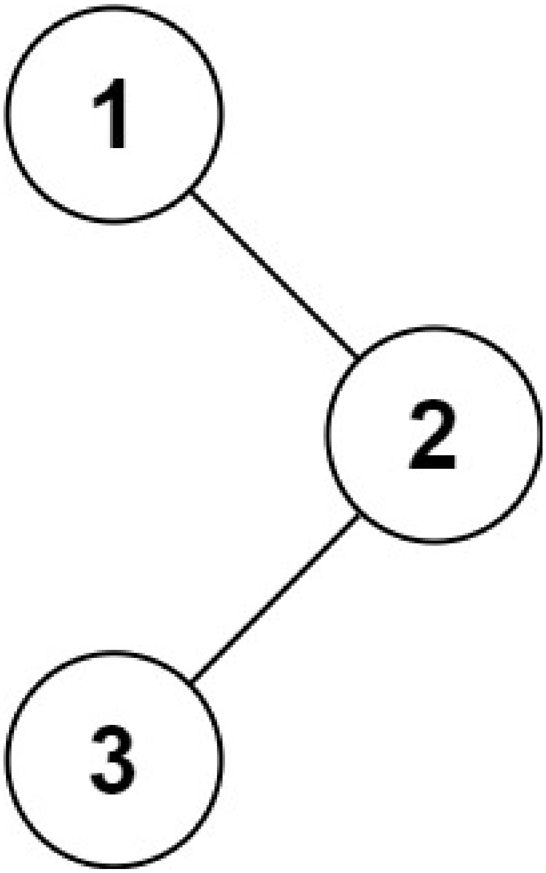
Module6-HW

June 18, 2024

Problem1-LeetcodeQ 94-Binary Tree Inorder Traversal-Easy

Given the root of a binary tree, return the inorder traversal of its nodes' values.



Example 1:

- Input: root = [1,null,2,3]
- Output: [1,3,2]

Example 2:

- Input: root = []
- Output: []

Example 3:

- Input: root = [1]
- Output: [1]

Constraints:

- The number of nodes in the tree is in the range [0, 100].
- -100 <= Node.val <= 100

0.0.1 Q94. Psuedocode

[17]:

```
function inorder_traversal(root)
initialize empty list res
initialize empty stack
set p to root
while p is not null or stack is not empty
    while p is not null
        push p to stack
        set p to p.left
    set p to stack.pop()
    append p.val to res
    set p to p.right
return res
```

```
  Cell In[17], line 1
    function inorder_traversal(root)
             ^

SyntaxError: invalid syntax
```

0.0.2 Q94. code.py

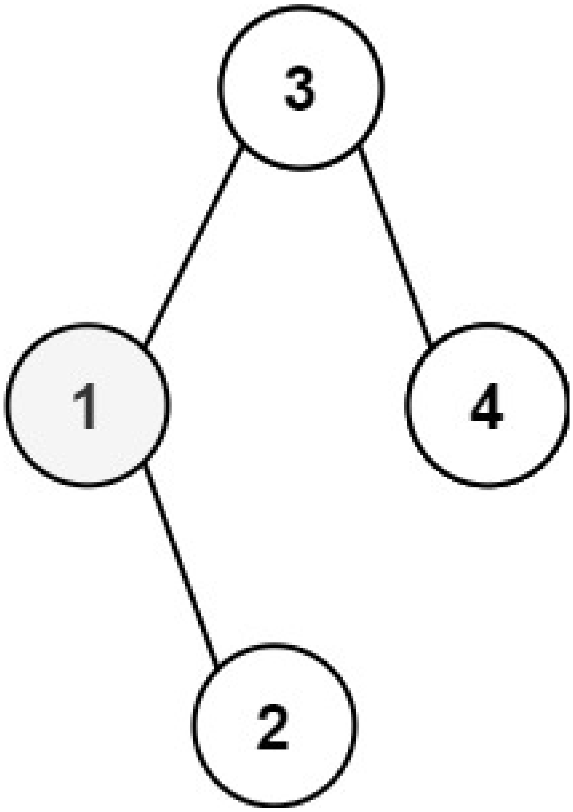
[18]:

```
from typing import List, Optional
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def inorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        stack = []
        p = root
        while p or stack:
            while p:
                stack.append(p)
                p = p.left
            p = stack.pop()
            res.append(p.val)
            p = p.right
        return res

# Creating the test case:
root = TreeNode(1)
root.right = TreeNode(2)
root.right.left = TreeNode(3)

# Creating an instance of Solution and testing the method
solution = Solution()
output = solution.inorderTraversal(root)
print(output)  # Output should be [1, 3, 2]
```

```
[1, 3, 2]
```

