Experiment - 9

Student Name: Diksha **UID:** 23BCS10994
Branch: BE-CSE **Section/Group:** KRG-2B
Semester: 5th **Date of Performance:** 13/10/25
Subject Name: Design and Analysis of Algorithms
Subject Code: 23CSH-301

1. **Aim:** Develop a program and analyze complexity to find all occurrences of a pattern P in a given stringS.

2. **Objective :** Analyze to find all occurrences of a pattern P in a given string S

3. **Input/Apparatus Used:** String is taken as input in order to find pattern from it.

4. **Procedure:**

1. We will first create the LPS array.

2. Initialize two variables - „strIdx‟ and „patIdx‟ to iterate over the string and the pattern, respectively.

3. If „pat[patIdx]‟ equals „str[strIdx]‟, we will increment both the indexes.

4. When „patIdx‟ equals the length of the pattern, this means that the pattern is found in the string. Therefore we print the index and set „patIdx‟ = LPS[patIdx-1].

5. If „pat[patIdx]‟ is not equal to „str[strIdx]‟, we update the patIdx witht he last index that matches with „str[strIdx]‟ using the LPS array.

Doing this, we will find all occurrences of the pattern in the string.

5. **Algorithm:**

String manipulation/matching algorithms: Rabin Karp algorithm

Naïve brute-force algorithm:

Naïve pattern searching is the simplest method among other pattern searching algorithms. It checks for all character of the main string to the pattern. This algorithm is helpful for smaller texts. It does not need any pre-processing phases. We can find substring by checking once for the string. It also does not occupy extra space to perform the operation.

```
NAIVE-STRING-MATCHER($T, P$)
$n = T.length$
$m = P.length$
for $s = 0$ to $n - m$
    if $P[1..m] == T[s+1..s+m]$
        print "Pattern occurs with shift" $s$
```

The Rabin-Karp-Algorithm

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.

```
RABIN-KARP-MATCHER($T, P, d, q$)
$n = T.length$
$m = P.length$
$h = d^{m-1} \bmod q$
$p = 0$
$t_0 = 0$
for $i = 1$ to $m$                         // preprocessing
    $p = (dp + P[i]) \bmod q$
    $t_0 = (dt_0 + T[i]) \bmod q$
for $s = 0$ to $n - m$                     // matching
    if $p == t_s$
        if $P[1..m] == T[s+1..s+m]$
            print "Pattern occurs with shift" $s$
    if $s < n - m$
        $t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q$
```

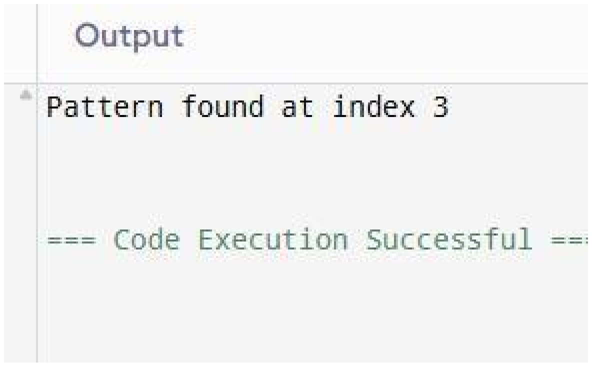
Example: For string matching, working module q = 11, how many spurious hits does the Rabin-Karp matcher encounters in Text T = 31415926535.......

1. T = 31415926535.......
2. P = 26
3. Here T.Length =11 so Q = 11
4. And P mod Q = 26 mod 11 = 4
5. Now find the exact match of P mod Q...
6. **Code and Output :**

```
#include

<iostream>

#include <string>

using namespace

std;


  void rabinKarp(string text, string pattern, int

     q) { int d = 256; // number of characters in

     input alphabet int n = text.length(); int m =

     pattern.length(); int p = 0; // hash value for

     pattern int t = 0; // hash value for text int h

     = 1;
```

```
// The value of h would be pow(d,

m-1) % q for (int i = 0; i < m - 1;

i++) h = (h * d) % q;


// Calculate hash value for pattern and first

window of text for (int i = 0; i < m; i++) { p = (d

* p + pattern[i]) % q; t = (d * t + text[i]) % q;

}


// Slide the pattern over the text one

by one for (int i = 0; i <= n - m; i++)

{

  // If the hash values match, check for characters

  one by one if (p == t) { bool match = true; for (int

  j = 0; j < m; j++) {

          if (text[i + j] !=

            pattern[j]) { match

            = false; break;
            } } if (match) cout << "Pattern found at
          index " << i << endl;
        }

        // Calculate hash value for next window
        of text if (i < n - m) { t = (d * (t - text[i]
        * h) + text[i + m]) % q; if (t < 0) t = (t +
        q);
        }
      }
    }

      int main() { string text =
        "ABCCDDAEFG";
        string pattern = "CDD";
        int q = 101; // A prime
        number
      rabinKarp(text, pattern, q);
      return 0;
```

}



```
Pattern found at index 3


=== Code Execution Successful ==
```

7. Complexity:

The running time of **RABIN-KARP-MATCHER** in the worst-case scenario **O ((n-m+1) m** .