Sunil Yadav

TOP INTERVIEW QUESTIONS

EASY COLLECTIONS

LINKED LIST

INTRODUCTION

Linked List problems are relatively easy to master. Do not forget the Two-pointer technique, which not only applicable to Array problems but also Linked List problems as well.

Another technique to greatly simplify coding in linked list problems is the dummy node trick.

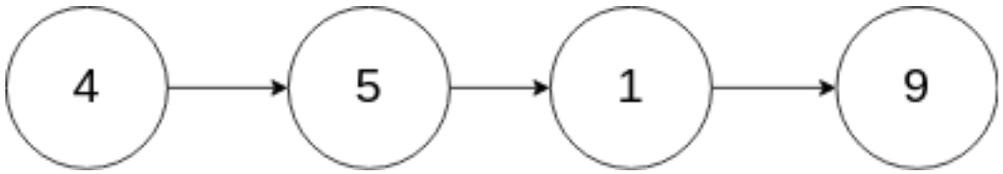
We recommend: Reverse Linked List, Merge Two Sorted Lists and Linked List Cycle.

For additional challenge, solve these problems recursively: Reverse Linked List, Palindrome Linked List and Merge Two Sorted Lists.

Delete Node in a Linked List

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.

Given linked list -- head = [4,5,1,9], which looks like following:



Example 1:

```
Input: head = [4,5,1,9], node = 5
Output: [4,1,9]
Explanation: You are given the second node with value 5, the linked list should become 4 -> 1 -> 9 after calling your function.
```

Example 2:

```
Input: head = [4,5,1,9], node = 1
Output: [4,5,9]
Explanation: You are given the third node with value 1, the linked list should become 4 -> 5 -> 9 after calling your function.
```

Note:

- The linked list will have at least two elements.
- All of the nodes' values will be unique.
- The given node will not be the tail and it will always be a valid node of the linked list.
- Do not return anything from your function.

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public void deleteNode(ListNode node) {
     ListNode Prev = null;
    while(node.next!=null){
        node.val = node.next.val;
        Prev = node;
        node = node.next;
    }
        if(Prev!=null)
        Prev.next = null;
    }
}
```

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        ListNode Prev = null;
        ListNode fN = head;
        ListNode slow = head;

        int counter = 1;
        while(head!=null){
            if(counter>n){
                Prev = slow;
                slow = slow.next;
            }
            counter++;
            head = head.next;
        }
        if(Prev!=null){
            if(slow!=null)
            Prev.next = slow.next;
            else
            Prev.next = null;
        }
        if(Prev == null){
            return fN.next;
        }
        return fN;
    }
}
```

Remove Nth Node From End of List

Given a linked list, remove the *n*-th node from the end of list and return its head.

Example:

```
Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.
```

Note:

Given *n* will always be valid.

Follow up:

Could you do this in one pass?

-

Reverse Linked List

Reverse a singly linked list.

Example:

```
Input: 1->2->3->4->5->NULL
Output: 5->4->3->2->1->NULL
```

Follow up:

A linked list can be reversed either iteratively or recursively. Could you implement both?

-

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode root = head;
        ListNode Prev = null;
        while(head!=null){
            ListNode nextNode = head.next;
            head.next = Prev;
            Prev = head;
            head = nextNode;
        }
        return Prev;
    }
}
```

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode x = l1;
        ListNode y = l2;
        ListNode result = new ListNode(-1);
        ListNode fR = result;
        while(x!=null && y!=null){
            if(x.val <= y.val){
                result.next = new ListNode(x.val);
                x = x.next;
            }else{
                result.next = new ListNode(y.val);
                y = y.next;
            }
            result = result.next;
        }
        if(x!=null){
            result.next = x;
            result = result.next;
        }
        if(y!=null){
            result.next = y;
            result = result.next;
        }

        return fR.next;
    }
```

Merge Two Sorted Lists

Merge two sorted linked lists and return it as a new **sorted** list. The new list should be made by splicing together the nodes of the first two lists.

Example:

Input: 1->2->4, 1->3->4
Output: 1->1->2->3->4->4

Palindrome Linked List

Given a singly linked list, determine if it is a palindrome.

Example 1:

```
Input: 1->2
Output: false
```

Example 2:

```
Input: 1->2->2->1
Output: true
```

Follow up:

Could you do it in O(n) time and O(1) space?

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public boolean isPalindrome(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        while(fast!=null && fast.next!=null){
            slow = slow.next;
            fast = fast.next.next;
        }
        Stack<ListNode> stack = new Stack<>();
        // Slow is at Middle Position
        while(slow!=null){
            stack.push(slow);
            slow = slow.next;
        }
        while(!stack.isEmpty()){
            if(head.val!=stack.pop().val){
                return false;
            }
            head = head.next;
        }
        return true;
    }
}
```

```java
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public boolean hasCycle(ListNode head) {
        if(head == null){
            return false;
        }
        ListNode slow = head;
        ListNode fast = head.next;
        while(true){
            if(slow==null || fast==null){
                return false;
            }
            if(slow==fast){
                return true;
            }
            slow = slow.next;
            if(fast.next==null){
                return false;
            }
            fast = fast.next.next;
        }
    }
}
```

Linked List Cycle

Given a linked list, determine if it has a cycle in it.

To represent a cycle in the given linked list, we use an integer `pos` which represents the position (0-indexed) in the linked list where tail connects to. If `pos` is `-1`, then there is no cycle in the linked list.

Example 1:

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