Colombian Collegiate Programming League

CCPL 2013

Contest 12 -- November 23

Problems

This set contains 10 problems; pages 1 to 20.

(Borrowed from several sources online.)

Official site `http://programmingleague.org`

Official Twitter account `@CCPL2003`

A - Langton's Ant

Source file name: `ant.c`, `ant.cpp`, *or* `ant.java`

Langton's ant, after the mathematician Christopher Langton, is a cellular automaton with a very simple set of rules but interesting emergent behavior; this behavior is currently the matter of research for some mathematicians. The analogy between ants and Langton's cellular automaton comes from the observation that one can arbitrarily identify the state of the automaton as the ''ant'', and the dynamics of the automaton with the ability of the ant to travel in a special world.

The ant world's is an $n \times n$ plane where the squares (or cells) on the plane are colored variously either blue or red. The number n is called the *size* of the world. A cell is denoted with a pair (i, j), $(1 \leq i, j \leq n)$. The ant lives and moves in single steps following the rules below:

- if it is on a blue cell, it flips the color of the cell, turns $\frac{\pi}{2}$ to the left, and moves forward to the next cell in the direction it is facing;
- if it is on a red cell, it flips the color of the cell, turns $\frac{\pi}{2}$ to the right, and moves forward to the next cell in the direction it is facing; and
- if a movement is impossible (because the ant cannot move out of the world), then the ant dies.

For example, let us assume the ant is in a red cell while facing east. If there is not a cell immediately to its south, then the ant dies. On the contrary, if there is a cell immediately to its south, then the ant takes a single step by moving to this cell to which it arrives facing south and the color of its source cell flips to blue. Then, the ant will try to take another single step, and so on.

Your task is to determine if the ant can go to the (n, n) cell of a world, given (i) the configuration of the world, and (ii) the initial position of the ant. You are to assume the ant's initial direction is north.

Input

The configuration of an $n \times n$ world can be codified as a natural number in binary notation by using n^2 bits. We adopt the following conventions: 0 = blue, 1 = red, and the binary representation of the configuration identifies the cells of the world from left to right and from bottom to top (considering the bits from the most significant bit to the least significant bit). For example, the binary number 0100 (4 in decimal notation) represents a 2×2 world with the following configuration:

blue	blue
blue	red

Coherently, the binary number 011010100 (212 in decimal notation) represents a 3×3 world with the following configuration:

red	blue	blue
blue	red	blue
blue	red	red

The problem input has several test cases. Each case consists of a single line containing a list of four natural numbers, n, c, x, y, separated by blanks, that should be interpreted as:

- n ($1 \leq n \leq 16$): the size of the world;
- c ($0 \leq c < 2^{(n^2)}$): decimal representation of an n^2-bit binary number that describes an initial configuration of the world, as above explained;
- (x, y): coordinates of the initial position of the ant in the world ($1 \leq x, y \leq n$), where the position (n, n) corresponds to the least significant bit of c.

The end of the input is indicated by a line where $n = c = x = y = 0$.

The input must be read from standard input.

Output

For each test case your solution should output:

- `Yes` if the ant reaches the cell (n, n) from the initial position;
- `Kaputt!` if the ant dies without reaching the cell (n, n) from the initial position.

For each test case, it's guaranteed that after a finite number of steps, the ant reaches the cell (n, n) or dies without reaching the cell (n, n).

The output must be written to standard output.

Sample Input	**Sample Output**

`2 8 1 1`
`2 4 1 1`
`2 15 1 1`
`0 0 0 0`
	`Yes`
	`Kaputt!`
	`Kaputt!`

B - Preferential Romance

Source file name: `romance.c`, `romance.cpp`, *or* `romance.java`

Marriage Success (MS) is a marriage counseling service advising couples on how to improve the 'get along' experience. MS's idea is simple: each spouse writes down his/her preferences for various criteria of common interest. ''Our criteria go beyond physical appearance and passion that guide early romance and tend to blind judgement. We want to understand your values as you live day by day. Happy couples are those whose preferences are compatible or can be made compatible.''

