PROBLEM 1 – WACMIAN NUMBERS

In the supposedly uninhabited Wacmahara Desert, a tribe of unusual people has been discovered. The Wacmians have only 2 fingers and a thumb on each hand, and have invented their own numbering system. The digits they use and the symbols they use for digits are quite unusual, but anthropologists have been able to represent them as follows:

`%`	represents	`0`
`)`	represents	`1`
`~`	represents	`2`
`@`	represents	`3`
`?`	represents	`4`
`\`	represents	`5`
`$`	represents	`-1` (yes, they even have a negative digit)

As you may expect, their system is base 6 where each place value is 6 times the value to its right, as in the following examples:

`)@%`	is	$1*6^2+3*6+0 = 36+18+0 = 54$
`?$~~`	is	$4*6^3+(-1)*6^2+2*6+2 = 864-36+12+2 = 842$
`$~~`	is	$(-1)*6^2+2*6+2 = -36+12+2 = -22$

Your task is to take Wacmian numbers and represent them as standard base 10 numbers.

Input Format

Input consists of Wacmian numbers, one per line. Each number consists of a sequence of 1 to 10 Wacmian digits. A single '#' on a line by itself indicates the end of input.

Sample Input:

```
)@%
?$~~
$~~
%
#
```

Output Format

Output will be the corresponding decimal numbers, one per line.

Sample Output:

```
54
842
-22
0
```

PROBLEM 2 – CD TITLES

A keen photographer, I found my hard disk quickly filling up with snapshots and videos. I decided to offload a lot of the files to CD. Each CD is in its own plastic case, and the contents are clearly described on the front of the case. As the number of CDs increased, I built myself a shelf on which the CDs stand vertically.

I need your help! I have written a title for each CD into a text file, but I need the titles to appear vertically so that I can put them in the spine of the CD cases and be able to read them easily from the shelf. I want you to write a program that will output my titles vertically. I need lines between each title so that, when I print them, I can easily cut along the lines.

I have worked out that I can fit 36 characters into the available space, so all output titles must be 36 characters long, padded with spaces at the end where necessary. If I accidentally make a title too long, only output the first 36 characters.

INPUT FORMAT

Input will be at most 50 titles, one to a line. Each title consists of 1 to 100 arbitrary characters. A single '#' on a line by itself indicates the end of input.

SAMPLE INPUT:

```
012345678901234567890123456789012345
David and Jane’s wedding, March 2002, Alexandria
Bahamas Holiday August 2001
#
```

OUTPUT FORMAT

Output will be the same titles presented vertically, where the left to right order will be the same as the order of the input. There will be a column of bar ('|') characters at each end, and separating each title, and a row of minus ('-') characters (1 per column) at the beginning and end.

SAMPLE OUTPUT:

```
-------
|0|D|B|
|1|a|a|
|2|v|h|
|3|i|a|
|4|d|m|
|5| |a|
|6|a|s|
|7|n| |
|8|d|H|
|9| |o|
|0|J|l|
|1|a|i|
|2|n|d|
|3|e|a|
|4|’|y|
|5|s| |
|6| |A|
|7|w|u|
|8|e|g|
|9|d|u|
|0|d|s|
|1|i|t|
|2|n| |
|3|g|2|
|4|,|0|
|5| |0|
|6|M|1|
|7|a| |
|8|r| |
|9|c| |
|0|h| |
|1| | |
|2|2| |
|3|0| |
|4|0| |
|5|2| |
-------
```

PROBLEM 3 – CAESAR CIPHER

Julia has decided to encrypt her notes so that nobody else could understand them. The code is based on the so–called Caesar Cipher where each letter is shifted a certain number of places left or right through the alphabet. In this context, the alphabet is treated as being circular so that the first letter follows after the last letter, and the last letter precedes the first letter.

Julia applies these ideas separately to uppercase letters, lower case letters, and digits. For example, with a shift of 1, 'A' becomes 'B', 'Z' becomes 'A', 'a' becomes 'b', 'z' becomes 'a', '0' becomes '1', '9' becomes '0'. Spaces, punctuation, and any other symbols are not affected in this scheme.

