Bombs

Michael the fighter pilot and you are on a special mission to destroy an evil spy network that has been operating extensively in the world today. Their base is located in the distant land of Azaria.

The spies, known as "targets", are based in a rectangular grid of size R-by-C, with R rows and C columns. You need to hit the targets using the bombs you carry inside your aircraft. These bombs have different explosion range and thus will destroy an area corresponding to the range of each bomb.

Michael can choose to drop the bomb in any cell inside the grid.

All the bombs have an odd-size explosion range. If a bomb with an explosion range of N is dropped at cell (row x, col y), then targets in the cells that form a square with $\left(x-\frac{N-1}{2}, y-\frac{N-1}{2}\right)$ as the top-left corner and $\left(x+\frac{N-1}{2}, y+\frac{N-1}{2}\right)$ as the bottom-right corner will be destroyed. Note that the corners could be outside of the grid.

In one of his missions, Michael discovered the following grid on the left: ("1" denotes a target)

1				
	1	1	1	
	1		1	
			1	

1				
	1	1	1	
	1		1	
			1	

He dropped a size-3 bomb in the middle cell (2, 2) and destroyed 6 targets as shown in the matrix on the right. The darker cell indicates the cell where the bomb was deployed, and the lighter ones indicate the exploded area (note that we use 0-based indexing).

As some of the grids are so huge with a great number of targets, Michael cannot devise an efficient strategy to destroy the targets. Since you are an excellent programmer, Michael will ask you two types of queries, and you need to give a fast answer to Michael using your program. The queries are:

1. If Michael has a bomb of size N, which is guaranteed to be an odd-number, in which cell should Michael deploy it such that it destroys the most number of targets? There can be more than one query of this type. Each query is based on the original grid and not the result of the previous query.

2. You want to destroy the most number of targets but want to minimize the resources used. Therefore, what bomb size and where should it be deployed, such that we minimize the bomb size and kill the targets efficiently and effectively?

For the second query type, Michael has defined a point system to determine the efficiency and effectiveness of the explosion. For each target hit, 3 (three) points are received. However, for each empty cell hit, 1 (one) point is deducted. Therefore, the most effective and efficient answer is the one

yielding the most number of points. For example, the bomb deployed in the grid above will yield 15 points since it hit 6 targets and 3 empty cells. It is the most efficient and effective drop for the above grid. Explosions that occur on cells outside of the grid are ignored for points calculation.

Before starting, there are a few guidelines that the program needs to adhere to:

1. Bomb sizes must be an odd number, with the smallest size being 1 (one). There are no limitations on the maximum size of the bombs.
2. There are no empty grids. That is, any given grid is guaranteed to have at least 1 target in it.
3. The bombs can be deployed in any cell inside the grid.
4. For query type 1, if there are multiple locations where the bomb will destroy the most number of targets, pick the one with the smallest row number first, followed by the smallest column number if there is still a tie.
5. For query type 2, if there is a tie, we first want to minimize the bomb size, followed by the deployment position similar to the condition for query type 1 if there is still a tie.

This is an important mission. Make your country proud by implementing this important program that can help Michael accomplish his mission.

Input

The first line of input consists of two numbers, R and C (3 <= R, C <= 50), separated by a single space, denoting the number of rows and columns of the grid respectively. The next R lines will have C digits, separated by a single space, each denoting whether a cell contains a target. A target is denoted by '1' and '0' denotes an empty cell.

The next line of input is a single integer Q (Q <= 5), the number of type-1 queries. The next Q lines will contain a single integer N. N is the size of the bomb being asked as part of the first query type.

Output

For the first Q lines of output, print two numbers, separated by a single space, the row and column index of the deployment position which satisfy the first query type. Note that we use 0-based indexing.

The last line of the output consists of three numbers, separated by a single space, denoting the row-coordinate, column-coordinate, and the bomb size which satisfies the second query type for the given grid.

For each row, there is no whitespace after the last number. Your last line of output should contain a newline character.

Sample Input

```
5 5
1 0 0 0 0
0 1 1 1 0
0 1 0 1 0
0 0 0 1 0
0 0 0 0 0
1
3
```

Sample Output

```
2 2
2 2 3
```

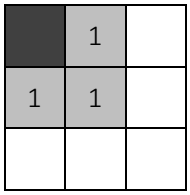
Explanation

The sample input is the map used in the problem description. There is only 1 type-one query being asked. The bomb with size 3 is best deployed in cell (2,2). For the second query, we also need to

deploy a size-3 bomb in the cell (2,2) to get the best solution. We can use a size-1 bomb in cell (0,0) which will hit 1 target and no empty cells, but it will only yield 3 points. We can also destroy the entire grid by dropping a size-5 bomb in the cell (2,2). However, the points yield is only 3 points because of the penalty.

Clarification

If we drop a size-3 bomb in cell (0,0) based on the map below, the explosion will look like this:



The drop will give 8 points since it destroys three targets and hit 1 empty cell. Explosions that occur on **cells outside of the grid are ignored** for points calculation.

Skeleton

You are given the skeleton file Bombs.java with the following contents. You should see a non-empty file when you open the skeleton file. Otherwise, you might be in the wrong working directory.

```
/*
 * Name           :
 * Matric No.     :
 * Plab Account   :
 */

import java.util.*;

public class Bombs {

    public static void main(String[] args) {

    }

}
```

Notes:

1. You should develop your program in the subdirectory ex1 and use the skeleton java file provided. You **should not create a new file or rename the file provided.**
2. You do not have to use OOP for this sit-in lab. However, you are reminded to pay attention to the **modularity** of your program.
3. You are free to define your own helper methods.
4. Please be reminded that the marking scheme is:

   Input : 10%
   Output : 10%
   Correctness : 50%
   Programming Style : 30% (awarded if you score **at least 20% from the above**):
   - Meaningful comments (pre- and post- conditions, comments inside the code): 10%
   - Modularity: 10%
   - Proper Indentation: 5%
   - Meaningful Identifiers (for both method and variable names): 5%

   Compilation Error : Deduction of **50% of the total marks obtained**.