20. 有效的括号
21. 合并两个有序链表
53. 最大子数组和
70. 爬楼梯
94. 二叉树的中序遍历
101. 对称二叉树
104. 二叉树的最大深度
121. 买卖股票的最佳时机
136. 只出现一次的数字
141. 环形链表
155. 最小栈
160. 相交链表
169. 多数元素
206. 反转链表
226. 翻转二叉树
234. 回文链表
283. 移动零
338. 比特位计数
448. 找到所有数组中消失的数字☆ ☆ ☆
461. 汉明距离
543. 二叉树的直径
617. 合并二叉树
2. 两数相加
3. 无重复字符的最长子串
4. 寻找两个正序数组的中位数☆ ☆ ☆ ☆
5. 最长回文子串
11. 盛最多水的容器☆ ☆ ☆
15. 三数之和
17. 电话号码的字母组合
19. 删除链表的倒数第 N 个结点
22. 括号生成
31. 下一个排列☆ ☆ ☆ ☆
33. 搜索旋转排序数组
34. 在排序数组中查找元素的第一个和最后一个位置☆
39. 组合总和
46. 全排列
48. 旋转图像☆
49. 字母异位词分组☆
55. 跳跃游戏☆
56. 合并区间
62. 不同路径
64. 最小路径和
75. 颜色分类
78. 子集
79. 单词搜索☆
96. 不同的二叉搜索树☆
98. 验证二叉搜索树
102. 二叉树的层序遍历
105. 从前序与中序遍历序列构造二叉树
114. 二叉树展开为链表☆ ☆ ☆
128. 最长连续序列☆ ☆ ☆ ☆
139. 单词拆分
142. 环形链表 II
148. 排序链表
152. 乘积最大子数组☆ ☆ ☆

198. 打家劫舍
200. 岛屿数量
207. 课程表
208. 实现 Trie (前缀树)☆
215. 数组中的第K个最大元素☆☆☆☆
221. 最大正方形☆
236. 二叉树的最近公共祖先
238. 除自身以外数组的乘积
240. 搜索二维矩阵 II
279. 完全平方数
287. 寻找重复数
300. 最长递增子序列
309. 最佳买卖股票时机含冷冻期☆☆☆☆
322. 零钱兑换
337. 打家劫舍 III☆
347. 前 K 个高频元素
406. 根据身高重建队列
416. 分割等和子集
394. 字符串解码
437. 路径总和 III
399. 除法求值☆☆☆
494. 目标和
438. 找到字符串中所有字母异位词☆☆☆☆
538. 把二叉搜索树转换为累加树
560. 和为 K 的子数组☆☆
581. 最短无序连续子数组☆☆
647. 回文子串☆
739. 每日温度
621. 任务调度器
146. LRU 缓存

20. 有效的括号

难度 简单 2942 收藏 分享 切换为英文 接收动态 反馈

给定一个只包括 `'('`，`')'`，`'{'`，`'}'`，`'['`，`']'` 的字符串 `s`，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。

