# DESIGN - Assignment 7

Brian Nguyen

December 3, 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.

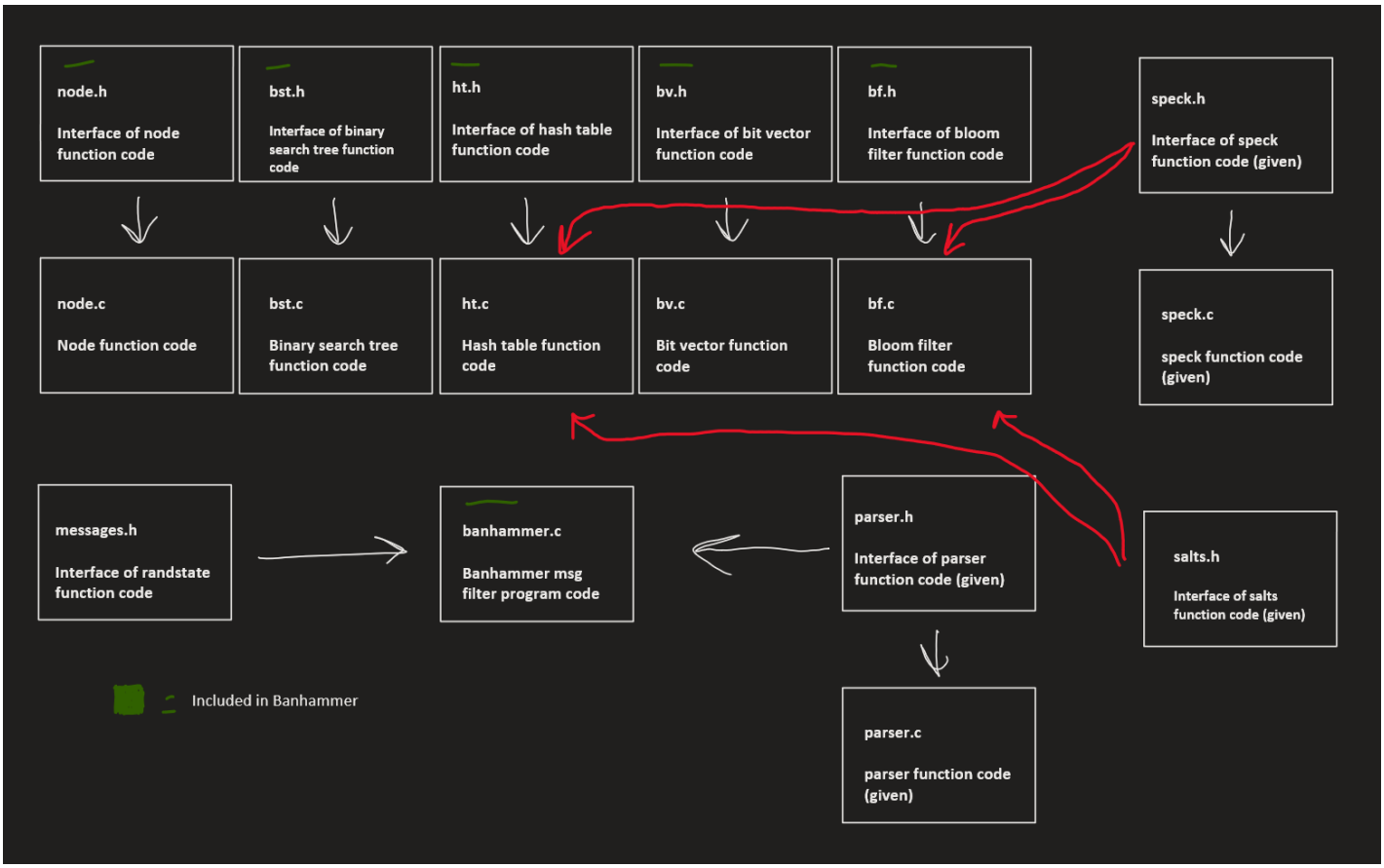
20. DESIGN.pdf

    - 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

```
#Dynamically allocates a bit vector that makes up the Bloom Filter + adds salts for hashing
define bf_create #input: size
    set filter to bv_create(size)
    set bf salts to hash function salts

#Frees the bit vector from the bloom filter then the bloom filter itself
define bf_delete #input: bf
    if bf
        if filter of bf
            bv_delete filter
        free bf
        set bf to Null

#Returns the size of the bit vector of the bloom filter
define bf_size #input: bf
    return bv_length of filter

#Sets the bits of the hashed index to 1 based on the oldspeak word
define bf_insert #input: bf, oldspeak
    hash oldspeak with 3 salts
    bv_set_bit at hashed indexes of oldspeak

#Checks if the oldspeak word was hashed into the bloom filter (all 3 indexes)
define bf_probe #input: bf, oldspeak
    hash oldspeak with 3 salts
    bv_get_bit at hashed indexes of oldspeak

#Returns the number of set bits in the bloom filter
define bf_count #input: bf
    set counter
    for loop i 0 to length
        if bv_get_bit is equal to 1
            increment counter
    return counter
```

```
#Debug function to print the vector of bf
define bf_print #input: bf
    use bv_print
```

### 4.2 Bit Vector

```
#Dynamically allocates an array of bytes to hold at least length bits and sets length
define bv_create #input: length
    Dynamically allocate array of bv with size of bv 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 bv to Dynamically allocated array of uint8_t sized elements
        set bv length to length
    return bv

#Frees memory of all bytes in vector then the vector itself
define bv_delete #input: bv
    if bv
        if vector of bv
            free vector of bv
        free bv
        set bv to NULL

#Returns the length in bits of the bit vector
define bv_length #input: bv
    return length of vector #from struct

#Sets the ith bit to 1 if in range
define bv_set_bit #input: bv, i
    if bv exists and vector of bv exists and in range
        set vector of bv at index (i/8) bit to 1 at (i mod 8) position
        return true
    return false

#Sets the ith bit to 0 (clears it) if in range
define bv_clr_bit #input: bv, i
    if bv exists and vector of bv exists and in range
        set vector of bv at index (i/8) bit to 0 at (i mod 8) position
