You can view this report online at : https://www.hackerrank.com/x/tests/1311130/candidates/52620379/report

Full Name:	Daniel Fernando Ladino Yate
Email:	dflyate@gmail.com
Test Name:	**20220209 Talent Matcht Prueba 1 backend**
Taken On:	25 May 2023 11:31:54 -05
Time Taken:	59 min 51 sec/ 60 min
Work Experience:	> 5 years
Personal Email Address:	dflyate@gmail.com
Invited by:	Carlos
Invited on:	23 May 2023 14:27:42 -05
Skills Score:	Problem Solving (Basic) 12/150
Tags Score:	Algorithms 6/50 Arrays 0/50 Easy 12/150 Greedy Algorithms 0/50 Interviewer Guidelines 6/50 Loops 6/50 Problem Solving 6/50 Sorting 0/50 Strings 12/100

8%

12/150

scored in **20220209 Talent Matcht Prueba 1 backend** in 59 min 51 sec on 25 May 2023 11:31:54 -05

Recruiter/Team Comments:

No Comments.

Question Description	Time Taken	Score	Status
Q1 **Shortest Substring** > **Coding**	37 min 4 sec	6/ 50	✓
Q2 **No Pairs Allowed** > **Coding**	20 min 51 sec	6/ 50	✓
Q3 **Even Difference** > **Coding**	1 min 58 sec	0/ 50	✗

QUESTION 1

Correct Answer

Score 6

Shortest Substring > Coding

Strings | Loops | Easy

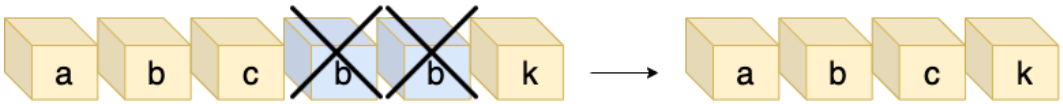
QUESTION DESCRIPTION

Given a string *s* of length *n*. The task is to find the length of the shortest substring, which upon deletion, makes the resultant string to be consisting of distinct characters only.

A substring is a contiguous sequence of characters within a string. When a substring is deleted, one needs to merge the rest of the character blocks of the string(s). If no substring needs to be deleted, the answer is 0.

Example

Consider the given string to be *s = "abcbbk"*, delete the substring "bb" in the range [3, 4] to get the remaining string "abck" which consists of distinct characters only. This is the minimum possible length of the string. Hence, the answer is 2.



Function Description

Complete the function *findShortestSubstring* in the editor below.

findShortestSubstring has the following parameter:

s: the given input string

Returns:

int: an integer representing the length of the shortest substring that should be deleted

Constraints

- $1 \leq n \leq 10^5$
- *s consists of lowercase English letters only.*

▼ Input Format For Custom Testing

The first and the only line contains the string *s*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN              FUNCTION
-----              --------
xabbcacpqr    →    s = "xabbcacpqr"
```

Sample Output

```
3
```

Explanation

Given string *s = "xabbcacpqr"*. Considering 0-based indexing, if we delete the substring in the range [3, 5], which is "*bca*". The resulting substring becomes "*xabcpqr*", in which all characters are distinct. This is the minimum length possible. Thus, the answer is 3.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN           FUNCTION
-----           --------
abc        →    s = "abc"
```

Sample Output

```
0
```

Explanation

The given string *s = "abc",* already contains distinct characters only. So no substring needs to be deleted. Hence, the answer is 0.

INTERVIEWER GUIDELINES

▼ Solution

Skills: Loops and counters, I/O, Strings.

Editorial:
Find the first repeating character from the start. Let this be in index l. Find the first repeating character from the end considering the characters in indices [0, l-1] as well. Let this be in index r. The length of the substring to be removed is r - l + 1. Now, we reverse the string and repeat the same technique. The answer is the minimum of the two lengths.
In case we find no repeating characters, the answer is 0.

