Алгоритмы и структуры данных. Обучение в записи

Задание 1. Преобразование дерева поиска в красно-черное дерево

Вам дано бинарное дерево поиска. Напишите метод, который преобразует его в левостороннее красно-черное дерево (РЧД). Красно-черное дерево должно удовлетворять следующим критериям:

1. Каждая нода имеет цвет (красный или черный).
2. Корень дерева всегда черный.
3. Новая нода всегда красная.
4. Красные ноды могут быть только левыми дочерними элементами.
5. У красной ноды все дочерние элементы черного цвета.

Для этого, реализуйте методы добавления новых элементов с балансировкой и выполняйте следующие операции для поддержания свойств РЧД:

- Левый малый поворот
- Правый малый поворот
- Смена цвета

Критерии применения этих операций:

- Если правый дочерний элемент красный, а левый черный, то применяем малый правый поворот.
- Если левый дочерний элемент красный и его левый дочерний элемент тоже красный, то применяем малый левый поворот.
- Если оба дочерних элемента красные, то делаем смену цвета.
- Если корень стал красным, то перекрашиваем его в черный.

Подсказка № 1

Каждый узел должен содержать значение, ссылки на левый и правый дочерние узлы, родительский узел и цвет узла (красный или черный). Новый узел всегда красный.

Подсказка № 2

При добавлении элемента, вставьте его как обычное бинарное дерево поиска, затем проведите балансировку для поддержания свойств красно-черного дерева.

Подсказка № 3

Если правый дочерний узел красный и левый черный, выполните левый малый поворот вокруг текущего узла. Если левый дочерний узел и его левый дочерний узел красные, выполните правый малый поворот вокруг текущего узла. Если оба дочерних узла красные, выполните смену цвета текущего узла и его дочерних узлов.

Подсказка № 4

Убедитесь, что корень дерева всегда черный после всех операций вставки и балансировки.

Подсказка № 5

Реализуйте метод для вывода дерева, чтобы визуализировать его структуру и цвета узлов. Это поможет проверить, что дерево сбалансировано правильно.

Эталонное решение:

```
// Определение класса узла дерева
class Node {
    int value;
    Node left, right, parent;
    boolean isRed;

    Node(int value) {
        this.value = value;
        this.left = null;
        this.right = null;
        this.parent = null;
        this.isRed = true; // Новый узел всегда красный
    }
}
```

```
// Определение класса красно-черного дерева
public class RedBlackTree {
    private Node root;

    // Конструктор
    public RedBlackTree() {
        this.root = null;
    }

    // Добавление элемента в РЧД
    public void add(int value) {
        root = add(root, value);
        root.isRed = false; // Корень всегда черный
    }

    private Node add(Node node, int value) {
        if (node == null) {
            return new Node(value);
        }

        if (value < node.value) {
            node.left = add(node.left, value);
            node.left.parent = node;
        } else if (value > node.value) {
            node.right = add(node.right, value);
            node.right.parent = node;
        }
```

```
        // Балансировка дерева после вставки
        if (isRed(node.right) && !isRed(node.left)) {
            node = rotateLeft(node);
        }
        if (isRed(node.left) && isRed(node.left.left)) {
            node = rotateRight(node);
        }
        if (isRed(node.left) && isRed(node.right)) {
            flipColors(node);
        }

        return node;
    }

    // Проверка, является ли узел красным
    private boolean isRed(Node node) {
        if (node == null) {
            return false;
        }
        return node.isRed;
    }

    // Левый малый поворот
    private Node rotateLeft(Node node) {
        Node x = node.right;
        node.right = x.left;
```

```
        if (x.left != null) {
            x.left.parent = node;
        }
        x.parent = node.parent;
        if (node.parent == null) {
            root = x;
        } else if (node == node.parent.left) {
            node.parent.left = x;
        } else {
            node.parent.right = x;
        }
        x.left = node;
        node.parent = x;
        x.isRed = node.isRed;
        node.isRed = true;
        return x;
    }

    // Правый малый поворот
    private Node rotateRight(Node node) {
        Node x = node.left;
        node.left = x.right;
        if (x.right != null) {
            x.right.parent = node;
        }
        x.parent = node.parent;
        if (node.parent == null) {
```

```
            root = x;
        } else if (node == node.parent.right) {
            node.parent.right = x;
        } else {
            node.parent.left = x;
        }
        x.right = node;
        node.parent = x;
        x.isRed = node.isRed;
        node.isRed = true;
        return x;
    }

    // Смена цвета узла и его дочерних элементов
    private void flipColors(Node node) {
        node.isRed = !node.isRed;
        if (node.left != null) {
            node.left.isRed = !node.left.isRed;
        }
        if (node.right != null) {
            node.right.isRed = !node.right.isRed;
        }
    }

    // Метод для вывода дерева в удобном виде
    public void printTree() {
        printTree(root, "", true);
```

```
    }

    private void printTree(Node node, String indent, boolean last) {
        if (node != null) {
            System.out.print(indent);
            if (last) {
                System.out.print("R----");
                indent += "   ";
            } else {
                System.out.print("L----");
                indent += "|  ";
            }
            System.out.println(node.value + (node.isRed ? "(R)" :
"(B)"));
            printTree(node.left, indent, false);
            printTree(node.right, indent, true);
        }
    }

    // Основной метод для тестирования
    public static void main(String[] args) {
        RedBlackTree tree = new RedBlackTree();
        tree.add(10);
        tree.add(20);
        tree.add(30);
        tree.add(15);
        tree.add(25);
```

```
        tree.add(5);



        System.out.println("Red-Black Tree:");

        tree.printTree();

    }

}
```