University of Baltistan Skardu

Submitted to:	**Maam Noreen Maryam**
Submitted By:	**Basit Ali**
Registration No:	**S23BSCS012**
Department:	**Computer Science**
Date:	**May 01,2024**
Section:	**“A”**

Lab No 13:

Q1: Add two more function preorder and post order traversal in the Example#1 Lab#13. Then explain the code of each function diagrammatically.

Program:

```
class Node:
    def __init__(self,left=None,item=None,right=None):
        self.left=left
        self.item=item
        self.right=right
class BST:
    def __init__(self):
        self.root=None
    def insert(self,data):
        self.root=self.rinsert(self.root,data)
    def rinsert(self,root,data):
        if root is None:
            return Node(None,data,None)
        if data<root.item:
            root.left=self.rinsert(root.left,data)
        elif data>root.item:
            root.right=self.rinsert(root.right,data)
        return root
    def inorder(self):
        result=[]
        self.rinorder(self.root,result)
        return result
    def rinorder(self,root,result):
        if root:
            self.rinorder(root.left,result)
            result.append(root.item)
            self.rinorder(root.right,result)
    def preorder(self):
        result=[]
        self.rpreorder(self.root,result)
        return result
```

```
def rpreorder(self,root,result):
    if root:
        result.append(root.item)
        self.rpreorder(root.left,result)
        self.rpreorder(root.right,result)
```

```
    def postorder(self):
        result=[]
        self.rpostorder(self.root,result)
        return result
    def rpostorder(self,root,result):
        if root:
            self.rpostorder(root.left,result)
            self.rpostorder(root.right,result)
            result.append(root.item)
t1=BST()
t1.insert(80)
t1.insert(100)
t1.insert(70)
t1.insert(90)
t1.insert(60)
print("Inorder traversing: ",t1.inorder())
print("Preorder traversing ",t1.preorder())
print("Postorder traversing: ",t1.postorder())
```

Output:

```
Inorder traversing:  [60, 70, 80, 90, 100]
Preorder traversing:  [80, 70, 60, 100, 90]
```

```
Postorder traversing:  [60, 70, 90, 100, 80]
```

Explanation:

```
print("Preorder traversing: ",t1.preorder())
```

```
def preorder(self):
    result=[]
    self.rpreorder(self.root,result)
    return result
```

Return 80,70,60,100,90

(1)

```
def rpreorder(self,root,result):
    if root:
        result.append(root.item)          1
        self.rpreorder(root.left,result)  2
        self.rpreorder(root.right,result) 3
```

Root=80

1)result= [80]

2)root.Left=70

2)

```
def rpreorder(self,root,result):
    if root:
        result.append(root.item)          2.1
        self.rpreorder(root.left,result)  2.2
        self.rpreorder(root.right,result) 2.3
```

Root=70

1)Result

[80,70]

2)

root.Left=600

2.2)

```
def rpreorder(self,root,result):
    if root:
        result.append(root.item)          2.2.1
        self.rpreorder(root.left,result)  2.2.2
        self.rpreorder(root.right,result) 2.2.3
```

Root=60

1)Result

[80,70,60]

2)

root.left:None

Step 02 complete because the root. Left is none now step 03 will be start.in the step 03 root is 80 and root. Right is 90. when the root is 90 then root. Right is 100.

3

```
def rpreorder(self,root,result):
    if root:
        result.append(root.item)          3.1
        self.rpreorder(root.left,result)  3.2
        self.rpreorder(root.right,result) 3.3
```

root=90

1)result

[80,70,60

90]

root.right=

100

3.3

```
def rpreorder(self,root,result):
    if root:
        result.append(root.item)          3.3.1
        self.rpreorder(root.left,result)  3.3.2
        self.rpreorder(root.right,result) 3.3.3
```

Root=100

1)result

[80,70,60

100,90]

Root.right=

None

The result is [80,70,60,100,90]

Explanation of post order

```
print("Postorder traversing: ",t1.postorder())
```

```
    def postorder(self):
        result=[]
        self.rpostorder(self.root,result)
        return result
```

Return [60,70,90,100,80]

The root is 80.but the root.left is 70.it append 60 at first at in result.

