Árvores-B (Parte II)

SCC-503 Algoritmos e Estruturas de Dados II

Modificado por Moacir Ponti Jr, do original de:

Thiago A. S. Pardo

Cristina D. A. Ciferri

Leandro C. Cintra

M.C.F. de Oliveira

Algoritmo: Inserção

- Dois momentos
  - 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 na página
  - pesquisa da página antes da chamada recursiva

  2) chamada recursiva
  - move a operação para os níveis inferiores da árvore

  3) inserção, *split* e *promotion*
  - executados após as chamadas recursivas
  - a propagação destes processos ocorre nos retornos das chamada recursivas

caminho inverso ao da pesquisa

Procedimento inicial

- 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 função de inserção de forma apropriada
  - cria uma nova raiz quando a função de inserção particionar a raiz atual

Relembrando Inserção

1. Se árvore está vazia, crie a 1ª página (raiz), insira a chave e FIM
2. Senão, localize a página folha que deveria conter a chave
3. Se existe espaço, insira a chave, reordene a página e FIM
4. Senão (overflow):
    4.1. Divida a página em duas (split) e redistribua as chaves entre elas
    4.2. Se a pág. dividida era raiz, crie nova raiz como pai das duas resultantes
    4.3. Promova a menor chave da pág. direita como separadora no nó pai
    4.4. Se nó pai não sofreu overflow, FIM.
    4.5. Senão, volte ao passo 4.1 para o nó pai.

Estrutura de Dados

Estrutura de dados

- Estrutura de dados
  - determina cada página de disco
  - pode ser implementada de diferentes formas

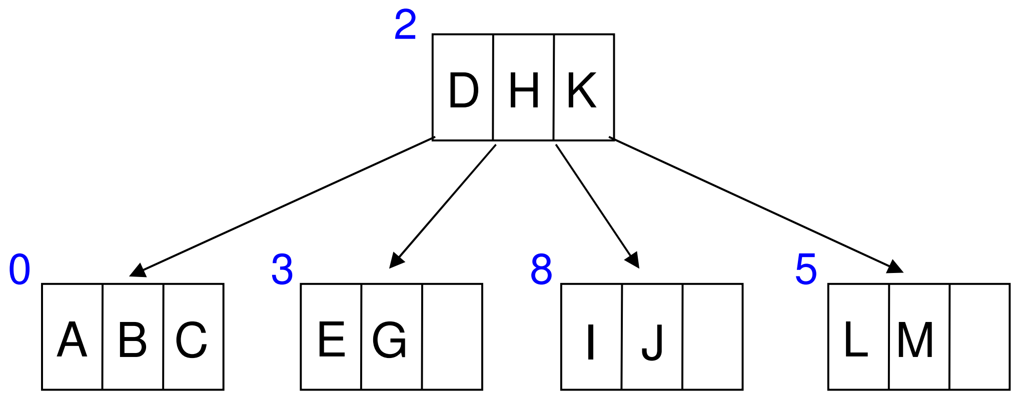
- Implementação adotada
  - contador de ocupação (número de chaves por página)
  - chaves
  - ponteiros $\Rightarrow$ campos de referência para as páginas filhas

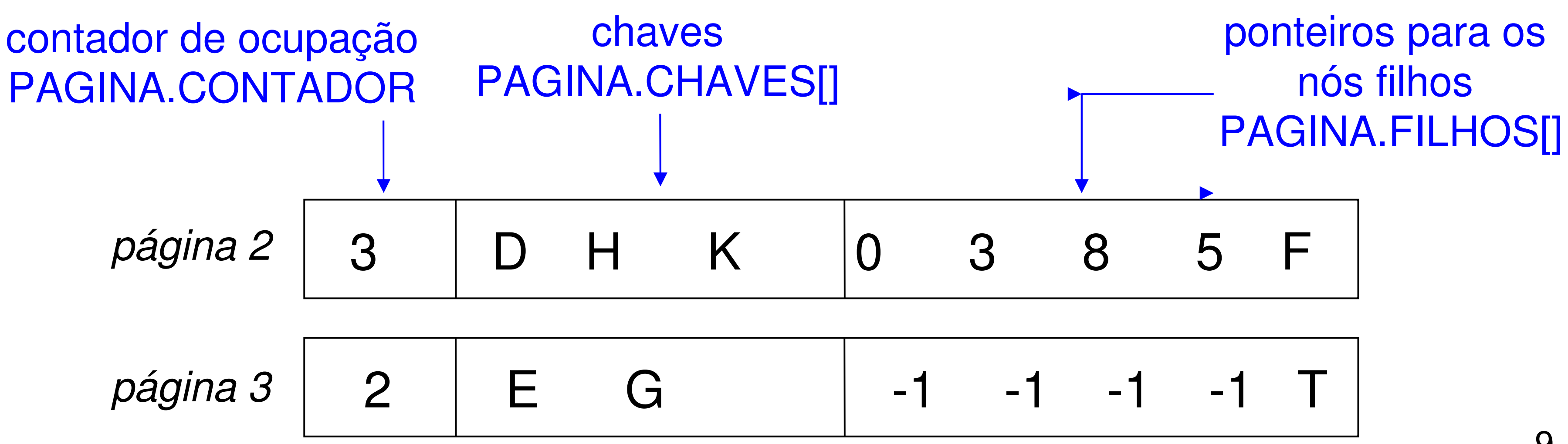
Estrutura de dados

- Possibilidade

```
 #define ordem 8

typedef struct pagina {
     int contador;
     char chaves[ordem-1]; //assumindo chaves char
     int filhos[ordem]; //armazena o RRN dos filhos
      bool folha;
  } PAGINA;
```

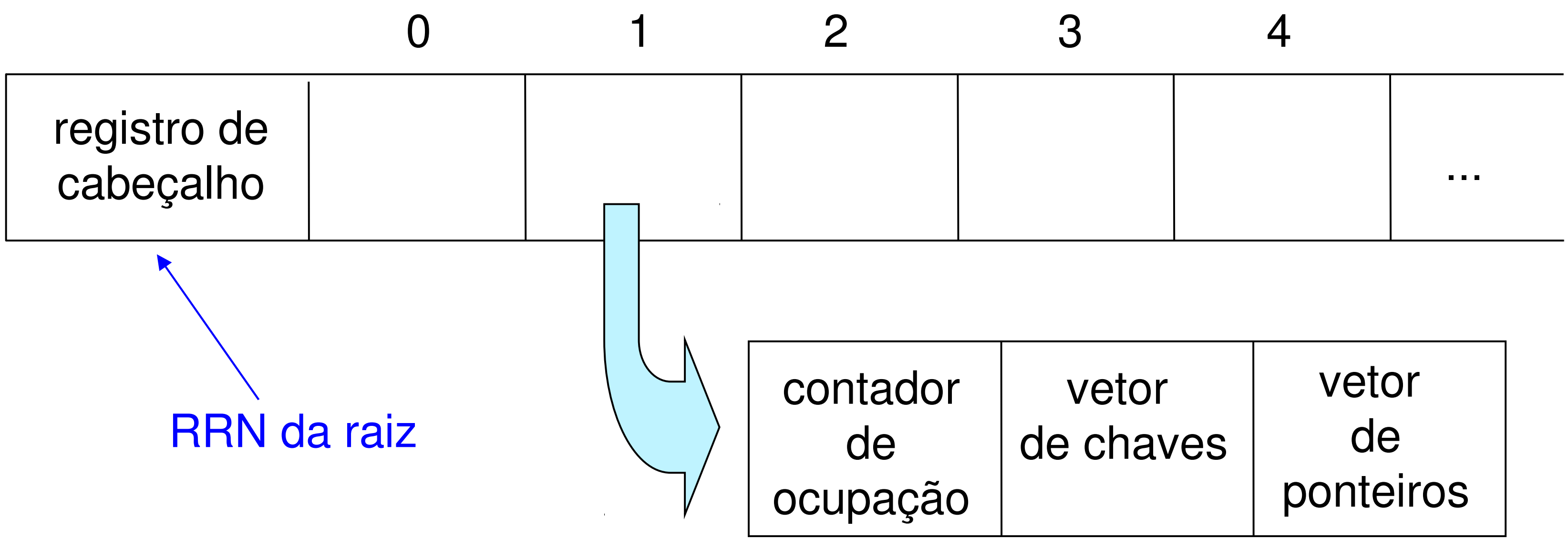
Arquivo da Árvore-B





Arquivo da Árvore-B

- Conjunto de registros de tamanho fixo



- Cada registro
  - contém uma página de disco

Algoritmo – pesquisa

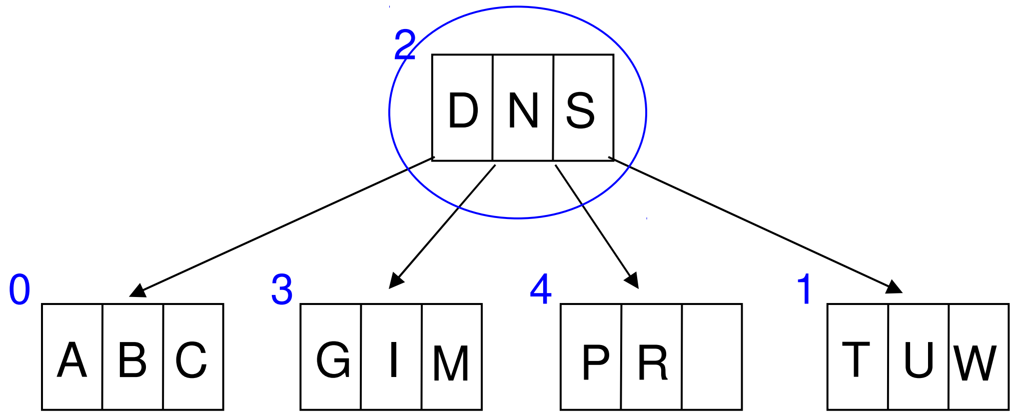
Algoritmos

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

Algoritmo: Pesquisa

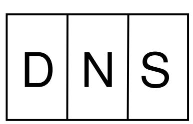
```
função busca(RRN,                       //página atual sendo pesquisada
                chave,                   //chave sendo procurada
                RRN_encontrado,          //retorna a página que contém a chave
                pos_encontrada)          //retorna posição da chave na página

se (RRN == -1) então
    retorne FALSO                        //chave de busca não encontrada

senão
    leia página P identificada por RRN
    procure chave em P, e atribua a POS a posição onde a chave deve ocorrer
    se (chave_encontrada) então
        RRN_encontrado = RRN             //RRN atual contém a chave
        pos_encontrada = POS             //POS contém a posição da chave na página
        retorne VERDADEIRO
    //se chave não encontrada, recomeça-se  busca no filho apropriado
    senão
        retorne busca(P.filhos[POS], chave, RRN_encontrado, pos_encontrada)
```

