Problem 1—Hailstone Sequences

No one takes Freddie Freshman seriously because of his young age of 7. Most people, in fact, would be quite surprised to learn that he is a mathematical genius. Freddie, in particular, has been making a study of hailstone sequences.

The following algorithm defines this sequence:
1. Start with any positive integer, N.
2. If that number is odd, then multiply it by three and add one; otherwise, divide it by two.
3. Repeat as many times as desired.

The hailstone sequence probably eventually reaches the pattern 4-2-1 (which then loops indefinitely), although this has never been proven. Determine the length of the hailstone sequence up to the point where the first 1 is reached (counting both the starting number and the first 1).

INPUT SPECIFICATION. Each input case consists of an unsigned decimal integer. The last integer in the file is a –1. This –1 is not to be processed; it merely signifies the end of input. There may be any number of spaces and/or <**EOLN**>'s preceding, following, or separating these integers.

OUTPUT SPECIFICATION. The output cases should appear in the same order as their corresponding input cases. For each case, you should print "Case *c*: length = *l* for N = *N*" followed by two <**EOLN**>'s, where *c* is the case number, *N* is the input number, and *l* is the length of the sequence.

SAMPLE INPUT.

```
1<EOLN>
5<EOLN>
-1<EOLN>
<EOF>
```

SAMPLE OUTPUT.

```
Case·1:·length·=·1·for·N·=·1<EOLN>
<EOLN>
Case·2:·length·=·6·for·N·=·5<EOLN>
<EOLN>
<EOF>
```

Problem 2—Triangular Word Search

Ol' Weird Harold loves to do word search puzzles in the newspaper, but, because he's so weird, he likes to do word searches in triangular rather than rectangular grids. In such a grid, a word can be hidden forward or backward, horizontally or diagonally, but not vertically. Given a word search triangle and an input word, you are to compute how many times that word appears in the grid.

INPUT SPECIFICATION. Each input case begins with an unsigned decimal integer n ($n < 100$) indicating the number of rows in the grid followed by **<EOLN>**. The next n lines of the input case are the triangular word grid. Look at the samples for the formatting of this grid. Note that the number of letters in the ith. row is i, that there is one space between adjacent letters in a row, that there are no trailing spaces at the end of the line, and that the only characters in the grid itself are uppercase letters. Following each grid is the word to search for, followed by **<EOLN>**. The only characters in the word are uppercase letters; in particular, there are no spaces between the letters in this word. An extra **<EOLN>** follows each input case. The last input case is followed by "0**<EOLN>**". This line is not to be processed; it merely signifies the end of input.

OUTPUT SPECIFICATION. The output cases should appear in the same order as their corresponding input cases. Each output case consists of the following line: "Case c: w appears in the triangle n time(s)." c is the number of the case being processed. w is the word to be searched for. n is the number of times that word appears in the grid. Each output line is followed by exactly one **<EOLN>**. Note that in palindromes the same word can be found more than once (e.g. both forward and backward), but you should count that as only a single occurrence of the word.

SAMPLE INPUT.

```
6<EOLN>
·····B<EOLN>
····A·A<EOLN>
···T·T·T<EOLN>
··B·A·A·A<EOLN>
·T·A·B·B·B<EOLN>
B·A·T·M·A·N<EOLN>
BAT<EOLN>
<EOLN>
3<EOLN>
··B<EOLN>
·0·0<EOLN>
B·0·B<EOLN>
B0B<EOLN>
<EOLN>
0<EOLN>
<EOF>
```

SAMPLE OUTPUT.

```
Case·1:·BAT·appears·in·the·triangle·9·time(s).<EOLN>
Case·2:·BOB·appears·in·the·triangle·3·time(s).<EOLN>
<EOF>
```

Problem 3— Invisible Ink

As an army intelligence officer, Diana Prince sometimes has to crack codes. This sounds like exciting work, but, actually, for Diana, it is quite dull.

The particular code to which she is presently assigned took her all of about four seconds to crack. It is cute, though. The code is based completely on white space; it is as though the entire secret message were written in invisible ink.

