Assignment 6

Down the Rabbit Hole and Through the Looking Glass: Bloom Filters, Hashing, and the Red Queen's Decrees

Pre-Lab Part 1

1. Write down the pseudocode for inserting and deleting elements from a Bloom filter.

```
filter_insert():
        Index1, index2, index3 = hash(all the salts of bloom)
        Set_bitvector(index1)
        Set_bitvector(index2)
        Set_bitvector(index3)
filter_delete():
        bitvector_delete(bloom filter)
        Bloom = NULL
```

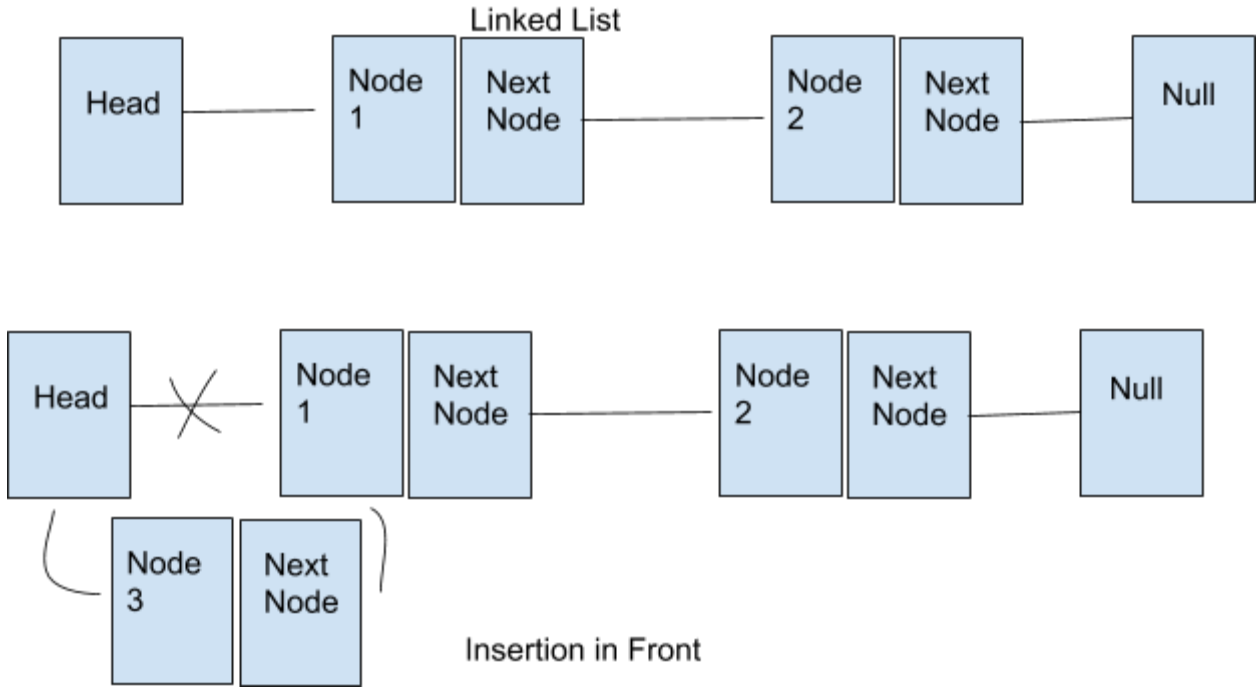
2. Assuming that you are creating a bloom filter with m bits and k hash functions, discuss its time and space complexity.

   For time complexity, bloom filters will have O(1) if not dealing with a false positive, else O(n). In terms of space complexity, bloom filters have O(nlog(n)) space complexity because they have to be large in order to reduce the chance of collisions.

