Problem A: Cut It Out!

Television's *Toddler Time*'s topic taught to toddler Tommy Tiwilliger today was triangles (and the letter T)! Tommy became so enamored by triangles that immediately after the show ended, he grabbed his safety scissors and the nearest sheets of paper he could find and started cutting out his own triangles. After about 15 minutes each paper had one triangular shaped hole cut out of it. But Tommy wasn't finished yet. He noticed he could divide each of the original triangles into two triangles with a single cut starting from one corner. He spent another 15 minutes doing just that. Things would have gone along swimmingly, except that Tommy's mother eventually came into the room and noticed that the original sheets of paper were part of a very important document for a legal case she was working on (involving a lover's triangle). After carefully removing Tommy from the room and counting slowly to 10 (a triangular number), she went about trying to reconstruct the pages after gathering together the now randomly scattered triangles. Your job is to help her by writing a little program to determine which triangles go where (and try to get it done before tomorrow's episode of *Toddler Time* on papier-mâchè).

Input

Each test case will start with an integer $n \leq 20$ indicating the number of holes cut out, followed by the coordinates of the holes, one hole per line. These holes are assumed to be numbered $1, 2, \ldots, n$. Following this will be the coordinates of the $2n$ triangles resulting from the bisections, one triangle per line. These triangles are assumed to be numbered $1, 2, \ldots, 2n$ and are listed in no particular order. The specification of any hole or triangle will have the form $x_1\ y_1\ x_2\ y_2\ x_3\ y_3$ where each x_i and y_i will be to the nearest thousandth and $|x_i|, |y_i| \leq 200$. No two holes will be congruent and no two triangles will be congruent. A value of $n = 0$ will terminate input.

Output

For each test case, you should output the case number and then n lines as follows:

```
Hole 1:  t1a, t1b
Hole 2:  t2a, t2b
...
Hole n:  tna, tnb
```

where `t1a, t1b` are the two triangles which fill hole 1, `t2a, t2b` are the two triangles which fill hole 2, etc. Always print the lower of the two numbers first on any line. Triangles should not be flipped over when filling a hole. Each test case will have a unique solution. Separate the output for each test case with a blank line. Note: when processing the triangles and checking for equality of lengths, angles or trigonometric values, you may assume that two items are equal if they differ by less than 0.01.

Sample Input

```
1
18.691 6.103 21.668 13.709 21.332 25.894
59.388 30.873 55.299 36.186 61.45 22.97
67.828 85.496 60.751 72.752 59.2 67.49
3
18.73 4.012 6.662 7.557 14.035 7.478
14.869 32.398 32.341 31.772 7.522 29.674
25.272 6.868 4.572 2.014 10.487 16.121
26.135 53.073 44.18 50.723 40.31 42.91
86.601 29.95 70.542 17.088 66.77 14.88
90.344 89.528 92.179 88.665 87.99 82.54
39.327 62.11 35.033 57.127 18.14 63.89
37.13 80.202 36.308 75.111 34.28 75.11
14.043 68.482 15.22 55.423 10.42 75.43
0
```

Sample Output

```
Case 1:
Hole 1: 1, 2

Case 2:
Hole 1: 3, 5
Hole 2: 2, 6
Hole 3: 1, 4
```

Problem B: Flip It!

Assume you have a set of cards laid out in an n by m grid. The cards are numbered and some are face up and others are face down. We can collapse the grid into a single pile by using a series of flips, each of which is one of the four following types:

Top Flip : Here the cards in the top row are flipped over onto the corresponding cards on the row beneath them. Note that if a card is face up in the top row, it becomes face down after the flip, and vice versa. If the top row contains one or more piles of cards, each entire pile is flipped over like a stack of pancakes as it is moved to the lower row.

Bottom Flip : Same as the Top Flip, but now the bottom row is flipped onto the next-to-bottom row.

Left Flip : Flip the cards in the left-most column onto the next-to-leftmost column.

Right Flip : Flip the cards in the rightmost column onto the next-to-rightmost column.

After a series of $n + m - 2$ flips, the cards will be in a single pile, some cards face up and some face down. Your job is to determine the order of the face up cards in this final pile.

Input

Each test case will start with a line containing two positive integers n m indicating the number of rows and columns in the grid. After this will come n rows of m integers indicating each card's number and its orientation. (The first row is the top row and the first value in each row is the leftmost card.) If a value is a positive integer k that means card k is at the location face up; if a value is a negative integer -k that means card k is at the location face down. (k will never be zero.) After these n rows there will be one more line containing $n + m - 2$ characters indicating the order of flips to apply to the grid. Each character will be either `T, B, L` or `R` corresponding to a top, bottom, left or right flip. All flip sequences will be legal, i.e., you won't be asked to do more than $n - 1$ top and bottom flips or $m - 1$ left and right flips. The maximum value for n and m is 20. A line containing two zeros will terminate input.

