Even Odds

You have been hired by the International Chair Production Company (ICPC) to solve a problem with the operation of the ski-lift chair equipment manufactured by the company. The company manufactures two types of ski-lift chairs: left-handed chairs and right-handed chairs. In order to keep track of the two types, a unique integer ID is assigned to each chair: left-handed chairs always have even ID numbers while right-handed chairs always have odd ID numbers. ID numbers are always greater than zero.

Chairs are stored in a warehouse while waiting to be brought out each morning and attached to the ski-lift cable. In order to keep the cable in balance, it is critical that left- and right-handed chairs be attached in strictly alternating order. It does not matter whether the first attached chair is left- or right-handed (that is, has an even number or an odd number): what is important is that as the subsequent chairs are attached they always alternate in type. In other words, it is a requirement that no two odd-numbered chairs be next to each other on the cable, and likewise that no two even-numbered chairs be next to each other on the cable. There will always be at least one chair attached to the cable.

You are to write a program that examines a list of the chair numbers attached to the cable and verifies that the chairs alternate in type correctly. The input to your program, to be read from "standard input", will be a sequence of integer chair numbers. Each input number will be separated from the others by an arbitrary amount of whitespace, including that the input numbers may extend over multiple input lines.

Your program is to determine whether all the integers in the input sequence follow the strictly-alternating rule. If the integers do correctly alternate between even and odd, your program should output the message "Yes"; if not, your program should output the message "No".

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 11 4 5</td>
<td>Yes</td>
</tr>
<tr>
<td>6 7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 22 34</td>
<td>No</td>
</tr>
</tbody>
</table>
CompuDump - the Exciting New Programming Language!

The newest and most exciting new technology to hit the computing world has arrived: CompuDump! CompuDump is an elegant new programming language that incorporates cutting-edge features such as integer addition, subtraction, if statements, and loops. Here is a sample program:

38
P 5
P 8
S 9
I -12
R 2
S 3
E 0

In CompuDump, the first line of code is always an integer between the values -1000 and +1000. There then follows a series of CompuDump commands that successively make modifications to that integer, which from now on will be referred to as the RUNVAL.

The commands are formatted one per line, and consist of a character command, then a space, then an integer. The allowable character commands are:

"P" which means "PLUS", which adds the specified integer to the RUNVAL;

"S" which means "SUBTRACT", which subtracts the specified integer from the RUNVAL;

"I" which means "IF", which adds the specified integer to the RUNVAL if the RUNVAL is greater than zero, and does not change the RUNVAL otherwise;

"R" which means "REPEAT". "REPEAT" causes the next line to be executed however many times are specified in the R command. You can assume that the next line is either a "P" or an "S" command, and that the integer specified on the R command is at least 1.

"E 0" which means END the program and display the final RUNVAL. Note that the "0" is the numeric digit "zero".

In the above example program, the RUNVAL starts at 38, then 5 is added making 43, then 8 is added making 51, then 9 is subtracted making 42, then -12 is added making 30 (because 42 is greater than zero), then 3 is subtracted twice making 24 (because R 2 causes the "S 3" to be done twice), then the program ends and the value 24 is displayed.

Your job is to write a CompuDump interpreter, that accepts as input a legal CompuDump program and outputs the final resulting RUNVAL.

Input
Your program reads in a legal CompuDump program, formatted as described above. You can assume that the program is legal - that is, you don't have to check for syntax errors.

Note that the smallest legal program is two lines long: one line for the initial RUNVAL, and one line for the "E 0". You can assume that the input program is at most 999 lines long. Note that the various P, S, I, and R commands can appear in any order, with the only exception being that whenever an R command appears, it must be followed by a P or S command.

Output
Your program outputs a single integer, the final RUNVAL resulting from running the program.
Masher

A frustrated student in the Association of Calm Members (ACM) had mashed his keyboard for a while in the text document that he had open. An observing statistics professor had an epiphany and decided to want to know some statistics of the angered fiasco.

Input

A single line of input containing the mashed text. The text will only include the capital letters A to Z. There will always be at least five distinct characters.

Output

Your program should output a list of the top five most likely characters and their probability of being pressed next (rounded to the nearest 0.001) in descending order and formatted as shown below. If two characters have the same probability after rounding, output them in ascending alphabetical order. The probability of being pressed next is the number of previous occurrences divided by the total number of keys pressed.

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>EATOJASDAPQWENVAPSDFJ</td>
<td>A 0.190, D 0.143, J 0.095, O 0.095, P 0.095</td>
</tr>
</tbody>
</table>

Problem compliments of Kenvy Ka ‘16.
Flip Out

An art professor at the Academy of Cracked Mirrors (ACM), needs a program to flip a block of text as his mirror has cracked. He wants the option to be able to flip the image across the X-axis or Y-axis. He does not want the characters to be changed, just the position.

Input

The first line of input will be an integer that designates the number of lines of text the block will contain. The second line will contain “X” or “Y” or “XY”, designating the direction of the flip. The following lines contain the text. Not all of the text lines will be the same length and no line will end with whitespace.

Output

Your program should output the inputted text flipped across the X axis if an “X” or “XY” was provided and flipped across the Y axis if a “Y” or “XY” was provided.

Note that flipping in either direction does not change which characters are neighbors of one another, it only changes which side the neighbors are on (above becomes below, right becomes left, etc). This is especially important when flipping across the Y axis because shorter lines may need to be indented to maintain their relative position with respect to characters above and below.

