Today's Content: ⟶ {Pen/Paper}

- Prefix & Suffix Strings
- LPS of a given String
- LPS[] of a given String

  Problems based on LPS[]

  - Pattern matching by LPS
  - Cyclic rotations

- Calculating LPS[] in optimized way.

Given a String S of N size :

Prefix Strings : Substrings starting at index 0

Suffix Strings : Substrings ending at index N-1

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

Prefix String	Suffix Strings
a	b
a b	a b
a b a	b a b
a b a b	a b a b

→

LPS of a String : length of longest prefix which is also suffix string

Note: Should not consider full string!

Eg1: S = a b c a b (indices 0 1 2 3 4)

Prefix Strings	Suffix Strings
a	b
a b	a b : ans: 2
a b c	c a b
a b c a	b c a b
a b c a b	a b c a b

↳ Neglect

S = a (index 0) ⇒ LPS(S) = 0

Prefix — Suffix

a — a ⇒ {Neglect}

S = a a a a a (indices 0 1 2 3 4)

Prefix	Suffix
a	a
a a	a a
a a a	a a a
a a a a	a a a a : ans: 4
a a a a a	a a a a a

↳ Neglect

TC to calculate LPS? ⇒ O(N²)

$S_6 = S_0 S_1 S_2 S_3 S_4 S_5$

Prefix	Suffix
S_0	S_5
$S_0 S_1$	$S_4 S_5$
$S_0 S_1 S_2$	$S_3 S_4 S_5$
$S_0 S_1 S_2 S_3$	$S_2 S_3 S_4 S_5$
$S_0 S_1 S_2 S_3 S_4$	$S_1 S_2 S_3 S_4 S_5$
$S_0 S_1 S_2 S_3 S_4 S_5$	$S_0 S_1 S_2 S_3 S_4 S_5$

Iterations

1
2.
3.
4.
5

} * (5)(6)/2 → 15.

Increased Str - N

TC: 1 + 2 + 3 + ... N-1

⇒ Sum of (N-1) Natural

⇒ (N-1)(N)/2

⇒ TC: O(N²)

Given a string S of length (N)

LPS[ ] = LPS value of all Strings starting at index 0

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

Expected TC
- N * {O(N^2)} = TC : O(N^3)
- → optimize → TC : O(N) } → Linear time

Ex:

	0	1	2	3	4	5	6
S =	a	a	b	a	a	b	a
LPS[7] =	0	1	0	1	2	3	4

LPS[0] = LPS substring [0 0]

= a

Prefix | Suffix

a | a → {we neglect full String}

LPS[1] = LPS substring [0 1]

= a a

Pref | Suff

a — a = 1

aa | aa → neglect

LPS[2] = LPS substring [0 2]

= a a b

Prefix	Suffix
a	b *
a a	a b *
a a b	a a b

LPS[3] = LPS substring [0 3]

a a b a

Pref	Suff
a	a : 1
a a	b a
a a b	a b a
a a b a	a a b a

LPS[5] = LPS substring [0 5]

a a b a a b

p	s
a	b
a a	a b
a a b	a a b : 3
a a b a	b a a b
a a b a a	a b a a b

a a b a a b a

p	s
a	a
a a	b a
a a b	a b a
a a b a	a a b a ✓
a a b a a	b a a b a
a a b a a b	a b a a b a

Assume: ~~lps[N] → O(N)~~ ?

