Problem Description

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree. LCA between two nodes p and q is defined as the lowest node in the tree that has both p and q as descendants (where we allow a node to be a descendant of itself)
Note: For the solution method provided, the input parameters are the nodes p and q, not node values. The return value expected from the method is the LCA node and not the node's value. The below Input and Output format show node values which the Driver file will take care of; the user doesn't have to edit the Driver code.

Input format

Line 1 has T: no of test cases given.
Line 2 to X: First test case details with first set of lines giving the binary tree structure (refer section below for the format) and last line giving two node values (integers) whose LCA node has to be found.
Line X+1 to Y: Second Test case details and so on.

Output format

Print the value of the node which is the common ancestor of two nodes.

Constraints

1 <= T <= 1000
1 <= N <= 10000, Number of nodes in the tree
0 <= Node values <= 10^9
It is guaranteed that the sum of the number of tree nodes for all test cases will be < 500000.

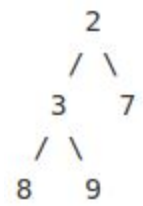
Sample Input 1

```
1
5
2 3 7 8 9
1 2 3
2 4 5
3 -1 -1
4 -1 -1
5 -1 -1
7 2
```

Sample Output 1

```
2
```

Explanation 1



The LCA of node with value 2 and node with value 7 is the node with value 2.

Instructions to create custom input for a Binary Tree

In order to specify a binary tree that can be used as custom input to the problem, you'll need to follow this convention.

- Line 1: Number of nodes in the Binary Tree (N)
- Line 2: N space separated node values. The position of the Nodes on this line will be used to refer to them in the below lines, starting from 1.
- Line 3 to N+2: Lines specifying the child nodes for each of the N nodes

Format of each line (space separated): Parent_node Left_child_node Right_child_node

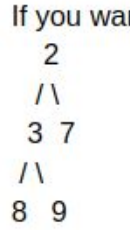
```
* Parent_node - Node at this Position on Line 2 is the Node to which we are assigning the Left and Right child here

* Left_child_node - Node at this position on Line 2 is the left child. Specify -1 if there is no Left child.

* Right_child_node - Node at this position on Line 2 is the right child. Specify -1 if there is no Right child.
```

Example1

If you want to create a Tree that looks like this:



Your input would be:

```
5               → Number of Nodes
2 3 7 8 9       → Node values
1 2 3           → Node 1(value 2) and its child nodes (left child value 3 and right child value 7)
2 4 5           → Node 2(value 3) and its child nodes (left child value 8 and right child value 9)
3 -1 -1         → Node 3(value 7) and its child nodes (left and right child are Null i.e. Leaf Node)
4 -1 -1         → Node 4(value 8) and its child nodes (left and right child are Null i.e. Leaf Node)
5 -1 -1         → Node 5(value 9) and its child nodes (left and right child are Null i.e. Leaf Node)
```