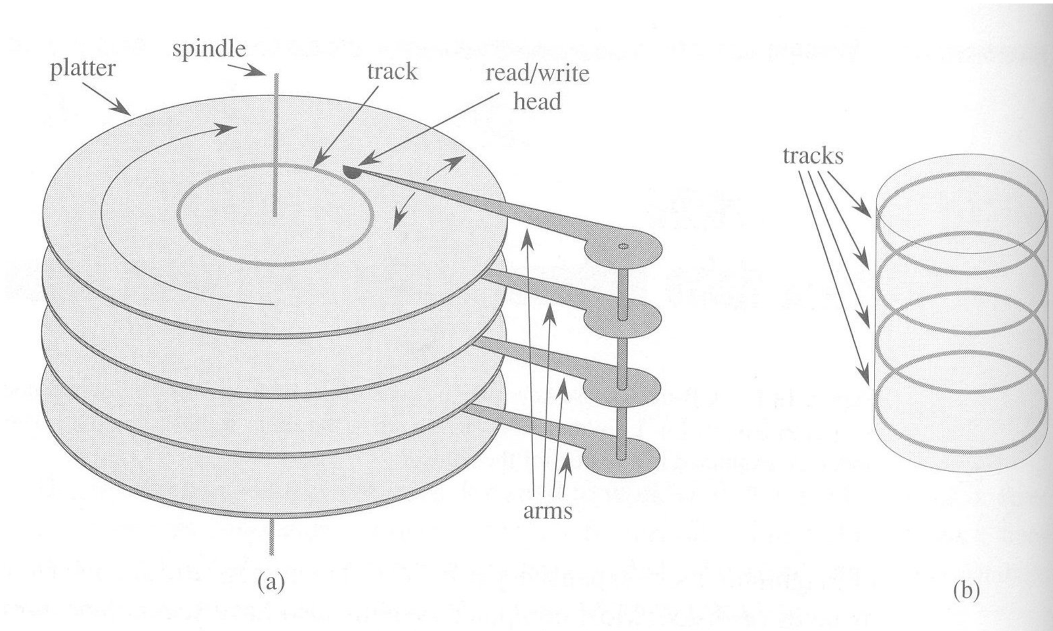
Acesso Sequencial Indexado

- Utiliza o princípio da pesquisa seqüencial $\rightarrow$ cada registro é lido seqüencialmente até encontrar uma chave maior ou igual a chave de pesquisa.
- Providências necessárias para aumentar a eficiência:
  - o arquivo deve ser mantido ordenado pelo campo chave do registro,
  - um arquivo de índices contendo pares de valores $< x, p >$ deve ser criado, onde x representa uma chave e p representa o endereço da página na qual o primeiro registro contém a chave x.
  - Estrutura de um arquivo seqüencial indexado para um conjunto de 15 registros:

3	14	25	41
1	2	3	4

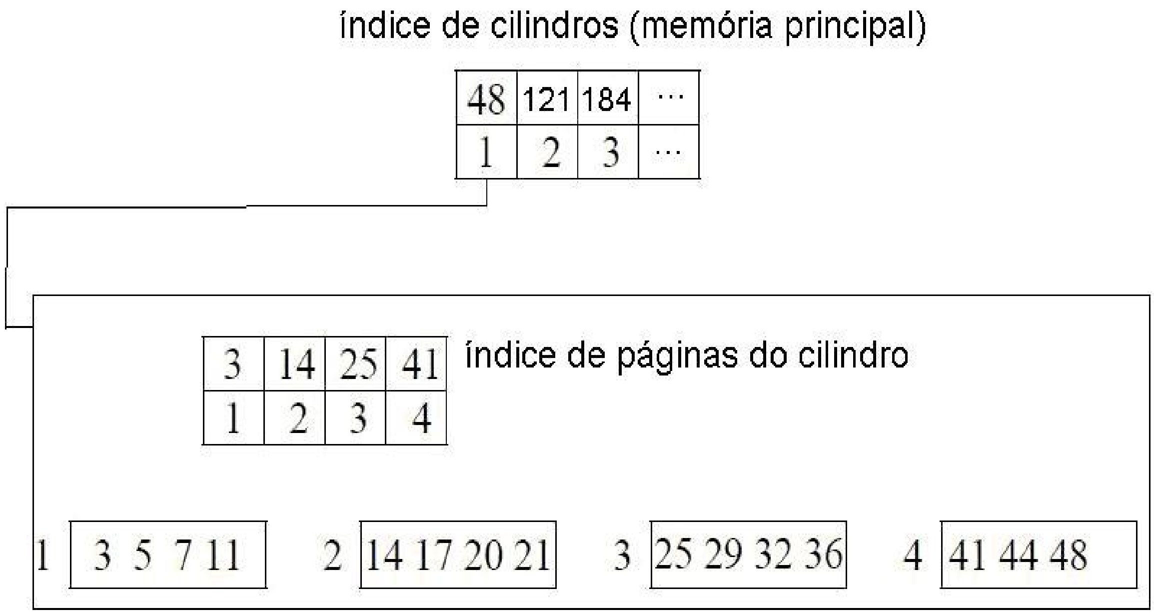
1	3 5 7 11	2	14 17 20 21	3	25 29 32 36	4	41 44 48