Suppose X and Y are qualities to be considered. If a person declares that $X > Y$, it means that this person prefers quality X to quality Y (it does not mean that his/her mate should have a quality, it is only an opinion). Preferences are obviously irreflexive (i.e., $X \not> X$) and they are transitive (i.e., if $X > Y$ and $Y > Z$, then $X > Z$ --which can be abbreviated as $X > Y > Z$).

A couple is *fully compatible* if the preferences of the spouses are *consistent*, that is, if it is possible to arrange the qualities of interest of the spouses in a *compatibility list* reflecting both of their preferences. In this case, if a spouse says $X > Y$, qualities X and Y must occur in the compatibility list and moreover X must be preferred over Y. If a couple is not fully compatible, then perhaps at least it is *passably compatible*: their preferences can be made consistent if some spouse drops at most one preference.

For example, newly-wed Alice and Bob declare their preferences with respect to the following qualities (that they observe in a possible mate): biker, cultured, enthusiastic, foodie, juggler, kayaker, movies, organized, puzzles, rich, theatre, and windsurfer. Their preferences are (observe that they could say nothing about qualities meaning that such quality does not have any importance):

Alice: organized > puzzles > rich, windsurfer > theatre, and rich > movies.

Bob: kayaker > movies > puzzles and rich > theatre.

In this case Alice and Bob are not fully compatible. To see that, a compatibility list should have rich before movies (Alice), movies before puzzles (Bob), and puzzles before rich (Alice), meaning that rich must occur before rich which is impossible. However, the couple is passably compatible: if Alice drops her preference rich > movies, then there is a compatibility list modeling both of their preferences:

kayaker > organized > movies > puzzles > rich > windsurfer > theatre.

MS needs a software solution to determine if clients are full compatible, passably compatible or none of these. Can you help?

Input

The problem input consists of several test cases, each one defined by a set of lines establishing preferences of a couple. A test case is defined as follows:

- the first line contains two strings A and B, separated by blanks, representing the name of the spouses,
- the second line is a sequence of strings of the form (one or more blanks separating items, including commas and final semicolon):

  $$q_{11}\ q_{12}\ \cdots\ q_{1r_1}\ ,\ q_{21}\ q_{22}\ \cdots\ q_{2r_2}\ ,\ \cdots\ ,\ q_{m1}\ q_{m2}\ \cdots\ q_{mr_m};$$

  meaning that person A has sets of preferences (q_{ij}'s are strings denoting qualities):

  $$q_{11} > q_{12} > \ldots > q_{1r_1}, q_{21} > q_{22} \ldots > q_{2r_2}, \ldots, q_{m1} > q_{m2} > \ldots > q_{mr_m}$$

- the third line is of the form above representing the preferences of B.

Please consider that a quality name is a string with more than 0 and less than 11 characters, that couples may declare opinions about at most 100 different qualities, and that is guaranteed that the given data is well defined with respect to the above rules. Also, it is guaranteed that the information corresponding to each person does not include preference cycles (i.e., each person is self-compatible).

The end of the input is recognized by a line with $A = B = *$.

The input must be read from standard input.

Output

For each given case, output one line with a single character `F`, `P`, or `N`, meaning that couple with spouses A and B is full compatible, passably compatible, or not compatible, respectively.

The output must be written to standard output.

Sample Input	**Sample Output**

`Alice1 Bob1`
`organized puzzles rich , windsurfer theatre , rich movies ;`
`kayaker movies puzzles , rich theatre ;`
`Alice2 Bob2`
`organized puzzles rich , windsurfer theatre ;`
`kayaker movies puzzles , rich theatre ;`
`Alice3 Bob3`
`young busy rich , wallet tennis , rich movies , busy toys ;`
`toys movies busy , rich tennis busy , rich movies ;`
`* *`
	`P`
	`F`
	`N`

C - Cargo Trains

Source file name: `cargo.c`, `cargo.cpp`, *or* `cargo.java`

The International Cargo and Packaging Company Inc. (ICPC Inc.) transports cargo between two cities: Source City and Sink City. ICPC Inc. does not have its own fleet, instead it contracts the service of two train companies, the company A and company B. Each company has its own network that connects some of the cities at prices that the company decides. For two given cities, it is possible that the route between them is served by both companies, only one company, or none. ICPC Inc. has reached an agreement with both companies that allows it to use their combined services at a discount price, but it has to follow these rules:

1. For a given shipment, ICPC Inc. must specify the percentage of participation of each company given by a for company A and $(1 - a)$ for company B, for a given real number a $(0 \leq a \leq 1)$.
2. When the segment between two cities is served by only one company, ICPC Inc. will pay the price corresponding to that company.
3. When the segment between two cities is served by both companies, the shipment can be shipped using a train from any of the two companies and will pay a fare equal to $a \times C_A + (1 - a) \times C_B$, where C_A and C_B correspond to the fares of company A and B respectively.
4. A shipment could pass through several intermediate cities. The total cost of the shipment corresponds to the sum of the costs of the individual segments between cities calculated according to rules 2 and 3.

ICPC Inc. needs your help to optimize its costs. Specificaly, ICPC Inc. needs to evaluate the cost of different alternatives that combine the participation of the two train companies in different proportions. Given the networks and prices of companies A and B, your task is to calculate the cost of k different combination alternatives. Each alternative is specified by the participation of company A, which corresponds to a real number $0 \leq a \leq 1$.

Input

The input contains multiple test cases. Each test case starts with a line with four integer values separated by spaces, n, m_a, m_b and k, that correspond to the number of cities, the number of edges in the network of company A, the number of edges in the network of company B, and the number of combination alternatives respectively. The ranges for the values are: $2 \leq n \leq 100$, $1 \leq m_a, m_b \leq 5000$, and $1 \leq k \leq 10000$.

The next m_a lines specify the network of company A. Each line has three integer values: N_i, N_j and $C_{i,j}$, separated by spaces, with $0 \leq N_i, N_j < n$ and $0 \leq C_{i,j} \leq 1000000$. The network is an undirected graph and each edge is only listed once. $C_{i,j}$ corresponds to the cost of sending one kilogram of cargo from city N_i to city N_j or the other way around. Source City corresponds to 0 and Sink City to $n - 1$. Then next m_b lines represent the graph corresponding to the network

of company B represented in the same way as the network of company A. The last k lines of the test case contain the different combinations to be evaluated, one combination per line. A combination is represented by a real number, $0 \leq a \leq 1$, with maximum 4 decimal digits. The value a specifies company A's participation. Company B's participation is implicitely defined and corresponds to $1 - a$. You can suppose that there is at least one path between the Source City and the Sink City using routes served by any company.

The end of the input is indicated by the line

```
-1 -1 -1 -1
```

The input must be read from standard input.

Output

For each combination alternative in each test case, the optimal cost of a trip from a city 0 to city $n - 1$ must be printed. If the answer has a decimal part, it has to be truncated without approximation.

The output must be written to standard output.

Sample Input	**Sample Output**
3 2 2 3 0 1 100 1 2 200 0 1 200 1 2 150 0 1 0.5 -1 -1 -1 -1	350 300 325

D - Dark Roads

Source file name: `dark.c`, `dark.cpp`, *or* `dark.java`

Economic times these days are tough, even in Byteland. To reduce the operating costs, the government of Byteland has decided to optimize the road lighting. Till now every road was illuminated all night long, which costs 1 Bytelandian Dollar per meter and day. To save money, they decided to no longer illuminate every road, but to switch off the road lighting of some streets. To make sure that the inhabitants of Byteland still feel safe, they want to optimize the lighting in such a way, that after darkening some streets at night, there will still be at least one illuminated path from every junction in Byteland to every other junction.

What is the maximum daily amount of money the government of Byteland can save, without making their inhabitants feel unsafe?

Input

The input file contains several test cases. Each test case starts with two numbers m and n, the number of junctions in Byteland and the number of roads in Byteland, respectively. Input is terminated by $m = n = 0$. Otherwise, $1 \leq m \leq 200000$ and $m - 1 \leq n \leq 200000$. Then follow n integer triples x, y, z specifying that there will be a bidirectional road between x and y with length z meters ($0 \leq x, y < m$ and $x \neq y$). The graph specified by each test case is connected. The total length of all roads in each test case is less than 2^{31}.

