Leetcode Top100题目和答案（Java完整版 面试必备）

2020年3月7日 来源: 网络转载

二刷完剑指Offer后又刷了一遍Leetcode Top 100专栏的题目，听说基本上能涵盖面试的算法题，总体来说收获还是很大的，下面贴出答案，又不懂的可以给我留言，博主会及时解答。
我的github (https://github.com/hanggegreat/Java-Note/blob/master/%E7%AE%97%E6%B3%95/leetcode-TOP100.md)
准备把春招复习的知识都整理到github上，一边是自己做个总结，一边也能供大家参考
——leetcode数据库 19道题 (https://github.com/hanggegreat/Java-Note/blob/master/%E6%95%B0%E6%8D%AE%E5%BA%93/leetcode.md)
——剑指Offer 66道题 (https://github.com/hanggegreat/Java-Note/blob/master/%E7%AE%97%E6%B3%95/%E5%89%91%E6%8C%87Offer.md)

以下摘自leetcode Top100精选题目

文章目录

- 53.最大子序和 (https://blog.csdn.net/weixin_38896998/article/details/88810177#53_1372)
- 55.跳跃游戏 (https://blog.csdn.net/weixin_38896998/article/details/88810177#55_1406)
- 56.合并区间 (https://blog.csdn.net/weixin_38896998/article/details/88810177#56_1452)
- 62.不同路径 (https://blog.csdn.net/weixin_38896998/article/details/88810177#62_1501)
- 64.最小路径和 (https://blog.csdn.net/weixin_38896998/article/details/88810177#64_1558)
- 70.爬楼梯 (https://blog.csdn.net/weixin_38896998/article/details/88810177#70_1609)
- 72.编辑距离 (https://blog.csdn.net/weixin_38896998/article/details/88810177#72_1658)
- 75.颜色分类 (https://blog.csdn.net/weixin_38896998/article/details/88810177#75_1724)
- 76.最小覆盖子串 (https://blog.csdn.net/weixin_38896998/article/details/88810177#76_1789)
- 78.子集 (https://blog.csdn.net/weixin_38896998/article/details/88810177#78_1843)
- 79.单词搜索 (https://blog.csdn.net/weixin_38896998/article/details/88810177#79_1897)
- 84.柱状图中最大的矩形 (https://blog.csdn.net/weixin_38896998/article/details/88810177#84_1968)
- 85.最大矩形 (https://blog.csdn.net/weixin_38896998/article/details/88810177#85_2025)
- 94.二叉树的中序遍历 (https://blog.csdn.net/weixin_38896998/article/details/88810177#94_2076)
- 96.不同的二叉搜索树 (https://blog.csdn.net/weixin_38896998/article/details/88810177#96_2118)
- 98.验证二叉搜索树 (https://blog.csdn.net/weixin_38896998/article/details/88810177#98_2157)
- 101.对称二叉树 (https://blog.csdn.net/weixin_38896998/article/details/88810177#101_2220)

- 102.二叉树的层序遍历 (https://blog.csdn.net/weixin_38896998/article/details/88810177#102_2270)
- 104.二叉树的最大深度 (https://blog.csdn.net/weixin_38896998/article/details/88810177#104_2331)
- 105.从前序遍历和中序遍历序列构造二叉树 (https://blog.csdn.net/weixin_38896998/article/details/88810177#105_2368)
- 114.二叉树展开为链表 (https://blog.csdn.net/weixin_38896998/article/details/88810177#114_2424)
- 121.买卖股票的最佳时机 (https://blog.csdn.net/weixin_38896998/article/details/88810177#121_2479)
- 124.二叉树中的最大路径和 (https://blog.csdn.net/weixin_38896998/article/details/88810177#124_2527)
- 128.最长连续序列 (https://blog.csdn.net/weixin_38896998/article/details/88810177#128_2586)
- 136.只出现一次的数字 (https://blog.csdn.net/weixin_38896998/article/details/88810177#136_2632)
- 139.单词拆分 (https://blog.csdn.net/weixin_38896998/article/details/88810177#139_2672)
- 141.环形链表 (https://blog.csdn.net/weixin_38896998/article/details/88810177#141_2735)
- 142.环形链表Ⅱ (https://blog.csdn.net/weixin_38896998/article/details/88810177#142_2801)
- 146.LRU缓存机制 (https://blog.csdn.net/weixin_38896998/article/details/88810177#146LRU_2875)
- 148.排序链表 (https://blog.csdn.net/weixin_38896998/article/details/88810177#148_2931)
- 152.乘机最大子序列 (https://blog.csdn.net/weixin_38896998/article/details/88810177#152_2995)
- 155.最小栈 (https://blog.csdn.net/weixin_38896998/article/details/88810177#155_3043)
- 160.相交链表 (https://blog.csdn.net/weixin_38896998/article/details/88810177#160_3102)
- 169.求众数 (https://blog.csdn.net/weixin_38896998/article/details/88810177#169_3183)

