GBC053 - Gerenciamento de Bancos de Dados

Aula 11
Indexação multinível e árvores B

Humberto Razente
humberto.razente@ufu.br

Introdução

- A ciência da computação é uma disciplina jovem
  - considere que em 1970 o homem já tinha ido à Lua
  - e não havia uma estrutura de índice dinâmica eficiente (a árvore B ainda não havia sido criada!)
    - atualmente é difícil pensar em um sistema de arquivos que não envolva uma árvore B
- Criação das árvores B
  - Bayer e McCreight (1972). "Organization and Maintenance of Large Ordered Indexes", Acta-Informatica 1:173-189.
  - Em *survey* de 1979, ”*the B-tree is, de facto, the standard organization for indexes in a database system*”

Introdução

- Problema fundamental de manter um índices em armazenamento secundário
  - armazenamento secundário é lento
- Pode ser separado em 2 problemas específicos
  - Busca no índice deve ser mais rápido que busca binária
  - Inserção e remoção devem ser tão rápidos quanto a busca
  - ISAM é apropriada para dados estáticos ou com poucas inserções/remoções

Problemas

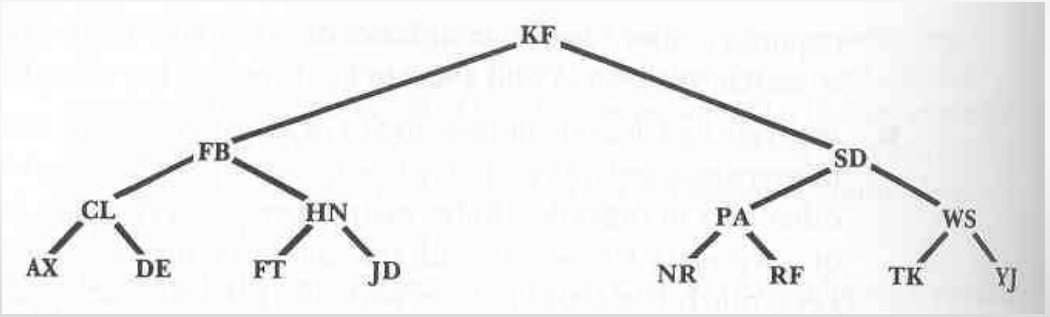
- Busca no índice deve ser mais rápido que busca binária
  - busca por uma chave envolve acessos (seeks) em setores de disco diferentes
  - busca binária: 4 seeks = busca em 15 itens, 9.5 seeks = busca em 1000 itens
- Inserção e remoção devem ser tão rápidos quanto a busca
  - até então, inserções no índice envolviam deslocamentos e a reescrita do índice

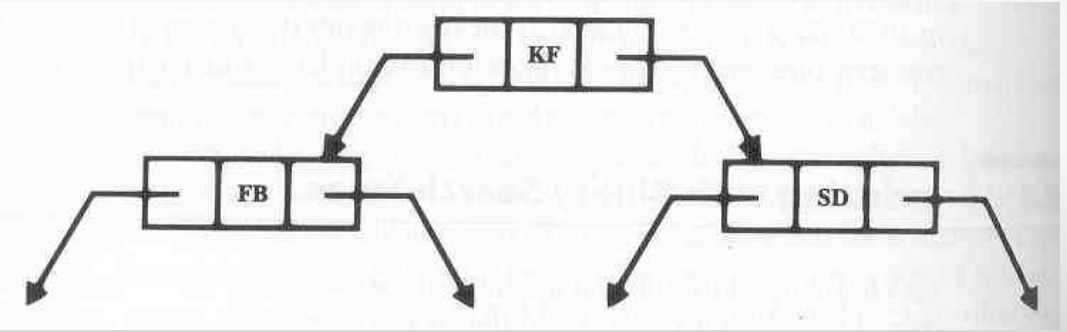
Motivação

- Uma árvore B com capacidade para 100 chaves por página para indexar um arquivo com 1 milhão de registros possibilita achar a chave de qualquer registro com 3 acessos à disco em média
  - $\log_{100} 1.000.000 = 3$
- Busca binária = $\log_2 1.000.000 = 20$ acessos

Indexação com árvore de busca binária

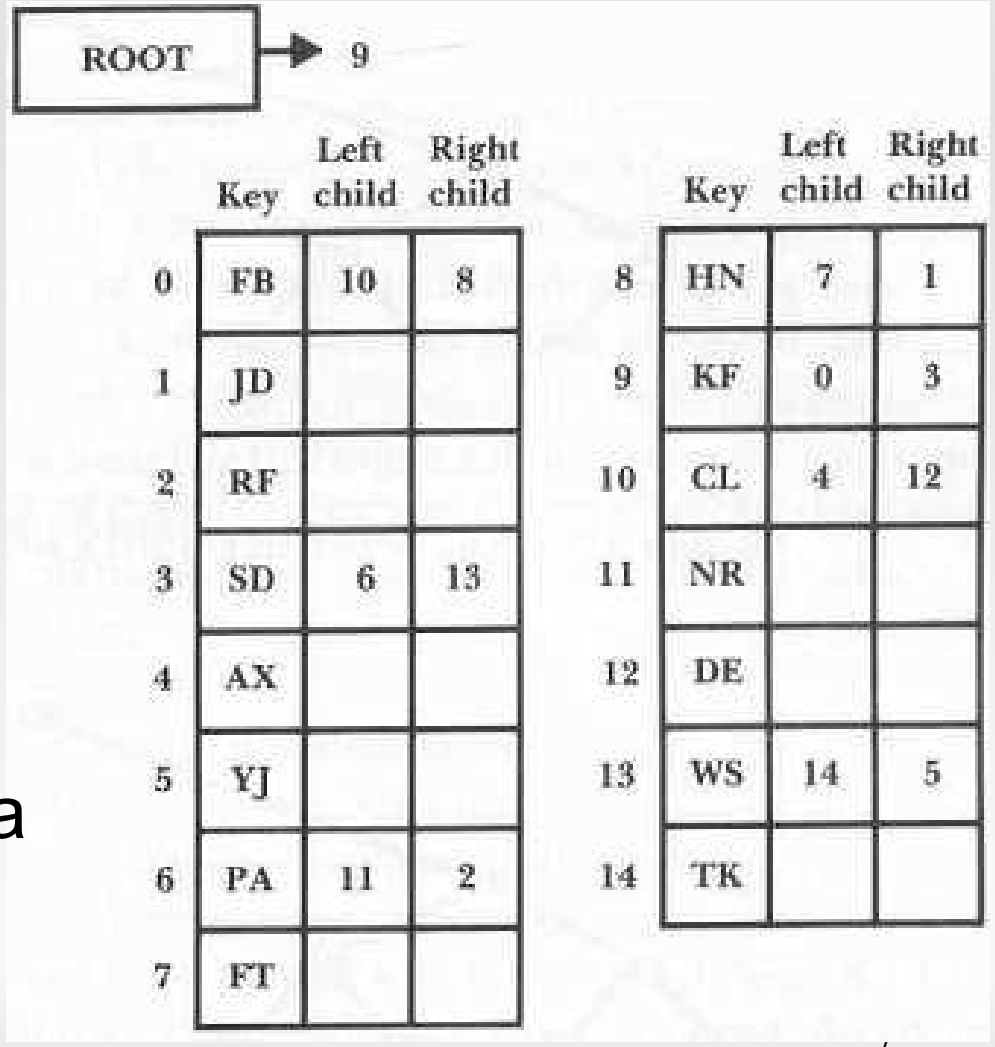
- Uma busca binária em uma lista ordenada pode ser expressa como uma árvore binária de busca
  - nós contêm campos de ponteiros esquerdo e direito
- Problemas: número de seeks
  - falta de estratégia no balanceamento
    - levou ao desenvolvimento da AVL e árvores binárias paginadas





Árvore binária de busca

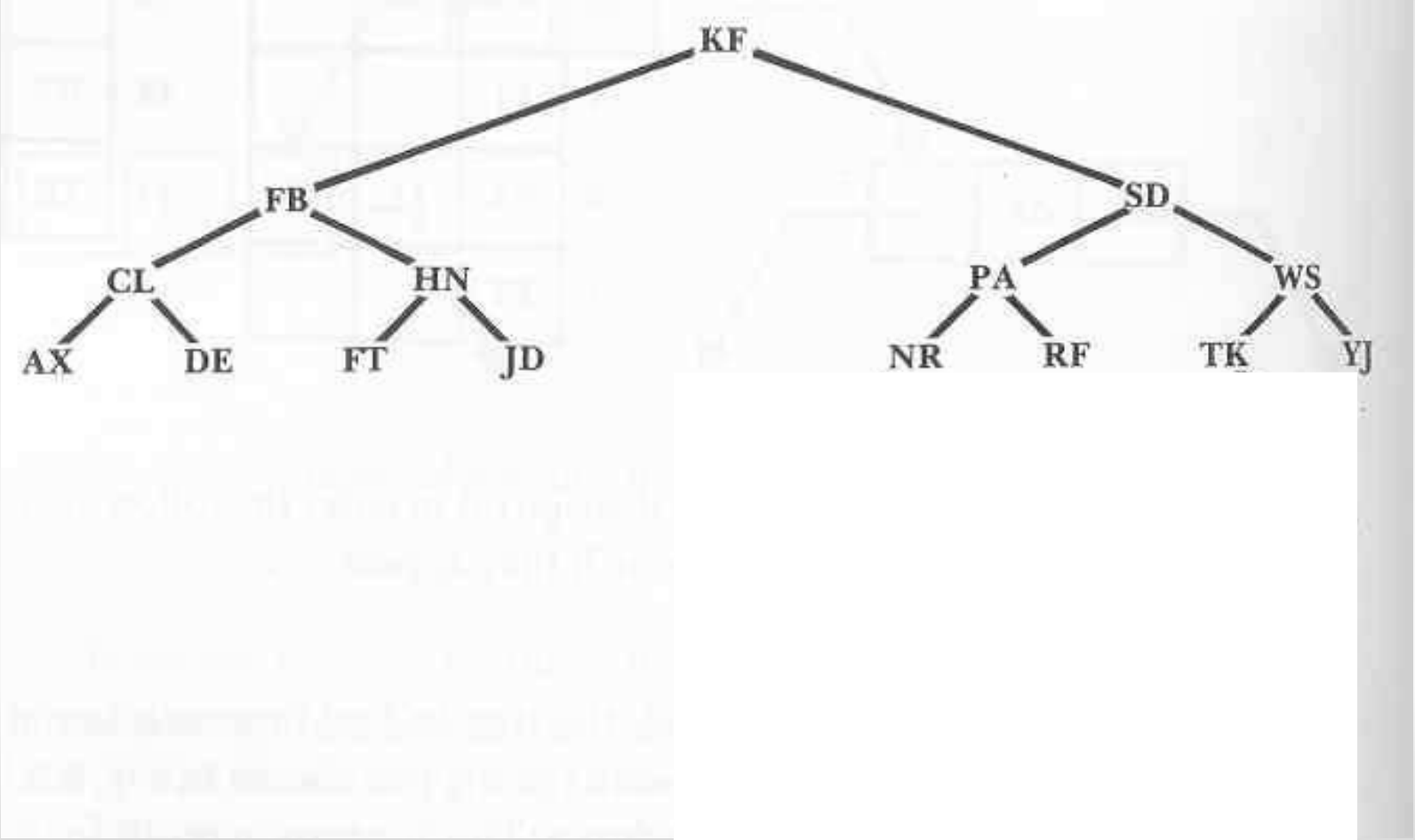
- Lista ordenada
  - pode ser representada como árvore binária de busca
  - não é preciso ordenar arquivo de registros
    - número registro ou offset
  - adição por meio da criação de um novo nó na folha apropriada



