Colombian Collegiate Programming League
CCPL 2013

Contest 2 -- March 16

Graphs

Problems

This set contains 8 problems; pages 1 to 13.

(Borrowed from several sources online.)

A - Random Walking

Source file name: `random.c`, `random.cpp` *or* `random.java`

The Army of Coin-tossing Monkeys (ACM) is in the business of producing randomness. Good random numbers are important for many applications, such as cryptography, online gambling, randomized algorithms and panic attempts at solutions in the last few seconds of programming competitions.

Recently, one of the best monkeys has had to retire. However, before he left, he invented a new, cheaper way to generate randomness compared to directly using the randomness generated by coin-tossing monkeys. The method starts by taking an undirected graph with 2^n nodes labelled $0, 1, \ldots, 2^n - 1$. To generate k random n-bit numbers, they will let the monkeys toss n coins to decide where on the graph to start. This node number is the first number output. The monkeys will then pick a random edge from this node, and jump to the node that this edge connects to. This new node will be the second random number output. They will then select a random edge from this node (possibly back to the node they arrived from in the last step), follow it and output the number of the node they landed on. This walk will continue until k numbers have been output.

During experiments, the ACM has noticed that different graphs give different output distributions, some of them not very random. So, they have asked for your help testing the graphs to see if the randomness is of good enough quality to sell.

They consider a graph good if, for each of the n bits in each of the k numbers generated, the probability that this bit is output as 1 is greater than 25% and smaller than 75%.

Input

The input will consist of several data sets. Each set will start with a line consisting of three numbers k, n, e separated by single spaces, where k is the number of n-bit numbers to be generated and e is the number of edges in the graph ($1 \leq k \leq 100$, $1 \leq n \leq 10$ and $1 \leq e \leq 2000$). The next e lines will consist of two space-separated integers v_1, v_2 where $0 \leq v_1, v_2 < 2^n$ and $v_1 \neq v_2$. Edges are undirected and each node is guaranteed to have at least one edge. There may be multiple edges between the same pair of nodes.

The last test case will be followed by a line with $k = n = e = 0$, which should not be processed.

The input must be read from standard input.

Output

For each input case, output a single line consisting of the word `Yes` if the graph is good, and `No` otherwise

The output must be written to standard output.

Sample input	**Sample output**
10 2 3 0 3 1 3 2 3 5 2 4 0 1 0 3 1 2 2 3 0 0 0	No Yes

B - Full Tank

Source file name: `tank.c, tank.cpp` *or* `tank.java`

After going through the receipts from your car trip through Europe this summer, you realised that the gas prices varied between the cities you visited. Maybe you could have saved some money if you were a bit more clever about where you filled your fuel?
To help other tourists (and save money yourself next time), you want to write a program for finding the cheapest way to travel between cities, filling your tank on the way. We assume that all cars use one unit of fuel per unit of distance, and start with an empty gas tank.

Input

The first line of input gives $1 \leq n \leq 1000$ and $0 \leq m \leq 10000$, the number of cities and roads. Then follows a line with n integers $1 \leq p_i \leq 100$, where p_i is the fuel price in the i-th city. Then follow m lines with three integers $0 \leq u, v < n$ and $1 \leq d \leq 100$, telling that there is a road between u and v with length d. Then comes a line with the number $1 \leq q \leq 100$, giving the number of queries, and q lines with three integers $1 \leq c \leq 100$, s and e, where c is the fuel capacity of the vehicle, s is the starting city, and e is the goal.

The input must be read from standard input.

Output

For each query, output the price of the cheapest trip from s to e using a car with the given capacity, or `impossible` if there is no way of getting from s to e with the given car.

The output must be written to standard output.

Sample input	**Sample output**
5 5 10 10 20 12 13 0 1 9 0 2 8 1 2 1 1 3 11 2 3 7 2 10 0 3 20 1 4	170 impossible

C - Kingdom

Source file name: `kingdom.c`, `kingdom.cpp` *or* `kingdom.java`

King Kong is the feared but fair ruler of Transylvania. The kingdom consists of two cities and $N < 150$ towns, with nonintersecting roads between some of them. The roads are bidirectional, and it takes the same amount of time to travel them in both directions. Kong has $G < 353535$ soldiers.

