📓 Most Frequent Subtree Sum - Documentation

📌 Table of Contents

1. Problem Statement
2. Intuition
3. Key Observations
4. Approach
5. Edge Cases
6. Complexity Analysis
    - Time Complexity
    - Space Complexity
7. Alternative Approaches
8. Test Cases
9. Final Thoughts

1. ✅ Problem Statement

Given the root of a binary tree, return the most frequent subtree sum(s).

- A subtree sum is the sum of all values in a subtree rooted at a node (including the node itself).
- If there are multiple sums with the same maximum frequency, return them in any order.

✨ Constraints:

- $1 <= \text{Number of nodes} <= 10^4$
- $-10^5 <= \text{Node.val} <= 10^5$

2. 💡 Intuition

Every node defines a subtree, and each subtree has a sum.To solve the problem:

- Traverse the tree
- Calculate the sum of each subtree
- Track how many times each sum appears
- Return the sum(s) that appear most frequently

3. 🔍 Key Observations

- A post-order traversal (left-right-root) is ideal since we need to calculate the subtree sum after computing the sums of left and right children.
- We can use a hash map to count how often each subtree sum occurs.

4. Approach

- Use a recursive post-order traversal (dfs) to:
  - Compute the left and right subtree sums
  - Add them with the current node's value
  - Update a Counter to store frequency of each subtree sum
- After the traversal:
  - Identify the maximum frequency
  - Return all subtree sums that match that frequency

5. ⚠ Edge Cases

- Tree has only one node → the subtree sum is the node value itself.
- All nodes have value 0 → subtree sum of any node is also 0.
- Negative values → must handle them without assumptions about positivity.

6. Complexity Analysis

 Time Complexity:

- O(n) where n is the number of nodes
    - Each node is visited exactly once

 Space Complexity:

- O(n) for:
    - Recursion stack (in worst case: skewed tree)
    - HashMap storing frequencies of subtree sums

7. 🔁 Alternative Approaches

- Iterative DFS or BFS: Possible but complex due to the need to compute child sums first.
- Using a global variable instead of returning sums: Less clean, more side-effects.
- Serialization-based approach: Inefficient for large trees, not recommended.

8. Test Cases

Test Case 1:

Input: root = [5, 2, -3]
Output: [2, -3, 4]

Test Case 2:

Input: root = [5, 2, -5]
Output: [2]

Test Case 3:

Input: root = [1]
Output: [1]

Test Case 4:

Input: root = [0, 0, 0]
Output: [0]

9. □ Final Thoughts

- This problem highlights the power of post-order traversal for bottom-up computations in trees.
- It's a great example of combining DFS with hash maps to track frequencies.
- Easily extendable to related problems like "Most Frequent Path Sum", "Max Subtree Product", etc.