UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 14

Tree Structure Analysis

Submitted by:
Catahan, Joshua A.

Instructor:
Engr. Maria Rizette H. Sayo

11, 10, 2025

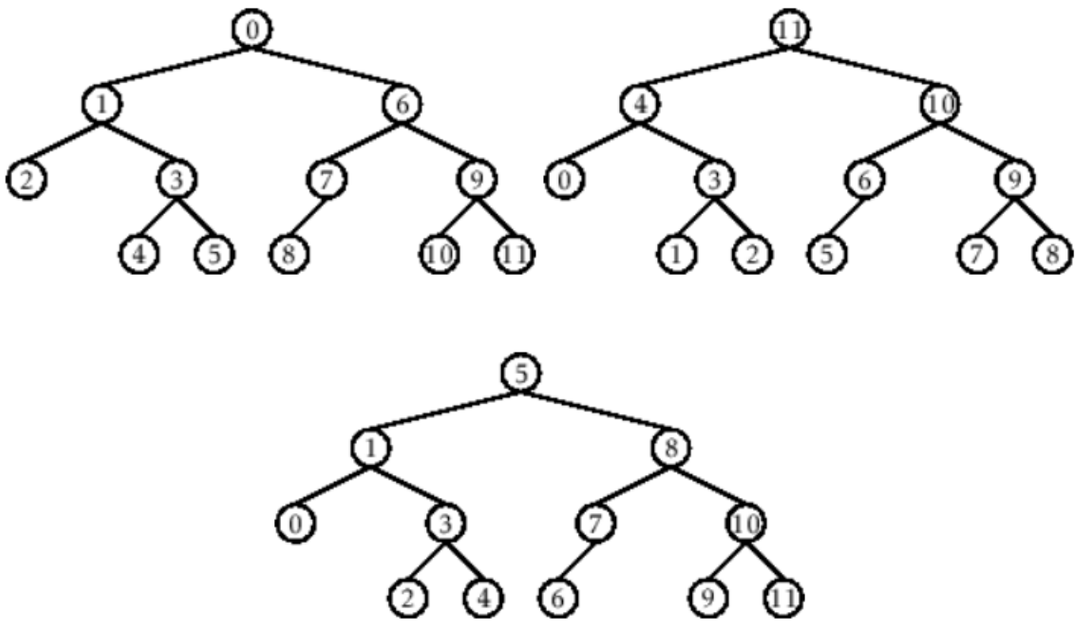
I. Objectives

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

When I first ran the original program, the tree structure printed correctly, and the traversal worked, but there was no method to properly delete the tree. I added a delete_tree() method using post-order traversal, which deletes all children before deleting the parent node. This ensures that nodes are removed in a safe order, preventing any dangling references. After this change, running the program not only printed the tree structure and traversal correctly but also showed the deletion of each node in the proper order.

```
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):
        # Pre-order traversal
        nodes = [self]
        while nodes:
            current_node = nodes.pop()
            print(current_node.value)
            nodes.extend(reversed(current_node.children))  # ensure left-to-right order

    def delete_tree(self):
        # Post-order traversal: delete children first
        for child in self.children:
            child.delete_tree()
        print(f"Deleting node: {self.value}")
        self.children = []
        self.value = None

    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("Traversal:")
root.traverse()

print("\nDeleting tree:")
root.delete_tree()
```

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

Traversal:
Root
Child 1
Grandchild 1
Child 2
Grandchild 2

Deleting tree:
Deleting node: Grandchild 1
Deleting node: Child 1
Deleting node: Grandchild 2
Deleting node: Child 2
Deleting node: Root
```

Figure 1: Screenshot of the Program

1. Main difference between a binary tree and a general tree:

A binary tree allows at most two children per node, which gives it a structured form suitable for searching and traversal. A general tree allows any number of children, making it more flexible but less strict in its organization.

2. Minimum and maximum in a Binary Search Tree (BST):

The minimum value is found at the leftmost node because smaller values are always placed to the left. The maximum value is found at the rightmost node, where larger values are placed.

3. Complete binary tree vs full binary tree:

A full binary tree has nodes with either 0 or 2 children, ensuring uniform branching. A complete binary tree fills all levels completely except possibly the last, which is filled from left to right, keeping the tree compact.

4. Traversal method for deleting a tree:

Post-order traversal is used because it visits all children before the parent, allowing the tree to be deleted safely without leaving any dangling references.

IV. Conclusion

When I completed this laboratory activity, I was able to implement and analyze a tree as a non-linear data structure. The tree structure and traversal methods worked as expected, and by adding the delete_tree() method using post-order traversal, I ensured that nodes could be removed safely without leaving any dangling references. This activity also helped me understand the differences between binary and general trees, locate minimum and maximum values in a Binary Search Tree, and distinguish between complete and full binary trees. Overall, the lab reinforced the principles of tree structures and traversal methods in a clear and practical way.

References

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