Assignment 7 - The Great Firewall of Santa Cruz

Teresa Joseph

CSE 13S - Professor Long

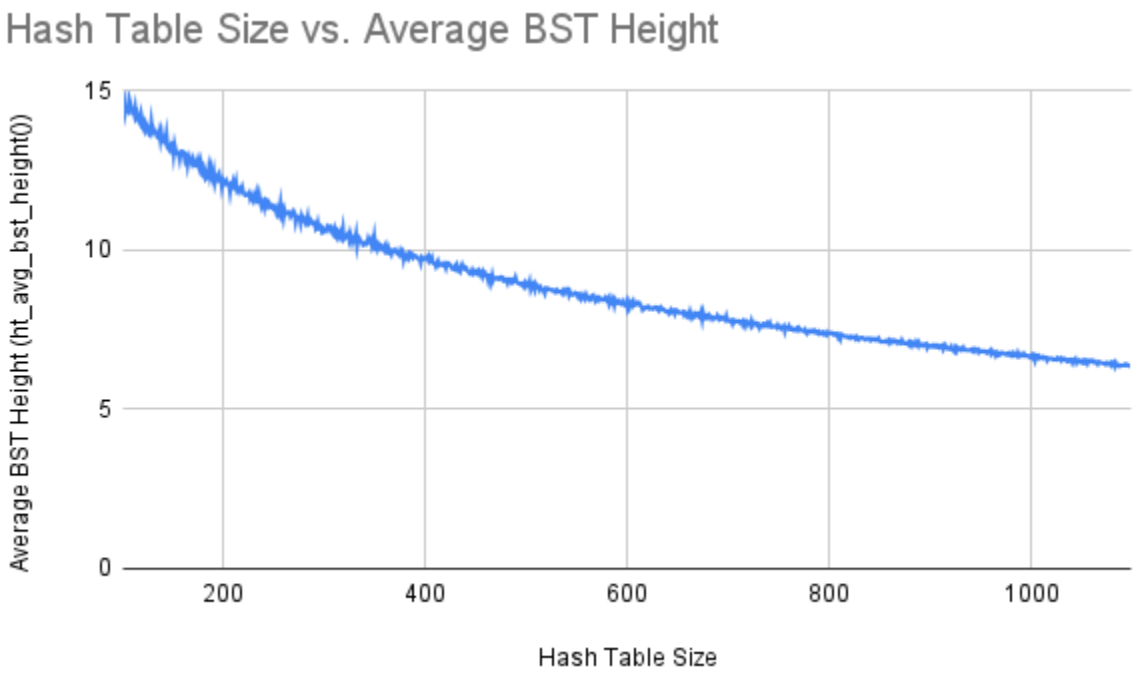
Fall 2021 - December 5

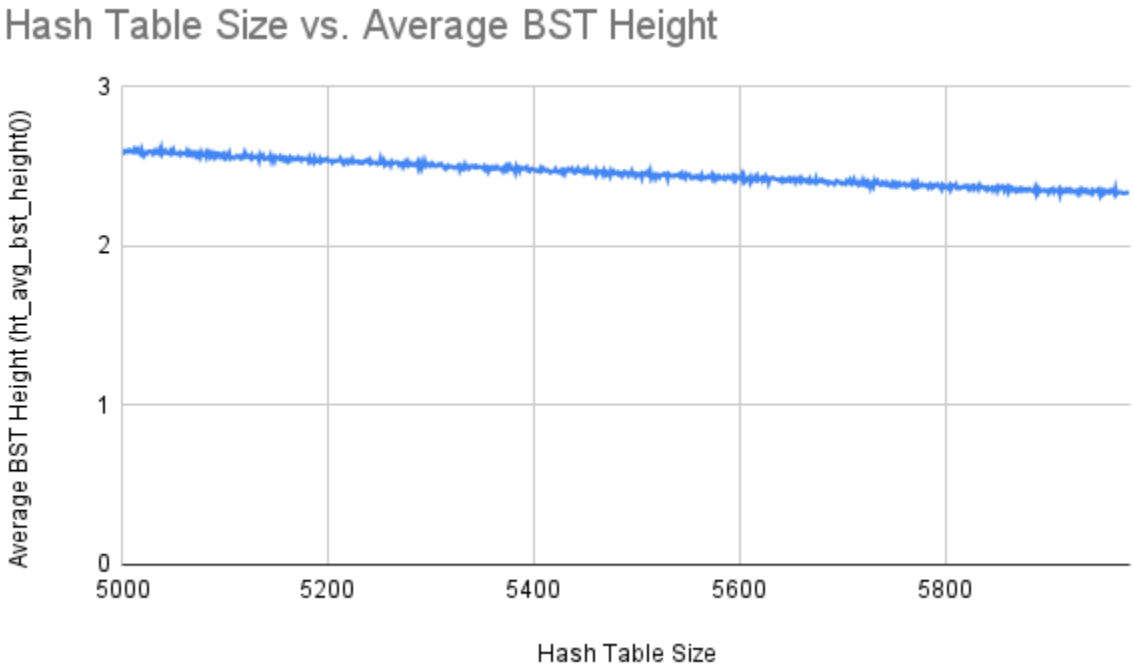
How do the heights of the binary search trees change?

