CI1057: Algoritmos e Estruturas de Dados III
Árvore B

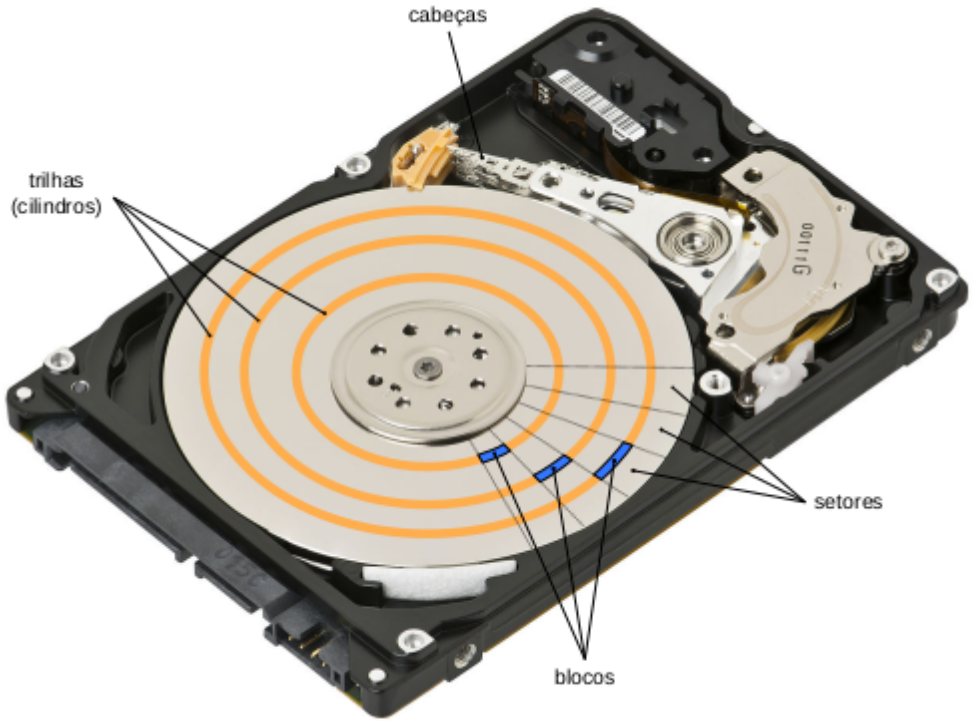
Profa. Carmem Hara

Departamento de Informática/UFPR

11 de abril de 2024

Árvore B

- Motivação: indexação de dados em memória secundária (discos)
- Acessar um registro em disco é algumas ordens de grandeza maior que o processamento na memória primária



Medida de Complexidade

- Quantidade de transferências entre a memória principal e secundária
  - disco rígido: 4.200 RPM a 15.000 RPM
    - tempo de busca (posicionamento do cabeçote): +- 10ms
    - latência rotacional: +- 5ms
    - este tempo nao é constante porque depende da localização do dado na trilha e o posicionamento do cabecote
  - RAM: 50 a 80 nanosegundos por acesso: 5 vezes mais rápido

Amortização do Custo de Acesso

- Transferência em blocos/páginas ao invés de itens individuais
  - blocos físicos (interseção de um trilha com um setor)
    - antigamente: 512 bytes (2^9)
    - recentemente: 4096 bytes (2^12))
  - capacidade de memória secundária e' em geral bem maior que a capacidade da memória principal
  - **Memória virtual:** permite que um programa utilize um espaço de endereçamento maior que o espaço de memória primária disponível

Memória Virtual

- **Espaço de endereçamento (N):** endereços usados pelo programa
- **Espaço de Memória (M):** endereços da memória principal no computador
- **Memória Virtual:** implementa uma função de mapeamento entre o espaço de endereçamento e o espaco de memória:

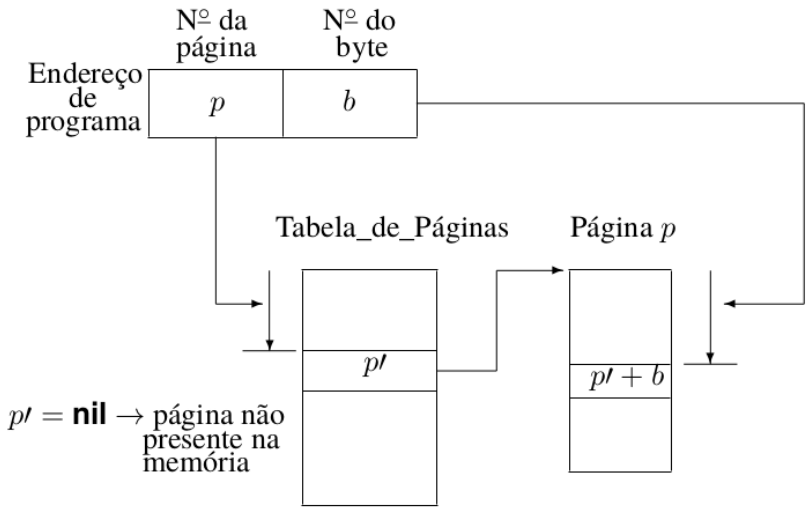
$$f : N \rightarrow M$$

Sistema de Paginação

- O espaço de endereçamento é dividido em páginas de tamanho igual (tamanho da página)
- O espaço de memória é dividida em **Molduras de páginas** de mesmo tamanho
- As molduras contém as páginas ativas. As demais ficam na memória secundária
- Mecanismo de paginação tem 2 funções:
  1. *Mapeamento de endereços:*
     determina qual página o programa está endereçando e encontra a moldura, se existir, que contém a página
  2. *Transferência de páginas:*
     transfere páginas da memória secundária para a primária e vice-versa

