

1. Print Linked List

The screenshot shows the 'Print Linked List' problem page on GeeksforGeeks. The problem has been solved successfully. The left sidebar displays the following statistics:

- Test Cases Passed: 1112 / 1112
- Attempts: Correct / Total: 1 / 2
- Accuracy: 50%
- Points Scored: 1 / 1
- Time Taken: 0.1
- Your Total Score: 3 ↑

The 'Solve Next' section includes buttons for 'Count Linked List Nodes', 'Delete Alternate Nodes', and 'Insert in Middle of Linked List'. The main code editor shows a C++ solution:

```
1 // Driver Code Ends
2
3 struct Node {
4     int data;
5     struct Node* next;
6 }
7
8 Node(int x) {
9     data = x;
10    next = nullptr;
11}
12
13 // Print elements of a linked list on console
14 // Head pointer input could be NULL as well for empty list
15
16 class Solution {
17 public:
18     // Function to display the elements of a linked list in same line
19     void printList(Node* head) {
20         Node* temp = head;
21         while(temp != NULL) {
22             cout << temp->data << " ";
23             temp = temp->next;
24         }
25     }
26 };
27 // Driver Code Ends
```

2. Remove duplicates from a sorted list

The screenshot shows the 'Remove Duplicates from Sorted List' problem page on LeetCode. The problem has been solved successfully. The left sidebar displays the following statistics:

- Accepted: 168 / 168 testcases passed
- Runtime: 0 ms | Beats 100.00%
- Memory: 16.09 MB | Beats 90.25%

The 'Testcase' section shows 'Accepted' with 'Runtime: 0 ms' and 'Case 1' selected. The main code editor shows a C++ solution:

```
1 class Solution {
2 public:
3     ListNode* deleteDuplicates(ListNode* head) {
4         ListNode* res = head;
5
6         while (head && head->next) {
7             if (head->val == head->next->val) {
8                 head->next = head->next->next;
9             } else {
10                head = head->next;
11            }
12        }
13
14        return res;
15    }
16};
```

3. Reverse a Linked List

The screenshot shows a LeetCode submission for the problem "Reverse Linked List". The submission is accepted, with 28/28 test cases passed. The runtime is 0 ms, which is 100.00% faster than other submissions. The memory usage is 13.47 MB, which is 40.49% less than other submissions. The code is written in C++ and implements a recursive solution to reverse the linked list.

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        // Initialize pointers
        ListNode* prev = nullptr; // Previous node starts as NULL
        // ... (rest of the code)
    }
};
```

The code on the right shows the implementation of the reverseList function. It uses a while loop to traverse the list, saving the next node, reversing the link, and moving pointers forward until the list is reversed.

4. Delete middle node of a list

The screenshot shows the LeetCode problem page for "2095. Delete the Middle Node of a Linked List". The problem is of medium difficulty. The description states: "You are given the head of a linked list. Delete the middle node, and return the head of the modified linked list. The middle node of a linked list of size n is the $\lfloor n / 2 \rfloor$ th node from the start using 0-based indexing, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x. For n = 1, 2, 3, 4, and 5, the middle nodes are 0, 1, 1, 2, and 2, respectively."

Example 1: A linked list with nodes [1, 3, 4, 7, 1, 2, 6]. The middle node is 7 (index 3). The output is [1, 3, 4, 1, 2, 6].

Input: head = [1,3,4,7,1,2,6]
Output: [1,3,4,1,2,6]
Explanation: The above figure represents the given linked list. The indices of the nodes are written below. Since n = 7, node 3 with value 7 is the middle node, which is marked in red. We return the new list after removing this node.

The code on the right shows the implementation of the deleteMiddle function. It uses a slow and fast pointer technique to find the middle node and then removes it by updating the next pointer of the previous node.

5. Merge two sorted linked lists

21. Merge Two Sorted Lists

Easy Topics Companies

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list.

Example 1:

```
graph LR
    subgraph list1
        n1_1((1)) --> n1_2((2))
        n1_2 --> n1_4((4))
    end
    subgraph list2
        n2_1((1)) --> n2_3((3))
        n2_3 --> n2_4((4))
    end
    subgraph merged
        m1((1)) --> m2((1))
        m2 --> m3((2))
        m3 --> m4((3))
        m4 --> m5((4))
        m5 --> m6((4))
    end
```

```
C++
ListNode* cur = dummy;
while (list1 && list2) {
    if (list1->val > list2->val) {
        cur->next = list2;
        list2 = list2->next;
    } else {
        cur->next = list1;
        list1 = list1->next;
    }
    cur = cur->next;
}
cur->next = list1 ? list1 : list2;
ListNode* head = dummy->next;
delete dummy;
return head;
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

list1 = [1, 2, 4]

6. Remove duplicate from sorted list 2

141. Linked List Cycle

Easy Topics Companies

Solved

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:

```
graph LR
    n3((3)) --> n2((2))
    n2 --> n0((0))
    n0 --> n4((-4))
    n4 --> n3
```