Your task is to help Julia encrypt her notes.

INPUT FORMAT

Each line of input begins with a number representing the shift. The number will be in the range -1,000,000,000 to 1,000,000,000. The number is followed by a colon (':'). The rest of the line consists of a string of 1 to 200 arbitrary characters and represents a fragment of the text to be encrypted. A single '#' on a line by itself indicates the end of input.

SAMPLE INPUT:

```
0:Clear text!
1:David and Jane's wedding, March 2002, Alexandria
-1:Bahamas Holiday August 2001
53:ACMZ, acmz, 0379!
26000000:ACMZ, acmz, 0379!
26000001:ACMZ, acmz, 0379!
#
```

OUTPUT FORMAT

Output will be the corresponding encrypted text fragments, one per line.

SAMPLE OUTPUT:

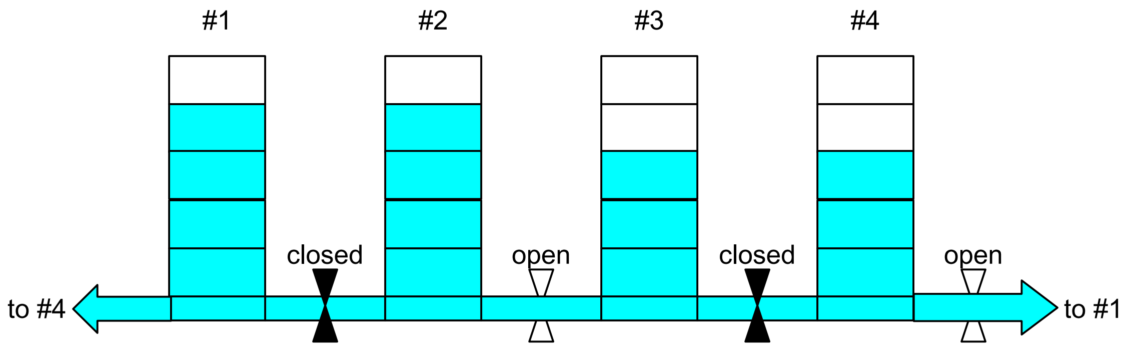
```
Clear text!
Ebwje boe Kbof’t xfeejoh, Nbsdi 3113, Bmfyboesjb
Azgzlzr Gnkhczx Ztftrs 1990
BDNA, bdna, 3602!
ACMZ, acmz, 0379!
BDNA, bdna, 1480!
```

PROBLEM 4 – WATER TANKS

There are *n* identical large cylindrical tanks for storing water. The tanks are arranged in a circle on level ground. Each tank is connected with its two neighbours by means of pipes situated at its base. There is a valve between each adjacent pair of tanks (tank *n* is next to tank 1). All valves are initially closed. All the outlets and the pipes are at the same level and are always full of water, even if all the tanks are deemed to be “empty”.

The volume in any tank is measured by the height of the surface of the water above the level of the top of the outlets. If all valves (or all valves but one) are opened so that water can flow between the tanks, then the levels will eventually equalise. Conversely, if all tanks are initially at the same level, no valves need be opened to equalise the levels. Thus it may be necessary to only open some of the valves to achieve this result.

For example, consider *n*=4 tanks each 5 metres high. Assume that the water level in these tanks is at 4, 4, 3, and 3 meters respectively. Their water level will equalise if we open the valves between tanks #2 and #3 and between #4 and #1, as suggested by the following diagram. Thus for this set we need to open only two valves:



Given a set of initial heights, determine the minimum number of valves to open so that the final water levels in all tanks is equal.

INPUT FORMAT [1]

The input will consist of one or more scenarios, each scenario consisting of two lines.

The first line contains a descriptive title, which is a string of letters or spaces no more than 200 characters long, containing at least 1 letter.

The second line starts with the number of basins n ($3 \leq n \leq 200$), a space, and then n integers in the range 0 to 99, separated by single spaces, representing the water levels in the tanks.

The scenarios sequence is terminated by a single '#' character on a line by itself.

