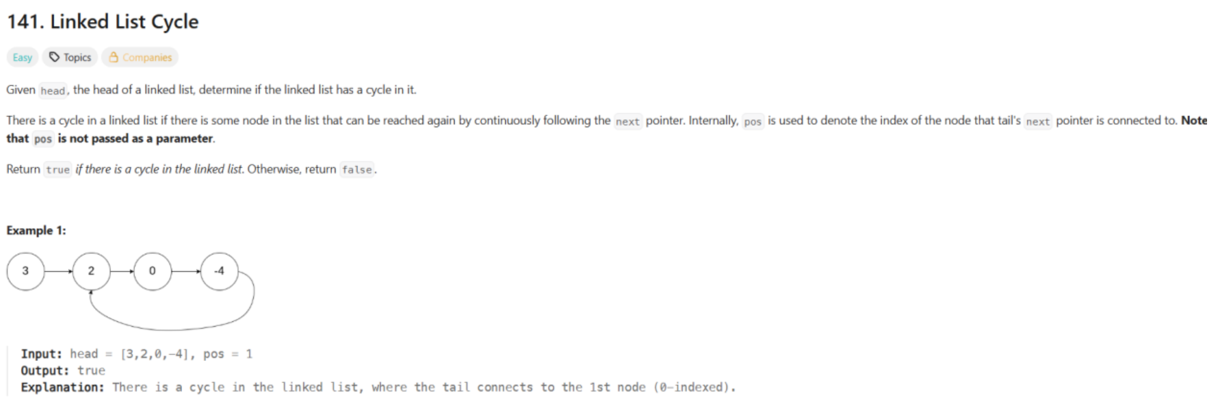
LeetCode Questions:-

Leetcode Question 01:-


141. Linked List Cycle

Easy Topics Companies

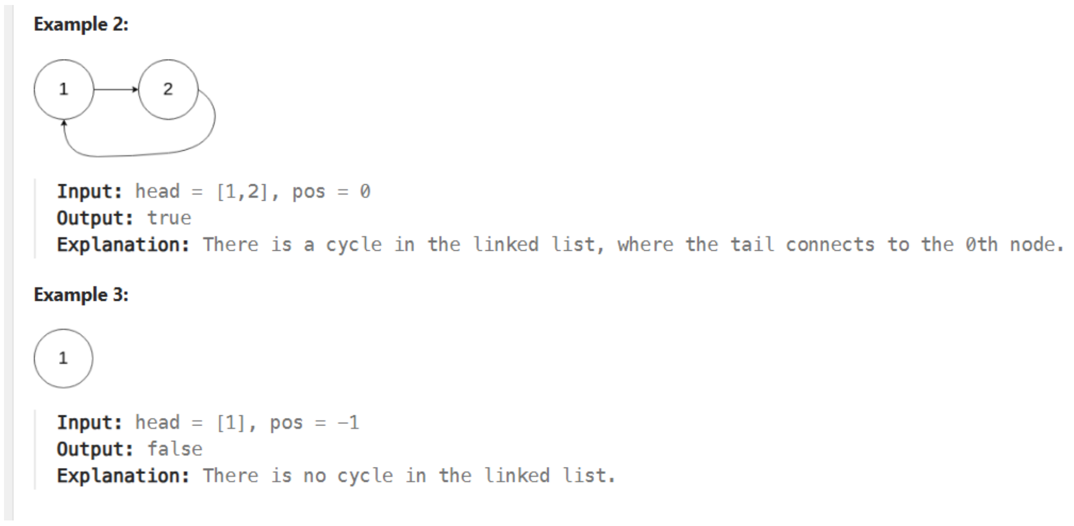
Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter.**

Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

Example 1:

```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).
```


Example 2:

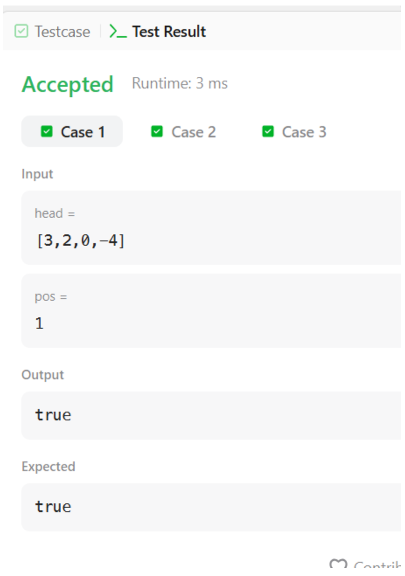
```
Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.
```

