2014 Mid-Atlantic Regional Programming Contest

This is a courtesy copy of the problem set for the Mid-Atlantic Regional contest. It is an abbreviated version of the problem set provided to the teams. Omitted are several pages of rules and explanations of the contest and judging environment.

And now, a word from the head judge

Without a team of creative volunteers, there would be no problem set for this contest each year. It's not too early to start thinking about the 2015 contest!

We do require that problem authors not be coaches of or otherwise directly associated with a participating team. But if you are a team coach and have a colleague who might be interested in contributing, please drop a word to them. If you have a creative team member who will be an alumnus or alumna next year, you might do the same.

Anyone interested in participating as a member of the authoring team for 2015 can send their contact information to this year's head judge for the region, Steven Zeil, at `zeil@cs.odu.edu`.

Problem	Problem Name	Balloon Color
A	Hy-phe-na-tion Rulez	Orange
B	A Stable Relationship	Green
C	All Things Being Equal	Silver
D	Everything in Excess!	Pink
E	Not Sew Difficult	Red
F	Tight Knight	Yellow
G	Stealth	Purple
H	Verti-Words	Black

Problem A: Hy-phe-na-tion Rulez

Word processors often split a word across lines using hyphenation, a technique requiring some knowledge of where the syllables in the word are divided. Generally, the word processor knows a handful of basic rules for dividing words into syllables, and then keeps a dictionary of known exceptional cases.

Write a program that accepts a list of words (consecutive string of non-whitespace characters) as input and prints each word, one per line, with hyphens inserted at each possible hyphenation point as defined by the following rules:

1. If you see the pattern *vowel-consonant-consonant-vowel*, hyphenate between the two consonants. (For the purpose of this program, the vowels are 'a', 'e', 'i', 'o', 'u', and 'y'. 'y' will always be treated as a vowel.)

2. If you see the pattern *vowel-consonant-vowel*, hyphenate before the consonant unless the second vowel is an 'e' and occurs at the end of the word.

3. The following character sequences are never divided by hyphens:

   "qu", "tr", "br", "str", "st", "sl", "bl", "cr", "ph", "ch".

   For the purpose of applying rules 1 and 2, these are all considered to be a single consonant.

4. Upper and lower-case distinctions are ignored for the purpose of applying the above rules, although the case in the input word must be preserved in the output.

Input

Input will consist of a single data set terminated by a line containing only "===" (three equal signs).

The data set consists of multiple lines of text, each line containing $0 \ldots 80$ characters (not including the line terminator).

Output

Each word from the input is to be printed on a single line, with hyphens inserted at all valid hyphenation points.

Example

Input:

Given the input

```
Word processors often split a word across lines using hyphenation,
a technique requiring some knowledge of where the syllables in
the word are divided.

The rules given in this problem are a bit crude. But they represent a
good starting point.
===
```

the output would be

Output:

```
Word
pro-ces-sors
of-ten
split
a
word
a-cross
li-nes
u-sing
hy-phe-na-tion,
a
tech-nique
re-qui-ring
some
know-led-ge
of
where
the
syl-la-bles
in
the
word
are
di-vi-ded.
The
ru-les
gi-ven
in
this
pro-blem
are
a
```

```
bit
cru-de.
But
they
rep-re-sent
a
good
star-ting
point.
```

Problem B: A Stable Relationship

You recently bought a 3D printer. Using the 3D printer has been a lot of fun, but, from time to time, your attempt to print an object is spoiled because the object topples while being printed.

An object is printed by depositing layers of material from bottom to top. We will assume that the layers are formed of identically-sized cubes of the printing material, each of which occupies a cell in a 3D grid. All cubes are made of the same material and therefore weigh the same. We will also assume that the forces exerted by the 3D printer on the object during printing are negligible.

Whether the object will topple will be determined only by the location of the center of mass of the partially printed object relative to the base layer of the object. Specifically, let P be the horizontal plane on which the bottom or base layer of the object lies. Let C be the projection on P of the partially- or wholly-printed object's center of mass. The object will not topple if and only if there exists a triangle strictly containing C, with each of the three vertices lying under one of the printed blocks of the bottom layer. (The vertices may lie under any combination of 1, 2, or 3 such blocks.)

The material in each layer is printed such that, if the object will not topple after the whole layer is deposited, it will not topple at any time during printing that layer. Moreover, we will assume that the printed object will always be in a single piece throughout the printing process and that there is at least one block printed on the base layer.

Given the 3D object to be printed, determine whether the object will topple during printing.

Input

The input will consist of one or more test cases.

Each test case starts with 3 positive integers, each less than or equal to 200: The width (w) , length (l), and height (h) of the object. End of input is signalled by a line containing 3 zeroes.

