LCA - Lowest Common Ancestor

*no tags*

A tree is an undirected graph in which any two vertices are connected by exactly one simple path. In other words, any connected graph without cycles is a tree. - Wikipedia

The lowest common ancestor (LCA) is a concept in graph theory and computer science. Let T be a rooted tree with N nodes. The lowest common ancestor is defined between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow a node to be a descendant of itself). - Wikipedia

Your task in this problem is to find the LCA of any two given nodes v and w in a given tree T.

**For example the LCA of nodes 9 and 12 in this tree is the node number 3.**

Input

The first line of input will be the number of test cases. Each test case will start with a number N the number of nodes in the tree, 1 <= N <= 1,000. Nodes are numbered from 1 to N. The next N lines each one will start with a number M the number of child nodes of the Nth node, 0 <= M <= 999 followed by M numbers the child nodes of the Nth node. The next line will be a number Q the number of queries you have to answer for the given tree T, 1 <= Q <= 1000. The next Q lines each one will have two number v and w in which you have to find the LCA of v and w in T, 1 <= v, w <= 1,000.

Input will guarantee that there is only one root and no cycles.

Output

For each test case print Q + 1 lines, The first line will have “Case C:” without quotes where C is the case number starting with 1. The next Q lines should be the LCA of the given v and w respectively.

Example

**Input:**

1

7

3 2 3 4

0

3 5 6 7

0

0

0

0

2

5 7

2 7

**Output:**

Case 1:

3

1

Required time for various technique using fast I/O:  
1. Naive Approach- preprocess: O(N), Query: O(N^2)--------------------------------0.23s  
2. using Square Root Decomposition: preprocess: O(N), Query: O(sqrt(N))--0.04s  
3. using Segment Tree: preprocess: O(NlogN), Query: O(logN)-------------------0.02s  
4. using Sparse Table: preprocess: O(NlogN), Query: O(1)-------------------------0.04s  
5. Binary Lifting: preprocess: O(N), Query: O(logN)-----------------------------------0.03s  
6. Farach Colton and Bender Algorithm: preprocess: O(N), Query: O(1)------0.03s  
7. Tarjan's Offline Algorithm: preprocess: O(N), Query: O(l1)---------------------0.03s