Q) Search for a given Pattern P in Text T using lps[]

Eg:

Text (N): a a b a c e

Pattern (M): a b a c

delimiter : A character which is not present in Text & Pattern

Separator :

S : { 0 1 2 3 | 4 | 5 6 7 8 9 10 }
{ a b a c | $ | a a b a c e }

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

Obs → 1) S = (Pattern) $ (Text), S.size() = (N+M+1)

2) Calculate lps[] of S ⟶ TC : (N+M)

SC : (N+M)

3) If(lps[i] == M)
Pattern present

→ Find no of occurences of Pattern P in Text T

Text : a b c d a b c

Pattern : a b c

S = P $ T

= a b c $ a b c d a b c

LPS[] = 0 0 0 0 1 2 3 0 1 2 3

Ans = 2 → how many occurences

Ex:

Patt : a a

Text : a a a a

with $

S = a a $ a a a a

lps[] : 0 1 0 1 2 2 2

cnt = 3

occurence = 3

without $

S = a a a a a a

lps[] = 0 1 2 3 4 5

{ We are not getting cnt occurences }

Q2) Given a binary string S (0/1)

Find no:of end-start cyclic rotations are same a given String

Ex: 1 0 1 0

1st rotation : 0 1 0 1

2nd rotat : 1 0 1 0 ⇒ 1

3rd rotat : 0 1 0 1

4th rot : 1 0 1 0 ⇒ 2

2 rotations:

Ex: a b c d

1st : d a b c

2nd : c d a b

3rd : b c d a

4th : a b c d

Start → end

b c d a

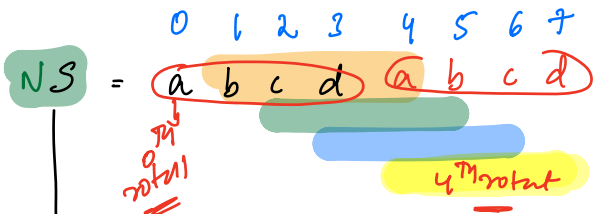
c d a b

d a b c

a b c d

Idea: S = a b c d

NS = Append S to itself



All rotations of S are present in NS as substrings

b c d a
c d a b
d a b c
a b c d

→ In NS itself we are getting all rotations of S

We have to calculate how many occurrences of S are coming in NS

Pattern = S

Text = NS

Eg: 1 0 1 0

P = 1 0 1 0 T = 1 0 1 0 1 0 1 0

S = P $ T

	0	1	2	3	4	5	6	7	8	9	10	11	12
	1	0	1	0	$	1	0	1	0	1	0	1	0
LPS[]	0	0	1	2	0	1	2	3	4	3	4	3	4

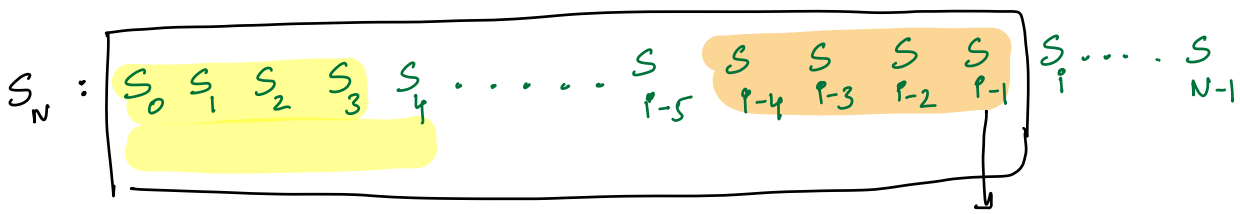
10:53 PM

$Scut = 3 - 1$ for counting same rotation 2 times

Calculating LPS[] ?

string → [0 i]

obs1) : Given S of length N & assume Lps[i] = x



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

$S_0 S_1 S_2 S_3$ = $S_{i-4} S_{i-3} S_{i-2} S_{i-1}$

Lps[i-1] >= 4 ✓ ?

