CI1057: Algoritmos e Estruturas de Dados III
Árvores 2-3-4

Profa. Carmem Hara

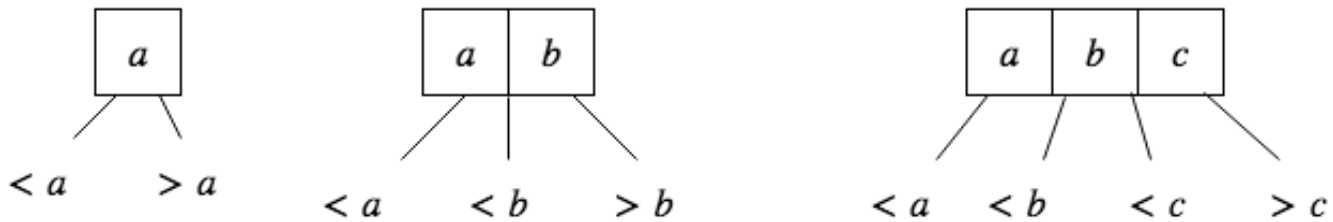
Departamento de Informática/UFPR

4 de abril de 2024

Árvore 2-3-4

É uma árvore que está vazia ou que é composta por 3 tipos nodos:

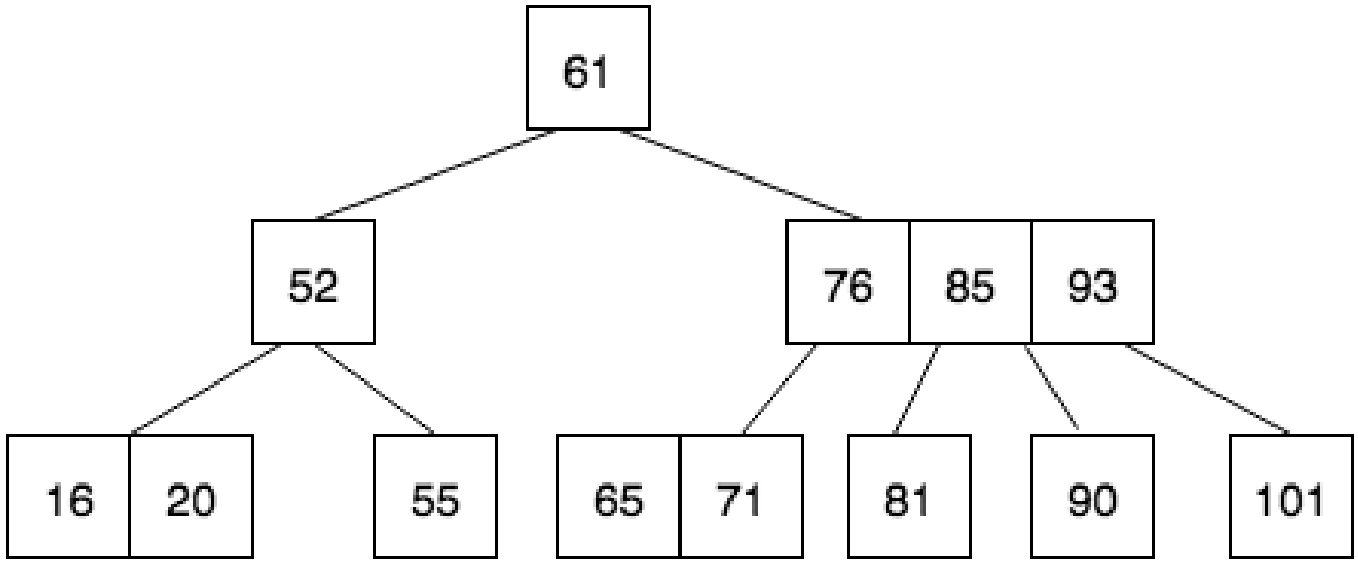
- **nodo-2:** contém uma chave a e dois apontadores, p_1 e p_2
- **nodo-3:** contém duas chaves a e b e três apontadores, p_1, p_2 e p_3
- **nodo-4:** contém três chaves a, b, c e quatro apontadores p_1, p_2, p_3 e p_4



(a) 2-node (b) 3-node (c) 4-node

Árvore 2-3-4 Balanceada

Árvore 2-3-4 na qual todos os nodos folha estão no mesmo nível.



