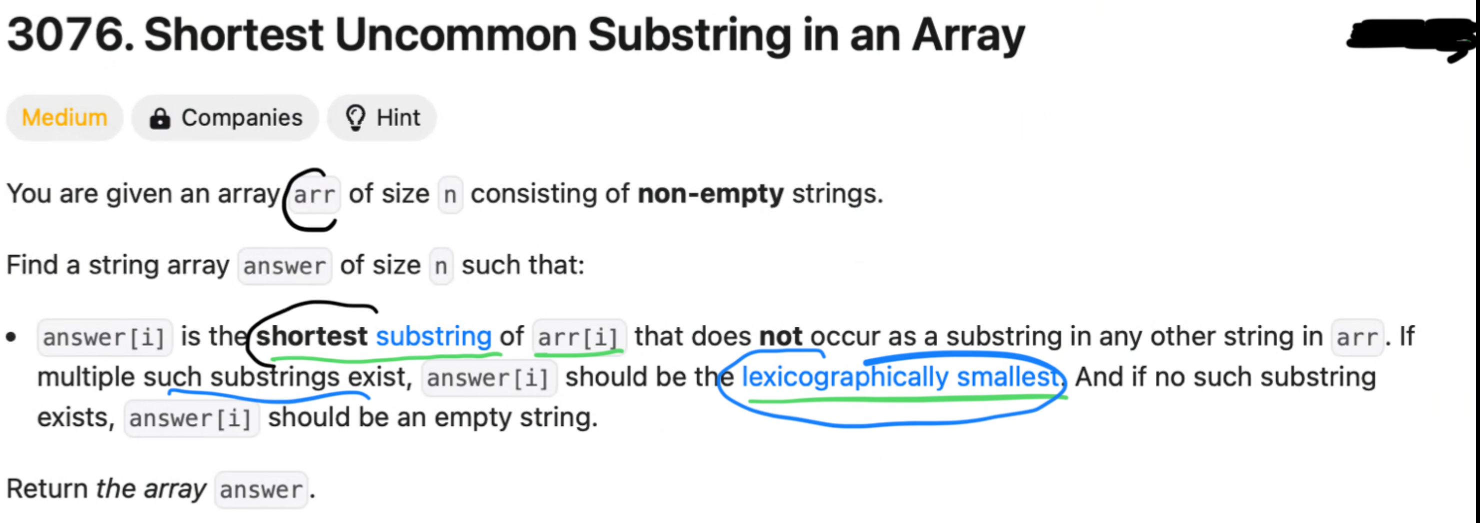
@codestorywithmik
(Instagram, Facebook)
cswithMIK → Twitter
codestorywithMIK → whatsapp

STRINGS...

Video - 36

Leetcode - 3076

~~Medium~~ → Straight Forward



3076. Shortest Uncommon Substring in an Array

Medium | Companies | Hint

You are given an array `arr` of size `n` consisting of **non-empty** strings.

Find a string array `answer` of size `n` such that:

- `answer[i]` is the **shortest** substring of `arr[i]` that does **not** occur as a substring in any other string in `arr`. If multiple such substrings exist, `answer[i]` should be the lexicographically smallest. And if no such substring exists, `answer[i]` should be an empty string.

Return *the array* `answer`.

arr = { "cab", "ad", "bad", "c" } n=4

Example :- arr = {

Output = { "ab", " ", "ba", " " } ,n=4

cab → { "c" ×, "ca" ✓ 2, "cab", "a" ×, "ab" ✓ 2, "b" × }

ad → {

A simple & straight forward thought Process

arr = { "cab", "ad", "bad", "c" }
i

cab → { "c", "ca", "cab", "a", "ab", "b" } ⇒ { "ab" }

ad → { "a" ×, "ad" ×, "d" × } = { " " }

bad → { "b" ×, "ba" ✓, "bad" ✓, "ad" ×, "d" } = { "ba" }

c → { "c" × } = { " " }

Out :- { "ab", " ", "ba", " " }.

Story Points :- ...

1. iterate on input arr.

   `for ( i=0 ; i< arr.len() ; i++) { //arr[i] }`

2. Find all substring of each arr[i]

3. check for each substring which is not present in any other string arr[j] as substring.

   " you need to find substrings of arr[j] also"

4. Choose shortest / lexicogr. smallest one for each arr[i] else " ".

Pseudo Code.

```
n ⇒ for(i=0 ;  i< arr.length();  i++) {  ⇐
n^3 <- substrings = { all subst of arr[i] } ; <= loop
        string shortest = " ";
n^2  for (each substr) {

   n -> for( j=0; j< arr.length(); j++) {
                  if(j == i) continue;

        n^3 -> CheckSubstr = { all subst of arr[j] } ; //loop

        n^2 -> Check substr is not present as substrings
               in checkSubstr // loop.
               updating shortest.
        }
     }

     result. push_back(shortest);

}
```

T.L.E.

Problem with Current Approach...

arr = { "cab", "ad", "bad", "c" }

i

cab → { "c", "ca", "cab", "a", "ab", "b" } ad → {"a", "ad", "d"}

bad → {"b", "ba", "bad", "a", "ad", "d"}

c → {"c"}

(•) Precompute all substring and store it.

substr_freq (map)

substring	count
"c"	2
"ca"	1
"a"	3
"d"	2
"ba"	1
"cab"	1
"bad"	1
"ab"	1
"b"	2
"ad"	2