Derek Johnson, dej3tc

10/17/2019

Lab at 12:30 on Tuesday

Post Lab Report: CS2150, Lab 5

1) For both the AVL and Binary trees, the commands that needed to be run in the test files were completed in similar time. This would lead us to believe that there is a similar run time between the two, especially in a situation like those provided where the trees are relatively evenly sorted. It should be noted that in all three tests, the AVL tree had to look through less nodes in order to find the target node. This means that for the cases tested, the AVL had better efficiency in its search method.
2) When comparing the space-time tradeoffs between the AVL tree and Binary tree, you can see that the relative efficiency of each depends on the method that is called and the shape of the tree. In their worst case, the AVL tree will take $\theta(\log n)$ while the Binary tree will take $\theta(n)$ for insert, find, and delete. When looking at a well-balanced tree however, the Binary tree can outperform the AVL tree on insert and delete because it does not have to rebalance the tree or recalculate the balance factor of the nodes affected by the rebalance. In the case of find, the AVL tree will always take equal or less time than a Binary tree. The tradeoff with this is that AVL trees will take more space in memory as they must store the balance factor of every node.
3) AVL trees will be preferable to Binary trees in situations in which the most common method call is find. This is because with the find command you will get the benefit of balancing without any of the time cost associated with rebalancing. Additionally, AVL trees would be used when space in memory is not more important than execution speed.