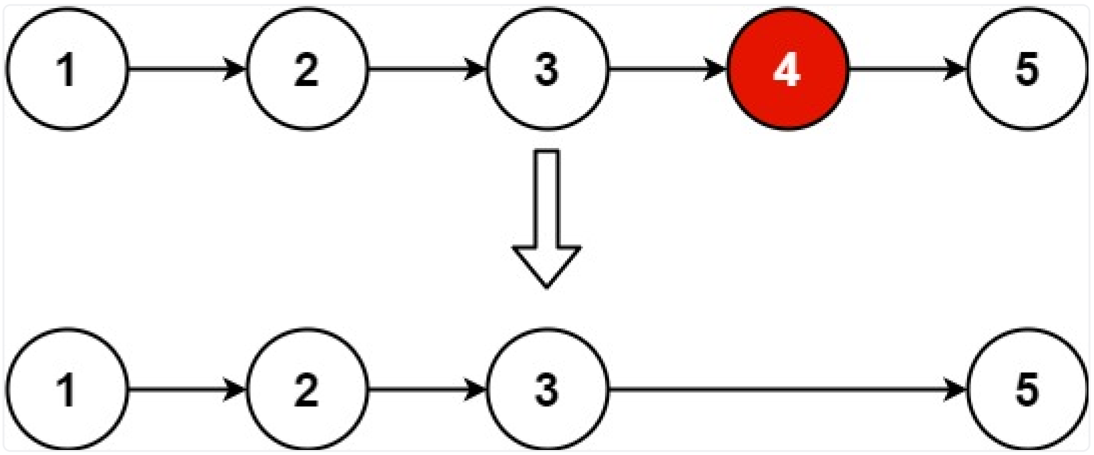
Remove Nth Node From End Of List(Day 22)

Prepared By Azan Imtiaz

Remove Nth Node From End Of List(Leatcode)

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

Example 1:



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

Output: [1,2,3,5]

Example 2:

Input: head = [1], n = 1

Output: []

Example 3:

Input: head = [1,2], n = 1

Output: [1]

Constraints:

The number of nodes in the list is `sz`.

`1 <= sz <= 30`

`0 <= Node.val <= 100`

`1 <= n <= sz`

Follow up: Could you do this in one pass?

Solution 1: Brute Force Solution

Intuition of the First Solution:

The brute force approach calculates the total length of the linked list to find the node to remove. We then make a second pass to adjust pointers and remove the node.

Approach Used in the First Solution:

Calculate the Length of the Linked List:

Traverse the entire linked list to determine its length.

Determine the Node to Remove:

Calculate the position of the node to be removed from the start, using the list's length and the given n.

Remove the N-th Node from the End:

Find the node just before the one we want to remove and adjust the next pointer to skip the node to be removed.

Exact Code:

```
class Solution {

    public ListNode removeNthFromEnd(ListNode head, int n) {
        // Edge case: if the list has only one node
        if (head.next == null) {
            return null;
        }

        // Calculate the length of the linked list
        ListNode temp = head;
        int length = 0;
        while (temp != null) {
            length++;
            temp = temp.next;
        }

        // If we need to remove the head
        if (n == length) {
            return head.next;
        }

        // Find the (length - n)th node
        int jump = length - n;
        temp = head;
        int i = 1; // Start from the first node
        while (i < jump) { // Stop one node before the target node
            temp = temp.next;
            i++;
        }

        // Remove the nth node from the end
        temp.next = temp.next.next;
        return head;
    }
}
```

Explanation of the Code:

Edge Case Handling:

Return null if there is only one node since removing it leaves the list empty.

Length Calculation:

A loop runs through the list to calculate its length.

Removing the Head Node:

If n equals length, we remove the head by returning head.next.

Traversal to the Target Node:

Calculate jumps needed to reach the node before the one to be removed and perform a second traversal.

Removing the Node:

Update the next pointer of the previous node to skip the node to be removed.

Dry Run on a Test Case:

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

Length is 5.

Node to remove is the 3rd from the start.

Traverse to the node before the 4th node.

Adjust pointers to remove node 4.

Result: `1 -> 2 -> 3 -> 5`.

Time Complexity of the First Solution:

Time Complexity: O(L), where L is the length of the linked list.

Space Complexity: O(1), as no additional space is used.

Defects in the First Solution:

Inefficiency: Two passes through the list can be inefficient for long lists.

Not Optimal: It works but isn't the best solution as it can be done in one pass.

Solution 2: One-Pass Solution Using Two Pointers

Intuition of the Second Solution:

We use two pointers with a fixed gap to traverse the list. When the second pointer reaches the end, the first pointer will be at the correct position to remove the node.

Approach Used in the Second Solution:

Create a Dummy Node:

Add a dummy node at the beginning to simplify edge cases.

Initialize Two Pointers:

Set both pointers at the dummy node.

Advance the Second Pointer:

Move the second pointer n nodes ahead to maintain the gap.

Move Both Pointers Until the End:

Move both pointers together until the second one reaches the end.

Remove the N-th Node from the End:

Adjust the next pointer of the first pointer to skip the node to be removed.

Exact Code:

```
class Solution {

    public ListNode removeNthFromEnd(ListNode head, int n) {
        // create a dummy node
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode ptr1 = dummy;
        ListNode ptr2 = dummy;

        // move ptr2 n nodes ahead
        for (int i = 0; i < n; i++) {
            ptr2 = ptr2.next;
        }

        // Now move both pointers until ptr2.next is equal to null
        while (ptr2.next != null) {
            ptr1 = ptr1.next;
            ptr2 = ptr2.next;
        }

        // delete the next node of ptr1
        ptr1.next = ptr1.next.next;

        return dummy.next;
    }
}
```

Explanation of the Code:

Dummy Node Creation:

A dummy node is created to handle cases like removing the head node.

Initialize Pointers:

Set both ptr1 and ptr2 to the dummy node.

Advance the Second Pointer:

Move ptr2 n steps ahead to ensure the correct gap.

Traverse the List with Two Pointers:

Move both pointers until ptr2 reaches the end.

Remove the Node:

Change ptr1.next to skip the node to be removed.

Dry Run on a Test Case:

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

Initialize dummy node.

Move ptr2 2 steps ahead.

Move both pointers together until ptr2 reaches the end.

Remove node 4 by adjusting pointers.

Result: `1 -> 2 -> 3 -> 5`.

Time Complexity of the Second Solution:

Time Complexity: O(L), with L being the list length.

Space Complexity: O(1), as no extra space is used.

Defects in the Second Solution:

None: This solution is optimal, handling all cases efficiently in one pass.

Using the two-pointer technique, the second solution improves efficiency over the brute force method, achieving the desired outcome with just one list traversal

Thank for Reading

Subscribe to My Youtube Channel **Azan's Coding Corner**

Follow Me on Linkedin **Azan Imtiaz**