UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 14

Tree Structure Analysis

Submitted by:
Dispo, Lei Andrew T.

Instructor:
Engr. Maria Rizette H. Sayo

11, 09, 2025

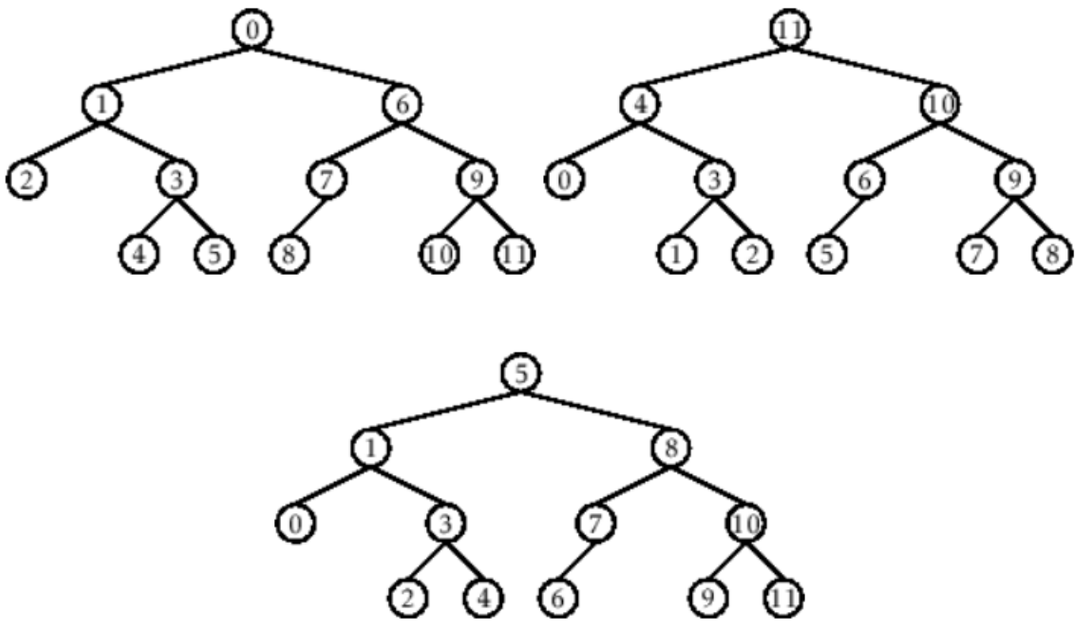
I. Objectives

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

1. Each node in a binary tree, a hierarchical data structure, may have a maximum of two offspring, commonly known as the left and right children. On the other hand, each node in a generic tree may have any number of offspring. Because of this, general trees are more adaptable for describing hierarchical data, but binary trees are more organized and appropriate for tasks like binary search.
2. Because smaller values are always positioned to the left, the minimal value in a Binary Search Tree (BST) is at the node on the left. Since larger values are always found to the right, the maximum value is located at the node on the right.
3. A complete binary tree is a type of binary tree where all levels are fully filled, except for the last level, which is filled sequentially from left to right. In contrast, a full binary tree is one where each node has either two children or none at all, meaning no node has only a single child.

4 The proper traversal method to delete a tree is post-order traversal, because it ensures that child nodes are deleted before their parent nodes. This prevents memory leaks and errors caused by deleting a parent before its children.

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

IV. Conclusion

In this laboratory activity, the concept and implementation of tree structures were explored, including their types and traversal methods. Through coding and analysis, the differences between general, binary, complete, and full binary trees were understood, along with the proper method for deleting a tree using post-order traversal. This activity enhanced understanding of hierarchical data organization and its importance in efficient data processing.

References

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