Busca da Chave K



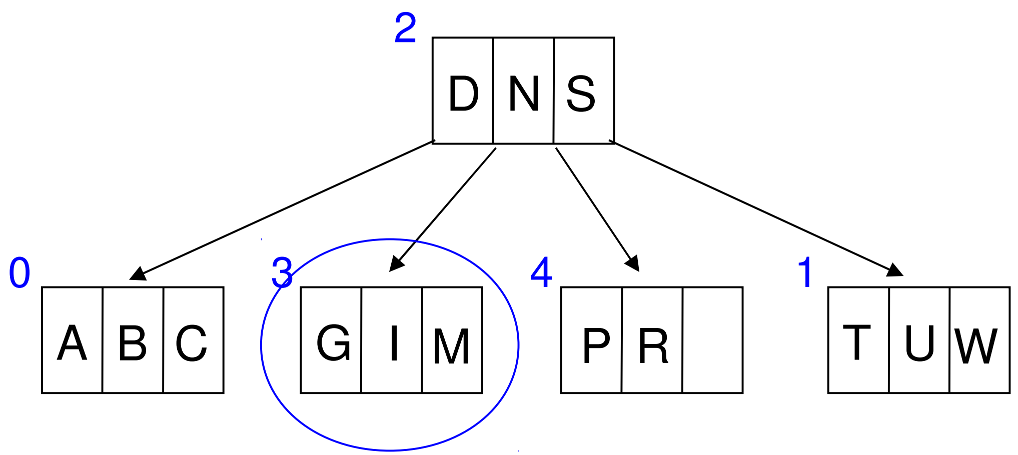
- busca(2, K, RRN_encontrado, pos_encontrada)

P =



não existe → POS = 1

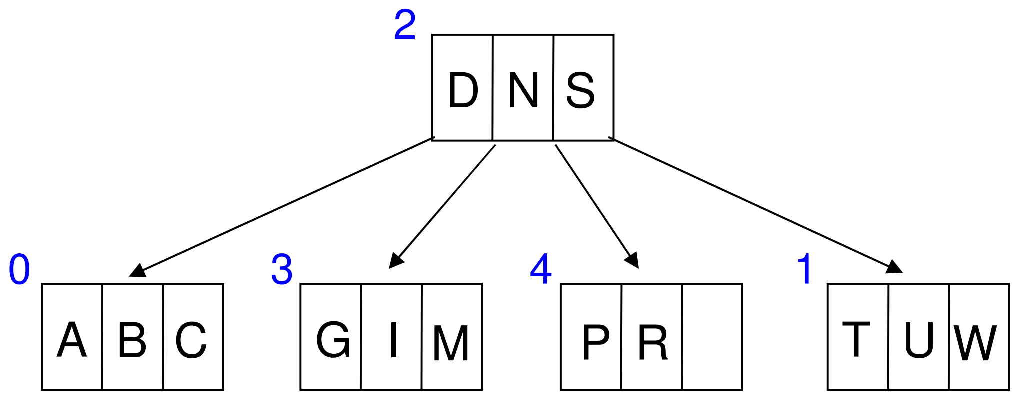
Busca da Chave K



- busca(P.filhos[1], K, RRN_encontrado, pos_encontrada)

P = [G | I | M] não existe → POS = 2

Busca da Chave K

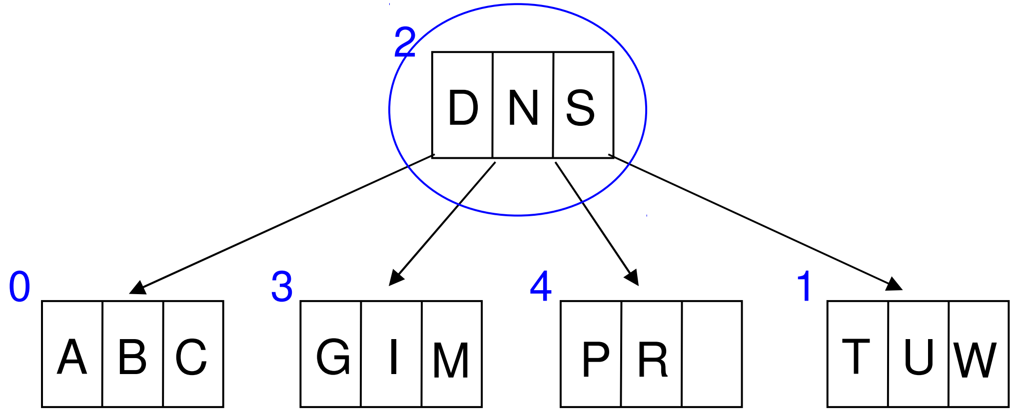


- busca(Pag.filhos[2], K, RRN_encontrado, pos_encontrada)

P.filhos[2] = -1 $\rightarrow$ chave de busca não encontrada

retorna **FALSO**

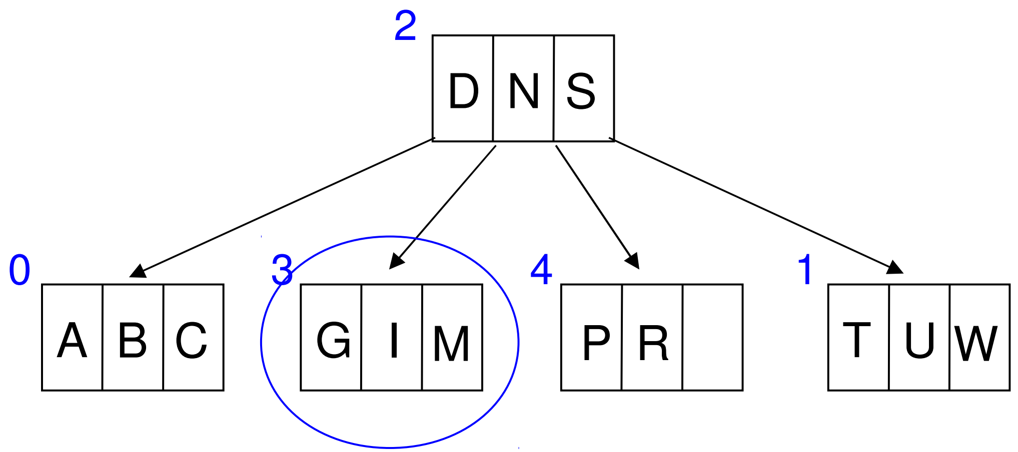
Busca da Chave M



- busca(2, M, RRN_encontrado, pos_encontrada)

P = | D | N | S | não existe $\rightarrow$ POS = 1

Busca da Chave M



- busca(P.filhos[1], M, RRN_encontrado, pos_encontrada)

P = | G | I | M |

chave de busca encontrada

pos_encontrada = 2

RRN_encontrado = 3

retorna **VERDADEIRO**

Exercício

- Em duplas, para entregar
  - Implemente em C o percurso em-ordem em uma árvore-B, imprimindo as chaves visitadas
  - Suponha que ela está em RAM, para facilitar

```
typedef struct pagina {
    int contador;
    char chaves[ordem-1];
    PAGINA *filhos[ordem];
} PAGINA;
```

```
em-ordem(no) {
  se (no.esq != NULL)
        em-ordem(no.esq)
  print no.value
  se (no.dir != NULL)
        em-ordem(no.dir)
}
```

Definição e Propriedades de Árvores-B

Relembrando...

- Ordem de uma árvore-B
  - Número máximo de descendentes que uma página, ou nó, pode possuir

- Em uma árvore-B de ordem **m**, o número máximo de chaves em uma página é **m-1**
  - Exemplo:
    - Uma árvore-B de ordem 8 tem, no máximo, 7 chaves por página

Propriedades das árvores-B

- Número mínimo de chaves por página
  - Quando uma página é particionada na inserção, as chaves são divididas (quase) igualmente entre as páginas velha e nova
    - o número mínimo de chaves em um nó é dado por $\lceil m/2 \rceil - 1$ (exceto para a raiz)
      - **Exemplo:** árvore B de ordem 8, armazena no máximo 7 chaves por página e tem, no mínimo, 3 chaves por página

Propriedades das árvores-B

- Para uma árvore-B de ordem **m**

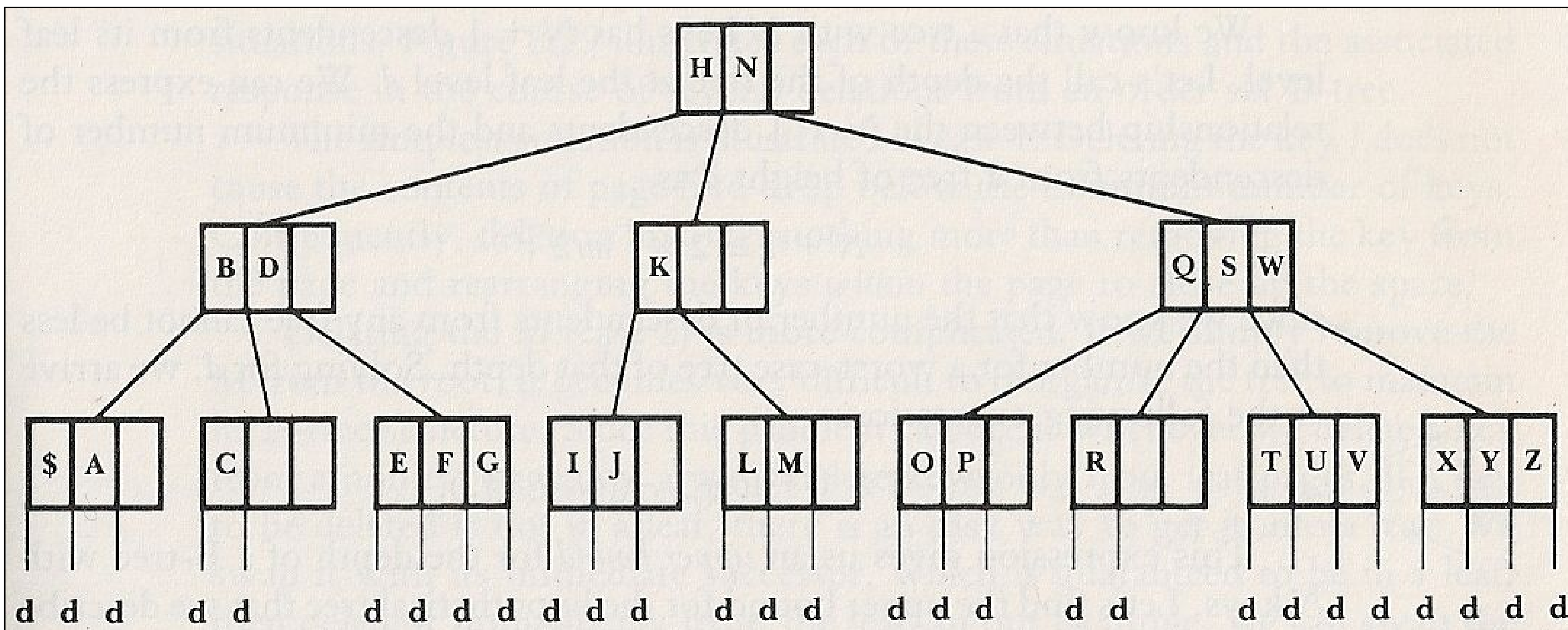
1. Cada página tem:
   - no máximo, **m** descendentes
   - no mínimo $\lceil$**m/2**$\rceil$ descendentes (exceto raiz e folhas)
2. A raiz tem, no mínimo, dois descendentes
   - a menos que seja uma folha
3. Todas as folhas estão no mesmo nível
4. Uma página não folha com **k** descendentes contém **k-1** chaves
5. Uma página folha contém, no mínimo $\lceil$**m/2**$\rceil$ **-1** e, no máximo, **m-1** chaves

Busca no pior caso

- Profundidade da busca no pior caso
  - Tendo X chaves e um tamanho de página Y, qual o número de acessos a disco necessário?
  - Ou “qual a profundidade da árvore”?

