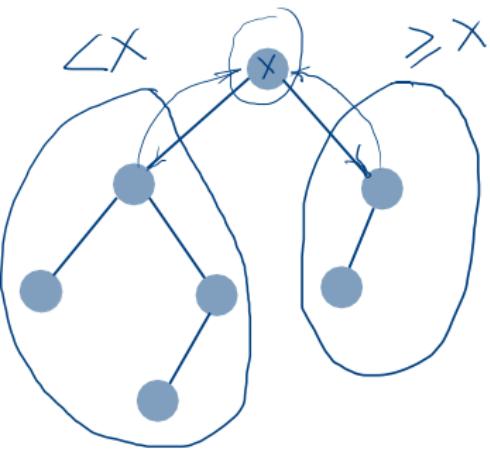


Двоичное дерево поиска



```
struct Node:  
    key  
    left  
    right  
    parent
```

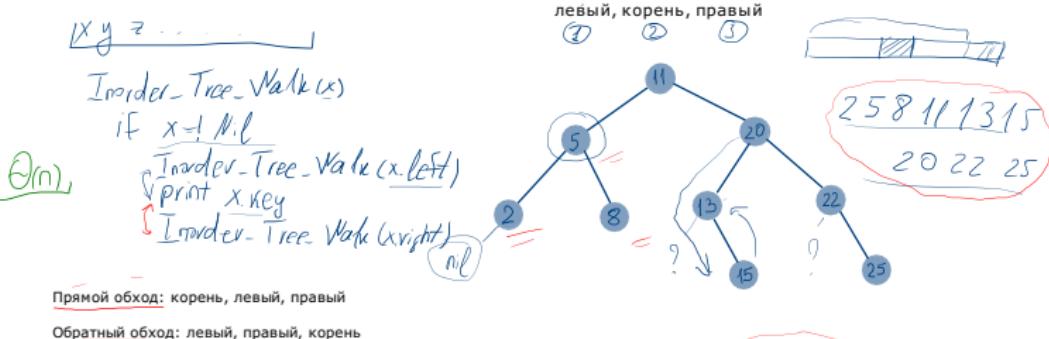
Большинство
операций за

$\Theta(\log n)$

свойство бинарного дерева поиска

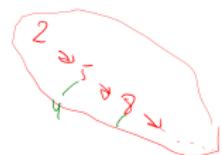
Обходы двоичного дерева поиска

Вывод элементов в отсортированном порядке с помощью **центрированного** обхода



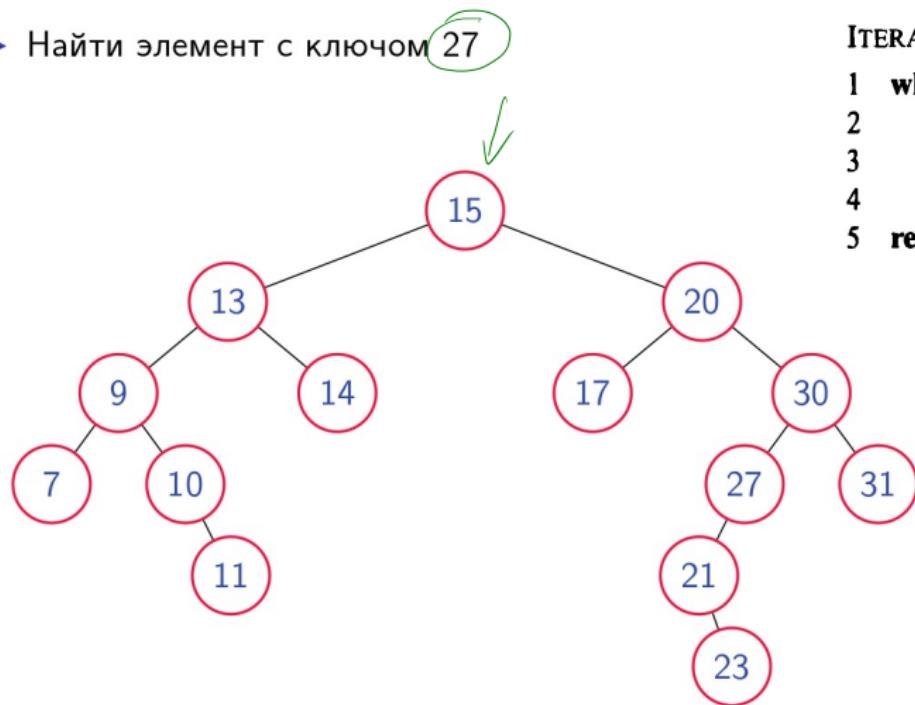
Нужны для описания дерева поиска: центрированный + прямой/обратный

Только прямого и обратного не достаточно для однозначного описания дерева



Операции с двоичным деревом поиска

- ▶ Найти элемент с ключом 27

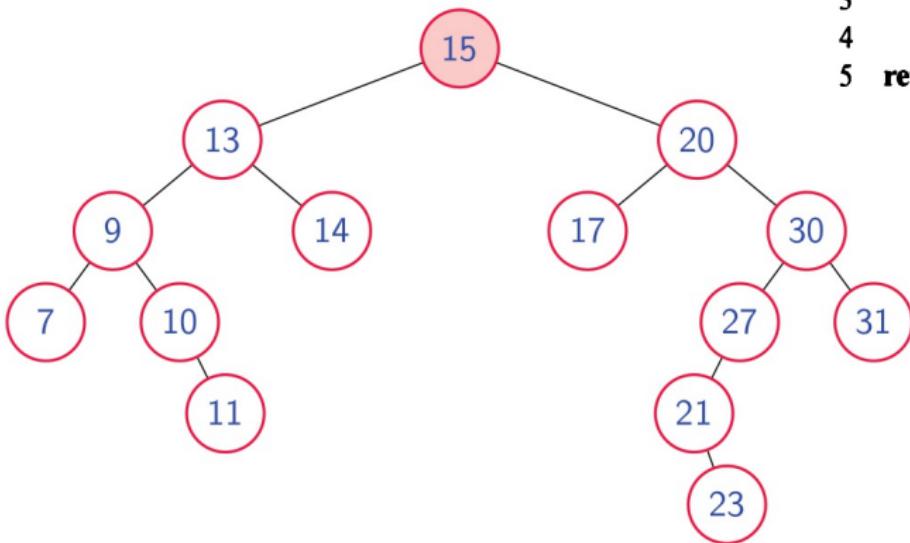


ITERATIVE-TREE-SEARCH(\underline{x}, k)

```
1 while  $x \neq \text{NIL}$  и  $k \neq x.\text{key}$ 
2   if  $k < x.\text{key}$ 
3      $x = x.\text{left}$ 
4   else  $x = x.\text{right}$ 
5 return  $x$ 
```

Операции с двоичным деревом поиска

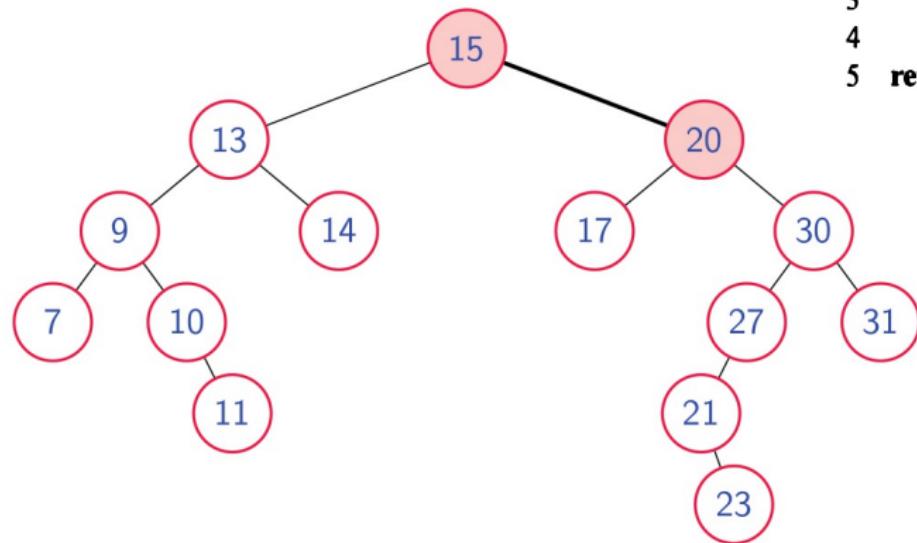
- ▶ Найти элемент с ключом 27



```
ITERATIVE-TREE-SEARCH( $x, k$ )  
1 while  $x \neq \text{NIL}$  и  $k \neq x.\text{key}$   
2   if  $k < x.\text{key}$   
3      $x = x.\text{left}$   
4   else  $x = x.\text{right}$   
5 return  $x$ 
```

Операции с двоичным деревом поиска

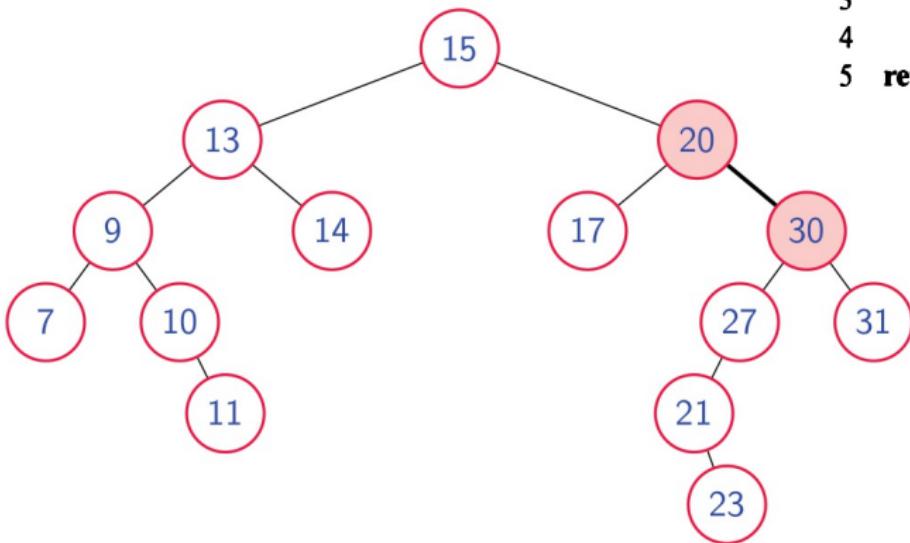
- ▶ Найти элемент с ключом 27



```
ITERATIVE-TREE-SEARCH( $x, k$ )  
1 while  $x \neq \text{NIL}$  and  $k \neq x.\text{key}$   
2   if  $k < x.\text{key}$   
3      $x = x.\text{left}$   
4   else  $x = x.\text{right}$   
5 return  $x$ 
```