Mapeamento de Endereços

- Endereçamento da página: parte dos bits compõem o número da página e a outra parte o número do byte dentro da paǵina (*offset*)
  Ex: considerando um espaço de enderecamento de 24 bits:
  - a memória virtual tem tamanho 2^{24}
  - se o tamanho da pagina e' 512 (2^9) então:
    15 bit para o número da página (p)
    9 bits para o *offset* (b)



Transferência de Páginas

- Responsável por carregar uma página na moldura de páginas
- O mapeamento entre o número de pagina (*Logical Block Addressing - LBA*) e o endereçamento físico (*Cylinder-Head-Sector - CHS*) é feito de forma transparente - firmware do disco rígido
- Se não houver moldura vazia, uma página deve ser removida (e gravada se tiver sido alterada)
- Adota uma política de reposição de páginas:
  - LRU: Least Recently Used
  - LFU: Least Frequently Used
  - FIFO: First In First Out

Árvore B

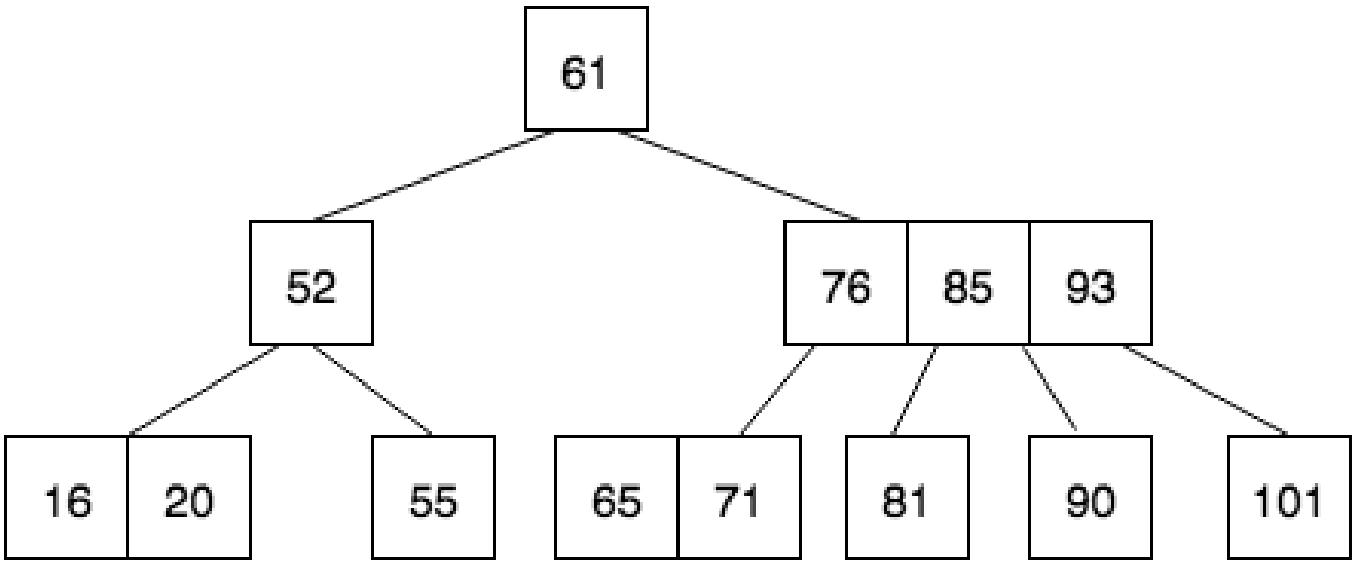
- Idéia: o tamanho do nodo da árvore coincide com o tamanho da página do sistema
- Quantidade de chaves em cada nodo: tipicamente entre 50 a 2000
- Transferência entre memória principal e secundária é feita pelo sistema operacional
- generalização da Árvore 2-3-4 = Árvore B de ordem 2

Árvore B

Uma árvore B de ordem t ($t \geq 2$) é uma árvore n-ária na qual:

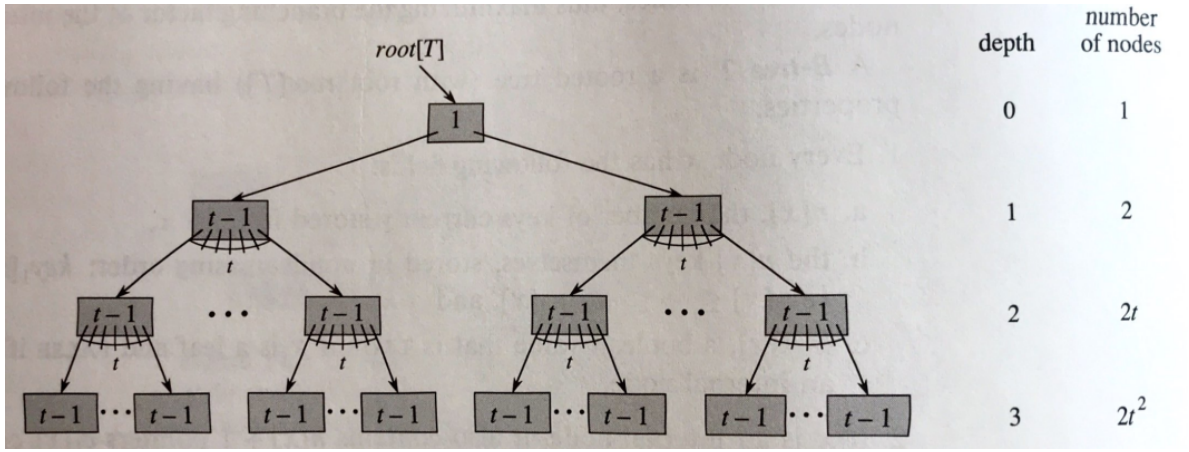
1. os nodos internos (páginas), exceto a raiz, contém no minimo $t-1$ registros (e t descendentes) e no máximo $2t-1$ registros ($2t$ descendentes)
2. a raiz pode conter entre 1 e $2t-1$ registros
3. todos os nodos folha estão no mesmo nivel.

Árvore B de ordem 2



Altura de uma Árvore B

A altura máxima de uma Árvore B de grau t com n registros é $log_t((n+1)/2)$.



$$n \geq 1 + (t-1)\sum_{i=1}^{h-1}(2t^{i-1}) = 1 + 2(t-1)((t^h-1)/(t-1)) = 2t^h - 1$$

Ou seja, $h \leq log_t((n+1)/2)$.

Quantidade de Chaves com Árvore de Altura 3

t	Número Mínimo de Chaves	
10	$2 * 10^3 - 1$	2.000
100	$2 * 100^3 - 1$	2.000.000
1000	$2 * 1000^3 - 1$	2.000.000.000

Árvore B - Estrutura de dados

```
1  #define T 3
2  #define MIN T-1
3  #define MAX 2*T-1
4  #define MEIO T-1
5
6  typedef int ItemArv;
7
8  typedef struct nodo *ApNodo;
9  typedef struct nodo {
10   ItemArv item[MAX];
11   ApNodo ap[MAX+1];
12   int numItem;
13   ApNodo pai;
14 } Nodo;
15 typedef ApNodo ArvB;
16
17 typedef struct posicao{
18   ApNodo ap;
19   int ind;
20 } Posicao;
```

Árvore B - Interface

```
1 void criaArvB( ArvB* );
2 void freeArvB( ArvB );
3 void escreveArvB( ArvB );
4 ArvB insereArvB( ItemArv, ArvB );
5 void buscaArvB( ItemArv, ArvB, Posicao* );
6 void removeArvB( ItemArv, ArvB* );
```

Busca

```
1  void buscaArvB( ItemArv v, ArvB p, Posicao *res ){
2    int i;
3
4    if( p == NULL ){
5      res->ap= NULL;
6      return;
7    }
8    i= 0;
9    while( i < p->numItem && lt( p->item[i], v ))
10     i++;
11   if( eq( v, p->item[i] ) ){
12     res->ap= p;
13     res->ind= i;
14     return;
15   }
16   /* readFile( p->ap[i] ) */
17   return buscaArvB( v, p->ap[i], res );
18 }
```

Propriedade:

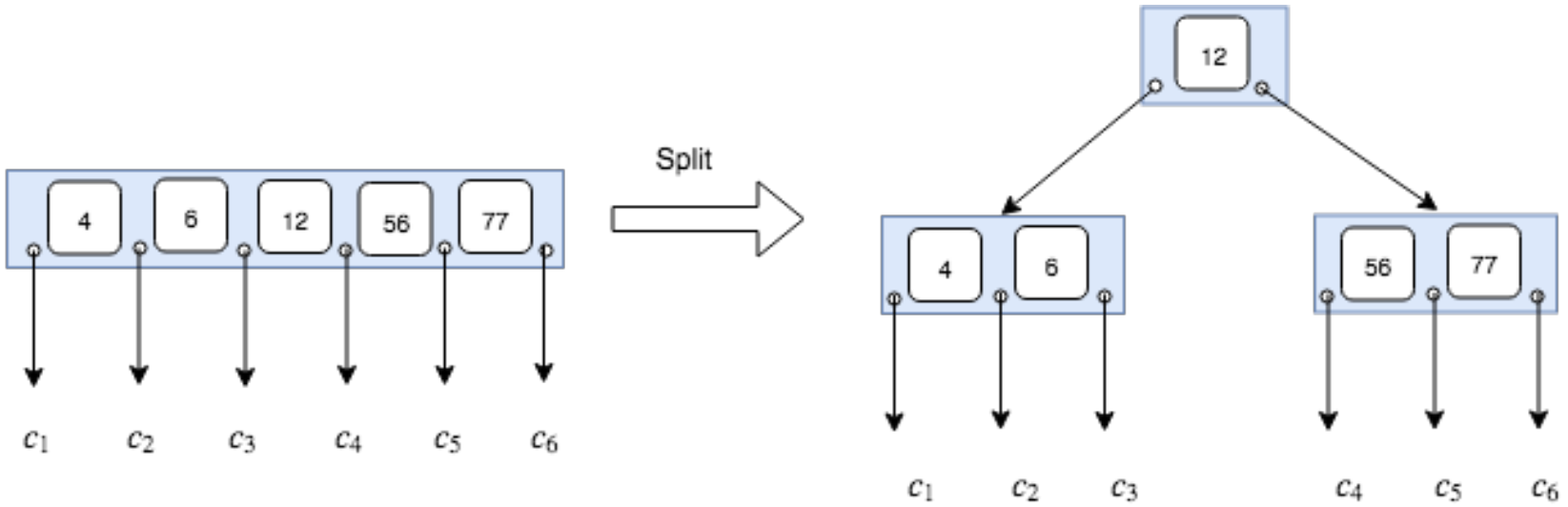
a busca de uma árv. B com n nodos visita no máximo $log_t(n) + 1$ nodos.

