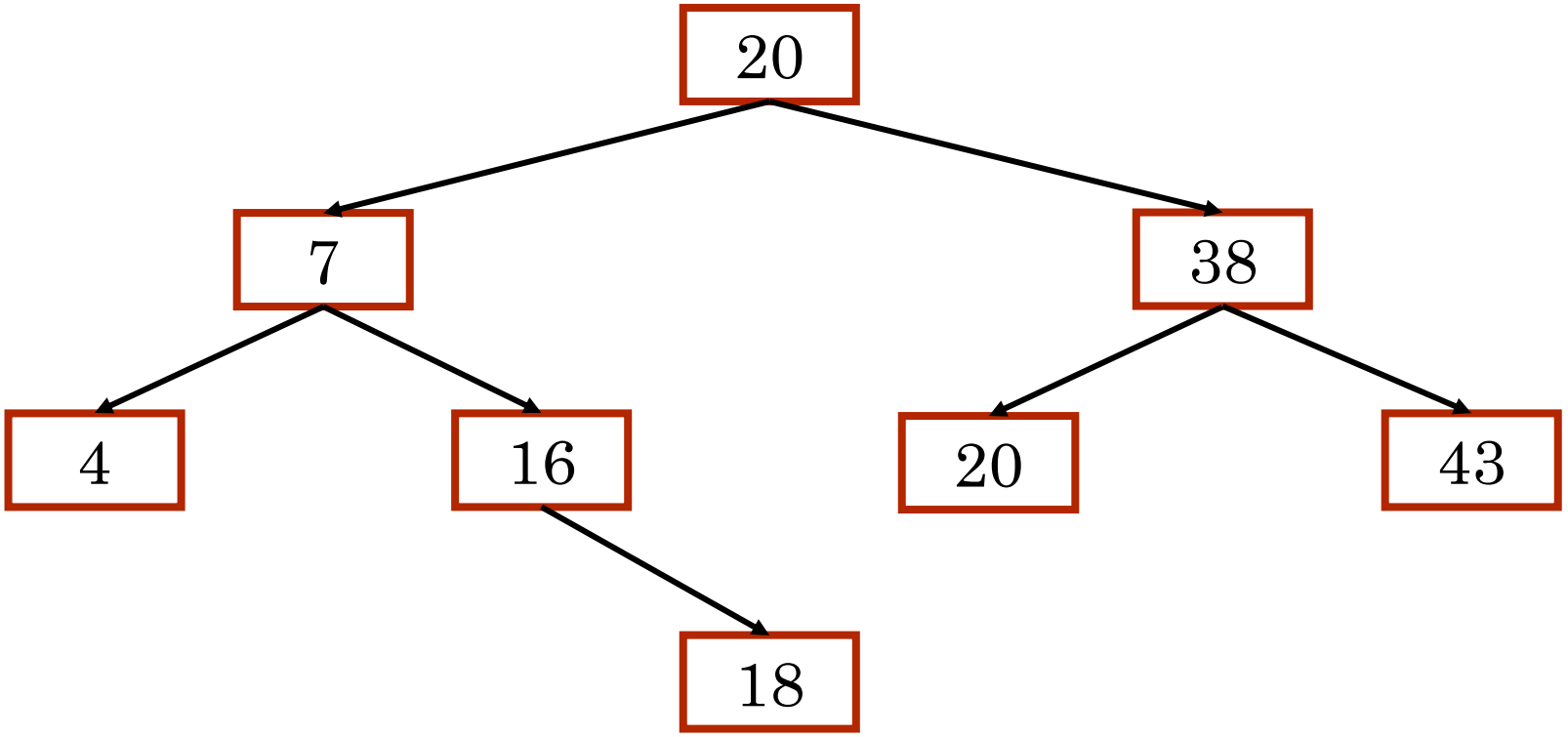
AVL- деревья

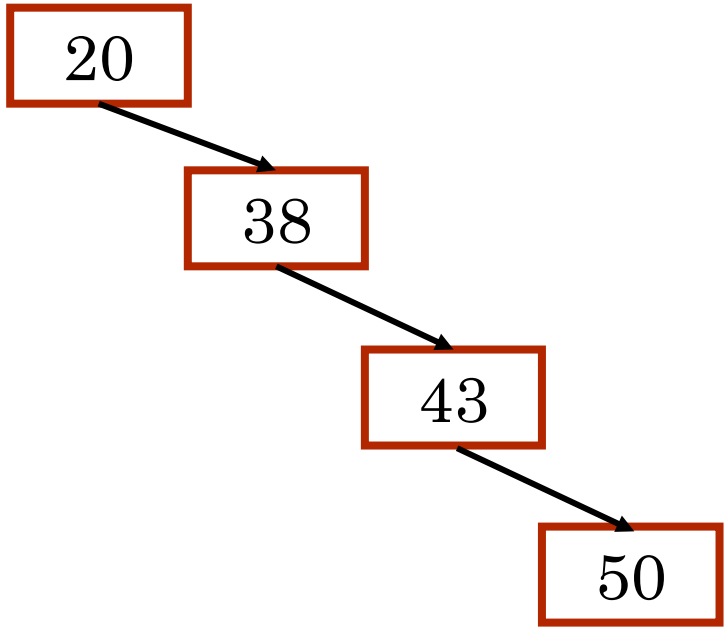
Двоичное дерево поиска (Binary Search Tree)



8 узлов, глубина = 3

Двоичное дерево поиска (Binary Search Tree)

В случае добавления элементов по возрастанию или убыванию дерево может выродиться в **список.**



Сбалансированные деревья поиска

Сбалансированными деревьями поиска называются деревья, в котором высота левого и правого поддеревьев любого узла отличается не более, чем на n единиц.

Виды сбалансированных деревьев:

- AVL-деревья
- Красно-черные деревья (Red-black tree)
- B-деревья

AVL-деревья

Авторы:

