



UNIVERSIDADE
FEDERAL DE
SERGIPE



DEPARTAMENTO
DE COMPUTAÇÃO

Árvore B

Estruturas de Dados

Bruno Prado

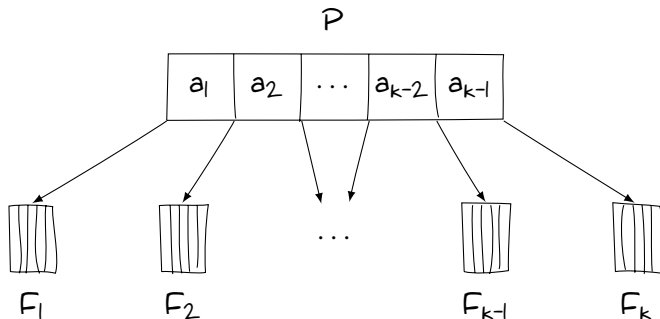
Departamento de Computação / UFS

Introdução

- ▶ O que é uma árvore B?
 - ▶ É uma árvore k -ária balanceada
 - ▶ Mantém todos os nós folha no mesmo nível
 - ▶ Foi criada em 1970 por R. **B**ayer e E. McCreight

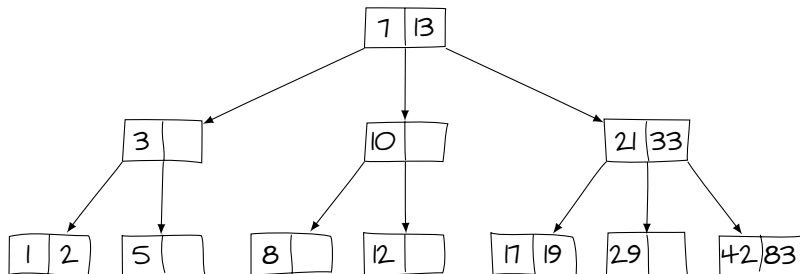
Introdução

- ▶ Propriedades de uma árvore B de ordem k
 - ▶ Todo nó tem $k \div 2$ até k filhos (exceto raiz e folha)
 - ▶ Chaves ordenadas $a_1 \leq a_2 \leq \dots \leq a_{k-1} \leq a_{k-1}$
 - ▶ Todos os nós folhas estão no mesmo nível



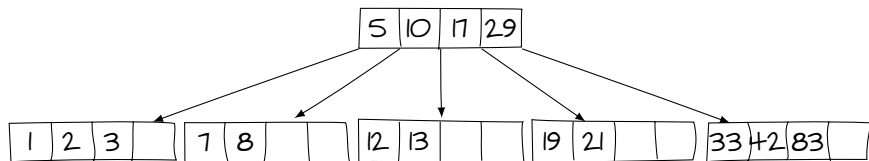
Introdução

► Árvore B de ordem 3



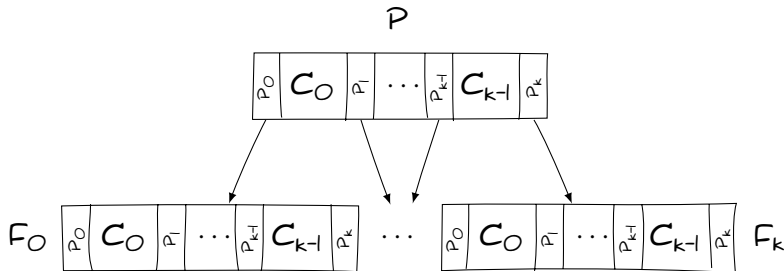
Introdução

► Árvore B de ordem 5



Árvore B

► Definição da estrutura



Árvore B

- ▶ Implementação em C
 - ▶ Estrutura e ponteiros

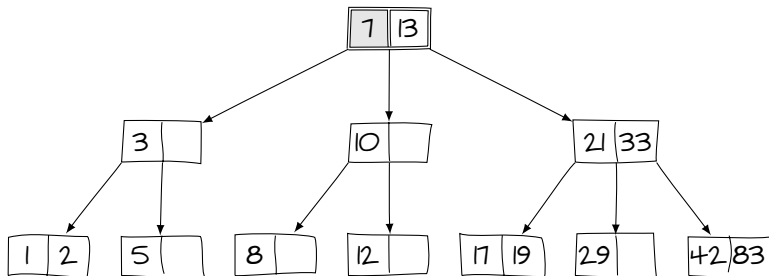
```
1 // Padrão de tipos por tamanho
2 #include <stdint.h>
3 // Estrutura de nó
4 typedef struct no {
5     // Vetor de chaves
6     uint32_t* C;
7     // Vetor de filhos
8     struct no** P;
9     // Quantidade utilizada
10    uint32_t n;
11 } no;
```

Árvore B

- ▶ Operações básicas
 - ▶ Busca
 - ▶ Inserção
 - ▶ Remoção

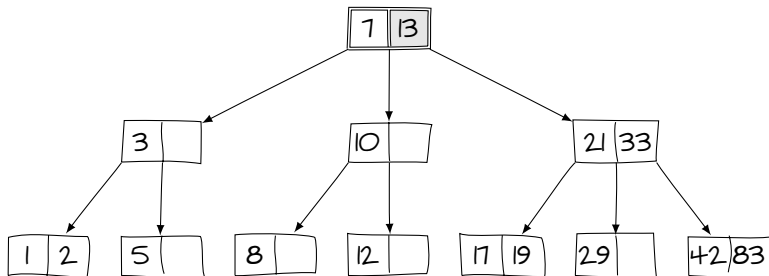
Árvore B

- ▶ Operação de busca
 - ▶ Parâmetro de chave: 19
 - ▶ A busca tem início pela raiz da árvore, checando as chaves do nó e acessando as subárvores



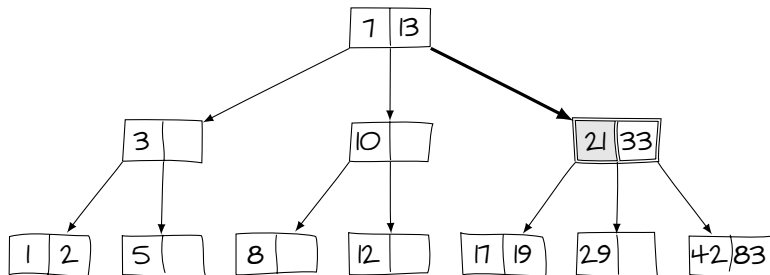
Árvore B

- ▶ Operação de busca
 - ▶ Parâmetro de chave: 19
 - ▶ A busca tem início pela raiz da árvore, checando as chaves do nó e acessando as subárvores



