#Consider two points, and . We consider the inversion or point reflection, , of point across point to be a rotation of point around .

#Given sets of points and , find for each pair of points and print two space-separated integers denoting the respective values of and on a new line.

#Function Description

#Complete the findPoint function in the editor below.

#findPoint has the following parameters:

#int px, py, qx, qy: x and y coordinates for points and

#Returns

#int[2]: x and y coordinates of the reflected point

#Input Format

#The first line contains an integer, , denoting the number of sets of points.

#Each of the subsequent lines contains four space-separated integers that describe the respective values of , , , and defining points and .

#Constraints

#Sample Input

#2

#0 0 1 1

#1 1 2 2

#Sample Output

#2 2

#3 3

import math

import os

import random

import re

import sys

def findPoint(px, py, qx, qy):

p2q\_x = qx - px

p2q\_y = qy - py

r\_pt = [qx + p2q\_x, qy + p2q\_y]

return r\_pt

if \_\_name\_\_ == '\_\_main\_\_':

fptr = open(os.environ['OUTPUT\_PATH'], 'w')

n = int(input().strip())

for n\_itr in range(n):

first\_multiple\_input = input().rstrip().split()

px = int(first\_multiple\_input[0])

py = int(first\_multiple\_input[1])

qx = int(first\_multiple\_input[2])

qy = int(first\_multiple\_input[3])

result = findPoint(px, py, qx, qy)

fptr.write(' '.join(map(str, result)))

fptr.write('\n')

fptr.close()

Find.py

#Cities on a map are connected by a number of roads. The number of roads between each city is in an array and city is the starting location. The number of roads from city to city is the first value in the array, from city to city is the second, and so on.

#How many paths are there from city to the last city in the list, modulo ?

#Example

#There are roads to city , roads to city and roads to city . The total number of roads is .

#Note

#Pass all the towns Ti for i=1 to n-1 in numerical order to reach Tn.

#Function Description

#Complete the connectingTowns function in the editor below.

#connectingTowns has the following parameters:

#int n: the number of towns

#int routes[n-1]: the number of routes between towns

#Returns

#int: the total number of routes, modulo 1234567.

#Input Format

#The first line contains an integer T, T test-cases follow.

#Each test-case has 2 lines.

#The first line contains an integer N (the number of towns).

#The second line contains N - 1 space separated integers where the ith integer denotes the number of routes, Ni, from the town Ti to Ti+1

#Constraints

#1 <= T<=1000

#2< N <=100

#1 <= routes[i] <=1000

#Sample Input

#2

#3

#1 3

#4

#2 2 2

#Sample Output

#3

#8

import math

import os

import random

import re

import sys

def connectingTowns(n, routes):

r = 1

for i in routes:

r \*= i

return r%1234567

if \_\_name\_\_ == '\_\_main\_\_':

fptr = open(os.environ['OUTPUT\_PATH'], 'w')

t = int(input().strip())

for t\_itr in range(t):

n = int(input().strip())

routes = list(map(int, input().rstrip().split()))

result = connectingTowns(n, routes)

fptr.write(str(result) + '\n')

fptr.close()

#Kristen loves playing with and comparing numbers. She thinks that if she takes two different positive numbers, the one whose digits sum to a larger number is better than the other. If the sum of digits is equal for both numbers, then she thinks the smaller number is better. For example, Kristen thinks that is better than and that is better than .

#Given an integer, , can you find the divisor of that Kristin will consider to be the best?

#Input Format

#A single integer denoting .

#Constraints

#Output Format

#Print an integer denoting the best divisor of .

#Sample Input 0

#12

#Sample Output 0

#6

#Explanation 0

#The set of divisors of can be expressed as . The divisor whose digits sum to the largest number is (which, having only one digit, sums to itself). Thus, we print as our answer.

#!/bin/python3

import math

import os

import random

import re

import sys

def divisor(n):

div = [i for i in range(1,n+1) if n%i==0]

return max(div,key=lambda x: sum(int(i) for i in str(x)))

if \_\_name\_\_ == '\_\_main\_\_':

n = int(input().strip())

print(divisor(n))

Connecting town.pg

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

#Example

#The array has elements that all start at . In the first operation, add to the first elements. Now the array is . In the second operation, add to the last elements (3 - 5). Now the array is 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, and , separated by a single space.

# lines follow. Each of them contains three integers, , , and , separated by spaces.

#Constraints

#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

#!/bin/python3

import math

import os

import random

import re

import sys

#

# 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

#

def solve(n, operations):

output = 0

for o in operations:

output += o[2] \* (o[1]-o[0]+1)

return output//n

if \_\_name\_\_ == '\_\_main\_\_':

fptr = open(os.environ['OUTPUT\_PATH'], 'w')

first\_multiple\_input = input().rstrip().split()

n = int(first\_multiple\_input[0])

m = int(first\_multiple\_input[1])

operations = []

for \_ in range(m):

operations.append(list(map(int, input().rstrip().split())))

result = solve(n, operations)

fptr.write(str(result) + '\n')

fptr.close()

Best divisor.py

Filling jars.py