rom collections import deque

from queue import Queue

class BinaryNode:

    data=0

    left=None

    right=None

    def \_\_init\_\_(self,data):

        self.data=data

        self.left=None

        self.right=None

class BinaryFunctions:

    def createBinaryTree(self):

        root=None

        data=int(input("enter the data:"))

        if(data==-1):

            return root

        newTreeNode = BinaryNode(data)

        root = newTreeNode

        print("enter left child for :",root.data,":")

        root.left = self.createBinaryTree()

        print("enter right child for :",root.data,":")

        root.right = self.createBinaryTree()

        return root

    def displayLevelWise(self,root):

        if(root == None):

            return Non

        print("Root Data : ",root.data)

        pendingNode = Queue()

        pendingNode.put(root)

        while(pendingNode.qsize() > 0):

            front = pendingNode.get()

            if(front.left!=None):

                print("Left Children of :",front.data," :",front.left.data, end=" ")

                pendingNode.put(front.left)

            if(front.right!=None):

                print("Right Children of :",front.data,": ",front.right.data, end=" ")

                pendingNode.put(front.right)

        print()

    def inorder(self,root):

        if(root==None):

            return None

        self.inorder(root.left)

        print(root.data,end='-')

        self.inorder(root.right)

        return root

    def search(root,key):

        if(root==None):

            return None

        if(key==root.data):

            return root

        if(key<root.data):

            ans=self.search(root.left,key)

        else:

            ans=self.search(root.right,key)

        return ans

    def insertion(self,root,key):

        if(root==None):

            return BinaryNode(key)

        if(key<root.data):

            root.left=self.insertion(root.left,key)

        else:

            root.right=self.insertion(root.right,key)

        return root

    def insertDuplicate(self,root,key):

        if(root==None):

            return None

        if(root.data==key):

            newnode=BinaryNode(key)

            newnode.right=root.right

            root.right=newnode

            return root

        if(key<root.data):

            root.left=self.insertDuplicate(root.left,key)

        else:

            root.right=self.insertDuplicate(root.right,key)

        return root

    def delnode(self,root,key):

        if(root==None):

            return None

        if(key<root.data):

            root.left=self.delNode(root.left,key)

            return root

        elif(key>root.data):

            root.right=self.delNode(root.right,key)

            return root

        else:

            if(root.left==None and root.right==None):

                del root

                return None

            elif(root.left==None):

                temp=root.right

                del root

                return temp

            elif(root.right==None):

                temp=root.left

                del root

                return temp

            else:

                temp=root.right

                minNode=root.right

                root.right=None

                while(minNode.left!=None):

                    minNode=minNode.left

                minNode.left=root.left

                del root

                return temp

    def delNode(self,root, key):

        if(root == None):

            return root

        if(root.data > key):

            root.left = delNode(root.left, key)

            return root

        elif(root.data < key):

            root.right = delNode(root.right, key)

            return root

        else:

            if(root.left == None and root.right == None):

                del root

                return root

            elif(root.left == None):

                temp = root.left

                root.left = None

                del root

                return temp

            elif(root.right == None):

                temp = root.left

                root.left = None

                del root

                return temp

            else:

                rightEl = root.right

                minNode = root.right

                while(minNode.left != None):

                    minNode = minNode.left

                minNode.left = root.left

                root.left = None

                root.right = None

                del root

                return right

obj=BinaryFunctions()

root=obj.createBinaryTree()

root=obj.inorder(root)

val=int(input("\nEnter the value to be insert:"))

root=obj.insertion(root,val)

root=obj.inorder(root)

val=int(input("\nEnter the value to be deleted:"))

root=obj.delNode(root,val)

root=obj.inorder(root)

print()

ele=int(input("enter the duplicate element U want to insert:"))

root=obj.insertDuplicate(root,ele)

obj.displayLevelWise(root)