# Problem A. Green tea

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 0.25 seconds |
| Memory limit: | 64 megabytes |

Programmer Basil likes to drink green tea. For brewing the green tea properly, one can't use hot water — it is recommended to use water with a temperature of exactly 80 degrees. In the room, where Basil works there is a boiler that maintains a constant water temperature of $t_1$ degrees, and a jug of cold water with a temperature of $t_2$ degrees. Basil has a mug, in which he makes tea, and a measuring spoon.

Help Basil — write a program that will determine by the temperature $t_1$ and $t_2$ how many spoons of water ($v_1$ and $v_2$) of each kind should be poured into a mug to get water with a temperature of exactly 80 degrees. If there are several solutions, type the one that has the minimal sum $v_1 + v_2$.

For simplicity, we assume that during the transfusion, the water temperature remains unchanged, and when the mixing happens, the law comes into force:

$$t_1 \cdot v_1 + t_2 \cdot v_2 = t_3 \cdot (v_1 + v_2),$$

where $t_1$ and $t_2$ are the temperatures of the portions of water being mixed, $v_1$ and $v_2$ are the volumes of these portions, and $t_3$ is the temperature obtained as a result of mixing.

## Input

The program receives 2 integers at the input, separated by a space: temperatures $t_1$ ($80 \le t_1 \le 100$) and $t_2$ ($0 \le t_2 < 80$).

## Output

The program is supposed to output two integers — volumes $v_1$ and $v_2$ ($0 \le v_1, v_2 \le 1000$), where $v_1$ is the number of spoons of water with the temperature $t_1$, $v_2$ is the number of spoons of water with the temperature $t_2$.

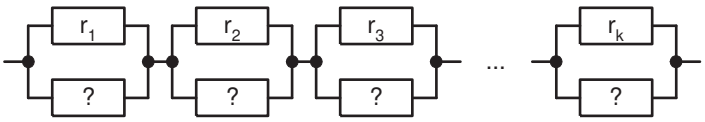
## Examples

| standard input | standard output |
|---|---|
| 100 20 | 3 1 |
| 100 30 | 5 2 |

# Problem B. Mysterious Resistors

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 0.4 seconds |
| Memory limit: | 64 megabytes |

When repairing an electronic board, Basil discovered that the board had already been under repair, and one of the resistors on it was replaced with a strange design. The design consisted of $k$ series of connected links, and each link in them — of two resistors connected in parallel.




Moreover, in each link, along with the typical resistor, with a clearly readable denomination, there was a resistor with a strange, non-standard marking. Having examined the non-standard resistors, Basil came to the conclusion that they were of the same type. However, he could not determine their value. The total resistance value of the entire circuit was equal to $R$ ohms. Having written out the nominal value of the known resistor for each link in the circuit, Basil received a series of integers $r_1$, $r_2$, ..., $r_k$.

Help Basil — write a program that calculates the value of the mysterious resistors from the total resistance of the circuit $R$, and the known values $r_1$, $r_2$, ..., $r_k$.

## Input

The first line of the input data contains two integers $k$ ($1 \le k \le 1000$) and $R$ ($1 \le R \le 100000$), separated by a space.

The second line contains $k$ integers separated by spaces: $r_1$, $r_2$,..., $r_k$ ($1 \le r_i \le 100000$; $2R \le r_1 + r_2 + \ldots + r_k$).

## Output

The program should output a single number — the estimated value of the mysterious resistors. The value must be outputed with an accuracy of $10^{-6}$.

## Examples

| standard input | standard output |
|---|---|
| 3 11<br>3 12 30 | 6.00000000 |
| 7 110<br>15 60 6 45 20 120 70 | 30.00000000 |

## Note

Recall that while connecting two resistors with the values of $R_1$ and $R_2$, the total resistance $R$ is calculated as $R = R_1 + R_2$ for the serial connection, and as $\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$ for the parallel one.

# Problem C. Emoticons

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 0.4 seconds |
| Memory limit: | 64 megabytes |

A programmer, Basil by name, is writing a new text editor. In order to display mail correspondence, Basil has written a special module that extracts all emoticons from a text file and puts them out onto a separate line for display. Unfortunately, an error has crept into the module, and the characters in the line of emoticons got mixed up.

Basil knows that the following emoticons «`;-)`», «`;-(`», «`:)`», «`:(`», «`:-\`», «`:-P`», «`:D`», «`:C`», «`:-0`», «`:-|`», «`8-0`», «`:-E`», «`%0`», «`:-X`», «`:~(`» (character "~"has code 126), «`[:|||:]`» have been used in the letter that is being processed.

