Zhilong Zhu: 001525601

Program Structures & Algorithms

Spring 2021

Mid-term Assignment

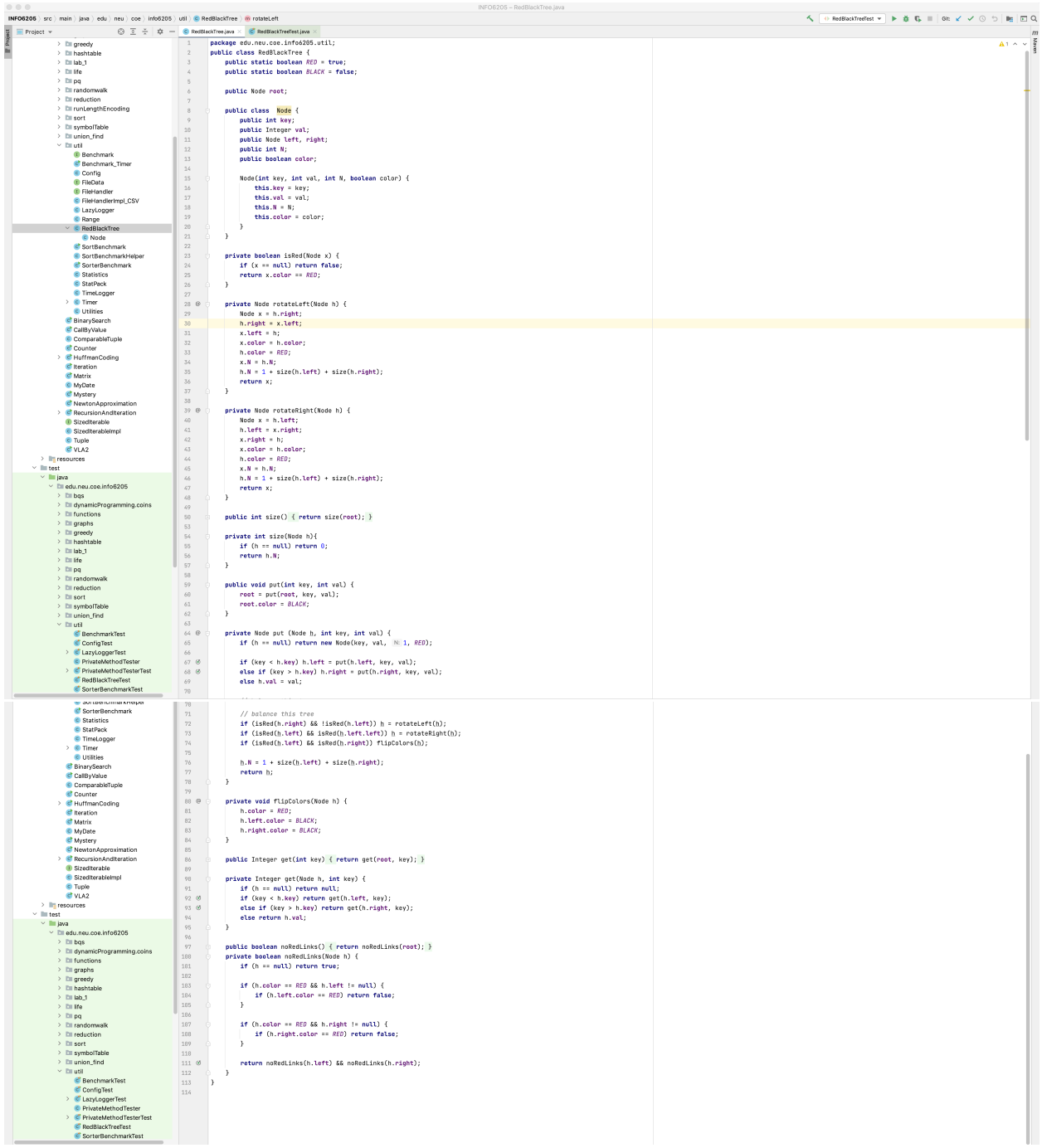
- ## Task

  You are to implement the red-black left-leaning tree as described in the slides. You will write suitable unit tests, for the successful and unsuccessful search, also the construction operations such as rotateLeft, rotateRight, flipColors. One additional test will add 255 keys *in order* (as in the demo) and you will check that there are no red links.

