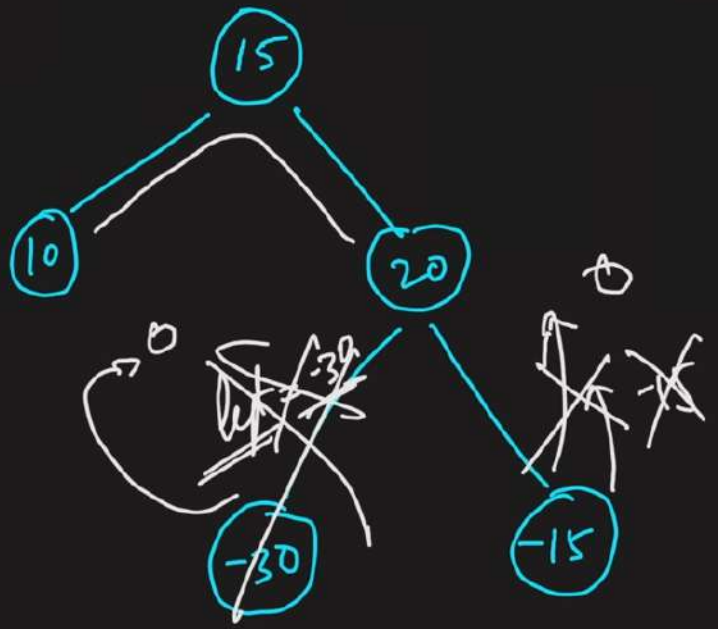
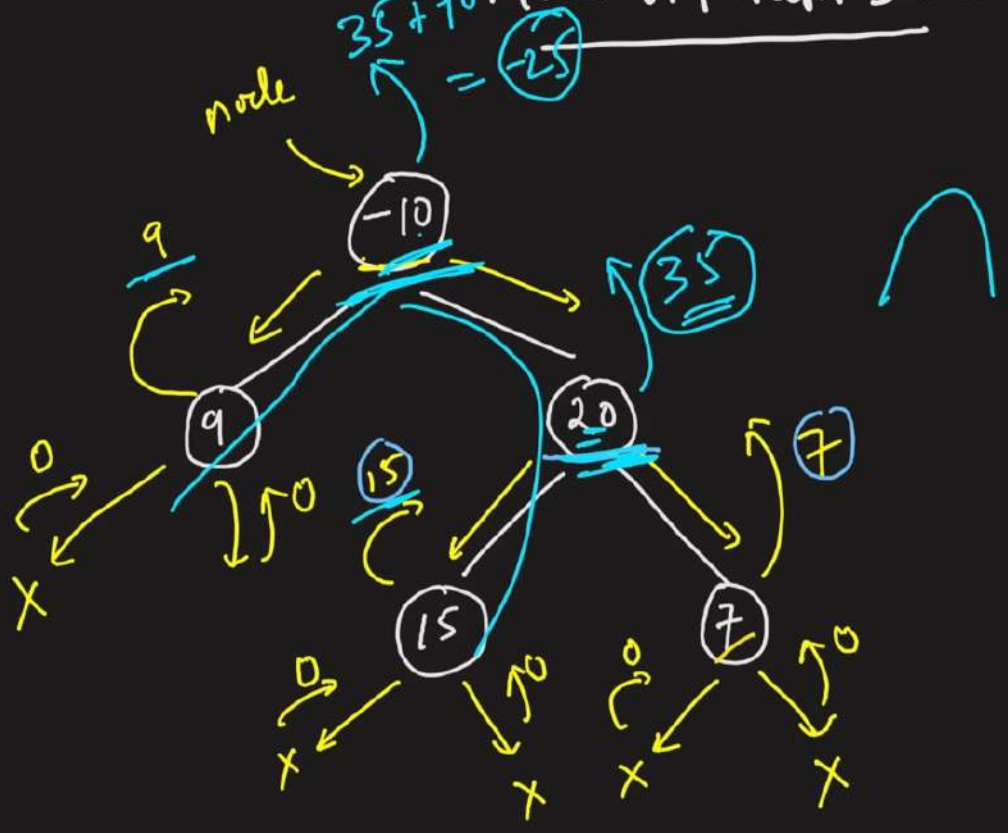


Max Path

Maximum Path Sum

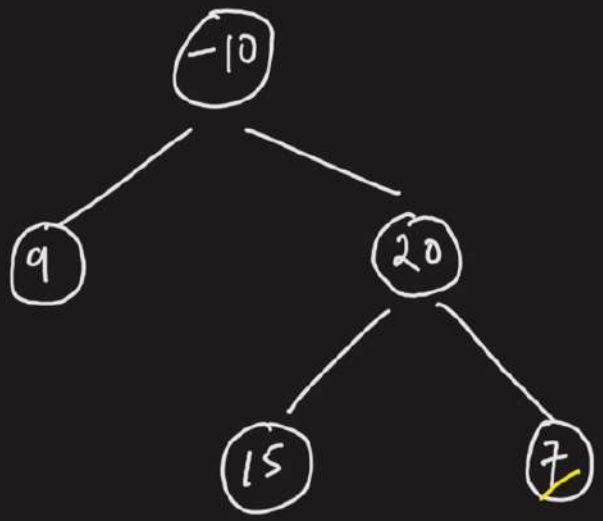


15 + 7 + 20

9 + -10 + 35 = 34

```
mani = 0 9 15 42

int maxPath(node, mani)
{
    if (node == null)
        return 0;

    leftsum = maxPath(node->left,
    rightsum = maxPath(node->right,

    mani = max(mani, leftsum + rightsum + node->val);

    return (node->val) + max(leftsum, rightsum);
}

1 + max(lh, rh)
```

