treeNotes

Problem 1.

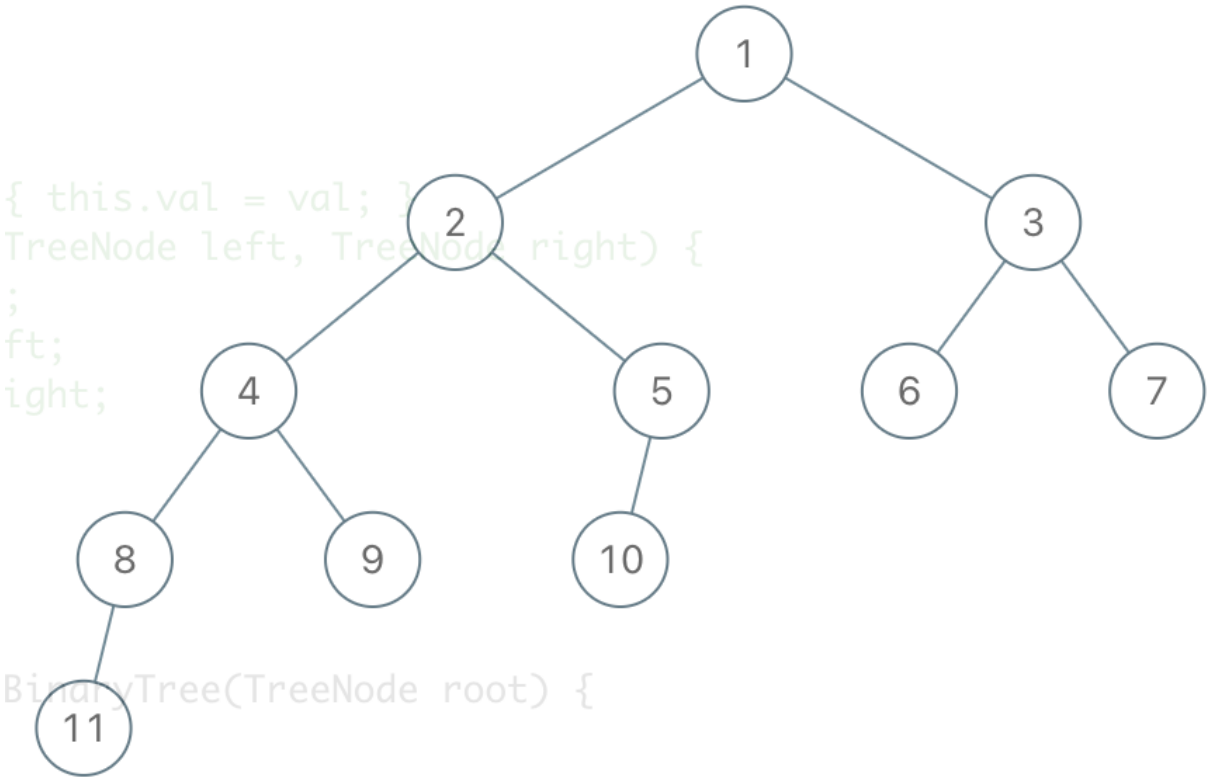
> Diameter of Binary Tree
>
> *Given the `root` of a binary tree, return the length of the **diameter** of the tree. The **diameter** of a binary tree is the **length** of the longest path between any two nodes in a tree. This path may or may not pass through the `root`.*
>
> *The **length** of a path between two nodes is represented by the number of edges between them.*

To solve this problem first we need to get full understanding of the question.

- Diameter may or may not include root node.
- It is the longest path between any two nodes.

Let's assume that at a given node, if we know the height of the left and right subtree, we can find a potential candidate for diameter, including the current node.

Example Tree




```
At Node 2
leftSubTreeHeight = 3
rightSubTreeHeight = 2length of diameter which pass through 2
is leftSubTreeHeight+rightSubTreeHeight which is 5.At Node 1
leftSubTreeHeight = 4
rightSubTreeHeight = 2
length of diameter which pass through 1 is 6You got the idea
```

So before calculating the potential candidate of diameter we need to know the height of left and right subtree. This clearly shows that we need to use post order traversal i.e process left and right subtree first.

Implementation is quite simple.

Observe how we are maintaining a global variable which is getting updated after processing left and right subtree. Going forward you will see that we will be using same pattern to solve all the problems in this article.

