UNIVERSITY OF CALIFORNIA SANTA CRUZ

Analysis - The Great Firewall of Santa Cruz

Student name: *Nicholas Reis*

Course: *CSE13S* – Professor: *Darrell Long*
Due date: *Decmber 5th, 2021*

Introduction

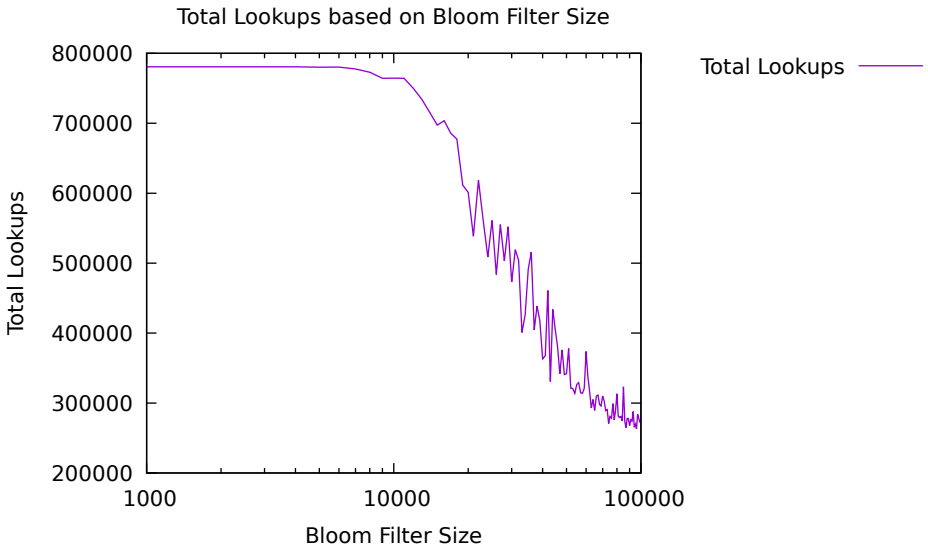
In this write up I will be demonstrating my findings pertaining to the relationship between total number of lookups and average binary search tree branches traversed depending on the sizes of a hash table and bloom filter. I created several graphs to show my findings, altering the sizes of either the hash table or bloom filter in each of them.

It is important to note that if it is a graph depicting a change in hash table size, the bloom filter size will be set to the default value of 2^{20} for consistency. If the graph depicts a change in bloom filter size, the hash table will be set to the default value of 2^{16}. Additionally, each of these graphs is influenced by the text input given by the user, so results will vary slightly depending on the amount of text input. I used a very large file - bible.txt - which contains over 780,000 words in it, so that changes could be more visible over a large data set.

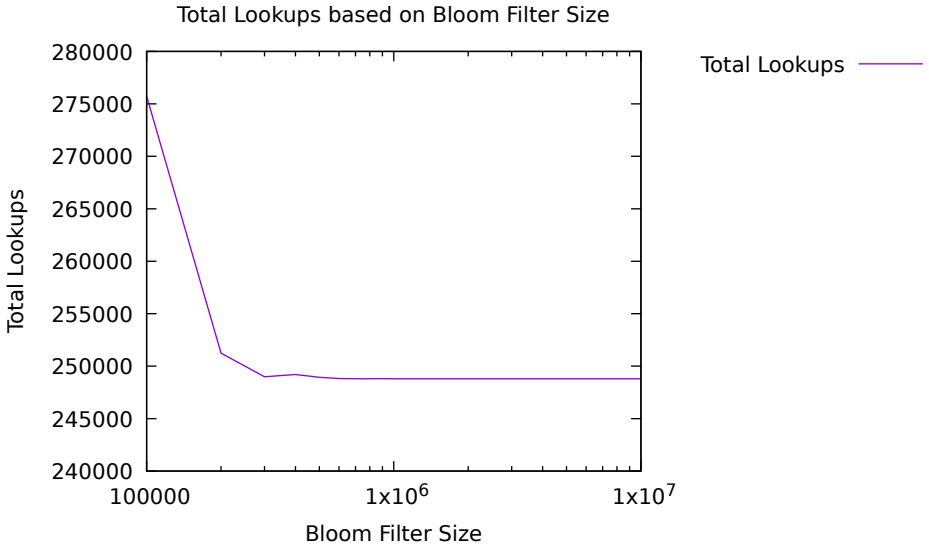
Results

Findings and Graphs for Total Lookups.