Inserção

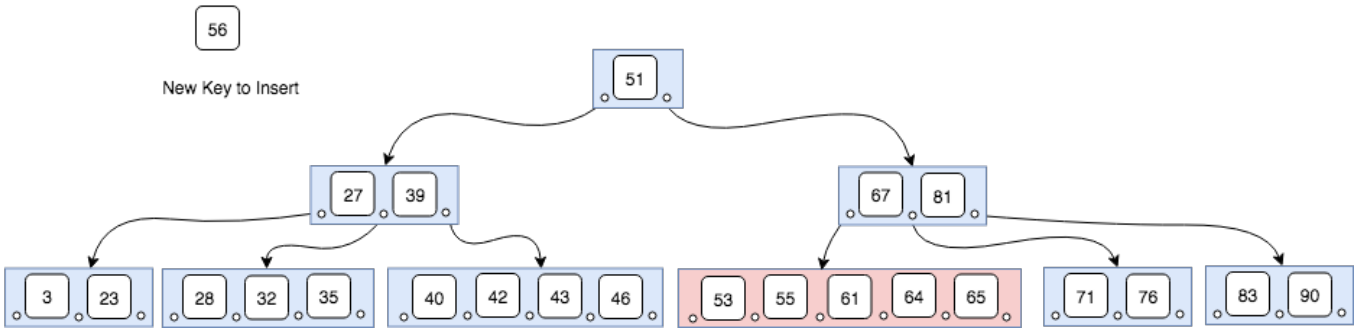
Idêntica à inserção na Árvore 2-3-4.

- Inserção sempre na folha
- Balanceamento com duas abordagens:
  - Top-down: *split* dos nodos cheios no caminho da raiz até a folha
  - Bottom-up: *split* da folha quando necessário, com propagação do valor do meio para o pai