SAMPLE INPUT:

```
High four dude basins
4 4 4 3 3
The Australasian eight
8 2 1 1 2 2 1 1 6
#
```

OUTPUT FORMAT

Output one line for each input scenario. The line consists of the first letter of each word in the descriptive title in upper case, followed by a colon (':'), a space, and then the minimum number of valves that need to be open to achieve equal heights in all tanks.

SAMPLE OUTPUT:

```
HFDB: 2
TAE: 5
```

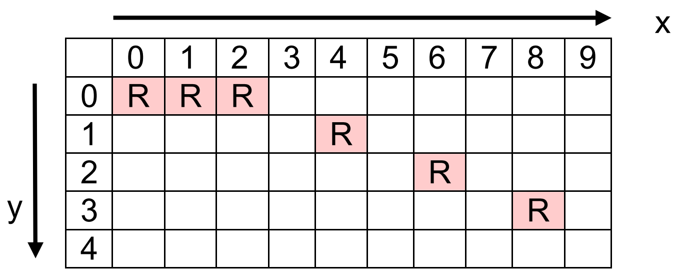
[1] Caveat: The input data for this program may contain lines up to 600 characters long.

PROBLEM 5 – SPOTS

Painting tiles is hard work. John has convinced his two sons Raul and George to do a painting job, by agreeing to pay one dollar per tile painted by each of them. Raul will paint red tiles and George green tiles at the places indicated by their father. However, they are still not convinced: How are they going to divide up the area to be painted, what happens if both want to paint the same tile at the same time, what are the rules, and, last, but not least, how much will they receive in the end?

To avoid arguments and to have some fun, John would like to show them a simulation of the problem before the hard work begins. The simulation will start with each son at opposite ends of a rectangular grid of width *N* and height *M* (*N*,*M* ≥ 2), Raul at (0,0) and George at (*N*-1, *M*-1). The two sons will begin by painting a spot in their respective colours on their starting tiles. Next, Raul and George are each given a series of individual instructions for moving to the next tile to paint. Each move can be repeated one or more times and is defined by a pair of increments along the horizontal x–axis and vertical y–axis, in this order; these increments can be positive, negative, or zero.

For example, assume that Raul is on his initial tile (0,0) on a grid with *N*=10, *M*=5, and receives the instructions to hop 2 times in the direction of (1,0) and then 3 times in the direction of (2,1). He will begin by painting a red spot on the tile (0,0). Then, according to these instructions, he will successively land on and paint red spots on the following tiles: (1,0), (2,0), (4,1), (6,2), (8,3). The following diagram uses the letter 'R' to mark the tiles spotted in red by Raul according to his instructions:



	0	1	2	3	4	5	6	7	8	9
0	R	R	R							
1					R					
2							R			
3									R	
4										

At each simulation step the two sons move in lock-step, each hopping according to his own instructions. To avoid conflicts, each time Raul and George are about to hop to the next tile the simulation must check whether they would land on the same tile. If so, the simulation ends without them moving, and the landing tile remains painted in its previous colour, if any. Otherwise, the simulation will end as soon as one of the sons ends his instructions.

Each tile can contain only one colour spot, either red or green. Since it is possible that Raul and George land on the same tile at different times, the simulation should only count the tile towards the son landing there last.

To avoid Raul and George falling off the edge of the tile grid, they are allowed to wrap around, in all directions. For example, for a grid of the same size as above, one hypothetical move from (8,3) in the direction (2,3) results in the tile (0,1).

Your task is to write a program that computes the results of such a simulation given the grid size and the instructions for each son.

INPUT FORMAT [1]

The input consists of one or more scenarios. Each scenario consists of 3 lines. The first line contains two numbers separated by a space, *N* and *M*, $2 \le N, M \le 1000$, respectively representing the width and the height of the grid. The second line contains the instructions for Raul and the third line the instructions for George. Each instruction line starts with a number *C*, in the range 1 to 100, followed by *C* groups of 3 numbers. In each group the first number is a repetition count in the range 1 to 1,000,000,000, the second number is an increment along the x–axis, and the third number an increment along the y–axis – both increments are in the range -1,000 to 1,000. All numbers are separated by single spaces. The end of the input is indicated by a "grid of size 0", i.e., a '0' on a line by itself.