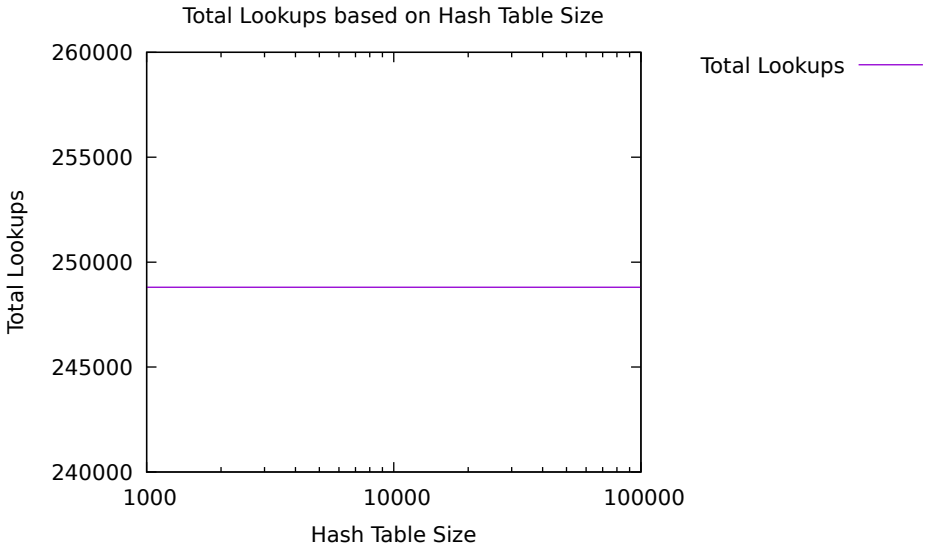
(*a*) In this section, I will evaluate the changes in total number of lookups as the bloom filter size and hash table size is altered.



As you can see in the graph above, the total number of lookups remains contant just below 800,000 when the bloom filter size is below 5,000, before quickly decreasing after that point. Lookups is heavily affected by bloom filter size because of the main purpose of the bloom filter - filtering out words that are definitely not in the hash table. However, if the bloom filter is simply not large enough for a given set of words (e.g. the bible) it will always be full, and thus return false positives for every word and causing the program to resort to using extra lookups to see if a word is actually in the hash table.



Above, we can see the results of a larger bloom filter size on lookups, and notice how there is a point where the number stagnates around 250,000. This is another point to note about bloom filter size - once it becomes large enough (also depends on given set of words), there will be no false positives and the program will have the same number of lookups each time.

