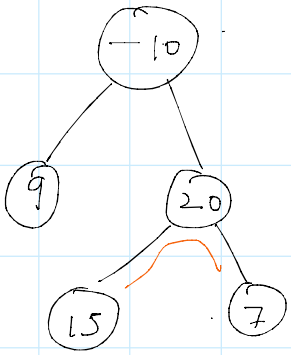
124. Binary Tree Maximum Path Sum

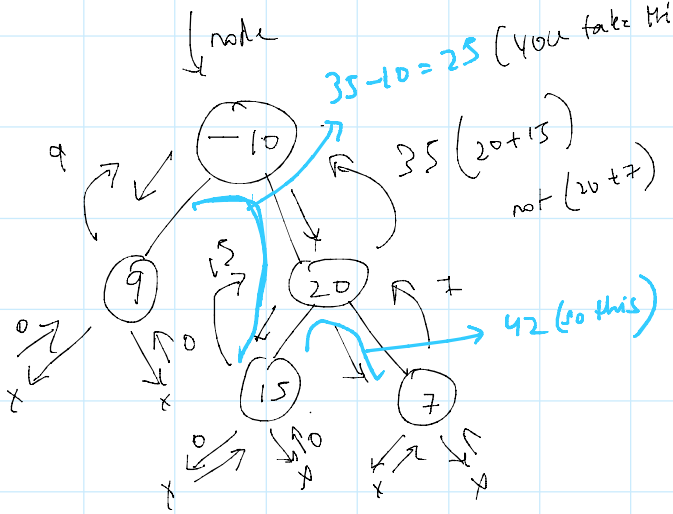
19 February 2022 10:41 AM



maxi path = 15 + 20 + 7 = 42

* Finding the maxi height of a BT / width of a B.T

formula ⇒ Val + (maxL + maxR)



maxi = 0 9 15 42

for node (9)

maxi = max(0, 0+0+9)

maxi = 9

```
int maxPath (node, maxi)
{
    if (node == null) return 0;
    leftSum = maxPath (node.left, maxValue)
    rightSum = maxPath (node.right, maxValue)
    maxi = max (maxi, leftsum + rightsum + node);
    return (node.val) + max (leftSum, rightSum);
}
```

return 9 + 0 = 9

for (-10) node

left sum it got = 9

now for right sum = 35 (found below)

maxi = max (42, 9 + (-10) + 35)

= 42 (< 34) X

for node (15)

maxi = max (9, 0 + 0 + 15)

maxi = 15

return 15 + 0 = 15

for node (7)

maxi = max (15, 0 + 0 + 7)

maxi = 15

return 7 + 0 = 7

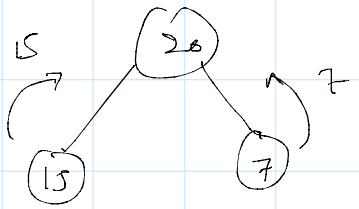
for node (20)

maxi = max (15, 15 + 7 + 20)

maxi = 42

So now if you look

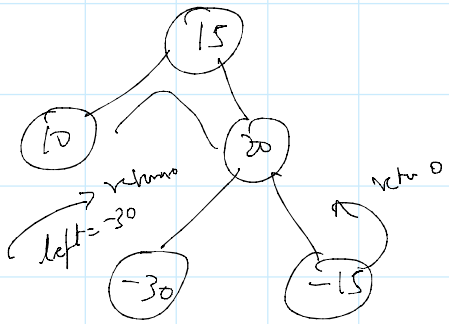
return (node.val) + max (ls + rs)



Here you consider ls as it max

return 20 + 15 = 35

Test Case



This logic won't work here

when there is (-ve) return 0.

taking (-ve) will not give max path

So return 0; and ignore -ve path.

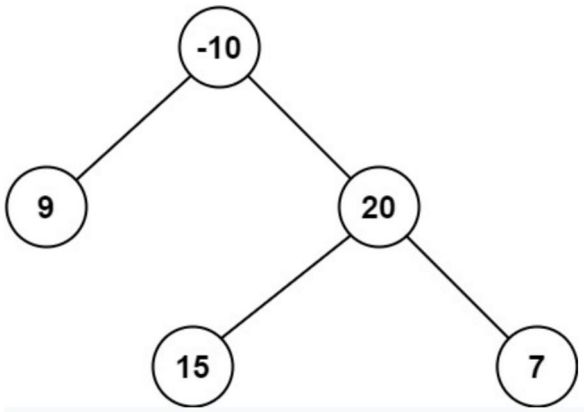
124. Binary Tree Maximum Path Sum

Hard 8568 508 Add to List Share

A **path** in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence **at most once**. Note that the path does not need to pass through the root.

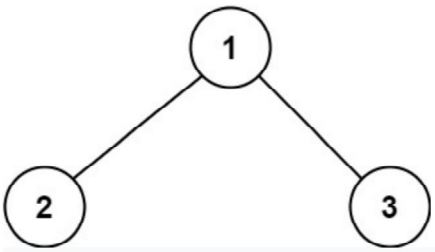
The **path sum** of a path is the sum of the node's values in the path.

Given the `root` of a binary tree, return *the maximum **path sum** of any **non-empty** path*.




```
Input: root = [-10,9,20,null,null,15,7]
Output: 42
Explanation: The optimal path is 15 -> 20 -> 7 with a path sum
```

Example 1:




```
Input: root = [1,2,3]
Output: 6
Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of
2 + 1 + 3 = 6.
```

```
Explanation: The optimal path is 15 -> 20 -> 7 with a path sum
of 15 + 20 + 7 = 42.
```

Constraints:

- The number of nodes in the tree is in the range `[1, 3 * 10^4]`.
- `-1000 <= Node.val <= 1000`

C++

```cpp
class Solution {
public:
    int maxPathSum(TreeNode* root) {
       int maxi = INT_MIN;
        maxPathDown(root, maxi);
        return maxi;
    }

    int maxPathDown(TreeNode* node, int &maxi) {
        if (node == NULL) return 0;
        int left = max(0, maxPathDown(node->left, maxi));
        int right = max(0, maxPathDown(node->right, maxi));
        maxi = max(maxi, left + right + node->val);
        return max(left, right) + node->val;
    }
};
```

Java

```java
class Solution {
    public int maxPathSum(TreeNode root) {
        // as in java taking it as array because we know that
        // variable and cannot pass it as reference so declared
        // with size 1 and stored the maxvalue in 0th index
        // in C++ direct int maxi = INT_MIN;
        int maxValue[] = new int[1];
        maxValue[0] = Integer.MIN_VALUE;
        maxPathDown(root, maxValue);
        return maxValue[0];
    }
```

```
    private int maxPathDown(TreeNode node, int maxValue[]){
        if(node == null) return 0;

        //if there is -ve then sum will be 0
        int leftSum = Math.max(0, maxPathDown(node.left, maxValue));
        int rightSum = Math.max(0, maxPathDown(node.right, maxValue));

        maxValue[0] = Math.max(maxValue[0], leftSum + rightSum + node.val);
        return Math.max(leftSum, rightSum) + node.val;
    }
}
```