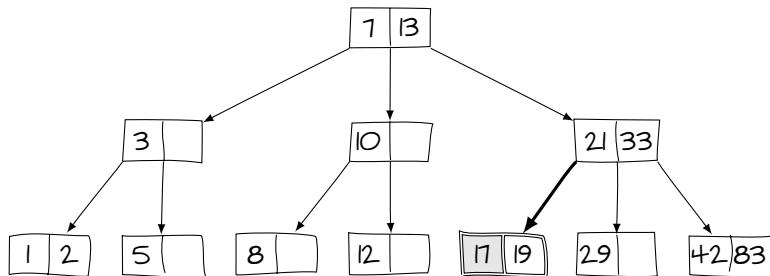
Árvore B

- ▶ Operação de busca
 - ▶ Parâmetro de chave: 19
 - ▶ A busca tem início pela raiz da árvore, checando as chaves do nó e acessando as subárvores



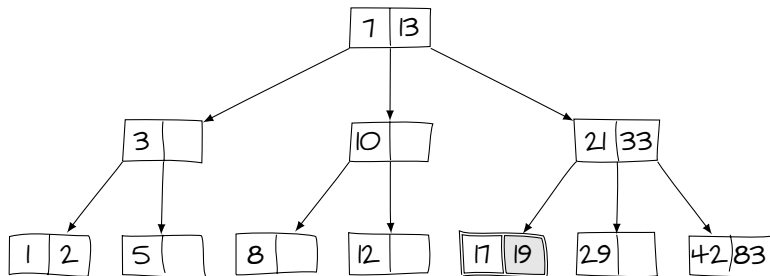
Árvore B

- ▶ Operação de busca
 - ▶ Parâmetro de chave: 19
 - ▶ A busca tem início pela raiz da árvore, checando as chaves do nó e acessando as subárvores



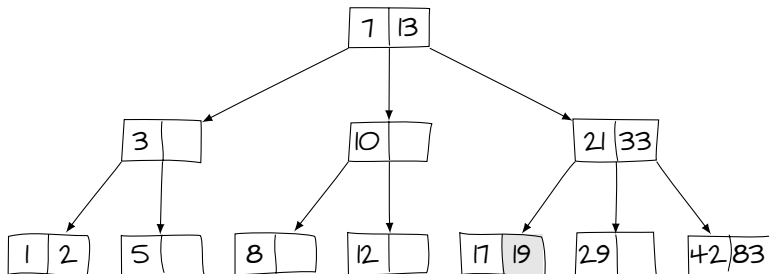
Árvore B

- ▶ Operação de busca
 - ▶ Parâmetro de chave: 19
 - ▶ A busca tem início pela raiz da árvore, checando as chaves do nó e acessando as subárvores



Árvore B

- ▶ Operação de busca
 - ▶ Parâmetro de chave: 19
 - ▶ A referência do nó encontrado é retornada



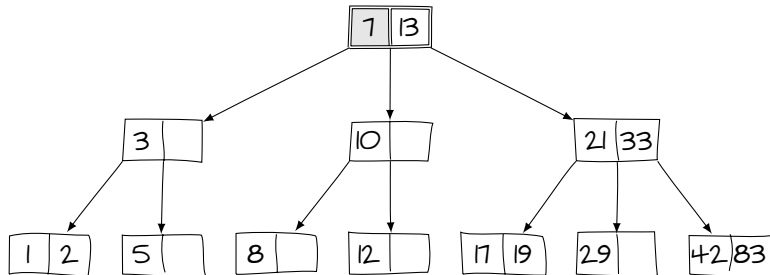
Árvore B

- Implementação em C
 - Busca na árvore

```
1 // Função de busca na árvore
2 no* busca(no* x, uint32_t c) {
3     no* r = NULL;
4     if(x != NULL) {
5         uint32_t i = 0;
6         while(i < x->n && c > x->C[i])
7             i++;
8         if(i < x->n && c == x->C[i])
9             r = x;
10        else
11            r = busca(x->P[i], c);
12    }
13    return r;
14 }
```

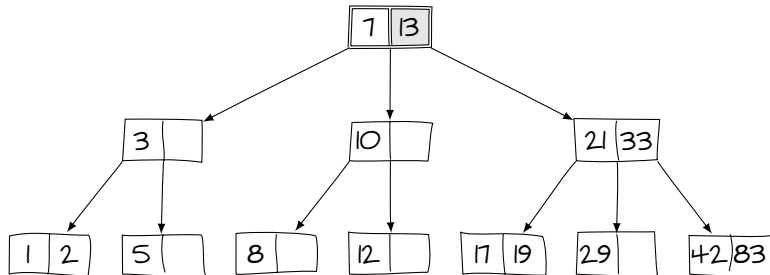
Árvore B

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 11
 - ▶ Caso 1: nó está incompleto



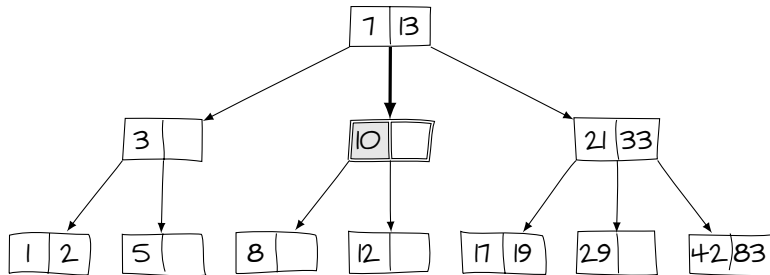
Árvore B

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 11
 - ▶ Caso 1: nó está incompleto



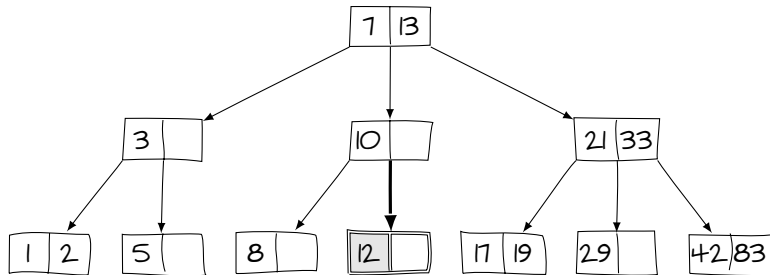
Árvore B

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 11
 - ▶ Caso 1: nó está incompleto



