Tree Notes

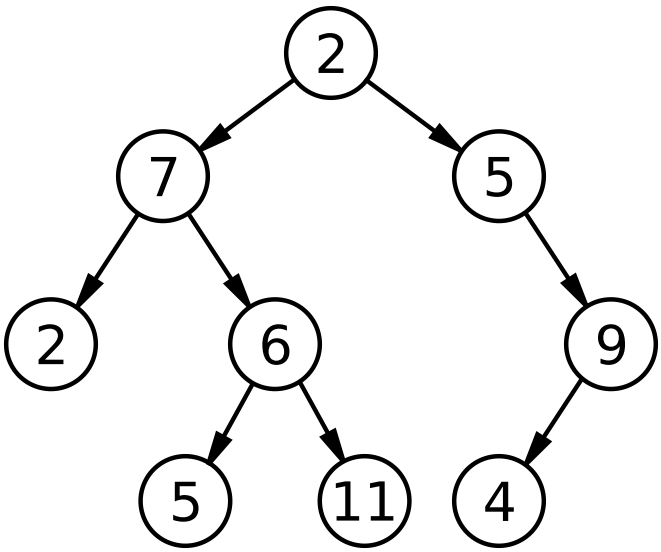
Introduction

- A binary tree is made of nodes, where each node contains a "left" pointer, a "right" pointer, and a data element.
- The "root" pointer points to the topmost node in the tree. The left and right pointers recursively point to smaller "subtrees" on either side
- A null pointer represents a binary tree with no elements -- the empty tree

Structure

```
class Node: def __init__(self, data): self.left = None self.right =
None self.data = data
```

Below is an example of a binary tree

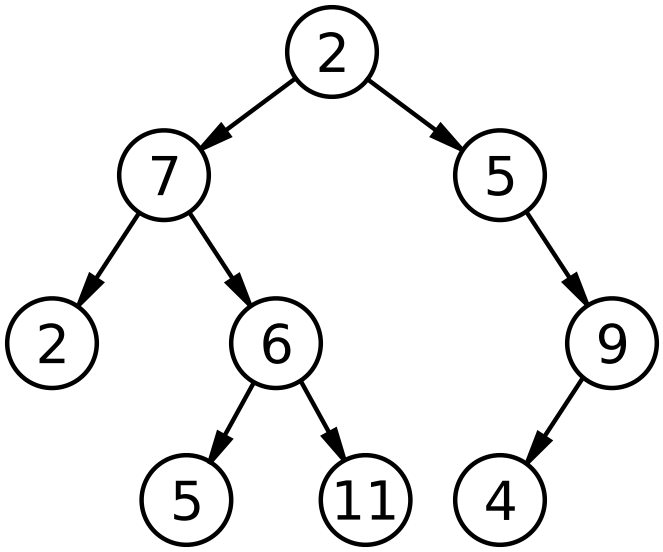


Basic definitions

1. **Root of the binary tree**
    - The topmost node is the root of the tree
    - In the above example, root of the tree is the node with value 2

2. **Leaves**
    - All nodes that have 0 children are leaves
    - In the above example, the leaves are the nodes with values 5, 11 and 4

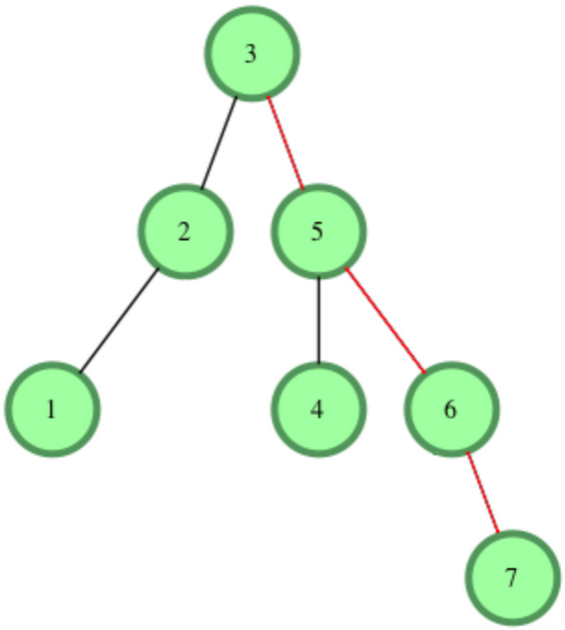
3. **Depth/max depth of a binary tree**
    - The number of nodes along the longest path from the root node down to the farthest leaf node
    - The maxDepth of the empty tree is 0



