**AP ASSIGNMENT**

**Name: Vipul Vidya | UID: 22bcs16942 | Section: 612-“B”**

**Print linked list – GFG**

class Solution {

public:

// Function to display the elements of a linked list in the same line without trailing space

void printList(Node \*head) {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data;

if (temp->next != nullptr) {

cout << " ";

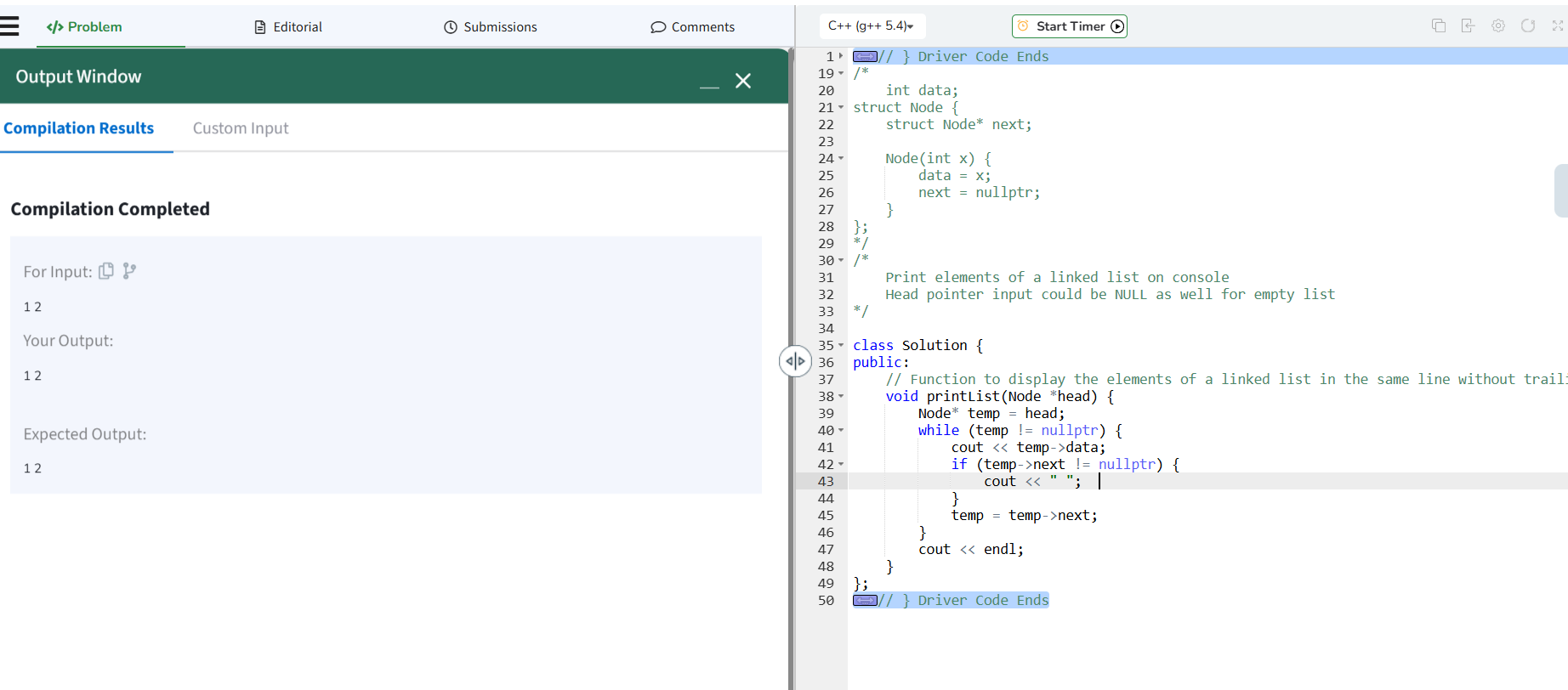
}

temp = temp->next;

}

cout << endl; }

};



**Remove duplicates from a linkedlist**

class Solution {

public:

    ListNode\* deleteDuplicates(ListNode\* head) {

         ListNode\* current = head;

        while (current && current->next) {

            if (current->val == current->next->val) {

                current->next = current->next->next;

            } else {

                current = current->next;

