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

Nelly Sin

December 2021

1 Introduction

In this assignment, we are detecting and blocking words that people use. At first, I was very confused on how this works, but I slowly by bits and pieces are able to understand the result of the assignment with the help of the ADTs. This assignment deals more with bit vectors considering the the hash tables and bloom filters are the bit vectors.

Imagine ruling over a country and you decided to block out certain things your citizens say on the internet. In order to do so, we must provide a way to reward their good deed of not using the words and discourage them to ever use such words that can be hurtful or offensive.

2 Bloom Filters

Bloom filter is a bit vector ...
Operate operate more than one hash functions.
Primary secondary and tertiary. Salts is for changing up for the hash functions. A good hash functions must be fast, every single time h(x) => y it will produce the same number. Must be deterministic.

How do we fake the hash functions?

By providing the salt.
These salts are called "primary", "secondary", and "tertiary" The Bloom filter will added first and then the hash tables.
The structure of the Bloom filter has already been provided in the assignment doc.

Example input: "I use python in cse13s"

Parsing all the words then for each word we put in the bloom filter

2.1 BloomFilter delete

```
If the bloom filter exists
 delete the bloom filter and set the pointer to NULL
```

2.2 BloomFilter size

```
return the bloom filter size
```

2.3 BloomFilter count

```
for i in the range of the bloom filter size
if the bit from get bit is return 1 then return true otherwise return false
```

2.4 BloomFilter Insert

When we are inserting we will be using the insert with the salt, as said before about using salts. In addition we will be using the has fucntion provided in the speck file. The hashing should implement the salt along with the oldspeak into each of the three salts. In addition, whenever we will be hashing we will be getting the modulo with the bfsize() this is the same method when we hash in the BloomFilter Probe.

2.5 BloomFilter Probe

How do we check if the word is in the BF?
We use BF probe, when it's hashed 3 times, and if it is inserted it 3 times and we would know if it's in the bloom filter.

We check 3 indices because it is less likely for a collision – all three having the same index, thus it collides.

Every time we see all 3 indices are set in the bf, we check Hash Table for confirmation that it's not a false positive.

bad = [”cse13s”] b/c it does not have a newspeak
revise = [”python” $->$ ”slow”] We have a list of words to revise to print.

3 Hashing with the SPECK

SPECK is a lightweight block of ciphers publicly released by the National Security Agency. Encryption is the process of taking some file you wish to protect, usually called plaintext, and transforming its data such that only authorized parties can access it.

The encryption algorithms to utilize the same key for both the encryption and decryption. And SPECK are symmetric-key algorithms. This means algorithms for cryptography that use the same cryptographic keys for both the encryption of plaintext and the decryption of ciphertext. The keys may be identical, or there may be a simple transformation to go between the two keys.

The SPECK block cipher has been provided for us. However we will be implementing them for our hash tables.

4 Bit Vectors

A bit vector is an ADT that represents a one dimensional array of bits, the bits in which are used to denote if something is true or false (1 or 0). My understanding of the and code are derived from assignment 5 and the professor's code comments repository.

4.1 Bit Vector delete

if the bit vector exists
delete and set the pointer to NULL

4.2 Bit Vector length

return the bit vector's length

4.3 Bit Vector set bit

set the bit vector by shifting it to the left by a specified amount and using the "or" condition

4.4 Bit Vector clear bit

clearing the bits by "and" operator and shifting it to the left by a specified amount.

4.5 Bit Vector get bit

getting the bit by shifitng the bit in a certain amount such that we get the correct bit when calling it.

5 Hash Tables

Hashing is just a double checker of the bloom filter, however hash table can never return false positives.

Hash tables are used in data storage and retrieval applications to access data at a nearly constant time per retrieval. They require a storage space that is only greater than total space for the needed data. However, it's useful because hashing avoids the non-linear access time of order and unordered lists and often the exponential storage requirements and direct access of large state spaces.