Output

For each test case, output the case number followed by a list of the numbers of all of the face up cards in the final deck, starting from the bottom of the deck. Follow the format used in the examples.

Sample Input

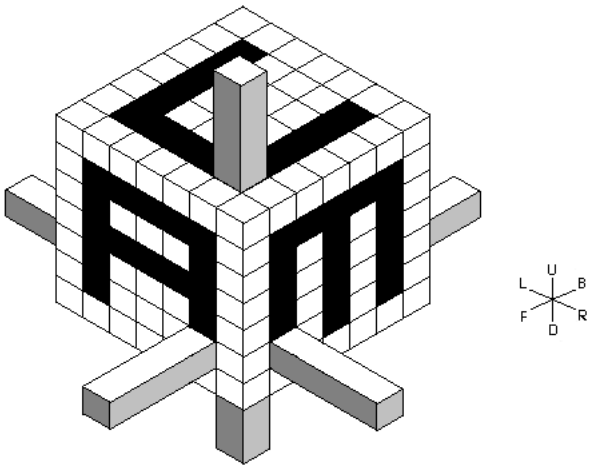
```
2 3
4 -17 -8
6 23 -5
LRB
1 1
-3

1 1
3

0 0
```

Sample Output

```
Case 1: 8 6
Case 2:
Case 3: 3
```

Problem C: Maze

We are all familiar with conventional mazes laid out on a 2-D grid. A 3-D maze can be constructed as follows: Consider a hollowed out cube aligned along the x, y and z axes with one corner at (0,0,0) and the opposite corner at $(n-1, n-1, n-1)$. On each face of the cube is a 2-D maze made by removing a subset of $1 \times 1 \times 1$ cubes from the face (no edge cubes are removed). The object of the maze is to move a marker located inside the cube from an initial location of (1,1,1) to the final destination of ($n-2$,$n-2$,$n-2$). However, attached to this marker are 6 rods, each protruding through one face of the cube. The movement of these rods is constrained by the 2-D mazes on the faces. The picture below gives an example of a $7 \times 7 \times 7$ maze. Note that this maze is not physically realizable since some faces (e.g., the front face containing the letter "A") have cubes that are disconnected from the edges of the face. Such mazes are allowed in this problem.



The black regions indicate open spaces where the rods can move. The figure to the right specifies the possible directions that the rods can move (Forward, Back, Left, Right, Up, Down) and also defines the labels for the six sides of the cube. In the maze above, the rods are shown in their initial position centered at (1,1,1). From here they can not move Forward, Backward, Right, Left or Down, but they can move Up (assuming there are open spaces for the two back rods to move to).

To specify a cube, a description of each face must be given. For this problem, the order and orientations of each face are given in the diagram below.

U	U	U	U	B	F
L F R	F R B	R B L	B L F	L U R	L D R
D	D	D	D	F	B

The first square represents the Forward face oriented so that the shared edge with the Up face is on top and the shared edge with the Right face is on the right, the second square represents the Right face oriented with the shared edge with the Up face on top and the shared edge with the Back face on the right, and so on. Your job is to solve such mazes in the minimum number of moves.

Input

Each test case will start with a single line containing the value of n, where n lies between 4 and 30, inclusive. Next will come descriptions of each face in the order and orientation shown above. The description of each face will consist of n lines each containing one string of length n. The characters in the string will be either a blank or `'X'`, indicating either an empty or full square, respectively. The last test case will be followed by a line containing 0.

Output

Output will consist of one line for each test case. Each line will contain a description of a minimum-move solution that moves the marker from cell (1,1,1) to cell $(n-2, n-2, n-2)$. Moves are either `F, B, L, R, U` or `D`. In case of a tie, choose the sequence of moves which is lexicographically first, where we consider `F` < `B` < `L` < `R` < `U` < `D`. All mazes will have solutions.

Sample Input

```
7
XXXXXXX
X     X
X XXX X
X     X
X XXX X
X XXX X
XXXXXXX
XXXXXXX
X     X
X X X X
X X X X
X X X X
X X X X
XXXXXXX
XXXXXXX
X     X
X     X
X     X
X     X
X     X
XXXXXXX
XXXXXXX
X     X
X X X X
X X X X
X X X X
X X X X
XXXXXXX
XXXXXXX
X     X
X XXX X
X X X X
X XXX X
X     X
XXXXXXX
0
```

(continued in next column)

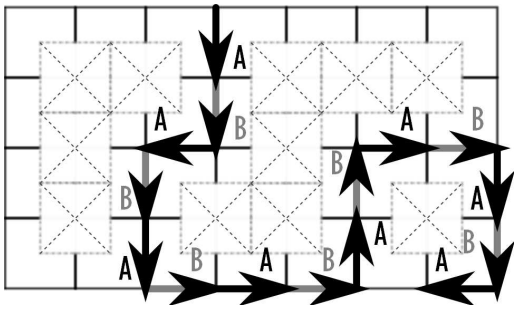
Sample Output

```
UULLLLUUBBBB
```

Problem D: Racing Car Trail

Have you ever read any description of some encryption algorithm? These descriptions almost always include messages being sent between *Alice* and *Bob*. We (the people organizing the 2011 Central Europe Regional Contest) think that those descriptions are too impersonal — considering these two people are probably the most famous cryptographers in the whole world, we know so little about them. They deserve more attention, don't you think? We can learn about their hobbies, for instance.

