Problem A
Low Cost Air Travel
Input File: air.in

Air fares are crazy! The cost of a ticket is determined by numerous factors, and is usually not directly related to the distance traveled. Many travelers try to be creative, sometimes using only parts of tickets with stops in various cities to achieve lower-cost travel. However, the airlines are aware of this behavior, and usually require that the travel covered by a ticket be completed in order and without intervening travel. For example, if you have a ticket for travel from City-1 to City-2 then to City-3, you are not allowed to use only the portion of the ticket for travel from City-2 to City-3. You must always start at the first city on the ticket. In addition, you are not allowed to travel from City-1 to City-2, fly elsewhere and return, and then continue your journey from City-2 to City-3.

Let's consider an example. Suppose you are allowed to purchase three types of tickets:

Ticket #1:	City-1 to City-3 to City-4	$225.00
Ticket #2:	City-1 to City-2	$200.00
Ticket #3:	City-2 to City-3	$50.00

Suppose you wanted to travel from City-1 to City-3. There are two ways to get there using only the available ticket choices:

Purchase Ticket #1 for $225.00 and use only the first leg of the ticket.
Purchase Ticket #2 for $200.00 and Ticket #3 for $50.

The first choice is the cheapest.

Given a set of airline ticket offers, and one or more trip itineraries, you must determine how to purchase tickets in order to minimize the cost of travel. Each trip will be possible.

Input

Input consists of multiple test cases, each describing a set of ticket offers and a set of trip itineraries.

Each case begins with a line containing *NT*, the number of ticket offers, followed by *NT* offer descriptions, one to a line. Each description consists of a positive integer specifying the price of the ticket, the number of cities in the ticket's route, and then that many cities. Each city in a case has an arbitrary, but unique, integer identification number. Note that several tickets may be purchased from the same offer.

The next line contains *NI*, the number of trips that are to have their cost minimized. *NI* lines follow, giving the itineraries for each trip. Each line consists of the number of cities in the itinerary (including the starting city), followed by that many city identification numbers, given in the order they are to be visited.

There will be no more than 20 ticket offers or 20 itineraries in a test case. Each offer and itinerary lists from 2 to 10 cities. No ticket price exceeds $10,000. Adjacent cities in a route or itinerary will be distinct. Tickets and trips are numbered sequentially in each set, starting with 1.

The last case is followed by a line containing a zero.

Output

For each trip, output two lines containing the case number, the trip number, the minimum cost of the trip, and the numbers of the tickets used for the trip, in the order they will be used. Follow the output format shown below. The output will always be unique.

Sample Input	Output for the Sample Input

3
225 3 1 3 4
200 2 1 2
50 2 2 3
1
2 1 3
3
100 2 2 4
100 3 1 4 3
200 3 1 2 3
2
3 1 4 3
3 1 2 4
0
	Case 1, Trip 1: Cost = 225
	  Tickets used: 1
	Case 2, Trip 1: Cost = 100
	  Tickets used: 2
	Case 2, Trip 2: Cost = 300
	  Tickets used: 3 1

Problem B

Remember the A La Mode!

Input File: alamode.in

Hugh Samston owns the "You Want It, Hugh Got It" catering service, which has been asked to supply desserts for the participants in this year's ICPC World Finals. Hugh will provide pie slices topped with ice cream at the various social functions scheduled throughout the week. As with any other dedicated entrepreneur, Hugh would like to offer the best service possible, so he has ordered a wide variety of pies and ice creams to satisfy even the most eclectic tastes.

Hugh plans to serve each pie slice with a single scoop of ice cream, leaving the exact combination up to the whim of the customer. But of course, as with any other dedicated entrepreneur, Hugh would also like to make as much profit as possible from this enterprise. He knows ahead of time how much profit he can make on each combination of pie slice and ice cream scoop, as well as which combinations of pie and ice cream should never be put together (example: Peppermint Banana Chunk ice cream on Key Lime pie).

Given this information, along with the number of slices and scoops he has of each variety of pie and ice cream, Hugh figures he can determine both the minimum and maximum profits he can expect. Since he hopes to be the caterer at subsequent World Finals, he would like a general program to solve this and future problems.

