

Método de Busca

- **O problema da busca (ou pesquisa) - Dado um conjunto de elementos, onde cada um é identificado por uma chave, o objetivo da busca é localizar, nesse conjunto, o elemento que corresponde a uma chave específica.**
- **Vários métodos e estruturas de dados podem ser empregados para se fazer busca.**
- **Certos métodos de organização/ordenação de dados podem tornar o processo de busca mais eficiente**

TIPO DE BUSCA

O conjunto de registros pode ser:

- **Um vetor de registros**
- **Uma lista encadeada**
- **Uma árvore**
- **Etc.**

O conjunto de registros pode ficar:

- **Totalmente na memória (busca interna)**
- **Totalmente no armazenamento auxiliar (busca externa)**
- **Dividida entre ambos**

TIPO DE BUSCA

1. Busca Seqüencial
2. Busca Binária
3. Arvore de Busca Binária
4. Hash

BUSCA SEQUENCIAL

Compara a chave com cada item na array ou lista, até encontrar um item de dado cujo valor é igual o valor da chave.

Coleção de Dados:

10 3 16 0 -1 104 23 -7 88 6 4 1000

Chave: 0

Exercício: Escreve uma função de busca seqüencial em C.

BUSCA SEQUENCIAL

Algoritmo de busca seqüencial em um vetor A, com N posições (0 até N-1), sendo x a chave procurada

```
for (i=0; i<n; i++)  
    if (A[i]==x)  
        return(i); /*chave encontrada*/  
return(-1); /*chave não encontrada*/
```

BUSCA SEQUENCIAL

Outra maneira de implementar o algoritmo é usar um sentinela

- **Sentinela: consiste em adicionar um elemento de valor x no final da tabela**
- **O sentinela garante que o elemento procurado será encontrado, o que elimina uma expressão condicional, melhorando a performance do algoritmo**

```
A[N]=x;  
for(i=0; x!=A[i]; i++);  
if (i<n) return(i); /*chave encontrada*/  
else return(-1); /*sentinela encontrado*/
```

BUSCA SEQUENCIAL

- **Limitações do vetor**
 - **Tamanho fixo**
 - **Pode desperdiçar ou faltar espaço**
- **Alternativa**
 - **Lista encadeada**

O que muda na busca seqüencial?

Exercício

Escrever em C a sub-rotina de busca de um elemento em uma lista encadeada

BUSCA SEQUENCIAL

Complexidade

- Se o registro for o primeiro: 1 comparação
- Se o registro procurado for o último: N comparações
- Se for igualmente provável que o argumento apareça em qualquer posição da tabela, em média: $(n+1)/2$ comparações
- Se a busca for mal sucedida: N comparações
- Logo, a busca seqüencial, no pior caso, é $O(n)$

BUSCA SEQUENCIAL

Para aumentar a eficiência

- Reordenar continuamente a lista de modo que os registros mais acessados sejam deslocados para o início**

BUSCA BINÁRIA

Se os dados estiverem ordenados em um arranjo, pode-se tirar vantagens dessa ordenação - Busca binária

O elemento buscado é comparado ao elemento do meio do arranjo

- Se igual, busca bem-sucedida**
- Se menor, busca-se na metade inferior do arranjo**
- Se maior, busca-se na metade superior do arranjo**