Depth here is 4 and one such path with maximum depth is 2→5→9→4

4. **Height of a binary tree**

- The height of a binary tree is the number of edges between the tree's root and its furthest leaf. For example, the following binary tree is of height



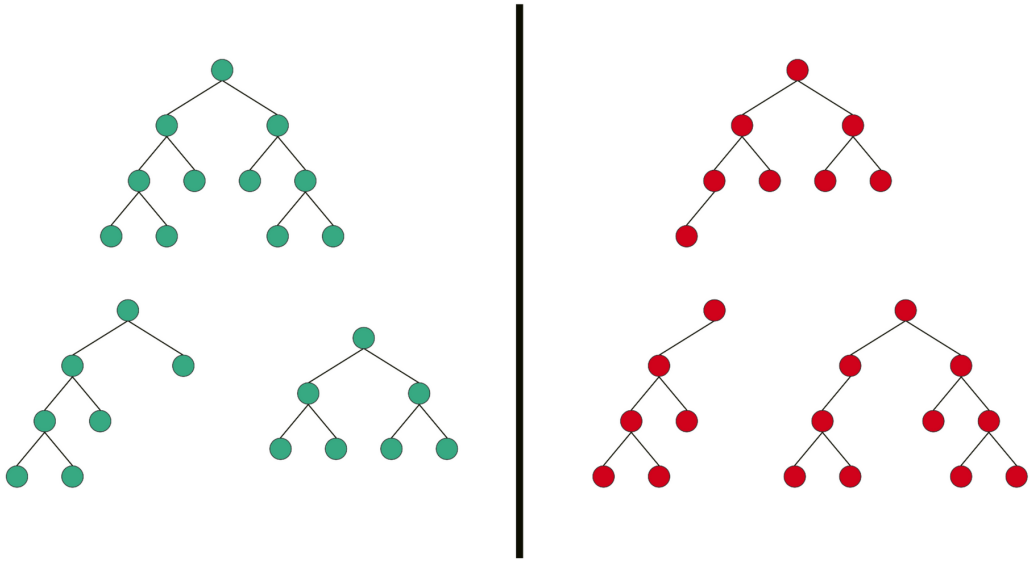
There are 4 nodes in this path that are connected by 3 edges, meaning our binary tree's $height = 3$.

