Assignment 7 DESIGN.pdf
Yash Sharma

Description of Program:

This assignment requires us to implement a program that attempts to identify the most likely authors for an anonymous sample of text when given a large database of texts with known authors. In other words, we will be using an algorithm that's commonly used in machine learning to identify authors of anonymous samples of text. This algorithm is also known as the k-nearest neighbor's algorithm. By utilizing this algorithm, we will classify samples of texts and determine which authors are most likely to have authored an anonymous sample of text.

Files to be included in directory "asgn7":

1. bf.c:
    - This file contains the implementations of the Bloom filter ADT.
2. bv.c:
    - This file contains the implementation of the bit vector ADT.
3. bf.h:
    - This file defines the interface for the Bloom filter ADT.
4. bv.h:
    - This file defines the interface of the bit vector ADT.
5. ht.h:
    - This file defines the interface for the hash table ADT and the hash table iterator ADT.
6. ht.h:
    - This file defines the interface for the hash table ADT and the hash table iterator ADT.
7. identify.c:
    - This file contains main() and the implementation of the author identification program.
8. metric.h:
    - This file defines the enumeration for the distance metrics and their respective names stored in an array of strings.
9. node.h:
    - This file defines the interface for the node ADT.

10. node.c:
    - This file contains the implementation of the node ADT.
11. parser.h:
    - This file defines the interface for the regex parsing module.
12. parser.c:
    - This file contains the implementation of the regex parsing module.
13. pq.h:
    - This file defines the interface for the priority queue ADT.
14. pq.c:
    - This file contains the implementation for the priority queue ADT.
15. salts.h:
    - This file defines the primary, secondary, and tertiary salts to be used in the Bloom filter implementation. This file also defines the salt used by the hash table in the hash table implementation.
16. speck.h:
    - This file defines the interface for the hash function using the SPECK cipher.
17. speck.c:
    - This file contains the implementation of the hash function using the SPECK cipher.
18. text.h:
    - This file defines the interface for the text ADT.
19. text.c:
    - This file contains the implementation for the text ADT.
20. Makefile:
    - A file that formats program into clang-format and compiles it into program executables with make/make all from Makefile.
    - Additionally, make clean from Makefile must remove compiler-generated files (such as the executables)
21. README.md:
    - This file describes how to use the program and MakeFile. It will also list and explain any command-line options the program accepts.
22. DESIGN.pdf:
    - This describes the design for the program thoroughly with pseudocode and descriptions.
23. WRITEUP.pdf:
    - This describes the detailed write-up for the program thoroughly with pseudocode

and descriptions.

Pseudocode / Structure:

bf.c

Include header files

Create the structure for Bloom Filter

Bf_create

- Allocate memory for the bloom filter
- Set the lower salt primary
- Set the higher salt primary
- Set the lower salt secondary
- Set the higher salty secondary
- Set the lower salt tertiary
- Set the higher salt tertiary
- Create the size for the bloom filter
- Return the bloom filter

Bf_delete

- Create the destructor for a Bloom Filter
- Free memory allocated by the constructor
- Set the pointer to null

Bf_size

- Return the length of the underlying bit vector

Bf_insert

- Set the index of the primary salt
- Set the index of the secondary salt
- Set the index of the tertiary salt
- Set the bit for index 1
- Set the bit for index 2
- Set the bit for index 3

Bf_probe

- Set the index for the primary salt
- Set the index for the secondary salt
- Set the index for the tertiary salt
- Get the bit for the filter at index 1
    - Add to count
- Get the bit for the filter at index 2
    - Add to count
- Get the bit for the filter at index 3

        - Add to count
- If count is equal to 3
    - Return true
- Else
    - Return false

Bf_print

- Debugging function to print out the bits of a bloom filter

bv.c

Include header files

Create the structure for the BitVector

Bv_create

- Allocate memory for the bit vector
- Check if the bit vector is not null
    - Initialize the length
    - Allocate memory for the vector
    - Return the bit vector
- Return null

