Description

124. Binary Tree Maximum Path Sum

Given a binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path must contain at least one node and does not need to go through the root.

For example:
Given the below binary tree,

```
       1
      / \
     2   3
```

Return 6.

Idea Report

The problem explicitly says:

> "The path must contain at least one node and does not need to go through the root."

So this is a child-to-parent-to-child path sum. So we can split this path sum into three parts. A parent-to-child path sum from left, the root value itself, and a parent-to-child path sum from right. So we traverse the whole tree, for each root (parent), we need to compare this path's sum to the global maximum. Since we need the left side and right side of the parent-to-child path sum, we can use bottom up method.

When writing the code, we can discuss the corner cases: if left/right side path sum is less than 0, we need carefully compare all kinds of the situation. Another trick I learned in the class is to use

int[] max; instead of the global variable.

Java

```java
class Solution {
  // AC
  public int maxPathSum(TreeNode root) {
    if (root == null) {
        return 0;
    }
    int[] max = new int[1];
    max[0] = root.val;
    helper(root, max);
    return max[0];
  }

  private int helper(TreeNode root, int[] max) {
    if (root == null) {
        return 0;
    }

    int left = helper(root.left, max);
    int right = helper(root.right, max);
    int temp = Math.max(root.val, Math.max(left + root.val, right + root.val));
    max[0] = Math.max(max[0], Math.max(temp, left + right + root.val));
    return temp;
  }
}
```

C++

```cpp
class Solution {
public:
    int maxPathSum(TreeNode* root) {
        if (root == NULL) {
            return 0;
        }

        int max = root->val;
        helper(root, max);
        return max;
    }

private:
    int helper(TreeNode* root, int& max) {
        if (root == NULL) {
```

```
            return 0;
        }

        int left = helper(root->left, max);
        int right = helper(root->right, max);
        int temp = std::max(root->val,
            std::max(left + root->val, right + root->val));
        max = std::max(max, std::max(temp, left + root->val + right));
        return temp;
    }
};
```

Summary

- If we need information from left/right child and do not need pass any parent information down to children, we can just use bottom up method.
- Use int[] max = new int[1]; instead of a global variable, C++ can pass reference.