After a line containing positive w, l, and h, the input contains h blocks representing the layers of the object from bottom to top. Each block has l lines of w characters each.

The characters used on each line are '.', representing an empty grid cell, and/or '#', representing a grid cell where material will be deposited.

Output

For each test case, print exactly one line with either Will topple or Will not topple. If the object will topple during printing or after printing is complete, print Will topple. Otherwise, print Will not topple.

Example

Input:

Given the input

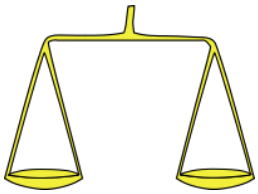
```
3 3 2
###
.#.
#.#
...
.#.
...
3 1 3
..#
.##
###
0 0 0
```

the output would be

Output:

```
Will not topple
Will topple
```

Problem C: All Things Being Equal



A pan balance consists of a horizontal arm, able to pivot around its center, with a light pan hanging from each end on which items of various weights can be placed. When the weights in the pans on each side are equal, the arm stays horizontal. If the weights on each side are unequal, the arm dips down on the heavier side.

A bored apothecary who received a mistaken shipment of several hundred pan balances began to rearrange them so that some of the pans were replaced by strings from which hung other pan balances. in a fashion similar to a mobile but restricted to having each bar pivoting about its center. He placed weights in the remaining pans sufficient to guarantee that each bar was balanced.

The pans and connecting bars are made from a lightweight material. Each pan and each bar weighs one gram. The weight of the strings may be ignored.

Adding to this construction day by day, he eventually had a rather impressive construction that he was pleased to display in his store. Periodically, he would disassemble it for cleaning or to move it to another location, but he had kept detailed notes on the weights in each pan to make reconstruction easy.

All was well until, one day, someone spilled coffee on his notes, leaving him unable to read the weight values recorded for several pans.

Given the remaining partial description of the construction, determine what weights, in grams, need to be placed in the pans whose notes were obscured in order to restore the balance.

Input

Input will consist of one or more test cases.

Each test case will consist of a description of the top balance arm.

A balance arm is described by an expression consisting of a left square bracket "[" followed by an expression x, followed by an expression y, followed by a right square bracket "]", where x and y can each be replaced by any of:

- an integer indicating the number of grams of weight in a pan, or
- a single upper-case alphabetic character, indicating a pan whose associated weight is unknown, or
- another balance arm expression indicating a pan balance suspended from one end of this bar in lieu of a pan.

Some alphabetic characters may be repeated in two or more positions, indicating that the apothecary's notes imply indicated that one pan contained the same weight as another, whose exact value is unknown.

No more than 50 balance arms will be used in any input case.

End of input will be indicated by a line containing the character sequence "[]".

Output

If there is a unique valid solution to the problem of adding weights to the empty pans to restore balance, then print, for each alphabetic symbol used for an empty pan, a line consisting of the alphabetic symbol identifying that pan, a single blank, and then the number of grams to be placed in that pan.

These lines should appear in ascending order of the alphabetic identifiers. The gram weights should be printed as a floating point number to two decimal places of precision. A valid solution will involve only non-negative values for the weights.

If there is no valid solution that would restore balance, print a line consisting of the word "NONE".

If there are multiple valid solutions that would restore balance, print a line consisting of the word "MANY".

Example

Input:

Given the input

```
[ [A B] 5]
[ Z
  [A [ 2 3 ] ]
]
[]
```

the output would be

Output:

```
A 1.50
B 1.50
NONE
```

Problem D: Everything in Excess!

The prime factorization of a positive integer n is the list of n's prime factors, together with their multiplicities:

$$n = \prod_{i=1}^{k} p_i^{m_i}$$

where the p_i are the factors (prime numbers) and the m_i are the corresponding multiplicities.

The *excess* of n is defined as the sum of the multiplicities ($\sum_{i=1}^{k} m_i$) minus the number of factors (k). It describes the number of times that factors get "re-used" in the factorization. For example, the excess of 8 is 2, the excess of 16 is 3, and the excess of 100 is 2.

Given a pair of integers $n_0 \leq n_1$, print the integer n that has the largest excess of any integer in the range $n_0 \ldots n_1$ (inclusive).

Input

The input will consist of one or more test cases.

Each test case will be presented on a single line as a pair of integers, in the range $2 \ldots 10,000,000$, denoting the values n_0 and n_1 as described above.

End of input will be indicated by a line containing "0 0".

Output

For each test case, print a single integer indicating the value in the range $n_0 \ldots n_1$ with the largest excess. If two or more values in the range tie for the largest excess, print the lowest such value.