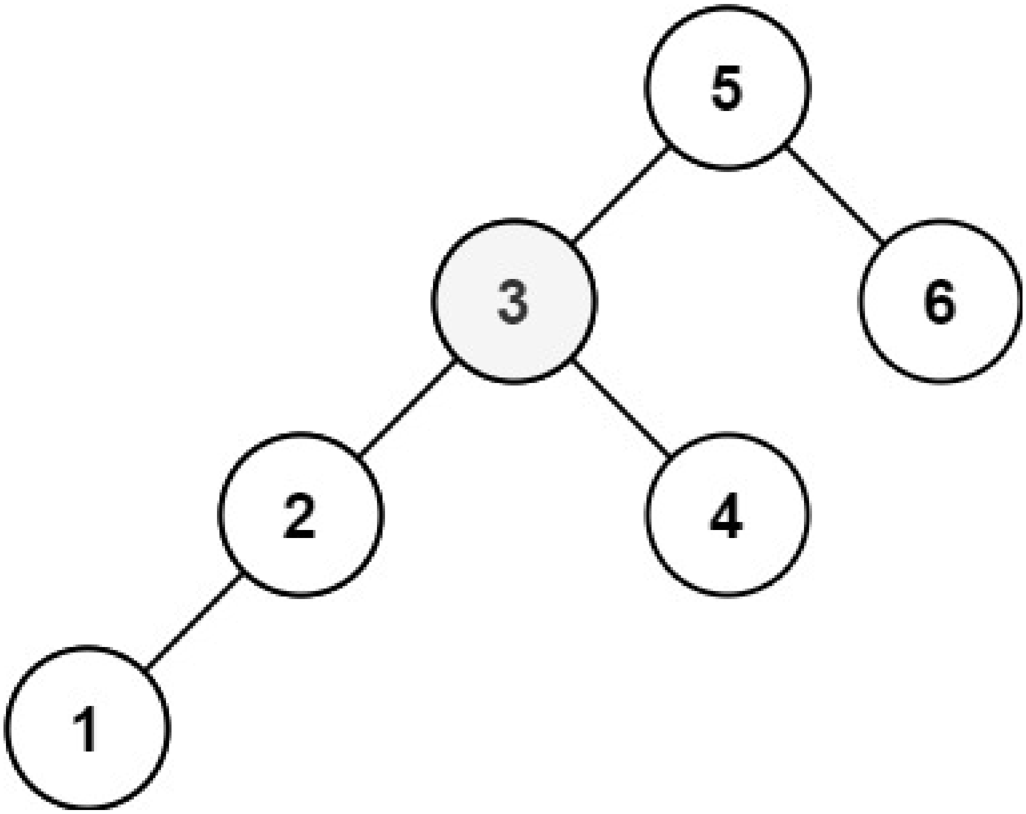
Problem2-LeetcodeQ 230-Kth Smallest Element ina BST-Medium

Given the root of a binary search tree, and an integer k, return the kth smallest value (1-indexed) of all the values of the nodes in the tree.