- # Implement

  I completed the red-black left-leaning tree by implementing construction operations such as rotateLeft, rotateRight and flipColors.

  In addition, I wrote the noRedLinks function to test whether the red-black tree contains red links.



```java
package edu.neu.coe.info6205.util;
public class RedBlackTree {
    public static boolean RED = true;
    public static boolean BLACK = false;

    public Node root;

    public class  Node {
        public int key;
        public Integer val;
        public Node left, right;
        public int N;
        public boolean color;

        Node(int key, int val, int N, boolean color) {
            this.key = key;
            this.val = val;
            this.N = N;
            this.color = color;
        }
    }

    private boolean isRed(Node x) {
        if (x == null) return false;
        return x.color == RED;
    }

    private Node rotateLeft(Node h) {
        Node x = h.right;
        h.right = x.left;
        x.left = h;
        x.color = h.color;
        h.color = RED;
        x.N = h.N;
        h.N = 1 + size(h.left) + size(h.right);
        return x;
    }

    private Node rotateRight(Node h) {
        Node x = h.left;
        h.left = x.right;
        x.right = h;
        x.color = h.color;
        h.color = RED;
        x.N = h.N;
        h.N = 1 + size(h.left) + size(h.right);
        return x;
    }

    public int size() { return size(root); }

    private int size(Node h){
        if (h == null) return 0;
        return h.N;
    }

    public void put(int key, int val) {
        root = put(root, key, val);
        root.color = BLACK;
    }

    private Node put (Node h, int key, int val) {
        if (h == null) return new Node(key, val, N: 1, RED);

        if (key < h.key) h.left = put(h.left, key, val);
        else if (key > h.key) h.right = put(h.right, key, val);
        else h.val = val;

        // balance this tree
        if (isRed(h.right) && !isRed(h.left)) h = rotateLeft(h);
        if (isRed(h.left) && isRed(h.left.left)) h = rotateRight(h);
        if (isRed(h.left) && isRed(h.right)) flipColors(h);

        h.N = 1 + size(h.left) + size(h.right);
        return h;
    }

    private void flipColors(Node h) {
        h.color = RED;
        h.left.color = BLACK;
        h.right.color = BLACK;
    }

    public Integer get(int key) { return get(root, key); }

    private Integer get(Node h, int key) {
        if (h == null) return null;
        if (key < h.key) return get(h.left, key);
        else if (key > h.key) return get(h.right, key);
        else return h.val;
    }

    public boolean noRedLinks() { return noRedLinks(root); }
    private boolean noRedLinks(Node h) {
        if (h == null) return true;

        if (h.color == RED && h.left != null) {
            if (h.left.color == RED) return false;
        }

        if (h.color == RED && h.right != null) {
            if (h.right.color == RED) return false;
        }

        return noRedLinks(h.left) && noRedLinks(h.right);
    }
}
```

- Test Code

1. testSize:
   This method tests the size () function. I created a new tree and inserted 1000 nodes. Finally, check whether the returned result is 1000.

```java
/**
*
* Method: size()
* this method also cloud test (Method: put)
*/
@Test
public void testSize() throws Exception {
//TODO: Test goes here...
    RedBlackTree rbt = new RedBlackTree();
    int n = 1000;
    for (int i = 0; i < n; i++) {
        rbt.put(i, i);
    }
    assertEquals( expected: 1000, rbt.size());
}
```

2. testGet
   This method tests the get () function. I created a new tree and inserted 100 nodes. Respectively checked whether 55 and 101 could be searched.

```java
/**
 *
 * Method: get(int key)
 * this method also cloud test (Method: put)
 */
@Test
public void testGet() throws Exception {
//TODO: Test goes here...
    RedBlackTree rbt = new RedBlackTree();
    int n = 100;
    for (int i = 0; i < n; i++) {
        rbt.put(i, i);
    }
    assertEquals((Integer)55, rbt.get(55)); // if the tree contains this key
    assertEquals( expected: null, rbt.get(101)); // if the tree does not contain this key

}
```

3. testRedLinks
   This test adds 255 keys *in order* (as in the demo) and checks that there are no red links.

