You can view this report online at : https://www.hackerrank.com/x/tests/756316/candidates/57454949/report

Full Name:	Ferney Tenorio Basto
Email:	ferney.tenorio@ukuepa.com
Test Name:	**20200331 Practicante 2**
Taken On:	26 Oct 2023 20:45:13 -05
Time Taken:	172 min 30 sec/ 180 min
Work Experience:	< 1 years
Invited by:	Carlos
Invited on:	26 Oct 2023 20:08:44 -05
Skills Score:	Problem Solving (Basic) 150/150 Problem Solving (Intermediate) 50/50
Tags Score:	Algorithms 100/100 Data Structures 150/150 Dynamic Programming 50/50 Easy 200/200 Hash Map 100/100 Interviewer Guidelines 50/50 Problem Solving 150/150 Sets 50/50 Strings 150/150 Theme: E-commerce 50/50

100%

200/200

scored in **20200331 Practicante 2** in 172 min 30 sec on 26 Oct 2023 20:45:13 -05

Recruiter/Team Comments:

No Comments.

Plagiarism flagged

We have marked questions with suspected plagiarism below. Please review it in detail here - https://www.hackerrank.com/x/tests/756316/candidates/57454949/report

	Question Description	Time Taken	Score	Status
Q1	Shortest Substring Containing Characters > **Coding**	1 hour 23 min 2 sec	50/ 50	✓
Q2	Fun with Anagrams > **Coding**	19 min 9 sec	50/ 50	✓
Q3	Construction Management > **Coding**	51 min 55 sec	50/ 50	!

QUESTION 1

Correct Answer

Score 50

Shortest Substring Containing Characters › Coding

Algorithms Strings Data Structures Problem Solving Easy Sets Theme: E-commerce Hash Map

QUESTION DESCRIPTION

Given a string comprised of lowercase letters in the range *ascii[a-z]*, find the length shortest substring that contains at least one of each of the letters in the string.

Example

givenString = 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. Return *4* as the answer.

Function Description

Complete the function *shortestSubstring* in the editor below.

shortestSubstring has the following parameter(s):
string givenString: the given string
Returns:
int: the length of the shortest substring that contains at least one of each character in *givenString*

Constraints

- $1 \le$ *size of givenString* $\le 10^5$
- each *givenString[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

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

Sample Output

```
2
```

Explanation

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

▼ Sample Case 1

Sample Input

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

Sample Output

```
13
```

Explanation

The *11* distinct characters in *givenString* 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

