Artyom Martirosyan
11/24/2019
Assignment 7 Design

V.0

For this assignment we are taking in a text file and finding words that are unacceptable(badspeak) and words which should be converted into ‘newspeak’(oldspeak). To start off I am going to use flex in order to read newspeak words as well as the oldspeak words(note doing this one document at a time). To start off are going to read the words and put them into both a linked list and hash tables. The hash tables will only hold words which are ‘badspeak’ and ‘oldspeak’ whereas the linked list will hold newspeak and oldspeak (having the goodspeak hold the words for oldspeak and their newspeak). The linked list will also hold the badspeak words(holding NULL before each badspeak word). By using two linked lists for the bloom filter the program will have a lower probability for a false negative, if the word is flagged it will run through the linked list in order to check and what the replacement is or if it is badspeak. The pseudocode is given below:

Hash functions:

Ht_create:

Hashtable *h;
H =(HashTable *)malloc(size(hashTable) + sizeof(linked list) * length)
Create linked lists
H->length = length
Return h

Ht_delete:

Loop through deleting linked lists
Set length to zero
Free h

List node ht lookup:

Run keyed hash on key
Run linked lookup with heads key
If found return finding if not return null

Ht_insert:

While[h[heads]]{
heads++
}
H =
h
Ll insert (ht for head, gs for goodspeak)

Hash:

Code provided by dl

Bloom filter function:

Bf_delete:

```
Free filter
Filter = null
Free bf
Filter = null
```

Bf_insert:

```
bf->primary[location] = 1;
bf->secondary[location] = 1;
```

Bf_probe:

```
If bf->primary[key] && bf->secondary[key]
{
        Return true;
}
Else return false
```

Linked list function:

ListNode *ll_node_create:

```
Node *p = (node *)malloc(sizeofnode)
if(p) {
        P ->gs = goodspeak
        p->next = (node *) 0
}
Return p
}
```

Ll_node_delete:(*Node)

```
if (node ->next = NULL) {
  free(node->gs)
  free(node->next)    //need to look at
  node->next = null
}
If (node -> next = NUll) {
Node *temp = node->next
free(node->gs)
free(node->next)    //need to look at
node->gs=temp->gs
node->next=temp->next
```

LL_delete:

```
        While q {
        Run delete node
        Node *t = q->next
        free(q->key)
        free(q->next)
        free(q)
        Q = t
        }
LL_insert:
        Q = node create("name")
        Node create();
        Q ->next = p
        P = q
LL_Lookup:
        Char q = p
        P = p->next
        T = q ->key
        Free (q)
        Return T
```

Goodspeak.c:
-get command line arguments
-Run flex and get the criteria/words for badspeak and newspeak and create the hash tables as well as the linked lists with those values.
        -create goodspeak struct where the for newspeak it holds the oldspeak word and the replacement, for badspeak it holds the badspeak word and null
-Run flex in a loop(running one word at a time) and check each value against the bloom filter(probing).
-Print responses.

V2.0

To start off I am going to do the linked list and test it then make and test the bloom filter and finally make and test the hash table I will have more in depth pseudo code below, I also made a seperate file holding the struct for goodspeak this makes the program a bit easier to read as there is only one definition rather than having one for each program.

Linked list function:

-Note I am sourcing professor Long for this as he provided most of the linked list Code I have also added a print linked list for a later purpose.

Extern variables:
        count(number of links in the list)
        distance(how far i went into each list)

Move_to_front

Note: Header has header for each function as well as struct holding goodspeak and node next

```
ListNode *ll_node_create:
        Node *p = (node *)malloc(sizeofnode)
        if(p) {
                P ->gs = goodspeak
                p->next = (node *) 0
                 Return node
        }
        Return false
        }
Ll_node_delete:(*Node)
note: when initially making this I had assumed this was going to be a double linked list
        If there is a node(check gs)
                free goodspeak struct
        Free node



Void LL_delete(ListNode *n):
        While q {
        Run delete node
        Node *t = q
        Run delete node(q)
        q = q->next
        }
ListNode *ll_insert(ListNode **head, GoodSpeak *gs);:
        If node is already there return that node(ll_lookup)
                Also need to do delete gs of the repeat
        Else
        Make a new node n
        n->head = head
        Return n
ListNode *ll_lookup(ListNode **head, const char *key):
        Make a Null node called *t
        for(while there are stills nodes (while n != null))
                Do a string compare(if(strcomp…))
                        If move to front is true move set t = n->next
                                And insert n in front of list
                                Return n
                        If no move to front rule just return N
```

Void LL_print(ListNode *n):

for(while there are still nodes)

If node->gs->newspeak = NULL

Print out as badword

Else print as good word

Hash functions:

Extern Variables:

None needed

In header will make headers for each function as well as a struct for:

Length, salt, and **heads

HashTable Ht_create(uint32_t length):

Code provided by dl

Malloc memory for hashtable

Make salts and set length

Allocate memory for heads

Void Ht_delete(HashTable *h):

Loop through deleting linked lists

Set length to zero

Free h

List node ht lookup(HashTable *ht, char *key):