The input message consists entirely of capital letters, spaces, and **<EOLN>**'s. Each letter is encoded as a line of spaces followed by **<EOLN>**. “A” is encoded as one space; “B” is encoded as two spaces, and so forth. A space itself is encoded as a single **<EOLN>**. The **<EOLN>** is encoded as two **<EOLN>**'s.

The primary advantage of the code is that paper copies of encoded messages cannot be analyzed. But, even so, it didn't take Diana long to break the code and lasso in the enemy agents.

INPUT SPECIFICATION. While most problems in this contest consist of multiple input cases, this input file will contain exactly one secret message, though, of course, the message could contain multiple lines. The input message will be exactly in the format described above.

OUTPUT SPECIFICATION. The output file should consist of the input message coded in invisible ink, exactly as described above.

SAMPLE INPUT.

```
HELP·ME<EOLN>
<EOF>
```

SAMPLE OUTPUT.

```
·····<EOLN>
·····<EOLN>
·····<EOLN>
·····<EOLN>
<EOLN>
·····<EOLN>
·····<EOLN>
<EOLN>
<EOLN>
<EOF>
```

Problem 4 — Invertible Years

As a crusading journalist, Clark Kent has been to nearly every nation on earth. During his travels, Clark has encountered many different cultures, languages, alphabets, and number systems. After encountering a strange system of numbers and letters that somewhat but not completely resembles English, Clark began thinking of invertible years. An invertible year is a year that reads the same upside down as right side up. The most recent invertible year was 1961. The next will be 6009. (Though fonts vary, we assume that "1", "0", and "8" invert as themselves and that "6" and "9" invert as each other.) Leading zeroes are unacceptable in invertible years. Even though 110 could be written as "0110", 110 is *not* an invertible year.

Clark is superb at mental arithmetic; however, most of us appreciate a little help from the computer. For any given year, you are to compute the first invertible year following the given year.

INPUT SPECIFICATION. Each data case consists of a single unsigned decimal integer no greater than 99999 representing the input year. Each input year is followed by exactly one <**EOLN**>. The last data case is followed by "0<**EOLN**>" This merely signifies the end of the input and should not be processed.

OUTPUT SPECIFICATION. The output cases should appear in the same order as their respective input cases. The output should be in the format "Case *c*: The first invertible year after *i* is *y*." Here, *c* is the case number; *i* is the input year; *y* is the desired output year. Each output case should be followed by exactly one <**EOLN**>.

SAMPLE INPUT.

```
1800<EOLN>
1900<EOLN>
1961<EOLN>
0<EOLN>
<EOF>
```

SAMPLE OUTPUT.

```
Case·1:·The·first·invertible·year·after·1800·is·1881.<EOLN>
Case·2:·The·first·invertible·year·after·1900·is·1961.<EOLN>
Case·3:·The·first·invertible·year·after·1961·is·6009.<EOLN>
<EOF>
```

Problem 5 — Logical Evaluation

Although Bruce Wayne was actually an excellent student in college (studying criminology and psychology), he does not evidently make much use of his expensive education these days, in that he was able to build his philanthropic organization, the Wayne Foundation, entirely with inherited wealth. However, despite the image his bored, womanizing persona might convey, Bruce's deductive mind is still just as sharp as it was in college. Bruce loves to play the logic games in the newspaper, and, while he could write a logical evaluation parser quite easily, he has left the task to you.

A logical statement may be:
A single capital letter (with assigned value of either TRUE (1) or FALSE (0))
~α (the negation of α)
(α|β) (the logical (inclusive) OR of α with β)
(α&β) (the logical AND of α with β)

Notice that | and & are ALWAYS surrounded by parentheses in these expressions. As a result, order of operations plays no part in the evaluation of logical expressions; each logical expression simplifies to a unique boolean value. You are to write a program to evaluate a logical expression. Bruce would do it himself but his other interests are presently driving him... (wait for it... wait for it...)
...batty.

