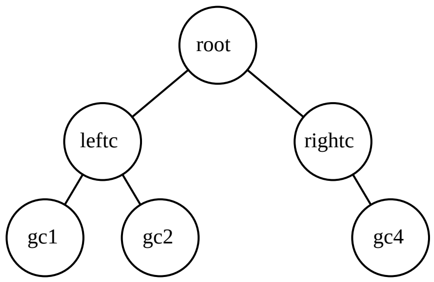
Problem 4: Describing Trees

Trees, particularly binary trees, are a common data structure, and describing them is a common task. One notation for describing a binary tree uses a triple (level, position, label) for each node. Level gives the depth of a node in the tree, with the root node at level 1, its children at level 2, and so forth. Position gives the position of the node among the nodes at the same depth, with the leftmost node at a given depth having position 1, the next node having position 2, and so forth. Label is the text to be used in labeling the node.

For example, the binary tree defined by the set of six input triples shown on the left would normally be illustrated as shown on the right.

```
(1, 1, root) (3, 1, gc1) (2, 2, rightc)
(2, 1, leftc) (3, 2, gc2) (3, 4, gc4)
```



Another representation of a binary tree is procedural. A node is specified using a line beginning with the word "node", followed by the node's position at the current depth and the label for the node. The initial depth is 1, and lines containing only the word "level" increase the tree depth by 1. Using this representation, the tree shown above would be specified like this:

```
node 1 root
level
node 1 leftc
node 2 rightc
level
node 1 gc1
node 2 gc2
node 4 gc4
```

In this problem you are to read a description of a tree using the "triple" notation or the procedural notation. If the tree is not viable – that is, there is a non-root node without a parent, or there are too many nodes on a level, display an appropriate message. If the tree is viable, then display the tree using the other notation (that is, the one not used for the input description). There will be no empty trees in the input, and no two nodes at the same level will have the same position.

Input

There will be multiple test cases to consider. Each case consists of a sequence of lines giving the description of a tree terminated by a line containing only whitespace. All tree descriptions will be syntactically correct, but all trees will not necessarily be viable. The last input case is followed by a line containing only whitespace. Node labels will consist only of alphanumeric characters, and will be no longer than 8 characters. In the triple notation for tree nodes, whitespace (blanks, tabs, and end of line characters) may appear anywhere except in the middle of the decimal integers giving the level or position of a node, or within a node's label. In the procedural notation, blanks or tabs may appear before or after any of the elements on an input line. No tree will have more than 32 levels.

Output

For each input case, first display the case number (1, 2, …). If the tree is viable, then display its representation using the notation not used to describe it in the input. Otherwise display a message indicating the tree is not viable. To display a tree using the "triple" notation, display one triple per line, ordering the triples first by level, and within a level by position. In the procedural notation, display node lines ordered by position. Use the format shown in the sample output.

Sample Input

```
   (1, 1, root)   (3, 1, gc1)   (2, 2, rightc)
      (2, 1, leftc)   (3, 2, gc2)   (3, 4, gc4)
blank line
                 (2, 2, rchild)
   (1, 1, root)
blank line
   (2, 1, root)
blank line
 (1, 1, root)(2, 1, c1)  ( 2,2,c2) (2, 3, c3)
blank line
  node 1 root
level
node 2 rchild
node 1 lchild
blank line
level
node 1 root
blank line
blank line
```

Output for the Sample Input

```
Case 1:
   node 1 root
   level
   node 1 leftc
   node 2 rightc
   level
   node 1 gc1
   node 2 gc2
   node 4 gc4

Case 2:
   node 1 root
   level
   node 2 rchild

Case 3:
    Not viable: a non-root node has no parent.

Case 4:
    Not viable: too many nodes on a level.

Case 5:
   (1, 1, root)
   (2, 1, lchild)
   (2, 2, rchild)

Case 6:
    Not viable: a non-root node has no parent.
```