BUSCA BINÁRIA

Busca-se por 25

inf=1 sup=N=8

12	25	33	37	48	57	86	92
----	----	----	----	----	----	----	----

inf=1 meio sup=N=8

12	25	33	37	48	57	86	92
----	----	----	----	----	----	----	----

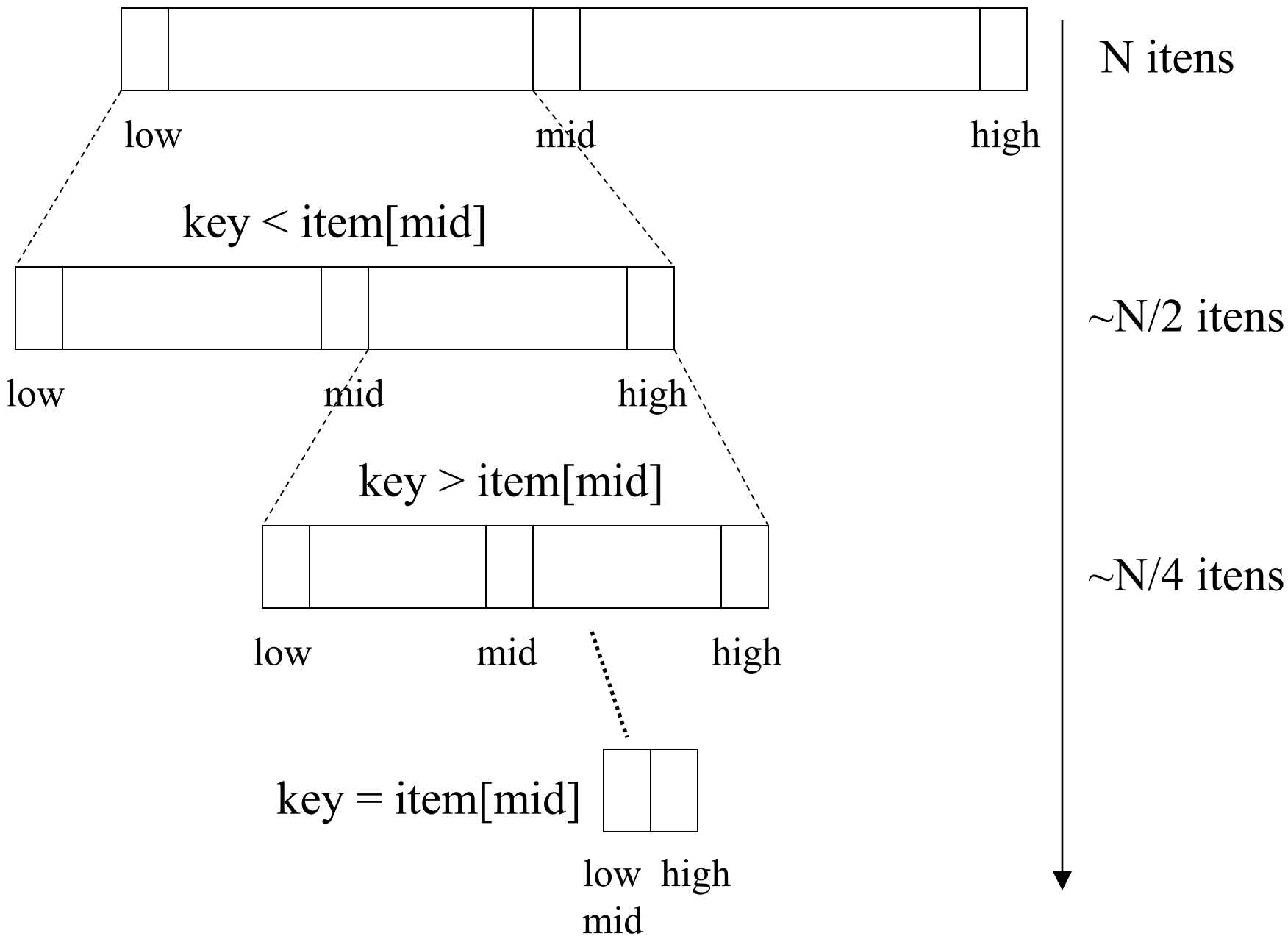
↑
25 < 37

BUSCA BINÁRIA

Busca-se por 25

inf=1	meio	sup=3					N=8
12	25	33	37	48	57	86	92
	↑						
	=25						

Em cada passo, o tamanho do arranjo em que se busca é dividido por 2



BUSCA BINÁRIA

Exercício

Escrever em C uma sub-rotina de busca binária por um elemento em um arranjo ordenado

BUSCA BINÁRIA

Bin-Search(collection c , low , $high$, k)

int mid ;

if $low > high$

then return NIL;

$mid = (high+low)/2$;

if $k = key[mid]$

then return $key[mid]$;

else if $k < key[mid]$

then return Bin_search(c , low , $mid-1$, k);

else return Bin_search(c , $mid+1$, $high$, k);

BUSCA BINÁRIA

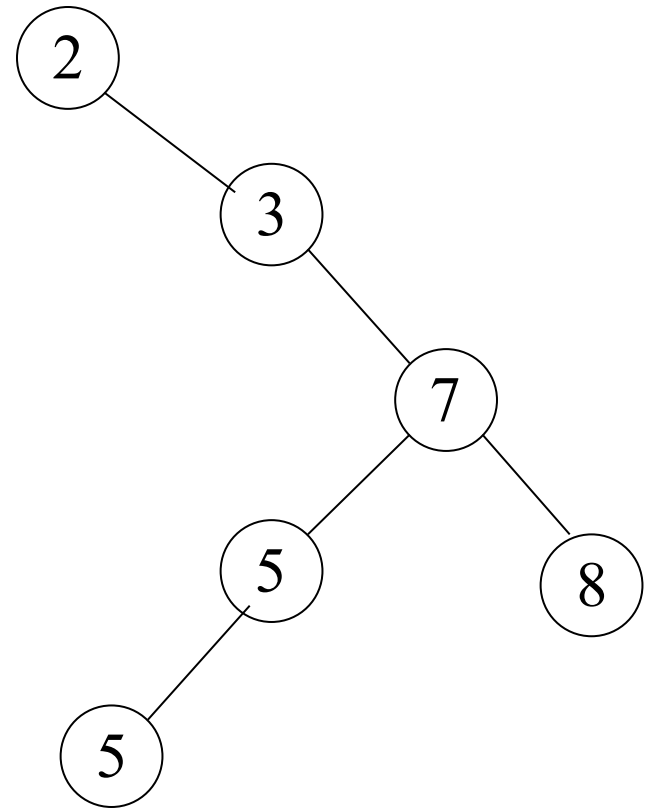
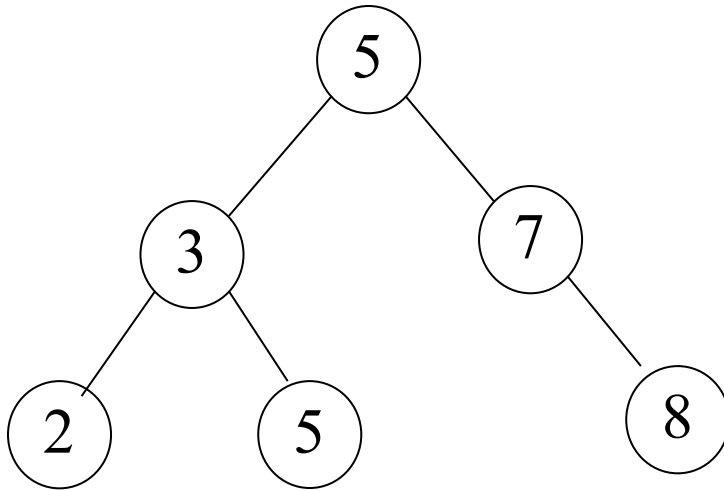
Complexidade?

