Saturday, 11 September 2021 1:13 PM

KMP Algorithm

text → "abc ab dy x dac"

pat → "cdy xd"

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 a sub string [0, ... i] which is also a suffix of the substring s[0, ... , i].

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

String : abca dabc

prefix : a, ab, abc, abca, abcad, ..., abcadabc ✗

suffix : c, bc, abc, dabc, ..., abcadabc ✗

prefix == actual string

longest prefix suffix (π)

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

① a b c a b c d

(lps) 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 → abcxabc dab x abc dab c dabcy

pat → abcdabcy

π → 0000120

②

j ↓

j = π[j-1]

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

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

Brute force

```
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);
                   ↓ O(n)    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

3+1 → (4) ✓

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

j ↓

j → π[j-1]

```
int lps ( pattern) {
    int lps[] = new int[pat.length];
    int j= 0;
    for ( 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++;
            }
        }
}
```

O(n)

Q Given a very large number (n), count the total no. of ways such that if we divide the numbers into 2 halves A & B, then no. A can be obtained by integral division of number B by some power p of 10. and 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 and 8 ✗

② a and b should not contain leading zeroes.

② $1 \leq$ no. of digits of $N \leq 10^5$