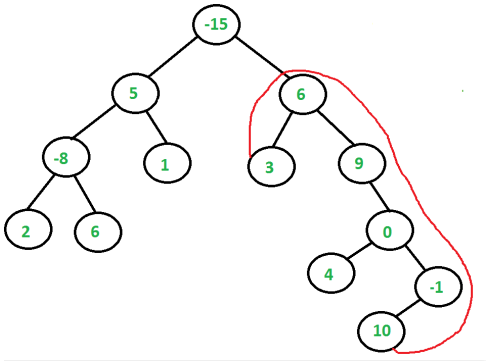
Find the maximum path sum between two leaves of a binary tree

Given a binary tree in which each node element contains a number. Find the maximum possible sum from one leaf node to another.
The maximum sum path may or may not go through root. For example, in the following binary tree, the maximum sum is **27**(3 + 6 + 9 + 0 – 1 + 10). Expected time complexity is O(n).

If one side of root is empty, then function should return minus infinite (INT_MIN in case of C/C++)



A simple solution is to traverse the tree and do following for every traversed node X.
1) Find maximum sum from leaf to root in left subtree of X (we can use this post for this and next steps)
2) Find maximum sum from leaf to root in right subtree of X.
3) Add the above two calculated values and X->data and compare the sum with the maximum value obtained so far and update the maximum value.
4) Return the maximum value.

The time complexity of above solution is $O(n^2)$

We can find the maximum sum using single traversal of binary tree. The idea is to maintain two values in recursive calls
1) Maximum root to leaf path sum for the subtree rooted under current node.
2) The maximum path sum between leaves (desired output).

For every visited node X, we find the maximum root to leaf sum in left and right subtrees of X. We add the two values with X->data, and compare the sum with maximum path sum found so far.

Following is the implementation of the above O(n) solution.

C++

```
// C++ program to find maximum path sum between two leaves of
// a binary tree
#include <bits/stdc++.h>
using namespace std;

// A binary tree node
struct Node
{
    int data;
    struct Node* left, *right;
};

// Utility function to allocate memory for a new node
struct Node* newNode(int data)
{
    struct Node* node = new(struct Node);
    node->data = data;
    node->left = node->right = NULL;
    return (node);
}

// Utility function to find maximum of two integers
int max(int a, int b)
{ return (a >= b)? a: b; }

// A utility function to find the maximum sum between any
// two leaves.This function calculates two values:
// 1) Maximum path sum between two leaves which is stored
//    in res.
// 2) The maximum root to leaf path sum which is returned.
// If one side of root is empty, then it returns INT_MIN
```

```
// If one side of root is empty, then it returns INT_MIN
int maxPathSumUtil(struct Node *root, int &res)
{
    // Base cases
    if (root==NULL) return 0;
    if (!root->left && !root->right) return root->data;

    // Find maximum sum in left and right subtree. Also
    // find maximum root to leaf sums in left and right
    // subtrees and store them in ls and rs
    int ls = maxPathSumUtil(root->left, res);
    int rs = maxPathSumUtil(root->right, res);


    // If both left and right children exist
    if (root->left && root->right)
    {
        // Update result if needed
        res = max(res, ls + rs + root->data);

        // Return maxium possible value for root being
        // on one side
        return max(ls, rs) + root->data;
    }

    // If any of the two children is empty, return
    // root sum for root being on one side
    return (!root->left)? rs + root->data:
                          ls + root->data;
}

// The main function which returns sum of the maximum
// sum path between two leaves. This function mainly
// uses maxPathSumUtil()
int maxPathSum(struct Node *root)
{
    int res = INT_MIN;
    maxPathSumUtil(root, res);
    return res;
}

// driver program to test above function
int main()
{
    struct Node *root = newNode(-15);
    root->left = newNode(5);
    root->right = newNode(6);
    root->left->left = newNode(-8);
    root->left->right = newNode(1);
    root->left->left->left = newNode(2);
    root->left->left->right = newNode(6);
    root->right->left = newNode(3);
    root->right->right = newNode(9);
    root->right->right->right= newNode(0);
    root->right->right->right->left= newNode(4);
    root->right->right->right->right= newNode(-1);
    root->right->right->right->right->left= newNode(10);
    cout << "Max pathSum of the given binary tree is "
         << maxPathSum(root);
    return 0;
}
```

