Problem Description

element of the array corresponds to the root level, and subsequent elements correspond to the succeeding levels of the tree.

we visit all the nodes at the current level before moving on to the nodes of the next level.

515. Find Largest Value in Each Tree Row

In this problem, we are provided with the root of a binary tree. Our task is to find the largest value in each row of the tree and return them in an array. The binary tree is not necessarily a binary search tree, which means that the tree is not guaranteed to follow the ordering property of binary search trees. Each row in the array corresponds to a level of the tree, indexed starting from 0. The first

Intuition

Here is how we can proceed:

the root itself. 2. While the queue is not empty, we will perform the following steps for each level:

1. Start with the root node by adding it to a queue. Since we only have the root, the largest value for the root level is the value of

To solve this problem, a common approach is to perform a level order traversal of the binary tree. A level order traversal means that

 Initialize a variable to keep track of the maximum value for the current level. Initially, it can be set to negative infinity to ensure that any node value will be larger.

- Get the number of nodes in the current level (the size of the queue). Iterate through these nodes and for each node:
- Add the left and right children of the current node to the queue if they exist.

Update the maximum value if the current node's value is greater.

- 3. After finishing the iteration for the current level, add the maximum value we found to the answer array. 4. Repeat the process for the next level.
- 5. Once the queue is empty, it means we have visited all levels and obtained the largest value for each level. The answer array is now complete, and we return it as the result.
- This approach ensures that for each level, we are able to assess all the nodes, and identify the largest value, thereby constructing our array of the largest values for each level.
- 1. Breadth-First Search (BFS): The level order traversal is a type of BFS, in which nodes are visited level by level from top to bottom. This traversal is suitable for finding the largest value in each row since it processes all nodes at each level before moving on to the next one.

The provided solution code implements the level order traversal approach to solve the problem. It uses the following algorithms, data

2. Queue Data Structure: The code utilizes a deque (double-ended queue) from Python's collections module to efficiently perform the BFS. A queue is ideal for BFS since it follows the First-In-First-Out (FIFO) principle, which ensures that nodes are processed in the order they are discovered.

the queue. 4. Tracking Maximum Values: The variable t is used to keep track of the maximum value for the current level. It's initially set to -

while q:

10

11

12

13

14

21

22

23

24

25

26

27

28

Here's a step-by-step breakdown of the code implementation:

Solution Approach

structures, and patterns:

3. Looping through Levels: The algorithm uses a while loop to keep processing nodes as long as the queue is not empty. Inside this loop, there is another for loop that iterates over each node of the current level, which is determined by the current size of

inf (negative infinity) to ensure any node's value will be greater, and hence it can be updated as soon as a node with a value is

- encountered. 5. Adding Children to Queue: While processing each node, the code checks whether the left or right child exists. If they do, these children are added to the queue for future processing. This step is crucial to expand the traversal to subsequent levels.
- 1 # The TreeNode structure is predefined; each node has a value and links to left and right children. class Solution: def largestValues(self, root: Optional[TreeNode]) -> List[int]: if root is None: # If the tree is empty, there are no values to process, return an empty list. return []

q = deque([root]) # Initialize the deque with the root node to start the level order traversal.

for _ in range(len(q)): # 'for' loop runs for the number of nodes in the current level.

After processing the current level, the largest value is appended to the answer.

ans = [] # This list will store the largest values for each level.

Return the array containing the largest values for each level.

Let's consider a small binary tree as our example to illustrate the solution approach:

2. Now, we begin iterating level by level. Our queue currently contains the root:

t = -inf # Set the maximum value for current level to negative infinity.

Continue the loop as long as there are nodes to process.

Process each node of the current level.

q.append(node.right)

if node.right:

ans.append(t)

return ans

15 node = q.popleft() # Dequeue the front node of the queue. t = max(t, node.val) # Update the maximum value if node's value is greater. 16 17 # Enqueue children of the current node if they exist. 18 if node.left: 19 20 q.append(node.left)

This implementation ensures that the traversal continues as long as there are nodes left to be processed. The largest value for each

1. First, we initiate the traversal by adding the root node (which has the value 3) to a queue. Since this is the only node at level 0,

3. We remove the root node from the queue and inspect its children. Since the root node has two children, with values 1 and 5, we

add these two nodes to the queue. The largest value at this level (level 0) is still 3, so we store 3 in our answer array.

