AEDS III

Pesquisa em Memória Secundária

Prof. Olga Goussevskaia

Conteúdo

- Localidade de Referência (Meira)
- Memória Virtual e Paginação
- Ordenação Externa
  - Intercalação balanceada
  - Seleção por substituição
  - Intercalação polifásica
  - Quicksort externo
- Acesso Sequencial Indexado em Discos Magnéticos
- Árvores B
- Árvores B*

Acesso Seqüencial 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

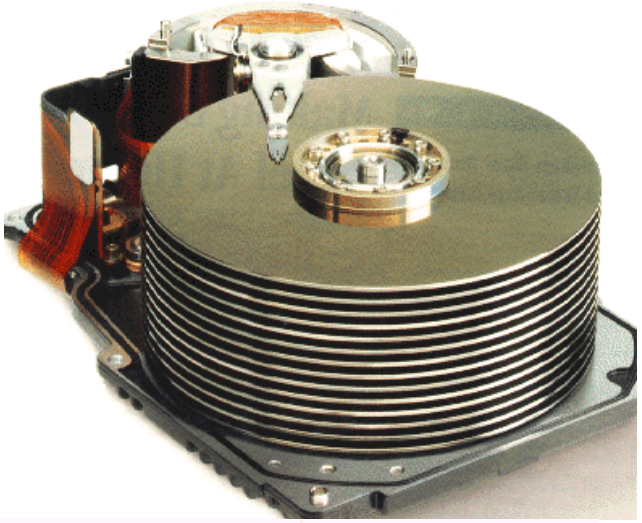
1 | 3 5 7 11 | 2 | 14 17 20 21 | 3 | 25 29 32 36 | 4 | 41 44 48 |

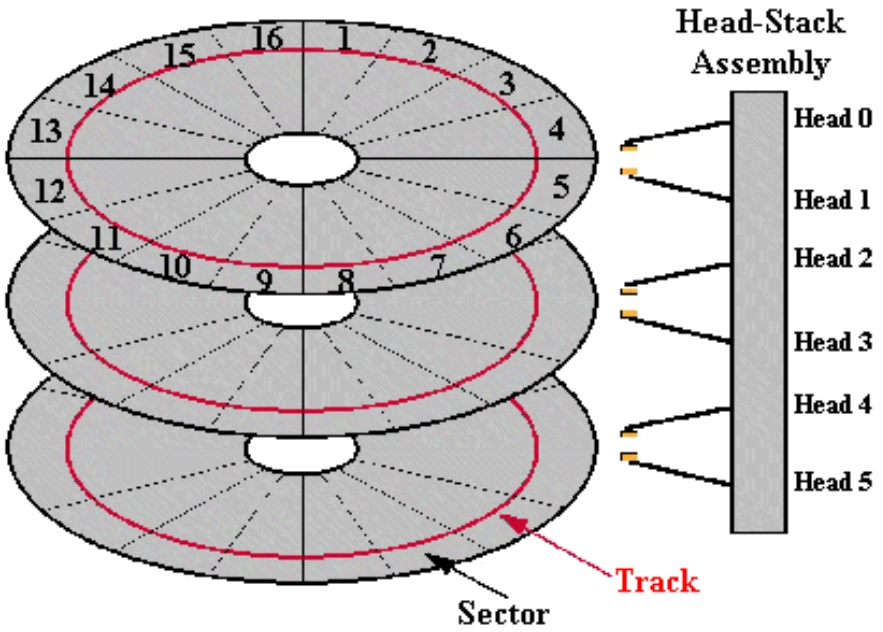
Acesso Seqüencial Indexado: Disco Magnético

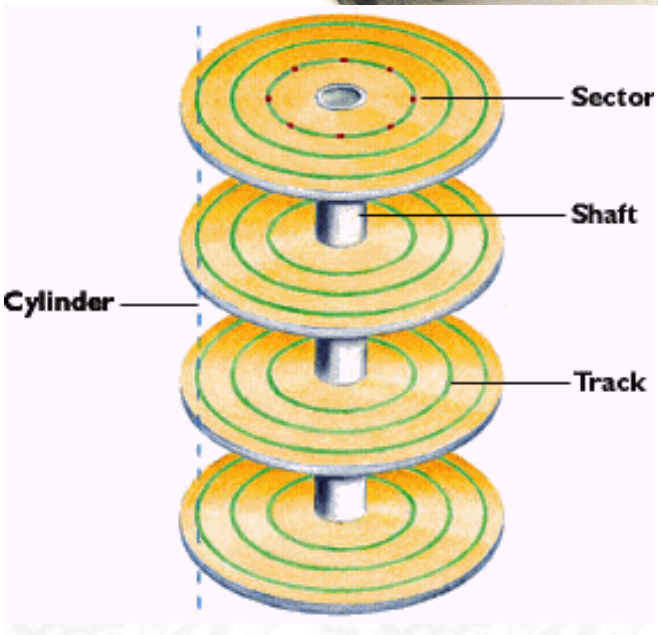
- Dividido em círculos concêntricos (trilhas).
- Cilindro $\rightarrow$ todas as trilhas verticalmente alinhadas e que possuem o mesmo diâmetro.
- Latência rotacional $\rightarrow$ tempo necessário para que o início do bloco contendo o registro a ser lido passe pela cabeça de leitura/gravação.
- Tempo de busca (*seek time*) $\rightarrow$ 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 $\rightarrow$ esquema de índices de cilindros e de páginas.

Acesso Sequencial Indexado

- Disco Magnético
  - Superfícies de gravação (heads)
  - Trilhas (tracks)
  - Cilindro: trilhas alinhadas verticalmente
- Operação mais cara: deslocamento do mecanismo de acesso (seek time)







Acesso Sequencial Indexado em HD

- Índice de cilindros
  - Valor de chave mais alto dentre os registros de cada cilindro
  - Mantido em memória principal
- Índice de páginas
  - Armazenado em cada cilindro
- Passos para localizar um registro
  1. Localizar o cilindro correspondente no índice de cilindros
  2. Deslocar o mecanismo de acesso até o cilindro
  3. Ler página contendo o índice de páginas do cilindro
  4. Ler página de dados contendo o registro

Acesso Sequencial Indexado

- Vantagem:
  - Possível ler qq registro de um arquivo com apenas um deslocamento do mecananismo de acesso (dependendo do tamanho do arquivo e da memória principal)
- Desvantagem:
  - Adequado somente para aplicações pouco dinâmicas (com poucas inserções e remoções, “somente leitura”)
  - Ex: inserir registro 6 no exemplo anterior
  - Possível solução: manter uma área de overflow (armazenamento temporário, atualização em blocos)

