Algoritmos e estruturas de dados II

autoria: Guilherme P. Telles
IC - UNICAMP

ICMC - Rosane Minghim

10 de junho de 2013

Árvore B

- Rudolf Bayer e Eduard M. McCreight, 1970.
- Intuitivamente uma árvore B é um índice multi-camadas com um mecanismo eficiente para gerenciar a inserção e a remoção de chaves.
- A nomenclatura em torno das árvores B é um pouco variada.

Árvores B

- Cada nó da árvore B contém chaves e apontadores para nós.
- (Associados à cada chave está um apontador para os dados de interesse da aplicação.)
- Generaliza a árvore binária de busca, no sentido de que as chaves em um nó particionam as chaves nas subárvores em intervalos.

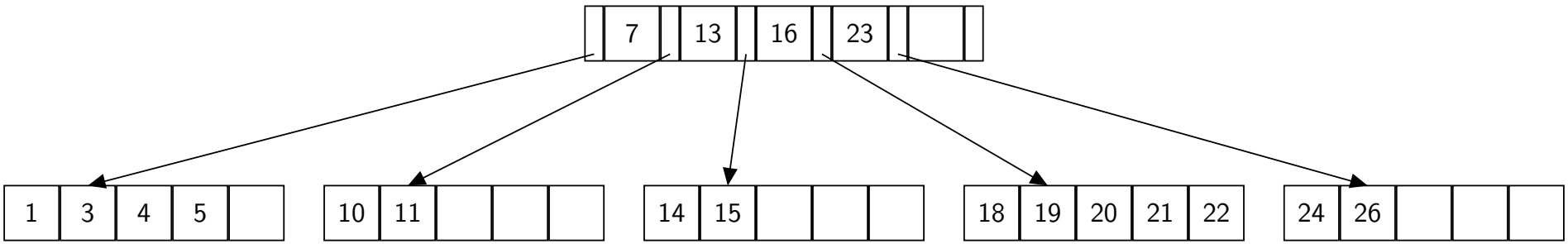
Definição

- Uma árvore B de grau t é uma árvore enraizada.
- Cada nó interno com n chaves tem $n+1$ filhos.
- Cada nó, exceto pela raiz, tem pelo menos $t-1$ chaves e no máximo $2t-1$ chaves.
- A raiz é uma folha ou tem pelo menos 1 chave.
- Todas as folhas têm a mesma altura h.
- As n chaves em um nó x estão em ordem não-decrescente e particionam as chaves armazenadas nas suas $n+1$ subárvores (filhos):

$$K^{c_1^x} \leq k_1^x \leq K^{c_2^x} \leq k_2^x \leq K^{c_3^x} \leq \ldots \leq k_n^x \leq K^{c_{n+1}^x},$$

onde k_i^x é a i-ésima chave em x e $K^{c_i^x}$ são as chaves na i-ésima subávore de x.

Exemplo



Árvores B

- A altura h de uma árvore B de grau t com m chaves é

$$h \leq \log_t \frac{m+1}{2}.$$

- O grau é escolhido para que seja possível ler e gravar um nó inteiro em disco com apenas uma operação de *seek* (p.ex. um bloco do disco).
- Normalmente a raiz é mantida sempre na memória.

Busca

- A busca por uma chave k é similar à busca em uma árvore binária de busca na memória.
- A busca começa na raiz.
- Cada nó x é trazido do disco para a memória e inspecionado para encontrar o primeiro valor de chave k_i^x maior ou igual a k.
- Se $k_i^x = k$, o registro foi encontrado e a busca termina.
- Se $k_i^x > k$ há duas possibilidades: x é uma folha e a busca termina sem encontrar o registro ou x não é uma folha e a busca prossegue recursivamente no nó apontado por k_i^x.

Busca

B-TREE-SEARCH(x, k)

```
 1  i = 1
 2  while i ≤ x.n and k > x.key[i]
 3        i = i + 1
 4  if i ≤ x.n and k == x.key[i]
 5        return (x, i)
 6  if x.leaf
 7        return NULL
 8  else
 9        DISK-READ(x.c[i])
10        return B-TREE-SEARCH(x.c[i], k)
```

Inserção e Remoção

- A árvore B mantém o balancemanto através da divisão e fusão de nós.
- Há duas abordagens para a inserção ou remoção:
  1. realizar a operção com uma passada começando da raiz e depois percorrer o caminho novamente consertando a árvore.
  2. realizar apenas uma passada pela árvore, ajustando os nós que potencialmente podem ser afetados pela operação antecipadamente, e realizar a operação.

Divisão de um nó

- A divisão de um nó é necessária sempre que um nó fica cheio.
- O procedimento para dividir um nó cheio y filho de um nó x que não está cheio é o seguinte:
  1. Seja med a mediana das chaves de y.
  2. Crie um novo nó z.
  3. Transfira os filhos de y com chaves maiores que med para z.
  4. Acrescente a chave med a x.
  5. Ajuste o apontador com chave med para que aponte para z.
  6. Grave x, y e z no disco.

