Carnegie Mellon University
Spring Programming Competition

March 28, 2009

You can program in C, C++, or Java; note that the judges will re-compile your programs before testing.

Your programs should read the test data from the standard input and write results to the standard output; you should not use files for input or output.

All communications with the judges should be through the PC^2 environment.

Each problem's estimated level of difficulty is shown next to the title. This guidance is only an estimate, and is based on many, possibly incorrect, assumptions. Take it as only one, of many, factors contributing to your estimate of your own team's ability to solve the problem. Remember: Your mileage may vary!

(☺ ☺ ☺ ☺ ☺)	Very Challenging
(☺ ☺ ☺ ☺)	Challenging
(☺ ☺ ☺)	Nuances or subtlety, or upper level math/algo/problem solving
(☺ ☺)	Some nuances or library or algorithmic knowledge required
(☺)	Straight-forward, few if any nuances

The hypothetical examples below illustrate the notation for a straight-forward problem, Problem Y, and a very challenging problem, Problem Z:

Problem Y: Easy points! (☺)

Problem Z: Badge of Honor, e.g. Killer (☺ ☺ ☺ ☺ ☺)

Problem A: Magic Scrolls (☺ ☺ ½)

www.elvish.org

A recent shipment of magical elvish scrolls has arrived but, unfortunately, it has gotten mixed up with orcish war chants. Fortunately, there is an easy way to identify if the text within a particular scroll is magical. To do this, we need to distinguish between *alphabetical order*, the order of the letters in the standard English alphabet, and *text order*, the order of the letters as they appear within the text of some scroll. We also need to draw the distinction between *uppercase* and *lowercase* letters for the purpose of interpreting the scrolls, but not for the purpose of determining alphabetical order.

A sequence of text within a scroll is magical if no uppercase letter appears in the *text order*, before a lowercase letter that precedes it in *alphabetical order*. For example,

"AllakHAzim" is magical, because none of the three instances of uppercase letters, 'A', 'H' and 'A' appear in *text order* before an alphabetically preceding letter.

"moriquEssieRniiBuyy" is, by contrast, not magical. Notice that the lowercase 'n' appears after the uppercase 'R' in *text order*, but observe that 'n' precedes 'r' alphabetically.

Your task is to write a program that can differentiate between these two types of text.

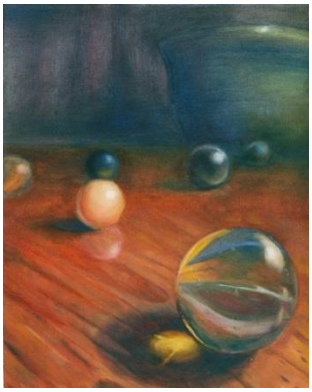
Input
The first line of the input will contain an integer n ($1 \leq n \leq 100$) which represents the number of test cases. Following that there will be *n* lines of input, each of which contains a non-empty string no longer than 80 characters. Each line may contain only upper and lowercase letters.

Output
For each test case, your program should output either "Magical" if the text is magical, or "Not Magical" otherwise.

Sample Input
```
2
AllakHAzim
moriquEssieRniiBuyy
```

Sample Output
```
Magical
Not Magical
```

Problem B: Marbles (☺)

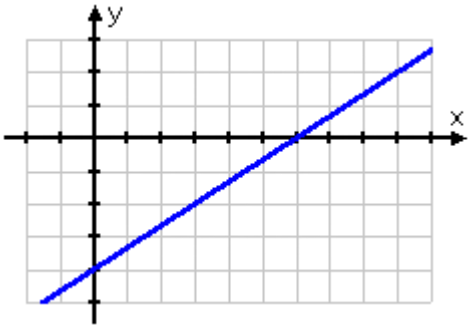


mark-story.com

Foundations and pillars tend to settle over the years, leaving the floors in older homes unlevel. Unfortunately, this means that when small, round items, such as pens, pencils, marbles, etc., are dropped they tend to roll to one side of a room or the other, coming to rest against the wall. Even worse, they often become obscured by furniture, making them difficult to find.

It would be much easier to find these lost items, if we knew which way they rolled. Knowing this, looking for lost items would only require moving the furniture on one side of the room, or the other, but not both.

The only good thing is that the floors of the rooms tend to sag in one direction or the other, so they can be modeled as straight lines in Cartesian planes. Consider the cross-section of the floor shown below. It is a tremendously problematic floor, the left side of which has sunken from above ground level all the way into the basement. Anything dropped on this floor that can roll, will roll. And it will roll to the left.



