAVE, RRN_ENCONTRADO, POS_ENCONTRADA)  
  se RRN == NULO então      /* condição de parada */  
    retorne NAO_ENCONTRADO  
  senão  
    leia a página armazenada no RRN para PAG  
    ENCONTRADA = busca_na_pagina(CHAVE, PAG, POS)  
    /* POS recebe a posição em que CHAVE ocorre  
       ou deveria ocorrer se estivesse em PAG */  
    se ENCONTRADA então  
      RRN_ENCONTRADO := RRN      /* RRN da página que contém a chave */  
      POS_ENCONTRADO := POS      /* posição da chave na página*/  
      retorne ENCONTRADO  
    senão      /* siga o ponteiro para a próxima página da busca */  
      retorne(busca(PAG.FILHOS[POS], CHAVE, RRN_ENCONTRADO, POS_ENCONTRADA))  
  fim se  
fim se  
fim FUNÇÃO
```

Parâmetros de retorno

Pesquisa em árvore-B

```
FUNÇÃO busca (RRN, CHAVE, RRN_ENCONTRADO, POS_ENCONTRADA)
  se RRN == NULO então      /* condição de parada */
    retorne NAO_ENCONTRADO
  senão
    leia a página armazenada no RRN para PAG
    ENCONTRADA = busca_na_pagina(CHAVE, PAG, POS)
    /* POS recebe a posição em que CHAVE ocorre
       ou deveria ocorrer se estivesse em PAG */
    se ENCONTRADA então
      RRN_ENCONTRADO := RRN      /* RRN da página que contém a chave */
      POS_ENCONTRADO := POS      /* posição da chave na página*/
      retorne ENCONTRADO
    senão      /* siga o ponteiro para a próxima página da busca */
      retorne(busca(PAG.FILHOS[POS], CHAVE, RRN_ENCONTRADO, POS_ENCONTRADA))
  fim se
fim se
fim FUNÇÃO
```



Chamada recursiva

Pesquisa em árvore-B

```
FUNÇÃO busca (RRN, CHAVE, RRN_ENCONTRADO, POS_ENCONTRADA)
  se RRN == NULO então      /* condição de parada */
    retorne NAO_ENCONTRADO
  senão
    leia a página armazenada no RRN para PAG
    ENCONTRADA = busca_na_pagina(CHAVE, PAG, POS)
    /* POS recebe a posição em que CHAVE ocorre
       ou deveria ocorrer se estivesse em PAG */
    se ENCONTRADA então
      RRN_ENCONTRADO := RRN      /* RRN da página que contém a chave */
      POS_ENCONTRADO := POS      /* posição da chave na página*/
      retorne ENCONTRADO
    senão      /* siga o ponteiro para a próxima página da busca */
      retorne(busca(PAG.FILHOS[POS], CHAVE, RRN_ENCONTRADO, POS_ENCONTRADA))
  fim se
fim se
fim FUNÇÃO
```

Pesquisa em árvore-B

→ Parâmetros de entrada

```
FUNÇÃO busca_na_pagina(CHAVE, PAG, POS) {  
    faça i receber 0  
    enquanto i < PAG.CONTACHAVES e CHAVE > PAG.CHAVE[i] faça  
        incremente i  
    faça POS receber i  
    se POS < PAG.CONTACHAVES e CHAVE é igual a PAG.CHAVE[POS] então  
        retorne ENCONTRADO  
    senão  
        retorne NAO_ENCONTRADO  
}
```


Pesquisa em árvore-B

FUNÇÃO *busca_na_pagina*(CHAVE, PAG, POS) {
 faça i receber 0
 enquanto i < PAG.CONTACHAVES **e** CHAVE > PAG.CHAVE[i] **faça**
 incremente i
 faça POS receber i
 se POS < PAG.CONTACHAVES **e** CHAVE é igual a PAG.CHAVE[POS] **então**
 retorne ENCONTRADO
 senão
 retorne NAO_ENCONTRADO
}

Parâmetro de retorno

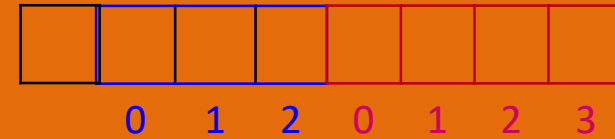
Pesquisa em árvore-B

```
FUNÇÃO busca_na_pagina(CHAVE, PAG, POS) {  
    faça i receber 0  
    enquanto i < PAG.CONTACHAVES e CHAVE > PAG.CHAVE[i] faça  
        incremente i  
    faça POS receber i  
    se POS < PAG.CONTACHAVES e CHAVE é igual a PAG.CHAVE[POS] então  
        retorne ENCONTRADO  
    senão  
        retorne NAO_ENCONTRADO  
}
```

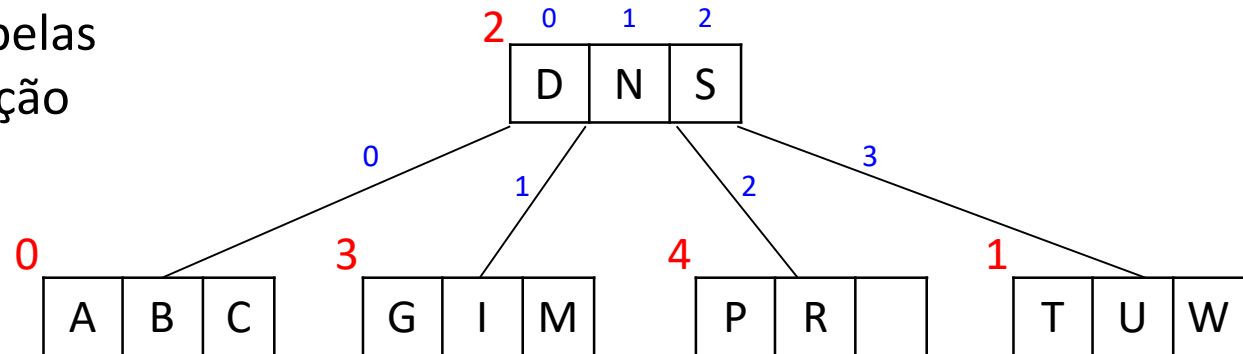
Pesquisa em árvore-B

- ❑ A **busca recursiva** se inicia na página raiz da árvore-B
- ❑ Para cada página lida, busca-se internamente pela chave (***busca_na_pagina***)
 - Se a chave for encontrada:
 - A função retorna ENCONTRADA
 - O RRN da página na qual a chave foi encontrada é retornado em **RRN_ENCONTRADO**
 - A posição chave no vetor de chaves da página volta em **POS_ENCONTRADA**
 - Se a chave não for encontrada, a busca prossegue até encontrar um ponteiro NULO em uma das folhas, retornando NAO_ENCONTRADO
- ❑ Na primeira chamada da função **busca**, o RRN da página raiz é passado como parâmetro

Pesquisa em árvore-B



Exercício: Simule a busca pelas
chaves **K** e **P** usando a função
search



```
FUNÇÃO busca (RRN, CHAVE, RRN_ENCONTRADO, POS_ENCONTRADA)
se RRN == NULO então
    retorne NAO_ENCONTRADO
senão
    leia a página armazenada no RRN para PAG
    ENCONTRADA = busca_na_pagina(CHAVE, PAG, POS)
    se ENCONTRADA então
        RRN_ENCONTRADO := RRN
        POS_ENCONTRADO := POS
        retorne ENCONTRADO
    senão
        retorne (busca (PAG.FILHOS[POS], CHAVE, RRN_ENCONTRADO, POS_ENCONTRADA))
fim se
fim se
fim FUNÇÃO
```

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Inserção, divisão e promoção

❑ Algoritmo de inserção em árvore-B

- Começa com uma busca
 - Inicia a busca pela raiz e continua até alcançar uma folha
- Uma vez localizada a posição de inserção (**SEMPRE em uma folha**), pode ser necessário realizar divisão e promoção, sempre de baixo para cima
- O algoritmo pode então ser pensado em 3 partes:
 1. Busca pela chave na página atual, como em **busca**, antes da chamada recursiva
 2. Chamada recursiva para fazer a “descida” na árvore (até encontrar um ponteiro nulo, que estará em uma folha)
 3. Inserção, divisão e promoção (se necessário) executadas no retorno da chamada recursiva, fazendo com que esses processos ocorram na “subida” da árvore

Inserção, divisão e promoção

❑ FUNÇÃO *insere* (RRN_ATUAL, CHAVE, FILHO_D_PRO, CHAVE_PRO)

– Argumentos:

1. **RRN_ATUAL**: contém o RRN da página que está atualmente em uso (inicialmente, a raiz)
2. **CHAVE**: contém a chave a ser inserida
3. **CHAVE_PRO**: usado para armazenar um valor de retorno.
Se a inserção da chave resultar em divisão e promoção, CHAVE_PRO conterá a chave promovida
4. **FILHO_D_PRO**: usado para armazenar um valor de retorno.
Se houver uma divisão, os níveis superiores da sequência de chamadas devem inserir não apenas a chave promovida, mas também o RRN da nova página criada na divisão
 - Quando houver uma CHAVE_PRO, FILHO_D_PRO conterá o ponteiro para o seu filho direito (que corresponde a nova página resultante da divisão)

Inserção, divisão e promoção

❑ Valores de retorno da função *insere*

1. PROMOCAO, se uma chave está sendo promovida
2. SEM_PROMOCAO, se a inserção foi feita sem necessidade de dividir a página
3. ERRO, se a inserção não puder ser realizada (chave duplicada)

❑ Variáveis locais importantes da função *insere*:

- PAG: página que está sendo examinada
- NOVAPAG: nova página que é criada caso ocorra uma divisão
- POS: posição da chave em PAG, se ela estiver lá; caso contrário, a posição em que deve ser inserida (ou a posição do ponteiro para a próxima página)
- RRN_PRO: recebe o valor do RRN da página promovida para o nível corrente (via FILHO_D_PRO)
 - Se uma divisão ocorre no nível imediatamente inferior, RRN_PRO contém o RRN da nova página criada durante a divisão. RRN_PRO é o filho direito que deve ser inserido junto com CHV_PRO em PAG
- CHV_PRO: recebe o valor da chave promovida para o nível corrente (via CHAVE_PRO)

Inserção, divisão e promoção

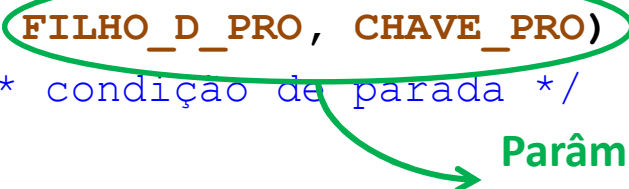
```
FUNÇÃO insere(RRN_ATUAL, CHAVE, FILHO_D_PRO, CHAVE_PRO)  
  se RRN_ATUAL == NULO então /* condição de parada */  
    CHAVE_PRO := CHAVE  
    FILHO_D_PRO := NULO  
    retorne PROMOCAO  
  senão  
    leia a página armazenada em RRN_ATUAL para PAG  
    ENCONTRADA = busca_na_pagina(CHAVE, PAG, POS)  
  fim se  
  se ENCONTRADA então  
    imprima mensagem de chave duplicada  
    retorne ERRO  
  fim se  
  RETORNO := insere(PAG.FILHOS[POS], CHAVE, RRN_PRO, CHV_PRO)  
  ...  
  /*continua no próximo slide*/
```

Parâmetros de entrada

Parâmetros de entrada

Inserção, divisão e promoção