INPUT SPECIFICATION. Each data case consists of two lines. The first line contains exactly 26 characters followed by <**EOLN**>. Each of these characters will be a zero or one and corresponds to the logical values of the capital letters. The first character is the logical value of A; the second is the logical value of B; and so forth. The second line will be no longer than 100 characters, and contain a valid logical expression as defined above, followed by <**EOLN**>. The expression will not contain any syntax errors. There will be no embedded spaces. There will be no extra parentheses (beyond those as defined in the expression). There will be no dirty tricks. The last data case will be followed by <**EOF**>.

OUTPUT SPECIFICATION. The output cases should appear in the same order as their respective input cases. The output should be in the format "Case *c*: TRUE" or "Case *c*: FALSE" depending on the evaluation of the logical expression, where *c* is the case number. Each output case should be followed by exactly one <**EOLN**>.

SAMPLE INPUT.

```
11111111111111111111111111<EOLN>
((P&~Q)|(Q&~P))<EOLN>
00000000000000000000000000<EOLN>
~P<EOLN>
00000000000000000000000000<EOLN>
~(~P&~Q)<EOLN>
<EOF>
```

SAMPLE OUTPUT.

```
Case·1:·FALSE<EOLN>
Case·2:·TRUE<EOLN>
Case·3:·FALSE<EOLN>
<EOF>
```

Problem 6 — Lighthouses

As a lighthouse keeper, Arthur Curry finds himself with copious free time. Although he spends a lot of time swimming in the ocean, and, presumably, making friends with fish, he has also taken up the time-honored tradition of ASCII art.

Arthur (no relation to King Arthur) would like you to draw pictures of lighthouses of various sizes.

INPUT SPECIFICATION. The input consists of several cases. Each case is an unsigned decimal integer greater than 1, representing the size of the lighthouse. The last case is followed by a zero. This zero should not be processed; it merely signifies the end of input. There may be any number of spaces and **<EOLN>**'s before, after, and/or between the integers in this file.

OUTPUT SPECIFICATION. The output cases should appear in the same order as their respective input cases. Although a full output specification is pointless for an ASCII art problem (just follow the examples below), there are a few things that should be made clear. First of all, the width of the lighthouse is 2*n*–1, where *n* is the size of the lighthouse. Second of all, there should be an extra **<EOLN>** character following each lighthouse. Although some lines must contain leading spaces for the picture to work, no line should contain trailing spaces.

SAMPLE INPUT.

```
3<EOLN>
4<EOLN>
0<EOLN>
<EOF>
```

SAMPLE OUTPUT.

```
··^<EOLN>
·/·\<EOLN>
X---X<EOLN>
|···|<EOLN>
|···|<EOLN>
|···|<EOLN>
X---X<EOLN>
<EOLN>
···^<EOLN>
··/·\<EOLN>
·/···\<EOLN>
X-----X<EOLN>
|·····|<EOLN>
|·····|<EOLN>
|·····|<EOLN>
|·····|<EOLN>
|·····|<EOLN>
X-----X<EOLN>
<EOLN>
<EOF>
```

Problem 8— Jigsaw Puzzle

Sally has a bad case of senioritious. When she's not watching the Brown Hornet, Sally whiles away her time doing jigsaw puzzles. As long as they're not too complicated, Sally can assemble them without too much difficulty. Right now, she's mastered puzzles with nine pieces! Given a jigsaw puzzle with nine pieces, your program should assemble it correctly. Each piece is a square with a positive integer along each edge. These pieces may not be flipped or rotated; their orientation is fixed. These pieces are to be arranged into a 3x3 grid such that any adjoining pieces have the same integer along their shared edge. If the puzzle can be assembled at all, the solution will be unique; however, it is possible that no solution will exist at all.

INPUT SPECIFICATION. The input file begins with an unsigned decimal integer indicating the number of input cases that will follow, followed by two <**EOLN**>'s. Each data case consists of nine lines, representing a piece of the puzzle. Each line contains four positive integers, separated by one space, and followed by one <**EOLN**>. The first number on the line is the top edge number; the other edge numbers are listed in a clockwise fashion. There is an extra <**EOLN**> following each input case.

