# DESIGN.pdf
# Assignment 7: Author Identification

CSE 13s Winter 2022

Darrell Long

By Santosh Shrestha

## Description:

This program will take in an anonymous or random text as an input and will be able to identify the most likely author of said text. The program uses a k-nearest neighbors algorithm to do so rather than machine learning compared to a more modern-day version of this program. The program will calculate the distance or the similarities between text using a cosine, manhattan, or Euclidean distance formula depending on what the user specifies

Files to be included in directory “asgn7”:

1. bf.h: Defines the interface for the Bloom filter ADT.
2. bf.c: Contains the implementation of the Bloom filter ADT.
3. bv.h: Defines the interface for the bit vector ADT.
4. bv.c: Contains the 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: Contains the implementation of the hash table ADT and the hash table iterator ADT.
7. identify.c: Contains main() and the implementation of the author identification program.
8. metric.h: Defines the enumeration for the distance metrics and their respective names stored in an array of strings.
9. node.h: Defines the interface for the node ADT.
10. node.c: Contains the 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: Defines the interface for the priority queue ADT.
14. pq.c: Contains the implementation for the priority queue ADT.
15. salts.h: Defines the primary, secondary, and tertiary salts to be used in your Bloom-filter implementation. Also defines the salt used by the hash table in your hash table implementation.
16. speck.h: Defines the interface for the hash function using the SPECK cipher.
17. speck.c: Contains the implementation of the hash function using the SPECK cipher.
18. text.h: Defines the interface for the text ADT.
19. text.c: Contains the implementation for the text ADT.

## Node.c

## Node_create

Functionality:

A function that creates a node and allocates the needed space for the node and returns the node pointer.

Pseudocode:

Initialize node_create as a pointer function to the struct that takes in a string pointer
- Initialize a Node pointer variable and allocate the needed space for each node
- Equate the nodes word to the input string. Note: The function strdup might be useful for this.
- Equate the nodes count to 0

## Node_delete

Functionality:

Deletes the node by freeing any allocated space within the node.

Pseudocode:

Initialize node_delete as a void function that takes in the parameters of a double pointer to the struct Node called n
- Free allocates space of the nodes' word
- Free the allocated space for the node
- Equate the node to NULL

## Node_print

Functionality:

Used for debugging and making sure that the node is created properly.

Pseudocode:

Initialize node_print as a void function that takes in a pointer node as a parameter
- Print the node and its' contents.

# ht.c

## HashTable(struct)

Functionality:

A struct that contains an array of nodes that holds a word and the number of times that word appears

Pseudocode:
Initialize HashTable as a struct
    Initialize a variable/array called salt that has the index of 2 in the heap as an uint64_t
    Initialize a variable called size as an uint32_t
    Initialize a Node double-pointer called slots

## HashTableIterator(struct)

Functionality:
    Keeps track of the current iteration of the HashTables slot.

Pseudocode:
Initialize HashTableIterator as a struct
    Initialize a HashTable pointer called a table
    Initialize a variable slot as a uint32_t

## Ht_create

Functionality:
    Creates a HashTable which contains an array of nodes that holds words and the number of times it appears in the given text

Pseudocode:
Initialize ht_create as a HashTable pointer function that takes in a uint32_t variable called size
    Initialize a Hashtable pointer variable and allocate the needed space for it
    Equate the hashtable pointers' size to the value of the inputted size
    Initialize the pointer's slots by allocating the needed space for it

## Ht_delete

Functionality:
    Deletes the hashtable by freeing all of the allocated space within it

Pseudocode:
Initialize ht_delete as a void function that takes in a double pointer hashtable
    Create a hashtable iterator with the given hash table
    Create a pointer node as a NULL
    While loop that iterates as long as the iterator doesn't return NULL
        Set each node within the hash table to NULL
        And once the node returns NULL delete the node n
    Free the allocated space for the hashtables slots
    Delete the hash table iterator

Free the allocated space of the hashtable
Equate the hashtable to NULL

## Ht_size

Functionality:

Returns the current size of the hashtable

Pseudocode:

Initialize ht_size as a uint32_t that takes in a HashTable pointer
Return the given hashtables' current size

