Julia Shea

10/17/19

12:30 Lab Section

CS 2150 Post-lab 5 Analysis Report

Although AVL trees and binary search trees are both ordered implementations of the tree data structure, they have noteworthy differences that can impact the runtime of tested programs. Although the input sizes of the three test files are all too small to notice a drastic difference in runtime, BSTs have a significantly slower runtime than AVL trees when the program begins to approach larger input sizes. When the input strings are generally in alphabetical order before insertion into the tree, BSTs can be particularly slow as all of their operations (insert, delete, and find) involve iterating through the entire tree–with ordered inputs, this can end up resembling a linked list. AVL trees, however, have a balance factor, so the tree will be faster to iterate through with large input sizes.

More specifically, a binary search tree can only guarantee linear runtime, meaning that the algorithms for finding, inserting, and deleting elements will increase proportionately with every input added. The height of a BST is always h, and the runtime is Big-Theta h. An AVL tree, however, can guarantee logarithmic runtime for all three operations, because the balance factor allows the algorithm to become more efficient with every input–the height of the tree is Big-Theta log(n).

Generally speaking, AVL trees are preferable to binary search trees in situations for which inputs are already ordered, like alphabetical lists of names or sequential identification numbers. In these situations, the binary search tree will end up resembling a linked list and will have the worst-case possible runtime, because the only ordering property is that the right children of each node must be greater than the node itself (and vice versa with their left children). As previously mentioned, AVL trees have the balance factor that prevents each node from simply becoming the right child of the previous node, so AVL trees will be more balanced and faster to iterate through.