```
FUNÇÃO insere(RRN_ATUAL, CHAVE, FILHO_D_PRO, CHAVE_PRO)  
  se RRN_ATUAL == NULO então /* condição de parada */  
    CHAVE_PRO := CHAVE  
    FILHO_D_PRO := NULO  
    retorne PROMOCAO
```



Parâmetros de retorno

senão

leia a página armazenada em RRN_ATUAL para PAG

ENCONTRADA = **busca_na_pagina**(CHAVE, PAG, **POS**)

fim se

se ENCONTRADA então

imprima mensagem de chave duplicada

retorne **ERRO**

fim se

RETORNO := **insere**(PAG.FILHOS[POS], CHAVE, **RRN_PRO, CHV_PRO**)



Variáveis que recebem os parâmetros de retorno

...

/*continua no próximo slide*/

Inserção, divisão e promoção

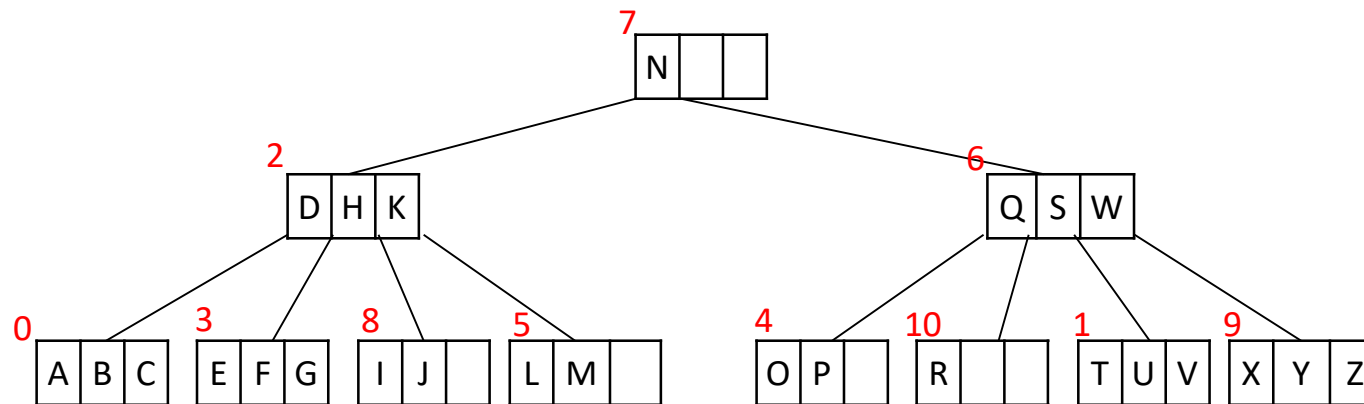
```
FUNÇÃO insere(RRN_ATUAL, CHAVE, FILHO_D_PRO, CHAVE_PRO)
  se RRN_ATUAL == NULO então /* condição de parada */
    CHAVE_PRO := CHAVE
    FILHO_D_PRO := NULO
    retorne PROMOCAO
  senão
    leia a página armazenada em RRN_ATUAL para PAG
    ENCONTRADA = busca_na_pagina(CHAVE, PAG, POS)
  fim se
  se ENCONTRADA então
    imprima mensagem de chave duplicada
    retorne ERRO
  fim se
RETORNO:= insere(PAG.FILHOS[POS], CHAVE, RRN_PRO, CHV_PRO)
...
/*continua no próximo slide*/
```

Inserção, divisão e promoção

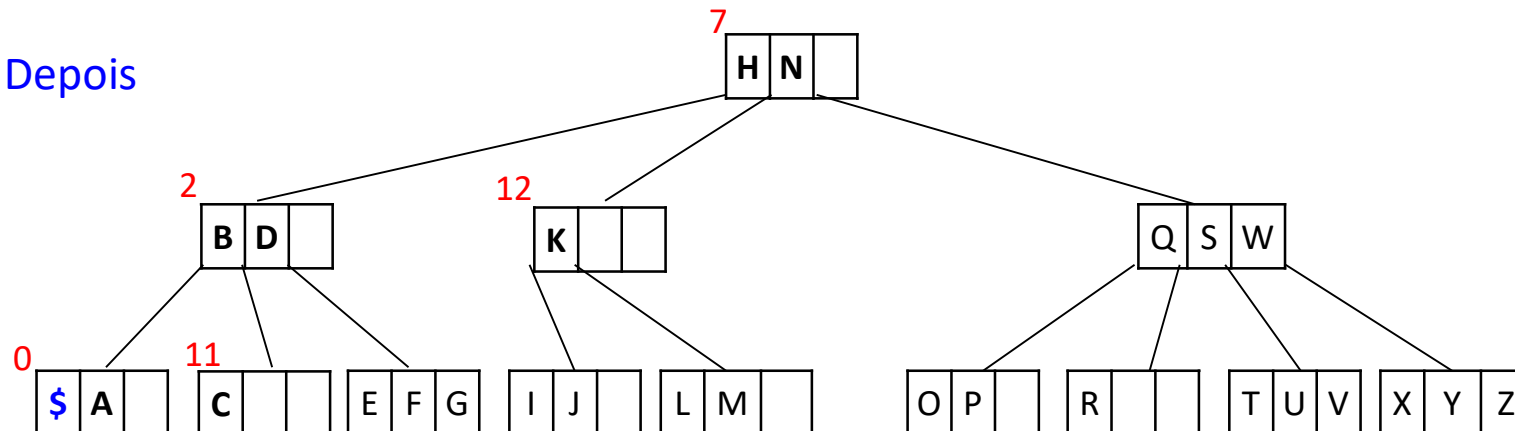
```
/*continuação da função insere, logo após a chamada recursiva*/  
...  
se RETORNO == SEM_PROMOCAO ou ERRO então  
    retorne RETORNO  
senão  
    se existe espaço em PAG para inserir CHV_PRO então  
        insira CHV_PRO e RRN_PRO (chave promovida e filha) em PAG  
        escreva PAG no arquivo em RRN_ATUAL  
        retorne SEM_PROMOCAO  
    senão  
        divide(CHV_PRO, RRN_PRO, PAG, CHAVE_PRO, FILHO_D_PRO, NOVAPAG)  
        escreva PAG no arquivo em RRN_ATUAL  
        escreva NOVAPAG no arquivo em FILHO_D_PRO  
        retorne PROMOCAO  