Acesso Sequencial Indexado: Disco Magnético



- Dividido em círculos concêntricos (trilhas).
- Cilindro → todas as trilhas verticalmente alinhadas e que possuem o mesmo diâmetro.
- Latência rotacional → tempo necessário para que o início do bloco contendo o registro a ser lido passe pela cabeça de leitura/gravação.
- Tempo de busca (seek time) → tempo necessário para que o mecanismo de acesso desloque de uma trilha para outra (maior parte do custo para acessar dados).
- Acesso seqüencial indexado = acesso indexado + organização seqüencial,
- Aproveitando características do disco magnético e procurando minimizar o número de deslocamentos do mecanismo de acesso → esquema de índices de cilindros e de páginas.

- Para tanto, um índice de cilindros contendo o valor de chave mais alto dentre os registros de cada cilindro é mantido na memória principal. Por sua vez, cada cilindro contém um índice de blocos ou índice de páginas.



- Para localizar o registro que contenha uma chave de pesquisa são necessários os seguintes passos:

  1. localize o cilindro correspondente à chave de pesquisa no índice de cilindros;
  2. desloque o mecanismo de acesso até o cilindro correspondente;
  3. leia a página que contém o índice de páginas daquele cilindro;
  4. leia a página de dados que contém o registro desejado.

- Vantagem: Apenas uma mudança de cilindro.

- Limitação: adequado apenas para aplicações nas quais as operações de inserção e retirada ocorrem com baixa freqüência.

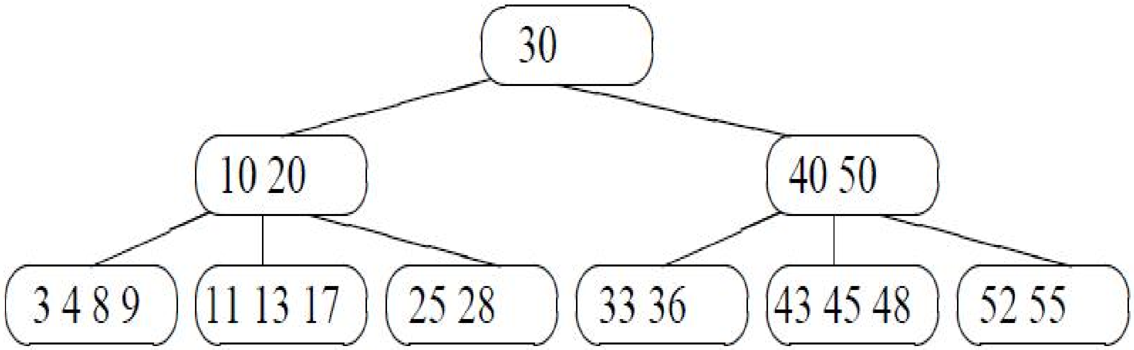
Árvores B

- Árvores n-árias: mais de um registro por nó
- Uma *árvore B* é uma árvore com as seguintes propriedades:

  1. Cada nó x contém os seguintes campos:
     - $n[x]$, o número de chaves atualmente armazenadas no nó x;
     - as $n[x]$ chaves, armazenadas em ordem não decrescente, de modo que $key_1[x] \leq key_2[x] \leq \ldots \leq key_{n[x]}[x]$;
     - $leaf[x]$, um valor booleano indicando se x é uma folha (TRUE) ou um nó interno (FALSE).
     - se x é um nó interno, x contém $n[x] + 1$ ponteiros $c_1[x], c_2[x], \ldots c_{n[x]+1}[x]$ para seus filhos.
  2. As chaves $key_i[x]$ separam as faixas de valores armazenados em cada subárvore: denotando por k_i uma chave qualquer armazenada na subárvore com nó $c_i[x]$, tem-se

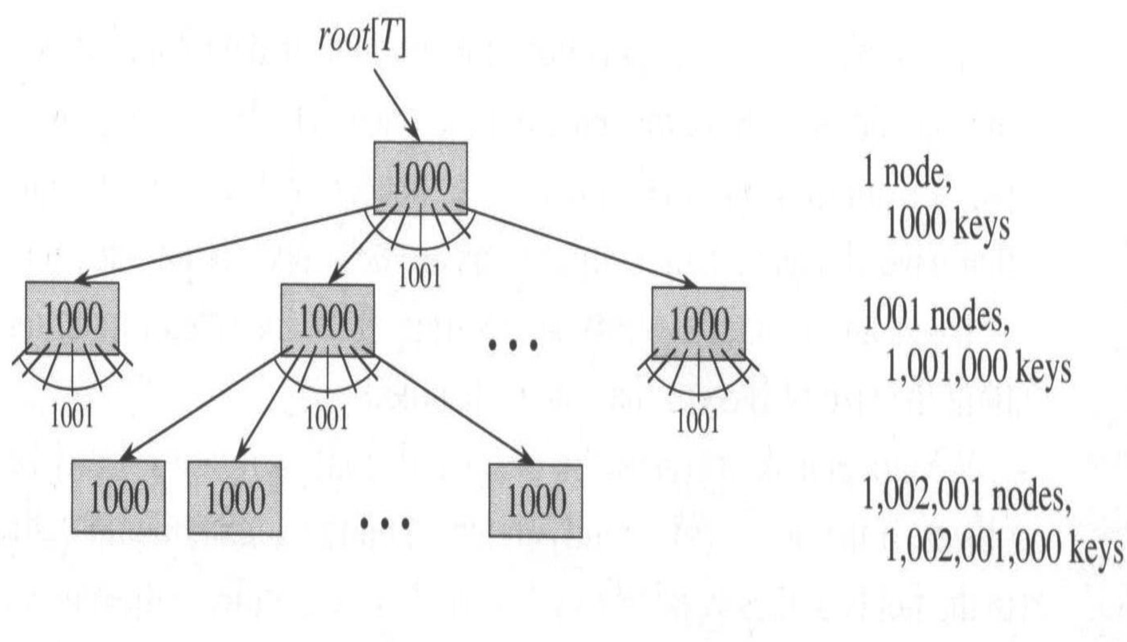
     $$k_1 \leq key_1[x] \leq k_2 \leq key_2[x] \leq \ldots \leq key_{n[x]}[x] \leq k_{n[x]+1}$$