Г.М. **А**дельсон-**В**ельский

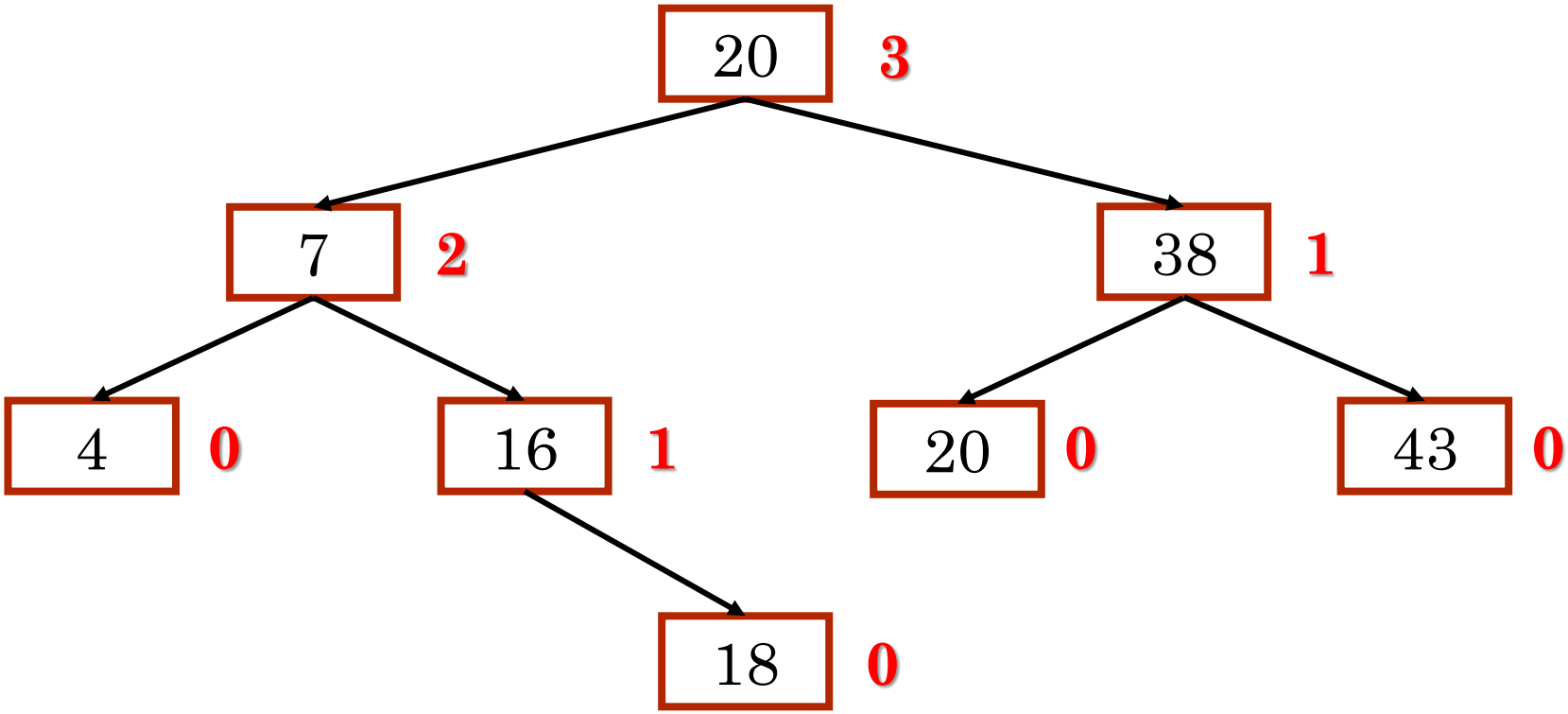
Е.М. **Л**андис

АВЛ-дерево – дерево, в котором высота левого и правого поддеревьев любого узла отличается не более, чем на 1.

AVL-деревья

- В случае, если при вставке или удалении узла нарушается сбалансированность дерева, выполняется его балансировка.
- В AVL-дереве коэффициент сбалансированности любого узла (высота дерева) может принимать значение -1, 0 или 1.
- Высота узла (height) – длина наибольшего пути от него до листа.
- Высота листа равна 0.
- Высота пустого дерева равна -1.

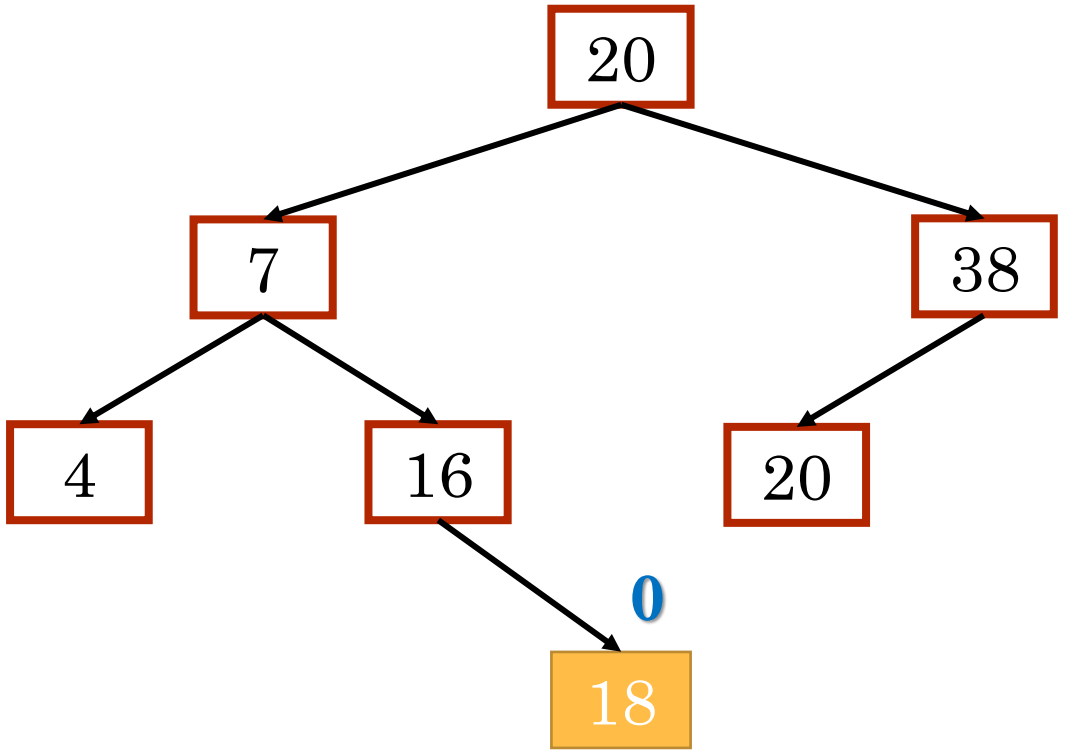
Высота узла



Коэффициент сбалансированности

Balance(t) = Height(Left) – Height(Right)

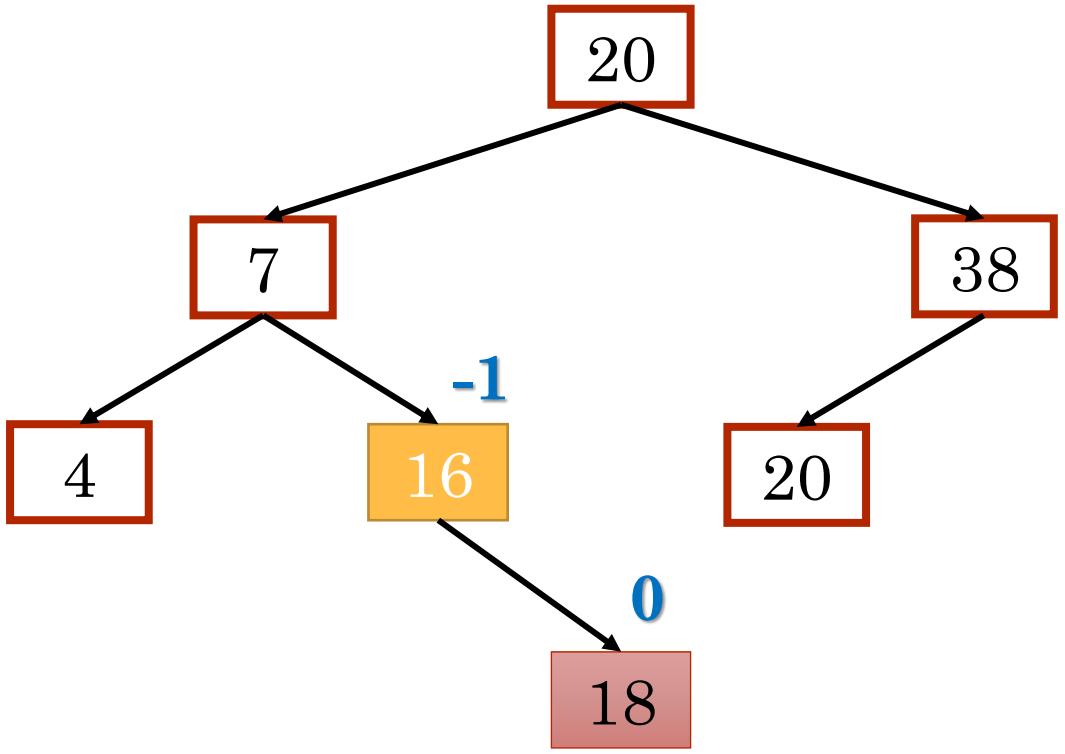
Balance(18) =
= Height(Left) –
Height(Right) =
= -1 – (-1) = 0



