You can view this report online at : https://www.hackerrank.com/x/tests/864906/candidates/16704370/report

Full Name:	**********
Email:	**********
Test Name:	**talegri - developer auction test**
Taken On:	17 Aug 2020 21:34:56 CEST
Time Taken:	60 min/ 60 min
Invited by:	Urs
Invited on:	12 Aug 2020 13:37:28 CEST
Skills Score:	Problem Solving (Basic) 0/50 Problem Solving (Intermediate) 37/75 SQL (Basic) 50/50
Tags Score:	Algorithms 37/75 Arrays 37/125 Binary Search 37/75 Data Structures 37/75 Easy 50/100 Interviewer Guidelines 50/100 Joins 50/50 Loops 0/50 Medium 37/75 Problem Solving 37/125 SQL 50/50 Theme: Finance 37/75

49.7%

87/175

scored in **talegri - developer auction test** in 60 min on 17 Aug 2020 21:34:56 CEST

Recruiter/Team Comments:

No Comments.

	Question Description	Time Taken	Score	Status
Q1	Growth in 2 Dimensions > Coding	36 min 42 sec	0/ 50	✗
Q2	Youngest Employees > DbRank	6 min 18 sec	50/ 50	✓
Q3	Profit Targets > Coding	17 min 53 sec	37/ 75	✓

QUESTION 1

✗ Wrong Answer

Growth in 2 Dimensions > Coding

Easy | Loops | Problem Solving | Arrays | Interviewer Guidelines

QUESTION DESCRIPTION

/

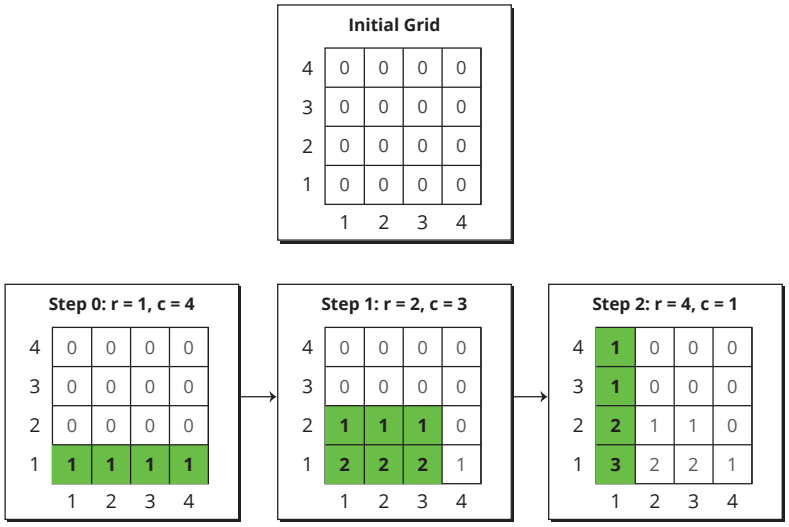
Score 0

Start with an infinite two dimensional grid filled with zeros, indexed from *(1,1)* at the bottom left corner with coordinates increasing toward the top and right. Given a series of coordinates *(r, c)*, where *r* is the ending row and *c* is the ending column, add *1* to each element in the range from *(1,1)* to *(r, c)* inclusive. Once all coordinates are processed, determine how many cells contain the maximal value in the grid.

Example

upRight = ["1 4", "2 3", "4 1"]

The two space-separated integers within each string represent *r* and *c* respectively. The following diagrams show each iteration starting at zero. The maximal value in the grid is *3,* and there is *1* occurrence at cell *(1, 1)*.



Function Description

Complete the function *countMax* in the editor below.

countMax has the following parameter(s):

string upRight[n]: an array of strings made of two space-separated integers, *r* and *c*.

Return

long: the number of occurrences of the final grid's maximal element

Constraints

- *1 ≤ n ≤ 100*
- *1 ≤ number of rows, number of columns ≤ 10^6*

▼ 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 *upRight*.
Each of the next *n* lines contains a string of two space-separated integers representing coordinates *r* and *c* for element *upRight[i]*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
3     →     upRight[] size n = 3
2 3   →     upRight = ['2 3', '3 7', '4 1']
```

/

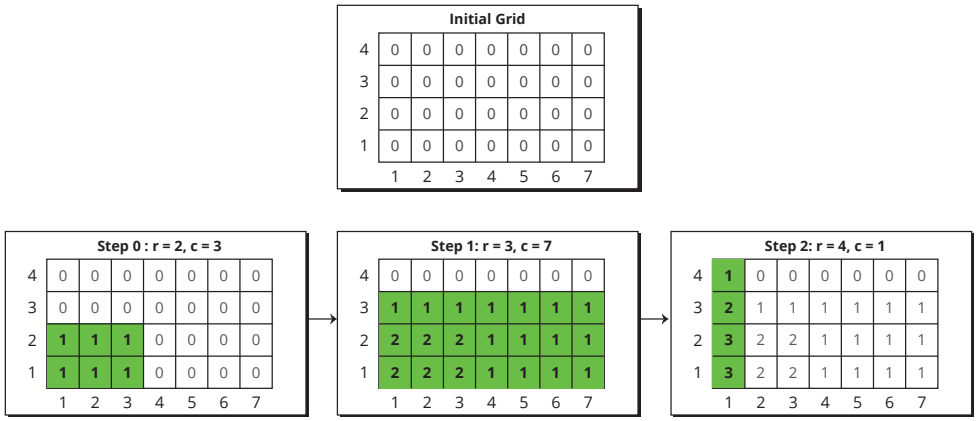
```
3 7
4 1
```

Sample Output

```
2
```

Explanation

Given *upRight = ["2 3", "3 7", "4 1"]*:



The portion of the infinite grid corresponding to cells *(r, c)* where *1 ≤ r ≤ 4 and 1 ≤ c ≤ 7*

After processing all *n = 3* coordinate pairs, the maximum value in any cell is *3*. Because there are two such cells with this maximal value, return *2* as the answer.

INTERVIEWER GUIDELINES

▼ Hint 1

Since number of rows and columns is of the order of 10^6, the construction of the matrix is impossible. Calculate the answer using rows and columns separately.

▼ Hint 2

Use difference array to update the range [1, row] and [1, column] in each operation.

▼ Solution

Concepts Covered: Basic Programming Skills, Loops, Arrays, Prefix sums, Counting, Problem Solving. The problem tests the candidate's ability to use loops, array handling, and the difference arrays. It requires the candidate to come up with an algorithm to find the number of cells with the maximum value after a series of range submatrix updates in a constrained time and space complexity.

Optimal Solution: We don't need to construct the whole matrix since it would not fit into the required space complexity.

Since for a particular operation, the whole submatrix from (1, 1) to (r, c) is updated by 1, we can calculate the number of times each row and column is updated. This can be done using difference array, which allows O(1) update over a range [l, r].

So let rows[] and columns[] be the two arrays of size = mx(where mx = 10^6 the maximum number of rows and columns possible), which denote that how many times a particular row or column, represented by rows[i] and columns[i] is updated.

Let (r, c) be the current operation. We can do the following:

rows[1]++, rows[r + 1]--

col[1]++, col[c + 1]--

/

col[r]++, col[c + 1]--

Finally we must take a prefix sum from left to right for each rows[] and columns[].
So to find the number of cells with the maximum number(max), we can calculate cnt_row and cnt_col.
cnt_row = The number of cells i, in rows[i] such that rows[i] = max
cnt_col = The number of cells i, in columns[i] such that columns[i] = max

So the answer = cnt_row * cnt_col.
Time Complexity: O(max(row, column)).

```
def countMax(upRight):
    n = len(upRight)
    # initialize arrays with zeros
    # size is fixed by constraints
    row = [0] * 1000005
    col = [0] * 1000005
    for i in range(n):
        # get the indices per query
        li = upRight[i].split(" ")
        # update appropriate rows and columns
        # for ranges where operations occur
        row[1] += 1
        row[int(li[0]) + 1] -= 1
        col[1] += 1
        col[int(li[1]) + 1] -= 1
    # calculate prefix sums by row and by column
    # while discovering the global maximum value
    sum1 = 0
    sum2 = 0
    mx = 0
    for i in range(1000005):
        sum1 += row[i]
        row[i] = sum1
        sum2 += col[i]
        col[i] = sum2
        mx = max(mx, row[i])
        mx = max(mx, col[i])
    # count the number of cells matching the global maximum
    cnt1 = 0
    cnt2 = 0
    for i in range(1000005):
        if(row[i] == mx):
            cnt1 += 1
        if(col[i] == mx):
            cnt2 += 1

    return cnt1 * cnt2
```

Brute Force Approach: Construct a matrix with the maximum possible rows and columns. For each query, update the whole submatrix from (1, 1) to (r, c) in O(n^2) complexity. After all the updates, find the number of cells in the matrix with the maximum value by iterating through the whole matrix.
Time Complexity: O(rows * columns * k), where the matrix has dimensions rows * columns, and k is the total number of operations.

Error Handling:
1. While updating the rows and columns, for implementing difference array trick, to update the range [l, r], its necessary to increment the index (r + 1) by 1 and not index r.
2. To count the value at each row and column index a prefix sum of both the difference arrays must be taken.
3. Since the maximum number of rows and columns is 10^6, so candidates must be careful to declare the row and column count array as the maximum size only and not less than that.

▼ Complexity Analysis

/

Time Complexity - O(mx), where mx = 10^6, the number of rows and columns possible.
Space Complexity - O(mx) .

▼ Follow up Question

Let's suppose we need now to count the number of cells in the matrix which have the minimum non-zero value.

Solution: We now count cnt_row and cnt_col, which denotes the number of rows and columns with value = min.

Psuedo Code -

```
def countMax(upRight):
    # Write your code here
    n = len(upRight)
    row = [0] * 1000005
    col = [0] * 1000005
    for i in range(n):
        li = upRight[i].split(" ")
        row[1] += 1
        row[int(li[0]) + 1] -= 1
        col[1] += 1
        col[int(li[1]) + 1] -= 1
    sum1 = 0
    sum2 = 0
    mn = 100000000
    for i in range(1000005):
        sum1 += row[i]
        row[i] = sum1
        sum2 += col[i]
        col[i] = sum2
        mn = min(mn, row[i])
        mn = min(mn, col[i])

    cnt1 = 0
    cnt2 = 0
    for i in range(1000005):
        if(row[i] == mn):
            cnt1 += 1
        if(col[i] == mn):
            cnt2 += 1

    return cnt1 * cnt2
```

▼ Follow up Question

Let's suppose now you are given queries Q, where you need to count the number of cells having value = Q[i].

Solution: After calculating rows[] and columns[], maintain two frequency arrays that denote the number of rows and columns which were updated freq[i] times.
So the solution to each query is ans[Q[i]] = freq_row[Q[i]] * freq_col[Q[i]].

Psuedo Code -

```
def countMax(upRight, query):
    # Write your code here
    n = len(upRight)
    row = [0] * 1000005
    col = [0] * 1000005
    for i in range(n):
        li = upRight[i].split(" ")
        row[1] += 1
        row[int(li[0]) + 1] -= 1
```

/

```
        col[1] += 1
        col[int(li[1]) + 1] -= 1
    sum1 = 0
    sum2 = 0
    mn = 100000000
    freq_row = [0] * (n + 1)
    freq_col = [0] * (n + 1)
    for i in range(1000005):
        sum1 += row[i]
        row[i] = sum1
        sum2 += col[i]
        col[i] = sum2
        freq_row[row[i]]++
        freq_col[col[i]]++

    ans = [0] * len(query)
    for i in range(len(query)):
        ans[i] = freq_row[query[i]] * freq_col[query[i]]

    return ans
```

CANDIDATE ANSWER

Language used: **C#**

```
class Result
{

    /*
     * Complete the 'countMax' function below.
     *
     * The function is expected to return a LONG_INTEGER.
     * The function accepts STRING_ARRAY upRight as parameter.
     */

    public static long countMax(List<string> upRight)
    {
        long max = long.MinValue;
        int[] rArray = new int[upRight.Count];
        int[] cArray = new int[upRight.Count];
        int i = 0;
        foreach(string s in upRight)
        {
            string[] splittedString = s.Split(' ');
            rArray[i] = int.Parse(splittedString[0]);
            cArray[i] = int.Parse(splittedString[1]);
            i++;
        }

        int rArrayMax = rArray.Max();
        int cArrayMax = cArray.Max();

        int[,] grid = new int[rArrayMax, cArrayMax];
        for(int j = 0; j < upRight.Count; j++)
        {
            int r = rArray[j];
            int c = cArray[j];
            for(int k = 0 ; k < r; k++)
            {
                for(int l = 0 ; l < c; l++)
```

/

```
                {
                    grid[k,l] ++;
                    if(max < grid[k,l])
                    {
                        max = grid[k,l];
                    }
                }
            }
        }

        return max;
    }

}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample case	Wrong Answer	0	0.0761 sec	18.3 KB
Testcase 1	Easy	Sample case	Wrong Answer	0	0.0866 sec	18.3 KB
Testcase 2	Easy	Sample case	Wrong Answer	0	0.1008 sec	18 KB
Testcase 3	Easy	Sample case	Wrong Answer	0	0.0827 sec	18.1 KB
Testcase 4	Easy	Sample case	Terminated due to timeout	0	3.0079 sec	110 KB
Testcase 5	Easy	Hidden case	Runtime Error	0	0.0778 sec	17.2 KB
Testcase 6	Easy	Hidden case	Runtime Error	0	0.079 sec	17.4 KB
Testcase 7	Easy	Hidden case	Runtime Error	0	0.0864 sec	17.4 KB
Testcase 8	Easy	Hidden case	Runtime Error	0	0.0988 sec	17.2 KB
Testcase 9	Easy	Hidden case	Runtime Error	0	0.0762 sec	17.3 KB
Testcase 10	Easy	Hidden case	Wrong Answer	0	0.0802 sec	18.1 KB

No Comments

QUESTION 2

Correct Answer

Score 50

Youngest Employees › DbRank

SQL | Easy | Joins | Interviewer Guidelines

QUESTION DESCRIPTION

There are two data tables with employee information: *EMPLOYEE* and *EMPLOYEE_UIN*. Query the tables to generate a list of all employees who are less than *25* years old first in order of *NAME*, then of *ID*, both ascending. The result should include the *UIN* followed by the *NAME*.

Note: While the secondary sort is by *ID*, the result includes *UIN* but not *ID*.

▼ Schema

/

EMPLOYEE		
Name	**Type**	**Description**
ID	Integer	The ID of the employee. This is a primary key.
NAME	String	The name of the employee having *[1, 20]* characters.
AGE	Integer	The age of the employee.
ADDRESS	String	The address of the employee having *[1, 25]* characters.
SALARY	Integer	The salary of the employee.

EMPLOYEE_UIN		
Name	**Type**	**Description**
ID	Integer	The ID of the employee. This is a primary key.
UIN	String	The unique identification number of the employee.

▼ Sample Data Tables

Sample Input

EMPLOYEE				
ID	**NAME**	**AGE**	**ADDRESS**	**SALARY**
1	Sherrie	23	Paris	74635
2	Paul	30	Sydney	72167
3	Mary	24	Paris	75299
4	Sam	47	Sydney	46681
5	Dave	22	Texas	11843

EMPLOYEE_UIN	
ID	**UIN**
1	57520-0440
2	49638-001
3	63550-194
4	68599-6112
5	63868-453

Sample Output

```
63868-453 Dave
63550-194 Mary
57520-0440 Sherrie
```

Explanation

/

- *Sherrie* is *23* years old and has UIN *57520-0440*. This record is printed.
- *Paul* is *30* years old and has UIN *49638-001*. This record is not printed.
- A similar analysis is done on the remaining records.

None of the three names of people less than *25* years old is repeated, so print them in alphabetical order. There is no additional sorting by *ID* in this case.

INTERVIEWER GUIDELINES

▼ Solution

Concepts covered: e.g. JOIN, ORDER BY

Solution:

Join the tables to get UIN. Filter results to age < 25 and sort ascending by name, id.

```
SELECT eu.uin, e.name
FROM employee e
JOIN employee_uin eu
ON e.id = eu.id
WHERE e.age < 25
ORDER BY e.name, e.id;
```

Note that the secondary sort is on ID but data reported is UIN.

CANDIDATE ANSWER

Language used: **MS SQL**

```
/*
Enter your query here.
Please append a semicolon ";" at the end of the query and enter your query in
a single line to avoid error.
*/
SELECT eUIN.UIN, e.NAME FROM EMPLOYEE e
LEFT JOIN EMPLOYEE_UIN eUIN on e.ID = eUIN.ID --INNER JOIN if we want to
exclude potential missing employees in EMPLOYEE_UIN
WHERE e.AGE < 25
ORDER BY e.NAME, e.ID;
```

Time taken: **0.04 sec**

No Comments

QUESTION 3

Correct Answer

Score 37

Profit Targets › Coding

Binary Search | Data Structures | Medium | Algorithms | Arrays | Problem Solving | Theme: Finance

QUESTION DESCRIPTION

A financial analyst is responsible for a portfolio of profitable stocks represented in an array. Each item in the array represents the yearly profit of a corresponding stock. The analyst gathers all distinct pairs of stocks that reached the target profit. Distinct pairs are pairs that differ in at least one element. Given the array of profits, find the number of distinct pairs of stocks where the sum of each pair's profits is exactly equal to the target profit.

Example

stocksProfit = [5, 7, 9, 13, 11, 6, 6, 3, 3]

target = 12 profit's target

/

- There are *4* pairs of stocks that have the sum of their profits equals to the target 12 . Note that because there are two instances of *3* in *stocksProfit* there are two pairs matching (9, 3): *stocksProfits* indices 2 and 7, and indices 2 and 8, but only one can be included.
- There are *3* distinct pairs of stocks: *(5, 7), (3, 9),* and *(6, 6)* and the return value is *3*.

Function Description

Complete the function *stockPairs* in the editor below.

stockPairs has the following parameter(s):

int stocksProfit[n]: an array of integers representing the stocks profits

target: an integer representing the yearly target profit

Returns:

int: the total number of pairs determined

Constraints

- $1 \le n \le 5 \times 10^5$
- $0 \le stocksProfit[i] \le 10^9$
- $0 \le target \le 5 \times 10^9$

▼ 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 *stocksProfit*.

The next *n* lines each contain an element *stocksProfit[i]* where $0 \le i < n$.

The next line contains an integer *target*, the target value.

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
-----       --------
6       →   stocksProfit[] size n = 6
1       →   stocksProfit = [1, 3, 46, 1, 3, 9]
3
46
1
3
9
47      →   target = 47
```

Sample Output 0

```
1
```

Explanation 0

There are *4* pairs where *stocksProfit[i] + stocksProfit[j]* = 47

1. *(stocksProfit0] = 1, stocksProfit[2] = 46)*
2. *(stocksProfit[2] = 46, stocksProfit[0] = 1)*
3. *(stocksProfit[2] = 46, stocksProfit[3] = 1)*
4. *(stocksProfit[3] = 1, stocksProfit[2] = 46)*

Since all four pairs contain the same values, there is only *1 distinct* pair of stocks : *(1, 46)*.

▼ Sample Case 1

Sample Input 1

```
STDIN       Function
-----       --------
7       →   stocksProfit[] size n = 7
```

/

```
6      →   stocksProfit = [6, 6, 3, 9, 3, 5, 1]
6
3
9
3
5
1
12     →   target = 12
```

Sample Output 1

```
2
```

Explanation 1

There are *5* pairs where *stocksProfit[i] + stocksProfit[j]* = 12:

1. *(stocksProfit[0] = 6, stocksProfit[1] = 6)*
2. *(stocksProfit[1] = 6, stocksProfit[0] = 6)*
3. *(stocksProfit[2] = 3, stocksProfit[3] = 9)*
4. *(stocksProfit[3] = 9, stocksProfit[2] = 3)*
5. *(stocksProfit[3] = 9, stocksProfit[4] = 3)*
6. *(stocksProfit[4] = 3, stocksProfit[3] = 9)*

The first *2* pairs are the same, as are the last *4*. There are only *2 distinct* pairs of stocks: *(3, 9)* and *(6, 6)*.

CANDIDATE ANSWER

Language used: **C#**

```
class Result
{

    /*
     * Complete the 'stockPairs' function below.
     *
     * The function is expected to return an INTEGER.
     * The function accepts following parameters:
     *  1. INTEGER_ARRAY stocksProfit
     *  2. LONG_INTEGER target
     */

    public static int stockPairs(List<int> stocksProfit, long target)
    {
        Dictionary<long, long> uniquePairs = new Dictionary<long, long>();
        for(int i = 0 ; i < stocksProfit.Count ; i++)
        {
            for(int j = i + 1 ; j < stocksProfit.Count ; j++)
            {
                if(stocksProfit[i] + stocksProfit[j] == target)
                {
                    uniquePairs[Math.Min(stocksProfit[i], stocksProfit[j])] =
Math.Max(stocksProfit[i], stocksProfit[j]);
                }
            }
        }
        return uniquePairs.Count;
    }

}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED

/

					TAKEN	USED
TestCase 0	Easy	Sample case	Success	1	0.0723 sec	17.4 KB
TestCase 1	Easy	Sample case	Success	1	0.0761 sec	17.5 KB
TestCase 2	Easy	Sample case	Success	1	0.0754 sec	17.5 KB
TestCase 3	Easy	Hidden case	Success	2	0.076 sec	17.5 KB
TestCase 4	Easy	Hidden case	Success	2	0.0727 sec	17.4 KB
TestCase 5 - O(N^2)	Easy	Sample case	Success	2	0.0736 sec	17.4 KB
TestCase 6 - O(N^2)	Easy	Hidden case	Success	2	0.0797 sec	17.5 KB
TestCase 7 - O(N^2)	Easy	Hidden case	Success	2	0.0666 sec	17.7 KB
TestCase 8 - O(N^2)	Medium	Hidden case	Success	4	0.1659 sec	17.7 KB
TestCase 9 - O(N^2)	Medium	Hidden case	Success	4	0.0979 sec	17.7 KB
TestCase 10 - O(N^2)	Medium	Hidden case	Success	5	0.2988 sec	17.7 KB
TestCase 11	Medium	Sample case	Success	5	0.477 sec	18.1 KB
TestCase 12 - O(N^2)	Medium	Hidden case	Success	6	0.2549 sec	17.6 KB
TestCase 14 - O(NlogN)	Hard	Hidden case	Terminated due to timeout	0	3.004 sec	21.3 KB
TestCase 16 - O(NlogN)	Hard	Hidden case	Terminated due to timeout	0	3.0063 sec	22.4 KB

No Comments

PDF generated at: 18 Aug 2020 09:09:33 UTC

/