  3. Todas as folhas aparecem no mesmo nível, que é a altura da árvore, h.
  4. Há um limite inferior e superior no número de chaves que um nó pode conter, expressos em termos de um inteiro fixo $t \geq 2$ chamado o *grau mínimo* (ou *ordem*) da árvore.
     - Todo nó que não seja a raiz deve conter pelo menos $t - 1$ chaves. Todo nó interno que não seja a raiz deve conter pelo menos t filhos.
     - Todo nó deve conter no máximo $2t - 1$ chaves (e portanto todo nó interno deve ter no máximo $2t$ filhos). Dizemos que um nó está *cheio* se ele contiver exatamente $2t - 1$ filhos.



- Número máximo de chaves (e filhos) por nó deve ser proporcional ao tamanho da página. Valores usuais de 50 a 2000. Fatores de ramificação altos reduzem drasticamente o número de acessos ao disco.

  Por exemplo, uma árvore B com fator de ramificação 1001 e altura 2 pode armazenar $\geq 10^9$ chaves. Uma vez que a raiz pode ser mantida permanentemente na memória primária, bastam *dois* acessos ao disco para encontrar qualquer chave na árvore.

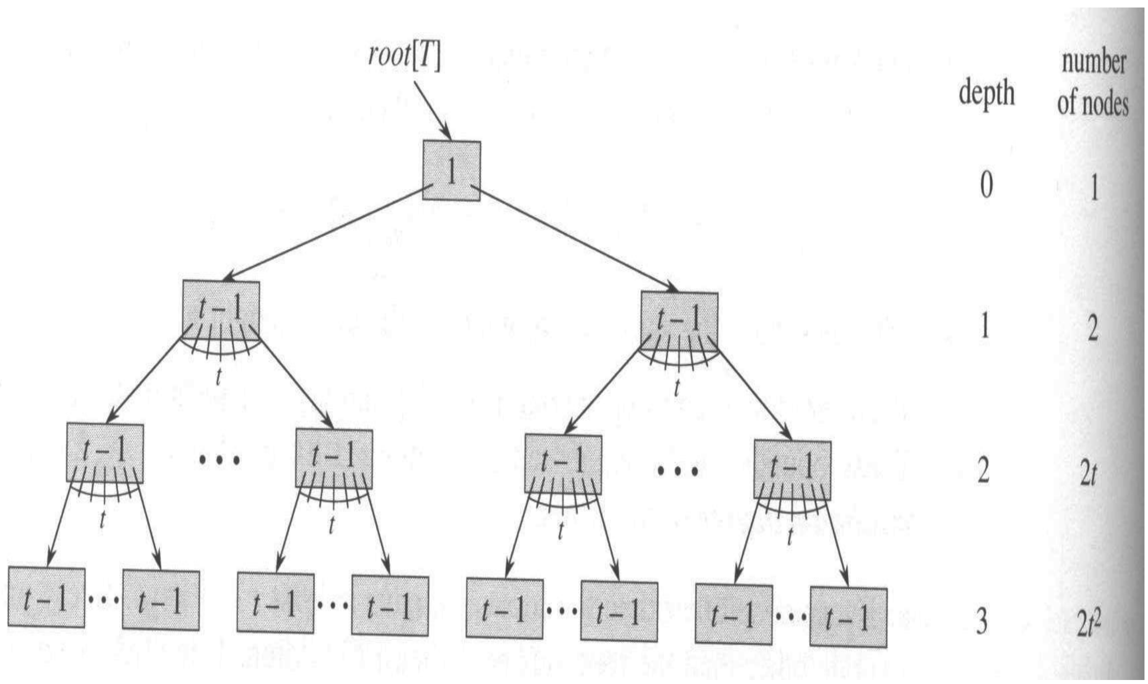


- **Teorema:** Para toda árvore B de grau mínimo $t \geq 2$ contendo n chaves, sua altura h máxima será:

$$h \leq log_t \frac{n+1}{2}$$

Demonstração: Se uma árvore B tem altura h:

  - Sua raiz contem pelo menos uma chave e todos os demais nós contêm pelo menos $t-1$ chaves.
  - Logo, há pelo menos 2 nós no nível 1, pelo menos $2t$ nós no nível 2, etc, até o nível h, onde haverá pelo menos $2t^{h-1}$ nós.