Коэффициент сбалансированности

Balance(t) = Height(Left) – Height(Right)

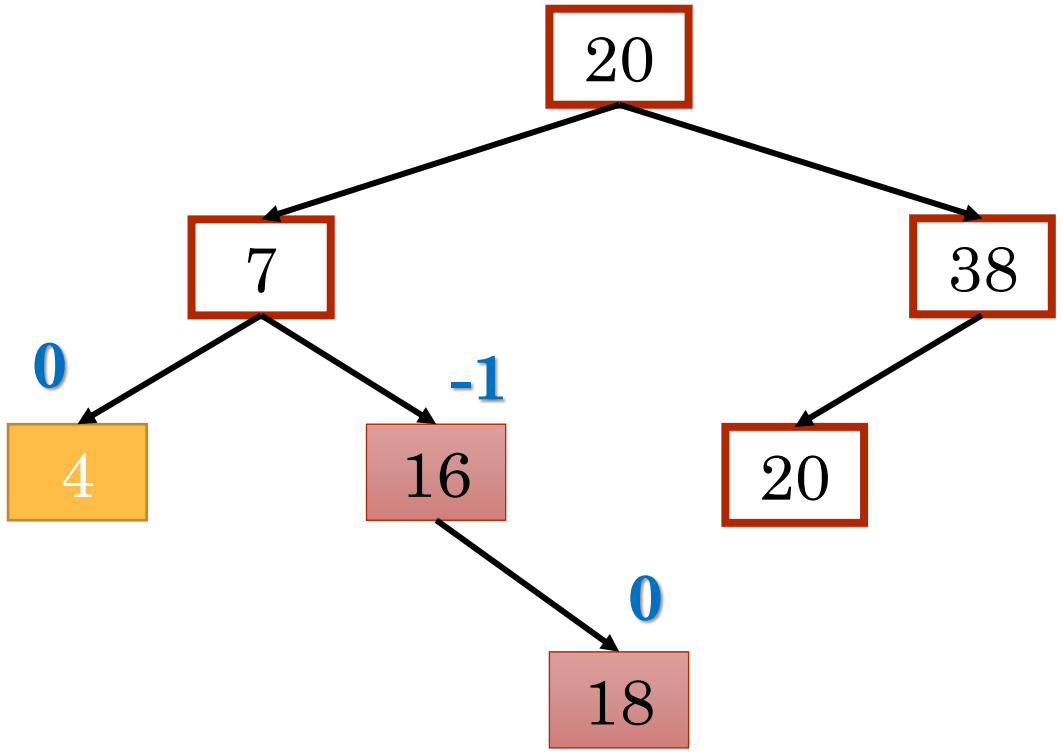
Balance(16) =
= Height(Left) –
Height(Right) =
= -1 – (0) = -1



Коэффициент сбалансированности

Balance(t) = Height(Left) – Height(Right)

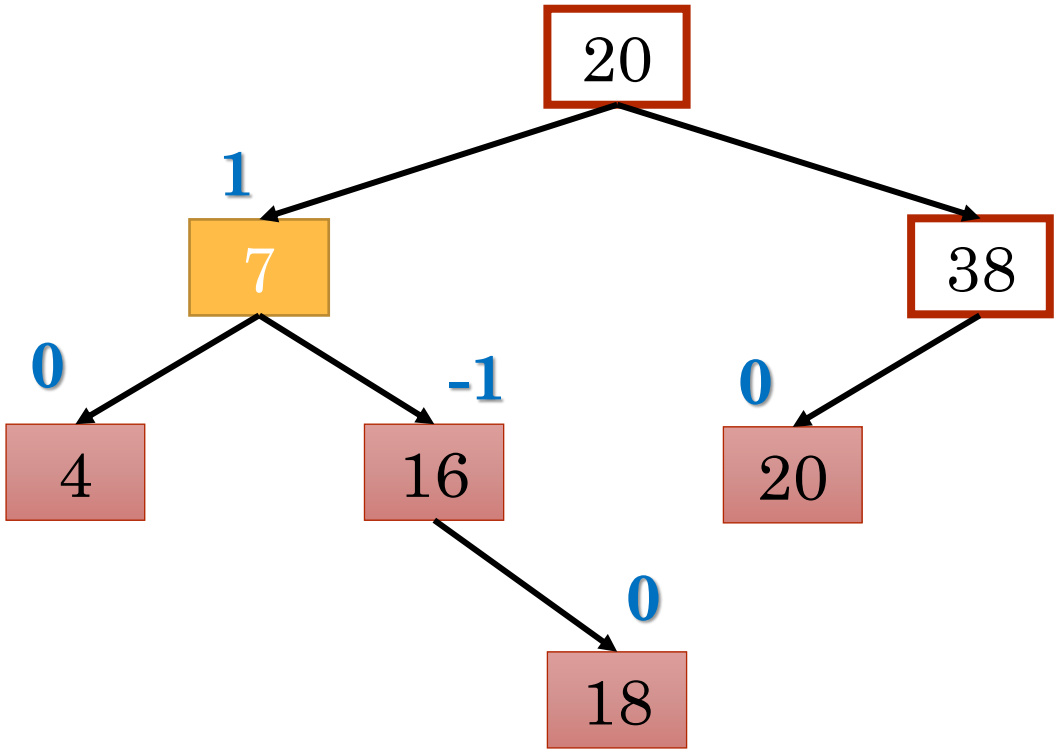
Balance(4) =
= Height(Left) –
Height(Right) =
= -1 – (-1) = 0



Коэффициент сбалансированности

Balance(t) = Height(Left) – Height(Right)

