Today's Content

- * Introduction to LPS
  - (a) Prefix & Suffix Strings
  - (b) LPS of a string
  - (c) LPS[] of a string
- * Code of LPS
- * Pattern matching by LPS.

Given a string s of length N:

(1) What are prefix strings? substrings starting at idx 0

(2) What are suffix strings? substrings ending at idx n-1

Eg :- s = a b a b (indices 0 1 2 3)

Prefix Strings	Suffix Strings
a	b
ab	ab
aba	bab
abab	abab

LPS of a string: Length of the longest prefix which is also a suffix Note: Except the complete string.

Eg: s = a b c a b (indices 0 1 2 3 4)

Prefix Strings	Suffix Strings	
a	b	
ab	ab	len=2
abc	cab	
abca	bcab	
abcab ✗		

$\quad\quad\quad 0\ 1\ 2\ 3\ 4$

Eg: $s = aaaaa$

Prefix	Suffix	
a	a	
aa	aa	
aaa	aaa	
aaaa	aaaa	len = 4

$S[6] = S_0\ S_1\ S_2\ S_3\ S_4\ S_5$

Prefix	Suffix	iterations
S_0	S_5	1
$S_0 S_1$	$S_4 S_5$	2
$S_0 S_1 S_2$	$S_3 S_4 S_5$	3
$S_0 S_1 S_2 S_3$	$S_2 S_3 S_4 S_5$	4
$S_0 S_1 S_2 S_3 S_4$	$S_1 S_2 S_3 S_4 S_5$	5

$$\frac{5 \times 6}{2} = 15$$

$\sim O(n^2)$

Generalised S_n : $S_0\ S_1\ S_2\ S_3 \ldots\ldots S_{n-2}\ S_{n-1}$

Prefix	Suffix	iterations
S_0	S_{n-1}	1
$S_0\ S_1$	$S_{n-2}\ S_{n-1}$	2
$S_0\ S_1\ S_2$	$S_{n-3}\ S_{n-2}\ S_{n-1}$	3
$\vdots$	$\vdots$	$\vdots$
$S_0 S_1 S_2 \ldots\ldots S_{n-2}$	$S_1\ S_2 \ldots\ldots\ldots S_{n-1}$	$n-1$

$$\frac{(n-1)(n)}{2} = O(n^2)$$

Qn: Given a string s of length N, return lps[].

LPS[i] = LPS value of substring [0 i]

	0	1	2	3	4	5	6
Eg:- s =	a	a	b	a	a	b	a
LPS[7] =	0	1	0	1	2	3	4

LPS[0] = LPS of substring [0 0] : 'a'

Prefix	Suffix
–	–

len = 0

LPS[1] = LPS of substring [0 1] : 'aa'

Prefix	Suffix
a	a

len = 1

LPS[2] = LPS of substring [0 2] : 'aab'

Prefix	Suffix
a	b
aa	ab

len = 0

LPS[3] = LPS of substring [0 3] : 'aaba'

Prefix	Suffix
a	a
aa	ba
aab	aba

len = 1

Qn: Create LPS[] for (TRY YOURSELF)

	0	1	2	3	4	5	6	7	8
s =	a	a	b	a	c	a	a	b	a
LPS[9] =	0	1	0	1	0	1	2	3	4

0 1 2 3 4 5 6 7

a a b a c a a b

Prefix	Suffix
a	b
aa	ab
aab	aab
aaba	caab
aabac	acaab
aabaca	bacaab
aabacaa	abacaab

	0	1	2	3	4	5	6	7	8
s =	a	a	b	a	c	a	a	b	a
LPS[9] =	0	1	0	1	0	1	2	3	4

LPS creation for 1 string takes $O(n^2)$ time

∴ To compute LPS[] ⇒ $O(n^3)$

↓

v. bad TC

optimise: $O(n)$

Calculating LPS[]:

string [0 : i]

