DAA Experiment-9
(Batch-A/A1)

Name	Ansari Mohammed Shanouf Valijan
UID Number	2021300004
Class	SY B.Tech Computer Engineering(Div-A)
Experiment Number	9
Date of Performance	20-04-23
Date of Submission	23-04-23

Aim:
To implement the following String-Matching Algorithms(SMA)-
- ✓ Naïve SMA.
- ✓ Rabin-Karp SMA.

Theory:
String-matching algorithms are a group of algorithms designed to find a specific pattern within a larger string or text. The pattern may be a single word, a phrase, or a sequence of characters. These algorithms are widely used in computer science and information retrieval systems. In this experiment, I have explored the Naïve and Rabin-Karp string-matching algorithms.

[A] Naïve SMA-

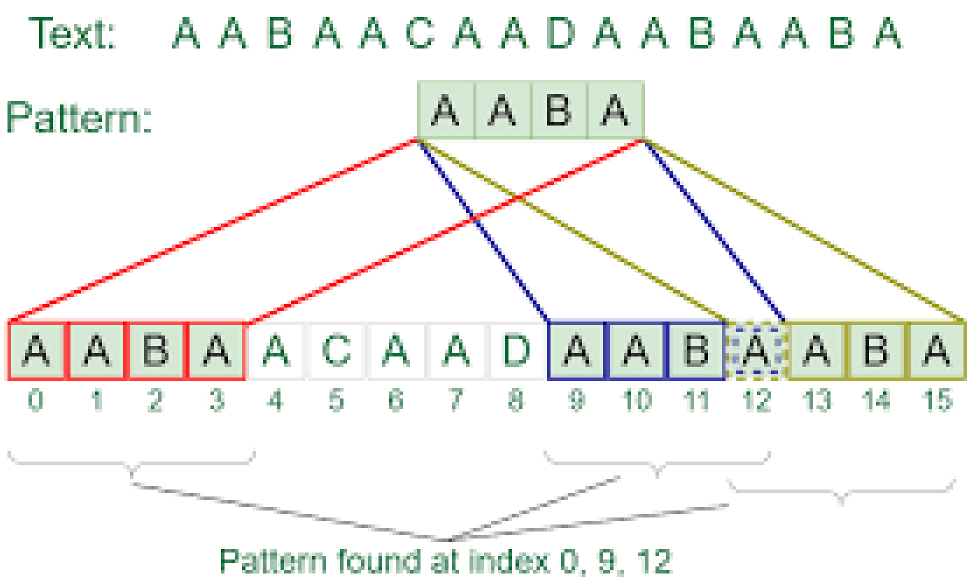
The Naive String-Matching Algorithm is a simple algorithm used to find occurrences of a pattern within a larger text or string. This algorithm works by comparing the pattern character-by-character to each position in the text, sliding the pattern by one character if there is no match until either a match is found or the end of the text is reached.

The algorithm is called "naïve" because it does not use any pre-processing or advanced techniques to speed up the search. It has a time complexity of O(mn), where m is the length of the pattern and n is the length of the text, which can make it inefficient for large texts or patterns.

[B] Rabin-Karp SMA-

The Rabin-Karp string matching algorithm is a string searching algorithm that uses hashing to find occurrences of a pattern within a larger text or string. The algorithm was developed by Michael O. Rabin and Richard M. Karp in 1987 and has a time complexity of O(n+m), where n is the length of the text and m is the length of the pattern.

The algorithm works by computing a hash value for the pattern and then sliding the pattern over the text, computing a hash value for each substring of the text that is the same length as the pattern. If the hash values match, the algorithm compares the actual characters of the pattern and the substring to verify the match.


Text: A A B A A C A A D A A B A A B A
Pattern:
A A B A
A A B A A C A A D A A B A A B A
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Pattern found at index 0, 9, 12

The image above depicts a schematic of how a string can be searched for a particular pattern. In different string-matching algorithms, its just the interpretation of these searching windows that is different.

