Deloitte.

Problem 1 – Consecutive Sums

Joe is a curious individual who enjoys math. He decided to count the number of unique sequences of positive integers whose sum is equal to a given number. To do this manually can take hours, so he decides to write a simple program to complete this task.

The number 6 can be written one way.
1+2+3.

The number 15 can be written two ways
4+5+6
7+8

Input

The input will consist of multiple input lines delimited by a new line character. EOF will signal the end of input. Each input line will consist of any number between 1 and 1000.

Output

The output will consist of a number representing the total number of different consecutive sums that add up to the number given in the input.

Sample Input

```
6
15
```

Output for the Sample Input

```
1
2
```

Deloitte.

Problem 2 – Mirrors

A room is full of mirrors. When we shine a laser at a mirror it is reflected. Maybe the reflected beam hits another mirror, maybe it doesn't. We want to know the position of the last mirror to receive any light without reflecting the light on to another mirror. The laser can be oriented in one of the four cardinal directions. Mirrors are oriented in one of the four inter cardinal directions.

Input

Input will consist of multiple Mirror Layout groups. EOF will signal the end of all input. Each mirror group will be identified with a "**START**" and "**END**" keyword. Each mirror group is made up of multiple lines. Each line defines either a light source or a mirror. Any line beginning with a "m" defines a mirror. Any line beginning with an "l" defines a light source. There will only be one light source, but any number of mirrors. Each line provides position and orientation information for the light source or mirror.

Light source:

l [x position] [y position] [n|s|e|w]

Mirror:

m [x position] [y position] [ne|se|nw|sw]

All position data will be integers in the range [0, MAX_INT]. East represents the +X axis, and south represents the +Y axis.

Output

Print the position of the last mirror to receive any light without reflecting the light on to another mirror. The result of each set should be printed on its own line.

Use the following format:

Last mirror: [x position], [y position]

Sample Input

```
START
l 2 2 e
m 20 2 sw
m 10 20 se
m 400 290 se
m 20 50 nw
m 10 50 ne
END
```

Output for the Sample Input

```
Last mirror: 10, 20
```

Deloitte.

Problem 3 – Strokes and Dots

A recent archaeological dig has revealed several artifacts etched with what appears to be characters of an ancient language. Many theories have emerged, everything from the possibility of an alien language, to the possibility of a language pre-dating Chinese writing 4000 years ago. Using the latest imaging technology, these ancient etchings are lifted from artifacts in the form of ASCII character grid. In order to effectively catalog and confirm these markings, data points such as the number strokes and dots need to be collected.

Your company has won the proposal to automate the process of scanning these grids and producing the desired datasets. The success of this process is critical to the company's future.

Given a glyph grid, scan it for asterisks (*) to detect the strokes and dots. A Stroke is any three or more consecutive asterisks. Strokes within a grid can run horizontally, vertically and diagonally. A dot is represented by a single asterisk which cannot be considered to be any part of a stroke, therefore it must NOT be connected to another asterisk horizontally, vertically or diagonally.

Input

Input will consist of multiple "glyphs" encapsulated between "**START**" and "**END**" keywords. Each glyph will be made up of multiple lines of asterisks and spaces delimited by newline characters. Each glyph will be no larger than 100 characters by 100 characters.

Output

Output will consist of a number of strokes "# Strokes" and the number of dots "# Dots" separated by a comma. Each output should appear on a separate line.

Sample Input

```
START
**** *
 *
  *  *
* *
******
END
START
  *
** **
*   *
  *
END
```

Output for the Sample Input

```
4 Strokes, 2 Dots
2 Strokes, 1 Dots
```

Deloitte.

Problem 4 – Not Pascal's Triangle

Nick was in his programming class talking about Pascal's Triangle and how easy it is to produce; the Instructor smiled and said I have a triangle let's see how easy you solve it. The instructor began writing out a triangle on the board, and asked Nick to make a program to solve it.

Here is the triangle pattern to 8 lines

```
       11
       21
      1211
     111221
     312211
    13112221
   1113213211
31131211131221
```

Input

Input will consist of a set of integers delimited by semi-colons and ended with an EOF

The integer will be from 1-16.

Output

For each integer, print the triangle to that many lines. Format it to form a rough triangle.

Sample Input

```
1;4;6;
```

Output for the Sample Input

```
11
  11
  21
 1211
111221
   11
   21
  1211
 111221
 312211
13112221
```

Deloitte.

