Design

Pre-Lab Part 1

1. Functions for inserting and deleting

   Inserting

   bf_insert(bloomfilter, key)

   Hash = hash(primary hash key, key)
   Set bit at hash in bloomfilter->filter
   Hash = hash(secondary hash key, key)
   Set bit at hash in bloomfilter->filter
   Hash = hash(tertiary hash key, key)
   Set bit at hash in bloomfilter->filter

   Deleting

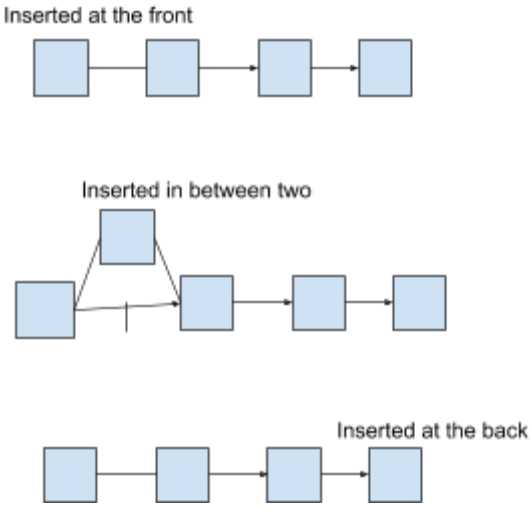
   Bf_delete(bloomfilter)

   Free bloomfilter->filter->vector
   Free bloomfilter->filter
   Free bloomfilter

2. The time complexity doesn't really change that much since it would just be linear change, it increases time complexity in a way but no huge amount since you are just increasing the amount of hashes but when doing that i would recommend increasing the size of the actual filter too.

Pre-Lab Part 2

1.



2.

ll_node_create(gs)

Allocate memory for node
Check if node is initialized
Set node->gs to gs

```
        Set node->next to Null
        Return n
ll_node_delete(node)
        Delete n->gs
        free  node
Ll delete(head)
        Set node to head
        Iterate through nodes at head
                Set next to node->next
                Call node delete of node
                Set node to next
ll_insert (head, gs)
        If linked list is NULL
                Node = ll_node_create(gs)
                Checks if node is initialized
                Sets node-> gs to gs
                Sets node->next to head
                Sets head to node
                Return node
        Delete hatterspeak struct
        Return head
ll_lookup(head, key)
        Node = head
        Prev = NULL
        Goes through the whole linked list
                Temp is oldspeak stored at node->gs
                If temp == key
                        If move to front is true
                                Set prev-next to node->next
                                node->next = head
                                Head = node
                                Return head
                        Return node
                Prev = node
                Node = node->next
        Return NULL if none match key
```

Design of Lab

Following files for this lab:

Hatterspeak.c	contains main() and linkedlist functions
Speck.c and speck.h	which has the hash functions
Hash.c and hash.h	has the functions for hash table structure
Ll.c and ll.h	has linked list functions that are used inside the hash table
Parser.c and parser.h	used to parse users input for the message that is checked
Bf.c and bf.h	has the functions for the bloom filter structure
Bv.c and bv.h from assignment 4	used for bitvector ADT used in bloomfilter
Hs.c and hs.h	has the functions to create and delete hatterspeak struct

So for the design of this lab, I started by looking at the flow of the program.

- Read in Options
- Create data structures needed
- Process both hatterspeak.txt and oldspeak.txt, the order doesn't matter
    - for oldspeak add each word into the bloom filter and into the hash table with a NULL value
    - for hatterspeak add each key into the bloom filter and into the hash table with its translation
- Parse standard input for words
    - if word is in bloom filter and in hash table
        - keep track of words found
- Display message depending on what words were found.

So this document is going to describe the design in order of the flow of the document.

Read in Options

This part is located in the file hatterspeak.c. Using getopt, it checked if option -h, -f, -s, -m, and -b. Before this I initialized the following bloom_size for size of bloom filter and hash_size. Also the bool value for stats. And the external variable move_to_front.

Create data structures needed

This part creates the hash table and bloom fitler structure where all the words are going to be stored in.

When calling ht_create, it calls this from the hash.h file.

ht_creates(length)

Allocates memory for hastable structures.

Sets salts and ht->length to length

Also allocates memory for heads of length given.

When calling bf_create it calls this from the bf.h file, allocating space and the hashes for the three hashes it is going to do when inserting a key. It uses the bv.c file to create bit vector ART in the bloomfilter

Process both hatterspeak.txt and oldspeak.txt.

It reads oldspeak.txt and hatterspeak.txt by using fopen and scanf

- **for oldspeak add each word into the bloom filter and into the hash table with a NULL value**

  By calling bf_insert it inserts the word into the bitvector. It inserts this by using the bv_insert function within the bv.c file. When inserting the oldspeak into the hashtable, I created the hs.c file where there are the two functions gs_create and gs_delete. So by **calling gs_create(oldspeak, NULL):**

  gs_create(oldspeak, NULL)

  Allocates space for hatterspeak struct
  Allocates space of length of oldspeak word and sets that to oldspeak
  Sets hatterspeak to NULL
  Return the hatterspeak struct

  After calling gs_create

  I call ht_insert(hash_table, hatterspeak struct), which creates a hash to index it in. it call ll_insert to insert the gs structure at the linke list at that index.