Árvore 2-3-4 - Estrutura de dados

Cada nodo da árvore contém:

- no máximo 3 chaves
- no máximo 4 filhos
- quantidade de chaves armazenadas

```
1  typedef int ItemArv;
2
3  typedef struct nodo *ApNodo;
4  typedef struct nodo {
5    ItemArv item[3];
6    ApNodo ap[4];
7    int numItem;
8    ApNodo pai;
9  } Nodo;
10
11 typedef ApNodo Arv234;
```

Árvore 2-3-4 - Interface

```
1 void criaArv234( Arv234* );
2 void freeArv234( Arv234 );
3 void escreveArv234( Arv234 );
4 Arv234 insereArv234( ItemArv, Arv234 );
5 Arv234 buscaArv234( ItemArv, Arv234 );
6 Arv234 removeArv234( ItemArv, Arv234 );
```

Busca

```
1  Arv234 buscaArv234( ItemArv v, Arv234 p ){
2    int i;
3
4    if( p == NULL )
5      return NULL;
6    i= 0;
7    while( i < p->numItem && lt( p->item[i], v ))
8      i++;
9    if( eq( v, p->item[i] ) )
10     return p;
11   return buscaArv234( v, p->ap[i] );
12 }
```

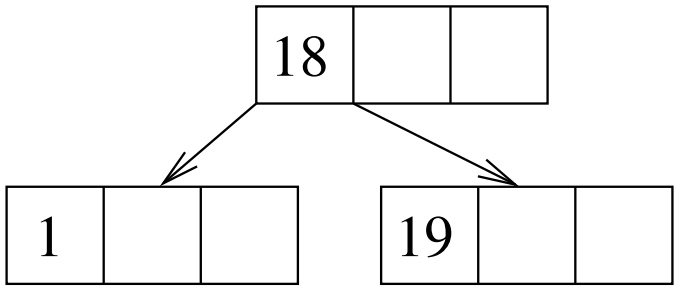
Propriedade:

a busca de uma árv. 2-3-4 com n nodos visita no máximo $lg(n) + 1$ nodos.

Inserção

A inserção de um novo valor k pode ser feita da mesma forma que em uma árvore binária. Após uma busca sem sucesso, inserir a chave na folha.

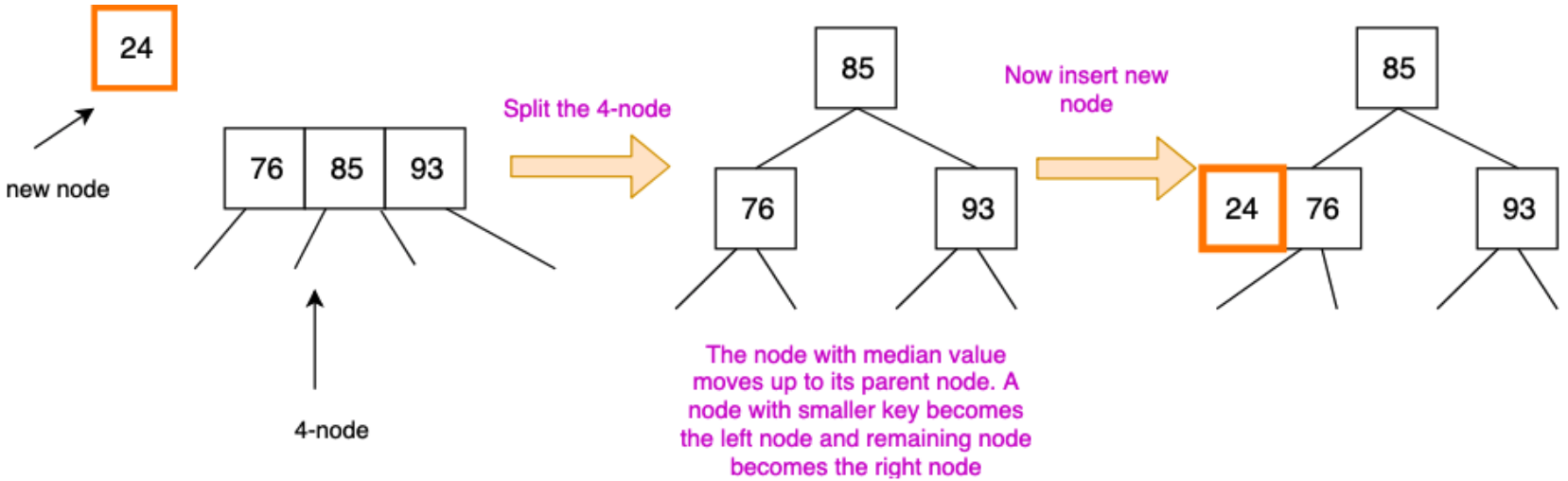
Problema: a árvore pode ficar desbalanceada



Inserir as chaves: 3 8 9

Balanceamento na inserção

Baseado no processo de dividir o nodo para inserir uma nova chave em um nodo-4 (*split*).

