Konan Le

Professor Long

CSE 13S

28 November 2021

Assignment 7 - The Great Firewall of Santa Cruz

This program takes in a list of “badspeak” words, words to filter out, and takes them into a bloom filter. A bloom filter is a data structure, represented by a bit vector, that holds various values set by hash functions. If hashing the value gives the same result in the bit vector, then that means the read word may be in the “badspeak” list. Oldspeak and newspeak, (unapproved language and their translations) along with the badspeak words are stored in a hash table, which is essentially a dictionary; the hashed values being the key and the unaltered data being the values. Proper action is taken depending on whether or not the read words using the parsing module appear in the bloom filter, and whether they appear to have a newspeak translation or not.

<u>PROGRAM FILES</u>

- banhammer.c
- messages.h
- salts.h
- speck.h
- speck.c
- ht.h
- ht.c
- bst.h
- bst.c
- node.h
- node.c

- bf.h
- bf.c
- bv.h
- bv.c
- parser.h
- parser.c
- Makefile
- DESIGN.pdf
- README.md
- WRITEUP.pdf

PSEUDOCODE

```
bf.c
  ● Define a bloom filter struct
      ○ Initialize primary, secondary, tertiary hash functions
      ○ Initialize Bitvector pointer called filter
  ● bloom filter create function, takes uint32_t size as a
    parameter, returns a bloom filter
      ○ allocate memory for a pointer to a bloom filter
      ○ Check if bloom filter has space
      ○ set corresponding primary, secondary, tertiary filters to
        their values in salt.h
      ○ malloc space for bit vector ADT
  ● bloom filter delete function, takes bloom filter double pointer
    as parameter, returns void
```

  - If pointer to the bloom filter exists, free the pointer, then set it to NULL
  - free the bit vector that makes up the bloom filter
- Bloom filter size function, takes pointer of bloom filter as parameter, returns uint32
  - Return the length of the bit vector in the bloom filter
- bloom filter insert function, takes oldspeak string and bloom filter as parameter
  - Hash the oldspeak word with the 3 hash functions
  - Set the indices in the bit vector in the bloom filter to their corresponding values given by the hash function
- bloom filter probe function takes bloom filter pointer, oldspeak string as parameter, returns boolean
  - Hash the oldspeak word with the 3 hash functions
  - If the bits at each corresponding index is there, return true for probably there
- bloom filter count function takes bloom filter pointer as parameter, returns uint32
  - iterates through bit vector, finds number of 1s in the vector
- bloom filter print takes bloom filter pointer as parameter, returns void
  - use bit vector print function to print out bits of the bit vector

bv.c (Bit vectors, from here on abbreviated to bv)

- bv struct definition
  - declare uint32 length, uint8 vector pointer
- bv create function, takes uint32 length as parameter
  - allocate memory for the vector pointer,
  - if the vector pointer exists (if enough memory was available to allocate), initialize the length to length taken into parameter and return the bit vector.
  - else, not enough memory, so return NULL.
- bv delete function, takes double pointer of bv
  - if the pointer to bv exists, free the pointer to vector and set it to NULL
- bv length function, takes bv pointer as parameter
  - returns length value of bv
- bv set bit function takes bv pointer and uint32 i as parameter
  - if i is in range,
    - sets the ith bit of the vector array to 1
    - return true
  - else return false
- bv clear bit function takes bv pointer and uint32 i as parameter
  - if i is in range,
    - sets the ith bit of the vector array to 0
    - return true
  - else, return false

- bv get bit function takes bv pointer, uint32 i as input
  - if i is in range or the ith value is 1
    - return true
  - if i is out of range or the ith value is 0
    - return false
- bv print function takes bv pointer as input
  - For loop iterating over bit vector
  - print the corresponding value

ht.c (Hash table abbreviated to ht)

- define struct
  - declare salt array as uint64
  - declare size as uint32
  - create a double pointer of object type Node, titled trees
- ht create function, takes uint32 size as parameter
  - sets salts to salt provided in salts.h
  - sets size to size taken in parameter
  - returns the created ht
- ht delete function takes double pointer of ht as parameter returns null
  - if ht pointer exists,
    - free the pointer
    - set the pointer to NULL
  - free all the nodes that make up the hash table
- ht size function takes ht pointer as parameter, returns uint32