Algorithm:

[A] For Naïve SMA-

I. Start.
II. From the beginning of string, select a window of characters with length same as the pattern's length.
III. If selected window is equal to the pattern to be searched, print the index of window's start.
IV. If not equal, slide the window by 1 and repeat till the end of string.
V. End.

[B] For Rabin-Karp SMA-

I. Start.

II. Convert the given strings into numerical strings by considering certain numeric values for corresponding characters.
III. Select a prime number, say p, to be considered in hash function calculations.
IV. Calculate the remainder when pattern is divided by p and store it in a variable, say q.
V. From the beginning of string, select a window with length same as the pattern's length.
VI. If selected window gives the same remainder with same hash function, check if two strings are equal.
VII. If equal, print the index.
VIII. If selected window does not give the required remainder, slide by 1 and repeat the process.
IX. End.

Program:

[A] For Naïve SMA-

```
//header files
#include<stdio.h>
#include<string.h>
#include<stdbool.h>

//global variables
char string[100];
char pattern[100];
int count=0; //to know how many occurences were found

//function to compare two strings
bool isMatching(int offset){
    for(int i=0; i<strlen(pattern); i++){
        if(string[i+offset]!=pattern[i])
            return false;
    }
    return true;
}

//function to search for the pattern
void naiveSearch(){
    for(int i=0; i<strlen(string)-strlen(pattern)+1; i++){
        if(isMatching(i)){
            if(count==0)
                printf("\nFollowing information about the index(es) were obtained-\n");
            printf("\nPattern found at index %d",i);
            count++;
        }
    }
}

//main function
```

```c
void main(){
    //taking user inputs
    printf("\nEnter a string -----> ");
    scanf("%[^\n]%*c",string);
    printf("Enter the pattern you want to search for in the provided string -----> ");
    scanf("%[^\n]%*c",pattern);

    //searching for the pattern and printing the found indexes
    naiveSearch();
    if(count!=0){
        if(count==1)
            printf("\n(%d occurrence was obtained!!)",count);
        else
            printf("\n(%d occurrences were obtained!!)",count);
    }
    else
        printf("\nNo such occurence of the provided pattern was found in the string!!");
    printf("\n\n");
}
```

