UW-Madison 2009
ICPC Team Practice 1

Username:_____

Password:_____

The problems will be referred to by the names (respective of order):

- dominos;
- tictactoe;
- crystal;
- signals;
- home;
- logo;
- subway;
- grid; and
- wifi.

Problem D: Dominos

Dominos are lots of fun. Children like to stand the tiles on their side in long lines. When one domino falls, it knocks down the next one, which knocks down the one after that, all the way down the line. However, sometimes a domino fails to knock the next one down. In that case, we have to knock it down by hand to get the dominos falling again.

Your task is to determine, given the layout of some domino tiles, the minimum number of dominos that must be knocked down by hand in order for all of the dominos to fall.

Input Specification

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing two integers, each no larger than 100 000. The first integer n is the number of domino tiles and the second integer m is the number of lines to follow in the test case. The domino tiles are numbered from 1 to n. Each of the following lines contains two integers x and y indicating that if domino number x falls, it will cause domino number y to fall as well.

Sample Input

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

Output Specification

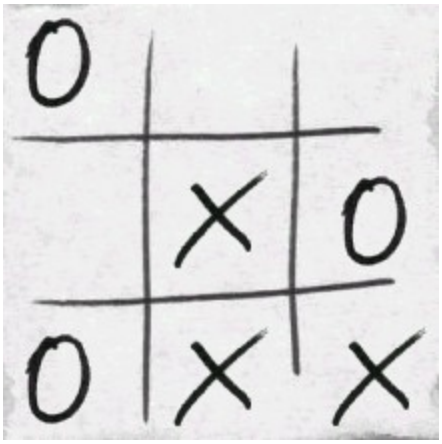
For each test case, output a line containing one integer, the minimum number of dominos that must be knocked over by hand in order for all the dominos to fall.

Output for Sample Input

```
1
```

Ondřej Lhoták

Problem A: Tic Tac Toe



The game of Tic Tac Toe is played on an *n*-by-*n* grid (where *n* is usually but not necessarily three). Two players alternate placing symbols on squares of the grid. One player places Xes and the other player places Os. The player placing Xes always goes first. When the grid contains a vertical, horizontal, or diagonal sequence of at least *m* consecutive squares all containing the same symbol, the game ends and the winner is the player who placed the last symbol. When all the squares of the grid are filled, if neither player has won, the game ends in a draw.

Your task is to analyze the state of a Tic Tac Toe board, and determine whether the game is still in progress, or if it has completed, who won, or if the game ended in a draw. You should also detect erroneous states of the Tic Tac Toe board that could never occur during an actual game.

Input Specification

The first line of input contains an integer indicating the number of test cases. The first line of each test case contains the two integers *n* and *m*, separated by spaces, with 1 <= *m* <= *n* <= 2000. The following *n* lines of input each contain one row of the Tic Tac Toe board. Each of these lines contains exactly *n* characters, and each of these characters is either an `X`, an `O`, or a period (.), indicating an empty square.

Sample Input

```
1
3 3
..X
OOX
..X
```

Output Specification

Output a single line for each test case containing the appropriate string `X WINS`, `O WINS`, or `DRAW` if the game is over, the string `IN PROGRESS` if the game has not yet finished, or `ERROR` if the state of the board could never occur during a game.

