ACM International Collegiate Programming Contest
2010 East Central Regional Contest
Grand Valley State University
University of Cincinnati
University of Windsor
Youngstown State University
October 23, 2010

Sponsored by IBM

Rules:

1. There are **eight** problems to be completed in **five hours**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. The allowed programming languages are C, C++ and Java.
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
6. The input to all problems will consist of multiple test cases.
7. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
8. All communication with the judges will be handled by the PC^2 environment.
9. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: Cut It Out!

Television's *Toddler Time*'s topic taught to toddler Tommy Tiwilliger today was triangles (and the letter T)! Tommy became so enamored by triangles that immediately after the show ended, he grabbed his safety scissors and the nearest sheets of paper he could find and started cutting out his own triangles. After about 15 minutes each paper had one triangular shaped hole cut out of it. But Tommy wasn't finished yet. He noticed he could divide each of the original triangles into two triangles with a single cut starting from one corner. He spent another 15 minutes doing just that. Things would have gone along swimmingly, except that Tommy's mother eventually came into the room and noticed that the original sheets of paper were part of a very important document for a legal case she was working on (involving a lover's triangle). After carefully removing Tommy from the room and counting slowly to 10 (a triangular number), she went about trying to reconstruct the pages after gathering together the now randomly scattered triangles. Your job is to help her by writing a little program to determine which triangles go where (and try to get it done before tomorrow's episode of *Toddler Time* on papier-mâchè).

Input

Each test case will start with an integer $n \leq 20$ indicating the number of holes cut out, followed by the coordinates of the holes, one hole per line. These holes are assumed to be numbered $1, 2, \ldots, n$. Following this will be the coordinates of the $2n$ triangles resulting from the bisections, one triangle per line. These triangles are assumed to be numbered $1, 2, \ldots, 2n$ and are listed in no particular order. The specification of any hole or triangle will have the form $x_1\ y_1\ x_2\ y_2\ x_3\ y_3$ where each x_i and y_i will be to the nearest thousandth and $|x_i|, |y_i| \leq 200$. No two holes will be congruent and no two triangles will be congruent. A value of $n = 0$ will terminate input.

Output

For each test case, you should output the case number and then n lines as follows:

```
Hole 1:  t1a, t1b
Hole 2:  t2a, t2b
...
Hole n:  tna, tnb
```

where `t1a, t1b` are the two triangles which fill hole 1, `t2a, t2b` are the two triangles which fill hole 2, etc. Always print the lower of the two numbers first on any line. Triangles should not be flipped over when filling a hole. Each test case will have a unique solution. Separate the output for each test case with a blank line. Note: when processing the triangles and checking for equality of lengths, angles or trigonometric values, you may assume that two items are equal if they differ by less than 0.01.

Sample Input

```
1
18.691 6.103 21.668 13.709 21.332 25.894
59.388 30.873 55.299 36.186 61.45 22.97
67.828 85.496 60.751 72.752 59.2 67.49
3
18.73 4.012 6.662 7.557 14.035 7.478
14.869 32.398 32.341 31.772 7.522 29.674
25.272 6.868 4.572 2.014 10.487 16.121
26.135 53.073 44.18 50.723 40.31 42.91
86.601 29.95 70.542 17.088 66.77 14.88
90.344 89.528 92.179 88.665 87.99 82.54
39.327 62.11 35.033 57.127 18.14 63.89
37.13 80.202 36.308 75.111 34.28 75.11
14.043 68.482 15.22 55.423 10.42 75.43
0
```

Sample Output

```
Case 1:
Hole 1: 1, 2

Case 2:
Hole 1: 3, 5
Hole 2: 2, 6
Hole 3: 1, 4
```

Problem B: Flip It!

Assume you have a set of cards laid out in an n by m grid. The cards are numbered and some are face up and others are face down. We can collapse the grid into a single pile by using a series of flips, each of which is one of the four following types:

Top Flip : Here the cards in the top row are flipped over onto the corresponding cards on the row beneath them. Note that if a card is face up in the top row, it becomes face down after the flip, and vice versa. If the top row contains one or more piles of cards, each entire pile is flipped over like a stack of pancakes as it is moved to the lower row.

Bottom Flip : Same as the Top Flip, but now the bottom row is flipped onto the next-to-bottom row.

