AGRA - Árboles y Grafos, 2018-2
Tarea 2: Semanas 3 y 4

Para entregar el viernes 17 de agosto/domingo 19 de agosto de 2018

Problemas conceptuales a las 16:00 (17 de agosto) en el Departamento de Electrónica y Ciencias de la Computación

Problemas prácticos a las 23:59 (19 de agosto) en la arena de programación

---

Tanto los ejercicios como los problemas deben ser resueltos, pero únicamente las soluciones de los problemas deben ser entregadas. La intención de los ejercicios es entrenarlo para que domine el material del curso; a pesar de que no debe entregar soluciones a los ejercicios, usted es responsable del material cubierto en ellos.

Instrucciones para la entrega

Para esta tarea y todas las tareas futuras, la entrega de soluciones es *individual*. Por favor escriba claramente su nombre, código de estudiante y sección en cada hoja impresa entregada o en cada archivo de código (a modo de comentario). Adicionalmente, agregue la información de fecha y nombres de compañeros con los que colaboró; igualmente cite cualquier fuente de información que utilizó.

¿Cómo describir un algoritmo?

En algunos ejercicios y problemas se pide "dar un algoritmo" para resolver un problema. Una solución debe tomar la forma de un pequeño ensayo (es decir, un par de párrafos). En particular, una solución debe resumir en un párrafo el problema y cuáles son los resultados de la solución. Además, se deben incluir párrafos con la siguiente información:

- una descripción del algoritmo en castellano y, si es útil, pseudo-código;
- por lo menos un diagrama o ejemplo que muestre cómo funciona el algoritmo;
- una demostración de la corrección del algoritmo; y
- un análisis de la complejidad temporal del algoritmo.

Recuerde que su objetivo es comunicar claramente un algoritmo. Las soluciones algorítmicas correctas y descritas *claramente* recibirán alta calificación; soluciones complejas, obtusas o mal presentadas recibirán baja calificación.

Ejercicios

22.1-1, 22.1-2, 22.1-3 (página 592), 22.2-1, 22.2-2, 22.2-4 (página 602), 22.3-1, 22.3-2, 22.3-3, 22.3-7, 22.3-12 (páginas 610-612).

Problemas conceptuales

1. Problema 22-2: *Articulation points, bridges, and biconnected components* (Cormen et al. páginas 621 y 622)
2. Ejercicio 4: *Butterflies* (Kleinberg & Tardos página 107)
3. Ejercicio 11: *Virus* (Kleinberg & Tardos página 111)

Problemas prácticos

Hay cuatro problemas prácticos cuyos enunciados aparecen a partir de la siguiente página.

A - The Seasonal War

Source file name: `war.py`
Time limit: 1 second

The inhabitants of Tigerville and Elephantville are engaged in a seasonal war. Last month, Elephantville successfully launched and orbited a spy telescope called the Bumble Scope. The purpose of the Bumble Scope was to count the number of War Eagles in Tigerville. The Bumble Scope, however, developed two problems because of poor quality control during its construction. Its primary lens was contaminated with bugs which block part of each image, and its focusing mechanism malfunctioned so that images vary in size and sharpness.

The computer programmers, who must rectify the Bumble Scope's problems are being held hostage in a Programming Contest Hotel in Alaland by elephants dressed like tigers. The Bumble Scope's flawed images are stored by pixel in a file called Bumble.in. Each image is square and each pixel or cell contains either a 0 or a 1. The unique Bumble Scope Camera (BSC) records at each pixel location a 1 if part or all of a war eagle is present and a 0 if any other object, including a bug, is visible. The programmers must assume the following:

1. A war eagle is represented by at least a single binary one.
2. Cells with adjacent sides on common vertices, which contain binary ones, comprise one war eagle. A very large image of one war eagle might contain all ones.
3. Distinct war eagles do not touch one another. This assumption is probably flawed, but the programmers are desperate.
4. There is no wrap-around. Pixels on the bottom are not adjacent to the top and the left is not adjacent to the right (unless, of course, there are only 2 rows or 2 columns).

Write a program that reads images of pixels from the input, correctly counts the number of war eagles in the images, and prints the image number and war eagle count for that image.

Input

The input consists of several test cases. The first line of each test case consists of a positive integer n, $1 \leq n \leq 100$, defining the side of the square image. Then n lines follow: the ith line contains a sequence $a_1 \cdots a_n$ of bits describing the pixels of the ith row of the image.

The input must be read from standard input.

Output

For each test case, your program must output the image number in one line and the war eagle count for that image in the next line.

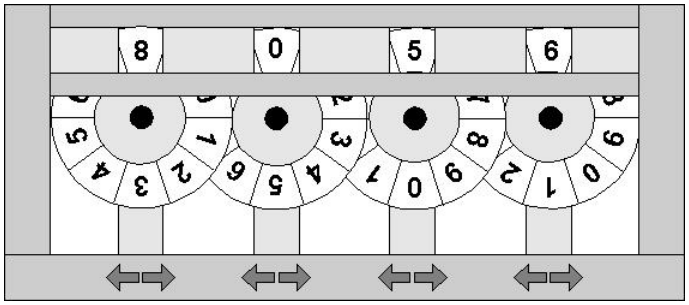
The output must be written to standard output.

Sample Input	**Sample Output**
6 100100 001010 000000 110000 111000 010100 8 01100101 01000001 00011000 00000010 11000011 10100010 10000001 01100000	Image number 1 contains 3 war eagles. Image number 2 contains 6 war eagles.