Divisão

B-Tree-Split-Child(x, i, y)

```
 1  z = Allocate-Node()
 2  z.leaf = y.leaf
 3  z.n = t − 1
 4  for j = 1 to t − 1
 5        z.key[j] = y.key[j + t]
 6  if not y.leaf
 7        for j = 1 to t
 8              z.c[j] = y.c[j + t]
 9  y.n = t − 1
10  for j = x.n + 1 downto i + 1
11        x.c[j + 1] = x.c[j]
12  for j = x.n downto i
13        x.key[j + 1] = x.key[j]
14  x.key[i] = y.key[t]
15  x.n = x.n + 1
16  Disk-Write(y);  Disk-Write(z);  Disk-Write(x)
```

Inserção

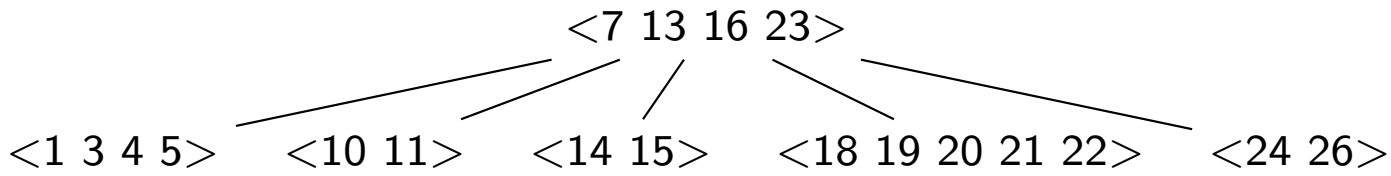
- A inserção na árvore acontece em uma folha que não está cheia.
- O procedimento avança da raiz para as folhas, dividindo os nós que estiverem cheios ao longo do caminho.
- A única forma pela qual a altura de uma árvore aumenta é pela divisão da raiz durante uma inserção.

B-Tree-Insert

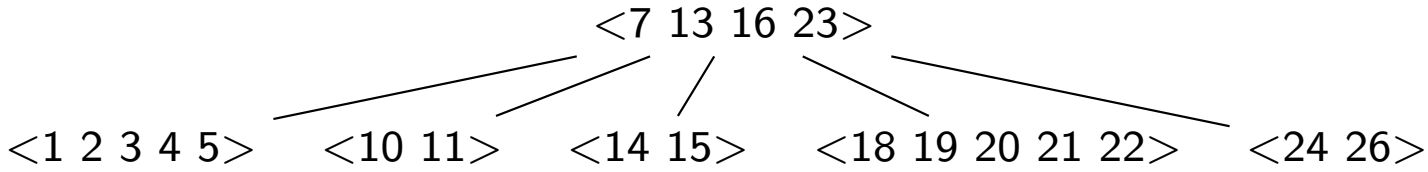
```
B-Tree-Insert(T, k)
 1  r = T.root
 2  if r.n == 2t − 1
 3       s = ALLOCATE-NODE()
 4       T.root = s
 5       s.leaf = FALSE
 6       s.n = 0
 7       s.c[1] = r
 8       B-TREE-SPLIT-CHILD(s, 1, r)
 9       B-TREE-INSERT-NONFULL(s, k)
10  else
11       B-TREE-INSERT-NONFULL(r, k)
```

B-Tree-Insert-Nonfull(x, k)

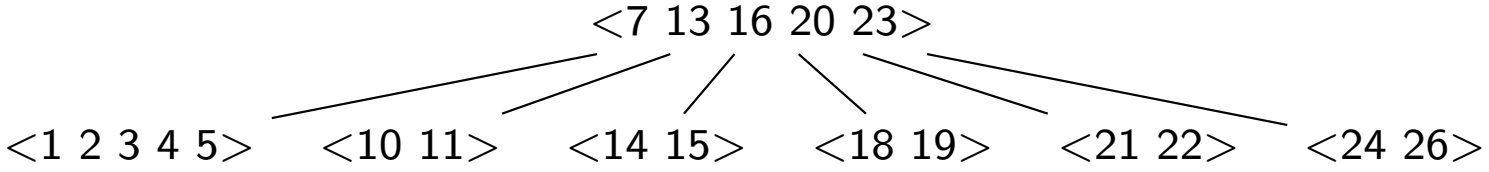
```
 1  i = x.n                                   // assume x is not full
 2  if x.leaf
 3       while i ≥ 1 and k < x.key[i]
 4            x.key[i + 1] = x.key[i]
 5            i = i − 1
 6       x.key[i + 1] = k
 7       x.n = x.n + 1
 8       Disk-Write(x)
 9  else
10       while i ≥ 1 and k < x.key[i]
11            i = i − 1
12       i = i + 1
13       Disk-Read(x.c[i])
14       if x.c[i].n == 2t − 1              // child x.c[i] is full
15            B-Tree-Split-Child(x, i, x.c[i])
16            if k > x.key[i]
17                 i = i + 1
18       B-Tree-Insert-Nonfull(x.c[i], k)
```