SAMPLE INPUT:

```
10 5
2 2 1 0 3 2 1
3 4 -2 0 1 0 -9 2 1 0
10 10
2 1 15 5 1 0 0
2 1 0 0 2 -4 -4
10 7
2 1000 2 -1 1000 0 -1
2 1000 -1 0 1000 -1 0
10 10
5 1000 2 -1 1000 1 0 2 5 5 3 1 1 10 1 1
5 1000 -1 0 1000 0 1 3 -4 -4 2 -1 -1 10 -1 -1
0
```

OUTPUT FORMAT

Output one line for each input scenario. Each output line should show two numbers separated by a single space, representing the earnings of Raul and George, in this order.

SAMPLE OUTPUT:

```
5 6
2 1
30 10
18 18
```

[1] Caveat: The input data for this program may contain lines up to 2000 characters long.

PROBLEM 6 – MOBILE PHONES

ACMIA mobile phones have a shortcut mode for typing text messages using the numerical phone keypad. In this mode, the system uses a dictionary of known words. After a sequence of digits is entered the system checks for and displays all possible matches in the dictionary. The ACMIA phone keypad for the English alphabet is as follows:

1	2 abc	3 def
4 ghi	5 jkl	6 mno
7 pqrs	8 tuv	9 wxyz
	0 (space)	

Your task is to write a program that displays all possible matches for given digit sequences, using a given dictionary.

A digit sequence corresponds to a sequence of words, with zero digits (‘0’) indicating spaces. Leading and trailing zeros are ignored, and multiple consecutive embedded zeros are treated as a single zero. For each sequence of non–zero digits, display the matching word from the dictionary. When more than one match is available, display all matches in dictionary order between round parentheses and separated by bars (‘|’). If there is no matching word, display a sequence of asterisks (‘*’) of the same length. For example, with a dictionary consisting solely of the words ‘i’, ‘loud’, ‘love’, ‘programming’, the digit sequence

‘00405683000776472664640777700’

will be displayed as the text

‘i (loud|love) programming ****’

INPUT FORMAT

The input will consist of one or more scenarios, each scenario consisting of a dictionary of permitted words and a series of digit sequences to be interpreted as text messages.

The dictionary consists of 1 to 1,000 words, one word per line, in increasing dictionary order, with no duplicates. Each word consists of 1 to 30 lowercase letters. For any given non–zero digit sequence there will be no more than 10 matching words in the dictionary. The end of the dictionary is indicated by a line consisting of a single ‘#’.

The digit sequences to interpret as text messages follow the dictionary, one per line. Each message line consists of 1 to 100 digits, with at least 1 non–zero digit. The end of messages is indicated by a line consisting of a single ‘#’.

The end of input is indicated by an empty dictionary (a dictionary with zero words).

SAMPLE INPUT:

```
i
loud
love
programming
#
0040568300077647266464077770
#
a
game
go
golf
good
hand
hold
hole
home
in
me
of
to
#
20466304263063046 53
4608602046630466 3
#
#
```

OUTPUT FORMAT

For each scenario output a line consisting of the word 'SET' (all uppercase) followed by a space and then the scenario number, starting with 1. Following this output the list of interpreted text messages, one message per line.

SAMPLE OUTPUT:

```
SET 1
i (loud|love) programming ****
SET 2
a (good|home) (game|hand) (me|of) (golf|hold|hole)
(go|in) to a (good|home) (good|home)
```

PROBLEM 7 – COLOUR GRIDS

You are given a number of plastic tiles, all of the same size, and a target pattern. Your task is to assemble the tiles into a pile so that, looking from the top, it matches the target pattern. Each tile is composed of 16 squares of transparent or coloured plastic (opaque and identically coloured on both faces), arranged in a 4x4 grid. The target pattern is also a grid of colours, similar to a tile, but without transparencies. The tiles can be rotated clockwise through multiples of 90° and/or flipped about a horizontal axis. This leads to 8 possible transformations – rotated 0, 1, 2 or 3 times, or a flip followed by the same sequence of rotations. These transformations, in the above order, are numbered 0 to 7. The following diagrams provide an example, with colours denoted by upper case letters, and dots ('.') representing transparency:

Transform #0 (original)	Transform #1	Transform #2	Transform #3
R R G G	Y Y R R	B B Y Y	G G B B
R R . G	Y Y R R	B B Y Y	G . B B
Y Y B B	B B . G	G . R R	R R Y Y
Y Y B B	B B G G	G G R R	R R Y Y

Transform #4	Transform #5	Transform #6	Transform #7
Y Y B B	R R Y Y	G G R R	B B G G
Y Y B B	R R Y Y	G . R R	B B . G
R R . G	G . B B	B B Y Y	Y Y R R
R R G G	G G B B	B B Y Y	Y Y R R

Write a program that will read in a sequence of tiles and a target pattern and determine whether the target pattern can be made by some arrangement of all or some of the input tiles.

- If there are several ways of achieving the target, then choose the pile with fewest tiles.
- If there are still several ways of achieving the target, then at every point in the pile, in a top–down order, choose first the lowest numbered appropriate tile and, if still needed, the lowest numbered appropriate transformation.

Input Format

The input will consist of one or more of problems. The first line of each problem contains the title of the problem as a string of 1 to 30 characters other than space, followed by a space, and the number of tiles n ($1 \leq n \leq 10$). This is followed in turn by n+1 lines, the first n lines specifying the tiles (implicitly numbered 0 to n-1); and the last line specifying the target pattern. Each of these n+1 lines contains 4 blocks of 4 characters, either an upper case letter representing a colour, or a full stop (‘`.`’) representing transparency. Blocks are separated by single spaces and represent grid rows, in successive row order. Characters composing blocks represent grid cells, in successive column order. The end of input is signified by a line consisting of a single ‘#’.

Sample Input:

```
OBVIOUS 1
RRGG RRGG YYBB YYBB
BBGG BBGG YYRR YYRR
EASY 2
BBBB BBBB ..BB ..BB
RRRR RRRR .... ....
RRRR RRRR BBBB BBBB
Is-this-possible? 2
BBBB BBBB ..BB ..BB
RRRR RRRR .... ....
RRBB RRBB BBRR BBRR
#
```

Output Format

Output consists of one line for each problem, consisting of the title of the problem, followed by a space, and either the word ‘noway’ (all lowercase) if the target cannot be achieved, or a description of the pile if the target can be achieved. The tiles in the pile should be listed in order from the top of the pile downwards, in the form: tile number (0 through n-1), slash (‘`/`’), transformation number (0 through 7). Separate descriptions of successive tiles by single spaces.

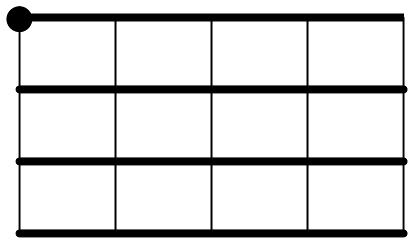
Sample Output:

```
OBVIOUS 0/7
EASY 1/0 0/1
Is-this-possible? noway
```

PROBLEM 8 – POSTAL VANS

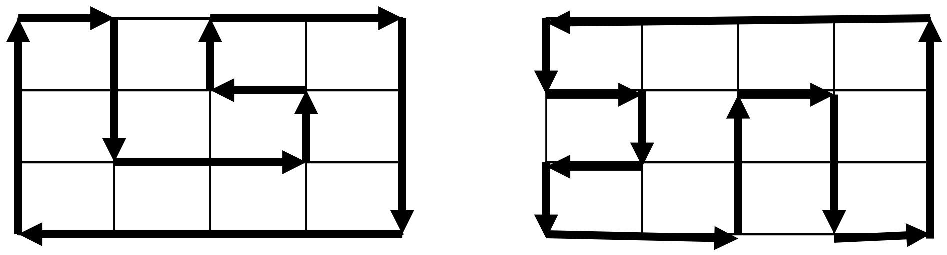
A new suburb has been established with 4 avenues running West–East and *N* streets running North–South, where 1 ≤ *N* ≤ 1000. On the map, the suburb is a rectangular grid and the post office is at its North–West corner.