Output for Sample Input

Ondřej Lhoták, Malcolm Sharpe

The 1998 22nd Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

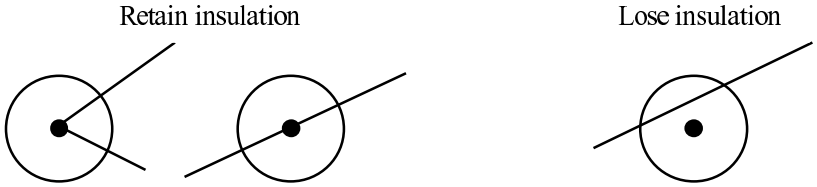
Problem A

Crystal Clear

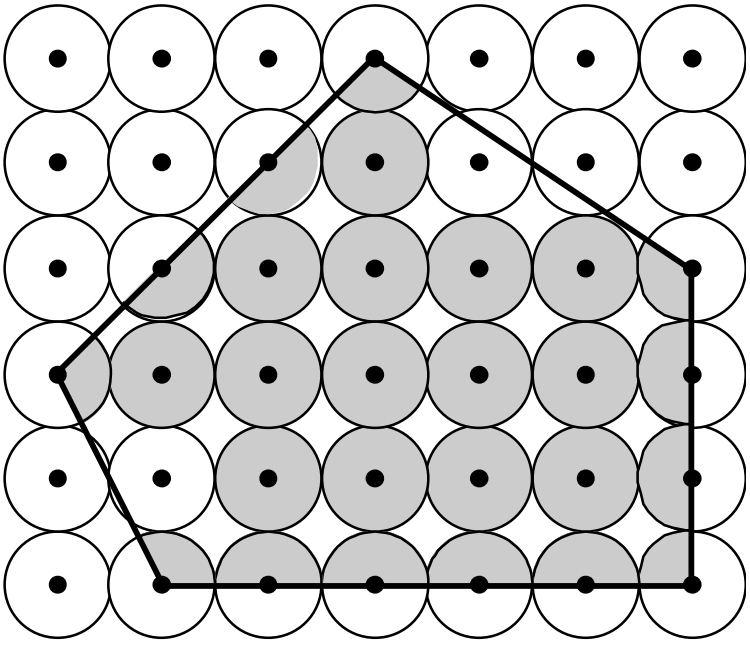
Input: crystal.in

A new high technology company has developed a material that it hopes to market as an insulator. The material consists of crystals and the square lattice on which the crystals are grown. The points on the lattice are at 1 centimeter intervals. The crystals are formed from seeds that are planted at the lattice points. Each crystal grows into a circle of diameter 1 centimeter.

Using this material in applications will require cutting the lattice into pieces. One of the problems in cutting the lattice is that some crystals will be sliced in the process. Slicing a crystal other than through the center completely destroys that crystal's insulation properties. (A cut touching a crystal tangentially does not destroy that crystal's insulation property.)



The insulation capacity of a piece is directly proportional to the total area of the insulating crystals (or parts of crystals) that are on the piece. The following figure shows a polygonal piece with its insulating crystals shaded.



Your job is to determine the insulating capacity of such polygonal pieces by computing the total area of the insulating crystals in it.

Input

The input consists of a sequence of polygon descriptions. Each description consists of a positive integer n ($3 \le n \le 25$) representing the number of vertices, followed by n pairs of integers. Each pair is the x and y coordinates of one vertex of the polygon. (The coordinate system is aligned with the lattice such that the integer coordinates are precisely the lattice points.)

Vertices of each polygon are given in clockwise order. No polygon will be degenerate. No coordinate will be larger than 250 in absolute value.

The input is terminated by zero for the value of n.

Output

For each polygon, first print its number ("`Shape 1`", "`Shape 2`", etc.) and then the area of the insulating crystals in cm^2, exact to three digits to the right of the decimal point.

The following sample corresponds to the previous illustration.

Sample Input

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

Output for the Sample Input

```
Shape 1
Insulating area = 15.315 cm^2
```

Problem A

Bridging signals

*Source code: signals.**

'Oh no, they've done it again', cries the chief designer at the Waferland chip factory. Once more the routing designers have screwed up completely, making the signals on the chip connecting the ports of two functional blocks cross each other all over the place. At this late stage of the process, it is too expensive to redo the routing. Instead, the engineers have to bridge the signals, using the third dimension, so that no two signals cross. However, bridging is a complicated operation, and thus it is desirable to bridge as few signals as possible. The call for a computer program that finds the maximum number of signals which may be connected on the silicon surface without crossing each other, is imminent. Bearing in mind that there may be thousands of signal ports at the boundary of a functional block, the problem asks quite a lot of the programmer. Are you up to the task?

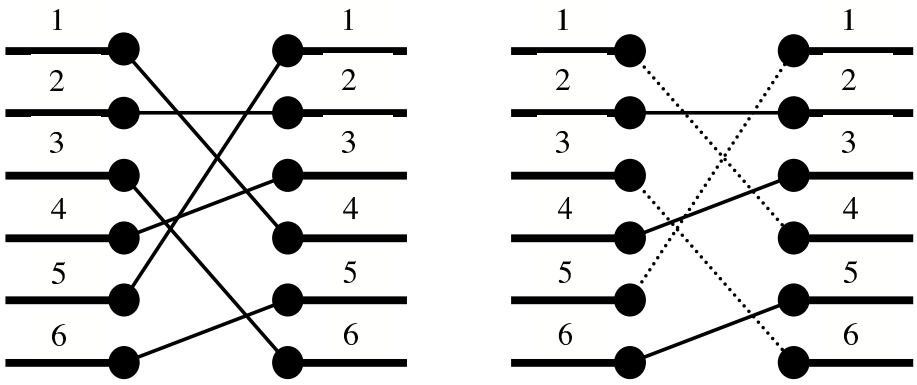


Figure 1. To the left: The two blocks' ports and their signal mapping (4,2,6,3,1,5). To the right: At most three signals may be routed on the silicon surface without crossing each other. The dashed signals must be bridged.

