Todays Content:

→ Prefix & Suffix Strings

→ LPS of a given string

→ LPS[] of a given string

Problems based on LPS[]:

→ Pattern Matching by LPS[]

→ Cyclic Rotations

→ Min characters to be added at start to make string palindrome

Q) S_1 & S_2 are given of N length

if($S_1 == S_2$) TC: O(N)

S_1: a b a c

↕ ↕ ↕ ↕ not same

S_2: a b a d

Q]: Given Patten (P) and Text (T) check, if pattern is present as a substring in T, len(T) >= len(P)

P_K = a c d a (0 1 2 3)

T_N = b c a c d a (0 1 2 3 4 5)

Dry run: Pat

[0-3] = {b c a c} = a c d a not match

[1-4] = {c a c d} = a c d a not match

[2-5] = {a c d a} = a c d a match

Ideal: For every substring of len k in Text compare with Pattern?

→ // no: of substring of len k in Text

TC: (N-k+1) * k → // time taken to compare 2 strings of len=k

↳ if $k \approx N/2$

TC: $(N - N/2 + 1) * (N/2) = [N/2+1][N/2] \approx O(N^2)$

Note:

Insert a Single String S_N in HashSet<String> TC: O(N) ?

Given a String S of N size:

Prefix Strings : All substrings starting at 0^{th} index

Suffix Strings : All substrings ending at last index [N-1]

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

Prefix Strings	Suffix Strings
a	a
a b	c a
a b c	b c a
a b c a	a b c a

Ex2: S = d e a (indices 0 1 2)

Suffix
a
e a
d e a

LPS of a String : length of longest Prefix String = Suffix String

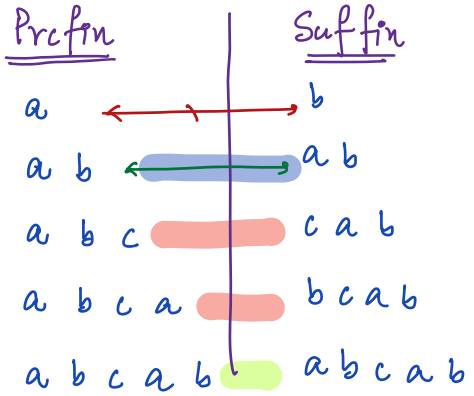
↓

value =

Note: Neglect complete while calculating

0 1 2 3 4

Ex1: S = a b c a b lps = 2



Note: Neglect complete string

Ex2: S = a (0) lps = 0

Prefix	Suffix
a	a

0 1 2 3 4

Ex3: S = a a a a a LPS = 4

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

how to calculate it LPS?

$S = S_0\ S_1\ S_2\ S_3\ S_4$

Prefix	Suffix	Iterations
S_0	S_4 →	1 ite
$S_0 S_1$	$S_3 S_4$ →	2 ite
$S_0 S_1 S_2$	$S_2 S_3 S_4$ →	3 ite
$S_0 S_1 S_2 S_3$	$S_1 S_2 S_3 S_4$ →	4 ite

Total ite = 10

$S_N = S_0\ S_1\ S_2\ S_3 \ldots S_{N-3}\ S_{N-2}\ S_{N-1}$

Prefix:	Suffix	Iterations
$S[0,0]$	$= S[N-1, N-1]$	1 ite
$S[0,1]$	$= S[N-2, N-1]$	2 ite
$S[0,2]$	$= S[N-3, N-1]$	3 ite
$S[0,3]$	$= S[N-4, N-1]$	4 ite
⋮	⋮	⋮
$S[0, N-2]$	$= S[1, N-1]$	N-1 ite

Total iter = $1+2+3+\ldots N-1$

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

Given S_N return LPS[N]

LPS[i] = [lps value of substring S[0, i]]

Obs: S_N, lps[N]
- We are calculating lps for N strings
- TC: N × O(N²) ≈ O(N³)
- Time taken To Calculate lps[] of S_N = TC: O(N)

	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[i] = lps of substring S[0 - i]

String	Lps Value
LPS[0] = lps of substring S[0 - 0] = a	0
LPS[1] = lps of substring S[0 - 1] = a a	1
LPS[2] = lps of substring S[0 - 2] = a a b	0
LPS[3] = lps of substring S[0 - 3] = a a b a	1
LPS[4] = lps of substring S[0 - 4] = a a b a a	2
LPS[5] = lps of substring S[0 - 5] = a a b a a b	3
LPS[6] = lps of substring S[0 - 6] = a a b a a b a	4

Q1) Search for a given Pattern P in Text T

Note: Both Contains only lower case alphabets (a b .. z)

Eg1:

	0	1	2	3	4	5
T_N =	a	a	b	a	c	d
P_k =	a	b	a	c		

Idea: Since lps will match prefix & suffix, append Pattern at start of Text & Create a new string

