
Pesquisa em Memória Secundária*

Última alteração: 17 de Outubro de 2006

*Transparências elaboradas por Wagner Meira Jr, Flávia Peligrinelli Ribeiro, Nívio Ziviani e Charles Ornelas Almeida

Introdução

- **Pesquisa em memória secundária:** arquivos contém mais registros do que a memória interna pode armazenar.
- Custo para acessar um registro é algumas ordens de grandeza maior do que o custo de processamento na memória primária.
- Medida de complexidade: custo de transferir dados entre a memória principal e secundária (minimizar o número de transferências).
- Memórias secundárias: apenas um registro pode ser acessado em um dado momento (acesso seqüencial).
- Memórias primárias: acesso a qualquer registro de um arquivo a um custo uniforme (acesso direto).
- Em um método eficiente de pesquisa, o aspecto sistema de computação é importante.
- As características da arquitetura e do sistema operacional da máquina tornam os métodos de pesquisa dependentes de parâmetros que afetam seus desempenhos.

Modelo de Computação para Memória Secundária - Memória Virtual

- Normalmente implementado como uma função do sistema operacional.
- Modelo de armazenamento em dois níveis, devido à necessidade de grandes quantidades de memória e o alto custo da memória principal.
- Uso de uma pequena quantidade de memória principal e uma grande quantidade de memória secundária.
- Programador pode endereçar grandes quantidades de dados, deixando para o sistema a responsabilidade de transferir o dado da memória secundária para a principal.
- Boa estratégia para algoritmos com pequena localidade de referência.
- Organização do fluxo entre a memória principal e secundária é extremamente importante.

Memória Virtual

- Organização de fluxo → transformar o endereço usado pelo programador na localização física de memória correspondente.
- *Espaço de Endereçamento* → endereços usados pelo programador.
- *Espaço de Memória* → localizações de memória no computador.
- O espaço de endereçamento N e o espaço de memória M podem ser vistos como um mapeamento de endereços do tipo:
 $f : N \rightarrow M$.
- O mapeamento permite ao programador usar um espaço de endereçamento que pode ser maior que o espaço de memória primária disponível.

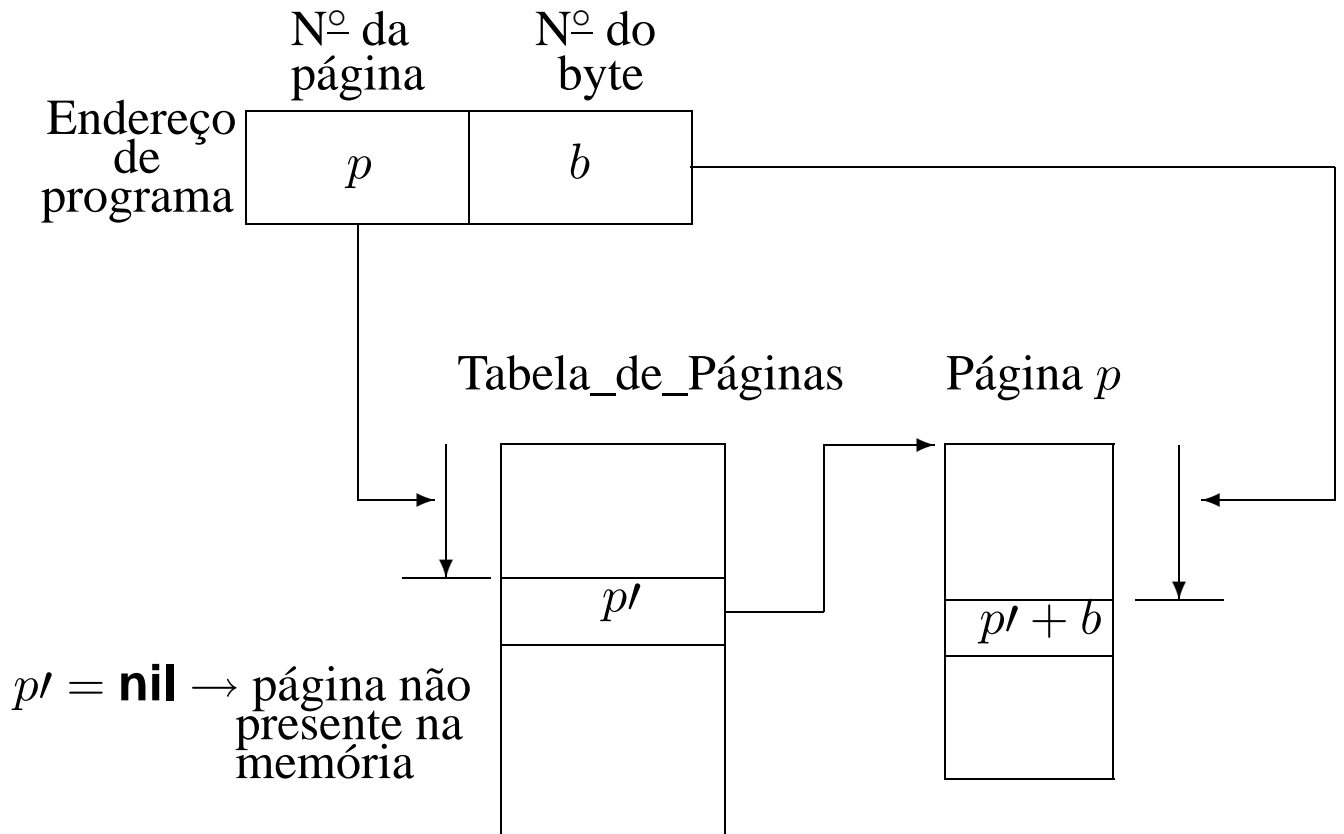
Memória Virtual: Sistema de Paginação

- O espaço de endereçamento é dividido em páginas de tamanho igual, em geral, múltiplos de 512 Kbytes.
- A memória principal é dividida em molduras de páginas de tamanho igual.
- As molduras de páginas contêm algumas páginas ativas enquanto o restante das páginas estão residentes em memória secundária (páginas inativas).
- O mecanismo possui duas funções:
 1. Mapeamento de endereços → determinar qual página um programa está endereçando, encontrar a moldura, se existir, que contenha a página.
 2. Transferência de páginas → transferir páginas da memória secundária para a memória primária e transferí-las de volta para a memória secundária quando não estão mais sendo utilizadas.

