Todays content

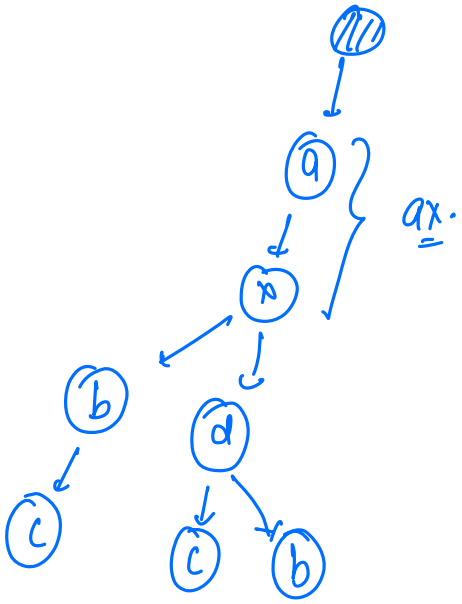
- → Longest prefix
- → Boring Substring
- → KMP Algorithm (Knuth-Morris-Pratt)

① Given an array of strings. find the longest string which is prefix of all strings in the array.

["axdc", "axbc", "axdb"] ans → ax.

N → no. of words
m → max length of a word.

1→ Trie.



[T.C → O(N x m)
S.C → O(N x m)]

2→ Brute force

a x d c
a x b c
a x d b
a x a b
ax

[T.C → O(N x m)
S.C → O(1)]

2. Given a string s, check whether it is possible to re-arrange the characters of the string s such that there is no boring substring in s.

Boring substring → length = 2 and consecutive alphabets.

↳ ab, cd, xy, yz, yx, dc, cb, ba

s = "abc"

a b c
a c b
b a c
b c a
c a b
c b a

false

s = "abcd"

abcd ✗

cadb
bdac

true

s = "aabccb"

ans → false

baaccb ✗
bbaacc ✗

idea-1 → Consider all the permutations of given string & check if they are having boring substring or not.

T.C → O(N! × N)

Idea-2

S = "aabccdb" bbdaacc ans → true.

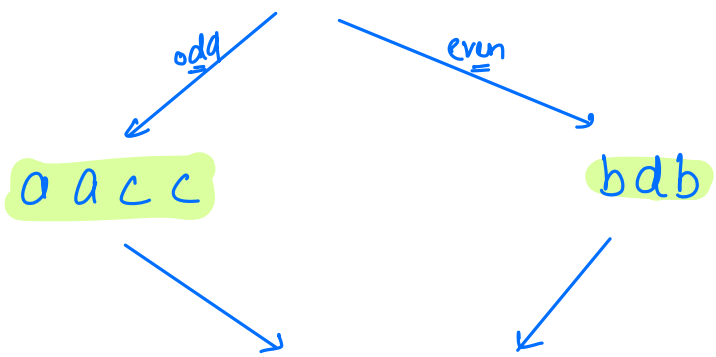
a b c d e f g h i j k ---w x y z

a c e g i k -- -- w y

b d f h j l -- -- x z

[a → 97
b → 98
c → 99
d → 100
⋮
z → 122]

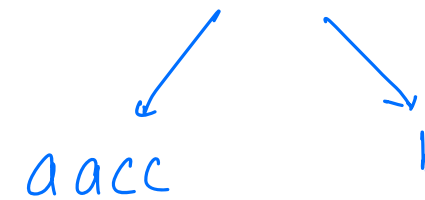
S = "a a b c c d b"



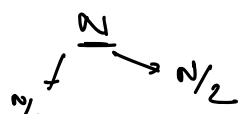
ans = true.

- - - a d - -

S = "a a b c c b"



ans = false.



T.C → $\frac{N}{2} \times \frac{N}{2} \rightarrow O(N^2)$

idea-2 →

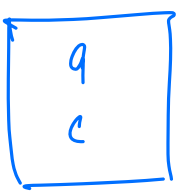
i) only check the unique characters.