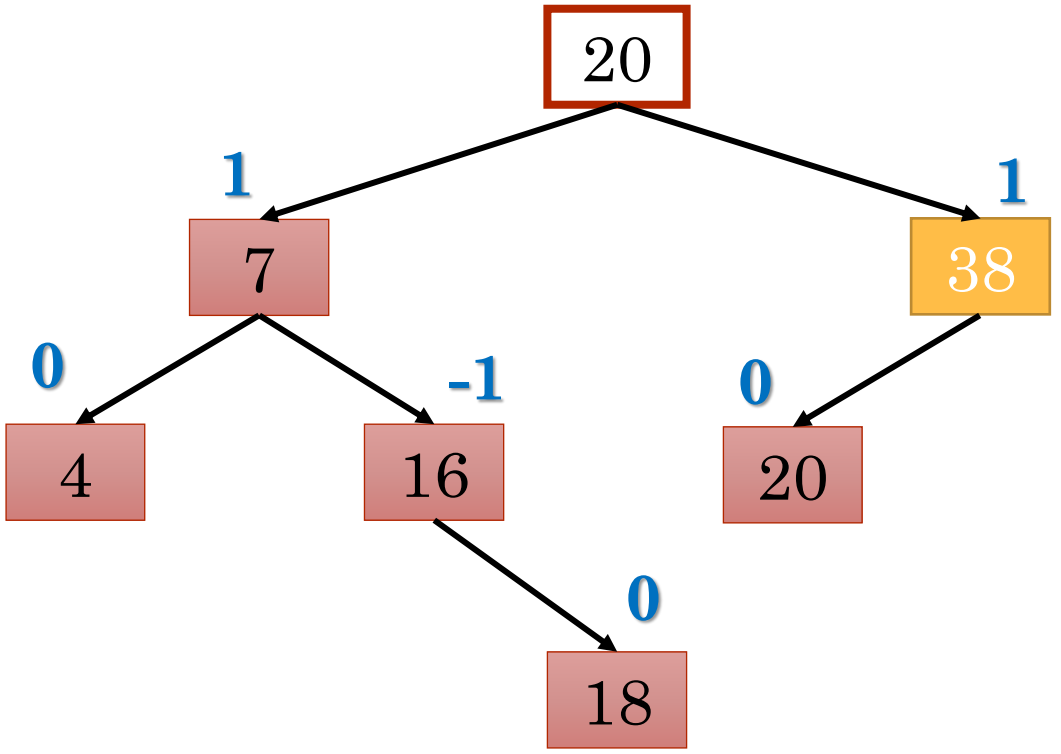
Balance(7) =
= Height(Left) –
Height(Right) =
= 0 – (-1) = 1



Коэффициент сбалансированности

Balance(t) = Height(Left) – Height(Right)

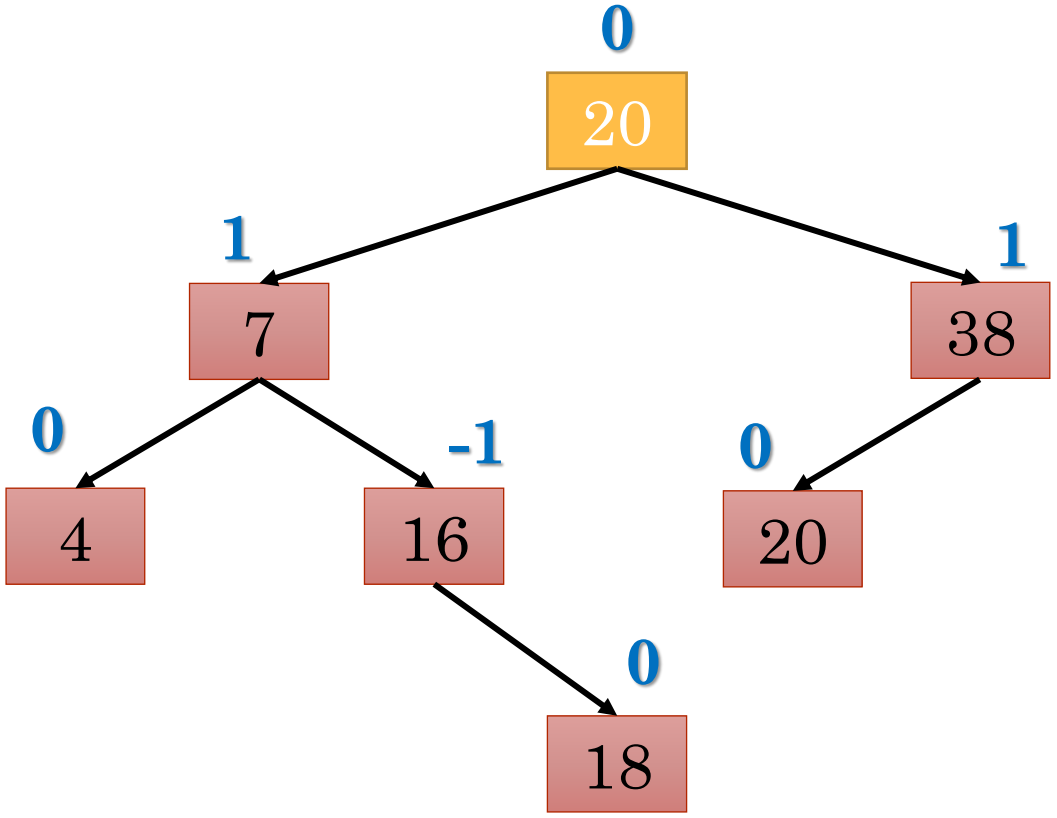
Balance(38) =
= Height(Left) –
Height(Right) =
= 0 – (-1) = 1



Коэффициент сбалансированности

Balance(t) = Height(Left) – Height(Right)