Chaining by using binary search trees.
Similar to a bloom filter, the hash table will be using salt and whenever a new oldspeak is being inserted we will be using binary search trees to resolve oldpseak hash collisions, another reason why the hash table contains an array of trees.
Lecture 28 would be very useful for this.

5.1 HashTable delete

for every node in the hash table we will have to delete it, free it, and return it to NULL

5.2 HashTable size

returns the size of the hashtable

5.3 HashTable average binary search tree size

for i in the range of the hash table size add all the bst size in the the binary search tree and divide it by the total number to get the average

5.4 HashTable average binary search tree height

for i in the range of the hash table height add all the bst height in the the binary search tree and divide it by the total number to get the average

5.5 Hash table Insert

This function will insert the specified oldspeak and the newspeak translation into the hash table. And the index of the binary search tree is calculated by hashing the oldspeak.

5.6 Hash table lookup

Searching for an entry – the node in the hash table that contains the oldspeak. The node stores oldspeak AND newspeak translation. And index the binary search tree to perform a look-up that is calculated by hashing the oldspeak. If the node is found, the pointer to the node will be returned or else it will be a NULL.

This is similar to bf probe considering that hash table and bloom filters are very much similar.

6 Nodes

Because binary search trees will be used to resolve hash collisions. And to initialize and prepare for the binary search trees for this assignment, each node contains oldspeak and its newspeak translation if it exists. The key to search with in a binary search tree is oldspeak. Each node, in typical binary search tree fashion, will contain pointers to its left and right children.

6.1 Node create

duplicate oldspeak and newspeak if it has not yet existed and store it to the node or their perspective node
otherwise the node for oldspeak and newspeak is NULL
and set the left and right node to NULL

6.2 Node delete

if the node exists, we must delete all oldspeak and newspeak nodes

6.3 Node print

if the oldspeak and newspeak is not NULL, we can print:
"oldspeak -¿ newspeak"

otherwise, if it is just oldspeak
"oldspeak"

7 Binary Search Trees

Binary search trees, starting off the root if it is greater than then then you go to the right side (total ordering).

A Bloom filter can be represented as an array of m bits, or a bit vector. A Bloom filter should utilize k different hash functions. Using these hash functions, a set element added to the Bloom filter is mapped to at most k of the m bit indices, generating a uniform pseudo-random distribution. Typically, k is a small constant which depends on the desired false error rate , while m is proportional to k and the number of elements to be added.

In this case we will be taking the list of the prescribed words such as oldspeak and add the word into our bloom filter. If the words that the citizens use is added to the bloom filter, then we must give them a warning message.

In the process to reduce the chance of a false positive we must use three salts for three different hash functions. Salt can be a initialization vector or a key. Using the different salts with the same hash function results in different hashing.

Lecture slide 18 will be useful for this.

7.1 Binary Search Tree delete

recursively call the binary search tree (traverse) such that we can delete the binary search tree right and left nodes.

7.2 Binary Search Tree height

if the node exists add one to the largest between the right and left nodes of the binary search trees

7.3 Binary Search Tree size

if the node exists add one to the right and let node of the binary search tree.

7.4 Binary Search Tree delete

if the root and if the left and right of the binary search tree exist delete the right and left nodes of the binary search tree.

7.5 Binary Search Tree print

if the root exists, we can traverse through the binary search tree to print the right, left, and the root node of the binary search tree.

7.6 Binary Search Tree Find

This function will search for a node containing the oldspeak in the binary search tree rooted at the root. If the node is found, the pointer to the node is returned. Else the pointer is NULL when returned. *if the root exists*

if the root of the oldspeak is greater to the oldspeak in the file then the root on the left with the oldspeak
else if the root of the oldspeak is less than to the oldspeak in the file then the root is on the right with the oldspeak
We will also be dealing with recursion in this function.

7.7 Binary Search Tree Insert

In this function we will be inserting a new node containing the specific oldspeak and newspeak into the binary search tree rooted at the root (this is similar to find, however, we will be dealing with newspeak). Also, we should not be duplicating the strings when inserting.

if the root exists
if the root of the oldspeak is greater to the oldspeak in the file then the root on the left with the oldspeak and newspeak
else if the root of the oldspeak is less than to the oldspeak in the file then the root is on the right with the oldspeak and newspeak
We will also be dealing with recursion in this function.