Операции с двоичным деревом поиска

- ▶ Найти элемент с ключом 27

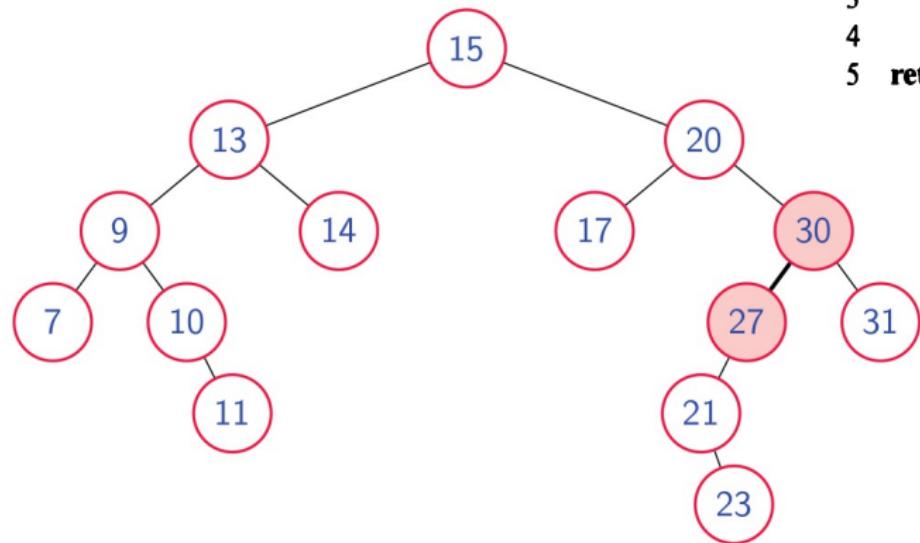


```
ITERATIVE-TREE-SEARCH( $x, k$ )  
1 while  $x \neq \text{NIL}$  и  $k \neq x.\text{key}$   
2   if  $k < x.\text{key}$   
3      $x = x.\text{left}$   
4   else  $x = x.\text{right}$   
5 return  $x$ 
```

(30, 24)

Операции с двоичным деревом поиска

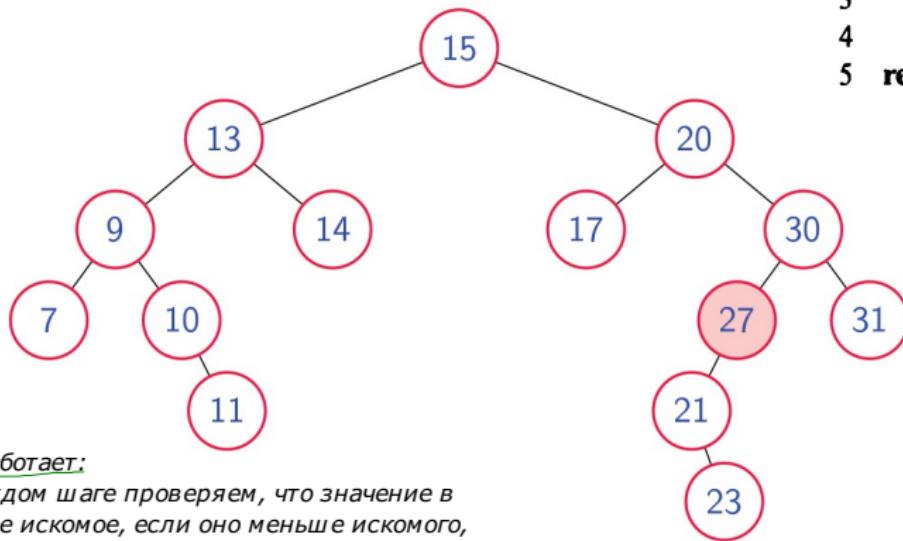
- ▶ Найти элемент с ключом 27



```
ITERATIVE-TREE-SEARCH( $x, k$ )  
1 while  $x \neq \text{NIL}$  и  $k \neq x.\text{key}$   
2   if  $k < x.\text{key}$   
3      $x = x.\text{left}$   
4   else  $x = x.\text{right}$   
5 return  $x$ 
```

Операции с двоичным деревом поиска

- ▶ Найти элемент с ключом 27



```
ITERATIVE-TREE-SEARCH( $x, k$ )  
1 while  $x \neq \text{NIL}$  и  $k \neq x.\text{key}$   
2     if  $k < x.\text{key}$   
3          $x = x.\text{left}$   
4     else  $x = x.\text{right}$   
5     return  $x$ 
```

$\mathcal{O}(\log n)$

Как работает:

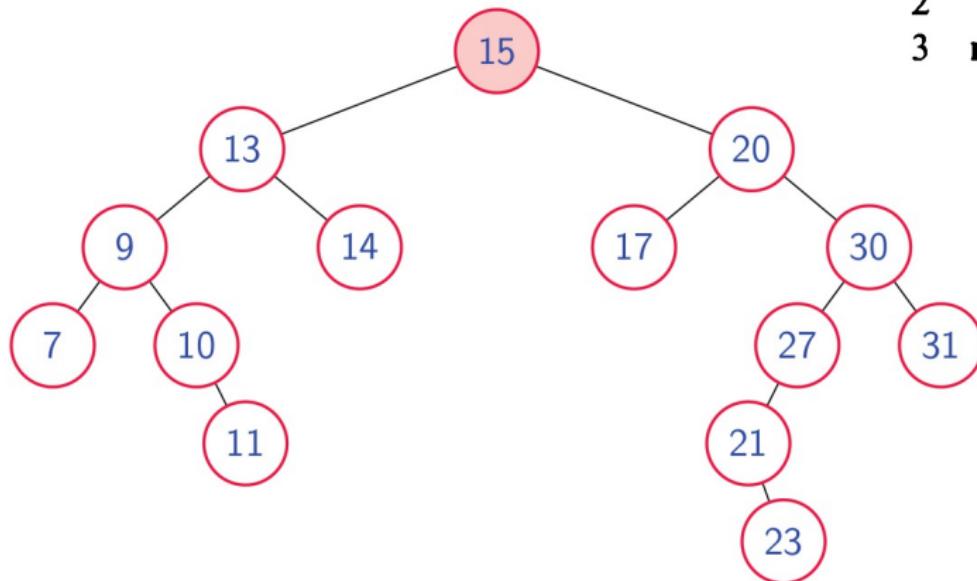
на каждом шаге проверяем, что значение в узле не искомое, если оно меньше искомого, то переход в правое поддерево, иначе — в левое

Операции с двоичным деревом поиска

- ▶ Найти минимальный элемент дерева

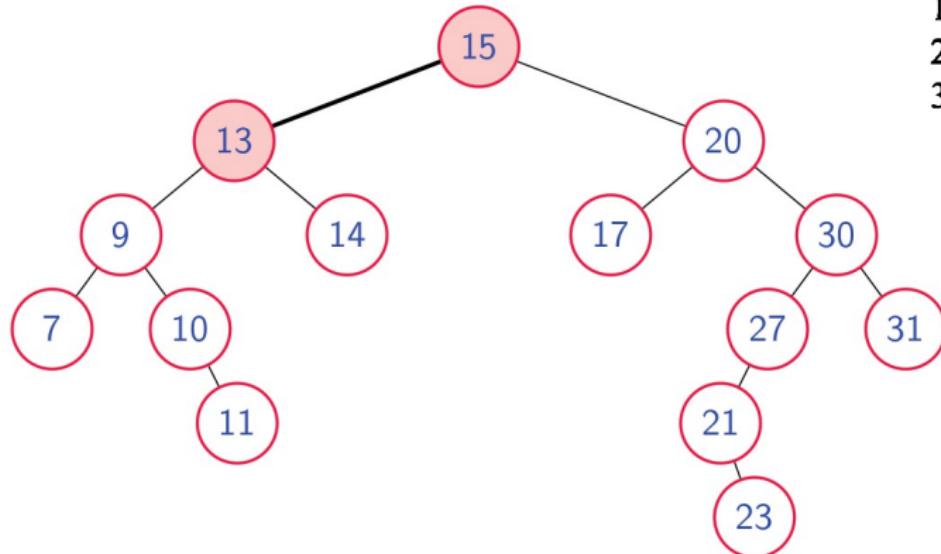
TREE-MINIMUM(x)

```
1 while  $x.\text{left} \neq \text{NIL}$ 
2      $x = x.\text{left}$ 
3 return  $x$ 
```



Операции с двоичным деревом поиска

- ▶ Найти минимальный элемент дерева



TREE-MINIMUM(x)

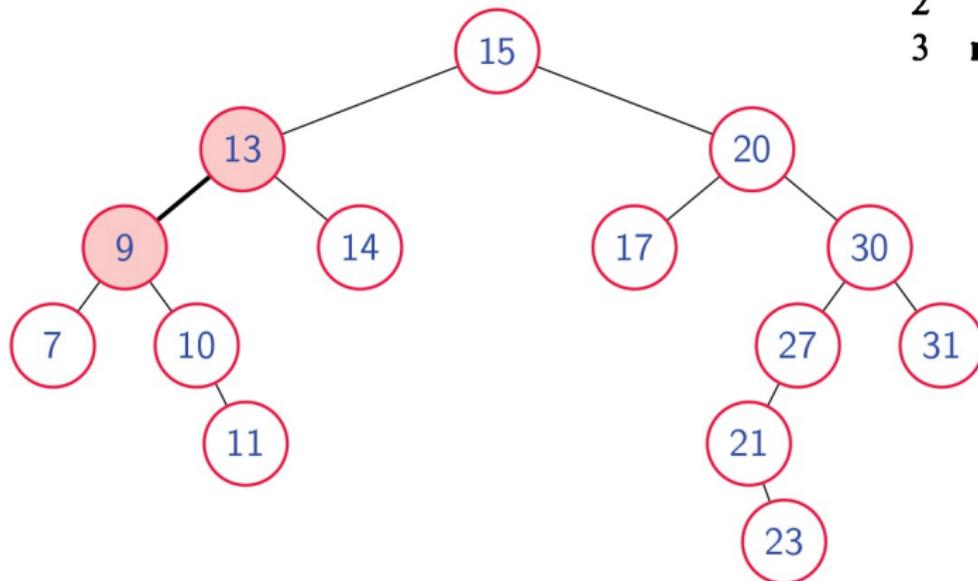
```
1 while  $x.\text{left} \neq \text{NIL}$ 
2      $x = x.\text{left}$ 
3 return  $x$ 
```

Операции с двоичным деревом поиска

- ▶ Найти минимальный элемент дерева

TREE-MINIMUM(x)

```
1 while  $x.\text{left} \neq \text{NIL}$ 
2      $x = x.\text{left}$ 
3 return  $x$ 
```

