Problem A
Radix 3
Input: radix3.txt

The Great Sand Council (GSC) of the planet Phleebutt (apologies to Sierra) have devised a base 3 number system to suit their physiology. The symbols used to represent the three valid digits of their number system are ‘`0`’, ‘`1`’, and ‘`-`’, following the unusual configuration of their *hands* (don’t ask). The decimal counterparts of the three symbols are, respectively, 0, 1 and -1.

Each position in a number has a value three times greater than the position immediately to its right. For example, the number ‘`10-`’ has the value 8 in decimal, since $1 \times 9 + 0 \times 3 + (-1) \times = 8$. Similarly, the number ‘`-1`’ has the decimal value -2, since $(-1) \times 3 + 1 = -2$. GSC representations of integers never start with a digit of zero, except for ‘`0`’ (a single zero) which has the decimal value 0.

You have to write a program that can convert 32-bit signed decimal integers into their equivalent GSC representations.

Input

Your program will receive a list of integers as input. Each integer will appear in a separate line and be between -2^{31} and $2^{31} - 1$. The end of list is indicated by the end of file.

Output

You have to echo the input numbers followed by their GSC representations, as shown in the Sample Output (including all extra symbols).

Sample Input

```
10
2
-17
42
1024
```

Output for the Sample Input

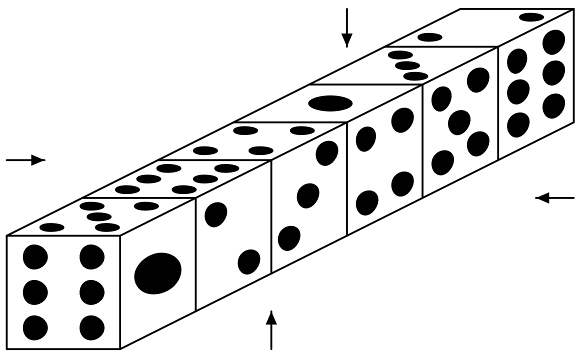
```
10 = 101 GSC
2 = 1- GSC
```

```
-17 = -101 GSC
42 = 1---0 GSC
1024 = 111-0-1 GSC
```

Problem B
Dice Instant Insanity
Input: dice.txt

Instant Insanity is a puzzle played with four cubes. Each face of the cubes is colored with red, blue, green, or yellow. The goal of this puzzle is to arrange the four cubes in a line so that all four colors appear on every long side of the line.

Now let us consider the modified version of Instant Insanity. In the modified version, we play with six different dice instead of four colored cubes. Each die has all numbers from one to six, but the sum of numbers on the opposite faces is not always seven. Our goal is to arrange the six dice in a line so that all six numbers appear on every long side. The figure below illustrates how the dice should be arranged.



In this problem, you are required to count up the number of different ways to arrange given six dice on the condition described above.

Two arrangements should be regarded as same if one can be obtained by

- changing the order of the dice of the other, and/or
- rotating the other entirely.

Note that the appearance does *not* matter to identity of arrangments. In other words, arrangements should be regarded as different if the numbers even on faces inside the line of dice are different (except for the case mentioned above, of course).

Input

The first line of the input contains an integer that indicates the number of data sets.

The rest of the input is a series of data sets. A data set consists of six lines. Each line represents a die and contains six integers that indicates the numbers on the top, forward, right, left, backword, and bottom faces of the die respectively.

No pair of dice in the same data set are identical, that is, have rotation so that the numbers on every pair of corresponding face match.

Output

Print the number of arrangements in a line for each data set.

Sample Input

```
2
2 3 4 5 6 1
6 1 3 5 2 4
5 2 3 1 6 4
6 4 1 5 2 3
5 4 6 3 1 2
5 1 2 4 6 3
2 6 5 4 1 3
5 3 1 6 2 4
5 2 3 1 6 4
1 3 4 5 6 2
5 2 6 3 1 4
6 1 4 5 3 2
```

Output for the Sample Input

```
3
96
```

Problem C
Visible Squares
Input: squares.txt

You are given n squares in the coordinate plane whose sides are parallel to the coordinate axes. All the corners have integer coordinates and the squares do not touch or overlap.

A square is *visible* from a point O, if there are two distinct points A and B on one of its sides, such that the interior of the triangle OAB has no common points with any of the remaining squares.