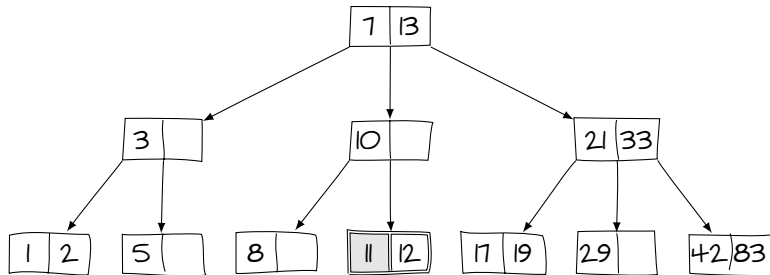
Árvore B

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 11
 - ▶ Caso 1: nó está incompleto



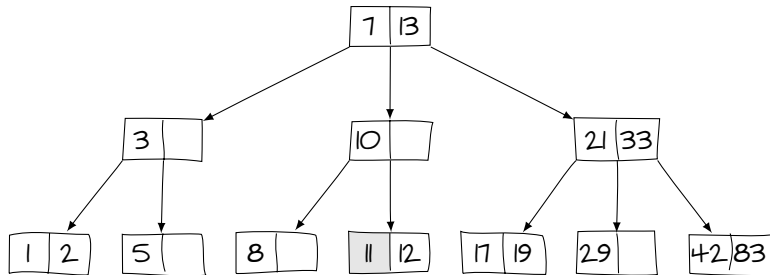
Árvore B

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 11
 - ▶ Caso 1: nó está incompleto



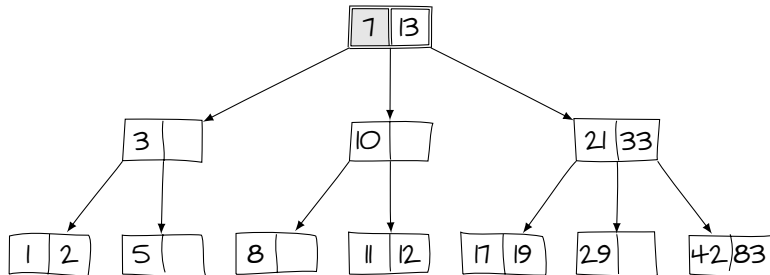
Árvore B

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 11
 - ▶ Caso 1: nó está incompleto



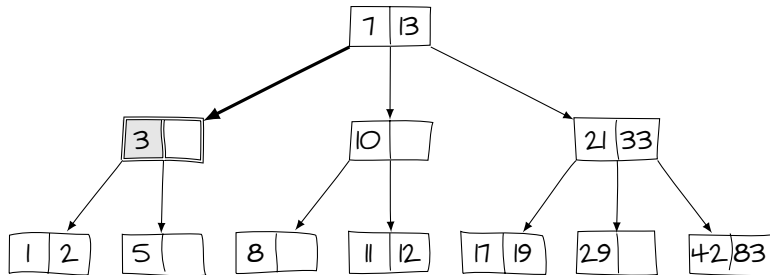
Árvore B

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 0
 - ▶ Caso 2: nó está completo



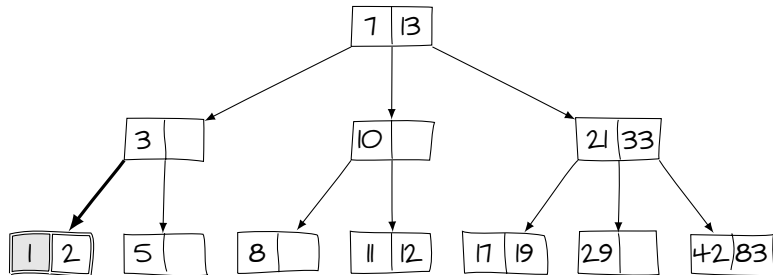
Árvore B

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 0
 - ▶ Caso 2: nó está completo



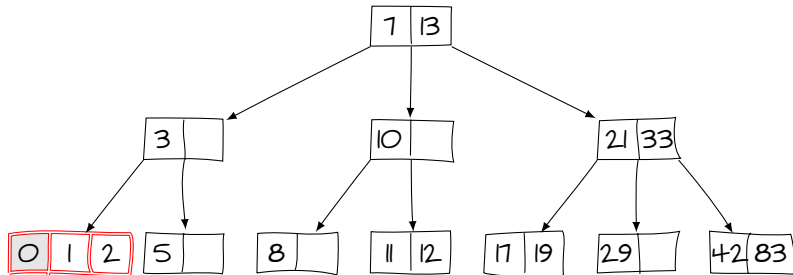
Árvore B

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 0
 - ▶ Caso 2: nó está completo



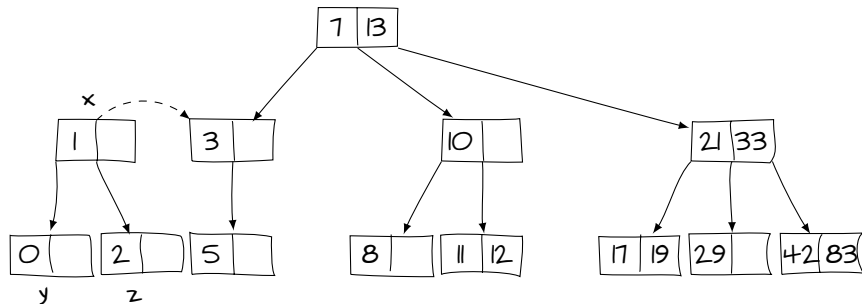
Árvore B

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 0
 - ▶ Caso 2: nó está completo