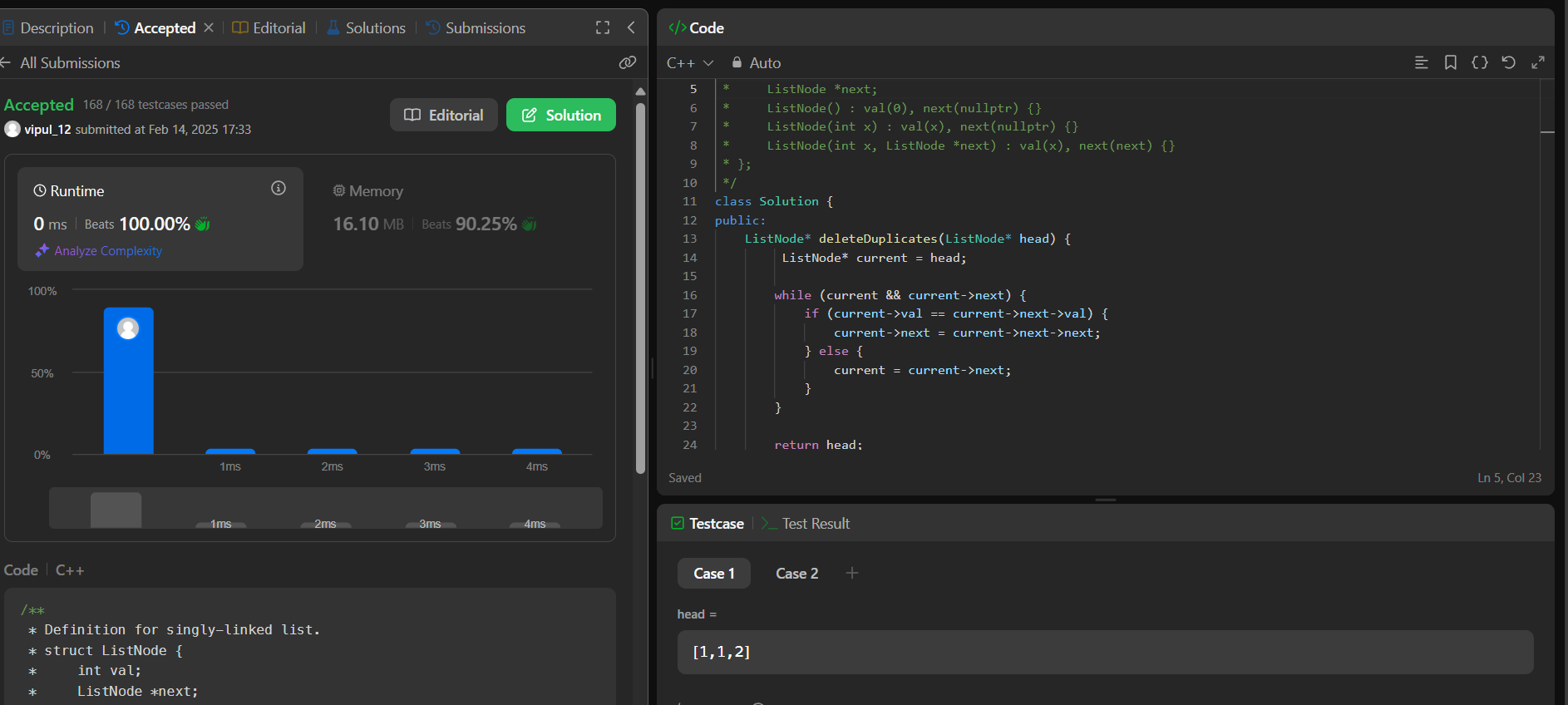
            }

        }

        return head;

    }

};



**Reverse a linked list**

class Solution {

public:

    ListNode\* reverseList(ListNode\* head) {

        ListNode\* prev = NULL;

    ListNode\* current = head;

    while (current != NULL) {

        ListNode\* nextNode = current->next;

        current->next = prev;

        prev = current;

