Problem A
Millionaire

Congratulations! You were selected to take part in the TV game show *Who Wants to Be a Millionaire*! Like most people, you are somewhat risk-averse, so you might rather take \$250,000 than a 50% chance of winning \$1,000,000. On the other hand, if you happen to already be rich, then you might as well take a chance on the latter. Before appearing on the show, you want to devise a strategy to maximize the expected *happiness* derived from your winnings.

More precisely, if your present net worth is W dollars, then winning v dollars gives you $\ln(1+v/W)$ units of happiness. Thus, the game's *expected happiness* is $\sum_v P(v)\ln(1+v/W)$, where $P(v)$ is the probability that you'll win v dollars, and the summation is taken over all possible values of v. Since happiness units are too abstract, you will be asked to measure the value of the game in dollars. That is, compute D such that a guaranteed payout of D dollars makes you as happy as a chance on the show, assuming optimal play.

On the show, you will be presented with a series of questions on trivia, each associated with a prize value of v_i dollars. Your analysis of past episodes reveals that if you attempt the ith question, your chances of being correct are p_i.

After answering correctly, you may choose to continue or to quit. If you quit, you win the value of the last correctly answered question; otherwise, the game continues and you must attempt the next question. If you correctly answer all the questions, you walk away with the value of the last question.

If you answer a question incorrectly, however, the game ends immediately and you win the value of the last correctly answered question that is labeled as `safe`, or nothing if you never solved a `safe` question.

For example, the game in the first sample input is worth $0.5\ln(1+5000/4000) \approx 0.405$ units of happiness. Getting \$2,000 would likewise grant $\ln(1+2000/4000) \approx 0.405$ happiness.

Input

The first line of input contains two space-separated integers n and W ($1 \le n \le 10^5, 1 \le W \le 10^6$). Line $i+1$ describes the ith question. It starts with a string, which is one of `safe` or `unsafe`, indicating whether the ith question is safe or not. The string is followed by a real number p_i and an integer v_i ($0 \le p_i \le 1, 1 \le v_i < v_{i+1} \le 10^6$).

Output

Print, on a single line, a `$` sign immediately followed by D, rounded and displayed to exactly two decimal places. See the samples for format clarification.

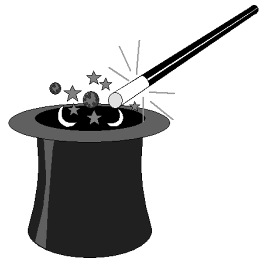
Sample Input	Sample Output
<pre>1 4000 unsafe 0.5 5000</pre>	<pre>$2000.00</pre>

Sample Input	Sample Output
<pre>4 4000 unsafe 1 2000 safe 0.4 5000 unsafe 0.75 10000 safe 0.05 1000000</pre>	<pre>$2316.82</pre>

Sample Input	Sample Output
<pre>2 4000 safe 0.003 1 safe 0.03 10</pre>	<pre>$0.00</pre>

Problem B

Magic Trick



Your friend has come up with a math trick that supposedly will blow your mind. Intrigued, you ask your friend to explain the trick.

First, you generate a random positive integer k between 1 and 100. Then, your friend will give you n operations to execute. An operation consists of one of the four arithmetic operations `ADD`, `SUBTRACT`, `MULTIPLY`, or `DIVIDE`, along with an integer-valued operand x. You are supposed to perform the requested operations in order.

You don't like dealing with fractions or negative numbers though, so if during the process, the operations generate a fraction or a negative number, you will tell your friend that he messed up.

Now, you know the n operations your friend will give. How many of the first 100 positive integers will cause your friend to mess up?

Input

The first line of input contains a single positive integer n ($1 \leq n \leq 10$). Each of the next n lines consists of an operation, followed by an operand. The operation is one of the strings `ADD`, `SUBTRACT`, `MULTIPLY`, or `DIVIDE`. Operands are positive integes not exceeding 5.

Output

Print, on a single line, a single integer indicating how many of the first 100 positive integers will result in you telling your friend that he messed up.

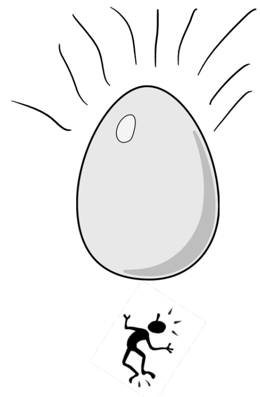
Sample Input	Sample Output
1 SUBTRACT 5	4

Sample Input	Sample Output
1 DIVIDE 2	50

Sample Input	Sample Output
2 ADD 5 DIVIDE 5	80

Problem C

Egg Drop



There is a classic riddle where you are given two eggs and a k-floor building and you want to know the highest floor from which you can drop the egg and not have it break.

