Analysis - Assignment 7: The Great Firewall of Santa Cruz:
Bloom Filters, Linked Lists, Binary Trees, and Hash Tables

James Gu

Fall 2021

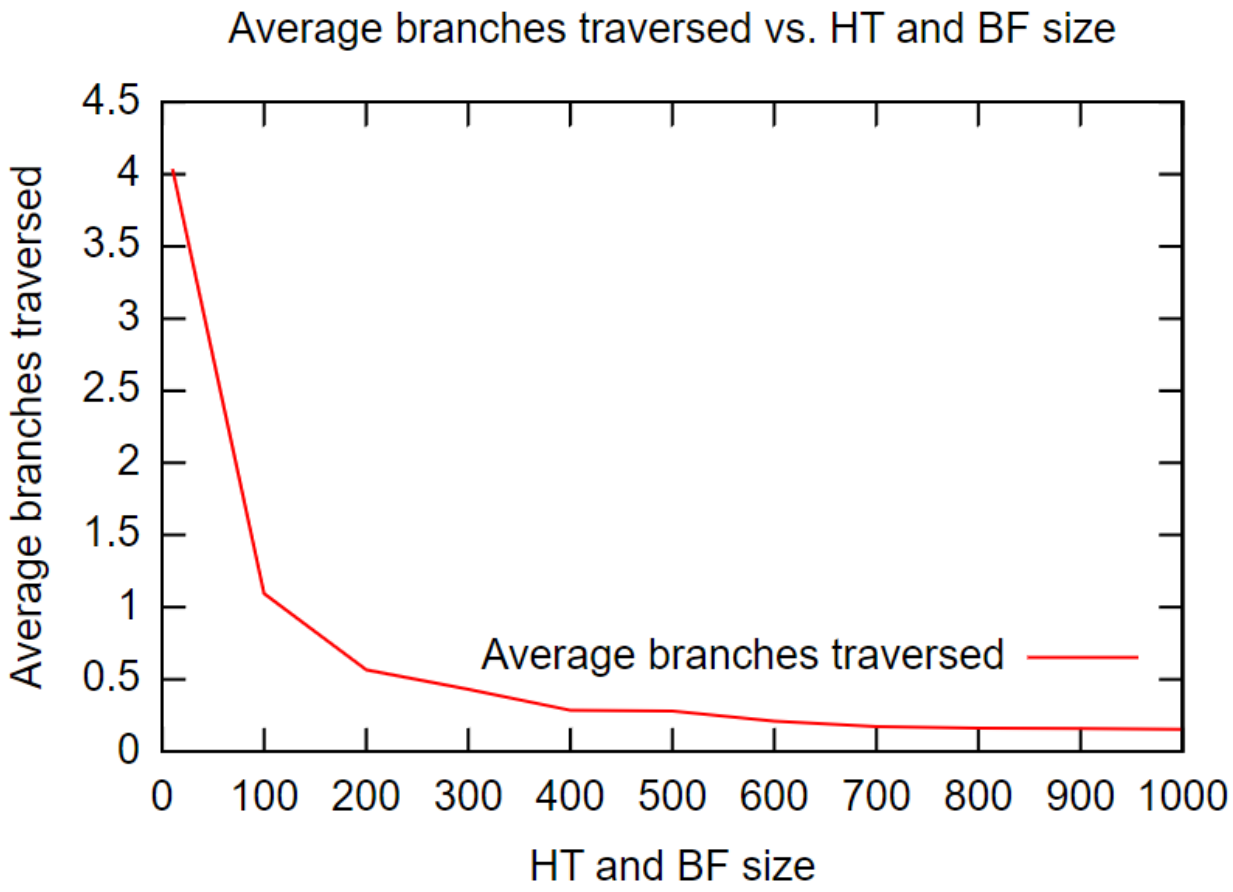
1. Introduction

You have been selected to be the Dear and Beloved Leader of the Glorious People's Republic of Santa Cruz. As the new leader your goal is to censor and filter Internet content as to avoid corrupting young children's minds. This project involves censoring "badthink" and replacing "oldspeak" with newspeak. This will be done using a Bloom filter, Hash Tables, and Binary Trees. These are very prevalent computer science concepts that we will be practicing in this assignment. A Bloom Filter is a space-efficient and probability based data structure and is used to test if an element is within the set. A Hash Table is a data structure that maps keys to values and will store the oldspeak to newspeak translations. Binary Trees involve several nodes that are branches off other nodes like a tree.

