RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Course Code	CSC202A
Course Name	Data Structure and Algorithms
Programme	B.Tech
Department	CSE
Faculty	FET

Name of the Student	Satyajit Ghana
Reg. No	17ETCS002159
Semester/Year	03/2018
Course Leader/s	Dr Pushphavathi T P

Declaration Sheet				
Student Name	Satyajit Ghana			
Reg. No	17ETCS002159			
Programme	B.Tech		Semester/Year	03/2018
Course Code	CSC202A			
Course Title	Data Structure and Algorithms			
Course Date		to		
Course Leader	Dr Pushphavathi T P			

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Contents

Question No. 1

Solution to Question No. 1 Part A:

A 1.1 Introduction to data structures:

Performing Natural Language processing requires a lot of data, with methods such as Data Mining to either train a ML model or to make sense of the data, the type of data structure chosen for a problem leads to a different algorithm and helps representing data is ways that can't be with other algorithms.

Data processing in a fast and efficient way is an important functionality in machine learning, especially with the growing interest in data storage. This exponential increment in data size has hampered traditional techniques for data analysis and data processing, giving rise to a new set of methodologies under the term Big Data. Many efficient algorithms for machine learning have been proposed, facing up time and main memory requirements.

Nevertheless, this process could still become hard when the number of features or records is extremely high. The goal is not to propose new efficient algorithms but a new data structure that could be used by a variety of existing algorithms without modifying their original schemata. Moreover, the proposed data structure enables sparse datasets to be massively reduced, efficiently processing the data input into a new data structure output. The results demonstrate that the proposed data structure is highly promising, reducing the amount of storage and improving query performance. (**Francisco Padillo, 2016**)

A 1.2 Efficient information retrieval techniques:

Many Natural Language Processing (NLP) techniques, including stemming, part of speech tagging, compound recognition, de-compounding, chunking, word sense disambiguation and others, have been used in Information Retrieval (IR). The core IR task we are investigating here is document retrieval. Several other IR tasks use very similar techniques, e.g. document clustering, filtering, new event detection, and link detection, and they can be combined with NLP in a way similar to document retrieval. Today, the amount of documents published on the internet grows dramatically, especially in education and e-learning activity and if the same information retrieval methods is used, then the time to find relevant document will also increase the same way. One possible solution is to develop software framework, which would improve the quality of current search engines and decrease time dedicated to searching process. (**Supreethi, K.P., Prasad, E.V., 2007**).

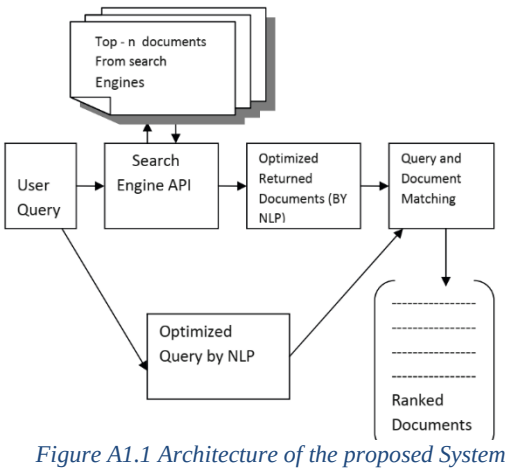

Figure A1.1 Architecture of the proposed System

Figure A1.1 shows the architecture of the proposed system. The uniqueness of our proposed method to the existing method is that it calculates the similarity measure for only the nouns and verbs of the query and documents optimized by NLP by using sharp NLP.

A 1.3 A real life example of data structure for massive data storage:

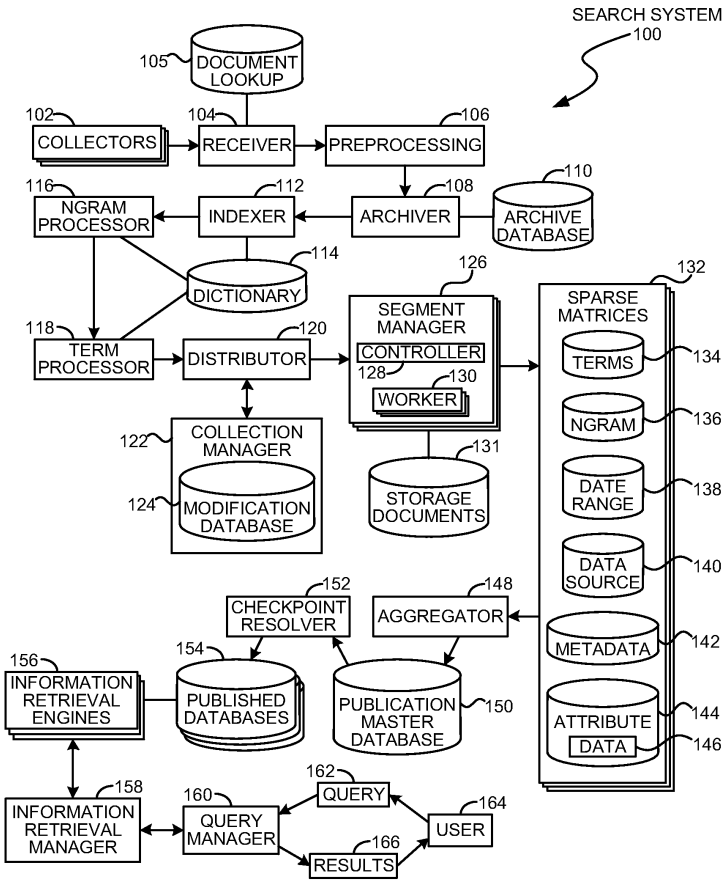

Figure A1.2 Google's Data Structure

Let's take the example of Google, which probably has the biggest Database of searches, The Search engine database used by Google is patented, and is a very complex set of representation of data, though is very efficient. It uses a combination of Hash Tables, Dictionaries, Indexed Documents, ngrams and Databases.

The storage document may be constructed by identifying and replacing text, tags, and other elements with term identifiers, ngrams, and other elements. In some embodiments, a copy of the original document may be stored for retrieval in a cached mode or when the original document may not be easily or immediately accessible. The storage document may be stored by a segment manager and used to compare any changes to an updated document or used to rebuild a portion of a sparse matrix.

In many embodiments, the matrix files may be arranged in a hierarchical fashion and managed by several different computers or processes. In a hierarchical arrangement, several smaller matrices may be created, merged together to form larger matrices. The larger matrices may be further merged together to form even larger matrices, and so forth. In embodiments with very large sets of data, such a hierarchical arrangement may allow the search system to scale to very large sizes. (**Patent US8375021B2**)

Question No. 2

Solution to Question No. 1 Part B:

B 1.1 Introduction to substitution ciphers:

Substitution Ciphers is a very simple and primitive method for data encryption, where the plaintext is replaces with the ciphered text and can be decrypted by a specific decryption algorithm or a key, which only the receiver must know. The message to be encrypted is split into units like words, or pair of words or even letters, the receiver then performs inverse substitution to get back those units in order, when placed together form the original plain text.

Different type of substitution ciphers:

- Simple substitution cipher
- Homophonic substitution
- Polyalphabetic substitution
- Polygraphic substitution
- Mechanical substitution ciper
- The one-time pad

B 1.2 Algorithm:

encrypt_algorithm:

```
Step 1: Start
Step 2: if mydata -> encrypted is true return -1
Step 3: if mydata is NULL return -1
Step 4: key <- malloc(length of mydata -> message * size of key)
Step 5: encrypted <- malloc(length of mydata -> message * size of key)
Step 6: declare i
Step 7: srand(time(0)) //seed rand with current system time
Step 8: while i < length of mydata -> message loop through Step 9 to Step 11
Step 9: key[i] = 65 + rand()%(CHAR_MAX - 65)
Step 10: encrypted[i] = mydata -> message[i] XOR key[i]
Step 11: i <- i + 1
Step 12: free(mydata -> message)
Step 13: mydata -> message = encrypted
Step 14: mydata -> is_encrypted = true
Step 15: Stop
```

decrypt_algorithm:

```
Step 1: Start
```

```
Step 2: if mydata -> encrypted is true return -1
Step 3: if mydata -> message is NULL return -1
Step 4: decrypted <- malloc(length of mydata -> message * size of decrypted)
Step 5: declare i
Step 6: while i < length of key loop through Step 7 to Step 8
Step 7: decrypted[i] = mydata -> message[i] XOR key[i]
Steo 8: i <- i + 1;
Step 9: free(mydata -> message)
Step 10: mydata -> message = decrypted
Step 11: mydata -> encrypted = false
Step 12: Stop
```