Left Flip : Flip the cards in the left-most column onto the next-to-leftmost column.

Right Flip : Flip the cards in the rightmost column onto the next-to-rightmost column.

After a series of $n + m - 2$ flips, the cards will be in a single pile, some cards face up and some face down. Your job is to determine the order of the face up cards in this final pile.

Input

Each test case will start with a line containing two positive integers n m indicating the number of rows and columns in the grid. After this will come n rows of m integers indicating each card's number and its orientation. (The first row is the top row and the first value in each row is the leftmost card.) If a value is a positive integer k that means card k is at the location face up; if a value is a negative integer -k that means card k is at the location face down. (k will never be zero.) After these n rows there will be one more line containing $n + m - 2$ characters indicating the order of flips to apply to the grid. Each character will be either `T`, `B`, `L` or `R` corresponding to a top, bottom, left or right flip. All flip sequences will be legal, i.e., you won't be asked to do more than $n - 1$ top and bottom flips or $m - 1$ left and right flips. The maximum value for n and m is 20. A line containing two zeros will terminate input.

Output

For each test case, output the case number followed by a list of the numbers of all of the face up cards in the final deck, starting from the bottom of the deck. Follow the format used in the examples.

Sample Input

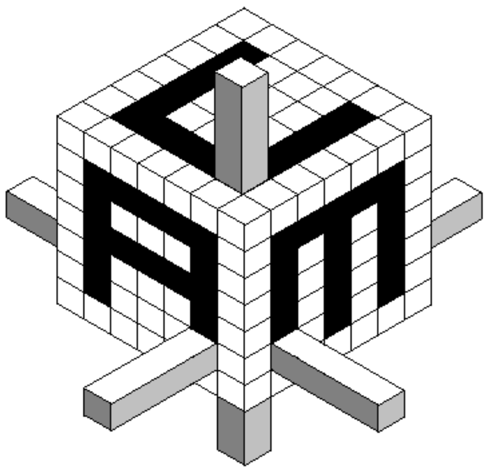
```
2 3
4 -17 -8
6 23 -5
LRB
1 1
-3

1 1
3

0 0
```

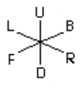
Sample Output

```
Case 1: 8 6
Case 2:
Case 3: 3
```

Problem C: Maze

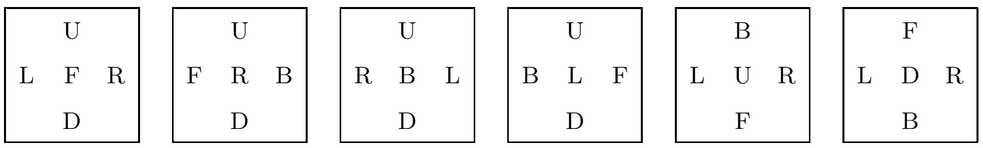
We are all familiar with conventional mazes laid out on a 2-D grid. A 3-D maze can be constructed as follows: Consider a hollowed out cube aligned along the x, y and z axes with one corner at (0,0,0) and the opposite corner at $(n-1, n-1, n-1)$. On each face of the cube is a 2-D maze made by removing a subset of $1 \times 1 \times 1$ cubes from the face (no edge cubes are removed). The object of the maze is to move a marker located inside the cube from an initial location of (1,1,1) to the final destination of ($n-2$,$n-2$,$n-2$). However, attached to this marker are 6 rods, each protruding through one face of the cube. The movement of these rods is constrained by the 2-D mazes on the faces. The picture below gives an example of a $7 \times 7 \times 7$ maze. Note that this maze is not physically realizable since some faces (e.g., the front face containing the letter "A") have cubes that are disconnected from the edges of the face. Such mazes are allowed in this problem.





The black regions indicate open spaces where the rods can move. The figure to the right specifies the possible directions that the rods can move (Forward, Back, Left, Right, Up, Down) and also defines the labels for the six sides of the cube. In the maze above, the rods are shown in their initial position centered at (1,1,1). From here they can not move Forward, Backward, Right, Left or Down, but they can move Up (assuming there are open spaces for the two back rods to move to).

To specify a cube, a description of each face must be given. For this problem, the order and orientations of each face are given in the diagram below.