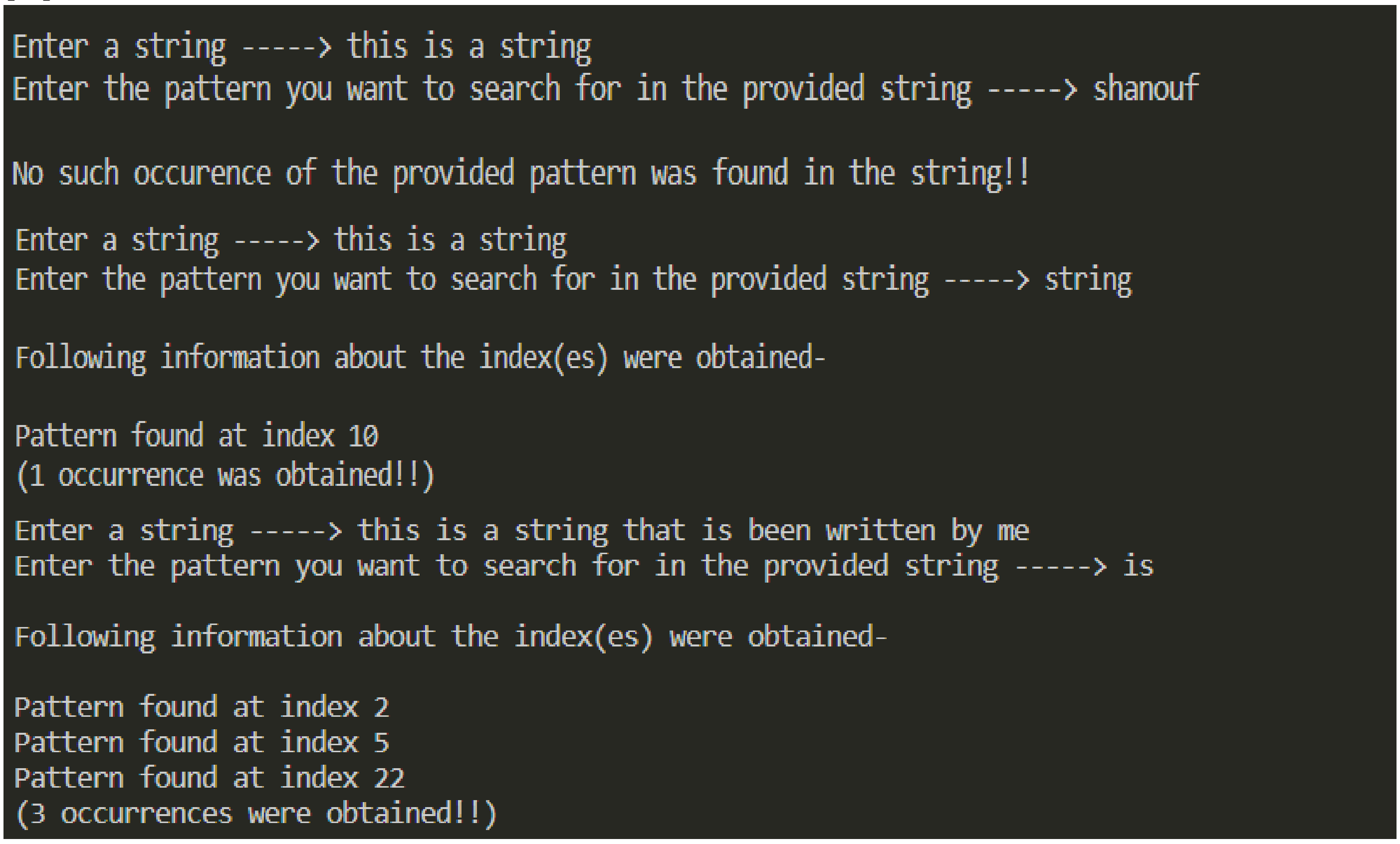
[B] For Rabin-Karp SMA-

```c
//header files
#include<stdio.h>
#include<string.h>
#include<stdbool.h>

//global variables
char string[100];
char pattern[100];
int modVal;
int count=0; //to know how many occurences were found
int spuriousHit=0, validHit=0;

//function to compare two strings
bool isMatching(int offset){
    for(int i=0; i<strlen(pattern); i++){
        if(string[i+offset]!=pattern[i])
            return false;
    }
    return true;
}

//function to translate character string to number string
void translate(char str[]){
    char temp[100];
    int tempCount=0, tempNum;
    for(int i=0; i<strlen(str); i++){
        if(str[i]==' '){
            temp[tempCount++]=(char)48;
            continue;
        }
        tempNum=(int)str[i]-48;
        if(tempNum<58)
```

```c
            temp[tempCount++]=(char)tempNum;
        else{
            tempNum-=48;
            temp[tempCount++]=(char)(tempNum/10+48);
            temp[tempCount++]=(char)(tempNum%10+48);
        }
    }
    temp[tempCount]='\0';
    strcpy(str,temp);
}

//function to obtain integar from num-string
int getInteger(char str[], int start, int end){
    int num=0, factor=1;
    for(int i=end; i>=start; i--){
        if(str[i]=='0'){
            factor*=10;
            continue;
        }
        num+=factor*((int)str[i]-48);
        factor*=10;
    }
    return num;
}

//function to search for the pattern
void rabinKarpSearch(){
    translate(string);
    translate(pattern);
    int temp;
    int valToCompare=(getInteger(pattern,0,strlen(pattern)-1))%modVal;
    for(int i=0; i<strlen(string)-strlen(pattern)+1; i++){
        temp=(getInteger(string,i,i+strlen(pattern)-1))%modVal;
        if(temp==valToCompare){
            if(isMatching(i)){
                if(count==0)
                    printf("\nFollowing information about the index(es) were obtained-
\n");
                printf("\nPattern found at index %d",i);
                count++;
                validHit++;
            }
            else
                spuriousHit++;
        }
    }
}

//main function
void main(){
    //taking user inputs
    printf("\nEnter a string -----> ");
    scanf("%[^\n]%*c",string);
    printf("Enter the pattern you want to search for in the provided string -----> ");
    scanf("%[^\n]%*c",pattern);
    printf("Enter a prime number that should be used in hash function -----> ");
    scanf("%d",&modVal);

    //searching for the pattern and printing the found indexes
    rabinKarpSearch();
    if(count!=0){
        if(count==1)
            printf("\n(%d occurrence was obtained!!)",count);
        else
            printf("\n(%d occurrences were obtained!!)",count);
    }
    else
        printf("\nNo such occurence of the provided pattern was found in the string!!");
    printf("\n\nTotal Spurious Hits-%d, Total Valid Hits-%d",spuriousHit,validHit);
    printf("\n\n");
}
```

