GBC053 – Gerenciamento de Banco de Dados
Índices baseados em Árvores

Ilmério Reis da Silva
ilmerio@facom.ufu.br
www.facom.ufu.br/~ilmerio/gbd
UFU/FACOM/BCC

ROTEIRO

- *Fundamentos*
- *Estrutura estática (ISAM)*
- *Estrutura dinâmica (Árvores B+)*
- *Exercícios*

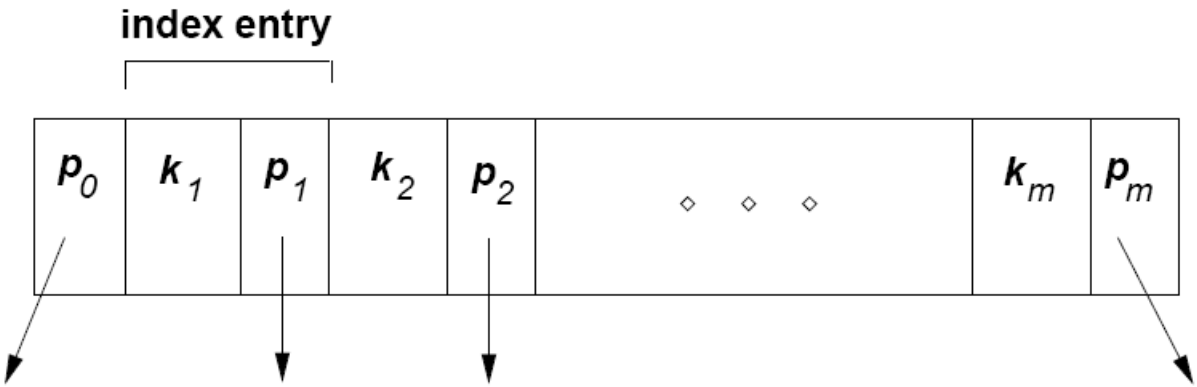
Fundamentos

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*

Formato dos nodos não-folha

Nós não folha com Entradas de Índice, "IE-index entry", do tipo : <K_i, P_i>



Formato das folhas

- *Folhas com Entrada de Dados, "DE-data entries", com três alternativas:*
  1. *Registro: <..., k, ...>*
  2. *Chave + identificador do registro: <k, rid>, onde*

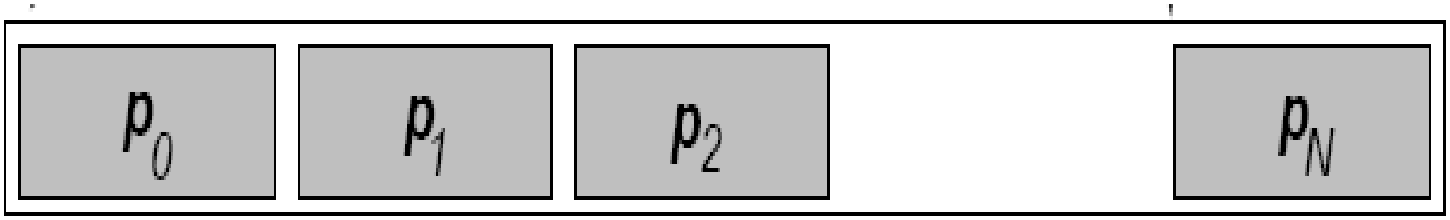
     o rid=<#page_id, #slot_id>

     identifica a página e o slot onde está localizado o registro no arquivo de dados
  3. *Chave + lista de identificadores de registros: <k, lista_rids>*

Estrutura estática (ISAM)

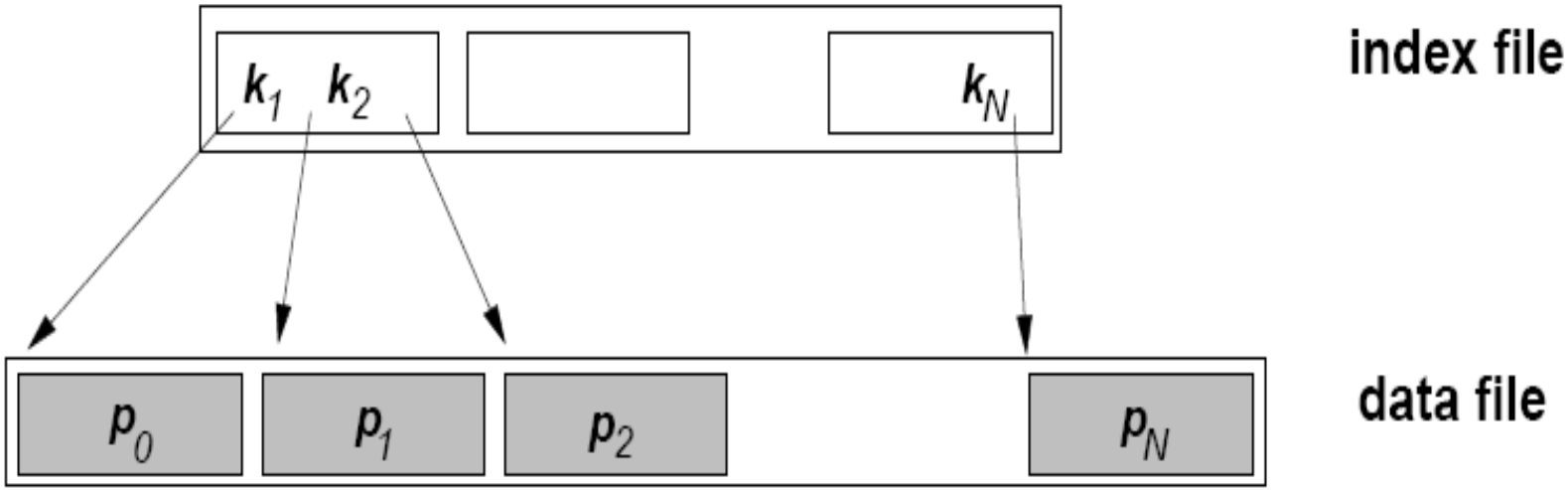
ISAM - Motivação

- *Motivação: melhorar a busca binária do arquivo ordenado*
- *Idéia: Diminuir o número de páginas da pesquisa*
- *Ilustração:*
  - *Seja um arquivo ordenado com B=N+1 páginas*
  - *Busca binária direta no arquivo ordenado tem Custo_IO=$log_2 B$*



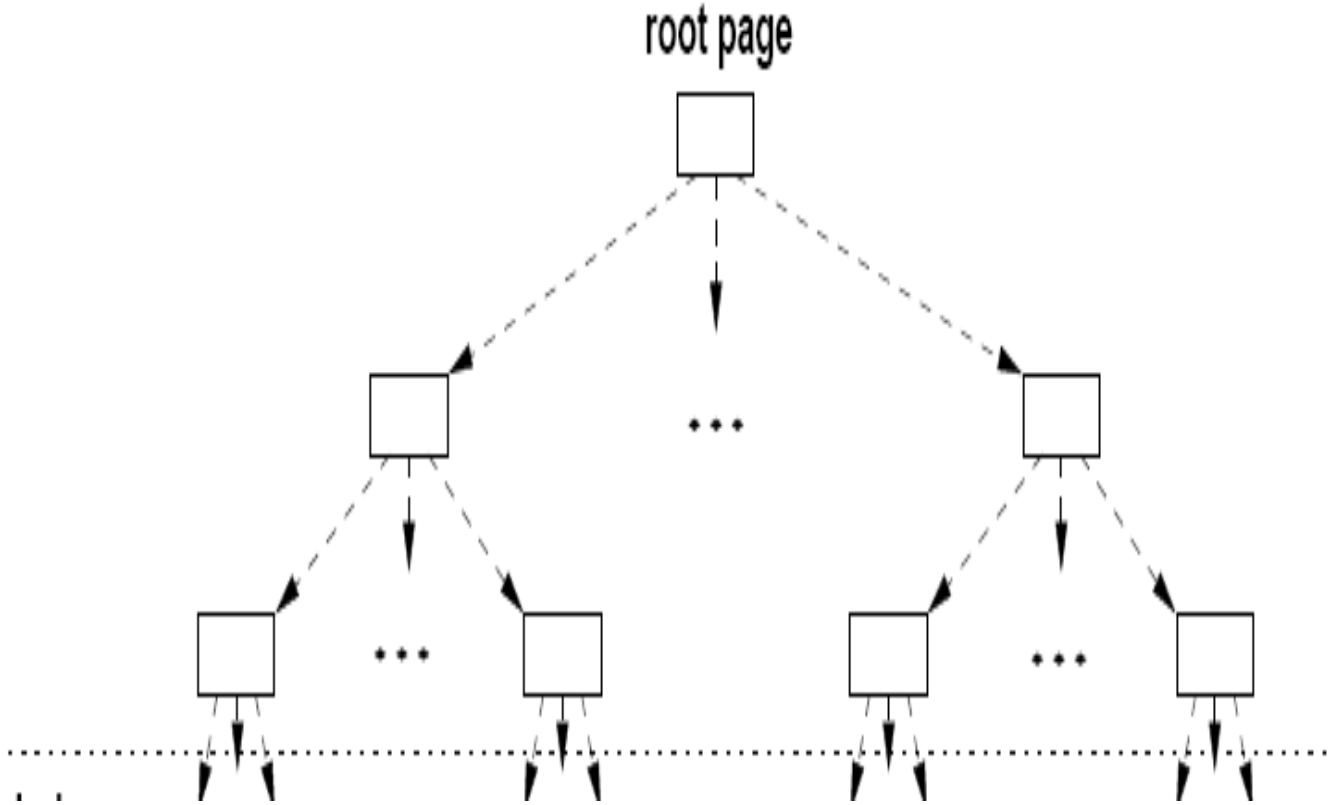
ISAM – um nível

- *Vamos criar um nível de índice(esparso) para diminuir a amplitute da busca binária*
- *Para a estrutura ISAM de um nível, o CustoIO=1+$log_2 B/F$, onde F é o número de ponteiros de cada nó do índice*



ISAM – vários níveis

- *Criando outros níveis no "index file" até uma raiz, não haveria busca binária, pois o acesso seria via ponteiros.*



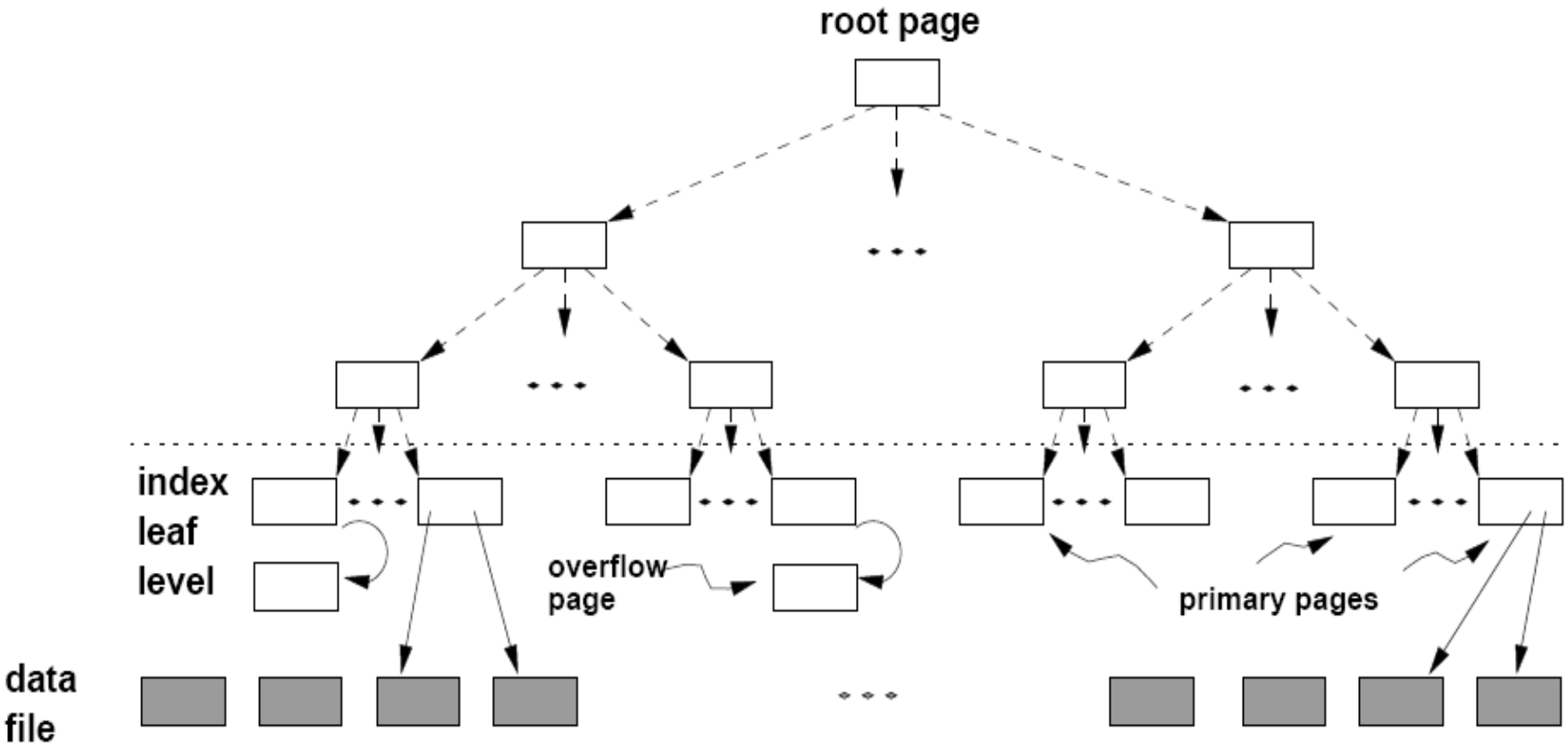
ISAM - Custo

- *Custo*
  - *A cada descida de nível na árvore o problema é dividido por F até chegar em 1*
    - ✓ *RAIZ:* B/F^0
    - ✓ *NÍVEL 1:* B/F^1
    - ✓ ...
    - ✓ *NÍVEL h:* $B/F^h = 1 \Rightarrow h = \log_F B$
- *Comparação com arquivo ordenado*
  - $\log_2 B / \log_F B = \log_2 F$
  - *Por exemplo, se F=64, o arquivo ordenado fará seis vezes mais IOs que o ISAM, mas F em geral é bem maior que 64*