A typical situation is schematically depicted in figure 1. The ports of the two functional blocks are numbered from 1 to p, from top to bottom. The signal mapping is described by a permutation of the numbers 1 to p in the form of a list of p unique numbers in the range 1 to p, in which the i:th number specifies which port on the right side should be connected to the i:th port on the left side. Two signals cross if and only if the straight lines connecting the two ports of each pair do.

Input

On the first line of the input, there is a single positive integer n, telling the number of test scenarios to follow. Each test scenario begins with a line containing a single positive integer p<40000, the number of ports on the two functional blocks. Then follow p lines, describing the signal mapping: On the i:th line is the port number of the block on the right side which should be connected to the i:th port of the block on the left side.

Output

For each test scenario, output one line containing the maximum number of signals which may be routed on the silicon surface without crossing each other.

Example input:

```
4
6
4
2
```

Problem A, cont.

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

Example output:

```
3
9
1
4
```

Problem 6: Going Home

A little region of the world is divided into equal-sized rectangular areas. In this little region there are *n* little men and *n* little houses. Every little man can move horizontally or vertically but not diagonally to an adjacent area, being paid a $1.00 travel fee for every move he makes between adjacent areas until he enters a little house. Your task is to compute the minimum travel fees required to get these *n* little men into those *n* little houses. The task is complicated by the restriction that each little house can accommodate only one little man.

The input is a map of the region, and has one of the characters '`.`', '`H`', or '`m`' in each area. A '`.`' identifies an empty area, an '`H`' identifies an area containing a little house, and an '`m`' identifies an area containing a little man.

Each area is quite large; it can hold up to *n* little men and a little house at the same time. A little man can also enter an area containing a little house without necessarily entering the little house. Initially, however, each area will hold at most one little man or one little house.

Input

The input will contain multiple cases. Each case starts with a line having two integers *N* and *M*; *N* is the number of rows (of areas) in the grid map, and *M* is the number of columns (of areas). The remainder of the input for the case will be *N* lines giving the map, one line for each row of areas. *N* and *M* are each between 2 and 100, inclusive. There may be one or more trailing whitespace (blank or tab) characters on a line. The number of 'H's on the map will equal the number of 'm's on the map, and there will be at most 100 houses. The last case will be followed by a line containing two integer zeroes.

Output

For each case, display the case number (they start with 1 and increase sequentially) and the minimum number of dollars required for travel fees. The output format should resemble that shown in the sample output.

Sample Input	Output for the Sample Input
2 2 .m H. 5 5 HH..m .... mm..H 7 8 ...H.... ...H.... ...H.... mmmHmmmm ...H.... ...H.... ...H.... 0 0	Case 1: $2 Case 2: $10 Case 3: $28

Problem E: Logo 2

Logo is a programming language built around a turtle. Commands in the language cause the turtle to move. The turtle has a pen attached to it. As the turtle moves, it draw lines on the page. The turtle can be programmed to draw interesting pictures.

We are interested in making the turtle draw a picture, then return to the point that it started from. For example, we could give the turtle the following program:

```
fd 100 lt 120 fd 100 lt 120 fd 100
```

The command `fd` causes the turtle to move forward by the specified number of units. The command `lt` causes the turtle to turn left by the specified number of degrees. Thus the above commands cause the turtle to draw an equilateral triangle with sides 100 units long. Notice that after executing the commands, the turtle ends up in the same place as it started. The turtle understands two additional commands. The command `bk` causes the turtle to move backward by the specified number of units. The command `rt` causes the turtle to turn right by the specified number of degrees. The distances and angles in all commands are always non-negative integers.

Unfortunately, we have been messy in writing the program down, and cannot read our own writing. One of the numbers in the program is missing. Assuming the turtle ends up at the place that it started at the end of its journey, can you find the missing number?

Input Specification

The first line of input contains one integer specifying the number of test cases to follow. Each test case starts with a line containing one integer, the number of commands to follow. The commands follow, one on each line. Each test case will contain no more than 1000 commands. The argument of each command is either an integer or a question mark (?). There will be exactly one question mark in each test case.

