Aleesha Nageer
anageer@ucsc.edu
11/15/2020

CSE 13S Fall 2020
Assignment 6: Down the Rabbit Hole and Through the Looking Glass:
Bloom Filters, Hashing, and the Red Queen's Decrees
Design Document

Bloom Filters optimize the time it takes to search through a hash table by eliminating the possibility of certain words being present in the table. In this lab, we utilize bloom filters and a hash table containing linked lists to pinpoint key value pairs and label them accordingly. Hash tables store these key value pair but are only so efficient if there aren't many items being placed into them. This is because the possibility of two items trying to go in one index of the table is likely, which will cause a collision. In order to prevent this from happening, this assignment has us create a linked list within each index of the hash table to prevent any collisions from occurring.

The inputs to the program are:
-s: Prints statistics for the number of seeks, average seek length, average linked list length, the hash table load, and the bloom filter load
-h <value>: Sets the hash table size
-f <value>: Sets bloom filter size
-m: Changes the move_to_front flag to true
-b: Changes the move_to_front flag to false

Pre-Lab

Part 1

1. Inserting Elements into a Bloom Filter

Get the hash values of a certain key: key mod length of bloom filter array Set this value equal to an index in the bloom filter array

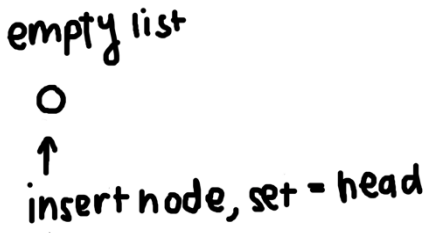
Deleting Elements from a Bloom Filter

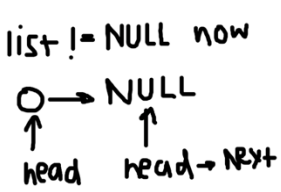
Search for key indices: key mod length of bloom filter array Clear these array indices and free the data

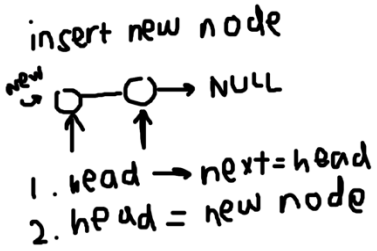
2. In creating a bloom filter with *m* bits and *k* hash functions, the time complexity would be O(k) and the space complexity would be O(m), both of which are linear due to the fact that the filter is storing the values in a singular bit making it simple to hash through and find the keys quickly.

Part 2

1.







2. Pseudocode for functions in Linked List data type:

ll_node_create(HatterSpeak *gs)

```
Allocate memory for the Node
Initialize HatterSpeak struct components in ListNode
Initialize next
Return Node
```

ll_node_delete(ListNode *n)

```
Free HatterSpeak data from node
Free the Node itself
```

ll_delete(ListNode *head)

```
Iterate through linked list (Check to see when current node equal NULL)
        Save Next node
        Delete current node
        Set Next node to current
```

ListNode *ll_insert(ListNode **head, HatterSpeak *gs)

```
Check if Node is a duplicate (lookup to see if node is already in linked list)
If it's a duplicate
        Free the HatterSpeak data from node
        Return Node
Else
        If Linked List is Empty (the head is NULL)
                Create a new node with HatterSpeak data
                Return Node
        Else
                Create a new node with HatterSpeak data
                Set current head of list to be next
                Set new node to be head
                Return Node
```

ListNode *ll_lookup (ListNode **head, char *key)

```
Iterate through list until the current node is not NULL
        Compare key and oldspeak words
        If they match
                If move_to_front is true & it's not already at the front
                        Rearrange links so that node is at the front of the list
                        Return head
                Else
                        Return current node
        Else
                Return NULL
```

Top-Level
hatterspeak.c

Main:

Read program arguments

Create flags for switch statement

Switch (method)
- Statistics
  - Set statistics flag to 1
- Hash Table Size
  - Get argument as an int
- Bloom Filter Size
  - Get argument as an int
- Move to front
  - Set move_to_front to true
- Don't move to front
  - Set move_to_front to false

Create Bloom Filter
Create Hash Table

Open oldspeak.txt
Iterate until end of file
- Insert oldspeak words into bloom filter
- Create a hatterspeak struct for each oldspeak.txt word
- Insert hatterspeak node into hash table

Close oldspeak.txt

Open hatterspeak.txt
Iterate until end of file
- Insert oldspeak words into bloom filter
- Create a hatterspeak struct for each hatterspeak.txt bad and translated word
- Insert hatterspeak node into hash table

Create regular expression
If input does not contain alphanum to start and potentially other special characters
- Expression is invalid
- Return 0

Create ListNodes for nonsense and translatable words

```
Loop through user input until it is NULL
    Make user input lower case

    Check to see if user word is potentially in hash table (bf_probe)
    If it is
        Check to see if the word is in the hash table
        If it isn't
            move on
        Else
            Check if there is a translation of word
            If there is
                Insert node in translated list
            Else
                Insert node is nonsense list
    Else
        Continue

If Statistics is 1
    Print Seeks, Average Seek Length, Average Linked List Length, Hash
    Table Load, and Bloom Filter Load
Else
    If there were translated and nonsense words
        Print specific message
        Print nonsense words
        Print translated words
    Else If there were just translated words
        Print specific message
        Print translated words
    Else If there were just nonsense words
        Print specific message
        Print nonsense words

FREE MEMORY
```

Design Process

Over the course of this lab, I hardly modified my design since my overall understanding of the what the program is doing matched the pseudocode I have, of course with the help of the TA's (specifically Eugene and Oly). However, I did have several little issues that caused me to have to debug for hours straight.

- I modified my regex multiple times
- Fixed my ll_lookup in that I was originally returning the head in both instances
- I fixed how I was inserting my nonsense and translated words

All in all, the design of this lab was not too difficult due to spending multiple hours in lab sections thoroughly understanding the specifications for this assignment.