Language used: **Python 3**

```python
#
# Complete the 'shortestSubstring' function below.
#
# The function is expected to return an INTEGER.
# The function accepts STRING givenString as parameter.
#
from collections import defaultdict as dictionary
def shortestSubstring(givenString):

    longitud=len(givenString)
    pDistinto= len(set([i for i in givenString]))

    iniciar=0
    longitudMinima=longitud
    total=0
    comienzo=0
    cuentaActual=dictionary(lambda:0)

    for i in range(longitud):
        cuentaActual[givenString[i]]+=1
        if cuentaActual[givenString[i]]==1:
            total+=1
        if total==pDistinto:
            while cuentaActual[givenString[comienzo]]>1:
                if cuentaActual[givenString[comienzo]]>1:
                    cuentaActual[givenString[comienzo]]-=1
                comienzo+=1
            nueva_dimension=i-comienzo+1
            if longitudMinima>nueva_dimension:
                longitudMinima=nueva_dimension
                iniciar=comienzo
    return len(givenString[iniciar:iniciar+longitudMinima])
    # Write your code here
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	✓ Success	1	0.0679 sec	10.7 KB
TestCase 1	Easy	Sample case	✓ Success	1	0.1531 sec	10.7 KB
TestCase 2	Easy	Sample case	✓ Success	1	0.0568 sec	10.8 KB
TestCase 3	Easy	Sample case	✓ Success	3	0.0787 sec	10.8 KB
TestCase 4	Easy	Sample case	✓ Success	3	0.1096 sec	10.7 KB
TestCase 5	Easy	Hidden case	✓ Success	3	0.0772 sec	10.9 KB
TestCase 6	Medium	Hidden case	✓ Success	4	0.0601 sec	10.7 KB
TestCase 7	Medium	Hidden case	✓ Success	4	0.0536 sec	10.8 KB
TestCase 8	Medium	Hidden case	✓ Success	5	0.0762 sec	10.8 KB
TestCase 9	Hard	Hidden case	✓ Success	5	0.0621 sec	10.9 KB
TestCase 10	Hard	Hidden case	✓ Success	5	0.0726 sec	10.9 KB
TestCase 11	Hard	Hidden case	✓ Success	5	0.105 sec	11 KB

TestCase 11	Hard	Hidden case	✓ Success	5	0.105 sec	11 KB
TestCase 12	Hard	Hidden case	✓ Success	10	0.0988 sec	11.9 KB

No Comments

QUESTION 2

✓

Correct Answer

Score 50

Fun with Anagrams › Coding

Data Structures | Strings | Problem Solving | Easy | Interviewer Guidelines | Hash Map

QUESTION DESCRIPTION

Two strings are anagrams if they are permutations of each other. In other words, both strings have the same size and the same characters. For example, "aaagmnrs" is an anagram of "anagrams". Given an array of strings, remove each string that is an anagram of an earlier string, then return the remaining array in sorted order.

Example

str = ['code', 'doce', 'ecod', 'framer', 'frame']

- "*code*" and "*doce*" are anagrams. Remove "doce" from the array and keep the first occurrence *"code"* in the array.
- *"code"* and "*ecod*" are anagrams. Remove "ecod" from the array and keep the first occurrence *"code"* in the array.
- *"code"* and "*framer*" are not anagrams. Keep both strings in the array.
- "*framer*" and "*frame*" are not anagrams due to the extra *'r'* in *'framer'*. Keep both strings in the array.
- Order the remaining strings in ascending order: *["code","frame","framer"].*

Function Description

Complete the function *funWithAnagrams* in the editor below.

funWithAnagrams has the following parameters:

string text[n]: an array of strings

Returns:

string[m]: an array of the remaining strings in ascending alphabetical order,.

Constraints

- $0 \le n \le 1000$
- $0 \le m \le n$
- $1 \le$ length of $text[i] \le 1000$
- Each string *text[i]* is made up of characters in the range ascii[a-z].

▼ Input Format For Custom Testing

The first line contains an integer, *n*, that denotes the number of elements in *text*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains a string that describes *text[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
4        →  n = 4
code     →  text = ["code","aaagmnrs","anagrams","doce"]
aaagmnrs
```

```
anagrams
doce
```

Sample Output

```
aaagmnrs
code
```

Explanation

- "*code*" and "*doce*" are anagrams. Remove "*doce*" and keep the first occurrence *"code"* in the array.
- *"aaagmnrs"* and "*anagrams*" are anagrams. Remove "a*nagrams*" and keep the first occurrence *"aaagmnrs"* in the array.
- Order the remaining strings in ascending order: *["aaagmnrs", "code"].*

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN     Function
-----     --------
4      →  n = 4
poke   →  text = ["poke","pkoe","okpe","ekop"]
pkoe
okpe
ekop
```

Sample Output

```
poke
```

Explanation

- *"poke*" and *"pkoe"* are anagrams. Remove *"pkoe"* and keep the first occurrence *"poke"* in the array.
- *"poke"* and *"okpe"* are anagrams. Remove *"okpe"* and keep the first occurrence *"poke"* in the array.
- *"poke"* and *"ekop"* are anagrams. Remove *"ekop"* and keep the first occurrence *"poke"* in the array.
- Order the remaining strings in ascending order: *["poke"].*

INTERVIEWER GUIDELINES

▼ Hint 1

What is an efficient way of comparing mixed up characters between 2 strings? Answer: Sort the characters before comparing.

▼ Hint 2

What is an efficient data structure for checking whether the sorted characters has been seen? Answer: A hash map of some kind. The best from a memory standpoint is a set that only allows one occurrence of a value.

▼ Solution

Concepts covered: Sorting, data type conversions, use of hash maps

Optimal Solution:

For each string, convert it to a hashable sorted list of characters. See if it has already been seen. If not, store the string to the answer array and the sorted list to the hash table. Finally, sort the resulting list of strings alphabetically.