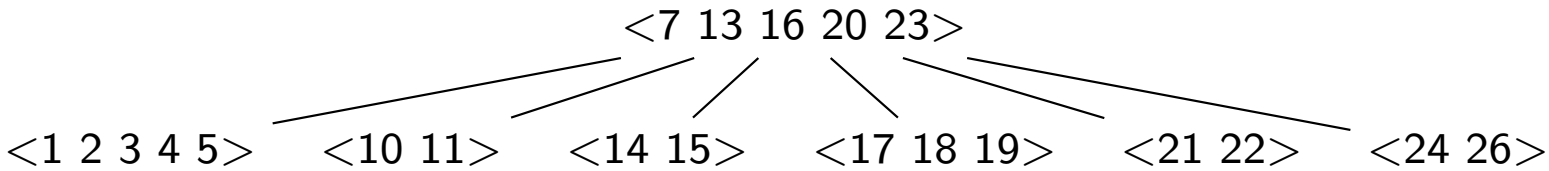

<7 13 16 23>
<1 3 4 5>
<10 11>
<14 15>
<18 19 20 21 22>
<24 26>

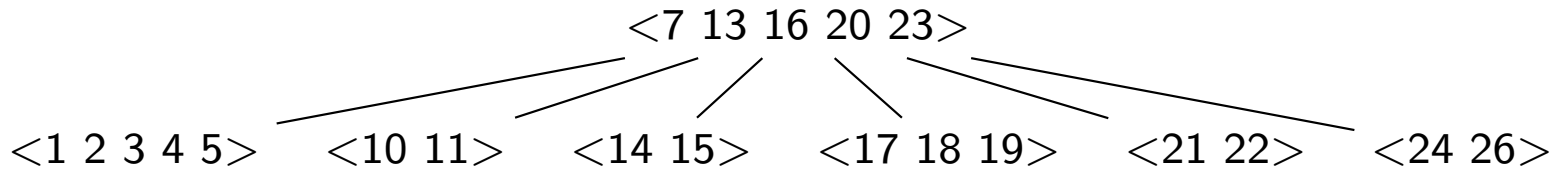
+2


<7 13 16 23>
<1 2 3 4 5>
<10 11>
<14 15>
<18 19 20 21 22>
<24 26>

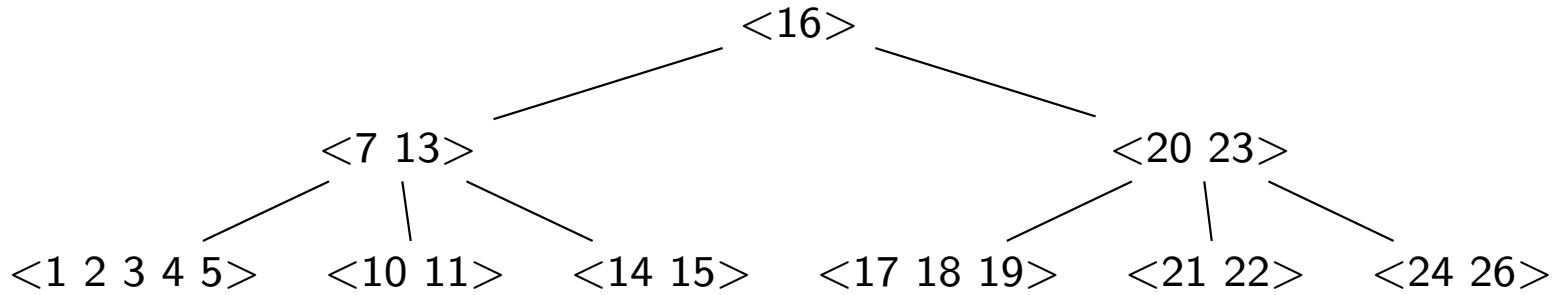
+17

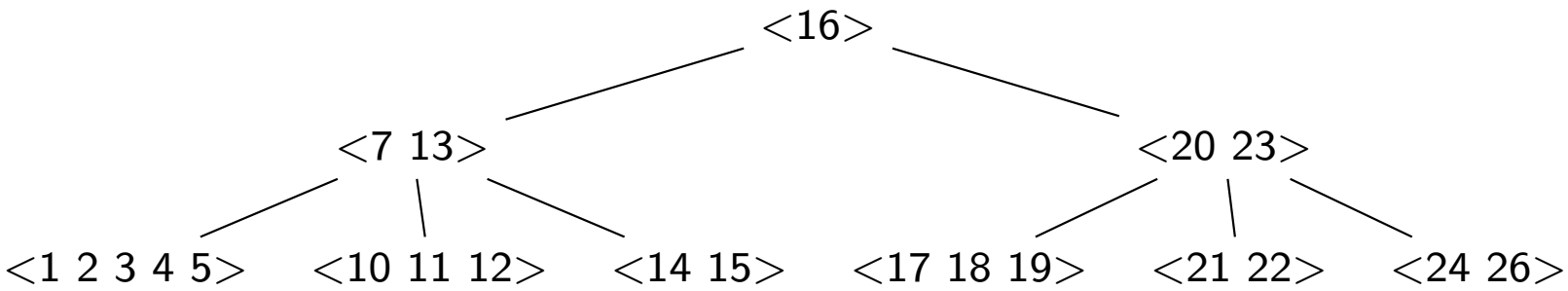

<7 13 16 20 23>
<1 2 3 4 5>
<10 11>
<14 15>
<18 19>
<21 22>
<24 26>

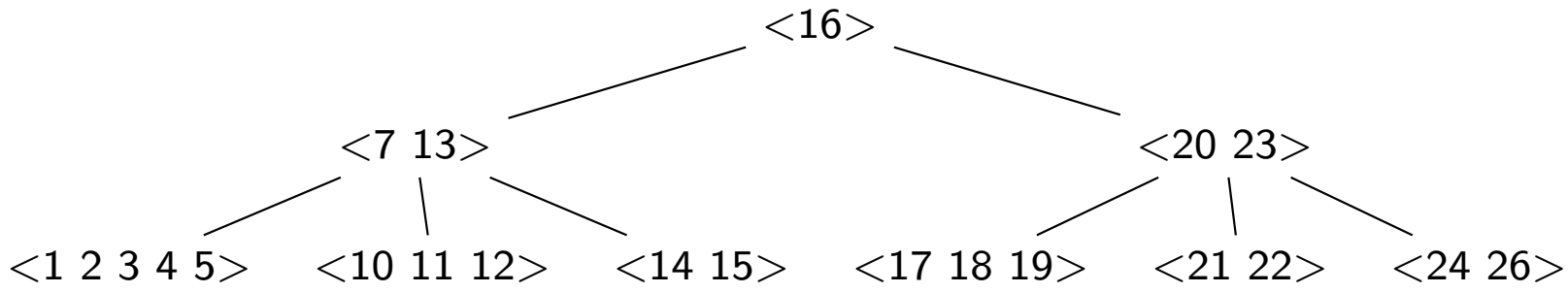

<7 13 16 20 23>
<1 2 3 4 5>
<10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>


<7 13 16 20 23>
<1 2 3 4 5>
<10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>