- The height of a tree with a single node is 0

Some basic types of binary trees

1. **Full Binary Tree**

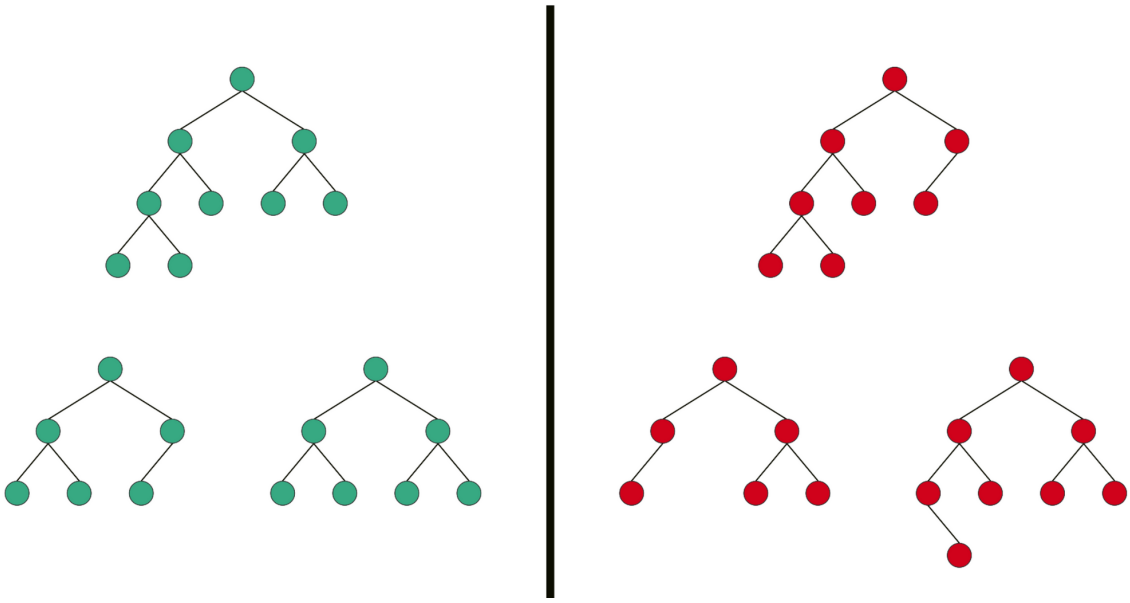
- Full Binary Tree is a Binary Tree in which every node has 0 or 2 children



RED - INVALID, GREEN - VALID

2. **Complete Binary Tree**

- Complete Binary Tree has all levels completely filled with nodes except the last level and in the last level, all the nodes are as left side as possible.

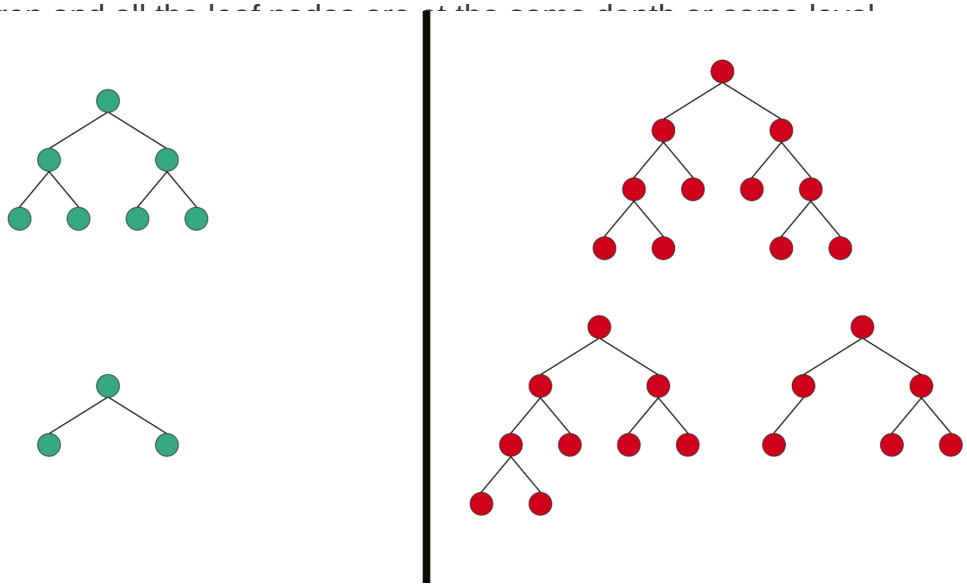


RED - INVALID, GREEN - VALID

3. **Perfect Binary Tree**

- Perfect Binary Tree is a Binary Tree in which all internal nodes have 2

children and all the leaf nodes are at the same depth or same level.

