UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 14

Tree Structure Analysis

Submitted by:
Mark Angel T. Talagtag

Instructor:
Engr. Maria Rizette H. Sayo

November, 9, 2025

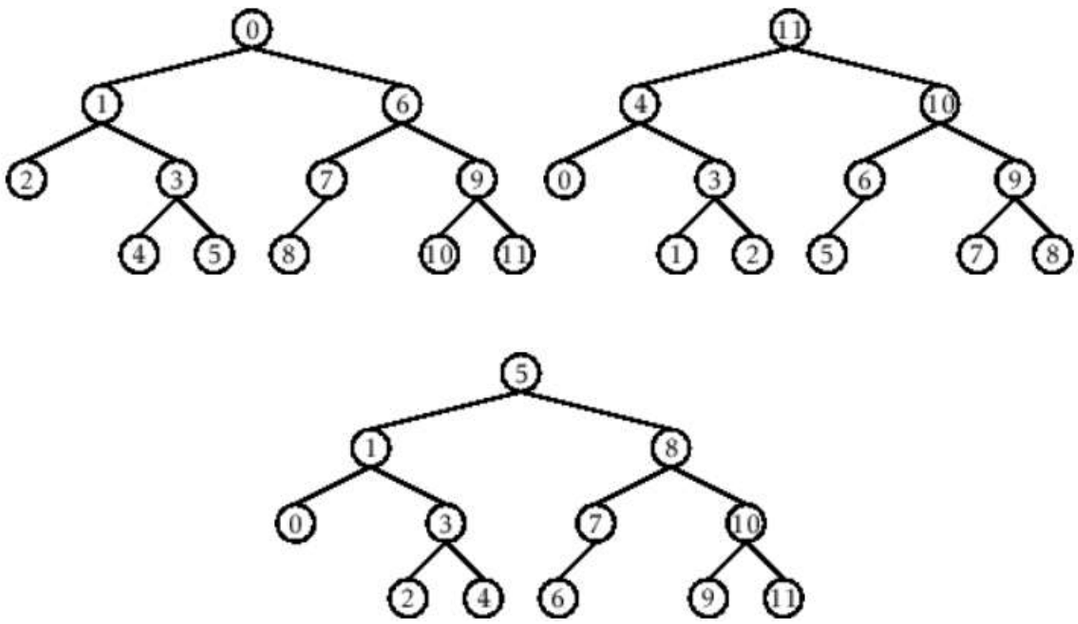
I. Objectives

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?
2. In a Binary Search Tree, where would you find the minimum value? Where would you find the maximum value?
3. How does a complete binary tree differ from a full binary tree?
4. What tree traversal method would you use to delete a tree properly? Modify the source codes.

III. Results

```
Tree structure:
Root
  Child 1
    Grandchild 1
  Child 2
    Grandchild 2


Traversal:
Root
Child 2
Grandchild 2
Child 1
Grandchild 1
```

1. The primary distinction between a binary tree and a general tree is that a binary tree restricts each node to having at most two children, referred to as the left and right child. In contrast, a general tree allows a node to have multiple children, offering greater flexibility but making traversal and manipulation more complex.

2. In a Binary Search Tree (BST), the smallest value can be located by continuously moving to the leftmost node starting from the root, while the largest value can be found by traversing to the rightmost node until no further right child exists.

3. A complete binary tree is one in which all levels are fully filled, except possibly the last level, which is filled from left to right. Meanwhile, a full binary tree is defined by every node having either exactly two children or none at all — no node can have only one child.

4. To properly delete a tree, post-order traversal is used since it ensures that all child nodes are deleted before their parent node, preventing dangling references or memory issues.

Modified Code Example:

```
def delete_tree(node):
    if node:
        for child in node.children:
            delete_tree(child)
        print(f"Deleting node: {node.value}")
        del node
Usage:
delete_tree(root)
```

IV. Conclusion

Through this activity, I learned how to create and visualize tree structures using Python. I was able to understand the hierarchical connections between nodes and the functionality of different types of trees, including binary and general trees. I also gained a clearer grasp of traversal methods such as pre-order, in-order, and post-order, and how these techniques are used in operations like searching and deletion. In summary, this laboratory task enhanced my comprehension of trees as essential non-linear data structures that enable efficient data organization and processing.

References

[1] GeeksforGeeks, “Difference Between General Tree and Binary Tree,” *GeeksforGeeks Data Structures Tutorial*, 2025. Available: https://www.geeksforgeeks.org/difference-between-general-tree-and-binary-tree/

[2] TutorialsPoint, “Tree Traversal Techniques in Data Structures,” *TutorialsPoint Computer Science Resources*, 2024. Available: https://www.tutorialspoint.com/data_structures_algorithms/tree_traversal.htm