Q. Given a string. Find the length of largest prefix substring which is also a a suffix substring.

Ex s = "abcab"

Ps	Ss
a	b
ab	ab
abc	cab
abca	bcab.

⇒ 2

Ex "abcdabc"

Ps	Ss
a	c
ab	bc
abc	abc
abcd	dabc
abcda	cdabc
abcdab	bcdabc

⇒ 3

LPS : Largest (length) Prefix which is also a suffix

s = 'a'
⇒ LPS = 0

s = 'aaaa'
LPS = 3.

Brute force

$S : s_0 s_1 s_2 s_3 s_4 s_5$

PS	SS	
$s_0 s_1 s_2 s_3 s_4$	$s_1 s_2 s_3 s_4 s_5$	⇒ $N-1$
$s_0 s_1 s_2 s_3$	$s_2 s_3 s_4 s_5$	⇒ $N-2$
$s_0 s_1 s_2$	$s_3 s_4 s_5$	⇒ $N-3$
$s_0 s_1$	$s_4 s_5$	⋮
s_0	s_5	1

# of iterations $= 1 + 2 + 3 + ---- + N-1$

$$= \frac{N(N-1)}{2}$$

TC : $O(N^2)$

HW :- Implement this.

* Given a string of length N, return the LPS Array.

LPS[i] : length of largest prefix which is also a suffix from index 0 to i.

Ex

	0	1	2	3	4	5	6
S :	a	a	b	a	a	b	a
LPS[]:	0	1	0	1	2	3	4

Quiz

	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

# for every substring starting at i=0 : O(N)

// Calculate the LPS. → O(N^2)

TC : O(N^3)

# Text] → find the no. of occurrences
Pattern] of (P) in (T).

T : aabace ⇒ (M)

P : abac ⇒ (N)

delimeter :- any random character.

	0	1	2	3	4	5	6	7	8	9	10
P + '$' + T :	a	b	a	c	$	a	a	b	a	c	e
LPS[]:	0	0	1	0	0	1	1	2	3	4	0

T : a a b a c a b a c

P : a b a c

P + '$' + T : abac$aabacabac

LPS[]: 0 0 1 0 0 1 1 2 3 4 1 2 3 4

⇒ 2 occurrences of P in T.

Steps :-

1) Create LPS Array. ⇒ $O((M+N)^3)$

2) if (LPS[i] == M) ⇒ $O(N)$
   count++

TC : $O((M+N)^3)$

* KMP (Knuth Morris Pratt) Algo.

↳ $O(M+N)$: TC

Quiz

S: abayaba

LPS[]: 0 0 1 0 1 2 3

Quiz

S: cacycaca

LPS[]: 0 0 1 0 1 2 3 2

$S: \; S_0 \, S_1 \, S_2 \, S_3 \, S_4 \, S_5 \cdots\cdots S_{i-5} \, S_{i-4} \, S_{i-3} \, S_{i-2} \, S_{i-1} \, \underset{\overset{\uparrow}{5}}{S_i} \cdots\cdots S_{N-1}$

$LPS[i] = \boxed{5}$

$S_0 S_1 S_2 S_3 S_4 \;=\; S_{i-4} S_{i-3} S_{i-2} S_{i-1} S_i$

min value of $LPS[i-1] = 4.$

$$\boxed{LPS[i] \geq 4}$$

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

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

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

Quiz

$(LPS[i])_{max} = LPS[i-1] + 1$

Quiz

$LPS[i] = 5$

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

$5 - 1 <= LPS[i-1]$

$$\boxed{4 <= LPS[i-1]}$$

Ex

	0	1	2	3	4	5	6	7	
S:	a	b	a	y	a	b	a	?	→ S[i]
LPS:	0	0	1	0	1	2	3	4	
							i-1	i	

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

$LPS[i] <= 4$

If $LPS[i] = 4 \Rightarrow S[i] = y \rightarrow S[LPS[i-1]]$

Ex

	0	1	2	3	4	5	6	7	8	9	
S	b	c	a	d	c	b	c	a	d	?	→ S[i]
LPS[]	0	0	0	0	0	1	2	3	4		
									i-1	i	

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

$LPS[i] <= 4 + 1$

$LPS[i] <= 5$

```
if ( S[i] == S[LPS[i-1]] ){
        LPS[i] = LPS[i-1] +1;
}
```

Ex

	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	
							⇑									i-1	i

i=16

LPS[i] <= 8 .

S[i] != 'a'

LPS[15] = 7

S[0-6] = S[9-15]

$S_0 S_1 S_2 S_3 S_4 S_5 S_6 = S_9 S_{10} S_{11} S_{12} S_{13} S_{14} S_{15}$

$S_0 S_1 S_2 = S_4 S_5 S_6 = S_9 S_{10} S_1 = S_{13} S_{14} S_{15}$

if (S[3] == S[i]) ⇒ LPS[i] = 3+1 = 4

i = 16

x = LPS[i-1] = 7

x	S[i] == S[x]	x new
7	S[16] == S[7] ✗ y a	⇒ x = LPS[x-1] = LPS[6] x = 3

3	S[16] == S[3] ✓
	y y
	LPS[i] = x+1
	LPS[16] = 3+1 = 4.	

Ex

					↓																			
	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	?
																							↑ i-1	↑ i

i = 23

x = LPS[i-1] = LPS[22] = 11

x	S[i] == S[x]	x new.
11	S[23] == S[11] ✗ c != e	⇒ x = LPS[x-1] = 5
5	S[23] == S[5] ✗ c != d	⇒ x = LPS[x-1] = LPS[4] x = 2
2	S[23] == S[2] ✓ c == c LPS[23] = x+1 = 3	

LPS[23] = 3

Code

```
LPS[0] = 0
for( i= 1; i< N; 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: O(N)

At max How many increasing steps ⇒ N

At max How many decreasing steps ⇒ N

TC: O(N)

S: a b c d e f g

LPS: 0 0 0 0 0 0 0

O(N)

i=1
↓

S: a a a a a a a a a

LPS: 0 1 2 3 4 5 6 7 8

O(N)

Pattern Matching

TC: O(M+N) {Using KMP}

SC: O(M+N)

Q: Given a string, count the no. of rotations that gives us the original string.

S: abcd
bcda
cdab
dabc
abcd

⇒ 1

S: abab
baba
abab
baba
abab

⇒ 2

S: aaa
aaa
aaa
aaa

⇒ 3

S = abacd
bacda
acdab
cdaba
dabac
abacd

⇒ 1

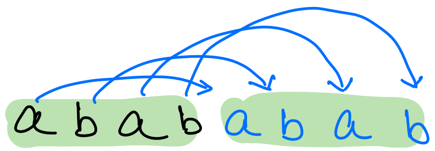
Brute force

Rotate the string N times & in every rotation Check if it is equal to original string.

TC : $O(N^2)$

Approach 2 :-

S : a b a b



1) S : abab ⇒ Pattern

S+S : abababab ⇒ Text.

2) S : abcd

S+S : abcdabcd ⇒ ①

```
String S;
Text ⇒ S+S
Pattern ⇒ S
P + '$' + T : S$(S+S)
// Create LPS.

LPS[i] == length(Pattern){
        Count++;
}
```

S : abab ⇒ Pattern

T : S+S : abababab ⇒ Text

P : S : abab

P + '$' + T :- abab$abababab.

LPS: 0 0 1 2 0 1 2 3 4 3 4 3 4

return count − 1;

——— * ———