Example 3:

```
Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.
```

```
/**
 * Definition for singly-linked list.
 */
struct ListNode {
   int val;
   struct ListNode *next;
};
bool hasCycle(struct ListNode *head) {
   struct ListNode *slow = head;
   struct ListNode *fast = head;
   while (fast != NULL && fast->next != NULL) {
       slow = slow->next;         // move one step
       fast = fast->next->next;    // move two steps
```

```
        if (slow == fast)
            return true;          // cycle found
    }
    return false;                 // no cycle
}
```



Testcase | Test Result

Accepted Runtime: 3 ms

Case 1 Case 2 Case 3

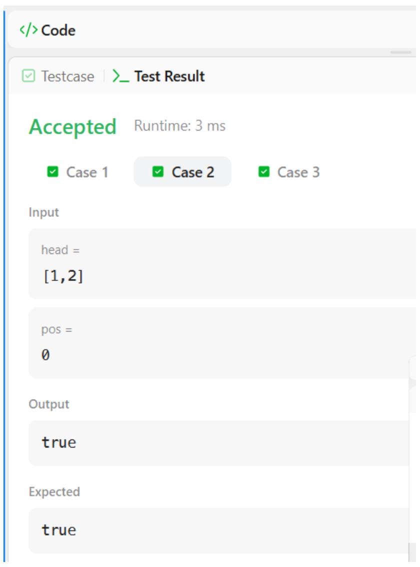
Input

head =
```
[3,2,0,-4]
```

pos =
```
1
```

Output
```
true
```

Expected
```
true
```



Code

Testcase | Test Result

Accepted Runtime: 3 ms

Case 1 Case 2 Case 3

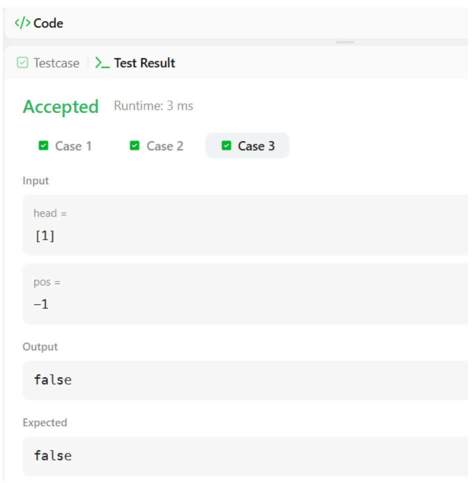
Input

head =
```
[1,2]
