**PDA CODING ASSIGNMENT WEEK 7**

**1. Write a program to create a tree using any language of your choice. Create a tree using the following numbers as the node.**

**5,4,1,2,3,6,7**

**CODE:-**

class newnode:

    def \_\_init\_\_(self,data):

        self.data=data

        self.left=None

        self.right=None

def printr(Root):

    if Root:

        print(Root.data)

        printr(Root.left)

        printr(Root.right)

Root= newnode(5)

Root.left=newnode(4)

Root.right=newnode(1)

Root.left.left=newnode(2)

Root.left.right=newnode(3)

Root.right.left=newnode(6)

Root.right.right=newnode(7)

printr(Root)

**OUTPUT:-**

5

4

2

3

1

6

7

**2. Once the above tree is created, write a function to search a value from the tree, if the value is found print “Found” or print “Not found”.**

**CODE:-**

class newnode:

    def \_\_init\_\_(self,data):

        self.data= data

        self.left= None

        self.right= None

def printr(Root):

    if Root:

        print(Root.data)

        printr(Root.left)

        printr(Root.right)

def value\_available\_or\_not(node,key):

    if(node==None):

        return False

    if(node.data==key):

        return True

    lsearch= value\_available\_or\_not(node.left,key)

    if lsearch:

        return lsearch

    rsearch= value\_available\_or\_not(node.right,key)

    return rsearch

Root= newnode(5)

Root.left=newnode(4)

Root.right=newnode(1)

Root.left.left=newnode(2)

Root.left.right=newnode(3)

Root.right.left=newnode(6)

Root.right.right=newnode(7)

printr(Root)

key=5

print("Value to be searched is 5")

final= value\_available\_or\_not(Root,key)

if final:

    print("Found")

else:

    print("Not Found")

**OUTPUT:-**

5

4

2

3

1

6

7

Value to be searched is 5

Found

**3. Once the above tree is created, get a particular value as input from the user and delete the value in the tree created.**

**CODE:-**

class newnode:

    def \_\_init\_\_(self,data):

        self.data=data

        self.left=None

        self.right=None

def deleteDeepest(node,deep\_key\_node):

    q = []

    q.append(node)

    while(len(q)):

        temp = q.pop(0)

        if temp is deep\_key\_node:

            temp = None

            return

        if temp.right:

            if temp.right is deep\_key\_node:

                temp.right = None

                return

            else:

                q.append(temp.right)

        if temp.left:

            if temp.left is deep\_key\_node:

                temp.left = None

                return

            else:

                q.append(temp.left)

def delete\_value(node,key):

    if(node==None):

        return node

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

        if (node.data==key):

            return None

        else:

            return node

    key\_node = None

    q = []

    q.append(node)

    temp = None

    while(len(q)):

        temp = q.pop(0)

        if temp.data == key:

            key\_node = temp

        if temp.left:

            q.append(temp.left)

        if temp.right:

            q.append(temp.right)

    if key\_node :

        x = temp.data

        deleteDeepest(node,temp)

        key\_node.data = x

    return node

def printr(Root):

    if Root:

        print(Root.data)

        printr(Root.left)

        printr(Root.right)

Root= newnode(5)

Root.left=newnode(4)

Root.right=newnode(1)

Root.left.left=newnode(2)

Root.left.right=newnode(3)

Root.right.left=newnode(6)

Root.right.right=newnode(7)

print("Original tree")

printr(Root)

del\_value=4

delete\_value(Root,del\_value)

print("New tree after deletion")

printr(Root)

**OUTPUT:-**

Original tree

5

4

2

3

1

6

7

New tree after deletion

5

7

2

3

1

6

**4. Perform Pre-order, Post-order and In-order traversal in the tree created above.**

**CODE:-**

class newnode:

    def \_\_init\_\_(self,data):

        self.data=data

        self.left=None

        self.right=None

def deleteDeepest(node,deep\_key\_node):

    q = []

    q.append(node)

    while(len(q)):

        temp = q.pop(0)

        if temp is deep\_key\_node:

            temp = None

            return

        if temp.right:

            if temp.right is deep\_key\_node:

                temp.right = None

                return

            else:

                q.append(temp.right)

        if temp.left:

            if temp.left is deep\_key\_node:

                temp.left = None

                return

            else:

                q.append(temp.left)

def delete\_value(node,key):

    if(node==None):

        return node

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

        if (node.data==key):

            return None

        else:

            return node

    key\_node = None

    q = []

    q.append(node)

    temp = None

    while(len(q)):

        temp = q.pop(0)

        if temp.data == key:

            key\_node = temp

        if temp.left:

            q.append(temp.left)

        if temp.right:

            q.append(temp.right)

    if key\_node :

        x = temp.data

        deleteDeepest(node,temp)

        key\_node.data = x

    return node

def printPreorder(Root):

    if Root:

        print(Root.data)

        printPreorder(Root.left)

        printPreorder(Root.right)

def printPostorder(Root):

    if Root:

        printPostorder(Root.left)

        printPostorder(Root.right)

        print(Root.data)

def printInorder(Root):

    if Root:

        printInorder(Root.left)

        print(Root.data)

        printInorder(Root.right)

Root= newnode(5)

Root.left=newnode(4)

Root.right=newnode(1)

Root.left.left=newnode(2)

Root.left.right=newnode(3)

Root.right.left=newnode(6)

Root.right.right=newnode(7)

print("Original\n")

printPreorder(Root)

del\_value=4

delete\_value(Root,del\_value)

print("\nAfter deletion of element")

print("\nPreorder\n")

printPreorder(Root)

print("\nPostorder\n")

printPostorder(Root)

print("\nInorder\n")

printInorder(Root)

**OUTPUT:-**

Original

5

4

2

3

1

6

7

After deletion of element

Preorder

5

7

2

3

1

6

Postorder

2

3

7

6

1

5

Inorder

2

7

3

5

6

1