Due to increased smuggling of goat cheese between the two cities, Kong has to place his soldiers on some of the roads in such a way that it is impossible to go from one city to the other without passing a soldier. The soldiers must not be placed inside a town, but may be placed on a road, as close as Kong wishes, to any town. Any number of soldiers may be placed on the same road. However, should any of the two cities be attacked by a foreign army, the king must be able to move all his soldiers fast to the attacked city. Help him place the soldiers in such a way that this mobilizing time is minimized.

Note that the soldiers cannot be placed in any of the cities or towns. The cities have ZIP-codes 95050 and 104729, whereas the towns have ZIP-codes from 0 to $N - 1$. There will be at most one road between any given pair of towns or cities.

Input

The input contains several test cases. The first line of each test case is N, G, and E, where N and G are as defined above and $E < 5000$ is the number of roads. Then E lines follow, each of which contains three integers: A and B, the ZIP-codes of the endpoints, and ϕ, the time required to travel the road, $\phi < 1000$. The last line of the input is a line containing a single 0.

The input must be read from standard input.

Output

For each test case in the input, print the best mobilizing time possible, with one decimal. If the given number of soldiers is not enough to stop the goat cheese, print ‘‘Impossible’’ instead.

The output must be written to standard output.

Sample input	**Sample output**
4 2 6 95050 0 1 0 1 2 1 104729 1 95050 2 1 2 3 3 3 104729 1 4 1 6 95050 0 1 0 1 2 1 104729 1 95050 2 1 2 3 3 3 104729 1 4 2 7 95050 0 1 0 1 2 1 104729 1 95050 2 1 2 3 3 3 104729 1 2 1 5 0	2.5 Impossible 3.0

D - The Bottom of a Graph

Source file name: `bottom.c`, `bottom.cpp` *or* `bottom.java`

We will use the following (standard) definitions from graph theory. Let V be a nonempty and finite set, its elements being called *vertices* (or nodes). Let $E \subseteq V \times V$, its elements being called *edges*. Then $G = (V, E)$ is called a *directed graph*.
Let n be a positive integer, and let $p = (e_1, ..., e_n)$ be a sequence of length n of edges $e_i \in E$ such that $e_i = (v_i, v_{i+1})$ for a sequence of vertices $(v_1, ..., v_{n+1})$. Then p is called a *path* from vertex v_1 to vertex v_{n+1} in G and we say that v_{n+1} is reachable from v_1, writing $(v_1 \to v_{n+1})$.
Here are some new definitions. A node v in a graph $G = (V, E)$ is called a *sink*, if for every node w in G that is reachable from v, v is also reachable from w. The bottom of a graph is the subset of all nodes that are sinks, i.e.,

$$bottom(G) = \{v \in V \mid (\forall w \in V) v \to w \implies w \to v\}.$$

Your task is to calculate the bottom of certain graphs.

Input

The input contains several test cases, each of which corresponds to a directed graph G. Each test case starts with an integer number v, denoting the number of vertices of $G = (V, E)$, where the vertices will be identified by the integer numbers in the set $V = \{1, \ldots, v\}$. You may assume that $1 \leq v \leq 5000$. That is followed by a non-negative integer e and then e pairs of vertex identifiers $v_1, w_1, \ldots, v_e, w_e$ with the meaning that $(v_i, w_i) \in E$. There are no edges other than specified by these pairs. The last test case is followed by a zero.

The input must be read from standard input.

Output

For each test case output the bottom of the specified graph on a single line. To this end, print the numbers of all nodes that are sinks in sorted order separated by a single space character. If the bottom is empty, print an empty line.

The output must be written to standard output.

Sample input	**Sample output**
3 3 1 3 2 3 3 1 2 1 1 2 0	1 3 2

E - Dark Roads

Source file name: `dark.c`, `dark.cpp` *or* `dark.java`

Economic times these days are tough, even in Byteland. To reduce the operating costs, the government of Byteland has decided to optimize the road lighting. Till now every road was illuminated all night long, which costs 1 Bytelandian Dollar per meter and day. To save money, they decided to no longer illuminate every road, but to switch off the road lighting of some streets. To make sure that the inhabitants of Byteland still feel safe, they want to optimize the lighting in such a way, that after darkening some streets at night, there will still be at least one illuminated path from every junction in Byteland to every other junction.
What is the maximum daily amount of money the government of Byteland can save, without making their inhabitants feel unsafe?

