**King Fahd University of Petroleum & Minerals**

**College of Computer Science and Engineering**

**Information and Computer Science Department**

**ICS 202 – Data Structures**

# String Matching

**Objectives:**

* Study the following sting matching algorithms: bruteforce, Knuth Moriss Prat (KMP), Rabin Karp

**Outcomes:**

After completing this lab, students will know:

* How to implement the following string matching algorithms: bruteforce and Knuth Moriss Prat (KMP)
* How to find the length of the longest non-overlapping suffix that is also a prefix of a given string.
* How to generate the table used in generating the lps array (or the next array) in the KMP algorithm.

**Submissions**: Submit, in your lab section Blackboard, the Java files for each of the lab task is a separate folder, named as: Task01, Task02, Task03, and Task04. Zip all the folders in a **single zip** file named in the format:

KFUPMID#\_FamilyName\_LabSection#.zip

Example:

200000000\_AlGhamdi\_51.zip

**Lab Activity**:

Study the given implementations of Knuth-Morris-Prat algorithm in the file **KMP\_Implementation.txt**

**Lab Task01**:

Write a Java program to find the longest prefix that is also a suffix of a given string. The prefix and suffix must not overlap.

Sample program runs:

|  |
| --- |
| Enter a text pattern: ABABABABAB Longest non-overlapping suffix that is also a prefix is: ABAB its length is: 4 |
| Enter a text pattern: AAAA Longest non-overlapping suffix that is also a prefix is: AA its length is: 2 |
| Enter a text pattern: AAAAA Longest non-overlapping suffix that is also a prefix is: AA its length is: 2 |
| Enter a text pattern: ABCDE No non-overlapping suffix that is also a prefix. |
| Enter a text pattern: abcdefghabcdefgh Longest non-overlapping suffix that is also a prefix is: abcdefgh its length is: 8 |

**Lab Task02:**

Write a Java program to implement the brute force string matching algorithm. The algorithm must detect all occurrences of a pattern. Use overlapping pattern search.

Sample program runs (The numbers represent starting text indexes where a match is found):

|  |
| --- |
| Enter a text string T: aaaaaaaaa Enter a pattern string P: aa aaaaaaaaa aa 0  aaaaaaaaa  aa  1  aaaaaaaaa  aa  2  aaaaaaaaa  aa  3  aaaaaaaaa  aa  4  aaaaaaaaa  aa  5  aaaaaaaaa  aa  6  aaaaaaaaa  aa  7 |
| Enter a text string T: ABABABCDABABK Enter a pattern string P: ABAB  ABABABCDABABK ABAB 0  ABABABCDABABK  ABAB  2  ABABABCDABABK  ABAB  8 |
| Enter a text string T: THIS IS KFUPM Enter a pattern string P: YES  Pattern not found. |

**Lab Task 03a:**

Consider the following Java program:

public class PSuffix{

static void countSamePrefixSuffix(String s, int n)

{

// Stores the prefix string

String prefix = "";

for(int i = 0; i < n - 1; i++)

{

// Add the current character to the prefix string

prefix += s.charAt(i);

// Store the suffix string

String suffix = s.substring(n - 1 - i, n);

System.out.print("Proper prefix: " + prefix + ", Proper suffix: " + suffix);

// Check if both the strings are equal or not

if (prefix.equals(suffix))

{

System.out.print(" \*"+ prefix.length());

}

System.out.println();

}

}

public static void main(String[] args)

{

String S = "ABCAABC";

int N = S.length();

countSamePrefixSuffix(S, N);

}

}

The program generates all **proper** overlapping suffixes and prefixes of the string "ABCAABC". It also highlights suffixes that are also prefixes and their lengths:

Proper prefix: A, Proper suffix: C  
Proper prefix: AB, Proper suffix: BC  
Proper prefix: ABC, Proper suffix: ABC \*3  
Proper prefix: ABCA, Proper suffix: AABC  
Proper prefix: ABCAA, Proper suffix: CAABC  
Proper prefix: ABCAAB, Proper suffix: BCAABC

Modify the program such that it generates all proper overlapping suffixes and prefixes of a given string, similar to the following table in the lecture notes on String matching:

Table

Description automatically generated

The output of your modified program must be in the form:

Enter the pattern: ABCAABC

Substring:   
-------------------------------  
Substring: A  
-------------------------------  
Substring: AB  
Proper prefix: A, Proper suffix: B  
-------------------------------  
Substring: ABC  
Proper prefix: A, Proper suffix: C  
Proper prefix: AB, Proper suffix: BC  
-------------------------------  
Substring: ABCA  
Proper prefix: A, Proper suffix: A \*1  
Proper prefix: AB, Proper suffix: CA  
Proper prefix: ABC, Proper suffix: BCA  
-------------------------------  
Substring: ABCAA  
Proper prefix: A, Proper suffix: A \*1  
Proper prefix: AB, Proper suffix: AA  
Proper prefix: ABC, Proper suffix: CAA  
Proper prefix: ABCA, Proper suffix: BCAA  
-------------------------------  
Substring: ABCAAB  
Proper prefix: A, Proper suffix: B  
Proper prefix: AB, Proper suffix: AB \*2  
Proper prefix: ABC, Proper suffix: AAB  
Proper prefix: ABCA, Proper suffix: CAAB  
Proper prefix: ABCAA, Proper suffix: BCAAB  
-------------------------------  
Substring: ABCAABC  
Proper prefix: A, Proper suffix: C  
Proper prefix: AB, Proper suffix: BC  
Proper prefix: ABC, Proper suffix: ABC \*3  
Proper prefix: ABCA, Proper suffix: AABC  
Proper prefix: ABCAA, Proper suffix: CAABC  
Proper prefix: ABCAAB, Proper suffix: BCAABC  
-------------------------------

**Lab Task 03b:**

Manually, find the nextArray (lps array) for each of the following patterns, by filling the given tables:

1. ABCDE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| j | Pattern [0 ..j-1] | Proper prefixes | Proper Suffixes | next[j] |
| 0 | - | null | null | -1 |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |

The next array is:

1. AAAAA

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| j | Pattern [0 ..j-1] | Proper prefixes | Proper Suffixes | next[j] |
| 0 | - | null | null | -1 |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |

The next array is:

1. ABABAMK

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| j | Pattern [0 ..j-1] | Proper prefixes | Proper Suffixes | next[j] |
| 0 | - | null | null | -1 |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |
| 7 |  |  |  |  |

The next array is:

Include the following **computeNextArray** method in a complete Java program and then verify each of the answers above.

public static int[] computeNextArray(String x){  
 int[] next = new int[x.length() + 1];  
 next[0] = -1;  
 int i = 0, j = -1;  
 while(i < x.length()){  
 while(j == -1 || i < x.length() && (x.charAt(i) == x.charAt(j))){  
 i++;  
 j++;  
 next[i] = j;  
 }  
   
 j = next[j];  
 }  
   
 return next;  
 }

**Lab Task 04**:

Include the methods in the file **KMP\_Implementation.txt** in a complete Java program that prompts for and reads a text string and then a text pattern to search for in the text. It then displays the result of the search.

Sample program runs:

|  |
| --- |
| Enter the text: ABABCABABABCABABCABABCABABKKKABABCABAB Enter the pattern to search for: ABABCABAB Pattern found at these text starting indexes: 0 7 12 17 29 |
| Enter the text: A KFUPM STUDENT Enter the pattern to search for: KFE Pattern not in text. |