Memória Virtual: Sistema de Paginação

- Endereçamento da página → uma parte dos bits é interpretada como um número de página e a outra parte como o número do byte dentro da página (*offset*).
- Mapeamento de endereços → realizado através de uma Tabela de Páginas.
 - a p -ésima entrada contém a localização p' da Moldura de Página contendo a página número p desde que esteja na memória principal.
- O mapeamento de endereços é:
 $f(e) = f(p, b) = p' + b$, onde e é o endereço do programa, p é o número da página e b o número do byte.

Memória Virtual: Mapeamento de Endereços



Memória Virtual: Reposição de Páginas

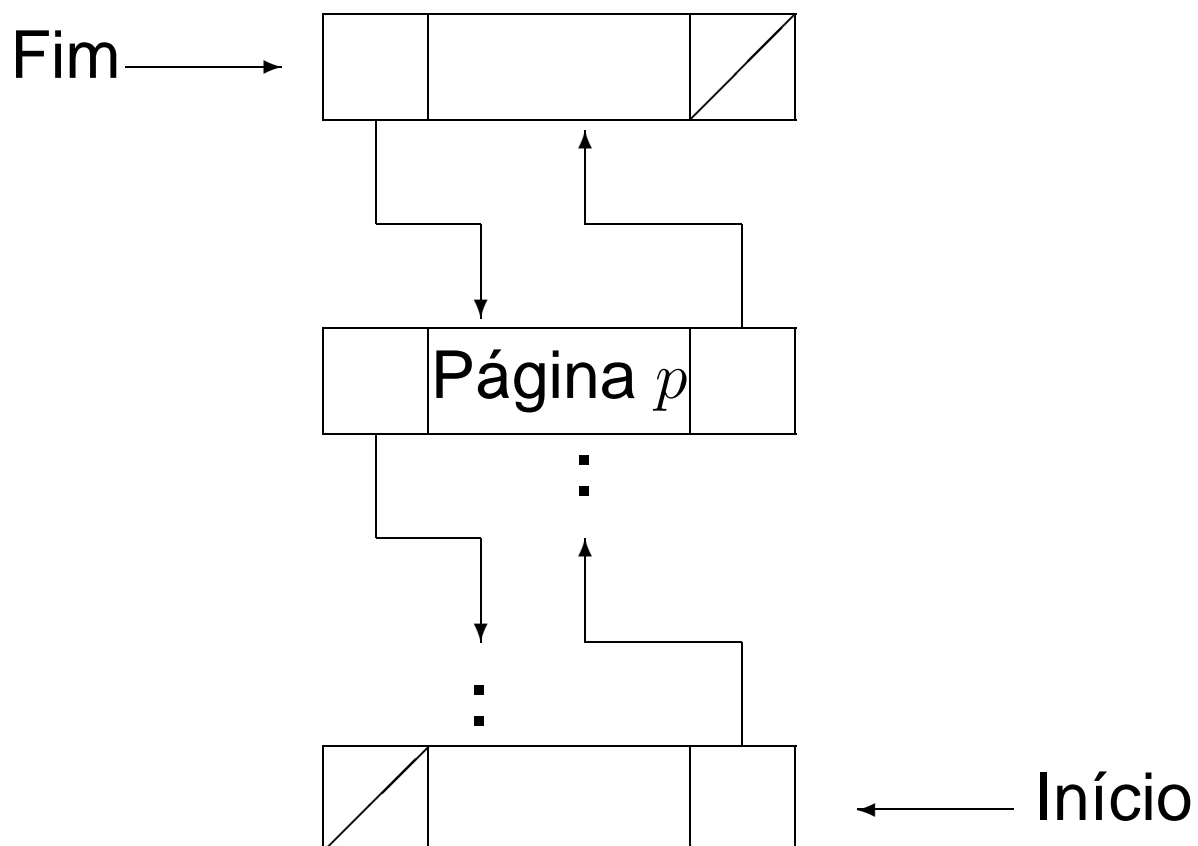
- Se não houver uma moldura de página vazia → uma página deverá ser removida da memória principal.
- Ideal → remover a página que não será referenciada pelo período de tempo mais longo no futuro.
 - tentamos inferir o futuro a partir do comportamento passado.

Memória Virtual: Políticas de Reposição de Páginas

- **Menos Recentemente Utilizada (LRU):**
 - um dos algoritmos mais utilizados,
 - remove a página menos recentemente utilizada,
 - parte do princípio que o comportamento futuro deve seguir o passado recente.
- **Menos Frequentemente Utilizada (LFU):**
 - remove a página menos frequentemente utilizada,
 - inconveniente: uma página recentemente trazida da memória secundária tem um baixo número de acessos registrados e pode ser removida.
- **Ordem de Chegada (FIFO):**
 - remove a página que está residente há mais tempo,
 - algoritmo mais simples e barato de manter,
 - desvantagem: ignora o fato de que a página mais antiga pode ser a mais referenciada.

Memória Virtual: Política LRU

- Toda vez que uma página é utilizada ela é removida para o fim da fila.
- A página que está no início da fila é a página LRU.
- Quando uma nova página é trazida da memória secundária ela deve ser colocada na moldura que contém a página LRU.



