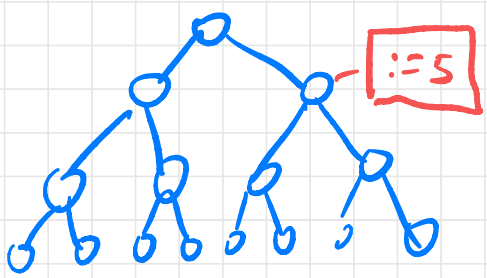
Дерево отрезков

с массовыми операциями

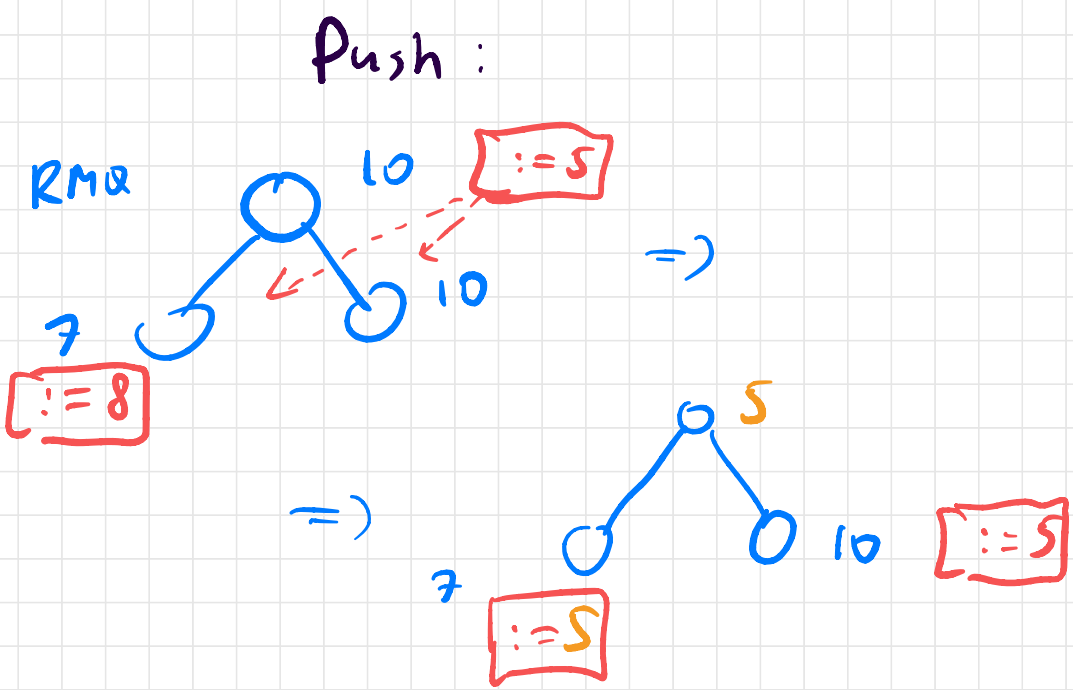
1) $a_l + \ldots + a_r = ?$
2) $min(a_l, \ldots, a_r) = ?$
3) $a_i := x$

Массовые операции:

4) $a_i := x$ for $i = l..r$
5) $a_i += x$ for $i = l..r$



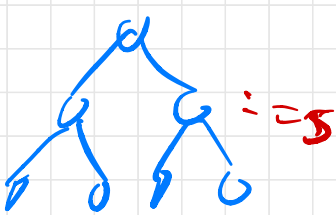
Сделать это во всём поддереве




```python
def push(no: int, nl: int, nr: int):
    if modif[no] == -1:
        return;

    tree[no] = modif[no]
    if (nl != nr - 1)
        modif[2 * no + 1] = modif[2 * no + 2] = modif[no]

    modif[no] = -1;
```

get(l...r)