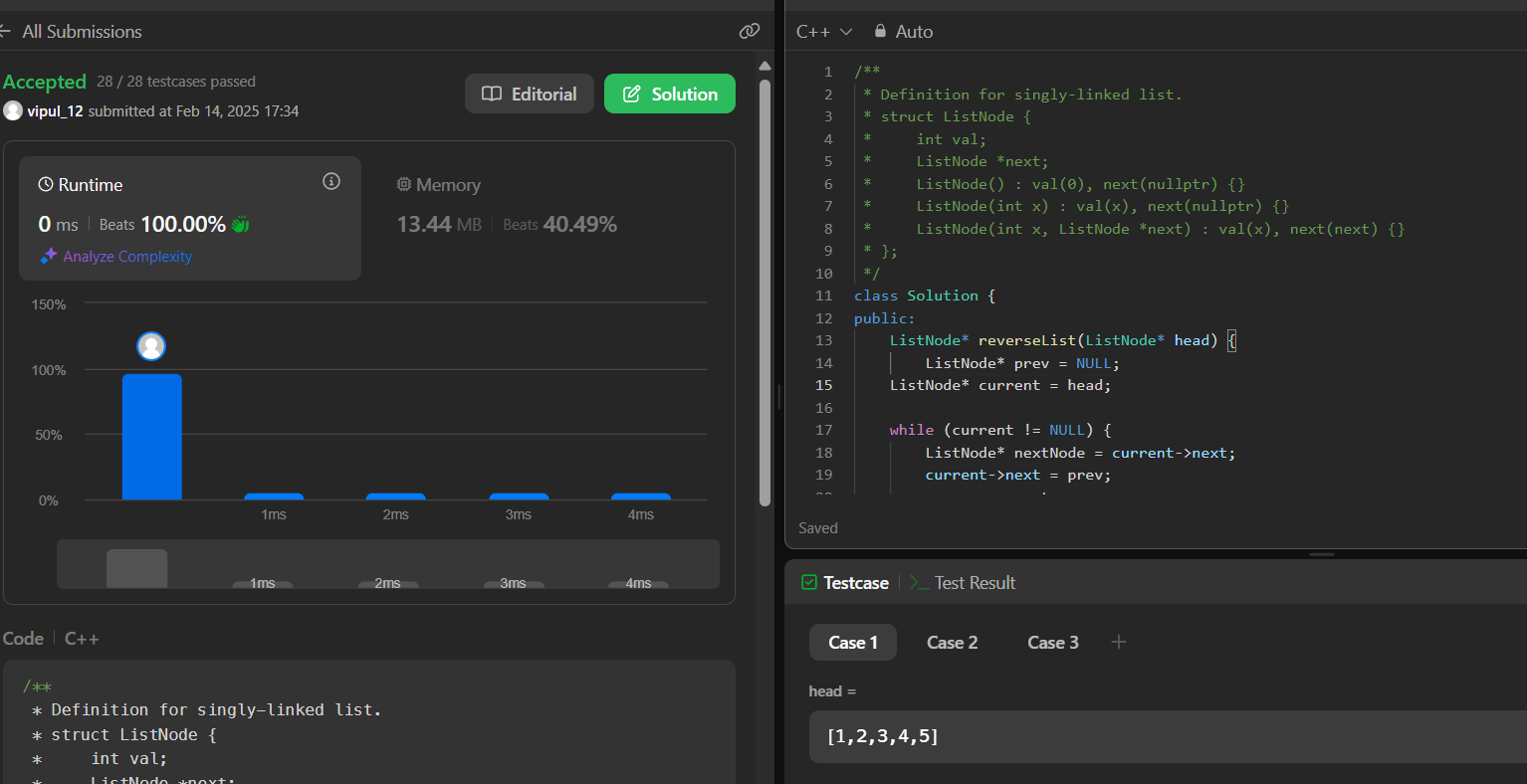
        current = nextNode;

    }

    return prev;

    }

};



**Delete middle node of a list**

class Solution {

public:

    ListNode\* deleteMiddle(ListNode\* head) {

         if (!head || !head->next) return nullptr;

        ListNode \*slow = head, \*fast = head, \*prev = nullptr;

        while (fast && fast->next) {

            prev = slow;

            slow = slow->next;

