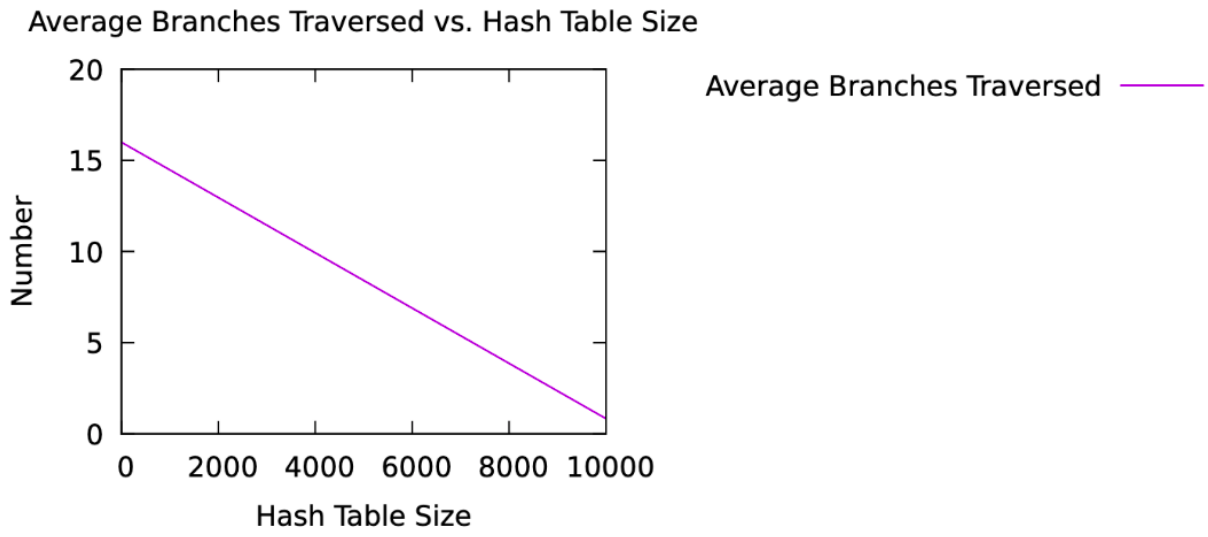
Diwa Ashwini Vittala
CSE 13S - Fall 2021

⁰❀ Assignment 7: Writeup ❀。

THE GREAT FIREWALL OF SANTA CRUZ

Branches: Calculated as the number of recursive calls taken during bst_find() and bst_insert().
Lookups: The number of times that ht_lookups() and ht_insert() are called.
Average Branches Traversed: Calculated by branches / lookups.
File used for graphing:
Text file with Ep 1 of Phineas and Ferb
Link: https://phineasandferb.fandom.com/wiki/Rollercoaster/Transcript
Since I used a script from Phineas and Ferb's first episode to create these graphs, there were many words in badspeak.txt (such as dr, files, and answer) as well as newspeak.txt (such as brother, great, money, and school).