Memória Virtual: Estrutura de Dados

const

TamANHodaPagina = 512;

ItensPorPagina = 64; { *TamANHodaPagina/TamANHodoItem* }

type Registro = **record**

Chave: TipoChave;

{ *outros componentes* }

end;

EnderecoTipo = **record**

p: **integer**;

b: 1..ItensPorPagina;

end;

ItemTipo = **record**

Reg: Registro;

Esq, Dir: EnderecoTipo;

end;

PaginaTipo = **array** [1..ItensPorPagina] **of** ItemTipo;

Memória Virtual

- Em casos em que precisamos manipular mais de um arquivo ao mesmo tempo:
 - A tabela de páginas para cada arquivo pode ser declarada separadamente.
 - A fila de molduras é única → cada moldura deve ter indicado o arquivo a que se refere aquela página.

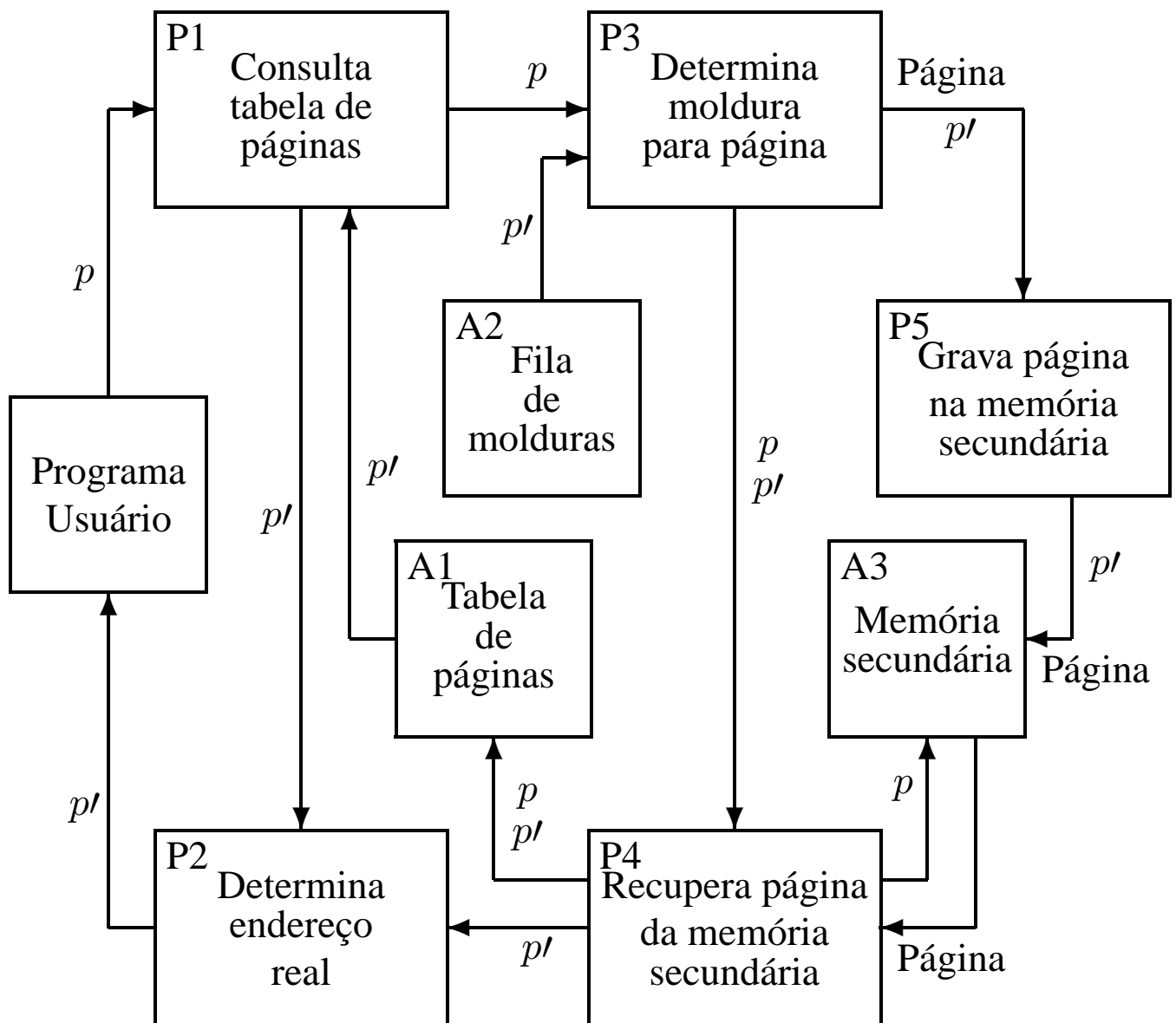
```
type PaginaTipo = record  
    case byte of  
        0: (Pa: PaginaTipoA);  
        1: (Pb: PaginaTipoB);  
        2: (Pc: PaginaTipoC);  
end;
```

Memória Virtual

- Procedimentos para comunicação com o sistema de paginação:
 - ObtemRegistro → torna disponível um registro.
 - EscreveRegistro → permite criar ou alterar o conteúdo de um registro.
 - DescarregaPaginas → varre a fila de molduras para atualizar na memória secundária todas as páginas que tenham sido modificadas.

Memória Virtual - Transformação do Endereço Virtual para Real

- Quadrados → resultados de processos ou arquivos.
- Retângulos → processos transformadores de informação.