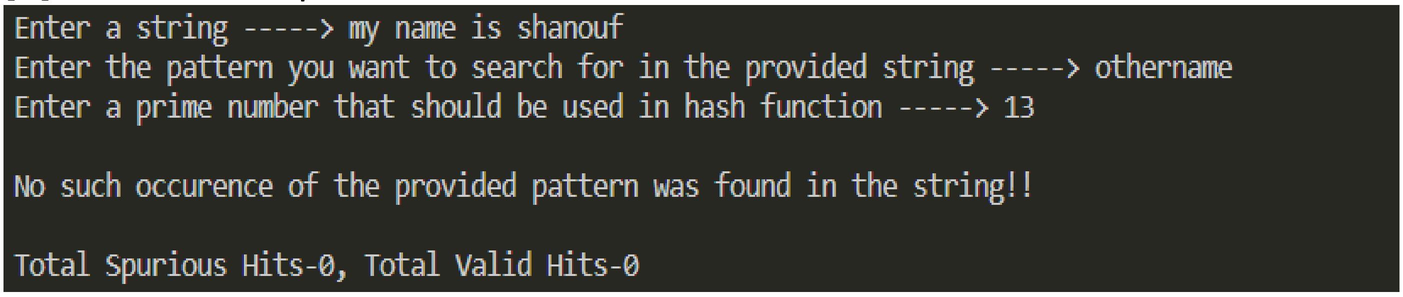
Implementation:

[A] For Naïve SMA-

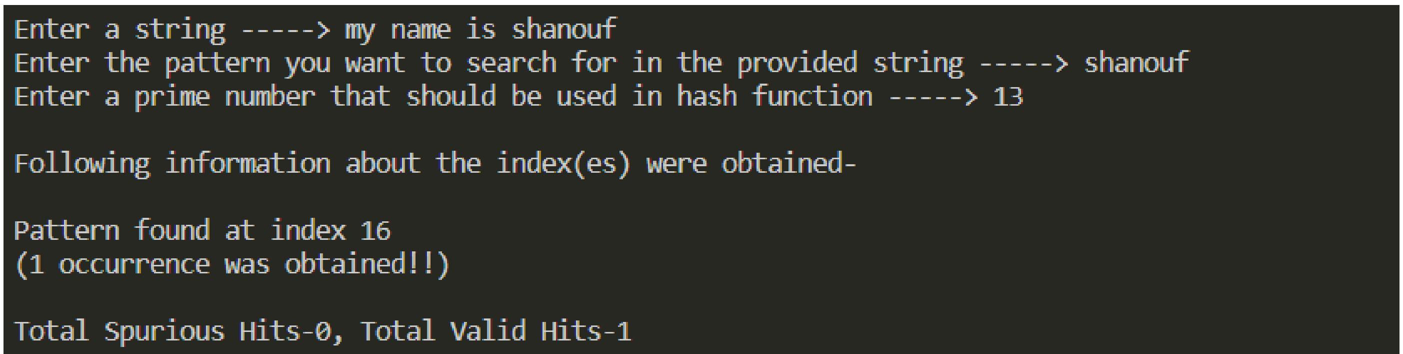


```
Enter a string -----> this is a string
Enter the pattern you want to search for in the provided string -----> shanouf

No such occurence of the provided pattern was found in the string!!

Enter a string -----> this is a string
Enter the pattern you want to search for in the provided string -----> string

Following information about the index(es) were obtained-

Pattern found at index 10
(1 occurrence was obtained!!)

Enter a string -----> this is a string that is been written by me
Enter the pattern you want to search for in the provided string -----> is

Following information about the index(es) were obtained-

Pattern found at index 2
Pattern found at index 5
Pattern found at index 22
(3 occurrences were obtained!!)
```