  - returns size value of ht
- ht lookup function takes ht pointer, oldspeak string
  - performs search through a binary tree, with indices calculated by hashes
  - if a node is found that has a newspeak translation, returns the pointer to that node
  - if no node found, return NULL pointer
- ht insert function takes ht pointer, oldspeak and newspeak strings as parameters
  - hash the oldspeak to find indices needed to insert into tree
  - uses node functions to add oldspeak and newspeak pair to the tree of the hashtable
- ht count takes ht pointer as parameter, returns uint32
  - traverse through trees, keeping count of number of nodes travelled to. if NULL binary search tree, do not add to the count.
- ht average binary search tree size takes an ht pointer as parameter, returns a double
  - use bst_size() to calculate the average binary search tree size and return it
- ht average bst height takes an ht pointer as parameter, returns a double
  - use bst_height() to calculate average binary search tree height, return it

- ht print function takes ht pointer as parameter
    - traverse through the tree, print the node

node.c

- Node struct declares oldspeak char, newspeak char, a left and a right subnode
- node create takes oldspeak string, newspeak string as parameters
    - if node exists, set oldspeak and newspeak pointers to strdup of parameters
    - set left and right node to NULL pointers
- Node print function takes node pointer as parameter
    - print both oldspeak and newspeak if available
    - if only oldspeak, print only that

bst.c (binary search tree abbreviated to bst)

#do I define my own struct here? that's what makes the most sense to me, not exactly spelled out in the asgn7 doc though. unless it is…

- bst create function takes no parameters, returns a node
    - Uses node create function to create a NULL tree
- bst delete function takes a node double pointers a parameter
    - If leaf of tree, delete current node
    - then, recursively call bst delete function on each subtree of the node to destroy it bottom up
- bst height node returns uint32, takes root of node pointer s parameter

  - Traverse through tree until you hit a node, find a straight path down
  - Number of edges down is the height of the tree
- bst size function returns uint32, takes root of node tree as parameter
  - perform traversal through tree, counting each node
  - return the number of nodes in the tree
- bst find function takes root node pointer, oldspeak string as parameters, returns Node object
  - Search through the nodes to see if they contain the oldspeak. If so, return a pointer to the node. Else, return a NULL pointer.
- bst insert function takes root pointer, oldspeak and newspeak strings as parameters, returns Node object
  - Use bst find (or some other function created earlier) to see if it is a duplicate or not
  - Check lexicographical location of the oldspeak, and insert the function there
  - rearrange the tree accordingly, changing the node to be a subtree of that subtree
- bst print function takes node root pointer as input
  - go to the leftmost root and use node print
  - go to the parent root and print
  - go to the right child node and print

        - recursively call until parent node has no parent any longer

banhammer.c

#Should just be various function calls

- use getopt to parse user input
- set default values = 2^16 for the hash filter, 2^20 for the bloom filter
- switch statement to decide what to do with user inputs
- Initialize bloom filter and hash table
- Use fscanf to read in badspeak words
- Add badspeak words to bloom filter
- add badspeak words to hash table
- add oldspeak words to bloom filter
- add oldspeak and newspeak pairs to hash table
- use parsing module to read words from stdin
- with each word read:
    - if bf probe function returns false, then not in bloom filter, do nothing
    - if bf probe returns true:
        - #could be a switch statement as well?
        - if word is also in hash table and does not have a newspeak translation, then it is badspeak
            - trigger thoughtcrime flag
            - hold word in a buffer to show citizen later

```
            ■ if word is also in hash table and has a translation,
              then it is minor transgression
                ● trigger counseling flag
                ● hold word in buffer to show citizen later
            ■ if word is not in hash table
                ● do nothing, false positive
  ● if both triggers are tripped for thoughtcrime and counseling,
    print a mixspeak message
      ○ print badspeak words
      ○ print oldspeak words followed by translation
  ● if only thoughtcrime is tripped
      ○ print thoughtcrime message
      ○ print badspeak words
  ● if only counseling is tripped
      ○ print goodspeak message
      ○ print oldspeak followed by newspeak translations
parser.c (included .c file)
  ● used to parse through input using regex
speck.c (included .c file)
  ● used to hash data
```

DESIGN CHANGES/BRAINSTORM

- Added usage of class included .c files (speck.c, parser.c)