示例 1：

```
输入：s = "()"
输出：true
```

示例 2：

```
输入：s = "()[]{}"
输出：true
```

示例 3：

```
输入：s = "(]"
输出：false
```

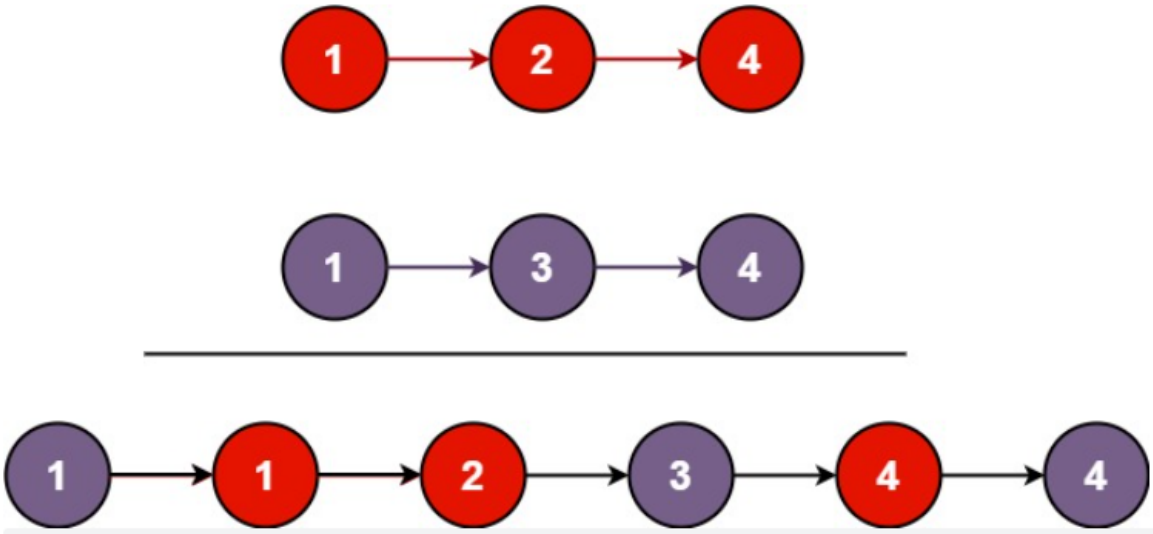
```python
class Solution:
    def isValid(self, s: str) -> bool:
        dic = {
            ')': '(',
            ']': '[',
            '}': '{'
        }
        stack = []
        for c in s:
            if c in dic:
                if stack and dic[c] == stack[-1]:
                    stack.pop()
                else:
                    return False
            else:
                stack.append(c)
        return len(stack) == 0
```

21. 合并两个有序链表

难度 简单 2166 收藏 分享 切换为英文 接收动态 反馈

将两个升序链表合并为一个新的 **升序** 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 1：



输入： l1 = [1,2,4], l2 = [1,3,4]
输出： [1,1,2,3,4,4]

示例 2：

输入： l1 = [], l2 = []
输出： []

```python
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2:
Optional[ListNode]) -> Optional[ListNode]:
        dummy = ListNode()
        p = dummy
        while list1 and list2:
            if list1.val < list2.val:
```

```
                p.next = list1
                list1 = list1.next
            else:
                p.next = list2
                list2 = list2.next
            p = p.next
        if list1:
            p.next = list1
        if list2:
            p.next = list2
        return dummy.next
```

53. 最大子数组和

难度 简单 4289 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 `nums` ，请你找出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。

子数组 是数组中的一个连续部分。

示例 1：

输入： nums = [-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 。

示例 2：

输入： nums = [1]
输出： 1

示例 3：

输入： nums = [5,4,-1,7,8]
输出： 23

提示：

- `1 <= nums.length <= 10^5`
- `-10^4 <= nums[i] <= 10^4`

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        res = nums[0]
        dp = [0] * len(nums)
        dp[0] = nums[0]
        for i in range(1, len(nums)):
            dp[i] = max(dp[i-1] + nums[i], nums[i])
            res = max(res, dp[i])
        return res
```

70. 爬楼梯

难度 简单 2150 收藏 分享 切换为英文 接收动态 反馈

假设你正在爬楼梯。需要 `n` 阶你才能到达楼顶。

每次你可以爬 `1` 或 `2` 个台阶。你有多少种不同的方法可以爬到楼顶呢?

示例 1:

```
输入：n = 2
输出：2
解释：有两种方法可以爬到楼顶。
1. 1 阶 + 1 阶
2. 2 阶
```

示例 2:

```
输入：n = 3
输出：3
解释：有三种方法可以爬到楼顶。
1. 1 阶 + 1 阶 + 1 阶
2. 1 阶 + 2 阶
3. 2 阶 + 1 阶
```

提示:

- `1 <= n <= 45`

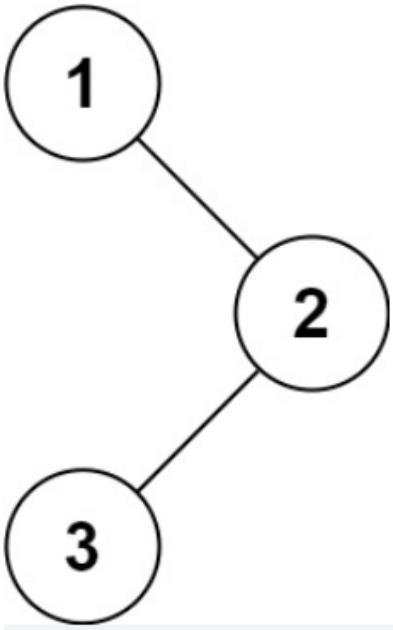
```python
class Solution:
    def climbStairs(self, n: int) -> int:
        if n == 1: return 1
        dp = [0] * (n + 1)
        dp[1] = 1
        dp[2] = 2
        for i in range(3, n + 1):
            dp[i] = dp[i-1] + dp[i-2]
        return dp[-1]
```

94. 二叉树的中序遍历

难度 简单 1251 收藏 分享 切换为英文 接收动态 反馈

给定一个二叉树的根节点 `root` ，返回它的 **中序** 遍历。

示例 1：



输入： root = [1,null,2,3]
输出： [1,3,2]

```
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def inorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        self.res = []
        def helper(root):
            if not root:
                return None
            helper(root.left)
            self.res.append(root.val)
            helper(root.right)
        helper(root)
        return self.res
```

```
class Solution:
    def inorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        if not root: return []
        res = []
        stack = [root]
        while stack:
            node = stack.pop()
            if node:
                if node.right: stack.append(node.right)
                stack.append(node)
```

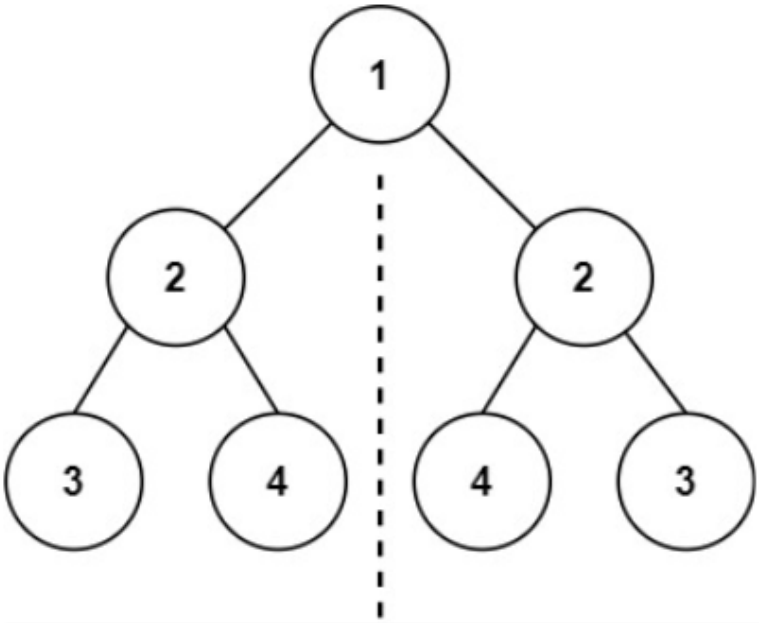
```
                stack.append(None)
                if node.left: stack.append(node.left)
            else:
                node = stack.pop()
                res.append(node.val)
        return res
```

101. 对称二叉树

难度 简单 1722 收藏 分享 切换为英文 接收动态 反馈

给你一个二叉树的根节点 `root` ， 检查它是否轴对称。

示例 1：




```
输入：root = [1,2,2,3,4,4,3]
输出：true
```

```
class Solution:
    def isSymmetric(self, root: TreeNode) -> bool:
        def helper(p, q):
            if not p and not q: return True
            if not p and q: return False
            if p and not q: return False
            if p.val != q.val: return False
            return helper(p.left, q.right) and helper(p.right, q.left)

        if not root: return True
        return helper(root.left, root.right)
```

104. 二叉树的最大深度

难度 简单 1100 收藏 分享 切换为英文 接收动态 反馈

给定一个二叉树，找出其最大深度。

二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

说明: 叶子节点是指没有子节点的节点。

示例:
给定二叉树 `[3,9,20,null,null,15,7]`，

