Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering
Semester: (Fall, Year:2024), B.Sc. in CSE (Day)

Lab Report NO : 04
Course Title: Algorithms Lab
Course Code: CSE 208 Section: D2

Lab Experiment Name: Implementation of KMP Algorithm in case of in case of integer or others.

Student Details

	Name	ID
1.	Promod Chandra Das	231002005

Lab Date : 04-12-2024
Submission Date : 11-12-2024
Course Teacher's Name : Farjana Akter Jui

Lab Report Status

Marks: …… **Signature:.....**
Comments:..... **Date:.....**

1. TITLE OF THE LAB REPORT EXPERIMENT :

Implementation of KMP Algorithm in case of in case of integer or others.

2. OBJECTIVES :

 To comprehend the operational principles of the Knuth-Morris-Pratt (KMP) algorithm in the context of pattern matching.

 To develop an implementation of the KMP algorithm for searching patterns within integer sequences or other types of data.

 To evaluate the time complexity of the KMP algorithm across different use cases and scenarios.

 To assess the efficiency of the KMP algorithm in comparison with other pattern-matching methods..

3. PROCEDURE :

 Define an integer array as the input sequence and specify the integer pattern to search for.

 Implement the preprocessing phase to compute the Longest Prefix Suffix (LPS) array for the given integer pattern.

- Initialize the LPS array with zeros and iterate through the pattern to calculate values based on matching prefixes and suffixes.

 Utilize the LPS array for pattern matching within the integer array:

- Compare the pattern with the elements of the integer array in a sequential manner.
- Upon encountering a mismatch, leverage the LPS array to skip redundant comparisons by adjusting the pattern accordingly.

 Test the implementation using integer arrays of different sizes and patterns to verify both accuracy and performance.

 Document the results, including the number of shifts, matches, and execution time for each test case.IMPLEMENTATION :

Here is the full java code implementation of KMP Algorithm in case of integer

```
class KMP_String_Matching {
    void KMPSearch(String pat, String txt)
    {
        int M = pat.length();
        int N = txt.length();


        int lps[] = new int[M];
        int j = 0;


        computeLPSArray(pat, M, lps);

        int i = 0;
        while (i < N) {
            if (pat.charAt(j) == txt.charAt(i)) {
                j++;
                i++;
            }
```

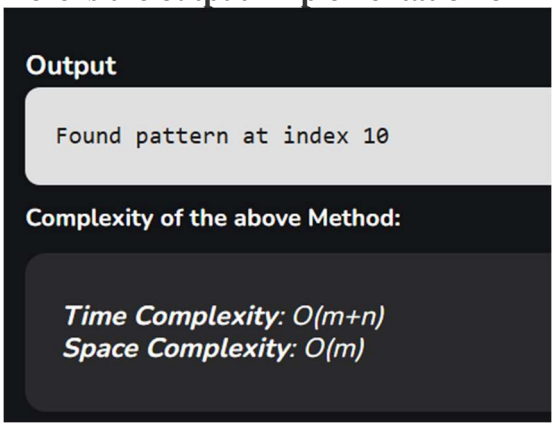
```
            i++;
        }
        if (j == M) {
            System.out.println("Found pattern "
                               + "at index " + (i - j));
            j = lps[j - 1];
        }


        else if (i < N && pat.charAt(j) != txt.charAt(i)) {

            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;
        }
    }
}

void computeLPSArray(String pat, int M, int lps[])
{
```

```
void computeLPSArray(String pat, int M, int lps[])
{

    int len = 0;
    int i = 1;
    lps[0] = 0; // lps[0] is always 0


    while (i < M) {
        if (pat.charAt(i) == pat.charAt(len)) {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {

            if (len != 0) {
                len = lps[len - 1];
```

```
            }
            else
            {
                lps[i] = len;
                i++;
            }
        }
    }
}

// Driver program to test above function
public static void main(String args[])
{
    String txt = "ABABDABACDABABCABAB";
    String pat = "ABABCABAB";
    new KMP_String_Matching().KMPSearch(pat, txt);
}
```

4. OUTPUT :

Here is the output implementation of KMP Algorithm in case of integer



5. DISCUSSION :

The Knuth-Morris-Pratt (KMP) algorithm is an efficient pattern-matching algorithm that can be applied to integer arrays or other data types. It operates by first preprocessing the pattern to create a Longest Prefix Suffix (LPS) array, which stores the lengths of the longest proper prefix that is also a suffix for each substring of the pattern. During the matching phase, the algorithm compares the pattern with the input sequence. If a mismatch occurs, the LPS array is used to skip over already checked positions, avoiding redundant comparisons. This leads to an optimal time complexity of O(n + m), where n is the sequence length and m is the pattern length