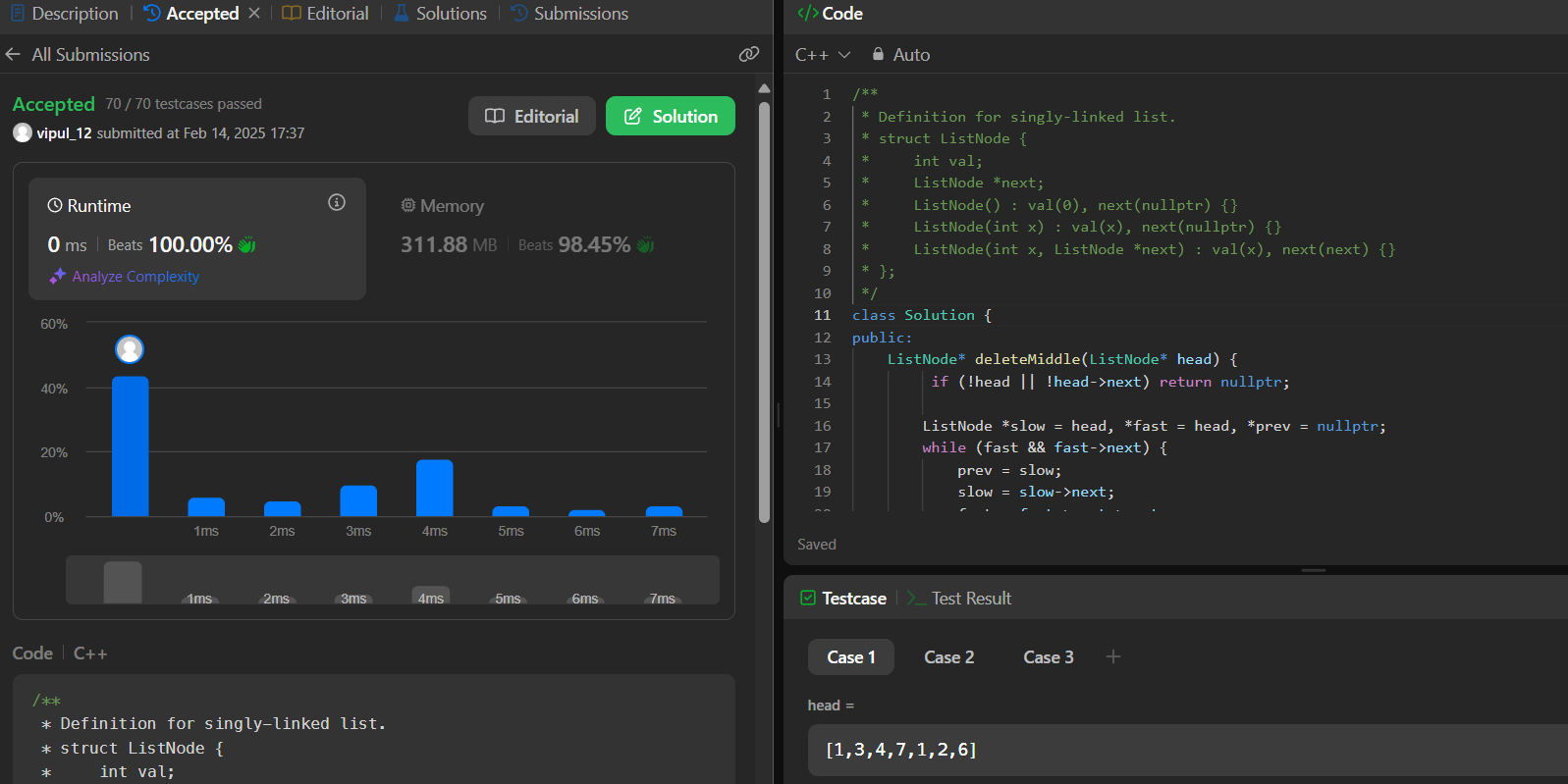
            fast = fast->next->next;   }

        prev->next = slow->next;

        return head;

    }

};



[**Merge two sorted linked lists**](https://leetcode.com/problems/merge-two-sorted-lists/description/)

class Solution {

public:

    ListNode\* mergeTwoLists(ListNode\* list1, ListNode\* list2) {

        ListNode dummy(0);

        ListNode\* tail = &dummy;

        while (list1 && list2) {

            if (list1->val < list2->val) {

                tail->next = list1;

                list1 = list1->next;

            } else {

                tail->next = list2;

                list2 = list2->next;

            }

            tail = tail->next;

