Tanya Gyanmote
11/10/22
1887664
tgyanmot@ucsc.edu

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

Description:

The purpose of this assignment is to keep the citizens of Santa Cruz in check as the glorious leader of the Glorious People's Republic of Santa Cruz. Creating a system for filtering out using the bloom filter, linked lists, nodes, bit vectors, and hashtables. In the bloom filter, which is composed of bit vectors, that is a one-dimensional array. A hash table is a structure that converts keys to values; nodes communicate with other data structures. A linked list consists of nodes, each of which refers to the next node. In the Bloom filter, the hash of a word is found using salts and indexes it using bit vectors. The hash table incorporates linked lists and is stored by nodes. All of the badspeak or newspeak phrases will be filtered in the message, similar to the bloom filter. Then in banhammer, it either punishes or rewards the user for their poor or good message.

Files:

- Banhammer.c
    - contains main()
- Messages.h
    - Defines the mixspeak, badspeak, and goodspeak messages that are used in banhammer
- Cityhash.h
    - Defines the interface for the hash function using CityHash
- Cityhash.c
    - Contains the implementation of the hash function using CityHash
- Ht.h
    - Defines the interface for the hash table ADT
- Ht.c
    - Contains the implementation of the hash table ADT.
- Ll.h
    - Defines the interface for the linked list ADT
- Ll.c
    - Contains the implementation of the linked list ADT.
- Node.h
    - Defines the interface for the node ADT

- Node.c
    - Contains the implementation of the node ADT
- Bf.h
    - Defines the interface for the Bloom filter ADT.
- Bf.c
    - Contains the implementation of the Bloom filter ADT.
- Bv.h
    - Defines the interface for the bit vector ADT.
- Bv.c
    - Contains the implementation of the bit vector ADT.
- Paser.h
    - Defines the interface for the parsing module
- Paser.c
    - Contains the implementation of the parsing module.
- Makefile
    - Contains program compliation
- Readme.md
    - Contains a description and usage of program
- Writeup
    - Analysis of program
-

Bloom Filter:

```
BloomFilter *bf_create(uint32_t size) {

  BloomFilter *bf = (BloomFilter *) malloc(sizeof(BloomFilter));
  if (bf) {
    bf->n_keys = bf->n_hits = 0;
    bf->n_misses = bf->n_bits_examined = 0;
    for (int i = 0; i < N_HASHES; i++) {
      bf->salts[i] = default_salts[i];
    }
    bf->filter = bv_create(size);
    if (bf->filter == NULL) {
      free(bf);
      bf = NULL;
    }
  }
  return bf;
}
```

bf_create

- Using the given code
- Dynamically allocating a bit vector which makes up the bloom filter and adding salts 1-5
- Setting all the stats to 0
- Afor loop which goes through the salts and sets them to their default values

- After that set the bloom filter to a bit vector
- Check if it's null

bf_delete
- Check if the bloom filter is null and if it isn't
  - Check if the filter of the bloom filter isn't null
    - Delete the filter
  - Free the bloom filter
  - Set the bloom filter to null

bf_size
- Return the number of bits the bloom filter can access
  - Using bv_length

bf_insert
- Using oldspeak and add it to bloomfilter
- hash it with each one of the salts, and mod it with the size of the bloomfilter
  - using bv_set_bit
- Add one to keys for stats

bf_probe
- Check if oldspeak was hashed into the bloomfilter, with all 5 salts and if they all equal 1 return true, if not then return false
- If the bit is false then add one to misses and return false
- If they are true then add one to hits, and return true

bf_count
- Return the number of set bits in the bloom filter
- Have a loop and goes through the blom filter and count every bit

bf_print
- Print the vector of bf using bit vector print
- bf_stats
- Setting each of the passed pointers for each variable from struct, keys,hits,misses,bits examined to a certain statistic corresponding with the variable

Bit Vector:

- Bv_create
  - Dynamically allocates an array of bites to hold with atleast a length bits and sets length
    - Using malloc
    - If bv isn't null
      - Have a check to see if the length mod 64 is 0
        - Then divide the length by 64
      - If it's not then divide the length by 64 and add 1