The input must be read from standard input.

Output

For each test case print one line containing the maximum daily amount the government can save.

The output must be written to standard output.

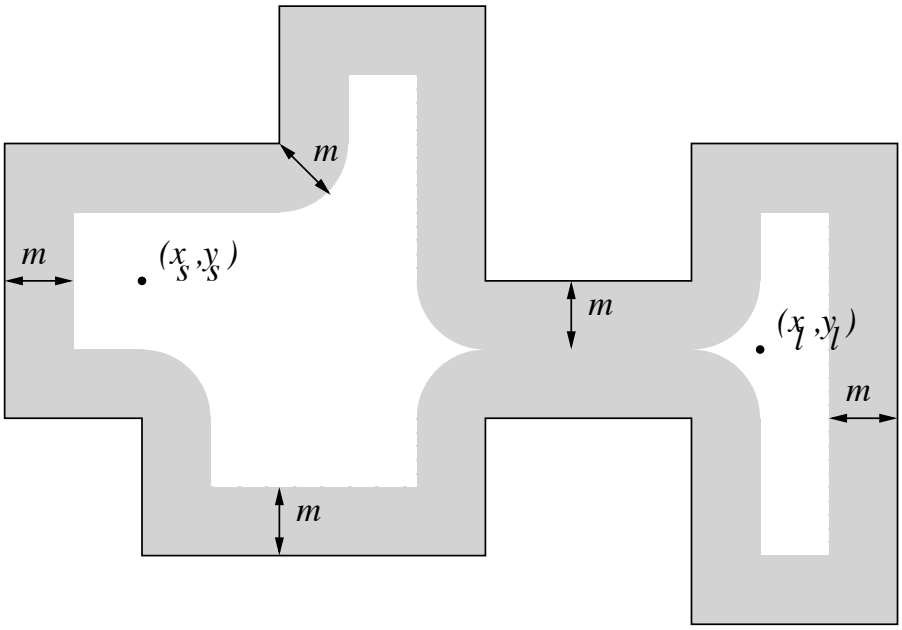
Sample Input	**Sample Output**
7 11 0 1 7 0 3 5 1 2 8 1 3 9 1 4 7 2 4 5 3 4 15 3 5 6 4 5 8 4 6 9 5 6 11 0 0	51

E - Message in Enemy Territory

Source file name: `enemy.c`, `enemy.cpp`, *or* `enemy.java`

A group of commandos has been caught and sent to a maximum-security prison in enemy territory. In order to escape from the prison, a soldier needs to give a message to the squadron leader.

The boundary of the prison is protected by electronic alarms: for his security, the soldier needs to keep a distance greater than m from the boundary. An additional restriction is that the soldier can only stand on those positions with integer coordinates. In each step, the soldier can move, from a given position (x, y), only to the nearby positions: $(x-1, y-1)$, $(x-1, y)$, $(x-1, y+1)$, $(x, y-1)$, $(x, y+1)$, $(x+1, y-1)$, $(x+1, y)$ and $(x+1, y+1)$, without going out of the interior of the prison. The walls of the prison form a simple polygon (no repeated vertices and no intersections between edges) and all of them are parallel to either the x-axis or the y-axis of a hypothetical coordinate system. The following figure shows a typical prison's plan:



Positions (x_s, y_s) and (x_l, y_l) corresponds to the position of the soldier and the squadron leader respectively. The gray area indicates those positions that are at distance less than or equal to m from the prison's boundary, i.e., the zone that the soldier cannot stand on.

A *safe* path is a sequence of pairs of integer coordinates, each one at a distance greater than m from the boundary of the prison, so that consecutive pairs are different and do not differ in more than one in each coordinate. In the depicted example, there is not a safe path from the soldier to the squadron leader.

Your task is to determine, for a given prison's plan, if there exists a safe path from the soldier position to the squadron leader position.

