**Final report of TSTL**

**Introduction**

The libray I have choosed called Red-Black tree, this is a tree similarity data structure and it have a lot of poperties such as I list follow.

1. first of all, the red black tree have to be a binary search tree as basic and then in this data structure each nodes is either red or black.

2. The root is black. This rule is sometimes omitted. Since the root can always be changed from red to black, but not necessarily vice versa, this rule has little effect on analysis.

3. All leaves (NIL) are black.

4. If a node is red, then both its children are black.

5. Every path from a given node to any of its descendant NIL nodes contains the same number of black nodes.

6. the number of black nodes from the root to a node is the node's black depth; the uniform number of black nodes in all paths from root to the leaves is called the black-height of the red–black tree.

These constraints enforce a critical property of red–black trees: the path from the root to the farthest leaf is no more than twice as long as the path from the root to the nearest leaf. The result is that the tree is roughly height-balanced. Since operations such as inserting, deleting, and finding values require worst-case time proportional to the height of the tree, this theoretical upper bound on the height allows red–black trees to be efficient in the worst case, unlike ordinary [binary search trees](https://en.wikipedia.org/wiki/Binary_search_tree).

**My Accomplishment**

The library I have been tested and after the TSTL random testing, I didn't find any big problems, means no bug has founded right now. I have test few functions in this library, which contains the red-black tree's initialization, insert, delete, rotate and search. Now I am about to introduce the method that I used in testing all these functions.

In this specific case, for the first edition, we need to create a new tree for us in order to proceed the rest of the operations, so we call redblack.RedBlackTree() to do initialization and we got a tree structure, then we let the random tester continue to insert random integers into the tree structure. Besides, we also need to make sure whether the operations success or not, in this case, we can first define whether the inserted number is exist or not by using .search function, if the integer that we just insert is in tree structure, the insert function is successful run, in verse, we can also using the similar way to test delete function, when we delete one node from red-black tree( or a subtree of the whole tree), we could then detect the number of remain nodes by using .size() if the missing size of the tree equal to the size before it deleted, Or delete a element that actually do not exist from a empty tree, we will see if there shown some of error that indicate we cannot eliminate the element anymore. In this step, we can move on to the others functions, for example, the minimum and the maximun functions.  
So, the question that mentioned in first edition has solved by few days ago, in this time I try to using some customizd functions that I write at the front of TSTL to include more than one specific testing functions in library classes. This time I test the minmum and maximun and the relationship between the left-most child. And the traversal functions actually already include a lot of operations during the test. In addition, I also wrote two self-defined function called “leafcolor” and “dfs”, the “leafcolor” is to test if the new insert node have correct color which is critical to determine whether the property of this tree is fullfill the criteria of RedBlackTree. Because I found that when I add a value to a tree it will show things correctly, but if I define a node and assign it a specific value, then I insert this node into the tree, the color will remain as a black node, and there is no way to change the color back to right way. The function dfs() is mainly test if the depth first search can be accomplished, the reason I use this in the tester is this function contains a lot of original functions, so that I could test all of them through one call, it manually reduce the burden of running the random tester.

**Performance of the tester**

The random tester can randomly generate the metadata for the libray as inputs, so that we could save our time and test the program's robust property in chaos environment. Basically, the TSTL runs well, it correctly generate the data that fullfill the criteria, besides, not only the data, the process sequences can also be random, but in this circumstance, The random tester is not perfect.

The operations that wait for test is randomly picked, but sometimes the tester will choose the operation no follow the resonable order, for instance, the adding and operational step proceed forward to the initialization, the tester will detect the error and stop running, but the program may not have errors, sometimes it works when I using ~ in front of the RedBlackTree.add(), it prevent the actions process before there have no initialization.

**What is good about TSTL**

This random tester has its advantages, as I known, the random tester is actually not total random during the process, it basically have strategy on reduce branches, the idea of choose operation for the next step is like determine tree, this move dramatically improve the efficiency of the tester, we could find the error in our program accurate and quick. Furthermore, it is easy to obtain and seize the process of program, we ususlly just click few lines of command to make the output and see the coverage of the test result. Besides, the unique language especially for TSTL is very similar to python coding habit, but they are not same, the way it defines the variable and array make a lot of sense, the self-defined functions just directly using python and embed it into TSTL code, convinient and user-friendly.

**Suggestion for Future work**

For instance, I realize that the error reflect machnism is not very convinient, if compile the TSTL file and have some alert about syntax error or data overflow, I may see the error directly in my tstl code, otherwise it will get the user confuse, because everytime when I try to find the location of the error, it will just show the error on File "/usr/local/bin/tstl" or File "/usr/local/bin/tstl", which is not easy for the user to understand where the mistake was happened, what if the output also can recognize the TSTL syntax error in file .tstl, that will be more user-friendly for this product.

**Coverage summery**

We consider this part is better than what we expected, I read the paper and find out the method of TSTL choose to test the target program is advance and powerful, it will go through the path in the program as much as possible and try to avoided the inevitable repeat of the progress.

At the end, the coverage of this new test file is 36.7% which is not make people satisfy with, 117 branches and 87 statement are covered. This low level of coverage is due to the way that I test has not go through as many as the library's capacity. As I mentioned from first two paragraphs, There only seven basic operations we have been done with it, there are still have inorder\_walk() and reverse, which may call the basic operations and cover more branches in this library, essentially, the revse() will adjust the position of the nodes in order to make it fullfill the criteria of the red-black tree, the root have to be color black and the leaf node also, the red color node cannot be cosequence in ray. So I planning to make more effort on this direction and test two big call in order to increase the coverage. the .out file is also uploaded to the directory.