Example

Input:

Given the input

```
2 11
600 700
0 0
```

the output would be

Output:

```
8
640
```

Problem E: Not Sew Difficult

A quilt will be made by laying a number of rectangular pieces of fabric onto a square cloth backing. The rectangular pieces will all be laid with one edge parallel to an edge of the cloth backing.

We plan to sew the overlapping pieces together, and need to know the maximum thickness of fabric (not counting the backing) that we will need to push a needle through at any point.

The rectangles will be positioned at non-negative integer coordinates on a $100,000$ by $100,000$ grid with axes defined by the cloth backing and one corner of the backing treated as the origin. All rectangular pieces will lie entirely within the bounds of the backing cloth.

Pieces overlap only if they do so along a non-zero area. Pieces that are simply adjacent along an edge or at a corner point are not considered overlapping.

Input

The input will consist of one or more test cases.

Each test case begins with a line containing an integer N, $1 \leq N \leq 1000$, denoting the number of rectangles. (End of input is signalled by a non-positive value for N.)

This is followed by N lines, each containing four non-negative integers $x_1\ y_1\ x_2\ y_2$, defining the coordinates of two opposite corners of a rectangle.

Output

For each dataset, print a single line containing an integer D, denoting the maximum depth of overlapping pieces of fabric.

Example

Input:

Given the input

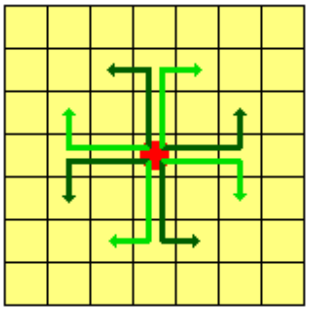
```
4
0 0 10 10
1 1 9 9
4 4 10 10
3 3 5 6
3
100 100 200 200
50 60 200 350
150 250 160 260
-1
```

the output would be

Output:

```
4
2
```

Problem F: Tight Knight



A knight in chess can move from its current position on a chessboard to any empty square that is either

- two steps vertically and one step horizontally, or
- one step vertically and two steps horizontally

away from its current position. Thus a knight positioned in the middle of an otherwise empty board could move to any of eight different locations, as shown in the accompanying diagram.

The empty/occupied status of any intermediate squares is irrelevant. A knight can only be blocked by an obstacle on the actual destination square.

Consider an n x m chessboard, $1 \leq n \leq 1000$, $1 \leq m \leq 1000$. A knight has been placed on square (i, j) of the board, where the rows and columns are numbered beginning at 1, so that $1 \leq i \leq n$, $1 \leq j \leq m$. There are c obstacles on squares of the board, $0 \leq c \leq 5000$. The knight cannot move to the squares with obstacles.

Can we prevent the knight from reaching another square (k, l), $1 \leq k \leq n$, $1 \leq l \leq m$, by adding at most one obstacle?

Input

Input may include multiple test cases.

Each test case starts with seven integers on a single line separated by spaces: n, m, i, j, k, l, c. End of input is signalled by a line containing seven integers with n being zero.

Following that first line of the test case are c lines, each with two integers x, y, specifying the location (x, y) of each of the obstacles. $1 \leq x \leq n$, $1 \leq y \leq m$. No obstacle will be placed at (i, j) or (k, l).

Output

For each test case, print exactly one line of output. If the knight cannot reach cell (k, l) or can be prevented from reaching cell (k, l) by adding at most one obstacle (at a location other than (i, j) or (k, l)), print 'Yes'. If not, print 'No'.

Example

Input:

Given the input

```
4 4 1 1 4 4 2
1 4
```

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

the output would be

Output:

```
No
Yes
```

Problem G: Stealth

A submersible robot is designed to walk, crab-like, across the ocean floor. It is presented with a rectangular area measuring w by h (integers, measured in meters) that it must cross, starting from $x = 0$ and ending at $x = w$.

$$1 \leq w \leq 100, 1 \leq h \leq 100$$

A series of N sonar probes, $0 < N \leq 40$ are suspended above the course, at positions (x, y, z) with $0 \leq x \leq w$, $0 \leq y \leq h$, $1 \leq z \leq 10$.

The robot may start at any $(0, y_0, 0)$ position where y is an integer, $0 \leq y_0 \leq h$. Its goal is to reach some position $(w, y_1, 0)$, where y_1 is similarly constrained.

Each second, the robot can move 1m in a positive or negative X direction or it can move 1m in a positive or negative Y direction.

