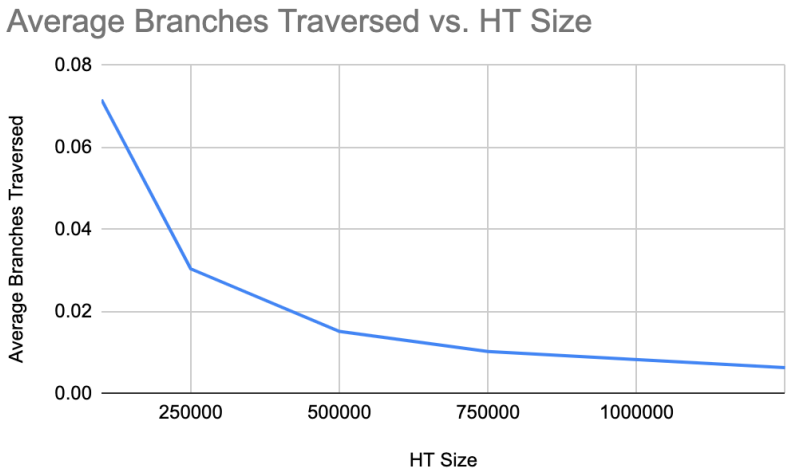
Karthi Sankar

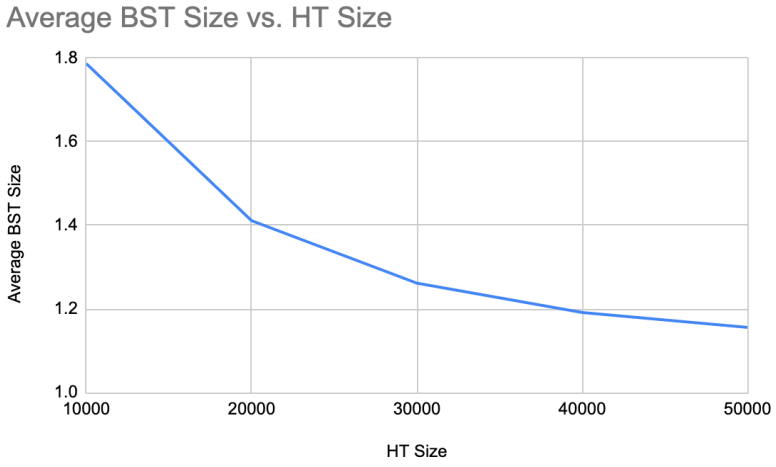
kasankar@ucsc.edu

12/05/21

CSE 13S Assignment 7 Write-up

Graphs:





Analysis:

The first graph I have relates to the number of average branches traversed as opposed to the hash table size. The overall trend is that the branches traversed decreases with a larger hash table size. With the characteristics of a hash table, this makes sense. The average branches traversed are calculated by dividing the number of branches in the BSTs by the number of lookups to the hash table. With a larger hash table size, you should not have to traverse as many branches because there will be less lookups to the hash table. There should be overall less lookups to the hash table because the likelihood of false positives decreases. With a larger hash table (or bloom filter), the number of false positives should decrease because it is more likely that nodes did not get inserted into the same index as each other and need to be checked further in the hash table.

I also want to discuss how the bloom filter size should affect the number of lookups specifically more in depth. The size of the bloom filter has a clear effect on the number of lookups. If a bloom filter is larger, then elements are more likely to not be inserted in the same index as others (no hash collisions). However, with a smaller bloom filter, there will inevitably be more hash collisions that will require the hash table to be checked. Checking the hash table for a word entails increasing the number of lookups because the table is being “looked at” or traversed for a certain node. So the relationship is inversely proportional: as the size of the bloom filter increases, the number of lookups to the hash table decreases.

If you decrease the size of the hashtable, the overall average size of the binary search trees decreases, as shown in my second graph. Overall, this is because with a larger

hash table, it is much less likely to have hash collisions or to have multiple nodes inserted in the same index in the hash table. If multiple nodes are inserted into an index (which contains a binary search tree), then the height of that tree will obviously grow. So, with a smaller hash table, the probability of larger (or taller) BSTs in the hash table increases. This relates to the “pigeonhole principle.” If there are more entries to put into a container (hash table) that doesn’t have enough slots, then it is much more likely for multiple things to be put into one slot (index in the hash table). This is why it makes sense that average BST height decreases linearly along with the hash table size.

Another factor that affects BST height is the order in which nodes are inserted into the trees. If nodes that are solely less than (or greater than) the root node, then the node will skew to either the left or right. This leads to much taller trees. A more balanced tree with nodes to both the right and left of the root and with more differentiated values will be shorter overall. If nodes with differing values are inserted into a binary search tree, the tree will be “wider” or more spread out, instead of taller and skewed to one side or the other. So, the order of insertion into the tree affects the height of the trees. So, in terms of our program, this corresponds to the order of the words that the user enters to the program will affect the average BST height calculated.

Another relevant relationship is that between the number of hash collisions (in the hash table) compared to average BST height. Hash collisions happen when an element (in our case, a node) is inserted (due to hashing) into the same index as another element. Inserting nodes into the same index (containing a binary search tree) directly increases the height of the tree in that index. So with more hash collisions, the height of the binary

search trees will increase. They have a positive correlation: more hash collisions (in the hash table) entails larger binary search trees (on average).

Another statistic that I wanted to discuss is the load for both the bloom filter and hash table. The loads for the bloom filter and the hash table are percentages that say how full each of these structures are. They are calculated by dividing the number of “elements” in them divided by their size (and multiplying by 100). So, if the load of either of these is larger, then that correlates to more false positives. If more and more indices in the bloom filter and hash table are filled, there is more likely for an element to be inserted into the same index as another. So, a larger load percentage for each of these structures corresponds to more false positives found.