LEETCODE REPORT

QUESTION 1:

Array | Run | Submit | Premium

Description | Editorial | Solutions | Submissions

26. Remove Duplicates from Sorted Array

Easy | Topics | Companies | Hint

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates in-place such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**. Then return *the number of unique elements in* `nums`.

Consider the number of unique elements of `nums` to be `k`, to get accepted, you need to do the following things:

- Change the array `nums` such that the first `k` elements of `nums` contain the unique elements in the order they were present in `nums` initially. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) {
    assert nums[i] == expectedNums[i];
}
```

Code

Testcase | Test Result

Accepted Runtime: 3 ms

Case 1 | Case 2

Input

nums =
[1,1,2]

Output

[1,2]

Expected

[1,2]

Contribute a testcase

CODE:

```
int removeDuplicates(int* nums, int numsSize) {
    if (numsSize == 0) return 0;

    int uniqueIndex = 0;

    for (int i = 1; i < numsSize; i++) {
        if (nums[i] != nums[uniqueIndex]) {
            uniqueIndex++;
            nums[uniqueIndex] = nums[i];
        }
    }

    return uniqueIndex + 1;
}
```

QUESTION 2:

Recursion | Run | Submit | Premium

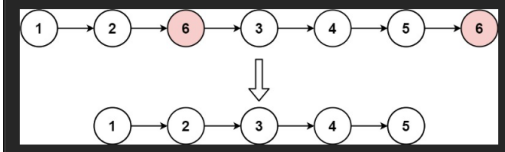
Description | Editorial | Solutions | Submissions

203. Remove Linked List Elements

Easy | Topics | Companies

Given the `head` of a linked list and an integer `val`, remove all the nodes of the linked list that has `Node.val == val`, and return *the new head.*

Example 1:




```
Input: head = [1,2,6,3,4,5,6], val = 6
Output: [1,2,3,4,5]
```

Example 2:

```
Input: head = [], val = 1
Output: []
```

Example 3:

```
Input: head = [7,7,7,7], val = 7
Output: []
```

Code

Testcase | Test Result

Case 1 | Case 2 | Case 3

head =

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

val =

6

CODE:

```c
/**
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* removeElements(struct ListNode* head, int val) {
    // Initialize a dummy node to handle cases where the head needs to be removed
    struct ListNode* dummy = (struct ListNode*)malloc(sizeof(struct ListNode));
    dummy->next = head;

    struct ListNode* prev = dummy;
    struct ListNode* current = head;

    while (current != NULL) {
        if (current->val == val) {
            // Remove the current node
            prev->next = current->next;
            free(current);
            current = prev->next;
        } else {
            // Move to the next node
            prev = current;
            current = current->next;
        }
    }
    head = dummy->next;
    free(dummy); // Free the dummy node

    return head;
}
```

Question 3:

Stack | Run | Submit | 0 | Premium

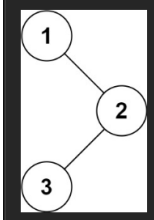
Description | Editorial | Solutions | Submissions

94. Binary Tree Inorder Traversal

Easy | Topics | Companies

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]
```

Code

Testcase | Test Result

Case 1 | Case 2 | Case 3

root =

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

CODE:

```c
void inorderTraversalHelper(struct TreeNode* root, int* result, int* index) {
    if (root == NULL)
        return;

    inorderTraversalHelper(root->left, result, index);
    result[(*index)++] = root->val;
    inorderTraversalHelper(root->right, result, index);
}

int* inorderTraversal(struct TreeNode* root, int* returnSize) {
    // Count the number of nodes
    int count = 0;
    void countNodes(struct TreeNode* node) {
        if (node == NULL)
            return;
        countNodes(node->left);
        count++;
        countNodes(node->right);
    }
    countNodes(root);

    // Allocate memory for result array
    int* result = (int*)malloc(count * sizeof(int));
    *returnSize = count;

    // Perform inorder traversal
    int index = 0;
    inorderTraversalHelper(root, result, &index);

    return result;
}
```

Question 4:

Queue | Run | Submit | Premium

Description | Editorial | Solutions | Submissions

387. First Unique Character in a String

Easy | Topics | Companies

Given a string `s`, *find the first non-repeating character in it and return its index.* If it does not exist, return `-1`.

