Add Two Number(Day 27)

Prepared By

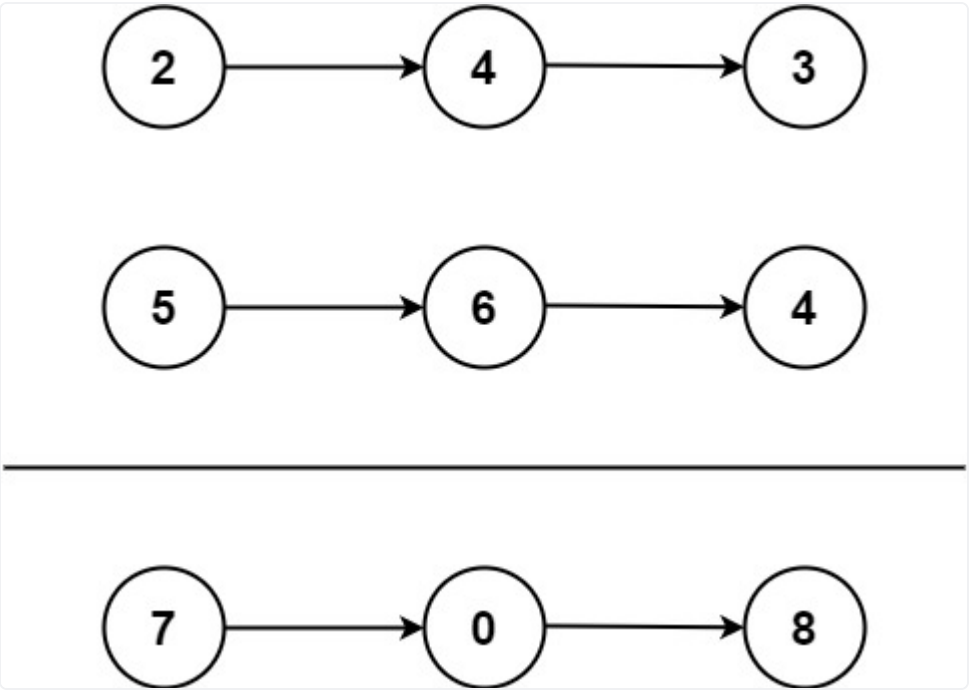
Azan Imtiaz

Add Two Number(Leatcode)

You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example 1:




```
Input: l1 = [2,4,3], l2 = [5,6,4]
Output: [7,0,8]
Explanation: 342 + 465 = 807.
```

Example 2:

```
Input: l1 = [0], l2 = [0]
Output: [0]
```

Example 3:

```
Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
Output: [8,9,9,9,0,0,0,1]
```

Constraints:

- The number of nodes in each linked list is in the range `[1, 100]`.
- `0 <= Node.val <= 9`
- It is guaranteed that the list represents a number that does not have leading zeros.

Problem Statement:

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, meaning that the 1's digit is at the head of the list. Each node contains a single digit. Add the two numbers and return the sum as a linked list.

- You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example:

Input:

`l1 = [2 -> 4 -> 3]`, `l2 = [5 -> 6 -> 4]`

Output:

`[7 -> 0 -> 8]`

Explanation: `342 + 465 = 807` (the lists are stored in reverse order).

Intuition:

The goal is to add two numbers, but instead of regular integers, they are represented as linked lists in reverse order. This means:

- Start by adding the least significant digits (head of both linked lists).
- Keep track of the sum, including any carry from the previous addition.
- Continue moving through the lists until both lists are exhausted.
- If the sum at any stage is greater than or equal to 10, handle the carry (by dividing the sum by 10) and store the last digit (sum % 10) in the resulting list.

Approach:

1. **Initialize a dummy node** to serve as the starting point of the result list, and a pointer (`current`) to build the list.
2. **Keep a carry** initialized to 0.