Optimal Solution:

```
int findMinLength(string s)
{
    int n = s.size();
    bool visited[26];
    for(int i = 0; i < 26; i++)
        visited[i] = false;
    int len = 0;
    for(int i = 0; i < n; i++)
    {
        if(visited[s[i] - 'a'] == true)
        {
            for(int j = n - 1; j >= i; j--)
            {
                if(visited[s[j] - 'a'] == true)
                {
                    len = j - i + 1;
                    break;
                }
                else
                    visited[s[j] - 'a'] = true;
            }
            break;
        }
        else
            visited[s[i] - 'a'] = true;
    }
    return len;
}

int findShortestSubstring(string s)
{
    int ans = findMinLength(s);
    reverse(s.begin(), s.end());
    ans = min(ans, findMinLength(s));
    return ans;
}
```

Tester's Solution:

```
def findFirstRepeating(s, found):
    for i in range(len(s)):
        if s[i] in found:
            return i
        found.add(s[i])
    return len(s)

def findShortestSubstring(s):
    n = len(s)
    l = findFirstRepeating(s, set())
    r = n - 1 - findFirstRepeating(s[::-1], set(s[:l+1]))
```

```
    ans = r - l + 1
    l = findFirstRepeating(s[::-1], set())
    r = n - 1 - findFirstRepeating(s, set(s[::-1][:l+1]))
    ans = min(ans, r - l + 1)
    return ans
```

▼ Complexity Analysis

Time Complexity - O(n) where n is the number of elements in array.
Space Complexity - O(n) where n is the number of elements in array.

CANDIDATE ANSWER

Language used: **Java 8**

```
1
2  class Result {
3
4      /*
5       * Complete the 'findShortestSubstring' function below.
6       *
7       * The function is expected to return an INTEGER.
8       * The function accepts STRING s as parameter.
9       */
10
11     public static int findShortestSubstring(String s) {
12         String vec[] = s.split("");
13         int cont = 0;
14         int indiceActual = 0;
15         int i = 0;
16         String actual = "";
17         while(i<s.length()){
18             if(i == indiceActual){
19                 actual = vec[i];
20             }
21             else if(vec[i].equals(actual)){
22                 cont++;
23             }
24             if(i == vec.length - 1){
25                 i = indiceActual+1;
26                 indiceActual = i;
27             }else{
28                 i++;
29             }
30
31         }
32         return cont;
33     }
34
35 }
36
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Success	1	0.0779 sec	23.3 KB

TestCase 1	Easy	Sample case	Success	1	0.0672 sec	23.5 KB
TestCase 2	Easy	Sample case	Success	1	0.1954 sec	23.6 KB
TestCase 3	Easy	Hidden case	Wrong Answer	0	0.0891 sec	24.2 KB
TestCase 4	Easy	Hidden case	Wrong Answer	0	0.0944 sec	23.5 KB
TestCase 5	Easy	Hidden case	Wrong Answer	0	0.0598 sec	23.6 KB
TestCase 6	Easy	Hidden case	Wrong Answer	0	0.1666 sec	25.3 KB
TestCase 7	Easy	Hidden case	Wrong Answer	0	0.0884 sec	24.3 KB
TestCase 8	Easy	Hidden case	Success	3	0.0909 sec	23.5 KB
TestCase 9	Easy	Hidden case	Terminated due to timeout	0	4.0059 sec	32.7 KB
TestCase 10	Easy	Hidden case	Wrong Answer	0	0.1066 sec	24.3 KB
TestCase 11	Easy	Hidden case	Terminated due to timeout	0	4.0059 sec	32.7 KB
TestCase 12	Easy	Hidden case	Wrong Answer	0	0.0993 sec	23.4 KB
TestCase 13	Easy	Hidden case	Terminated due to timeout	0	4.0062 sec	32.7 KB
Testcase 14	Easy	Hidden case	Terminated due to timeout	0	4.0053 sec	32.6 KB

No Comments

QUESTION 2

Correct Answer

Score 6

No Pairs Allowed › Coding

Strings | Easy | Algorithms | Problem Solving | Interviewer Guidelines

QUESTION DESCRIPTION

For each word in a list of words, if any two adjacent characters are equal, change one of them. Determine the minimum number of substitutions so the final string contains no adjacent equal characters.

Example

words = ['add', 'boook', 'break']

1. *'add':* change one *d* (1 change)
2. *'boook':* change the middle *o* (1 change)
3. *'break':* no changes are necessary (0 changes)

The return array is *[1,1,0]*.

Function Description

Complete the function *minimalOperations* in the editor below.

minimalOperations has the following parameter(s):

string words[n]: an array of strings

Returns:

int[n]: each element *i* is the minimum substitutions for *words[i]*

Constraints

- $1 \le n \le 100$
- $2 \le length\ of\ words[i] \le 10^5$
- Each character of *words[i]* is in the range *ascii[a-z]*.

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the size of the array *words*.
Each of the next *n* lines contains a string *words[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         Function Parameters
-----         -------------------
5          →  words[] Size = 5
ab         →  words[] = [ 'ab','aab','abb', 'abab','abaaaba' ]
aab
abb
abab
abaaaba
```

Sample Output 0

```
0
1
1
0
1
```

Explanation 0

- *words = 'ab'* is already acceptable, so *0* replacements are needed.
- *words = 'aab'* Replace an *'a'* with an appropriate character so *1* replacement.
- *words = 'abb'* is not acceptable. Replace a *'b'* with an appropriate character, again *1* replacement.
- *words = 'abab'* is already acceptable so *0* replacements are needed.
- *words = 'abaaaba'* is not acceptable. Replace the middle *'a'* in *'aaa'*, *1* replacement.

