链表+数学：两数相加

题目来源：Leetcode 2：https://leetcode-cn.com/problems/add-two-numbers/

暴力解法：转数字求和再转链表

Java代码

链表实现代码：

```
class ListNode {
    int val; // 数据域
    ListNode next; // 指针域，指向下一个节点

    ListNode() {
    }

    ListNode(int x) {
        val = x;
    }
}
```

存在 `long` 类型溢出的问题，需要手动测试。

```
/**
 * 暴力解法：
 *  遍历两个链表使用数学思维分别将他们转成整数
 *  对两个整数进行求和得到sum
 *  将sum按照数学思维再转成链表
 * 手动测试：
 *  若超过语言支持的数据类型范围，则报错
 *  解决办法：BigInteger
 *
 *
 * @param l1
 * @param l2
 * @return
 */
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
    //把链表转成数字，注意次序为逆序
    long l1Value = 0;
    int digit = 0;
    while (l1 != null) {
```

```
            //该位对应的 单位
            int pow = (int) Math.pow(10, digit);
            //在当前数值基础上增加新的一个高位
            l1Value += (long)l1.val * pow;
            digit++;
            //链表指向下一个节点
            l1 = l1.next;
        }
        long l2Value = 0;
        digit = 0;
        while (l2 != null) {
            //该位对应的 单位
            int pow = (int) Math.pow(10, digit);
            //在当前数值基础上增加新的一个高位
            l2Value += (long)l2.val * pow;
            digit++;
            //链表指向下一个节点
            l2 = l2.next;
        }
        //创建一个新链表，头部为空节点
        ListNode head = new ListNode();
        ListNode cur = head;
        //数字相加
        long sum = l1Value + l2Value;
        if (sum == 0) {
            head = new ListNode(0);
            return head;
        }
        //数字再转成链表
        while (sum > 0) {
            //每次取当前最低位
            int val = (int) (sum % 10);
            //移除最低位
            sum = sum / 10;
            //创建新节点
            ListNode node = new ListNode(val);
            //插入链表尾部
            cur.next = node;
            //链表尾部指针移动
            cur = cur.next;
        }
        return head.next;
    }
```

解决办法：用 `java.math.BigInteger` 替代 `long` 类型。注意全类名写法。

```
/**
 * 暴力解法：存在的问题：使用long也会存在溢出
```

```
 *  解决办法：java.math.BigInteger
 *  该类在leetcode默认环境没有导入，所以使用全类名编写
 *
 * @param l1
 * @param l2
 * @return
 */
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
    //把链表转成数字，注意次序为逆序
    java.math.BigInteger l1Value = java.math.BigInteger.valueOf(0);
    int digit = 0;
    while (l1 != null) {
        java.math.BigInteger carry =
java.math.BigInteger.valueOf(10).pow(digit);
        l1Value =
l1Value.add(carry.multiply(java.math.BigInteger.valueOf(l1.val)));
        digit++;
        l1 = l1.next;
    }
    java.math.BigInteger l2Value = java.math.BigInteger.valueOf(0);
    digit = 0;
    while (l2 != null) {
        java.math.BigInteger carry =
java.math.BigInteger.valueOf(10).pow(digit);
        l2Value =
l2Value.add(carry.multiply(java.math.BigInteger.valueOf(l2.val)));
        digit++;
        l2 = l2.next;
    }
    ListNode head = new ListNode();
    ListNode cur = head;
    //数字相加，然后再转成链表
    java.math.BigInteger sum = l1Value.add(l2Value);
    if (sum.compareTo(java.math.BigInteger.valueOf(0)) == 0) {
        head = new ListNode(0);
        return head;
    }
    while (sum.compareTo(java.math.BigInteger.valueOf(0)) > 0) {
        int val = sum.mod(java.math.BigInteger.valueOf(10)).intValue();
        sum = sum.divide(java.math.BigInteger.valueOf(10));
        ListNode node = new ListNode((int) val);
        cur.next = node;
        cur = cur.next;
    }
    return head.next;
}
```