For example, the following diagram shows such a suburb with *N*=5 streets, with the avenues depicted as horizontal lines, and the post office as a dark blob at the top–left corner:

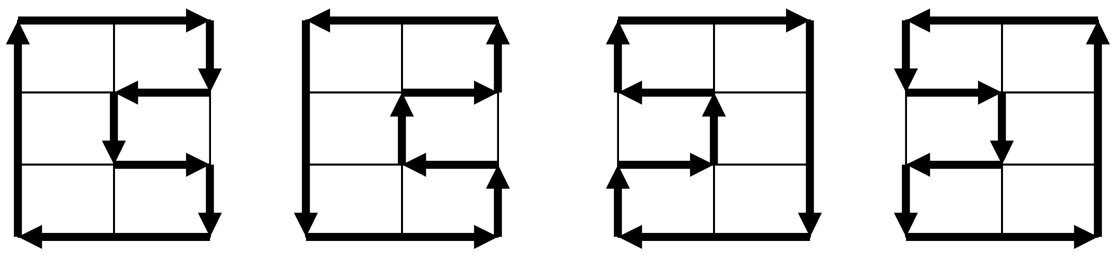


Each day the postal van leaves the post office, drives around the suburb and returns to the post office, passing exactly once through every intersection (including those on borders or corners). The executives from the post company want to know how many distinct routes can be established for the postal van (of course, the route direction is significant in this count).

For example, the following diagrams show 2 such routes for the above suburb:



As another example, the following diagrams show all the 4 possible routes for a suburb with *N*=3 streets.



Write a program that will determine the number of such distinct routes given the number of streets.

INPUT FORMAT

The input text consists of one or more lines, each containing a single number from 1 to 1000 inclusive – the number of parallel streets. A single '#' on a line by itself indicates the end of input.

SAMPLE INPUT:

```
1
2
3
4
10
30
#
```

OUTPUT FORMAT

There is a single line of output for each input value. Each output line consists of the number of streets followed by a colon (':') and a space, followed by the number of possible distinct routes corresponding to that many streets. Each number is displayed as a decimal number, with commas (',') used as separators between groups of 3 digits, counting from the right.

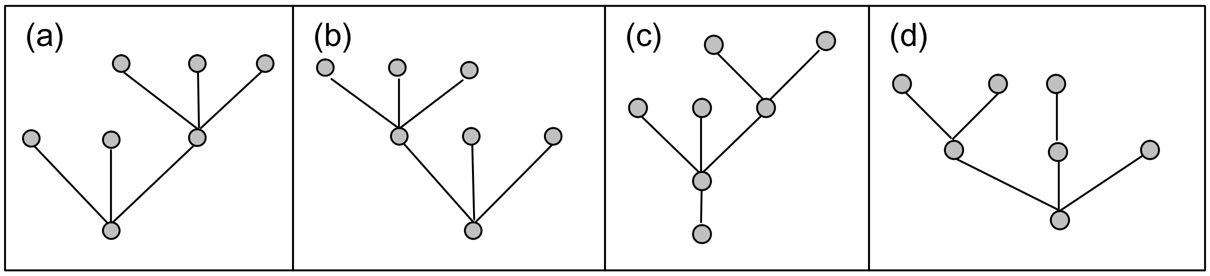
SAMPLE OUTPUT:

```
1: 0
2: 2
3: 4
4: 12
10: 3,034
30: 374,605,036,706
```