  When calling ll_insert:

  ll_insert (head, gs)

  If linked list is NULL
  Node = ll_node_create(gs) (this can be referenced in prelab)
  Checks if node is initialized
  Sets node-> gs to gs
  Sets node->next to head
  Sets head to node
  Return node
  Delete hatterspeak struct (this call the gs_delete function defined in hs.c)
  Return head

- **for hatterspeak add each key into the bloom filter and into the hash table with its translation**

  By calling bf_insert it inserts the word into the bitvector. It inserts this by using the bv_insert function within the bv.c file. When inserting the oldspeak into the hashtable, I created the hs.c file where there are the two functions gs_create and gs_delete. So by **calling gs_create(oldspeak, hatterspeak):**

  gs_create(oldspeak, hatterspeak)

  Allocates space for hatterspeak struct

Allocates space of length of oldspeak word and sets that to oldspeak
Allocates space of length of hatterspeak word and sets that to hatterspeak
Return the hatterspeak struct

After calling gs_create

I call ht_insert(hash_table, hatterspeak struct), which creates a hash to index it in. it calls ll_insert to insert the gs structure at the linke list at that index.

When calling ll_insert:

ll_insert (head, gs)

If linked list is NULL

Node = ll_node_create(gs) (this can be referenced in prelab)
Checks if node is initialized
Sets node-> gs to gs
Sets node->next to head
Sets head to node
Return node

Delete hatterspeak struct (this call the gs_delete function defined in hs.c)
Return head

Parse standard input for words

Using the parser.c and regex.h library, we can use regex to parse through the stdin imputed by the user when running hatterspeak. First by setting infile to stdin and than running the using the next_word function from parser.c. Than creating while loop to parse through the input of the user. It than loops through each word and sets them to lower cases using tolower(). It than check if word is hashed in bloomfilter making runtime way faster and than checking if it is hashtable if so add it to non_talk linkedlist or hatter_speak linked list depending on if it has gs->hatterspeak using ll_lookup(psuedo in prelab).

Creating linkedlist non_talk struct
Creasting hatter_speak struct
Regex_t re;
Char new_word
While new_word == next_word(infile &re)

If in bloomfilter

If in hashtable

If it doesn't have a hatterspeak translate

Push to linkedlist non_talk

Else

Push to linkedlist hatter_speak

Close infile
clear_words()
regfree(&re)

For keeping track of the words that are either non_talk or old_speak, i created a new structure at the beginning of hatterspeak.c and lookup and push functions, these are basically the same as ll_lookup and ll_insert but take a word instead of a gs structure.

Display message depending on what words were found

If statistics is requested, there will never be an error message printed with what the user did wrong. It will just print, seeks, average seek length, average linked list length, hash table load, bloom filter load. Else it will check if only oldpseak

If statistics is true

Print “seeks:” seeks(which is counted by an external variable everytime ll_lookup is called)

Avg_links = links / seeks

(links, an external variable is counted by everytime an node in ll_lookup is accessed)

“Print average seek length:” avg_links

Avg_ll = sum_ll / hash_size

(hash_size is determined by the beginning or changed by user)

(sum_ll is incremented everytime a node is created)

Print “average link list length:” avg_ll

Ht_load_ratio = (ht_count(hash_table) / hash_size) * 100(in percentage)

(ht_count in hash.c that counts the amount of used with in the hast_table)

Print “hash table load:” ht_load_ratio “%”

Bf_counter = 0

For i in range bloom_size((hash_size is determined by the beginning or changed by user)

If bv_get_bit(bloomfiler->filter, i) == 1

Bf_counter += 1

Bf_load_ratio = (bf_counter / boom_size) * 100 (in percentage)

Print “bloom filter load:” bf_load_ratio “%”

Else

If non_talk != NULL & hatter_speak == NULL

Print error message given in lab document

Print “your errors”

print(non_talk, hash_table)

Else If non_talk == NULL & hatter_speak != NULL

Print error message given in lab document\

Print “"The list shows how to turn the oldspeak words into hatterspeak.”

print(hatter_speak, hash_table)

Else If non_talk == NULL & hatter_speak == NULL

Print error message given in lab document

```
        Print “your errors”
        print(non_talk, hash_table)
        Print “Appropriate hatterspeak translations.”
        print(hatter_speak, hash_table)
```

The print function is defined in the beginning of hatterspeak.c and prints the non_talk or hattespeak with translation

```
print(head, hash_table)
    Node = head
    While n is not NULL
        Set hash_node to the node returned from ht_lookup
        If hatterspeak in hash_node->gs is NULL
            Print node->word
        Else
            Print node->word “->” hash_node->gs
        Node = node->next
```

At the end of the code can’t forget to free memory.

delete_linked(hatter_speak)
delete_linked(non_talk)
(delete_linked()is almost the same a ll_delete without the hatterspeak structure involved)

bf_delete(bloom_filter)(from bf.c and uses bv_delete to also free the bit vector ADT)
ht_delete(hash_tanle)(from hash.c and uses ll_delete and ll_node delete to free the linked list at each head)