

Árvores B de busca

MC202 - Estrutura de dados

Alexandre Xavier Falcão (afalcao@ic.unicamp.br)

Thiago Vallin Spina (tvspina@ic.unicamp.br)

Árvores B

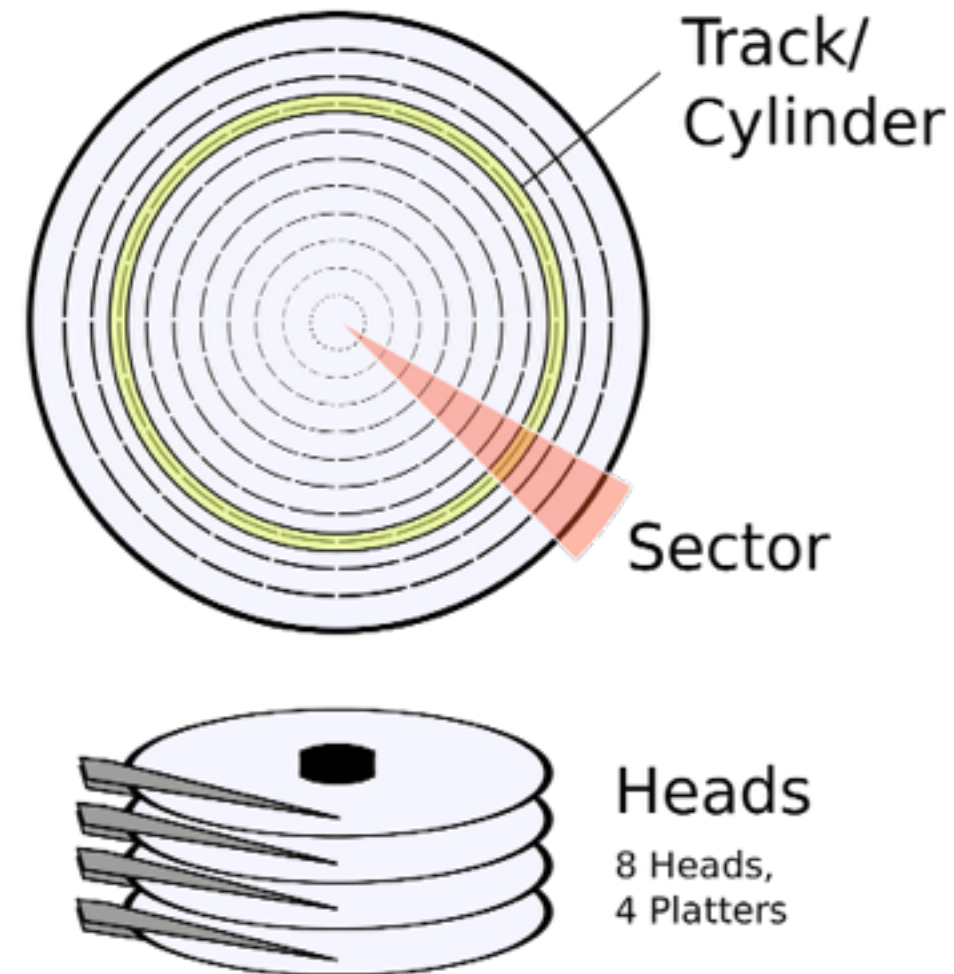
- Generalização de árvores binárias de busca
- Otimizada para acesso a grandes volumes de dados em disco
- Nós contêm múltiplas chaves e múltiplos filhos

Árvores B: Motivação

- Acesso a dados em disco é lento (ordens de magnitude mais devagar que na memória – milissegundos versus nanosegundos)
- Elementos em disco são armazenados contiguamente em blocos de uma mesma faixa ("páginas")
- O acesso de dados no HD é otimizado para trazer múltiplas páginas por vez

Árvores B: Motivação

- Acesso a dados em disco é lento (ordens de magnitude mais devagar que na memória – milissegundos versus nanosegundos)
- Elementos em disco são armazenados contiguamente em blocos de uma mesma faixa ("páginas")
- O acesso de dados no HD é otimizado para trazer múltiplas páginas por vez



Acesso a dados

- Suponha, por exemplo, um arquivo binário armazenado em disco com vários registros de tamanho fixo, contendo informações sobre clientes de uma empresa
- Cada registro é identificado por uma chave primária

RRN	0	1	2	...
Registros	10	15	5	

- O Sistema Operacional agrupa esses registros por página e cada operação de leitura/escrita envolve trazer para a memória cache ou levar para o disco uma página
- Leitura e gravações são custosas e o objetivo é minimizá-las quando inserimos, buscamos e removemos registros
- O arquivo de dados normalmente não cabe na memória principal

Acesso a dados: Estratégia

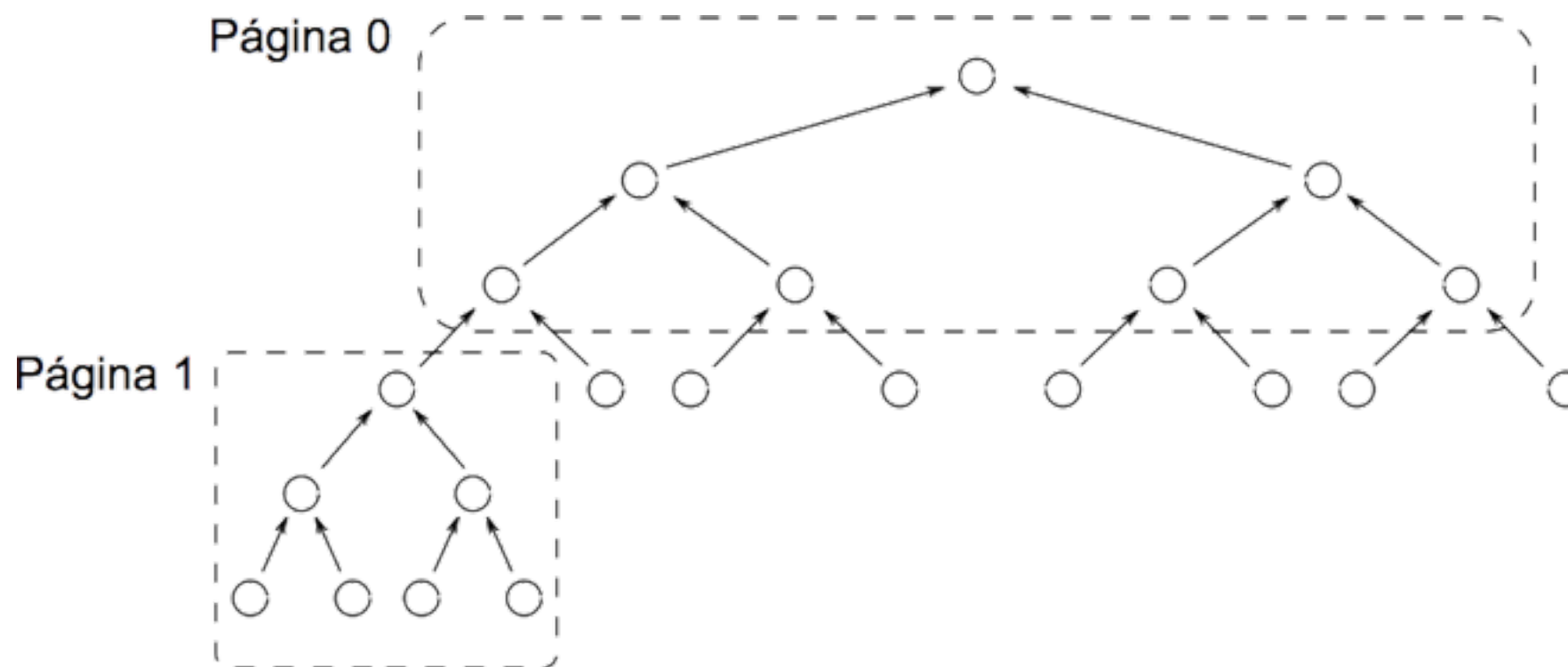
- Guardar a chave dos registros em um arquivo separado (índice primário)
- Menor que o arquivo de registros (cabe parcial/totalmente na memória)
- As operações de inserção, remoção e busca envolvem carregar o índice de acordo com alguma estrutura de dados

Página	0					1	
Índice	5	2	10	0	15	1	...

- Exemplo: durante a busca, ao encontrar o elemento de acordo com sua chave primária o acesso aos dados é feito diretamente via seu RRN (relative record number)

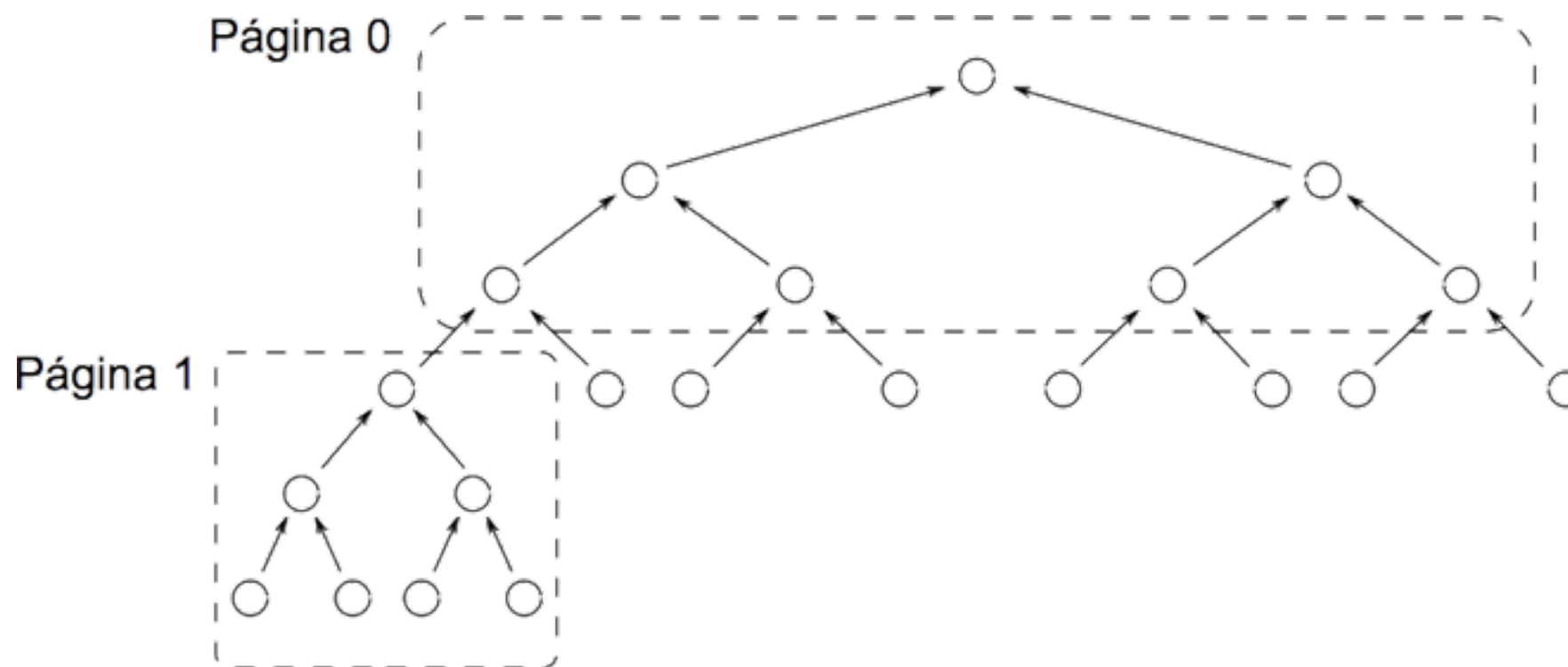
Acesso a dados: Exemplo com AVL

- Supondo 7 registros por página em uma árvore binária AVL, com 2 acessos a disco encontramos qualquer um de 63 registros
- O número de acessos é $\log_{k+1}(N + 1)$, onde N é o número de registros e k o número de registros por página



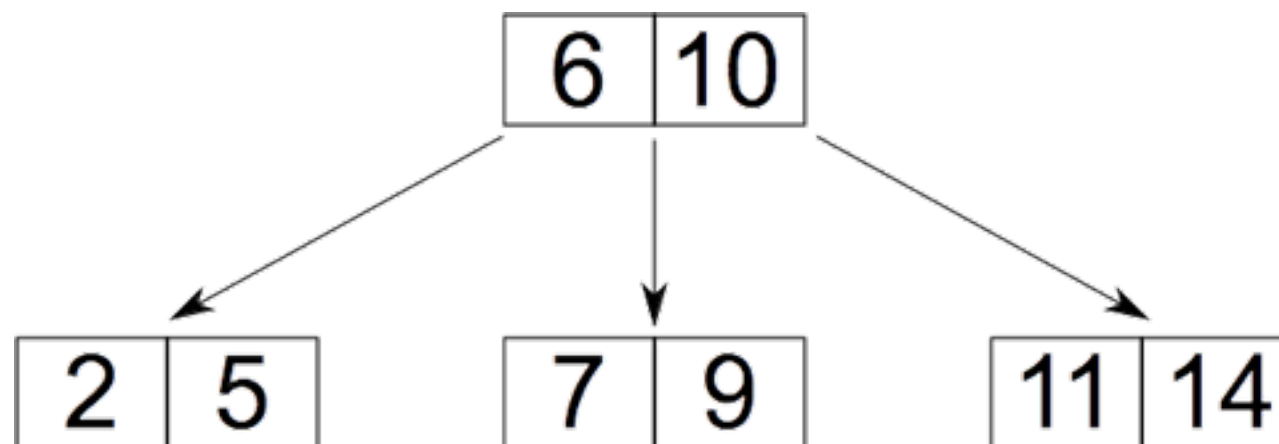
Acesso a dados: Exemplo com AVL

- Para $N = 2^{27} - 1$ registros e $k = 511$ registros por página, apenas 3 acessos a disco seriam necessários
- Contudo, árvores AVL são construídas de cima para baixo, envolvendo o acesso à outras páginas para sua manutenção



Árvores B

- Criadas por R. Bayer e E. M. McCreight em 1972
- Armazenam até $b > 1$ registros com chave por nó (ordem da árvore)
- Construção de baixo para cima
- Na memória, árvores AVL requerem $\log_2(N + 1)$ acessos para achar um nó, ao passo que árvores B necessitam de $\log_b(N + 1)$



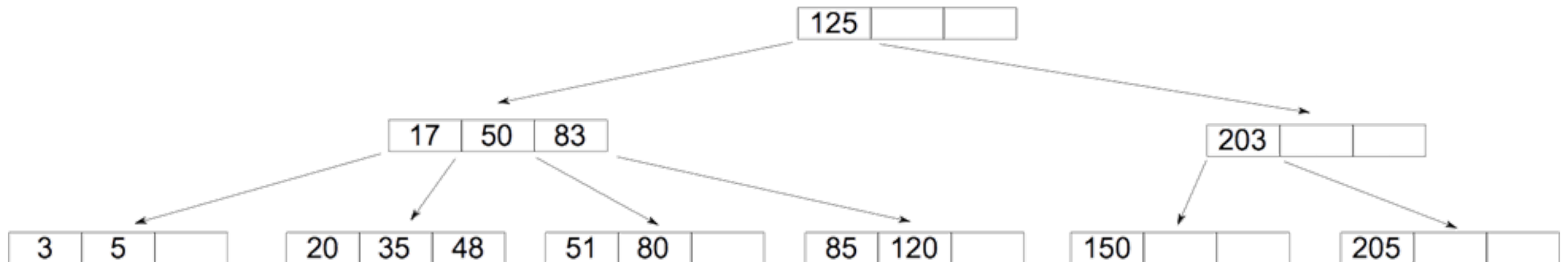
* Omitimos o RRN ou outros dados do registro na representação gráfica

Árvores B: Definição

- Uma árvore é considerada B de ordem $b > 1$ se:
 1. Todas as folhas tem o mesmo nível
 2. Cada nó interno tem um número variável r de registros e $r+1$ filhos, onde:
 - A. $\left\lfloor \frac{b}{2} \right\rfloor \leq r \leq b$ se o nó não é raiz
 - B. $1 \leq r \leq b$ se o nó é raiz
 3. Cada folha tem um número variável r de registros obedecendo à mesma restrição do item 2

Árvores B: Exemplo

- Árvore B de ordem $b = 3$



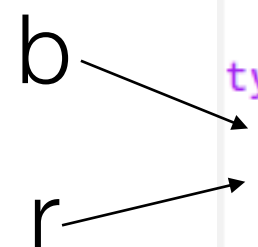
Árvores B

- Em uma árvore B com ordem $b = 255$ pode-se armazenar:

Nível	Mínimo		Máximo	
	Nós	Registros	Nós	Registros
1	1	1	1	255
2	2	2×127	256	256×255
3	2×128	$2 \times 128 \times 127$	256×256	256
4	32.768	4.161.536	16.777.216	4.278.190.080

Árvores B: Implementação

- Em cada nó, os elementos são guardados por ordem crescente de chave



```
#define TAMMAX 511 /*Este eh o tamanho maximo de filhos */

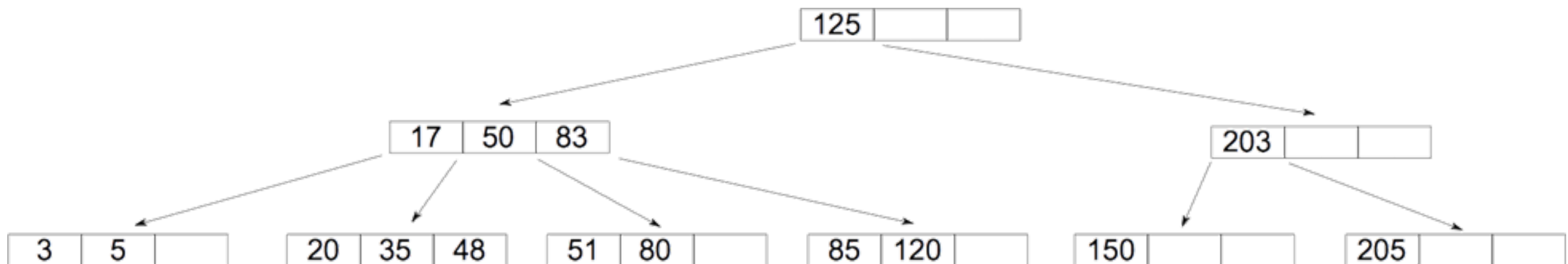
typedef struct arvB { /* Registro que representa nó em árvore B */
    int ordem; /*tamanho maximo desta arvore, que pode ser variavel*/
    int elems; /*quantidade de elementos ocupados da arvore*/
    int info[TAMMAX+1]; /*Vetor contendo todas as informacoes do no*/
    struct arvB *filhos[TAMMAX+2]; /*Vetor contendo todos os filhos */
} ArvB;

ArvB *ArvBCria(int ordem);
/* Cria uma arvore B com a ordem passada pelo parametro. Devolve NULL
   Caso a ordem seja maior do que o tamanho maximo especificado, ou
   se deu algum pau de memoria
*/

int ArvBDestroi(ArvB **arvore);
/* Destroi a Arvore B. O apontador arvore deve retornar Nulo.
   int retorna 1 se tudo deu certo, ou 0 se deu algum tipo de pau.
*/
```

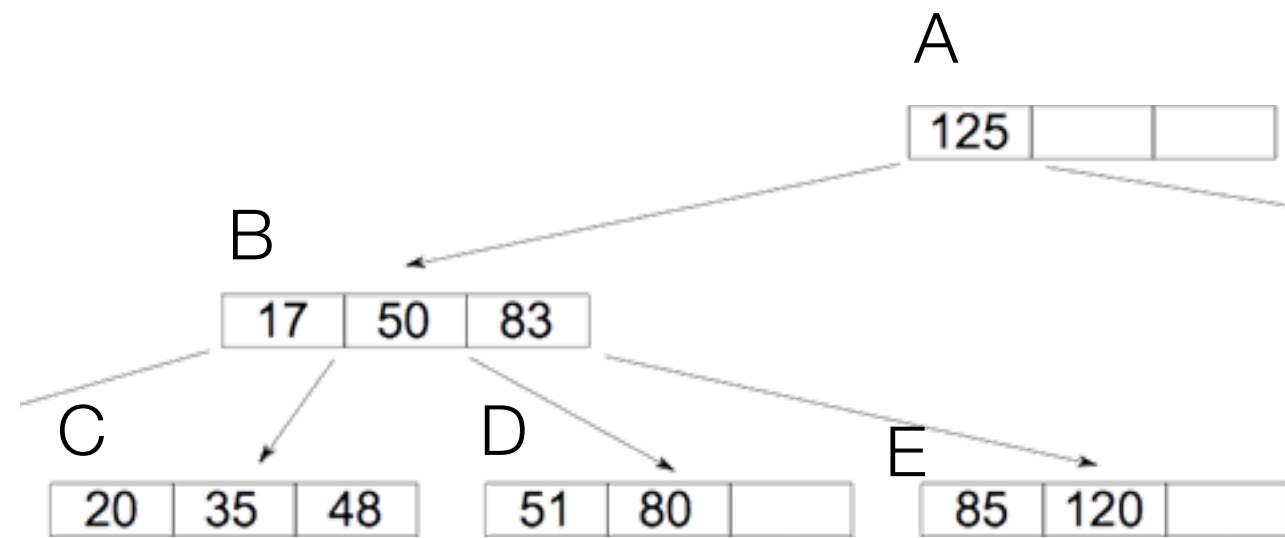
Árvores B: Busca de Elementos

- Como achar o elemento 48?



Árvores B: Implementação da Busca

```
int ArvBBusca(ArvB *arvore, int valor)
/* Busca o valor "Valor" na arvore "arvore".
   A funcao deve retornar 0 se o valor nao existir na arvore ou se
   ocorrer algum tipo de Pau. Caso o valor esteja na arvore, a funcao
   deve retornar o nivel no qual o valor se encontra (a raiz
   possui nivel 1
*/
{
    int nivel = 1, i = 0;
    ArvB *aux;
    aux = arvore;
    if (aux == NULL)
        return 0;
    while(nivel) {
        while((aux->info[i]<valor)&&(i<aux->elems))
            i++;
        /*procura o primeiro elemento maior que o valor, ou a posicao depois
        da arvore*/
        if ((aux->info[i] == valor)&&(i<aux->elems))
            return(nivel);
        /*verifica se achou o elemento (fora da arvore nao vale*/
        else {
            if (aux->filhos[i] != NULL) {
                aux = aux->filhos[i];
                i = 0;
                nivel++;
            } else
                return 0;
            /* se nao achou o elemento neste nivel, tenta descer um nivel.
            se nao conseguir, retorna 0 que faiou.
            */
        }
    }
    return 0;
}
```



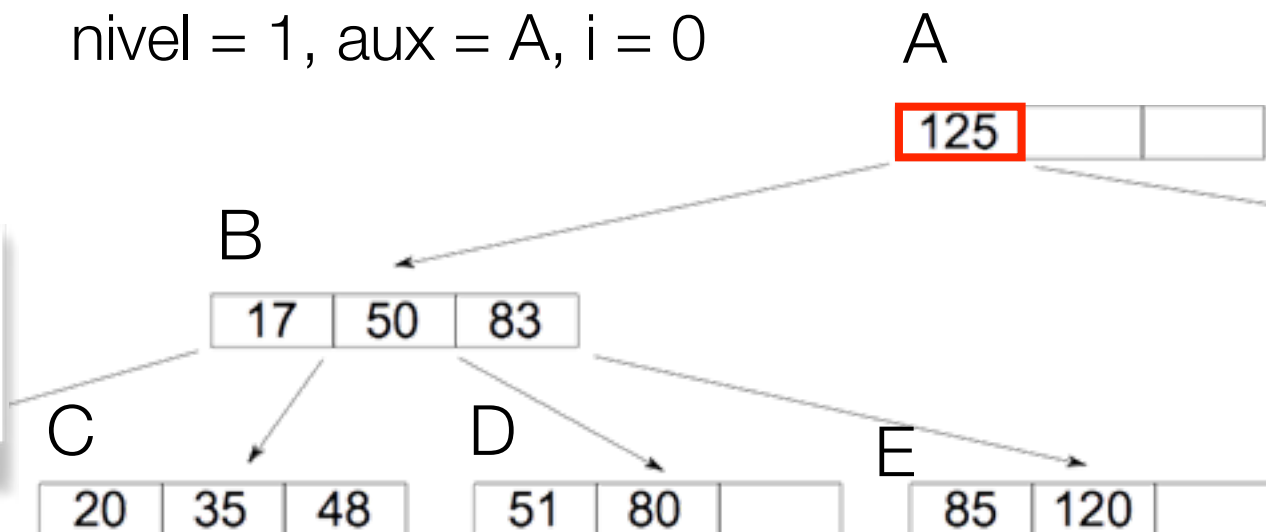
Árvores B: Implementação da Busca

```
int ArvBBusca(ArvB *arvore, int valor)
/* Busca o valor "Valor" na arvore "arvore".
   A funcao deve retornar 0 se o valor nao existir na arvore ou se
   ocorrer algum tipo de Pau. Caso o valor esteja na arvore, a funcao
   deve retornar o nivel no qual o valor se encontra (a raiz
   possui nivel 1
   */
{
    int nivel = 1, i = 0;
    ArvB *aux;
    aux = arvore;
    if (aux == NULL)

        while(nivel) {
            while((aux->info[i]<valor)&&(i<aux->elems))
                i++;
            /*procura o primeiro elemento maior que o valor, ou a posicao depois

            if ((aux->info[i] == valor)&&(i<aux->elems))
                return(nivel);
            /*verifica se achou o elemento (fora da arvore nao vale*/
            else {
                if (aux->filhos[i] != NULL) {
                    aux = aux->filhos[i];
                    i = 0;
                    nivel++;
                } else
                    return 0;
            /* se nao achou o elemento neste nivel, tenta descer um nivel.
               se nao conseguir, retorna 0 que faiou.
            */
        }
    }
    return 0;
}
```

nivel = 1, aux = A, i = 0



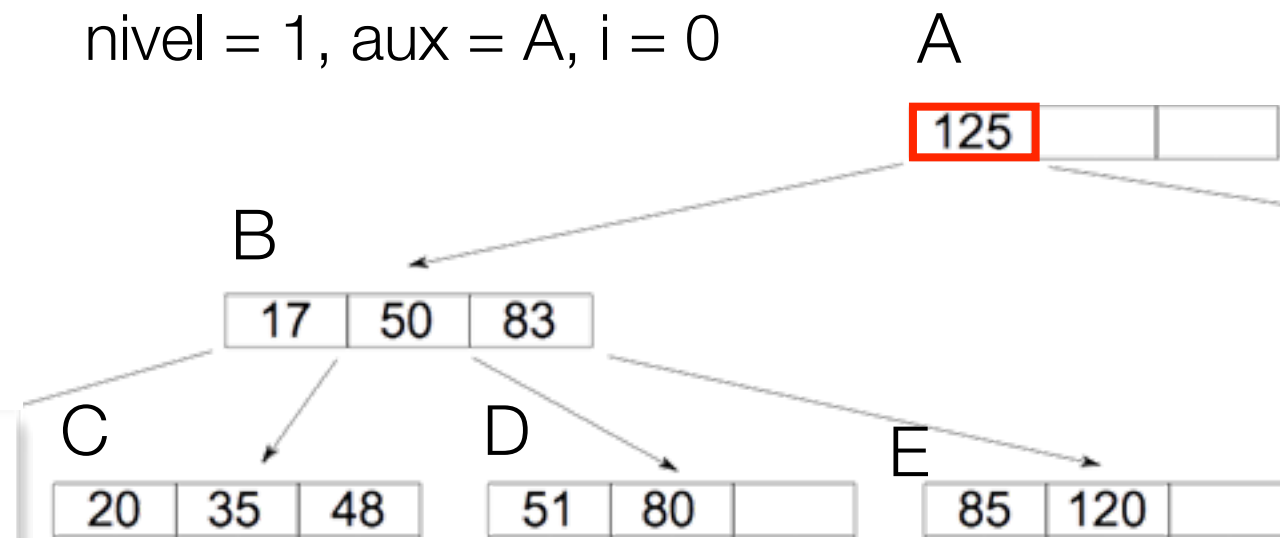
Árvores B: Implementação da Busca

```
int ArvBBusca(ArvB *arvore, int valor)
/* Busca o valor "Valor" na arvore "arvore".
   A funcao deve retornar 0 se o valor nao existir na arvore ou se
   ocorrer algum tipo de Pau. Caso o valor esteja na arvore, a funcao
   deve retornar o nivel no qual o valor se encontra (a raiz
   possui nivel 1
*/
{
    int nivel = 1, i = 0;
    ArvB *aux;
    aux = arvore;
    if (aux == NULL)
        return 0;
    while(nivel) {
        while((aux->info[i]<valor)&&(i<aux->elems))
            i++;
        /*procura o primeiro elemento maior que o valor, ou a posicao depois

        if ((aux->info[i] == valor)&&(i<aux->elems))
            return(nivel);
        /*verifica se achou o elemento (fora da arvore nao vale*/

        if (aux->filhos[i] != NULL) {
            aux = aux->filhos[i];
            i = 0;
            nivel++;
        } else
            return 0;
        /* se nao achou o elemento neste nivel, tenta descer um nivel.
           se nao conseguir, retorna 0 que faiou.
        */
    }
    return 0;
}
```

