# 题目

给定一个用链表表示的非负整数， 然后将这个整数 再加上 1 。

这些数字的存储是这样的：最高位有效的数字位于链表的首位 head 。

示例 1:

输入: head = [1,2,3]

输出: [1,2,4]

示例 2:

输入: head = [0]

输出: [1]

提示：

链表中的节点数在 [1, 100] 的范围内。

0 <= Node.val <= 9

由链表表示的数字不包含前导零，除了零本身。

# 分析

## 方法一：反转链表

要解决这个问题，我们需要将一个用链表表示的非负整数加1。由于链表的首位是最高位，直接从头部开始处理进位会比较困难，因此可以通过反转链表来简化操作，使得我们能够从最低位（原链表的尾部）开始处理加1和进位逻辑。

思路分析

1、反转链表：将原链表反转，使最低位（原链表的尾部）变为新链表的头部，便于从低位到高位处理加1和进位。

2、处理加1和进位：从反转后的链表头部开始，先将第一个节点（原最低位）加1，然后依次处理可能产生的进位（如果当前节点值为10，则设置为0并向前进位）。

3、再次反转链表：将处理完进位的链表再次反转，恢复原有的高位在前的顺序。

4、特殊情况处理：如果加1后产生了新的最高位（如原链表为[9,9]，加1后变为[1,0,0]），需要在反转前添加新的节点。

代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

class Solution {

public:

ListNode\* plusOne(ListNode\* head) {

// 步骤1：反转链表，使最低位在头部

ListNode\* reversedHead = reverseList(head);

// 步骤2：处理加1和进位

ListNode\* curr = reversedHead;

int carry = 1; // 初始进位为1（因为要加1）

while (curr != nullptr && carry > 0) {

int sum = curr->val + carry;

curr->val = sum % 10; // 当前位的值

carry = sum / 10; // 新的进位

// 如果还有进位且是最后一个节点，需要新增节点

if (carry > 0 && curr->next == nullptr) {

curr->next = new ListNode(carry);

carry = 0; // 进位已处理，重置

}

curr = curr->next;

}

// 步骤3：再次反转链表，恢复原顺序

return reverseList(reversedHead);

}

private:

// 辅助函数：反转链表

ListNode\* reverseList(ListNode\* head) {

ListNode\* prev = nullptr;

ListNode\* curr = head;

while (curr != nullptr) {

ListNode\* nextTemp = curr->next; // 保存下一个节点

curr->next = prev; // 反转当前节点的指针

prev = curr; // 移动prev到当前节点

curr = nextTemp; // 移动curr到下一个节点

}

return prev; // 反转后prev为新的头节点

}

};

代码解释：

1、反转链表：reverseList函数通过迭代方式反转链表，将原链表的尾部变为头部（即原数字的最低位变为新链表的首位），方便从低位开始处理加1操作。

2、处理加1和进位：

- 初始进位carry设为1（因为要给整个数字加1）。

- 遍历反转后的链表，将当前节点值与进位相加，更新当前节点值为sum % 10，进位更新为sum / 10。

- 如果遍历到链表末尾仍有进位（如[9,9]加1后需要新增高位1），则创建新节点存储进位值。

3、恢复链表顺序：将处理完进位的链表再次反转，使最高位回到链表头部，得到最终结果。

复杂度分析

时间复杂度：O(n)，其中n是链表的长度。反转链表需要O(n)时间，处理进位需要O(n)时间，整体为线性时间。

空间复杂度：O(1)，仅使用常数个额外指针，没有使用额外的线性空间（反转操作是原地进行的）。

这种方法通过反转链表将“从高位到低位”的处理转化为“从低位到高位”，简化了进位逻辑，是处理此类问题的高效方式。