The first square represents the Forward face oriented so that the shared edge with the Up face is on top and the shared edge with the Right face is on the right, the second square represents the Right face oriented with the shared edge with the Up face on top and the shared edge with the Back face on the right, and so on. Your job is to solve such mazes in the minimum number of moves.

Input

Each test case will start with a single line containing the value of n, where n lies between 4 and 30, inclusive. Next will come descriptions of each face in the order and orientation shown above. The description of each face will consist of n lines each containing one string of length n. The characters in the string will be either a blank or `'X'`, indicating either an empty or full square, respectively. The last test case will be followed by a line containing 0.

Output

Output will consist of one line for each test case. Each line will contain a description of a minimum-move solution that moves the marker from cell (1,1,1) to cell $(n-2, n-2, n-2)$. Moves are either `F, B, L, R, U` or `D`. In case of a tie, choose the sequence of moves which is lexicographically first, where we consider `F` $<$ `B` $<$ `L` $<$ `R` $<$ `U` $<$ `D`. All mazes will have solutions.

Sample Input

```
7
XXXXXXX
X     X
X XXX X
X     X
X XXX X
X XXX X
XXXXXXX
XXXXXXX
X     X
X X X X
X X X X
X X X X
X X X X
XXXXXXX
XXXXXXX
X     X
X     X
X     X
X     X
X     X
XXXXXXX
```

(continued in next column)

```
XXXXXXX
X     X
X X X X
X     X
X X X X
X     X
XXXXXXX
XXXXXXX
X XXX X
X XXX X
X XXX X
X XXX X
X     X
XXXXXXX
XXXXXXX
X     X
X XXX X
X X X X
X XXX X
X     X
XXXXXXX
0
```

Sample Output

```
UULLLLUUBBBB
```

Problem D: Photo Shoot

Adam Ansels is a photographer who specializes in impromptu photos of his clients. At the moment Adam is standing in the middle of a field surrounded by a large group of people. The camera Adam is using has a set field-of-view angle f, which means that if he points the camera in a direction d (measured in degrees from the x-axis) anything within the range $d - f/2$ to $d + f/2$ will be in the picture. Even with digital photography, Adam still likes to take as few pictures as possible. Given the locations of the people around Adam and the field-of-view angle, Adam would like to know the minimum number of photos he needs to take to make sure that everyone is in at least one photo. (Well, almost everyone: Some people insist on standing in a spot where they are blocked from Adam's camera by another person. So, only the arms and top of their head will show up. There is just so much poor Adam can do. Those are the breaks.)

Input

Each test case will start with a line containing four integers n x y f indicating the number of people surrounding Adam ($n \geq 0$), the location of Adam (x, y), and the field-of-view of his camera ($f > 0$), measured in degrees. The maximum value for n, $|x|$, and $|y|$ is 100 and for f is 180. After this will be n pairs of numbers x_i y_i indicating the locations of the n people ($|x_i|, |y_i| \leq 1000$). No two people (including Adam) will be standing in the same spot. All locations use the standard Cartesian x-y coordinate system. A row containing four zeros will terminate input.

Output

For each test case, output the case number followed by the minimum number of photos Adam needs to take to ensure everyone is in at least one picture. You may assume that no two people are exactly f degrees apart from each other relative to Adam. Follow the format used in the examples.

Sample Input

```
6 5 5 90
1 4 5 10 6 9 7 4
8 6 10 6
20 20 20 180
1 21 3 21 5 21 7 21 9 21 11 21 13 21 15 21 17 21 19 21
21 21 23 21 25 21 27 21 29 21 31 21 33 21 35 21 37 21 39 21
0 0 0 0
```

Sample Output

```
Case 1: 3
Case 2: 1
```

Problem E: Polar Bear

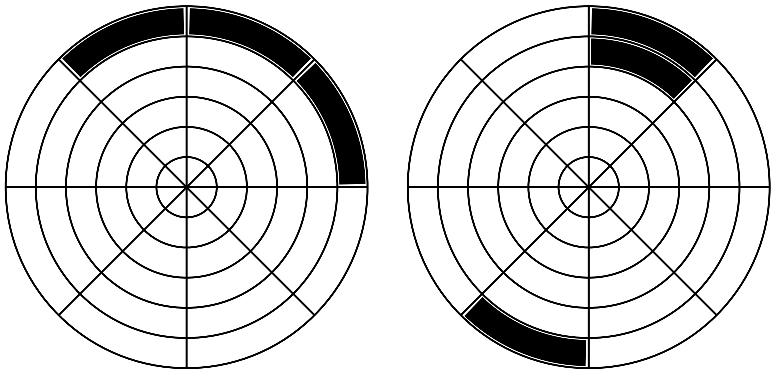
"Hey, John, you know John Conway's Game of Life?"

