Problem G

Whac-a-Mole

While visiting a traveling fun fair you suddenly have an urge to break the high score in the Whac-a-Mole game. The goal of the Whac-a-Mole game is to... well... whack moles. With a hammer. To make the job easier you have first consulted the fortune teller and now you know the exact appearance patterns of the moles.

The moles appear out of holes occupying the n^2 integer points (x, y) satisfying $0 \leq x, y < n$ in a two-dimensional coordinate system. At each time step, some moles will appear and then disappear again before the next time step. After the moles appear but before they disappear, you are able to move your hammer in a straight line to any position (x_2, y_2) that is at distance at most d from your current position (x_1, y_1). For simplicity, we assume that you can only move your hammer to a point having integer coordinates. A mole is whacked if the center of the hole it appears out of is located on the line between (x_1, y_1) and (x_2, y_2) (including the two endpoints). Every mole whacked earns you a point. When the game starts, before the first time step, you are able to place your hammer anywhere you see fit.

Input specifications

The input consists of several test cases. Each test case starts with a line containing three integers n, d and m, where n and d are as described above, and m is the total number of moles that will appear ($1 \leq n \leq 20$, $1 \leq d \leq 5$, and $1 \leq m \leq 1000$). Then follow m lines, each containing three integers x, y and t giving the position and time of the appearance of a mole ($0 \leq x, y < n$ and $1 \leq t \leq 10$). No two moles will appear at the same place at the same time.

The input is ended with a test case where $n = d = m = 0$. This case should not be processed.

Output specifications

For each test case output a single line containing a single integer, the maximum possible score achievable.

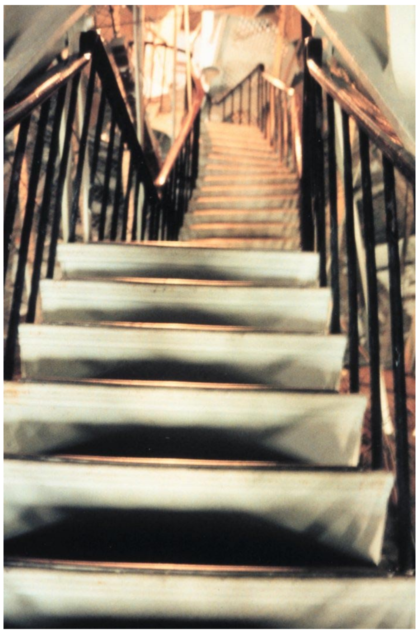
Sample input

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

Output for sample input

```
4
2
```

Problem I: Up the Stairs



John is moving to the penthouse of a tall sky-scraper. He packed all his stuff in boxes and drove them to the entrance of the building on the ground floor. Unfortunately the elevator is out of order, so the boxes have to be moved up the stairs.

Luckily John has a lot of friends that want to help carrying his boxes up. They all walk the stairway at the same speed of 1 floor per minute, regardless of whether they carry a box or not. The stairway however is so narrow that two persons can't pass each other on it. Therefore they deciced to do the following: someone with a box in his hands is always moving up and someone empty-handed is always moving down. When two persons meet each other somewhere on the stairway, the lower one (with a box) hands it over to the higher one (without a box). (And then the lower one walks down again and the higher one walks up.) The box exchange is instantaneous. When someone is back on the ground floor, he picks up a box and starts walking up. When someone is at the penthouse, he drops the box and walks down again.

After a while the persons are scattered across the stairway, some of them with boxes in their hands and some without. There are still a number of boxes on the ground floor and John is wondering how much more time it will take before all the boxes are up. Help him to find out!

Input

One line with a positive number: the number of test cases. Then for each test case:

- One line with three numbers N, F, B with $1 \leq N, F \leq 1,000$ and $1 \leq B \leq 1,000,000$: the number of persons, the number of floors (0=ground floor, F=penthouse) and the number of boxes that are still on the ground floor.

- N lines with two numbers f_i and b_i with $0 \leq f_i \leq F$ and $b_i = 0$ or $b_i = 1$: the floors where the persons are initially and whether or not they have a box in their hands (1=box, 0=no box).