Example 1:

- Input: root = [3,1,4,null,2], k = 1
- Output: 1



Example 2:

- Input: root = [5,3,6,2,4,null,null,1], k = 3
- Output: 3

0.0.3 Q230. psuedocode

[19]:
```
define inner function dfs(root)
    if root is null, return
    call dfs(root.left)
    if k is equal to 0, return
    decrement k by 1
    if k is equal to 0, set res to root.val
    call dfs(root.right)
set k to input k
call dfs(root)
return res
```

```
  Cell In[19], line 1
    define inner function dfs(root)
           ^
SyntaxError: invalid syntax
```

0.0.4 Q230. code.py

[8]:

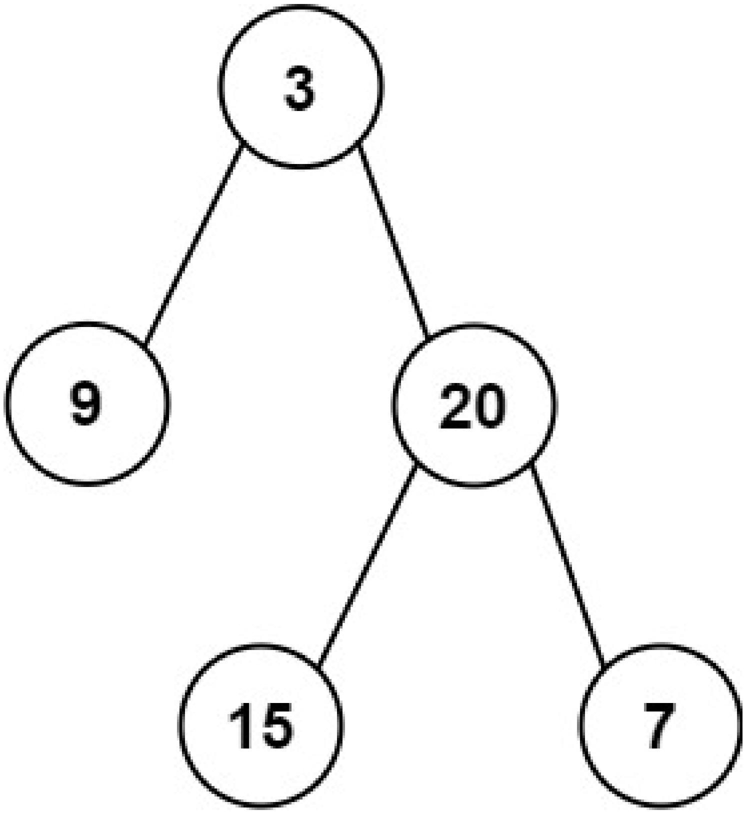
```python
from typing import Optional

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def kthSmallest(self, root: Optional[TreeNode], k: int) -> int:
        def dfs(root):
            if not root:
                return
            dfs(root.left)
            if self.k == 0:
                return
            self.k -= 1
            if self.k == 0:
                self.res = root.val
            dfs(root.right)

        self.k = k
        dfs(root)
        return self.res


root = TreeNode(3)
root.left = TreeNode(1)
root.right = TreeNode(4)
root.left.right = TreeNode(2)

# Creating an instance of Solution and testing the method
solution = Solution()
output = solution.kthSmallest(root, 1)
print(output)
```

```
1
```

Problem3-LeetcodeQ 105-Construct Binary Tree from Preorder and Inorder Traversal-Medium

Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree.

Example 1:



- Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
- Output: [3,9,20,null,null,15,7]

Example 2:

- Input: preorder = [-1], inorder = [-1]
- Output: [-1]

0.0.5 Q150. pseudocode

The algorithm works by leveraging the properties of preorder and inorder tree traversals to reconstruct the binary tree.

In a preorder traversal, the first element is always the root of the tree. The inorder traversal helps determine the structure of the tree by providing the relative positions of the nodes with respect to the root.