Help Basil — write a program that will restore the emoticons by a given line of characters. If several recovery options are possible, then any of them will be correct.

## Input

A string of characters that one may come across when entering the emoticons of the kind described above is presented as input data. The line length does not exceed 10000 characters. The line is not empty. It is obtained by mixing the correct sequence of emoticons.

## Output

The program should write the restored emoticons: one per line and finalize the notation with "LOL" (without quotes).

## Example

| standard input | standard output |
|---|---|
| `\|:\|;[\|)-:]` | `[:\|\|\|:]`<br>`;-)`<br>`LOL` |

# Problem D. Power play

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 64 megabytes |

While analyzing a mathematical problem, a programmer, Basil by name, noticed an interesting fact: for the numbers 2 and 4 holds the equality $2^4 = 4^2$. 'This could be a great challenge for the participants of the programming championship,' he thought. Unfortunately, Basil could not find other such pairs of numbers, the pair 2 and 4 was his only combination that he could think of. 'Okay, then we'll change the conditions, let there be three numbers,' Basil decided. The written program of enumeration options confirmed that with three numbers the task made sense.

Your task: find the integer $x$ by the given integers $a$ and $b$ such that falls in the range from 1 to $10^{18}$ (Basil's program didn't deal with greater numbers) such that $a^x = x^b$. If there are several such numbers, type the smaller of them. If such a number does not exist, type 0.

## Input

The input data consist of two integers $a$ and $b$ ($2 \le a, b \le 10000$; $a \ne b$), separated by a blank space.

## Output

The integer $x$ if there is a solution, or 0 (zero) if there is no solution.

## Examples

| standard input | standard output |
|---|---|
| 2 4 | 16 |
| 2 6 | 0 |
| 2 32 | 256 |
| 100 20 | 10 |

# Problem E. Printed circuit board

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 64 megabytes |

Printed circuit boards are one of the key notions of modern electronics. There are a huge number of programs for working with the printed circuit boards, and for transferring the complete drawing to the surface of the board, there exist quite a lot of methods (sometimes very exotic). One way to transfer a picture is to use a plotter, a device that can draw continuous lines on the surface. Write a program that, according to the printed circuit board drawing, will make up a set of commands for the plotter to draw this drawing.

The plotter works on a square grid of $n \times m$ size. The grid squares are numbered from left to right (first coordinate), and from top to bottom (second coordinate). The command for drawing a line consists of two numbers — the coordinates of the beginning of the line, and a sequence of letters "L", "R", "U", "D", which draw a line one cell from the current position to the left, right, up or down, respectively. For example, the "3 4 DRUL" command will draw a square with side 1 and the upper left corner at coordinates 3 4.

In order to minimize the time spent on drawing, the program should build a set of the minimum possible number of commands (continuous lines). If there are several such sets, then any of them is to be considered a solution.

Please note that:

- Any cell of the board either does not contain a line, or contains one line, or two intersecting lines (intersection point);
- A line can begin and end at the same point;
- If two lines start at one point, then they can be combined into one line;
- The line length cannot be less than 2 (the line cannot connect 2 adjacent squares);
- A line or lines cannot begin or end in neighboring cells (the cells are considered neighboring if they have a common side);
- A line cannot be drawn over an existing one.

## Input

The first line of input contains integers $n$ and $m$ ($1 \le n, m \le 100$) — the size of the printed circuit board. This is followed by $n$ lines of $m$ characters in each. The following characters are used to encode a board picture:

- "." ("Point", code 0x2E) — empty (blank) square;
- "*" ("Asterisk", code 0x2A) — The beginning or end of the line, or the point where more than two lines meet;
- "-" ("Minus", code 0x2D) - Horizontal line;
- "|" ("Vertical bar", code 0x7C) — Vertical line;
- "L" ("Capital L", code 0x4C) — The line connecting the upper and the right cells;
- "J" ("Capital J", code 0x4A) — A line connecting the upper and the left cells;
- "r" ("Lowercase r", code 0x72) — The line connecting the lower and the right cells;

- "7" ("Digit 7", code 0x37) — The line connecting the lower and the left cells.

For any line in the image, there is a starting and a final point. The line can begin and end at the same point, then such a point is indicated as "*". Through one square either one line passes, or, if several lines intersect in this square, then they are considered connected, and the square contains the "*" sign. The passing of lines through one square without a connection is not possible. The signs "*" cannot be found in the neighboring cells (cells having a common side).

