Assignment 7: The Great Firewall of Santa Cruz - Writeup

Thadeus Ballmer

December 2021

1 Introduction

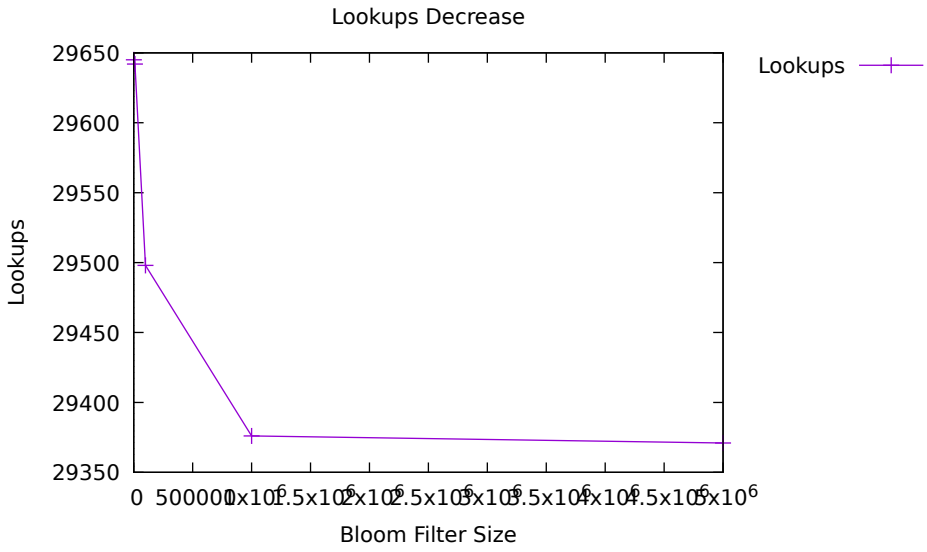
In assignment 7 a text censorship and correction program was implemented. The program operates given a list of *badspeak*, words which carry severe punishment if used, and a list of *oldspeak*, outdated terms, which must be replaced with a list of *newspeak*. Two major data structures made this program possible: a Bloom Filter and a hash table.

2 Bloom Filer

A Bloom Filter is a distinct implementation of a bit vector. A *badspeak* word is hashed with three unique hashing salts then the bit at the index of each hash value is set. To check if a word is probably on the list, simply hash the word with the same salts and check if the bits are all set. A Bloom Filter is not deterministic however because hash collisions are possible. This is why a deterministic hash table is needed. But why use a Bloom Filter if a hash table will determine if a word is truly in the list? Traversing a hash table of binary search trees is costly, but probing the state of three bits is cheap. The Bloom Filter is only deterministic when checking if a word is definitely *not* on the list. This primary check saves much computing time. But by how much?

2.1 Lookups

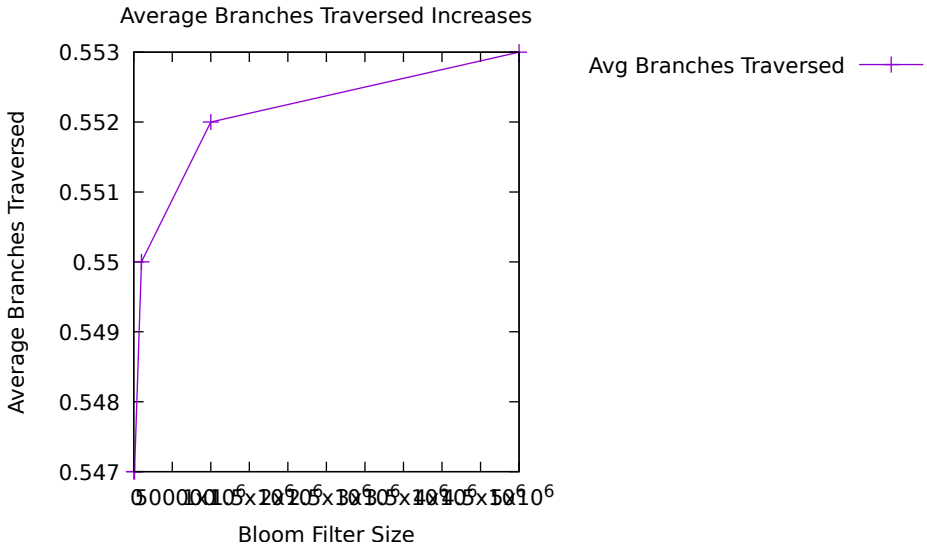
The `lookups` total is defined as the number of times a hash table is probed or has a node inserted. The number of times a node is inserted is constant in this program because the list of words is constant. A node is only inserted to fill the list, but then is probed to check if a word is on that list. So the only factor to change the lookup count as the bloom filter size changes is the probe function.