```
    3
   / \
  9  20
    /  \
   15   7
```

返回它的最大深度 3 。

```python
class Solution:
    def maxDepth(self, root: TreeNode) -> int:
        def helper(root):
            if not root: return 0
            left = helper(root.left)
            right = helper(root.right)
            return max(left, right) + 1
        return helper(root)
```

121. 买卖股票的最佳时机

难度 简单 2087 收藏 分享 切换为英文 接收动态 反馈

给定一个数组 `prices` ，它的第 `i` 个元素 `prices[i]` 表示一支给定股票第 `i` 天的价格。

你只能选择 **某一天** 买入这只股票，并选择在 **未来的某一个不同的日子** 卖出该股票。设计一个算法来计算你所能获取的最大利润。

返回你可以从这笔交易中获取的最大利润。如果你不能获取任何利润，返回 `0` 。

示例 1:

```
输入：[7,1,5,3,6,4]
输出：5
解释：在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。
     注意利润不能是 7-1 = 6, 因为卖出价格需要大于买入价格；同时，你不能在买入前卖出股票。
```

示例 2:

```
输入：prices = [7,6,4,3,1]
输出：0
解释：在这种情况下，没有交易完成，所以最大利润为 0。
```

提示:

- `1 <= prices.length <= 10^5`
- `0 <= prices[i] <= 10^4`

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        dp = [[0] * 2 for _ in range(len(prices))]
        dp[0][0] = -prices[0]
        for i in range(1, len(prices)):
            dp[i][0] = max(dp[i-1][0], -prices[i])
            dp[i][1] = max(dp[i-1][1], dp[i-1][0] + prices[i])
        return dp[-1][-1]
```

136. 只出现一次的数字

难度 简单 2229 收藏 分享 切换为英文 接收动态 反馈

给定一个**非空**整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

说明：

你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？

示例 1:

```
输入: [2,2,1]
输出: 1
```

示例 2:

```
输入: [4,1,2,1,2]
输出: 4
```

```python
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        res = 0
        for i in nums:
            res ^= i
        return res
```

141. 环形链表

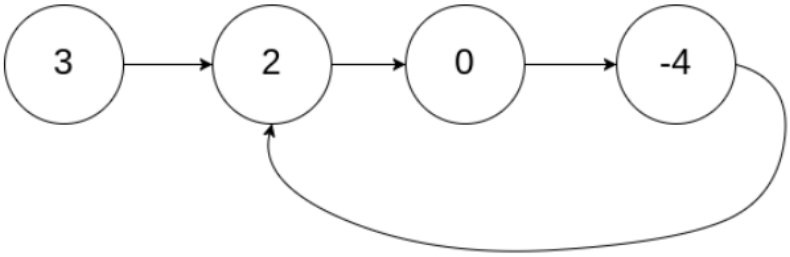
难度 简单 1336 收藏 分享 切换为英文 接收动态 反馈

给你一个链表的头节点 `head` ，判断链表中是否有环。

如果链表中有某个节点，可以通过连续跟踪 `next` 指针再次到达，则链表中存在环。 为了表示给定链表中的环，评测系统内部使用整数 `pos` 来表示链表尾连接到链表中的位置（索引从 0 开始）。**注意：** **`pos` 不作为参数进行传递** 。仅仅是为了标识链表的实际情况。

如果链表中存在环 ，则返回 `true` 。 否则，返回 `false` 。

示例 1：



输入： head = [3,2,0,-4], pos = 1
输出： true
解释： 链表中有一个环，其尾部连接到第二个节点。

```python
class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        fast, slow = head, head
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
            if fast == slow: return True
        return False
```

155. 最小栈

难度 简单 1156 收藏 分享 切换为英文 接收动态 反馈

设计一个支持 `push` ， `pop` ， `top` 操作，并能在常数时间内检索到最小元素的栈。

- `push(x)` —— 将元素 x 推入栈中。
- `pop()` —— 删除栈顶的元素。
- `top()` —— 获取栈顶元素。
- `getMin()` —— 检索栈中的最小元素。

示例:

输入：
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

输出：
[null,null,null,null,-3,null,0,-2]

解释：
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin();   --> 返回 -3.
minStack.pop();
minStack.top();   --> 返回 0.
minStack.getMin();   --> 返回 -2.