- $O(\log(n))$, pois cada comparação reduz o número de possíveis candidatos por um fator de 2.

ÁRVORES DE BUSCAS BINÁRIAS

Arvore de busca binária é representada por uma estrutura de dados ligada. Cada nó contem 4 campos:

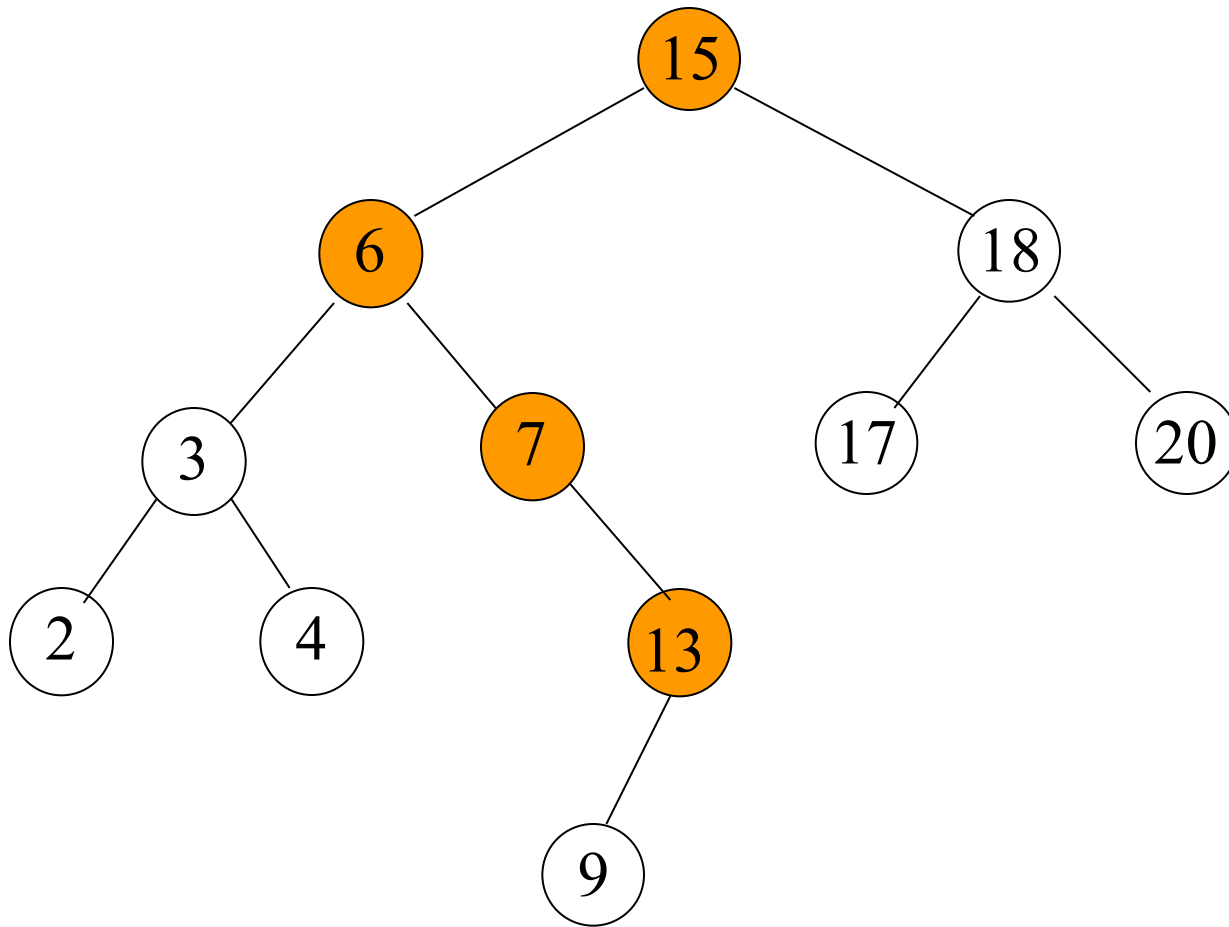
- *key*: valor do nó;
- *left*: ponteiro que aponta para seu filho esquerdo;
- *right*: ponteiro que aponta para seu filho direito;
- *p*: ponteiro que aponta para seu pai.



