DESIGN

Purpose: The purpose of assignment 7 or "The Great Firewall" is to take in a list of badspeak words and newspeak words and determine whether any given user input violates those outline transgressions. To accomplish this system our directory makes use of a bloom filter, hash table, and linked lists of nodes.

Layout / Structure

Bloom Filter - ADT to determine if a word by the user is definetly not a bad word or probably is a bad word. It uses a bitvector as its "Filter". If a word has 3 set bits in the three indexes assigned to it then it probably is a bad word. If it is missing one then it definetly isn't. This is because bad words can share indices, thus "false positives" occur.

Its attributes are 3 salts and a BitVector type filter.

bf insert() - Function that generates 3 indices & sets a bit at all three indices. (for a word)

Pseudocode

```
index1, index2, index3 = get.index(), get.index(), get.index()
setbit(index1, index2, index3).
```

bf_probe() • Function to check the appropriate indices generated from the word

Pseudocode

```
index1, index2, index3 = getindex(), getindex(), getindex()
getbit(index1, index2, index3)
if (getbit() == true):
    return true
else {
    return false
```

• to return true all indices must have a bit set.

bf_count() • Returns number of bits set.

```
count = 0
for bit in bits:
    if b == 1:
        count += 1
return count
```

Bit Vector · has components length (of bits) and a vector which is an array of unsigned integers that has memory allocated precisely according to the byte length.

· We calculate byte length by

Byte length = $x / 8$ (if x is divisible by 8)

Byte length = $(x/8)+1$ (if x is not divisible by 8)

For example $\underline{x = 17 \text{ bits}}$

Byte length = $\left(\frac{17}{8}\right)+1 = 3$ bytes.

We must waste a little memory since we must store a single bit in one whole byte.

Set_Bit - We set a bit by finding the index of the vector we'd like to set and then OR ing it w/ a 1 left-shifted to p

If we want to set 3rd bit of 0000 0101 ↑

then:

OR 0001 0000 = 1001 0101

Note: OR operation preserves all other bits b/c OR w/ anything returns identity.

Clr_bit - Similarly to clear a bit we use AND operation w/ not and a left shifted 1

1000 0101 ↑ 0000 0101

AND 0111 1111

0000 0101

Note: All zeroes will stay zero when AND'd w/ 1 and the 1 we wish to clear will turn to ze

Get_bit - We right shift our vector by the index AND'd w/ 1.

Xor_bit - We locate index and add the bit we wish to xor w/

- We Modulo 2 to ensure that if it were two 1's to be zero

100
100
000

capacity: 7
14 : 4 7 - 4 = 3

Hash Table • ADT to check if a word is for certain a bad word
• Uses linked lists ~~and an array to store~~ inside of a linked list

ht_lookup() • we get index generated w/ the word and look for the appropriate list

Pseudocode

```
index = getIndex(word)
if (list[index] == NULL);
    return NULL
ll_lookup(list[index], word)
Node = ll_lookup()
return Node
```

• If we don't have a list we return NULL. Ultimately we look through the list to find our appropriate node

ht_insert() • Inserts a node for specified word

Pseudocode

```
index = getIndex(word)
if (list[index] == NULL);
    list[index] = newList()
ll_insert(list[index])
```

• If a list does not exist we create one
• Ultimately we use ll_insert() to insert the new node into the linked list

ht_count() • We wish to return the amount of non-NULL lists ~~[illegible]~~ in our list.

Linked List • ADT is used by hash table & it uses nodes to link together a linked lists,
• It has two sentinel nodes; Head & tail
•

ll_lookup() • Wish to check if a given bad word is in the linked list

Psudocode

```
for node in nodes:
    if node.word == word:
        return true
return false
```

ll_insert() • Wish to insert a new word in node form at the head of list. In front of head.

Psudocode

```
node.next = head.next
head.next = head
node.prev = head
node.next.prev = node
```

• Special case if it's just Head & tail

```
head.next = node
tail.prev = node
node.prev = head
node.next = tail
```

Node - ADT used by linked list ADT
- Can tell us what is next, previos, and the word/new translation

node_create()

Pseudocode

```
node.word = word
node.new = new
```

banhammer.c • This file is the main file that uses all modules and ADT's to determine transgressions & whether a user violated them.

• We begin by opening badspeak.txt & newspeak.txt and add them to our Bloom Filter & Hash Table

• ~~How we~~ We add them using the API functions bf_insert() & ht_insert()

• Now we parse through a user's message using our parser.c module. Our regular expression is "[[a-zA-Z]]*([-']*[[a-zA-Z0-9]])+" this insures that we accept letters w/ dashes, and apostrophes as a word.