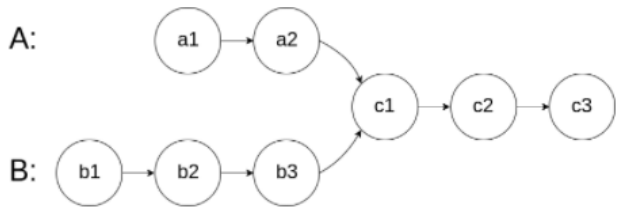
```python
class MinStack:
    def __init__(self):
        self.a = []
        self.b = []

    def push(self, val: int) -> None:
        self.a.append(val)
        if not self.b or self.b[-1] >= val:
            self.b.append(val)

    def pop(self) -> None:
        val = self.a.pop()
        if self.b[-1] == val:
            self.b.pop()

    def top(self) -> int:
        return self.a[-1]

    def getMin(self) -> int:
        return self.b[-1]
```

160. 相交链表

难度 简单 1523 收藏 分享 切换为英文 接收动态 反馈

给你两个单链表的头节点 `headA` 和 `headB` ，请你找出并返回两个单链表相交的起始节点。如果两个链表不存在相交节点，返回 `null` 。

图示两个链表在节点 `c1` 开始相交：



题目数据 **保证** 整个链式结构中不存在环。

注意，函数返回结果后，链表必须 **保持其原始结构** 。

自定义评测：

评测系统 的输入如下（你设计的程序 **不适用** 此输入）：

- `intersectVal` - 相交的起始节点的值。如果不存在相交节点，这一值为 `0`
- `listA` - 第一个链表
- `listB` - 第二个链表
- `skipA` - 在 `listA` 中（从头节点开始）跳到交叉节点的节点数
- `skipB` - 在 `listB` 中（从头节点开始）跳到交叉节点的节点数

评测系统将根据这些输入创建链式数据结构，并将两个头节点 `headA` 和 `headB` 传递给你的程序。如果程序能够正确返回相交节点，那么你的解决方案将被 **视作正确答案** 。

```python
class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
        a, b = headA, headB
        while a != b:
            a = a.next if a else headB
            b = b.next if b else headA
        return a
```

169. 多数元素

难度 简单 1286 收藏 分享 切换为英文 接收动态 反馈

给定一个大小为 *n* 的数组，找到其中的多数元素。多数元素是指在数组中出现次数 **大于** `⌊ n/2 ⌋` 的元素。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

示例 1：

```
输入：[3,2,3]
输出：3
```

示例 2：

```
输入：[2,2,1,1,1,2,2]
输出：2
```

进阶：

- 尝试设计时间复杂度为 O(n)、空间复杂度为 O(1) 的算法解决此问题。

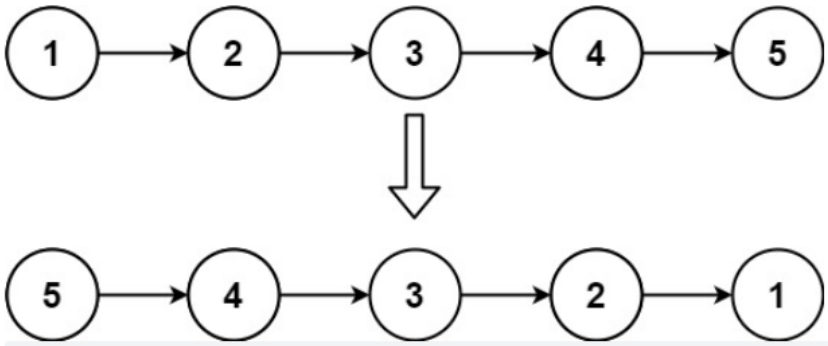
```python
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        cnt = 1
        candidate = nums[0]
        for i in range(1, len(nums)):
            if nums[i] == candidate:
                cnt += 1
            else:
                cnt -= 1
                if cnt == 0:
                    cnt = 1
                    candidate = nums[i]
        return candidate
```