The return array is *[0, 1, 1, 0, 1]*.

INTERVIEWER GUIDELINES

▼ Hint 1

As you iterate through the string, which character(s) need to be tested for equivalence? For each character check only characters adjacent to it on the left.

▼ Hint 2

If you replace a character, can you always assume the replacement differs from the character to its right as well?
Why, and how can you use this fact?

The characters left and right can either be the same or different. There are 25 or 24 letters available in all cases.

This allows you to skip over the next character after a replacement.

▼ Solution

Concepts covered: This problem covers the concepts of strings and arrays.

Optimal Solution:

For each string, start with the character at index 1. Compare each character to the one to its left, with one exception. If the two letters are equal, assume the character to its left remains the same and the current character is replaced. It can always be replaced with a character different from both adjacent characters, left and right. The next character after a replacement can be skipped.

```
def minimalOperations(words):
    ans = []
    for w in words:
        count = 0
        i = 1
        while i < len(w):
            # test for match
            if w[i] == w[i-1]:
                # yes: increment counter and skip the next character
                count += 1
                i += 2
            else:
                # no: move to the next character
                i += 1
        ans.append(count)
    return ans
```

Sub-optimal approach: For each string, iterate its characters, checking if they are equal to the one to their left. If the characters match, replace the current character with '#'. For example: string "abbca". We check pairs one by one, 'ab', 'bb', here characters are the same, so we replace the second character with '#'. Continue checking symbols one by one, '#c', 'ca'. This finishes the process.

```
def minimalOperations(words):

    ans = []

    for i in range(len(words)):
        # replacement counter
        cur_ans = 0
        # convert the string to a list so it is mutable
        cur_word = list(words[i])

        for j in range(1, len(words[i])):
            # if characters match, replace the current character
            if cur_word[j-1] == cur_word[j]:
                # replace with a character guaranteed to be different
from the next character
                cur_word[j] = "#"
                cur_ans += 1

        ans.append(cur_ans)

    return ans
```

Error Handling:

1. The case of a zero length string must be handled separately.

▼ Complexity Analysis

Time Complexity - O(N) where N is the total number of characters in all words.

Accessing all characters in all words requires O(N) time

Space Complexity - O(1) - For the optimal solution only two integer variables are required.

