Assignment 3 — The Great Firewall of Santa Cruz
Bloom Filters, Linked Lists and Hash Tables

Prof. Darrell Long
Department of Computer Engineering

Spring 2017

1 Introduction

> *War is peace. Freedom is slavery. Ignorance is strength.*
>
> —George Orwell, 1984

You have been selected through thoroughly democratic processes (and the machinations of your friend Ernst Blofeld) to be the Dear and Beloved Leader of the Glorious People's Republic of Santa Cruz following the failure of the short-lived anarcho-syndicalist commune, where each person in turn acted as a sort of executive officer for the week. In order to promote virtue and prevent vice, and to preserve social cohesion and discourage unrest, you have decided that the Internet content must be filtered so that your beloved children are not corrupted through the use of unfortunate, hurtful, offensive, and far too descriptive language.

1.1 Bloom Filters

> *The Ministry of Peace concerns itself with war, the Ministry of Truth with lies, the Ministry of Love with torture and the Ministry of Plenty with starvation. These contradictions are not accidental, nor do they result from from ordinary hypocrisy: they are deliberate exercises in* doublethink.
>
> —George Orwell, 1984

The Internet is very large, very fast, and filled with degeneracy. The proliferation of foreign persons who not only do not speak English, they use it in corrupt and deviant ways. The untermenschen spend their days sending each other cat videos. Your hero, E.B. decided that in his country of Pacifica that a more neutral *newspeak* was required to keep the people content, pure, and from thinking too much. But how do you process so many words as they flow into your little country at 10 Gbits/second? The answer that comes to your brilliant and pure mind is that you use a *Bloom filter*.

A Bloom filter is a hash table, where the entries in the hash table are simply single bits. Consider h(*text*) = k, then if $B_k = 0$ then the entry is definitely missing; if $B_k = 1$ then the entry may be present. The latter is called a false positive, and the false positive rate depends on the size of the Bloom filter and the number of texts that hash to the same position (hash collisions).

You assign your programming minions to take every *potentially offensive* word that they can find in an English dictionary and hash them into the Bloom filter. That is, for each word the corresponding bit in the filter is set to 1.

You will make *two* such Bloom filters. Why? to reduce the chance of a *false positive*. Each Bloom filter should be relatively large (thousands of entries), and use a different hash function. This is easily accomplished by giving our hash function a different input block.

```
# ifndef NIL
# define NIL (void *) 0
# endif
# ifndef _BF_H
# define _BF_H
# include <stdint.h>
# include <stdlib.h>
# include <stdio.h>

typedef struct bloomF {
    uint8_t *v; // Vector
    uint32_t l; // Length
    uint32_t s[4];  // Salt
} bloomF;

// Each function has its own hash function, determined by the salt.

uint32_t hashBF(bloomF *, char *);

// Create a new Bloom Filter of a given length and hash function.

static inline bloomF *newBF(uint32_t l, uint32_t b[])
{
    // Code
}

// Delete a Bloom filter

static inline void delBF(bloomF *v)
{
    // Code
}

// Return the value of position k in the Bloom filter

static inline uint32_t valBF(bloomF *x, uint32_t k)
```

```
{
    // Code
}

static inline uint32_t lenBF(bloomF *x) { return x->l; }

// Count bits in the Bloom filter

static inline uint32_t countBF(bloomF *b)
{
        // Code
}

// Set an entry in the Bloom filter

static inline void setBF(bloomF *x, char * key)
{
    // Code
}

// Clear an entry in the Bloom filter

static inline void clrBF(bloomF *x, char *key)
{
    // Code
}

// Check membership in the Bloom filter

static inline uint32_t memBF(bloomF *x, char *key)
{
    // Code
}


static inline void printBF(bloomF *x)
{
    // Code
}

# endif
```

bf.h

When presented with a stream of text, it is first passed to the first layer Bloom filter. If the Bloom filter rejects any words then the person responsible is innocent of a *thoughtcrime*. But if the word has a corresponding bit set in the Bloom filter, it is likely that they are *guilty* of a thoughtcrime, and that is very ungood indeed. To make certain, we must consult the *hash table* and if the work is there as a *proscribed* word then they will be placed into the care of *Miniluv* and is sent off to *joycamp*. If the word passed both Bloom filters, and is not a forbidden word then the hash table will provide a translation that will replace offensive, insensitive, and otherwise dangerous words with new approved words. The advantage is that your government can augment this list at any time via the *Minitrue*. Minitrue can also add words from the Bloom filter that you, in your wisdom, have determined are not wholesome for your people to use. There are three cases to consider:

1. Words that are approved, these will not appear in the Bloom filter.
2. Words that should be replaced, which will have a mapping from the old word to the new approved word, and
3. Words where no mapping to new approved words means that it's off to *joycamp*.