nivel = 1, aux = A, i = 0

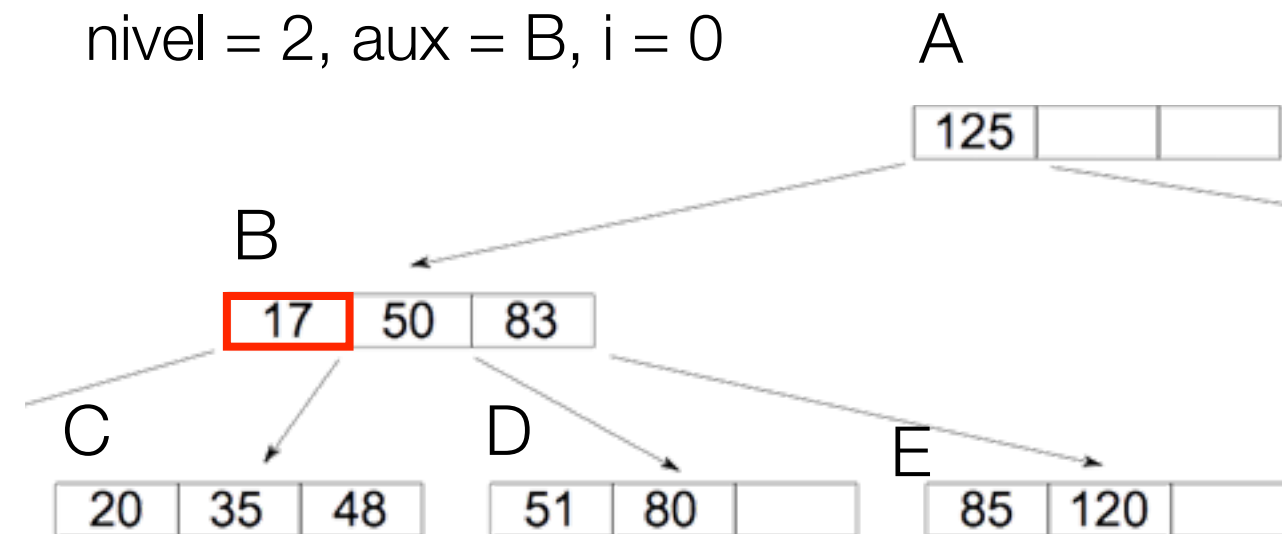


Árvores B: Implementação da Busca

```
int ArvBBusca(ArvB *arvore, int valor)
/* Busca o valor "Valor" na arvore "arvore".
   A funcao deve retornar 0 se o valor nao existir na arvore ou se
   ocorrer algum tipo de Pau. Caso o valor esteja na arvore, a funcao
   deve retornar o nivel no qual o valor se encontra (a raiz
   possui nivel 1
*/
{
    int nivel = 1, i = 0;
    ArvB *aux;
    aux = arvore;
    if (aux == NULL)
        return 0;
    while(nivel) {
        while((aux->info[i]<valor)&&(i<aux->elems))
            i++;
        /*procura o primeiro elemento maior que o valor, ou a posicao depois
        da arvore*/
        if ((aux->info[i] == valor)&&(i<aux->elems))
            return(nivel);
        /*verifica se achou o elemento (fora da arvore nao vale*/

        if (aux->filhos[i] != NULL) {
            aux = aux->filhos[i];
            i = 0;
            nivel++;
        } else
            return 0;
        /* se nao achou o elemento neste nivel, tenta descer um nivel.
        se nao conseguir, retorna 0 que faiou.
        */
    }
    return 0;
}
```

nivel = 2, aux = B, i = 0



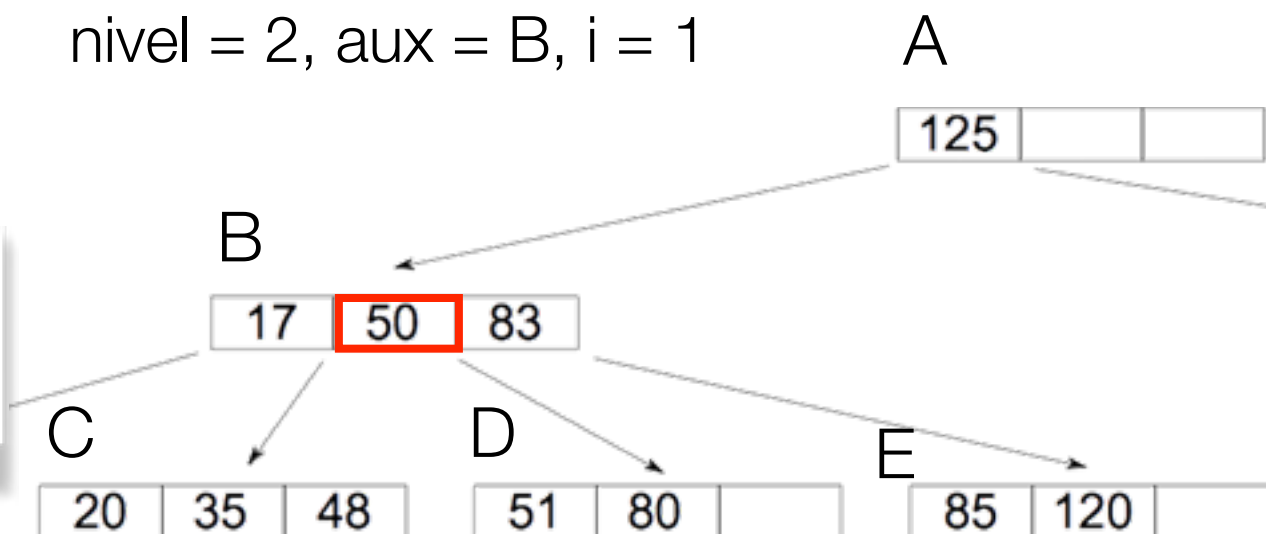
Árvores B: Implementação da Busca

```
int ArvBBusca(ArvB *arvore, int valor)
/* Busca o valor "Valor" na arvore "arvore".
   A funcao deve retornar 0 se o valor nao existir na arvore ou se
   ocorrer algum tipo de Pau. Caso o valor esteja na arvore, a funcao
   deve retornar o nivel no qual o valor se encontra (a raiz
   possui nivel 1
*/
{
    int nivel = 1, i = 0;
    ArvB *aux;
    aux = arvore;
    if (aux == NULL)

        while(nivel) {
            while((aux->info[i]<valor)&&(i<aux->elems))
                i++;
            /*procura o primeiro elemento maior que o valor, ou a posicao depois

            if ((aux->info[i] == valor)&&(i<aux->elems))
                return(nivel);
            /*verifica se achou o elemento (fora da arvore nao vale*/
            else {
                if (aux->filhos[i] != NULL) {
                    aux = aux->filhos[i];
                    i = 0;
                    nivel++;
                } else
                    return 0;
            /* se nao achou o elemento neste nivel, tenta descer um nivel.
               se nao conseguir, retorna 0 que faiou.
            */
        }
    }
    return 0;
}
```

nivel = 2, aux = B, i = 1



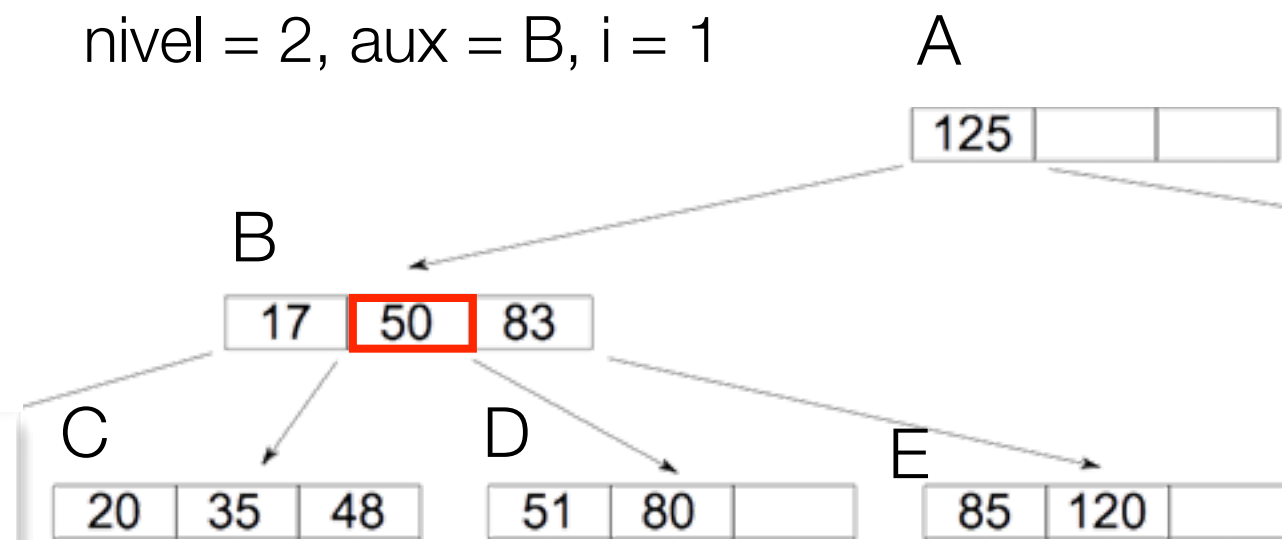
Árvores B: Implementação da Busca

```
int ArvBBusca(ArvB *arvore, int valor)
/* Busca o valor "Valor" na arvore "arvore".
   A funcao deve retornar 0 se o valor nao existir na arvore ou se
   ocorrer algum tipo de Pau. Caso o valor esteja na arvore, a funcao
   deve retornar o nivel no qual o valor se encontra (a raiz
   possui nivel 1
*/
{
    int nivel = 1, i = 0;
    ArvB *aux;
    aux = arvore;
    if (aux == NULL)
        return 0;
    while(nivel) {
        while((aux->info[i]<valor)&&(i<aux->elems))
            i++;
        /*procura o primeiro elemento maior que o valor, ou a posicao depois

        if ((aux->info[i] == valor)&&(i<aux->elems))
            return(nivel);
        /*verifica se achou o elemento (fora da arvore nao vale*/

        if (aux->filhos[i] != NULL) {
            aux = aux->filhos[i];
            i = 0;
            nivel++;
        } else
            return 0;
        /* se nao achou o elemento neste nivel, tenta descer um nivel.
           se nao conseguir, retorna 0 que faiou.
        */
    }
    return 0;
}
```

nivel = 2, aux = B, i = 1

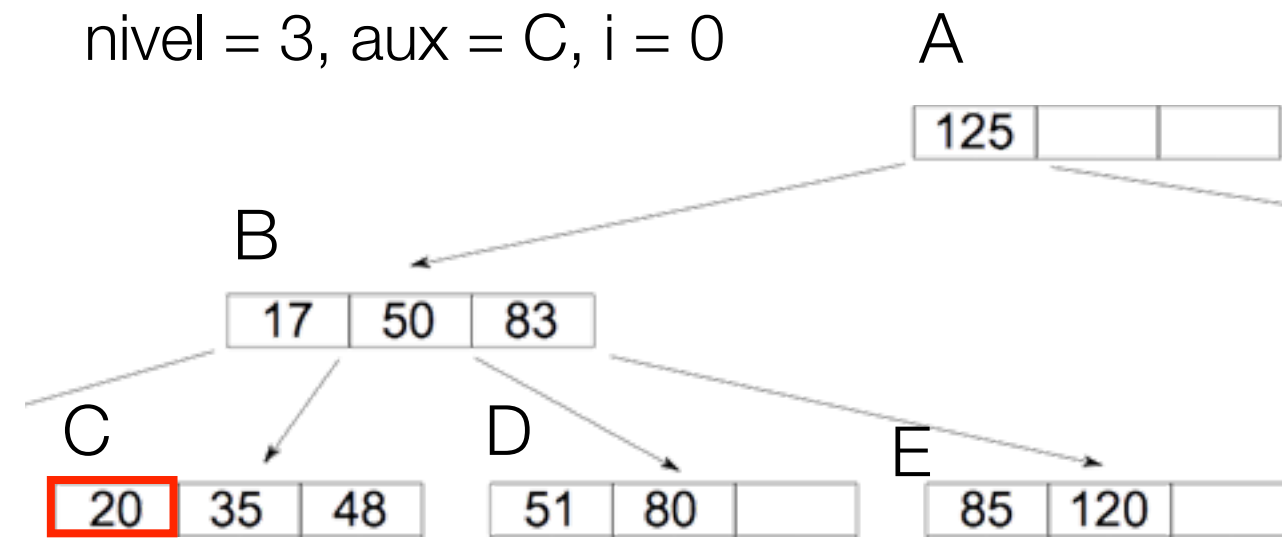


Árvores B: Implementação da Busca

```
int ArvBBusca(ArvB *arvore, int valor)
/* Busca o valor "Valor" na arvore "arvore".
   A funcao deve retornar 0 se o valor nao existir na arvore ou se
   ocorrer algum tipo de Pau. Caso o valor esteja na arvore, a funcao
   deve retornar o nivel no qual o valor se encontra (a raiz
   possui nivel 1
*/
{
    int nivel = 1, i = 0;
    ArvB *aux;
    aux = arvore;
    if (aux == NULL)
        return 0;
    while(nivel) {
        while((aux->info[i]<valor)&&(i<aux->elems))
            i++;
        /*procura o primeiro elemento maior que o valor, ou a posicao depois
        da arvore*/
        if ((aux->info[i] == valor)&&(i<aux->elems))
            return(nivel);
        /*verifica se achou o elemento (fora da arvore nao vale*/

        if (aux->filhos[i] != NULL) {
            aux = aux->filhos[i];
            i = 0;
            nivel++;
        } else
            return 0;
        /* se nao achou o elemento neste nivel, tenta descer um nivel.
        se nao conseguir, retorna 0 que faiou.
        */
    }
    return 0;
}
```

nivel = 3, aux = C, i = 0



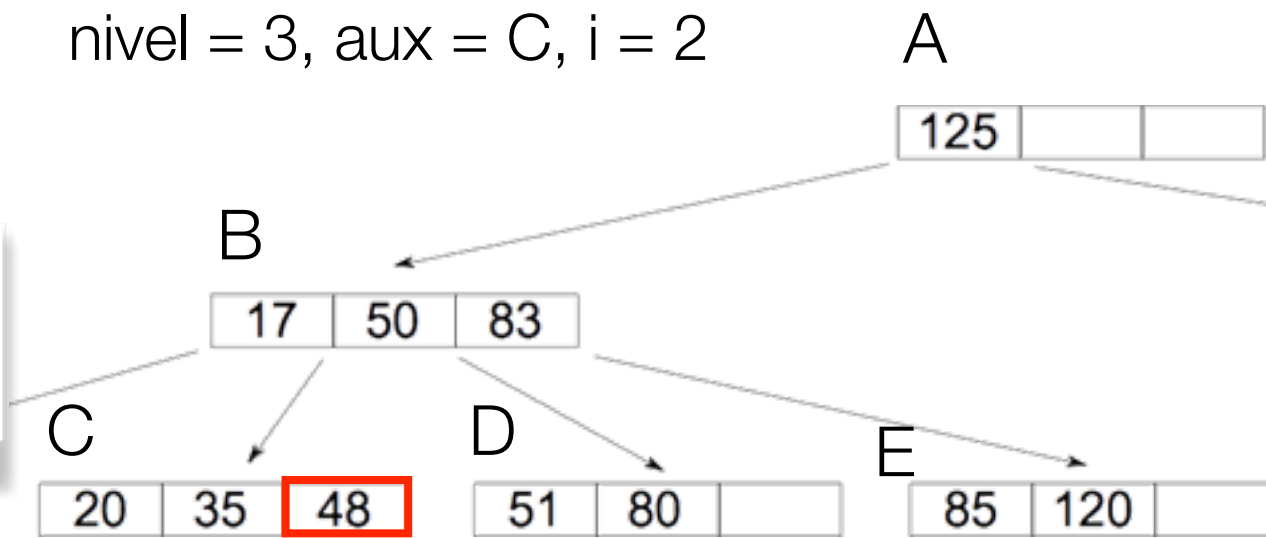
Árvores B: Implementação da Busca

```
int ArvBBusca(ArvB *arvore, int valor)
/* Busca o valor "Valor" na arvore "arvore".
   A funcao deve retornar 0 se o valor nao existir na arvore ou se
   ocorrer algum tipo de Pau. Caso o valor esteja na arvore, a funcao
   deve retornar o nivel no qual o valor se encontra (a raiz
   possui nivel 1
*/
{
    int nivel = 1, i = 0;
    ArvB *aux;
    aux = arvore;
    if (aux == NULL)

        while(nivel) {
            while((aux->info[i]<valor)&&(i<aux->elems))
                i++;
            /*procura o primeiro elemento maior que o valor, ou a posicao depois

            if ((aux->info[i] == valor)&&(i<aux->elems))
                return(nivel);
            /*verifica se achou o elemento (fora da arvore nao vale*/
            else {
                if (aux->filhos[i] != NULL) {
                    aux = aux->filhos[i];
                    i = 0;
                    nivel++;
                } else
                    return 0;
            }
            /* se nao achou o elemento neste nivel, tenta descer um nivel.
               se nao conseguir, retorna 0 que faiou.
            */
        }
    return 0;
}
```

nivel = 3, aux = C, i = 2



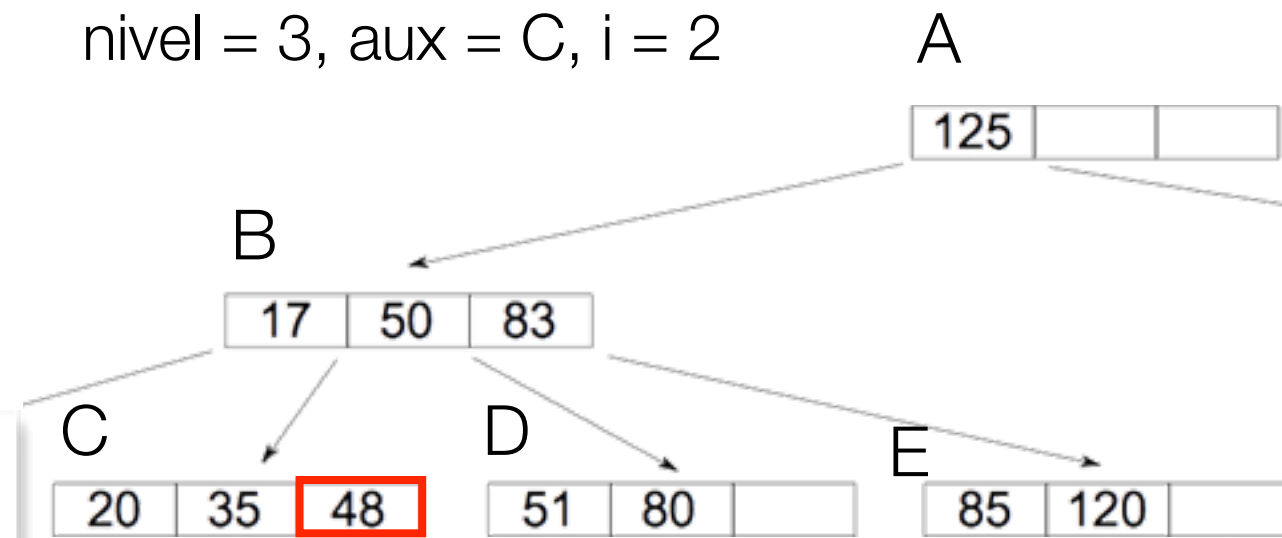
Árvores B: Implementação da Busca

```
int ArvBBusca(ArvB *arvore, int valor)
/* Busca o valor "Valor" na arvore "arvore".
   A funcao deve retornar 0 se o valor nao existir na arvore ou se
   ocorrer algum tipo de Pau. Caso o valor esteja na arvore, a funcao
   deve retornar o nivel no qual o valor se encontra (a raiz
   possui nivel 1
*/
{
    int nivel = 1, i = 0;
    ArvB *aux;
    aux = arvore;
    if (aux == NULL)
        return 0;
    while(nivel) {
        while((aux->info[i]<valor)&&(i<aux->elems))
            i++;
        /*procura o primeiro elemento maior que o valor, ou a posicao depois

        if ((aux->info[i] == valor)&&(i<aux->elems))
            return(nivel);
        /*verifica se achou o elemento (fora da arvore nao vale*/

        if (aux->filhos[i] != NULL) {
            aux = aux->filhos[i];
            i = 0;
            nivel++;
        } else
            return 0;
        /* se nao achou o elemento neste nivel, tenta descer um nivel.
           se nao conseguir, retorna 0 que faiou.
        */
    }
    return 0;
}
```

nivel = 3, aux = C, i = 2



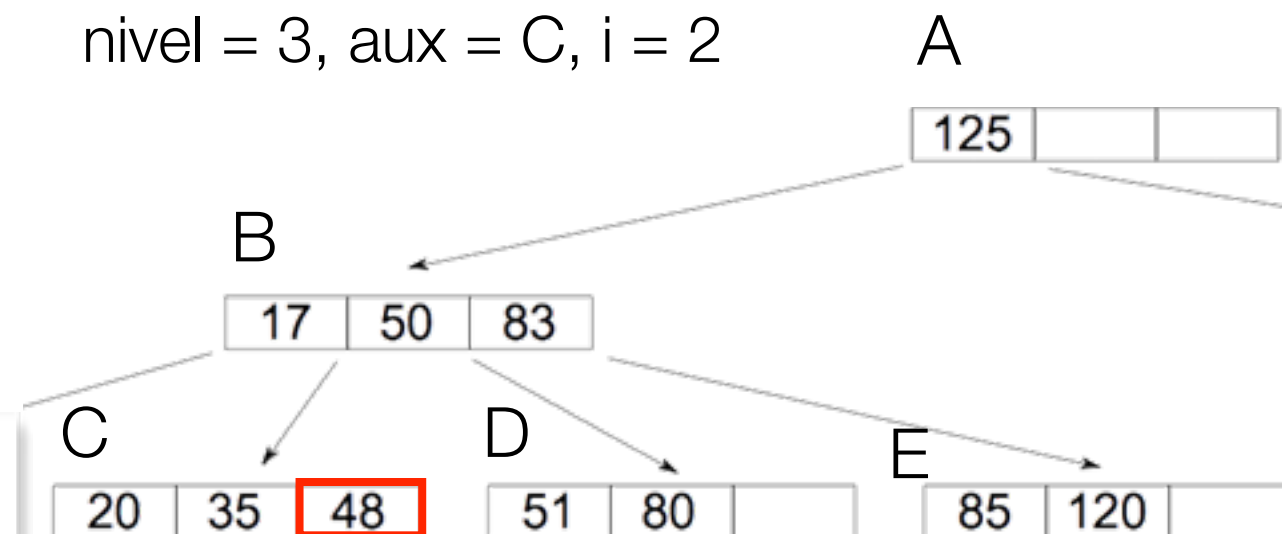
Árvores B: Implementação da Busca

```
int ArvBBusca(ArvB *arvore, int valor)
/* Busca o valor "Valor" na arvore "arvore".
   A funcao deve retornar 0 se o valor nao existir na arvore ou se
   ocorrer algum tipo de Pau. Caso o valor esteja na arvore, a funcao
   deve retornar o nivel no qual o valor se encontra (a raiz
   possui nivel 1
*/
{
    int nivel = 1, i = 0;
    ArvB *aux;
    aux = arvore;
    if (aux == NULL)
        return 0;
    while(nivel) {
        while((aux->info[i]<valor)&&(i<aux->elems))
            i++;
        /*procura o primeiro elemento maior que o valor, ou a posicao depois

        if ((aux->info[i] == valor)&&(i<aux->elems))
            return(nivel);
        /*verifica se achou o elemento (fora da arvore nao vale*/

        if (aux->filhos[i] != NULL) {
            aux = aux->filhos[i];
            i = 0;
            nivel++;
        } else
            return 0;
        /* se nao achou o elemento neste nivel, tenta descer um nivel.
           se nao conseguir, retorna 0 que faiou.
        */
    }
    return 0;
}
```

nivel = 3, aux = C, i = 2



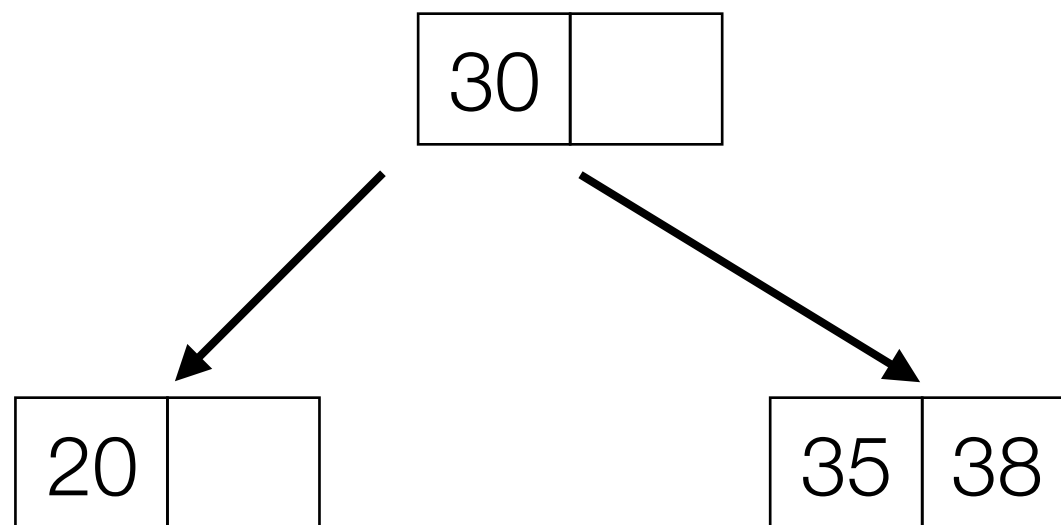
- **Exercício:** a busca no vetor leva tempo linear na ordem b para verificar cada nó, muito embora os registros estão ordenados por chave no vetor. **Otimize a busca!** 14

Árvores B: Inserção de Elementos

- Três casos:
 1. Ávore vazia: cria nó, insere registro e retorna *verdadeiro* para indicar que a altura da árvore aumentou.
 2. Encontre o nó folha onde será feita a inserção e a posição da inserção por busca no nó. Se o nó acomoda o registro ($r < b$), então insira-o e retorne *falso*.
 3. Caso contrário, o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.
- Este caso pode se repetir recursivamente até que o caso 2 ocorra (a árvore aumenta de altura no pior dos casos -> caso especial de raiz cheia).

Árvores B: Exemplos de Inserção

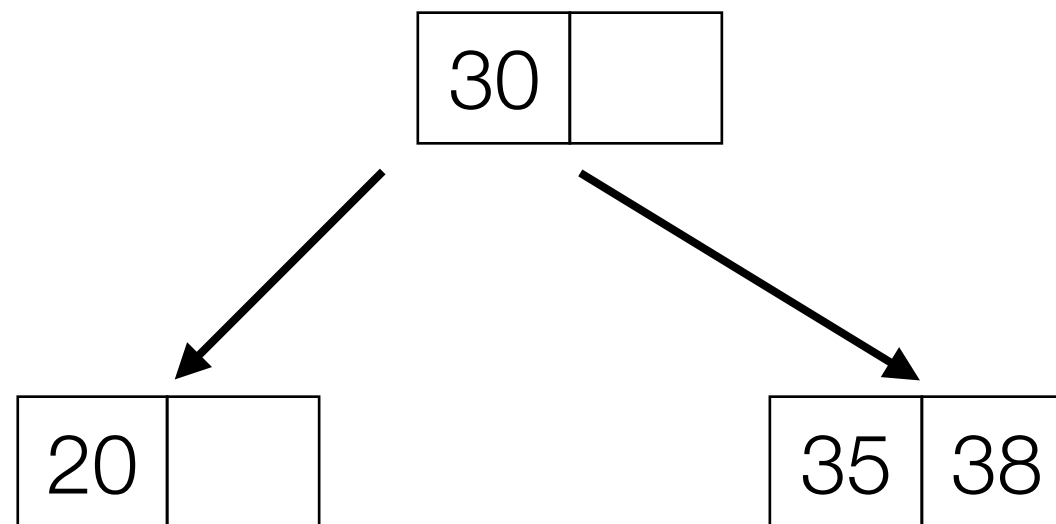
- Árvore inicial



- Insira 25
- Insira 28
- Insira 32

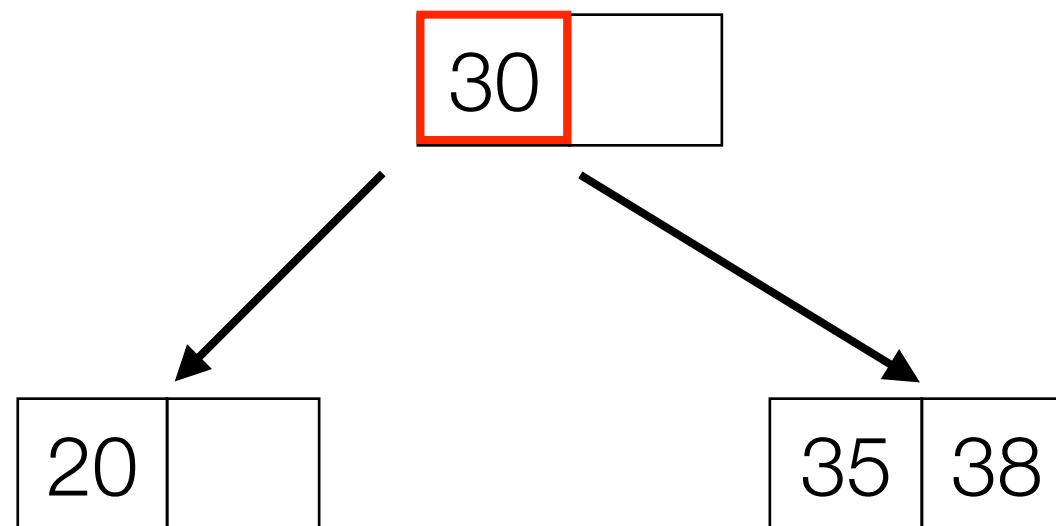
Árvores B: Exemplos de Inserção

- Insira 25:
- Caso 2: Encontre **recursivamente** o nó folha onde será feita a inserção e a posição da inserção por busca no nó. Se o nó acomoda o registro ($r < b$), então insira o registro e retorne *falso*.



