**Eagle View Company - Bangalore Location - March 3rd Online Assessment**

**Interview Platform:** Third-Party Platform  
 **Difficulty Level:** Easy to Medium

## **Assessment Overview**

The assessment consists of two sections:

1. **Java + Selenium Testing** (Login Page Validation)
2. **Data Structures & Algorithms (DSA) Problem**

## **Section 1: Java + Selenium Testing**

### **Task: Validate a Login Page Using Selenium & JUnit**

You are provided with a login page, and your task is to validate its functionality by writing JUnit test cases in the AppTest class. The webDriver object is already initialized and pointing to the page being tested.

### **Test Cases to Implement:**

#### **1. Check if email and password fields exist on the main screen:**

* Ensure the presence of elements using the following IDs:
  + email-input (Email field)
  + password-input (Password field)
  + login-button (Login button)

#### **2. Validate login with correct credentials:**

* Use:
  + Email: login@codility.com
  + Password: password
* Expected Outcome:
  + A div with class message success containing the text: **"Welcome to Codility"** should be visible.

#### **3. Validate login with incorrect credentials:**

* Use:
  + Email: unknown@codility.com
  + Password: password
* Expected Outcome:
  + A div with class message error containing the text: **"You shall not pass! Arr!"** should be visible.

#### **4. Validate incorrect email format:**

* Use an invalid email format (e.g., invalid-email).
* Expected Outcome:
  + A div with class validation error containing the text: **"Enter a valid email"** should be visible.

#### **5. Validate empty credentials:**

* Leave email and/or password empty.
* Expected Outcomes:
  + A div with class validation error containing the text: **"Email is required"** (if the email field is empty).
  + A div with class validation error containing the text: **"Password is required"** (if the password field is empty).

#### **6. Validate keyboard navigation (TAB and ENTER keys):**

* Ensure the TAB key moves focus between fields correctly.
* Ensure the ENTER key submits the form properly.

### **Assessment Notes:**

* The provided WebDriver instance (public static WebDriver webDriver) must be used.
* Annotate each test case with @Test.
* **Performance is not assessed**, focus only on correctness.
* Debugging can be done using System.out.println().

## **Section 2: Data Structures & Algorithms (DSA)**

### **Problem:** [**Maximum Number of Balloons**](https://leetcode.com/problems/maximum-number-of-balloons/description/)

**Problem Statement:** Given a string text, return the maximum number of times the word **"balloon"** can be formed using the characters in text.

**Constraints:**

* 1 <= text.length <= 10^4
* text consists of lowercase English letters.

### **Solution Approach:**

#### **1. Frequency Count Method (Optimal Approach - O(n) Complexity)**

* Count the occurrences of each character in text.
* Since "balloon" consists of specific letters (b, a, l, o, n), determine the number of times we can form "balloon" based on character availability.
* Note that l and o appear **twice** in "balloon", so their counts must be halved.
* Find the limiting character by taking the minimum count among b, a, l/2, o/2, and n.

#### **2. Implementation in Python:**

from collections import Counter

def maxNumberOfBalloons(text: str) -> int:

count = Counter(text)

return min(count.get('b', 0), count.get('a', 0), count.get('l', 0) // 2, count.get('o', 0) // 2, count.get('n', 0))

# Example usage

text = "loonbalxballpoon"

print(maxNumberOfBalloons(text)) # Output: 2

#### **3. Implementation in Java:**

import java.util.\*;

public class MaxBalloons {

public static int maxNumberOfBalloons(String text) {

Map<Character, Integer> count = new HashMap<>();

for (char c : text.toCharArray()) {

count.put(c, count.getOrDefault(c, 0) + 1);

}

return Math.min(count.getOrDefault('b', 0),

Math.min(count.getOrDefault('a', 0),

Math.min(count.getOrDefault('l', 0) / 2,

Math.min(count.getOrDefault('o', 0) / 2, count.getOrDefault('n', 0)))));

}

public static void main(String[] args) {

System.out.println(maxNumberOfBalloons("loonbalxballpoon")); // Output: 2

}

}

### **Time Complexity Analysis:**

* **O(n):** Since we iterate over the string once to count occurrences.

### **Edge Cases Considered:**

1. **Minimum length input**: "b" → Output: 0
2. **Only one occurrence of each letter**: "balon" → Output: 0
3. **Characters scattered randomly**: "nlaebolko" → Output: 1
4. **Large input cases** → Should still run efficiently within O(n).

## **Conclusion**

* The **Java + Selenium** section tests the ability to write functional UI test cases.
* The **DSA question** focuses on efficient string processing using **hash maps**.
* Preparation should include practicing Selenium test case writing and revising common **string manipulation techniques**.

Good luck with your online assessment!