It turns out that you have stumbled upon some logs detailing someone trying this experiment! The logs contain a series of floor numbers as well as the results of dropping the egg on those floors. You need to compute two quantities—the lowest floor that you can drop the egg from where the egg could break, and the highest floor that you can drop the egg from where the egg might not break.

You know that the egg will not break if dropped from floor 1, and will break if dropped from floor k. You also know that the results of the experiment are consistent, so if an egg did not break from floor x, it will not break on any lower floors, and if an egg did break from floor y, it will break on all higher floors.

Input

The first line of input contains two space-separated integers n and k ($1 \leq n \leq 100$, $3 \leq k \leq 100$), the number of egg drops and the number of floors of the building, respectively. Each of the following n lines contains a floor number and the result of the egg drop, separated by a single space. The floor number will be between 1 and k, and the result will be either `SAFE` or `BROKEN`.

Output

Print, on a single line, two integers separated by a single space. The first integer should be the number of the lowest floor from which you can drop the egg and it could break and still be consistent with the results. The second integer should be the number of the highest floor from which you can drop the egg and it might not break.

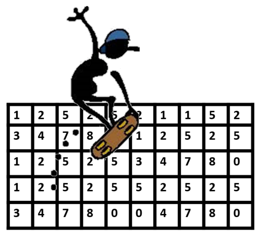
Sample Input	Sample Output
2 10 4 SAFE 7 BROKEN	5 6

Sample Input	Sample Output
3 5 2 SAFE 4 SAFE 3 SAFE	5 4

Sample Input	Sample Output
4 3 2 BROKEN 2 BROKEN 1 SAFE 3 BROKEN	2 1

Problem D

Grid



You are on the top left square of an $m \times n$ grid, where each square on the grid has a digit on it. From a given square that has digit k on it, a *move* consists of jumping exactly k squares in one of the four cardinal directions. What is the minimum number of moves required to get from the top left corner to the bottom right corner?

Input

The first line of input contains two space-separated positive integers m and n ($1 \leq m, n \leq 500$). It is guaranteed that at least one of m and n is greater than 1. The next m lines each consists of n digits, describing the $m \times n$ grid. Each digit is between 0 and 9.

Output

Print, on a single line, a single integer denoting the minimum number of moves needed to get from the top-left corner to the bottom-right corner. If it is impossible to reach the bottom-right corner, print `IMPOSSIBLE` instead.

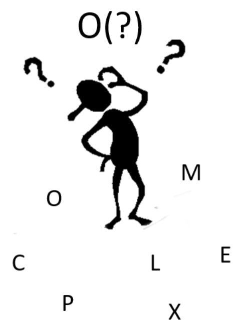
Sample Input	Sample Output
2 2 11 11	2

Sample Input	Sample Output
2 2 22 22	IMPOSSIBLE

Sample Input	Sample Output
5 4 2120 1203 3113 1120 1110	6

Problem E

Complexity



Define the *complexity* of a string to be the number of distinct letters in it. For example, the string `string` has complexity 6 and the string `letter` has complexity 4.

You like strings which have complexity either 1 or 2. Your friend has given you a string and you want to turn it into a string that you like. You have a magic eraser which will delete one letter from any string. Compute the minimum number of times you will need to use the eraser to turn the string into a string with complexity at most 2.

Input

The input consists of a single line that contains a single string of at most 100 lowercase ASCII letters (‘`a`’–‘`z`’).

Output

Print, on a single line, the minimum number of times you need to use the eraser.

Sample Input	Sample Output
`string`	`4`

Sample Input	Sample Output
`letter`	`2`

Sample Input	Sample Output
`aaaaaa`	`0`

Sample Input	Sample Output
`uncopyrightable`	`13`

Sample Input	Sample Output
`ambidextrously`	`12`

Sample Input	Sample Output
`assesses`	`1`

Sample Input	Sample Output
`assassins`	`2`

Problem F
KenKen You Do It?

