

WILSON RAMON MARTINEZ BARRERA

Candidate

E-mail: wilsonmb1989@gmail.com

Session

ID: FYR48N-V2P

Time limit: 180 min.

Report recipients:

luisa.chica@pragma.com.co

Accessed from 181.136.109.40


Status: closed

Invitation: [sent](#)

Created on: 2014-07-21 22:43 UTC

Started on: 2014-07-23 02:06 UTC

Finished on: 2014-07-23 05:06 UTC

 Your notes:
No sigue

Tasks in test

1 |  SqlSensorsMostRecent

2 |  FrogPond

3 |  Gcd

4 |  CountriesCount

Correctness

58%

0%

16%

0%

Performance

20%

0%

0%

0%

Task score

47%

0%

11%

0%

Test score

15%

58 out of 400 points

1. SqlSensorsMostRecent

Compute the most recent value for each sensor and event type.

score: 47 of 100**Task description**

Given a table events with the following structure:

```
create table events (
  sensor_id integer not null,
  event_type integer not null,
  value integer not null,
  time timestamp unique not null
);
```

write an SQL query that, for each sensor and event type (sensor_id, event_type), returns the most recent value (in terms of time). The table should be ordered by sensor_id (in ascending order), event_type (in ascending order).

For example, given the following data:

| sensor_id | event_type | value | time |
|-----------|------------|-------|---------------------|
| 2 | 2 | 5 | 2014-02-13 12:42:00 |
| 2 | 4 | -42 | 2014-02-13 13:19:57 |
| 2 | 2 | 2 | 2014-02-13 14:48:30 |
| 3 | 2 | 7 | 2014-02-13 12:54:39 |
| 2 | 3 | 54 | 2014-02-13 13:32:36 |

your query should return the following rowset:

| sensor_id | event_type | value |
|-----------|------------|-------|
| 2 | 2 | 2 |
| 2 | 3 | 54 |
| 2 | 4 | -42 |
| 3 | 2 | 7 |

The names of the columns in the rowset don't matter, but their order does.

Copyright 2009–2014 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

Solution

Programming language used: SQL

Total time used: 25 minutes

Effective time used: 25 minutes

Notes: *not defined yet*

Source code

Code: 02:30:31 UTC, sql, final, score: **47.00**

```
1 -- write your code in SQLite
2 SELECT events.sensor_id, events.event_type, events.value
3 FROM (SELECT sensor_id, event_type, value, MAX(time)
4       FROM events
5       GROUP BY sensor_id, event_type) AS tabla
6 JOIN events on tabla.sensor_id = events.sensor_id
7 AND tabla.event_type = events.event_type
8 AND tabla.value = events.value
9 ORDER BY events.sensor_id ASC
```

Analysis

| test | time | result |
|---|---------|---|
| Example tests | | |
| example example test | 0.088 s | OK |
| Correctness tests | | |
| simple1 (0,0), (1,1) | 0.108 s | OK |
| simple_double_event (0,0), (0,0) | 0.108 s | WRONG ANSWER got [(0, 0, 5)] expected [(0, 0, 2)] |
| simple_double_type (0, 0), (1, 0) | 0.080 s | OK |
| simple_double_sensor (0, 0), (0, 1) | 0.108 s | OK |
| all_unique (0,0), (1,1), ... , (10,10) | 0.120 s | OK |
| all_unique_reversed (10,10), (9,9), ... , (0,0) | 0.100 s | OK |
| single_et (0,0), (1,0), ... , (10, 0) | 0.096 s | OK |
| double_events two events with the same event type for every sensor | 0.104 s | OK |
| double_shuffled Two events with the same et for every sensor, shuffled | 0.100 s | WRONG ANSWER got [(0, 4, 42)], '(0, 4, 42)', '(1, 4, .. expected [(0, 4, 42)', '(1, 4, 43)', '(2, 4, .. |
| medium_random1 random sequence; N=1,000, 50 sensors, 100 types | 0.168 s | WRONG ANSWER got ..., 8442)', '(1, 31, -608)', '(1, 45, 431).. expected ..., 8442)', '(1, 31, -3837)', '(1, 45, 431).. |
| medium_random2 random sequence; N=1,000, 4 sensors, 70 types | 0.148 s | WRONG ANSWER got [(1, 1, 1817)', '(1, 2, -954.. expected [(1, 1, -544)', '(1, 2, 4702.. |
| medium_random3 random sequence; N=1,000, 20 sensors, 3 types | 0.148 s | WRONG ANSWER got [(1, 1, 8664)', '(1, 2, 9735.. expected [(1, 1, 5974)', '(1, 2, 5984.. |
| Performance tests | | |
| big_different (0, 0), (1, 0), ... , (3000, 0) | 0.268 s | OK |
| big_same (0, 0) multiple times (3000) | 0.236 s | WRONG ANSWER got [(0, 0, 1078)] expected [(0, 0, 3003)] |
| big_single_sensor 3,000 readings for the same sensor, random types | 0.228 s | WRONG ANSWER got ..4, -5071)', '(1, 5, -5834)', '(1, 6, 216.. expected ..4, -5071)', '(1, 5, 3417)', '(1, 6, 2168.. |

| | | |
|--|---------|--|
| big_random1 3,000 events, up to 1000 sensors | 0.260 s | WRONG ANSWER got ..2)', '(834, 429, -6542)', '(834, 539, 80.. expected ..2)', '(834, 429, -6522)', '(834, 539, 80.. |
| big_random2 3,000 events, up to 300 sensors | 0.264 s | WRONG ANSWER got ..1808)', '(28, 779, -4435)', '(28, 883, -.. expected ..1808)', '(28, 779, -1004)', '(28, 883, -.. .. |

HARD

2. FrogPond

Find the earliest time when a frog can jump across a pond.

score: 0 of 100

Task description

A small frog wants to get to the other side of a pond. The frog is initially located on one bank of the pond (position 0) and wants to get to the other bank (position X). The frog can jump any (integer) distance between 1 and D. If $X > D$ then the frog cannot jump right across the pond. Luckily, there are leaves falling from a tree onto the surface of the pond, and the frog can jump onto and from the leaves. You are given a zero-indexed array A consisting of N integers. This array represents falling leaves. Initially there are no leaves. A[K] represents the position where a leaf will fall in second K. The goal is to find the earliest time when the frog can get to the other side of the pond. The frog can jump from and to positions 0 and X (the banks of the pond) and every position with a leaf. For example, you are given integers $X = 7$, $D = 3$ and array A such that:

```
A[0] = 1
A[1] = 3
A[2] = 1
A[3] = 4
A[4] = 2
A[5] = 5
```

Initially, the frog cannot jump across the pond in a single jump. However, in second 3, after a leaf falls at position 4, it becomes possible for the frog to cross. This is the earliest moment when the frog can jump across the pond (by jumps of length 1, 3 and 3). Write a function:

```
class Solution { public int solution(int[] A, int X,
int D); }
```

that, given a zero-indexed array A consisting of N integers, and integers X and D, returns the earliest time when the frog can jump to the other side of the pond. If the frog can leap across the pond in just one jump, the function should return 0. If the frog is never able to jump to the other side of the pond, the function should return -1. For example, given $X = 7$, $D = 3$ and array A such that:

```
A[0] = 1
A[1] = 3
A[2] = 1
A[3] = 4
A[4] = 2
A[5] = 5
```

the function should return 3 as explained above. Assume that:

- N is an integer within the range $[0..100,000]$;
- X and D are integers within the range $[1..100,000]$;
- each element of array A is an integer within the range $[1..X-1]$.

Complexity:

- expected worst-case time complexity is $O(N)$;
- expected worst-case space complexity is $O(X)$, beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

Copyright 2009–2014 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

Solution

Programming language used: Java

Total time used: 38 minutes

Effective time used: 38 minutes

Notes: not defined yet

Source code

Code: 03:07:52 UTC, java, final, score: 0.00

```
1 // you can also use imports, for example:
2 // import java.math.*;
3
4 // you can use System.out.println for debugging
  purposes, e.g.
5 // System.out.println("this is a debug message");
6
7 class Solution {
8     public int solution(int[] A, int X, int D) {
9         if(X > D){
10             int cont = 0;
11             int sum = 0;
12             for(int val = 0 ; val < A.length ; val++){
13                 sum = A[val] + sum;
14                 if(sum >= X){
15                     break;
16                 }else{
17                     cont++;
18                 }
19             }
20             if(sum >= X){
21                 return cont;
22             }else{
23                 return -1;
24             }
25         }else{
26             return -1;
27         }
28     }
29 }
```

Analysis

| test | time | result |
|---|---------|--------------------------------------|
| Example tests | | |
| example example test | 1.332 s | OK |
| Correctness tests | | |
| extreme_empty empty array | 1.304 s | WRONG ANSWER got -1 expected 0 |
| simple simple test | 1.108 s | WRONG ANSWER got 2 expected 3 |
| simple2 simple test | 1.364 s | WRONG ANSWER got 1 expected 6 |
| single single element | 1.316 s | WRONG ANSWER got -1 expected 0 |
| extreme_frog1 frog can never get across the pond | 1.324 s | WRONG ANSWER got 2 expected -1 |
| extreme_frog2 frog can never get across the pond | 1.344 s | WRONG ANSWER got 1 expected 0 |
| small_random1 3 random permutation, D = 1 | 1.268 s | WRONG ANSWER got 3 expected 83 |
| small_random2 3 random permutation, D = 2 | 1.376 s | WRONG ANSWER got 3 expected 76 |
| Performance tests | | |
| medium_random1 6 random permutation, D = 1, 3 | 1.484 s | WRONG ANSWER got 1 expected 13472 |
| medium_random2 2 random permutation, D = 11, 193 | 1.392 s | WRONG ANSWER got 1 expected 3768 |
| large_random1 10 random permutation, D = 8, 1976 | 1.544 s | WRONG ANSWER got 2 expected 9416 |

| | | |
|---|---------|---|
| large_random2 100 random permutation, D = 1, 53, 900 | 1.516 s | WRONG ANSWER got 3 expected 6725 |
| large_permutation permutation tests | 1.552 s | WRONG ANSWER got 423 expected 88084 |
| large_range arithmetic sequences, D = 3, 1942 | 1.356 s | WRONG ANSWER got 244 expected 34998 |
| extreme_leaves all leaves in the same place | 1.380 s | WRONG ANSWER got 2 expected -1 |
| large_jump very big jumps | 1.560 s | WRONG ANSWER got -1 expected 0 |

EASY

3. Gcd

Find greatest common divisor of two positive integers.

score: 11 of 100

Task description

Two positive integers N and M are given. Their *greatest common divisor* is the largest integer D that divides both N and M. For example, the greatest common divisor of 24 and 18 is 6.

Write a function

```
class Solution { public int solution(int N, int M);
}
```

that, given two positive integers N and M, returns their greatest common divisor.

Assume that:

- M and N are integers within the range [1..2,000,000,000].

For example, given N = 24 and M = 18, the function should return 6.

Complexity:

- expected worst-case time complexity is $O(\log(N+M))$;
- expected worst-case space complexity is $O(1)$.

Copyright 2009–2014 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

Solution

Programming language used: Java

Total time used: 28 minutes

Effective time used: 28 minutes

Notes: not defined yet

Source code

Code: 03:35:48 UTC, java, final, score: 11.00

```
1 // you can also use imports, for example:
2 // import java.math.*;
3
4 // you can use System.out.println for debugging
  purposes, e.g.
5 // System.out.println("this is a debug message");
6
7 class Solution {
8     public int solution(int N, int M) {
9         // write your code in Java SE 8
10        int conDivN = 0;
11        int conDivM = 0;
12        for(int i = 0 ; i < N ; i++){
13            if(N % (i+1) == 0){
14                conDivN++;
15            }
16        }
17        for(int i = 0 ; i < M ; i++){
18            if(M % (i+1) == 0){
19                conDivM++;
20            }
21        }
22        int divN[] = new int[conDivN];
23        int divM[] = new int[conDivM];
24        int contN = 0;
25        for(int i = 0 ; i < N ; i++){
26            if(N % (i+1) == 0){
27                divN[contN] = (i+1);
28                contN++;
29            }
30        }
31        int contM = 0;
32        for(int i = 0 ; i < M ; i++){
33            if(M % (i+1) == 0){
34                divM[contM] = (i+1);
35                contM++;
36            }
37        }
38        int iterator = contN <= contM ? contN : contM;
39        int MCM = 1;
40        for(int i = 0; i < iterator ; i++){
41            if(divN[i] == divM[i]){
42                MCM = MCM * divN[i];
43            }
44        }
45        return MCM;
46    }
47 }
```

Analysis

| test | time | result |
|--------------------------------------|---------|---|
| Example tests | | |
| example1 example test, N=24, M=18 | 1.300 s | OK |
| Correctness tests | | |
| extreme1 N=M | 1.280 s | WRONG ANSWER got 3111696 expected 42 |
| extreme2 N or M equals 1 | 1.244 s | WRONG ANSWER got 100 expected 10 |
| small1 N=123, M=321 | 1.224 s | OK |
| small2 N=100, M=80 | 1.332 s | WRONG ANSWER got 40 expected 20 |
| small3 N=119, M=544 | 1.320 s | WRONG ANSWER got 1 expected 17 |

| | | |
|---|----------|---|
| medium1 N=10000, M=8000 | 1.296 s | WRONG ANSWER got 25600000 expected 2000 |
| Performance tests | | |
| large1 N=123M, M=789M | >9.000 s | TIMEOUT ERROR running time: >9.00 sec., time limit: 3.30 sec. |
| large2 N=(3**10)*(2**15), M=(2**15)*(2**15) | >9.000 s | TIMEOUT ERROR running time: >9.00 sec., time limit: 3.51 sec. |
| extreme3 Maximal values | >8.000 s | TIMEOUT ERROR running time: >8.00 sec., time limit: 2.04 sec. |

4. CountriesCount

The number of different countries that a map contains.

score: 0 of 100

Task description

A rectangular map consisting of N rows and M columns of square areas is given. Each area is painted with a certain color. Two areas on the map *belong to the same country* if the following conditions are met:

- they have the same color;
- it is possible to travel from one area to the other by moving only north, south, west or east without moving over areas of a different color.

The map can be described by a zero-indexed matrix consisting of N rows and M columns of integers. The color of each area is described by the corresponding element of the matrix. Two areas have the same color if and only if their corresponding matrix elements have the same value.

For example, consider the following matrix A consisting of seven rows and three columns:

| | | |
|-------------|-------------|-------------|
| A[0][0] = 5 | A[0][1] = 4 | A[0][2] = 4 |
| A[1][0] = 4 | A[1][1] = 3 | A[1][2] = 4 |
| A[2][0] = 3 | A[2][1] = 2 | A[2][2] = 4 |
| A[3][0] = 2 | A[3][1] = 2 | A[3][2] = 2 |
| A[4][0] = 3 | A[4][1] = 3 | A[4][2] = 4 |
| A[5][0] = 1 | A[5][1] = 4 | A[5][2] = 4 |
| A[6][0] = 4 | A[6][1] = 1 | A[6][2] = 1 |

Matrix A describes a map that is colored with five colors. Areas on the map belong to eleven different countries:

- area A[0][0] forms a one-area country;
- areas A[0][1], A[0][2], A[1][2], A[2][2] belong to the same country;
- area A[1][0] forms a one-area country;
- area A[1][1] forms a one-area country;
- area A[2][0] forms a one-area country;
- areas A[2][1], A[3][0], A[3][1], A[3][2] belong to the same country;
- areas A[4][0], A[4][1] belong to the same country;
- areas A[4][2], A[5][1], A[5][2] belong to the same country;
- area A[5][0] forms a one-area country;
- area A[6][0] forms a one-area country;
- areas A[6][1], A[6][2] belong to the same country.

Write a function

```
class Solution { public int solution(int[][] A); }
```

that, given a zero-indexed matrix A consisting of N rows and M columns of integers, returns the number of different countries that the areas of the map described by matrix A belong to.

Assume that:

- N and M are integers within the range [1..300,000];
- the number of elements in matrix A is within the range [1..300,000];
- each element of matrix A is an integer within the range [-1,000,000,000..1,000,000,000].

For example, given matrix A consisting of seven rows and three columns corresponding to the example above, the function should return 11.

Complexity:

- expected worst-case time complexity is $O(N \cdot M)$;
- expected worst-case space complexity is $O(N \cdot M)$.

Copyright 2009–2014 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

Solution

Programming language used: Java

Total time used: 91 minutes

Effective time used: 91 minutes

Notes: not defined yet

Source code

Code: 05:05:53 UTC, java, final, score: 0.00

```
1 // you can also use imports, for example:
2 // import java.math.*;
3
4 // you can use System.out.println for debugging
  purposes, e.g.
5 // System.out.println("this is a debug message");
6
7 class Solution {
8     public int solution(int[][] A) {
9         // write your code in Java SE 8
10        int pivotF = A.length;
11        int pivotC = A[0].length;
12        System.out.println("pivotF: "+pivotF);
13        System.out.println("pivotC: "+pivotC);
14        int matrixSize = pivotF * pivotC;
15        int cont = -1;
16        int pivot = 0;
17        for(int y = 0 ; y < matrixSize ; y++){
18            int contF = 0;
19            int contC = 0;
20            for(int x = 0 ; x < matrixSize ; x++){
21                System.out.println("A["+contF+"]["+contC+"]");
22                pivot = A[contF][contC];
23                if(contC - 1 >= 0 && (A[contF][contC-1]
24                == pivot)){
25                    A[contF][contC-1] = cont;
26                }
27                if(contC + 1 <= pivotC-1 &&
28                A[contF][contC+1] == pivot){
29                    A[contF][contC+1] = cont;
30                }
31                if(contF - 1 >= 0 && A[contF-1][contC]
32                == pivot){
33                    A[contF-1][contC] = cont;
34                }
35                if(contF + 1 <= pivotF-1 &&
36                A[contF+1][contC] == pivot){
37                    A[contF+1][contC] = cont;
38                }
39                contC++;
40                if(contC > pivotC-1){
41                    contC = 0;
42                    contF++;
43                    if(contF > pivotF-1){
44                        contF = 0;
45                    }
46                }
47                cont--;
48            }
49        }
50        return 0;
51    }
52 }
```

Analysis

| test | time | result |
|--|---------|-----------------------------------|
| Example tests | | |
| example | 1.564 s | WRONG ANSWER got 0 expected 11 |
| stdout: | | |
| <pre> pivotF: 7 pivotC: 3 A[0][0] A[0][1] A[0][2] A[1][0] A[1][1] </pre> | | |

| Correctness tests | | |
|--|-----------|--|
| small_1x1 | 1.448 s | WRONG ANSWER got 0 expected 1 |
| stdout: pivotF: 1 pivotC: 1 A[0][0] | | |
| positive_negative_zeros | 1.480 s | WRONG ANSWER got 0 expected 5 |
| stdout: pivotF: 3 pivotC: 3 A[0][0] A[0][1] A[0][2] A[1][0] A[1][1] | | |
| small_2x2 | 1.432 s | WRONG ANSWER got 0 expected 4 |
| stdout: pivotF: 2 pivotC: 2 A[0][0] A[0][1] A[1][0] A[1][1] A[0][0] | | |
| small_3x3 | 1.360 s | WRONG ANSWER got 0 expected 9 |
| stdout: pivotF: 3 pivotC: 3 A[0][0] A[0][1] A[0][2] A[1][0] A[1][1] | | |
| matrix_12x10 | 3.784 s | WRONG ANSWER got 0 expected 120 |
| stdout: pivotF: 10 pivotC: 12 A[0][0] A[0][1] A[0][2] A[0][3] A[0][4] | | |
| matrix_10x10_labyrinth | 3.008 s | WRONG ANSWER got 0 expected 15 |
| stdout: pivotF: 10 pivotC: 10 A[0][0] A[0][1] A[0][2] A[0][3] A[0][4] | | |
| matrix_wide | 1.348 s | WRONG ANSWER got 0 expected 5 |
| stdout: pivotF: 1 pivotC: 11 A[0][0] A[0][1] A[0][2] A[0][3] A[0][4] | | |
| large_numbers | >12.000 s | TIMEOUT ERROR running time: >12.00 sec., time limit: 8.60 sec. |
| stdout: pivotF: 1 pivotC: 200 A[0][0] A[0][1] A[0][2] A[0][3] A[0][4] | | |
| anti_heuristics | 1.508 s | WRONG ANSWER got 0 expected 2 |
| stdout: pivotF: 2 pivotC: 3 A[0][0] A[0][1] A[0][2] A[1][0] A[1][1] | | |
| Performance tests | | |

| | | |
|---|-----------|--|
| matrix_max | >14.000 s | TIMEOUT ERROR running time: >14.00 sec., time limit: 8.09 sec. |
| stdout: pivotF: 711 pivotC: 400 A[0][0] A[0][1] A[0][2] A[0][3] A[0][4] | | |
| matrix_big | >12.000 s | TIMEOUT ERROR running time: >12.00 sec., time limit: 6.70 sec. |
| stdout: pivotF: 603 pivotC: 402 A[0][0] A[0][1] A[0][2] A[0][3] A[0][4] | | |
| large_square_matrix | >12.000 s | TIMEOUT ERROR running time: >12.00 sec., time limit: 6.71 sec. |
| stdout: pivotF: 500 pivotC: 500 A[0][0] A[0][1] A[0][2] A[0][3] A[0][4] | | |
| large_horizontal_matrix | >12.000 s | TIMEOUT ERROR running time: >12.00 sec., time limit: 6.57 sec. |
| stdout: pivotF: 5 pivotC: 50000 A[0][0] A[0][1] A[0][2] A[0][3] A[0][4] | | |
| large_one_country | >11.000 s | TIMEOUT ERROR running time: >11.00 sec., time limit: 5.99 sec. |
| stdout: pivotF: 1 pivotC: 300000 A[0][0] A[0][1] A[0][2] A[0][3] A[0][4] | | |
| large_square_one_country | >11.000 s | TIMEOUT ERROR running time: >11.00 sec., time limit: 5.32 sec. |
| stdout: pivotF: 400 pivotC: 400 A[0][0] A[0][1] A[0][2] A[0][3] A[0][4] | | |

