



You can view this report online at : <https://www.hackerrank.com/x/tests/746399/candidates/13651734/report>

Full Name: Varun Kumar
Email: varun-kumar@live.com
Test Name: Pre-requisite Strings Online Assessment 1 (Coding & Problem solving)
Taken On: 20 Mar 2020 22:53:40 IST
Time Taken: 48 min 3 sec/ 150 min
Work Experience: < 1 years
City: Patna
Student Roll Number: 17BCS2411
Personal Email Address: varun-kumar@live.com
CGPA: 7.2
Contact Number: 8289090667
Stream/Branch: BE-CSE
Section: CIK2
Resume: <https://cdn.hackerrank.com/files/uploads/recruit-resumes/4b389a21-0310-4d03-8ccc-b6f40fd672c4/Resume1.pdf>
Invited by: Jagandeep
Tags Score:
Algorithms 1/175
Arrays 0/50
Core CS 0/75
Data Structures 1/100
Easy 51/150
Hashing 1/50
Implementation 0/50
Medium 0/75
Problem Solving 1/125
Sets 1/50
Strings 51/225

22.7%

51/225

scored in **Pre-requisite Strings Online Assessment 1 (Coding & Problem solving)** in 48 min 3 sec on 20 Mar 2020 22:53:40 IST

Recruiter/Team Comments:

No Comments.

Question Description

Time Taken

Score

Status

Q1 String Reduction > Coding

11 min 2 sec

50/ 50



Q2 Fewest Coins > Coding

37 min 3 sec

1/ 50





QUESTION 1



Correct Answer

Score 50

String Reduction > Coding

Easy

Strings

QUESTION DESCRIPTION

Given a string, reduce it in such a way that all of its substrings are distinct. To do so, you may delete any character of the string at any index. What is the minimum number of deletions needed in order to complete this task?

Note: A substring is a contiguous sequence of characters within a string. It can be formed by deleting some (0 or more) characters from the left of the string and some (0 or more) characters from the right of the string.

For example, let's say the given string is $s = \text{"abab"}$. Currently, the substrings are not distinct—the substring "ab" is found starting at both index 0 and index 2. By deleting $s[2]$ and $s[3]$, the string becomes "ab", where all substrings are distinct. Therefore, the answer is 2 because this required 2 deletions. (Note that "aba" is not acceptable because the character 'a' counts as a substring. In "aba", there are two instances of the substring "a".)

Function Description

Complete the function `getMinDeletions` in the editor below.

`getMinDeletions` has the following parameter(s):

string s : the given string

Returns:

int: the minimum number of deletions needed to make s have only distinct substrings in it

Constraints

- $1 \leq n \leq 10^5$

▼ Input Format For Custom Testing

The first line contains a string, s .

▼ Sample Case 0

Sample Input For Custom Testing

STDIN	Function
-----	-----
abcab	=> s = "abcab"

Sample Output

2

Explanation

By deleting the first 2 characters, the string becomes "cab", which has only distinct substrings in it. Therefore, the answer is 2.

▼ Sample Case 1

Sample Input For Custom Testing

abcabc

Sample Output

3

Explanation

By deleting the characters at indices 0, 4, and 5, the string becomes "bca", which has only distinct substrings in it. Because this required 3 deletions, the answer is 3.

INTERNAL NOTES

For distinct sub-strings, all the characters in the string must be distinct. So we calculate the number of distinct characters and hence our answer equals $n - \text{distinct_characters}$.

```
int getMin(string s) {
    int ans = 0;

    int freq[26] = {0};
    for (int i = 0; s[i]; i++) {
        if (freq[ s[i] - 'a' ] == 0) {
            freq[ s[i] - 'a' ] = 1;
            ans++;
        }
    }
    ans = s.length() - ans;

    return ans;
}
```

Tester's solution:

```
def getMin(s):

    diff_chr = set()

    for i in range(len(s)):
        diff_chr.add(s[i])

    return (len(s) - len(diff_chr))
```

