	Marwari University **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing String Matching Approach	
Experiment No: 11	**Date:**	**Enrollment No: 92200133030**

Aim: Implementing String Matching Approach

IDE: Visual Studio Code

1. Implement the Naive based approach to find the pattern within the string

Theory: -

- The problem involves identifying all the positions where a given pattern appears in a longer text. Given the text of length n and a pattern of length mmm (n > m), the goal is to efficiently find all the starting indices in the text where the pattern matches.

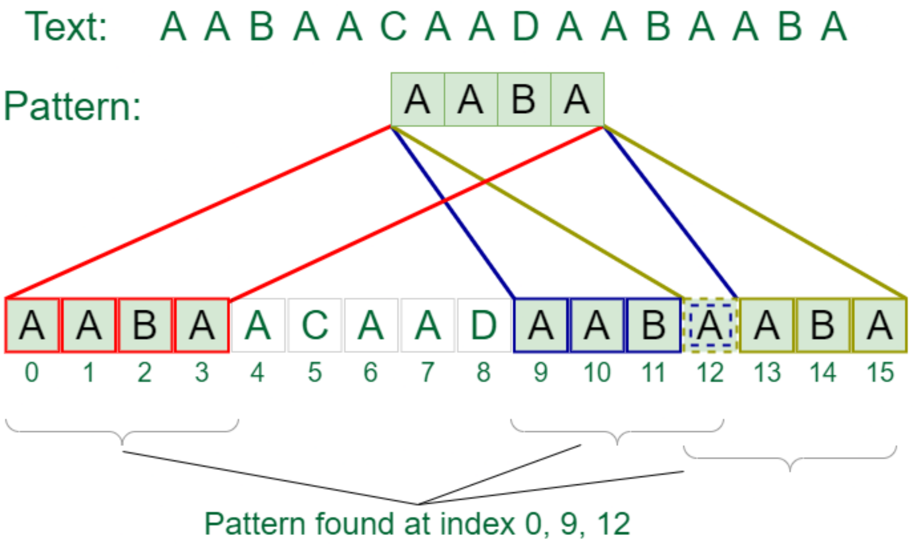
1. Key Concepts:-

- **Pattern Matching:** The task of locating a specific sequence of characters (the pattern) within a larger sequence (the text).
- **Occurrences:** The indices in the text where the first character of the pattern aligns with a matching substring.

2. Naive Approach:-

- A straightforward way to solve the problem is to:

1. Slide the pattern over the text from the beginning to n−m.
2. At each position, compare the substring of length mmm with the pattern.
3. If all characters match, record the starting index.


Text: A A B A A C A A D A A B A A B A
Pattern: A A B A
A A B A A C A A D A A B A A B A
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Pattern found at index 0, 9, 12

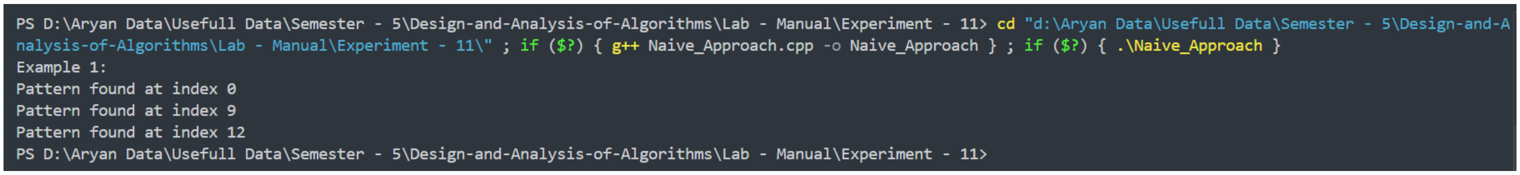
Algorithm: -

Programming Language: - C++

Code :-

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

void search(string& pat, string& txt) {
    int M = pat.size();
    int N = txt.size();

    for (int i = 0; i <= N - M; i++) {
        int j;
        for (j = 0; j < M; j++) {
            if (txt[i + j] != pat[j]) {
                break;
            }
        }
        if (j == M) {
            cout << "Pattern found at index " << i << endl;
        }
    }
}

int main() {
    string txt1 = "AABAACAADAABAABA";
    string pat1 = "AABA";
    cout << "Example 1: " << endl;
    search(pat1, txt1);
    return 0;
}
```

Output :-