PROPRIEDADE:

Suponho que x é um nó da árvore de busca binária, para qualquer nó y , se y está na sub-árvore esquerda do x , então $key[y] \leq key[x]$.

Se y está na sub-árvore direita do x , então $key[x] \leq key[y]$.



Para encontrar a chave 13 na árvore, seguimos o caminho
 $15 \rightarrow 6 \rightarrow 7 \rightarrow 13$

Algoritmo de Busca Geral

Algoritmo Recursivo

```
Tree-Search( $x, k$ )  
if  $x = \text{NIL}$  or  $k = \text{key}[x]$   
    then return  $x$   
if  $k < \text{key}[x]$   
    then return Tree-Search( $\text{left}[x], k$ )  
    else return Tree-Search( $\text{right}[x], k$ )
```

Algoritmo Iterativo

```
Iterative-Tree-Search( $x, k$ )  
while  $x \neq \text{NIL}$  and  $k \neq \text{key}[x]$   
    do if  $k < \text{key}[x]$   
        then  $x \leftarrow \text{left}[x]$   
        else  $x \leftarrow \text{right}[x]$   
return  $x$ 
```

Algoritmo de Busca do Valor Mínimo

```
Tree-Minimum(x)  
while left[x] ≠ NIL  
    do x ← left[x]  
return x
```

Algoritmo de Busca do Valor Máximo

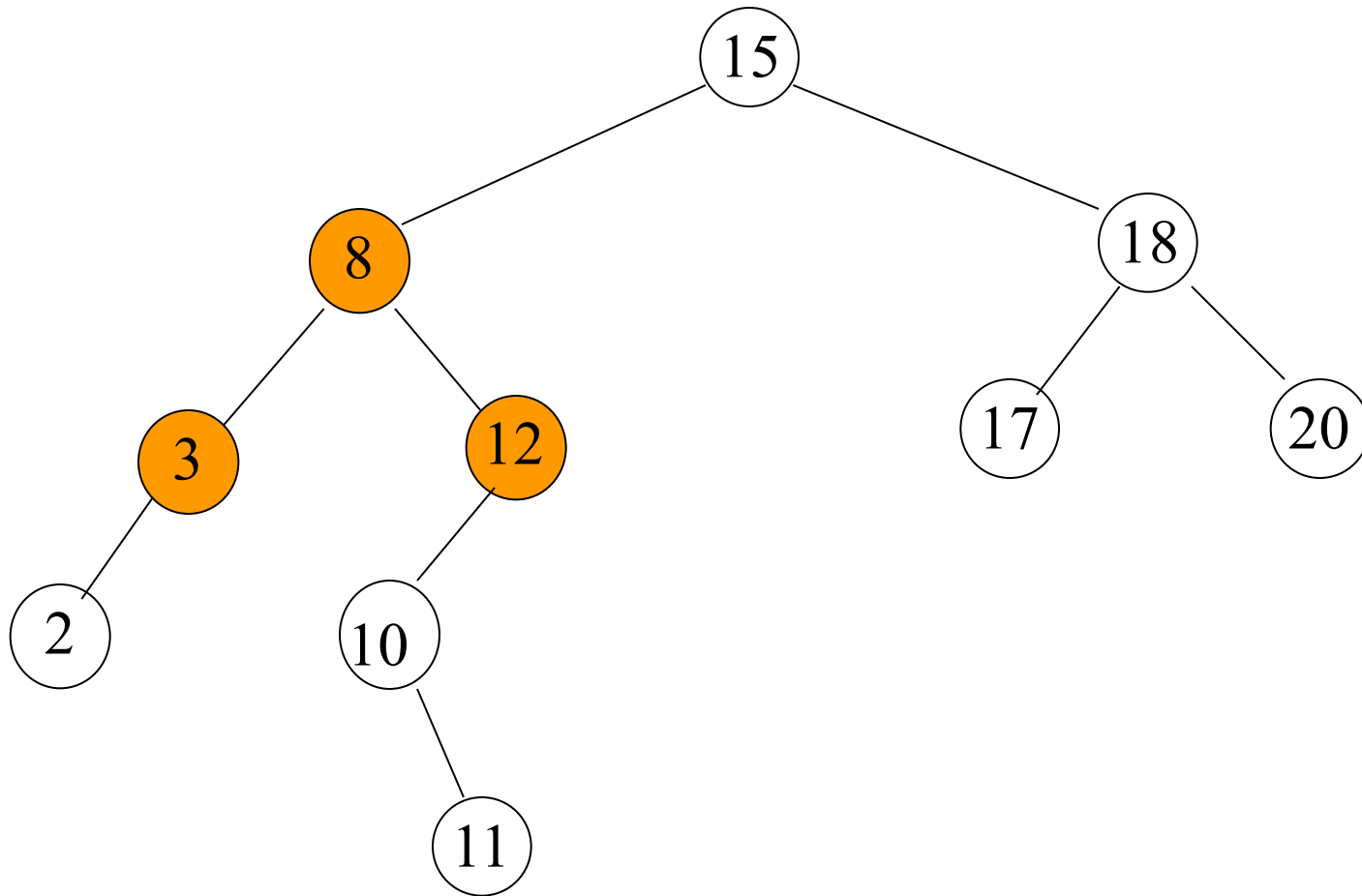
```
Tree-Maximum(x)  
while right[x] ≠ NIL  
    do x ← right[x]  
return x
```

Algoritmo de Busca do Valor Sucessor

O sucessor do nó x é o nó com o menor chave maior que $key[x]$.