  - Assim, o número n de chaves satisfaz a desigualdade:

$$n \geq 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1} = 1 + 2(t-1)\frac{t^h - 1}{t-1} = 2t^h - 1$$

Obs: Usamos acima a igualdade: $\sum_{i=1}^{h} t^{i-1} = \dfrac{t^h-1}{t-1}$.

  - Logo,

$$t^h \leq (n+1)/2 \Rightarrow h \leq log_t(n+1)/2.$$

Operações Básicas em Árvores B

Criação de uma árvore B vazia

```
B-TREE-CREATE(T)
x ← ALLOCATE-NODE()
leaf[x] ← TRUE
n[x] ← 0
DISK-WRITE(x)
root[T] ← x
```

Busca na árvore B

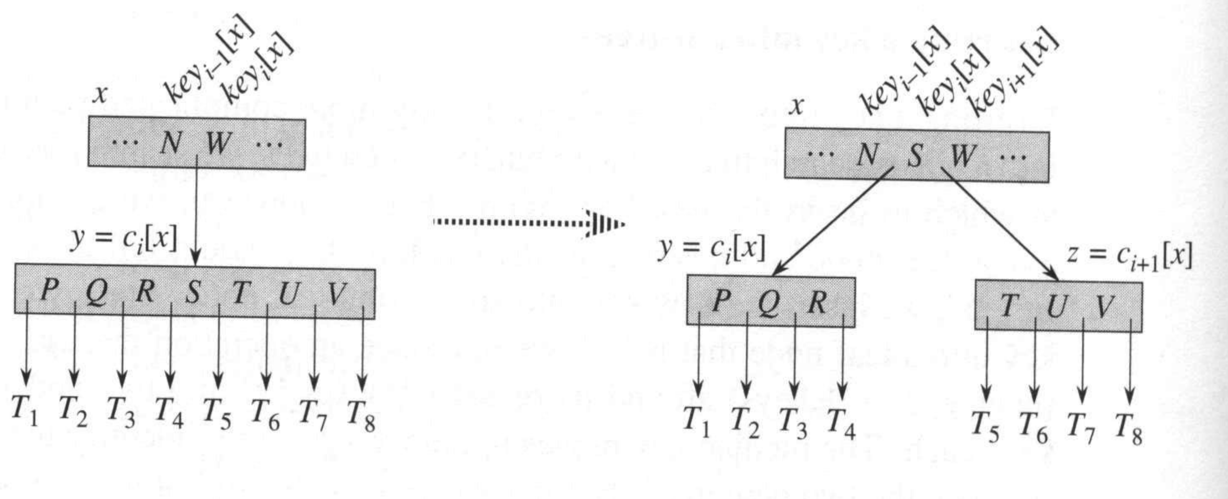
- B-Tree-Search(x, k): tem como parâmetros um ponteiro para a raiz do nó x de uma subárvore e uma chave k a ser procurada na subárvore. Se k está na subárvore, retorna o par ordenado (y, i) composto pelo ponteiro do nó y e o índice i tal que $key_i[y] = k$. Caso contrário, retorna NIL.
- Chamada inicial: B-Tree-Search$(root[T], k)$.

```
B-TREE-SEARCH(x, k)
i ← 1
while i ≤ n[x] and k > key_i[x]
     do i ← i + 1
if i ≤ n[x] and k = key_i[x]
   then return (x, i)
if leaf[x]
   then return NIL
   else DISK-READ(c_i[x])
        return B-TREE-SEARCH(c_i[x], k)
```

Inserção na árvore B

- Localizar o nó apropriado no qual o regisro deve ser inserido.
- Se o registro a ser inserido encontra um nó com menos de $2t-1$ registros, o processo de inserção fica limitado ao nó.
- Se o registro a ser inserido encontra um nó cheio, é criado um novo nó; no caso do nó pai estar cheio o processo de divisão se propaga.
- Divisão de um nó na árvore:

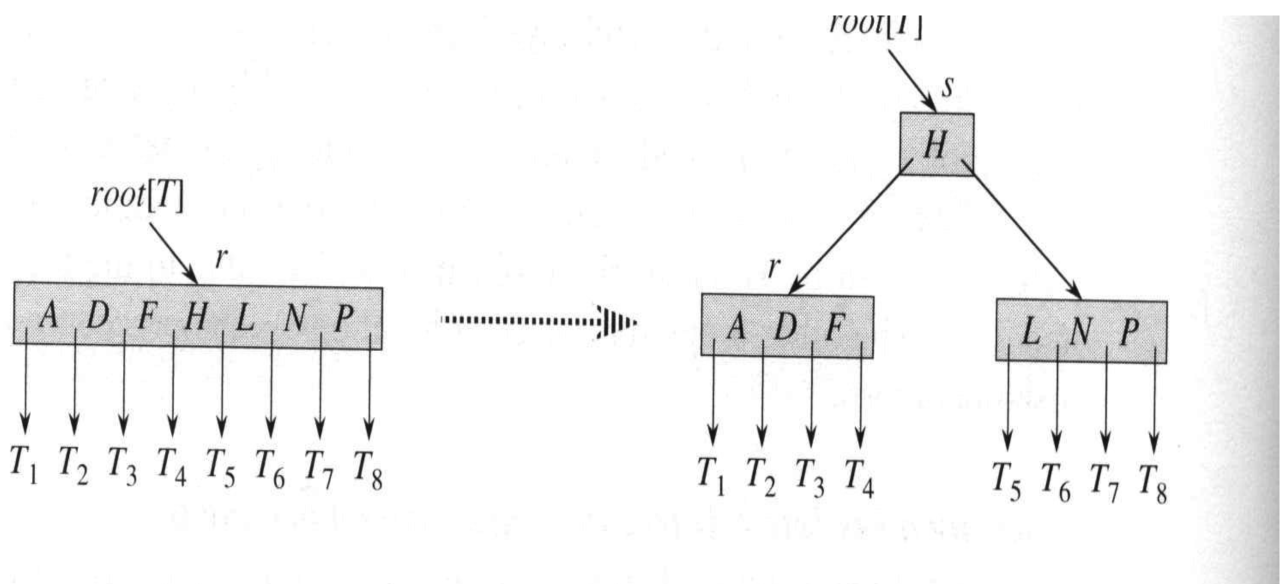
  B-Tree-Split-Child(x, i, y): tem como entrada um nó interno x *não cheio*, um índice i e um nó y tal que $y = c_i[x]$ é um filho *cheio* de x. O procedimento divide y em 2 e ajusta x de forma que este terá um filho adicional.



```
B-TREE-SPLIT-CHILD(x, i, y)
z ← ALLOCATE-NODE()
leaf[z] ← leaf[y]
n[z] ← t − 1
for j ← 1 to t − 1
     do key_j[z] ← key_{j+t}[y]
if not leaf[y]
   then for j ← 1 to t
             do c_j[z] ← c_{j+t}[y]
n[y] ← t − 1
for j ← n[x] + 1 downto i + 1
     do c_{j+1}[x] ← c_j[x]
c_{i+1}[x] ← z
for j ← n[x] downto i
     do key_{j+1}[x] ← key_j[x]
key_i[x] ← key_t[y]
n[x] ← n[x] + 1
DISK-WRITE(y)
DISK-WRITE(z)
DISK-WRITE(x)
```

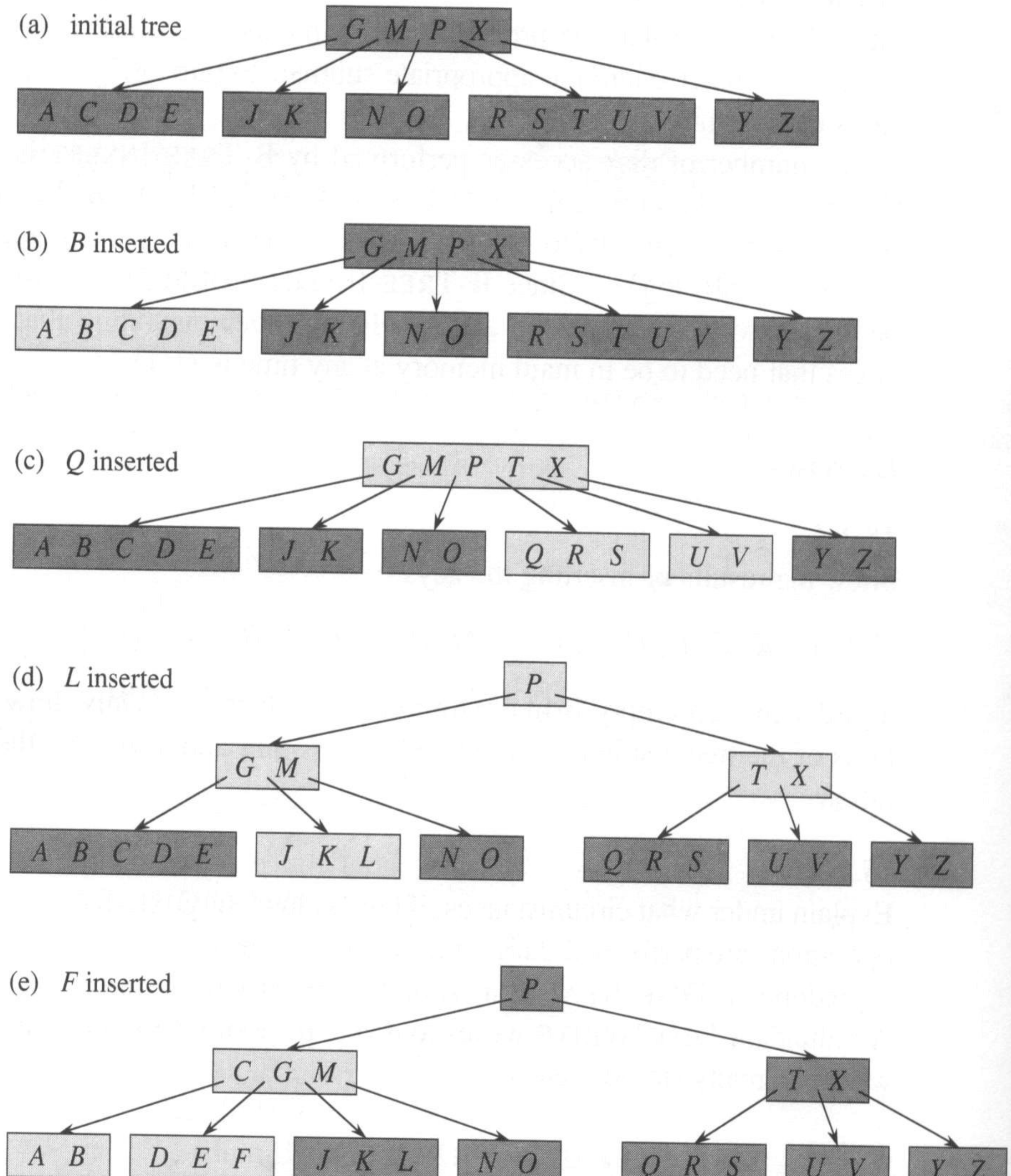
• Inserção de uma chave na árvore com raiz T:

```
B-TREE-INSERT(T, k)
r ← root[T]
if n[r] = 2t − 1
   then s ← ALLOCATE-NODE()
        root[T] ← s
        leaf[s] ← FALSE
        n[s] ← 0
        c1[s] ← r
        B-TREE-SPLIT-CHILD(s, 1, r)
        B-TREE-INSERT-NONFULL(s, k)
   else B-TREE-INSERT-NONFULL(r, k)
```