Acesso Seqüencial Indexado

- Utiliza o princípio da pesquisa seqüencial → 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 $\langle x, p \rangle$ 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

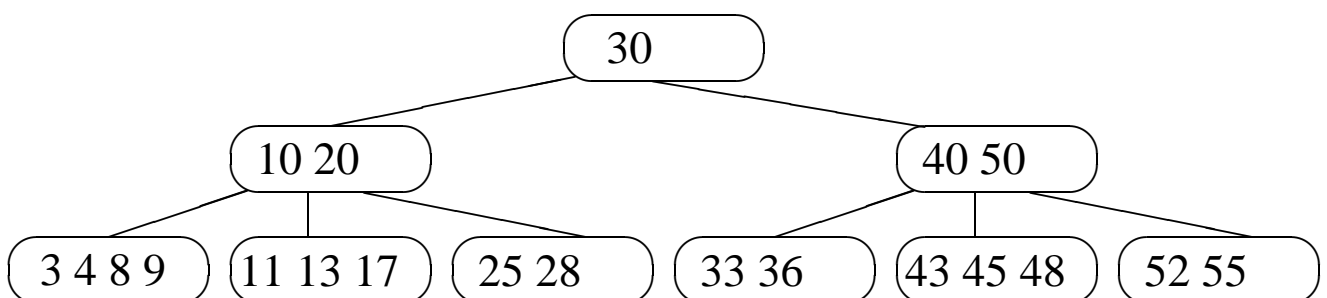
- Dividido em círculos concêntricos (trilhas).
- Cilindro → todas as trilhas verticalmente alinhadas e que possuem o mesmo diâmetro.
- Latência rotacional → tempo necessário para que o início do bloco contendo o registro a ser lido passe pela cabeça de leitura/gravação.
- Tempo de busca (*seek time*)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

Acesso Seqüencial Indexado: Discos Óticos de Apenas-Leitura (CD-ROM)

- Grande capacidade de armazenamento (600 MB) e baixo custo para o usuário final.
- Informação armazenada é estática.
- A eficiência na recuperação dos dados é afetada pela localização dos dados no disco e pela seqüência com que são acessados.
- Velocidade linear constante → trilhas possuem capacidade variável e tempo de latência rotacional varia de trilha para trilha.
- A trilha tem forma de uma espiral contínua.
- Tempo de busca: acesso a trilhas mais distantes demanda mais tempo que no disco magnético. Há necessidade de deslocamento do mecanismo de acesso e mudanças na rotação do disco.
- Varredura estática: acessa conjunto de trilhas vizinhas sem deslocar mecanismo de leitura.
- Estrutura seqüencial implementada mantendo-se um índice de cilindros na memória principal.

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.
- Os registros aparecem 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:

```

type Registro = record
    Chave: TipoChave;
    {outros componentes}
end;

Apontador = ^Pagina;

Pagina = record
    n: 0..mm;
    r: array [1..mm] of Registro;
    p: array [0..mm] of Apontador
end;

TipoDicionario = Apontador;

```

- Operações:

- Inicializa

```

procedure Inicializa (var Dicionario: TipoDicionario);
begin
    Dicionario := nil;
end;

```

- Pesquisa

- Insere

- Remove

Árvores B - Pesquisa

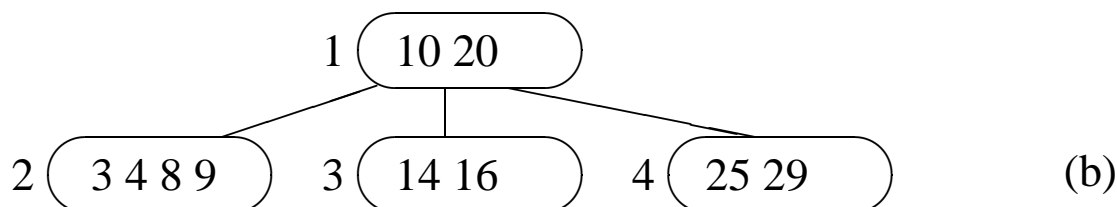
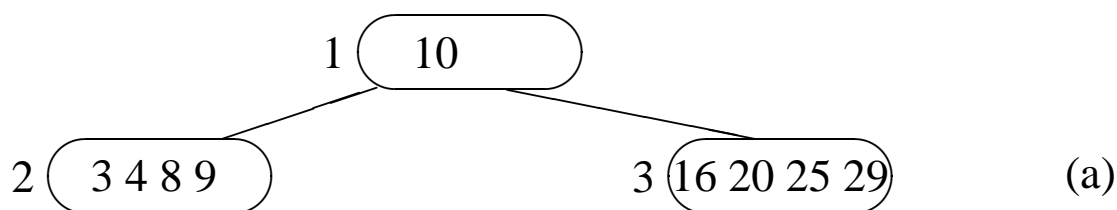
- Semelhante ao algoritmo de pesquisa para árvores binárias.

```
procedure Pesquisa (var x: Registro; Ap: Apontador);  
var i: Integer;  
begin  
    if Ap = nil  
    then writeln ('Registro nao esta presente na arvore')  
    else with Ap^ do  
        begin  
            i := 1;  
            while (i < n) and (x.Chave > r[i].Chave) do  
                i := i + 1;  
            if x.Chave = r[i].Chave  
            then x := r[i]  
            else if x.Chave < r[i].Chave  
                then Pesquisa (x, p[i-1])  
                else Pesquisa (x, p[i])  
        end;  
end;
```

