02 June 2024 10:08

KMP algorithm

Pre processing: lps array → longest prefix that is also a suffix.

→ a b c d a b c

prefix: a, ab, abc, abcd, abcda, abcdab, ~~abcdabc~~

suffix: c, bc, abc, dabc, cdabc, bcdabc, ~~abcdabc~~

lps[i] → longest prefix that is also a suffix for string 0 → i.

abcdābeab

P1 : a b c d a b e a b f

lps: 0 0 0 0 1 2 0 1 2 0

P2 : a b c d e a b f a b c

lps: 0 0 0 0 0 1 2 0 1 2 3

P3 : a a b c a d a a b e

lps: 0, 1, 0, 0, 1, 0, 1, 2, 3, 0

P4 : a a a a b a a c d

lps: 0 1 2 3 0 1 2 0 0

KMP eg.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
String:	a	b	a	b	c	a	b	c	a	b	a	b	a	b	d

j	0	1	2	3	4	5
Pat:	a	b	a	b	d	
lps:	0	0	1	2	0	

```
if( string[i]== pat[j+1]){
    i++;
    j++;                    3
```

mismatch:

→ j= lps[j] until j=0

else i++

Pre processing

Compute LPS:

1) int len=0; lps[0]=0, i=1.
2) while(i< pat.length()):
   a) if(pat[i]== pat[len])
      len++
      lps[i]=len;
      i++
   b) else
      i) if(len!=0)
         len= lps[len-1]
      ii) else lps[i]=0
          i++;

i	0	1	2	3	4	5	6	7	8	9	10
	a	b	c	d	e	a	b	f	a	b	c
lps	0	0	0	0	0	1	2	0	1	2	3

i	len
1	0
2	1
3	2
4	3
5	0
6	1
7	2
8	3

KMP Search

(pat, txt)

n = length of txt

i	0	1	2	3	4
	a	a	a	a	b

(pat, txt)

1) m = length of pat, N = length of txt

2) Create lps array and run Compute LPS.

3) i = 0, j = 0

while ((N-i) >= (M-j)) {

if (pat[j] == txt[i])

i++; j++;

if (j == M)

return true;

else if (i < N && pat[j] != txt[i]) {

if (j != 0)

j = lps[j-1];

else

i++

}

a a a a b

a a b

| 0 | 1 | 0 |