	Key	Left child	Right child
0	FB	10	8
1	JD		
2	RF		
3	SD	6	13
4	AX		
5	YJ		
6	PA	11	2
7	FT		
8	HN	7	1
9	KF	0	3
10	CL	4	12
11	NR		
12	DE		
13	WS	14	5
14	TK		

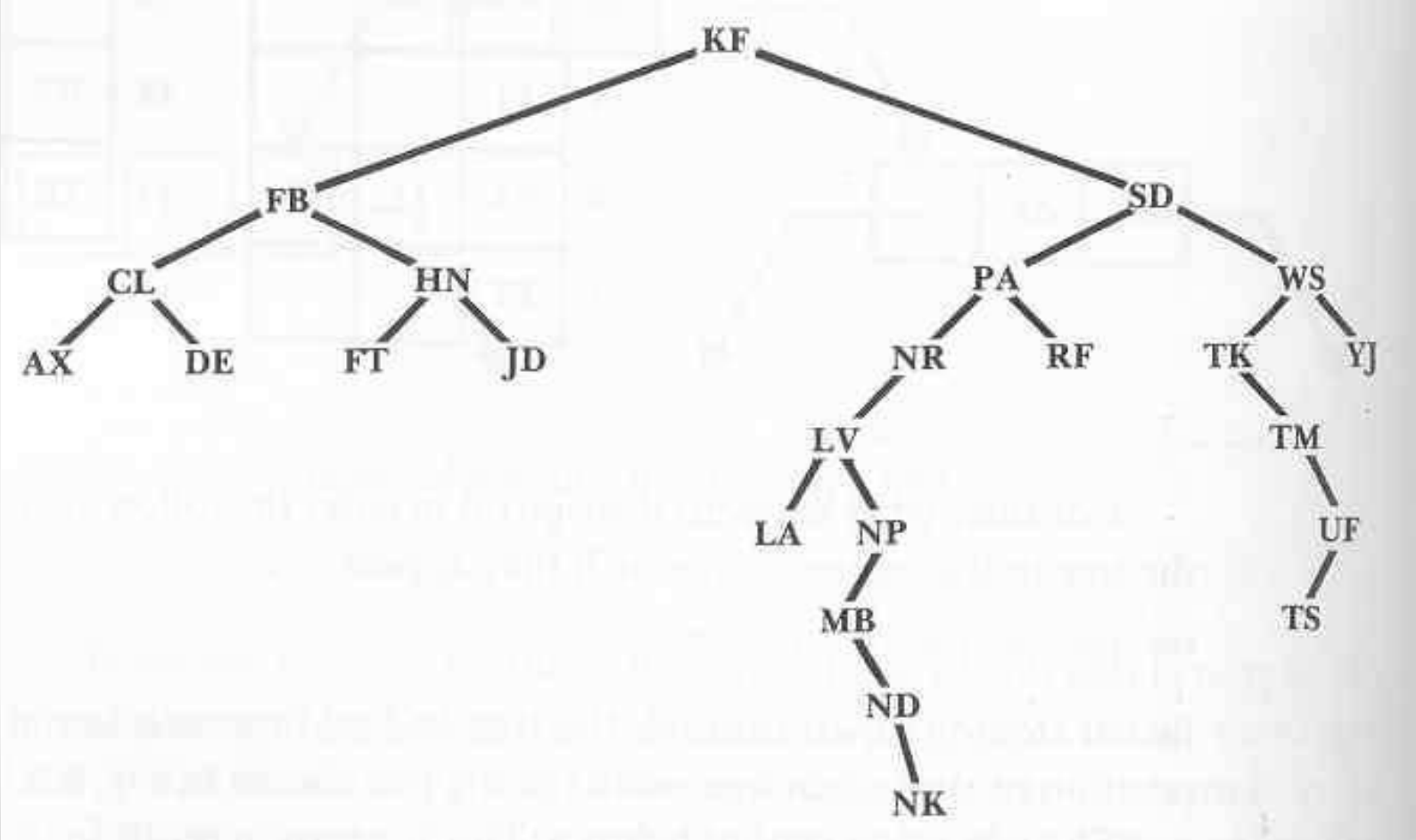
Árvore binária de busca

- Considere a inserção das chaves na seqüência: LV, NP, MB, TM, LA, UF, ND, TS e NK



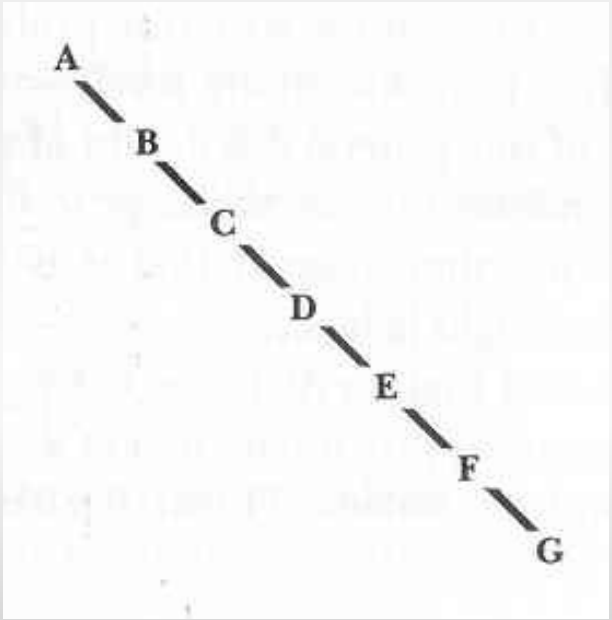
Árvore binária de busca

- Considere a inserção das chaves na seqüência: LV, NP, MB, TM, LA, UF, ND, TS e NK



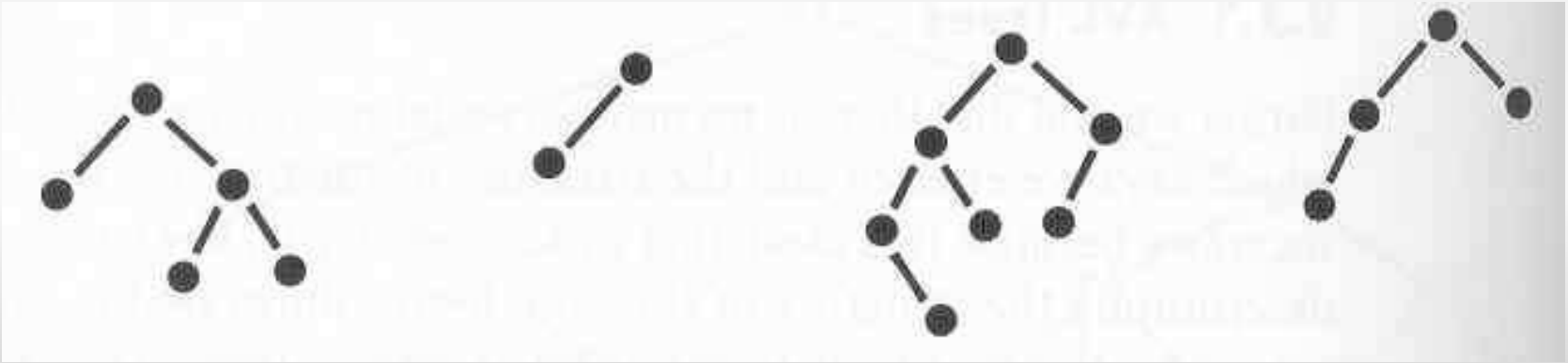
Árvore binária de busca

- Problemas:
  - desbalanceamento
  - embora a árvore tenha evitado a ordenação do arquivo, há nós que precisam de até 9 comparações
- Pior caso para árvore binária
  - lista ligada



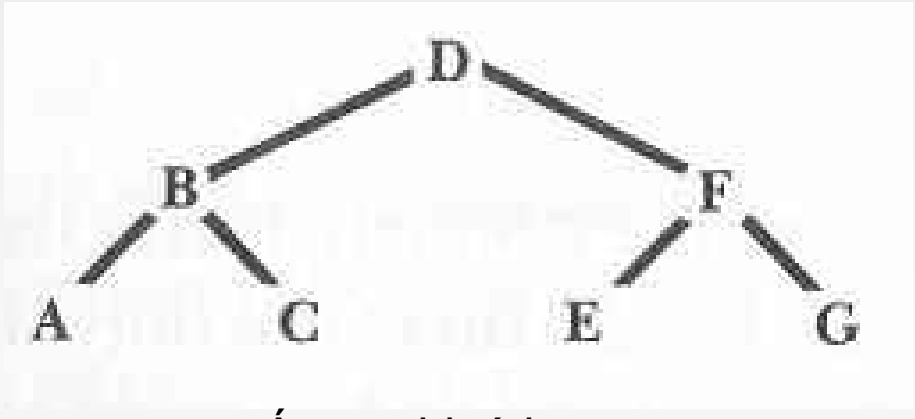
Árvore AVL

- Solução: reorganizar nós ao inserir novas chaves
  - mantendo próximo da estrutura ótima
  - AVL: homenagem aos matemáticos G. M. **A**del'son-**V**el'skii e E. M. **L**andis
    - AVL: é uma árvore balanceada pela altura
    - diferença máxima permitida entre altura das folhas é 1

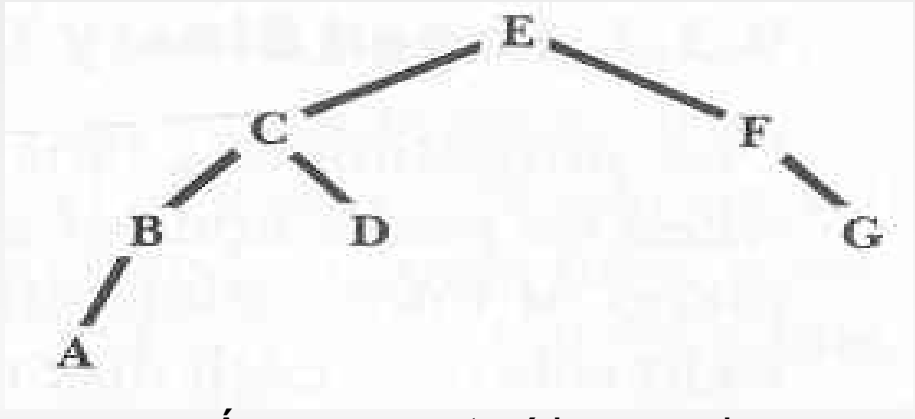


Árvore AVL

- Desempenho aproximado ao de uma árvore completamente balanceada:
  - AVL: pior caso entre n chaves: $1.44 \log_2 (n+2)$
  - Completamente balanceada: pior caso $\log_2 (n+1)$
- Sequência: B C G E F D A