Is it possible : $S_0 S_1 S_2 S_3 S_4$ = $S_{i-5} S_{i-4} S_{i-3} S_{i-2} S_{i-1}$

Is it possib : $S_0 S_1 S_2 S_3 S_4 S_5$ = $S_{i-6} S_{i-5} S_{i-4} S_{i-3} S_{i-2} S_{i-1}$

Assuming

$$\begin{cases} Lps[i] = x \\ Lps[i-1] >= (x-1) \\ Lps[i-1] >= Lps[i]-1 \\ 1 + Lps[i-1] >= Lps[i] \end{cases}$$

obs $\quad Lps[i] <= Lps[i-1] + 1$

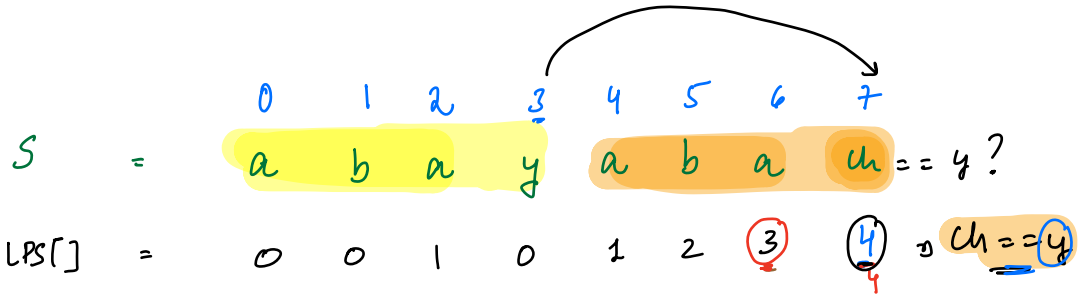
our Lps[] value can at max increase by 1

Ex:

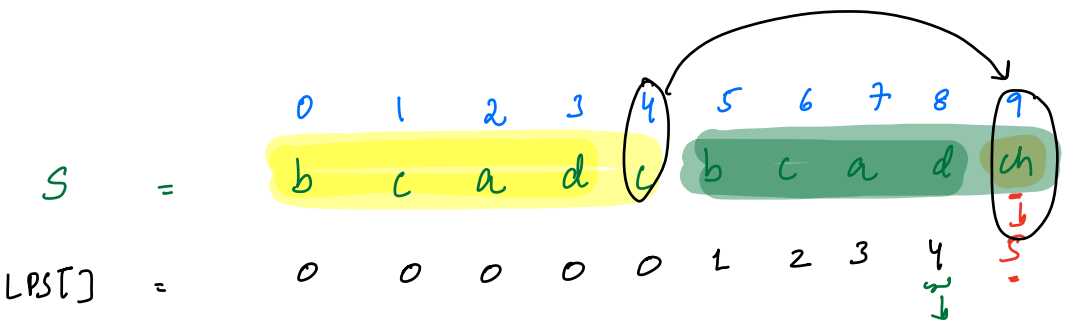
0	1	2	3	4	5	6	7
c	a	c	y	c	a	c	a

Step 2:

Eg1:



Eg2:



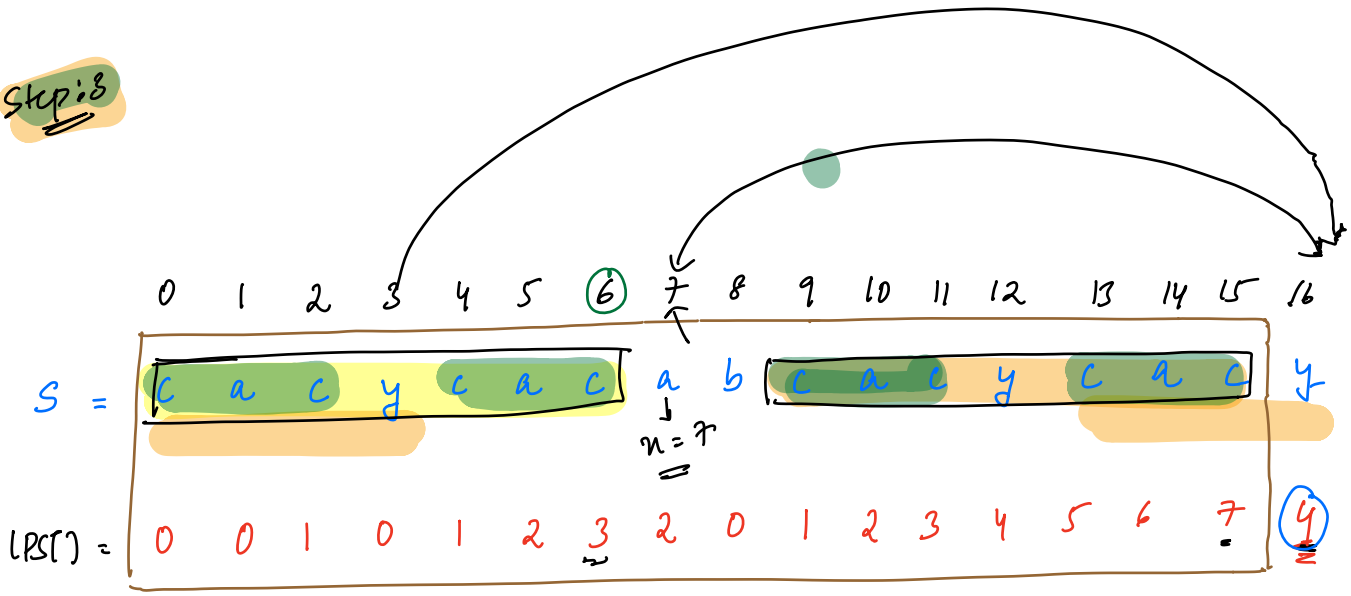
// **Generalize:**

Cal lps[i] = ?

Say x = lps[i-1]

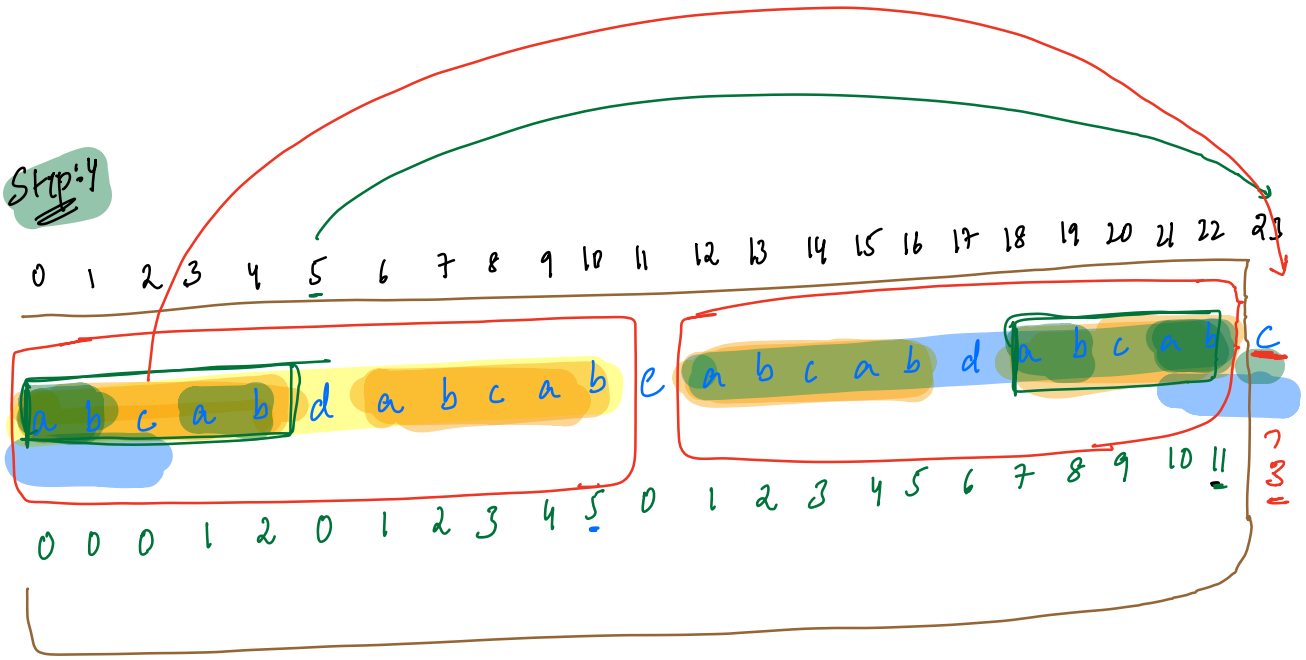
```
if ( s[i] == s[x] ) {
    lps[i] = x+1
}
```