4. For the next level (level 1), both nodes in the queue are processed. We start by setting the maximum value for this level to

5. We dequeue the first node (with value 1) and compare it with our max value t. Since 1 is greater than -inf, t becomes 1. This

8. We repeat this process for the next level, iterating over the remaining nodes in the queue (0, 4, and 7). We'll find that 7 is the

9. Our process continues until the queue is empty, which means we've processed all the levels and found the maximum value for

This example demonstrates how the approach processes each level of the binary tree and finds the largest value for each level,

level is determined by comparing it with the default minimum value of -inf, and updating it with the node's value if it's greater. By

the end of the traversal, the ans list contains the largest values for each level, which is the desired output of the problem.

Now, let's walk through the solution step-by-step, applying the level order traversal approach described above.

1 Queue: [3]

1 Queue: [1, 5]

negative infinity:

1 Queue: [5, 0]

1 Queue: [0, 4, 7]

each.

1 Answer: [3, 5, 7]

class TreeNode:

class Solution:

10

12

13

14

20

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

43

45

46

9

11

12

13

14

16

21

23

24

25

26

27

28

29

30

31

33

34

35

36

15 }

*/

Java Solution

class TreeNode {

int val;

class Solution {

/**

*/

TreeNode left;

TreeNode() {}

TreeNode right;

this.val = val;

this.left = left;

this.right = right;

if (root == null) {

TreeNode *left;

13 class Solution {

9

11 };

14 public:

10

12

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

52

10

11

13

14

22

23

24

25

26

28

29

30

31

34

35

36

37

38

39

40

41

43

42 }

12 }

/**

51 };

TreeNode *right;

TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

// If the tree is empty, return an empty vector

vector<int> largestValues(TreeNode* root) {

// Queue for holding nodes to process

// Vector to store the largest values

queue<TreeNode*> nodes queue;

vector<int> largest_values;

// Process nodes level by level

nodes_queue.pop();

return largest_values;

this.val = val === undefined ? 0 : val;

this.left = left === undefined ? null : left;

this.right = right === undefined ? null : right;

while (!nodes_queue.empty()) {

if (!root) return {};

nodes_queue.push(root);

// Method to find the largest value in each level of a binary tree

int current_level_largest = nodes_queue.front()->val;

TreeNode* current_node = nodes_queue.front();

// Remove the processed node from the queue

largest_values.push_back(current_level_largest);

// The TreeNode class represents a node in a binary tree with a numeric value.

constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null) {

const node = nodeQueue.shift(); // Retrieves and removes the head of the queue

// Append the largest value for this level to the results

// Update the largest value if the current node is greater

if (current_node->left) nodes_queue.push(current_node->left);

if (current_node->right) nodes_queue.push(current_node->right);

// Iterate through nodes in the current level

// Get the next node in the queue

for (int i = nodes_queue.size(); i > 0; --i) {

TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}

// Initialize the current level's largest value with the first node's value

current_level_largest = max(current_level_largest, current_node->val);

// Add child nodes of the current node to the queue for the next level

return largestValues;

/**

completing the task successfully.

from collections import deque

self.val = val

self.left = left

if root is None:

return []

queue = deque([root])

largest_values = []

while queue:

self.right = right

from typing import List, Optional

Definition for a binary tree node.

2 Answer: [3]

the largest value for this level is 3.

Example Walkthrough

1 t = -inf

node has a left child with value 0, which we enqueue:

with values 4 and 7, which we enqueue:

maximum value at this level (level 2).

10. Finally, we return the completed ans array as the result:

- 6. The next node with value 5 is dequeued and compared with t. Since 5 is greater than 1, t becomes 5. This node has two children
- 7. With all nodes at level 1 processed, we update our answer array with the maximum value found at this level, which is 5: 1 Answer: [3, 5]
- Python Solution
- 15 16 :param root: Optional[TreeNode] - root of the binary tree 17 :return: List[int] - a list containing the largest value of each level 18 # If the root is none, return an empty list as there are no levels in the tree. 19

Traverse the tree level by level.

max_value = float('-inf')

if node.left:

return largest_values

* Definition for a binary tree node.