    fim se  
fim se  
fim FUNÇÃO
```

Inserção, divisão e promoção

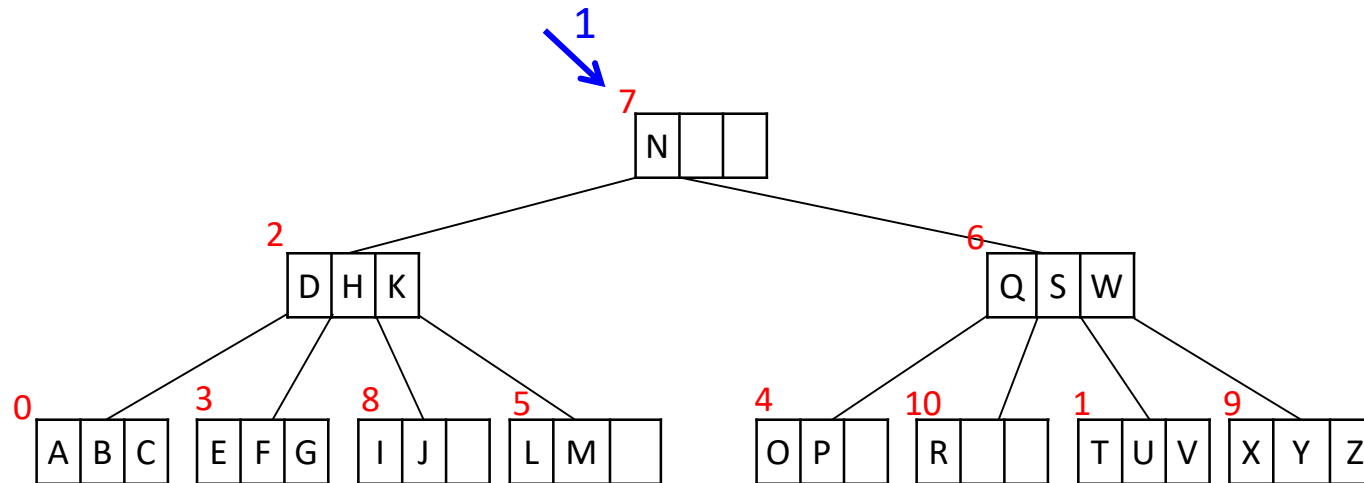
❑ Exemplo: inserção do caractere **\$** na árvore-B abaixo



Depois

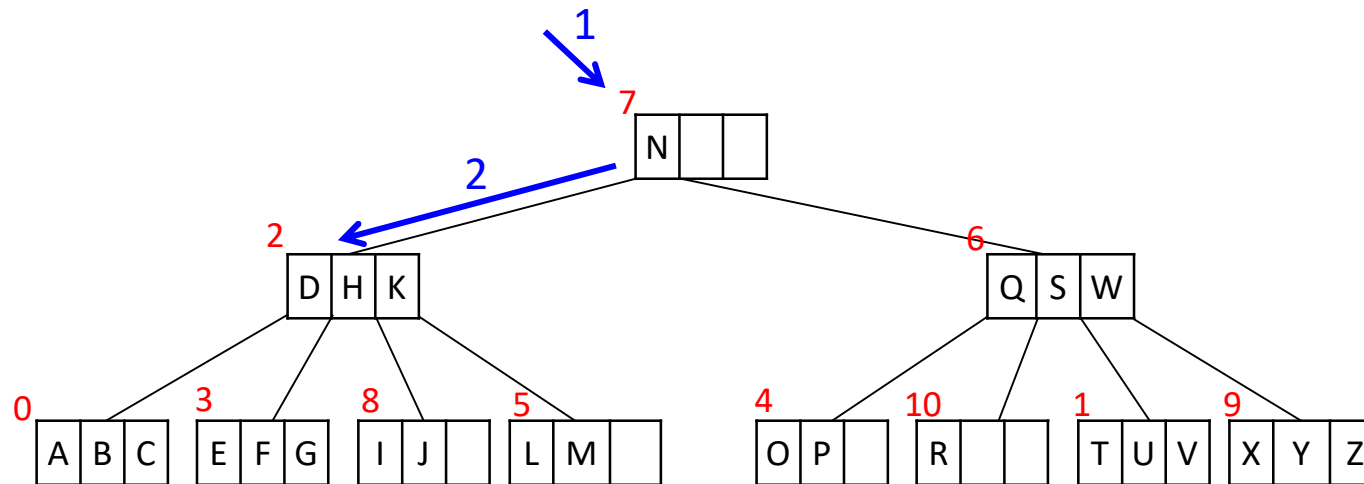


Inserção, divisão e promoção



1: insere(RRN_ATUAL=7, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)

Inserção, divisão e promoção

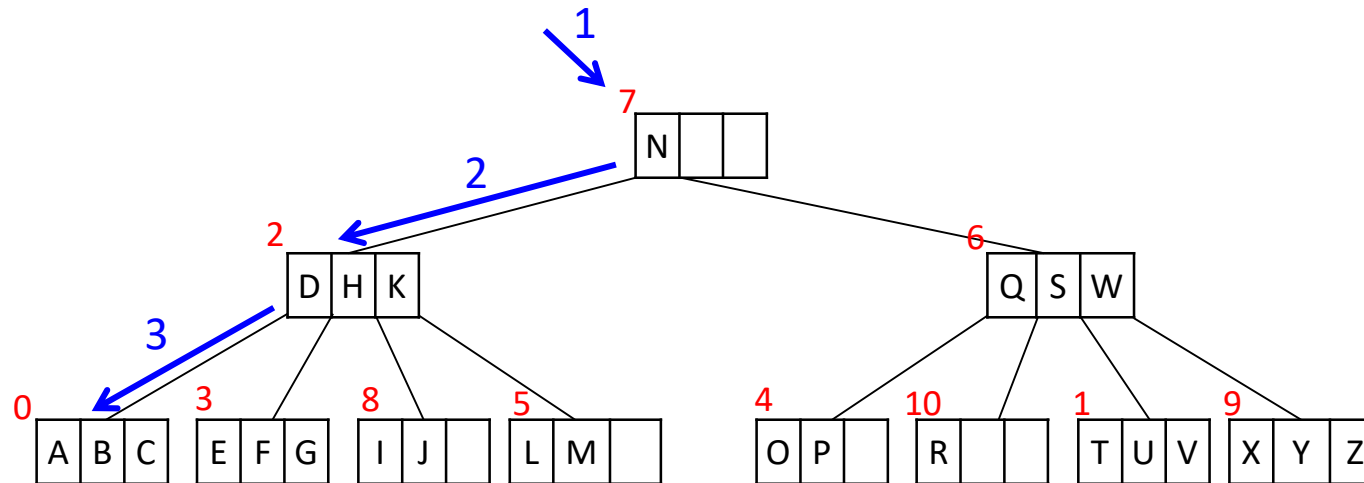


1: insere(RRN_ATUAL=7, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



2: insere(RRN_ATUAL=2, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)

Inserção, divisão e promoção



1: insere(RRN_ATUAL=7, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)

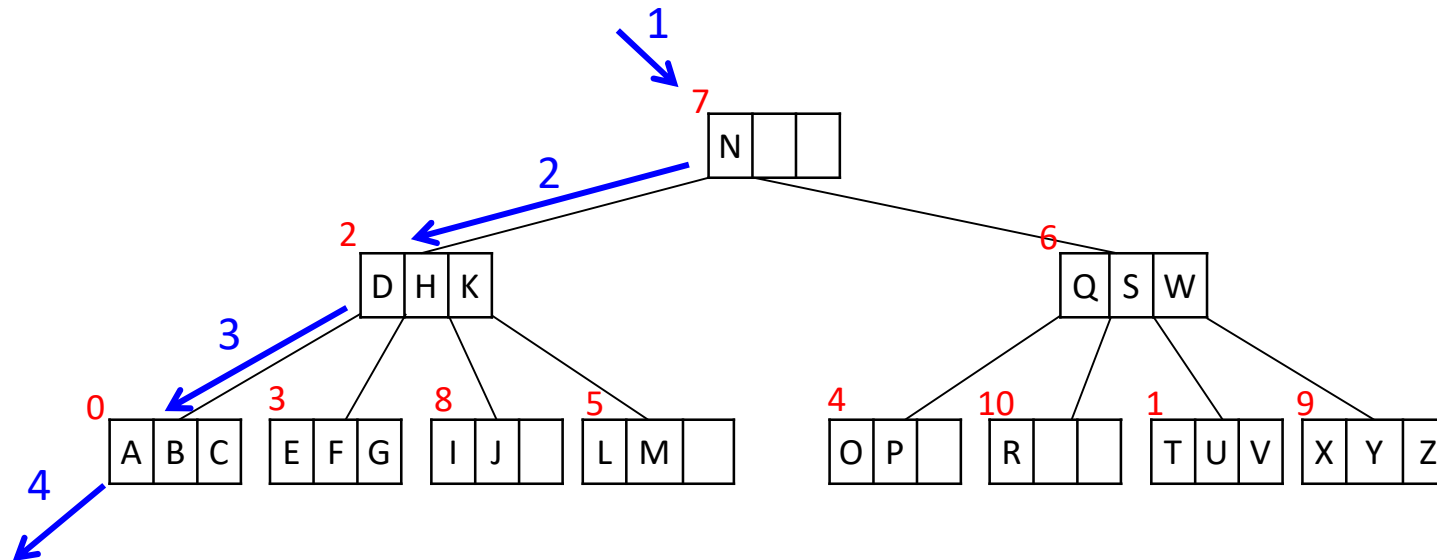


2: insere(RRN_ATUAL=2, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



3: insere(RRN_ATUAL=0, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)

Inserção, divisão e promoção



1: insere(RRN_ATUAL=7, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



2: insere(RRN_ATUAL=2, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)

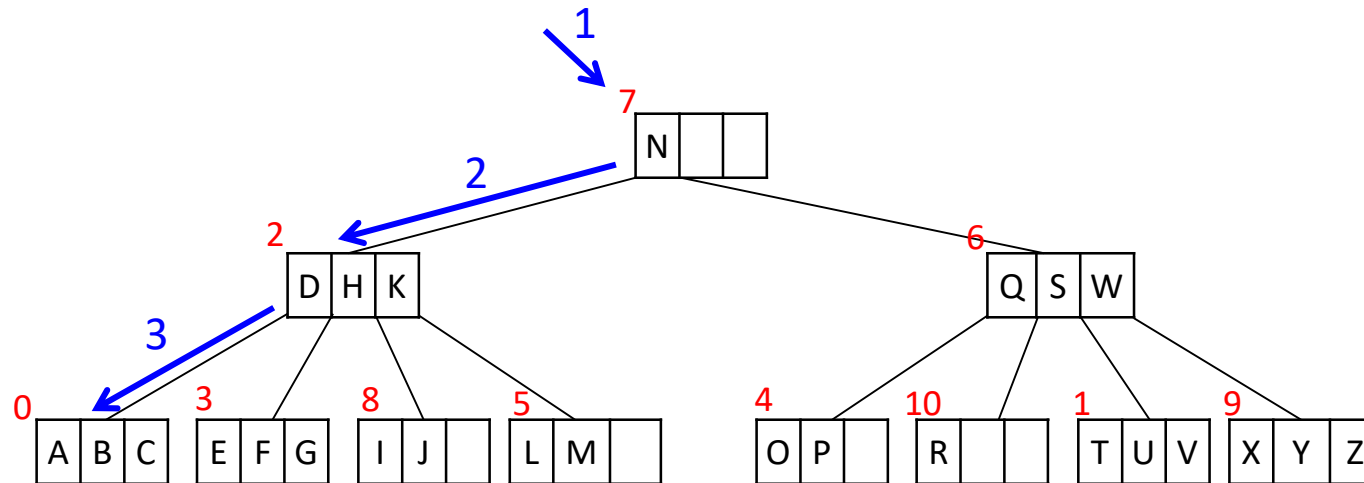


3: insere(RRN_ATUAL=0, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



4: insere(RRN_ATUAL=NULO, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)

Inserção, divisão e promoção



1: insere(RRN_ATUAL=7, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



2: insere(RRN_ATUAL=2, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



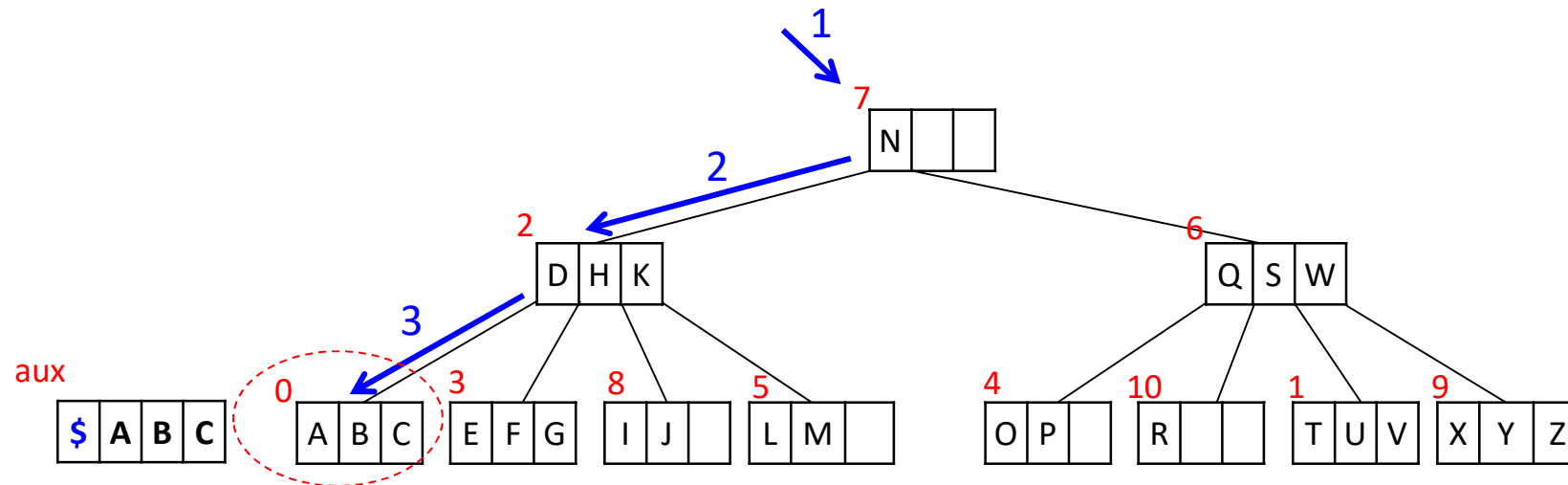
3: insere(RRN_ATUAL=0, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



4: insere(RRN_ATUAL=NULO, CHAVE=\$, FILHO_D_PRO=NULO, CHAVE_PRO=\$)

↑ PROMOÇÃO

Inserção, divisão e promoção



1: insere(RRN_ATUAL=7, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



