banhammer:
need Node ADT: very similar to asgn5
need a hash function: going to use the speck cipher
need a bit vector: similar to code for asgn5
going to use the bit vector in order to make a bloom filter!
going to build a binary search tree (or multiple!) using the Node ADT.
going to use Eugene's parser to go through text for lexical analysis with Regex

bit vector:

need to see how many bytes you need to hold length bits!

$\left\lceil \frac{\text{length}}{8} \right\rceil$ bytes! the vector is an array of Uint8_t's

For set and clr bits, use the bit twiddling from asgn 5

set vector[index/8] or= (1 shifted left (index %8))

clr vector[index/8] and= ~(1 shifted left (index %8))

node ADT:

has two pointers, to left and right children. NULL if none

they also contain, oldspeak → newspeak translation.

if it is badspeak (no translation) newspeak is set to NULL

these will be used to build binary search trees!

binary search tree: tree of Nodes

create: return NULL (empty tree)

delete: Post order traversal delete nodes

height: return 1 + Max(height of left tree, height of right tree)
use an if statement!

Size: return 1 + Size of right + Size of left
if NULL, return 0

insert: traverse the tree, comparing strings in order to find the correct spot.

find: traverse the tree, comparing strings to go to the right spot, if you hit NULL, the node is not there.

Bloom-filter:
bloom filter is a bit vector that uses hashes as indicies
create: make a bv / delete: delete a bv
size: size of the bv
insert: set bit @ hash of each oldspeak with 3 salts.
probe: check 3 indicres from the hash of oldspeak if they are all 1 return true
count: count the set bits in the bv check each indicy.

hash table: array of roots to binary trees.
size: return size
look up: hash word, check indicy of ht if ~NULL
then look up word in Tree.
insert: hash word, insert into binary tree
count: traverse table counting all Non NULL entries
avg_bst_size: average size of bst's in table
avg_bst_height: average height of bst's in table

hash-table

avg_size:
take size of every tree and divide it by the amount of Non NULL entries in the table

avg_height:
take the height of all the trees and divide it by the amount of Non NULL entries in the table

bloom filter

Size: Return the bf → size from the struct

Set Salts from "Salts.h"

Regex

- we need to recognize words with A-Z, a-z, 0-9 and "_", "-", and "'".

from Eugene's example [A-Za-z]+ captures word with A-Z,a-z so add 0-9, "_", "-", and "'" to allow for those symbols to work!

[A-Za-z0-9_-']+

banhammer.c:

we're going to create a hash table and a bloom filter with the size from the user!

we then read in from badspeak.txt and newspeak.txt and populate the hash table and bloom filter. we then parse through the text and checking the bloomfilter, if it is "in" the bloom filter check the hash table.

if it's not there then it's fine, if it's in there then we do one of two things. if there's no newspeak translation then add to badspeak. if there is, add to newspeak. after parsing text, check newspeak and badspeak for words. deliver message for which words were used in "messages.h" and then the words that were illegal!

banhammer.c:

keep found illegal words in one of two bst's: newspeaks or badspeaks

You then print this tree out when you need to tell the user which words were illegal. this way, you can print in alphabetic order!

Check size of these trees for which message to print. (or not to print!)