In their free time, Alice and Bob like to play a game inspired by *Tron*. In this game, you race a car through a square grid and you need to avoid hitting obstacles placed in the grid. Furthermore, the car leaves a permanent trail, which you also need to avoid. The car only moves in the four cardinal directions (east, west, north, or south). In their version of the game, Alice and Bob alternate in controling the car—Alice starts, moves the car from its initial position to one of the adjacent positions in the grid, then Bob takes over and moves the same car to another adjacent position, etc.



The player who crashes the car (i.e., moves it to a position occupied by an obstacle, or to one of the previously visited positions) loses. Both Alice and Bob are incredibly skilled players and never make mistakes; in particular, they only crash if there is no possible move from their current position that would avoid it. Given the map of the obstacles, your task is to determine which player wins from which initial position.

Input Specification

The input contains descriptions of several game fields. The first line of each description contains two integers N and E ($1 \leq N, E \leq 100$) — the size of the grid in the north-south and in the east-west directions. The following N lines describe the map. Each of the lines contains a string of E characters, where the j-th character on the i-th line determines the state of the position with coordinates (j, i). The possible characters are "`.`" (a dot) if the position is empty and the uppercase letter "`X`" if there is an obstacle. All positions not covered by the map (i.e., with coordinates (j, i) such that $i \leq 0$ or $j \leq 0$ or $i > N$ or $j > E$) are forbidden and not used in the game, they work as if there were obstacles.

The last game field is followed by a line containing two zeros.

Output Specification

For each game field, output N lines of strings of length E, showing whether Alice or Bob wins when the game starts from the given location. The j-th character on the i-th line should be "`A`" if Alice wins when starting from the position (j, i), "`B`" if Bob wins, or "`X`" if the position contains an obstacle.

After each output, print one empty line.

Sample Input

```
1 1
.
3 3
...
.X.
...
1 4
....
3 3
X.X
...
X.X
5 8
........
.XX.XXX.
.X..X...
.X.XX.X.
........
0 0
```

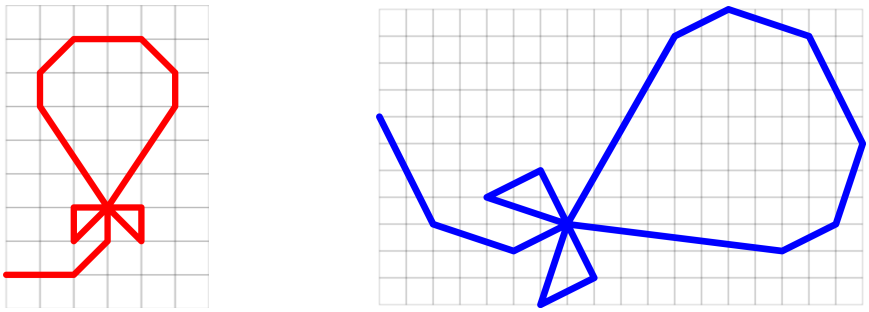
Output for Sample Input

```
B

AAA
AXA
AAA

AAAA

XBX
BAB
XBX

BABABABA
AXXBXXXB
BXBAXABA
AXAXXBXB
BABABABA
```

Problem E: Unchanged Picture

Steganography is a special way to protect messages — instead of encryption, the message is somehow *hidden*. Historically, it was quite a popular technique, but nowadays it is superseded by ciphers, especially those based on keys. However, sometimes it may still be in use. One of the digital steganographic techniques is to hide small pieces of information into a digital image. The image modification is so small that it cannot be spotted by a human eye, but the information (such as a text message) is there and readable by computers. Your task is to compare two images and find such (possibly small) differences.

In this problem, we will focus on vector pictures. Your program is given two pictures and it should decide whether they contain the same image. Geometrically speaking, decide whether the two picture are *similar*, that means whether they can be transformed into each other using translation, rotation, and uniform scaling (but not mirroring). An example of two similar pictures follows.



Input Specification

The input file consists of several test cases, each of them containing two vector pictures. Each picture is described by a sequence of instructions for a plotter device, one instruction per row. Every instruction begins with an uppercase letter followed by one space character and two integer coordinates separated by another space. The letter is either "`L`" (draw a line) or "`M`" (move without drawing). The coordinates specify the place to which the line is to be drawn or the current position moved. Coordinates are always given relatively to the end position of the previous instruction. The first instruction is relative to some (unspecified) starting point.

