Shivani Wanjara
CS562

Applied Testing Software Engineering Project: Part 3

Library :
https://github.com/nryoung/algorithms/tree/master/algorithms/data_structures

Using TSTSL we write test cases for the given library of program. I have used the library as mentioned above, where I have used the data structures section for testing. In which so far I have completed running test on few data structures

- Binary Tree
- Stack
- Queue
- Diagraph
- Undirected Graph
- Singly Linked List

Where I have concentrated on the basic functionality of the codes. As provided in the previous reports are the test cases that test the basic functionality of the code and also checks if the program works with random input.

Tester works:

Using Random tester as the required input for my test written in TSTL. The Random tester file provides inputs randomly to my test cases. The sut.py is generated after I run my tstl command on my tstl file. After which I run the randomtester.py next for the assignment on input to the test case. However as we know when we use random tester will assign the values randomly at the same time randomly run the test coded as well.

Bugs Found:

While testing the code, I did not find any bugs as such. The functionality of the code so far seems correct and appropriate for the functions coded. After using TSTL to generate different kinds of inputs fir the test cases that are designed. After distinctly checking the logs and results, no bugs were found in the library that I was testing. I also think that SUT is of a good average quality because I have tested all the basic functions of multiple structures. I think however if tested

rigorously may be there would be some bugs possibly. The programmer who wrote the library seems to be very accustomed with the concepts as there was no bugs found from my end.

Good with TSTL:

TSTL I think is really time saving when it comes to using random tester values for a given test case. It saves the effort for generating random data manually. This definitely helps the tester.

Bad with TSTL:

When using random data , it was easy with int values , there is no control if you want a ascending or descending number sequence. More over it's an issue when you want to use string or ordered string values for your test case.

Coverage :

In the initial stages I had compatibly issues with the version of coverage and TSTL and hence my test cases would not run my code unless I use --no cover. Later after the TSTL Update I was able to run it without it as well. However occasionally I would get coverage warnings like:
"Coverage .py warning: no data was collected". And having the coverage as 0.
I did have tough time with coverage. But eventually was able to find the correct coverage percentage for all of the following cases .

Binary tree:

Where each node has a key and the value of each key should be larger than the keys on the left side of the node and should be smaller than the keys on the right side of the node. This is tested, by assigning the key and value to the tree. I have tested put, contains, is_empty, delete, max and min functions of the binary search tree. The few cases I considered are as follows: Random tester assigns key and value to the tree, next it is checked if they contain the values which it would and hence not be empty, hence the condition would be not empty which is true, this is tested in case1.I have also tried to find the max and min node of the tree, using the key value of the Tree. Also deletion of the key once it has been inserted, to check if the condition is false or true. Testing of deletion of the key value in the tree is the deletion itself is successful or not.

Code Coverage :

Test.tstl :

44.578313253 PERCENT COVERED
108 BRANCHES COVERED
74 STATEMENTS COVERED

Name Stmts Miss Branch BrPart Cover Missing
---
binary_search_tree.py 169 95 80 7 45% 14-19, 27-32, 35, 41, 51, 61, 63, 72, 82, 92, 111, 121, 128, 135, 145, 149, 156, 163, 173, 177-216, 231-304, 306, 325, 335, 343, 354-395, 62->63, 127->128, 144->145, 155->156, 172->173, 224->231, 305->306

Coverage.out

name Stmts Miss Branch BrPart Cover Missing
---
binary_search_tree.py 169 96 80 7 44% 14-19, 27-32, 35, 41, 51, 61, 63, 72, 82, 92, 111, 121, 128, 135, 145, 149, 156, 163, 173, 177-216, 231-304, 306, 325, 335, 343-395, 62->63, 127->128, 144->145, 155->156, 172->173, 224->231, 305->306

Test1 .tstl :

23.2492489815 TOTAL RUNTIME
21 BRANCHES COVERED
15 STATEMENTS COVERED

Coverage.out

Name Stmts Miss Branch BrPart Cover Missing
---
binary_search_tree.py 169 154 80 5 8% 14-19, 27-32, 35-92, 99-111, 121-304, 306, 308, 310, 315-325, 335-395, 95->99, 305->306, 307->308, 309->310, 313->315

Stack:

Similarly for stack testing I have added new values for the stack and check if the size of the stack has been increased or no after successful insertion of value in stack. The test will also increase the value of the size of stack. Function usability in Stack tested are stack.add() , stack.remove . stack.isempty() , stack.size() . These functions work as expected and no bugs were found in this

Code Coverage :
Test2.TSTL
25.0 PERCENT COVERED
6 BRANCHES COVERED
3 STATEMENTS COVERED
Coverage.out

```
Name       Stmts   Miss Branch BrPart  Cover   Missing
-------------------------------------------------------
stack.py      12      9      0      0    25%   13-16, 19, 27, 36-50
```

Queue:
With the queue I have tested the basic functionality of it. Addition of an element in the queue and is empty and remove functions. . Function usability in Queue tested are add,remove ,is_empty . These functions work as expected and no bugs were found in this function.
Code Coverage :
Test4.tstl :
23.0769230769 PERCENT COVERED
6 BRANCHES COVERED
3 STATEMENTS COVERED
Coverage.out

```
Name       Stmts   Miss Branch BrPart  Cover   Missing
-------------------------------------------------------
queue.py      13     10      0      0    23%   14-20, 23, 31, 40-54
```

Diagraph : For Diagraph as well I have test the basic concepts or functionality of the graph with defined destination and source for the graph . I have checked the edges, counts and vertices of the graph.
Code Coverage:
33.3333333333 PERCENT COVERED
19 BRANCHES COVERED
14 STATEMENTS COVERED
Coverage.out :

```
Name         Stmts   Miss Branch BrPart  Cover   Missing
-------------------------------------------------------
```

diagraph.py 43 29 14 1 33% 10-11, 16-32, 52-59, 67-99, 64->67

Undirected Graph: Similar to the diagraph I have tested the undirected graph functions as well, the basic functions tested are addition of edge, edge count and the vertices of the graph as well.

Code Coverage:

34.7826086957 PERCENT COVERED
16 BRANCHES COVERED
12 STATEMENTS COVERED

Coverage.out :

Name Stmts Miss Branch BrPart Cover Missing
---
undirected.py 36 24 10 0 35% 10-11, 16-32, 51-85

Singly Linked List: For this library I have test the basic functions of the singly linked list like add, remove, search and size functions. Each of the test run and show the desired output for the same. So in one of the test cases linked list is not 0 hence will search for an node. If it is present them remove thee node and then check the size to the previous size ie size-1 . After which I add a node and check the current size with previous size increment.

Code Coverage :

50.9090909091 PERCENT COVERED
30 BRANCHES COVERED
22 STATEMENTS COVERED

Coverage.out

Name Stmts Miss Branch BrPart Cover Missing
---
singly_linked_list.py 45 23 10 0 51% 14-16, 20-26, 29-35, 39, 49, 73-93

Conclusion:

TSTL is a well coded language specially focused for testers. I think random tester makes the job really easy for testers specially in random allocation of inputs . I had difficulties with coverage but that was a really interesting aspect of TSTL. It

really helps to understand the system you would be testing. TSTL is definitely a tool to be used for faster testing.