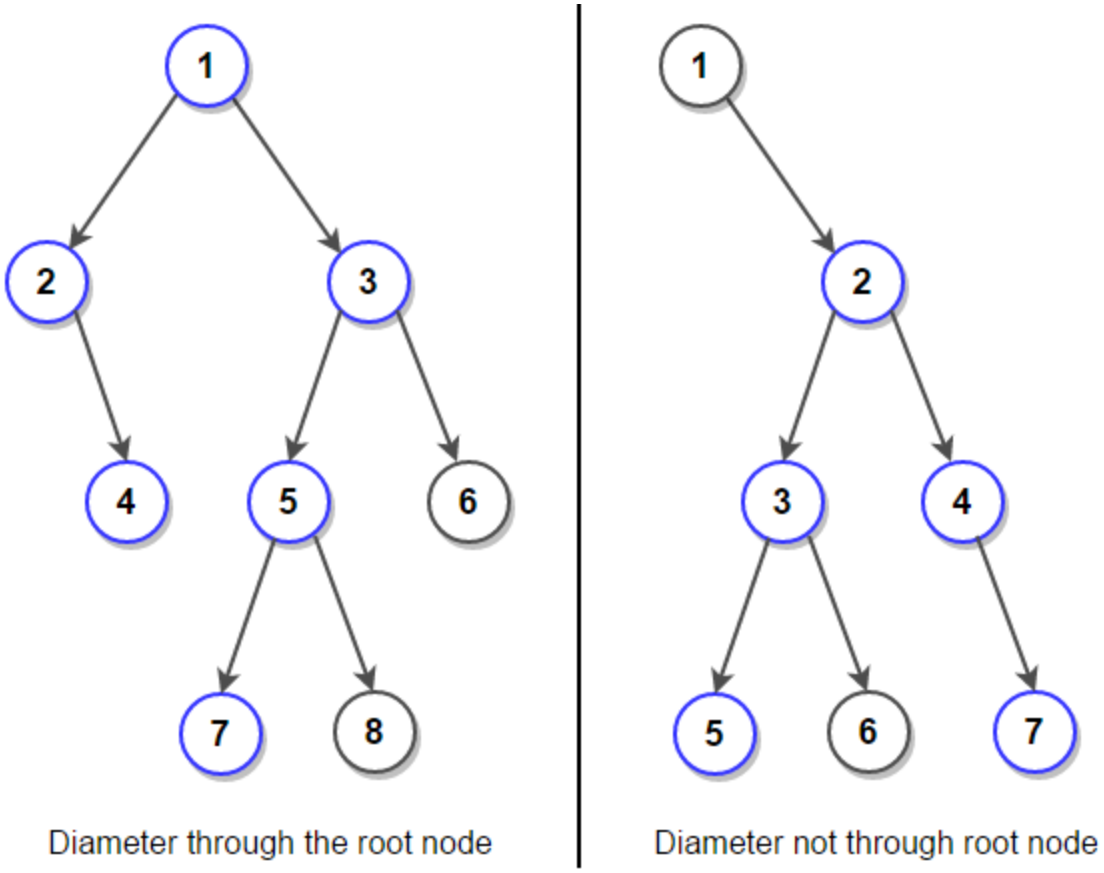
Find the diameter of a binary tree

Given a binary tree, write an efficient algorithm to compute the diameter of it. A binary tree diameter equals the total number of nodes on the longest path between any two leaves in it.

The following figure shows two binary trees with diameters 6 and 5, respectively (nodes highlighted in blue). The binary tree diameter shown on the left side passes through the root node, while the diameter of the binary tree shown on the right side does not pass through the root node.



Practice this problem

A simple solution would be to calculate the left and right subtree's height for each node in the tree. The *maximum node path* that passes through a node will have a value one more than the sum of the height of its left and right subtree. Finally, the diameter is maximum among all *maximum node paths* for every node in the tree. The time complexity of this solution is $O(n^2)$ as there are `n` nodes in the tree, and for every node, we are calculating the height of its left and right subtree that takes $O(n)$ time.

We can solve this problem in linear time by doing a postorder traversal on the tree. Instead of calculating the height of the left and the right subtree for every node in the tree, get the height in constant time. The idea is to start from the bottom of the tree and return the height of the subtree rooted at a given node to its parent. The height of a subtree rooted at any node is one more than the maximum height of the left or right subtree.

The algorithm can be implemented as follows in C++, Java, and Python. Here, we pass diameter by reference to the function (*instead of returning it*) and update its value within the function itself using the left and right subtree height.

C++

```
#include <iostream>
using namespace std;

// Data structure to store a binary tree node
struct Node
{
    int data;
    Node *left, *right;

    Node(int data)
    {
        this->data = data;
        this->left = this->right = nullptr;
    }
};

// Function to find the diameter of the binary tree. Note that the functio
// returns the height of the subtree rooted at a given node, and the diame
// is updated within the function as it is passed by reference
int getDiameter(Node* root, int &diameter)
{
    // base case: tree is empty
    if (root == nullptr) {
        return 0;
    }

    // get heights of left and right subtrees
    int left_height = getDiameter(root->left, diameter);
    int right_height = getDiameter(root->right, diameter);

    // calculate diameter "through" the current node
    int max_diameter = left_height + right_height + 1;

    // update maximum diameter (note that diameter "excluding" the current
    // node in the subtree rooted at the current node is already updated
    // since we are doing postorder traversal)
    diameter = max(diameter, max_diameter);

    // it is important to return the height of the subtree rooted at the c
    return max(left_height, right_height) + 1;
}

int getDiameter(Node* root)
{
    int diameter = 0;
    getDiameter(root, diameter);
```

```
47
48      return diameter;
49  }
50
51  int main()
52  {
53      Node* root = new Node(1);
54      root->left = new Node(2);
55      root->right = new Node(3);
56      root->left->right = new Node(4);
57      root->right->left = new Node(5);
58      root->right->right = new Node(6);
59      root->right->left->left = new Node(7);
60      root->right->left->right = new Node(8);
61
62      cout << "The diameter of the tree is " << getDiameter(root);
63
64      return 0;
65  }
```

Download Run Code

Output:

```
The diameter of the tree is 6
```

Java

Python

The time complexity of the above solution is `O(n)`, where `n` is the total number of nodes in the binary tree. The program requires `O(h)` extra space for the call stack, where `h` is the height of the tree.

Binary Tree

Amazon, Depth-first search, Medium, Microsoft, Recursive