Step:8



i = 16, n = lps[i-1]

n	if (str[i] == str[n]	Matching
7	str[16] == str[7]	✗ : n = lps[n-1] n = lps(6) = 3
3	str[16] == str[3]	✓ : lps[i] = n+1



$P = 23$, $n = lps[P-1] = n = 11$

n	if (str[i] == str[n]	
11	str[23] == str[11]	* $n = lps[n-1]$ $n = lps[10] = 5$
5	str[23] == str[5]	* $n = lps[n-1]$ $n = lps[4] = 2$
2	str[23] == str[2]	✓ $lps[P] = n+1 = 3$

Note : Doing pattern Matching with LPS[] → KMP

Knuth - Morris Pratt

Pseudocode // for a given S_N, we need to calculate lps[N]

```
lps[0] = 0;

for(i=1; i<N; i++) {

    // Say we need to get lps[i]

    x = lps[i-1]

    while( str[i] != str[x] ) {

        if(x == 0) { x = -1; break }

        x = lps[x-1]

    }

    lps[i] = x+1

}
```

TC: Your lps[] value can at max increase by 1 : +1 (Pivot)

Your lps[] can decrease, it can come to : 0

→ We are calculating lps[] N values

At max how many increasing steps : N

At max how decrease steps : N

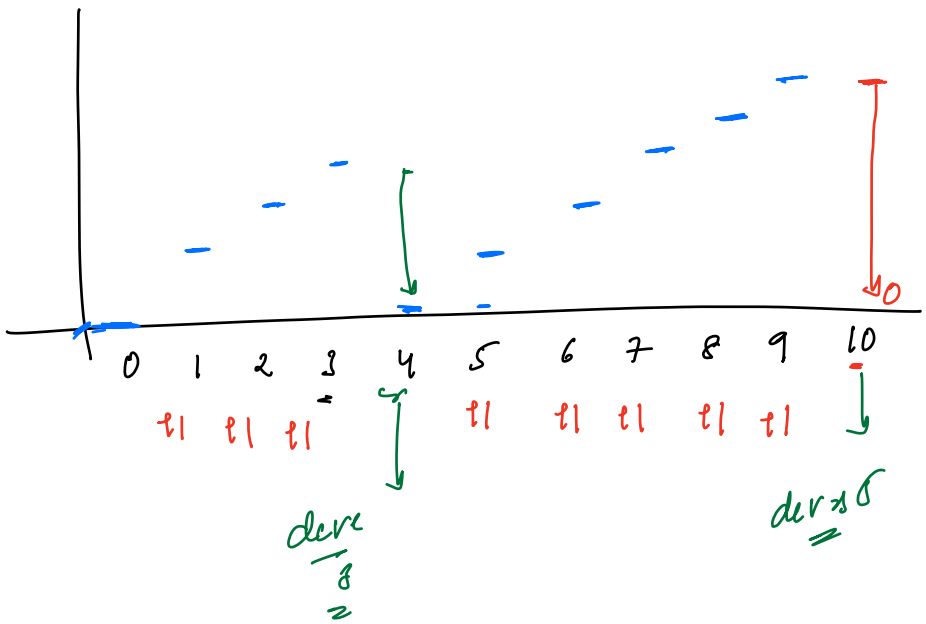
2N

SC : O(N)

lps[] of size N

TC : O(N)

Steps:



→ inc → 8
→ dec → 8

At most → N
At min → N
→ O(N)