Example 1:

```
Input: s = "leetcode"
Output: 0
```

Example 2:

```
Input: s = "loveleetcode"
Output: 2
```

Example 3:

```
Input: s = "aabb"
Output: -1
```

Code

Testcase | Test Result

Case 1 | Case 2 | Case 3

s =

```
"leetcode"
```

CODE:

```c
int firstUniqChar(char* s) {
    int count[26] = {0}; // Assuming input string contains only lowercase English letters

    // Count occurrences of each character
    for (int i = 0; s[i] != '\0'; i++) {
        count[s[i] - 'a']++;
    }

    // Find the index of the first unique character
    for (int i = 0; s[i] != '\0'; i++) {
        if (count[s[i] - 'a'] == 1) {
            return i;
        }
    }

    return -1; // If no unique character found
}
```

Question 5:

Linked List | Run | Submit | Premium

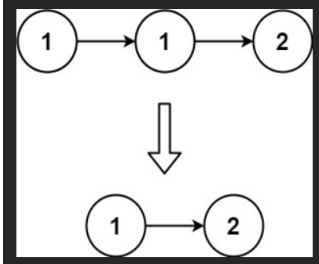
Description | Editorial | Solutions | Submissions

83. Remove Duplicates from Sorted List

Easy | Topics | Companies

Given the `head` of a sorted linked list, *delete all duplicates such that each element appears only once.* Return *the linked list* ***sorted*** *as well.*

Example 1:




```
Input: head = [1,1,2]
Output: [1,2]
```

Code

Testcase | Test Result

Case 1 | Case 2

head =

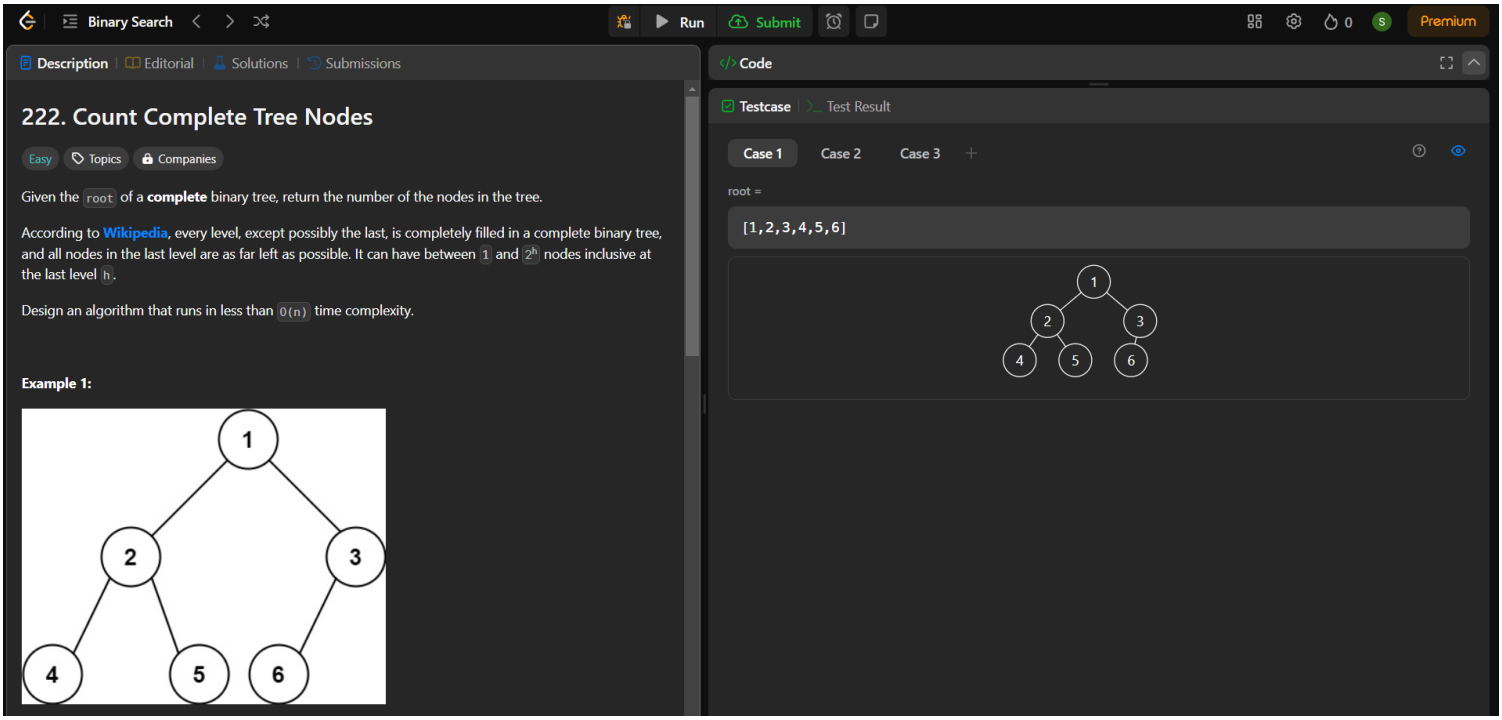
```
[1,1,2]
```

CODE:

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* deleteDuplicates(struct ListNode* head) {
    if (head == NULL || head->next == NULL)
        return head;

    struct ListNode* current = head;
    while (current->next != NULL) {
        if (current->val == current->next->val) {
            struct ListNode* nextNode = current->next;
            current->next = nextNode->next;
            free(nextNode);
        } else {
            current = current->next;
        }
    }
    return head;
}
```

Question 6:



CODE:

```c
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
int countNodes(struct TreeNode* root) {
    if (root == NULL)
        return 0;
    int leftHeight = 0, rightHeight = 0;
    struct TreeNode *leftNode = root, *rightNode = root;

    while (leftNode != NULL) {
        leftHeight++;
        leftNode = leftNode->left;
    }

    while (rightNode != NULL) {
        rightHeight++;
        rightNode = rightNode->right;
    }

    if (leftHeight == rightHeight)
        return (1 << leftHeight) - 1;
    else
        return 1 + countNodes(root->left) + countNodes(root->right);
}
```

Question 7:

Linked List | Run | Submit | 0 | Premium

Description | Editorial | Solutions | Submissions

237. Delete Node in a Linked List

Medium | Topics | Companies

There is a singly-linked list `head` and we want to delete a node `node` in it.

You are given the node to be deleted `node`. You will **not be given access** to the first node of `head`.

All the values of the linked list are **unique**, and it is guaranteed that the given node `node` is not the last node in the linked list.

Delete the given node. Note that by deleting the node, we do not mean removing it from memory. We mean:

- The value of the given node should not exist in the linked list.
- The number of nodes in the linked list should decrease by one.
- All the values before `node` should be in the same order.
- All the values after `node` should be in the same order.

Code

Testcase | Test Result

Case 1 | Case 2

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

```
5
```

CODE:

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
void deleteNode(struct ListNode* node) {

    node->val = node->next->val;

    struct ListNode* temp = node->next;
    node->next = node->next->next;

    free(temp);
}
```

Question 8:

Graph | Run | Submit | 0 | Premium

Description | Editorial | Solutions | Submissions

684. Redundant Connection

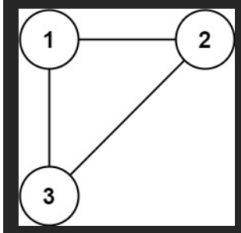
Medium | Topics | Companies

In this problem, a tree is an **undirected graph** that is connected and has no cycles.

You are given a graph that started as a tree with `n` nodes labeled from `1` to `n`, with one additional edge added. The added edge has two **different** vertices chosen from `1` to `n`, and was not an edge that already existed. The graph is represented as an array `edges` of length `n` where `edges[i] = [$a_i$, $b_i$]` indicates that there is an edge between nodes `$a_i$` and `$b_i$` in the graph.

Return *an edge that can be removed so that the resulting graph is a tree of* `n` *nodes*. If there are multiple answers, return the answer that occurs last in the input.

Example 1:




```
Input: edges = [[1,2],[1,3],[2,3]]
Output: [2,3]
```

Code

Testcase | Test Result

Case 1 | Case 2

edges =

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

CODE:

```c
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int parent[1001];

int find(int x) {
    if (parent[x] == -1) return x;
    return parent[x] = find(parent[x]);
}

int* findRedundantConnection(int** edges, int edgesSize, int* edgesColSize, int* returnSize) {
    *returnSize = 2;
    for (int i = 0; i <= 1000; i++) {
        parent[i] = -1;
    }

    for (int i = 0; i < edgesSize; i++) {
        int u = find(edges[i][0]);
        int v = find(edges[i][1]);
        if (u == v) {
            int* result = (int*)malloc(2 * sizeof(int));
            result[0] = edges[i][0];
            result[1] = edges[i][1];
            return result;
        }
        parent[u] = v;
    }

    return NULL;
}
```

Question 9:

Hash Function | Run | Submit | Premium

Description | Editorial | Solutions | Submissions

718. Maximum Length of Repeated Subarray

Medium | Topics | Companies | Hint

Given two integer arrays `nums1` and `nums2`, return *the maximum length of a subarray that appears in* ***both*** *arrays.*

