Assignment 7:

Ban Hammer Writeup

By Derfel Terciano

Introduction

As the very charismatic and cool leader of Santa Cruz, you would need some sort of filtering system that filters any badspeak words from your citizens. You would need to apply some sort of filtering system to the internet, text messages, and any daily conversations to your citizens. In order to do this, we would need to create some sort of banhammer that would do the filtering for you.

In order to implement our banhammer, we would need to use a series of ADTs in order for us to successfully filter a file of text. The first ADT that we used was a **bloom filter**. The bloom filter is an ADT that can probabilistically tell us whether or not the hashed value of the word is in the corresponding bit vector. This is a probabilistic data structure due to the fact that it can only tell that the word isn't in the bit vector *or* it *may be* in the bit vector. We can never get a definite *yes* from the filter. The only way we can increase our chances of getting a definite *yes* from the filter is if we make our filter size extremely large. So is there another way to combat a false positive from the bloom filter? Yes! We can use a hash table to confirm whether or not our bloom filter was correct in the first place.

A hash table is a data structure with an O(1) look-up time. The hash table uses an array where each value is hashed into a random index in the array. The problem with hash tables is that there is a chance that a hash collision will occur. Hash collisions only occur when two keys are hashed into the same index of the array. In order to mitigate this problem, you either 1) use open addressing or use another data structure that can mitigate hash collisions. In this assignment, we used an array of binary search trees in order to deal with hash collisions.

Experimenting and Testing

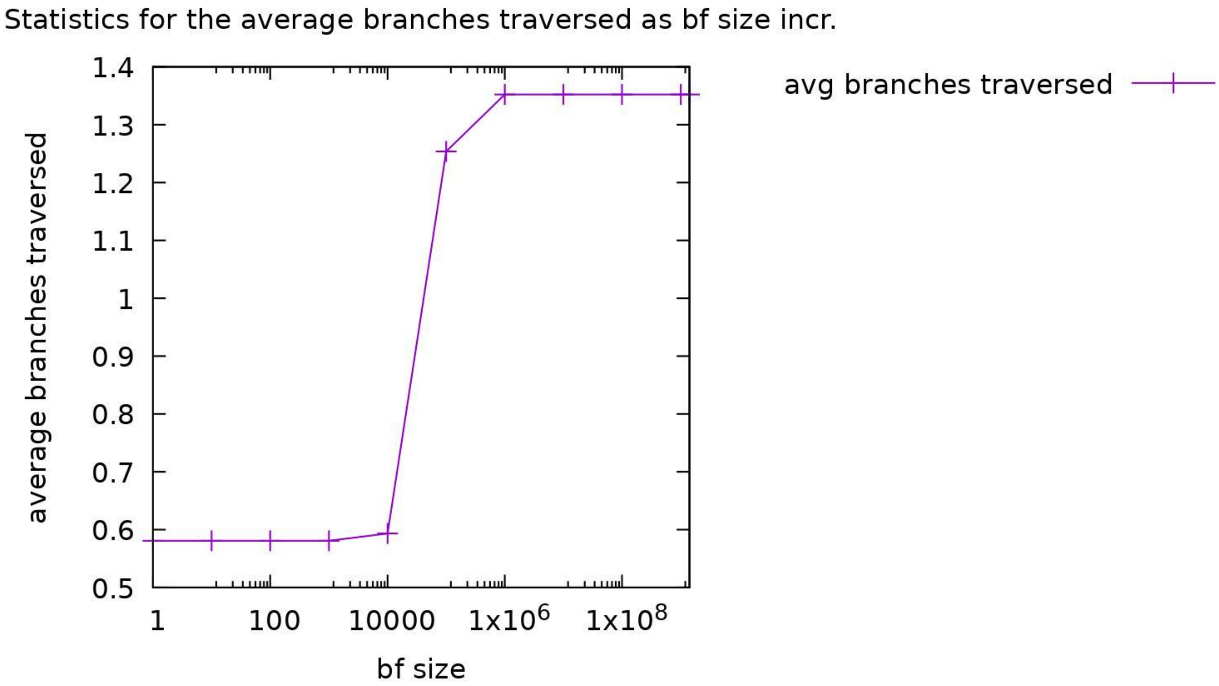
Once I had fully implemented the banhammer program, I wrote a shell script that would graph any statistics from the banhammer graph. The text file I used for my statistics is the *news* text file locate in the ~/resources/corpra/calgary folder. The following statistics is what is outputted to my graph:

- Bloom Filter size
- Hash Table
- The number of lookups we used on our hashtable
- The average number of branches traversed.