The last instruction of each picture is followed by a row containing the letter "`E`" (end) and an empty line. The last test case will be followed by a row containing the letter "`Q`" (quit).

The number of instructions for any picture is between 0 and 1000, inclusive. No instruction has both coordinates equal to zero. The absolute value of all (relative) coordinates is at most 1000.

Output Specification

For each test case, print one line containing either the word "`YES`" (two pictures are similar) or "`NO`" (pictures are not similar).

Sample Input

```
L 3 1
L 3 1
M 3 0
M 0 1
L -3 -1
E

L 1 0
L -1 0
E

L 1 0
L 0 1
E

L 1 0
L -1 -1
E

L 2 0
L 1 1
L 0 1
L 2 3
L 0 1
L -1 1
L -2 0
L -1 -1
L 0 -1
L 2 -3
M 1 0
L -2 0
M 2 0
L 0 -1
L -1 1
L -1 -1
L 0 1
E

L 2 -4
L 3 -1
L 2 1
L 1 -2
L -2 -1
L 1 3
L -3 1
L 2 1
L 1 -2
L 8 -1
L 2 1
L 1 3
L -2 4
L -3 1
L -2 -1
L -4 -7
E

Q
```

Output for Sample Input

```
YES
NO
YES
```

Problem F: Analog Clock Display

Our old clock chimed four o'clock, and with that last sound they dissolved into a pile of sticks. Since the craftsmanship necessary to fix them was forgotten and lost long time ago, we decided to replace them by a computer program.

An easy task, you say? There is one more little thing to mention: Recent studies of extraterrestrial aliens showed that they live in a digital world and therefore they are unable to read the traditional clock face with two moving hands. It is simply something beyond their technical capabilities. Therefore, the Security Board decided that our new clock must use such a traditional "analog" display to protect the time information against non-humans.

Input Specification

The input contains several instances, each of them consisting of one row containing two integers H ($0 \leq H \leq 23$) and M ($0 \leq M \leq 59$) separated by a colon. M is always given with two digits, H has one or two digits, i.e., no leading zero unless $H = 0$.

The last test case is followed by the word "`END`".

Output Specification

For each input instance H:M, draw an "ASCII art" clock face depicting the time of H hours and M minutes according to the following specification.

The face frame is a square, with 51 characters per each side. These characters are uppercase letters "`X`", with the exception of four corners and every tenth character, which is always "`@`". The numbers 3, 6, 9, and 12 are centered at their appropriate sides, with exactly one space between them and the frame. There is one space between digits 1 and 2. The center of the face always contains the asterisk symbol: "`*`". All visually "empty" characters are simple spaces.

Two hands are the only elements of the clock face that depend on the time. In the following description of the placement of the hands, we assume that each character (both space or occupied) is a 1×1 square and the start $(0, 0)$ of the coordinate system is placed in the *center* of the square containing the asterisk symbol, with the first axis pointing to the right, and the second axis upwards.

The hour hand is drawn as a line segment of length 15 starting at $(0, 0)$. The hand points upwards at 12 o'clock and moves uniformly by the same angle each minute. Similarly, the minute hand is drawn as a line segment of length 21 starting at $(0, 0)$. That hand points upwards every hour and also moves uniformly by the same angle every minute in the clockwise direction (what a surprise). The minute hand is considered to be *above* the hour hand, i.e., the characters representing the hour hand may be hidden by parts of the minute hand.

A *line* whose angle from the vertical direction is D degrees should be drawn as follows. If $0 \leq D \leq 45$, one character is printed for each row i at (integer) coordinates (n_i, i) as close as possible to the point (x_i, i) that lies exactly on the geometric line (x_i is a *real* number). If $45 \leq D \leq 90$, there is one character printed for each column i at the (integer) coordinates (i, k_i), the closest possible square to the (real) point (i, y_i) on the line.

The character displayed to draw the line at some position (i, j) depends on the two "neighboring" characters of the line. The character is

- minus symbol "`-`" if there are also characters at both positions $(i-1, j)$ and $(i+1, j)$,
- pipe symbol "`|`" if there are also characters at both positions $(i, j-1)$ and $(i, j+1)$,
- backslash symbol "`\`" if there are characters at positions $(i-1, j+1)$ and $(i+1, j-1)$,
- slash symbol "`/`" if there are also characters at positions $(i-1, j-1)$ and $(i+1, j+1)$,
- lowercase letter "`o`" otherwise.

