1. Binary Tree Maximum Path Sum

Given a **non-empty** 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.

**Example 1:**

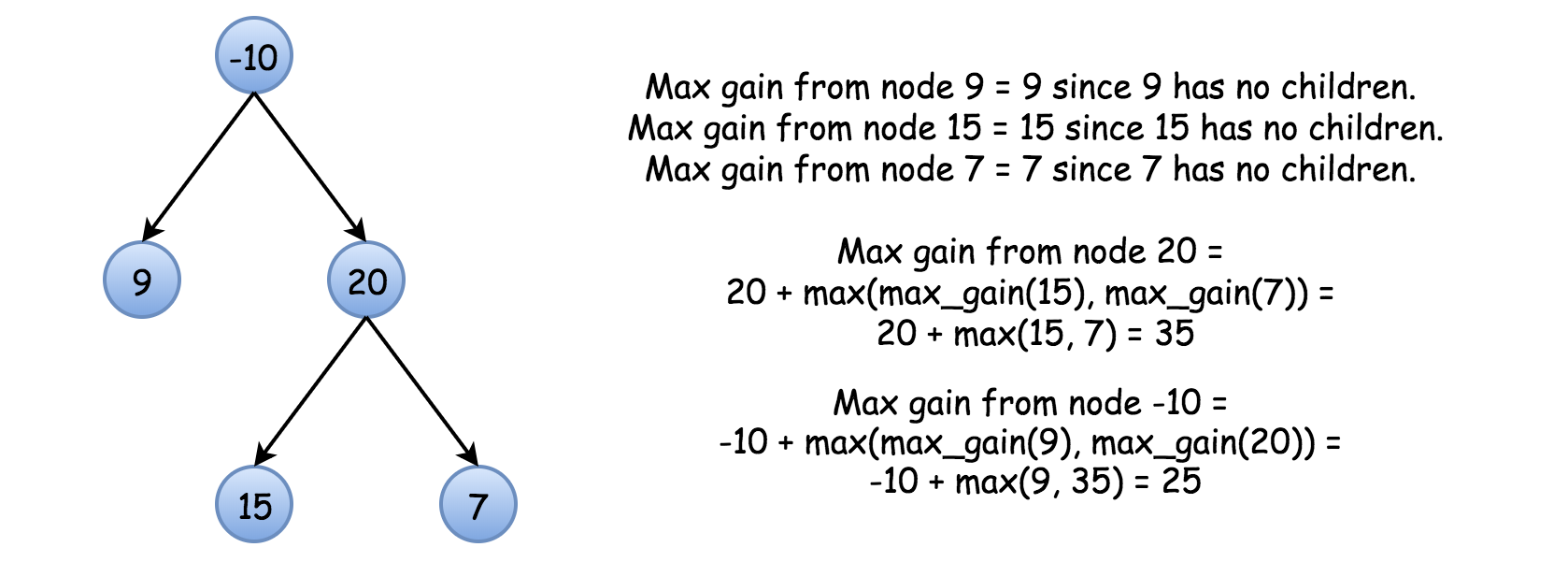
Input: [1,2,3]  
  
 1  
 / \  
 2 3  
  
Output: 6

**Example 2:**

Input: [-10,9,20,null,null,15,7]  
  
 -10  
 / \  
 9 20  
 / \  
 15 7  
  
Output: 42

**解**

定义节点node加入路径后能够贡献的最大效益：max\_gain



算法：

Now everything is ready to write down an algorithm.

* Initiate max\_sum as the smallest possible integer and call max\_gain(node = root).
* Implement
* with a check to continue the old path/to start a new path:
  + Base case : if node is null, the max gain is 0.
  + Call max\_gain recursively for the node children to compute max gain from the left and right subtrees : left\_gain = max(max\_gain(node.left), 0) and right\_gain = max(max\_gain(node.right), 0).
  + Now check to continue the old path or to start a new path. To start a new path would cost price\_newpath = node.val + left\_gain + right\_gain. Update max\_sum if it’s better to start a new path.
  + For the recursion return the max gain the node and one/zero of its subtrees could add to the current path : node.val + max(left\_gain, right\_gain).

/\*\*  
 \* Definition for a binary tree node.  
 \* struct TreeNode {  
 \* int val;  
 \* TreeNode \*left;  
 \* TreeNode \*right;  
 \* TreeNode() : val(0), left(nullptr), right(nullptr) {}  
 \* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}  
 \* TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}  
 \* };  
 \*/  
class Solution {  
public:  
 int max\_sum = INT\_MIN;  
 int maxPathSum(TreeNode\* root) {  
 max\_gain(root);  
 return max\_sum;  
 }  
 int max\_gain(TreeNode\* ptr){  
 if(!ptr) return 0;  
 int left\_val = max(max\_gain(ptr->left),0);  
 int right\_val = max(max\_gain(ptr->right),0);  
 int cur\_gain = ptr->val + left\_val + right\_val;  
 max\_sum = max(max\_sum, cur\_gain);  
 return ptr->val + max(left\_val, right\_val);  
 }  
};