**Alex Asch**
**ASGN7 Design**
The great FIrewall of UCSC
**Deliverables**

- banhammer.c
  - This file contains the implementation of main,
- messages.h
  - Defines the mixspeak badspeak and goodspeak messages used.
  - This document provided in the resources repository by Professor Long.
- salts.h
  - Defines the primary secondary and tertiary salts to be used in the Bloom filter.
  - Also defines the salt of the hash table.
  - This document provided in the resources repository by Professor Long.
- speck.h
  - Defines the interface for the has function using the SPECK cipher.
  - This document provided in the resources repository by Professor Long.
- speck.c
  - Contains the implementations of the has function using the SPECK cipher.
  - This document provided in the resources repository by Professor Long.
- ht.h
  - Defines the interface for the hash table ADT.
  - This document provided in the resources repository by Professor Long.
- ht.c
  - Contains the implementation of the has table ADT.
- bst.h
  - Defines the interface for the binary search tree ADT.
  - This document provided in the resources repository by Professor Long.
- bst.c
  - Contains the implementation of the binary search tree ADT.
- node.h
  - Defines the interface for the node ADT.
  - This document provided in the resources repository by Professor Long.
- node.c
  - Contains the implementation of the node ADT.
- bf.c
  - Contains the implementation of the bloom filter ADT.
- bv.h
  - Defines the interface for the bit vector ADT.
  - This document provided in the resources repository by Professor Long.
- bv.c
  - Contains the implementations of the bit vector ADT.
- parser.h
  - Defines the interface for the regex parsing module.
  - This document provided in the resources repository by Professor Long.

- parser.c
    - Contains the implementation of the regex parsing module
- Makefile
    - The file that compiles the program using make.
- README.md
    - A document in markdown syntax describing program usage
- Design.pdf
    - The document being viewed currently, describes the design of the implementation of the program
- WRITEUP.pdf
    - Analysis and graphs of the program as Bloom filter and hash table size is changed.

**General Notes**

- Bloom filter, can report false positive but not false negative
- Make a bloom filter off of a bit vector
- Bit vector
    - Like a stack
    - instead of a static length you have a dynamically allocated length
- BLoom filter has a primary secondary and tertiary salts. The ones to be used are in salts.h
- Hashing with spec cipher
    - Fast and deterministic
    - Add a salt to the has function
- Bloom filter
    - An overlay for the hash table
    - Probabilistic
- Nodes
    - Binary trees will be used to resolve hash collisions
- Make a DB of bad words
- **Bloom filter**
    - create
        - the salts are given in a header file
        - Bits are bit vector count
    - delete
        - delete bitvec
        - free self
    - bf_size
        - Return the length of the bit vector
    - bf_insert
        - Take the character, hash it, set the bits at those indices in the bit vector
    - bf_probe
        - Probes the Bloom filter for oldspeak, hashes with the salts, and if all of the bits at these indices are set, return true to signify that oldspeak was mostlikely added to the Bloom filter

    - bf_count
        - returns the number of set bits in the bloom filter,
        - do that with bitvec
    - bf_print
        - debug use the bitvec one
- **Bit Vector**
    - bv_create
        - initializes an array of uint8_t that is length/8 for bits
        - use calloc to zero
    - bv_delete
        - free array
    - bv_length
        - returns length of a bit vector
    - bv_set bit
        - bit arithmetic to set ith bit false if oor
    - bv_clr_bit
        - sets ith bit to 0 using bit arithm
    - get bit
        - false if it is 0 true if it is 1
    - bv_print
        - debug funct, print the array
- **Nodes**
    - node_create
        - Use strdup to make copies of the given strings
        - If newspeak is null it’s badspeak, just let it be null
    - node_delete
        - Only the node is freed, free the strings too
        - set ptr to null
    - node_print
        - If oldspeak and newspeak, print both
        - If only oldspeak print one
- **BST**
    - bst_create
        - Start it as NULL, empty tree.
    - bst_delete
        - Postorder Traversal and delete each node
    - bst_height
        - returns the height of the bst rooted at root,
        - if root is null return
        - If it is not, take the larger of the L/R heights recursively
    - bst_find
        - Searches for a node containing oldspeak in the tree rooted at root, if found ptr to node is returned
    - bst_insert

        - Lexicographically insert something into the bst
- Hast table
    - ht_create
        - Create an array of length size
    - ht_delete
        - free each tree by using bst_delete
    - ht_size
        - the size of all the trees
    - ht_lookup
        - has the oldspeak, look in the bst at the point of oldpeak
    - ht_insert
        - inserts the oldspeak and the newspeak into hash table
        - hash it, this is the index to put it in
        - Put it into the tree there
    - ht_count
        - how many non_null trees are in the table?
    - ht_avg_bst_size
        - use bst_size for all trees over ht_count
    - ht_avg_bst_height
        - as above but with height
    - ht_pinrt
        - print each tree
        -