You can view this report online at : https://www.hackerrank.com/x/tests/1612898/candidates/53772294/report

Full Name:	이이현호
Email:	swethom1@naver.com
Test Name:	**2023년 하반기 ICT 인턴십 국내과정 코딩테스트**
Taken On:	14 Jul 2023 15:30:07 JST
Time Taken:	351 min 11 sec/ 360 min
Personal Email Address:	swethom1@naver.com
Contact Number:	01075203758
Univ/College Name:	상명대학교
Major:	컴퓨터과학과
Invited by:	Young-Guk
Invited on:	13 Jul 2023 08:48:53 JST
Skills Score:	Problem Solving (Advanced) 1/100 Problem Solving (Basic) 75/100 Problem Solving (Intermediate) 150/150
Tags Score:	Algorithms 201/300 Arrays 1/100 Data Structures 75/75 Dynamic Programming 1/100 Easy 75/100 Hard 1/100 Hash Map 75/75 Implementation 125/125 Interviewer Guidelines 150/175 Medium 150/150 Prefix Sum 25/50 Problem Solving 201/300 Real-World 25/50 Sets 50/50 Strings 75/75

64.6%

226/350

scored in **2023년 하반기 ICT 인턴십 국내과정 코딩테스트** in 351 min 11 sec on 14 Jul 2023 15:30:07 JST

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/1612898/candidates/53772294/report

	Question Description	Time Taken	Score	Status
Q1	Meeting Scheduler › Coding	2 hour 6 min 53 sec	25/ 50	
Q2	Duplicated Products › Coding	16 min 38 sec	50/ 50	
Q3	Product Defects › Coding	22 min 47 sec	75/ 75	
Q4	Programmer String › Coding	1 hour 47 min 30 sec	75/ 75	
Q5	Coloring Houses › Coding	1 hour 1 min 29 sec	1/ 100	

QUESTION 1

Correct Answer

Score 25

Meeting Scheduler › Coding

Easy | Real-World | Prefix Sum | Interviewer Guidelines

QUESTION DESCRIPTION

On a given day, there are *n* meetings scheduled. There is a list of the meetings given as a 2D array, *meetingTimings,* of size *n x m*, that contains the start and end times of the meetings. Determine the minimum number of meeting rooms needed to conduct all the meetings so that no meetings overlap in a meeting room.

Note: Meetings can end and begin at the same time in one room. For example, meetings at times [10, 15], and [15, 20] can be held in the same room.

Example

Suppose *n* = 5, *meetingTimings* = [[1, 4], [1, 5], [5, 6], [6, 10], [7, 9]].

Time	Event	Meetings Running
1	meetings 1 and 2 start	1, 2
2		1, 2
3		1, 2
4	meeting 1 ends	2
5	meeting 2 ends, meeting 3 starts	3
6	meeting 3 ends, meeting 4 starts	4
7	meeting 5 starts	4, 5
8		4, 5
9	meeting 5 ends	4
10	meeting 6 ends	

At least two meeting rooms are required. Meetings 1 and 2 overlap, as do 4 and 5. Return 2 as the answer

Function Description

Complete the function *getMinRooms* in the editor below.

getMinRooms has the following parameter:

int meetingTimings[n][2]: the start and end times of the meetings

Returns

int: the minimum number of meeting rooms required

Constraints

- $1 \leq n \leq 2 * 10^5$
- $1 \leq meetingTimings[i][0] \leq meetingTimings[i][1] \leq 2 * 10^6$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of meetings to be scheduled.

The next line contains 2.

Each of the next *n* lines contains two space-separated integers, *meetingTimings[i][0],* and *meetingTimings[i][1],* the start and end times of the meetings.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
6       →   meetingTimings[] size, n = 6
2       →   constant integer, 2
2 8     →   meetingTimings = [[2, 8],
3 9                           [3, 9],
5 11                          [5, 11],
3 4                           [3, 4],
11 15                         [11, 15],
8 20                          [8, 20]]
```

Sample Output

```
3
```

Explanation

An optimal way to assign rooms for meetings is:

Room 1 - [2, 8], [8, 20]

Room 2 - [3, 4], [5, 11], [11, 15]

Room 3 - [3, 9]

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN        FUNCTION
-----        --------
5       →    meetingTimings[] size, n = 6
2       →    constant integer, 2
2 8     →    meetingTimings = [[2, 8],
3 7                           [3, 7],
4 6                           [4, 6],
1 9                           [1, 9],
2 5                           [2, 5]]
```

Sample Output

```
5
```

Explanation

An optimal way to assign rooms for meetings is:
Room 1 - [2, 8]
Room 2 - [3, 7]
Room 3 - [4, 6]
Room 4 - [1, 9]
Room 5 - [2, 5]

INTERVIEWER GUIDELINES

▼ Solution

Skills: Sorting, Problem Solving

To find the maximum number of overlapping meetings at any point in time, iterate through the meetings and add the start and end times to a common array after pairing them with 1 and -1 respectively, then sort them. This helps make sure that any meeting *x* that starts before *y* occurs before *y* with 1 and similarly with -1 and end times.

Iterate the sorted list, adding and subtracting 1 when we encounter a meeting's start and end times respectively. The maximum value of this count is the answer.

Optimal Solution:

```
def getMinRooms(meetingTimings):
    n = len(meetingTimings)
    ans = count = 0
    times = []
    for st, en in meetingTimings:
        times.append([st, 1])
        times.append([en, -1])

    times.sort()

    for _, val in times:
        count += val
        ans = max(ans, count)

    return ans
```

▼ Complexity Analysis

Time Complexity - O(n logn)
Sorting takes O(n logn) time in the worst case.

Space Complexity - O(n)
The *times* array takes O(n) extra space in the worst case.

CANDIDATE ANSWER

Language used: **Swift**

```
1 /*
2  * Complete the 'getMinRooms' function below.
3  *
4  * The function is expected to return an INTEGER.
5  * The function accepts 2D_INTEGER_ARRAY meetingTimings as parameter.
6  */
7 struct MaxHeap<Element: Comparable> {
8     var elements: [Element] = []
```

```
    var isEmpty: Bool {
        return elements.isEmpty
    }

    var count: Int {
        return elements.count
    }

    mutating func insert(_ element: Element) {
        elements.append(element)
        siftUp(from: elements.count - 1)
    }

    mutating func siftUp(from index: Int) {
        var child = index
        var parent = parentIndex(of: child)

        while child > 0 && elements[child] > elements[parent] {
            elements.swapAt(child, parent)
            child = parent
            parent = parentIndex(of: child)
        }
    }

    mutating func pop() -> Element? {
        guard !isEmpty else {
            return nil
        }

        elements.swapAt(0, count - 1)
        let element = elements.removeLast()
        siftDown(from: 0)
        return element
    }

    mutating func siftDown(from index: Int) {
        var parent = index

        while true {
            let leftChild = leftChildIndex(of: parent)
            let rightChild = rightChildIndex(of: parent)
            var candidate = parent

            if leftChild < count && elements[leftChild] > elements[candidate]
{
                candidate = leftChild
            }

            if rightChild < count && elements[rightChild] >
elements[candidate] {
                candidate = rightChild
            }

            if candidate == parent {
                return
            }

            elements.swapAt(parent, candidate)
            parent = candidate
        }
    }

    func parentIndex(of index: Int) -> Int {
```

```
        return (index - 1) / 2
    }

    func leftChildIndex(of index: Int) -> Int {
        return index * 2 + 1
    }

    func rightChildIndex(of index: Int) -> Int {
        return index * 2 + 2
    }
}

func getMinRooms(meetingTimings: [[Int]]) -> Int {
    let meetings = meetingTimings.sorted(by: { $0[0] < $1[0] })
    var maxHeaps: [MaxHeap<Int>] = []

    for meeting in meetings {
        var allocatedRoom = false

        for i in 0..<maxHeaps.count {
            if let endTime = maxHeaps[i].elements.first, endTime <=
meeting[0] {
                // 이전 회의가 끝난 시간보다 현재 회의의 시작 시간이 같거나 늦은 경우
                // 기존 회의실을 이용
                maxHeaps[i].insert(meeting[1])
                allocatedRoom = true
                break
            }
        }

        if !allocatedRoom {
            // 새로운 회의실 필요
            var newHeap = MaxHeap<Int>()
            newHeap.insert(meeting[1])
            maxHeaps.append(newHeap)
        }
    }

    return maxHeaps.count
}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Success	1	0.0256 sec	17.2 KB
TestCase 1	Easy	Sample case	Success	1	0.0248 sec	16.8 KB
TestCase 2	Easy	Sample case	Success	1	0.0189 sec	17.3 KB
TestCase 3	Medium	Hidden case	Success	3	0.0188 sec	17.6 KB
TestCase 4	Medium	Hidden case	Success	3	0.0358 sec	16.9 KB
TestCase 5	Medium	Hidden case	Success	3	0.0294 sec	17.1 KB
TestCase 6	Medium	Hidden case	Success	3	0.0277 sec	17.1 KB
TestCase 7	Medium	Hidden case	Success	3	0.0726 sec	17.5 KB
TestCase 8	Medium	Hidden case	Success	3	0.2075 sec	17.8 KB
TestCase 9	Medium	Hidden case	Success	4	0.875 sec	18.5 KB

TestCase 10	Hard	Hidden case	Terminated due to timeout	0	2.0031 sec	23.1 KB
TestCase 11	Hard	Hidden case	Terminated due to timeout	0	2.003 sec	26.5 KB
TestCase 12	Hard	Hidden case	Terminated due to timeout	0	2.0029 sec	29 KB
TestCase 13	Hard	Hidden case	Terminated due to timeout	0	2.0047 sec	31.7 KB
TestCase 14	Hard	Hidden case	Terminated due to timeout	0	2.0035 sec	32.8 KB

No Comments

QUESTION 2

Correct Answer

Score 50

Duplicated Products › Coding

Easy | Implementation | Problem Solving | Algorithms | Sets | Interviewer Guidelines

QUESTION DESCRIPTION

You are given a complex list of *n* products, each with a *name, price,* and *weight.* Find out how many duplicate products are present within the list. Duplicate products contain identical parameters for all fields in the list (i.e. *name*, *price,* and *weight).*

Example:

There are *n = 5* products with attributes listed in three arrays, aligned by index.

name = [ball, bat, glove, glove, glove]
price = [2, 3, 1, 2, 1]
weight = [2, 5, 1, 1, 1]

A complete item description for item 0: *(name[0], prices[0], weight[0])* is *(ball, 2, 2)*

```
Name     Price Weight
ball     2     2
bat      3     5
glove    1     1
glove    2     1
glove    1     1
```

The first two items are unique.
The two gloves at indices *2* and *4* are equal in all three attributes so there is *1* duplicate.
The last glove at index 3 has a different price from the other two, so it is not a duplicate.

There is *1* duplicate item in the original list.

Function Description

Complete the function *numDuplicates* in the editor below. The function must return an integer denoting the number of duplicates within the product list.

numDuplicates has the following parameter(s):

string name[n]: string array of size *n,* where *names[i]* denotes the name of the product at the index of *i.*

int price[n]: int array of size *n,* where *prices[i]* denotes the price of the product at the index of *i.*

int weight[n]: int array of size *n,* where *weights[i]* denotes the weight of the product at the index of *i.*

Constraints

- $1 \le n \le 10^5$
- *name[i]* is non-empty, has at most *10* characters, and all its characters are lowercase English letters, ascii[a-z]
- *1 ≤ price[i], weight[i] ≤ 1000*

▼ Input Format Format for Custom Testing

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

In the first line there is a single integer *n* which is the length of *name*.
Each of the following *n* lines contains a string for *name[i]*
In the next line, *n* is repeated denoting the length of *price*.
Each of the following *n* lines contains an integer for *price[i].*
In the next line, *n* is repeated denoting the length of *weight*.
Each of the following *n* lines contains an integer for *weight[i].*

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
5     →     name[] size n = 5
ball  →     name = ['ball', 'box', 'ball', 'ball', 'box']
box
ball
ball
box
5     →     price[] size n = 5
2     →     price = [2, 2, 2, 2, 2]
2
2
2
2
5     →     weight[] size n = 5
1     →     weight = [1, 2, 1, 1, 3]
2
1
1
3
```

Sample Output

```
2
```

Explanation

```
Name Price Weight
ball 2     1
box  2     2
ball 2     1
ball 2     1
box  2     3
```

The *3* balls have the same name, price and weight, so there are *2* duplicates. The two other products are boxes but they have different weights.

▼ Sample Case 1

Sample Input

```
STDIN     Function
-----     --------
5    →    name[] size n = 5
ball →    name = ['ball', 'box', 'lamp', 'brick', 'pump']
box
lamp
brick
pump
5    →    price[] size n = 5
2    →    price = [2, 2, 2, 2, 2]
2
2
2
2
5    →    weight[] size n = 5
2    →    weight = [2, 2, 2, 2, 2]
2
2
2
2
```

Sample Output

```
0
```

Explanation

```
Name  Price Weight
ball  2     2
box   2     2
lamp  2     2
brick 2     2
pump  2     2
```

Each product is unique so there are no duplicates.

INTERVIEWER GUIDELINES

▼ Hint 1

Hash each item depending on the entities (Name, Price, Weight), such that each item can be uniquely denoted.

▼ Hint 2

One such way is to append them, space-separated, with each other, and form a string. Add each of these strings to a set and find the number of duplicate items = N - size_of_set.

▼ Solution

Concepts covered: This problem tests the candidates on arrays and strings.

Optimal Solution:

Hash each item depending on the entities (Name, Price, Weight), such that each item can be uniquely denoted. One such way is to append them, space-separated, with each other and form a string. Add each of these strings to a set and find the number of duplicate items = N - size_of_set.

```
def numDuplicates(name, price, weight):
    # Write your code here
    s = set()
    for i in range(len(name)):
        s.add(name[i] + ' ' + str(price[i]) + ' ' +str(weight[i]))
    return len(name) - len(s)
```

Error Handling:

1. It is important to join the entities using space and not without space.

▼ Complexity Analysis

Time Complexity - O(N).
We iterate through all the given set of items only once.

Space Complexity - O(N)
The set requires O(N) space in the worst case.

CANDIDATE ANSWER

Language used: **Swift**

```
/*
 * Complete the 'numDuplicates' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. STRING_ARRAY name
 *  2. INTEGER_ARRAY price
 *  3. INTEGER_ARRAY weight
 */

 struct Item: Hashable{
     let name: String
     let price: Int
     let weight: Int
 }

func numDuplicates(name: [String], price: [Int], weight: [Int]) -> Int {
    var duplicates = [(name: String, price: Int, weight: Int)]()
    var seen = Set<String>()
    var seenItems = Set<Item>()

    for i in 0..<name.count{
        let item = Item(name: name[i], price: price[i], weight: weight[i])
        if seen.contains(item.name) && seenItems.contains(item){
            duplicates.append((item.name, item.price, item.weight))
        }else{
            seen.insert(item.name)
            seenItems.insert(item)
        }
    }
    return duplicates.count

}

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Success	1	0.0216 sec	15.2 KB
TestCase 1	Easy	Sample case	Success	1	0.0248 sec	15.6 KB
TestCase 2	Easy	Sample case	Success	1	0.0234 sec	15.6 KB
TestCase 3	Easy	Sample case	Success	4	0.0372 sec	16.2 KB

TestCase 4	Easy	Sample case	Success	4	0.0377 sec	15.8 KB
TestCase 5	Easy	Hidden case	Success	4	0.0214 sec	16.2 KB
TestCase 6	Easy	Hidden case	Success	4	0.0206 sec	15.9 KB
TestCase 7	Medium	Hidden case	Success	5	0.611 sec	33.2 KB
TestCase 8	Hard	Hidden case	Success	5	0.6213 sec	32.5 KB
TestCase 9	Hard	Hidden case	Success	5	0.611 sec	33.5 KB
TestCase 10	Hard	Hidden case	Success	5	0.5385 sec	27.3 KB
TestCase 11	Hard	Hidden case	Success	5	0.5561 sec	35.4 KB
TestCase 12	Medium	Hidden case	Success	6	0.4969 sec	26.9 KB

No Comments

QUESTION 3

Needs Review

Score 75

Product Defects › Coding

Algorithms | Data Structures | Implementation | Medium | Problem Solving

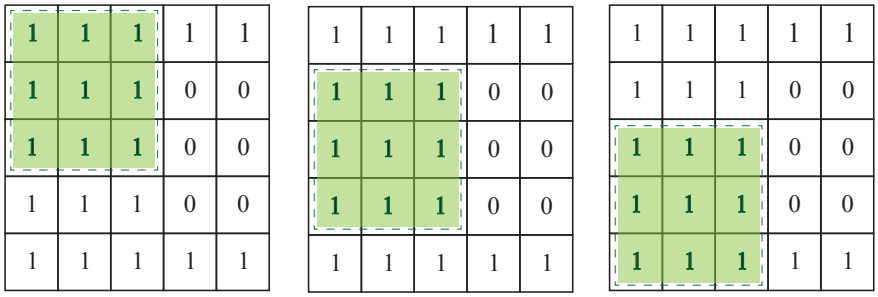
QUESTION DESCRIPTION

A quality agent is responsible for inspecting samples of the finished products in the production line. Each sample contains defective and non-defective products represented by 1 and 0 respectively. The product samples are placed sequentially in a two-dimensional square matrix. The goal is to determine the size of the largest square of defective products in the two-dimensional square matrix.

Example

n x n = 5 x 5 matrix of product samples

samples = [[1,1,1,1,1], [1,1,1,0,0], [1,1,1,0,0], [1,1,1,0,0], [1,1,1,1,1]]



- The first square of defective products is a sub-matrix *3 x 3* starting at (0,0) and ending at (3,3)
- The second square of defective products is also a sub-matrix *3 x 3* starting at (1,0) and ending at (4,3)
- The third square of defective products is also a sub-matrix *3 x 3* starting at (2,0) and ending at (5,3)
- The size of the largest square of defective products is 3

Function Description

Complete the function *findLargestSquareSize* in the editor below.

findLargestSquareSize has the following parameter:

int samples[n][n]: a two-dimensional array of integers

Returns:

int: an integer that represents the size of the largest square sub-matrix of defective products.

Constraints

- *0 ≤ n ≤ 500*
- *samples[i][j]* is in the set *{0, 1}* (*0* denotes anon-defective products and *1* denotes a defective product)

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of rows (the number of samples).
The second line contains the integer, *n*, the number of columns (the number of products in a sample).

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains *n* space-separated integers that describe *samples[i]*.

▼ Sample Case 0

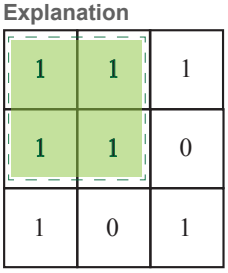
Sample Input For Custom Testing

```
STDIN       Function
-----       --------
3       →   samples[] size n = 3
3       →   samples[i][] size n = 3
1 1 1   →   samples=[[1,1,1],[1,1,0],[1,0,1]]
1 1 0
1 0 1
```

Sample Output

```
2
```

Explanation



- The first square of defective products is a sub-matrix *2 x 2* starting at (0,0) and ending at (1,1)
- The other square of defective products are a sub-matrix *1 x 1* at (2,0) , (0,2) and (2,2)
- The size of the largest square of defective products is 2

▼ Sample Case 1

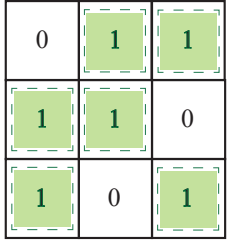
Sample Input For Custom Testing

```
STDIN     Function
-----     --------
3     →   samples[] size n = 3
3     →   samples[i][] size n = 3
0 1 1 →   samples=[[0,1,1],[1,1,0],[1,0,1]]
1 1 0
1 0 1
```

Sample Output

```
1
```

Explanation



- All square of defective products are a sub-matrix *1 x 1* at (0,1) , (0,2) , (1,0), (1,1), (2,0) and (2,2).
- The size of the largest square of defective products is 1

CANDIDATE ANSWER

Language used: **Swift**

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

func findLargestSquareSize(samples: [[Int]]) -> Int {
    // Write your code here
    var matrix = samples

    let rows = matrix.count
    let cols = matrix[0].count
    var dp = [[Int]](repeating: [Int](repeating: 0, count: cols), count:
rows)
    var maxLength = 0

    for i in 0..<rows{
        for j in 0..<cols{
            if matrix[i][j] == 1{
                if i==0 || j==0{
                    dp[i][j] = 1
                }else{
                    dp[i][j] = min(dp[i-1][j-1], dp[i][j-1], dp[i-1][j]) + 1
                }

                maxLength = max(maxLength, dp[i][j])
            }
        }
    }
    return maxLength


}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Success	1	0.0366 sec	17.5 KB
TestCase 1	Easy	Sample case	Success	1	0.0501 sec	17.5 KB
TestCase 2	Easy	Sample case	Success	1	0.0506 sec	17.3 KB
TestCase 3	Easy	Hidden case	Success	3	0.0314 sec	17.6 KB
TestCase 4	Easy	Hidden case	Success	3	0.0383 sec	17.2 KB

TestCase 5	Easy	Hidden case	Success	3	0.0284 sec	17.5 KB
TestCase 6	Medium	Sample case	Success	6	0.0992 sec	17.7 KB
TestCase 7	Medium	Hidden case	Success	6	0.0803 sec	17.8 KB
TestCase 8	Medium	Hidden case	Success	6	0.0452 sec	17.3 KB
TestCase 9	Hard	Sample case	Success	10	0.0585 sec	17.9 KB
TestCase 10	Hard	Hidden case	Success	10	0.0788 sec	17.8 KB
TestCase 11	Hard	Hidden case	Success	10	0.0704 sec	17.5 KB
TestCase 12	Hard	Hidden case	Success	15	0.2016 sec	21.2 KB

No Comments

QUESTION 4

Correct Answer

Score 75

Programmer String › Coding

Strings | Medium | Algorithms | Problem Solving | Hash Map | Interviewer Guidelines

QUESTION DESCRIPTION

A *programmer string* contains letters that can be rearranged to form the word '*programmer*' and is a substring of a longer string. Note that the strings '*programmer*', '*grammproer*', and '*prozmerqgram*' are all classified as programmer strings by this definition. Given a string, determine the number of indices that lie between the rightmost and leftmost programmer strings that it contains.

Example

s = 'programmerxxxprozmerqgram'

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
p	r	o	g	r	a	m	m	e	r	x	x	x	p	r	o	z	m	e	r	q	g	r	a	m

In this example, indices *0 - 9* form one programmer string and indices *13 - 24* contain another. There are *3* indices between the programmer, so the function will return *3*.

Function Description

Complete the function *programmerStrings* in the editor below.

programmerStrings has the following parameter(s):

string s: a string containing *2* programmer strings

Returns:

int: the number of indices which are between the rightmost and leftmost programmer strings within *s*

Constraints

- String *s* consists of lowercase English alphabetic letters only, *ascii[a-z]*.
- $1 \leq$ the length of $s \leq 10^5$.
- There will always be two non-overlapping programmer strings.

▼ Input Format for Custom Testing

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

The only line contains a string *s*.

▼ Sample Case 0

Sample Input

```
STDIN                          Function
-----                          ------
progxrammerrxproxgrammer   →   s = 'progxrammerrxproxgrammer'
```

Sample Output

```
2
```

Explanation

There are two indices, *i = 11* and *i = 12* between the leftmost and rightmost programmer strings.

▼ Sample Case 1

Sample Input

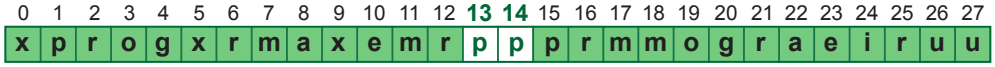
```
STDIN                                  Function
-----                                  -----
xprogxrmaxemrppprmmograeiruu   →   s = 'xprogxrmaxemrppprmmograeiruu'
```

Sample Output

```
2
```

Explanation

There are two indices, *i = 13* and *i = 14*, between the leftmost and rightmost programmer strings.



▼ Sample Case 2

Sample Input

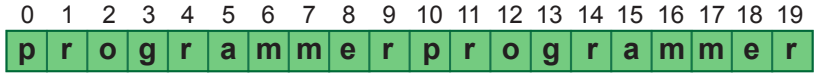
```
STDIN                        Function
-----                        -----
programmerprogrammer   →   s = 'programmerprogrammer'
```

Sample Output

```
0
```

Explanation

There are no indices between the leftmost and rightmost programmer strings.



INTERVIEWER GUIDELINES

▼ Hint 1

First try to solve the subproblem: given a string t can you rearrange its letters such that it would have the string "programmer" as a substring. Think with respect to the frequency of the characters.

▼ Hint 2

To find the leftmost programmer string, you find the smallest prefix which is a programmer string. Similarly for the rightmost programmer string, you find the smallest suffix which is a programmer string.

▼ Solution

Concepts covered: String, Hash Map

Optimal Solution:

Let's first try to solve the subproblem: given a string t can you rearrange its characters such that you can have "programmer" as its substring? It is easy to see that if the frequency of characters in string "programmer" is a subset(less than or equal to) the frequency of characters of string t, it can be done. For example, let's take t = "grammproezr" and calculate the frequency of each character: {'g': 1, 'r': 3, 'a': 1, 'm': 2, 'p': 1, 'o': 1, 'e': 1, 'z': 1}. And also calculate the frequency of each character in string "programmer", it is: {'g': 1, 'r': 2, 'a': 1, 'm': 2, 'p': 1, 'o': 1, 'e': 1}. You will notice that the frequency of characters in "programmer" is a subset of the frequency of characters in "grammproezr".

To find the leftmost programmer string, we iterate over each prefix and find the smallest prefix which is the programmer string. To do so, we maintain a frequency table of each character in the current prefix and update it as we move to the next prefix. For the current prefix, to check if it is programmer string or not we apply the algorithm described above. It is important to note that the above algorithm will just take 7 passes to find whether it is a programmer string or not. This gives us the ending point of the smallest prefix (say p) which is a programmer string. Do the same thing for each suffix to find out the starting point of the last programmer string (say s). The final answer is s - p - 1.

```
def isSubset(dict1, dict2):
    valid = True
    for c in dict1:
        if c not in dict2:
            valid = False
        elif dict2[c] < dict1[c]:
            valid = False
    return valid

def programmerStrings(s):
    t = 'programmer'
    req_freq = {}
    for i in range(len(t)):
        req_freq[t[i]] = req_freq.get(t[i], 0) + 1
    first_end = 0
    last_begin = len(s) - 1
    cur = {}
    for i in range(0,len(s)):
        cur[s[i]] = cur.get(s[i], 0) + 1
        if isSubset(req_freq, cur) == True:
            first_end = i
            break
    cur = {}
    for i in range(len(s)-1,-1,-1):
        cur[s[i]] = cur.get(s[i], 0) + 1
        if isSubset(req_freq, cur) == True:
            last_begin = i
            break
    return last_begin - first_end - 1
```

Brute Force Approach: This solution takes too long to execute in all cases.

```
def permute(a, l, r):
    if l == r:
        return ''.join(a)
    else:
        permutations = []
        for i in range(l,r+1):
            a[l], a[i] = a[i], a[l]
            p = permute(a, l+1, r)
            for j in p:
                permutations.append(j)
            a[l], a[i] = a[i], a[l]
        return permutations

def programmerStrings(s):
    t = 'programmer'
    first_end = 0
    last_begin = len(s) - 1
    for i in range(0,len(s)):
        prefix = s[0:i+1]
        permutations = permute(list(prefix), 0, i)
        for p in permutations:
            if t in p:
                first_end = i
                break
        if first_end != 0:
            break
    for i in range(len(s)-1,-1,-1):
        suffix = s[i : len(s)]
        permutations = permute(list(suffix), 0, i)
        for p in permutations:
            if t in p:
                last_begin = i
                break
        if last_begin != 0:
            break
    return last_begin - first_end - 1
```

The above brute force approach is quite naive and will run in O(n*n!) time complexity.

Error Handling: There are no edge cases to handle.

▼ Complexity Analysis

Time Complexity - O(n).
Since we are iterating over every prefix and suffix to find out if it is a programmer string, we take O(n) passes. For each pass we can find whether it is a valid programmer string in constant time since the number of distinct characters in "programmer" is just 7.

Space Complexity - O(1) - Constant extra space is required.
We are maintaining a hash table to store the frequencies of each character in the string, but since the number of letters are bounded by 26, the extra space taken to build the hash table is constant.

▼ Follow up Question

Find the total number of substrings in the given string which are programmer strings.
Let's iterate over each index i from 0 to n-1 and suppose that the string which are currently in consideration start from index i. We can observe that if the substring from index i to j is a programmer string, then substring i to k would also be a programmer string when k > j. Now, we just need to find the smallest index j which gives us a programmer string. This can be determined using binary search or two pointers.

(HARD) We can change any character in the original string to any other character. What is the minimum number of characters we must change so that every substring of length greater than

or equal to 10(the length of the string "programmer") is a programmer string?

Let's look at the first substring of length 10 (from index 0 to 9), since it must be a programmer string, it is a permutation of "programmer". Let's say that the permutation is as follows: p_0, p_1, p_2,... p_9. Now, let's take the next substring of length 10 (from index 1 to 10). Since it must also be a programmer string, it should end with p_0. Hence, we can prove that the complete string must be a periodic concatenation of one permutation of "programmer". Hence we need to choose a permutation which leads to the minimum number of changes. We can observe that there are a total of 302400 permutations possible. We can try all of them and find the one that minimizes cost. Overall time complexity: O(26*n + 10*(10!/12))

CANDIDATE ANSWER

Language used: **Swift**

```
/*
 * Complete the 'programmerStrings' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts STRING s as parameter.
 */


func programmerStrings(s: String) -> Int {
    var targetString = Array("programmer")
    var inputCharacters = Array(s)
    var startIndex = 0
    var endIndex = 0

    for i in 0..<inputCharacters.count{
        if targetString.isEmpty{
            targetString = Array("programmer")
            startIndex = i
            break
        }

        let currentCharacter = inputCharacters[i]

        for j in 0..<targetString.count{
            if targetString[j] == currentCharacter{
                targetString.remove(at: j)
                break
            }
        }
    }

    for i in stride(from: inputCharacters.count-1, through: 0, by: -1){
        if targetString.isEmpty{
            endIndex = i
            break
        }
        let currentCharacter = inputCharacters[i]

        for j in 0..<targetString.count{
            if targetString[j] == currentCharacter{
                targetString.remove(at: j)
                break
            }
        }
    }
```

```
    return endIndex - startIndex + 1

}

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	Success	1	0.0174 sec	16 KB
TestCase 1	Easy	Sample case	Success	1	0.0364 sec	16 KB
TestCase 2	Easy	Sample case	Success	1	0.0179 sec	15.7 KB
TestCase 3	Easy	Hidden case	Success	8	0.0176 sec	15.6 KB
TestCase 4	Easy	Hidden case	Success	8	0.0164 sec	16 KB
TestCase 5	Easy	Hidden case	Success	8	0.03 sec	15.6 KB
TestCase 6	Easy	Hidden case	Success	8	0.0202 sec	16 KB
TestCase 7	Easy	Sample case	Success	8	0.0181 sec	15.7 KB
TestCase 8	Easy	Sample case	Success	6	0.0361 sec	16.9 KB
TestCase 9	Easy	Hidden case	Success	6	0.0198 sec	16.5 KB
TestCase 10	Easy	Hidden case	Success	6	0.0304 sec	17.1 KB
TestCase 11	Easy	Hidden case	Success	7	0.019 sec	17.2 KB
TestCase 12	Easy	Hidden case	Success	7	0.0562 sec	17.3 KB

No Comments

QUESTION 5

Correct Answer

Score 1

Coloring Houses › Coding

Dynamic Programming | Algorithms | Problem Solving | Hard | Arrays

QUESTION DESCRIPTION

The city of Hackerland can be represented with an even number *n* houses arranged in a row. A painter must paint the houses using at most three colors. The following conditions must hold true:

1. No two adjacent houses are the same color.
2. The houses which are at the same distance from both ends must not be colored with the same color. For example, n=6 then houses will be [1,2,3,4,5,6], so the houses at the same distance from both the ends will be [1,6], [2,5], [3,4].

The task is to find the number of ways to paint the houses using at most three colors such that both the above conditions hold true. Since the answer can be large, report it modulo $10^9 + 7$. Two ways are considered different if at least one house is colored differently.

Example

For n = 4, some of the possible valid arrangements are:

- (color1, color2, color3, color2)
- (color1, color3, color1, color3)

The number of ways to paint 4 houses using three colors is 18. Return 18 modulo ($10^9 + 7$) which is 18.

Function Description

Complete the *countWaysToColorHouses* function in the editor below.

countWaysToColorHouses takes in a single parameter:

int n: the number of houses

Return

int : the number of ways in which the houses can be colored, calculated as a modulo of (10^9+7)

Constraints

- $2 \leq n \leq 100000$
- n is an even integer.

▼ Input Format For Custom Testing

The only line of input contains an integer, n.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
2        →     n = 2
```

Sample Output

```
6
```

Explanation

The valid arrangements for 2 houses are:

- (color1, color2)
- (color1, color3)
- (color2, color1)
- (color3, color1)
- (color2, color3)
- (color3, color2)

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
4        →     n = 4
```

Sample Output

```
18
```

Explanation

Total valid arrangements for 4 houses are 18.

Some of the valid arrangements are:

- (color1, color2, color1, color2)
- (color1, color3, color1, color3)
- (color2, color1, color2, color1)
- (color3, color1, color3, color1)
- (color2, color3, color2, color3)
- (color3, color2, color3, color2)

INTERVIEWER GUIDELINES

▼ Solution

Skills: Dynamic Programming

We can solve this problem using dynamic programming. Let dp[i][c1][c2] represent the number of ways to paint the first *i* houses and the last *i* houses such that the i^{th} house is painted with the color c1 and the $(n - i)^{th}$ house is painted with the color c2.

Now let the $(i + 1)^{th}$ house and the $(n - (i + 1))^{th}$ house be painted with the colors c3 and c4. We can establish the following conditions:

c1 ≠ c3 and c4 ≠ c2 as adjacent colors can not be the same

c1 ≠ c2 and c3 ≠ c4 as the colors at the same distance from both ends can not be the same

Based on the above conditions, we can establish the following transitions for our dp:

dp[i][c1][c2] = sum(dp[i - 1][c3][c4]) where c1, c2, c3, c4 = 1, 2, 3 and c1 ≠ c3 and c4 ≠ c2 and c1 ≠ c2 and c3 ≠ c4

Finally the total number of ways would be the sum of dp[n / 2][c1][c2] where c1, c2 = 1, 2, 3.

Optimal Solution:

```
def countWaysToColorHouses(n):
    dp = [[[0] * 4 for _ in range(4)] for _ in range(n)]
    for c1 in range(1, 4):
        for c2 in range(1, 4):
            if c1 != c2:
                dp[1][c1][c2] = 1

    mod = 10 ** 9 + 7
    for i in range(2, n // 2 + 1):
        for c1 in range(1, 4):
            for c2 in range(1, 4):
                for c3 in range(1, 4):
                    for c4 in range(1, 4):
                        if c1 != c4 and c2 != c3 and c1 != c2 and c3 != 
c4:
                            dp[i][c3][c4] += dp[i - 1][c1][c2]
                            dp[i][c3][c4] %= mod

    ans = 0
    for c1 in range(1, 4):
        for c2 in range(1, 4):
            ans = (ans + dp[n // 2][c1][c2]) % mod
    return ans
```

▼ Complexity Analysis

Time Complexity - O(n x 3^4)

We iterate over each index and for each index, we iterate over all 4 possible combinations of colors. Hence the overall time complexity is O(n x 3^4).

Space Complexity - O(n x 3^2)

The dp array takes O(n x 3^2) extra space. Note that since dp[i] is dependent only on dp[i - 1], we can optimize the space complexity to O(3^2) ~ O(1) by storing only the last dp state.

CANDIDATE ANSWER

Language used: **Swift**

```
1
2 /*
3  * Complete the 'countWaysToColorHouses' function below.
4  *
5  * The function is expected to return an INTEGER.
```

```swift
 * The function accepts INTEGER n as parameter.
 */

func countWaysToColorHouses(n: Int) -> Int {
  if n <= 0 {
        return 0
    }


    if n == 1 {
        return 3
    }

    if n == 2 {
        return 6
    }

    var dp = [[Int]](repeating: [Int](repeating: 0, count: 3), count: n + 1)


    dp[1][0] = 3
    dp[2][0] = 6

    for i in 3...n {
        dp[i][0] = dp[i - 1][0] + dp[i - 1][1] + dp[i - 1][2]
        dp[i][1] = dp[i - 1][0] + dp[i - 1][2]
        dp[i][2] = dp[i - 1][0] + dp[i - 1][1]
    }

    let totalWays = dp[n][0] + dp[n][1] + dp[n][2]

    return totalWays

}

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample case	Success	1	0.0162 sec	15.9 KB
Testcase 1	Easy	Sample case	Wrong Answer	0	0.035 sec	16 KB
Testcase 2	Easy	Sample case	Wrong Answer	0	0.0218 sec	16 KB
Testcase 3	Easy	Hidden case	Wrong Answer	0	0.0205 sec	16 KB
Testcase 4	Easy	Hidden case	Wrong Answer	0	0.0213 sec	16 KB
Testcase 5	Easy	Hidden case	Wrong Answer	0	0.0214 sec	16 KB
Testcase 6	Easy	Hidden case	Wrong Answer	0	0.0401 sec	15.7 KB
Testcase 7	Easy	Hidden case	Runtime Error	0	0.0523 sec	15.4 KB
Testcase 8	Medium	Hidden case	Runtime Error	0	0.034 sec	15.8 KB
Testcase 9	Medium	Hidden case	Runtime Error	0	0.0468 sec	15.4 KB
Testcase 10	Medium	Hidden case	Runtime Error	0	0.0355 sec	15.7 KB
Testcase 11	Medium	Hidden case	Runtime Error	0	0.0333 sec	15.9 KB
Testcase 12	Hard	Hidden case	Runtime Error	0	0.03 sec	15.3 KB
Testcase 13	Hard	Hidden case	Runtime Error	0	0.0504 sec	16 KB
Testcase 14	Hard	Hidden case	Runtime Error	0	0.0507 sec	16.5 KB

No Comments

PDF generated at: 14 Jul 2023 12:23:01 UTC