Bradley Allen

Professor Frank Li

CSCIU321-01M

10 June 2022

AVL vs BST: Project Report

The program randomly generated five input files containing random integers. These five input files, named A1, A2, A3, to A5, contained 2000, 4000, to 10000 randomly generated integers, respectively. These input files were used to build an array that contained all of the integers in the order of the file. Then, the array was used to build a binary search tree. To remove half of the input size nodes, an array was created that held all of the nodes to be removed. These nodes were generated randomly by generating integers that represented indices of the array that held all of the integers. The building of the tree and removing half of the nodes from the tree were timed in nanoseconds.

In addition to creating the trees with randomized integers, the program also creates the trees with incrementing integers from 1 to the input size of each file. This was done by replacing the file reading with a loop that incremented the index along with the value stored at the index.

The table below shows the execution times (in nanoseconds) of building a binary search tree and an AVL tree on input data group 1, which is input from the five randomized integer files, and input data group 2, which is the five incrementing input files.



The next table shows the execution times (in nanoseconds) of removing half of the input size nodes from a binary search tree and an AVL tree on input data group 1 and group 2.



The graph below shows a comparison between the building times of each group and both trees. The graph is heavily skewed due to data group 2’s binary search tree building taking a much longer time than the other times.

The next graph shows the comparison between the removing times of each group and both trees. The second group’s binary search tree still completely dominates the scaling of the graph.

While the binary search tree is efficient at building and removing randomly generated integers, the tree structure does not handle incrementing integers as efficiently. This is due to the recursive calling to place each node, as the incrementing integers must be compared when inserted. Unlike the randomly generated integers, the incrementing integers continuously place to the right child of the current node, making the tree unbalanced and each node result in an additional recursive method call.

While the AVL tree is not as efficient at handling randomly generated integers, it does handle the incrementing integers much faster than the BST. The AVL tree can handle this faster because as the nodes are inserted into the tree, the tree is rebalanced if need be. This results in less recursive calls to place nodes since each added layer of the AVL tree will be balanced.

This project required the creation of multiple Java programs. The first program that was created, IntegerFileGenerator.java, handled the generation of the five input files holding randomly generated integers. The second and third program were copies from the textbook of the required tree classes, BinarySearchTree.java and AvlTree.java. These allowed the building and removing of nodes from the trees. The fourth program was Step\_2\_1.java, which handled input group 1’s binary search tree. The fifth was Step\_2\_2.java and handled input group 2’s binary search tree. The sixth and seventh, Step\_3\_1.java and Step\_3\_2.java, handled input group 1 and 2’s AVL trees, respectively.

From this project, I learned how to create both a binary search tree and an AVL tree in Java. I learned more about the efficiencies of both of the trees and why one was more efficient than the other. In addition to learning about the trees, I also learned more about how to organize my code while including enough documentation for later reading.