ISAM - Construção

- *Idéia pressupõe pouca inserção/remoção*
- *Bottom-up*
  - *Alocação de páginas para os dados*
  - *Ordenação do arquivo de dados (sorting)*
  - *Alocação de páginas para índice*
  - *Criação das folhas seguido de níveis superiores*
  - *Alocação de páginas para overflow*

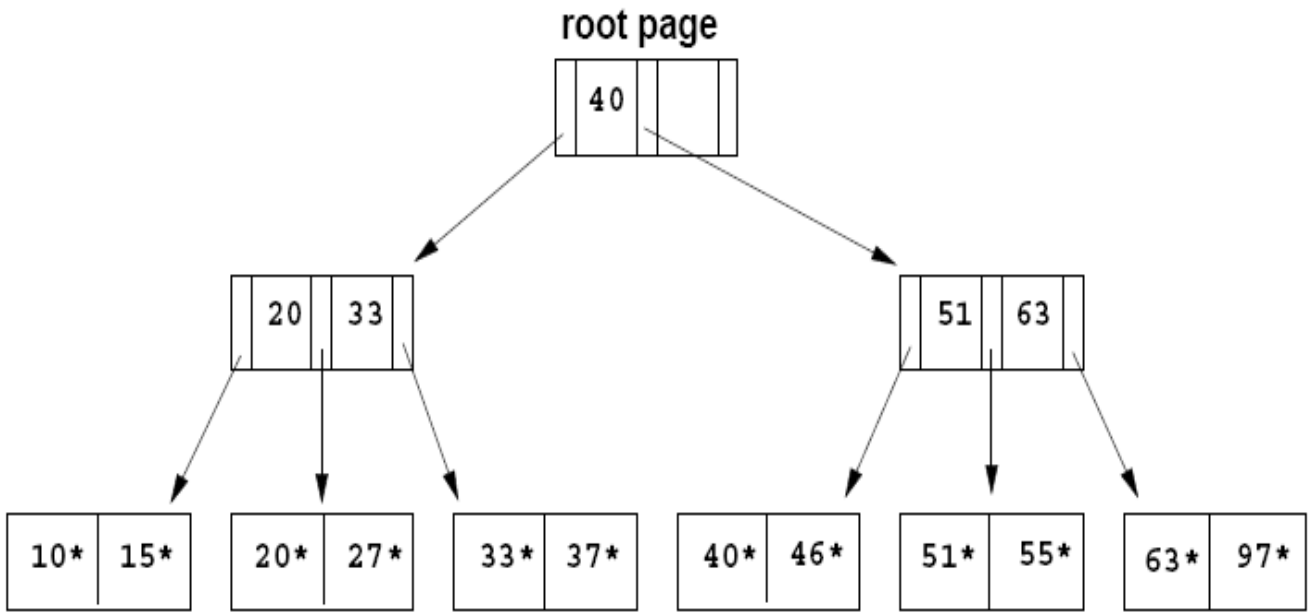
ISAM - ESTRUTURA TÍPICA



ISAM – Evolução após construção

- *Inserções e remoções não afetam os níveis intermediários e a raiz, mas somente as folhas*
- *Havendo inserções, as folhas serão encadeadas em páginas de overflow*
- *Isso pode degradar o desempenho*
- *A seguir um exemplo de ISAM e exemplos de operações na estrutura.*

Exemplo de ISAM

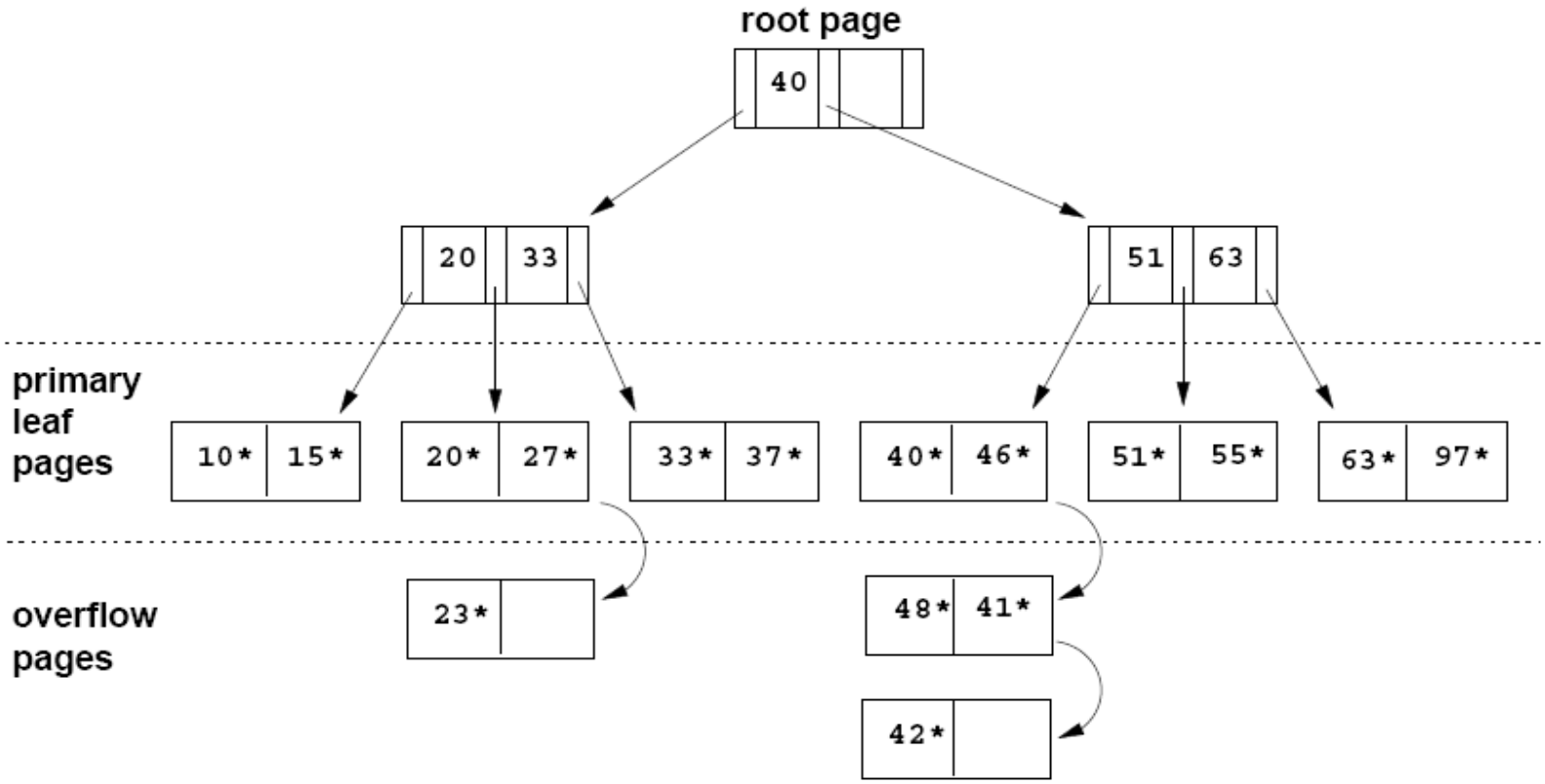


Exemplo de ISAM – cont. (inserção)

Inserir registros com chaves: 23. 48, 41 e 42

Exemplo de ISAM – cont. (após inserção)

Apos inserir registros com chaves: 23, 48, 41 e 42

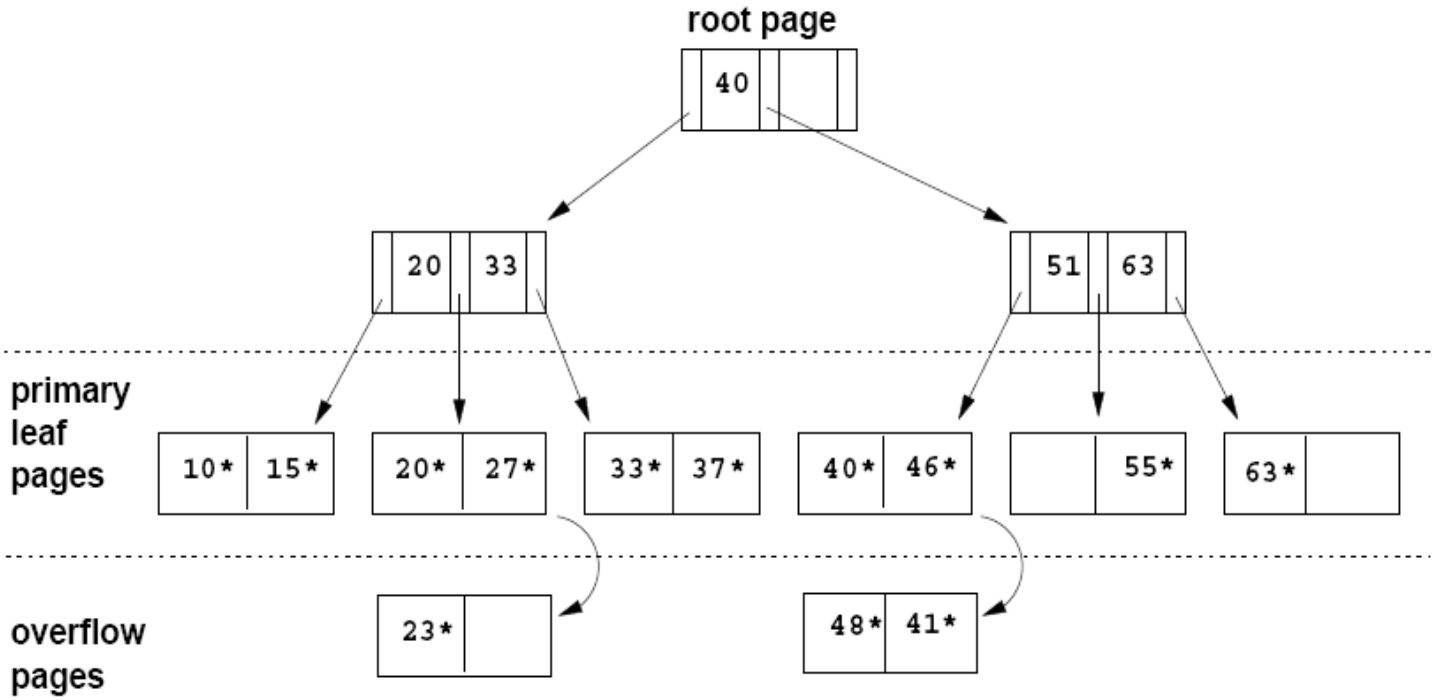


Exemplo de ISAM – cont. (remoção)

Remover registros com chaves: 42, 51 e 97

Exemplo de ISAM – cont. (remoção)

Após remover registros com chaves: 42, 51 e 97



ISAM - Considerações finais

- *Nodos internos e root não são alterados*
- *Podem aparecer chaves nos nodos internos que não aparecem nas folhas*
- *Pode-se gerar cadeias de overflow que prejudicam o desempenho, principalmente se as inserções forem desbalanceadas*
- *Pode-se deixar áreas livres nas folhas (por exemplo, 20%)*
- *O fato de não haver mudança nos nodos internos facilita o processo de controle de concorrência*

Índices baseados em árvores – Árvore B+

Estrutura dinâmica (Árvores B+)

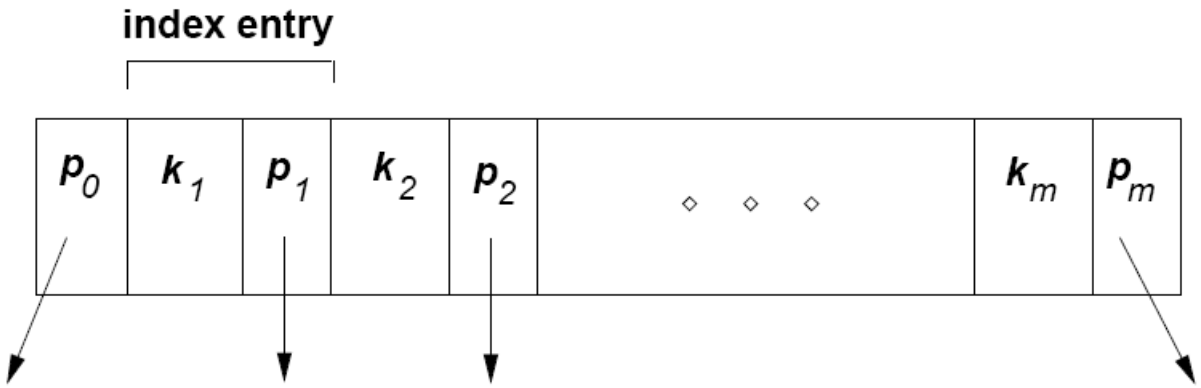
Árvore B+
Objetivos

Estrutura dinâmica derivada do ISAM com objetivo de:

