Varun Rajgopal
CS 562
Dr. Alex Groce

Final Report

Introduction

This report details about the results of my final version of the project. The python library that I have tested using the TSTL is "Bintrees 2.0.2". It can be downloaded from https://bitbucket.org/mozman/bintrees. AThe binary tree package provides Binary, RedBlack, and AVL trees written in Python and Cython/C. All trees provide the same API and pickle protocol is supported. For my term project I have tested the methods and interfaces for binary trees.

Accomplishment

For the term project I concentrated on checking the correctness of the methods and interfaces for binary trees. The basic methods and interfaces that I have tested for binary trees are:

- Insert – inserts an element in to the binary tree.
- Remove – removes an element from the binary tree.
- Get-value – returns the value for the specified key.
- __contains__(k) - True if T has a key k, else False
- __getitem__(y) - returns the value for the key
- __len__() – returns the length of the tree
- __max__() – returns the maximum key, value pair in the tree
- __min__() - returns the minimum key, value pair in the tree
- clear() -> None, remove all items from Tree
- is_empty() – returns True if len(Tree) is 0.
- foreach(f, [order]) -> visit all nodes of tree

I have also tested the heap methods of the binary trees. Following are the methods that I have tested:

- max_item() -> get largest (key, value) pair of T
- max_key() -> get largest key of T
- min_item() -> get smallest (key, value) pair of T

- min_key() -> get smallest key of T,
- pop_min() -> (k, v), remove item with minimum key
- pop_max() -> (k, v), remove item with maximum key
- nlargest(i[,pop]) -> get list of i largest items (k, v)
- nsmallest(i[,pop]) -> get list of i smallest items (k, v)

The bintree_for_tstl.tstl file contains codes for testing the binary tree. I have written several self-defined methods such as displayTree(), printHelper(), traversal(), maxNode(), minNode() and bst() in order to check if the tree is a binary tree during insertion and deletion operations. I have used a list data structure as a reference to see if an element is present in the tree or not. During insertion operation the key is also stored in the list and during deletion operation the list is referred to ensure that the element in present in the tree.

For inputs, I have used the integers as the key and upper case alphabets as the value. I have also used guards to testing the heap methods of the binary trees.

Bugs

No bugs were found in the python library. Several failing tests were generated by the random tester. However, these failing testes were related to incorrect actions in the test harness and not to actual bugs in the library. As described in the bug report the quality of the library is high and is bug free. The functionality of the code works correctly and appropriate for the functions coded.

Tester

The tester worked well for all of the test cases that I have implemented for testing the bintrees python library. I have tested almost all the methods and interfaces of the Binary tree to ensure correctness and tester worked pretty well. All the insertion and deletion operations worked flawless. The binary trees extended from an abstract class called

abctree.py. I was able to test the methods that are implemented in the abctree.py using the tester.

Pros and Cons of TSTL

TSTL is a powerful and useful tester suite that contains pretty good features that required for testing a software. The features such as grammar structure, capability to implement raw python code inside the tstl file, test normalization and reduction. The important feature that I would like to talk about is the code coverage of the TSTL. Using code coverage, we can figure out percentage of the code that has been covered for testing, number of branches and statements that have been covered.

The downside that I would like to highlight about TSTL are documentation and tutorial. It would be difficult for a newbie to understand TSTL at the start. An instruction document highlighting the usage would be preferable.

Coverage summaries:

The code coverage for binary trees are as follows:

4.12371134021 PERCENT COVERED
240.016350031 TOTAL RUNTIME
6428 EXECUTED
642701 TOTAL TEST OPERATIONS
152.561074495 TIME SPENT EXECUTING TEST OPERATIONS
80.5538187027 TIME SPENT EVALUATING GUARDS AND CHOOSING ACTIONS
1.02543616295 TIME SPENT CHECKING PROPERTIES
153.586510658 TOTAL TIME SPENT RUNNING SUT
0.619472265244 TIME SPENT RESTARTING
0.0 TIME SPENT REDUCING TEST CASES
35 BRANCHES COVERED
24 STATEMENTS COVERED

Varun Rajgopal
CS 562
Dr. Alex Groce

Coverage.out:

Name	Stmts	Miss	Branch	BrPart	Cover	Missing
bintree.py	73	73	24	0	4%	9-124