DESIGN - Assignment 7

Brian Nguyen

November 26, 2021

1 Description

This assignment contains a program that implements a message filtering system with the use of a bloom filter, hash table, node, bit vector, and binary search tree ADTS. The bloom filter gets the hash of a word using salts provided by the assignment and indexes them with the use of bit vectors. The hash table incorporates binary search tree (made of nodes) which contains all of the badspeak or newspeak words that will be filtered or punished in the message also using hashing similar to the bloom filter. This is all then used in the main ban hammer program which either punishes or rewards the user for expressing a bad or good message respectively.

2 Files

1. banhammer.c
   - This source file contains the code for the message filtering program and contains a main().
2. speck.c
   - This source file contains the code for the hash functions using the SPECK cipher.
3. ht.c
   - This source file contains the code for the hash table ADT.
4. bst.c
   - This source file contains the code for the binary search tree ADT.
5. node.c
   - This source file contains the code for the node ADT.
6. bf.c
   - This source file contains the code for the bloom filter ADT.

7. bv.c
    - This source file contains the code for the bit vector ADT.
8. parser.c
    - This source file contains the code for the regex parsing module.
9. messages.h
    - This header file defines the mixspeak, badspeak, and goodspeak messages that are used in banhammer.
10. salts.h
    - This header file defines the primary, secondary, and tertiary salts to be used in the bloom filter and hash table.
11. speck.h
    - This header file contains the interface for the hash functions using the SPECK cipher.
12. ht.h
    - This header file contains the interface for the hash table ADT.
13. bst.h
    - This header file contains the interface for the binary search tree ADT.
14. node.h
    - This header file contains the interface for the node ADT.
15. bf.h
    - This header file contains the interface for the bloom filter ADT.
16. bv.h
    - This header file contains the interface for the bit vector ADT.
17. parser.h
    - This header file contains the interface for the regex parsing module.
18. Makefile
    - This make file contains the code that builds and compiles the program to be run. It also cleans all compiler generated files and formats the code to be submitted.

19. README.md

    - This markdown file describes the program, how to build it, how to run it, and also lists and explains all the command-line options that the program accepts. It also documents any false positives given by scan-build.

20. DESIGN.pdf

    - This pdf is the manual that explains the program, files included, layout or structure, and pseudo-code of the program.

21. WRITEUP.pdf

    - This pdf is the graphical analysis of the different variation/factors that affect the accuracy and speed of the message filtering program.

3 Structure

4 Pseudocode

4.1 Bloom Filter

```
define bf_create
    set filter to bv_create(size)
    set bf salts to hash function salts

define bf_delete
    bv_delete filter
    set point of filter to Null

define bf_size
    return bv_length of filter

define bf_insert
    hash oldspeak with 3 salts
    bv_set_bit at hashed indexes of oldspeak

define bf_probe
    hash oldspeak with 3 salts
    bv_get_bit at hashed indexes of oldspeak
```

```
define bf_count
    set counter
    for loop i 0 to length
        if bv_get_bit is equal to 1
            increment counter
    return counter

define bf_print
    use bv_print
```

4.2 Bit Vector

```
define bv_create
    Dynamically allocate array of bitvector with size of bitvector struct #use malloc
    if vector exists
        set bytes to length/8 + (if modulo of length and 8 greater > 1 use 1 else use 0)
        set vector of vector to Dynamically allocated array of uint8_t sized elements
        set vector length to length
        return vector
    else
        return pointer of vector 0

define bv_delete
    if vector and vector of vector exist
        free vector of vector
    if vector exists
        free vector
    return

define bv_length
    return length of vector #from struct

define bv_set_bit
    if bv exists and vector of bv exists
        set vector of bv at index (i/8) bit to 1 at (k mod 8) position
        return true
    return false

define bv_clr_bit
```

```
    if bv exists and vector of bv exists
        set vector of bv at index (i/8) bit to 0 at (k mod 8) position

define bv_get_bit
    if bv exists and vector of bv exists
        return vector of bv at index (i/8) bit with bitwise-and 1 at (k mod 8) position

define bv_print
    for loop 0 to length
        for loop i from 0 to 7
            print vector of bv at index (i/8)
```

