

KMP

```
package com.wipro;

public class Main {
    // JAVA program for implementation of KMP pattern searching algorithm

    // Method to search for the pattern in the given text using KMP algorithm
    public 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];
            } else if (i < N && pat.charAt(j) != txt.charAt(i)) {
                // Mismatch after j matches
                if (j != 0) {
                    j = lps[j - 1];
                } else {
                    i++;
                }
            }
        }
    }

    // Method to preprocess the pattern and create lps[] array
    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

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

```

    }
}

// Driver code
public static void main(String[] args) {
    String txt = "ABABDABACDABABCABAB";
    String pat = "ABABCABAB";
    new Main().KMPSearch(pat, txt);
}
}

```

```

1  public class Main {
2      // JAVA program for implementation of KMP pattern searching algorithm
3
4      // Method to search for the pattern in the given text using KMP algorithm
5      public void KMPSearch(String pat, String txt) {
6          int M = pat.length();
7          int N = txt.length();
8
9          // Create lps[] that will hold the longest prefix suffix values for pattern
10         int lps[] = new int[M];
11         int j = 0; // Index for pat[]
12
13         // Preprocess the pattern (calculate lps[] array)
14         computeLPSArray(pat, M, lps);
15
16         int i = 0; // Index for txt[]
17         while ((N - i) >= (M - j)) {
18             if (pat.charAt(j) == txt.charAt(i)) {
19                 j++;
20                 i++;
21             }
22             if (j == M) {
23                 System.out.println("Found pattern at index " + (i - j));
24                 j = lps[j - 1];
25             } else if (i < N && pat.charAt(j) != txt.charAt(i)) {
26                 // Mismatch after j matches
27                 if (j != 0) {
28                     j = lps[j - 1];
29                 } else {
30                     i++;
31                 }
32             }
33         }
34     }
35
36     // Compute the longest prefix suffix values for pattern
37     private void computeLPSArray(String pat, int M, int lps[]) {
38         int len = 0; // length of the previous longest prefix suffix
39         int i = 1;
40         while (i < M) {
41             if (pat.charAt(i) == pat.charAt(len)) {
42                 len++;
43                 lps[i] = len;
44                 i++;
45             } else {
46                 // If the characters do not match, then there is no prefix suffix
47                 // valid up to i, set lps[i] to 0, and reset the value of len
48                 lps[i] = 0;
49                 len = 0;
50                 i++;
51             }
52         }
53     }
54 }

```

input

Found pattern at index 10