"You mean the one where you have a rectangular grid of cells, some alive, some dead, and cells all simultaneously update their status at regular clock ticks to form a new generation of cells, using the simple rules that a dead cell with exactly three live neighbors (the neighbors of a cell are the eight cells surrounding it) becomes live in the next generation, a live cell with fewer than 2 or more than 3 neighbors become dead in the next generation, and other cells' status remain unchanged?"

"Yeah, that's the one. I've been trying to program it ..."

"Big deal – everybody has done that!'

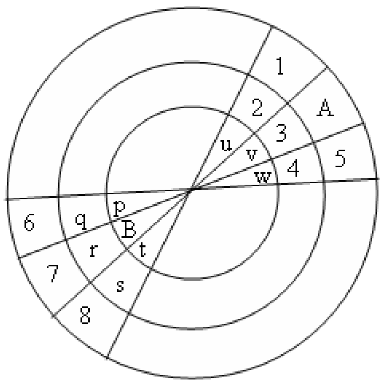
"Wait, let me finish – I've been trying to program it to work on polar coordinate graph paper, like this:"



Two generations of Polar Life

"Hmmm. That looks like a real bear."

"It's not so bad – most of the cells have eight neighbors, just like Conway's game. The only tricky ones are the ones on the outer circumference and the wedge-shaped cells in the center. But I think I've figured out what to do with them. If the number of radial lines is even, then every cell on the outer and inner rings has eight neighbors: the five ordinary neighbors on the ring and the nearest adjacent ring, plus the cell diametrically opposite on the grid, plus the left and right neighbors of that cell. For instance:"



Neighbors in Polar Life

"The neighbors of A are the cells numbered 1 through 8, and the neighbors of B are the cells labeled p through w."

"What about an odd number of radial lines?"

"Don't ask."

Input

Input will consist of multiple problem instances. Each problem instance begins with two positive integers m and n, $3 \leq m \leq 100$, $6 \leq n \leq 100$, where m is the number of rings and n is the number of radii (in the first figure there are 6 rings and 8 radii). The value of n is always even. Following this is a positive integer k and a list of k distinct pairs of positive integers, which may extend over several lines. Each pair denotes the location of a live cell, where the first integer indicates a ring (counting from the outer ring inwards, starting at 0) and the second integer indicates the cell number in this ring (starting at 0 and going clockwise around the ring, beginning at some arbitrary but fixed radius line). This is followed by a nonnegative integer $g \leq 500$, indicating the number of generations. The last test case is followed by two zeros.

Output

For each test case, print the test case number followed by five integers: the number of live cells after g iterations of the Game of Life rules; the location r_1 c_1 of the first live cell, and the location r_2 c_2 of the last live cell, where first and last are relative to the lexicographic ordering of ring and cell numbers of the live cells. If no cells have survived, the output should be `0 -1 -1 -1 -1`.

Sample Input

```
6 8
3
0 7 0 0
0 1
1
4 6
1
1 0
10
0 0
```

Sample Output

```
Case 1: 3 0 0 1 0
Case 2: 0 -1 -1 -1 -1
```

Problem F: Pro-Test Voting

Old Bob Test is currently running for Mayor in the Hamlet of Kerning. Kerning is divided up into a number of precincts (numbered 0, 1, 2, ...), and after extensive polling by his crack staff, Bob knows the current percentage of voters in each precinct who plan to vote for him. Needless to say, he would like to increase these percentages in all precincts, but he has limited funds to spend. Based on past results, the effects of spending on any precinct obey the following equation:

$$F_p = I_p + \left(\frac{M}{10.1 + M}\right)\Delta$$

where I_p is the current percentage of pro-Test voters, Δ is the maximum increase in this percentage possible, M is the amount of money spent in the precinct, in integer multiples of \$1, and F_p is the final expected percentage. What Bob needs to know is the best way to spend his money to maximize the number of votes he can get.

