Caterin Duarte Reynosa
cduarter@ucsc.edu

CSE13S Fall 2020
Assignment 6: The Great Firewall of Santa Cruz:
Design Document

Goal

The goal for this assignment is to use hashtables, linked lists, and bloom filters to fix corrupted illegal words, oldspeak, and turn them into acceptable words and to newspeak, or decide if the word requires a punishment. In our program we will be reading words from a file and checking if the words are valid words or not. If they are not valid, we will translate them to newspeak.

PreLab Questions

1.) Write down the pseudocode for inserting and deleting elements from a Bloom filter.
   a.) void bf_insert(BloomFilter *bf, char *oldspeak)
      i.) h1 =hash(bf->primary, oldspeak)
      ii.) h2=hash(bf->secondary, oldspeak)
      iii.) h3=hash(bf->tertiary, oldspeak)
      iv.) bv_set(h1)
      v.) bv_set(h2)
      vi.) bv_set(h3)
   b.) After reading the piazza discussion from the professor and watching a few youtube videos explaining what bloom filters are, I learned that we don't want to delete any elements from a bloom filter because if we clear an element, it might accidentally clear other elements that we didn't mean to clear and cause some issues with our elements. If we had to delete an element what we could do is create a new bloom filter and insert the element we want to set to the second bloom filter. That way, if we have an element that is present in both bloom filters it is considered 'deleted'.
2.) Write down the pseudocode for each of the functions in the interface for the linked list ADT
   a.) LinkedList *ll_create(bool mtf)
      i.) Linkedlist * L = (Linkedlist *)calloc(1, sizeof(LinkedList))
      ii.) length = 0
      iii.) Mtf =bool mtf
      iv.) Head = node_create()
      v.) Tail = node_create()
      vi.) Return L
   b.) void ll_delete(LinkedList **ll)
      i.) For i in range(i, node_size)
         (1) Delete node

ii.) Free ll
iii.) *L = NULL

c.) uint32_t ll_length(LinkedList *ll)
i.) Return ll length

d.) Node *ll_lookup(LinkedList *ll, char *oldspeak)
i.) For (node *n = l-> head->next; n != tail: n-> next)
(1) Compare old and new speak
(a) If mtf is true
(i) Previous = next
(ii) Next = previous
(iii) Next = next
(iv) Head (next and tail)= x
(v) Head (next) = x
(vi) Return n
(b) Return 0

e.) void ll_insert(LinkedList *ll, char *oldspeak, char *newspeak)
i.) *n = node_create(oldspeak, newspeak)
ii.) Next = L->head
iii.) Prey - L->head
iv.) Length +=1

f.) void ll_print(LinkedList *ll)
i.) For i in range(1, ll_size)
(1) node_print()

g.) Node *node_create(char *oldspeak, char *newspeak)
i.) Node *n = (Node*)calloc(1, sizeof(Node))
ii.) n->Oldspeak = strdup(oldspeak)
iii.) n->newspeak = strdup(newspeak)
iv.) Next = null
v.) Prev = null

h.) void node_delete(Node **n)
i.) free(oldspeak)
ii.) free(newspeak)
iii.) Free (*n)
iv.) *n = null

i.) void node_print(Node *n)
i.) If node != null in newspeak & oldspeak
(1) printf("%s -> %s\n", n->oldspeak , n->newspeak);
ii.) Else
(1) printf("%s\n", n->oldspeak);

3.) Write down the regular expression you will use to match words with. It should match hyphenations and concatenations as well.
    a.) regularExpressions “[a-z_A-Z_0-9_]”+ ((‘|-)[a-z_A-Z_0-9_]+)*”

Files I need to Implement + pseudocode

1.) Hash.c
    a.) HashTable *ht_create(uint32_t size, bool mtf)
        i.) given
    b.) void ht_delete(HashTable **ht)
        i.) For i in range of (1, ht_size)
            (1) free(lists)
        ii.) Free(ht)
        iii.) *ht = null
    c.) uint32_t ht_size(HashTable *ht)
        i.) Return ht_size
    d.) Node *ht_lookup(HashTable *ht, char *oldspeak)
        i.) Create a hash for the location
        ii.) If LL doesn’t exist
            (1) Create it
        iii.) x= ll_lookup(hash_location, oldspeak)
        iv.) If x if found
            (1) Return x
        v.) Else
            (1) Return null

    e.) void ht_insert(HashTable *ht, char *oldspeak, char *newspeak)
        i.) Create a hash for the location
        ii.) Ht = ll_insert(ht, oldpeak, newspeak)
    f.) void ht_print(HashTable *ht)
        i.) For i in range(i, length of bt)
            (1) Print h[i]