                - Allocate an array of uint64_t sized elements using calloc
            - If the bit vector is null then free it
        - Return the vector
- Bv_delete
    - Free the memory of all bytes in the vector, and the vector
    - Free the vector of bit vector
    - Free the bit vector
    - Set the bit vector to null
- Bv_length
    - Return the length in bytes of the bit vector
- Bv_set_bit
    - Get the location and position
    - Using the bitwise operator to left shift by 1, which gets bit of i setting it to 1
- Bv_clr_bit
    - Get the location and position
    - Using the bitwise operator to left shift by 1, which gets bit of i setting it to 0
- Bv_get_bit
    - Gets the ith bit of the vector
        - Gets the bit of the vector at index of i divided by 64,, bitwise at 1 moding it by 64
- Bv_print
    - Prints the bytes of the vector
        - Have a loop which goes until the length
            - And print each bitvector using bv_get_bit

Hash Table

```
HashTable *ht_create(uint32_t size, bool mtf) {
  HashTable *ht = (HashTable *) malloc(sizeof(HashTable));
  if (ht != NULL) {
    ht->mtf = mtf;
    ht->salt = 0x9846e4f157fe8840;
    ht->n_hits = ht->n_misses = ht->n_examined = 0;
    ht->n_keys = 0;
    ht->size = size;
    ht->lists = (LinkedList **) calloc(size, sizeof(LinkedList *));
    if (!ht->lists) {
      free(ht);
      ht = NULL;
    }
  }
  return ht;
}
```

- Ht_create

    - Using the given code
    - Dynamically allocate hash table using malloc
    - Set the variables from struct: salt,size,n_hits,n_keys,lists, mtf
    - Se lists to an array based on the size of Linked lists
- Ht_delete
    - Free the memroty ftom all lists and the hash table
    - If the hash table isn't null
        - Have a for loop that goes through the hash table
        - If the list exists then delete the list at ht lists[i] using ll_delete
        - Free the pointer of ht lists
    - Free the pointer of ht
    - Set the pointer to NULL
- Ht_size
    - Return the size of the hash table
- Ht_lookup
    - Create 4 variables for the original links, seeks and updated links, and seeks to get n examined
    - Get the linked lists stats of original seeks and links
    - Hash oldspeak using salts and mod by the size of the hash table
    - If the list is null return null
        - Add one to n misses
    - If not then call linked list lookup
        - And add one to hits
    - Get the ll stats of updated links and seeks
    - Return ll lookup of the hash table list and oldpseak
- Ht_insert
    - Hash oldspeak using salts and mod by the size of the hash table
    - Add one to n keys
    - If the list is null
        - Set the hash table list to linked list create of mtf
    - If not then call linked list insert
- Ht_count
    - Have a counter variable
    - Have a for loop that goes till the size of the hash table
    - If the element in list isn't null
        - Add 1 to counter variable
    - Return the counter variable
- Ht_print
    - Have a for loop that goes till the size of the hash table
    - If the element in list isn't null

        - Call linked list print and print
- Ht_stats
    - Setting each of the passed pointers for each variable from struct, keys,hits,misses,bits examined to a certain statistic corresponding with the variable

Node

- Included string functions given to us in lecture 5 from Professor Miller
    - String length
    - String copy
- Node_create
    - Allocate memory for each word then make a string copy of it into allocated memory
    - Since there is a null value add one to the size of char
    - Check if the node is null
    - Set next to the previous pointer to null because ther is only 1 node
- Node_delete
    - If the node is true then
        - Free the pointer of node n
        - Free oldspeak
        - Free newspeak
        - Set pointer n to null
- Node_print
    - If the node n contains oldspeak and newspeak, print out the node with oldspeak and newspeak
    - If node n only contains oldspeak then print out with just oldspeak

Linked Lists

- Ll_create
    - Have dummy chars, initalize them and size it a size
    - Allocating memory for linked list with the size of linked list
    - If the linked list is null
        - Return null
    - Set the length to 0
    - Create the head using node_create
    - Set the previous to null
    - Create the tail using node_create
    - Set the previous to null
    - Set mtf to the mtf passed
    - Return the linked list
- Ll_delete

    - Set the current node to the lead of the linked list
    - While it's not null
        - Set the next node to current
        - Set the current to next
        - Delete the next node
    - Free the linked list
    - Set the linked list to null
- ll_length
    - Return the length of the linked list
- ll_lookup
    - Add one to seeks
    - Have a null check for the linked list
    - Loop through the array until its reached the end
        - If the node with oldspeak is found and if the move to the front is true move it
            - Add one to links
            - Return the node which was found
        - Add one to links
    - If node isn't found then return null
- ll_insert
    - If lookup of the linked list and oldspeak are null then
        - Node a node with the passed vraibles of oldspeak and newspeak
        - Set the next node to the next of linked list
        - Set the previous to the linked list head
        - Set the next linked list to node
        - Set the previous ndoe to node
        - Add the linked list length to 1
- ll_print
    - Prints out each node in the linked list except for the head and tail sentinel nodes
- ll_stats
    - Setting each of the passed pointers for each variable from struct, links and seeks examined to a certain statistic corresponding with the variable