Hash Table Size in comparison to Average Branches Traversed

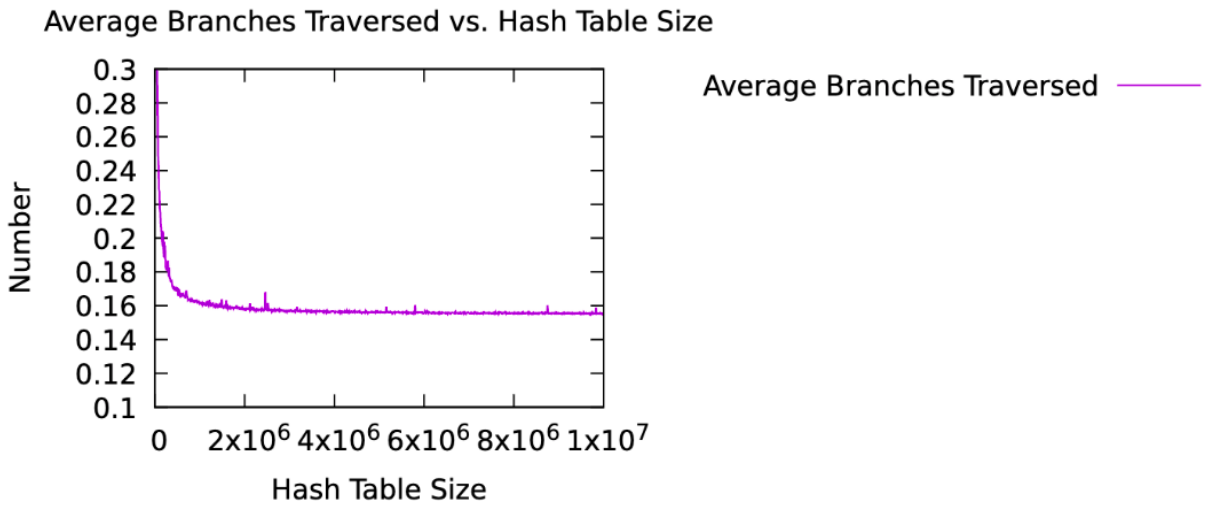


As the size of the hash table increases, the average number of branches traversed decreases, which means either the number of branches decreases or the number of lookups increases. Exploring the way that I have implemented the functions will help answer which of these two options is likely happening in the graph above.

A hash table is built using an array of nodes, where each node is the root of a tree. The downward slope of this graph indicates that there are less branches traversed on average as the size of the hash table increases. This makes sense because when the size of the hash table is smaller, there are less trees (with more branches, and larger height) and therefore the number of branches traversed are larger. The number of lookups, as will be discussed in "Hash Table Size in comparison to Lookups", does not change. The number of lookups is constant and the number of branches continues to decrease, therefore the average number of branches traversed

(branches/lookups) is decreasing as the hash table size grows (which is consistent with the graph above).

Hash Table Size in comparison to Average Branches Traversed (zoomed out)



The same graph, when zoomed out as the hash table size becomes extremely large (1x10^7) but with a smaller y-axis range, shows more information about the behavior of the average branches traversed. If more "badspeak" words are found, the number of branches increases because I use bst_insert to add bad words to the trees called "bad words list with newspeak" and "bad words list". Regardless of whether the node exists or not, after probing the bloom filter for a word, I use ht_lookup to see if the word is in the hash table, which means it is likely to have a larger number of lookups than branches (hence the downward trend).

Unlike the conclusion that could have been derived from "Hash Table Size in comparison to Average Branches Traversed", we can see that the average number of branches traversed does not approach zero as the hash table size increases. In fact, it approaches a value of about 0.16, because in the formula branches/lookups, the value for branches is never actually 0 - we insert oldspeak and newspeak values to the hash table regardless of what we read.

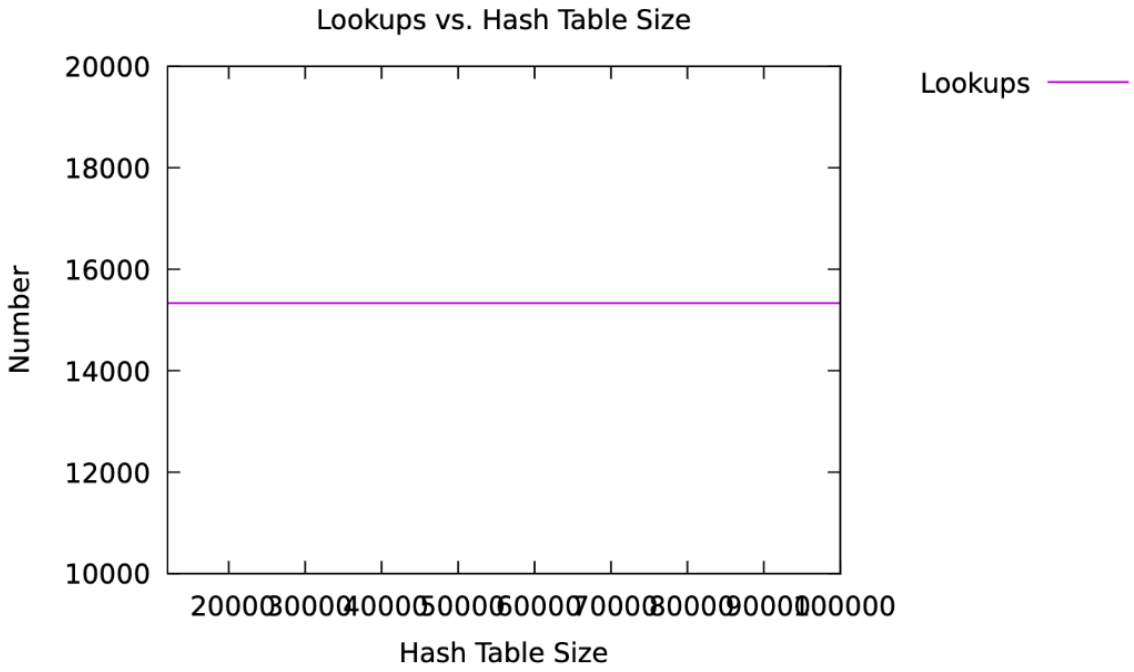
Another conclusion we can draw from these graphs is that it takes more lookups for a smaller hash table. The hash table consists of root nodes, so when there are less trees, it takes more calls to bst_find in order to find something in the large tree (the height of the tree is larger), and more calls to bst_insert to add something.

