AGENDA

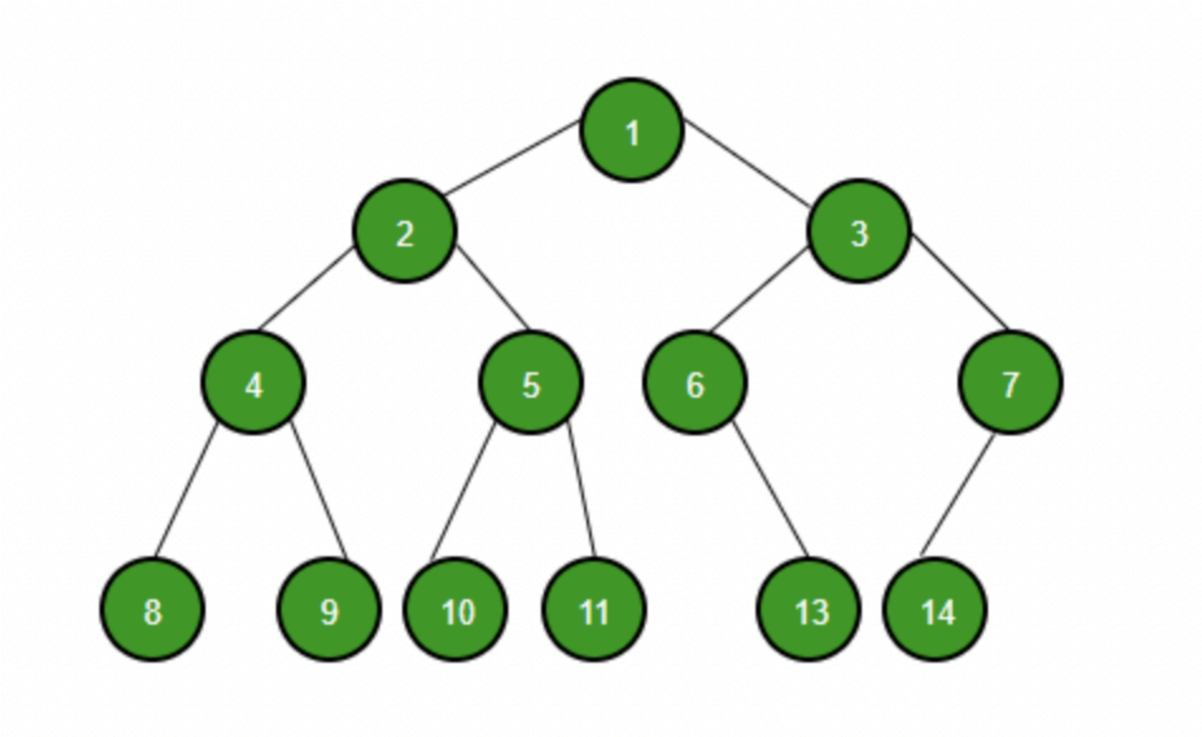
Trees / Binary Trees:

- Introduction
- Implementation
- Tree traversals:
  - Preorder
  - Postorder
  - Inorder
  - Level-order
- Left view / Right view of a tree (Discuss Level-order approach)
- Height of a Binary Tree
- Diameter of a Binary Tree
- Lowest Common Ancestor (LCA)

Introduction

A Tree is a non-linear data structure where each node is connected to a number of nodes with the help of pointers or references.

A Binary Tree is an application of a Tree where each node will have at most two child nodes.



- Root: The root of a tree is the first node of the tree. In the above image, the root node is node 1.
- Edge: An edge is a link connecting any two nodes in the tree. For example, in the above image, there is an edge between nodes 3 and 6.
- Siblings: The child nodes of the same parent are called siblings. That is, the nodes with the same parent are called siblings. In the above tree, nodes 4 and 5 are siblings.
- Leaf Node: A node is said to be the leaf node if it has no children. In the above tree, nodes 8, 9, 10, 11, 13, and 14 are leaf nodes.
- Height of a Tree: The height of a tree is defined as the total number of levels in the tree. The above tree is of height 4.

Full Binary Tree/Complete Binary Tree/Perfect Binary Tree: Read from the course material [IMP]

Implementation

```
class Node {
      int data;
      Node *left;
      Node *right;
}
```

Tree Traversals

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc.), which have only one logical way to traverse them, trees can be traversed in different ways.