Case 1: Se a subarvore direita do nó x não for vazio, então, o sucessor do x é o nó mais esquerdo na subarvore direita;

Case 2: Se a subarvore direita do nó x for vazio, o sucessor do x (se x é um filho esquerdo) é o antecessor de nível mais baixa ou é o antecessor de nível mais baixa , cujo filho esquerdo também é antecessor do x (se x é um filho direito) .



Sucessor do nó 8?

Sucessor do nó 3?

Sucessor do nó 12?

Algoritmo de Busca do Valor Sucessor

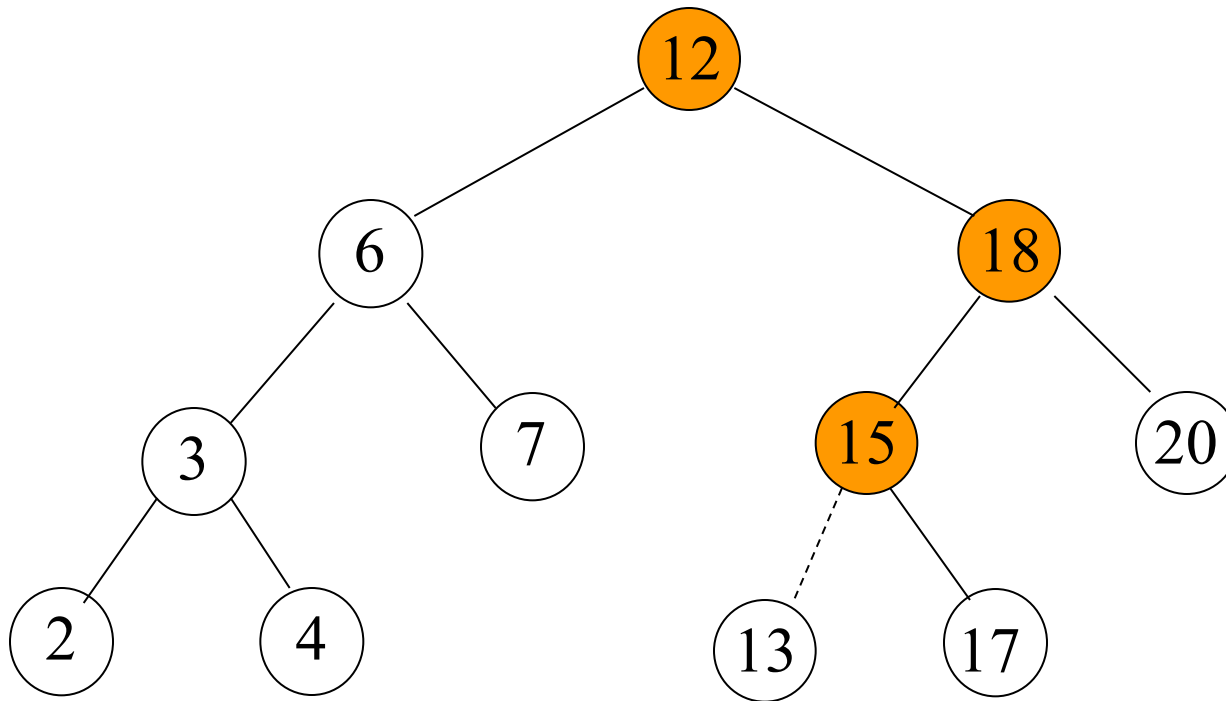
```
Tree-Successor( $x$ )  
if  $right[x] \neq \text{NIL}$   
    then return Tree-Minimum( $right[x]$ )  
 $y \leftarrow p[x]$   
while  $y \neq \text{NIL}$  and  $x = right[p[x]]$   
    do  $x \leftarrow y$   
         $y \leftarrow p[y]$   
return  $y$ 
```

Exercício:

Escreva o algoritmo de busca do valor predecessor de um nó x em uma árvore de busca binária.