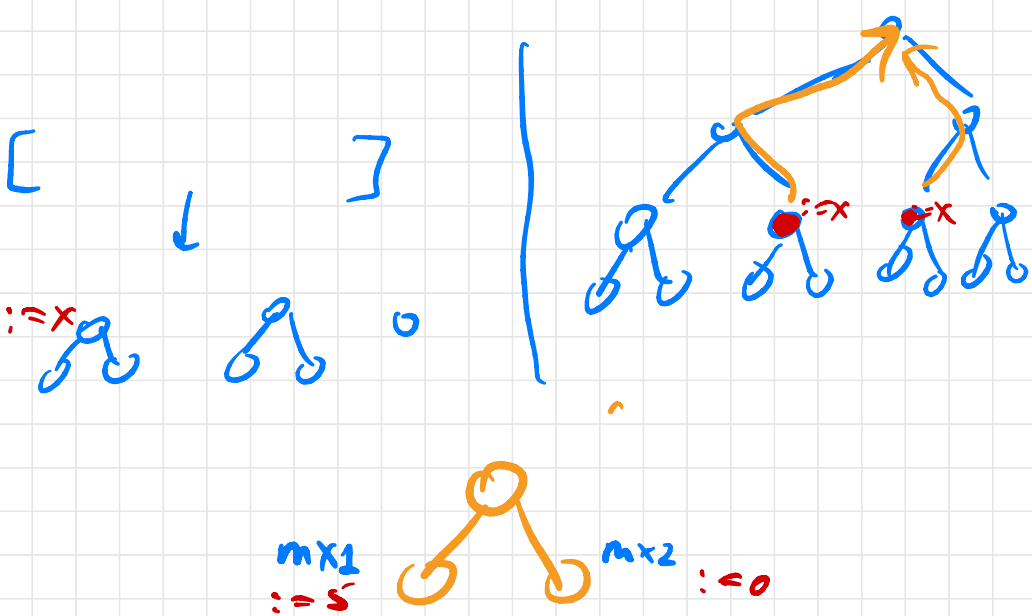
assign(l, r, x)



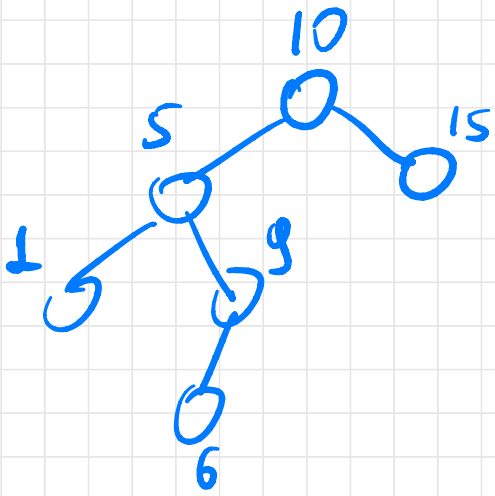
Assign(l,r):

```
def assign(no: int, nl: int, nr: int, l: int, r: int, val: int)
    push(no, nl, nr)

    if l <= nl and nr <= r:
        modif[no] = val
        return

    if nr <= l or r <= nl:
        return

    mid = nl + (nr - nl) // 2
    assign(2 * no + 1, nl, mid, l, r, val)
    assign(2 * no + 2, mid, nr, l, r, val)

    tree[no] = -1
    for ch in [2 * no + 1, 2 * no + 2]:
        child_max = modif[ch] if (modif[ch] != -1) else tree[ch]
        tree[no] = max(tree[no], child_max);
```

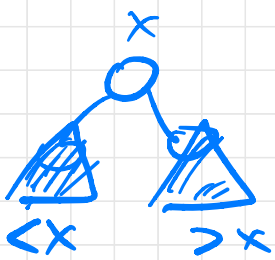
recalc(no)

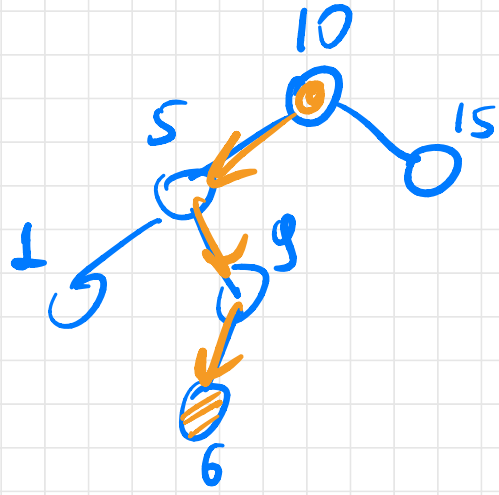


Binary Search Tree

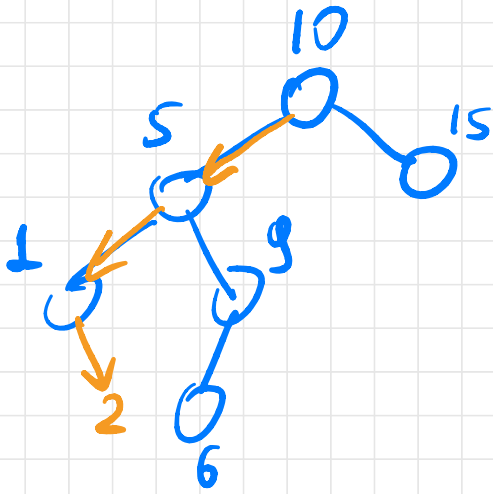


хеш-таблица.