You are required to count the number of squares visible from the origin point $(0, 0)$.

Note: It is not allowed to use classes in the `java.awt.*` package.

Input

The input file may contain several instances of the problem. Each instance is described as follows:

1. The first line contains an integer n ($1 \leq n \leq 1000$), representing the number of squares.
2. Each of the following n lines describes a square: it contains a triple of integers x y l ($1 \leq x, y, l \leq 10000$), where coordinate (x, y) is the lower left corner (the corner with the least x and y coordinates) and l is the side length of that square.

The input file is terminated by a line only containing the integer 0 (zero). All numbers are separated by an arbitrary number of blank spaces.

Output

For each instance of the problem, your program should print a line containing the number of squares visible from the origin.

Sample Input

```
3
2 6 3
1 4 1
3 4 1
```

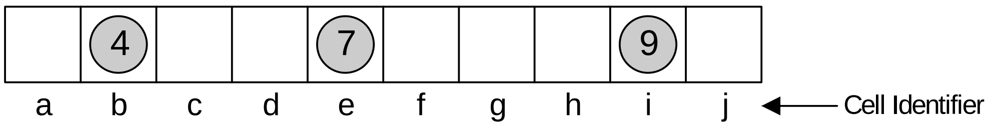
```
4
1 2 1
3 1 1
2 4 2
3 7 1
0
```

Sample Output

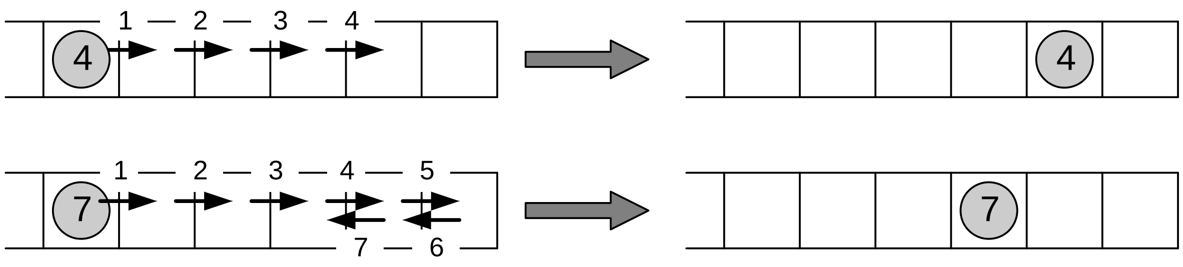
```
3
2
```

Problem D
Disappearance No.0
Input: zero.txt

Disappearance No.0 is a puzzle game invented by a reader of a computer magazine. The goal of this game is to erase all stones placed on a game field by moves. The game field consists of ten cells identified by alphabets from `a` to `j`. At the beginning of the game, some cells contain stones as shown in the figure below as an example. Each stone is labeled by a digit from 1 to 9. There may be more than one stone with the same label, though no cell may contain more than one stone.



At each move, a player chooses a stone to move among ones on the game field, as well as a direction of right or left. Then the chosen stone moves in the chosen direction by the number represented by the digit labeling the stone. In case the stone reaches the rightmost or leftmost cell, it moves in the opposite direction by the remaining number. The figure below illustrates movements of stones.



A player is not allowed to choose to move a stone in the rightmost cell to the right or to move a stone in the leftmost cell to the left, as it is nonsense to do so.

When the stone moves to the cell containing another stone, the two stones are merged into one stone labeled by the last digit of the sum of the two labeling digits. For example, merging stones labeled by 9 and labeled by 4 yields a stone labeled by 3, since the last digit of 13 $(= 9 + 4)$ is 3. The merged stone disappears if it is labeled by zero.

You are requested to write a program that shows the shortest sequence of moves to erase all stones on the game field, given a set of initial states.

Input

The first line contains a positive integer n representing the number of test cases. Each of the following n lines contains ten characters representing the initial state of the cells from `a` to `j` respectively, where a digit indicates the cell contains a stone labeled by it and a hyphen (`-`) indicates the cell contains no stone.

The first test case of the Sample Input corresponds to the example shown in the problem.

Output

For each test case, you should output the sequence printing one move per line. Each move should consist of two characters: the first character denoting the direction, where `R` and `L` indicate right and left respectively, and the second character denoting the identifier of the cell occupied by a stone to move. No extra blanks should appear.