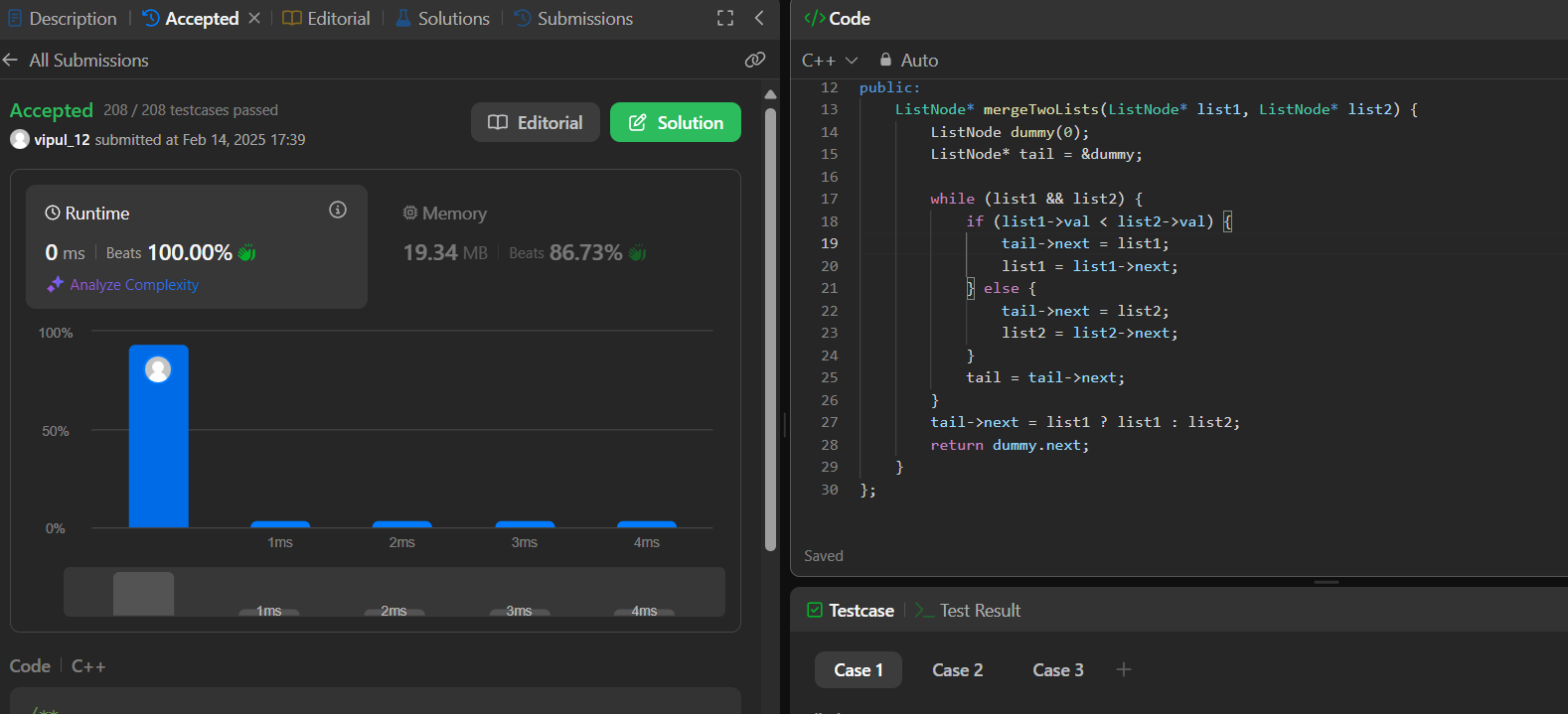
        }

        tail->next = list1 ? list1 : list2;

        return dummy.next;

    }

};



[**Remove duplicates from sorted lists 2**](https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/description/)

class Solution {

public:

    ListNode\* deleteDuplicates(ListNode\* head) {

        if (!head || !head->next) return head;

        ListNode dummy(0);

        dummy.next = head;

        ListNode\* prev = &dummy;

        while (head) {

            if (head->next && head->val == head->next->val) {

                while (head->next && head->val == head->next->val) {

                    head = head->next;

                }

                prev->next = head->next;

            } else {

                prev = prev->next;

            }

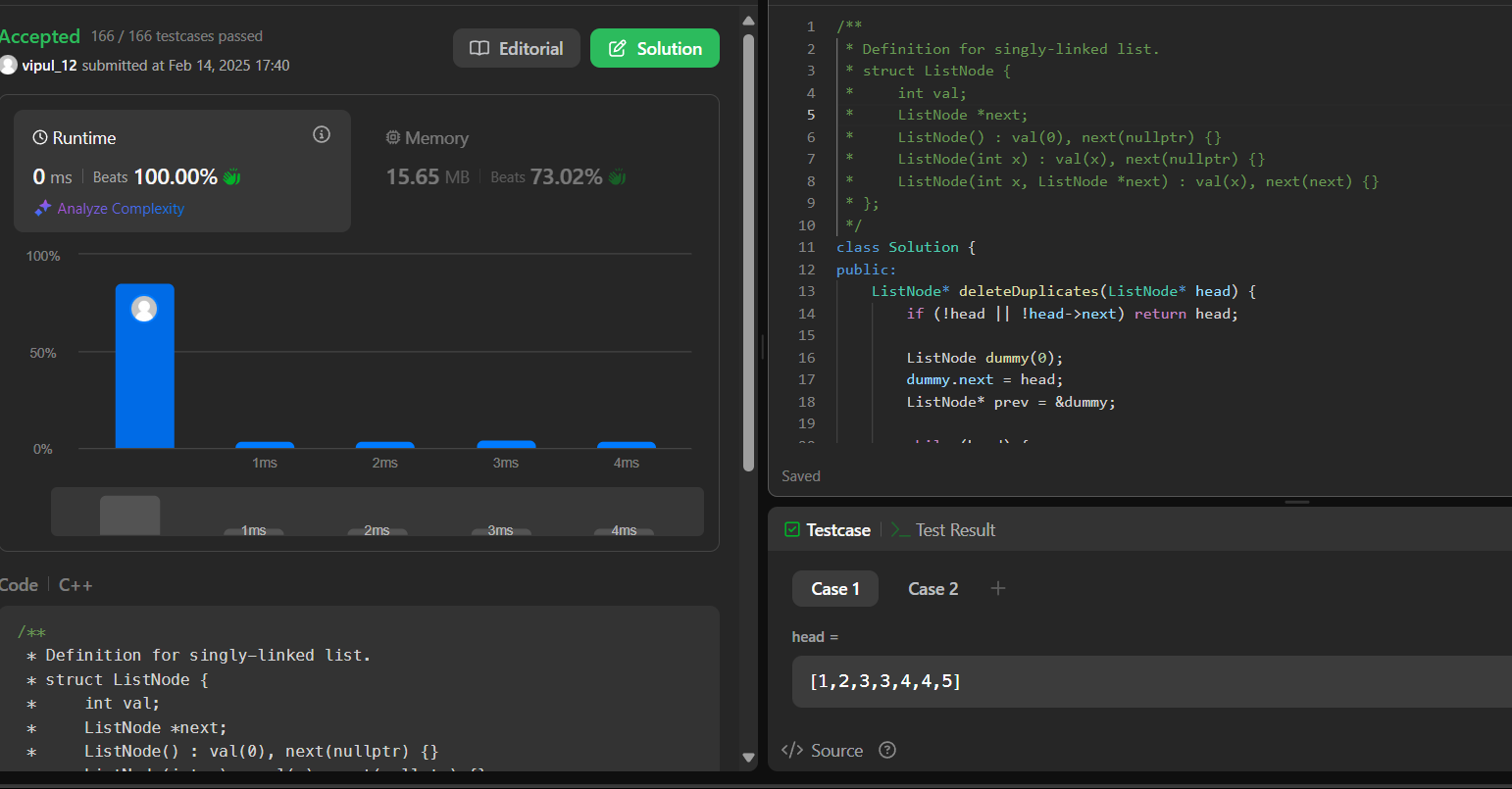
            head = head->next;

