Adamson University
College of Engineering
Computer Engineering Department

Data Structure and Algorithm

Laboratory Activity No.10

Nonlinear Data Structure - Tree

Submitted by:
Manigbas, Jeus Miguel T. <Leader>
Barredo, Alwin P.
Bela, Lorenzo Miguel D.
Callorina, Robert Victor A.
Guzon, Kean Louiz I.

Instructor:
Engr. Maria Rizette H. Sayo

December 01, 2004

I. Objectives

Introduction

An abstract non-linear data type with a hierarchy-based structure is a tree. It is made up of links connecting nodes (where the data is kept). The root node of a tree data structure is where all other nodes and subtrees are connected to the root.

This laboratory activity aims to implement the principles and techniques in:

- To introduce Tree as Non-linear data structure
- To implement pre-order, in-order, and post-order of a binary tree

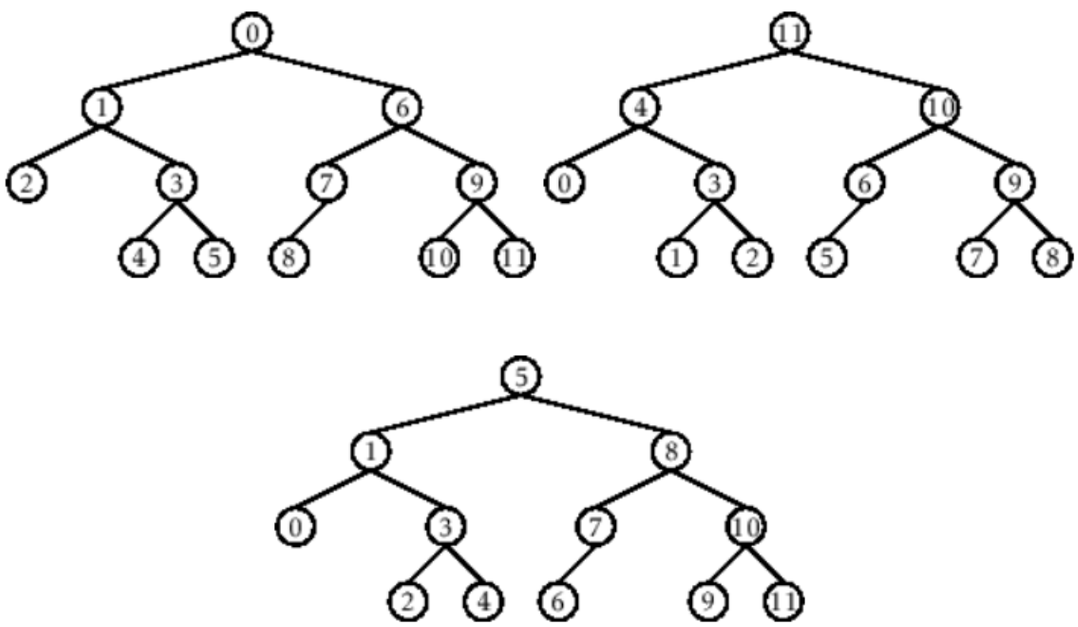

Figure 1. Pre-order, In-order, and Post-order numberings of a binary tree

II. Methods

Instruction: Create a subclass of BinaryTree whose nodes have fields for storing pre-order, post order, and in-order numbers. Write recursive methods preOrderNumber(), inOrderNumber(), postOrderNumbers() that assign these numbers correctly.