At the end of the second, all of the sonar probes emit a ping. The probability of probe i detecting the robot with a ping is $1/d_i^2(x, y)$, where $d_i(x, y)$ is the distance of the robot's location $(x, y, 0)$ from the i$^{\text{th}}$ probe. The probability that the i$^{\text{th}}$ probe fails to detect a robot at $(x, y, 0)$ is therefore $1 - 1/d_i^2(x, y)$, and the probability that none of the probes detect the robot at $(x, y, 0)$ is

$$\prod_{i=1}^{N} \left(1 - \frac{1}{d_i^2(x, y)}\right)$$

If the robot follows a path $((x_0, y_0, 0), (x_1, y_1, 0), \ldots, (x_k, y_k, 0))$, the probability that it will go undetected along the entire path is:

$$\prod_{j=0}^{k} \prod_{i=1}^{N} \left(1 - \frac{1}{d_i^2(x_j, y_j)}\right)$$

Find a path that minimizes the chances of the robot being detected at any point along its journey.

- The ping occurring just after the robot reaches $x = w$ should be included in the probability of detection.
- The robot may not roam outside of the wxh area.
- The robot always has $z = 0$ in its (x, y, z) location.

Input

Input will consist of one or more data sets.

Each data set begins with a line containing a single integer, N, denoting the number of probes. A zero value for N signals the end of input.

The next line of the data set contains two integers w, and h, giving the dimensions of the course. (Note, the legal X coordinates range from 0 to h, inclusive, not from 0 to $h - 1$. Similarly, legal Y coordinates range from 0 to w, inclusive.)

This is followed by N lines, each containing three integers, giving the (x, y, z) coordinates of one probe.

Output

For each data set, print the length of the path that minimizes the probability of the robot being detected during its journey. If there are multiple paths that achieve the same minimum probability, print the length of the shortest such path.

Example

Input:

Given the input

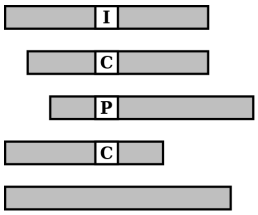
```
4
4 4
2 0 5
2 1 5
2 2 5
2 4 5
8
100 4
2 1 2
2 2 2
2 3 2
2 4 2
99 0 2
99 1 2
99 2 3
99 3 3
0
```

the output would be

Output:

```
4
104
```

Problem H: Verti-Words



Given a target phrase and a paragraph of text, reposition the lines of the paragraph by adding blank characters to the left of each line so that the target phrase appears vertically, reading down, in some column of the rearranged text.

- The original lines of the paragraph may not be split into two or more lines, nor may the characters already within a line be rearranged.
- The characters in the target word must appear on consecutive lines with no blanks or other characters between them. Upper/lower case is significant.

If there are multiple ways to reposition the lines to reveal the target phrase, the tie breakers are, in decreasing order of precedence,

- Minimize the total width of the rearranged paragraph.
- Minimize the total number of inserted blanks.
- Start the target phrase as close to the top of the paragraph as possible.
- Start the target phrase as close to the left as possible.

Input

Input will consist of one or more test cases.

Each test case will begin with a line containing the target phrase, left-justified. A target of "END" signals the end of input.

This is followed by $1 \ldots 20$ lines of text, each up to 80 characters in width, that comprise the paragraph. The end of the paragraph is signaled by an empty line (no characters other than the line termination).

Output

For each test case, print the paragraph with appropriate indentation supplied as blank spaces, denoting the desired solution.

If no solution is possible, print the paragraph unchanged.

Example

Input:

Given the input

```
acm
Given a target word and a paragraph of text,
rearrange the paragraph by adding blank characters to
the left of each line so that the target word appears
vertically in some column,
top-to-bottom in some column of the rearranged text.

Moo
If there are multiple ways to make this happen, tie breakers are,
in decreasing order of precedence,
* Minimize the total width of the rearranged paragraph.
* Minimize the total number of inserted blanks.
* Start the target phrase as close to the top of the paragraph as possible.
* Start the target phrase as close to the left as possible.

END
```

the output would be

Output:

```
Given a target word and a paragraph of text,
rearrange the paragraph by adding blank characters to
the left of each line so that the target word appears
        vertically in some column,
 top-to-bottom in some column of the rearranged text.
If there are multiple ways to make this happen, tie breakers are,
in decreasing order of precedence,
* Minimize the total width of the rearranged paragraph.
                              * Minimize the total number of inserted blanks
* Start the target phrase as close to the top of the paragraph as possible.
* Start the target phrase as close to the left as possible.
```

The target phrases can be seen more clearly below:

```
Given a target word and a paragraph of text,
rearrange the paragraph by adding blank characters to
the left of ea ch line so that the target word appears
        verti cally in some column,
 top-to-botto m in some column of the rearranged text.
If there are multiple ways to make this happen, tie breakers are,
in decreasing order of precedence,
* Minimize the total width of the rearranged paragraph.
                              * M inimize the total number of inserted blan
* Start the target phrase as cl o se to the top of the paragraph as possibl
* Start the target phrase as cl o se to the left as possible.
```