Bv_delete

- Free the pointer to the bitvector
- Set the pointer of the bitvector to null
- Free the bitvector
- Null the bitvector

Bv_length

- Return the length of the bit vector

Bv_set_bit

- Sets the bit of a bit vector
- If the iterator is out of the range
    - Return false
- Else
    - Return true

Bv_clr_bit

- Clears the bit of a bit vector
- If the iterator is out of range
    - Return false
- Else
    - Return true

Bv_get_bit

- Return the bit in the bit vector
- If the iterator is out of range
    - Return false
- If iterator value is 0
    - Return false

- If iterator value is 1
    - Return true

Bv_print

- A debugging function that prints out the bits of a bit vector

ht.c

Include header files

Create a structure for the Hash Table

Create a structure for the Hash Table Iterator

ht_create

- Create the constructor for a hash table
- Allocate memory for the hash table and allow the function to return NULL
- Initialize hash table to 0 for each slot
- Initialize the salts for low and high on the hashtable
- Allocate memory for the slots
- Return the hash

ht_delete

- Loop through the hash table size
- If the slots of the hash table do not equal null,
    - Delete the node from the hash table for each slot iterated
- Free the hash table slots
- Set the hashtable slots to NULL
- Free the hashtable
- Set the hashtable to NULL

ht_size

- Return the hash table's size and the number of slots it can index up to

Ht_lookup

- Initialize the count var
- Call the hash function when calculating the index
- While count is less than the size of the hash table
    - Initialize the node to the index of the slot
    - If the second node and the word from the first node equal 0,
        - Return the second node
    - Update the index
    - Update count
- Return null

Ht_insert

- Initialize the count var
- Call the hash function when calculating the index
- Loop through count when it is less than the size of the hash table
- Check if the index of slots is null
    - Create a node using word and set it equal to the index
    - Update the count in slots to 1

        - Return the index of the slots
    - Compare the index word with input word using an if statement
        - Add to the count
        - Return slots to the index
    - Update index
    - Add to the count
    - Return null

Ht_print

- Create hash table iterator
- Set a node equal to null
- While the hash table iterator exists, print out the node
- Delete the hash table iterator

Hti_create

- Allocate memory for the hash table iterator
- Initialize the slot equal to 0
- Initialize the table to the hashtable
- If the hash table iterator exists, then return it
- Return null

Hti_delete

- Free the hash table iterator
- Set pointer to null

Hti_iter

- Loop through the slot of the hash table iterator when it is not equal to the table size
    - If the slot of the hash table iterator exists, update slot
    - Return the index of the slot within the hash table iterator
    - Else
        - Update the slot
- Return null

node.c

Node_create

- Allocate memory for a node on the heap
- If the node exists
    - Return the pointer to the word string
    - Initialize the count
    - Return the node
- Return the node

Node_delete

- Free the pointer of the node to word
- Set the pointer to null
- Free the node
- Set the node equal to null

Node_print

- Debugging function to print out the word and count of the node

parser.c

Set header files

Initialize static and nonstatic variables for byte, mask, index, max index, buffer, bytes read, and bytes written

Create a program that prints out the words input to stdin using the parsing module

pq.c

Create a struct for the PriorityQueue
Fill the struct with head, tail, capacity and a double pointer Q for the Node

Create a struct for Author
Fill the struct with author and dist

Pq_create
- Allocate memory on the heap for the priority queue using malloc
- If the priority queue exists
- Set the head and tail = 0
- Set the capacity pointer equal to capacity
- Set the Node pointer equal to allocating memory for the Node pointer
- If the priority queue points to Node Q
- Return the priority queue
- Free the priority queue
- Return the priority queue

Author_create
- Function purpose is to create an author
- Allocate memory on the heap
- If the author exists
    - Set a pointer to the string of the author
    - Initialize the dist
    - Return the author
- Return the author