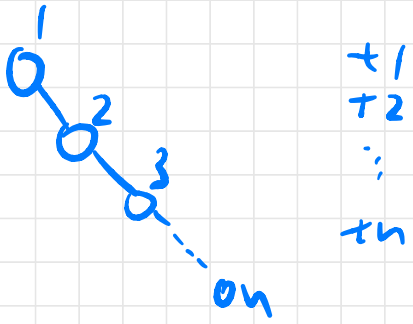


? 6



+2

Пример с плохой глубиной



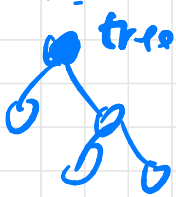
- Балансировка: depth = $O(\log n)$

Утв: Если вставлять ключи в случ. порядке, то E[depth] = $O(\log n)$.

Реализация простого BST

Node {
    left, right
    key,
    (value; если key→value)
}

tree = None | Node { ... }




```python
def find(root: Union[Node, None], key) -> bool:
    if root is None:
        return False

    if root.key == key:
        return True # or return root.value

    if key < root.key:
        return find(root.left, key)
    else:
        return find(root.right, key)
```

Find(tree, key) ← использование в внешнем мире.

новая безусловная в-на

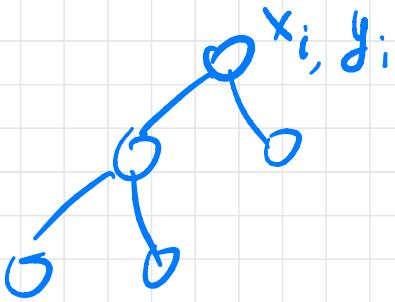
```python
def insert(root: Union[Node, None], newnode: Node) -> Node:
    if root is None:
        return newnode

    if key < root.key:
        root.left = insert(root.left, newnode)
    else:
        root.right = insert(root.right, newnode)
    return root
```

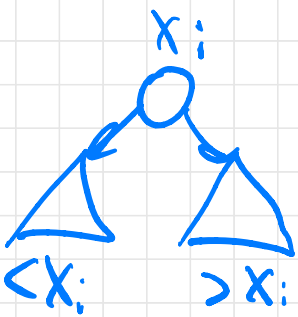
Использование

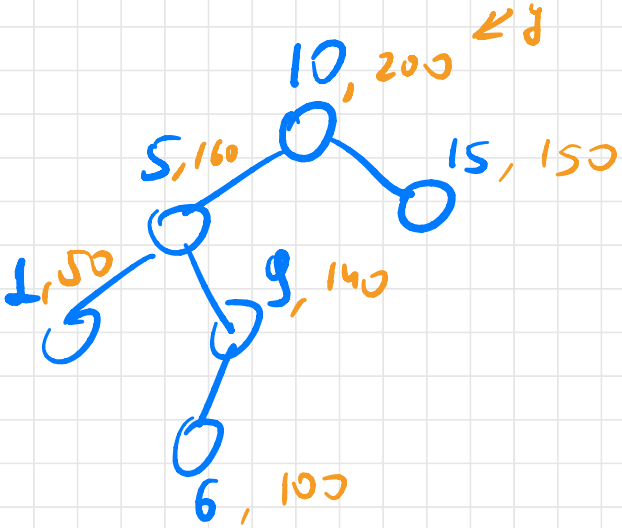
tree = insert(tree, Node(x, ...))

Как сделать дерево балансированным?

Декартово дерево







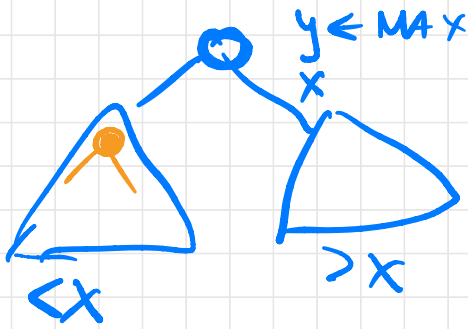
Утв: Пусть даны $\{(x_i, y_i)\}$ и

все x_i - различны и

все y_i - различны.

$\exists$ единственное Д.Д. по $\{(x_i, y_i)\}$.