2: insere(RRN_ATUAL=2, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



3: insere(RRN_ATUAL=0, CHAVE=\$, FILHO_D_PRO=11, CHAVE_PRO=B)

Divisão + Promoção

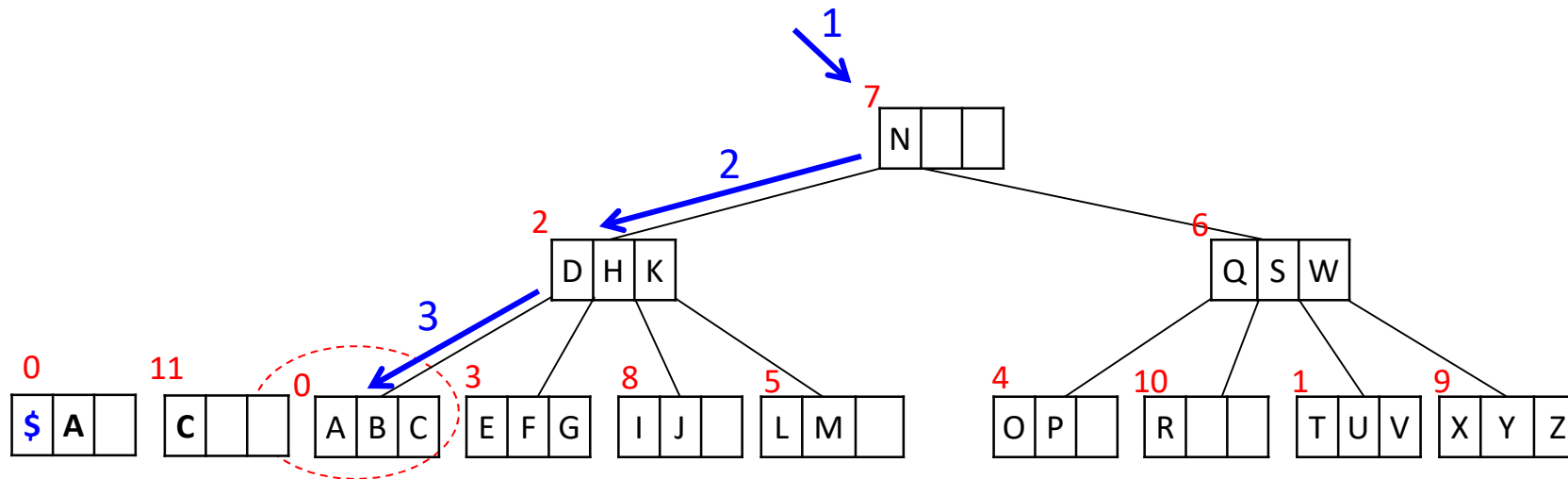


4: insere(RRN_ATUAL=NULL, CHAVE=\$, FILHO_D_PRO=NULL, CHAVE_PRO=\$)



PROMOCAO

Inserção, divisão e promoção



1: insere(RRN_ATUAL=7, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



2: insere(RRN_ATUAL=2, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



3: insere(RRN_ATUAL=0, CHAVE=\$, FILHO_D_PRO=11, CHAVE_PRO=B)

Divisão + Promoção

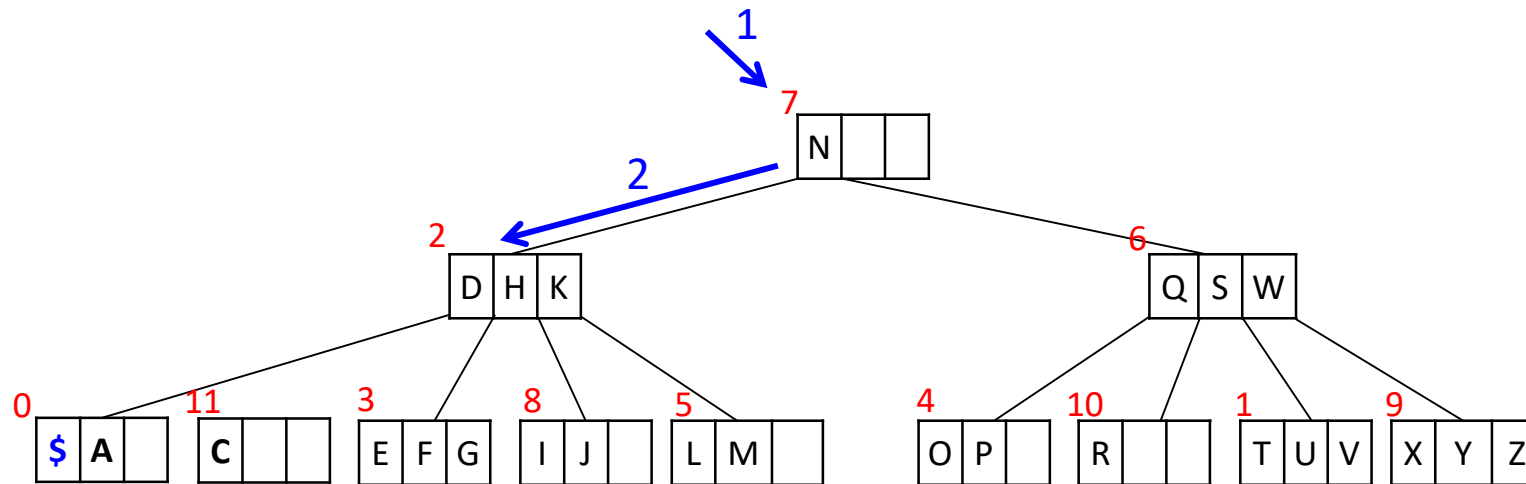


4: insere(RRN_ATUAL=NULLO, CHAVE=\$, FILHO_D_PRO=NULLO, CHAVE_PRO=\$)



PROMOCAO

Inserção, divisão e promoção



1: insere(RRN_ATUAL=7, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



2: insere(RRN_ATUAL=2, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



3: insere(RRN_ATUAL=0, CHAVE=\$, FILHO_D_PRO=11, CHAVE_PRO=B)

Divisão + Promoção



4: insere(RRN_ATUAL=NULLO, CHAVE=\$, FILHO_D_PRO=NULLO, CHAVE_PRO=\$)

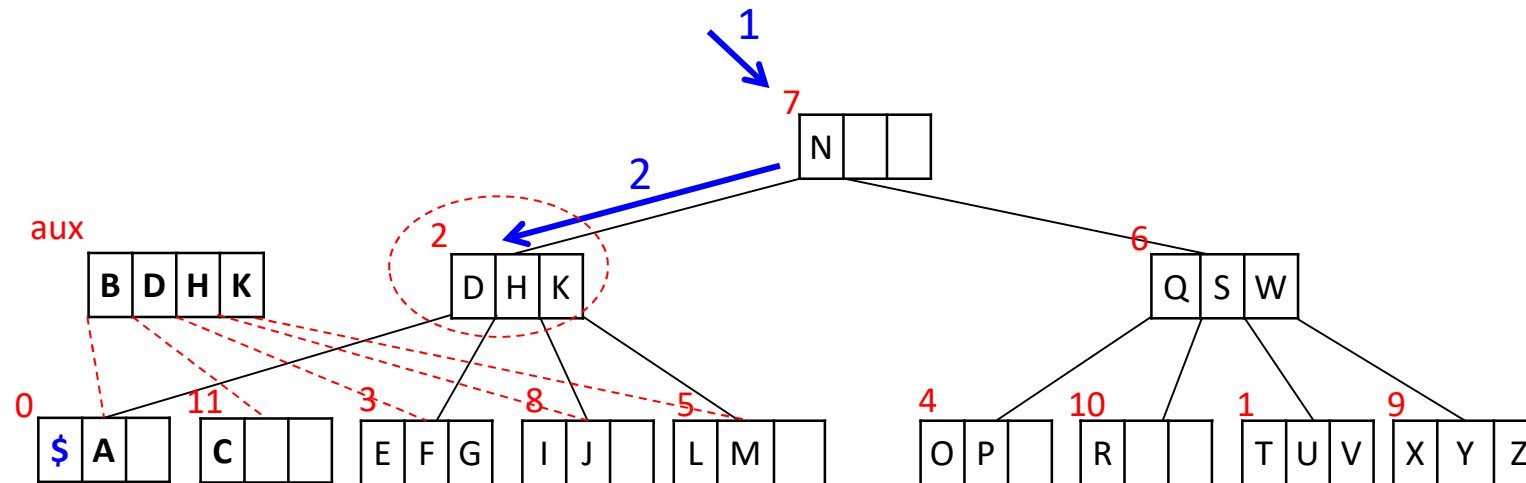


PROMOCAO



PROMOCAO

Inserção, divisão e promoção



1: insere(RRN_ATUAL=7, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



2: insere(RRN_ATUAL=2, CHAVE=\$, FILHO_D_PRO=12, CHAVE_PRO=H)

Divisão + Promoção

↑
PROMOCAO



3: insere(RRN_ATUAL=0, CHAVE=\$, FILHO_D_PRO=11, CHAVE_PRO=B)

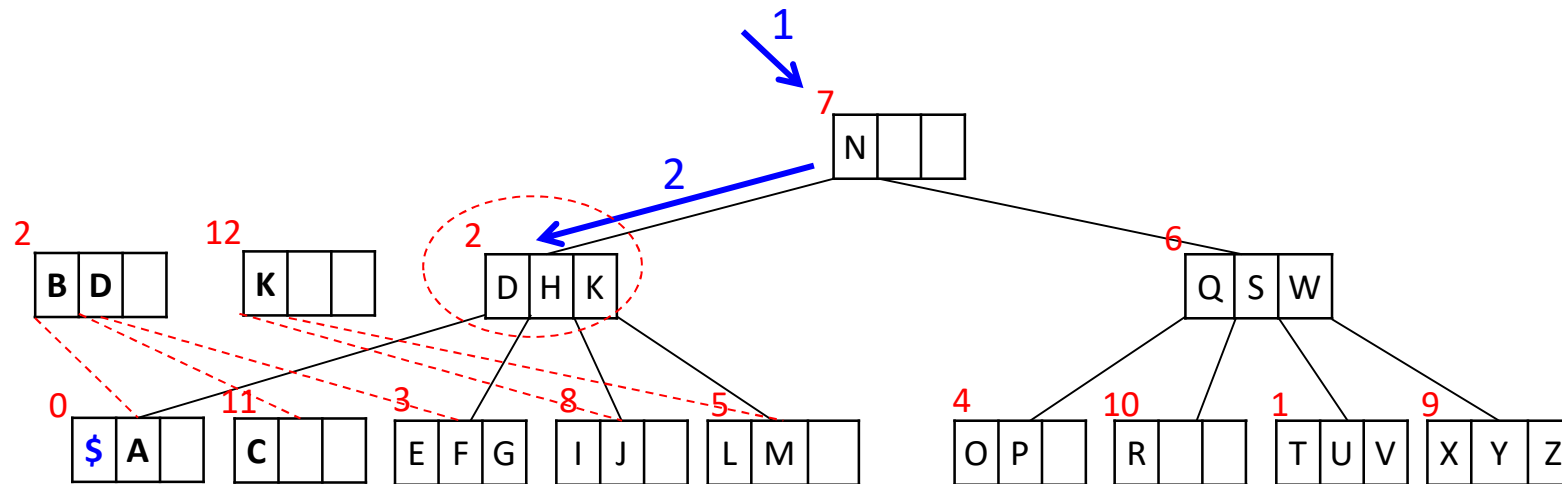
Divisão + Promoção

↑
PROMOCAO



4: insere(RRN_ATUAL=NULO, CHAVE=\$, FILHO_D_PRO=NULO, CHAVE_PRO=\$)

Inserção, divisão e promoção



1: insere(RRN_ATUAL=7, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



2: insere(RRN_ATUAL=2, CHAVE=\$, FILHO_D_PRO=12, CHAVE_PRO=H)

Divisão + Promoção

↑
PROMOCAO



3: insere(RRN_ATUAL=0, CHAVE=\$, FILHO_D_PRO=11, CHAVE_PRO=B)