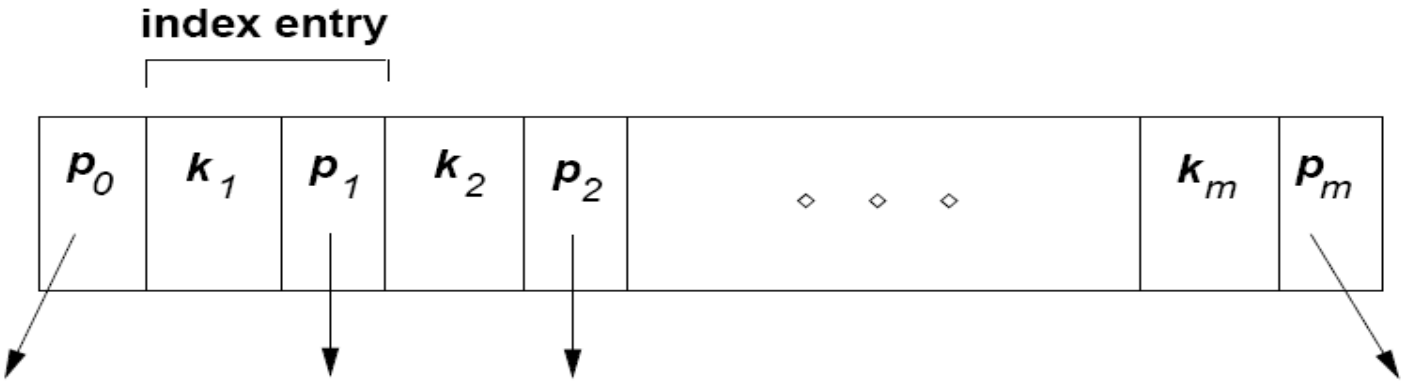
- *Manter desempenho dependente somente da altura da árvore*
- *Eliminar cadeias de overflow*
- *Manter árvore balanceada sempre*
- *Manter eficiência em insert/delete*
- *Exceto o root, manter ocupação mínima do nó em 50%*

Árvore B+
Nodos

- *Nós, exceto folhas, tem a mesma estrutura do ISAM(abaixo)*
- *Seja d a ordem da Árvore B+, então todo nó, exceto o root, terá m chaves, onde $d \leq m \leq 2d$*
- *Já o root terá $1 \leq m \leq 2d$*
- *O número de ponteiros no nó será m+1*



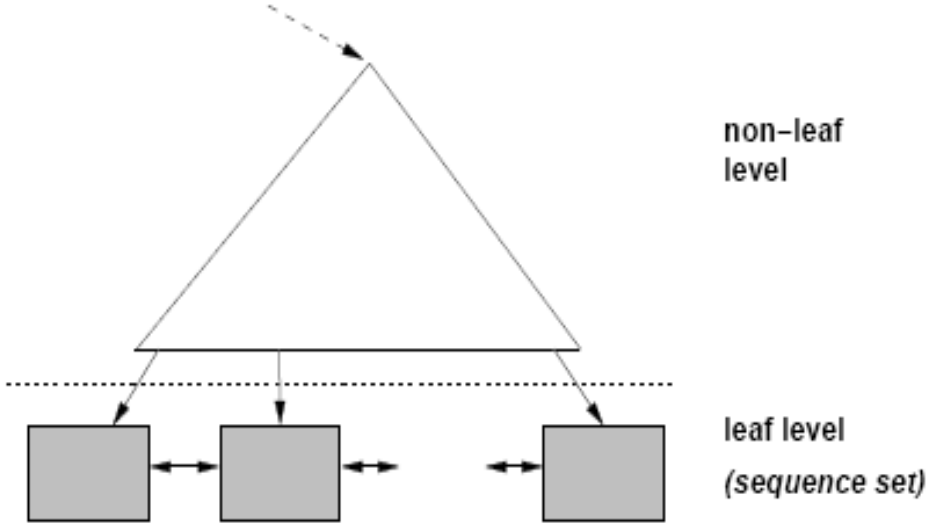
Árvore B+ - Index Entry (ie) no nodo interno



- p_0 *aponta para subárvore com chaves* $k < k_1$
- p_m *aponta para subárvore com chaves* $k \geq k_m$
- $p_{i|0<i<m}$ *aponta para subárvores com chaves* $k_i \leq k < k_{i+1}$

Árvore B+ - Folhas

- *Entradas nas folhas seguem alternativas 1, 2, ou 3*
- *Folhas com ponteiros para folhas adjacentes(next, previous), pois a alocação é dinâmica*



Árvore B+ - Exemplo de tamanho

Exemplo de dados práticos de uma Árvore B+

- *Ordem d=100 (para alguns autores a ordem é 2d)*
- *Fator de ocupação médio = 67%*
- *Fan-out médio = 133*
- *Capacidade com h=4: 133^4 = 312.900.700 registros*
- *Necessidades de buffer-pool*
  - ✓*Nível 0: root = 1pg = 8k*
  - ✓*Nível 1: 133 pgs = 1MB*
  - ✓*Nível 2: 17.689 pgs = 133MB*

Árvore B+ - Operações

Operções em Árvores B+:

- *Busca*
- *Inserção*
- *Remoção*

inicialmente considerando chaves únicas

Busca registro com chave k na Arvore B+

1. Inicie busca no root
2. Se página corrente é folha busque k* na folha
3. SeNão

   Se $k < k_1$ Faça i=0

   SeNao

   Se $k \geq k_m$ Faça i=m

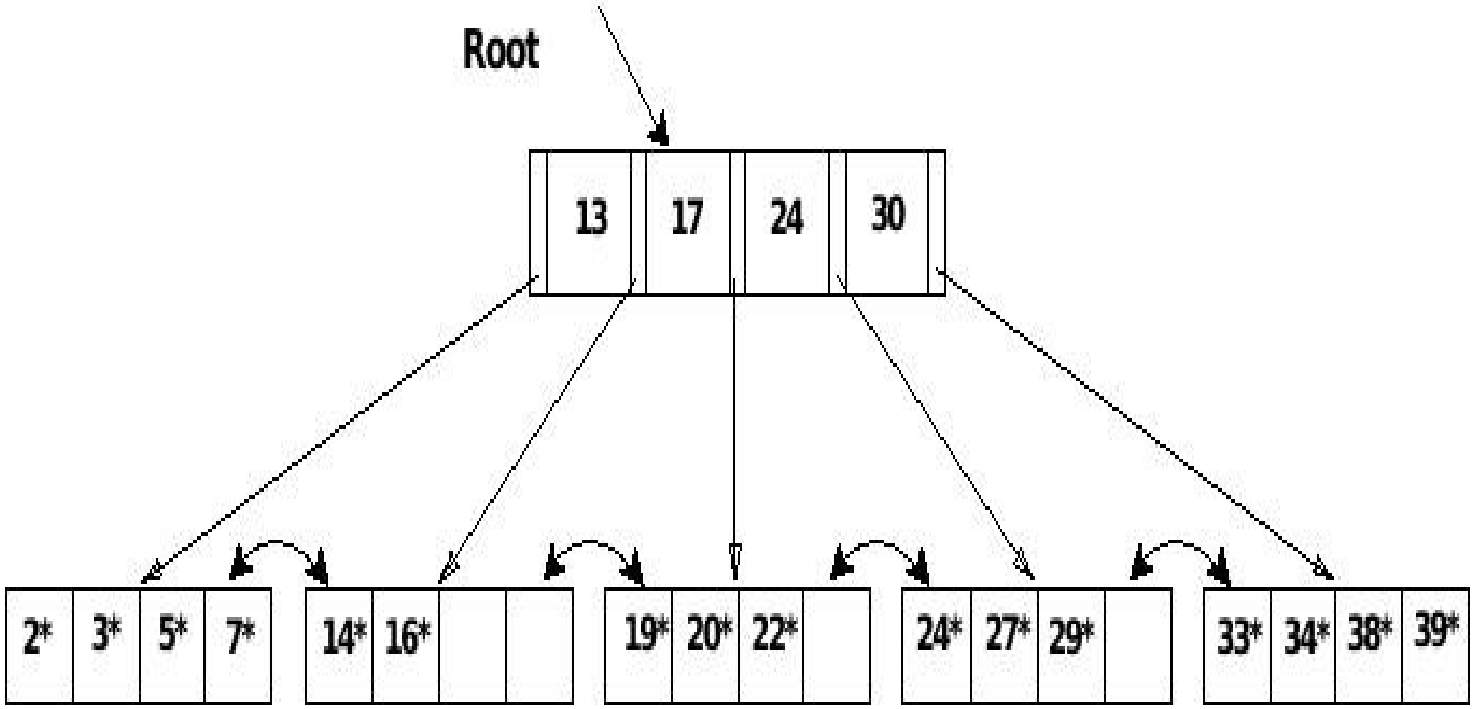
   SeNao determine i tal que $k_i \leq k < k_{i+1}$;

   Desça para subárvore p_i

   Vá para 2.

Árvores B+ - Exemplo

Exemplo Árvore B+ com d=2 e F=5, busca 5, 15, ≥24



Árvores B+
Função recursiva para Busca

Busca nodo com registro de chave k

*(nptr é ponteiro para nodo e *nptr é o nodo)*

func *treeSearch (nptr, K)* ***returns*** *nodepointer*
if **nptr é folha,* ***returns*** *nptr;*
else
  if $K < K_1$, ***returns*** $treeSearch(P_0,K)$
  else
    if $K \geq K_m$, ***returns*** $treeSearch(P_m,K)$
    else
      find $i | K_i \leq K < K_{i+1}$;
      returns $treeSearch(P_i,K)$
endfunc

Árvores B+- Algoritmo para Inserção

Insere registro com chave k na Arvore B+

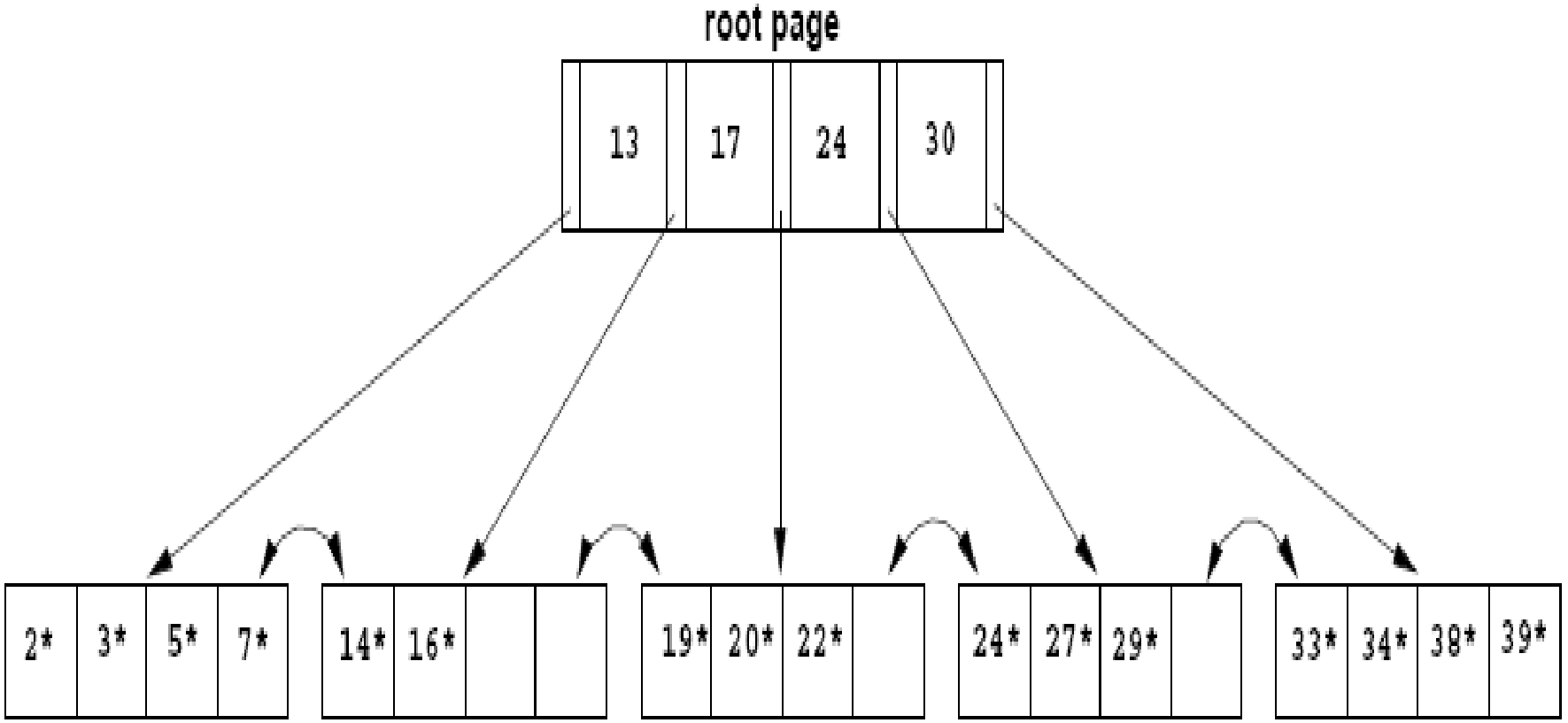
1. busca folha L
2. insere registro em L
   se não há espaço em L (split da folha L)
      cria novo nó L'
         distribui DE: d em L e d+1 em L' (ou vice-versa)
         3. insere IE para L' no pai de L, seja I o pai de L
            se não há espaço suficiente (split de nó interno I)
               cria novo nó I'
               distribui IE: d em I e d em I'
               sobe IE do meio para inserção (3.);

