UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 14

Tree Structure Analysis

Submitted by:
Enverzo, Kyle Andrey D.

Instructor:
Engr. Maria Rizette H. Sayo

November 8, 2025

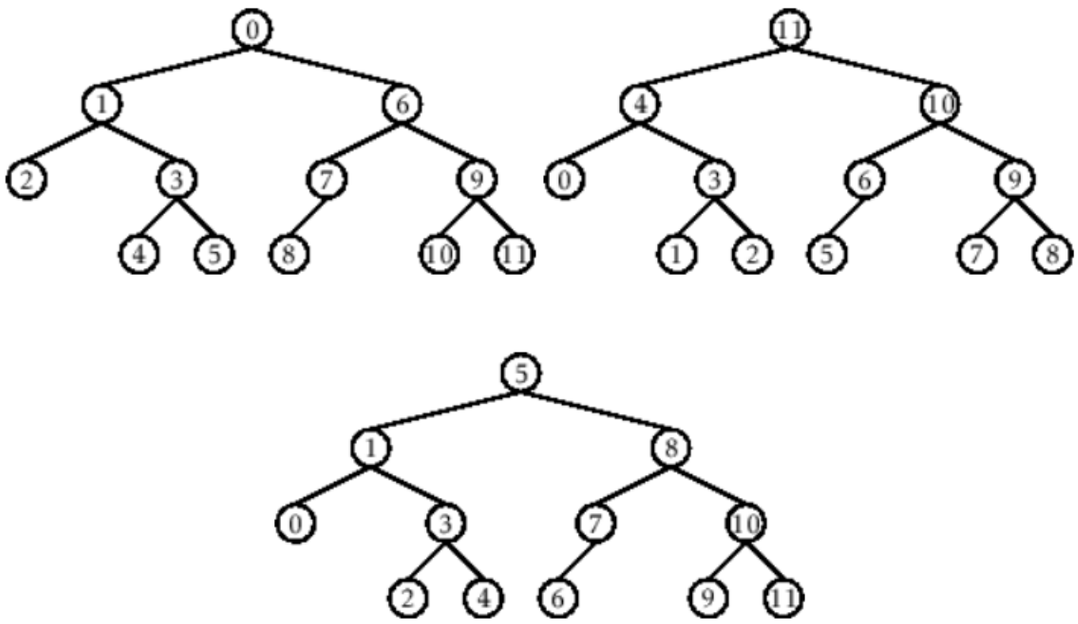
I. Objectives

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

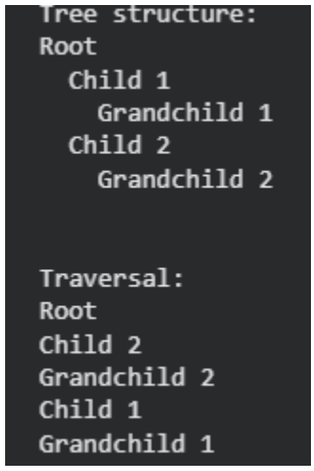


Figure 1 Screenshot of program

ANSWER

1. A binary tree and a general tree differ mainly in such a way that a binary tree is a tree in which each node has at most two children, the left and the right ones being the usual terms for them. On the other hand, a general tree means that each node can have the maximum number of children that the tree can accommodate. The Python code shown represents the TreeNode class which reflects a general tree as it is capable of having an unlimited number of children which are represented in a list.

2. Always the leftmost node in a Binary Search Tree (BST) contains the minimum value, since on the left side of the tree smaller values are stored. On the other hand, the rightmost node contains the maximum value because larger values are stored on the right side. Hence, you will arrive at the smallest value if you keep traversing the left child pointers and you will arrive at the largest value if you keep traversing the right child pointers.

3. A complete binary tree is a tree in which every node has either zero or two children, that is, no node has one child only, which is the case with a full binary tree. Moreover, a complete binary tree has all levels entirely filled except for the last one which is filled from left to right without gaps. In a nutshell, a full binary tree takes into account the number of children per node, whereas a complete binary tree considers the distribution of nodes across the levels.

4. The recommended tree traversal method for tree deletion is postorder traversal method. It is because the algorithm deletes the child nodes first and then deletes the parent node in postorder traversal. If the parent node is deleted first, its children will become inaccessible. In the revised Python code, the delete_tree() function implements postorder traversal by deleting all children

recursively before removing the current node, thereby making sure the whole tree is cleared correctly.

IV. Conclusion

To sum up, the various kinds of trees and their characteristics must be comprehended in computer science, particularly in data structure implementation and algorithm optimization. The main difference between a binary tree and a general tree is the number of children per node, with binary trees restricting it to two and general trees permitting more than two. In a Binary Search Tree (BST), the leftmost node has the minimum value and the rightmost has the maximum value. Furthermore, the difference between a complete and full binary tree is crucial for tree balance and storage efficiency. Also, postorder traversal is the best method for deleting a tree since it first deletes child nodes and then proceeds to parent nodes, thus maintaining data integrity and preventing memory leaks. All in all, these principles are the basis for more advanced data structures and efficient algorithm design.

References

[1] M. A. Weiss, *Data Structures and Algorithm Analysis in C++*, 4th ed. Boston, MA, USA: Pearson, 2014.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[3] S. Lipschutz, *Data Structures with C*, New York, NY, USA: McGraw-Hill, 2011.

[4] N. Wirth, *Algorithms + Data Structures = Programs*, Englewood Cliffs, NJ, USA: Prentice-Hall, 1976.

[5] M. Goodrich, R. Tamassia, and M. Goldwasser, *Data Structures and Algorithms in Python*, Hoboken, NJ, USA: John Wiley & Sons, 2013.