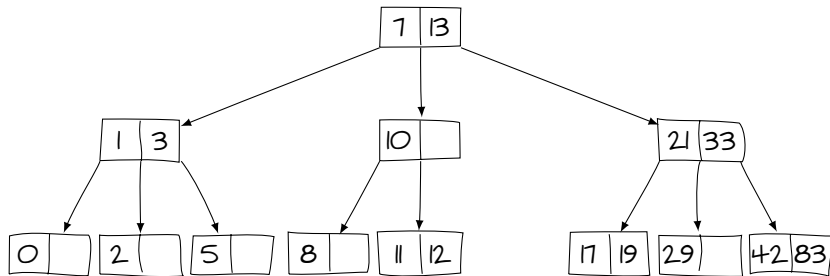
Árvore B

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 0
 - ▶ Caso 2: nó está completo (divisão do nó)



Árvore B

- ▶ Operação de inserção
 - ▶ Parâmetro de chave: 0
 - ▶ Caso 2: nó está completo (divisão do nó)



Árvore B

- Implementação em C
 - Divisão do nó

```
1 // Procedimento de divisão do nó
2 void divisao_no(no* x) {
3     uint32_t i = 0;
4     no* y = criar_no(k);
5     no* z = criar_no(k);
6     for(i = 0; i < k / 2; i++) {
7         y->C[i] = x->C[i]; y->P[i] = x->P[i]; y->n++;
8     } y->P[i] = x->P[i];
9     for(i = (k / 2) + 1; i < x->n; i++) {
10         z->C[i - (k / 2) - 1] = x->C[i]; z->P[i - (k
11             / 2) - 1] = x->P[i]; z->n++;
12     } z->P[i - (k / 2) - 1] = x->P[i];
13     x->C[0] = x->C[k / 2]; x->P[0] = y; x->P[1] = z;
14     x->n = 1;
15 }
```

Árvore B

- Implementação em C
 - Divisão do nó

```
1 // Procedimento de divisão do nó
2 void divisao_no(no* x) {
3     uint32_t i = 0;
4     no* y = criar_no(k);
5     no* z = criar_no(k);
6     for(i = 0; i < k / 2; i++) {
7         y->C[i] = x->C[i]; y->P[i] = x->P[i]; y->n++;
8     } y->P[i] = x->P[i];
9     for(i = (k / 2) + 1; i < x->n; i++) {
10         z->C[i - (k / 2) - 1] = x->C[i]; z->P[i - (k
11             / 2) - 1] = x->P[i]; z->n++;
12     } z->P[i - (k / 2) - 1] = x->P[i];
13     x->C[0] = x->C[k / 2]; x->P[0] = y; x->P[1] = z;
14     x->n = 1;
15 }
```

Instanciação dos novos nós y e z

Árvore B

- Implementação em C
 - Divisão do nó

```
1 // Procedimento de divisão do nó
2 void divisao_no(no* x) {
3     uint32_t i = 0;
4     no* y = criar_no(k);
5     no* z = criar_no(k);
6     for(i = 0; i < k / 2; i++) {
7         y->C[i] = x->C[i]; y->P[i] = x->P[i]; y->n++;
8     } y->P[i] = x->P[i];
9     for(i = (k / 2) + 1; i < x->n; i++) {
10         z->C[i - (k / 2) - 1] = x->C[i]; z->P[i - (k
11             / 2) - 1] = x->P[i]; z->n++;
12     } z->P[i - (k / 2) - 1] = x->P[i];
13     x->C[0] = x->C[k / 2]; x->P[0] = y; x->P[1] = z;
14     x->n = 1;
15 }
```

Cópia da metade inferior de x para y

Árvore B

- Implementação em C
 - Divisão do nó

```
1 // Procedimento de divisão do nó
2 void divisao_no(no* x) {
3     uint32_t i = 0;
4     no* y = criar_no(k);
5     no* z = criar_no(k);
6     for(i = 0; i < k / 2; i++) {
7         y->C[i] = x->C[i]; y->P[i] = x->P[i]; y->n++;
8     } y->P[i] = x->P[i];
9     for(i = (k / 2) + 1; i < x->n; i++) {
10         z->C[i - (k / 2) - 1] = x->C[i]; z->P[i - (k
11             / 2) - 1] = x->P[i]; z->n++;
12     } z->P[i - (k / 2) - 1] = x->P[i];
13     x->C[0] = x->C[k / 2]; x->P[0] = y; x->P[1] = z;
14     x->n = 1;
15 }
```

Cópia da metade superior de x para z

Árvore B

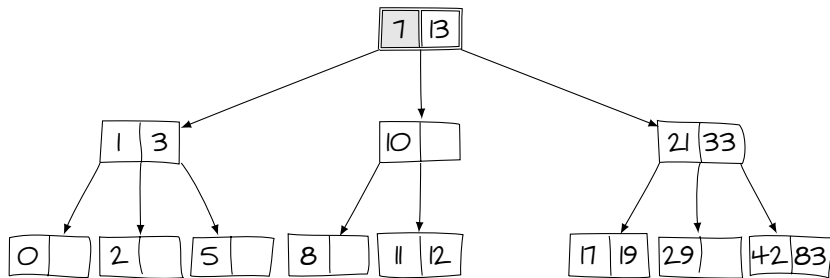
- Implementação em C
 - Divisão do nó

```
1 // Procedimento de divisão do nó
2 void divisao_no(no* x) {
3     uint32_t i = 0;
4     no* y = criar_no(k);
5     no* z = criar_no(k);
6     for(i = 0; i < k / 2; i++) {
7         y->C[i] = x->C[i]; y->P[i] = x->P[i]; y->n++;
8     } y->P[i] = x->P[i];
9     for(i = (k / 2) + 1; i < x->n; i++) {
10         z->C[i - (k / 2) - 1] = x->C[i]; z->P[i - (k
11             / 2) - 1] = x->P[i]; z->n++;
12     } z->P[i - (k / 2) - 1] = x->P[i];
13     x->C[0] = x->C[k / 2]; x->P[0] = y; x->P[1] = z;
14     x->n = 1;
15 }
```

A chave de x e os ponteiros para y e z são atualizados

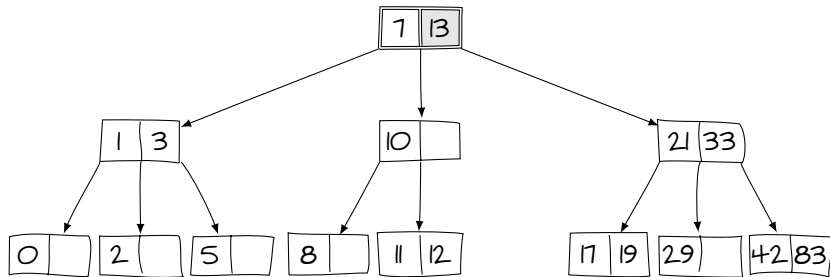
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 17
 - ▶ Caso 1: nó folha com $n > \frac{k}{2}$