Sample Input

```
1
5
fd 100
lt 120
fd ?
lt 120
fd 100
```

Output Specification

For each test case, output line containing a single integer *n* such that when the question mark in the program is replaced by *n*, the turtle ends up at the same point that it started from once the program completes. If the question mark is the argument of an `lt` or `rt` command, the angle in the output must be between 0 and 359 degrees, inclusive. The correct answer will always be an integer, and we guarantee that for every test case, there will be only one correct answer.

Output for Sample Input

```
100
```

Ondřej Lhoták

Subway

You have just moved from a quiet Madison neighbourhood to a big, noisy city. Instead of getting to ride your bike to school every day, you now get to walk and take the subway. Because you don't want to be late for class, you want to know how long it will take you to get to school.

You walk at a speed of 10 km/h. The subway travels at 40 km/h. Assume that you are lucky, and whenever you arrive at a subway station, a train is there that you can board immediately. You may get on and off the subway any number of times, and you may switch between different subway lines if you wish. All subway lines go in both directions.

The input begins with an integer representing the number of cases to solve. Each case consists of the x,y coordinates of your home and your school, followed by the number s of subway lines in the city. The descriptions of these s lines follow- each description consists of the non-negative integer x,y coordinates of each stop on the line, in order. You may assume the subway runs in a straight line between adjacent stops, and the coordinates represent an integral number of metres. Each line has at least two stops. The end of each subway line is followed by the dummy coordinate pair -1,-1. In total there are at most 200 subway stops in the city.

For each test case, output the case number followed by the number of minutes it will take you to get to school in that case, rounded to the nearest minute, taking the fastest route. Print the answer for each test case on a separate line.

Sample Input

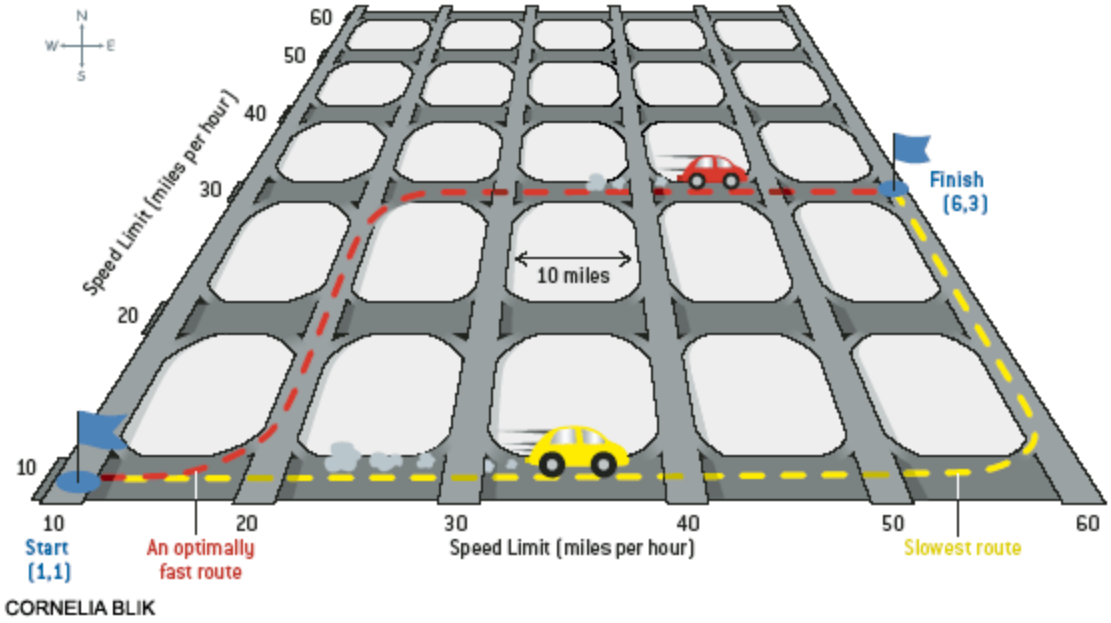
```
1
0 0 10000 1000 2
0 200 5000 200 7000 200 -1 -1
2000 600 5000 600 10000 600 -1 -1
```

Sample Output

```
Case 1: 21 minutes
```

Problem D: Grid Speed

Consider a grid in which north-south streets, separated by gaps of 10 miles each, are elevated above east-west streets laid out in a similar fashion (see illustration for the case of a 6 by 6 grid). All streets are two-way. Entrance and exit ramps connect the streets at every intersection. Because there are no traffic lights, switching from a north-south street to an east-west street, and vice versa, takes essentially no time. The grid has very little traffic, but the local police patrol so carefully for speeding that there are virtually no speeders.



