Experiment - 9

Student Name: Himanshu Gupta
UID: 23BCS10889
Branch: BE-CSE
Section/Group: KRG-2B
Semester: 5th
Date of Performance: 17/10/25
Subject Name: DAA
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)
1  n = T.length
2  m = P.length
3  for s = 0 to n − m
4      if P[1..m] == T[s + 1..s + m]
5          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)
 1  n = T.length
 2  m = P.length
 3  h = d^(m−1) mod q
 4  p = 0
 5  t_0 = 0
 6  for i = 1 to m                    // preprocessing
 7      p = (dp + P[i]) mod q
 8      t_0 = (dt_0 + T[i]) mod q
 9  for s = 0 to n − m                // matching
10      if p == t_s
11          if P[1..m] == T[s + 1..s + m]
12              print "Pattern occurs with shift" s
13      if s < n − m
14          t_{s+1} = (d(t_s − T[s + 1]h) + T[s + m + 1]) mod 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 :

```
public class RabinKarp {
    public static final int d = 256;

    public static void RabinKarpSearch(String text, String pattern, int q) {
        int n = text.length();
        int m = pattern.length();
        if (n < m) return;
        int p = 0, t = 0, h = 1;
        for (int i = 0; i < m - 1; i++)
            h = (h * d) % q;
        for (int i = 0; i < m; i++) {
            p = (d * p + pattern.charAt(i)) % q;
            t = (d * t + text.charAt(i)) % q;
        }
        for (int i = 0; i <= n - m; i++) {
            if (p == t) {
                int j;
                for (j = 0; j < m; j++) {
                    if (text.charAt(i + j) != pattern.charAt(j))
                        break;
                }
                if (j == m)
                    System.out.println("Pattern found at index " + i);
            }
            if (i < n - m) {
                t = (d * (t - text.charAt(i) * h) + text.charAt(i + m)) % q;
                if (t < 0)
                    t = t + q;
            }
        }
    }

    public static void main(String[] args) {
        java.util.Scanner sc = new java.util.Scanner(System.in);
        String text, pattern;
        int q;
        System.out.print("Enter the text: ");
        text = sc.nextLine();
        System.out.print("Enter the pattern: ");
        pattern = sc.nextLine();
```

```
        System.out.print("Enter a prime number for hashing: ");
        q = sc.nextInt();
        RabinKarpSearch(text, pattern, q);
        sc.close();
    }
}
```

```
Enter the text: HimanshuGUptahhhhh
Enter the pattern: im
Enter a prime number for hashing: 3
Pattern found at index 1
PS C:\Users\ASUS\Desktop\Sem 5\DAA_23BCS10889_KRG-2B>
```

7. Complexity:

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