Shaan Mistry
6 March 2022
Professor Long

Assignment 7: DESIGN.pdf

Description of Program:
This program identifies the most likely authors of an anonymous sample text given a database of text with known authors. The program uses the k-Nearest Neighbors algorithm (KNN) to classify samples of text and determine which authors are most likely to have written the anonymous sample of the text.

The program creates n-dimensional vectors for each text where n is the number of unique words in the text. It then computes the distance between the known samples of text and the anonymous text. The three possible distance metrics are:

Manhattan Distance:

$$\text{MD} = \sum_{i \in n} |u_i - v_i|$$

Euclidian Distance

$$\text{ED} = \sqrt{\sum_{i \in n} (u_i - v_i)^2}$$

Cosine Distance

$$\vec{u} \cdot \vec{v} = \sum_{i \in n} u_i \times v_i$$

Files to be include in "asgn7":

1. bf.h
   - Interface for the Bloom filter ADT.
2. bf.c
   - Implementation of the Bloom filter ADT.
3. bv.h
   - Interface of the bit vectors ADT.
4. bv.c
   - Implementation of the bit vector ADT.
5. ht.h
   - Defines the interface for the hash table ADT and the hash table iterator ADT.
6. ht.c
   - Implementation of hash table ADT and the hash table iterator ADT.
7. indentify.c

- .

8. metric.h
   - Contains the enumeration for distance metrics and their respective names.
9. node.h
   - Interface for the node ADT.
10. node.c
    - Implementation of the node ADT.
11. parser.h
    - Defines the interface for the regex parsing module.
12. parser.c
    - Contains the implementation of the regex parsing module.
13. pq.h
    - Interface for the priority queue ADT.
14. pq.c
    - Implementation of the priority queue ADT.
15. salts.h
    - Contains definitions for the primary, secondary, and tertiary salts which are used in the Bloom filter.
    - Defines the salt used by the hash table.
16. speck.h
    - Defines the interface for the has function using the SPECK cipher.
17. speck.c
    - Contains the implementation of the hash function using the SPECK cipher.
18. text.h
    - Interface for the text ADT.
19. text.c
    - Implementation of the text ADT.
20. Makefile
    - Directs compilation processes for all **C** files.
    - Formats .c and .h files.
    - Cleans object files and executables.
21. README.md
    - Markdown Format.
    - Contains information on how to build the program, how to run the program, and how to clean up and object files and binaries created from building the program. It can also contain information on possible bugs or errors in the program.
22. DESIGN.pdf
    - This file.
    - Contains pseudocode and explanations of the program.
23. WRITEUP.pdf
    - Contains observations about the program.
    - Contains analysis on the output of the program based on varied input.
    - Provides explanations on how the different metrics compare against each other.

Pseudocode/ Structure:

node.c

```
FUNCTION node_create:
    Allocate memory to node struct.
    Copy inputted word to the node word.
    Set the count of the node to 0.
    Return pointer to node.

FUNCTION: node_delete:
    IF the node pointer is not NULL:
    Free the allocated memory for the Node struct.
    Set the node pointer to NULL.

FUNCTION node_print:
    Print out the word and count of the node.
```

ht.c:

```
FUNCTION: ht_create:
    Allocate memory for a hash table struct.
    Set the salt array to their defined values.
    Set the size of the hash table to the inputted size.
    Set allocate memory for an array of nodes.

FUNCTION: ht_print:
    FOR i in range (0, size of the hash table):
        IF hash table slot at index i is NULL:
            Print "Empty Slot"
        ELSE:
            Print the node at slot index i.

FUNCTION: ht_delete:
    IF the pointer to the hash table is not NULL:
        Free the memory allocated for the array of nodes.
        Free the memory allocated to the hash table struct.
        Set the pointer to NULL.

FUNCTION ht_size:
    Return the size of the inputted hash table.

FUNCTION ht_lookup:
    WHILE  the count is less than the size of the hash table:
        IF the current Node is not NULL and has same word:
            Return the node;
        IF the current node is NULL:
            Return NULL:
        Increment the index around the hashtable
        Increment the count by 1.

    Return NULL.

FUNCTION ht_insert:
    IF (ht_lookup is not NULL):
        Increate the count at the returned node pointer by 1.
        Return the node pointer.

    Calculate the hash of the inputted word and create an index for the table based on its size.
    DO;
        IF the slot at the current index is NULL:
            Create a new node with the inputted word.
            Increment the count of the node by 1.
            Return the pointer to the node.
        Increment the index by 1 and mod index by the size of the table.
    WHILE h does not equal the stop index.
    Return NULL.


STRUCT HashTableIterator: table, lots.

FUNCTION hti_create:
    Allocate memory for a has table iterator struct.
    Set the table of the struct to the inputted hash table.
    Set the slot of the iterator to 0.
    Return the pointer to the hash table iterator.

FUNCTION hti_delete:
    IF the hti pointer is not NULL:
        Free the memory allocated the the hash table iterator struct.
        Set the pointer to NULL.

FUNCTION ht_iter:
    IF the hash table iterator slot is equal to the size of its hash table:
        Return NULL.
    Increment the slot by 1.
    Return the hash table iterator table at index of the slot – 1.
```