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:

```
graph LR
    n1((1)) --> n2((2))
    n2 --> n3((3))
    n3 --> n4((4))
    n4 --> null(( ))
```

```
C++
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode* fast = head;
        ListNode* slow = head;
        while (fast != nullptr && fast->next != nullptr) {
            fast = fast->next->next;
            slow = slow->next;
            if (fast == slow) {
                return true;
            }
        }
        return false;
    }
};
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

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

7. Delete a cycle in a linked list

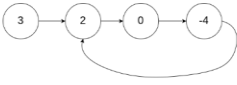
141. Linked List Cycle

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:



```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode* fast = head;
        ListNode* slow = head;

        while (fast != nullptr && fast->next != nullptr) {
            fast = fast->next->next;
            slow = slow->next;
        }

        if (fast == slow) {
            return true;
        }

        return false;
    }
};
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

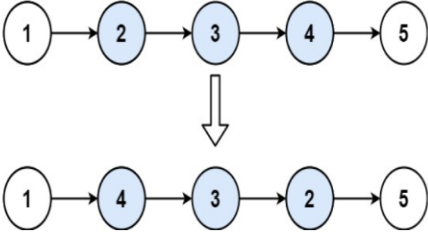
head = [3,2,0,-4]

8. Reverse linked list 2

92. Reverse Linked List II

Given the `head` of a singly linked list and two integers `left` and `right` where `left <= right`, reverse the nodes of the list from position `left` to position `right`, and return the reversed list.

Example 1:



Input: `head = [1,2,3,4,5]`, `left = 2`, `right = 4`
Output: `[1,4,3,2,5]`

Example 2:

Input: `head = [5]`, `left = 1`, `right = 1`
Output: `[5]`

```
ListNode* prev = dummy;

for (int i = 0; i < left - 1; i++) {
    prev = prev->next;
}

ListNode* cur = prev->next;

for (int i = 0; i < right - left; i++) {
    ListNode* temp = cur->next;
    cur->next = prev->next;
    prev->next = cur;
    cur = temp;
}

return dummy->next;
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

head = [1,2,3,4,5]

9. Rotate a list

Rotate List - LeetCode

leetcode.com/problems/rotate-list/description/

Problem List < > Run Submit

Description Editorial Solutions Submissions

61. Rotate List

Medium Topics Companies

Given the `head` of a linked list, rotate the list to the right by `k` places.

Example 1:

1 → 2 → 3 → 4 → 5

rotate 1 5 → 1 → 2 → 3 → 4

rotate 2 4 → 5 → 1 → 2 → 3

Input: head = [1,2,3,4,5], k = 2
Output: [4,5,1,2,3]

Example 2:

0 → 1 → 2

```
C++  
class Solution {  
public:  
    ListNode* rotateRight(ListNode* head, int k) {  
        if (!head) return nullptr;  
        int n = 1;  
        ListNode* temp = head;  
        while (temp->next) {  
            n++;  
            temp = temp->next;  
        }  
        k %= n;  
        temp->next = head;  
        for (int i = 0; i < n - k; i++) temp = temp->next;  
        ListNode* new_head = temp->next;  
        temp->next = nullptr;  
        return new_head;  
    }  
};
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input: head = [1,2,3,4,5]

10. Sort List

Sort List - LeetCode

leetcode.com/problems/sort-list/description/

Problem List < > Run Submit

Description Editorial Solutions Submissions

148. Sort List

Medium Topics Companies

Given the `head` of a linked list, return the list after sorting it in **ascending order**.

Example 1:

4 → 2 → 1 → 3

1 → 2 → 3 → 4

Input: head = [4,2,1,3]
Output: [1,2,3,4]

Example 2:

-1 → 5 → 3 → 4 → 0

0 → 1 → 3 → 4 → 5

```
C++  
ListNode dummy(0);  
ListNode* tail = &dummy;  
  
while (l1 && l2) {  
    if (l1->val < l2->val) {  
        tail->next = l1;  
        l1 = l1->next;  
    } else {  
        tail->next = l2;  
        l2 = l2->next;  
    }  
    tail = tail->next;  
}  
  
tail->next = l1 ? l1 : l2;  
return dummy.next;
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input: head = [4,2,1,3]

11. Detect a cycle in a linked list 2

Linked List Cycle II - LeetCode

leetcode.com/problems/linked-list-cycle-ii/description/

Problem List < > Run Submit

Description

Editorial

Solutions

Submissions

142. Linked List Cycle II

Medium Topics Companies

Given the `head` of a linked list, return the node where the cycle begins. If there is no cycle, return `null`.

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 (0-indexed). It is `-1` if there is no cycle. **Note that `pos` is not passed as a parameter.**

Do not modify the linked list.

Example 1:

```
graph LR; 3((3)) --> 2((2)); 2 --> 0((0)); 0 --> -4((-4)); -4 --> 2;
```

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

14.1K 179 116 Online

Code

C++ Auto

```
7 while (fast && fast->next) {
8     slow = slow->next;
9     fast = fast->next->next;
10
11     if (slow == fast) break;
12 }
13
14 if (!fast || !fast->next) return nullptr;
15
16 fast = head;
17 while (fast != slow) {
18     fast = fast->next;
19     slow = slow->next;
20 }
21
22 return slow;
23
24 };
```

Saved Ln 24, Col 3

Testcase Test Result

Accepted Runtime: 3 ms

Case 1 Case 2 Case 3

Input

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