## Ht_lookup

Functionality:

Returns the current index of the input word within the hashtables array of nodes

Pseudocode:

Initialize ht_lookup as a Node pointer function that takes in a HashTable and a char pointer called word
Initialize a counter variable with the value of 0
Initialize a variable called index that contains the
While loop that iterates as long as the count is less than the hash tables size
Initialize a node pointer called thing and equate it to the hashtables array with the index of index
If thing and the arrays string is the same as the inputted string
Return the node of the index in the hash tables slots

Equate index to the value of index plus one modulus the size of the has table
Increment the count of the nodes words by one

## Ht_insert

Functionality:

Adds a word to the hashtable and increases the count of the word

Pseudocode:

Initialize insert as a Node pointer function that takes in a HashTable pointer and char pointer
Initialize a counter with the value of 0
Initialize a variable called index with the value of hash with the parameters of the hashtables salt and the imputed word
While the count is less than the size of the hashtable

If the hashtables array at the index of an index is null
Equate the array at the index of the index with the value of the node created with the given word
Equate the hash table slot with the index of index count and increment it by 1
Increment the count by 1
Return the hash table slots with the index of index
If the input word is in the hashtable
Increment the hashtables node that contains the word by 1
Return the node that contains the same word
Increment count by 1
Equate index to an index plus one modulus the size of the hashtable
Return null

## Ht_print

Functionality:

A function that prints the hashtable to make sure its values are correct and that it has been created

Pseudocode:

Initialize ht_print as a void function that takes in a hashtable pointer
Print the hashtable and the values within it

## hti_create

Functionality:

Creates the hashtable iterator allowing the program to go along the hashtable since C does not provide one

Pseudocode:

Initialize hti_create as a HashTableIterator pointer that takes in a hashtable pointer
Allocate the needed space for the hashtable iterator
Equate the hashtable iterators' table to the hashtable
Equate the slots to 0
Return the hash table iterator

## hti_delete

Functionality:

Deletes the inputted hashtable by freeing the allocated space within it

Pseudocode:

Initialize hti_delete as a pointer that takes in a hashtable pointer

Free the hash table iterator

Equate the hash table iterator to NULL

## ht_iter

Functionality:

A function that iterates through the hash table

Pseudocode:

Initialize ht_iter as a Node pointer function that takes in a HashTableIterator pointer variable

Initialize a Node pointer with the currents hashtable iterators tables' slots with the

The current iteration of the hash tables iterators slot

Increment the hashtable iterators slot by 1

Return the Node pointer that was created earlier

# bv.c

## BitVector(struct)

Functionality:

A struct that contains an array of bit values that correlates to a words bit value

Pseudocode:

Initialize a struct called BitVector

Initialize a uint32_t called length

Initialize a uint8_t pointer called vector

## bv_create

Functionality:

A function that creates a bit vector and allocates the needed space for the bit vector. The function should also make sure there are enough bits created. For an example of the length of 10 is inputted it should create 16 bits.

Pseudocode:

Initialize bv_create as a BitVector pointer function that takes in a uint32_t called length

Initialize a variable called mod that has the value of length modulus 8

Initialize a variable called div that holds the values of length divided by 8

If mod is greater than 0

Increment div by 1
Equate length to div times 8
Allocate the needed space for the bit vector
Equate the bit vectors' length to the variable length
Allocate the needed space of the bit vectors vector
Return the bit vector pointer

## bv_delete

Functionality:

A function that deletes and frees all the allocated space of the function.

Pseudocode:

Initialize bv_delete as a void function that takes in a BitVector double-pointer called bv
Free the bvs' vector
Free bv
Equate bv to null

## bv_length

Functionality:

A function that returns the imputed bit-vectors length.

Pseudocode:

Initialize bv_length as a uint32_t that takes in a BitVector pointer
Return the bit vectors length

## bv_set_bit

Functionality:

A function that sets the inputted bit index of the bit vector to 1 and returns true if the task has been completed properly. However, the function will return false if the task wasn't properly completed.