CANDIDATE ANSWER

Language used: **Java 8**

```
class Result {

    /*
     * Complete the 'minimalOperations' function below.
     *
     * The function is expected to return an INTEGER_ARRAY.
     * The function accepts STRING_ARRAY words as parameter.
     */

    public static List<Integer> minimalOperations(List<String> words) {
        List<Integer> listaResultado = new ArrayList<>();
        for(String w : words){
            String vec[] = w.split("");
            int cont = 0;
            String ant = "";
            for(int i = 0; i< vec.length;i++){
                if(!ant.equals(vec[i])){
                    ant = vec[i];
                }else {
                    cont++;
                }
            }
            listaResultado.add(cont);
        }
        return listaResultado;

    }

}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Wrong Answer	0	0.1193 sec	29.7 KB
TestCase 1	Easy	Sample case	Wrong Answer	0	0.138 sec	29.8 KB
TestCase 2	Easy	Sample case	Wrong Answer	0	0.1419 sec	29.8 KB
TestCase 3	Easy	Sample case	Success	6	0.1179 sec	29.9 KB
TestCase 4	Easy	Hidden case	Wrong Answer	0	0.1569 sec	29.7 KB
TestCase 5	Easy	Sample case	Wrong Answer	0	0.247 sec	39.6 KB
TestCase 6	Easy	Hidden case	Wrong Answer	0	0.2403 sec	40.5 KB
TestCase 7	Easy	Hidden case	Wrong Answer	0	0.5074 sec	121 KB
TestCase 8	Easy	Hidden case	Wrong Answer	0	0.4875 sec	125 KB
TestCase 9	Easy	Hidden case	Wrong Answer	0	0.5541 sec	126 KB
TestCase 10	Easy	Hidden case	Wrong Answer	0	0.7175 sec	182 KB
TestCase 11	Easy	Hidden case	Wrong Answer	0	0.8884 sec	159 KB
TestCase 12	Easy	Hidden case	Wrong Answer	0	0.9603 sec	161 KB
TestCase 13	Easy	Hidden case	Wrong Answer	0	0.9506 sec	173 KB

No Comments

QUESTION 3

Wrong Answer

Score 0

Even Difference › Coding

Easy | Arrays | Greedy Algorithms | Sorting

QUESTION DESCRIPTION

Consider every subsequence of an array of integers.

- Sort the subsequence in increasing order.
- Determine the sum of differences of elements in the subsequence.
- Return the length of the longest subsequence where this sum is even.

Example

Given *n = 4* elements and *arr = [2, 4, 1, 7]*, these are some of the subsequences.

Subsequence	Sorted Subsequence	Sum of diff of Adjacent elements	Is Valid	Length
[2, 4, 1]	[1, 2, 4]	1 + 2 = 3 (Odd)	No	3
[2, 1, 7]	[1, 2, 7]	1 + 5 = 6 (Even)	Yes	3
[2, 4, 1, 7]	[1, 2, 4, 7]	1 + 2 + 3 = 6 (Even)	Yes	4
[2, 1]	[1, 2]	1 (Odd)	No	2

We can see that the maximum possible length of a valid subsequence is 4.

Function Description

Complete the function *findLongestSubsequence* in the editor below.

findLongestSubsequence has the following parameter(s):

int arr[n]: an array of integers

Returns

int: the length of the longest subsequence as described

Constraints

- $3 \le n \le 10^5$
- $0 \le arr[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *arr*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *arr[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       ----------
7     →     arr size [] n = 7
7     →     arr = [7, 5, 6, 2, 3, 2, 4]
5
6
```

```
2
3
2
4
```

Sample Output

```
6
```

Explanation

Consider the subsequence [5, 6, 2, 3, 2, 4].

- arrange the subsequence in ascending order, 2, 2, 3, 4, 5, 6
- the differences are 0,1,1,1,1
- 0+1+1+1+1 = 4

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN     FUNCTION
-----     ----------
4     →   arr size[] n = 4
1     →   arr = [1, 3, 5, 7]
3
5
7
```

Sample Output

```
4
```

Explanation

The entire array can be used.

- arrange the subsequence in ascending order, 1,3,5,7
- the adjacent differences are 2, 2, 2
- 2+2+2 = 6

INTERVIEWER GUIDELINES

▼ Solution

Skills: Greedy, Sorting

Optimal Solution:

To have the sum of adjacent elements even, the difference between the first and last elements has to be even in the sorted array.

Sort the array.

Find the first and last occurrence of the even number in the array.

Find the first and last occurrence of the odd number in the array.

The answer is the maximum difference between the first and last occurrence of the even number in the array and the first and last occurrence of the odd number in the array.

```
def findLongestSubsequence(arr):
    arr.sort()
    firstOdd = firstEven = lastOdd = lastEven = -1
    for i in range(len(arr)):
        if arr[i] % 2 == 0:
            lastEven = i
```

```
            if firstEven == -1:
                firstEven = i
        else:
            lastOdd = i
            if firstOdd == -1:
                firstOdd = i
    ans = 1
    if firstOdd != -1:
        ans = max(ans, lastOdd - firstOdd + 1)
    if firstEven != -1:
        ans = max(ans, lastEven - firstEven + 1)
    return ans
```

▼ Complexity Analysis

Time Complexity - O(n)

The array is traversed once, so the time complexity is O(n).

Space Complexity - O(n)

Sorting the array requires O(n) space.

CANDIDATE ANSWER

The candidate did not manually submit any code. The last compiled version has been auto-submitted and the score you see below is for the auto-submitted version.

Language used: **Java 8**

```
1
2  class Result {
3
4      /*
5       * Complete the 'findLongestSubsequence' function below.
6       *
7       * The function is expected to return an INTEGER.
8       * The function accepts INTEGER_ARRAY arr as parameter.
9       */
10
11     public static int findLongestSubsequence(List<Integer> arr) {
12     // Write your code here
13
14     }
15
16 }
17
```

Result: Compilation Failed

Compile Message

```
                Solution.java:27: error: missing return statement
    }
    ^
1 error
```

No Comments

PDF generated at: 25 May 2023 17:34:02 UTC