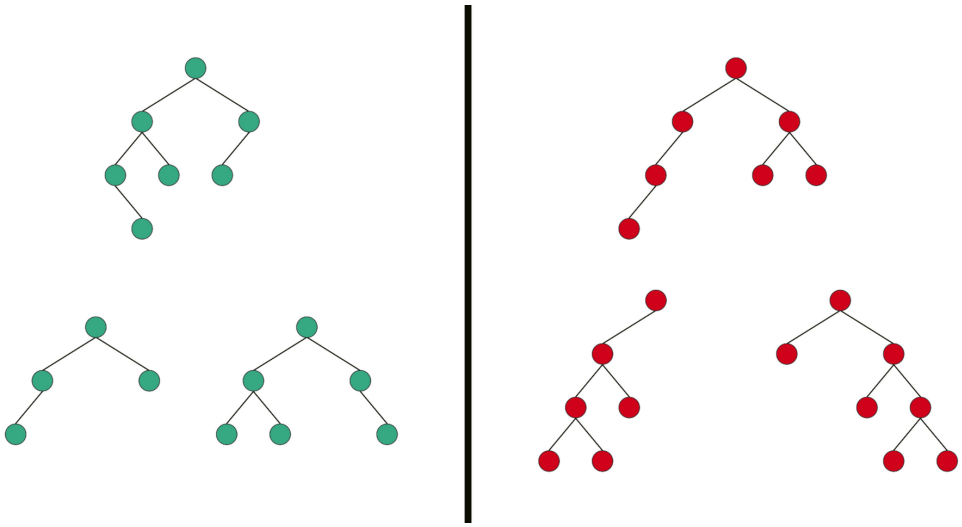


RED - INVALID, GREEN - VALID

- **IMPORTANT** - Total number of nodes in a Perfect Binary Tree with depth **D** is $2^d—1$

4. Balanced Binary Tree

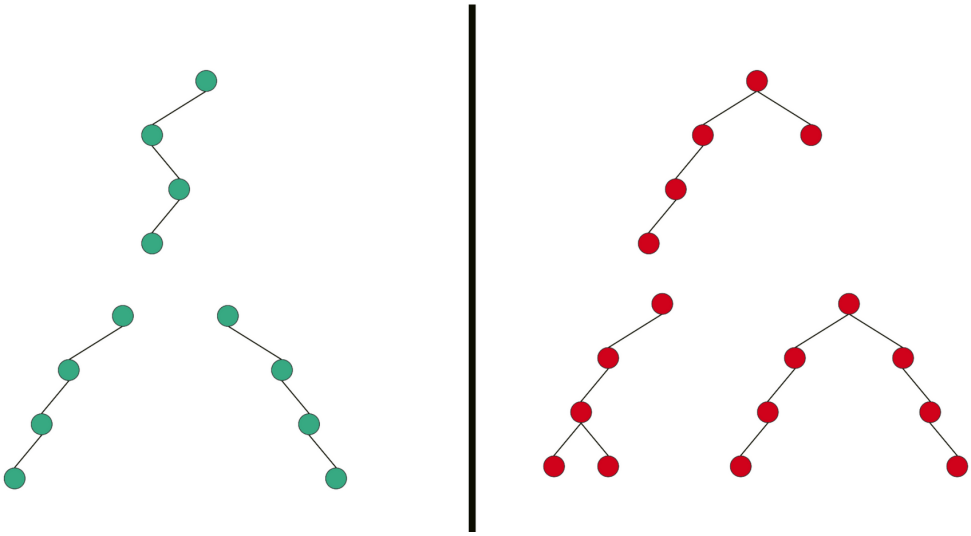
- Balanced Binary Tree is a Binary tree in which height of the left and the right sub-trees of every node may differ by at most 1. This property must be true at every single node