Pseudocode:

Initialize bv_set_bit as a bool that takes in a BitVector pointer and a uint32_t called i
Complete a bitwise or between the bit vector at the index and the binary value that has all 0's
except for in the bit value that correlates the index of i
Return this value because it will either return a 0 or 1 indicating a true or false

## bv_clr_bit

Functionality:

A function that sets the inputted bit index of the bit vector to 0 and returns true if the task has been completed properly. However, the function will return false if the task wasn't properly completed.

Pseudocode:

Initialize bv_clr_bit as a bool that takes in a BitVector pointer and a uint32_t called i
Complete a bitwise operation between the bit vector at the index and the binary value that has all 1's except for in the bit value that correlates the index of i where it will have a 1
Return this bit value of the indexed bit-vector index

## bv_get_bit

Functionality:

A function that returns whether the bit value at the given index is a 1 or a 0.

Pseudocode:

Initialize bv_get_bit as a bool that takes in a BitVector pointer and a uint32_t called i.
Return the wanted bit vector bit with the given index divided by 8 and right shift it by i modulus 8 then do a bit-wise operation with the binary value of 1

## bv_print

Functionality:

A useful debugging tool.

Pseudocode:

Print every bit value within the bit vector.

# bf.c

## BloomFilter(struct)

Functionality:

A struct that contains an array of bit-vectors that will be used to determine if a word is within a given text.

Pseudocode:

Initialize a struct called BloomFilter
Initialize a uint64_t called primary with 2 slots of space on the stack
Initialize a uint64_t called secondary with 2 slots of space on the stack
Initialize a uint64_t called tertiary with 2 slots of space on the stack
Initialize a BitVector pointer called filter

## bf_create

Functionality:

A function that creates the bloom filter and allocates the needed space for the bloom filter and its' bit vector. The function will assign the proper salts to the primary and secondary, etc struct values of the bloom filter

Pseudocode:

Initialize bv_create as a BloomFilter pointer function that takes in a uint32_t called size
Initialize a BloomFilter pointer called bf and allocate the needed space for it
Assign the proper salt variables to the corresponding salt holding variables within the bloom Filter
Create the bloom filters bit vector called filter with the given size
Return the bloom filter pointer

## bf_delete

Functionality:

A function that deletes and frees all the allocated space of the function.

Pseudocode:

Initialize bf_delete as a void function that takes in a BloomFIlter double-pointer called bf
Delete the bloom filters bit-vector called filter
Free bf
Equate bf to null

## bf_size

Functionality:

A function that returns the bloom filter's size.

Pseudocode:

Initialize bv_delete as a void function that takes in a BitVector double-pointer called bv
Return the bit vector length of the bloom filters filter times 8

## bf_insert

Functionality:

A function that inserts the inputted words' bit value into the bloom filter.

Pseudocode:

Initialize bf_insert as a void function that takes in a BloomFilter pointer called bf and a string called word

Initialize a variable called bits that contains the hash value of the bloom filters primary and the inputted word
Equate bits to bits modulus the bloom filters size
Call the function bv_set_bit() with the bloom filters filter as the bit vector and bits as the Index
Repeat this process for the tertiary and secondary salts

## bf_probe

Functionality:

That probes through the bloom filter to see if the inputted word is within the bloom filter

Pseudocode:

Initialize bf_probe as a boolean function that takes in a BloomFilter pointer called bf and a string called word
Initialize a variable called pits that equates to the value returned from hashing the primary salts and the word
Equate pits to pits modulus bloom filter size
Repeat this for the tertiary and secondary salts with the variable names sits and bits
If the bloom filters bit vector are all 1's at the index of pits, sits, and bits
Return true
Otherwise, return false

## bf_print

Functionality:

A useful debugging tool.

Pseudocode:

Print every bit value within the bloom filter.

# pq.c

## Auth(struct)

Functionality:

A struct that contains the author names and the correlated distance values.

