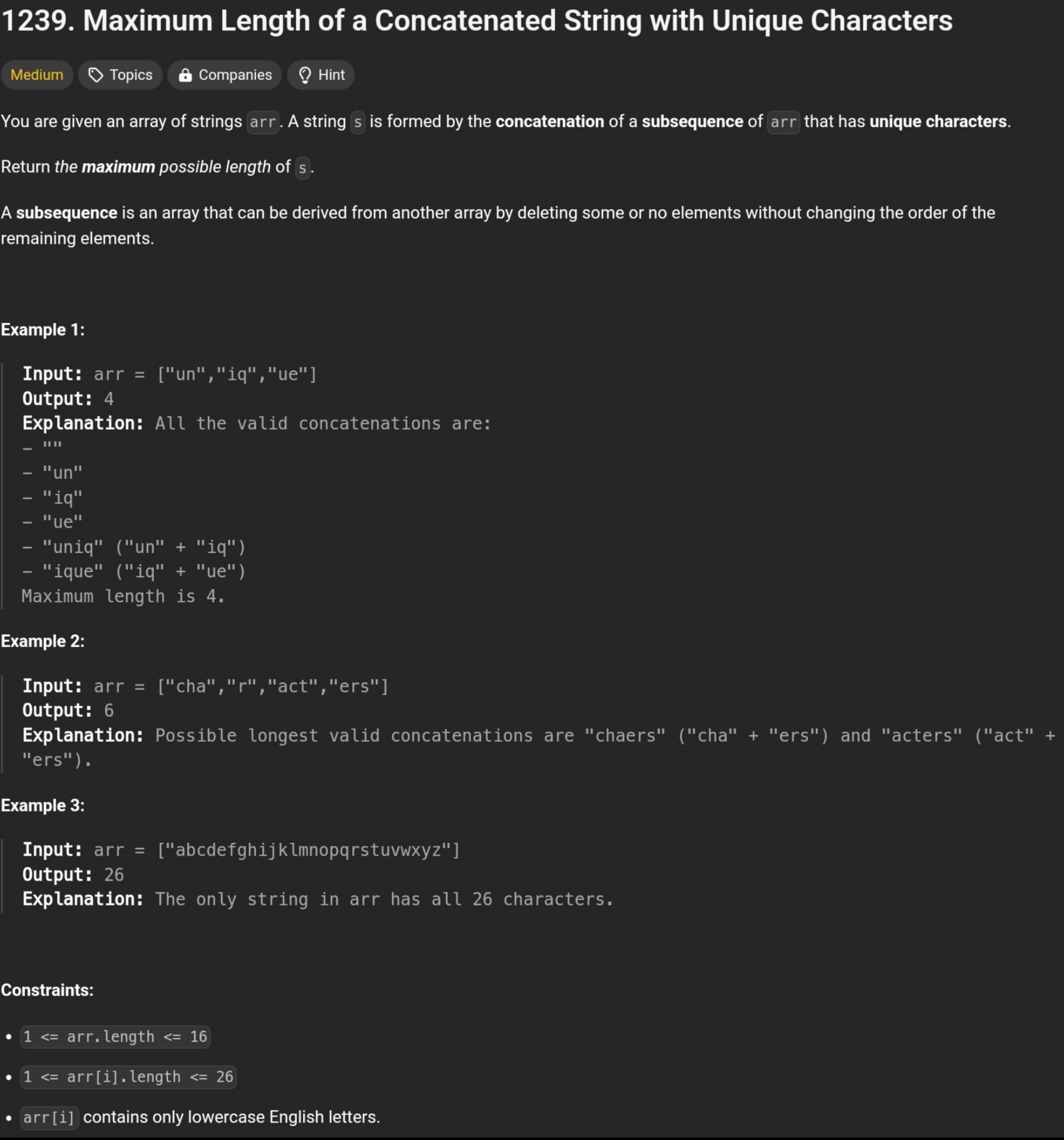


1239. Maximum Length of a Concatenated String with Unique Characters

You are given an array of strings `arr`. A string `s` is formed by the **concatenation** of a **subsequence** of `arr` that has **unique characters**.

Return *the **maximum** possible length* of `s`.