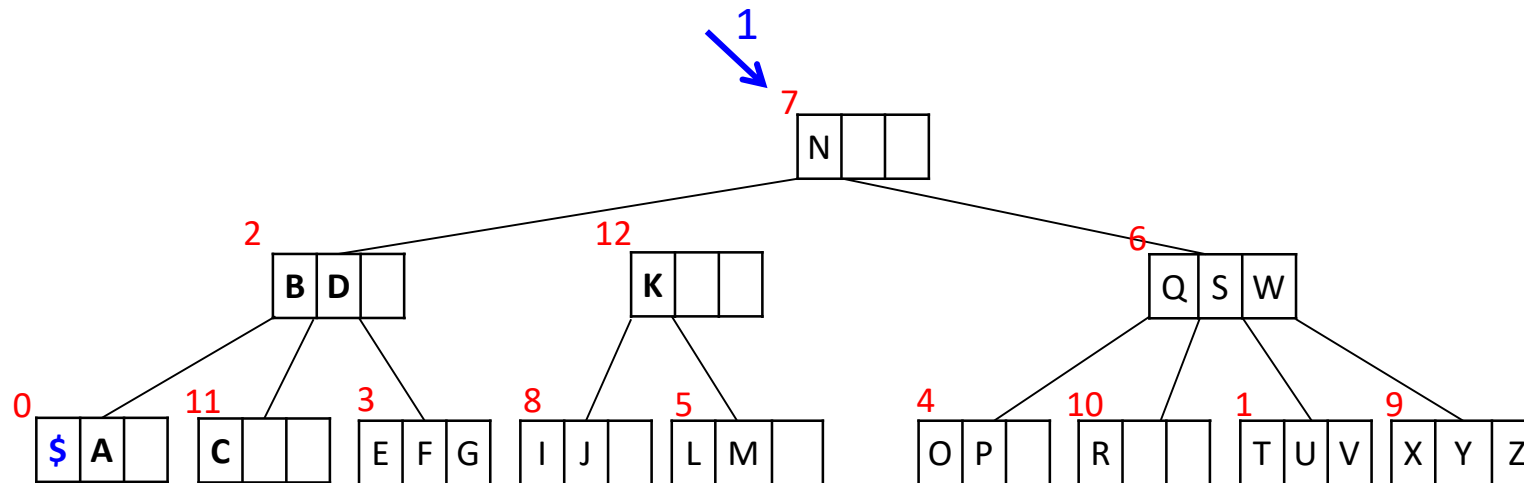
Divisão + Promoção

↑
PROMOCAO



4: insere(RRN_ATUAL=NULLO, CHAVE=\$, FILHO_D_PRO=NULLO, CHAVE_PRO=\$)

Inserção, divisão e promoção



1: insere(RRN_ATUAL=7, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)



2: insere(RRN_ATUAL=2, CHAVE=\$, FILHO_D_PRO=12, CHAVE_PRO=H)

Divisão + Promoção



3: insere(RRN_ATUAL=0, CHAVE=\$, FILHO_D_PRO=11, CHAVE_PRO=B)

Divisão + Promoção



4: insere(RRN_ATUAL=NULO, CHAVE=\$, FILHO_D_PRO=NULO, CHAVE_PRO=\$)



PROMOCAO

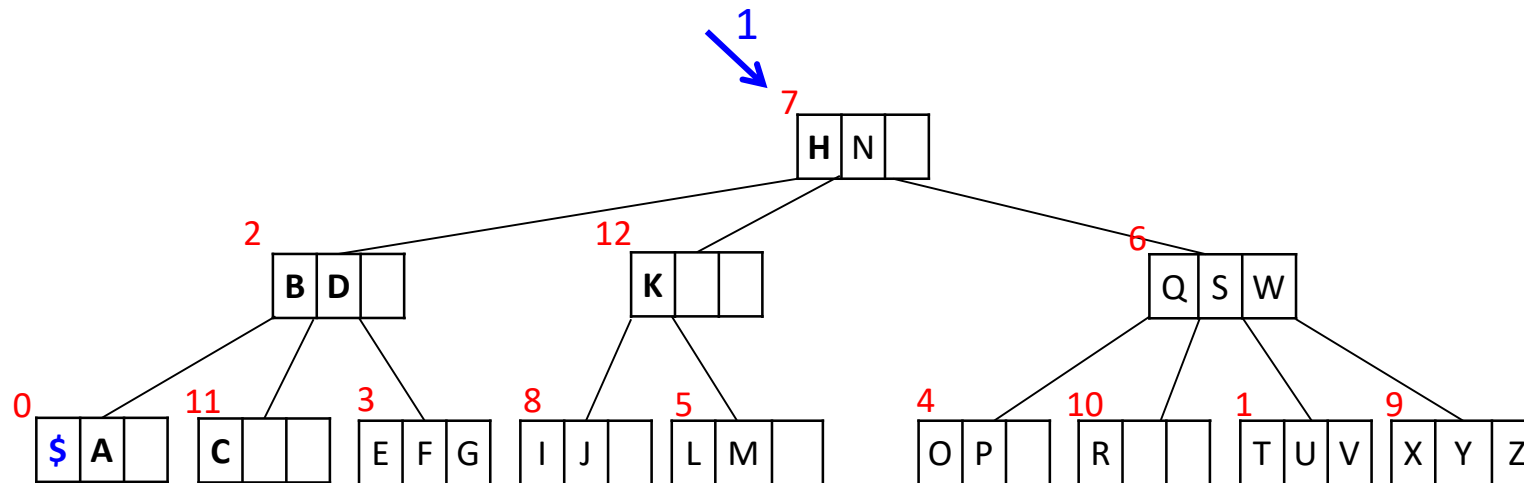


PROMOCAO



PROMOCAO

Inserção, divisão e promoção



1: insere(RRN_ATUAL=7, CHAVE=\$, FILHO_D_PRO=?, CHAVE_PRO=?)

Inserção Simples



2: insere(RRN_ATUAL=2, CHAVE=\$, FILHO_D_PRO=12, CHAVE_PRO=H)

Divisão + Promoção



3: insere(RRN_ATUAL=0, CHAVE=\$, FILHO_D_PRO=11, CHAVE_PRO=B)

Divisão + Promoção



4: insere(RRN_ATUAL=NULO, CHAVE=\$, FILHO_D_PRO=NULO, CHAVE_PRO=\$)



PROMOCAO

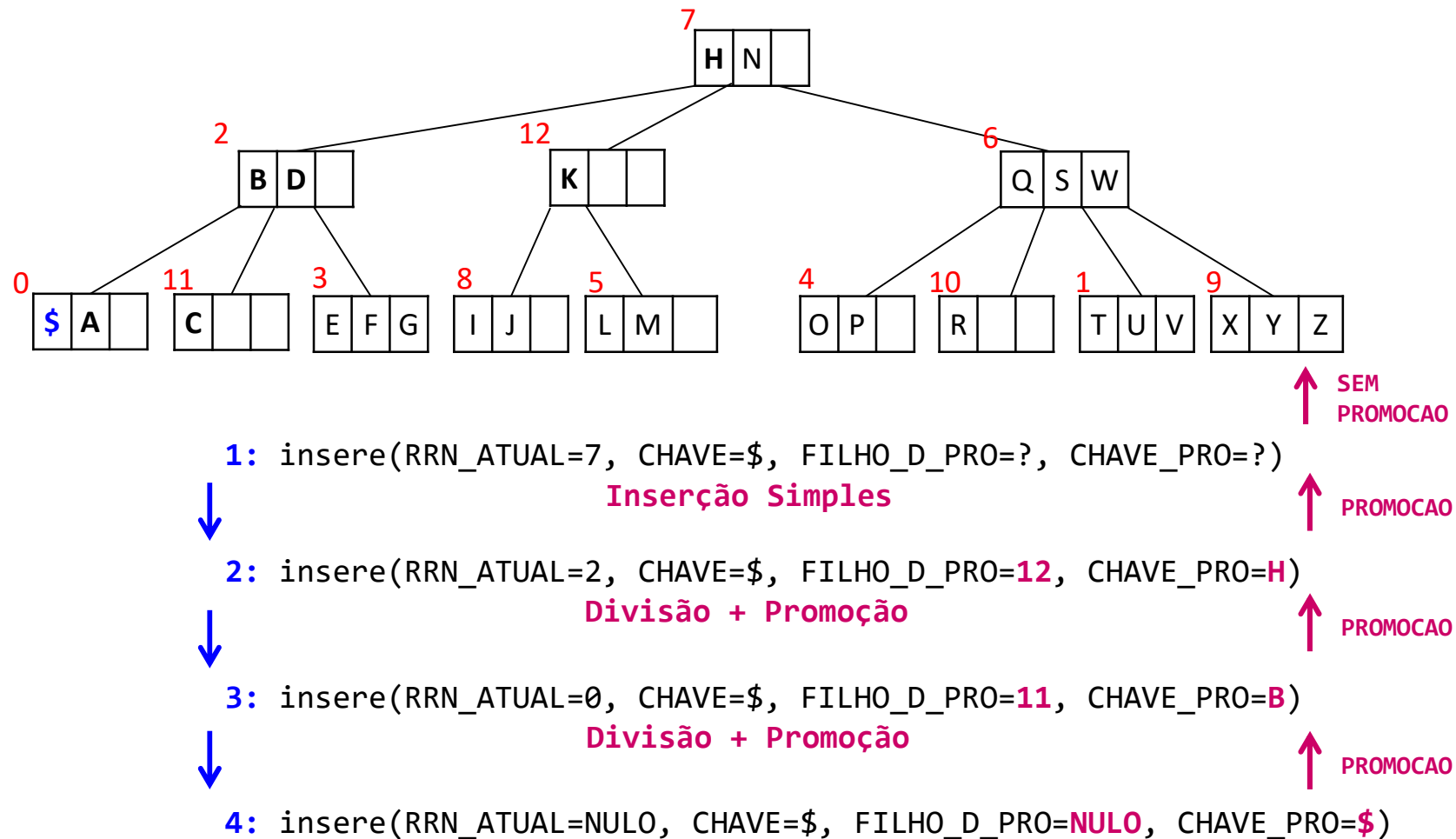


PROMOCAO



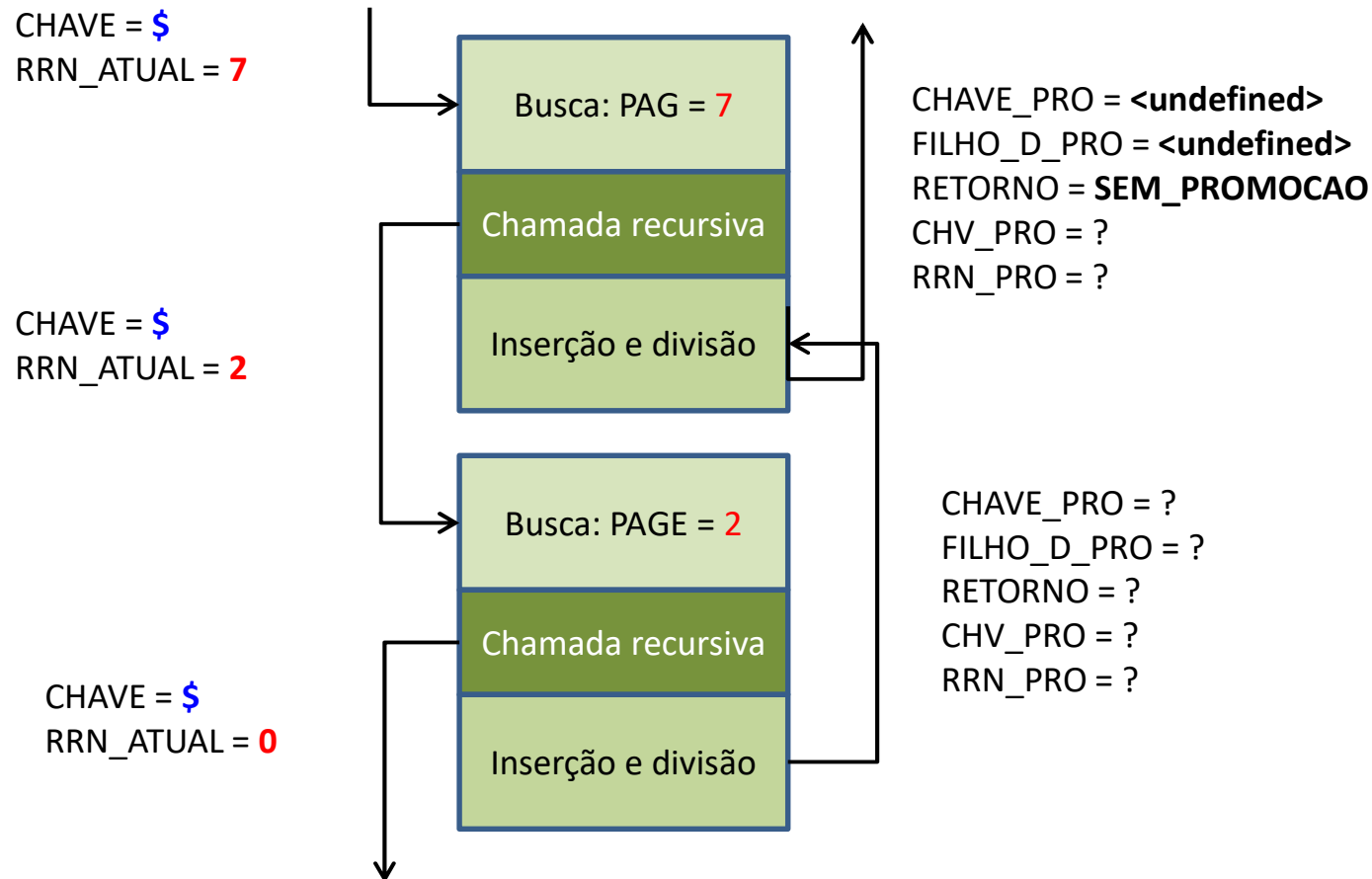
PROMOCAO

Inserção, divisão e promoção



Inserção, divisão e promoção

- ❑ Exercício: simular as chamadas recursivas para a **inserção da chave “\$”** na árvore do slide anterior usando a função *insere* (na tabela ASCII, o símbolo “\$” vem antes de qualquer letra)




Inserção, divisão e promoção

❑ Na sua implementação, a função *insere* usa várias funções auxiliares, tais como:

- `le_pagina(RRN, PAG)`
- `escreve_pagina(RRN, PAG)`
- `inicializa_pagina(PAG)`
- `busca_na_pagina(CHAVE, PAG, POS)`
- `insere_na_pagina(CHAVE, FILHO_D, PAG)`
- `divide(CHAVE, FILHO_D, PAG, CHAVE_PRO, FILHO_D_PRO, NOVAPAG)`


Inserção, divisão e promoção

FUNÇÃO *le_pagina* (RRN, **PAG)**  **Parâmetro de retorno**