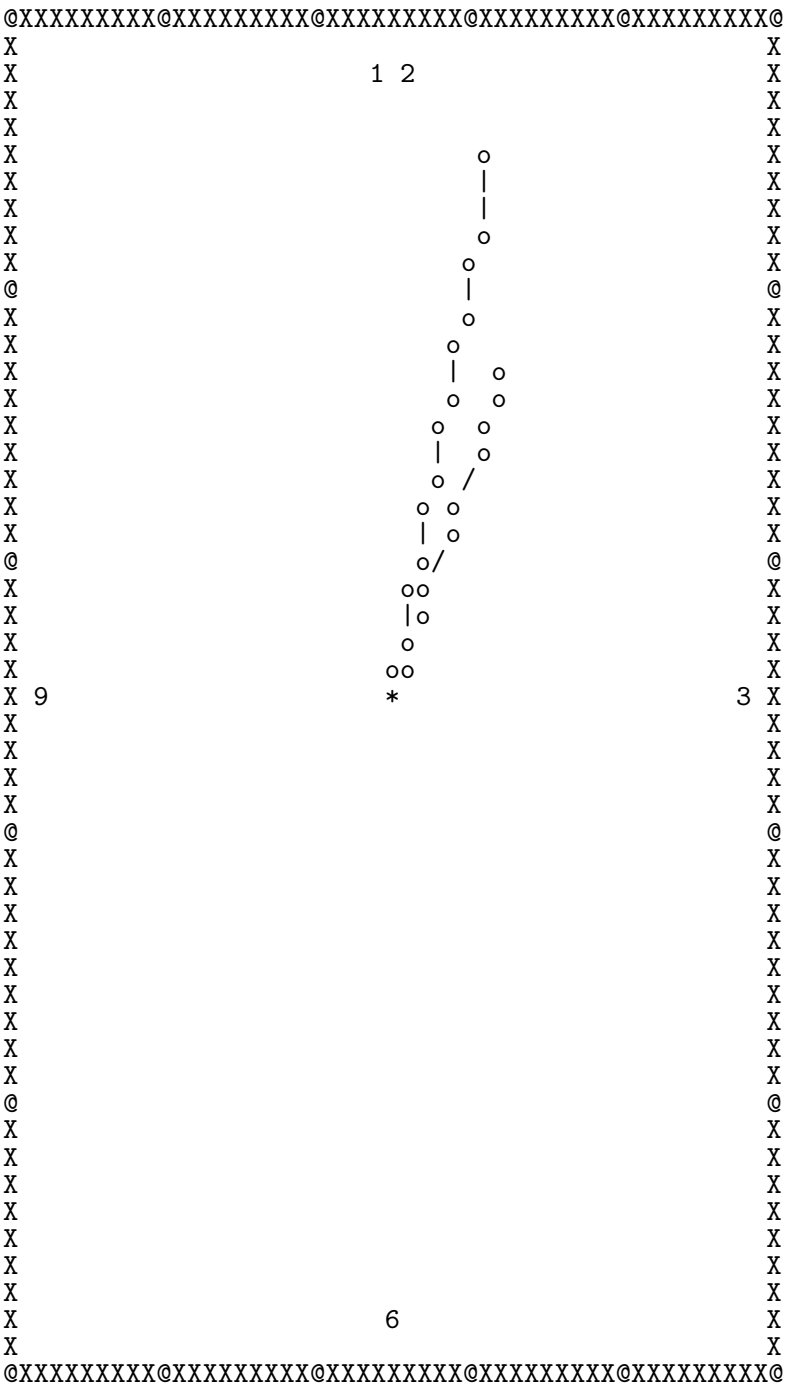
A line *segment* of length S starting at $(0, 0)$ is drawn by displaying characters in the same way as drawing the corresponding line, but we only print such characters whose distance between the *center* of the square and the origin $(0, 0)$ is *at most* S, $0 < |(0, 0), (i, j)| \leq S$, and only in one direction of the line.