Árvore binária completamente balanceada



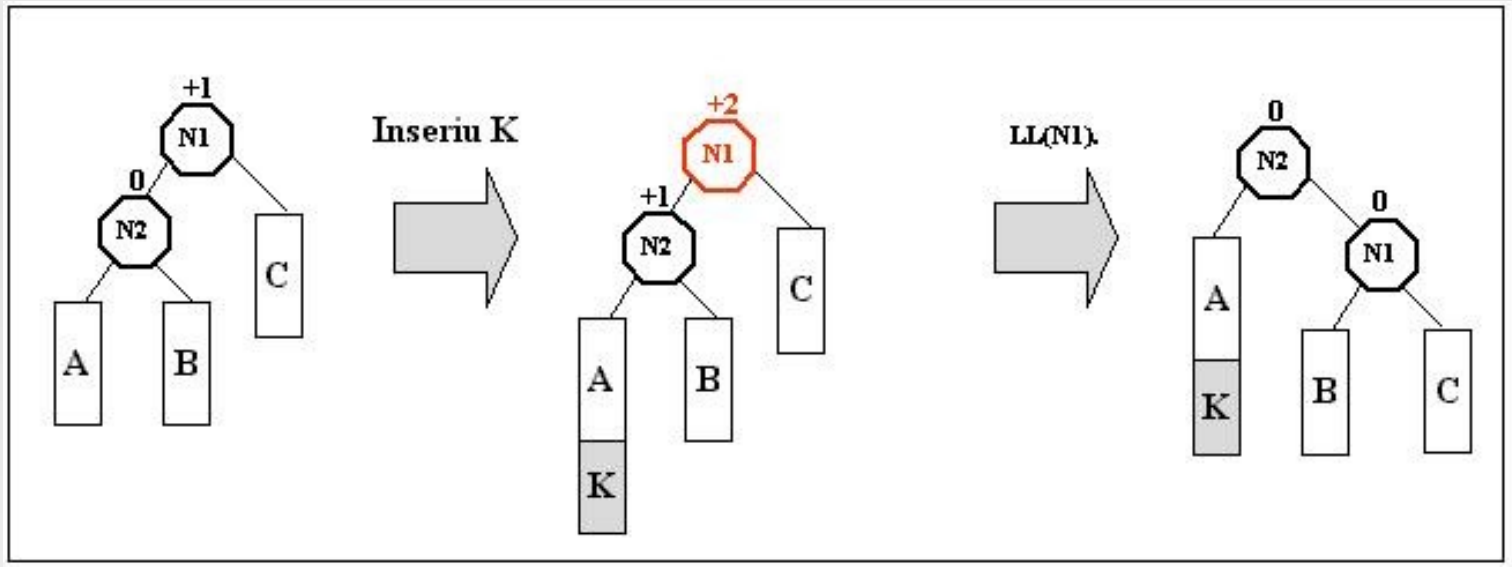
Árvore construída usando procedimentos da AVL

Árvore AVL

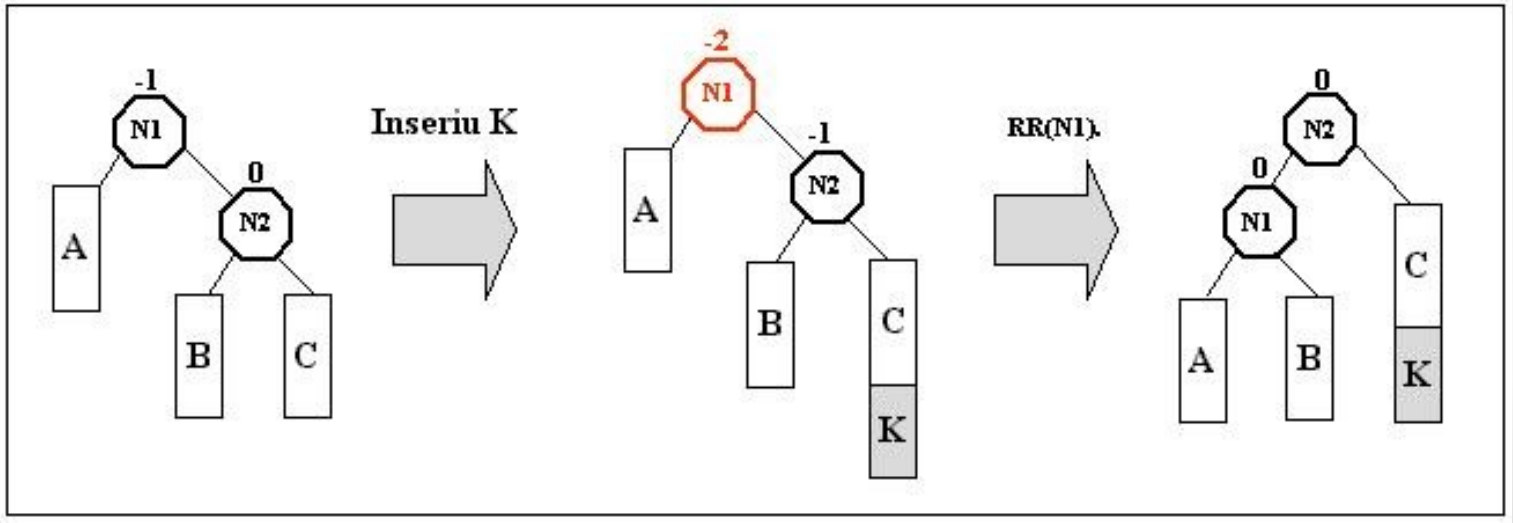
- Construção baseada em rotações
  - Rotação (LL): O novo nó X é inserido na sub-árvore da esquerda do filho esquerdo de A
  - Rotação (LR): X é inserido na sub-árvore da direita do filho esquerdo de A
  - Rotação (RR): X é inserido na sub-árvore da direita do filho direito de A
  - Rotação (RL): X é inserido na sub-árvore da esquerda do filho direito de A