TreeNode(int val) { this.val = val; }

TreeNode(int val, TreeNode left, TreeNode right) {

* Class containing the method to find the largest values in each tree row.

* @return A list of the largest values in each row of the tree.

* Finds the largest values in each row of a binary tree.

* @param root The root of the binary tree.

public List<Integer> largestValues(TreeNode root) {

// List to store the largest values in each row.

List<Integer> largestValues = new ArrayList<>();

// Return the empty list if the tree is empty.

// Queue to perform level order traversal

for _ in range(len(queue)):

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

def largestValues(self, root: Optional[TreeNode]) -> List[int]:

Iterate over all nodes at the current level.

Return the list of largest values, one from each level.

Function to find the largest value in each level of a binary tree.

Initialize a queue for BFS traversal and a list to store the answers.

Initialize the largest element of the current level to negative infinity.

If left child exists, append it to the queue for the next level.

node = queue.popleft() # Get the next node from the queue.

37 queue.append(node.left) 38 # If right child exists, append it to the queue for the next level. 39 if node.right: 40 queue.append(node.right) # After visiting all nodes at the current level, add the largest value to the results list. 41 42 largest_values.append(max_value)

max_value = max(max_value, node.val) # Update the largest value found in this level.

```
37
           Deque<TreeNode> queue = new ArrayDeque<>();
38
           // Start the traversal with the root node.
39
           queue.offer(root);
40
41
42
           // Loop until the queue is empty, indicating all levels have been visited.
           while (!queue.isEmpty()) {
43
               // Initialize the variable to store the max value for the current level.
44
45
               int maxInLevel = queue.peek().val;
46
               // Traverse all nodes at this level.
47
               for (int i = queue.size(); i > 0; --i) {
48
                   // Get the next node from the queue.
49
50
                    TreeNode node = queue.poll();
51
52
                   // Update the max value with the max of current max and node's value.
53
                   maxInLevel = Math.max(maxInLevel, node.val);
54
55
                   // Add the left child to the queue if it exists.
56
                   if (node.left != null) {
                        queue.offer(node.left);
57
58
59
                   // Add the right child to the queue if it exists.
60
                   if (node.right != null) {
61
62
                        queue.offer(node.right);
63
64
65
66
               // After finishing the level, add the max value found to the result list.
                largestValues.add(maxInLevel);
67
68
69
70
           // Return the list of maximum values found.
           return largestValues;
74
C++ Solution
  1 #include <vector>
  2 #include <queue>
    // Definition for a binary tree node.
  5 struct TreeNode {
         int val;
```

* Finds the largest value in each level of a binary tree. * @param {TreeNode | null} root - The root of the binary tree. * @returns {number[]} An array containing the largest values from each level. function findLargestValues(root: TreeNode | null): number[] { const largestValues: number[] = []; const nodeQueue: TreeNode[] = [];

Typescript Solution

left: TreeNode | null;

right: TreeNode | null;

if (root) nodeQueue.push(root);

while (nodeQueue.length > 0) {

let levelMax = -Infinity;

largestValues.push(levelMax);

if (node) {

return largestValues;

const queueLength = nodeQueue.length;

for (let i = 0; i < queueLength; i++) {</pre>

levelMax = Math.max(levelMax, node.val);

if (node.left) nodeQueue.push(node.left);

if (node.right) nodeQueue.push(node.right);

class TreeNode {

val: number;

```
Time and Space Complexity
Time Complexity
The time complexity of the provided code is O(n), where n is the number of nodes in the binary tree. This is because the code uses a
breadth-first search (BFS) algorithm that visits each node exactly once to find the largest value in each level of the tree.
```

Space Complexity The space complexity is also O(n). This is the worst-case scenario where the tree is extremely unbalanced, and all nodes are on a single side of the tree, leading to all nodes being held in the queue at one point in time. For a balanced tree, however, the largest space consumption would occur at the level with the most nodes, which is approximately n/2 nodes for a full binary tree at the last level, but it is still expressed as O(n) in Big O notation for simplicity as it remains proportional to the number of nodes.