Input

The first line of each test case contains two integers m and n, representing the amount of money Bob has to spend (in dollars) and the number of precincts. The maximum value for both of these is 100. After this will be n lines of the form N I_p Δ, all positive integers, which contain information on each precinct: N is the population of the precinct and I_p and Δ are as described above. The value of N will be less than 10000. The first of these lines refers to precinct 0, the next to precinct 1, and so on. A line containing 0 0 follows the last test case. NOTE: When calculating the number of pro-Test voters in a precinct, you should first perform a double calculation of F_p using the formula above, then multiply this percentage by the population N and round to get the final result.

Output

Output for each test case should consist of two lines. The first should contain the case number followed by the maximum number of votes Bob can obtain through optimum spending. The second line should list each precinct and the amount of money which Bob should spend there. The format for each precinct should be `precinctnum:money`, and each such pair should be separated by 1 blank. In the case where there is more than one way to spend Bob's money that yields the maximum number of votes, give the one that spends the most on precinct 0. If there is more than one with the same spent on precinct 0, take the one that spends the most on precinct 1, etc.

Sample Input

```
100 2
3000 45 15
2000 60 10
100 3
3000 45 15
2000 60 10
4000 20 8
100 3
3000 45 15
2000 60 10
4000 20 7
100 3
3000 45 15
2000 60 10
4000 20 6
100 3
3000 45 15
2000 60 10
4000 20 5
0 0
```

Sample Output

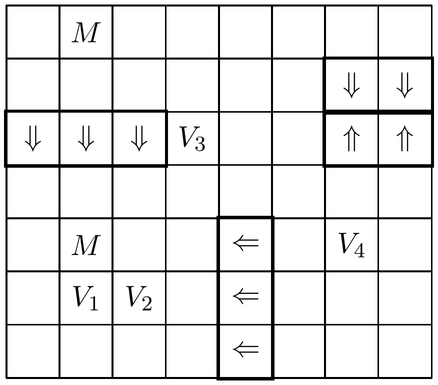
```
Case 1: 3095
0:64 1:36
Case 2: 4101
0:42 1:24 2:34
Case 3: 4070
0:45 1:27 2:28
Case 4: 4040
0:46 1:27 2:27
Case 5: 4011
0:46 1:27 2:27
```

Problem G: Vampires!

The big media gimmick these days seems to be vampires. Vampire movies, vampire television shows, vampire books, vampire dolls, vampire cereal, vampire lipstick, vampire bunnies – kids and teenagers and even some adults spend lots of time and money on vampire-related stuff. Surprisingly, nowadays vampires are often the good guys. Obviously, the ACM Programming Contest had better have a vampire problem in order to be considered culturally relevant.

As eveyone knows, vampires are allergic to garlic, sunlight, crosses, wooden stakes, and the Internal Revenue Service. But curiously they spend a good part of their time smashing mirrors. Why? Well, mirrors can't hurt vampires physically, but it's embarrassing to be unable to cast a reflection. Mirrors hurt vampire's feelings. This problem is about trying to help them avoid mirrors.

In a room full of vampires and ordinary mortals there are a number of mirrors. Each mirror has one of four orientations – north, south, east, or west (the orientation indicates which side of the mirror reflects). A vampire is in danger of embarrassment if he or she is in a direct horizontal or vertical line with the reflecting side of a mirror, unless there are intervening objects (mortals or other mirrors). For example, in the following room layout


M
$\Downarrow$ $\Downarrow$
$\Downarrow$ $\Downarrow$ $\Downarrow$ V_3 $\Uparrow$ $\Uparrow$
M $\Leftarrow$ V_4
V_1 V_2 $\Leftarrow$
$\Leftarrow$

vampire V_2 is exposed to a south-facing mirror and both vampires V_1 and V_2 are exposed to a west-facing mirror (note that a vampire can't protect another vampire from embarrassment since neither one casts a reflection.) Your job is to notify each vampire of the directions in which there is danger of experiencing ENR (embarrassing non-reflectivity).