Árvores B: Introdução

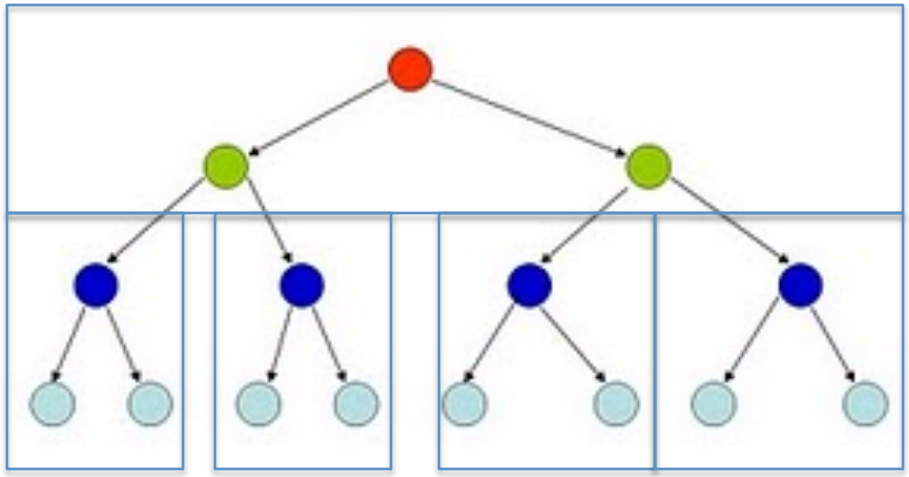
- Árvores binárias (AEDS 2)
  - Estruturas eficientes quando os dados cabem inteiramente na memória principal
  - Permitem:
    - Acesso direto e sequencial
    - Facilidade de inserção e remoção
    - Boa utilização de memória

Árvores B: Introdução

- Problema: recuperar informação em grande arquivos de dados armazenados em memória secundária
- Solução 1: usar uma árvore binária
  - Armazenar nós em disco
  - Apontadores esq. dir. apontam para endereços em disco
  - Custo de uma leitura: O(log n) acessos em disco
    - N=10^6, log_2(10^6) ~= 20 acessos em disco!

Árvores B: Introdução

- Solução 2: agrupar nós da árvore binária em páginas

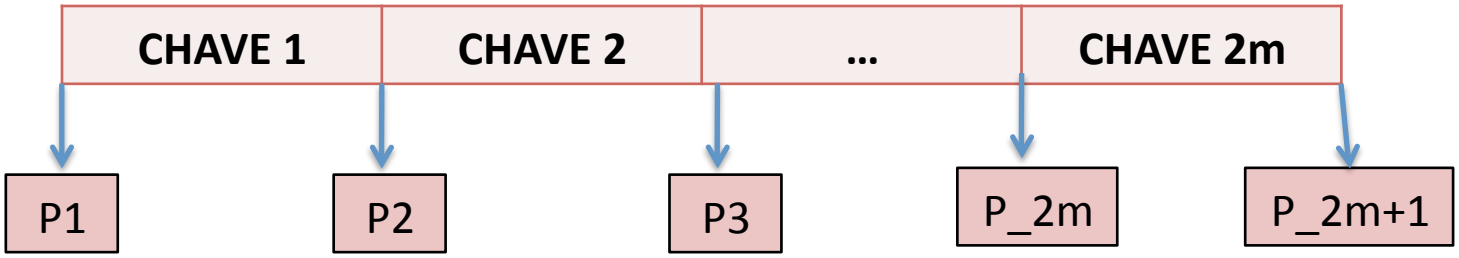


- Árvore binária -> quaternária (4 filhos por página)
- Problema: qual a melhor forma de distribuir os registros entre as páginas? (problema de otimização complexo)

Árvores B

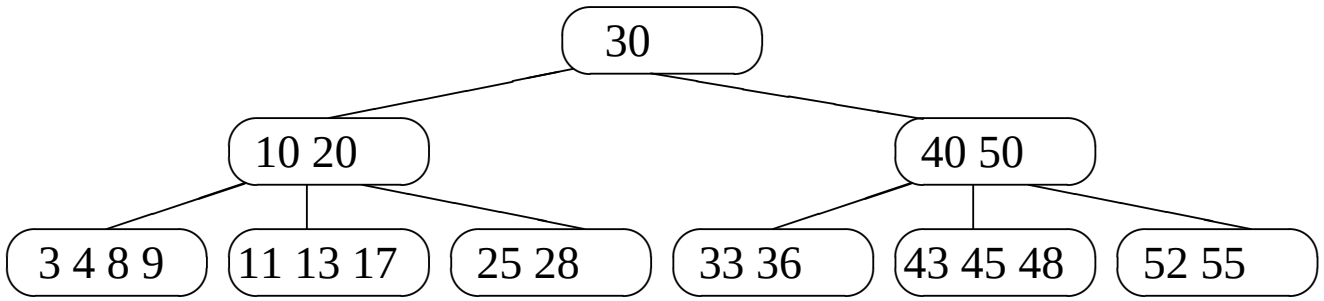
- Solução genial para o problema
- Proposto por Bayer e McCreight em 1972 (Boeing Research Lab)

- Página de uma árvore B de ordem m:



Árvores B

- Árvores n-árias: mais de um registro por nodo.
- Em uma árvore B de ordem m:
  - página raiz: 1 e $2m$ registros.
  - demais páginas: no mínimo m registros e $m+1$ descendentes e no máximo $2m$ registros e $2m+1$ descendentes.
  - páginas folhas: aparecem todas no mesmo nível.
- Registros em ordem crescente da esquerda para a direita.
- Extensão natural da árvore binária de pesquisa.
- Árvore B de ordem $m = 2$ com três níveis:



Árvores B - TAD Dicionário

- Estrutura de Dados:

```
typedef long TipoChave;
typedef struct TipoRegistro {
   TipoChave Chave;
   /*outros componentes*/
} TipoRegistro;
typedef struct TipoPagina* TipoApontador;
typedef struct TipoPagina {
   short n;
   TipoRegistro r[MM];
   TipoApontador p[MM + 1];
} TipoPagina;
```

Árvores B - TAD Dicionário

- Operações:
  - Inicializa

```
void Inicializa(TipoApontador *Dicionario)
{  *Dicionario = NULL; }
```

  - Pesquisa
  - Insere
  - Remove

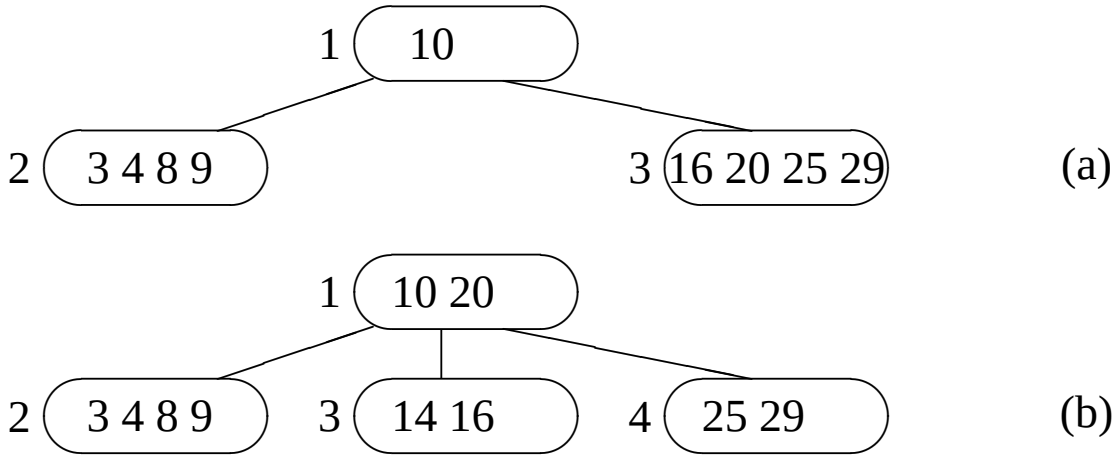
Árvores B - Pesquisa

```
void Pesquisa(TipoRegistro *x, TipoApontador Ap)
{ long i = 1;
  if (Ap == NULL)
  { printf("TipoRegistro nao esta presente na arvore\n");
    return;
  }
  while (i < Ap->n && x->Chave > Ap->r[i-1].Chave) i++;
  if (x->Chave == Ap->r[i-1].Chave)
  { *x = Ap->r[i-1];
    return;
  }
  if (x->Chave < Ap->r[i-1].Chave)
  Pesquisa(x, Ap->p[i-1]);
  else Pesquisa(x, Ap->p[i]);
}
```