- Inserção de uma chave em uma subárvore cuja raiz x não está cheia:

```
B-TREE-INSERT-NONFULL(x, k)
i ← n[x]
if leaf[x]
   then while i ≥ 1 and k < key_i[x]
              do key_{i+1}[x] ← key_i[x]
                 i ← i − 1
        key_{i+1}[x] ← k
        n[x] ← n[x] + 1
        DISK-WRITE(x)
   else while i ≥ 1 and k < key_i[x]
              do i ← i − 1
        i ← i + 1
        DISK-READ(c_i[x])
        if n[c_i[x]] = 2t − 1
           then B-TREE-SPLIT-CHILD(x, i, c_i[x])
                if k > key_i[x]
                   then i ← i + 1
        B-TREE-INSERT-NONFULL(c_i[x], k)
```


(a) initial tree
G M P X
A C D E
J K
N O
R S T U V
Y Z
(b) B inserted
G M P X
A B C D E
J K
N O
R S T U V
Y Z
(c) Q inserted
G M P T X
A B C D E
J K
N O
Q R S
U V
Y Z
(d) L inserted
P
G M
T X
A B C D E
J K L
N O
Q R S
U V
Y Z
(e) F inserted
P
C G M
T X
A B
D E F
J K L
N O
Q R S
U V
Y Z

Remoção na árvore B

- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.

1. Se a chave k está no nó x e x é uma folha, exclua a chave k de x.
2. Se a chave k está no nó x e x é um nó interno, faça:

   a) Se o filho y que precede k no nó x tem pelo menos t chaves, então encontre o predecessor k' de k na subárvore com raiz y. Delete recursivamente k', e substitua k por k' em x.

