Problem Name: Add Two Numbers

Link: https://leetcode.com/problems/add-two-numbers/description/

Solution Language(s): Java

Approach

For this problem we are adding two linked lists together as if they represented reversed integers. At first, my idea was to reverse both linked lists, sum them, and then reverse them again. Although this seemed intuitive at first, I quickly realized that this would require me to balance the linked lists with leading zeros after reversing them. Additionally, the actual summing of the linked lists involves nearly identical logic anyways, first summing the values and checking for sums over 10 in the case of carry overs. At this point it was apparent that there was no need for reversing the lists, and that I only had to iron out the details of the carry over logic. Now, I had an idea for my pseudocode.

Pseudocode

```
Node for the head of a list
Another temporary node set to the head
Carry integer
While there are values in the linked lists or the carry integer has a nonzero value
    If the linked lists have values, assign them to variables
    Create a variable for the sum the value variables with the carry variable
    Create a node after the temporary node and set its value to the sum, accounting for carry over
    Set the value of the carry variable based on the sum variable
    Move the temporary node to the newly created node
    If valid, advance the linked lists for the next iteration
Return the node after the head node since the head node was never assigned a value
```

Solution

```java
/**
 * 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 addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode head = new ListNode();
        ListNode temp = head;
        int carry = 0;
        // while one of the lists or the carry over still is still valid,
        // continue to loop
        while (l1 != null || l2 != null || carry != 0) {
            int val1 = 0, val2 = 0;
            // if l1/l2 are valid, then set the respective value
            //accordingly
            if (l1 != null) val1 = l1.val;
            if (l2 != null) val2 = l2.val;
            // calculate the sum for the current node position
            int sum = val1 + val2 + carry;
            // if the sum is greater than 9, there will be a carry to the
            //next position
            carry = sum / 10;
            // create the next node and set its value accordingly
            temp.next = new ListNode(sum % 10);
            temp = temp.next;
            // advance l1 and l2 if possible
            if (l1 != null) l1 = l1.next;
            if (l2 != null) l2 = l2.next;
        }
        // since head was never assigned a value, the real head node is the
        // node after it
        return head.next;
    }
}
```

Time Complexity: O(max(n, m))

For my solution the time complexity is O(n) or O(m), whichever is greater, for n and m, which are the list lengths of l1 and l2 respectively.

Conclusion

In conclusion, this problem required an understanding of linked lists and involved a simple rolling carried digit in order to solve. Although my first idea for the solution was unnecessarily complicated and convoluted, I did derive my final solution from the actual addition portion of the initial approach. The problem was rather short and straightforward once I figured out the logic for the *carry* variable, but it did take me a couple of minutes to ensure that the very last node was properly set to the the last remainder in cases that the returned list was longer than either of the given lists.