# Most Frequent Subtree Sum

Given the root of a tree, you are asked to find the most frequent subtree sum. The subtree sum of a node is defined as the sum of all the node values formed by the subtree rooted at that node (including the node itself). So what is the most frequent subtree sum value? If there is a tie, return all the values with the highest frequency in any order.

**Examples 1**
Input:

```
    5
   /  \
  2   -3
```

return [2, -3, 4], since all the values happen only once, return all of them in any order.
**Examples 2**
Input:

```
    5
   /  \
  2   -5
```

return [2], since 2 happens twice, however -5 only occur once.
**Note:** You may assume the sum of values in any subtree is in the range of 32-bit signed integer.

## Solution 1

For sake of saving time during contest, can't write so concise solution :)
Idea is `post-order` traverse the tree and get sum of every sub-tree, put `sum` to `count` mapping to a HashMap. Then generate result based on the HashMap.

```
public class Solution {
    Map<Integer, Integer> sumToCount;
    int maxCount;

    public int[] findFrequentTreeSum(TreeNode root) {
        maxCount = 0;
        sumToCount = new HashMap<Integer, Integer>();

        postOrder(root);

        List<Integer> res = new ArrayList<>();
        for (int key : sumToCount.keySet()) {
            if (sumToCount.get(key) == maxCount) {
                res.add(key);
            }
        }

        int[] result = new int[res.size()];
        for (int i = 0; i < res.size(); i++) {
            result[i] = res.get(i);
        }
        return result;
    }

    private int postOrder(TreeNode root) {
        if (root == null) return 0;

        int left = postOrder(root.left);
        int right = postOrder(root.right);

        int sum = left + right + root.val;
        int count = sumToCount.getOrDefault(sum, 0) + 1;
        sumToCount.put(sum, count);

        maxCount = Math.max(maxCount, count);

        return sum;
    }
}
```

written by shawngao original link here

## Solution 2

I have used a hash-map `ctr` to count the sum occurrence.

I have wrote a sub function `countSubtreeSum` to travel through a tree and return the sum of the tree.

In `countSubtreeSum`, I will travel through the left node and the right node, calculate the sum of the tree, increment the counter `ctr`, and return the sum.

```
def findFrequentTreeSum(self, root):
    import collections
    ctr = collections.Counter()
    if root == None: return []
    self.countTreeSum(root, ctr)
    frequent = max(ctr.values())
    return [i for i in ctr.keys() if ctr[i] == frequent]

  def countTreeSum(self, root, ctr):
    if root == None: return 0
    else:
      val = self.countSubtreeSum(root.left, ctr) + self.countSubtreeSum(root.righ
t, ctr) + root.val
      ctr[val] += 1
      return val
```

Please upvote if it makes sense

written by lee215 original link here

## Solution 3

```java
public class Solution {
    int max = 0;
    public int[] findFrequentTreeSum(TreeNode root) {
        if(root==null) return new int[0];
        Map<Integer, Integer> map = new HashMap<>();
        helper(root, map);
        List<Integer> res = new LinkedList<>();
        for(Map.Entry<Integer, Integer> me: map.entrySet()){
            if(me.getValue()==max) res.add(me.getKey());
        }
        return res.stream().mapToInt(i->i).toArray();
    }

    private int helper(TreeNode n, Map<Integer, Integer> map){
        int left = (n.left==null) ? 0 : helper(n.left, map);
        int right = (n.right==null) ? 0 : helper(n.right, map);
        int sum = left + right + n.val;
        map.put(sum, map.getOrDefault(sum,0)+1);
        max = Math.max(max, map.get(sum));
        return sum;
    }
}
```

written by Chidong original link here