**NAME :** DUDHE ANIL AJINATH

**ROLL NO :** TEAD21269

**SUBJECT :** AI

**CLASS :** TE

**BRANCH :** AI&DS

**EXPERIMENT NO :**

**TITLE :**

**Implement depth first search algorithm and Breadth First Search algorithm. Use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.**

**CODE:**

class Node:

def \_\_init\_\_(self, data):

self.left = None

self.right = None

self.data = data

# Insert Node

def insert(self, data):

if self.data:

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

elif data > self.data:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

else:

self.data = data

# Print the Tree

def PrintTree(self):

if self.left:

self.left.PrintTree()

print(self.data, end=" ")

if self.right:

self.right.PrintTree()

# Inorder traversal

def inorderTraversal(self):

res = []

if self.left:

res = self.left.inorderTraversal()

res.append(self.data)

if self.right:

res = res + self.right.inorderTraversal()

return res

# Preorder traversal (Root -> Left -> Right)

def PreorderTraversal(self):

res = []

res.append(self.data)

if self.left:

res = res + self.left.PreorderTraversal()

if self.right:

res = res + self.right.PreorderTraversal()

return res

# Postorder traversal

def PostorderTraversal(self):

res = []

if self.left:

res = self.left.PostorderTraversal()

if self.right:

res = res + self.right.PostorderTraversal()

res.append(self.data)

return res

# Function to print level order traversal of tree

def printLevelOrder(root):

h = height(root)

for i in range(1, h + 1):

printCurrentLevel(root, i)

# Print nodes at a current level

def printCurrentLevel(root, level):

if root is None:

return

if level == 1:

print(root.data, end=" ")

elif level > 1:

printCurrentLevel(root.left, level - 1)

printCurrentLevel(root.right, level - 1)

def height(node):

if node is None:

return 0

else:

lheight = height(node.left)

rheight = height(node.right)

if lheight > rheight:

return lheight + 1

else:

return rheight + 1

# Input for tree

n = int(input("Enter the number of nodes in the tree: "))

flag = False

for i in range(n):

if flag == False:

r = int(input("Enter the value of root: "))

root = Node(r)

flag = True

else:

r = int(input("Enter the value of node: "))

root.insert(r)

getInput = int(input("Enter the number of operation to perform (1.BFS, 2.DFS): "))

if getInput == 1:

print("Level Order Traversal:")

printLevelOrder(root)

elif getInput == 2:

order = int(input("Enter the order of DFS (1.Inorder, 2.Pre-Order, 3.Post-order): "))

if order == 1:

print("Inorder Traversal:")

print(root.inorderTraversal())

elif order == 2:

print("Preorder Traversal:")

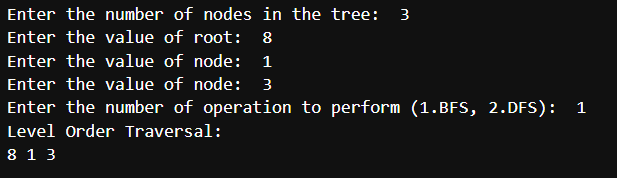
print(root.PreorderTraversal())

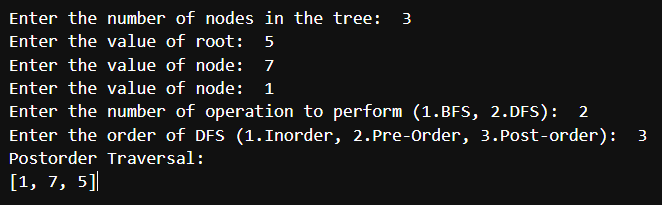
elif order == 3:

print("Postorder Traversal:")

print(root.PostorderTraversal())

**OUTPUT:**

****

****