        }

        return dummy.next;

    }

};



[**Detect a cycle in a linked list**](https://leetcode.com/problems/linked-list-cycle/description/)

class Solution {

public:

    bool hasCycle(ListNode \*head) {

        if (!head || !head->next) return false;

        ListNode \*slow = head;

        ListNode \*fast = head;

        while (fast && fast->next) {

            slow = slow->next;

            fast = fast->next->next;

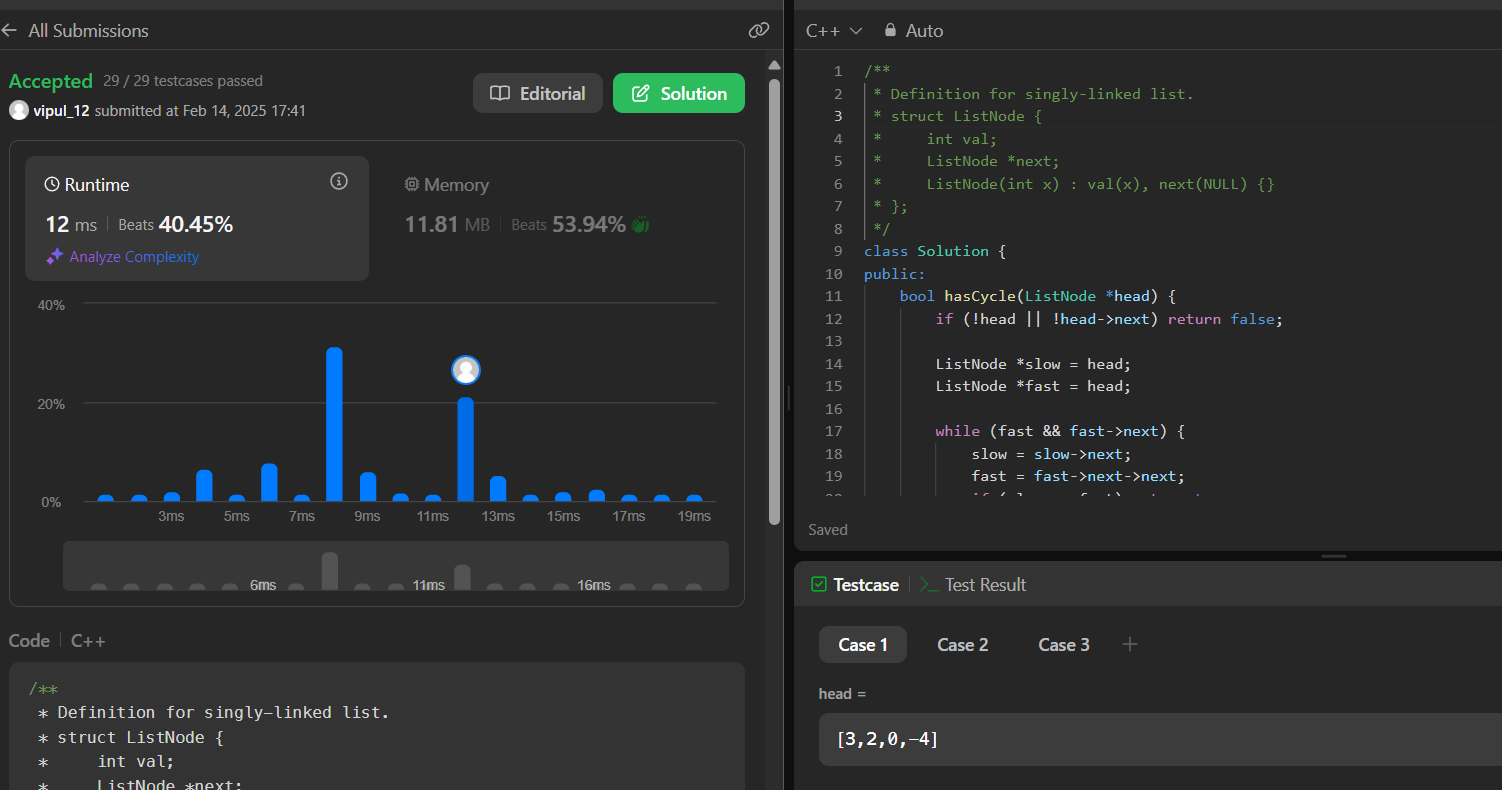
            if (slow == fast) return true;