The algorithm begins by identifying the root from the first element of the preorder list. It then finds the root's position in the inorder list, which divides the inorder list into left and right subtrees. The left subtree consists of elements before the root in the inorder list, and the right subtree consists of elements after the root. The algorithm recursively applies the same logic to the left and right subtrees, using appropriate slices of the preorder and inorder lists. This recursive process continues until the entire tree is reconstructed.

The correctness is guaranteed because each recursive call precisely identifies the root and splits the tree into left and right subtrees based on the consistent properties of preorder and inorder traversals.

[ ]:

```
function build_tree(preorder, inorder)
if preorder is empty
    return null
set left_size to the index of preorder[0] in inorder
set left to the result of calling build_tree with
    preorder from index 1 to 1 + left_size and
    inorder from start to left_size
set right to the result of calling build_tree with
    preorder from 1 + left_size to end and
    inorder from 1 + left_size to end
return a new TreeNode with value preorder[0], left, and right
```

0.0.6 Q150. code.py

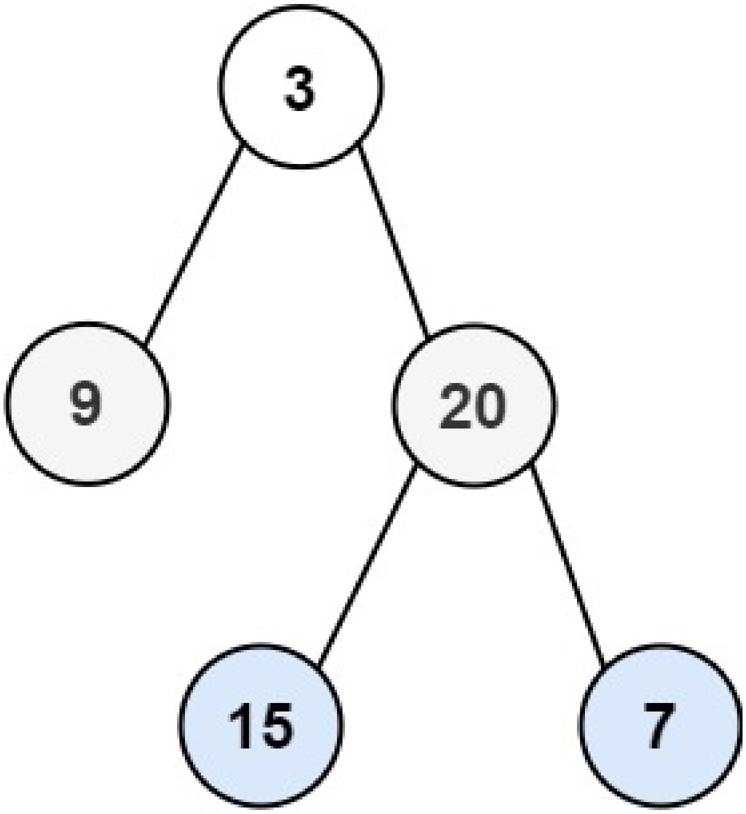
[11]:

```python
from typing import List, Optional

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) -> Optional[TreeNode]:
        if not preorder:
            return None
        left_size = inorder.index(preorder[0])
        left = self.buildTree(preorder[1: 1 + left_size], inorder[:left_size])
        right = self.buildTree(preorder[1 + left_size:], inorder[1 + left_size:])
        return TreeNode(preorder[0], left, right)

##########################################################################
from collections import deque

def print_tree(root):
    if not root:
        return []
    result = []
    queue = deque([root])
    while queue:
        node = queue.popleft()
        if node:
            result.append(node.val)
            queue.append(node.left)
            queue.append(node.right)
        else:
            result.append(None)
    # Remove trailing None values
    while result and result[-1] is None:
        result.pop()
    return result

preorder = [3, 9, 20, 15, 7]
inorder = [9, 3, 15, 20, 7]
solution = Solution()
root = solution.buildTree(preorder, inorder)

output = print_tree(root)
print(output)
```

```
[3, 9, 20, None, None, 15, 7]
```

Problem4-LeetcodeQ. 102-Binary Tree Level Order Traversal-Medium

Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).