        return true
```

```
    return false

#Gets the ith bit of the vector if in range
define bv_get_bit #input: bv, i
    if bv exists and vector of bv exists and in range
        get bit of bv(vector) at index (i/8) bit with bitwise-and 1 at (i mod 8) position
        return true if 1
    return false

#Debug function to print the bytes of the vector
define bv_print #input: bv
    for loop 0 to length
        for loop i from 0 to 7
            print vector of bv at index (i/8)
```

### 4.3 Hash Table

```
#Dynamically allocates an array of bst trees based the size + sets its size and salts
#for hashing
define ht_create #input: size
    Dynamically allocate hash table memory #calloc()
    set ht salts to salts in header
    set ht size to input size
    set ht trees to array of bsts based on size (made of nodes)

#Frees the memory from all the bst trees and then the hash table
define ht_delete #input: ht
    if root exists
        for loop 0 to size (if trees exists)
            bst_delete root at ht trees[i]
        free pointer of ht trees
    free pointer of ht
    set pointer of ht to NULL

#Returns the size of the hash table
define ht_size #input: ht
    return size of ht #directly from struct

#Finds whether the oldspeak word is in the hash table
```

```
define ht_lookup #input: ht, oldspeak
    hash oldspeak using salts
    return bst_find node at hashed index, oldspeak

#Inserts the node into the hash table if not a duplicate
define ht_insert #input: ht, oldspeak, newspeak
    hash oldspeak with salts
    bst_insert at hashed index, oldspeak, newspeak

#Finds the amount of non null bsts in the hash table
define ht_count #input: ht
    initialize counter var
    for loop 0 to size
        if bst_size of index greater than 0
        increment counter by 1
    return counter

#Gets the total size of all bst trees and divides by the amount of non null bsts to get
#avg bst size
define ht_avg_bst_size #input: ht
    initialize sum var
    for loop 0 to size
        if bst_size of index greater than 0
        incrememnt sum by size
    return sum/ht_count

#Gets the total height of all bst trees and divides by the amount of non null bsts to get
#avg bst height
define ht_avg_bst_height #input: ht
    initialize sum var
    for loop 0 to size
        if bst_height of index greater than 0
        increment sum by height
    return sum/ht_count

#Prints all the bsts in the hash table in order
void ht_print #input: ht
    print array of bst trees #for loop
```