Input

The problem input consists of several test cases. Each test case consists of three lines:

- The first line contains two integer numbers separated by blanks, n and m, with $4 \leq n \leq 1000$ and $1 \leq m \leq 30$, indicating the number of the prison's boundary vertices and the alarm range respectively.
- The second line contains a list of $2 \cdot n$ integer numbers, $x_1, y_1, \ldots, x_n, y_n$, separated by blanks: the list of vertices of a simple n-polygon that describes the boundary of the prison. $0 \leq x_i, y_i \leq 1000$.
- The last line contains four integer numbers separated by blanks, x_s, y_s, x_l, and y_l, indicating the position of the soldier and the position of the squadron leader ($0 \leq x_s, y_s \leq 1000$, $0 \leq x_l, y_l \leq 1000$).

The end of the input is indicated by a line with ''`0 0`''.

The input must be read from standard input.

Output

For each test case the output includes a line with the word ''`Yes`'' if there exists a path from the soldier to the squadron leader. Otherwise the word ''`No`'' must be printed.

The output must be written to standard output.

Sample Input	**Sample Output**

`4 1`
`0 0 0 5 5 5 5 0`
`2 2 3 3`
`8 3`
`0 16 0 6 4 6 4 0 12 0 12 10 8 10 8 16`
`4 12 8 4`
`0 0`
	`Yes`
	`No`

F - Flatland

Source file name: `flat.c`, `flat.cpp`, *or* `flat.java`

Once upon a time there was *Flatland*, a world whose inhabitants believed was a 2D rectangular region. Flatlanders (the people inhabiting Flatland) assumed that if somebody traveled long enough in one direction, he/she would fall down over the edge of Flatland. Animated Caribbean Movies (ACM) plans to produce a film about Flatland. Before the script of the film is approved, ACM wants to become familiar with life as it was in Flatland by simulating how the world could evolve from given initial situations and some conditions determining life and death.

Flatlanders were nomads by nature as they were always traveling: all of them traveled at the same speed rate on straight lines but each individual had its own direction. As you may imagine, if one observed the life of a lonely Flatlander, he/she would eventually reach Flatland's edge and die. Nevertheless, if two Flatlanders collided, then their fate was improved as their directions changed: the resulting directions were as the former ones reflected in a mirror bisecting the angle between the former directions of the crashing inhabitants. So, a Flatlander survived because a collision with another one could change his/her direction.

However, there were bad news when more than two Flatlanders collided in a single crash: in that case all of them died, disappearing right there. Note that some Flatlanders could die at the same time. If this was the case, the last name of the list of deads was remembered as the *Last Dead One* in that moment (to simplify, we assume they used our modern English alphabet and lexicographic order). The survivors venerated the name of the Last Dead One until a new last dead appeared (and some Flatlander disappeared).

ACM's film begins with a given population in Flatland, where names, positions, and directions of every single individual are known. ACM wants you to help them to determine which would be the name of the last Last Dead One in the whole Flatland's life.

Input

There are NC test cases to solve, $0 < NC < 100$. The first line of the input file has NC. After that, for each testcase, a set of lines:

- the first line contains a number n, the number of Flatlanders in the initial world ($1 \leq n \leq 100$);
- the second line contains two positive integer numbers B and H separated by a space, representing the dimensions of Flatland ($2 \leq B, H \leq 100$). Coordinates in Flatland are points (i, j), with $0 \leq i \leq B$, $0 \leq j \leq H$. Flatland's edges are points with coordinates of the form $(0, j)$, $(i, 0)$, (B, j) or (i, H);
- n lines (one per Flatlander) with four numbers and one string: x, y, d_1, d_2 and *name* separated by a blank. (x, y) represents the position of the Flatlander ($0 < x < B$, $0 < y < H$, and two Flatlanders cannot start in the same position), (d_1, d_2) represents the direction:

(d_1, d_2) is a point on some Flatland's edge, so that the Flatlander is moving towards it; *name* is a string of one to 10 alphabetical uppercase characters, which represents the name of the Flatlander. You may assume that every Flatlander has a unique name.

The input must be read from standard input.

Output

For each given case, output one line with the name of the Last Dead One.