2.) Ll.c
    a.) LinkedList *ll_create(bool mtf)
        i.) Linkedlist * L = (Linkedlist *)calloc(1, sizeof(LinkedList))
        ii.) length = 0
        iii.) Mtf =bool mtf
        iv.) Head = node_create()
        v.) Tail = node_create()
        vi.) Return L
    b.) void ll_delete(LinkedList **ll)
        i.) For i in range(i, node_size)

(1) Delete node

ii.) Free ll

iii.) *L = NULL

c.) uint32_t ll_length(LinkedList *ll)

i.) Return ll length

d.) Node *ll_lookup(LinkedList *ll, char *oldspeak)

i.) For (node *n = l-> head->next; n != tail: n-> next)

(1) Compare old and new speak

(a) If mtf is true

(i) Previous = next

(ii) Next = previous

(iii) Next = next

(iv) Head (next and tail)= x

(v) Head (next) = x

(vi) Return n

(b) Return 0

e.) void ll_insert(LinkedList *ll, char *oldspeak, char *newspeak)

i.) *n = node_create(oldspeak, newspeak)

ii.) Next = L->head

iii.) Prey - L->head

iv.) Length +=1

f.) void ll_print(LinkedList *ll)

i.) For i in range(1, ll_size)

(1) node_print()

3.) Node.c

a.) Node *node_create(char *oldspeak, char *newspeak)

i.) Node *n = (Node*)calloc(1, sizeof(Node))

ii.) n->Oldspeak = strdup(oldspeak)

iii.) n->newspeak = strdup(newspeak)

iv.) Next = null

v.) Prev = null

b.) void node_delete(Node **n)

i.) free(oldspeak)

ii.) free(newspeak)

iii.) Free (*n)

iv.) *n = null

c.) void node_print(Node *n)

i.) If node != null in newspeak & oldspeak

(1) printf("%s -> %s\n", n->oldspeak , n->newspeak);

ii.) Else
(1) printf("%s\n", n->oldspeak);

4.) Bf.c
a.) This file contains the implementation of the bloom filter data structure. The functions in this file will aid to add oldspeak words into the bloom filter and later on check punish the citizens for using oldspeak
b.) BloomFilter *bf_create(uint32_t size)
i.) Given
c.) void bf_delete(BloomFilter **bf)
i.) Free filter
ii.) Free bf
iii.) *bf = null
d.) uint32_t bf_size(BloomFilter *bf)
i.) Return bv_length(bf)
e.) void bf_insert(BloomFilter *bf, char *oldspeak)
i.) H1 =hash(bf->primary, oldspeak)
ii.) H2 =hash(bf->secondary, oldspeak)
iii.) H3 =hash(bf->tertiary, oldspeak)
iv.) Bv_set H1 and H2 and H3
f.) bool bf_probe(BloomFilter *bf, char *oldspeak)
i.) H1 =hash(bf->primary, oldspeak)
ii.) H2 =hash(bf->secondary, oldspeak)
iii.) H3 =hash(bf->tertiary, oldspeak)
iv.) Bv_get H1 and H2 and H3
v.) If all true == 1
(1) Return true
vi.) Else
vii.) return false
g.) void bf_print(BloomFilter *bf)
i.) For i in range(0, len(bf))
(1) ll_print

5.) Vb.c
a.) BitVector *bv_create(uint32_t length)
i.) BitVector v * = (BitVector *) calloc(1, sizeof(BitVector))
ii.) u-> length = length
iii.) If allocation fails,
(1) return null
iv.) Else
v.) Return v

b.) void bv_delete(BitVector **bv)
    i.) Free v
c.) uint32_t bv_length(BitVector *bv)
    i.) Return length
d.) void bv_set_bit(BitVector *bv, uint32_t i)
    i.) mask = 1 << ((i % 8);
    ii.) vector[i/8] |= maks
e.) void bv_clr_bit(BitVector *bv, uint32_t i)
    i.) mask by shifting
    ii.) vector[i/8] &= maks
f.) uint8_t bv_get_bit(BitVector *bv, uint32_t i)
    i.) mask = 1 << ((i % 8);
    ii.) Mask & bv(i/8) >> bv(i/8)
g.) void bv_print(BitVector *bv)
    i.) For i in range(0, length)
        (1) Print get_bit

6.) Parser.c
    a.) Given to us
7.) Bangammer.c
    a.) Main function that creates the ht, bf, and two LL
    b.) Reads in from files and takes in command line options

UPDATES

I actually ended up changing a lot of things for a lot of my assignments. I mostly change my functions for the bf ADT and the ht ADT. Since we didn't have any lectures on these before the design document was due, I wasn't able to understand a lot of the ADT. After the lectures, I was able to implement the ADTS. This was a really fun asgn. I really enjoyed seeing all the ADTs come together. It was a little tricky to debug since there were so many ADTS, the leeks could come from anywhere. Overall, I think this asgn made me more confident in ADTS and how to implement different kinds of them.