```

pos =
```
0
```

Output
```
true
```

Expected
```
true
```



Code

Testcase | Test Result

Accepted Runtime: 3 ms

Case 1 Case 2 Case 3

Input

head =
```
[1]
```

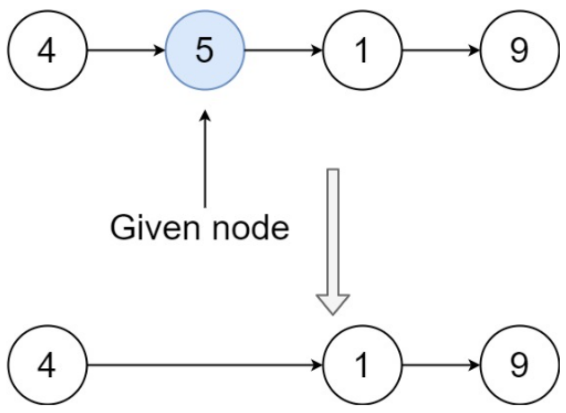
pos =
```
-1
```

Output
```
false
```

Expected
```
false
```

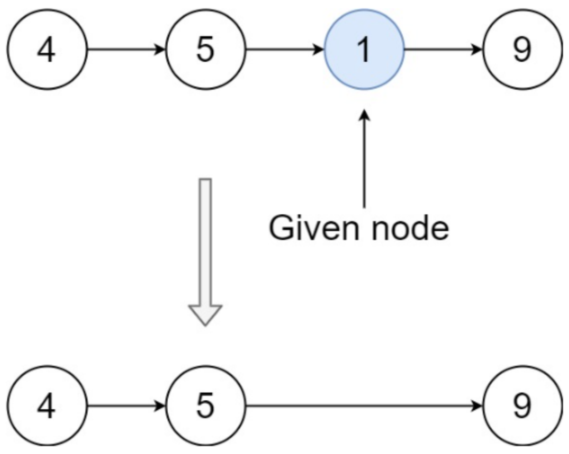
LeetCode 2:-

Delete node from Linked List



Input: head = [4,5,1,9], node = 5
Output: [4,1,9]
Explanation: You are given the second node with value 5, the linked list should become 4 -> 1 -> 9 after calling your function.

Example 2:

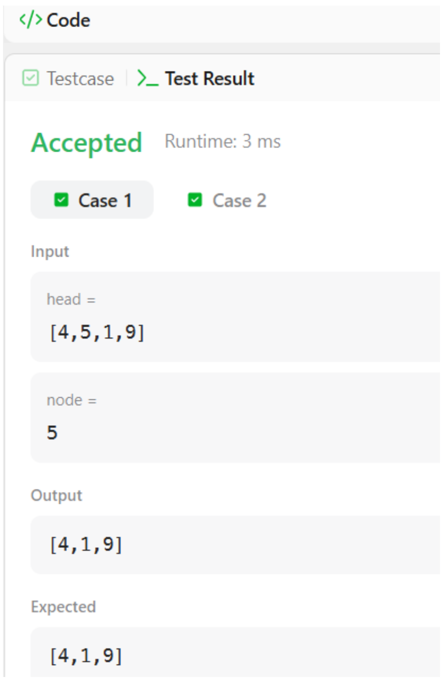


Input: head = [4,5,1,9], node = 1
Output: [4,5,9]
Explanation: You are given the third node with value 1, the linked list should become 4 -> 5 -> 9 after calling your function.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
void deleteNode(struct ListNode* node) {
   if (node == NULL || node->next == NULL)
```

```
        return;