Run keyed hash on key mod by length

Run linked lookup with heads key

If found return finding if not return null

Void Ht_insert(HashTable *ht, GoodSpeak *gs):

Run keyed hash on key mod by length

Do an ll_insert at that position

Hash/spekc.c:

Code provided by dl

Goodspeak functions:

Note you need to make a struct in header

GoodSpeak *make_good_speak(char *oldspeak, char *newspeak):

Allocate the memory for gs(malloc)

Strdup oldspeak

If there is a newspeak strdup as well else set it to NUll

Return gs

void delete_goodspeak(GoodSpeak *g)

First empty the water in the bucket(if there is an oldspeak free it)

Then throw away the bucket(free gs)

Bloom filter function:

Extern variables:

bfones

BloomFilter *bf_create(uint32_t size):

Note this code was provided to us by professor long
Allocate memory for bf
Set the salts to something
bf->filter = make bit vector(note this was brought back from a previous assignment, but
For the chance that it is required there is design for bv below)
Return bf

Void Bf_delete(bf):

Delete the bitvector(has a function)
Free bf
Set to null

Void Bf_insert(bf, word):

Bv_set bit(bf->filter, hash(salt, word) % bv_get_length)
Do this for all three salts
Bfones += 3;

Bool Bf_probe(bf, word):
If bf_get_bit(bf->filter, hash(salt, word) % bv_get_length) && same for other two salts
{

Return true;

}
Else return false

Bit Vector functions:

Struct:

Uint 8 v->vector
Uint 32 v->length

BitVector *bv_create(uint32_t bit_len):

Malloc memory for bv
int byte length
If the length given is not divisible by 8 add 1
Calloc v->vector
v->length = bit_len

void bv_delete(BitVector *v):

Free v->vector
Free v

uint32_t bv_get_len(BitVector *v)
    Return v->length(so that's what this is used for)

void bv_set_bit(BitVector *v, uint32_t i)
    v->vector[i / 8] |= (0x1 << (i % 8));

void bv_clr_bit(BitVector *v, uint32_t i)
    v->vector[i / 8] &= ~(0x1 << (i % 8));

uint8_t bv_get_bit(BitVector *v, uint32_t i)
    uint8_t bit = (v->vector[i / 8] & (0x1 << (i % 8))) >> (i % 8);

void bv_set_all_bits(BitVector *v)
    int byte length
    If the length given is not divisible by 8 add 1
    memset(v->vector, byte_length)

There is also a file called words.l which was provided in which all you do is define what a letter, digit, and punctuation is for the lexical analyzer

For the main function I started off by creating the larger components(hashtable and bloom filter) and later on I created two list nodes to hold either the old/new or bad words I also had to bools which would be activated if something is inserted into the two nodes. In order to read

Newspeak.c

Global variables:

    Move_to_front
    Bfones
    Count
    Distance
    Also need all of these for using yylex:extern FILE *yyin; extern char *yytext; extern int yylex(void); extern int yylex_destroy(void);
    Void main(...) {
    Make an int called falsepositive
    uint32_t words
    Uint32_t trans
    Uint32_t text
    Uint32_t hash size = 10000
    Uint 32_t bloom size = 1048576
    Make a bool called move_to_front
    Do get-opt("smbh:f:") s = statistics, m = move to front, b = dont move to front, f for bloom filter -I also checked if the numbers for the bloom filter and hash table were negative of zero.

```
Next I make the hashtable and the bloomfilter
fopen(oldspeak.txt, r)
while(yylex() != EOF){
        Make a temp char * and strdup yytext
        Do the same for a second one(need to call yylex() again)
        Insert the first temp into the bloomfilter
        Make a goodspeak struct holding both words
        Store that into the hashtable
        Free both temp vars
        Words++
}
fclose(yylex)
Repeat the same process but opening badspeak.txt
(note now you only call one temp rather then doing the same for the second one you set
to null)
fclose(badspeak.txt)
Listnode oldnew and bad = NULL
Yyin = stdin
while(yylex() != EOF) {
        Make a tempchar which holds strdup(...)
        text++
        if(probe bf with the tempchar)
                if(ht_lookup != NULL)
                        if(ht_lookup(h, temp)->gs->new_speak == NIL)
                                Make two temps and strdup the contents of the hashtable
                                        (note you should just set the new_speak to null)
                                Make a new gs holding the two temps
                                Bad = ll_insert(&bad, newgs)
                                Badopt = true
                                Free temps
                        Else
                                Same logic expect for oldnew
                trans++
}
                else(false positive+=1)
        Free tempchar
}

if (!badoption && !suppress && goodoption) {
        Print a small phrase and do ll_print(oldnnew)
}

if (badoption && !suppress) {
```

```
        Print a small phrase and do ll_print(bad)
        if (goodoption) {
                Print small phrase with ll_print(oldnew)
        }
}
if (stats && count > 0) {
        Print distance, distance/count
        Print number of words their translations and text
        Print ones in bf and lengthofbf, bf/lengthofbf
}
If theres something in oldnew do lldelete
Same with bad
Delete hashtable
Delete bloomfilter
yylex_destroy();
Return 0;
```