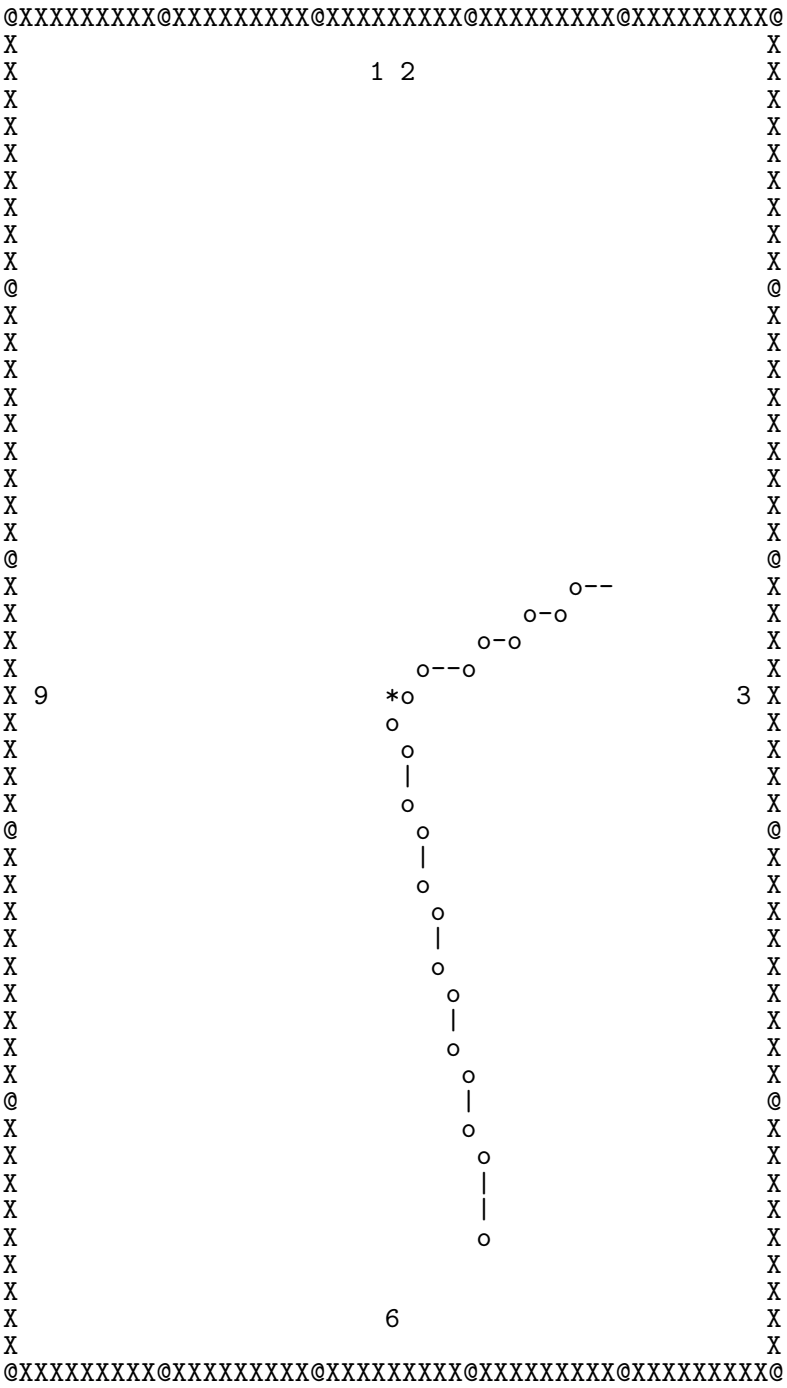
Please see the sample output to resolve any ambiguities in the above description. Print one empty line after each clock face.

Sample Input

```
1:03
14:27
END
```

Output for Sample Input




1 2
9
3
6

Problem G

Locksmith

Description

Mooks A, C, and M have been charged with creating locks for the Riddler's lair. They need to design locks comprised of flat interlocking pieces that can be unlocked only by separating these pieces while sliding them around on the surface of the doors. Unfortunately, while the mooks have come up with quite a few designs, they are not very good at determining which designs can actually be unlocked.

The pieces of the lock are axis-aligned polygons (that is, the sides are all either horizontal or vertical) in the plane. The lock can be manipulated by sliding any one polygon at a time in any direction, so long as doing so does not cause the polygon to overlap with any other polygon. Rotating a polygon is not allowed. A polygon is considered separable from the rest of the lock if there exists a sequence of moves (possibly involving several polygons) leading to a configuration such that it is possible to draw a straight line between the polygon and the remainder of the lock. Given a proposed lock configuration, your job is to determine the number of separable polygons.

Input

The input file will contain multiple test cases. Each test case begins with a line with a single integer N denoting the number of pieces in the lock. This line is followed by N lines of space-separated integers each with the following format:

$c \quad x_1 \quad y_1 \quad \cdots \quad x_c \quad y_c$

Here, c denotes the number of vertices in the piece, and x_i and y_i give the locations of the vertices in clockwise order. The sides of each piece are all either horizontal or vertical, and they alternate between the two. The pieces are guaranteed to have no overlapping area. Each test case has either 2 or 3 pieces, and for each test case, these pieces have a total of at most 30 vertices. All coordinates given are integers between 0 and 1000 inclusive. Input will be terminated by a case with 0 pieces, which should not be processed.

Output

For each input test case, print the number of separable pieces in the lock.

Sample Input	Sample Output
2 12 0 0 0 6 9 6 9 0 6 0 6 1 8 1 8 5 1 5 1 1 3 1 3 0 4 2 2 2 4 7 4 7 2 2 12 0 0 0 6 9 6 9 0 6 0 6 1 8 1 8 5 1 5 1 1 3 1 3 0 4 4 2 4 4 7 4 7 2 0	0 2

H
Peer Review

For scientific conferences, scientists submit papers presenting their ideas, and then review each other's papers to make sure only good papers are presented at the conference. Each paper must be reviewed by several scientists, and scientists must not review papers written by people they collaborate with (including themselves), or review the same paper more than once.