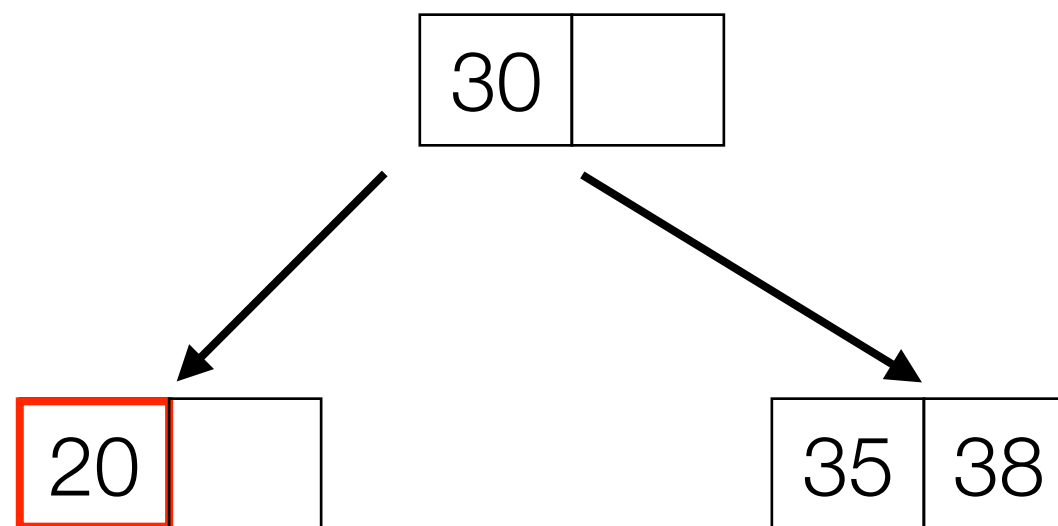
Árvores B: Exemplos de Inserção

- Insira 25:
- Caso 2: Encontre **recursivamente** o nó folha onde será feita a inserção e a posição da inserção por busca no nó. Se o nó acomoda o registro ($r < b$), então insira o registro e retorne *falso*.



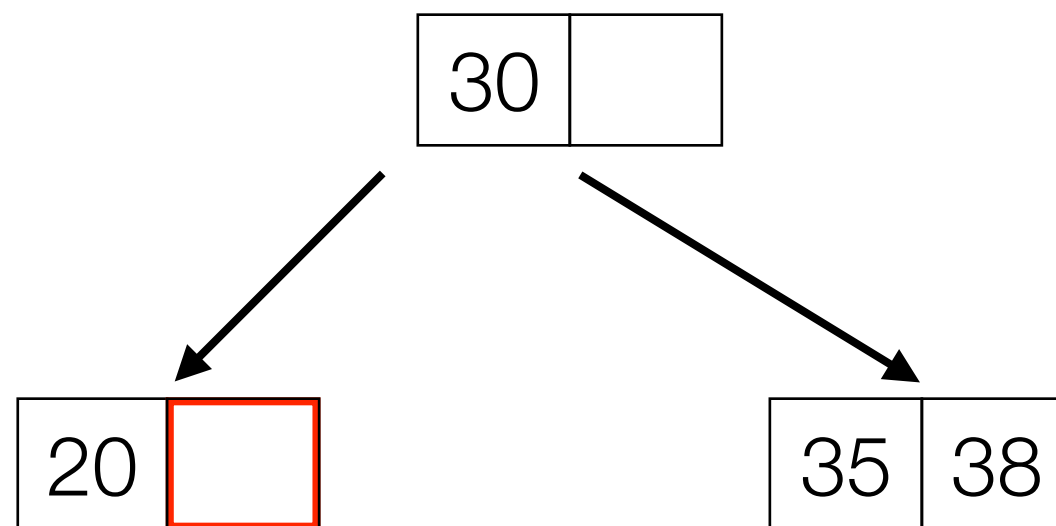
Árvores B: Exemplos de Inserção

- Insira 25:
- Caso 2: Encontre **recursivamente** o nó folha onde será feita a inserção e a posição da inserção por busca no nó. Se o nó acomoda o registro ($r < b$), então insira o registro e retorne *falso*.



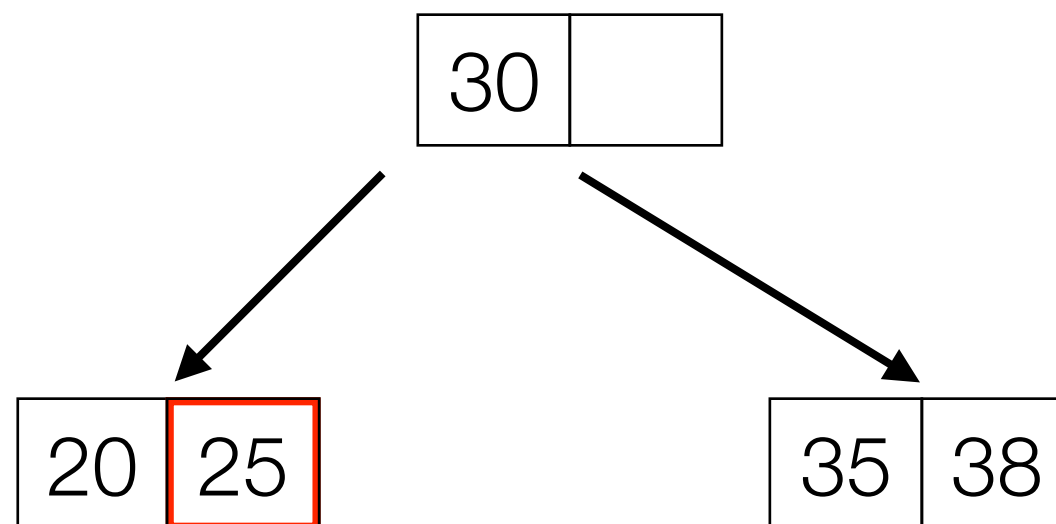
Árvores B: Exemplos de Inserção

- Insira 25:
- Caso 2: Encontre **recursivamente** o nó folha onde será feita a inserção e a posição da inserção por busca no nó. Se o nó acomoda o registro ($r < b$), então insira o registro e retorne *falso*.



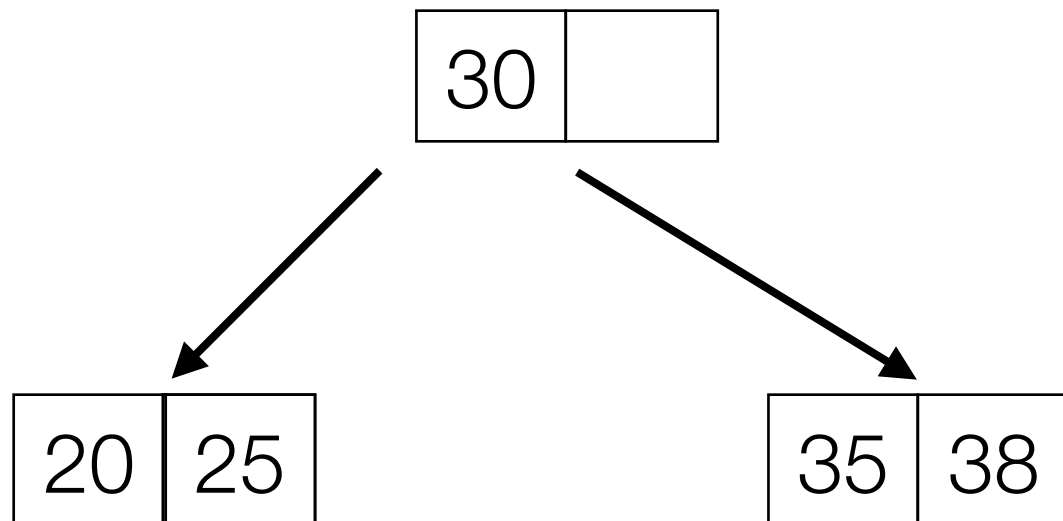
Árvores B: Exemplos de Inserção

- Insira 25:
- Caso 2: Encontre **recursivamente** o nó folha onde será feita a inserção e a posição da inserção por busca no nó. Se o nó acomoda o registro ($r < b$), então insira o registro e retorne *falso*.



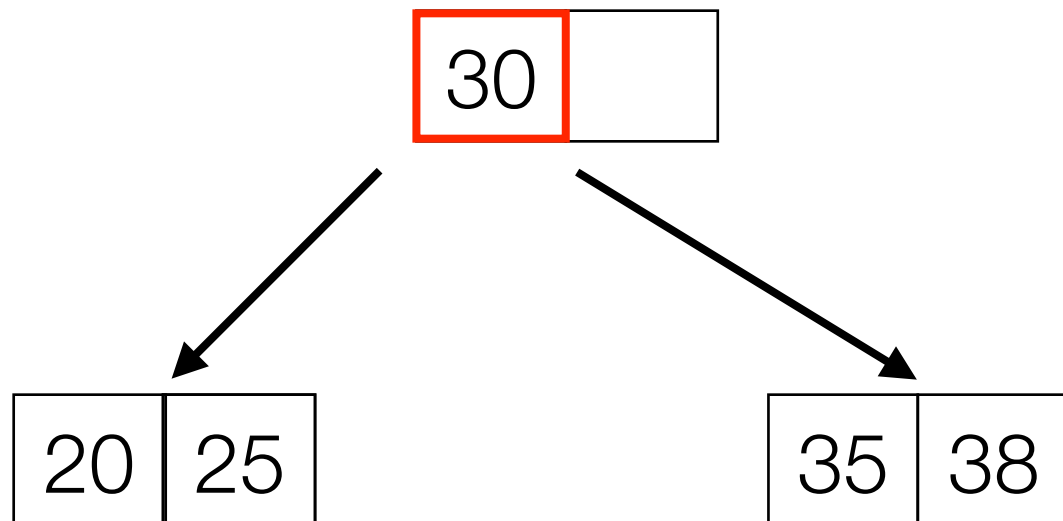
Árvores B: Exemplos de Inserção

- Insira 28:
- Caso 3: o o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



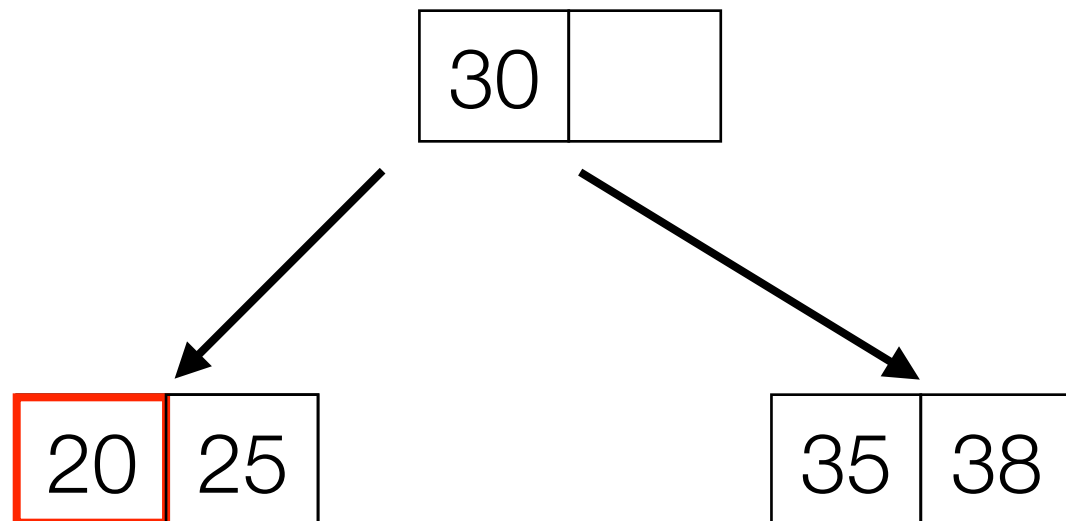
Árvores B: Exemplos de Inserção

- Insira 28:
- Caso 3: o o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



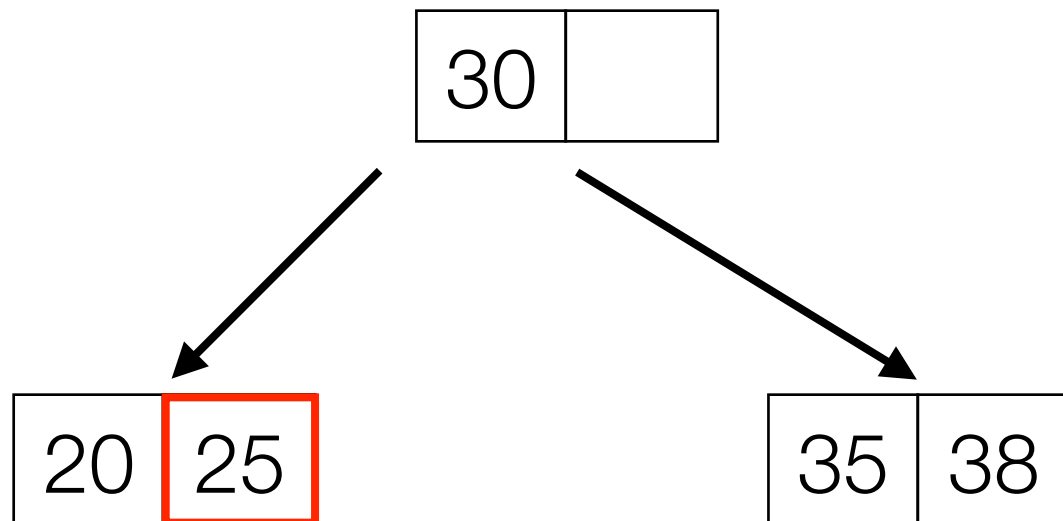
Árvores B: Exemplos de Inserção

- Insira 28:
- Caso 3: o o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



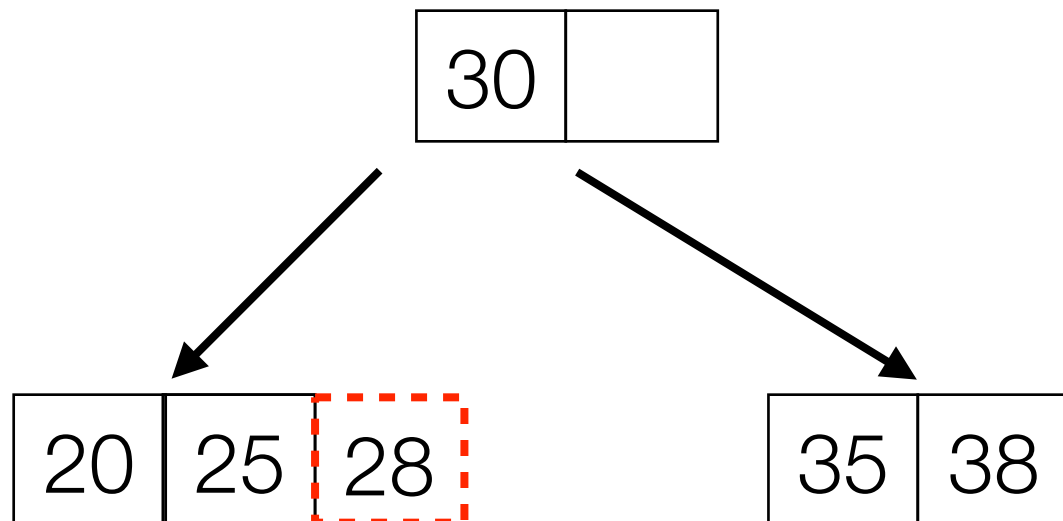
Árvores B: Exemplos de Inserção

- Insira 28:
- Caso 3: o o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



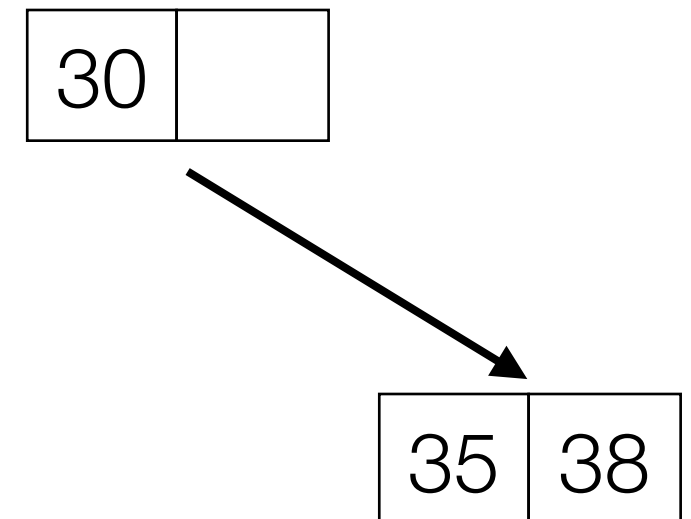
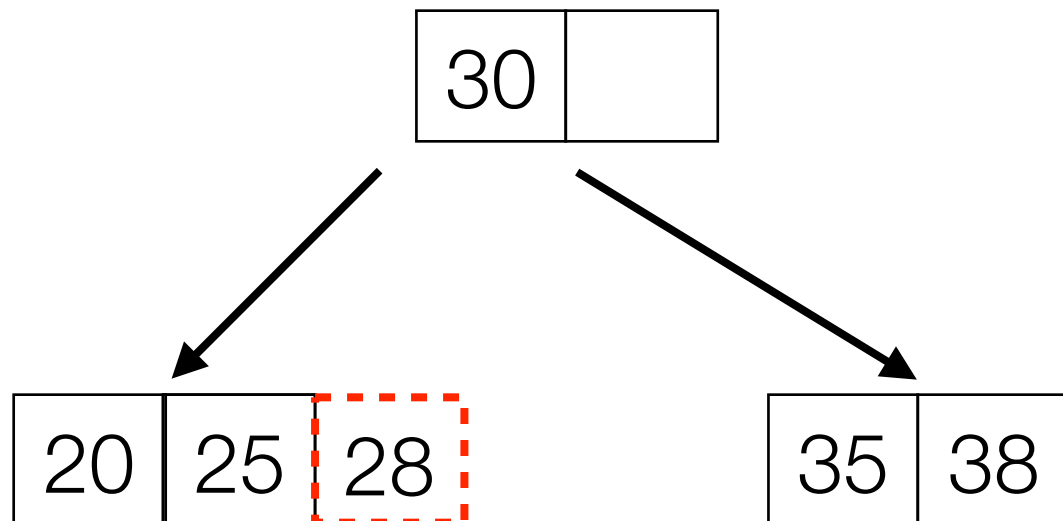
Árvores B: Exemplos de Inserção

- Insira 28:
- Caso 3: o o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



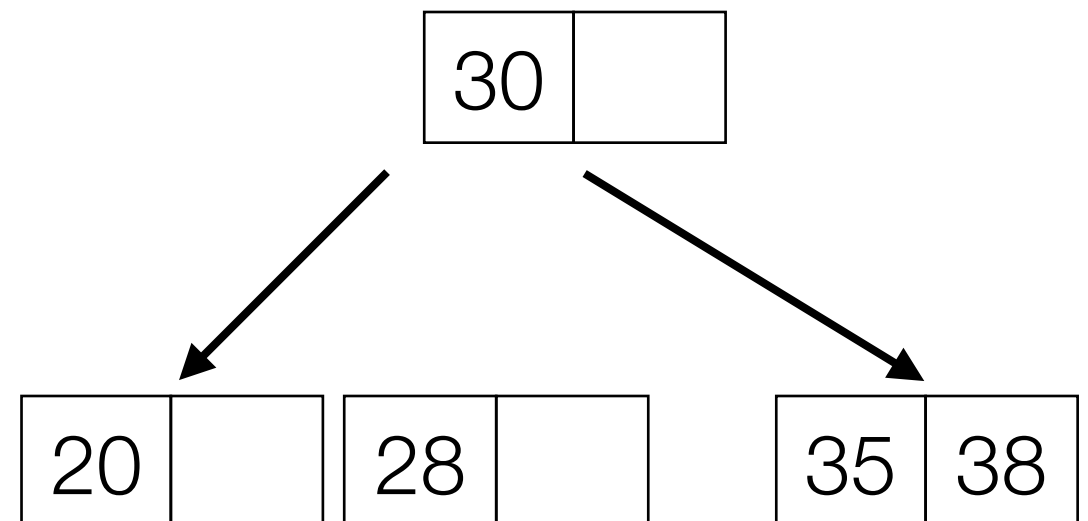
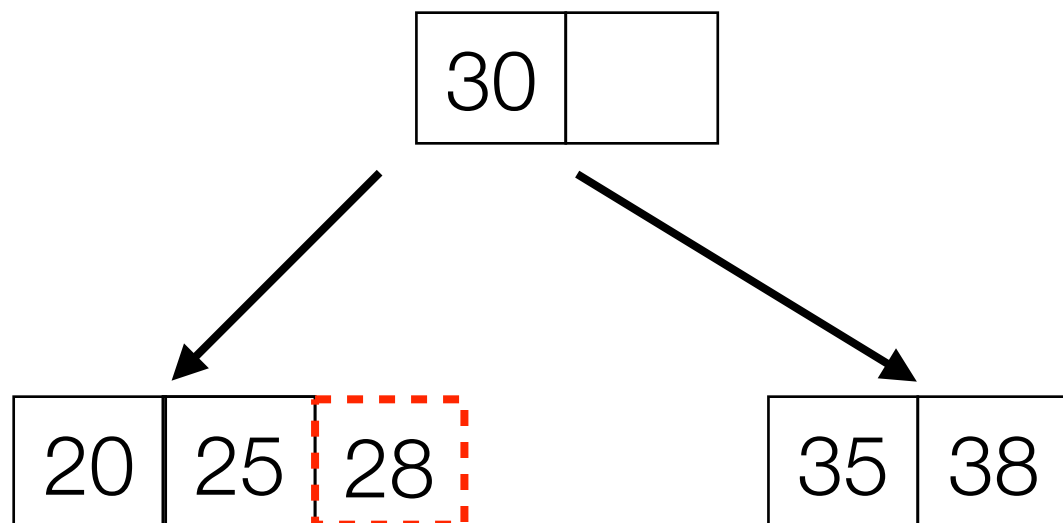
Árvores B: Exemplos de Inserção

- Insira 28:
- Caso 3: o o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



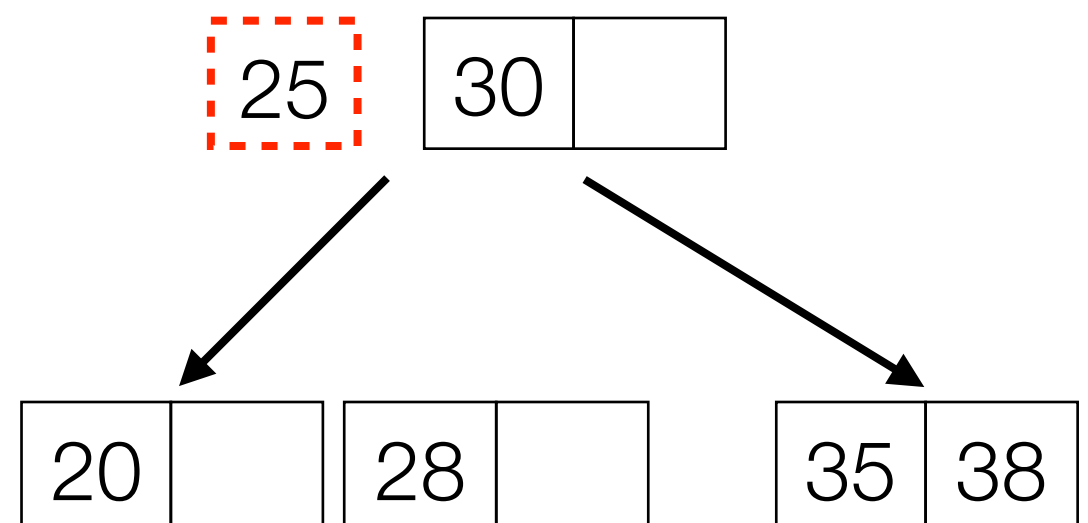
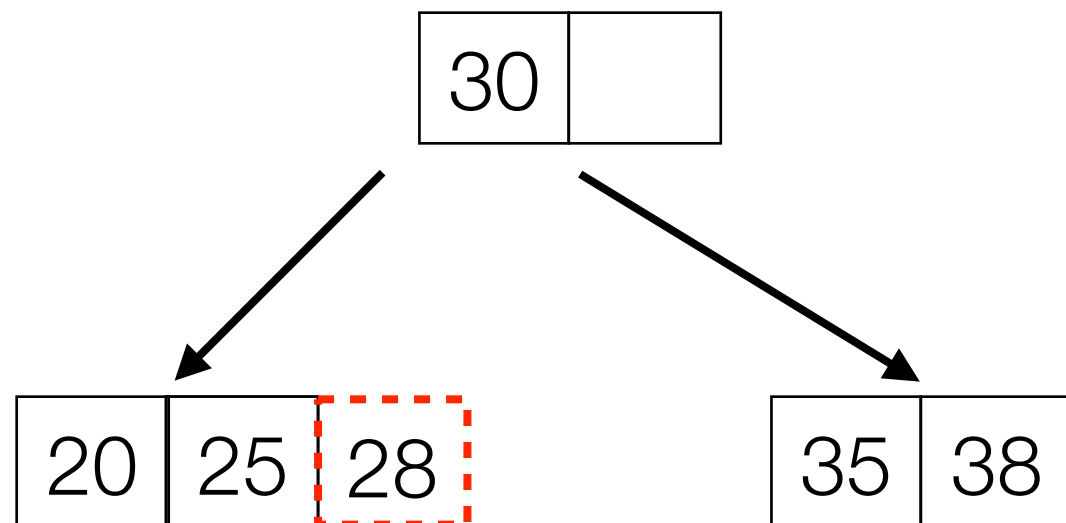
Árvores B: Exemplos de Inserção

- Insira 28:
- Caso 3: o o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



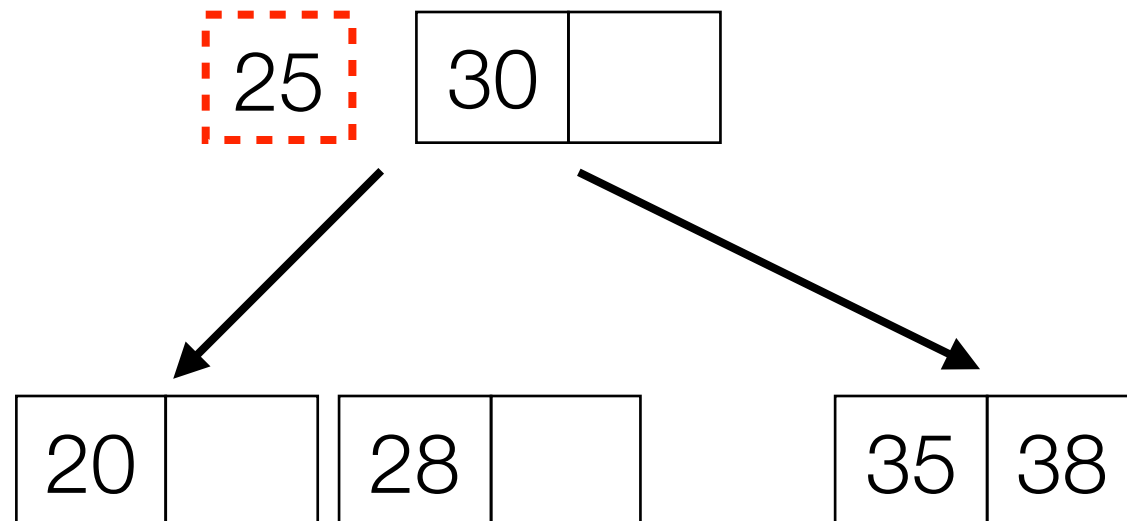
Árvores B: Exemplos de Inserção

- Insira 28:
- Caso 3: o o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



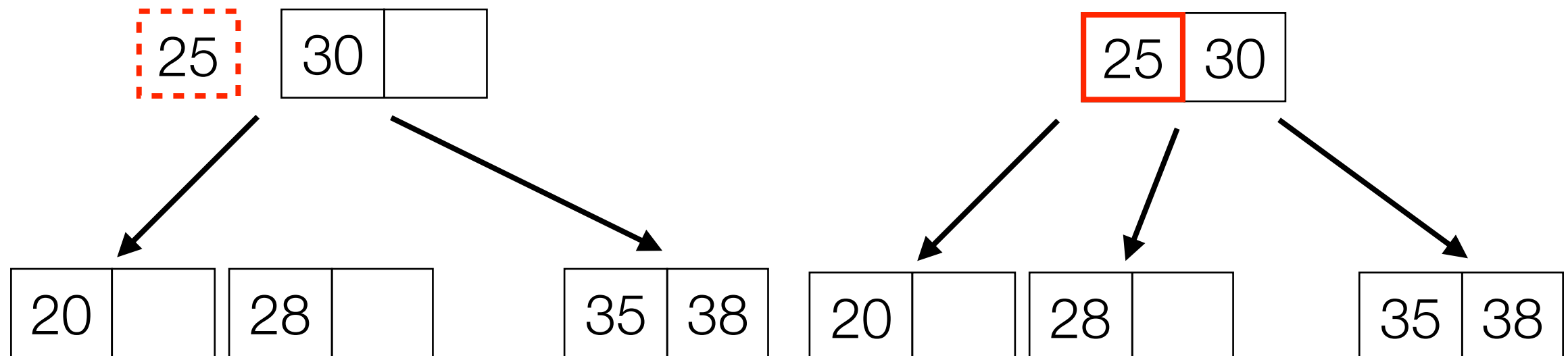
Árvores B: Exemplos de Inserção

- Insira 25 no nó pai:
- Volta recursiva da inserção.



