Filling Jars

locked

Problem Submissions Leaderboard Discussions

Animesh has n empty candy jars, numbered from 1 to n, with infinite capacity. He performs m operations. Each operation is described by 3 integers, a, b, and k. Here, a and b are indices of the jars, and k is the number of candies to be added inside each jar whose index lies between a and b (both inclusive). Can you tell the average number of candies after m operations?

Example

$n = 5$
$operations = [[1, 2, 10], [3, 5, 10]]$

The array has 5 elements that all start at 0. In the first operation, add 10 to the first 2 elements. Now the array is $[10, 10, 0, 0, 0]$. In the second operation, add 10 to the last 3 elements (3 - 5). Now the array is $[10, 10, 10, 10, 10]$ and the average is 10. Sincd 10 is already an integer value, it does not need to be rounded.

Function Description

Complete the *solve* function in the editor below.

solve has the following parameters:

- *int n:* the number of candy jars
- *int operations[m][3]:* a 2-dimensional array of operations

Returns

- *int:* the floor of the average number of canidies in all jars

Input Format

The first line contains two integers, n and m, separated by a single space.
m lines follow. Each of them contains three integers, a, b, and k, separated by spaces.

Constraints

$3 \leq n \leq 10^7$
$1 \leq m \leq 10^5$
$1 \leq a \leq b \leq N$
$0 \leq k \leq 10^6$

Sample Input

```
STDIN       Function
-----       --------
5 3         n = 5, operations[] size = 3
1 2 100     operations = [[1, 2, 100], [2, 5, 100], [3, 4, 100]]
2 5 100
3 4 100
```

Sample Output

```
160
```

Explanation

Initially each of the jars contains *0* candies

```
0 0 0 0 0
```

First operation:

```
100 100 0 0 0
```

Second operation:

```
100 200 100 100 100
```

Third operation:

```
100 200 200 200 100
```

Total = *800*, Average = *800/5* = *160*

Submissions: 17
Max Score: 40
Difficulty: Easy

Rate This Challenge:

More

Clojure

```
;
; Complete the 'solve' function below.
;
; The function is expected to return an INTEGER.
; The function accepts following parameters:
;  1. INTEGER n
;  2. 2D_INTEGER_ARRAY operations
;

(defn solve [n operations]

)

(def fptr (get (System/getenv) "OUTPUT_PATH"))

(def first-multiple-input (clojure.string/split (clojure.string/trimr (read-line)) #" "))

(def n (Integer/parseInt (nth first-multiple-input 0)))

(def m (Integer/parseInt (nth first-multiple-input 1)))

(def operations [])

(doseq [_ (range m)]
    (def operations (conj operations (vec (map #(Integer/parseInt %) (clojure.string/split (clojure.string/trimr (read-line)) #" "))))))

(def result (solve n operations))

(spit fptr (str result "\n") :append true)
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code