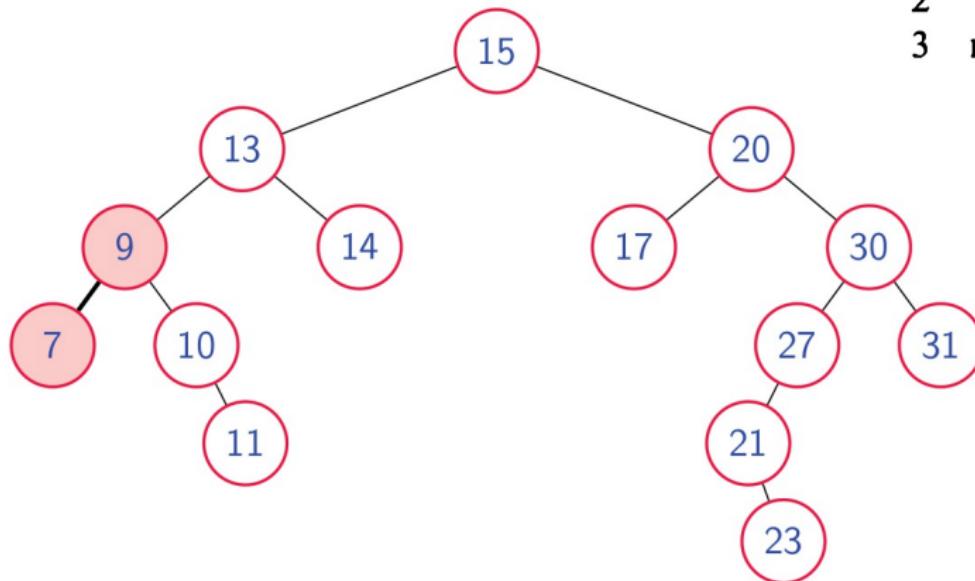


Операции с двоичным деревом поиска

- ▶ Найти минимальный элемент дерева

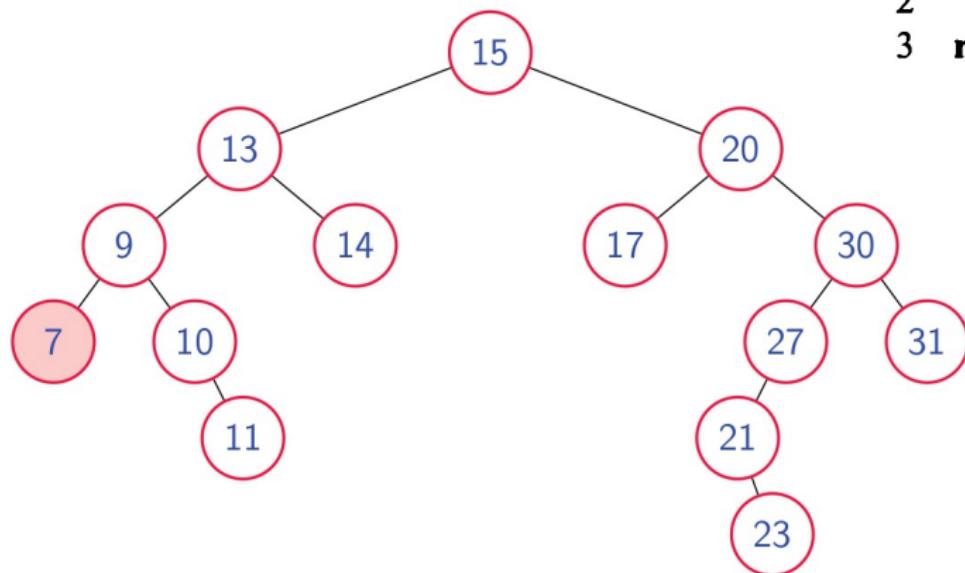
TREE-MINIMUM(x)

```
1 while  $x.\text{left} \neq \text{NIL}$ 
2      $x = x.\text{left}$ 
3 return  $x$ 
```



Операции с двоичным деревом поиска

- ▶ Найти минимальный элемент дерева



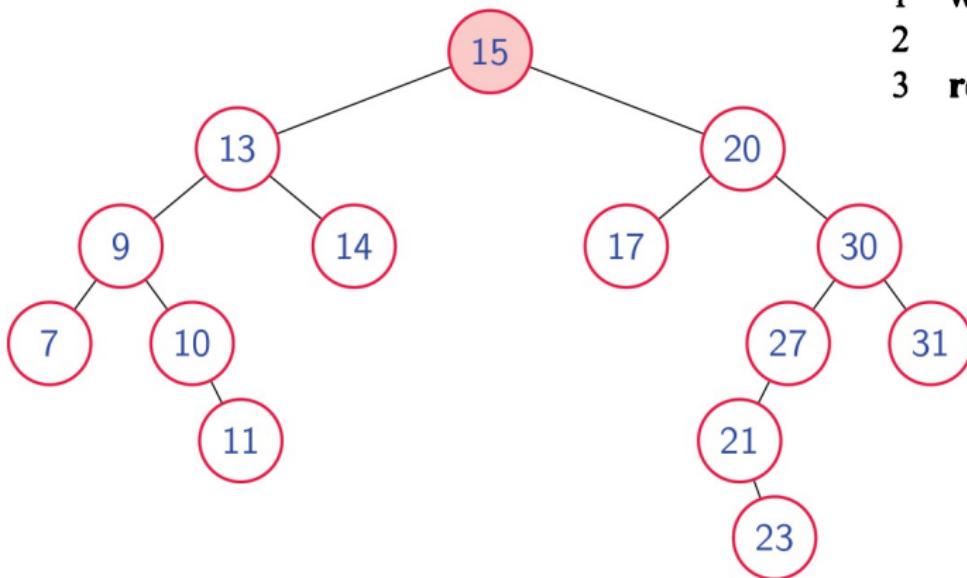
TREE-MINIMUM(x)

```
1 while  $x.\text{left} \neq \text{NIL}$ 
2      $x = x.\text{left}$ 
3 return  $x$ 
```

$\mathcal{O}(\log n)$

Операции с двоичным деревом поиска

- ▶ Найти максимальный элемент дерева

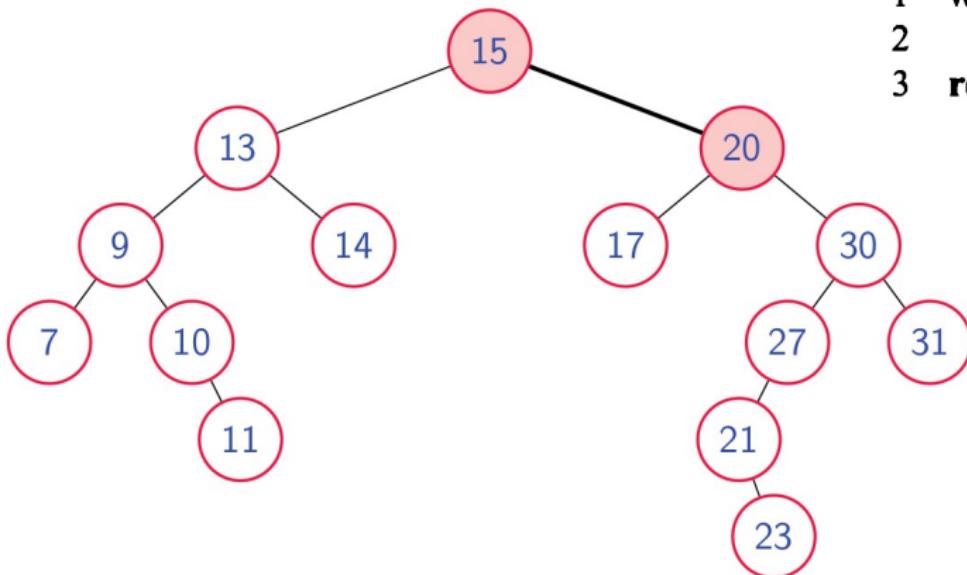


TREE-MAXIMUM(x)

```
1 while  $x.\text{right} \neq \text{NIL}$ 
2    $x = x.\text{right}$ 
3 return  $x$ 
```

Операции с двоичным деревом поиска

- ▶ Найти максимальный элемент дерева

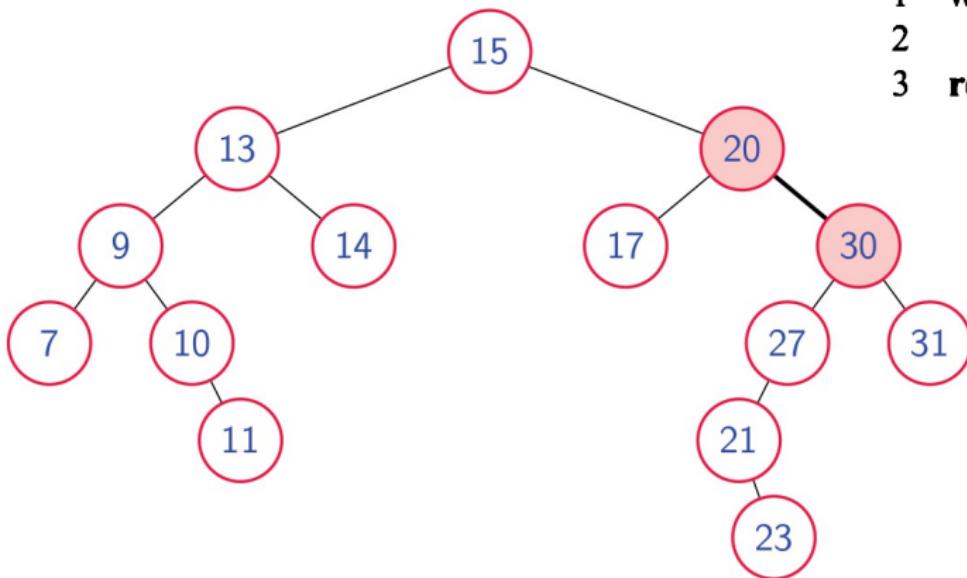


TREE-MAXIMUM(x)

```
1 while  $x.\text{right} \neq \text{NIL}$ 
2      $x = x.\text{right}$ 
3 return  $x$ 
```