- 198.打家劫舍 (https://blog.csdn.net/weixin_38896998/article/details/88810177#198_3233)
- 200.岛屿的个数 (https://blog.csdn.net/weixin_38896998/article/details/88810177#200_3283)
- 206.反转链表 (https://blog.csdn.net/weixin_38896998/article/details/88810177#206_3353)
- 207.课程表 (https://blog.csdn.net/weixin_38896998/article/details/88810177#207_3394)
- 208.实现Trie(前缀树) (https://blog.csdn.net/weixin_38896998/article/details/88810177#208Trie_3481)
- 215.数组中的第K个最大元素 (https://blog.csdn.net/weixin_38896998/article/details/88810177#215K_3562)
- 221.最大正方形 (https://blog.csdn.net/weixin_38896998/article/details/88810177#221_3605)
- 226.翻转二叉树 (https://blog.csdn.net/weixin_38896998/article/details/88810177#226_3656)
- 234.回文链表 (https://blog.csdn.net/weixin_38896998/article/details/88810177#234_3708)
- 236.二叉树的最近公共祖先 (https://blog.csdn.net/weixin_38896998/article/details/88810177#236_3772)
- 238.除自身以外数组的乘积 (https://blog.csdn.net/weixin_38896998/article/details/88810177#238_3838)
- 239.滑动窗口的最大值 (https://blog.csdn.net/weixin_38896998/article/details/88810177#239_3878)
- 240.搜索二维矩阵Ⅱ (https://blog.csdn.net/weixin_38896998/article/details/88810177#240_3941)
- 279.完全平方数 (https://blog.csdn.net/weixin_38896998/article/details/88810177#279_3996)
- 283.移动零 (https://blog.csdn.net/weixin_38896998/article/details/88810177#283_4041)
- 287.寻找重复数 (https://blog.csdn.net/weixin_38896998/article/details/88810177#287_4081)
- 297.二叉树的序列化和反序列化 (https://blog.csdn.net/weixin_38896998/article/details/88810177#297_4136)

- 300.最长上升子序列 (https://blog.csdn.net/weixin_38896998/article/details/88810177#300_4201)
- 309.最佳买卖股票时期含冷冻期 (https://blog.csdn.net/weixin_38896998/article/details/88810177#309_4247)
- 312.戳气球 (https://blog.csdn.net/weixin_38896998/article/details/88810177#312_4289)
- 322.零钱兑换 (https://blog.csdn.net/weixin_38896998/article/details/88810177#322_4338)
- 337.打家劫舍III (https://blog.csdn.net/weixin_38896998/article/details/88810177#337_4388)
- 338.比特位计数 (https://blog.csdn.net/weixin_38896998/article/details/88810177#338_4450)
- 347.前K个高频元素 (https://blog.csdn.net/weixin_38896998/article/details/88810177#347K_4487)
- 394.字符串解码 (https://blog.csdn.net/weixin_38896998/article/details/88810177#394_4539)
- 406.根据身高重建队列 (https://blog.csdn.net/weixin_38896998/article/details/88810177#406_4591)
- 416.分割等和子集 (https://blog.csdn.net/weixin_38896998/article/details/88810177#416_4629)
- 437.路径总和III (https://blog.csdn.net/weixin_38896998/article/details/88810177#437_4693)
- 438.找到字符串中所有字母异位词 (https://blog.csdn.net/weixin_38896998/article/details/88810177#438_4751)
- 448.找到所有数组中消失的数字 (https://blog.csdn.net/weixin_38896998/article/details/88810177#448_4828)
- 461.汉明距离 (https://blog.csdn.net/weixin_38896998/article/details/88810177#461_4872)
- 494.目标和 (https://blog.csdn.net/weixin_38896998/article/details/88810177#494_4914)
- 538.把二叉搜索树转换为累加树 (https://blog.csdn.net/weixin_38896998/article/details/88810177#538_4969)
- 543.二叉树的直径 (https://blog.csdn.net/weixin_38896998/article/details/88810177#543_5020)

- 560.和为K的子数组 (https://blog.csdn.net/weixin_38896998/article/details/88810177#560K_5066)
- 572.另一个树的子树 (https://blog.csdn.net/weixin_38896998/article/details/88810177#572_5111)
- 581.最短无序连续子数组 (https://blog.csdn.net/weixin_38896998/article/details/88810177#581_5193)
- 617.合并二叉树 (https://blog.csdn.net/weixin_38896998/article/details/88810177#617_5248)
- 647.回文子串 (https://blog.csdn.net/weixin_38896998/article/details/88810177#647_5292)
- 771.宝石与石头 (https://blog.csdn.net/weixin_38896998/article/details/88810177#771_5346)

1.两数之和

题目描述：

给定一个整数数组 `nums` 和一个目标值 `target`，请你在该数组中找出和为目标值的那 **两个** 整数，并返回他们的数组下标。

你可以假设每种输入只会对应一个答案。但是，你不能重复利用这个数组中同样的元素。

示例:

```
给定 nums = [2, 7, 11, 15], target = 9

因为 nums[0] + nums[1] = 2 + 7 = 9
所以返回 [0, 1]
```

Solution:

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            if (map.containsKey(nums[i])) {
                return new int[]{map.get(nums[i]), i};
            }
            map.put(target - nums[i], i);
        }
        return null;
    }
}
```

2.两数相加

题目描述：

给出两个 **非空** 的链表用来表示两个非负的整数。其中，它们各自的位数是按照 **逆序** 的方式存储的，并且它们的每个节点只能存储 **一位** 数字。

如果，我们将这两个数相加起来，则会返回一个新的链表来表示它们的和。

您可以假设除了数字 0 之外，这两个数都不会以 0 开头。

示例：

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

节点结构：

