Assignment 7

Design: Newspeak

Linhao Chen
CSE 13S - Fall 2019

Date:11/22/2019

1 Introduction

This program will read a list of forbidden words and oldspeak, newspeak pairs. Then set each of them into Bloom Filter and Hash Table, which is using linked list to prevent hash collision. After setting these words, the program will read a text from standard input and test each word in the text. If the text is a forbidden word, the program will show print a list of error words. If the text is an oldspeak and have a translation to newspeak, the program will show correct them. Expected to use Bloom Filter, Bit Vector, Hash Table, Linked List, Hash Function, Dynamic Memory Allocation, etc.

2 Newspeak

Main {
Set two variables for size of Hash Table and Bloom Filter
Set a variable to count how many words is added
Set the default value of variables above to 10000, 2^20
Set two bool variables to represent whether user choose statistic function and the rule
Set the default of bool variables to false
Read the command line
Print error and exit the program if arguments does not satisfy the requirement
Switch to case depend on the command-line option
In case ‘s’
User choose statistic function, set the bool variable to true

In case ‘h’
Set the size of Hash Table to the parameter
Check whether the user input is valid

In case ‘f’

Set the size of Bloom Filter to the parameter
Check whether the user input is valid

In case ‘m’
If the bool variable of rule is true
Print error: user cannot choose -m -b at same time
Else
Set the global variable move_to_front to true
Set the bool variable of rule to true

In case ‘b’
If the bool variable of rule is true
Print error: user cannot choose -m -b at same time
Else
Set the global variable move_to_front to false
Set the bool variable of rule to true

Create a Hash Table and Bloom Filter by using specific size
Open the badspeak.txt file and check
Use While loop and Flex to get all forbidden words (Until read EOF)
Insert them to Bloom Filter
Create a goodspeak structure by setting the words is old speak, the translation is NULL
Insert the goodspeak structure to Hash Table
Count for num of inserted word increment 1
Finish Reading badspeak.txt

Open the goodspeak.txt file and check
Set two string variables named oldspeak and newspeak
Set a bool variable to flip the scan
Set the default value to false
Use While loop and Flex to get all word pairs(Until read EOF)
If bool variable is false
Scan a word and store to oldspeak
Set the bool vairbale to true
Else
Scan a word and store to goodspeak
Insert oldspeak to Bloom Filter
Create a goodspeak structure by using oldspeak and newspeak
Insert the goodspeak structure to Hash Table
Count for num of inserted word increment 1
Finish Reading goodspeak.txt

```
    Read the Standard Input
    Set two strings for storing the badspeak and goodspeak pairs
    Set a counter to calculate how many words is tested
    Use while loop and Flex to get each word (Until read EOF)
        Counter of word examined increment 1
        Probe each word from Bloom Filter
        If the return value is true
            Lookup in Hash Table
            If the returned Node is not NULL
                Get the value of new_speak stored in this Node
                If the new_speak is NULL
                    Store the word in badspeak
                Else
                    Store the word pairs in goodspeak by using format (old -> new)

    If Bool variable of statistic is false
        Print the forbidden words and goodspeak pairs with instruction words
    Else
        Test how many bits is filled by 1 in Bloom Filter by calling function bf_density
        Print the data
        Test the average length of Linked List by calling function ht_avg_len
        Print the data
        Test the possible false positive rate by calling function bf_rate
        Print the data

    Delete the Hash Table and Bloom Filter
    Return 0
}


bf_density (Parameter Filter) {
    Set a counter
    Use for loop to Go through all the bit in bit vector
        If the bit is filled by 1
            Counter increment 1
    Return the counter
}

ht_avg_len (Parameter Hash Table and counter for num of words inserted) {
    Return  num of words inserted / size of Hash Table

bf_rate (Parameter Filter, counter for num of words inserted) {
```

Use the Formula below to calculate the rate (Credit: This formula derived from Wikipedia)

$$(1 - \left[1 - \frac{1}{m}\right]^{kn})^k$$

While k is the number of Hash function (k is 3 there), m is the size of Bloom Filter, and n is the num of words inserted

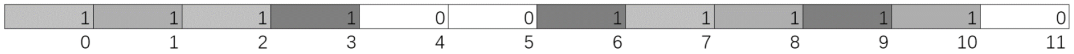
Return the value

}

3 Bloom Filter

The idea of Bloom Filter will use three salts and hash functions for letting each word's corresponding bits in Bloom Filter will set to 1. For example, we have a world "hello", and the corresponding bits are 3, 6, 9, then the program will set these bits to 1. When the program is trying to probe a word, it will only test whether all corresponding bits in Bloom Filter is 1. If the answer is yes, this word may exist, otherwise is no. The Bloom Filter will use Bit Vector to set bit. The diagram below is a simple example of Bloom Filter by using Bit Vector.