Preorder Traversal: In preorder traversal, a node is processed before processing any of the nodes in its subtree.

```
Algorithm Preorder(tree)
   1. Visit the root.
   2. Traverse the left subtree, i.e.,
      call Preorder(left-subtree)
   3. Traverse the right subtree, i.e.,
      call Preorder(right-subtree)
```

Postorder Traversal: In post order traversal, a node is processed after processing all the nodes in its subtrees.

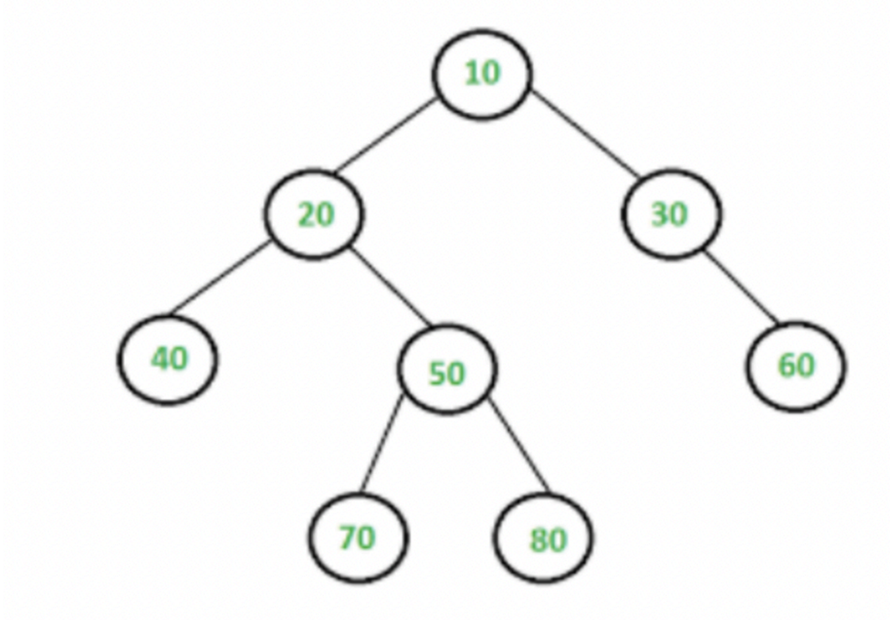
```
Algorithm Postorder(tree)
   1. Traverse the left subtree, i.e.,
      call Postorder(left-subtree)
   2. Traverse the right subtree, i.e.,
      call Postorder(right-subtree)
   3. Visit the root.
```

Inorder Traversal: In Inorder traversal, a node is processed after processing all the nodes in its left subtree. The right subtree of the node is processed after processing the node itself.

```
Algorithm Inorder(tree)
   1. Traverse the left subtree, i.e.,
      call Inorder(left->subtree)
   2. Visit the root.
   3. Traverse the right subtree, i.e.,
      call Inorder(right->subtree)
```

Level order Traversal

Traversing each node of the binary tree, level-by-level.



Level order: 10, 20, 30, 40, 50, 60, 70, 80

[IMP: Discuss how it can be used to find the left-view or the right-view of a binary tree].

Height of a Binary Tree

Given a binary tree, find its height.

Example 1:

```
Input:
     1
    /  \
   2    3
Output: 2
```

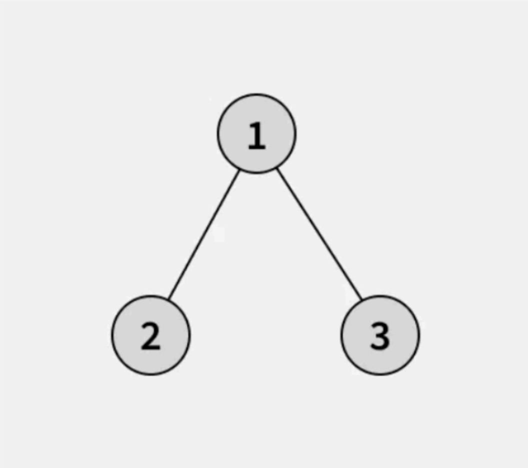
Example 2:

```
Input:
  2
   \
    1
   /
 3
Output: 3
```