        }

        return false;

    }

};



[**Reverse linked list 2**](https://leetcode.com/problems/reverse-linked-list-ii/description/)

class Solution {

public:

    ListNode\* reverseBetween(ListNode\* head, int left, int right) {

        if (!head || left == right) return head;

        ListNode dummy(0);

        dummy.next = head;

        ListNode\* prev = &dummy;

        for (int i = 1; i < left; i++) {

            prev = prev->next;

        }

        ListNode\* curr = prev->next;

        ListNode\* nextNode;

        for (int i = 0; i < right - left; i++) {

            nextNode = curr->next;

            curr->next = nextNode->next;

            nextNode->next = prev->next;

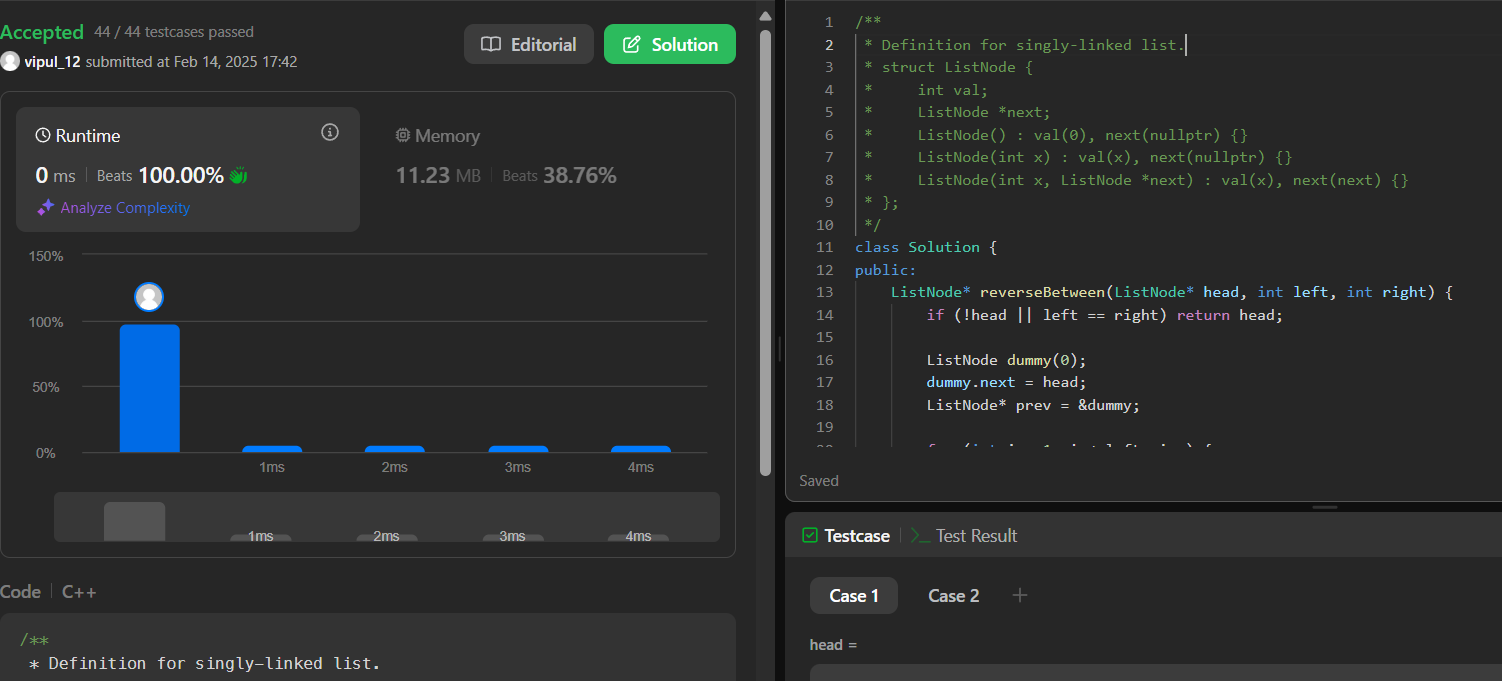
            prev->next = nextNode;

