118 Mutant Flatworld Explorers

Robotics, robot motion planning, and machine learning are areas that cross the boundaries of many of the subdisciplines that comprise Computer Science: artificial intelligence, algorithms and complexity, electrical and mechanical engineering to name a few. In addition, robots as “turtles” (inspired by work by Papert, Abelson, and diSessa) and as “beeper-pickers” (inspired by work by Pattis) have been studied and used by students as an introduction to programming for many years.

This problem involves determining the position of a robot exploring a pre-Columbian flat world.

Given the dimensions of a rectangular grid and a sequence of robot positions and instructions, you are to write a program that determines for each sequence of robot positions and instructions the final position of the robot.

A robot *position* consists of a grid coordinate (a pair of integers: x-coordinate followed by y-coordinate) and an orientation (`N`,`S`,`E`,`W` for north, south, east, and west). A robot *instruction* is a string of the letters ‘`L`’, ‘`R`’, and ‘`F`’ which represent, respectively, the instructions:

- *Left*: the robot turns left 90 degrees and remains on the current grid point.
- *Right*: the robot turns right 90 degrees and remains on the current grid point.
- *Forward*: the robot moves forward one grid point in the direction of the current orientation and mantains the same orientation.

The direction *North* corresponds to the direction from grid point (x, y) to grid point $(x, y + 1)$.

Since the grid is rectangular and bounded, a robot that moves “off” an edge of the grid is lost forever. However, lost robots leave a robot “scent” that prohibits future robots from dropping off the world at the same grid point. The scent is left at the last grid position the robot occupied before disappearing over the edge. An instruction to move “off” the world from a grid point from which a robot has been previously lost is simply ignored by the current robot.

Input

The first line of input is the upper-right coordinates of the rectangular world, the lower-left coordinates are assumed to be 0,0.

The remaining input consists of a sequence of robot positions and instructions (two lines per robot). A position consists of two integers specifying the initial coordinates of the robot and an orientation (`N`,`S`,`E`,`W`), all separated by white space on one line. A robot instruction is a string of the letters ‘`L`’, ‘`R`’, and ‘`F`’ on one line.

Each robot is processed sequentially, i.e., finishes executing the robot instructions before the next robot begins execution.

Input is terminated by end-of-file.

You may assume that all initial robot positions are within the bounds of the specified grid. The maximum value for any coordinate is 50. All instruction strings will be less than 100 characters in length.

Output

For each robot position/instruction in the input, the output should indicate the final grid position and orientation of the robot. If a robot falls off the edge of the grid the word ‘`LOST`’ should be printed after the position and orientation.

Sample Input

```
5 3
1 1 E
RFRFRFRF
3 2 N
FRRFLLFFRRFLL
0 3 W
LLFFFLFLFL
```

Sample Output

```
1 1 E
3 3 N LOST
2 3 S
```

UVa Online Judge

280 Vertex

Write a program that searches a directed graph for vertices which are inaccessible from a given starting vertex.

A directed graph is represented by n vertices where $1 \leq n \leq 100$, numbered consecutively $1 \ldots n$, and a series of edges p `->` q which connect the pair of nodes p and q in one direction only.

A vertex r is reachable from a vertex p if there is an edge p `->` r, or if there exists some vertex q for which q is reachable from p and r is reachable from q.

A vertex r is inaccessible from a vertex p if r is not reachable from p.

Input

The input data for this program consists of several directed graphs and starting nodes.

For each graph, there is first one line containing a single integer n. This is the number of vertices in the graph.

Following, there will be a group of lines, each containing a set of integers. The group is terminated by a line which contains only the integer '`0`'. Each set represent a collection of edges. The first integer in the set, i, is the starting vertex, while the next group of integers, $j \ldots k$, define the series of edges i `->` $j \ldots i$ `->` k, and the last integer on the line is always '`0`'. Each possible start vertex i, $1 \leq i \leq n$ will appear once or not at all. Following each graph definition, there will be one line containing a list of integers. The first integer on the line will specify how many integers follow. Each of the following integers represents a start vertex to be investigated by your program. The next graph then follows. If there are no more graphs, the next line of the file will contain only the integer '`0`'.