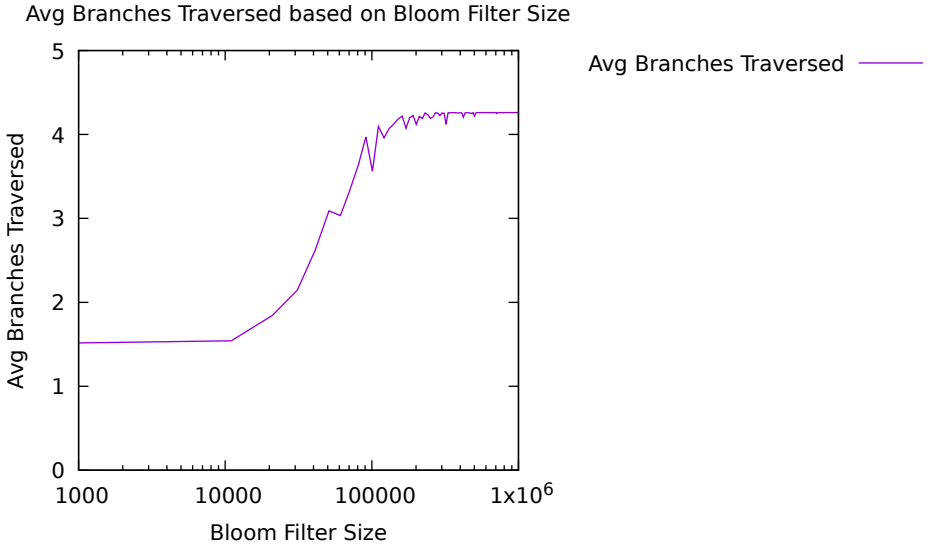


The graph above compares total lookups to hash table size, and we can very clearly see that it has no influence on it. This is because lookups is only incremented

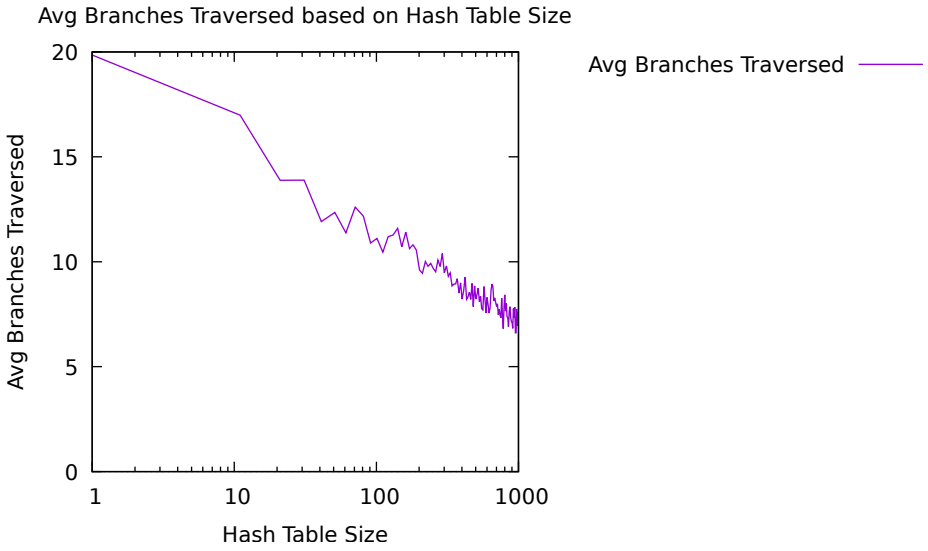
once for each call to either the ht_insert() or ht_lookup() functions. From the previous graphs, we know that if the bloom filter size is large enough (which it is in this case), the number of lookups will remain the same, as the amount of times ht_insert() and ht_lookup() are called only changes if there are more bloom filter false positives.

Findings and Graphs for Average BST Branches Traversed.

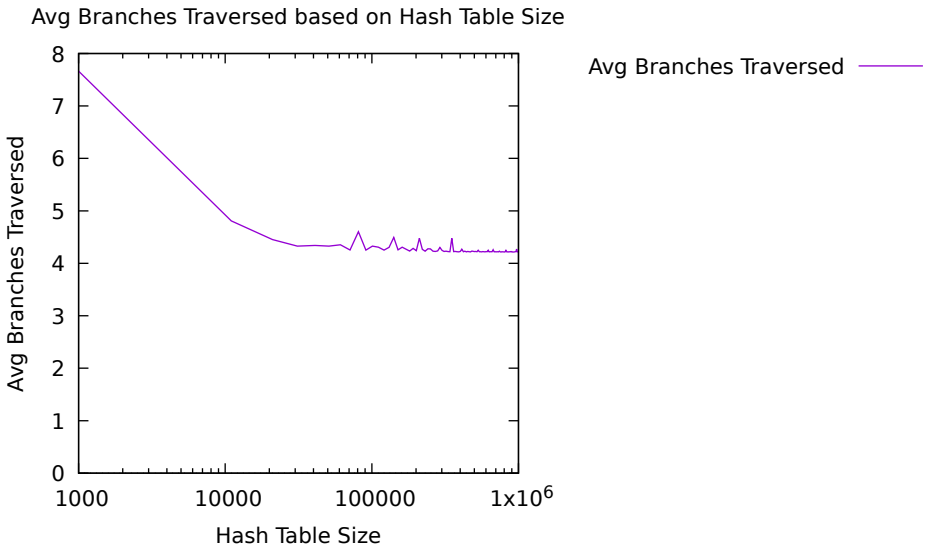
(*a*) In this section, I will evaluate the changes in average binary search tree (BST) branches traversed as the bloom filter size and hash table size is altered. Average BST branches traversed is calculated as follows: total branches traversed / total lookups



