Knuth Morris Pratt Algorithm(KMP ALgorithm)

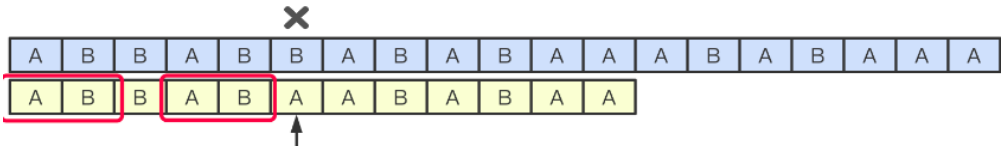
We first define the prefix function of the string - The prefix function of a string s is an array lps of length same as that of s such that lps[i] stores the information about the string s[0.. i]. It stores the length of the maximum prefix that is also the suffix of the substring s[0..i].

For example :

For the pattern "AAAABAA",

lps[] is [0, 1, 2, 0, 1, 2, 3]

lps[0] is 0 by definition. The longest prefix that is also the suffix of string s[0..1] which is AA is 1. (Note that we are only considering the proper prefix and suffix). Similarly, For the whole string AAABAAA it is 3, hence the lps[6] is 3.



Algorithm for Computing the LPS array.

- We compute the prefix values lps[i] in a loop by iterating from 1 to n - 1.
- To calculate the current value lps[i] we set the variable j denoting the length of best suffix for i - 1. So j = lps[i - 1].
- Test if the suffix of length j + 1 is also a prefix by comparing s[j] with s[i]. If they are equal then we assign lps[i] = j + 1 else reduce j = lps[j - 1].
- If we have reached j = 0 we assign lps[i] = 0 and continue to the next iteration .

```
function PrefixArray(s)
        n = s.length;
        //  initialize to all zeroes
        lps = array[n];

        for i from 1 to n - 1
                j = lps[i-1];
                //  update j untill s[i] becomes equal to s[j]
                while j greater than 0 && s[i] no equal to s[j]
                        j = lps[j-1];

                //  if extra character matches increase j
                if s[i] equal to s[j]
                        j += 1;

                //  update lps[i]
```

```
            lps[i] = j;

        //   return the array
        return lps
```

Time Complexity: O(N), where N is the length of the pattern.

Algorithm for searching the pattern.

Now consider a new string S' = pattern + '#' + 'text' where + denotes the concatenation operator. Now, what is the condition that pattern appears at position [i - M + 1... i] in the string text. The lps[i] should be equal to M for the corresponding position of i in S'. Note that lps[i] cannot be larger than M because of the '#' character.

- Create S' = pattern + '#' + 'text'
- Compute the lps array of S'
- For each i from 2*M to M + N check the value of lps[i].
- If it is equal to M then we have found an occurrence at the position i - 2*M in the string text.

```
function StringSearchKMP(text, pattern)
        //   construct the new string
        S' = pattern + '#' + text

        //   compute its prefix array
        lps = PrefixArray(S')
        N = text.length
        M = pattern.length

        for i from 2*M to M + N
            //   longest prefix match is equal to the length of pattern
            if lps[i] == M
                    //   print the corresponding position
                    print the occurrence i - 2*M
        return
```

Time Complexity: O(N + M), where N is the total length of the pattern and M is the length of the pattern we need to search.