Output

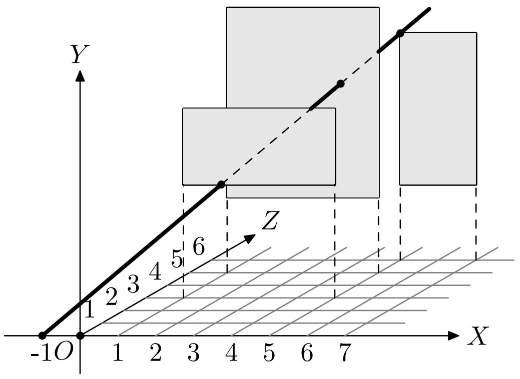
One line with the amount of time (in minutes) it will take to get all the remaining boxes to the penthouse.

Sample input	**Sample output**
`2`	`30`
`3 10 5`	`8`
`0 0`	
`0 0`	
`0 0`	
`2 5 1`	
`2 1`	
`3 0`	

Problem G. Gunman

Input file:	`gunman.in`
Output file:	`gunman.out`

Consider a 3D scene with $OXYZ$ coordinate system. Axis OX points to the right, axis OY points up, and axis OZ points away from you. There is a number of rectangular windows on the scene. The plane of each window is parallel to OXY, its sides are parallel to OX and OY. All windows are situated at different depths on the scene (different coordinates $z > 0$).



A gunman with a rifle moves along OX axis ($y = 0$ and $z = 0$). He can shoot a bullet in a straight line. His goal is to shoot a single bullet through all the windows. Just touching a window edge is enough.

Your task is to determine how to make such shot.

Input

The first line of the input file contains a single integer number n ($2 \le n \le 100$) — the number of windows on the scene. The following n lines describe the windows. Each line contains five integer numbers x_{1i}, y_{1i}, x_{2i}, y_{2i}, z_i ($0 < x_{1i}, y_{1i}, x_{2i}, y_{2i}, z_i < 1000$). Here (x_{1i}, y_{1i}, z_i) are coordinates of the bottom left corner of the window, and (x_{2i}, y_{2i}, z_i) are coordinates of the top right corner of the window ($x_{1i} < x_{2i}$, $y_{1i} < y_{2i}$). Windows are ordered by z coordinate ($z_i > z_{i-1}$ for $2 \le i \le n$).

Output

Output a single word "`UNSOLVABLE`" if the gunman cannot reach the goal of shooting a bullet through all the windows.

Otherwise, on the first line output a word "`SOLUTION`". On the next line output x coordinate of the point from which the gunman must fire a bullet. On the following n lines output x, y, z coordinates of the points where the bullet goes through the consecutive windows. All coordinates in the output file must be printed with six digits after decimal point.

Sample input and output

gunman.in	gunman.out

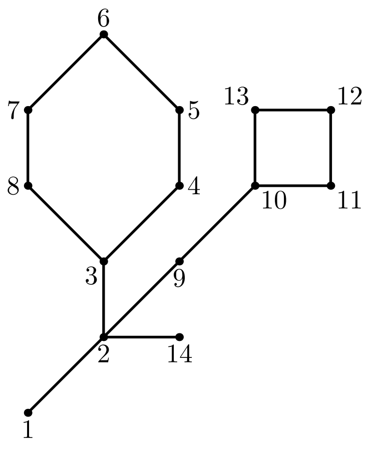
3
1 3 5 5 3
1 2 5 7 5
5 2 7 6 6
	SOLUTION
	-1.000000
	2.000000 3.000000 3.000000
	4.000000 5.000000 5.000000
	5.000000 6.000000 6.000000

3
2 1 5 4 1
3 5 6 8 2
4 3 8 6 4
	UNSOLVABLE

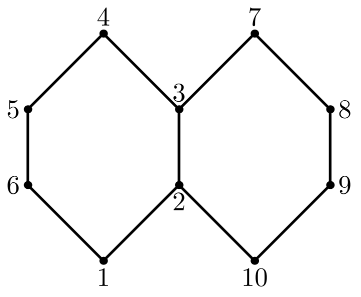
Problem C. Cactus

Input file:	`cactus.in`
Output file:	`cactus.out`

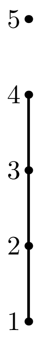
Cactus is a connected undirected graph in which every edge lies on at most one simple cycle. Intuitively cactus is a generalization of a tree where some cycles are allowed. Your task first is to verify if the given graph is a cactus or not. Important difference between a cactus and a tree is that a cactus can have a number of spanning subgraphs that are also cactuses. The number of such subgraphs (including the graph itself) determines *cactusness* of a graph (this number is one for a cactus that is just a tree). The cactusness of a graph that is not a cactus is considered to be zero.



