UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 14

Tree Structure Analysis

Submitted by:
Sorellano, John Kenneth T.

Instructor:
Engr. Maria Rizette H. Sayo

November 9, 2025

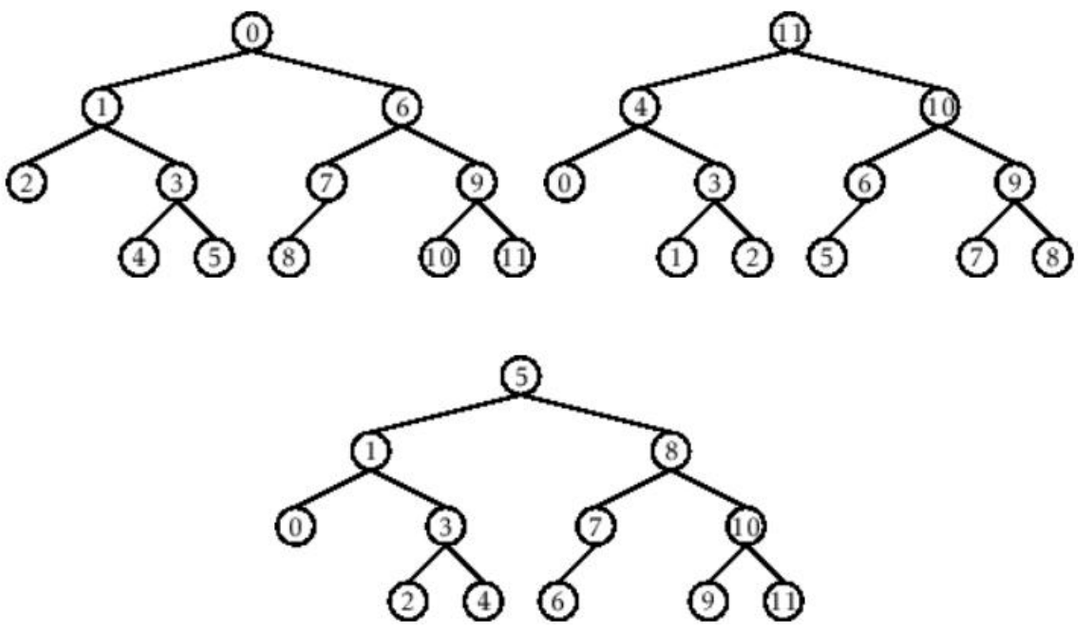
I. Objectives

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

Questions:

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

- Every node in a binary tree, which is a hierarchical data structure, can possess a maximum of two descendants, typically referred to as the left and right offspring. In every node in A generic tree can have an arbitrary number of descendants. As a result, general trees are increasingly suitable for representing hierarchical information, although binary trees offer more structure and suitable for activities such as binary search.

2. In a Binary Search Tree, where would you find the minimum value? Where would you find the maximum value?

- Since lesser values are consistently placed on the left, the smallest value in a Binary The left node contains the Binary Search Tree (BST). As larger values are consistently observed to the correct, the highest value is found at the node to the right.

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

- A complete binary tree is a form of binary tree in which every level is entirely filled, except for the final level, which is populated in order from left to right. In comparison, a complete binary tree is one in which every node has either two offspring or none at all, indicating that no node has solely a only child.

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

- The correct way to delete a tree is by using post-order traversal, as it guarantees that child nodes are removed prior to their parent nodes. This avoids memory leaks and mistakes, resulting from the deletion of a parent before its child.

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

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


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

IV. Conclusion

In conclusion, the idea and execution of tree structures were examined, such as their categories and ways to navigate through them. Using programming and examination, the variations among general, binary, complete, and full binary trees were comprehended, along with the correct approach for removing a tree by utilizing post-order traversal. This task improved comprehension of hierarchical data. structure and its significance in effective data management.

References

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