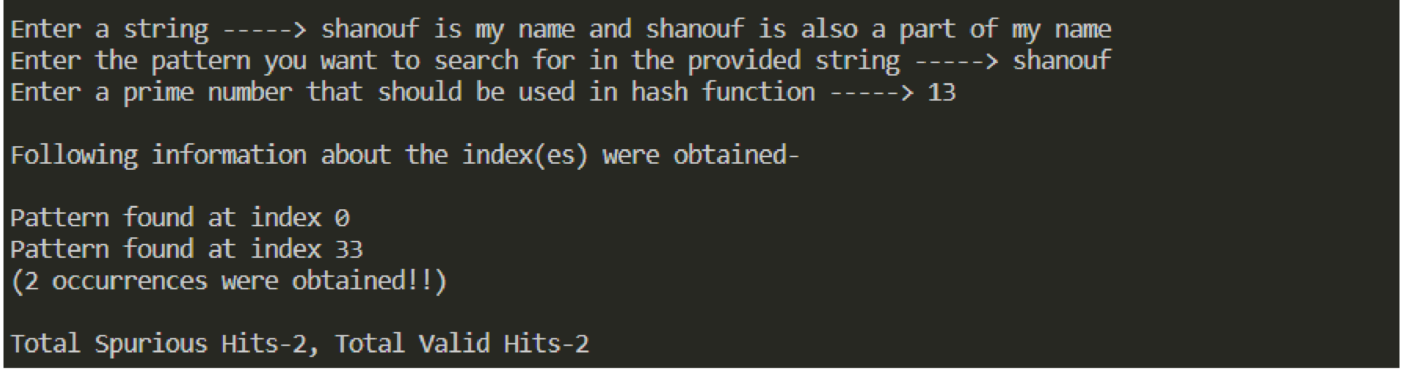
[B] For Rabin-Karp SMA-



```
Enter a string -----> my name is shanouf
Enter the pattern you want to search for in the provided string -----> othername
Enter a prime number that should be used in hash function -----> 13

No such occurence of the provided pattern was found in the string!!

Total Spurious Hits-0, Total Valid Hits-0
```



```
Enter a string -----> my name is shanouf
Enter the pattern you want to search for in the provided string -----> shanouf
Enter a prime number that should be used in hash function -----> 13

Following information about the index(es) were obtained-

Pattern found at index 16
(1 occurrence was obtained!!)

Total Spurious Hits-0, Total Valid Hits-1
```



```
Enter a string -----> shanouf is my name and shanouf is also a part of my name
Enter the pattern you want to search for in the provided string -----> shanouf
Enter a prime number that should be used in hash function -----> 13

Following information about the index(es) were obtained-

Pattern found at index 0
Pattern found at index 33
(2 occurrences were obtained!!)

Total Spurious Hits-2, Total Valid Hits-2
```

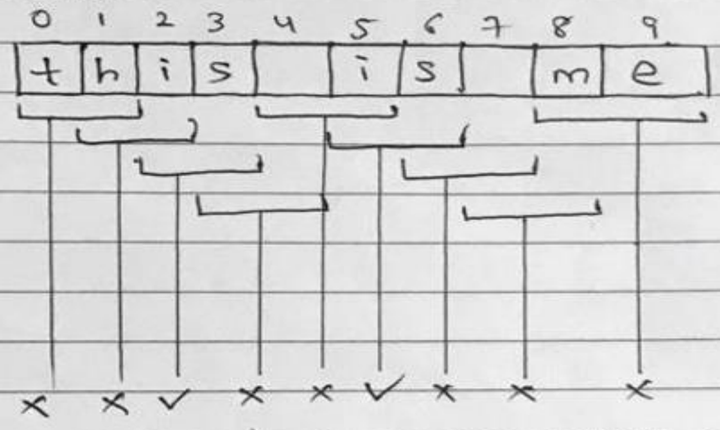
Following example was solved to understand the algorithms-

