John Fishbein - Lab Section 106(5pm) Thursday, February 28, 2019

CS 2150 Lab 5 Report - Trees

After implementing both a Binary search tree and an AVL tree, I was able to compare their performances on test file1.txt, testfile2.txt, and testfile3.txt. Both data structures were given the same data to process, and both performed the same operations. Since these test files were relatively small, both the unmodified Binary search tree and the AVL tree performed about the same when handling the data.

There is a very clear space-time tradeoff between an Binary search tree and an AVL tree. While they both ran almost instantaneously on the three test files, given a much larger dataset, the AVL tree would have run much faster. In the Binary search tree, find() insert() and remove() are all worst case O(n) runtime because the tree is not guaranteed to be balanced. In the AVL tree implementation however, the tree is guaranteed to be as balanced as possible, allowing find() insert() and remove() to run in Θ(log n) time. The tradeoff occurs because, the AVL tree requires more memory to implement than the unmodified Binary search tree.

AVL trees are preferable to unmodified Binary search trees in cases when a large dataset needs to be processed. In that case, it is worth extra computer memory to ensure a faster runtime. When dealing with large datasets, it will take to long to perform find() insert() and remove() on a data structure with worst case linear time for those operations.