Jessie Zhu Thursday, May 27, 2021

Assignment 7: The Great Firewall of Santa Cruz

Design Document

For this assignment, we will create a Firewall to filter through “unfortunate, hurtful, offensive, and far too descriptive language.”

The purpose of this assignment is to learn how to create a filtering system for words. This will include using a bloom filter, hash tables, linked lists, and many other previously learned data structures and ADTs to implement the filter. Regex will also be used to parse in words from stdin.

Design Process:

- I used regex101 to test my regex pattern
- My code implementation didn’t change very much from my original design. My original plan didn’t have as much detail, but I was able to quickly fill in the details after watching Sabrina and Sahiti’s section (thank you TAs!, very helpful).
- My biggest struggle was creating the Nodes for the sentinel nodes. I didn’t realize that I need to include a special if statement for the null sentinel nodes. Originally I had thought my problem was with strdup, then I realized that the issue came from trying to copy a null node (I wasted a couple hours trying to find the error on this).
- Scan-build originally flashed errors when reading in the badspeak.txt and newspeak.txt words in. I had a variable to count in the number of words read in a line which was a variable that I had no purpose for (kinda like: c = fscanf(file, %s %s, arg1, arg2). c serves no purpose), so I got rid of it and then the error disappeared.
- Regex originally was super confusing for me because its literally random symbols and letters, but after this assignment, I feel more familiar with it.

Pseudocode:

Bloom filter:

Will be used to test whether an element is in a set. Can tell you whether something is definitely not in the set or if something is possible in the set. Bloom filter will be represented as array of bits. The bloom filter will have three salts stored in primary, secondary, tertiary fields and are of 128 bit size. They are held as two items of an array.

ADT functions (struct is given):

- bf_create - given in assignment doc
- bf_delete - will delete the bf
- bf_insert - hash oldspeak with three salts, then set the bits at those three indices
- bf_probe - hash oldspeak with the three salts for three indices. If all bits at those indices are set, return true to signify that old speak was likely added to bloom filter, otherwise false
- bf_count - number of set bits in bf
- bf_print - prints the bf

Bit Vectors:

Bit vectors will be used to set and clear bits in an array. This is reused from assignment 5.

Hash Tables:

Hash table will map keys to values to help quickly look things up. The struct will use linked lists to resolve oldspeak hash collision. The struct will contain an array of 2 items for the salt, uint for the size, bool for mtf (move to front technique), and a linked list.

ADT function (struct is given):

- ht_create - constructs the hash table, set salts and other initial values, allocates for linked list.

- ht_delete - deletes each of the linked lists in lists, array of linked lists is freed, pointer is set to NULL
- ht_size - return ht's size
- ht_lookup - hashes the oldspeak, check if linked list at that index is created (return null if dne) then search for the word in said linked list
- ht_insert - hashes oldspeak, check if linked list is created at that index (create if dne). Insert word into said linked list.
- ht_count - returns number of non-NULL linked lists in the hash table
- ht_print - prints the contents of the hash table

Linked Lists:

Will be used to resolve hash collisions. Each node in the linked list will have oldspeak and newspeak. Oldspeak will be used as the key. As a doubly linked list would, each node has a pointer to the previous and the next node in the linked list. All these thing will be in the struct.

Node ADT functions (struct is given):

- node_create - constructor, make copy of oldspeak and newspeak and set it in the node. Set next and prev to null
- node_delete - current node is freed, previous and next are not touched
- node_print - print is given (must "produce correct program output")

Linked list ADT functions (struct is given):

- ll_create mtf - if mtf is true, then all nodes found in linked list is moved to front of the linked list. Linked list will be initialized with two sentinel nodes to serve as head and tail of the linked list.
- ll_delete - each node of the linked list will be freed. Pointer set to null
- ll_length - length of the linked list, not including the sentinel nodes
- ll_lookup - searches the node that contains oldspeak and pointer returns if found. Otherwise null pointer returned. If node was found and mtf, then move the new node to front of linked list.

- ll_insert - inserts new node containing newspeak and oldspeak into linked list. Before inserting the node, check for duplicate. Otherwise, new node is inserted at head of the linked list, right after head sentinel node.
- ll_print - prints the nodes in the linked list except for head and tail sentinel nodes, node_print()

Parser:

This function will parts out the words that they speak. Hyphenations and apostrophes are accepted. Given in repository

Banhammer:

The main of our program. Begin by initializing bloom filter and hash table. Read in the bad speak with fscanf. Add each badspeak to the bloom filter and hash table. Then read in oldspeak and newspeak pairs with fscanf. Add these only to the hash table. Now we can start filtering words. Read from stdin using the parsing module. For each word read in, if word is most likely in the filter (bf_probe) then you need to deal with the word: if the hash table contains the word but does not have a newspeak translation, then insert this badspeak word into list of badspeak words to notify of “thoughtcrime”, if hash table does have the newspeak translation, then insert the oldspeak world into a list of oldspeak words with newspeak translation to notify “counseling on proper Rightspeak” if hash table doesn’t contain the word, then the word was a false positive, nothing needs to happen. There are three possible messages, display the correct one corresponding to status of whether “thoughtcrime” or “counseling on proper Rightspeak” is committed/necessary.

(Pseudocode block on next page for banhammer)

Banhammer:

```
Main:
    get opt stuff
    initialize bloom filter and hash table
    read badspeak and add to bloom filter and hash table
    read oldspeak and newspeak pairs, add to hash table
    initialize list for thoughtcrime and counseling
    read stdin using parsing module, for each word read
        if word most likely in bloom filter:
            if hash contains word but no newspeak translation:
                insert oldspeak into list of thoughtcrime words
                yes thoughtcrime
            if hash contains word and newspeak translation:
                insert oldspeak into list of counseling words
                yes counseling
            else:
                false positive, do nothing
    if thoughtcrime and counseling:
        display corresponding message
    else if thoughtcrime:
        display corresponding message
    else if counseling:
        display corresponding message
```