A cactus



Not a cactus



Not a cactus

The first graph on the picture is a cactus with cactusness 35. The second graph is not a cactus because edge $(2, 3)$ lies on two cycles. The third graph is not a cactus because it is not connected.

Input

The first line of the input file contains two integer numbers n and m ($1 \le n \le 20000$, $0 \le m \le 1000$). Here n is the number of vertices in the graph. Vertices are numbered from 1 to n. Edges of the graph are represented by a set of edge-distinct paths, where m is the number of such paths.

Each of the following m lines contains a path in the graph. A path starts with an integer number k_i ($2 \le k_i \le 1000$) followed by k_i integers from 1 to n. These k_i integers represent vertices of a path. Path can go to the same vertex multiple times, but every edge is traversed exactly once in the whole input file. There are no multiedges in the graph (there is at most one edge between any two vertices).

Output

Write to the output file a single integer number — the cactusness of the given graph. Note that cactusness can be quite a large number.

Sample input and output

`cactus.in`	`cactus.out`

`14 3`
`9 1 2 3 4 5 6 7 8 3`
`7 2 9 10 11 12 13 10`
`2 2 14`
	`35`

`10 2`
`7 1 2 3 4 5 6 1`
`6 3 7 8 9 10 2`
	`0`

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

I Tower Parking

There is a new revolution in the parking lot business: the parking tower. The concept is simple: you drive your car into the elevator at the entrance of the tower, and the elevator and conveyor belts drag the car to an empty parking spot, where the car remains until you pick it up. When you return, the elevator and conveyor belts move your car back to the entrance and you're done.

The layout of the tower is simple. There is one central elevator that transports the cars between the different floors. On each floor there is one giant circular conveyor belt on which the cars stand. This belt can move in clockwise and counterclockwise direction. When the elevator arrives on a floor, it becomes part of the belt so that cars can move through it.

At the end of the day the tower is usually packed with cars and a lot of people come to pick them up. Customers are processed in a first come first serve order: the elevator is moved to the floor of the first car, the conveyor belt moves the car on the elevator, the elevator is moved down again, and so on. We like to know how long it takes before the last customer gets his car. Moving the elevator one floor up- or downwards takes 10 seconds and moving a conveyor belt one car in either direction takes 5 seconds.

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line with two integers h and l with $1 \leq h \leq 50$ and $2 \leq l \leq 50$: the height of the parking tower and the length of the conveyor belts.
- h lines with l integers: the initial placement of the cars. The jth number on the ith line describes the jth position on the ith floor. This number is -1 if the position is empty, and r if the position is occupied by the rth car to pick up. The positive numbers form a consecutive sequence from 1 to the number of cars. The entrance is on the first floor and the elevator (which is initially empty) is in the first position. There is at least one car in the parking tower.

Output

Per testcase:

- One line with the number of seconds before the last customer is served.

Sample in- and output

Input	Output

2
1 5
-1 2 1 -1 3
3 6
-1 5 6 -1 -1 3
-1 -1 7 -1 2 9
-1 10 4 1 8 -1
	25
	320

Problem 2: Making Pals

A *palindrome* is a sequence that is the same when read forward or backward. For example, "pop" is a palindrome, as are "Poor Dan is in a droop" (ignoring spaces and case), and "12321".