Д-во:



$\exists$ 300

Утв: Пусть $y \leftarrow$ rand(), тогда

$E[depth(Tree)] = O(\log n)$

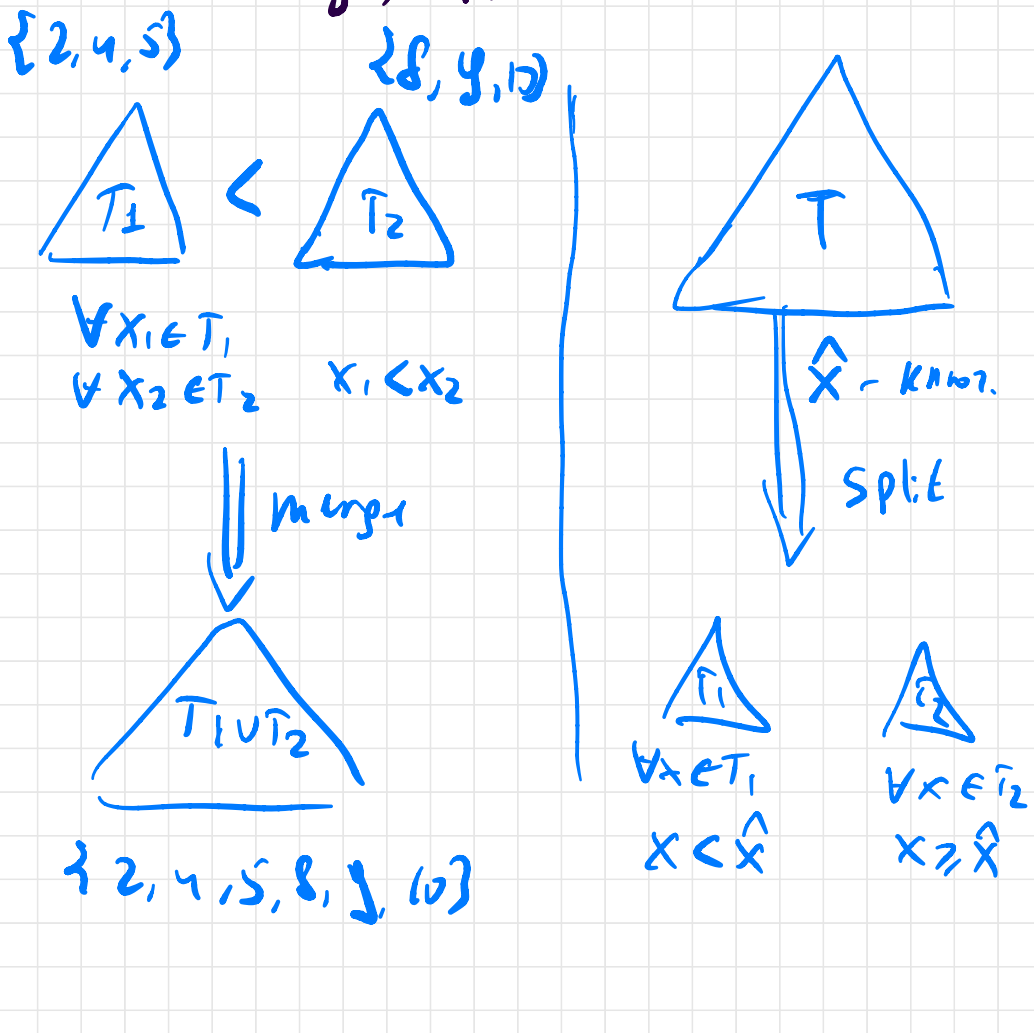
Д-во: опустим.

Rem: Возможно $y_i = y_j$

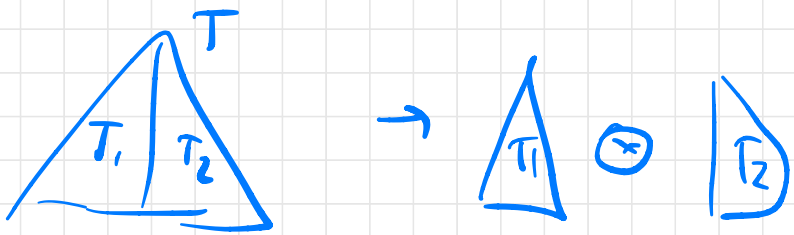
Решение: $O(?, y \leftarrow \text{rand}())$

и будем поддерживать D.D. по этим (x, y).

