Experiment-9

Student Name: Tejinderpal Singh
UID: 23BCS80343
Branch: BE-CSE
Section/Group: 23BCS_KRG_3A
Semester: 5th
Subject Code: 23CSH-301
Subject Name: DAA

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

2. **Objective:** To implement Rabin-Karp algorithm in C++ for string matching and efficiently find all occurrences of a pattern in a given text.

3. **Procedure:**

   1. Start
   2. Compute the hash value of the pattern P and the first substring of S of length equal to the pattern.
   3. Initialize indexes strIdx and patIdx to iterate over the text and the pattern.
   4. For each substring of S:
      - If hash values match, compare characters of substring with the pattern to confirm the match.
      - If pattern matches, print the index of occurrence.
   5. Compute the hash for the next substring of S using a rolling hash formula to avoid recomputation.
   6. Repeat until the end of the text is reached.
   7. This approach finds all occurrences of the pattern efficiently.

4. **Code:**

```
#include <iostream>
#include <string>
using namespace std;

#define d 256

void rabinKarp(string text, string pattern, int q) {
    int n = text.length();
    int m = pattern.length();
    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 + pattern[i]) % q;
       t = (d*t + text[i]) % q;
    }

    for(i = 0; i <= n - m; i++) {
       if(p == t) {
          for(j = 0; j < m; j++) {
             if(text[i+j] != pattern[j])
                break;
          }
          if(j == m)
             cout << "Pattern found at index " << i << endl;
       }
       if(i < n - m) {
          t = (d*(t - text[i]*h) + text[i+m]) % q;
          if(t < 0) t += q;
       }
    }
}

int main() {
    string text, pattern;
    int q;
    cout << "Enter text: ";
    cin >> text;
    cout << "Enter pattern: ";
    cin >> pattern;
    cout << "Enter prime modulus q: ";
    cin >> q;
    rabinKarp(text, pattern, q);
    return 0;
}
```

5. Observations:

```
Enter text: 31415926535
Enter pattern: 26
Enter prime modulus q: 11
Pattern found at index 6
```

6. Time Complexity:

Case	Time Complexity
Best Case	O(n+m)
Average Case	O(n+m)
Worst Case	$O((n-m+1)\cdot m) \approx O(n\cdot m)$

7. Learning Outcome:

- Learned how to implement Rabin-Karp string matching algorithm using C++.
- Understood the concept of hashing and rolling hash for efficient pattern searching.
- Gained experience in finding all occurrences of a pattern in a text.
- Learned the difference between naive and hash-based string matching.
- Strengthened understanding of average-case vs worst-case complexity for string algorithms.