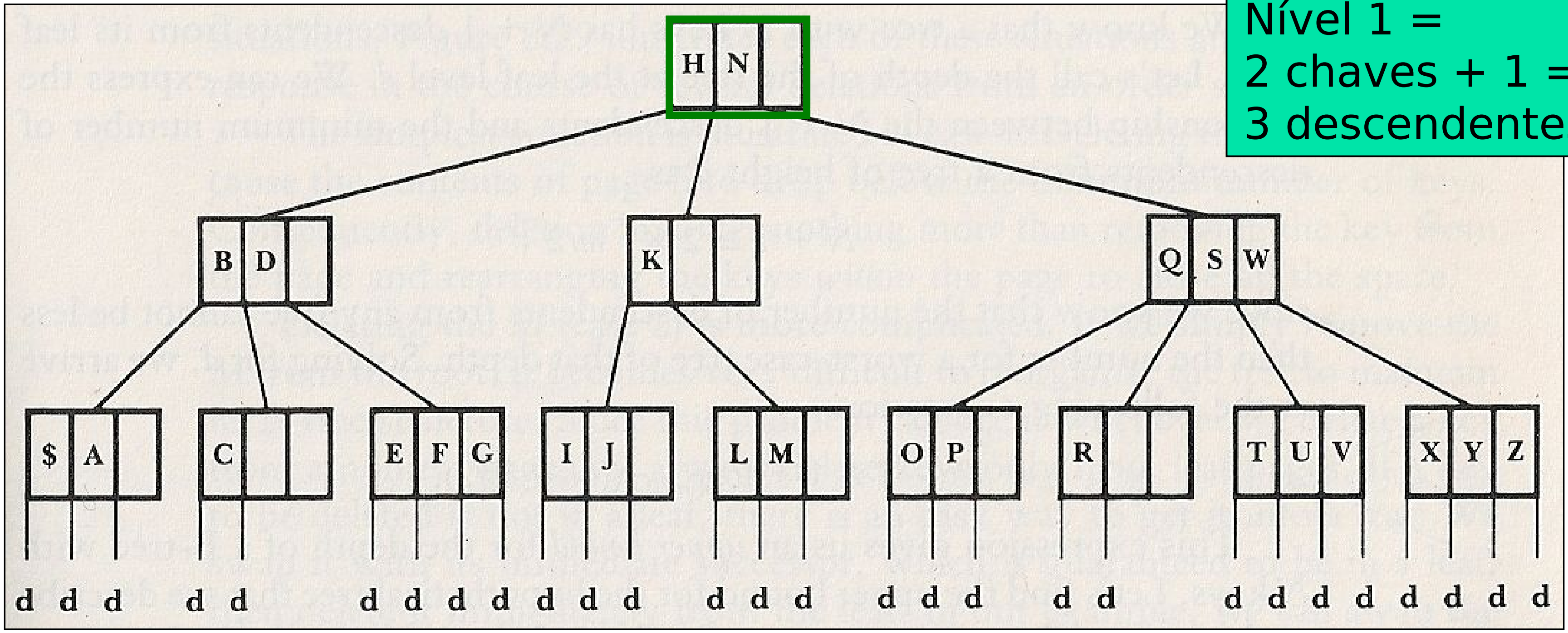
Propriedades das árvores-B

- Antes de mais nada, é importante observar
  - número de descendentes possíveis de um nível da árvore = número de chaves até o nível atual + 1



Propriedades das árvores-B

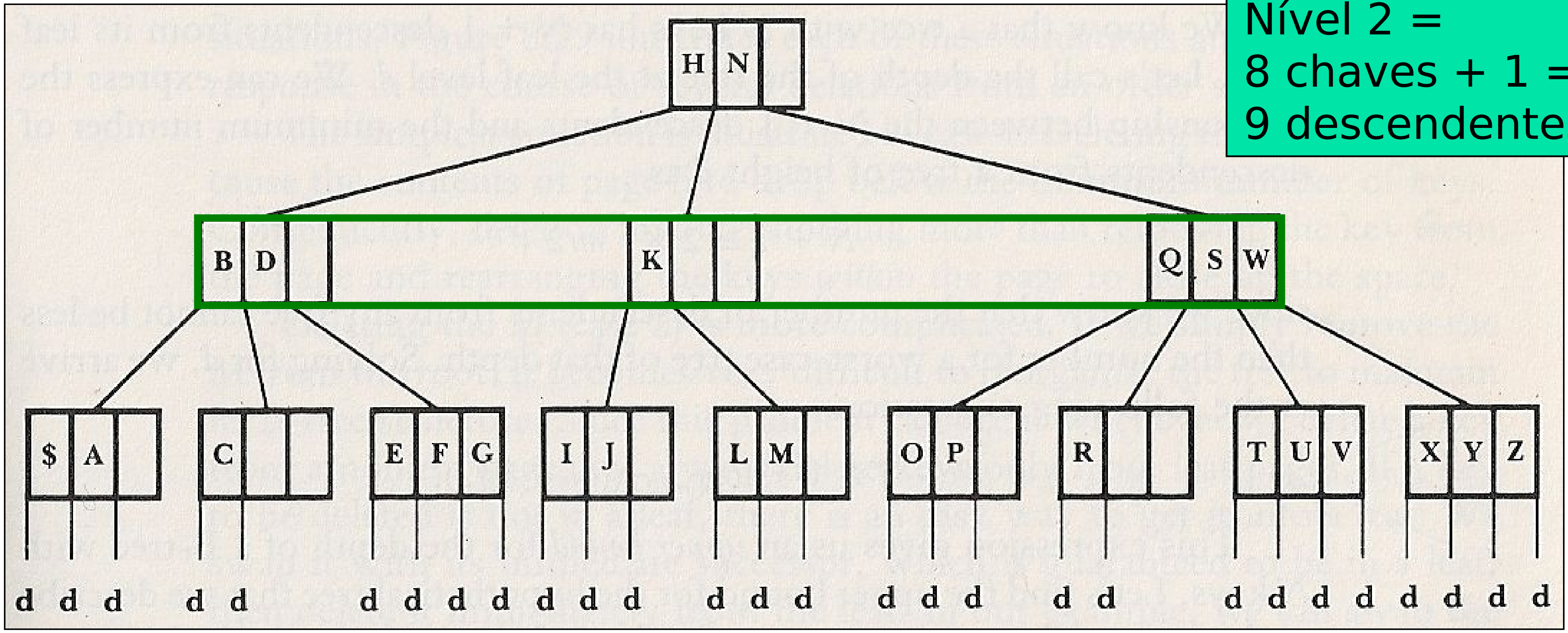
- Antes de mais nada, é importante observar
  - número de descendentes possíveis de um nível da árvore = número de chaves até o nível atual + 1



Nível 1 =
2 chaves + 1 =
3 descendentes

Propriedades das árvores-B

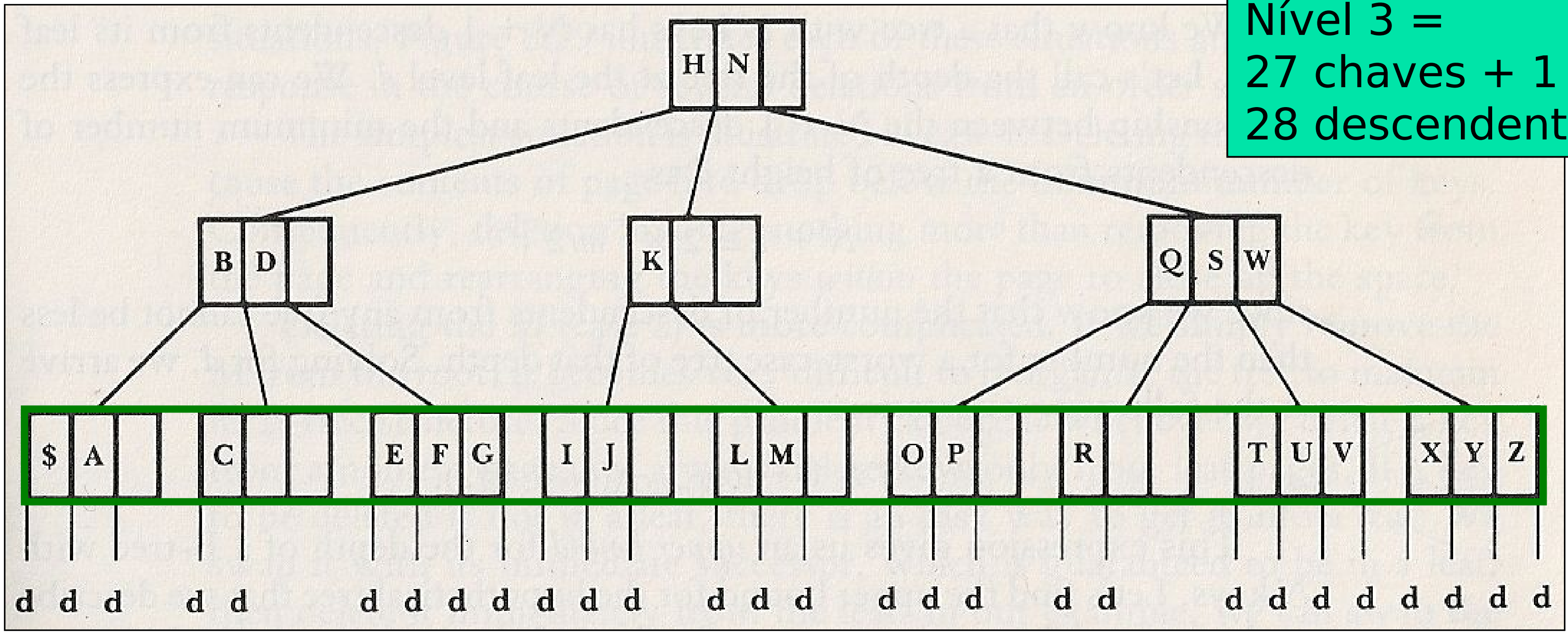
- Antes de mais nada, é importante observar
  - número de descendentes possíveis de um nível da árvore = número de chaves até o nível atual + 1



Nível 2 =
8 chaves + 1 =
9 descendentes

Propriedades das árvores-B

- Antes de mais nada, é importante observar
  - número de descendentes possíveis de um nível da árvore = número de chaves até o nível atual + 1



Nível 3 =
27 chaves + 1 =
28 descendentes

Altura de pior caso

- No pior caso, cada nó terá o mínimo de descendentes
  - A árvore terá sua <u>maior altura</u> e <u>menor largura</u>

    - Exceto raiz e folhas, há no mínimo $\lceil m/2 \rceil$ descendentes para cada nó

- Para uma árvore de ordem m
  - A raiz (primeiro nível) terá no mínimo 2 descendentes
  - O segundo nível terá somente 2 páginas, tendo cada uma $\lceil \mathbf{m/2} \rceil$ descendentes, ou seja, há 2 x $\lceil m/2 \rceil$ descendentes para o segundo nível
  - O terceiro nível contém 2 x $\lceil m/2 \rceil$ nós x $\lceil m/2 \rceil$ descendentes para cada nó, ou seja 2 x $\lceil m/2 \rceil^2$ descendentes
  - O nível d terá ?