Example 1:

- Input: root = [3,9,20,null,null,15,7]
- Output: [[3],[9,20],[15,7]]

Example 2:

- Input: root = [1]
- Output: [[1]]

Example 3:

- Input: root = []
- Output: []

0.0.7 Q102. pseudocode

[ ]:

```
function level_order(root)
if root is null
    return empty list
initialize empty list ans
initialize deque with root
while deque is not empty
    initialize empty list vals
    for each element in deque
        remove node from deque
        append node.val to vals
        if node.left is not null, append node.left to deque
        if node.right is not null, append node.right to deque
    append vals to ans
return ans
```

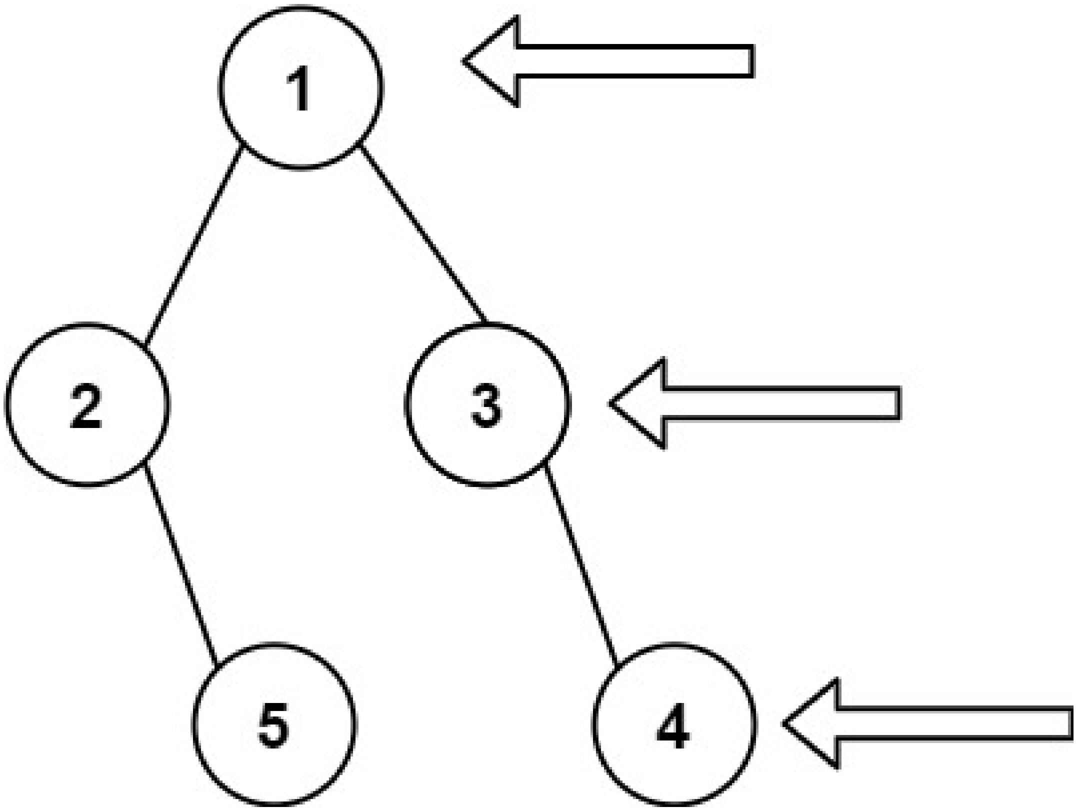
0.0.8 Q102 code.py

[14]:

```python
from typing import List, Optional
from collections import deque

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        if root is None:
            return []
        ans = []
        q = deque([root])
        while q:
            vals = []
            for _ in range(len(q)):
                node = q.popleft()
                vals.append(node.val)
                if node.left:
                    q.append(node.left)
                if node.right:
                    q.append(node.right)
            ans.append(vals)
        return ans


root = TreeNode(3)
root.left = TreeNode(9)
root.right = TreeNode(20)
root.right.left = TreeNode(15)
root.right.right = TreeNode(7)

solution = Solution()
output = solution.levelOrder(root)
print(output)
```

```
[[3], [9, 20], [15, 7]]
```