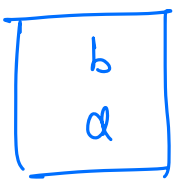
a c e — — y (13) b d f — — x z (13)

13 x 13 iterations

T.C → O(1)

ii) Hashmap / Hashset to check if odd ASCII characters, if anything other than (ch-1) or (ch+1) is present.





T.C → O(1)

iii) odd set → smallest, largest

even set → smallest, largest

a a a c c e g g

b b d f f

a	b	X
g	f	X
a	f	✓
g	b	✓

true.

S = " c c e b b d f "

smallest-odd ASCII char → c

largest-odd ASCII char → e

smallest-even ASCII char → b

largest-even ASCII char → f

ans → true.

T.C → O(N)
S.C → O(1)

KMP [Knuth-Morris-Pratt Algorithm]

Given a string s of N size.

prefix substrings → substrings starting with idx → 0

suffix substrings → substrings ending with idx → N-1

Eg → s = "a b a b"
0 1 2 3

prefix substrings

a
ab
aba
abab

suffix substrings

b
ab
bab
abab

LPS of a string → length of longest prefix substring which is also a suffix sub-string.

Note: → Except the whole string.

Eg1 : s = a b c a b → [2]

0	1
2	3
4	

prefix substrings	suffix substrings
a	b
ab	ab
abc	cab
abca	bcab

Eg2 : s = "a" → 0

Eg3 : s = "a a a a a" → 4.

0	1
2	3
4	

prefix	suffix
a	a
aa	aa
aaa	aaa
aaaa	aaaa

Eg 4 : S = a a b a a b a → 4.
0 1 2 3 4 5 6

prefix	suffix
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

Given a string s of length n. Return the LPS[]

LPS[i] → LPS value of substring [0, i].

s → a a b a a b a
0 1 2 3 4 5 6

LPS[] → | 0 | 1 | 0 | 1 | 2 | 3 | 4 |

LPS[1] → "aa"
a a

LPS[2] → "aab"
a b
aa ab

LPS[3] → "aaba"
a a
aa ba
aab aba

LPS[4] → "aabaa"

a	a
aa	aa
aab	baa
aaba	abaa

LPS[5] → "aabaab"

a	b
aa	ab
aab	aab
aaba	baab
aabaa	abaab

Q→ S = a a b a c a a b a

0	1	2	3	4	5	6	7	8
0	1	0	1	0	1	2	3	4

LPS[] →

Q- S = a a b a c d

0	1	2	3	4	5
0	1	0	1	0	0

LPS[] →

→ Pattern Matching

Text → a a b a c [N]

pattern → a b a c [M].

B.f. → Go to each idx & compare characters one by one.

T.C → O(N x M)

T → d a b c e a b c

p → a b c

delimeter

p # T

S → a b c # d a b c e a b c

LPS[] → 0 0 0 0 0 1 2 3 0 1 2 3

pattern is matched.

T → a a a a

P → a a

S : P+T → a a a a a a

LPS[] → | 0 | 1 | 2 | 3 | 4 | 5 |

S : P + "#" + T → a a # a a a a

LPS → | 0 | 1 | 0 | 1 | 2 | 2 | 2 |

ans : 3.

→ Calculate LPS[N]

Obs-1: If LPS[i] = 5.

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

LPS[] → $\dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \underline{5}$

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

$\times \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \times$

$\downarrow$

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

$LPS[i-1] \geq 4.$

Assuming.

$LPS[i] = x$

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

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

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

$\boxed{LPS[i] \leq LPS[i-1] + 1}$

$LPS[i] = LPS[i-1] + 1$ (At max)

Step-2.

S → a b a y a b a ?

lps[] → 0 0 1 0 1 2 3

if ? == y.