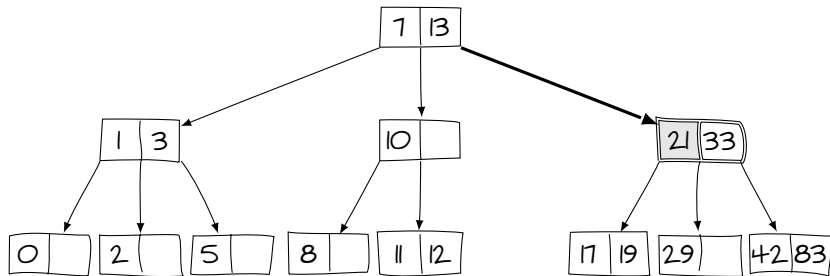
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 17
 - ▶ Caso 1: nó folha com $n > \frac{k}{2}$



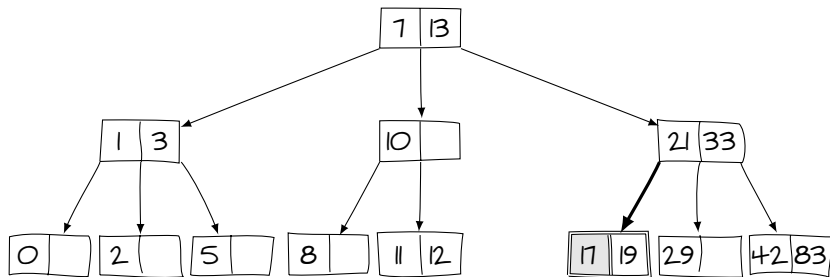
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 17
 - ▶ Caso 1: nó folha com $n > \frac{k}{2}$



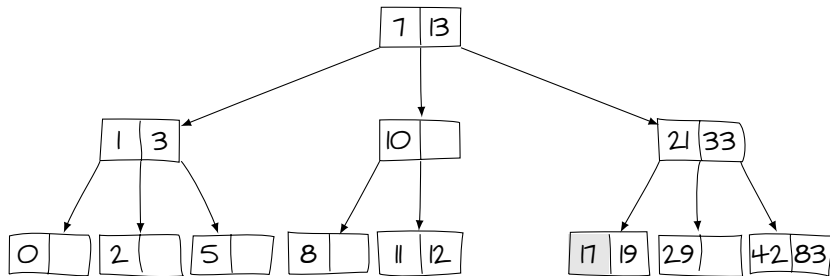
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 17
 - ▶ Caso 1: nó folha com $n > \frac{k}{2}$



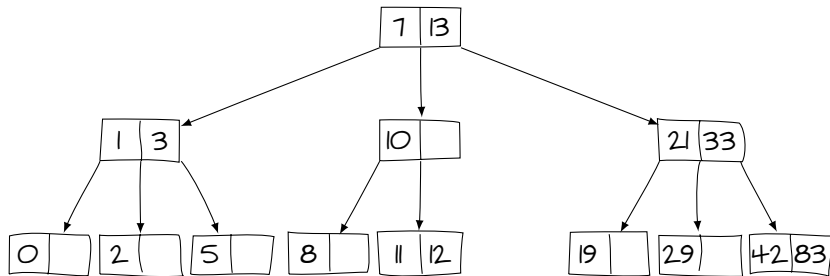
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 17
 - ▶ Caso 1: nó folha com $n > \frac{k}{2}$



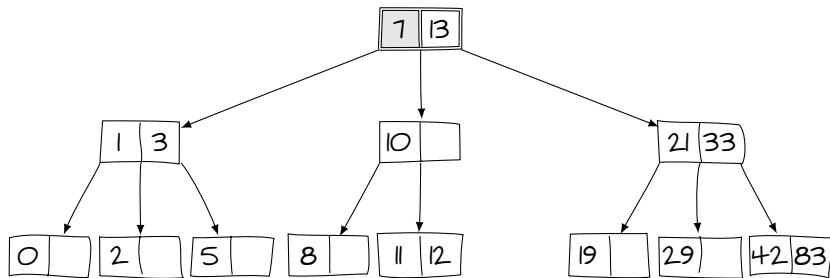
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 17
 - ▶ Caso 1: nó folha com $n > \frac{k}{2}$



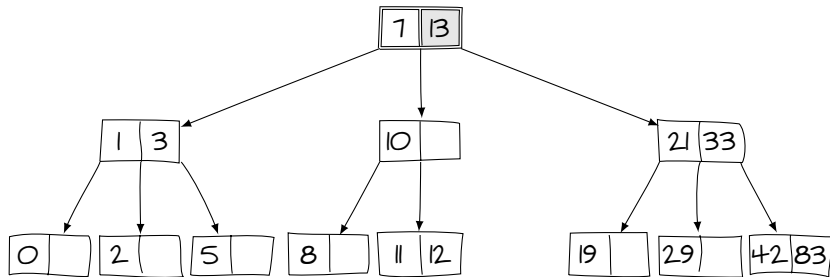
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 19
 - ▶ Caso 2: nó folha com $n = \frac{k}{2}$



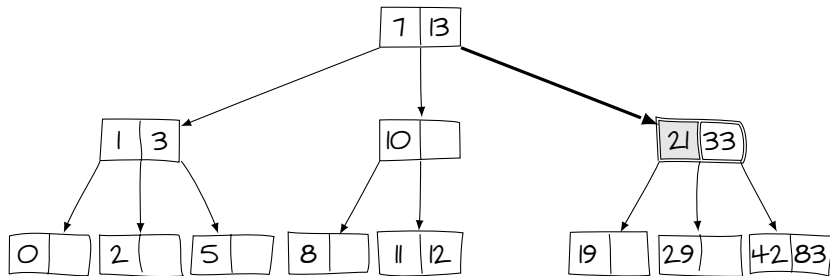
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 19
 - ▶ Caso 2: nó folha com $n = \frac{k}{2}$