Операции с двоичным деревом поиска

- ▶ Найти максимальный элемент дерева

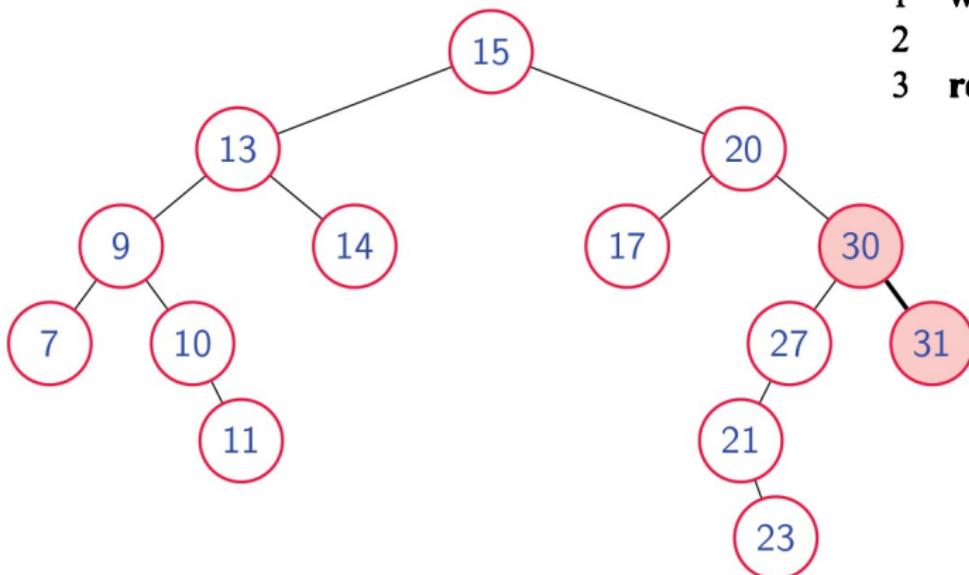


TREE-MAXIMUM(x)

```
1 while  $x.\text{right} \neq \text{NIL}$ 
2      $x = x.\text{right}$ 
3 return  $x$ 
```

Операции с двоичным деревом поиска

- ▶ Найти максимальный элемент дерева

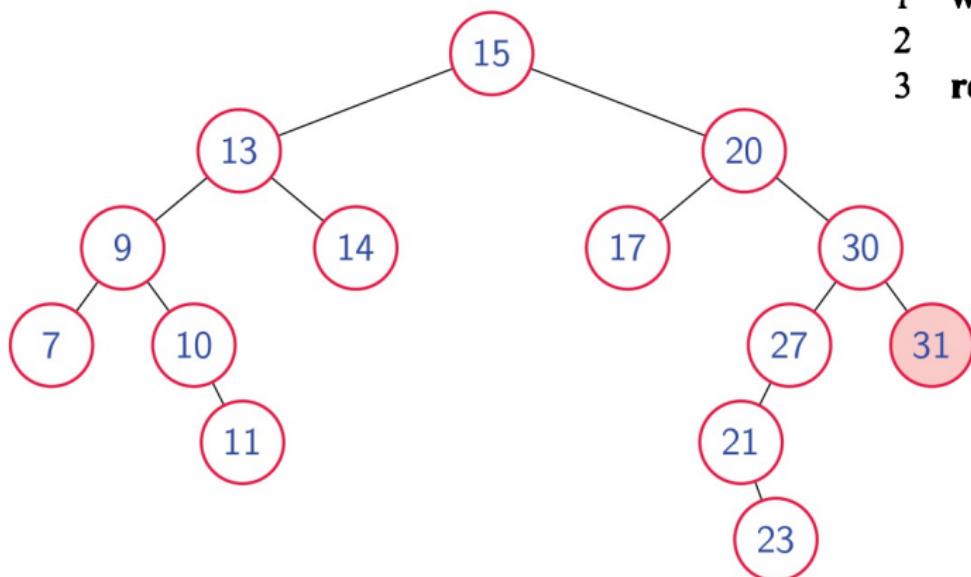


TREE-MAXIMUM(x)

```
1 while  $x.\text{right} \neq \text{NIL}$ 
2    $x = x.\text{right}$ 
3 return  $x$ 
```

Операции с двоичным деревом поиска

- ▶ Найти максимальный элемент дерева



TREE-MAXIMUM(x)

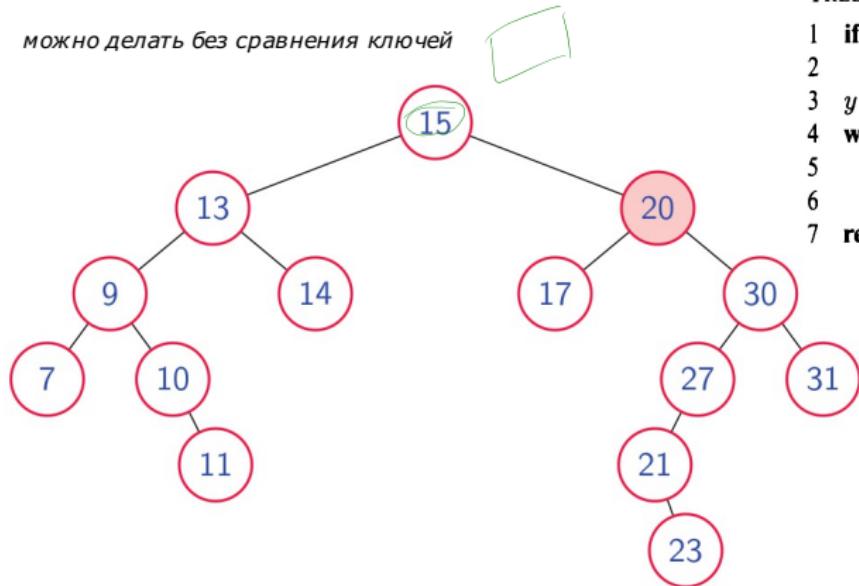
```
1 while  $x.\text{right} \neq \text{NIL}$ 
2    $x = x.\text{right}$ 
3 return  $x$ 
```

Операции с двоичным деревом поиска

20
π

- ▶ Найти последующий за 20

можно делать без сравнения ключей



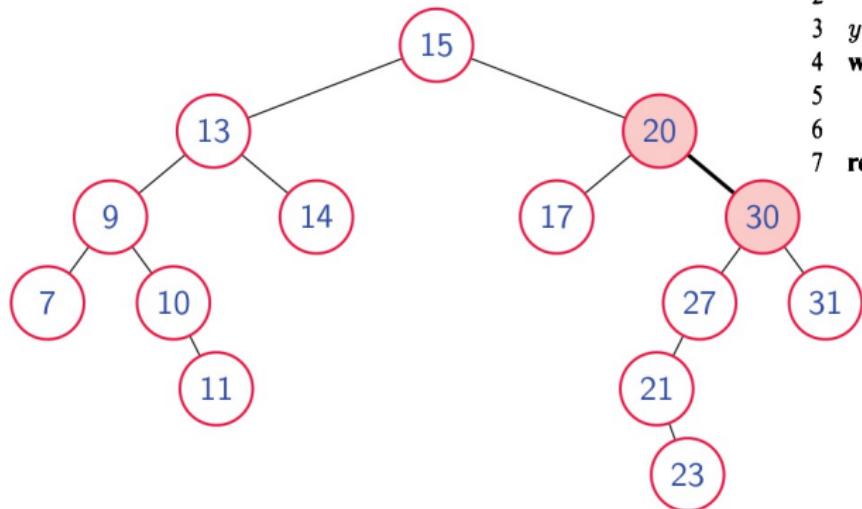
TREE-SUCCESSOR(x)

```
1 if  $x.right \neq NIL$ 
2   return TREE-MINIMUM( $x.right$ )
3  $y = x.p$ 
4 while  $y \neq NIL$  и  $x == y.right$ 
5    $x = y$ 
6    $y = y.p$ 
7 return  $y$ 
```

Операции с двоичным деревом поиска

- ▶ Найти последующий за 20

можно делать без сравнения ключей



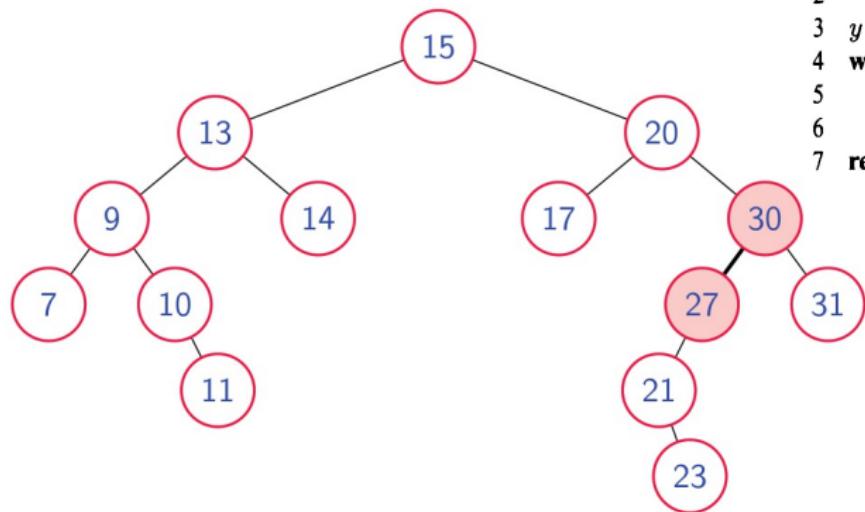
TREE-SUCCESSOR(x)

```
1 if  $x.\text{right} \neq \text{NIL}$ 
2   return TREE-MINIMUM( $x.\text{right}$ )
3  $y = x.p$ 
4 while  $y \neq \text{NIL}$  и  $x == y.\text{right}$ 
5    $x = y$ 
6    $y = y.p$ 
7 return  $y$ 
```

Операции с двоичным деревом поиска

- ▶ Найти последующий за 20

можно делать без сравнения ключей



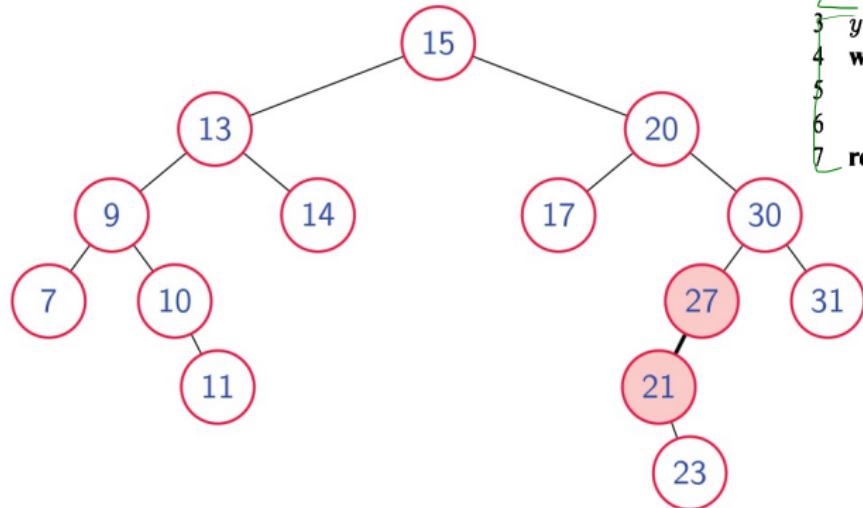
TREE-SUCCESSOR(x)

```
1 if  $x.\text{right} \neq \text{NIL}$ 
2   return TREE-MINIMUM( $x.\text{right}$ )
3  $y = x.p$ 
4 while  $y \neq \text{NIL}$  и  $x == y.\text{right}$ 
5    $x = y$ 
6    $y = y.p$ 
7 return  $y$ 
```