Árvores B+- Inserção

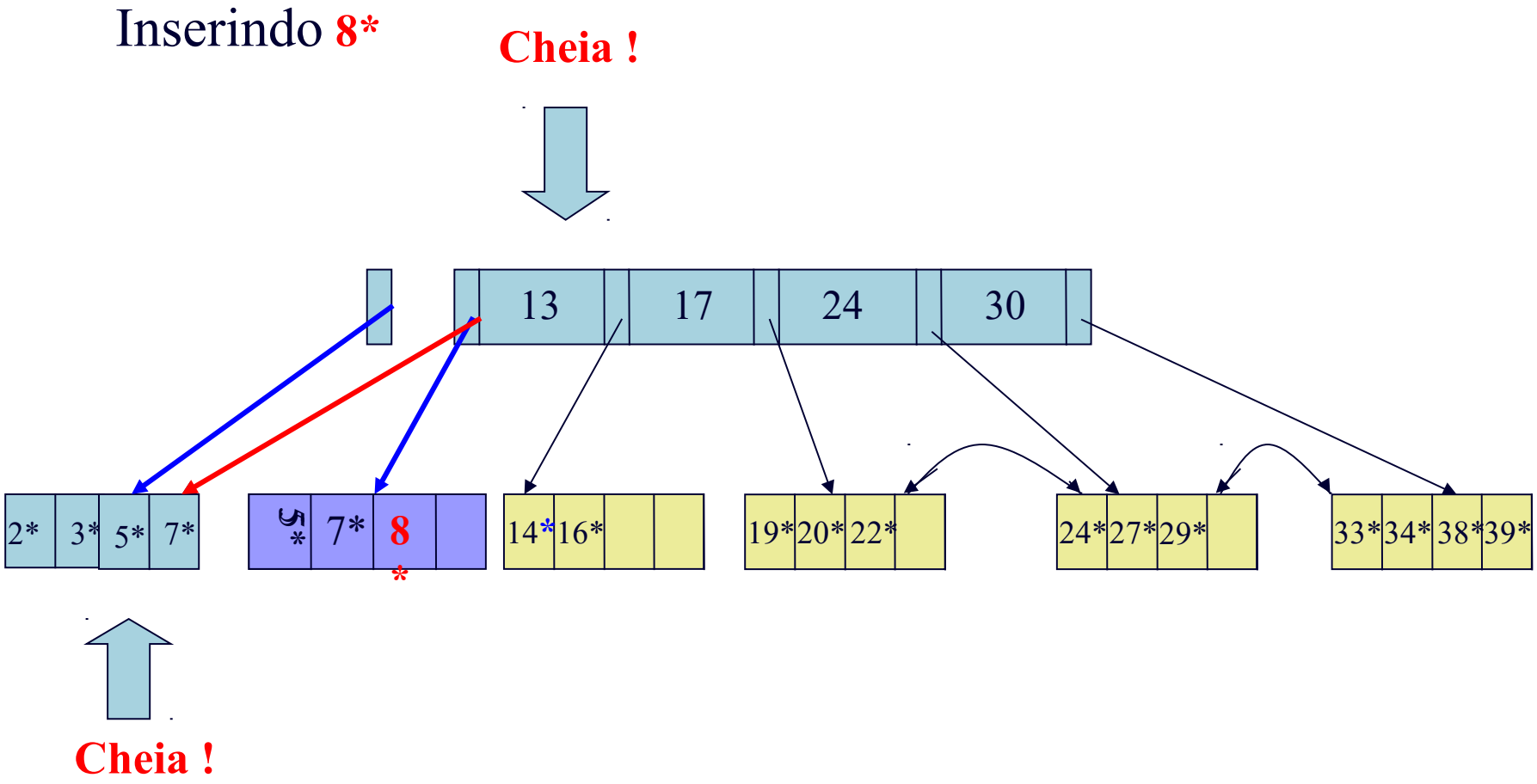
- *inserção com split aumenta tamanho da árvore*
- *inserção recursiva com split pode chegar até a raiz, aumentando a altura da árvore*
- *observe que:*
  - *no split da folha ocorre uma cópia da menor chave em L' para o pai de L*
  - *já no split de nodo interno a chave do meio é movida para o pai de I*
- *Uma variante é tentar redistribuição em folhas antes do split*

Árvores B+- Exemplo Inserção

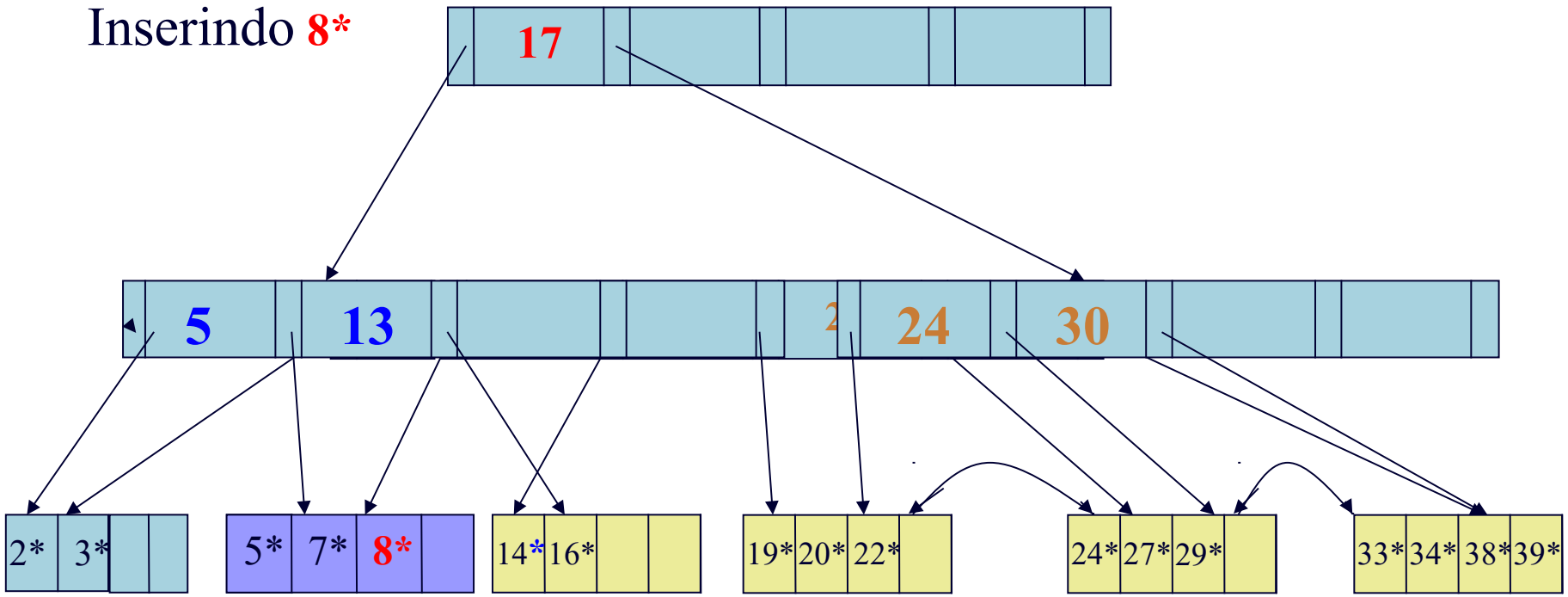
Exemplos: inserção de 8 na árvore abaixo*



Árvores B+- Exemplo Inserção com Split



Árvores B+- Exemplo de Inserção



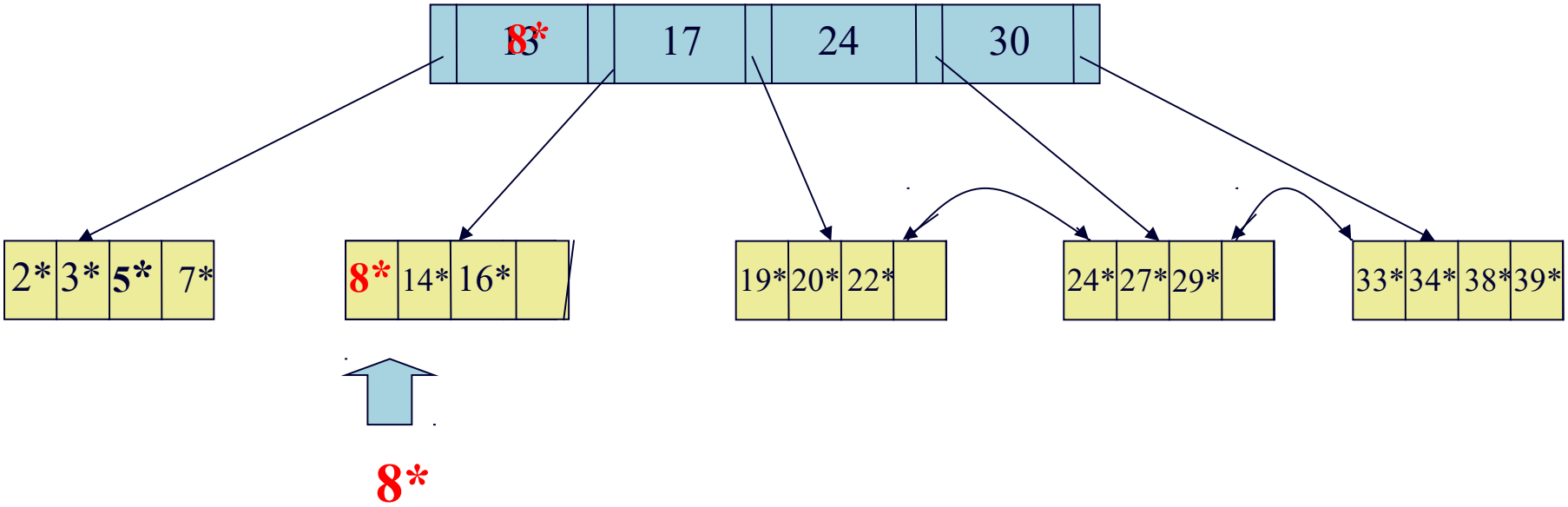
Árvores B+- Inserção com redistribuição

VARIANTE NA INSERÇÃO

antes do split, tentar redistribuição em folhas

Árvores B+- Inserção com redistribuição

Inserindo **8***



Árvores B+- Inserção

OUTROS EXEMPLOS:

- *inserção sem split: 23*
- *inserção com split sem propagação: 40*

Árvores B+: Função de Inserção

function *insert (nptr, de) returns(ieptr);% ieptr é nulo até que um split*
if **nptr não é folha, seja N = *nptr % N é o conteúdo do nó*
  find *i |((i=0* ***if*** *de.K < K_1)* ***or*** *(i=m* ***if*** *de.K≥K_m)* ***or*** *($K_i \leq$ de.K < K_{i+1}));*
  ieptr=insert(P_i, de); % insere entrada na subárvore P_i
  if *ieptr é nulo,* ***return****(null); % não houve split, nada mais a fazer*
  elsif *N tem espaço,* ***put*** **ieptr em N,* ***return****(null); % nada mais a fazer*
  else *% split de nodo interno N ao incluir *ieptr em N*
    altere N, mantendo primeiras d chaves e d + 1 ponteiros
    cria novo nodo S, com últimas d chaves e d + 1 ponteiros
    ieptr=&(<chavedomeio, &S) >; % nova entrada que subirá
    if *N era o root*
      ieptr=&(< &N, ieptr>); % altera raiz da ávore
  return*(ieptr); % se mudou o root retornará a nova raiz da árvore,*
                % senão subirá a nova entrada para inserção
else*...continua com processamento de folha.*

Árvores B+:Função de Inserção cont.

```
else.....continuação insert, processamento de folha
    seja L=*nptr,
    if L tem espaço, put de em L, return(null);
    else  % a folha está cheia split L
        altere L, mantendo primeiras d entradas
        crie novo nodo S, com restante de entradas (d+1 entradas)
        altere ponteiros adjacentes em L, S e dos adjacentes a L
        return(&(<S.K1, &S>)); % entrada para S para inserção em
                               %ancestrais
endfunc;
```

Árvores B+: Remoção

Remoção de registro com chave k na Arvore B+

- Se ocupação não fica abaixo do mínimo, remove, não altera ponteiros, nem nós ancestrais
- Se ocupação fica abaixo do mínimo
  - Tenta redistribuição com nó vizinho, se possível
  - Caso contrário junta com vizinho

Árvores B+: Algoritmo de Remoção

Remoção de registro com chave k na Arvore B+

1. busca folha L

2. remove entrada em L

3. Se L ficar com d − 1 entradas

tente redistribuir entradas de folhas adjacentes a L

ou

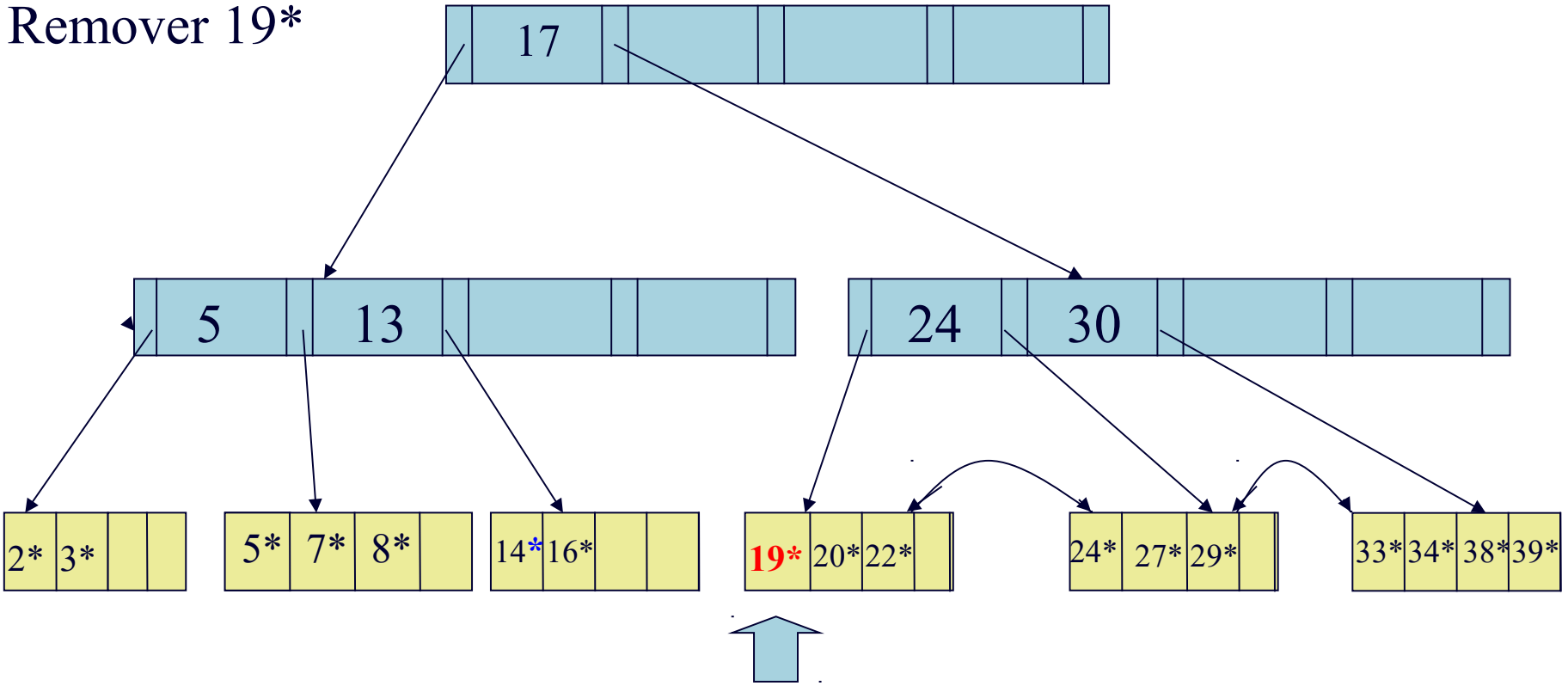
faça merge de L com S, adjacentes a L, e remova um