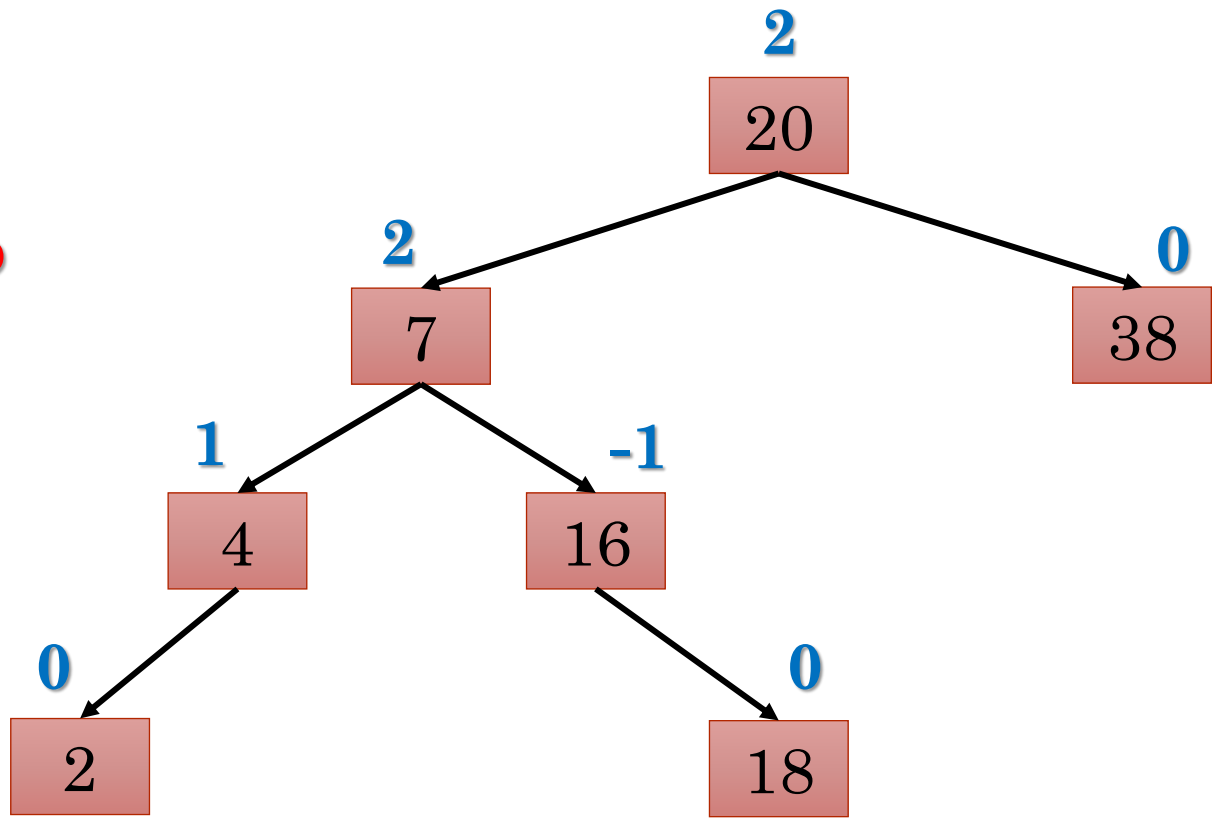
Balance(20) =
= Height(Left) –
Height(Right) =
= 1 – (1) = 0



Коэффициент сбалансированности

Balance(t) = Height(Left) – Height(Right)

НЕ AVL-дерево



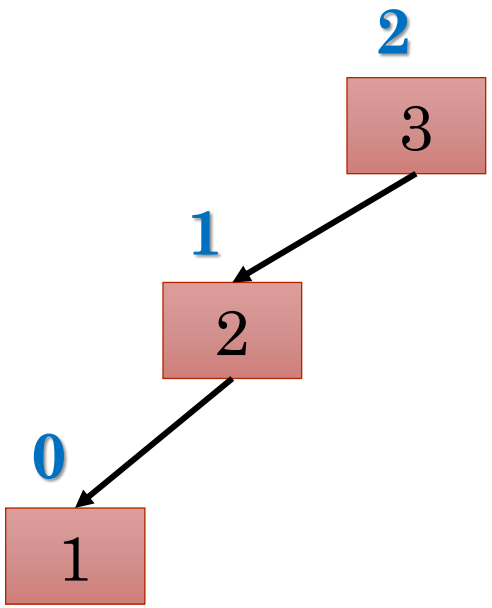
Балансировка дерева

- После добавления нового узла необходимо обновить коэффициенты сбалансированности родительских узлов.

- Если в родительском узле коэффициент сбалансированности стал равным 2 или -2, необходимо выполнить балансировку с помощью **поворотов**.

- **Типы поворотов**:
  - Одиночный правый поворот (R-rotation)
  - Одиночный левый поворот (L-rotation)
  - Двойной лево-правый поворот (LR-rotation)
  - Двойной право-левый поворот (RL-rotation)

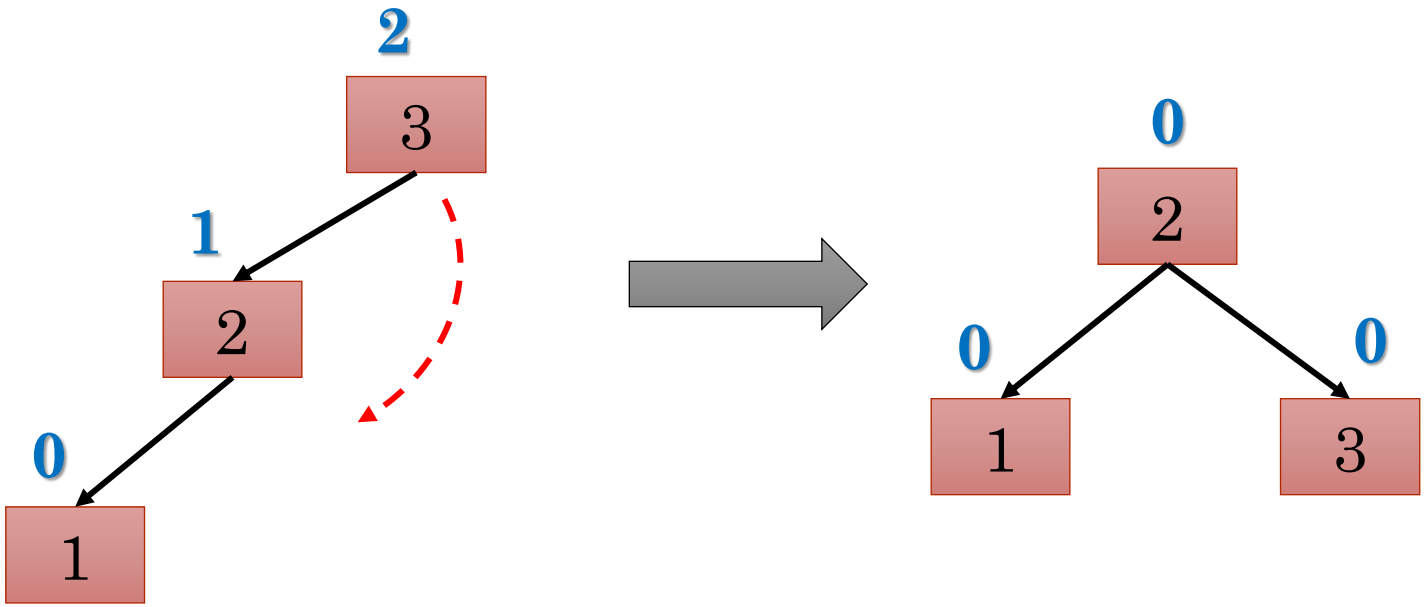
Одиночный правый поворот (R-rotation)