If there is more than one acceptable sequence, you may output any one of them. If there is no sequence to erase all stones, you should output "`Impossible`" (without quotation marks) in a line.

Output a blank line after each test case.

Sample Input

```
2
-4--7---9-
12-3--4---
```

Output for the Sample Input

```
Li
Rb

Ra
Rd
Lg
```

Problem E
Walk an Equation
Input: equation.txt

A $m \times n$ table is filled with symbols from the set $S = \{\texttt{0}, \texttt{1}, \texttt{2}, \ldots, \texttt{9}, \texttt{+}, \texttt{-}, \texttt{*}, \texttt{(}, \texttt{)}\}$, satisfying $3 \le m, n \le 8$. The symbols `0`, `1`, `2`,. . ., `9` represent digits, and `+`, `-`, and `*` represent addition, subtraction, and multiplication respectively. The notation `(` stands for opening parenthesis and `)` for closing one.

You have to trace all possible paths starting from any cell in the first row to any cell in the last row. Any path should yield a valid numeric equation that contains exactly one equal sign. You are allowed to move only in four primary directions: up, down, left, and right. No diagonal moves are allowed. The numerical contents of adjacent cells along the path may form a number, but that number should not exceed more than four digits or contain leading zeros (except for 0, which should consist of a single digit of zero). You cannot use any cellular content more than once in an equation. Symbols `+` or `-` cannot be used for positive or negative signs.

The figure below is a tabular representation of the sample input.

5	2	3	+	=
5	6	+	=	-
-	1	=	8	6
2	9	5	6	-
1	5	6	3	2

You are required to print all possible valid equations.

Input

The input file comprises of several blocks. Each block begins with a line containing two integers for m (the number of rows) and n (the number of columns), separated by one or more spaces. The rest of the block is an $m \times n$ table filled with elements from the set S separated by one or more spaces.

Blocks are separated by a blank line. The input file is terminated by two zeros on a line by itself following the last block.

Output

For each block, first print its number (Table 1, Table 2, etc.) and then all possible equations.

Each line should represent a single equation. Equations may be printed in any order, but no equation is allowed to appear more than once even if the equation can be obtained by more than one paths. In this problem, equations are identical if and only if they match literally.

If no equation could be obtained from a block simply print "`No Solution`" instead of equations.

Print a blank line after each block. No extra spaces or blank lines should be printed.

Sample Input

```
5 5
5 2 3 + =
5 6 + = -
- 1 = 8 6
2 9 5 6 -
1 5 6 3 2

0 0
```

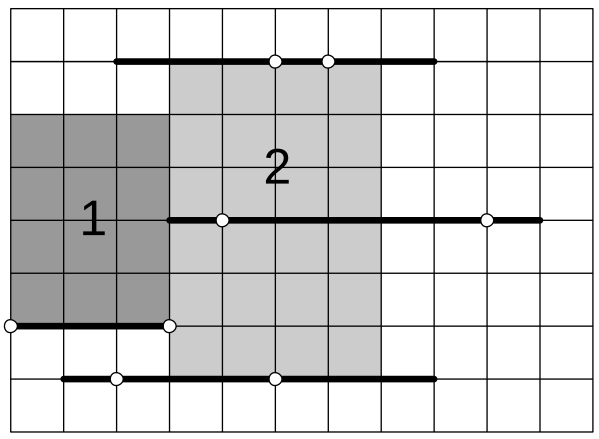
Output for the Sample Input

```
Table 1
23+61=86-2
55-1=56-2
```

Problem F
Library
Input: library.txt

Castaway Robinson Crusoe is living alone on a remote island. One day a ship carrying a royal library has wrecked nearby. Usually Robinson brings any useful stuff from the shipwreck to his island, and this time he has brought a big chest with books.

Robinson has decided to build a bookcase for these books to create his own library. He cut a rectangular niche in the rock for that purpose, hammered in wooden pegs, and placed wooden planks on every pair of pegs that have the same height, so that all planks are situated horizontally and suit to act as shelves.



Unfortunately, Robinson has discovered that one especially old and big tome does not fit in his bookcase. He measured the height and width of this tome and has decided to redesign his bookcase in such a way, as to completely fit the tome on one of the shelves, taking into account locations of other shelves and the dimensions of the niche. With each shelf in the bookcase, one of the following operations should be made:

1. Leave the shelf on its original place.
2. Move the shelf to the left or to the right.
3. Shorten the shelf by cutting off a part of the plank and optionally move it to the left or to the right.
4. Move one of the pegs to a different place at the same height and move the shelf to the left or to the right.