Diameter of a Binary Tree

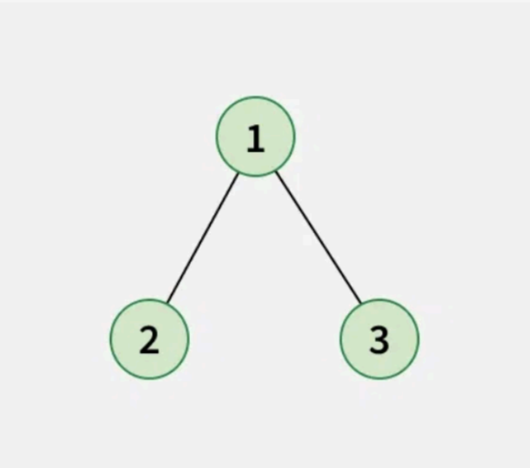
Given a binary tree, the **diameter** (also known as the width) is defined as the number of **edges** on the longest path between two leaf nodes in the tree. This path may or may not pass through the root. Your task is to find the diameter of the tree.

Input: root[] = [1, 2, 3]

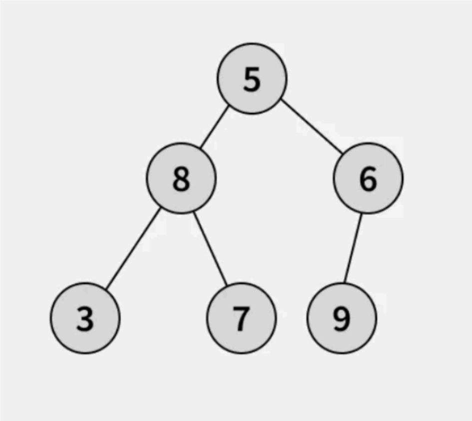


Output: 2

Explanation: The longest path has 2 edges (node 2 -> node 1 -> node 3).

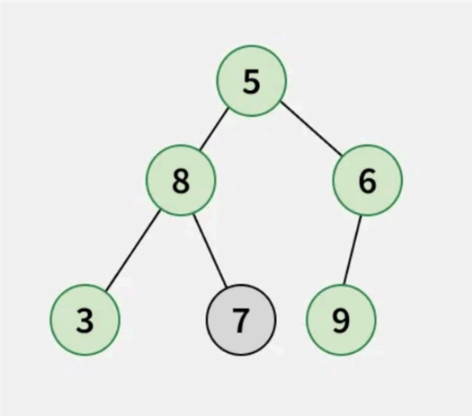


Input: root[] = [5, 8, 6, 3, 7, 9]



Output: 4

Explanation: The longest path has 4 edges (node 3 -> node 8 -> node 5 -> node 6 -> node 9).



Approaches:

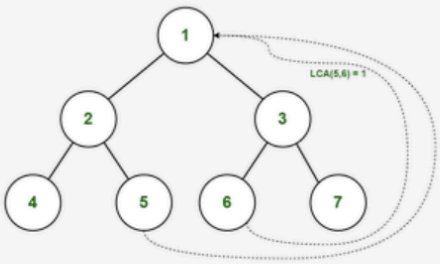
1. Brute force
2. Precompute heights
3. Optimal by modifying the height function

LCA in a Binary Tree

Given a Binary Tree with all **unique** values and two nodes value, **n1** and **n2**. The task is to find the **lowest common ancestor** of the given two nodes. We may assume that either both n1 and n2 are present in the tree or none of them are present.

LCA: It is the first common ancestor of both the nodes n1 and n2 from bottom of tree.

Input: root = [1,2,3,4,5,6,7], n1 = 5 , n2 = 6



Output: 1

Explanation: LCA of 5 and 6 is 1.

Input: root = [5, 2, N, 3, 4], n1 = 3 , n2 = 4

```
      5
     /
    2
   / \
  3   4
```

Output: 2

Explanation: LCA of 3 and 4 is 2.

Input: root = [5, 2, N, 3, 4], n1 = 5 , n2 = 4

```
      5
     /
    2
   / \
  3   4
```

Output: 5

Explanation: LCA of 5 and 4 is 5.