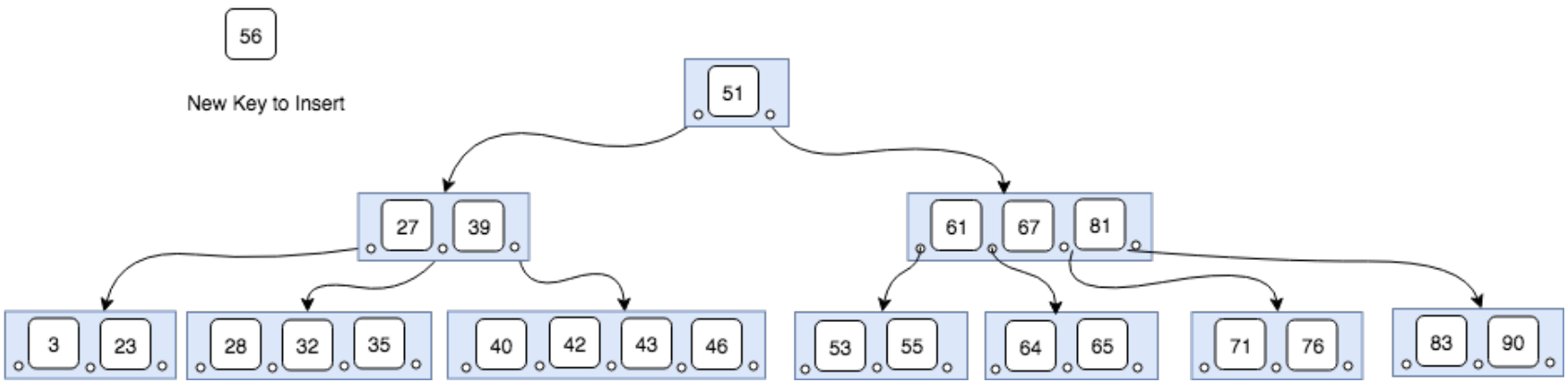
Split em Árvore B de ordem 3



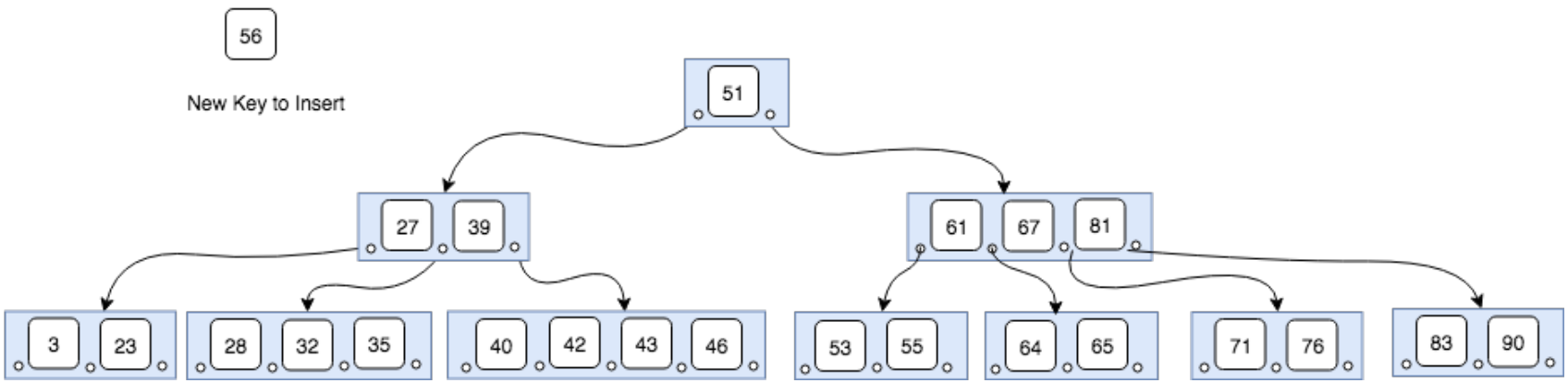
Exemplo - Inserção de 56 na Árvore B de ordem 3



Depois do Split



Árvore Resultante Após a Inserção



Inserção na árvore - Implementação

```
1  ArvB insereArvB( ItemArv v, ArvB raiz ){
2    ArvB p, pai, esq, dir, novoDir, novoEsq;
3    Posicao pos;
4    int i, split;
5    ItemArv splitV, novoSplitV;
6
7    if( raiz == NULL ){
8      p= criaNodo();
9      insereItem( v, NULL, NULL, p );
10     return p;
11   }
12   p= buscaFolha( v, raiz );
13   if( p == NULL ) /* v ja existe */
14     return raiz;
15   split= insereNoNodo( v, NULL, NULL, p, &splitV, &esq, &dir );
16   p= p->pai;
17   while( split && p != NULL ){
18     split= insereNoNodo( splitV, esq, dir, p, &novoSplitV, &novoEsq, &novoDir );
19     splitV= novoSplitV;  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

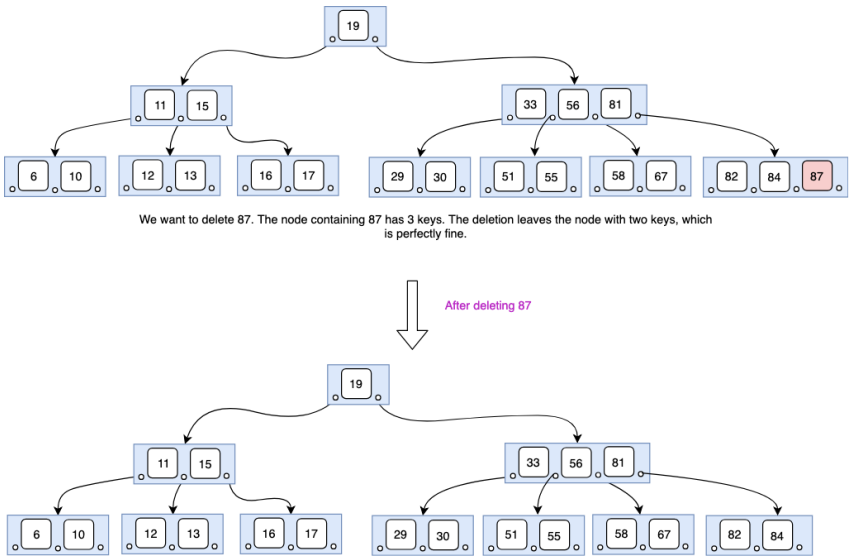
```
1  int insereNoNodo( ItemArv v, ArvB esq, ArvB dir, ArvB p,
2        ItemArv *splitV, ArvB *novoEsq, ArvB *novoDir){
3    int i;
4
5    if( p->numItem < MAX ){
6      insereItem( v, esq, dir, p );
7      return 0;
8    }
9    *splitV= p->item[MEIO];
10   *novoDir= criaNodo();
11   for( i=MEIO+1; i<MAX; i++ )
12     insereItem( p->item[i], p->ap[i], p->ap[i+1], *novoDir );
13   *novoEsq= p;
14   p->numItem= MIN;
15   /* writeFile( novoDir ); writeFile( novoEsq ); */
16   if( lt( v, *splitV ) )
17     insereItem( v, esq, dir, *novoEsq );
18   else
19     insereItem( v, esq, dir, *novoDir );
20   return 1;
21 }
```

Inserção Item em Nodo já existente - Implementação