Algoritmo de Inserção

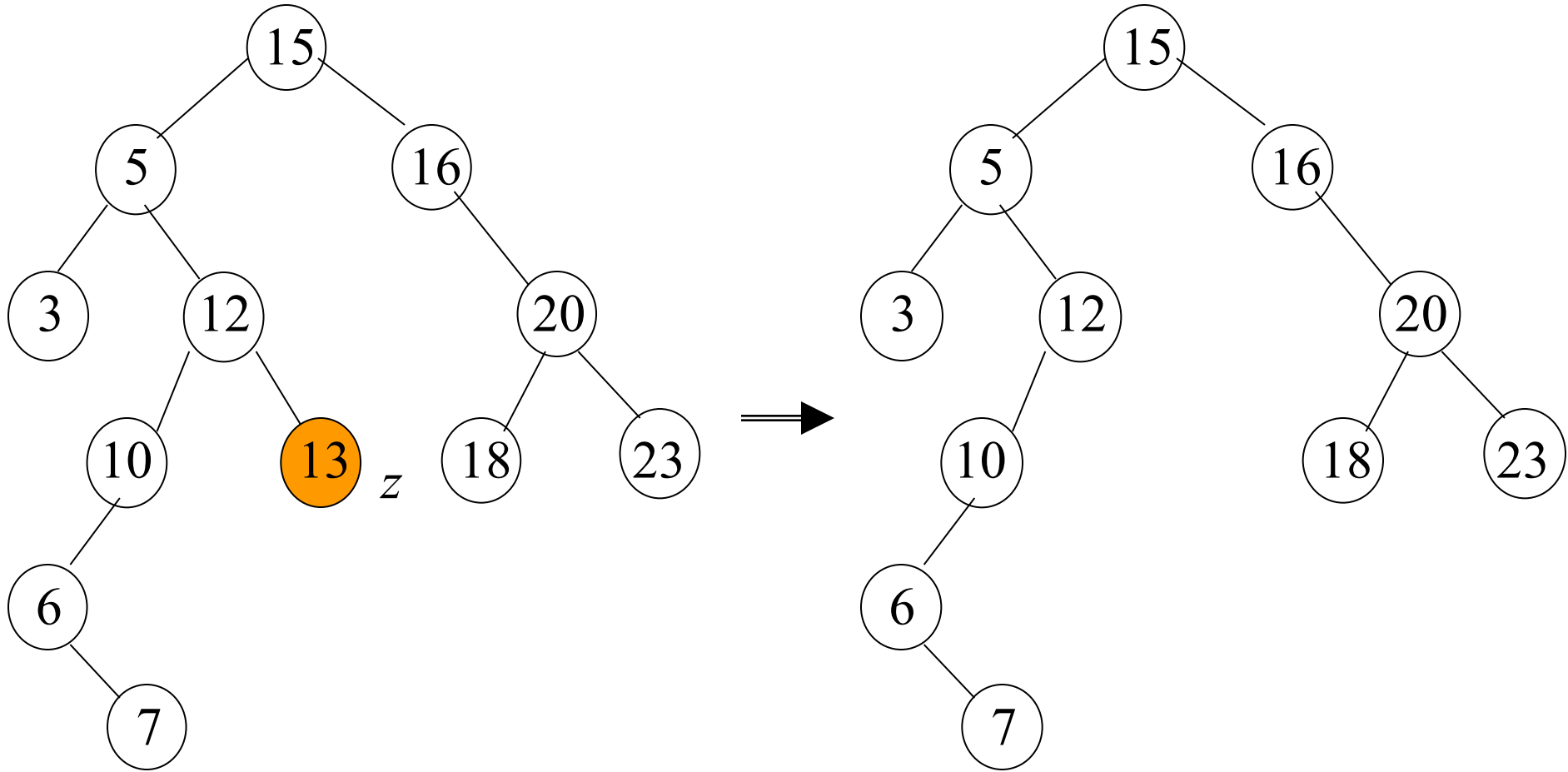
```
Tree-Insert( $T, z$ )  
 $y \leftarrow \text{NIL}$   
 $x \leftarrow \text{root}[T]$   
while  $x \neq \text{NIL}$   
    do  $y \leftarrow x$   
        if  $\text{key}[z] < \text{key}[x]$   
            then  $x \leftarrow \text{left}[x]$   
            else  $x \leftarrow \text{right}[x]$   
 $p[z] \leftarrow y$   
if  $y = \text{NIL}$   
    then  $\text{root}[T] \leftarrow z$   
    else if  $\text{key}[z] < \text{key}[x]$   
        then  $\text{left}[y] \leftarrow z$   
        else  $\text{right}[y] \leftarrow z$ 
```



