Practice Set III

More Contents on GitHub.com/RajSDE

Implementing a Linked List in Java using Class

```
class Node{
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
```

```
import java.io.*;
import java.util.*;

public class LinkedListCreation {
    class Node {
        int data;
        Node next;
        // constructor to create new node
        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    // Initially both head and tail are not pointing to any other node
    Node head = null;
    Node tail = null;
    void addNode(int data) {
        Node newNode = new Node(data);
        // Checks if the list is empty
        if (head == null) {
            // If list is empty, both head and tail will point to new node
            head = newNode;
            tail = newNode;
        } else {

            tail.next = newNode;
            // storing newnode in tail
            tail = newNode;
        }
```

```
    }
    void displayNodes() {
        Node current = head;
        if (head == null) {
            System.out.println("Empty");
            return;
        }
        System.out.println("Nodes : ");
        while (current != null) {

            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
    int countNodes() {
        // Initially zero
        int count = 0;
        Node currentNode = head;
        // iterate until all the nodes are present
        while (currentNode != null) {

            count++;
            currentNode = currentNode.next;
        }
        return count;
    }
    public static void main(String[] args) {
        LinkedListCreation L1 = new LinkedListCreation();
        L1.addNode(1);
        L1.addNode(2);
        L1.addNode(3);
        L1.addNode(4);
        L1.displayNodes();
        // Counts the nodes present in the given list
        System.out.println("Total Nodes: " + L1.countNodes());
    }
}
```

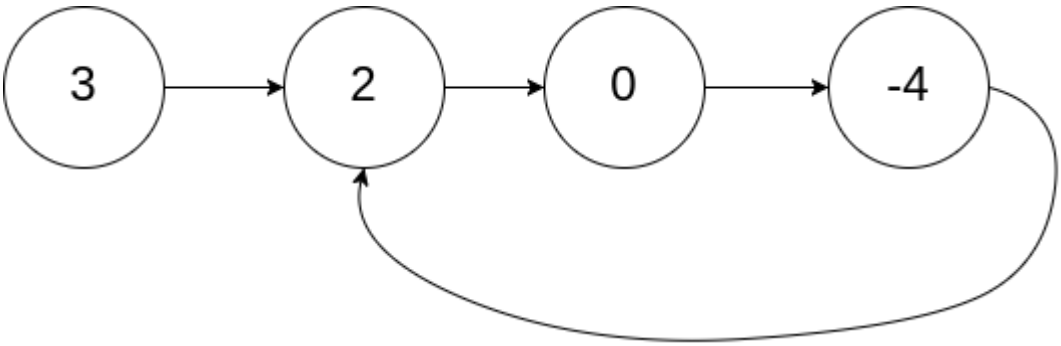
141. Linked List Cycle

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:

Input: head = [1], pos = -1

Output: false

Explanation: There is no cycle in the linked list.

```java
public class Solution {
    public boolean hasCycle(ListNode head) {
        ListNode s = head;
        ListNode f = head;
        while(f!=null && f.next!=null){
            s = s.next;
            f = f.next.next;
            if(s==f) return true;
        }
        return false;
    }
}
```

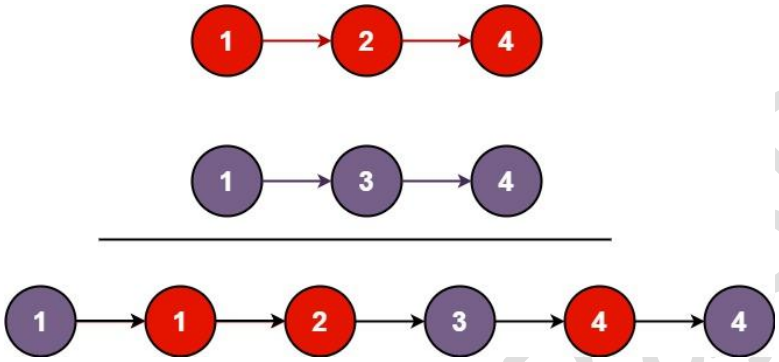
21. Merge Two Sorted Lists

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

Merge the two lists in a 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:



Input: list1 = [1,2,4], list2 = [1,3,4]

Output: [1,1,2,3,4,4]

Example 2:

Input: list1 = [], list2 = []

Output: []

Example 3:

Input: list1 = [], list2 = [0]

Output: [0]

1st Approach

```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        if(list1 == null)
            return list2;
        if(list2 == null)
            return list1;

        if(list1.val <= list2.val){
            list1.next = mergeTwoLists(list1.next, list2);
            return list1;
        }
        else{
            list2.next = mergeTwoLists(list1, list2.next);
            return list2;
        }
```

```
    }
}
```