```
1  int insereItem( ItemArv v, ArvB esq, ArvB dir, ArvB 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     /* writeFile( esq ); writeFile( dir ); */
18   }
19   /* writeFile( p ); */
20   return i;
21 }
```

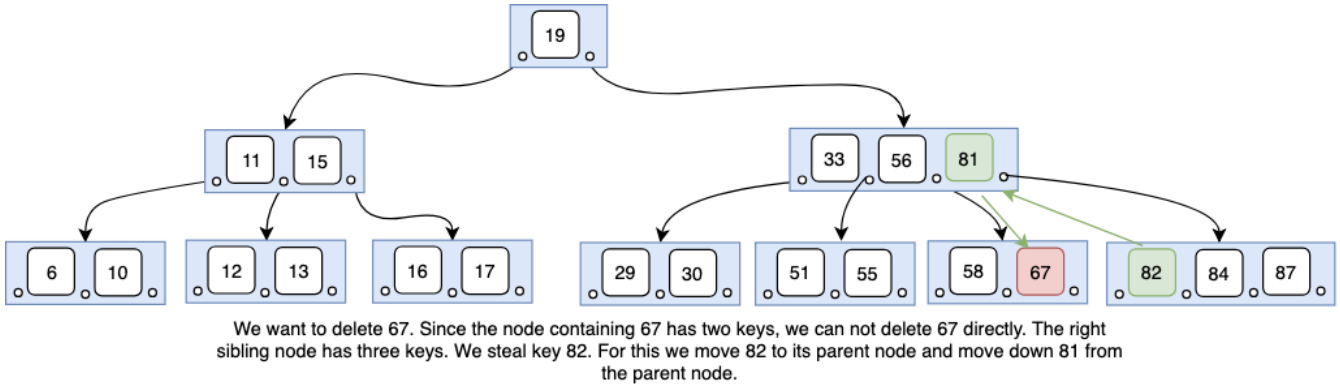
Remoção

1. Busque o nodo N que contém o valor V a ser removido
2. Se N não é um nodo folha, remover o sucessor de V ($remV$), localizado em uma folha ($nodoRem$) e depois substituir V por $remV$

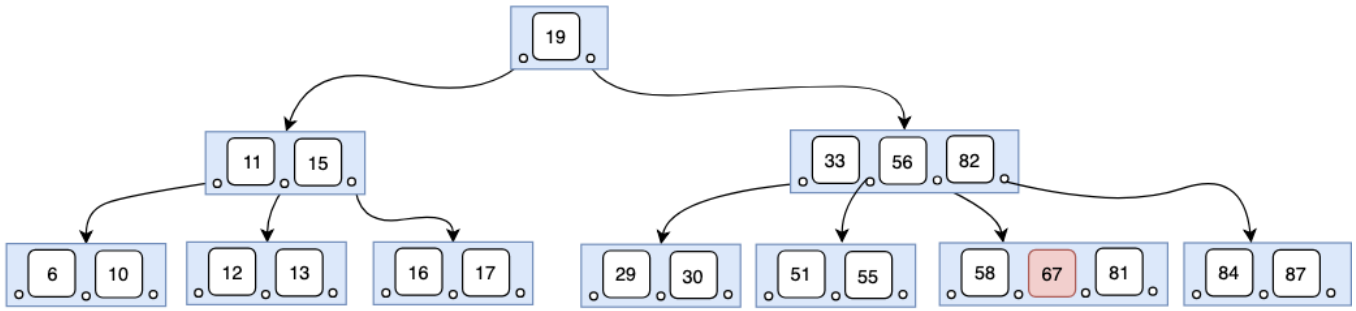


Remoção - "empréstimo do irmão"