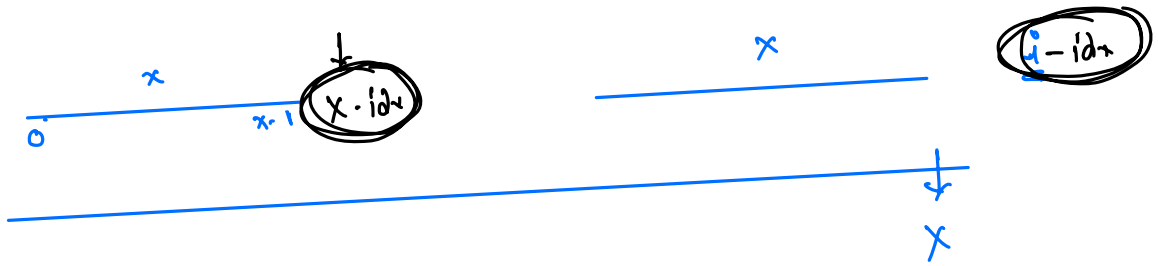
then lps[i] → 4.

S → b c a d c b c a d ?

0 1 2 3 4 5 6 7 8 9

lps[] → 0 0 0 0 0 1 2 3 4

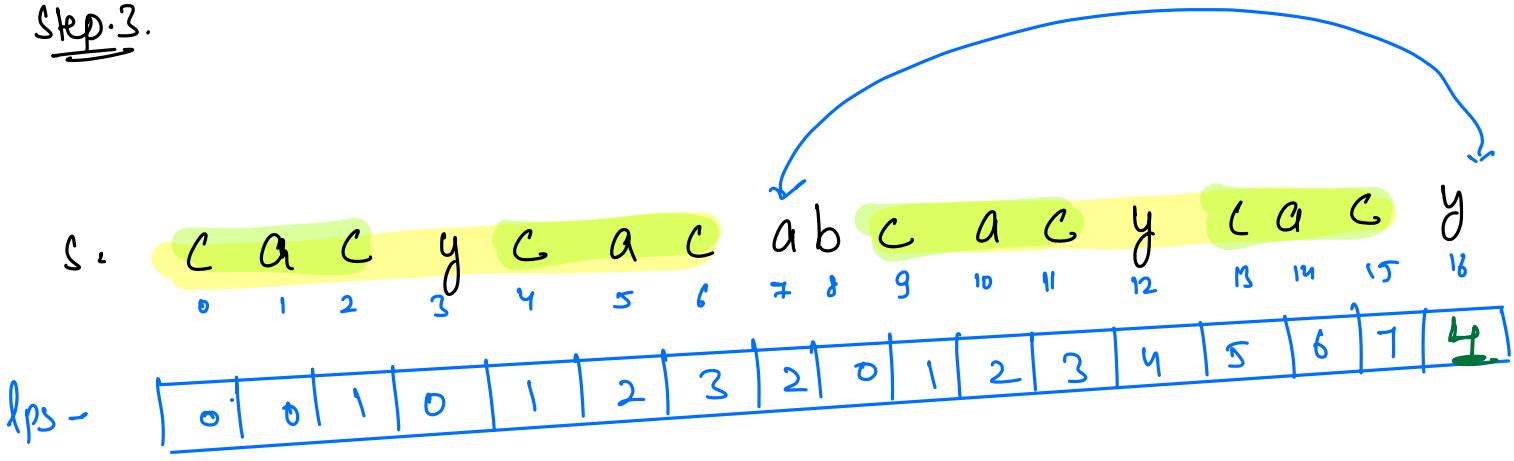
↳ 5.



