Deven Ronquillo

27/10/17

CSCI 2125 - 001

Data Structures HW 3

Times

AVL Insert: 5117 ms

Hash Insert: 1216 ms

AVL contains: 3742 ms

Hash contains: 1070 ms

AVL Remove: 505 ms

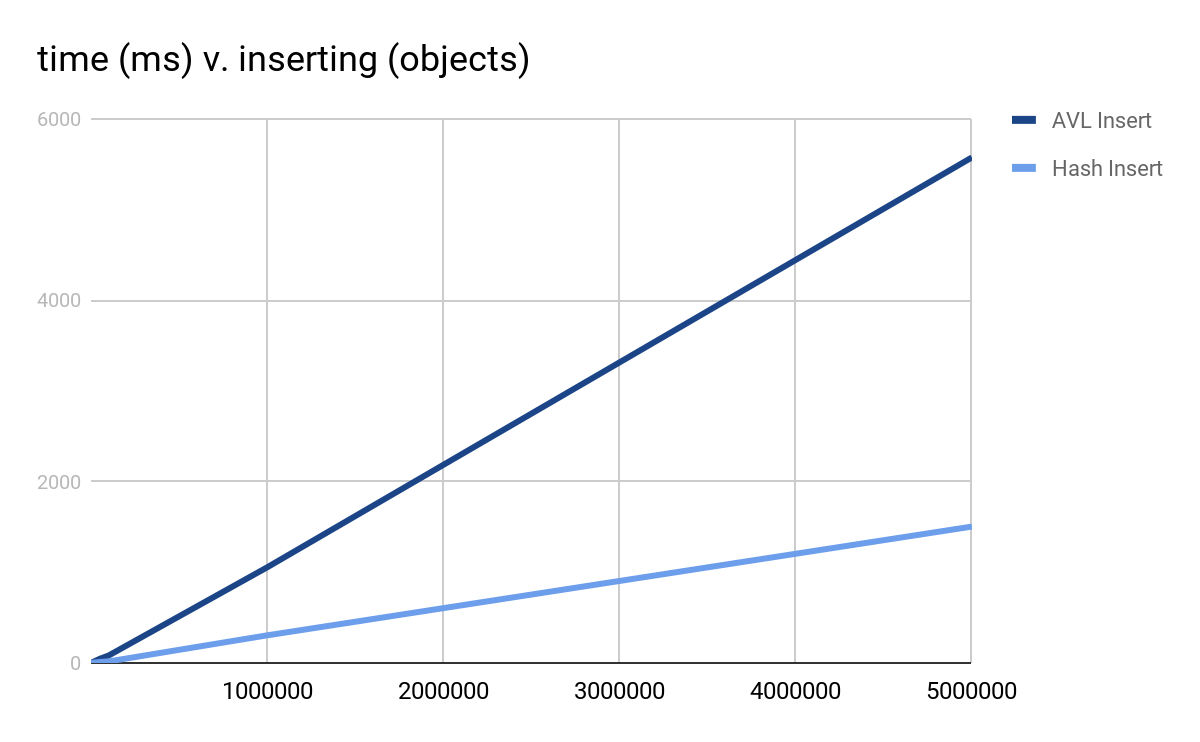
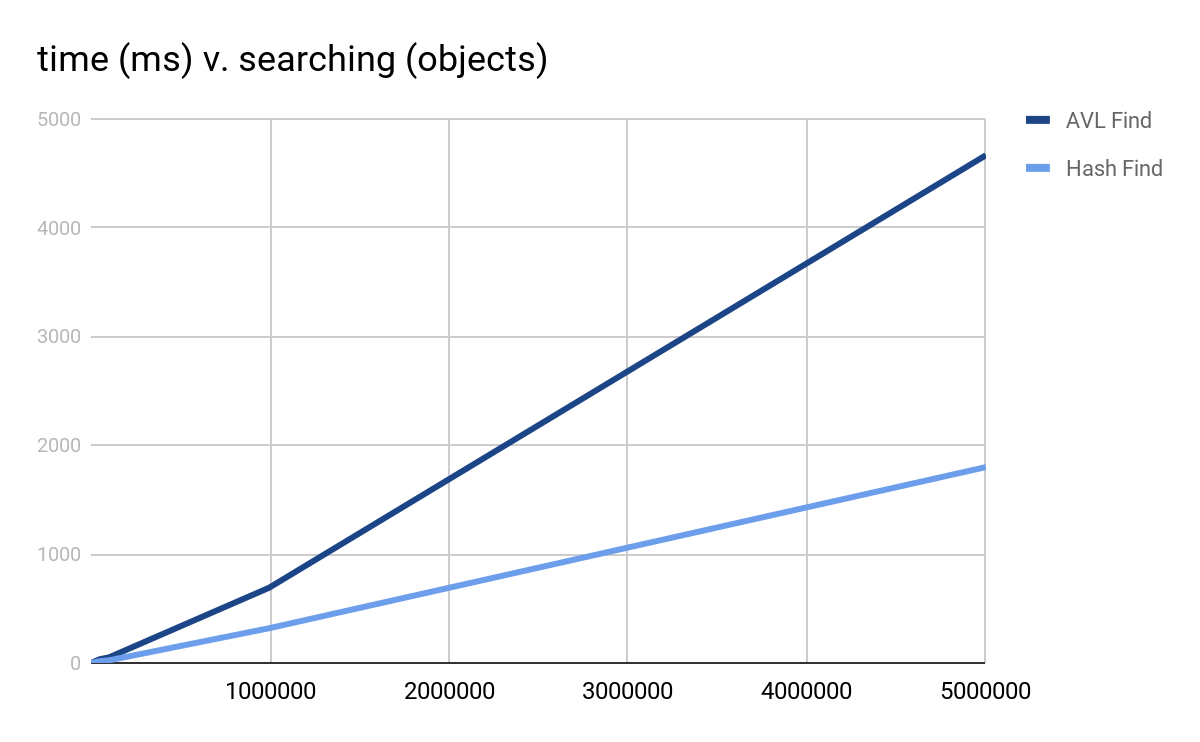
Hash Remove: 1046 ms