B 1.3 Justification of data structure used in the solution:

The Data Structure used in this problem is a Structure with Homogeneous Elements that contains a

- **bool isEncrypted** -> to know if the data is encrypted or not
- **char* message** -> the message that is either encrypted or decrypted
- **encrypt** -> a function to encrypt the decrypted data
- **decrypt** -> a function to decrypt the encrypted data

The Advantage of using this structure is that the data is more organized, and easily accessible for use, hence making the development much easier, the methods/function encrypt and decrypt provide an interface like structure or ADT for the model, which makes it easy to work with the data.

```c
#ifndef VERNAM_H
#define VERNAM_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
/* Read from the file, put it in message and set the bool flag, then do the necessary
 * operation in the ADT, simple */
struct SecureData {
    bool isEncrypted;
    char* message;
    int (*encrypt)(struct SecureData*, char**);
    int (*decrypt)(struct SecureData*, char*);
};
typedef struct SecureData SecureData;
SecureData* newSecureData(int (*encryptAlgo)(SecureData*, char**),
        int (*decryptAlgo)(SecureData*, char*));
int encrypt_vernam(SecureData*, char**);
int decrypt_vernam(SecureData*, char*);
#endif
```

B 1.4 C Program:

vernam.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>
#include <time.h>
#include "vernam.h"
#include "debug_helper.h"

SecureData* newSecureData(int (*encryptAlgo)(SecureData*, char**),
        int (*decryptAlgo)(SecureData*, char*)) {
    SecureData* myData = (SecureData*) malloc(sizeof *myData);
    if (myData == NULL) return NULL;
    myData -> isEncrypted = false;
    myData -> message = NULL;
    myData -> encrypt = encryptAlgo;
    myData -> decrypt = decryptAlgo;
    return myData;
}

/* pass a &char* to this method, and it encrypts the data and stores the key in the
 * parameter passed */
int encrypt_vernam(SecureData* myData, char** key) {
    if (myData -> isEncrypted)
        return -1;
    if (myData -> message == NULL)
        return -1;
    /* Encrypt the Data and store the key */

    *key = malloc(strlen(myData -> message) * sizeof **key);
    char *encrypted = malloc(strlen(myData -> message) * sizeof *encrypted);

    int i; srand(time(0));
    for (i = 0 ; i < strlen(myData -> message); i++) {
        (*key)[i] = 65 + rand()%(CHAR_MAX-65);
        encrypted[i] = (myData -> message)[i] ^ (*key)[i];
    }
    encrypted[i] = (*key)[i] = '\0';
    free(myData -> message);
    myData -> message = encrypted;
    myData -> isEncrypted = true;
    return 0;
}

int decrypt_vernam(SecureData* myData, char* key) {
    if (!myData -> isEncrypted)
        return -1;
    if (myData -> message == NULL)
        return -1;

    /* Decrypt the data using the Key */
    char* decrypted = malloc(strlen(myData -> message) * sizeof *decrypted);
    int i;
    for (i = 0 ; i < strlen(key) ; i++)d
        decrypted[i] = (myData -> message)[i] ^ key[i];
    decrypted[i] = '\0';
    free(myData -> message);
    myData -> message = decrypted;
    myData -> isEncrypted = false;
```

```
    return 0;
}
```

main.c

```
#include <stdio.h>
#include "vernam.h"
#include "debug_helper.h"
#include "file_reader.h"
#include "vector.h"

enum {
    ENCRYPT,
    DECRYPT
};

int OPERATION;

char *input_file;
char *input_key;