Output

For each start vertex to be investigated, your program should identify all the vertices which are inaccessible from the given start vertex. Each list should appear on one line, beginning with the count of inaccessible vertices and followed by the inaccessible vertex numbers.

Sample Input

```
3
1 2 0
2 2 0
3 1 2 0
0
2 1 2
0
```

Sample Output

```
2 1 3
2 1 3
```

1057 Routing

As more and more transactions between companies and people are being carried out electronically over the Internet, secure communications have become an important concern. The Internet Cryptographic Protocol Company (ICPC) specializes in secure business-to-business transactions carried out over a network. The system developed by ICPC is peculiar in the way it is deployed in the network.

A network like the Internet can be modeled as a directed graph: nodes represent machines or routers, and edges correspond to direct connections, where data can be transmitted along the direction of an edge. For two nodes to communicate, they have to transmit their data along directed paths from the first node to the second, and from the second node to the first.

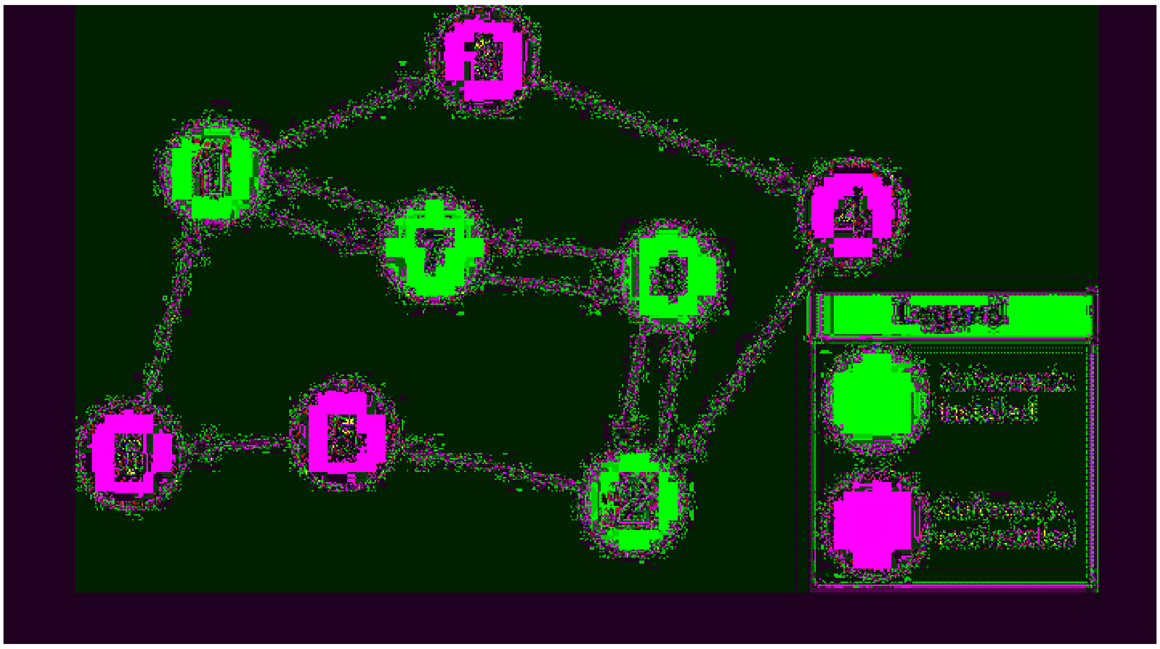


Figure: An arrow from node X to node Y means that it is possible to connect to node Y from node X but not vice versa. If software is installed on nodes 1, 2, 7, and 8, then communication is possible between node 1 and node 2. Other configurations are also possible but this is the minimum cost option. This figure corresponds to the first sample input.

To perform a secure transaction, ICPC's system requires the installation of their software not only on the two endnodes that want to communicate, but also on all intermediate nodes on the two paths connecting the end-nodes. Since ICPC charges customers according to how many copies of their software have to be installed, it would be interesting to have a program that for any network and end-node pair finds the cheapest way to connect the nodes.