```
public class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; }
}
```

Solution:

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

        int sum = 0; // 结果
        int more = 0; // 进位
        ListNode pre = dummy;
        while (l1 != null || l2 != null || more > 0) {
            sum = (l1 == null ? 0 : l1.val) + (l2 == null ? 0 : l2.val) + more;
            more = sum / 10;
            sum %= 10;
            ListNode node = new ListNode(sum);
            pre.next = node;
            pre = node;
            l1 = l1 == null ? null : l1.next;
            l2 = l2 == null ? null : l2.next;
        }
        return dummy.next;
    }
}
```

3.无重复字符的最长子串

题目描述：

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: "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: "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: "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。
```

Solution:

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        if (s == null || s.length() < 1) {
            return 0;
        }

        int[] map = new int[256];
        int l = 0;
        int r = 0; // 滑动窗口为[l, r)，其间为不重复的元素
        int res = 0;
        while (l < s.length()) {
            if (r < s.length() && map[s.charAt(r)] == 0) {
                map[s.charAt(r++)]++;
                res = Math.max(res, r - l);
            } else {
                map[s.charAt(l++)]--;
            }
        }
        return res;
    }
}
```

4.寻找两个有序数组的中位数

题目描述：

给定两个大小为 m 和 n 的有序数组 `nums1` 和 `nums2` 。

请你找出这两个有序数组的中位数，并且要求算法的时间复杂度为 O(log(m + n))。

你可以假设 `nums1` 和 `nums2` 不会同时为空。

示例 1:

```
nums1 = [1, 3]
nums2 = [2]

则中位数是 2.0
```

示例 2:

```
nums1 = [1, 2]
nums2 = [3, 4]

则中位数是 (2 + 3)/2 = 2.5
```

Solution:

```
public class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        // 保证nums1不是最长的，时间复杂度可转化为O(log(Min(m, n)))
        if (nums1.length > nums2.length) {
            return findMedianSortedArrays(nums2, nums1);
        }

        int left = 0;
        int right = nums1.length;
        int halfLen = (nums1.length + nums2.length + 1) >> 1;

        while (left <= right) {
            int i = (left + right) >> 1; // nums1[i, nums1.length)为要分割的右半部分
            int j = halfLen - i; // nums2[j, nums2.length)为要分割的右半部分
            if (i < right && nums2[j - 1] > nums1[i]) { // nums1分割点此时需要右移
                left++;
            } else if (i > left && nums1[i - 1] > nums2[j]) { // nums1 分割点此时需要左移
                right--;
            } else {
                int leftMax = (i == 0) ? nums2[j - 1] :
                        (j == 0 ? nums1[i - 1] : Math.max(nums1[i - 1], nums2[j - 1]));
                if (((nums1.length + nums2.length) & 1) == 1) {
                    return leftMax * 1.0;
                }
                int rightMin = (i == nums1.length) ? nums2[j] :
                        (j == nums2.length ? nums1[i] : Math.min(nums1[i], nums2[j]));
                return (leftMax + rightMin) / 2.0;
            }
        }
        return 0.0;
    }
}
```

5.最长回文子串

题目描述：

给定一个字符串 `s`，找到 `s` 中最长的回文子串。你可以假设 `s` 的最大长度为 1000。

示例 1：

```
输入: "babad"
输出: "bab"
注意: "aba" 也是一个有效答案。
```

示例 2：

```
输入: "cbbd"
输出: "bb"
```

Solution：

```
/** * 中心扩展法 */
public class Solution {
    private int left;
    private int len;

    public String longestPalindrome(String s) {
        if (s == null || s.length() < 2) {
            return s;
        }

        for (int i = 0; i < s.length(); i++) {
            find(s, i, i); // 奇数长度
            find(s, i, i + 1); // 偶数长度
        }
        return s.substring(left, left + len);
    }

    private void find(String s, int left, int right) {
        while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
            if (right - left + 1 > len) {
                len = right - left + 1;
                this.left = left;
            }
            right++;
            left--;
        }
    }
}
```

10.正则表达式匹配

题目描述:

给定一个字符串（`s`）和一个字符模式（`p`）。实现支持 `'.'` 和 `'*'` 的正则表达式匹配。

```
'.' 匹配任意单个字符。
'*' 匹配零个或多个前面的元素。
```

匹配应该覆盖**整个**字符串（`s`），而不是部分字符串。

说明:

- `s` 可能为空，且只包含从 `a-z` 的小写字母。
- `p` 可能为空，且只包含从 `a-z` 的小写字母，以及字符 `.` 和 `*`。

示例 1:

```
输入:
s = "aa"
p = "a"
输出: false
解释: "a" 无法匹配 "aa" 整个字符串。
```

示例 2:

```
输入:
s = "aa"
p = "a*"
输出: true
解释: '*' 代表可匹配零个或多个前面的元素, 即可以匹配 'a' 。因此, 重复 'a' 一次, 字符串可变为 "aa"。
```

