AGRA - Árboles y Grafos, 2023-2
Tarea 2: Semanas 6 y 7

Para entregar el miércoles 13 de septiembre de 2023

Problemas conceptuales a las 23:59 por BrightSpace

Problemas prácticos a las 23:59 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

La siguiente colección de ejercicios, tomados del libro de Cormen et al. es para repasar y afianzar conceptos, pero no deben ser entregados como parte de la tarea:

20.4-3, 20.4-4, 20.4-5 (páginas 575-6), 20.5-1 , 20.5-3, 20.5-4, 20-5.7, 20-5.8 (páginas 580-1).

Problemas conceptuales

1. Ejercicio 20.4-2: *Counting Simple Paths* (Cormen et al., página 575).
2. Problema 20-2: *Articulation points, bridges, and biconnected components* (Cormen et al., página 581).

Problemas prácticos

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

A - Doves and Bombs

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

It is the year 95 ACM (After the Crash of Microsoft). After many years of peace, a war has broken out. Your nation, the island of Evergreen Macros And Confusing Shortcuts (EMACS), is defending itself against the railway empire ruled by Visually Impaired Machinists (VIM).

In the days leading up to the outbreak of war, your government devoted a great deal of resources toward gathering intelligence on VIM. It discovered the following:

- The empire has a large network of railway stations connected by bidirectional tracks.
- Before the outbreak of the war, each railway station was directly connected to at most 10 other stations.
- Information is constantly being exchanged in VIM, but, due to a design flaw, the only way it can be exchanged is to send the messages by train. Before the outbreak of the war, it was possible to send a message by train from any station to any other station.
- As a last resort, the empire's central command can send messages by carrier pigeon, but it tries to avoid this at all costs, as the only pigeons suitable for the job must be imported, at great expense, from the far-away land of Pigeons In Courier Outfits (PICO).
- Once a pigeon has delivered a message to a railway station, it must rest, and thus cannot be used again. If a pigeon has delivered a broadcast message to a particular station, the message is passed on, if possible, by train.

Based on this information, the government of EMACS has come up with a plan to disrupt the activities of the evil empire. They will send bomber planes to bomb the railway stations, thus hampering communications in the empire. This will necessitate to acquire many carrier pigeons by the empire, distracting it from its deadly wartime activities.

Unfortunately, your government spent so much money on gathering intelligence that it has a very limited amount left to build bombs. As a result, it can bomb only one target. You have been charged with the task of determining the best candidate railway stations in the empire to bomb, based on their "pigeon value".

The "pigeon value" of a station is the minimum number of pigeons that after bombing this station, will be required to broadcast a message from the empire central command to all non-bombed stations. The location of the empire central command is unknown but we know that it is not located at a railway station. This implies, that when the central command needs to send a message to some non-bombed station they have to use at least one pigeon and then the message can be further transmitted by the railway.

Input

The input file contains several test cases. The data for each case begins with a line containing the following two integers:

- n the number of railway stations in the empire ($3 \leq n \leq 10\,000$). The stations will be numbered starting from 0, up to $n - 1$
- m the number of stations to be identified as candidate bombing targets ($1 \leq m \leq n$).

Next few lines consists of pairs of integers. Each pair (x, y) indicates the presence of a bidirectional railway line connecting railway stations x and y. This sequence is terminated by a line containing two minus 1 as shown in the sample input.

Input is terminated by a case where the value of $n = m = 0$. This case should not be processed.

The input must be read from standard input.

Output

For each case of input the output should give *m* most desirable railway stations to bomb. There should be exactly *m* lines, each with two integers separated by a single space. The first integer on each line will be the number of a railway station, and the second will be the “pigeon value” of the station. This list should be sorted, first by “pigeon value”, in descending order, and within the same “pigeon value” by railway station numbers, in ascending order. Print a blank line after the output for each set of input.

The output must be written to standard output.

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

B - Come and Go

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

In a certain city there are N intersections connected by one-way and two-way streets. It is a modern city, and several of the streets have tunnels or overpasses. Evidently it must be possible to travel between any two intersections. More precisely given two intersections V and W it must be possible to travel from V to W and from W to V.

Your task is to write a program that reads a description of the city street system and determines whether the requirement of connectedness is satisfied or not.

Input

The input contains several test cases. The first line of a test case contains two integers N and M, separated by a space, indicating the number of intersections ($2 \leq N \leq 2000$) and number of streets ($2 \leq M \leq N(N-1)/2$). The next M lines describe the city street system, with each line describing one street. A street description consists of three integers V, W and P, separated by a blank space, where V and W are distinct identifiers for intersections ($1 \leq V, W \leq N, V \neq W$) and P can be 1 or 2; if $P = 1$ the street is one-way, and traffic goes from V to W; if $P = 2$ then the street is two-way and links V and W. A pair of intersections is connected by at most one street.

The last test case is followed by a line that contains only two zero numbers separated by a blank space.

The input must be read from standard input.

