

## LEETCODES

**4b. 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.**

</> Code

C ▾ 🔒 Auto

```
1
2 struct ListNode* middleNode(struct ListNode* head) {
3     struct ListNode *slow = head;
4     struct ListNode *fast = head;
5
6     while (fast != NULL && fast->next != NULL) {
7         slow = slow->next;
8         fast = fast->next->next;
9     }
10
11     return slow;
12 }
13
```

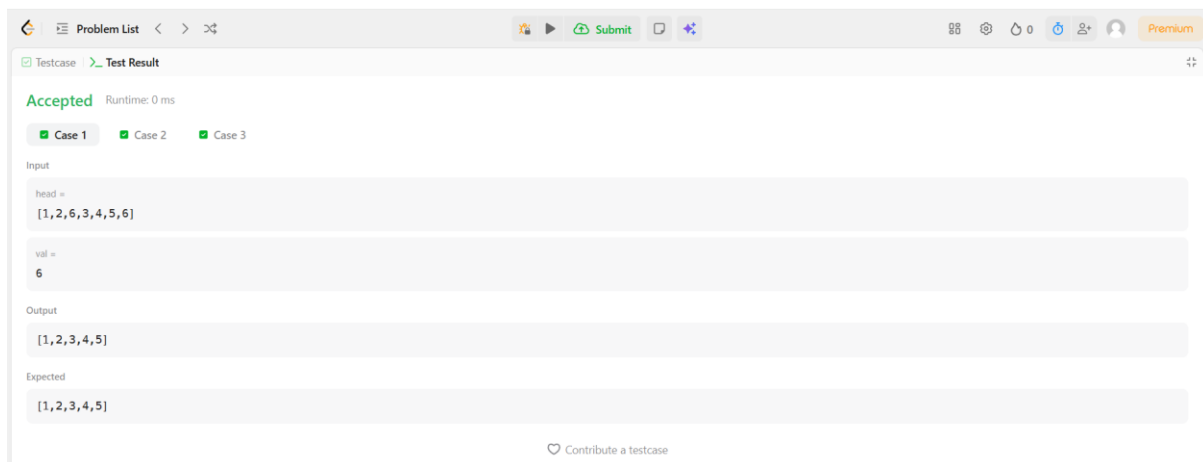
The screenshot shows the LeetCode test result interface for the 'middleNode' problem. At the top, it says 'Accepted' with a runtime of '0 ms'. Below this, there are two tabs: 'Case 1' and 'Case 2', both of which are marked as passed. The 'Input' section shows 'head = [1,2,3,4,5]'. The 'Output' section shows '[3,4,5]'. The 'Expected' section also shows '[3,4,5]'. At the bottom, there is a link to 'Contribute a testcase'.

**5b 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***

 Code

C   Auto

```
1 struct ListNode* removeElements(struct ListNode* head, int val) {
2     // Remove nodes from beginning if needed
3     while (head != NULL && head->val == val) {
4         struct ListNode* temp = head;
5         head = head->next;
6         free(temp);
7     }
8
9     struct ListNode* curr = head;
10
11    // Remove nodes from rest of the list
12    while (curr != NULL && curr->next != NULL) {
13        if (curr->next->val == val) {
14            struct ListNode* temp = curr->next;
15            curr->next = temp->next;
16            free(temp);
17        } else {
18            curr = curr->next;
19        }
20    }
21
22    return head;
23 }
24
```



The screenshot shows a coding platform interface. At the top, there's a 'Problem List' tab and a 'Test Result' tab. The 'Test Result' tab is active, showing 'Accepted' status with a runtime of 0 ms. Below this, there are three test cases: 'Case 1', 'Case 2', and 'Case 3', all marked as passed. The input section shows 'head = [1, 2, 6, 3, 4, 5, 6]' and 'val = 6'. The output section shows '[1, 2, 3, 4, 5]'. The expected section also shows '[1, 2, 3, 4, 5]'. At the bottom, there's a 'Contribute a testcase' link.

**7b. 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.

</>Code

C ▾ 🔒 Auto

```
1  /**
2   * Definition for singly-linked list.
3   * struct ListNode {
4   *     int val;
5   *     struct ListNode *next;
6   * };
7   */
8
9  bool hasCycle(struct ListNode *head) {
10     struct ListNode *slow = head;
11     struct ListNode *fast = head;
12
13     while (fast != NULL && fast->next != NULL) {
14         slow = slow->next;        // move 1 step
15         fast = fast->next->next;   // move 2 steps
16
17         if (slow == fast) {
18             return true;          // cycle found
19         }
20     }
21
22     return false;                // no cycle
23 }
24
```

Testcase > Test Result

Accepted Runtime: 4 ms

Case 1 Case 2 Case 3

Input

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

pos =  
1

Output

true

Expected

true

♥ Contribute a testcase

8b. 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.

The image displays two screenshots from a coding platform. The top screenshot shows a C++ code editor with the following code:

```
1 struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2) {
2     // If both nodes are NULL
3     if (root1 == NULL && root2 == NULL)
4         return NULL;
5
6     // If one of them is NULL, return the other
7     if (root1 == NULL)
8         return root2;
9
10    if (root2 == NULL)
11        return root1;
12
13    // Both nodes exist
14    root1->val = root1->val + root2->val;
15
16    // Merge left and right subtrees
17    root1->left = mergeTrees(root1->left, root2->left);
18    root1->right = mergeTrees(root1->right, root2->right);
19
20    return root1;
21 }
22
```

The bottom screenshot shows the test result interface. It indicates that the solution was "Accepted" with a runtime of 0 ms. Two test cases are listed: Case 1 and Case 2. The input for Case 1 is:

```
root1 = [1,3,2,5]
```

The input for Case 2 is:

```
root2 = [2,1,3,null,4,null,7]
```

The output for both cases is:

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

The expected output for both cases is:

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

At the bottom of the interface, there is a link to "Contribute a testcase".