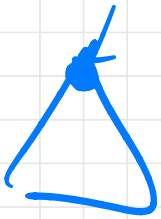
Merge, Split



```
add(T, x): // гарантия x ∉ T
    T1, T2 = split(T, x)
    T = merge(merge(T1, Node(x)), T2)
    return T
```

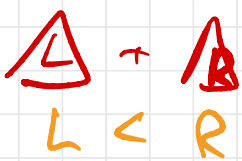


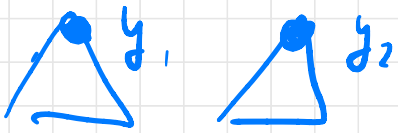

```
delete(T, x):
    T1, T2 = split(T, x)
    (<x)  (≥x)   с T работать больше нельзя!
    T3, T4 = split(T2, x+1)
    x   ≥x+1     с T2 работать больше нельзя
    return merge(T1, T4)
```

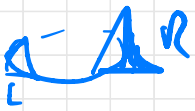


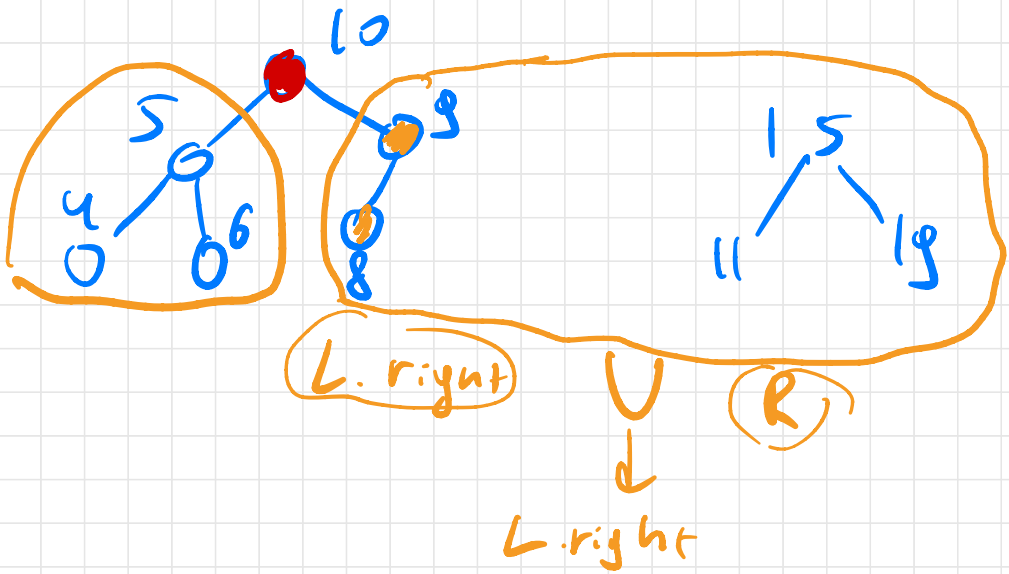
Реализуем merge и split:

```
merge(L, R):
    if not L:
        return R
    if not R:
        return L
    if L.y > R.y:
        // L-новое корень.
        L.right = merge(L.right, R)
        return L
    else:
        R.left = merge(L, R.left)
        return R.
```







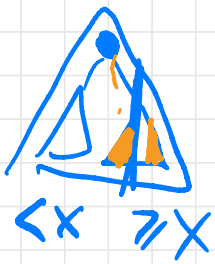


Время работы: $O(depth(L) + depth(R))$

$y \leftarrow next()$.

Время работы: $O(\log n)$

```
Split(T, x):
    if T is None:
        return None, None

    if T.key < x:
        L, R = Split(T.right, x)
        T.right = L
        return T, R
    else: // T.key >= x:
        L, R = Split(T.left, x)
        T.left = R
        return L, T
```



$O(depth)$

$=$

$O(\log n)$

Пример использования

```
tree = None // {}
tree = add(tree, 10) // {10}
tree = add(tree, 12)
tree = remove(tree, 11)
print(find(tree, 10))
```

AVL

Адельсон-Вельского, Ландис

1968, СССР

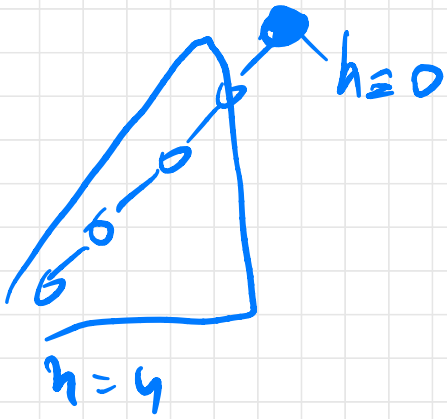
def: height(T) = наибольшая глубина в-ти в дер (глубина в в-ах)

height(○) = 1 height( ) = 0

height( ) = 3

def AVL-инвариант

$\forall v \in V(T): |h(L) - h(R)| \le 1$

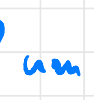


=) AVL-инвариант не выполнен.