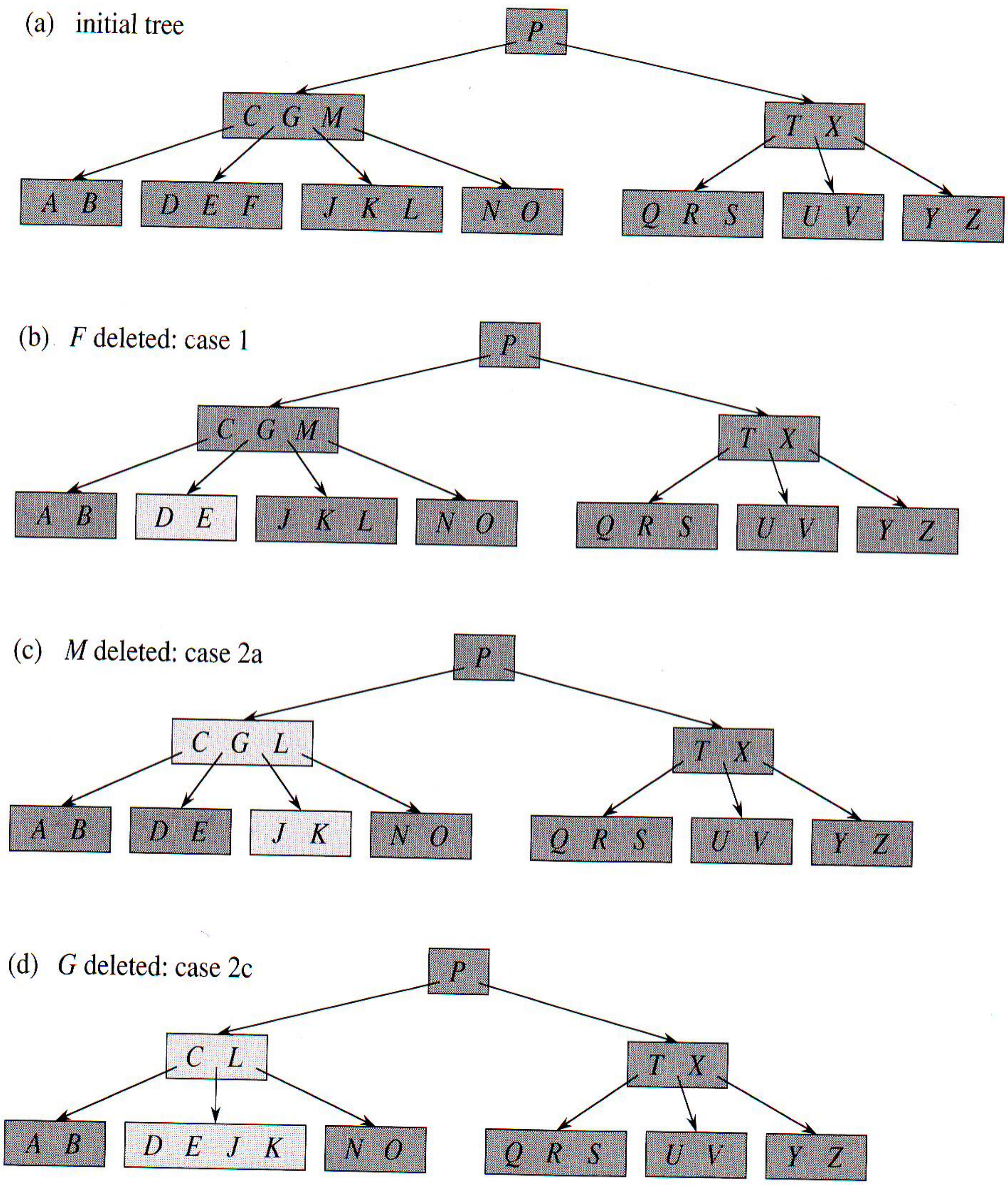
   b) Simetricamente, se o filho z imediatamente após k no nó x tem pelo menos t chaves, então encontre o sucessor k' de k na subárvore com raiz z. Delete recursivamente k', e substitua k por k' em x.

   c) Caso contrário, se ambos y e z possuem apenas $t-1$ chaves, faça a junção de k e todas as chaves de z em y, de forma que x perde tanto a chave k como o ponteiro para z, e y agora contém $2t-1$ chaves. Então, libere z e delete recursivamente k de y.

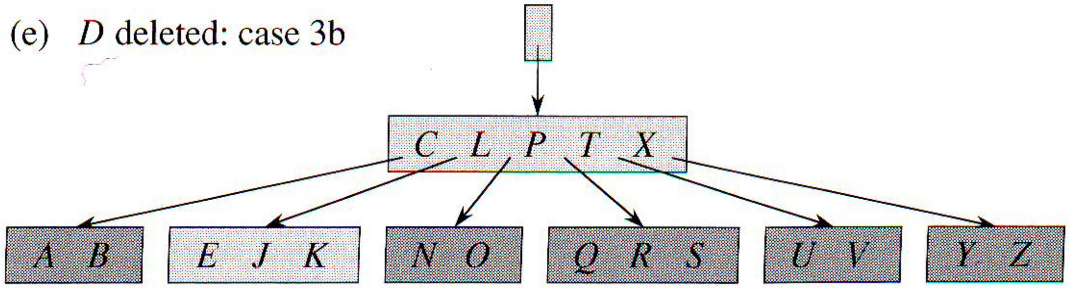
3. Se a chave k não está presente no no interno x, determine a raiz $c_i[x]$ da subárvore apropriada que deve conter k (se k estiver presente na árvore). Se $c_i[x]$ tem apenas $t-1$ chaves, execute o passo 3a ou 3b conforme necessário para garantir que o algoritmo desça para um nó contendo pelo menos t chaves. Então, continue no filho apropriado de x.

   a) Se $c_i[x]$ contém apenas $t-1$ chaves mas tem um irmão imediato com pelo menos t chaves, dê para $c_i[x]$ uma chave extra movendo uma chave de x para $c_i[x]$, movendo uma chave do irmão imediato de $c_i[x]$ à esquerda ou à direita, e movendo o ponteiro do filho apropriado do irmão para o nó $c_i[x]$.

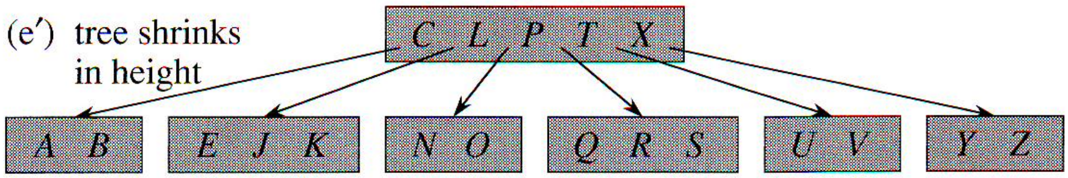
   b) Se $c_i[x]$ e ambos os irmãos imediatos de $c_i[x]$ contêm $t-1$ chaves, faça a junção de $c_i[x]$ com um de seus irmãos. Isso implicará em mover uma chave de x para o novo nó fundido (que se tornará a chave mediana para aquele nó).