示例 3:

```
输入:
s = "ab"
p = ".*"
输出: true
解释: ".*" 表示可匹配零个或多个('*')任意字符('.')。
```

示例 4:

```
输入:
s = "aab"
p = "c*a*b"
输出: true
解释: 'c' 可以不被重复, 'a' 可以被重复一次。因此可以匹配字符串 "aab"。
```

示例 5:

```
输入:
s = "mississippi"
p = "mis*is*p*."
输出: false
```

Solution:

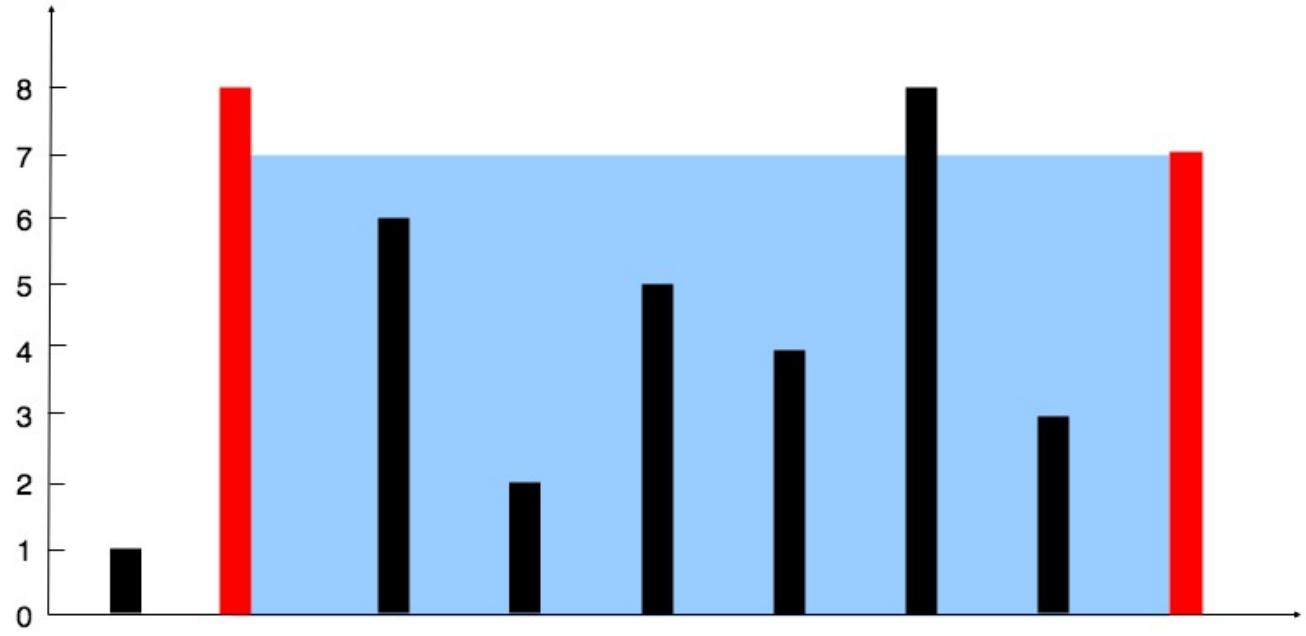
```
public class Solution {
    public boolean isMatch(String s, String p) {
        if (s == null || p == null) {
            return false;
        }

        return isMatch(s, p, 0, 0);
    }

    private boolean isMatch(String str, String pattern, int s, int p) {
        // 正则表达式已用尽，如果字符串还未匹配完，则返回false
        if (p == pattern.length()) {
            return str.length() == s;
        }
        // 正则表达式下一位为*，此时考虑两种情况
        if (p + 1 < pattern.length() && pattern.charAt(p + 1) == '*') {
            // 若正则表达式当前位字符与字符串当前位置相匹配，则匹配1位或者0位
            if (s < str.length() && (str.charAt(s) == pattern.charAt(p) || pattern.charAt(
p) == '.')) {
                return isMatch(str, pattern, s, p + 2) || isMatch(str, pattern, s + 1, p);
            }
            // 若正则表达式当前位字符与字符串当前位置不匹配，则匹配0位
            return isMatch(str, pattern, s, p + 2);
        }

        // 匹配1位
        if (s < str.length() && (str.charAt(s) == pattern.charAt(p) || pattern.charAt(p) =
= '.')) {
            return isMatch(str, pattern, s + 1, p + 1);
        }
        return false;
    }
}
```

11.盛最多水的容器

题目描述：

给定 n 个非负整数 a1，a2，...，an，每个数代表座标中的一个点 (i, ai) 。在座标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, ai) 和 (i, 0)。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

说明：你不能倾斜容器，且 n 的值至少为 2。



图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。

示例:

```
输入: [1,8,6,2,5,4,8,3,7]
输出: 49
```

Solution:

```
/** * 利用滑动窗口解决 */
public class Solution {
    public int maxArea(int[] height) {
        int res = 0;
        int left = 0;
        int right = height.length - 1;

        while (left < right) {
            res = Math.max(res, Math.min(height[left], height[right]) * (right - left));
            if (height[left] < height[right]) {
                left++;
            } else {
                right--;
            }
        }

        return res;
    }
}
```

