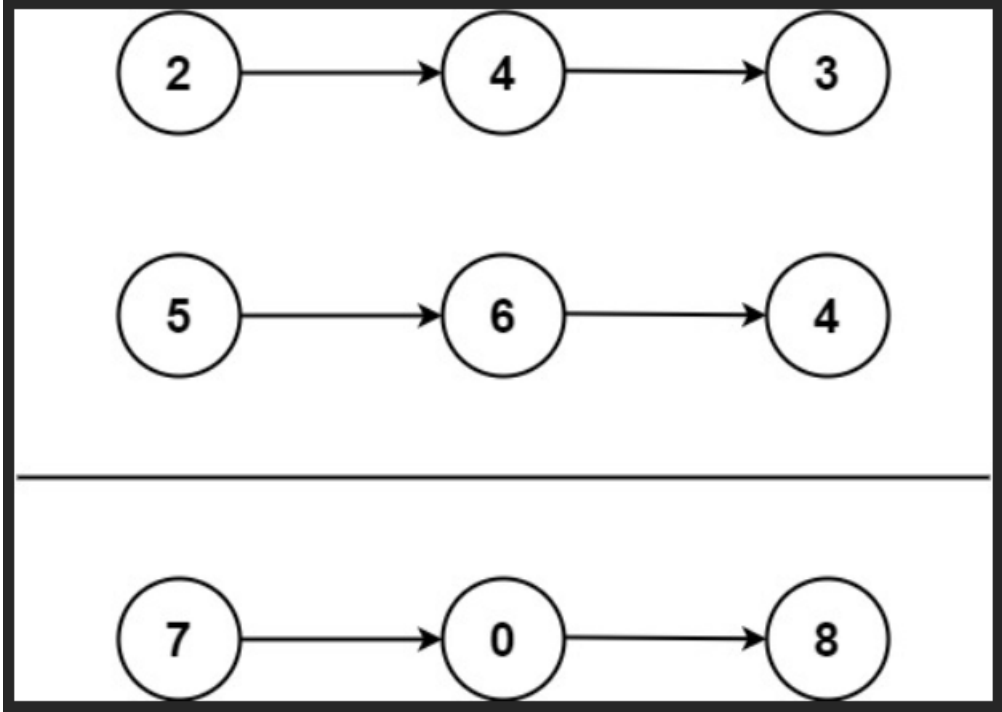
Leetcode problem number 2

Statement:-[2. Add Two Numbers]

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.

Explanation

यह problem क्या कहना चाहता है

1. आपको **दो linked list** दी गई हैं
2. हर linked list एक **संख्या (number)** को represent करती है
3. हर node में **सिर्फ एक digit (0 से 9)** होता है
4. digits **reverse order** में store हैं

Reverse order का मतलब

- पहला node → इकाई का अंक (ones place)
- दूसरा node → दहाई का अंक (tens place)
- तीसरा node → सैकड़ा (hundreds place)

उदाहरण:

[2,4,3] का मतलब है 342

[5,6,4] का मतलब है 465

आपको करना क्या है

1. दोनों linked list से बनी **संख्याओं को जोड़ना है**
2. जोड़ का result भी **linked list** में देना है
3. result की linked list भी **reverse order** में होगी

Example 1

Input:

l1 = [0]

l2 = [0]

Explanation:

0 + 0 = 0

Output:

[0]

Example 2

Input:

l1 = [9,9,9,9,9,9,9]

l2 = [9,9,9,9]

Explanation:

1. l1 represent करता है: 9999999
2. l2 represent करता है: 9999
3. जोड़ करने पर:

9999999

\+ 9999

10009998

4. Reverse order में result:

[8,9,9,9,0,0,0,1]

जोड़ने की सही प्रक्रिया (Logic)

1. एक नई linked list बनानी होगी result के लिए
2. शुरुआत पहले node से होगी
3. हर step पर:
    - दोनों nodes के digit जोड़ो
    - पिछले step का carry जोड़ो
4. अगर sum 10 या उससे ज्यादा हो:
    - digit = sum % 10
    - carry = sum / 10
5. हर digit के लिए नया node बनाओ
6. जब दोनों list खत्म हो जाएं, तब भी अगर carry बचा हो तो उसे add करो

Important बातें

- हर node में सिर्फ 1 digit होगा
- leading zero नहीं होगा
- maximum 100 nodes हो सकते हैं
- carry को ignore नहीं करना है

आसान शब्दों में

यह problem बस यह check करता है कि
क्या आप **linked list का use करके normal addition** सही तरीके से कर सकते हो या नहीं।

Example (Dry Run)

Input:

l1 = [2, 4, 3]

l2 = [5, 6, 4]

मतलब:

l1 → 342

l2 → 465

हमें जोड़ना है:

342 + 465 = 807

Output होना चाहिए (reverse order):

[7, 0, 8]

Initial Setup

- carry = 0
- result linked list = खाली
- pointer = result का start

Step 1 (पहला node)

l1.val = 2

l2.val = 5

carry = 0

Calculation:

sum = 2 + 5 + 0 = 7

digit = 7 % 10 = 7

carry = 7 / 10 = 0

Result list:

[7]

Move pointers:

l1 → 4

l2 → 6

Step 2 (दूसरा node)

l1.val = 4

l2.val = 6

carry = 0

Calculation:

sum = 4 + 6 + 0 = 10

digit = 10 % 10 = 0

carry = 10 / 10 = 1

Result list:

[7 → 0]

Move pointers:

l1 → 3

l2 → 4

Step 3 (तीसरा node)

l1.val = 3

l2.val = 4

carry = 1

Calculation:

sum = 3 + 4 + 1 = 8

digit = 8 % 10 = 8

carry = 8 / 10 = 0

Result list:

[7 → 0 → 8]

Move pointers:

l1 → null

l2 → null

Loop खत्म

- l1 = null
- l2 = null
- carry = 0

अब कोई extra node नहीं बनेगा।

Final Output

[7, 0, 8]

जो represent करता है:

807

Short Summary

1. एक-एक digit जोड़ते गए
2. carry को साथ लेकर चले
3. हर step पर नया node बनाया
4. reverse order में result मिला

code

java

```
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode();
        ListNode res = dummy;
        int total = 0, carry = 0;
```

```
        while (l1 != null || l2 != null || carry != 0) {
            total = carry;

            if (l1 != null) {
                total += l1.val;
                l1 = l1.next;
            }
            if (l2 != null) {
                total += l2.val;
                l2 = l2.next;
            }

            int num = total % 10;
            carry = total / 10;
            dummy.next = new ListNode(num);
            dummy = dummy.next;
        }

        return res.next;
    }
}
```