Input

The input consists of several descriptions of networks. The first line of each description contains two integers N and M ($2 \leq N \leq 100$), the number of nodes and edges in the network, respectively. The nodes in the network are labeled $1, 2, \ldots, N$, where nodes 1 and 2 are the ones that want to communicate. The first line of the description is followed by M lines containing two integers X and Y ($1 \leq X, Y \leq N$), denoting that there is a directed edge from X to Y in the network.

The last description is followed by a line containing two zeroes.

Output

For each network description in the input, display its number in the sequence of descriptions. Then display the minimum number of nodes on which the software has to be installed, such that there is a directed path from node 1 to node 2 using only the nodes with the software, and also a path from node 2 to node 1 with the same property. (Note that a node can be on both paths but a path need not contain all the nodes.) The count should include nodes 1 and 2.

If node 1 and 2 cannot communicate, display ‘IMPOSSIBLE’ instead.

Follow the format in the sample given below, and display a blank line after each test case.

Sample Input

```
8 12
1 3
3 4
4 2
2 5
5 6
6 1
1 7
7 1
8 7
7 8
8 2
2 8
2 1
1 2
0 0
```

Sample Output

```
Network 1
Minimum number of nodes = 4

Network 2
IMPOSSIBLE
```

10687 Monitoring the Amazon

A network of autonomous, battery-powered, data acquisition stations has been installed to monitor the climate in the region of Amazon. An order-dispatch station can initiate transmission of instructions to the control stations so that they change their current parameters. To avoid overloading the battery, each station (including the order-dispatch station) can only transmit to two other stations. The destinataries of a station are the two closest stations. In case of draw, the first criterion is to chose the westernmost (leftmost on the map), and the second criterion is to chose the southernmost (lowest on the map).

You are commissioned by Amazon State Government to write a program that decides if, given the localization of each station, messages can reach all stations.

Input

The input consists of an integer N, followed by N pairs of integers Xi, Yi, indicating the localization coordinates of each station. The first pair of coordinates determines the position of the order-dispatch station, while the remaining $N-1$ pairs are the coordinates of the other stations. The following constraints are imposed: $-20 \leq Xi, Yi \leq 20$, and $1 \leq N \leq 1000$. The input is terminated with $N = 0$.

Output

For each given expression, the output will echo a line with the indicating if all stations can be reached or not (see sample output for the exact format).

Sample Input

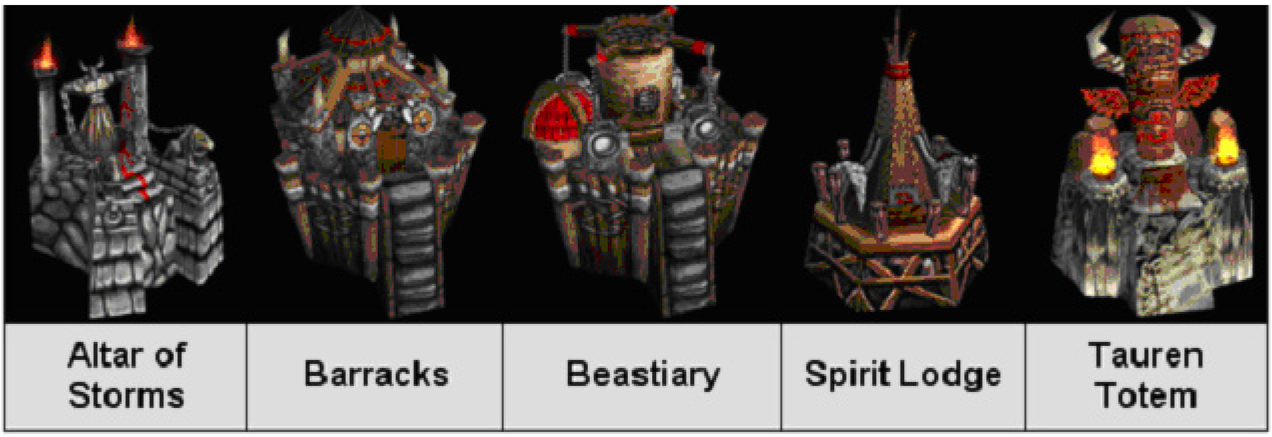
```
4
1 0 0 1 -1 0 0 -1
8
1 0 1 1 0 1 -1 1 -1 0 -1 -1 0 -1 1 -1
6
0 3 0 4 1 3 -1 3 -1 -4 -2 -5
0
```

