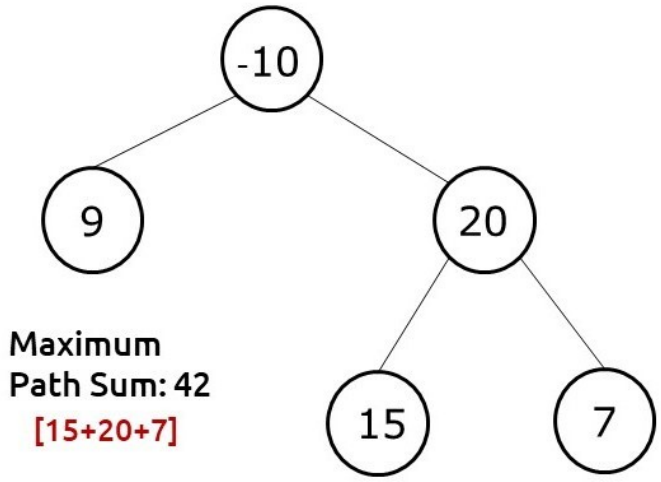
Maximum Sum Path in Binary Tree

Problem Statement: Write a program to find the maximum sum path in a binary tree. A path in a binary tree is a sequence of nodes where every adjacent pair of nodes are connected by an edge. A node can only appear in the sequence at most once. A path need not pass from the root. We need to find the path with the maximum sum in the binary tree.

Example:

Input:

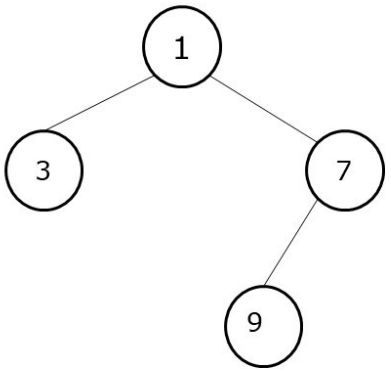


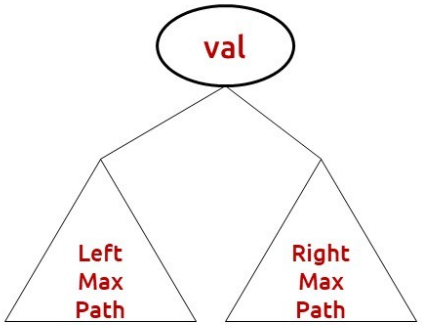
Output: The Max Path Sum for the Tree is 42

Disclaimer: *Don't jump directly to the solution, try it out yourself first.*

Solution:

Approach: A brute force approach would be to generate all paths and compare them. Generating all paths will be a time-costly activity therefore we need to look for something else.
We first need to define what is the maximum path sum through a given node (when that node is acting as the root node/curving point). At a given node with a value, if we find the max leftSumPath in the left-subtree and the max rightSumPath in the right subtree, then the maxPathSum through that node is value+ (leftSumPath+rightSumPath).





At node 1:

value: 1

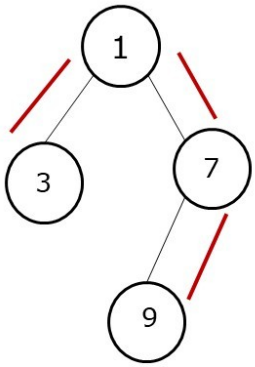
leftMaxPath: 3

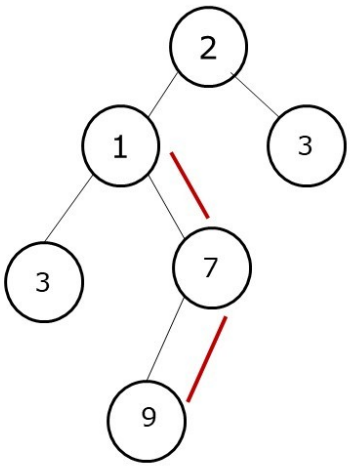
rightMaxPath: 16

Max Path sum
through a node1 : 17

Max Path sum
through a node
(when that node is acting
as a turning point)
: val + (leftMaxPath + rightMaxPath)

Now we can apply this formula at every node by doing a simple tree traversal and storing the maximum value (our answer) in a reference variable.
For our recursion to work, it is very important to understand what value we return from our function. In our recursive function, we find and compare the maxPathSum from a given node when it is the root/turning point of the path. But what we return is the maxPathSum of that same node when it is **NOT** the root/turning point of the path. To find the latter maxPath, we no longer have the liberty to consider both leftMaxPath and rightMaxPath, we will simply take the maximum of the two and it to the value of the node.





Max Path sum through a node : val + max (leftMaxPath + rightMaxPath)
(when that node is NOT acting as a turning point)

Max Path sum through a node1 (when acting as root/curving point): 20
value: 1
leftMaxPath: 3
rightMaxPath: 16

Max Path sum through a node1 (when NOT acting as root/curving point):
1+max(3,16) = 17

To summarize:
- Initialize a maxi variable to store our final answer.
- Do a simple tree traversal. At each node, find recursively its leftMaxPath and its rightMaxPath.
- Calculate the maxPath through the node as val + (leftMaxPath + rightMaxPath) and update maxi accordingly.
- Return the maxPath when node is not the curving point as val + max(leftMaxPath, rightMaxPath).

Code:

- C++ Code
- Java Code

```cpp
#include <bits/stdc++.h>

using namespace std;

struct node {
  int data;
  struct node * left, * right;
};

int findMaxPathSum(node * root, int & maxi) {
  if (root == NULL) return 0;

  int leftMaxPath = max(0, findMaxPathSum(root -> left, maxi));
  int rightMaxPath = max(0, findMaxPathSum(root -> right, maxi));
  int val = root -> data;
  maxi = max(maxi, (leftMaxPath + rightMaxPath) + val);
  return max(leftMaxPath, rightMaxPath) + val;

}

int maxPathSum(node * root) {
  int maxi = INT_MIN;
  findMaxPathSum(root, maxi);
  return maxi;

}

struct node * newNode(int data) {
  struct node * node = (struct node * ) malloc(sizeof(struct node));
  node -> data = data;
  node -> left = NULL;
  node -> right = NULL;

  return (node);
```

```
}

int main() {

   struct node * root = newNode(-10);
   root -> left = newNode(9);
   root -> right = newNode(20);
   root -> right -> left = newNode(15);
   root -> right -> right = newNode(7);

   int answer = maxPathSum(root);
   cout << "The Max Path Sum for this tree is " << answer;

   return 0;
}
```

The Max Path Sum for this tree is 42

Time Complexity: O(N).

Reason: We are doing a tree traversal.

Space Complexity: O(N)

Reason: Space is needed for the recursion stack. In the worst case (skewed tree), space complexity can be O(N).