Операции с двоичным деревом поиска

- ▶ Найти последующий за 20

можно делать без сравнения ключей

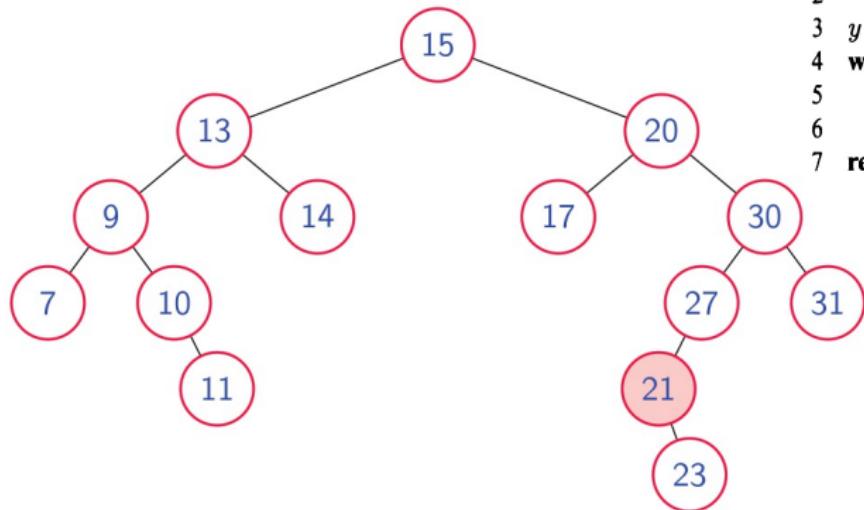


```
TREE-SUCCESSOR(x)
1   if x.right ≠ NIL
2       return TREE-MINIMUM(x.right)
3   y = x.p
4   while y ≠ NIL и x == y.right
5       x = y
6       y = y.p
7   return y
```

Операции с двоичным деревом поиска

- ▶ Найти последующий за 20

можно делать без сравнения ключей



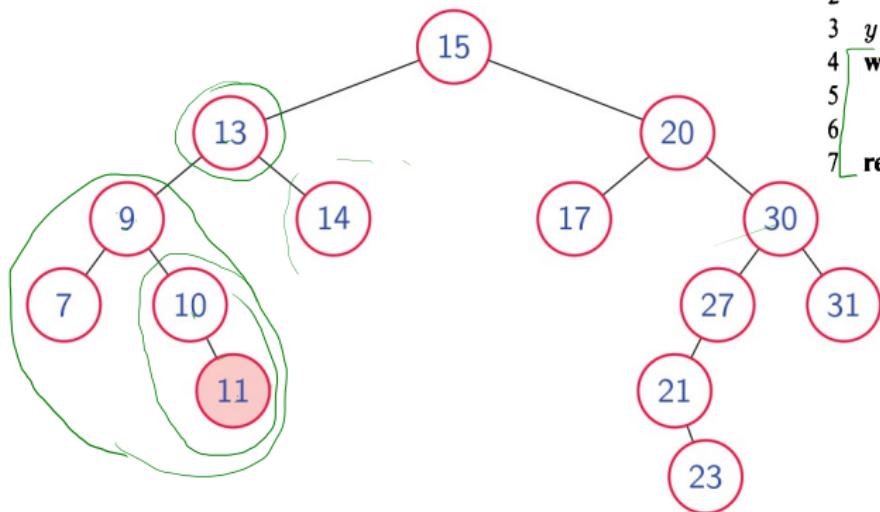
TREE-SUCCESSOR(x)

```
1 if  $x.right \neq \text{NIL}$ 
2   return TREE-MINIMUM( $x.right$ )
3  $y = x.p$ 
4 while  $y \neq \text{NIL}$  и  $x == y.right$ 
5    $x = y$ 
6    $y = y.p$ 
7 return  $y$ 
```

Операции с двоичным деревом поиска

- ▶ Найти последующий за 11

можно делать без сравнения ключей



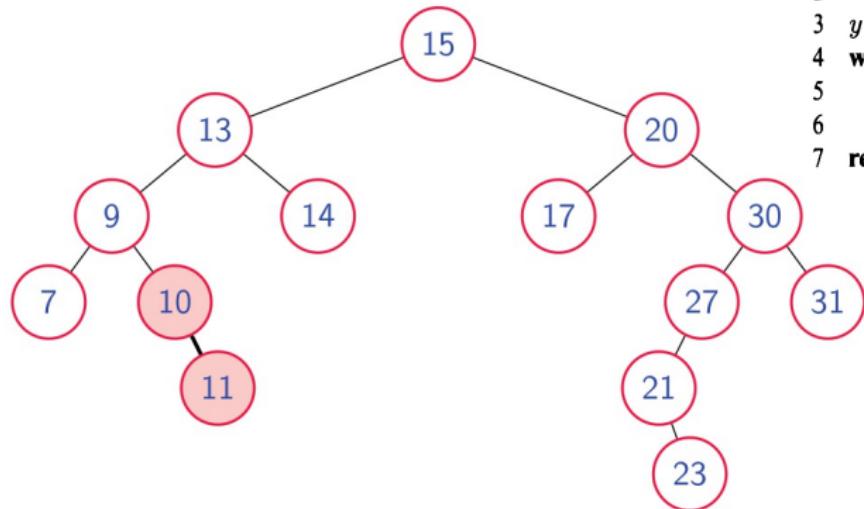
TREE-SUCCESSOR(x)

```
1 if  $x.\text{right} \neq \text{NIL}$ 
2   return TREE-MINIMUM( $x.\text{right}$ )
3  $y = x.p$ 
4 while  $y \neq \text{NIL}$  и  $x == y.\text{right}$ 
5    $x = y$ 
6    $y = y.p$ 
7 return  $y$ 
```

Операции с двоичным деревом поиска

- ▶ Найти последующий за 11

можно делать без сравнения ключей

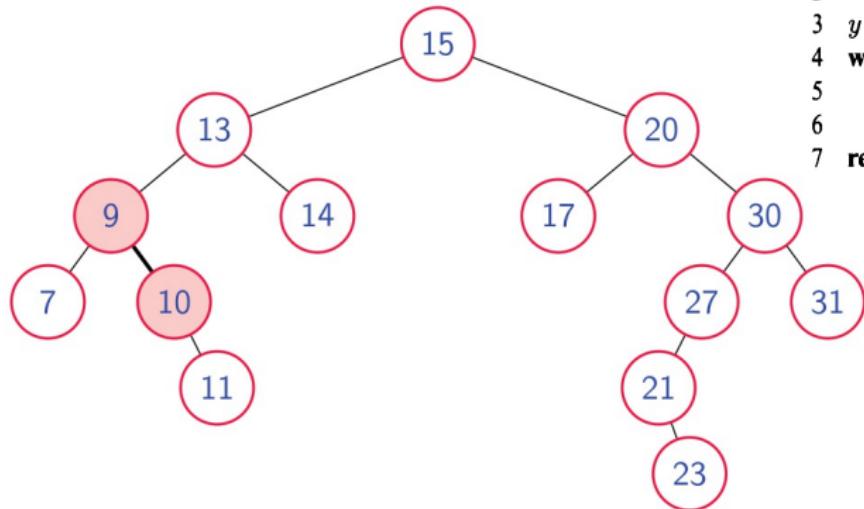


```
TREE-SUCCESSOR(x)
1 if x.right ≠ NIL
2   return TREE-MINIMUM(x.right)
3 y = x.p
4 while y ≠ NIL и x == y.right
5   x = y
6   y = y.p
7 return y
```

Операции с двоичным деревом поиска

- ▶ Найти последующий за 11

можно делать без сравнения ключей

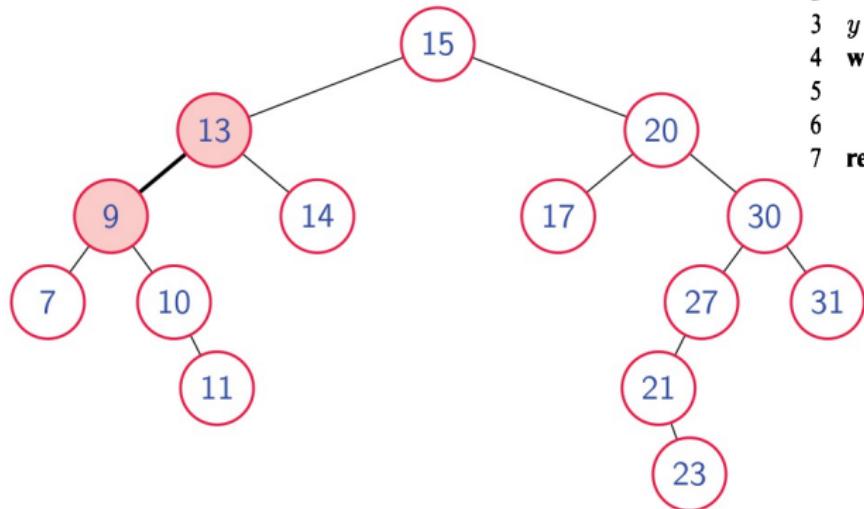


```
TREE-SUCCESSOR(x)
1 if x.right ≠ NIL
2   return TREE-MINIMUM(x.right)
3 y = x.p
4 while y ≠ NIL и x == y.right
5   x = y
6   y = y.p
7 return y
```

Операции с двоичным деревом поиска

- ▶ Найти последующий за 11

можно делать без сравнения ключей



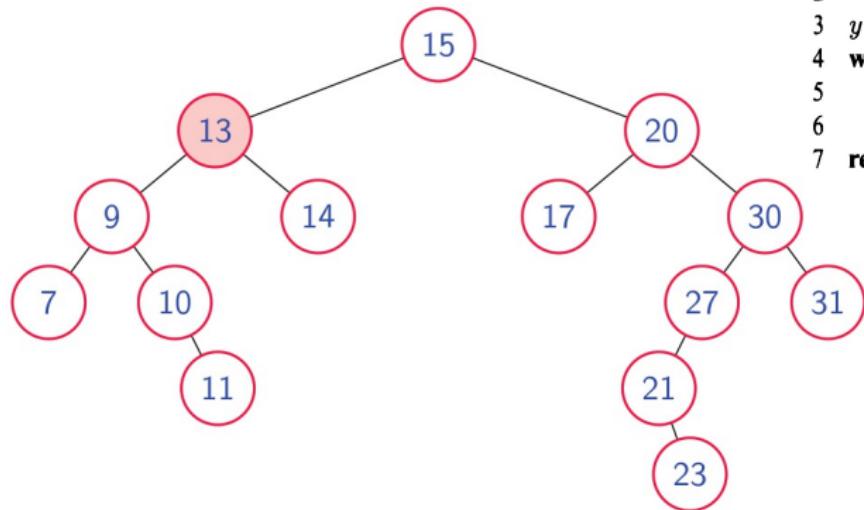
TREE-SUCCESSOR(x)

```
1 if  $x.\text{right} \neq \text{NIL}$ 
2   return TREE-MINIMUM( $x.\text{right}$ )
3  $y = x.p$ 
4 while  $y \neq \text{NIL}$  и  $x == y.\text{right}$ 
5    $x = y$ 
6    $y = y.p$ 
7 return  $y$ 
```

