Pune Institute of Computer Technology

Department of Computer Engineering

(2022- 2023)

“Implement Naïve string matching algorithm and Rabin-Karp algorithm”

Submitted to the

Savitribai Phule Pune University

In partial fulfilment for the award of the Degree of

Bachelor of Engineering

in

Computer Engineering

By

1)	**Medha Badamikar**	**41108**
2)	**Udayan Chavan**	**41117**
3)	**Tejas Deshpande**	**41122**

Under the guidance of

<u>Prof. Vaishali Kandekar</u>

Problem Statement

Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching. Observe difference in working of both the algorithms for the same input.

Objective

To implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching.

Theory

String matching

Given, a string of characters (say text) and a smaller string of characters (say pattern), we need to find the number of occurrences of the pattern in the string.To define it more formally, we can say text is an array T[*1…n*] and pattern is an array P[*1…m*] where $m \leq n$.

Naïve String matching

In the naive algorithm, we check if P[1…m] matches with T[s+1…s+m] for each of the above (n — m +1) possible values of s. I know, tedious right? Let's take a smaller string to look at the steps real quick so that we have a clear understanding of what's going on.

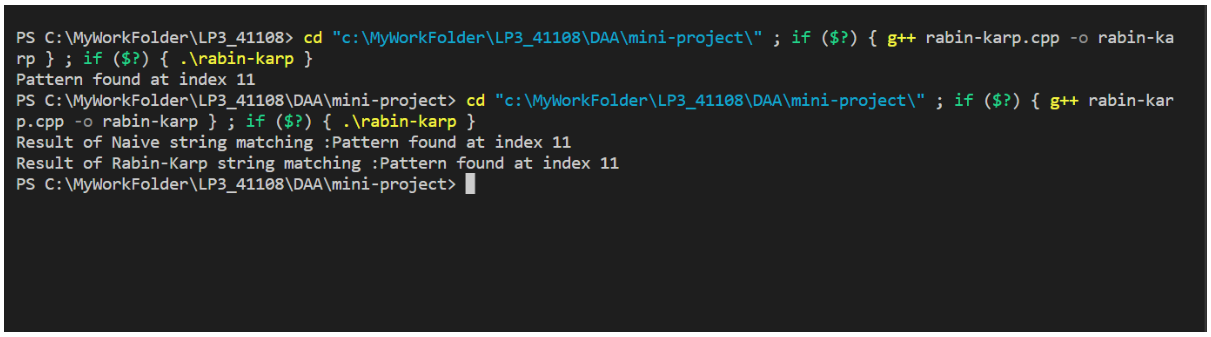
Rabin Karp algorithm:

- We divide the text T into *m*-sized windows and give them a **hash value**.
- This hash value is matched with the pre-calculated hash value of the pattern P.
- If the values match, we perform a complete string comparison from that position. This is done to avoid **false positives**, i.e., the text window hash value matches the pattern hash value even if they are not the same.
- If the value does not match, we move (or better "roll") to the next window using the **rolling hash technique** (I shall explain soon).

CODE :

```
#include <bits/stdc++.h>
using namespace std;

// d is the number of characters in the input alphabet
#define d 256
//Naive search
void search(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    /* A loop to slide pat[] one by one */
    for (int i = 0; i <= N - M; i++) {
        int j;

        /* For current index i, check for pattern match */
        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;

        if (j
            == M) // if pat[0...M-1] = txt[i, i+1, ...i+M-1]
            printf("Pattern found at index %d \n", i);
    }
}

void RabinKarpsearch(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;
    }

    for (i = 0; i <= N - M; i++) {
        // Check the hash values of current window of text
        // and pattern. 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 (j == M)
                cout << "Pattern found at index " << i
                     << endl;
        }

        // 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;

            // We might get negative value of t, converting
            // it to positive
            if (t < 0)
                t = (t + q);
        }
    }
}

int main()
{
    char txt[] = "Today is a dull day";
    char pat[] = "dull";

    // we mod to avoid overflowing of value but we should
    // take as big q as possible to avoid the collison
    int q = INT_MAX;
    search(pat,txt);
    RabinKarpsearch(pat, txt, q);
    return 0;
}
```

OUTPUT :



```
PS C:\MyWorkFolder\LP3_41108> cd "c:\MyWorkFolder\LP3_41108\DAA\mini-project\" ; if ($?) { g++ rabin-karp.cpp -o rabin-ka
rp } ; if ($?) { .\rabin-karp }
Pattern found at index 11
PS C:\MyWorkFolder\LP3_41108\DAA\mini-project> cd "c:\MyWorkFolder\LP3_41108\DAA\mini-project\" ; if ($?) { g++ rabin-kar
p.cpp -o rabin-karp } ; if ($?) { .\rabin-karp }
Result of Naive string matching :Pattern found at index 11
Result of Rabin-Karp string matching :Pattern found at index 11
PS C:\MyWorkFolder\LP3_41108\DAA\mini-project>
```

Conclusion

We have implemented Naive string matching and Rabin-Karp algorithm and analysed their performance.