Step 1: Given S of length N & assume $LPS[i] = 5$

$$S_N = S_0\ S_1\ S_2\ S_3\ S_4\ S_5 \ldots\ldots S_{i-5}\ S_{i-4}\ S_{i-3}\ S_{i-2}\ S_{i-1}\ S_i.$$

$$LPS[\,] = -\ -\ -\ -\ -\ \ \ -\ -\ -\ -\ -\ 5$$

first 5 chars = last 5 chars

$$LPS[i-1] \geq 4 \begin{cases} S_0\ S_1\ S_2\ S_3\ \cancel{S_4} = S_{i-4}\ S_{i-3}\ S_{i-2}\ S_{i-1}\ \cancel{S_i} \\ \quad LPS[i-1] = 4 \\ \text{Assume} \begin{cases} S_0\ S_1\ S_2\ S_3\ S_4 = S_{i-5}\ S_{i-4}\ S_{i-3}\ S_{i-2}\ S_{i-1} \\ \quad LPS[i-1] = 5 \\ S_0\ S_1\ S_2\ S_3\ S_4\ S_5 = S_{i-6}\ S_{i-5}\ S_{i-4}\ S_{i-3}\ S_{i-2}\ S_{i-1} \\ \quad LPS[i-1] = 6 \end{cases} \end{cases}$$

Assuming $LPS[i] = x$

$$LPS[i-1] \geq x-1$$

$$LPS[i-1] \geq LPS[i] - 1$$

$$\therefore\ LPS[i] \leq 1 + LPS[i-1]$$

<u>At max value</u>

$$LPS[i] = 1 + LPS[i-1]$$

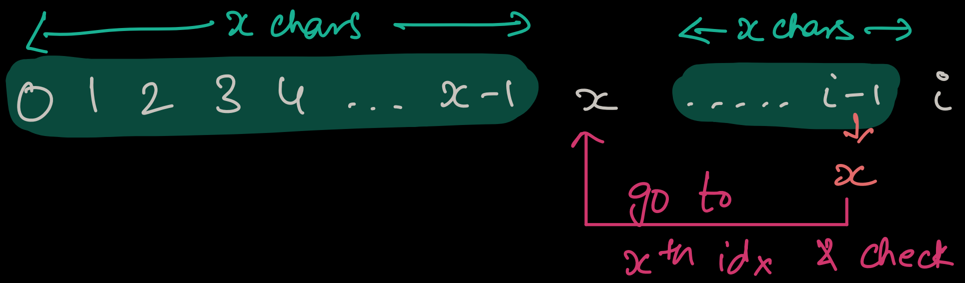
Whenever LPS increases, it will always increase by 1.

Step 2:

Eg 1:

				x					
	0	1	2	3	4	5	6	7	
s =	a	b	a	y	a	b	a	ch	= y
LPS[] =	0	0	1	0	1	2	3	?	4

					x						
	0	1	2	3	4	5	6	7	8	9	
s =	b	c	a	d	c	b	c	a	d	ch	= c
LPS[] =	0	0	0	0	0	1	2	3	4	?	5



// Generalize:

```
Calc lps[i]   // Assume lps[0 i-1] is calculated.
  x = lps[i-1]
  if( s[i] == s[x]) {
  |      lps[i] = x+1
  }
```

Step 3:

				x													
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
s =	c	a	c	y	c	a	c	a	b	c	a	c	y	c	a	c	y
LPS[]	0	0	1	0	1	2	3	2	0	1	2	3	4	5	6	7	4

i = 16 $x = lps[i-1] = 7$

x	str[x] == str[i]	Action?
7	str[7] == str[16] ✗	x = lps[x-1]
3	str[3] == str[16]	lps[i] = x+1 => 4

Step 4:-

			x																					
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
s =	a	b	c	a	b	d	a	b	c	a	b	e	a	b	c	a	b	d	a	b	c	a	b	c
lps[] =	0	0	0	1	2	0	1	2	3	4	5	0	1	2	3	4	5	6	7	8	9	10	11	3

i = 23 $x = lps[i-1] = 11$

x	str[x] == str[i]	Action?
11	str[11] == str[23] ✗	x = lps [x-1]
5	str[5] == str[23] ✗	x = lps [x-1]
2	str[2] == str[23]	lps[i] = x+1

	x							
	0	1	2	3	4	5	6	7
s =	a	b	a	d	a	b	a	c
LPS[]	0	0	1	0	1	2	3	0

$i = 7$ $\quad\quad$ $x = lps[i-1] = 3$

x	str[x] == str[i]	Action?
3	str[3] == str[7]	$x = lps[x-1]$
1	str[1] == str[7]	$x = lps[x-1]$
0	str[0] == str[7]	$x = lps[x-1]$