Input

Input will consist of multiple problem instances. Each problem instance will start with a line containing two integers P ($P \leq 50$) and I ($I \leq 50$), indicating the number of types of pie and ice cream, respectively. The next line will contain P integers indicating the number of slices available for each of the pie types. The line after that will contain I integers indicating the number of scoops available for each of the ice cream types. The total number of pie slices will always equal the total number of ice cream scoops available, and it is assumed that all pie slices and ice cream scoops will be used.

Each problem instance will end with P lines each containing I floating point numbers indicating the profit for each pie/ice cream combination: the first value indicates the profit if a slice of pie type 0 is topped with a scoop of ice cream type 0; the next value indicates the profit if a slice of pie type 0 is topped with a scoop of ice cream type 1, and so on. A profit value of -1 indicates that no combinations of that pie type and ice cream type should ever be sold. All other integers (number of slices for each type of pie and number of scoops for each type of ice cream) will be less than or equal to 100 and the profit on each one of the pie/ice cream combinations (other than -1) will be larger than 0 and less than or equal to 10, with at most two digits after the decimal point.

The last problem instance is followed by a line containing two zeroes.

Output

For each problem instance, output the problem number (starting at 1) followed by the minimum and maximum profits, using the format shown in the sample output. Display all numbers with two fractional digits. All problem instances are guaranteed to have at least one solution using all of the pie slices and ice cream scoops.

Sample Input	Output for the Sample Input
2 3 40 50 27 30 33 1.11 1.27 0.70 -1 2 0.34 4 4 10 10 10 10 10 10 10 10 1.01 -1 -1 -1 -1 1.01 -1 -1 -1 -1 1.01 -1 -1 -1 -1 1.01 0 0	Problem 1: 91.70 to 105.87 Problem 2: 40.40 to 40.40

Problem C
Ars Longa
Input File: ars.in

You have been struck with inspiration, and are designing a beautiful new art sculpture for the foyer of your local museum. For highly important artistic reasons, you are designing it using very specific materials. However, you are not sure if physics is on your side. Will your sculpture actually stand up?

The sculpture will be composed of various ball joints weighing 1 kilogram each, and various rods (of negligible weight) connecting the joints. Rods cannot be stretched or compressed, and they can never detach from a joint. However, they are free to rotate around the joints in any direction. The joints that lie on the ground are glued in place; all others are free to move. For simplicity, you may ignore the effects of intersections of rods; each rod exerts force only on the 2 joints connected to it. Also, any joint that is in the air will have at least one rod coming out that is not parallel to the ground. This prevents the degenerate case where a ball is supported only horizontally by a rigid structure. In real life, it would sag just a little.

Write a program to determine whether your structure is static (that is, will not immediately move from the effects of gravity). Note that each rod can transmit an arbitrarily large tensional force along its length, and that "being static" means that the tensional forces at each joint balance the weight of the joint.

If the structure is static, you must also determine whether it is stable (that is, will not move if perturbed slightly by pulling its joints).

Input

The input contains several sculpture descriptions. Every description begins with a line containing integers J and R, the number of joints and rods in the structure, respectively. Joints are numbered from 1 to J. The description continues with J lines, one per joint, each containing 3 floating point numbers giving the x, y, z coordinates of that joint. Following are R lines, one per rod, with 2 distinct integers indicating the joints connected by that rod.

Each rod is exactly the right length to connect its joints. The z coordinates will always be non-negative; a z coordinate of 0 indicates that the joint is on the ground and fixed in place. There are at most 100 joints and 100 rods.

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

Output

For each sculpture description, output `NON-STATIC`, `UNSTABLE`, or `STABLE`, as shown in the sample output.

Sample Input	Output for the Sample Input
4 3 0 0 0 -1.0 -0.5 1.0 1.0 -0.5 1.0 0 1.0 1.0 1 2 1 3 1 4 4 6 0 0 0 -1.0 -0.5 1.0 1.0 -0.5 1.0 0 1.0 1.0 1 2 1 3 1 4 2 3 2 4 3 4 7 9 0 0 0 -1.0 -0.5 1.0 1.0 -0.5 1.0 0 1.0 1.0 -1.0 -0.5 0 1.0 0.5 0 0 1.0 0 1 2 1 3 1 4 2 3 2 4 3 4 2 5 3 6 4 7 0 0	Sculpture 1: NON-STATIC Sculpture 2: UNSTABLE Sculpture 3: STABLE

Problem D
Bipartite Numbers
Input File: bipartite.in

The executive officers of the company where you work want to send each other encrypted messages. Rather than use off-the-shelf encryption software such as PGP, they have tasked the IT staff with handling the encryption problem. The IT staff decided on a solution that requires "public" and "private" integer keys. The idea is that everyone can see your public key, but only you know your private key.