You will use the *bit vector* data structure that you developed for *Assignment 1* to implement your Bloom filter.

1.2 Hash Tables

> *To send men to the firing squad, judicial proof is unnecessary . . . These procedures are an archaic bourgeois detail.*
>
> Ernesto Ché Guevara

You decide that the easiest way to make word replacements is through a translation table, with entries of the form:

```
typedef struct word {
    char *oldspeak, *newspeak;
} goodSpeak;
```

goodSpeak

Once a word passes the Bloom filter and thus is likely to be in your pure form of English, *Ingspeak*, you will use the hash table to locate that word, and if it is found then the system helpfully provides the appropriate new improved word in its place.

A hash table is one that is indexed by a function, `hash`, applied to the *key*. The key in this case will be the word in *oldspeak*. As we have said, this provides a mapping from *oldspeak* to *newspeak*. Words for which there is no mapping result in *joycamp*. All other words must have a replacement *newspeak* word.

What happens when two *oldspeak* words have the same hash value? This is called a *hash collision*, and must be resolved. Rather than doing *open addressing* (as discussed in class), we will be using *linked lists* to hold all of the translations for *oldspeak* words that have the same hash value.

1.3 Linked Lists

> *Education is a weapon, whose effect depends on who holds it in his hands and at whom it is aimed.*
>
> —Joseph Stalin

A *linked list* will be used to resolve hash collisions. Each element of the linked list contains a mapping from degenerate *oldspeak* to pure *newspeak*. The *key* to be searched in the linked list is `p->translation->oldspeak`.

```
# ifndef NIL
# define NIL (void *) 0
# endif
# ifndef MTF
# define MTF     true
# endif

# ifndef _LL_H
# define _LL_H

# include <stdbool.h>

extern bool moveToFront;

typedef struct listNode listNode;

struct listNode
{
    char *oldspeak, *newspeak;
    listNode *next;
};

listNode *newNode(const char *, const char *);

void delNode(listNode *);

void delLL(listNode *);

listNode *insertLL(listNode **, const char *, const char *);

listNode *findLL(listNode **, const char *);

void printLL(listNode *);
# endif
```

ll.h

You will be implementing this in two forms, and comparing the performance of the methods:

- Inserting each new word at the front of the list; and
- Inserting each new word at the front of the list, but *each* time it is accessed it is *moved to the front*.

You will keep track of the *average number of links followed* in order to find each word.

2 Your Task

- Read in a list of *forbidden* words, setting the corresponding bit for each word in both Bloom filters. This list will be made available.
- Read in a space-separated list of *oldspeak,newspeak* pairs. This list will be made available.
- Insert these mappings into the hash table.
- Read from *standard input* text (I/O redirection must be supported).
- If words are pass both Bloom filters, and have no mapping in the hash table this constitutes a *thoughtcrime*. You will send them a message with your decision (for their own good).
- If there is a mapping from *oldspeak* to *newspeak* then you will send them a message with your decision (again for their own good).

```
Dear Comrade,

You have chosen to use degenerate words that may cause hurt
feelings or cause your comrades to think unpleasant thoughts.
This is doubleplus bad. To correct your wrongthink and
save community consensus we will be sending you to joycamp
administered by Miniluv.

Your errors:

kalamazoo
antidisestablishmentarianism
```

Thoughtcrime

- If there is a mapping from *oldspeak* to *newspeak*, then you will write a note of encouragement.

```
Dear Comrade,

Submitting your text helps to preserve feelings and prevent
badthink. Some of the words that you used are not goodspeak.
The list shows how to turn the oldspeak words into newspeak.

sad -> happy
liberty -> badfree
music -> noise
read -> papertalk
write -> papertalk
```

