Домашнее задание

Пример идеального решения

```
public class Tree<V extends Comparable<V>> {
    private Node root;

    public boolean add(V value) {
        if (root != null) {
            boolean result = addNode(root, value);
            root = rebalance(root);
            root.color = Color.BLACK;
            return result;
        } else {
            root = new Node();
            root.color = Color.BLACK;
            root.value = value;
            return true;
        }
    }
    private boolean addNode(Node node, V value) {
        if (node.value == value) {
            return false;
        } else {
            if (node.value.compareTo(value) > 0) {
                if (node.left != null) {
                    boolean result = addNode(node.left, value);
                    node.left = rebalance(node.left);
                    return result;
                } else {
                    node.left = new Node();
                    node.left.color = Color.RED;
                    node.left.value = value;
                    return true;
                }
            } else {
                if (node.right != null) {
                    boolean result = addNode(node.right, value);
```

```
                    node.right = rebalance(node.right);
                    return result;
                } else {
                    node.right = new Node();
                    node.right.color = Color.RED;
                    node.right.value = value;
                    return true;
                }
            }
        }
    }

    private Node rebalance(Node node) {
        Node result = node;
        boolean needRebalance;
        do {
            needRebalance = false;
            if (result.right != null && result.right.color == Color.RED &&
                    (result.left == null || result.left.color ==
Color.BLACK)) {
                needRebalance = true;
                result = rightSwap(result);
            }
            if (result.left != null && result.left.color == Color.RED &&
                    result.left.left != null && result.left.left.color ==
Color.RED) {
                needRebalance = true;
                result = leftSwap(result);
            }
            if (result.left != null && result.left.color == Color.RED &&
                    result.right != null && result.right.color ==
Color.RED) {
                needRebalance = true;
                colorSwap(result);
            }
        }
        while (needRebalance);
        return result;
    }

    private Node rightSwap(Node node) {
        Node rightChild = node.right;
        Node betweenChild = rightChild.left;
        rightChild.left = node;
        node.right = betweenChild;
        rightChild.color = node.color;
        node.color = Color.RED;
        return rightChild;
    }

    private Node leftSwap(Node node) {
        Node leftChild = node.left;
        Node betweenChild = leftChild.right;
        leftChild.right = node;
        node.left = betweenChild;
        leftChild.color = node.color;
        node.color = Color.RED;
        return leftChild;
    }

    private void colorSwap(Node node) {
```

```
        node.right.color = Color.BLACK;
        node.left.color = Color.BLACK;
        node.color = Color.RED;
    }

    private class Node {
        private V value;

        private Color color;
        private Node left;
        private Node right;
    }

    private enum Color {
        RED, BLACK
    }
}
```