    struct ListNode *temp = node->next;
    node->val = temp->val;        // copy next node value
    node->next = temp->next;      // unlink next node
    free(temp);                   // free memory
}
```



Code

Testcase | Test Result

Accepted Runtime: 3 ms

Case 1 Case 2

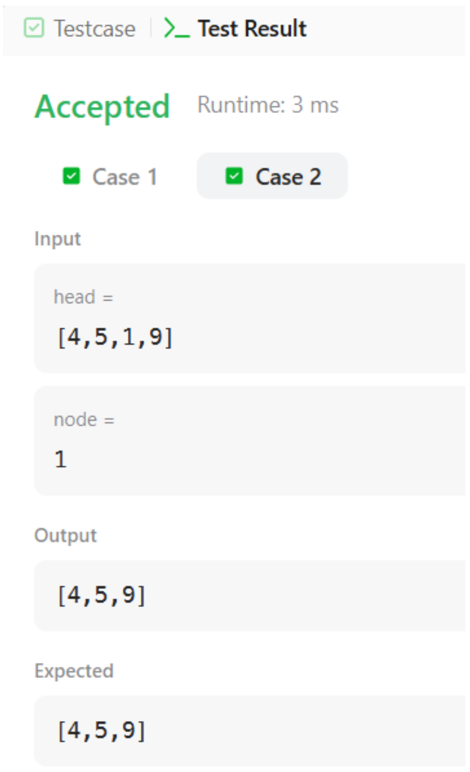
Input

head =
```
[4,5,1,9]
```

node =
```
5
```

Output
```
[4,1,9]
```

Expected
```
[4,1,9]
```



Testcase | Test Result

Accepted Runtime: 3 ms

Case 1 Case 2

Input

head =
```
[4,5,1,9]
```

node =
```
1
```

Output
```
[4,5,9]
```

Expected
```
[4,5,9]
```

LeetCode Question 3:-

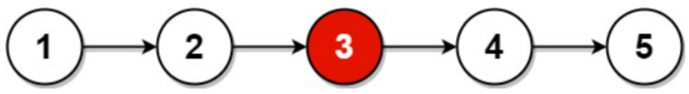
876. Middle of the Linked List

Easy Topics Companies

Given the `head` of a singly linked list, return *the middle node of the linked list*.

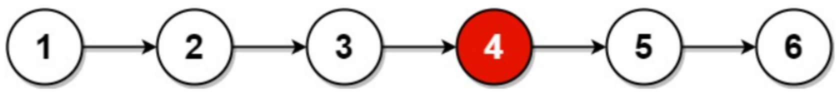
If there are two middle nodes, return **the second middle** node.

Example 1:



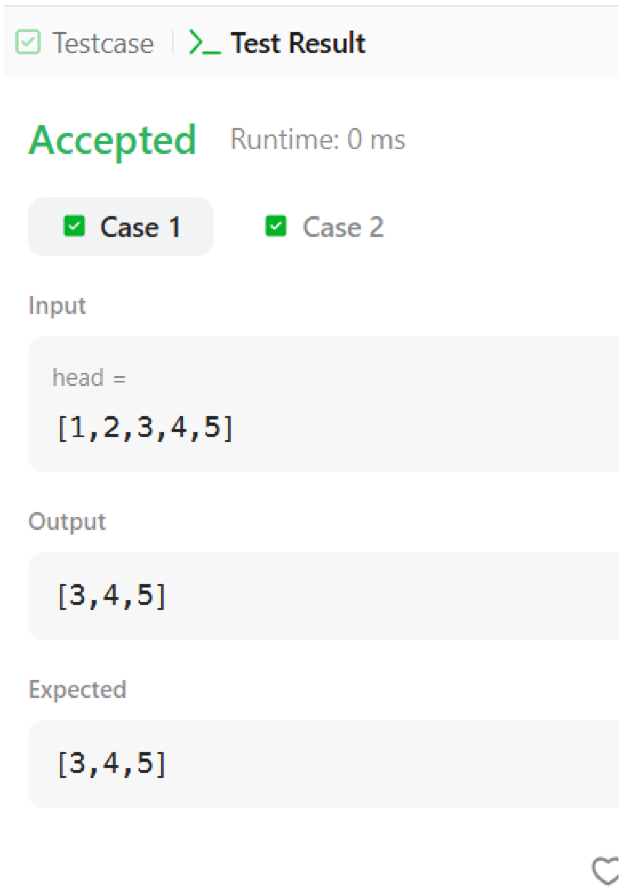
```
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.
```

Example 2:



```
Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.
```

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* middleNode(struct ListNode* head) {
    struct ListNode *slow = head;
    struct ListNode *fast = head;
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;        // move 1 step
        fast = fast->next->next; // move 2 steps
    }
    return slow;              // middle node
}
```