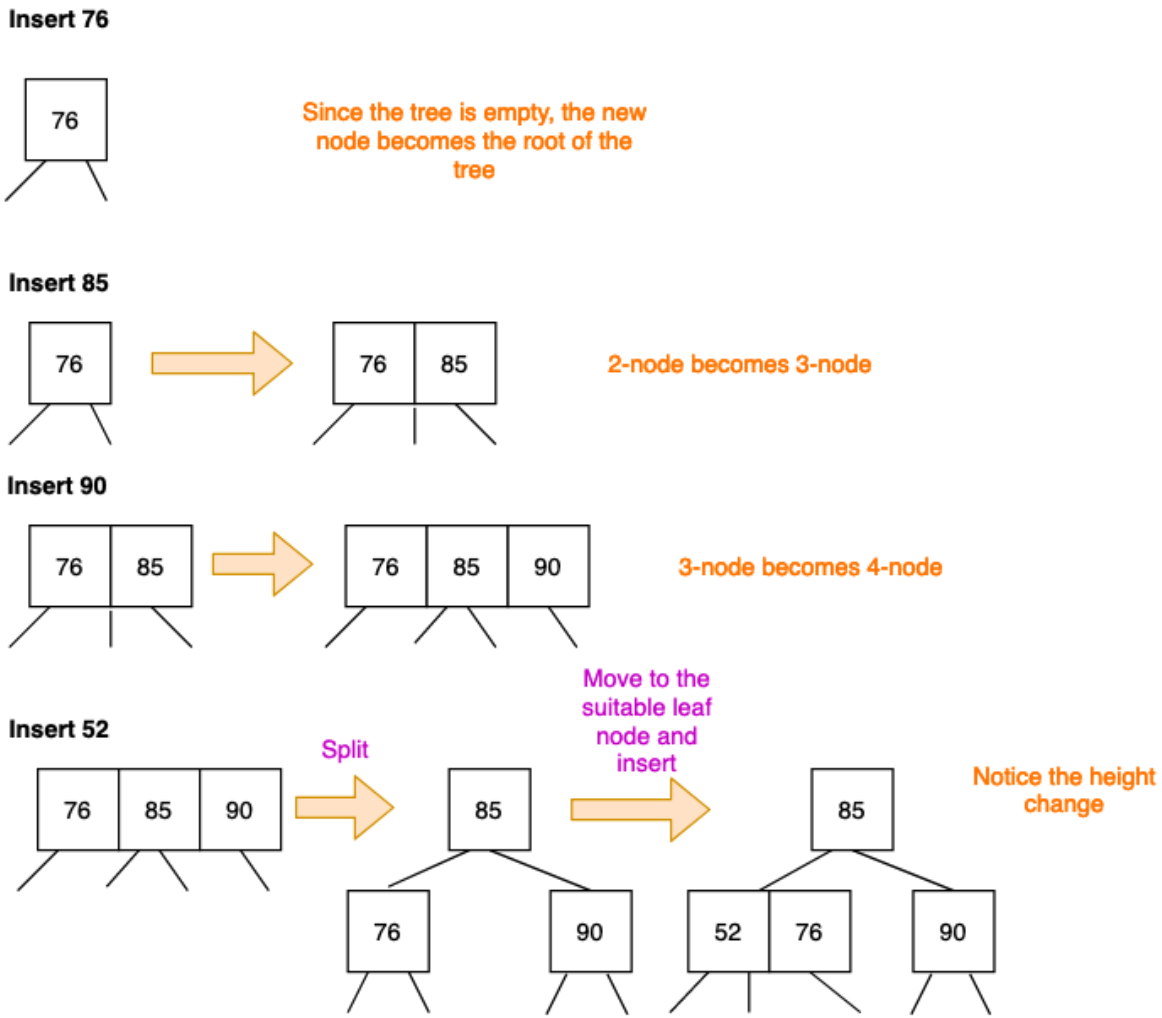


Inserção - Duas Abordagens

- *Bottom-up:*
  Inserir sempre em um nodo folha. Caso o nodo contenha mais de 3 chaves, divide o nodo e "sobe"com a chave do meio
- *Top-down:*
  Dividir os nodos com 3 chaves no caminho da raiz até o nodo folha

