Task AB03. MINIMUM POINT

⏳0.1 sec. 💾 1024 MB

Our somewhat forgotten heroine **D**eni got herself into a new adventure. She got lost in a labyrinth shaped like a grid, filled not with just anything – but with numbers. The grid has dimensions $N \times M$, and the numbers in it are positive integers. **D**eni is a smart girl, so she has figured out that the exits of the grid are located at the so-called *local minimum* points. A *local minimum* point of the grid is a cell whose value is strictly smaller than the values of its neighbors. Two cells are neighbors if they share a common side.

Deni can explore the value of arbitrary cells in the grid and wants to find an exit from the grid using as few explorations as possible. Write a program **`locmin`** that finds a *local minimum* point of an unknown grid.

Implementation details

You should implement the function `find_locmin_point`:

```
std::pair<int, int> find_locmin_point(int N, int M)
```

- N, M: the number of rows and the number of columns of the grid.

This function is called once for each test case and has to return an ordered pair containing the row number and the column number of a *local minimum* point. Rows are numbered consecutively from 1 to N, and columns – consecutively from 1 to M. In order to do this, your program can call the jury function `value`:

```
long long value(int r, int c)
```

- r, c: the row and the column of the cell you are exploring.

The function will return the value of the explored cell. If the parameters do not specify a valid cell in the grid, you will receive `Wrong answer` for the corresponding test. The complexity of this function is constant. You may call this function at most $N \times M$ times; otherwise you will also receive `Wrong answer`.

Constraints

- $1 \leq N, M \leq 500$.
- The values in the grid are integers in the interval $[1; 10^{18}]$.
- It is guaranteed that there exists at least one *local minimum* point in the grid.
- **There are no neighboring cells in the grid with equal values.**

Subtasks

Subtask	Points	N	Other constraints
0	0	—	The grid from the sample interaction.
1	21	$= 1$	-
2	79	≤ 500	There are tests for which the jury's program is **adaptive** – in them there is no fixed grid in advance; instead it is constructed dynamically depending on the performed explorations.

*The points for a given subtask are obtained only if all the tests for it are **successfully** passed, and are equal to the minimum score of a test in it multiplied by the points of the subtask.*

Scoring

Each test receives a score, which is a real number between 0 and 1 inclusive. If a test has a positive score, it is considered **successfully** passed for your solution. A test has a positive score if you correctly find a *local minimum* point in the grid within the memory and time limits. Let cnt denote the number of calls to the function `value` for a given test; then the score of the test is calculated as follows:

- Subtask 1.
  - If $cnt < 30$, then the score is 1.
  - If $30 \leq cnt < 300$, then the score is $\min(\frac{30}{cnt}, 1)$.
  - If $cnt \geq 300$, then the score is 0.1.
- Subtask 2.
  - If $cnt < 2026$, then the score is 1.
  - If $2026 \leq cnt < 5000$, then the score is $\min((\frac{2026}{cnt})^{\max(\frac{4000}{cnt}, 1)}, 1)$.
  - If $5000 \leq cnt < 10000$, then the score is $\min((\frac{2026}{cnt})^{1.2}, 0.4)$.
  - If $cnt \geq 10000$, then the score is 0.1.

Sample interaction

Let us have the following grid with 3 rows and 4 columns:

5	3	5	6
10	6	4	10
2	9	8	2

Contestant action	Jury's action
	`find_locmin_point(3,4)`
`value(1, 2)`	`return 3`
`value(1, 1)`	`return 5`
`value(1, 3)`	`return 5`
`value(2, 2)`	`return 6`
`return {1, 2}`	

Sample grader

Input format:

- line 1: two positive integers – the number of rows N and the number of columns M;
- line 2-$(N+1)$: M positive integers specifying the values in the grid in order from the first row and first column to the last row and last column.

Output format:

- line 1: two integers – the returned ordered pair of the call.