Name: Rishi Tiku

Class: SE DS

UID: 2021700067

Subject: DAA LAB

Experiment: 10

Aim: To implement Rabin Karp's String Matching Algorithm.

Problem:

Given a sequence S[1….n] and a pattern P[1…m], m<=n, find the position at which the pattern exists in the sequence.

Theory:

Rabin-Karp algorithm is an algorithm used for searching/matching patterns in the text using a hash function. Unlike Naive string matching algorithm, it does not travel through every character in the initial phase rather it filters the characters that do not match and then performs the comparison.
Suppose p represents values corresponding to pattern P[1…m] of length m. And t_s represents values of m-length substrings T[(s + 1) … (s + m)] for s = 0, 1, 2, …, n – m.

We can compute p in O(m) time and all t_s can be computed in O(n – m + 1) time.

Rabin Karp algorithm is based on hashing technique. It first computes the hash value of p and t_s. If hash values are same, i.e. if hash(p) = hash(t_s), we check the equality of inverse hash similar to a naïve method. If hash values are not same, no need to compare actual string.

When the hash value of the pattern matches with the hash value of a window of the text but the window is not the actual pattern then it is called a spurious hit.

Spurious hit increases the time complexity of the algorithm. In order to minimize spurious hit, we use modulus. It greatly reduces the spurious hit.

Time Complexity: O(m+n), worst: O(mn) if all hash to same value (all spurious hits)
Auxiliary Space: O(1)

Algorithm

1. RABIN_KARP(T, P)
2. // T is text of length n
3. // P is pattern of length m

4. h ← power(d, m – 1) mod q
5. p ← 0
6. t_0 ← 0

7. **for** i ← 1 to m **do**
8. p ← (d*p + P[i]) mod q
9. t_0 ← ((d * t_0) + T[i]) mod q
10. **end**

11. **for** s ← 0 to n – m **do**
12. **if** p == t_0 **then**
13. **if** P[1…m] == T[s+1 … s + m] **then**
    a. **print** “Match found at shift”, s
14. **end**
15. **end**
16. **if** s < (n – m) **then**
17. t_{s+1} ← (d*(t_s – T[s + 1]) * h) + T[s + m + 1]
18. **end**
19. **end**

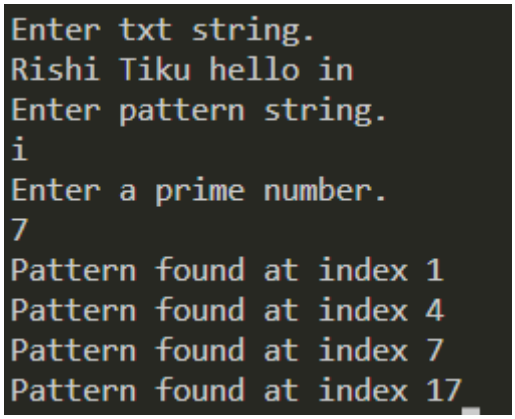
Code

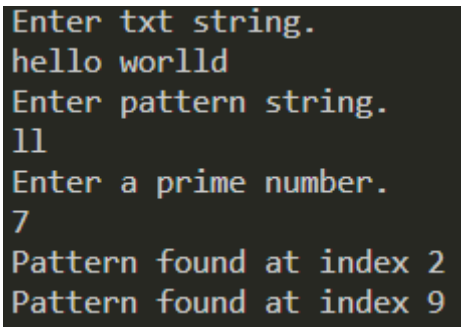
```
#include <stdio.h>
#include <string.h>
```

```c
// d is the number of characters in the input alphabet
#define d 256

int flag = 0;

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++) {

        // If the hash values match then only
        // check for characters one 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);
```

```
                flag = 1;
            }
        }

        // 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;
            // if t<0, change t to positive
            if (t < 0)
                t = (t + q);
        }
    }
}

/* Driver Code */
int main()
{
    char txt[500] = "";
    char pat[500] = "";
    int q = 101;// q should be a prime number

    printf("Enter txt string.\n");
    scanf("%[^\n]s", txt);
    fflush(stdin);
    printf("Enter pattern string.\n");
    scanf("%[^\n]s", pat);
    printf("Enter a prime number.\n");
    scanf("%d", &q);
    search(pat, txt, q);
    if(flag == 0){
        printf("No sequence matched.\n");
    }
    return 0;
}
```

Output





Conclusion

Solved the problem of string matching by using Rabin Karp's Algorithm.