Árvores B - Inserção

1. Localizar a página apropriada aonde o registro deve ser inserido.
2. Se o registro a ser inserido encontra uma página com menos de $2m$ registros, o processo de inserção fica limitado à página.
3. Se o registro a ser inserido encontra uma página cheia, é criada uma nova página, no caso da página pai estar cheia o processo de divisão se propaga.

Exemplo: Inserindo o registro com chave 14.

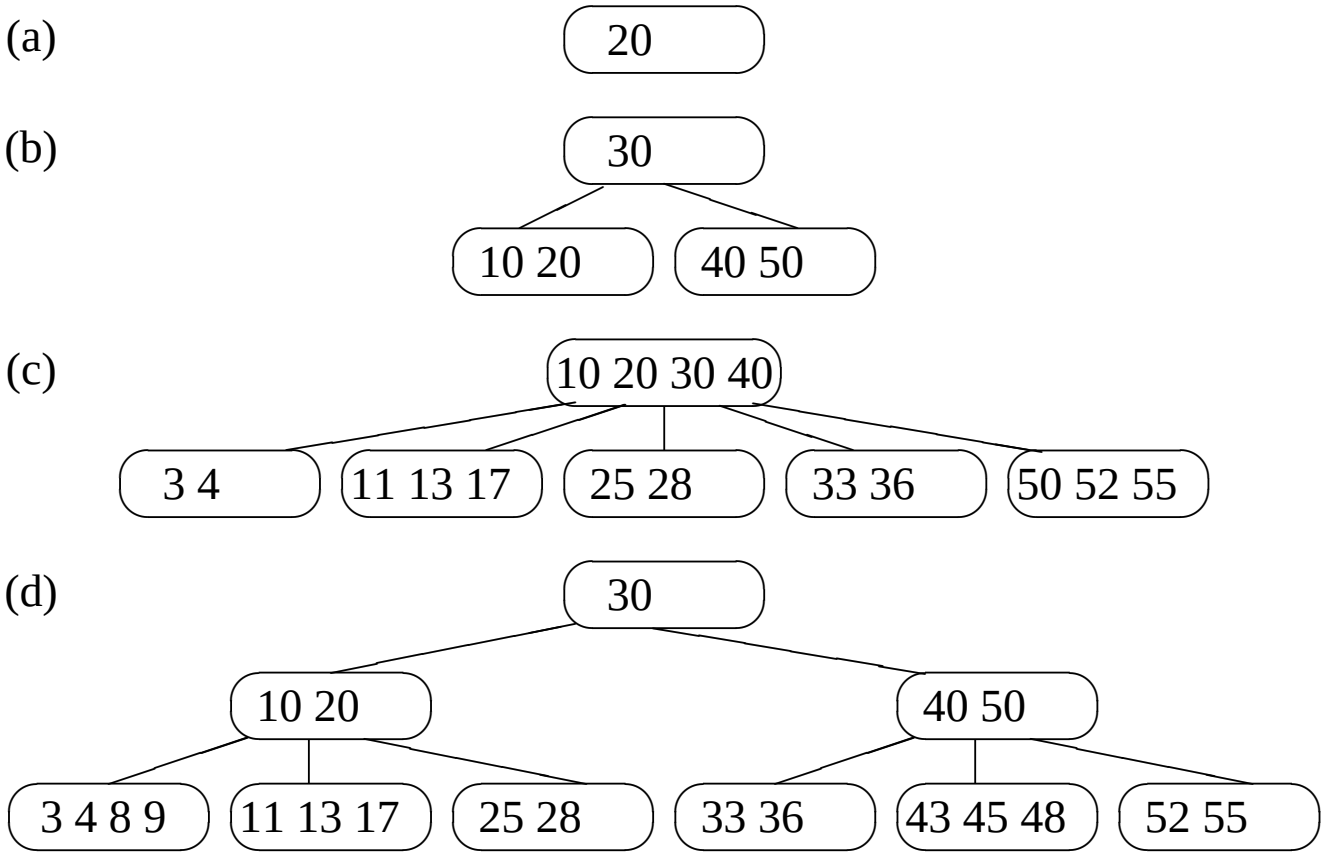


Árvores B

- Inserção
  - Se a página a receber o novo registro contém 2m registros, a mesma é quebrada em duas, cada uma com m registros. O (m+1)ésimo registro (do meio) é movido para o nó-pai.
  - Se a página-pai estiver cheia, o mesmo procedimento de divisão é repetido recursivamente
  - No pior caso, uma nova raíz é criada, aumentando a altura da árvore.
  - Obs: uma árvore B somente aumenta a altura com a divisão da raíz

Árvores B - Inserção

Exemplo de inserção das chaves: 20, 10, 40, 50, 30, 55, 3, 11, 4, 28, 36, 33, 52, 17, 25, 13, 45, 9, 43, 8 e 48



Árvores B - Remoção

- Página com o registro a ser retirado é folha:
  1. retira-se o registro,
  2. se a página não possui pelo menos de m registros, a propriedade da árvore B é violada. Pega-se um registro emprestado da página vizinha. Se não existir registros suficientes na página vizinha, as duas páginas devem ser fundidas em uma só.
- Pagina com o registro não é folha:
  1. o registro a ser retirado deve ser primeiramente substituído por um registro contendo uma chave adjacente.