Credit: www.purplemath.com

This problem presents you with two distinct (x, y) points along the cross-section of a floor. The two points define a line, such as the one shown above. Left is considered to be in the direction of lower x-values. Right is considered to be in the direction of greater x-values. Similarly, higher y-values are considered to be up with respect to lower y-values. In other words, our left-right/up-down frame of reference matches the natural orientation of the line as you see it in the Cartesian plane.

The points are not necessarily given to you in left-to-right, top-to-bottom, or any other order. Given these two points you are asked to determine if a marble placed on the floor will roll to the left, to the right, or neither. You should neglect friction and inertia. No floor is completely vertical.

Input
The first line of the input will contain an integer n ($1 \leq n \leq 100$) which represents the number of test cases. Following that there will be n lines of input, each of which will contain four integers, x_1 and y_1, and x_2 and y_2. These four integers represent the two points defining the line as follows: (x_1, y_1) and (x_2, y_2)

Output
For each test case, your program should output "Left", "Right", or "Neither" to indicate the direction that a marble would roll.

Sample Input

```
2
1 2 3 4
3 4 10 -1
```

Sample Output

```
Left
Right
```

Problem C: Matching Bookends (☺ ☺)

www.dragonsfire.co.uk

When it comes to fashion and style, academics are among the biggest offenders of common decency. It is sad, but true, that our civil society no longer looks toward the Ivory Tower for a sense of style and fashion. None-the-less, the long-standing association between the world's greatest schools and fine bound texts presents a unique opportunity for those in the learned professions to maintain leadership in stylishly blurring form and function. In order to ensure that proper office decorum does not go the way of a stylish wardrobe in our fine academic community, we are establishing guidelines for the placement of bookends.

In order to understand these guidelines, it is necessary to observe that, although historically functional, today's rigid bindings enable texts to remain standing without supporting bookends. For this reason, bookends are purely accents in the modern office, and their placement is in no way constrained by actual books.

It is also necessary to observe certain subtle aesthetic details. In particular, bookends are not symmetric blocks. They may be oriented *left-side* or *right-side*. Left-side bookends are rotated such that the decorative side is to the left and the flat-side is to the right. Right-side bookends are turned in the opposite way, such that the flat side is to the left and the decorative side is to the right. Historically, though not presently, books were placed between the flat sides of the bookends.

With these preliminaries out of the way, you are asked to write a program that verifies the compliance of faculty bookshelves with the following set of guidelines:

(a) No bookshelf may be without bookends
(b) Each bookshelf must have the same number of left-side and right-side bookends.
(c) If one scans a bookshelf from left-to-right, counting each of left-side and right-side bookends along the way, at no point should the number of right-side bookends exceed the number of left-side bookends

	
Example compliant use of bookends	Example non-compliant use of bookends

Input

The input will consist of multiple test cases, one per line. Each test case consists of a string of length at most 200 characters. The '(' character represents a left-side bookend. The ')' character represents a right-side bookend. Any other alpha-numeric character represents a book. No other characters, for example punctuation or whitespace, will be present in the input. Each test case will contain at least one book or bookend. The input is terminated by a line containing exactly "0".

Output
For each test case, you should print a single line containing either, “Compliant” or “Not Compliant” to indicate the shelf’s compliance or non-compliance with the guidelines above.

Sample input
```
abababab(b)ccsadasdasd
((()aasjdaslk
)asd(1234123)(xfv
()asdsa()asdasd
0
```

Sample output
```
Compliant
Not Compliant
Not Compliant
Compliant
```

Problem D: Steam Tunnels(☺☺☺)

www.undergroundozarks.com

Hidden beneath most college campus is a vast network of hundreds of steam tunnels that, for the adventurous who don't mind risking injury or jail, can provide many hours of excitement. The tunnels all connect together, directly or indirectly, to form a single tunnel system. Being the carefree type, you have spent many hours exploring the underworld. And, being the organized type, you have taken good notes.

On one cold, windy, stormy, and generally unpleasant winter day, you decide to show off and take a less adventurous friend to his next class via the tunnels. A warm way to get from building to building just can't be beat. But, you know that your friend is claustrophobic. If, somewhere along the way, you have to cross a tunnel that is too narrow for his taste, he won't come along. And, for his comfort, you want to maximize the width of the narrowest tunnel that you cross along the way.

But, until he gets out of class and you can ask him, you don't know how narrow is too narrow. And, you've forgotten his schedule, so you don't know where he has to go next, anyway. In order to be prepared, you decide to write a program that can, upon getting the missing information, quickly help you out. If you feed it your starting location, your destination location, his minimum tunnel size, and the information about all of the tunnels, it should be able to tell you if you can get him there via the steam tunnels, and, if so, the width of the narrowest tunnel that you'll cross along the way.

