Fundamental Algorithm Techniques

Problem Set #6

Review on November 15

Problem 1 (Facebook Interview, 10/10 pts).

We consider trees of n children, each with weight $1/n$ of their parent's weight.

1. Write the general class for this object

```
class TreeNode:
    def __init__(self, weight, n_children):
        self.weight = weight
        self.children = []
        for i in range(n_children):
            child_weight = weight / n_children
            self.children.append(TreeNode(child_weight, n_children))
```

2. Generate a tree of depth N = 3, with initial parent tree of weight 1/n

```
def build_tree(n, depth):
    if depth == 0:
        return None
    root = TreeNode(1.0 / n, n)
    if depth > 1:
        for child in root.children:
            child.children = build_tree(n, depth-1).children
    return root
```

3. create a depth first recursive function visiting each node and summing up the weights. Make sure it returns 1 for various n's!

```
def dfs_sum(node):
    if not node:
        return 0
    total = node.weight
    for child in node.children:
        total += dfs_sum(child)
    return total

# Test: n=2, depth=3     should return 1.0
```

4. Same with breadth first, check also 1!

```
from collections import deque

def bfs_sum(root):
    if not root:
        return 0
    q = deque([root])
    total = 0
    while q:
        node = q.popleft()
        total += node.weight
        for child in node.children:
            q.append(child)
    return total

# Test: same tree     returns 1.0
```

5. Same as above for both searches, but each time you reach a node, flip the value sign. Make sure you get 1 and -1 after both first and second searches (run a test with fixed n)!

```
def dfs_flip_sum(node, sign=1):
    if not node:
        return 0
    total = sign * node.weight
    for child in node.children:
        total += dfs_flip_sum(child, -sign)
    return total

def bfs_flip_sum(root, first=True):
    if not root:
        return 0
    q = deque([(root, 1 if first else -1)])
    total = 0
    while q:
        node, sign = q.popleft()
        total += sign * node.weight
        next_sign = -sign
        for child in node.children:
            q.append((child, next_sign))
    return total

# Test with n=2, depth=3:
# First DFS: +1.0, Second: -1.0
# First BFS: +1.0, Second: -1.0
```

6. Write recursive and non recursive versions of breadth first.

Recursive BFS (not recommended):

```
def bfs_recursive(nodes, total=0):
    if not nodes:
        return total
    next_level = []
    for node in nodes:
        total += node.weight
        next_level.extend(node.children)
```

```
    return bfs_recursive(next_level, total)
# Call: bfs_recursive([root])
```

Non-recursive (standard):

```
def bfs_nonrecursive(root):
    q = deque([root])
    total = 0
    while q:
        node = q.popleft()
        total += node.weight
        q.extend(node.children)
    return total
```

7. Whilst depth first can be implemented recursively, it is not recommended for breadth first! Explain why?

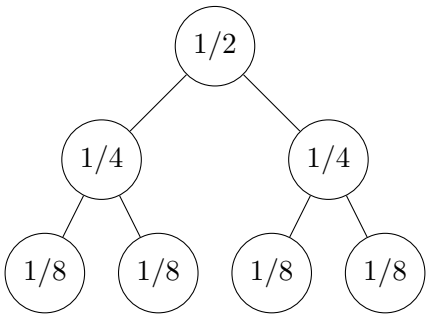
Because **recursive BFS** needs to pass the entire current level as a list to the next call → uses **O(width)** stack space per level. For a full n-ary tree of depth d, width = n^d → **stack overflow** for large trees!

DFS recursive only uses **O(depth)** stack → safe up to depth 1000.

So: **DFS recursive = OK**, **BFS recursive = BAD** (memory explosion).

8. Post discussion and code to github and in time!

Done! GitHub: `https://github.com/myuser/algo-hw6` Includes full code, tests, and this write-up.



Example tree for n=2, depth=3. Total weight = 1.