Output

For each test case your program should print a single line containing an integer G, where G is equal to one if the condition of connectedness is satisfied, and G is zero otherwise.

The output must be written to standard output.

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

C - Cactus

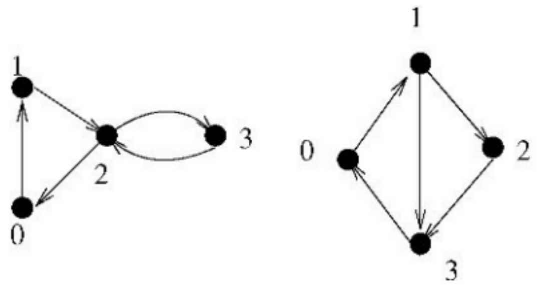
Source file name: `cactus.py`
Time limit: x seconds

A directed graph is a set V of *vertices* and a set of $E \subset V \times V$ *edges*. An edge (u, v) is said to be directed from u to v (the edge (v, u) has the opposite direction). A directed cycle in a directed graph is a sequence of edges

$$(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)$$

such that $u_{i+1} = v_i$ for $i = 1, \ldots, k-1$, and $u_1 = v_k$. The directed cycle is simple if $u_i \neq u_j$ whenever $i \neq j$ (i.e., if it does not pass through a vertex twice).

In a *strongly connected* directed graph, there is for every pair u, v of vertices some directed cycle (not necessarily simple) that visits both u and v.



A directed graph is a cactus if and only if it is strongly connected and each edge is part of exactly one directed simple cycle. The first graph is a cactus, but the second one is not since for instance the edge (0,1) is in two simple cycles.

The reason for the name is that a "cactus" consists of several simple cycles connected to each other in a tree-like fashion, making it look somewhat like a cactus.

Write a program that given a directed graph determines if it is a cactus or not. The graph can have several thousand vertices.

Input

The first line contains an integer which is the number of test cases (less than 20). Each test case starts a line with an integer $n > 0$ followed by line with an integer $m > 0$ giving the number of vertices (n) and edges (m) in a graph (at most 10000 of each). The vertices are numbered 0 through $n - 1$. The following m lines describe the edges as pairs of numbers u, v denoting an edge directed from u to v.

There will never be more than one edge from u to v for any pair of vertices u and v. There are no loops, i.e., no edges from a vertex to itself.

The input must be read from standard input.

Output

For each test case output a single line with a single string. Output `'YES'` if the graph is a cactus, and output `'NO'` if it is not.

The output must be written to standard output.

Sample Input

```
2
4
5
0 1
1 2
2 0
2 3
3 2
4
5
0 1
1 2
2 3
3 0
1 3
```

Sample Output

```
YES
NO
```

D - Hedge Mazes

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

The Queen of Nlogonia is a fan of mazes, and therefore the queendom's architects built several mazes around the Queen's palace. Every maze built for the Queen is made of rooms connected by corridors. Each corridor connects a different pair of distinct rooms and can be transversed in both directions.

The Queen loves to stroll through a maze's rooms and corridors in the late afternoon. Her servants choose a different challenge for every day, that consists of finding a simple path from a start room to an end room in a maze. A simple path is a sequence of distinct rooms such that each pair of consecutive rooms in the sequence is connected by a corridor. In this case the first room of the sequence must be the start room, and the last room of the sequence must be the end room. The Queen thinks that a challenge is good when, among the routes from the start room to the end room, exactly one of them is a simple path. Can you help the Queen's servants to choose a challenge that pleases the Queen?

For doing so, write a program that given the description of a maze and a list of queries defining the start and end rooms, determines for each query whether that choice of rooms is a good challenge or not.

Input

Each test case is described using several lines. The first line contains three integers R, C and Q representing respectively the number of rooms in a maze ($2 \leq R \leq 10^4$), the number of corridors ($1 \leq C \leq 10^5$), and the number of queries ($1 \leq Q \leq 1000$). Rooms are identified by different integers from 1 to R. Each of the next C lines describes a corridor using two distinct integers A and B, indicating that there is a corridor connecting rooms A and B ($1 \leq A < B \leq R$). After that, each of the next Q lines describes a query using two distinct integers S and T indicating respectively the start and end rooms of a challenge ($1 \leq S < T \leq R$). You may assume that within each test case there is at most one corridor connecting each pair of rooms, and no two queries are the same.

The last test case is followed by a line containing three zeros.

The input must be read from standard input.

Output

For each test case output $Q + 1$ lines. In the i-th line write the answer to the i-th query. If the rooms make a good challenge, then write the character `'Y'` (uppercase). Otherwise write the character `'N'` (uppercase). Print a line containing a single character `'-'`(hyphen) after each test case.

The output must be written to standard output.

Sample Input

```
6 5 3
1 2
2 3
2 4
2 5
4 5
1 3
1 5
2 6
4 2 3
1 2
2 3
1 4
1 3
1 2
0 0 0
```

Sample Output

```
Y
N
N
-
N
Y
Y
```