Java

```
// Java program to find maximum path sum between two leaves
// of a binary tree
class Node {

    int data;
    Node left, right;
```

```
    Node(int item) {
        data = item;
        left = right = null;
    }
}

// An object of Res is passed around so that the
// same value can be used by multiple recursive calls.
class Res {
    int val;
}

class BinaryTree {

    static Node root;

    // A utility function to find the maximum sum between any
    // two leaves.This function calculates two values:
    // 1) Maximum path sum between two leaves which is stored
    //    in res.
    // 2) The maximum root to leaf path sum which is returned.
    // If one side of root is empty, then it returns INT_MIN
    int maxPathSumUtil(Node node, Res res) {

        // Base cases
        if (node == null)
            return 0;
        if (node.left == null && node.right == null)
            return node.data;

        // Find maximum sum in left and right subtree. Also
        // find maximum root to leaf sums in left and right
        // subtrees and store them in ls and rs
        int ls = maxPathSumUtil(node.left, res);
        int rs = maxPathSumUtil(node.right, res);

        // If both left and right children exist
        if (node.left != null && node.right != null) {

            // Update result if needed
            res.val = Math.max(res.val, ls + rs + node.data);

            // Return maxium possible value for root being
            // on one side
            return Math.max(ls, rs) + node.data;
        }

        // If any of the two children is empty, return
        // root sum for root being on one side
        return (node.left == null) ? rs + node.data
                : ls + node.data;
    }

    // The main function which returns sum of the maximum
    // sum path between two leaves. This function mainly
    // uses maxPathSumUtil()
    int maxPathSum(Node node)
    {
        Res res = new Res();
        res.val = Integer.MIN_VALUE;
        maxPathSumUtil(root, res);
        return res.val;
    }

    //Driver program to test above functions
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(-15);
        tree.root.left = new Node(5);
        tree.root.right = new Node(6);
        tree.root.left.left = new Node(-8);
        tree.root.left.right = new Node(1);
        tree.root.left.left.left = new Node(2);
```

```
        tree.root.left.left.left = new Node(2);
        tree.root.left.left.right = new Node(6);
        tree.root.right.left = new Node(3);
        tree.root.right.right = new Node(9);
        tree.root.right.right.right = new Node(0);
        tree.root.right.right.right.left = new Node(4);
        tree.root.right.right.right.right = new Node(-1);
        tree.root.right.right.right.right.left = new Node(10);
        System.out.println("Max pathSum of the given binary tree is "
                + tree.maxPathSum(root));
    }
}

// This code is contributed by Mayank Jaiswal
```

Python

```
# Python program to find maximumpath sum between two leaves
# of a binary tree

INT_MIN = -2**32

# A binary tree node
class Node:
    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Utility function to find maximum sum between any
# two leaves. This function calculates two values:
# 1) Maximum path sum between two leaves which are stored
#    in res
# 2) The maximum root to leaf path sum which is returned
# If one side of root is empty, then it returns INT_MIN

def maxPathSumUtil(root, res):

    # Base Case
    if root is None:
        return 0

    if root.left is None and root.right is None:
        return root.data

    # Find maximumsum in left and righ subtree. Also
    # find maximum root to leaf sums in left and righ
    # subtrees ans store them in ls and rs
    ls = maxPathSumUtil(root.left, res)
    rs = maxPathSumUtil(root.right, res)

    # If both left and right children exist
    if root.left is not None and root.right is not None:

        # update result if needed
        res[0] = max(res[0], ls + rs + root.data)

        # Return maximum possible value for root being
        # on one side
        return max(ls, rs) + root.data

    # If any of the two children is empty, return
    # root sum for root being on one side
    if root.left is None:
        return rs + root.data
    else:
        return ls + root.data

# The main function which returns sum of the maximum
```

```
# sum path betwee ntwo leaves. THis function mainly
# uses maxPathSumUtil()
def maxPathSum(root):
        res = [INT_MIN]
        maxPathSumUtil(root, res)
        return res[0]


# Driver program to test above function
root = Node(-15)
root.left = Node(5)
root.right = Node(6)
root.left.left = Node(-8)
root.left.right = Node(1)
root.left.left.left = Node(2)
root.left.left.right = Node(6)
root.right.left = Node(3)
root.right.right = Node(9)
root.right.right.right= Node(0)
root.right.right.right.left = Node(4)
root.right.right.right.right = Node(-1)
root.right.right.right.right.left = Node(10)

print "Max pathSum of the given binary tree is", maxPathSum(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
Max pathSum of the given binary tree is 27.
```