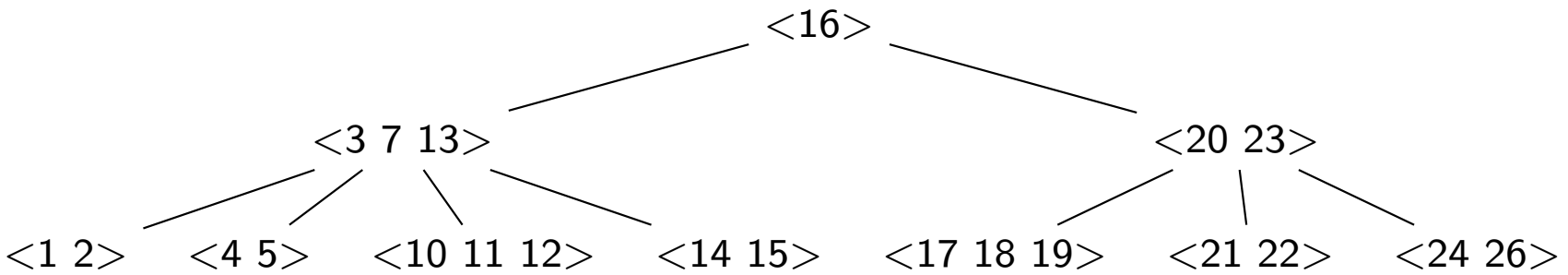
+12

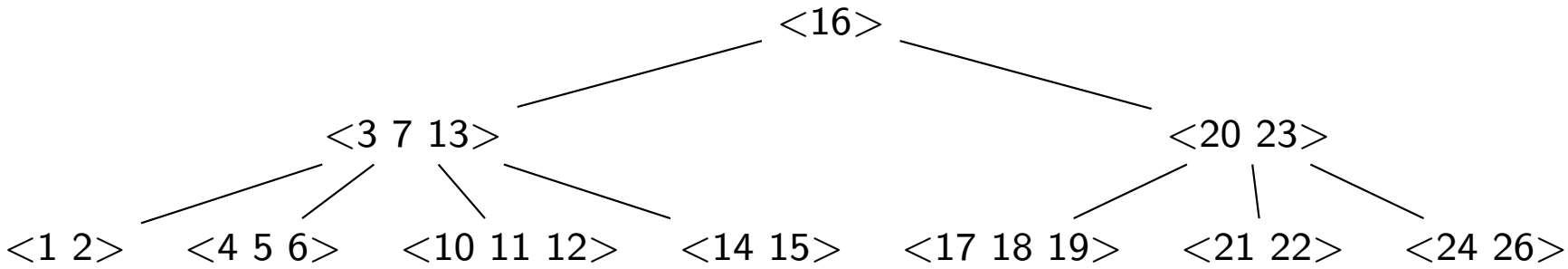

<16>
<7 13>
<20 23>
<1 2 3 4 5>
<10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>


<16>
<7 13>
<20 23>
<1 2 3 4 5>
<10 11 12>
<14 15>
<17 18 19>
<21 22>
<24 26>


<16>
<7 13>
<20 23>
<1 2 3 4 5>
<10 11 12>
<14 15>
<17 18 19>
<21 22>
<24 26>

+6


<16>
<3 7 13>
<20 23>
<1 2>
<4 5>
<10 11 12>
<14 15>
<17 18 19>
<21 22>
<24 26>

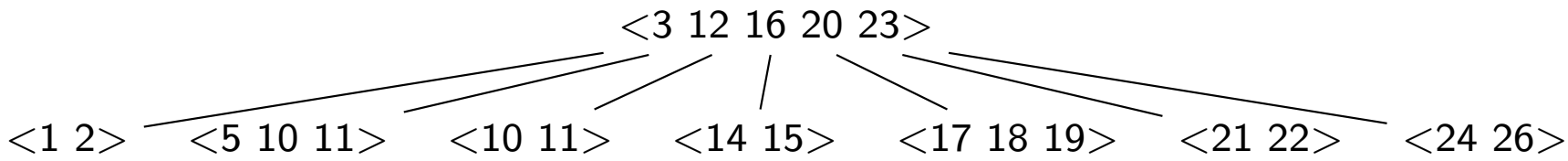

<16>
<3 7 13>
<20 23>
<1 2>
<4 5 6>
<10 11 12>
<14 15>
<17 18 19>
<21 22>
<24 26>