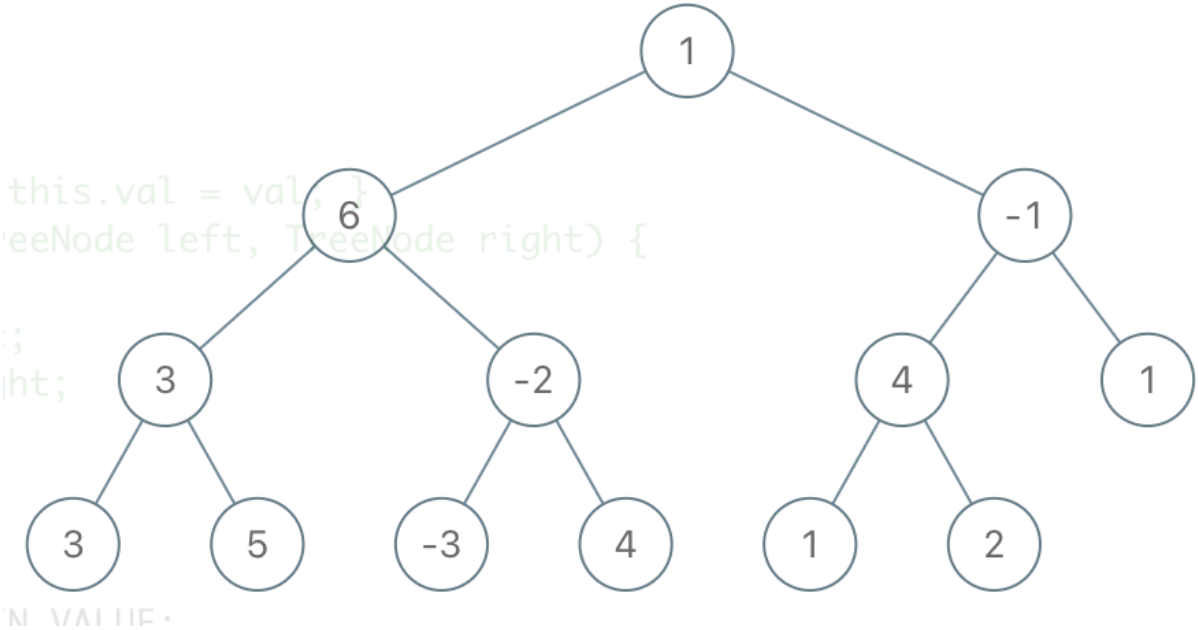
Problem 2

> Binary Tree Maximum Path Sum
>
> *A **path** in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence **at most once**. Note that the path does not need to pass through the root.*
>
> *The **path sum** of a path is the sum of the node's values in the path.*
>
> *Given the `root` of a binary tree, return the maximum **path sum** of any **non-empty** path.*

This problem shares a lot of similarity with previous problem.

Let's assume that at a given node, if we know the maximum linear path sum of the left and right subtree which includes left or right node, we can find a potential candidate for maximum path sum, including the current node or path which includes the current node.

Example




```
At node 6
max linear path sum of left subtree including left node is 8
(3+5)
max linear path sum of left subtree including right node is 2
(-2+4)potential max path sum which includes 6 is 16 (6 + 8 +
2)
```

Again we just need to use post order traversal to find max linear path sum for left and right subtree.

Implementation.

Observe carefully that if maxLinearPathSum is negative we are making it 0 in line no 9 and 10 before assigning it to left and right.

Problem 3

> Validate Binary Search Tree
>
> *Given the `root` of a binary tree, determine if it is a valid binary search tree (BST).*

> *A **valid BST** is defined as follows:*
>
> *The left subtree of a node contains only nodes with keys **less than** the node's key.*
>
> *The right subtree of a node contains only nodes with keys **greater than** the node's key.*
>
> *Both the left and right subtrees must also be binary search trees.*

In actual interview, interviewer might not explain what is binary search tree, he will definitely expect candidate to know the right definition of binary search tree.

Alternative way to define binary search tree.

- root's left and right subtree should be binary search tree and max element in left subtree should be less than root node's value and min element in right subtree should be greater than root node's value.

```
isBST(root) = isBST(root.left) &&
              isBST(root.right) &&
              max(root.left) < root.val &&
              root.val < min(root.right)
```

Let's assume that at a given node, if we know the max of left subtree and min of right subtree and if left subtree and right subtree is BST or not then we will be able to know if current subtree is BST or not.

Here is the implementation.

There is another simpler way to solve this problem by using a different definition of BST.