SecureData* myData;

void encrypt(char*);
void decrypt(char*, char*);

char* program_name;

int main(int argc, char** argv) {
    int c;
    program_name = *argv;
    while (--argc > 0 && (*++argv)[0] == '-')
        while (c = *++argv[0])
            switch(c) {
                case 'e':
                    OPERATION = ENCRYPT;
                    break;
                case 'd':
                    OPERATION = DECRYPT;
                    break;
            }

    if ((OPERATION == ENCRYPT && argc < 1) || (OPERATION == DECRYPT && argc < 2)) {
        printf("USAGE : %s -e <file to encrypt> \n"
               "%s -d <file to decrypt> <file key>\n", program_name, program_name);
        return 0;
    }
    /* Initialize my SecureData */
    myData = newSecureData(encrypt_vernam, decrypt_vernam);

    printf("\nVERNAM XOR ENCRYPTION - A Highly Secure One-Time Pad ----------------------
------\n");

    switch (OPERATION) {
        case ENCRYPT:
            encrypt(*argv);
            printf("\n%s is now Encrypted in %s_encrypted, Key stored in %s_key\n\n",
*argv, *argv, *argv);
            break;
        case DECRYPT: {
                char* fileName = *argv;
                char* fileKey = *++argv;
                decrypt(fileName, fileKey);
```

```
                printf("\n%s is now decrypted into file: message\n\n", fileName);
            }
            break;
    }
    return 0;
}

void encrypt(char* file_name) {
    char* to_encrypt = read_string_file(file_name);
    ds(to_encrypt);

    myData -> message = to_encrypt;
    myData -> isEncrypted = false;
    char* key;
    myData -> encrypt(myData, &key);
    char* output_file = strcat(new_string(file_name), "_encrypted");
    char* output_key = strcat(new_string(file_name), "_key");
    /* now write the key and the encrypted data */
    write_string_file(output_file, myData -> message);
    write_string_file(output_key, key);
}

void decrypt(char* file_name, char* key) {
    char* to_decrypt = read_string_file(file_name);
    char* decryption_key = read_string_file(key);
    myData -> message = to_decrypt;
    myData -> isEncrypted = true;
    myData -> decrypt(myData, decryption_key);

    ds(myData -> message);

    char* output_file = "message_decrypted";
    write_string_file(output_file, myData -> message);
}
```

OUTPUT:

```
./vernanXOR -e message

VERNAM XOR ENCRYPTION - A Highly Secure One-Time Pad ---------------------------
-

DEBUG--*to_encrypt : The domestric is a small, typically furry, carnivorous
mammal.
*

message is now Encrypted in message_encrypted, Key stored in message_key
```

Explanation:

Here the data to be encrypted is stored in a file named message and passed as an argument to the program, the program then takes the string stored in the file and performs vernam XOR for each and every character of the input, by generating a random character and then XORing this with the input, the random character generated is stores in the key, the Boolean value isEncrypted is not set to true since now the message is encrypted and then stores the key in message_key and the encrypted message is stored in message_encrypted. The relevant message is displayed.

./vernanXOR –d message_encrypted message_key

```
VERNAM XOR ENCRYPTION - A Highly Secure One-Time Pad ---------------------------
-

DEBUG--*myData -> message : The domestric is a small, typically furry,
carnivorous mammal.
*

message_encrypted is now decrypted into file: message
```

Explanation:

Here the data to be encrypted is passed as a command line argument to the program along with the key file, it then reads both of these files and performs XOR operation of each and every character of the respective character of the input file to be decrypted and the key. The output is store in the Data Structure SecureData and the Boolean isEncrypted is set to false. The decrypted file is then saved into message_decrypted.

Solution to Question No. 2 Part B:

B 2.1 Plagiarism rule and threshold:

The rules and threshold of plagiarism in my world are as follows:

1. If there are some important words in a document that are found to be copied in the same manner then the document is plagiarized.
2. The threshold can be defined by the user, by default the program will calculate how much of the content in the document is plagiarized, or could be plagiarized.
3. If the document is one-to-one copied then the program will output 100% plagiarized.
4. If the document contains parts of the original document then the percentage that is copied is shown in the output along with the copied content.

B 2.2 Pseudocode for checking plagiarized content:

```
bag_words:
   1. Begin
   2. Create a HashMap mymap
   3. Read the data from the file and store it in char* data
   4. Declare variable tok and initilialize it with strtok(data, ".\n\t ")
   5. While (tok ≠ 0)
      5.1   Convert every character of tok to lower alphabet, remove the trailing
            and leading spaces and store it in to_add
      5.2   If (to_add≠ '\n' && to_add ≠ '\0')
         5.2.1 If (mymap contains to_add) then increment the value of key to_add
               in mymap
         5.2.2 Else add to_add to hashmap and assign 1.0 as the value to the key
               to_add
      5.3   tok = strtok(0, ".\n\t ")
   6. return mymap
   7. End
normalize_words:
   1. Begin
   2. Declare total_words <- 0
   3. for (j = 0; j < length of bag ; j++)
      3.1          total_words = total_words + bag -> data[j].value
   4. for (j=0; j < length of bag ; j++)
      4.1   set the value for key bag -> data[j].key as current value for the key
            / total_words
   5. End
compute_tf_idf:
```

1. Begin
2. for (i = 0 ; i < length of bag ; i++)
   1.1 curr_word = bag - > data[i].key
   1.2 tf = bag -> data[i].value
   1.3 number_of_documents_with_word = 0
   1.4 for (j = 0 ; j < length of bags ; j++)
      1.4.1 curr_bag = bags -> data[j]
      1.4.2 if (curr_bag contains curr_word) then number_of_documents_with_word++
   1.5 idf = 1 + log (length of bags / number_of_documents_with_word)
   1.6 set the bag with key curr_word as tf * idf
3. End

compute_cosine_similarity:

1. Begin
2. modQuery = 0
3. for (i = 0; i < length of query ; i++)
   3.1 modQuery = modQuery + (query -> data[i].value)^2
4. modQuery = $\sqrt{modQuery}$
5. modDocument = 0
6. for (i = 0; i < length of document ; i++)
   6.1 modDocument = modDocument + (document -> data[i].value)^2
7. modDocument = $\sqrt{modDocument}$
8. querytimesdocument = 0
9. for (i = 0; i < length of document ; i++)
   9.1 if (document)
10. End

compare_query_with_documents:

1. Begin
2. Declare Vecor bags, HashMap query_bag
3. Read all the document files and use bag_words to bag the words and store it in a HashMap and add it to the Vector bags
4. Bag the Query using bag_words and store it in query_bag
5. Normalize the bags using normalize_bags
6. Normalize the Query using normalize_bags
7. Compute TF*IDF using compute_tf_idf for bags
8. Compute TF*IDF using compute_tf_idf fro query
9. End

B 2.3 C Program:

`main.c`

```c
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <dirent.h>
#include <math.h>
#include "hash_map.h"
#include "pcheck.h"
#include "debug_helper.h"
#include "file_reader.h"
#include "input_helper.h"
#include "vector.h"

int main(int argc, char** argv) {

    char* file_name = argv[0];
    char* directory = "./data";
    char* query_file;
    float threshold = 80.0f;;

    switch(argc) {
        case 2:
            query_file = argv[1];
            break;
        case 3:
            query_file = argv[1];
            threshold = atoi(argv[2]);
            break;
        case 4:
            directory = argv[1];
            query_file = argv[2];
            threshold = atoi(argv[3]);
            break;
        default:
            printf( "\nUSAGE: %s <directory> <file> <threshold>"
                    "\ndefault directory = ./data"
                    "\ndefault threshold = 80.0%%"
                    "\nExample: %s data query 92.2\n", file_name, file_name);
            break;
    }

    if (argc <= 1 || argc > 4)
        return -1;

    struct dirent* in_file;
    FILE* entry_file;
    DIR *FD;

    if ((FD = opendir(directory)) == NULL) {
        printf("\nError Opening %s Directory\n", directory);
        return -1;
    }

    Vector* file_list = newMinimalVector();

    while (in_file = readdir(FD)) {
        if (!strcmp (in_file->d_name, "."))
            continue;
        if (!strcmp (in_file->d_name, ".."))
            continue;
```

```
        //ds(in_file -> d_name);
        char* path = new_string(directory);
        strcat(path, "/");
        strcat(path, in_file -> d_name);
        file_list -> add(file_list, path);
    }

    //return 0;

    Vector* bags;
    HashMap* query_bag;

    bags = newMinimalVector();
    query_bag = newHashMap();

    /* Bag the Worlds */
    for (int i = 0 ; i < file_list -> length ; i++)
        bags -> add(bags, bag_words(file_list -> data[i]));

