Final Report

Class: cs562

Name: Xiaotao Yang

Student ID: 932-237-336

About the library:

In this project, I test a tree data structure library by using tstl. The python library is called pyTree. This library is developed by yoyzhou. The source code can be found from: https://github.com/yoyzhou/pyTree.

1: What did you accomplish?

For this project, I tested all the function provided by the library. I will briefly talk about those functions and how I test them.

(1): addChild() : add a child to this tree.

In order to test this function and the future work, I create a helper function called test_helper1(self, child). This helper function will check the self's data first, if it doesn't contain a data, then return false. Then it will check the data of child's parent. If self's data equal to data of child's parent, then return true, else return false. Then back to test the addChild() function. If addChild() function success, then the new added child's parent must be equal to the node where you use the addChild() function. By using test_helper1, I can check the true or false of it.

(2): addChildren():add many children to this tree.

This function is just like addChild(), but the argument of it is a list of trees, it will add multiple child at the same time, so for each new child added, I will use the test_helper1() to check true or false, and using 'and' connect them all. All the child test by test_helper1() function must be true, if one of them is false, then addChildren() function has bugs. In my test, it passed well.

(3) getParent() : get node's parent node.

I also use the test_helper1() function I created to test this function. Usually I use the test_helper1() function like: test_helper1(root, child01) to test if I success add the new child of root, now I use it like: test_helper1(child01.getParent(), child01) to test if I success get the parent() of child01.

(4) getChild() : get node's child node

This one is just like getParent() function. I use the test_helper1() function like: test_helper1(root, root.getChild(index)) to check if I success get the child of root.

(5): getChildren() : get node's all children nodes.

This one tested like getChild(), the difference is I will check all the child I get. And use 'and' connected them all. It is like: (test_helper1(root, root.getChildren()[0]) == Ture) and (test_helper1(root, root.getChildren()[1]) == Ture).

(6): getNode() : get the node.

For this function, I create a particular tree like:

Root

|__C01

| |_C11

|__C02

So I just check whether getNode('C01') equal to root's first child. If equal, then I get the right node and if false, getNode() must has bugs.

(7): delChild(): delete node's child.

By test this function, I create a new function called test_helper2(Node), it is a simple function check that whether the Node contains a value. If it has a value, then true, else false. So After I delete a Child, I will use this test_helper2(Child) to check if I really delete the child.

(8): delNode(): delete node.

It tested just like delChild(). After delete a Node, then use test_helper2(Node).

(9): getRoot() : get the tree root.

For this function, I create a particular tree like I tested getNode(), I test it like: getRoot() == child01.getParent(). Child01 is root's child.

(10): isRoot() : determine whether node is a root or not.

This one is a easy test, I tested it like: (<child>.isRoot == Flase) and (<root>.isRoot == True).

(11): isBranch(): determine whether node is a branch node or not.

This one is a easy test, I tested it like: (<child>. isBranch == Ture) and (<root>. isBranch == False).

The test result is pretty good, this library passed all the test I created.

2: Did you find bugs?

The bug I found is:

“ERROR: (<type 'exceptions.TypeError'>, TypeError('super() takes at least 1 argument (0 given)',), <traceback object at 0x1014e7908>)”

The original code is:

“super().__setattr__(name, value)”

As the error message says, I can’t use “super()” function like this. The “super()” function need 2 arguments. The name of the child class which is calling super()(Tree is this case) is the first argument to the method, and a reference to the object calling super() is the second argument.

The corrected code is:

“super(Tree, self).__setattr__(name, value)”

Install the TSTL, try the examples in the tstl folder.

3: How well does the tester work?

I think the tester does a very good work. Random-tester did a very good job to find the bugs and analysis them. It not just give us the location of the bugs, but also give us how they did wrong about it. By using this, the tester work becomes easier than before, all I need to do is write a simple test code and then waiting for result. The random part is amazing, it can create and execute random test case with random values I provided. And I think this is the most important and most useful part of the tester. It will help us to find some bugs I can’t find that easily. Overall, it is a very good and powerful tester.

4: What is wrong with TSTL?

TSTL is a good test program, or a language. But it still has some cons. The first point is it lacks a systematized tutorial. It is hard to understand at first and hard to understand how to program at first. The reason is not it is a hard language or program. Instead, it is very simple and basic. The reason is it’s hard to get access in this language. All I have is just some examples provided by professor. We need some tutorial for it. Another weak point is it

just supported by python. It's good, python is a good language and it is widely used, but we need it supported by more language like c, java, or something else. Consider the TSTL is still at a low version, I think TSTL doing a very good job, but in the future, add the support by new language will be better.

5: What is good about TSTL?

It is a very concise and powerful language or program for test. I think it provide a simpler way to test, it gives us a automate tester. Also, the random part is amazing, it can create and execute random test case with random values I provided. It will help us to find some bugs I can't find that easily. Another part is TSTL is very simple and easy to understand if it provides a good tutorial. By using this, doing more work by less time.

6: Coverage summaries, any other important data on the test effort.

It is strange that the coverage is just about 10% after I tested all the function provided in the library. I still confused how that coverage calculated and implemented. The result shows as blow:

```
Coverage.py warning: No data was collected.
Coverage.py warning: No data was collected.
Coverage.py warning: No data was collected.
Coverage.py warning: No data was collected.
Coverage.py warning: No data was collected.
STOPPING TEST DUE TO TIMEOUT, TERMINATED AT LENGTH 50
STOPPING TESTING DUE TO TIMEOUT
9.85915492958 PERCENT COVERED
5.0203909874 TOTAL RUNTIME
248 EXECUTED
24750 TOTAL TEST OPERATIONS
4.32059264183 TIME SPENT EXECUTING TEST OPERATIONS
0.431261777878 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.0174021720886 TIME SPENT CHECKING PROPERTIES
4.33799481392 TOTAL TIME SPENT RUNNING SUT
0.134613513947 TIME SPENT RESTARTING
0.0 TIME SPENT REDUCING TEST CASES
19 BRANCHES COVERED
14 STATEMENTS COVERED
```