8 Lexical Analysis with Regular Expression

We are using this to lowercase all the files. According the the assignment doc. the lexicon of badspeak and oldspeak/newspeak translations has been populated and then we can start filtering the words. The parsing should be before checking the word that is read in.

BF + BV + BST + HT => data table of bad words + fast querying

Given a parsing module, (given) if you give it a regular expression it will give a regular expression for that.

They are based off of state machines – Regular Expression:
"+" = 1 or more
"x" = 0 or more
"1" = or
"?" = optional (0 or 1)
= symbol set

(0—1)+ = binary
Means 1 or 0 in one of more times
0x([a-f 0-9])+ = hex
a-f means (a,b,c,d,e,f)
same with 0-9 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

If the word is likely to be added to the bloom filter, using bfprobe we do not need to take action. However, if the hash table contains the word and the word does not have newspeak translation then we must notify the citizen who used the word is guilty and must send a message to and list the badspeak words that are used.

If the hash table contains the word, then the word does have newspeak translation –indicating the usage of rightspeak. We then reward the citizen by sending a message to them.

However, if the citizen is accused of using badspeak and newspeak. They are warned with a message that contains the usaged of mix speak and list the oldspeak words and the newspeak words that are used.

This would be a very helpful website to test regex visually:
https://regex101.com/r/66AuYV/2

9 Statistics

In our statistics, we must calculate our branches and lookups inorder to calculate the average number of branches that are traversed. To do so: we set a

global variable in our bst file. Each time we find or insert we must increment the branches variable. Similarly, in lookups we will be calculating in the ht file, due to the fact that we are accessing the hashtable. So, each time we call the htlookup function() we are increment the lookup global variable whenever we call it. Again, this will also be in our htinsert as we call it each time. I chose to think of setting it as a global integer because we must call it in our

banhammer file to calculate and print our statistics.
Here are the equations we we provided to by Professor Long and TA Eugene.

$$Average = \frac{branches}{lookups} \tag{1}$$

$$htLoad = 100 * \frac{htCount()}{htSize()} \tag{2}$$

$$bfLoad = 100 * \frac{bfCount()}{bfSize()} \tag{3}$$

10 Banhammer

In our banhammer file we will be putting everything together. We will be reading the files of the badspeak and newspeak. – Badspeak is our oldspeak without the translation for newspeak. While the badspeak words are added to the bloom filter and the hastable, the oldspeak and newspeak are only added to the hastable. Such that the list of oldspeak and newspeak are pairs in the actual newspeak text file.

After populating the lexicon of the words, we can begin using the filtering by using out ADT functions. We will be reading from a input file and for each word we read in, we have to check if it has been added to the bloom filter – one of our bf functions – if it has not added then it indicates that it is not a proscribed word and we do not need to take any further action. But if it is true – indicating that it is in the bloom filter, we do not need to take further action. By doing this we will be using bf probe.

If the hashtable contains the word and if the word does not have a newspeak translation, this will indicate that the cititzen who is using the guilty word as a ”thoughtcrime”. We then insert the words that is considered bad into a list of badspeak words and will later be printed to notify the user their errors later on.

I used the bst create for printing these words considering that we will be using a data structure that could be used to store the words.

So, if the hashtable does contain the word and the word have a newspeak translation then the citizen will be praised by printing the ”Rightspeak” message. Doing so, we are inserting the oldspeak word into a list of oldspeak words with the newspeak translation to know whether to print the ”Rightspeak” message.

When we print out the contained words of badwords or oldspeak and newspeak, we must print them out alphabetically.

I thought about using booleans to do the printing of ”Rightspeak”, ”Badspeak”, and ”Mixspeak” messages. This is similar to assignment 6. Because the hash table does not contain the word, then it is fine because only the Bloom filter create a false positive.