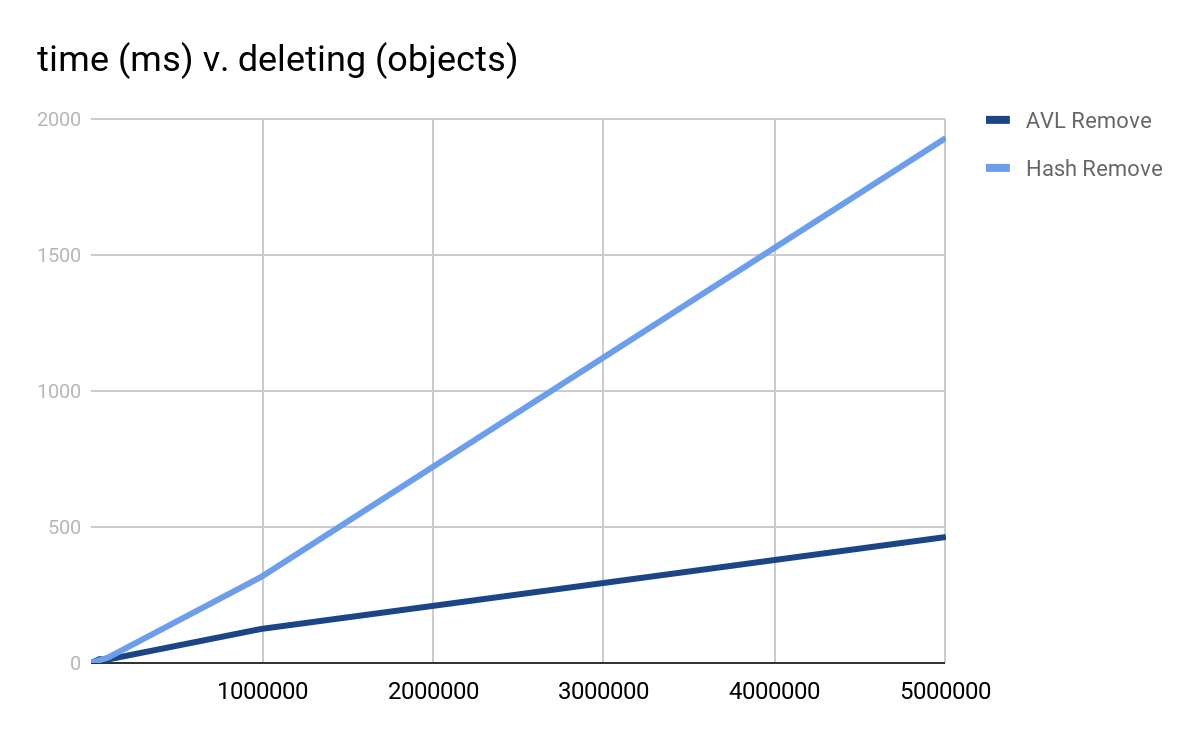
(recorded times are a total across 100 runs of different sizes)

For my HashTable I went for the simplistic approach. My hash function simply takes the key and mods it by the table size. Because of this there is a pre condition written in that the table size should be prime for best hashing results. For collisions the old value is replaced with the new value and we return the old value to the user. When piecing together the HashTable i was thinking of games and storing data as needed like currency, health, inventory space, mana etc. for things like ui and such. In this case we never need to know anything but the most recent value and rarely if ever need to get older values so this works out really well here.

The AVL tree mostly follows the book's code minus the methods they didn’t talk about so that one is straight forward.

My first graph shows the difference in insertion time of the two data structures. Since the AVL tree is balanced after every insertion the bigger the data set the longer it takes to complete an operation since you may start a cascade effect when inserting whereas the hash table is constant every time. (I was having some trouble figuring out how to work the graphs so they may be a little off to their true values)

Again there is a bit of a similar story here. As the data set grew, the avl tree started to take longer and longer to complete where the hash table stayed static across the board beating out the avl tree by a long shot.



Lastly there is deletion, which is pretty interesting. Here the avl tree took the lead time-wise and almost seemed like it was approaching a wall. As the size increased the amount the time increased by got smaller. As for the hash table, as the data size got larger the time it took to delete data grew larger and faster.