CANDIDATE ANSWER

Language used: **C++14**

```
1  /*
2   * Complete the 'getMinDeletions' function below.
3   *
4   * The function is expected to return an INTEGER.
5   * The function accepts STRING s as parameter.
6   */
7
8  int getMinDeletions(string s) {
9      set<char> x;
10     for(auto const ch : s) {
11         x.insert(ch);
12     }
13     return s.size() - x.size();
14 }
15
16
17
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCOPE	TIME TAKEN	MEMORY USED
----------	------------	------	--------	-------	------------	-------------

TEST CASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	✓ Success	1	0.1271 sec	9.11 KB
TestCase 1	Easy	Sample case	✓ Success	1	0.1259 sec	9.02 KB
TestCase 2	Easy	Sample case	✓ Success	1	0.1046 sec	9 KB
TestCase 3	Easy	Hidden case	✓ Success	2	0.1308 sec	8.87 KB
TestCase 4	Easy	Hidden case	✓ Success	2	0.1062 sec	9.02 KB
TestCase 5	Easy	Hidden case	✓ Success	3	0.1088 sec	9.11 KB
TestCase 6	Easy	Hidden case	✓ Success	3	0.1134 sec	8.96 KB
TestCase 7	Easy	Hidden case	✓ Success	3	0.1968 sec	8.98 KB
TestCase 8	Easy	Hidden case	✓ Success	3	0.133 sec	8.91 KB
TestCase 9	Easy	Hidden case	✓ Success	3	0.0998 sec	8.95 KB
TestCase 10	Easy	Hidden case	✓ Success	5	0.1125 sec	9.2 KB
TestCase 11	Easy	Hidden case	✓ Success	5	0.3622 sec	9.05 KB
TestCase 12	Easy	Hidden case	✓ Success	6	0.1032 sec	9.23 KB
TestCase 13	Easy	Hidden case	✓ Success	6	0.1436 sec	9.33 KB
TestCase 14	Easy	Hidden case	✓ Success	6	0.1261 sec	9.31 KB

No Comments

QUESTION 2



Correct Answer

Score 1

Fewest Coins

Coding

Algorithms

Strings

Data Structures

Problem Solving

Easy

Sets

Hashing

QUESTION DESCRIPTION

An online coin dealer offers bags of coins that are guaranteed to contain at least one full set. Given a string comprised of lowercase letters in the range *ascii[a-z]*, where each letter represents a coin type, determine the length of the shortest substring that contains at least one of each type of coin.

Example:

coins = dabbcabcd

The list of all characters in the string is *[a, b, c, d]*.

Two of the substrings that contain all letters are *dabbc* and *abcd*.

The shortest substring that contains all of the letters is 4 characters long.

Function Description

Complete the function *fewestCoins* in the editor below.

fewestCoins has the following parameter:

string coins: a string

Return

int: the length of the shortest substring that contains at least one of each characters in *coins*

Constraints

- $1 \leq \text{size of coins} \leq 10^5$
- each *coins[i]* is in the set *ascii[a-z]*

▼ Input Format For Custom Testing

The first line contains a string. *coins*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN      Function
-----
bab  →    coins = 'bab'
```

Sample Output

```
2
```

Explanation

"ba" is a substring that contains all the characters in *coins*.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN      Function
-----
asdfkjeghfalawefhaef → coins = 'asdfkjeghfalawefhaef'
```

Sample Output

```
13
```

Explanation

The 11 distinct characters in *coins* are *[a, d, e, f, g, h, j, k, l, s, w]*. The shortest substring with all of the characters is 13 characters long: *sdfkjeghfalaw*.

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: **C++14**

```
1  /*
2   * Complete the 'fewestCoins' function below.
3   *
4   * The function is expected to return an INTEGER.
5   * The function accepts STRING coins as parameter.
6   */
7
8  int fewestCoins(string coins) {
9      // count total distinct coins
10     set<char>s;
11     for(auto const ch : coins) {
12         s.insert(ch);
13     }
14     int numChar = s.size();
15
16     int start = 0, res = INT_MAX;
17     int count = 0;
18     map<int, int>m;
19     for(int j=0;j<coins.size();j++) {
20         m[j]++;
21         if(m[j] == 1) {
22             count++;
23         }
24     }
```

```

24         if(count == numChar) {
25             while(m[coins[start]] > 1) {
26                 if(m[coins[start]] > 1) {
27                     m[coins[start]]--;
28                 }
29                 start++;
30             }
31         }
32         int len_window = j - start + 1;
33         if(res > len_window) {
34             res = len_window;
35         }
36     }
37     return res+1;
38 }
39
40

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	 Success	1	0.1031 sec	8.99 KB
TestCase 1	Easy	Sample case	 Wrong Answer	0	0.1097 sec	9 KB
TestCase 2	Easy	Sample case	 Wrong Answer	0	0.1242 sec	9.04 KB
TestCase 3	Easy	Sample case	 Wrong Answer	0	0.1041 sec	9.28 KB
TestCase 4	Easy	Sample case	 Wrong Answer	0	0.1064 sec	9.36 KB
TestCase 5	Easy	Hidden case	 Wrong Answer	0	0.1098 sec	9.53 KB
TestCase 6	Medium	Hidden case	 Wrong Answer	0	0.1143 sec	9.64 KB
TestCase 7	Medium	Hidden case	 Wrong Answer	0	0.1233 sec	9.9 KB
TestCase 8	Medium	Hidden case	 Wrong Answer	0	0.1377 sec	10.2 KB
TestCase 9	Hard	Hidden case	 Wrong Answer	0	0.1156 sec	10.3 KB
TestCase 10	Hard	Hidden case	 Wrong Answer	0	0.1234 sec	10.3 KB
TestCase 11	Hard	Hidden case	 Wrong Answer	0	0.1462 sec	10.5 KB
TestCase 12	Hard	Hidden case	 Wrong Answer	0	0.1551 sec	13.8 KB

No Comments

QUESTION 3



Not Submitted

Score 0

Good Binary Strings > Coding Strings Medium Algorithms Problem Solving Core CS

QUESTION DESCRIPTION

We define the following:

- A *binary string* is a string consisting only of 0's and/or 1's. For example, 01011, 1111, and 00 are all binary strings.
- The *prefix* of a string is any substring of the string that includes the beginning of the string. For example, the prefixes of 11010 are 1, 11, 110, 1101, and 11010.