## Output

The first line of the output contains the number $k$ ($0 \le k \le 10000$) — the number of commands. This is followed by $k$ lines, each of which describes a separate command. The command begins with two integers $x$ and $y$, separated by a blank space, — the coordinates of the beginning of the line. Further, also through a blank space, comes a sequence of capital letters "L", "R", "U" or "D", describing the line to be drawn. Letters go in a row without spaces. The command ends with a line feed.

## Examples

| standard input | standard output |
|---|---|
| 3 4<br>r--*<br>*...<br>L--* | 1<br>4 3 LLLUURRR |
| 3 4<br>r--*<br>L--7<br>*--J | 1<br>1 3 RRRULLLURRR |
| 3 3<br>r-*<br>\|.\|<br>L-J | 1<br>3 1 LLDDRRUU |
| 5 5<br>..*..<br>..\|..<br>*-*-*<br>..\|..<br>..*.. | 2<br>3 5 UUUU<br>1 3 RRRR |

## Note

Please, note that the board drawing is described in a line by line fashion and the commands use the "column row" coordinates.

# Problem F. A word game

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

Alice and Bob are playing the following game:

- a word is written on paper in front of the players
- in one move, a player can erase one letter, or two identical letters, or, all letters of the same type
- players take turns
- Alice is the first to start
- it is the player, who erases the last letter, that wins.

Write a program that will determine the winner of the described above game for a given word (given that the players play optimally (successfully).

## Input

The input contains a single line — the starting word. The word consists of capital Latin letters, and its length does not exceed 40 characters.

## Output

Type "Alice" (without quotes) if Alice wins, or "Bob" (without quotes) if Bob wins.

## Examples

| standard input | standard output |
|---|---|
| ZADACHA | Alice |
| WORD | Bob |

## Note

For example, in the word "ZADACHA", Alice erases all the letters "A" by her first move. The word "Z.D.CH." remains, and she wins because, in subsequent moves, the players are forced to erase only one letter at a time.

# Problem G. Hourglass

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 0.5 seconds |
| Memory limit: | 24 megabytes |

In the middle ages, artisans used a hourglass for measuring time. As such devices were quite rare and expensive, not every master had a set of them that made it possible to measure any time interval. Artisans were very inventive, and managed to produce very complex measurements with the help of minimum options (choice of devices). For example, with the help of 3-minute and 5-minute devices any number of minutes (greater than 4) could be calculated. And can you surpass our ancestors?

You have $n$ (up to 4) hourglasses numbered 1, 2, ..., $n$, the sand in which is poured during $t_1$, $t_2$, ..., $t_n$ (up to 20) minutes, respectively. Compile a program that will find a way to measure out $k$ minutes with the help of this device. Note that:

- any hourglass can be turned over only at the very beginning of measurements, or at the moment when one of the device has run out of sand;
- at such a moment, you can turn over any number of devices (hourglasses) at the same time. The hourglass is turned over instantly;
- it is forbidden to put the clock onto one side (to stop the time);
- the measured interval of $k$ minutes starts from the very first turning of any device;
- the measured interval of $k$ minutes should end at the moment when any device has run out of sand.

For example, let us have a device for 3 and 5-minute duration. Let's show how they can be used to measure 7 minutes:

- 0 minute. At the same time, we turn both the devices for 3 and 5-minute duration.
- 3 minutes. The 3 minute hourglass is empty and the 5 minute device has sand for 2 minutes more. Turn the device over for 3 minutes.
- 5 minutes. The 5 minute device is empty, in the 3 minute device there is sand still for 1 minute from above, and for 2 minutes from below. Turn the hourglass over for 3 minutes.
- 7 minutes. The hourglass for 3 minutes is empty. The specified interval is measured.

## Input

The first line of input contains the integer $n$ ($1 \le n \le 4$) — the number of hourglasses. The second line contains integers $t_1$, $t_2$, ..., $t_n$ ($1 \le t_1 < t_2 < \ldots < t_n \le 20$) separated by spaces — hourglass denominations. The third line contains the integer $k$ ($1 \le k \le 1440$) — the time interval to be measured.

## Output

The first line of the output contains $-1$ if the time interval $k$ cannot be measured.

Otherwise, it contains an integer $r$ — the number of time moments ($1 \le r \le k$) at which the hourglass is turned over. This is followed by $r$ lines describing the moments of turning the clock over.

Each such line begins with the number $t_i$ — the number of minutes from the start of the countdown, up to the moment of this inversion. Next, a space is followed by the number $m_i$ ($0 \le m_i \le n$) — the number of hourglasses flipped at the time of $t_i$. Further, after a space, the numbers of the turned over hourglasses are specified.