KenKen is a popular logic puzzle developed in Japan in 2004. It consists of an $n \times n$ grid divided up into various non-overlapping sections, where each section is labeled with an integer target value and an arithmetic operator. The object is to fill in the entire grid with the numbers in the range 1 to n such that

- no number appears more than once in any row or column
- in each section you must be able to reach the section's target using the numbers in the section and the section's arithmetic operator

For this problem we are only interested in single sections of a KenKen puzzle, not the entire puzzle. Two examples of sections from an 8×8 KenKen puzzle are shown below along with some of their possible assignments of digits.

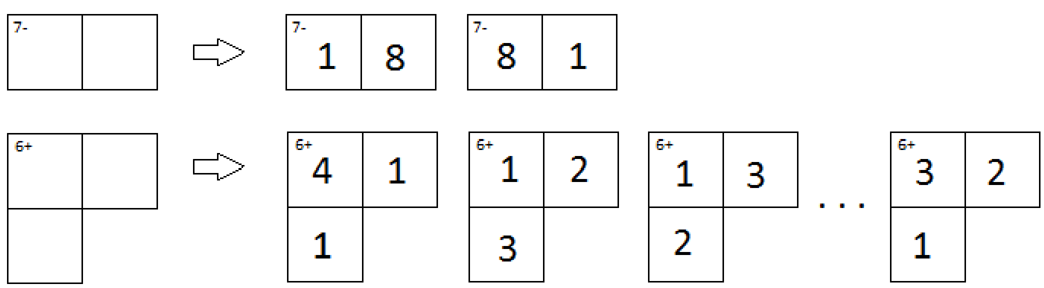


Figure C.1

Note that while sections labeled with a subtraction or division operator can consist of only two grid squares, those labeled with addition or multiplication can have any number. Also note that in a 9×9 puzzle the first example would have two more solutions, each involving the numbers 9 and 2. Finally note that in the first solution of the second section you could not swap the 1 and 4 in the first row, since that would result in two 1's in the same column.

You may be wondering: for a given size KenKen puzzle and a given section in the puzzle, how many valid ways are there to fill in the section? Well, stop wondering and start programming!

Input

The input will start with a single line of the form n m t op, where n is the size of the KenKen puzzle containing the section to be described, m is the number of grid squares in the section, t is the target value and op is either '+', '-', '*' or '/' indicating the arithmetic operator to use for the section.

Next will follow m grid locations of the form r c, indicating the row and column number of the grid square. These grid square locations will take up one or more lines.

All grid squares in a given section will be connected so that you can move from any one square in the section to any other by crossing shared lines between grid squares.

The values of n, m and t will satisfy $4 \leq n \leq 9$, $2 \leq m \leq 10$, $0 < t$ and $1 \leq r, c \leq n$.

Output

Output the number of valid ways in which the section could be filled in for a KenKen puzzle of the given size.

Sample Input 1	Sample Output 1
8 2 7 - 1 1 1 2	2

Sample Input 2	Sample Output 2
9 2 7 - 1 1 1 2	4

Sample Input 3	Sample Output 3
8 3 6 + 5 2 6 2 5 1	7

Problem G

Class Time

It's the first day of class! Tom is teaching class and first has to take attendance to see who is in class. He needs to call the students' names in alphabetical order by last name. If two students have the same last name, then he calls the students with that same last name in alphabetical order by first name. Help him!

Input

The first line of input contains an integer n ($1 \le n \le 100$), the number of students in Tom's class. Each of the following n lines contains the name of a single student: first name, followed by a single space, then last name. The first and last name both start with an uppercase letter ('A'–'Z') and then be followed by one or more lowercase letters ('a'–'z'). The first and last name of each student is no more than 10 letters long each.

It is guaranteed that no two students have exactly the same name, though students may share the same first name, or the same last name.

Output

Output n lines, the names of the students as Tom calls them in the desired order.

Sample Input	Sample Output
3 John Adams Bob Adam Bob Adams	Bob Adam Bob Adams John Adams

Sample Input	Sample Output
1 Coursera Educators	Coursera Educators

Problem H
Trick Shot

Your game development studio, Ad Hoc Entertainment, is currently working on a billiards-based app they're calling Pool Shark. Players face a sequence of increasingly devious pool puzzles in which they need to carefully position and aim a single billiards shot to sink multiple pool balls.

You've just done the first round of user testing and the feedback is terrible — players complain that the physics of your pool game is neither fun nor intuitive. After digging into it, you realize that the problem isn't that your physics code is bad, but rather that most people just don't have much intuition about how physics works. Fortunately, no one requires your physics to be realistic. After this liberating realization, your team experiments with a few models, eventually settling on the following rule for how to resolve pool-ball collisions:

When a moving pool ball B hits a stationary ball A, A begins moving in the direction given by the vector from the center of B to the center of A at the time of the collision. Ball B's new velocity vector is B's original vector reflected across A's new vector (Figure H.1). Note that A's resulting vector is what real physics predicts, but B's is not (unless A is glued to the table or has infinite mass). For the purposes of this problem, the speed at which the balls move is irrelevant.