Input

The input file contains several test cases. Each test case starts with two numbers m and n, the number of junctions in Byteland and the number of roads in Byteland, respectively. Input is terminated by $m = n = 0$. Otherwise, $1 \leq m \leq 200000$ and $m - 1 \leq n \leq 200000$. Then follow n integer triples x, y, z specifying that there will be a bidirectional road between x and y with length z meters ($0 \leq x, y < m$ and $x \neq y$). The graph specified by each test case is connected. The total length of all roads in each test case is less than 2^{31}.

The input must be read from standard input.

Output

For each test case print one line containing the maximum daily amount the government can save.

The output must be written to standard output.

Sample input	**Sample output**
7 11 0 1 7 0 3 5 1 2 8 1 3 9 1 4 7 2 4 5 3 4 15 3 5 6 4 5 8 4 6 9 5 6 11 0 0	51

F - Beacons

Source file name: `beacons.c`, `beacons.cpp` *or* `beacons.java`

In ancient times, communication was not as swift as it is today. When a kingdom was at war, it could take months to muster all the armed forces. But by using fire-lit beacons at strategic locations, it was still possible to quickly send emergency signals.
When the first beacon is lit, all other beacons within sight from it are also lit. All beacons within sight of these are then lit, and so on until all beacons are lit - assuming of course that all beacons are within sight of each other, directly or indirectly. If they are not, the dire news must be carried by riders between some beacons.
Given the location of all beacons in the kingdom as well as the location and size of all mountain peaks, write a program that determines how many messages must be sent by riders in order for all beacons to be lit when an enemy threatens the country.
For simplicity, we model the country in the following way: a beacon is represented as a point in the xy-plane and a mountain peak is represented as a circle. Two beacons are considered to be within sight of each other if no mountain peak blocks the straight line between the two beacons. *The input will be constructed so that the straight line between any pair of beacons will not touch the circumference of a mountain peak, unless it passes through the interior of another mountain peak. Mountain peaks will not overlap or touch, nor will any beacon be on a mountain peak or on its circumference.*

Input

There will be multiple test cases in the input. Each test case will begin with a line with s two integers n ($1 \leq n \leq 1000$) and m ($0 \leq m \leq 1000$) the number of beacons and the number of mountain peaks, respectively. Then follow n lines specifying the locations of the beacons. The location of each beacon is given as a pair of integers x and y ($0 \leq x, y \leq 10000$). Then follow m lines describing the mountain peaks. Each mountain peak is given as a pair of integers x and y ($0 \leq x, y \leq 10000$) specifying the location of the peak and a radius r ($1 \leq r \leq 5000$). The input will end with a line containing '0 0'.

The input must be read from standard input.

Output

For each test case, output a single integer: the number of messages that must be carried by riders for all beacons to be lit.

The output must be written to standard output.

Sample input	**Sample output**
6 3 1 8 5 4 7 7 9 2 16 6 17 10 4 7 2 6 3 1 12 6 3 4 4 0 4 8 4 4 0 4 8 2 2 1 6 2 1 2 6 1 6 6 1 0 0	2 1

G - Food Review

Source file name: `foodreview.c`, `foodreview.cpp` *or* `foodreview.java`

Frida is a writer for Cosmopolitan who writes restaurant reviews. She enjoys it a lot, but it seems that, throughout the years, she has reviewed all the restaurants on Earth. It's now time to move one level up; she is going to review the food served by the airlines, so that the readers can make better decisions on which flights to take.
Her boss gave her a list of flight connections that she needs to review for the upcoming issue of Cosmopolitan. She knows that they serve the same food in both directions of every flight, so she only needs to take it once. She realized that she will need to take some additional flights, because she can not make all reviews using only flights in the list from her boss. Therefore she did some quick research and made a list of additional flights which she might take. She will not review the food on these flights; they will only be used so that she can make all the reviews.
Frida's goal is to make all the reviews while spending the least money on flight tickets. Her office is in Stockholm, so she starts and ends her journey there. Each flight is both ways between two cities and has a fixed price in both directions. You can assume that it is possible to finish all the reviews using some of the additional flights.
For the purposes of this problem we ignore the price Frida has to pay for accommodation and we also ignore the departure and arrival times of flights by assuming that every flight is very often and reasonably short. We only focus on the total price of the flights.