5. Shorten the shelf by cutting off a part of the plank, move one of the pegs to a different place at the same height, and optionally move the shortened shelf to the left or to the right.

6. Remove the shelf from the bookcase along with both supporting pegs.

We say that the shelf is properly supported by its pegs, if exactly two distinct pegs support the shelf and the center of the shelf is between its pegs or coincides with one of the pegs. The original design of Robinson's library has all the shelves properly supported by their pegs and lengths of all shelves are integer number of inches. The Robinson may move and/or cut off planks only by an integer number of inches, because he has no tools for more precise measurements. All remaining shelves after the redesign must be properly supported by their pegs.

You are to find the way to redesign Robinson's library to fit the special old tome without changing original design too much. You have to minimize the number of pegs that are to be removed from their original places during the redesign (operations 4 and 5 remove one peg, and operation 6 removes two pegs). If there are different ways to solve the problem, then you are to find the one that minimizes the total length of planks that are to be cut off (operations 3 and 5 involve cutting something from the planks, and operation 6 counts as if cutting off the whole plank). Width of planks and diameter of pegs shall be considered zero.

The tome may not be rotated. The tome should completely (to all its width) stand on one of the shelves and may only touch other shelves, their pegs or niche's edge.

Input

The first line of the input file contains an integer number that represents the number of test cases. Then test cases follow. They are separated by a blank line.

The first line of each test case contains four integer numbers *XN*, *YN*, *XT*, and *YT*, separated by spaces. They are, correspondingly, width and height of the niche, and width and height of the old tome in inches ($1 \leq XN, YN, XT, YT \leq 1000$).

The next line contains a single integer number N ($1 \leq N \leq 100$) that represents the number of the shelves. Then N lines follow. Each line represents a single shelf along with its two supporting pegs, and contains five integer numbers y_i, x_i, l_i, x_{1i}, x_{2i}, separated by spaces, where:

- y_i ($0 < y_i < YN$) is the height of the ith shelf above the bottom of the niche in inches,
- x_i ($0 \leq x_i < XN$) is the distance between the left end of the ith shelf and the left edge of the niche in inches,
- l_i ($0 < l_i \leq XN - x_i$) is the length of the ith shelf in inches,
- x_{1i} ($0 \leq x_{1i} \leq l_i/2$) is the distance between the left end of the ith shelf and its leftmost supporting peg in inches, and

- x_{2i} ($l_i/2 \leq x_{2i} \leq l_i; x_{1i} < x_{2i}$) is the distance between the left end of the ith shelf and its rightmost supporting peg in inches.

All shelves are situated on different heights and are properly supported by their pegs. The problem is guaranteed to have a solution for the input data.

Output

For each case, print a line containing two integer numbers separated by a space. The first one is the minimal number of pegs that are to be removed by Robinson from their original locations to place the tome. The second one is the minimal total length of planks in inches that are to be cut off during the redesign that removes the least number of pegs.

Sample Input

```
2

11 8 3 4
4
1 1 7 1 4
4 3 7 1 6
7 2 6 3 4
2 0 3 0 3

11 8 4 6
4
1 1 7 1 4
4 3 7 1 6
7 2 6 3 4
2 0 3 0 3
```

Output for the Sample Input

```
0 0
1 3
```

Problem G Permutation Input: permutation.txt

Given a permutation of n elements $(1, 2, ..., n)$: $A = (a_1, a_2, \ldots, a_n)$. We define a sequence $P(A) = (p_1, p_2, \ldots, p_{n-1})$ where $p_i = 0$ if $a_i > a_{i+1}$ and $p_i = 1$ if $a_i < a_{i+1}$. Given a permutation B, find the number of all permutations C where $P(C) = P(B)$ including the permutation B itself.

Input

The input text file contains several tests, each on a separate line. The first number in the test is n $(0 < n \leq 1024)$ followed by n numbers representing the permutation all of them separated by a single space. The last line in the input contains only 0 and should not be processed.

Output

The output should be written on the standard output. For each line in the input (excluding the last one, 0) you should write the result i.e. the number of the permutations having the same value for $P(A)$ when given the permutation A.

