Árvores

Vimos inicialmente um conceito mais amplo de árvore e depois o restringimos fixando o número máximo de filhos que um nó pode ter em dois.

Posteriormente, voltamos a permitir a determinação de mais itens de dados e mais filhos por nó tendo como resultado árvores denominadas **Multivias** ou **M-vias**.

Vimos que uma estrutura multivia com algoritmo eficiente deve considerar :

- Tempo de acesso a cada nó
- Balanceamento da árvore

UNIVASF

Árvores

Trabalhamos como exemplo de uma árvore multivia a árvore 2-3-4.

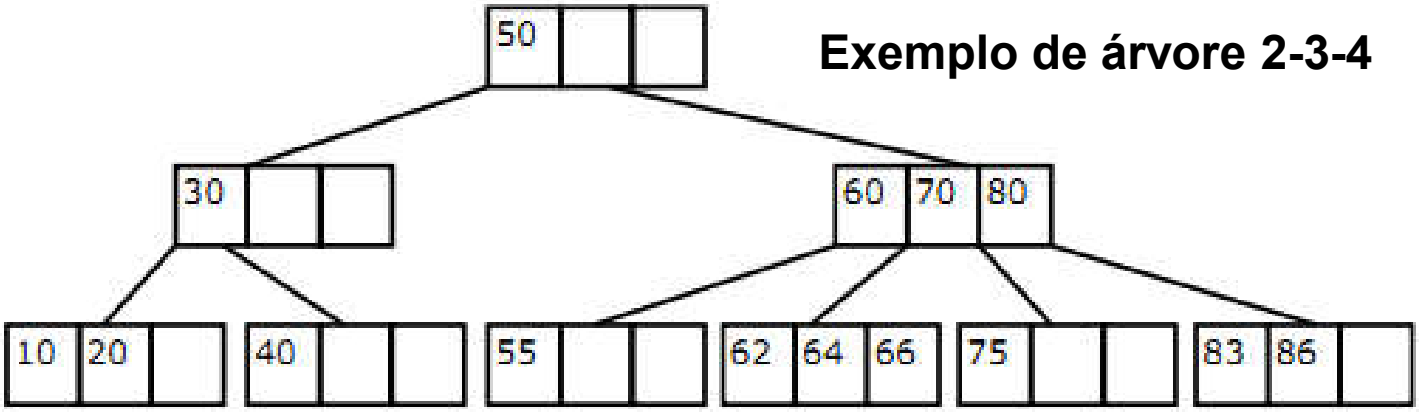
Lembrando, uma árvore 2-3-4 pode ter até quatro filhos e três itens de dados por nó.

Vimos o porque de estudar árvores 2-3-4:

- São árvores balanceadas;

- São fáceis de implementar;

- Servem como uma introdução para o estudo de árvores B.



Exemplo de árvore 2-3-4

Árvores

Em uma árvore 2-3-4 cada nó pode conter um, dois ou três itens de dados.

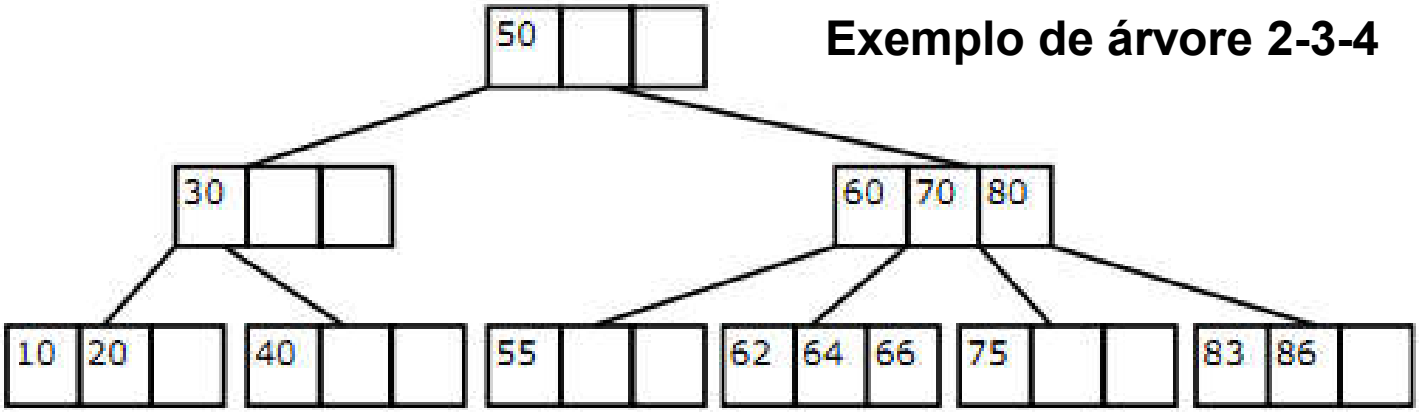
Um nó interno deve sempre ter um filho a mais que seus itens de dados.

Devido a uma árvore 2-3-4 possuir nós com até quatro filhos, ela é chamada de árvore multivias de ordem 4.

Em uma árvore binária, todos os filhos com chaves menores que a chave do nó estão “enraizados” no nó filho à esquerda, e todos os filhos com chaves maiores estão “enraizados” no nó filho à direita.

UNIVASF

Árvores



Exemplo de árvore 2-3-4

Na árvore 2-3-4 o princípio é o mesmo, com alguns detalhes a mais:

- todos os itens de dados na subárvore “enraizada” no filho 0, possuem valores menores que o da chave 0;