```
def funWithAnagrams(text):
    # Write your code here
    # a set of words as sorted character tuples
    cs = set()
    # words remaining
    ans = []
    for t in text:
        # store text as a tuple of sorted characters
        # hash map requires immutable type
        tt = tuple(sorted(list(t)))
        # if the character tuple has not been seen
        if not tt in cs:
            ans.append(t)
            cs.add(tt)
    # the results are sorted alphabetically
    return sorted(ans)
```

Error Handling: Hash tables require immutable types. The sorted list must be cast as a valid type for hashing.

▼ Complexity Analysis

Time Complexity - O(N log N).

All characters must be sorted, so N is the sum of the lengths of all strings.

Space Complexity - O(N)

Space is required for a hash map. The worst case is that there are no anagrams, so all strings will be stored in the hash map.

CANDIDATE ANSWER

Language used: **Python 3**

```python
#
# Complete the 'funWithAnagrams' function below.
#
# The function is expected to return a STRING_ARRAY.
# The function accepts STRING_ARRAY text as parameter.
#
def validarAnagramas(palabra,lista):
    for i in lista:
        if(sorted(palabra)==sorted(i)):
            return True
    return False


def funWithAnagrams(text):
    # Write your code here
    limites=len(text)
    text.reverse()

    copiaTextoLista=list(text)

    contador=0
    for i in range(0,limites):
        if text[i+1:] and validarAnagramas(text[i],text[i+1:]):
            copiaTextoLista.pop(i-contador)
            contador=contador+1
    return sorted(copiaTextoLista)
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	✓ Success	2	0.0717 sec	10.9 KB
TestCase 1	Easy	Sample case	✓ Success	2	0.1129 sec	10.7 KB
TestCase 2	Easy	Sample case	✓ Success	2	0.0744 sec	10.8 KB
TestCase 5	Easy	Sample case	✓ Success	4	0.0772 sec	10.9 KB
TestCase 6	Medium	Hidden case	✓ Success	6	0.092 sec	10.8 KB

TestCase 7	Medium	Sample case	✓ Success	8	0.1331 sec	10.9 KB
TestCase 9	Hard	Hidden case	✓ Success	12	0.0959 sec	10.5 KB
TestCase 11	Hard	Hidden case	✓ Success	14	0.0821 sec	10.8 KB

No Comments

QUESTION 3

Needs Review

Score 50

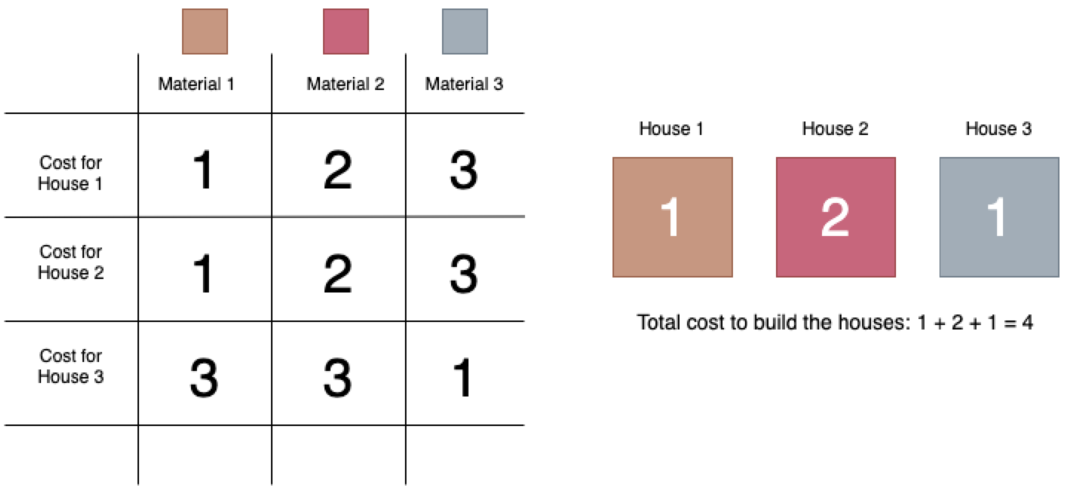
Construction Management › Coding

Algorithms | Data Structures | Dynamic Programming | Easy

QUESTION DESCRIPTION

A construction company is building a new neighborhood, and they are currently working on the design. Each house will be built using one of three main materials (e.g., wood, brick, or concrete), but no side-by-side houses can be made of the same material. Because each house will be of varying size and complexity, the cost of the materials for each house varies. Given the cost of using each material for each house, what is the minimum cost needed to complete the neighborhood?