Pseudocode:

Initialize Auth as a struct
Initialize a char pointer called name
Initialize a double called dista

## PriorityQueue(struct)

Functionality:

A struct that contains an array of the author names and the associated distance values that will be used to determine if a word is within a given text. This struct will also be used to determine the authors with the highest priority

Pseudocode:

Initialize PriorityQueue as a struct
- Initialize a uint32_t variable called head
- Initialize a uint32_t variable called tail
- Initialize a uint32_t variable called capacity
- Initialize a struct Auth double-pointer called A

## Auth_create

Functionality:

A function that allocates the needed space for the Auth struct

Pseudocode:

Initialize Author_create as an Auth pointer function that takes in a string called name and a double called distances
- Allocate the needed space for an Auth pointer variable
- Equate the pointers name variable to the inputted string
- Equate the pointers dista to the value of double integer that was inputted

## Auth_delete

Functionality:

A function that deletes and frees all the allocated space of the Auth type struct variable.

Pseudocode:

Initialize Auth_delete as a void function that takes in an Auth double pointer
- Free the pointers name
- Free the pointer
- Equate the pointer to NULL

## insertion_sort

Functionality:

The insertion code iterates through the values within the given array and compares the previous value with the current one. If the current value is less than the previous value it will then swap the two values positions. The program will continue to do this until the current distance value is of a lower value than the previous distance value.

Pseudocode:

Initialize the insertion sort as a void, that takes in a PriorityQueue pointer
  For loop starting at iteration 1, increments iteration by 1, and continues as long as iteration value is less than priority queues head
    Initialize j as the value of the current iteration
    Initialize tmp as an Auth pointer with the value of the priority queues array of authors with The current iteration of i as the index
    While loop that continues as long as j >0 and a[j-1] distance is greater than the tmp distance value
      Set a[j] equals to a[j-1]
      Set j to j minus 1
  Set a[j] to tmp
Return statement

## Pq_create

Functionality:

A function that allocates the needed space for the priority queue

Pseudocode:

Initialize pq_create as a pointer to PriorityQueue that takes in the parameters of a uint32_t called capacity
  Initialize a PriorityQueue pointer q with the needed allocated size of a PriorityQueue
  If q exists
    Set q's head to 0
    Set q's tail to 0
    Set q's capacity to the variable capacity
    Allocate the needed space of a double node pointer to qs' A
    If pointer q's A exists
      Return q
  Otherwise free q
  Return PriorityQueue pointer NULL

## Pq_delete

Functionality:

A function that deletes and frees all the space that was made in the pq_create.

Pseudocode:

Initialize pq_delete as a void function that takes in a PriorityQueue pointer variable called q
  If pointer q exists
    Iterate through all the Auths in the q and delete them
    Free pointer q's A(array of authors)
    Free pointer q
    Set pointer q to NULL

## Pq_empty

Functionality:

A function that returns whether a priority queue is empty or not

Pseudocode:

Initialize pq_empty as a bool that takes in the parameters of a PriorityQueue pointer variable called q
  If q exists
    If head and tail equate to each other
      Return true
  Return false

## Pq_full

Functionality:

A function that returns whether the priority queue is empty or not.

Pseudocode:

Initialize pq_full as a bool that takes in the parameters of a PriorityQueue pointer variable called q
  Initialize a uint32_t with the value of q's capacity
  If q exists
    If q's head is equal to cap
      Return true
  Return false

## Pq_size

Functionality:

A function that returns the current size of the priority queue in otter words the number of elements within it.

Pseudocode:
Initialize pq_size as a uint32_t that takes in the parameters of a PriorityQueue pointer variable called q
    Return q's head

## Enqueue

Functionality:
    A function that adds the inputted author names and the associated distance into the priority queue.

Pseudocode:
Initialize enqueue as a boolean that takes in the parameters of a PriorityQueue pointer called q, a string called author, and a double called dist.
    If q exists
        If q is full
            Return false
        Create and Auth pointer called a with the given author string and distance
        Set the current index value head of A to the value of a
        Increment q's head by 1
        Call the function insertion_sort with the values of q, and pq_size(q)
        Return true
    Else
        Return false

## Dequeue

Functionality:
    A function that sets the inputted string and double integer with the value of the highest priority variable in the priority queue

Pseudocode:
Initialize dequeue as a boolean that takes in the parameters of a PriorityQueue pointer called q, a string called author, and a double called dist
    If q exists
        If q is empty
            Return false
        Decrement q's head by one
        Set author to the current q's author with the index of the heads name
        Set the distance to the distance of q's A at the index of the current priorities queue
        Return true

Else
Return false

## Pq_print

Functionality:

A function used for debugging and making sure the priority queue was properly created.

