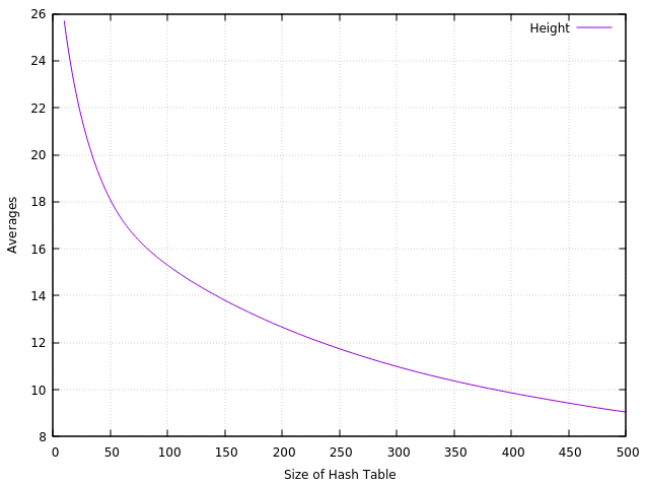
WRITEUP - Assignment 7 - The Great Firewall of Santa Cruz

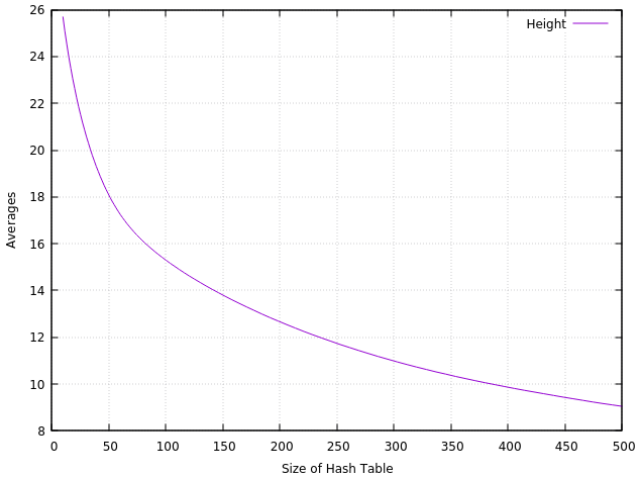
In this writeup, the properties of a hash table and a bloom filter are compared and analyzed, when the size of the hash table is changed. As the size of the two data structures increase, the chance for false positives decrease in the Bloom filter, and the hash values are more uniformly distributed, but before that is further discussed, it's important to first understand what's happening:

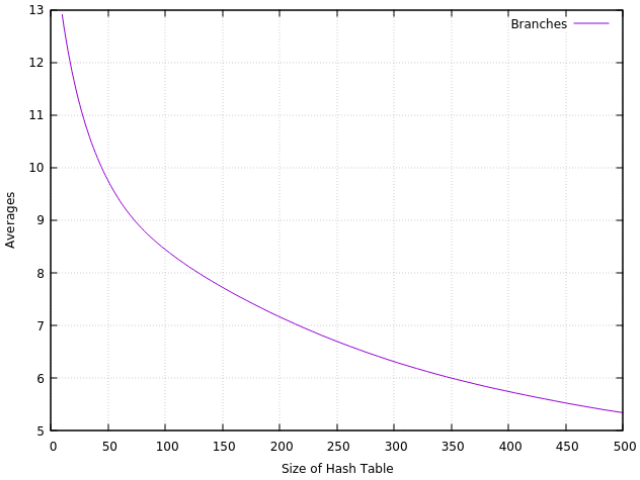
```
salamander1223@Vivobook1223:~/cse13s/asgn7$ ./banhammer -s
Hello, what a lovely day!
Average BST size: 1.121552
Average BST height: 1.118038
Average branches traversed: 1.155235
Hash table load: 19.972229%
Bloom filter load: 4.082680%
salamander1223@Vivobook1223:~/cse13s/asgn7$
```

The banhammer function takes in text from standard input, parses through the words using regular expressions, and checks them to see if they're within the hash table of nodes, previously filled from a separate text file. The process for which they were checked involved hashing the word to see if it was in the Bloom filter, and if so, checking once more to see if it was within the hash table.

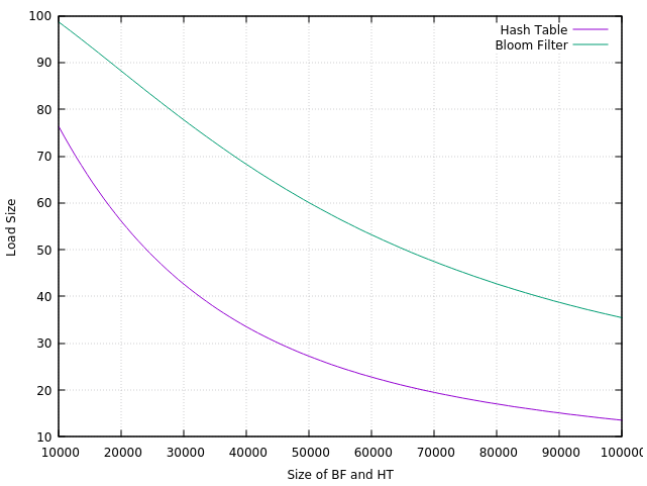
So why does the size of the Bloom filter and hash table matter? In this particular case, the program would not fail if the Bloom filter was much too small to indicate any use, at least one byte, as any hash collisions are resolved with a binary search tree at each index; the existence of the Bloom filter is merely an exercise in functionality. However, allow me to expand this problem to a much greater scale, such that it's more comparable to the real world: what if the hash table was much bigger than the contents of a single text file? What if there was no bloom filter and only a hash table, or vice versa? In these particular cases, even if you were to hash over a small table, you would still have to deal







with the possibility of an otherwise sizable binary tree. In a hash table, you take either an address, or some abstract number, and you perform an operation on it that will give you a random enough index to which to place your data. The properties of a good hash function are that two values that are otherwise very similar can be placed at very different indices. This allows for a massive amount of data to be stored, and to be accessed in constant time. However, the size of the hash table is limited, and to store the index, a modulus operation would have to be performed, limiting the index to a certain range. Because of this, hash collisions can occur, when two different kinds of data can have the same address. To resolve this, one can use a binary tree, so that each index can store an unbounded number of nodes, if need be. In the above graphs, one can see that as the size of the hash table increases, the average of the size and branches of the individual binary search trees decreases, indicating that there were less and less hash collisions.



As for the Bloom filter, the logic for the size scaling with the efficiency is similar to that of the hash table. The Bloom filter does not contain nodes of data that can be checked, but rather, each set of data corresponds to three separate indices in a bit vector that are set. If these bits are set, then there is a strong likelihood that the value exists in the set, otherwise it's a false positive. If and when a Bloom filter tries to read in too many different

values, then too many of the bits can be set, and the possibility of a false positive skyrockets, and if every bit was set, then the Bloom filter would never work as intended. That's why the Bloom filter and the hash table work in tandem, so that the size of the filter assures a low rate of false positives, and just to be certain, the hash table is checked, and found rather quickly.

Credit for the code provided and other helpful information goes to Professor Long and the staff of CSE-13S in the Fall 2021 quarter.