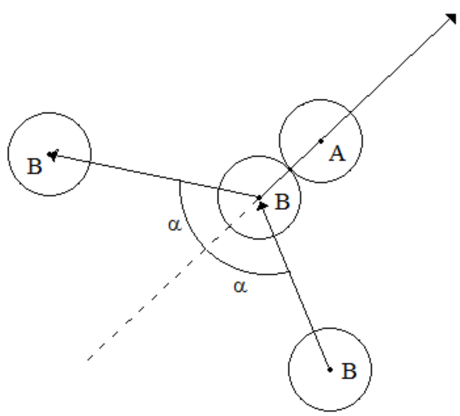


Figure H.1

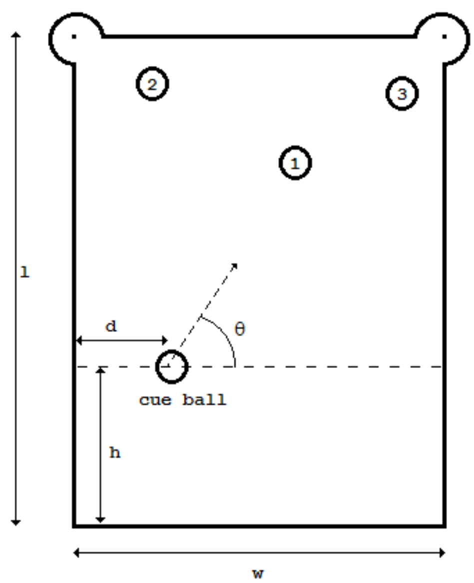


Figure H.2

This actually allows for more interesting challenges, but requires new code to determine whether a particular level is feasible. You've been tasked with solving a very particular case:

Three balls labelled 1, 2, and 3 are placed on a table with width w and length l (Figure H.2). The player must place the cue ball somewhere on a dashed line lying h units above the bottom edge of the table. The goal is to pick a distance d from the left side, and an angle θ such that when the cue ball is shot, the following events happen:

- The cue ball strikes ball 1, and then ricochets into ball 2, sinking ball 2 in the top left hole.
- Ball 1, having been struck by the cue ball, hits ball 3, sinking ball 3 in the top right hole.

For simplicity, assume that sinking a ball requires the center of the ball to pass directly over the center of the hole. Further assume that the table has no sides — a ball that goes out of the w-by-l region simply falls into a digital abyss — and thus you don't need to worry about balls colliding with the table itself.

You need to write a program that, given values for w, l, h, the position of balls 1–3, and the radius r of the balls, determines whether the trick shot is possible.

Input

The input begins with a line containing two positive integers w l, the width and length of the pool table, where $w, l \leq 120$. The left hole is at location $(0, l)$ and the right hole is at location (w, l).

The next line will contain 8 positive integers r x_1 y_1 x_2 y_2 x_3 y_3 h, where $r \leq 5$ is the radius of all the balls (including the cue ball), x_i y_i is the location of ball i, $1 \leq i \leq 3$, and h is the distance the dashed line is from the front of the pool table (see the figure above, where $r \leq h \leq (1/2)l$). No two balls will ever overlap, though they may touch at a point, and all balls will lie between the dashed line and the back of the table. All balls will lie completely on the table, and the cue ball must also lie completely on the table (otherwise the shot is impossible).

Output

For each test case, display the distance d to place the ball on the dashed line and the angle θ to shoot the ball, or the word "`impossible`" if the trick shot cannot be done. Output θ in degrees, and round both d and θ to the nearest hundredth. Always show two digits after the decimal point, even if the digits are zero.

Sample Input 1	Sample Output 1
`20 30` `2 10 20 2 24 18 28 10`	`12.74 127.83`

Sample Input 2	Sample Output 2
`20 30` `2 15 20 2 24 18 28 10`	`impossible`

Problem I
What's on the Grille?

The *grille cipher* is a technique that dates back to 1550 when it was first described by Girolamo Cardano. The version we'll be dealing with comes from the late 1800's and works as follows. The message to be encoded is written on an $n \times n$ grid row-wise, top to bottom, and is overlaid with a card with a set of holes punched out of it (this is the grille).