15.三数之和

题目描述：

给定一个包含 n 个整数的数组 `nums` ，判断 `nums` 中是否存在三个元素 *a，b，c ，*使得 a + b + c = 0 ？找出所有满足条件且不重复的三元组。

注意：答案中不可以包含重复的三元组。

```
例如，给定数组 nums = [-1, 0, 1, 2, -1, -4]，

满足要求的三元组集合为：
[
  [-1, 0, 1],
  [-1, -1, 2]
]
```

Solution：

```
/** * 采用滑动窗口，时间复杂度为：O(n log(n)) */
public class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> list = new ArrayList<>();
        if (nums == null || nums.length < 3) {
            return list;
        }

        // 先排序，同时避免求重复解
        Arrays.sort(nums);

        for (int i = 0; i < nums.length - 2 && nums[i] <= 0;) {
            int l = i + 1;
            int r = nums.length - 1;
            while (l < r) {
                int sum = nums[i] + nums[l] + nums[r];
                if (sum == 0) {
                    list.add(Arrays.asList(nums[i], nums[l++], nums[r--]));
                    while (l < r && nums[l] == nums[l - 1]) {
                        l++;
                    }
                    while (r > l && nums[r] == nums[r + 1]) {
                        r--;
                    }
                } else if (sum < 0) {
                    l++;
                } else {
                    r--;
                }
            }
            i++;
            while (i < nums.length - 2 && nums[i] == nums[i - 1]) {
                i++;
            }
        }
        return list;
    }
}
```

17.电话号码的字母组合

题目描述：

给定一个仅包含数字 `2-9` 的字符串，返回所有它能表示的字母组合。

给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。

示例:

```
输入：“23”
输出：["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
```

说明:

尽管上面的答案是按字典序排列的，但是你可以任意选择答案输出的顺序。

Solution：

```
public class Solution {
    private List<String> res = new ArrayList<>();
    private String[] map = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxy
z"};

    public List<String> letterCombinations(String digits) {
        if (digits == null || digits.length() < 1) {
            return res;
        }

        dfs(digits, 0, "");
        return res;
    }

    private void dfs(String digits, int index, String str) {
        if (index == digits.length()) {
            res.add(str);
            return;
        }

        String dict = map[digits.charAt(index) - '0'];
        for (int i = 0; i < dict.length(); i++) {
            dfs(digits, index + 1, str + dict.charAt(i));
        }
    }
}
```

19.删除链表的倒数第N个节点

题目描述：

给定一个链表，删除链表的倒数第 n 个节点，并且返回链表的头结点。

示例：

```
给定一个链表: 1->2->3->4->5, 和 n = 2.

当删除了倒数第二个节点后，链表变为 1->2->3->5.
```

说明：

给定的 n 保证是有效的。

进阶：

你能尝试使用一趟扫描实现吗？

Solution：

```
/* * 双指针 */
public class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode fast = head;
        ListNode slow = dummy;
        for (int i = 0; i < n; i++) {
            fast = fast.next;
        }
        while (fast != null) {
            fast = fast.next;
            slow = slow.next;
        }
        slow.next = slow.next.next;
        return dummy.next;
    }
}
```

20有效的括号

题目描述：

给定一个只包括 `'('`，`')'`，`'{'`，`'}'`，`'['`，`']'` 的字符串，判断字符串是否有效。

有效字符串需满足：

1. 左括号必须用相同类型的右括号闭合。

2. 左括号必须以正确的顺序闭合。

注意空字符串可被认为是有效字符串。

示例 1:

```
输入: "()"
输出: true
```

示例 2:

```
输入: "()[]{}"
输出: true
```

示例 3:

```
输入: "(]"
输出: false
```

示例 4:

```
输入: "([)]"
输出: false
```

示例 5:

```
输入: "{[]}"
输出: true
```

Solution:

```
public class Solution {
    public static boolean isValid(String s) {
        if (s == null || (s.length() & 1) == 1) {
            return false;
        }

        Stack<Character> stack = new Stack<>();
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == '[' || s.charAt(i) == '{' || s.charAt(i) == '(') {
                stack.push(s.charAt(i));
            } else if (s.charAt(i) == ']' && (stack.isEmpty() || stack.pop() != '[')) {
                return false;
            } else if (s.charAt(i) == '}' && (stack.isEmpty() || stack.pop() != '{')) {
                return false;
            } else if (s.charAt(i) == ')' && (stack.isEmpty() || stack.pop() != '(')) {
                return false;
            }
        }
        return stack.isEmpty();
    }
}
```

21.合并两个有序链表

题目描述：

将两个有序链表合并为一个新的有序链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。

示例：

```
输入：1->2->4, 1->3->4
输出：1->1->2->3->4->4
```

Solution：

```
public class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(0);
        ListNode pre = dummy;

        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {
                pre.next = l1;
                l1 = l1.next;
            } else {
                pre.next = l2;
                l2 = l2.next;
            }
            pre = pre.next;
        }
        pre.next = l1 == null ? l2 : l1;

        return dummy.next;
    }
}
```

22.生成括号

题目描述：

给出 n 代表生成括号的对数，请你写出一个函数，使其能够生成所有可能的并且**有效的**括号组合。

例如，给出 n = 3，生成结果为：

```
[
  "((()))",
  "(()())",
  "(())()",
  "()(())",
  "()()()"
]
```