Árvores B

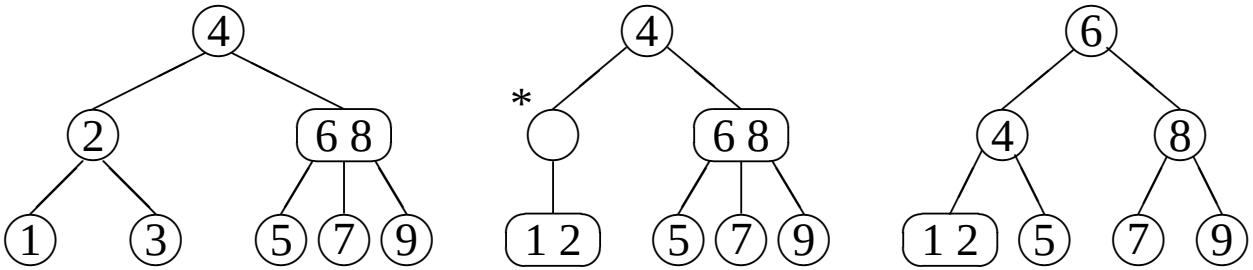
- Remoção
  - Quando o registro a ser removido não pertence a uma página-folha, o mesmo deve ser substituído pelo registro de chave adjacente
  - Chave adjacente antecessora: está na página-folha mais à direita na subárvore à esquerda
  - Chave adjacente sucessora: está na página-folha mais à esquerda na subárvore à direita

Árvores B

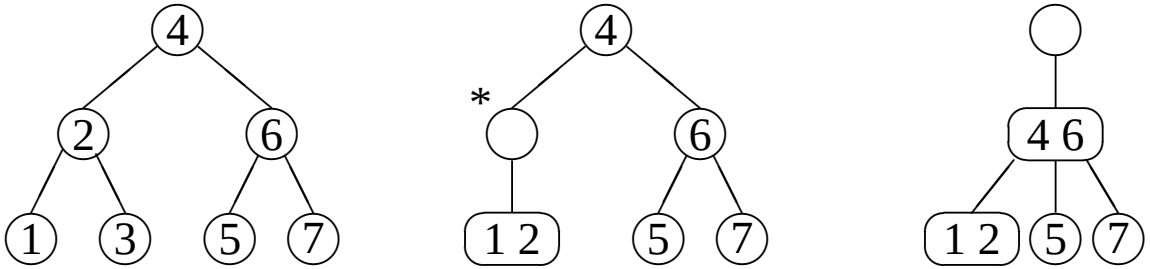
- Remoção
  - Se o número de registros restantes na página-folha for <m, um registro da página vizinha deve ser emprestado
  - Se a página vizinha tiver =m registros apenas, as duas páginas devem ser fundidas, pois possuem 2m-1 registros.
  - Fundindo duas páginas:
    - O registro do meio deve ser emprestado do nó-pai
    - O procedimento é propagado até a raíz
    - Se num.registros(raíz)=0, reduzir altura

Árvores B - Remoção

Exemplo: Retirando a chave 3.



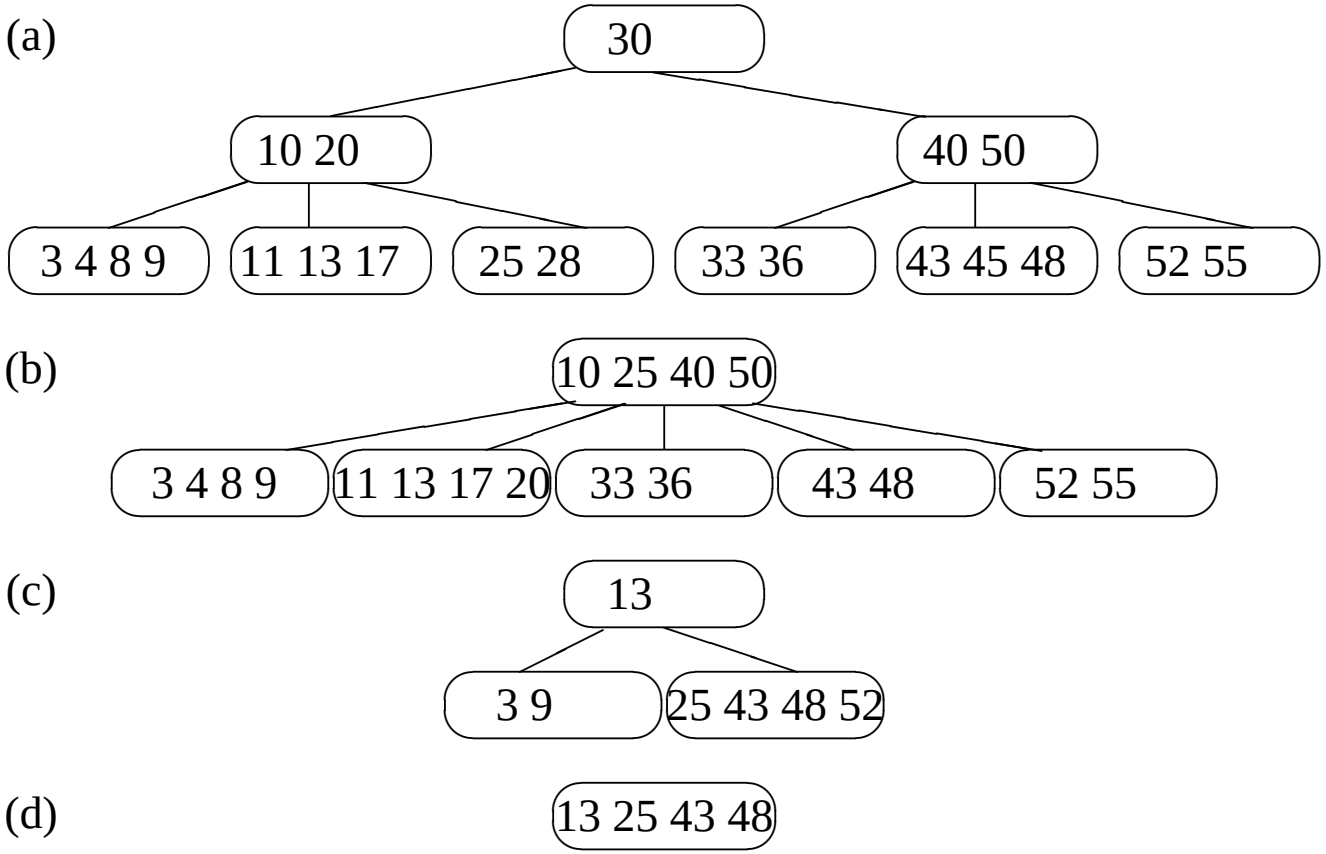
(a) Página vizinha possui mais do que m registros



(b) Página vizinha possui exatamente m registros

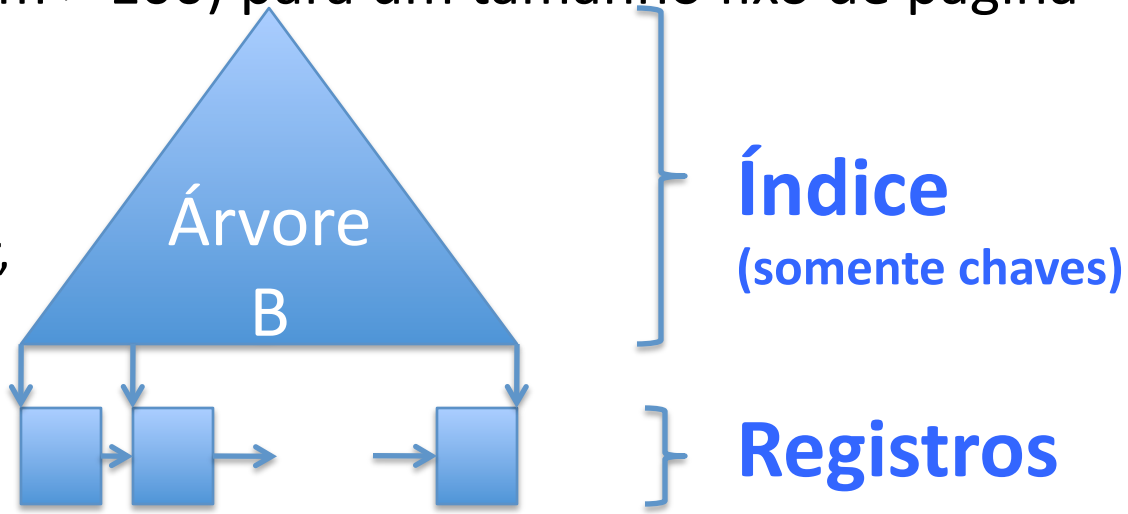
Árvores B - Remoção

Remoção das chaves 45 30 28; 50 8 10 4 20 40 55 17 33 11 36; 3 9 52.