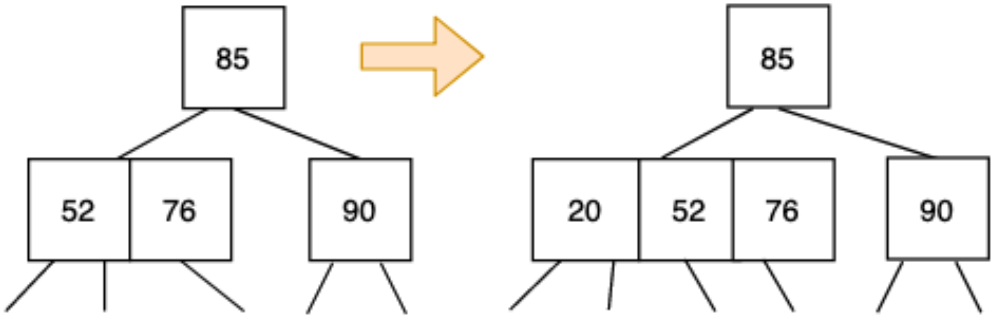
Exemplo:
Construir a árvore 2-3-4 com as chaves
76 85 90 52 20 33 25 70

Exemplo

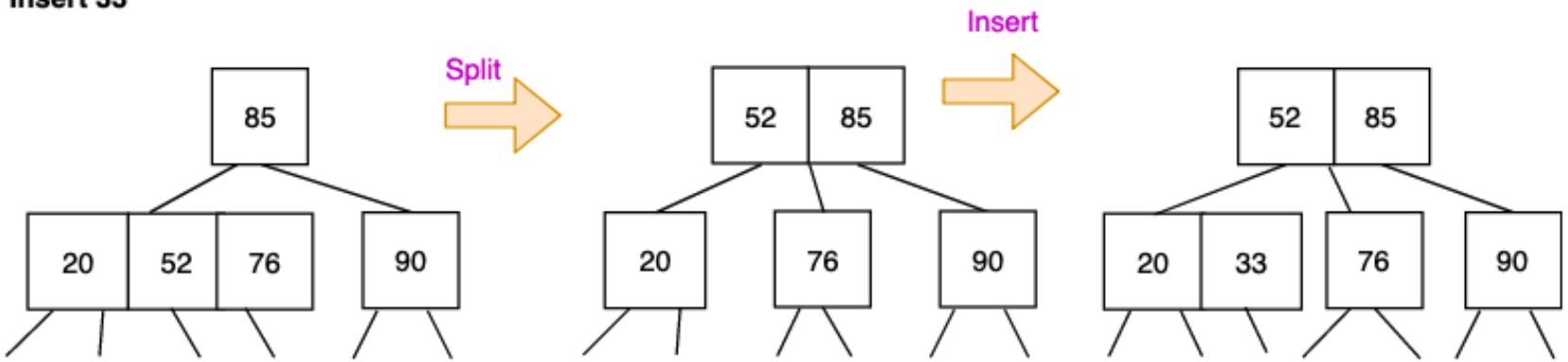

Insert 76
76
Since the tree is empty, the new node becomes the root of the tree
Insert 85
76
76
85
2-node becomes 3-node
Insert 90
76
85
76
85
90
3-node becomes 4-node
Insert 52
76
85
90
Split
85
76
90
Move to the suitable leaf node and insert
85
52
76
90
Notice the height change

Exemplo

Insert 20



Insert 33



Inserção na árvore - Implementação

```
1  Arv234 insereArv234( ItemArv v, Arv234 raiz ){
2    Arv234 p, pai, esq, dir, novoDir, novoEsq;
3    int i, split;
4    ItemArv splitV, novoSplitV;
5
6    if( raiz == NULL ){
7      p= criaNodo();
8      insereItem( v, NULL, NULL, p );
9      return p;
10   }
11   p= buscaFolha( v, raiz );
12   if( p == NULL ) /* v ja existe */
13     return raiz;
14   split= insereNoNodo( v, NULL, NULL, p, &splitV, &esq, &dir );
15   p= p->pai;
16   while( split && p != NULL ){
17     split= insereNoNodo( splitV, esq, dir, p, &novoSplitV, &novoEsq, &novoDir );
18     splitV= novoSplitV;
19     esq= novoEsq; dir= novoDir;
20     p= p->pai;
21   }
22   if( split ){  /* split da raiz */
23     p= criaNodo();
24     insereItem( splitV, esq, dir, p );
25     return p;
26   }
27   return raiz;
28 }
```

