ADA - Análisis y Diseño de Algoritmos, 2019-1
Tarea 4: Semanas 8 y 9

Para entregar el viernes 22 de marzo/lunes 25 de marzo de 2019

Problemas conceptuales a las 16:00 (22 de marzo) en el Departamento de Electrónica y Ciencias de la Computación

Problemas prácticos a las 23:59 (25 de marzo) 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). 22.4-1, 22.4-2, 22.4-3, 22.4-4 (páginas 614 y 615), 23.1-1, 23.1.4 (página 629), 23.2-1, 23.2-2 (página 637).

Problemas conceptuales

1. Ejercicio 3.5: *Binary Trees* (Kleinberg y Tardos, página 108).
2. Ejercicio 3.7: *Wireless Networks* (Kleinberg y Tardos, página 108).
3. Ejercicio 4.2: *Minimum Spanning Trees* (Kleinberg y Tardos, página 189).
4. Ejercicio 4.5: *Quiet Country Roads* (Kleinberg y Tardos, página 191).

Problemas prácticos

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

A - Heavy Cycle Edges

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

Given an undirected graph with edge weights, a minimum spanning tree is a subset of edges of minimum total weight such that any two nodes are connected by some path containing only these edges. A popular algorithm for finding the minimum spanning tree T in a graph proceeds as follows:

- let T be initially empty;
- consider the edges $e_1, \ldots, e_m$ in increasing order of weight: add e_i to T if the endpoints of e_i are not connected by a path in T.

An alternative algorithm is the following:

- let T be initially the set of all edges;
- while there is some cycle C in T, remove edge e from T where e has the heaviest weight in C.

Your task is to implement a function related to this algorithm. Given an undirected graph G with edge weights, your task is to output all edges that are the heaviest edge in some cycle of G.

Input

The first input of each case begins with integers n and m with $1 \leq n \leq 1\,000$ and $0 \leq m \leq 25\,000$ where n is the number of nodes and m is the number of edges in the graph. Following this are m lines containing three integers u, v, and w describing a weight w edge connecting nodes u and v where $0 \leq u, v < n$ and $0 \leq w < 2^{31}$. Input is terminated with a line containing $n = m = 0$; this case should not be processed. You may assume no two edges have the same weight and no two nodes are directly connected by more than one edge.

The input must be read from standard input.

Output

Output for an input case consists of a single line containing the weights of all edges that are the heaviest edge in some cycle of the input graph. These weights should appear in increasing order and consecutive weights should be separated by a space. If there are no cycles in the graph then output the text 'forest' instead of numbers.

The output must be written to standard output.

Sample Input	**Sample Output**
3 3 0 1 1 1 2 2 2 0 3 4 5 0 1 1 1 2 2 2 3 3 3 1 4 0 2 0 3 1 0 1 1 0 0	3 2 4 forest

B - Getting Gold

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

We're building an old-school back-to-basics computer game. It's a very simple text based adventure game where you walk around and try to find treasure, avoiding falling into traps. The game is played on a rectangular grid and the player gets very limited information about her surroundings.

The game will consist of the player moving around on the grid for as long as she likes (or until she falls into a trap). The player can move up, down, left and right (but not diagonally). She will pick up gold if she walks into the same square as the gold is. If the player stands next to (i.e., immediately up, down, left, or right of) one or more traps, she will "sense a draft" but will not know from what direction the draft comes, or how many traps she's near. If she tries to walk into a square containing a wall, she will notice that there is a wall in that direction and remain in the position where she was.

For scoring purposes, we want to show the player how much gold she could have gotten safely. That is, how much gold can a player get playing with an optimal strategy and always being sure that the square she walked into was safe. The player does not have access to the map and the maps are randomly generated for each game so she has no previous knowledge of the game.

Input

The input contains several test cases, each of them as described below. The first line of input contains two positive integers W and H, neither of them smaller than 3 or larger than 50, giving the width and the height of the map, respectively. The next H lines contain W characters each, giving the map. The symbols that may occur in a map are as follows:

`P` the player's starting position.

`G` a piece of gold.

`T` a trap.

`#` a wall.

`.` normal floor.

There will be exactly one 'P'in the map, and the border of the map will always contain walls.

The input must be read from standard input.

Output

For each test case, write to the output the number of pieces of gold the player can get without risking falling into a trap, on a line by itself.

The output must be written to standard output.