C_{N+k} =

	0	1	2	3	4	5	6	7	8	9
	a	b	a	c	a	a	b	a	c	d

Pattern == Text[5-8]

lps[10] = 0 0 1 0 1 1 2 3 4 0

= k length of Pattern

Steps: Given P_k, T_N : TC: O(N+k) SC: O(N+k)

Step1: Created a new Concatenated String — TC: O(N+k) SC: O(N+k)

$C_{N+k} = P_k + "@" + T_N$

↳ "@" added to differentiate pattern & Text

Step2: Create lps[N+k] for String C → O(N+k) O(N+k)

Step3: Search for k in lps[], → O(N+k) O(1)

: if lps[i] == k Pattern present

if k is not lps[] Pattern not found

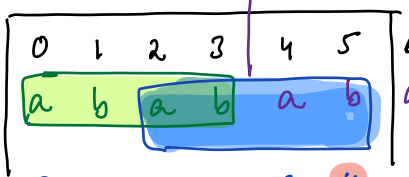
Edge Case:

P_4 = a b a b

T_5 = a b a c a

C_9 = P_4 + T_5

issue: We are not able Separate Pattern & Text



lps[9] = 0 0 1 2 3 4 _ _ _

↳ 4 == k present present * wrong

Idea: To Separate Pattern & Text add a separator = @

↳ Any character not present in both strings

P_4 = a b a b

T_5 = a b a c a

C_{10} = a b a b @ a b a c a

(0 1 2 3 4 5 6 7 8 9)

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

↳ In lps[], {k = 4} not present: pattern not present

Q2) # Count no: of Substrings of T are same as Pattern (P)

P: a a T_N: a a a a (0 1 2 3)
k

	0	1	2	3	4	5	6
C_{N+k+1} =	a	a	@	a	a	a	a

lps[] = 0 1 0 1 2 2 2 ⇒ Count of 2 = 3

Idea: find no: of k in lps[], no: of substrings of T = = P

TC: O(N+k) SC: O(N+k)

Q3) Given a Binary String S_N

Find no: of start → end rotations of S will give same string S

Note: At max N rotations are possible

Ex: S = 1 0 1 0 (0 1 2 3) ans = 2

S_1 = 0 1 0 1 == S

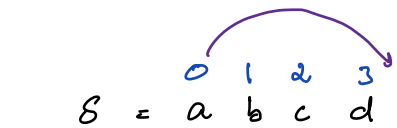
S_2 = 1 0 1 0 == S +1

S_3 = 0 1 0 1 == S

S_4 = 1 0 1 0 == S +1

obs1: N^{th} rotation of given string will be original string, Atleast ans will increase by +1

// Rotation explanation



After1 = b c d a == S

After2 = c d a b == S

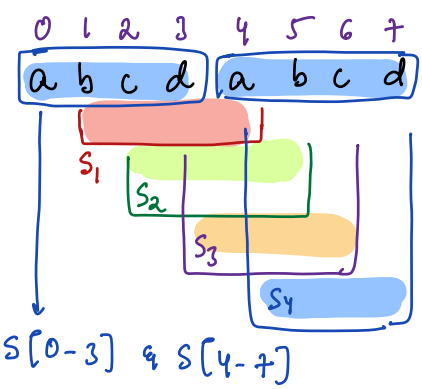
After3 = d a b c == S

After4 = a b c d == S

Obs: After every rotation we are comparing it with S,

P = S T = S + S // will contain all rotations

C = P @ T



C = S @ SS

↳ calulate freq of S in C

TC : ∃N → O(N)

SC : ∃N → O(N)

Obs: In general we only need to have one abcd in Text but we have 2, because of this in our ans, we are getting extra 2, in your final ans remove extra, ans-1

Ex:

S = 1010

P = 1010 T = 1010 1010

	0	1	2	3	4	5	6	7	8	9	10	11	12
C =	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

↳ cnt of k = 3, return cnt-1 = 2

-1 because text contains an extra $

4Q) Given a String S_N, min characters to be added at start of String to make entire string palindrome

{Sunday doubts Session}

Eg1: S = a d a c d ans =

S = a b b a a c d ans =

S = a b c b a d e f ans =

S = a a a e a a a g h ans =

S = a a d a b a ans =

S = a b c ans =

ans =

=

Idea: S = a b a c d

0 1 2 3 4

Edge Case : S = a a a

Steps :

//Lps[] : How to optimize this

For a given S_N, to calculate Lps[N]

```
Lps[0] = 0 ✓
i = 1; i < N; i++) {   TC → O(N) ✓
    //Say we good Lps[i] ✓
    x = Lps[i-1]
    while (s[i] != s[x])
        if (x == 0) { x = -1; break}
        x = Lps[x-1]
    }
    Lps[i] = x + 1
}
```

Sunday Problem Solving Sess : 10am :⟶ 12:30pm