For example, let's say there are *n = 3* houses to be built. Also, *cost = [[1, 2, 3], [1, 2, 3], [3, 3, 1]]*, denoting the cost of materials for each of the 3 houses. The minimum cost to build all the houses is 4, as seen below:



For the first house, the cheapest material is the first one, which costs 1. For the second house, the materials cost the same as with the first house, but the same material can't be used because the houses are side by side. The next best option is the second material, which costs 2. Finally, the cheapest material for the third house is the third material, which costs 1. Therefore, the total cost to build all the houses is 1 + 2 + 1 = 4.

Function Description

Complete the function *minCost* in the editor below.

minCost has the following parameter:

int *cost[n][3]*: a 2-dimensional array of integers where *cost[i][j]* denotes the cost of using the j^{th} material on the i^{th} house

Returns:

int: the minimum cost to build all the houses in the neighborhood

Constraints

- $1 \le n \le 100$

- 0 ≤ *cost[i][j]* ≤ 100

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the size of the array *cost*.
The next line always contains the number 3, denoting the number of columns in each *cost[i]*.
Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains three space-separated integers that denote the costs of each material, *cost[i][j]*, for each house.

▼ Sample Case 0

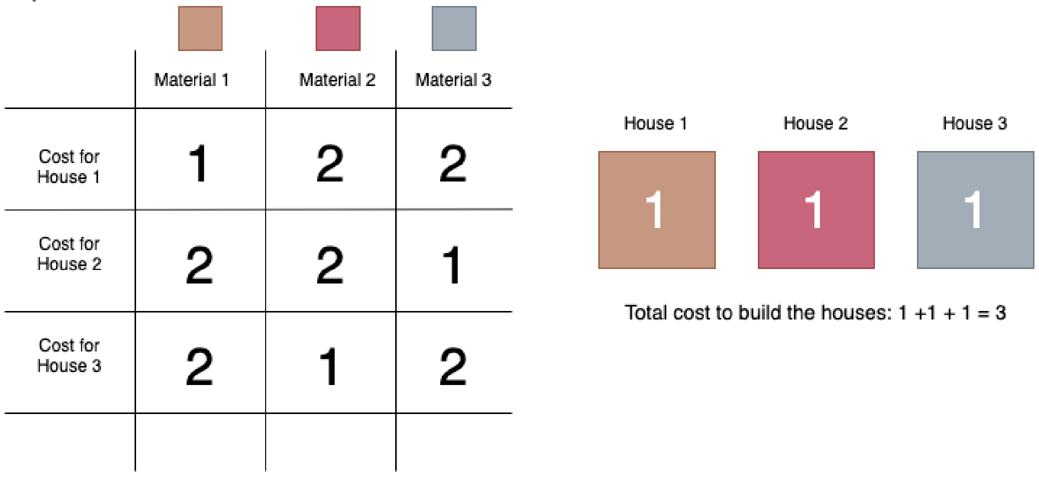
Sample Input For Custom Testing

```
STDIN     Function
-----     --------
3      =>  n = 3
3      =>  number of columns in cost = 3
1 2 2 =>  cost = [[1, 2, 2], [2, 2, 1], [2, 1, 2]]
2 2 1
2 1 2
```

Sample Output

```
3
```

Explanation



Here, it is possible to select the cheapest material for each house because it is different for each house. The cost will be 1 for the first house, 1 for the second house, and 1 for the third house, giving a total cost of 3.

▼ Sample Case 1

Sample Input For Custom Testing

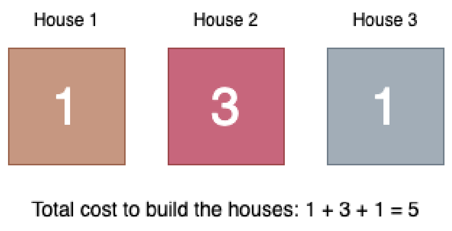
```
STDIN     Function
-----     --------
3      => n = 3
3      => number of columns in cost = 3
1 2 2 => cost = [[1, 2, 2], [2, 3, 3], [3, 3, 1]]
2 3 3
3 3 1
```

Sample Output

```
5
```

Explanation

	Material 1	Material 2	Material 3
Cost for House 1	1	2	2
Cost for House 2	2	3	3
Cost for House 3	3	3	1



One optimal solution is to choose the first material for the first house (which costs 1), the second material for the second house (which costs 3), and the third material for the third house (which costs 1), giving a total cost of 3. Note that even though the first material is cheaper for the second house, it can't be used because the first house, which is next-door, is already using that material.

CANDIDATE ANSWER

Language used: **Java 8**

```java

class Result {

    /*
     * Complete the 'minCost' function below.
     *
     * The function is expected to return an INTEGER.
     * The function accepts 2D_INTEGER_ARRAY cost as parameter.
     */

    public static int minCost(List<List<Integer>> cost) {
    // Write your code here
    int logitud,costoMinimo;
    Integer[][] const1= new Integer[cost.size()][];
    Integer[] const2= new Integer[0];

    for(int i=0;i<cost.size();i++){
        const1[i]=cost.get(i).toArray(const2);
    }

    for(int i=1;i<const1.length;i++){
        const1[i][0]+=Math.min(const1[i-1][1],const1[i-1][2]);
        const1[i][1]+=Math.min(const1[i-1][0],const1[i-1][2]);
        const1[i][2]+=Math.min(const1[i-1][0],const1[i-1][1]);

    }

    logitud=const1.length;
    costoMinimo=Math.min(const1[logitud-1][0],Math.min(const1[logitud-1]
[1],const1[logitud-1][2]));

    return costoMinimo;
    }

}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	✓ Success	1	0.1244 sec	30 KB
TestCase 1	Easy	Sample case	✓ Success	1	0.1351 sec	30 KB
TestCase 2	Easy	Sample case	✓ Success	1	0.1165 sec	29.8 KB
TestCase 3	Easy	Hidden case	✓ Success	3	0.1401 sec	29.8 KB
TestCase 4	Easy	Hidden case	✓ Success	3	0.1424 sec	29.9 KB
TestCase 5	Easy	Hidden case	✓ Success	3	0.2072 sec	29.9 KB
TestCase 6	Medium	Hidden case	✓ Success	5	0.143 sec	30.1 KB
TestCase 7	Medium	Hidden case	✓ Success	5	0.1227 sec	29.9 KB
TestCase 8	Medium	Hidden case	✓ Success	5	0.1586 sec	30.2 KB
TestCase 9	Hard	Hidden case	✓ Success	5	0.1406 sec	29.8 KB
TestCase 10	Hard	Hidden case	✓ Success	6	0.1568 sec	29.8 KB
TestCase 11	Hard	Hidden case	✓ Success	6	0.171 sec	30.3 KB
TestCase 12	Hard	Hidden case	✓ Success	6	0.148 sec	30.1 KB

No Comments

QUESTION 4

✓ Correct Answer

Score 50

Last and Second-Last › Coding   Easy   Problem Solving   Strings

QUESTION DESCRIPTION

Given a string, create a new string made up of its last two letters, reversed and separated by a space.

Example
Given the word '*bat*', return '*t a*'.

Function Description
Complete the function *lastLetters* in the editor below.

lastLetters has the following parameter(s):
 string word: a string to process

Returns:
 string: a string of two space-separated characters

Constraint

- *2 ≤* length of *word ≤ 100*

▼ Input Format for Custom Testing

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

The line contains a string, *word*.

▼ Sample Case 0

Sample Input

```
STDIN     Function
-----     -----
APPLE  →  word = 'APPLE'
```

Sample Output

```
E L
```

Explanation

The last letter in '*APPLE*' is *E* and the second-to-last letter is *L*, so return *E L*.

CANDIDATE ANSWER

Language used: **Python 3**

```python

#
# Complete the 'lastLetters' function below.
#
# The function is expected to return a STRING.
# The function accepts STRING word as parameter.
#

def lastLetters(word):
    recortar=word[-2:]
    lista=[i for i in recortar]
    retornar=str(lista[1]) + ' '+str(lista[0])
    return retornar
    # Write your code here

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample case	✓ Success	1	0.0572 sec	10.7 KB
Testcase 1	Easy	Hidden case	✓ Success	7	0.0566 sec	10.6 KB
Testcase 2	Easy	Hidden case	✓ Success	8	0.0603 sec	10.7 KB
Testcase 3	Easy	Hidden case	✓ Success	8	0.0522 sec	10.8 KB
Testcase 4	Medium	Hidden case	✓ Success	11	0.0675 sec	10.7 KB
Testcase 5	Hard	Hidden case	✓ Success	14	0.0779 sec	10.6 KB
Testcase 6	Easy	Sample case	✓ Success	1	0.0528 sec	10.6 KB

No Comments