Your best friend in the company is a wonderful person but a not-so-wonderful programmer. He has created a public-private key scheme as follows. A public key can be any positive integer. The corresponding private key is the smallest bipartite number that is greater than and a multiple of the public key.

A bipartite number is any positive integer that contains exactly 2 distinct decimal digits *s* and *t* such that *s* is not 0 and all occurrences of *s* precede all occurrences of *t*. For example 44444411 is bipartite (*s* is 4 and *t* is 1), So are 41, 10000000, and 5555556. However, neither 4444114 nor 44444 are bipartite.

Notice that the large bipartite number 88888888888800000 can be nicely described as 12 8's followed by 5 0's. You can express any bipartite number using four numbers: *m s n t*. The numbers *s* and *t* are the leading and trailing digits as described above, *m* is the number of times the digit *s* appears in the bipartite number, and *n* is the number of times the digit *t* appears.

The trouble with your friend's scheme is that it is not too difficult to compute a private key if you know the public key. You need to convince your friend that his public-private key scheme is inadequate before he loses his job over his bad decision! You must write a program that takes public keys as input and displays the corresponding private keys.

Input
The input consists of several test cases. Each test case is on a separate line, and it consists of a single public key in the range 1 ... 99999.

The last case is followed by a line containing the integer zero.

Output
For each test case, display a line consisting of the public key, a colon, then 4 integers *m s n t* where *m*, *n*, *s*, and *t* are as described above.

Sample Input	Output for the Sample Input
125 17502 2005 0	125: 1 5 2 0 17502: 4 7 4 8 2005: 3 2 3 5

Problem E
Bit Compressor
Input File: bits.in

The aim of data compression is to reduce redundancy in stored or communicated data. This increases effective data density and speeds up data transfer rates. One possible method to compress any binary message is the following:

> Replace any maximal sequence of *n* 1's with the binary version of *n* whenever it shortens the length of the message.

For example, the compressed form of the data **11111111**00100**111111111111111**0011 becomes **1000**00100**1111**0011. The original data is 32 bits long while the compressed data is only 17 bits long.

The drawback of this method is that sometimes the decompression process yields more than one result for the original message, making it impossible to obtain the original message. Write a program that determines if the original message can be obtained from the compressed data when the length of the original message (*L*), the number of 1's in the original message (*N*) and the compressed data are given. The original message will be no longer than 16 Kbytes and the compressed data will be no longer than 40 bits.

Input

The input file contains several test cases. Each test case has two lines. The first line contains *L* and *N* and the second line contains the compressed data.

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

Output

For each test case, output a line containing the case number (starting with 1) and a message `YES`, `NO` or `NOT UNIQUE`. `YES` means that the original message can be obtained. `NO` means that the compressed data has been corrupted and the original message cannot be obtained. `NOT UNIQUE` means that more than one message could have been the original message. Follow the format shown in the sample output.

Sample Input	Output for the Sample Input
32 26 10000010011110011 9 7 1010101 14 14 111111 0 0	Case #1: YES Case #2: NOT UNIQUE Case #3: NO

Problem F
Building a Clock
Input File: clock.in

In Old Town Square in the city of Prague, there is a beautiful Astronomical Clock, constructed in the year 1410. For centuries, the clockmaker's art consisted of using gears to connect a shaft, turning at a known rate, to other shafts until, by the proper combination of gears, two shafts could be made to turn at the correct rates to represent minutes and hours.

You must write a program that, given an input shaft speed and a collection of gears, computes how the gears can be connected to create a clock with an hour hand and a minute hand. You may use as many shafts as you like, but each shaft may have a maximum of three gears. All the gears on a shaft turn at the same rate. If a gear having *T1* teeth turning at a rate *R1* is engaged with another gear having *T2* teeth, the turning rate of the second gear is $-R1(T1/T2)$. Your solution must include two shafts, a minute shaft that turns clockwise at the rate of one revolution per hour, and an hour shaft that turns clockwise at the rate of one revolution per twelve hours. Your solution is not required to use all the available gears.

