Report: Tree of n-Children with Weighted Nodes

Orysbay Ulykpan

1 Introduction

In this assignment, we analyze a special type of rooted tree structure where:

- Each node has exactly n children,
- Each child has weight equal to $\frac{1}{n}$ of its parent's weight,
- The total tree depth is fixed as $N = 3$.

The objective was to implement and test several forms of graph traversal (Depth-First and Breadth-First), as well as investigate the effect of dynamically changing the node weights during traversal.

2 Tree Structure and Weighting

Each node is represented by a class `TreeNode`, containing:

- A real-valued **weight**
- A list of **children nodes**

Every child node is assigned weight:

$$w_{\text{child}} = \frac{w_{\text{parent}}}{n}$$

Two root configurations were used:

1. **Literal interpretation:**

$$w_{\text{root}} = \frac{1}{n}$$

Total weight of the tree:

$$\sum = \frac{N+1}{n}$$

2. **Normalized version:**

$$w_{\text{root}} = \frac{1}{N+1}$$

This guarantees:

$$\text{Total sum} = 1$$

The normalized version was used to validate consistency for various values of n.

3 Depth-First Search (DFS)

DFS is implemented using recursion. Each node contributes its weight to the total sum as well as recursively calling DFS on its children:

$$S = w_{node} + \sum_{child} S_{child}$$

This method explores the tree deeply, following each branch to its end before moving on to the next.

4 Breadth-First Search (BFS)

BFS is implemented using a queue (FIFO). All nodes at the same depth level are visited before moving to the next level.

$$Q = [root]$$

Then repeatedly:

$$\text{pop}(Q) \rightarrow \textit{add children to } Q$$

Both DFS and BFS yield the same total weight of 1 (for the normalized version), for all values of:

$$n = 2, 3, 5, 10$$

5 Sign-Flipping Traversals

An additional variation required flipping the sign of each node's weight when it is reached:

$$w_i := -w_i$$

This produces alternating total sums:

Traversal Number	Total Weight
1st	+1
2nd	-1
3rd	+1

This occurs because each traversal multiplies the sum by (-1):

$$S_{k+1} = -S_k$$

Both DFS and BFS demonstrated this behavior correctly.

6 Recursive vs Iterative BFS

A recursive implementation of BFS was also demonstrated by processing the tree level-by-level.
However, BFS is not recommended to be recursive because:

- BFS uses a queue (FIFO) naturally – not a stack.
- Recursion uses a stack (LIFO).
- Recursive BFS can cause stack overflow on large or wide trees.
- Iterative BFS is simpler and more efficient.

On the other hand, DFS is well-suited for recursion.

7 Results

The following results were verified experimentally:

- DFS sum = 1
- BFS sum = 1
- DFS flip: $1, -1$

- BFS flip: $1, -1$
- Recursive BFS and Iterative BFS gave identical results

These confirm that the tree structure and traversal algorithms are implemented correctly and behave consistently.

8 Conclusion

This task demonstrated:

- Tree construction using recursive data structures
- Correct implementation of DFS and BFS
- Understanding of recursion and queue processing
- Mathematical validity of hierarchical weight distribution

The solution is correct, efficient, and scalable for any n.

9 Appendix: Code Reference

The full Python implementation was committed to GitHub and includes:

- TreeNode class
- Tree generation functions
- DFS / BFS (recursive and iterative)
- Sign-flipping traversal
- Unit tests for different values of n