Propriedades das árvores-B

- O nível d terá $2 \times \lceil m/2 \rceil^{d-1}$ descendentes

Propriedades das árvores-B

- Ou seja, o número mínimo de descendentes para um nível d da árvore é $2 \times \lceil m/2 \rceil^{d-1}$

- Vimos que, no máximo, há N+1 descendentes em um nível da árvore com N chaves

- Então, podemos calcular o limite superior da profundidade da árvore (e portanto o número máximo de acessos a disco) em termos do número de chaves nas folhas **N** e da ordem **m**

$$N \geq 2 \times \lceil m/2 \rceil^{d-1} \quad \Rightarrow \quad d \leq 1 + \log_{\lceil m/2 \rceil}(N/2)$$

Exemplo

Considerando que temos 1,000,000 chaves e uma árvore de ordem 512, temos que $d \leq 1 + \log_{\lceil 256 \rceil}(500{,}000)$, ou seja, $d \leq 3.37$

- Podemos esperar, portanto, não mais do que 3 acessos a disco para acessar qualquer uma das chaves

Algoritmos – parte 2

Eliminação

- O ***split*** garante a ***manutenção das propriedades da árvore-B*** durante a inserção

- Essas propriedades precisam ser mantidas, também, durante a eliminação de chaves

- Há <u>vários casos</u> para se analisar

Eliminação: Caso 1

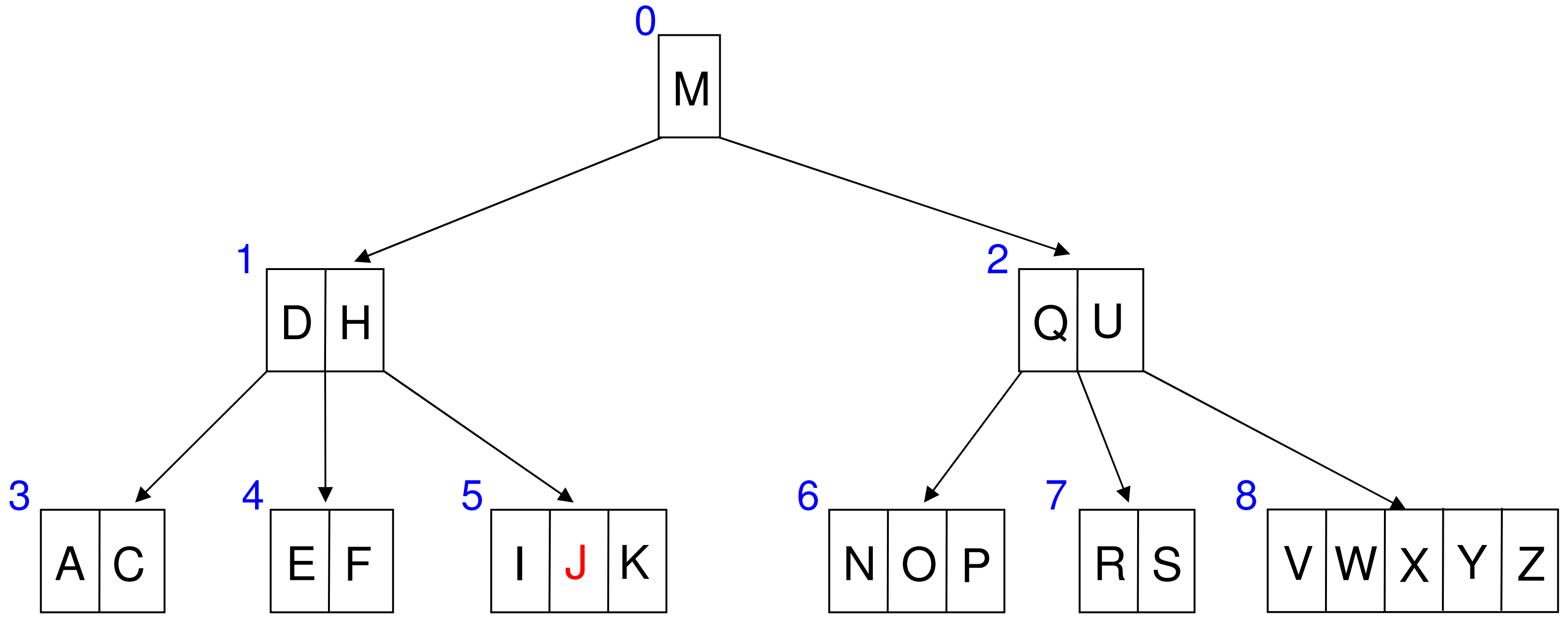
- **Caso 1:** eliminação de uma chave em uma página folha, sendo que o número mínimo de chaves na página é respeitado

- **Solução:** chave é retirada e os registros internos da página são reorganizados

Eliminação: Caso 1

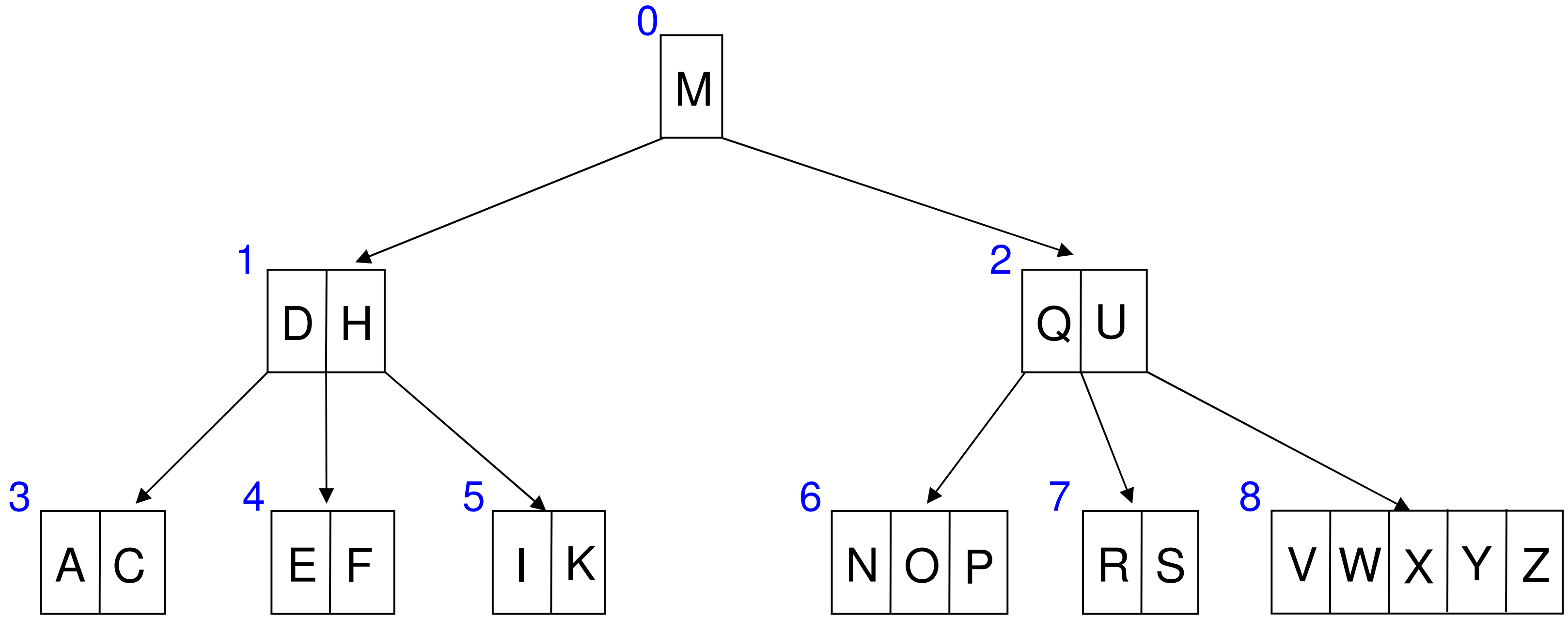
- Eliminando J
  - O que acontece?