Input

The input consists of several trials, each described by one line of input. Each input line begins with an integer N ($3 \le N \le 6$), the number of gears available for building a clock. N is followed by another integer R ($-3600 \le R \le 3600$, $R \ne 0$), the turning rate of the input shaft, which is the number of revolutions made by the shaft in 24 hours. (A positive number represents clockwise rotation, and a negative number represents counter-clockwise rotation.) R is followed by N gear descriptions. Each gear description is a pair: a one-character name that identifies the gear, followed by an integer T ($6 \le T \le 120$), that is the number of teeth on the gear. The names and numbers on each input line are separated by spaces, as shown in the sample input.

The last trial is followed by a line containing a single zero.

Output

For each trial, print a line containing the trial number, as shown in the sample output. If it is possible to construct a clock using the given set of gears, the line containing the trial number must be followed by two more output lines, one for the minute hand and one for the hour hand. Otherwise, the line containing the trial number must end with the words `IS IMPOSSIBLE` as shown in the sample output.

The line for the minute hand starts with `Minutes:` followed by a plan that shows how the input shaft is connected by a sequence of gears to the minute shaft. The plan consists of a sequence of shafts, separated by hyphens. Each shaft is represented by one or two characters. The first character is the name of the driven gear—the gear on the shaft that is engaged with a gear on the previous shaft. For the input shaft, use an asterisk (*) to represent the absence of a driven gear. The second character describing a shaft is the name of the driving gear—the gear on the shaft that is engaged with a gear on the next shaft. The driven gear and the driving gear can be the same gear, in which case the shaft is described by a single character which is the name of this gear. The last shaft in the plan is the minute shaft, described by a single letter which is the name of its driven gear.

The line for the hour hand starts with `Hours:` followed by a plan for connecting the input shaft to the hour shaft. Use the same format as the minute plan.

Each gear may occur only once in the clock. The minute plan and the hour plan may have an initial part in common, however. A gear in a common initial part will occur both in the minute plan and the hour plan. For the same reason, a given shaft can be used in both the hour plan and the minute plan. If a shaft is used in both plans, it may or may not have the same description in both plans. For example, a shaft containing a single gear named A will be represented as A in both plans. On the other hand, a shaft containing three gears named A, B, and C might be represented as AB in the minute plan (if B is the driving gear in that plan) and as AC in the hour plan (if C is the driving gear in that plan).

The following lines represent valid output lines:

`Hours:  *A-BC-D`	An input shaft having one gear, engaged with an intermediate shaft having two gears, engaged with an hour shaft having one gear.
`Minutes: *A-B-C`	An input shaft having one gear, engaged with an intermediate shaft having one gear, engaged with a minute shaft having one gear.
`Minutes: *`	A plan in which no gears are needed because the input shaft is turning at the correct rate for the minute shaft.

If there are multiple ways to build a clock using the given gears, print the solution that uses the minimum number of shafts. In case of a tie for the minimum number of shafts, print the solution that uses the minimum number of gears. In case of a tie for both the minimum numbers of shafts and gears, print the solution whose string description is alphabetically first. The string description of a solution is its minute plan, followed by its hour plan, concatenated together with asterisks and hyphens removed. For example, a solution in which the minute plan is `*A-B` and the hour plan is `*A-BC-D-E` has the string description `ABABCDE`.

Print one blank line between trials.

Sample Input	Output for the Sample Input

`6 40 P 7 Q 84 R 50 A 40 B 30 C 14`
`6 40 P 7 Q 84 R 45 A 40 B 30 C 14`
`0`
	`Trial 1`
	`Minutes: *B-A-R`
	`Hours:  *B-A-RP-C-Q`
	
	`Trial 2 IS IMPOSSIBLE`

Problem G
Pilgrimage
Input File: pilgrimage.in

Jack is making a long distance walk with some friends along the old pilgrim road from Vézelay to Santiago de Compostela. Jack administers money for the group. His administration is quite simple. Whenever an amount (€ 60, say) has to be paid for the common good he will pay it, and write in his booklet: `PAY 60`.

When needed, Jack will ask every member of the group, including himself, to pay an amount (€ 50, say) to the collective purse, and write in his booklet: `COLLECT 50`. If the group size is 7, he collects € 350 in total.