Sample Input	**Sample Output**
7 4 ####### #P.GTG# #..TGG# ####### 8 6 ######## #...GTG# #..PG.G# #...G#G# #..TG.G# ########	1 4

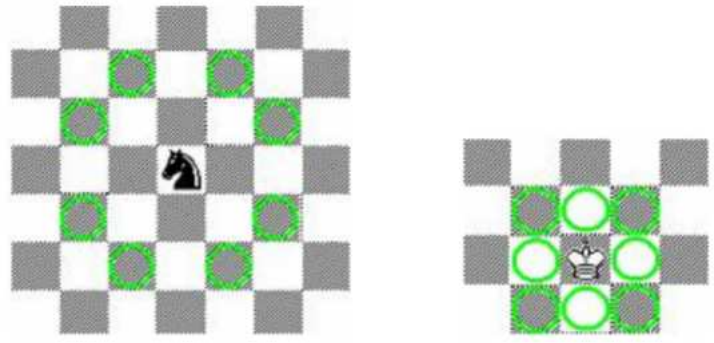
C - Crazy King

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

King Peter lives in kingdom A, and his daughter in kingdom B. King recieved a letter telling that her daughter gave birth to a child. King is incredibly curious to see his grandchild! Unfortunately that's not gonna be that easy. Kingdoms A and B are separated by a forest. There are lots of enemies in the forest, and King is not that curious to see them. If they attack king on his way to kingdom B, then he will never ever see his grandchild and daughter again because of lethal consequences.

Security Council of the King disposes information about location of the enemies, which makes the things easier for king. For some unknown reason a forest is $M \times N$ chessboard (M is the number of rows and N is the number of columns, with $N, M \leq 100$ are positive integers).

Enemies of the King can ride horses as showed in the picture. Usually horses ride (or jump) that way in Chess. Unfortunately king can't take an airplane from point A to point B because it is not invented yet. So he moves the same way as chess-king does (refer to picture for details).



King can't move to a square X, if a horse of the enemy is on that square. While the king is moving horses are not, but if at least one horse can reach square X in one move, then king can't move to that square (except for the case when square X is either kingdom A or B).

You are the chief of Electronic Intelligence dapartment of kingdom A (by the way the computers are already invented). And you're asked to find the length of the shortest route L from kingdom A to B, as king can't wait any longer.

Input

The first line of input contains the number of tests $T \geq 0$. The first line of each test contains 2 numbers M and N. Then M lines follow each containing N symbols from the set $S = \{$`'.'`, `'Z'`, `'A'`, `'B'`$\}$. The `'.'` character means that square is not occupied. `'Z'`: horse occupies that square. `'A'`: kingdom A, `'B'`: kingdom B. Each test contains exactly one kingdom A and B.

The input must be read from standard input.

Output

Find number L for each test and print line 'Minimal possible length of a trip is L' if King can reach kingdom B. Replace L with corresponding number. If King can't safely reach the kingdom B print line 'King Peter, you can't go now!'.

The output must be written to standard output.

Sample Input	**Sample Output**
4 5 5 .Z..B ..Z.. Z...Z .Z... A.... 3 2 ZB .Z AZ 6 5 ....B ....Z.. ..... A..Z. 3 3 ZZ. ... AB.	King Peter, you can't go now! Minimal possible length of a trip is 2 King Peter, you can't go now! Minimal possible length of a trip is 1

D - The Mysterious X Network

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

One of the reasons for which École polytechnique (nicknamed "X" for reasons to be explained during the debriefing talk) is so deeply rooted in French society is its famous network of camarades —former students of the same school. When one camarade wants something (money, job, etc.), he can ask this network for help and support. In practice, this means that when he/she wants to reach some other camarade, not always of the same year, then surely he can find intermediate camarades to get to her/him. Note that the "camarade" relationship is symmetric. Due to the magic of the X network, there is always a means to reach anybody.

The program you have to write is supposed to help to minimize the number of these intermediate camarades.

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 huge file of all living camarades is simplified so as to obey the following format. The first line in the file is the number of camarades, say N, an integer $1 \leq N \leq 105$. Camarades are labeled from 0 to $N-1$. Follow N lines. Each line starts with the camarade label c, followed by the number of other camarades he/she knows, say n_c, followed by the labels of those n_c camarades. All these integers are separated by a single blank. It is assumed that n_c is always less than 100. The last line in the file is the label of the camarade seeking help (say c_1) followed by the label of the camarade he wants help from, say c_2 ($c_2 \neq c_1$).

The input must be read from standard input.

Output

For each test case, your program should output three integers separated by a blank: c_1, c_2, and the minimal number of intermediate camarades to reach c_2. The outputs of two consecutive cases will be separated by a blank line.

The output must be written to standard output.

Sample Input	**Sample Output**
1 4 0 3 1 2 3 1 1 0 2 2 0 3 3 2 0 2 1 2	1 2 1