Remoção

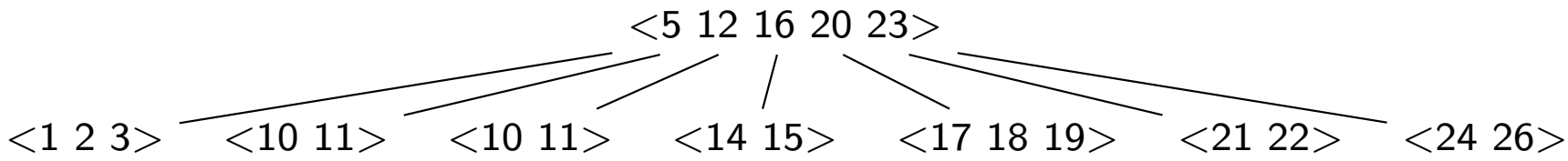
- A remoção de uma chave k avança da raiz até o nó em que a chave a ser removida está, ajustando os nós ao longo do caminho para que cada nó tenha pelo menos t chaves.
- Isso permite remover uma chave percorrendo a árvore da raiz até as folhas apenas uma vez.
- Para evitar voltar ao nó que tem a chave, se ele for interno, a chave é movida para baixo. A remoção de fato acontece sempre em uma folha.
- São 3 casos.

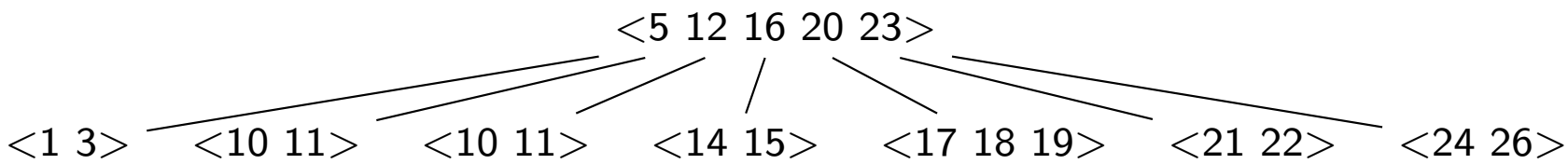
B-Tree-Remove(x, k), caso 3

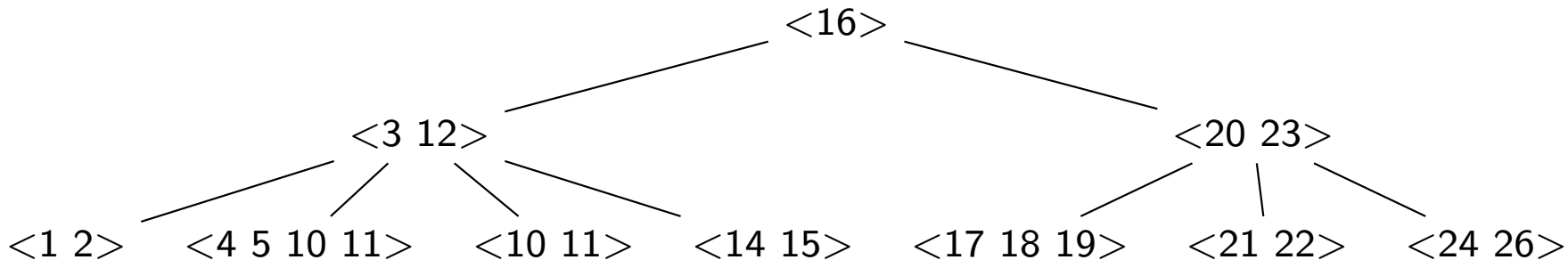
- x é nó interno e k não está em x: encontre o nó y que é filho de x e é raiz da subárvore que provavelmente contém k. Se y tem apenas $t-1$ chaves, execute os passos abaixo tantas vezes quantas forem necessárias para garantir que y tenha t chaves. Depois execute a remoção recursivamente em y.
  1. Se y tem apenas $t-1$ chaves mas tem um irmão imediato z com t chaves, adiciona uma chave extra a y movendo uma chave de x para y e movendo uma chave de z para x (e movendo os apontadores para subárvores adequadamente).
  2. Se y e seus dois irmãos imediatos têm $t-1$ chaves, faça a fusão de y com seu irmão z, movendo uma chave de x para baixo, que se torna a mediana do nó resultante da fusão.