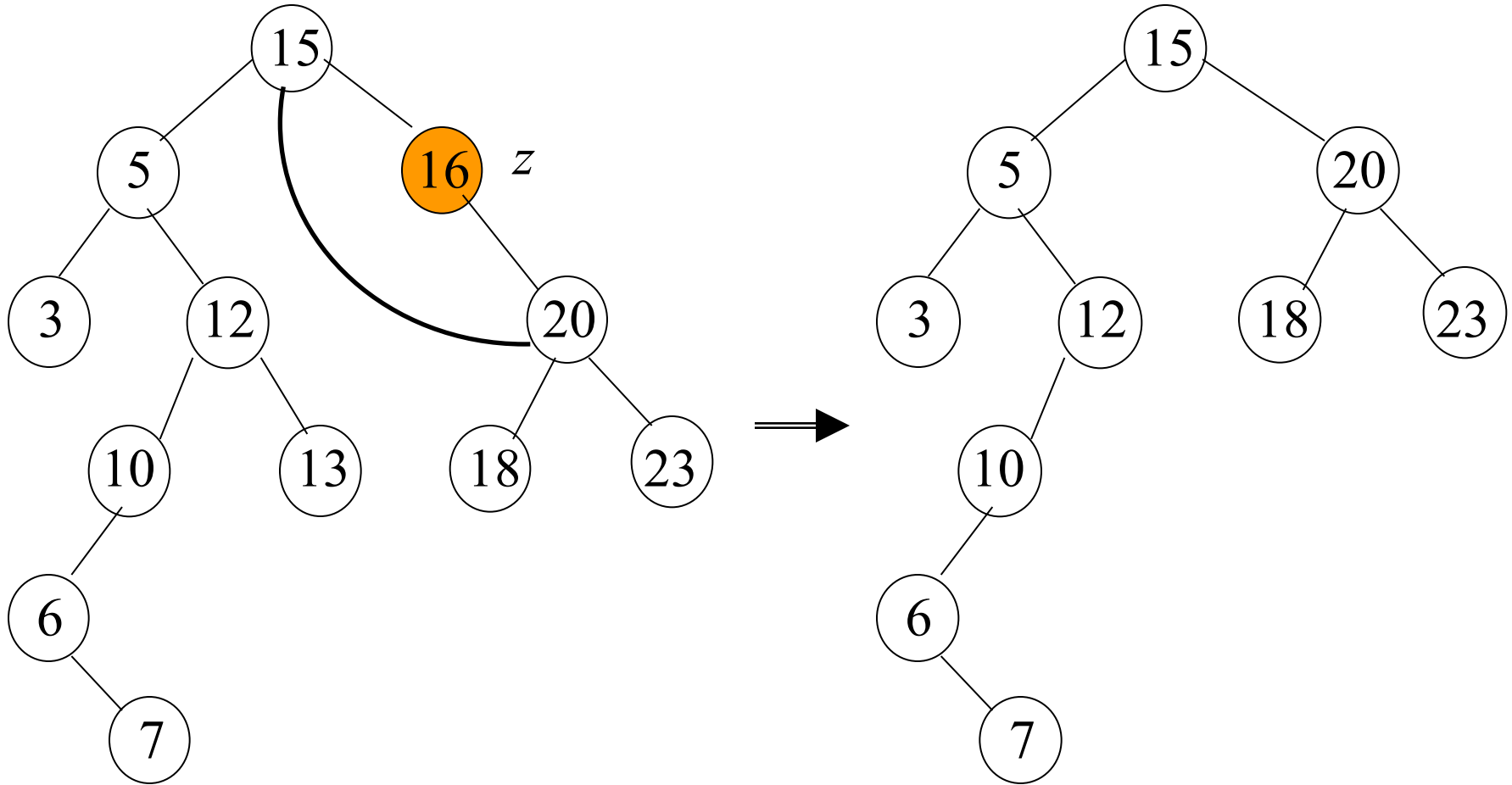
Inserir um nó com chave 13.

