Documentation: Find Largest Value in Each Tree Row

1. **Problem Statement**

   Given the root of a binary tree, return an array where each element is the largest value found in each row (or level) of the tree, indexed from 0.

   Example:
   Input: root = [1,3,2,5,3,null,9]
   Output: [1,3,9]

2. **Intuition**

   In a binary tree, nodes are organized in levels. To find the largest value in each row, we can traverse the tree level-by-level and track the maximum value for each level. This is naturally suited to a Breadth-First Search (BFS) approach, which processes nodes one level at a time.

3. **Key Observations**

   - Each level of the tree can be processed independently.
   - BFS traversal inherently groups nodes level-wise.
   - Keeping track of the maximum value during the traversal of each level is straightforward.
   - The tree might be empty, so handle the empty input case.

4. **Approach**

   - Use a queue to perform a BFS traversal starting from the root.
   - For each level:
     - Determine the number of nodes (size of the queue).
     - Traverse all nodes at this level, updating the maximum value.
     - Add their children to the queue for the next level.
   - Append the maximum value found at each level to the result list.
   - Return the result list after traversing all levels.

5. **Edge Cases**

- Empty tree (root = None): Return an empty list.
- Tree with only one node: Result is a list with one element, the node's value.
- Tree with negative values: Ensure maximum tracking handles negative numbers correctly.
- Large tree depth: Use efficient traversal to avoid stack overflow and optimize performance.

6. **Complexity Analysis**

Time Complexity

- O(N) where N is the number of nodes in the tree.
- Each node is processed exactly once during BFS.

Space Complexity

- O(M) where M is the maximum number of nodes at any level (width of the tree).
- This corresponds to the maximum size of the queue during traversal.

7. **Alternative Approaches**

- Depth-First Search (DFS):
  Recursively traverse the tree, passing the current level as a parameter. Update a list of maximum values per level during the recursion.
  This approach is also valid but requires careful management of the current level and max values.
- Using two queues:
  Use two queues to alternate between levels, though this adds more complexity and does not improve efficiency.

8. Test Cases

Test Case	Input	Expected Output	Explanation
Empty tree	root = None	[]	No nodes, return empty list
Single node	root = [7]	[7]	Only one node, max is node's value
Balanced tree	root = [1,3,2,5,3,null,9]	[1,3,9]	Largest values per row from example
All negative values	root = [-1,-2,-3,-4,null,null,-5]	[-1,-2,-4]	Handles negative numbers correctly
Left skewed tree	root = [1,2,null,3,null,4]	[1,2,3,4]	Single chain down left, max per row is node

9. Final Thoughts

- BFS is a natural fit for level-wise tree problems like this.
- The solution is efficient in both time and space.
- Handling edge cases like empty trees and negative values is important.
- Alternative DFS solutions can also solve this but BFS is more intuitive here.
- This approach scales well for large trees up to the problem's constraints.