Операции с двоичным деревом поиска

- ▶ Найти последующий за 11

можно делать без сравнения ключей

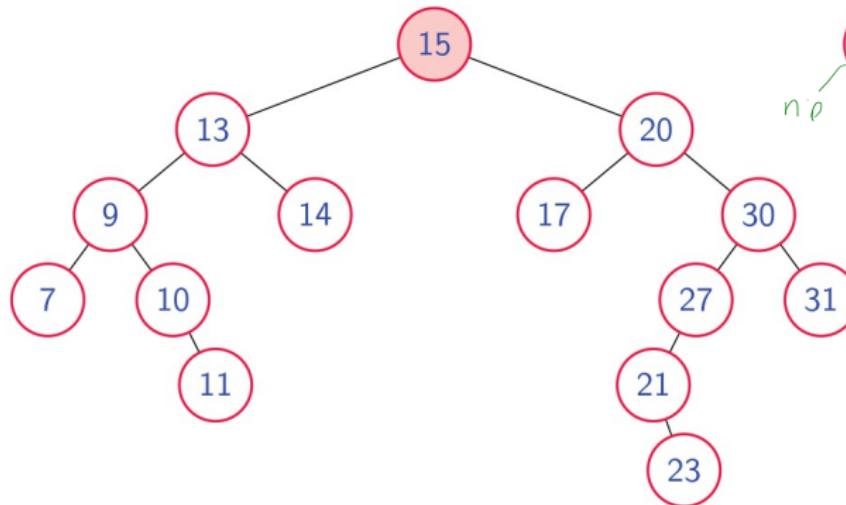


TREE-SUCCESSOR(x)

```
1 if  $x.\text{right} \neq \text{NIL}$ 
2   return TREE-MINIMUM( $x.\text{right}$ )
3  $y = x.p$ 
4 while  $y \neq \text{NIL}$  и  $x == y.\text{right}$ 
5    $x = y$ 
6    $y = y.p$ 
7 return  $y$ 
```

Операции с двоичным деревом поиска

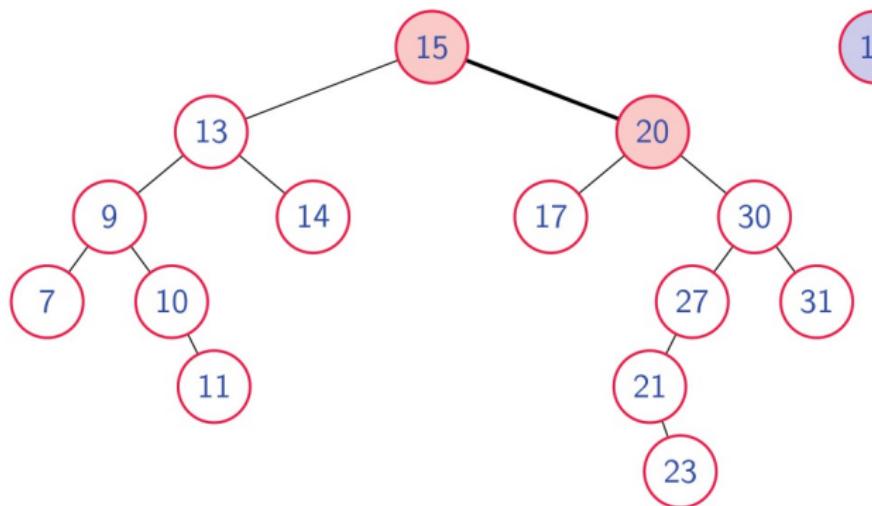
- ▶ Вставить элемент 16



z
nil
Insert(x, z)
if $x == \text{nil}$
return z
if $z.\text{key} < x.\text{key}$
 $x.\text{left} = \text{Insert}(x.\text{left}, z)$
else
 $x.\text{right} = \text{Insert}(x.\text{right}, z)$
return x

Операции с двоичным деревом поиска

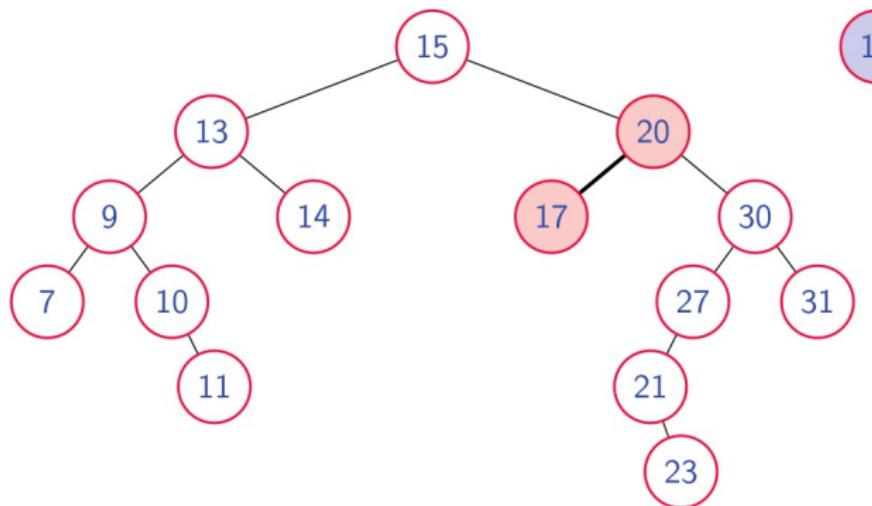
- ▶ Вставить элемент 16



```
Insert(x, z)
if x == nil
    return z
if z.key < x.key
    x.left = Insert(x.left, z)
else
    x.right = Insert(x.right, z)
return x
```

Операции с двоичным деревом поиска

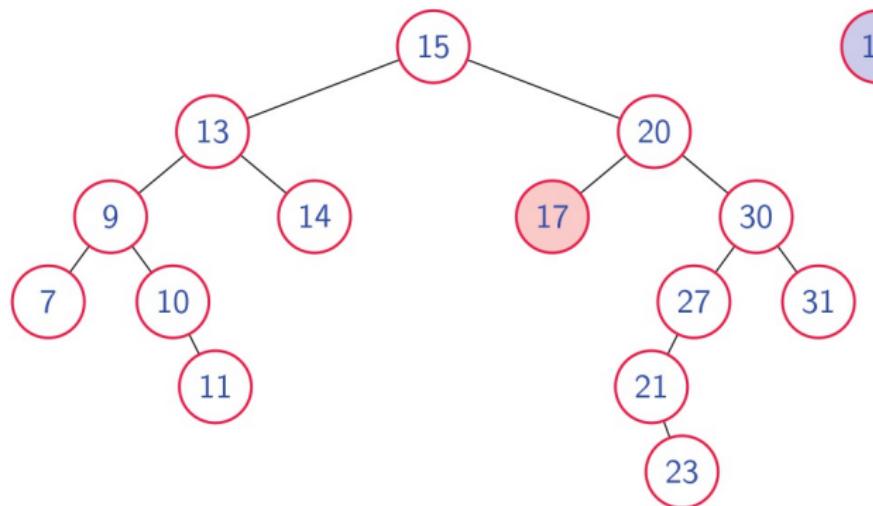
- ▶ Вставить элемент 16



```
Insert(x, z)
if x == nil
    return z
if z.key < x.key
    x.left = Insert(x.left, z)
else
    x.right = Insert(x.right, z)
return x
```

Операции с двоичным деревом поиска

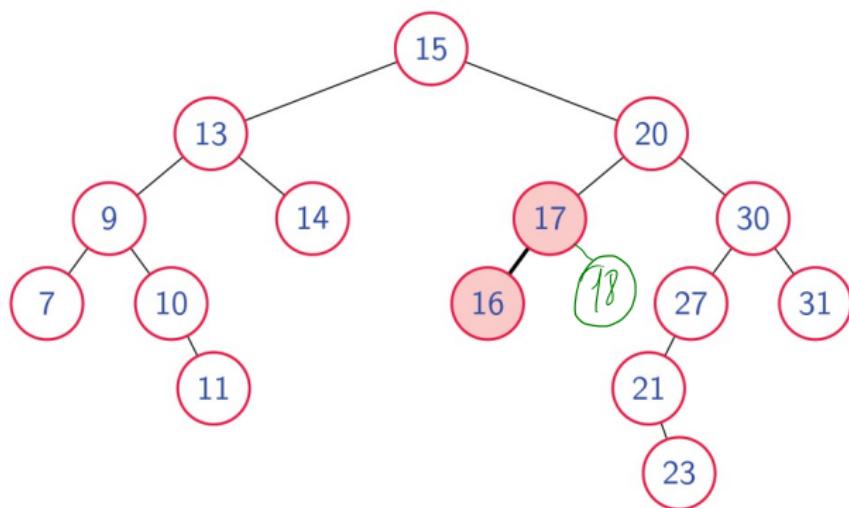
- ▶ Вставить элемент 16



```
Insert(x, z)
if x == nil
    return z
if z.key < x.key
    x.left = Insert(x.left, z)
else
    x.right = Insert(x.right, z)
return x
```

Операции с двоичным деревом поиска

- ▶ Вставить элемент 16



18

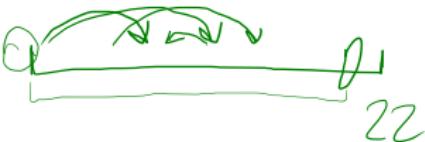
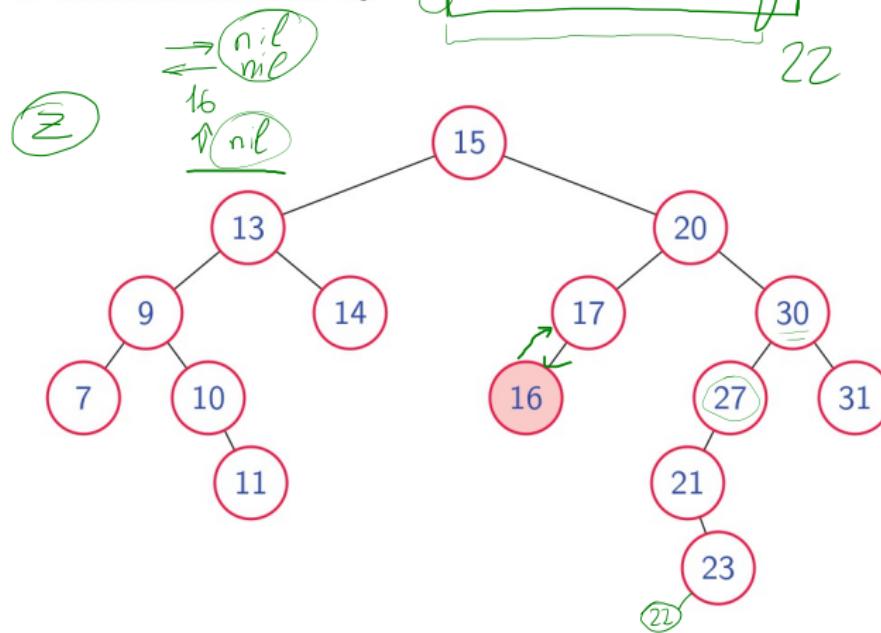
$\Theta(\log n)$.

```
Insert(x, z)
if x == nil
    return z
if z.key < x.key
    x.left = Insert(x.left, z)
else
    x.right = Insert(x.right, z)
return x
```

