Kai Muramoto Lai
CMPT 225 D-100

Assignment 2 Coding Write up

Functions testing

The functions that I tested for in this assignment are void displayLinks(), height(), and depth ().

The first thing I tested was my displayLinks() function. I did this by first creating a tree in DLtest.cpp. Then I manually looked at the addresses of each parent to see if they matched the addresses of the right and left children. If a parent did not have a right or left child it would print 0, indicating that there is no child and therefore not giving us an address. Displaying the links gave me a visual representation of each link in the BST.

The second thing I tested was my depth() function. I did this by keeping track of the root node at all times in a tree once a BST was constructed. Then I would input an int into depth(). First it recursively checks to see if the given int was in my BST, if it is it traverses the tree from (x, root) and returns the depth. If a particular int is not in the BST it would return -1 just like my implementation in Dtest.cpp. I tested my correctness and validity by inserting multiple values into my trees and calling the depth() function again to change the height of the tree. Which it did. Finally, I tested my height () function. I first made sure if the tree was empty it would return 0. Then I computed the height of both the left and right subtree. After that, if the left subtree was larger I would return (leftheight + 1), and else return the right + 1. This program worked as intended as seen in my Htest.cpp. To make sure my program was robust I tested the function on multiple BST trees. All of these function tests were performed on both a BST and an AVL.

Exp1.cpp

The functions I tested in experiment 1 are height(), depth(), contains(), and elapsed_time() on a BST. In my experiment, I first created a "tall and skinny tree" called T1, then a "natural tree" called T2. Then I performed membership checks using contains () on both trees. After performing membership checks the times were interesting. The more natural tree, T2, took less time to perform membership checks. The total time for membership checks on T1 was around 664ms while T2 was around 34ms. Just to check for validity and robustness I ran it a couple of times and these were valid. For average time, the same thing occurred. The average time taken for T2 was 0.00226652ms while on T1 it was 0.0442637ms. This information is correct since in class we discussed how it is easier to traverse a "natural tree" than a "tall and skinny tree". Additionally comparing the depth, the depth for T1 was larger than the depth for T2. For depth (300), T1 had a depth of 300 while T2 had a depth of 38. Moreover, the average depth of node 300 is 169 for T1 and T2. Again this is true since tall skinny trees have more levels than natural trees therefore skinny trees having greater depth than natural trees are correct.

Exp2.cpp

The functions I tested in experiment 2 are displayLinks(), height(), depth(), contains(), and elapsed_time() on AVL trees. In my experiment, I created two trees, one BST tree named T1 and one AVL tree named T2. Both of these insertions were the same (tall skinny tree insertions). After running insertion times on both trees, AVL trees are longer for insertion. AVL tree 3737ms while BST, is

715ms. This is most likely due to the fact that AVL trees balance during insertion so, therefore, took longer for insertion. The average insertion time for the BST was 0.0480699ms while the AVL tree was 0.252017. Moreover for membership checks, AVL was much quicker. AVL trees have a time of 1ms while BST has a time of 591ms. This is also true since membership checks on an AVL tree are much quicker since the tree is balanced. For the height of the BST the height was 14999 while the AVLtree had a height of 13. Again this is true since AVL trees balance during insertion so therefore having a much lower height compared to the BST (which is tall and skinny). Finally, the average depth of 300 for both the BST and AVL tree was 96. Overall I believe the experiments and test cases I have created for this assignment were robust and tested for my functions and timings.