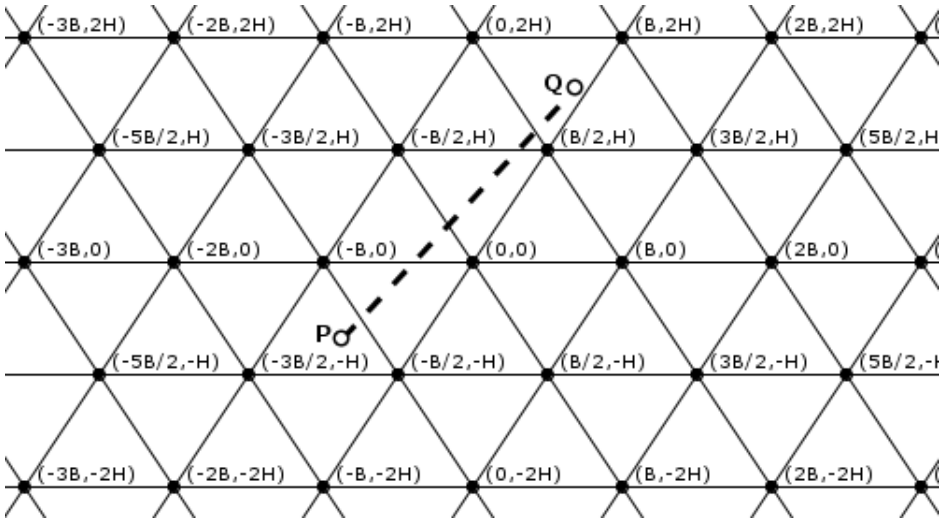
The output must be written to standard output.

Sample Input	**Sample Output**
2 2 20 23 1 1 0 0 BOB 3 3 3 0 ALICE 3 20 23 2 2 4 0 ALICE 4 2 2 0 BOB 1 3 0 3 CHARLES	ALICE BOB

G - Triangular Grid

Source file name: `grid.c`, `grid.cpp`, *or* `grid.java`

There is an infinite grid in the Cartesian plane composed of isosceles triangles, with the following design:



A *single triangle* in this grid is a triangle with vertices on intersections of grid lines that has not other triangles inside it.

Given two points P and Q in the Cartesian plane you must determine how many single triangles are intersected by the segment $\overline{PQ}$. A segment intersects a polygon if and only if there exists one point of the segment that lies inside the polygon (excluding its boundary).

Note that the segment $\overline{PQ}$ in the example intersects exactly six single triangles.

Input

The problem input consists of several cases, each one defined in a line that contains six integer values B, H, x_1, y_1, x_2 and y_2 ($1 \leq B \leq 200,\ 2 \leq H \leq 200,\ -1000 \leq x_1, y_1, x_2, y_2 \leq 1000$), where:

- B is the length of the base of all isosceles single triangles of the grid.
- H is the height of all isosceles single triangles of the grid.
- (x_1, y_1) is the point P, that defines the first extreme of the segment.
- (x_2, y_2) is the point Q, that defines the second extreme of the segment.

You can suppose that neither P nor Q lie in the boundary of any single triangle, and that $P \neq Q$.

The end of the input is specified by a line with the string ''`0 0 0 0 0 0`''.

The input must be read from standard input.

Output

For each case in the input, print one line with the number of single triangles on the grid that are intersected by the segment $\overline{PQ}$.

The output must be written to standard output.

Sample Input	**Sample Output**
100 120 -20 -100 160 160 10 8 5 5 5 4 10 8 5 5 10 5 10 8 5 5 10 10 0 0 0 0 0 0	6 1 2 3

H - Seasonal War

Source file name: `war.c`, `war.cpp`, *or* `war.java`

The inhabitants of Tigerville and Elephantville are engaged in a seasonal war. Last month, Elephantville successfully launched and orbited a spy telescope called the Bumble Scope. The purpose of the Bumble Scope was to count the number of War Eagles in Tigerville. The Bumble Scope, however, developed two problems because of poor quality control during its construction. Its primary lens was contaminated with bugs which block part of each image, and its focusing mechanism malfunctioned so that images vary in size and sharpness.

