Problem Description

Least Common Ancestor (LCA)

Problem Statement

You are given a binary tree of n nodes, rooted at T. The lowest common ancestor between two nodes ***n1*** and ***n2*** is defined as the lowest node in T that has both ***n1*** and ***n2*** as descendants. (For this problem, we allow a node to be an ancestor/descendant of itself.) You are also given reference of two nodes **a** & **b**, You need to find the LCA of both the nodes.

From wikipedia the definition of LCA is as follows:

The LCA of ***n1*** and ***n2*** in ***T*** is the shared ancestor of ***n1*** and ***n2*** that is located farthest from the root. Computation of lowest common ancestors may be useful, for instance, as part of a procedure for determining the distance between pairs of nodes in a tree: the distance from ***n1*** to ***n2*** can be computed as the distance from the root to ***n1***, plus the distance from the root to ***n2***, minus twice the distance from the root to their lowest common ancestor.

Input Format:

There are three arguments in input, denoting the pointer to the root of the tree ***T*** and reference of two nodes ***a*** & *b* for which you need to return the LCA.
Structure of tree node is as :

```
class Node {
  public:
     int data;
     Node *left;
     Node *right;
};
```

Output Format:

Return an integer denoting the LCA for the given nodes a and b.

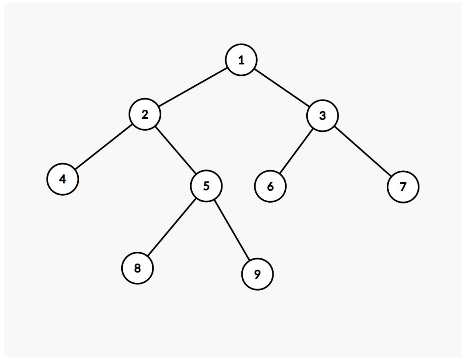
Constraints:

1 <= N <= 100000
1 <= Value at a <= n
1 <= Value at b <= n
Given the value stored at any node will be between 1 to n and unique.

Sample Test Case:

Sample Input:

Let us assume this is the tree, you are given the pointer to 1(Root), and two nodes 8,9



Sample Output:

5

Explanation:

Parent of 8 = 5
Parent of 9 = 5
Clearly we can see that the LCA(8,9) = 5

More examples,
LCA(2,5) = 2
LCA(2,3) = 1

Code Editor

Python 3 (python 3.6.6) Theme: Light Reset My Code Auto Complete On Auto Complete Off

☐ Show Input/Output Code

```
#class Node(object):
#    def __init__(self, data, left=None, right=None):
#        self.data = data
#        self.left = left
#        self.right = right

def lca(root, a, b):
    import sys
    sys.setrecursionlimit(1000000)
    #catching invalid tree
    if root is None:
        return None

    return recurseAncestor(root, a, b)

def recurseAncestor(root, p, q):
    #return None to parent if we go all the way to leaf node and p or q is not found
    if root is None:
        return None

    #if root(current node) is p or q return the root to parent
    if root == p or root == q:
        return root.data
    #recurse on left subtree
    left_data = recurseAncestor(root.left, p, q)
    #recurse on right subtree
    right_data = recurseAncestor(root.right, p, q)
    #if nodes exists on left and right subtree, that means the root must be LCA of both
    if left_data != None and right_data != None:
        return root.data

    #could not find value in left or right subtree, no LCA
    if left_data == None and right_data == None:
        return None

    #could not find in left subtree, that means the first node hit on right is the LCA, the other is child
    if left_data == None and right_data != None:
        return right_data

    #could not find in right subtree, that means the first node hit on left is the LCA, the other is child
    if right_data == None and left_data != None:
        return left_data
```

☐ **Run against custom input**

Quick Test (Takes ~15s)   Full Test (Takes ~45s)

Quick Test Result

No result to show!

Full Test Result

No result to show!