This graph depicts that as the size of the Bloom Filter increases, the number of probes into the hash table decreases. With more spaces in the bloom filter, there are less chances for hash collisions which means less false positives. Less false positives means that there are less probes needed to check if the word is definitely *badspeak*. Even though `lookups` does not decrease comparatively as much as the size increases, this fundamental operation of a bloom filter is important to understand.

2.2 Average Branches Traversed

The average branches traversed is calculated as `lookups` / `branches`. Branches is the number of links traversed through the binary search trees implemented in the hash table. A link may be traversed when inserting a node into a BST or when probing it.



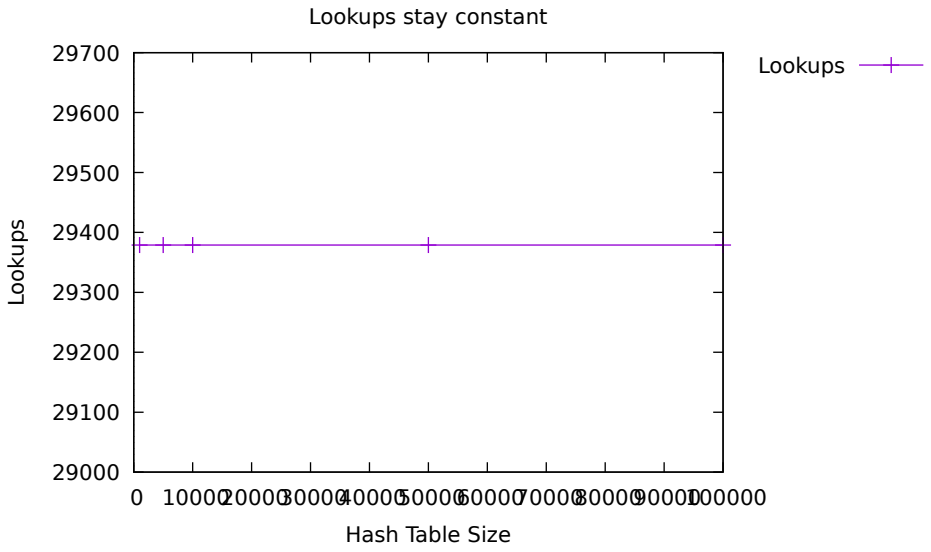
If `lookups` decreases as the bloom filter size increases, then branches must decrease at a faster rate than `lookups` decreases. Why would branches decrease as the bloom filter size increases? Inserting a node into the hash table is really just inserting a node into the correct binary search tree. This means that just like `lookups`, the number of times a node is inserted stays consistent. So probing into the BST is the only factor to change branches. The total number of probes decreases as the bloom filter size increases because less collisions occur. For the same reason that `lookups` decreases, average branches traversed increases.

3 Hash Table

The hash table is made of binary search trees. Each node has *oldspeak* and its *newspeak* translation if applicable.

