Table of Contents

Delete Node in a Linked List

Source

https://leetcode.com/problems/delete-node-in-a-linked-list/

Question

> Write a function to delete a node (except the tail) in a singly linked list, given only access to that node. Supposed the linked list is `1 -> 2 -> 3 -> 4` and you are given the third node with value `3`, the linked list should become `1 -> 2 -> 4` after calling your function.

Idea

- Typically, to remove a node, we change the prev to point to n.next
- Alternatively, we can copy the next val over, and skip the next node

Solution

Code

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

Complexity

- Time: `O(1)`
- Space: `O(1)`

Things learnt

- Remember that removal can also be accomplished by *overwriting*

Remove Linked List Elements

Source

https://leetcode.com/problems/remove-linked-list-elements/

Question

> Remove all elements from a linked list of integers that have value val. Example Given: `1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6`, `val = 6` Return: `1 --> 2 --> 3 --> 4 --> 5`

Idea

- We can't assume that the first node won't be removed
- We have two choices:
  - Use a temporary node (prehead)
  - Use recursion

Solution (Prehead)

Code

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode preHead = new ListNode(0);
        preHead.next = head;
        ListNode p = preHead;
        while (p.next != null) {
            if (p.next.val == val) {
                p.next = p.next.next;
            } else {
```

```
                p = p.next;
            }
        }
        return preHead.next;
    }
}
```

Complexity

- Time: `O(n)`
- Space: `O(1)`

Things learnt

- Remember that you can always use an additional node

Solution (Recursion)

Code

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode removeElements(ListNode head, int val) {
        if (head == null) {
            return head;
        }
        ListNode tail = removeElements(head.next, val);
        if (head.val == val) {
            return tail;
        } else {
            head.next = tail;
            return head;
        }
    }
}
```

Complexity

- Time: `O(n)`
- Space: `O(n)` due to implicit stack

Reverse Linked List

Source

https://leetcode.com/problems/reverse-linked-list/

Question

> Reverse a singly linked list

Idea

- We can do this iteratively or recursively. Iteratively uses less space, since we don't use an implicit stack

Solution (Iterative)

Approach

Assume that we have linked list `1 → 2 → 3 → Ø`, we would like to change it to `Ø ← 1 ← 2 ← 3`. While you are traversing the list, change the current node's next pointer to point to its previous element. Since a node does not have reference to its previous node, you must store its previous element beforehand. You also need another pointer to store the next node before changing the reference. Do not forget to return the new head reference at the end!

Code

```
public ListNode reverseList(ListNode head) {
    ListNode prev = null;
    ListNode curr = head;
    while (curr != null) {
        ListNode nextTemp = curr.next;
        curr.next = prev;
        prev = curr;
        curr = nextTemp;
    }
    return prev;
}
```

Complexity

- Time: `O(n)`
- Space: `O(1)`

Solution (Recursion)

Approach

The recursive version is slightly trickier and the key is to work backwards. Assume that the rest of the list had already

been reversed, now how do I reverse the front part? Let's assume the list is: `n1 → … → nk-1 → nk → nk+1 → … → nm → Ø` Assume from node `nk+1` to `nm` had been reversed and you are at node nk. `n1 → … → nk-1 → nk → nk+1 ← … ← nm` We want `nk+1`'s next node to point to `nk`. So, `nk.next.next = nk;` Be very careful that n1's next must point to Ø. If you forget about this, your linked list has a cycle in it. This bug could be caught if you test your code with a linked list of size 2.

Code

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode removeElements(ListNode head, int val) {
        if (head == null) {
            return head;
        }
        ListNode tail = removeElements(head.next, val);
        if (head.val == val) {
            return tail;
        } else {
            head.next = tail;
            return head;
        }
    }
}
```

Complexity

- Time: `O(n)`
- Space: `O(1)` due to implicit stack