EX.	char* key	hello	world	chen	zero	santa
	hf1	3	10	0	1	3
	hf2	6	8	2	5	8
	hf3	9	1	7	6	1



Check	hello	world	zero	santa
	3	10	1	3
	6	8	5	8
	9	1	6	1
	probably	probably	NO	probably

Structure of Bloom Filter {

Three Salts value for hash function

A BitVector as Filter

}

bf_create (Parameter Size) {

Allocate memory space for Bloom Filter

Check whether allocate successfully

Set the value of three salts

Create a BitVector as filter, the size will be parameter

Return Bloom Filter

If allocation fail

Return NULL

}

```
bf_delete (Parameter Filter) {
    If the pointer does not point to NULL
       Free the memory space for filter (bit vector)
       Free the memory space for Bloom Filter
       Set the pointer to NULL
    Return
}


bf_insert (Parameter Filter and string) {
    Put three different salt and one string to three hash functions
    Get the hash value from each function and mod the length of filter
    Set three corresponding bits to 1 in the filter (Bit Vector)
}

bf_probe (Parameter Filter and string) {
    Put three different salt and one string to three hash functions
    Get the hash value from each function and mod the length of filter
    Check whether three corresponding bits are filled by 1
       If true
          Return true
        Else
          Return false
}
```

The Bit Vector is created early in Assignment 5 for Factoring, below is the copy of Bit Vector design for asgn 5.

3.1 Bit Vector

```
bv_create (Parameter bit_len) {
   Allocate a memory for BitVector v
   Length of byte vector equals to (bit_len/8) + 1
   Set length of BitVector v equals to bit_len
   Allocate a memory space for byte vector
   Return BitVector v
}

bv_create (Parameter BitVector *v) {
   Free memory for vector of byte
   Free memory for BitVector v
   Set the pointer to NULL
```

}

bv_get_len (Parameter BitVector *v) {
  return the length of BitVector v
}

bv_set_bit (Parameter BitVector *v, number i) {
  Block equals to i divide by 8
  Index equals to remainder of i divide by 8
  Set the bit to ByteVector [Block]'s index bit to 1
  (e.g. if i is 9, then Block is 1 and Index is 1, so the location will be vector [1]'s second bit)
  (if vector [1] in bit is 00100000, then we should use OR operation with 01000000 to set,
  And then it will be 01100000)
}

bv_clr_bit (Parameter BitVector *v, number i) {
  Block equals to i divide by 8
  Index equals to remainder of i divide by 8
  Set the bit to ByteVector [Block]'s index bit to 0
  (e.g. if i is 8, then Block is 1 and Index is 0, so the location will be vector [1]'s first bit)
  (if vector [1] in bit is 10100000, then we should use AND and NOT operation to clear
  e.g. 10100000 & ~(10000000) = 10100000 & 01111111 = 00100000)
}

bv_get_bit (Parameter BitVector *v, number i) {
  Block equals to i divide by 8
  Index equals to remainder of i divide by 8
  Get the value of specific bit by using short division
  (e.g. if i is 7, then Block is 0 and Index is 7, so the location will be vector [0]'s last bit)
  Pseudocode for short division (by using i equals 7):
    Temp equals to ByteVector [0], 00110101 in bit, and the value should be 53
    Do a for loop (o from 1 to index, use 1 as increment)
      Temp equals temp divide by 2
    Return the value of the remainder of temp divide by 2
}

  Diagram to explain the short division:

2	53	1	
2	26	1	
2	13	0	
2	6	1	
2	3	0	
2	1	1	$\leftarrow return\ 1$

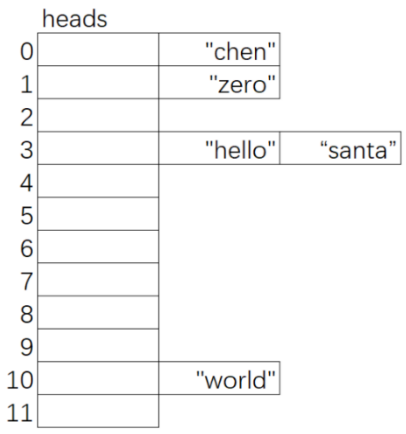
bv_set_all_bit (Parameter BitVector *v) {
  Do a for loop (i from 0 to length to bit vector, use 1 as increment)
    Call bv_set_bit to set bit by using parameter i
}

3.2 Hash Function

The Hash Function is provided in PDF design which created by Ray Beaulieu, Stefan TreatmanClark, Douglas Shors, Bryan Weeks, Jason Smith and Louis Wingers

4 Hash Table

Hash Table is a thing to store each forbidden words and newspeak pairs. We need to use a salt and hash function to locate the place to store words. Once a word is passed by Bloom Filter, we need to search such word in Hash Table. For prevent the Hash Collision (two word may have same hash value), the program will use hash function to prevent this.



4.1 Hash Table

Structure of Hash Table {
  One Salt value for hash function
  The number of entries in the Hash Table
  An array of the heads of Linked List
}

ht_create (Parameter length) {
  Allocate memory space for Hash Table
  Check whether allocate successfully
    Set the value of salt
    Set the length of hash table by using parameter
    Allocating memory space for heads by using parameter as size
    Return Hash Table
  If allocation fail
    Return NULL
}

ht_delete (Parameter Hash Table) {
  If the pointer does not point to NULL
    Free memory of each linked list by calling function in List

```
        Free memory of the array of the heads of linked list
        Set the pointer to NULL
    Return to main
}


ht_lookup (Parameter Hash Table and string) {
    Put salt and one string to hash function
    Get the hash value from function and mod the length of hash table
    Search the string in corresponding linked lists of hash table by calling function in list
    Return the result
}


ht_insert (Parameter Hash Table and goodspeak) {
    Put salt and one string to hash function
    Get the hash value from function and mod the length of hash table
    Insert the goodspeak into corresponding linked lists of hash table by calling function in list
    Get the pointer of ListNode which created by function above
    Set the newest ListNode as the head of corresponding linked lists
    Return to main
}
```

4.2 Linked List

```
Structure of ListNode {
    Goodspeak structure
    A pointer to next ListNode
}

ll_create (Parameter goodspeak structure) {
    Allocate memory space for ListNode
    Check whether allocate successfully
        Set the goodspeak
        Set the pointer to next ListNode as NULL
        Return the Node
    If allocation fail
        Return NULL
}

ll_node_delete (Parameter ListNode) {
    If the pointer does not point to NULL
        Save a pointer to next Node
        Free the memory for current Node
```

```
        Set the pointer of current Node to NULL
        Free the memory for next Node by recalling function
        Set the pointer of next Node to NULL
    Return to main
}

ll_delete (Parameter ListNode) {
    Free the memory of Linked List start with head by calling function ll_node_delete
    Return
}

ht_lookup (Parameter The pointer to head of list and string) {
    Set a pointer of ListNode n to the head
    Use a string variable to store some values
    While the pointer does not point to NULL
        Set the value of string variable to current old_speak value
        Compare whether the value of variable is equal to parameter string
            If true
                Break the loop
            Else
                Set the pointer to next Node
    Move to Front Rule (If chosen)
        Set the next pointer of Current Node n is the pointer to head
        Set the head of linked list to current Node n
    Return the pointer
}


ht_insert (Parameter The pointer to head of list and goodspeak) {
    Create a ListNode by using parameter goodspeak;
    Set the next pointer of new Node is the pointer to current head
    Return the new pointer of Node
}
```

4.2.1 Good Speak

```
Structure of GoodSpeak {
    One String for old_speak
    One String for new_speak
}
Speak_create (Parameter Two strings) {
    Allocate Memory space for Goodspeak
    Check whether allocate successfully
```

```
        Set the value of old speak
        Set the value of new speak
        Return the pointer to goodspeak
    If allocation fail
        Return NULL
}

Speak_delete(Parameter Goodspeak Structure) {
    If the pointer does not point to NULL
        Free the memory of goodspeak
        Set the pointer to NULL
    Return
}
```