# Lowest Common Ancestor of a Binary Tree

Description

Given the root and two nodes in a Binary Tree. Find the lowest common ancestor(LCA) of the two nodes.

The lowest common ancestor is the node with largest depth which is the ancestor of both nodes.

```
`Input: {4,3,7,#,#,5,6},3,5
Output: 4
Explanation:
 For the following binary tree:

      4
     / \
    3   7
       / \
      5   6

 LCA(3, 5) = 4`
This is very very important problem.
The code is:
```

```
def lowestCommonAncestor(self, root, A, B):
        # write your code here
        if root is None:  #Line1
            return None
        if root.val==A.val or root.val==B.val: #Line2
            return root
        left = self.lowestCommonAncestor(root.left,A,B)  #Line3
        right = self.lowestCommonAncestor(root.right,A,B) #Line4

        if left!=None and right!=None:  #Line5
            return root
        else:
            return right if left is None else left   #Line6
```

Now, what we are doing in Line1  we are checking the node. If it is none then we return as we wont have anything beyond this point.

In line 2, we see if our current node is atleast equal to any of the values. If yes we reuturn that node. This is important.

Now, line3 and line4 isnt of much use. It is just recursion.

Now, in line4, if both the left subtree returns non-none and right subtree returns non-none, we know for sure that our this node is LCA.

If not then return that node which is not None(Line6)

## ***Iterative Solution***

```
def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q:
'TreeNode') -> 'TreeNode':
        parent = {}
        parent[root] = None
        stack = [root]

        while p not in parent or q not in parent:

            temp = stack.pop()
            if temp.left:
                parent[temp.left] = temp
                stack.append(temp.left)
            if temp.right:
                parent[temp.right] = temp
                stack.append(temp.right)
        ancestors = set()
        while p:
            ancestors.add(p)
            p = parent[p]
        while q not in ancestors:
            q = parent[q]
        return q
```