UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 14

Tree Structure Analysis

Submitted by:
Acebedo, Sebastian C.

Instructor:
Engr. Maria Rizette H. Sayo

November 9, 2025

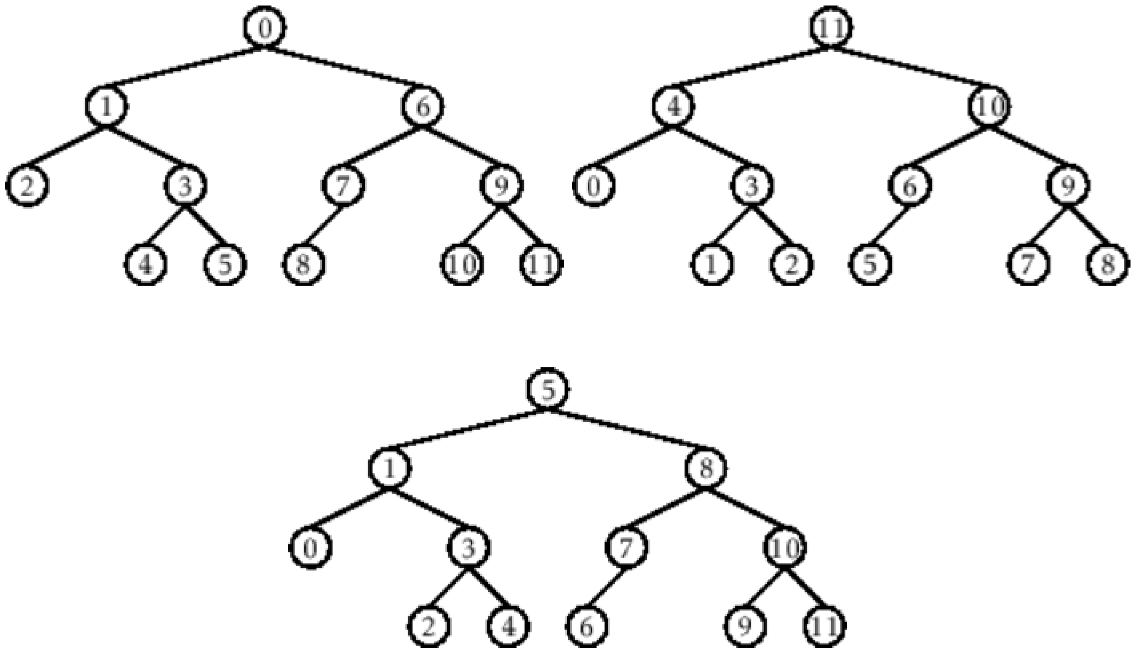
I. Objectives

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

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 is that in a binary tree, each node can only have up to two children, usually called the left and right child. But in a general tree, a node can have any number of children. So basically, binary trees are more organized and limited, while general trees are more flexible and can have many branches.

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

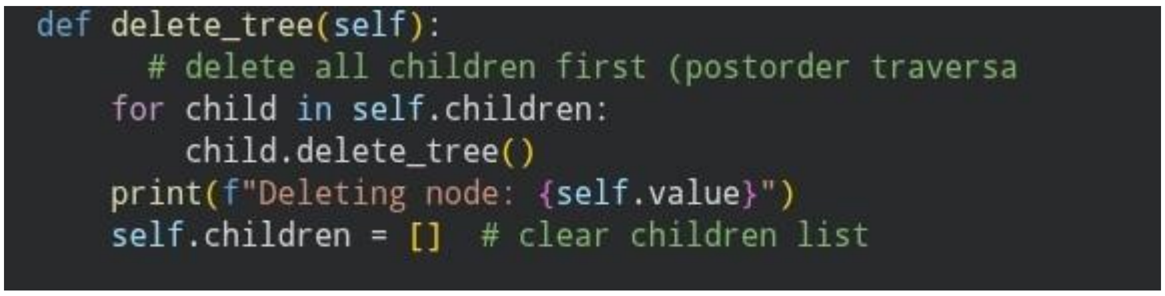
- In a Binary Search Tree (BST), the minimum value is found at the leftmost node, because all smaller values are stored on the left side.
- The maximum value is found at the rightmost node, since all larger values are stored on the right side of the tree.

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

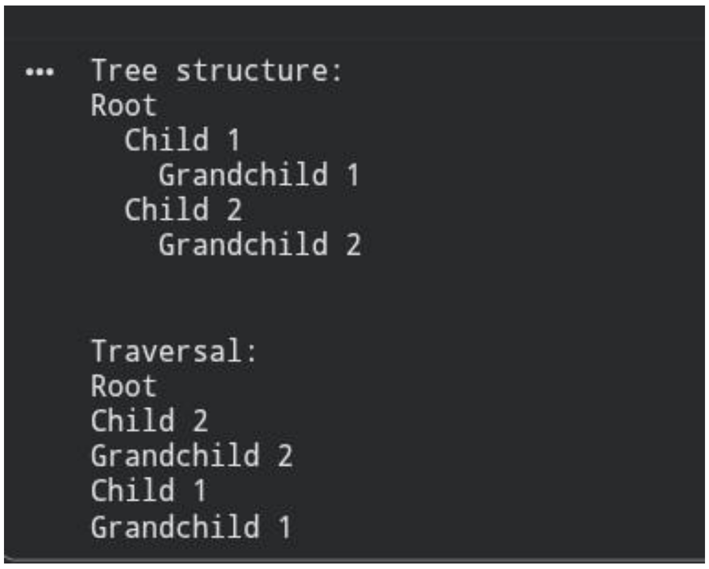
- A complete binary tree is different from a full binary tree because in a complete binary tree, all levels are filled except maybe the last one, and the nodes are as far left as possible. But in a full binary tree, every node has either 0 or 2 children. No node has only one child.

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

- I added the delete_tree() method which uses postorder traversal. It deletes all children first before deleting the parent node. This makes sure the tree is properly deleted without leaving any child nodes behind.



```
def delete_tree(self):
      # delete all children first (postorder traversa
    for child in self.children:
        child.delete_tree()
    print(f"Deleting node: {self.value}")
    self.children = []  # clear children list
```



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


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

Figure 1.2 Screenshot of output code

IV. Conclusion

In this laboratory activity, I learned that a tree is a non-linear data structure that organizes data in a hierarchical form using nodes and links. I was able to understand how each node connects back to a single root, forming parent-child relationships. Through implementing different tree traversal methods: preorder, inorder, and postorder. I learned how data in a tree can be accessed and processed in various ways depending on the order of traversal. Overall, this activity helped me understand the importance of trees in organizing and managing data efficiently..

References

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