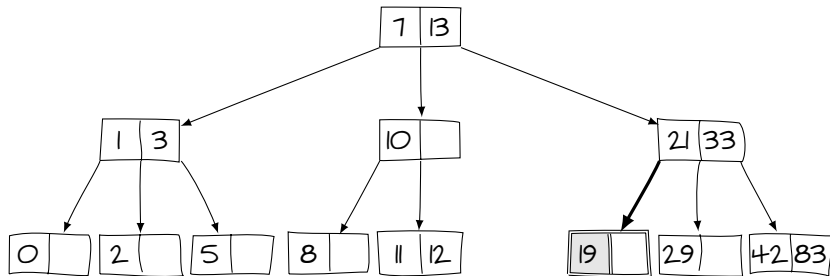
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 19
 - ▶ Caso 2: nó folha com $n = \frac{k}{2}$



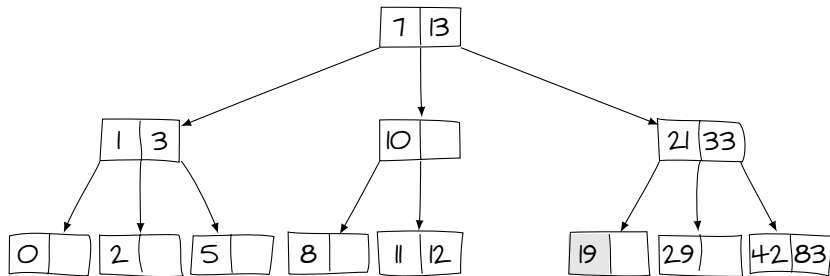
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 19
 - ▶ Caso 2: nó folha com $n = \frac{k}{2}$



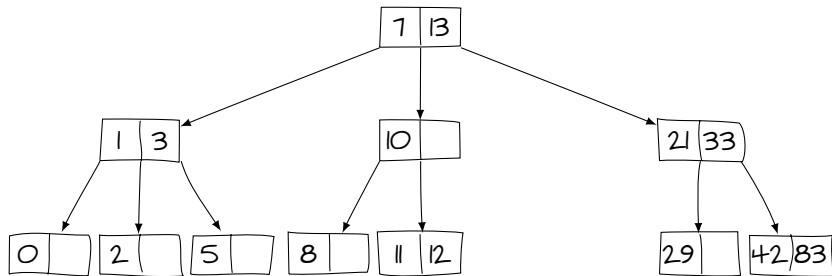
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 19
 - ▶ Caso 2: nó folha com $n = \frac{k}{2}$



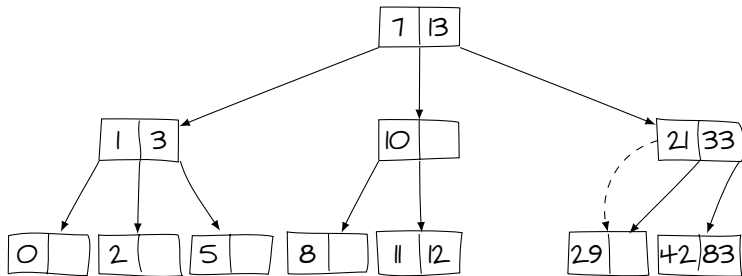
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 19
 - ▶ Caso 2: nó folha com $n = \frac{k}{2}$



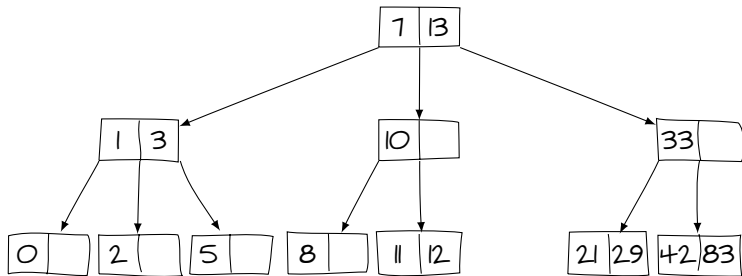
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 19
 - ▶ Caso 2: nó folha com $n = \frac{k}{2}$



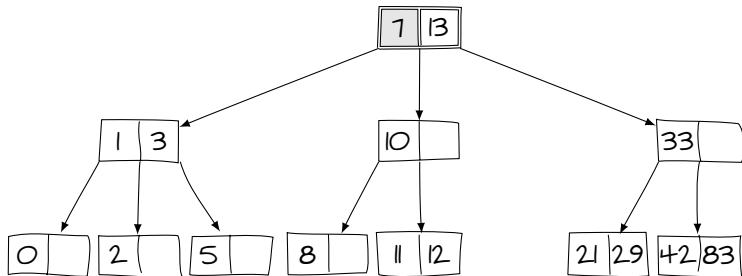
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 19
 - ▶ Caso 2: nó folha com $n = \frac{k}{2}$



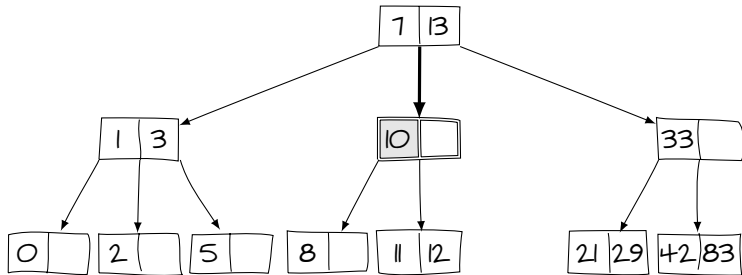
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 10
 - ▶ Caso 3: nó interno com um filho $n > \frac{k}{2}$



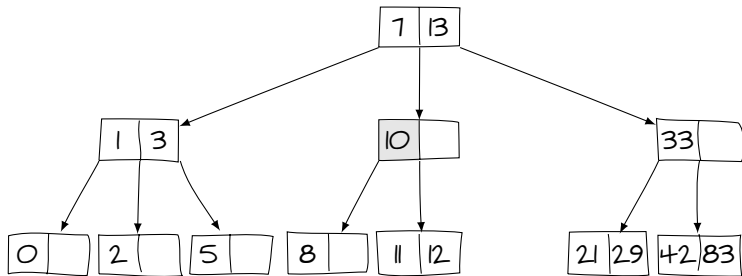
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 10
 - ▶ Caso 3: nó interno com um filho $n > \frac{k}{2}$