Pseudocode:

Initialize the function pq_print as a void function that takes in the parameters of a PriorityQueue pointer variable called q
Print a helpful message to make sure the pq.c file works properly

# Text.c

## Text(struct)

Functionality:

A struct that contains a bloom filter and hashtable to contain the words within a text.

Pseudocode:

Initialize Text as a struct
Initialize a HashTable pointer called ht
Initialize a BloomFilter pointer called bf
Initialize a uint32_t called word_count

## euc_dist

Functionality:

A helper function that calculates the euclidean distance between two texts using the euclidean distance formula.

Pseudocode:

Initialize euc_dist as a double that takes in two text pointer variables
Initialize a double variable, Node pointers, and hash table iterators for each of the texts
Initialize a long variable that will hold that the total values
Iterate through each word in text1 via a hash table
If text2 contains the same word
subtract the frequencies of the word in both texts into the total and square the values
Otherwise
Increment the total by the number of occurrences in text1 squared
Iterate through each word in text2 via a hash table
If text1 doesn’t contain the word

Increment the total by the frequency of the word in text2 squared
Square root the value of total and return it

## cos_dist

Functionality:

A helper function that calculates the cosine distance between two texts using the cosine distance formula.

Pseudocode:

Initialize cos_dist as a double that takes in two text pointer variables
Initialize a variable called total to hold the value of 1
Iterate through each word in text1 via a hashtable
If text2 contains the word
Multiply the two frequencies and subtract them from the value of total
Return total

## man_dist

Functionality:

A helper function that calculates the Manhattan distance between two texts using the Manhattan distance formula.

Pseudocode:

Initialize man_dist as a double that takes in two text pointer variables
Initialize a variable called total to hold the total value
Iterate through each word in text1
If text2 contains the word
Subtract the frequencies from one another and add the absolute value to a total
Else
Add the frequency of the word in text1 to the total
Iterate through each word in text2
If the word isn't in text1
Add the frequency of the word in text2 into the total
Return the total

## text_create

Functionality:

A function creates a text or a noise depending on if the inputted text is null or not. This function will (if the text is null) will read in from the file and copy the words over into the texts bloom filter and the hash table and this will become the noise. However, if the text input isn't

null then it will use the noise to make sure that specific words aren't added into the text bloom filter and hash table.

Pseudocode:

Initialize text_create as a text pointer function that takes in a file and another text pointer called noise
- Initialize the regex
- Allocate the needed space for the text pointer
- Create a hash table with 2^19
- Create a bloom filter with 2^21
- If neither was properly created delete and free them
- If the noise is null
  - Iterate through each word from the file as long as the word count is less than the noise limit
    - Lowercase all the words
    - Insert the word into the hashtable and bloom filter
    - Increment the word count
    - Return the text
- Else
  - Iterate through each word in the infile
    - Lowercasify all the words
    - If the word isn't in the noise bloom filter
      - Insert it into the texts bloom filter and hashtable
      - Increment the word count
  - Return text
- Return null

## text_delete

Functionality:

A function that frees and deletes all the allocated space that was created in text_create

Pseudocode:

Initialize text_delete as a void function that takes in a Text double-pointer called text
- Delete texts bloom filter
- Delete texts hashtable
- Free text
- Set text to NULL

## text_dist

Functionality:

A function that uses finds the distance between two texts using the manhattan, euclidean, or the cosine distance formulas depending on which one is specified.

Pseudocode:

Initialize text_dist as a double that takes in two text pointer variables and a Metric
  If the metric is manhattan
    Return the value returned from man_dist with text1 and text2 as the parameters
  If the metric is cosine
    Return the value returned from cos_dist with text1 and text2 as the parameters
  If the metric is euclidean
    Return the value returned from euc_dist with text1 and text2 as the parameters

## text_frequency

Functionality:

A function returns the frequency of a word within the text divided by the number of words within the text.