3. Iterate through both linked lists (`l1` and `l2`) simultaneously:
   - Add the values of the current nodes of `l1` and `l2`, along with the carry from the previous step.
   - Calculate the new carry by dividing the sum by 10 (integer division).
   - Create a new node in the result list with the value equal to the sum mod 10.
4. Move the pointers for `l1`, `l2`, and the result list to the next nodes.
5. **Handle the final carry** after both lists are processed by creating a new node if carry is non-zero.
6. Return the next node of the dummy node as the result list (since the dummy node is a placeholder).

```
/**
 * 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 dummy = new ListNode(-1);  // Dummy node to simplify result list
handling
        ListNode current = dummy;  // Pointer to build the result list
        int carry = 0;  // Initialize carry

        // Iterate through both lists while there are remaining nodes or carry
        while (l1 != null || l2 != null || carry != 0) {
            int sum = carry;  // Start with carry

            if (l1 != null) {
                sum += l1.val;  // Add value from the first list
                l1 = l1.next;   // Move to the next node in l1
            }
            if (l2 != null) {
                sum += l2.val;  // Add value from the second list
                l2 = l2.next;   // Move to the next node in l2
            }

            carry = sum / 10;  // Update carry
            int digit = sum % 10;  // Get last digit of the sum
```

```
            current.next = new ListNode(digit);  // Add the digit to the result list
            current = current.next;  // Move the result pointer forward
        }

        return dummy.next;  // Return the next node of dummy (head of the result list)
    }
}
```

Description:

- `ListNode dummy = new ListNode(-1);`: This is a dummy node to simplify handling of the result linked list. We don't directly modify this dummy node; instead, we use it to store the head of our result list.
- `ListNode current = dummy;`: This pointer helps us build the result list node by node.
- `int carry = 0;`: This variable stores the carry for sums greater than or equal to 10.
- `while (l1 != null || l2 != null || carry != 0)`: This loop ensures we process all nodes from both lists. Even after both lists are fully processed, if there's a carry left, the loop continues to handle it.
- `sum = carry + l1.val + l2.val`: Add the values from the two lists along with the carry.
- `carry = sum / 10` **and** `digit = sum % 10`: Calculate the carry for the next iteration and the digit to store in the current node.
- `current.next = new ListNode(digit);`: Create a new node with the current sum's last digit and link it to the result list.

Dry Run:

Let's dry run the solution for:

- `l1 = [2 -> 4 -> 3]` representing `342`
- `l2 = [5 -> 6 -> 4]` representing `465`

Step-by-step:

1. **Initialization**:
   - `dummy = -1`
   - `carry = 0`
   - `l1` is pointing to `2`, `l2` is pointing to `5`.
2. **First Iteration**:
   - `sum = carry + l1.val + l2.val = 0 + 2 + 5 = 7`

- `carry = sum / 10 = 7 / 10 = 0`
- `digit = sum % 10 = 7 % 10 = 7`
- Create new node with value `7`.
- Move `l1` to `4` and `l2` to `6`.

3. **Second Iteration**:
   - `sum = carry + l1.val + l2.val = 0 + 4 + 6 = 10`
   - `carry = 10 / 10 = 1`
   - `digit = 10 % 10 = 0`
   - Create new node with value `0`.
   - Move `l1` to `3` and `l2` to `4`.
4. **Third Iteration**:
   - `sum = carry + l1.val + l2.val = 1 + 3 + 4 = 8`
   - `carry = 8 / 10 = 0`
   - `digit = 8 % 10 = 8`
   - Create new node with value `8`.
   - Both `l1` and `l2` are `null` now.
5. **End of Iteration**:
   - Since `carry` is `0`, we don't need to add any more nodes.

Final result: `[7 -> 0 -> 8]`, which represents `807`.

Time Complexity:

- **O(max(m, n))**, where `m` and `n` are the lengths of the linked lists `l1` and `l2`.

Space Complexity:

- **O(max(m, n))**, for the space needed to store the resulting linked list.

Thank for Reading

Subscribe to My Youtube Channel [Azan's Coding Corner]

Follow Me on Linkedin Azan Imtiaz