TASK OVERVIEW SHEET / DAY-1

TASK		FROG	UTOPIA	XOR
Task material directory	Linux	`~/frog`	`~/utopia`	`~/xor`
	WinXP	`C:\IOI\frog`	`C:\IOI\utopia`	`C:\IOI\xor`
Time limit per test		2 secs	2 secs	-
Memory limit		64 MB	32 MB	-
Compiler options	C and C++	`-O2 -static -lm`	`-O2 -static -lm`	-
	Pascal	`-So -O2 -XS`	`-So -O2 -XS`	-
Number of tests		25	25	10
Maximum points per test		4	4	10
Maximum total points		100	100	100

Program header comment when using C		`/*`
		`TASK: frog`
		`LANG: C`
		`*/`
			`/*`
			`TASK: utopia`
			`LANG: C`
			`*/`
				-

Program header comment when using C++		`/*`
		`TASK: frog`
		`LANG: C++`
		`*/`
			`/*`
			`TASK: utopia`
			`LANG: C++`
			`*/`
				-

Program header comment when using Pascal		`{`
		`TASK: frog`
		`LANG: PASCAL`
		`}`
			`{`
			`TASK: utopia`
			`LANG: PASCAL`
			`}`
				-

Submission is accepted, if;		Example 1 is solved.	Example 1 is solved.	The file format is correct.

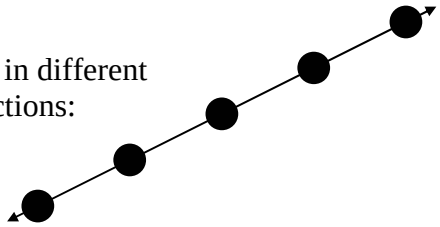
The Troublesome Frog

PROBLEM

In Korea, the naughtiness of the *cheonggaeguri*, a small frog, is legendary. This is a well-deserved reputation, because the frogs jump through your rice paddy at night, flattening rice plants. In the morning, after noting which plants have been flattened, you want to identify the path of the frog which did the most damage. A frog always jumps through the paddy in a straight line, with every hop the same length:

Different frogs can jump with different hop lengths:

And in different directions:



Your rice paddy has plants arranged on the intersection points of a grid as shown in Figure-1, and the troublesome frogs hop completely through your paddy, starting outside the paddy on one side and ending outside the paddy on the other side as shown in Figure-2:

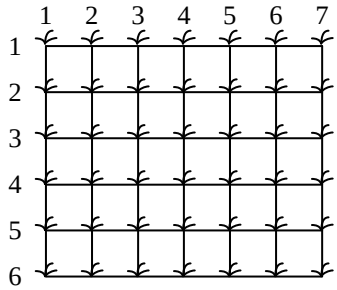


Figure-1

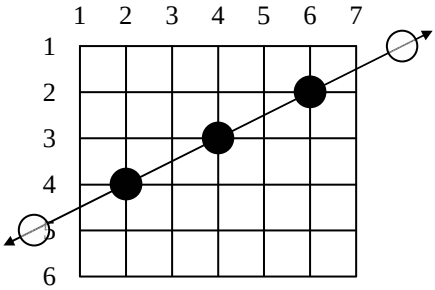


Figure-2

Many frogs can jump through the paddy, hopping from rice plant to rice plant. Every hop lands on a plant and flattens it, as in Figure-3. Note that some plants may be landed on by more than one frog during the night. Of course, you can not see the lines showing the paths of the frogs or any of their hops outside of your paddy – for the situation in Figure-3, what you can see is shown in Figure-4:

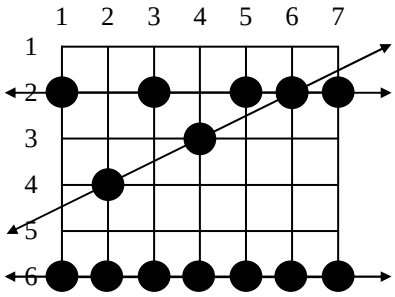


Figure-3

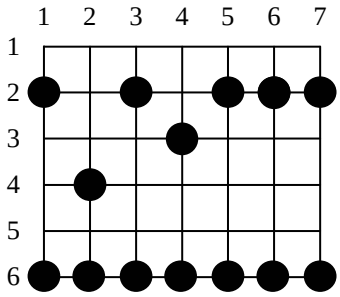


Figure-4

From Figure-4, you can reconstruct all the possible paths which the frogs may have followed across your paddy. You are only interested in frogs which have landed on at least 3 of your rice plants in their

voyage through the paddy. Such a path is said to be a frog path. In this case, that means that the three paths shown in Figure-3 are frog paths (there are also other possible frog paths). The vertical path down column 1 might have been a frog path with hop length 4 except there are only 2 plants flattened so we are not interested; and the diagonal path including the plants on row 2 col. 3, row 3 col. 4, and row 6 col. 7 has three flat plants but there is no regular hop length which could have spaced the hops in this way while still landing on at least 3 plants, and hence it is not a frog path. Note also that along the line a frog path follows there may be additional flattened plants which do not need to be landed on by that path (see the plant at (2, 6) on the horizontal path across row 2 in Figure-4), and in fact some flattened plants may not be explained by any frog path at all.

Your task is to write a program to determine the maximum number of landings in any single frog path (where the maximum is taken over all possible frog paths). In Figure-4 the answer is 7, obtained from the frog path across row 6.

INPUT

Your program is to read from standard input. The first line contains two integers *R* and *C*, respectively the number of rows and columns in your rice paddy, $1 \le R,C \le 5000$. The second line contains the single integer *N*, the number of flattened rice plants, $3 \le N \le 5000$. Each of the remaining *N* lines contains two integers, the row number ($1 \le$ *row number* $\le R$) and the column number ($1 \le$ *column number* $\le C$) of a flattened rice plant, separated by one blank. Each flattened plant is only listed once.

OUTPUT

Your program is to write to standard output. The output contains one line with a single integer, the number of plants flattened along a frog path which did the most damage if there exists at least one frog path, otherwise, 0.

EXAMPLE INPUTS AND OUTPUTS

Example 1: input output

(the example of Figure-4)

```
6 7
14
2 1
6 6
4 2
2 5
2 6
2 7
3 4
6 1
6 2
2 3
6 3
6 4
6 5
6 7
```

```
7
```

Example 2: input (the example of Figure-5)

```
6 7
18
1 1
6 2
3 5
1 5
4 7
1 2
1 4
1 6
1 7
2 1
2 3
2 6
4 2
4 4
4 5
5 4
5 5
6 6
```

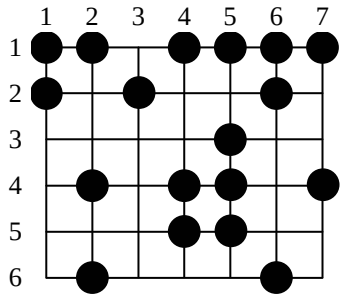


Figure-5

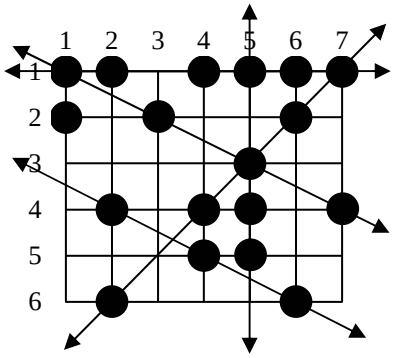


Figure-6: The maximum number of plants flattened by a frog is 4.

output

```
4
```

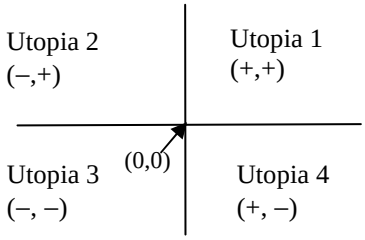
SCORING

If your program outputs the correct answer for a test case within the time limit, then you get full points for the test case, and otherwise you get 0 points.