The message is encrypted by writing down the letters that appear in the holes, row by row, then rotating the grille 90 degrees clockwise, writing the new letters that appear, and repeating this process two more times. Of course the holes in the grille must be chosen so that every letter in the message will eventually appear in a hole (this is actually not that hard to arrange).

An example is shown below, where the message "Send more monkeys" is encrypted as "noeesrks-dmnyemoj", after adding a random letter to fill out the grid (this example corresponds to the first sample input.)

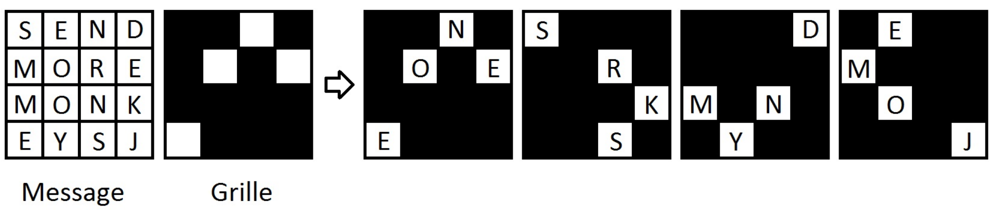


Figure I.1

If the message is larger than the $n \times n$ grid, then the first n^2 letters are written in the grid and encrypted, then the next n^2 are encrypted, and so on, always filling the last grid with random letters if needed. Here, we will only be dealing with messages of length n^2.

Your job, should you choose to accept it, is to take an encrypted message and the corresponding grille and decrypt it. And we'll add one additional twist: the grille given might be invalid, i.e., the holes used do not allow every location in the grid to be used during the encryption process. If this is the case, then you must indicate that you can't decrypt the message.

Input

The input starts with a line containing a positive integer $n \leq 10$ indicating the size of the grid and grille. The next n lines will specify the grille, using '.' for a hole and 'X' for a non-hole. Following this will be a line containing the encrypted message, consisting solely of lowercase alphabetic characters. The number of characters in this line will always be n^2.

Output

Output the decrypted text as a single string with no spaces, or the phrase "`invalid grille`" if the grille is invalid.

Sample Input 1	Sample Output 1
4 XX.X X.X. XXXX .XXX noeesrksdmnyemoj	sendmoremonkeysj

Sample Input 2	Sample Output 2
4 .XX. XXXX XXXX .XX. abcdefghijklmnop	invalid grille

Sample Input 3	Sample Output 3
2 X. XX aybb	baby

Problem J
Surf

Now that you've come to Florida and taken up surfing, you love it! Of course, you've realized that if you take a particular wave, even if it's very fun, you may miss another wave that's just about to come that's even more fun. Luckily, you've gotten excellent data for each wave that is going to come: you'll know exactly when it will come, how many *fun points* you'll earn if you take it, and how much time you'll have to wait before taking another wave. (The wait is due to the fact that the wave itself takes some time to ride and then you have to paddle back out to where the waves are crashing.) Obviously, given a list of waves, your goal will be to maximize the amount of fun you could have.

Consider, for example, the following list of waves:

Minute	Fun points	Wait time
2	80	9
8	50	2
10	40	2
13	20	5

In this example, you could take the waves at times 8, 10 and 13 for a total of 110 fun points. If you take the wave at time 2, you can't ride another wave until time 11, at which point only 20 fun points are left for the wave at time 13, leaving you with a total of 100 fun points. Thus, for this input, the correct answer (maximal number of fun points) is 110.

Given a complete listing of waves for the day, determine the maximum number of fun points you could earn.

Input

The first line of input contains a single integer n ($1 \leq n \leq 300{,}000$), representing the total number of waves for the day. The ith line ($1 \leq i \leq n$) that follows will contain three space separated integers: m_i, f_i, and w_i, ($1 \leq m_i, f_i, w_i \leq 10^6$), representing the time, fun points, and wait time of the ith wave, respectively. You can ride another wave occurring at exactly time $m_i + w_i$ after taking the ith wave. It is guaranteed that no two waves occur at the same time. The waves may not be listed in chronological order.

Output

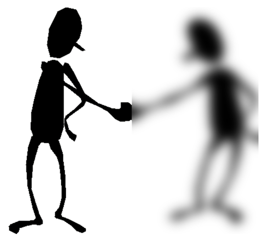
Print, on a single line, a single integer indicating the maximum amount of fun points you can get riding waves.