    /* bag the query */
    query_bag = bag_words(query_file);

    /* Normalize the Bags */
    for (int i = 0 ; i < bags -> length ; i++)
        normalize_words(bags -> data[i]);

    /* Normalize the Query */
    normalize_words(query_bag);

    /* Compute tf*idf */
    /* Make sure that you have already computed TF and also normalized it */
    for (int i = 0 ; i < bags -> length ; i++)
        compute_tf_idf(bags -> data[i], bags);

    compute_tf_idf(query_bag, bags);

    printf("\nTF-IDF COSINE SIMILARITY PLAGIARIZATION CHECKER ------------------\n\n");

    /* find the cosine similarity */
    int count = 0;
    printf("\nQUERY ----------------------------------------: \n");
    printf("%s\n\n", read_string_file(query_file));
    for (int i = 0 ; i < bags -> length ; i++) {
        double similarity = compute_cosine_similarity(query_bag, bags -> data[i]);
        if (similarity * 100 > threshold) {
            count++;
            printf("\nFOUND PLAGIARIZATION with %.10f%% Cosine similarity\n",
similarity*100);
            printf("\nORIGINAL FILE ----------------------------------------: \n");
            printf("%s\n",read_string_file(file_list -> data [i]));
        }
    }

    printf("\nSearched %d documents, found %d matches with minimum threshold of
%.2f%%\n\n",
        bags -> length, count, threshold);

    return 0;
}
```

pcheck.c

```c
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include "hash_map.h"
#include "pcheck.h"
#include "debug_helper.h"
#include "file_reader.h"
#include "input_helper.h"
#include "vector.h"

const char *delim = ".\n\t ";

HashMap *bag_words(char *file) {
    HashMap *mymap = newHashMap();
    char *data = read_string_file(file);
    char *tok;
    tok = strtok(data, delim);
    while (tok != 0) {
        char *to_add = strlwr(new_string(sanitize_string(tok)));
        if (*to_add != '\n' && *to_add != '\0' && strlen(to_add) > 0) {
            if (mymap->contains(mymap, to_add))
                mymap->set(mymap, to_add, new_double((*(double *)mymap->get(mymap,
to_add)) + 1.0));
            else
                mymap->add(mymap, to_add, new_double(1.0));
        }
        tok = strtok(0, delim);
    }
    return mymap;
}

/* found normalized TF */
void normalize_words(HashMap *bag) {
    double total_words = 0;
    for (int j = 0; j < bag->length; j++)
        total_words += *(double *)bag->data[j].value;
    for (int j = 0; j < bag->length; j++)
        bag->set(bag, bag->data[j].key, new_double((*(double *)bag->data[j].value) /
total_words));
}

/* TF*IDF
** IDF = 1 + log(Total number of documents / Number of documents with that word)*/
void compute_tf_idf(HashMap *bag, Vector *bags) {
    for (int i = 0; i < bag->length; i++) {
        char *curr_word = bag->data[i].key;
        double tf = *(double *)bag->data[i].value;
        int number_of_documents_with_word = 0;
        for (int j = 0; j < bags->length; j++) {
            HashMap *curr_bag = (HashMap *)(bags->data[j]);
            if (curr_bag->contains(curr_bag, curr_word))
                number_of_documents_with_word++;
        }
        double idf = 1.0 + log(((double)bags->length) / number_of_documents_with_word);
        bag->set(bag, curr_word, new_double(tf * idf));
    }
}

/* cos(theta) = A.B / (||A||*||B||)
** here ||A|| = (TF*IDF for word 1)^2 + (TF*IDF for word 2)^2 and so on,
** same is applied for ||B||*/
```

```c
double compute_cosine_similarity(HashMap *query, HashMap *document) {

    /* computer ||query|| */
    double modQuery = 0.0;
    for (int i = 0; i < query->length; i++)
        modQuery += pow(*(double *)(query->data[i].value), 2);
    modQuery = sqrt(modQuery);
    //dd(modQuery);

    /* compute ||document||, only take the words that are present in the query */
    double modDocument = 0.0;
    for (int i = 0; i < document->length; i++)
        if (query->contains(query, document->data[i].key))
            modDocument += pow(*(double *)(document->data[i].value), 2);
    modDocument = sqrt(modDocument);
    //dd(modDocument);