Utopia Divided

PROBLEM

The beautiful land of Utopia was once ravaged by war. When the hostilities subsided the country was divided into four regions by a longitude (north-south line) and a latitude (east-west line). The intersection of these lines became known as the point (0,0). All four parts claimed the name Utopia, but as time went by they generally became known as Utopia 1 (northeast), 2 (northwest), 3 (southwest) and 4 (southeast). A point in any of the regions was identified by its distance east and its distance north of (0,0). These distances could be negative; hence a point in Utopia 2 was designated by a (negative, positive) pair, in Utopia 3 by a (negative, negative) pair, in Utopia 4 by (positive, negative) and in Utopia 1 by a pair of positive numbers.



A major problem was that citizens were not permitted to cross borders. Fortunately, some ingenious IOI contestants from Utopia developed a safe means of teleportation. The machine requires code numbers, each of which can only be used once. Now the challenge facing the team, and you, is to guide the teleporter from its initial position of (0,0) to the regions of Utopia in the order requested. You don't care where in a region you land, but you will have a sequence of N region numbers that specify the regions in which the teleporter is to land. You may be asked to land in the same region in two or more consecutive stops. After leaving the initial (0,0) point, you must never land on a border.

You will receive as input a sequence of $2N$ code numbers and are to write them as a sequence of N code pairs, placing a plus or a minus sign before each number. If you are currently at the point (x,y) and use the code pair $(+u,–v)$, you will be teleported to the point $(x+u, y–v)$. You have the $2N$ numbers, and you can use them in any order you like, each with a plus or a minus sign.

Suppose you have code numbers 7, 5, 6, 1, 3, 2, 4, 8 and are to guide the teleporter according to the sequence of region numbers 4, 1, 2 ,1. The sequence of code pairs (+7,–1), (–5,+2), (–4,+3), (+8,+6) achieves this as it teleports you from (0,0) to the locations (7,–1), (2,1), (–2,4) and (6,10) in that order. These points are located in Utopia 4, Utopia 1, Utopia 2, and Utopia 1, respectively.

TASK

You are given $2N$ distinct code numbers and a sequence of N region numbers indicating where the teleporter is to land. Construct a sequence of code pairs from the given numbers that guide the teleporter to go through the given region sequence.

INPUT

Your program is to read from standard input. The first line contains a positive integer N ($1 \le N \le 10000$). The second line contains the $2N$ distinct integer code numbers ($1 \le$ *code number* ≤ 100000) separated by single spaces. The last line contains a sequence of N region numbers, each of which is 1, 2, 3 or 4.

OUTPUT

Your program is to write to standard output. The output consists of N lines, each containing a pair of code numbers each preceded by a sign character. These are codes pairs that will direct the teleporter to the given region sequence. Note that there must be no blank following a sign, but there must be a single space after the first code number.

If there are several solutions your program can output any one of them. If there are no solutions your program should output the single integer 0.

EXAMPLE INPUTS AND OUTPUTS

Example 1: input

```
4
7 5 6 1 3 2 4 8
4 1 2 1
```

output

```
+7 -1
-5 +2
-4 +3
+8 +6
```

Example 2: input

```
4
2 5 4 1 7 8 6 3
4 2 2 1
```

output

```
+3 -2
-4 +5
-6 +1
+8 +7
```

SCORING

If your program outputs a correct answer for a test case within the time limit, then you get full points for that test, and otherwise you get 0 points for the test case.

XOR

PROBLEM

You are implementing an application for a mobile phone, which has a black-and-white screen. The *x*-coordinates of the screen start from the left and the *y*-coordinates from the top, as shown in the figures. For the application, you need various images, which are not all of the same size. Instead of storing the images, you want to create the images using the phone's graphics library. You may assume that at the start of drawing an image, all pixels of the screen are white. The only graphics operation in the phone's library is `XOR(L,R,T,B)`, which will reverse the pixel values in the rectangle with top left coordinate (`L`,`T`) and bottom right coordinate (`R`,`B`), where `L` stands for the left, `T` for the top, `R` for the right and `B` for the bottom coordinate. Note that in some other graphics libraries the order of the arguments is different.