2. Analysis

bf.c:

The file bf.c involved the implementation of the Bloom Filter data structure. How this bloom filter works is that it is represented by an array of bits, or a bit vector which we will discuss in more detail in bv.c. Mapping elements involves hashing them which is a pseudorandom way to get an index for the element. This index is actually three different bits that are set to 1. No matter what, the hashing of the same element will result in the same random indices. If one were to try to check if an element exists in this bloom filter, they would simply need to check if the indices from hashing the element are all set to 1. Of course, this could lead to false positives because sometimes one of the bits the element is associated with will already be 1. That is why the size of the bloom filter matters. The smaller the bloom filter is, the more likely false positives will occur. Additionally, the more bloom filter collisions there are the more hash table lookups will occur. This is shown in the following graph:



This graph shows that the smaller the hash table and bloom filter size is, the more times the program has to traverse branches. This is because there will be more words that are falsely marked as being in the bloom filter since there aren't a lot of open slots available in the bit vector. The graph is also logarithmic in shape which means there is less drop off of traversals the larger the hash table and bloom filter size is.

bv.c:

The file bv.c involved the data structure, bit vector. Bit vectors are a one dimensional array made up of bits. Cost efficiency was taken into account in my implementation of this data structure since the bit vector only needs to be sized at ceiling(n/8), n being the number of items. Basically to access 8 indices, you only need 1 integer. This function is a pivotal part of the bloom filter because it is used to map elements using the hash values.

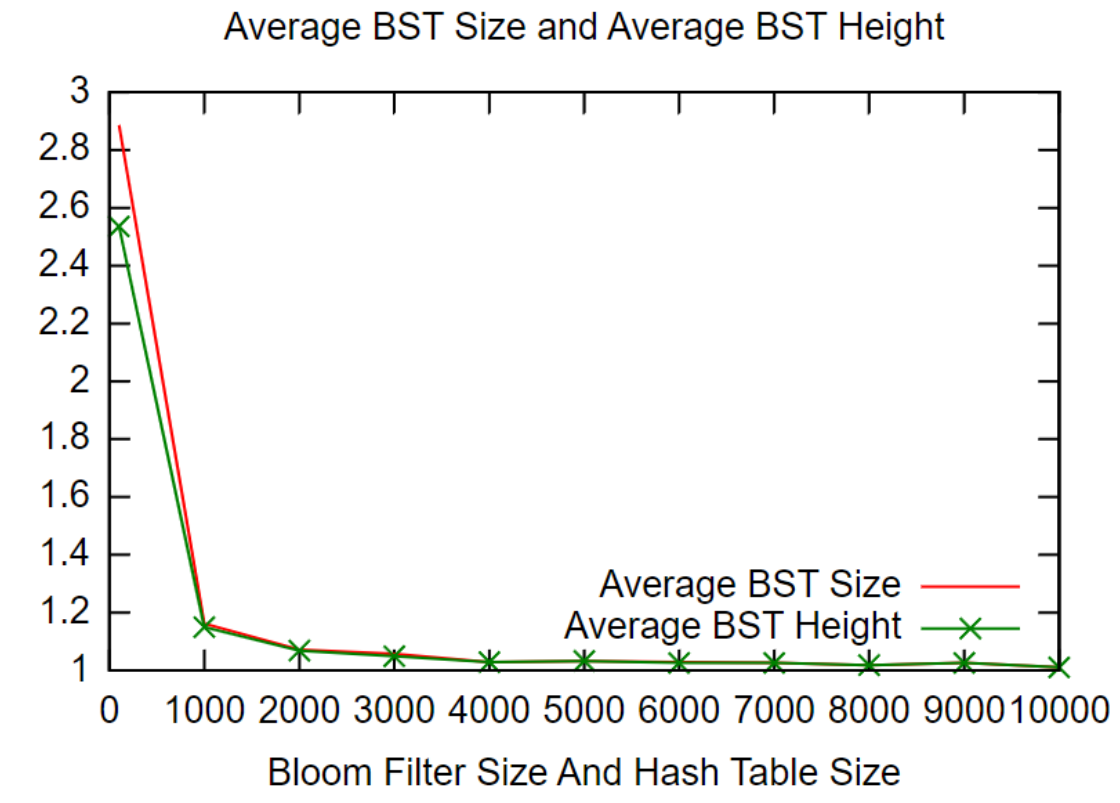
ht.c:

Hash table is another data structure and it matches keys to values. Specifically, my implementation of a hash table organizes oldspeak translations to their newspeak counterparts. If there are no newspeak translations for the word, then that means that the word is a badspeak word. This implementation also involves using binary search trees which I will discuss in bst.c. This tree holds all the oldspeak to newspeak translations and the badspeak words of the hash