4. Se houve redistribuição atualize chave no pai

5. Se houve merge remova ponteiro do pai, o que pode provocar propagação de merge/redistribuição

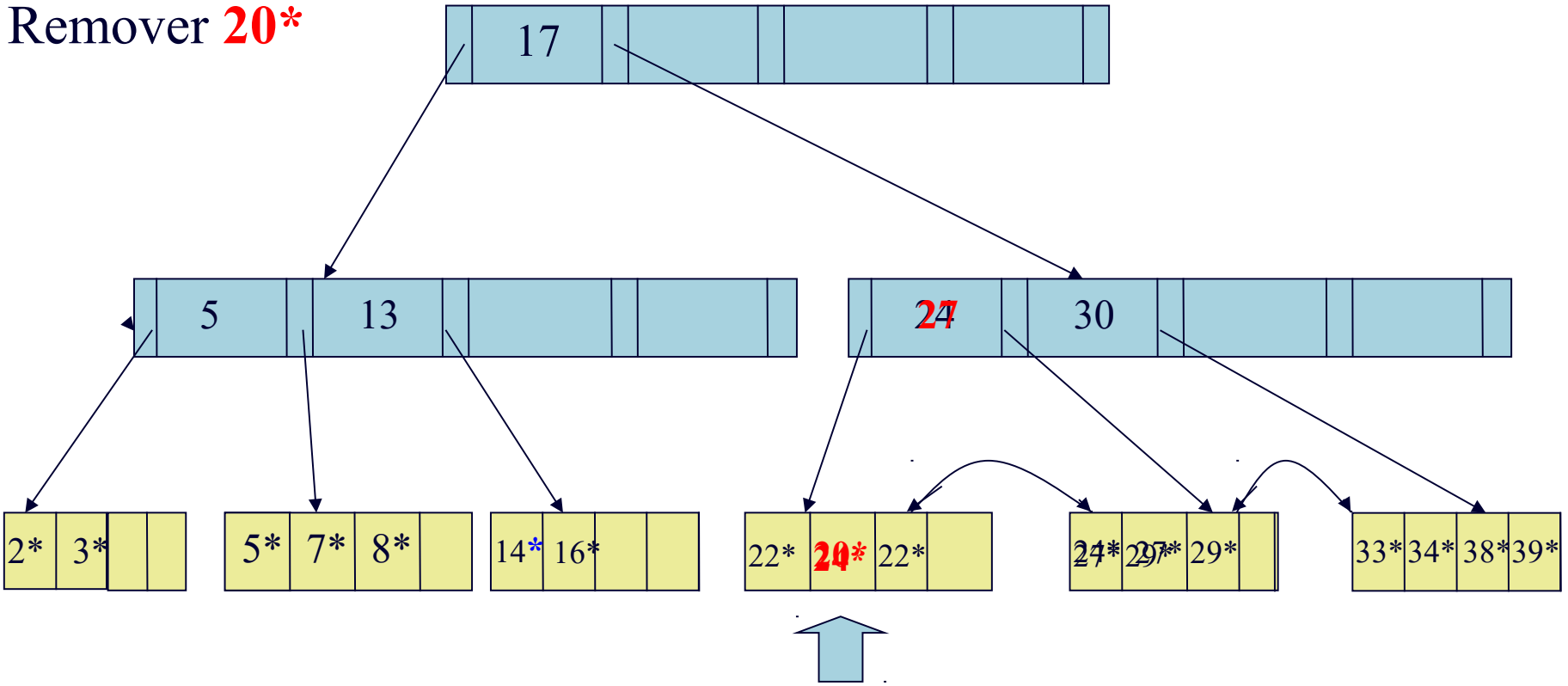
Remoção em Árvores B+

Simples, sem merge/redistribuição



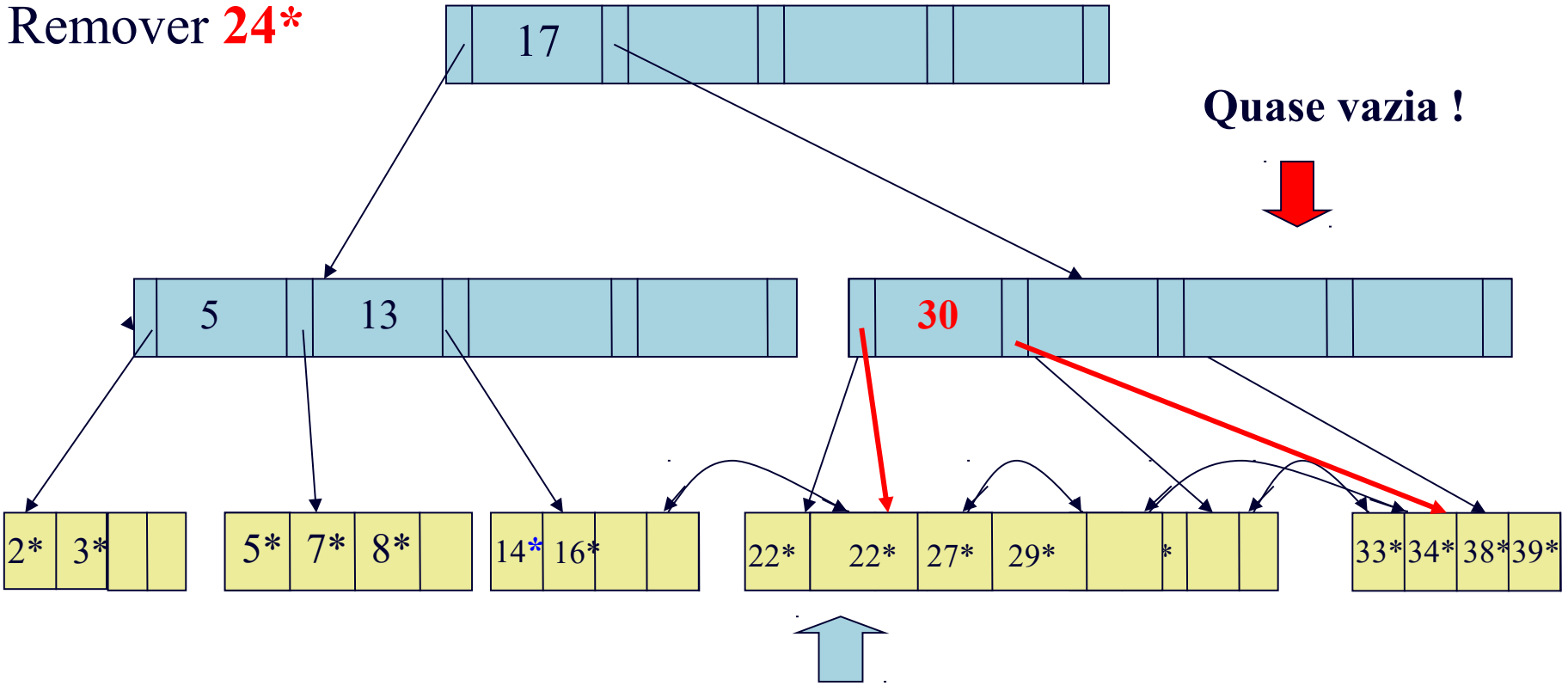
Remoção em Árvores B+

Com redistribuição nas folhas



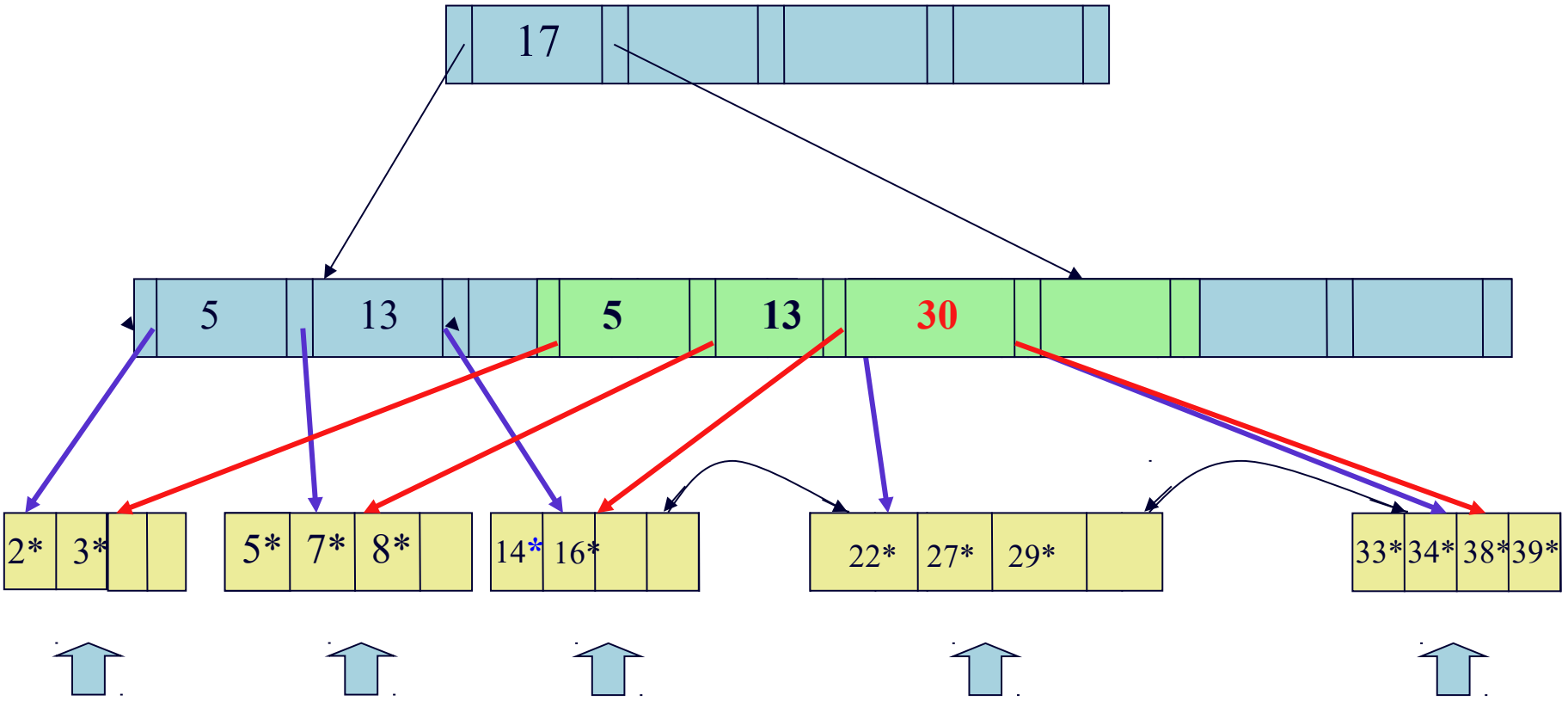
Remoção em Árvores B+

Merge de folhas



Remoção em Árvores B+

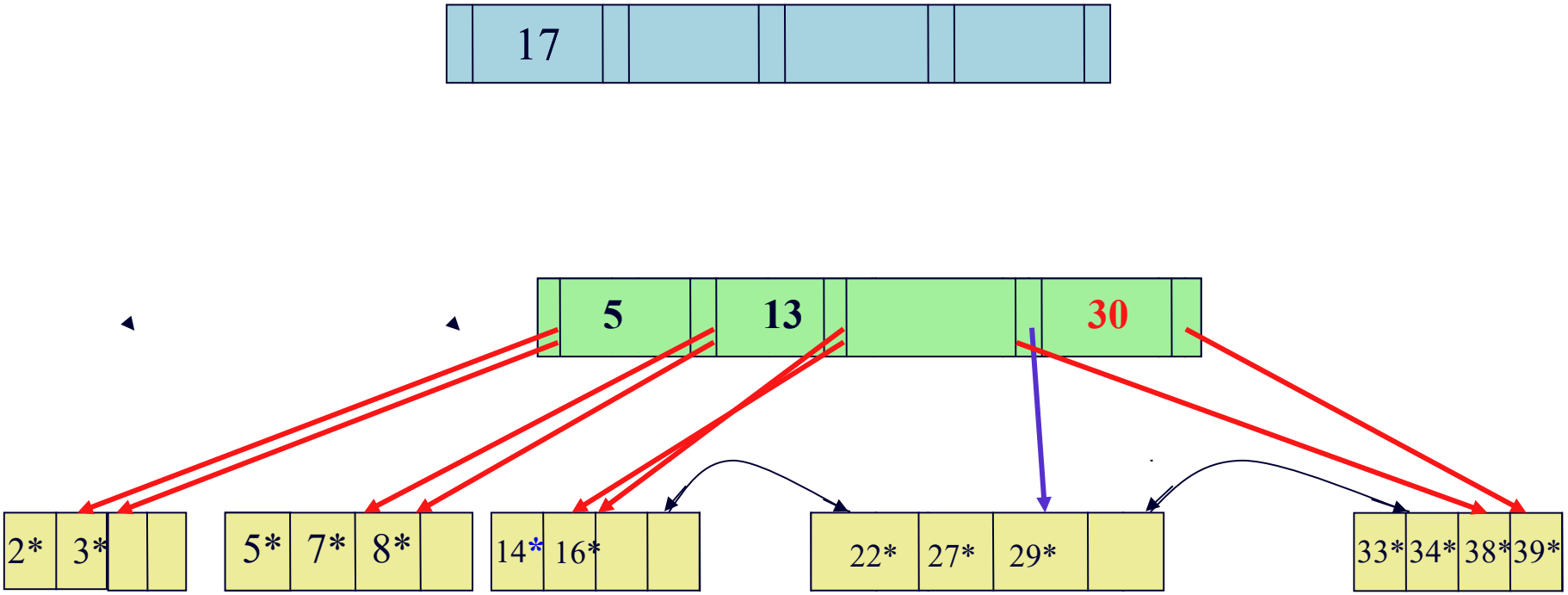
Merge de dois nós intermediários



São necessários 5 ponteiros

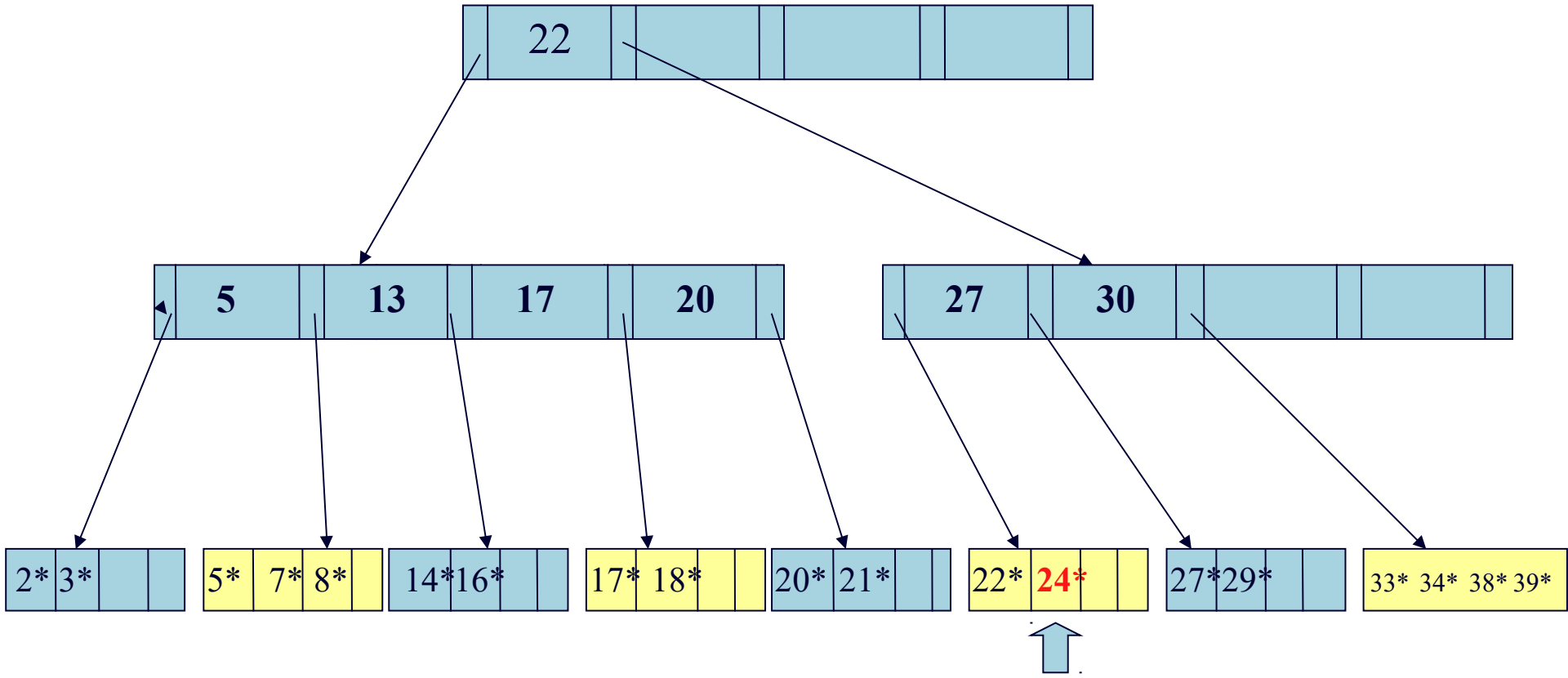
Remoção em Árvores B+

Merge de dois nós intermediarios



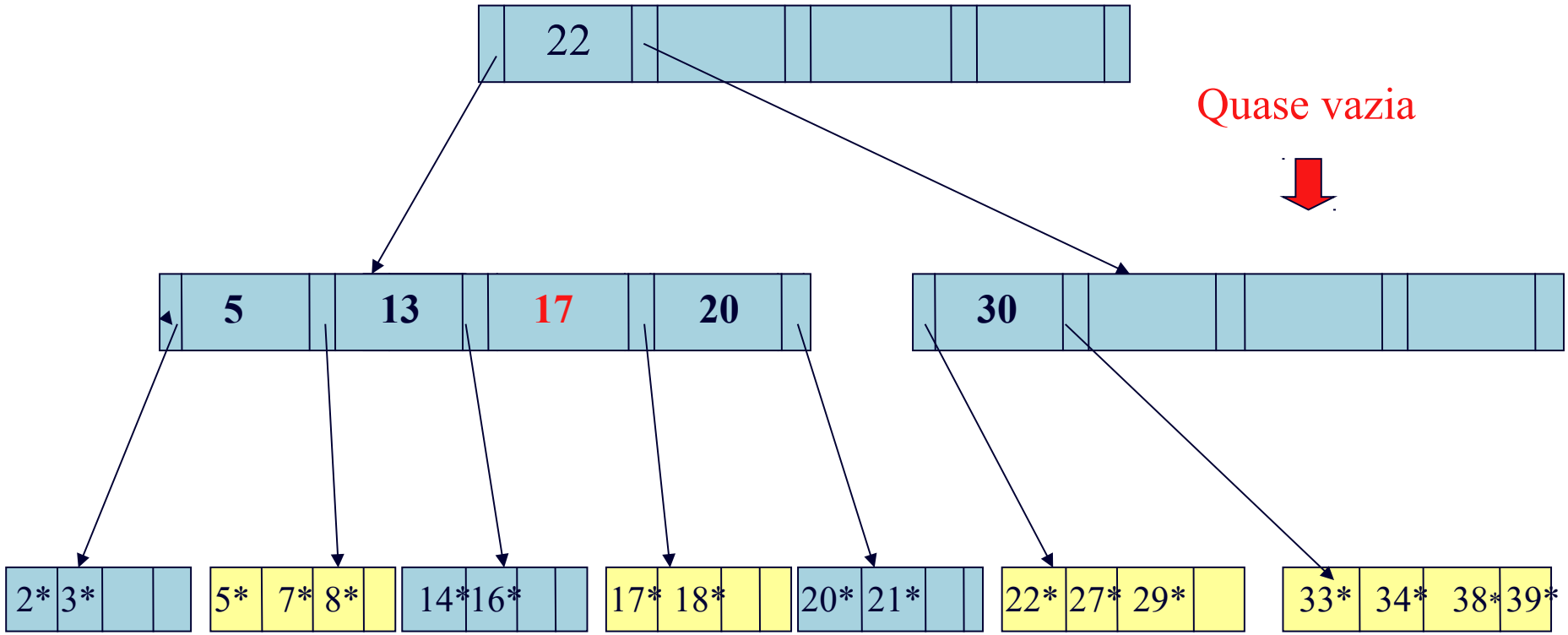
Remoção em Árvores B+

Redistribuição em nós intermediários



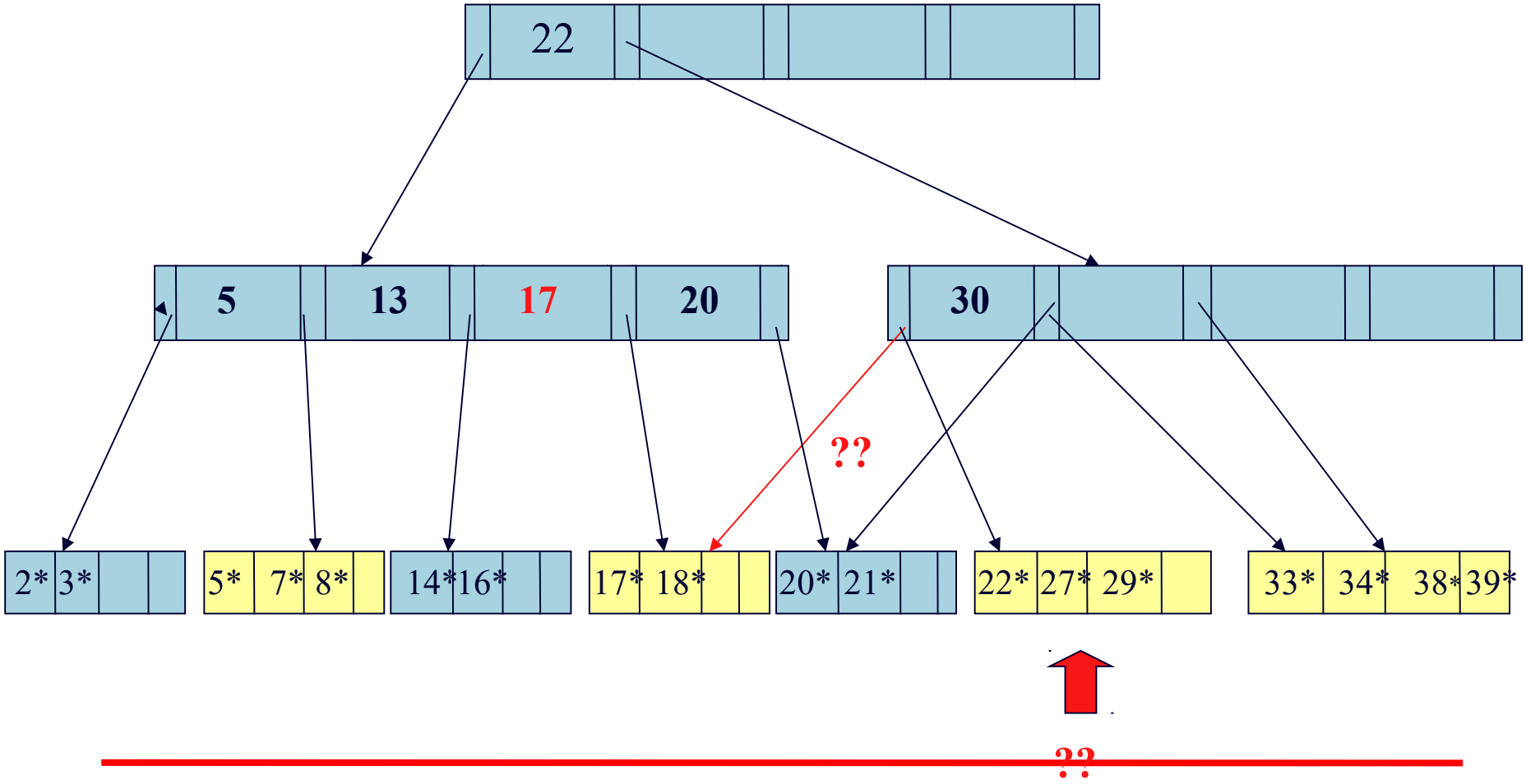
Remoção em Árvores B+

Redistribuição em nos intermediarios



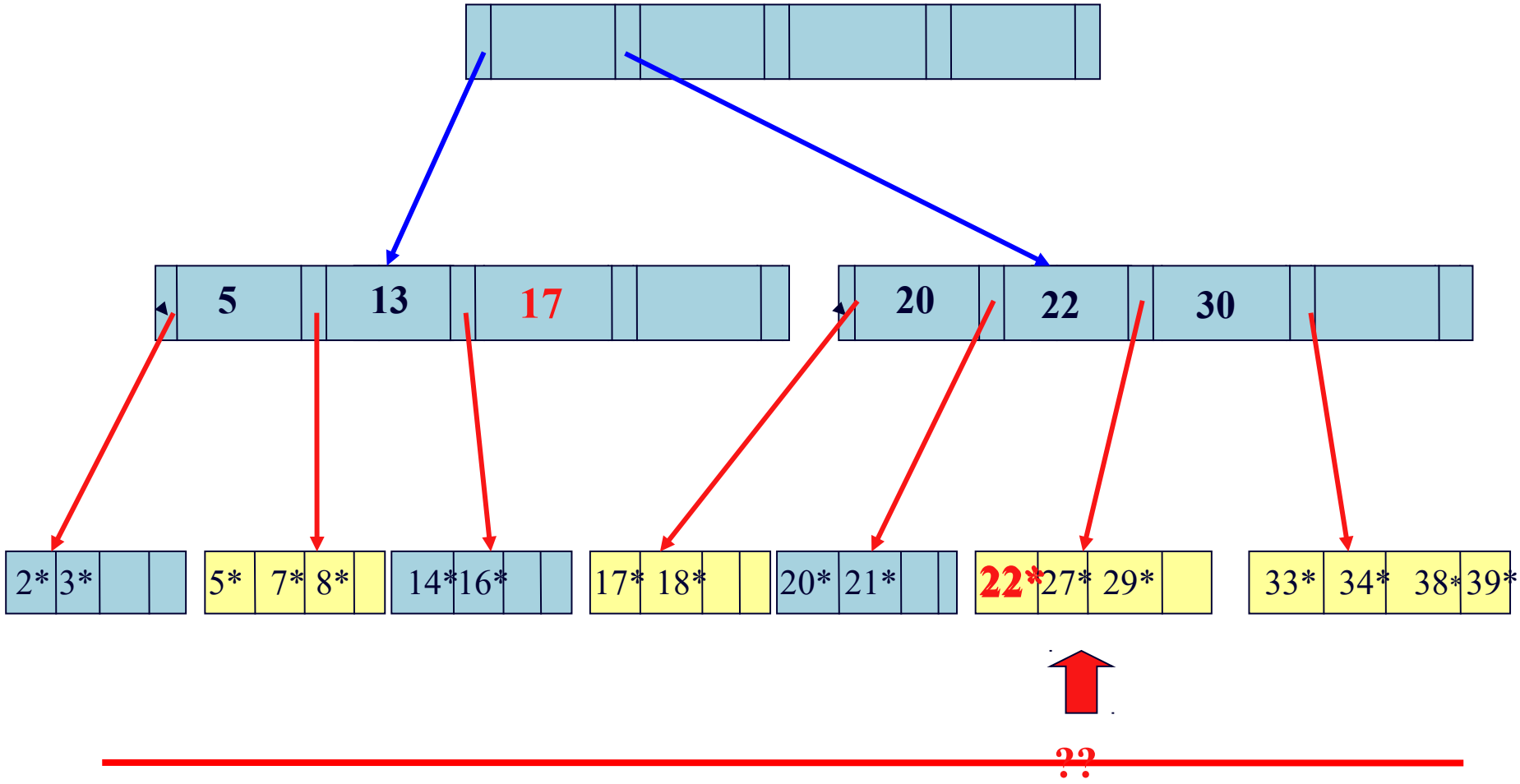
Remoção em Árvores B+

Redistribuição em nós intermediários



Remoção em Árvores B+

Redistribuição em nós intermediários



Remoção em Árvores B+: Função

```
func delete (pptr, nptr, k) returns(ieptr)% ieptr é nulo até que haja um merge
if nptr  é folha, seja L = *nptr, remova entrada relativa a k em L
   if L tem entradas suficientes, return(null); % nada mais a fazer
   else, % a folha estava com d entradas, fazer redistribuição ou merge
      seja S um vizinho de L, filho do mesmo pai; % usa pptr
      if S tem entradas suficientes,
         redistribua entradas entre L e S
         seja M o nó mais a direita entre {L, S}
         seja < k_r, p_r > a entrada de M no pai;
         atualize k_r para o menor valor em M