4. Se o nodo contiver menos que $t - 1$ valores, "empresta" um valor do irmão



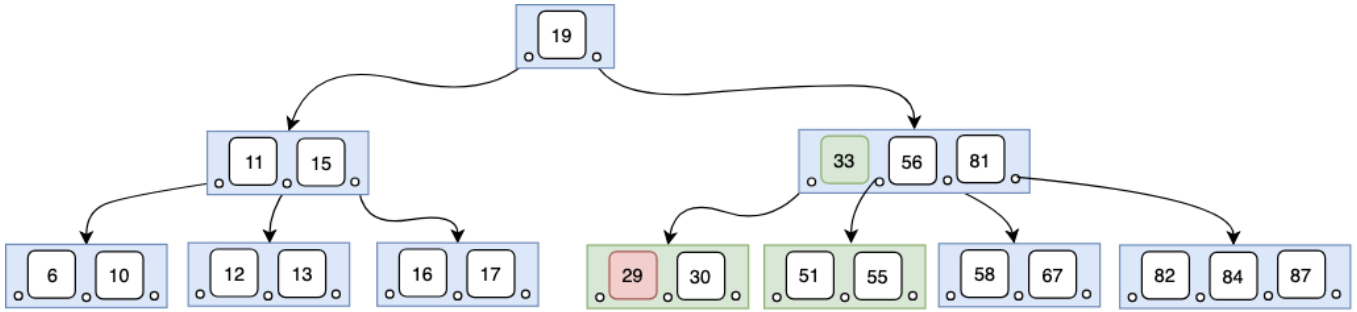
We want to delete 67. Since the node containing 67 has two keys, we can not delete 67 directly. The right sibling node has three keys. We steal key 82. For this we move 82 to its parent node and move down 81 from the parent node.



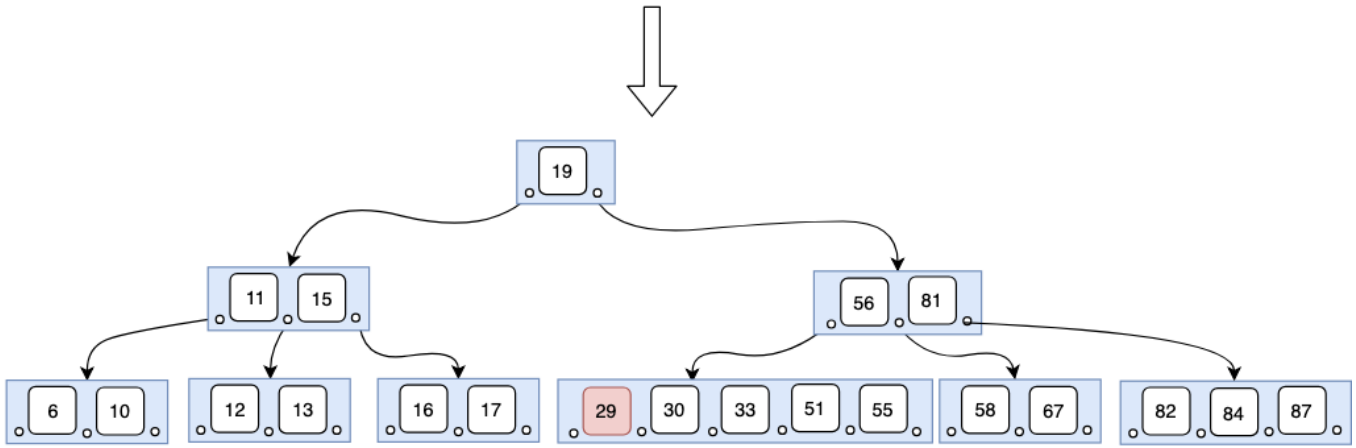
Now the node containing 67 has three keys. We can delete 67 using the step 2a.

Remoção- "merge com o pai"

5. Se os irmãos tem somente $t-1$ valores, faz o "merge" com um valor do pai

