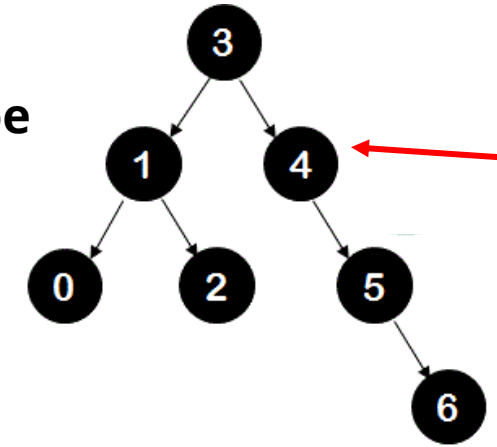
АиСД

АВЛ-деревья

Что такое АВЛ-дерево?

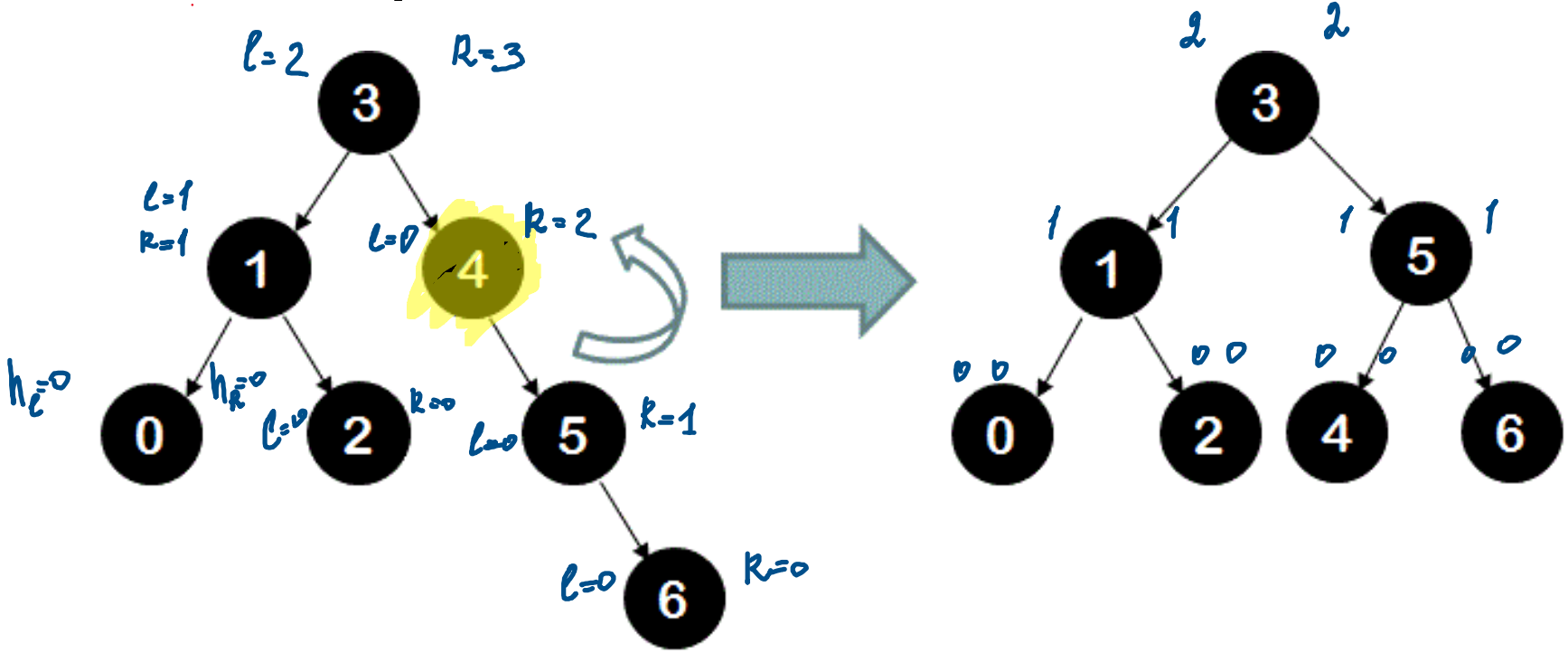
Определение:

АВЛ-дерево — **сбалансированное двоичное дерево поиска**, в котором поддерживается следующее свойство: для каждой его вершины высота её двух поддеревьев различается **не более чем на 1**.

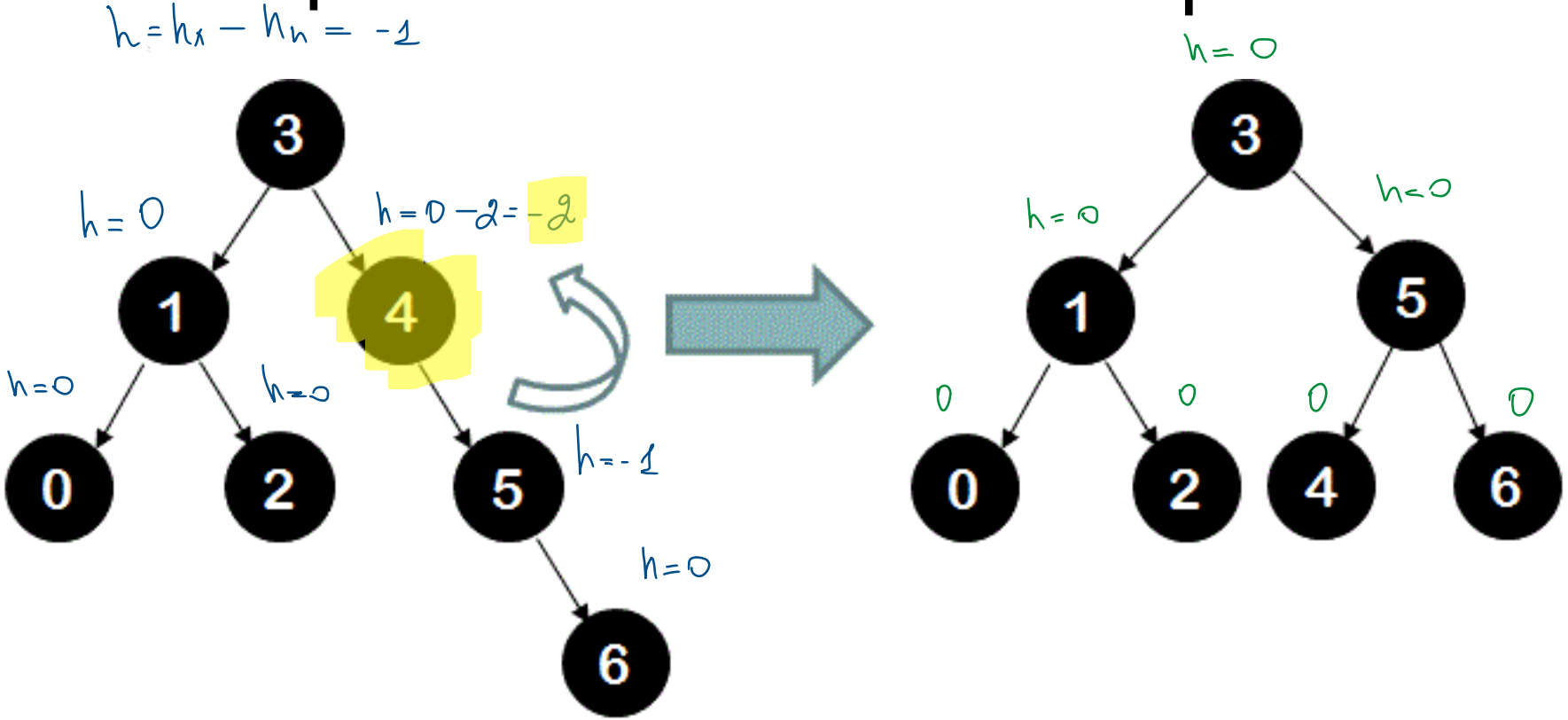


не сбалансированное (у вершины **4** высота левого и правого поддерева различаются на **2**)

АВЛ-дерево


l=2
R=3
3
l=1
R=1
1
l=0
4
R=2
h_L=0
h_R=0
0
l=0
2
R=0
l=0
5
R=1
l=0
6
R=0
2
2
3
1
1
1
5
1
0
0
0
0
0
0
0
0
0
2
4
6

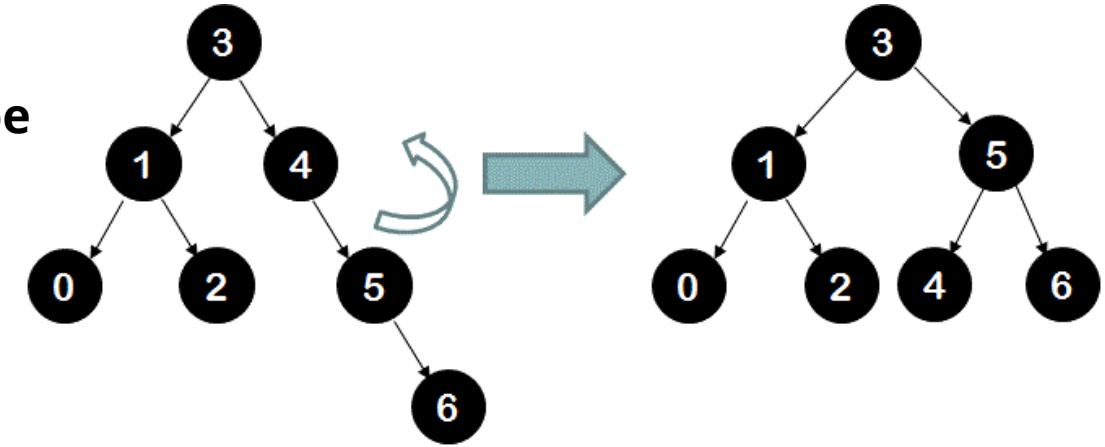
Разница высот левого и правого



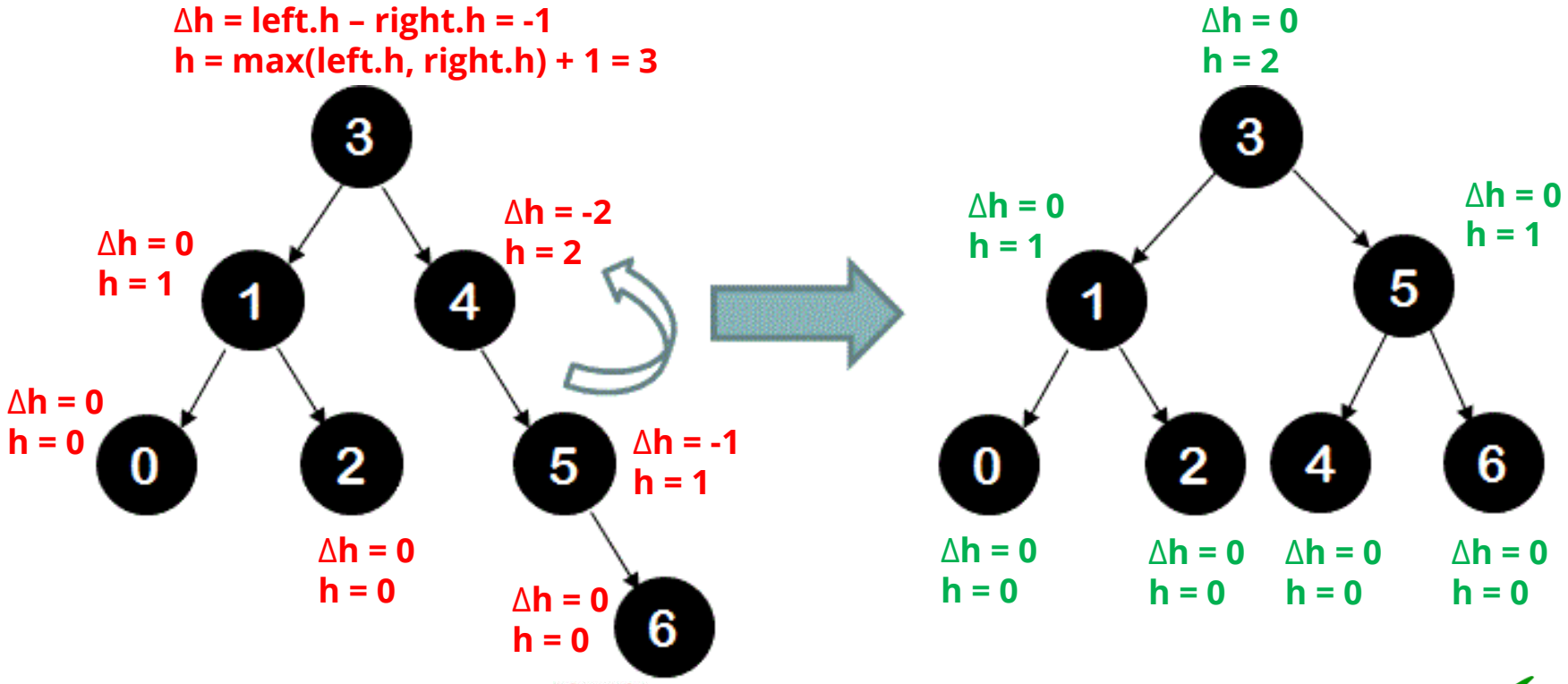
Что такое АВЛ-дерево?

Определение:

АВЛ-дерево — **сбалансированное двоичное дерево поиска**, в котором поддерживается следующее свойство: для каждой его вершины высота её двух поддеревьев различается **не более чем на 1**.



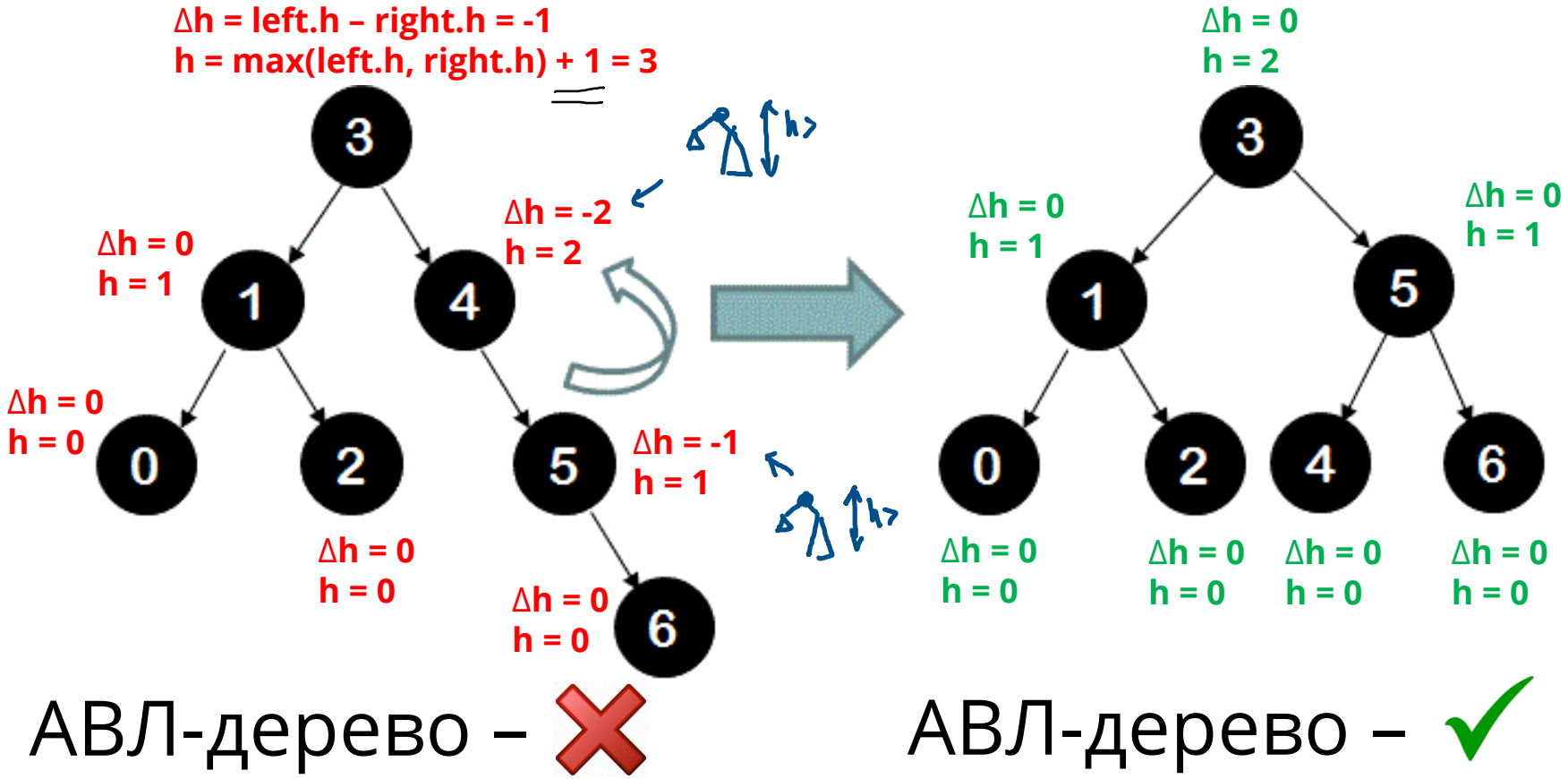
АВЛ-дерево. Разница высот



АВЛ-дерево – ✗

АВЛ-дерево – ✓

АВЛ-дерево. Разница высот



АВЛ-дерево – ✗

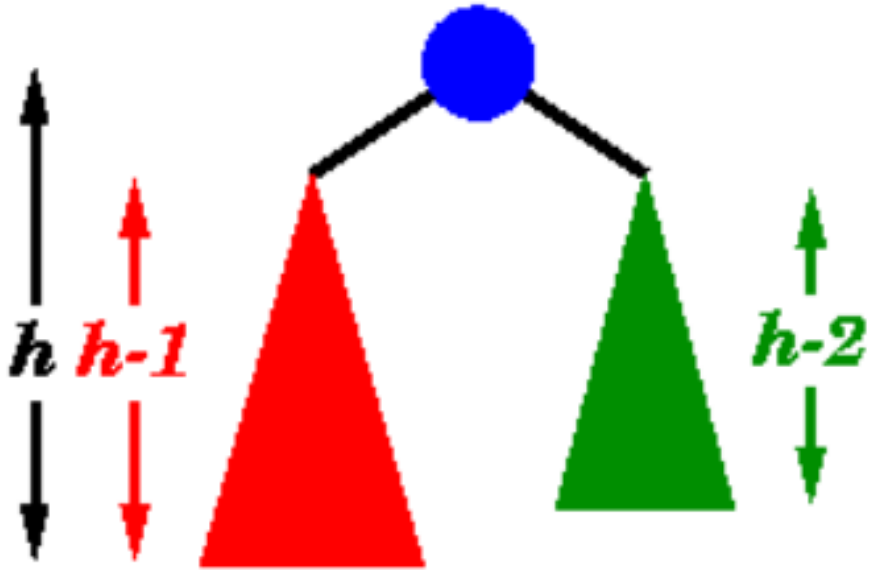
АВЛ-дерево – ✓

Высота АВЛ-дерева

Теорема:

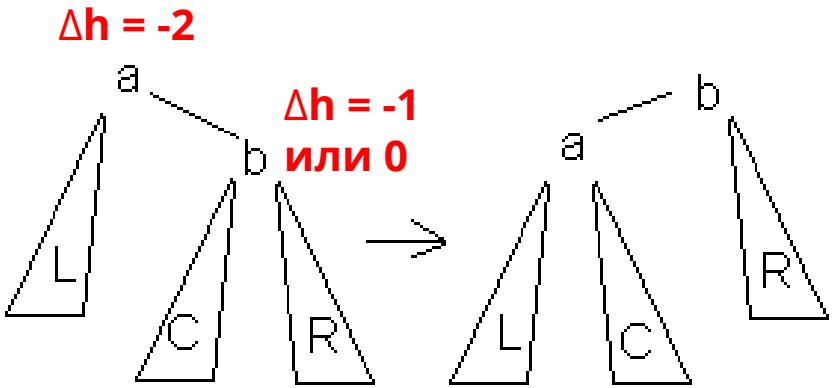
АВЛ-дерево с **n** ключами имеет высоту $\boldsymbol{h = O(log\, n)}$

Высоту поддерева с корнем **x** будем обозначать как **h(x)**,
высоту поддерева **T** – как **h(T)**



АВЛ-дерево. Типы вращений

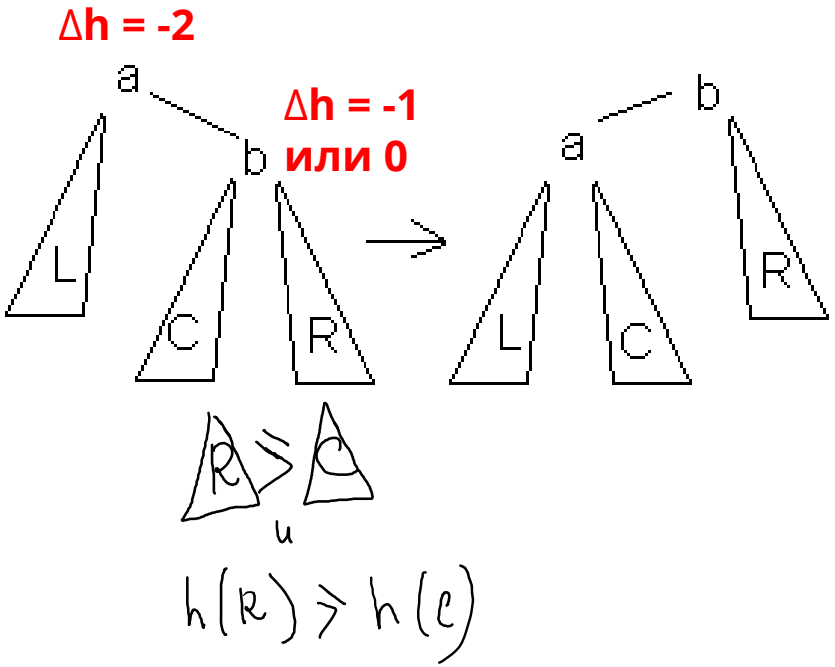
Малое левое вращение:



a, b – вершины
L, C, R - поддеревья

АВЛ-дерево. Типы вращений

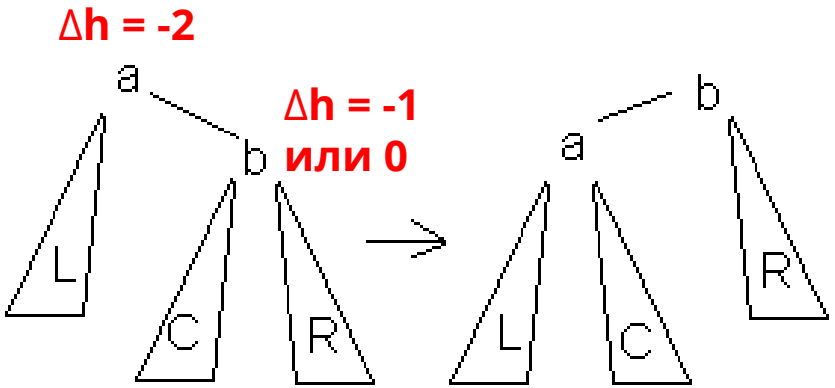
Малое левое вращение:



a, b – вершины
L, C, R - поддеревья

АВЛ-дерево. Типы вращений

Малое левое вращение:



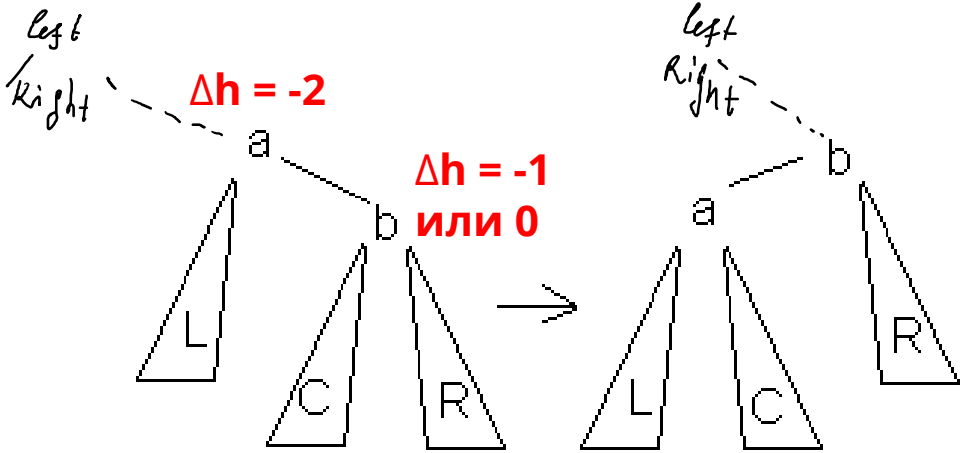
a, b – вершины
L, C, R - поддеревья

$h(R) \geq h(C)$ и $h(L) < h(b)$

$\Delta h = -2 = h(L) - h(b) = -2$

АВЛ-дерево. Типы вращений

Малое левое вращение:



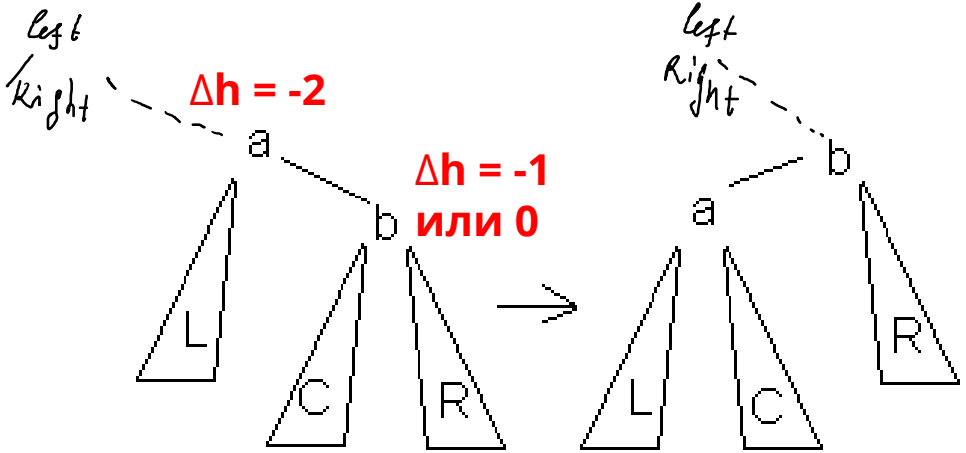
a, b – вершины
L, C, R - поддеревья

Данное вращение используется, когда

(h(L) — h(b)) == -2 && h(C) <= h(R)

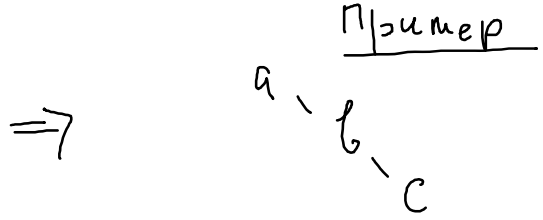
АВЛ-дерево. Типы вращений

Малое левое вращение:



a, b – вершины
L, C, R - поддеревья

Пример

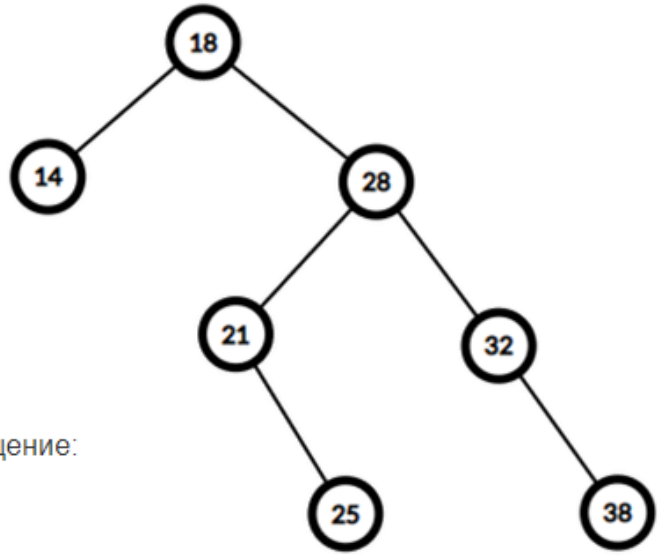


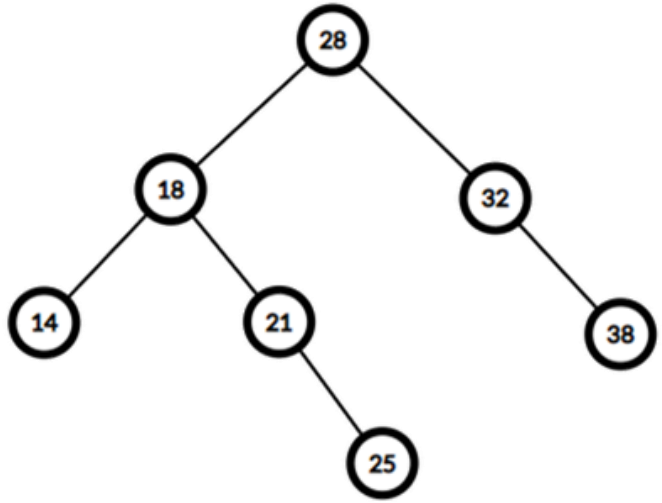
т.к.
$L = C = \phi$

Данное вращение используется, когда

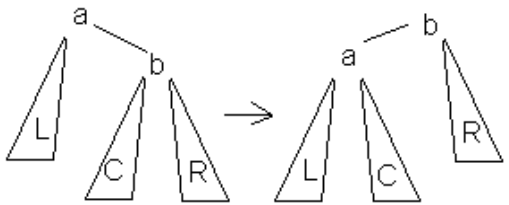
(h(L) — h(b)) == -2 && h(C) <= h(R)

Код малого левого поворота

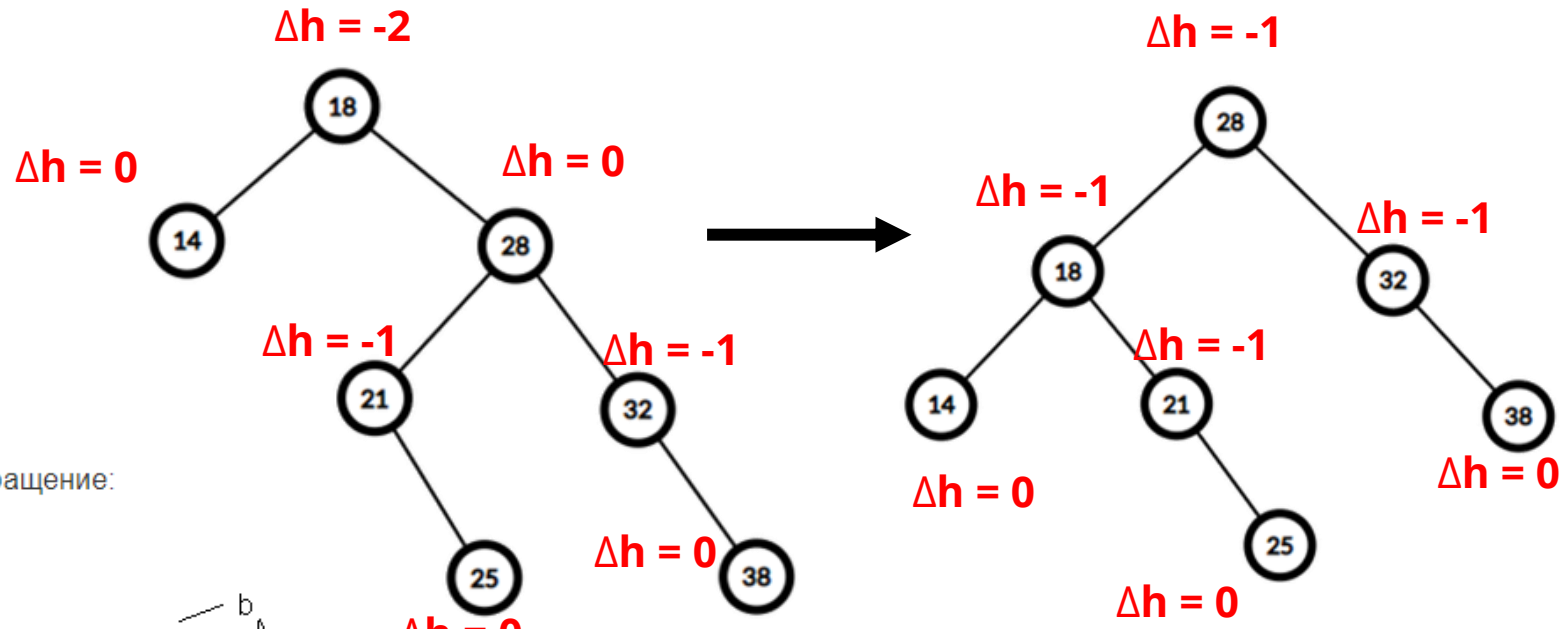




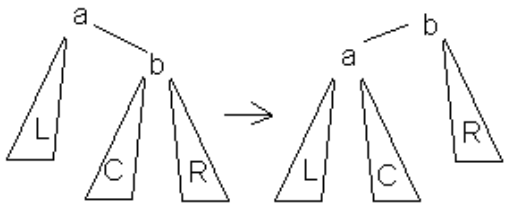
Малое левое вращение:



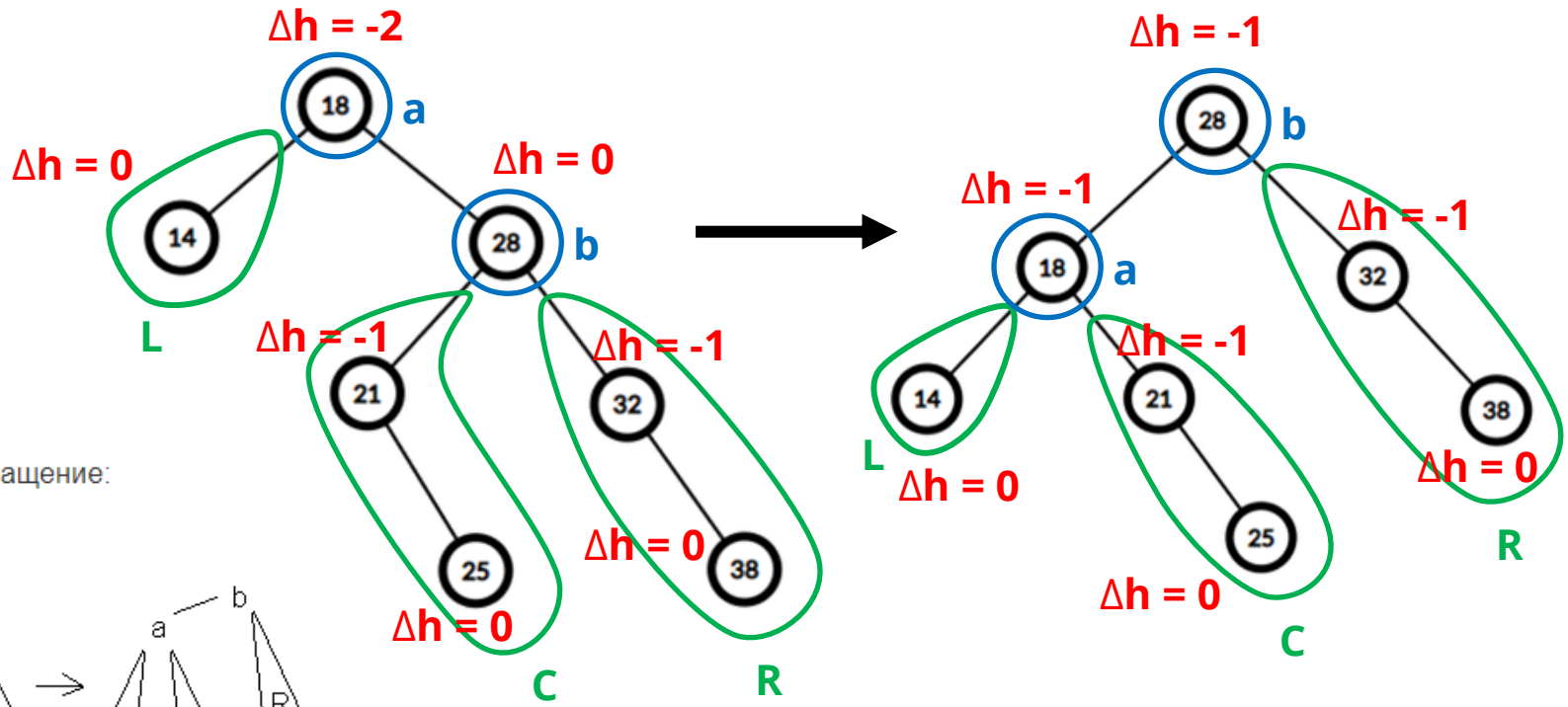
Код малого левого поворота



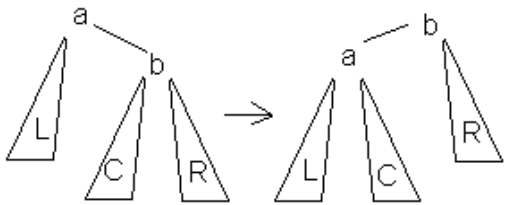
Малое левое вращение:



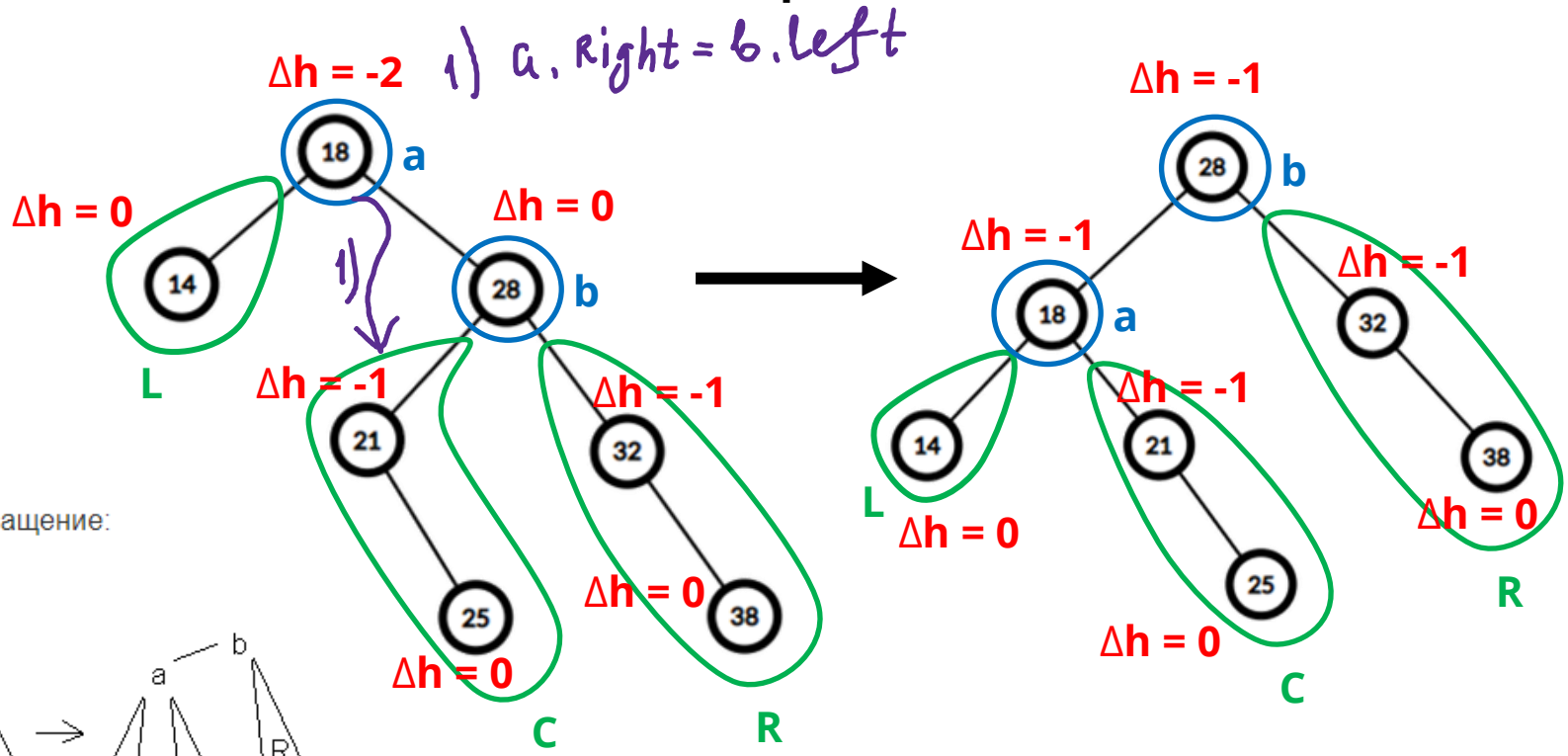
Код малого левого поворота



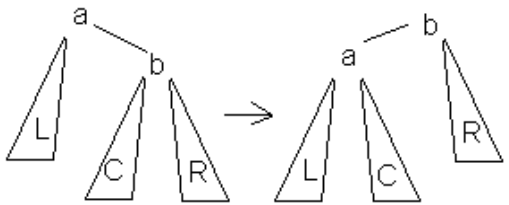
Малое левое вращение:



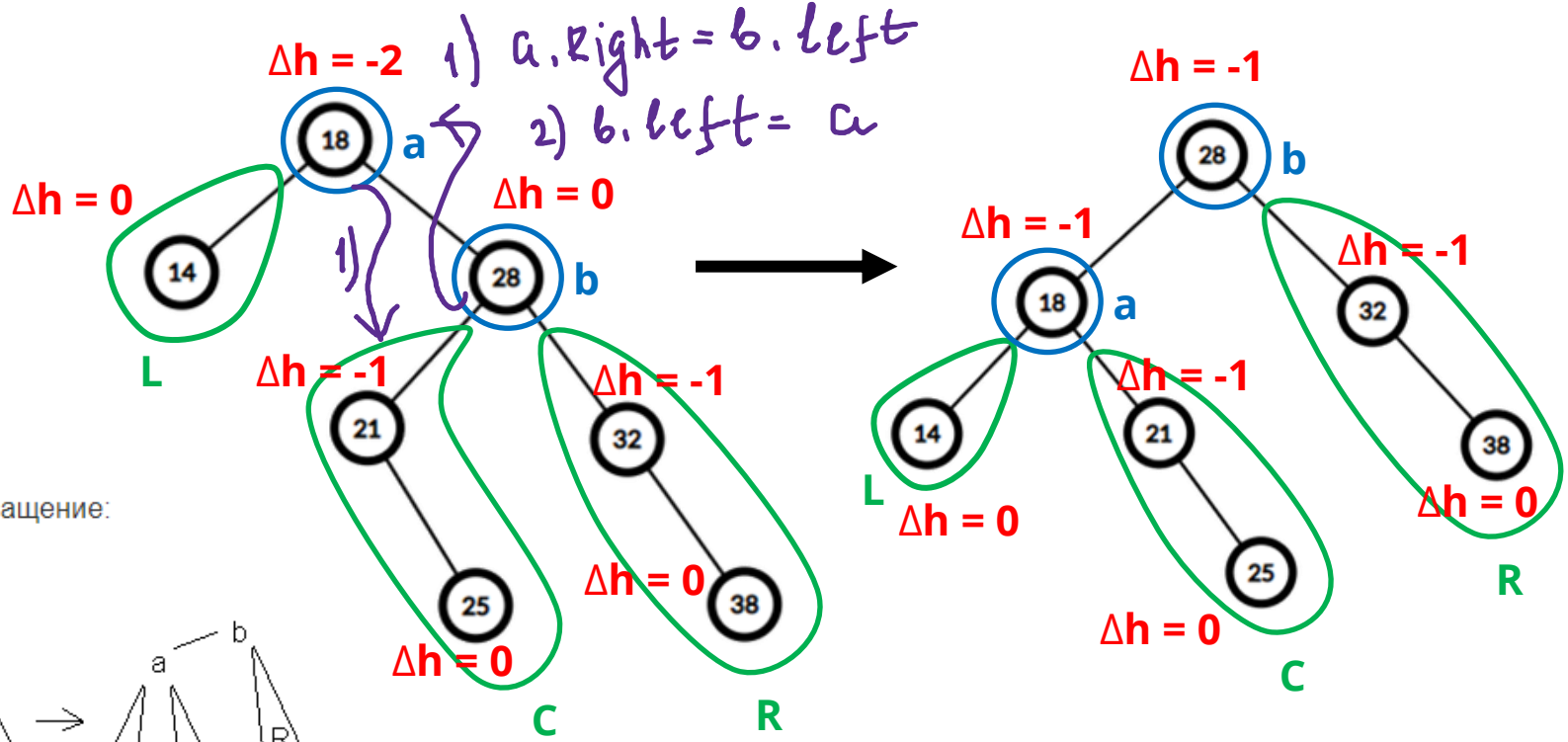
Код малого левого поворота



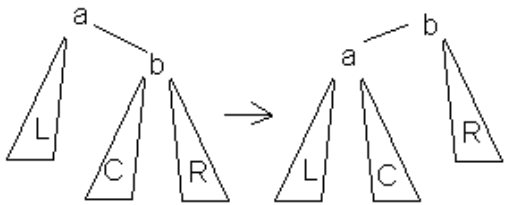
Малое левое вращение:



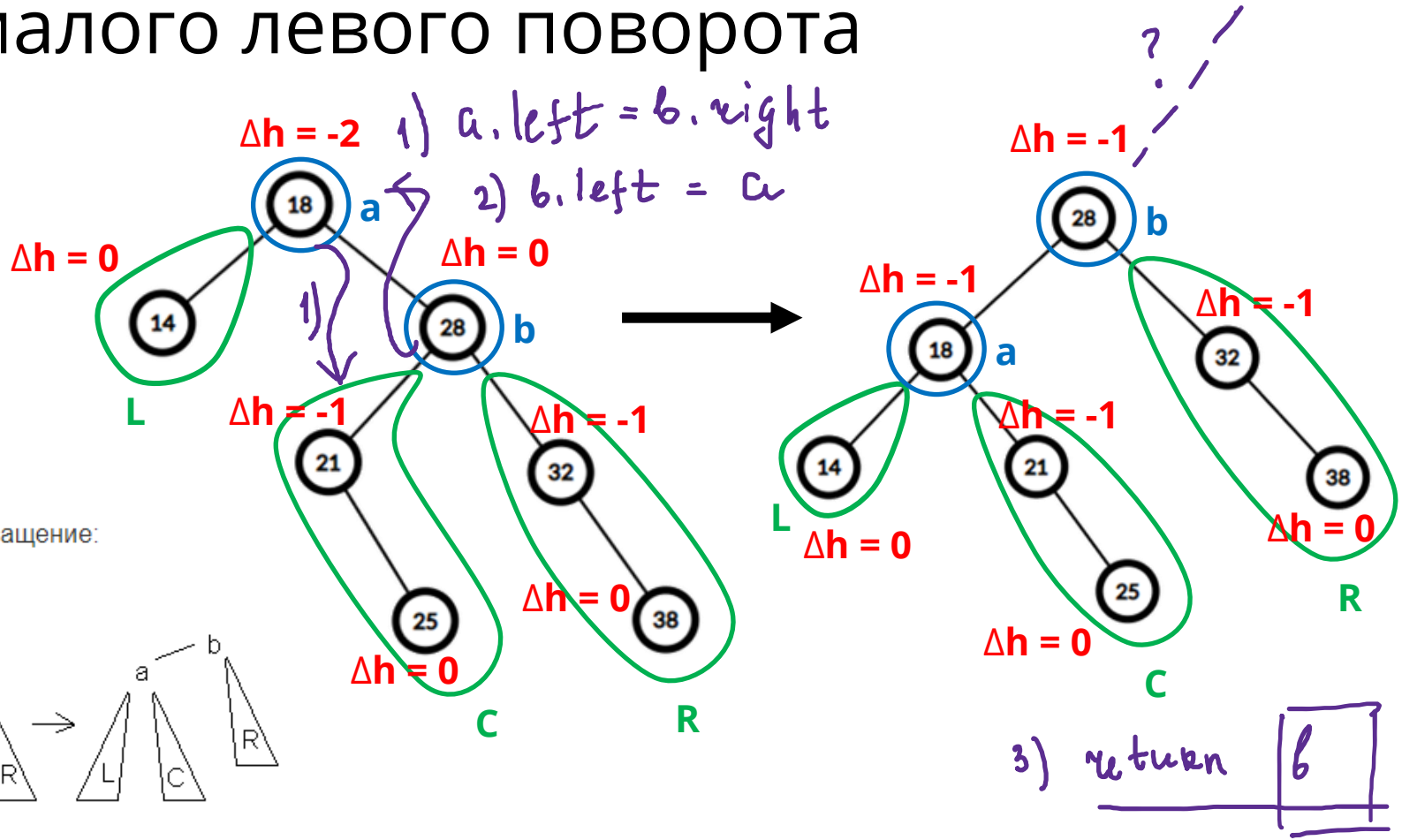
Код малого левого поворота



Малое левое вращение:



Код малого левого поворота

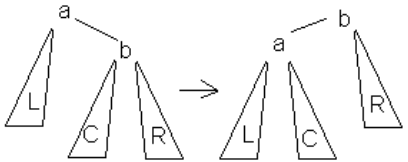


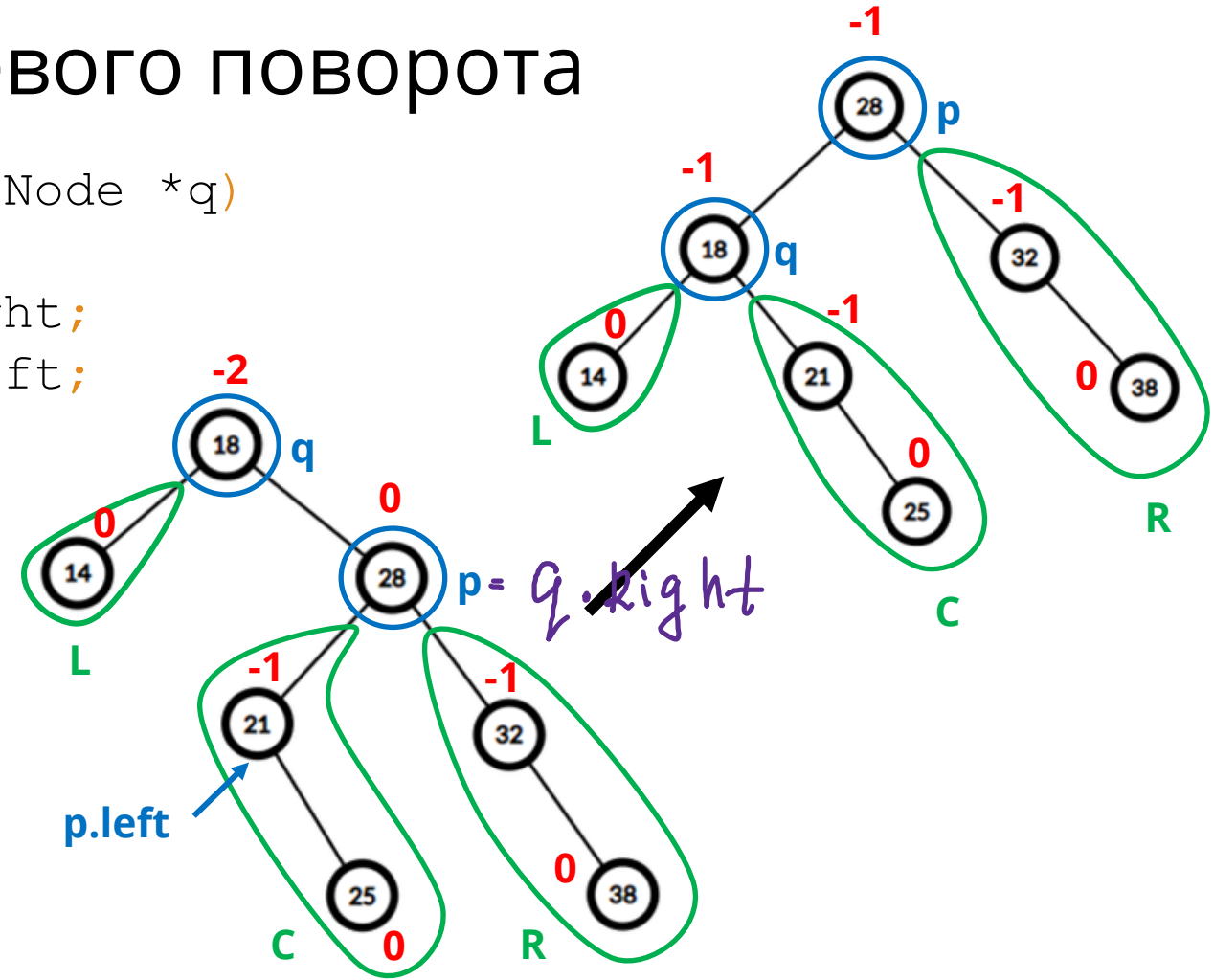
Малое левое вращение:

Код малого левого поворота

```
Node* rotate_left(Node *q)
{
    Node *p=q->right;
1)  q->right=p->left;
2)  p->left=q;
    fix_height(q);
    fix_height(p);
3)  return p;
}
```

Малое левое вращение:



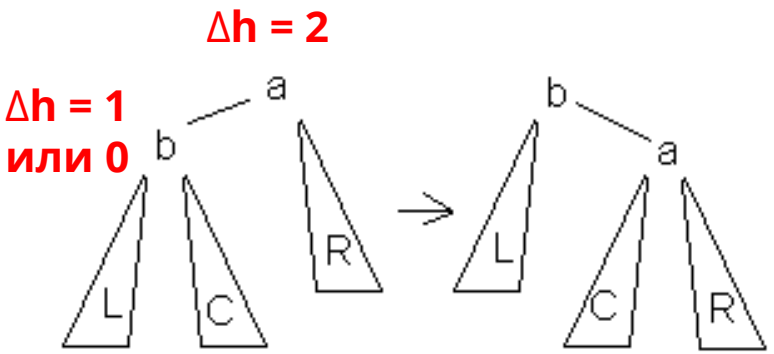


АВЛ-дерево. Типы вращений

Малое правое вращение:

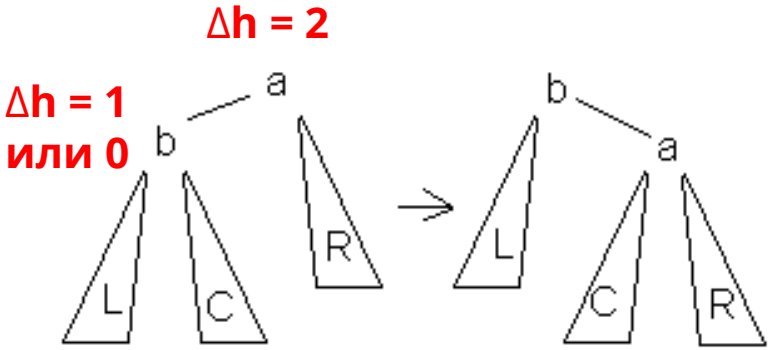
1)

2)



АВЛ-дерево. Типы вращений

Малое правое вращение:



1) $h(b) > h(R)$ на 2

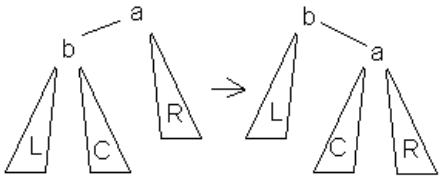
2) $L \geq C$ $h(L) \geq h(C)$

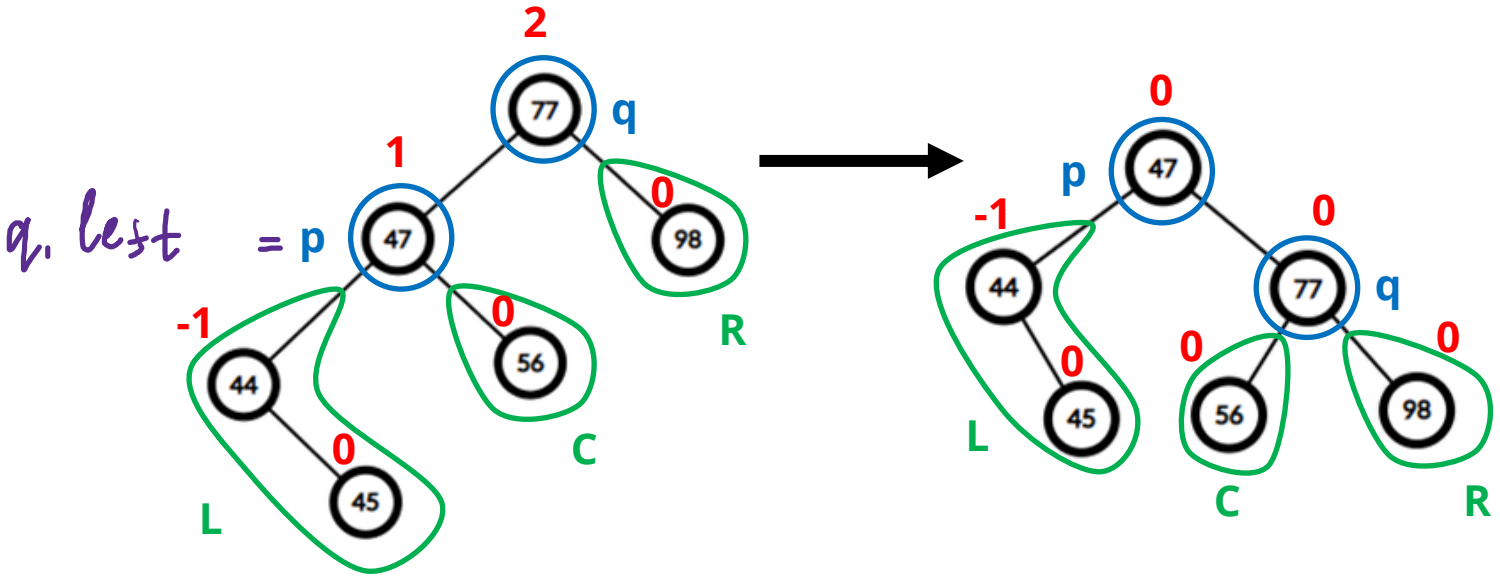
Данное вращение используется, когда

(h(b) — h(R)) == 2 && h(C) <= h(L)

Код малого правого поворота

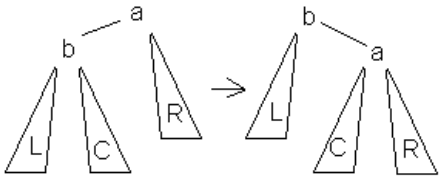
Малое правое вращение:



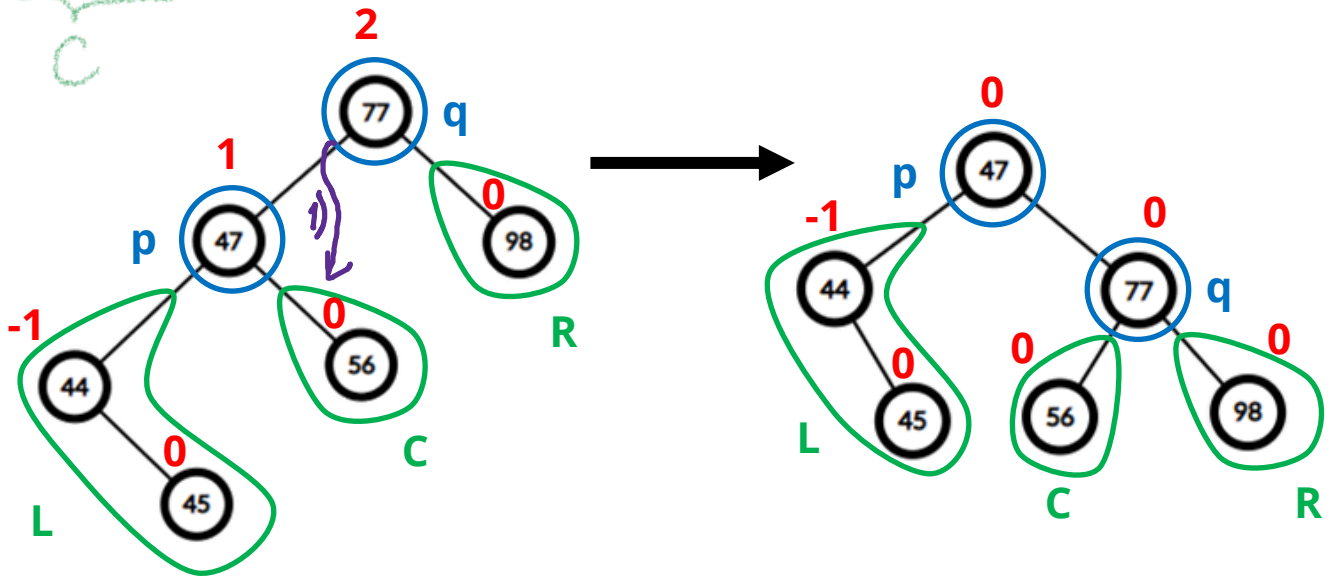


Код малого правого поворота

Малое правое вращение:

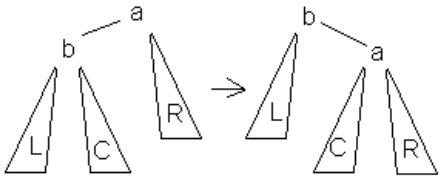


1) q.left = p.right



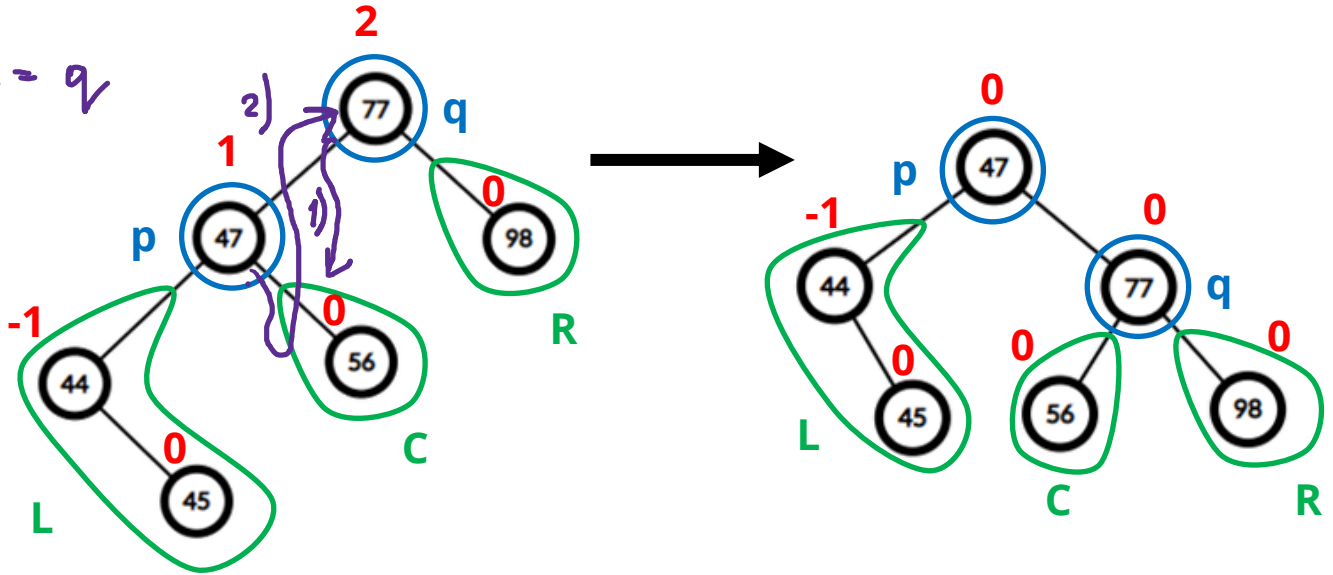
Код малого правого поворота

Малое правое вращение:



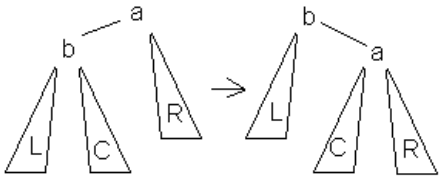
1) q.left = p.right

2) p.right = q



Код малого правого поворота

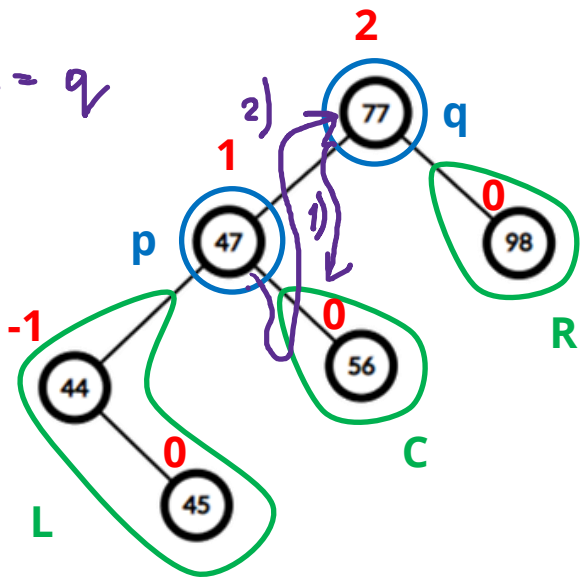
Малое правое вращение:

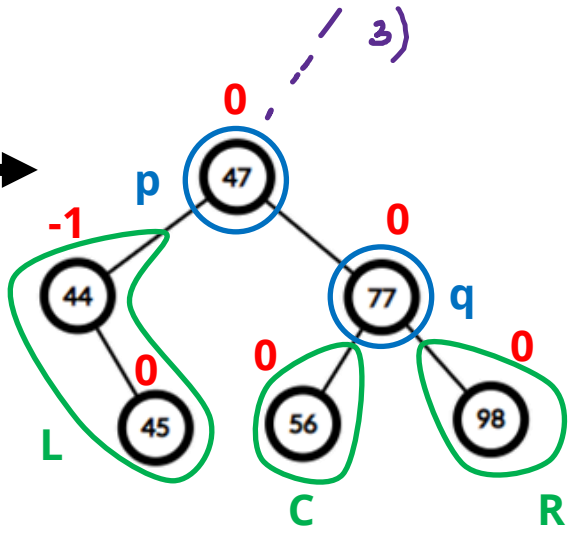


1) q.left = p.right

2) p.right = q

3) return p

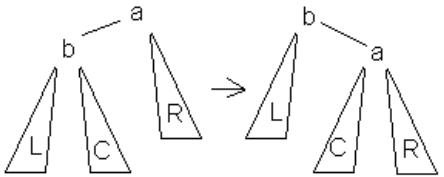


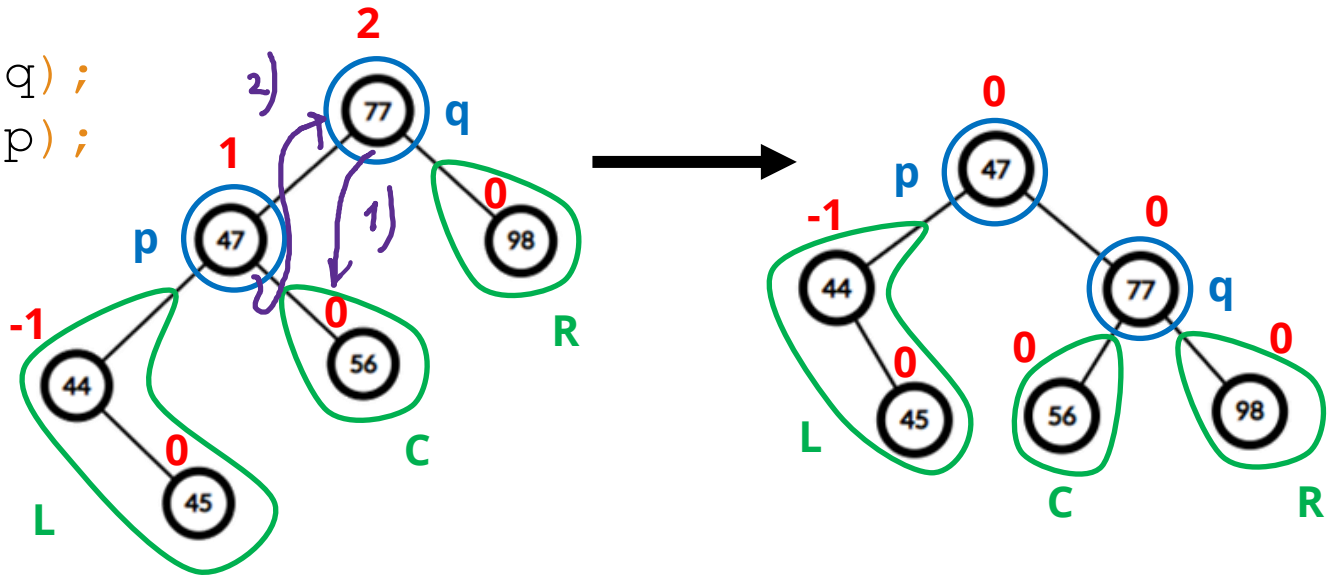


Код малого правого поворота

```
Node* rotate_right(Node *q)
{
    Node *p=q->left;
1)  q->left=p->right;
2)  p->right=q;
    fix_height(q);
    fix_height(p);
3)  return p;
}
```

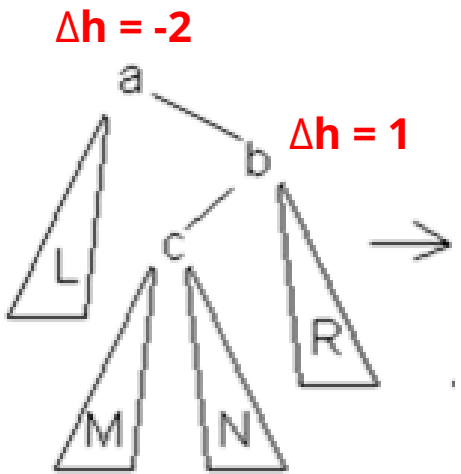
Малое правое вращение:





АВЛ-дерево. Типы вращений

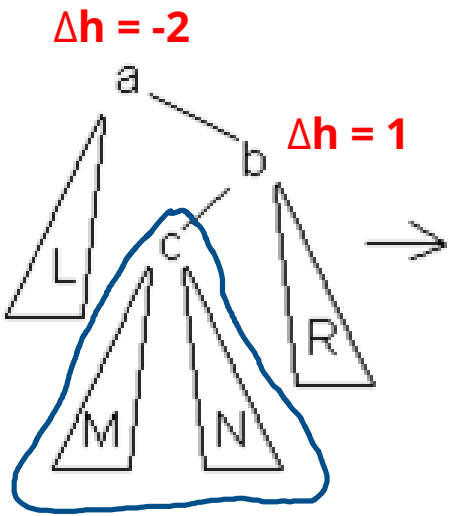
Большое левое вращение:



a, b, c – вершины
L, M, N, R - поддеревья

АВЛ-дерево. Типы вращений

Большое левое вращение:



тут больше чем R

c > R

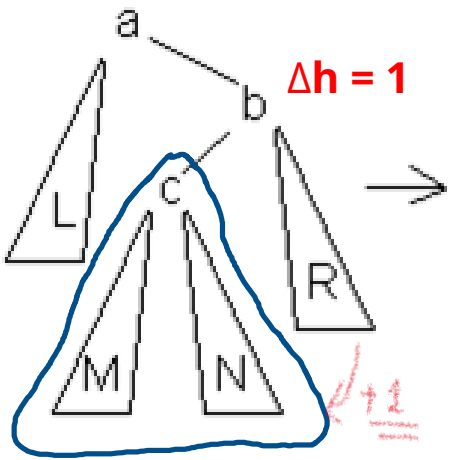
a, b, c – вершины
L, M, N, R - поддеревья

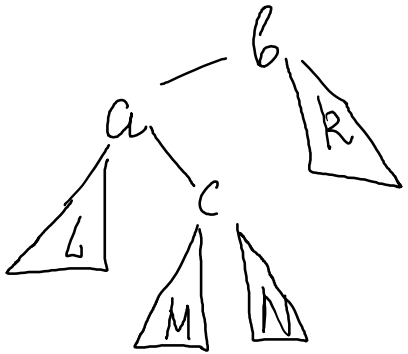
АВЛ-дерево. Типы вращений

Большое левое вращение:

a, b, c – вершины
L, M, N, R - поддеревья

Δh = -2





тут больше чем R

c > R

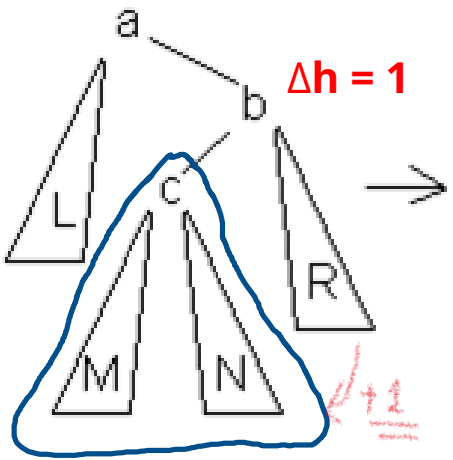
АВЛ-дерево. Типы вращений

Большое левое вращение:

a, b, c – вершины
L, M, N, R - поддеревья

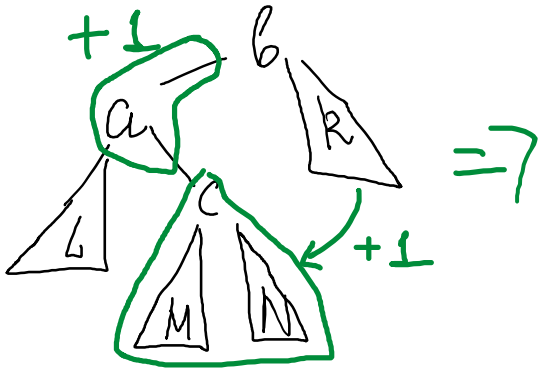
Δh = -2

Δh = 1



тут больше чем R

c > R



a > R

на 2

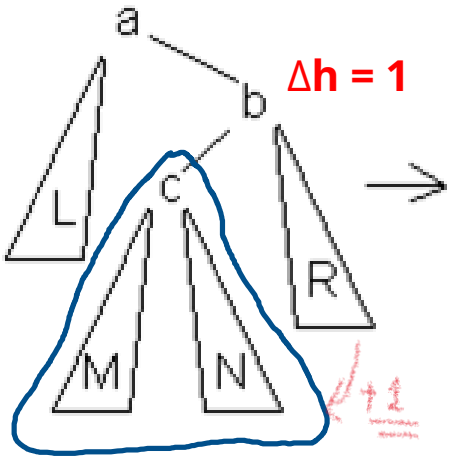
АВЛ-дерево. Типы вращений

Большое левое вращение:

a, b, c – вершины
L, M, N, R - поддеревья

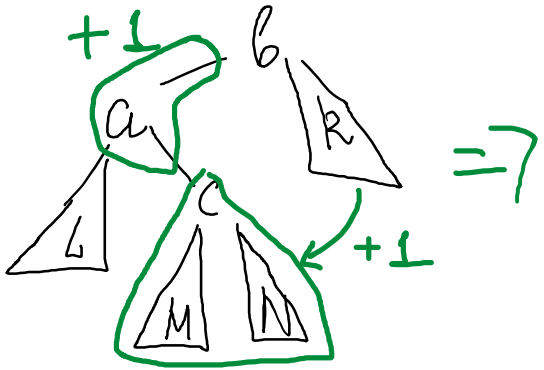
Δh = -2

Δh = 1



тут больше чем R

c > R



a > R

на 2

⇓

нет баланса

т.к

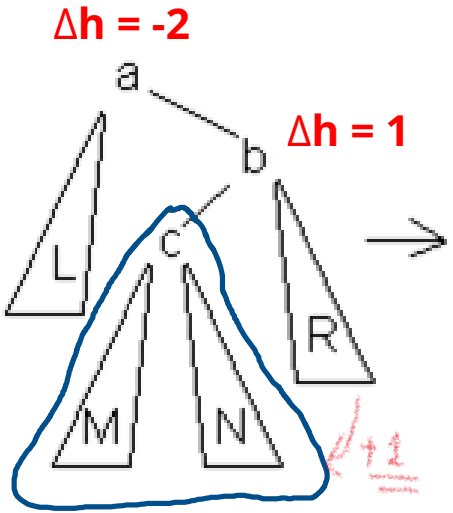
c

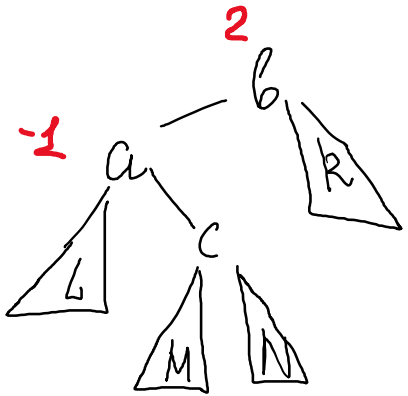
большое

АВЛ-дерево. Типы вращений

Большое левое вращение:

a, b, c – вершины
L, M, N, R - поддеревья





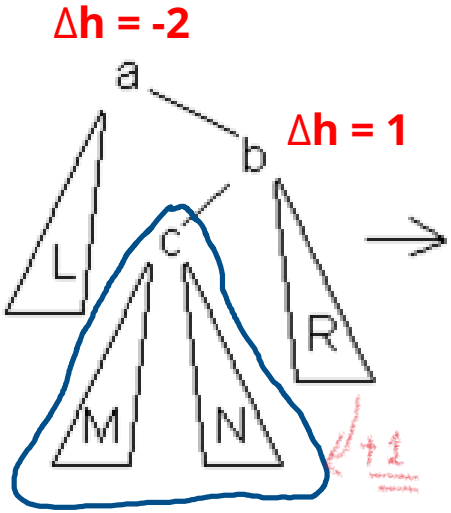
тут больше чем R

c > R

АВЛ-дерево. Типы вращений

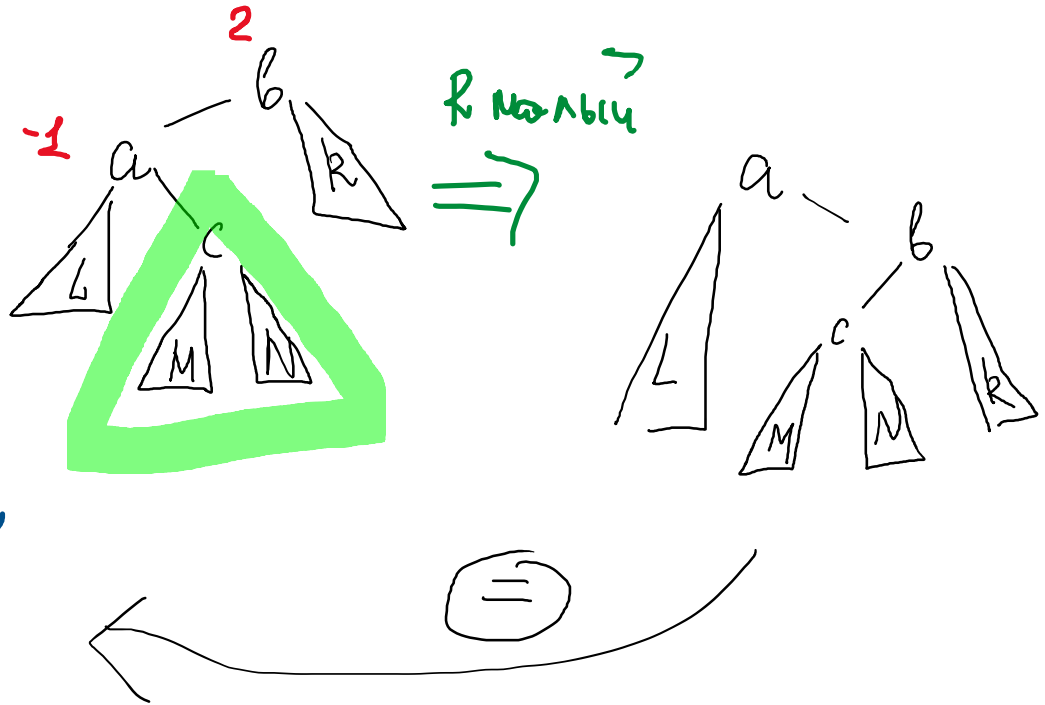
Большое левое вращение:

a, b, c – вершины
L, M, N, R - поддеревья



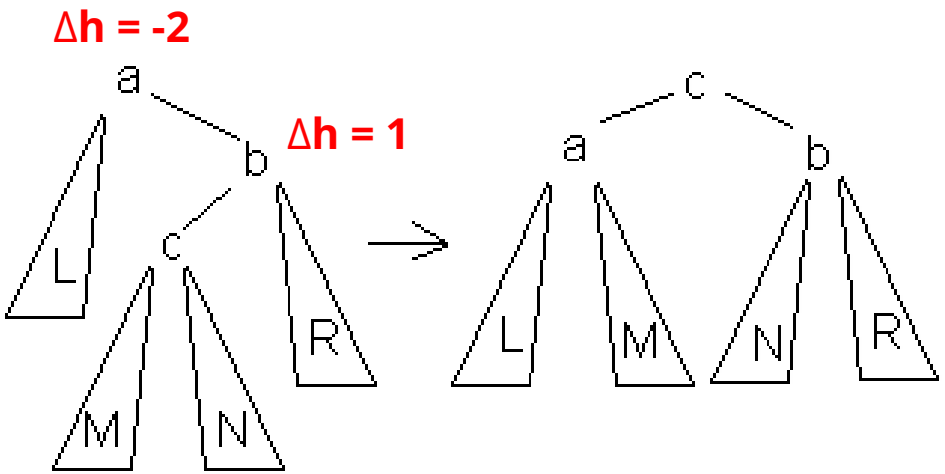
тут больше чем R

c > R



АВЛ-дерево. Типы вращений

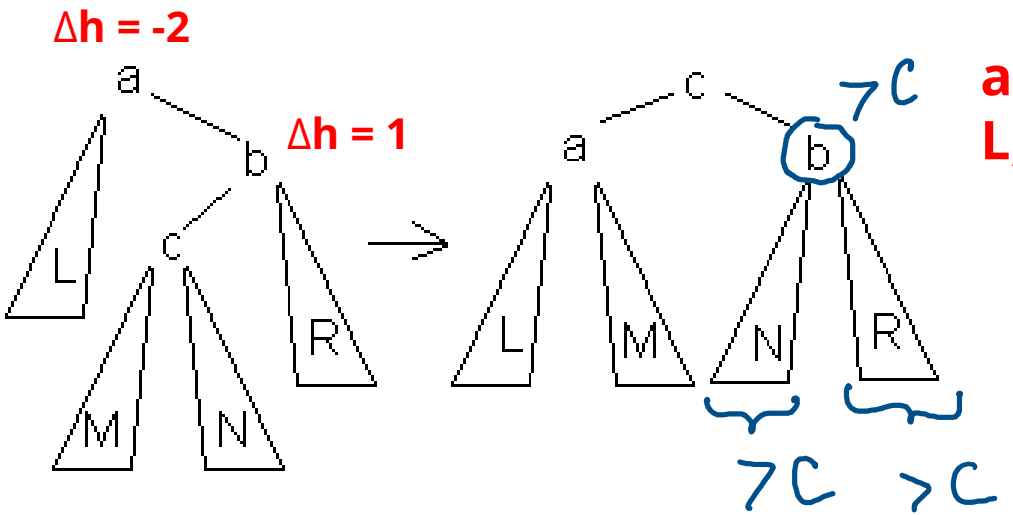
Большое левое вращение:



