AVL & BST Performance Report

Josue Montenegro

CS 130A

I ran three tests that timed the operations *Insert, Access,* and *Delete* in both BST and AVL tree data structures. Each of the tests considered the following scenarios:

TEST 1: inserts N elements in increasing order, accesses them and deletes them in the same order

TEST 2: inserts N elements in increasing order, accesses them and deletes them in reverse order.

TEST 3: inserts N elements in random order, accesses them and deletes them in random order

Each of the tests performed those three operations using 1000, 10000 or 50000 numbers and compared the time it took to finish them depending the data structured that was used. This is a sample of the test results:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ATTENTION!!! Next tests use 1000 numbers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEST1 inserts N elements in increasing order, accesses them and deletes them in the same order.

TEST1 starting...

------AVL------

timeInsertAVL: 0.001363

timeAccessAVL: 0.000327

timeDeleteAVL: 0.000741

AVL Total Test Time: 0.00249

------BST------

timeInsertBST: 0.011086

timeAccessBST: 0.008923

timeDeleteBST: 4.8e-05

BST Total Test Time: 0.020072

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEST2 inserts N elements in increasing order, accesses them and deletes them in reverse order.

TEST2 starting...

------AVL------

timeInsertAVL: 0.000785

timeAccessAVL: 0.000216

timeDeleteAVL: 0.000579

AVL Total Test Time: 0.001591

------BST------

timeInsertBST: 0.008746

timeAccessBST: 0.007772

timeDeleteBST: 0.007373

BST Total Test Time: 0.023902

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEST3 inserts N elements in random order, accesses them and deletes them in random order.

TEST3 starting...

------AVL------

timeInsertAVL: 0.0008

timeAccessAVL: 0.00024

timeDeleteAVL: 0.000695

AVL Total Test Time: 0.001747

------BST------

timeInsertBST: 0.000296

timeAccessBST: 0.000274

timeDeleteBST: 0.000267

BST Total Test Time: 0.000847

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ATTENTION!!! Next tests use 100000 numbers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEST1 inserts N elements in increasing order, accesses them and deletes them in the same order.

TEST1 starting...

------AVL------

timeInsertAVL: 0.008762

timeAccessAVL: 0.002131

timeDeleteAVL: 0.005764

AVL Total Test Time: 0.016671

------BST------

timeInsertBST: 0.402276

timeAccessBST: 0.336249

timeDeleteBST: 0.000165

BST Total Test Time: 0.738698

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEST2 inserts N elements in increasing order, accesses them and deletes them in reverse order.

TEST2 starting...

------AVL------

timeInsertAVL: 0.003956

timeAccessAVL: 0.001025

timeDeleteAVL: 0.002876

AVL Total Test Time: 0.007864

------BST------

timeInsertBST: 0.355395

timeAccessBST: 0.336336

timeDeleteBST: 0.340845

BST Total Test Time: 1.03259

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEST3 inserts N elements in random order, accesses them and deletes them in random order.

TEST3 starting...

------AVL------

timeInsertAVL: 0.005024

timeAccessAVL: 0.001585

timeDeleteAVL: 0.004549

AVL Total Test Time: 0.011166

------BST------

timeInsertBST: 0.002213

timeAccessBST: 0.001867

timeDeleteBST: 0.001849

BST Total Test Time: 0.005936

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ATTENTION!!! Next tests use 500000 numbers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEST1 inserts N elements in increasing order, accesses them and deletes them in the same order.

TEST1 starting...

------AVL------

timeInsertAVL: 0.02441

timeAccessAVL: 0.006093

timeDeleteAVL: 0.017274

AVL Total Test Time: 0.047785

------BST------

timeInsertBST: 9.12756

timeAccessBST: 8.53522

timeDeleteBST: 0.000828

BST Total Test Time: 17.6636

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEST2 inserts N elements in increasing order, accesses them and deletes them in reverse order.

TEST2 starting...

------AVL------

timeInsertAVL: 0.024121

timeAccessAVL: 0.006111

timeDeleteAVL: 0.017179

AVL Total Test Time: 0.047419

------BST------

timeInsertBST: 9.12008

timeAccessBST: 8.57747

timeDeleteBST: 8.85046

BST Total Test Time: 26.548

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TEST3 inserts N elements in random order, accesses them and deletes them in random order.

TEST3 starting...

------AVL------

timeInsertAVL: 0.032394

timeAccessAVL: 0.010818

timeDeleteAVL: 0.029327

AVL Total Test Time: 0.072552

------BST------

timeInsertBST: 0.014377

timeAccessBST: 0.012489

timeDeleteBST: 0.012415

BST Total Test Time: 0.039288

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

After running these tests several times, an average of each of the times could be estimated and plotted in graphs that could reflect the drastic performance difference between BSTs and AVL trees.

The performance results of function *Insert (int key)* were the following:

The graph for TEST 1 and 2 shows how inserting elements in increasing order in a BST takes significantly more time than in an AVL tree since every time a new element is inserted it needs to become the tail of the ascending order list. Interestingly a BST seems to perform slightly better than an AVL tree when the order of the numbers is randomized. This is expected since with randomized numbers the BST has higher chances of becoming a tree rather than looking like a linked list.

The performance results of function *Access (int key)* were the following:

The results for *Access* are very similar to the ones of *Insert* since in order to insert an element in the trees we first need to find (access) the place where the element needs to be inserted. We can note again a very slight difference between BST and AVL for Test 3 since that test represents an average case.

Finally, the performance results of function *Delete (int key)* were the following:

*Delete* was the function that presented the most different results among tests. This is due the way deletion is performed. In Test 1 for example, deletion in a BST takes lesser time than in an AVL since the element that needs to be deleted every time is found at the root while in an AVL such element needs to be found, deleted and the tree balanced. Despite of that, the time it takes to delete 50000 elements from an AVL tree doesn’t reach 1 second. In Test 2 every element that we want to delete will be at the end of the BST which will be an initial height of N which is terrible. Since an element needs to be found first in order to be deleted, the deletion time in Test 2 will resemble the time of the function *Access* for test 1 and 2. For Test 3, we again find a slight different among both data structures since this test represents an average case. In Test 3, deletion from an AVL tree takes slightly longer due to the height balancing that happens after certain deletions.

In addition to the previous analysis of each function, I’ve also decided to show with a graph what was the average performance time of each data structure depending on the test. The times shown next represent the times each data structure took to perform the three functions *Insert, Access,* and *Delete* altogether in each of the tests.

This graph clearly shows that in worst case scenarios for a BST, an AVL tree performs in a time similar to the average case scenario which is highly appreciated in complexity time.