         return(null);
      else, % merge L e S ,
         seja M o nó à direita entre {L, S}  (será removido)
         move todas entradas de M para nó à esquerda
         ajuste ponteiros adjacentes
         return(&(entrada para M no pai));
else...continua com processamento de nodo interno
```

Remoção em Árvores B+: Função

else...*continua com processamento de nodo interno*

*seja N = *nptr,*

find *i |((i=0* ***if*** *k < k_1)* ***or*** *(i=m* ***if*** *k≥k_m)* ***or*** *($k_i \leq k < k_{i+1}$));*

oc = delete(nptr, p_i, k); *% delete recursivo, nptr passa a ser o pai*

if *oc é nulo,* ***return****(null);* *% não houve merge*

else *remove *oc de N;*

if *N tem entradas suficientes* ***return****(null);* *% ((≥ d) ou (≥ 1 se root))*

elseif *pptr=null o root da árvore para o N.p_0;* ***return****(N.p_0);*

else *seja S o vizinho de N;* *% usa ptr no pai*

if *S tem entrada extra* *% ((> d),* *redistribua entre N e S, fazendo rotação no pai;*

return*(null);*

else *% merge N e S*

seja M o nó à direita (a remover)

oc = &(entrada para M no pai)

copia chave de spliting do pai (oc.k) para nó à esquerda

move M.p_0 e entradas de M para nó à esquerda

return*(oc);*

Chaves duplicadas em Árvores B+

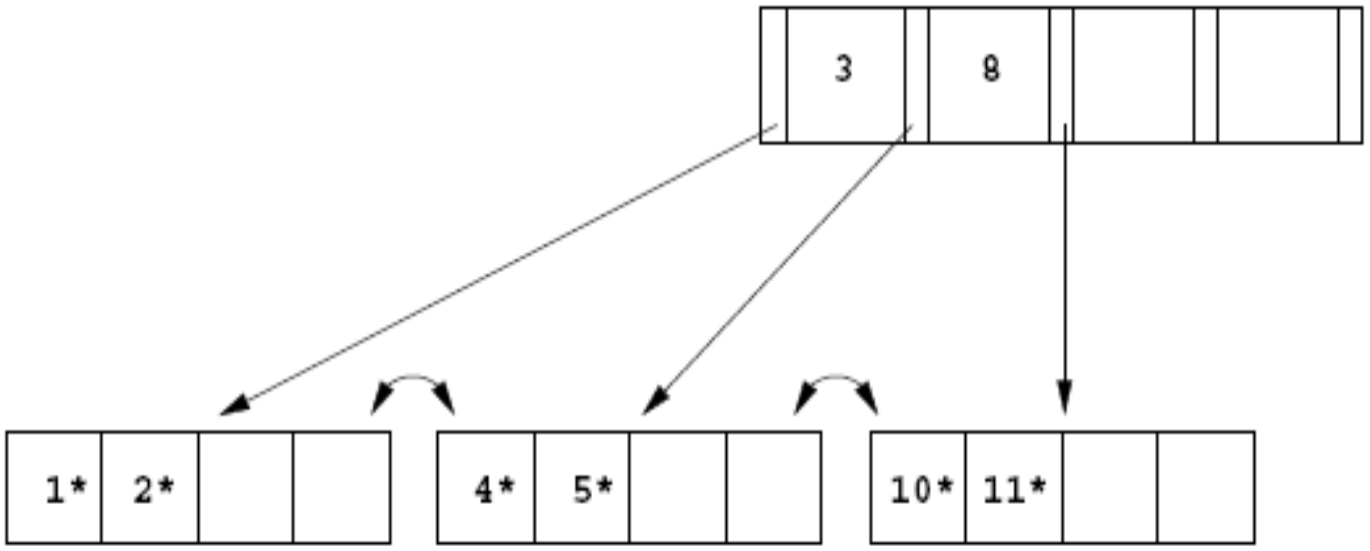
- *Técnica 1:*
  - ✓ *todas entradas de uma chave na mesma página,*
  - ✓ *se (repetições > 2d) use páginas de overflow*
  - ✓ *alterar algoritmo de busca, inserção e remoção somente para acesso à cadeia de overflow*
- *Técnica 2:*
  - ✓ *localização de página mais à esquerda*
  - ✓ *varredura do sequencial set*
  - ✓ *demora para identificar registro da remoção*