Árvores B*

- Em inglês, referenciado como “B+ trees”
- Uma implementação alternativa de árvore B
- Todos os registros são armazenados no primeiro nível
- Os níveis acima das folhas constituem um índice
- Vantagens:
  - Acesso sequencial mais eficiente
  - Maior ordem (m > 100) para um tamanho fixo de página
- Aplicações:
  - Filesystems: NTFS
  - SGBDs: SQL Sever, Oracle, MySQL, etc.



Árvores B*

- Pesquisa:
  - Semelhante à pesquisa em árvore B
  - Não pára se a chave procurada for encontrada em uma página do índice (não folha)
  - Ao encontrar a chave, segue o apontador à direita até uma página folha
  - Os valores encontrados no meio do caminho são irrelevantes, desde que conduzam à página-folha correta
  - Como páginas-folhas não têm os *(2m+1)* apontadores, esse espaço pode ser usado para armazenar mais registros em uma página-folha. Portanto, o *m* pode ser diferente para as páginas-folhas.

Árvores B* - TAD Dicionário

- Estrutura de Dados:

```
typedef int TipoChave;
typedef struct TipoRegistro {
  TipoChave Chave;
  /* outros componentes */
} TipoRegistro;
typedef enum {
  Interna, Externa
} TipoIntExt;
typedef struct TipoPagina *TipoApontador;
typedef struct TipoPagina {
  TipoIntExt Pt;
  union {
    struct {
      int ni;
      TipoChave ri[MM];
      TipoApontador pi[MM + 1];
    } U0;
    struct {
      int ne;
      TipoRegistro re[MM2];
    } U1;
  } UU;
} TipoPagina;
```

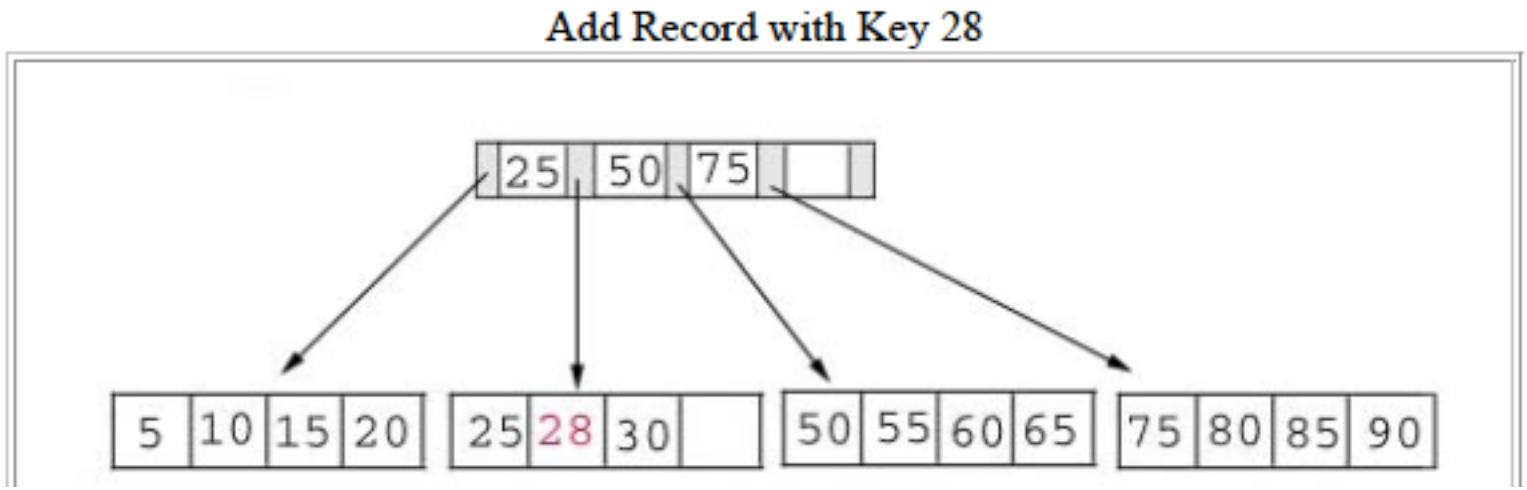
Árvores B* - Procedimento para pesquisar na árvore B*

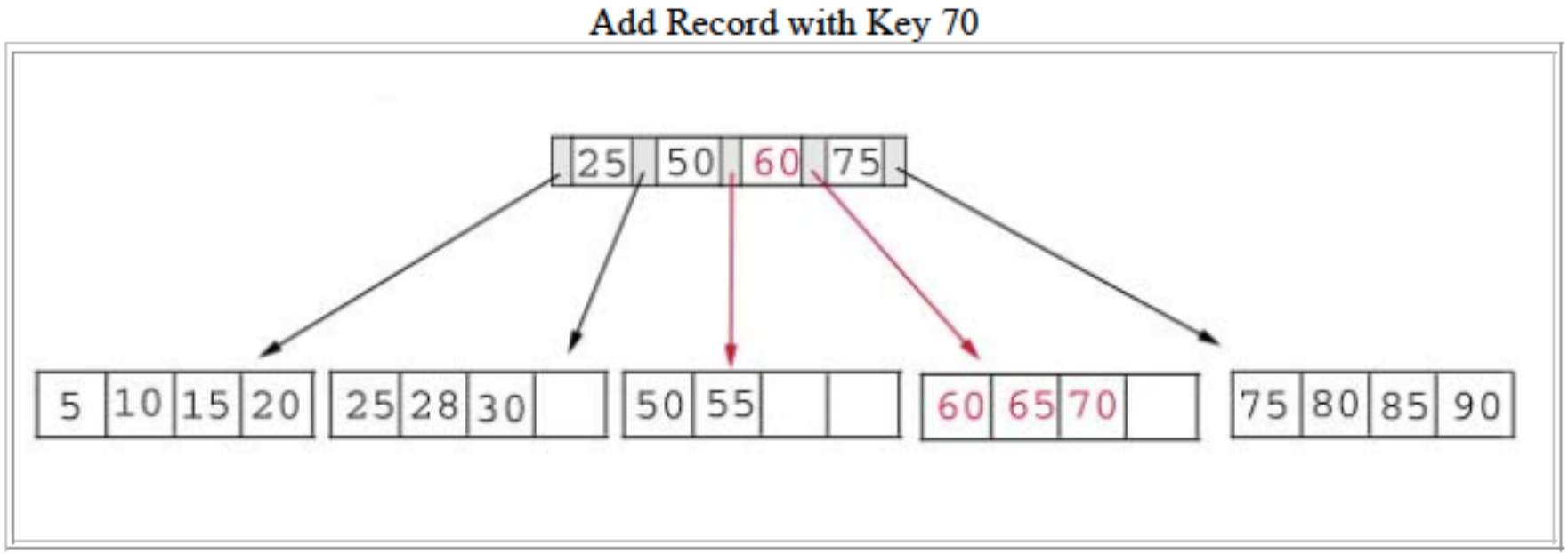
```
void Pesquisa(TipoRegistro *x, TipoApontador *Ap)
{ int i;
  TipoApontador Pag;
  Pag = *Ap;
  if ((*Ap)->Pt == Interna)
  { i = 1;
    while (i < Pag->UU.U0.ni && x->Chave > Pag->UU.U0.ri[i - 1]) i++;
    if (x->Chave < Pag->UU.U0.ri[i - 1])
    Pesquisa(x, &Pag->UU.U0.pi[i - 1]);
    else Pesquisa(x, &Pag->UU.U0.pi[i]);
    return;
  }
  i = 1;
  while (i < Pag->UU.U1.ne && x->Chave > Pag->UU.U1.re[i - 1].Chave)
    i++;
  if (x->Chave == Pag->UU.U1.re[i - 1].Chave)
  *x = Pag->UU.U1.re[i - 1];
  else printf("TipoRegistro nao esta presente na arvore\n");
}
```