Exemplo: árvore-B de ordem 6



Eliminação: Caso 1

- Eliminando J
  - Página 5 observa o número mínimo de elementos



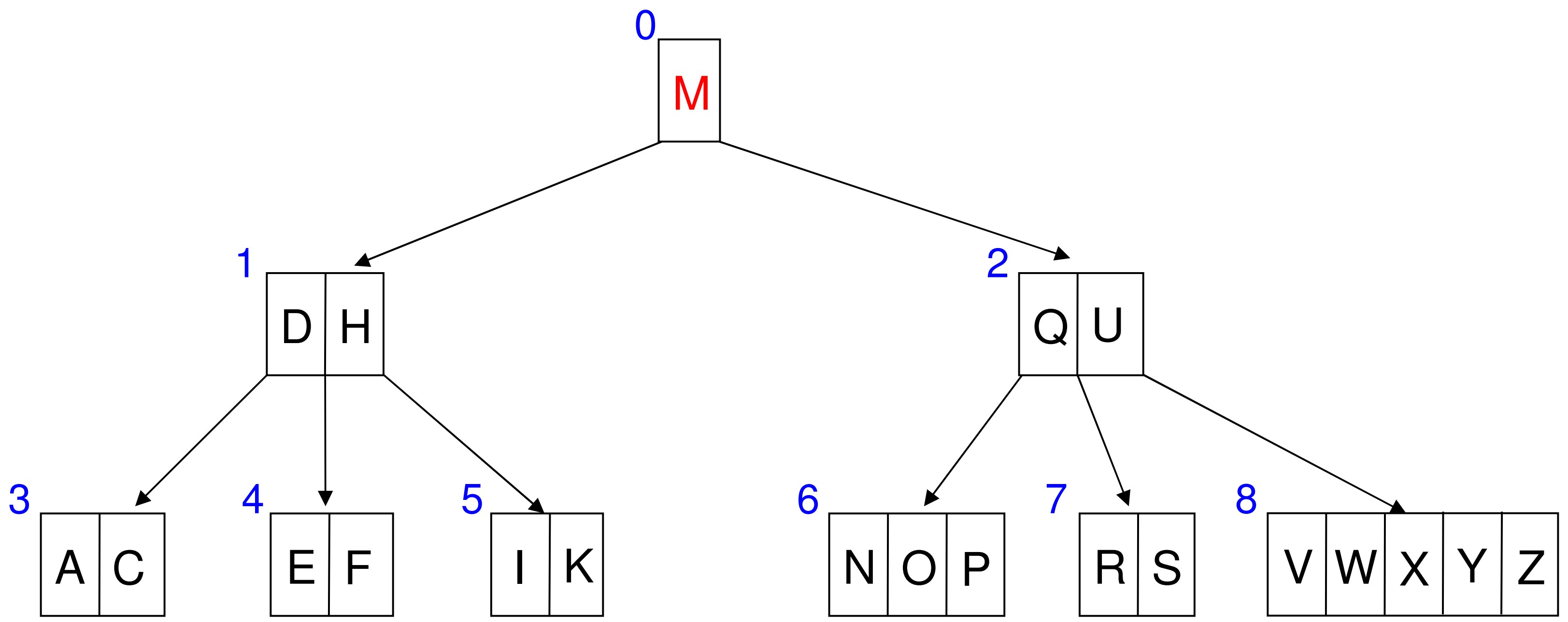
Eliminação: Caso 2

- **Caso 2:** eliminação de uma chave que não está em uma folha

- **Solução:** sempre eliminamos de páginas folha
  - Se uma chave deve ser eliminada de uma página que não é folha, trocamos a chave com sua sucessora imediata (ou com a predecessora imediata) que está numa folha
  - A seguir, eliminamos a chave da folha

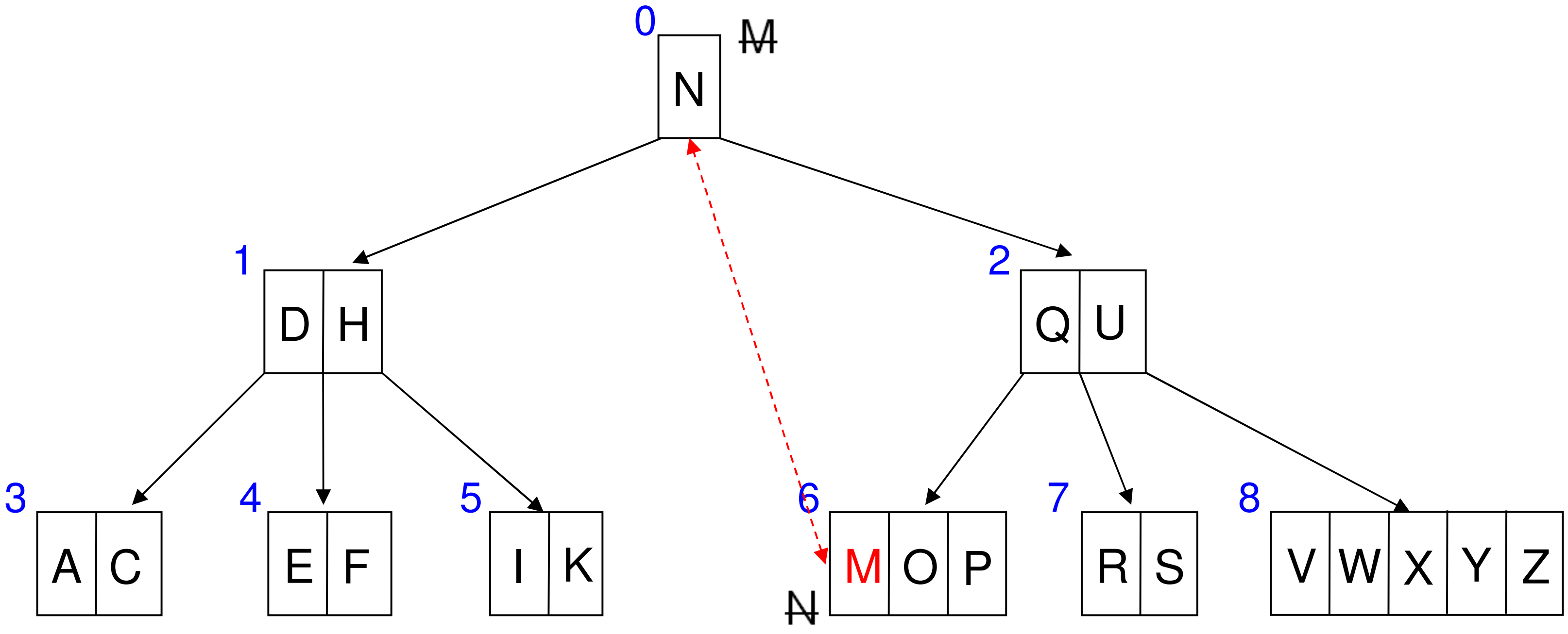
Eliminação: Caso 2

- Eliminando M
  - O que acontece?



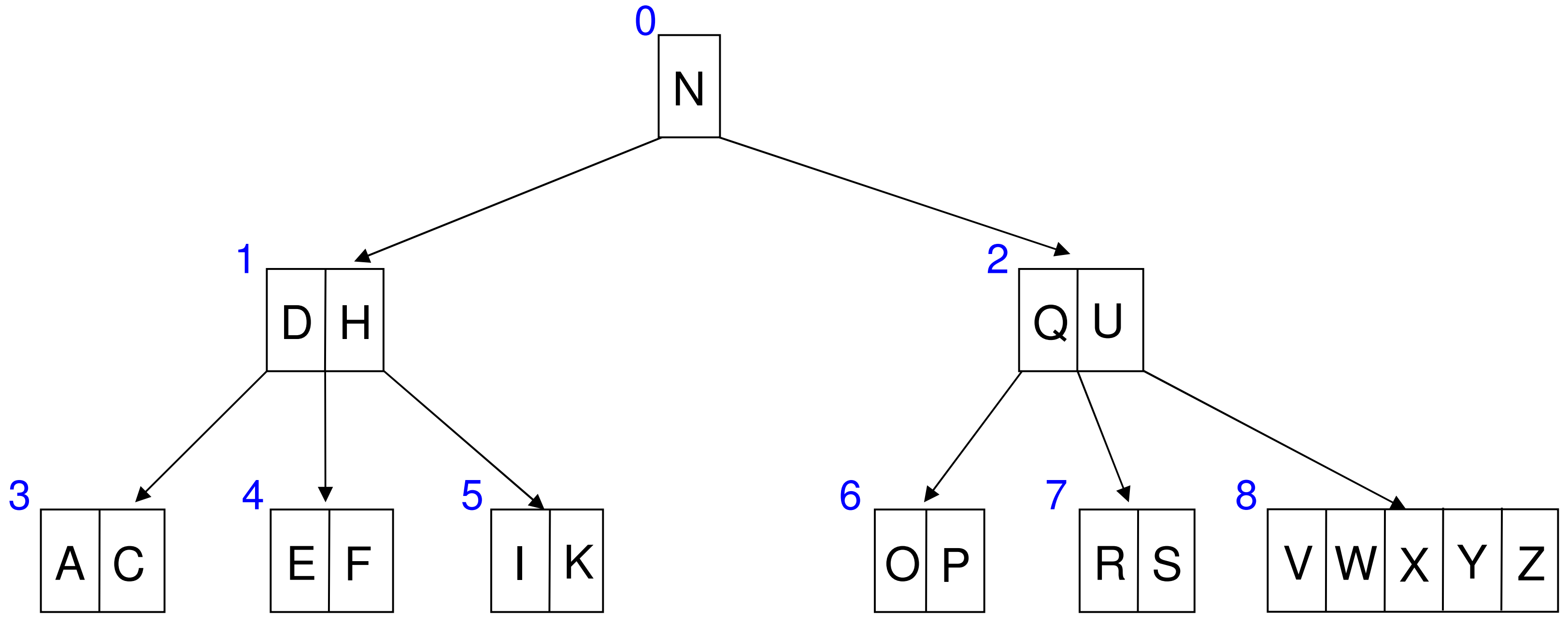
Eliminação: Caso 2

- Eliminando M
  - Troca-se M com N, então se elimina M



Eliminação: Caso 2

- Eliminando M
  - Troca-se M com N, então se elimina M



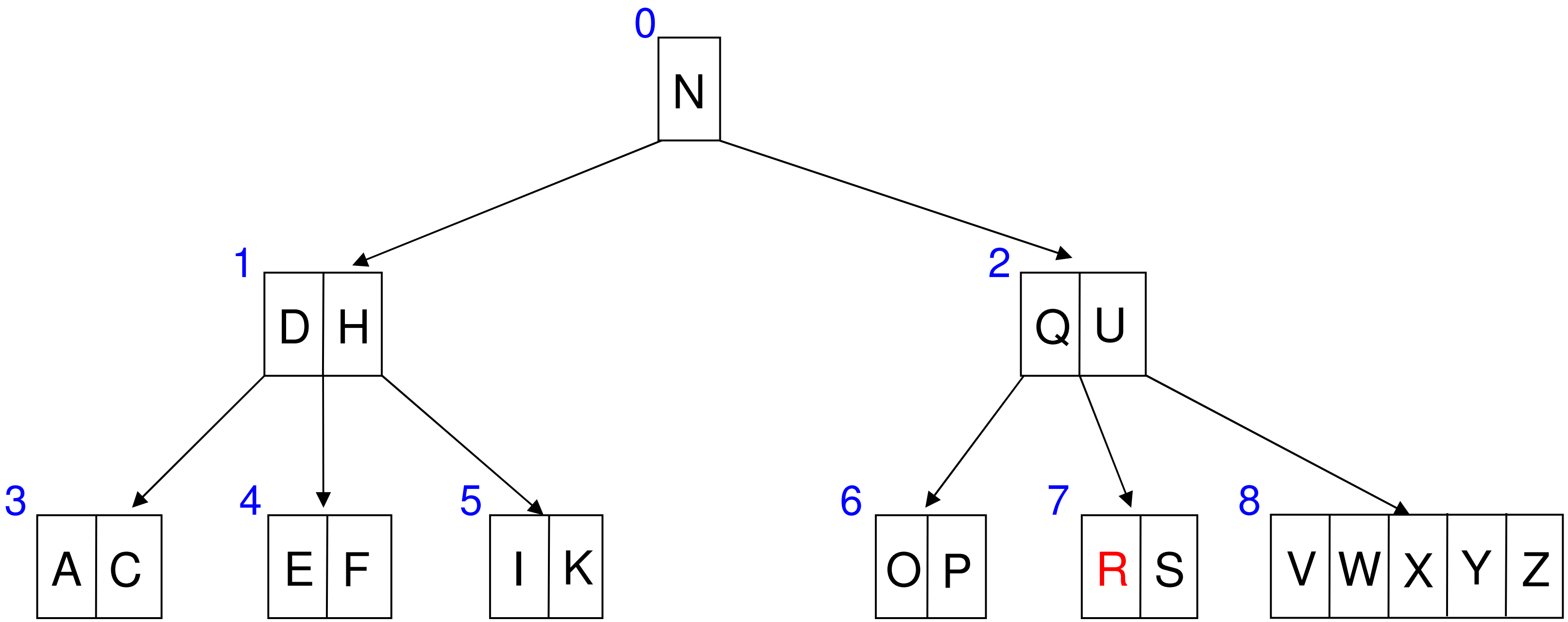
Eliminação: Caso 3

- **Caso 3:** eliminação causa *underflow* na página

- **Solução:** redistribuição
  - Procura-se uma página irmã (mesmo pai) que contenha mais chaves do que o mínimo: se existir, redistribuem-se as chaves entre essas páginas
  - A redistribuição pode provocar uma alteração na chave separadora que está no nó pai