You have been asked to write a program to check if your favorite conference is doing things right. Whether a paper is being reviewed too much, too little, or by the wrong people - the organizers must know before it is too late!

Input

The first line in each test case has two integers, K ($1 \leq K \leq 5$) and N ($1 \leq N \leq 1000$). K is the number of reviews that each paper will receive, while N is the number of papers to be reviewed. The conference only accepts papers with a single author, and authors can only present a single paper at the conference.

Each of the next N lines describes an author and includes the name of the institution to which the author belongs, followed by the list of the K papers he or she has been requested to review. It is assumed that researchers from the same institution collaborate with each other, whereas researchers from different institutions don't. All institution names are shorter than 10 characters, and contain only upper or lowercase letters and no whitespace. Since we have as many papers as authors, papers are identified by their author's index; paper 1 was written by the first author in the list, and paper N was written by the last author.

The end of the test cases is marked with a line containing $K = 0$ and $N = 0$. You should generate no output for this line.

Output

For each test case, your program should output `NO PROBLEMS FOUND` (if all rules are being followed) or P `PROBLEMS FOUND`, where P is the number of rule violations found (counting at most 1 violation per paper). If there is exactly one rule violation overall, your program should output `1 PROBLEM FOUND`.

Sample Input

```
2 3
UCM 2 3
UAM 1 3
UPM 1 2
2 3
UCM 2 3
UAM 1 2
UPM 2 2
0 0
```

Sample Output

```
NO PROBLEMS FOUND
3 PROBLEMS FOUND
```

I
Regular Convex Polygon

A regular convex polygon is a polygon where each side has the same length, and all interior angles are equal and less than 180 degrees. A square, for example, is a regular convex polygon. You are given three points which are vertices of a regular convex polygon R; can you determine the minimum number of vertices that R must have?

Input

Each test case consists of three lines. Line i consists of two floating point values x_i and y_i ($-10^4 \leq x_1, y_1 \leq 10^4$) where (x_i, y_i) are the coordinates of a vertex of R. The coordinates are given with a precision of 10^{-6}, i.e., they differ from the exact coordinates by at most 10^{-6}. You may assume that for each test case the Euclidean distance between any two given points is at least 1, and R has at most 1000 vertices. The input will finish with a line containing the word `END`.

Output

For each test case, print one line with the minimum number of vertices that R must have.

Sample Input

```
-1385.736326 -146.954822
430.000292 -2041.361203
1162.736034 478.316025
0.000000 4147.000000
-4147.000000 0.000000
0.000000 -4147.000000
END
```

Sample Output

```
3
4
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
University of Žilina
Matej Bel University in Banská Bystrica

Czech Technical University in Prague

Technical University of Ostrava
Pavol Jozef Šafárik University in Košice
Masaryk University
University of West Bohemia

CTU Open Contest 2011

Problem J: Text Encryption

To keep privacy of messages and prevent the aliens from reading them, we may use various encryption algorithms. These algorithms encode a message into the so-called *ciphertext* that is difficult (or impossible) to decode for anyone else than the intended recipient. *Transposition ciphers* are a type of encryption that do not change the letters of the message but only change their order ("shuffle" the letters). Of course, the shuffling must be reversible to allow later decryption.

In this problem, we will consider a simple transposition cipher which shuffles the letters in such a way that the *decryption* algorithm always takes every n-th letter. More specifically: when decrypting, the first letter of the ciphertext is taken first, then the next $n-1$ letters are (repeatedly) skipped and the next letter taken, and so on until we reach the end of the ciphertext. After that, we repeat the procedure starting with the second letter of the ciphertext, and so on until all letters are used.

Your task is to implement the encryption algorithm for this cipher. For a given message, produce the encrypted text (ciphertext). To make the cipher a little bit stronger, you should convert all letters to uppercase and leave out all spaces between words.

Input Specification

The input contains several messages. Each message is described by two lines. The first line contains one integer number N ($1 \leq N \leq 1000$). The second line contains the message. The message will be at most 10 000 characters long, it will only contain letters and spaces, and there will be at least one letter in each message.

The last message is followed by a line containing zero.

Output Specification

For each message, output the ciphertext that, after using the described decryption algorithm, will result in the original message (with all spaces removed and all letters in uppercase).

Sample Input

```
2
CTU Open Programming Contest
7
This is a secret message that noone should ever see Lets encrypt it
15
text too short
0
```

Output for Sample Input

```
CMTMUIONPGECNOPNRTOEGSRTA
TESNUECHCAOLERIRGODLYSEENEEPITTEVTTSMHSESIAEAHRETSSTOSN
TEXTTOOSHORT
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
University of Žilina
Matej Bel University in Banská Bystrica

