Prathveer Rai

CS 562

Date : 24th January 2016

Project Proposal

The project I will implement is based on the data structure of Binary Search Trees and B+ trees. A Binary Search Tree are used to store data items in memory. They are a tree shaped structure which contain nodes and leafs. These nodes and leafs contain key values stored and are organized in a sorted order where nodes with key values greater than the root node are towards its right and values less than that of the root are to its left. This criteria holds good for all the nodes. I have additionally decided to test for B+ trees on the possibility that generating test cases for just the BST would not be a big enough task. A B+ tree is a more efficient self-balancing tree that is used more commonly in databases today.

The Binary search tree source file is https://github.com/ashish0x90/pytree/tree/master/src/pytree. And the B+ tree source file is https://github.com/aholyoke/BPlusTree/blob/master/BPlusTree.py. These are not part of standard python library and hence should probably contain some amount of bugs. The sources being tested has several modules that represent the manipulation features of the Binary Search Tree and B+ trees. Some of these are as follows:

BST:

InOrder() – Returns node/leaf values in the ascending order of their key values.

PreOrder() – Returns the preordered traversal of the tree.

Find()- Returns the node that matches the given key value

Findall()- Returns all matching node.

Insert()- for a given key we need to insert the particular node/leaf key value in the tree structure

Delete() – for a given key we have to delete the particular key node/leaf structure from the tree.

B+ Tree:

Constructor – Initializes the tree with specified degree

Insert() – Inserts key to specific node and split when necessary.

Find()- returns true if key in tree

Height() – Returns height of tree

These modules will need to be tested based on various inputs that could identify some bugs in the program. Additionally the properties of the tree have to also be tested. Certain properties that need to be check are the sorted key value nodes of the entire tree. For BST, the left child node of the root node has to be smaller than the root value and the right child node has to be greater than the root node. Hence this tree has to adhere to this sorted property always. Also with insertion, deletion certain boundary conditions have to be checked such as deleting from an empty tree or inserting the same values into a tree. For the find functions, finding values in an empty tree can a valid test case, also finding invalid inputs and testing for expected behavior. Additionally, the traversal modules of InOrder() and PreOrder() need to be tested as well so as to check whether they follow the right path on all test inputs.

As for B+ trees, the number of keys in a node have to be equivalent to the criteria of the degree specified (d to 2d keys per node and d – 2d+1 pointer values). This is the general property that the tree has to adhere to throughout its changes. Some additional test inputs that extend from the BST is checking the property of total number of nodes in the tree and the height of the tree. Test inputs can generated to check whether the number of nodes always returns the right value for a large size and also the height of the tree should return the right value for the height of the tree when big inputs or no inputs are given. These are some of the properties that could be checked for correctness. Also the insert method used in B+ trees need a different check as compared to BST, since it works differently from a BST. A test case could be written to check correctness of the split of a particular node, when a value extending the degree size is inserted.

These checks can be done with the help of TSTL. TSTL helps create random test cases of various input sizes to help illustrate any faulty implementations in a source code. It helps to detect these bugs in the initial stages of their occurrence or intervention and lists out a detailed report of various statistics. With TSTL automated scripts, we can assign various sizes of inputs to all these modules and test their correctness and reliability. We would also need to check if it can handle different inputs. And also there is a need to assign various boundary analysis cases to test the robustness of the modules and the properties it needs to follow.

Following this an additional aim here is to try and attain maximum Coverage and if this not achieved then there may be dead paths that need to be identified and corrected.

In Conclusion, with the help of TSTL automation scripts. It will be a good challenge to try and generate several random test cases that would help me identify any potential bugs that exist in the sources mentioned. And by doing so I hope to understand more intricate details of testing and also be a consistent user of the TSTL software in my future projects.

References:

https://github.com/ashish0x90/pytree/tree/master/src/pytree

https://github.com/aholyoke/BPlusTree/blob/master/BPlusTree.py