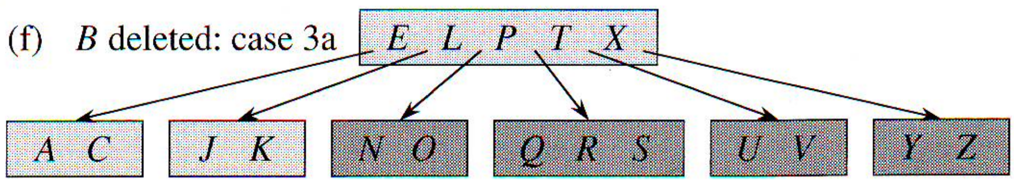
(e) *D* deleted: case 3b



(e′) tree shrinks in height

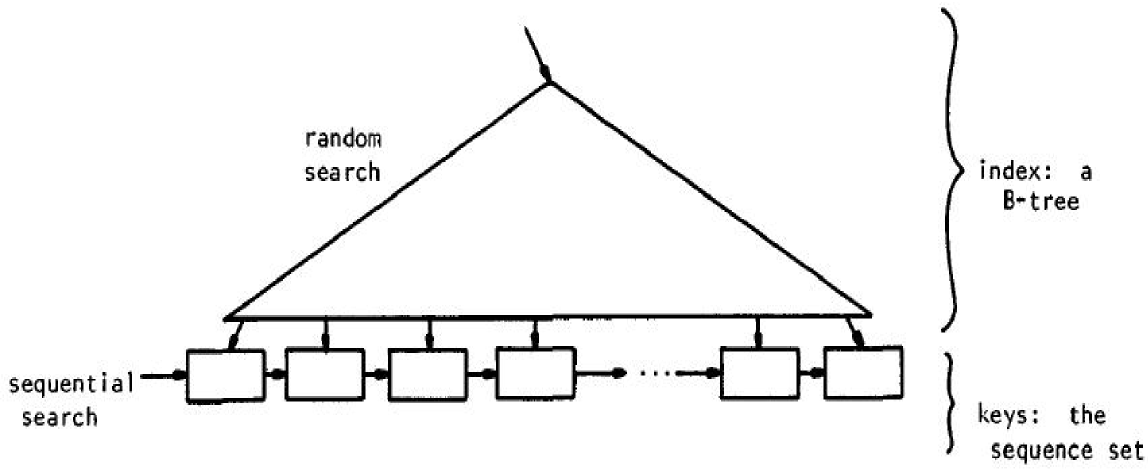


(f) *B* deleted: case 3a



Árvores B^+

- Variantes de árvores B nas quais os registros (e suas chaves) são armazenados no último nível (páginas folhas). Os níveis acima do último nível constituem um índice cuja organização é a organização de uma árvore B.



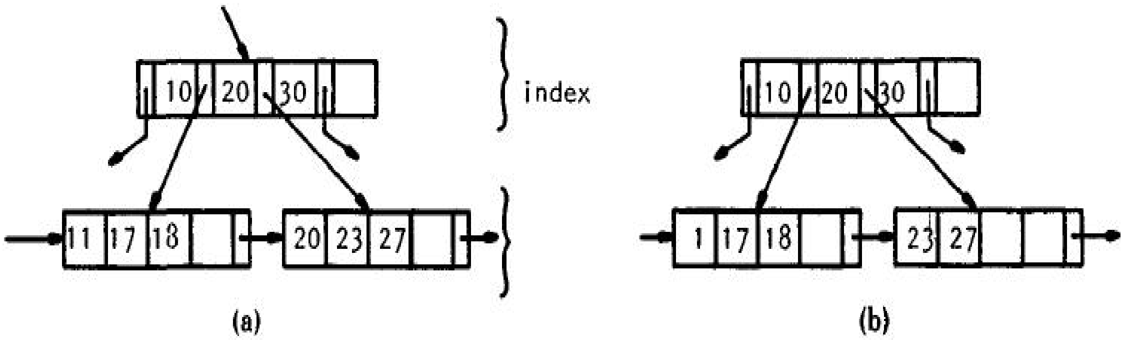
- No índice só aparecem as chaves, sem nenhuma informação associada, enquanto nas páginas folha estão todos os registros do arquivo. Páginas folha são conectadas da esquerda para a direita, permitindo acesso sequencial mais eficiente do que acesso via índice.

- Estrutura e número de chaves nos nós internos e nós folhas podem (e devem) ser diferentes.

- Recuperação de um registro: o processo de pesquisa inicia-se na raiz e continua até uma folha.

- Inserção e retirada é sempre feito em nós folhas. Desde que a página folha contenha o número mínimo de chaves requerido após a remoção, as páginas do índice não precisam ser modificadas, mesmo que uma cópia da chave que pertence ao registro a ser retirado esteja no índice.

  Ex: Remoção da chave 20:



10
20
30
index
11
17
18
20
23
27
(a)
10
20
30
1
17
18
23
27
(b)