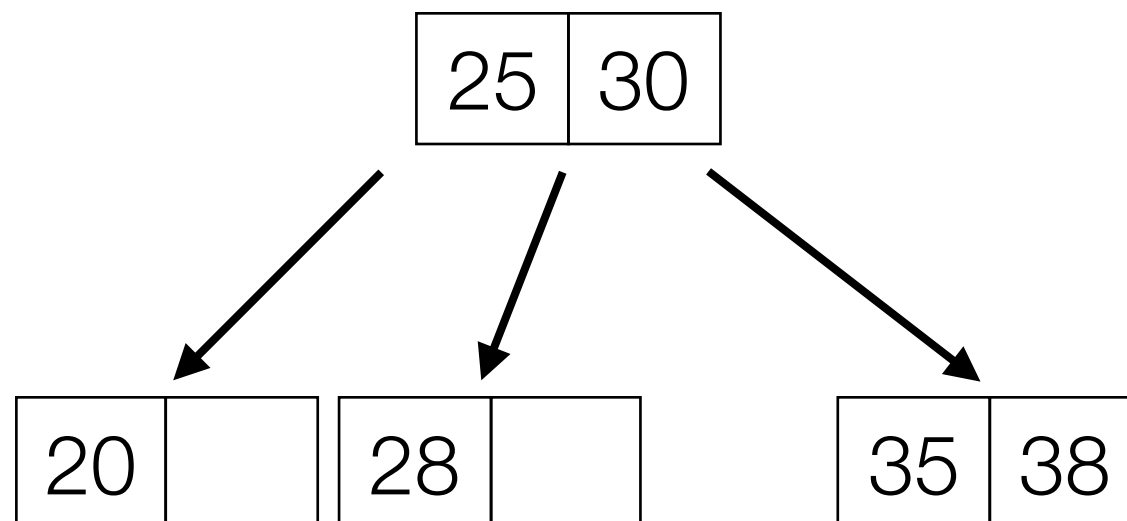
Árvores B: Exemplos de Inserção

- Insira 25 no nó pai:
- Volta recursiva da inserção.



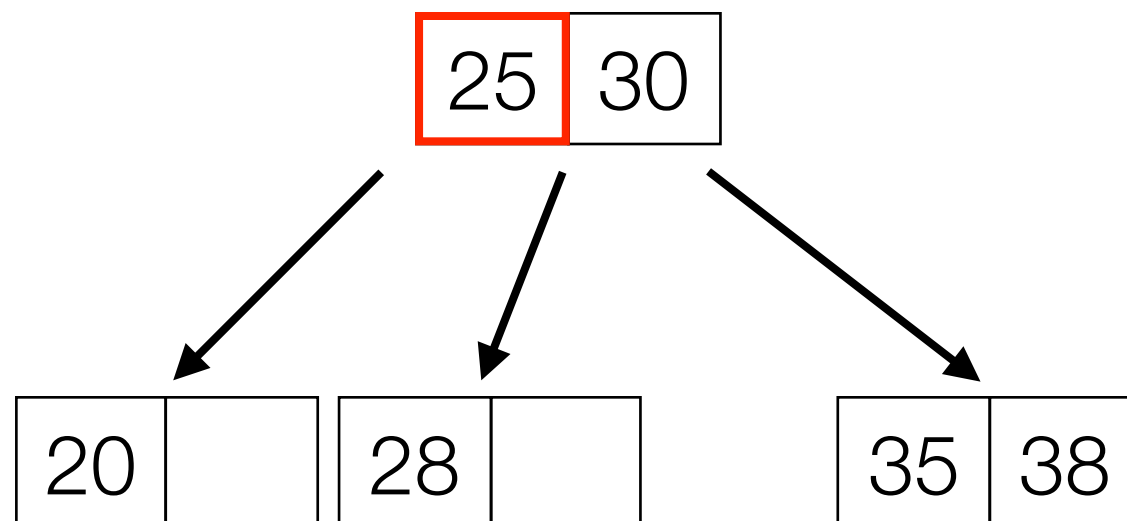
Árvores B: Exemplos de Inserção

- Insira 32:
- Caso contrário, o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



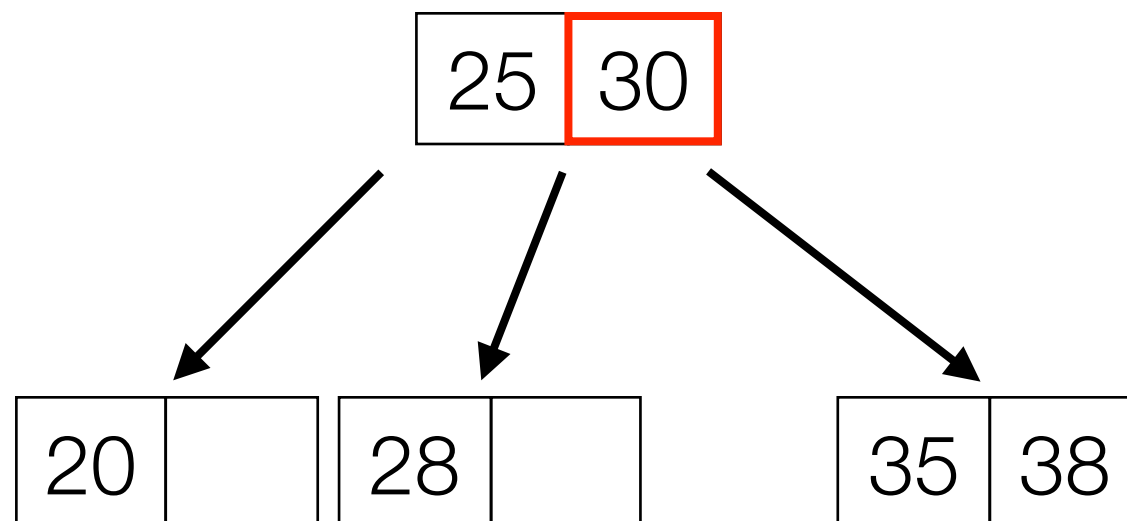
Árvores B: Exemplos de Inserção

- Insira 32:
- Caso contrário, o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



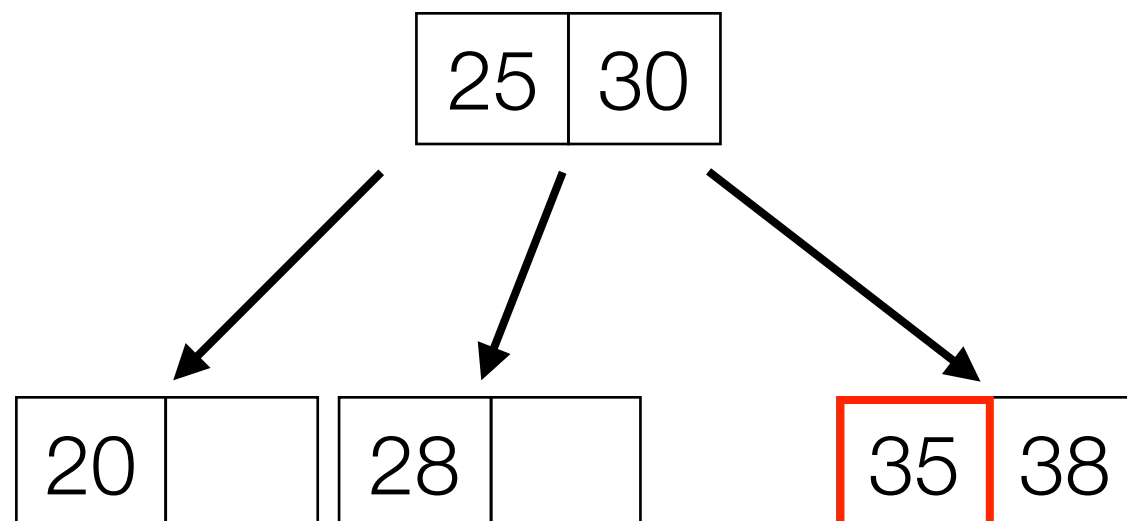
Árvores B: Exemplos de Inserção

- Insira 32:
- Caso contrário, o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



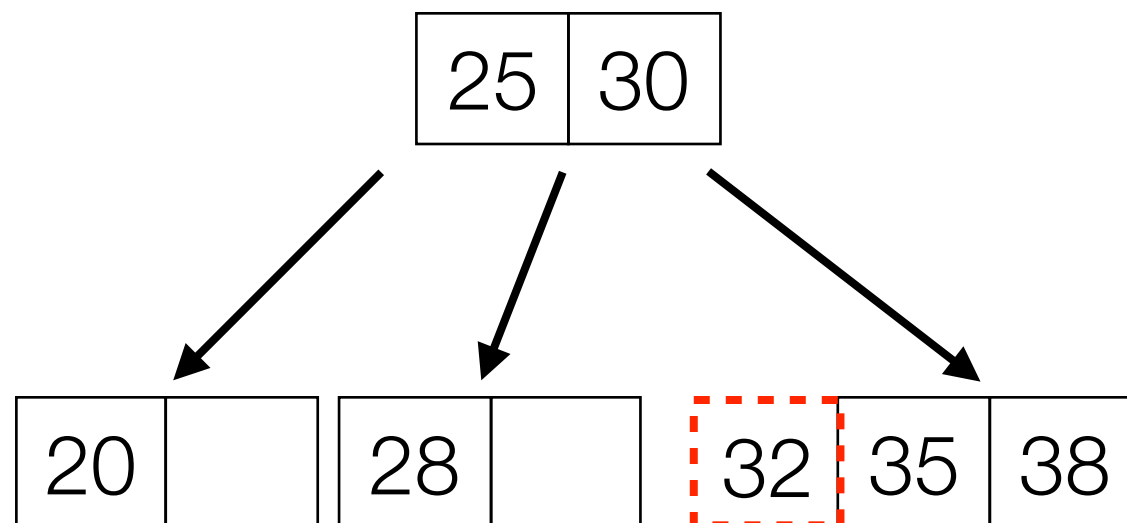
Árvores B: Exemplos de Inserção

- Insira 32:
- Caso contrário, o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



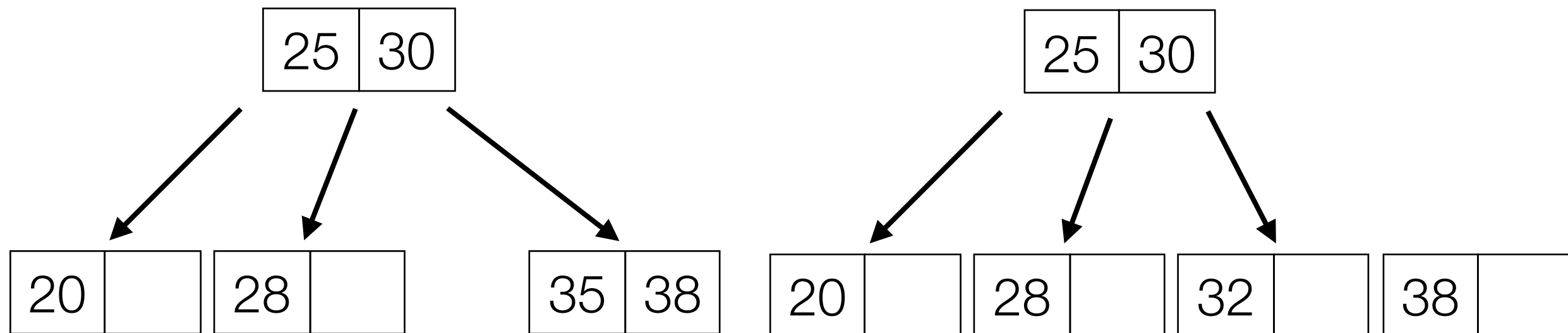
Árvores B: Exemplos de Inserção

- Insira 32:
- Caso contrário, o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



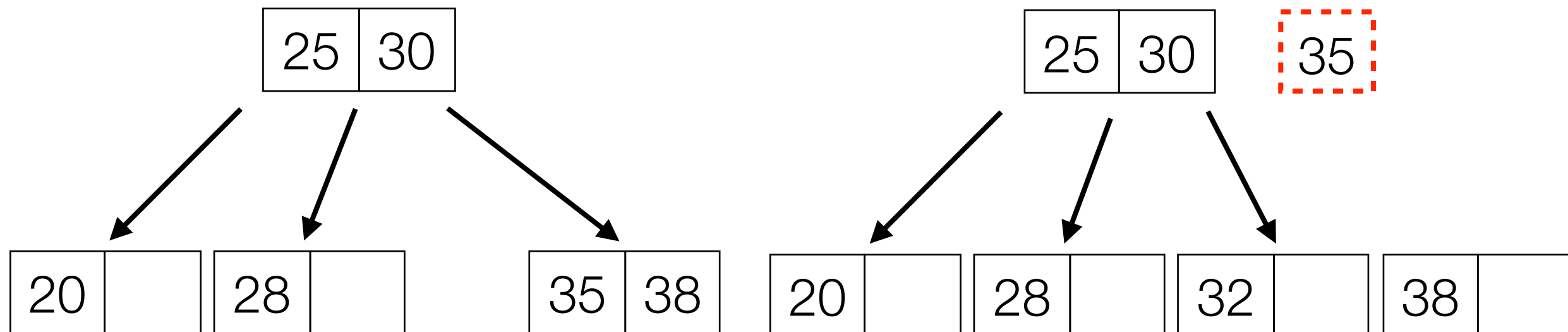
Árvores B: Exemplos de Inserção

- Insira 32:
- Caso contrário, o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.



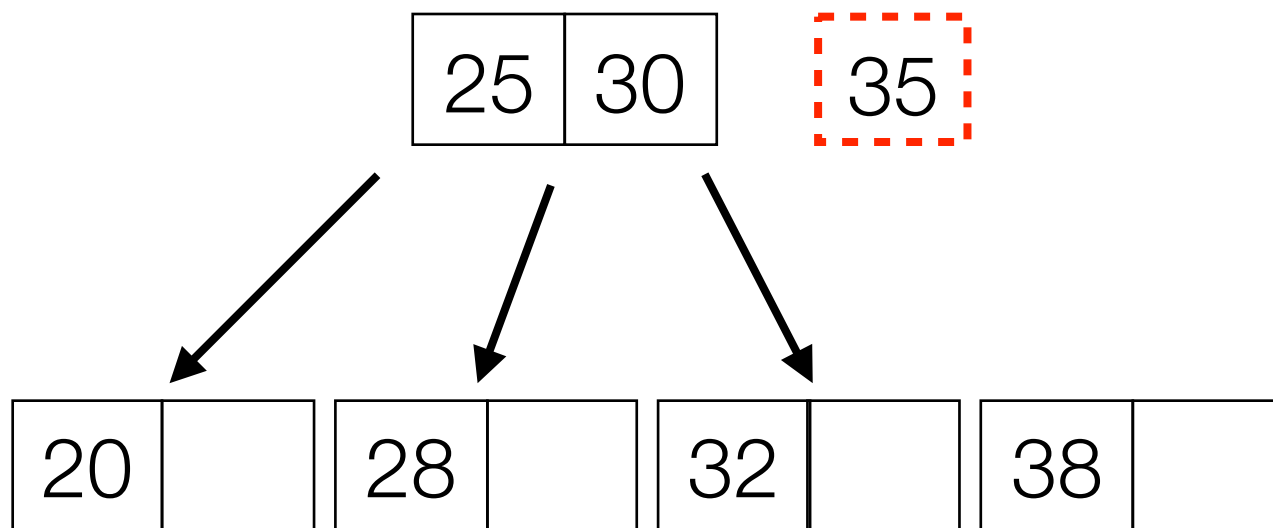
Árvores B: Exemplos de Inserção

- Insira 32:
- Caso contrário, o nó folha não acomoda o novo registro e divide-se em dois após colocá-lo no local candidato. Em seguida, o registro com chave mediana $\text{info}[m]$, onde $m = (b+1)/2$, deve ser inserido no nó pai e a função retorna *verdadeiro*.
- Insere 35 recursivamente no nó pai



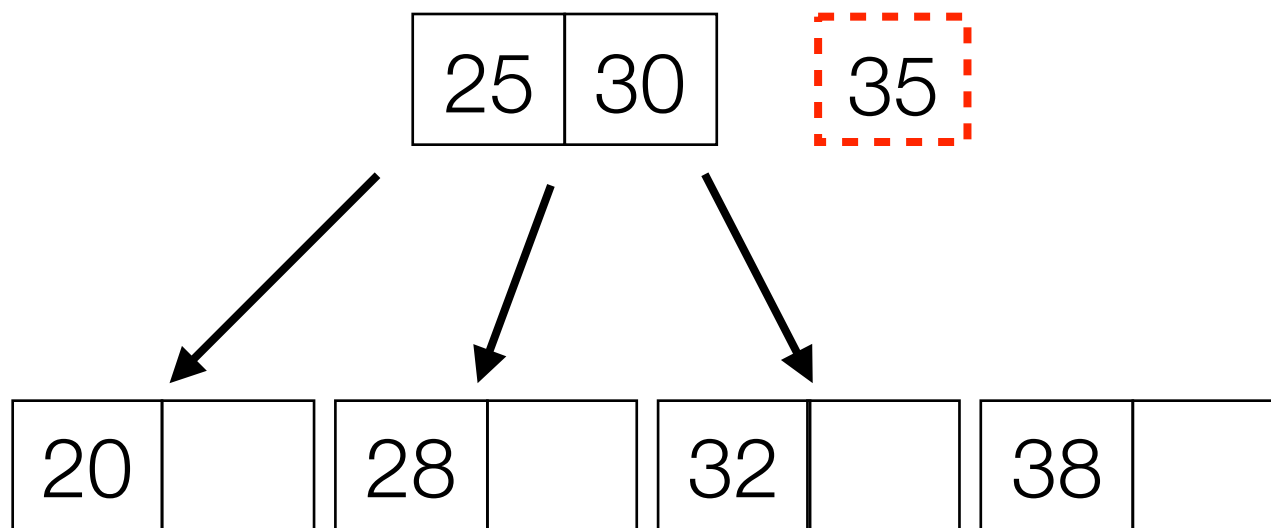
Árvores B: Exemplos de Inserção

- Insere 35 recursivamente no nó pai



Árvores B: Exemplos de Inserção

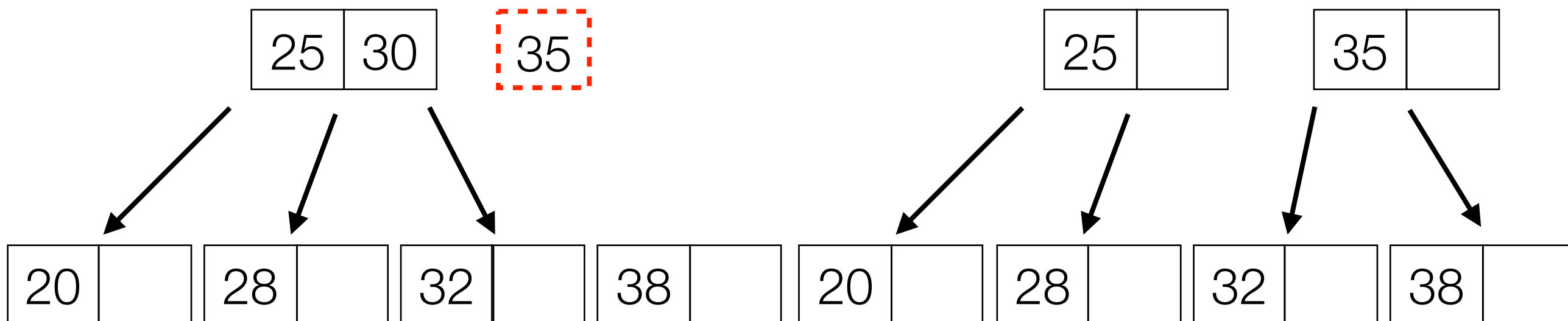
- Insere 35 recursivamente no nó pai
- Caso 3: nó pai (raiz) cheio



Árvores B: Exemplos de Inserção

- Insere 35 recursivamente no nó pai
- Caso 3: nó pai (raiz) cheio
- Caso especial: quando a raiz está cheia ela é quebrada em duas e a chave mediana (30) vira uma nova raiz

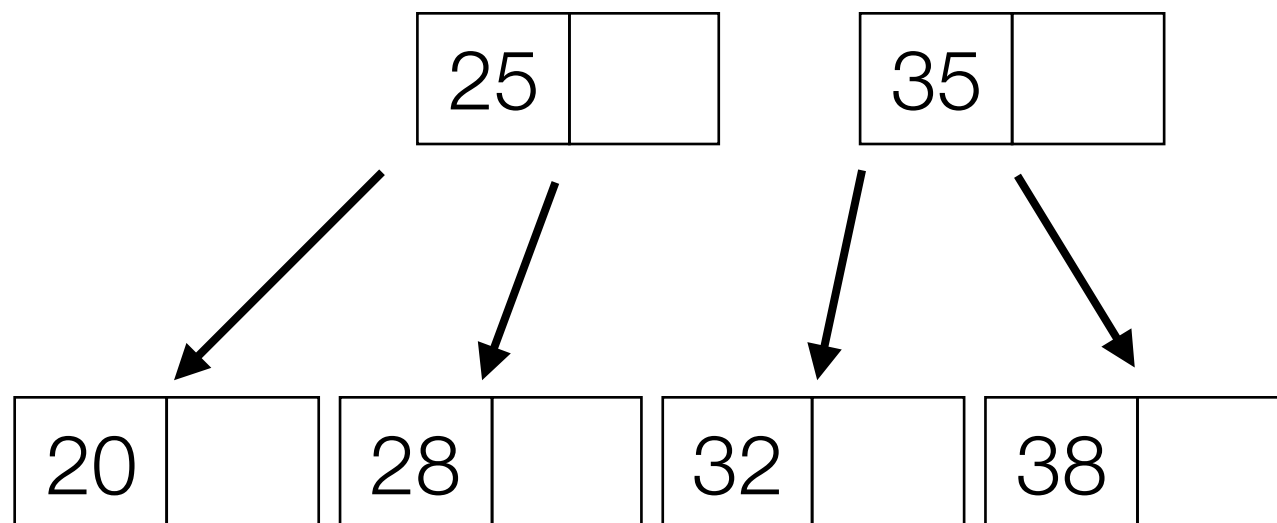
30



Árvores B: Exemplos de Inserção

- Caso especial: quando a raiz está cheia ela é quebrada em duas e a chave mediana (30) vira uma nova raiz

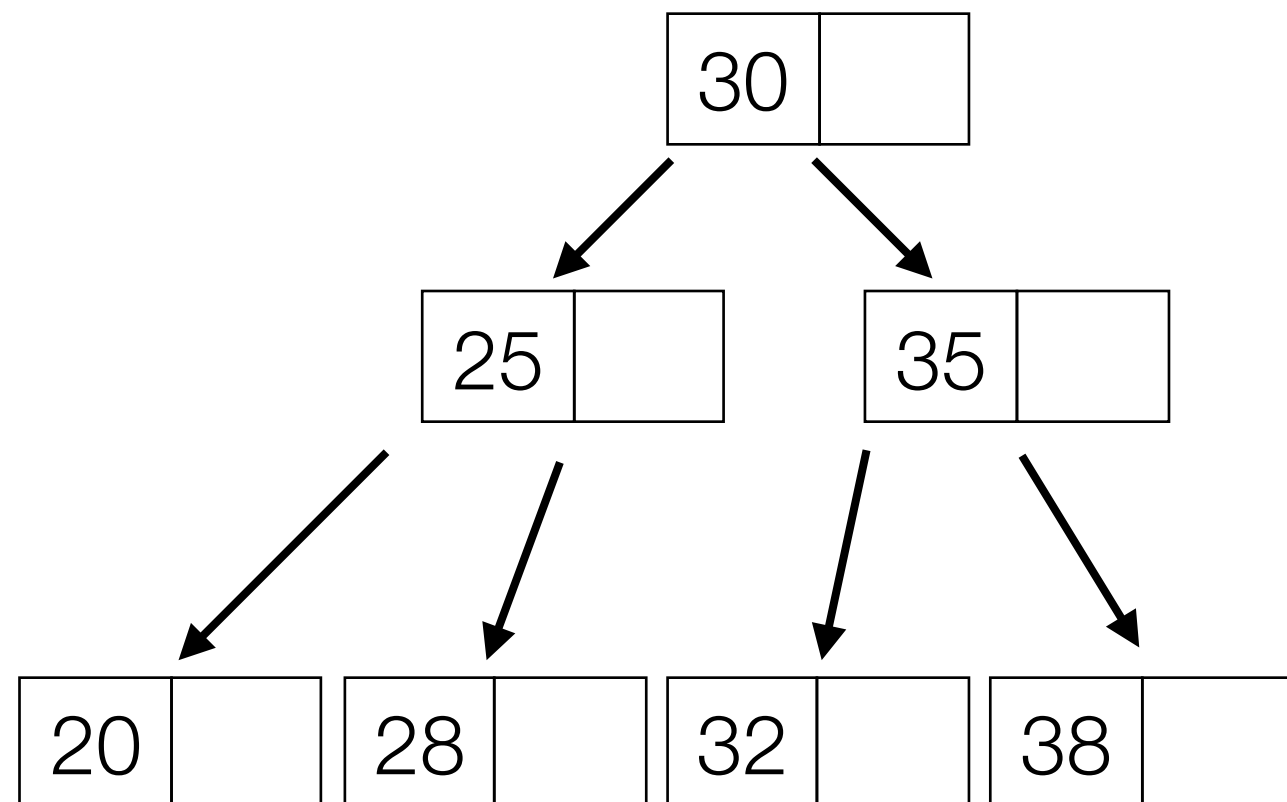
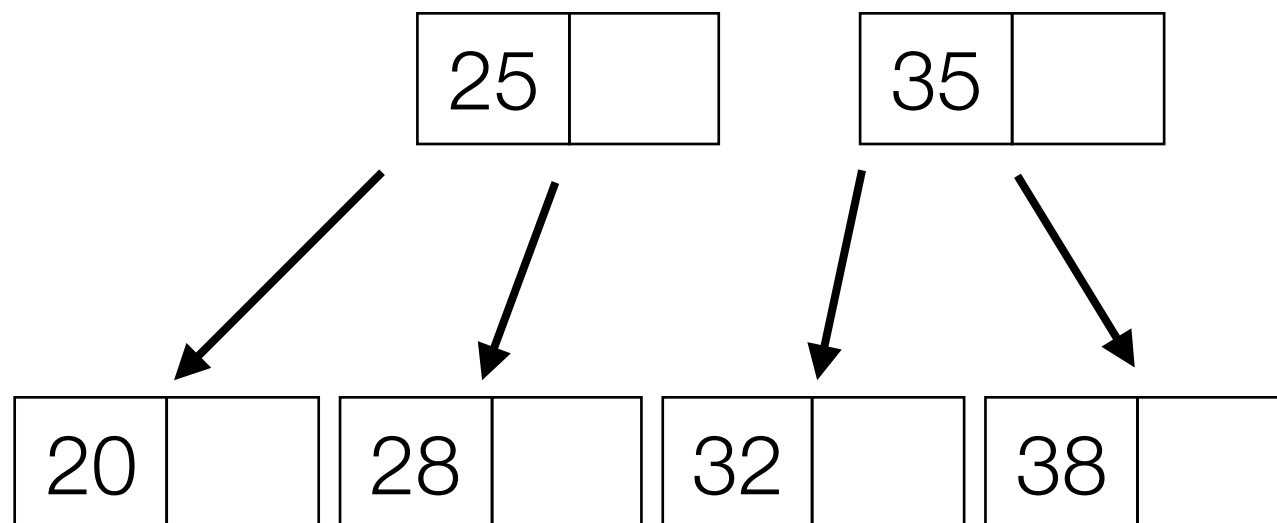
30



Árvores B: Exemplos de Inserção

- Caso especial: quando a raiz está cheia ela é quebrada em duas e a chave mediana (30) vira uma nova raiz
- Árvore cresce de altura

30



Árvores B: Implementação da Inserção

```
int ArvBInsererec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        } else {
            if (ArvBInsererec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
                /* Insere recursivamente o elemento no nó apropriado */
                if (*quebrou) {
                    /* O nó filho foi quebrado, então precisamos inserir a chave
                     do filho no lugar de i (retornada em *valor) e verificar
                     se o nó atual (pai) deve ser quebrado também */

                    InserirNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                    if (!VerificaOverflow(*arvore)) {
                        (*quebrou) = 0;
                        /* Verifica se houve estouro de folha */
                    } else {
                        /* Arvore arrebentada.*/

                        (*valor) = TrataOverflow(arvore, aux);

                        (*quebrou) = 1;
                    }
                }
            }
            return 1;
        }
    }
    return 0;
}
```

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        } else {
            if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
                /* Insere recursivamente o elemento no nó apropriado */
                if (*quebrou) {
                    /* O nó filho foi quebrado, então precisamos inserir a chave
                    do filho no lugar de i (retornada em *valor) e verificar
                    se o nó atual (pai) deve ser quebrado também */

                    InserirNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                    if (!VerificaOverflow(*arvore)) {
                        (*quebrou) = 0;
                        /* Verifica se houve estouro de folha */
                    } else {
                        /* Arvore arrebitada */

                        (*valor) = TrataOverflow(arvore, aux);

                        (*quebrou) = 1;
                    }
                }
            }
            return 1;
        }
        return 0;
    }
}

```

da Inserção

- Insira *valor = 25

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        } else {
            if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
                /* Insere recursivamente o elemento no nó apropriado */
                if (*quebrou) {
                    /* O nó filho foi quebrado, então precisamos inserir a chave
                     do filho no lugar de i (retornada em *valor) e verificar
                     se o nó atual (pai) deve ser quebrado também */

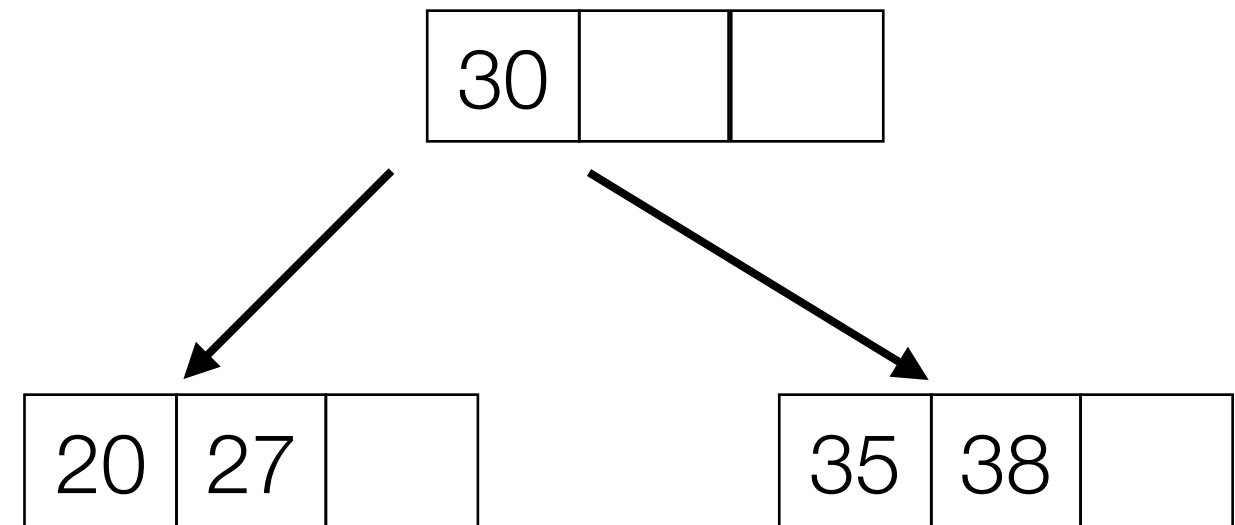
                    InserirNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                    if (!VerificaOverflow(*arvore)) {
                        (*quebrou) = 0;
                        /* Verifica se houve estouro de folha */
                    } else {
                        /* Arvore arrebitada.*/

                        (*valor) = TrataOverflow(arvore, aux);

                        (*quebrou) = 1;
                    }
                }
            }
            return 1;
        }
        return 0;
    }
}

```



- Insira *valor = 25

```
int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    ...

