thakuro@oregonstate.edu

CS 562 – Applied Software Engineering – Part 4

Omkar Thakur – 932704163

Prof. Alex Groce

My library: PyBst

Pythoncode :

1.bst.py (binary search tree)

2. splaytree.py (splay tree)

3. avltree.py (avl tree)

4. rbtree.py (red black tree)

Summary:

The main aim of doing this project was to find bugs using TSTL (Template Scripting Testing Language). I have tested tree functions in this library using TSTL tools. I am currently a student of Computer Science and have profound interest in tree structures. That is the main reason I have chosen this library. I was always mesmerized by the data storing techniques in various databases.

PyBst is a library which contains splaytree, rbtree, avltree, and bstree. With this library, red black tree, splay tree, avl tree, rbtree and binary search tree can be created. Various constructors are available consisting of initializing the tree from start and in between are available. PyBst contains various functions for creating, inserting, deleting a particular node or tree. In the latter part of this document, the advantages, disadvantages, quality of the SUT and possible improvements in the future are also discussed.

Operating system used: Ubuntu

Functions tested:

Constructors:

1.BSTree()

2.AVLTree()

3.SplayTree()

4.RBTree()

thakuro@oregonstate.edu

Methods:

1.insert()

2.insert_from()

3.delete()

4.delete_from()

5.is_valid()

6.delete_node()

7.insert_node()

8.inorder()

9.preorder()

10.postorder()

11.levelorder()

12.deletecase()

13.insertcase()

14.switch_node()

15._insert_case_one()

16._insert_case_two()

17._insert_case_three()

18._insert_case_four()

19._insert_case_five()

20._insert_case_six()

21._delete_case_one()

22._delete_case_two()

23._delete_case_three()

24._delete_case_four()

25._delete_case_five()

26._delete_case_six()

27._delete_leaf()

28._delete_leaf_parent()

thakuro@oregonstate.edu

Code Coverage:

Since each python file was approximately 450 lines, therefore in total it was 1600 lines. My code coverage is 48% for bst.py (binary search tree), 27 % for rbtree.py (Red Black Tree), 43% for splaytree.py (Splaytree), 70% for avltree.py (AVL Tree). The time limitation was a major factor in restriction of code coverage

AVL Tree:

```
70.1058201058 PERCENT COVERED
51.983299017 TOTAL RUNTIME
211 EXECUTED
21094 TOTAL TEST OPERATIONS
22.7513079643 TIME SPENT EXECUTING TEST OPERATIONS
0.217758893967 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.0105638504028 TIME SPENT CHECKING PROPERTIES
22.7618718147 TOTAL TIME SPENT RUNNING SUT
0.0138401985168 TIME SPENT RESTARTING
28.7894330025 TIME SPENT REDUCING TEST CASES
238 BRANCHES COVERED
178 STATEMENTS COVERED
```

Binary Search Tree:

```
47.6454293629 PERCENT COVERED
13.9420440197 TOTAL RUNTIME
664 EXECUTED
66313 TOTAL TEST OPERATIONS
12.886370182 TIME SPENT EXECUTING TEST OPERATIONS
0.70184135437 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.0344252586365 TIME SPENT CHECKING PROPERTIES
12.9207954407 TOTAL TIME SPENT RUNNING SUT
0.0333666801453 TIME SPENT RESTARTING
0.0677318572998 TIME SPENT REDUCING TEST CASES
151 BRANCHES COVERED
113 STATEMENTS COVERED
```

Splay Tree:

```
43.0868167203 PERCENT COVERED
16.9696900845 TOTAL RUNTIME
882 EXECUTED
88116 TOTAL TEST OPERATIONS
15.5937540531 TIME SPENT EXECUTING TEST OPERATIONS
0.931241989136 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.0432858467102 TIME SPENT CHECKING PROPERTIES
15.6370398998 TOTAL TIME SPENT RUNNING SUT
0.045567035675 TIME SPENT RESTARTING
0.072958946228 TIME SPENT REDUCING TEST CASES
115 BRANCHES COVERED
87 STATEMENTS COVERED
```

Red Black Tree:

```
26.8707482993 PERCENT COVERED
24.1969361305 TOTAL RUNTIME
1171 EXECUTED
117011 TOTAL TEST OPERATIONS
22.3297626972 TIME SPENT EXECUTING TEST OPERATIONS
1.28979802132 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.0578722953796 TIME SPENT CHECKING PROPERTIES
22.3876349926 TOTAL TIME SPENT RUNNING SUT
0.0678660869598 TIME SPENT RESTARTING
0.0704369544983 TIME SPENT REDUCING TEST CASES
141 BRANCHES COVERED
109 STATEMENTS COVERED
```

Did you find bugs?

No, I was not able to find any bugs. But my test cases were running approximately 900000 testcases in a frame of 12 seconds on average. This library is very well written.

Quality of SUT

The quality of the sut is very well written. This is because I ran approximate 900000 test operations in span of 12- 15 seconds and was not able to find any errors. But I do think that certain non-compatibility of functions exists. The four files combined is approximate 2000 lines of code. The library is also very well tested. But still this library is not well documented.

How well does the tester work?

The tester works good overall. The location of most of the bugs is found by the TSTL. The tester is well written. Sometimes it gives indentation error for a line when there is a colon missing in the previous line. The keyboard interrupt works well. But it would be an improvement if it worked in a single press, since three to four presses are required. The good thing is that the tests are run in a good frame of speed. I was using i7 4790k processor for these tests. Displaying error on perfect line everytime, is a difficult task. But if it is accomplished, TSTL can be popular among popular testing tools. But is very well written and powerful tester.

What is wrong with TSTL?

It was tough to understand in the beginning. This can be resolved if documentation about TSTL is available. Testing a specific functions which require *args can be difficult with TSTL. While most of the functions can be tested easily using TSTL. It displays indententation error for a missing semi colon. The key board interrupts should work with a single press. It requires many key presses in order to stop the working. The previous version of TSTL had a coverage problem. I used to run my tests with a –nocover configuration. The current version is better. It's difficult to give a specific input of variables as opposed to random input of variables. I found python pretty uninteresting as compared to C and java. If TSTL supported such languages, then making test cases can be more interesting

thakuro@oregonstate.edu

What is good about TSTL?

It was easier to define a majority of functions for testing, excluding a few functions with *args. The testing is done in a good speed. It's easier to check your functions since, TSTL assigns random input. It also assigns a step number. Due to which it's easy to track bugs. But we have seen its capability to detect bugs in big libraries, which is great.

Accomplishment:

Due to CS 562 course, I was able to learn python and different software testing techniques. I got accustomed to python over time. I would personally like to thank Dr. Alex Groce for teaching this class. Due to this class, I came to know some of the fundamentals in software testing. I am planning to put this project in my resume.

References:

http://www.cs.cmu.edu/~agroce/issta15.pdf

https://pypi.python.org/pypi/pybst/1.0