Derick Pan
Dpan7

ASGN7 Design

What is a Bloom Filter	A Bloom filter is a space-efficient probabilistic data structure, conceived by Burton H. Bloom in 1970, and is used to test whether an element is a member of a set.
What is a linked list	A list where each element points to the next and before.
What is a Hash Table?	A data structure that maps keys to values that provides O(1) look up times
What is a hash collision	When two or more objects have the same hash value that means same array indexes
What is a salt	Salt is additional data to add to the data through hashing.

Some of the functions/ADT's used in this assignment are from previous assignments such as the NODE adt, bitvector ADT.

General Scope of the task:

1. Getopt to parse choices in
2. Initialize Bloom Filter and Hash tables
3. Read in badspeak and add it to bf and ht. Then Do the same with Newspeak
4. Start up the Regex to filter the words.
5. Create Two linked lists to keep track of the badwords and Oldwords
6. Test each of the words by converting them to connect with the regex and Lowercase everything.
7. Finally, print out the message from GPRSC and badwords/oldwords
8. Free up leftover memory

The I/O used in this assignment will be an executable named banhammer.

USAGE

./banhammer [-hsm] [-t size] [-f size]

OPTIONS

-h Program usage and help.
-s Print program statistics.
-m Enable move-to-front rule.
-t size Specify hash table size (default: 10000).
-f size Specify Bloom filter size (default: 2^20).

Regex

- I need a way to parse out words in the form of an input stream. My input characters should include a-z, A-Z, 0-9, underscore, and quotation.

To do this I will use the regex library. In regex:
+ is one or more
* is zero or more
? is zero or one
| is or

Do this in accompaniment with parser.h.
next_word gives me the next word and clear_words frees up any leftover memory.

" [a - zA - Z]+ " means one or more characters between a-z lowercase and A-Z uppercase.

Because I need to include underscores, quotations and dashes. The necessary unique marks are “ '_- ”

BloomFilter

The Bloom filters struct has tree salts:

Primary	Secondary	Tertiary

The reason that there's three salts is to reduce the chance of a *false* positive. Each salt will have three different hash functions. This basically means there are three different hash functions.

BloomFilter Constructor

The constructor is supplied by the lab documentation, but I will need to implement the bit vector ADT for it.

BloomFilter Destructor

As the name implies this function will free memory allocated by the constructor and set to null pointer. The main thing to free is the filter.

BloomFilter size

Return the # of bits accessible by Bloom filter. // Return length of bit vector

BloomFilter Insert

Insert the argument into the Bloom filter.
-Hash oldspeak with Primary
-Hash oldspeak with Secondary
-Hash oldspeak with Tertiary
-Set the bits at those new indices with the bit vector

BloomFilter Probe

Return true if the bits at the hashed indices are set. // This means that it will return true if oldspeak was probably added to the Bloom filter.

BloomFilter count

Return the # of bits in the Bloom Filter. This is different from the size!!!

BloomFilter print

Debugger to print out the Bloom filter.

SPECK CIPHER

Fortunately for this assignment, a SPECK cipher is provided to us to reduce the probable pain seeing us create a poor filter.

What is SPECK	Speck is an XOR cipher.
What type of algorithm is this?	This is an asymmetric-key algorithm that uses the same key for encryption and decryption.

speck.h holds the interface for hash() that takes two parameters, a 128 bit salt in the form of an array, and a key to hash. Returning a uint32_t

BitVector

This bit-vector is similar to my assignment 5 so I have based the following design off of it.

Struct:

- length // Length in bits
- *vector // Array of bytes. It's uint8_t because we extended the last bit to be p3

Constructor and Destructor functions that do as the function name implies

Setbit(n)

- Byte =n/8
    - Now we have to locate the specific bit inside of that byte.
- Offset n % 8
    - byte= Byte (OR) 1 << (offset)
- One line: v-> vector[n/8] |= (1 << (n%8)

Ex.

Set bit (14)

Byte = 14/8 = 1 // Byte 1

Offset = 14 %8 = 6// Bit 14 is in byte 1 and bit 6 offset

To set the 6th bit

To set a bit in a byte, shift a "1" to the index then bitwise OR operation

= byte w/ bit set

Bv_clr_bit(n):

- Get 1's in every position BUT the bit
- MSB 0111 0110 LSB, Shift to the position, Invert it = 1101 1111 AND it, to become 0101 0110.
- v->vector[n/8] &= ~(1<< (n%8))

Bv_get_bit(n):

- v->vector[n/8] >> (n %8) & 0x1

MSB 0011 0010 LSB LS AND 0001 0000 0001 0000 Now shift right 4 back to position	MSB 0011 0010 LSB Right shift 4, 0000 0011 AND 0000 0001 = 0000 0001

Bv print(n):

- Debug function to print the bit vector

Hash Tables

The following is for the hash table that contains a salt which is passed to hash() when called to insert.
The struct contains a bool option called “mtf” that stands for move-to-front. This boolean chooses whether or not the linked lists uses move-to-front technique. Along as
size, // # of indices or linked lists that hash table can index up to.
salt,
and double pointer to linked list

HashTable Constructor & Destructor to do the standard creating and destroying by calling calloc then destroying by freeing memory and setting pointer to NULL
HashTable Size
    Return the hash table’s size
HashTable Lookup
    Search for a node that contains oldspeak
    If found{ return pointer to that node } Else {return NULL pointer}
HashTable Insert
    -If linked list isn’t initialized then create
    -Create a node for newspeak and oldspeak and insert it into the linked list by calculating the hash of oldspeak
HashTable Count
    -Returns # of non-NULL linked lists in hash table
HashTable Print
    -Debugger to print the contents of the hash table.

Node

In this asgn7, I use linked lists to avoid problems with hases. Linked Lists uses Nodes (similar to asgn6). Each node holds oldspeak and newspeak, and a pointer to prev and next node.

tdlr; Node contains the oldspeak, newspeak, pointer to prev node, and pointer to next node.

If newspeak is NULL then oldspeak = badspeak.

Node-create; arguments: oldspeak, newspeak

Create The node, allocate mem and copy the characters for old and newspeak.

- Optional to create my own STRDUP.

Node-Delete

destroy ONLY the node passed in, AND free mem for oldspeak and newspeak.
Set ONLY this pointer to NULL

Node-Print

Debugger function to print out the node.
Print out if oldspeak and or newspeak exists

Linked Lists

Struct:
length, head, tail, mtf

LinkedList Create (boolean mtf)

Linked list is initialized with two sentinel nodes that serve as head and tail.
-Every inserted node will go between two nodes
-IF mtf is true then nodes found through look-up is moved to the front of the list
-EVERY inserted node goes in between two nodes because there are already two nodes to start with. Though, inserting a node into the linked list means inserting at the head.

LinkedList Destructor

call each node using node_delete then set pointer to linked list as NULL

LinkedList length

Return the # of nodes in the linked list excluding the head and tail sentinel nodes

LinkedList lookup

Search for a node with the oldspeak argument. If found then return it. Else return NULL
If MTF then ALSO move the node to the front. (Increasing the time complexity.)

LinkedList Insert

Perform lookup to make sure the linked list doesn't contain a node with oldspeak already.
If it doesn't exist then Insert new node(old, new) into the linked list right after the head sentinel.
else{ Don't insert it}

LinkedList Print

Print the nodes EXCEPT for head and tail using node_print()