Inserção no nodo - Implementação

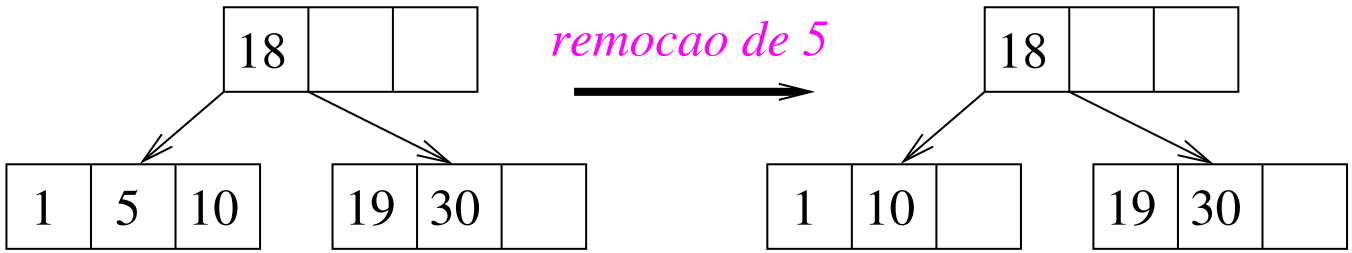
```
1  int insereNoNodo( ItemArv v, Arv234 esq, Arv234 dir, Arv234 p,
2        ItemArv *splitV, Arv234 *novoEsq, Arv234 *novoDir){
3    if( p->numItem < 3 ){
4      insereItem( v, esq, dir, p );
5      return 0;
6    }
7    *splitV= p->item[1];
8    *novoDir= criaNodo();
9    insereItem( p->item[2], p->ap[2], p->ap[3], *novoDir );
10   *novoEsq= p;
11   p->numItem= 1;
12   if( lt( v, *splitV ) )
13     insereItem( v, esq, dir, *novoEsq );
14   else
15     insereItem( v, esq, dir, *novoDir );
16   return 1;
17 }
```

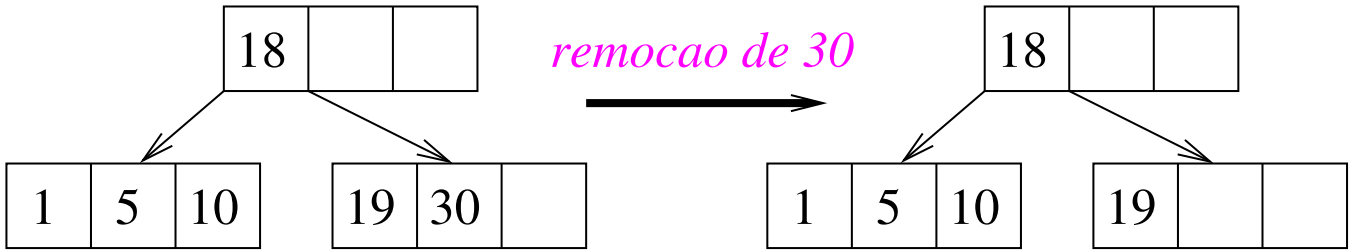
Inserção Item em Nodo já existente - Implementação

```
1  int insereItem( ItemArv v, Arv234 esq, Arv234 dir, Arv234 p ){
2    int i;
3
4    i=p->numItem;
5    while( i>0 && lt( v, p->item[i-1]) ){
6      p->item[i]= p->item[i-1];
7      p->ap[i+1]= p->ap[i];
8      i--;
9    }
10   p->item[i]= v;
11   p->ap[i]= esq;
12   p->ap[i+1]= dir;
13   p->numItem++;
14   if( esq != NULL ){
15     esq->pai= p;
16     dir->pai= p;
17   }
18   return i;
19 }
```

