Report on N-ary Tree with Weights and Traversals

Student: Yessengaliyeva G.

November 15, 2025

1 Objective

The aim of this work is to study the structure of an N-ary tree, generate a weighted tree, perform DFS and BFS traversals, and experiment with changing the sign of node weights. The implementation is in C++ and the code is available on GitHub: `<link to repository>`.

2 Tree Structure

The `Node` class represents a tree node, containing:

- `weight` — the weight of the node;
- `children` — a list of pointers to child nodes;
- methods to add children.

The weight of each child node is calculated as:

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

where n is the number of children.

3 Tree Generation

- Tree depth: 3
- Root weight: 1
- Each node has n children (here $n = 5$)

4 Tree Traversals

4.1 Depth-First Search (DFS)

DFS recursively visits all nodes and sums their weights. The total weight of all nodes is verified to be 1.

4.2 Breadth-First Search (BFS)

BFS uses a queue for iterative traversal. The total weight is also 1, confirming the correctness of the tree structure.

4.3 Traversals with Weight Flipping

During traversal, node weights are inverted:

- DFS Flip #1: sum = -1
- DFS Flip #2: sum = 1
- BFS Flip #1: sum = -1
- BFS Flip #2: sum = 1

Alternating signs helps test the algorithms with weight modification.

4.4 Recursive BFS

Recursive BFS processes the tree level by level. The resulting sum is correct (1). However, recursive BFS is rarely used because:

- It may require a deep recursion stack for large trees;
- Iterative BFS using a queue is simpler and safer.

5 Tree Structure

Example for $n = 5$, depth 3, root weight 0.25:

```
0.25000000
    0.05000000
       0.01000000
          0.00200000
          ...
       ...
    0.05000000
       ...
    0.05000000
        ...
```

6 Results

- DFS sum: 1
- BFS sum: 1
- DFS Flip #1: -1
- DFS Flip #2: 1
- BFS Flip #1: -1
- BFS Flip #2: 1
- Recursive BFS sum: 1

7 Conclusions

- The weighted N-ary tree is generated correctly.
- DFS and BFS yield the same total weight.
- Traversals with alternating signs produce the expected sums.
- Iterative BFS is preferred over recursive BFS due to memory and simplicity.