Goodspeak

3 Specifics

First of all, you need a good hash function. We have discussed hash functions in class, and rather than risk having a poor one implemented, here is one based on the *Advanced Encryption Standard* (AES). You will find `aes.c` and `aes.h` in your repository.

```
# ifndef NIL
# define NIL (void *) 0
# endif

# ifndef _HASH_H
# define _HASH_H
# include <stdint.h>
# include "ll.h"

typedef struct hashTable hashTable;

struct hashTable
```

```
{
    uint32_t s[4]; // Salt
    uint32_t l;    // Length
    listNode **h;  // Array of pointers to linked lists
};

hashTable *newHT(uint32_t, uint32_t []);

void delHT(hashTable *);

listNode *findHT(hashTable *, const char *);

void insertHT(hashTable *, const char *, const char *);

uint32_t hash(hashTable *, const char *);

void printHT(const hashTable *);
# endif
```

hash.h

```
# include <stdlib.h>
# include <stdint.h>
# include <string.h>
# include "aes.h"
# include "hash.h"

static inline int realLength(int l)
{
    return 16 * (l / 16 + (l % 16 ? 1 : 0));
}

uint32_t hash(hashTable *h, const char *key)
{
    uint32_t output[4]  = { 0x0 };
    uint32_t sum        = 0x0;
    int keyL            = strlen(key);
    uint8_t *realKey    = (uint8_t *) calloc(realLength(keyL), sizeof(
   uint8_t));

    memcpy(realKey, key, keyL);

    for (int i = 0; i < realLength(keyL); i += 16)
    {
        AES128_ECB_encrypt((uint8_t *) h->s,             // Salt
                                    (uint8_t *) realKey + i, // Input
                    (uint8_t *) output);          // Output
        sum ^= output[0] ^ output[1] ^ output[2] ^ output[3];
    }
    free(realKey);
    return sum;
}

// And so forth
```

hash.c

Second, you will need a function to pick words from the text stream. The words should be just valid words, and these can include *contractions*. To aid you with this, here is a simple *lexical analyzer*. It is built using the `flex` program, and your access it by calling `yylex()`.

You are *not required* to use it, and can write you own in **C** if you wish. But it is good to learn to use the tools that are available.

```
%top{
# include <stdio.h>
}

LET [A-Za-z_]
DIG [0-9]
WS  [ \t\r\n]
PUNC    [~‘!@#$%^&*()_+-={}|\\\[\]:;\"\’<>,./?]

%%

{LET}+              { return  0; }
{LET}+"’"({LET}|{LET}{LET}) { return  0; }
{LET}+{DIG}+            { return  0; }
{PUNC}+             {}
{WS}+               {}
<<EOF>>             { return -1; }
.               {}

%%

// int main(void) { int res = yylex(); printf("%d - %s", res, yytext); }
```

words.l

You will need to transform all of your words from *mixed case* to *lowercase*.

Submission

You *must* turn in your assignment in the following manner:

1. The list of proscribed words is available as `~darrell/badspeak.txt` on `unix.ucsc.edu`.
2. The replacement mapping is available as `~darrell/newspeak.txt` on `unix.ucsc.edu`.
3. You must provide `./banhammer` with the following options:

- `./banhammer -s` will suppress the letter from the censor, and instead print the statistics that were computed.
- `./banhammer -h size` specifies that the hash table will have `size` entries (the default will be 10000).
- `./banhammer -f size` specifies that the Bloom filter will have `size` entries (the default will be 2^{20}).
- `./banhammer -m` will use the *move-to-front rule*.
- `./banhammer -b` will not use the *move-to-front rule*.
- Any combination of these flags *must be supported*.

4. Have a file called `Makefile` that when the grader types `make` will compile your program.

   - `CFLAGS=-Wall -Wextra -Werror -pedantic` must be included.
   - `CC=gcc` must be specified.
   - `make clean` must remove all files that are compiler generated.
   - `make` should build your program, as should `make all`.
   - Your program executable must be named `newspeak`.

5. You program *must* have the source and header files:

   - `aes.c` and `aes.h`.
   - `hash.c` and `hash.h`.
   - `banhammer.c` contains `main()` and *may* contain the other functions necessary to complete the assignment.

6. You may have other source and header files, but *do not try to be overly clever*.

7. A plain text file called `README` that describes how your program works.

8. The executable file produced by the compiler *must be called* `newspeak`.

9. These files must be in the directory `assignment3`.

10. You must `commit` and `push` the directory and its contents using `git`.