Problem 5 – Doom Negative One

In the classic game Doom, one of the goals in the game was to navigate a maze like level to find keys to unlock doors to allow the player to make it to the next stage. What we have devised here is new game a maze with a series of doors and keys, a starting point and an ending point. The object of the game is to make it from the starting point to the end point. The only problem is there is a maze of doors in your path. Write a program that takes in a maze and reports whether or not it is solvable.

Movement in the maze is limited to the four cardinal directions. (N,S,E,W)

Input

Input will consist of multiple sets surrounded by a "**START**" and "**END**" keyword. EOF will trigger the end of all input sets. Each set will layout an ASCII art maze: Asterisk (*) will represent impassible obstacles; upper-case letters (A-Z) will represent locked doors; lower-case letters (a-z) will represent keys to those doors. The starting location is indicated by the number 1, and the ending location is indicated by the number 0. The ASCII Map shown is not the complete world, but only a defined area of an otherwise infinite empty grid. Tab Characters will not be used to describe whitespace, that will be left to the space character where needed.

Output

The output should consist of one of two possible phrases for each set of input. Those phrases are as follows: "GG NEXT MAP!" if it is possible to navigate from the start to end position. "QQ IM LOST!" if it is not possible to navigate the maze. Each output phrase should appear on its own line.

Sample Input

```
START
****A*****0
*         **
*           *
* a         *
*           *
*1          *
***********
END
START
*****
*1*0*
*****
END
```

Output for the Sample Input

```
GG NEXT MAP!
QQ IM LOST!
```

Deloitte.

Problem 6 – Optimal Tic Tac Toe

To play tic tac toe and never lose you simply have to prioritize your moves based on the following questions. Starting from the top if the answer is yes make the move associated with that question.

1. Can you win?
2. Can you block your opponent from winning?
3. Can you fork? (forking is setting yourself up to win in more than one way)
4. Can you block your opponent from forking?
5. Can you play the center?
6. Can you play the opposite corner of your opponent?
7. Can you play an empty corner?
8. Can you play an empty side?

Write a program that will make the "best" move regardless of the current state of the board. You are player X and for each board O has just moved and now it's your turn.

Input

Input will be multiple game boards, EOF will signal the end of all input. Each game board will be a 3x3 ASCII representation of a Tic Tac Toe board: consisting of exactly 3 lines of 3 characters not including newlines. Valid characters in a game board will be (X,O, space). Each game board will be separated by

"---".

Output

Output will be the same number of boards in the same order separated by "---" . Each with an additional "X" positioned by your program to the alleged "best" play position for each board.

Sample Input

```
XX
O
O
---
XO
   X
O
```

Output from Sample Input

```
XXX
O
O
---
XO
 XX
O
```

Deloitte.

Problem 7 – Classic Palindromes

Palindromes one of the most common wheels re-invented by programmers. Today we will ask you to write a program to detect palindromes in a block of text and print them. A palindrome is a sequence of characters that is the same frontwards and backwards. It has to be at least 3 characters in length, and cannot span multiple lines. White space and character case in input can be ignored.

Input

Input will consist of multiple lines of text. EOF will signal the end of all input.

Output

Output will consist of each palindrome found in the order that it was given in the input. Each palindrome should appear on its own line.

Sample Input

```
Racecar Radar
Lol dude
Stressed Desserts
```

Output from Sample Input

```
racecar
radar
lol
stresseddesserts
```

Deloitte.

Problem 8 – BGP Anyone

The classic problem of getting from here to there, we will define a series of bidirectional routes between two nodes to form a network. Then ask for directions from one node to another node across the network we just defined. The solution is to find the cheapest route between two nodes. When two or more solutions exist with the same cost the shortest route wins.

Input

The input will consists of multiple sets ending with EOF.
Each test set will consist of a non-negative integer indicating the number of routes to be declared followed immediately by those route sets ending with a semi-colon. Then a non-negative integer indicating the number of route lookups to be preformed immediately followed by the lookups and ended with a semi-colon.

Route sets will consists of three values separated by spaces: two string node names and a non-negative integer cost. Multiple route sets will be comma delimited.

Lookups will consist of an origin and a destination node strings separated by a space. Multiple lookup sets will be comma delimited.

Output

The output should be one line per lookup consisting of the path taken from origin to destination with the total cost of the trip. If no path exists between the two nodes “NO ROUTE” should be printed instead.

Sample Input

5a b 10,b c 10,c d 10,d e 30,e a 10;2a d,a g;1h k 12;1k h;

Output for the Sample Input

a b c d 30
NO ROUTE
k h 12