The lines should be displayed in the increasing order of time: $t_i > t_{i-1}$. At the moment of time specified as $t_i$, at least, one hourglass should run out of sand.

The first line should contain a description of the initial time ($t_1 = 0$). The last line must contain the time point $k$ and 0 of the hourglasses that are being flipped. Lines with time specification $t_i > k$ can't be displayed. Lines with time specification $m_i = 0$ are allowed and are not considered a mistake.

## Examples

| standard input | standard output |
|---|---|
| 2<br>3 5<br>7 | 4<br>0 2 1 2<br>3 1 1<br>5 1 1<br>7 0 |
| 2<br>3 5<br>4 | -1 |
| 2<br>7 9<br>13 | 5<br>0 2 1 2<br>7 1 1<br>9 2 1 2<br>11 1 2<br>13 0 |

# Problem H. Men's showdown

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

Once, a big debate broke out in Room 113: which of the two students will go to wipe out the board? The problem began to be solved in a manly way! There is a box with $N$ crayons in this room. Each student in turn takes 1, 5, or 13 crayons. Everything is extremely simple: the student after whose move there are no crayons left will go to wipe the board covered with formulas. Both students make their moves optimally.

Who will win this fight and not wipe out the ill-fated board?

## Input

The first line introduces the number $N$ — the number of crayons in Room 113 ($1 \leq N \leq 10000$).

## Output

Type the number (№) of the player who wins.

## Examples

| standard input | standard output |
|---|---|
| 4 | 1 |
| 5 | 2 |

# Problem I. Andrew and Python

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 0.5 seconds |
| Memory limit: | 64 megabytes |

This is an interactive problem.

The first thing that a programming olympiad teacher should do is treat "the recruit" for a serious, but absolutely curable disease — pythonism.

Marat V. is a 'specialist' in the sphere of treatment of this disease, but, unfortunately, there is no so called "permanganate acid" to "treat" it. Then the teachers had no choice but to hide Andrew's knowledge of the ill-fated programming language in a very reliable place: the stash in the floor of the Room 113.

The parquet in Room 113 is a square with side $n - 1$ on the coordinate plane and contains $n^2$ points with natural coordinates. "The stash" is a point on the plane of the inner side of the square or on the side of the square. Andrew studies programming for the first year, so he does not know where it is. The only thing Andrew can do is ask stupid questions to his neighbor Zuckerman. There are three types of questions that he asks:

1) Andrew may ask about a specific point whether it is a stash;

2) Andrew may ask his neighbor about the segment given by the coordinates of its ends. However, the ends do not coincide — does the hidden point lie on this segment;

3) Andrew may ask his mate about the triangle formed by the coordinates of its vertices. The triangle is nondegenerate — does the hidden point lie inside or on the border of this triangle.

In any query, the coordinates of the points must be natural numbers.

Zuckerman, can answer the question either "Yes", or "No".

Andrew does not want his teachers to know that he is looking for his knowledge of the Python language, so he must ask no more than 60 questions so that no one suspects anything.

Help Andrew guess the hidden point in the floor for no more than 60 questions.

## Interaction Protocol

First, your program receives one integer $n$ — the number of points on the side of the square where you want to search for a point ($1 \le n \le 10^8$).

After that, your program can make requests:

- In order to ask whether the point with the natural coordinates $(x, y)$ is the desired one (query type 1), you need to output it to the output stream in a separate line "? 1 $x$ $y$" .
- In order to ask whether the desired point lies on the segment with the ends at the natural points $(x_1, y_1)$ and $(x_2, y_2)$ (query type 2), you need to output "? 2 $x_1$ $y_1$ $x_2$ $y_2$" into the output stream in a separate line.
- In order to ask whether the desired point lies inside the triangle with vertices at the natural points $(x_1, y_1)$, $(x_2, y_2)$, $(x_3, y_3)$ (request type 3), you need to output "? 3 $x_1$ $y_1$ $x_2$ $y_2$ $x_3$ $y_3$" into the output stream in a separate line.

In response to the request, the string "Yes" will be written in the input stream if the answer is positive, or "No", otherwise.

In any query, the coordinates of the points must be natural numbers and must not exceed $10^9$.

If your program makes more than 60 requests or makes an incorrect request (for example, requests a degenerate triangle), it will receive any verdict other than "Accepted" .

Having determined at what point in the plane there is a stash with the knowledge of the Python language, your program should output "! $x$ $y$", where $(x, y)$ are the coordinates of the desired point.