1

```
def rpostorder(self,root,result):
    if root:
        self.rpostorder(root.left,result)
        self.rpostorder(root.right,result)
        result.append(root.item)
```

1

2

3

Root=60

Result= [60]

root.left=None

1.1)

```
def rpostorder(self,root,result):
    if root:
        self.rpostorder(root.left,result)
        self.rpostorder(root.right,result)
        result.append(root.item)
```

1.1.1

1.1.2

1.1.3

root=70

result = [60,70]

root.left=60

2.1)

```
def rpostorder(self,root,result):
    if root:
        self.rpostorder(root.left,result)
        self.rpostorder(root.right,result)
        result.append(root.item)
```

2.`1.1

2.1.2

2.1.3

root=90

result= [60,70,90]

root.right=100

2.2)

```
def rpostorder(self,root,result):
    if root:
        self.rpostorder(root.left,result)
        self.rpostorder(root.right,result)
        result.append(root.item)
```

2.2.1

2.2.2

2.2.3

result=

[60,70,90,100]

root.right=None

3.1)

```
def rpostorder(self,root,result):
    if root:
        self.rpostorder(root.left,result)
        self.rpostorder(root.right,result)
        result.append(root.item)
```

3.1.1

3.1.2

3.1.3

root=80

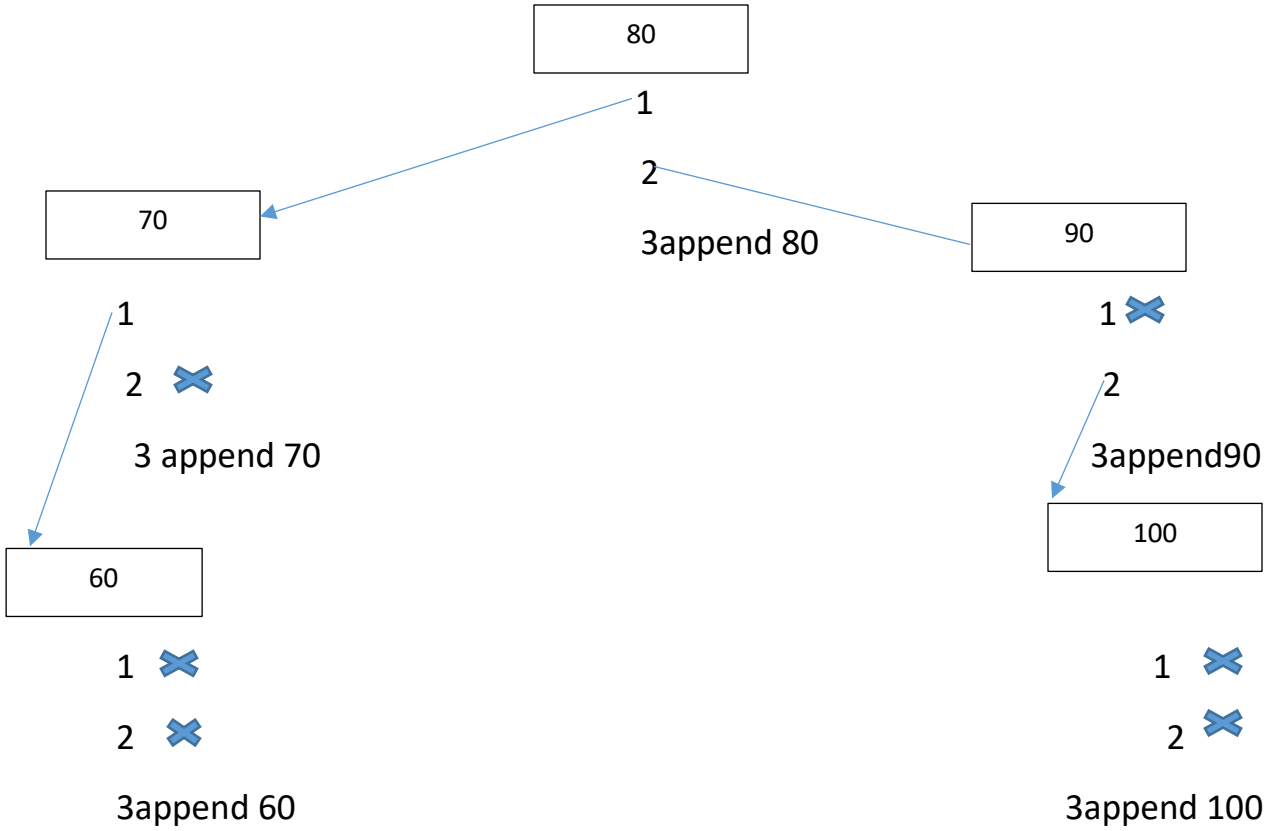
result= [60,70,

90,100,80]

root.left=70

root.right=90

result= [60,70,90,100,80]



RESULT IN Post order= [60,70,90,100,80]

Lab No 14:

Q1:

Write a program to insert remove index in the lab#14 to remove any index.

Program:

```
class graph:
    def __init__(self,vn0):
        self.vertex_count=vn0
        self.matrix=[[0]*vn0 for e in range(vn0)]
    def add_edge(self,u,v,weight=1):
        if 0<=u<self.vertex_count and 0<=v<self.vertex_count:
            self.matrix[u][v]=weight
            self.matrix[v][u]=weight
        else:
            print("Invalid vertex")
    def remove_edge(self,u,v):
        if 0<=u<self.vertex_count and 0<=v<self.vertex_count:
            self.matrix[u][v]=0
            self.matrix[v][u]=0
        else:
            print("Invalid vertex")
    def print_matrix(self):