RED - INVALID, GREEN - VALID

5. Degenerate Binary Tree

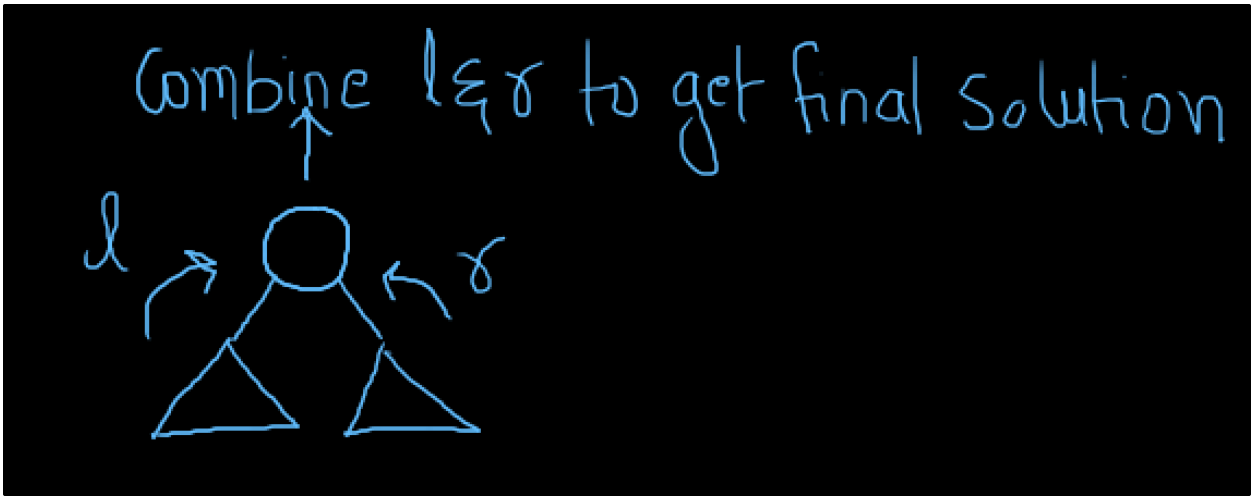
- Degenerate Binary Tree is a Binary Tree where every parent node has only one child node.



- **IMPORTANT** - Height of a Degenerate Binary tree **is equal to Total number of nodes** in that tree.

Basic binary tree problem solving technique

- Most binary tree problems can be solved recursively
- If you carefully look at the structure above, the left and right subtrees are also binary trees
- So if we solve a given problem for the left and subtrees, we can use these solutions and combine them to get a solution for the overall binary tree

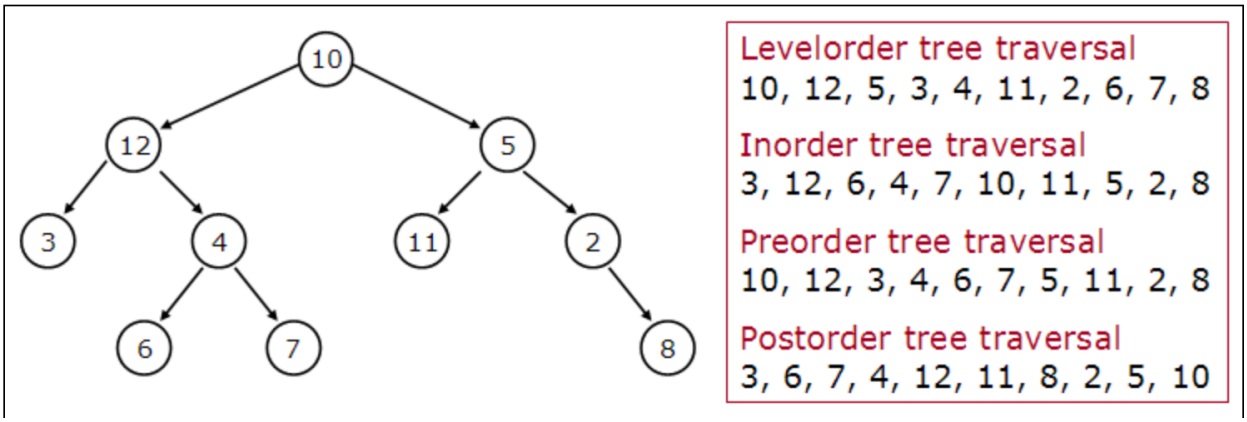


- The above image tries to illustrate this idea of problem solving
- We will explore a few problems with this technique