```
FUNÇÃO le_pagina (RRN, PAG) {  
    calcule o byte-offset da página a partir do RRN  
    faça seek no arquivo árvore-B para o byte-offset calculado  
    leia PAG do arquivo árvore-B  
}
```

FUNÇÃO *escreve_pagina* (RRN, PAG)

```
FUNÇÃO escreve_pagina (RRN, PAG) {  
    calcule o byte-offset da página a partir do RRN  
    faça seek no arquivo árvore-B para o byte-offset calculado  
    escreva PAG no arquivo árvore-B  
}
```

FUNÇÃO *inicializa_pagina* (PAG**)**  **Parâmetro de retorno**

```
FUNÇÃO inicializa_pagina (PAG) {  
    PAG.CONTACHAVES = 0  
    para i de 0 até MAXCHAVES-1 faça  
        PAG.CHAVES[i] = VAZIO  
        PAG.FILHOS[i] = NIL  
    PAG.FILHOS[MAXCHAVES] = NIL  
}
```

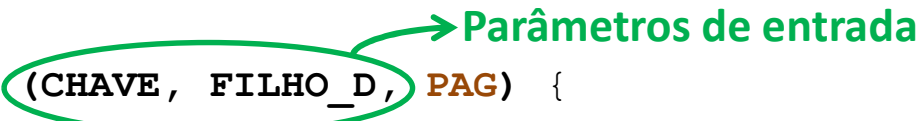
Inserção, divisão e promoção