  - ✓ *alterar algoritmo de busca, inserção e remoção:*
    (PERCORRA SEQUENCIAL SET NAS DUAS DIREÇÕES)
- *Técnica 3*
  - ✓ *usar rid como parte da chave,*
  - ✓ *então não haverá duplicatas*
  - ✓ *não altera algoritmos de busca, inserção e remoção*

Chaves duplicadas em Árvores B+

Problemas com o algortimo de inserção e busca

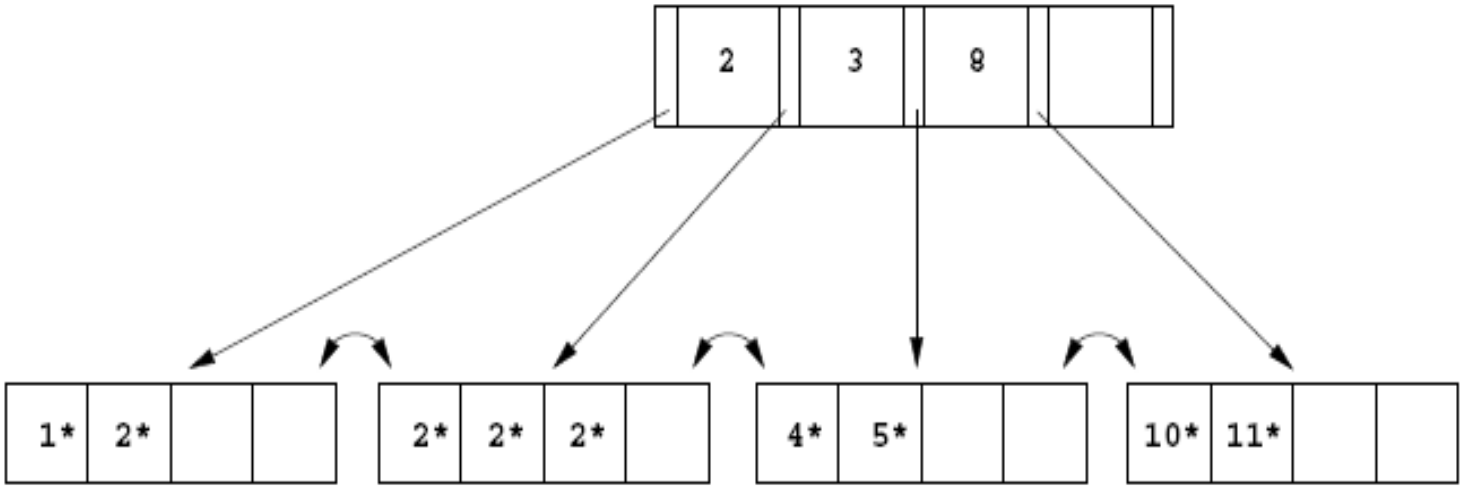
Seja a Árvore B+ abaixo com d=2

inserir 3 registros com chave=2



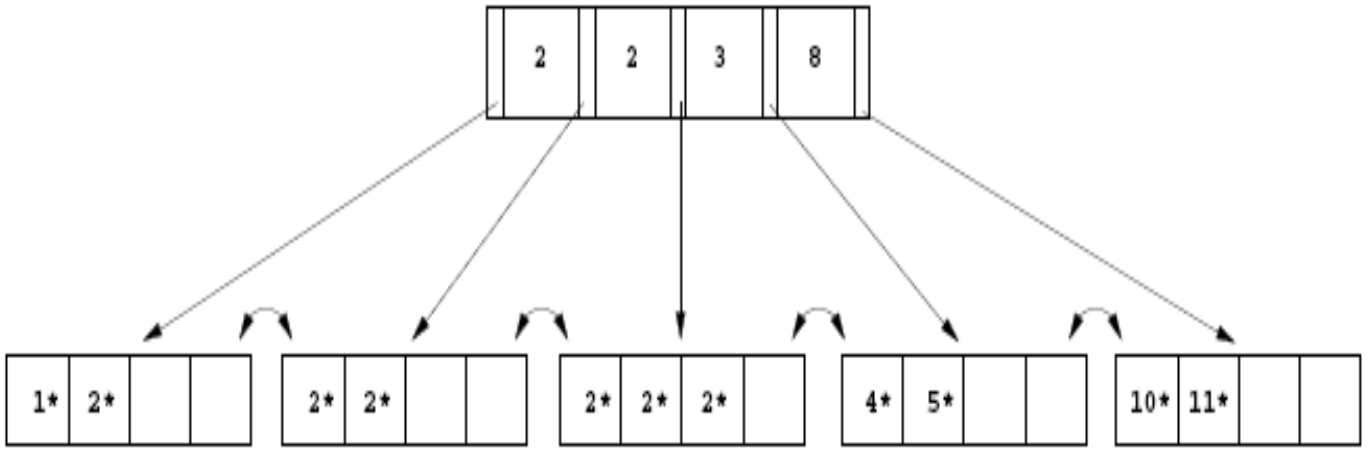
Chaves duplicadas em Árvores B+

Após inserção de três registros com chave=2



Chaves duplicadas em Árvores B+

- Após inserção de seis registros com chave=2



- Modificações na busca:
  - Não-folha: encontrar ponteiro p_i mais à esquerda tal que

$$K_i \leq K \leq K_{i+1}$$

  - Folha: se a menor entrada for k*, seguir ponteiros adjacentes para a esquerda e depois para a direita.

Compressão de Chave em Árvore B+

- *Compressão aumenta fanout, diminuindo altura, pois* $h = log_F B$
- *chave apenas direciona busca no índice, então:*
  - *{dannon yougurt, david smith, devarakonda murthy}*

pode ser

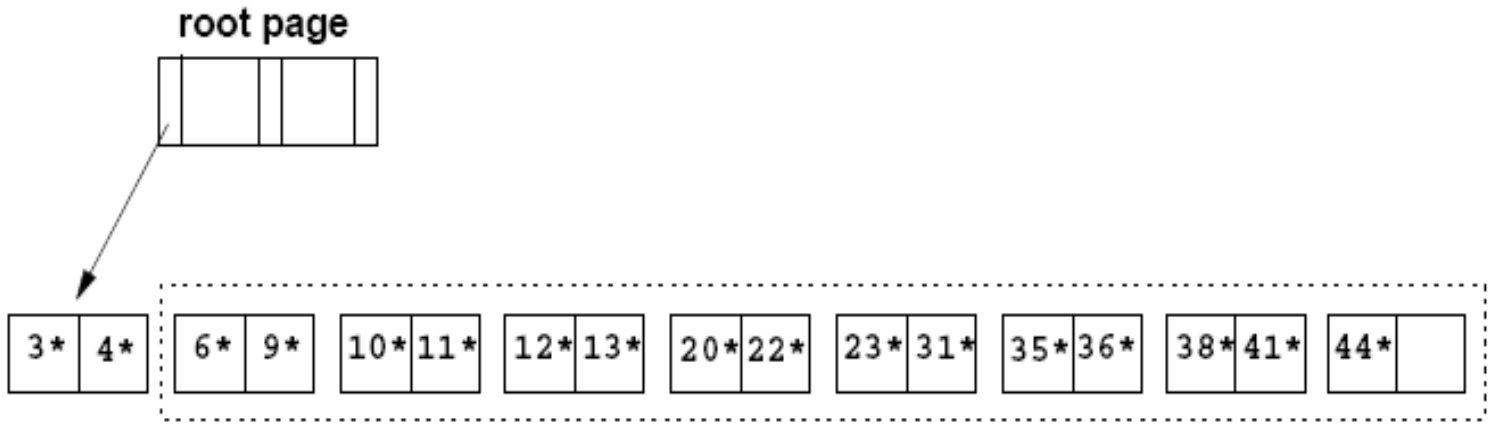
  - *{dan, dav, de}*

Carga de uma Árvore B+

- *carga de uma coleção de registros*
- *repetições de insert não são eficientes*
- *Usando bulkloading*
  - *ordenação de registros em disco*
  - *insere ptr da esquerda para direita*
  - *splits quando for necessário*
- *Um SGBD fará bulkloading se o create index for após a carga da tabela, por exemplo:*
  - *Insert 1.000.000 linhas em uma tabela*
  - *create index para a tabela*
- *O contrário, create index antes do insert, é ineficiente*