```

```

if (*arvore==NULL) {
    (*quebrou) = 1;
    (*aux) = NULL;
    return 1;
} else {
    /* Elemento encontrado na arvore, ignora inserção */
    (*quebrou) = 0;
    return 0;
} else {
    if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
        /* Insere recursivamente o elemento no nó apropriado */
        if (*quebrou) {
            /* O nó filho foi quebrado, então precisamos inserir a chave
            do filho no lugar de i (retornada em *valor) e verificar
            se o nó atual (pai) deve ser quebrado também */

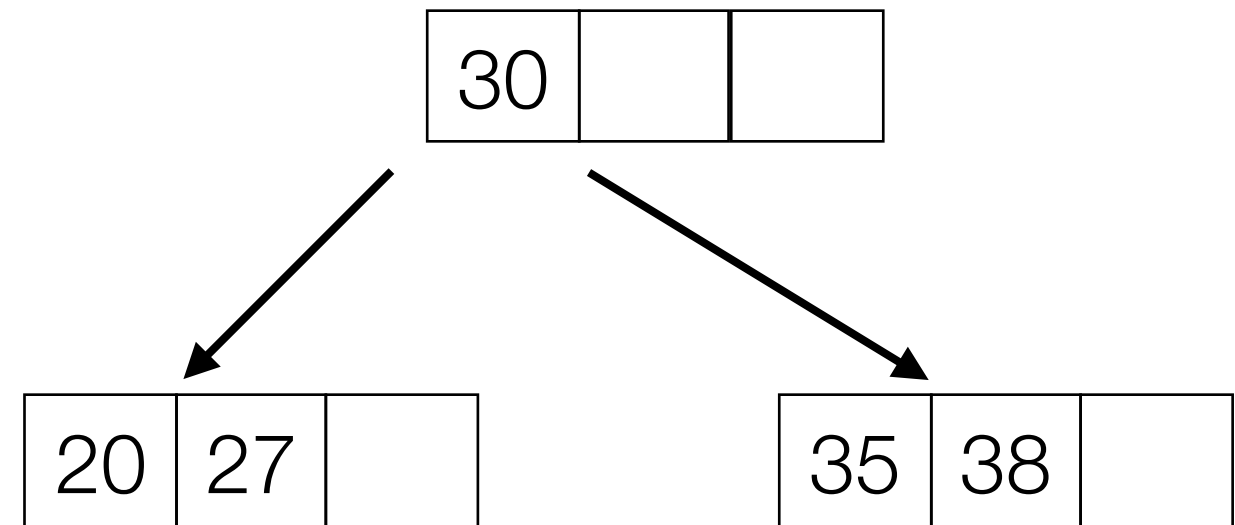
            InsereNovoValorEFilhoNo(*arvore, *aux, *valor, i);

            if (!VerificaOverflow(*arvore)) {
                (*quebrou) = 0;
                /* Verifica se houve estouro de folha */
            } else {
                /* Arvore arrebitada */

                (*valor) = TrataOverflow(arvore, aux);

                (*quebrou) = 1;
            }
        }
        return 1;
    }
    return 0;
}
}
}

```



- Insira *valor = 25

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    }

    if(BuscaChaveNo(*arvore, *valor, &i)) {

        (*quebrou) = 0;
        return 0;
    } else {
        if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
            /* Insere recursivamente o elemento no nó apropriado */
            if (*quebrou) {
                /* O nó filho foi quebrado, então precisamos inserir a chave
                do filho no lugar de i (retornada em *valor) e verificar
                se o nó atual (pai) deve ser quebrado também */

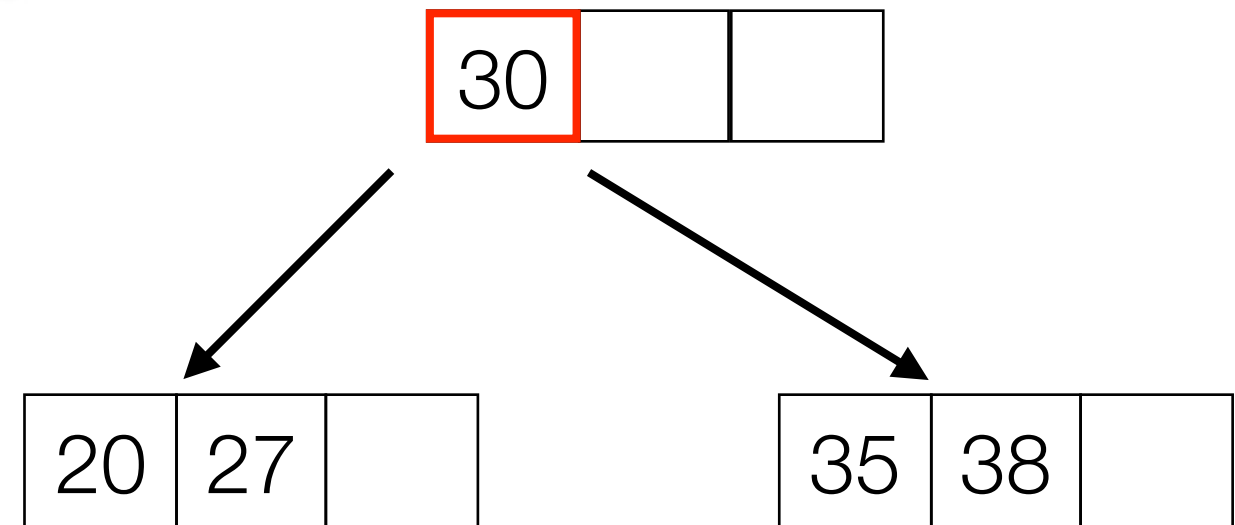
                InserirNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                if (!VerificaOverflow(*arvore)) {
                    (*quebrou) = 0;
                    /* Verifica se houve estouro de folha */
                } else {
                    /* Arvore arrebitada */

                    (*valor) = TrataOverflow(arvore, aux);

                    (*quebrou) = 1;
                }
            }
            return 1;
        }
        return 0;
    }
}

```



- Insira *valor = 25

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        }
        if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
            /* Insere recursivamente o elemento no nó apropriado */

            /* O nó filho foi quebrado, então precisamos inserir a chave
            do filho no lugar de i (retornada em *valor) e verificar
            se o nó atual (pai) deve ser quebrado também */

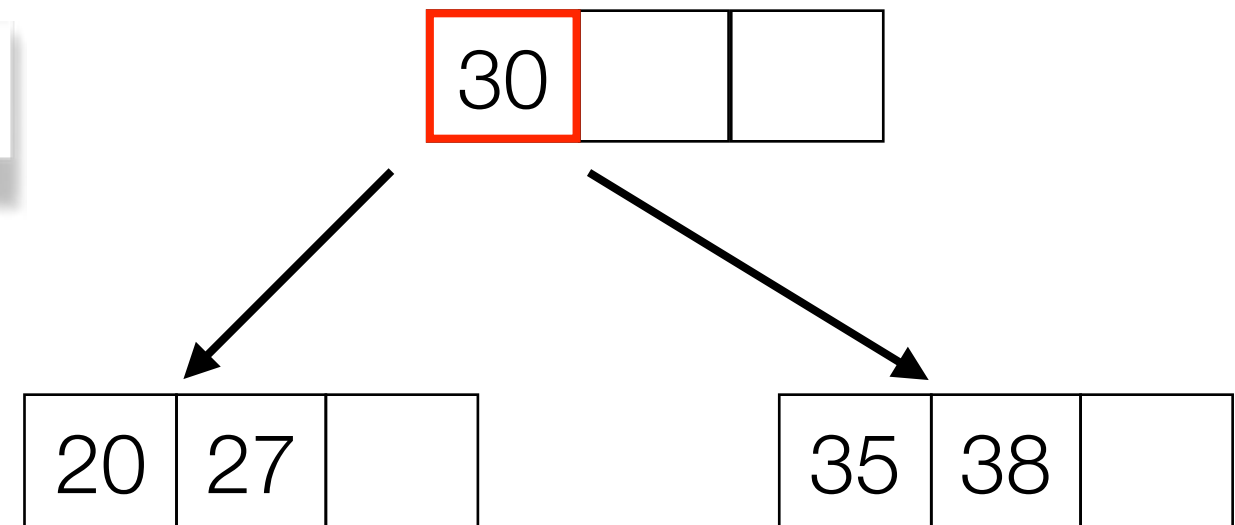
            InsereNovoValorEFilhoNo(*arvore, *aux, *valor, i);

            if (!VerificaOverflow(*arvore)) {
                (*quebrou) = 0;
                /* Verifica se houve estouro de folha */
            } else {
                /* Arvore arreventada.*/

                (*valor) = TrataOverflow(arvore, aux);

                (*quebrou) = 1;
            }
        }
        return 1;
    }
    return 0;
}

```



- Insira *valor = 25

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    }

    if(BuscaChaveNo(*arvore, *valor, &i)) {

        (*quebrou) = 0;
        return 0;
    } else {
        if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
            /* Insere recursivamente o elemento no nó apropriado */
            if (*quebrou) {
                /* O nó filho foi quebrado, então precisamos inserir a chave
                do filho no lugar de i (retornada em *valor) e verificar
                se o nó atual (pai) deve ser quebrado também */

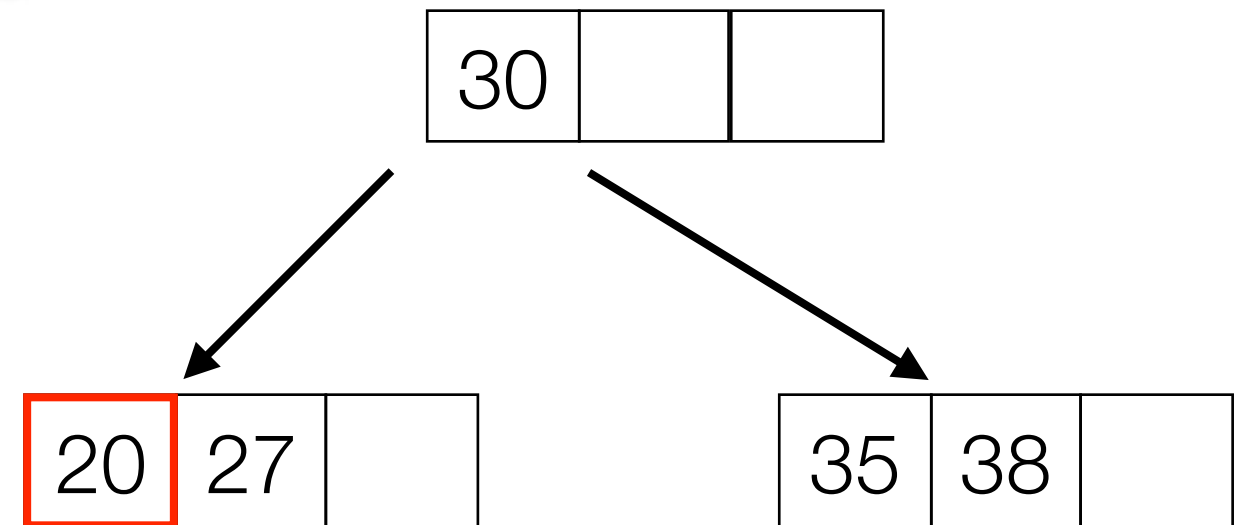
                InserirNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                if (!VerificaOverflow(*arvore)) {
                    (*quebrou) = 0;
                    /* Verifica se houve estouro de folha */
                } else {
                    /* Arvore arrebitada */

                    (*valor) = TrataOverflow(arvore, aux);

                    (*quebrou) = 1;
                }
            }
            return 1;
        }
        return 0;
    }
}

```



- Insira *valor = 25

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    }

    if(BuscaChaveNo(*arvore, *valor, &i)) {

        (*quebrou) = 0;
        return 0;
    } else {
        if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
            /* Insere recursivamente o elemento no nó apropriado */
            if (*quebrou) {
                /* O nó filho foi quebrado, então precisamos inserir a chave
                do filho no lugar de i (retornada em *valor) e verificar
                se o nó atual (pai) deve ser quebrado também */

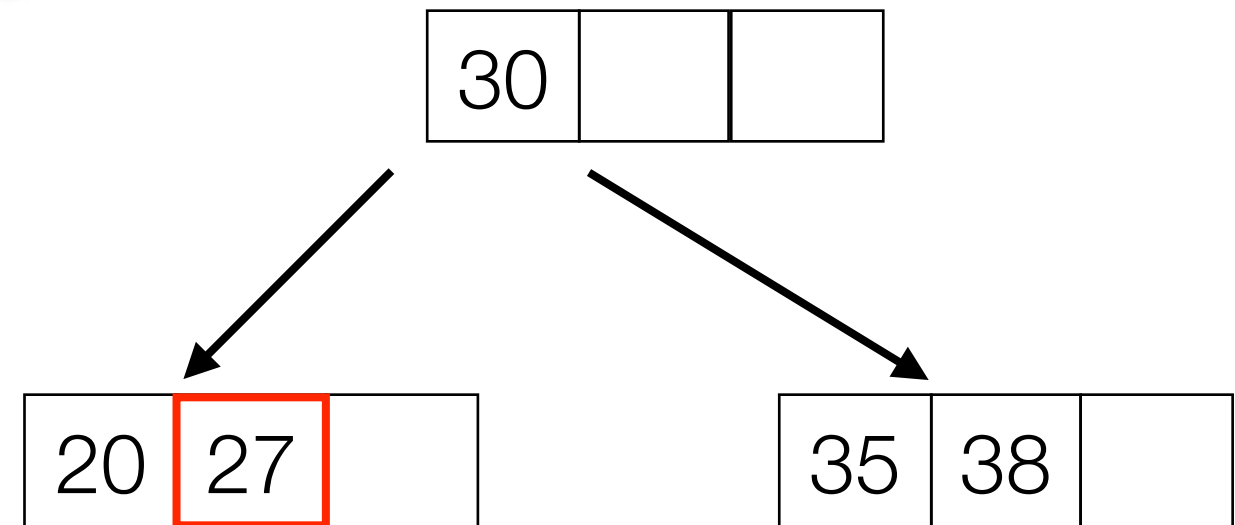
                InsereNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                if (!VerificaOverflow(*arvore)) {
                    (*quebrou) = 0;
                    /* Verifica se houve estouro de folha */
                } else {
                    /* Arvore arrebitada */

                    (*valor) = TrataOverflow(arvore, aux);

                    (*quebrou) = 1;
                }
            }
            return 1;
        }
        return 0;
    }
}

```



- Insira *valor = 25

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        }
        if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
            /* Insere recursivamente o elemento no nó apropriado */

            /* O nó filho foi quebrado, então precisamos inserir a chave
            do filho no lugar de i (retornada em *valor) e verificar
            se o nó atual (pai) deve ser quebrado também */

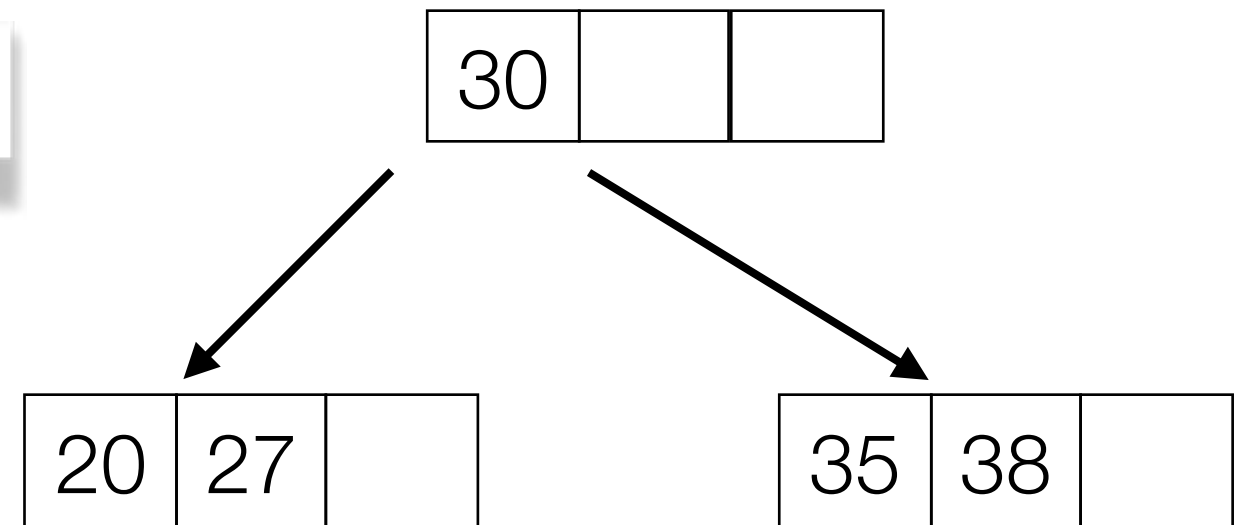
            InsereNovoValorEFilhoNo(*arvore, *aux, *valor, i);

            if (!VerificaOverflow(*arvore)) {
                (*quebrou) = 0;
                /* Verifica se houve estouro de folha */
            } else {
                /* Arvore arrebitada */

                (*valor) = TrataOverflow(arvore, aux);

                (*quebrou) = 1;
            }
        }
        return 1;
    }
    return 0;
}

```



- Insira *valor = 25

```
int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    ...
}
```

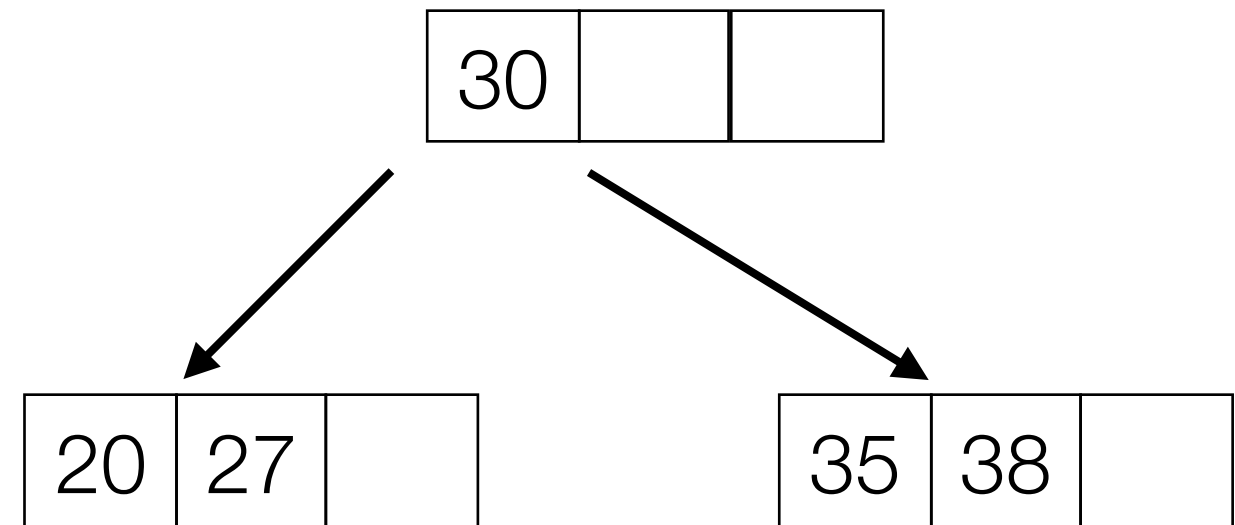
```
if (*arvore==NULL) {
    (*quebrou) = 1;
    (*aux) = NULL;
    return 1;
} else {
    /* Elemento encontrado na arvore, ignora inserção */
    (*quebrou) = 0;
    return 0;
} else {
    if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
        /* Insere recursivamente o elemento no nó apropriado */
        if (*quebrou) {
            /* O nó filho foi quebrado, então precisamos inserir a chave
            do filho no lugar de i (retornada em *valor) e verificar
            se o nó atual (pai) deve ser quebrado também */

            InsereNovoValorEFilhoNo(*arvore, *aux, *valor, i);

            if (!VerificaOverflow(*arvore)) {
                (*quebrou) = 0;
                /* Verifica se houve estouro de folha */
            } else {
                /* Arvore arrebitada.*/

                (*valor) = TrataOverflow(arvore, aux);

                (*quebrou) = 1;
            }
        }
        return 1;
    }
    return 0;
}
}
```



- Insira *valor = 25

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        } else {
            if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
                /* Insere recursivamente o elemento no nó apropriado */
                /* 0 nó filho foi quebrado, então precisamos inserir a chave
                 do filho no lugar de i (retornada em *valor) e verificar
                 se o nó atual (pai) deve ser quebrado também */

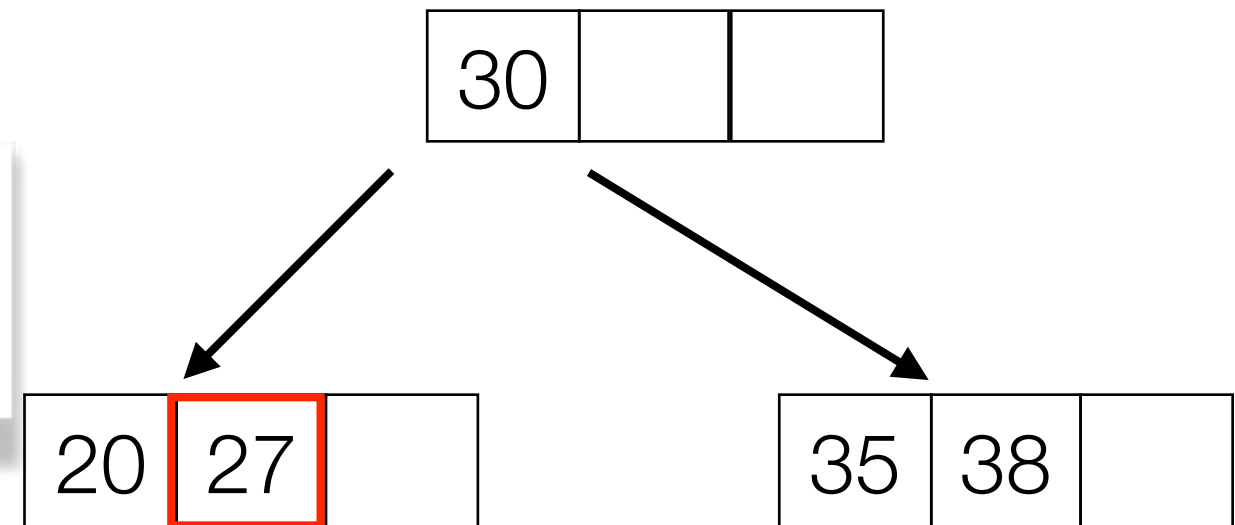
                InserirNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                (*quebrou) = 0;
                /* Verifica se houve estouro de folha */
            } else {
                /* Arvore arrebitada.*/

                (*valor) = TrataOverflow(arvore, aux);

                (*quebrou) = 1;
            }
        }
        return 1;
    }
    return 0;
}

```



- Insira *valor = 25

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        } else {
            if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
                /* Insere recursivamente o elemento no nó apropriado */
                /* 0 nó filho foi quebrado, então precisamos inserir a chave
                do filho no lugar de i (retornada em *valor) e verificar
                se o nó atual (pai) deve ser quebrado também */

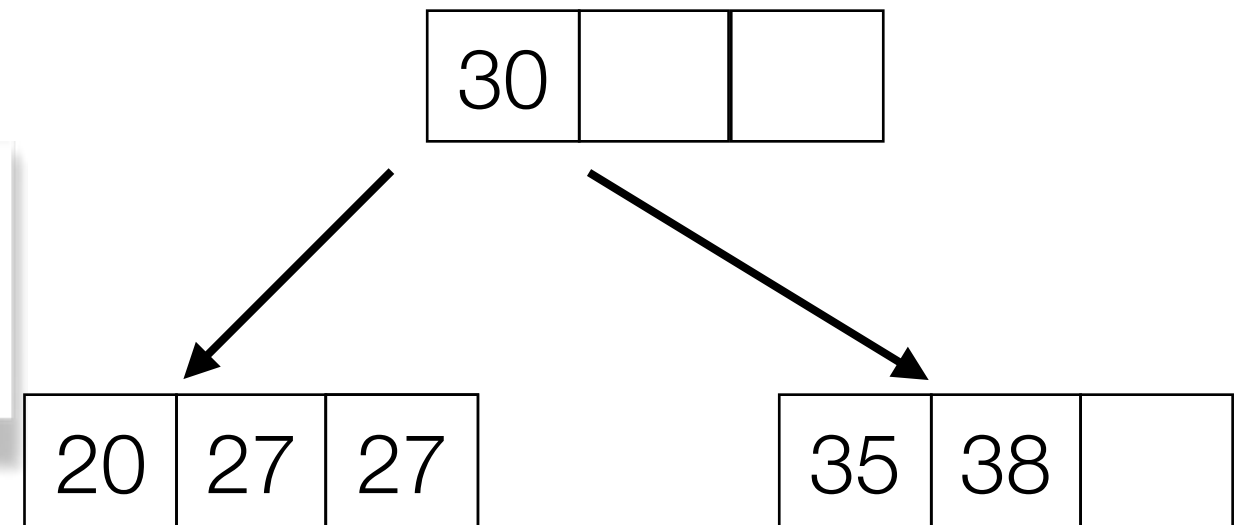
                InserirNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                (*quebrou) = 0;
                /* Verifica se houve estouro de folha */
            } else {
                /* Arvore arrebitada.*/

                (*valor) = TrataOverflow(arvore, aux);