Remoção - Casos Simples

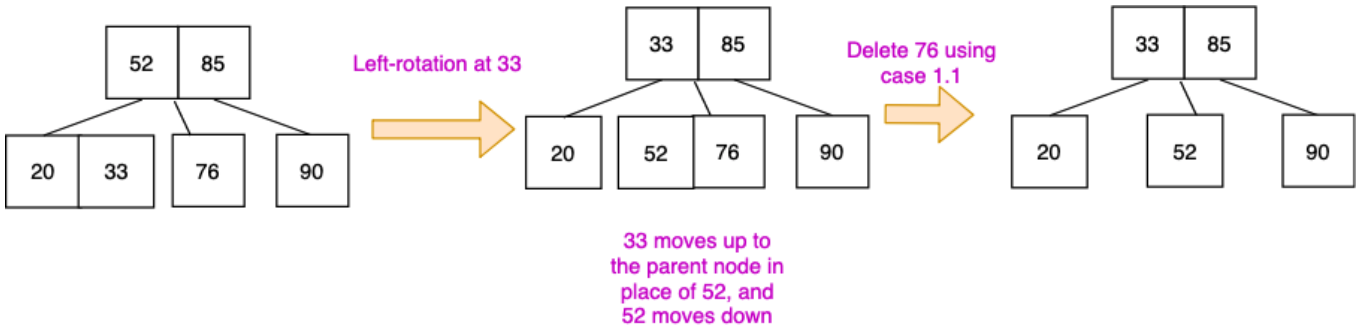
- Remoção na folha de nodo-3 ou nodo-4





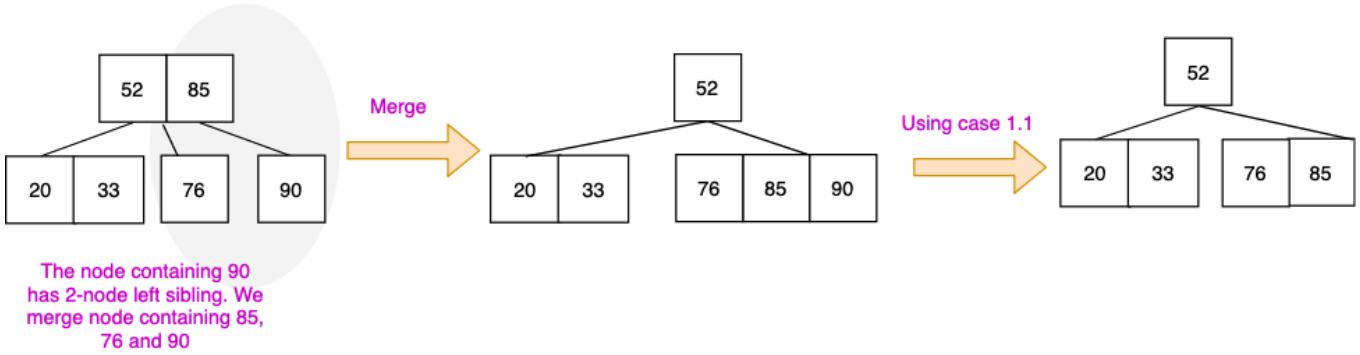
Remoção- Nodo-2

- **Caso 1:** Com irmão do tipo nodo-3 ou nodo-4 "empréstimo" de chave do irmão e rotação



Remoção- Nodo-2

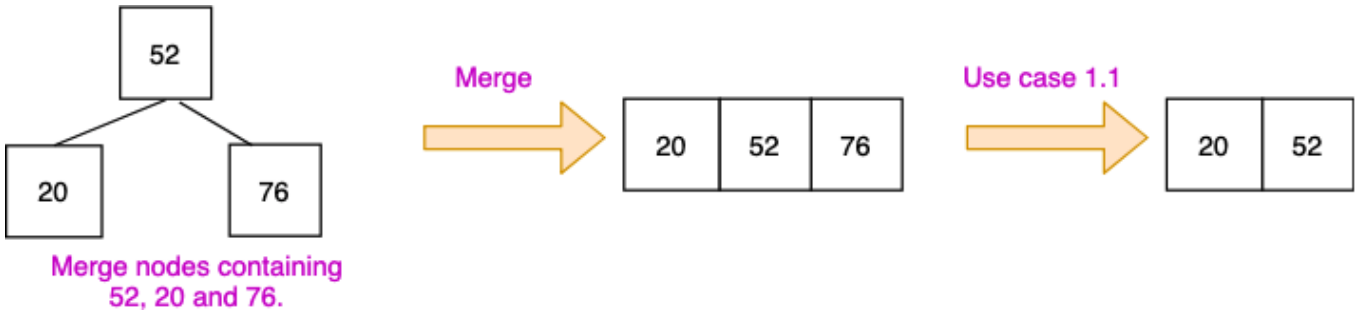
- **Caso 2:** Ambos os irmãos são nodo-2
  "merge" dos irmãos com o pai



Aplicar os Casos 1 e 2 em **todos** os nodos-2 no caminho da raiz até a folha com a chave a ser removida (exceto a raiz).