Hash Table Size in comparison to Lookups

The number of lookups depends on the number of times that ht_lookups and ht_insert is called. These functions are called after probing the bloom filter and while adding words from "badspeak" and "newspeak" to the hash table. In the case that only the table size is being changed, the graph for lookups would look like graph #1. In the case that both the bloom filter

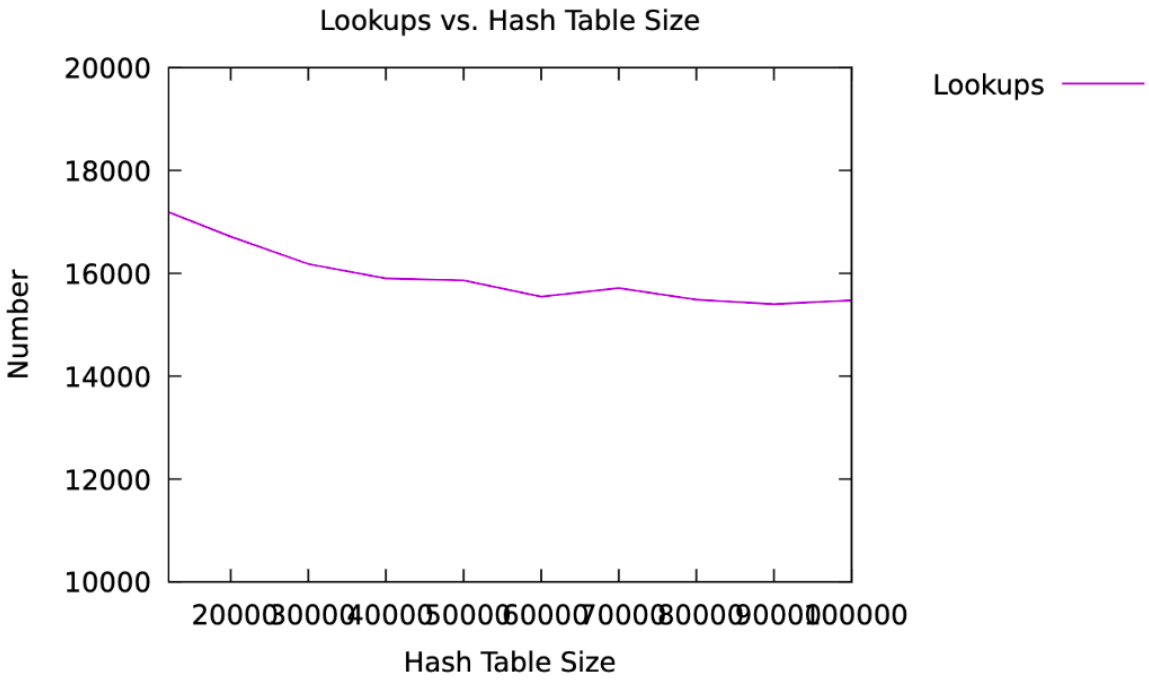
size and the hash table size are changing at the same rate, the graph for lookups would look for like graph #2.