In this problem, you are to find the "cheapest" way to transform a sequence of decimal digits into a palindrome. There are only two types of modifications you may make to the sequence, but each of these may be repeated as many times as necessary. You may delete a digit from either end of the sequence, or you may add a digit to either end of the sequence. Each of these operations incurs a "cost" of 1. For each input sequence, determine the smallest cost of transforming the sequence into a palindrome, and the length of the resulting palindrome. If two palindromes can be produced with the same cost, the length of the longer palindrome (the one with more digits) is to be reported.

For example, suppose the initial sequence was "911". This can be transformed into a palindrome by deleting the leading "9" (yielding "11") or by adding an additional "9" to the right end of the sequence (yielding "9119"). Since both of these transformations have a cost of 1, and the second transformation yields a longer palindrome, it is this one which would be reported as your result.

Note that the particular palindrome produced by the cheapest sequence of transformations is not necessarily unique, but since you are not required to report the resulting palindrome, any of these will suffice.

Input

There will be multiple cases to consider. Each case has a single line of input that contains one or more decimal digits followed by the end of line. The maximum number of digits in a sequence will be 6. The last case is followed by an empty line (that is, only an end of line).

Output

For each input case, display the case number (1, 2, …), the input sequence, the cost of the cheapest transformation, and the length of the resulting palindrome. Your output should follow the format shown in the examples below.

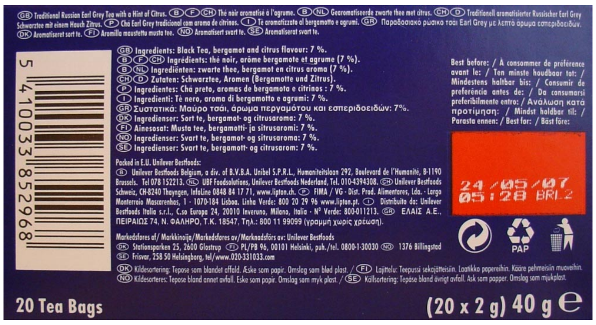
Sample Input

```
911
9118
11234
      <-- This line is blank
```

Expected Output

```
Case 1, sequence = 911, cost = 1, length = 4
Case 2, sequence = 9118, cost = 2, length = 4
Case 3, sequence = 11234, cost = 3, length = 8
```

Problem B: Declaration of Content



Most food you can buy at your local grocery store has a declaration of content. The declaration of content lists the ingredients of the product. It does not necessarily tell you the exact amount of every ingredient, only the ordering of the ingredients, from most common to least common. For some ingredients, an exact percentage might be given, either required by law or because the producer wants you to know how much of the fine expensive ingredients they have used.

Given a set of different products and their respective declarations of content you should determine which contain the most or the least of some given ingredients. For simplicity, we assume in this problem that the percentage of each ingredient always is an integer.

Input

The input consists of several test cases. Each test case consists of two parts.

The first part of a test case begins with an integer P, $1 \leq P \leq 10$, the number of different products in this test case, on a line of its own. Then follows the description of the P products. Each product description consists of a line giving the name of the product, followed by a line containing an integer n, $1 \leq n \leq 100$, giving the number of ingredients in this product. Then follow n lines, the i:th of which contains the name of the i:th most common ingredient of the product. In case of ties, the ingredients will be listed in arbitrary order. Optionally, every ingredient name can be followed by space, an integer p, $0 \leq p \leq 100$ and a percentage sign. If this is present, it specifies the exact amount of this ingredient in the product. Otherwise, because all percentages in this problem are integers, the ingredient makes up at least one percent of the total product.

The second part of a test case begins with an integer Q, $1 \leq Q \leq 100$, the number of queries. Then follow Q lines, each containing a query. A query is of the form "least X", or "most X", where X is the name of an ingredient. In the "most X" case, the ingredient X is guaranteed to be present in at least one of the products.

A name of a product or an ingredient is a string of alphabetic characters (A-Z and a-z), digits (0-9) and underscore. Case is significant. No name will be longer than 30 characters. You may assume that each declaration of content is valid.

The last test case to be processed is followed by a line consisting of the integer 0.

