Todays Content:

a) Calculate lps[]
b) Pattern Matching lps[]
c) High Focused Class

Given a String S of N characters

Prefix Strings: Substrings starting at index 0

Suffix Strings: Substrings ending at index N-1

```
        0 1 2 3
Ex: S = a b c a
```

Prefix Strings:	Suffix Strings:
S[0 0] = a	S[3 3] = a
S[0 1] = ab	S[2 3] = ca
S[0 2] = abc	S[1 3] = bca
S[0 3] = abca	S[0 3] = abca

```
        0 1 2
Ex: S = a e d
```

Prefix String	Suffix String
S[0 0] = a	S[2 2] = d
S[0 1] = ae	S[1 2] = ed
S[0 2] = aed	S[0 2] = aed

LPS: lps of a string defined as
length of longest prefix, which is also suffix string.

Note: Neglect complete string, in Prefix & Suffix?

Reason: If we consider comp. string lps value will always be = N.

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

Prefix	Suffix
S[0 0] = a	S[4 4] = b
S[0 1] = ab	S[3 4] = ab } len = 2
S[0 2] = abc	S[2 4] = cab
S[0 3] = abca	S[1 4] = bcab
~~S[0 4] = abcab~~	~~S[0 4] = abcab~~ } len = 5

Ex2: S = a a b a a b lps = 3

Prefix:	Suffix:
S[0 0] = a	S[5 5] = b
S[0 1] = aa	S[4 5] = ab
S[0 2] = aab	S[3 5] = aab } len = 3
S[0 3] = aaba	S[2 5] = baab
S[0 4] = aabaa	S[1 5] = abaab

Ex3: S = a a a a a = 4, lps = 4

Prefix	Suffix
S[0 0] = a	S[4 4] = a
S[0 1] = aa	S[3 4] = aa
S[0 2] = aaa	S[2 4] = aaa
S[0 3] = aaaa	S[1 4] = aaaa } len = 4

Ex4: S = a lps = 0

Prefix	Suffix
~~S[0 0] = a~~	~~S[0 0] = a~~ } len = 0

Given S_N, Calculate lps[N]

lps[i] = lps value of prefix String [0..i]

	0	1	2	3	4	5	6
Ex: S =	a	a	b	a	a	b	c

	0	1	2	3	4	5	6
lps[7] =	0	1	0	1	2	3	0

lps[0] = lps value of prefix String s[0 0] = "a" = 0

0
a

Prefix	Suffix

lps[1] = lps value of prefix String s[0 1] = "aa" = 1

0 1
a a

Prefix	Suffix
s[0 0] = a	s[1 1] = a

lps[2] = lps value of prefix String s[0 2] = "aab" = 0

0 1 2
a a b

Prefix	Suffix
s[0 0] = a	s[2 2] = b
s[0 1] = aa	s[1 2] = ab

lps[3] = lps value of prefix String s[0 3] = "aaba" = 1

0 1 2 3
a a b a

Prefix	Suffix
s[0 0] = a	s[3 3] = a
s[0 1] = aa	s[2 3] = ba
s[0 2] = aab	s[1 3] = aba

lps[4] = lps value of prefix String s[0 4] = "aabaa" = 2

```
  0 1 2 3 4
  a a b a a
```

Prefix	Suffix
s[0 0] = a	s[4 4] = a
s[0 1] = aa	s[3 4] = aa
s[0 2] = aab	s[2 4] = baa
s[0 3] = aaba	s[1 4] = abaa

lps[5] = lps value of prefix String s[0 5] = "aabaab" = 3

```
  0 1 2 3 4 5
  a a b a a b
```

Prefix	Suffix
s[0 0] = a	s[5 5] = b
s[0 1] = aa	s[4 5] = ab
s[0 2] = aab	s[3 5] = aab
s[0 3] = aaba	s[2 5] = baab
s[0 4] = aabaa	s[1 5] = abaab

lps[6] = lps value of prefix String s[0 6] = "aabaabc" = 0 Try it

```
  0 1 2 3 4 5 6
  a a b a a b c
```

Prefix	Suffix

Conclusion:

Given S_N, Calculate lps[N]

lps[i] = lps value of prefix String [0..i]

```
       0 1 2 3 4 5 6
Eg: S = a a b a a b c

       0 1 2 3 4 5 6
lps[7] = 0 1 0 1 2 3 0
         +1 0 +1 +1 +1 0
```