        for row in self.matrix:
            print(' '.join(map(str,row)))
g=graph(3)
g.add_edge(1,1)
g.add_edge(0,2)
g.add_edge(2,2)
g.print_matrix()
print("After removing the element of 2,2")
g.remove_edge(2,2)
g.print_matrix()
```

Output:

```
0 0 1
0 1 0
1 0 1
After removing the element of 2,2
0 0 1
0 1 0
1 0 0
```

Lab No 15:

Q1: In Lab#15 example 2 insert a new function to remove the edge and the weight is taken by the user?

Program

```
class Graph:
    def __init__(self,vn0):
        self.vertex_count=vn0
        self.adj_list={v:[] for v in range(vn0)}

    def add_edge(self,u,v,weight=input("Enter the weight")):
        if 0<=u<self.vertex_count and 0<=v<self.vertex_count:
            self.adj_list[u].append((v,weight))
            self.adj_list[v].append((u,weight))
        else:
            print("Invalid vertices")
    def remove_edge(self,u,v):
        if 0<=u<self.vertex_count and 0<=v<self.vertex_count:
            self.adj_list[u]=[(vertex,weight)for vertex,weight in self.adj_list[u] if vertex!=v]
            self.adj_list[v]=[(vertex,weight)for vertex,weight in self.adj_list[v] if vertex!=v]
        else:
            print("Invalid vertices")
    def print_adj_list(self):
        for vertex,n in self.adj_list.items():
            print('v',vertex,':',n)
g=Graph(3)
g.add_edge(0,0)
g.add_edge(0,1)
g.add_edge(0,2)
g.add_edge(1,0)
g.add_edge(1,1)
g.add_edge(1,2)
g.add_edge(2,0)
g.add_edge(2,1)
g.add_edge(2,2)
g.print_adj_list()
```

```
print("After removing")
g.remove_edge(0,0)
g.remove_edge(1,1)
g.remove_edge(2,2)
g.remove_edge(0,1)
g.remove_edge(0,2)
g.remove_edge(1,2)
g.remove_edge(1,0)
g.remove_edge(2,0)
g.remove_edge(2,1)
g.print_adj_list()
```

Output:

```
Enter the weight4
v 0 : [(0, '4'), (0, '4'), (1, '4'), (2, '4'), (1, '4'), (2, '4')]
v 1 : [(0, '4'), (0, '4'), (1, '4'), (1, '4'), (2, '4'), (2, '4')]
v 2 : [(0, '4'), (1, '4'), (0, '4'), (1, '4'), (2, '4'), (2, '4')]
After removing
v 0 : []
v 1 : []
v 2 : []
PS E:\coding\lab6.py>
```