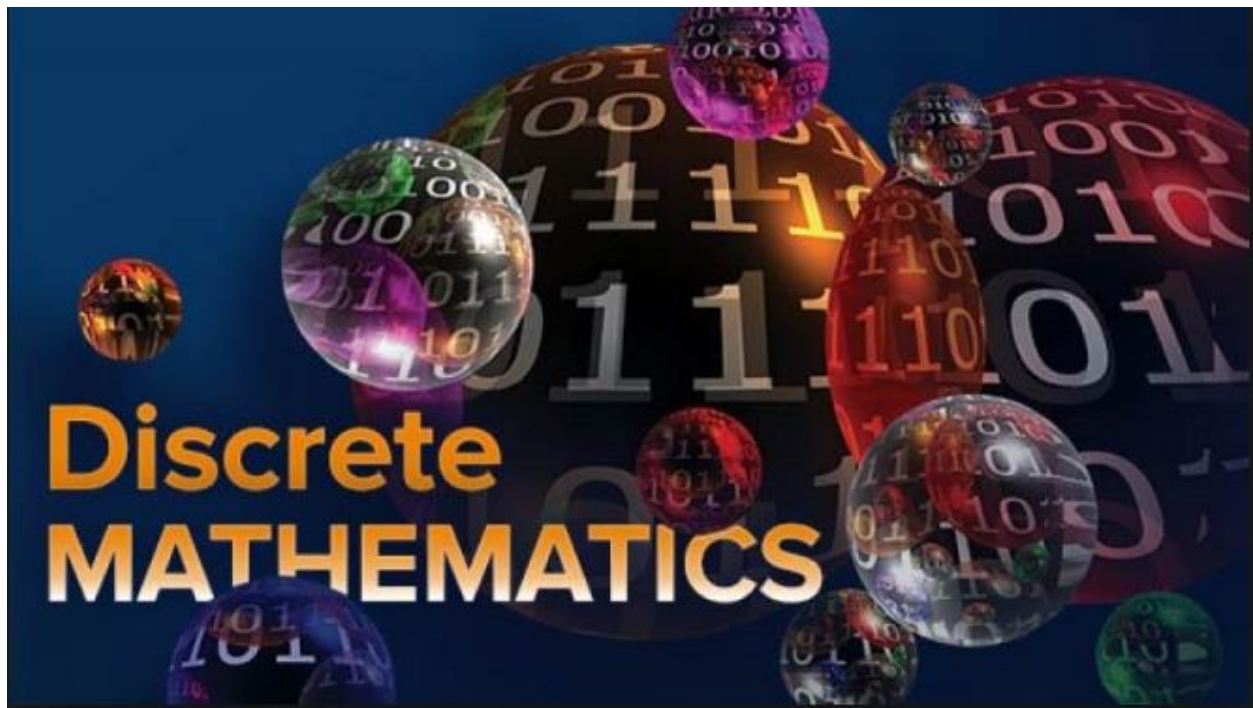


PROJECT NAME:

STRING MATCHING BY RABIN KARP
ALGORITHM

SUBMITTED TO: SIR ADNALLAH



GROUP MEMBER NAMES

JAWERIA ASIF (9442)

JAVERIA HASSAN (9517)

PROJECT TITLE:

“STRING MATCHING BY RABIN KARP ALGORITHM”

(AN IMPLEMENTATION OF HASH TABLE)

TOOLS:

- **Microsoft Visual Studio (2010) On C# Language**

INTRODUCTION:

- In computer science, the **Rabin-Karp algorithm** or **Karp-Rabin algorithm** is a string-searching algorithm created by Richard M. Karp and Michael O. Rabin (1987) that uses hashing to find any one of a set of pattern strings in a text.
- For text of length n and p patterns of combined length m , its average and best case running time is $O(n+m)$ in space $O(p)$, but its worst-case time is $O(nm)$.
- In contrast, the Aho-Corasick string-matching algorithm has asymptotic worst-time complexity $O(n+m)$ in space $O(m)$.
- A practical application of the algorithm is detecting plagiarism. Given source material, the algorithm can rapidly search through a paper for instances of sentences from the source material, ignoring details such as case and punctuation. Because of the abundance of the sought strings, single-string searching algorithms are impractical.

