



UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT



Data Structure and Algorithm

Laboratory Activity No. 14

Tree Structure Analysis

Submitted by:
Disomnong, Jalilah M.

Instructor:
Engr. Maria Rizette H. Sayo

November 6, 2025

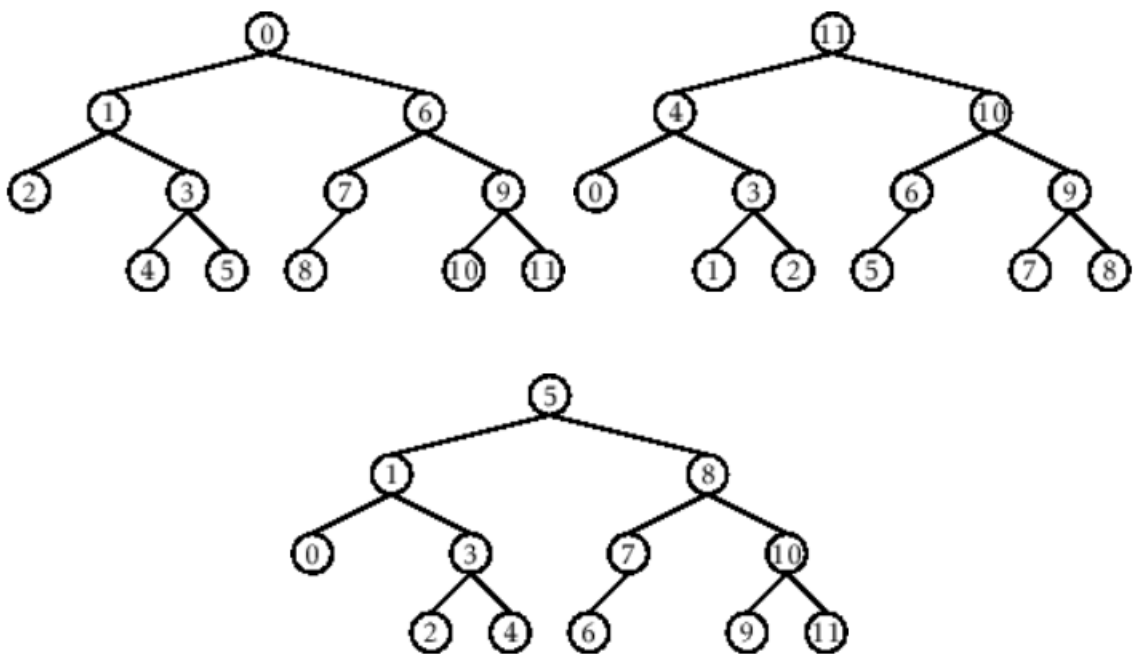
I. Objectives

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]
✓ 0s
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()
```

Figure 1 Screenshot of sourcecode link: [LAB 14 - Colab](#)

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

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

Figure 2 Screenshot of output link: [LAB 14 - Colab](#)

In question 1, the main difference between a binary tree and a general tree is how many children each node may have and the ordering/structure constraints. In a general tree, each node can have zero or many children, meaning there is no upper bound fixed at two; the structure is flexible and you may have any number of branches at each node. In contrast, a binary tree restricts each node to at most two children (typically called the left child and the right child). Furthermore, the subtrees in a binary tree are ordered (i.e., left vs right) whereas in a general tree the children are typically unordered with no inherent “left” or “right” distinction. According to GeeksforGeeks,

a general tree has no limitation on the degree of a node, the subtree of a general tree is unordered, while a binary tree can have at most two nodes and subtree of a binary tree is ordered.

In question 2, in the case of a Binary Search Tree (BST), the minimum value is found by starting at the root and continuously following the left child pointers until you reach a node that has no left child; that node will contain the minimum value in the tree. Conversely, the maximum value is found by starting at the root and continuously traversing the right child pointers until you reach a node that has no right child; that node holds the maximum value. This holds because in a BST all left subtree values are smaller than the node, and all right subtree values are greater.

In question 3, the difference between a complete binary tree and a full binary tree is about how full the levels are and how many children nodes have. A full binary tree means every node has either two children or none (i.e., zero children), so no node has exactly one child. A complete binary tree, on the other hand, is filled level by level from the left, and all levels are filled except possibly the last level, which is filled from the left as well — though nodes in the last level may not have two children. So full focuses on children-count per node, while complete focuses on how densely and orderly the tree is filled.

In question 4, to properly delete a tree, the traversal method you would use is post-order traversal (i.e., process all children first, then the parent). This ensures that you delete or free all descendant nodes before you delete the node itself, thereby avoiding orphaned references or memory leaks.

IV. Conclusion

In this laboratory activity, I was able to explore and understand the concept of tree structures in data structures and algorithms. By implementing the `TreeNode` class in Python by using Google Colab, I observed how nodes are connected in a hierarchical manner, starting from a single root and branching out into child nodes. Through the process of adding children, displaying the tree structure, and performing traversal, I gained a clearer view of how trees operate as non-linear data structures. Answering the guide questions further helped me distinguish different types of trees, such as binary trees, general trees, complete binary trees, and full binary trees. I also learned how Binary Search Trees manage values and how traversal methods like post-order are used in operations such as deleting nodes correctly.

References

- [1] Co Arthur O.. "University of Caloocan City Computer Engineering Department Honor Code," UCC-CpE Departmental Policies, 2020.
- [2] GeeksforGeeks. (n.d.). *Difference between General tree and Binary tree*. Retrieved from <https://www.geeksforgeeks.org/difference-between-general-tree-and-binary-tree/>
- [3] Google. (2024). Google Colaboratory. Retrieved April 18, 2024, from <https://colab.research.google.com/>