Binary tree traversals

- Any process for visiting all of the nodes in some order is called a traversal.
- The following are some important tree traversals

1. **Preorder Traversal**
   - Sequence - **Visit Node, then visit left subtree, then visit right subtree**
2. **Inorder Traversal**
   - Sequence - **Visit left subtree, then visit node, then visit right subtree**
3. **Postorder Traversal**
   - Sequence - **Visit left subtree, then visit right subtree, then visit node**
4. **Level order Traversal**
   - Sequence - **Visit all nodes level by level from top to bottom**
   - In each level, you can traverse the nodes either from left to right or right to left (in the example below, its been done from left to right in each level)

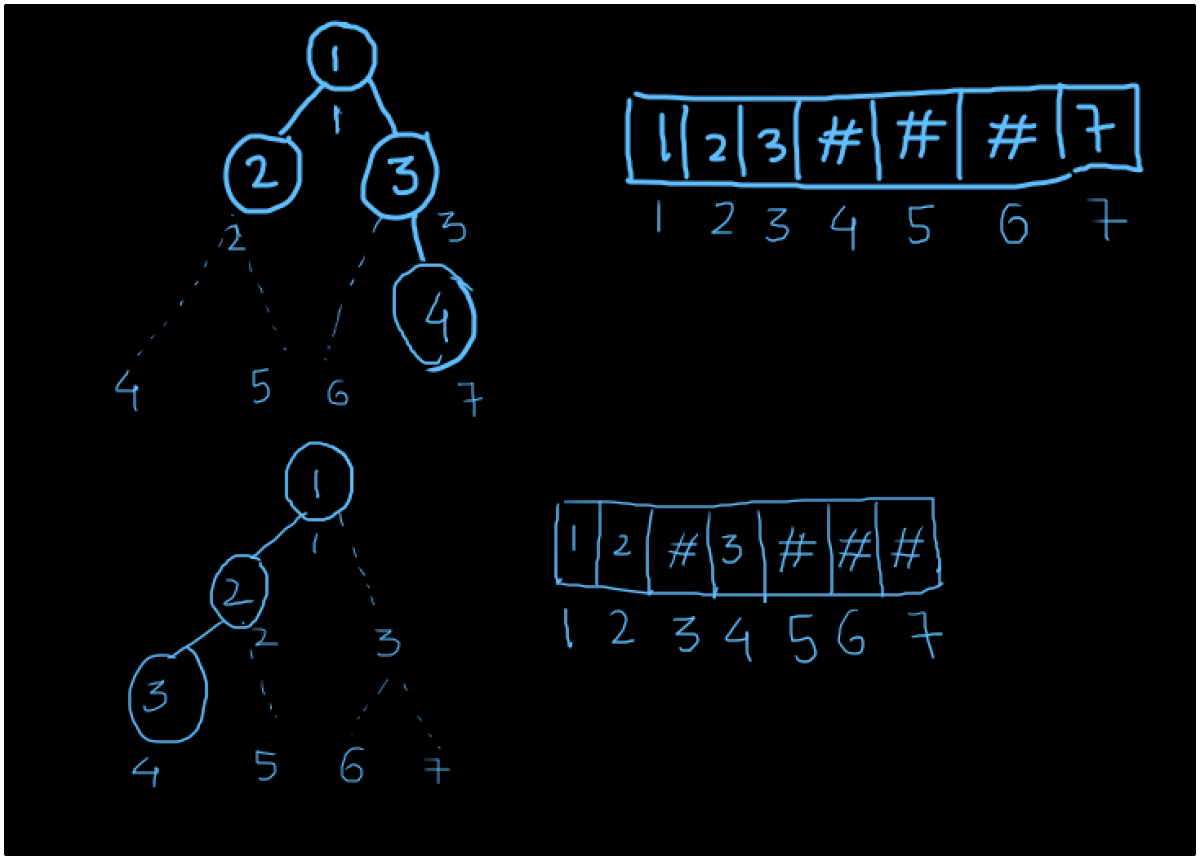


Array representation of binary trees

- Till now, we have just seen binary trees as a collection of nodes where each node has a left pointer, a right pointer, and a value field
- But, binary trees can also be represented using arrays

