Agenda

i) Introduction to LPS
   - → prefix and suffix strings
   - → LPS of a string
   - → LPS[] of a string

ii) optimised code of LPS

iii) Pattern matching of a string

⇒ Given a string of N length:

what is prefix strings : substrings starting from 0

what is suffix strings : substrings ending at n-1

	0	1	2	3
S :	a	b	a	b

prefix	suffix
a	b
ab	ab
aba	bab
abab	abab

S : a b a

prefix	suffix
a	a
ab	ba
aba	aba

LPS of a string : longest length of prefix which is also a suffix. { exclude complete string }

s = a b c a b

lps(s) => 2

prefix	suffix
a	b
ab	ab
abc	cab
abca	bcab

s = a a a a

lps(s) => 3

prefix	suffix
a	a
aa	aa
aaa	aaa

s = a b c d a b c

lps(s) => 3

prefix	suffix
a	c
ab	bc
abc	abc
abcd	dabc
abcda	cdabc
abcdab	bcdabc

s = a b c a b

lps(S) => 2

prefix	suffix	
a	b	1 (1)
ab	ab	+ 2 (2)
abc	cab	+ 3 (3)
abca	bcab	+ 4 ⋮ (n-1)

$\sim O(n^2)$

to find lps of a single string

TC => $O(n^2)$

Q. Given a string s, return lps[].

lps[i] => lps value of substring 0 to i

	0	1	2	3	4	5	6
s =	a	a	b	a	a	b	c
lps[]	0	1	0	1	2	3	0

lps[0], s[0,0] => a, ans = 0

lps[1], s[0,1] => aa, ans = 1 (a)

lps[2], s[0,2] => aab, ans = 0

lps[3], s[0,3] => aaba, ans = 1 (a)

lps[4], s[0,4] => aabaa, ans = 2 (aa)

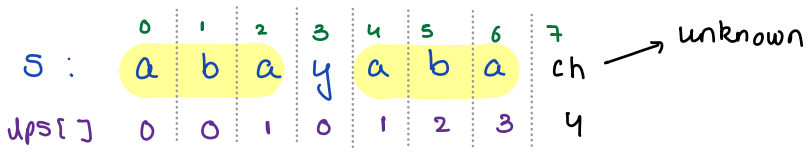
⋮

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

TC to calculate lps of a single string => $O(n^2)$

TC to calculate lps[] => $O(n^3)$ ↓ $O(n)$

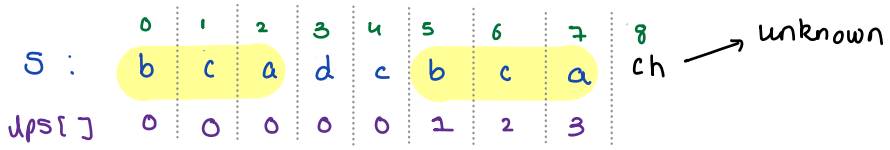
understanding examples



i = 7

x = lps[i-1]
= 3

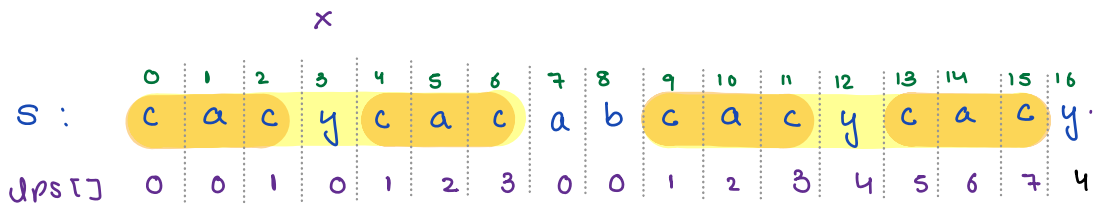
```
if (s.charAt(i) == s.charAt(x)){
    lps[i] = x+1;
}
```



i = 8

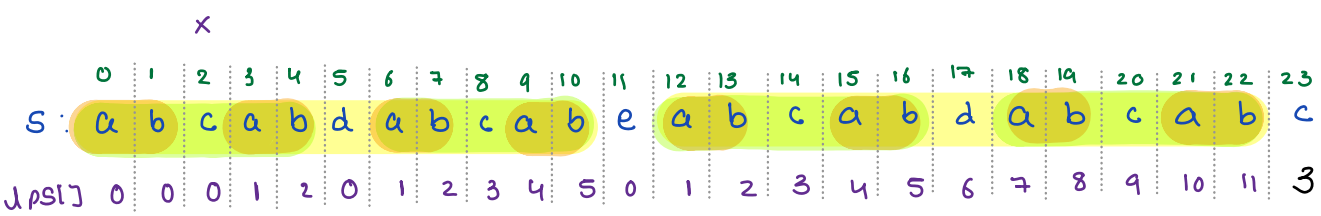
x = lps[i-1]
= lps[7]
= 3

```
if (s.charAt(i) == s.charAt(x)){
    lps[i] = x+1;
}
```