// Calculating S_N lps[N]

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

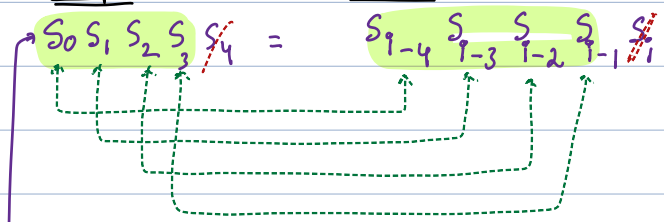
lps[ ] : _ _ _ _ _ _ _ _ _ _ 4? 5 ... _

Ass: Say lps[i] = 5 = lps of substring S[0 i] = 5

10:20pm

lps[i] = 5

Prefix Suffix



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

lps[i-1] = 4

Is it possible?

$S_0\ S_1\ S_2\ S_3\ S_4$ = $S_{i-5}\ S_{i-4}\ S_{i-3}\ S_{i-2}\ S_{i-}$

lps[i-1] >= 4.

Conclusion:

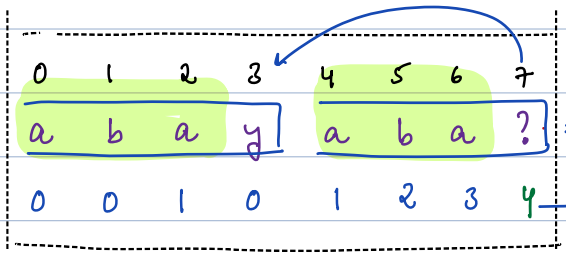
lps[i] = x lps[i-1] >= x-1

1. lps[i-1] >= x-1
2. lps[i-1] >= lps[i]-1
3. lps[i-1]+1 >= lps[i]
4. lps[i] <= lps[i-1]+1

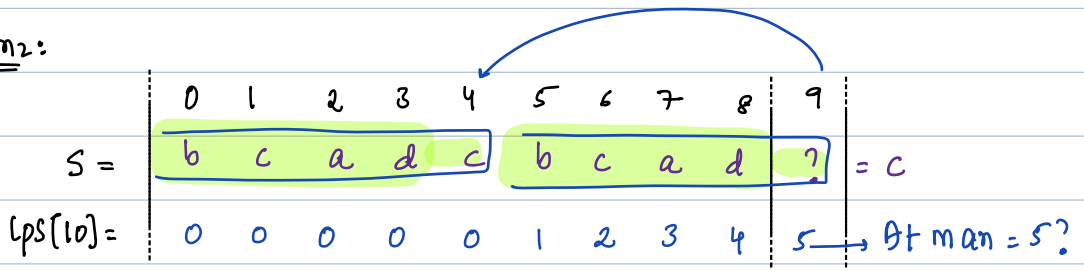
lps[i], when compared lps[i-1] data can at max inc by +1

Obs2:

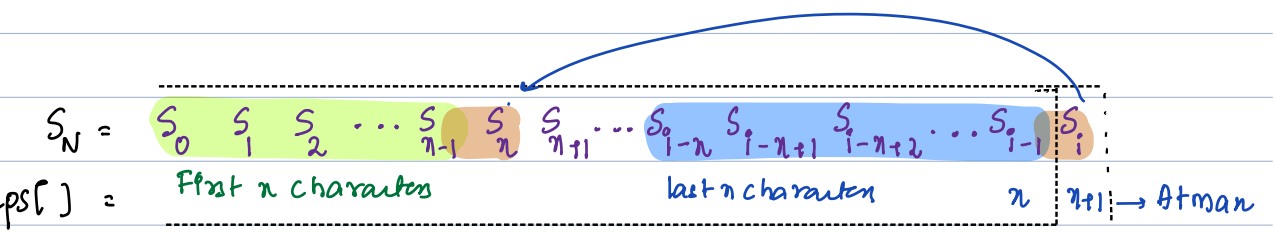
Eg1:



Eg2:



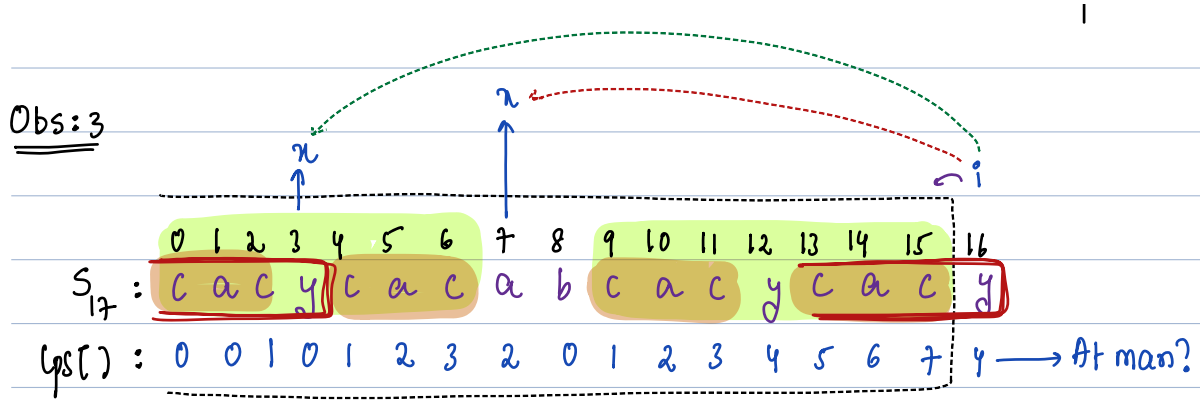
Conculsim:



Say $n = lps[i-1]$

```
if(S[i] == S[n]){
    lps[i] = n+1
}
```

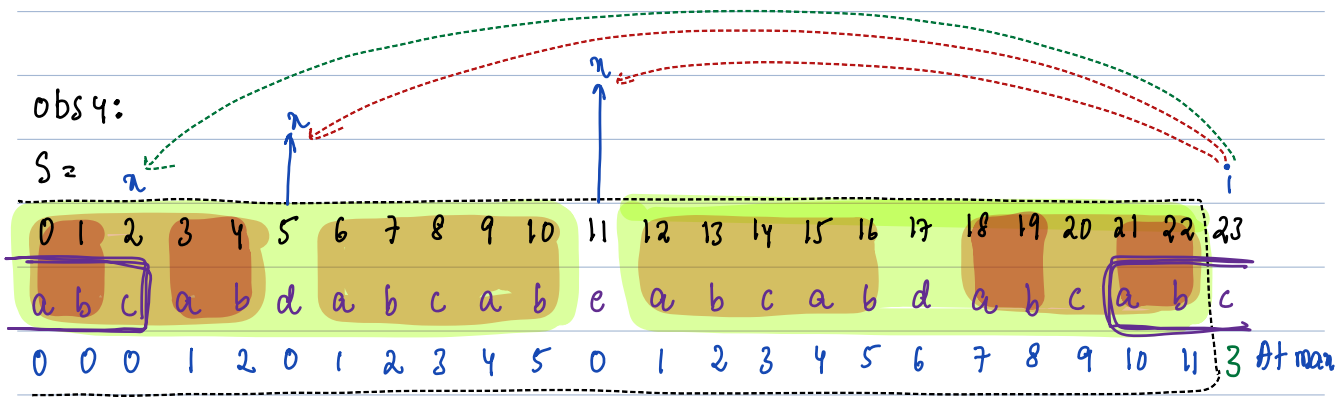
Obs: 3



Calculate lps[16]

$i = 16$, $n = lps[i-1] \Rightarrow lps[15] = 7$

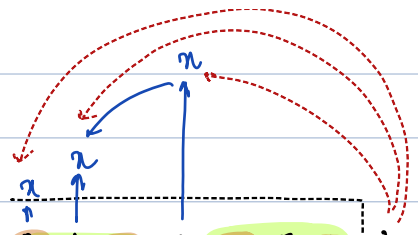
n	S[i] == S[n]	
7	S[16] ≠ S[7]	$n = lps[n-1] = lps[7-1] = lps[6] = 3$
3	S[16] == S[3]	$lps[i] = n+1 = 3+1 = 4$



Calculate lps[23]