206. 反转链表

难度 简单 2243 收藏 分享 切换为英文 接收动态 反馈

给你单链表的头节点 `head` ，请你反转链表，并返回反转后的链表。

示例 1：




```
输入：head = [1,2,3,4,5]
输出：[5,4,3,2,1]
```

```python
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        pre, cur = None, head
        while cur is not None:
            nxt = cur.next
            cur.next = pre
            pre = cur
            cur = nxt
        return pre
```

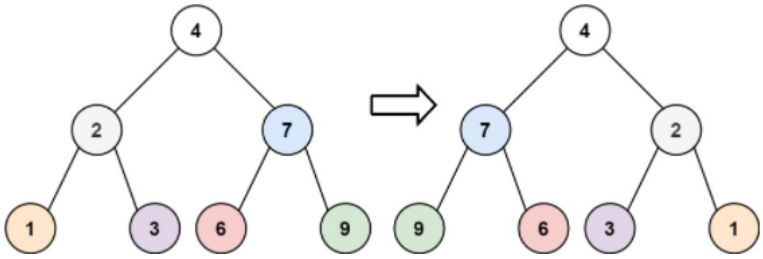
```python
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        def helper(head):
            if not head: return None
            if not head.next: return head
            node = helper(head.next)
            head.next.next = head
            head.next = None
            return node
        return helper(head)
```

226. 翻转二叉树

难度 简单 1158 收藏 分享 切换为英文 接收动态 反馈

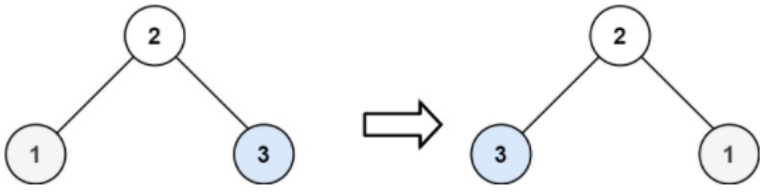
给你一棵二叉树的根节点 `root` ，翻转这棵二叉树，并返回其根节点。

示例 1：



输入： root = [4,2,7,1,3,6,9]
输出： [4,7,2,9,6,3,1]

示例 2：



输入： root = [2,1,3]
输出： [2,3,1]

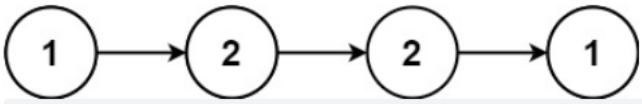
```python
class Solution:
    def invertTree(self, root: TreeNode) -> TreeNode:
        def helper(root):
            if not root: return None
            left = helper(root.left)
            right = helper(root.right)
            root.left, root.right = right, left
            return root
        return helper(root)
```

234. 回文链表

难度 简单 1249 收藏 分享 切换为英文 接收动态 反馈

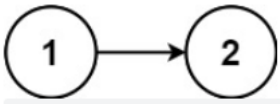
给你一个单链表的头节点 `head` ，请你判断该链表是否为回文链表。如果是，返回 `true` ；否则，返回 `false` 。