```
FUNÇÃO le_pagina (RRN, PAG) {  
    calcule o byte-offset da página a partir do RRN  
    faça seek no arquivo árvore-B para o byte-offset calculado  
    leia PAG do arquivo árvore-B  
}
```

```
FUNÇÃO escreve_pagina (RRN, PAG) {  
    calcule o byte-offset da página a partir do RRN  
    faça seek no arquivo árvore-B para o byte-offset calculado  
    escreva PAG no arquivo árvore-B  
}
```

```
FUNÇÃO inicializa_pagina (PAG) {  
    PAG.CONTACHAVES = 0  
    para i de 0 até MAXCHAVES-1 faça  
        PAG.CHAVES[i] = VAZIO  
        PAG.FILHOS[i] = NIL  
    PAG.FILHOS[MAXCHAVES] = NIL  
}
```

Inserção, divisão e promoção

Parâmetros de entrada

```
FUNÇÃO insere_na_pagina (CHAVE, FILHO_D, PAG) {  
    faça i receber PAG.CONTACHAVES  
    enquanto i > 0 e CHAVE < PAG.CHAVES[i-1] faça  
        PAG.CHAVES[i] = PAG.CHAVES[i-1];  
        PAG.FILHOS[i+1] = PAG.FILHOS[i];  
        decmente i  
    fim enquanto  
    incremente PAG.CONTACHAVES  
    faça PAG.CHAVES[i] receber CHAVE  
    faça PAG.FILHOS[i+1] receber FILHO_D  
}
```

Inserção, divisão e promoção

```
FUNÇÃO insere_na_pagina (CHAVE, FILHO_D, PAG) {  
    faça i receber PAG.CONTACHAVES  
    enquanto i > 0 e CHAVE < PAG.CHAVES[i-1] faça  
        PAG.CHAVES[i] = PAG.CHAVES[i-1];  
        PAG.FILHOS[i+1] = PAG.FILHOS[i];  
        decrémente i  
    fim enquanto  
    incremente PAG.CONTACHAVES  
    faça PAG.CHAVES[i] receber CHAVE  
    faça PAG.FILHOS[i+1] receber FILHO_D  
}
```

Parâmetro de retorno

Inserção, divisão e promoção

```
FUNÇÃO insere_na_pagina (CHAVE, FILHO_D, PAG) {  
    faça i receber PAG.CONTACHAVES  
    enquanto i > 0 e CHAVE < PAG.CHAVES[i-1] faça  
        PAG.CHAVES[i] = PAG.CHAVES[i-1];  
        PAG.FILHOS[i+1] = PAG.FILHOS[i];  
        decmente i  
    fim enquanto  
    incremente PAG.CONTACHAVES  
    faça PAG.CHAVES[i] receber CHAVE  
    faça PAG.FILHOS[i+1] receber FILHO_D  
}
```


Inserção, divisão e promoção

❑ Função **divide**

- Cria uma nova página
- Distribui as chaves entre a página atual e a nova página
 - Usa uma página auxiliar para isso
- Determina qual chave e qual RRN (ponteiro para o filho direito) promover
 - Chave → sempre é a chave mediana da página auxiliar
 - Como a página auxiliar tem um tamanho fixo, a chave mediana sempre estará na mesma posição
 - RRN (ponteiro do filho direito) → sempre é o RRN da nova página
 - A nova página sempre é gravada no fim do arquivo → função **RRN_novapag()**

Inserção, divisão e promoção

→ Parâmetros de entrada

PROC **divide** (**CHAVE**, **FILHO_D**, **PAG**, **CHAVE_PRO**, **FILHO_D_PRO**, **NOVAPAG**)

Copie PAG para **PAGAUX**, que terá espaço para uma chave e um ponteiro extras

Insira CHAVE e FILHO_D nos lugares apropriados em PAGAUX

Faça CHAVE_PRO receber o valor da chave mediana de PAGAUX, que será promovida após o retorno da função **divide**

Faça FILHO_D_PRO receber o RRN de NOVAPAG

Copie as chaves e ponteiros que vêm antes de CHAVE_PRO em PAGAUX para PAG

Copie as chaves e ponteiros que vêm depois de CHAVE_PRO em PAGAUX para NOVAPAG

fim PROCEDIMENTO

Inserção, divisão e promoção

→ Parâmetros de retorno

```
PROC divide (CHAVE, FILHO_D, PAG, CHAVE_PRO, FILHO_D_PRO, NOVAPAG)
```

Copie PAG para **PAG AUX**, que terá espaço para uma chave e um ponteiro extras

Insira CHAVE e FILHO_D nos lugares apropriados em PAG AUX

Faça CHAVE_PRO receber o valor da chave mediana de PAG AUX, que será promovida após o retorno da função **divide**

Faça FILHO_D_PRO receber o RRN de NOVAPAG

Copie as chaves e ponteiros que vêm antes de CHAVE_PRO em PAG AUX para PAG

Copie as chaves e ponteiros que vêm depois de CHAVE_PRO em PAG AUX para NOVAPAG