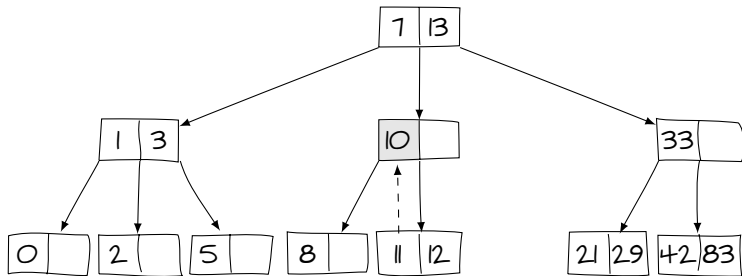
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 10
 - ▶ Caso 3: nó interno com um filho $n > \frac{k}{2}$



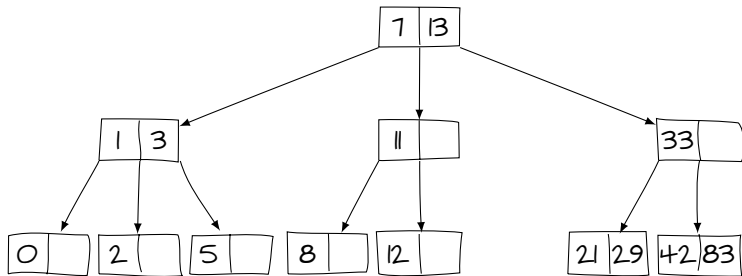
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 10
 - ▶ Caso 3: nó interno com um filho $n > \frac{k}{2}$



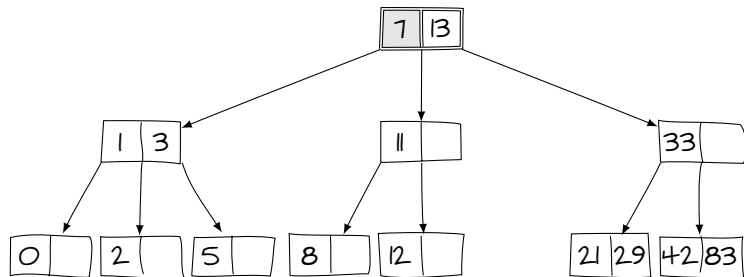
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 10
 - ▶ Caso 3: nó interno com um filho $n > \frac{k}{2}$



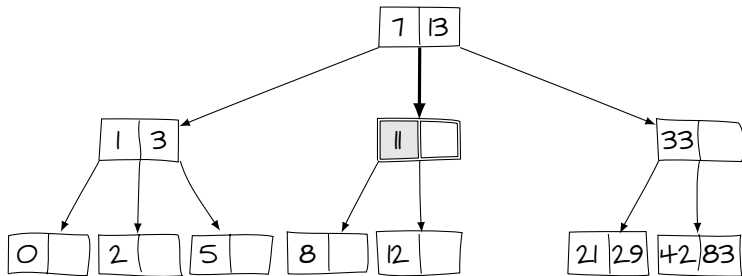
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 11
 - ▶ Caso 4: nó interno com filhos $n = \frac{k}{2}$



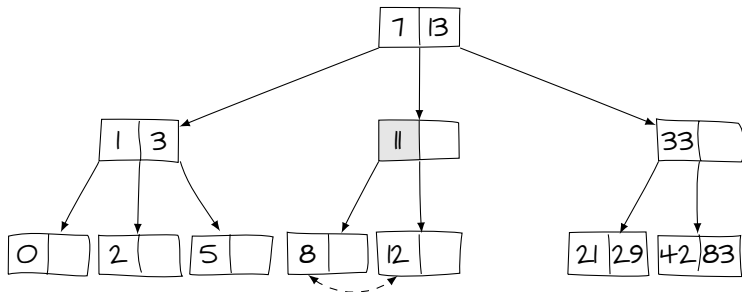
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 11
 - ▶ Caso 4: nó interno com filhos $n = \frac{k}{2}$



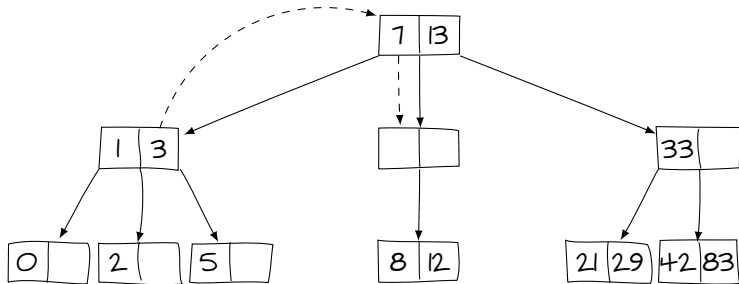
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 11
 - ▶ Caso 4: nó interno com filhos $n = \frac{k}{2}$



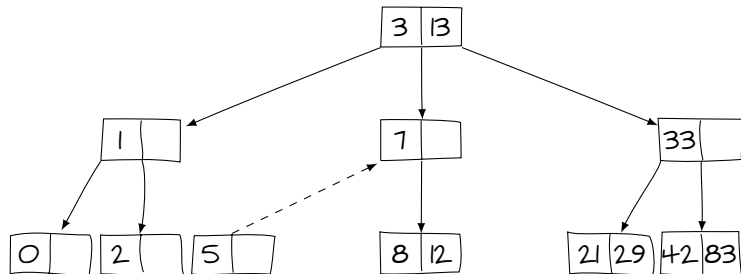
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 11
 - ▶ Caso 4: nó interno com filhos $n = \frac{k}{2}$



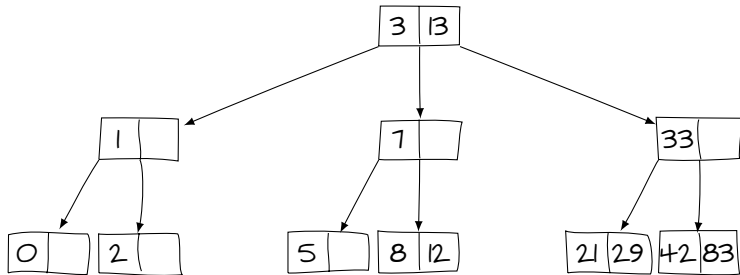
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 11
 - ▶ Caso 4: nó interno com filhos $n = \frac{k}{2}$