Árvores B* - Inserção e Remoção

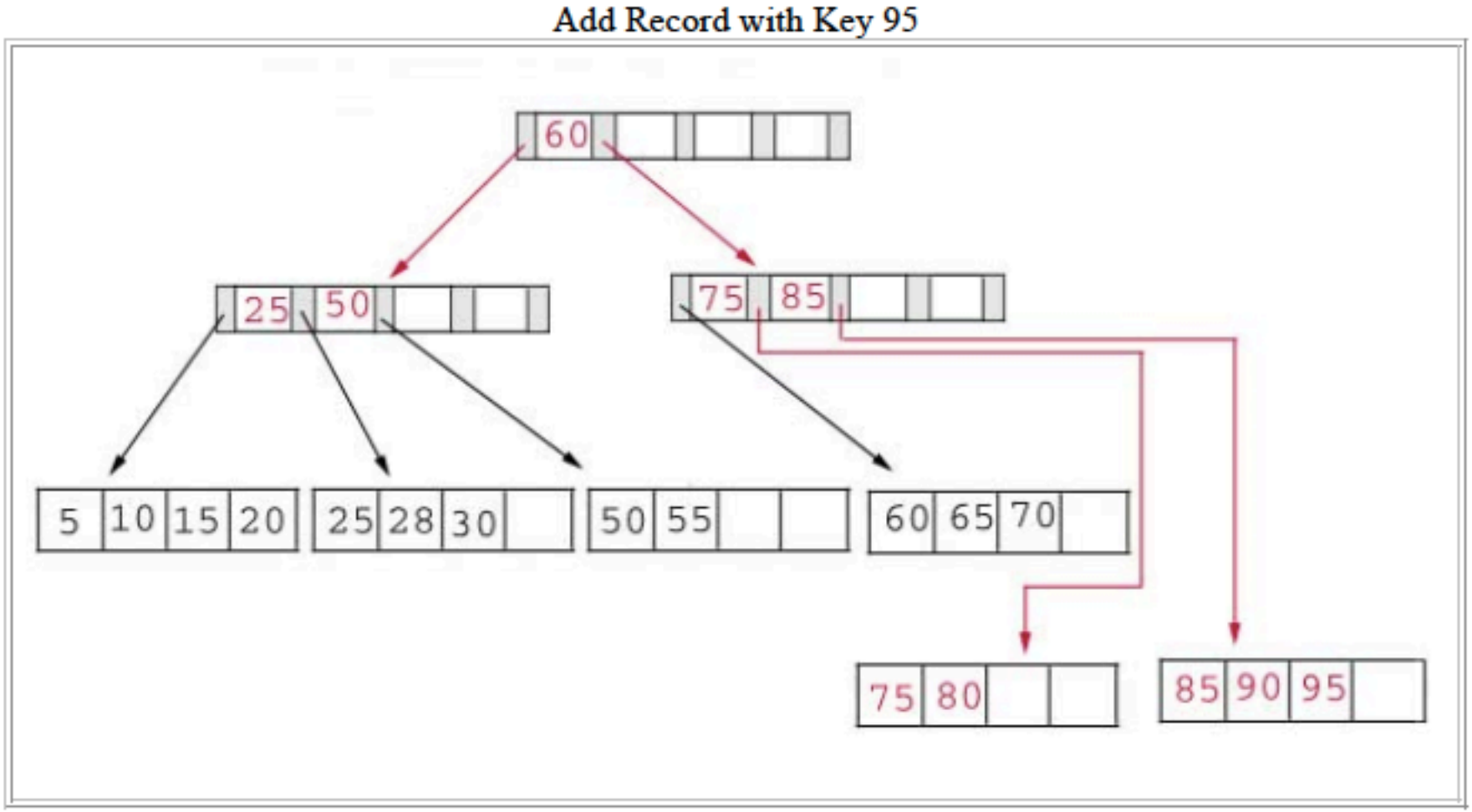
- Inserção na árvore B*
  - Semelhante à inserção na árvore B,
  - Diferença: quando uma folha é dividida em duas, o algoritmo promove uma cópia da chave que pertence ao registro do meio para a página pai no nível anterior, retendo o registro do meio na página folha da direita.
- Remoção na árvore B*
  - Relativamente mais simples que em uma árvore B,
  - Todos os registros são folhas,
  - Desde que a folha fique com pelo menos metade dos registros, as páginas dos índices não precisam ser modificadas, mesmo se uma cópia da chave que pertence ao registro a ser retirado esteja no índice.