$i = 23$, $n = lps[i-1] = lps[23-1] = lps[22] = 11$

n	S[i] == S[n]	
11	S[23] == S[11]	$n = lps[n-1] = lps[10] = 5$
5	S[23] == S[5]	$n = lps[n-1] = lps[4] = 2$
2	S[23] == S[2]	$lps[i] = n+1 = 3$



obs: 5

	0	1	2	3	4	5	6	7
s :	a	b	a	d	a	b	a	e
lps[] :	0	0	1	0	1	2	3	→ at max

Calculate lps[7]

i = 7, n = lps[i-1] = lps[6] = 3

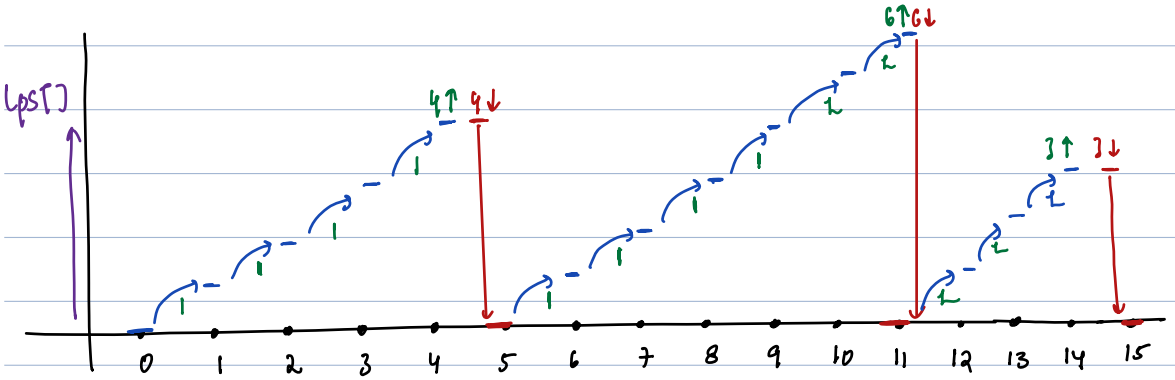
n	s[i] == s[n]	
3	s[7] == s[3]	n = lps[n-1] = lps[2] = 1
1	s[7] == s[1]	n = lps[n-1] = lps[0] = 0
0	s[7] == s[0]	if n == 0, condition fails : no update break

```
int[] createLps(String s) { TC: O(N) SC: O(N)
    int N = s.length;
    int lps[N];
    lps[0] = 0;
    for(int i=1; i<N; i++){
        // Calculate lps[i] ?
        int n = lps[i-1]
        while(s[i] != s[n]){
            if(n==0) { n=-1; break }
        }   n = lps[n-1];
        lps[i] = n+1;
    }
}   return lps;
```

obs 6: In lps[i] value at most increases by 1

When ever it increases by 1, it will have 1 iteration

Qn: Given Str of 16, lps[16] ...



Total inc iterations = N
Total dec iteration = N
} Total iterations = 2N, TC: O(N) SC: O(N)

Q: Count occurences of pattern P_k in Text T_N

Ex1:

T = a a b a c d ans = 1

P = a b a c

Ex2:

```
    0 1 2 3 4 5 6 7 8 9 10
T = a b a d c a b a b a e   ans = 3
P = a b a
```

Idea1: 1. Get all substrings of len k from $Text_N$ & compare with Pattern P

TC: O(N-k+1) * O(k) ≈ O(N²)

Idea2:

```
          0 1 2 3 4 5 6 7 8 9 10
T_N =     a b a d c a b a b a e   ans = 3
P_k = a b a
```

2 diff both P & T

S = P + "#" + T separator is diff from P & T

```
          0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
S =       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
```

obs: if lps[i] == k

Pattern found.

```
int count(String P(k), String T(N)) {   TC: O(N+k+1) = O(N+k)  SC: O(N+k)
    Str  S = P + "#" + T
    int lps[] = createLps(s);
    int c=0;
    for(int i=0; i< lps.length; i++) {
        if (lps[i] = P.length()) { c=c+1 }
    }
    return c;
}
```

```
boolean match(String P, String T){   TC: O(N+k)  SC: O(N+k)
    int N = T.length();
    int k = P.length();
    String C = P + "@" + T;  ---> TC: O(N+k)
    int lps[N+k+1];
    lps = createLps(C);  ---> TC: O(N+k)
    for(i = 0; i <= (N+k); i++){  ---> TC: O(N+k)
        if( lps[i] == k){
            return True
        }
    }
    return False;
}

int createLps(String s){
    int N = s.length();
    int lps[N];
    lps[0] = 0;
    for(int i = 1; i < N; i++){
        // Calculate lps[i]
        int x = lps[i-1];
        while(s[i] != s[x]){ // at x==0, s[i] != s[0]
            if(x==0){ x=-1; break;}
            x = lps[x-1]
        }
        lps[i] = x+1
    }
    return lps;
}
```