The graph above tells us a few things about how bloom filter size affects average branches traversed. First, notice that the as the bloom filter grows larger, more branches are traversed on average. This is odd, as you would assume that we would have to traverse less branches on average if this were the case. The reasoning for this lays in the formula for calculating average BST branches traversed - total branches traversed / total lookups. Recall the graphs above and notice that total lookups is largest when bloom filter size is small, and decreases as bloom filter size becomes larger. As total lookups is the number in the denominator, when it is a larger number, the calculated average branches traversed will actually be a smaller number. Despite the fact that more total branches are traversed with a small bloom filter size, lookups is such a large number that this is simply how the math works out.



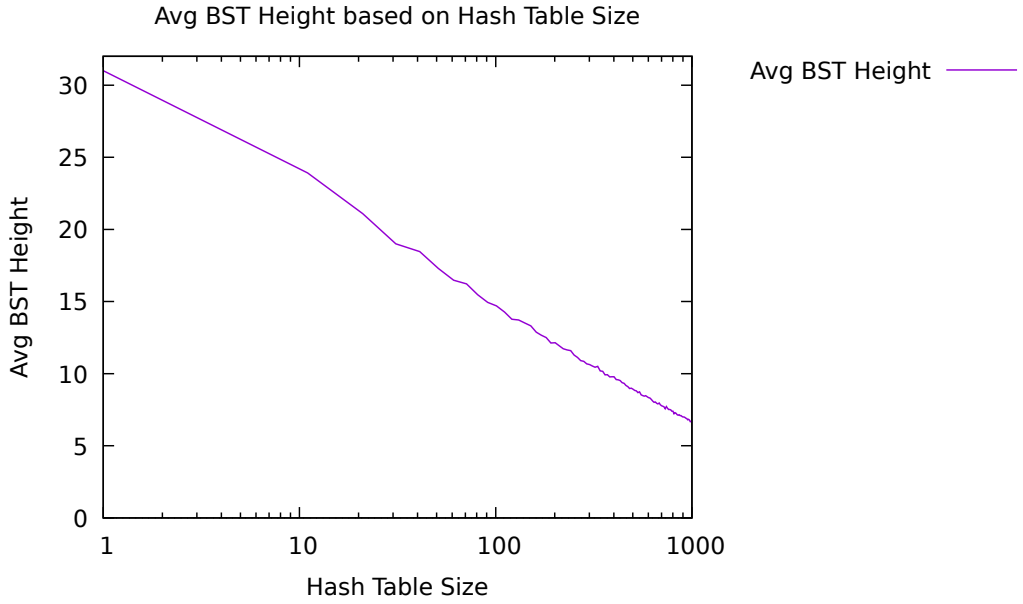
Above, it is a clear trend that average branches traversed mostly decreases over between hash table sizes of 1 and 1,000. The reasoning behind this is that with a small hash table size, there is a much higher chance that there will be hash collisions. This means that two words output the same value when hashed, and thus go into the same binary search tree, making the tree taller, causing there to be more branches traversed to find the word in the tree.



