Documentation for Removing Linked List Elements

Problem Overview:

- The problem is to remove all nodes from a singly-linked list that contain a specific value (val). The goal is to return the modified linked list, where no node has a value equal to val.

Key Concepts:

- **Singly-linked list:** A data structure where each element (node) contains a value and a reference (or pointer) to the next node in the sequence.

- **Node removal:** To remove a node from a singly-linked list, we adjust the pointers so that the node is skipped, and the previous node points to the next node.

Problem Statement:

- Given the head of a singly-linked list and an integer val, the task is to remove all nodes from the list where the value of the node (Node.val) equals val. The function should return the head of the updated list.

Input:

- **head:** A reference to the head of the singly-linked list. If the list is empty, head is None.

- **val:** An integer between 0 and 50, inclusive. This is the value of the nodes to be removed from the list.

Output:

- The modified head of the linked list after all nodes with value val are removed. If all nodes are removed or the list is initially empty, return None.

Example 1:

- **Input:** head = [1, 2, 6, 3, 4, 5, 6], val = 6
- **Output:** [1, 2, 3, 4, 5]
- **Explanation:** Nodes with value 6 are removed, resulting in a list [1, 2, 3, 4, 5].

Example 2:

- **Input:** head = [], val = 1
- **Output:** []
- **Explanation:** The list is empty, so no changes are made.

Example 3:

- **Input:** head = [7, 7, 7, 7], val = 7
- **Output:** []
- **Explanation:** All nodes have the value 7, so the entire list is removed, resulting in an empty list.

Constraints:

- The number of nodes in the list is in the range [0, 10^4].
- Each node's value (Node.val) is between 1 and 50.
- The target value val for removal is between 0 and 50.

Solution Strategy:

1. **Use a Dummy Node:**

- The head node itself may need to be removed if its value equals val. To simplify handling such edge cases, introduce a "dummy" node before the head. This dummy node does not represent an actual list element but points to the head.
- The dummy node allows for uniform handling of all nodes, including the head, since we can always operate on the node following dummy.

2. **Traversal and Deletion:**

- *Start from the dummy node and iterate through the list. For each node:*
  - ➢ If the node's value matches val, update the next pointer of the previous node (or the dummy) to skip the current node.
  - ➢ Otherwise, move the pointer to the next node and continue checking.
- This ensures that all nodes with value val are removed, and other nodes remain intact.

3. **Returning the Modified List:**

- After the traversal is complete, the new head of the list will be the node following the dummy node (dummy.next). This handles the case where the head node itself was removed.

Edge Cases:

- **Empty list:** If the list is empty (head is None), the function should return None.
- **No nodes to remove:** If no node has the value val, the list remains unchanged.
- **All nodes have the value val:** If all nodes have the value val, the entire list will be removed, and the function will return None.
- **Consecutive nodes with the target value:** The algorithm must be able to skip over consecutive nodes with the target value, updating pointers correctly.

Time and Space Complexity:

Time Complexity:

- The algorithm runs in O(n) time, where n is the number of nodes in the linked list. Each node is visited exactly once during traversal.

Space Complexity:

- The space complexity is O(1), as no extra space proportional to the size of the input is used, other than the dummy node and a few pointers.

Algorithm Steps:

1. **Initialize a Dummy Node:** Create a dummy node and point it to the head of the list. This handles cases where the head itself might be removed.

2. **Iterate Through the List:**

- Set a pointer (current) to the dummy node.
- ***Traverse the list:***
  - ➢ If the next node's value equals val, skip that node by adjusting the next pointer to the node after it.
  - ➢ If the value is different, move the pointer to the next node.

3. **Return the New Head:** After the traversal, return the node following the dummy (dummy.next) as the new head of the list.

Advantages of Using a Dummy Node:

- Simplifies the logic when the head node needs to be removed.
- Prevents the need for special handling of the head in a separate case, as all nodes are treated uniformly.
- Makes the code cleaner and easier to understand, especially when removing nodes from the beginning of the list.

Summary:

- By using a dummy node and iterating through the list while adjusting pointers, we efficiently remove all nodes with the target value val and return the modified list. The solution works in linear time and constant space, ensuring scalability even for large linked lists.