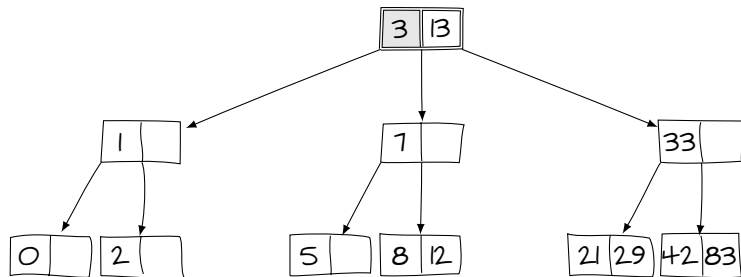
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 11
 - ▶ Caso 4: nó interno com filhos $n = \frac{k}{2}$



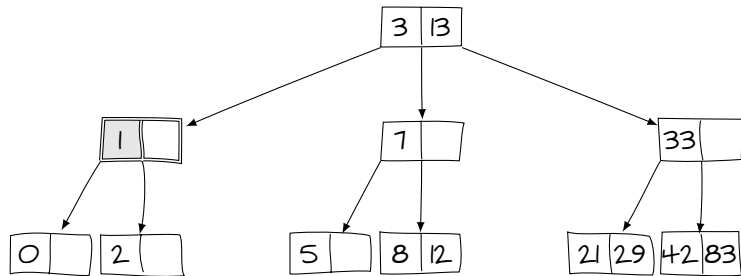
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 1
 - ▶ Caso 4: nó interno com filhos $n = \frac{k}{2}$



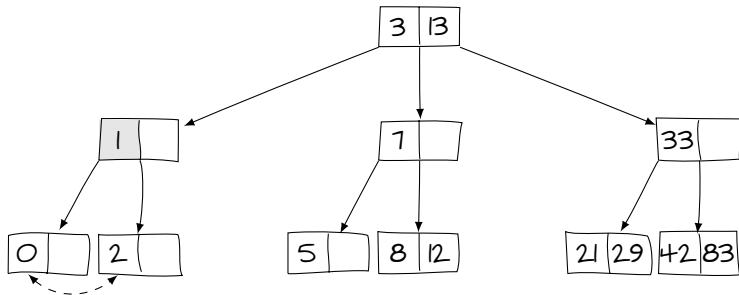
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 1
 - ▶ Caso 4: nó interno com filhos $n = \frac{k}{2}$



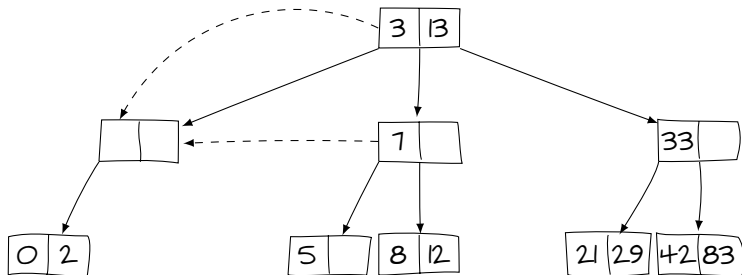
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 1
 - ▶ Caso 4: nó interno com filhos $n = \frac{k}{2}$



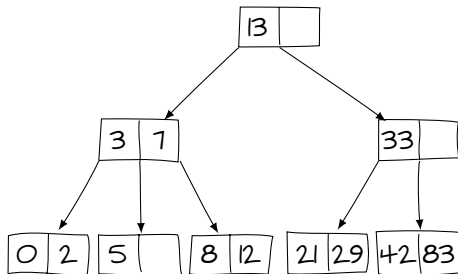
Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 1
 - ▶ Caso 4: nó interno com filhos $n = \frac{k}{2}$



Árvore B

- ▶ Operação de remoção
 - ▶ Parâmetro de chave: 1
 - ▶ Caso 4: nó interno com filhos $n = \frac{k}{2}$



Árvore B

- ▶ Análise de complexidade
 - ▶ Com ordem k , no pior caso, as operações percorrem a altura $h = \log_k n$ da árvore com n nós
 - ▶ Espaço: $\Theta(n)$
 - ▶ Tempo: $\Omega(1)$ e $O(\log_k n)$

Exemplo

- ▶ Construa uma árvore B de ordem 3
 - ▶ Insira os elementos com chaves 13, 2, 34, 11, 7, 43 e 9
 - ▶ Realize a remoção dos elementos de chave 7 e 9
 - ▶ Compare a eficiência desta estrutura com relação às árvores binárias balanceadas

Exercício

- ▶ A empresa de tecnologia Poxim Tech está desenvolvendo um banco de dados distribuído para arquivos baseado em *blockchain* e árvore B
 - ▶ Os arquivos possuem nomes + extensão com até 30 caracteres, compostos exclusivamente por letras
 - ▶ A codificação do código *hash* é feita em hexadecimal de 128 bits com caracteres maiúsculos, sendo utilizado como chave para buscas
 - ▶ Operações disponíveis:
 - ▶ **INSERT** nome tamanho hash
 - ▶ **SELECT** hash

Exercício

- ▶ Formato de arquivo de entrada
 - ▶ [*#Ordem da árvore*]
 - ▶ [*#Quantidade de arquivos (n)*]
 - ▶ [*Nome₁*] [*Tamanho₁*] [*Hash₁*]
 - ▶ ⋮
 - ▶ [*Nome_n*] [*Tamanho_n*] [*Hash_n*]
 - ▶ [*#Número de operações (m)*]
 - ▶ [*Operação₁*]
 - ▶ ⋮
 - ▶ [*Operação_m*]

Exercício

► Formato de arquivo de entrada

```
1 3
2 5
3 f.b_1_00000000000000000000000000000000
4 f.c_2_00000000000000000000000000000001
5 f.d_3_00000000000000000000000000000002
6 f.e_4_00000000000000000000000000000003
7 f.f_5_00000000000000000000000000000004
8 3
9 INSERT_f.a_0_FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
10 SELECT_00000000000000000000000000000004
11 SELECT_123456789ABCDEF0123456789ABCDEF0
```

Exercício

- ▶ Formato de arquivo de saída
 - ▶ Conteúdo armazenado pelo nó da árvore

```
1 [0000000000000000000000000000000004]  
2 f.f:size=5,hash=00000000000000000000000000000004  
3 f.a:size=0,hash=FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF  
4 [123456789ABCDEF0123456789ABCDEF0]  
5 -
```