- Consider an array **A**
  - The root of the tree is **A[0]**
  - For any element at index i
    - Left child = element at index $2 * i + 1$
    - Right child = element at index $2 * i + 2$
  - Another option is, for **1-indexed arrays**
    - Left child = element at index $2 * i$
    - Right child = element at index $2 * i + 1$

- **Examples for 1 indexed arrays**



  - This type of numbering for nodes(1,2,3...note that this node numbering is diff from the node value) is sometimes helpful in solving certain problems efficiently
  - Ex - https://leetcode.com/problems/maximum-width-of-binary-tree/

Example problems based on l,r pattern

```
# counting number of nodes in a tree def countTreeNodes(root): if root
== None: return 0 else: l = countTreeNodes(root.left) r =
countTreeNodes(root.right) return l + r + 1
```

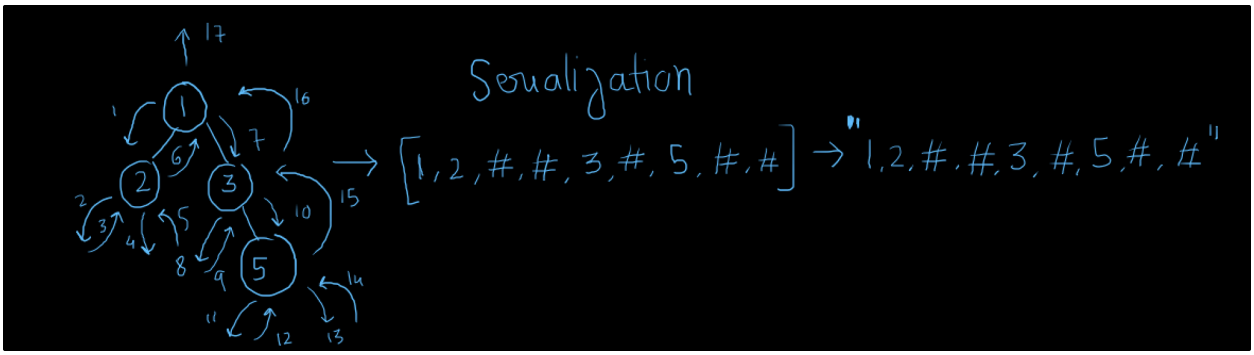
```
# finding depth of a tree def depthOfTree(root): if root == None:
return 0 else: l = depthOfTree(root.left) r = depthOfTree(root.right)
return max(l,r) + 1
```

```
# inorder recursive def inOrderTraversal(root): if root == None:
return else: inOrderTraversal(root.left) print(root.val)
inOrderTraversal(root.right)
```

```
# LNR # inorder iterative def inOrderIterative(root): if root == None:
return stack = [(root, False)] while len(stack) > 0: curr, visited =
stack.pop() if visited: print(curr) else: if curr.right != None:
stack.append((curr.right, False)) stack.append((curr, True)) if
curr.left != None: stack.append((curr.left, False)) return ans
```

