Report of the A3 of CS562

Student: Yifan Shen

The procedure of the project:

From current testing work, the insertion of number, and the character of color do not have bugs. However, there is some bad design of function. It is not the bug, but it can not show the result directly and I used to think that is a bug. For example, the inorder walk and reverse inorder walk. It could not output the number when it operate the walk.

Expain what I did in the test code:

To test the red black tree library, we must write some code to test its feature. Red black tree is similar to AVL tree to a certain extent, because the red node could not have the red children nodes, the height of different branches of the tree will be similar. In the first version of tstl testing code, I create four different int number and a red black tree. To test the basic characteristic of the tree, I design some actions and test. Because the red black tree is a tree, we can insert number to the tree, and the tree can help us find the correct position to insert the number. At the same time, each node inserted into the tree will have its own color and the color is either black or red. To this feature, we can assert that the height of black nodes are smaller than the size of tree. To make the tstl code abort at some degree, I add a restrained requirement: the size of the tree will not be larger than 15. The color is the most important feature of red black tree, so in the second version of tstl testing code, I add some functions to test the color feature of the tree. Because the black red tree has a feature that the each route to the leave nodes will have the same number of black nodes, I write a code to test the number of black nodes on each route. I use the recursion to get all the leave nodes and put the number of black nodes along the route to a array. Then I test whether the sorted array is equal to the reverse sorted array. If they are the same, the black nodes along each route is the same and the character of red black tree is correct.

The future work can be done with tstl:

I think there are still some improvements that can be done by the end of the term. For example, test the children color to check whether there are two neighboring nodes have the red color which is against the red black tree law. And I think I can add more actions such as delete, search and so on.

Discussion about sut:

System under test is the final operation in the software development, it will test the whole functions in the system. So from my point of view, it will help us find more problems in the software. Maybe we can correctly deal with each single function, but when we combine these single functions together, there are usually some errors. And this is the sut works for. From the current work, I think the sut is a very good testing system, because it can implement what I want to test and randomly generate the test data which is very useful to test whether the library has considered all the input situation. And I think, by using sut system, we can help a big program to debug and it can widely used in python library.

Some contents about coverage:
To the code coverage, I still need learn more to use that. The current work will shows that there are no data.

The reference codes:

```
@import red

<@
#define a function to test number of black nodes along each branch.

def colorOk(self,node,value):

#result array is to save all the numbers of black nodes of each branch
    result = [ ]

#define a recursive function to calculate the number of black nodes.

    def dfs(self, node, value):
        if node == None :
            return
        if node.color == False:
            value +=1
        if node.left==None and node.right==None:
            result.append(value)
            return
        dfs(self,node.left, value)
        dfs(self,node.right, value)
    dfs(self,node,value)
    print(value)

#judge whether all the number of black nodes are the same

    if result.sort() ==result.sort(reverse =True):
        return True
    else:
        return False
```

```
@>


pool: <INT> 4
pool: <TREE> 1

property: <TREE>.size >= <TREE>.black_height()
property: <TREE>.size < 15
property: colorOk(self,node =<TREE>.root,value=0)

<TREE>:=red.RedBlackTree( )
<INT>:=<[1..20]>
<TREE>.add(<INT>) =>  ((<TREE,1>.size) == PRE<(<TREE,1>.size)>+1) ; print
<TREE>
```