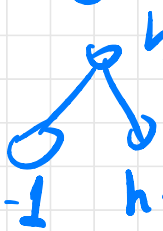
Утв: дерево высоты $h \geq 1$ имеет размер $\geq Fib_{h+1}$

Д-во:

База: $h=1$ ○ $Size \geq 1 = fib_2$

$h=2$  или  $Size \geq 3 = fib_3$

переход:



$Size(h)$ - min размер дерева height=h

$Size\ h \geq Size_{h-1} + Size_{h-2} + 1 \geq Fib_h + Fib_{h-1} = Fib_{h+1}$ □

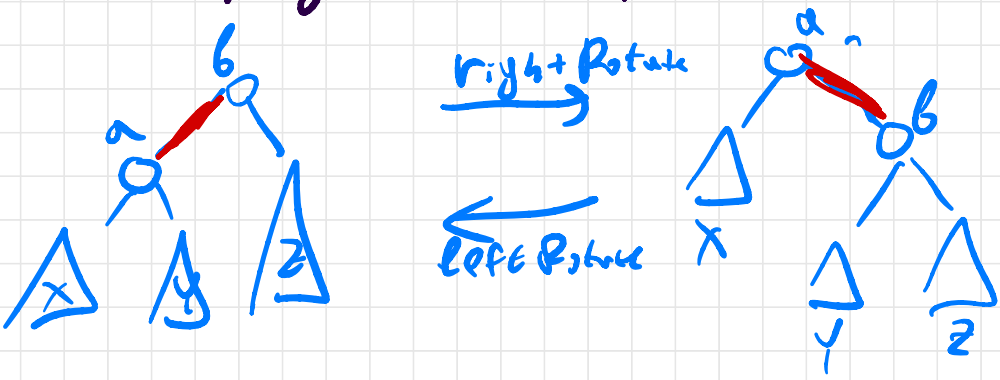
Следствие: height (дерева размера h) =

$= O(\log h)$

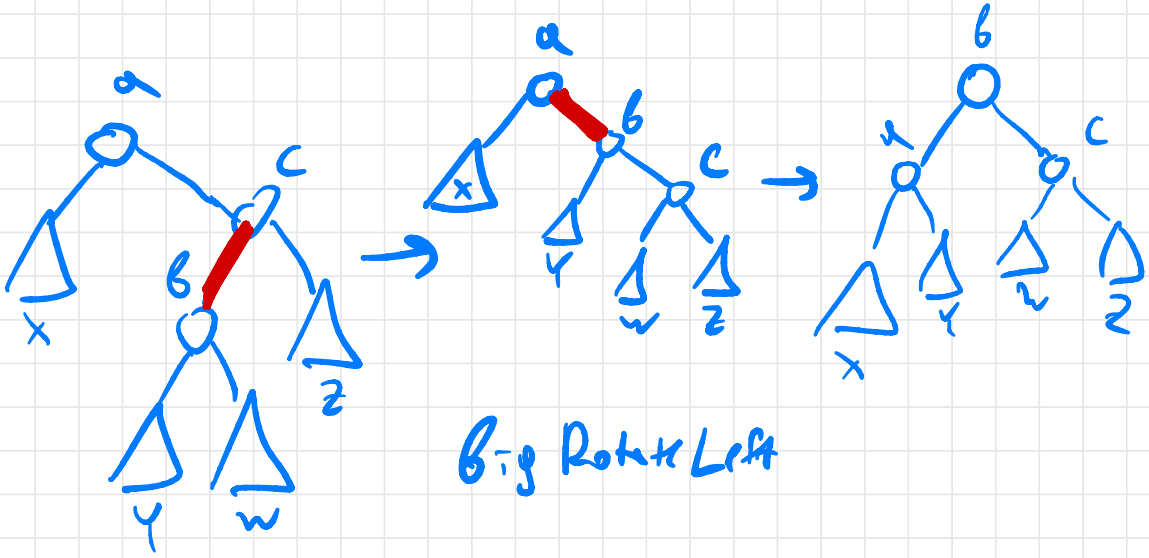
т.к. $Fib_h = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^h\right)$

```
Node {
    key;
    value
    left, right
    (h) ← в каждой в-не подд. храним
          высоту её поддерева
}
```

Вращение Дерева.