Árvores B*: inserção

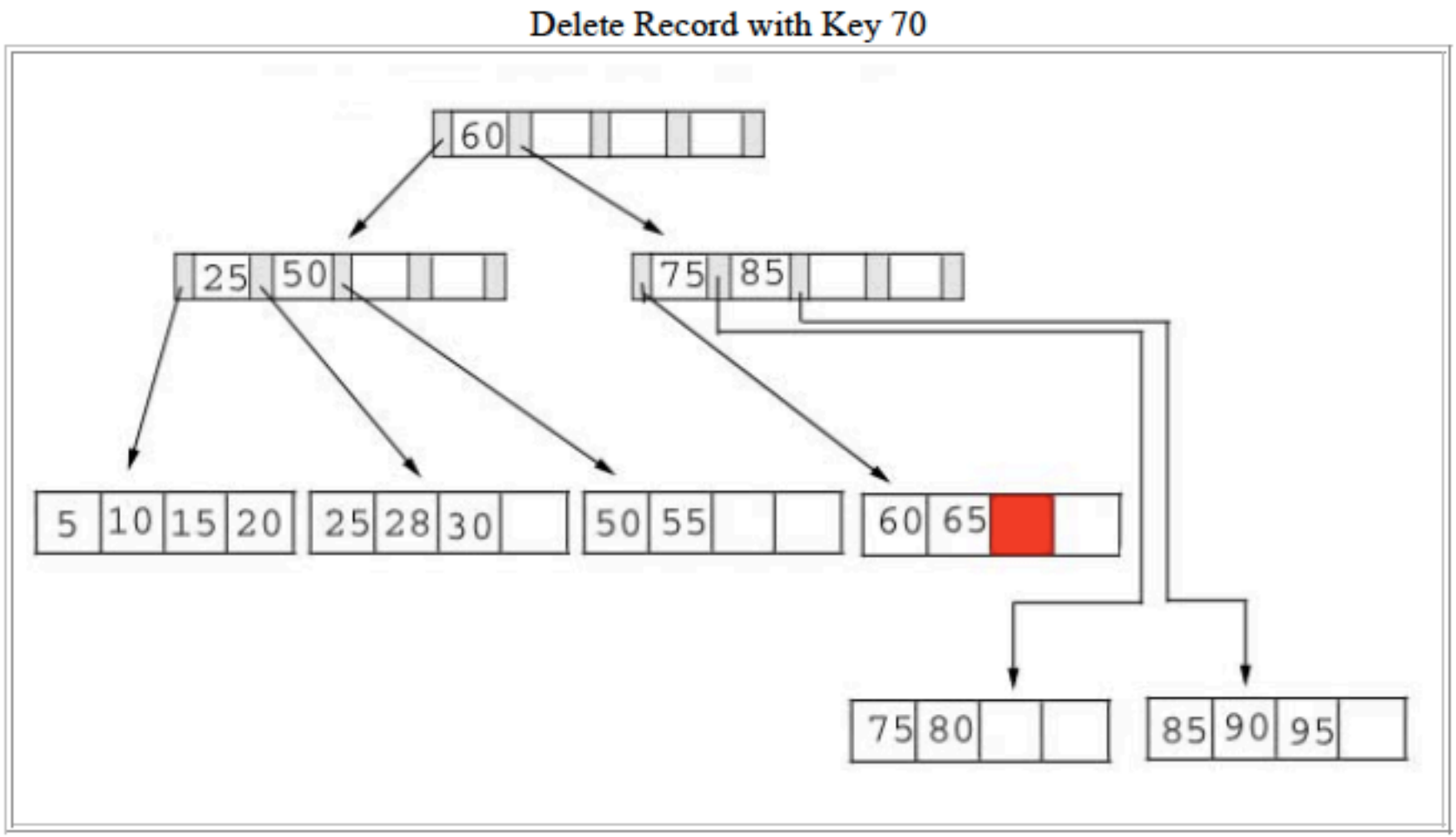




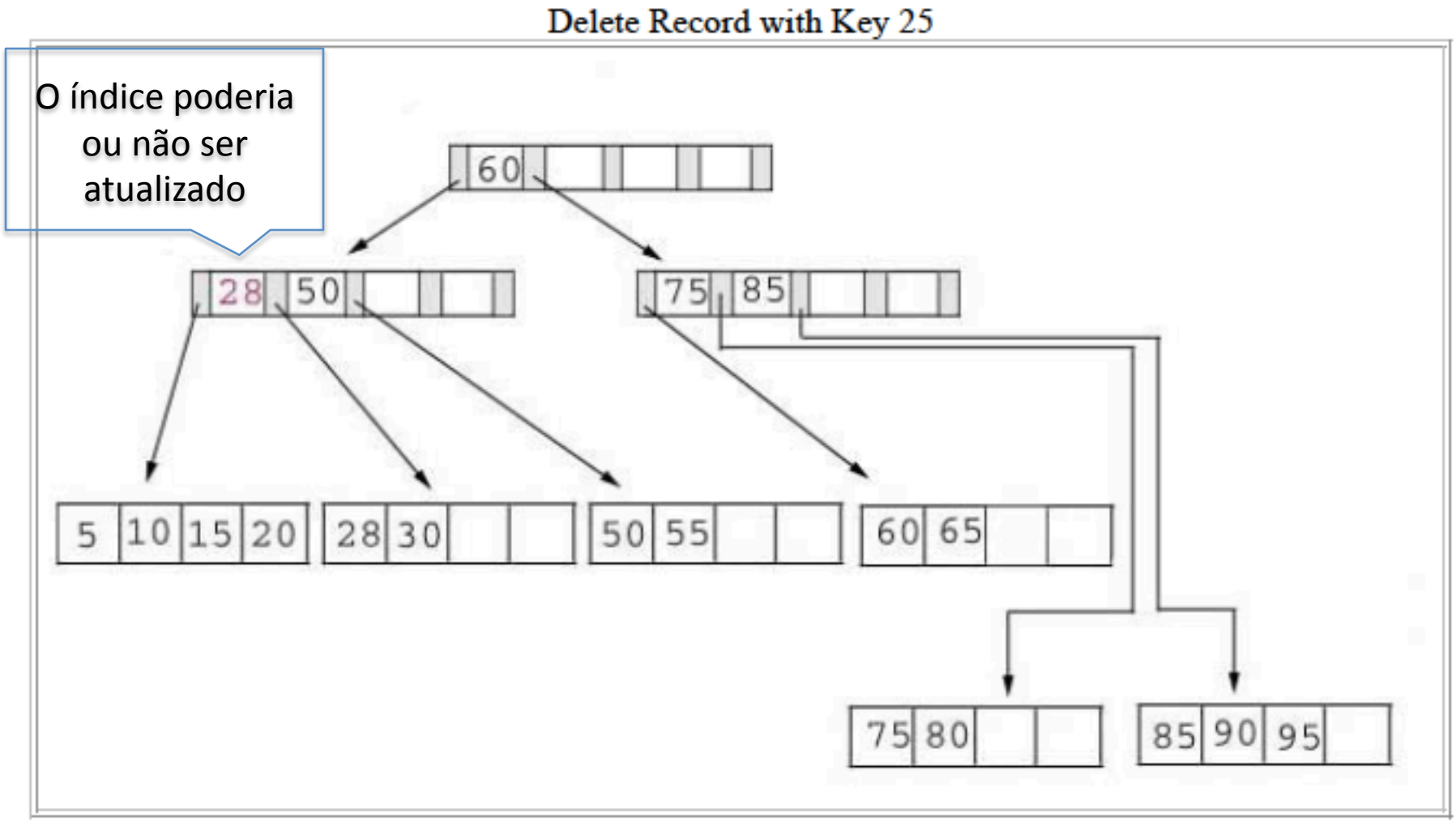
Árvores B*: inserção



Árvores B*: remoção



Árvores B*: remoção



Acesso Concorrente em Árvores B*

- Exemplo: Processos p1 e p2 acessam simultaneamente um banco de dados:
  - P1 pesquisa em qual sub-árvore está uma chave
  - P2 insere um novo registro, causando uma divisão de página
- Necessidade de criar mecanismos chamados protocolos para garantir a integridade tanto dos dados quanto da estrutura.
- ***Página segura***: não há possibilidade de modificações na estrutura da árvore como consequuência de inserção ou remoção.
  - **Inserção: página segura se o número de chaves é <2m**
  - **Remoção: página segura se o número de chaves é >m.**
- Os algoritmos para acesso concorrente fazem uso dessa propriedade para aumentar o nível de concorrência.

