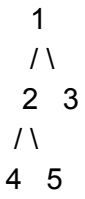
https://leetcode.com/problems/diameter-of-binary-tree/#/description

Question
Given a binary tree, you need to compute the length of the diameter of the tree. The diameter of a binary tree is the length of the **longest** path between any two nodes in a tree. This path may or may not pass through the root.
Example:
Given a binary tree



Return **3**, which is the length of the path [4,2,1,3] or [5,2,1,3].
Note: The length of path between two nodes is represented by the number of edges between them.

Solution

Approach #1 : Brute Force

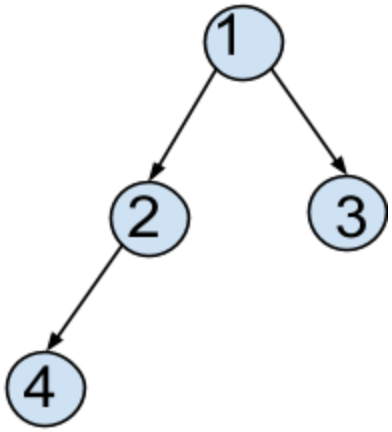
Brute Force Approach would be to list out all possible permutation of paths between two nodes I.e. take all pairs of two nodes and find distance between them and finally print the largest distance.

Complexity Analysis

Taking permutation of two nodes will be O(n^2) and per two node finding distance would be O(n)
So total would be O(n^3).

Approach #2 [Accepted]
Computing height of left and right subtree at each node

Note
height =number of vertices on the *longest path* from the node to a leaf
Or recursively height = max(left subtree height , right subtree height) + 1
Where NULL nodes have height 0 and leaf nodes have height 1
For example:

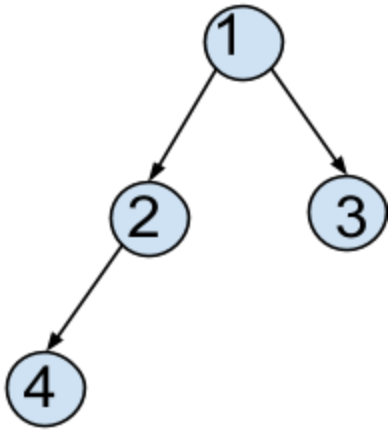


this tree have height as 3 for path 1 - 2 - 4 comprising of three nodes.

[Algorithm]
For each node there would be two cases

1. the diameter passes through the node
2. the diameter doesn't passes through the node

Case 1 : diameter passes through the node
Then diameter would be height of left subtree + height of right subtree



For e.g. in above figure diameter is 3 as node 1 is part of path of diameter which is 4 - 2 - 1 - 3
Diameter = left subtree height (2) + right subtree height (1)

Case 2 : diameter does not passes through the node

If this is the case, then diameter would be
maximum of (diameter of left subtree , diameter of right subtree)
It is because there is some path in lower parts of tree which is longer than including current node.

So, for each node recursive formula would be maximum of three quantity

1. Diameter of left subtree
2. Diameter of right subtree
3. Left subtree height + right subtree height

Where first two point assumes that diameter doesn't passes through that node
And third point assumes that diameter passes through that node.

```
class Solution {
public:

        int height(TreeNode *root)
        {
        if(!root) // if node is null
        return 0;
        return max(height(root->left),height(root->right))+1;
        }

        int diameterOfBinaryTree(TreeNode* root) {

        if(!root)return 0;
        // computing left subtree height and right subtree height at each node
        int lh = height(root->left);
        int rh = height(root->right);

        return
max(max(diameterOfBinaryTree(root->left),diameterOfBinaryTree(root->right)),lh+rh);

        }

};
```

Complexity Analysis
O(n^2)

At each node we are calculating height for left and right subtree
height() -----> O(n)
We are calculating height for each node
So complexity = O(n*n) = O(n^2)

Approach #3 [Accepted]
Storing height at each node
[Algorithm]

Here main concern is calculation of height at each node.
What we can do is to save height calculated so far and use that to calculate height of the parent node. For that we need to return height of child to its parent in recursive call.

As we are calculating diameter by return statement we can pass a variable by reference so that its value is changed and seen by the calling parent.

We pass another variable to the function
dia(node ptr, var height passed by reference)

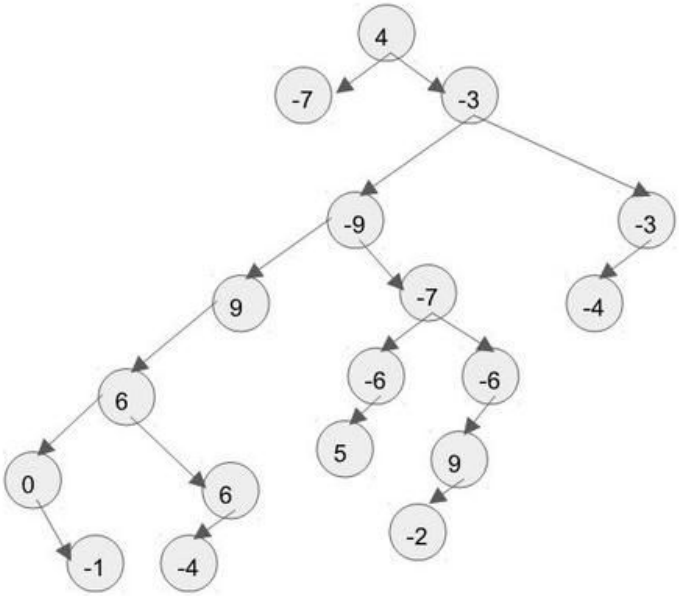
Considering base case for leaf nodes : left child and right child are NULL
Height of left subtree = 0
Height of right subtree = 0
As variable is passed by reference
Left height = 0
Right height = 0
So height = 0+0+1 = 1
As this variable is also passed by reference the parent of leaf node will get its left subtree or right subtree(depending on whether leaf node is left child or right child) height as 1. Same process goes on recursively.

Intuitively, there are basically two variables which are changing with each call

1. Diameter calculated till that node so far (by return statement)
2. Height (by passed by reference variable)

Finally, logic for diameter is same as above algo i.e. maximum of three quantities
1. Diameter of left subtree
2. Diameter of right subtree
3. Left subtree height + right subtree height

Visualization
Test Case
[4,-7,-3,null,null,-9,-3,9,-7,-4,null,6,null,-6,-6,null,null,0,6,5,null,9,null,null,-1,-4,null,null,null,-2]
Output - 8



[Gif here]

```
class Solution {
public:
        int dia(TreeNode * root, int &height)
        {
        int lh=0,rh=0,ld,rd;
        if(!root)
        {
        height=0;
        return 0; // dia = 0
        }
        ld = dia(root->left,lh); // lh passed as a reference
        rd = dia(root->right,rh); // rh passed as a reference
        height = max(lh,rh)+1;

        return max(max(ld,rd),lh+rh);
```

```
        }
        int diameterOfBinaryTree(TreeNode* root) {
        int h=0;
        if(!root)return 0;
        return dia(root,h);
        }
};
```

Complexity Analysis
O(n)
For each node, height is calculated in O(1) and recursion goes for all nodes in the tree.
Therefore complexity is O(n).