Input

Each test case begins with three integers v o m, indicating the number of vampires, ordinary mortals, and mirrors in the room. Each of the following v lines contains a pair of integers x, y, $0 \leq x, y \leq 100$, giving the grid square of a vampire ($x = 0$ corresponds to the westernmost side of the grid and $y = 0$ corresponds to the southernmost side). Each of the following o lines contains the grid square of an ordinary mortal in the same format. Each of the following m lines contains a letter (either N, S, E, or W) indicating the orientation of a mirror, followed by four integers x_1 y_1 x_2 y_2 indicating the start and end squares of the mirror (same bounds as above). Mirrors can be of any positive length, have a thickness of 1 grid square, and will always be aligned along either the east-west or north-south axis. Each grid square contains no more than 1 vampire, mortal, or mirror section. The last test case is followed by a line containing 0 0 0.

Output

For each test case print the case number followed by one line for each vampire that is in danger of embarrassment. On each of these lines, print the vampire's number (vampires are numbered from 1 to v in order of appearance in the input) followed by a list of directions to avoid. For instance, if a vampire is exposed to a south-facing mirror and an east-facing mirror, he/she is exposed in the directions north and west. Directions should be printed in alphabetical order (e.g., `north west`, not `west north`). If no vampires are suffering from embarrassment, output the word `none`. Imitate the sample output.

Sample Input

```
4 2 4
1 1
2 1
3 4
6 2
1 2
1 6
S 0 4 2 4
W 4 2 4 0
N 6 4 7 4
S 6 5 7 5
1 0 2
20 20
W 30 10 30 30
N 25 20 27 20
0 0 0
```

Sample Output

```
Case 1:
vampire 1 east
vampire 2 east north
Case 2:
none
```

Problem H: We've Got Chemistry, Babe

Barry Liam is a chemistry student at Uranium University (good ol' UU), and does pretty well except in one area — balancing chemical equations. As I'm sure you recall from your last chemistry class, a chemical reaction can be represented by a chemical equation, with the reactants on the left and the products on the right, separated by an arrow. For example, the chemical equation to form aluminum oxide out of aluminum and oxygen is

$$\mathrm{Al} + \mathrm{O_2} \rightarrow \mathrm{Al_2O_3}.$$

However, the above equation is not balanced - there is one aluminum (Al) atom on the left hand side and two on the right, and there are two oxygen (O) atoms on the left and three on the right. A properly balanced version of this equation would be:

$$4\mathrm{Al} + 3\mathrm{O_2} \rightarrow 2\mathrm{Al_2O_3}.$$

Note that not only are all the coefficients positive integers, but the greatest common divisor of all of them is 1 (two requirements when balancing). While Barry understands all this, he is flummoxed when asked to do it, especially when asked to balance equations such as:

$$\mathrm{Fe_2(SO_4)_3} + \mathrm{KSCN} \rightarrow \mathrm{K_3Fe(SCN)_6} + \mathrm{K_2SO_4}.$$

Notice here that atoms in parentheses are all multiplied by the number that follows (so in this equation there are 12 oxygen atoms on the left and six carbon (C) atoms on the right, for example). Also, a symbol appears at most once in any reactant or product. Barry would like to ask his roommate for help, but he's a Ballet major who thinks balancing equations have something to do with dancing en pointe (actually, he thought Barry said "Balanchine equations"). I guess Barry will have to look for a new major, unless you can help him.

Input

Each test case will consist of a single line starting with two positive integers r and p indicating the number of reactants and products (the maximum value for each is 10). There will then follow the r reactants. Each reactant will consist of one or more terms, where each term is either a single chemical symbol, a single chemical symbol followed by an integer ≥ 2, or a set of parentheses around a term, followed by an integer ≥ 2. All chemical symbols consist of one or two alphabetic characters, only the first of which is capitalized. Following the r reactants will be the p products which follow the same rules as the reactants. There will be no more than 20 different chemical symbols in any test case. A line containing two zeros will terminate the input.

Output

For each test case, you should output the appropriate coefficients needed to balance the equation, listed in the order that the reactants and products are given. Use a single space to separate the values, and preface each with the case number, using the format shown below. If an equation cannot be balanced correctly or uniquely (see the last sample input, which has solutions 1 2 1 1, 1 3 1 2, 1 4 1 3, etc.), you should output the word `No`.

Sample Input

```
2 1 Al O2 Al2O3
2 2 Fe2(SO4)3 KSCN K3Fe(SCN)6 K2SO4
1 1 CO2 CO
2 2 A B AB B
0 0
```

Sample Output

```
Case 1: 4 3 2
Case 2: 1 12 2 3
Case 3: No
Case 4: No
```