        }

        return dummy.next;

    }

};



[**rotate a list**](https://leetcode.com/problems/rotate-list/description/)

class Solution {

public:

    ListNode\* rotateRight(ListNode\* head, int k) {

         if (!head || !head->next || k == 0) return head;

        ListNode\* tail = head;

        int length = 1;

        while (tail->next) {

            tail = tail->next;

            length++;

        }

        k = k % length;

        if (k == 0) return head;

        tail->next = head;

        for (int i = 0; i < length - k - 1; i++) {

            head = head->next;

        }

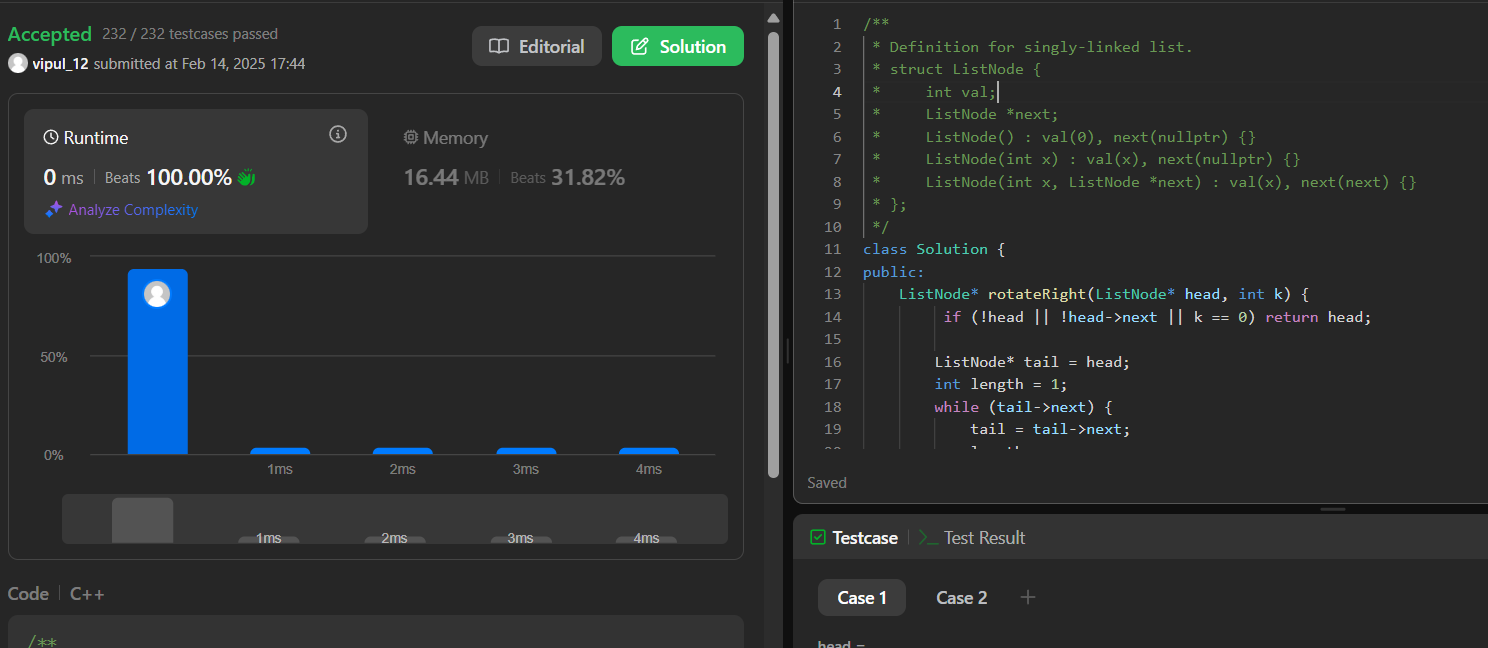
        ListNode\* newHead = head->next;

        head->next = nullptr;

        return newHead;

    }

};



[**Detect a cycle in a linked list 2**](https://leetcode.com/problems/linked-list-cycle-ii/description/)

class Solution {

public:

    ListNode \*detectCycle(ListNode \*head) {

         ListNode \*slow = head, \*fast = head;

        while (fast && fast->next) {

            slow = slow->next;

            fast = fast->next->next;

            if (slow == fast) {

                slow = head;

                while (slow != fast) {

                    slow = slow->next;

                    fast = fast->next;

                }

                return slow;

            }

        }

        return nullptr;

    }

};

