AGRA - Árboles y Grafos, 2018-2
Tarea 5: Semanas 11 y 12

Para entregar el miércoles 17 de octubre de 2018

Problemas prácticos a las 23:59 (17 de octubre) 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.

Problemas conceptuales

No hay en esta tarea.

Problemas prácticos

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

A - Airports

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

The government of a certain developing nation wants to improve transportation in one of its most inaccessible areas, in an attempt to attract investment. The region consists of several important locations that must have access to an airport.

Of course, one option is to build an airport in each of these places, but it may turn out to be cheaper to build fewer airports and have roads link them to all of the other locations. Since these are long distance roads connecting major locations in the country (e.g. cities, large villages, industrial areas), all roads are two-way. Also, there may be more than one direct road possible between two areas. This is because there may be several ways to link two areas (e.g. one road tunnels through a mountain while the other goes around it etc.) with possibly differing costs.

A location is considered to have access to an airport either if it contains an airport or if it is possible to travel by road to another location from there that has an airport.

You are given the cost of building an airport and a list of possible roads between pairs of locations and their corresponding costs. The government now needs your help to decide on the cheapest way of ensuring that every location has access to an airport. The aim is to make airport access as easy as possible, so if there are several ways of getting the minimal cost, choose the one that has the most airports.

Input

The first line of input contains the integer T ($T \geq 0$), the number of test cases. The rest of the input consists of T cases. Each case starts with two integers N, M and A ($0 < N \leq 10\,000, 0 \leq M \leq 100\,000, 0 < A \leq 10\,000$) separated by white space. The expression N denotes the number of locations, M is the number of possible roads that can be built, and A is the cost of building an airport.

The following M lines each contain three integers X, Y and C ($1 \leq X, Y \leq N, 0 < C \leq 10\,000$), separated by white space. X and Y are two locations, and C is the cost of building a road between X and Y.

The input must be read from standard input.

Output

Your program should output exactly T lines, one for each case. Each line should be of the form '`Case #X: Y Z`', where X is the case number Y is the minimum cost of making roads and airports so that all locations have access to at least one airport, and Z is the number of airports to be built. As mentioned earlier, if there are several answers with minimal cost, choose the one that maximizes the number of airports.

The output must be written to standard output.

Sample Input	**Sample Output**

`2`
`4 4 100`
`1 2 10`
`4 3 12`
`4 1 41`
`2 3 23`
`5 3 1000`
`1 2 20`
`4 5 40`
`3 2 30`
	`Case #1: 145 1`
	`Case #2: 2090 2`

B - Bank Robbery

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

Arsène Lupin is a gentleman thief and a master of disguise; he has been responsible for heists no right-minded individual would believe possible. He is also, very much, the ladies' man.

Lupin is about to drop everything he is currently doing to come to the aid of some of his friends who are planning a bank robbery in the infamous Kingdom of Aksum: his friends have identified the location of banks they are willing to rob, as well as the location of the police stations that serve the city. As a matter of fact, they have come up with a map of the entire city in which bidirectional roads connecting sites and traveling times between sites have been detailed.

Despite the criminal nature of his activities, Lupin has a strict code he follows in order to avoid tainting his reputation: he has never been caught by the authorities. In order to help his friends and, at the same time, keep his well-earned reputation, Lupin is wondering which banks can be robbed so that they are the ones furthest away from any police station serving the Kingdom of Aksum.

Input

The input consists of several test cases. Each test case begins with 4 blank-separated integer numbers N, M, B, P ($1 \leq N \leq 1\,000$, $0 \leq M$, $1 \leq B \leq N$, $0 \leq P < N$) denoting, respectively, the number of sites in the city, the number of roads in the city, the number of banks in the city, and the number of police stations in the city. The next M lines contain each three blank-separated integers U, V, T ($0 \leq U < N$, $0 \leq V < N$, $U \neq V$, $0 \leq T \leq 10\,000$) denoting that there is a road between sites U and V which takes T time units to transit. The next line contains B blank-separated and pairwise distinct site numbers identifying the location of banks. If $P \neq 0$, then follows a line with P blank-separated and pairwise distinct site numbers identifying the location of police stations. You can assume that a bank and a police station are never located at the same site.