GRAPH #1



The number of lookups do not change too drastically over larger hash table sizes (or they don't change at all in the case of graph #1) because unlike branches, there are no recursive calls to ht_lookups and ht_insert used, therefore they depend more on the file that is being used, and the type of text inside that file (the number of words that are oldspeak and newspeak). I kept the text file as a constant to only observe how the number of lookups reacts to a change in the hash table, and the result was a flat value between 14000 and 16000 lookups because the number of words that need to be looked up in ht_lookups() and the number of words inserted using ht_insert() are constant.

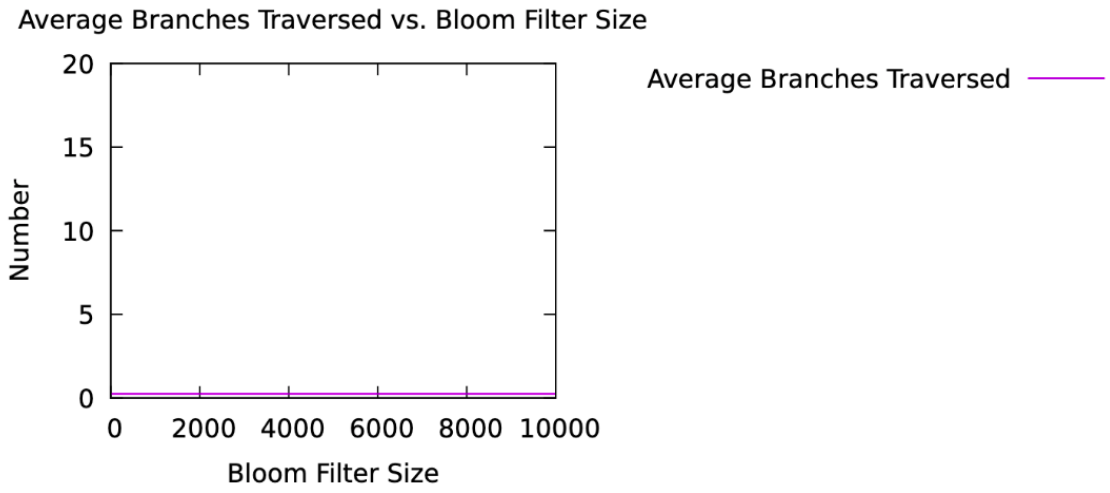
GRAPH #2



To reconfirm that the hash table size did not change the number of lookups, I created graph #2, changing the bloom filter size just like I change the hash table size. The result is a graph that is non-linear, showing that the change in the number of lookups is connected to the bloom filter rather than the hash table. This makes sense because the number of lookups is calculated by the number of calls to ht_lookups() and ht_insert(), which are more reliant on the text itself rather than the size of the hash table which will hold the strings.

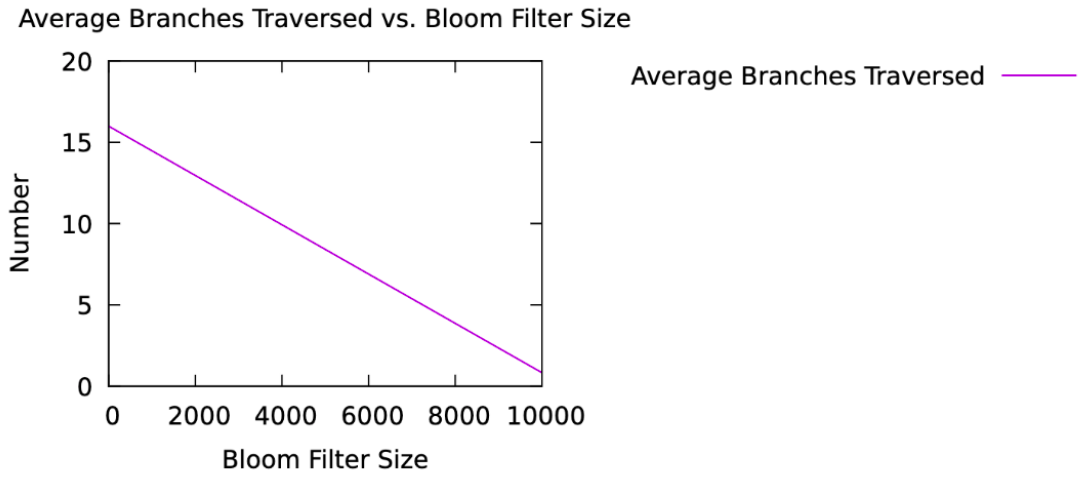
Bloom Filter Size in comparison to Average Branches Traversed

GRAPH #1



Changing only the bloom filter size, the graph comparing the average number of branches traversed to the size of the bloom filter is close to 0, and flat as in graph #1. As explored in "Hash Table Size in comparison to Lookups", the number of lookups likely depends on the size of the bloom filter. In the same way, the average number of branches traversed depends on the hash table size and does not change as a result of a change in the bloom filter size (hence the flat line).