• We initialized two new linked list in which we will store the words that the user typed that is a transgression of bad words or/and bad words w/ translations

• Depending on our lists we will present what transgressions the user committed

Psuedocode

```
open(badspeak.txt)
open(newspeak.txt)
bf = Bloom_filter()
ht = Hash_table()
for word in badspeak.txt:
    bf_insert(word)
    ht_insert(wad);
for word in newspeak.txt:
    bf_insert(word)
    ht_insert(word)
Linkedlist thought, correction

for word in UserInput.txt:
    if(word == bf_probe):
        If( bf_probe(word) == true):
            if( word.newspeak != NULL):
                Correction_insert(word)
        else:
            thought_insert(word)
If (thought != 0 && correction != 0):
    Print(Message)
    print(thought)
    print(correction)
If (correction != 0):
    print(message)
    print(correction)
else:
    print(message)
    print(thought)
```

BadSpeak.txt

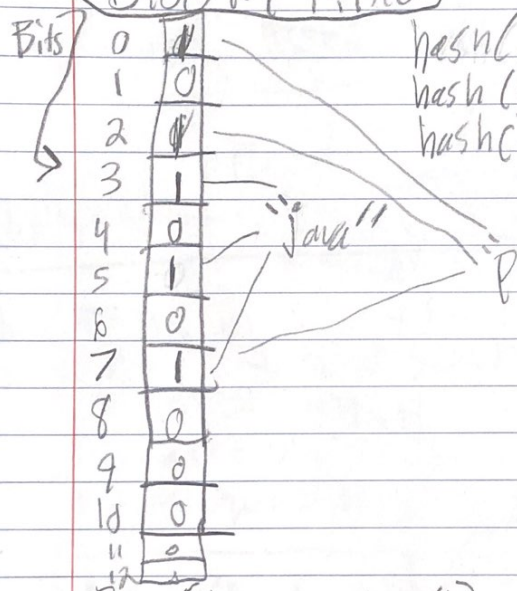
Java
Python
bash
Vscode

Newspeak.txt

covid → plague
C++ → C--
banana → berry

1.7 Make Bloom Filter
- An array of bits w/ 3 hash funcs

Bloom Filter



hash(pui, "python") => 0
hash(sec, "python") => 2
hash(text, "python") => 7

Note: 0 means definetly not there, 1 means probably

Probe(bf, "python")
hash(pui, "python") = 0
hash(sec, "python") = 2
hash(text, "python") = 7

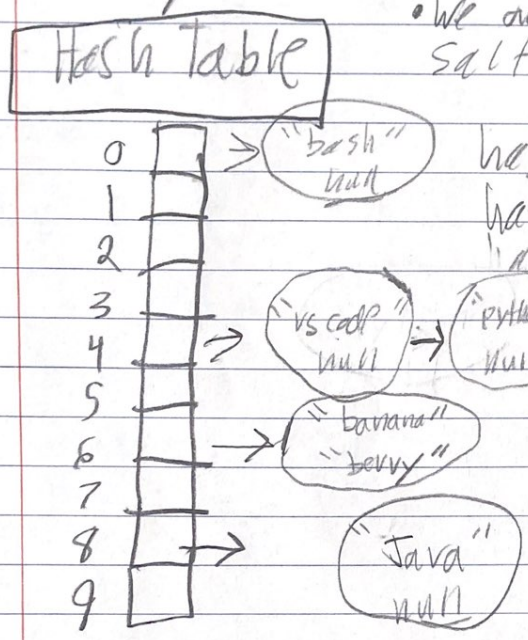
Checking all indices of "python" is index 0, 2, 7 set? We have most likely seen python before

Probe(bf, "dog")
hash() = 3
hash() = 7
hash() = 11

Since index 11 isn't set, we can definetly say "dog" is not in bloom filter.

Also must add to our hash table... but why?

- Array of linked lists...
- We only add once w/ a hash table's salt



hash(ht→salt, "Java") = 8
hash(ht→salt, "python") = 4
hash(ht→salt, "bash") = 0
hash(ht→salt, "vscode") = 4
hash(ht→salt, "banana") = 6

Is "computer" a bad word?

hash() = 2
hash() = 4
hash() = 9

It is in all 3 indices of B.F., so we check HaT.

h(ht→sat, "computer") = 4

Is "computer" the ddSpeak word of node "vscode"? node "python"? No, it is not. Computer is definitely not a bad word.