The input must be read from standard input.

Output

For each test case, output two lines. The first line should contain two blank-separated figures S, E denoting, respectively, the number of banks furthest away from any police station and the minimum time it would take to transit from any police station to these banks. If E is not an integer number, then output '*' instead. The second line should contain S blank-separated integers, in ascending order, corresponding to the sites where banks are located with minimum time from any police station being exactly E time units.

The output must be written to standard output.

Sample Input	**Sample Output**
5 6 2 1 0 1 5 0 2 2 1 3 6 1 4 1 2 3 4 3 4 3 1 4 2 5 4 2 1 0 1 5 0 2 2 1 3 6 2 3 4 1 4 2 5 6 2 2 0 1 5 0 2 2 1 3 6 1 4 1 2 3 4 3 4 3 1 4 2 3	2 7 1 4 1 * 4 1 4 1

C - Space Invaders

Source file name: `space.py`
Time limit: 2 seconds

It is year 2551. The *Nostalgia for Infinity*, captain John Brannigan's ship, is cruising towards Resurgam, a planet considered a backwater on the edge of colonized human space. Some of the scientists aboard the ship will lead a team excavating the remains of the Amarantin, a long-dead, 900,000-year-old civilization that once existed on Resurgam. It is suspected that the entire Amarantin race achieved a much higher level of technological sophistication than was previously known.

As the Nostalgia for Infinity approaches Cerberus, a planet near Resurgam, the ship's defenses detect the presence of a highly hostile alien ship. Captain Brannigan is ready to fight and has stationed the ship behind a shield that seems to have been previously built by the Amarantin. Nostalgia for Infinity instruments depict space battle scenarios in $2D$. In particular, the shield is depicted as a rectangular region with r rows and c columns. Each cell in a shield can be either *active* (represented by '`#`') or *disabled* (represented by '`.`'). It is said that a shield is *breached* iff there is a connected sequence of disabled cells between the first and the last rows of the shield. Two cells are *connected* iff they are adjacent vertically or horizontally. For example, the figure below depicts two shields of r=4 rows and c=5 columns (numbers added for clarity: on top, they indicate the column number; on the right, they indicate the row number). The shield on the left is not breached, while the one on the right is breached.

```
12345          12345
##..# 1        ##..# 1
..### 2        ...## 2
#.#.. 3        #.#.. 3
..##. 4        ..##. 4
```

If it is assumed that the first row of a shield is facing north, then the alien ship is located north of the first row and it fires in the south direction; also, the Nostalgia for Infinity is located south of the last row and it fires in the north direction. In this way, the shield is a barrier between the two enemy ships. The effect of a shot fired from a ship is to disable the first active cell it encounters in the shield, if any. Shots are indicated by the column on which they are fired and always follow a vertical path. For example, if the aliens shoot at either column $1, 2, 3, 4$ the left shield depicted above would be breached. The right shield corresponds to the result of shooting column 3.

You, as the ship's new gunnery officer, have been assigned with the task of determining whether or not, during battle, the shield is breached. If so, captain Brannigan would also like to know when and who breached the shield first.

Input

The input consists of several test cases. The first line of a test case contains three blank-separated integers r, c, and s ($1 \leq r \leq 1000$, $1 \leq c \leq 1000$, $0 \leq s \leq 10000$) indicating, respectively, the number of rows and columns in the shield, and the number of shots during the battle. Each of the next r lines contain c characters '`#`' or '`.`'. The next s lines contain each an integer a ($1 \leq |a| \leq c$) indicating a shot fired at column $|a|$ of the shield: if $a > 0$ then it is an alien shot; otherwise it is Nostalgia for Infinity shot.

The input must be read from standard input.