Unfortunately some of the group members cannot participate in the full walk. So sometimes the group will grow, sometimes it will shrink. How does Jack handle these comings and goings of group members in terms of collective money? Suppose, for example, the group size is 7, and that Jack has € 140 in cash, which is € 20 for every group member. If two group members leave, each will receive € 20, and Jack will write in his booklet: `OUT 2.` If under the same circumstances three new group members arrive, they will each have to pay € 20, and Jack will write: `IN 3`.

In these cases the amount in cash could easily be divided, without fractions. As a strange coincidence, this happened during the whole trip. Jack never had to make calculations with fractional numbers of euros.

Near the end of the trip, Jack was joined by all his fellow travelers. Nobody was willing to miss the glorious finale of the trip. It was then that Jack tried to remember what the group size had been during each part of the trip. He could not remember.

Given a page of Jack's booklet, could you figure out the size of the group at the beginning of that page?

Input

The input file contains several test cases. Each test case is a sequence of lines in Jack's booklet. The first line of each test case will give the number N ($0 < N \leq 50$) of lines to follow. The next N lines have the format: <*keyword*> <*num*>, where

<*keyword*> = `PAY | COLLECT | IN | OUT`

and <*num*> is a positive integer, with the following restrictions:

`IN` k	$k \leq 20$
`OUT` k	$k \leq 20$
`COLLECT` k	$k \leq 200$
`PAY` k	$k \leq 2000$

The last case is followed by a line containing a single zero.

Output

For each test case, print a single line describing the size of the group at the beginning of the part of the trip described in the test case. This line contains:

- The word `IMPOSSIBLE`, if the data are inconsistent.
- A single number giving the size of the group just prior to the sequence of lines in Jack's booklet, if this size is uniquely determined by the data.
- Several numbers, in increasing order, separated by spaces, giving the possible sizes of the group, in case the number of solutions is finite, but the solution is not unique.

- A statement in the format: `SIZE >=` *N*, giving a lower bound for the size of the group, in case the number of solutions is infinite. Observe that the inequality `SIZE >= 1` always applies, since at least Jack himself did the whole trip.

Sample Input	Output for the Sample Input
5 IN 1 PAY 7 IN 1 PAY 7 IN 1 7 IN 1 COLLECT 20 PAY 30 PAY 12 IN 2 PAY 30 OUT 3 3 IN 1 PAY 8 OUT 3 1 OUT 5 0	IMPOSSIBLE 2 3 7 SIZE >= 6

Problem H
Pockets
Input File: pockets.in

Origami, or the art of paper folding, often makes use of "pockets" of paper, particularly in complicated models made from multiple pieces of paper where a tab on one piece of paper needs to fit into a pocket in another piece of paper. In this problem you must count pockets in a flat folded square of paper. A pocket is defined as any opening (lying between two surfaces of paper) that is accessible from the boundary of the folded piece of paper. Note that one accessible opening can account for several pockets since each open side contributes one pocket. Figure 1 shows an example. Observe that the "middle" opening (between the second and third layers of paper) contributes 3 to the total pocket count.

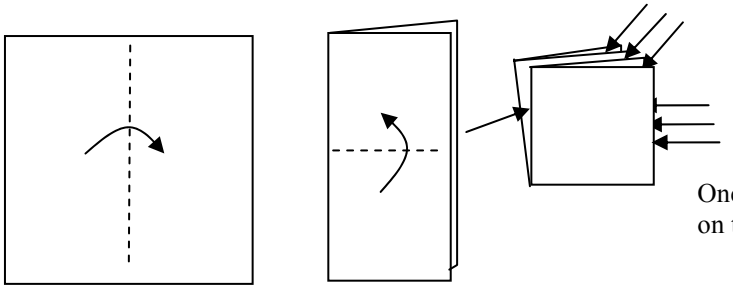


Figure 1: Pockets

Assume the paper is initially lying on a flat surface and is never completely lifted from the surface. All folds will be horizontal or vertical. Fold lines will fall only along equally-spaced crease lines, *N* in each direction. On the original unfolded square, creases and edges are numbered from top to bottom and from left to right as shown in Figure 2. Each fold reduces the boundary of the folded piece of paper to a smaller rectangle; the final fold results in a square one unit in each direction. Folds are described using a crease line and a direction. For instance, `2 U` means to fold the bottom edge up using horizontal crease 2; `1 L` means to fold the right edge to the left using crease 1. (See Figure 2.) After several folds, creases may be aligned (for instance, creases 1 and 3 in Figure 2). Either number may be used to specify a fold along that line (so, in Figure 2, `1 D` and `3 D` are equivalent instructions after the first fold). Pockets are to be counted for the boundary of the final one-unit square. Once a crease is made it cannot be undone. All creases go through every layer of paper from top to bottom; disregard paper thickness.