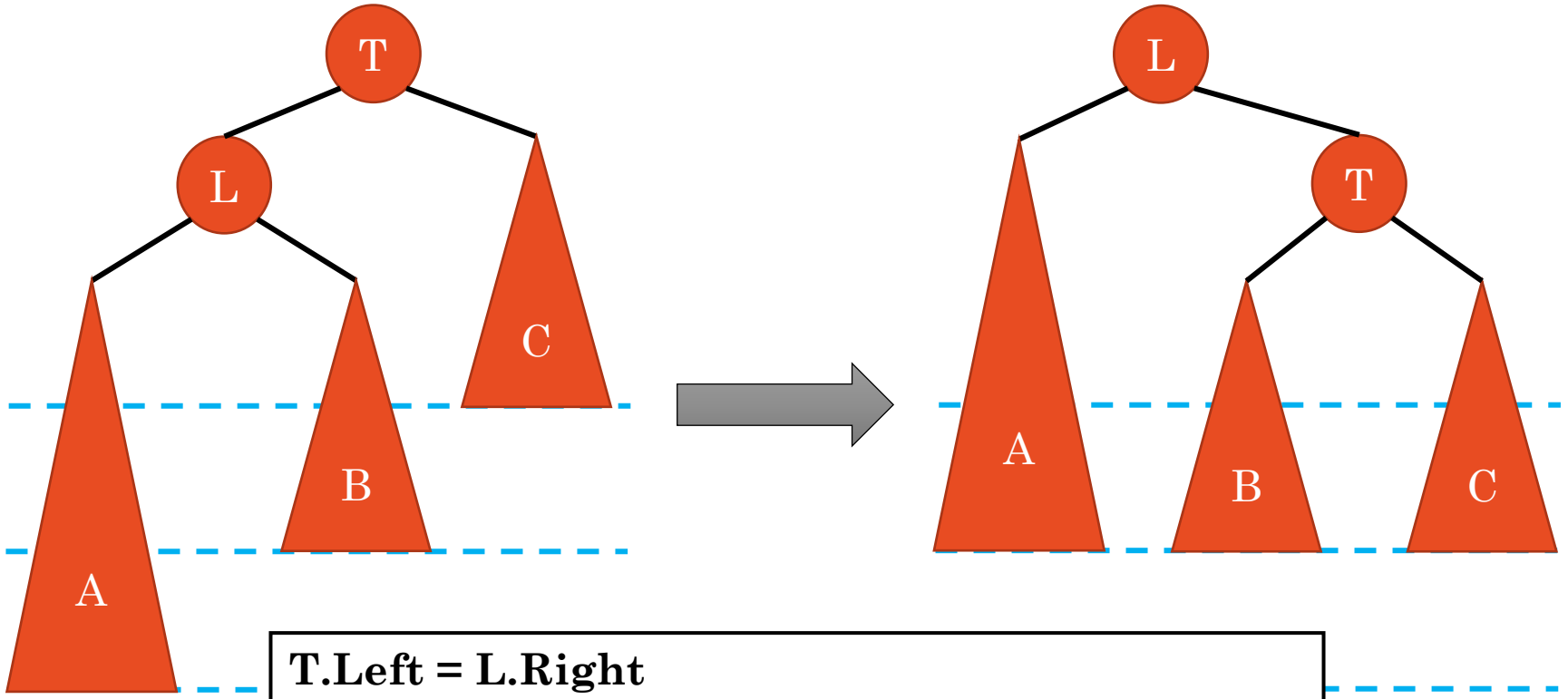
- После добавления элемента 1 дерево перестает быть сбалансированным.
- Height(Left) = 1 > Height(Right) = -1
- Необходимо увеличить высоту правого поддерева

Одиночный правый поворот (R-rotation)

Поворачиваем ребро, связывающее ***корень*** и его ***левый*** дочерний узел ***вправо***.



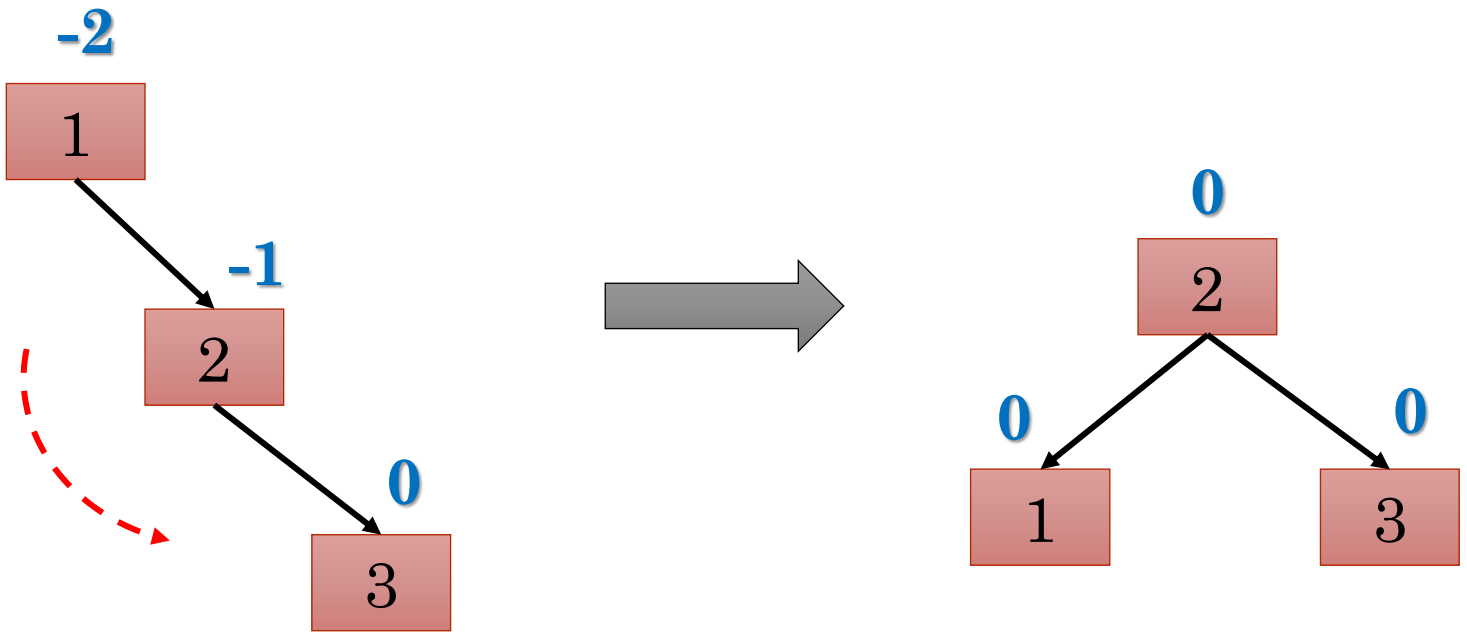
Одиночный правый поворот в общем случае



T.Left = L.Right
L.Right = T
T.Height = max(T.Left.Height, T.Right.Height) + 1
L.Height = max(L.Left.Height, T.Height) + 1

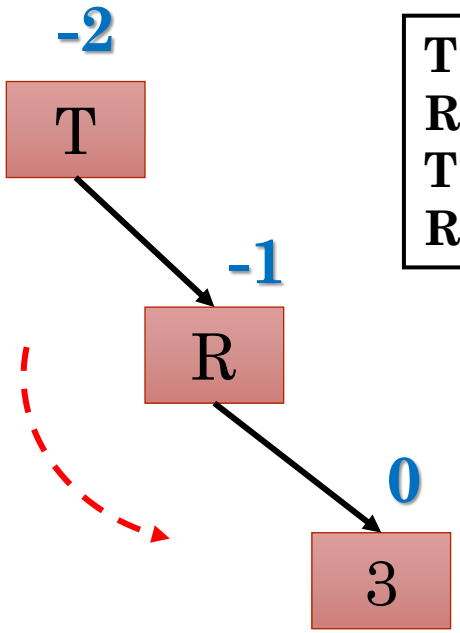
Одиночный левый поворот (L-rotation)

Поворачиваем ребро, связывающее ***корень*** и его ***правый*** дочерний узел ***влево***.