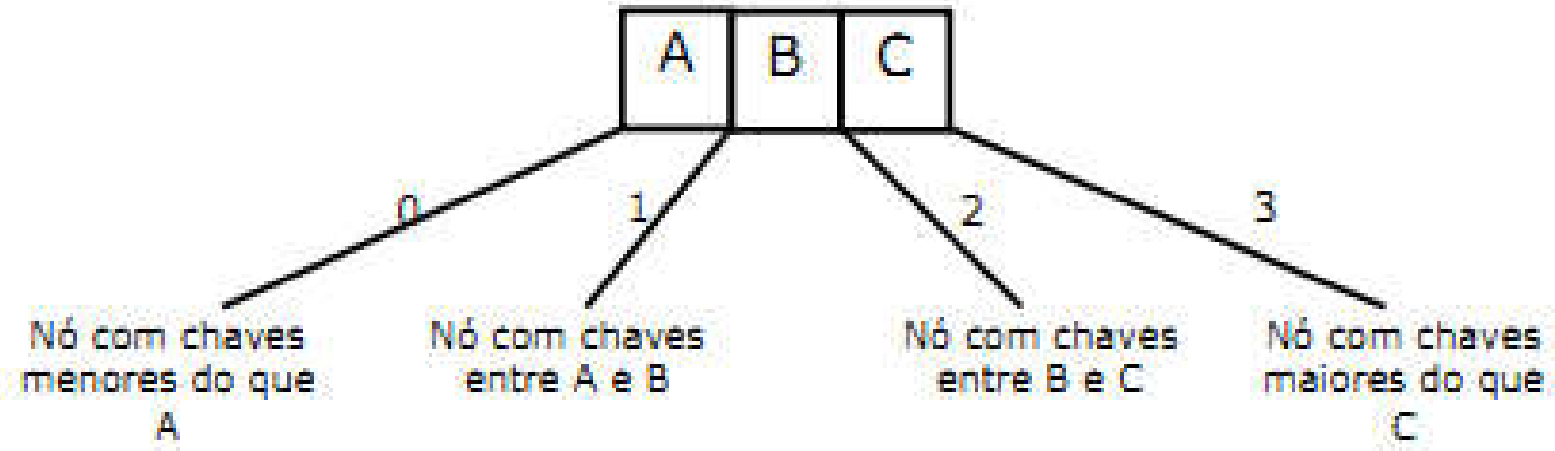
- todos os itens de dados na subárvore “enraizada” no filho 1, possuem valores maiores do que o da chave 0, mas menores do que a chave 1;

Árvores

- todos os itens de dados na subárvore “enraizada” no filho 2, possuem valores maiores do que o da chave 1, mas menores do que a chave 2;

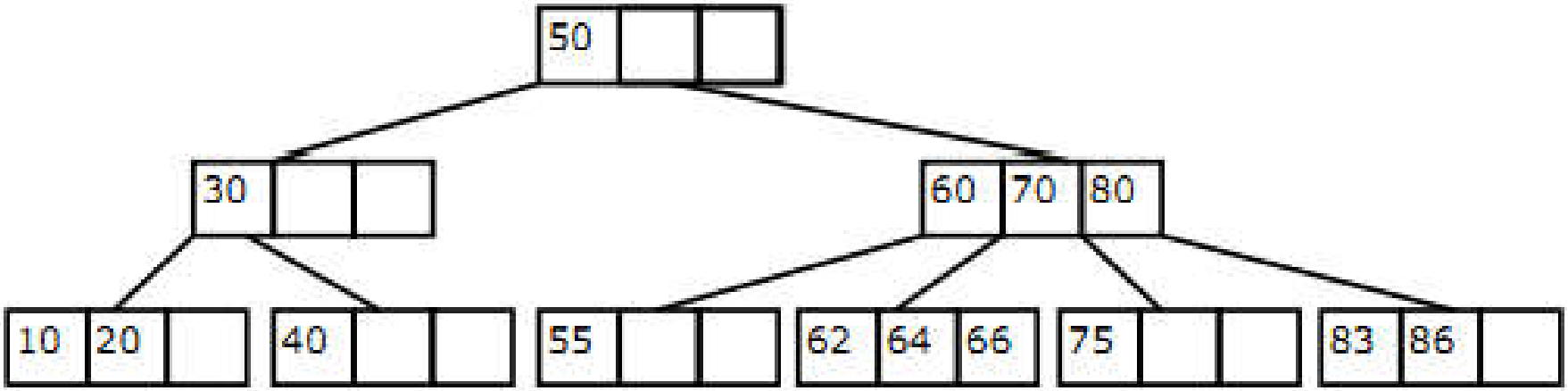
- todos os itens de dados na subárvore “enraizada” no filho 3, possuem valores maiores do que o da chave 2.

Valores duplicados geralmente não são permitidos, o que possibilita que não nos preocupemos com comparações de chaves iguais.



Árvores

Um exemplo de como ocorre a pesquisa por uma chave em uma árvore 2-3-4. Cosiderando a busca pela chave 64.



Iniciamos a busca pela raiz e ao não entrar a chave percebemos que a chave é maior que 50 e portanto a busca segue no filho 1, o qual também não contém a chave em seus itens de dados. Como 64 está entre 60 e 70 a busca segue novamente para o filho 1. Desta vez a chave é localizada.

Árvores

Trataremos agora do processo de inserção de uma nova chave em uma árvore 2-3-4.

Por que novos itens de dados são sempre inseridos nas folhas?

Porque se os itens forem inseridos em um nó com filhos, então o número de filhos necessitará ser mudado para manter a estrutura da árvore, que estipula que esta árvore deve ter um filho a mais do que os itens de dados em um nó.

O processo de inserção começa pela busca do nó folha apropriado.