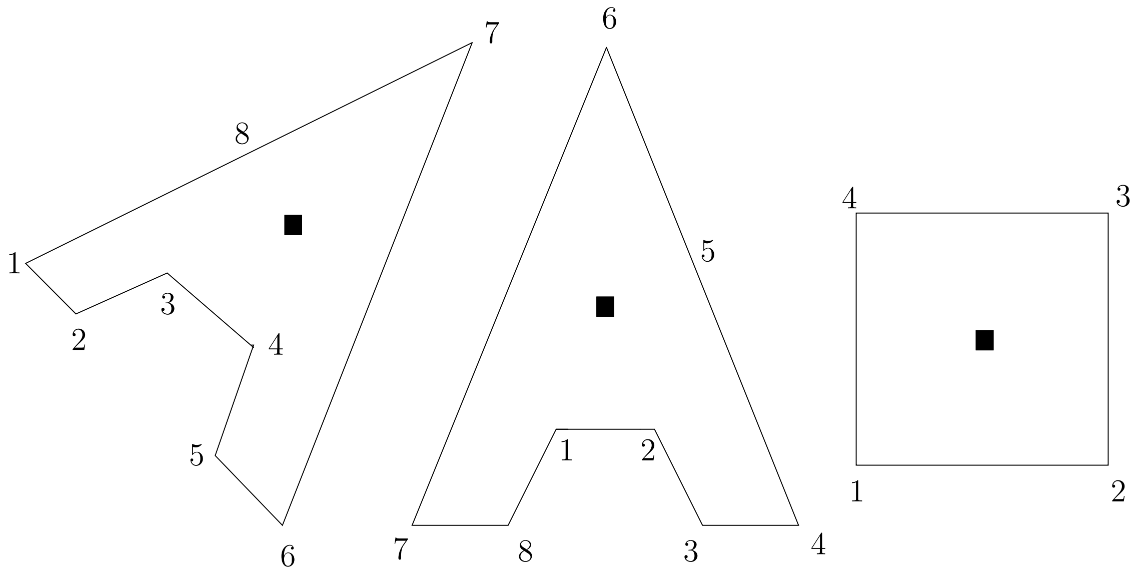
Output

The output consists of one line for every query in the input data. For each query, output the name of the product containing the most or the least of ingredient X, as indicated by the query. If there are several possible such products, output all of them, in the same order as the products were presented in the test case input data. The product names should be separated by a single space.

Sample input	**Sample output**

`3`
`Product_1`
`3`
`A`
`B`
`C`
`Product_2`
`3`
`C`
`B`
`A`
`Product_3`
`2`
`B`
`C 35%`
`4`
`most A`
`most B`
`most C`
`least D`
`0`
	`Product_1`
	`Product_3`
	`Product_2 Product_3`
	`Product_1 Product_2 Product_3`

Problem A: Bumpy Objects



Object 1 Object 2 Square

Consider objects such as these. They are polygons, specified by the coordinates of a centre of mass and their vertices. In the figure, centres of mass are shown as black squares. The vertices will be numbered consecutively anti-clockwise as shown.

An object can be rotated to stand stably if two vertices can be found that can be joined by a straight line that does not intersect the object, and, when this line is horizontal, the centre of mass lies above the line and strictly between its endpoints. There are typically many stable positions and each is defined by one of these lines known as its base line. A base line, and its associated stable position, is identified by the highest numbered vertex touched by that line.

Write a program that will determine the stable position that has the lowest numbered base line. Thus for the above objects, the desired base lines would be 6 for object 1, 6 for object 2 and 2 for the square. You may assume that the objects are possible, that is they will be represented as non self-intersecting polygons, although they may well be concave.

Successive lines of a data set will contain: a string of less than 20 characters identifying the object; the coordinates of the centre of mass; and the coordinates of successive points terminated by two zeroes (0 0), on one or more lines as necessary. There may be successive data sets (objects). The end of data will be defined by the string '#'.

Output will consist of the identification string followed by the number of the relevant base line.

Sample input

```
Square
2 2
1 1  3 1  3 3  1 3 0 0
#
```

Sample output

```
 1                    22
 Object1               6
 Object2               6
 Square                2
```

Problem G. Graveyard

Input file:	`graveyard.in`
Output file:	`graveyard.out`

Programming contests became so popular in the year 2397 that the governor of New Earck — the largest human-inhabited planet of the galaxy — opened a special Alley of Contestant Memories (ACM) at the local graveyard. The ACM encircles a green park, and holds the holographic statues of famous contestants placed equidistantly along the park perimeter. The alley has to be renewed from time to time when a new group of memorials arrives.