Pseudocode:

Initialize text_frequency as a double that takes in a text pointer and a string called word
  If the word is not in the bloom filter
    Return 0
  Else
    Initialize a node with the value returned from an ht_look up of the hashtable and the word
    Return the nodes count divided by the texts word count

## text_contains

Functionality:

A function that determines if a word is within a text or not.

Pseudocode:

Initialize text_contains as a bool that takes in a Text pointer and a word
  If the bloom filter doesn't have the word
    Return false
  Initialize a node with the value of an ht_lookup of the hash table and word
  If the node isn't NULL and the node count isn't 0
    Return true
  Return false

# Identify.c

Functionality:

A function that takes in a file directory that holds a text and a library that holds other directories to other files. The text file will be used to compare the similarity between that file and the other text files within the library. The similarity will be calculated by filtering out specific words and calculating the Euclidean, Cosine, or Manhattan distance. The associated author of the text and the similarity that was calculated will then be printed.

Pseudocode:

## Usage

Initialize usage as a void and its parameters *exec as a char
  Call fprintf stderr and write about the function and how to use it and call exec at the end of the Message

## Main

Initialize main as an int that takes in the parameters of an int argc and a char **argv
  Initialize opt to the value of 0
  Initialize a string called database with the string lib.db
  Initialize a string called noise_input with the string noise.txt
  Initialize the stdin as a file type variable called fillin
  Preset the number of matches that printed to 5
  Preset the noise limit to 100
  Initialize a variable called mets with the value of 0
  Initialize the getopt while loop so that it can take in an h, d, n, k, l, e, m, and c as inputs
    In this case, h is called
      Return the usage page
    In the case, that d is entered
      Equate the database to the user input
    In the case, that n is called
      Equate the noise_input to whatever file the user wants to set as the noise
    In the case that k is pressed
      Equate match to the integer value of whatever they entered afterward
    In the case that l is pressed
      Equate the noiselimit to the integer value that the user inputted
    In the case, that e is pressed
      Equate mets to 0
    In the case, that m is pressed
      Equate mets to 1
    In the case that c is pressed
      Equate mets to 2

Set the default to send the user to the usage page
Initialize a uint32_t variable called n
Initialize a char pointer called author_name with the allocated space of 2048
Initialize a char pointer called path with the allocated space of 2048
Initialize a double called distance
Initialize a file pointer called noise_file and open the noise_input string to read
Initialize a file pointer called data and open the database string to read
Initialize a file pointer called pile
Create a Text pointer called noise with noise_file as the input you will be creating the noise Here
Create a Text pointer called text that will hold the file the user specified from stdin with fillin while filtering the words within the noise
Scan the file file called data and get the 1st integer without the new line and set it to the variable n this will be the amount of text and author pairs in the directory or library the user specified.
Create a priority queue with the value number of pairs there are
Iterate through each pair in the database
Get the authors names and the path of their specified text and remove the new lines of both the author name and the path
Equate pile to the open file of the specified text and read from it
If the file opened without any issues
Create a Text called tex that uses the noise to filter words out of the file names pile
Equate distance to the distance between tex and text using the specified distance formula
Enqueue the authors' names and distance into the priority queue
Print the number of matches wanted, the metric used, and the noise limit
Print the author's name, and their distance by dequeuing the values from the priority queue. This should also take into account the priority queue is smaller than the amount of matches wanted
Free and delete any Text type variables and free any other allocated space

## Credits


- I got how to implement the hashtable_insert and the hash able_lookup from the lecture slide Hash.
- Ben helped me structure my text_create.
- I got the idea of how to allocate the needed memory of the character pointer from ben and Eugene in the asgn 7 cse 13s discord.
- I got how to lower case words from the user Elmer in the cse13s discord.
- I got how to remove the newline that rails after calling fgets() from the user gecko10000#7137 in the asgn 7 cse13s discord.

- I got how to implement my bv.c from the bv.8 file in the code comments of the cse13s git lab.
- I as well used the code for the given structs such as the bloom filter, hashtable, text, etc.
- I go the how to use the regex from the asgn 7 documents that were provided.