

白盒测试实验报告

赵钰-1712990

一、算法测试

a) 算法题目

给定一个二叉搜索树，编写一个函数 `kthSmallest` 来查找其中第 `k` 个最小的元素。

说明：

你可以假设 `k` 总是有效的， $1 \leq k \leq$ 二叉搜索树元素个数。

示例 1:

```
输入: root = [3,1,4,null,2], k = 1
      3
     / \
    1   4
     \
      2
输出: 1
```

- b) 算法思路: 模拟栈的方式, 使用二叉树非递归中序遍历的方法, 将 `k` 传入 `kthSmallest` 方法, 每次访问到需要加入结果的结点时 `k--`, 直到 `k=0` 时返回当前结点的值.

c) 测试代码

```
package com.test;
import java.util.Stack;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int x) {
        val = x;
    }
}

public class Solution {
    private enum Action {
        // GO 表示递归处理
        // ADDTORESULT 表示当前马上执行将结点的值添加到结果集中
        GO, ADDTORESULT
    }
}
```

```

private class Command {
    private Action action;
    private TreeNode node;

    public Command(Action action, TreeNode node) {
        this.action = action;
        this.node = node;
    }
}

public int kthSmallest(TreeNode root, int k) {
    Stack<Command> stack = new Stack<>();
    stack.add(new Command(Action.GO, root));
    int result=0;
    while (!stack.isEmpty()) {
        Command cur = stack.pop();
        TreeNode node = cur.node;
        System.out.println("STACKTOP: "+node.val);
        if (cur.action == Action.ADDTORESULT) {
            System.out.println("ADDTORESULT cur node: "+node.val);
            assert cur.action == Action.ADDTORESULT;
            k--;
            if (k == 0) {
                result = node.val;
                System.out.println("K==0: "+node.val);
                break;
            }
        }
        else {
            System.out.println("RECURSE cur node: "+node.val);
            assert cur.action == Action.GO;
            if (node.right != null) {
                System.out.println("RIGHT node add to stack: "+node.val);
                assert node.right != null;
                stack.add(new Command(Action.GO, node.right));
            }
            System.out.println("CUR node add to stack: "+node.val);
            stack.add(new Command(Action.ADDTORESULT, node));
            if (node.left != null) {
                System.out.println("LEFT node add to stack: "+node.val);
                assert node.left != null;
                stack.add(new Command(Action.GO, node.left));
            }
        }
    }
}

```

```

        return result;
    }

    public static void main(String[] args) {
        TreeNode[] tree;
        tree = new TreeNode[5];
        for(int i=0;i<5;i++){
            tree[i] = new TreeNode(i+1);
        }
        TreeNode root = tree[3];
        root.left=tree[1];
        root.right=tree[4];
        tree[1].left=tree[0];
        tree[1].right=tree[2];
        Solution solution = new Solution();
        int kthSmallest = solution.kthSmallest(root, 2);
        System.out.println(kthSmallest);
    }
}

```

d) JUnit 代码

```

package com.test;

import static org.junit.Assert.*;

public class SolutionTest {

    @org.junit.Before
    public void setUp() throws Exception {
    }

    @org.junit.After
    public void tearDown() throws Exception {
    }

    @org.junit.Test
    public void kthSmallest() {

    }

    @org.junit.Test
    public void main() {
        TreeNode[] test;
    }
}

```

```

test = new TreeNode[5];
for(int i=0;i<5;i++){
    test[i] = new TreeNode(i+1);
}

TreeNode root1 = test[3];
root1.left=test[1];
root1.right=test[4];
test[1].left=test[0];
test[1].right=test[2];

/*
      4
     / \
    2   5
   / \
  1   3

*/

Solution solution1 = new Solution();
int kthSmallest1 = solution1.kthSmallest(root1, 2);
assertEquals(kthSmallest1, 2);

for(int i=0;i<5;i++) {
    test[i] = new TreeNode(i + 1);
}
TreeNode root2 = test[4];
root2.left = test[3];
test[3].left = test[2];
test[2].left = test[1];
test[1].left = test[0];

Solution solution2 = new Solution();
int kthSmallest2 = solution2.kthSmallest(root2, 2);
assertEquals(kthSmallest2, 2);

for(int i=0;i<5;i++) {
    test[i] = new TreeNode(i + 1);
}
TreeNode root3= test[0];
root3.right = test[1];
test[1].right = test[2];
test[2].right = test[3];
test[3].right = test[4];

