PROBLEM

Kth common ancestor in BST

Medium Accuracy: 61.97% Submissions: 11K+ Points: 4

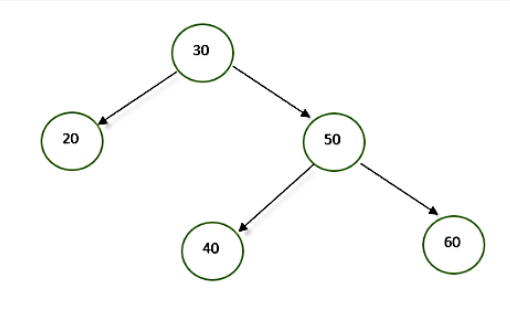
Given a BST with **n (n>=2)** nodes, find the **kth** common ancestor of nodes **x** and **y** in the given tree. Return **-1** if kth ancestor does not exist.

Nodes x and y will always be **present** in the input of a BST, and **x != y**.

Example 1:

Input:

Input tree

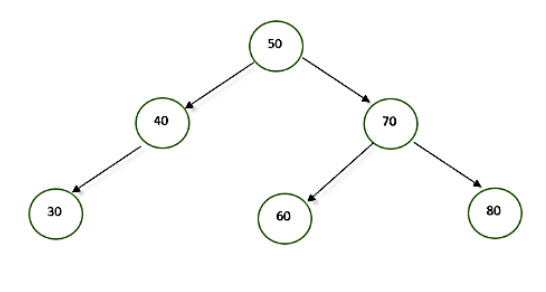


k = 2, x = 40, y = 60

Example 2:

Input:

Input tree



k = 2, x = 40, y = 60

Output:

-1

Explanation:

LCA of 40 and 60 is 50, which is root itself. There does not exists 2nd common ancestor in this case.

Your task :

You don't have to read input or print anything. Your task is to complete the function **kthCommonAncestor()** that takes the **root** of the tree, **k**, **x** and **y** as input and returns the kth common ancestor of x and y.

Expected Time Complexity: O(Height of the Tree)

Expected Space Complexity: O(Height of the Tree)

Your Task :

$1 <= n, k <= 10^5$

$1 <= node->data, x, y <= 10^9$

CODE

```
#User function Template for python3

'''
class Node:
    def init(self, data):
        self.data = data
        self.left = None
        self.right = None
'''
```

```
class Solution:
    def kthCommonAncestor(self, root, k, x, y):
        # Code here

        ans=-1
        d={}
        par=None
        while root:
            d[root]=par
            par=root
            if (x<root.data and y<root.data):
                root=root.left
            elif (x>root.data and y>root.data):
                root=root.right
            else:
                break

        while k>1 and root:
            root=d[root]
            k-=1

        if root:
            return root.data
        return -
```

ArrayList and Queue:

- An ArrayList is like a flexible array that can grow or shrink as needed. Here, it's used to store the numbers we find in the tree in the order we encounter them.
- A Queue is like a line of people waiting their turn. In this case, it's a line of nodes waiting to be visited in the tree.

Traversal:

We start by putting the root of the tree into the queue.

Then, we repeat these steps:

- Take out the first node from the queue.
- If this node has a left child, put it into the queue.
- If this node has a right child, put it into the queue.
- Remember the number in this node.
- We keep doing this until there are no more nodes in the queue.

Result:

- As we visit each node, we remember the number in it and store it in our ArrayList.
- After we've visited all the nodes, we return this list of numbers. This list is our level order traversal of the tree, meaning it shows the numbers in the order we encountered them, level by level, from left to right.

In simple terms, this code helps us explore each "floor" of the tree one by one, from top to bottom, left to right, and write down the numbers we find on each "floor."