Analysis - The Great Firewall of Santa Cruz

Deeraj Gurram

Fall 2021

1 Introduction

In this assignment, we were asked to implement an input filtering program in which specific words are disallowed. We created various data structures to help with the handling of each word, and from the results, we can make various observations on the behaviour of each function. In this writeup I will be examining how changing HashTable and BloomFilter size changes the height of a Binary Search Tree, what factors affect the height of a binary search tree, and how the BloomFilter size affects the number of lookups performed in a hash table.

2 Code

```
double ht_avg_bst_height(HashTable *ht) {
    double avg_height = 0;
    for (uint32_t i = 0; i < ht->size; i++) {
        avg_height += bst_height(ht->trees[i]);
    }
    return avg_height / ht_count(ht);
}
```

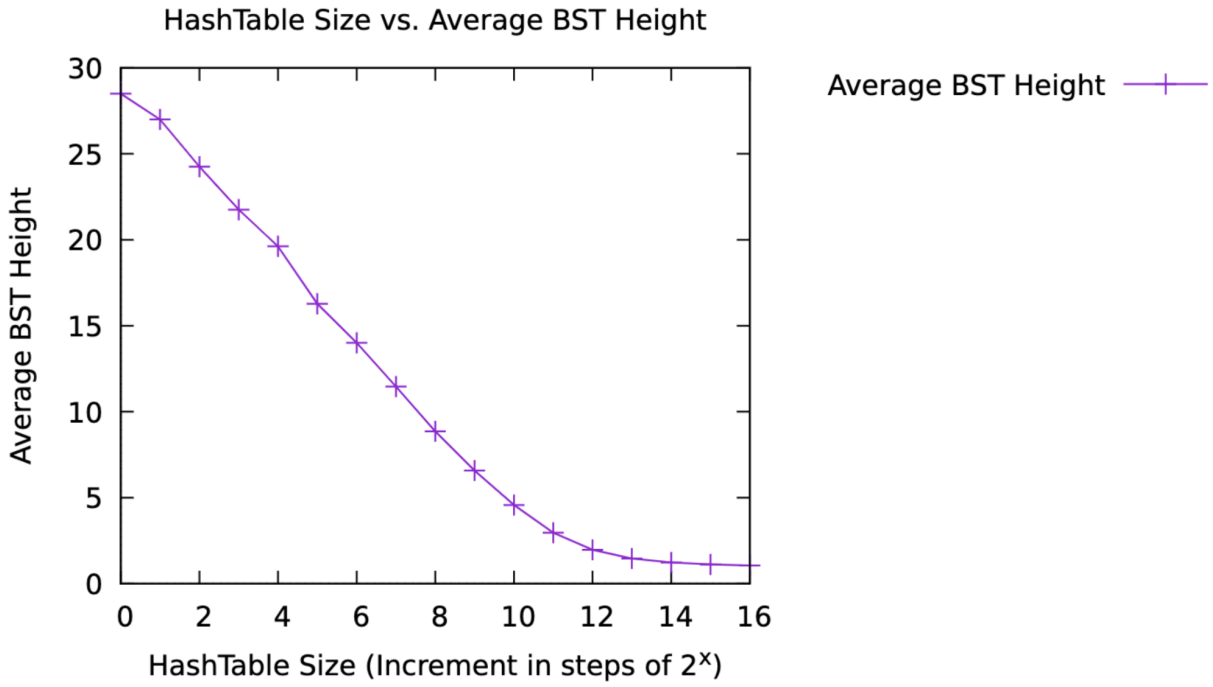
In this writeup, I'll be using ht_avg_bst_height for a majority of my graphs. This function goes through the entirety of a Hashtable and increments a variable avg_height by the height of the current element in the tree. At the end, it returns the summed heights divided by the number of non-NULL Binary Search Trees in the HashTable.

```
Node *ht_lookup(HashTable *ht, char *oldspeak) {
    uint32_t index = hash(ht->salt, oldspeak) % ht->size;
    lookups += 1;
    return bst_find(ht->trees[index], oldspeak);
}
```

I'll be using this function to compare the BloomFilter size with the number of lookups that were performed. This function is the place in which I increment the variable lookups. In the function, the number of lookups is incremented by 1 every time ht_lookup is called. In banhamer.c, I placed ht_lookup inside a while loop, which means the number of lookups is increased if bf_probe() fulfills the while loop condition and loops more often.