A **subsequence** is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Example 1:

```
Input: arr = ["un","iq","ue"]
Output: 4
Explanation: All the valid concatenations are:
- ""
- "un"
- "iq"
- "ue"
- "uniq" ("un" + "iq")
- "ique" ("iq" + "ue")
Maximum length is 4.
```

Example 2:

```
Input: arr = ["cha","r","act","ers"]
Output: 6
Explanation: Possible longest valid concatenations are "chaers" ("cha" + "ers") and "acters" ("act" +
"ers").
```

Example 3:

```
Input: arr = ["abcdefghijklmnopqrstuvwxyz"]
Output: 26
Explanation: The only string in arr has all 26 characters.
```

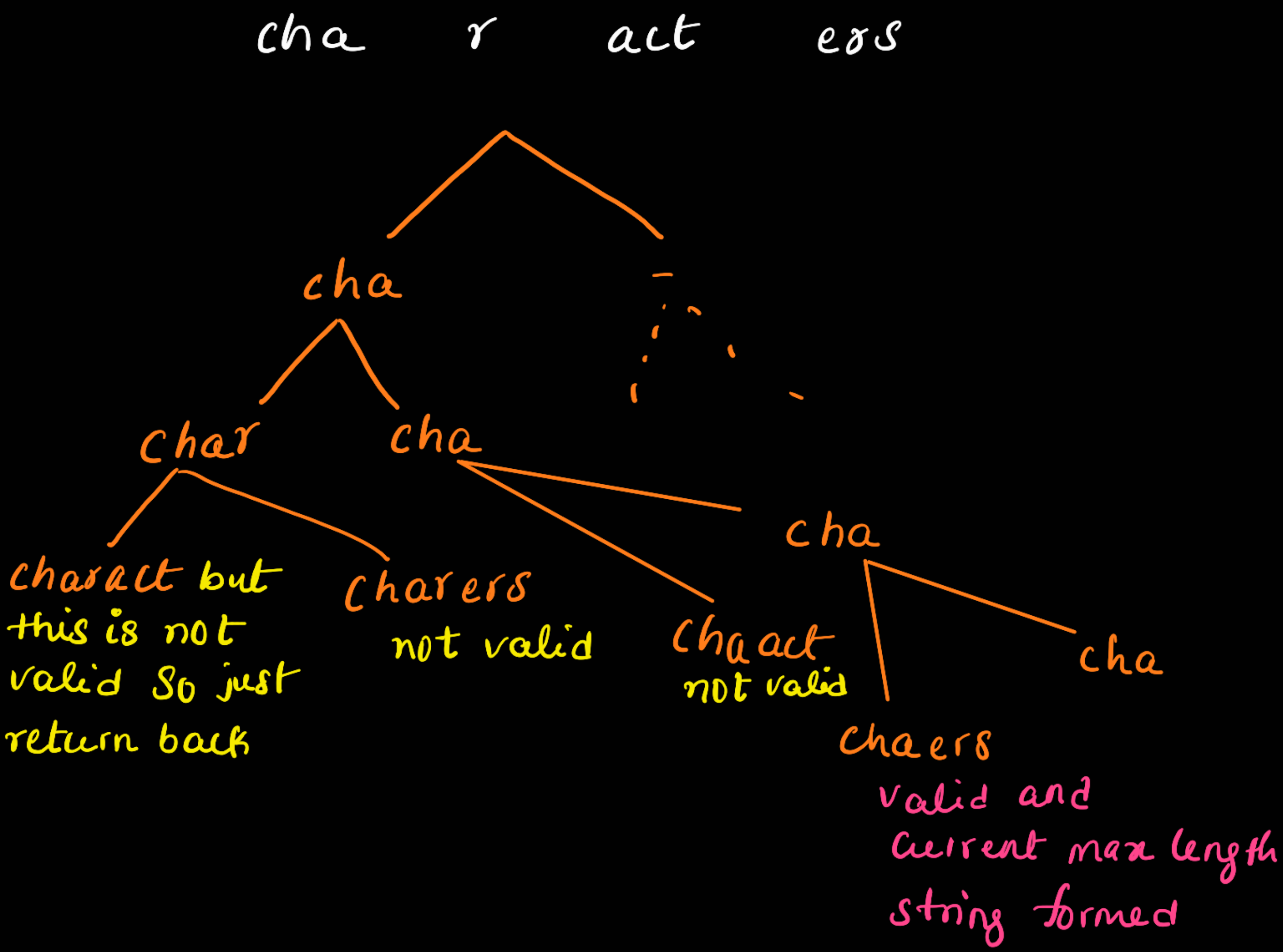
Constraints:

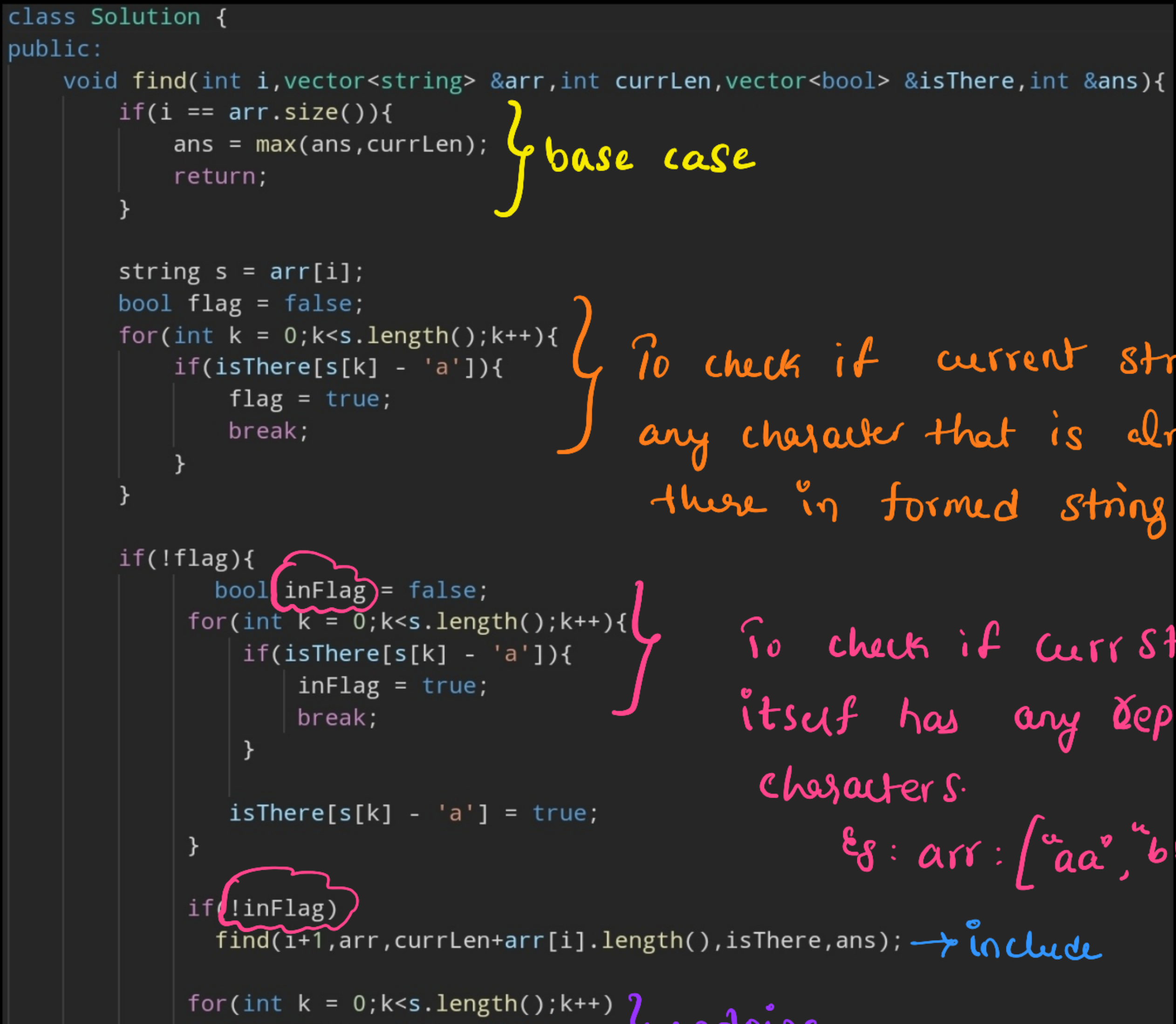
- `1 <= arr.length <= 16`
- `1 <= arr[i].length <= 26`
- `arr[i]` contains only lowercase English letters.