3.1 Lookups

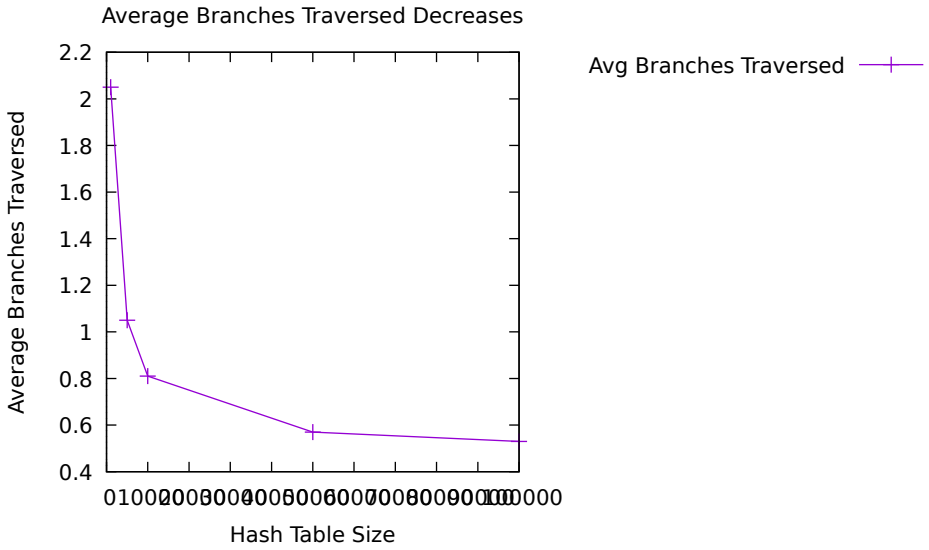
`Lookups` have the same definition from the bloom filter application. However, this analysis will center around how `lookups` changes when the hash table size changes.



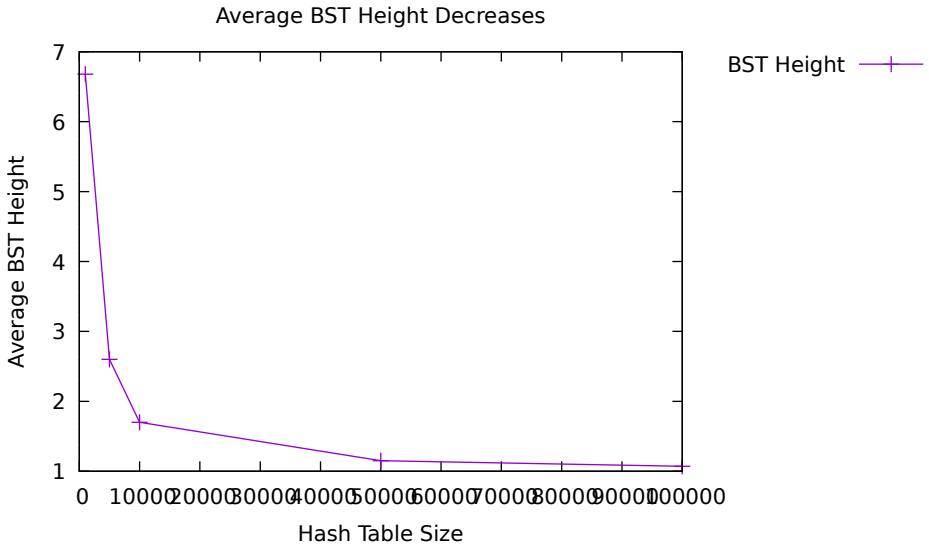
Why would `lookups` stay constant when the hash table size changes? As with the bloom filter, the only factor that can change `lookups` is the number of times the hash table is probed. The hash table is only probed when the bloom filter flags a word as probably *badspeak*. Thus, probing the hash table is unrelated to the hash table size. Therefore, `lookups` stays constant when the hash table size changes.

3.2 Average Branches Traversed

The average branches traversed keeps its same definition from earlier. However, this analysis will center around how average branches traversed changes when the hash table size changes.



As the hash table size increases the average branches traversed decreases. With a larger hash table size, there is more room for nodes to be inserted into the table. This means that that the average height of each binary search tree is smaller.



With a smaller average BST size, when a word is probed it traverses through less links because the tree its traversing is smaller.

4 Conclusion

Although it may be seen that allocating larger sized data structures may decrease certain computations, the effects on computation speed are trivial (in this sized program). With larger data sets the effects would be more pronounced.