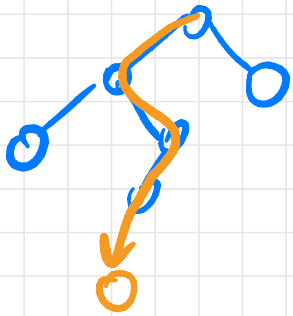
$$X < a < Y < b < Z$$



Big Rotate Left

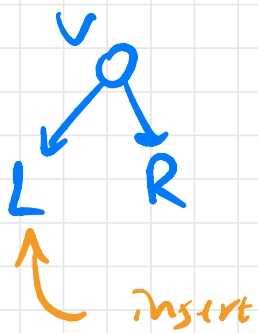
$$X < a < Y < b < W < C < Z$$

Insert



Шаг 1: вставим ноду вставится

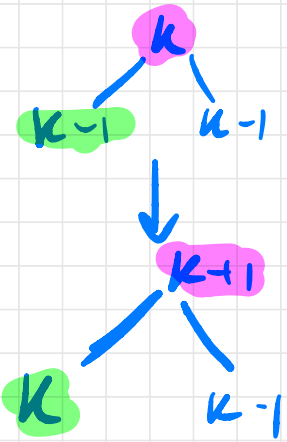
Утв: Алгоритм будет устроен так, что высота каждого поддерева увеличится ≤ 1.



Если h(L) не изменилось, то делать ничего не надо

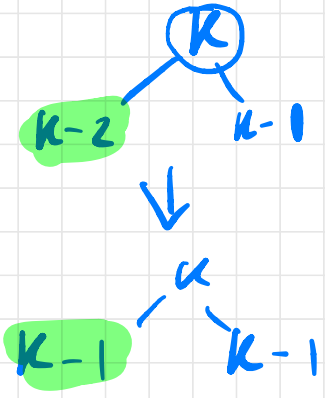
Пусть h(L) увеличилось на 1.

Вариант-1:

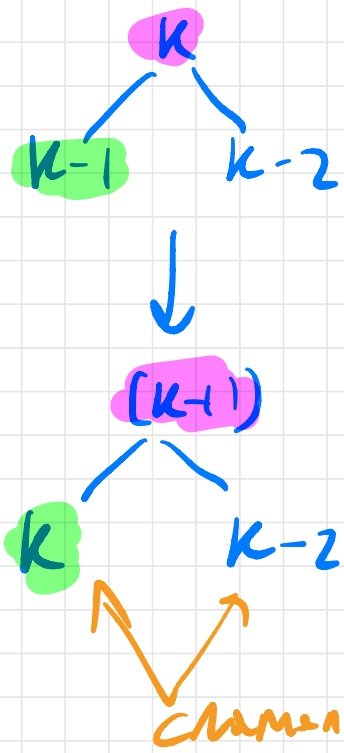


=) просто увеличим h у родителя.

Вариант-0:

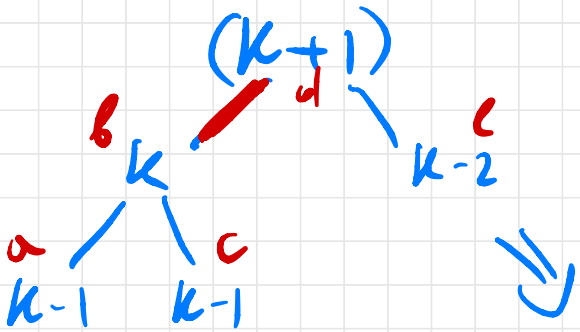


Вариант-2:

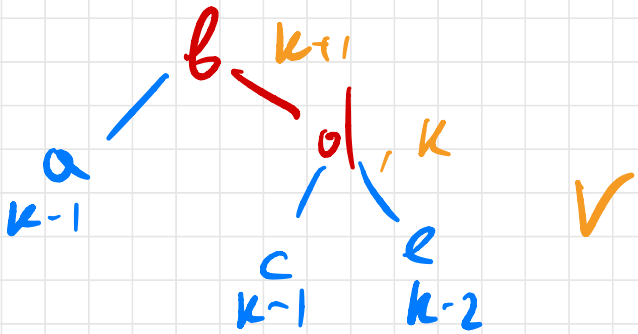


слишком

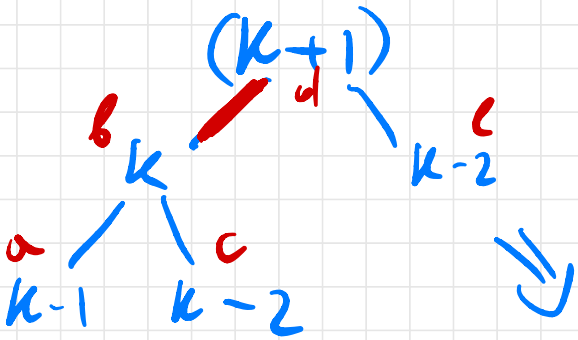
AVL-инвариант!!



