Brackets

Let's define a *correct string of brackets* as follows:

- () and [] are correct strings of brackets;
- if A is a correct string of brackets, then (A) and [A] are also correct strings of brackets;
- if A and B are both correct strings of brackets, then the concatenation AB is also a correct string of brackets;

In a correct string of brackets which contains at least one pair of square brackets: [and corresponding], each square bracket (both opening and closing) was replaced by the **opening** round bracket, therefore obtaining a *broken string of brackets*.

For example, `((` and `((((()))` both are broken strings of brackets. First string is obtained from the correct strings of brackets `[]`. Second string may be obtained only from the following four correct strings of brackets: `[]((()))`, `([](()))`, `(([]()))` or `((([])))`.

Your task is for a given broken string of brackets calculate the number of possible correct strings of brackets from which the given broken string may have been obtained.

Input data

The first line of text file **brackets.in** contains a single even integer N ($2 \le N \le 30000$) - the length of the given broken string of brackets. The second line contains N characters '(' and ')' - the given broken string of brackets.

Output data

The single line of the text file **brackets.out** should contain one integer - the required number of correct strings of brackets. Because the number of correct strings of brackets can be large, you should output the answer **modulo 1 000 000 009**.

Examples

Input data (file `brackets.in`)	Output data (file `brackets.out`)	Corresponding correct strings of brackets
4 **((()**	**2**	[](), ([])

Input data (file `brackets.in`)	Output data (file `brackets.out`)	Corresponding correct strings of brackets
8 **((((((((**	**14**	[][][][], [[]][][], [[]][[]], [][][[]], [[[]]][], [[][]][], [][[][]], [][[[]]], [[[[]]]], [[][[]]], [[[]][]], [[][][]], [[[][]]], [][[]][]

Grading

Test cases where $N \le 50$ are worth 20 points.
Test cases where $N \le 1000$ are worth 45 points.

Mobile

The well-known mobile network operator Totalphone has set up a number of new base transceiver stations in order to cover a newly-built highway with its network. As always the programmers of Totalphone have been sloppy; as a result, the transmission power cannot be set up individually for the stations, but one can only set the transmission power to a fixed common value for all the stations. In order to minimize power consumption, the company wants to know the maximal distance of a point on the highway to the nearest base transceiver station.

Input data

The first line of text file **mobile.in** consists of two integers N($1 \le N \le 10^6$) and L($1 \le L \le 10^9$) representing the number of base transceiver stations and the length of the highway, respectively. N lines follow, each containing a pair of integers x_i, y_i ($-10^9 \le x_i, y_i \le 10^9$) which describes the coordinates of a base transceiver station. All points are distinct. Coordinates are sorted in the non-decreasing order with respect to x_i coordinates. If two values of x_i are the same, then coordinates are sorted with respect to y_i coordinates in increasing order.

The highway is a straight line ranging from (0; 0) to (L; 0).

Output data

The first and only line of the text file **mobile.out** should contain a single number - the maximal distance of a point on the highway to the nearest base transceiver station. Your output will be regarded as correct if it differs by at most 10^{-3} from the precise result.

Example

Input data (file `mobile.in`)	Output data (file `mobile.out`)

`2 10`
`0 0`
`11 1`
	`5.545455`

Grading

Test cases where N≤5000 are worth 25 points.

Test cases where N≤100000 are worth 50 points.

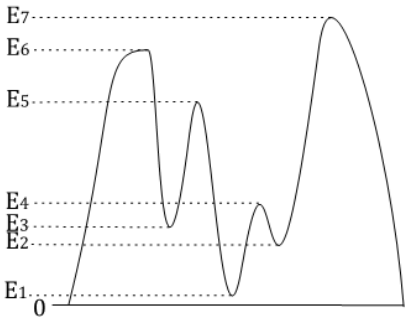
Warning

Use at least double precision floating point numbers for your computations, as smaller types may fail to give the precision required for solving the problem.

Peaks

An alpinist who lives on a mountainous island has climbed to some peak and now wants to reach a higher peak.

