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.

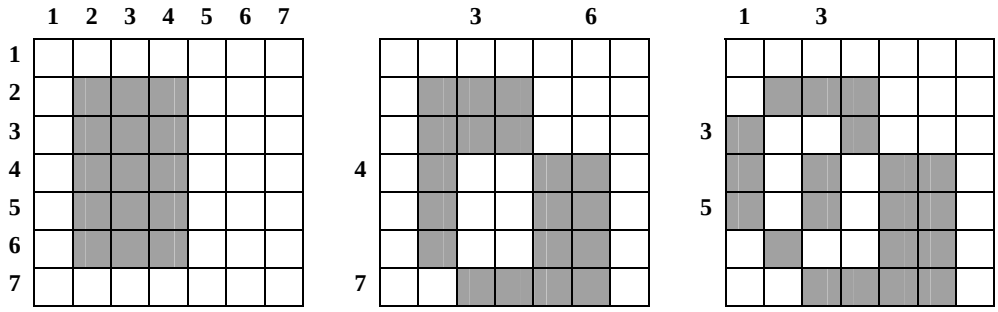

Figure-1 **Figure-2** **Figure-3**

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.