                (*quebrou) = 1;
            }
        }
        return 1;
    }
    return 0;
}

```



- Insira *valor = 25

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        } else {
            if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
                /* Insere recursivamente o elemento no nó apropriado */
                /* 0 nó filho foi quebrado, então precisamos inserir a chave
                do filho no lugar de i (retornada em *valor) e verificar
                se o nó atual (pai) deve ser quebrado também */

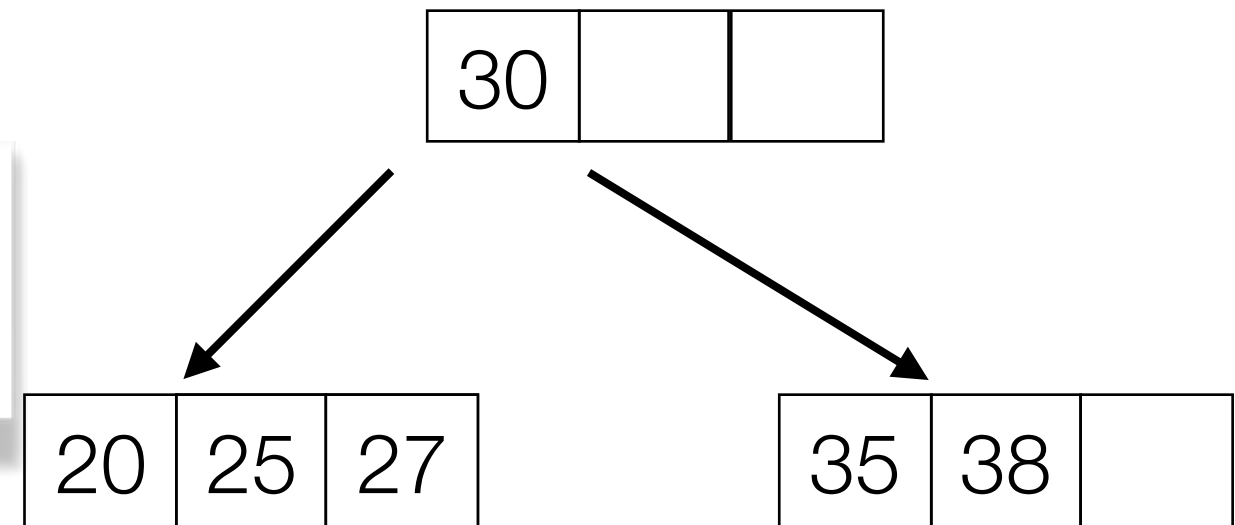
                InserirNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                (*quebrou) = 0;
                /* Verifica se houve estouro de folha */
            } else {
                /* Arvore arrebitada.*/

                (*valor) = TrataOverflow(arvore, aux);

                (*quebrou) = 1;
            }
        }
        return 1;
    }
    return 0;
}

```



- Insira *valor = 25

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        } else {
            if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
                /* Insere recursivamente o elemento no nó apropriado */
                /* 0 nó filho foi quebrado, então precisamos inserir a chave
                do filho no lugar de i (retornada em *valor) e verificar
                se o nó atual (pai) deve ser quebrado também */

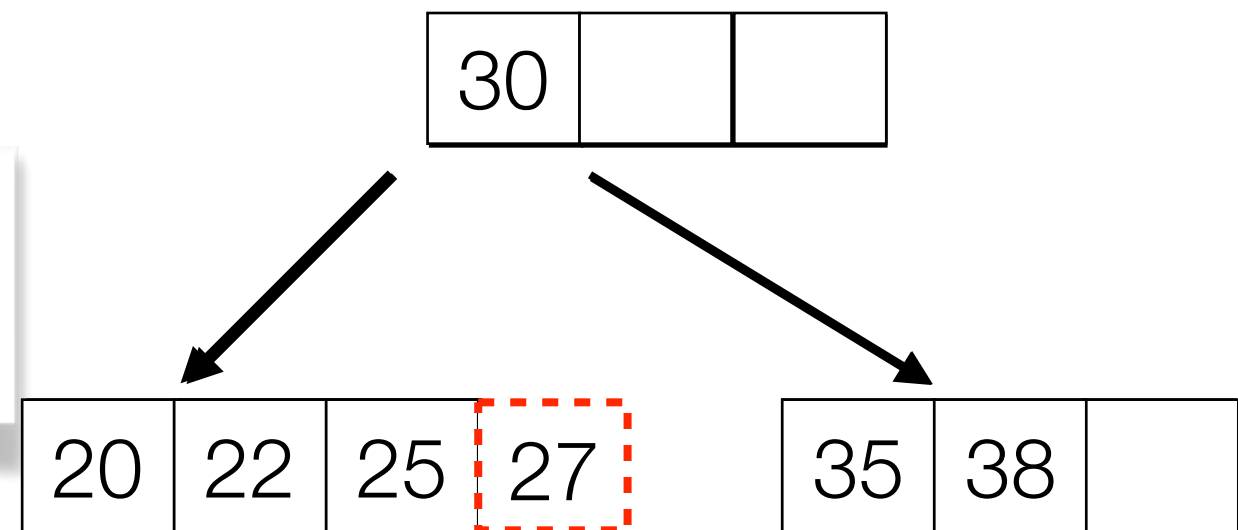
                InserirNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                (*quebrou) = 0;
                /* Verifica se houve estouro de folha */
            } else {
                /* Arvore arrebitada.*/

                (*valor) = TrataOverflow(arvore, aux);

                (*quebrou) = 1;
            }
        }
        return 1;
    }
    return 0;
}

```



- Insira *valor = 22

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        } else {
            if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
                /* Insere recursivamente o elemento no nó apropriado */
                if (*quebrou) {
                    /* O nó filho foi quebrado, então precisamos inserir a chave
                     do filho no lugar de i (retornada em *valor) e verificar
                     se o nó atual (pai) deve ser quebrado também */

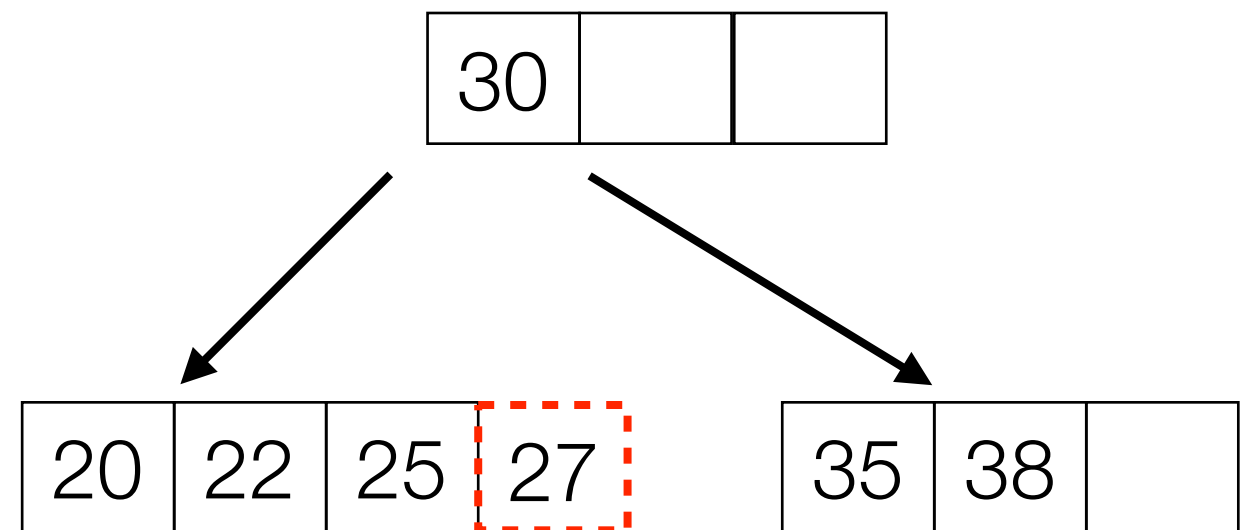
                    InsereNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                    if (!VerificaOverflow(*arvore)) {
                        (*quebrou) = 0;
                        /* Verifica se houve estouro de folha */
                    } else {
                        /* Arvore arreventada.*/

                        (*valor) = TrataOverflow(arvore, aux);

                        (*quebrou) = 1;
                    }
                }
                return 1;
            }
            return 0;
        }
    }
}

```



- Insira *valor = 22

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        } else {
            if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
                /* Insere recursivamente o elemento no nó apropriado */
                if (*quebrou) {
                    /* O nó filho foi quebrado, então precisamos inserir a chave
                     do filho no lugar de i (retornada em *valor) e verificar
                     se o nó atual (pai) deve ser quebrado também */

                    InserirNovoValorEFilhoNo(*arvore, *aux, *valor, i);

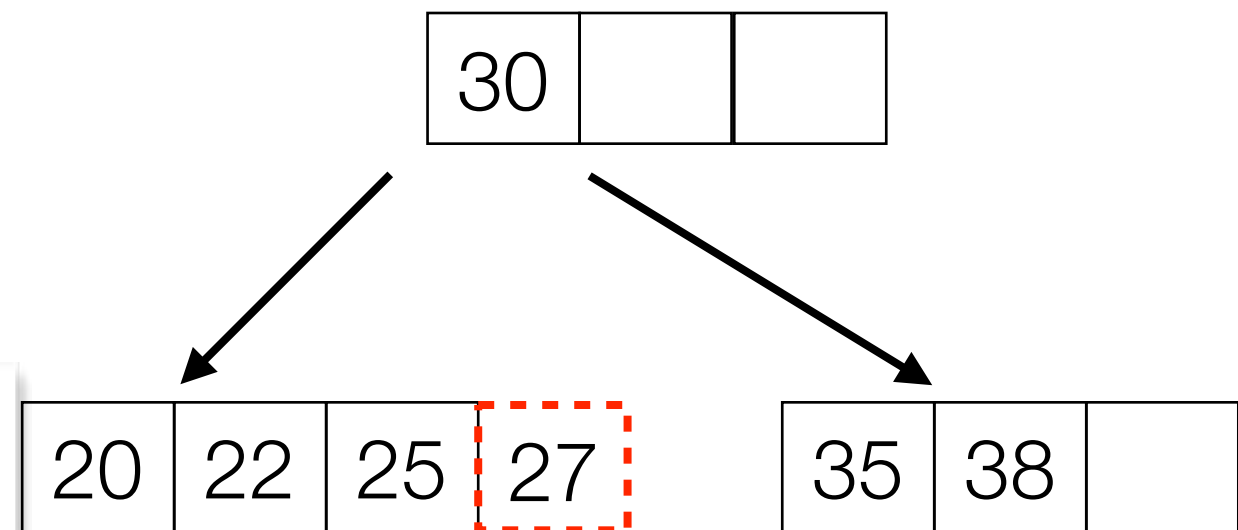
                    if (!VerificaOverflow(*arvore)) {
                        (*quebrou) = 0;
                        /* Verifica se houve estouro de folha */

                        /* Arvore arrebitada.*/

                        (*valor) = TrataOverflow(arvore, aux);

                        (*quebrou) = 1;
                    }
                }
            }
            return 1;
        }
        return 0;
    }
}

```



- Insira *valor = 22
- Recursão insere 22 no nó pai

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        } else {
            if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
                /* Insere recursivamente o elemento no nó apropriado */
                if (*quebrou) {
                    /* O nó filho foi quebrado, então precisamos inserir a chave
                    do filho no lugar de i (retornada em *valor) e verificar
                    se o nó atual (pai) deve ser quebrado também */

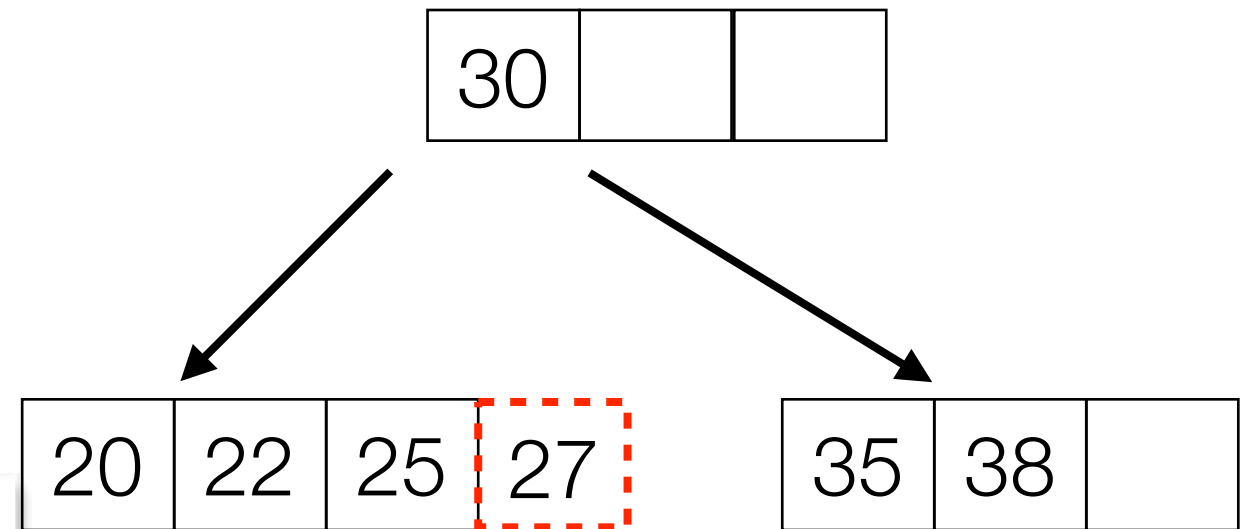
                    InserirNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                    if (!VerificaOverflow(*arvore)) {
                        (*quebrou) = 0;
                    }
                } else {
                    /* Arvore arrebitada.*/

                    (*valor) = TrataOverflow(arvore, aux);

                    (*quebrou) = 1;
                }
            }
            return 1;
        }
    }
    return 0;
}

```



- Insira *valor = 22
- Recursão insere 22 no nó pai

```

int ArvBInsereRec(ArvB **arvore, ArvB **aux, int *valor, int *quebrou)
{
    int i=0, j=0;
    if (*arvore==NULL) {
        (*quebrou) = 1;
        (*aux) = NULL;
        return 1;
    } else {
        if(BuscaChaveNo(*arvore, *valor, &i)) {
            /* Elemento encontrado na arvore, ignora inserção */
            (*quebrou) = 0;
            return 0;
        } else {
            if (ArvBInsereRec(&((*arvore)->filhos[i]),aux,valor,quebrou)) {
                /* Insere recursivamente o elemento no nó apropriado */
                if (*quebrou) {
                    /* O nó filho foi quebrado, então precisamos inserir a chave
                    do filho no lugar de i (retornada em *valor) e verificar
                    se o nó atual (pai) deve ser quebrado também */

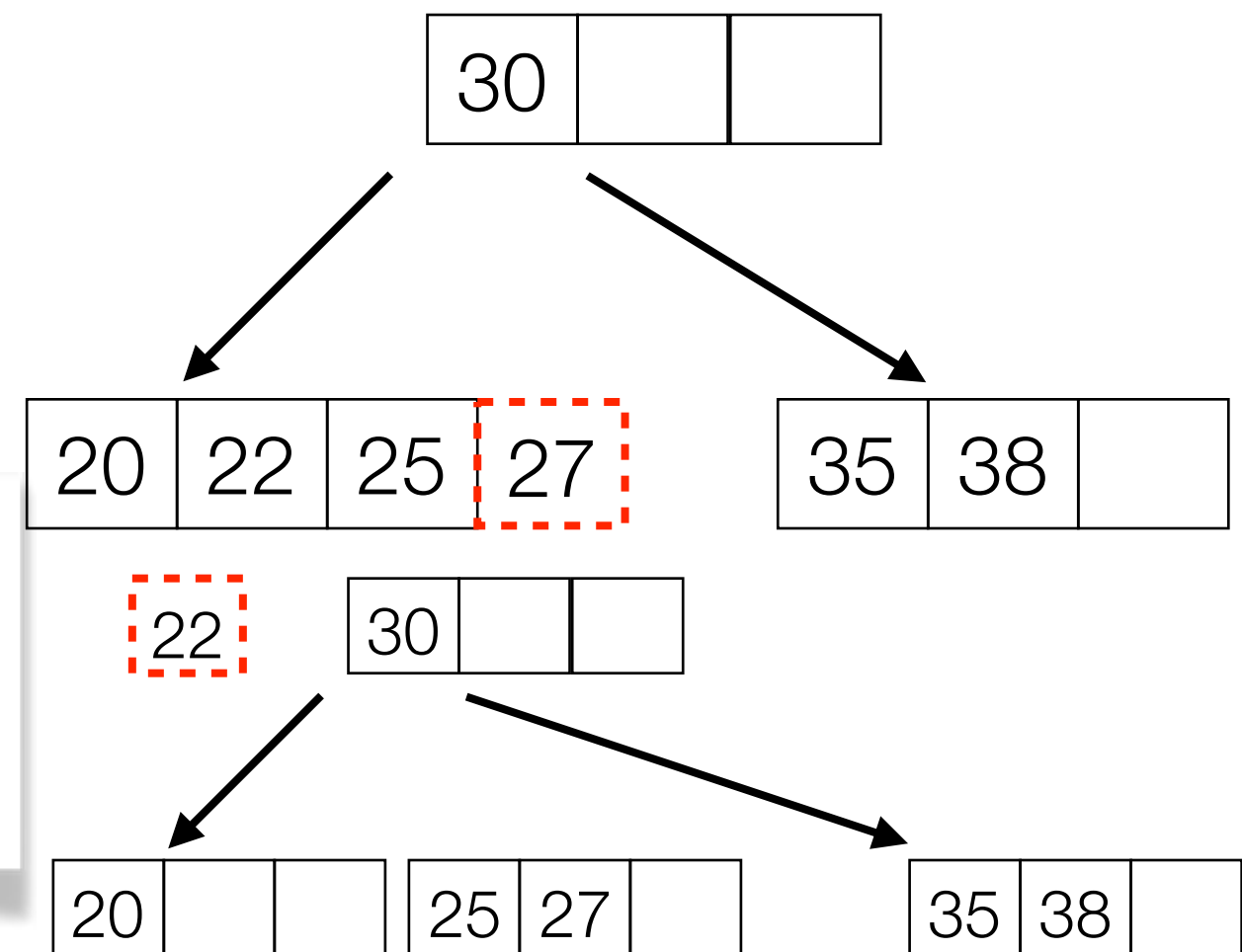
                    InserirNovoValorEFilhoNo(*arvore, *aux, *valor, i);

                    if (!VerificaOverflow(*arvore)) {
                        (*quebrou) = 0;
                    }
                } else {
                    /* Arvore arrebitada.*/

                    (*valor) = TrataOverflow(arvore, aux);

                    (*quebrou) = 1;
                }
            }
            return 1;
        }
    }
    return 0;
}

```



Árvores B: Remoção de Elementos

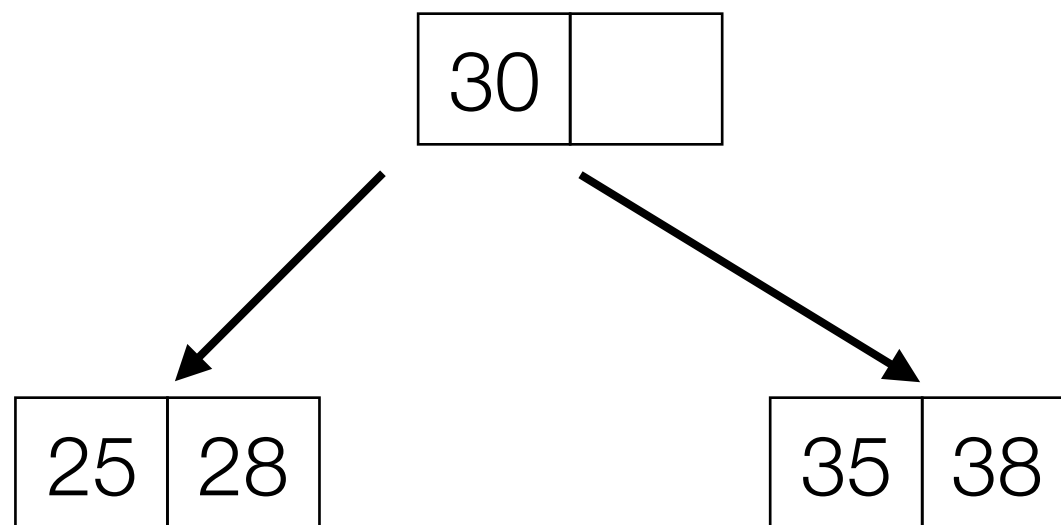
- Reduzida a remover elementos da folha
- Para remover registro em nó interno, ele é trocado por seu menor sucessor (na própria folha ou maior predecessor de outro nó)

Árvores B: Remoção de Elementos

- Três casos:
 1. Nó folha tem $r > \lfloor b/2 \rfloor$ registros. Neste caso, remova o registro e retorne *falso* (altura da árvore não diminuiu).
 2. Nó folha tem $r \leq \lfloor b/2 \rfloor$ registros, mas um dos irmãos do nó (mais velho, à direita, ou mais novo, à esquerda) tem um registro para emprestar. O pai desce e o irmão sobe, retornando *falso*.
 3. Nó folha tem $r \leq \lfloor b/2 \rfloor$ registros e não há a possibilidade de empréstimo. Deve ser feita a união do nó folha com seu irmão, inserindo o registro pai no meio deles. A altura pode diminuir neste caso, então retorne *verdadeiro*.

Árvores B: Exemplos de Remoção

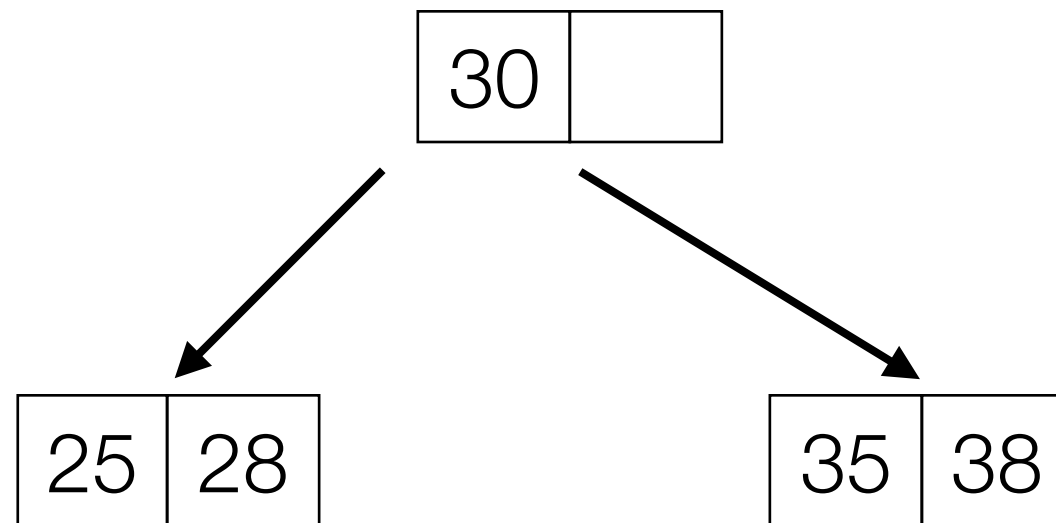
- Árvore inicial



- Remova 28
- Remova 25
- Remova 30

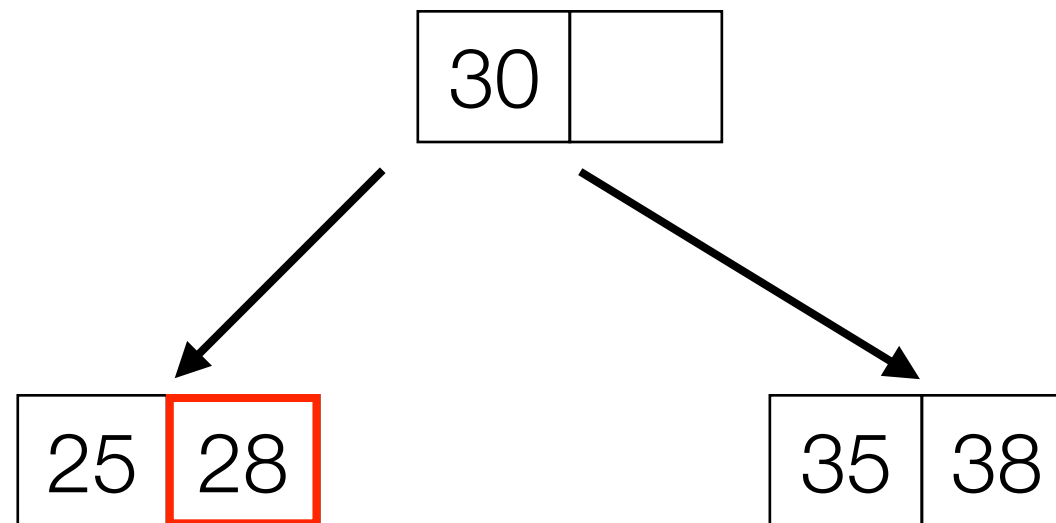
Árvores B: Exemplos de Remoção

- Remova 28:
- Caso 1: Nó folha tem $r > \lfloor b/2 \rfloor$ registros. Neste caso, remova o registro e retorne *false* (altura da árvore não diminuiu).



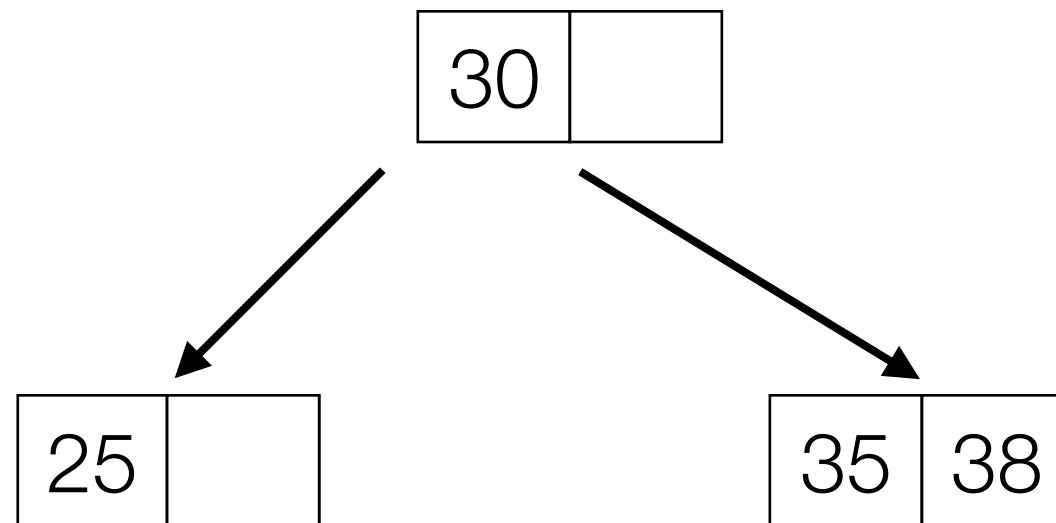
Árvores B: Exemplos de Remoção

- Remova 28:
- Caso 1: Nó folha tem $r > \lfloor b/2 \rfloor$ registros. Neste caso, remova o registro e retorne *false* (altura da árvore não diminuiu).



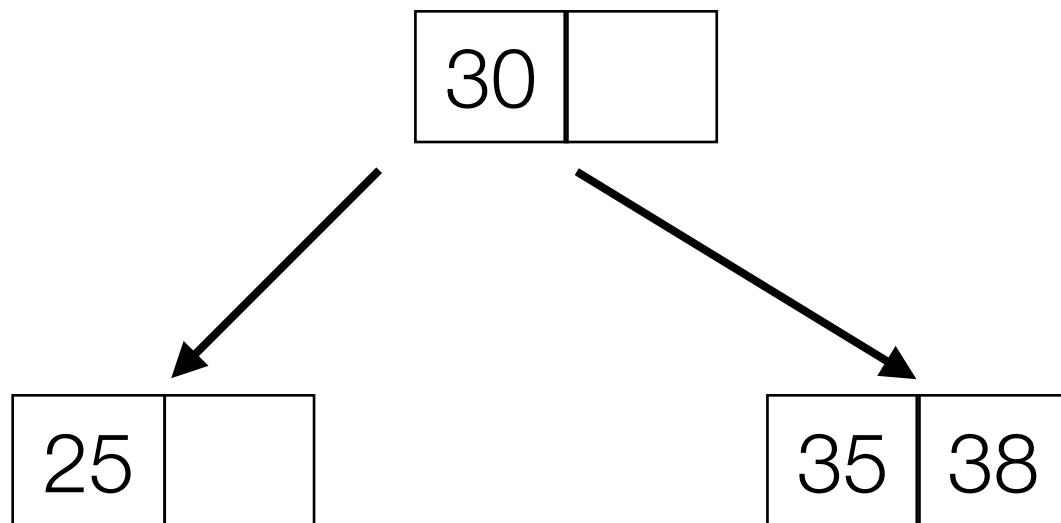
Árvores B: Exemplos de Remoção

- Remova 28:
- Caso 1: Nó folha tem $r > \lfloor b/2 \rfloor$ registros. Neste caso, remova o registro e retorne *false* (altura da árvore não diminuiu).