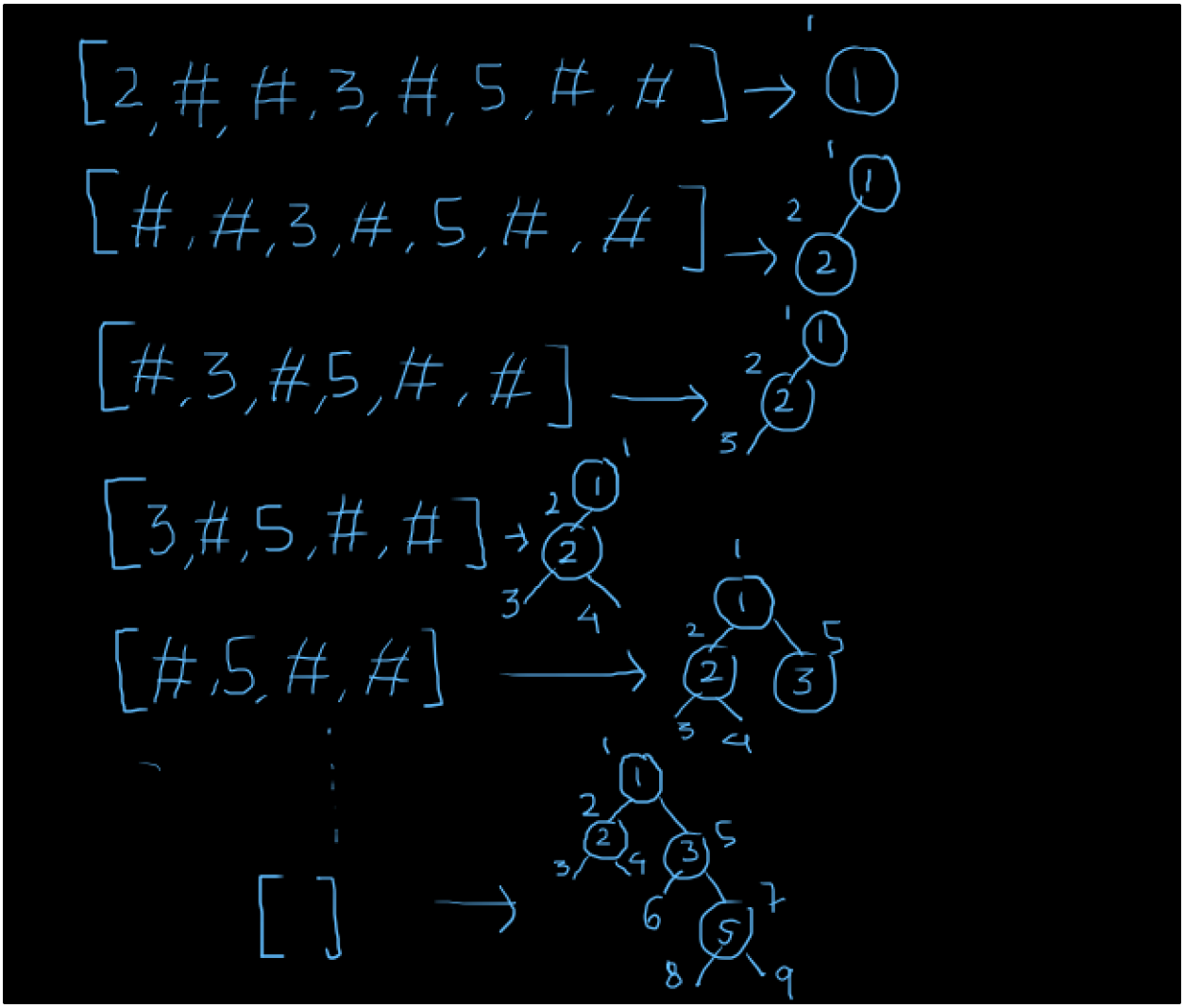
Serialization and deserialization of a binary tree

- The main idea here is to convert a binary tree into a string and to be able to retrieve the binary tree back from that string
- Preorder traversal is one very easy way of doing it



- # can be used to denote None

- Now, using this preorder traversal, we can convert it back to the binary tree using a deque/queue



- Note - the numbers on top of the nodes indicate the way in which the nodes are created and pointers are assigned