测试计划

第 25 组

2022 年 4 月 15 日

1 概述

1.1 测试需求

本次作业要求使用白盒测试。我们选择了一个红黑树的 java 实现作为测试对象，使用 Junit 框架进行单元测试。

1.2 任务分配

本小组为四人小组，任务分配为:

1.3 总体思路

由于被测软件有很多方法是 private 的方法，**为了使得测试方便，我们将所有方法全部改成 public**。原软件的所有 public 接口，写在 RBTree.java 顶部的注释中。

首先，colorOf 和 parentOf 等简单的 accessor 方法，以及 setColor 等简单的 set 方法，可以看做类似于宏，不需要进行单元测试，**只要其他方法的单元测试通过了，这些方法必然被覆盖到**。

因此，先测试各个查找方法。然后在测试修改树结构有关的方法时，首先需要测试左旋、右旋方法，然后测试重新平衡方法。最后再测试插入、删除方法。在本测试计划中，以下各个的单元测试是按照顺序进行的。

2 对 search 方法的测试

search 方法代码如下。在实际使用中，search 一定是从根节点开始查找，因此两个方法可以合并测试。

```
1  public RBTNode<T> search(RBTNode<T> x, T key) {
2      if (x==null)
3          return x;
4
5      int cmp = key.compareTo(x.key);
6      if (cmp < 0)
7          return search(x.left, key);
8      else if (cmp > 0)
9          return search(x.right, key);
10     else
11         return x;
12 }
13
14 public RBTNode<T> search(T key) {
15     return search(mRoot, key);
16 }
```