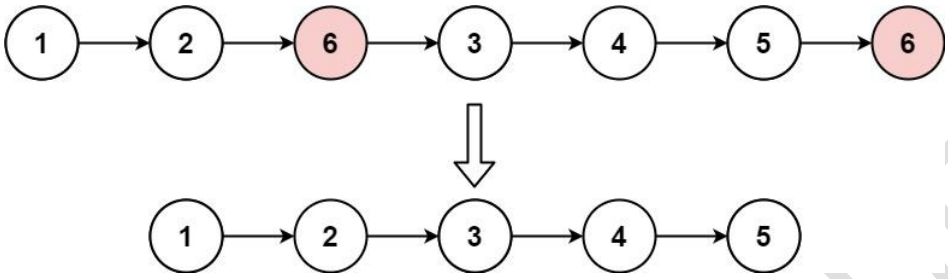
2nd Approach

```
public class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode head = new ListNode(0);
        ListNode handler = head;
        while(l1 != null && l2 != null) {
            if (l1.val <= l2.val) {
                handler.next = l1;
                l1 = l1.next;
            } else {
                handler.next = l2;
                l2 = l2.next;
            }
            handler = handler.next;
        }

        if (l1 != null) {
            handler.next = l1;
        } else if (l2 != null) {
            handler.next = l2;
        }

        return head.next;
    }
}
```

203. Remove Linked List Elements

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 = [7,7,7,7], val = 7

Output: []

1st Approach

```
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        if(head==null) return head;
        ListNode resultHead = new ListNode(-1);
        resultHead.next=head;
        ListNode curr = head, prev=resultHead;

        while(curr!=null){
            // ListNode temp2=temp.next;
            if(curr.val == val){
                prev.next=curr.next;
            }
            else{
                prev=curr;
            }
            curr=curr.next;
        }
        return resultHead.next;
    }
}
```

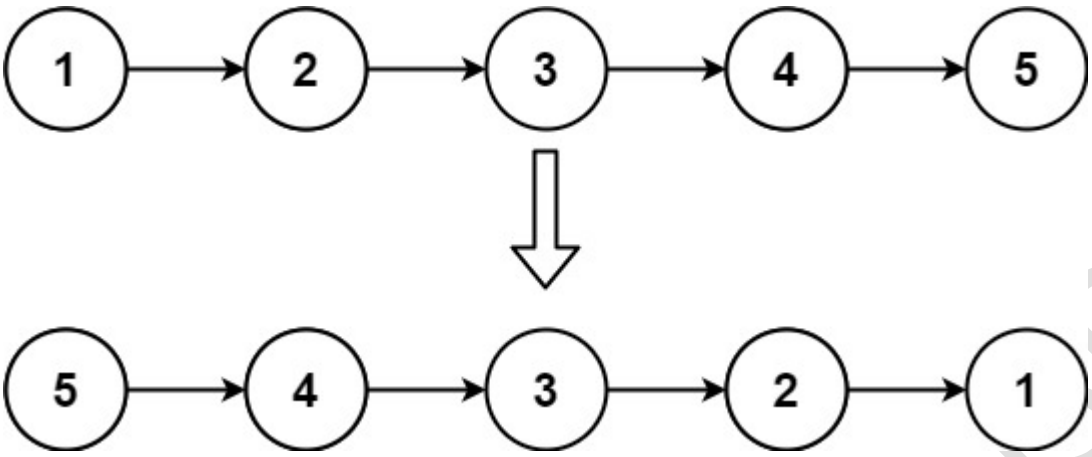
2nd Approach

```
public ListNode removeElements(ListNode head, int val) {
        if (head == null) return null;
        head.next = removeElements(head.next, val);
        return head.val == val ? head.next : head;
}
```

206. Reverse Linked List

Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:



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

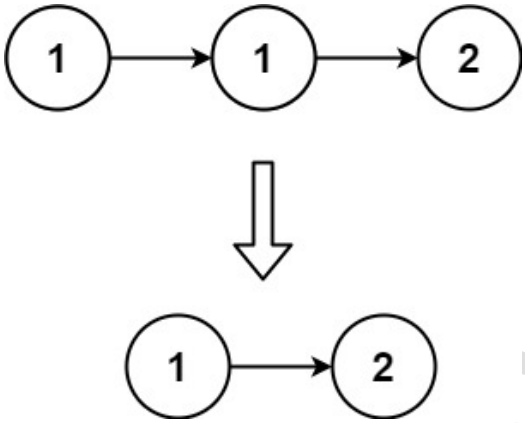
Output: [5,4,3,2,1]

```
class Solution {
    public ListNode reverseList(ListNode head) {
        // base case
        if(head==null || head.next == null)
            return head;
        // iterate till last node
        ListNode newHead = reverseList(head.next);
        head.next.next = head;
        head.next=null;
        return newHead;
    }
}
```

83. Remove Duplicates from Sorted List

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]

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if(head==null || head.next==null)
            return head;

        head.next=deleteDuplicates(head.next);
        if(head.val==head.next.val){
            return head.next;
        }
        else{
            return head;
        }
    }
}
```

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode temp = head;
        while(temp!=null && temp.next!=null){
            if(temp.val==temp.next.val){
                temp.next=temp.next.next;
            }
            else
                temp=temp.next;
        }
        return head;
    }
}
```