

## Questions

23 September 2024 17:23

**38. Count and Say** Solved

Medium Topics Companies Hint

The **count-and-say** sequence is a sequence of digit strings defined by the recursive formula:

- `countAndSay(1) = "1"`
- `countAndSay(n)` is the run-length encoding of `countAndSay(n - 1)`.

Run-length encoding (RLE) is a string compression method that works by replacing consecutive identical characters (repeated 2 or more times) with the concatenation of the character and the number marking the count of the characters (length of the run). For example, to compress the string "33322251", we replace "333" with "323", replace "222" with "32", replace "5" with "15" and replace "1" with "11". Thus the compressed string becomes "32321511".

Given a positive integer `n`, return the  $n^{\text{th}}$  element of the **count-and-say** sequence.

**Example 1:**

Input: `n = 4`

Output: "1211"

Explanation:

`countAndSay(1) = "1"`

4.1K 143 1

```
class Solution {
    public String countAndSay(int n) {
        String ans="1";
        for(int i=2; i<=n; i++){
            String ans2=ans;
            ans="";
            int size=ans2.length();
            char curr_char=ans2.charAt(0);
            int count=0;
            for(int j=0; j<size; j++){
                if(ans2.charAt(j)==curr_char){
                    count++;
                }else{
                    ans+=count+" "+curr_char;
                    curr_char=ans2.charAt(j);
                    count=1;
                }
            }
            ans=ans+count+" "+curr_char;
        }
        return ans;
    }
}
```

**1662. Check If Two String Arrays are Equivalent** Solved

Easy Topics Companies Hint

Given two string arrays `word1` and `word2`, return `true` if the two arrays **represent** the same string, and `false` otherwise.

A string is **represented** by an array if the array elements concatenated **in order** forms the string.

**Example 1:**

Input: `word1 = ["ab", "c"], word2 = ["a", "bc"]`

Output: `true`

Explanation:

`word1` represents string "ab" + "c" -> "abc"

`word2` represents string "a" + "bc" -> "abc"

The strings are the same, so return true.

**Example 2:**

Input: `word1 = ["a", "cb"], word2 = ["ab", "c"]`

Output: `false`

**Example 3:**

Input: `word1 = ["abc", "d", "defn"], word2 = ["abcddefn"]`

3K 69 1

```
class Solution {
    public boolean arrayStringsAreEqual(String[] word1, String[] word2) {
        StringBuilder sb1=new StringBuilder("");
        StringBuilder sb2=new StringBuilder("");
        for(int i=0; i<word1.length; i++){
            sb1.append(word1[i]);
        }
        String first=sb1.toString();
        for(int i=0; i<word2.length; i++){
            sb2.append(word2[i]);
        }
        String second=sb2.toString();
        if(first.equals(second)){
            return true;
        }
        return false;
    }
}
```

Do leetcode 859

Make The String Great - LeetCode

1544. Make The String Great

Easy

Given a string `s` of lower and upper case English letters.

A good string is a string which doesn't have **two adjacent characters** `s[i]` and `s[i + 1]` where:

- `s[i]` is a lower-case letter and `s[i + 1]` is the same letter but in upper-case or **vice-versa**.

To make the string good, you can choose **two adjacent** characters that make the string bad and remove them. You can keep doing this until the string becomes good.

Return the string after making it good. The answer is guaranteed to be unique under the given constraints.

**Notice** that an empty string is also good.

**Example 1:**

Input: `s = "leEeetcode"`

Accepted

Runtime: 0 ms

```
class Solution {
    public String makeGood(String s) {
        StringBuilder sb=new StringBuilder();
        sb.append(s);

        for(int i=0; i<sb.length()-1; i++){
            int a=(int)sb.charAt(i);
            int b=(int)sb.charAt(i+1);
            if(Math.abs(a-b)==32){
                sb.deleteCharAt(i);
                sb.deleteCharAt(i);
                i=i-1;
            }
        }
        return sb.toString();
    }
}
```

h/w 1657 and 520

Detect Capital - LeetCode

520. Detect Capital

Easy

We define the usage of capitals in a word to be right when one of the following cases holds:

- All letters in this word are capitals, like `"USA"`.
- All letters in this word are not capitals, like `"leetcode"`.
- Only the first letter in this word is capital, like `"Google"`.

Given a string `word`, return `true` if the usage of capitals in it is right.

**Example 1:**

Input: `word = "USA"`  
Output: `true`

**Example 2:**

Input: `word = "Flag"`  
Output: `false`

Accepted

Runtime: 0 ms

```
class Solution {
    private static boolean bigChar(String str) {
        int[] check = new int[str.length()];
        check[0]=1;
        for(int i=1; i<str.length(); i++){
            if(str.charAt(i)>=97 && str.charAt(i)<=122){
                return smallChar(str);
            }
            check[i]=1;
            if(check[i]!=check[i-1]){
                return false;
            }
        }
        return true;
    }

    private static boolean smallChar(String str) {
        int[] check = new int[str.length()];
        check[0] = 0;
        for (int i = 1; i < str.length(); i++) {
            int a = (int) (str.charAt(i));
            if (a>=97 && a<=122) {
                check[i]=0;
                if(check[i]!=check[i-1]){
                    return false;
                }
            }
        }
        return true;
    }
}
```