Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year: 2023), B.Sc. in CSE (Day)

Lab Report NO: 09

Course Title : Algorithm Lab
Course Code : CSE 204
Section : D - 9

Lab Experiment Name : Implement String and Pattern Matching Problems using KMP Algorithm.

Student Details

Name		ID
1.	Jahidul Islam	221002504

Lab Date : 04/12/2023
Submission Date : 08/12/2023
Course Teacher's Name : Md. Abu Rahman Refat

Lab Report Status

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

1. TITLE:

Implement String and Pattern Matching Problems using KMP Algorithm

2. OBJECTIVES/AIM:

The primary objectives or aim of this lab experiment are:

- Understand the significance of efficient string searching algorithms in computer science.
- Explore and implement the KMP algorithm as a solution for improving string matching efficiency.
- Develop a Java program to apply the KMP algorithm in the context of finding indexes where a given pattern matches a given array.
- Analyze the time complexity and advantages of the KMP algorithm in comparison to other string-matching algorithm

3. PROCEDURE:

The procedure for implementing the KMP algorithm for string and pattern matching involves the following procedural steps:

1. **Understanding the KMP Algorithm:**
    - Review the theoretical concepts behind the KMP algorithm, including the Longest Prefix Suffix (LPS) array.
    - Gain insight into the algorithm's structure and its advantages in optimizing string matching.
2. **Implementing the KMP Algorithm in Java:**
    - Develop a Java program that includes methods for computing the LPS array and performing the actual string and pattern matching using the KMP algorithm.
    - Ensure the program is modular and can be easily integrated into different applications.
3. **Testing the Implementation:**
    - Create sample test cases with diverse input strings and patterns to validate the correctness of the implemented algorithm.
    - Verify that the algorithm accurately identifies the indexes where the given pattern matches the provided text.
4. **Exploring Multiple String-Pattern Pairs:**
    - Extend the implementation to handle multiple pairs of strings and patterns.
    - Implement a method to repeat the algorithm for various combinations and analyze the results.

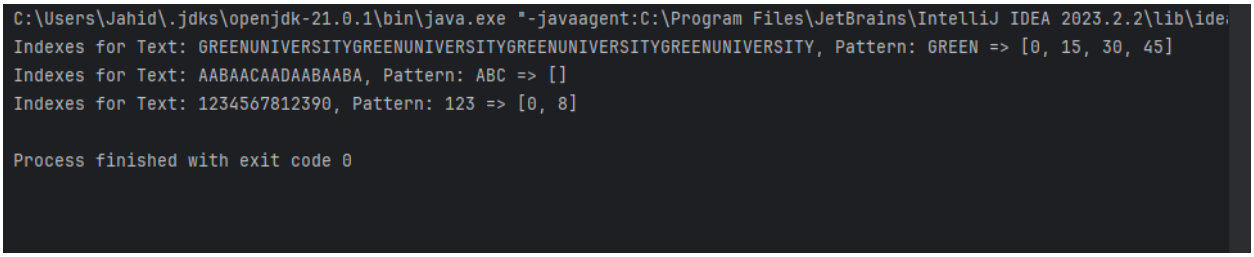
4. IMPLEMENTATION:

Code in Java:

```
import java.util.ArrayList;
import java.util.List;

public class kmpFindIndexMultiplePairs {

    public static List<Integer> kmpSearch(String text, String pattern)
{
        List<Integer> indexes = new ArrayList<>();

        int[] lps = computeLPSArray(pattern);

        int i = 0; // index for text
        int j = 0; // index for pattern

        while (i < text.length()) {
            if (pattern.charAt(j) == text.charAt(i)) {
                i++;
                j++;
            }

            if (j == pattern.length()) {
                // Pattern found at index i - j
                indexes.add(i - j);
                j = lps[j - 1];
            } else if (i < text.length() && pattern.charAt(j) !=
text.charAt(i)) {
                if (j != 0) {
                    j = lps[j - 1];
                } else {
                    i++;
                }
            }
        }

        return indexes;
    }

    private static int[] computeLPSArray(String pattern) {
        int[] lps = new int[pattern.length()];
        int len = 0;
        int i = 1;

        while (i < pattern.length()) {
```

```
            if (pattern.charAt(i) == pattern.charAt(len)) {
                len++;
                lps[i] = len;
                i++;
            } else {
                if (len != 0) {
                    len = lps[len - 1];
                } else {
                    lps[i] = 0;
                    i++;
                }
            }
        }

        return lps;
    }

    public static void main(String[] args) {
        // Example Usage
        List<String> texts =
List.of("GREENUNIVERSITYGREENUNIVERSITYGREENUNIVERSITYGREENUNIVERSITY"
, "AABAACAADAABAABA", "1234567812390");
        List<String> patterns = List.of("GREEN", "ABC", "123");

        searchMultiplePatterns(texts, patterns);


    }

    public static void searchMultiplePatterns(List<String> texts,
List<String> patterns) {
        for (int i = 0; i < texts.size() && i < patterns.size(); i++)
{
            String text = texts.get(i);
            String pattern = patterns.get(i);

            List<Integer> indexes = kmpSearch(text, pattern);

            System.out.println("Indexes for Text: " + text + ",
Pattern: " + pattern + " => " + indexes);
        }
    }
}
```

5. TEST RESULT / OUTPUT:



6. DISCUSSION AND ANALYSIS:

The discussion and analysis phase focuses on evaluating the effectiveness and efficiency of the implemented KMP algorithm:

We implemented the code for **multiple pairs of given strings and patterns.**

- **Time Complexity Analysis:**
    1. Computing the Longest Prefix Suffix (LPS) Array: **O(M)**
        a) The time complexity for computing the LPS array is linear in the length of the pattern, as each element of the array is computed in a single pass through the pattern.
    2. Matching the Pattern in the Text: O(N)
        a) The actual matching phase iterates through the entire text once, and at each step, it compares characters until a match is found or the end of the pattern is reached.

- **Advantages of the KMP Algorithm:**
    - Linear Time Complexity.
    - Avoid Redundant Comparisons.
    - Efficiently Handles Pattern Repeats.
    - No Backtracking in Text Scanning.

- **Use Cases and Applications:**
    - Text Search Engines.
    - Genomic Data Analysis.

- Data Mining and Information Retrieval.
- Compiler Design.
- Natural Language Processing (NLP).

- **Limitations:**

i. **Extra Space Requirement:** The KMP algorithm requires additional space to store the LPS array, which has a length equal to the pattern. In memory-constrained environments, this extra space requirement may be a limitation.

ii. **Not Adaptive to Dynamic Pattern Changes:** The algorithm assumes a static pattern and is not inherently adaptive to dynamic changes in the pattern during runtime. If patterns change frequently, the algorithm may need to be reinitialized.

7. CONCLUSION:

The Knuth-Morris-Pratt (KMP) algorithm is a powerful and efficient solution for string and pattern matching. With its linear time complexity of O(N + M), it excels in handling large texts and patterns. The algorithm's avoidance of redundant comparisons and adaptability to streaming data contribute to its versatility. While the KMP algorithm has preprocessing overhead and additional space requirements, its deterministic behavior and widespread applications in bioinformatics, network security, and natural language processing make it a valuable tool. As a foundational algorithm, the KMP algorithm continues to play a crucial role in addressing fundamental challenges in computer science, providing insights for innovative solutions in the ever-evolving technological landscape.