```
fim PROCEDIMENTO
```

Inserção, divisão e promoção

PROC **divide** (CHAVE, FILHO_D, PAG, CHAVE_PRO, FILHO_D_PRO, NOVAPAG)

Copie PAG para **PAG AUX**, que terá espaço para uma chave e um ponteiro extras

Insira CHAVE e FILHO_D nos lugares apropriados em PAG AUX

Faça CHAVE_PRO receber o valor da chave mediana de PAG AUX, que será promovida após o retorno da função **divide**

Faça FILHO_D_PRO receber o RRN de NOVAPAG

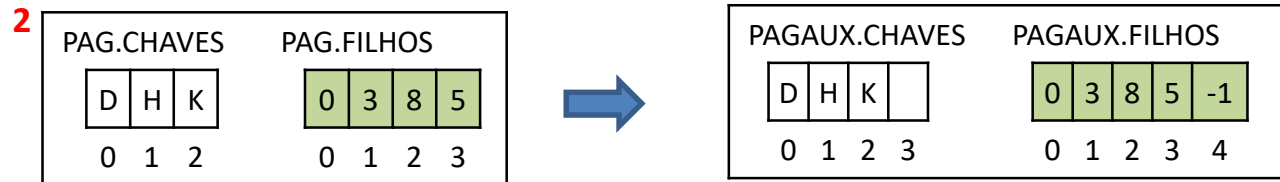
Copie as chaves e ponteiros que vêm antes de CHAVE_PRO em PAG AUX para PAG

Copie as chaves e ponteiros que vêm depois de CHAVE_PRO em PAG AUX para NOVAPAG

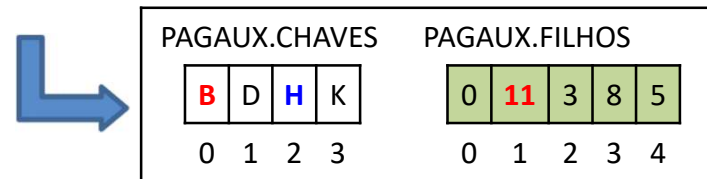
fim PROCEDIMENTO

Inserção, divisão e promoção

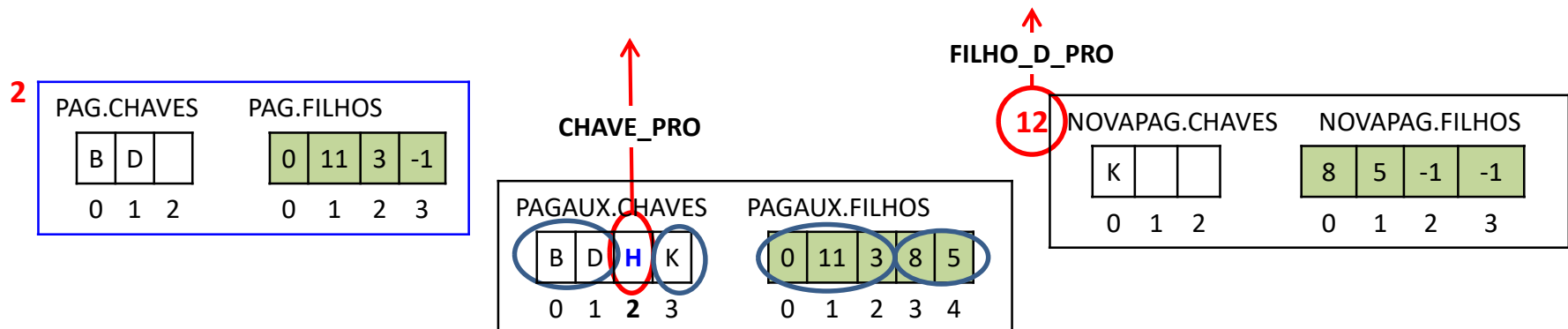
❑ Exemplo: *divide* (CHAVE = 'B', FILHO_D = 11, PAG = 2, CHAVE_PRO, FILHO_D_PRO, NOVAPAG)



— Insira CHAVE (**B**) e FILHO_D (**11**) na posição apropriada em PAGAUX



— Divida o conteúdo de PAGAUX entre PAG e NOVAPAG, exceto pela chave mediana (**H**), que será promovida juntamente com o RRN da NOVAPAG (**12**)



Inserção, divisão e promoção

```
PROC divide (CHAVE, FILHO_D, PAG, CHAVE_PRO, FILHO_D_PRO, NOVAPAG)
  copie PAG para PAG AUX
  /* Insira CHAVE e FILHO_D nos lugares apropriados em PAG AUX */
  faça i receber MAXCHAVES
  enquanto i > 0 e CHAVE < PAG AUX.CHAVES[i-1] faça
    PAG AUX.CHAVES[i] = PAG AUX.CHAVES[i-1]
    PAG AUX.FILHOS[i+1] = PAG AUX.FILHOS[i]
    decmente i
  fim enquanto
  PAG AUX.CHAVES[i] = CHAVE
  PAG AUX.FILHOS[i+1] = FILHO_D
  faça MEIO receber (MAXCHAVES + 1)/2
  faça FILHO_D_PRO receber o RRN que o NOVAPAG terá no arquivo árvore-b
  faça CHAVE_PRO receber PAG AUX.CHAVES[MEIO]
  /*continua no próximo slide*/
```

Inserção, divisão e promoção

```
/*continuação da função divide*/

/* Copie as chaves e ponteiros que vêm antes de CHAVE_PRO
   para PAG */

faça i receber 0

inicialize PAG

enquanto i < MEIO faça

    PAG.CHAVES[i] = PAG AUX.CHAVES[i]

    PAG.FILHOS[i] = PAG AUX.FILHOS[i]

    incremente PAG.CONTACHAVES

    incremente i

fim enquanto

PAG.FILHOS[i] = PAG AUX.FILHOS[i]

/*continua no próximo slide*/
```

Inserção, divisão e promoção

```
                /*continuação da função divide*/

/* Copie as chaves e ponteiros que vêm depois de CHAVE_PRO para
   NOVAPAG */

inicialize NOVAPAG

faça i receber MEIO + 1

enquanto i < MAXCHAVES + 1 faça
    NOVAPAG.CHAVES[NOVAPAG.CONTACHAVES] = PAG AUX.CHAVES[i]
    NOVAPAG.FILHOS[NOVAPAG.CONTACHAVES] = PAG AUX.FILHOS[i]
    incremente NOVAPAG.CONTACHAVES
    incremente i
fim enquanto

NOVAPAG.FILHOS[NOVAPAG.CONTACHAVES] = PAG AUX.FILHOS[i]

fim PROCEDIMENTO
```


Inserção, divisão e promoção

- ❑ Como saber qual RRN a NOVAPAG terá no arquivo árvore-b?
 - Sempre que uma página é criada, ela é gravada no fim do arquivo
 - As páginas têm tamanho fixo e conhecido → *sizeof(PAGINA)*

FUNÇÃO ***RRN_novapag()***

faça TAMANHOPAG receber o tamanho em bytes de uma página

faça TAMANHOCAB receber o tamanho em bytes do cabeçalho

faça BYTEOFFSET receber o byte-offset do fim do arquivo

retorne (BYTEOFFSET - TAMANHOCAB) / TAMANHOPAG

fim FUNÇÃO

Árvore-B

- ❑ Procedimento gerenciador: usado para ativar a função de inserção
 - Abre/cria o arquivo com a árvore-B e identifica/cria a página raiz
 - Assume que o RRN da raiz está armazenado no cabeçalho do arquivo da árvore-B, se o arquivo existir
 - Se a árvore-B ainda não existe, cria o arquivo, inicializa a raiz e grava a primeira página
 - Lê as chaves a serem armazenadas na árvore-B e chama a função *insere()*
 - Cria uma nova raiz quando houver divisão da raiz atual
 - Quando a função *insere()* retornar PROMOÇÃO
 - Cria a página que será a nova raiz
 - Atualiza o RRN da raiz

Árvore-B

```
PROCEDIMENTO gerenciador()
  se o arquivo árvore-B existe então
    abra-o para leitura e escrita
    leia o cabeçalho e armazene-o em RAIZ
  senão
    crie o arquivo árvore-B
    faça RAIZ receber 0 e a escreva no cabeçalho do arquivo
    inicialize NOVAPAG e a escreva no arquivo
    leia uma chave e armazene-a em CHAVE
  enquanto existirem chaves a serem inseridas faça
    se insere(RAIZ, CHAVE, FILHO_D_PRO, CHAVE_PRO) == PROMOCAO então
      inicialize NOVAPAG
      NOVAPAG.CHAVES[0] = CHAVE_PRO           /* nova chave raiz */
      NOVAPAG.FILHOS[0] = RAIZ                /* filho esquerdo */
      NOVAPAG.FILHOS[1] = FILHO_D_PRO         /* filho direito */
      escreva NOVAPAG no arquivo árvore-B
      faça RAIZ receber o RRN de NOVAPAG      /* RRN nova raiz */
    fim se
    leia a próxima chave e armazene-a em CHAVE
  fim enquanto
  escreva RAIZ no cabeçalho do arquivo
  feche o arquivo
fim PROCEDIMENTO
```