Camouflaged camp (Lithuania)

TASK

Squad commander is looking for a location to build a camouflaged camp of a certain size. He has a digital topographic map of the area, which is a rectangular grid with every element defining the altitude at a certain coordinate. Coordinates of any point on the map are identified by the row and the column in the table.

2	2	2	2	2	2
2	6	6	4	3	2
3	5	8	7	7	4
4	6	8	9	8	6
5	7	8	8	8	7

Digital topographic 5x6 map
Camp of size 3×5 at position (3,2)

The camp location should be a rectangle fully located within the map area and satisfy certain characteristics. Each characteristic consists of two identical adjacent rectangular areas and the requirement for their altitudes. It is defined by

- *location,* i.e. the coordinates of the top-left corner of the first (i.e. the left or the top) rectangular area; The coordinates are given in respect of the camp area.
- *size* (length and width) of the first (and the second because they are equal) rectangular areas;
- *rectangle arrangement flag*, 0 indicates horizontal arrangement of the rectangular areas (i.e. areas have common vertical side), while 1 – vertical arrangement (i.e. areas have common horizontal side);
- *altitude flag*, 0 indicates that the average altitude of the first (i.e. the left or the top) rectangular area should be strictly less (<) than the average altitude of the second. 1 indicates the opposite (≥) situation.

2	2	2	2	2	2
2	6	6	4	3	2
3	5	8	7	7	4
4	**6**	**8**	**9**	8	6
5	7	8	8	8	7

Characteristic A:
Location – (1, 1)
Size – (1, 3)
Arrangement flag: 1
Altitude flag: 0

The selected location satisfies characteristic A.

2	2	2	2	2	2
2	6	6	4	3	2
3	5	8	7	7	4
4	6	8	9	**8**	**6**
5	7	8	8	**8**	**7**

Characteristic B:
Location – (2, 2)
Size – (2, 2)
Arrangement flag: 0
Altitude flag: 0

The selected location does not satisfy characteristic B.

The camp location satisfies the characteristic if the altitude requirement is satisfied.

Write a program which given the topographic map of the area and the characteristics would find the best (the one that satisfies most characteristics) location for building the camouflaged camp. In case of several solutions, output any of them.

INPUT

The input file name is `CAMP.IN` The first line contains two integers R and C ($2 \le R,C \le 1000$). They correspond to the number of *rows* and *columns* in the topographic map. The following R lines with C nonnegative integer numbers in each of them describe the topographic map. The altitude does not exceed 255 at any coordinate.

Two integer numbers L (number of rows) and W (number of columns) ($1 \le L,W \le 1000$; $L \le R$; $W \le C$) defining the size of the camp are written in the following line.

The next line contains one integer H ($1 \le H \le 1000$) – the *number* of characteristics.

Finally, the following H lines describe characteristics. Each of them contains 6 integers: coordinates of the characteristic's top-left corner, size of the first rectangular area, the arrangement and the altitude flags. All characteristics fit within the camp.

OUTPUT

The first line of the output file `CAMP.OUT` should contain two integers – the coordinates of top-left corner of camp location.

EXAMPLES

INPUT	OUTPUT	COMMENT
5 6 2 2 2 2 2 2 2 6 6 4 3 2 3 5 8 7 7 4 4 6 8 9 8 6 5 7 8 8 8 7 3 5 3 1 1 1 3 1 0 2 2 2 2 0 0 2 4 1 1 1 1	3 1	Location (3, 1) satisfies all three characteristics

Magic Parenthesis (Finland)

TASK

In the LISP programming language ***everything*** is written inside balanced parentheses `(LIKE THIS)`. This means that LISP code sometimes contains long stretches "`)))...)`" of closing parentheses. It is very tedious for the LISP programmer to get the number of these closing parentheses '`)`' to correspond exactly to the number of opening parentheses '`(`'.

To prevent such syntax errors, some LISP dialects introduce a ***magic closing parenthesis*** '`]`' which substitutes *one or more closing parentheses '`)`' as needed to properly balance the opening parentheses '`(`'*. But then the LISP compiler must calculate how many closing parentheses '`)`' each magic parenthesis '`]`' really substitutes. How?

Write a program which is given a string of opening, closing, and magic parentheses, and which calculates for each occurrence of the magic parenthesis the number of opening parentheses it corresponds to. In case of multiple solutions, the program should find any one of them.

INPUT

The input file name is `LISP.IN`. The first line consists of two integers $0 \le N \le 10\,000\,000$ and $0 \le M \le 5\,000\,000$ separated by a space. The first number N is the length of the input string. The second number M is the number of magic closing parentheses in the string. The rest of the input file starts on the second line and is a string of length N consisting of characters '`(`', '`)`' and '`]`'. The character '`]`' occurs exactly $M \le N$ times in this string. This string is divided into lines of at most 72 characters each for readability.

OUTPUT

The output file name is `LISP.OUT`. The first line consists of an integer '`0`' or '`1`'.