Sample Output

```
All stations are reachable.
All stations are reachable.
There are stations that are unreachable.
```

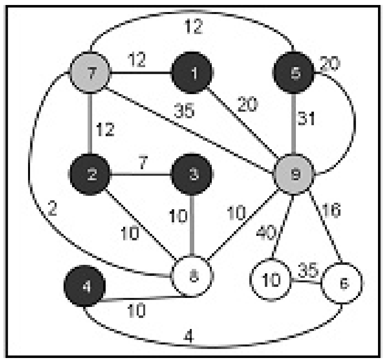
10793 The Orc Attack

Gar'dal Grimsight, the hero of the Orc horde, is planning a massive attack on the human stronghold. But to overcome the stiff resistance of the humans, Grimsight would need an army of mammoth size. Realizing this, he wastes no time – all his five buildings start to produce units on his command. The Altar of Storms, the Barracks, the Beastiary, the Spirit Lodge and the Tauren Totem – these are his five unit producing buildings. These buildings are strategically placed in different locations but can have a rally point where the units are sent as soon as they are produced. This gives Grimsight a good coverage of the territory as well as the flexibility of unit movement.



Now before Grimsight wages the war, the units produced need to meet at this rally point in the map. This point should be chosen in a way that the *shortest distance* from the five unit producing buildings to this location is equal. If there are multiple such spots, the units would gather in a place from where the *shortest distance to the farthest point* in the map is the minimum. Your task is to help the belligerent hero in determining this minimum shortest distance of the farthest point.

However, a small problem remains. The shortest distance between two points in the map is not necessarily the straight line distance between them. Trees, hills and water sheds can act as obstacle for the free movement of the units. So the units may need to go through different locations to get to the point where they are intended to go. If the distances between all these locations in the map are given, then it is possible to compute the shortest distance traveled by the units. The rally point is one such location in the map that we need to find. To keep it simple, we can treat the buildings as locations in the map as well. You may be relieved to know that in Grimsight's world all the units move from one location to another in the shortest possible ways only.



Here is a sample scenario. The locations 1 through 5 are the five unit producing buildings. Apart from those, there are five more locations in the map. As we can see, the locations 9 and 7 are equidistant from the five unit producing buildings. So they are the candidates for the rally point. But we would choose location 9 over location 7 as the farthest location from 7 has a cost of 51 whereas the farthest location from 9 would incur a cost of 40. So the cost that Grimsight wants to measure is this distance cost 40.

Please note that the sample input/output section does not include the example shown here.

Input

There can be multiple test cases. The first line of input gives you the number of test cases, T ($1 \leq T \leq 20$). Then T test cases follow. The first line of each input gives you the number of locations, L ($5 < L \leq 100$), followed by the number of distance information between the locations, D ($1 \leq D \leq 1000$). Each of the next D lines would contain three integers U ($1 \leq U \leq L$), V ($1 \leq V \leq L$) and C ($1 \leq C \leq 1000$) indicating a distance of cost C between the locations U and V. There may be multiple occurrences of the same U, V pair in the input – it is up to you to decide which one to use. You should remember that a unit does not need to cover any distance if it remains static. We would consider the locations 1 through 5 to be the five buildings of our interest.

Output

For each of the test case, you need to print one line of output. The output for each test case should start with the serial number of the map, followed by the minimum possible distance from the farthest point. This distance, as we have already mentioned, has to be measured from the rally point which is equidistant from the five buildings. It is implied that all the locations in the map should be reachable from this point. When such a point does not exist you should print a '`-1`' instead.

Sample Input

```
2
7 11
1 7 2
2 7 2
3 7 2
5 7 2
6 7 1
1 6 1
2 6 1
3 6 1
4 6 1
5 6 1
7 6 1
6 1
1 2 3
```

Sample Output

```
Map 1: 1
Map 2: -1
```

11906 Knight in War Grid

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 (≤ 50) denoting the number of cases.

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

Output

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

Sample Input

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

Sample Output

```
Case 1: 8 0
Case 2: 4 10
```