**Introduction:**

In Lab 4 we were given the task to implement a version of a binary search tree and several methods manipulating the search tree. The first method we were tasked in creating was a method called size, given a tree return the total size of a tree, another one called minimum that returns the smallest item in the bst and a similar one called maximum which returned the largest item in the tree, the height method which returns the total height of the tree, the inTree method which takes a tree and item I and returns true if I is in the tree and false if it isn’t, a method called printByLevel which prints the tree in level order, a method called tree2List which returns the sorted tree in a python list, a method called leaves that returns a list of items in the tree which are leaves, a method called itemsAtDepthD which takes a tree and a depth and returns the items at that depth in a list, a method called depthOfK that takes a tree and a node and returns the depth of that node if its in the list and if its not in the list return -1, and finally a draw function that draws the tree.

**Proposed Solution Design and Implementation:**

Method size:

The Size Method was very simple, we can figure out the size of the tree recursively.The base case of the method being if the tree is Null just to return 0 otherwise return 1 + the size of the left and right subtree.

Method minimum:

The minimum method was also very simple, I decided to traverse the tree recursively the base case being if the tree has a left subtree return the minimum of the left subtree otherwise return the item, this will get to the minimum item in the tree.

Method maximum:

The maximum method is similar to the previous minimum method and I completed it in the same way, if there is a right subtree find the maximum of that otherwise return the item, this will traverse all the way to the rightmost item and return it.

Method height:

In the Height Method I calculate the height of the tree recursively, first we check of the tree is equal to null and if it is return 0 if its not we return 1 plus the maximum the height of the left subtree and the height subtree so whichever is picked it only counts the highest branch of the tree.

Method inTree:

In the inTree method we check the tree recursively, first we check f the tree is not null, if it isn’t null then we check the root of the tree if its =equal to I then we return true if its not inTree of the left and right subtrees and if its other then that return false.

Method printByLevel:

The printByLevel method was given to us in a previous class we make use of a queue structure and a while loop to print the tree level by level, so first we check if the tree is null if its not we create a queue with the tree in it. Then while the q is not empty a temp variable stores the popped item from the queue then as long as that tree isn’t null it prints it and adds the left and right subtree to the queue this will eventually print the tree level by level.

Method tree2List:

The tree2List method is pretty simple we recursively return a list sorted by traversing it in preorder left, root, then right. First we check if the tree is null if it is return a empty list other wise return tree2list of the left subtree, the root in a list and the tree2list of the right subtree.

Method leaves:

The Leaves method was done similar to the tree2List method with the key difference that to be added to the list you have to be a leaf, first we check if the tree is null if it is return an empty list, then the second base case is if there is no left and right subtree we return the root in a list otherwise return leaves of the left subtree and leaves the right subtree

Method itemsAtDepthD:

The itemsAtDepthD method wis similar to the leaves method except our second basecase is replaced with checking the depth d, so first we check if the Tree is null if it is return an empty list if its not we then check the depth, if the depth is 1 we return a list containing the root, otherwise we return itemsAtDepthD of the left subtree and d minus one and itemsAtDepthD of the right subtree and d minus one

Method depthOfK:

The depth of k method was the most difficult for me personally I was truing to figure out away to traverse the tree and return the depth of a item completely recursively, eventually I realized I had to store the depth in a variable instead so first we check the root and if it equals k return 0 otherwise we check if k is bigger or smaller then the root so if its bigger we traverse right or if its smaller we traverse left, so we set depth is equal to 1 plus depthofK and depending on k the left or right subtree, then we check if depth is greater then 0 if it is return the depth if its not return -1.

Method Draw

The Draw method was a modified version of the previous draw method used for the binary search tree node implementation modified to work with the list implementation here.

**Experimental results:**

All of the methods in this lab work and many of the methods here are very similar and based on traversing the tree, A couple of the methods I believe could be improved would be the printByLevel method and the depthOfK method. The printByLevel method makes use of a queue to print the levels of the tree, this could be done much simpler, because the tree is in a list it makes it easy to print by level because the list is already in level order from left to right the problem with this approach was the fact that the subtrees a list inside of list at first I tried to flatten the list and just print it, none of the ways I found to do this worked because of the list inside of list, then I decided I could use the tree2list method that I wrote, but there was a problem with that because that method sorts the list as well, so that left me with using a queue but if I could figure out a way to flatten the list it could be done like that. The second method I feel could be improved was the depthOfK in this method I had to use k and traverse either left or right and I had to store depth in a variable, it could probably be done differently using a base case and just calling the same method and left and right subtree in one call but I couldn’t get it to work.

**Conclusions:**

There are many uses for binary search trees and trees in general, learning this implementation of a binary search tree was very essential in understanding the nature of the data structure itself. This lab shows the possibilities of different ways we can implement and manipulate disticnt data structures as well as shows us certain pros and cons associated with doing so.