Saturday, 23 October 2021 8:39 PM

KMP Algorithm

text → "abcabdyxdac"

pat → "cdyxd"

Find the occurances of pattern in text.

For the given pattern, we calculate its prefix function.

You have a string of length (n). The prefix function for the string is defined as an array (π), of length (n), where π[i] is the length of proper prefix of substring [0.... i] which is also a suffix of substring s[0, ..., i]

Proper prefix: of a string is a prefix that is not equal to the string itself.

String → abcadabc

prefix:
a
ab
abc
abca
abcad
⋮
abcadabc ✗

suffix:
c
bc
abc
dabc
⋮
abcadabc ✗

✗ prefix, ✗ suffix == actual string

longest prefix suffix π

$$\pi[i] = \max_{k=0,\dots,i} \{k : s[0\dots k-1] - s[i-(k-1)\dots i]\}$$

(1) a b c a b c d
π → | 0 | 0 | 0 | 1 | 2 | 3 | 0 |

(2) a a b a a a b
π → | 0 1 0 1 2 2 3 |

(3) a a b c a d a a b c
π → | 0 1 0 0 1 0 1 2 3 0 |

(4) a a a a
π → | 0 1 2 3 |

text → abcxabcdabxabcdabcdabcy

pat → abcdabcy

π → 00001230

j = π[j-1]

```
KMP ( text , pat) {
    lps []
    i=0
    j=0
    while ( i< text.length && j < pat.length) {
        if (txt [i] == pat[j]) {
            i++;
            j++;
        } else {
            if (j==0) {
                i++;
            } else {
                j = lps[j-1];
            }
        }
    }
    if (j == pat.length) {
        return true;
    }
}
```

How to calculate prefix function for a string of length (n)

Brute force approach:

```
vector <int> lps (string s) {
    int n = s.length;
    vector <int> pi (n);
    for ( int i=0; i<n; i++) {
        for ( k=0; k<=i; k++) {
            if (substr (0, k) == substr (i-k+1, k)) {
                pi[i] = k;
        }
    }
    return pi;
}
```

TC → $O(n^3)$

pat → a c a c a b a c a c a b a c a c a c
0 1 2 3 4 5 6 7 8 9 10 11

π → 0 0 1 2 3 0 1 2 3 4 5 6 7 8 9 10 11 4

j π[j-1]

```
int lps ( pattern) {
    int lps [] = new int [pat.length];
    int j=0;
    for (int i=1 ; i< pat.length) {
        if (pat[i] == pat[j])
            lps[i] = j+1;
            i++;
            j++;
        } else {
            if (j != 0) {
                j = lps[j-1];
            } else {
                lps[i] = 0;
                i++;
            }
    }
}
```

Q Given a very large no. (n), count the total no. of ways such that if we divide the numbers in 2 halves, A & B, then no. (A), can be contained by integral division of number B, by some power p of 10. & p ≥ 0.

Eg: 220
output → (1)
Explanation:
2 | 20
A = 2 ✓   B = 20
p = 1
$\frac{20}{10^1} = 2$

Eg. 2202200
ans → (2)
Explanation:
(1) 2202200
A = 2 ✓   B = 20200
p = 5
$= \frac{20200}{10^5} = \lfloor 2.02 \rfloor$

(2) 2202200
A = 220 ✓   B = 2200
p = 1
$\frac{2200}{10^1} = 220$

(1) Irregular division is not allowed.
289 → 29 & 8 → ✗

(2) a & b should not contain leading zeroes

(3) 1 ≤ no of digits in N ≤ 10^5