The computer programmers, who must rectify the Bumble Scope's problems are being held hostage in a Programming Contest Hotel in Alaland by elephants dressed like tigers. The Bumble Scope's flawed images are stored by pixel in a file called Bumble.in. Each image is square and each pixel or cell contains either a 0 or a 1. The unique Bumble Scope Camera (BSC) records at each pixel location a 1 if part or all of a war eagle is present and a 0 if any other object, including a bug, is visible. The programmers must assume the following:

1. A war eagle is represented by at least a single binary one.
2. Cells with adjacent sides on common vertices, which contain binary ones, comprise one war eagle. A very large image of one war eagle might contain all ones.
3. Distinct war eagles do not touch one another. This assumption is probably flawed, but the programmers are desperate.
4. There is no wrap-around. Pixels on the bottom are not adjacent to the top and the left is not adjacent to the right (unless, of course, there are only 2 rows or 2 columns).

Write a program that reads images of pixels from the input, correctly counts the number of war eagles in the images, and prints the image number and war eagle count for that image.

Input

The input consists of several test cases. The first line of each test case consists of a positive integer n, $n \geq 1$, defining the side of the square image. Then n lines follow: the ith line contains a sequence $a_1 \cdots a_n$ of bits describing the pixels of the ith row of the image.

The end of the input is given by $n = 0$.

The input must be read from standard input.

Output

For each test case, your program must output the image number in one line and the war eagle count for that image in the next line.

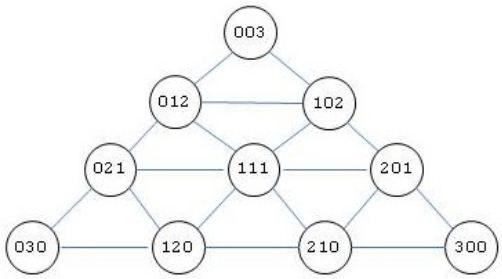
The output must be written to standard output.

Sample Input	**Sample Output**
6 100100 001010 000000 110000 111000 010100 8 01100101 01000001 00011000 00000010 11000011 10100010 10000001 01100000 0	Image number 1 contains 3 war eagles. Image number 2 contains 6 war eagles.

I - Y-Game

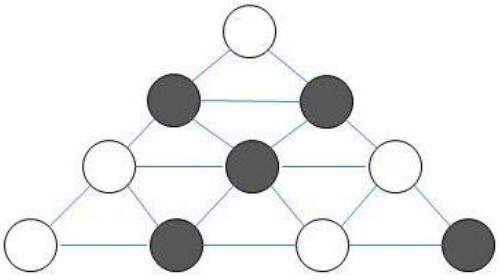
Source file name: `game.c`, `game.cpp`, *or* `game.java`

Willy and Benny enjoy very much playing *Y-game*! This is a game in which white and black tokens are placed on a triangular *n-grid*, $n \geq 0$, where n is called the order of the grid. A 3-grid is depicted in the figure below:



In general, an n-grid has $(n + 2)(n + 1)/2$ points with nonnegative ''baricentric coordinates'' (x, y, z), where $x + y + z = n$. Coordinates in a n-grid are assigned in such way that along right to left paths x-coodinates are constant, y-coordinates increase by one unit, and z-coordinates decrease by one unit (observe that this construction maintains $x + y + z = n$ true). Symmetric situations may be observed for left to right (where y-coordinates are constant) and horizontal (where z-coordinates are constant) paths. A point (x, y, z) in a n-grid is said to lay on the *x side* (resp., *y side*, *z side*) if and only if $x = 0$ (resp., $y = 0$, $z = 0$).

Willy uses white tokens and Benny uses black ones. Y-game rules are rather complicated, but the end of the game is attained when there is a token placed on every node of the grid. The winner is that player that has formed a *Y*, that is, his/her tokens are so placed that they include a connected set of points with a point on each side. For example, the following figure represents an end situation where Benny wins:



The winner is rather easy to determine when the grid is small. But Willy and Benny are not interested in that discussion today. Actually, they just want a software solution that computes the winner of ended *Y*-games. Could you help them?

