Asgn 7 Writeup

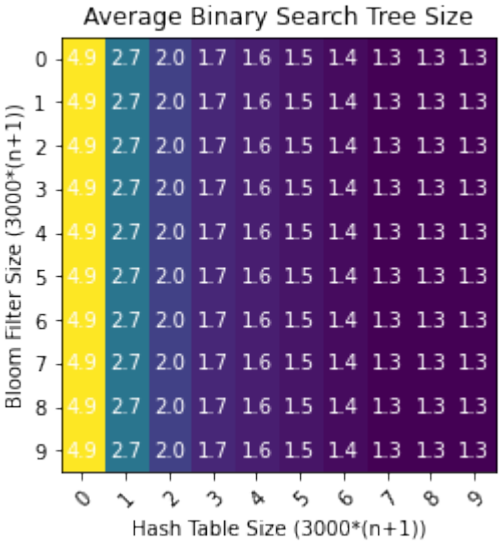
Brandon Chuang

12/4/2021

1 Introduction

This is a writeup on the text filterer. You'll find that many of the statistics here are only affected by one of two manageable values: either the hash table size or the bloom filter size. All these tests are determined by filtering the Declaration of Independence.

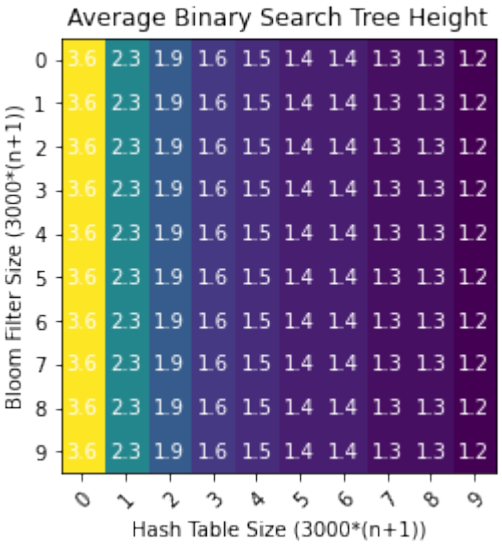
2 Average Binary Search Tree Size



You can see by this heatmap that the bloom filter's size does not affect the average binary search tree(BST)'s size. The only determining factor of the BST's sizes is the hash table size. You'll notice that ther average BST size gets smaller as the hash table size. This is because the hash table size dictates how many different BST's a node can go to, and the less BST's there are means more nodes have to be packed into a singular tree.
It is important to note that each node that isn't a root is considered a hash

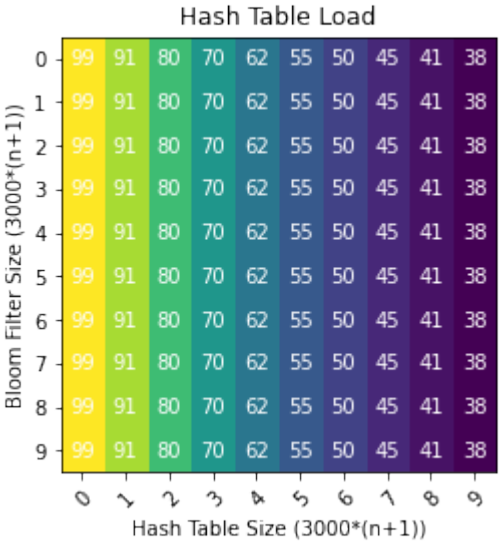
collision. These trees are made to support multiple nodes to make hash collisions not fatal and not impact the accuracy. You'll notice that the average binary search tree size's trend is logarithmic, since the curve gets steeper as it approaches 0.

3 Average Binary Search Tree Height



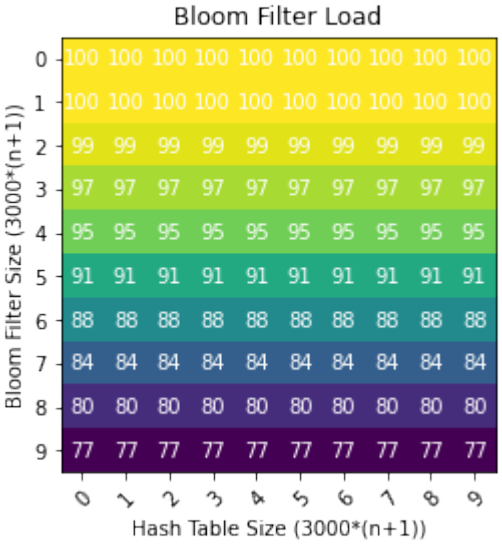
The bloom filter size also has no affect on the BST's height. The height is affected by the size. a completely binary tree can will have a height of the sum of 2^n from zero to n, but that is assuming everything is perfectly balanced. This is why the height seems to get larger faster as the hash table size approaches 0, much like the size, but does not increase at the same rate.

4 Hash Table Load/Hash Collisions



The hash table load is the number of non null binary trees divided by the amount of nodes in all the trees. This number represents what percent of nodes are roots, meaning that the percentage goes down as the hash table size decreases. When more nodes than there are binary search trees are present, at least one node has to not be a root (as per the pidgeonhole principle).
Each node that is not a root is a hash collision, because that means both share the same hash value. Therefore, the percentage that the load represents is the likelyhood of a node being a hash collision. The hash colissions do not affect the accuracy of the program as a hole. Rather, it dampents the efficiency, because it will have to search binary trees to check for badspeak/oldspeak rather than knowing based just off of the hash value. The trend here is also logarithmic.

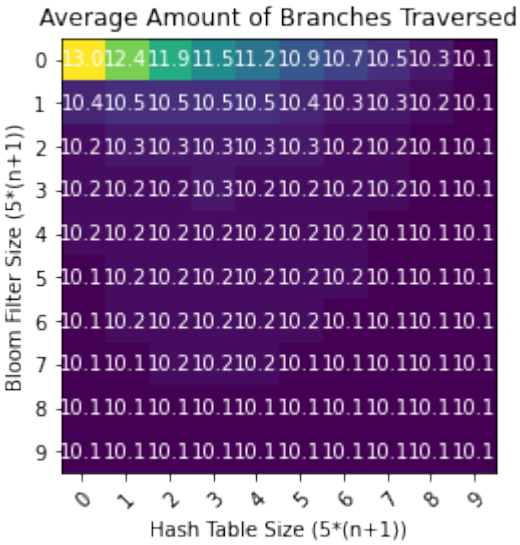
5 Bloom Filter Load/False Positives



Much like the hash table equivalent, this number is also not affected by the hash table's size. It is the number of set bits in the bloom filter divided by the number of bits that the bloom filter has access to, which means that the load is the percentage of bits that are marked

Because more bits being set means that your bloom filter is getting full, The higher percentage the bloom filter load is, the more likely you are to get a false positive. For example, if the bloom filter load was 100%, that would mean that every single bit in the bloom filter is set. If every bit is set, it is not possible for the bloom filter believe that a word isn't badspeak or oldspeak. Of course, this does not affect the accuracy of the program as a whole, rather, it dampens the efficiency, because it will have to check the hash table more often.

6 Average Number of Branches Traversed



This is the only value that is affected by both the bloom filter size and the hash table size. I think this value marks the efficiency of the program, where the smaller the number means the more efficient it is. You'll notice that the bloom filter only very significantly affects the value if it is near zero. This is most likely due to all the efficiency-dampening variables being logarithmic, where the efficiency spikes down as the size approaches 0. It looks like the hash table has more of an impact on efficiency than the bloom filter does.