Одиночный левый поворот (L-rotation)

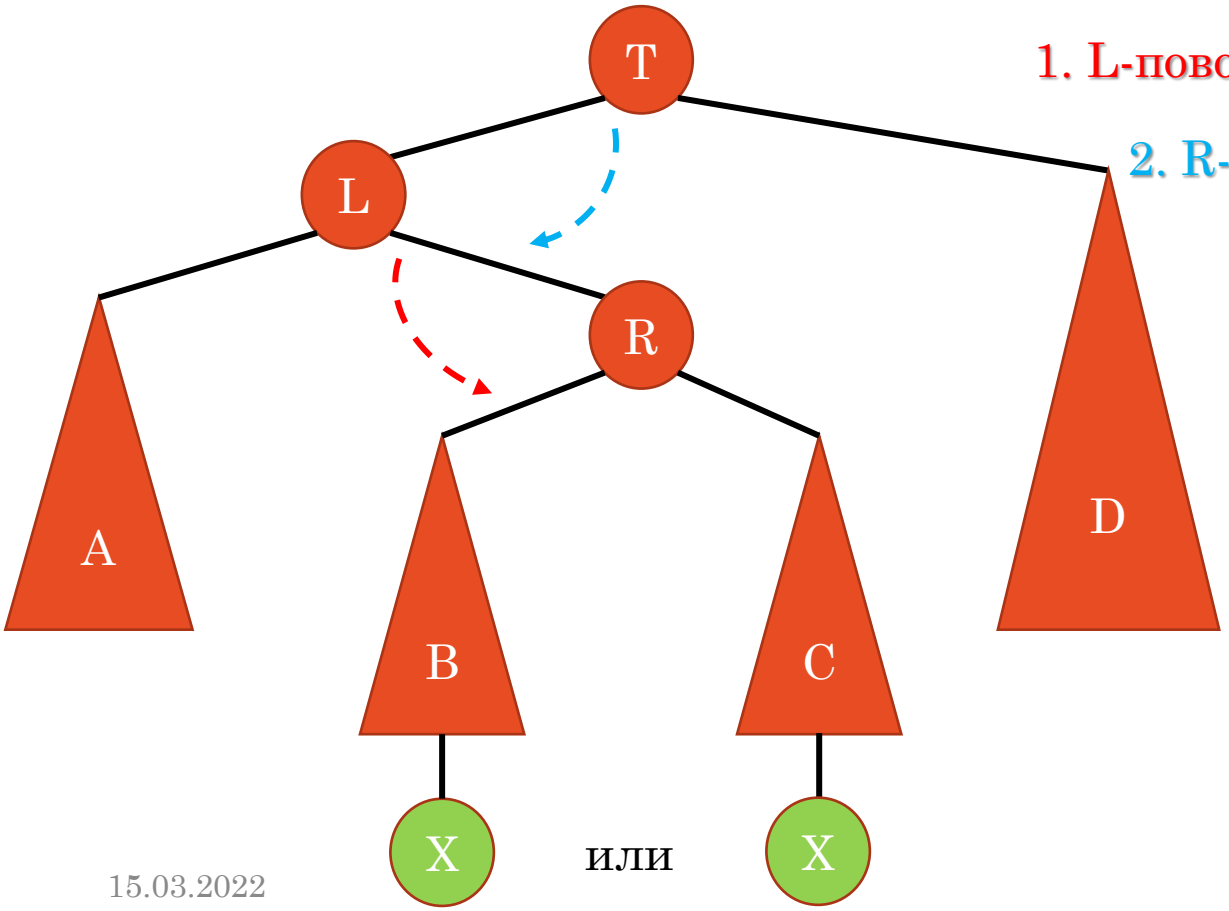
Поворачиваем ребро, связывающее ***корень*** и его ***правый*** дочерний узел ***влево***.



T.Right = R.Left
R.Left = T
T.Height = max(T.Left.Height, T.Right.Height) + 1
R.Height = max(R.Right.Height, T.Height) + 1

Двойной лево-правый поворот (LR-rotation)

LR-поворот выполняется после добавления элемента в **правое** поддерево **левого** дочернего узла дерева.

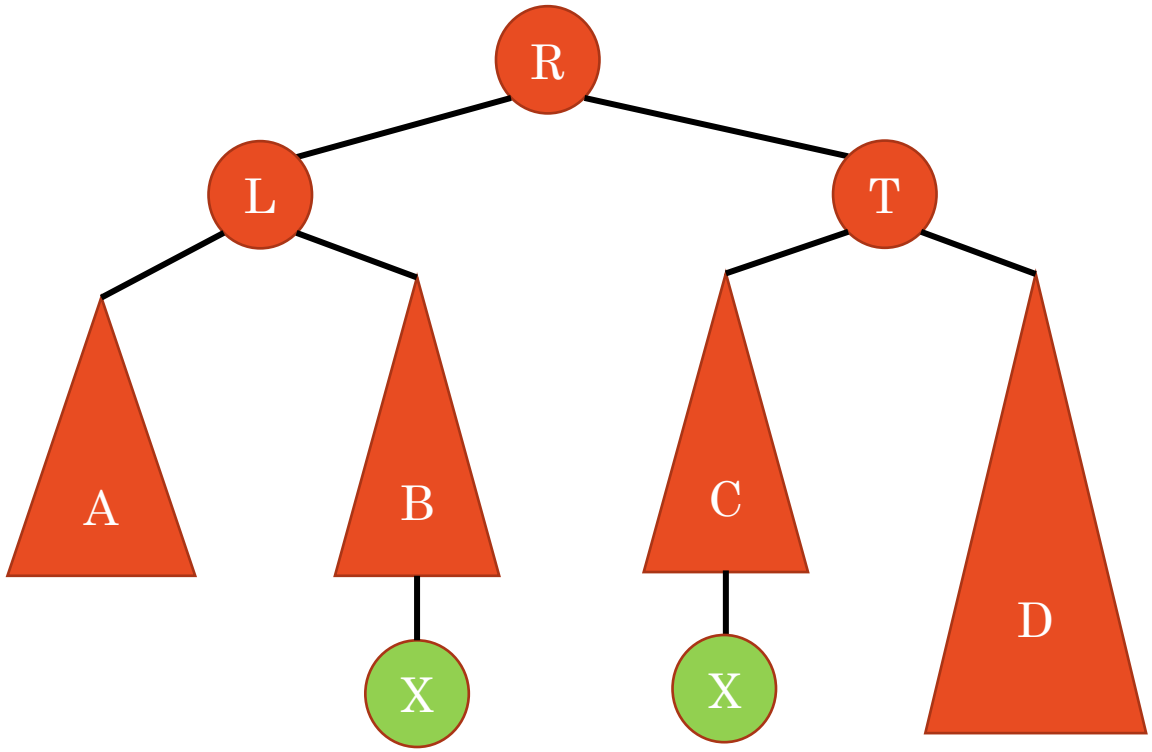


1. L-поворот левого поддерева T

2. R-поворот нового поддерева T

Двойной лево-правый поворот (LR-rotation)