Input

The problem input consists of several cases. A case begins with a line with two integer numbers, n and m, where n is the order of the grid and m the number of positions that have a black-coloured token (Benny's tokens), with $0 \leq n \leq 20$ and $0 \leq m \leq (n + 2)(n + 1)/2$.

Then, m lines follow, each one with 3 values x, y and z representing coordinate (x, y, z) of a point in the n-grid with a black token. Values on each input line are separated by one or more spaces.

The end of the input is signaled by a line

```
0 0
```

The input must be read from standard input.

Output

Output texts for each input case are presented in the same order that the input is read. For an input case in the puzzle statement, the output should be a single line with the left-justified text

```
Willy
```

or

```
Benny
```

depending on whether Willy or, respectively, Benny wins.

The output must be written to standard output.

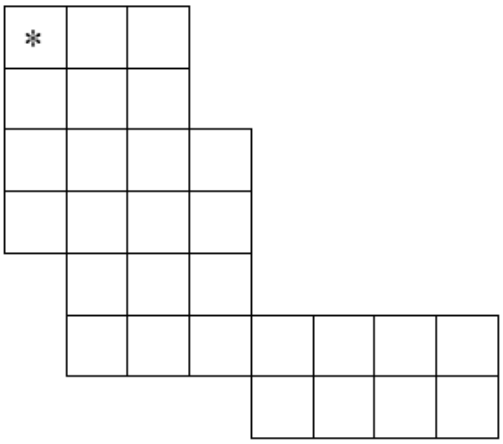
Sample Input	**Sample Output**
3 5 0 1 2 1 0 2 3 0 0 1 1 1 1 2 0 2 3 0 0 2 1 0 1 0 2 0 1 1 1 0 0 0 0	Benny Willy Willy

J - Making Jumps

Source file name: `jumps.c`, `jumps.cpp`, *or* `jumps.java`

A knight is a piece used in the game of chess. The chessboard itself is square array of cells. Each time a knight moves, its resulting position is two rows and one column, or two columns and one row away from its starting position. Thus a knight starting on row r, column c --which we'll denote as (r, c)-- can move to any of the squares $(r-2, c-1)$, $(r-2, c+1)$, $(r-1, c-2)$, $(r-1, c+2)$, $(r+1, c-2)$, $(r+1, c+2)$, $(r+2, c-1)$, or $(r+2, c+1)$. Of course, the knight may not move to any square that is not on the board.

Suppose the chessboard is not square, but instead has rows with variable numbers of columns, and with each row offset zero or more columns to the right of the row above it. The figure to the left illustrates one possible configuration. How many of the squares in such a modified chessboard can a knight, starting in the upper left square (marked with an asterisk), not reach in any number of moves?



If necessary, the knight is permitted to pass over regions that are outside the borders of the modified chessboard, but as usual, it can only move to squares that are within the borders of the board.

Input

There will be multiple cases to consider. The input for each case begins with an integer n, between 1 and 10, that specifies the number of rows in the modified chessboard. Following n there will be n pairs of blank separated integers, with the i-th pair corresponding to the i-th row of the chessboard. The first integer of each pair indicates the number of squares skipped at the beginning of the row. The second integer indicates the number of squares in the row (which will always be at least 1).

The last case will be followed by the integer 0.

For example, input for the case illustrated by the chessboard shown above would be:

```
7 0 3 0 3 0 4 0 4 1 3 1 7 4 4
```

The maximum dimensions of the board will be 10 rows and 10 columns. That is, any modified chessboard specified by the input will fit completely on a 10 row, 10 column board.

The input must be read from standard input.

Output

For each input case, display the case number (1,2,...), and the number of squares that the knight can not reach. Display the results in the format shown in the output sample below.

The output must be written to standard output.

Sample Input	**Sample Output**
7 0 3 0 3 0 4 0 4 1 3 1 7 4 4 3 0 3 0 3 0 3 2 0 1 2 1 0	Case 1, 4 squares can not be reached. Case 2, 1 square can not be reached. Case 3, 0 squares can not be reached.