**Assignment 21 - Tree**

**Question-1**

You are given a binary tree. The binary tree is represented using the TreeNode class. Each TreeNode has an integer value and left and right children, represented using the TreeNode class itself. Convert this binary tree into a binary search tree.

Input:

10

/ \\

2 7

/ \

8 4

Output:

8

/ \\

4 10

/ \

2 7

**Sol.**

class TreeNode:

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

self.val = value

self.left = None

self.right = None

def convert\_binary\_tree\_to\_bst(root):

if not root:

return

# Step 1: Traverse the binary tree and store values in a list

values = []

traverse\_in\_order(root, values)

# Step 2: Sort the list of values

values.sort()

# Step 3: Traverse the binary tree again and replace node values

index = 0

replace\_values(root, values, index)

return root

def traverse\_in\_order(node, values):

if not node:

return

traverse\_in\_order(node.left, values)

values.append(node.val)

traverse\_in\_order(node.right, values)

def replace\_values(node, values, index):

if not node:

return

replace\_values(node.left, values, index)

node.val = values[index]

index += 1

replace\_values(node.right, values, index)

# Create the binary tree

root = TreeNode(10)

root.left = TreeNode(2)

root.right = TreeNode(7)

root.left.left = TreeNode(8)

root.left.right = TreeNode(4)

# Convert the binary tree to a binary search tree

converted\_root = convert\_binary\_tree\_to\_bst(root)

# Print the resulting binary search tree

print(converted\_root.val)

print(converted\_root.left.val, converted\_root.right.val)

print(converted\_root.left.left.val, converted\_root.left.right.val)

8

4 10

2 7

**Question-2:**

Given a Binary Search Tree with all unique values and two keys. Find the distance between two nodes in BST. The given keys always exist in BST.

Example:

Consider the following BST:

**Input-1:**

n = 9

values = [8, 3, 1, 6, 4, 7, 10, 14,13]

node-1 = 6

node-2 = 14

**Output-1:**

The distance between the two keys = 4

**Input-2:**

n = 9

values = [8, 3, 1, 6, 4, 7, 10, 14,13]

node-1 = 3

node-2 = 4

**Output-2:**

The distance between the two keys = 2

**Sol:**

class TreeNode:

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

self.val = value

self.left = None

self.right = None

def find\_distance(root, node1, node2):

if not root:

return 0

# Find the Lowest Common Ancestor (LCA) of the given nodes

lca = find\_lca(root, node1, node2)

# Find the distances from LCA to node1 and node2

dist1 = find\_distance\_from\_node(lca, node1)

dist2 = find\_distance\_from\_node(lca, node2)

# Return the sum of the distances

return dist1 + dist2

def find\_lca(root, node1, node2):

if not root:

return None

# If both nodes are smaller, move to the left subtree

if node1.val < root.val and node2.val < root.val:

return find\_lca(root.left, node1, node2)

# If both nodes are greater, move to the right subtree

if node1.val > root.val and node2.val > root.val:

return find\_lca(root.right, node1, node2)

# Found the Lowest Common Ancestor

return root

def find\_distance\_from\_node(node, target):

if not node:

return 0

# If the target node is found, return the distance

if node.val == target.val:

return 0

# If the target is smaller, move to the left subtree

if target.val < node.val:

return 1 + find\_distance\_from\_node(node.left, target)

# If the target is greater, move to the right subtree

if target.val > node.val:

return 1 + find\_distance\_from\_node(node.right, target)

# Create the binary search tree

root = TreeNode(8)

root.left = TreeNode(3)

root.right = TreeNode(10)

root.left.left = TreeNode(1)

root.left.right = TreeNode(6)

root.left.right.left = TreeNode(4)

root.left.right.right = TreeNode(7)

root.right.right = TreeNode(14)

root.right.right.left = TreeNode(13)

# Specify the nodes

node1 = TreeNode(6)

node2 = TreeNode(14)

# Find the distance between the two nodes

distance = find\_distance(root, node1, node2)

# Print the result

print("The distance between the two keys =", distance)

The distance between the two keys = 4

**Question-3:**

Write a program to convert a binary tree to a doubly linked list.

**Input:**

10

/ \\

5 20

/ \\

30 35

**Output:**

5 10 30 20 35

class TreeNode:

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

self.val = value

self.left = None

self.right = None

self.prev = None

self.next = None

def convert\_to\_doubly\_linked\_list(root):

global prev\_node

prev\_node = None

convert(root)

# Find the head of the doubly linked list

head = prev\_node

while head and head.prev:

head = head.prev

return head

def convert(node):

global prev\_node

if node is None:

return

convert(node.left)

# Set the prev pointer

node.prev = prev\_node

if prev\_node:

prev\_node.next = node

prev\_node = node

convert(node.right)

# Create the binary tree

root = TreeNode(10)

root.left = TreeNode(5)

root.right = TreeNode(20)

root.right.left = TreeNode(30)

root.right.right = TreeNode(35)

# Convert the binary tree to a doubly linked list

head = convert\_to\_doubly\_linked\_list(root)

# Traverse the doubly linked list and print the values

current = head

while current:

print(current.val, end=" ")

current = current.next

5 10 30 20 35

**Question-4:**

Write a program to connect nodes at the same level.

**Input:**

1

/ \\

2 3

/ \ / \

4 5 6 7

**Output:**

1 → -1

2 → 3

3 → -1

4 → 5

5 → 6

6 → 7

7 → -1

**Sol:**

from collections import deque

class TreeNode:

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

self.val = value

self.left = None

self.right = None

self.nextRight = None

def connect\_nodes\_at\_same\_level(root):

if not root:

return None

queue = deque([root])

while queue:

level\_size = len(queue)

for i in range(level\_size):

node = queue.popleft()

# Set the nextRight attribute

if i < level\_size - 1:

node.nextRight = queue[0]

# Enqueue the left and right children

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

return root

# Create the binary tree

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.left.left = TreeNode(4)

root.left.right = TreeNode(5)

root.right.left = TreeNode(6)

root.right.right = TreeNode(7)

# Connect nodes at the same level

connect\_nodes\_at\_same\_level(root)

# Print the connections

current = root

while current:

node = current

while node:

if node.nextRight:

print(f"{node.val} -> {node.nextRight.val}")

else:

print(f"{node.val} -> -1")

node = node.nextRight

current = current.left

**output**

1 -> -1

2 -> 3

3 -> -1

4 -> 5

5 -> 6

6 -> 7

7 -> -1