Analyzing Binary Search Trees
Lab #8

By
Michael Harden

CS 303 Algorithms and Data Structures
October 26, 2021

1. Problem Specification

The goal of this lab is to implement a Binary Search Tree and populate it with data. Then, Test the implementation by searching for a key and verifying the correct description is returned. Finally, calculate the time it takes to find a list of items in the Tree.

2. Program Design

This program contains two classes. First is the BST class. This class implements the Binary Search Tree. It provides methods to search, insert, and in-order traverse the tree. The second class is the Driver class. This class provides methods to parse a csv file and load it into a BST, load a test file, and time the amount of time to find all the test keys in the tree.

The following methods are provided in the BST class:

a) __init__(value):

Initializes a new Binary Search Tree instance. If a value is passed, the root nodes is set to the value passed.

b) search(key):

Searches the tree for the key. If the value is found, the reference to the node with the key is returned. Otherwise, None is returned

This method works by first, setting the current node to the root node. Then the current nodes value is compared to the key. If they are equal, the current node is returned. If the current nodes value is greater than the key, then the current node is set to the current node's left child, otherwise the current node is set to the right child. This process is repeated until the current node is None. In this case, the tree has been searched and the value was not present in the tree so None is returned.

c) insert(value):

Searches the tree to find the properly spot to insert the value passed.

This method works by first checking if a root node is present. If not, The value is inserted at the root. Otherwise, the same process is followed for search to find the properly ordered spot to insert the value. As the value for the current node is moving down the tree, the parent node is also kept track of. This is done so that, once the proper place is found, the parents left or right child can be set as the value passed in.

In this lab, the values from the csv are stored as tuples. In python tuple comparison is done by checking the first element of the tuples and comparing them. If they are equivalent, then the next items are checked.

d) inorder():
Prints each nodes value in the pattern left, self, right.

This is done recursively by first calling on the left child until it is None, then back tracking, printing the value of the previous node. Then the right child. For each node visited this pattern is continued.

e) Class _Node:
The Node class is nested withing BST. It has values left, right, parent, and value. Left, right, and parent are pointers to the respective nodes. Value is the value the node stores. For this lab it stores a tuple, with index 0 containing the key, and index 1 containing the description

The following methods are provided in the Driver class:

a) _loadfile(filen_path):
Loads the file passed into file_path into the format of an array where each index is storing a tuple which stores at the first index the key and at the second index the description.

b) csv_to_bst(file_path):
using _loadfile, the file at file_path is parsed, and its content is randomized then loaded into a BST.

It is randomized because each row is sorted. If this is loaded into a BST it would just be a linked list.

c) load_test(file_path):
using the _loadfile method, the file at file_path is parsed into a list of test cases.

d) time_BST():
times how long it takes for all search cases to be found. The timing is done by first storing the time right before searching in to start. Then the time is taken for directly after searching for each key. The difference between the end and start time is the total time.

3. Testing

To test the BST class, a tree was created with the UPC.csv file. Then test cases were selected which contains nodes that are in the tree and nodes that are not in the tree.

Test cases:

Key	Expected Description
2056135	LB, BREWSTER CLBY JACK CUBES
6913403	1 lb,*KRAFT SHREDDED CHEESE BLEND

10016503	Tesco 2 little gem lettuce
11772728	None
20262630	None
15414542298	None

To test the inorder method, the following numbers were put into a tree [5, 7, 3, 4, 9, 8, 10, 2, 1, 6]. If the inorder method works correctly the expected output is [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], which is the output gotten.

4. Analysis and Conclusions

In conclusion a Binary Search Tree that is mostly balanced is very fast. To find all 17 test keys only takes 7.295608520507812e-05 second when searching through a tree with over 100,000 nodes, This is because searching takes O(log n) time since each level down the tree divides the searching space in half. One disadvantage to the tree is how it is not naturally balanced, and one side can become more densely populated than the other, or in the worst case, if all the elements are presorted the tree would turn in to a linked list. This would cause the time to search for or insert an element O(n) because each step down the tree only removes one node at a time. In this case to build one of these trees would be O(n^2) time. That is why it is so important to maintain a balanced tree to have O(log n) time complexity for searching and inserting nodes, and O(n log n) time to build the tree

5. Credits

- The implementations of the BST in part used the pseudocode provided on canvas
- The example lab report provided on canvas was used as a template for this lab report.