Solution：

```
public class Solution {
    private List<String> list = new ArrayList<>();

    public List<String> generateParenthesis(int n) {
        if (n < 1) {
            return list;
        }

        generate(n, 0, 0, "");
        return list;
    }

    private void generate(int n, int left, int right, String str) {
        if (left == right && left == n) {
            list.add(str);
        }
        if (left < n) {
            generate(n, left + 1, right, str + "(");
        }
        if (left > right) {
            generate(n, left, right + 1, str + ")");
        }
    }
}
```

23.合并K个排序链表

题目描述：

合并 k 个排序链表，返回合并后的排序链表。请分析和描述算法的复杂度。

示例:

```
输入:
[
  1->4->5,
  1->3->4,
  2->6
]
输出: 1->1->2->3->4->4->5->6
```

Solution：

```
public class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
        if (lists == null || lists.length == 0) {
            return null;
        }

        ListNode dummy = new ListNode(0);
        ListNode pre = dummy;

        // 使用小顶堆，每次取出的都是最小的节点
        Queue<ListNode> minHeap = new PriorityQueue<>(Comparator.comparingInt(node -> node
.val));
        for (ListNode list : lists) {
            if (list != null) {
                minHeap.offer(list);
            }
        }

        while (!minHeap.isEmpty()) {
            pre.next = minHeap.poll();
            pre = pre.next;
            if (pre.next != null) {
                minHeap.offer(pre.next);
            }
        }

        return dummy.next;
    }
}
```

31.下一个排列

题目描述：

实现获取下一个排列的函数，算法需要将给定数字序列重新排列成字典序中下一个更大的排列。

如果不存在下一个更大的排列，则将数字重新排列成最小的排列（即升序排列）。

必须**原地**修改，只允许使用额外常数空间。

以下是一些例子，输入位于左侧列，其相应输出位于右侧列。

`1,2,3` → `1,3,2`

`3,2,1` → `1,2,3`

`1,1,5` → `1,5,1`

Solution：

```
/** * 求下一个全排列，可分为两种情况： * 1.例如像 5 4 3 2 1这样的序列，已经是最大的排列，即每个位置上的数非递增，这时只需要翻转整个序列即可 * 2.例如像 1 3 5 4 2这样的序列，要从后往前找到第一个比后面一位小的元素的位置，即第二个位置的3，然后与其后第一个比它大的元素交换位置，得到 1 4 5 3 2，再将 5 3 2翻转得到 1 4 2 3 5即可 */

public class Solution {
    public void nextPermutation(int[] nums) {
        if (nums == null || nums.length == 0) {
            return;
        }

        int firstSmall = -1;
        for (int i = nums.length - 2; i >= 0; i--) {
            if (nums[i] < nums[i + 1]) {
                firstSmall = i;
                break;
            }
        }

        if (firstSmall == -1) {
            reverse(nums, 0, nums.length - 1);
            return;
        }

        for (int i = nums.length - 1; i > firstSmall; i--) {
            if (nums[i] > nums[firstSmall]) {
                swap(nums, i, firstSmall);
                reverse(nums, firstSmall + 1, nums.length - 1);
                return;
            }
        }
    }

    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }

    private void reverse(int[] nums, int start, int end) {
        while (start < end) {
            swap(nums, start++, end--);
        }
    }
}
```

32.最长有效括号

题目描述：

给定一个只包含 `'('` 和 `')'` 的字符串，找出最长的包含有效括号的子串的长度。

示例 1:

```
输入: "(()"
输出: 2
解释: 最长有效括号子串为 "()"
```

示例 2:

```
输入: ")()())"
输出: 4
解释: 最长有效括号子串为 "()()"
```

Solution:

```
public class Solution {
    public int longestValidParentheses(String s) {
        if (s == null || s.length() < 2) {
            return 0;
        }

        int res = 0;
        int start = 0;
        Stack<Integer> stack = new Stack<>();
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == '(') {
                stack.push(i);
            } else {
                if (stack.isEmpty()) {
                    start = i + 1;
                } else {
                    stack.pop();
                    res = stack.isEmpty() ? Math.max(res, i - start + 1) : Math.max(res, i
- stack.peek());
                }
            }
        }
        return res;
    }
}
```

33.搜索旋转排序数组

题目描述:

假设按照升序排序的数组在预先未知的某个点上进行了旋转。

(例如，数组 `[0,1,2,4,5,6,7]` 可能变为 `[4,5,6,7,0,1,2]`)。

搜索一个给定的目标值，如果数组中存在这个目标值，则返回它的索引，否则返回 `-1` 。

你可以假设数组中不存在重复的元素。

你的算法时间复杂度必须是 O(log n) 级别。

示例 1:

```
输入: nums = [4,5,6,7,0,1,2], target = 0
输出: 4
```

示例 2:

```
输入: nums = [4,5,6,7,0,1,2], target = 3
输出: -1
```

Solution:

```
public class Solution {
    public int search(int[] nums, int target) {
        if (nums == null
            || nums.length < 1
            || (target < nums[0] && target > nums[nums.length - 1])) {
            return -1;
        }

        int low = 0;
        int high = nums.length - 1;
        while (low <= high) {
            int mid = (low + high) >> 1;
            if (nums[mid] == target) {
                return mid;
            }

            if (nums[mid] >= nums[low]) {// 左边有序
                if (nums[mid] > target && nums[low] <= target) {// 在有序边
                    high = mid - 1;
                } else{// 在无序边
                    low = mid + 1;
                }
            } else {// 右边有序
                if (nums[mid] < target && nums[high] >= target) {// 在有序边
                    low = mid + 1;
                } else {// 在无序边
                    high = mid - 1;
                }
            }
        }

        return -1;
    }
}
```

