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 as the glorious leader of the Glortious People's republic of Santa Cruz, and the leader of this nation the citizens must be kept in check. Crating a filtering out system using bloomfilter, linked lists, nodes, bit vectors, and hashtables. Using the bloomfilter, which is made up of bit vectors. A bit vector is a abstract data type representing a one dimensional array. A hash table is a structure that maps keys to values, a node which plays a part with the other data structures. A linked list consists of nodes where each node contains a data field and reference to the next node. There is a use of badspeak and newspeak words that will be filtered out, in banhammer program which is respond to certain messages

Files:

1. Banhammer.c
    a. contains main()
2. Messages.h
    a. Defines the mixspeak, badspeak, and goodspeak messages that are used in banhammer
3. Cityhash.h
    a. Defines the interface for the hash function using CityHash
4. Cityhash.c
    a. Contains the implementation of the hash function using CityHash
5. Ht.h
    a. Defines the interface for the hash table ADT
6. Ht.c
    a. Contains the implementation of the hash table ADT.
7. Ll.h
    a. Defines the interface for the linked list ADT
8. Ll.c
    a. Contains the implementation of the linked list ADT.
9. Node.h
    a. Defines the interface for the node ADT
10. Node.c
    a. Contains the implementation of the node ADT

11. Bf.h
    a. Defines the interface for the Bloom filter ADT.
12. Bf.c
    a. Contains the implementation of the Bloom filter ADT.
13. Bv.h
    a. Defines the interface for the bit vector ADT.
14. Bv.c
    a. Contains the implementation of the bit vector ADT.
15. Paser.h
    a. Defines the interface for the parsing module
16. Paser.c
    a. Contains the implementation of the parsing module.
17. Makefile
    a. Contains program compliation
18. Readme.md
    a. Contains a description and usage of program
19. Writeup
    a. 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

bf_delete

- Check if bf is null and if it isn't
    - Free the bit vector from bloom filter
    - Free the bloom filter

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

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

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 thebit of the vector at index(i/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
    - Hash oldspeak using salts and mod by the size of the hash table
    - If the list is null return null
    - If not then call linked list lookup
- Ht_insert
    - Hash oldspeak using salts and mod by the size of the hash table
    - If the list is null return null
    - 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

- Node_create
    - Allocate memory for each word then make a string copy of it into allocated memory
    - Add 1 to the size of char because of null
    - Set next to the previous pointer to null because ther is only1 node
- Node_delete
    - 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
  - 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 tf passed
  - Return the linked list
- ll_delete
  - Return the length of the linked list
- ll_lookup
  - Using extern variable
    - Add one to it
  - Loop through the array until its reached the end
    - Add 1 to the other extern variable
    - If the node with oldspeak is found and if the move to the front is true move it
      - Return the node which was found
  - If node isn't found then return null
- ll_insert
  - If lookup of the linked list and oldspeaka re 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 temp variable set to false
  - If the current line is a new line then
    - Then call fgets
    - Set lineoffset to 0
  - Have a for loop that goes max_parser_line_length minus lineoffset
    - Add 1 to lineoffset
    - set another variable to linepffset
  - If j is at the end of file
    - Return false
      - If j is true then
        - Append to the word
      - If temp is true
        - Set it to false
    - If j and tmep are false
      - Set temp to true
    - If tempo is true then return the word

Banhammer

- Have all the includes
- Define all the options
- Have a print function for help
- Set corresponding variables needed to the values used
- Have a while for user input, use get opt
- Have different cases forr each input
  - Default:
    - Call the help function

        - Return 0
    - 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
    - Case f
        - Sets the filter size
            - Make sure its not less than 0
- Initalize hash table.
- Initialize bloom filter
    - Make sure they aren't null
- Read in a list of oldspeak using fgets()
- Read in a list of oldspeak and newspeak pairs with fgets()
- filter out words, while reading words from stdin with the created parsing module.
    - For each word that is read in, check to see if it has been added to the Bloom filter.
    - 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