```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 11> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 11\" ; if ($?) { g++ Naive_Approach.cpp -o Naive_Approach } ; if ($?) { .\Naive_Approach }
Example 1:
Pattern found at index 0
Pattern found at index 9
Pattern found at index 12
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 11>
```

Space Complexity:- __________

Justification: -

____

____

____

____

____

Time Complexity:

Best Case Time Complexity: __________

Justification: -

____

____

____

____

____

Worst Case Time Complexity:- ____________

Justification: -

2. Implement the Rabin-Karp approach to find the pattern within the string

Theory: -

- The Rabin-Karp algorithm is an efficient pattern-matching algorithm that uses hashing to find all occurrences of a pattern in a given text. Instead of comparing substrings character by character, it compares their hash values, significantly improving performance in many cases.

Key Concepts:-

1) Hash Function:

- A hash function maps a string to a numeric value.
- For this algorithm, a rolling hash function is used, which allows efficient computation of hash values for consecutive substrings in O(1) time.

2) Rolling Hash:

- The hash of a substring s[i:i+m] is computed based on the hash of s[i−1:i+m−1].
- Formula: $\text{hash}(s[i:i+m])=(\text{base}\cdot(\text{hash}(s[i-1:i+m-1])-\text{ord}(s[i-1])\cdot \text{base}^{m-1})+\text{ord}(s[i+m-1]))\bmod \text{modulus}$
- This avoids recalculating the entire hash from scratch.

3) Collision:

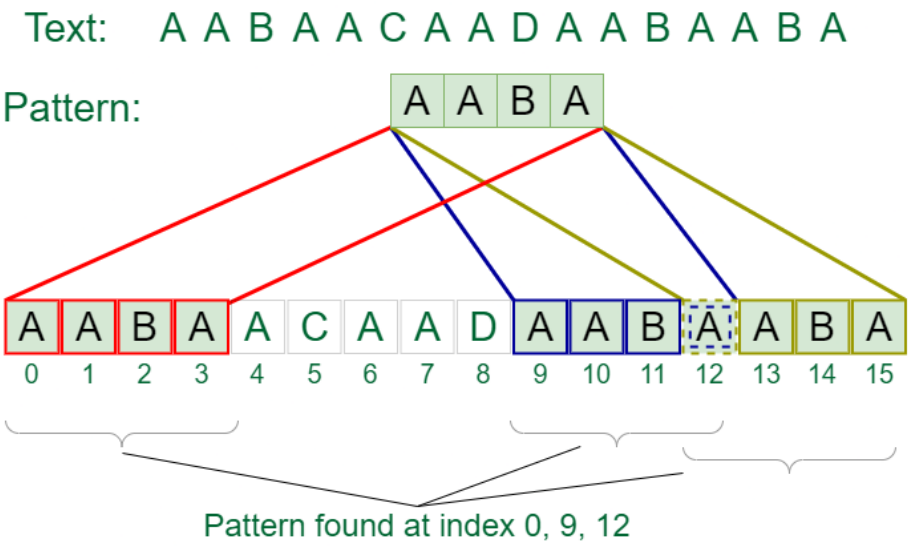
- Two different strings may have the same hash value due to hash collisions.
- To verify a match, the algorithm performs a character-by-character comparison of the substring and the pattern.

Steps of the Rabin-Karp Algorithm:-

1. Compute the hash value of the pattern ($h_{pattern}$) and the first window of the text (h_{text}).
2. Slide the pattern over the text:
   - Update the hash value of the current window using the rolling hash formula.
   - Compare $h_{pattern}$ and h_{text}.

- If the hash values match, perform a direct string comparison to confirm.

3. Record the index if a match is found.
4. Repeat until the entire text has been scanned.



Text: A A B A A C A A D A A B A A B A
Pattern: A A B A
A A B A A C A A D A A B A A B A
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Pattern found at index 0, 9, 12

Algorithm: -

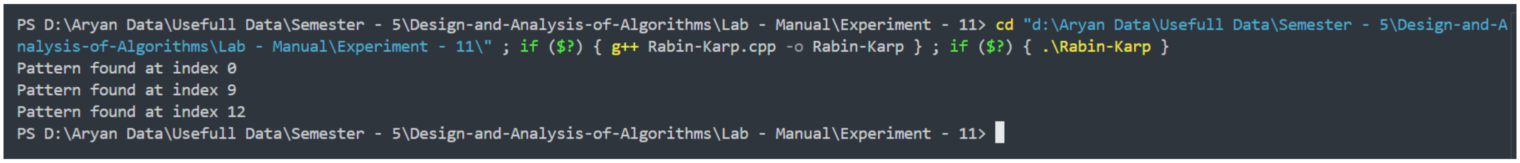
	Marwari University **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Design and Analysis of Algorithms (01CT0512)	**Aim:** Implementing String Matching Approach	
Experiment No: 11	**Date:**	**Enrollment No: 92200133030**

Programming Language: - C++

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[] = "AABAACAADAABAABA";
    char pat[] = "AABA";

    int q = INT_MAX;

    search(pat, txt, q);
    return 0;
}
```

Output :-



```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 11> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-A
nalysis-of-Algorithms\Lab - Manual\Experiment - 11\" ; if ($?) { g++ Rabin-Karp.cpp -o Rabin-Karp } ; if ($?) { .\Rabin-Karp }
Pattern found at index 0
Pattern found at index 9
Pattern found at index 12
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 11>
```

Space Complexity:- __________
Justification: -

____

____

____

____

____

Time Complexity:

Best Case Time Complexity: __________
Justification: -

____

____

____

____

____

Worst Case Time Complexity:- ___________

Justification: -

Conclusion:-