3 Analysis

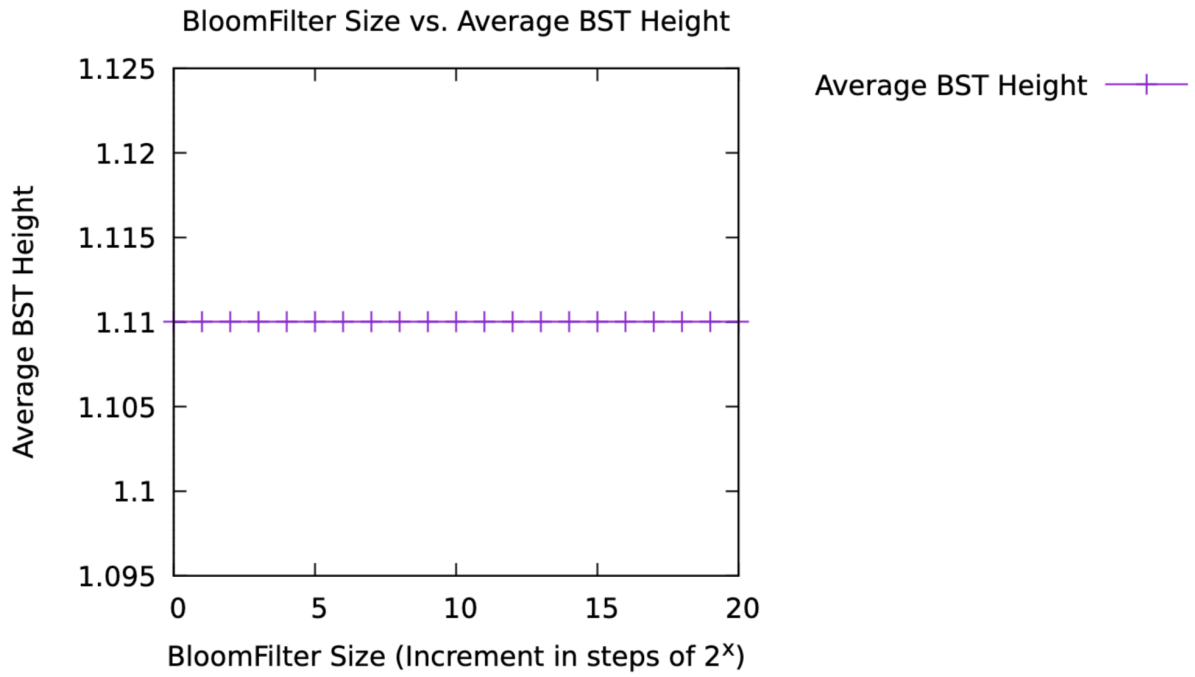
How do the heights of the binary search trees change when HashTable Size is varied?



It appears that when the HashTable size increases from 2^0 to 2^{16}, the average BST height steadily decreases until it seems to approach a straight line at 2^{16}. This can be attributed to the Pigeonhole principle in which the BST's height decreases as the HashTable size increases. Because an increase in the HashTable size leads to less collisions[1], it would lead to the BST height decreasing.

Collision[1]: The occurrence of two separate oldspeak words returning the same value when hashed.

How do the heights of the binary search trees change when BloomFilter Size is varied?



From this graph we can see that altering the BloomFilter size seems to have no effect on the Average BST Height. When looking at 2^0 to 2^{20}, the Average BST Height stays constant at 1.11. This is because the function ht_avg_bst_height doesn't use any function or variable from bf.c. As such, the BloomFilter size has no bearing on the Average BST height.

