UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 14

Tree Structure Analysis

Submitted by:
Ruperto, April Anne A.

Instructor:
Engr. Maria Rizette H. Sayo

November 7, 2025

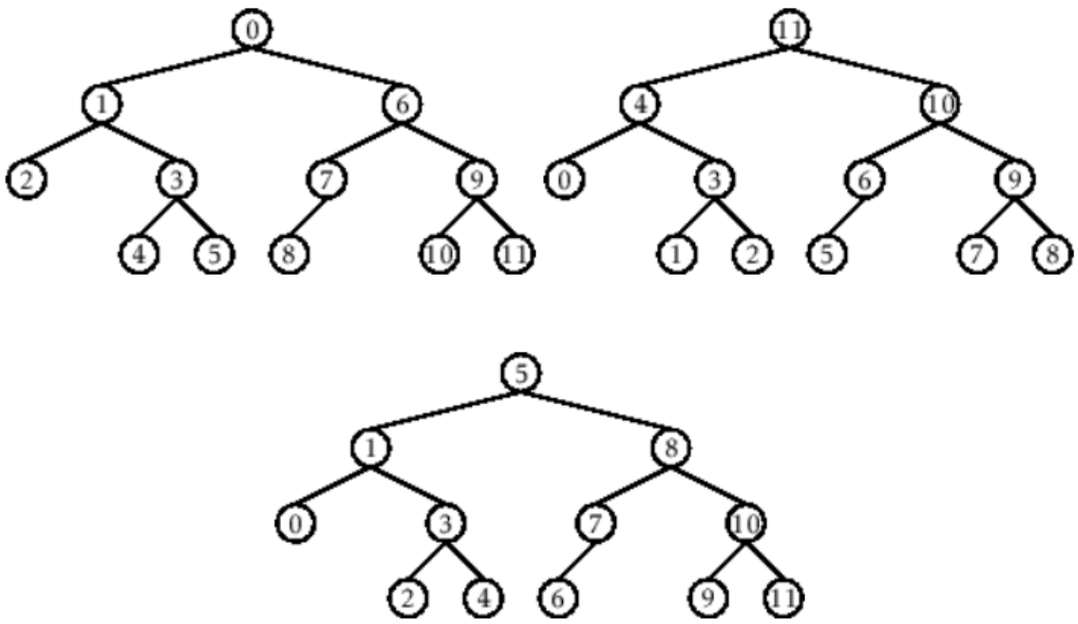
I. Objectives

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

Please follow this link: https://github.com/Ruperto-April-Anne/CPE-201L-DSA-2-A/DSA_Lab14.ipynb

Answer:

1. The main difference between a binary tree and a general tree is how many children a node can have. A binary tree allows a maximum of two children for any node, which are usually called the left and right child. While a general tree, on the other hand, can have any number of children for a single node, thus it is more versatile though harder to manage. Generally, binary trees find their application in data storage and searching, whereas general trees can represent hierarchical data such as file systems or organizational charts.

2. The minimum value in a Binary Search Tree (BST) can be located at the leftmost node, as smaller values are always inserted in the left subtree. On the other hand, the maximum value can be identified at the rightmost node, since larger values are placed in the right subtree. Thus, to find the minimum or maximum value in a BST, one needs to keep going to the left or right child until a leaf node is reached.
3. A complete binary tree is defined as a binary tree in which all levels, except possibly the last, are filled, and the nodes in the last level are as left as possible. Whereas a full binary tree is a tree in which each node has either 0 or 2 children, i.e., no node has only one child. It follows from this that full binary trees are not necessarily complete and complete binary trees are not necessarily full. The difference between them is in how nodes are spread over levels and whether every node has two children.
4. The best way to delete a tree is by using the post-order traversal method. This method first goes down the left subtree, then the right subtree, and finally visits the root node. In this way, the deletion of child nodes preceding their parent is guaranteed.

IV. Conclusion

In conclusion, knowing the different kinds of trees and their structures is a must if one wants to be able to efficiently organize and manipulate data in computer science. Binary trees, being limited to having only two children, are a simpler and quicker way of handling data. This is mostly evident in Binary Search Trees were looking for the minimum and maximum values becomes very easy. Where general trees have more advantages when it comes to depicting complicated hierarchies. Being aware of these differences let developers decide which structure is the best for solving problems scenario.

Having knowledge of such variations as full and complete binary trees helps one to better understand tree organization and the efficient use of it. The correct use of traversal techniques, like post-order traversal when deleting, makes it possible that data are handled in a safe and efficient way without mistakes. In general, understanding tree structures and their operations is important if one wants to be able to create efficient algorithms and have better problem-solving skills in programming and data management.

References

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