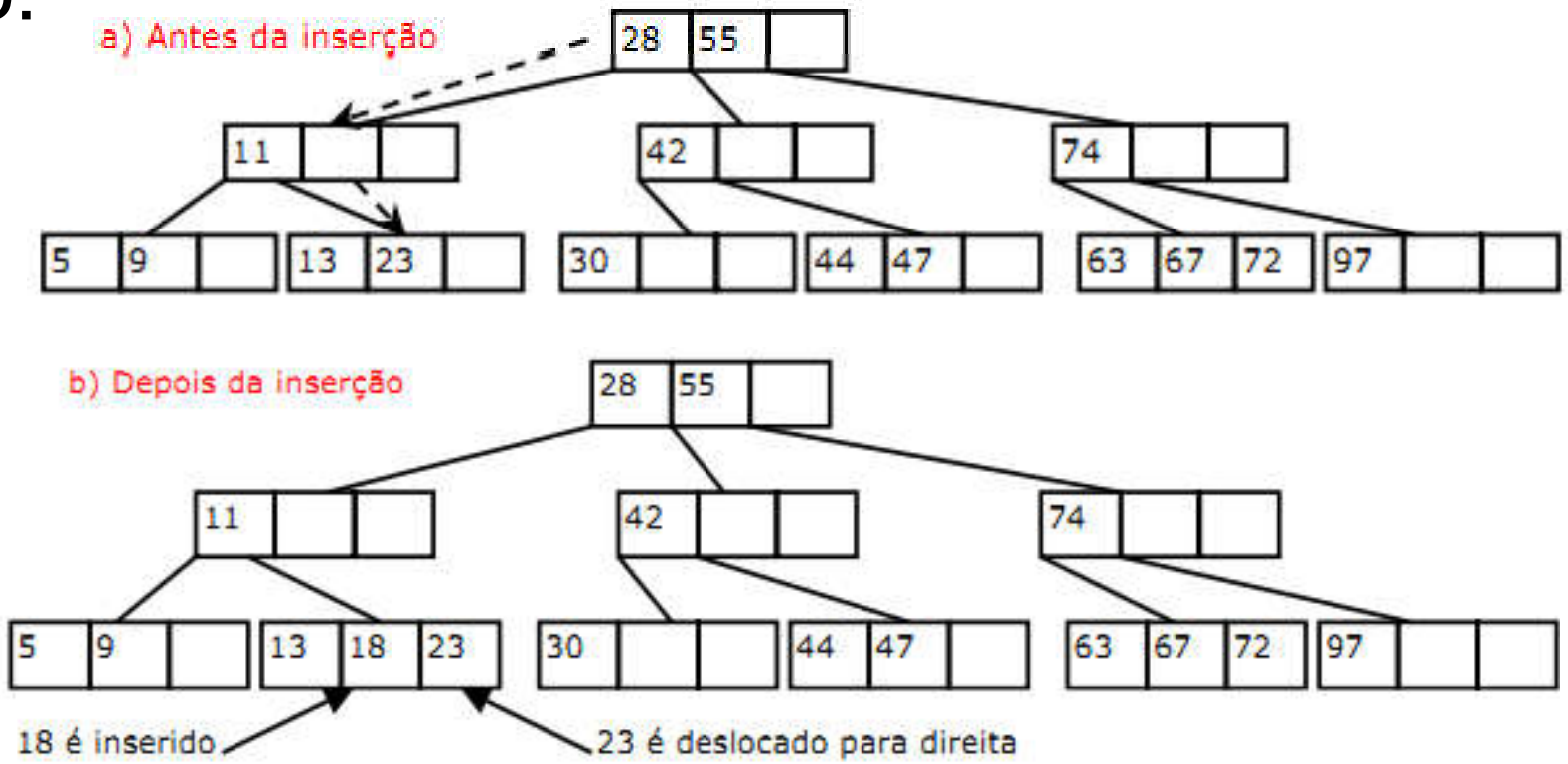
Se apenas nós que não estão cheios são encontrados durante a busca, a inserção é mais fácil.

Árvores

A inserção pode envolver a movimentação de um ou dois itens de dados em um nó.

Pois, as chaves deverão estar na ordem correta após o novo item ser inserido.

Exemplo da inserção item 18 na árvore abaixo:



Árvores

Como dito, as inserções tornam-se mais complicadas se um nó cheio é encontrado no caminho abaixo do ponto de inserção.

Quando isso ocorre o nó precisa ser dividido.

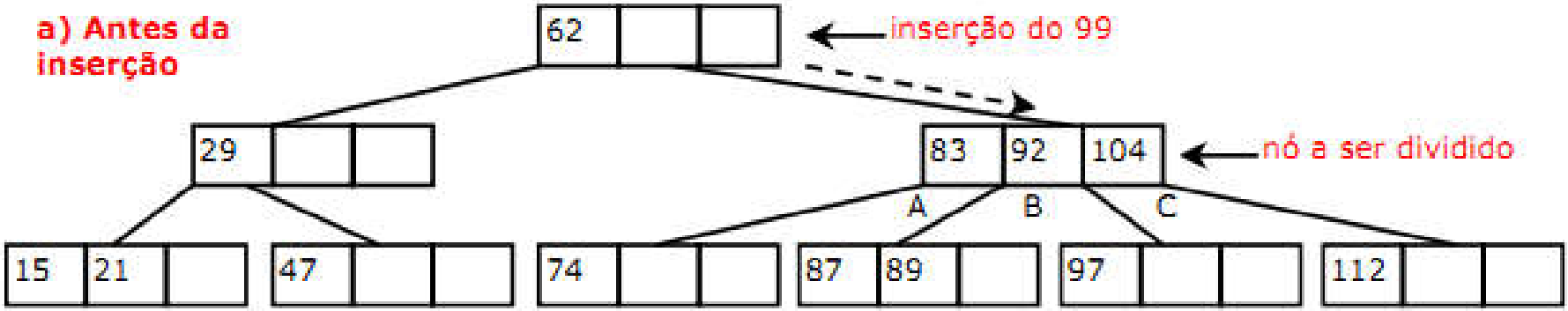
É este processo de divisão que mantém a árvore balanceada.

O tipo de árvore 2-3-4 que estudamos é frequentemente chamado de árvore 2-3-4 top-down, por que os nós são divididos "para baixo" do ponto de inserção.

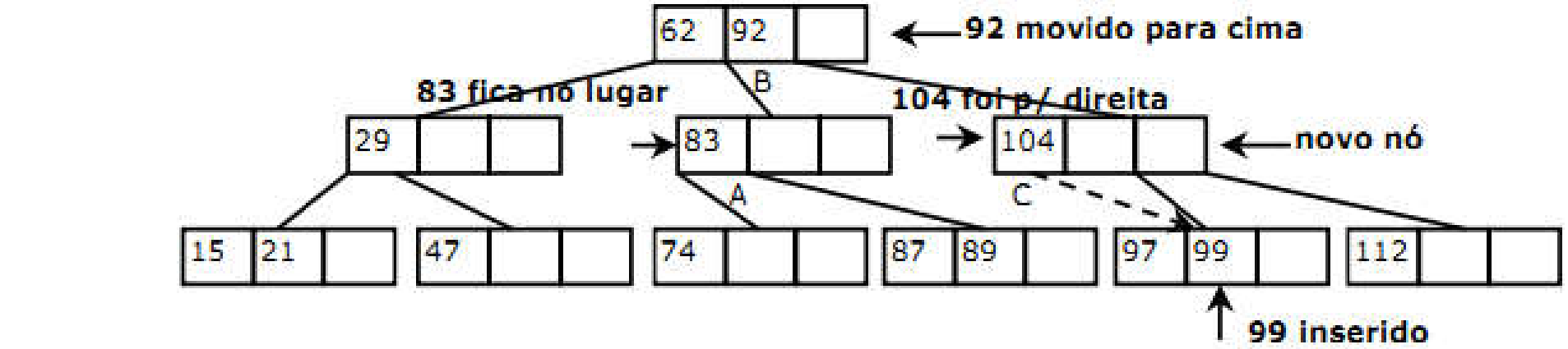
Nomeando os itens de dados a serem divididos como A, B e C.