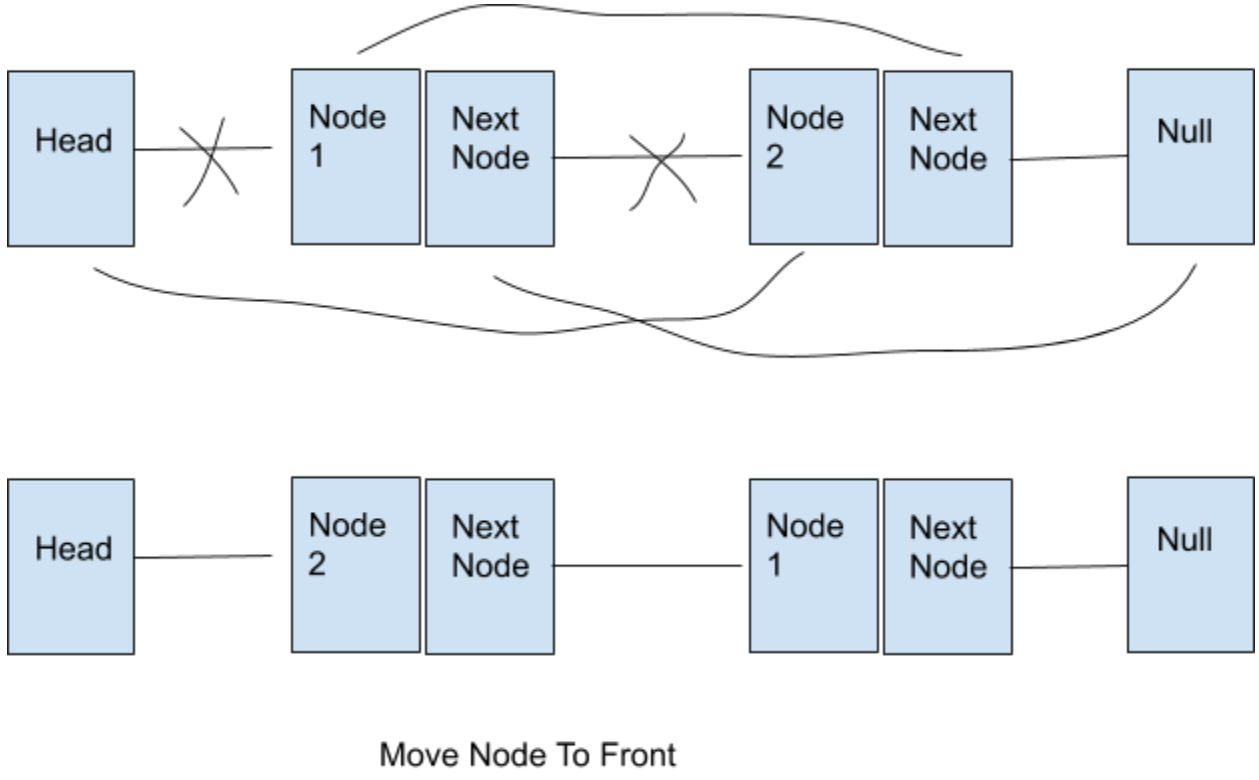
Pre-Lab Part 2

1. Draw the pictures to show the how elements are being inserted in different ways in the Linked list

Linked Lists look like this:



When applying move to front rule:



2. Write down the pseudocode for the above functions in the Linked List data type.

```
ll_node_create(gs):
        Listnode ln = Allocate space in memory to store a listnode
        Ln.hatterword = gs
        Return ln
ll_node_delete(ln):
        Free space of its hatterword
        Free ln

ll_delete(head):
         Listnode node1 = whatever head points to
        If node1 is NULL:
                Return
        While node1.next != NULL:

                Node2 = node1.next#save the next in another variable
                ll_node_delete(node1)
                Node1 = node2
        ll_node_delete(node1)
```

Program Details

Hatterspeak.c: This is the c file that produces the program executable. It defines the variables that store the seeks, average linked list length, average links seeked, and all the other statistics and passes the pointer of these variables to all the different functions in other C files in the project. Deals with the getopt() and user input commands, calls function in regex_fill.c to to fill the hash table variable with text from hatterspeak.txt and oldspeak.txt. After filling the hashtable, it creates a listnode to store all the bad words in and parses the words written by the user to see if any of them breach the rules. Finally, depending on severity of the breach, there is the command message.

Regex_fill.c: This is the C file with functions used to fill the hash table with nodes of words from oldspeak.txt and hatterspeak.txt. The function fill_hash_filter() is only meant to be used to decipher the hatterspeak.txt since using fscaf() involved more complications. The function fill_oldspeak_fscanf() is only meant for oldspeak.txt since parsing using regex expressions like in fill_hash_filter() wasn't very accurate and yielded in parsing some NULL words as well.

Ll_c: Includes all the constructor as well as all the helper functions for a linked node and linked list.

Hash.c: Includes the constructor and all the helper functions for the hash table. Constantly refers to functions in ll_c from its own from functions.

Speck.c: Given by the professor. Used to get the index to include a word in the hash table.

Program Pseudocode

Hatterspeak.c:

#include everything you need
Ht_size = 10000, bf_size = 1048578
Seeks = 0, links_transversed = 0, move_to_front = false, print_stats = false # Define all the stats variables

while(c = getopt):
    set the bool variables and size variables accordingly #getopt loop is fairly straightforward to look at
Create hash table and bloom filter variables
Fill the filter and hash table using functions in regex_fill.c

Bad_words = []
Loop in user words:
    If user word in bloom filter:
        If word in hash table:
            bad_words.append(user word)
            If user word has a next node:
                Set boolean var implying message will not be a hatterspeak message
            If user word doesn't have a nex node:
                Set boolean var implying message will not be a nontalk

if user did not want stats
    if implied not hatterspeak and not nontalk:
        Print hybrid message
        print(bad_words)
    Elif implied not hatterspeak and nontalk:
        Print nontalk message
        print(bad_words)
    Elif implied not nontalk and hatterspeak
        Print hatterspeak message
        print(bad_words)
Else:
    Print the stats

Regex_fill.c:

fill_hash_filter():

```
        F = Open file
        Word = next_word(f)
        while(word):
                Lookup in hash table
                If not already in hash table:
                        Create and add in hash table and bloom filter

fill_oldspeak_fscanf():
        F = open file
        Word = fscanf(F)
        whie(wrod){
                Lookup in hash table
                If not already in hash table:
                        Create and add in hash table and bloom filter
```

Hash.c:

```
ht_create():
        #provided by prof

ht_insert(word):
        Index = hash(salt, word) % hash_table.length
        ll_insert(index, word)

Ht_delete():
        Loop through every index in hash table:
                ll_delete(linked list of that index) #delete every linked list
        Free the heads
        Free the hash table

ht_count():
        Loop through all index in table:
                While the node's next node in linked list exists:
                        ans+=1
                        Change current node to the next node
        Return ans
```