GRAPH #2

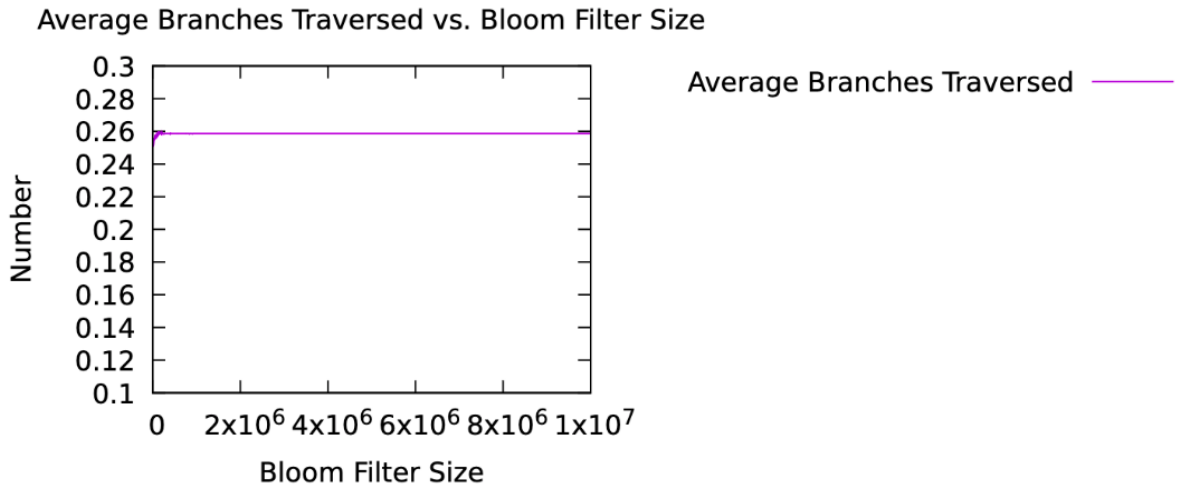


When I set the hash table size in the same way I set the bloom filter size (graph #2), we find the graph linear and downward sloping as they were in the hash table comparison graph. A bloom filter is built using an array of bit vectors, and I use a bloom filter to check if a given word is likely to be oldspeak. The value for branches depends on calls to bst_find() and bst_insert(), which reference nodes instead of bit vectors, which is why changing the bitvector-dependent component does not change the node-dependent components.

Bloom Filter Size in comparison to Average Branches Traversed (zoomed out)

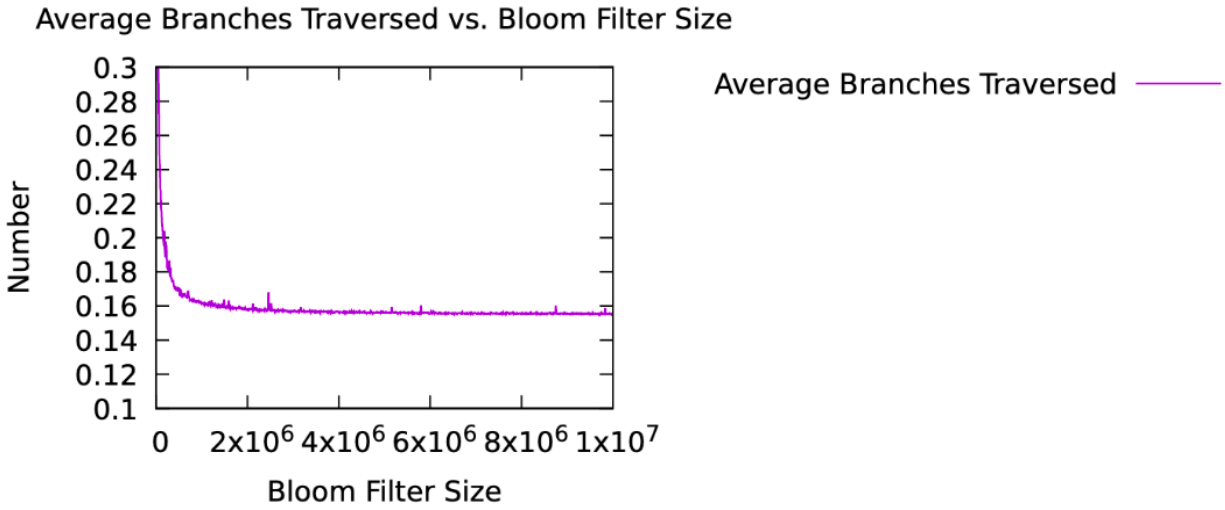
When zoomed in on the y-axis and looking at larger values for the bloom filter size, we can see that the bloom filter behaves quite differently when I also change the hash table size (graph #2) in comparison to when I only change the bloom filter size (graph #1).