PROJECT DESCRIPTION:

- Rabin-Karp algorithm slides the pattern one by one. But,
- Rabin Karp algorithm matches the hash value of the pattern with the hash value of current substring of text
- If the hash values match then only it starts matching individual characters.
- Rabin Karp algorithm needs to calculate hash values for following strings.
 - ❖ Pattern itself.
 - ❖ All the substrings of text of length m.
- Since we need to efficiently calculate hash values for all the substrings of size m of text, we must have a hash function which has following property.
- Hash at the next shift must be efficiently computable from the current hash value and next character in text or we can say $hash(txt[s+1 .. s+m])$ must be efficiently computable from $hash(txt[s .. s+m-1])$ and $txt[s+m]$ i.e., $hash(txt[s+1 .. s+m]) = rehash(txt[s+m], hash(txt[s .. s+m-1]))$ and rehash must be $O(1)$ operation.
- The hash function suggested by Rabin and Karp calculates an integer value. The integer value for a string is numeric value of a string. For example, if all possible characters are from 1 to 10, the numeric value of "122" will be 122.
- The number of possible characters is higher than 10 (256 in general) and pattern length can be large.
- The numeric values cannot be practically stored as an integer. Therefore, the numeric value is calculated using modular arithmetic to make sure that the hash values can be stored in an integer variable (can fit in memory words).

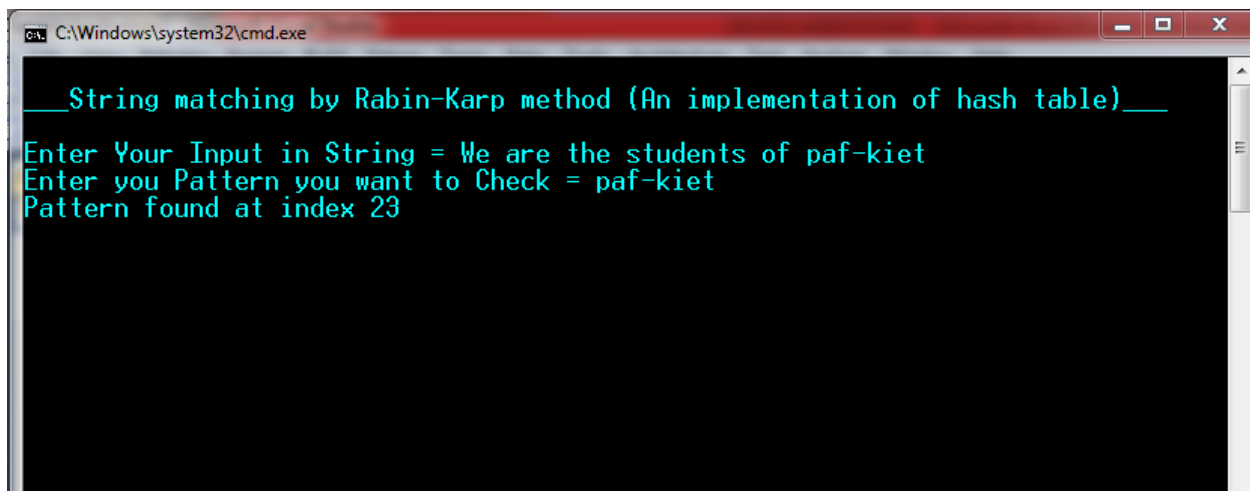
[DOCUMENTATION OF DM PROJECT]

- To do rehashing, we need to take off the most significant digit and add the new least significant digit for in hash value.
- Rehashing is done using the following formula.

$$\text{hash}(\text{txt}[s+1 .. s+m]) = (d (\text{hash}(\text{txt}[s .. s+m-1]) - \text{txt}[s] * h) + \text{txt}[s + m]) \mod q$$

- $\text{hash}(\text{txt}[s .. s+m-1])$: Hash value at shift s .
- $\text{hash}(\text{txt}[s+1 .. s+m])$: Hash value at next shift (or shift $s+1$)
- d : Number of characters in the alphabet
- q : A prime number
- $h: d^{(m-1)}$

PROJECT FIGURE/OUTPUT:



```
C:\Windows\system32\cmd.exe

__String matching by Rabin-Karp method (An implementation of hash table)__
Enter Your Input in String = We are the students of paf-kiet
Enter you Pattern you want to Check = paf-kiet
Pattern found at index 23
```

SOURCE CODE:

```
// Following program is a C# implementation
// of Rabin Karp Algorithm given in the CLRS book
using System;
public class GFG
{
    // d is the number of characters in the input alphabet
    public readonly static int d = 256;

    /* pat -> pattern
       txt -> text
       q -> A prime number
    */
    static void search(String pat, String txt, int q)
    {
        int M = pat.Length;
        int N = txt.Length;
        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++)
        {
            // Check the hash values of current window of text
            // and pattern. If the hash values match then only
            // check for characters on by one
            if ( p == t )
            {
                /* Check for characters one by one */
                for (j = 0; j < M; j++)
                {
                    if (txt[i+j] != pat[j])
                        break;
                }
            }
        }
    }
}
```

[DOCUMENTATION OF DM PROJECT]

```
        // if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]
        if (j == M)
            Console.WriteLine("Pattern found at index " + i);
    }

    // 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);
    }
}

/* Driver program to test above function */
public static void Main()
{
    String txt = "We are the students of KIET";
    String pat = "KIET";
    int q = 101; // A prime number
    search(pat, txt, q);
}
}
```