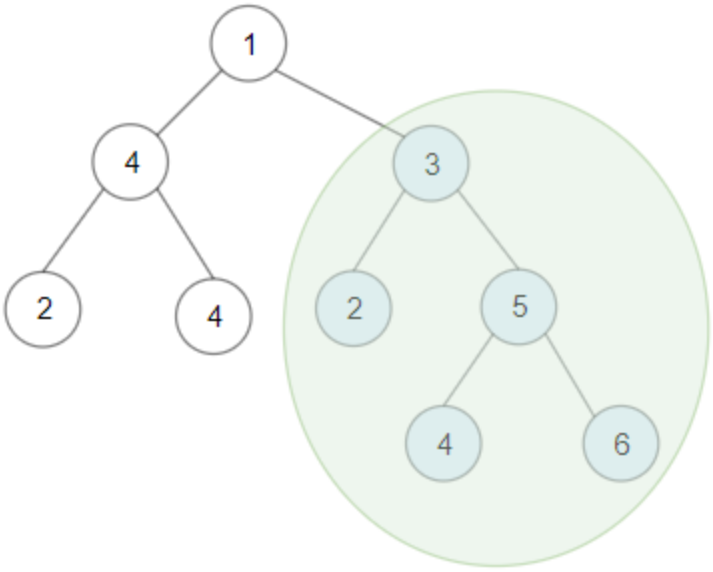
- All the elements in left subtree should be smaller than current node and all the elements in right subtree should be greater than current node.
- Above statement should be valid for all the nodes.

With this definition we can define a range where current node's value should lie.

Here is the implementation of this approach.

Problem 4

> Maximum Sum BST in Binary Tree
>
> *Given a **binary tree** `root`, return the maximum sum of all keys of **any** sub-tree which is also a Binary Search Tree (BST).*
>
> *Assume a BST is defined as follows:*
>
> *The left subtree of a node contains only nodes with keys **less than** the node's key.*
>
> *The right subtree of a node contains only nodes with keys **greater than** the node's key.*
>
> *Both the left and right subtrees must also be binary search trees.*

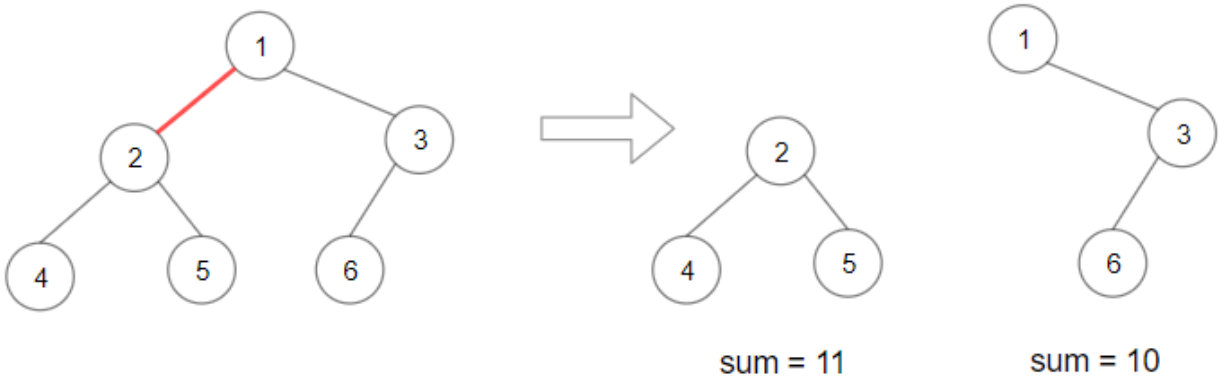


Very similar to previous problem we apart from min, max, isBST we also need to include sum of all the nodes.

Let us straight away look at the implementation.

Problem 5

> Maximum Product of Splitted Binary Tree
>
> *Given the `root` of a binary tree, split the binary tree into two subtrees by removing one edge such that the product of the sums of the subtrees is maximized.*
>
> *Return the maximum product of the sums of the two subtrees. Since the answer may be too large, return it **modulo** `10^9 + 7`.*
>
> ***Note** that you need to maximize the answer before taking the mod and not after taking it.*



Let's assume that at a given node, if we know the sum of left and right subtree then will we know the sum of subtree including current node. If we break the current subtree of the original tree then we will know a potential candidate for maxProduct (sum of elements of current subtree) * (sum of remaining elements)

Cleary we need to apply post order traversal to find sum of node's value also we need to know total sum so that we can calculate remaining sum.