You may assume the result is less than 2^{32}.

Sample Input

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

Output for the Sample Input

```
1
5
```

Problem H
Get out!
Input: getout.txt

Tom was captured by aliens and brought into a mysterious region.

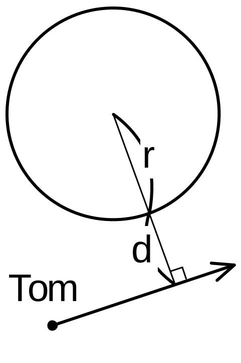
Since the event was all of a sudden, he did not understand what happened to him until he found some circular objects in the region. They looked really ominous so he got much horrified. He seriously began to consider escaping there, evading the objects, as soon as possible.

The region can be considered on a two-dimensional field. Tom was told that he got caused so that he could do nothing but move straight just in one direction, before the aliens dissapeared. Needless to say, he will fail to escape if he crashes with the objects as his move is blocked, though he may touch them.

In addition to that, when he passes by the objects, he will be robbed of his energy at the closest point to each object. The amount of energy absorbed by the object i is represented by

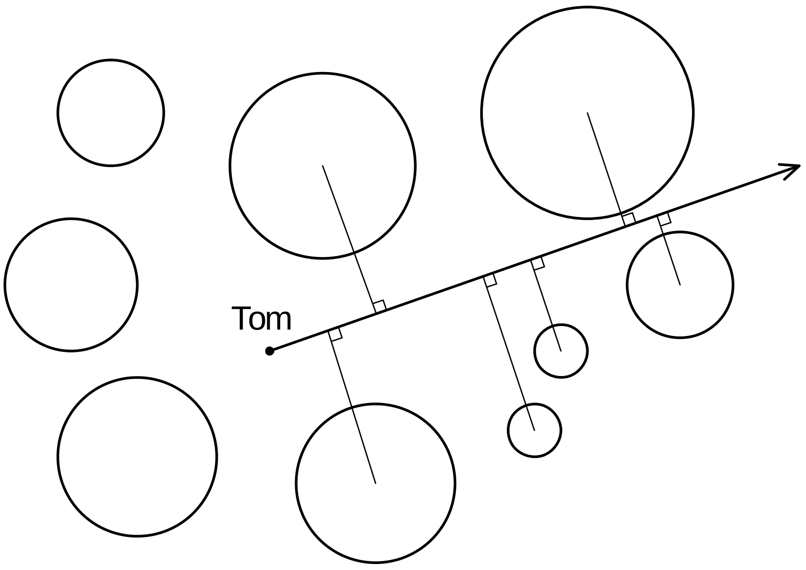
$$\begin{cases} P_i \times (r_i - d_i)/r_i & \text{if } d_i \leq r_i \\ 0 & \text{otherwise} \end{cases}$$

where P_i is a positive number called the *power* of the object i, r_i is the radius of the object i, and d_i is the distance between Tom and the circumference of the object when he gets to the closest point.



Tom gets unable to move anymore if he loses all his energy.

Your task is to write a program that shows Tom's remaining energy after his escape when he moves in such a direction that he remains his energy as much as possible.



NOTE: It is not allowed to use classes in the `java.awt.*` package.

Input

The input consists of a series of data sets.

The first line of a data set contains an integer N ($1 \leq N \leq 50$) that indicates the number of the circular objects. The $(i+1)$-st line contains four real numbers x_i ($-1000 \leq x_i \leq 1000$), y_i ($-1000 \leq y_i \leq 1000$), r_i ($0.1 \leq r_i \leq 500$), and P_i ($0.1 \leq P_i \leq 500$), which denote the x and y coordinates of the center, the radius, and the power of the object i respectively. The $(N+2)$-nd line contains three real numbers x ($-1000 \leq x \leq 1000$), y ($-1000 \leq y \leq 1000$), and E ($0.1 \leq E \leq 100$), which denote the x and y coordinates of Tom's initial position, and Tom's initial energy respectively.

There is always difference of at least 10^{-5} degrees between directions Tom can escape touching objects on his left-hand, and that Tom can escape touching objects on his right-hand.

An input line containing a single integer 0 indicates the end of input.

Output

For each data set, print a single line.

