UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 14

Tree Structure Analysis

Submitted by:
Uy, Junichiro H.

Instructor:
Engr. Maria Rizette H. Sayo

November 09, 2025

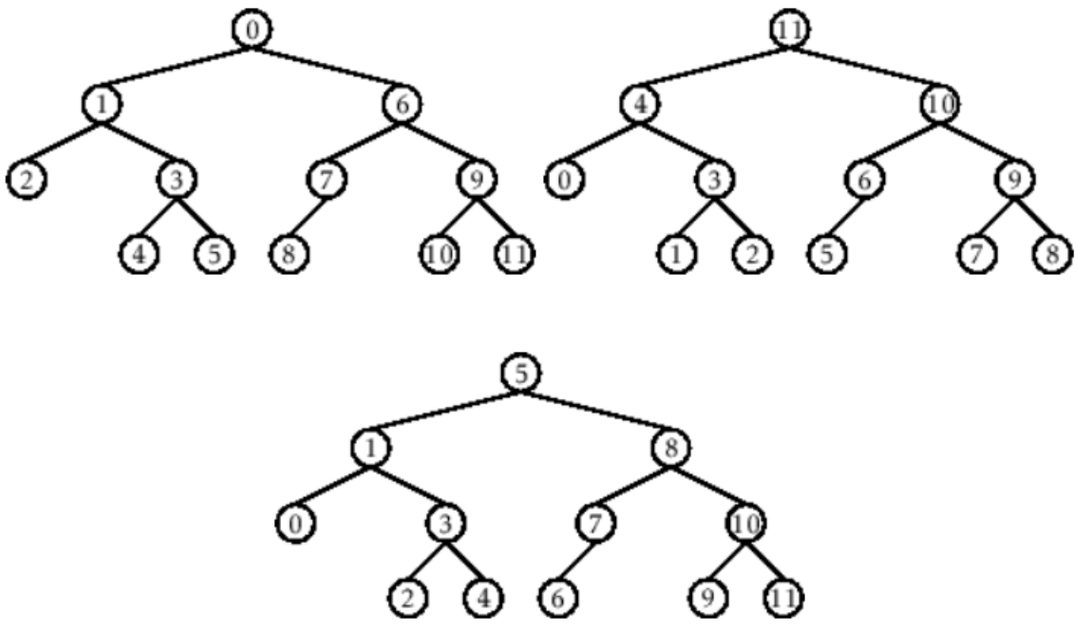
I. Objectives

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

Questions:

1. What is the main difference between a binary tree and a general tree?
   - The main difference is the number of child each node can have. A binary tree's node can only have 2 childrens, left and right. While a general tree's nodes can have 0 to many childrens.
2. In a Binary Search Tree, where would you find the minimum value? Where would you find the maximum value?
   - From what I searched, in a binary search tree, the minimum value are on the leftmost side and the maximum are on the right most side.

3. How does a complete binary tree differ from a full binary tree?
   - A Full binary tree can only have a node with 0 or 2 childrens, meaning having 1 children is not allowed. While a complete binary must all be filled, only 2 nodes left and right not 1 or 0, all levels should be filled, except possibly the last layer.
4. What tree traversal method would you use to delete a tree properly? Modify the source codes.
   - We will be using Post-order to delete it every done and the tree. Here's the code:

```
def delete_tree(self):

      print(f"Checking node: {self.value}")


      for child in list(self.children):
          child.delete_tree()
          self.remove_child(child)

      print(f"Deleting node: {self.value}")

      self.value = None
```

Output:

Tree structure before deletion:

Root

  Child 1

    Grandchild 1

  Child 2

    Grandchild 2

--- Deleting Tree ---

Checking node: Root

Checking node: Child 1

Checking node: Grandchild 1

Deleting node: Grandchild 1

Deleting node: Child 1

Checking node: Child 2

Checking node: Grandchild 2

Deleting node: Grandchild 2

Deleting node: Child 2

Deleting node: Root

Tree structure after deletion:

None

IV. Conclusion

This laboratory activity successfully introduced trees as a non-linear data structure. Through the implementation of a Python `Tree Node` class, we demonstrated the creation of a general tree and a depth-first traversal method . The exercise also clarified key theoretical differences between binary and general trees , Binary Search Trees , and full vs. complete binary trees. We also got to use post-order to delete nodes and trees. Overall, this lab provided valuable hands-on experience with the practical implementation and theory of tree structures.

References

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