Input

The first line of the input will contain an integer n ($1 \leq n \leq 100$) which represents the number of test cases. The first line of each test case contains four non-negative integers: *s, d ,t* and *m*. *s* and *d* are room numbers of the starting location, and the destination location respectively. *t* is the number of tunnels in the system, and *m* is the minimum width for a tunnel to be usable by your friend.

The subsequent t lines within each test case contain a description of a single tunnel consisting of three integers: l_1, l_2, w ($50 > l_1, l_2 \geq 0$; $w > 0$), representing a tunnel linking locations l_1 and l_2, with a width of w. Locations are numbered sequentially and densely beginning with 0. Tunnels may be traversed in both directions. There is, at most, one tunnel between any pair of locations.

Output

For each test case, you should print out a single line. If it is possible to get from *s* to *d*, without violating your friend's minimum tunnel width, you should print out a line containing a single integer, the width of the narrowest tunnel that you'll cross along the way. If it is not possible you should print out a line containing the string "NOT POSSIBLE".

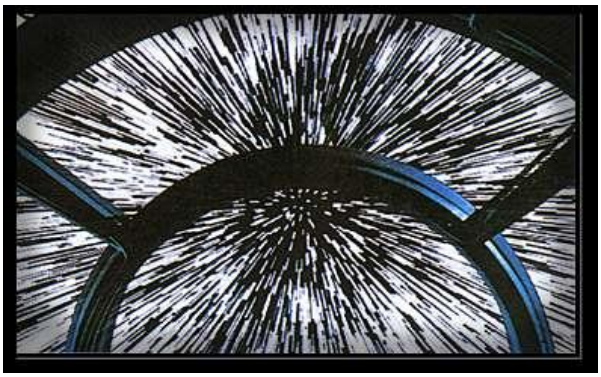
Sample Input

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

Sample Output

```
5
NOT POSSIBLE
```

Problem E: Hyperspace (☺☺)

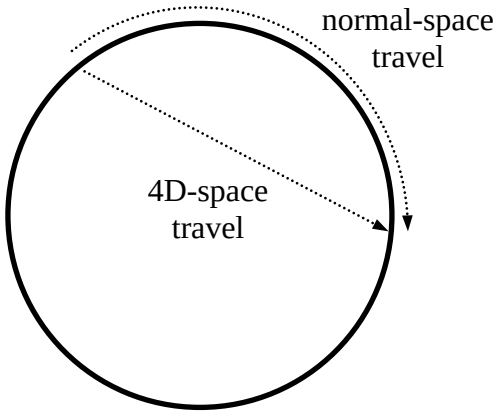


www.nobeliefs.com

Should you hear that the government secretly communicates with little green men from Alpha Centauri, who travel to Earth through the hyperspace, you should not believe such irresponsible rumors. Hyperspace does not exist, and the little green men from Alpha Centauri travel through four-dimensional Euclidean space, which is completely different from the hyperspace.

Specifically, the little aliens use the fact that the universe is a sphere in a four-dimensional space (see the picture), much in the same way as the Earth's surface is a sphere in a three dimensions. When aliens travel between two points, they can go straight through the four-dimensional space, rather than traveling in the normal universe, which is similar to cutting straight through Earth instead of following its surface. Although the straight path is somewhat shorter than the corresponding arc, the length difference is usually not that large (see the picture). Furthermore, the travel in the four-dimensional space requires 10% more fuel per light year. For example, if a spaceship burns 1.0 gallon of antimatter per light year in the normal space, then it burns 1.1 gallons per light year when cutting straight through the four-dimensional space.

Before each trip, the aliens have to determine whether traveling through the normal space or through the four-dimensional space takes less fuel, and your task is to write a program that answers this question. **For simplicity, you may assume that the universe is a circle in two dimensions** rather than a sphere in four dimensions. The distance of travel in the normal space is the length of the arc between two given points, and the respective distance of the straight travel in four dimensions is the length of the respective chord.



Input

The input includes multiple test cases, which correspond to different trips; the number of cases is at most 20. Each test case includes two positive integer values on a single line, separated by a single space. The first integer is the diameter of the universe, and the second is the distance between the start and end of the trip in the normal space, that is, along an arc; this distance is no larger than the diameter of the universe. The last line of the input is "`-1 -1`", which does not represent a test case.

Output
For each test case, the output indicates whether the trip in the normal space or four-dimensional space requires less fuel. If the normal space requires less fuel, the output is “use normal space”; else, it is “use 4D space.”