Árvore AVL

Rotação LL

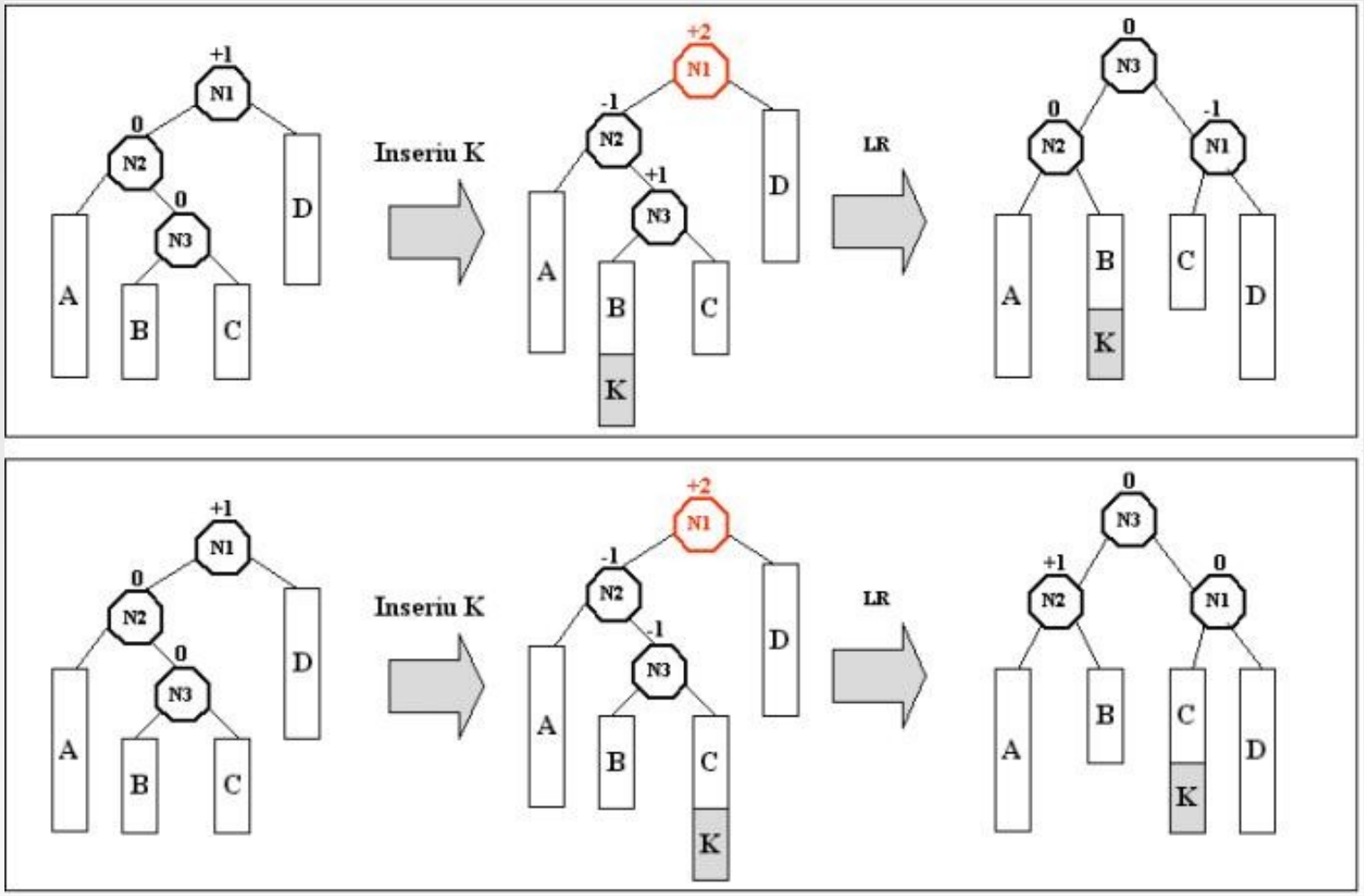


Rotação RR



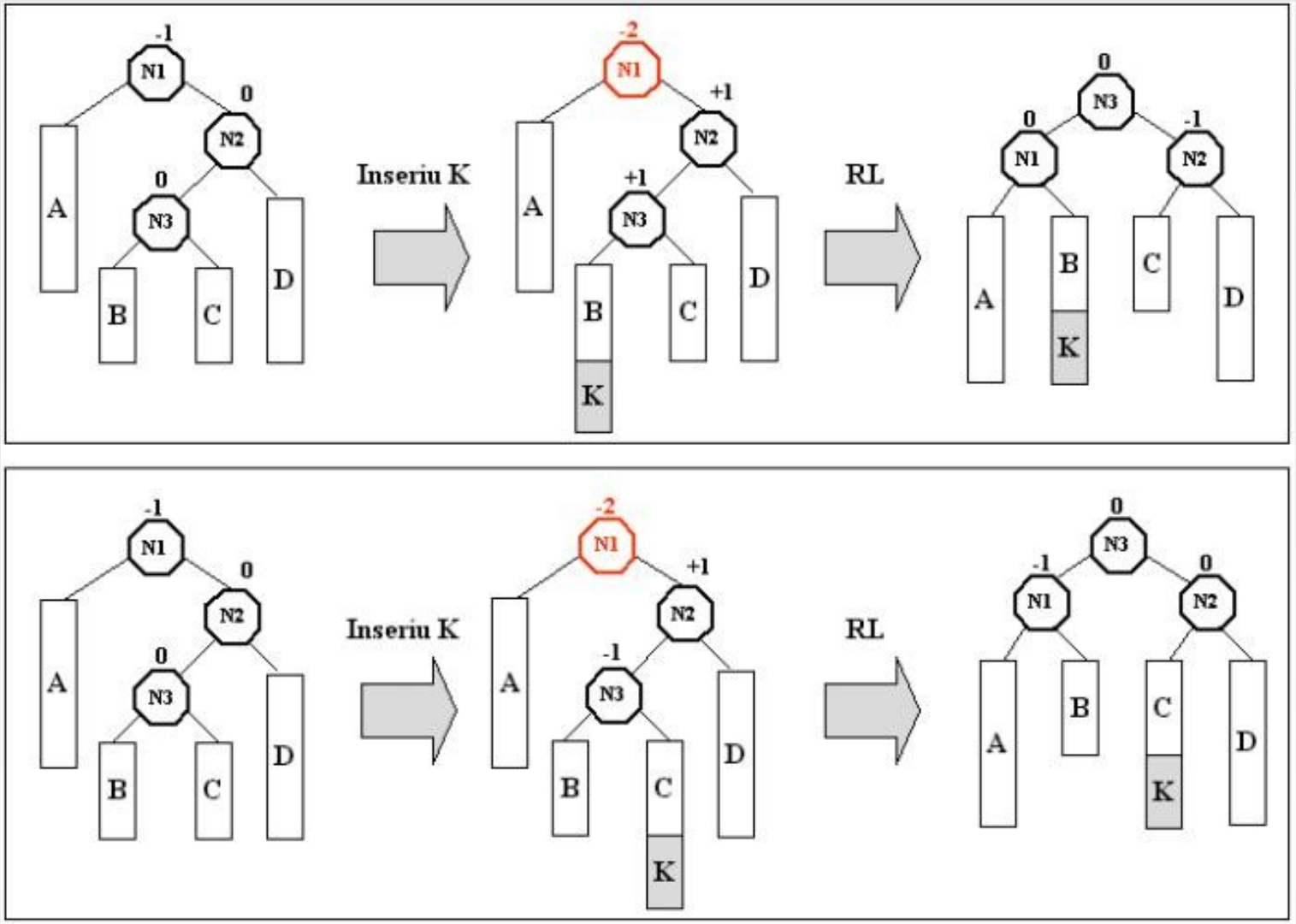
Árvore AVL

Rotação LR



Árvore AVL

Rotação RL



Árvores binárias paginadas

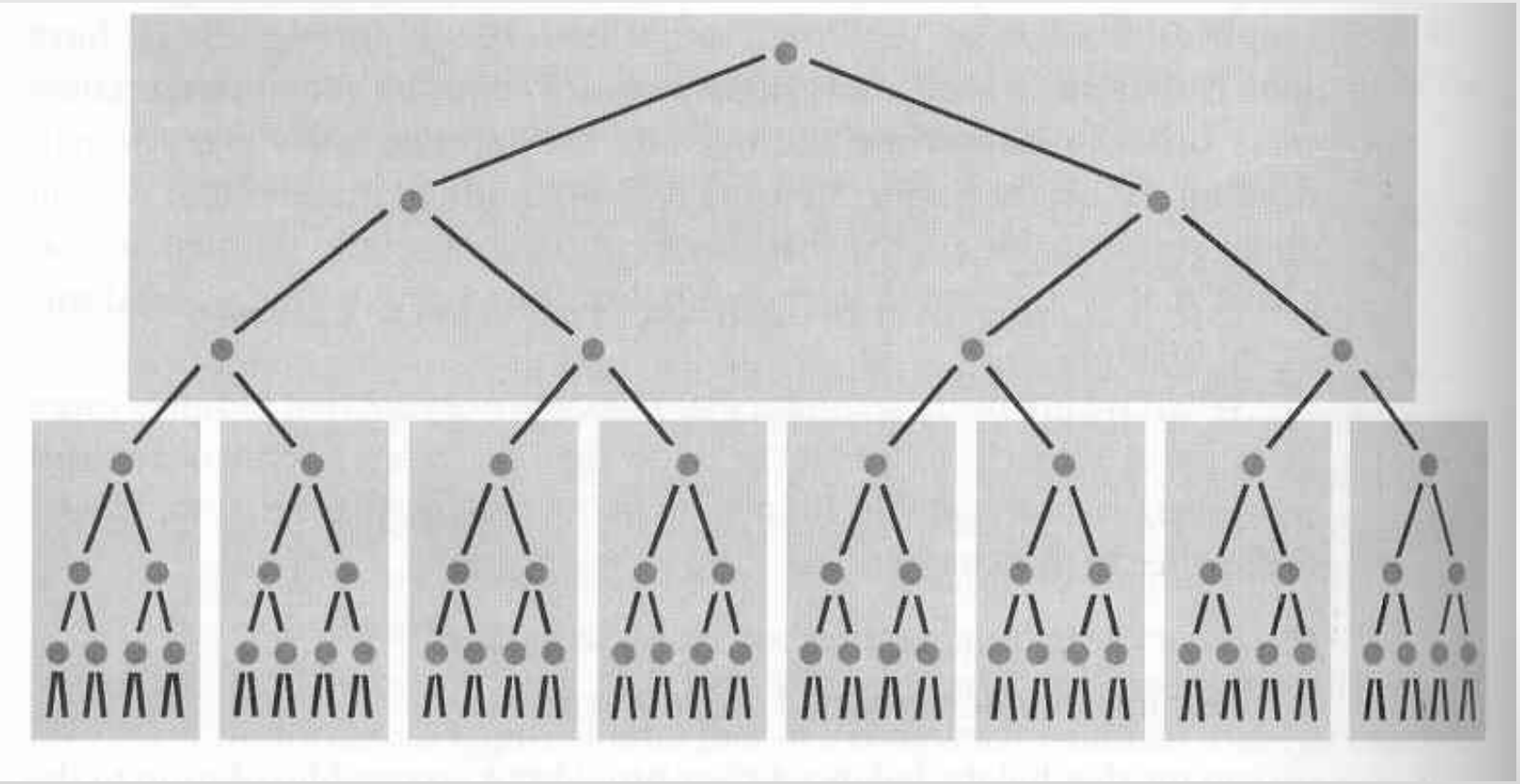
- A utilização do disco nas árvores de busca binárias é extremamente ineficiente
  - Uso de um registro de índice que use todo o espaço alocado no disco
- Árvore binária paginada
  - Vários nós na mesma página do disco
    - Uma página de disco, que pode conter vários registros, é lida em um único acesso
    - Se a informação que é necessária estiver na página lida, economiza-se um acesso ao disco

Árvores binárias paginadas

- Em um sistema paginado
  - não se ”paga” o custo de ler uma página para recuperar uns poucos bytes
  - uma vez que faz-se um seek para uma página
    - objetivo é ler a página inteira
    - objetivo é ter nessa página o máximo de registros
    - se o próximo byte necessário para a aplicação está nessa página, então economizou-se um acesso ao disco

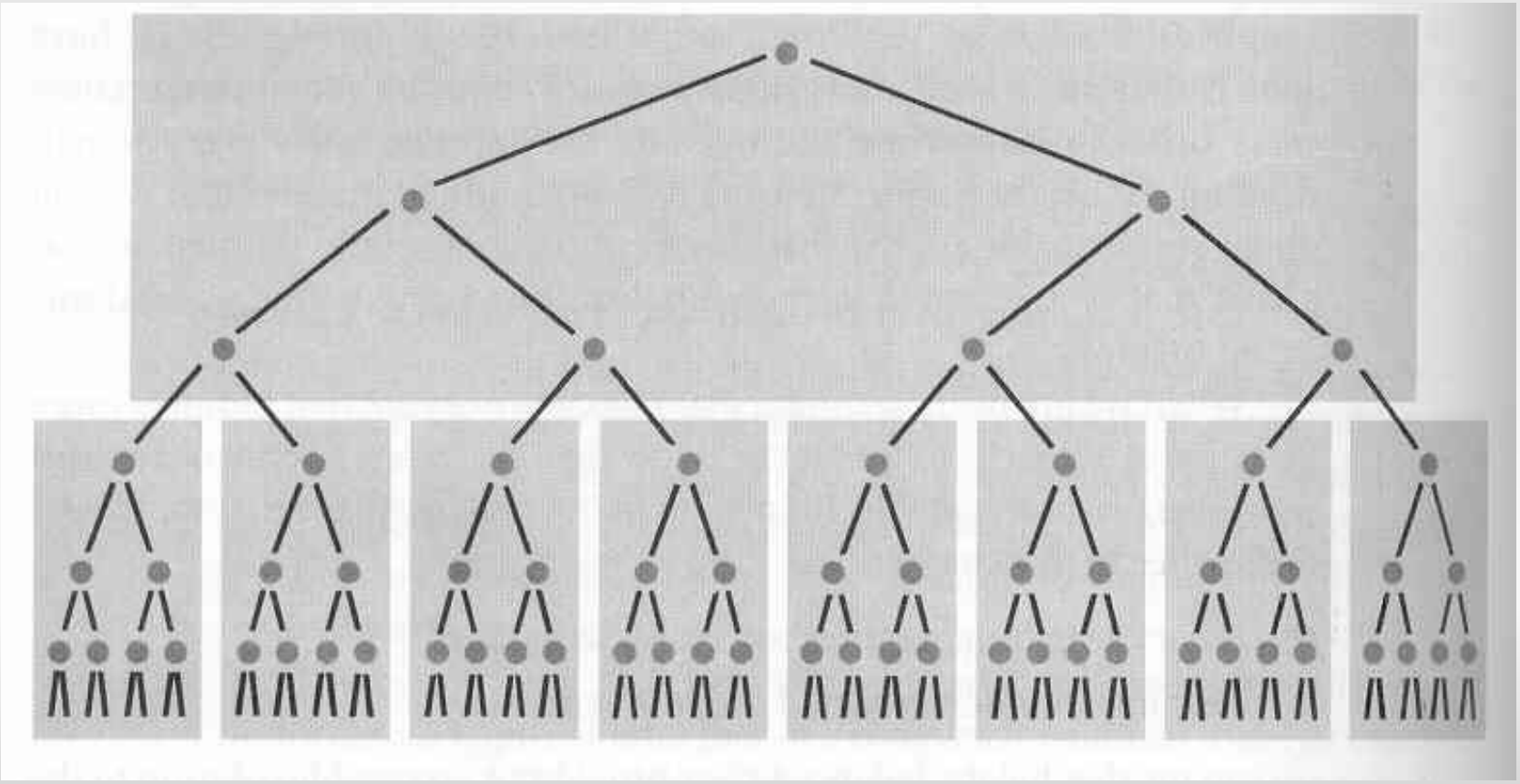
Árvores binárias paginadas

- Divisão de uma árvore binária em páginas
  - com o armazenamento das páginas em blocos contíguos no disco, é possível reduzir o número de buscas no disco



Árvores binárias paginadas

- Árvore de 63 nós e tamanho de página de 7 nós
  - mais um nível: acesso a 511 nós com 3 acessos
  - mais outro nível: acesso a 4.095 nós com 4 acessos
  - busca binária em 4.096 elementos pode ter 12 acessos