Árvores B - Inserção

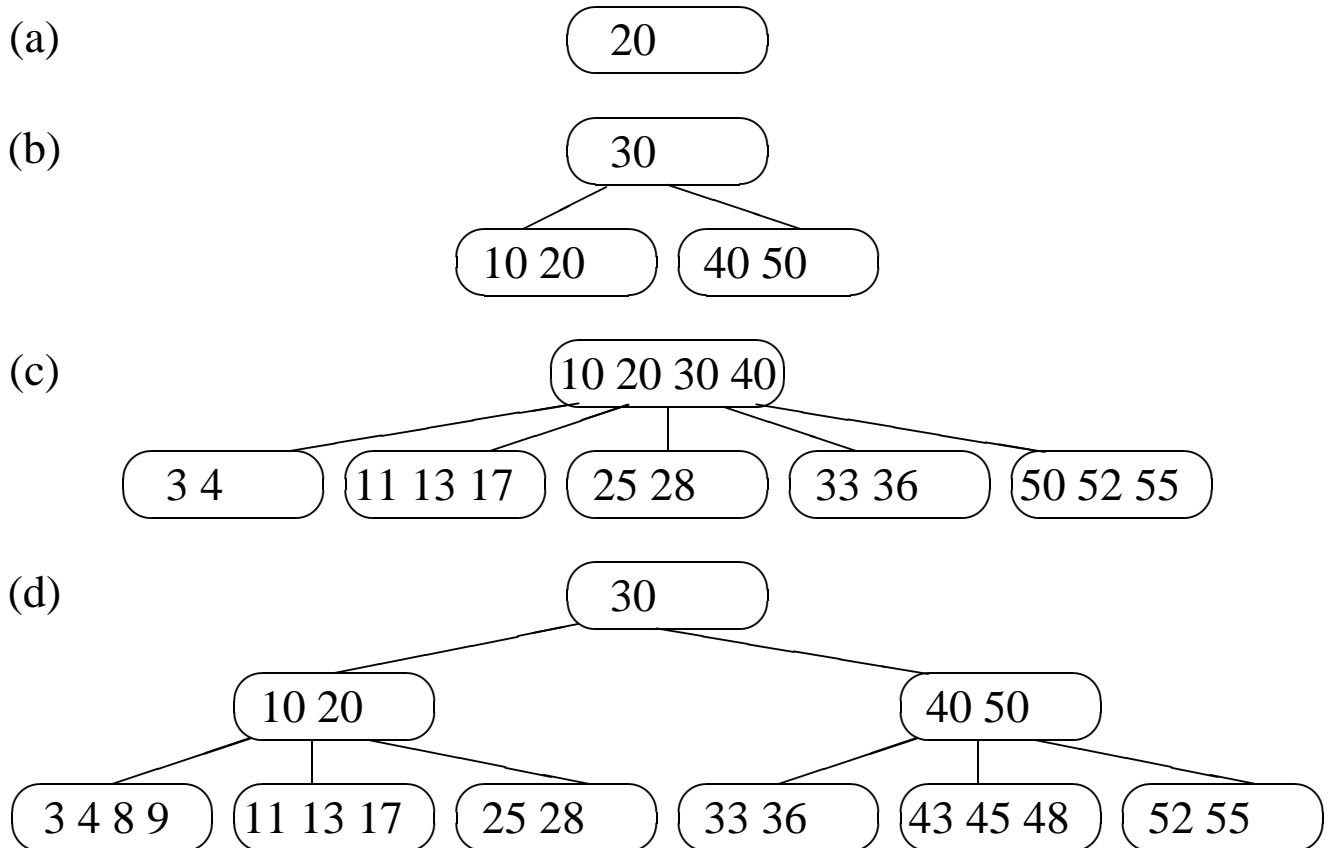
1. Localizar a página apropriada aonde o registro deve ser inserido.
2. Se o registro a ser inserido encontra uma página com menos de $2m$ registros, o processo de inserção fica limitado à página.
3. Se o registro a ser inserido encontra uma página cheia, é criada uma nova página, no caso da página pai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

Exemplo: Inserindo o registro com chave 14.



Árvores B - Inserção

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 - Primeiro refinamento do algoritmo Insere na árvore B

```

procedure Insere (Reg: Registro; var Ap: Apontador);

  procedure Ins (Reg: Registro; Ap: Apontador; var Cresceu: Boolean;
                var RegRetorno: Registro; var ApRetorno: Apontador);
  var i: integer;
  begin
    if Ap = nil
    then begin
      Cresceu := true;
      Atribui Reg a RegRetorno;
      Atribui nil a ApRetorno;
    end
    else with Ap^ do
      begin
        i := 1;
        while (i < n) and (x.Chave > r[i].Chave) do i := i + 1;
        if x.Chave = r[i].Chave
        then writeln ('Erro: Registro ja esta presente na arvore')
        else if x.Chave < r[i].Chave
          then Ins (x, p[i-1], Cresceu, RegRetorno, ApRetorno)
          else Ins (x, p[i], Cresceu, RegRetorno, ApRetorno);
        if Cresceu
        then if (Numero de registros em Ap) < mm
          then Insere na pagina Ap e Cresceu := false
          else begin { Overflow: pagina tem que ser dividida }
            Cria nova pagina ApTemp;
            Transfere metade dos registros de Ap para ApTemp;
            Atribui registro do meio a RegRetorno;
            Atribui ApTemp a ApRetorno;
          end;
        end;
      end;
    end;

  begin {Insere}
  Ins (Reg, Ap, Cresceu, RegRetorno, ApRetorno);
  if Cresceu then Cria nova pagina raiz para RegRetorno e ApRetorno;
  end;

```

Árvores B - Procedimento InsereNaPágina

```
procedure InsereNaPagina (Ap: Apontador;  
                          Reg: Registro; ApDir: Apontador);  
var NaoAchouPosicao: Boolean;  
    k: Integer;  
begin  
with Ap^ do  
    begin  
        k := n;  
        NaoAchouPosicao := k > 0;  
        while NaoAchouPosicao do  
            if Reg.Chave < r[k].Chave  
            then begin  
                r[k+1] := r[k]; p[k+1] := p[k];  
                k := k - 1;  
                if k < 1 then NaoAchouPosicao := false;  
            end  
            else NaoAchouPosicao := false;  
            r[k+1] := Reg; p[k+1] := ApDir;  
            n := n + 1;  
        end;  
end;
```

Árvores B - Refinamento final do algoritmo Insere

```
procedure Insere (Reg: Registro; var Ap: Apontador);
var Cresceu: Boolean; RegRetorno: Registro; ApRetorno, ApTemp:Apontador;
```

```
    procedure Ins (Reg: Registro; Ap: Apontador; var Cresceu: Boolean;
        var RegRetorno: Registro; var ApRetorno: Apontador);
    var i, j: Integer; ApTemp: Apontador;
    begin
```

```
        if Ap = nil
        then begin Cresceu := true; RegRetorno := Reg; ApRetorno:=nil; end
        else with Ap^ do
```

```
            begin
```

```
                i := 1;
```

```
                while (i < n) and (Reg.Chave > r[i].Chave) do i := i + 1;
```

```
                if Reg.Chave = r[i].Chave
```

```
                then begin
```

```
                    writeln('Erro: Registro ja esta presente'); Cresceu:=false;
```

```
                    end
```

```
                else begin
```

```
                    if Reg.Chave < r[i].Chave
```

```
                    then i := i - 1;
```

```
                    Ins (Reg, p[i], Cresceu, RegRetorno, ApRetorno);
```

```
                    if Cresceu
```

```
                    then if n < mm
```

```
                        then begin { Pagina tem espaco }
```

```
                            InsereNaPagina (Ap, RegRetorno, ApRetorno);
```

```
                            Cresceu := false;
```

```
                        end
```

```
{— Continua na próxima transparência —}
```