Maximum Path Sum

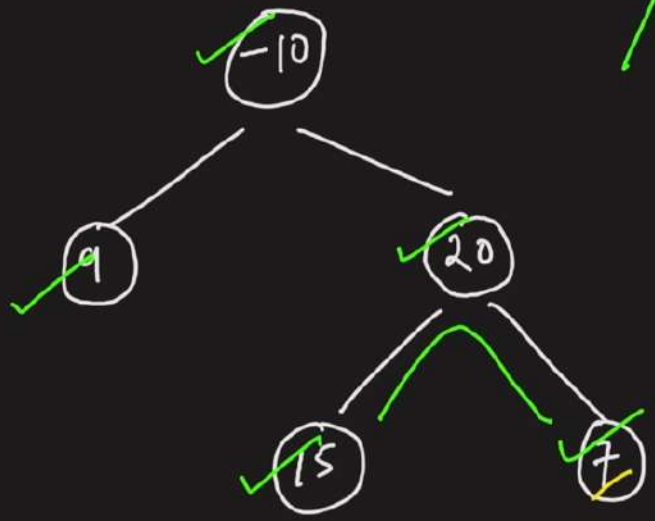


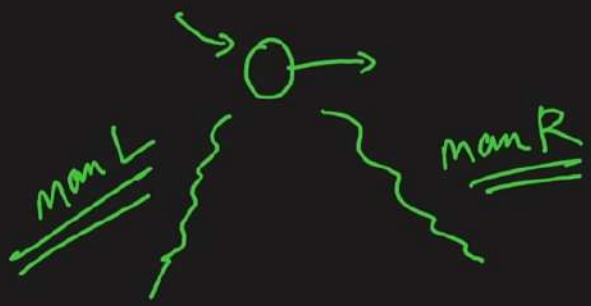
[(node->val) + (right + left)]

```
int maxPath(node,
{
    if (node == null)
        return 0;

leftsum = maxPath(node->left,
rightsum = maxPath(node->right,

return (node->val) + max(leftsum,
                         rightsum);
}
```

1 + max(lh, rh)

Maximum Path Sum

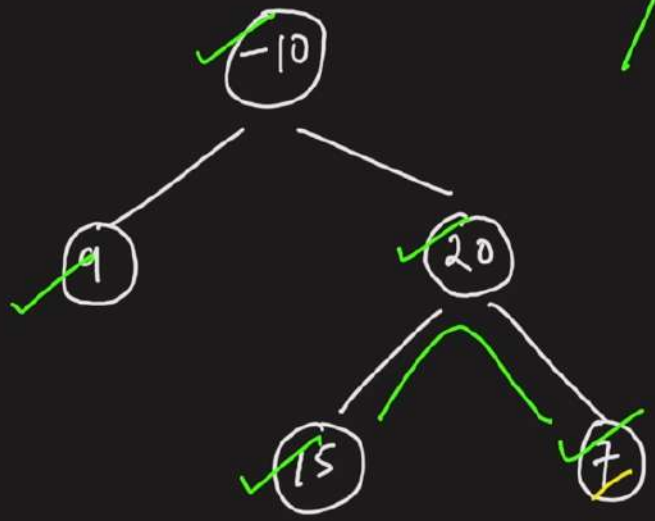


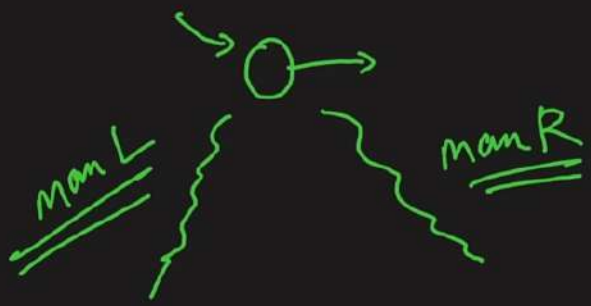


$\Rightarrow$ [val + (maxL + maxR)]

maximum

Maximum Path Sum





$\Rightarrow$ [val + (maxL + maxR)]

maximum

Java

Autocomplete

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public int maxPathSum(TreeNode root) {
        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;
        int left = Math.max(0, maxPathDown(node.left, maxValue));
        int right = Math.max(0, maxPathDown(node.right, maxValue));
        maxValue[0] = Math.max(maxValue[0], left + right + node.val);
        return Math.max(left, right) + node.val;
    }
}
```