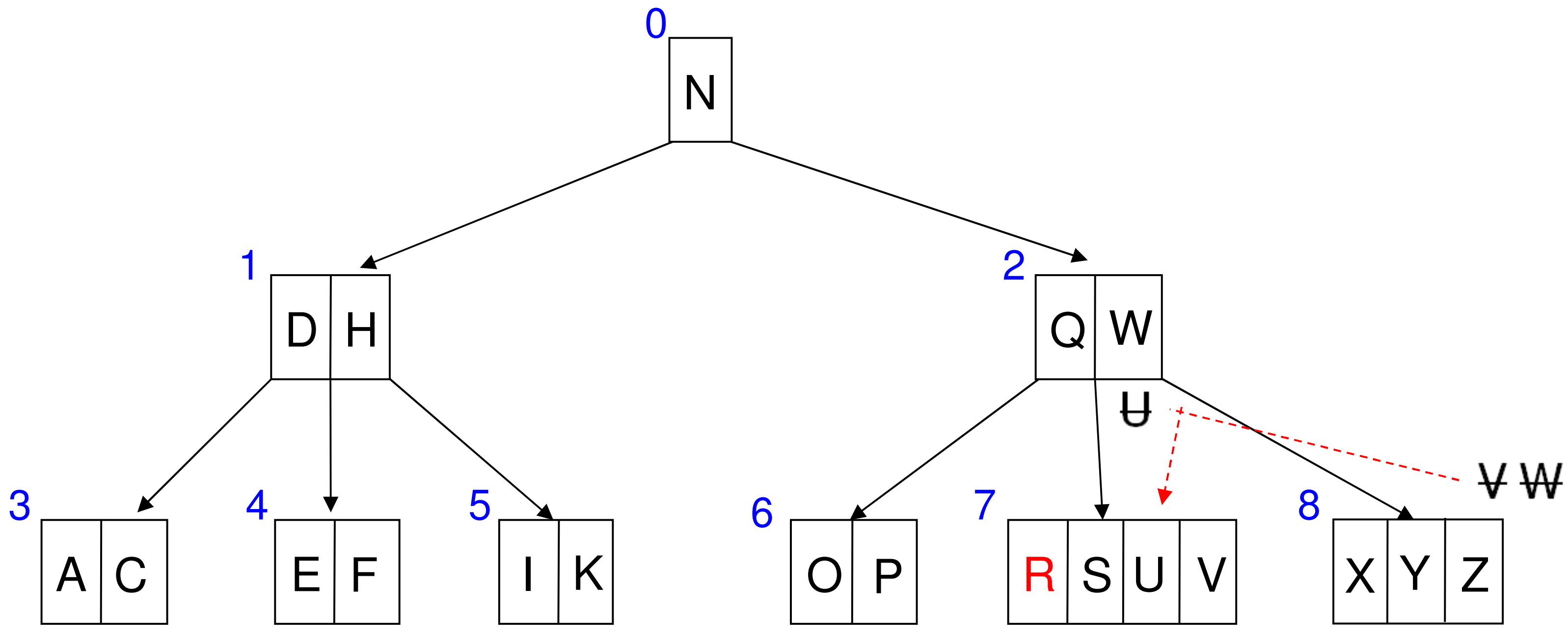
Eliminação: Caso 3

- Eliminando R
  - O que acontece?



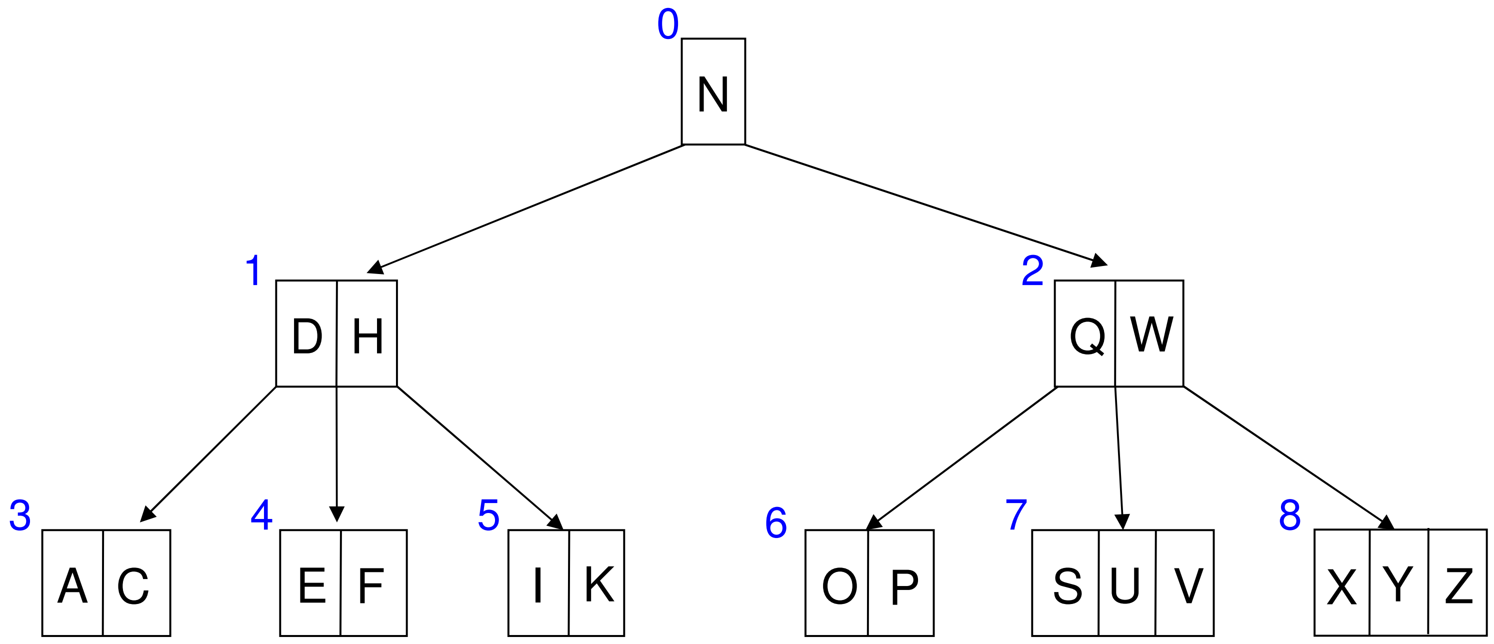
Eliminação: Caso 3

- Eliminando R
  - Ocorre *underflow* em página 7, redistribuem-se chaves entre páginas 7 e 8 (via página 2)



Eliminação: Caso 3

- Eliminando R
  - Ocorre *underflow* em página 7, redistribuem-se chaves entre páginas 7 e 8 (via página 2)



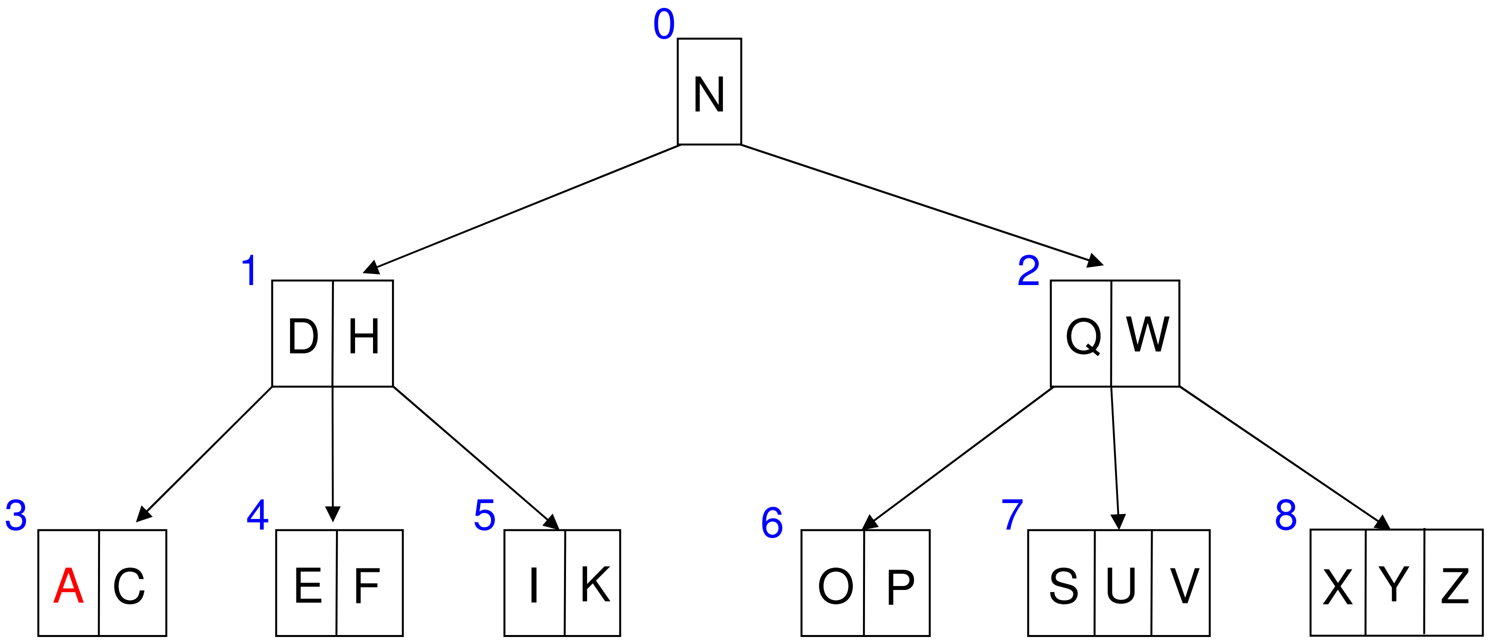
Eliminação: Caso 4

- **Caso 4:** ocorre *underflow* e a redistribuição não pode ser aplicada
  - Não existem chaves suficientes para dividir entre as duas páginas irmãs