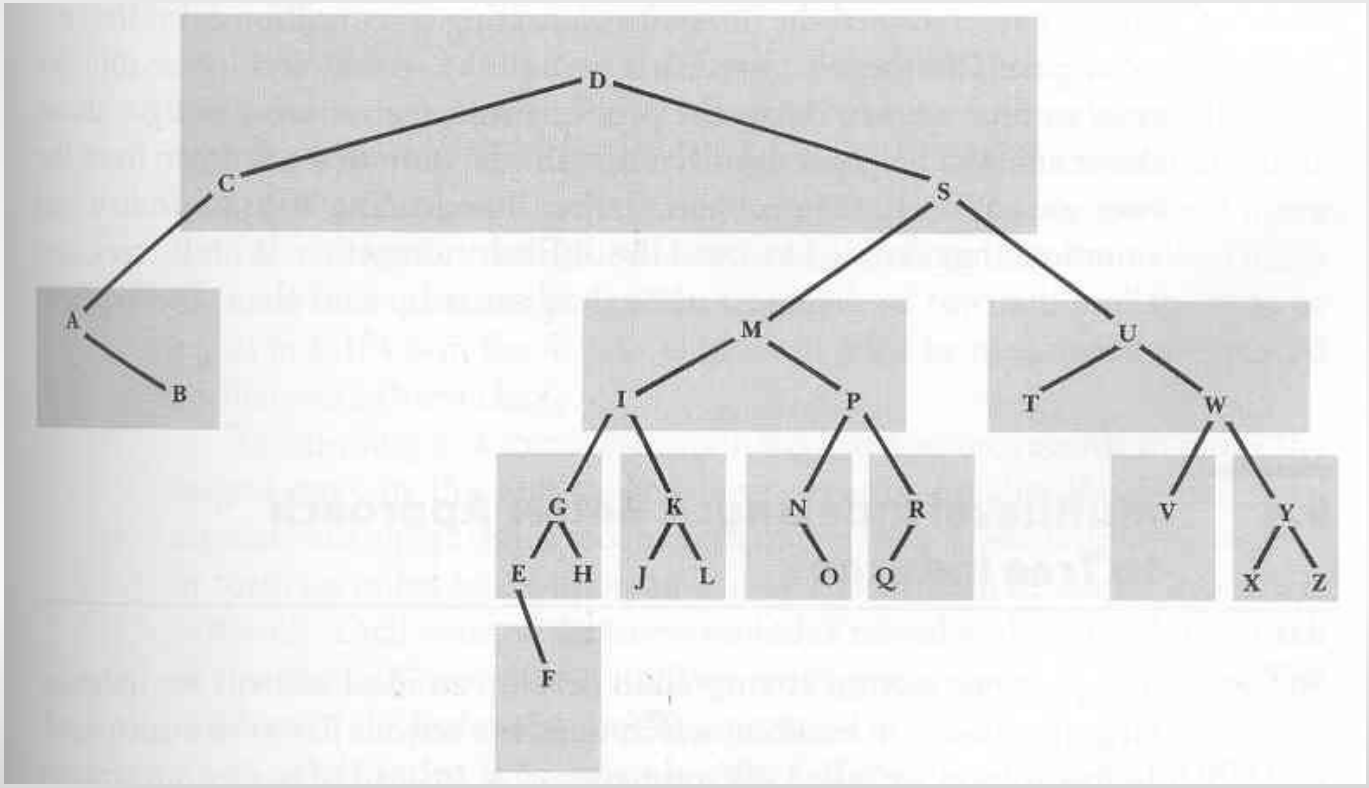
Árvores binárias paginadas

- Página típica de 8 KB
  - 512 pares: chave (8 bytes) + referência (8 bytes)
  - se armazenar uma árvores completamente balanceada, é possível encontrar uma chave entre 134.217.728 chaves com 3 acessos a disco (512^3)
    - isso é um comportamento que se almeja!

Árvores binárias paginadas

- Problemas
  - se tem-se as chaves previamente, pode-se construir a árvore com os elementos ordenados
  - entretanto, se chaves são inseridas aleatoriamente, chaves podem forçar o balanceamento da árvore

balanceamento pode forçar re-escrita de muitas páginas



Árvores binárias paginadas

- Questões não-resolvidas:
  - Como garantir que as chaves na página raiz sejam boas separadoras, dividindo o conjunto das outras chaves mais ou menos igualmente?
  - Como evitar que chaves muito distantes compartilhem a mesma página?
  - Como garantir que cada página contém um número mínimo de chaves?
  - Solução: construção da árvore de baixo para cima
    - A raiz emerge do conjunto de chaves

ISAM

- *Estrutura ISAM*
  - *Indexed Sequential Access Method (ISAM)*

- *Fundamentos:*
  - *Inspiração em arquivo ordenado:*
    - *Motivação 1: reduzir tamanho da busca binária*
    - *Motivação 2: facilitar inserções e remoções*
  - *Eficiência em busca de intervalo, varredura ordenada, inserção e remoção*
  - *Eficiência em busca com igualdade, embora inferior a Hash*

ISAM

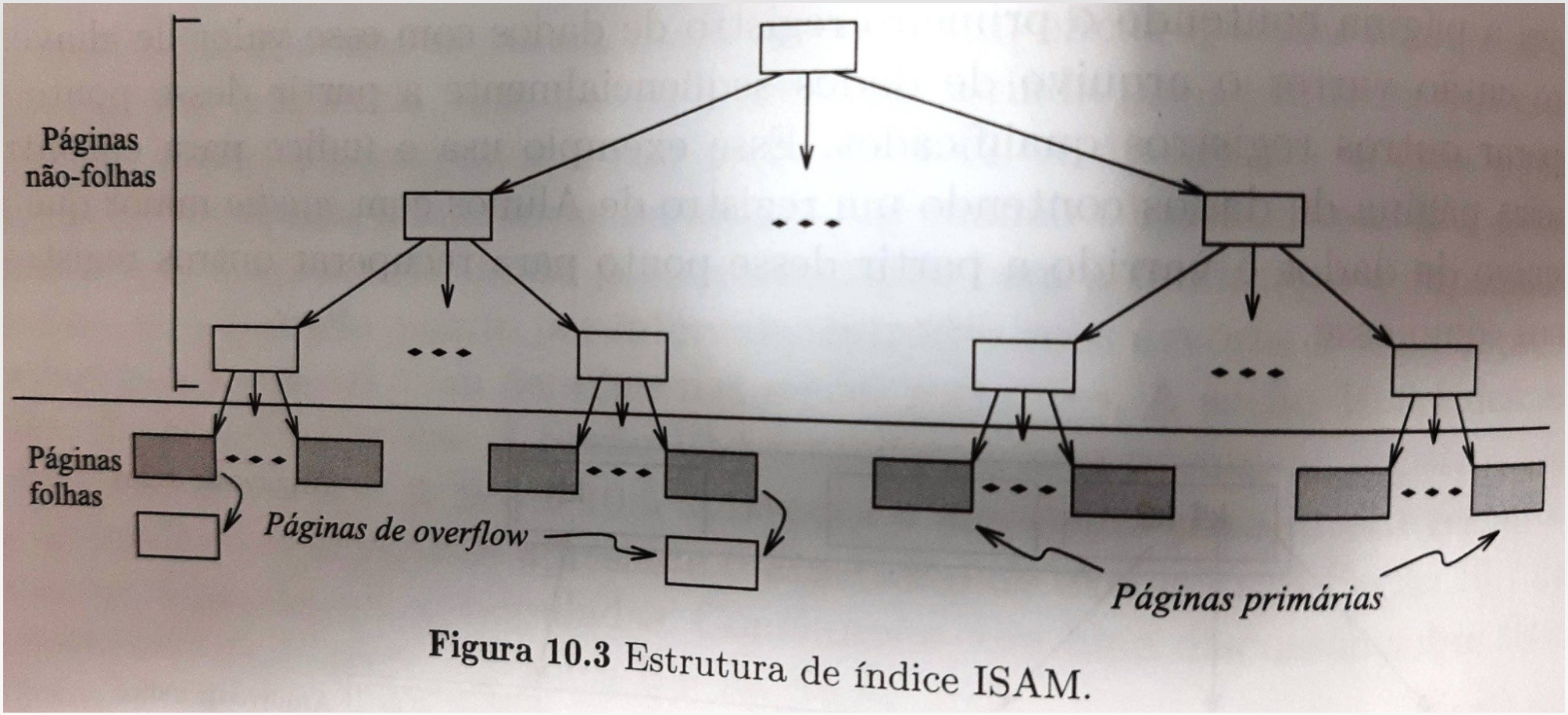


Figura 10.3 Estrutura de índice ISAM.

Árvores B

- Uma árvore B é um índice multinível que resolve o problema do custo linear da inserção e remoção
  - não requer que páginas índice sejam completas
  - não extende overflow para próxima página
    - overflow gera divisão da página em duas, cada uma com metade dos registros
  - remoção tem estratégia semelhante
    - pode resultar na junção de páginas quando necessário

Árvores B

- Cada nó de uma árvore B é um índice
- Ordem da árvore
  - número máximo de chaves de um nó
- Número mínimo de chaves por nó
  - metade da ordem da árvore
  - exceção: raiz, mínimo de 2 chaves

Árvores B

- Inserção
  - Basta atualizar o registro de índices
  - Se a nova chave é a nova maior chave no registro de índices, ela se torna a chave de maior nível naquele registro e o índice do nível superior deve ser atualizado
  - Custo é limitado pela altura da árvore

Árvores B

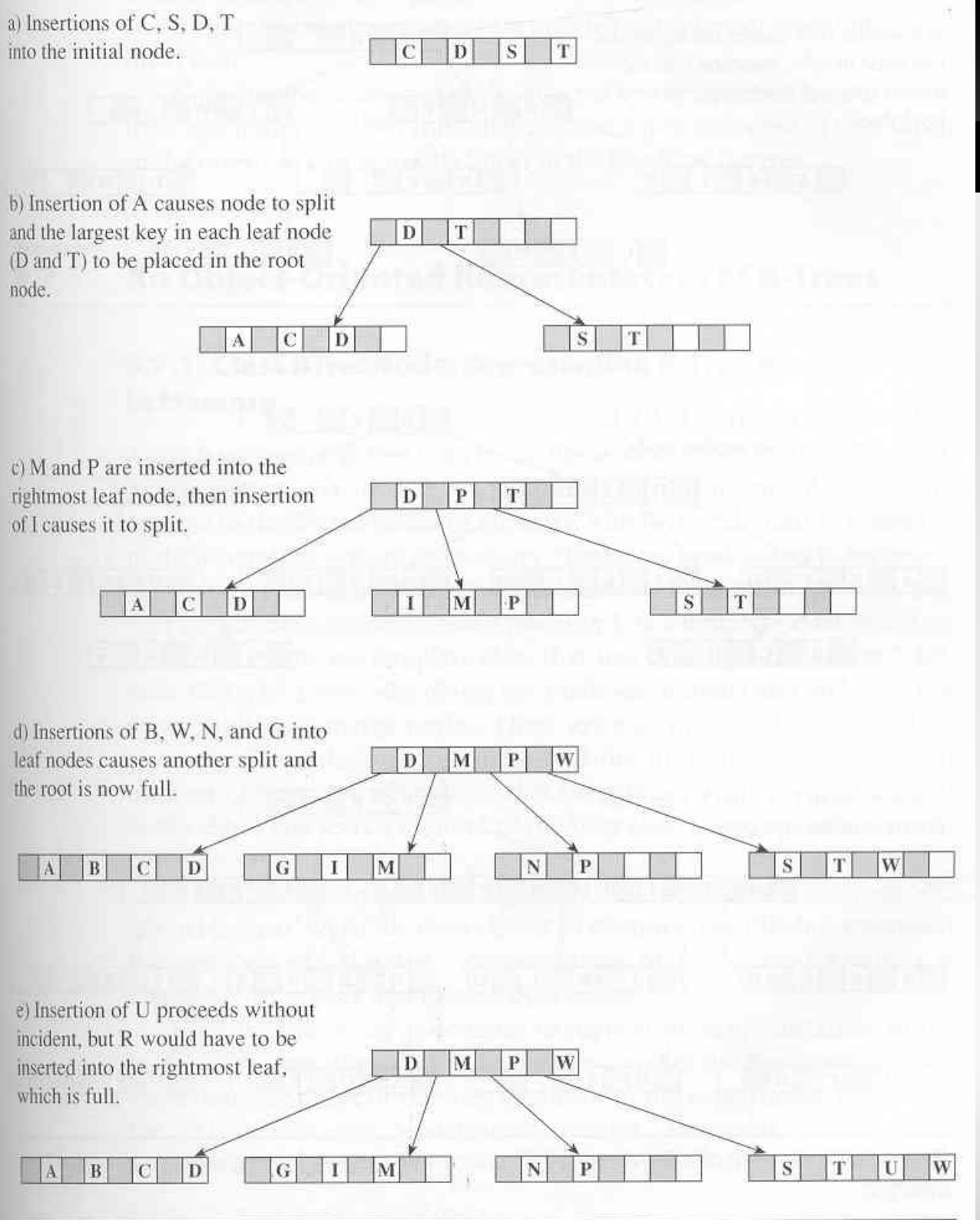
- Inserção
  - Se página estiver cheia:
    - divide-se em 2 páginas, cada uma com metade das chaves
    - como um novo nó de registro foi criado nesse nível, a maior chave desse novo nó deve ser inserida no nó de nível superior → promoção da chave
      - promoção também pode causar um overflow naquele nível, que se propaga pelos níveis superiores recursivamente
      - custo limitado pela altura da árvore