As an example, consider the image in Figure-3. Applying `XOR(2,4,2,6)` to an all white image gives the image in Figure-1. Applying `XOR(3,6,4,7)` to the image of Figure-1 gives the image in Figure-2, and applying `XOR(1,3,3,5)` to the image in Figure-2 finally gives the image in Figure-3.

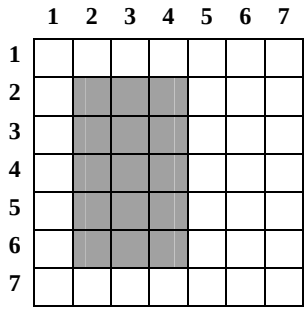


Figure-1

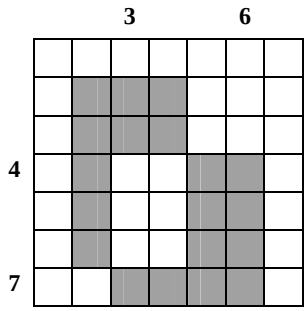


Figure-2

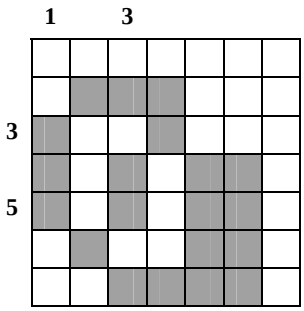


Figure-3

Given a set of black-and-white pictures, your task is to generate each picture from an initially white screen using as few `XOR` calls as you can. You are given the input files describing the images, and you are to submit files including the required `XOR` call parameters, not a program to create these files.

INPUT

You are given 10 problem instances in the text files named `xor1.in` to `xor10.in`. Each input file is organized as follows. The first line of an input file contains one integer N, $5 \le N \le 2000$, meaning that there are N rows and N columns in the image. The remaining lines represent the rows of the image from top to bottom. Each line contains N integers: the pixel values in the row from left to right. Each of these integers is either a 0 or a 1, where 0 represents a white pixel and 1 represents a black pixel.

OUTPUT

You are to submit 10 output files corresponding to the given input files.

The first line contains the text
`#FILE xor I`
where integer I is the number of the respective input file. The second line contains an integer *K*: the number of XOR calls specified in the file. The following *K* lines represent these calls from the first call to the last call to be executed. Each of these *K* lines contains four integers: the XOR call parameters L, R, T, B in that order.

EXAMPLE INPUT AND OUTPUT

`Example: xor0.in`

```
7
0 0 0 0 0 0 0
0 1 1 1 0 0 0
1 0 0 1 0 0 0
1 0 1 0 1 1 0
1 0 1 0 1 1 0
0 1 0 0 1 1 0
0 0 1 1 1 1 0
```

`xor0.out`

```
#FILE xor 0
3
2 4 2 6
3 6 4 7
1 3 3 5
```

SCORING

If
- the XOR calls specified in your output file do not generate the required image, or
- the number of XOR calls specified in your output file is not *K*, or
- in your output file, K > 40000, or
- your output file contains such an XOR call that L>R or T>B, or
- your output file contains an XOR call which does not have positive coordinates, or
- your output file contains an XOR call with a coordinate value exceeding *N*,

then your score is zero. Otherwise, your score is
1+9×CallsInBestAnswerOfAllContestants/CallsInYourAnswer.

The score is rounded to the first decimal place for each case. The total score is rounded off to the nearest integer.

Suppose that you submit a solution with 121 XOR calls. If that is the best submission of all contestants, your score is 10. If the best of the submitted solution of all contestants uses 98 XOR calls, your score becomes 1+9×98/121(=8.289…), which will be rounded to 8.3.