a, b, c – вершины
L, M, N, R - поддеревья

АВЛ-дерево. Типы вращений

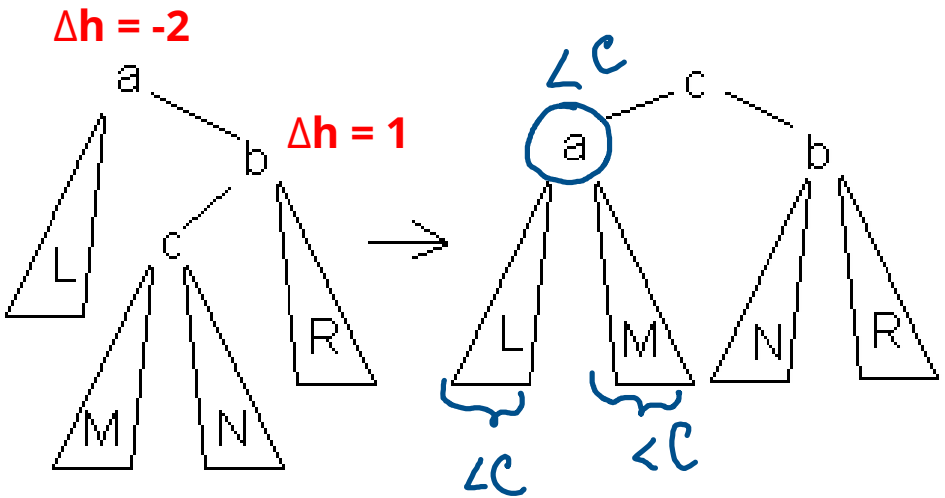
Большое левое вращение:



a, b, c – вершины
L, M, N, R - поддеревья

АВЛ-дерево. Типы вращений

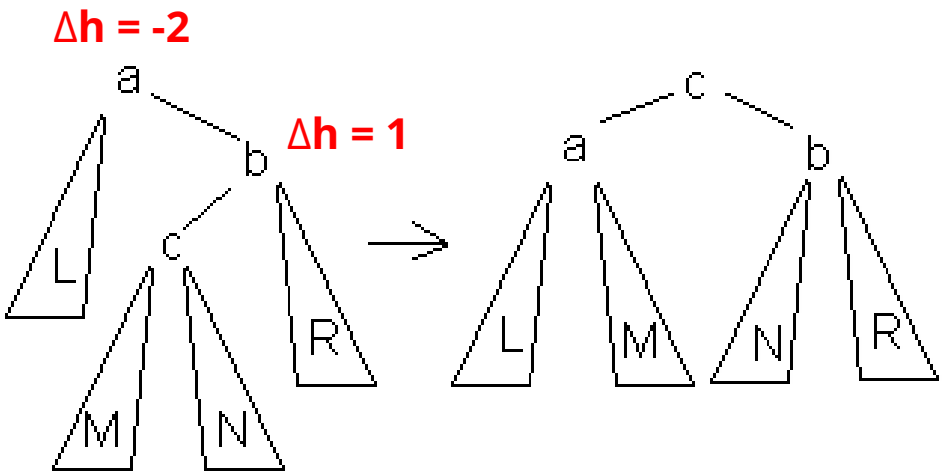
Большое левое вращение:



a, b, c – вершины
L, M, N, R - поддеревья

АВЛ-дерево. Типы вращений

Большое левое вращение:



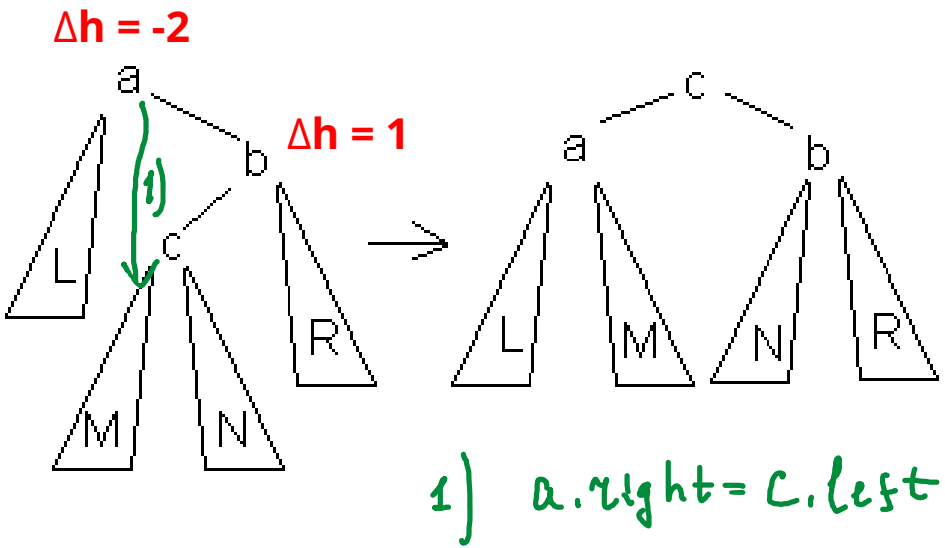
a, b, c – вершины
L, M, N, R - поддеревья

Данное вращение используется, когда

(h(L) — h(b)) == -2 && h(c) > h(R)

АВЛ-дерево. Типы вращений

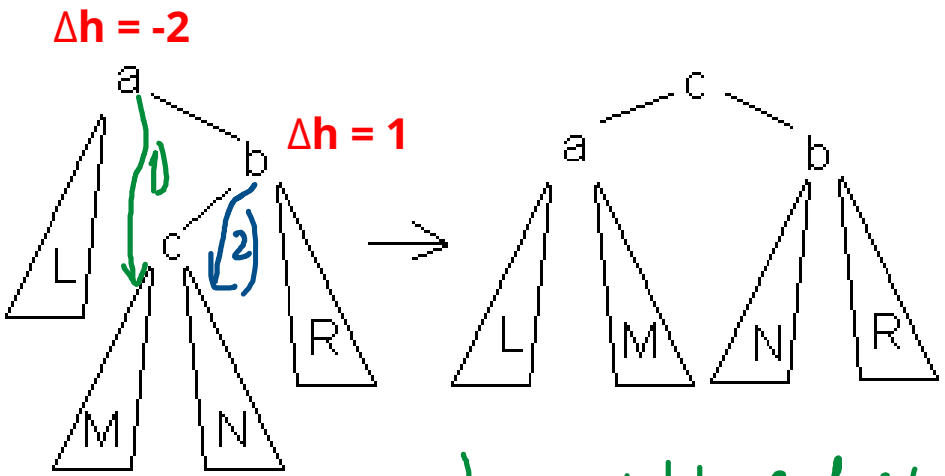
Большое левое вращение:



a, b, c – вершины
L, M, N, R - поддеревья

АВЛ-дерево. Типы вращений

Большое левое вращение:

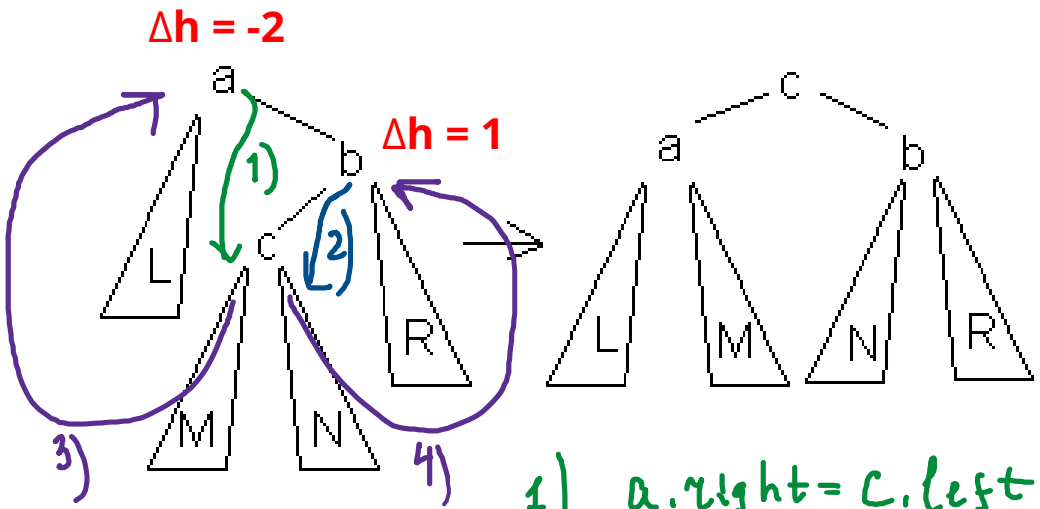


a, b, c – вершины
L, M, N, R - поддеревья

1) a.right = c.left
2) b.left = c.right

АВЛ-дерево. Типы вращений

Большое левое вращение:



a, b, c – вершины
L, M, N, R - поддеревья

1) a.right = c.left

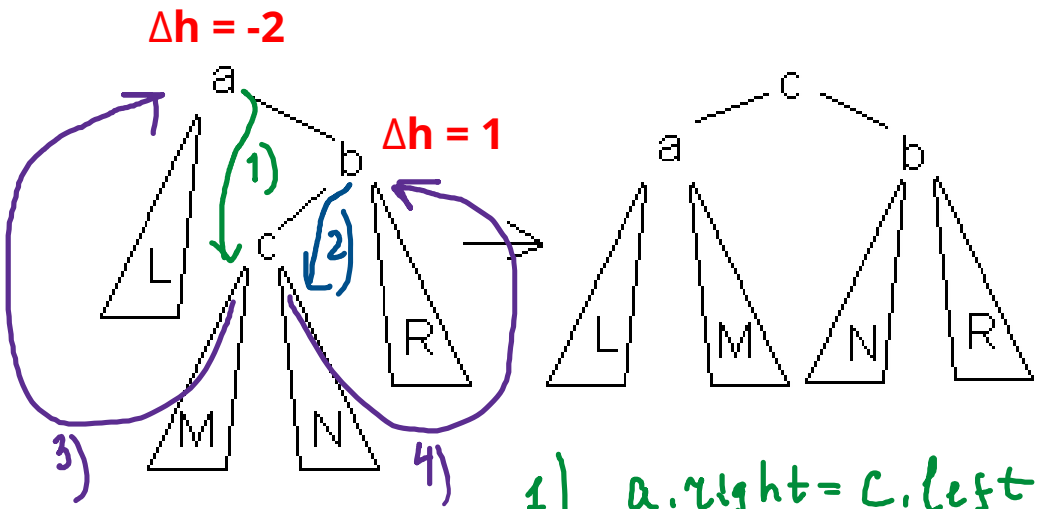
2) b.left = c.right

3) c.left = a

4) c.right = b

АВЛ-дерево. Типы вращений

Большое левое вращение:



a, b, c – вершины
L, M, N, R - поддеревья

1) a.right = c.left

2) b.left = c.right

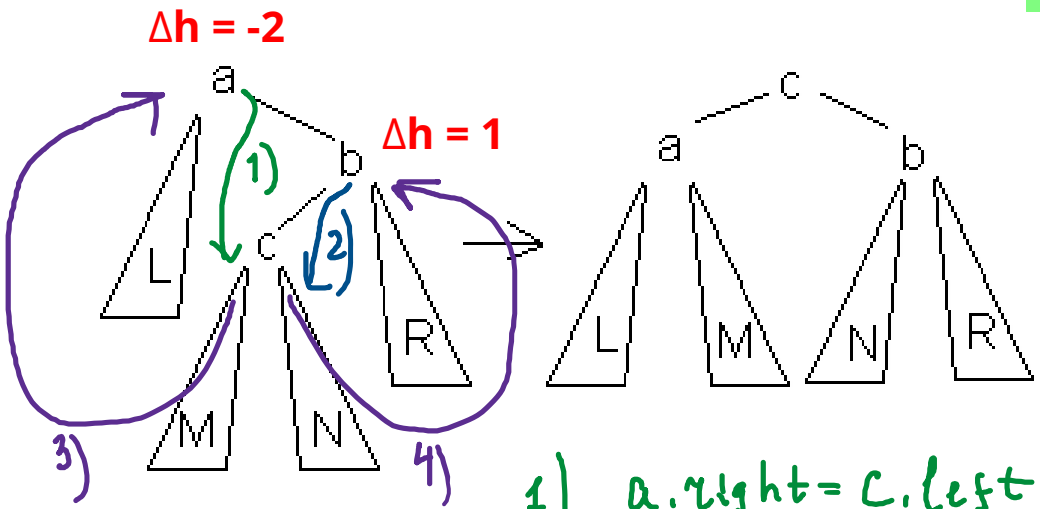
3) c.left = a

4) c.right = b

return c

АВЛ-дерево. Типы вращений

Большое левое вращение:



На что похоже?

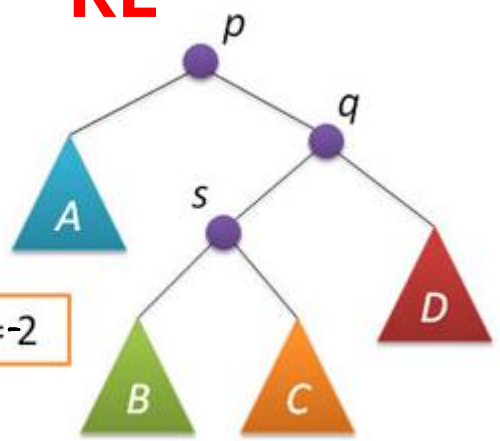
a, b, c – вершины
L, M, N, R - поддеревья

1) a.right = c.left
2) b.left = c.right
3) c.left = a
4) c.right = b
return c

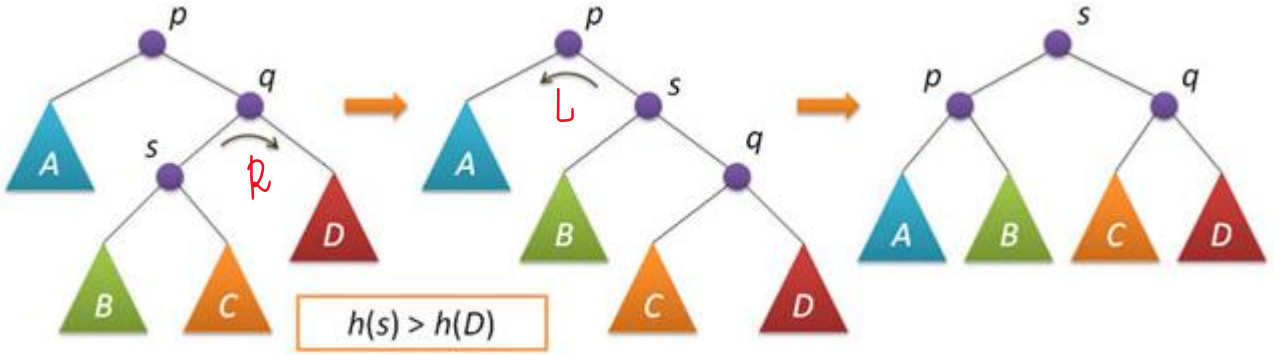
Большой левый поворот

RL

Рассмотрим теперь ситуацию дисбаланса, когда высота правого поддерева узла **p** на **2** больше высоты левого поддерева (обратный случай является симметричным и реализуется аналогично).