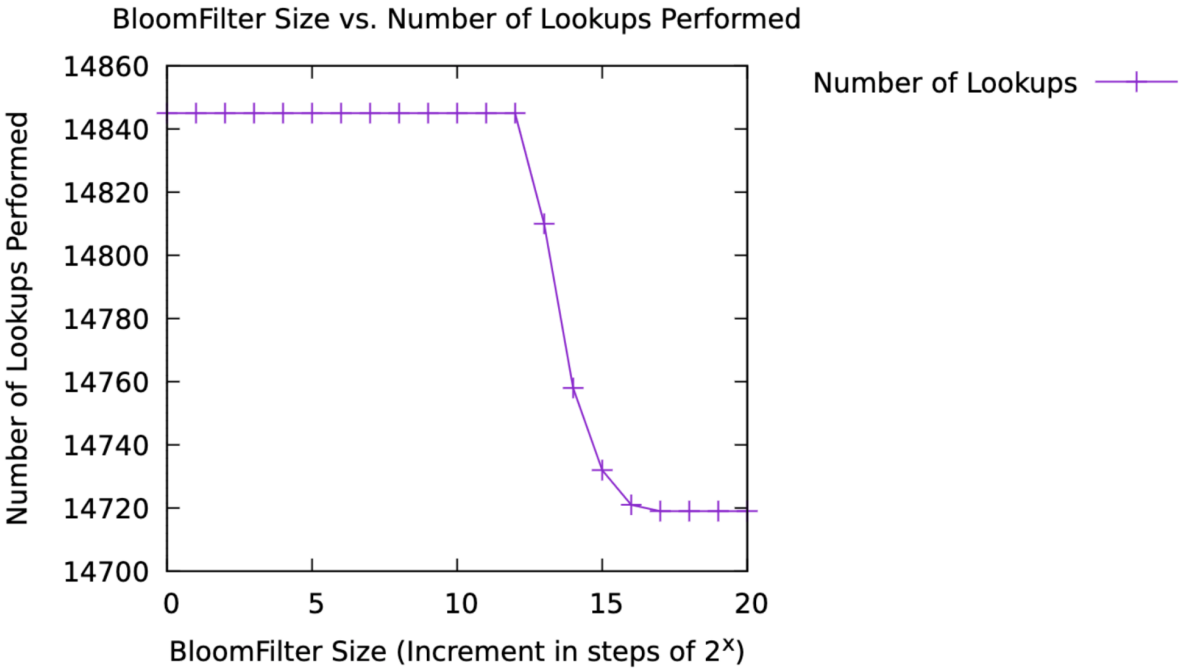
What are some factors that can change the height of a binary search tree?

As seen in my first graph, the HashTable Size is a factor that affects the height of a binary search tree. As explained by the Pigeonhole Principle, an increase in the HashTable size leads to less collisions between any hashed oldspeak that is found, which leads to the BST height decreasing.

Another factor that can cause the BST height to increase/decrease is the order in which an input is given. Because a BST is ordered lexicographically, if an input is consistently given at a higher lexicographic ordering, it will always be inputted into the current node's right child which would mean that every word inputted causes the BST height to increase by 1. Instead, if an input is given with randomized ordering there is a much higher chance that a balanced tree will be achieved as bst_insert will put the word into lexicographic order where each word has a chance of being inputted into the current node's left child or right child.

Note: I can't graph how the alphabetic ordering changes the BST height as I have no way of changing the x axis based on how many words are inputted which would let me see how the height changes.

How does changing the Bloom filter size affect the number of lookups performed in the hash table?



It appears that from a BloomFilter Size of 2^0 to 2^{11} the number of lookups stays the same. This is caused by the number of collisions occurring during the hashing of the oldspeak. Eventually at a BloomFilter Size of 2^{12}, the BloomFilter Size has increased to the point where there is enough space to store all of the oldspeak hashes, which results in less collisions. This drastically decreases the number of lookups performed as there are less times where bf_probe(bf, word) equals true. This becomes increasingly apparent as the BloomFilter size increases to 2^{15}, as the number of calls to ht_lookup decreases, therefore decreasing the number of lookups performed. Between a BloomFilter Size of 2^{16} and 2^{20} is when it becomes apparent that there are no further collisions happening, as the number of lookups steadies and becomes a straight line.

4 Conclusion

From the analysis made on the graphs, I've answered questions regarding how changing HashTable and BloomFilter size changes the height of a Binary Search Tree, what factors affect the height of a binary search tree, and how the BloomFilter size affects the number of lookups performed in a hash table. The second question poses an interesting answer in the form of input order which I explained in my analysis. To further expound on the subject, if input was given in ever increasing lexicographic order, then the height of a BST would be directly proportional to the number of words inputted. However, as most text files scanned in will not be in such an order, their BST will be more balanced which wouldn't result in a linear relationship between number of words inputted and height of the BST. To conclude this analysis, we can see that there are a variety of factors that affect the results of the input filtering ranging from varying the HashTable size, changing the BloomFilter size, or checking how differing the previous two values would change the number of lookups performed, or even the number of branches traversed.

The text I tested my graphs with:

Encryption is the process of taking some file you wish to protect, usually called plaintext, and trans- forming its data such that only authorized parties can access it. This transformed data is referred to as ciphertext. Decryption is the inverse operation of encryption, taking the ciphertext and transforming the encrypted data back to its original state as found in the original plaintext. Encryption algorithms that utilize the same key for both encryption and decryption, like SPECK, are symmetric-key algorithms, and algorithms that don't, such as RSA, are asymmetric-key algorithms. You will be given two files, speck.h and speck.c. The former will provide the interface to using the SPECK hash function which has been named hash(), and the latter contains the implementation. The hash function hash() takes two parameters: a 128-bit salt passed in the form of an array of two uint64_ts, and a key to hash. The function will return a uint32_t which is exactly the index the key is mapped to.