NAME:	Deepanshu Aggarwal
UID:	2021300002
BRANCH:	Computer engineering
BATCH:	A
SUBJECT:	DAA
EXPT NO:	9
AIM:	To implement Rabin Karp and Naive String Matching Algorithm
THEORY:	The Naive String Matching algorithm slides the pattern one by one. After each slide, one by one checks characters at the current shift, and if all characters match then print the match. Like the Naive Algorithm, the Rabin-Karp algorithm also slides the pattern one by one. But unlike the Naive algorithm, the Rabin Karp algorithm matches the hash value of the pattern with the hash value of the current substring of text, and if the hash values match then only it starts matching individual characters. So Rabin Karp algorithm needs to calculate hash values for the following strings. Pattern itself All the substrings of the text of length m
PROGRAM: **(Rabin-Karp)**	(code below)

```
#include <stdio.h>
#include <string.h>
#define d 256

void search(char pat[], char txt[], int q)
{
    int M = strlen(pat);
    int N = strlen(txt);
    int i, j;
    int p = 0;
    int t = 0;
    int h = 1;

    for (i = 0; i < M - 1; i++)
        h = (h * d) % q;


    for (i = 0; i < M; i++) {
        p = (d * p + pat[i]) % q;
        t = (d * t + txt[i]) % q;
    }

    for (i = 0; i <= N - M; i++) {

        if (p == t) {

            for (j = 0; j < M; j++) {
                if (txt[i + j] != pat[j])
                    break;
            }

            if (j == M)
                printf("Pattern found at index
%d \n", i);
        }

        if (i < N - M) {
            t = (d * (t - txt[i] * h) + txt[i +
M]) % q;
            if (t < 0)
                t = (t + q);
        }
    }
}
```

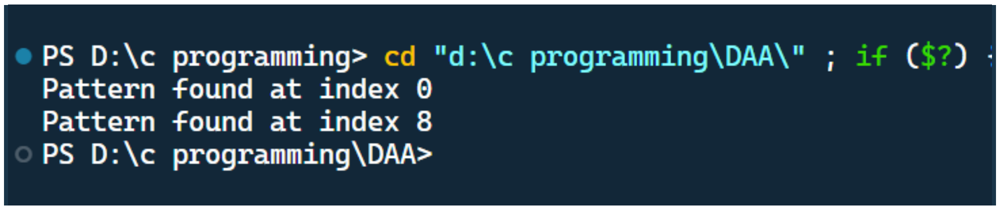
```c

/* Driver Code */
int main()
{
    char txt[] = "Deep Is Deep";
    char pat[] = "Deep";

    // A prime number
    int q = 101;

    // function call
    search(pat, txt, q);
    return 0;
}
```

RESULT:



```
PS D:\c programming> cd "d:\c programming\DAA\" ; if ($?)
Pattern found at index 0
Pattern found at index 8
PS D:\c programming\DAA>
```

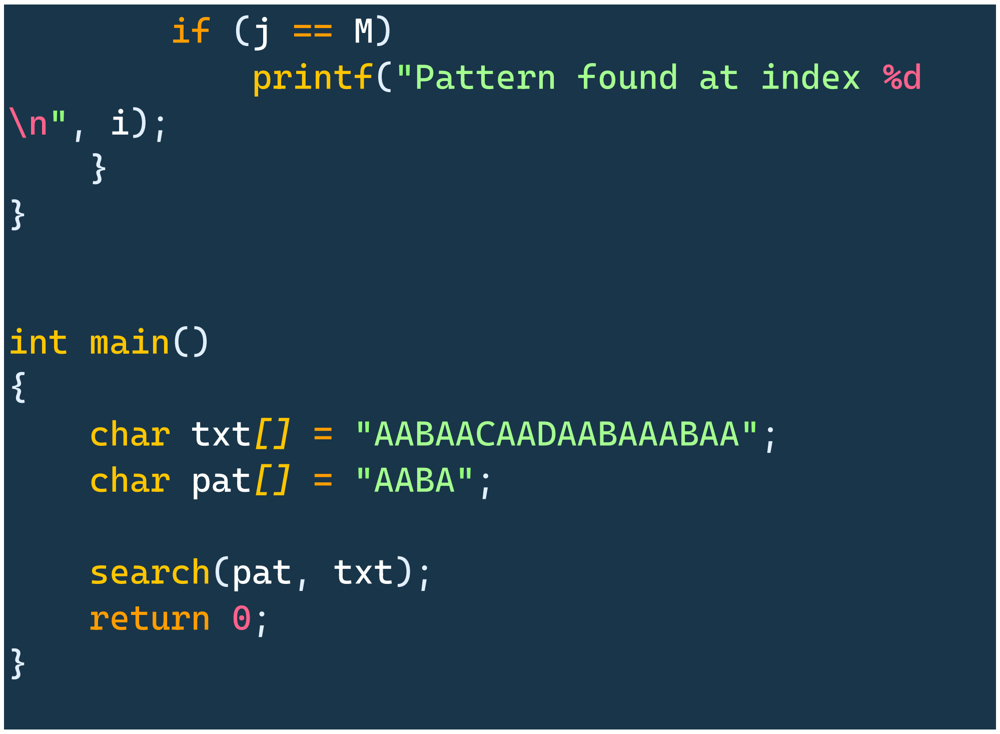
PROGRAM:
(Naive approach)

```c
#include <stdio.h>
#include <string.h>

void search(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    for (int i = 0; i <= N - M; i++) {
        int j;

        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;
```

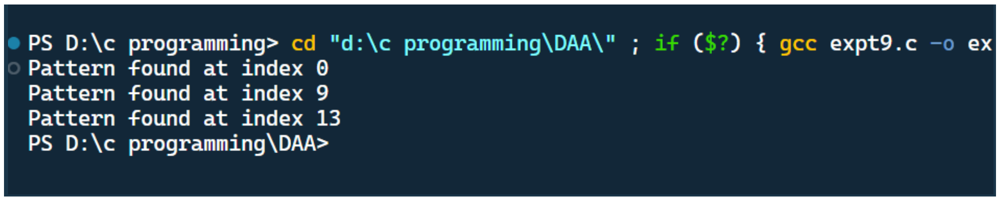


```
        if (j == M)
            printf("Pattern found at index %d
\n", i);
    }
}


int main()
{
    char txt[] = "AABAACAADAABAAABAA";
    char pat[] = "AABA";

    search(pat, txt);
    return 0;
}
```

RESULT:



```
PS D:\c programming> cd "d:\c programming\DAA\" ; if ($?) { gcc expt9.c -o ex
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13
PS D:\c programming\DAA>
```

CONCLUSION:	From this experiment, I learnt the rabin karp method for string matching which uses hash value to match the string and reduce the time complexity as compared to the naive method.