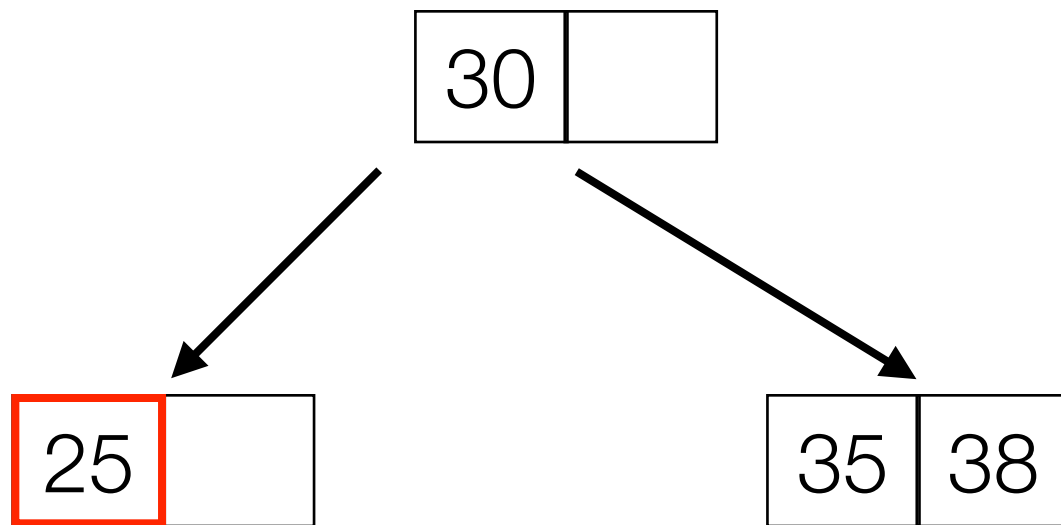
Árvores B: Exemplos de Remoção

- Remova 25:
- Caso 2: Nó folha tem $r \leq \lfloor b/2 \rfloor$ registros, mas um dos irmãos do nó (mais velho, à direita, ou mais novo, à esquerda) tem um registro para emprestar. O pai desce e o irmão sobe, retornando *false*.



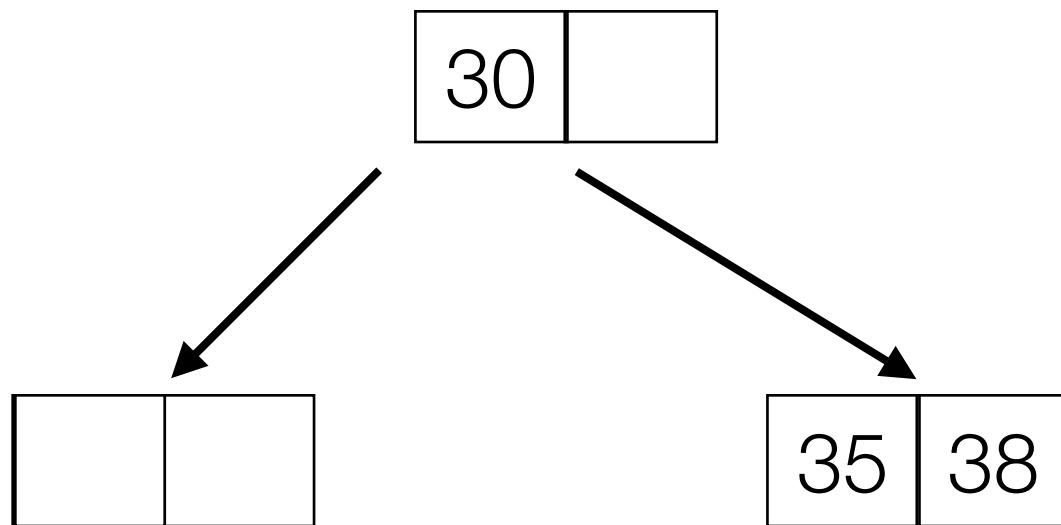
Árvores B: Exemplos de Remoção

- Remova 25:
- Caso 2: Nó folha tem $r \leq \lfloor b/2 \rfloor$ registros, mas um dos irmãos do nó (mais velho, à direita, ou mais novo, à esquerda) tem um registro para emprestar. O pai desce e o irmão sobe, retornando *false*.



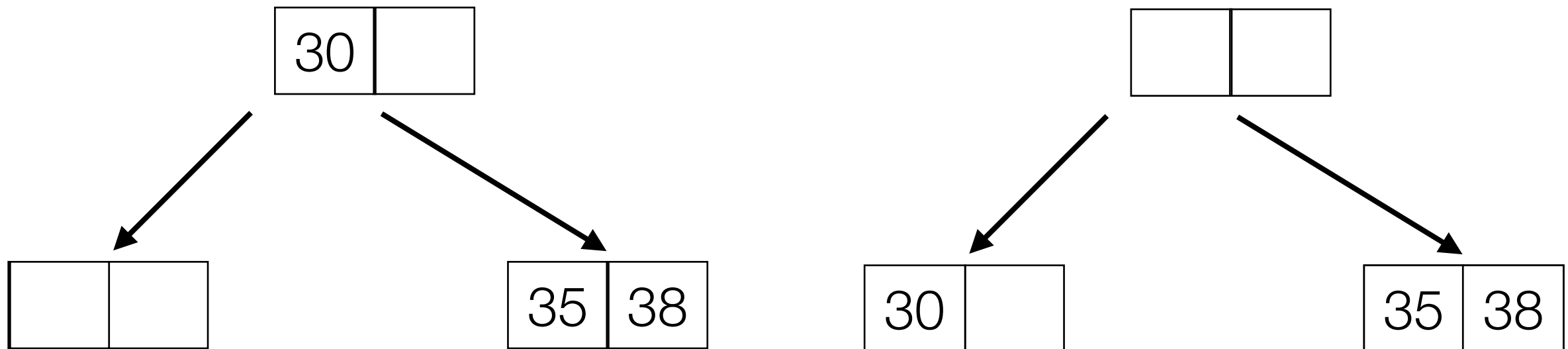
Árvores B: Exemplos de Remoção

- Remova 25:
- Caso 2: Nó folha tem $r \leq \lfloor b/2 \rfloor$ registros, mas um dos irmãos do nó (mais velho, à direita, ou mais novo, à esquerda) tem um registro para emprestar. O pai desce e o irmão sobe, retornando *false*.



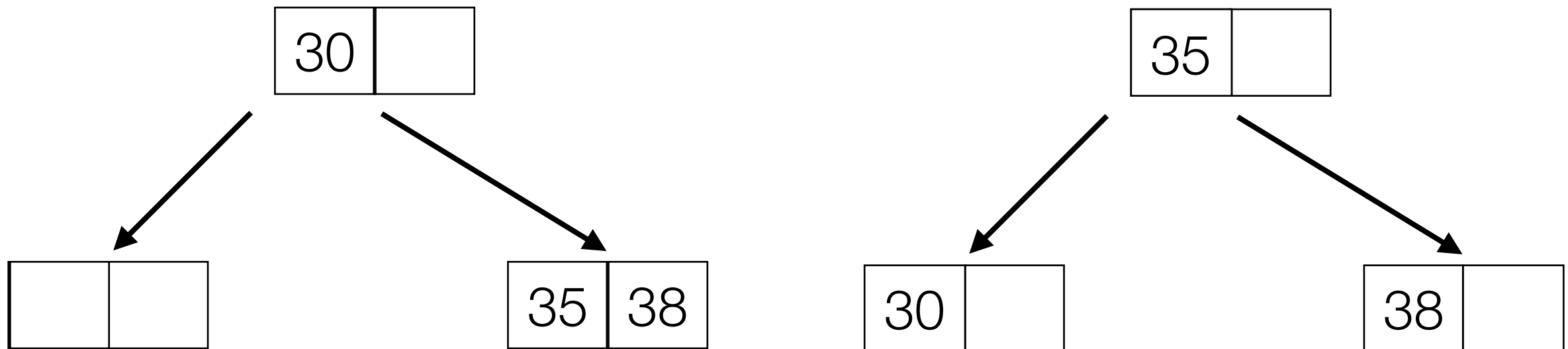
Árvores B: Exemplos de Remoção

- Remova 25:
- Caso 2: Nó folha tem $r \leq \lfloor b/2 \rfloor$ registros, mas um dos irmãos do nó (mais velho, à direita, ou mais novo, à esquerda) tem um registro para emprestar. O pai desce e o irmão sobe, retornando *false*.



Árvores B: Exemplos de Remoção

- Remova 25:
- Caso 2: Nó folha tem $r \leq \lfloor b/2 \rfloor$ registros, mas um dos irmãos do nó (mais velho, à direita, ou mais novo, à esquerda) tem um registro para emprestar. O pai desce e o irmão sobe, retornando *false*.

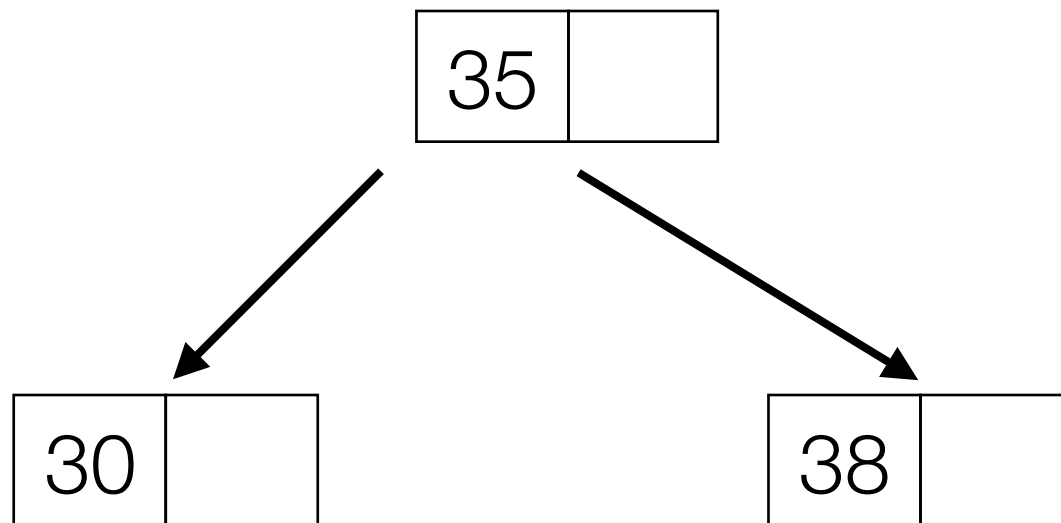


Árvores B: Exemplos de Remoção

- Remova 30:
 - Caso 3: Nó folha tem $r \leq \lfloor b/2 \rfloor$ registros e não há a possibilidade de empréstimo. Deve ser feita a união do nó folha com seu irmão, inserindo o registro pai no meio deles. A altura pode diminuir neste caso, então retorne *verdadeiro*.

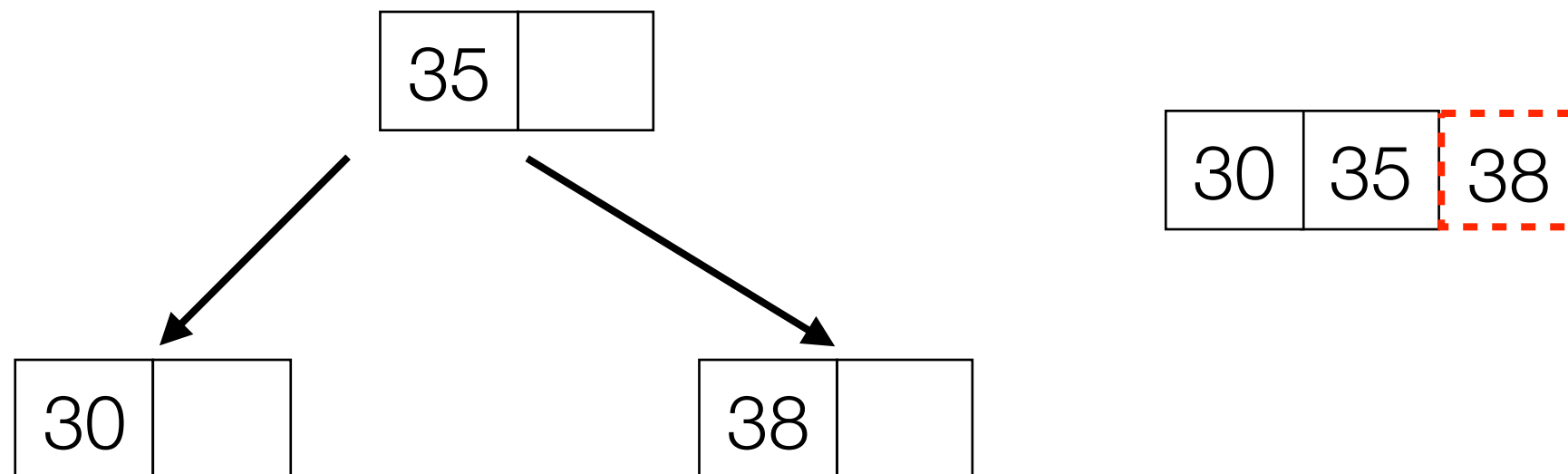
Árvores B: Exemplos de Remoção

- Remova 30:
- Caso 3: Nó folha tem $r \leq \lfloor b/2 \rfloor$ registros e não há a possibilidade de empréstimo. Deve ser feita a união do nó folha com seu irmão, inserindo o registro pai no meio deles. A altura pode diminuir neste caso, então retorne *verdadeiro*.



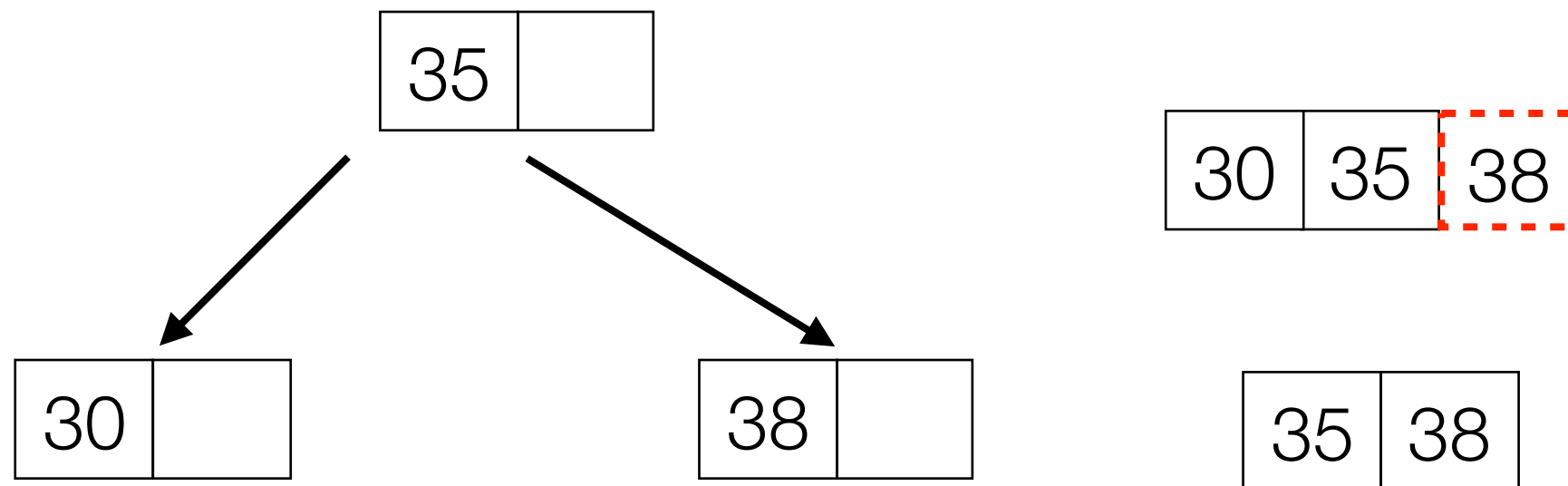
Árvores B: Exemplos de Remoção

- Remova 30:
- Caso 3: Nó folha tem $r \leq \lfloor b/2 \rfloor$ registros e não há a possibilidade de empréstimo. Deve ser feita a união do nó folha com seu irmão, inserindo o registro pai no meio deles. A altura pode diminuir neste caso, então retorne *verdadeiro*.



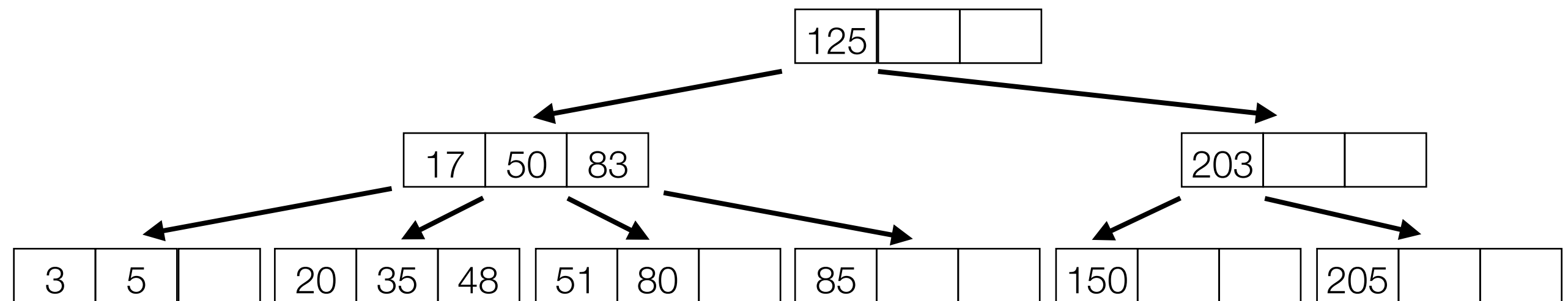
Árvores B: Exemplos de Remoção

- Remova 30:
- Caso 3: Nó folha tem $r \leq \lfloor b/2 \rfloor$ registros e não há a possibilidade de empréstimo. Deve ser feita a união do nó folha com seu irmão, inserindo o registro pai no meio deles. A altura pode diminuir neste caso, então retorne *verdadeiro*.



Árvores B: Exemplos de Remoção 2

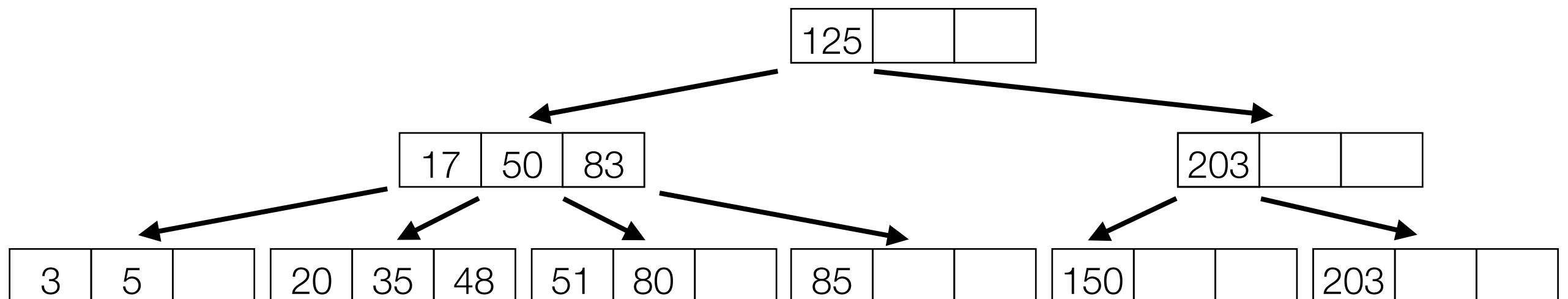
- Árvore inicial



- Remova 50
- Remova 150

Árvores B: Exemplos de Remoção 2

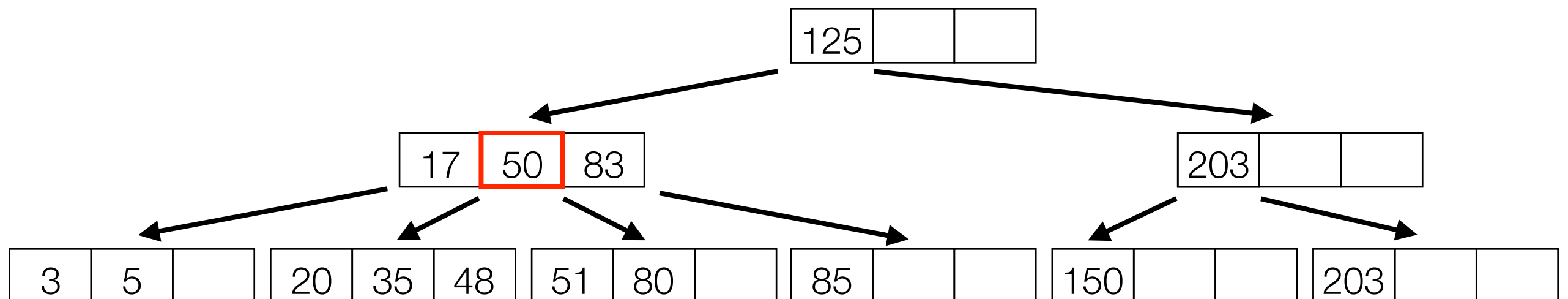
- Remova 50



Árvores B: Exemplos de Remoção 2

- Remova 50

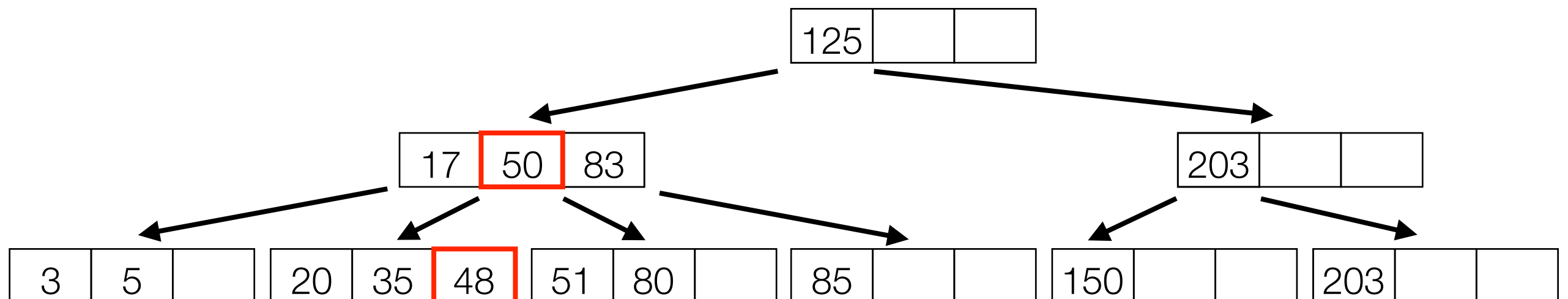
1. Como a chave 50 está em um nó interno, devemos buscar seu maior antecessor em um nó folha e trocar ambos de lugar



Árvores B: Exemplos de Remoção 2

- Remova 50

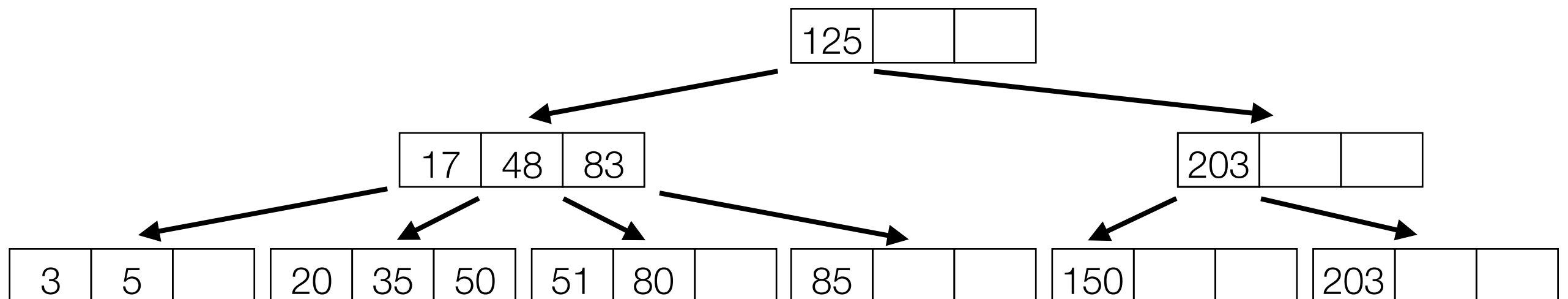
1. Como a chave 50 está em um nó interno, devemos buscar seu maior antecessor em um nó folha e trocar ambos de lugar



Árvores B: Exemplos de Remoção 2

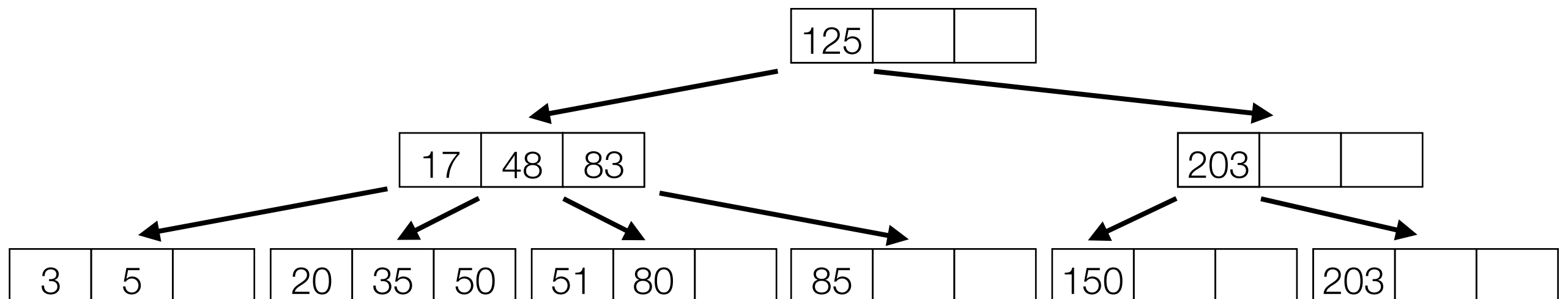
- Remova 50

1. Como a chave 50 está em um nó interno, devemos buscar seu maior antecessor em um nó folha e trocar ambos de lugar



Árvores B: Exemplos de Remoção 2

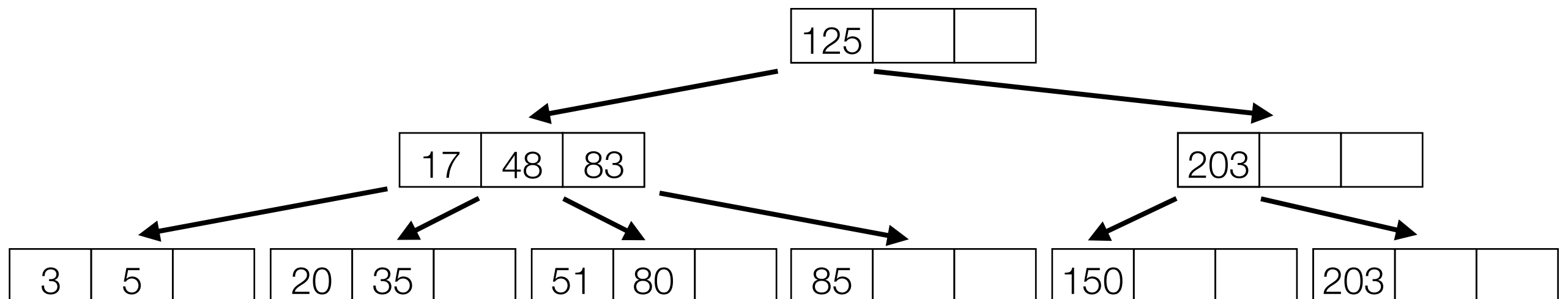
- Remova 50
 1. Como a chave 50 está em um nó interno, devemos buscar seu maior antecessor em um nó folha e trocar ambos de lugar
 2. Em seguida chamamos a função recursivamente para remover 50 no nó folha correspondente, como anteriormente



Árvores B: Exemplos de Remoção 2

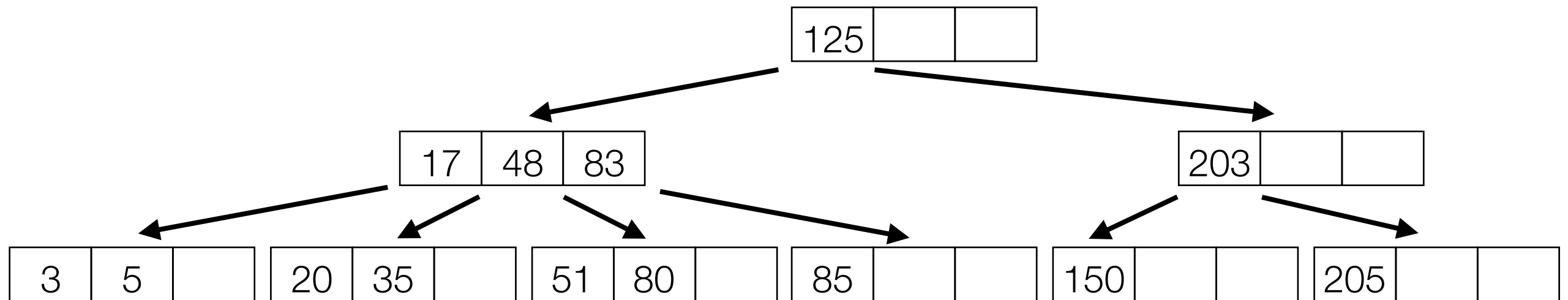
- Remova 50

1. Como a chave 50 está em um nó interno, devemos buscar seu maior antecessor em um nó folha e trocar ambos de lugar
2. Em seguida chamamos a função recursivamente para remover 50 no nó folha correspondente, como anteriormente
3. Caso 1 de remoção