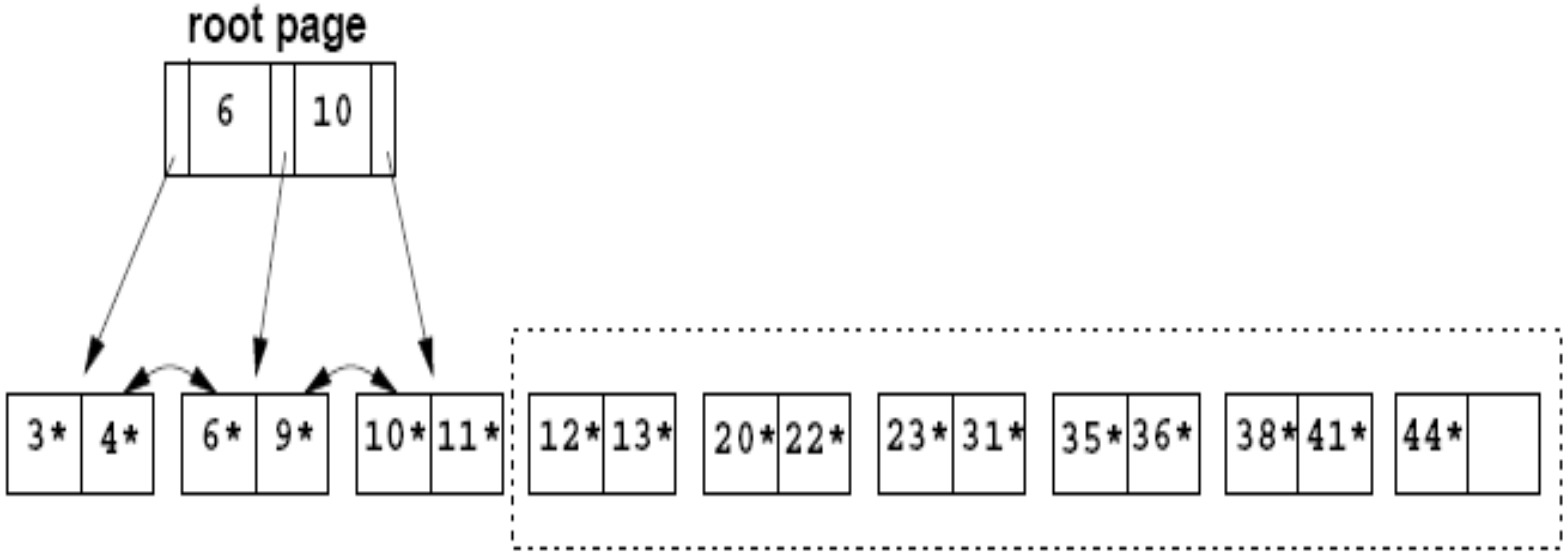
Bulkloading de Árvores B+

- *Crie uma lista ordenada de folhas com k**
- *Aloque uma página de root vazia com p_0 apontando para a primeira página da lista ordenada (exemplo para d=1)*



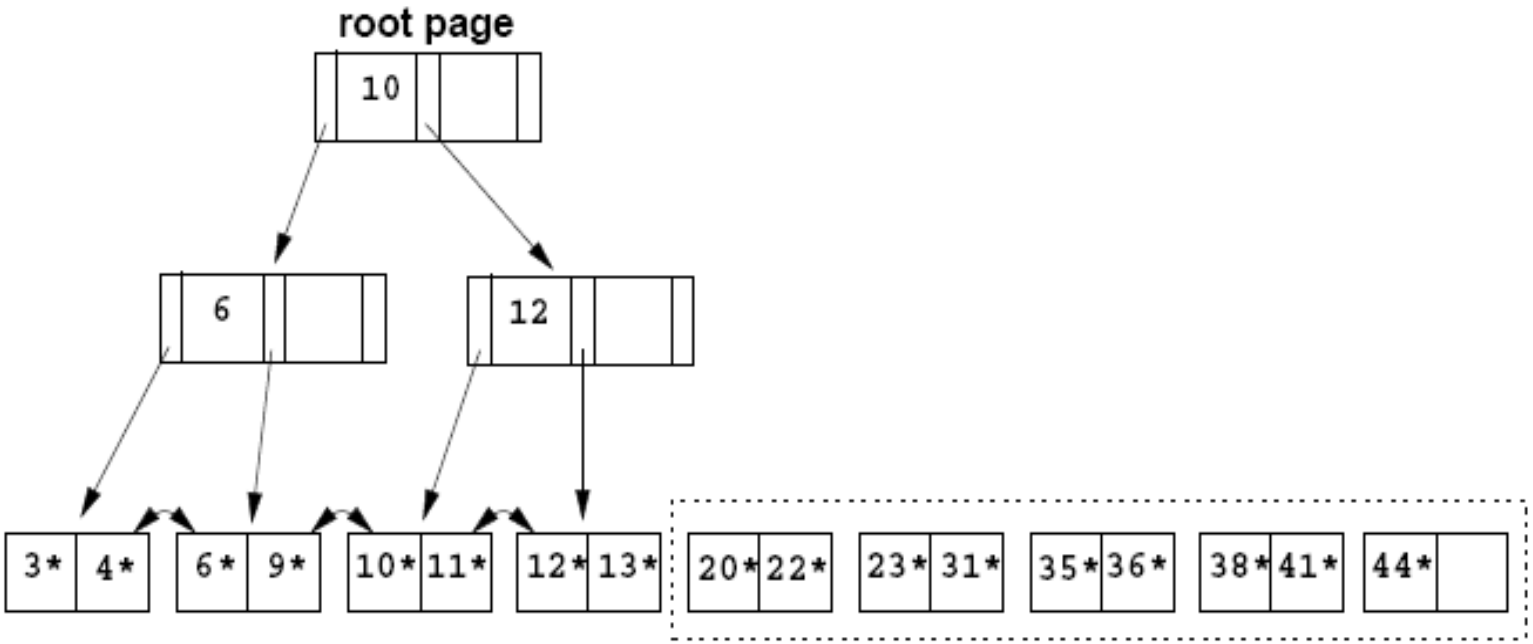
Bulkloading de Árvores B+

Para cada folha L, insira uma index entry contendo a menor chave de L e ponteiro para L, na figura <6,p_1> e <10,p_2>:

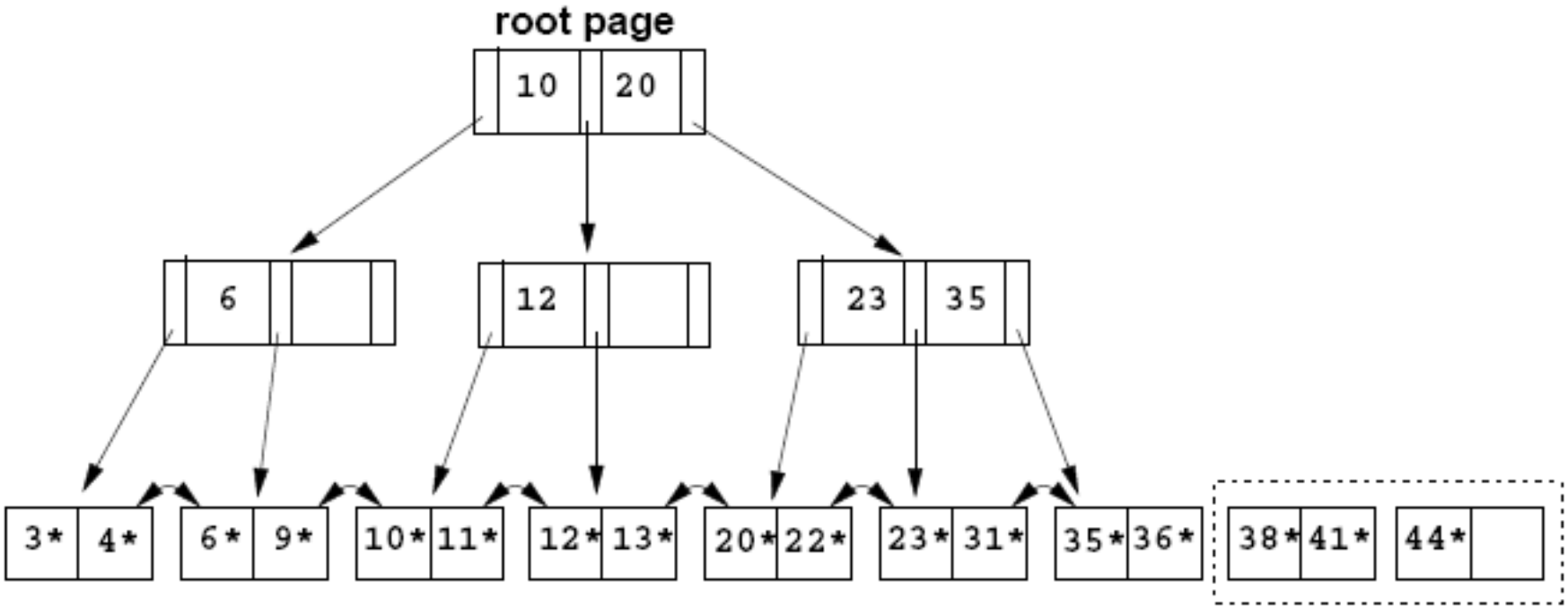


Bulkloading de Árvores B+

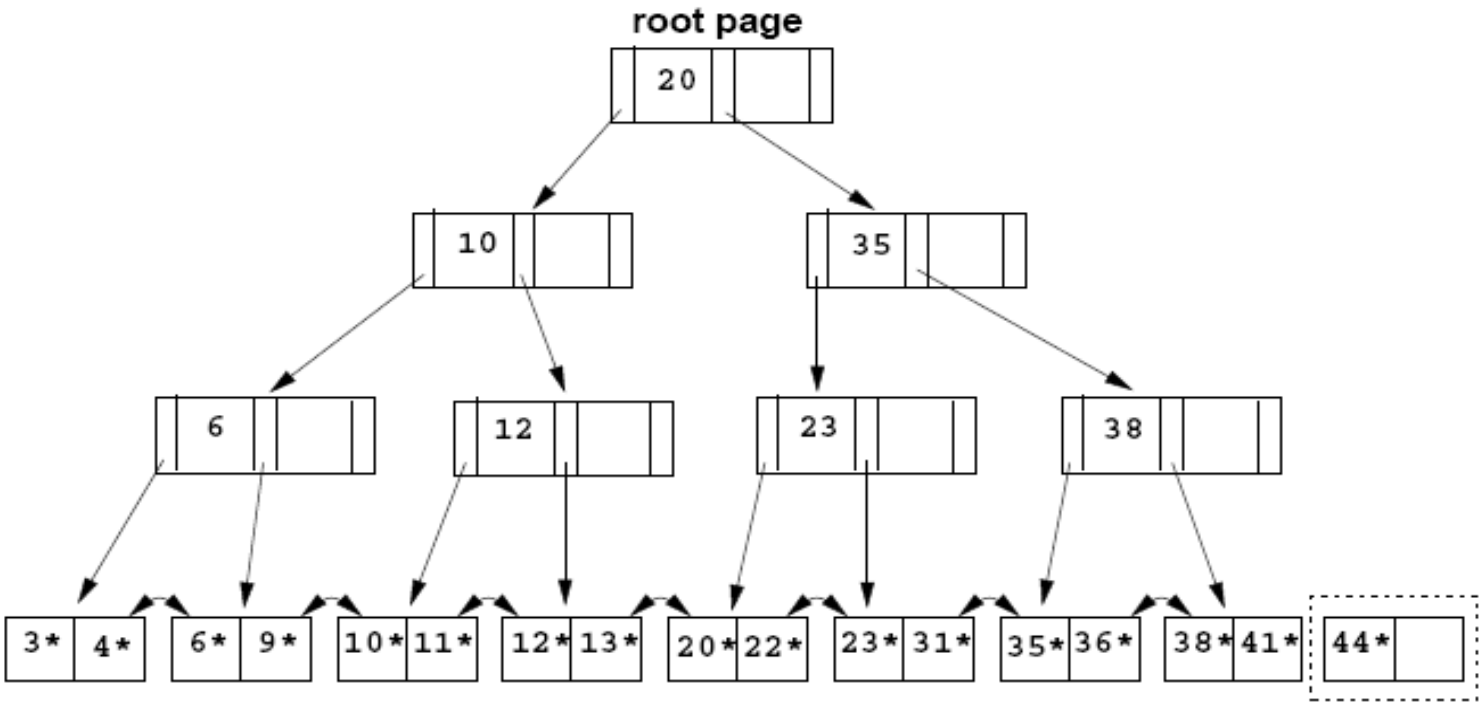
Splits ocorrerão ao completarem os nodos da esquerda para direita



Bulkloading de Árvores B+



Bulkloading de Árvores B+



Considerações finais sobre árvores

- *Ordem d de nodos não-folha pode ser diferente da ordem das folhas*
- *Nas folhas, registros de tamanho variável com alternativa 1 ou alternativa 3 para chaves duplicadas, tornam a ordem dinâmica*
- *Splits de alternativa 1 podem mudar rid de outros índices*
- *Compressão de chaves pode reduzir altura*
- *Carga do índice tem melhor desempenho que múltiplos inserts*
- *ISAM é uma boa estrutura estática, mas Árvore B+ é a melhor estrugura genérica e a mais utilizada em SGBDs e outros gerenciadores de arquivos.*

Exercícios - Índices baseados em árvores

EXERCÍCIOS

*FIM - Índices baseados em árvores**

** material baseado no livro-texto e slides da Profa. Sandra de Amo*