The number '`0`' means that the input cannot be balanced. (For example, the string which consists of just a single magic parenthesis "`]`" obviously cannot be balanced.) In this case there is no more output.

The number '`1`' means that the input can be balanced. In this the output continues with the following M extra lines.

The 1st of these extra lines consists of the number $C_1 \ge 1$ of closing parentheses '`)`' the 1st magic parenthesis '`]`' in the input substitutes to. The 2nd extra line consists of the corresponding number $C_2 \ge 1$ for the 2nd '`]`' in the input, and so on.

If there are many ways in which the given string can be balanced, then your program may output any one of them.

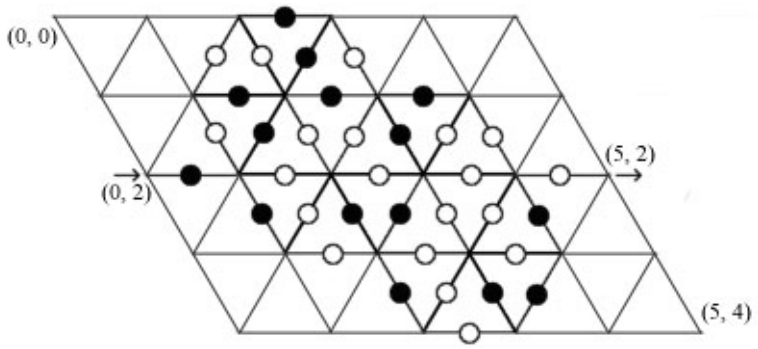
EXAMPLE

The input on the left describes a string with 8 characters, of which 2 are magic. The output on the right shows one way of balancing this input: the first magic parenthesis corresponds to 3 opening parentheses, and the second magic parenthesis corresponds to 1 opening parenthesis. And indeed, the magic-free string ((((()))) is properly balanced, where the closing parentheses corresponding to the magic parentheses are underlined.

INPUT	**OUTPUT**
8 2 (((((])]	1 3 1

Maze (Lithuania)

Consider the following maze made of equilateral triangles:



Each vertex is described by two coordinates *x* and *y* as in the picture. Some of the edges have a white or a black circle on them. There are two major rules that control movement within the maze:

- it is only allowed to pass edges with circles on them.
- while walking through the maze it is obligatory to alternate white and black circles; i.e. it is only allowed to pass an edge with white circle if the last circle passed was black and vice versa. It is allowed to pass an edge with either black or white circle on it during the first move.

TASK

Write a program to find the length of the shortest path from the entrance point to the exit in the maze. The length of the path is defined as the number of edges (or circles) passed. You may assume that such path always exists.

INPUT

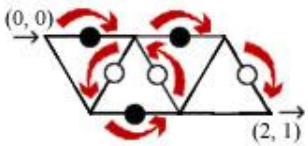
The input file name is `MAZE.IN` The first line contains two integers W and H which are the *width* and the *height* of the maze respectively ($1 \le W, H \le 500$). The second line consists of four integer values: $X_1\ Y_1\ X_2\ Y_2$ ($0 \le X_1, X_2 \le W$; $0 \le Y_1, Y_2 \le H$). (X_1, Y_1) are the coordinates of the entry point in the maze and (X_2, Y_2) are the exit coordinates.

The next $2H+1$ lines provide the description of the edges: odd lines (3rd, 5th, etc) describe horizontal edges, and even lines (4th, 6th, etc) describe non-horizontal ones. Each line consists of a string of characters **`n`**, **`w`** and **`b`**, where **`n`** means, that there is no circle on the edge, and **`w`** or **`b`** means that there is white or black circle on the edge respectively. There are no spaces between these characters. Naturally, odd lines consist of exactly W characters, and even lines consist of exactly $2W+1$ characters.

OUTPUT

Your program should output a single integer (the length of the shortest path from entrance point to the exit in the maze) in the first (and the only) line of the file `MAZE.OUT`.

EXAMPLES

INPUT	OUTPUT	COMMENTS
2 1 0 0 2 1 bb nwwnw bn	6	A simple maze. One possible shortest path is this: (0, 0) → (1, 0) → (0, 1) → (1, 1) → (1, 0) → (2, 0) → (2, 1) Here is the illustration of the maze and the shortest path: 

INPUT	OUTPUT	COMMENTS
5 4 0 2 5 2 nnbnn nnnwwbwnnnn nbbbn nnwbwwbwwnn bwwww nnbwbbwwbnn nwwwn nnnnbwbbnnn nnwnn	22	This is the description of the maze given in the picture on the first page. The shortest path is this: (0, 2) → (1, 2) → (1, 1) → (2, 1) → (2, 0) → (3, 0) → (3, 1) → (3, 2) → (4, 1) → (3, 1) → (3, 0) → (2, 0) → (2, 1) → (1, 1) → (1, 2) → (1, 3) → (2, 3) → (2, 4) → (3, 4) → (3, 3) → (4, 3) → (4, 2) → (5, 2) (Length: 22)