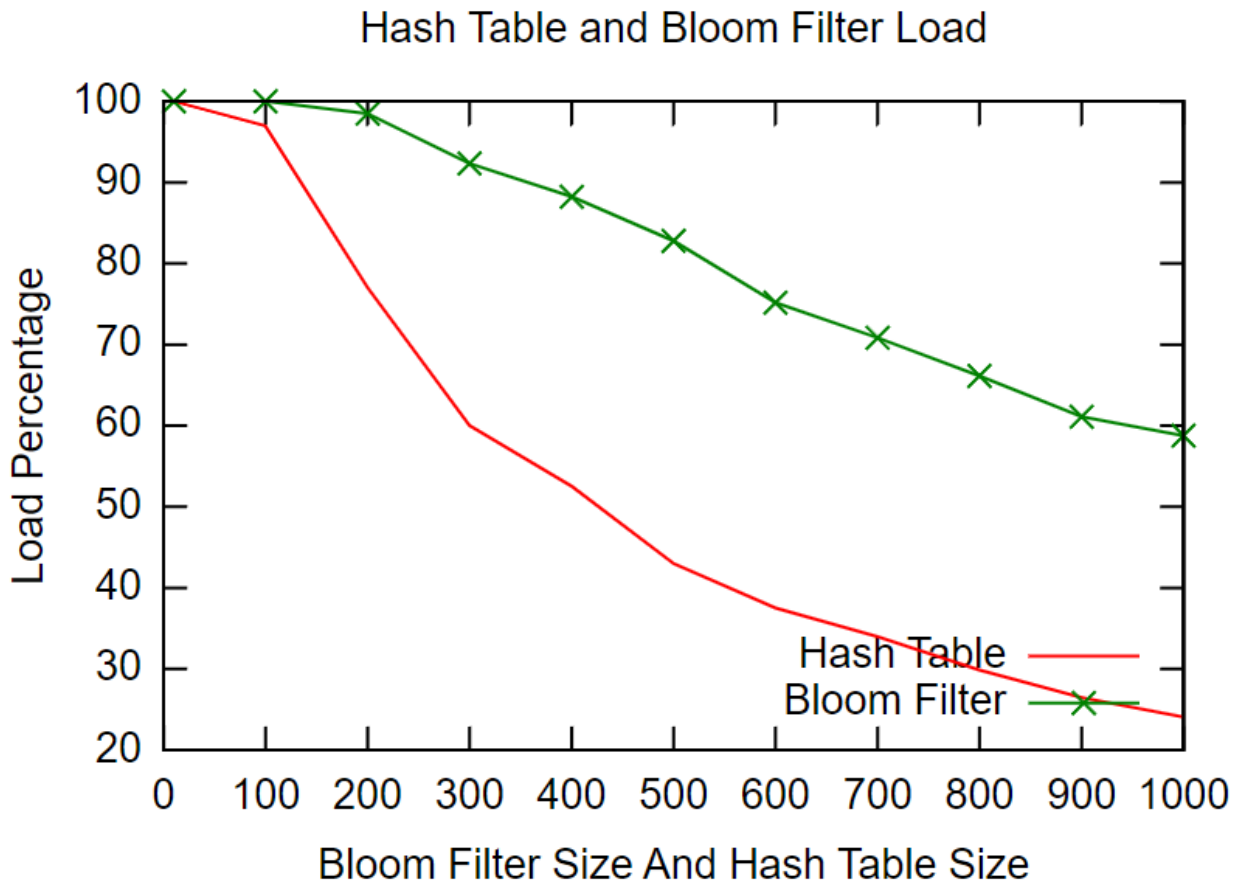
table. These binary trees are crucial because they will be used to resolve hash collisions. Additionally, hashing needs to be used again for lookups. If a user wants to look up if a word is in the hash table, they need to lookup if the word exists using hashing. Because hashing produces the same number, but emulates randomness, the location of the word will be the same every time.

node.c:
Nodes is another concept that we use in this assignment. Each node in the binary search tree contains the oldspeak and newspeak translation. If there is no newspeak translation, that means that word is a badspeak word. Each node will have a left and right child node as well.

bst.c:
Binary search trees are another data structure that we used for this assignment. A binary search tree is a tree of nodes to put it simply. In this case the nodes represent each translation or badspeak word as described above in the node.c description. Two of the more important functions in this file were bst_height and bst_size. These functions allow us to find out how the BST's height changes and what factors influence that change. Additionally, bst_size allows us to know how changing the size affects the number of lookups in the hash table because it just so happens that binary search trees are used to resolve hash collisions.



As shown from the graph above, the smaller the bloom filter and hash table is the higher the size and height of the binary search tree is. In other words, the more hash table collisions there are, the larger the binary search tree's size and height is. Besides hash table collisions affecting the height of the binary search tree, the order of inserts also determines the height of the tree. Since the order is in order lexicographically, inserting words directly already in order will make each layer of the tree a different word. Mixing up the words will make it so most nodes will have both a word that is less than and greater than it. It will have a left and right node.



This graph shows how much the bloom filter and the hash table get loaded the more or less the size of them it is. The less the size they are, the more likely they are to be full and thus get more collisions. As the size increases so does the percentage they are loaded.