Accepted Runtime: 0 ms

Case 1 Case 2

Input

head =

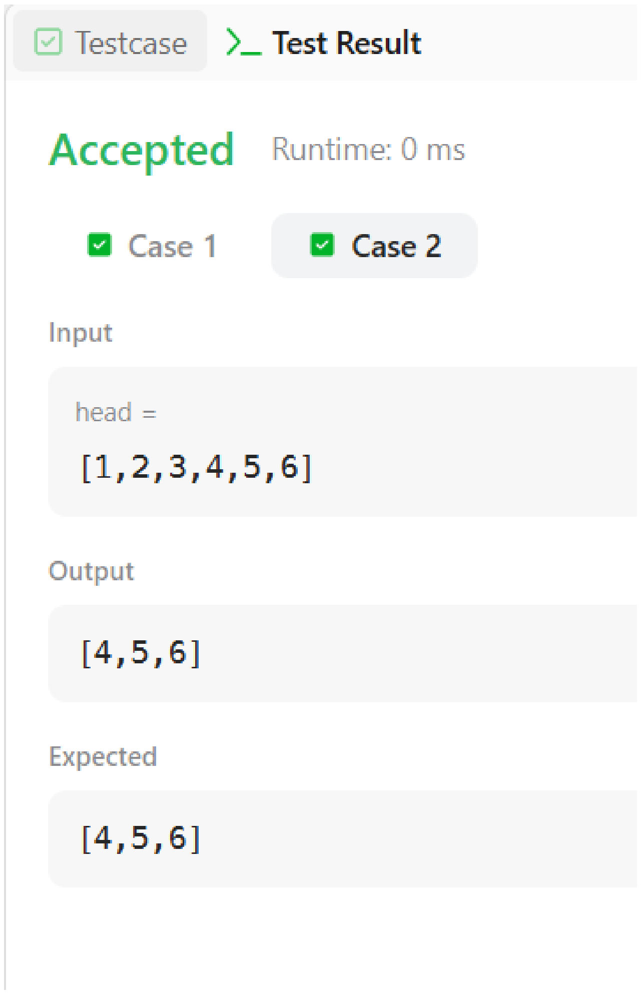
```
[1,2,3,4,5]
```

Output

```
[3,4,5]
```

Expected

```
[3,4,5]
```



Accepted Runtime: 0 ms

Case 1 Case 2

Input

head =

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

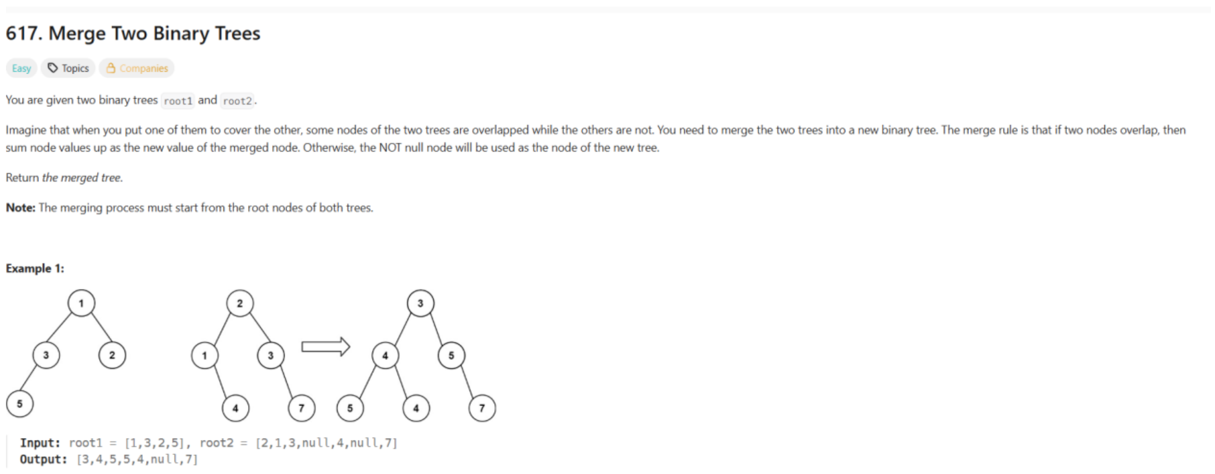
Output

```
[4,5,6]
```

Expected

```
[4,5,6]
```

LeetCode Question 4:-



617. Merge Two Binary Trees

Easy Topics Companies

You are given two binary trees `root1` and `root2`.

Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of the new tree.

Return *the merged tree*.

Note: The merging process must start from the root nodes of both trees.

Example 1:

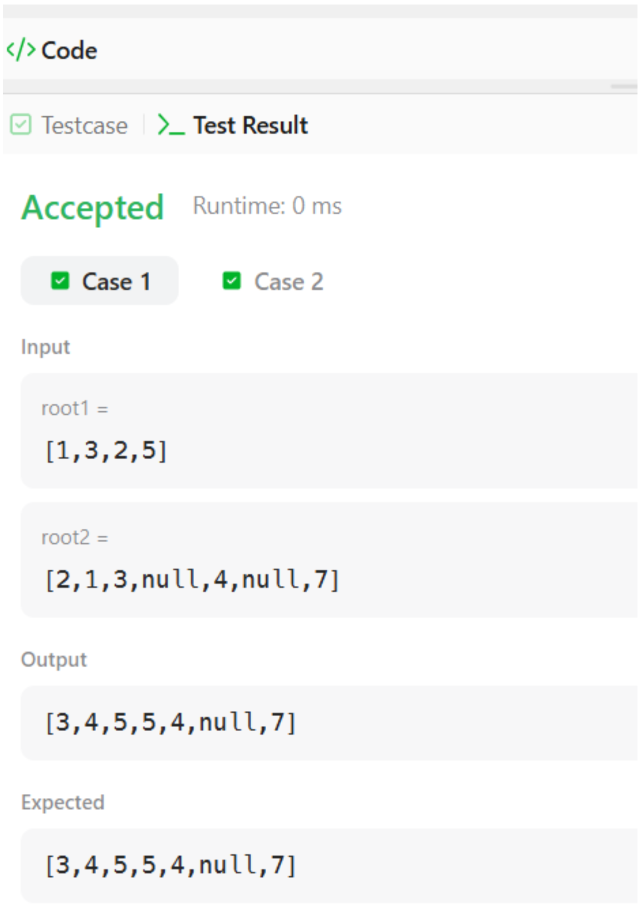
```
Input: root1 = [1,3,2,5], root2 = [2,1,3,null,4,null,7]
Output: [3,4,5,5,4,null,7]
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2) {
    if (root1 == NULL)
        return root2;
    if (root2 == NULL)
        return root1;

    root1->val += root2->val;

    root1->left = mergeTrees(root1->left, root2->left);
    root1->right = mergeTrees(root1->right, root2->right);

    return root1;
}
```



Accepted Runtime: 0 ms

Input

root1 =

```
[1,3,2,5]
```

root2 =

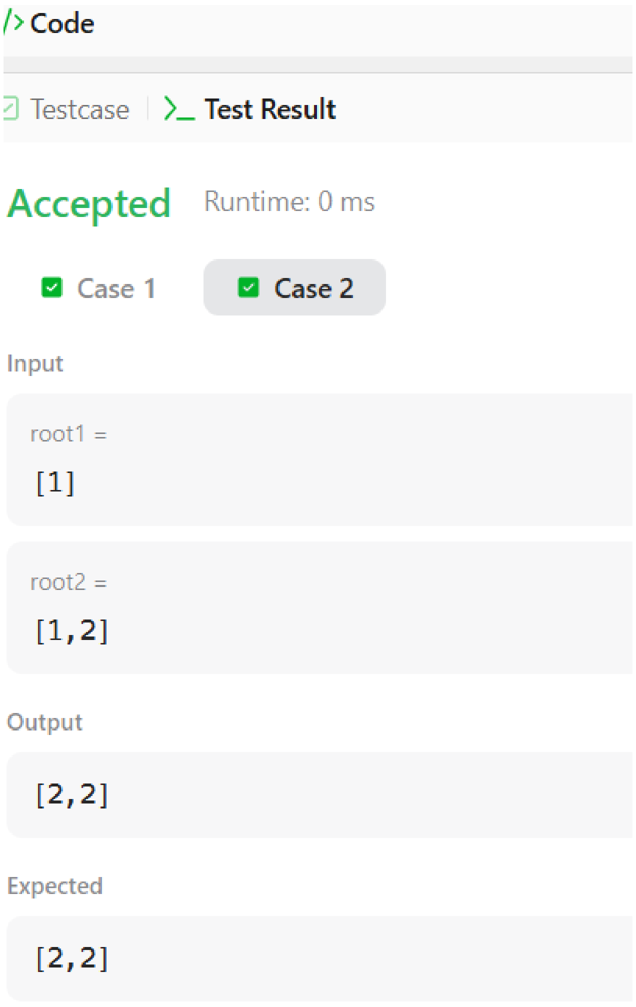
```
[2,1,3,null,4,null,7]
```

Output

```
[3,4,5,5,4,null,7]
```

Expected

```
[3,4,5,5,4,null,7]
```



Accepted Runtime: 0 ms

Input

root1 =

```
[1]
```

root2 =

```
[1,2]
```

Output

```
[2,2]
```

Expected

```
[2,2]
```