Árvores B

- Exemplo:
  - inserir 26 chaves: C S D T A M P I B W N G U R K E H O L J Y Q Z F X V
  - em uma árvore B de ordem 4

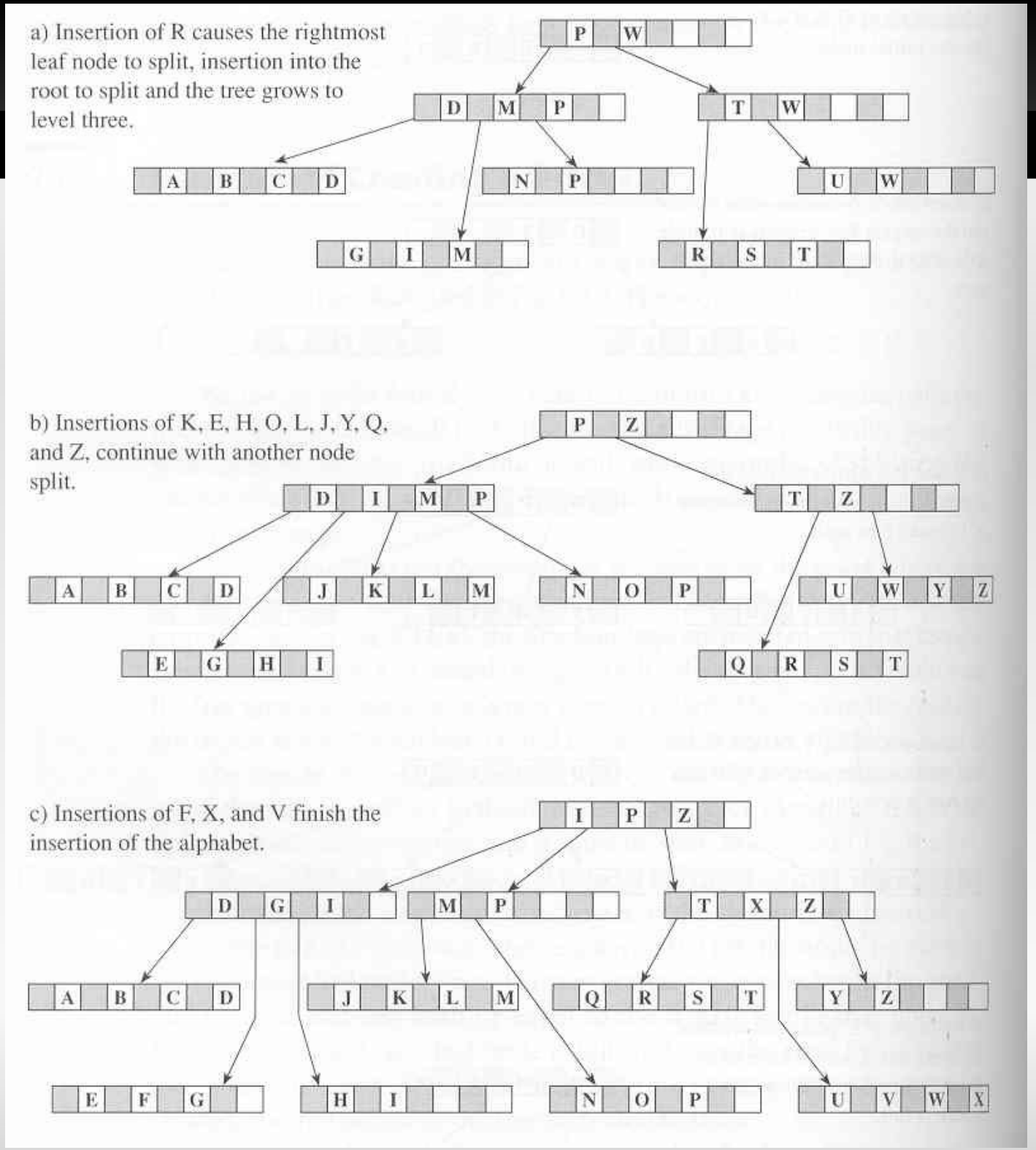
Árvores B

- Chaves:
  C S D T
  A M P I
  B W N
  G U R K
  E H O L
  J Y Q Z
  F X V



Árvores B

- Chaves:
  C S D T
  A M P I
  B W N
  G U R K
  E H O L
  J Y Q Z
  F X V



Árvores B – Algoritmos

- Busca
  - A busca é iterativa
    - carrega-se a página na memória, pesquisa-se na página → procura pela chave em níveis sucessivamente mais baixos até que se alcance o nível das folhas

Árvores B – Algoritmos

- Inserção, divisão de nó e promoção
  - Inicia com uma busca que vai até o nível das folhas
  - Depois de descobrir o local da inserção no nível das folhas, o trabalho de inserção, detecção de overflow e divisão se propaga para cima
    - Pode ser necessário criar um novo nó raiz, caso o raiz atual seja dividida, aumentando a altura da árvore

Árvores B – Propriedades

- Cada página tem um máximo de *m* descendentes
- Cada página, exceto a raiz e as folhas, tem ao menos $\lceil m/2 \rceil$ descendentes
- A raiz tem ao menos 2 descendentes (exceto se for uma folha)
- Todas as folhas estão no mesmo nível
- As folhas formam um índice ordenado completo dos dados associados no arquivo

Árvores B – Análise busca

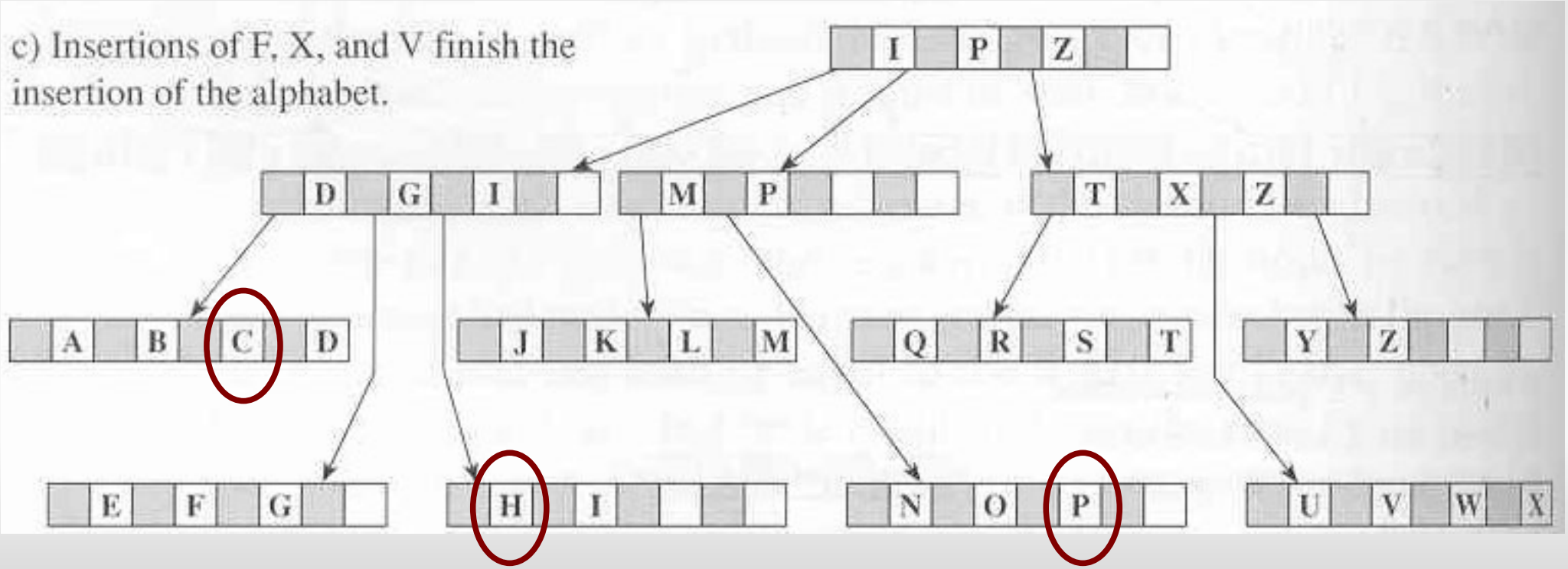
- Qual é o número máximo de acessos a disco para encontrar uma chave em uma árvore B de ordem m no pior caso?
  - Qual a profundidade da árvore?
    - todas as chaves aparecem no nível das folhas
  - Pior caso: quando todas as páginas da árvore têm somente o número mínimo de descendentes
    - Raiz (nível 1) = 2 descendentes
    - Nível 2 = $2\lceil m/2 \rceil$
    - Nível 3 = $2\lceil m/2 \rceil \lceil m/2 \rceil = 2\lceil m/2 \rceil^2$
    - Nível d = $2\lceil m/2 \rceil^{(d-1)}$ descendentes

Árvores B

- Para uma árvore com n chaves em suas folhas
  - $n \geq 2 \lceil m/2 \rceil^{(d-1)}$
- Resolvendo para d
  - $d \leq 1 + \log_{\lceil m/2 \rceil} (n/2)$
- Árvore de ordem 512 com 1 milhão de chaves
  - $d \leq 1 + \log_{\lceil 256 \rceil} 500.000 \approx 3.37$
- Pode-se dizer que dados 1 milhão de chaves, uma árvore B de ordem 512 tem profundidade de não mais de 3 níveis

Árvores B – Remoção, junção de páginas e redistribuição

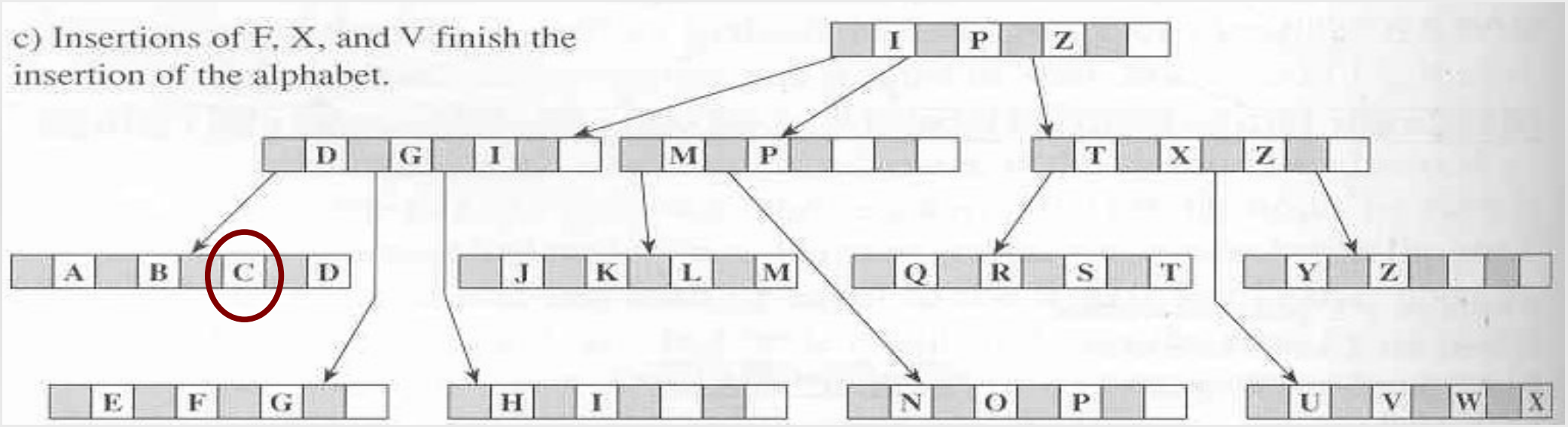
- Considere a árvore B gerada anteriormente
  - remover chave C
  - remover chave P
  - remover chave H