The following commands to run the binary:

-h prints out the program usage. Refer to the reference program in the resources repository for what to print.

-t size specifies that the hash table will have size entries (the default will be 2 to the power of 16).

-f size specifies that the Bloom filter will have size entries (the default will be 2 to the power of 20).

- s will enable the printing of statistics to stdout. The statistics to calculate are:

1. The average binary search tree size
2. The average binary search tree height
3. Average branches traversed
4. Hash table load
5. Bloom filter load

11 Citation

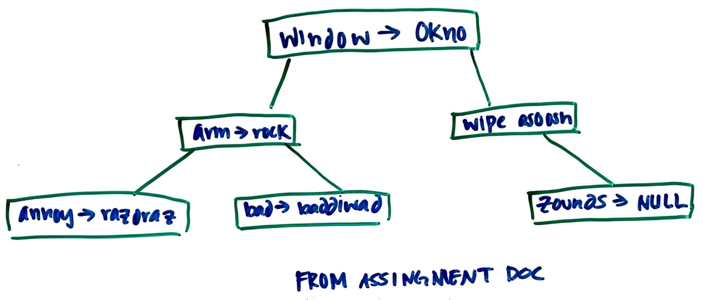
Special acknowledgements to the TA's, tutors, and Professor Long for helping me throughout the concepts and providing some advice for my pseudo code and publishing some pseudo code or code in the assignment document.

The code I have provided in my work are not all written by me, but by tutors and TA's that provided further help for me. And credit to the professor for providing additional help from his public repository, code comments, and lecture slides 28 and 18 that he sent to the class.

In addition, most of the code that we have worked on for the ADTs are very similar or the same as our previous assignments we have created. I am referencing off those assignments and using them for my advantage. And I did not want repetitiveness in my DESIGN document.

12 Graphics

There are some additional graphics I have added to understand this assignment all of these are from Eugene's lecture or from the assignment



document.

Figure 1: Structure of binary search tree.

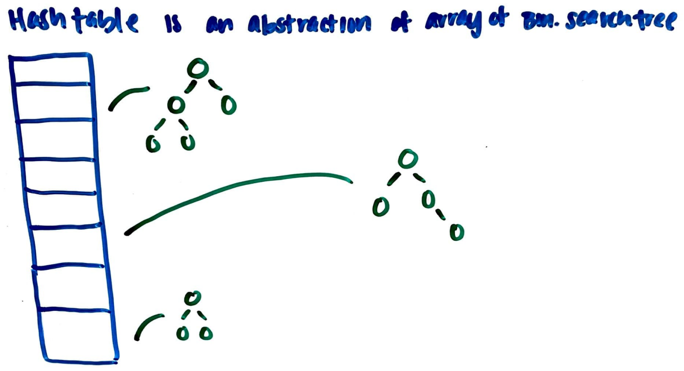
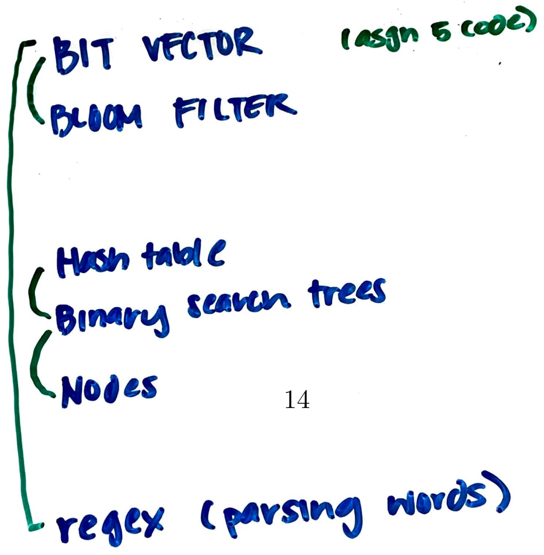


Figure 2: How the binary search tree relates to the hash table.


BIT VECTOR
(asgn 5 code)
BLOOM FILTER
Hash table
Binary search trees
Nodes
regex (parsing words)