Input

There will be multiple test cases in the input.
Each test case will begin with a line contains 2 space separated integers N, R,($2 \leq N \leq 13$, $0 \leq R \leq 78$), where N is the number of airports mentioned in the input and R is the number of flights to review. The airports are numbered $1, \ldots, N$ and Stockholm has number 1.
The next R lines describe the R flights to review. Each line contains 3 space separated integers a, b, c ($1 \leq a, b \leq N$, $1 \leq c \leq 10000$), where a, b denote 2 distinct airports and c is the cost of the flight in Swedish kronor in both directions. No pair of 2 cities is listed twice.
The next line contains an integer F, ($0 \leq F \leq 200$), the number of additional flights available.
The next F lines contain descriptions of flights in the same format as above and there may be more flights between a pair of cities. You may assume that it is possible to make all the reviews using some of these additional flights.
The input will end with a line containing '0 0'.

The input must be read from standard input.

Output

For each test case, output a single integer, the lowest total cost of flight tickets, such that Frida can make all the reviews and return back to Stockholm.

The output must be written to standard output.

Sample input	**Sample output**
5 3 1 2 1000 2 3 1000 4 5 500 2 1 4 300 3 5 300 6 5 1 2 1000 2 3 1000 1 3 1000 2 4 1000 5 6 500 2 2 5 300 4 6 300 0 0	3100 5100

H - From Dusk Till Down

Source file name: `dusk.c, dusk.cpp` *or* `dusk.java`

Vladimir has white skin, very long teeth and is 600 years old, but this is no problem because Vladimir is a vampire. Vladimir has never had any problems with being a vampire. In fact, he is a very successful doctor who always takes the night shift and so has made many friends among his colleagues. He has a very impressive trick which he shows at dinner partys: He can tell tell blood group by taste.
Vladimir loves to travel, but being a vampire he has to overcome three problems.

- First, he can only travel by train because he has to take his coffin with him. (On the up side he can always travel first class because he has invested a lot of money in long term stocks.)
- Second, he can only travel from dusk till dawn, namely from 6 pm to 6 am. During the day he has to stay inside a train station.
- Third, he has to take something to eat with him. He needs one litre of blood per day, which he drinks at noon (12:00) inside his coffin.

You should help Vladimir to find the shortest route between two given cities, so that he can travel with the minimum amount of blood. (If he takes too much with him, people will ask funny questions like ‘‘What do you do with all that blood?’’)

Input

The first line of the input will contain a single number telling you the number of test cases. Each test case specification begins with a single number telling you how many route specifications follow. Each route specification consists of the names of two cities, the departure time from city one and the total travelling time. The times are in hours.
Note that Vladimir can’t use routes departing earlier than 18:00 or arriving later than 6:00. There will be at most 100 cities and less than 1000 connections. No route takes less than one hour and more than 24 hours. (Note that Vladimir can use only routes with a maximum of 12 hours travel time (from dusk till dawn).) All city names are shorter than 32 characters.
The last line contains two city names. The first is Vladimir’s start city, the second is Vladimir’s destination.

The input must be read from standard input.

Output

For each test case you should output the number of the test case followed by

```
Vladimir needs # litre(s) of blood.
```

or

`There is no route Vladimir can take.`

The output must be written to standard output.

Sample input	**Sample output**
2 3 Ulm Muenchen 17 2 Ulm Muenchen 19 12 Ulm Muenchen 5 2 Ulm Muenchen 10 Lugoj Sibiu 12 6 Lugoj Sibiu 18 6 Lugoj Sibiu 24 5 Lugoj Medias 22 8 Lugoj Medias 18 8 Lugoj Reghin 17 4 Sibiu Reghin 19 9 Sibiu Medias 20 3 Reghin Medias 20 4 Reghin Bacau 24 6 Lugoj Bacau	Test Case 1. There is no route Vladimir can take. Test Case 2. Vladimir needs 2 litre(s) of blood.