Name: Shahroz Imtiaz

Date: 10/18/18

Lab: CS 2150-107 LAB (16877)

The results from testfile1.txt, testfile2.txt, and testfile3.txt suggest the relative performance of AVL trees is much faster for searching than Binary search trees. The AVL tree output for the path was always shorter than the path that the Binary search tree outputted because of the fact that an AVL tree is self-balancing, and therefore isn't heavier on side like you might see in a Binary Search Tree.

AVL Tree vs. Binary Search Tree

AVL trees and Binary search trees have a lot of similarities, but the key difference between the two is that AVL trees are self-balancing. Because AVL trees are self-balancing, we don't have to worry about our tree being heavy on side. This is great because we want our program to be fast as possible, and if we're using a Binary search tree to store data and the tree is heavy on one side, our program will be slow. Binary search trees are known to generally have a O(log(n)) when accessing, searching, inserting, deleting. HOWEVER, if the Binary search tree is really heavy on one side, this is not the case! Accessing, searching, inserting, deleting can be O(n) if the tree is heavy (ouch!). That's why AVL trees are great. Their ability to self-balance guarantees a O(log(n)) for accessing, searching, inserting, deleting. AVL tress insure balance by keeping track of a tree's balance factor (height Of Left Subtree – height Of Right Subtree). Unbalanced trees have a balance factor of 2 or -2. The balance factor will never be higher than 2 or -2 because once a 2 or -2 balance factor appears, it is immediately "fixed" by doing

rotations. There are four types of possible rotations: Left-Left (single rotation), Right-Right (single rotation), Left-Right (double rotation), Right-Left (double rotation). These rotations ensure a balanced tree.

Having a tree that's always O(log(n)) is great. That's why you might want to use an AVL tree if you're inserting sorted data into a tree, because if you use a Binary search tree, your tree will be extremely heavy on side and you will end up with linear run-time. But with an AVL tree, it'll self-balance and you'll still have O(log(n)) for any operation except for printing, which would still be O(n). If you're going to be searching through data a lot, again, an AVL is better because you're guaranteed O(log(n)).

In conclusion, implementing an AVL tree is more time consuming than a Binary search tree and can sometimes be a hassle if you're not an experienced programmer. But the benefits of an AVL tree's log run-time far outweigh any drawbacks. If you know that you will be searching for data a lot, you might want to implement an AVL tree because it's a safe bet for being faster or as fast as a Binary search tree.