To be more precise, every point on the island has a positive *elevation* above sea level (the elevation of the sea is 0) and if the peak the alpinist is currently on has elevation E_i, then his aim is to reach some peak with elevation E_j(E_j>E_i). Because he is on a peak there is no immediate path uphill – to get to a higher point the alpinist first needs to go downhill to some lower level and only then he can go uphill again. The way down is never as remarkable as the way up, thus, the alpinist wants to maximize the elevation of the lowest point on the path from the current location to the higher peak.



For example, if the profile of the island is as shown in the figure and the alpinist is at the peak with elevation E_4, then there are three peaks with higher elevation (E_5, E_6 and E_7), but the path with the lowest point having the highest elevation is the path to the peak with elevation E_7 – on this path he never goes below level E_2 (in the other cases he will be forced to go down to level E_1). If he started from E_5, the corresponding lowest level would be E_3 (path to E_6), but if he started from E_6 it would be E_1.

The map of the island is a two-dimensional rectangular table containing N×M squares and it describes the elevation of particular parts of the island – the number in a cell describes the elevation of the corresponding region of the island. Two cells are adjacent if they share a common point. Thus, each cell (except those on the border) is adjacent to eight other. A path is a sequence of cells where each two consecutive cells are adjacent. A *flat area* is a set of one or more cells having the same elevation, any pair of them being connected by a path only visiting cells within the set. Any two adjacent cells with equal elevation belong to the same flat area. A *peak* is a flat area whose cells don't have any adjacent cells with higher elevation.

Write a program which finds all peaks on the island and for each of them finds the elevation of the highest possible lowest point on a path to some peak with a higher elevation. For the highest peak on the island (for which there is no higher peak on this island) we assume that the alpinist will leave the island looking for higher peaks, thus, the lowest point will be 0 (the level of the sea).

Input data

The first line of the text file **peaks.in** contains two positive integers N and M ($1 \le N,M \le 2000$, $N \times M \le 10^5$), the height and the width of the map, respectively. The next N lines contain the description of the map of the island. Each of these lines contains M integers E_{ij}($1 \le E_{ij} \le 10^6$) separated by spaces. The elevation of the cell E_{ij} (corresponding to i-th row and j-th column on the map) is given as the j-th number in the i+1-st line of the file.

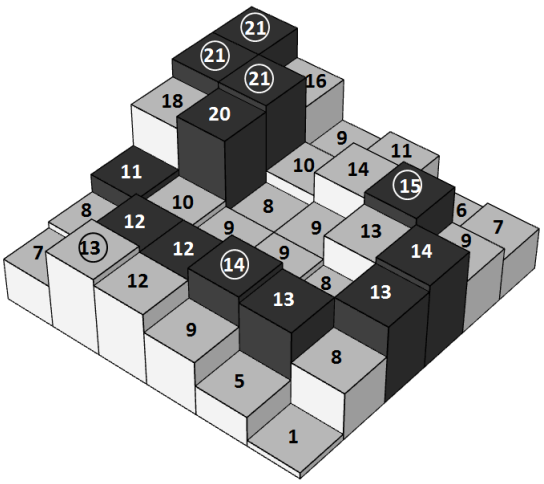
Output data

The first line of the text file **peaks.out** must contain one integer P, the number of peaks found on the island. The next P lines must each contain two integers: the elevation of the particular peak and the elevation of the highest possible lowest point on the path to some higher peak. The information about peaks should be written in descending order of their

elevation; if several peaks have the same elevation then they should be sorted in descending order of the lowest point elevation.

Example 1

Input data (file `peaks.in`)	Output data (file `peaks.out`)	Comment:

`6 6`
`21 16 9 11 6 7`
`21 21 10 14 15 9`
`18 20 8 9 13 14`
`11 10 9 9 8 13`
`8 12 12 14 13 8`
`7 13 12 9 5 1`
	`4`
	`21 0`
	`15 11`
	`14 13`
	`13 12`
		
		All peaks are marked by circles.
		One of the possible paths from peak with elevation 15 is shown with dark colouring.

Example 2

Input data (file `peaks.in`)	Output data (file `peaks.out`)

`5 3`
`16 14 16`
`14 14 15`
`12 17 16`
`12 13 10`
`16 11 16`
	`5`
	`17 0`
	`16 15`
	`16 14`
	`16 13`
	`16 13`

Grading

Test cases where N≤2 or M≤2 are worth 15 points.
Test cases where P≤500 are worth 50 points.
Test cases where P≤5000 are worth 80 points.