Sample Input	Sample Output
4 8 50 2 10 40 2 2 80 9 13 20 5	110

Sample Input	Sample Output
10 2079 809484 180 8347 336421 2509 3732 560423 483 2619 958859 712 7659 699612 3960 7856 831372 3673 5333 170775 1393 2133 989250 2036 2731 875483 10 7850 669453 842	3330913

Problem K

Blur



You have a black and white image that is w pixels wide and h pixels high. You decide to represent this image with one number per pixel: black is 0, and white is 1. Your friend asks you to blur the image, resulting in various shades of gray. The way you decide to blur the image is as follows: You create a new image that is the same size as the old one, and each pixel in the new image has a value equal to the average of the 9 pixels in the 3×3 square centered at the corresponding old pixel. When doing this average, wrap around the edges, so the left neighbor of a leftmost pixel is in the rightmost column of the same row, and the top neighbor of an uppermost pixel is on the bottom in the same column. This way, the 3×3 square always gives you exactly 9 pixels to average together. If you want to make the image blurrier, you can take the blurred image and blur it again using the exact same process.

Given an input image and a fixed number of times to blur it, how many distinct shades of gray does the final image have, if all the arithmetic is performed exactly?

Warning: Floating point numbers can be finicky; you might be surprised to learn, for example, that $2/9 + 5/9$ *may not equal* $3/9 + 4/9$ *if you represent the fractions with floating point numbers! Can you figure out how to solve this problem without using floating point arithmetic?*

Input

The first line of input contains three space-separated integers w, h, and b ($3 \leq w, h \leq 100, 0 \leq b \leq 9$), denoting the width and height of the image, and the number of times to blur the image, respectively. The following h lines of w space-separated integers describe the original image, with each integer being either 0 or 1, corresponding to the color of the pixel.

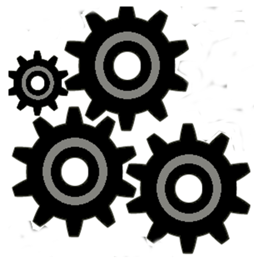
Output

Output, on a single line, a single integer equal to the number of distinct shades of gray in the final image.

Sample Input	**Sample Output**
`5 4 1` `0 0 1 1 0` `0 0 1 1 0` `0 0 1 1 0` `0 0 1 1 0`	`3`

Sample Input	**Sample Output**
`3 3 2` `1 0 0` `0 1 0` `0 1 0`	`1`

Problem L
Gears



A set of gears is installed on the plane. You are given the center coordinate and radius of each gear. For a given input and output gear, indicate what happens to the output gear if you attempt to rotate the input gear.

Input

The first line of input contains a single positive integer n ($2 \leq n \leq 1{,}000$), the total number of gears. Following this will be n lines, one per gear, containing three space-separated integers x_i, y_i, and r_i ($-10^4 \leq x_i, y_i \leq 10^4$, $1 \leq r_i \leq 10^4$), indicating the center coordinate and the radius of the ith gear. Assume the tooth count for each gear is sufficiently high that the gears always mesh correctly. It is guaranteed that the gears do not overlap with each other. The input gear is the first gear in the list, and the output gear is the last gear in the list.

Output

If the input gear cannot move, print, on a single line, "`The input gear cannot move.`" (without the quotation marks).

If the input gear can move but is not connected to the output gear, print, on a single line, "`The input gear is not connected to the output gear.`" (without the quotation marks).

Otherwise, print, on a single line, the ratio the output gear rotates with respect to the input gear in the form of "`##:##`" (without the quotation marks), in reduced form. If the output gear rotates in the opposite direction as the input gear, write the ratio as a negative ratio. For example, if the output gear rotates clockwise three times as the input gear rotates counterclockwise twice, the output should be `-3:2`.

Sample Input	Sample Output
`2` `0 0 100` `200 0 100`	`-1:1`

Sample Input	Sample Output
`` 3 0 0 100 200 0 100 400 0 100 ```	``` 1:1 ```

Sample Input

```
3
0 0 100
200 0 100
400 0 100
```

Sample Output

```
1:1
```

Sample Input

```
16
10 10 5
20 10 5
30 10 5
40 10 5
10 20 5
20 20 5
30 20 5
40 20 5
10 30 5
20 30 5
30 30 5
40 30 5
10 40 5
20 40 5
30 40 5
40 40 5
```

Sample Output

```
1:1
```

Sample Input

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

Sample Output

```
The input gear cannot move.
```