UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 14

Tree Structure Analysis

Submitted by:
Adoracion, Jerick Dave D.

Instructor:
Engr. Maria Rizette H. Sayo

November 11, 2025

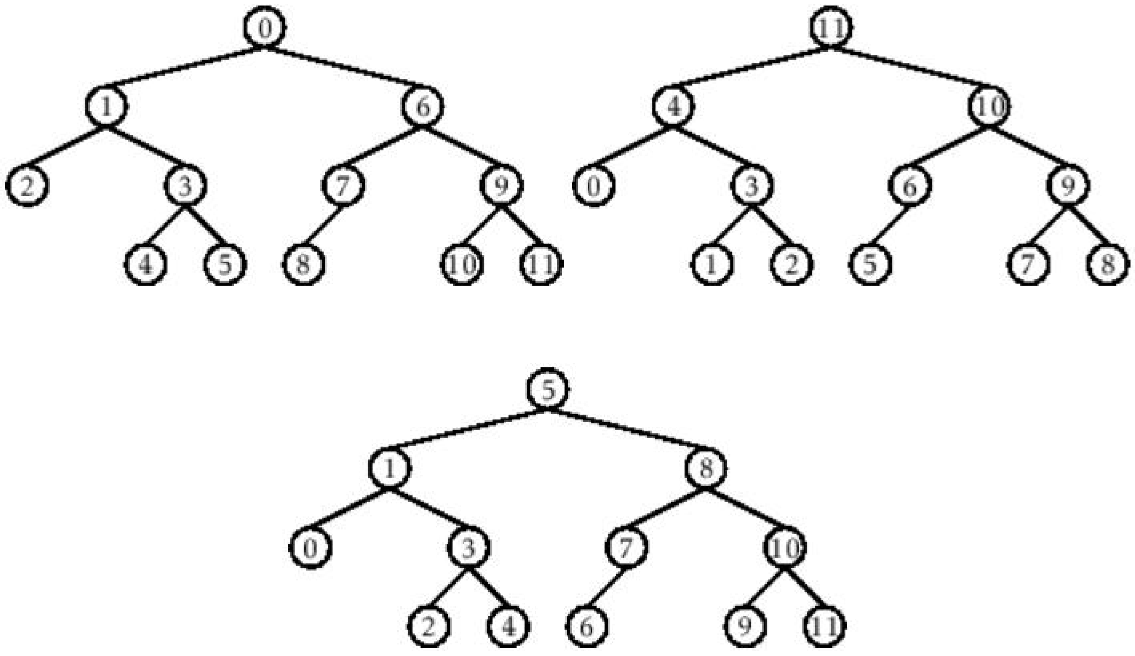
I. Objectives

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. What is the main difference between a binary tree and a general tree?

The main difference between a binary tree and a general tree lies in the number of children each node can have. In a binary tree, every node can have at most two children, commonly referred to as the left and right child. This restriction makes binary trees suitable for structured operations such as searching, sorting, and expression representation. In contrast, a general tree allows each node to have any number of children, making it more flexible but less structured. General trees are often used to represent hierarchical data such as organizational charts or file systems.

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), the placement of nodes follows a specific order where the left child of a node contains a smaller value, and the right child contains a larger value. Therefore, the minimum value in a BST can be found by traversing as far left as possible from the root node until reaching the last left child, which holds the smallest key. Similarly, the maximum value can be found by moving as far right as possible from the root node until the last right child is reached. This property of BSTs allows efficient searching for both the smallest and largest values.

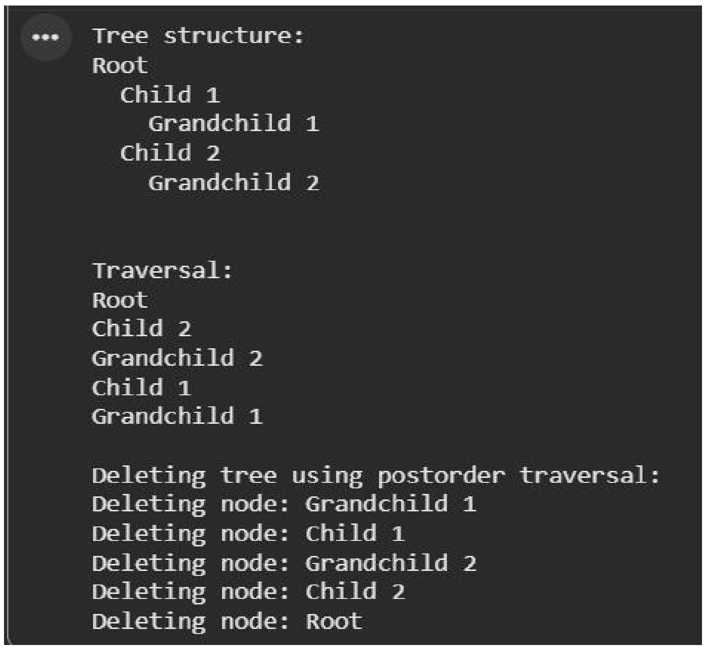
3,How does a complete binary tree differ from a full binary tree?

A complete binary tree and a full binary tree differ in their structure and how nodes are arranged. A complete binary tree is one in which all levels are fully filled except possibly the last level, and all nodes on the last level are aligned to the left. On the other hand, a full binary tree is defined as a tree where every node has either zero or two children, meaning no node has only one child. While both types of trees have a balanced appearance, their definitions emphasize different structural properties,completeness focuses on filling levels, while fullness focuses on the number of children per node.

4.What tree traversal method would you use to delete a tree properly? Modify the source codes.

To **properly delete a tree**, we should use **postorder traversal,** meaning you delete all the children of a node **before** deleting the node itself.

This ensures that no node is deleted before its descendants, preventing memory access or reference issues.



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


Traversal:
Root
Child 2
Grandchild 2
Child 1
Grandchild 1

Deleting tree using postorder traversal:
Deleting node: Grandchild 1
Deleting node: Child 1
Deleting node: Grandchild 2
Deleting node: Child 2
Deleting node: Root
```

Figure 2: Output

IV. Conclusion

In conclusion, learning about tree data structures and their traversal methods has been an essential part of my growth as a computer engineering student. Understanding the differences between general trees, binary trees, and their special forms such as complete and full binary trees has strengthened my ability to organize and manage data efficiently. Studying Binary Search Trees has taught me how to locate minimum and maximum values quickly, and implementing traversal algorithms has enhanced my programming and problem-solving skills. These concepts are fundamental for many areas in computer engineering, including databases, file systems, and network structures.

Furthermore, learning how to properly delete a tree using postorder traversal has given me practical insights into memory management and resource handling. It has shown me the importance of systematically processing data structures to avoid errors and maintain program stability. This experience has reinforced the value of algorithmic thinking and careful planning in software development. Overall, mastering tree structures and traversal techniques has not only improved my technical knowledge but also prepared me to tackle more complex problems in my future career as a computer engineer.

References

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

[2] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Boston, MA: Addison-Wesley, 2011.

[3] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Hoboken, NJ: Wiley, 2018.

[4] E. Horowitz, S. Sahni, and S. Rajasekaran, *Fundamentals of Computer Algorithms*, 2nd ed. Rockville, MD: Computer Science Press, 2008.

[5] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, 2nd ed. Boston, MA: Pearson, 2007.