Algoritmo de Remoção

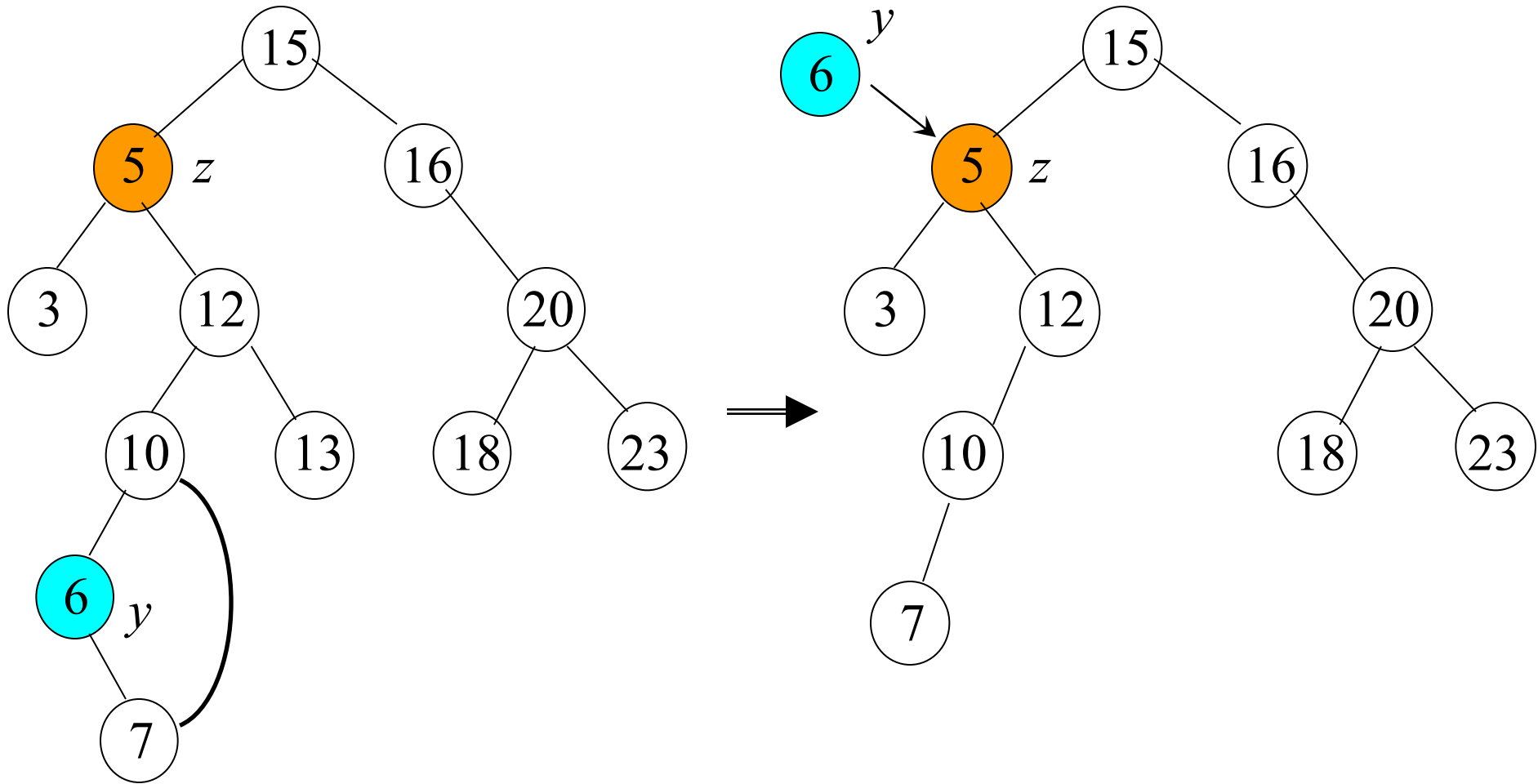
```
Tree-Delete( $T, z$ )
if  $left[z] = \text{NIL}$  ou  $right[z] = \text{NIL}$ 
    then  $y \leftarrow z$ 
    else  $y \leftarrow \text{Tree-Successor}(z)$ 
if  $left[y] \neq \text{NIL}$ 
    then  $x \leftarrow left[y]$ 
    else  $x \leftarrow right[y]$ 
if  $x \neq \text{NIL}$ 
    then  $p[y] \leftarrow p[x]$ 
if  $p[y] = \text{NIL}$ 
    then  $root[T] \leftarrow x$ 
    else if  $y = left[p[y]]$ 
        then  $left[p[y]] \leftarrow x$ 
        else  $right[p[y]] \leftarrow x$ 
if  $y \neq z$ 
    then  $key[z] \leftarrow key[y]$ 
return  $y$ 
```



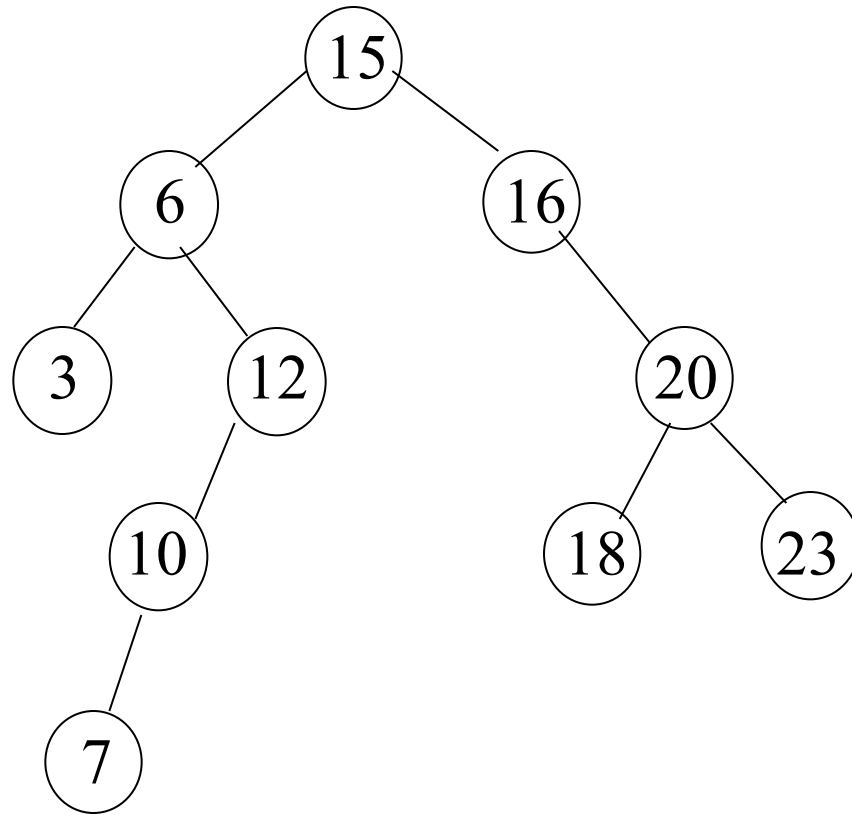
Caso 1: deletar o nó com chave 13.



Caso 2: deletar o nó com chave 16.



Caso 3: deletar o nó com chave 5.



Caso 3: deletar o nó com chave 5.