Output

For each test case, output exactly one line:

- If the shield was breached before any shot was fired, then output `0`.
- If the shield was never breached during the battle, then output `X`.
- If the shield was first breached by the Nostalgia for Infinity during the battle at shot k (counting from 1), then output the number $-k$.
- If the shield was first breached by the aliens during the battle at shot k (counting from 1), then output the number k.

The output must be written to standard output.

Sample Input	**Sample Output**
2 2 2 .# .. 1 1 2 3 2 ##. .## 2 -2 2 3 2 ##. .## -2 2 2 1 1 # # -1	0 -2 2 X

D - X-Plosives

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

A secret service developed a new kind of explosive that attain its volatile property only when a specific association of products occurs. Each product is a mix of two different simple compounds, to which we call a *binding pair*. If $N > 2$, then mixing N different binding pairs containing N simple compounds creates a powerful explosive. For example, the binding pairs $A + B$, $B + C$, $A + C$ (three pairs, three compounds) result in an explosive, while $A + B$, $B + C$, $A + D$ (three pairs, four compounds) does not.

You are not a secret agent but only a guy in a delivery agency with one dangerous problem: receive binding pairs in sequential order and place them in a cargo ship. However, you must avoid placing in the same room an explosive association. So, after placing a set of pairs, if you receive one pair that might produce an explosion with some of the pairs already in stock, you must refuse it, otherwise, you must accept it. Compute the number of refusals given a sequence of binding pairs.

Lets assume you receive the following sequence: $A + B$, $G + B$, $D + F$, $A + E$, $E + G$, $F + H$. You would accept the first four pairs but then refuse $E + G$ since it would be possible to make the following explosive with the previous pairs: $A + B$, $G + B$, $A + E$, $E + G$ (4 pairs with 4 simple compounds). Finally, you would accept the last pair, $F + H$.

Input

The input will contain several test cases, each of them as described below. Consecutive test cases are separated by a single blank line. Instead of letters we will use integers to represent compounds. The input contains several lines. Each line (except the last) consists of two integers (each integer lies between 0 and 10^5) separated by a single space, representing a binding pair. Each test case ends in a line with the number '−1'. You may assume that no repeated binding pairs appears in the input.

The input must be read from standard input.

Output

For each test case, output a single line with the number of refusals.

The output must be written to standard output.

Sample Input	**Sample Output**
1 2 3 4 3 5 3 1 2 3 4 1 2 6 6 5 -1	3

E - Mice and Maze

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

A set of laboratory mice is being trained to escape a maze. The maze is made up of cells, and each cell is connected to some other cells. However, there are obstacles in the passage between cells and therefore there is a time penalty to overcome the passage Also, some passages allow mice to go one-way, but not the other way round.

Suppose that all mice are now trained and, when placed in an arbitrary cell in the maze, take a path that leads them to the exit cell in minimum time.

We are going to conduct the following experiment: a mouse is placed in each cell of the maze and a count-down timer is started. When the timer stops we count the number of mice out of the maze.

Write a program that, given a description of the maze and the time limit, predicts the number of mice that will exit the maze. Assume that there are no bottlenecks is the maze, i.e. that all cells have room for an arbitrary number of mice.

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The maze cells are numbered $1, 2, ..., N$, where N is the total number of cells. You can assume that $N \leq 100$.

The first three input lines contain N (the number of cells in the maze), E (the number of the exit cell), and T (the starting value for the count-down timer, in some arbitrary time unit).

The fourth line contains the number M of connections in the maze, and is followed by M lines, each specifying a connection with three integer numbers: two cell numbers a and b (in the range $1, .., N$) and the number of time units it takes to travel from a to b. Notice that each connection is one-way, i.e., the mice can't travel from b to a unless there is another line specifying that passage. Notice also that the time required to travel in each direction might be different.

The input must be read from standard input.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

The output consists of a single line with the number of mice that reached the exit cell E in at most T time units.

The output must be written to standard output.

Sample Input	**Sample Output**
1 4 2 1 8 1 2 1 1 3 1 2 1 1 2 4 1 3 1 1 3 4 1 4 2 1 4 3 1	3