Govind Pillai
gopillai@ucsc.edu
5/27/2021

CSE 13s Spring 2021

Assignment 7: The Great Firewall of Santa Cruz
Design Document

PURPOSE:

The purpose of this lab is to create a Firewall that will censor “bad words” given in the input. This program uses a parser which will parse the input file into words defined by regular expressions. These words will then be checked by a bloom filter to see if the word said was a bad word. If the bloom filter detects a bad word it will run it by a hash table of linked lists to make sure we did not run into a false positive. The user will have the option to enable “move-to-front”m which will move an element that has been detected in the input file to be moved to the front of the linked list. This helps with efficiency. Depending on what type of words are found, the program will print various messages.

DEFINITIONS:

Bloom Filter: A data structure that is used to test if an element is in a set.

Hash Table (of Linked Lists): The index of an element in a hash table is determined by calling hash on the element as well as passing in a salt to the hash function. The elements of the hash table are linked lists of nodes in order to deal with collisions.

Linked Lists: A data structure similar to arrays except that the elements of the list itself are linked using pointers.

Regular Expressions: A sequence of characters that are used against inputs to see if they match a type of search patterns

Tree dump: An encoded version of the tree which is used by the decoder to recreate the tree when decoding the file itself

DIAGRAMS:

Bloom Filter:

A bloom filter looks very much like an array, with all its elements set to 0:

Index	0	1	2	3	4	5
Element	0	0	0	0	0	0

If we hash a bad or oldspeak word, we set the bit (element = 1) at that hash:

hash(primary, [bad word]) → 3
hash(secondary,[bad word]) → 2
hash(tertiary,[bad word]) → 0

Index	0	1	2	3	4	5
Element	1	0	1	1	0	0

Now, when we want to check if a word is bad, we can see if all of its hash numbers are set in the bloom filter. Obviously, there can be collisions which is why we use a hash table to make another check.

Hash Table:

Our hash table looks very similar to our bloom filter, however each element is a linked list.

Insert	0	1	2	3	4	5	6	7
Element	{}	{}	{}	{}	{}	{}	{}	{}

We can add bad words to our hash table by hashing them using the given salts and adding the element to the linked list at the given index (hash):

hash(salt1, [bad word]) → 5

Insert	0	1	2	3	4	5	6	7
Element	{}	{}	{}	{}	{}	{bad word, null}	{}	{}

The NULL means that this bad word is not an oldspeak word and has no translation. If we have another bad word that hashes to the same index, we just add on that element to the linked list:

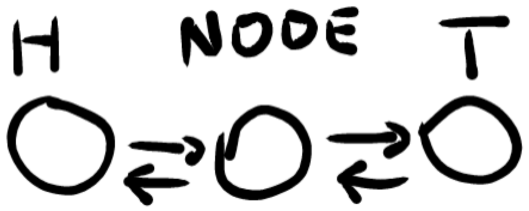
hash(salt1, (swear, curse)) → 5

Insert	0	1	2	3	4	5	6	7
Element	{}	{}	{}	{}	{}	{([bad word], null), (swear, curse)}	{}	{}

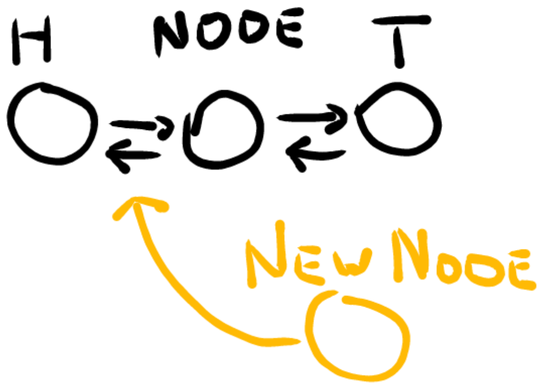
In this case, "swear" is an oldspeak word with the translation "curse".Now, if a false positive gets past our bloom filter AND hashes to the same index as an occupied list, we can simply check if that word is in the list in order to determine if it is or isn't a bad word.

Linked Lists:

The linked lists that are going to be used in this program will be elements of a hash table. A linked list is similar to an array however each element is linked together:



If we wanted to add another node to this linked list, it would have to be linked to the head:



But in order to do this, we need to change the links of the list itself:

New node → next = H → next

Explanation: New node is going to be in front of node. Therefore, New node's "next" is going to be node. Currently, node is H's "next". That is why we set new node's "next" to H's "next".

New node → previous = H

Explanation: Head is going to be in front of new node. That is why it will be New Node's previous.

H → next = New Node

Explanation: The opposite of the last one. Head is going to be in front of new node. That is why H's "next" will be new node.

New Node → next → previous = New Node

Explanation: The opposite of the first one. Now that we have set New Node's "next" to be Node, we have to set Node's "previous" to be New Node.

Regular Expressions:

These are used in order to match strings to various patterns we can assemble. Here is an example:

[a-z]: This will match one character a-z
[a-z]+: This will match one or more characters a-z
[a-zA-z]+: This will match one or more characters a-z disregarding case
[a-zA-z]*: Zero or more characters a-z disregarding case

If we had the expression:

[a-zA-z]@gmail.com. This will match:

govindpillai@gmail.com
sdgosdof@gmail.com

But will not match:

govindpillai9@gmail.com
skldjfasdjfgh44@gmail.com

Because we did not identify numbers [0-9] in our regular expression.