GRAPH #1



When the bloom filter size is the only thing that I am changing, graph #1 is produced. Although there is a slight curve in the beginning (due to 0 not being used as a possible input size), the graph is linear. The conclusions made in "Bloom Filter Size in comparison to Average Branches Traversed" still hold, however we see that instead of staying at 0, this fixed value is about 0.26 average branches traversed.

GRAPH #2

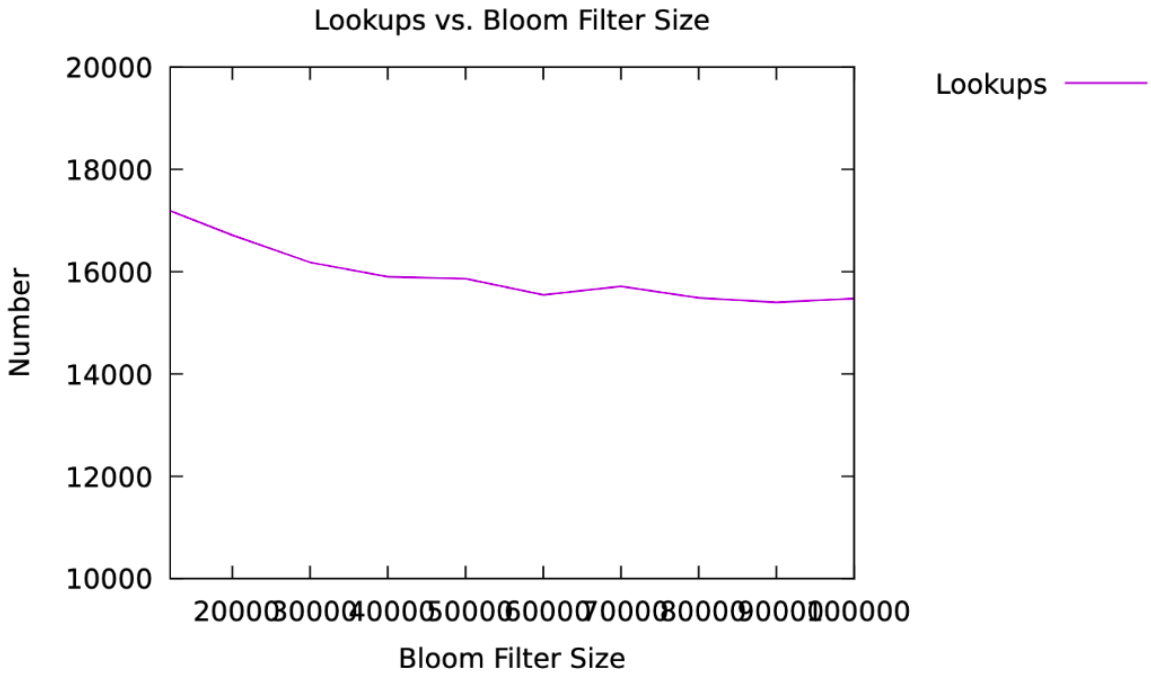


Just to reconfirm the idea that the bloom filter size is not what changes the average branches traversed value, I also created graph #2 which changes the value of the hash table size in the

same way that I change the bloom filter size. Now, we see a drastic change in the average branches traversed, similar to the impact it had on the hash table previously. The conclusion that the bloom filter size does not directly change the average branches traversed still holds.