Remoção- Nodo-2

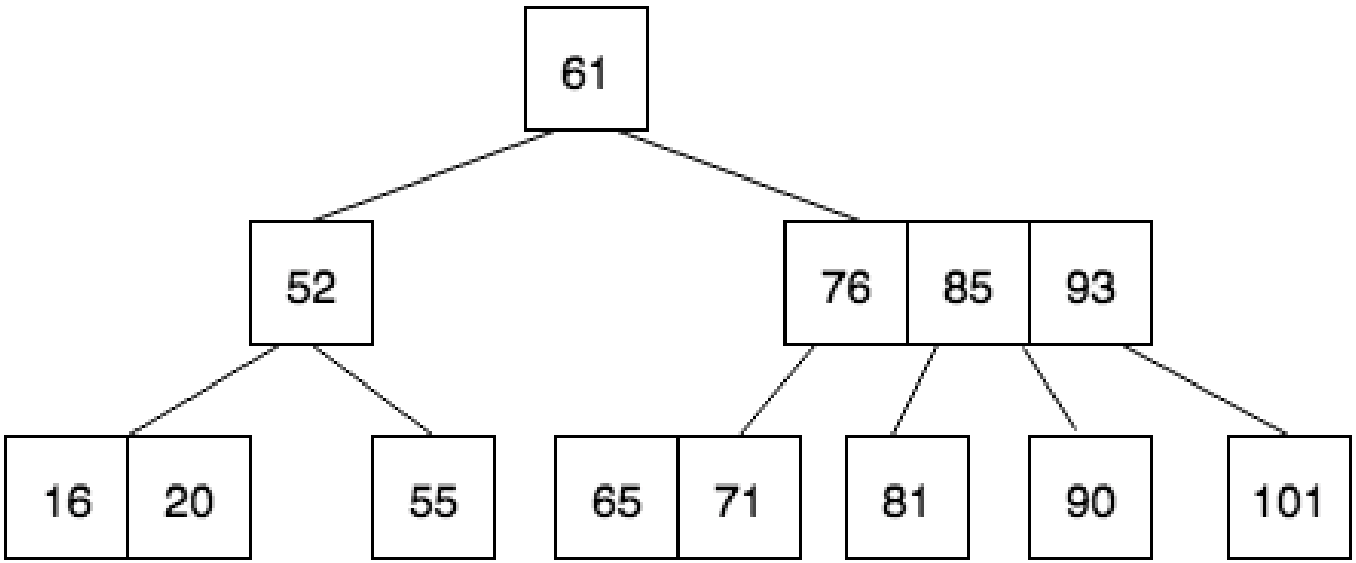
- **Caso 3:** Ambos os irmãos e pai são nodo-2
  "merge" dos irmãos com o pai, com redução da altura
  pai é a raiz



Remoção- Chave em nodo não-folha

1. busque o predecessor (*predVal*) da chave a ser removida
2. troque o valor de *predVal* pela chave a ser removida (coloca a chave na folha)
3. remova a chave