calculating lps[i] →

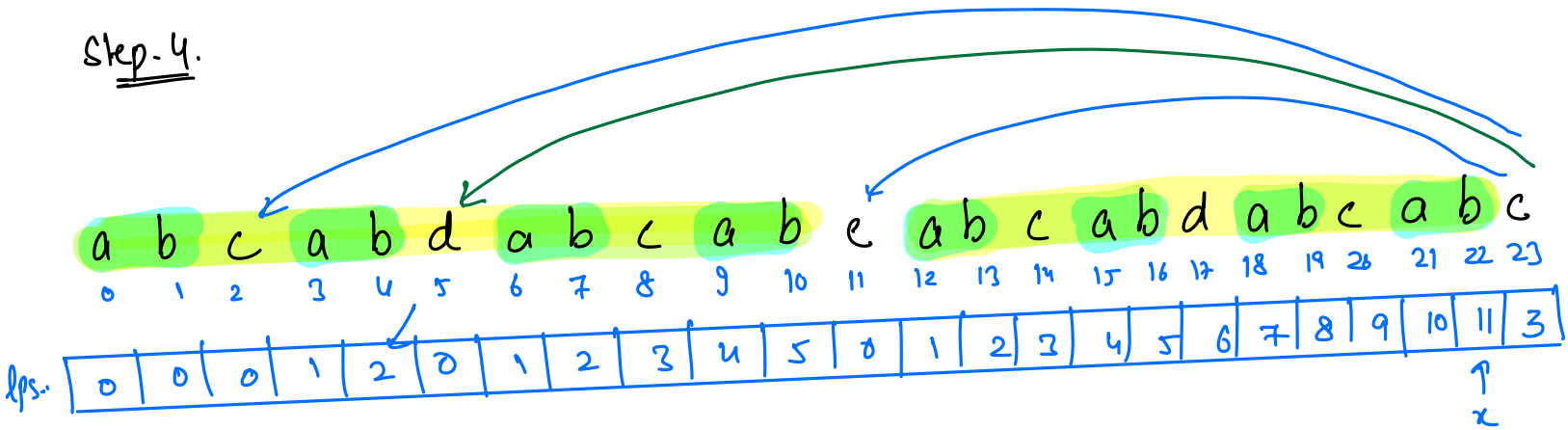
```
x = lps[i-1]
if( s[x] == s[i]){
    lps[i] = lps[i-1] + 1;
}
```

Step-3.



lps[16].

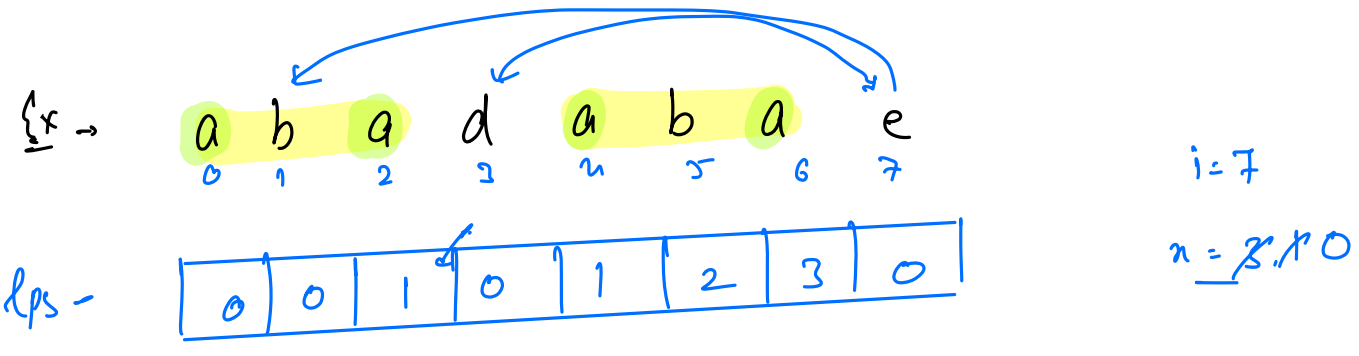
Step-4.