bv.c

```
STRUCT BitVector: length, vector.
```

```
FUNCTION bv_create:
    Allocate memory for a bit vector struct.
    Calculated the number of bytes needed to hold inputted length.
    Set the length of the bit vector to the inputted length.
    Allocate memory for the array of bytes based on the calculated number of bytes needed.

FUNCTION bv_delete:
    IF the pointer to the bit vector is not NULL:
        Free the memory allocated to the array of bytes.
        Free the memory allocated the the bit vector struct.
        Set the pointer to NULL.

FUNCTION bv_length:
    Return the length of the inputted bit vector.

FUNCTION bv_set_bit:
    IF the inputted index is greater than or equal to the length of the bit vector:
        Return false.
    Set the bit in the bytes vector at inputted index to 1.


FUNCTION bv_clear_bit:
    IF the inputted index is greater than or equal to the length of the bit vector:
        Return false.
    Set the bit in the bytes vector at inputted index to 1.

FUNCTION bv_get_bit:
    IF the inputted index is greater than or equal to the length of the bit vector:
        Return false.
    Return the bit in the vector at the inputted index.

FUNCTION bv_print:
    FOR i in range(0, number of bytes in the vector):
        Print the the array at index i.
```

<u>bf.c</u>

```
STRUCT BloomFilter: primary, secondary, tertiary.

FUNCTION: bf_create:
    Allocate memory for a BloomFilter struct.
    Set the defined salts for primary.
    Set the defined salts for seconday.
    Set the defined salts for tertiary.
```

```
        Set the filter the result of calling the function bv_create with the inputted size.

        Return the pointer to the bloom filter.

FUNCTION bf_delete:
        IF the pointer to the bloom filer is not NULL:
                Free the memory allocated to the filter.
                Free the memory allocated to the bloom filter struct.
                Set the bloom filter pointer to NULL.

FUNCTION bf_size:
        Return the length of the filter.

FUNCTION bf_insert:
        Calculate the hash of the inputted word with the primary salt.
        Calculate the hash of the inputted word with the secondary salt.
        Calculate the hash of the inputted word with the tertiary salt.

        Set the index from the primary hash in he bit vector to 1.
        Set the index from the secondary hash in he bit vector to 1.
        Set the index from the tertiary hash in he bit vector to 1.


FUNCTION bf_probe:
        Calculate the hash of the inputted word with the primary salt.
        Calculate the hash of the inputted word with the secondary salt.
        Calculate the hash of the inputted word with the tertiary salt.

        IF the all three of the bits at the hash indices are already set:
                Return True.
        Return False.


FUNCTION bf_print:
        Print out the filter of the bloom filter.
```