### 4.4 Node

```
#Creates node array consisting of oldspeak and newspeak and NULL pointers to their children
define node_create #input: oldspeak, newspeak
    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

#Frees oldspeak and newspeak strdup() memory then frees the node and NULLs the pointer
define node_delete #input: n
    free pointer of node n
    free oldspeak
    free newspeak
    set pointer n to NULL

#Prints out oldspeak its newspeak translation of the node if not NULL
define node_print #input: n
    print oldspeak to newspeak #see asgn7 for format
    print oldspeak only if no newspeak #see asgn7 for format
```

### 4.5 Binary Search Tree

```
#Creates a NULL bst tree (just for semantics)
define bst_create
    return NULL

#Recursively frees all the memory from the nodes of the tree
define bst_delete #input: root
    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

#Gets the height of the tree by adding 1 per traversal and comparing the max height
```

```
#(left vs right) via recursion
define bst_height #input: root
    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

#Gets the size of the tree by adding 1 per traversal via recursion
define bst_size #input: root
    if root exists
        return (bst_size of left root + 1 + bst_size of right root) #recursive
    return 0 #if root is NULL

#Finds whether the oldspeak word is in the bst via recursion
define bst_find #input: root, oldspeak
    if root exists
        if oldspeak of root is less than oldspeak #use strcmp()
            return bst_find of left root, oldspeak
        else if oldspeak of root is greater than oldspeak #use strcmp()
            return bst_find of right root, oldspeak
        return root #if equal
    return NULL #if not in tree

#Inserts the oldspeak & newspeak pair into an empty tree node if it is not already in the
#tree via recursion
define bst_insert #input: root, oldspeak, newspeak
    if root exists
        if oldspeak of root is greater than oldspeak
            set left root to bst_insert left root, oldspeak
        else if oldspeak of root is less than oldspeak
            set right root to bst_insert right root, oldspeak
        return root #if duplicate
    return node_create of oldspeak, newspeak

#Prints all the nodes of the tree by in order traversal
define bst_print #input: root
    if root exists
        bst_print left root
        node_print root
```

```
        bst-print right root
```

### 4.6 Ban Hammer

```
define main
    use getopt to get user input to start off
    #switch cases: h, t (table size), f (filter size), s
    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
    Initialize bst to hold bad words
    Initialize bst to hold mixed words
    use regex module to get input #stdin
    while scan each input word with parser #while loop
        lowercase input word
        if word in bloom filter #bf_probe()
            Set checker node to ht_lookup of input word
                if checker not NULL and newspeak of checker not NULL
                    bst insert to mixed words
                if checker not NULL and newspeak of checker is NULL
                    bst insert to bad words
    if stats
        print stats
    else
        if bst size of badwords > 0 and bst size of mixedwords > 0
            print mixspeak msg
            print bad and mixed words
        if bst size of badwords > 0 and bst size of mixedwords == 0
            print badspeak msg
            print bad words
        if bst size of badwords == 0 and bst size of mixedwords > 0
            print goodspeak msg
            print mixed words
    free and clear memory
```

## 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. Some edge cases and negative table sizes do not properly handle errors.

## 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.
3. I looked at Professor Long's Lecture 18 slides for bst explanations/code.
4. I looked at Eric Hernandez's, the tutor, notes.
5. I used bv8.c from the Code Comments GitLab repository.