- 上传 2个 蓝桥杯、ACM等相关的题目及其解答过程，题目自选。
  - 要求
    - 1.有题目，有示例
    - 2.有题目分析过程
    - 3.有代码和结果
- 1. 两数之和（简单）
  - 朴素解法
- 2. 两数相加（中等）
  - 朴素解法（哨兵技巧）
- 3. 无重复字符的最长子串（中等）
  - 双指针 + 哈希表

1 上传 2个 蓝桥杯、ACM等相关的题目及其解答过程，题目自选。

1-1 要求

I 1.有题目，有示例

如：

题目 ：给定一个字符串 s ，请你找出其中不含有重复字符的最长子串的长度。

示例：

输入：s="abcabcbb"

输出：3

II 2.有题目分析过程

III 3.有代码和结果

2 1. 两数之和（简单）

这是 LeetCode 上的 **1. 两数之和** ，难度为 **简单** 。

Tag : 「哈希表」、「模拟」

给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出「和为目标值」的那「两个」整数，并返回它们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

你可以按任意顺序返回答案。

示例 1：

```
输入：nums = [2,7,11,15], target = 9

输出：[0,1]

解释：因为 nums[0] + nums[1] == 9 ，返回 [0, 1] 。
```

示例 2：

```
输入：nums = [3,2,4], target = 6

输出：[1,2]
```

示例 3：

```
输入：nums = [3,3], target = 6

输出：[0,1]
```

提示：

- $2 <= nums.length <= 10^3$
- $-10^9 <= nums[i] <= 10^9$
- $-10^9 <= target <= 10^9$
- 只会存在一个有效答案

2-1 朴素解法

由于我们每次要从数组中找两个数。

因此一个很简单的思路是：使用两重循环枚举下标 i 和 j ，分别代表要找的两个数。

然后判断 $nums[i] + nums[j] = target$ 是否成立。

另外为了防止得到重复的解，我们需要在第一层循环中让 i 从 0 开始，到 $n - 2$ 结束（确保能取到下一位数作为 j ）；在第二层循环中让 j 从 $i + 1$ 开始，到 $n - 1$ 结束。

代码：

```java
class Solution {
    public int[] twoSum(int[] nums, int t) {
        int n = nums.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                if (t == nums[i] + nums[j]) return new int[]{i,j};
            }
        }
        return new int[]{};
    }
}
```

* 时间复杂度：两重循环，以复杂度是 $O(n^2)$

* 空间复杂度：$O(1)$

3 2. 两数相加（中等）

这是 LeetCode 上的 **2. 两数相加** ，难度为 **中等** 。

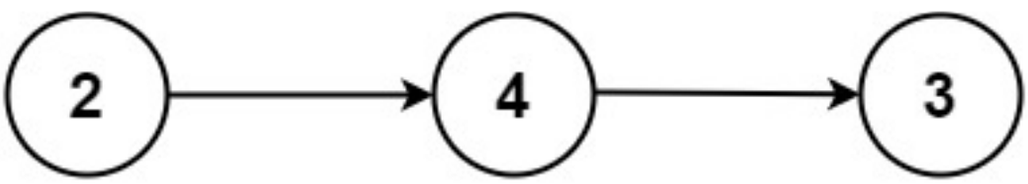
Tag : 「递归」、「链表」、「数学」、「模拟」

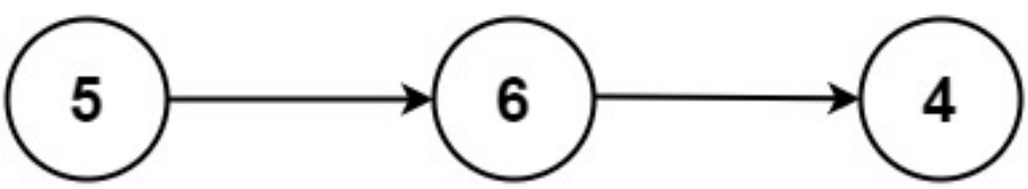
给你两个 非空 的链表，表示两个非负的整数。它们每位数字都是按照 逆序 的方式存储的，并且每个节点只能存储 一位 数字。

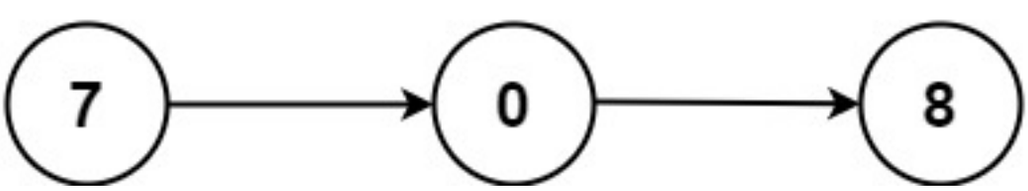
请你将两个数相加，并以相同形式返回一个表示和的链表。

你可以假设除了数字 0 之外，这两个数都不会以 0 开头。

示例 1：








```
输入: l1 = [2,4,3], l2 = [5,6,4]

输出: [7,0,8]

解释: 342 + 465 = 807.
```