4.3 Hash Table

```
define ht_create
    set ht salts to salts in header
    set ht size to input size

define ht_delete
    for loop 0 to size
        if root exists
            bst_delete root at ht[i]
    set pointer of ht to NULL

define ht_size
    return size of ht #directly from struct

define ht_lookup
    hash oldspeak using salts
    return bst_find node at hashed index, oldspeak

define ht_insert
    hash oldspeak with salts
    if ht_count is less than size
        bst_insert at hashed index, oldspeak, newspeak

define ht_count
    for loop 0 to size
        if bst_size of index greater than 0
        increment counter by 1
    return counter

define ht_avg_bst_size
    initialize sum var
    for loop 0 to size
        if bst_size of index greater than 0
        incrememnt sum by size
    return sum/ht_count

define ht_avg_bst_height
    initialize sum var
    for loop 0 to size
        if bst_height of index greater than 0
        increment sum by height
    return sum/ht_count

void ht_print
    print array of bst trees
```

4.4 Node

```
define node_create
    Dynamically allocate Node array (n) with sizeof(Node) #use malloc()
    set oldspeak of n to copy of oldspeak #use strdup()
    set newspeak of n to copy of newspeak #use strdup()
    set left of n to NULL
    set right of n to NULL

define node_delete
    free pointer of node n
    free oldspeak
    free newspeak
    set pointer n to NULL

define node_print
    print oldspeak to newspeak #see asgn7 for format
    print oldspeak only if no newspeak #see asgn7 for format
    #for debug
    print node left
    print node right
```

4.5 Binary Search Tree

```
define bst_create
    use node_create()

define bst_delete
    if root exists
        bst_delete left
        bst_delete right
        node_delete root

define max #helper function for bst_height
    return x if x > y
    return y if y > x
define bst_height
    if root exists
        return max of (bst_height of left root vs bst_height of right root) + 1 #recursive
    return 0 #if root is NULL

define bst_size
    if root exists
        return (bst_size of left root + 1 + bst_size of right root) #recursive
    return 0 #if root is NULL

define bst_find #replace key with oldspeak
    if root exists
        if key of root is less than key #use strcmp()
            return bst_find of left root, key
        else if key of root is greater than key #use strcmp()
            return bst_find of right root, key
        return root #if equal
    return root #returns a NULL pointer

define bst_insert #replace key with oldspeak
    if root exists
        if key of root is equal to key #no duplicates allowed
```

```
            return
        else if key of root is greater than key
            set left root to bst_insert left root, key
        else if key of root is less than key
            set right root to bst_insert right root, key
        return root
    return node_create(key)

define bst_print
    if root exists
        bst_print left root
        node_print root
        bst-print right root
```

4.6 Ban Hammer

```
define main
    initialize bloom filter and hash table #use bf_create and ht_create
    use fscanf() to read in badspeak & oldspeak -> newspeak words
    insert them into bloom filter and hash table
    use regex module to get input #stdin
    scan each word with bloom filter
    if word in bloom filter #bf_probe()
        if word in hash table
            if newspeak is NULL
                return thoughtcrime msg with badspeak words used
            else if newspeak is not NULL
                return rightspeak msg with newspeak translation words #if not thoughtcrime
        else
            return good msg
    return good msg
```

5 Error Handling

1. The Bloom Filter is probabilistic, meaning that False Positives can appear which is where the Hash table verifies if it is truly a bad word.

2. None so far.

6 Credits

1. I used the asgn7.pdf from Professor Long for explanations and pseudocode.
2. I watched Eugene's lab section recordings held on 11/23.