Assumindo que o nó a ser dividido não é a raiz, recordaremos a divisão do nó raiz depois.

Árvores



- Um novo nó vazio é criado. Ele é parente (*sibling*) do nó que está sendo dividido, e é colocado a sua direita;
- O item de dado C é movido para o novo nó;
- O item de dado B é movido para o pai do nó que está sendo dividido;
- O item de dado A fica aonde ele está;
- Os dois filhos mais à direita são desconectados do nó que está sendo dividido e são conectados no novo nó.



Árvores

Quando uma raiz cheia é encontrada no início da busca para encontrar o ponto de inserção, o resultado da inserção é ligeiramente mais complicado:

Um novo nó é criado, tornando-se a nova raiz, e a antiga raiz é dividida criando um novo nó irmão;

O item de dado C é movido para o novo nó irmão da antiga raiz;

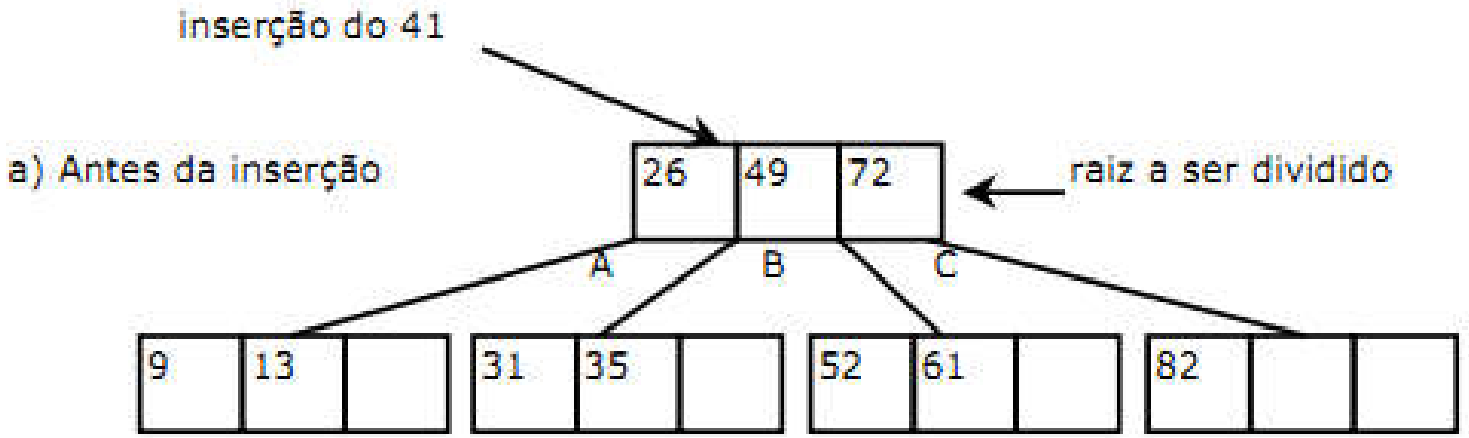
O item de dado B é movido para a nova raiz;

O item de dados A é deixado onde está;

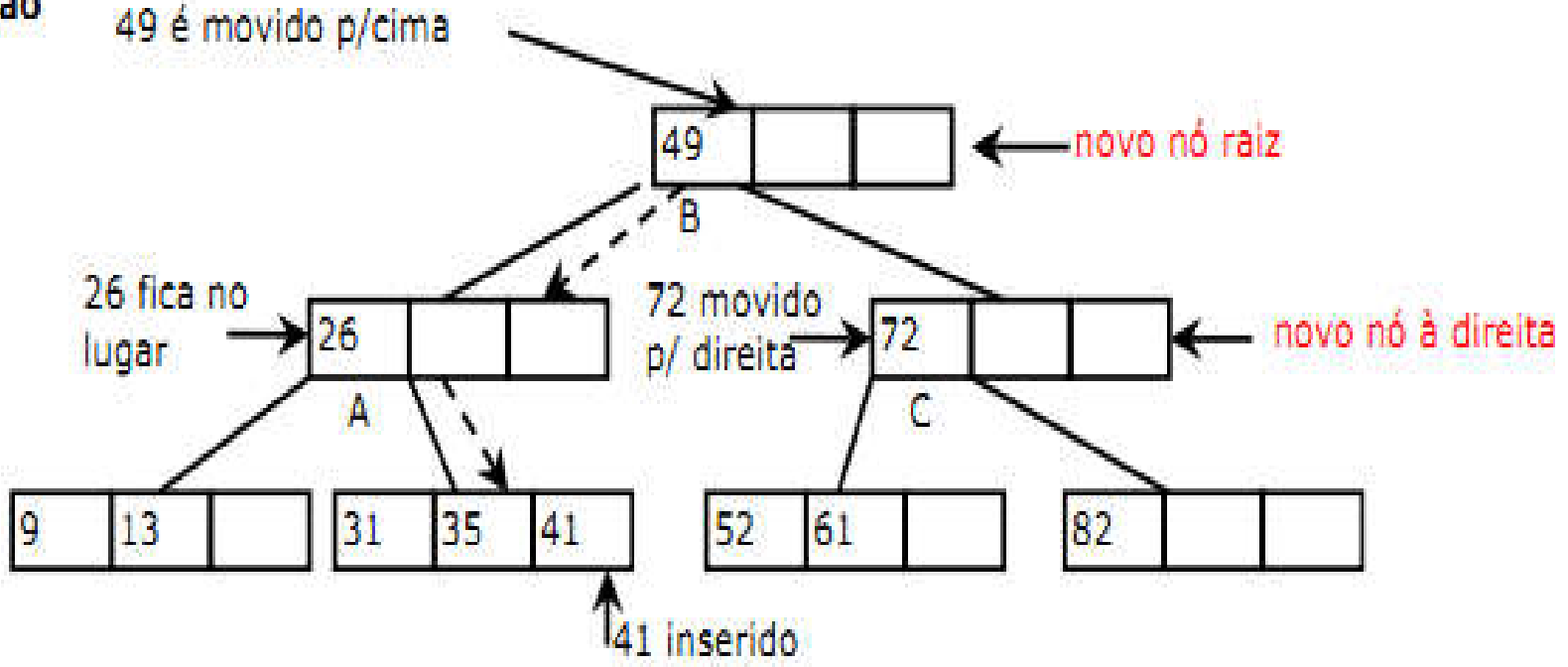
Os dois filhos mais a direita do nó que está sendo dividido são desconectados dele e conectados no novo nó do lado direito.