34.在排序数组中查找元素的第一个和最后一个位置

题目描述：

给定一个按照升序排列的整数数组 `nums` ，和一个目标值 `target` 。找出给定目标值在数组中的开始位置和结束位置。

你的算法时间复杂度必须是 O(log n) 级别。

如果数组中不存在目标值，返回 `[-1, -1]` 。

示例 1:

```
输入: nums = [5,7,7,8,8,10], target = 8
输出: [3,4]
```

示例 2:

```
输入: nums = [5,7,7,8,8,10], target = 6
输出: [-1,-1]
```

Solution：

```
public class Solution {
    public int[] searchRange(int[] nums, int target) {
        if (nums == null || nums.length < 1) {
            return new int[]{-1, -1};
        }

        int low = 0;
        int high = nums.length - 1;
        while (low <= high) {
            int mid = (low + high) >> 1;
            if (nums[mid] < target) {
                low = mid + 1;
            } else if (nums[mid] > target) {
                high = mid - 1;
            } else {
                int left = mid;
                int right = mid;
                while (left >= low && nums[left] == target) {
                    left--;
                }
                while (right <= high && nums[right] == target) {
                    right++;
                }
                return new int[]{left + 1, right -1};
            }
        }

        return new int[]{-1, -1};
    }
}
```

39.组合总和

题目描述：

给定一个**无重复元素**的数组 `candidates` 和一个目标数 `target` ，找出 `candidates` 中所有可以使数字和为 `target` 的组合。

`candidates` 中的数字可以无限制重复被选取。

说明：

- 所有数字（包括 `target` ）都是正整数。
- 解集不能包含重复的组合。

示例 1：

```
输入: candidates = [2,3,6,7], target = 7,
所求解集为:
[
  [7],
  [2,2,3]
]
```

示例 2:

```
输入: candidates = [2,3,5], target = 8,
所求解集为:
[
  [2,2,2,2],
  [2,3,3],
  [3,5]
]
```

Solution:

```
public class Solution {
    private List<List<Integer>> res = new ArrayList<>();;

    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        if (candidates == null || candidates.length == 0) {
            return res;
        }
        dfs(candidates, 0, target, new ArrayList<>());
        return res;
    }

    private void dfs(int[] candidates, int start, int target, List<Integer> list) {
        if (target == 0) {
            res.add(new ArrayList<>(list));
            return;
        }

        for (int i = start; i < candidates.length; i++) {
            if (target >= candidates[i]) {
                list.add(candidates[i]);
                dfs(candidates, i, target - candidates[i], list);
                list.remove(list.size() - 1);
            }
        }
    }
}
```

42.接雨水

题目描述:

给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。

上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。 **感谢 Marcos** 贡献此图。

示例:

```
输入: [0,1,0,2,1,0,1,3,2,1,2,1]
输出: 6
```

Solution:

```
public class Solution {
    public int trap(int[] height) {
        if (height == null || height.length < 3) {
            return 0;
        }

        int low = 0;
        int high = height.length - 1;
        int res = 0;
        int lowMax = 0;
        int highMax = 0;
        while (low < high) {
            if (height[low] < height[high]) {
                lowMax = Math.max(lowMax, height[low]);
                res += lowMax - height[low];
                low++;
            } else {
                highMax = Math.max(highMax, height[high]);
                res += highMax - height[high];
                high--;
            }
        }

        return res;
    }
}
```

46.全排列

题目描述：

给定一个**没有重复**数字的序列，返回其所有可能的全排列。

示例:

```
输入: [1,2,3]
输出:
[
  [1,2,3],
  [1,3,2],
  [2,1,3],
  [2,3,1],
  [3,1,2],
  [3,2,1]
]
```

Solution：

```
public class Solution {
    private List<List<Integer>> res = new ArrayList<>();
    private boolean[] visited;

    public List<List<Integer>> permute(int[] nums) {
        if (nums == null) {
            return res;
        }

        visited = new boolean[nums.length];
        permute(0, nums, new ArrayList());
        return res;
    }

    private void permute(int index, int[] nums, List<Integer> list) {
        if (index == nums.length) {
            res.add(new ArrayList(list));
            return;
        }

        for (int i = 0; i < nums.length; i++) {
            if (!visited[i]) {
                list.add(nums[i]);
                visited[i] = true;
                permute(index + 1, nums, list);
                list.remove(list.size() - 1);
                visited[i] = false;
            }
        }
    }
}
```