$h(A) - h(q) = -2$

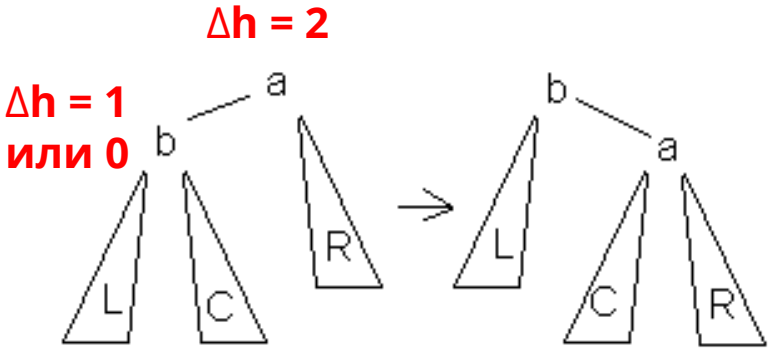


Сперва R
Потом L

RL

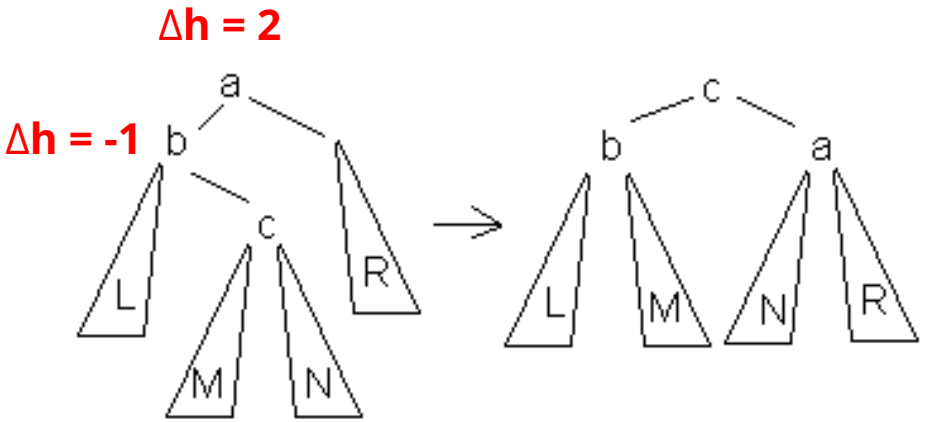
АВЛ-дерево. Типы вращений

Малое правое вращение:



Данное вращение используется, когда
(h(b) — h(R)) == 2 && h(C) <= h(L)

Большое правое вращение:



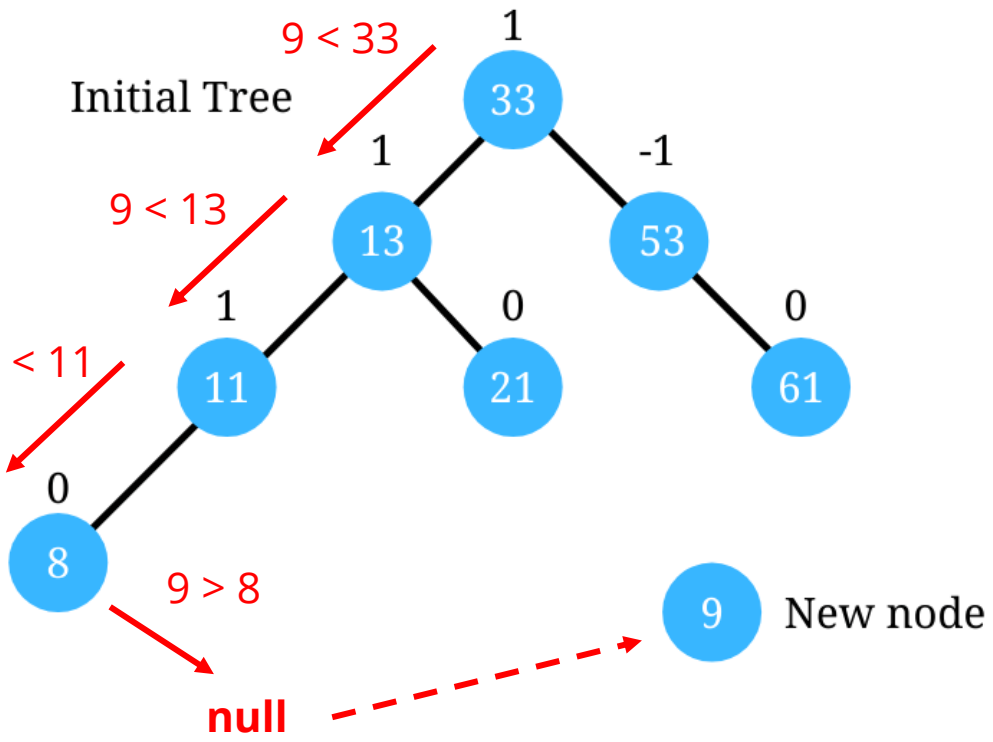
Данное вращение используется, когда
(h(b) — h(R)) == 2 && h(c) > h(L)

Правые вращения симметричны левым

Добавление вершины

- спускаемся вниз по дереву, выбирая правое или левое направление движения в зависимости от результата сравнения ключа в текущем узле и вставляемого ключа.
- при возвращении из рекурсии выполняется балансировка текущего узла.

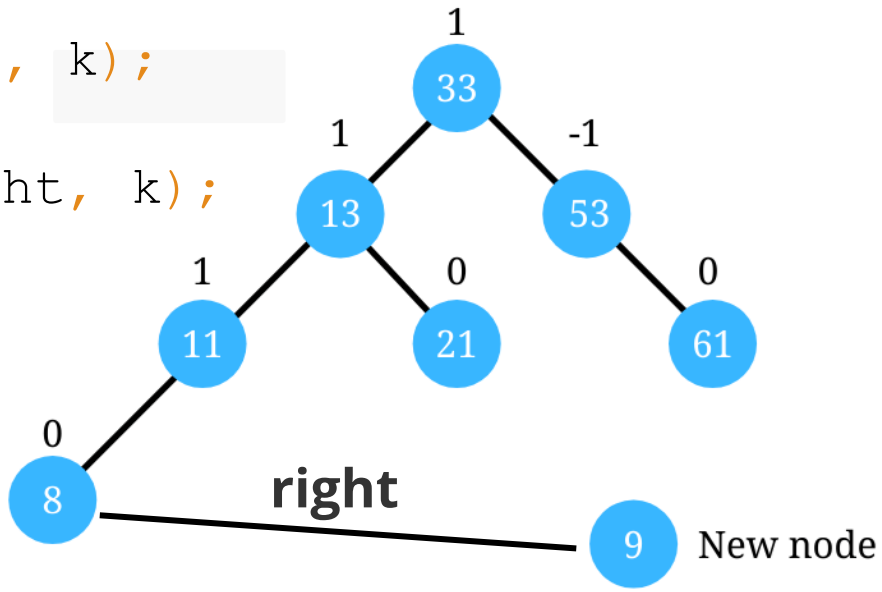
Пытаемся вставить **9**

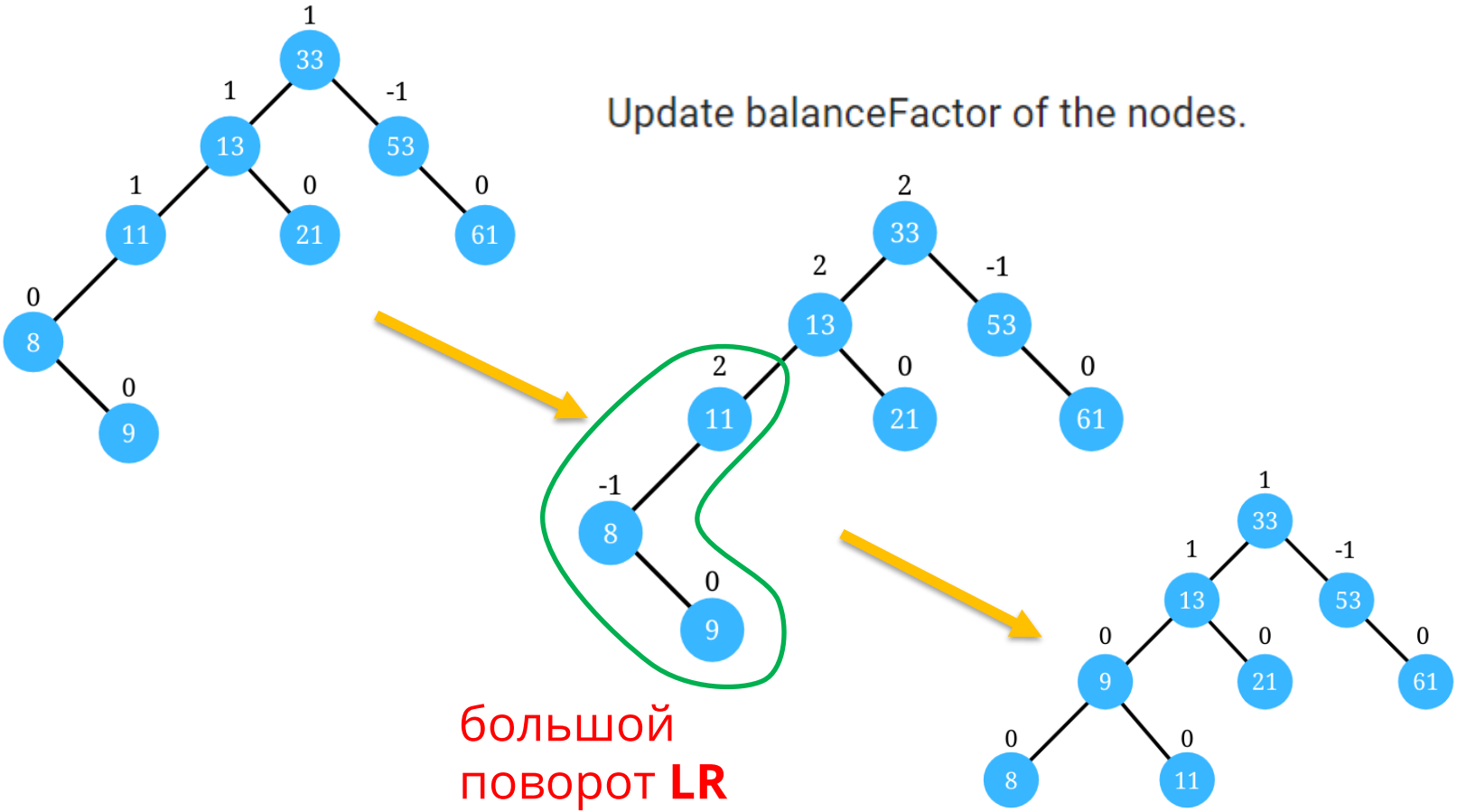


Код добавления вершины

```
Node *Insert(Node *x, int k)
{
  if (!x) return new Node(k);
  if (k<x->key)
      x->left=Insert(x->left, k);
  else
      x->right=Insert(x->right, k);
!!! return Balance(x);
}
```

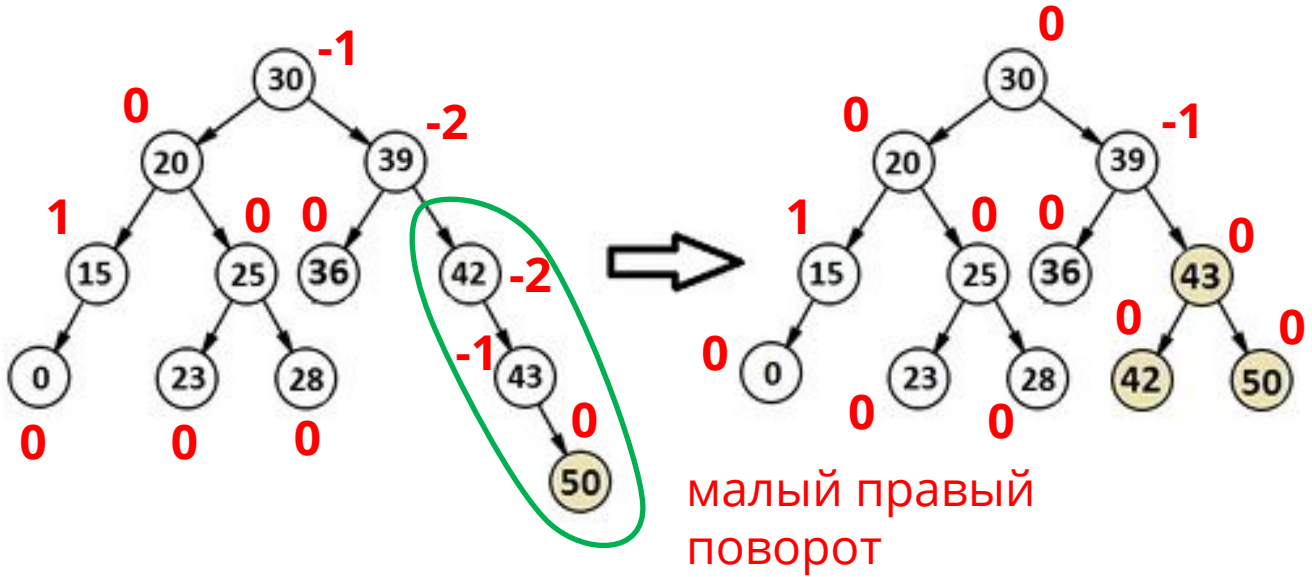
Добавление вершины
требует **O(log n)** операций.




Update balanceFactor of the nodes.
большой
поворот **LR**
Final balanced tree, after rebalancing the node

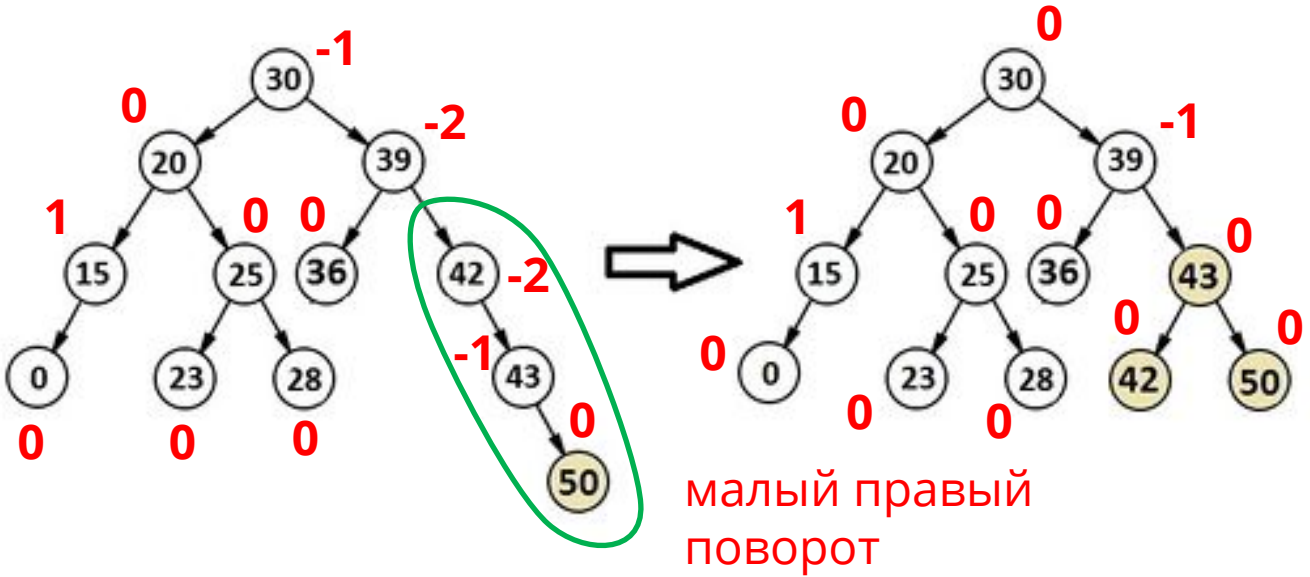
Балансировка узлов

В процессе добавления или удаления узлов в АВЛ-дереве возможно возникновение ситуации, когда **balance factor** некоторых узлов оказывается равными **2** или **-2**, т.е. возникает разбалансировка поддерева.



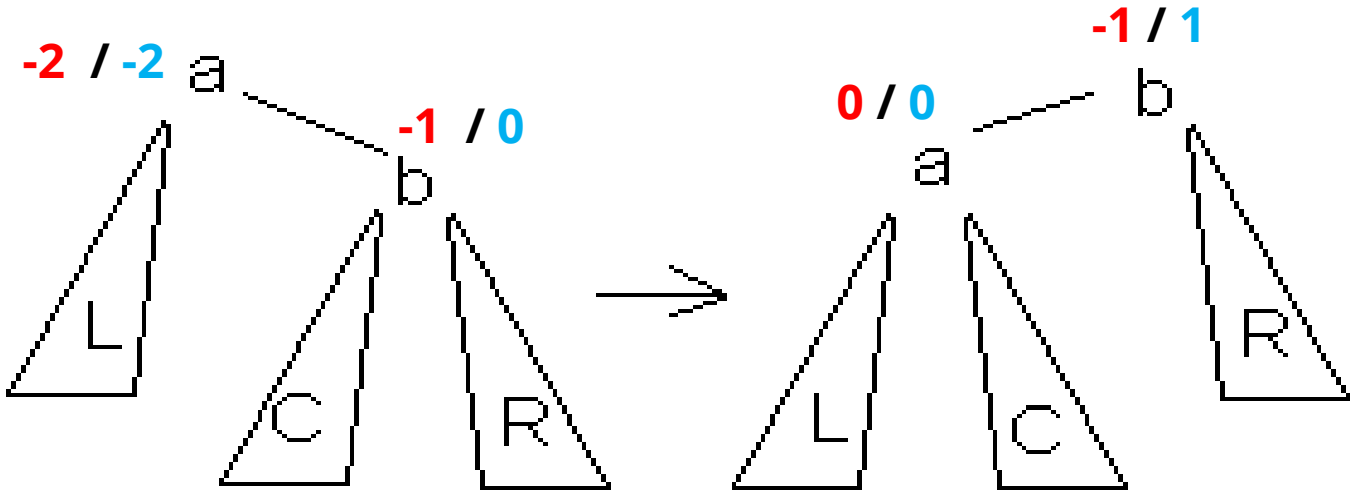
Балансировка узлов

Нужно ли пересчитывать значения всех узлов при балансировке?