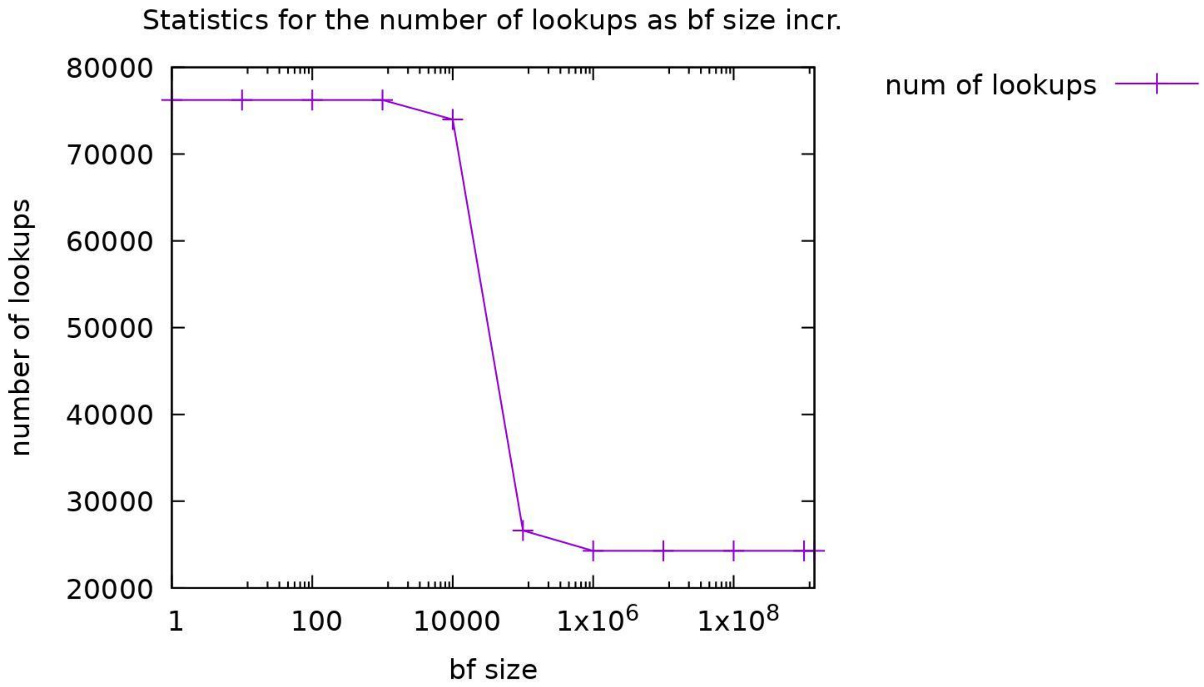
Results

The following graphs are the outputs of my statistics:

- **Graph 1:**

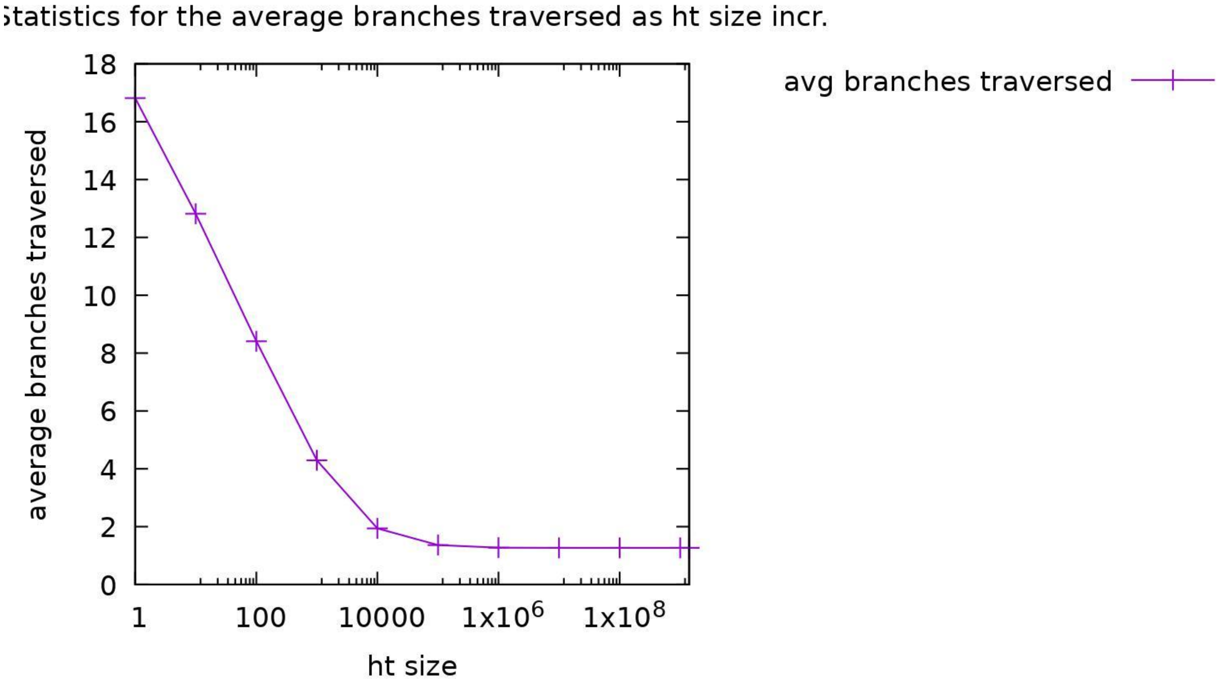


-
- **Graph 2:**



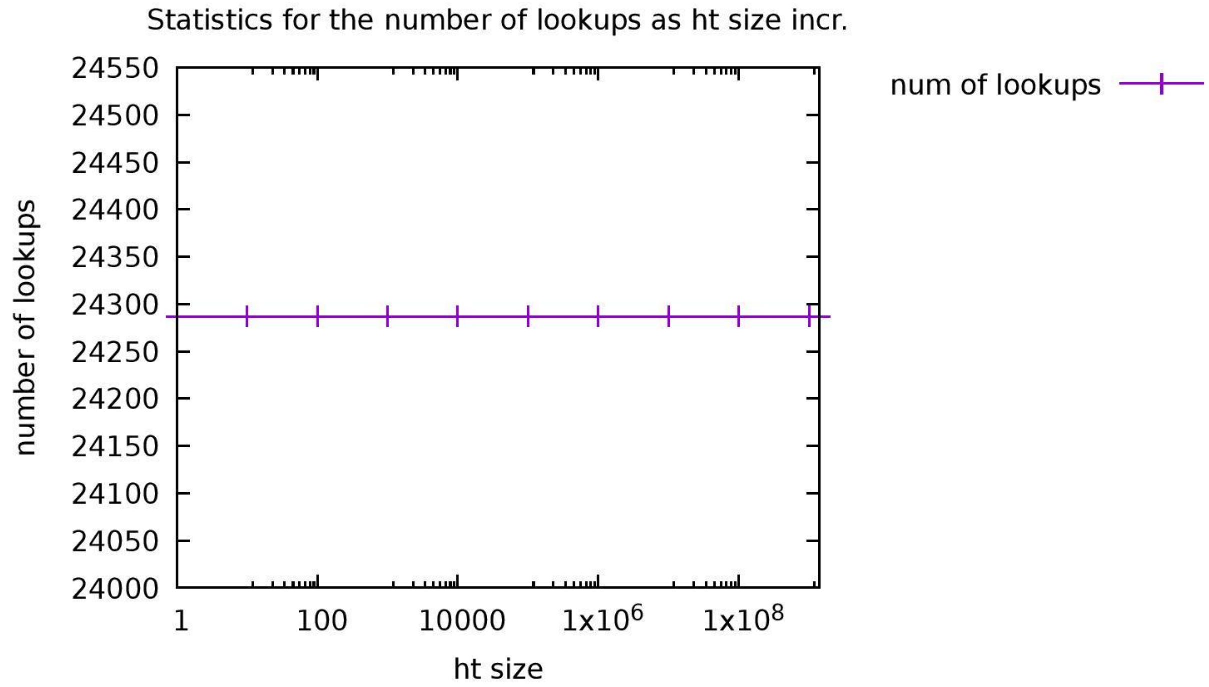
-

- **Graph 3:**



-

- **Graph 4:**



Analysis: The Bloom Filter

After processing the news file, we can see that if we keep the hash table size as the default value (2^{16}) but by increasing the bloom filter size, we can see that the number of branches traversed rapidly increases when our bloom filter size is set to 10,000 (**Graph 1**). In addition, when the bloom filter size is increased, we can see that the number of lookups decrease when our bloom filter size is between 10,000 and 1x10^6. (**Graph 2**).

Why is this? One big discussion about bloom filters that is known to computer scientists is the output of false positives that bloom filters tend to return. In order to mitigate this, the bloom filter size must be extremely big. With that being said, if we look at **Graph 1**, we can see that as the number of false positives decreases, the number of branches traversed increases due to the fact that we would need to search more binary search trees in the hash table.

Let's now examine **Graph 2**. We can see in **Graph 2**, that the number of lookups decrease as the bloom filter size increases. This makes logical sense due to the fact that as the number of false positives decreases, then the more confident we are in the bloom filter telling us that the word is in the filter. The more we become confident in the bloom filter, then we would utilize the hash table less since we wouldn't need to confirm with the hash table as much.

Analysis: The Hash Table

After processing the news file, we can see that as the size of the hash table increases, the average number of branches traversed decreases (**Graph 3**) while the number of lookups stays roughly the same (**Graph 4**).

Let's take a closer look at **Graph 3**. In the graph, we can see that the average number of branches traversed decreases. This is due to the fact that as the size of the hash table grows, then the number of hash collisions decreases thus, the number of branches we would need to look at decreases. So how does this affect the tree height? To simply put: as the number of hash collisions increases, then the height of the binary search trees increases since we are stuffing more items in the hash table than what the hash table can truly handle.

Let's now take a look at **Graph 4**. In the graph, we see something very odd; we see that the number of lookups stays the same. Why is this? This mainly due to the fact that the hash table lookups rely on whatever file we are processing. Since we are only processing the news text, the number of lookups will stay the same. If I varied the processing file, let's say I remove a word from the news file, then the number of lookups will then vary.

Conclusion

In conclusion, we discover that as our Bloom Filter size increases, then the confidence level in searching for an item in our Bloom Filter increases. Moreover, we discovered that as the size of the bloom filter increases, then the average number of branches traversed increases as well. Additionally, as the number of hash table collisions increases, then the size of the binary trees increases as well. Lastly, we discovered that the number of hash table lookups depends solely on the file that is being processed.