<3 12 16 20 23>
<1 2>
<5 10 11>
<10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>

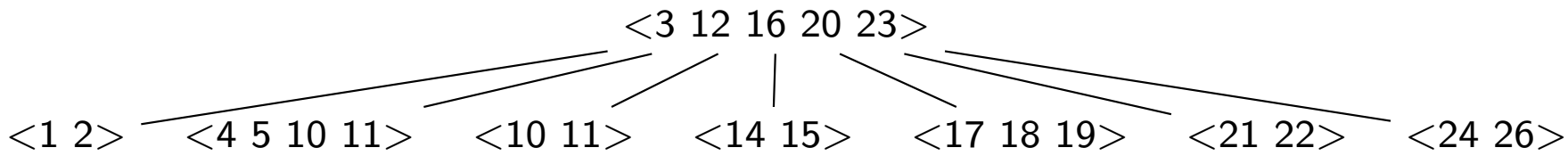
-2 (caso 3-1)

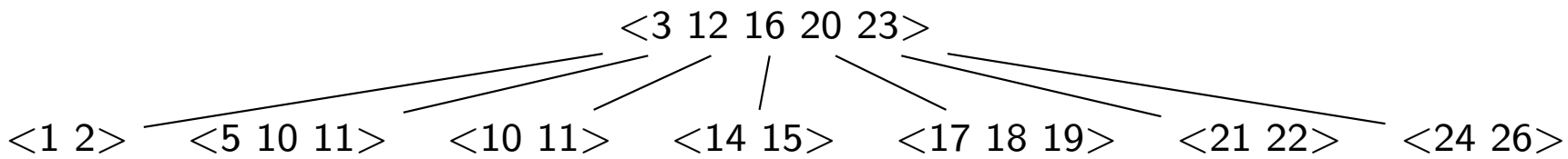

<5 12 16 20 23>
<1 2 3>
<10 11>
<10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>


<5 12 16 20 23>
<1 3>
<10 11>
<10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>


<16>
<3 12>
<20 23>
<1 2>
<4 5 10 11>
<10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>

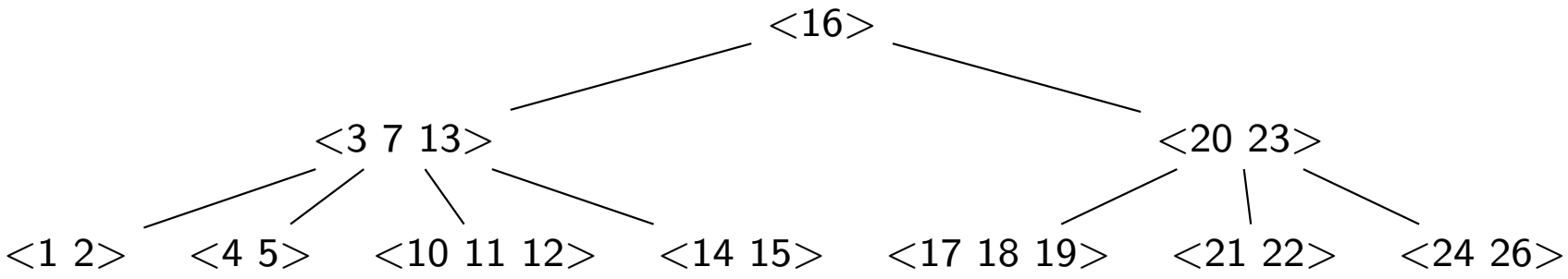
-4 (caso 3-2)


<3 12 16 20 23>
<1 2>
<4 5 10 11>
<10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>

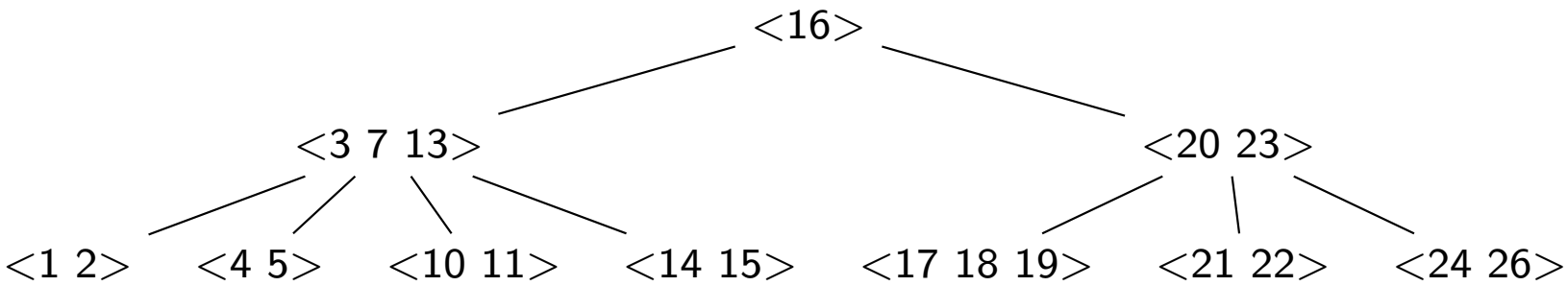

<3 12 16 20 23>
<1 2>
<5 10 11>
<10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>

