EXPERIMENT NO- 10

AIM: Program to implement Rabin-Karp Algorithm for Pattern Searching algorithm.

PROBLEM STATEMENT : Write a Program to implement Rabin-Karp Algorithm for Pattern Searching algorithm.

Resource Required: Pentium IV, Turbo C, Printer, Printout Stationary

THEORY: Rabin-Karp is another pattern searching algorithm to find the pattern in a more efficient way. It also checks the pattern by moving window one by one, but without checking all characters for all cases, it finds the hash value. When the hash value is matched, then only it tries to check each character. This procedure makes the algorithm more efficient.
The Rabin-Karp string matching algorithm calculates a hash value for the pattern, as well as for each M-character subsequences of text to be compared. If the hash values are unequal, the algorithm will determine the hash value for next M-character sequence. If the hash values are equal, the algorithm will analyze the pattern and the M-character sequence. In this way, there is only one comparison per text subsequence, and character matching is only required when the hash values match.

ALGORITHM:

RABIN-KARP-MATCHER (T, P, d, q)

```
1.      n ← length [T]

2.      m ← length [P]

3.      h ← dm-1 mod q 4. p ← 0
5. t0 ← 0

6.      for i ← 1 to m

7.      do p ← (dp + P[i]) mod q
8.      t0 ← (dt0+T [i]) mod q

9.      for s ← 0 to n-m

10.     do if p = ts

11. then if P [1.....m] = T [s+1.    s + m]

12.     then "Pattern occurs with shift" s

13.     If s < n-m

14.     then ts+1 ← (d (ts-T [s+1]h)+T [s+m+1])mod q
```

Input:

txt[] = "AABAACAADAABAABA" pat[] = "AABA"

Output:

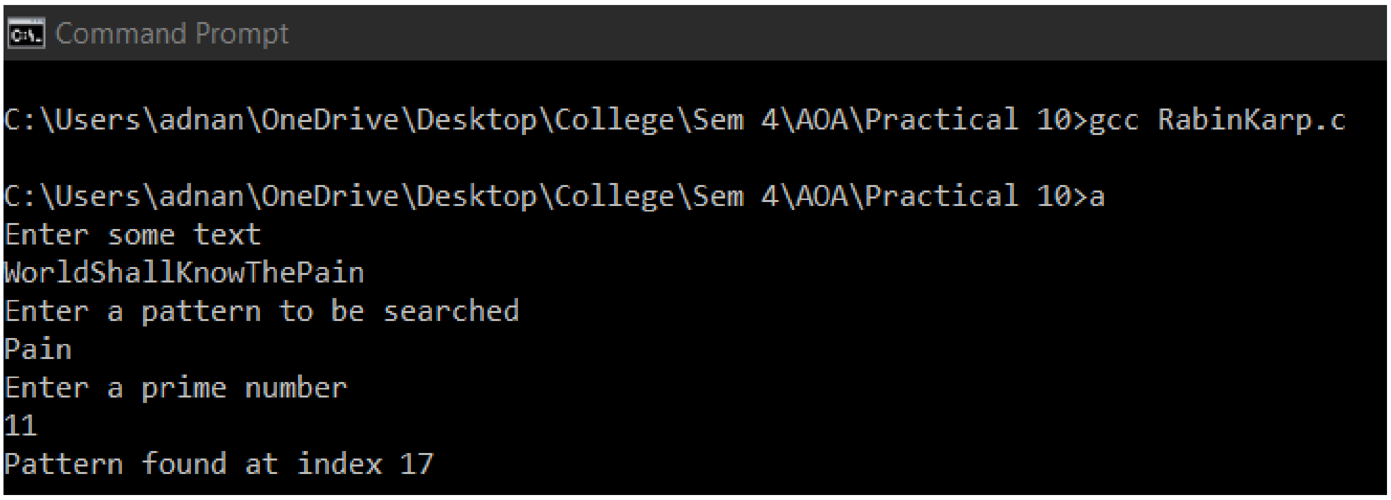
```
Pattern found at index 0 Pattern found at index 9 Pattern found at index 12
```

CODE:

```
#include<stdio.h>
#include<string.h>
#include<conio.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; // hash value for pattern
   int t = 0; // hash value for txt
   int h = 1;

   // The value of h would be "pow(d, M-1)%q"
   for (i = 0; i < M-1; i++)
      h = (h*d)%q;

   // Calculate the hash value of pattern and first
   // window of text
   for (i = 0; i < M; i++)
   {
      p = (d*p + pat[i])%q;
      t = (d*t + txt[i])%q;
   }

   // Slide the pattern over text one by one
   for (i = 0; i <= N - M; i++)
   {

      // Check the hash values of current window of text
      // and pattern. If the hash values match then only
      // check for characters on by one
      if ( p == t )
      {
         /* Check for characters one by one */
         for (j = 0; j < M; j++)
         {
            if (txt[i+j] != pat[j])
               break;
         }

         // if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]
         if (j == M)
            printf("Pattern found at index %d \n", i);
      }
```

```
        // Calculate hash value for next window of text: Remove
        // leading digit, add trailing digit
        if ( i < N-M )
        {
            t = (d*(t - txt[i]*h) + txt[i+M])%q;

            // We might get negative value of t, converting it
            // to positive
            if (t < 0)
            t = (t + q);
        }
    }
}


int main()
{   char txt[80],pat[80];
    int q;

    printf("Enter some text \n");
    scanf("%s",txt);
    printf("Enter a pattern to be searched \n");
    scanf("%s",&pat);
    printf("Enter a prime number \n");
    scanf("%d",&q);
    search(pat, txt, q);

    return 0;
}
```

OUTPUT:



Command Prompt

```
C:\Users\adnan\OneDrive\Desktop\College\Sem 4\AOA\Practical 10>gcc RabinKarp.c

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\AOA\Practical 10>a
Enter some text
WorldShallKnowThePain
Enter a pattern to be searched
Pain
Enter a prime number
11
Pattern found at index 17
```

CONCLUSION: The average and best-case running time of the Rabin-Karp algorithm is O(n+m), but its worst-case time is O(nm). Worst case of Rabin-Karp algorithm occurs when all characters of pattern and text are same as the hash values of all the substrings of txt[] match with hash value of pat[]. For example pat[] = “AAA” and txt[] = “AAAAAAA”.