WORKSHEET 9

Student Name: Ishu **UID: 22BCS15695**

Branch: CSE **Section/Group: FL_IOT_603'B'**

Semester: 5th **Date of Performance:10/10/24**

Subject Name: Design and Analysis of Algorithms **Subject Code: 22CSH-311**

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

2. **Objectives:** Analyze to find all occurrences of a pattern P in a given string S using Rabin Karp algorithm.

3. **Algorithm:**

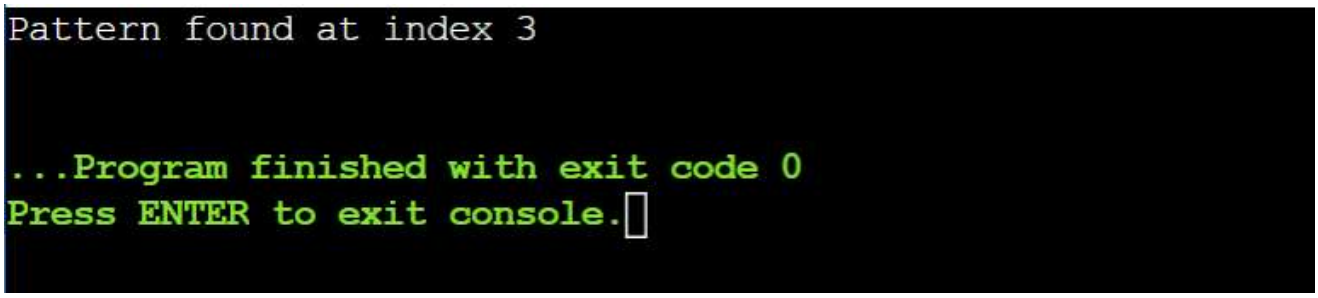
- Calculate the hash of the pattern and the first window (substring) of the text.
- For a string of length M, the hash value is computed as: $hash=(d_0\times char[0]+d_1\times char[1]+...+d_{M-1}\times char[M-1]) \bmod q$
- d is the number of characters in the input alphabet (usually 256 for ASCII).
- q is a prime number used to keep the hash value within a range, reducing the chance of overflow and hash collisions.
- Slide the pattern over the text, one character at a time.
- Compare hash values of the pattern and the current text window.
- If the hashes match, compare the actual characters to confirm the match.
- Recalculate the hash for the next window efficiently by removing the first character and adding the next.
- Repeat this until the pattern is found or the entire text is checked.

4. Implementation/Code:

```
#include <bits/stdc++.h>
using namespace std;
#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)
				cout << "Pattern found at index " << i
					<< endl;
		}
		if (i < N - M) {
			t = (d * (t - txt[i] * h) + txt[i + M]) % q;

			if (t < 0)
				t = (t + q);
		}
	}
}
```

```
int main()
{
        char txt[] = "ABCCDDAEFG";
        char pat[] = "CDD";
        int q = INT_MAX;
        search(pat, txt, q);
        return 0;
}
```

5. Output:



6. Time Complexity:

Worst Case: O((n-m+1) m).

7. Learning Outcome:

1) Learnt how to perform efficient string matching using hashing techniques.
2) Learnt how to slide over a text efficiently by recalculating hash values without recomputing them from scratch, optimizing performance.
3) Learnt how Rabin-Karp algorithm is used to search for a pattern in a text by using hash values.