**Binary Seach Tree Implementation**

class TreeNode:

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

self.key = key

self.left = None

self.right = None

class BST:

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

self.root = None

**#choose operation**

def operate(self,key,operation):

if(operation == 'insert'):

self.root = self.insert\_recursive(self.root,key)

elif(operation == 'inorder'):

self.inorder\_recursive(self.root)

elif(operation == 'preorder'):

self.preorder\_recursive(self.root)

elif(operation == 'postorder'):

self.postorder\_recursive(self.root)

elif(operation == 'find\_min'):

result = self.find\_min(self.root)

return result

elif(operation == 'find\_max'):

result = self.find\_max(self.root)

return result

elif(operation == 'find\_ele'):

key = int(input("Enter the element you want to search"))

result = self.search\_element(self.root,key)

print("The element search status : ",result)

elif(operation == 'levelorder'):

self.levelorder\_traversal(self.root)

elif(operation == 'height'):

result = self.height\_tree(self.root)

return result

elif(operation == 'delete'):

key = int(input("Enter the element you want to delete : "))

self.inorder\_recursive(self.root)

self.delete\_node(self.root,key)

print("\n")

self.inorder\_recursive(self.root)

**#insert elements**

def insert\_recursive(self,root,key):

if(root is None):

return TreeNode(key)

if(key < root.key):

root.left = self.insert\_recursive(root.left,key)

elif(key > root.key):

root.right = self.insert\_recursive(root.right,key)

return root

**#inorder traversal of BST**

def inorder\_recursive(self,root):

if(root is not None):

self.inorder\_recursive(root.left)

print(root.key,end=" ")

self.inorder\_recursive(root.right)

**#preorder traversal of BST**

def preorder\_recursive(self,root):

if(root is not None):

print(root.key,end=" ")

self.preorder\_recursive(root.left)

self.preorder\_recursive(root.right)

**#post order traversal of BST**

def postorder\_recursive(self,root):

if(root is not None):

self.postorder\_recursive(root.left)

self.postorder\_recursive(root.right)

print(root.key,end=" ")

**#level order traversal of BST**

def levelorder\_traversal(self,root):

if(root is None):

return None

queue = []

queue.append(root)

while(len(queue) != 0):

ele = queue.pop(0)

print(ele.key,end=" ")

if(ele.left is not None):

queue.append(ele.left)

if(ele.right is not None):

queue.append(ele.right)

**#find minimum element in BST**

def find\_min(self,root):

if(root is None):

return None

while(root.left is not None):

root = root.left

return root.key

**#find maximum element in BST**

def find\_max(self,root):

if(root is None):

return None

while(root.right is not None):

root = root.right

return root.key

**#search for an element in BST**

def search\_element(self,root,key):

if(root is None):

return None

if(root.key == key):

return True

elif(key<root.key):

return self.search\_element(root.left,key)

else:

return self.search\_element(root.right,key)

return False

**#height of a BST**

def height\_tree(self,root):

if(root is None):

return -1

return(max(self.height\_tree(root.left),self.height\_tree(root.right))+1)

**#delete a node from BST**

def delete\_node(self,root,key):

if(root is None):

print("Tree is Empty")

return

if(key < root.key):

root.left = self.delete\_node(root.left,key)

elif(key > root.key):

root.right = self.delete\_node(root.right,key)

else:

if(root.left is None):

return root.right

elif(root.right is None):

return root.left

root.key = self.find\_min\_node(root.right).key

root.right = self.delete\_node(root.right,root.key)

return root

**#secondary function used to find the minimum value from RST in BST**

def find\_min\_node(self,root):

current = root

while(current.left is not None):

current = current.left

return current

bst = BST()

elements = [10,1,13,133,100,23,22]

for i in elements:

bst.operate(i,"insert")

print("Inorder Traversal:")

bst.operate(None, 'inorder')

print("\nPreorder Traversal:")

bst.operate(None, 'preorder')

print("\nPostorder Traversal:")

bst.operate(None, 'postorder')

print("\nminimum element in the tree :")

print(bst.operate(None,'find\_min'))

print("maximum element in the tree :")

print(bst.operate(None,'find\_max'))

#bst.operate(None,'find\_ele')

print("\nlevelorder Traversal:")

bst.operate(None, 'levelorder')

print("\nHeight of the tree : ")

print(bst.operate(None,'height'))