The height of a binary search tree (BST) is the number of "levels" of nodes in said tree. The root of the tree makes up one such level and its children make up the next level, and it continues as such. In this assignment, BSTs are found in the indices of the hash table that I made in banhammer.c. Some of these indices are empty, meaning that there is no BST at that index and you would only find NULL there. To decide which indices have BSTs, I wrote a function that hashes my oldspeak terms and uses its return value to represent an index on the hash table. This function is ht_insert(), found in ht.c. In this function, I call another function named bst_insert(), which creates a new root if there is no pre-existing BST and otherwise adds to a pre-existing BST. This action of adding nodes is what affects the height of a BST, namely by increasing its height if inserting creates a new level. Removing nodes can also affect a BST's height by decreasing the overall number of levels, but the only instance in which I do this is at the end of banhammer.c when I free memory and get ready to end the program. Furthermore, as I will explain below, the size of a hash table and the order in which I insert nodes of different oldspeak can affect height.

What are some factors that can change the height of a binary search tree?

Starting with varying hash table sizes, I have graphed the difference in the average BST height given hash table sizes from 0 to 1100 and then 5000 to 6000 using Google Sheets. In both cases, there are downward trends that show that the average BST height decreases as the hash table size increases.





This change makes sense from a logical perspective. If we were to look at the complement of this, namely that the average BST height increases as the size of the hash table decreases, then we see that hash collisions are more likely to occur. This is because there are less options to assign an oldspeak term to a hash table's BST. This relates to the pigeon-hole principle. With more oldspeak to assign than hash table indices to assign to, it is certain that two oldspeak terms (at the very least) will be assigned the same index. As such, with less indices in general, the likelihood of overlaps, which are known as hash collisions, increase. These hash collisions can cause the height of a BST to increase. If we add multiple oldspeak to the same index and thus the same BST, we would be adding the overall size (and thus height) of the tree.

The trend seen in the graphs tells us exactly this. If we were to think of the two graphs as one, then we could imagine the shape of the curve from hash table sizes 1100 and 5000 to also be decreasing. The overall shape suggests that the curve will level off at some point, which thus suggests that at a very large hash table size, the probability of a hash collision is near 0. This would reduce the average BST height.

Another factor that can affect average BST height is the order in which we insert oldspeak. Adding oldspeak where each oldspeak is lexicographically greater than the oldspeak before it will make a skewed BST. The same happens if each oldspeak is less than the oldspeak before it, skewing the BST to the left instead of the right. This makes the height of the BST rather large. In fact, the height would be the number of oldspeak. A balanced BST, where the left and right subtrees do not significantly differ in height, would have a smaller overall height. Its nodes would not be skewed despite being the same ones as in the other example, and for this reason, it would have a different height.

How does changing the Bloom filter size affect the number of lookups performed in the hash table?

The relationship between bloom filter size and the number of hash table lookups performed has to do with false positive rates. The only reason why a lookup would be conducted is if the bloom filter suggested that it contains a word. However, the bloom filter is not deterministic. It is likely to result in false positives if its bits were set by another word. This is especially likely if the bloom filter has a small size. Hashing an oldspeak three times is required when setting a bloom filter's bit vector, so if it has a small size, multiple words can end up attempting to set the same bit. We would have no way of knowing if this happened or not by only looking at the bloom filter. As such, as the bloom filter size decreases, the number of hash table lookups increases. Its complement is also valid: as the bloom filter size increases, the number of hash table lookups decreases.

When the size of the bloom filter decreases, the probability of a false positive increases. This is because there are less options to assign an oldspeak term to a bloom filter's bit vector. This relates to the pigeon-hole principle. With more oldspeak to assign than bit vector indices to assign to, it is certain that two oldspeaks at the very least will be assigned the same index. As such, with less indices in general, the likelihood for hash collisions increase and