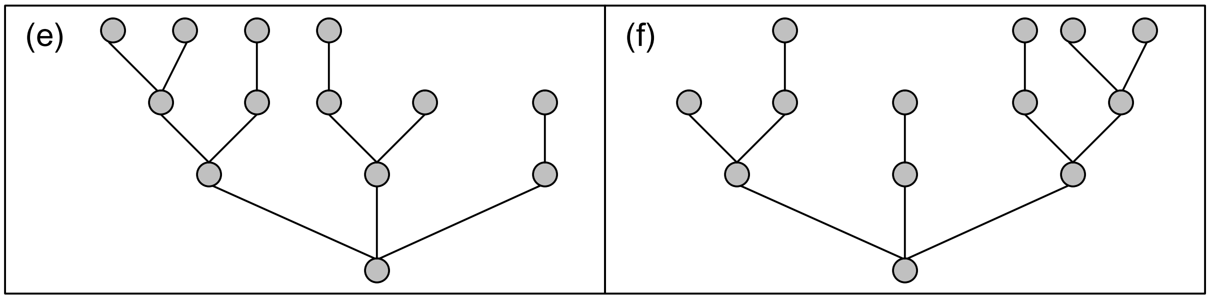
PROBLEM 9 – TRICKY TREES

The stream of tourists passing through the enchanted forest of Acmagorn is drying up because most people have seen it already and don't see the point of seeing it again. This is worrying the custodians of the forest because they rely on the money that tourists spend to manage the forest (and to pay their salaries). In an attempt to bring the tourists back again, they have devised a plan to make the forest look a bit different every day someone sees it. According to this plan, every night the trees randomly rearrange the order of their branches. Although the rearrangement doesn't change the structure of the trees, it can make their appearance quite different.

For example, images (a) and (b) below could show two possible rearrangements of the same tree. Of course, trees that are structurally different trees will always look different, no matter what rearrangement occurs. The trees in images (c) and (d) will never look the same as each other, because they are structurally different – nor will either (c) or (d) look the same as either (a) or (b).



As another example, the images (e) and (f) below could also show two possible rearrangements of the same tree.



This plan is a great success and the tourists are flocking back to see the constantly changing forest. Our friends Mary and Paul have been there several times and each time they've brought home many good pictures. They have now decided to put a bit of order in their collection of pictures, by identifying all matching tree images, i.e., all images that could represent the same tree. To achieve this result, they intend to associate a tag number with each image, such that images of trees are given the same tag if, and only if, they match. For examples, the images (a), (b), (c), (d) above could receive the tags 0, 0, 1, 2, in this order.

They have started by coding their collection of tree images. For each tree they labelled each node, in no particular order, with a distinct number in the range 0 to *k*-1, where *k* is the number of the nodes in that tree.

Your task is to write a program that will allow them to identify matching tree images.

INPUT FORMAT [1]

Input consists of one or more scenarios, with each scenario consisting of a number of tree images. Each scenario starts with a line containing a single number, *n* (1 ≤ *n* ≤ 100) specifying the number of images in the scenario, followed by *n* lines containing image descriptions. Each image description line has at most 5,000 characters and consists of a number *k* (1 ≤ *k* ≤ 1000) specifying the number of nodes in the tree, followed by *k* numbers in the range -1 to *k*-1, specifying the parent of each node in turn, using -1 for the root node (which has no parent). The sequence of scenarios is terminated by an "empty scenario", i.e., a line consisting of single zero ('`0`').

SAMPLE INPUT:

```
4
7 -1 0 0 6 6 6 0
7 -1 3 3 0 3 0 0
7 -1 0 1 1 6 6 1
7 -1 3 3 0 5 0 0
2
13 -1 0 0 0 1 1 2 2 3 4 4 5 6
13 -1 0 0 0 1 1 2 3 3 5 7 8 8
5
2 -1 0
3 -1 0 0
2 1 -1
3 -1 0 1
3 2 2 -1
0
```

OUTPUT FORMAT

Output consists of one line for each scenario, beginning with a scenario sequence number (starting with 1), a colon ('`:`'), a space, and followed by a succession of tag numbers, one for each image, in input order, and separated by single spaces. Tag numbers are allocated successively in input image order, starting with 0, and increased by 1 at the first occurrence of an image not matching any of the previous images.

SAMPLE OUTPUT:

```
1: 0 0 1 2
2: 0 0
3: 0 1 0 2 1
```

[1] Caveat: The input data for this program may contain lines up to 5000 characters long.