Problem5-LeetcodeQ 199-Binary Tree Right Side View-Medium



Example 1: - Input: root = [1,2,3,null,5,null,4] - Output: [1,3,4]

Example 2:

- Input: root = [1,null,3]

- Output: [1,3]

Example 3:

- Input: root = []
- Output: []

0.0.9 Q199. pseudocode

[ ]:

```
function right_side_view(root)
initialize empty list res
define inner function dfs(root, depth)
    if root is null, return
    if length of res is less than or equal to depth
        append 0 to res
    set res[depth] to root.val
    call dfs(root.left, depth + 1)
    call dfs(root.right, depth + 1)
call dfs with root and depth 0
return res
```

0.0.10 Q199. code.py

[20]:

```python
from typing import List, Optional

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def rightSideView(self, root: Optional[TreeNode]) -> List[int]:
        res = []
        def dfs(root, depth):
            if not root:
                return
            if len(res) <= depth:
                res.append(0)
            res[depth] = root.val
            dfs(root.left, depth + 1)
            dfs(root.right, depth + 1)

        dfs(root, 0)
        return res


from collections import deque
```

```
def print_tree(root):
    if not root:
        return []
    result = []
    queue = deque([root])
    while queue:
        node = queue.popleft()
        if node:
            result.append(node.val)
            queue.append(node.left)
            queue.append(node.right)
        else:
            result.append(None)
    # Remove trailing None values
    while result and result[-1] is None:
        result.pop()
    return result


root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.right = TreeNode(5)
root.right.right = TreeNode(4)

# Creating an instance of Solution and testing the method
solution = Solution()
output = solution.rightSideView(root)
print(output)
```