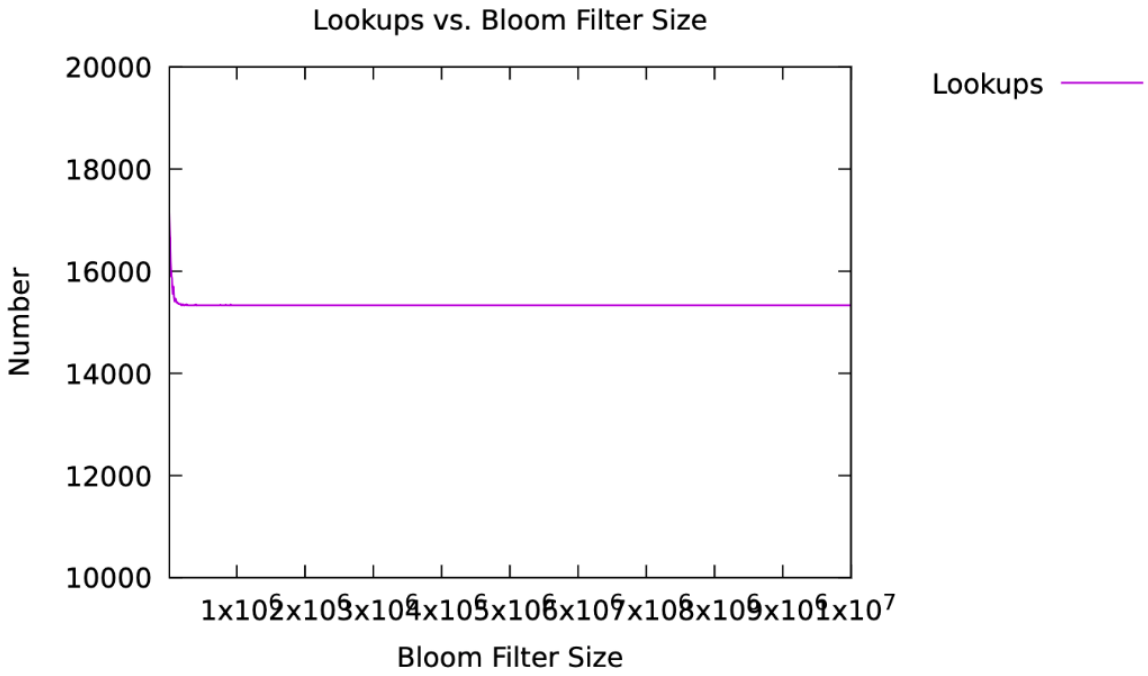
Bloom Filter Size in comparison to Lookups

GRAPH #1



The number of lookups is the number of times that ht_lookups() and ht_insert() are called. There is a small variation in values because when the bloom filter is smaller, it takes more tries to find a value using bf_probe, therefore the number of lookups (and insert, due to similar function structures) is initially inconsistent. A possible explanation for this is that lookups only increments relative to the text that is given to banhammer, and since I tested with only one text file, the number of lookups is also constant.

GRAPH #2



However, when we zoom out and see the values for the number of lookups in comparison to the bloom filter size, we see that the value is a constant number between 14000 and 16000. In opposition to the findings from “Hash Table Size in comparison to Lookups”, it seems that the number of lookups is still constant over large numbers when we change the bloom filter size. The significant changes in lookups only occur when the bloom filter size is smaller than ~100000. This must mean that the number of times that bst_insert() and bst_find() are called is proportional to the number of times ht_lookups() and ht_insert() are called.

Summary of analysis:

01. The height of the binary search trees are large when the hash table is small. The height of the binary search tree is smaller when the hash table size increases.
02. More branches are traversed when the hash table size is small.
03. The average number of branches traversed changes significantly when the hash table size changes.
04. The number of lookups initially changes when the bloom filter size changes, however the value eventually reaches some constant between 14000 and 16000 lookups. This value is the same for when only the hash table size changes, and when only the bloom filter size changes.