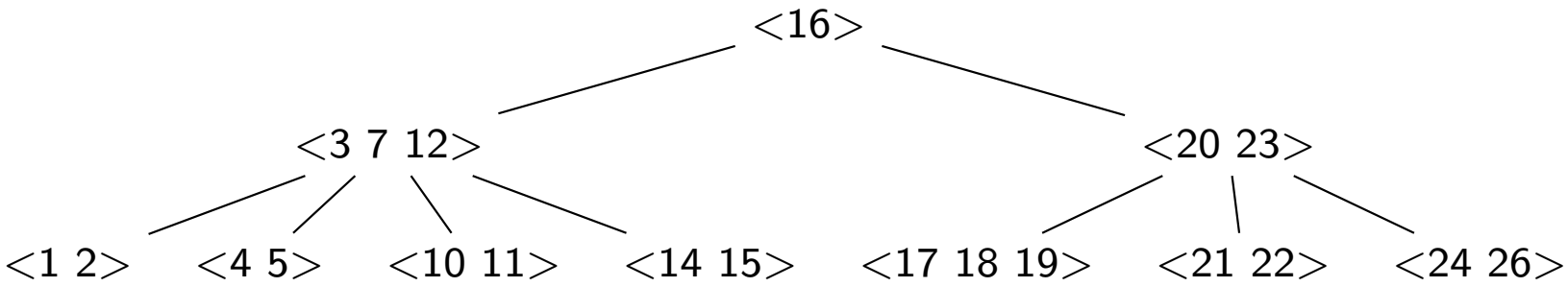
B-Tree-Remove(x, k), caso 2

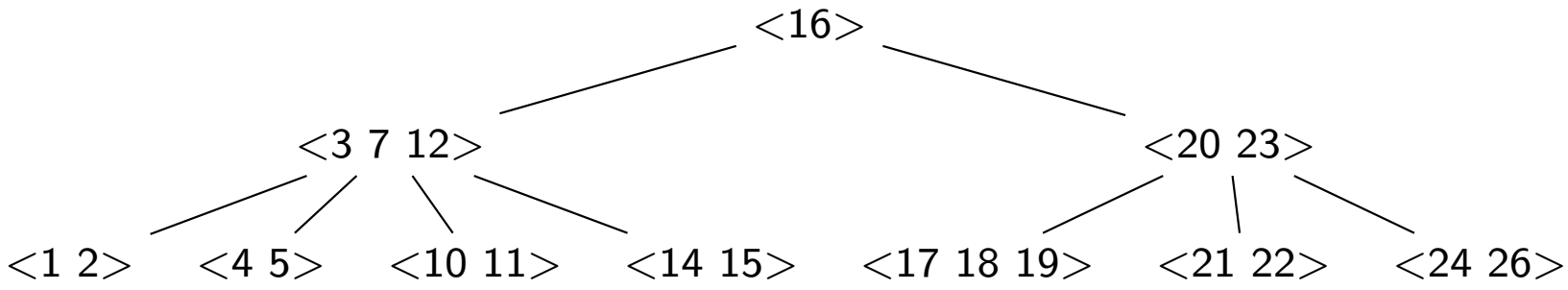
- x é um nó interno e k está em x:
  1. Seja y o filho de x que precede k. Se y tem pelo menos t chaves, encontre o predecessor de k na subárvore enraizada em y, k'. Remova k' de y recursivamente e então substitua k por k' em x.
  2. Senão, seja z o filho de x que sucede k. Se z tem pelo menos t chaves, encontre o sucessor de k na subárvore enraizada em z, k'. Remova k' de z recursivamente e substitua k por k' em x.
  3. Senão, y e z têm $t - 1$ chaves. Transfira as chaves em z e a chave k para y e remova z. Remova k de y recursivamente.


<16>
<3 7 13>
<20 23>
<1 2>
<4 5>
<10 11 12>
<14 15>
<17 18 19>
<21 22>
<24 26>

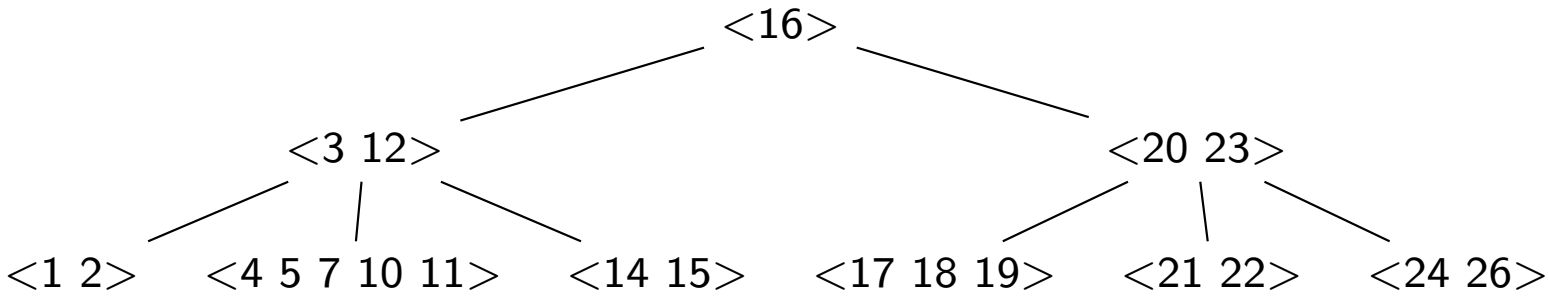
-13 (caso 2-1)