III. Results

```
class BinaryTreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
        self.pre_order = None
        self.in_order = None
        self.post_order = None

class BinaryTree:
    def __init__(self, root):
        self.root = BinaryTreeNode(root)

    def preOrderNumber(self, node, num=1):
        if node:
            node.pre_order = num
            num = self.preOrderNumber(node.left, num + 1)
            num = self.preOrderNumber(node.right, num)
        return num

    def inOrderNumber(self, node, num=1):
        if node:
            num = self.inOrderNumber(node.left, num)
            node.in_order = num
            num += 1
            num = self.inOrderNumber(node.right, num)
        return num
```

Figure 1.1: Code of binary tree

```
    def postOrderNumber(self, node, num=1):
        if node:
            num = self.postOrderNumber(node.left, num)
            num = self.postOrderNumber(node.right, num)
            node.post_order = num
            num += 1
        return num

tree = BinaryTree(1)
tree.root.left = BinaryTreeNode(2)
tree.root.right = BinaryTreeNode(3)
tree.root.left.left = BinaryTreeNode(4)
tree.root.left.right = BinaryTreeNode(5)
tree.root.right.left = BinaryTreeNode(6)
tree.root.right.right = BinaryTreeNode(7)

tree.preOrderNumber(tree.root)
tree.inOrderNumber(tree.root)
tree.postOrderNumber(tree.root)

def print_tree_numbers(node):
    if node:
        print(f"Node: {node.value}, Pre-order: {node.pre_order}, In-order: {node.in_order}, Post-order: {node.post_order}")
        print_tree_numbers(node.left)
        print_tree_numbers(node.right)

print_tree_numbers(tree.root)
```

Figure 1.2: Code of binary tree

```
Node: 1, Pre-order: 1, In-order: 4, Post-order: 7
Node: 2, Pre-order: 2, In-order: 2, Post-order: 3
Node: 4, Pre-order: 3, In-order: 1, Post-order: 1
Node: 5, Pre-order: 4, In-order: 3, Post-order: 2
Node: 3, Pre-order: 5, In-order: 6, Post-order: 6
Node: 6, Pre-order: 6, In-order: 5, Post-order: 4
Node: 7, Pre-order: 7, In-order: 7, Post-order: 5
```

Figure 1.3: Output of the code of binary tree

- **'BinaryTreeNode'** represents a node in the binary tree.

- Every node possesses a **'value'** and points to its left and right children.
- Extra fields (pre-order, in-order, post-order) are used to store the respective traversal numbers.
- ‘BinaryTree’ represents the binary tree itself.
- The **‘__init__’** initializes the tree with a root node.
- Three methods, preOrderNumber, inOrderNumber, and postOrderNumber, are created to go through the tree in pre-order, in-order, and post-order, giving each node a number based on the traversal.
- The methods use recursion and take a node and a number as inputs. The number helps keep track of the unique order assigned to each node.
- The **‘print_tree_numbers’** function is used with the tree's root, printing values and assigned numbers for each node in pre-order.

IV. Conclusion

Trees are a non-linear data structure that can be used to store and organize data. This activity implemented pre-order, in-order, and post-order traversals of a binary tree. The pre-order traversal visits the root node first, then the left subtree, and then the right subtree. The in-order traversal visits the left subtree, then the root node, and then the right subtree. The post-order traversal visits the left subtree, then the right subtree, and then the root node. This activity also created a subclass of BinaryTree to store pre-order, post-order, and in-order numbers for each node. The recursive methods preOrderNumber(), inOrderNumber(), and postOrderNumbers() were written to assign these numbers correctly. The print_numbers() method was written to initiate the printing of traversal numbers in the binary tree based on the specified type.

References

[1] A. Ioannou, “Trees,” *@xnorcode*, 12-Oct-2018. [Online]. Available: https://xnorcode.com/trees-non-linear-data-structure-part-one/. [Accessed: 04-Dec-2023].

[2] “Tree data structure,” *GeeksforGeeks*, 09-Jun-2023. [Online]. Available: https://www.geeksforgeeks.org/tree-data-structure/. [Accessed: 06-Dec-2023].

[3] “Tree data structure,” *Programiz.com*. [Online]. Available: https://www.programiz.com/dsa/trees. [Accessed: 07-Dec-2023].

[4] I. V. Abba, “Binary search tree traversal – inorder, preorder, post order for BST,” *freecodecamp.org*, 26-Jan-2022. [Online]. Available: https://www.freecodecamp.org/news/binary-search-tree-traversal-inorder-preorder-post-order-for-bst/. [Accessed: 07-Dec-2023].