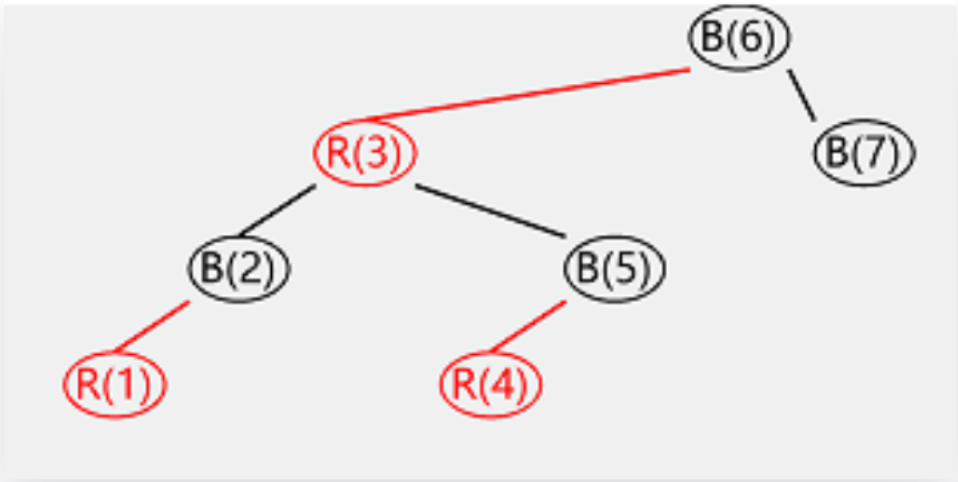
```java
/**
 *
 * Method: noRedLinks()
 * this method also cloud test (Method: put)
 */
    @Test
    public void testRedLinks() throws Exception {
        RedBlackTree rbt = new RedBlackTree();
        int n = 255;
        for (int i = 0; i < n; i++) {
            rbt.put(i, i);
        }
        assertEquals( expected: true, rbt.noRedLinks());
    }
```

4. testConsturctionFunctions

In this test, I will insert 7 numbers. Check whether the final result matches the result of a red-black left-leaning tree (including the color of the node and the position of the node). This test will check all construction operations.

```java
// Testing the construction functions of this class
@Test
    public void testConsturctionFunctions() throws Exception {
        RedBlackTree rbt = new RedBlackTree();
        rbt.put( key: 6, val: 6);
        rbt.put( key: 7, val: 7);
        rbt.put( key: 1, val: 1);
        rbt.put( key: 5, val: 5);
        rbt.put( key: 3, val: 3);
        rbt.put( key: 4, val: 4);
        rbt.put( key: 2, val: 2);

        // Compare the val and color of each node
        assertEquals((Integer)6, rbt.root.val);
        assertEquals( expected: false, rbt.root.color);

        assertEquals((Integer)3, rbt.root.left.val);
        assertEquals( expected: true, rbt.root.left.color);

        assertEquals((Integer)2, rbt.root.left.left.val);
        assertEquals( expected: false, rbt.root.left.left.color);

        assertEquals((Integer)1, rbt.root.left.left.left.val);
        assertEquals( expected: true, rbt.root.left.left.left.color);

        assertEquals((Integer)5, rbt.root.left.right.val);
        assertEquals( expected: false, rbt.root.left.right.color);

        assertEquals((Integer)4, rbt.root.left.right.left.val);
        assertEquals( expected: true, rbt.root.left.right.left.color);

        assertEquals((Integer)7, rbt.root.right.val);
        assertEquals( expected: false, rbt.root.right.color);
    }
```

This is the expected output.



- Unit tests result: