Brendan Boone
Brboone
11/8/2022

DESIGN:
Assignment 6: The Great Firewall of Santa Cruz: Bloom Filters, Linked Lists and Hash Tables

Description:

In this assignment, the objective is to make a “firewall” that understands ‘oldspeak’ and gives the ‘newspeak’ translation. As well as, give recommendations to the speaker. Like an “educational” language filter. By implementing a hashtable and a bloom filter, this can be done.

Pseudocode:

bf.c

Struct BloomFilter:
Declare array of salts N_HASHES long(uint64_t), declare number of keys input into bloom filter(uint32_t), declare number of proves that return true(uint32_t), declare number of probes that return false(uint32_t), declare number of bits actually examined(uint32_t). Declare an array filter “bloom filter”(BitVector).

Define uint64_t default salts[]

BloomFilter *bf_create(uint32_t size):
Allocate memory for the bloom filter. Check if the memory allocation failed. Set its number of keys, hits, misses, and bits examined to 0. For loop through the salts array assigning each index to corresponding BitVector. Then bv_create the filter(use size). Check if the allocation for the filter failed. If it did, then free it and return null. Else return the allocated bloom filter struct.

Void bf_delete(BloomFilter **bf):
Free the bloom filter BitVector using bv_delete. Then free Bloom filter and set its pointer to NULL.

Uint32_t bf_size(BloomFilter *bf):
Return this bloom filter’s BitVector filter’s length.

Void bf_insert(BloomFilter *bf, char *oldspeak):

Brendan Boone
Brboone
11/8/2022

Hash the oldspeak with each of the five salts. Indices found by modulo hash 64 bit integers by the size of the bit vector. Set the bits at those indices in the BitVector filter. Increment n_keys.

Bool bf_probe(BloomFilter *bf, char *oldspeak):
Hash the old speak with each of the five salts. Check if the bits at those indices are set in the BitVector. If all are set return true, else false. Increment hits or misses accordingly.

Uint32_t bf_count(BloomFilter *bf):
Length of the bit vector in terms of bytes is determined by ceiling division by 64(because each “byte” is defined as 64 bit integers). Traverse through the entire bit vector and keep count of any that are 1.
Return the count.

Void bf_print(BloomFilter *bf):
Print out the bloom filter’s bit vector.

Void bf_stats(BloomFilter *bf, uint32_t *nk, uint32_t *nh, uint32_t *nm, uint32_t *ne):
Set each passed pointer to the value of the BloomFilter attributes.

Bv.c

Struct BitVector:
Declare length(uint32_t) and vector array(uint64_t)

BitVector *bv_create(uint32_t length):
Contiguous allocate an array ‘length’ long. Check if the allocation failed. Return the pointer to the allocated memory. Length of the bit vector is in terms of “bytes” but because the “bytes” are in 64 bit variables, check if the size needs to be more than 1 or not. If not 1, use ceiling division to allocate length.

Void bv_delete(BitVector **bv):
Free the BitVector bv and set its pointer to NULL.

Uint32_t bv_length(BitVector *bv):
Return the length of the BitVector.

Void bv_set_bit(BitVector *bv, uint32_t i):
Byte_i is i/64. Bit_i is i % 64. Bitwise OR a byte with 1 left shift bit_i to set the bit in the byte to 1.

Brendan Boone
Brboone
11/8/2022

Void bv_clr_bit(BitVector *bv, uint32_t i):
Same as set_bit but bitwise AND a byte that is the opposite of 1 left shift bit_i to set the bit in the byte to 0.

Uint8_t bv_get_bit(BitVector *bv, uint32_t i):
Shift the byte bits to the right and bitwise AND it to 1.

Void bv_print(BitVector *bv):
Print the bit vector.

Ht.c

Struct HashTable:
Declare the salt of the hash(this is uint64 rest is 32), its size, the number of keys, number of probes hit, missed, and number examined. A boolean move-to-front variable. Then the hash's associated linked list.

HashTable *ht_create(uint32_t size, bool mtf):
Allocate memory for a hashtable. Check if the allocation failed. Set hash's mtf to mtf. Salt is given in assignment and initialized. Hits,misses,number of keys, and number examined are initialized as zero. Hash table size is initialized as 'size'. The hash table's linked lists are contiguously allocated in memory of length size. If the allocation fails, free it and set the Hashtable's pointer to NULL.

Void ht_delete(HashTable **ht):
Free up the memory allocated in the hash's linked lists. Free the hash table. Set the pointer passed in as NULL.

Uint32_t ht_size(HashTable *ht):
Return hashtable size.

Node *ht_lookup(HashTable *ht, char *oldspeak):
Find the index of the linked list by hashing the oldspeak. Modulo it to the hash table size to find its 32bit index, but if the size is >= 1 then set the index to 0. Call ll_lookup to look for the oldspeak at the LL index. If found, return the pointer to the node, else return NULL. update each of the stats. Call ll_stats before and after this function is run for debugging.

Void ht_insert(HashTable *ht, char *oldspeak, char *newspeak):

Brendan Boone
Brboone
11/8/2022

To find the linked list index to use, hash the oldspeak. Check if oldspeak is in the index first by using ll_lookup. If not, insert oldspeak and newspeak to the Linked List. If the Linked List hasn't been created yet, create it first.

Uint32_t ht_count(HashTable *ht):
Return the number of non-NULL linked lists in the hash table

Void ht_print(HashTable *ht):
Print the contents of the hash table.

Void ht_stats(HashTable *ht, uint32_t *nk, uint32_t *nh, uint32_t *nm, uint32_t *ne):
Set each passed pointer to the hashtable's respective stored element.

Node.c

Statc char *my_strdup(const char *s):
Implement a string strdup function to allocate memory for oldspeak and newspeak of the node.

Static int my_scmp_equal(char *wl, char *w2):
Compare if the two words are equal. If the words are ever unequal return 1. Else return 0. 1 is false, 0 is true. Used to confirm strdup worked.

Node *node_create(char *oldspeak, char *newspeak):
Allocate memory for oldspeak and newspeak. Copy over the characters from the parameters. Mimic strdup().

Void node_delete(Node **n):
Free memory for oldspeak and newspeak. Free the node then set its pointer to NULL. Direct the nodes this node is connected to to each other first, and then delete it.

Void node_print(Node *n):
If the node contains both oldspeak and newspeak, print both in a translational manner. If only the oldspeak, and the newspeak is NULL, then print only the oldspeak.

LI.c

Struct LinkedList:
Declare length, head sentinel node, tail sentinel node, and a boolean move-to-front variable.

Brendan Boone
Brboone
11/8/2022

Uint64_t seeks: used to track total seeks

Uint64_t links: used to track total links traversed

Static bool my_strcmp(char* s1, char* s2):
Check if the two strings are the same. Returns false if unequal, true if equal.

LinkedList *ll_create(bool mtf):
Initialize and allocate for a linked list. In the struct, initialize the length (0) and mtf(mtf). Allocate memory for nodes head and tail.

Void ll_delete(LinkedList **ll):
Free each node in between the head and tail of the linked list. Use tmp node to traverse. Free the linked list and set its pointer to NULL.

Uint32_t ll_length(LinkedList *ll):
Return the length of the linked list.

Node *ll_lookup(LinkedList *ll, char *oldspeak):
Traverse the linked list from head to the last element, if the node is found return its pointer, else return NULL. if mtf was set to true, the found node is moved to the front of the linked list. Move the node by unassociating the nodes around it first, then attach the node to the head, then fix the head and “2nd element”.

Void ll_print(LinkedList *ll):
Print each node in the linked list.

Void ll_stats(uint32_t * n_seeks, uint32_t * n_links):
Copy the number of n_seeks and n_links to the parameters.

Parcer.c

Struct Parser:
Declare file variable f. Declare current_line string with max length MAX_PARSER_LENGTH. Declare unsigned 32 bit integer line_offset.

Static void my_set_string(char *res, char *string):
Set res to a copy of string then terminate it with a ‘\0’

Static void add_char(char *w, char c):

Brendan Boone
Brboone
11/8/2022

Traverse to the end of the word, add c to the end, then add the null terminator at the end.

Static void emptyS(char *s):
Traverse s, null every character.

Static int get_strlen(char *s):
Traverse s. Return i.

Parser *parser_create(File *f):
Allocate memory for struct. Set its file pointer to f. Current_line and line_offset to equivalents to zero. Set the current_line to the next available line in *f.

Void parser_delete(Parser **p):
Free memory of p. Set the pointer to Null.

Bool next_word(Parser *p, char *word):
Empty word so it can be filled up again. Declare a buffer string pointer and a char used to keep track of what the current seen character is. While true: c is the next character in the line by the lines line_offset. If c doesn't exist, break if the buffer is empty, else set word to the buffer and return true. Else if the character is a space or newline, return true and set the word to buffer if buffer has any contents. Else c is lowered to its lower case. If c is an invalid character, return what's in the buffer. Else add the current character to the buffer. If the while loop was broken then word returns NULL and the function returns false.

Banhammer.c

Static void print_h(void):
Prints the help message

Static void empty_s(char *s):
Empties the string pointer

Static void trim_char(char *s):
Trims the last character in the string by setting a null terminator. Used to get rid of newline characters from newspeak and badspeak.txt.

static void nextw(char *res, char *string, int *i):

Brendan Boone
Brboone
11/8/2022

Small word parser using *i as the line_offset. Used to find oldspeak and newspeak in newspeak.txt

Static bool my_scmp_equal(char *w1, char *w2):
Returns false if unequal at all. Else true.

Static int get_strlen(char *s):
Traverse s. Return i.

main():
Parse these flags with getopt:
-h: prints out the program usage
-t size: specifies that the hash table will have ‘size’ entries. Default 10000
-f size: specifies that the Bloom filter will have ‘size’ entries. Default 2^19
-m: enable move-to-front
-s enable printing of statistics to stdout. The stats from hash and bloom.

Initialize bloom filter and hashtable. Then grab all the words from badspeak using fgets. Also newspeak.txt after, but this time implementing the nextw function and a predeclared int i variable. Then use parser to look through the stdin of any words. Keep creating new parsers until NULL is returned. Each parser reads a line in the stdin, next word grabs every valid word found in that line, and then this process repeats until EOF is reached. For every word found, check if it’s length is greater than 0. Probe it in the Bloom filter. If it is in the bloom filter, check the hashtable. If the hashtable has the oldspeak with the a NULL newspeak, insert it into a predefined LinkedList holding found badspeak words. Do the same with found words with newspeak translations and insert them in a separate Linked List. If stats wasn’t flagged, depending on the length of the badspeak list and the oldspeak list, print the correct message from message.h. Find the stats for Bloom Filter load, average seek length, False positives and bits examined per miss. If stats wasn’t flagged, print all the found badspeak and oldspeak in that order. Else only print the stats calculated during the execution. At the very end delete all allocated memory.