48.旋转图像

题目描述：

给定一个 n × n 的二维矩阵表示一个图像。

将图像顺时针旋转 90 度。

说明：

你必须在**原地**旋转图像，这意味着你需要直接修改输入的二维矩阵。**请不要**使用另一个矩阵来旋转图像。

示例 1:

```
给定 matrix =
[
  [1,2,3],
  [4,5,6],
  [7,8,9]
],

原地旋转输入矩阵，使其变为:
[
  [7,4,1],
  [8,5,2],
  [9,6,3]
]
```

示例 2:

```
给定 matrix =
[
  [ 5, 1, 9,11],
  [ 2, 4, 8,10],
  [13, 3, 6, 7],
  [15,14,12,16]
],

原地旋转输入矩阵，使其变为:
[
  [15,13, 2, 5],
  [14, 3, 4, 1],
  [12, 6, 8, 9],
  [16, 7,10,11]
]
```

Solution：

```
/** * 以第一个数据为例分析： * 1 2 3 * 4 5 6 * 7 8 9 * 先做左右对称翻转，得到： * 3 2 1 * 6 5 4 * 9 8 7 * 再以副对角线为轴做翻转，得到： * 7 4 1 * 8 5 2 * 9 6 3 * 此时即为要求的结果 * 故—要先做左右对称翻转，再做一次副对角线翻转即可 */
public class Solution {
    public void rotate(int[][] matrix) {
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
            return;
        }

        int n = matrix.length;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < (n >> 1); j++) {
                swap(matrix, i, j, i, n - j - 1);
            }
        }

        for (int i = 0; i < n; i++) {
            for (int j = 0; j + i + 1 < n; j++) {
                swap(matrix, i, j, n - 1 - j, n - 1 - i);
            }
        }
    }

    private void swap(int[][] matrix, int i, int j, int p, int q) {
        int temp = matrix[i][j];
        matrix[i][j] = matrix[p][q];
        matrix[p][q] = temp;
    }
}
```

49.字母异位词分组

题目描述：

给定一个字符串数组，将字母异位词组合在一起。字母异位词指字母相同，但排列不同的字符串。

示例:

```
输入: ["eat", "tea", "tan", "ate", "nat", "bat"],
输出:
[
  ["ate","eat","tea"],
  ["nat","tan"],
  ["bat"]
]
```

说明：

- 所有输入均为小写字母。
- 不考虑答案输出的顺序。

Solution:

```
public class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        List<List<String>> res = new ArrayList<>();
        if (strs == null || strs.length == 0) {
            return res;
        }

        Map<String, List<String>> map = new HashMap<>();
        for (String str : strs) {
            int[] counts = new int[26];
            for (int i = 0; i < str.length(); i++) {
                counts[str.charAt(i) - 'a']++;
            }

            StringBuilder sb = new StringBuilder();
            for (int count : counts) {
                sb.append(' ').append(count);
            }
            String key = sb.toString();
            if (!map.containsKey(key)) {
                map.put(key, new ArrayList<>());
            }
            map.get(key).add(str);
        }
        return new ArrayList<>(map.values());
    }
}
```

53.最大子序和

题目描述:

给定一个整数数组 `nums` ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

示例:

```
输入: [-2,1,-3,4,-1,2,1,-5,4],
输出: 6
解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。
```

Solution:

```
public class Solution {
    public int maxSubArray(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }

        int res = nums[0];
        int max = 0;
        for (int num : nums) {
            max = Math.max(max + num, num);
            res = Math.max(res, max);
        }
        return res;
    }
}
```

55.跳跃游戏

题目描述：

给定一个非负整数数组，你最初位于数组的第一个位置。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

判断你是否能够到达最后一个位置。

示例 1:

```
输入: [2,3,1,1,4]
输出: true
解释: 从位置 0 到 1 跳 1 步, 然后跳 3 步到达最后一个位置。
```

示例 2:

```
输入: [3,2,1,0,4]
输出: false
解释: 无论怎样，你总会到达索引为 3 的位置。但该位置的最大跳跃长度是 0 ， 所以你永远不可能到达最后一个位置。
```

Solution：

```
/* * 从后往前跳 */
public class Solution {
    public boolean canJump(int[] nums) {
        int last = nums.length - 1;
        for (int i = nums.length - 1; i >= 0; i--) {
            if (nums[i] + i >= last) {
                last = i;
            }
        }
        return last == 0;
    }
}
```

56.合并区间

题目描述：

给出一个区间的集合，请合并所有重叠的区间。

示例 1:

```
输入: [[1,3],[2,6],[8,10],[15,18]]
输出: [[1,6],[8,10],[15,18]]
解释: 区间 [1,3] 和 [2,6] 重叠, 将它们合并为 [1,6].
```

示例 2:

```
输入: [[1,4],[4,5]]
输出: [[1,5]]
解释: 区间 [1,4] 和 [4,5] 可被视为重叠区间。
```

Solution：

```
public class Solution {
    public List<Interval> merge(List<Interval> intervals) {
        if (intervals == null || intervals.size() < 2) {
            return intervals;
        }

        List<Interval> list = new ArrayList<>();
        intervals.sort(Comparator.comparingInt(interval -> interval.start));

        Interval pre = null;
        for (Interval interval : intervals) {
            if (pre == null || pre.end < interval.start) {
                list.add(interval);
                pre = interval;
            } else {
                pre.end = Math.max(pre.end, interval.end);
            }
        }
        return list;
    }
}
```