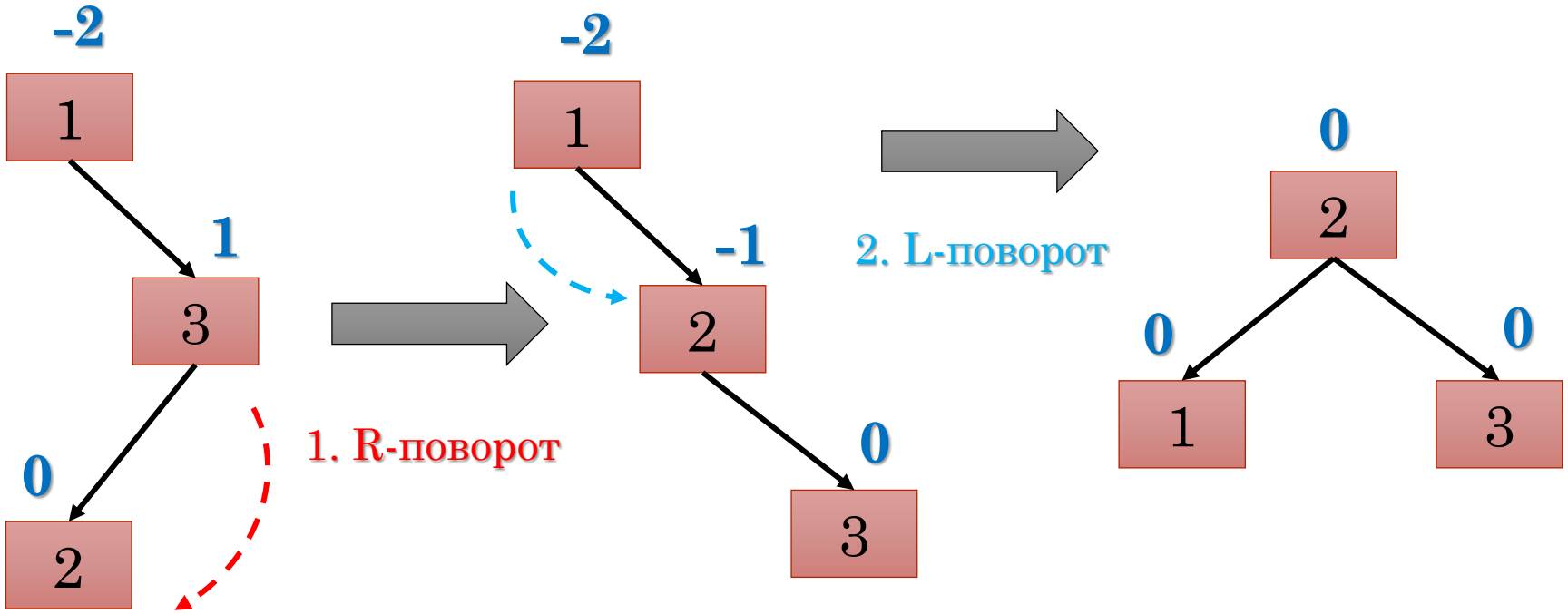
LR-поворот выполняется после добавления элемента в **правое** поддерево **левого** дочернего узла дерева.



или

Двойной право-левый поворот (RL-rotation)

RL-поворот выполняется после добавления элемента в **левое** поддерево **правого** дочернего узла дерева.



Структура узла AVL-дерева

```
struct AvlTree
{
    int _data;
    int _height;
    AvlTree * _left, * _right;
};
```

Построение дерева

```
int main()
{
        AvlTree * root = nullptr;
        root = Insert(root, 1);
        root = Insert(root, 3);
        root = Insert(root, 5);
        root = Insert(root, 7);
        root = Insert(root, 6);
        Print(root, 3);
        system("pause");
}
```

```
left rotate
right left rotate

        7, 0
    6, 1
        5, 0
  3, 2
    1, 0
Для продолжения нажмите любую клавишу . . .
```

Создание узла

```
AvlTree * CreateNode(int data)
{
    AvlTree * node = new AvlTree;

    node->_data = data;
    node->_left = nullptr;
    node->_right = nullptr;
    node->_height = 0;

    return node;
}
```

Высота узла (поддерева), максимальная высота

```
int MaxHeight(int h1, int h2)
{
     return h1 > h2 ? h1 : h2;
}

int Height(AvlTree *t)
{
     return (t != nullptr) ?
          t->_height : -1;
}
```

Добавление узла

```
AvlTree * Insert(AvlTree *t, int
data)
{
    if (t == nullptr)
    {
        return CreateNode(data);
    }
```

Добавление узла (продолжение)

```
if (data < t->_data) {
        t->_left = Insert(t->_left, data);
        if (Height(t->_left) - Height(t->_right) == 2) {
                // дерево разбалансировано
                if (data < t->_left->_data)
                {
                        t = RightRotate(t);
                        cout << "right rotate" << endl;
                }
                else
                {
                        t = LeftRightRotate(t);
                        cout << "left right rotate" << endl;
                }
        }
}
```

Добавление узла (продолжение)

```
else if (data > t->_data) {
        t->_right = Insert(t->_right, data);
        if (Height(t->_right) - Height(t->_left) == 2)
        {
                // дерево разбалансировано
                if (data > t->_right->_data) {
                        t = LeftRotate(t);
                        cout << "left rotate" << endl;
                }
                else {
                        t = RightLeftRotate(t);
                        cout << "right left rotate" << endl;
                }
        }
}
```

Добавление узла (конец)

```
        t->_height=MaxHeight(Height(t->_left),
                             Height(t->_right)) + 1;
        return t;
}
```

R-поворот

```
AvlTree * RightRotate(AvlTree * t)
{
        AvlTree * left;

        left = t->_left;
        t->_left = left->_right;
        left->_right = t;

        t->_height = MaxHeight(Height(t->_left),
                Height(t->_right)) + 1;
        left->_height = MaxHeight(Height(left->_left),
                t->_height) + 1;
        return left;
}
```

L-поворот

```
AvlTree * LeftRotate(AvlTree * t)
{
        AvlTree * right;
        right = t->_right;
        t->_right = right->_left;
        right->_left = t;

t->_height = MaxHeight(Height(t->_left),Height(t->_right)) + 1;
right->_height = MaxHeight(Height(right->_right),t->_height)+1;

        return right;
}
```

Двойные повороты

```
AvlTree * LeftRightRotate(AvlTree *t)
{
      t->_left = LeftRotate(t->_left);
      return RightRotate(t);
}

AvlTree * RightLeftRotate(AvlTree *t)
{
      t->_right = RightRotate(t->_right);
      return LeftRotate(t);
}
```

Печать дерева по уровням

```
void PrintByLevel(AvlTree * t, int level)
{
	int i;
	if (t == nullptr) {
		return;
	}
	for (i = 0; i < level; i++) {
		cout << "\t";
	}
	cout << t->_data << endl;
	PrintByLevel(t->_left, level++);
	PrintByLevel(t->_right, level++);
}
```

Вывод дерева