UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 14

Tree Structure Analysis

Submitted by:
Ampong, J-kevin L.

Instructor:
Engr. Maria Rizette H. Sayo

November, 9, 2025

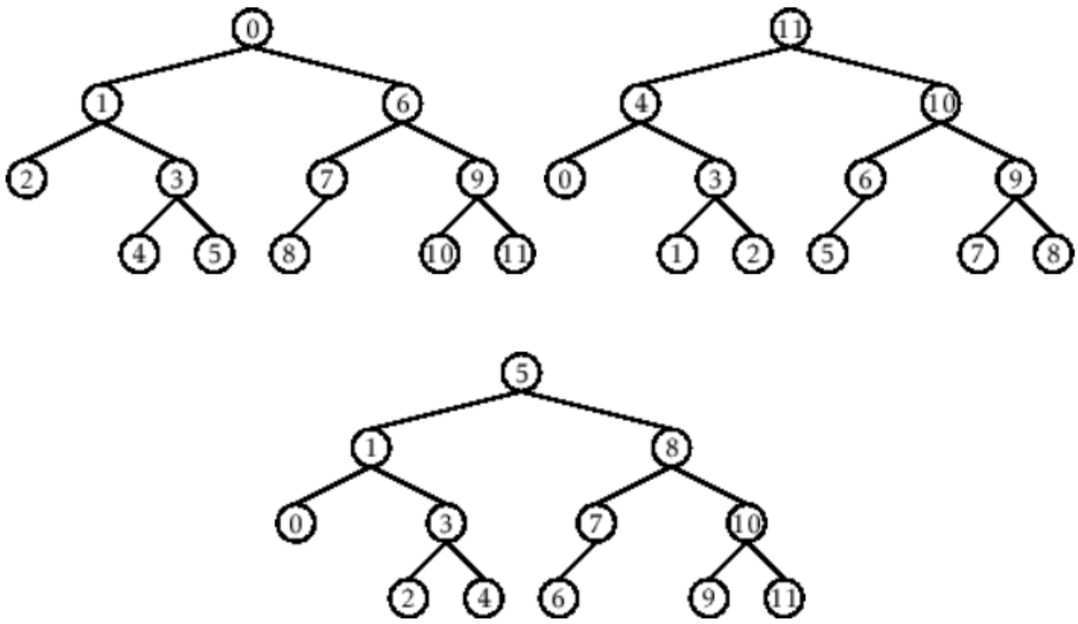
I. Objectives

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

- Copy and run the Python source codes.
- If there is an algorithm error/s, debug the source codes.
- Save these source codes to your GitHub.
- Show the output

1. Tree Implementation

```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.children = []

    def add_child(self, child_node):
        self.children.append(child_node)

    def remove_child(self, child_node):
        self.children = [child for child in self.children if child != child_node]
```

```
    def traverse(self):
        nodes = [self]
        while nodes:
            current_node = nodes.pop()
            print(current_node.value)
            nodes.extend(current_node.children)

    def __str__(self, level=0):
        ret = "  " * level + str(self.value) + "\n"
        for child in self.children:
            ret += child.__str__(level + 1)
        return ret

# Create a tree
root = TreeNode("Root")
child1 = TreeNode("Child 1")
child2 = TreeNode("Child 2")
grandchild1 = TreeNode("Grandchild 1")
grandchild2 = TreeNode("Grandchild 2")

root.add_child(child1)
root.add_child(child2)
child1.add_child(grandchild1)
child2.add_child(grandchild2)

print("Tree structure:")
print(root)

print("\nTraversal:")
root.traverse()
```

Questions:

1 What is the main difference between a binary tree and a general tree?
The primary difference between a general tree (like the one in your code) and a binary tree lies in the number of children a node can have. In a general tree, a node can have any number of children (zero, one, or many), which are often stored in a list or other dynamic collection. In a strict binary tree, each node can have at most two children, which are explicitly designated as a 'left' child and a 'right' child.

2 In a Binary Search Tree, where would you find the minimum value? Where would you find the maximum value?
In a Binary Search Tree (BST), values are organized by a specific property: all nodes to the left of a parent are smaller, and all nodes to the right are larger. To find the minimum value, you start at the root and continuously traverse as far left as possible; the last node you reach (the leftmost node) holds the minimum value. Conversely, to find the maximum value, you start at the root and continuously traverse as far right as possible; the rightmost node holds the maximum value.

3 How does a complete binary tree differ from a full binary tree?
A full binary tree is a tree where every node has exactly zero or two children. No node in a full binary tree has only one child. A complete binary tree, on the other hand, is a tree where all levels are completely filled except possibly the last level, and on that final level, all nodes are as far left as possible. This means a complete tree might have nodes with one child, but only on the second-to-last level.

4 What tree traversal method would you use to delete a tree properly? Modify the source codes.
To properly delete a tree and ensure all nodes are processed without orphaning any (which causes memory leaks), you must use a post-order traversal. This method ensures that a parent node is processed (i.e., deleted) only after all of its children have been processed and deleted. For the general tree class provided, this means recursively calling the "destroy" method on every child in the self.children list before processing the current node itself.

III. Results

```
Root
  Child 1
    Grandchild 1
  Child 2
    Grandchild 2


Depth-First Traversal:
Root
Child 2
Grandchild 2
Child 1
Grandchild 1

--- Deleting Tree (Post-Order Traversal) ---
Deleting node: Grandchild 1
Deleting node: Child 1
Deleting node: Grandchild 2
Deleting node: Child 2
Deleting node: Root

After deletion, root's children: []
Root variable set to: None
```

IV. Conclusion

The core takeaway from the provided code and your questions is the distinction between general trees and binary trees, and the importance of traversal methods in managing tree data structures.

The TreeNode class you provided represents a general tree where nodes can have multiple children. Your questions focused on the key theoretical differences, such as the limit of two children in a binary tree versus a general tree. Crucially, your exploration of deletion highlighted that a post-order traversal is necessary to properly delete or clear a tree by ensuring all children are processed before their parent, thereby preventing memory leaks.

References

[1] Co Arthur O.. “University of Caloocan City Computer Engineering Department Honor Code,” UCC-CpE Departmental Policies, 2020.