Árvores B: Exemplos de Remoção 2

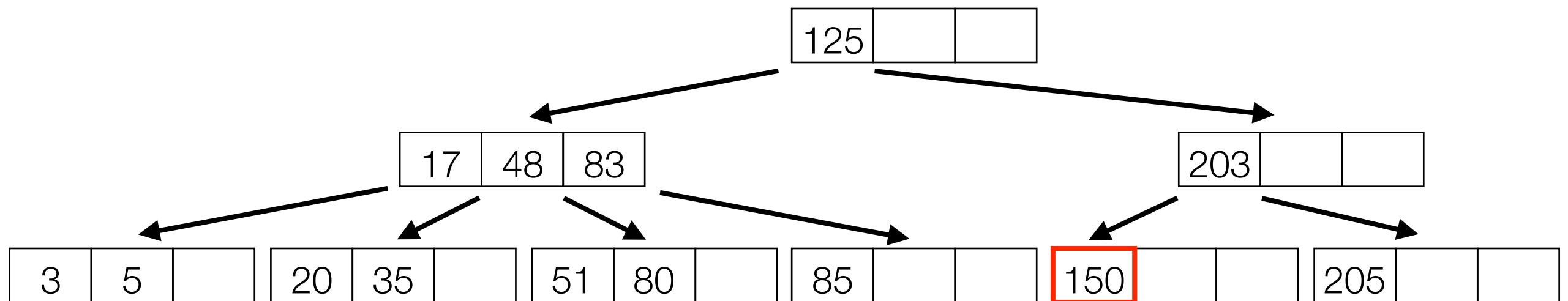
- Remova 150



Árvores B: Exemplos de Remoção 2

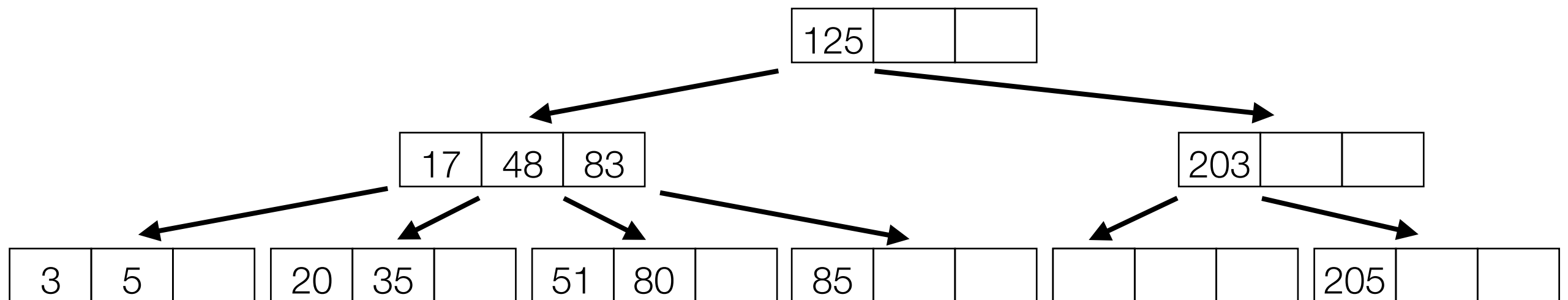
- Remova 150

1. Ache recursivamente a chave correspondente



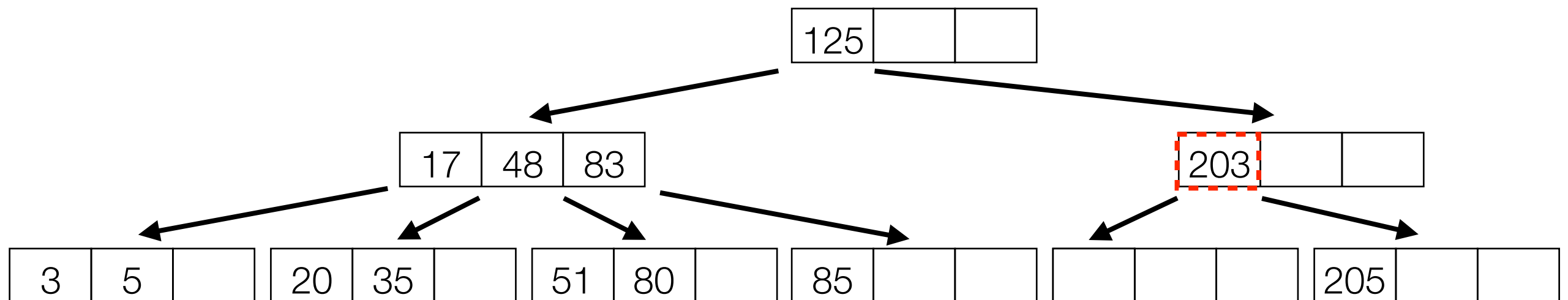
Árvores B: Exemplos de Remoção 2

- Remova 150
 1. Ache recursivamente a chave correspondente
 2. Elimine o elemento



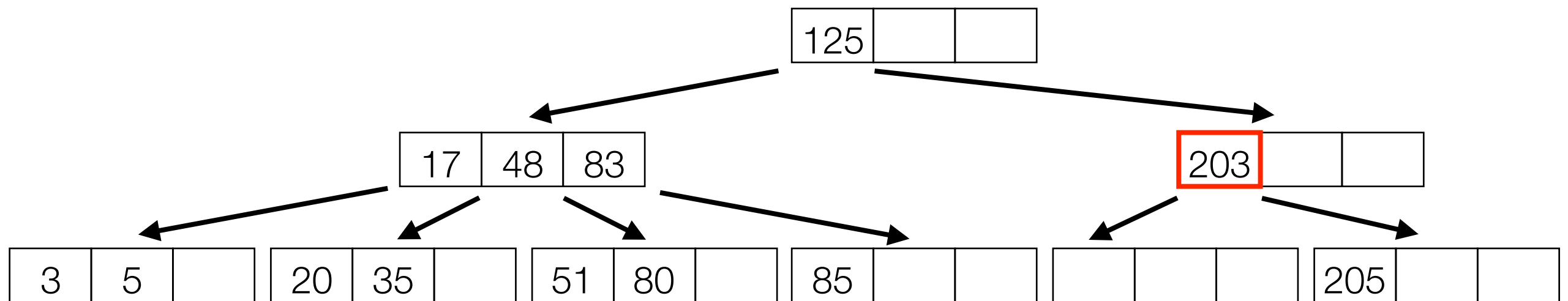
Árvores B: Exemplos de Remoção 2

- Remova 150
 1. Ache recursivamente a chave correspondente
 2. Elimine o elemento
 3. Trate o *underflow* do filho no retorno da recursão



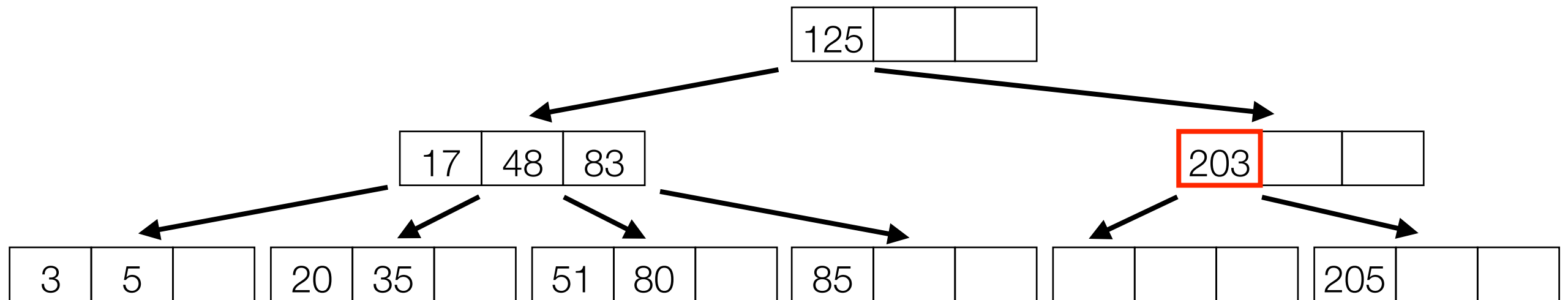
Árvores B: Exemplos de Remoção 2

- Tratamento de underflow #1 (volta da recursão #1)



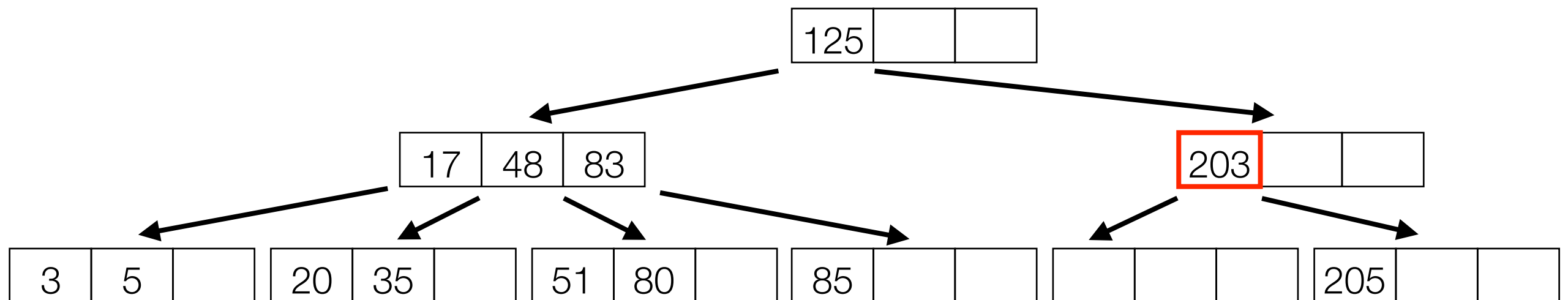
Árvores B: Exemplos de Remoção 2

- Tratamento de underflow #1 (volta da recursão #1)
- Verifica casos 1, 2, 3



Árvores B: Exemplos de Remoção 2

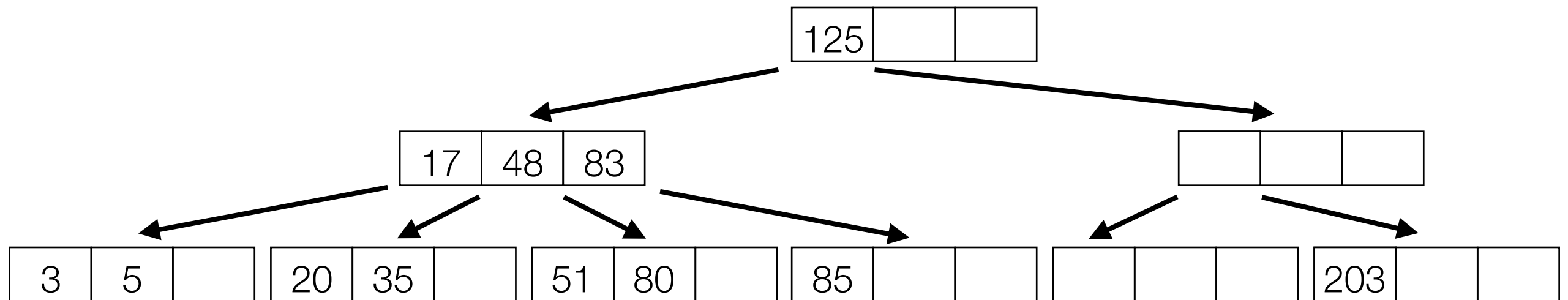
- Tratamento de underflow #1 (volta da recursão #1)
- Verifica casos 1, 2, 3
 - Caso 3:



Árvores B: Exemplos de Remoção 2

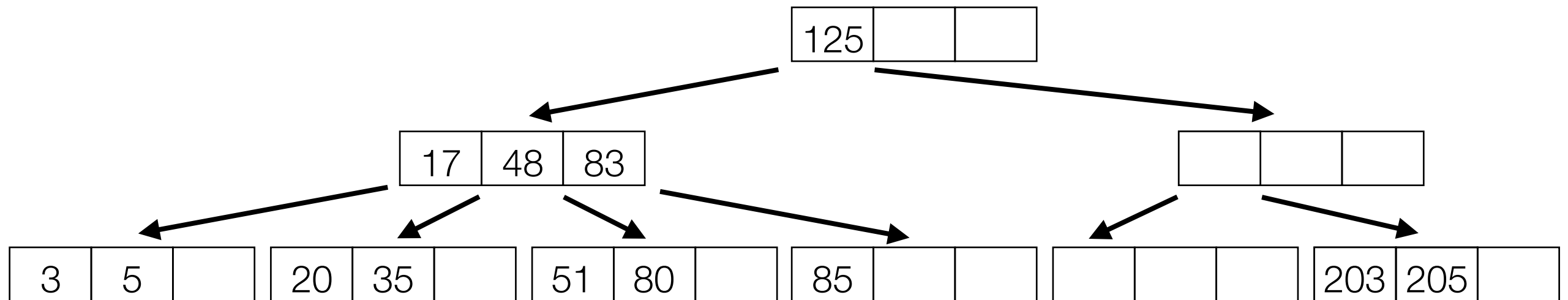
- Tratamento de underflow #1 (volta da recursão #1)
- Verifica casos 1, 2, 3
 - Caso 3:

1. Empurra chave para baixo



Árvores B: Exemplos de Remoção 2

- Tratamento de underflow #1 (volta da recursão #1)
- Verifica casos 1, 2, 3
 - Caso 3:
 1. Empurra chave para baixo
 2. Faz união com o nó irmão (neste caso, [vazio] + [203, 205, _])



Árvores B: Exemplos de Remoção 2

- Tratamento de underflow #1 (volta da recursão #1)

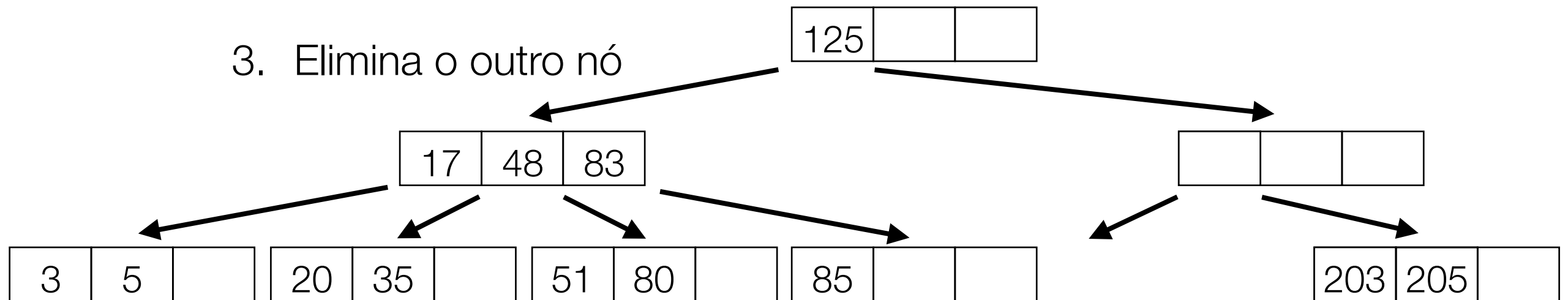
- Verifica casos 1, 2, 3

- Caso 3:

1. Empurra chave para baixo

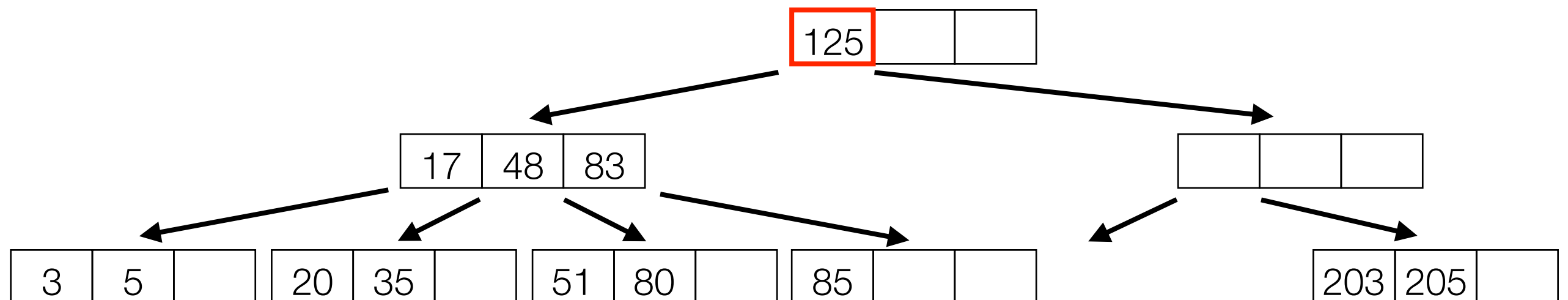
2. Faz união com o nó irmão (neste caso, [vazio] + [203, 205, _])

3. Elimina o outro nó



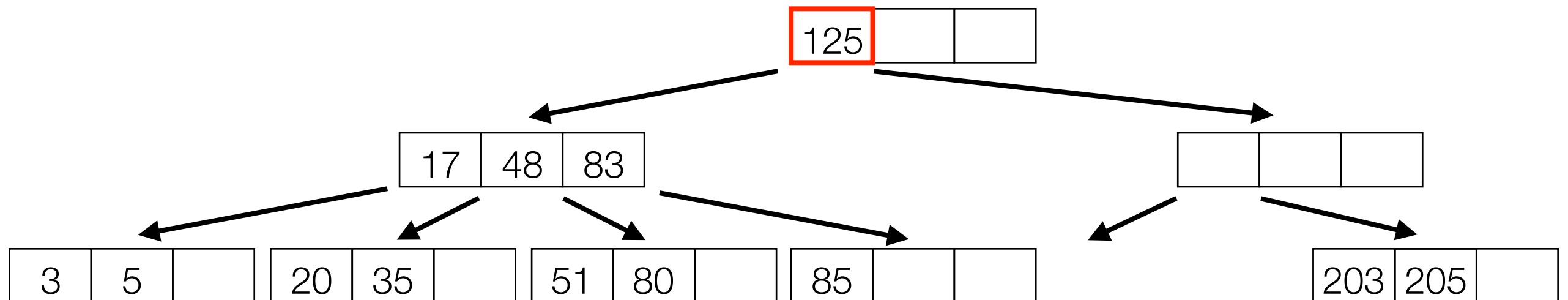
Árvores B: Exemplos de Remoção 2

- Tratamento de underflow #2 (volta da recursão #2)



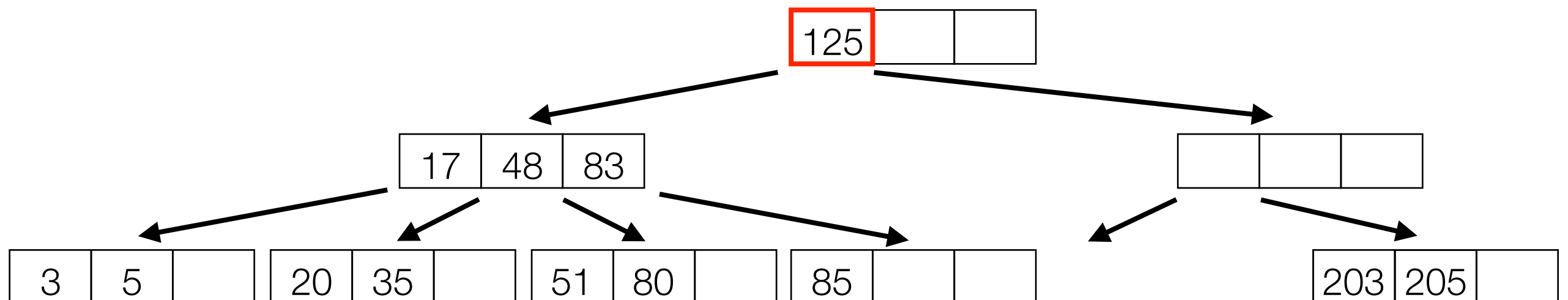
Árvores B: Exemplos de Remoção 2

- Tratamento de underflow #2 (volta da recursão #2)
- Verifica casos 1, 2, 3



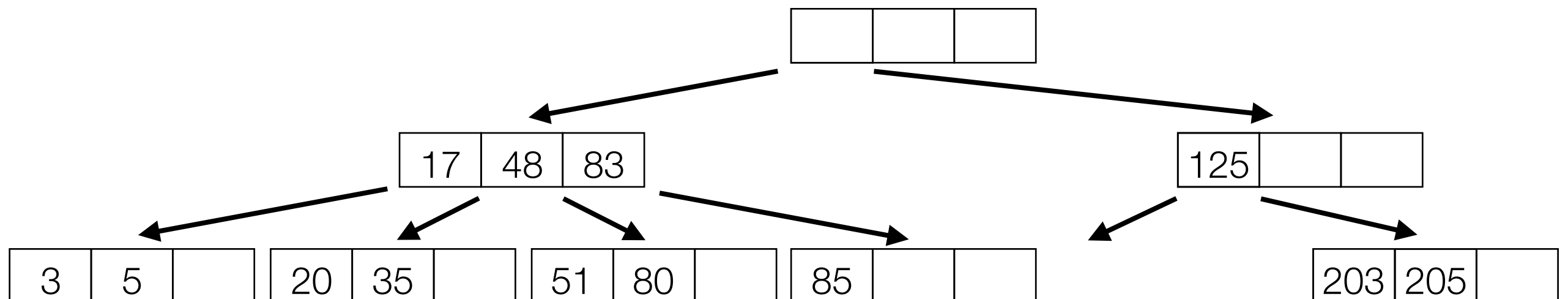
Árvores B: Exemplos de Remoção 2

- Tratamento de underflow #2 (volta da recursão #2)
- Verifica casos 1, 2, 3
 - Caso 2:



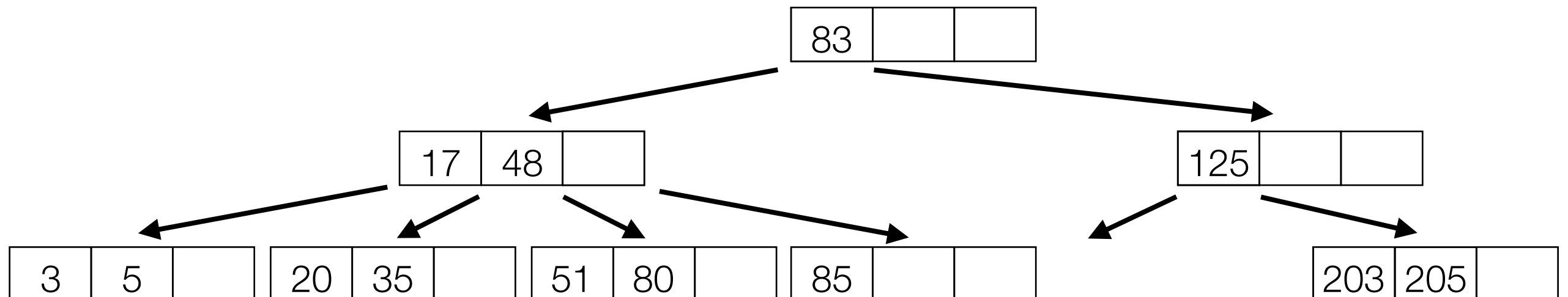
Árvores B: Exemplos de Remoção 2

- Tratamento de underflow #2 (volta da recursão #2)
- Verifica casos 1, 2, 3
 - Caso 2:
 1. Empurra a chave pai 125 para baixo



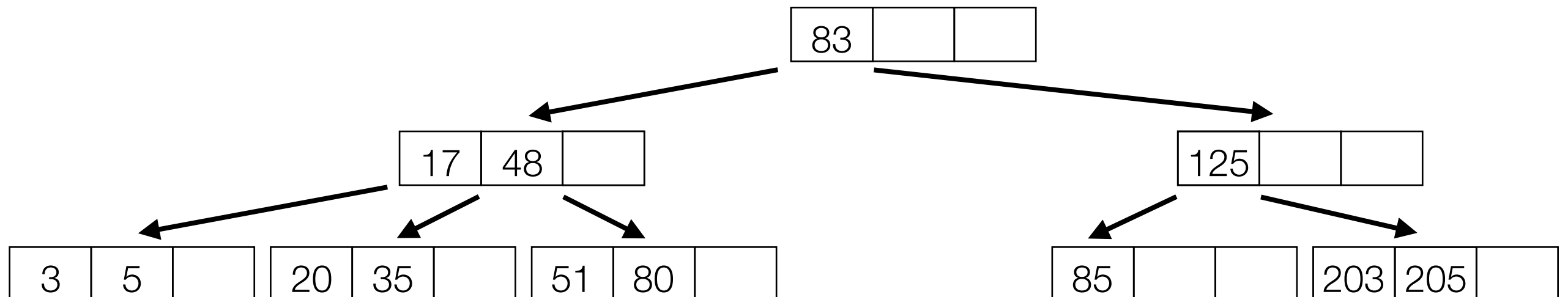
Árvores B: Exemplos de Remoção 2

- Tratamento de underflow #2 (volta da recursão #2)
- Verifica casos 1, 2, 3
 - Caso 2:
 1. Empurra a chave pai 125 para baixo
 2. Emprста a maior chave do irmão à esquerda (83) -> caso o irmão não possa emprestar, verificar o irmão à direita (caso simétrico)



Árvores B: Exemplos de Remoção 2

- Tratamento de underflow #2 (volta da recursão #2)
- Verifica casos 1, 2, 3
 - Caso 2:
 1. Empurra a chave pai 125 para baixo
 2. Emprasta a maior chave do irmão à esquerda (83) -> caso o irmão não possa emprestar, verificar o irmão à direita (caso simétrico)
 3. O nó entre 83 e 125 acompanha a chave pai



Árvores B: Implementação da Remoção

```
int ArvBRemoveRec(ArvB **arv, int chave, int *underflow) {
    int pos;
    /* Primeiro, procura onde esta a chave */
    if (BuscaChaveNo(*arv, chave, &pos)) {
        /* a chave esta neste nivel */
        if ((*arv)->filhos[pos] != NULL) { /* Nao estamos na raiz */
            TrocaChaveAnterior(*arv, pos); /* manda a chave para baixo */
            ArvBRemoveRec(&((*arv)->filhos[pos]), chave, underflow);
            if (*underflow) TrataUnderflow(arv, pos);
        } else {
            RemoveChaveNo(*arv, pos);
        }
        (*underflow) = VerificaUnderflow(*arv);
        return 1;
    } else { /* a chave nao esta neste nivel */
        if ((*arv)->filhos[pos] != NULL) {
            if (ArvBRemoveRec(&((*arv)->filhos[pos]), chave, underflow)) {
                /* existe o no abaixo */
                if (*underflow) TrataUnderflow(arv, pos);
                (*underflow) = VerificaUnderflow(*arv);
                return 1;
            } else {
                return 0; /* nao existe o no abaixo */
            }
        } else { /* estamos na raiz e a arvore nao esta neste nivel,
                    logo, ela nao existe */
            (*underflow) = 0;
            return 0;
        }
    }
}
```

```

int ArvBRemoveRec(ArvB **arv, int chave, int *underflow) {
    int pos;
    /* Primeiro, procura onde esta a chave */
    if (BuscaChaveNo(*arv, chave, &pos)) {
        /* a chave esta neste nivel */
        if ((*arv)->filhos[pos] != NULL) { /* Nao estamos na raiz */
            TrocaChaveAnterior(*arv, pos); /* manda a chave para baixo */
            ArvBRemoveRec(&((*arv)->filhos[pos]), chave, underflow);
            if (*underflow) TrataUnderflow(arv, pos);
        } else {
            RemoveChaveNo(*arv, pos);
        }
        (*underflow) = VerificaUnderflow(*arv);
        return 1;
    } else { /* a chave nao esta neste nivel */
        if ((*arv)->filhos[pos] != NULL) {
            if (ArvBRemoveRec(&((*arv)->filhos[pos]), chave, underflow)) {
                /* existe o no abaixo */
                if (*underflow) TrataUnderflow(arv, pos);
                (*underflow) = VerificaUnderflow(*arv);
                return 1;
            } else {
                return 0; /* nao existe o no abaixo */
            }
        } else { /* estamos na raiz e a arvore nao esta neste nivel,
                    logo, ela nao existe */
            (*underflow) = 0;
            return 0;
        }
    }
}

```



```

int ArvBRemoveRec(ArvB **arv, int chave, int *underflow) {
    int pos;
    /* Primeiro, procura onde esta a chave */
    if (BuscaChaveNo(*arv, chave, &pos)) {
        /* a chave esta neste nivel */
        if ((*arv)->filhos[pos] != NULL) { /* Nao estamos na raiz */
            TrocaChaveAnterior(*arv, pos); /* manda a chave para baixo */
            ArvBRemoveRec(&((*arv)->filhos[pos]), chave, underflow);
            if (*underflow) TrataUnderflow(arv, pos);
        } else {
            RemoveChaveNo(*arv, pos);
        }
        (*underflow) = VerificaUnderflow(*arv);
        return 1;
    } else { /* a chave nao esta neste nivel */
        if ((*arv)->filhos[pos] != NULL) {
            if (ArvBRemoveRec(&((*arv)->filhos[pos]), chave, underflow)) {
                /* existe o no abaixo */
                if (*underflow) TrataUnderflow(arv, pos);
                (*underflow) = VerificaUnderflow(*arv);
                return 1;
            } else {
                return 0; /* nao existe o no abaixo */
            }
        } else { /* estamos na raiz e a arvore nao esta neste nivel,
                    logo, ela nao existe */
            (*underflow) = 0;
            return 0;
        }
    }
}

```


Árvores B: Implementação da Remoção

- **IMPORTANTE:** a redução da altura da árvore ocorre no caso 3, quando a recursão chegou na raiz e utilizou a única chave restante para unir dois filhos em underflow
- No algoritmo anterior, isso deve ser tratado fora da função `ArvBRemoveRec`, avaliando se o retorno da mesma foi 1 e se ocorreu underflow (`*underflow == 1`)
- No caso, `*underflow` será 1 porque ao utilizar a única chave para tratar o underflow de seus filhos, a raiz ficará vazia e `VerificaUnderflow` atualizará a variável apropriadamente