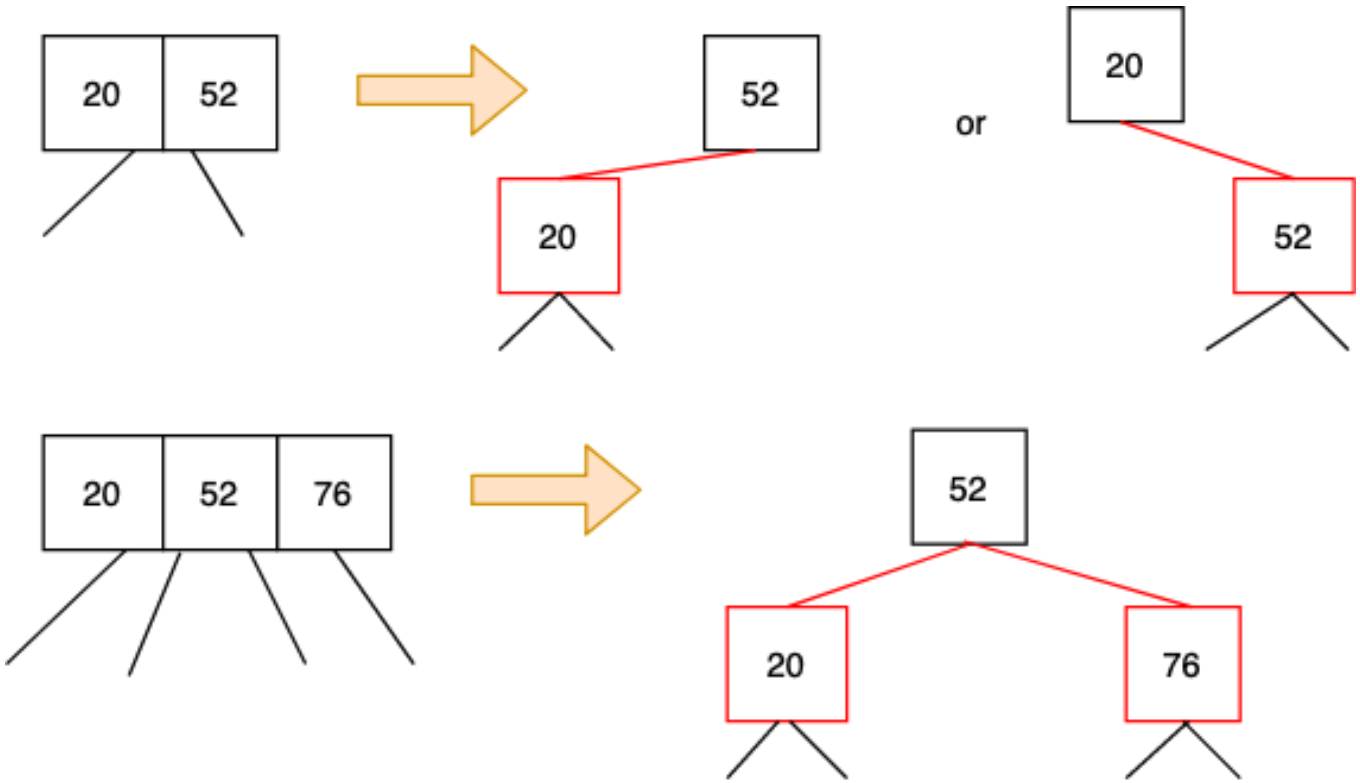
Exemplo de remoção



Remoção das chaves:
71 20 101 90 61

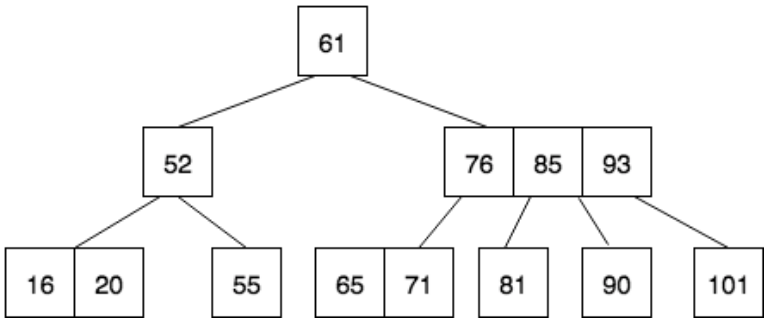
Árvore Rubro-Negra

É uma representação da árvore 2-3-4 em uma árvore binária.

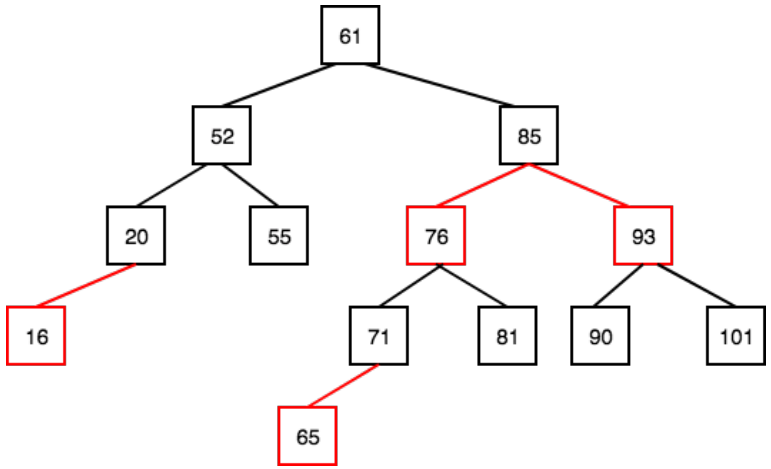


Árvore Rubro-Negra

Árvore 2-3-4



Árvore Rubro-Negra



Referências

- Livro Sedgewick, Seção 13.3
- `https://algorithmtutor.com/Data-Structures/Tree/2-3-4-Trees/`