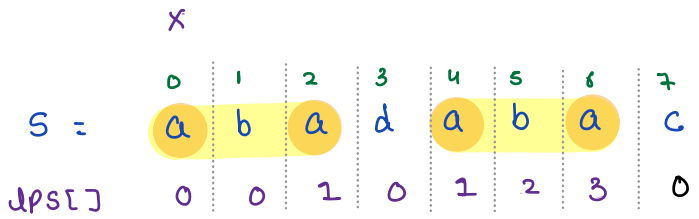
i = 16, x = dps[i-1] = 7

x	s.charAt(i) == s.charAt(x)	
7	s.charAt(16) == s.charAt(7) y ≠ a	x = dps[x-1] x = dps[6] = 3
3	s.charAt(16) = s.charAt(3) y == a	ans ⇒ x+1



i = 23 x = dps[i-1] = 11

x	s.charAt(i) == s.charAt(x)	action x = dps[x-1]
11	s.charAt(23) == s.charAt(11)	No, x = dps[10] = 5
5	s.charAt(23) == s.charAt(5)	No, x = dps[4] = 2
2	s.charAt(23) == s.charAt(2)	Yes ans ⇒ x+1



i = 7, x = lps[i-1] = 3

X	s.charAt(i) == s.charAt(x)	action x = lps[x-1]
3	s.charAt(7) == s.charAt(3)	no, x = lps[2] = 1
1	s.charAt(7) == s.charAt(1)	no, x = lps[0] = 0
0	s.charAt(7) == s.charAt(0)	no, if(x == 0) { ans → 0 }

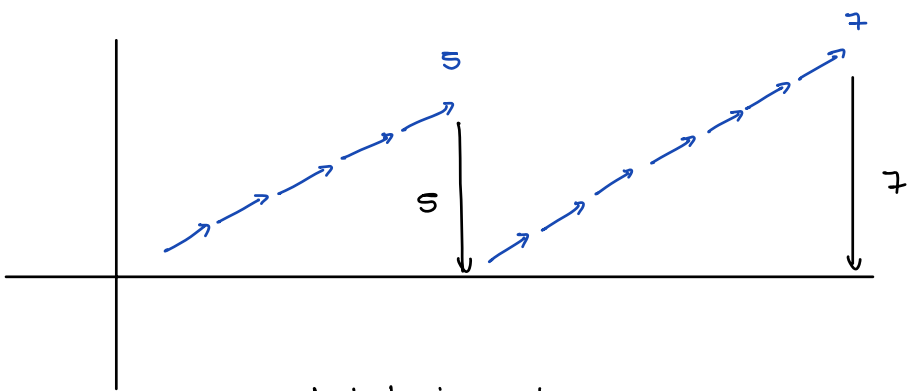
```
int[] lps(String[] str) {
    int n = str.length();
    int[] dps = new int[n];
    dps[0] = 0;
    for (int i=1; i<n; i++) {
        int x = dps[i-1];
        while (str.charAt(i) != s.charAt(x)) {
            if (x == 0) {
                x = -1;
                break;
            }
            x = dps[x-1];
        }
        dps[i] = x+1;
    }
    return dps;
}
```

Dry run

	0	1	2	3	4	5	6	7
s :	a	b	a	y	a	b	a	b
lps[]	0	0	1	0	1	2	3	2

```
for (int i=1; i<n; i++) {
    int x = lps[i-1];
    while (str.charAt(i) != s.charAt(x)) {
        if( x == 0) {
            x = -1;
            break;
        }
        x = lps[x-1];
    }
    lps[i] = x+1;
}
```

i	x	
1	0 -1	lps[1] = 0
2	0	lps[2] = 1
3	~~0~~ -1	lps[3] = 0
4	0	lps[4] = 1
5	1	lps[5] = 2
6	2	lps[6] = 3
7	~~3~~ 1	lps[7] = 2



totot inc steps → n

total dec steps → n

itr : 2n

TC : O(n)

Q. Count occurrences of pattern P in Text T.

eg1
T = a a b a c d
P = a b a c

ans = 1

eg2
T = a b a d c a b a b a e
P = a b a

ans = 3

length of T => n

length of P => m

i) Brute force idea : match P with all substrings of length m in T. TC : O(n*m)

ii) Expected TC : O(n+m)

T = a b a d c a b a b a e

P = a b a

Str => P + "#" + T

str => a b a # a b a d c a b a b a e

lps[] 0 0 1 0 1 2 3 0 0 1 2 3 2 3 0

```
if( lps[i] == P.length() ) {
        ans++;
}
```

KMP Algo for pattern maching

↓

knuth morris pratt

T = a b a a b a b a

P = b a

Str : P + "#" + T

Str : b a # a b a a b a b a

lps[] 0 0 0 0 1 2 0 1 2 1 2

```
int patternMatching(String T, String P) {
    String str = P + "#" + T;
    int ans = 0;
    int[] lps = LPS(str);

    for(int i=0; i<lps.length(); i++) {
        if(lps[i] == P.length()) {
            ans++;
        }
    }

    return ans;
}
```

TC : O(n+m)

Why putting "#" in b/w of P and T is important.

T = a a a a

P = a a

If we don't put "#"

str = P + T

str = a a a a a a

lps => 0 1 2 3 4 5

If we put "#"

str = P + "#" + T

str = a a # a a a a

lps => 0 1 0 1 2 2 2

ans = 3 ✓