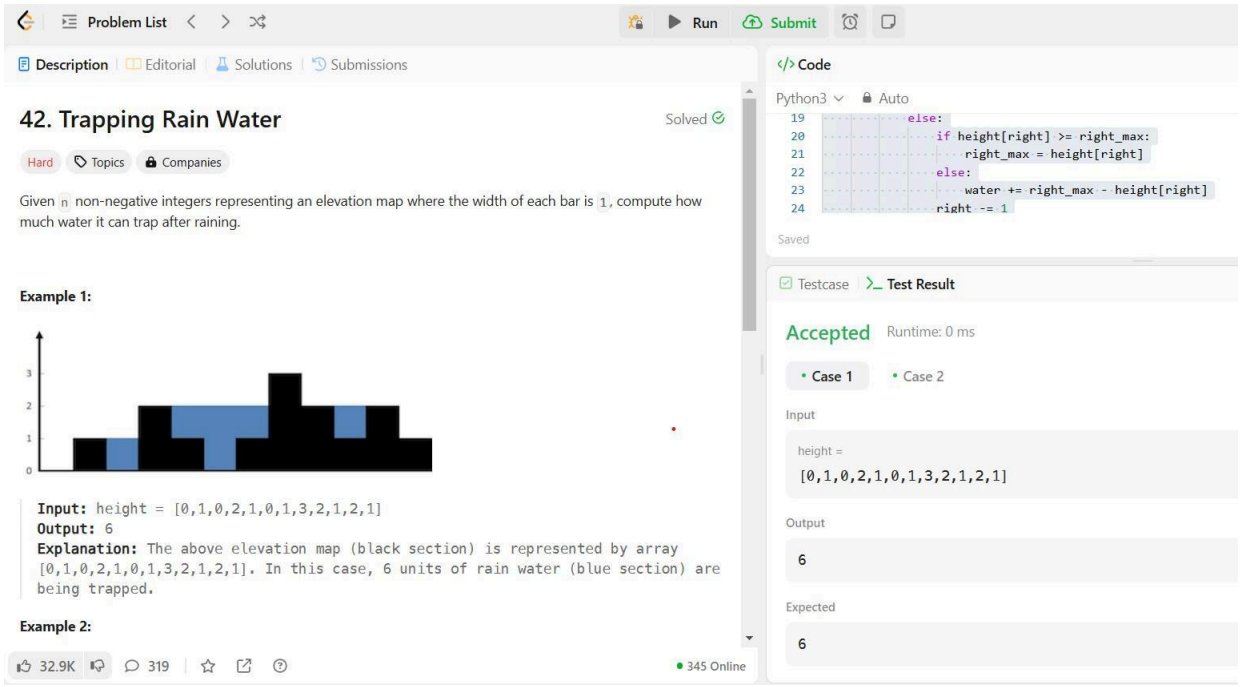
ADVANCED CODING - 2

CODE:

```python
from typing import List

class Solution:
    def trap(self, height: List[int]) -> int:
        if not height:
            return 0

        left, right = 0, len(height) - 1
        left_max, right_max = 0, 0
        water = 0

        while left < right:
            if height[left] < height[right]:
                if height[left] >= left_max:
                    left_max = height[left]
                else:
                    water += left_max - height[left]
                left += 1
            else:
                if height[right] >= right_max:
                    right_max = height[right]
                else:
                    water += right_max - height[right]
                right -= 1

        return water
```

OUTPUT:



CODE:

```
from typing import Optional

# Definition for a binary tree node.
class TreeNode:
    def _init_(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def flatten(self, root: Optional[TreeNode]) -> None:
        """
        Do not return anything, modify root in-place instead.
        """
        if not root:
            return
```

```
        # Helper function to recursively flatten the tree
        def flatten_tree(node):
            if not node:
                return None

            # Flatten the left and right subtrees
            left_tail = flatten_tree(node.left)
            right_tail = flatten_tree(node.right)

            # If there is a left subtree, attach it to the right of the
current node
            if node.left:
                if left_tail:
                    left_tail.right = node.right # Connect the tail of the
left subtree to the start of the right subtree
                node.right = node.left # Move the left subtree to the right
                node.left = None # Set the left child to None
                                                # Return the tail of the flattened
                                                tree
                                                return right_tail if right_tail else
node                                            left_tail if left_tail else


        flatten_tree(root)
```

OUTPUT:

Description | Accepted ✕ | Editorial | Solutions | Submissions

114. Flatten Binary Tree to Linked List

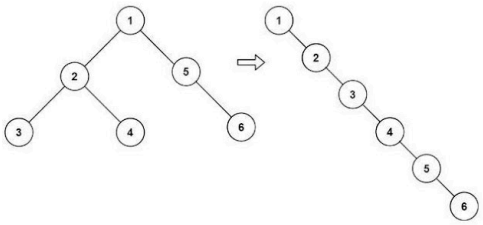
Solved

Medium | Topics | Companies | Hint

Given the `root` of a binary tree, flatten the tree into a "linked list":

- The "linked list" should use the same `TreeNode` class where the `right` child pointer points to the next node in the list and the `left` child pointer is always `null`.
- The "linked list" should be in the same order as a **pre-order traversal** of the binary tree.

Example 1:



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

12.6K | 86 | 146 Online

Code

Python3 | Auto

```python
from typing import Optional

# Definition for a binary tree node.
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
```

Saved

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

root =
[1,2,5,3,4,null,6]

Output

[1,null,2,null,3,null,4,null,5,null,6]

Expected

[1,null,2,null,3,null,4,null,5,null,6]