Problem Set #6

Ainur Amirtayeva

Full Explanation of the Work

In this assignment we study how weight is distributed in a perfectly balanced n-ary tree.

The root node starts with some initial weight w_0. Each node splits its weight equally among its n children. If a parent has weight w_{parent}, then every child receives

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

At the next level there are n children, so the total weight on that level is

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

Therefore the total weight is preserved from level to level. No matter how deep the tree becomes, the sum of all node weights remains equal to the root weight w_0.

In the code we choose the root weight as

$$w_0 = \frac{1}{d+1},$$

where d is the depth of the tree. With this choice the total weight of all nodes in the whole tree is

$$\text{Total Weight} = 1.$$

The main goal of the work is to build such a tree and to check, by different traversal methods, that the total weight is indeed approximately 1 (up to floating–point rounding).

What We Implement

We build an n-ary tree of depth d using the weight rule above, and we test several traversal functions:

- **DFS sum (depth-first):** visit a node, then recursively visit all its children and sum all weights.
- **BFS sum (breadth-first):** visit nodes level by level using a queue.
- **DFS flip:** during DFS, change the sign of every node ($w \leftarrow -w$) and sum the new weights.
- **BFS flip:** change signs during BFS, level by level.

- **Recursive BFS:** compute a breadth-first sum using recursion instead of an explicit queue.

All traversal methods should visit every node exactly once, so the sums produced by them should be the same.

Python Code

```
class Node:
    def __init__(self, weight):
        self.weight = weight
        self.children = []


def build_tree(depth, weight, n):
    node = Node(weight)

    if depth == 0:
        return node

    for _ in range(n):
        child = build_tree(depth - 1, weight / n, n)
        node.children.append(child)

    return node


def dfs_sum(node):
    total = node.weight
    for child in node.children:
        total += dfs_sum(child)
    return total


def bfs_sum(root):
    queue = [root]
    total = 0.0

    while queue:
        current = queue.pop(0)
        total += current.weight

        for child in current.children:
            queue.append(child)

    return total
```

```
def dfs_flip(node):
    node.weight = -node.weight
    total = node.weight

    for child in node.children:
        total += dfs_flip(child)

    return total


def bfs_flip(root):
    queue = [root]
    total = 0.0

    while queue:
        current = queue.pop(0)
        current.weight = -current.weight
        total += current.weight

        for child in current.children:
            queue.append(child)

    return total


def bfs_recursive(level_nodes):
    if not level_nodes:
        return 0.0

    next_level = []
    level_sum = 0.0

    for node in level_nodes:
        level_sum += node.weight
        for child in node.children:
            next_level.append(child)

    return level_sum + bfs_recursive(next_level)


def print_tree(node, level=0):
    indent = "    " * level
    print(f"{indent}weight = {node.weight:.6f}")
    for child in node.children:
        print_tree(child, level + 1)


if __name__ == "__main__":
    n = 5
    depth = 3

    # starting weight for the root
    root_weight = 1.0 / (depth + 1)

    root = build_tree(depth, root_weight, n)

    print("Tree Structure (basic print)")
    print_tree(root)

    print("\nSum Checks")
    print("DFS sum:", dfs_sum(root))
    print("BFS sum:", bfs_sum(root))

    print("\nFirst Flip")
    print("DFS flip:", dfs_flip(root))
    print("BFS flip:", bfs_flip(root))

    print("\nSecond Flip")
    print("DFS flip #2:", dfs_flip(root))
    print("BFS flip #2:", bfs_flip(root))

    print("\nRecursive BFS")
    print("Recursive BFS sum:", bfs_recursive([root]))

    print("\nProgram Finished")
```

Program Output (Short)

For the test case with $n = 5$ and depth $d = 3$ the program prints (for example):

```
Sum Checks
DFS sum: 1.0
BFS sum: 1.0000000000000002

First Flip
DFS flip: -1.0
BFS flip: 1.0000000000000002

Second Flip
DFS flip #2: -1.0
BFS flip #2: 1.0000000000000002

Recursive BFS
Recursive BFS sum: 1.0000000000000002
```

Short Summary

- The tree is built so that each child gets $\frac{1}{n}$ of its parent weight, which preserves the total weight.
- DFS, BFS, and recursive BFS all give a total weight close to 1, which confirms the formula and the correctness of the traversals.
- After one flip the total weight becomes about -1; after the second flip it returns to $+1$, showing that all nodes were visited again.