- **Solução:** concatenação
  - Combina-se o conteúdo das duas páginas e a chave separadora da página pai para formar uma única página
  - A concatenação é o inverso do processo de particionamento
    - Como conseqüência, pode ocorrer *underflow* da página pai

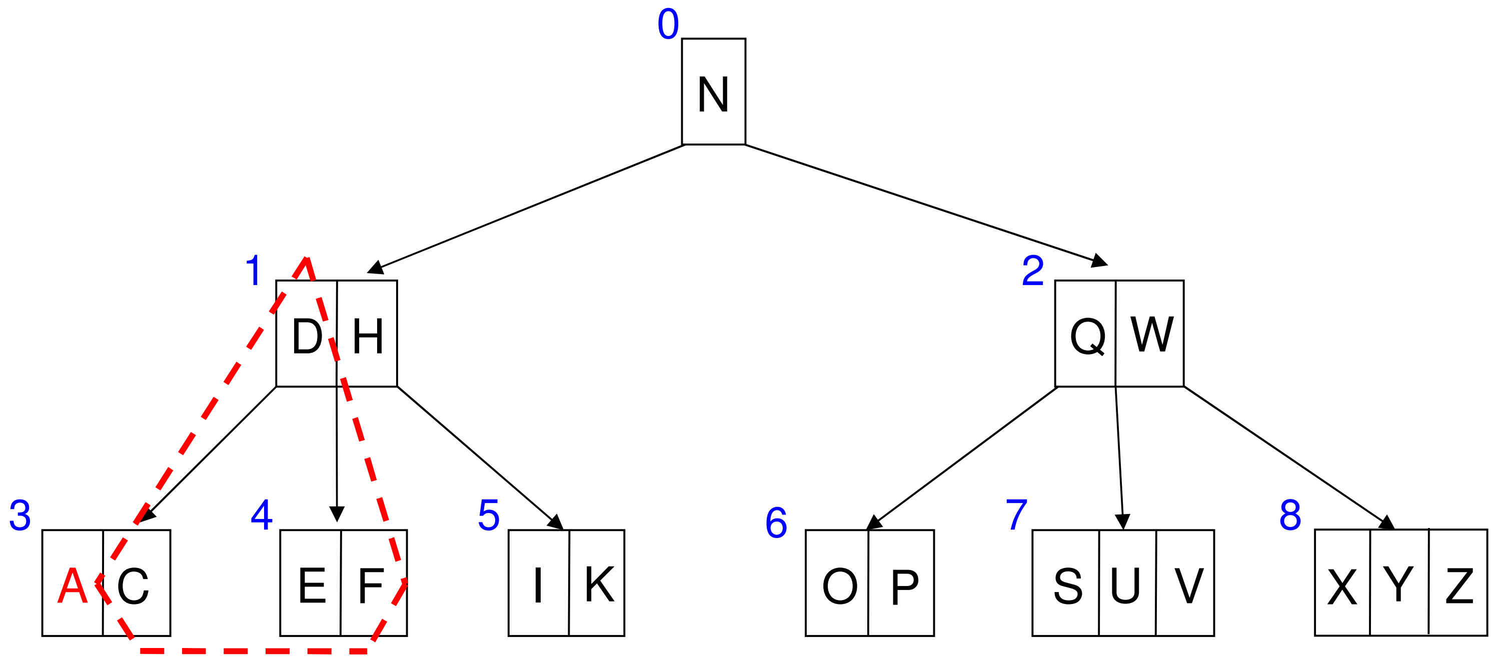
Eliminação: Caso 4

- Eliminando A
  - O que acontece?



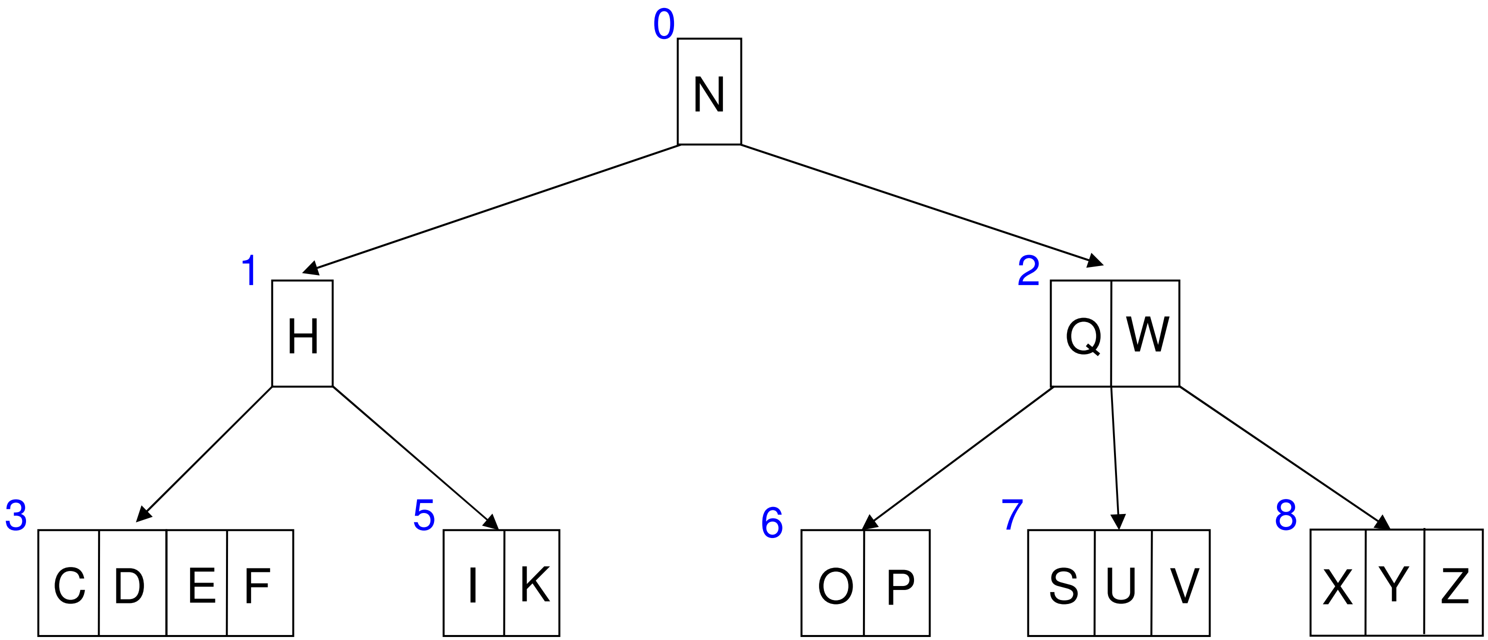
Eliminação: Caso 4

- Eliminando A
  - Ocorre underflow da página 3, não é possível fazer redistribuição, concatenam-se páginas 3 e 4, além da chave separadora D



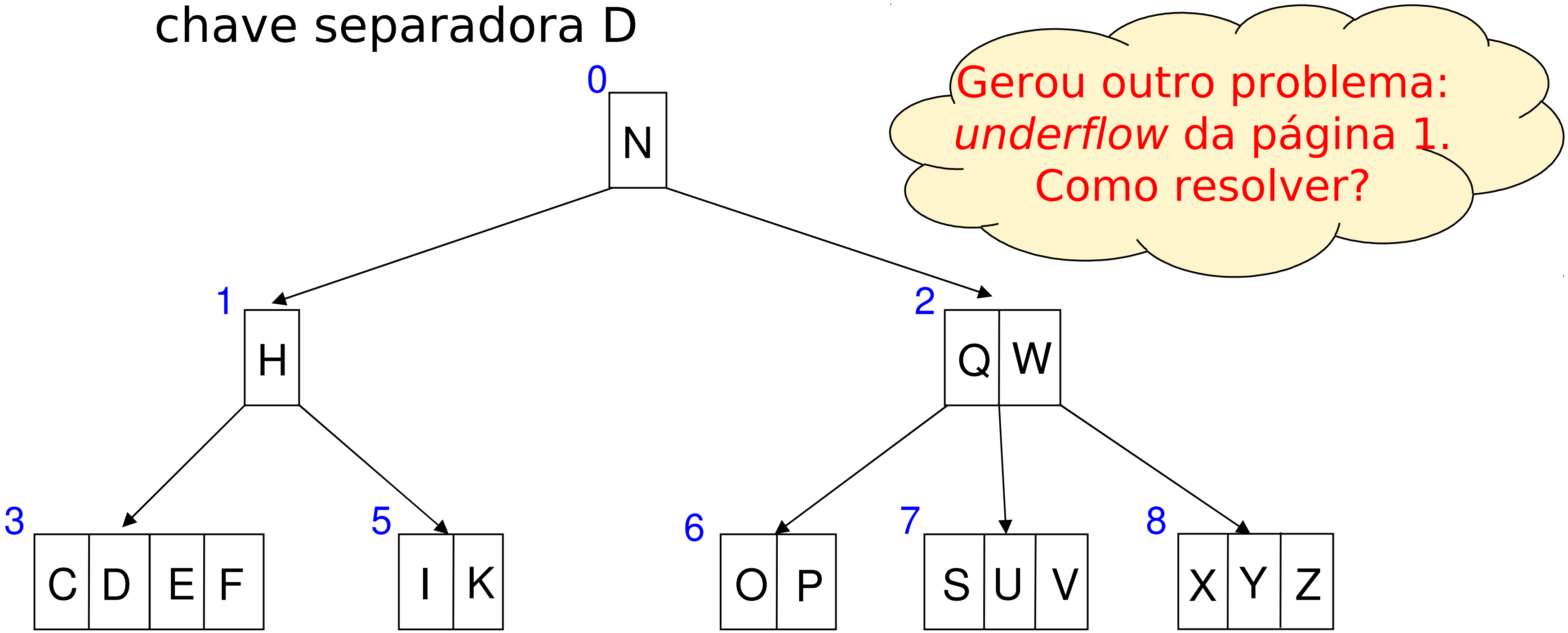
Eliminação: Caso 4

- Eliminando A
  - Ocorre underflow da página 3, não é possível fazer redistribuição, concatenam-se páginas 3 e 4, além da chave separadora D



Eliminação: Caso 4

- Eliminando A
  - Ocorre underflow da página 3, não é possível fazer redistribuição, concatenam-se páginas 3 e 4, além da chave separadora D



