Saturday, 8 May 2021 8:53 PM

KMP Algorithm

text → "abcabdyxdac"

pat → "cdyxd"

Find the occurances of pattern in the text.

→ For the given pattern, we will 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 the substring s[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.

abc a dabc

prefix → a, ab, abc, abca, abcad, abcada, …, abcadabc ✗✗ prefix == actual string

suffix → c, bc, abc, dabc, adabc, cadabc, …

→ prefix is also a suffix

longest prefix suffix (π)

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

	a	b	c	a	b	c	d
(lps) π (longest prefix suffix)	0	0	0	1	2	3	0

a a b a a a b

π → 0 1 0 1 2 2 3

a a b c a d a a b e

π → 0 1 0 0 0 1 0 1 2 3 0

	a	a	a	a
π →	0	1	2	3

a a   a a a

a a a a

text → abcxabcdabxabcdabcdabcy

pat → abcdabcy

π → (lps)

lps → 2   lps → 3

① First calculate the π array for the pattern

text → abxabcabcaby

pat → abcaby

π → 0 0 0 1 2 0

π[j−1] = 2

π [0 → j−1]

j = 2

π [0 → 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).

O(n³)

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

Observation:

+1, 0, 2

s → abcdabeabf

0 0 0 0 1 2 0 1 2 0

max +1

pat → acacabacacabacacac

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

j → lps[j−1]

acacab

π[j−1] → 5

acaca

2

```
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)