OUTPUT SPECIFICATION. The output cases should appear in the same order as their corresponding input cases. Each output case begins with the following line: "Case *c*" followed by two <**EOLN**>'s, where *c* is the input case number. The next three lines correspond to the grid of the completed puzzle, with three numbers on each line, one space following each number and each line terminated by <**EOLN**>. Piece 1 is the first piece listed in the input case, piece 2 is the second piece listed in the input case, and so on. There is an extra <**EOLN**> following each output case.

SAMPLE INPUT.

```
2<EOLN>
<EOLN>
1·2·3·4<EOLN>
5·6·7·2<EOLN>
8·9·10·6<EOLN>
3·11·14·10<EOLN>
7·12·15·11<EOLN>
10·13·16·12<EOLN>
14·18·21·17<EOLN>
15·19·22·18<EOLN>
16·20·23·19<EOLN>
<EOLN>
1·2·3·4<EOLN>
5·6·7·8<EOLN>
9·10·11·12<EOLN>
13·14·15·16<EOLN>
17·18·19·20<EOLN>
21·22·23·24<EOLN>
25·26·27·28<EOLN>
29·30·31·32<EOLN>
33·34·35·36<EOLN>
<EOLN>
<EOF>
```

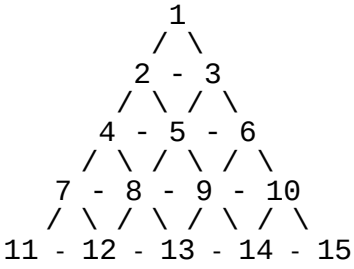
SAMPLE OUTPUT.

```
Case·1<EOLN>
<EOLN>
1·2·3·<EOLN>
4·5·6·<EOLN>
7·8·9·<EOLN>
<EOLN>
Case·2<EOLN>
<EOLN>
No·Solution<EOLN>
<EOLN>
<EOF>
```

Problem 9 — 15-Hole Peg Game

Bucky is called “Bucky” not only for his buck teeth but also for his skill at Buck-Buck, a game not unlike leapfrog but far more dangerous! A safer variant of leapfrog is the 15-hole peg game.

The 15-hole peg game consists of 15 holes (see diagram below) and 14 pegs. To start the game, a player places 14 pegs in the holes and leaves the 15th hole (their choice) empty. Then moves are made by taking any peg, jumping over another peg and landing in an empty hole. Pegs may be moved in 6 directions, horizontally and diagonally. Each jumped over peg is removed from the board. If a player can make 13 moves or jumps leaving 1 peg on the board, the player wins.



Given the current status of the peg board, determine the set of all possible moves. Note that you are NOT being asked to solve the game, merely to give the set of possible next moves given a specific configuration.

INPUT SPECIFICATION. Each input case consists of a line of unsigned decimal integers between 1 and 15. The line will contain at least one integer; no integer will appear more than once in the line; the integers will be separated by one space and the line terminated by <**EOLN**>. The last case is followed by “–1<**EOLN**>”. This –1 is not to be processed; it merely signifies the end of input. The integers in each line represent the empty holes on the board. All other locations contain pegs.

OUTPUT SPECIFICATION. The output cases should appear in the same order as their corresponding input cases. For each case, you should print “Case *c*” followed by <**EOLN**>. Then you should print for each move “*sp* -> *dp*<**EOLN**>”, where *sp* is the source position for a peg and *dp* is the destination position. If there is more than one valid move, you are to order the list so that lower source positions are printed before higher source positions; if there is more than one move from the same source position, you are to order them so that lower destination positions are printed before higher destination positions. An extra <**EOLN**> should follow each output case.

SAMPLE INPUT.

```
3<EOLN>
6·11<EOLN>
-1<EOLN>
<EOF>
```

SAMPLE OUTPUT.

```
Case·1<EOLN>
8·->·3<EOLN>
10·->·3<EOLN>
<EOLN>
Case·2<EOLN>
1·->·6<EOLN>
4·->·6<EOLN>
4·->·11<EOLN>
13·->·6<EOLN>
13·->·11<EOLN>
15·->·6<EOLN>
<EOLN>
<EOF>
```