If Tom can escape from the region, print Tom's remaining energy with three digits after the decimal point. The value should not contain an error greater than 0.001.

Otherwise, print "`No way to escape`".

You may assume Tom remains his energy of at least 10^{-5} as long as he moves in such a direction that he can escape. In other words, there is not such a path that Tom can escape remaining his energy less than 10^{-5}. Note that the output line may contain `0.000`, though.

No extra spaces or blank lines should be printed.

Sample Input

```
4
50.0 0.0 15.0 10.0
-30.0 0.0 15.0 20.0
10.0 40.0 15.0 30.0
10.0 -40.0 15.0 40.0
10.0 0.0 100.0
0
```

Output for the Sample Input

```
95.425
```

Problem I
Frame Stacking
Input: frame.txt

Consider the following 5 picture frames placed on an 9 × 8 array.

```
........ ........ ........ ........ .CCC....
EEEEEE.. ........ ........ ..BBBB.. .C.C....
E....E.. DDDDDD.. ........ ..B..B.. .C.C....
E....E.. D....D.. ........ ..B..B.. .CCC....
E....E.. D....D.. ....AAAA ..B..B.. ........
E....E.. D....D.. ....A..A ..BBBB.. ........
E....E.. DDDDDD.. ....A..A ........ ........
E....E.. ........ ....AAAA ........ ........
EEEEEE.. ........ ........ ........ ........
   1        2        3        4        5
```

Now place them on top of one another starting with 1 at the bottom and ending up with 5 on top. If any part of a frame covers another it hides that part of the frame below.

Viewing the stack of 5 frames we see the following.

```
.CCC....
ECBCBB..
DCBCDB..
DCCC.B..
D.B.ABAA
D.BBBB.A
DDDDAD.A
E...AAAA
EEEEEE..
```

In what order are the frames stacked from bottom to top? The answer is `EDABC`. Your problem is to determine the order in which the frames are stacked from bottom to top given a picture of the stacked frames. Here are the rules:

1. The width of the frame is always exactly one character and the sides are never shorter than three characters.

2. It is possible to see at least one part of each of the four sides of a frame. A corner shows two sides.
3. The frames will be lettered with capital letters, and no two frames will be assigned the same letter.

Input

Each input block contains the height, h ($h \leq 30$) on the first line and the width w ($w \leq 30$) on the second. A picture of the stacked frames is then given as h strings with w characters each. Each block contains at least one frame.

The input may contain multiple blocks of the format described above, without any blank lines in between. All blocks in the input must be processed sequentially. Two lines each containing a single zero indicate the end of the input.

Output

Write the solution to the standard output. Give the letters of the frames in the order they were stacked from bottom to top. If there are multiple possibilities for an ordering, list all such possibilities in alphabetical order, each one on a separate line. There will always be at least one and at most one thousand legal ordering for each input block. List the output for all blocks in the input sequentially, with a blank line after each block.

Sample Input

```
9
8
.CCC....
ECBCBB..
DCBCDB..
DCCC.B..
D.B.ABAA
D.BBBB.A
DDDDAD.A
E...AAAA
EEEEEE..
0
0
```

Sample Output

```
EDABC
```

About Problem Set

Izumi, Yusuke

November 15, 2004

Clarifications

- Problem D: You may assume each test case contains at least one stone.

出典

Problem A	(Radix 3)	:	South Africa 2002
Problem B	(Dice Instant Insanity)	:	オリジナル（下記参照）
Problem C	(Visible Squares)	:	South America 1998
Problem D	(Disappearance No.0)	:	オリジナル（下記参照）
Problem E	(Walk an Equation)	:	Dhaka 1998
Problem F	(Library)	:	Northeastern Europe 2001
Problem G	(Permutation)	:	Southeastern Europe 2001
Problem H	(Get out!)	:	オリジナル
Problem I	(Frame Stacking)	:	South Africa 2001

補足事項

- 問題のあいまいさを減らすなどの理由で，ほとんどの問題に手を加えています．とくに Walk an Equation は相当な数の条件文を追加しています.
- Dice Instant Insanity は C MAGAZINE（1998 年 3 月号）の「C マガ電脳クラブ」で扱われていた問題をコンテスト向けに改変したものです.
- Disappearance No.0 は PJ（1992 年 12 月号）の「ゲームデザイン館」のコーナーに掲載された同名のゲームのルールを題材としています.