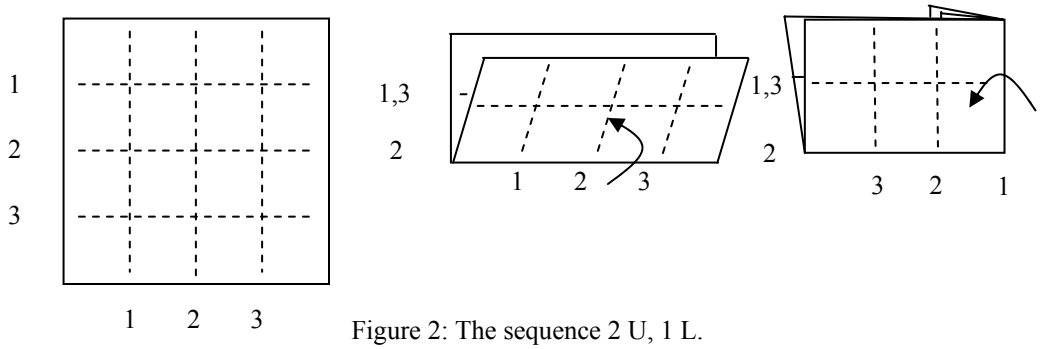


Figure 2: The sequence 2 U, 1 L.

Input

Input is a sequence of test cases. Each test case begins with a line containing two integers, *N* and *K*. *N* is the number of horizontal crease lines (the same as the number of vertical crease lines) of the square. Creases are numbered 1, 2, ..., *N* from top to bottom and from left to right. *K* is the number of folds to be made. *N* and *K* are each less than or equal to 64.

Following *N* and *K* are *K* fold descriptions. Each fold description consists of an integer crease number *C* and a direction, either `U`, `D`, `L`, or `R` (for up, down, left or right) separated by whitespace. Whitespace also precedes and follows each fold description.

The final result for each test case will be a square one unit in size.

The final test case is followed by a line containing two zeroes.

Output

For each input case, display the case number followed by the number of pockets in the final one-unit square. Use the format shown in the sample output.

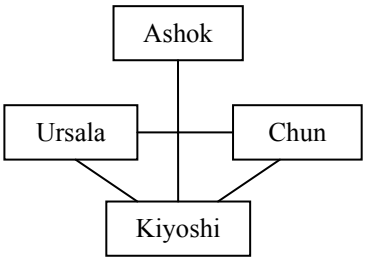
Sample Input	Output for the Sample Input
1 2 1 R 1 U 3 5 2 U 1 L 3 D 3 R 2 L 0 0	Case 1: 7 pockets Case 2: 17 pockets

Problem I
Degrees of Separation
Input File: relatives.in

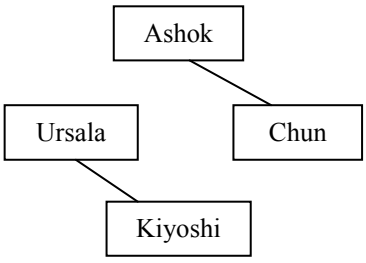
In our increasingly interconnected world, it has been speculated that everyone on Earth is related to everyone else by no more than six degrees of separation. In this problem, you must write a program to find the maximum degree of separation for a network of people.

For any two people, the degree of separation is the minimum number of relationships that must be traversed to connect the two people. For a network, the maximum degree of separation is the largest degree of separation between any two people in the network. If there is a pair of people in the network who are not connected by a chain of relationships, the network is disconnected.

As shown below, a network can be described as a set of symmetric relationships each of which connects two people. A line represents a relationship between two people. Network A illustrates a network with 2 as the maximum degree of separation. Network B is disconnected.



Network A:
Max. degree of separation = 2



Network B:
Disconnected

Input

The input consists of data sets that describe networks of people. For each data set, the first line has two integers: P ($2 \le P \le 50$), the number of people in the network, and R ($R \ge 1$), the number of network relationships. Following that first line are R relationships. Each relationship consists of two strings that are names of people in the network who are related. Names are unique and contain no blank spaces. Because a person may be related to more than one other person, a name may appear multiple times in a data set.