    /* compute query.document = q1*d1+q1*d2 . . . .qn*dn */
    double querytimesdocument = 0.0;
    for (int i = 0; i < query->length; i++)
        if (document->contains(document, query->data[i].key))
            querytimesdocument += *(double *)(query->data[i].value) * *(double
*)(document->get(document, query->data[i].key));
    //dd(querytimesdocument);

    double cosine_similarity = 0.0;
    cosine_similarity = querytimesdocument / (modQuery * modDocument);
    //dd(cosine_similarity);

    return cosine_similarity;
}
```

OUTPUT:

```
./pcheck data query 90

TF-IDF COSINE SIMILARITY PLAGIARIZATION CHECKER ------------------


QUERY ------------------------------------------:
The domestic cat (Felis silvestris catus or Felis catus) is a small,
typically furry, carnivorous mammal. They are often called house cats
when kept as indoor pets or simply cats when there is no need to
distinguish them from other felids and felines.


FOUND PLAGIARIZATION with 93.2975787527% Cosine similarity

ORIGINAL FILE ----------------------------------------:
The domestic cat (Felis silvestris catus or Felis catus) is a small,
typically furry, carnivorous mammal. They are often called house cats
when kept as indoor pets or simply cats when there is no need to
distinguish them from other felids and felines. They are often valued by
humans for companionship and for their ability to hunt vermin. There are
more than seventy cat breeds recognized by various cat registries.

Searched 4 documents, found 1 matches with minimum threshold of 90.00%
```

Explanation:

All the Source Documents are stored in a folder named data, which are text files, the query is also stored in a file named query that contains the potentially plagiarized document. These files are read, they are sanitized, i.e. the leading spaces, and trailing spaces are removes, they are split into words, bagged, and then each of the documents are converted to n-dimensional vector, the query is also converted to n-dimensional vector, these vectors are then normalized, and the TF*IDF is calculated for each of them, the advantage of using TF*IDF is that common words across the documents are weighted down, i.e. they are assigned a lower score, and the unique words are weighted more, using logarithmic, now that we have a vector for each of the document and also the query, it can simply be compared using Dot Product, which is

$$A.B = \big||A|\big| \times \big||B|\big| \times \cos\theta$$

$$\cos\theta = \frac{A.B}{\big||A|\big| \times ||B||}$$

$\cos\theta$ will output a value from $[-1, 1]$, but for our implementation it won't output a value less than 0, hence the output of Cosine Similarity will be from $[0, 1]$, this can be mapped from $[0, 100]$ to get a percentage of potential plagiarisation.

It then checks which documents have a minimum plagiarisation which was passed as a command line argument to the program, and those plagiarized documents are displayed along with the query.

B 2.4 Conclusion and future scope:

The current implementation of the Cosine Similarity for checking plagiarism converts the entire document into a Vector, this can be improved further by converting each paragraph into individual vectors and the same to be done for the query, and then each vector of query can be compared to each of the vector of each of the document, this will yield better and more accurate results, though the time complexity of the program will increase more. The words like 'is', 'the', 'a', 'was' are to be weighted less so there should be minimum number of documents to do the same.

The Future Scope would be make a API of the same, such that it can be fed with documents and at input file, and it would search data from different sources, such as Google, Yahoo Answers, Quora, DuckDuckGo, and several other Search Engines / Social Sites / Question-Answer Site, and compare with them and display the sources from which it was plagiarized.

Another improvement that needs to be done is the currently it only accepts ASCII files, it should be able to accept PDF files or even DOCX files and sanitize them into plain text files, then the usual Algorithm can be used to check for potential plagiarism and the relevant matching documents can be displayed.

1. Francis Padillo (2016) A Data Structure to Speed-Up Machine Learning Algorithms on Massive Datasets.
2. Supreethi, K.P., Prasad, E.V.: Web Document Clustering Technique Using Case Grammar Structure. In: IEEE International Conference on Computational Intelligence and Multimedia Applications 2007 (2007).
3. US Patent US8375021B2, https://patents.google.com/patent/US8375021B2/en.

For Complete Source Code refer to:

4. Satyajit Ghana (2018), https://github.com/satyajitghana/University-Work