Parser

- parser_create
    - Allocationg memory for pointer parser p
    - Point it to the file
    - Set line_offset to 0
    - Return the pointer
- parser_delete
    - Set the file to null

    - Free the pointer
    - Set the pointer to null
- Next_word
    - Have a while loop to check if it's not a "-" , "', or a character
        - Check if you are at the end of a line
            - Using fgets to the new line
            - Count the number of characters in the line so you know where u are in the line, so subtract one from lineoffset
            - Set the length of the word to 0
        - Add one to p_lineoffset
        - Have a null check for the line
            - Set a char to the end of a file
            - Set the word to null
            - Return false
    - Have a for loop that goes max_parser_line_length plus 1 minus lineoffset
        - Set a variable to this value
    - Check if the character is a valid character
    - If its true
        - Then add the word to the char of words
    - If it's false then
        - Check if its a dash or a single quote and if it is then its a valid character
            - Then add the word to the char of words
            - Increment the variables to count the number of characters
        - Else you want to set the word to the end if a line
            - Add to line offset
            - Increment the variable which counts the number of characters
    - Add one to line offset
    - Return true once u finish reading the file

Banhammer

- Have all the includes
- Print_h
    - With all the print statements including stderr
- Define all the options
- Set corresponding variables needed to the values used
        - opt is 0
        - Set statistics variable to false
        - Table size to 10000
        - Filter size to 2 to the power of 19
        - Mtf to false

- Have a while for user input, use get opt
- Have different cases for each input
    - Default:
        - Call the help function
        - Return -1
    - Case h:
        - Call the help function
        - Return 0
    - Case s:
        - Set stats variable to true
            - To print out the stats
    - Case t:
        - Sets the table size
            - Make sure its not less than 0
                - If it is then print a error message
                    - And return -1
    - Case f
        - Sets the filter size
            - Make sure its not less than 0
                - If it is print error message
                    - And return -1
- Initalize char's for storing the words
    - One for newspeak, oldspeak,badspeak and words
- Initalize hash table.
    - Using table size and mtf
    - Have a null check
- Initialize bloom filter
    - Using filter size
    - check if it isn't null
- Open the badspeak file and read it
- Set it to a parser using parser create
- Check if it's null
    - If it is print a error message and return -1
- Have a loop that inserts the badspeak words
    - Using next_word check if the word isn't false, if thats true
        - Use bf insert and ht insert to add the word
- Have a loop that inserts the newspeak words
    - Using next_word check if the word isn't false, if thats true
        - Use bf insert and ht insert to add the word
- Create two linked lists one for rightspeak and one for thought crime using mtf

    - Have null checks
- filter out words, while reading words from stdin with the created parsing module. With next word
    - For each word that is read in, check to see if it has been added to the Bloom filter using bf prove
    - If it has not been added to the Bloom filter, then do nothing
    - If it was added to the bloom filter then
        - if the hash table contains the word and the word does not have a newspeak translation insert badspeak into the list of badspeak words
        - if the hash table contains the word and the word does have a newspeak translation insert oldspeak into the list of oldspeak words
- Using the ll length for thought crime and right speak and make sure that stats isn't called to see which message to print
    - If the length of thought crime and right speak is greater than zero then print the mixspeak message thought crime, and rightspeak linked list
    - If the length of thought crime is greater than 0 and right speak is equal zero then print the badspeak message thought crime
    - If the length of thought crime is equal to 0 and right speak is greater than zero then print the goodspeak message and rightspeak crime
- If stats boolean is called it turns true so if it's true then print out statistics
    - Have variables for all the struct variables from ht and bf
    - Call hash tables stats and bloom filter stats to grab the values
    - Print them all out
    - Have float vraibles for bloom filter bits examined, false positives, seek length, and bloom filter load
        - If they equal to zero print 0
        - If they aren't 0 then
            - For bloom filter bits examined
                - It equals to bloom filter examined minus 5 times bloomf filter hits divided by bloom filter misses
            - For false positives
                - The value equals to hash= tables misses divided by bloom filter hits
            - For seek length
                - It equals to hash table examined divided by hash table hits plus hash table misses
            - For bloom filter load
                - It equals to bloom filter count divided by bloom filter size
- Delete all the parsers
- Delete the bloom filter

- Delete the hashtable
- Delete both linked lists for rightspeak and thought crime
- Return 0