The final test case is followed by a line containing two zeroes.

Output

For each network, display the network number followed by the maximum degree of separation. If the network is disconnected, display `DISCONNECTED`. Display a blank line after the output for each network. Use the format illustrated in the sample output.

Sample Input	Output for the Sample Input
4 4 Ashok Kiyoshi Ursala Chun Ursala Kiyoshi Kiyoshi Chun 4 2 Ashok Chun Ursala Kiyoshi 6 5 Bubba Cooter Ashok Kiyoshi Ursala Chun Ursala Kiyoshi Kiyoshi Chun 0 0	Network 1: 2 Network 2: DISCONNECTED Network 3: DISCONNECTED

Problem J
Routing
Input: routing.in

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.

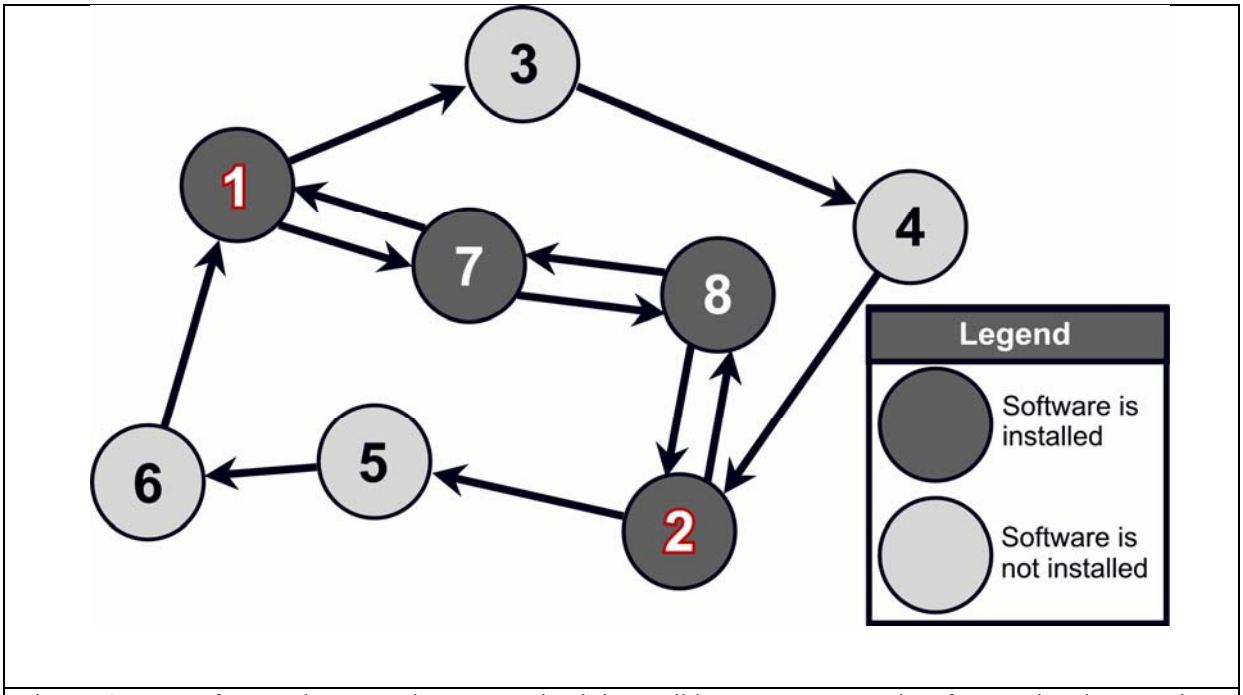


Figure: An arrow from node *X* to node *Y* means that it is possible to connect to node *Y* from node *X* but not vice versa. If software is installed on nodes 1, 2, 7, and 8, then communication is possible between node 1 and node 2. Other configurations are also possible but this is the minimum cost 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 end-nodes 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 ($2 \le N \le 100$), the number of nodes and edges in the network, respectively. The nodes in the network are labeled 1, 2, ..., 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 ($1 \le X, Y \le N$), 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	Output for the 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	Network 1 Minimum number of nodes = 4 Network 2 IMPOSSIBLE