Example 1:

```
Input: nums1 = [1,2,3,2,1], nums2 = [3,2,1,4,7]
Output: 3
Explanation: The repeated subarray with maximum length is [3,2,1].
```

Example 2:

```
Input: nums1 = [0,0,0,0,0], nums2 = [0,0,0,0,0]
Output: 5
Explanation: The repeated subarray with maximum length is [0,0,0,0,0].
```

Code

Testcase | Test Result

Case 1 | Case 2

nums1 =

[1,2,3,2,1]

nums2 =

[3,2,1,4,7]

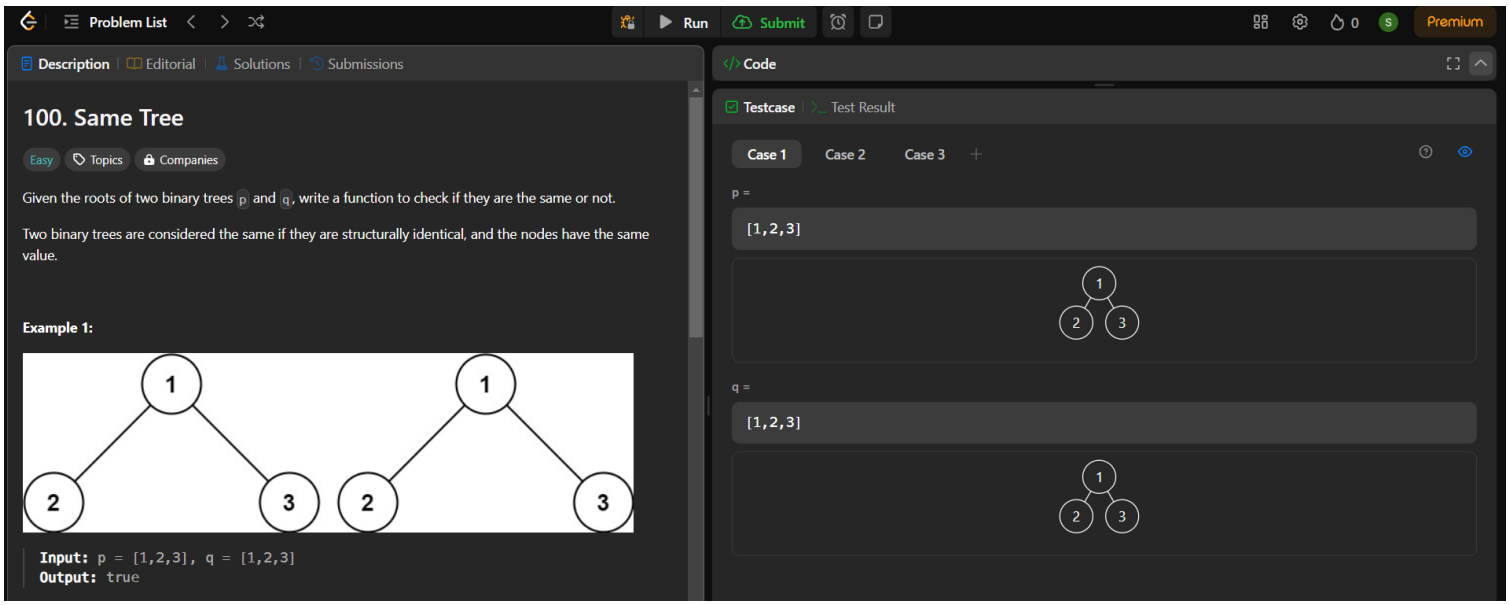
CODE:

```c
int max(int a, int b) {
    return a > b ? a : b;
}

int findLength(int* nums1, int nums1Size, int* nums2, int nums2Size) {
    int maxLen = 0;
    int dp[nums1Size + 1][nums2Size + 1];

    for (int i = 0; i <= nums1Size; i++) {
        for (int j = 0; j <= nums2Size; j++) {
            if (i == 0 || j == 0) {
                dp[i][j] = 0;
            } else if (nums1[i - 1] == nums2[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
                maxLen = max(maxLen, dp[i][j]);
            } else {
                dp[i][j] = 0;
            }
        }
    }
    return maxLen;
}
```

Question 10:



CODE:

```c
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
bool isSameTree(struct TreeNode* p, struct TreeNode* q) {
    if (p == NULL && q == NULL)
        return true;
    if (p == NULL || q == NULL || p->val != q->val)
        return false;
    return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
}
```