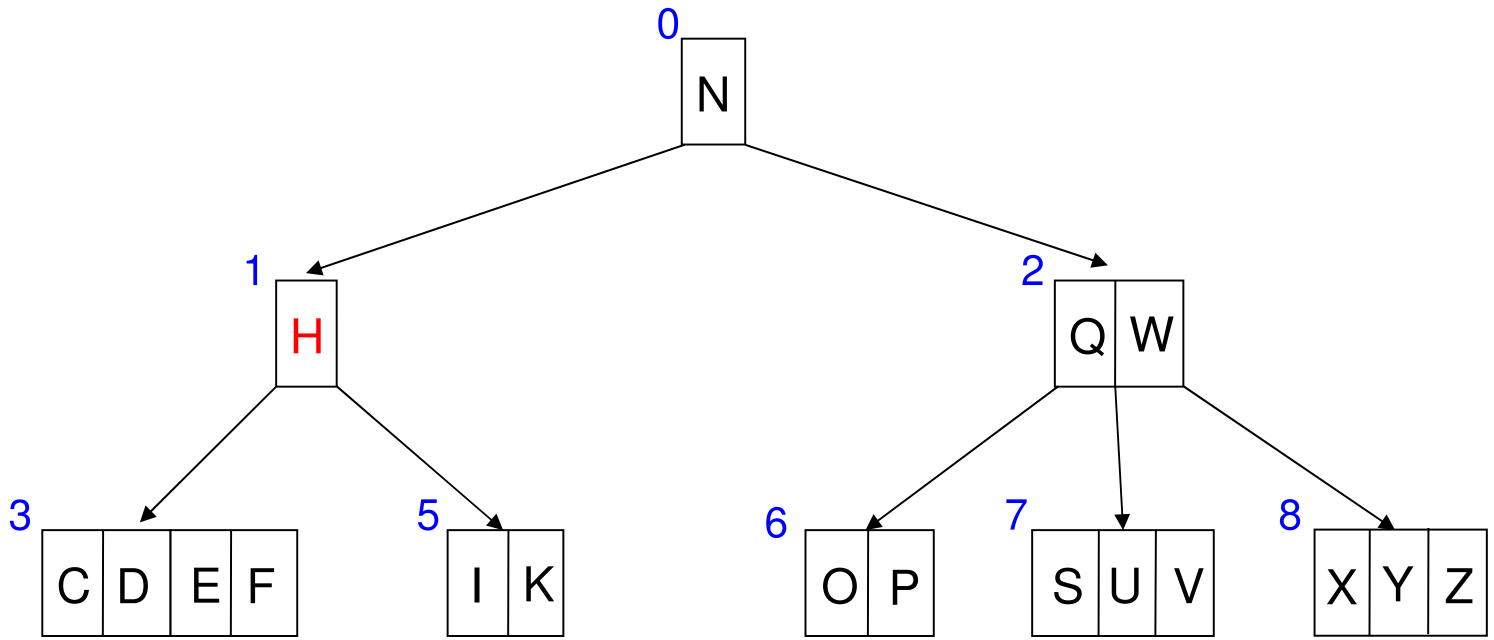
Eliminação: Caso 5

- **Caso 5:** *underflow* da página pai

- **Solução:** utiliza-se redistribuição ou concatenação novamente

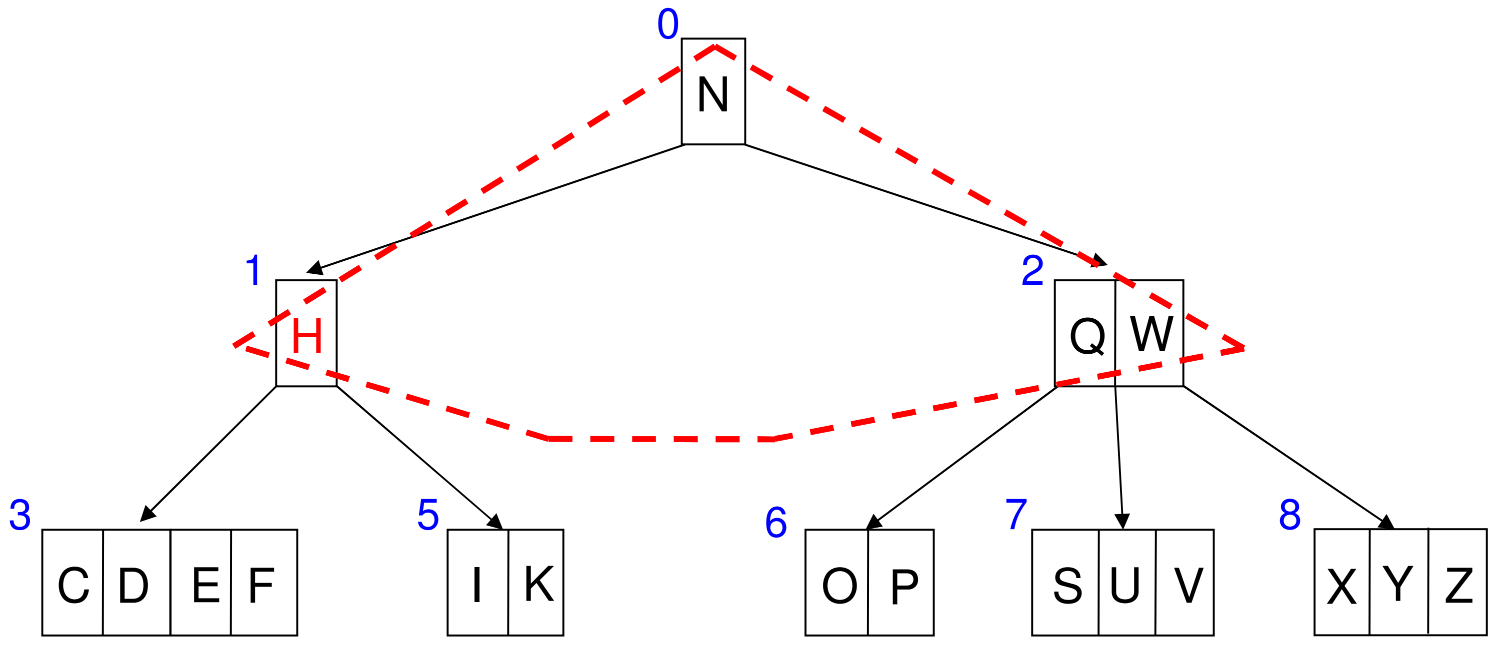
Eliminação: Caso 5

- Propagação do *underflow*
  - O que acontece?



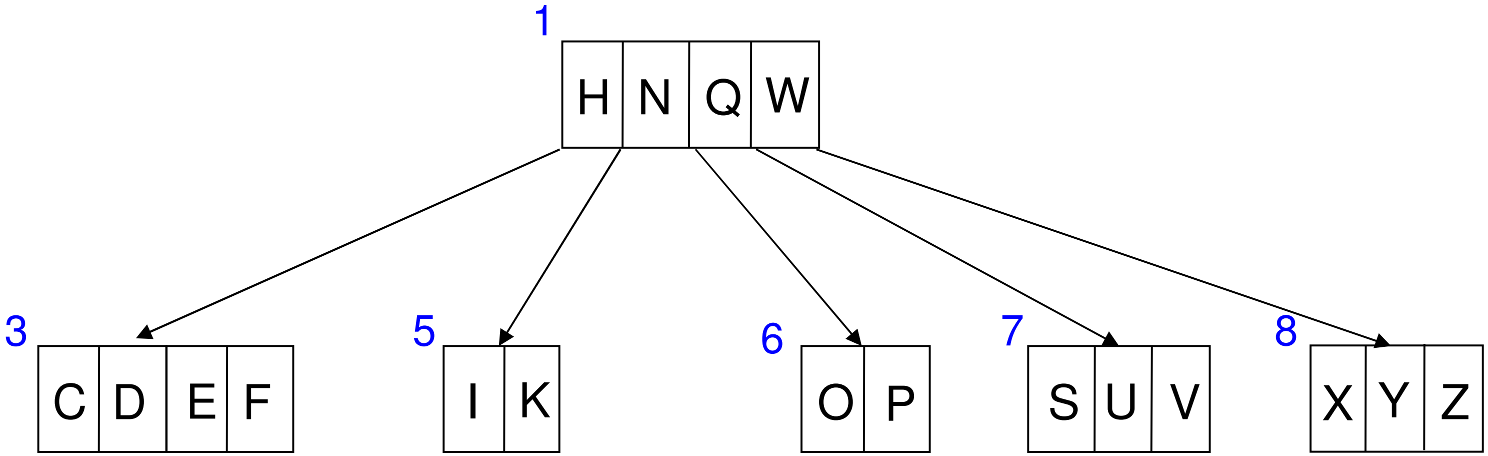
Eliminação: Caso 5

- Propagação do *underflow*
  - Não dá para fazer redistribuição, mas dá para fazer concatenação



Eliminação: Caso 5

- Propagação do *underflow*
  - Não dá para fazer redistribuição, mas dá para fazer concatenação



Eliminação: Caso 6

- **Caso 6:** diminuição da altura da árvore
  - Ocorre quando o nó raiz tem uma única chave e aplica-se a concatenação de seus nós filhos
    - Como ocorreu no exemplo anterior

Eliminação (resumo)

1. Se a chave não estiver numa folha, troque-a com sua sucessora

2. Elimine a chave da folha

3. Se a folha continuar com o número mínimo de chaves, FIM

4. Senão (*underflow*)

4.1. se uma das páginas irmãs diretas (esquerda ou direita) tiver mais que o mínimo de chaves, aplique *redistribuição* e FIM

4.2. senão

a) *concatene* a pág. com uma das irmãs e a chave separadora do nó pai

b) se nó pai for raiz e sua última chave foi rebaixada, *elimine a raiz* e FIM

c) senão, se nó pai continuar com o mínimo de chaves, FIM

d) senão (*underflow* no pai), volte ao item **(a)** para o nó pai

Desempenho de Árvores-B

- Complexidade computacional para **busca**, inserção e remoção de chaves
  - No pior caso a altura é dada pelo maior inteiro **d** tal que: $d \leqslant 1 + \log_{\lceil m/2 \rceil}(N/2)$
  - A altura é $O(\log_{\lceil m/2 \rceil} N)$
  - E portanto no pior caso uma busca requer $O(\log_{\lceil m/2 \rceil} N)$ acessos

Desempenho de Árvores-B

- Complexidade computacional para busca, **inserção** e remoção de chaves
  - Toda inserção realiza busca
  - Além disso pode realizar split
    - Cada split opera sobre um número fixo de páginas e é portanto constante, ou $O(1)$
    - No pior caso, *overflows* se propagam até a raiz e são realizados $O(\log_{\lceil m/2 \rceil} N)$ splits em tempo constante
  - Assim, no pior caso uma inserção também requer $O(\log_{\lceil m/2 \rceil} N)$ acessos

Desempenho de Árvores-B

- Complexidade computacional para busca, **inserção** e **remoção** de chaves
    - Toda remoção realiza busca
    - Além disso pode realizar concatenação/redistribuição
        - Cada concatenação/redistribuição opera sobre um número fixo de páginas e é portanto constante
        - No pior caso, *underflows* se propagam até a raiz e são realizados $O\left(\log_{\lceil m/2 \rceil} N\right)$ operações em tempo constante
    - Assim, no pior caso uma remoção também requer $O\left(\log_{\lceil m/2 \rceil} N\right)$ acessos

Exercício

- Usando o algoritmo anterior, remova as chaves A, B, Q e R da árvore-B de ordem 5 abaixo