BST AND AVL TREE REPORT

By Bendik Svalastog and Riadiani Marcelita

PROJECT 2

For our second project, we learned how to implement two kinds of trees: Binary Search Tree (BST) and AVL tree. A BST is a tree that could only have two children, and its left subtree must always contain values smaller than its root, and the right subtree containing values bigger. An AVL tree is a BST that is balanced, meaning the height of the left and right subtree must never differ by more than one.

For our first operation of the second part of the project, we generated n-random integers to then manually insert into an initially empty BST and AVL tree. We chose an n-value of 10,000, with the values ranging from 1 to 9,999. We then inserted them to our BST and AVL tree and printed out the time it took for each tree to complete the task. We produced the following println statements after our program compiled:

AVL inserts 10000 values in ms: 56

BST inserts 10000 values in ms: 14

After running the tests several times, we kept receiving roughly similar time for the two trees' operations. In each case, AVL always takes longer time to insert values than the BST does. We conclude that this is due to the fact that whenever values are inserted into an AVL tree, the tree needs to run through its entire height to check if it is balanced or not. If it is not, the AVL tree additionally has to balance itself, and this takes more runtime and operation cost. This case does not occur in BST, however, which only has to compare the values inserted with its root without having to balance itself, causing it to be able to insert integers faster.

BST and AVL tree have different runtime, or T(n). BST, in its average case, has a runtime of O(log(N)), and a worst case of O(N). Whereas AVL tree has a constant runtime of O(log(N)), be it average or worst case. O(log(N)) is the runtime for single insertions, deletions, or searches in AVL trees. However, the time it takes to insert values into and build the tree is O(Nlog(N)), which explains why inserting values into, and building, an AVL tree, takes a considerately longer time than does the BST.

For our second operation, we had to search our BST and AVL tree and check its runtime. We also generated k-random integers for this search operation, and we also use 10,000 for our value of k. In one of our operations, we produced the following println statement about its runtime:

AVL searches 10000 values in ms: 4

BST searches 10000 values in ms: 17

After our experiment, we can conclude that AVL is more effective in terms of searching for values than BST. AVL takes around 10-13 milliseconds less time to search for values than the BST does. This is due to the fact that AVL trees have a runtime of O(log(N)) for its average and worst case, whereas BST has a runtime of O(log(N)) for its average/best case, and O(N) for its worst case. So when we run search in our AVL tree, it will always run O(log(N)) time, but when we run search in BST, it might run O(log(N)) time, or O(N) time. If it runs O(N) time, it will run slower than an AVL tree, resulting in it being more inefficient than a BST.

For our third operation, we had to mix and match the values that we inserted and searched for in our trees. We did several experiments and tried values ranging from small, as small as 10, to high values, up to 100,000.

Based on our experiment, we found out that in terms of inserting values, a BST is always more efficient than an AVL tree. From the beginning, with n = 10, up to the end, with n = 100,000, inserting values into an AVL tree always takes more time than it does into a BST. This gap in time increases linearly with the value of n. However, in terms of searching for values, an AVL tree is always more efficient than an AVL tree. In cases where n is a relatively small value, in example, n = 10 and 100, the runtime for search in both trees are considerately the same. However, as the value of n increases, it is visible that an AVL tree searches significantly faster than a BST. These findings are correlated with the runtime AVL tree and BST have as we mentioned above.