i=23.

```
for( i=1; i<N; i++){
    x = lps[i-1];
    while( s[i] != s[x]){
        if(x==0) { x = -1, break }
        x = lps[x-1];
    }
    lps[i] = x+1;
}
```

x = ~~4~~ 2



i = 7

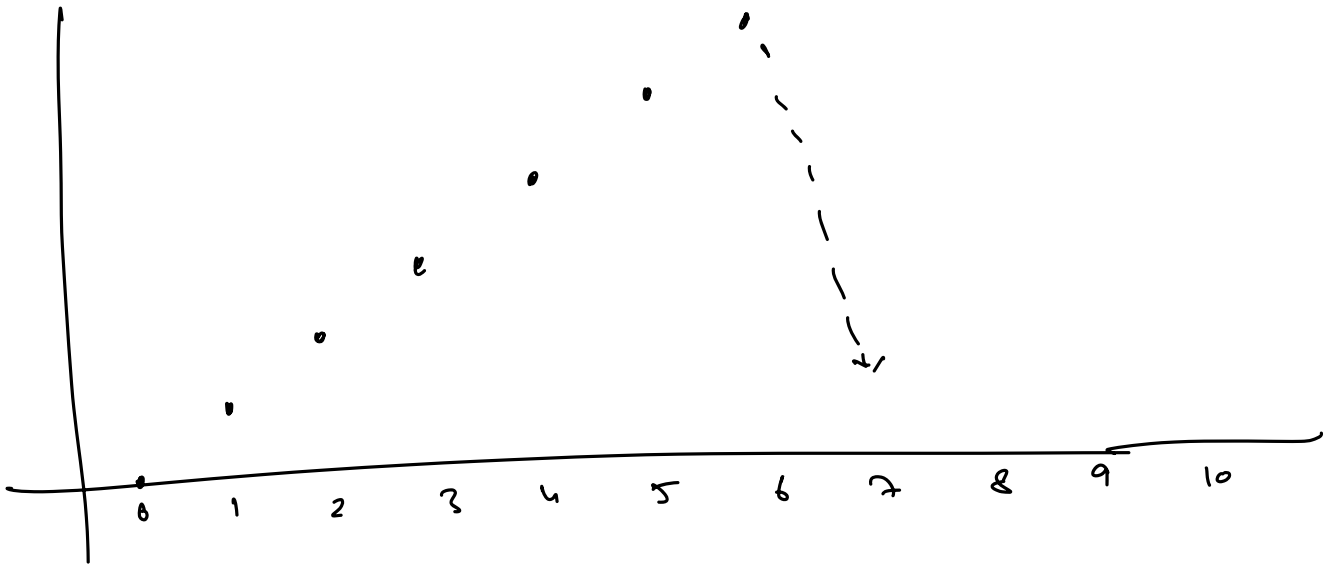
x = 3 → 0

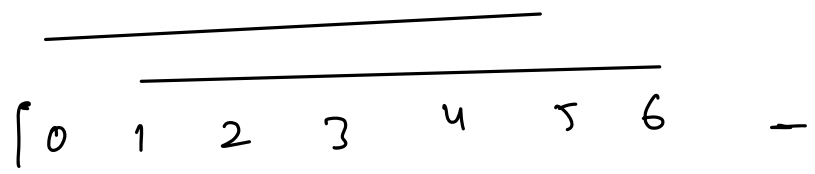
x = lps[i-1] ✗ ⇒ x = lps[x-1]

code →

```
LPS (string s){
    int n = str.length() , lps[N];
    lps[0] = 0;
    for( i = 1; i < N; i++){
        x = lps[i-1];
        while( s[i] != s[x]){
            if(x==0) { x = -1, break }
            x = lps[x-1];
        }
        lps[i] = x+1;
    }
    return lps[];
}
```

[TC → O(N)]






```
x = lps[i-1];
while( s[i] != s[x]){
        if(x==0) { x = -1, break }
        x = lps[x-1];
}
lps[i] = x+1;
```

i:7.

x=6
5
4
3
2
1
0

[T.C → O(N)]

⇒ Online Assignment problem.