DESIGN.pdf ASGN7

Nathan Ong

November 28, 2021

1 Description

This collection of files contains a program that parses through an input file and scans for any words that do not fit a set criteria, and prints out a message that informs them of their wrongdoing and the consequences. This main program utilizes several ADT and modules, including a regex parsing module and binary search trees, hash tables, nodes, a Bloom filter, and bit vector ADTs.

The main ban hammer program has the following command line options:

- -h Prints out help message the exits the program.
- -t *hash table size* Specifies the number of hash table entries. Default is 2^{16} entries.
- -f *Bloom filter size* Specifies the number of Bloom filter entries. Default is 2^{20} entries.
- -s Enables the printing of statistics to stdout, and suppresses any message the program may print otherwise.

2 Files Included in the Directory

1. banhammer.c
   This file contains the implementation of the ban hammer program.

2. messages.h
   This file contains the definitions of the "mixspeak", "badspeak", and "goodspeak" messages used in the ban hammer program.

3. salts.h
   This file contains the definitions of the salts used in the Bloom filter, along with the salt used in the hash table.

4. speck.h
   This file contains the specification of the interface for the hash function using the SPECK cipher.

5. speck.c
   This file contains the implementation of the hash function using the SPECK cipher.

6. ht.h
   This file contains the specification of the interface for the hash table ADT.

7. ht.c
   This file contains the implementation of the hash table ADT.

8. bst.h
   This file contains the specification of the interface for the binary search tree ADT.

9. bst.c
   This file contains the implementation of the binary search tree ADT.

10. node.h
    This file contains the specification of the interface for the node ADT.

11. node.c
    This file contains the implementation of the node ADT.

12. bf.h
    This file contains the specification of the interface for the Bloom filter ADT.

13. bf.c
    This file contains the implementation of the Bloom filter ADT.

14. bv.h
    This file contains the specification of the interface for the bit vector ADT.

15. bv.c
    This file contains the implementation of the bit vector ADT.

16. parser.h
    This file contains the specification of the interface for the regex parsing module.

17. parser.c
    This file contains the implementation of the regex parsing module.

3 Pseudocode and Structure

3.1 banhammer.c

```
while opt isnt -1
    indice through arguments
    check for required arguments if necessary
initialize bloom filter and hash table
scan badspeak words
scan oldspeak and newspeak pairs, add pairs to hash table
add oldspeak to bloom filter
read words from stdin
    if word is in bloom filter take corresponding action
        if hash table has the word and no newspeak translation, notify of thoughtcrime
        if hash table has word and has a newspeak translation, notify of counseling on rightspeak
        if hash table doesnt have the word, no action taken
        if both thoughtcrime and counseling detected, send to joycamp
if statistics are enabled, print statistics and disable other messages
```

3.2 ht.c

```
ht create:
allocate memory for hash table
set salts
set size to size argument
allocate memory for trees array

ht delete:
delete trees in trees array
free tree array pointer and hash table pointer
null pointers

ht size:
return size of hash table

ht lookup:
hash oldspeak to get binary search tree index
use bst find to find node with oldspeak
if node is found, return node pointer
otherwise return null

ht insert:
hash oldspeak to get binary search tree index
insert oldspeak newspeak pair into binary search tree

ht count:
indice through hash table array
    if element is not NULL, increment count by 1
return count value

ht avg bst size:

ht avg bst height:

ht print:
indice through hash table array
    if element is not NULL, print binary search tree that is stored
```

3.3 bst.c

```
bst create:
bst height:
bst size:
bst find:
bst insert:
bst print:
bst delete:
```

3.4 node.c

```
node create:
allocate memory for Node
set oldspeak and newspeak pointers to given arguments
null left and right pointers
```

```
return node

node delete:
free node pointer
null node pointer

node print:
print node oldspeak and newspeak
    if newspeak is null, print oldspeak only
print left and right nodes
```

3.5 bf.c

```
bf create:
set low and high salts
create bit vector for filter

bf delete:
delete bit vector for filter pointer
free bf pointer and nullify

bf size:
return bit vector length of filter

bf insert:
hash oldspeak with salts
set bits at hash value in bit vector

bf probe:
hash oldspeak with salts
if bits are set, return true
otherwise return false

bf count:
indice through filter and get bits
    if bit is set increment count by 1
return count value

bf print:
print filter using bv print
```

3.6 bv.c

```
bv create:
allocate memory for bit vector
    if memory cannot be allocated, return NULL
set length to length argument
allocate memory for vector pointer

bv delete:
free vector pointer and bit vector
null pointers
free second bit vector pointer
null pointer
```

```
bv length:
return length value

bv set bit:
if argument out of range return false
set bit in vector index to 1
return true

bv clr bit:
if argument out of range return false
set bit in vector index to 0
return true

bv get bit:
if argument out of range return false
if bit at index in code array is 0 return false
if bit at index in code array is 1 return true

bv print:
indice through bits of bit vector
    check if bit is set
    if set, print 1, otherwise print 0
```