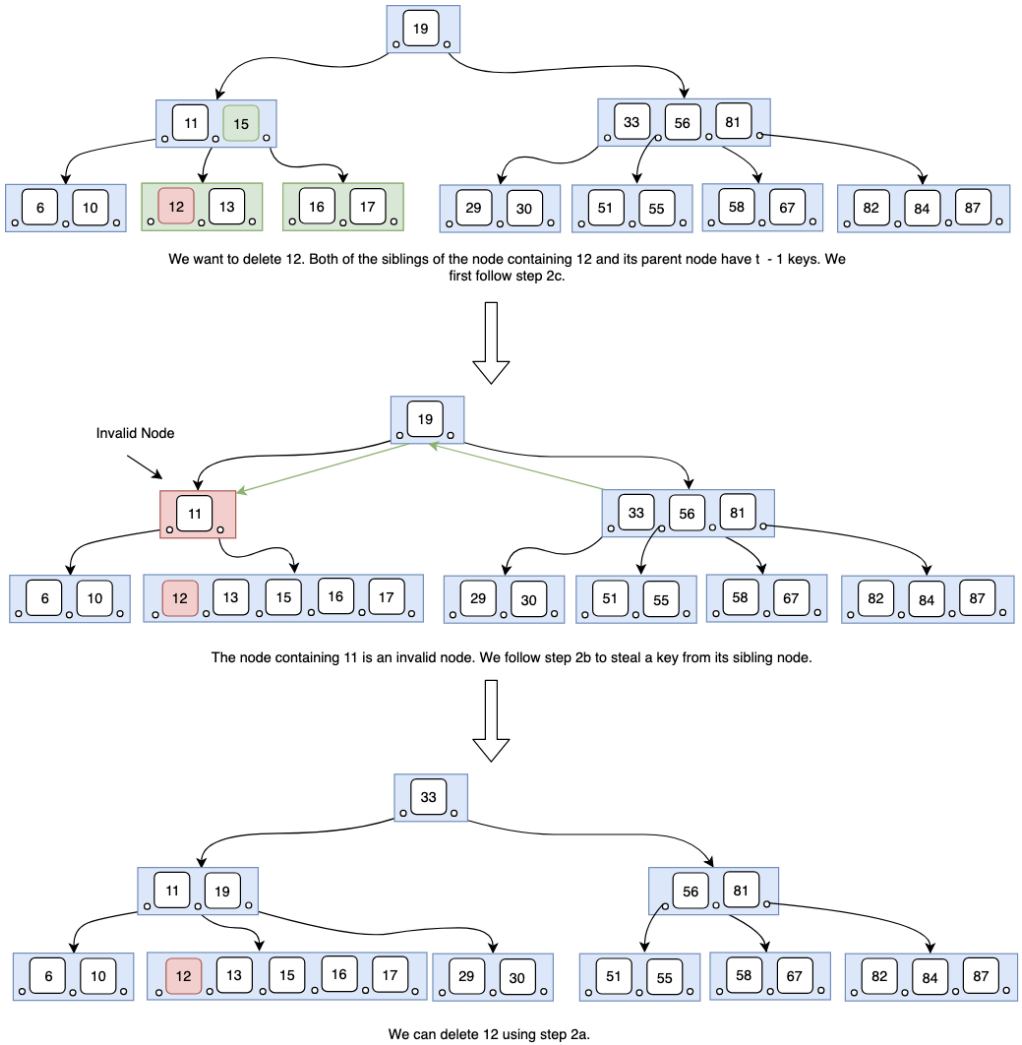


We want to delete 29. Immediate sibling of the node containing 29 has only t - 1 keys. In this case, we merge the node containing 29 with its immediate sibling and one key from the parent node.

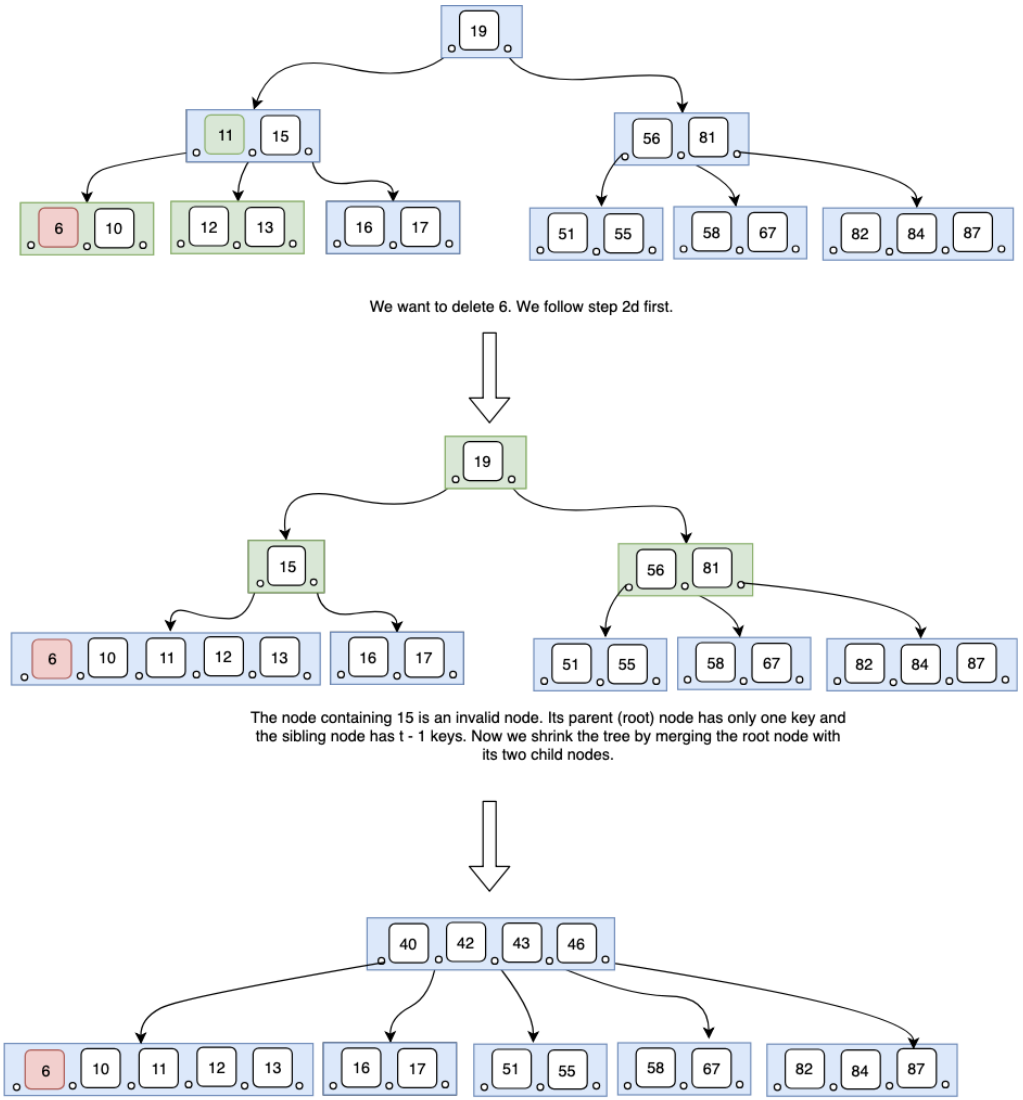


Now we can delete 29 using step 2a.

Remoção- propagação do “merge”



Remoção- propagação do "merge"



Referências

- Livro Nivio Ziviani, Cap. 6
- Livro Cormen, Cap. 19
- Livro Sedgewick Cap. 16 (Seção 16.3)
- `https://algorithmtutor.com/Data-Structures/Tree/B-Trees/`