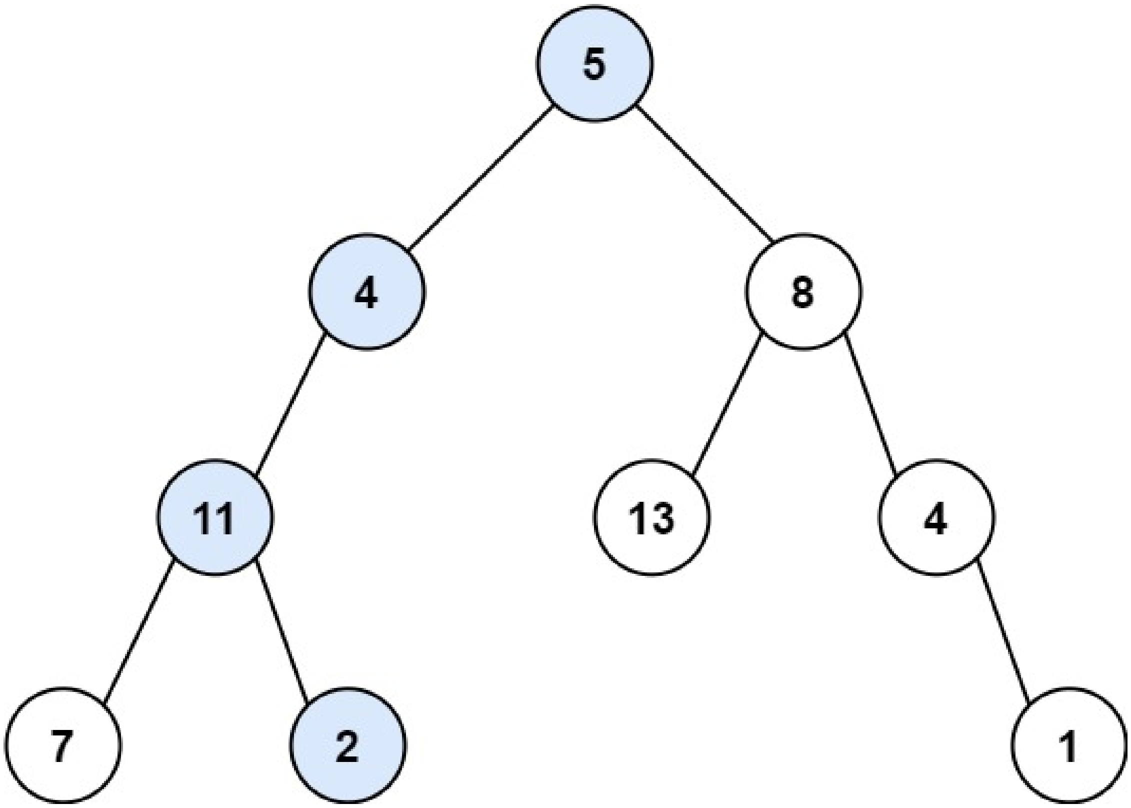
```
[1, 3, 4]
```

Problem6-LeetcodeQ 112-Path Sum-Easy

Given the root of a binary tree and an integer targetSum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals targetSum.

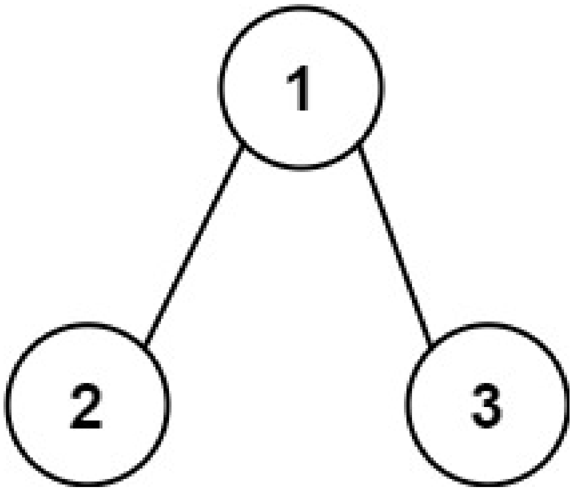
A ***leaf** is a node with no children.

Example 1:



- Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
- Output: true
- Explanation: The root-to-leaf path with the target sum is shown.

Example 2:



- Input: root = [1,2,3], targetSum = 5
- Output: false
- Explanation: There two root-to-leaf paths in the tree:
- − (1 –> 2): The sum is 3.
- − (1 –> 3): The sum is 4.

- - There is no root-to-leaf path with sum = 5.

Example 3:

- Input: root = [], targetSum = 0
- Output: false
- Explanation: Since the tree is empty, there are no root-to-leaf paths.

0.0.11 Q112. pseudocode

[ ]:

```
function has_path_sum(root, target_sum)
if root is null
    return false
subtract root.val from target_sum
if root.left is null and root.right is null
    return target_sum is equal to 0
return has_path_sum(root.left, target_sum) or has_path_sum(root.right, target_sum)
```

0.0.12 Q112. code.py

[24]:

```python
from typing import Optional

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def hasPathSum(self, root: Optional[TreeNode], targetSum: int) -> bool:
        if root is None:
            return False
        targetSum -= root.val
        if root.left is None and root.right is None:
            return targetSum == 0
        return self.hasPathSum(root.left, targetSum) or self.hasPathSum(root.right, targetSum)


from collections import deque

def print_tree(root):
    if not root:
        return []
    result = []
    queue = deque([root])
    while queue:
        node = queue.popleft()
        if node:
            result.append(node.val)
            queue.append(node.left)
            queue.append(node.right)
        else:
            result.append(None)
    # Remove trailing None values
    while result and result[-1] is None:
        result.pop()
    return result


root = TreeNode(5)
root.left = TreeNode(4)
root.right = TreeNode(8)
root.left.left = TreeNode(11)
root.left.left.left = TreeNode(7)
root.left.left.right = TreeNode(2)
root.right.left = TreeNode(13)
root.right.right = TreeNode(4)
root.right.right.right = TreeNode(1)

solution = Solution()
output = solution.hasPathSum(root, 22)
print(output)
```

```
True
```

Problem7-LeetcodeQ 78-Subsets-Medium

Given an integer array nums of unique elements, return all possible subsets (the power set).