↳ Out of bound

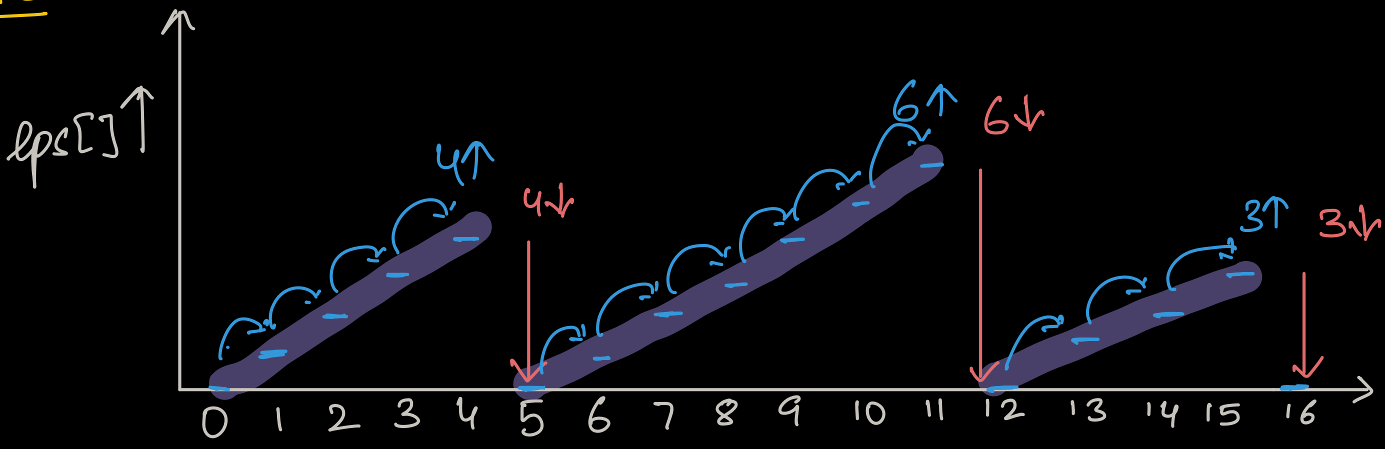
$\therefore$ $lps[i] = 0$

Pseudocode:

KMP Algorithm

```
lps (string s) {
    n = s.length
    lps[n]
    lps[0] = 0
    for(i=1; i<n; i++) {
        // Calc lps[i]
        x = lps[i-1]
        while( s[x] != s[i]) {
            if (x==0) {
                x = -1 , break
            }
            x = lps[x-1]
        }
        lps[i] = x+1
    }
}
```

TC:



Total inc. iterations ⇒ O(n)

Total dec. iterations ⇒ O(n)

} ⇒ O(n)

SC: O(1)

Qn: Search for a given pattern P in text T.

	0	1	2	3	4	5	
T_N =	a	a	b	a	c	d	len = 6
P_M =	a	b	a	c			len = 4

BF idea: Compare every char of T with P & try to match.

TC: O(N*M)

Better idea:

P+T ⇒	a	b	a	c	a	a	b	a	c	d
LPS[] ⇒	0	0	1	0	1	1	2	3	4	0

∴ Pattern found!

Qn: # Count no. of pattern P in given text T.

T → a a a a

P → a a

P+T ⇒	a	a	a	a	a	a
LPS[]:	0	1	2	3	4	5

Half of P & Half of T is matching.

∴ Use a special character "$"

P+$+T ⇒	a	a	$	a	a	a	a
LPS[]:	0	1	0	1	2	2	2

Eg: T_N = a a b a c d

P_M = a b a c

P + $ + T =	a	b	a	c	$	a	a	b	a	c	d
LPS[] :	0	0	1	0	0	1	1	2	3	4	0

Soln to Qn:

① Create P $ T → O(N+M)

② Create LPS[] → O(N+M)

③ Count how many times len(P) occurs in LPS[].

↳ O(N+M)