Árvores B - Refinamento final do algoritmo Insere

```

    else begin { Overflow: Pagina tem que ser dividida }
      new (ApTemp);
      ApTemp^.n := 0; ApTemp^.p[0] := nil;
      if i < m+1
      then begin
        InseNaPagina (ApTemp, r[mm], p[mm]);
        n := n - 1;
        InseNaPagina (Ap, RegRetorno, ApRetorno)
      end
      else InseNaPagina (ApTemp, RegRetorno, ApRetorno);
      for j := m + 2 to mm do
        InseNaPagina (ApTemp, r[j], p[j]);
      n := m; ApTemp^.p[0] := p[m+1];
      RegRetorno := r[m+1]; ApRetorno := ApTemp;
      end;
    end;
  end;
end; { Ins }
begin
  Ins (Reg, Ap, Cresceu, RegRetorno, ApRetorno);
  if Cresceu
  then begin { Arvore cresce na altura pela raiz }
    new (ApTemp);
    ApTemp^.n := 1;
    ApTemp^.r[1] := RegRetorno;
    ApTemp^.p[1] := ApRetorno;
    ApTemp^.p[0] := Ap; Ap := ApTemp
  end
end; { Insere }

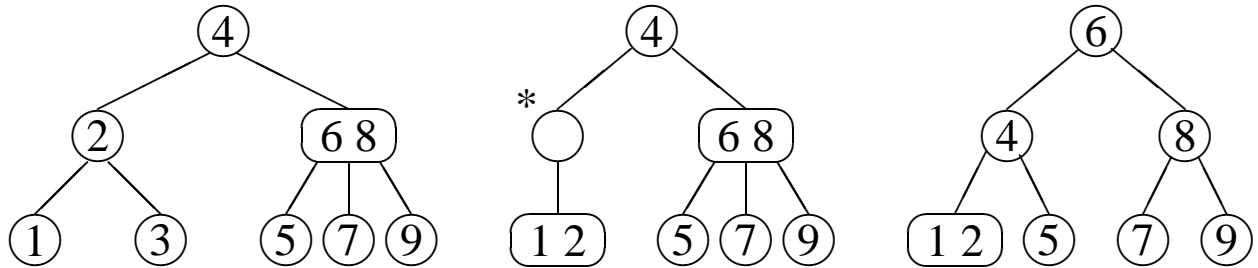
```

Árvores B - Remoção

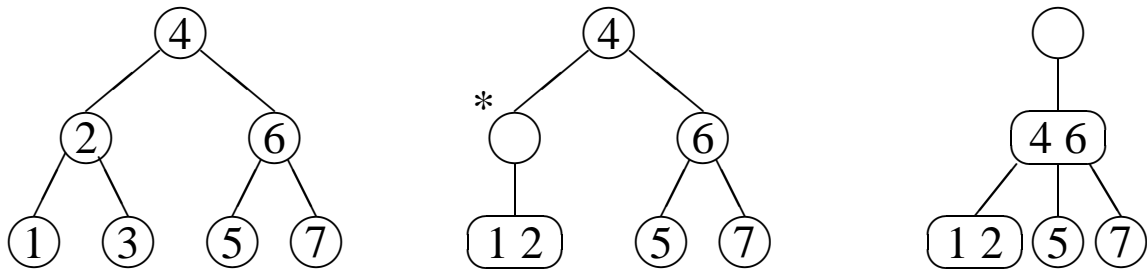
- Página com o registro a ser retirado é folha:
 1. retira-se o registro,
 2. se a página não possui pelo menos de m registros, a propriedade da árvore B é violada. Pega-se um registro emprestado da página vizinha. Se não existir registros suficientes na página vizinha, as duas páginas devem ser fundidas 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

Exemplo: Retirando a chave 3.



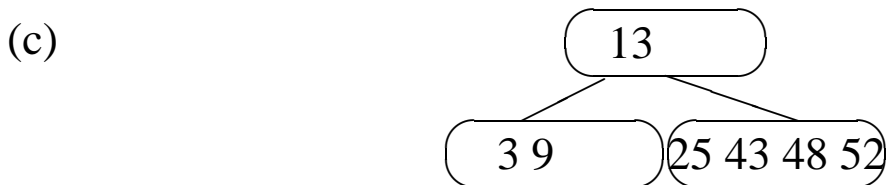
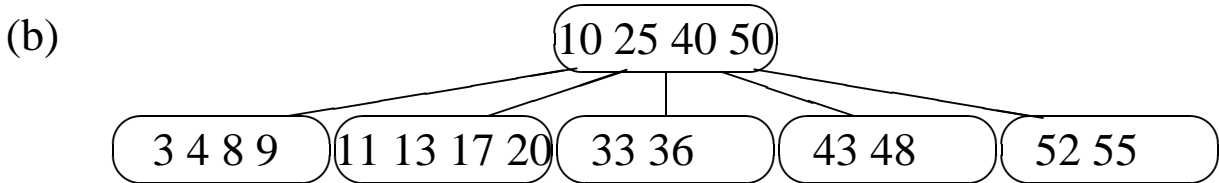
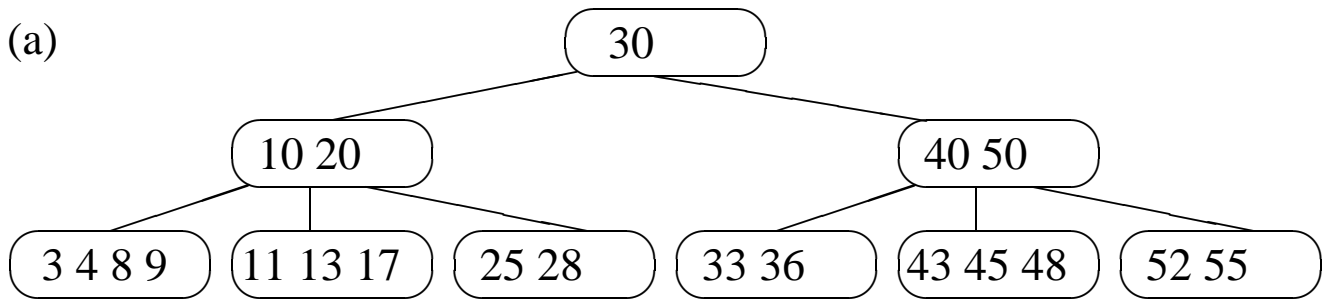
(a) Página vizinha possui mais do que m registros