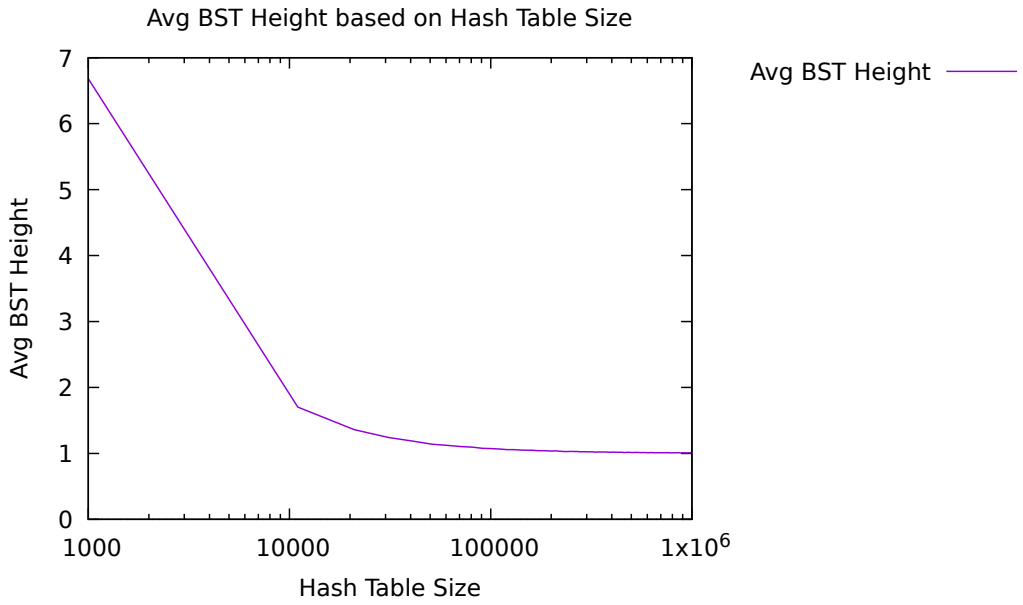
We see the same logic with larger hash table sizes, however, similarly to bloom filter size and total lookups, there becomes a point where the hash table is large enough such that every word has a different hashed value. This causes each binary search tree to be a height of one, and thus causes there to be a very low number of total branches traversed. The following section may help for more visualization of the tree heights in the two above graphs.

Findings and Graphs for Average BST Height.

(*a*) In this section, I will evaluate the changes in average binary search tree height as the hash table size is altered.



If you refer to the graph two above this one, you can see a direct correlation to binary search tree height and average branches traversed with different hash table sizes. As I mentioned before, the more hash collisions there are, the larger trees will be, and the graph shows this very conclusion.



Once again, if you refer to the graph two above this one, you can see the correlation once again. And as I said there, the tree height will eventually trend to 1 once the hash table is so large that there are no hash collisions.

Conclusions

There are many conclusions that I have been able to make in my analysis of each of the above factors in this program. First, I determined that bloom filter size has a strong effect on the total number of lookups performed. If the bloom filter size is too small for a given set of words, the total number of lookups will be at a maximum value and remain constant until the bloom filter is not completely filled up. Once the bloom filter has some bits that are zero, the total lookups will decrease, as there are less false positives produced from checking if a word is in the bloom filter. At a large enough size depending on the given set of words, the total lookups will hit a minumum value, where checking the bloom filter produces no false positives and the total lookups is only what is absolutely required in the program. I also noticed that hash table size does not effect the total number of lookups.

I found that average binary search tree branches traversed increases as the bloom filter size increases. This is primarily because of the way average branches traversed is calculated, with total lookups in the denominator, and as I just mentioned, total lookups decreases when bloom filter size increases. This causes the calculated number to be larger, until the bloom filter is large enough that the lookups does not change, and neither does the average branches traversed. Hash table size also influences average branches traversed. We saw in the graphs that the larger the hash table was, there were less average branches traversed. This has to do with hash collisions, meaning that two words produce the same value when hashed, and thus are inserted into the same binary search tree in the hash table. This causes the binary search trees to become taller, in turn requiring more branch traversals to find or insert more words. As the hash table gets larger, it avoids these collisions more often, and eventually becomes large enough such that there are no collisions and the average number of branches traversed will be at its lowest point.

To corroborate this conclusion, I created graphs depicting the average binary search tree height as the hash table size changed, and the results were as expected. The average binary search tree height decreases as hash table size increases, until it reaches a value of 1. When there are no hash collisions, this will be the case, and it means that if there is a call to find a word in a binary search tree, it will not have to traverse multiple branches to find it.