UNIVASF

Árvores



b) Após a inserção



Árvores B, B* e B+

Árvores Múltiplas

Como vimos na revisão efetuada, a definição de árvore não restringe o número de filhos que um nó pode ter.

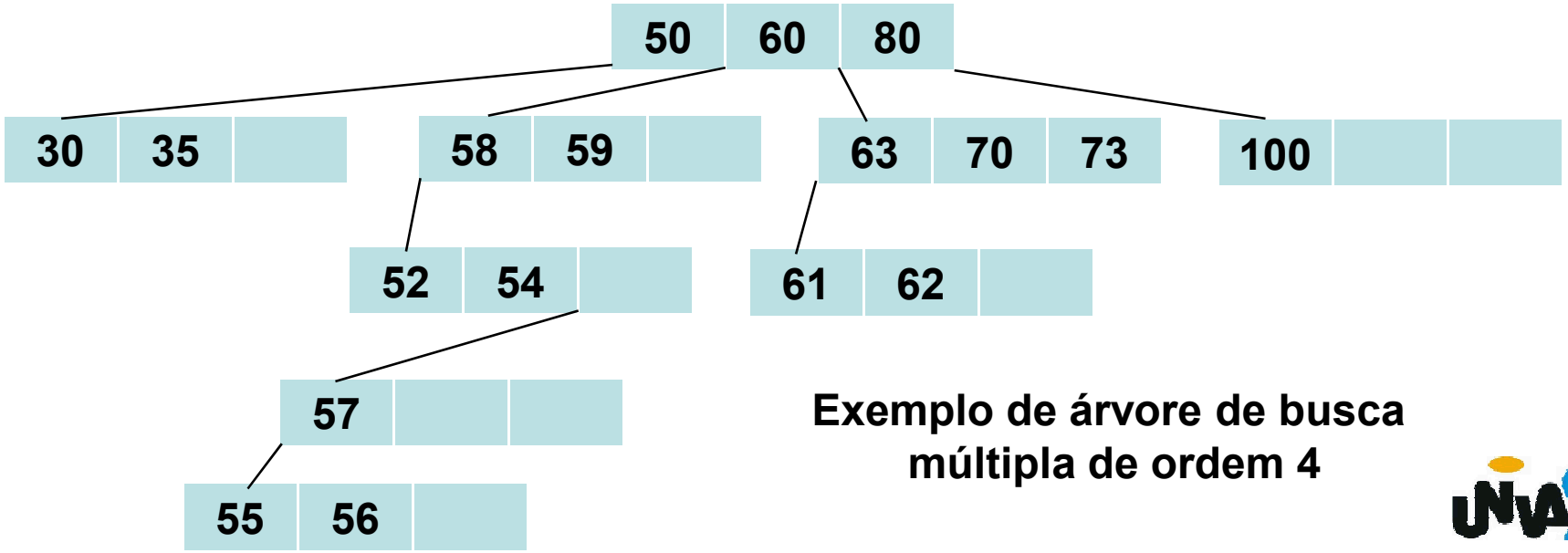
Relembramos os conceitos e procedimentos associados às árvores 2-3-4, um tipo particular de árvore M-vias, também denominada árvore múltipla.

Contudo, para que uma árvore mantenha características úteis, relacionadas à sua eficiência no que tange localizar um determinado dado, restrições são impostas. Veremos agora restrições impostas a uma árvore múltipla para que esta receba o rótulo de ***árvore de busca múltipla de ordem m***.

UNIVASF

Árvores Múltiplas

1. cada nó tem até m filhos e m - 1 chaves;
2. as chaves em cada um dos nós estão em ordem ascendente;
3. as chaves nos primeiros i filhos são menores do que na chave i-ésima;
4. as chaves nos últimos m - i filhos são maiores do que na chave i-ésima.



Exemplo de árvore de busca múltipla de ordem 4

Árvores Múltiplas

Neste ponto algumas perguntas podem atormentar os discentes, como por exemplo:

Por que estudar árvores múltiplas?

Qual a vantagem de uma árvore múltipla com relação a uma árvore binária?

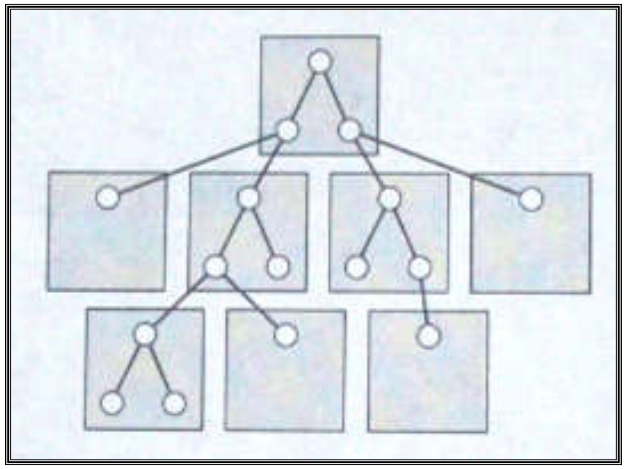
Para respondermos a estas perguntas devemos ter em mente que a unidade básica de operações de E/S é o bloco. Ou seja, quando uma informação é lida ou gravada em um disco, o bloco inteiro que a contem é transferido. Outra informação relevante é que o tempo para acesso a um dado é obtido através da seguinte equação:

tempo de acesso = tempo de procura + atraso rotacional (latência) + tempo de transferência

UNIVASF

Árvores Múltiplas

A árvore binária abaixo nos ajuda a compreender melhor.