1/1

Операции с двоичным деревом поиска

- ▶ Вставить элемент 16



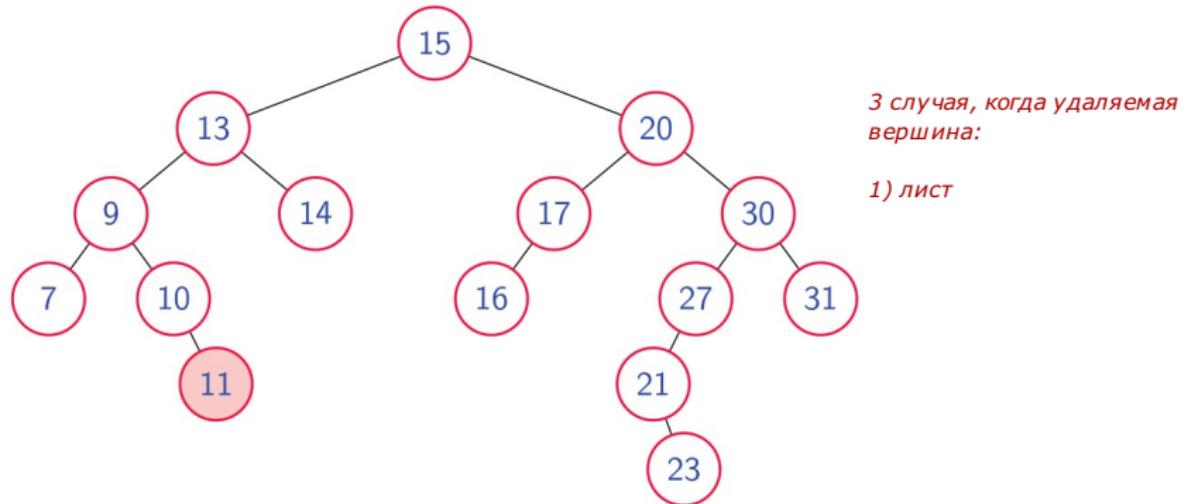
Node.



Insert (x, z)
if $x == \text{nil}$
return z
if $z.\text{key} < x.\text{key}$
 $x.\text{left} = \text{Insert}(x.\text{left}, z)$
else
 $x.\text{right} = \text{Insert}(x.\text{right}, z)$
return x

Операции с двоичным деревом поиска

- ▶ Удалить элемент 11

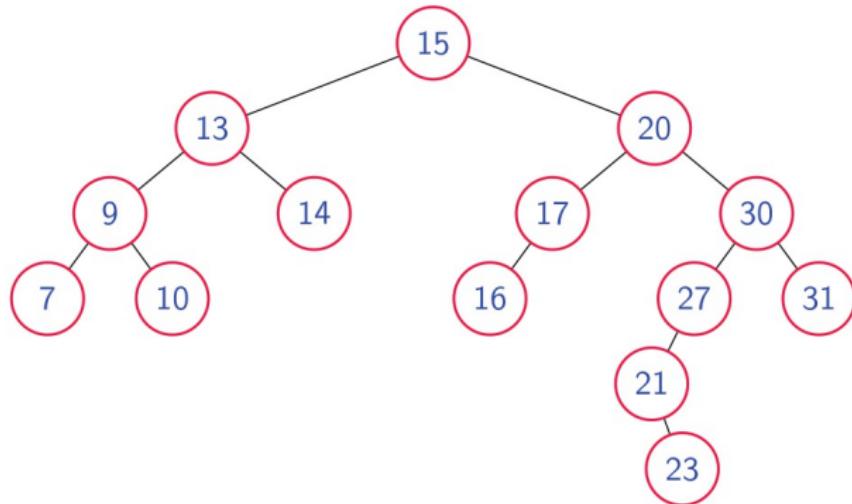


3 случая, когда удаляемая
вершина:

1) лист

Операции с двоичным деревом поиска

- ▶ Удалить элемент 11

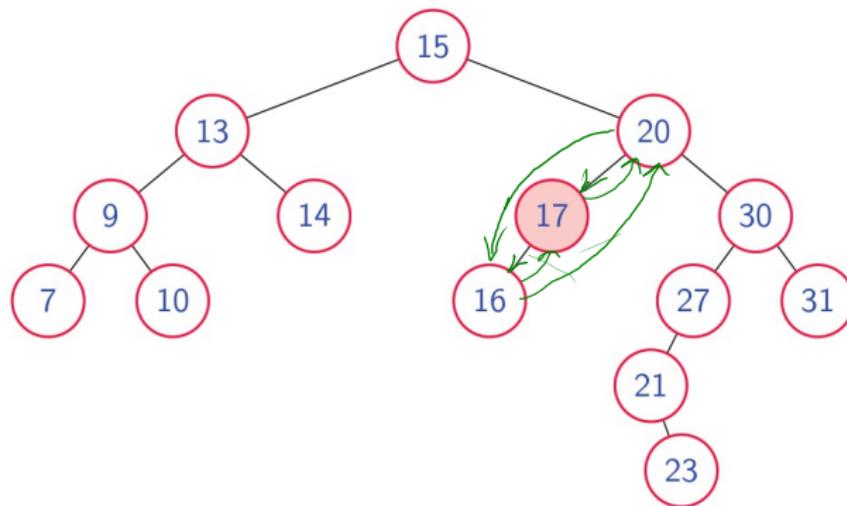


3 случая, когда удаляемая вершина:

1) лист - убрать из дерева (убрать у родителя указатель на неё)

Операции с двоичным деревом поиска

- ▶ Удалить элемент 17

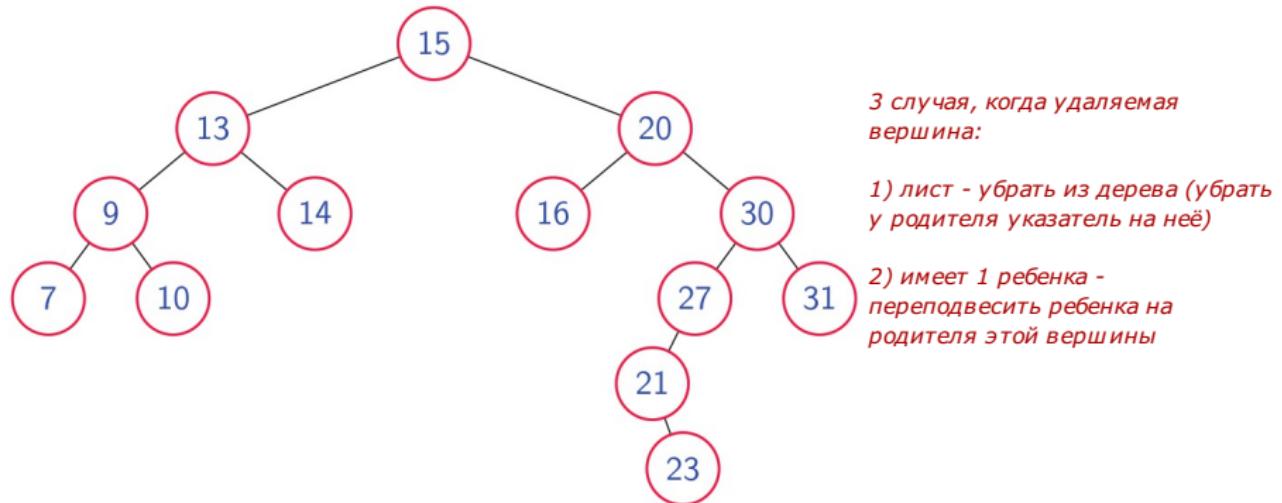


3 случая, когда удаляемая вершина:

- 1) лист - убрать из дерева (убрать у родителя указатель на неё)
- 2) имеет 1 ребенка

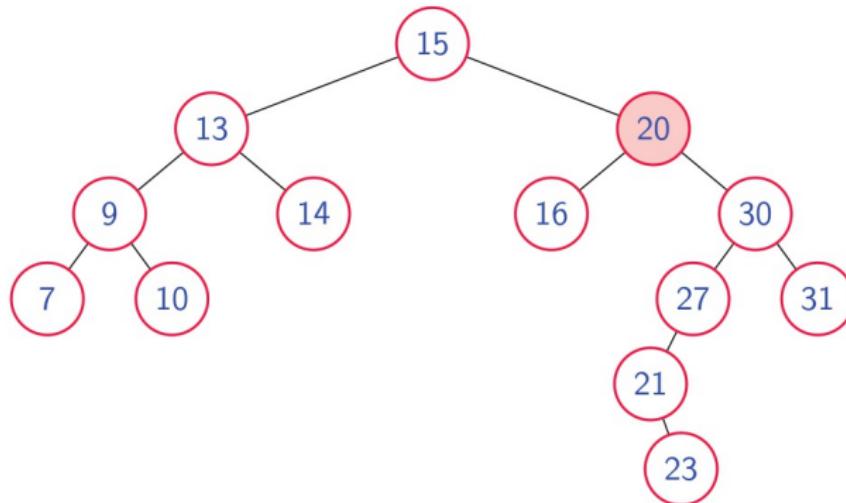
Операции с двоичным деревом поиска

- ▶ Удалить элемент 17



Операции с двоичным деревом поиска

- ▶ Удалить элемент 20



3 случая, когда удаляемая вершина:

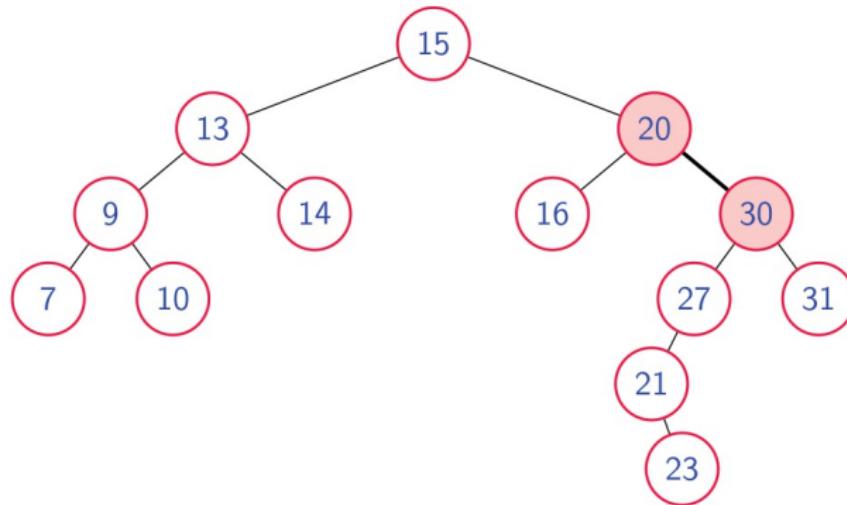
1) лист - убрать из дерева (убрать у родителя указатель на неё)

2) имеет 1 ребенка - переподвесить ребенка на родителя этой вершины

3) имеет 2 ребенка

Операции с двоичным деревом поиска

- ▶ Удалить элемент 20. Поиск последующего.