Pq_delete
- If the pointer to the priority queue exists
- Free the priority queue pointing to the Node
- Free the priority queue pointer
- Set the priority queue pointer to NULL

Pq_empty
- Set the head of the priority queue equal to 0

Pq_full
- Set the head of the priority queue equal to the capacity

Pq_size

- Set the head of the priority queue equal to the pointer

Enqueue

- Make a bool for full and set that equal to the pq_full function with the priority queue as a parameter
- If the queue is not full
- If the priority queue is empty
    - Set the pointer of the priority queue equal to a variable
    - Set the pointer of the priority queue pointing at the header +1
    - Return true
- Else
    - Set the pointer of the priority queue equal to a variable
    - Set a for loop that checks the range between 1 and the length of the array
    - Set the iterator equal to the variable
    - Initialize and move the value to a temp variable
    - Create another loop that checks the distance
    - Subtract from the index of the variable pointer
    - Set the distance equal to the temp
- Add to the header
- Return false

Dequeue

- If the priority queue exists
    - If the priority queue is empty
        - Subtract from the head of the queue
        - Return true
- Return false

Pq_print

- Create a for loop that loops through the head of the priority queue
    - Initialize and set a pointer to the author
    - Initialize and set a pointer to the distance
- Create a debugging function to check the head, tail, and capacity of the priority queue

speck.c

Include header files

- Create a function that will return a uin32_t which is exactly the index the key is mapped to

Text.c

Include header files

Create the structure for text

Fill it with hash table, bloom filter, and word count

Text_create

- Allocate memory for the text

- Create the hash table
- Create the bloom filter
- Set the regex expression
- Initialize the word count to 0
- Set the character word to null
- Loop through the length of the word
- Set the word to lower case based on the iterations
- If there is no noise in the file,
    - Insert the word into the hash table
    - Insert the word into the bloom filter
    - Add to the word count
- If there is noise in the file
    - If the word doesnt contain noise
        - Insert the word into the hash table
        - Insert the word into the bloom filter
        - Add to the word count
- Free the regex
- Return text

Text_delete

- Deletes a text
- Free the hash table and bloom filter
- Free the text
- Set pointer to null

Text_dist

- If the metric is euclidean
    - Create a hashtable iterator
        - Create a for loop that iterates through the hash table
        - If the text contains the word that is iterated through the hash table,
            - Check the frequency of the first text
            - Check the frequency of the second text
            - Add to the metric by subtracting the two frequencies
    - Create a hashtable iterator
        - Create a for loop that iterates through the hash table
        - If the text contains the word that is iterated through the hash table,
            - Check the frequency of the first text
            - Check the frequency of the second text
            - Add to the metric by subtracting the two frequencies
    - Metric equals the square root of the metric
- Return metric

Text_frequency

- Lookup the word and set it equal to the node
- If word is not in the text
    - Return 0
- Else
    - Return frequency

Text_contains

- Prob the bloom filter for word if true
    - Return the word found from looking in the hash table
- Return false

Text_print

- Debugging function that prints out the contents of a text

identify.c

Include header files

- Setup get-opt functions for d, n, k, l, e, m, c, and h
- Create a new Text from text passed into stdin
- Create a new Text that contains word into the noise.txt file
- Using fopen, open the database of authors and text files via lib.db
- Call the priority queue function and create a queue that can hold up to n elements
- Use fgets to read the name of an author
- Use fgets to read the path
- Remove newline character from fgets result
- Open author's file of text and use it to create a new Text instance
    - If file cannot be opened
        - Read next line in the database file
- Compute the distance between author's and anonymous Text
    - Filter out words in the noise word Text file
- Dequeue the priority queue and get the top k

Credit:

- I attended Eugene's section on 3/4/21, which helped give me general guidance on how to approach this lab, as well as sketch out how the program runs.
- I used the pseudocode from the asgn7 documentation.
- I also used the asgn7 documentation and reviewed the pseudocode provided by Elmer in discord.