Mattiwos Belachew
mbelache@ucsc.edu
6/06/2021

CSE 13s Spring 2021
Assignment 7:
The Great Firewall of Santa Cruz:
Bloom Filters, Linked List, and Hash Table
Design Document

Description:

For this assignment, I will be implementing a bloom filter. A bloom filter is a space-efficient probabilistic data structure, used to test whether an element is a member of a set and return its probability of it being in the set. The goal of the project is to catch and punish those who practice wrongthink and continue to use oldspeak,which is not allowed in the new regime. Thus, using a bloom filter I will exercise mercy and counsel the old speakers so that they atone and use newspeak, a replacement of the oldspeak words.

TOP LEVEL:

Banhammer:

```
banhammer(){
        //parse program arguments
        //initialize Bloom filter and Hash Table
        //create a buffer of char of 4096 length called badword
        //open bad word list text file
        //pass in the bad word list and bad word buffer into scanf
        //while scanf has not reached the end of the file
                //insert badword into the Bloom Filter
                //insert badword and NULL into the Hash Table
        //close the bad word list file

        //open newspeak text file
        //create oldword and newword char buffer of 4096
        //pass newspeak and oldword, newword into scanf
        //while scan has not reached the end of file
```

```
            //insert oldword into Bloom Filter
            //insert oldword,newword into Hash Table
        //close newspeak text file

        //create word
        //create linkedlist called badspeak
        //create linkedlist called rightspeak

        //while next word in stdin isn't null
            //check if word is inside of Bloom FIlter
                //if so look up work in HashTable and set it to a Node
                //if Node is a NULL
                    //continue;
                //else if Node newspeak is NULL
                    //insert Node old speak to badspeak linked list
                //else
                    //insert Node old and new speak to badspeak linked list
        //if linked list length of badspeak and right speak is greater than 0
            //print mixspeak_message
            //print linkedlist of badspeak
            //print linkedlist of rightspeak
        //if linked list length of badspeak is greater than 0
            //print badspeak_message
            //print linkedlist of badspeak
        //if linked list length of rightspeak is greater than 0
            //print goodspeak_message
            //print linkedlist of rightspeak
        //close programs
        //regex free re
        //delete Hash Table
        //delete Linked List badspeak and right speak
        //delete Bloom filter
        //return success


}
```

- Filters the input and prints out what needs to be corrected or words that need to be omitted from a person's vocabulary since it's not allowed in the new regime.

- Banhammer uses Bloom filter in order to get the probability of it being in the hash table. If a word is in a Hash Table then it will add it to the linked list of words that needs to be corrected.

Ht (Hash Table):

```
HashTable *ht_create(uint32_t size, bool mtf) {
        //given in document
}

void ht_delete(HashTable **ht) {
   //check if ht exists
   //check if ht* exists
   //Loop for (*ht)->size
        //if the list isnt null at that location then
                //link list delete it

    //free (*ht)->lists
   //set (*ht)->lists to NULL;
   //free *ht
   //set *ht to NULL
}

uint32_t ht_size(HashTable *ht) {
   //check if ht exists
   //return ht->size
}

Node *ht_lookup(HashTable *ht, char *oldspeak) {
   //check if ht exists
   //set index to the hash value of ht salt and oldspeak and mod it to ht size
   //if the lists at index is NULL then
      //return null
   // use ll_lookup(ht->lists[index], oldspeak) to find value and return it
}

void ht_insert(HashTable *ht, char *oldspeak, char *newspeak) {
   //check if ht exists
   //set index to the hash value of ht salt and oldspeak and mod it to ht size
```

```
    //if the lists at index is NULL then
        //create a linked list at that index of lists ht
    //insert (ht->lists[index], oldspeak, newspeak) using ll_insert
}

uint32_t ht_count(HashTable *ht) {
    //check if ht exists
    //set count to 0;
    //Loop for (*ht)->size
          //if the list isnt null at that location then
             //increase count by one
    //return count
}

void ht_print(HashTable *ht) {
    //check if ht exists
    //Loop for (*ht)->size
          //if the list isnt null at that location then
                  //link list print the list at the location
}
```

- Used to create a Hash table which contains a list of linked lists. This is meant to prevent hash collision if ever the hashing of different words return the same index of the hash table.
- Hash tables are used to contain old speak and newspeak and an efficient way to do lookups and inserts. In other words, it is a dictionary of key and value.

LI (Linked List):

```
LinkedList *ll_create(bool mtf) {
        //create linked list by allocating space for it
        //Set ll head to a node with both head and tail to NULL
        //Set ll tail to a node with both head and tail to NULL
        //set ll head next to ll tail
        //Set ll tail prev to ll head
        //set ll length to 0
        //set ll mtf to mtf
```

```
        //return linked list
}

void ll_delete(LinkedList **ll) {
        //check if linked list exists
        //check if * linked list exists
        //set current node to ll head next
        //create a temp node
        //while current node isn't equal to ll tail
                //set temp = curr next
                //delete current node
                //set curr to temp
        //delete node at ll tail
        //delete node at ll head
        //free * linked list
        //set * linked list to NULL
}

uint32_t ll_length(LinkedList *ll) {
        //check if linked list exists
        //return length of linked list
}

Node *ll_lookup(LinkedList *ll, char *oldspeak) {
        //check if linked list exists
        //increment seeks by 1
        //loop through linked list
                //increment links by 1
                //if current old speak equal to oldspeak
                        //bridge over curr to head
                        //now move it back
                        //return current node

        //return NULL
}

void ll_insert(LinkedList *ll, char *oldspeak, char *newspeak) {
        //check if linked list exists
        //if ll at oldspeak is null
                //Create a node
```

```
                //Set next to ll head next
                //Set the node prev to ll head
                //ll head next to the node
                //Set the next prev to node
                //increment ll length by 1
}

void ll_print(LinkedList *ll) {
        //check if linked list exists
        //loop through linked list
                //node print at the current location
}
```

- This is used in hash tables to prevent hash collisions. The linked list is a double linked list with sentinel nodes to know when head and tail has been reached.

Design Process:

1. I was able to watch Eugene's section on Yuju and blast through most of the program quickly. I am having problems with the final phase of printing the message because my current linked list doesn't seem to work.
2. Currently confused about where I should be increasing the seeks and links extern variables in the linked list.
3. I was able to figure it out but my links count was off by a small amount causing the average seek length to be off by 1 percent. I am not sure what's wrong with it.

What I learned:

1. I learned how to create a bloom filter and how to use it in order to make my program more efficient. In addition, I learned how to create a hash table with a linked list in order to detect whether an element is in the dictionary.
2. I learned how to use regex in order to recognize patterns, to parse stdin for my banhammer program and distinguish what is a word and what isn't.
3. I learned how to manipulate linked lists and how to insert nodes into the list and move nodes from anywhere to after the list head.

Resources Used:

- Asgn7.pdf
- asgn5Design by William Santosa from Piazza
- Example_design.pdf found in CSE 13s Discord.