链表

203. 移除链表元素

203. 移除链表元素 - 力扣

给你一个链表的头节点 `head` 和一个整数 `val` ，请你删除链表中所有满足 `Node.val == val` 的节点，并返回 **新的头节点** 。

```
class Solution {
public:
    ListNode* removeElements(ListNode* head, int val) {
        ListNode* dummyHead = new ListNode(0);
        dummyHead->next = head;
        ListNode* cur = dummyHead;
        while (cur->next != nullptr) {
            if(cur->next->val == val) {
                ListNode* tmp = cur->next;
                cur->next = cur->next->next;
                delete tmp;
            } else {
                cur = cur->next;
            }
        }
        head = dummyHead->next;
        delete dummyHead;
        return head;
    }
};
```

707. 设计链表

707. 设计链表 - 力扣

你可以选择使用单链表或者双链表，设计并实现自己的链表。

单链表中的节点应该具备两个属性：`val` 和 `next` 。`val` 是当前节点的值，`next` 是指向下一个节点的指针/引用。

如果是双向链表，则还需要属性 `prev` 以指示链表中的上一个节点。假设链表中的所有节点下标从 **0** 开始。

实现 `MyLinkedList` 类：

- `MyLinkedList()` 初始化 `MyLinkedList` 对象。
- `int get(int index)` 获取链表中下标为 `index` 的节点的值。如果下标无效，则返回 `-1` 。
- `void addAtHead(int val)` 将一个值为 `val` 的节点插入到链表中第一个元素之前。在插入完成后，新节点会成为链表的第一个节点。
- `void addAtTail(int val)` 将一个值为 `val` 的节点追加到链表中作为链表的最后一个元素。

- `void addAtIndex(int index, int val)` 将一个值为 `val` 的节点插入到链表中下标为 `index` 的节点之前。如果 `index` 等于链表的长度，那么该节点会被追加到链表的末尾。如果 `index` 比长度更大，该节点将 **不会插入** 到链表中。
- `void deleteAtIndex(int index)` 如果下标有效，则删除链表中下标为 `index` 的节点。

```
class MyLinkedList {
public:
    struct ListNode
    {
        int val;
        ListNode* next;
        ListNode(int val): val(val),next(nullptr){};
    };
    MyLinkedList() {
        size = 0;
        dummyhead = new ListNode(0);
    }

    int get(int index) {
        if ((index<0)||(index>=size))
        {
            return -1;
        }
        ListNode* cur = dummyhead;
        for (int i = 0; i<index;i++)
        {
            cur = cur->next;
        }
        int result = cur->next->val;
        return result;


    }

    void addAtHead(int val) {
        ListNode* newNode = new ListNode(val);
        ListNode* cur = dummyhead;
        newNode->next = cur->next;
        cur->next = newNode;
        size++;
    }

    void addAtTail(int val) {
        ListNode* newNode = new ListNode(val);
        ListNode* cur = dummyhead;
        while (cur->next!=nullptr)
        {
            cur = cur->next;
        }
        newNode->next = cur->next;
        cur->next = newNode;
        size++;
    }
```

```
    void addAtIndex(int index, int val) {
        if (index <= size)
        {
            ListNode* newNode = new ListNode(val);
            ListNode* cur = dummyhead;
            for (int i = 0; i<index;i++)
            {
                cur = cur->next;
            }
            newNode->next = cur->next;
            cur->next = newNode;
            size++;
        }
    }

    void deleteAtIndex(int index) {
        ListNode* cur = dummyhead;
        if (index <= size-1)
        {
            for (int i = 0; i<index;i++)
            {
                cur = cur->next;
            }
            ListNode* temp = cur->next;
            cur->next = cur->next->next;
            delete temp;
            size--;
        }
    }

private:
    int size;
    ListNode* dummyhead;
};

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * MyLinkedList* obj = new MyLinkedList();
 * int param_1 = obj->get(index);
 * obj->addAtHead(val);
 * obj->addAtTail(val);
 * obj->addAtIndex(index,val);
 * obj->deleteAtIndex(index);
 */
```

注意AddAtIndex函数是在第 index 个节点之前添加元素，所以判断条件为index <= size

206. 反转链表

206. 反转链表 - 力扣

给你单链表的头节点 `head` ，请你反转链表，并返回反转后的链表。

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* cur = head;
        ListNode* pre = nullptr;
        while(cur != nullptr){
            ListNode *temp = cur->next;
            cur->next = pre;
            pre = cur;
            cur = temp;
        }
        return pre;
    }
};
```

24. 两两交换链表中的节点

24. 两两交换链表中的节点 - 力扣

```
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        ListNode* dummyhead = new ListNode(0,head);
        ListNode*cur = dummyhead;
        while (cur!=nullptr && cur->next!=nullptr && cur->next->next!=nullptr)
        {
            ListNode* temp1 = cur->next;
            ListNode* temp2 = cur->next->next->next;
            cur->next = cur->next->next;
            cur = cur->next;
            cur->next = temp1;
            cur = cur->next;
            cur->next = temp2;
        }
        return dummyhead->next;
    }
};
```

19. 删除链表的倒数第N个结点

19. 删除链表的倒数第 N 个结点 - 力扣

给你一个链表，删除链表的倒数第 `n` 个结点，并且返回链表的头结点。

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        int size = 0;
        ListNode* dummyhead = new ListNode(0,head);
        ListNode* cur = dummyhead;
```

```
        while( cur->next!=nullptr )
        {
            cur=cur->next;
            size++;
        }
        cur = dummyhead;
        int index = size-n;
        for (int i = 0; i<index; i++)
        {
            cur = cur->next;
        }
        ListNode* temp = cur->next;
        cur->next= cur->next->next;
        delete temp;
        return dummyhead->next;
    }
};
```