Chapter 25: Knuth Morris Pratt (KMP) Algorithm

The KMP is a pattern matching algorithm which searches for occurrences of a "word" **W** within a main "text string" **S** by employing the observation that when a mismatch occurs, we have the sufficient information to determine where the next match could begin.We take advantage of this information to avoid matching the characters that we know will anyway match.The worst case complexity for searching a pattern reduces to **O(n)**.

Section 25.1: KMP-Example

Algorithm

This algorithm is a two step process.First we create a auxiliary array lps[] and then use this array for searching the pattern.

Preprocessing :

1. We pre-process the pattern and create an auxiliary array lps[] which is used to skip characters while matching.
2. Here lps[] indicates longest proper prefix which is also suffix.A proper prefix is prefix in which whole string is not included.For example, prefixes of string **ABC** are “ ”, “**A**”, “**AB**” and “**ABC**”. Proper prefixes are “ ”, “**A**” and “**AB**”. Suffixes of the string are “ ”, “**C**”, “**BC**” and “**ABC**”.

Searching

1. We keep matching characters **txt[i]** and **pat[j]** and keep incrementing i and j while **pat[j]** and **txt[i]** keep matching.
2. When we see a mismatch,we know that characters **pat[0..j-1]** match with **txt[i-j+1**…**i-1]**.We also know that lps[j-1] is count of characters of **pat[0**…**j-1]** that are both proper prefix and suffix.From this we can conclude that we do not need to match these **lps[j-1]** characters with **txt[i-j**…**i-1]** because we know that these characters will match anyway.

Implementaion in Java

```
public class KMP {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String str = "abcabdabc";
        String pattern = "abc";
        KMP obj = new KMP();
        System.out.println(obj.patternExistKMP(str.toCharArray(), pattern.toCharArray()));
    }

    public int[] computeLPS(char[] str){
        int lps[] = new int[str.length];

        lps[0] = 0;
        int j = 0;
        for(int i =1;i<str.length;i++){
            if(str[j] == str[i]){
                lps[i] = j+1;
                j++;
```

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

        }

        return lps;
    }

    public boolean patternExistKMP(char[] text,char[] pat){
        int[] lps = computeLPS(pat);
        int i=0,j=0;
        while(i<text.length && j<pat.length){
            if(text[i] == pat[j]){
                i++;
                j++;
            }else{
                if(j!=0){
                    j = lps[j-1];
                }else{
                    i++;
                }
            }
        }

        if(j==pat.length)
            return true;
        return false;
    }

}
```