Naïve SMA Example:

string = "this is me"   Pattern = "is"

Thus, we have:

Iterations count = 10-2+1 = 9



Hence, we found 2 occurrences at indices 2 & 5

Rabin-Karp Example:

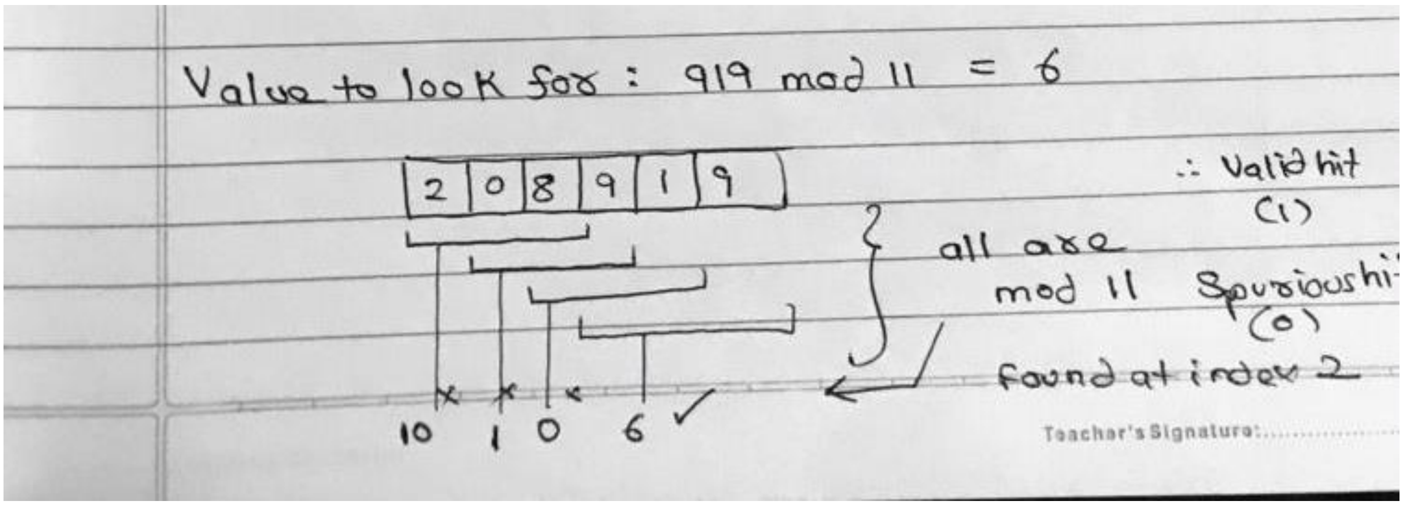
string = "this"   Pattern = "is"

Let encoding logic be: a=1, b=2, . . . z=26

prime number be 11

Thus, translated string = "208919"

translated pattern = "919"



Inference:

On writing the logic of both these algorithms, I came to the realization that there is a similar approach followed in them. However, Rabin-Karp algorithm is a two-step algorithm. In Naïve algorithm, we directly compare the pattern with all the windows of characters to check if a match is found. On the other hand, Rabin-Karp algorithm first checks if there is a possibility of match using hash function. If there is a possibility, then only it goes ahead checking the window character by character. This, thus, helps to save some time and hence, Rabin-Karp algorithm is more efficient.

Conclusion:

By performing this experiment, I was able to understand Naïve and Rabin-Karp string matching algorithms. I was also able to implement them and verify the results obtained.