Considerando que cada quadrado representa um bloco e imaginando a situação hipotética na qual deseja-se transferir 10KB de um disco onde são necessários 40ms para localizar uma trilha, com taxa de transferência de 1000KB/s e latência de 10ms. Temos:

tempo de acesso = 40ms + 10ms + 10ms = 60ms

Árvores Múltiplas

Agora imagine que estes 10KB estivem separados em dois blocos. Logo, teríamos:

tempo de acesso = 2*(40ms+10ms+5ms) = 110ms

Em programas que manipulam bancos de dados, onde a maioria das informações está guardada em discos (armazenamentos secundários), a escolha de uma estrutura de dados que minimize os acessos ao dispositivo de armazenamento secundário é crucial.

Estudaremos agora uma estrutura de dados que apresenta tal característica.

UNIVASF

Árvores B

Proposta por Bayer e McCreight em 1972 a **árvore B** opera junto com o armazenamento secundário e pode ser sintonizada para reduzir os transtornos impostos por essa armazenagem.

Uma propriedade importante das árvores B é o tamanho de cada nó, que pode ser tão grande quanto o tamanho de um bloco.

Uma árvore B é uma árvore de busca múltipla de ordem **m** com as seguintes propriedades:

1. a raiz tem pelo menos duas subárvores, a menos que ela seja uma folha;

2. cada nó não-raiz e não-folha contém k-1 chaves e k ponteiros para as subárvores onde $\lceil m/2 \rceil \leq k \leq m$;

Árvores B

3. cada nó folha (não raiz) contém k-1 chaves, onde $\lceil m/2 \rceil \leq k \leq m$;

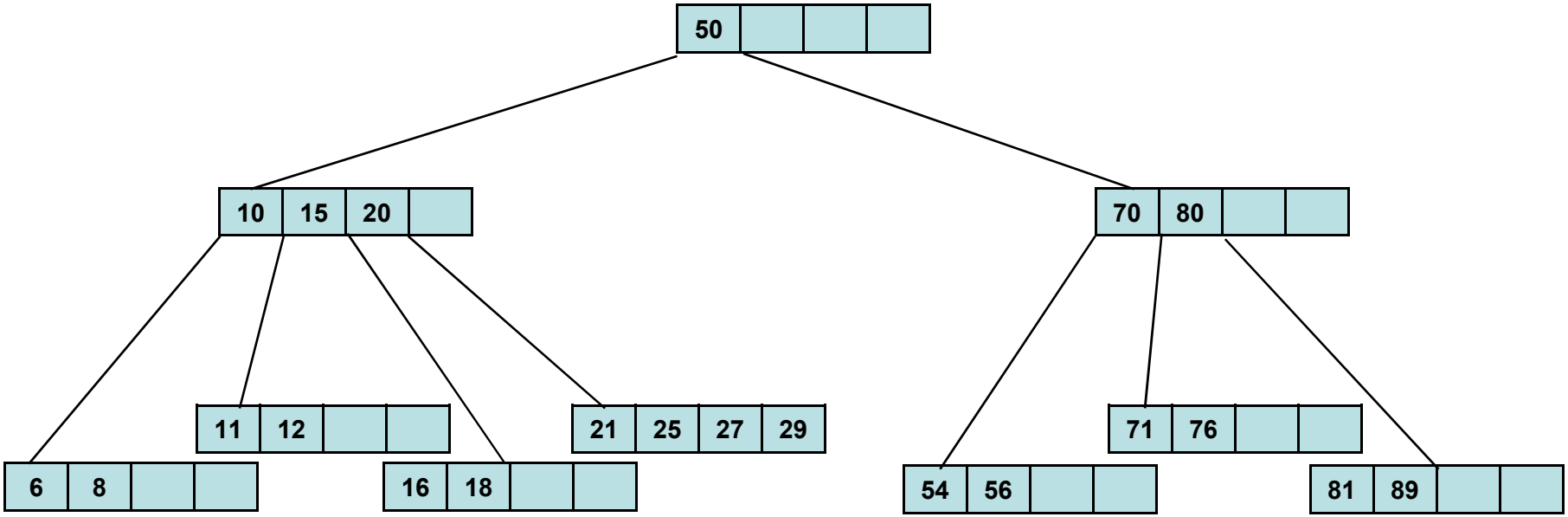
4. todas as folhas estão no mesmo nível.

Com base no que foi dito, proponha um tipo de dado que represente adequadamente um nó de uma árvore B.

```
#define ordem 5
typedef int TipoChave;
typedef struct _No{
   char ehFolha;
   int numChaves;
   TipoChave chaves[ordem-1];
   struct _No *filhos[ordem];
}No;
typedef No *ArvoreB;
```

Árvores B

Abaixo vemos um exemplo de uma árvore B de ordem 5.



Árvores B

Qual a altura de uma árvore B no pior caso?

O pior caso ocorre quando a árvore possui o menor número permissível de ponteiros por nó não-raiz, $q = \lceil m/2 \rceil$, e a busca tem que atingir uma folha. Neste caso, em uma árvore B de altura h, existem

1 chave na raiz +

2(q-1) chaves no segundo nível +

2q(q-1) chaves no terceiro nível +

$2q^2(q-1)$ *chaves no quarto nível +*

...

$2q^{h-2}(q-1)$ *chaves nas folhas (nível h) =*

$$1 + \left(\sum_{i=0}^{h-2} 2q^i\right)(q-1) \textit{ chaves na árvore B}$$

Com a fórmula para a soma dos primeiros n elementos em uma progressão geométrica,

$$\sum_{i=0}^{n} q^i = \frac{q^{n+1} - 1}{q - 1}$$

O número de chaves na árvore B, no pior caso, pode ser expresso como

$$1 + 2(q-1)\left(\sum_{i=0}^{h-2} q^i\right) = 1 + 2(q-1)\left(\frac{q^{h-1} - 1}{q - 1}\right) = -1 + 2q^{h-1}$$

A relação entre o número de chaves em qualquer árvore B e a altura da árvore B é então expressa como

$$n \geq -1 + 2q^{h-1}$$

Logo temos:

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

Árvores B

Isso significa que para uma ordem m suficientemente grande a altura é pequena mesmo para um grande número de chaves armazenadas na árvore B. **Obs.: m tende a ser grande, normalmente varia entre 50 e 500.**

Por exemplo, para m=200 e n=2.000.000, temos h<=4.