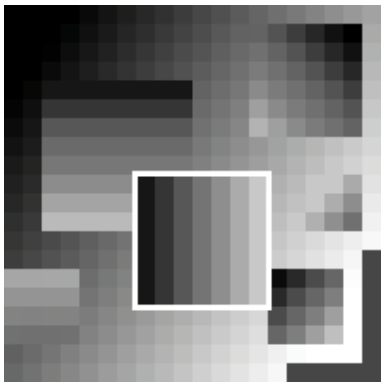
Czech Technical University in Prague

Technical University of Ostrava
Pavol Jozef Šafárik University in Košice
Masaryk University
University of West Bohemia

CTU Open Contest 2011

Problem K: Mine the Gradient

If we need to be prepared for the possible danger of an alien invasion, the first thing to do is to find out where can they come from. One way to determine if a planet is inhabited by intelligent creatures is to study high resolution pictures of planets trying to find typical characteristics of effects of an intelligent life. There are so many habitable planets that a computer program must be written for this task. One distinctive feature of a colonized planets is the presence of surface mines. An alien mine is a square structure with a depth uniformly decreasing from one of the sides of the square towards the opposite.



Your task is to find the largest mine on a provided bitmap image of a planet. The picture is a rectangular grid of numbers between 0 and 65535, representing the shades of grey. Mines will appear as squares of either the same shade or with shade levels gradually and uniformly changing from one side to another. See the above figure for an example. Your program will consider only square mines in some special orientations.

An axis-parallel square is the set of pixels (i, j) such that $c_1 \leq i \leq c_2$ and $r_1 \leq j \leq r_2$ for some c_1, c_2, r_1, r_2, $c_2 - c_1 = r_2 - r_1$.

A vertically (horizontally) oriented mine is an axis-parallel square for which there exist integers S and K such that the shade of every pixel (i, j) of the square is equal to $S + iK$ ($S + jK$ for horizontally oriented mines).

A diagonally oriented mine is an axis-parallel square for which there exist integers $Q \in \{1, -1\}$, S and K such that the shade of every pixel (i, j) of the square is equal to $S + (i + Qj)K$.

Input Specification

The input contains several descriptions of pictures. The first line of each description contains two numbers N and M ($1 \leq N, M \leq 2000$), the height and width of the picture. The following N lines contain M space-separated integers each — there is at least one space character between numbers but there may be more spaces and also additional spaces at the beginning or end of the line are allowed. The j-th number in the i-th row $A_{i,j}$ ($0 \leq A_{i,j} \leq 65535$) describes the shade of grey of the pixel in the ith row and jth column of the picture bitmap.

The last description is followed by a line containing two zeros.

Output Specification

For each picture, output the area (number of pixels inside) of the largest horizontal, vertical, or diagonal mine.

Sample Input

```
4 4
10  1 13 20
18  9 11 13
 5  7  9  6
 6  5  7  7
3 3
10  1   13
18  9   11
 5 1000  9
4 4
10 12 15 20
 5  9 13 10
 5  9 13  6
 5  9 13  7
0 0
```

Output for Sample Input

```
4
1
9
```

Czech ACM Student Chapter

Charles University in Prague
Slovak University of Technology
University of Žilina
Matej Bel University in Banská Bystrica

Czech Technical University in Prague

Technical University of Ostrava
Pavol Jozef Šafárik University in Košice
Masaryk University
University of West Bohemia

CTU Open Contest 2011

Problem L: Invasion

Alien invasion began, and scary man-eating aliens are establishing their bases all over the country. You are only safe on the places that are sufficiently far away from all current alien bases. You need to quickly write a program to help you determine where to move.

Input Specification

The input contains several instances, each of them consisting of several lines. The first line of each instance contains integers N ($1 \leq N \leq 10\,000$), M ($0 \leq M \leq 100\,000$), A ($0 \leq A \leq N$) and K ($1 \leq K \leq 100$) separated by spaces, giving the number of towns in the country, the number of roads between them, the number of bases that the aliens are going to build, and the minimum safe distance from alien bases, respectively. The towns are assigned numbers $1, \ldots, N$.

The following M lines describe the roads; each of them contains integers T_1, T_2 ($1 \leq T_1 < T_2 \leq N$) and D ($1 \leq D \leq 100$) separated by spaces, where D is the length of the road between towns T_1 and T_2. There is at most one direct road between any two towns. All roads can be used in both directions.

The following A lines describe the positions of alien bases; the i-th of them contains the number B_i ($1 \leq B_i \leq N$) of the town where the aliens build their i-th base.

Each instance is followed by one empty line. The empty line after the last instance is followed by a line containing four zeros.

Output Specification

The output for each input instance consists of A lines. On i-th line, write the number of towns that are safe after the aliens build their i-th base. The town is safe if its distance from *any* of the towns $B_1, B_2, \ldots, B_i$ is at least K.

Print one empty line after each instance.

Sample Input

```
7 6 3 3
1 2 1
1 3 1
2 5 1
3 6 1
1 4 1
4 7 2
2
1
4

1 0 1 1
1

0 0 0 0
```

Output for Sample Input

```
2
1
0

0
```