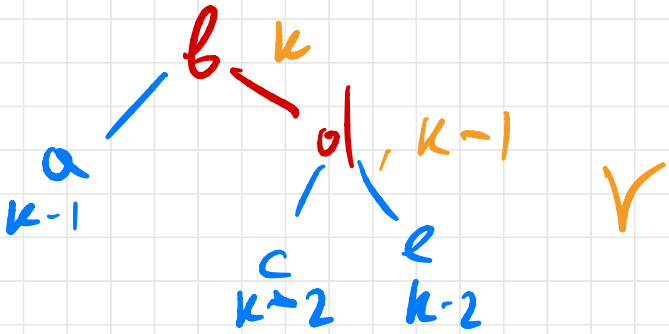
Случай 1



V

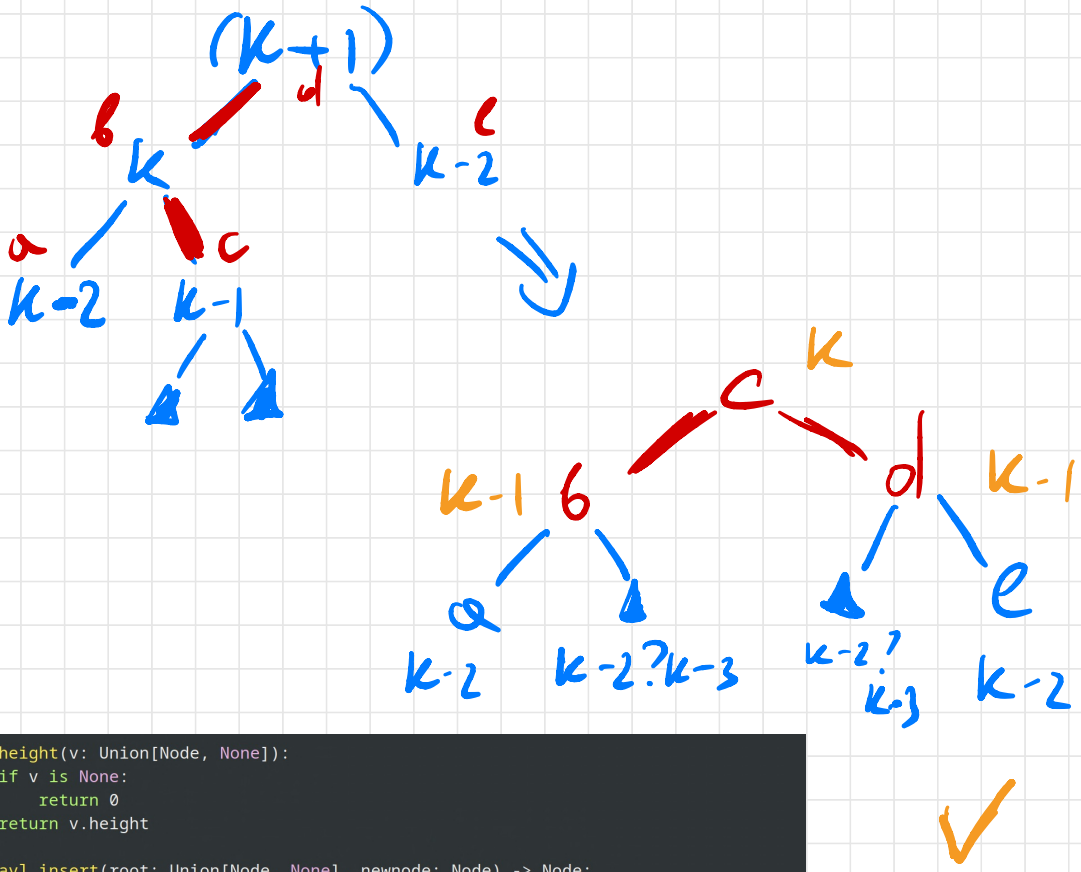


Случай 2.



V

Случай 3




```python
def height(v: Union[Node, None]):
    if v is None:
        return 0
    return v.height

def avl_insert(root: Union[Node, None], newnode: Node) -> Node:
    if root is None:
        return newnode

    if key < root.key:
        root.left = avl_insert(root.left, newnode)
        root.height = 1 + max(height(root.left), height(root.right)) # recalc(root)

        if height(root.left) == height(root.right) + 2
            root = do_needed_rotation(root) # root теперь другая вершина

            # do_needed_rotation определяет какой поворот сделать
            # и обновляет высоту вершин у которых она меняется.

    else:
        # аналогично вставке в левое поддерево
        pass

    return root
```