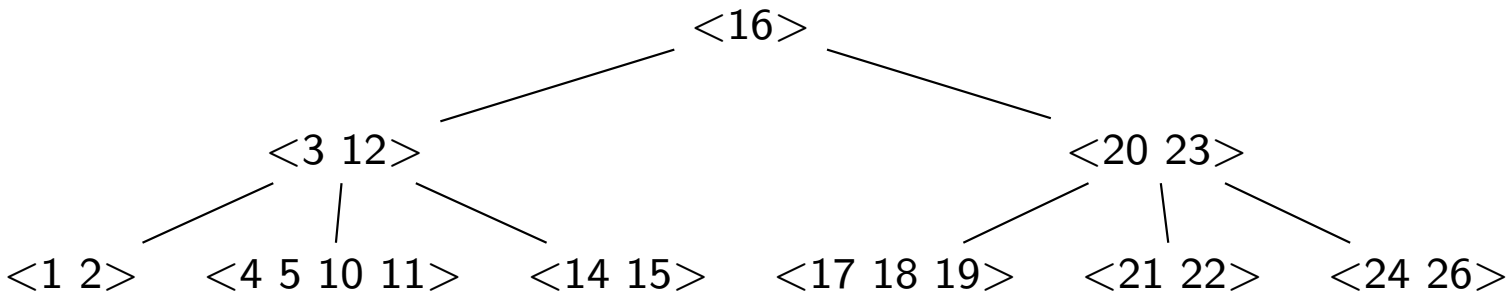

<16>
<3 7 13>
<20 23>
<1 2>
<4 5>
<10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>


<16>
<3 7 12>
<20 23>
<1 2>
<4 5>
<10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>


<16>
<3 7 12>
<20 23>
<1 2>
<4 5>
<10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>

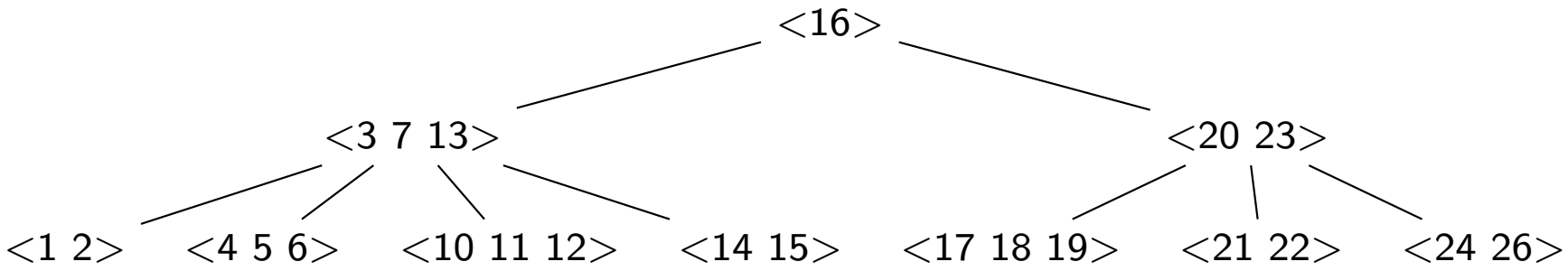
-7 (caso 2-3)


<16>
<3 12>
<20 23>
<1 2>
<4 5 7 10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>

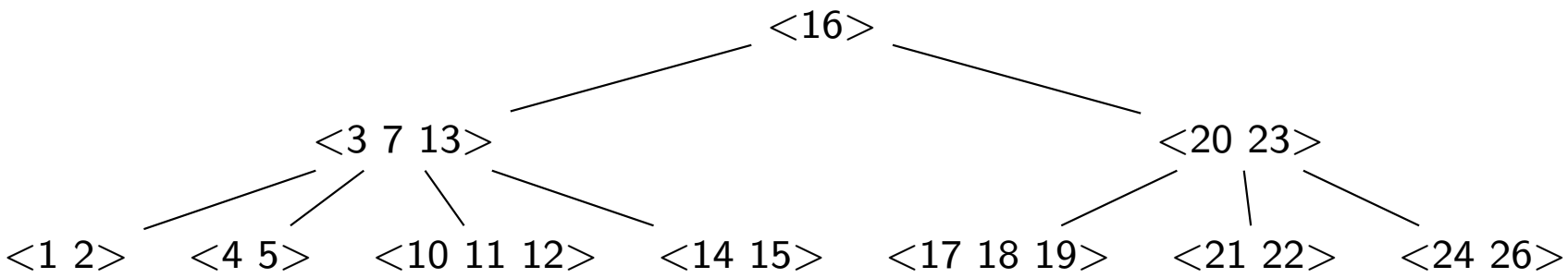

<16>
<3 12>
<20 23>
<1 2>
<4 5 10 11>
<14 15>
<17 18 19>
<21 22>
<24 26>

B-Tree-Remove(x, k), caso 1

- x é folha e k está em x: remova k de x.


<16>
<3 7 13>
<20 23>
<1 2>
<4 5 6>
<10 11 12>
<14 15>
<17 18 19>
<21 22>
<24 26>

-6 (caso 1)


<16>
<3 7 13>
<20 23>
<1 2>
<4 5>
<10 11 12>
<14 15>
<17 18 19>
<21 22>
<24 26>

Árvore B+

- A árvore B+ é uma variação da árvore B em que os registros ficam todos nas folhas.
- Assim os nós internos podem armazenar mais apontadores e intervalos e a árvore fica mais compacta.
- As folhas podem ser encadeadas, o que permite recuperar todos os registros em ordem.