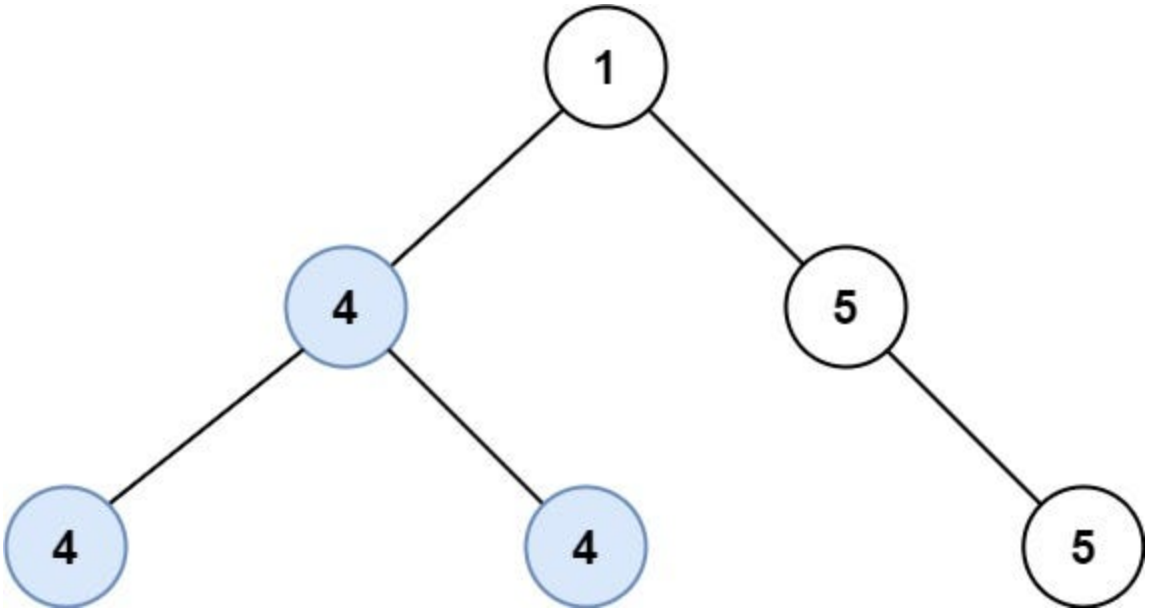
Here is the implementation

Problem 6

> Longest Univalue Path

> *Given the `root` of a binary tree, return the length of the longest path, where each node in the path has the same value. This path may or may not pass through the root.*
>
> ***The length of the path** between two nodes is represented by the number of edges between them.*

Example




```
max path length containing all same element is 2
```

Let's assume that at a given node, if we know max linear path length for left and right subtree which includes left and right node respectively then we can find a potential candidate for max path length including current node.

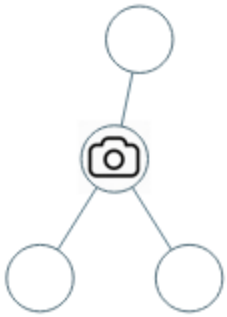
Implementation of this approach.

Problem 7

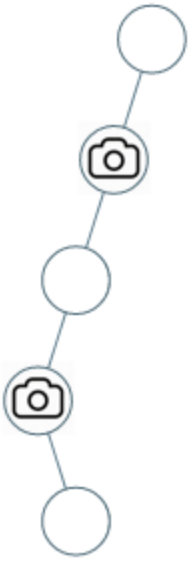
> Binary Tree Cameras

> *You are given the `root` of a binary tree. We install cameras on the tree nodes where each camera at a node can monitor its parent, itself, and its immediate children.*
>
> *Return the minimum number of cameras needed to monitor all nodes of the tree.*

Examples



1 camera needed



2 camera needed

Intuitions

- Each node can have 2 status either it has camera or it is covered by camera.+ Look at first example 2nd node has camera and other 3 nodes are covered by this camera.

Let’s assume that at a given node, if we know whether the left and right node is covered with camera or has camera, then we can take a decision for current node.

- If left or right node has camera then current node is covered.
- if left and right node do not have camera then we need to add camera at current node.

We can maintain 3 type of status for each node.

- has camera
- covered by camera
- need a camera

Implementation.

Summary of this pattern.

- We apply post order traversal.
- We need to take decision at every node based of values returned by processing left and right subtree.
- We maintain a global variable to update results.