最优解：数学思维解法

java代码

```
    /**
     * 最优解：数学思维解法
     * 1.遍历两个链表
     * 2.对应位置的节点数值相加
     * 3.将计算结果插入新链表尾部
     *  大于10，则进位，将进位加到下个节点
     *
     * 边界问题
     *  两个链表边界：next==null
     * 细节问题
     *  两个链表长度不一致，短链表高位视为0
     *  链表最高位发生进位，结果链表需要增加一个节点存放进位数字
     *
     * @param l1
     * @param l2
     * @return
     */
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode p = l1, q = l2; // 原链表的两个遍历指针
        ListNode resultHead = new ListNode(-1); // 结果链表的头结点head
        ListNode curr = resultHead; // 结果链表的遍历指针，代表当前操作的节点

        int carry = 0; // 进位
        // 1.遍历两个链表
        while (p != null || q != null) { // 以长链表为准
            // 获取当前节点的值：链表较短，已无节点，取0
            int x = p != null ? p.val : 0;
            int y = q != null ? q.val : 0;
            // 2.对应位置的节点数值相加
            int sum = x + y + carry;
            carry = sum / 10; // 如何得到进位：和对10求整，得到此次计算的进位
            int num = sum % 10; // 存放到新链表节点中的数值

            // 3.将计算结果插入新链表尾部
            curr.next = new ListNode(num); // 创建新节点
            curr = curr.next;

            p = p == null ? p : p.next;
            q = q == null ? q : q.next;
        }
        if (carry > 0) { // 处理进位节点
            curr.next = new ListNode(carry);
        }
        return resultHead.next;
```

```
    }
```

C++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) {}
 * };
 */
struct ListNode {
     int val;
     ListNode *next;
     ListNode() : val(0), next(NULL) {}
     ListNode(int x) : val(x), next(NULL) {}
     ListNode(int x, ListNode *next) : val(x), next(next) {}
};

class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode* p = l1;
        ListNode* q = l2; // 原链表的两个遍历指针
        ListNode* resultHead = new ListNode(-1); // 结果链表的头结点head
        ListNode* curr = resultHead; // 结果链表的遍历指针，代表当前操作的节点

        int carry = 0; // 进位
        // 1.遍历两个链表
        while (p != NULL || q != NULL) { // 以长链表为准
            // 获取当前节点的值：链表较短，已无节点，取0
            int x = p != NULL ? p->val : 0;
            int y = q != NULL ? q->val : 0;
            // 2.对应位置的节点数值相加
            int sum = x + y + carry;
            carry = sum / 10; // 如何得到进位：和对10求整，得到此次计算的进位
            int num = sum % 10; // 存放到新链表节点中的数值

            // 3.将计算结果插入新链表尾部
            curr->next = new ListNode(num); // 创建新节点
            curr = curr->next;

            p = p == NULL ? p : p->next;
```

```
            q = q == NULL ? q : q->next;
        }
        if (carry > 0) { // 处理进位节点
            curr->next = new ListNode(carry);
        }
        return resultHead->next;
    }
};
```

Python代码

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
class Solution:
    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:
        p = l1;
        q = l2; # 原链表的两个遍历指针
        resultHead = ListNode(-1); # 结果链表的头结点head
        curr = resultHead; # 结果链表的遍历指针，代表当前操作的节点

        carry = 0; # 进位
        # 1.遍历两个链表
        while (p != None or q != None) : # 以长链表为准
            # 获取当前节点的值：链表较短，已无节点，取0
            x = p.val if p != None else 0;
            y = q.val if q != None else 0;
            # 2.对应位置的节点数值相加
            sum = x + y + carry;
            carry = (int)(sum / 10);  # 如何得到进位：和对10求整，得到此次计算的进位
            num = (int)(sum % 10);  # 存放到新链表节点中的数值

            # 3.将计算结果插入新链表尾部
            curr.next = ListNode(num); # 创建新节点
            curr = curr.next;

            p = p if p == None else p.next;
            q = q if q == None else q.next;
```

```
        if (carry > 0) : # 处理进位节点
            curr.next = ListNode(carry);

        return resultHead.next;
```

测试用例

辅助数据结构：链表。代码如下：

```
  class ListNode {
        int val; // 数据域
        ListNode next; // 指针域，指向下一个节点

        ListNode() {
        }

        ListNode(int x) {
            val = x;
        }
    }
```

测试用例：

```
输入：(2 -> 4 -> 3) + (5 -> 6 -> 4)
输出：7 -> 0 -> 8
原因：342 + 465 = 807
```

测试代码：

```
    public static void main(String[] args) {
        Solution solution=new AddTwoNumbers2().new Solution();

        int[] arr1 = {2, 4, 3};
        int[] arr2 = {5, 6, 4};
        ListNode l1 = new AddTwoNumbers2().new ListNode();
        ListNode l2 = new AddTwoNumbers2().new ListNode();

        ListNode l1Cur = l1;
        ListNode l2Cur = l2;
        for (int i = 0; i < arr1.length; i++) {
            ListNode node1 = new AddTwoNumbers2().new ListNode(arr1[i]);
            ListNode node2 = new AddTwoNumbers2().new ListNode(arr2[i]);
            l1Cur.next = node1;
            l1Cur = node1;
            l2Cur.next = node2;
            l2Cur = node2;
```

```
        }

        ListNode result = solution.addTwoNumbers(l1.next, l2.next);
        while (result != null) {
            System.out.print(result.val + "  "); // 输出：7 0 8
            result = result.next;
        }
    }
```