**Assignment Questions 21**

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

Solve:-

class TreeNode:

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

self.value = value

self.left = None

self.right = None

def binary\_tree\_to\_bst(root):

values = []

inorder\_traversal(root, values)

# Sort the list

values.sort()

# Traverse the binary tree and replace the values with the sorted values

index = [0] # Mutable index to keep track of the current value index in the sorted list

inorder\_update(root, values, index)

def inorder\_traversal(node, values):

if node is None:

return

inorder\_traversal(node.left, values)

values.append(node.value)

inorder\_traversal(node.right, values)

def inorder\_update(node, values, index):

if node is None:

return

inorder\_update(node.left, values, index)

node.value = values[index[0]]

index[0] += 1

inorder\_update(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

binary\_tree\_to\_bst(root)

def inorder\_print(node):

if node is None:

return

inorder\_print(node.left)

print(node.value)

inorder\_print(node.right)

inorder\_print(root)

💡 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

Solve:-

class TreeNode:  
 def \_\_init\_\_(self, value):  
 self.value = value  
 self.left = None  
 self.right = None  
  
def construct\_bst(values):  
 root = None  
 for value in values:  
 root = insert\_node(root, value)  
 return root  
  
def insert\_node(root, value):  
 if root is None:  
 return TreeNode(value)  
 if value < root.value:  
 root.left = insert\_node(root.left, value)  
 else:  
 root.right = insert\_node(root.right, value)  
 return root  
  
def find\_lca(root, node1, node2):  
 if root is None or root.value == node1 or root.value == node2:  
 return root  
  
 if node1 < root.value and node2 < root.value:  
 return find\_lca(root.left, node1, node2)  
 elif node1 > root.value and node2 > root.value:  
 return find\_lca(root.right, node1, node2)  
 else:  
 return root  
  
def find\_distance(root, node, distance):  
 if root is None:  
 return -1  
  
 if root.value == node:  
 return distance  
  
 if node < root.value:  
 return find\_distance(root.left, node, distance + 1)  
 else:  
 return find\_distance(root.right, node, distance + 1)  
  
def distance\_between\_nodes(root, node1, node2):  
 lca = find\_lca(root, node1, node2)  
 distance1 = find\_distance(lca, node1, 0)  
 distance2 = find\_distance(lca, node2, 0)  
 return distance1 + distance2  
  
#Example input  
values = [8, 3, 1, 6, 4, 7, 10, 14, 13]  
node1 = 6  
node2 = 14  
  
#Construct the BST  
root = construct\_bst(values)  
  
# Find the distance between the 2 nodes  
distance = distance\_between\_nodes(root, node1, node2)  
  
# Print the result  
print("The distance between the two keys =", distance)

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

</aside>

<aside> 💡 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

Solve:-

class TreeNode:  
 def \_\_init\_\_(self, value):  
 self.value = value  
 self.left = None  
 self.right = None  
  
def binary\_tree\_to\_dll(root):  
 if root is None:  
 return None  
  
 # Initialize previous node as None (to keep track of the previous node in the in-order traversal)  
 prev = None  
  
 #Convert the left subtree  
 head = binary\_tree\_to\_dll(root.left)  
  
  
 root.left = prev  
  
  
 if prev is not None:  
 prev.right = root  
  
 # Update the previous node to the current node  
 prev = root  
  
 # Convert the right subtree  
 binary\_tree\_to\_dll(root.right)  
  
  
 if head is None:  
 head = root  
  
 return head  
  
def print\_doubly\_linked\_list(head):  
 curr = head  
 while curr is not None:  
 print(curr.value, end=" ")  
 curr = curr.right  
 print()  
  
# 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 = binary\_tree\_to\_dll(root)  
  
# Print the doubly linked list  
print\_doubly\_linked\_list(head)