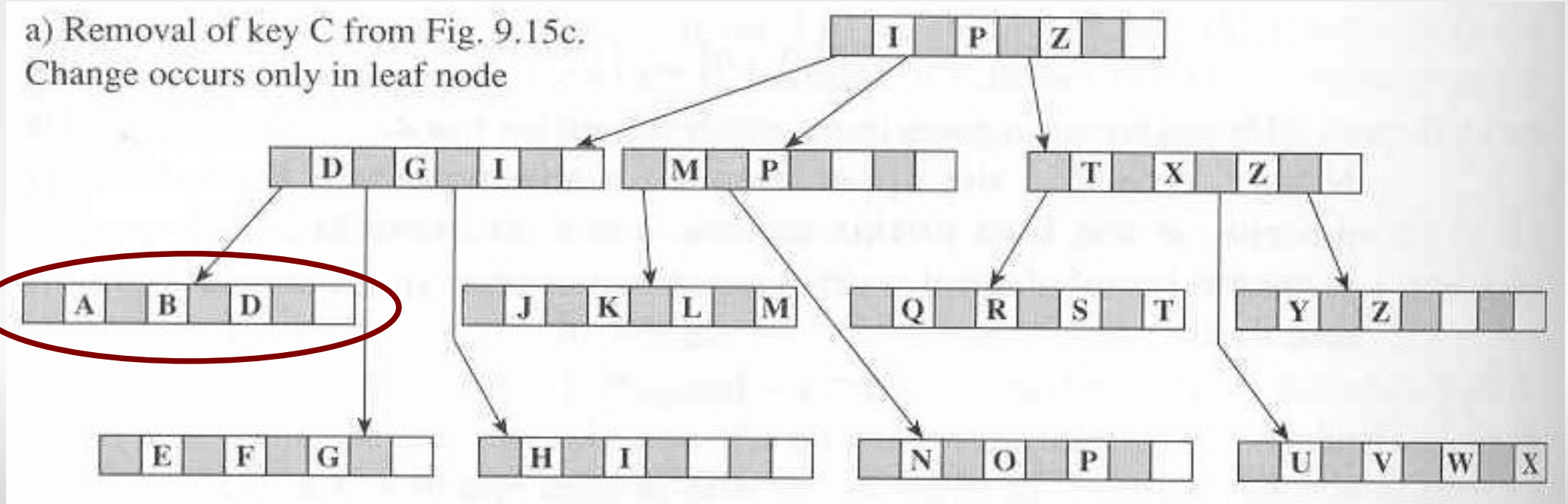


Árvores B – Remoção, junção de páginas e redistribuição

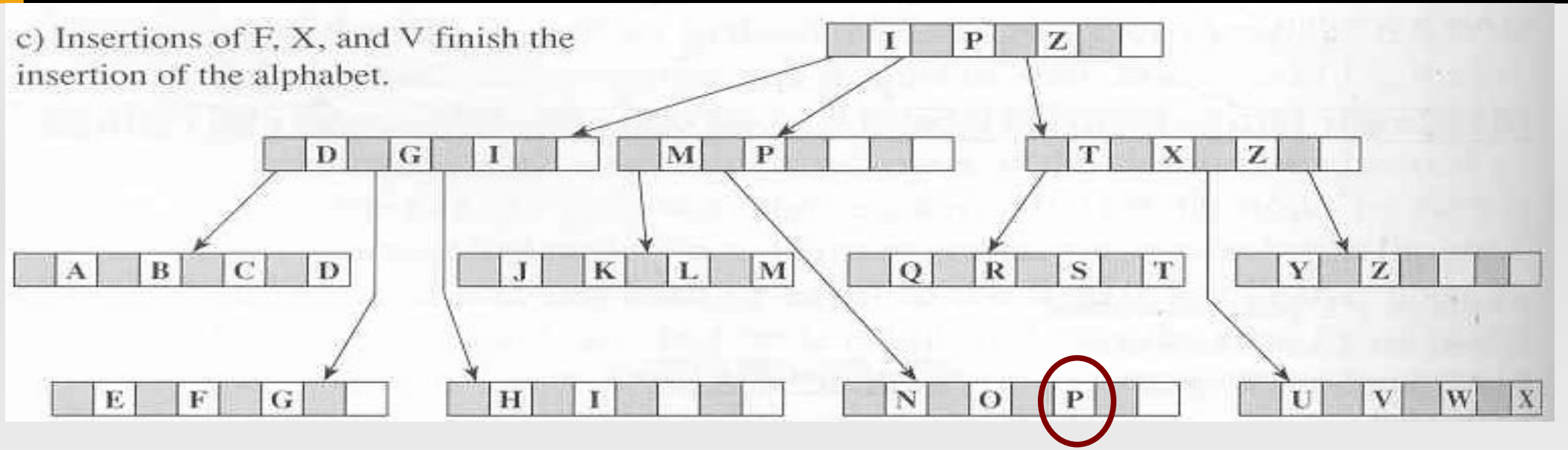
- Remoção das chaves podem resultar em:
  - (a) Remoção da chave C
    - Não causa underflow no nó nem muda o seu maior valor de chave → apenas remover a chave do nó
  - (b) Remoção da chave P
    - Não causa underflow, mas muda a maior chave do nó → o nó de segundo nível deve ser alterado. A chave da segunda folha se torna O e o segundo nível é modificado para conter O no lugar de P. Como P era a maior chave no segundo nível, o processo deve ser repetido no nó raiz
  - (c) Remoção de H
    - Causa underflow na terceira folha. Após a remoção, a chave I é inserida em um nó vizinho e a terceira folha é removida. O nó de segundo nível é modificado para refletir o estado corrente da folha