Sample input
```
10 1
10 10
-1 -1
```

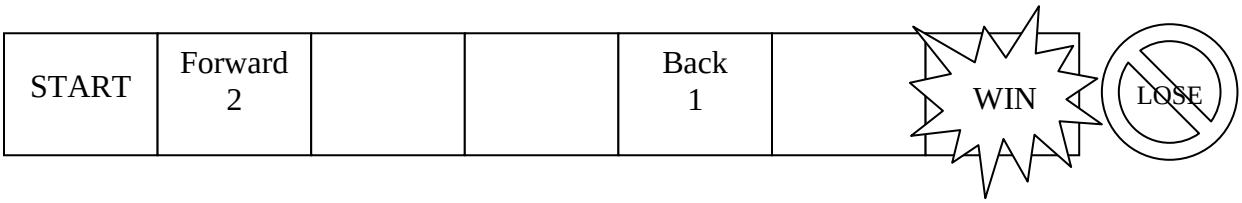
Sample output
```
use normal space
use 4D space
```

Problem F: Optimal Cheating (☺ ☺ ☺ ☺)

thefuntimesguide.com

Bob is bored and all alone, so he decides to invent a new one-player game. The game is played on a long board that is one square wide. The game piece starts on the first square, and the player rolls three 6-sided dice, and moves forward a number of spaces equal to the sum of the three dice. This repeats until the piece lands on the WIN square or advances off of the board, which is a loss.

To make the game interesting, though, some squares on the board are special "jump" squares. If the piece lands on a jump square, it is transported backward or forward the prescribed number of spaces. The board is configured such that jump squares never cause a jump to another jump square or off of the board.



Frustrated with his bad luck, Bob has resorted to cheating. Instead of actually rolling the three dice, he will simply choose values for the dice and pretend that he rolled them. To streamline the cheating process, Bob wants you to figure out the minimum number of moves required to get Bob from square 1 to the end. Unfortunately, sometimes it is impossible for Bob to win, and you should detect this case as well.

Input

The input will consist of multiple test cases. Each test case will contain the description of a board. The description will start with two integers, L ($0 < L \leq 1000$) and N ($0 \leq N \leq 100$). L is the length of the board, and N is the number of jump squares.

This will be followed by N lines, each containing two integers, x and D, indicating that square x moves the player D squares forward. D is the non-zero distance of the jump. If D is negative, then it moves the piece backward by d squares. Neither the first nor the last square will be a jump square.

The first square on the board is square 1 and the last square is square L. The input will end with a line containing "0 0"

Output

For each test case, you should print a single line. If there is no way to win the game for the test case, print the word “IMPOSSIBLE”. Otherwise, print a single integer, the minimum number of moves required to win the game.

Sample input

```
200 4
40 70
20 -5
2 198
50 149
2 0
300 2
30 70
95 205
1000 0
0 0
```

Sample output

```
8
IMPOSSIBLE
6
56
```

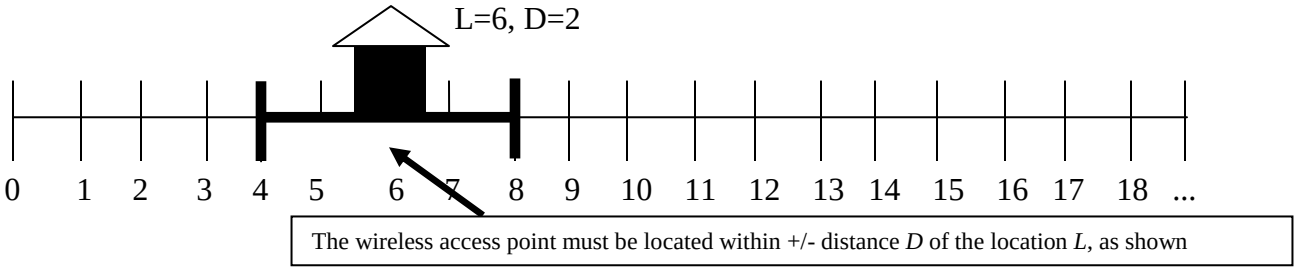
Problem G: Wireless Coverage (☺ ☺ ☺ ☺)

xtremedevices.com

The local internet and telephone company is deploying a wireless network throughout the city. This new network is being rolled out on a street-by-street basis. In planning the project, the company observed that many customers are willing to pay extra for faster service, which can be achieved by reducing the distance between the house and the access point serving it. They neglected giving any consideration to the impact that the sharing of access points between customers might have on performance.