We consider a non-empty binary string to be *good* if the following two conditions are true:

- The number of 0's is equal to the number of 1's.
- For every prefix of the binary string, the number of 1's should not be less than the number of 0's.

For example, 11010 is not good because it doesn't have an equal number of 0's and 1's, but 110100 is good because it satisfies both of the above conditions.

A good string can contain multiple good substrings. If two *consecutive substrings* are good, then we can *swap* the substrings as long as the resulting string is still a good string. Given a good binary string, *binString*, perform zero or more swap operations on its consecutive good substrings such that the resulting string is as *lexicographically large* as possible. Two substrings are considered to be consecutive if the last character of the first substring occurs exactly one index before the first character of the second substring.

For example, if we look at the good binary string *binString* = *1010111000*, we see two good binary substrings, *1010* and *111000* among others. If we swap these two substrings we get a larger value: *1110001010*. This is the largest possible good substring that can be formed.

Function Description

Complete the function *largestGood* in the editor below. The function must return a string denoting the lexicographically largest possible good string that can be formed by performing zero or more swap operations on consecutive good substrings of *binString*.

largestGood has the following parameter(s):

binString: a string

Constraints

- Each character of *binString* $\in \{01\}$.
- $1 \leq |binString| \leq 50$
- *binString* is a good string.

▼ Input Format For Custom Testing

The only line of input contains the string *binString*.

▼ Sample Case 0

Sample Input 0

STDIN	Function Parameters
-----	-----
110111000 →	binString = "110111000"

Sample Output 0

11100100

Explanation 0

Given the good string *binString* = *110111000*, we can choose two consecutive good substrings, *10* and *1100*, to swap such that the resultant string, *str* = *11100100*, is the lexicographically largest good string possible.

▼ Sample Case 1

Sample Input 1

STDIN	Function Parameters
-----	-----
1100 →	binString = "1100"

Sample Output 1

1100

Explanation 1

The only good substring of *binString* is *1100*. So none of the operations can be applied on the string.

▼ Sample Case 2

Sample Input For Custom Testing

STDIN	Function Parameters
-----	-----

```
1101001100 → binString = "1101001100"
```

Sample Output

```
1101001100
```

Explanation

The only consecutive good substrings of *binString* are *110100* and *1100*. Note that *100* is not a good substring because it contains more zeroes than ones. If we were to swap them, it would result in a lexicographically smaller string. Thus, *binString* is already the lexicographically largest good string that can be formed.

CANDIDATE ANSWER

i This candidate has not answered this question.

No Comments

QUESTION 4



Not Submitted

Score 0

Balancing Parentheses > Coding

Easy

Implementation

Data Structures

Strings

Arrays

Algorithms

QUESTION DESCRIPTION

Given a string that consists of only two types of characters: '(' and ')', balance the parentheses by inserting either a '(' or a ')' as many times as necessary. Determine the minimum number of characters that must be inserted.

Example

```
s = '(()))'
```

To make it a valid sequence, insert a '(' at the beginning of the string, resulting in "((()))". The string is balanced after 1 insertion.

Function Description

Complete the function *getMinOperations* in the editor below. The function must return the minimum number of operations needed to make the parentheses sequence valid.

getMinOperations has the following parameter(s):

string s: a string of parentheses

Return

int: the minimum number of insertions required to balance the parentheses

Constraints

- $1 \leq \text{length of } s \leq 10^5$

▼ Input Format For Custom Testing

The first line contains a string, *s*, the initial parentheses sequence.

▼ Sample Case 0

Sample Input For Custom Testing

STDIN	Function
-----	-----
())	→ s = '())'

Sample Output

2

Explanation

Insert a '(' 2 times at the beginning of the string to make it valid: "((()))".

▼ Sample Case 1

Sample Input For Custom Testing

STDIN	Function
-----	-----
()()	→ s = '()()'

Sample Output

0

Explanation

The sequence is already valid, so no insertions are needed.

INTERNAL NOTES

We iterate through the string and maintain a balance of the parenthesis and store the minimum amount it touches throughout the string. That minimum value is the number of '(' we add in the beginning of the string and then the balance of the string at the end is added.

```
int getMin(string s) {
    int ans = 0;

    int bal = 0, mi = 0;
    for (int i = 0; s[i]; i++) {
        if (s[i] == '(') {
            bal++;
        }
        else {
            bal--;
        }
        mi = min(mi, bal);
    }
    ans = -mi + (bal - mi);

    return ans;
}
```

Tester's code:

```
def getMin(s):
    n = len(s)
    assert 1 <= n <= 10 ** 5
    for i in s:
        assert i == '(' or i == ')'
    st = []
    for i in s:
        if i == '(':
            st.append(0)
        else:
            if len(st) > 0 and st[-1] == 0:
                del st[-1]
            else:
                st.append(1)
    # return len(s)
    return len(st)
```

CANDIDATE ANSWER

 This candidate has not answered this question.

No Comments

PDF generated at: 20 Mar 2020 19:56:24 UTC