pq.c

```
STRUCT: Pair: author, distance

STRUCT: PriorityQueue: tail, capacity, pairs

FUNCTION pq_create:
        Allocate memory for priority queue struct.
        IF the allocation was successful:
```

```
        Set the tail of the priorty queue to 0.
        Set the capacity of the priority queue to the inputted capacity.
        Allocate memory for the array of nodes.
        IF the allocation of the array of nodes is successful:
            Return the pointer to the priory queue
        Free the memory allocated.
    Return NULL.

FUNCTION: pair_create:
    Allocate memory for a Pair struct.
    Duplicate the inputted word.
    Set the distance to the inputted distance.

FUNCTION pair_delete:
    IF the pointer is not NULL:
        Free the author string.
        Free the pair struct.

FUNCTION pq_empty:
    Return the tail of the queue.

FUNCTION pq_full:
    IF the tail of the priority queue equals the capacity of the priority queue:
        Return true.
    ELSE:
        Return false.


FUNCTION fix_heap:
    Set the min index to the inputted index.
    IF the priority queue size is less than 2:
        Return;
    IF the left is not NULL and the left pair is less than the min pair:
        Set the min pair to the left pair index.
    IF the right pair is not NULL and the right pair is less than min pair:
        Set the min pair to the right pair index.

    IF min index does not equal the inputted index:
        Swap the min node with the smaller node.


    FUNCTION min_heap:
        IF the child node is smaller than the parent node:
            Swap the child node and the parent node.
        IF the parent is a root:
            Call min_heap with the parent.
```

```
FUNCTION enqueue:
    Create a pair with the inputted parameters.

    IF the priority queue is full:
        Free the pair.
        Return false.

    Set the next open slot to the created pair.

    IF the priority queue is not empty:
        Call function min heap.

    Increment the tail.


FUNCTION dequeue:
    IF the priority queue:
        Return false.

    Set the author distance pointers to the values from the index 0 of the priority queue.
    Delete the pair.
    Call fix_heap function at index 0.
    Decrement the tail.

FUNCTION pq_print:
    FOR i in range(0, size of the priority queue):
        Print out the author and distance for the current pair.
```

<u>text.c</u>

```
Define the regex WORD:

Define the

Text Struct: ht, bf, word_count.

FUNCTION text_create:
    Allocate memory for a text program.

    IF the allocation was successful:
        Create a hash table of size 2^19
        Create a bloom filter of size 2^21
        Set the word count to 0.

    IF the hash table and bloom filter creations were successful:
        Parse the text file.
        WHILE the next word is not NULL:
            Convert the word to lowercase.
            IF the word is in the noise text:
                Continue.
            IF the noise text is NULL and the word count is equal to the nosie limit:
                Break.
            Insert the word into the hash table.
            Insert the word into the bloom filter.
            Increment the word count.
        Fre the regex comp
        Return the text.

    Return NULL.

FUNCTION text_delete:
    If the text pointer is not NULL:
        Delete the text hash table.
        Delete the text bloom filter.
        Free the text struct.
        Set the pointer to NULL.

FUNCTION text_dist:
    Create a hash table iterator for text1 and text2.
    IF the metric is equal to MANHATTAN:
        WHILE c
            IF the current word is in the second text:
                Add the difference between the frequency of the text1 and text2
words to sum.
            ELSE:
                Add the text1 frequency of the current word.
        WHILE there are words left in the hash table of the second text:
            IF the current word is not in the first text:
                Add the text2 frequency of the current word.
        Delete the hash table iterators.
        Return the sum.
    IF the metric is equal to COSINE:
        WHILE there are words left in the hash table of the first text:
            IF the word is also in the second text:
                Add the product between the frequency of the text1 and text2
words to sum.

        Delete the hash table iterators.
        Return the sum.

    WHILE:
        IF the current word is in the second text:
```

```
        Add the squard difference between the frequency of the text1 and text2 words to sum.
            ELSE:
                Add the squared text1 frequency of the current word.
        WHILE there are words left in the hash table of the second text:
            IF the current word is not in the first text:
                Add the square text2 frequency of the current word.
        Delete the hash table iterators.
        Return the  the square root of the sum.


    FUNCTION text_frequency:
        IF the word is not in the text:
            Return 0.
        Return the normalized frequency of the word in the text.


    FUNCTION text_contains:
        IF the word is in the bloom filter:
            IF the word is in the hash table:
                Return true
        Return false.

    FUNCTION text_print:
        Print out the word count of the text.
        Print out the hash table of the text.
        Print out the bloom filter of the text.
```

<u>identify.c</u>

```
    FUNCTION usage:
        Print out the usage message for the program.

    main:

        Parse the command line options using getopt:

        Check that the database file is valid.
        Check that the noise file is valid.

        Create a new noise Text with the noise file.
        Create a new anonymous sample Text with stdin.


        Read the first line fscanf
```

```
Create a new priority queue.

WHLE fgets() hasnt reached the end of the file:
    Remove the trailing newline from the author string and path strings.

    Open the author file using fopen().
    Check that the author file is valid.

    Compute the distance between the author's Text and the  sample Text.

    Enqueue the author and distance into the priority queue.
    Close the author file
    Delete the text.


Print out the results.

Close the database and noise files.
Delete noise and sample texts.


Delete the priority queue.
```

Error Handling:
Check the database file is valid.
Check the noise file is valid.
Check that the author file is valid.