we can view this problem as a combination problem.

for every string we have 2 choices - "include or not include"

This problem is pretty simple intuitively but things get a bit complicated while implementing.


cha r act ers
cha
char
cha
charact but this is not valid So just return back
charers not valid
cha
chaact not valid
cha
chaers valid and current max length string formed



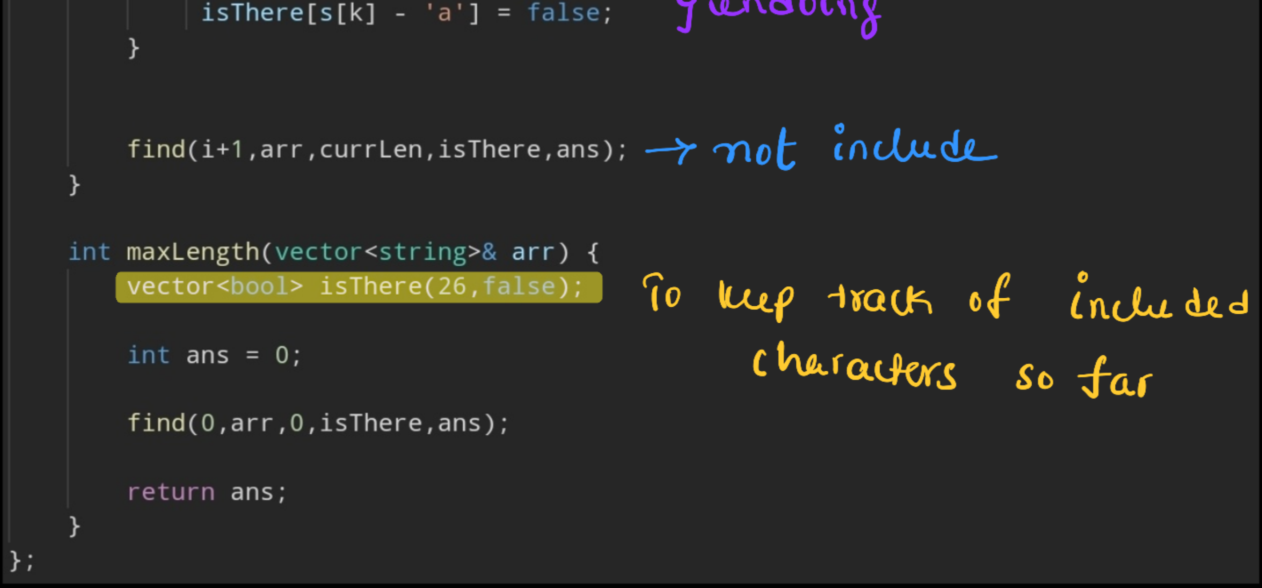
```
class Solution {
public:
    void find(int i,vector<string> &arr,int currLen,vector<bool> &isThere,int &ans){
        if(i == arr.size()){
            ans = max(ans,currLen);
            return;
        }

        string s = arr[i];
        bool flag = false;
        for(int k = 0;k<s.length();k++){
            if(isThere[s[k] - 'a']){
                flag = true;
                break;
            }
        }

        if(!flag){
             bool inFlag = false;
            for(int k = 0;k<s.length();k++){
                if(isThere[s[k] - 'a']){
                    inFlag = true;
                    break;
                }

                isThere[s[k] - 'a'] = true;
            }

            if(!inFlag)
              find(i+1,arr,currLen+arr[i].length(),isThere,ans);

            for(int k = 0;k<s.length();k++)
```

} base case

To check if current string has any character that is already there in formed string so far.

To check if curr String itself has any repeating characters.
Eg : arr : ["aa", "bb"]

→ include



```cpp
            isThere[s[k] - 'a'] = false;
        }


        find(i+1,arr,currLen,isThere,ans);
    }

    int maxLength(vector<string>& arr) {
        vector<bool> isThere(26,false);

        int ans = 0;

        find(0,arr,0,isThere,ans);

        return ans;
    }
};
```

[illegible]

→ not include

To keep track of included characters so far