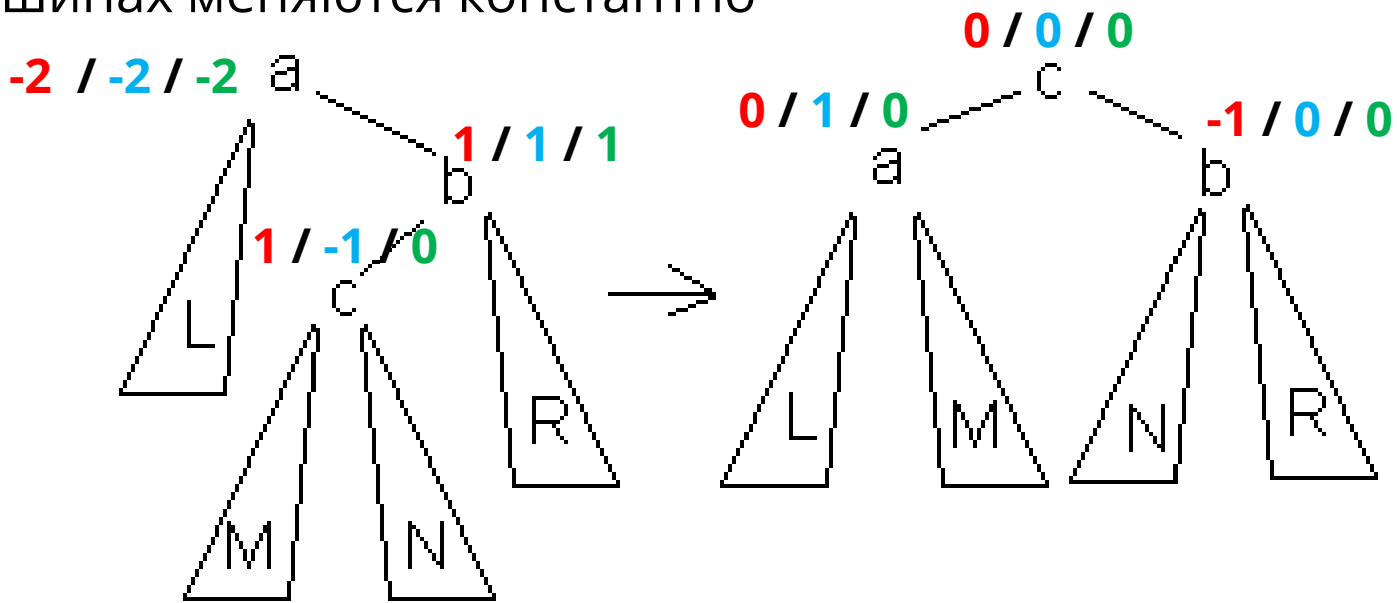
Балансировка узлов. Малое левое вращение

Нужно ли пересчитывать значения всех узлов при балансировке? **Нет**
При каждом отдельном повороте значения баланса в вершинах меняются константно



Балансировка узлов. Большое левое вращение

Нужно ли пересчитывать значения всех узлов при балансировке? **Нет**
При каждом отдельном повороте значения баланса в вершинах меняются константно



Код, выполняющий балансировку

```
Node *balance(Node *p) // балансировка узла p
{
  fix_height(p);
  if (bfactor(p)==-2)
  {
    if (bfactor(p->right)<0);
      p->right=rotate_right(p->right);
    return rotate_left(p);
  }
  if (bfactor(p)==2)
  {
    if (bfactor(p->left)>0);
      p->left=rotate_left(p->left);
    return rotate_right(p);
  }
  return p; // балансировка не нужна
}
```

R

L

L

R

Код, выполняющий балансировку

```
Node *balance(Node *p) // балансировка узла p
{
  fix_height(p);
  if (bfactor(p)==-2)
  {
    if (bfactor(p->right)<0);
      p->right=rotate_right(p->right);
    return rotate_left(p);
  }
  if (bfactor(p)==2)
  {
    if (bfactor(p->left)>0);
      p->left=rotate_left(p->left);
    return rotate_right(p);
  }
  return p; // балансировка не нужна
}
```

R

L

L

R

Что это на самом деле такое?

Код, выполняющий балансировку

```
Node *balance(Node *p) // балансировка узла p
{
  fix_height(p);
  if (bfactor(p)==-2)
  {
    if (bfactor(p->right)> 0);
      p->right=rotate_right(p->right);
    return rotate_left(p);
  }
  if (bfactor(p)==2)
  {
    if (bfactor(p->left)<0);
      p->left=rotate_left(p->left);
    return rotate_right(p);
  }
  return p; // балансировка не нужна
}
```

big_rotate_left
else:
rotate_left

big_rotate_right
else:
rotate_right

Когда остановиться?

Нужно ли на возврате пересчитывать баланс каждой вершины?

Когда остановиться?

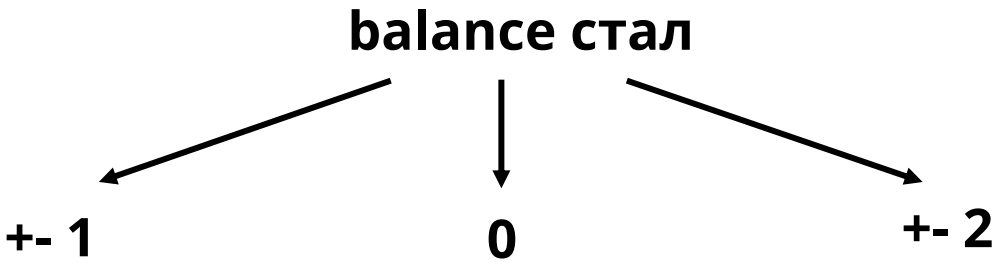
Нужно ли на возврате пересчитывать баланс каждой вершины? **Нет**

Когда остановиться?

Нужно ли на возврате пересчитывать баланс каждой вершины? **Нет**

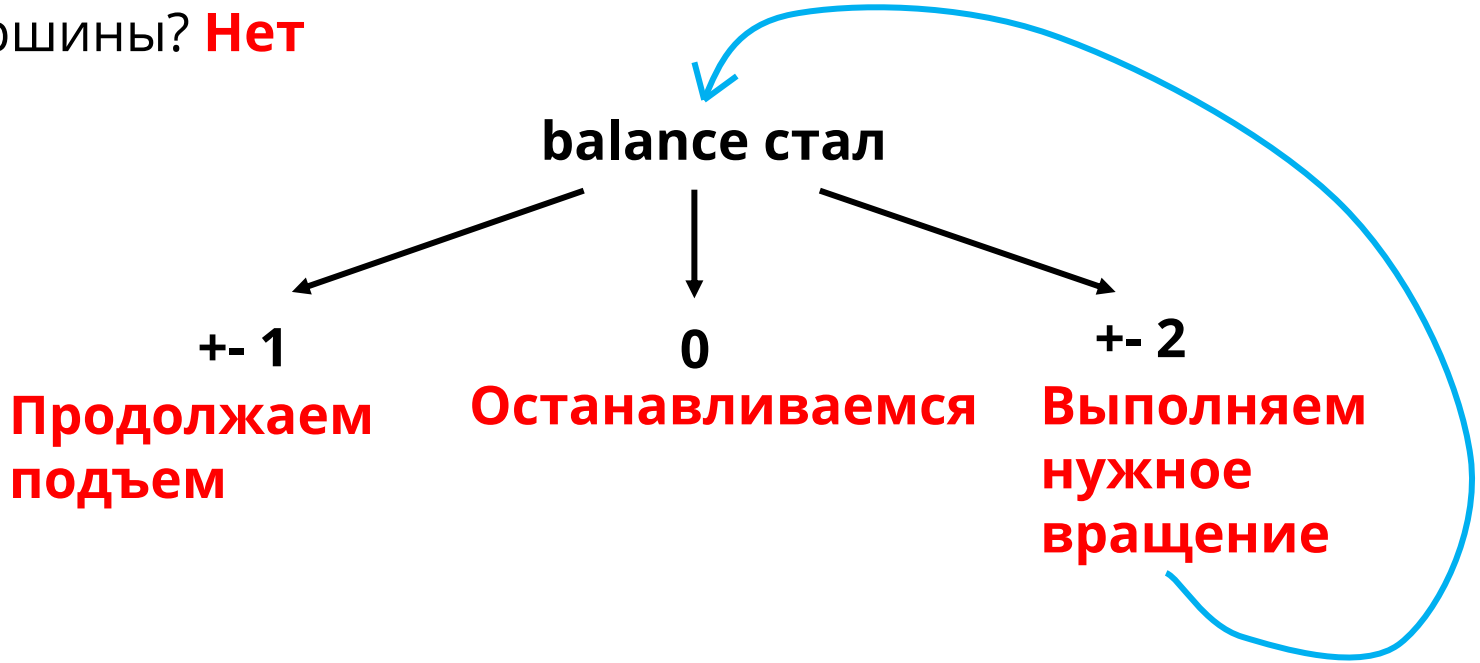
Когда остановиться?

Нужно ли на возврате пересчитывать баланс каждой вершины? **Нет**



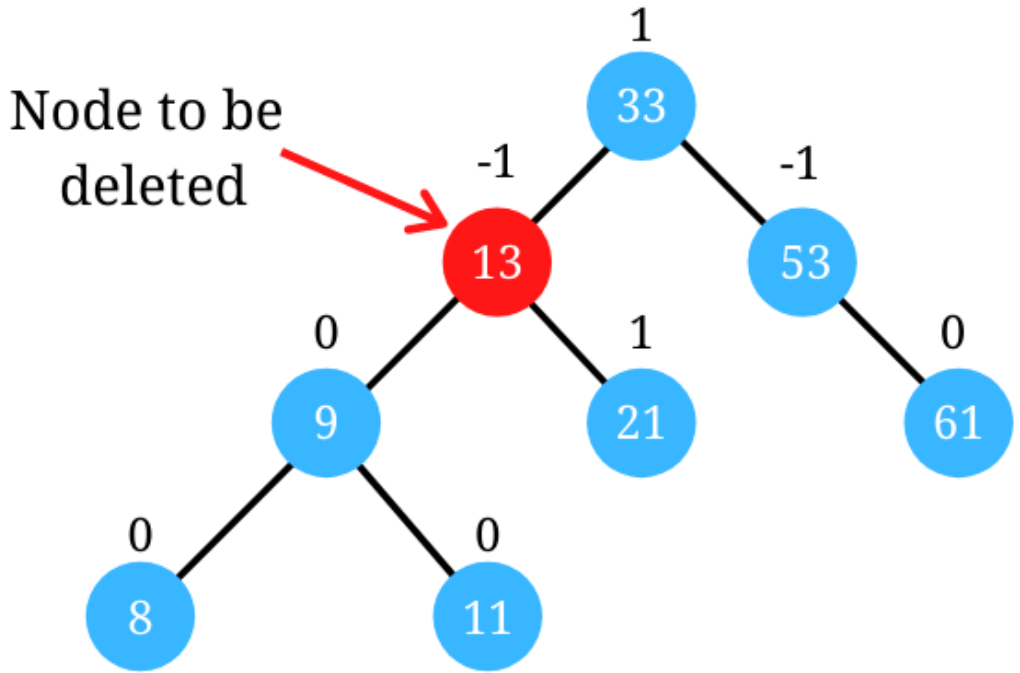
Когда остановиться?

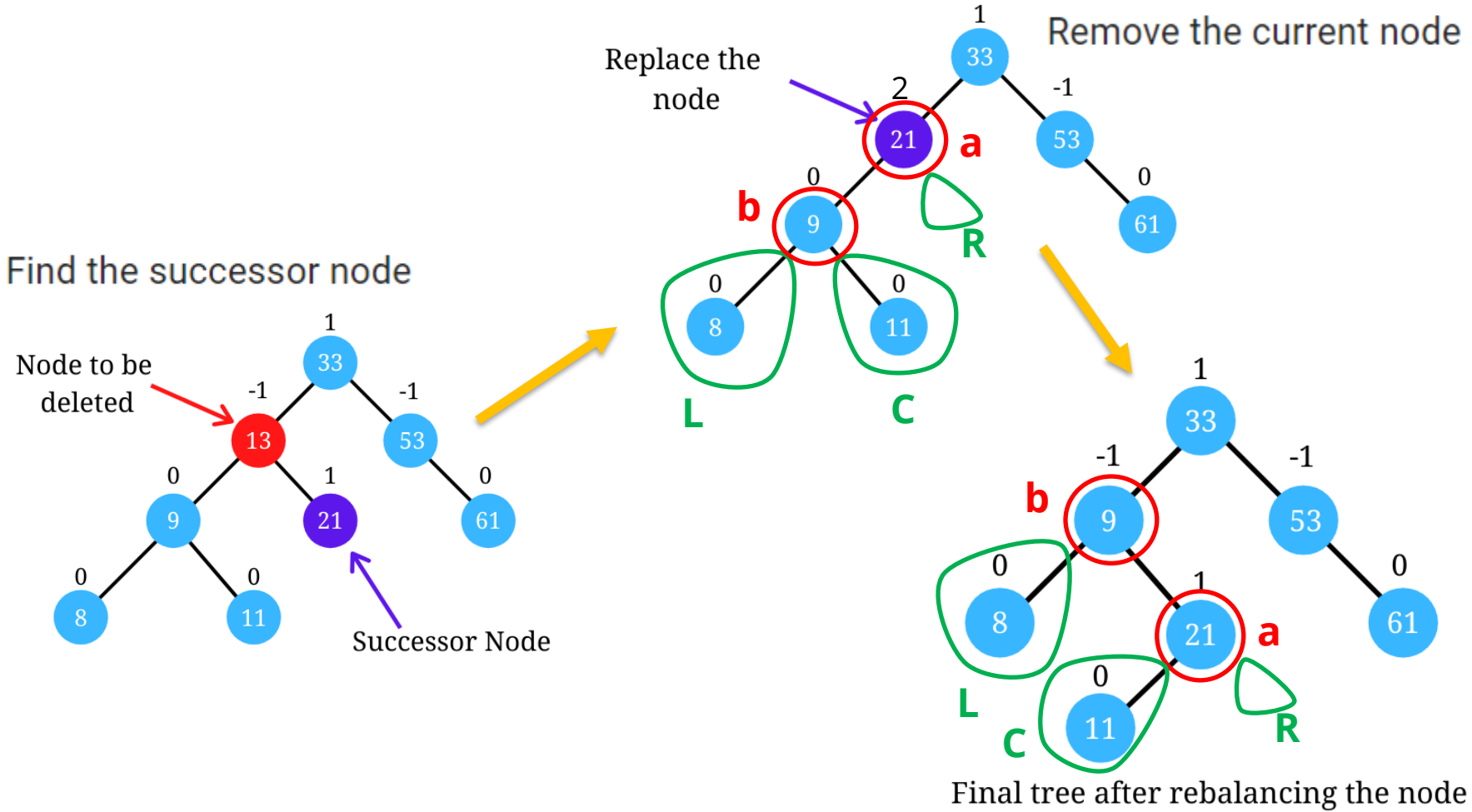
Нужно ли на возврате пересчитывать баланс каждой вершины? **Нет**



Удаление вершины

Идея следующая: находим узел **p** с заданным ключом **k** (если не находим, то делать ничего не надо), в правом поддереве находим узел **min** с наименьшим ключом и заменяем удаляемый узел **p** на найденный узел **min**.




Find the successor node
Node to be deleted
Successor Node
Replace the node
Remove the current node
a
b
R
L
C
Final tree after rebalancing the node

Код удаления вершины

```
Node *Delete(Node *x, int k)
{
    if (!x) return 0;
    if (k<x->key)
        x->left=Delete(x->left, k);
    else if (k>x->key)
        x->right=Delete(x->right, k);
    else
    {
        Node *y=x->left;
        Node *z=x->right;
        if (!z) return y;
        Node* min=SearchMin(z);
        min->right=DeleteMin(z);
        min->left=y;
        delete x;
        return Balance(min);
    }
     return Balance(x);
}
```

Удаление вершины требует **O(log n)** операций.