(b) Página vizinha possui exatamente m registros

Árvores B - Remoção

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



Árvores B - Procedimento Retira

```

procedure Retira (Ch: TipoChave; var Ap: Apontador);
var Diminuiu: Boolean;
    Aux      : Apontador;
procedure Ret(Ch:TipoChave; var Ap:Apontador; var Diminuiu:Boolean);
var Ind, j: Integer;
    procedure Reconstitui (ApPag: Apontador; ApPai: Apontador;
                          PosPai: Integer; var Diminuiu: Boolean);
var Aux      : Apontador;
    DispAux, j: Integer;
begin
    if PosPai < ApPai^.n
    then begin { Aux = Pagina a direita de ApPag }
        Aux := ApPai^.p[PosPai+1];
        DispAux := (Aux^.n – m + 1) div 2;
        ApPag^.r[ApPag^.n+1] := ApPai^.r[PosPai+1];
        ApPag^.p[ApPag^.n+1] := Aux^.p[0];
        ApPag^.n := ApPag^.n + 1;
        if DispAux > 0
        then begin { Existe folga: transfere de Aux para ApPag }
            for j := 1 to DispAux – 1 do
                InseNaPagina (ApPag, Aux^.r[j], Aux^.p[j]);
            ApPai^.r[PosPai+1] := Aux^.r[DispAux];
            Aux^.n := Aux^.n – DispAux;
            for j := 1 to Aux^.n do Aux^.r[j] := Aux^.r[j+DispAux];
            for j := 0 to Aux^.n do Aux^.p[j] := Aux^.p[j+DispAux];
            Diminuiu := false
        end
    else begin { Fusao: intercala Aux em ApPag e libera Aux }
        for j := 1 to m do
            InseNaPagina (ApPag, Aux^.r[j], Aux^.p[j]);
        dispose (Aux);
        for j := PosPai + 1 to ApPai^.n – 1 do with ApPai^ do
            begin
                r[j] := r[j+1];
                p[j] := p[j+1]
            end;
    end

```

{— Continua na próxima transparência —}

Árvores B - Procedimento Retira

```

    ApPai.n := ApPai.n - 1;
    if ApPai.n >= m then Diminuiu := false;
    end
end
else begin { Aux = Pagina a esquerda de ApPag }
    Aux := ApPai.p[PosPai-1];
    DispAux := (Aux.n - m + 1) div 2;
    for j := ApPag.n downto 1 do
        ApPag.r[j+1] := ApPag.r[j];
    ApPag.r[1] := ApPai.r[PosPai];
    for j := ApPag.n downto 0 do
        ApPag.p[j+1] := ApPag.p[j];
    ApPag.n := ApPag.n + 1;
    if DispAux > 0
    then begin { Existe folga: transfere de Aux para ApPag }
        for j := 1 to DispAux - 1 do with Aux. do
            InereNaPagina (ApPag, r[Aux.n+1-j], p[n+1-j]);
        ApPag.p[0] := Aux.p[Aux.n+1-DispAux];
        ApPai.r[PosPai] := Aux.r[Aux.n+1-DispAux];
        Aux.n := Aux.n - DispAux;
        Diminuiu := false
        end
    else begin { Fusao: intercala ApPag em Aux e libera ApPag }
        for j := 1 to m do
            InereNaPagina (Aux, ApPag.r[j], ApPag.p[j]);
        dispose (ApPag);
        ApPai.n := ApPai.n - 1;
        if ApPai.n >= m then Diminuiu := false;
        end;
    end;
end; { Reconstitui }
{— Continua na próxima transparência —}

```

Árvores B - Procedimento Retira

```
procedure Antecessor (Ap: Apontador; Ind: Integer;  
                    ApPai: Apontador;  
                    var Diminuiu: Boolean);  
  
begin  
with ApPai^ do  
  begin  
    if p[n] <> nil  
    then begin  
      Antecessor (Ap, Ind, p[n], Diminuiu);  
      if Diminuiu then Reconstitui (p[n], ApPai, n, Diminuiu);  
    end  
  else begin  
    Ap^.r[Ind] := r[n];  
    n := n - 1;  
    Diminuiu := n < m;  
  end;  
  
  end  
end; { Antecessor }
```

