Lëtzebuerger Informatiksolympiad 2022

Semi-Finals

Task Descriptions

Instructions

- The allowed programming languages are Python 3, Java and C/C++.
- All the programs must be realized in the form of a console application. For instructions how to realize a console application in the allowed programming languages, please refer to the remarks on the site *www.infosolympiad.lu* under the heading *Les questionnaires.*
- Under the input of the program is meant either the direct entry of data from the keyboard or the redirection from a text file in console mode. Under output of the program is meant either the direct display of data to the screen or the redirection to a text file in console mode.
- The formats of the input and output data shown in the execution examples must absolutely be respected.
- For testing, submitting and evaluating a program, the source file with a file extension "PY", "java" or "c/cpp") must be uploaded to the automated online judge CMS (Contest Management System), accessible via the homepage *www.infosolympiad.lu* or directly via the URL *http://158.64.46.20*. Please use your personal login (username & password) to access your account on the CMS. The filename of the single source file should be the same than the task name. Please refer to the CMS for technical details on how to test and submit a program.
- Please refer to the CMS for technical details like time limits and memory limits as well as compilation commands.
- You have the right to ask questions via the CMS, but the answers will not teach you how to use a programming language nor tell you how to solve the tasks by using a specific algorithm. The questions should be in relation with the CMS or should treat clarification issues concerning the task descriptions.

Any participant found to be in a situation of fraud during the semi-finals will be excluded from the competition. Is considered as fraud any use of programs or portions of programs which would represent a plagiarism. However, documentation of the use and implementation of programming language instructions is permitted.

Task 1 — parentheses — 30 points

Description

You were asked to code an important part of a new programming language, namely the check if parentheses `()`, brackets `[]`, or braces `{}` are correctly paired. But since it is a programming language, there are letters, digits and other symbols and most importantly the codes can contain all these symbols in strings denoted by `' '` and therefore should not be counted in the pairing check. You should also check if the strings are closed.

Task

Implement an algorithm that checks if the `()[]{}` symbols are paired correctly, with the correct handling of strings. You should also check that at the end you have no opened string.

Example(s)

- `(abc 123)` is valid.
- `({)}` is not valid since the closed parenthesis is before the closed brace.
- `('['a)` is valid since the `[` is in a string.
- `([]')` is not valid since there is a missing `)` and the string is not closed.
- `x*((7+5)*9-int('2'))` is valid

Constraints

Every character will be a printable character of the ASCII table.

Every line will have a maximal length of 150.

There will be a maximum of 100 lines to check.

Input and output of the program

Input data

The first line contains an integer describing the number of lines to check.

The following lines contain the different test cases

Output data

For each test case, print a line containing `OK` if the test case is valid or `NOK` if it is not.

Execution example(s)

Input file	Result
5 (abc 123) ({)} ('['a) ([]') x*((7+5)*9-int('2'))	OK NOK OK NOK OK

Distribution of points

Subtask	Points	Description
1	10	No strings and only parentheses to check
2	10	No strings
3	10	No additional constraints

Task 2 — pond — 40 points

Description

Your friend Lio is currently setting up a pond in her backyard, and is asking for your help. So far she dug out a rectangular area of **n** x **m** centimetres, but she did not do so in a uniform manner. In fact, to give her pond a more natural look, she dug the pond according to a depth profile. More precisely, she subdivided the pond into 1 by 1 centimetre squares, so that the pond can be thought of as an **n** by **m** grid. The 1 cm^2 patch at position (**i**, **j**) is at depth of $\mathbf{d_{ij}}$ centimetres below the rest of the yard. Here $1 \leq \mathbf{i} \leq \mathbf{n}$ and $1 \leq \mathbf{j} \leq \mathbf{m}$. She is now in the process of filling up the pond and pours water into the pond at position (1, 1). As Lio pours, water will spill over onto neighbouring squares, and then to their neighbours and so forth. We say that the pond is filled up to depth **L** if the square at (1, 1) has exactly **L** units of water above it and the water on (1, 1) does not spill over onto other squares anymore.

Lio noticed a problem while filling up the pond, namely that the fish may not be able to swim very far if the water is not deep enough! The fish that Lio wants in her pond can swim in water that is 1 cm deep, but not in water that is shallower. To be precise, she wants her fish to be able to travel between the squares at (1, 1) and (**n**, **m**).

Task

Implement an algorithm that outputs the smallest depth she can fill her pond up to such that the fish can swim between the squares at (1, 1) and (**n**, **m**).

Example

Suppose we have the following depth profile for a 3 x 4 pond:

10	8	2	1
8	8	8	5
7	7	5	10

Notice that if Lio fills her pond to depth 5 there will be water in the top left corner at position (1, 1), but no water in the bottom right corner at position (3, 4). Thus, there is no way for the fish to cross between the two corners, since the squares (2, 4), (3, 3) and (3, 4) are all dry. By increasing the water's depth by one centimetre to 6 centimetres, it becomes possible for the fish to travel to every area that has water in it. This is also the minimum depth, so 6 is the answer.

Constraints

$1 \leq \mathbf{n}, \mathbf{m} \leq 200$

$1 \leq \mathbf{d_{ij}} \leq 100\,000$

Input and output of the program

Input data

The first line contains the integers **n** and **m**, separated by a space.