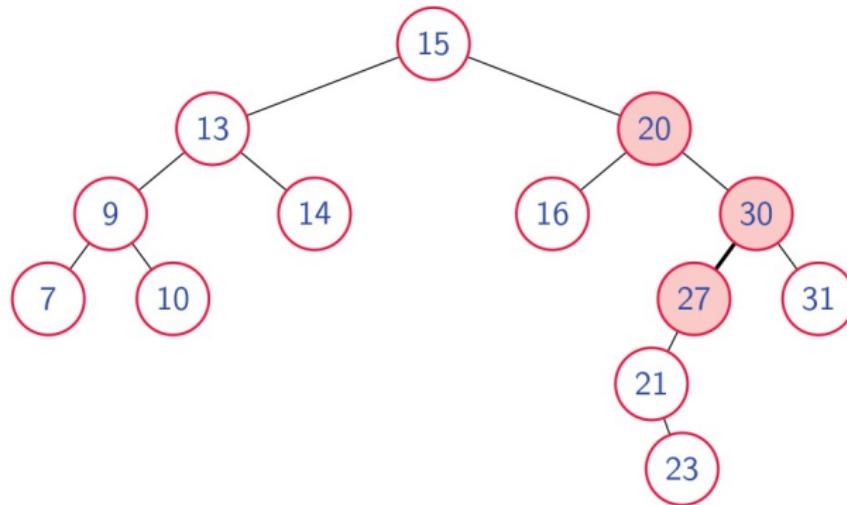


3 случая, когда удаляемая вершина:

- 1) лист - убрать из дерева (убрать у родителя указатель на неё)
- 2) имеет 1 ребенка - переподвесить ребенка на родителя этой вершины
- 3) имеет 2 ребенка - найти последующий (у него точно нет левого ребенка) и записать его значение на место удаляемого, а сам последующий удалить, как в варианте 1 или 2.

Операции с двоичным деревом поиска

- Удалить элемент 20. Поиск последующего.

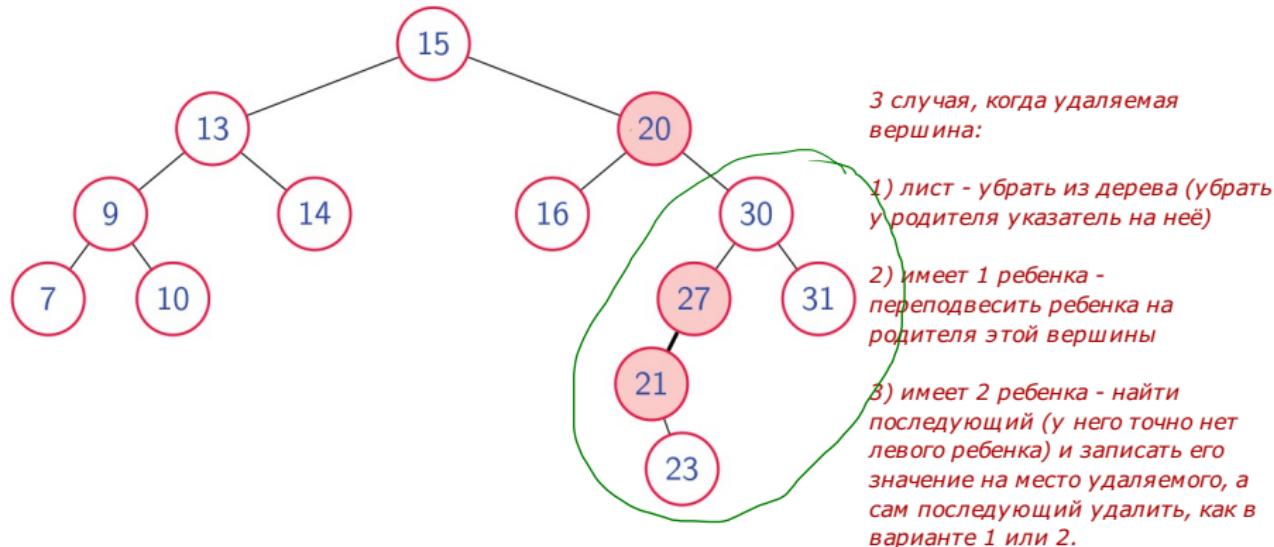


3 случая, когда удаляемая вершина:

- 1) лист - убрать из дерева (убрать у родителя указатель на неё)
- 2) имеет 1 ребенка - переподвесить ребенка на родителя этой вершины
- 3) имеет 2 ребенка - найти последующий (у него точно нет левого ребенка) и записать его значение на место удаляемого, а сам последующий удалить, как в варианте 1 или 2.

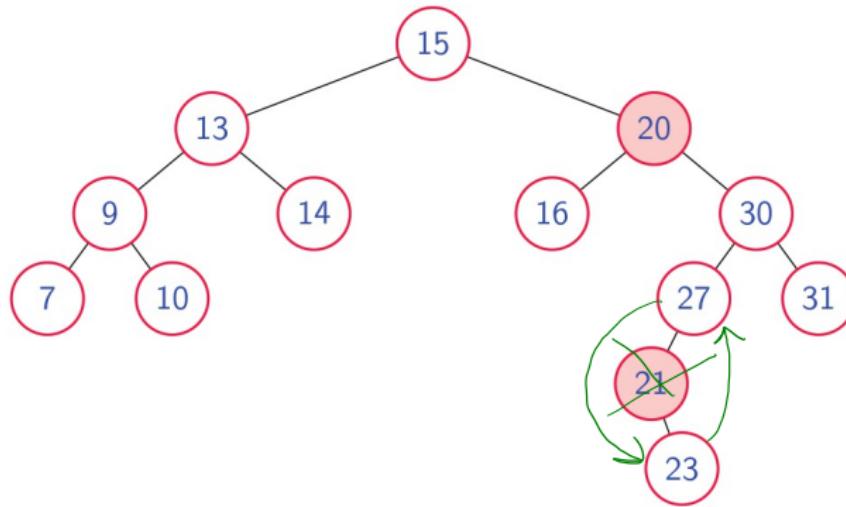
Операции с двоичным деревом поиска

- ▶ Удалить элемент 20. Поиск последующего.



Операции с двоичным деревом поиска

- ▶ Удалить элемент 20. Поиск последующего.

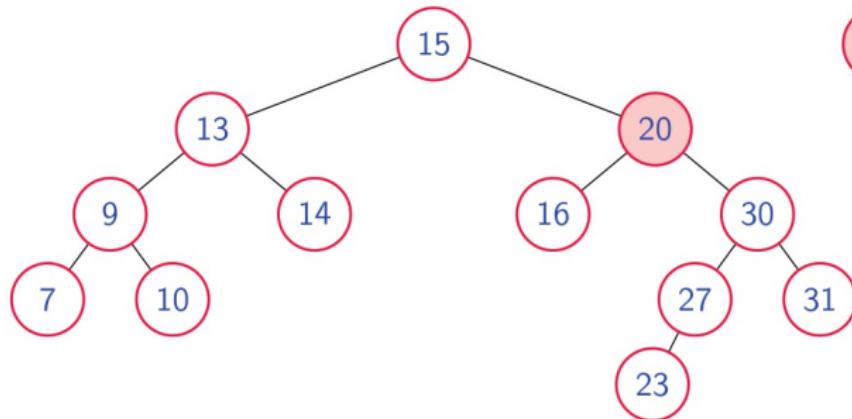


3 случая, когда удаляемая вершина:

- 1) лист - убрать из дерева (убрать у родителя указатель на неё)
- 2) имеет 1 ребенка - переподвесить ребенка на родителя этой вершины
- 3) имеет 2 ребенка - найти последующий (у него точно нет левого ребенка) и записать его значение на место удаляемого, а сам последующий удалить, как в варианте 1 или 2.

Операции с двоичным деревом поиска

- Удалить элемент 20. Обменять последующий с удаляемым.



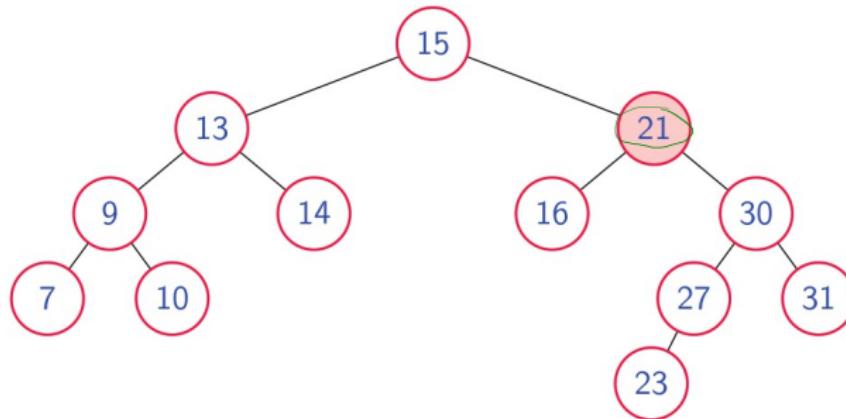
21

3 случая, когда удаляемая вершина:

- лист - убрать из дерева (убрать у родителя указатель на неё)
- имеет 1 ребенка - переподвесить ребенка на родителя этой вершины
- имеет 2 ребенка - найти последующий (у него точно нет левого ребенка) и записать его значение на место удаляемого, а сам последующий удалить, как в варианте 1 или 2.

Операции с двоичным деревом поиска

- ▶ Удалить элемент 20. Обменять последующий с удаляемым.



3 случая, когда удаляемая вершина:

- 1) лист - убрать из дерева (убрать у родителя указатель на неё)
- 2) имеет 1 ребенка - переподвесить ребенка на родителя этой вершины
- 3) имеет 2 ребенка - найти последующий (у него точно нет левого ребенка) и записать его значение на место удаляемого, а сам последующий удалить, как в варианте 1 или 2.