Sample Input

5
XY

Sample Output

Sample Input

5
X
ABCDEF
DJANGO
IS NUM
TRUE
FALSE

Sample Output

FALSE
TRUE
IS NUM
DJANGO
ABCDEF
Non-uniform

A cumulative distribution function (cdf) is a non-decreasing function from a set of numbers to the range [0, 1]. (The notation [a, b) is shorthand for the set {x ∈ R, a ≤ x < b}.) When \( f \) is a cdf describing the probability distribution of random variable \( X \), the function \( f(x) \) is defined as \( \Pr(X ≤ x) \).

A cdf \( f \) can be used to generate random values for the probability distribution described by \( f \). Here’s how you do it. First, you choose a uniformly distributed random real \( y \) in [0, 1]. Next, you determine which is the smallest value of \( x \) from \( f \)’s domain that makes \( f(x) ≥ y \). This process yields \( x \) which is randomly distributed according to the probability distribution described by \( f \).

As an example, here is a hypothetical cdf graph from [0,10] to [0,1]. If you wanted a random value in [0,10] according to this distribution, you’d choose a uniform random value in [0,1], draw a horizontal line to find which \( x \) is the first to make \( f(x) ≥ y \), and that \( x \) would be your random value in [0,10].

Write a program that generates random values according to the exponential distribution with parameter \( \lambda \) and random values according to the geometric distribution with parameter \( p \). The exponential cdf is \( f(x) = 1 - e^{-\lambda x} \) which maps non-negative reals to [0,1]. The geometric cdf is \( f(x) = 1 - (1 - p)^{x+1} \) which maps non-negative integers to [0,1]. The values of \( \lambda \) and \( p \) are constants in each cdf and will be supplied as part of the problem input.

**Input**

Read a triple of real numbers. The triple will consist of random \( y \) in [0,1], positive real \( \lambda \), and \( p \) in [0,1]. Each item will be separated by a space.

**Output**

You should output a single line consisting of the exponentially distributed real value indicated by \( y \) and \( \lambda \), and the geometrically distributed integer value indicated by \( y \) and \( p \), separated by a space. Real values within 0.001 of the correct answer will be considered correct.

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5 1.0 0.25</td>
<td>0.6931 2</td>
</tr>
</tbody>
</table>
Gray Codes

We are going to generate a sequence of integers in binary. Start with the sequence

0
1

Reflect it horizontally below, prepend a zero to the numbers in the top half and a one to the numbers on the bottom and you will get

00
01
11
10

Repeat this again, and you will have eight numbers

000 0
001 1
011 3
010 2
110 6
111 7
101 5
100 4

On this last list, the value you get when interpreting each number in binary is shown on the right. These sequences are called Reflected Gray Codes for 1, 2 and 3 bits respectively. A Gray Code for n bits is a sequence of $2^n$ different n-bit integers with the property that every two neighbouring integers differ in exactly one bit. A Reflected Gray Code is a Gray Code constructed in the way shown above.

Program Input

Read a line with 2 integers: $n$ (1 ≤ $n$ ≤ 14) and $k$ (0 ≤ $k$ < $2^n$), separated by a space.

Program Output

Print the position in the $n$-bit Reflected Gray Code sequence that integer $k$ can be found. Position indices begin at zero.

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 7</td>
<td>5</td>
</tr>
</tbody>
</table>
Colored Bead Strings

Imagine that you are working in a factory that produces colored beads on a string. These bead strings are later used to make necklaces, Christmas ornaments, etc.

The factory is using 35 different bead colors to produce their strings. Each color is assigned to an id number between 1 and 35 (eg. red is 1, yellow is 2, ..., blue is 35).

Unfortunately the factory has received a potentially ambiguous order for a string of beads. A sequence of digits has been received, but is missing any kind of separator between individual bead ids. You begin to wonder how many different legal bead string specifications would have this same number if all its digits were concatenated. For example, 35-2-2-4, 35-2-2-4, 3-5-22-4, 3-5-2-24, 35-22-4, and 35-2-24 are all legal bead string specifications and all produce 35224 if you remove the separator between each individual bead id.

Input

A sequence of 1-5000 digits with no spaces between each digit. It is possible that the input does not represent any legal bead string specification (eg. it could start with 0 and no bead id starts with 0).

Output

The number of different legal bead string specifications the person placing the order might have meant. If the input does not represent any legal bead string specification, 0 should be printed as the output.

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>444444444444</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1067891122</td>
<td>5</td>
</tr>
</tbody>
</table>
Food Sack

You are about to go on a long hike and want to bring along as many calories of food as you can carry. So, you go to your cupboard and start packing your food sack. There are various sizes of food with various calorie contents. You're not sure what to pack and decide to write a program to do the choosing for you.

Program Input

The input to your program will be two non-negative integers, \( N \) and \( C \), indicating how many food items you have to choose from and how many cubic centimeters of volume your food sack can hold. There will then be \( N \) more pairs of positive integers, each pair representing an available food item. The first integer of each pair is the volume of the item (in cubic centimeters) and the second integer of each pair is the number of calories in the item. All integers are separated by whitespace and have values less than 10,000.

Program Output

Output the collection of food items that maximizes the number of calories you are able to fit in your food sack. The output format should be one food item per line – each line containing the item’s volume, a space, and the item’s calories – listed in decreasing energy-intensity (calories per cubic-centimeter) order. If there is more than one combination of food items maximizing calories, you may list any of them.

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 1000</td>
<td>600 550</td>
</tr>
<tr>
<td>600 550</td>
<td></td>
</tr>
<tr>
<td>500 200</td>
<td></td>
</tr>
<tr>
<td>500 300</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 1000 600</td>
<td>500 300</td>
</tr>
<tr>
<td>450 500</td>
<td></td>
</tr>
<tr>
<td>500 200</td>
<td></td>
</tr>
<tr>
<td>500 300</td>
<td></td>
</tr>
</tbody>
</table>