B - Playing with Wheels

Source file name: `wheels.py`
Time limit: 1 second

In this problem we will be considering a game played with four wheels. Digits ranging from 0 to 9 are printed consecutively (clockwise) on the periphery of each wheel. The topmost digits of the wheels form a four-digit integer. For example, in the following figure the wheels form the integer 8056. Each wheel has two buttons associated with it. Pressing the button marked with a left arrow rotates the wheel one digit in the clockwise direction and pressing the one marked with the right arrow rotates it by one digit in the opposite direction.



The game starts with an initial configuration of the wheels. Say, in the initial configuration the topmost digits form the integer $S_1\ S_2\ S_3\ S_4$. You will be given some (say, n) forbidden configurations $F_{i_1}\ F_{i_2}\ F_{i_3}\ F_{i_4}$ ($1 \leq i \leq n$) and a target configuration $T_1\ T_2\ T_3\ T_4$. Your job will be to write a program that can calculate the minimum number of button presses required to transform the initial configuration to the target configuration by never passing through a forbidden one.

Input

The first line of the input contains an integer N giving the number of test cases to follow.

The first line of each test case contains the initial configuration of the wheels specified by 4 digits. Two consecutive digits are separated by a space. The next line contains the target configuration. The third line contains an integer n giving the number of forbidden configurations. Each of the following n lines contains a forbidden configuration. There is a blank line between two consecutive input sets.

The input must be read from standard input.

Output

For each test case in the input print a line containing the minimum number of button presses required. If the target configuration is not reachable then print `-1`.

The output must be written to standard output.

Sample Input	**Sample Output**
2 8 0 5 6 6 5 0 8 5 8 0 5 7 8 0 4 7 5 5 0 8 7 5 0 8 6 4 0 8 0 0 0 0 5 3 1 7 8 0 0 0 1 0 0 0 9 0 0 1 0 0 0 9 0 0 1 0 0 0 9 0 0 1 0 0 0 9 0 0 0	14 -1

C - Knight in War Grid

Source file name: `knight.py`
Time limit: 1 second

Once upon an ancient time, a knight was preparing for the great battle in GridLand. The GridLand is divided into square grids. There are R horizontal and C vertical grids. Our particular knight in this case can always give an (M, N) move, i.e. he can move M squares horizontally and N squares vertically or he can move M squares vertically and N squares horizontally in a single move. In other words he can jump from square (a, b) to square (c, d) if and only if, either ($|a - c| = M$ and $|b - d| = N$) or ($|a - c| = N$ and $|b - d| = M$). However, some of the squares in the war field are filled with water. For a successful jump from one square to another, none of the squares should contain water. Now, the knight wants to have a tour in the war field to check if everything is alright or not. He will do the following:

a) He will start and end his tour in square $(0, 0)$ but visit as many squares as he can.

b) For each square s_i , he counts the number k_i of distinct squares, from which he can reach s_i in one jump (satisfying the jumping condition). Then he marks the square as an even square if k_i is even or marks it odd if k_i is odd. The squares he cannot visit remain unmarked.

c) After coming back to square $(0, 0)$ he counts the number of even and odd marked squares. He can visit a square more than once.

You, as an advisor of the knight, suggested that, he can do it without visiting all the squares, just by writing a program. So the knight told you to do so. He will check your result at the end of his visit.

Input

The first line of input will contain T ($T \geq 0$) denoting the number of cases.

Each case starts with four integers R, C, M, N ($1 < R,\ C \leq 100,\ 0 \leq M,\ N \leq 50,\ M + N \neq 0$). The next line contains an integer W ($0 \leq W < R \cdot C$), which is the number of distinct grids containing water. Each of the next W lines contains a pair of integer x_i , y_i ($0 \leq x_i < R$, $0 \leq y_i < C$, $0 \neq x_i + y_i$).

The input must be read from standard input.

Output

For each case, print the case number and the number of even and odd marked squares.

The output must be written to standard output.

Sample Input	**Sample Output**
2 3 3 2 1 0 4 4 2 1 2 3 3 1 1	Case 1: 8 0 Case 2: 4 10

D - Forwarding Emails

Source file name: `emails.py`
Time limit: x seconds

"... so forward this to ten other people, to prove that you believe the emperor has new clothes."

Aren't those sorts of emails annoying?

Martians get those sorts of emails too, but they have an innovative way of dealing with them. Instead of just forwarding them willy-nilly, or not at all, they each pick one other person they know to email those things to every time - exactly one, no less, no more (and never themselves). Now, the Martian clan chieftain wants to get an email to start going around, but he stubbornly only wants to send one email. Being the chieftain, he managed to find out who forwards emails to whom, and he wants to know: which Martian should he send it to maximize the number of Martians that see it?

Input

The first line of input will contain T ($T \geq 0$) denoting the number of cases. Each case starts with a line containing an integer N ($2 \leq N \leq 50\,000$) denoting the number of Martians in the community. Each of the next N lines contains two integers: u, v ($1 \leq u, v \leq N, u \neq v$) meaning that Martian u forwards email to Martian v.

The input must be read from standard input.

Output

For each case, print the case number and an integer m, where m is the Martian that the chieftain should send the initial email to. If there is more than one correct answer, output the smallest number.

The output must be written to standard output.

Sample Input	**Sample Output**
3 3 1 2 2 3 3 1 4 1 2 2 1 4 3 3 2 5 1 2 2 1 5 3 3 4 4 5	Case 1: 1 Case 2: 4 Case 3: 3