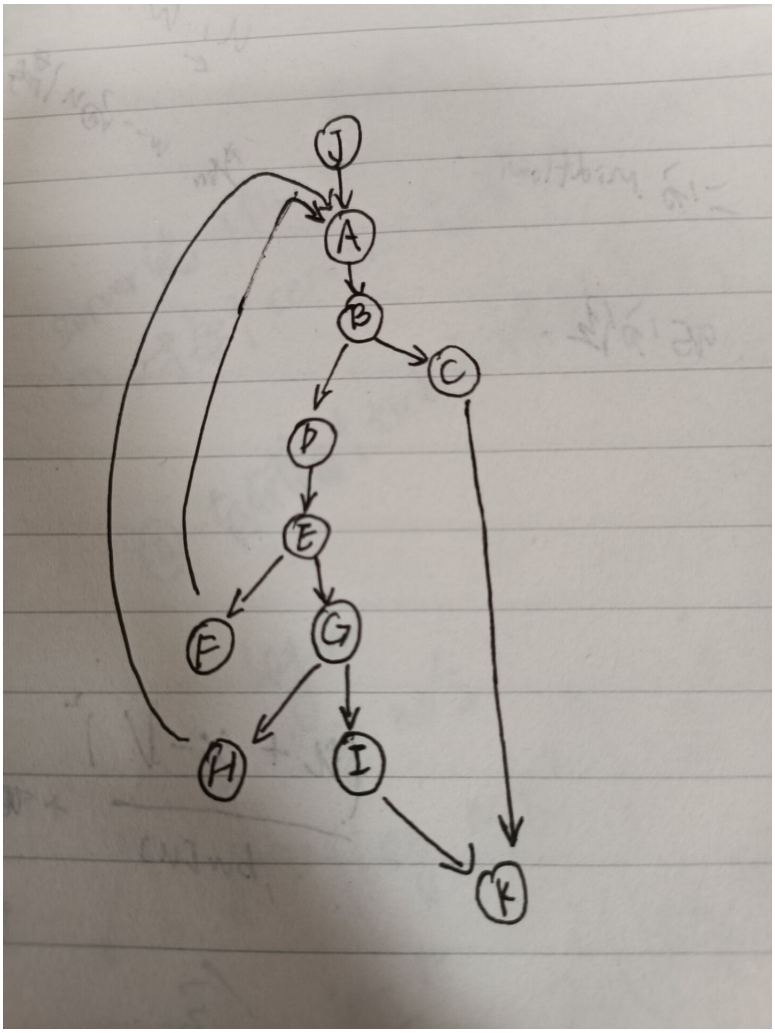
各个节点的定义如下

代码行	DD 路径名称
1	A
2	B
3	C
5	D
6	E
7	F
8	G
9	H
10-11	I
12	J
12-end	K

其中，12-end 描述的行为是在接收函数返回值并退出。

2.1 DD 路径分析和数据流分析

递归是一种特殊的循环。因此可以分析 DD 路径如下



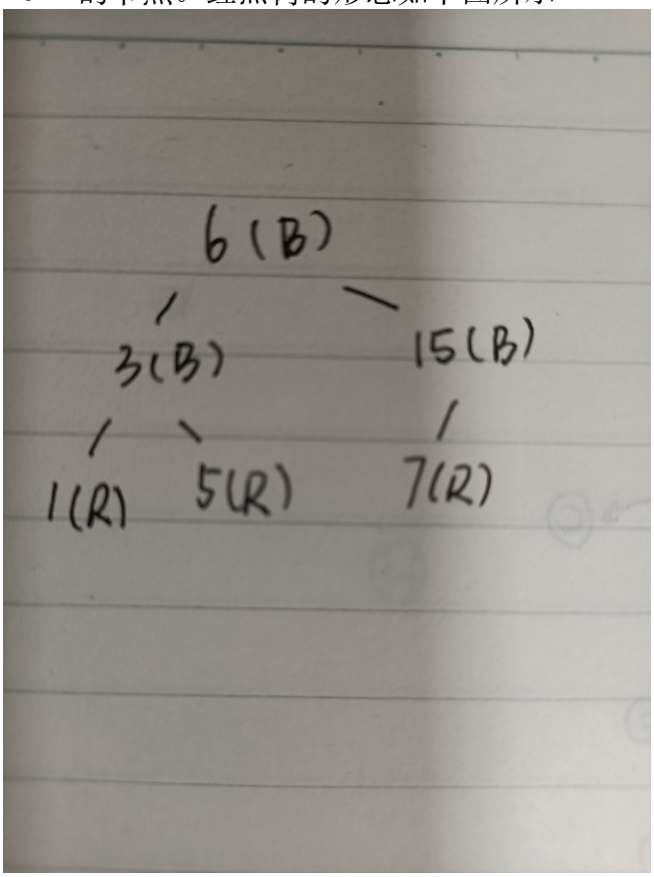
数据流分析如下

变量名	定义节点	使用节点
x	A	B C D F H I
key	A	F H
cmp	D	E G

在这个方法的单元测试中，使用数据流分析的结果设计用例，覆盖指标采用**全使用准则**。因此，测试用例的执行路径集合，需要覆盖以下路径: A-B, A-C, A-D, A-F, A-H, A-I, D-E, D-G.

2.2 用例设计

用例的设计代码如下。首先构造一个红黑树，然后依次查询权值为 5, 6, 114514 的节点。红黑树的形态如下图所示：




```
1 @Test
2 void search() {
3     tree = new RBTree<>();
4     tree.mRoot = new RBTNode<>(6, true,null,null,null);
5     RBTNode<Integer> node1 = new RBTNode<>(3, true, null, null,
          null);
6     RBTNode<Integer> node2 = new RBTNode<>(1, false, null, null,
          null);
7     RBTNode<Integer> node3 = new RBTNode<>(5, false, null, null,
          null);
8     RBTNode<Integer> node4 = new RBTNode<>(15, true, null, null,
```

```
        null);
9       RBTNode<Integer> node5 = new RBTNode<>(7, false, null, null,
            null);
10
11      tree.mRoot.left = node1; tree.mRoot.right = node4;
12      node1.parent = tree.mRoot; node1.left = node2; node1.right =
            node3;
13      node2.parent = node1;
14      node3.parent = node1;
15      node4.parent = tree.mRoot; node4.left = node5;
16      node5.parent = node4;
17
18      RBTNode<Integer> node = tree.search(6);
19      assertEquals(node, tree.mRoot);
20      node = tree.search(5);
21      assertEquals(node, node3);
22      node = tree.search(114514);
23      assertNull(node);
24  }
```