The following **n** lines contain **m** integers each, separated by spaces. The **j**-th integer on the **i**-th line is $\mathbf{d_{ij}}$.

Output data

A single integer, the smallest depth which satisfies Lio's conditions.

Execution example

Input file	Result
3 4 10 8 2 1 8 8 8 5 7 7 5 10	6

Distribution of points

Subtask	Points	Description
1	10	**n** = 1
2	20	1 ≤ **n**, **m**, $\mathbf{d_{ij}}$ ≤ 1 000
3	10	No additional constraints

Task 3 vrperformance 30 points

Description

The new VR Performance (VRP) attracts so many spectators that it is organized in a room with **N** seats numbered from 1 to **N**. When the VRP opens and closes, the room is empty.

Throughout the day, when a spectator arrives, we first check if a place is free.

If no place is free, the spectator must wait. If other spectators arrive while there are already some waiting, they must line up strictly corresponding to the order of arrival. The first in line will occupy the next liberated place.

If only one place is free, the spectator must occupy that place.

If several places are free, the spectator must occupy the place with the smallest number.

The revenue per visit will be calculated according to the basic revenue of the assigned place and the weight of the spectator, according to which a multiplier is applied. The length of time the spectator stays in his place is not considered.

Write a program that, knowing the number of **M** spectators who visit the new VRP as well as the order of their arrivals and departures, calculates what will be the total revenue over a day.

Restrictions

The following restrictions should be considered:

- the number of places **N** : 1 ≤ **N** ≤ 100,
- the number of spectators **M** : 1 ≤ **M** ≤ 2000,
- the basic revenue applicable to a given place $\mathbf{R_i}$: 2 ≤ $\mathbf{R_i}$ ≤ 100,
- the multiplier relating to the weight of a given spectator $\mathbf{P_j}$: 2 ≤ $\mathbf{P_j}$ ≤ 10000.

Input of the program

Principle

The data is entered as follows:

- an integer denoting the number of seats in the room, with **N** > 0,
- a whole number indicating the number of spectators, with **M** > 0,
- **N** integers $\mathbf{R_i}$ designating each – for places numbered from 1 to **N** – the price applicable to the numbered place **i**, with 1 ≤ **i** ≤ **N**,
- **M** integers $\mathbf{P_j}$ each denoting the multiplier relative to the weight of the spectator numbered **j**, with 1 ≤ **j** ≤ **M**,
- 2 x **M** whole numbers each indicating the arrival or departure of a spectator **j**. A positive **j** denotes the arrival of the spectator **j**, a negative **j** (**j** preceded by the symbol `'-'`) designates the departure of the spectator **j**.

Example 1: Without a queue

	Explanations
`3`	3 places are available.
`4`	4 spectators wish to enter.
`2`	The basic revenue for place number 1 is 2.
`3`	The basic revenue for place number 2 is 3.
`5`	The basic revenue for place number 3 is 5.
`200`	The multiplier for the weight of the spectator number 1 is 200.
`100`	The multiplier for the weight of the spectator number 2 is 100.
`300`	The multiplier for the weight of the spectator number 3 is 300.
`800`	The multiplier for the weight of the spectator number 4 is 800.
`3`	Spectator 3 arrives and takes place 1. (Revenue: 2 x 300 = 600.)
`2`	Spectator 2 arrives and takes place 2. (Revenue: 3 x 100 = 300.)
`-3`	Spectator 3 leaves and frees place 1.
`1`	Spectator 1 arrives and takes place 1. (Revenue: 2 x 200 = 400.)
`4`	Spectator 4 arrives and takes place 3. (Revenue: 5 x 800 = 4000.)
`-4`	Spectator 4 leaves and frees place 3.
`-2`	Spectator 2 leaves and frees place 2.
`-1`	Spectator 1 leaves and frees place 1.

Example 2: With queue

	Explanations
`2`	2 places are available.
`4`	4 spectators who wish to enter.
`5`	The basic revenue for place number 1 is 5.
`2`	The basic revenue for place number 2 is 2.
`100`	The multiplier for the weight of the spectator number 1 is 100.
`500`	The multiplier for the weight of the spectator number 2 is 500.
`1000`	The multiplier for the weight of the spectator number 3 is 1000.
`2000`	The multiplier for the weight of the spectator number 4 is 2000.
`3`	Spectator 3 arrives and takes place 1. (Revenue: 5 x 1000 = 5000.)
`1`	Spectator 1 arrives and takes place 2. (Revenue: 2 x 100 = 200.)
`2`	Spectator 2 arrives and has to wait in first position.
`4`	Spectator 4 arrives and has to wait in second place.
`-1`	Spectator 1 leaves and frees place 2. Spectator 2 takes place 2. (Revenue: 2 x 500 = 1000.)
`-3`	Spectator 3 leaves and frees place 1. Spectator 4 takes place 1. (Revenue: 5 x 2000 = 10000.)
`-2`	Spectator 2 leaves and frees place 2.
`-4`	Spectator 4 leaves and frees place 1.

Output of the program

The program returns the total revenue for the day.

Example 1 (corresponding to the data in Example 1 above)

```
5300
```

Example 2 (corresponding to the data in Example 2 above)

```
16200
```

Distribution of points

Subtask	Points	Description
1	15	Input data is always without queue.
2	15	Input data is always with a queue.