UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 13

Tree Algorithm

Submitted by:
Barbas, Steven Jade P.

Instructor:
Engr. Maria Rizette H. Sayo

November, 9, 2025

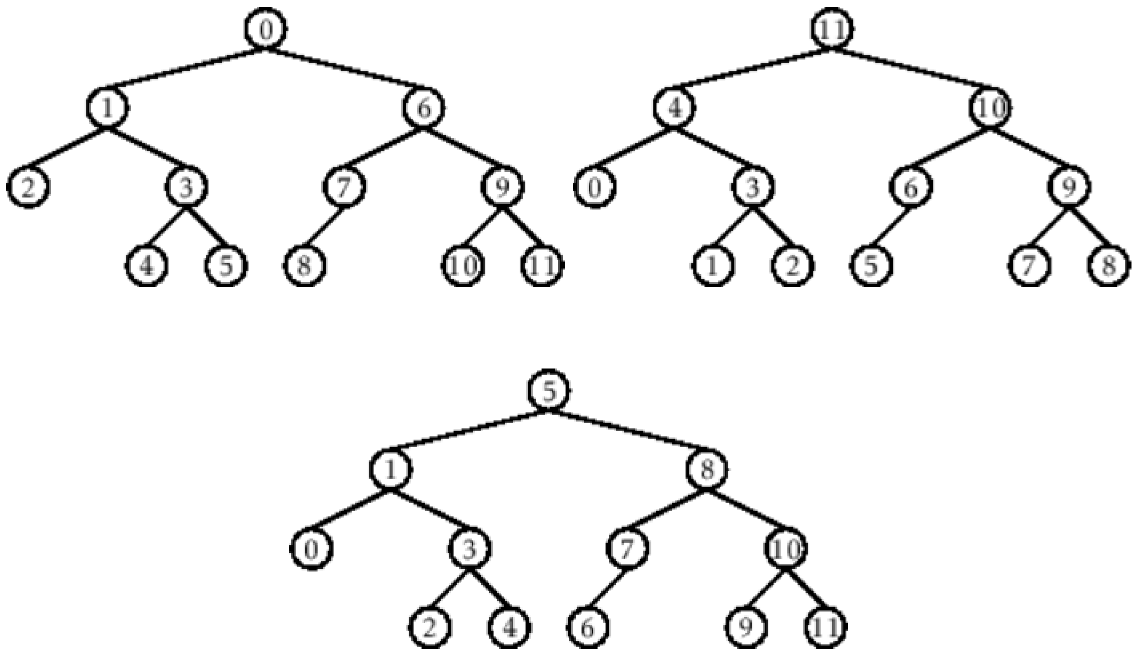
I. Objectives

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 When would you prefer DFS over BFS and vice versa?
2 What is the space complexity difference between DFS and BFS?
3 How does the traversal order differ between DFS and BFS?
4 When does DFS recursive fail compared to DFS iterative?

III. Results

1. When would you prefer DFS over BFS and vice versa?

I would use DFS when you need to explore all the way to the bottom of a path. Use BFS when you need to find the shortest path or explore level by level. Choose DFS for deep trees and BFS for wide trees.

2. What is the space complexity difference between DFS and BFS?

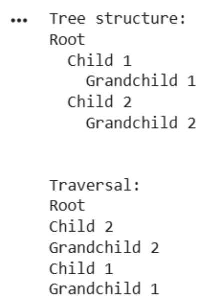
DFS uses memory based on the tree's height, which is good for deep, narrow trees. BFS uses memory based on the tree's width, which is good for shallow trees but bad for wide ones. In the worst case, BFS generally needs to store more nodes at once than DFS. This makes DFS more memory-efficient for certain types of trees.

3. How does the traversal order differ between DFS and BFS?

The code's DFS prints "Root", then "Child 2", then "Grandchild 2", then "Child 1", and finally "Grandchild 1". A BFS would print the tree in level order: "Root", "Child 1", "Child 2", "Grandchild 1", "Grandchild 2". The order is completely different.

4. When does DFS recursive fail compared to DFS iterative?

If the tree in the code were very deep, a recursive DFS would crash. Traverse function in this code is repeated it uses a while loop and a list as a stack so it can handle much deeper trees without crashing because a list can grow much larger than the program's call stack.



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


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

Figure 1 Screenshot of program output

IV. Conclusion

In conclusion, this laboratory activity successfully implemented a tree data structure and its fundamental traversal algorithm. The Python program created a tree with a root, children, and grandchildren, showing how nodes are linked. The main achievement was the implementation of the traverse method, which performed a Depth-First Search (DFS) using an repeated, stack-based approach.

References

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

[2] GeeksforGeeks. (2025, July 23). When to use DFS or BFS to solve a Graph problem? SGeeksforGeeks. https://www.geeksforgeeks.org/dsa/when-to-use-dfs-or-bfs-to-solve-a-graph-problem/

[3] GeeksforGeeks. (2025a, July 11). Difference between BFS and DFS. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/difference-between-bfs-and-dfs/

[4] Sauce, T. (2024, November 5). Time Complexity and Space Complexity of DFS and BFS Algorithms, Medium. https://techsauce.medium.com/time-complexity-and-space-complexity-of-dfs-and-bfs-algorithms-671217e43d58

[5] DFS Recursive VS DFS Iterative. (2014, November 20). Stack Overflow. https://stackoverflow.com/questions/27033429/dfs-recursive-vs-dfs-iterative