示例 2：

```
输入：l1 = [0], l2 = [0]

输出：[0]
```

示例 3：

```
输入：l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]

输出：[8,9,9,9,0,0,0,1]
```

提示：

- 每个链表中的节点数在范围 $[1, 100]$ 内
- $0 <= Node.val <= 9$
- 题目数据保证列表表示的数字不含前导零

3-1 朴素解法（哨兵技巧）

这是道模拟题，模拟人工竖式做加法的过程：

从最低位至最高位，逐位相加，如果和大于等于 10，则保留个位数字，同时向前一位进 1 如果最高位有进位，则需在最前面补 1。

做有关链表的题目，有个常用技巧：添加一个虚拟头结点（哨兵），帮助简化边界情况的判断。

代码：

```java
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(0);
        ListNode tmp = dummy;
        int t = 0;
        while (l1 != null || l2 != null) {
            int a = l1 != null ? l1.val : 0;
            int b = l2 != null ? l2.val : 0;
            t = a + b + t;
            tmp.next = new ListNode(t % 10);
            t /= 10;
            tmp = tmp.next;
            if (l1 != null) l1 = l1.next;
            if (l2 != null) l2 = l2.next;
        }
        if (t > 0) tmp.next = new ListNode(t);
        return dummy.next;
    }
}
```

- 时间复杂度：m 和 n 分别代表两条链表的长度，则遍历到的最远位置为 $max(m, n)$，复杂度为 $O(max(m, n))$
- 空间复杂度：创建了 $max(m, n) + 1$ 个节点（含哨兵），复杂度为 $O(max(m, n))$（忽略常数）

注意：事实上还有可能创建 $max(m+n)+2$ 个节点，包含哨兵和最后一位的进位。但复杂度仍为 $O(max(m,n))$。

4 3. 无重复字符的最长子串（中等）

这是 LeetCode 上的 **3. 无重复字符的最长子串** ，难度为 **中等** 。

Tag : 「哈希表」、「双指针」、「滑动窗口」

给定一个字符串，请你找出其中不含有重复字符的「最长子串」的长度。

示例 1:

```
输入: s = "abcabcbb"

输出: 3

解释: 因为无重复字符的最长子串是 "abc", 所以其长度为 3。
```

示例 2:

```
输入: s = "bbbbb"

输出: 1

解释: 因为无重复字符的最长子串是 "b", 所以其长度为 1。
```

示例 3:

```
输入: s = "pwwkew"

输出: 3

解释: 因为无重复字符的最长子串是 "wke", 所以其长度为 3。
     请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。
```

示例 4:

```
输入: s = ""

输出: 0
```

提示：

- $0 <= s.length <= 5 \times 10^4$
- `s` 由英文字母、数字、符号和空格组成

4-1 双指针 + 哈希表

定义两个指针 `start` 和 `end`，表示当前处理到的子串是 `[start,end]`。

`[start,end]` 始终满足要求：无重复字符。

从前往后进行扫描，同时维护一个哈希表记录 `[start,end]` 中每个字符出现的次数。

遍历过程中，`end` 不断自增，将第 `end` 个字符在哈希表中出现的次数加一。

令 `right` 为 下标 `end` 对应的字符，当满足 `map.get(right) > 1` 时，代表此前出现过第 `end` 位对应的字符。

此时更新 `start` 的位置（使其右移），直到不满足 `map.get(right) > 1`（代表 `[start,end]` 恢复满足无重复字符的条件）。同时使用 `[start,end]` 长度更新答案。

代码：

```java
class Solution {
    public int lengthOfLongestSubstring(String s) {
        Map<Character, Integer> map = new HashMap<>();
        int ans = 0;
        for (int start = 0, end = 0; end < s.length(); end++) {
            char right = s.charAt(end);
            map.put(right, map.getOrDefault(right, 0) + 1);
            while (map.get(right) > 1) {
                char left = s.charAt(start);
                map.put(left, map.get(left) - 1);
                start++;
            }
            ans = Math.max(ans, end - start + 1);
        }
        return ans;
    }
}
```

- 时间复杂度：虽然有两层循环，但每个字符在哈希表中最多只会被插入和删除一次，复杂度为 $O(n)$
- 空间复杂度：使用了哈希表进行字符记录，复杂度为 $O(n)$