```

```
Solution solution3 = new Solution();
int kthSmallest3 = solution3.kthSmallest(root3, 2);
assertEquals(kthSmallest3, 2);

for(int i=0;i<5;i++) {
    test[i] = new TreeNode(i + 1);
}

TreeNode root4 = test[2];
root4.left = test[0];
test[0].right = test[1];

Solution solution4 = new Solution();
int kthSmallest4 = solution4.kthSmallest(root4, 2);
assertEquals(kthSmallest4, 2);

for(int i=0;i<5;i++) {
    test[i] = new TreeNode(i + 1);
}
TreeNode root5 = test[0];
root5.right = test[2];
test[2].left = test[1];
Solution solution5 = new Solution();
int kthSmallest5 = solution5.kthSmallest(root5, 2);
assertEquals(kthSmallest5, 2);

    }
}
```

二、测试结果

测试用例	BST 先序&中序遍历	K 值	栈顶元素	ADD TO RESULT	k==0	Node.right != null	Node.left != null
1	先序: 42134 中序: 12345	2	4	FALSE	-	TRUE	TRUE
			2	FALSE	-	TRUE	TRUE
			1	FALSE	-	FALSE	FALSE
			1	TRUE	FALSE	-	-
			2	TRUE	TRUE	-	-
2	先序: 54321 中序: 12345	2	5	FALSE	-	FALSE	TRUE
			4	FALSE	-	FALSE	TRUE
			3	FALSE	-	FALSE	TRUE
			2	FALSE	-	FALSE	TRUE
			1	FALSE	-	FALSE	FALSE
			1	TRUE	FALSE	-	-
			2	TRUE	TRUE	-	-
3	先序: 12345 中序: 12345	2	1	FALSE	-	TRUE	FALSE
			1	TRUE	FALSE	-	-
			2	FALSE	-	TRUE	FALSE
			2	TRUE	TRUE	-	-
4	先序: 312 中序: 123	2	3	FALSE	-	FALSE	TRUE
			1	FALSE	-	TRUE	FALSE
			1	TRUE	FALSE	-	-
			2	FALSE	-	FALSE	FALSE
			2	TRUE	TRUE		
5	先序: 132 中序: 123	2	1	TRUE	FALSE	-	-
			3	FALSE	-	FALSE	TRUE
			2	FALSE	-	FALSE	FALSE
			2	TRUE	TRUE		

a) 覆盖率

Coverage: Solution ×				⚙	—
Loading...					
Element	Class, %	Method, %	Line, %		
com.t...	100% (5/5)	100% (10/10)	100% (96/96)		

b) 测试时间: 2019.12.10. 15:20

Class transformation time: 0.02038029s for 475 classes or
4.2905873684210526E-5s per class

c) 是否发现缺陷: 否

```
1  package com.test;
2  import java.util.Stack;
3
4  class TreeNode {
5      int val;
6      TreeNode left;
7      TreeNode right;
8
9      TreeNode(int x) {
10         val = x;
11     }
12 }
13 public class Solution {
14     private enum Action {
15         // GO 表示递归处理
16         // ADDTORESULT 表示当前马上执行将结点的值添加到结果集中
17         GO, ADDTORESULT
18     }
19     private class Command {
20         private Action action;
21         private TreeNode node;
22
23         public Command(Action action, TreeNode node) {
24             this.action = action;
25             this.node = node;
26         }
27     }
28
29     public int kthSmallest(TreeNode root, int k) {
30         Stack<Command> stack = new Stack<>();
31         stack.add(new Command(Action.GO, root));
32         int result=0;
33         while (!stack.isEmpty()) {
34             Command cur = stack.pop();
35             TreeNode node = cur.node;
36             if (cur.action == Action.ADDTORESULT) {
37                 assert cur.action == Action.ADDTORESULT;
38                 k--;
39                 if (k == 0) {
40                     result = node.val;
41                     break;
42                 }
43             }
44             else {
45                 assert cur.action == Action.GO;
46                 if (node.right != null) {
47                     assert node.right != null;
48                     stack.add(new Command(Action.GO, node.right));
49                 }
50                 stack.add(new Command(Action.ADDTORESULT, node));
51                 if (node.left != null) {
52                     assert node.left != null;
53                     stack.add(new Command(Action.GO, node.left));
54                 }
55             }
56         }
57         return result;
58     }
59 }
```

```
58  ▶ public static void main(String[] args) {
59      TreeNode[] tree;
60      tree = new TreeNode[5];
61      for(int i=0;i<5;i++){
62          tree[i] = new TreeNode(x: i+1);
63      }
64      TreeNode root = tree[3];
65      root.left=tree[1];
66      root.right=tree[4];
67      tree[1].left=tree[0];
68      tree[1].right=tree[2];
69      Solution solution = new Solution();
70      int kthSmallest = solution.kthSmallest(root, k: 2);
71      System.out.println(kthSmallest);
72  }
73  }
74
```