Prathveer Rai

Date: 3/14/2016

TSTL Final Project Report

Objective

To generate automatic test cases for data structures to test their adherence to properties and to check for performance between different data structures.

Test Description

Across, the length of the course, I have tested the Binary Search tree data structure and also the Linked List data structure. I have tested most of the modules present in the Binary Search tree. Subsequently, I have tested all the modules in the Linked List source. Additionally, I have also tested the performance of the two sources based on insertion and deletion of the keys from the data structures and produced the average time taken from both these sources. Both these data structure sources have been used from the Github Library and are referenced at the end of the document.

Test Case Description

Binary Search Tree

Quality of System Under Test

The Binary Search Tree script does not report any bugs after rigorous testing. Compared to the beginning, the test scenario has been improved by generating more test cases for more modules in the system.

Previous completed tasks

Test cases were only focused on simple conditions of insertion and deletion of values in to and from the tree

Current completed tasks

Modules tested in the Binary Search Tree:

- put module - This module enters key value data in to the tree.
- delete module – This module deletes a value that exists in the tree.
- max_key module – This module returns the maximum key that exists in the tree.
- min_key module – This module returns the minimum most key that exists in the tree.
- contains – This module checks if key element exists in tree.
- Is_empty – This module checks if the tree is empty and returns conditional value.
- keys module – This module returns all the keys that exist in the tree.

In the Binary Search Tree source file I have tested the following:-

- I have checked if the key value exists after the element has inserted into the list and also verified that the size of the tree increases by 1 after insertion of an element that does not exist in the tree.

- Also the property of the Tree is checked such that every node to the left of the root is smaller than the root and the node to the right of the root is larger than the root.

- I have checked if the element exists after the key is deleted. If the respective element is deleted then the size of the tree is checked, such that it is less than the state of the tree before deletion. Also the module is checked when the tree has no elements in it.

- Simultaneously the Is_empty module is checked to verify that the tree is empty.

- I have also tested the max_key module such that if the key value being entered is greater than all key elements in the tree then after insertion the max_key should return that particular key as the output.

- The max_key module is also checked when the tree is empty which should return None.

- I have also tested the min_key module such that if the key value being entered is smaller than all keys elements in the tree then after insertion the min_key should return that particular key as the output.

- The min_key module is also checked when the tree is empty which should return None.

- Periodically the keys module is tested to check if all key elements in the tree are returned accurately.

Bugs Found in the System:

No bugs found in the script to report. Each of these modules tested do not report any bugs or faults in their behavior.

Code Coverage:

The amount of code covered recorded by the coverage.py program is 44%. And the program has covered 111 branch statements.

Output Snapshot:

Name	Stmts	Miss	Branch	BrPart	Cover	Missing
binary_search_tree.py	182	103	86	8	44%	14-19, 27-33, 36, 42-93, 112, 122,

131-136, 148-150, 159-164, 176-217, 232-305, 307, 326, 336, 344-373, 387,

Linked List :

Quality of System Under Test:

Compared to the beginning the test scenario has been improved by generating more test cases for more modules in the system. Couple of bugs were discovered and mentioned below.

Previous Completed Tasks:

No tests were made on this data structure system.

Current Completed Tasks:

Modules tested in the Linked List are

- add – This module adds a value into the list.
- remove – This module removes the given value from the list.
- search – This module searches for the given value in the list.
- Length/size – This module returns the length of the list.

In the Linked List source file I have tested the following:

- Adding a value after checking the element does not exist in the list. The property checked is that the size of the list has increased by 1 and the search function returns true on the given value inserted.

- Deleting a value after checking the element does exist in the list. The post property checked is that the size of the list has decreased by 1 compared to the previous state of the list. And the search function returns false on the given value deleted.

- Both search and remove modules are checked when the list is empty. For the search module it should return False and for the remove module it should return None.

Bugs found in the System Under Test:

There were few bugs/faults found in the System

- Size declared as member variable and function name hence function would not return value. Thereafter changed the name to function “length”.
- Cases for empty Linked Lists are not handled by the remove and search function. So cases where the Linked List is empty and these modules are executed, None types are not handled by them and hence generate an error.

Code Coverage:

I have achieved 57.4% of code coverage and have covered about 40 branch statements.

Output Snapshot

Name	Stmts	Miss	Branch	BrPart	Cover	Missing
singly_linked_listmodified.py	49	22	14	1	57%	15-17, 21-27, 30-36, 40, 52, 80, 89-100, 86->89

Performance of Binary Search Tree V/S Linked List

- Finally, I have also tested the performance of insertions and deletions of the data structures with small and large data values. Keeping the property that only unique values exist in both data structures.

- While testing for performance, I return the microsecond average amount of time the Binary Search Tree takes to insert and delete an element based on certain number of calls. Similarly, with the same state of data, I test the same operations and return the microsecond average time taken by the Linked List.

Good Features of TSTL

- Gives an easy interface to test various properties and boundaries in a script.
- Gives in depth post analysis data based on the test cases generated.
- Logged Steps help to identify correctness of generated test cases and also sequence of actions.
- Even though syntax and semantics are hard to understand at first, it does not take long to figure out how it exactly works and thereafter it is simple.

Conclusion

In Conclusion, I have tested the properties of most of the modules in Binary Search Tree and all the modules in Linked List. The properties that these data structures should follow have been checked for, and the bugs found in fault scenarios have been reported. Finally, I have checked the performance of both these data structures against each other.

References

The scripts used have been imported from the Github library

For Binary Search Tree
https://github.com/nryoung/algorithms/tree/master/algorithms/data_structures/binary_search_tree

For Linked List

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