CS562 – Applied Software Engineering.

Bhargav Pandya, 932690891.

Final Report

Introduction:

I am using TSTL to test four data structure libraries viz., binary_search_tree, queue, singly_linked_list and stack. I have tested each of these libraries thoroughly.

BST: The basic structure of the binary search tree in the system under test is that each node holds a key/value pair and nodes on the left side of the root are smaller than the root and the nodes on the right side of the root are bigger than the root. I have tested the functions of this library rigorously and they work and produce output as expected. These are not all but some functions of BST:

Size: Which returns the number of nodes/length in/of the BST.
Is_empty: Check if the tree is empty or not.
Get: Returns the value paired with a particular 'key'.
Contains: To check if the BST contain a particular 'key'.
Put: Insert a given key/value pair in the BST.
Min_key/Max_key: Return the minimum/maximum key present in the BST.
Delete: Remove a particular key/value pair based on a given 'key'.

Queue: The queue library is quite straightforward with four essential functions which deal with all cases viz., add, remove, is_empty, size.

Singly Linked List: This is a singly linked list library as the name suggests and has four functions; add, remove, search (for a particular value in the linked list), size.

Stack: The stack library is similar to queue and contains four functions: add, remove, is_empty, size.

Accomplishment:

I have tested each of the four libraries thoroughly, making sure that I test each function in multiple ways so that they might 'reveal' under what cases they break or produce error. Following are brief explanations of the tests I wrote for each library along with their ***Coverage Summaries***:

BST:

1. Insert a key/value pair in the BST, check if the insert worked and check if the BST is empty now or not.

   <bstree>.put(<key>,<value>) ; <bstree,1>.contains(<key,1>) => (<bstree,1>.is_empty() == False)

2. Check if the value returned by the ‘max_node’ function and ‘max_key’ function are the same. Tested ‘min_node’ and ‘min_key’ functions in the same way.

   <bstree>.put(<key>,<value>) => ((<bstree,1>._max_node().key) == (<bstree,1>.max_key()))

   <bstree>.put(<key>,<value>) ; minkey = <bstree,1>.min_key() ; minnode = <bstree,1>._min_node() => (minkey == minnode.key)

3. Insert a particular key/value pair and then delete that key/value pair from the BST and check if the deletion occurred successfully.

   <bstree>.put(<key>,<value>) ; <bstree,1>.contains(<key,1>) == True ; <bstree,1>.delete(<key,1>) => (<bstree,1>.contains(<key,1>) == False)

4. Delete the min and max key and checking if they still exist in the BST or not.

   <bstree>.put(<key>,<value>) ; min = <bstree,1>.min_key() ; x = <bstree,1>.delete_min() ; (min == x)

   <bstree>.put(<key>,<value>) ; max = <bstree,1>.max_key() ; y = <bstree,1>.delete_max() ;(max == y)

   46.9879518072 PERCENT COVERED
   2.30398511887 TOTAL RUNTIME
   36 EXECUTED
   3526 TOTAL TEST OPERATIONS
   1.52657151222 TIME SPENT EXECUTING TEST OPERATIONS
   0.0986979007721 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
   0.00423502922058 TIME SPENT CHECKING PROPERTIES
   1.53080654144 TOTAL TIME SPENT RUNNING SUT
   0.015417098999 TIME SPENT RESTARTING
   0.509826898575 TIME SPENT REDUCING TEST CASES
   113 BRANCHES COVERED
   78 STATEMENTS COVERED

Name	Stmts	Miss	Branch	BrPart	Cover	Missing
binary_search_tree.py	169	91	80	7	47%	14-19, 27-32, 35, 41-61, 63, 72, 82, 92, 111, 121, 128, 135, 145, 149, 156, 163, 173, 177-216, 231-304, 306, 325, 335, 343, 354, 362, 372-395, 62->63, 127->128, 144->145, 155->156, 172->173, 224->231, 305->306

Queue:

1. Check if the remove function works correctly.

   <queue>.add(<value>) -> not <queue>.is_empty() -> <queue,1>.remove()

2. Insert a single element in the queue and remove the element. Now check if the queue is empty and also its size is zero.

   <queue>.add(<value>) -> <queue,1>.remove() => (<queue>.is_empty() == True) => <queue,1>.size() == 0

3. Checking if adding an element increases the queue’s size by one.

   <queue>.add(<value>) => <queue,1>.size() == pre<(<queue,1>.size())>+1

4. Similarly check if adding and then removing an element from the queue does not affect the size.

<queue>.add(<value>) -> not <queue>.is_empty() -> <queue,1>.remove() => <queue,1>.size() == pre<(<queue,1>.size())>

15.3846153846 PERCENT COVERED
18.9970669746 TOTAL RUNTIME
451 EXECUTED
45037 TOTAL TEST OPERATIONS
12.8218586445 TIME SPENT EXECUTING TEST OPERATIONS
4.57999038696 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.0577335357666 TIME SPENT CHECKING PROPERTIES
12.8795921803 TOTAL TIME SPENT RUNNING SUT
0.13917016983 TIME SPENT RESTARTING
0.883938074112 TIME SPENT REDUCING TEST CASES
4 BRANCHES COVERED
2 STATEMENTS COVERED

Name	Stmts	Miss	Branch	BrPart	Cover	Missing
queue.py	13	11	0	0	15%	13-19, 22, 29-49

Singly Linked List:

1. Check if the list size is not zero. If it is not zero then search for a value, remove that value and check if the removal occurred successfully.

