William Santosa
wsantosa@ucsc.edu
28 February 2021

CSE13S Winter 2021
Assignment 6 : The Great Firewall of Santa Cruz: Bloom Filters, Linked Lists and Hash Tables
Design Document

PRELAB

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

```
void bf_insert(BloomFilter *bf, char *oldspeak){
        Point toward bf -> filter -> vector
        Filter oldspeak through primary, secondary, tertiary
        Set the indices of vector using output of hash functions (should have 3, primary,
secondary, tertiary)
}

void bf_delete(BloomFilter **bf){
        Free elements in primary, secondary, tertiary
        Free primary, secondary, tertiary
        Free BitVector filter (length, vector)
        Free BloomFilter bf
}
```

IMPORTANT NOTE :

Deleting a bit from a bloom filter is not advised because that bit could represent multiple words, not just one single word. As such, deleting a bit would break the BloomFilter as all of the words that the deleted bit represents will not be found if trying to look it up. Thus, the BloomFilter would incorrectly state that the word is not in the filter.

Reference:

1. https://piazza.com/class/khyix5qk2sw2nm?cid=961
2. https://www.youtube.com/watch?v=Bay3X9PAX5k&feature=youtu.be&ab_channel=TechDummiesNarendraL

2. Write the pseudocode for each of the functions in the interface for the Linked List ADT

```
LinkedList **ll_create(bool mtf){
        Initialize Head and Tail as Node
        Head -> next = Tail
        Head -> prev = Null
```

```
        Tail -> prev = Head
        Tail -> next = Null
}

uint32_t ll_length(LinkedList *ll){
        Return ll -> length
}

Node *ll_lookup(LinkedList *ll, char *oldspeak){
        Iterate through LinkedList from head until tail (Check from head until null)
        See if oldspeak is within a node, save address
        If ll move to front, change node address to before the head, alter next and prev using
temporary pointer
        Return pointer to node with oldspeak
        If none, return NULL
}

void ll_insert(LinkedList *ll, char *oldspeak, char *newspeak){
        If ll_lookup(ll, oldspeak) returns node (not NULL){
                Return
        } else {
                Create node with oldspeak and newspeak
                Set node prev to head
                Set node next to the former starting node
                Set forming starting node prev to new node
        }
        Return
}

void ll_print(LinkedList *ll){
        Iterate through LinkedList from head until tail (Check from head until null)
                node_print(array)
}
```

3. Write down the regular expression you will use to match words with. It should match hyphenations and concatenations as well.

```
[a-zA-Z0-9]+\-?\'?
```

DESCRIPTION

Oddly, I, along with some other students, have been notified that we have been selected "through thoroughly democratic processes" to be the leader of the Glorious People's Republic of Santa Cruz. As such, in an attempt to promote virtue, prevent vice, and preserve social cohesion, I have decided to filter internet content by implementing a variety of different methods using the files listed below.

Files:

- speck.{c, h}
    - Specifies the interface and implementation of the SPECK cipher
- hash.{c, h}
    - Specifies the interface and implementation of Hash Table ADT
- ll.{c, h}
    - Specifies the interface and implementation of Linked List ADT
- node.{c, h}
    - Specifies the interface and implementation of Node ADT
- bf.{c, h}
    - Specifies the interface and implementation of Bloom Filter ADT
- bv.{c, h}
    - Specifies the interface and implementation of Bit Vector ADT
- parser.{c, h}
    - Specifies the interface and implementation of Regex Parsing module
- banhammer.c
    - Contains main() and other things needed to complete the assignment
- Makefile
    - Runs program and creates an executable named banhammer
- README.md
    - Information about building, running, and options of the program
- DESIGN.pdf
    - Describes purpose, covers the layout, clear description of program parts, pseudo code, and contains the pre lab questions.

TOP LEVEL DESIGN / PSEUDOCODE

node.c

```
create(oldspeak, newspeak){
Node n with malloc
If not null:
        Set old speak and next speak to arguments
        Next and prev to null
Return Node
}

delete(node){
Free oldspeak and newspeak
Free node
}

print(node){
Check newspeak and oldspeak not null:
        Print first case
Check newspeak null and oldspeak not null:
        Print second case
}
```

ll.c

```
create(mtf){
LinkedList with malloc
If not null:
        Set stuff
Return LinkedList
}

delete(LinkedList){
Same as above
}

lookup(LinkedList, oldspeak){
Go through everything in LinkedList:
        Return node pointer
Return NULL
}

insert(LinkedList, oldspeak, newspeak){
```

Check if oldspeak is in the LinkedList already:
    Return
Create node
Node next to head next
Node prev to head
Head next previous to node
Head next to node
Increment length
}

print(LinkedList){
Print using node_print until null
}

hash.c

create(size, mtf){
INCLUDED IN LAB DOC
}

delete(HashTable){
Delete each list LinkedList
Delete list
Delete HashTable
}

lookup(HashTable, oldspeak){
Get index by doing hash % size
Set node to ll_lookup(index, oldspeak)
Return Node
}

insert(Hashtable, Oldspeak){
Same as above but initialize if not there
Insert instead of return pointer
}

print(HashTable){
Print using ll_print until NULL
}

bv.c

Everything is the same as BitMatrix (asgn4) except it's 1 dimensional instead of 2 dimensional

bf.c

```
create(size){
INCLUDED IN LAB DOC
}

delete(BloomFilter){
Delete filter using bv_delete
Free bf
}

insert(BloomFilter, oldspeak){
Hash everything
Set bits
}

probe(BloomFilter, oldspeak){
Hash everything
Check if bits == 0:
        Return false
Return true
}

print(BloomFilter){
        Print using bv_print
}
```

banhammer.c

```
Include everything

Get opt:
        Switches for each:
                Set to true/false or set number
Initialize BF, HT, thoughtcrime, rightspeak
Check if any of them are null
Open badspeak/newspeak.txt
Check if null
fscanf to take from badspeak and insert into bf/ht
fscanf to take from newspeak and insert into bf/ht
Check if standard input words match the stuff in bf/ht
Print out messages
Free memory
```

NOTE : Will definitely be altered (Listed below)

DESIGN PROCESS / MODIFICATIONS

- Changed regex to ([a-zA-Z0-9]+\-?\'?)+
    - Changed again ([a-zA-Z0-9]+\-?\'?_?)+
    - Had to get rid of backslashes “\” in the C code
        - ([a-zA-Z0-9]+-?'?_?)+
- Error cases (No badspeak/newspeak.txt, invalid hash/bloom size, default, etc)
- Set pointers to NULL after freeing
- Surprisingly straightforward lab