Before the network is deployed, customers along the street are signed up with contracts specifying the maximum distance between their house and the access point serving it. These contracts are then given to engineers who determine the minimum number of access points needed to satisfy the demand along the street. These access points are ultimately ordered from the supplier and installed.

For our purposes, each street can be modeled as a number line. Each house's location is represented by its non-negative integer position on this number line. Access points are also located along the same number line, so the maximum distance between a house and its access point is measured along this line, not in two or three dimensions



One of your colleagues wrote a program that translates the customer orders into a nice computer-friendly list, each line of which contains the location, *L*, and maximum acceptable distance to the nearest access point *D*, as shown above.

Your task is to write a program that, given this list, computes the minimum number of access points needed to satisfy everyone. You do not need to determine the locations of the access points, just the number required to satisfy the needs of each and every customer along the street.

Input
The input file may contain multiple test cases. The first line of each test case contains a single integer *N*, where $0<N\leq1000$, which represents the number of houses on the street. Each of the *N* subsequent lines contains two integers, *L*, and *D*, representing a particular house's location, where $0\leq L \leq 100000$, and $0\leq D\leq100000$, the maximum distance between that house and the closest access point. $N=0$ denotes the end of input.

Output
For each test case output the minimum number of access points needed to satisfy each house.

Sample input

```
1
5 5
2
5 5
15 5
2
5 5
16 5
0
```

Sample output

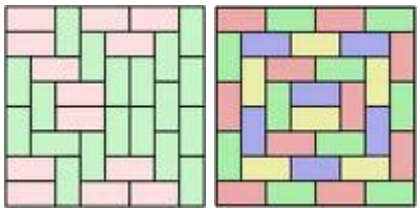
```
1
1
2
```

Problem H: Dominoes (☺☺☺☺☺)

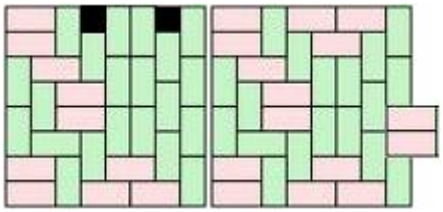
www.bcgameworks.com

One famous puzzle asks, "Is it possible to tile a chessboard, modified by removing the upper left and the lower right corners, with dominoes?" (A domino is a 2x1 rectangular tile.) Theoretical computer scientists have proven, by a simple parity argument, that this can't be done. Fortunate for the contestants, we managed to find some of those smart theorists before writing this question.

So, we ask you to do something that is doable: Write a program to count the number of ways to tile rectangular boards with dominos. A board is tiled by dominoes if the dominoes are arranged in such a way as to completely cover the board -- without going over the edges, or overlapping each other. Each unique arrangement of 2x1 tiles that covers the board in this way is a different tiling. For the purpose of this problem, a domino is just a generic 2x1 rectangular tile – we are not concerned with the dots, etc.



----------Valid Tilings----------



----------***Not*** Tilings----------

(Images from wikipedia)

Input

The input may consist of multiple test cases. Each test is a line containing the height and the width (separated by a space) of the rectangle to be tiled. The height and width will be in the range 1 to 10 (inclusive). The input will be terminated by a line containing "0 0".

Output

For each test case, you should print a single line containing the number of different ways that the given rectangle can be tiled with dominos. Note that this number may exceed the bounds of an *int*, but not a *long*.

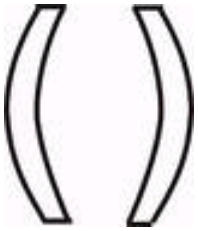
Sample input

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

Sample output

```
2
3
0
11
36
```

Problem I: Revenge of the Parenthesis (☺ ☺ ☺ ☺ ½)



www.webster-dictionary.com

A string of parentheses is *balanced* if it's either "()" or it is "(" followed by the concatenation of two balanced strings followed by ")". For example "(()())" is balanced, and so is "((()())())". But "()()" is not.

You're given a string containing only the characters '(', ')', and '*'. Compute the number of different balanced strings that can be constructed where a '*' can be replaced by either type of parentheses.

Input

The input will consist of multiple test cases. Each test case will be a line containing a string of length between 1 and 100 (inclusive), containing only '(', ')', and '*'. The input is terminated by a line containing '0'.

Output

For each test case, you should print a single line containing the number of different balanced strings of parenthesis consistent with the given string. Note that this number may exceed the bounds of an *int*, but not a *long*.

Sample input

```
()
(()())
*****
**********
(((******(***)****
0
```

Sample output

```
1
1
0
2
3
```