The solution set must not contain duplicate subsets. Return the solution in any order.

Example 1:

- Input: nums = [1,2,3]
- Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]

Example 2:

- Input: nums = [0]
- Output: [[],[0]]

Note: All the numbers of nums are **unique**.

0.0.13 Q78. pseudocode

[ ]:

```
function subsets(nums)
initialize empty list ans
initialize empty list path
set n to length of nums

define inner function dfs(i)
    if i is equal to n
        append a copy of path to ans
        return
    call dfs(i + 1)
    append nums[i] to path
    call dfs(i + 1)
    remove the last element from path

call dfs(0)
return ans

#######################
ans is the list of all subsets to be returned.
path is the current subset being constructed.
```

0.0.14 Q78. code.py

[27]:

```python
from typing import List

class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        ans = []
        path = []
        n = len(nums)

        def dfs(i: int) -> None:
            if i == n:
                ans.append(path.copy())
                return
            dfs(i + 1)
            path.append(nums[i])
            dfs(i + 1)
            path.pop()

        dfs(0)
        return ans


nums = [1, 2, 3]
```

```
solution = Solution()
output = solution.subsets(nums)
print(output)
```

```
[[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]
```

Problem8-LeetcodeQ 39-Combination Sum-Medium

Given an array of distinct integers candidates and a target integer target, return a list of all unique combinations of candidates where the chosen numbers sum to target. You may return the combinations in any order.

The same number may be chosen from candidates an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

Example 1:

- Input: candidates = [2,3,6,7], target = 7
- Output: [[2,2,3],[7]]
- Explanation:
- − 2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
- − 7 is a candidate, and 7 = 7.
- − These are the only two combinations.

Example 2:

- Input: candidates = [2,3,5], target = 8
- Output: [[2,2,2,2],[2,3,3],[3,5]]

Example 3:

- Input: candidates = [2], target = 1
- Output: []

Constraints:

- 1 <= candidates.length <= 30
- 2 <= candidates[i] <= 40
- All elements of candidates are distinct.
- 1 <= target <= 40

0.0.15 Q39. pseudocode

[ ]:

```
function combination_sum(candidates, target)
sort candidates in ascending order
initialize empty list ans
initialize empty list path

define inner function dfs(i, left)
if left is 0
```

```
    append a copy of path to ans
    return
if i is equal to length of candidates or left is less than candidates[i]
    return

call dfs(i + 1, left)

append candidates[i] to path
call dfs(i, left - candidates[i])
remove the last element from path

call dfs(0, target)
return ans

#########################
candidates is sorted in ascending order to facilitate early termination of the␣
  ↪DFS when the remaining target is less than the smallest candidate.
ans is the list to store all valid combinations.
path is the current combination being constructed.
```

0.0.16 Q39. code.py

[29]:

```python
from typing import List

class Solution:
    def combinationSum(self, candidates: List[int], target: int) ->␣
  ↪List[List[int]]:
        candidates.sort()
        ans = []
        path = []

        def dfs(i: int, left: int) -> None:
            if left == 0:
                ans.append(path.copy())
                return
            if i == len(candidates) or left < candidates[i]:
                return

            dfs(i + 1, left)

            path.append(candidates[i])
            dfs(i, left - candidates[i])
            path.pop()

        dfs(0, target)
        return ans
```

```python
candidates = [2, 3, 5]
target = 8

solution = Solution()
output = solution.combinationSum(candidates, target)
print(output)
```

```
[[3, 5], [2, 3, 3], [2, 2, 2, 2]]
```