It is guaranteed that the hidden point has natural coordinates, it is also the fixed kind of point and will not change during the testing of your program.

## Example

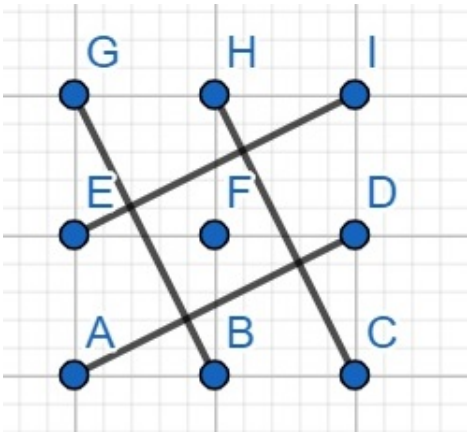
| standard input | standard output |
|---|---|
| 3 | ? 2 1 1 3 2 |
| No | ? 2 3 1 2 3 |
| No | ? 2 3 3 1 2 |
| No | ? 2 1 3 2 1 |
| No | ? 3 1 1 2 3 3 1 |
| Yes | ? 1 2 2 |
| Yes | ! 2 2 |

## Note

Output a line feed character after every action of your program. After printing each query you need to use special function to flush the output stream, because it is possible that some part of your output is left in the buffer. For example, you can use "`fflush(stdout)`" in C++, "`System.out.flush()`" in Java, "`flush(output)`" in Pascal and "`stdout.flush()`" in Python.

**EXPLANATION TO THE EXAMPLE**

In the test, a point with coordinates $(2, 2)$ (F in the figure) is given on a square with side 3. For convenience, the figure is presented.




The first query is whether the point belongs to the segment $(1, 1)$ $(3, 2)$ (AD in the figure). The answer is no.

The second query is whether the point belongs to the segment $(3, 1)$ $(2, 3)$ (CH in the figure). The answer is no.

The third query is whether the point belongs to the segment $(3, 3)$ $(1, 2)$ (IE in the figure). The answer is no.

The fourth query is whether the point belongs to the segment $(1, 3)$ $(2, 1)$ (GB in the picture). The answer is no.

The fifth query is whether the point belongs to the triangle $(1, 1)$ $(2, 3)$ $(3, 1)$ (AHC in the figure). The answer is yes.

The sixth query is whether point $(2, 2)$ is the required one. The answer is yes.

The displayed answer is ! 2 2.

# Problem J. Something that resembles Waring's problem

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 4 seconds |
| Memory limit: | 256 megabytes |

Waring's problem is a number-theoretic statement, according to which for every integer $n > 1$ there exists a number $k = k(n)$ such that any natural number $N$ can be represented as:

$$x_1^n + x_2^n + ... + x_k^n = N$$

with non-negative integers $x_1$, $x_2$, ..., $x_n$.

You are required to represent the number $x$ in the form of $a_1^3 + a_2^3 + ... + a_k^3 = x$, with integers $a_1$, ..., $a_k$ and $k \leq 5$.

## Input

The first line contains the number $x$ ($1 \leq x \leq 10^{100000}$).

## Output

If such a representation does not exist, type $-1$. Otherwise, in the first line, type the number $k$ — the number of numbers. In the second line, type $k$ integers $a_i$ ($|a_i| \leq 10^{110000}$).

## Examples

| standard input | standard output |
|---|---|
| 5 | 5<br>1 1 1 1 1 |
| 17 | 3<br>1 2 2 |

## Note

In the first example: $1^3 + 1^3 + 1^3 + 1^3 + 1^3 = 5$.

In the second example: $1^3 + 2^3 + 2^3 = 17$.

# Problem K. Parabolic sorting

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

Professor R. is the greatest specialist in the field of discrete mathematics. He has already got bored with all the standard sorting algorithms, and has come up with a new algorithm — the parabolic array sorting algorithm.

Let there be an array of $N$ different integer numbers $a_i$. We want to sort it in such a way that the first part of the array strictly decreases, and the second part of the array strictly increases. In this case, we can only interchange the neighboring elements. Both the first and second parts of the array can be of zero length. The professor is interested in the following question: what is the minimum number of actions we need?

## Input

In the input $N+1$ there is a number, the first of which is $N$ ($1 \le N \le 10^5$). Then follow the numbers $a_i$, whose module does not exceed $10^9$.

## Output

Print a single number — the answer to the problem.

## Examples

| standard input | standard output |
|---|---|
| 4 2 3 1 4 | 1 |
| 5 3 2 1 4 5 | 0 |