Com base no que foi apresentado elabore uma função, na linguagem C, que efetue uma busca por uma determinada chave em uma árvore B.

UNIVASF

```
ArvoreB buscaEmArvoreB (TipoChave chave,
ArvoreB arvore)
{
  if (arvore != NULL)
  {
     int i;
     for (i=0; i<arvore->numChaves &&
    arvore->chaves[i]<chave; i++);
     if (i+1>arvore->numChaves ||
    arvore->chaves[i]>chave)
        return buscaEmArvoreB(chave,
        arvore->filhos[i]);
     else
        return arvore;
   }
   else
     return NULL;
}
```

Árvores B

Vamos agora analisar o processo de inserção de uma nova chave em uma árvore B.

Em uma análise inicial temos uma tendência, considerando as características de uma árvore B, como por exemplo, a restrição de todas as folhas terem que estar no mesmo nível (restrição esta que não é imposta em uma árvore binária balanceada), que a operação será muito complexa.

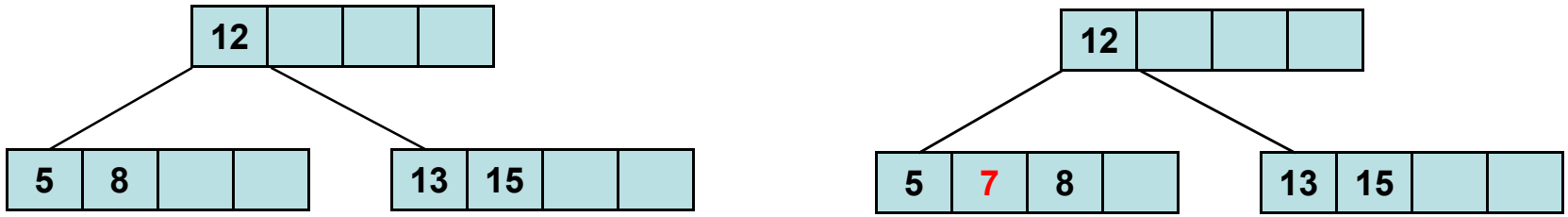
Contudo, quando mudamos a estratégia de construção da árvore a tarefa de inserção, mantendo o balanceamento, torna-se mais fácil. Vamos esquematizar o processo de inserção e analisar exemplos.

UNIVASF

Árvores B

Existem três situações comuns encontradas quando se insere uma chave em uma árvore B:

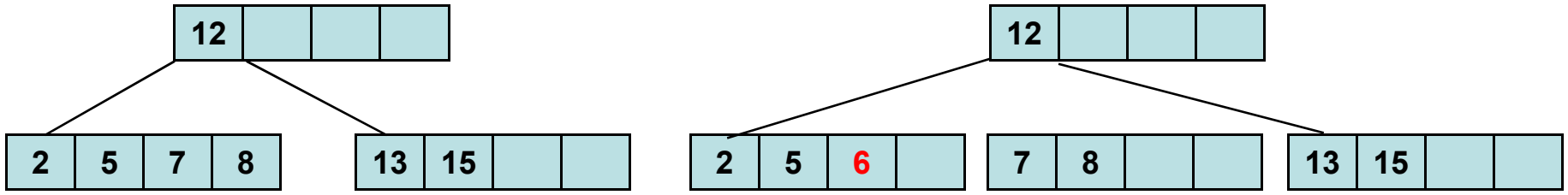
1. Uma chave é colocada em uma folha que ainda tem algum espaço, como na árvore abaixo (considerando a inserção da chave 7).



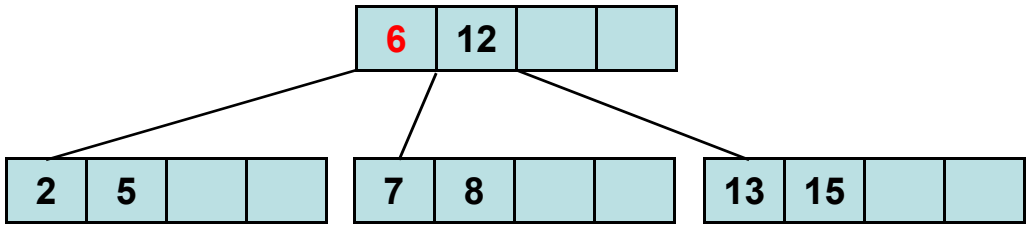
Nesta árvore B de ordem 5, uma nova chave, 7, é colocada na árvore, preservando a ordem das chaves na folha, de modo que a chave 8 precisou ser deslocada para a direita em uma posição.

Árvores B

2. A folha na qual uma chave precisa ser colocada está cheia, como na árvore abaixo (considerando a inserção da chave 6).



Neste caso, a folha é dividida, criando uma nova folha, e metade das chaves é movida da folha cheia para a nova.

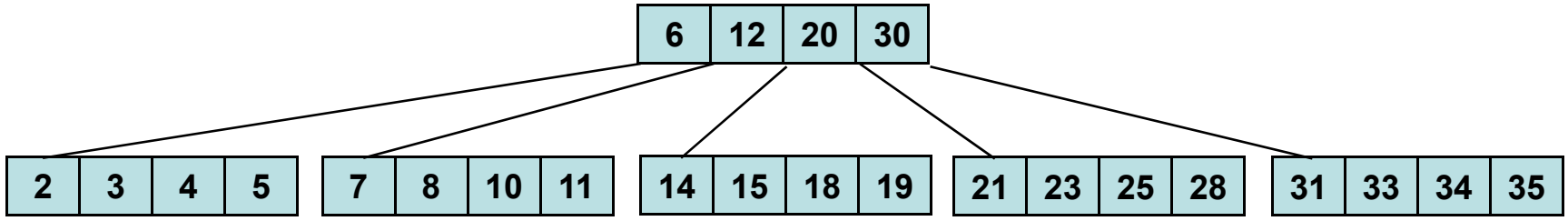


Mas a nova folha tem que ser incorporada na árvore B. A última chave da folha velha é movida para o ascendente e um ponteiro para a nova folha é colocado no ascendente em questão.