The speed limits follow an unusual pattern. The speed limits are separately posted for each street and are the same for the entire street in both directions. In the illustration above, let us label the intersections using their column and row numbers: the southwestern corner of the grid is (1, 1), the southeastern corner is (6, 1), and so on. Part of your task is to determine the shortest time in which we can get from (1, 1) to (6, 3) while obeying speed limits.

However, after the Kyoto disagreement, just being fast is not good enough, one also has to be fuel efficient. Fuel consumption of a car is given in miles-per-gallon (mpg) and depends on speed of the car. Speed of a car is given in miles-per-hour (mph) and, in this digital age, the speed of a car is always a positive integer multiple of 5. The formula relating mpg to mph is a very simple one: a car travelling at v mph makes $80-0.03*v^2$ mpg. In a given grid of streets we would like to travel from intersection (x_s, y_s) to intersection (x_t, y_t). You are to

determine the fastest and the most fuel efficient way of making the trip such that:

- the car does not change speed between intersections,
- the car obeys all speed limits,
- the car travels the shortest possible distance between the start and finish, and
- the car arrives at the destination in the given time interval.

Input

The first line of input contains an integer t, the number of scenarios to be processed. The data for each scenario occupy 5 lines. The first line contains an integer n<=10 which is the number of horizontal and vertical streets. The second line contains an integer which is the grid unit size in miles, smaller than 100. The third and fourth lines contain n integers each, specifying the speed limits on the horizontal and vertical streets, respectively. The largest speed limit is 50. The last line of data for a scenario contains 6 integers. The first four are x_s, y_s, x_t, and y_t. The last two integers give the shortest and the longest allowed time to travel in minutes, inclusive, both not bigger than 1000.

Output

For each scenario, output two or three lines in the format given in the sample output. If the travel is possible then, on the second line of output, report the earliest possible arrival time (but within the imposed limits) and fuel consumed (least possible for this travel time) and, on the third line, report the earliest arrival time (but within the imposed limits) that consumes the minimum amount of fuel. The time is to be reported in minutes (integer), rounded up.

Sample input

```
3
8
20
10 20 30 40 50 50 50 50
50 50 50 50 50 50 40 50
2 3 7 8 300 320
8
2
10 20 20 30 10 20 10 10
10 20 20 30 10 20 10 20
6 8 2 4 10 39
10
10
30 20 20 10 10 20 10 10 20 20
40 20 10 20 10 20 20 10 10 20
1 1 10 10 100 500
```

Output for sample input

```
Scenario 1:
The earliest  arrival: 300 minutes, fuel 6.25 gallons
The economical travel: 318 minutes, fuel 5.60 gallons
Scenario 2:
IMPOSSIBLE
Scenario 3:
The earliest  arrival: 405 minutes, fuel 4.14 gallons
The economical travel: 498 minutes, fuel 2.76 gallons
```

Piotr Rudnicki, based on Dennis E. Shasha's column, Scientific American, March 2004

Problem B: WiFi

One day, the residents of Main Street got together and decided that they would install wireless internet on their street, with coverage for every house. Now they need your help to decide where they should place the wireless access points. They would like to have as strong a signal as possible in every house, but they have only a limited budget for purchasing access points. They would like to place the available access points so that the maximum distance between any house and the access point closest to it is as small as possible.

Main Street is a perfectly straight road. The street number of each house is the number of metres from the end of the street to the house. For example, the house at address 123 Main Street is exactly 123 metres from the end of the street.

Input Specification

The first line of input contains an integer specifying the number of test cases to follow. The first line of each test case contains two positive integers *n*, the number of access points that the residents can buy, and *m*, the number of houses on Main Street. The following *m* lines contain the house numbers of the houses on Main Street, one house number on each line. There will be no more than 100 000 houses on Main Street, and the house numbers will be no larger than one million.

Sample Input

```
1
2 3
1
3
10
```

Output Specification

For each test case, output a line containing one number, the maximum distance between any house and the access point nearest to it. Round the number to the nearest tenth of a metre, and output it with exactly one digit after the decimal point.

Output for Sample Input

```
1.0
```

Ondřej Lhoták