{— Continua na próxima transparência —}

Árvores B - Procedimento Retira

```

begin { Ret }
  if Ap = nil
  then begin
    writeln ( 'Erro: registro nao esta na arvore' );
    Diminuiu := false;
  end
  else with Ap^ do
    begin
      Ind := 1;
      while (Ind < n) and (Ch > r[Ind].Chave) do Ind := Ind + 1;
      if Ch = r[Ind].Chave
      then if p[Ind-1] = nil
            then begin { Pagina folha }
                  n := n-1; Diminuiu := n<m;
                  for j := Ind to n do
                    begin
                      r[j] := r[j+1];
                      p[j] := p[j+1];
                    end;
                  end
            else begin { Pagina nao e folha: trocar com antecessor }
                  Antecessor (Ap, Ind, p[Ind-1], Diminuiu);
                  if Diminuiu
                  then Reconstitui (p[Ind-1], Ap, Ind-1, Diminuiu);
                  end
            else begin
                  if Ch > r[Ind].Chave then Ind := Ind + 1;
                  Ret (Ch, p[Ind-1], Diminuiu);
                  if Diminuiu
                  then Reconstitui (p[Ind-1], Ap, Ind-1, Diminuiu);
                  end
            end
      end
    end; { Ret }
  {— Continua na próxima transparência —}

```

Árvores B - Procedimento Retira

```
begin { Retira }  
  Ret (Ch, Ap, Diminuiu);  
  if Diminuiu and (Ap.n = 0)  
  then begin { Arvore diminuir na altura }  
    Aux := Ap; Ap := Aux.p[0];  
    dispose (Aux);  
  end  
end; { Retira }
```

Árvores B* - TAD Dicionário

- Estrutura de Dados:

type

Registro = **record**

 Chave: TipoChave;

 { *outros componentes* }

end;

Apontador = ^Pagina;

PaginaTipo = (Interna, Externa);

Pagina = **record**

case Pt: PaginaTipo **of**

 Interna: (ni: 0..mm;

 ri: **array** [1..mm] **of** TipoChave;

 pi: **array** [0..mm] **of** Apontador);

 Externa: (ne: 0..mm2;

 re: **array** [1..mm2] **of** Registro);

end;

TipoDicionario = Apontador;

Árvores B* - Pesquisa

- Semelhante à pesquisa em árvore B,
- A pesquisa sempre leva a uma página folha,
- A pesquisa não pára se a chave procurada for encontrada em uma página índice. O apontador da direita é seguido até que se encontre uma página folha.

Árvores B* - Procedimento para pesquisar na árvore B*

```
procedure Pesquisa (var x: Registro; var Ap: Apontador);
var i: integer;
begin
  if Ap^.Pt = Interna
  then with Ap^ do
    begin
      i := 1;
      while (i < ni) and (x.Chave > ri[i]) do i := i + 1;
      if x.Chave < ri[i]
      then Pesquisa(x, pi[i-1])
      else Pesquisa(x, pi[i])
      end
    else with Ap^ do
      begin
        i := 1;
        while (i < ne) and (x.Chave > re[i].Chave) do i := i + 1;
        if x.Chave = re[i].Chave
        then x := re[i]
        else writeln('Registro nao esta presente na arvore');
      end;
    end;
end;
```

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.

Acesso Concorrente em Árvore B*

- Acesso simultâneo a banco de dados por mais de um usuário.
- Concorrência aumenta a utilização e melhora o tempo de resposta do sistema.
- O uso de árvores B* nesses sistemas deve permitir o processamento simultâneo de várias solicitações diferentes.
- 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 consequência de inserção ou remoção.
 - inserção → página segura se o número de chaves é igual a $2m$,
 - remoção → página segura se o número de chaves é maior que 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 → executa uma operação de recuperação
- Processo modificador → executa uma operação de inserção ou retirada.
- Dois tipos de travamento:
 - Travamento para leitura → permite um ou mais leitores acessarem os dados, mas não permite inserção ou retirada.
 - Travamento exclusivo → nenhum outro processo pode operar na página e permite qualquer tipo de operação na página.

Árvore 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.
- Uma árvore B de ordem m com N registros contém no máximo cerca de $\log_{m+1}N$ páginas.

Árvore B - Considerações Práticas

- Limites para a altura máxima e mínima de uma árvore B de ordem m com N registros:
$$\log_{2m+1}(N + 1) \leq altura \leq 1 + \log_{m+1}\left(\frac{N+1}{2}\right)$$
- Custo para processar uma operação de recuperação de um registro cresce com o logaritmo base m do tamanho do arquivo.
- 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 folhas em direção à raiz) de uma **árvore 2-3** (árvore B de ordem $m = 1$).
- Conjetura: a altura esperada de uma árvore **2-3 randômica** com N chaves é
$$\bar{h}(N) \approx \log_{7/3}(N + 1).$$

Árvores B Randômicas - Outras 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.
 - $Pr\{j \text{ partições}\}$: probabilidade de que j partições ocorram durante a N -ésima inserção randômica.
 - Árvore 2-3: $Pr\{0 \text{ partições}\} = \frac{4}{7}$,
 $Pr\{1 \text{ ou mais partições}\} = \frac{3}{7}$.
 - Árvore B de ordem m :
 $Pr\{0 \text{ partições}\} = 1 - \frac{1}{(2 \ln 2)^m} + O(m^{-2})$,
 $Pr\{1 \text{ ou + partições}\} = \frac{1}{(2 \ln 2)^m} + O(m^{-2})$.
 - 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* (P_{sm}p) 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: $Pr\{\text{P}_{sm}p \text{ esteja no } 1^{\circ} \text{ nível}\} = \frac{4}{7}$,
 $Pr\{\text{P}_{sm}p \text{ esteja acima do } 1^{\circ} \text{ nível}\} = \frac{3}{7}$.

- Árvore B de ordem m :

$$Pr\{\text{P}_{sm}p \text{ esteja no } 1^{\circ} \text{ nível}\} =$$

$$1 - \frac{1}{(2 \ln 2)^m} + O(m^{-2}),$$

$$Pr\{\text{P}_{sm}p \text{ esteja acima do } 1^{\circ} \text{ nível}\} = \frac{3}{7} =$$

$$\frac{1}{(2 \ln 2)^m} + O(m^{-2}).$$

Árvores B Randômicas - Acesso Concorrente

- Novamente, em árvores B de ordem $m = 70$: 99% das vezes a Psmmp 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 também com que as páginas a serem particionadas em uma inserção estejam automaticamente 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*.