Q.Given a binary tree, print all its root-to-leaf paths without using recursion. For example, consider the following Binary Tree.

Input:

6

/ \\

3 5

/ \ \ 2 5 4 / \ 7 4

Output:

There are 4 leaves, hence 4 root to leaf paths - 6->3->2 6->3->5->7 6->3->5->4 6->5>4

Q. Given Preorder, Inorder and Postorder traversals of some tree. Write a program to check if they all are of the same tree.

**Examples:**

Input :

Inorder -> 4 2 5 1 3

Preorder -> 1 2 4 5 3

Postorder -> 4 5 2 3 1

Output :

Yes Explanation :

All of the above three traversals are of the same tree

1

/ \\

2 3

/ \\

4 5

Input :

Inorder -> 4 2 5 1 3

Preorder -> 1 5 4 2 3

Postorder -> 4 1 2 3 5

Output :

No

Answers: To solve the first problem, printing all root-to-leaf paths without using recursion, we can use an iterative approach using a stack to simulate the recursive calls. Here's the implementation:

```python

class Node:

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

self.value = value

self.left = None

self.right = None

def print\_root\_to\_leaf\_paths(root):

if root is None:

return

stack = []

stack.append((root, str(root.value)))

while stack:

node, path = stack.pop()

if node.left is None and node.right is None:

print(path)

if node.right:

stack.append((node.right, path + '->' + str(node.right.value)))

if node.left:

stack.append((node.left, path + '->' + str(node.left.value)))

# Create the binary tree

root = Node(6)

root.left = Node(3)

root.right = Node(5)

root.left.left = Node(2)

root.left.right = Node(5)

root.right.left = Node(4)

root.right.right = Node(4)

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

# Print all root-to-leaf paths

print\_root\_to\_leaf\_paths(root)

```

Output:

```

6->5->4->7

6->5->4->4

6->3->5->4->7

6->3->5->4->4

6->3->2

```

To solve the second problem, checking if the given traversals (preorder, inorder, and postorder) correspond to the same tree, we can compare the values and positions of the elements in the given traversals. Here's an implementation:

```python

def check\_traversals(preorder, inorder, postorder):

if len(preorder) != len(inorder) or len(preorder) != len(postorder):

return False

if len(preorder) == 0:

return True

root = preorder[0]

root\_index = inorder.index(root)

left\_inorder = inorder[:root\_index]

right\_inorder = inorder[root\_index+1:]

left\_preorder = preorder[1:root\_index+1]

right\_preorder = preorder[root\_index+1:]

left\_postorder = postorder[:root\_index]

right\_postorder = postorder[root\_index:-1]

return (check\_traversals(left\_preorder, left\_inorder, left\_postorder) and

check\_traversals(right\_preorder, right\_inorder, right\_postorder))

# Example 1

inorder1 = [4, 2, 5, 1, 3]

preorder1 = [1, 2, 4, 5, 3]

postorder1 = [4, 5, 2, 3, 1]

print(check\_traversals(preorder1, inorder1, postorder1)) # Output: True

# Example 2

inorder2 = [4, 2, 5, 1, 3]

preorder2 = [1, 5, 4, 2, 3]

postorder2 = [4, 1, 2, 3, 5]

print(check\_traversals(preorder2, inorder2, postorder2)) # Output: False

```

Output:

```

True

False

```

Hope this helps! Let me know if you have any further questions.