

# Implementing the KMP Algorithm

Code the Knuth-Morris-Pratt (KMP) algorithm in C# for pattern searching which pre-processes the pattern to reduce the number of comparisons. Explain how this pre-processing improves the search time compared to the naive approach.

Code: -

```
package com.wipro.assignment;
//JAVA program for implementation
of KMP pattern
//searching algorithm

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

```
        // create lps[] that will  
hold the longest  
        // prefix suffix values for  
pattern
```

```
        int lps[] = new int[M];  
        int j = 0; // index for  
pat[]
```

```
        // Preprocess the pattern  
(calculate lps[]  
// array)
```

```
        computeLPSArray(pat, M,  
lps);
```

```
        int i = 0; // index for  
txt[]
```

```
        while ((N - i) >= (M - j)) {  
            if (pat.charAt(j) ==  
txt.charAt(i)) {  
                j++;  
                i++;  
            }  
            if (j == M) {
```

```

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

    // mismatch after j
matches
    else if (i < N
            && pat.charAt(j) !=
txt.charAt(i)) {
        // Do not match
lps[0..lps[j-1]] characters,
        // they will match
        anyway
        if (j != 0)
            j = lps[j - 1];
        else
            i = i + 1;
    }
}
}
}

```

```

    void computeLPSArray(String
pat, int M, int lps[])
    {
        // length of the previous
longest prefix suffix
        int len = 0;
        int i = 1;
        lps[0] = 0; // lps[0] is
always 0

        // the loop calculates
lps[i] for i = 1 to M-1
        while (i < M) {
            if (pat.charAt(i) ==
pat.charAt(len)) {
                len++;
                lps[i] = len;
                i++;
            }
            else // (pat[i] !=
pat[len])
            {

```

```

        // This is tricky.
Consider the example.
        // AAACAAAA and i = 7.
The idea is similar
        // to search step.
        if (len != 0) {
            len = lps[len - 1];

            // Also, note that
we do not increment
            // i here
        }
        else // if (len == 0)
        {
            lps[i] = len;
            i++;
        }
    }
}

// Driver code
public static void main(String
args[])

```

```

{
    String txt =
"ABABDABACDABABCABAB";
    String pat = "ABABCABAB";
    new
KMP_String_Matching().KMPSearch(pa
t, txt);
}
}

```

**Output: -**

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with packages like `com.wipro.assignment` and `com.wipro`.
- Editor:** Displays the `KMP_String_Matching.java` file. The code includes:
 

```

2 //JAVA program for imple
3 //searching algorithm
4
5 class KMP_String_Matchin
6     void KMPSearch(Strin
7     {
8         int M = pat.leng
9         int N = txt.leng
10
11         // create lps[]
12         // prefix suffi
13         int lps[] = new
14         int j = 0; // ir
15
16         // Preprocess th
17         // array)
18         computeLPSArray(

```
- Console:** Shows the output: `Found pattern at index 10`.
- Taskbar:** Displays the system clock as 11:23:249, date 25-05-2024, and weather 26°C Mostly cloudy.