#### Approach 2: One pass algorithm

**Algorithm**

The above algorithm could be optimized to one pass. Instead of one pointer, we could use two pointers. The first pointer advances the list by n+1n+1 steps from the beginning, while the second pointer starts from the beginning of the list. Now, both pointers are exactly separated by nn nodes apart. We maintain this constant gap by advancing both pointers together until the first pointer arrives past the last node. The second pointer will be pointing at the nnth node counting from the last. We relink the next pointer of the node referenced by the second pointer to point to the node's next next node.

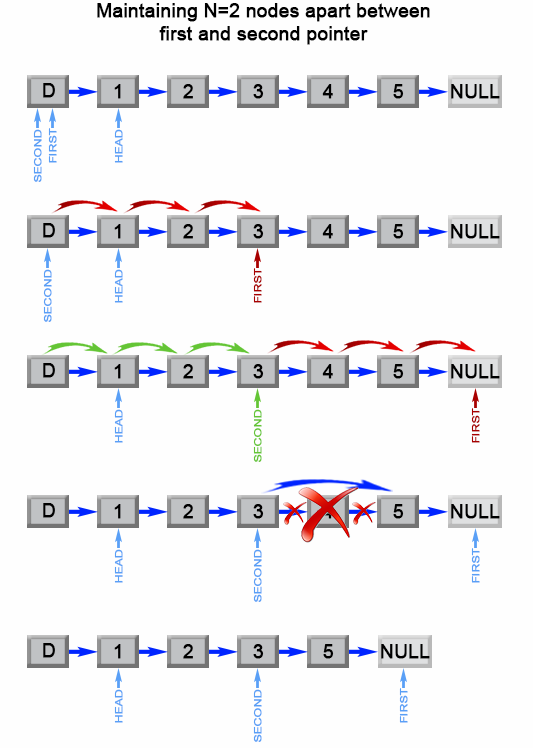


Figure 2. Remove the nth element from end of a list.

**public ListNode removeNthFromEnd(ListNode head, int n) {**

**ListNode dummy = new ListNode(0);**

**dummy.next = head;**

**ListNode first = dummy;**

**ListNode second = dummy;**

**// Advances first pointer so that the gap between first and second is n nodes apart**

**for (int i = 1; i <= n + 1; i++) {**

**first = first.next;**

**}**

**// Move first to the end, maintaining the gap**

**while (first != null) {**

**first = first.next;**

**second = second.next;**

**}**

**second.next = second.next.next;**

**return dummy.next;**

**}**

**Complexity Analysis**

* Time complexity : O(L)*O*(*L*).

The algorithm makes one traversal of the list of L*L* nodes. Therefore time complexity is O(L)*O*(*L*).

* Space complexity : O(1)*O*(1).

We only used constant extra space.