   (<llist>.size != 0) and <llist,1>.search(<val>) -> <llist,1>.remove(<val,1>) => (<llist,1>.size == pre<(<llist,1>.size)>-1)

2. Check if adding a new element increases the size of the list by one.

   <llist>.add(<val>) -> (<llist,1>.size == pre<(<llist,1>.size)>+1)

3. Do subsequent additions to the list and check if the size increases properly.

   <llist>.add(<val>) -> len = <llist,1>.size -> <llist,1>.add(<val>) -> new = <llist,1>.size -> (len == new-1)

4. Add an element to an empty list, check if size is not zero now, remove that element and now check if size is zero.

   <llist>.add(<val>) -> (<llist,1>.size != 0) => <llist,1>.remove(<val,1>) -> (<llist,1>.size == 0)

40.0 PERCENT COVERED
1.82265496254 TOTAL RUNTIME
29 EXECUTED
2818 TOTAL TEST OPERATIONS
0.847866535187 TIME SPENT EXECUTING TEST OPERATIONS
0.735840559006 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.00327849388123 TIME SPENT CHECKING PROPERTIES
0.851145029068 TOTAL TIME SPENT RUNNING SUT
0.00965452194214 TIME SPENT RESTARTING
0.121841907501 TIME SPENT REDUCING TEST CASES
23 BRANCHES COVERED
16 STATEMENTS COVERED

```
Name                 Stmts   Miss Branch BrPart  Cover   Missing
----------------------------------------------------------------------
singly_linked_list.py      45     29     10      0    40%   14-26, 29-35, 39-49, 73-93
```

Stack:

1. Add an element in an empty stack and check again if it is empty or not.

   <stack>.add(<value>) -> boole = <stack>.is_empty() -> (boole == True)

2. Add a single element in the stack. Remove it, now check if the length/size of the stack is zero.

   <stack>.add(<value>) -> <stack,1>.remove() -> list1 = <stack,1>.stack_list => (len(list1) == 0)

3. Check if the remove function works like the ‘pop’ function should work, i.e. removing the element which was added last.

   (not <stack>.is_empty()) -> list = <stack,1>.stack_list ; val = list[len(list)-1] ; popval = <stack,1>.remove() => val == popval

4. Check if after adding an element, the element is present in the stack and if it increases the stack’s size by one.

   <stack>.add(<value>) => (<value,1> in <stack,1>.stack_list) and (<stack,1>.size() == (pre<(<stack,1>.size())>+1))

```
25.0 PERCENT COVERED
2.34182405472 TOTAL RUNTIME
64 EXECUTED
6317 TOTAL TEST OPERATIONS
1.65406775475 TIME SPENT EXECUTING TEST OPERATIONS
0.422525882721 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
0.00741958618164 TIME SPENT CHECKING PROPERTIES
1.66148734093 TOTAL TIME SPENT RUNNING SUT
0.0156817436218 TIME SPENT RESTARTING
0.179431915283 TIME SPENT REDUCING TEST CASES
6 BRANCHES COVERED
3 STATEMENTS COVERED

Name       Stmts   Miss Branch BrPart  Cover   Missing
-------------------------------------------------------
stack.py      12      9      0      0    25%   12-15, 18, 25, 33-45
```

Were there any bugs?

Until part 3 in the project I had only tested the BST library. I did not find any bugs in BST until that point. I expected to find bugs as I tested more functions in BST and also in the other libraries which I completed testing recently.

- I found a bug in the stack and queue library. The code does not handle removal of an element when the stack/queue is empty.

Apart from this, I didn't find any bugs in the libraries. Looking at the code it is clear that it is very well written and all possible 'special' or 'catch' cases are handled except the one I mentioned. I tried finding bugs in the libraries by testing each of the functions in multiple ways; testing boundary cases etc. I did not come across this bug during initial testing; at the face value the code seemed bug-free. However, when I wrote test cases for the individual functions in different ways I came across this bug.

How well does the tester work?

The tester worked very well for this SUT. The logging of the initialization of variables was quite helpful when looking at how the tests performed, what values were assigned to the variables; this was particularly helpful while finding the bug in the code because it clearly mentioned the reason why a particular test failed.
With help of the randomtester it becomes possible to test a certain library with large amounts of input values. Also, since it uses 'random' inputs it becomes possible to test the code with a wide variety of permutations which is very difficult to do while testing manually.

What is wrong with TSTL?

First thing I realised when I started using TSTL was that it can be only used to test python libraries. It would be good if it would be possible use TSTL for other languages like Java. Another place where I think TSTL could improve is in its documentation. Initially, I found myself stuck with the syntax. It would be great to have detailed explanation of TSTL in this area. Another place where I got confused was, even though I was writing test cases for all functions in the library, TSTL still didn't give me a hundred percent code coverage. For example, in my queue library, even though I have written test cases covering all the functions, it still gives me only about 15% coverage (same is the case for other libraries).

What is good about TSTL?

Even though I mentioned that there is very less documentation on the syntax, once I picked it up it was very easy to write test cases. I will say the learning curve to start testing with TSTL is very short. Once I picked up speed it was very easy and at the same time fun to write tests.

It is very efficient in catching bugs which would be quite difficult when performing manual testing. It is a great tool for teaching oneself automated testing. The error logging and coverage summaries are well detailed. It clearly explains where and why the code/test is wrong. Giving the user the control to define what range of inputs should be used is another good point about TSTL. Also, enabling the user to write helper functions in <@...@> just for testing purposes is very useful. Once I came up to speed with the syntax and proper understanding of TSTL, I realised that the number of ways to write tests in TSTL is indeed very large. It was possible to test each function in multiple ways very easily.

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