When new memorials are added, the exact place for each can be selected arbitrarily along the ACM, but the equidistant disposition must be maintained by moving some of the old statues along the alley.

Surprisingly, humans are still quite superstitious in 24th century: the graveyard keepers believe the holograms are holding dead people souls, and thus always try to renew the ACM with minimal possible movements of existing statues (besides, the holographic equipment is very heavy). Statues are moved along the park perimeter. Your work is to find a renewal plan which minimizes the sum of travel distances of all statues. Installation of a new hologram adds no distance penalty, so choose the places for newcomers wisely!

Input

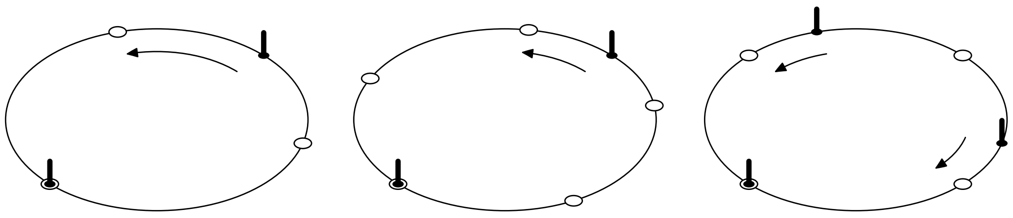
Input file contains two integer numbers: n — the number of holographic statues initially located at the ACM, and m – the number of statues to be added ($2 \le n \le 1000, 1 \le m \le 1000$). The length of the alley along the park perimeter is exactly 10 000 feet.

Output

Write a single real number to the output file — the minimal sum of travel distances of all statues (in feet). The answer must be precise to at least 4 digits after decimal point.

Sample input and output

`graveyard.in`	`graveyard.out`
`2 1`	`1666.6667`
`2 3`	`1000.0`
`3 1`	`1666.6667`
`10 10`	`0.0`



Pictures show the first three examples. Marked circles denote original statues, empty circles denote new equidistant places, arrows denote movement plans for existing statues.

Problem G: Prime Path

The ministers of the cabinet were quite upset by the message from the Chief of Security stating that they would all have to change the four-digit room numbers on their offices.
— It is a matter of security to change such things every now and then, to keep the enemy in the dark.
— But look, I have chosen my number 1033 for good reasons. I am the Prime minister, you know!
— I know, so therefore your new number 8179 is also a prime. You will just have to paste four new digits over the four old ones on your office door.
— No, it's not that simple. Suppose that I change the first digit to an 8, then the number will read 8033 which is not a prime!
— I see, being the prime minister you cannot stand having a non-prime number on your door even for a few seconds.
— Correct! So I must invent a scheme for going from 1033 to 8179 by a path of prime numbers where only one digit is changed from one prime to the next prime.

Now, the minister of finance, who had been eavesdropping, intervened.
— No unnecessary expenditure, please! I happen to know that the price of a digit is one pound.
— Hmm, in that case I need a computer program to minimize the cost. You don't know some very cheap software gurus, do you?
— In fact, I do. You see, there is this programming contest going on...
Help the prime minister to find the cheapest prime path between any two given four-digit primes! The first digit must be nonzero, of course. Here is a solution in the case above.

```
1033
1733
3733
3739
3779
8779
8179
```

The cost of this solution is 6 pounds. Note that the digit 1 which got pasted over in step 2 can not be reused in the last step – a new 1 must be purchased.

Input

One line with a positive number: the number of test cases (at most 100). Then for each test case, one line with two numbers separated by a blank. Both numbers are four-digit primes (without leading zeros).

Output

One line for each case, either with a number stating the minimal cost or containing the word `Impossible`.

Sample input	**Sample output**
`3`	`6`
`1033 8179`	`7`
`1373 8017`	`0`
`1033 1033`	