Is "banana" a bad word?

hash(), hash(), hash() = ...

h(ht→salt, "banana") = 6

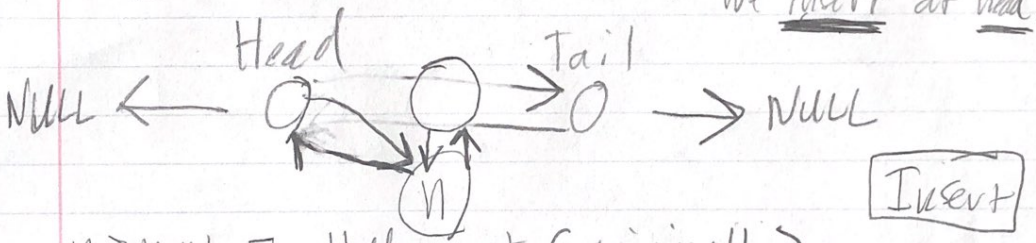
"banana" matches ddSpeak and was a newSpeak "berry"

Phase 1
- Add words from badspeak.txt to Bloom filter & Hash table
- Add words from Newspeak.txt to B.F. & H.T.

Phase 2
- Firewall part... Read user input, parse each word
- query the B.F.
- If in B.F., check H.T.
- Else not a bad word

Linked Lists

- Say, we add node "banana" "bruv"
- We insert at head.

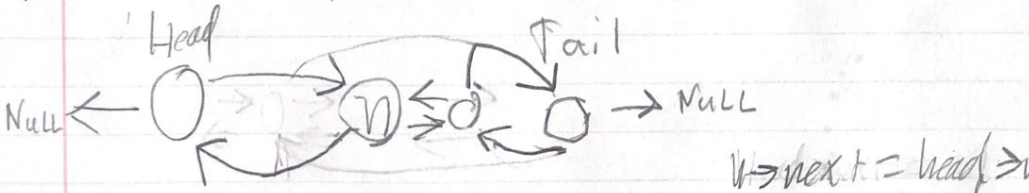


Insert

n→next = Head→next (originally)
n→prev = Head
Head→next = n
n→next→prev = n

Why move to front? "the" null is common and it should be in front for time efficiency

- If move to front is enabled



n→prev→next = n→next
n→next→prev = n→prev

n→next = head→next
n→prev = head
Head→next = n
n→next→prev = n

Regular Expressions

• Pattern matching

Basic expressions for regex

- `+` : one or more
- `*` : zero or more
- `?` : zero or one (optional)
- `|` : OR
- `(...)` : for precedence

`[ chars ]` : match chars in the set

`a+` : matches strings w/ one or more a's ...
`a*` : matches strings w/ zero or more a's ...

"aaaaaaa" matched by "a+"

"a" matched or accepted by "a+"
"" does not match/accepted by "a+"

`(0|1)+` ; binary (a string of 0's & 1's)

"0 110100111101" accepted by "(0|1)+"

`[a-z]` : matches any symbol b/w a-z inclusive
`[a-z]+` : matches one or more symbols b/w a-z.

"([a-z]+)-([a-z]+)"
↖ hyphen
matches hyphenated words

"([a-z]+)'([a-z]+)"
Contractions
don't
can't

Purpose of Regex is to parse out input

Input.txt

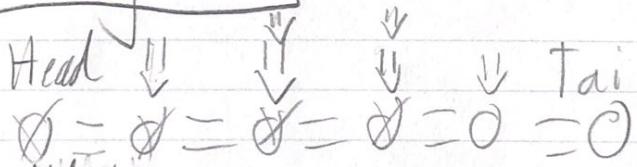
Dear class, ...
Today is my penultimate section.

- All different words... we need regular expression to parse these words out
- We use parsing module to do this
- "..." would not be a word

"[a-z]+-[a-z]+" will match...
(match) pseudo-code
(not match) pseudo

Whatever we read from input.txt needs to be converted to lowercase

Deleting L.L



Node Save Node;
for each node:
    save pointer to next
    delete_node
    go to saved pointer

Begin assignment w/...

1.) Node
2.) Linked List
3.) Hash Table (try adding to hash table)
4.) Bloom Filter
5.) ~~BS~~ Add parser
6.) Messages

Note! hash table ~~BF~~ ~~size~~ default size = 10000
B.F. default size is 2^{20}

Note: If hash table is smaller, we will have longer ~~list~~ L.L's bc more collisions

If B.F is smaller we will have more false-positives bc we have less bits to use which means ~~more~~ words will share bits more often.