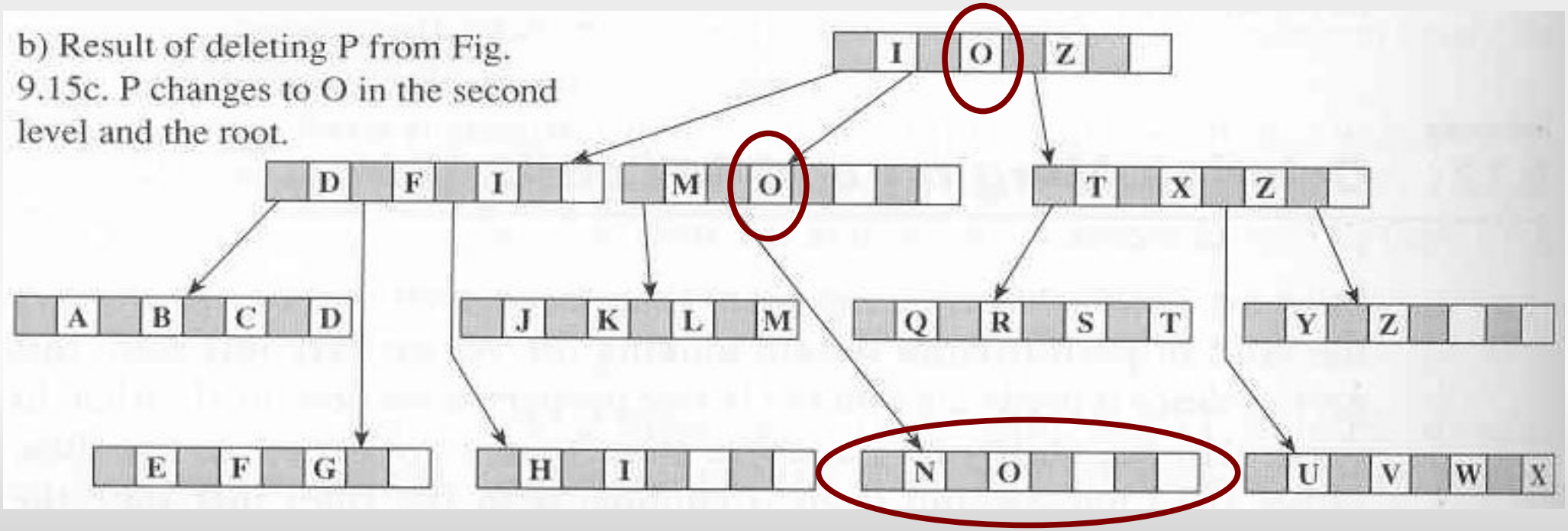
Remoção da chave C



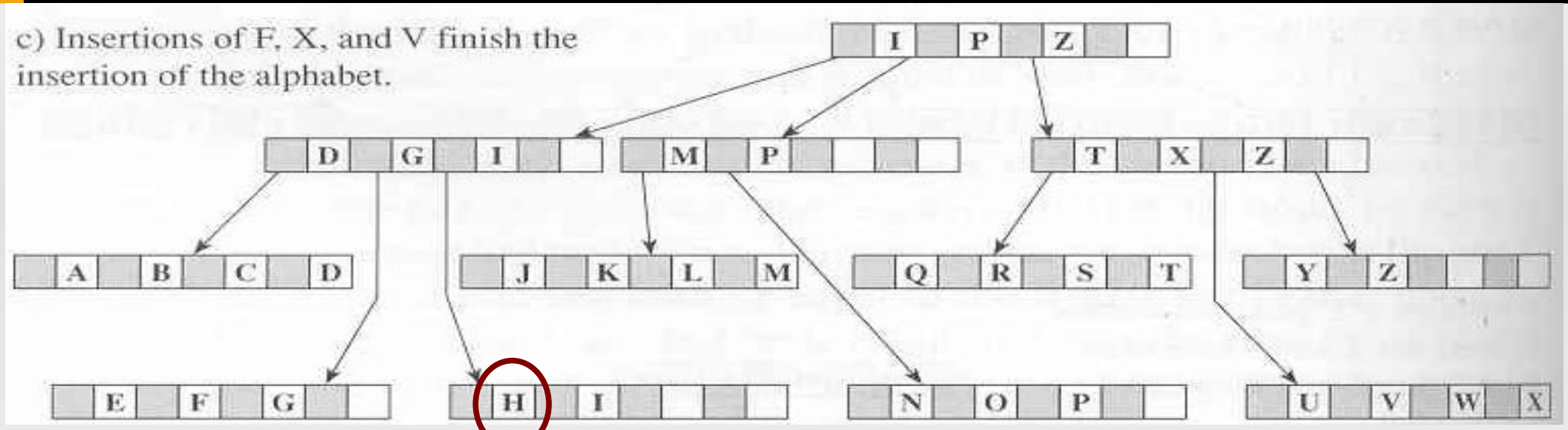


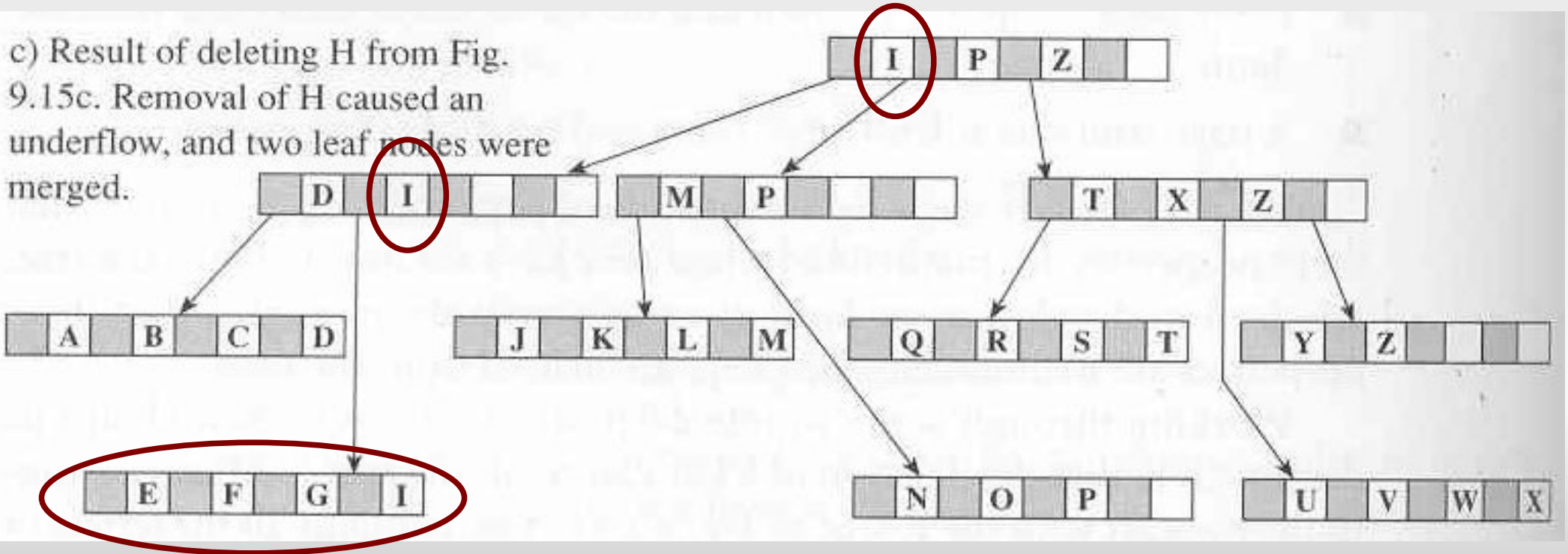
Remoção da chave P





Remoção da chave H





Árvores B – Remoção, junção de páginas e redistribuição

- A junção e as outras operações podem propagar-se até a raiz da árvore B
  - Se a raiz acaba com uma única chave (e consequentemente um filho), ela pode ser eliminada
    - Seu único filho se torna a raiz da árvore e a árvore diminui um nível

Árvores B – Remoção, junção de páginas e redistribuição

- Regras para remoção de uma chave k:
  - (1) se nó tem mais que o número mínimo de chaves e k não é a maior chave
    - apenas remova k do nó
  - (2) se nó tem mais que o número mínimo de chaves mas k é a maior chave
    - remova k do nó
    - modifique os índices superiores para refletir a nova maior chave no nó
  - (3) se nó tem exatamente o número mínimo de chaves e um dos irmãos do nó tem poucas chaves
    - juntar nó com irmão e remover a chave do nó pai

Árvores B – Remoção, junção de páginas e redistribuição

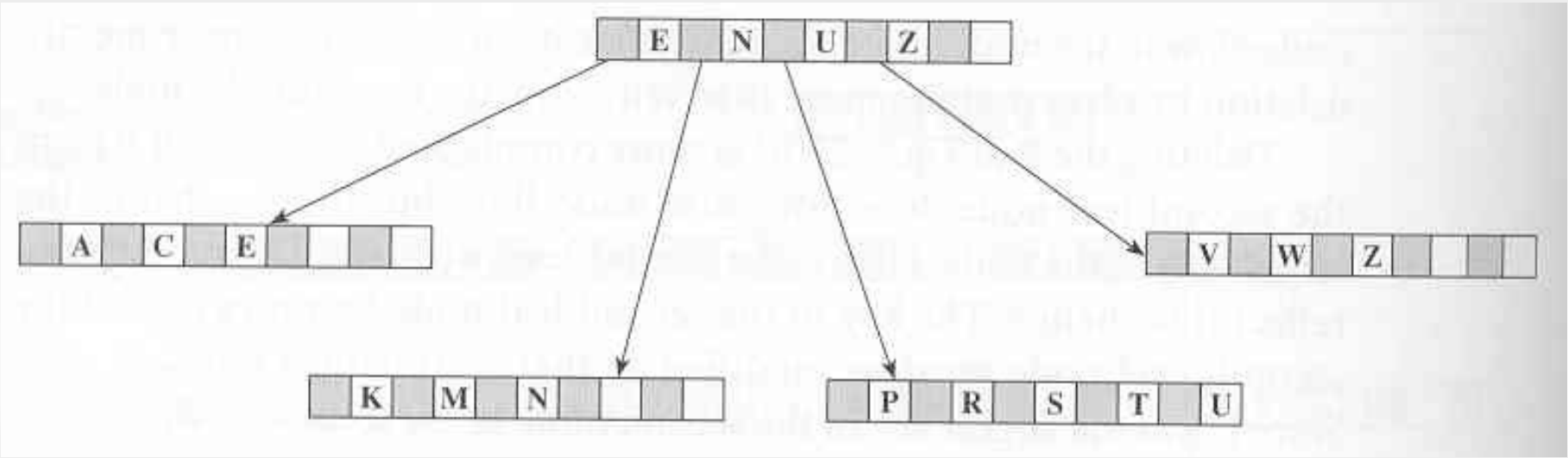
- Regras para remoção de uma chave k:
  - (4) se nó tem exatamente o número mínimo de chaves e um dos irmãos do nó tem chaves extras
    - redistribuir chaves do nó e do irmão entre os dois nós e modificar nós superiores para refletir as maiores chaves em cada nó
      - remoção da chave causaria underflow, que resultaria em junção do nó com nó irmão, que resultaria em divisão

Árvores B – Remoção, junção de páginas e redistribuição

- As duas últimas regras não são exclusivas
  - Implementação deve decidir qual regra aplicar quando ambas são aplicáveis
  - Exemplo: ordem 5
    - como 3 é o número mínimo de chaves, a remoção de chaves pode requerer o ajuste nos nós folha

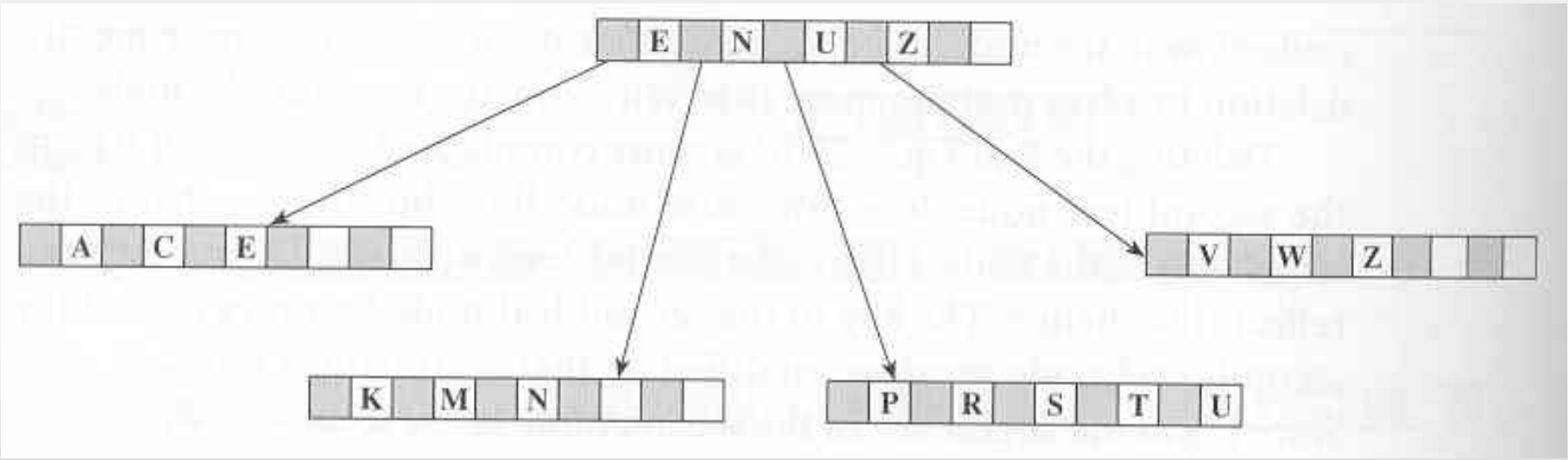
Árvores B – Remoção, junção de páginas e redistribuição

- Remover C:
  - junção do nó {A,E} com nó {K,M,N}
    - Nenhuma redistribuição é possível, pois o irmão tem o número mínimo de chaves



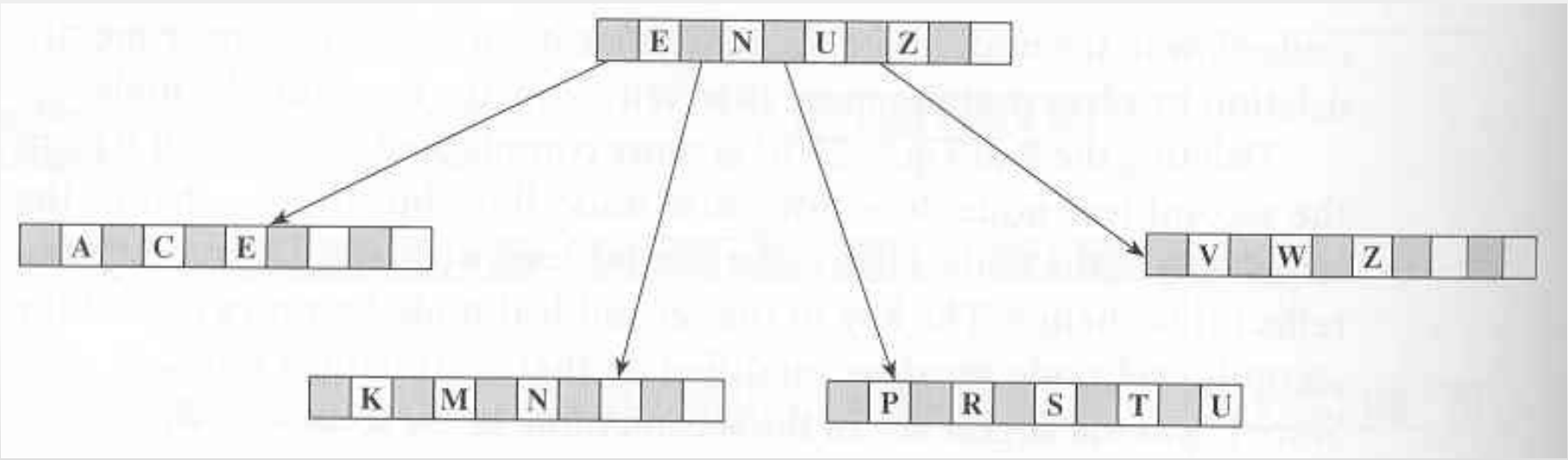
Árvores B – Remoção, junção de páginas e redistribuição

- Remover W:
  - o único irmão tem cinco chaves, de forma que uma ou duas chaves podem ser movidas para esse nó, e não é possível juntar os nós, pois haveriam 7 chaves



Árvores B – Remoção, junção de páginas e redistribuição

- Remover M:
  - há duas opções: intercalação com o irmão esquerdo ou redistribuição de chaves com o irmão direito



Árvores B – Redistribuição

- Apenas entre nós descendentes do mesmo pai
  - ou seja, apenas entre nós irmãos
- Distribuição das chaves
  - pode-se distribuir as chaves uniformemente
  - não causa mudanças recursivas nos índices
    - como ocorre na divisão e junção
    - apenas efeitos locais, nos nós envolvidos e no nó pai

Leitura complementar

- Capítulo **”Multilevel Indexing and B-trees”** do livro
  - Folk et al. ”File Structures: An Object-Oriented Approach with C++”, Editora Pearson, 3ª edição, 1998