Árvores B

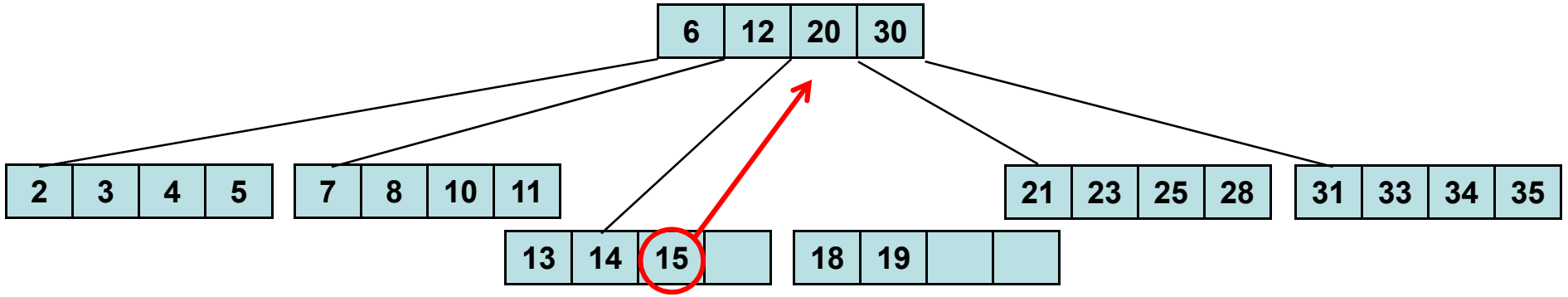
O mesmo procedimento pode ser repetido para cada nó interno da árvore B, de modo que cada divisão adicione mais um nó à árvore B. Além disso, tal divisão garante que cada folha nunca tenha menos do que $\lceil n/2 \rceil - 1$ chaves.

3. Um caso especial surge se a raiz da árvore B está cheia. Neste caso, uma nova raiz e um novo irmão da raiz existente têm que ser criados. Essa divisão resulta em dois novos nós na árvore B. Por exemplo, depois de inserir a chave 13 na terceira folha da árvore abaixo,

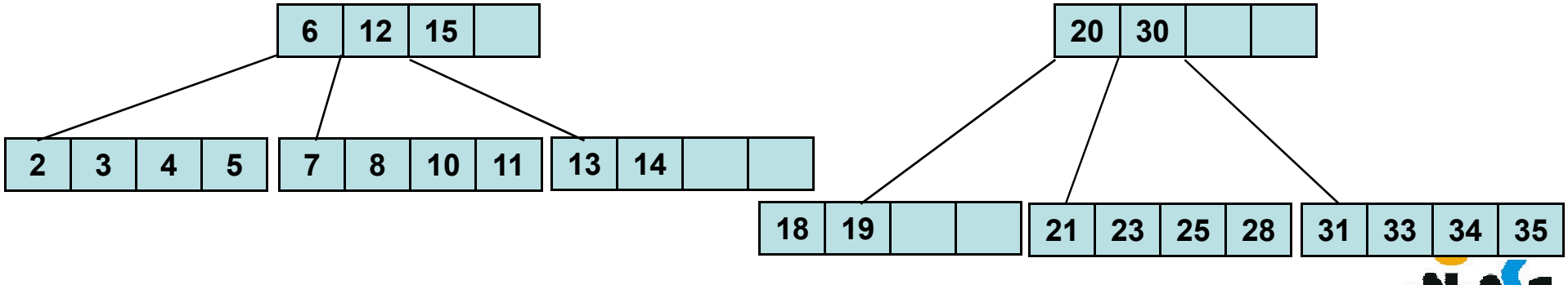


Árvores B

a folha é dividida (como no caso 2), uma nova folha é criada e a chave 15 está para ser movida para o ascendente, mas este não tem espaço para ela.

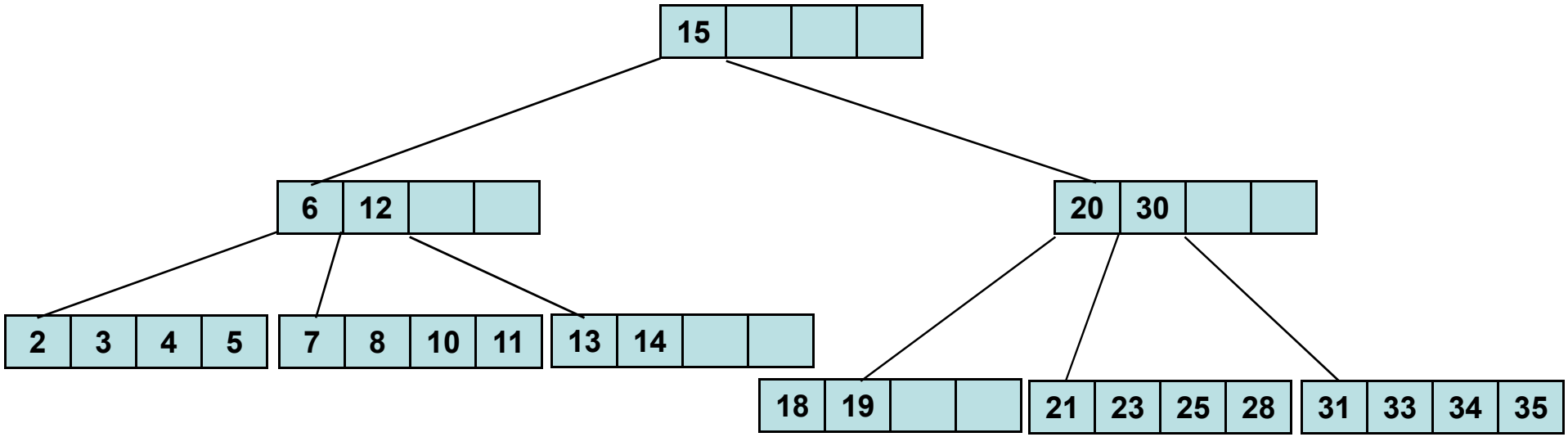


Assim, o ascendente é dividido,



Árvores B

mas agora duas árvores B têm que ser combinadas em uma. Isso é obtido criando-se uma nova raiz e movendo-se a última chave da velha raiz até ela. Deve ser óbvio que é o único caso no qual a árvore B aumenta em altura.



Com base no procedimento descrito elabore um algoritmo em alto nível, representando-o através de um pseudocódigo, descrevendo o processo de inserção.