Acesso Concorrente em Árvore B* - Protocolos de Travamentos

- Quando uma página é lida, a operação de recuperação a trava, assim, outros processos, não podem interferir com a página.
- A pesquisa continua em direção ao nível seguinte e a trava é liberada para que outros processos possam ler a página .
- Processo leitor $\rightarrow$ executa uma operação de recuperação
- Processo modificador $\rightarrow$ executa uma operação de inserção ou retirada.
- Dois tipos de travamento:
  - Travamento para leitura $\rightarrow$ permite um ou mais leitores acessarem os dados, mas não permite inserção ou retirada.
  - Travamento exclusivo $\rightarrow$ nenhum outro processo pode operar na página e permite qualquer tipo de operação na página.

Acesso Concorrente em Árvores B*

- Protocolo para processos leitores:
  1. Coloque um travamento-leitura na raíz;
  2. Leia a página raíz e faça-a página corrente;
  3. Enquanto página corrente não é folha, faça:
     - Coloque um travamento-leitura no descendente apropriado;
     - Libere o travamento-leitura da página corrente;
     - Leia o descendente da página corrente e faça-o página corrente;
- Protocolo para processos modificadores:
  1. Coloque um travamento-exclusivo na raíz;
  2. Leia a página raíz e faça-a página corrente;
  3. Enquanto página corrente não é folha, faça:
     - Coloque um travamento-exclusivo no descendente apropriado;
     - Leia o descendente da página corrente e faça-o página corrente;
     - Se a página corrente é segura, então libere todos os travamentos mantidos sobre as páginas antecessoras da página corrente;

Árvores B: Considerações práticas

- Simples, fácil manutenção, eficiente e versátil.
- Permite acesso seqüencial eficiente.
- Custo para recuperar, inserir e retirar registros do arquivo é logaritmico.
- Espaço utilizado é, no mínimo 50% do espaço reservado para o arquivo.
- Emprego onde o acesso concorrente ao banco de dados é necessário, é viável e relativamente simples de ser implementado.
- Inserção e retirada de registros sempre deixam a árvore balanceada.
- Nunca há necessidade de reorganização completa do banco de dados!

Árvores B: Medidas de Complexidade

- O caminho mais longo em uma árvore B de ordem m com N registros contém no máximo cerca de log_{m+1} (N) páginas. (tempo de busca)
- Limites para a altura máxima e mínima de uma árvore B de ordem m com N registros:

$$\log_{2m+1}(N + 1) \leq \textit{altura} \leq 1 + \log_{m+1}((N+1)/2)$$

- Altura esperada: não é conhecida analiticamente.
- Há uma conjectura proposta a partir do cálculo analítico do número esperado de páginas para os quatro primeiros níveis (das folha em direção à raiz) de uma **árvore 2-3 (árvore B de ordem m = 1). [Ziviani et al. 1982]**
- Conjetura: a altura esperada de uma árvore 2-3 **randômica com N** chaves é (7/3~=2.33)

$$E(h(N)) \sim= \log_{7/3}(N + 1)$$

Árvores B Randômicas - Medidas de Complexidade

- A utilização de memória é cerca de $\ln 2$.
  - Páginas ocupam $\approx$ 69% da área reservada após N inserções randômicas em uma árvore B inicialmente vazia.
- No momento da inserção, a operação mais cara é a partição da página quando ela passa a ter mais do que $2m$ chaves. Envolve:
  - Criação de nova página, rearranjo das chaves e inserção da chave do meio na página pai localizada no nível acima.
  - $Pr\{$j partições$\}$: probabilidade de que j partições ocorram durante a N-ésima inserção randômica.
  - Árvore 2-3: $Pr\{$0 partições$\} = \frac{4}{7}$, $Pr\{$1 ou mais partições$\} = \frac{3}{7}$.
  - Árvore B de ordem m: $Pr\{$0 partições$\} = 1 - \frac{1}{(2\ln 2)m} + O(m^{-2})$, $Pr\{$1 ou + partições$\} = \frac{1}{(2\ln 2)m} + O(m^{-2})$.
  - Árvore B de ordem $m = 70$: 99% das vezes nada acontece em termos de partições durante uma inserção.

Árvores B Randômicas - Acesso Concorrente

- Foi proposta uma técnica de aplicar um travamento na *página segura mais profunda* (Psmp) no caminho de inserção.
- Uma página é **segura** se ela contém menos do que $2m$ chaves.
- Uma página segura é a mais profunda se não existir outra página segura abaixo dela.
- Já que o travamento da página impede o acesso de outros processos, é interessante saber qual é a probabilidade de que a página segura mais profunda esteja no primeiro nível.
- Árvore 2-3: $Pr\{\text{Psmp esteja no } 1^{\underline{o}} \text{ nível}\} = \frac{4}{7}$,
  $Pr\{\text{Psmp esteja acima do } 1^{\underline{o}} \text{ nível}\} = \frac{3}{7}$.
- Árvore B de ordem m:
  $Pr\{\text{Psmp esteja no } 1^{\underline{o}} \text{ nível}\} = 1 - \frac{1}{(2\ln 2)m} + O(m^{-2})$,
  $Pr\{\text{Psmp esteja acima do } 1^{\underline{o}} \text{ nível}\} = \frac{3}{7} = \frac{1}{(2\ln 2)m} + O(m^{-2})$.

Árvores B Randômicas - Acesso Concorrente

- Novamente, em árvores B de ordem $m = 70$: 99% das vezes a Psmp está em uma folha. (Permite alto grau de concorrência para processos modificadores.)
- Soluções muito complicadas para permitir concorrência de operações em árvores B não trazem grandes benefícios.
- Na maioria das vezes, o travamento ocorrerá em páginas folha. (Permite alto grau de concorrência mesmo para os protocolos mais simples.)

Árvore B - Técnica de Transbordamento (ou Overflow)

- Assuma que um registro tenha de ser inserido em uma página cheia, com $2m$ registros.
- Em vez de particioná-la, olhamos primeiro para a página irmã à direita.
- Se a página irmã possui menos do que $2m$ registros, um simples rearranjo de chaves torna a partição desnecessária.
- Se a página à direita também estiver cheia ou não existir, olhamos para a página irmã à esquerda.
- Se ambas estiverem cheias, então a partição terá de ser realizada.
- Efeito da modificação: produzir uma árvore com melhor utilização de memória e uma altura esperada menor.
- Produz uma utilização de memória de cerca de 83% para uma árvore B randômica.

Árvore B - Influência do Sistema de Paginação

- O número de níveis de uma árvore B é muito pequeno (três ou quatro) se comparado com o número de molduras de páginas.
- Assim, o sistema de paginação garante que a página raiz esteja sempre na memória principal (se for adotada a política LRU).
- O esquema LRU faz com que as páginas a serem particionadas em uma inserção estejam disponíveis na memória principal.
- A escolha do tamanho adequado da ordem m da árvore B é geralmente feita levando em conta as características de cada computador.
- O tamanho ideal da página da árvore corresponde ao tamanho da página do sistema, e a transferência de dados entre as memórias secundária e principal é realizada pelo sistema operacional.
- Estes tamanhos variam entre 512 *bytes* e 4.096 *bytes*, em múltiplos de 512 *bytes*.