示例 1：



输入： head = [1,2,2,1]
输出： true

示例 2：



输入： head = [1,2]
输出： false

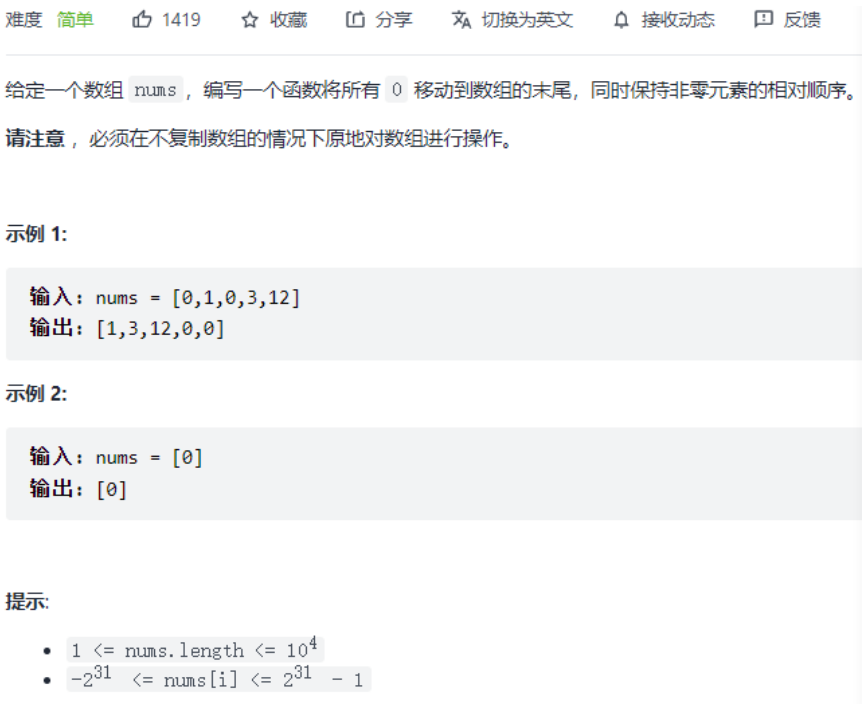
提示：

- 链表中节点数目在范围 $[1, 10^5]$ 内
- `0 <= Node.val <= 9`

```python
class Solution:
    def isPalindrome(self, head: ListNode) -> bool:
        def reverse(head):
            pre, cur = None, head
            while cur:
                nxt = cur.next
                cur.next = pre
                pre = cur
                cur = nxt
            return pre

        dummy = ListNode()
        dummy.next = head
        fast, slow = dummy, dummy
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
        left_tail = slow
        right_head = slow.next
        slow.next = None
```

```
        l2 = reverse(right_head)
        l1 = head
        while l1 and l2:
            if l1.val == l2.val:
                l1 = l1.next
                l2 = l2.next
            else:
                return False
        return True
```

283. 移动零



难度 简单 1419 收藏 分享 切换为英文 接收动态 反馈

给定一个数组 `nums`，编写一个函数将所有 `0` 移动到数组的末尾，同时保持非零元素的相对顺序。

请注意，必须在不复制数组的情况下原地对数组进行操作。

示例 1:

```
输入: nums = [0,1,0,3,12]
输出: [1,3,12,0,0]
```

示例 2:

```
输入: nums = [0]
输出: [0]
```

提示:

- `1 <= nums.length <= 10^4`
- `-2^31 <= nums[i] <= 2^31 - 1`

```
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        cur = 0
        for i in range(len(nums)):
            if nums[i] != 0:
                nums[cur] = nums[i]
                cur += 1
        for i in range(cur, len(nums)):
            nums[i] = 0
```

338. 比特位计数

难度 简单 878 收藏 分享 切换为英文 接收动态 反馈

给你一个整数 `n` ，对于 `0 <= i <= n` 中的每个 `i` ，计算其二进制表示中 **`1` 的个数** ，返回一个长度为 `n + 1` 的数组 `ans` 作为答案。

示例 1：

```
输入：n = 2
输出：[0,1,1]
解释：
0 --> 0
1 --> 1
2 --> 10
```

```python
class Solution:
    def countBits(self, n: int) -> List[int]:
        def bit(i):
            cnt = 0
            while i > 0:
                cnt += i & 1
                i >>= 1
            return cnt
        ans = []
        for i in range(n + 1):
            ans.append(bit(i))
        return ans
```

448. 找到所有数组中消失的数字☆☆☆

难度 简单 889 收藏 分享 切换为英文 接收动态 反馈

给你一个含 `n` 个整数的数组 `nums` ，其中 `nums[i]` 在区间 `[1, n]` 内。请你找出所有在 `[1, n]` 范围内但没有出现在 `nums` 中的数字，并以数组的形式返回结果。

示例 1：

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

示例 2：

```
输入：nums = [1,1]
输出：[2]
```

提示：

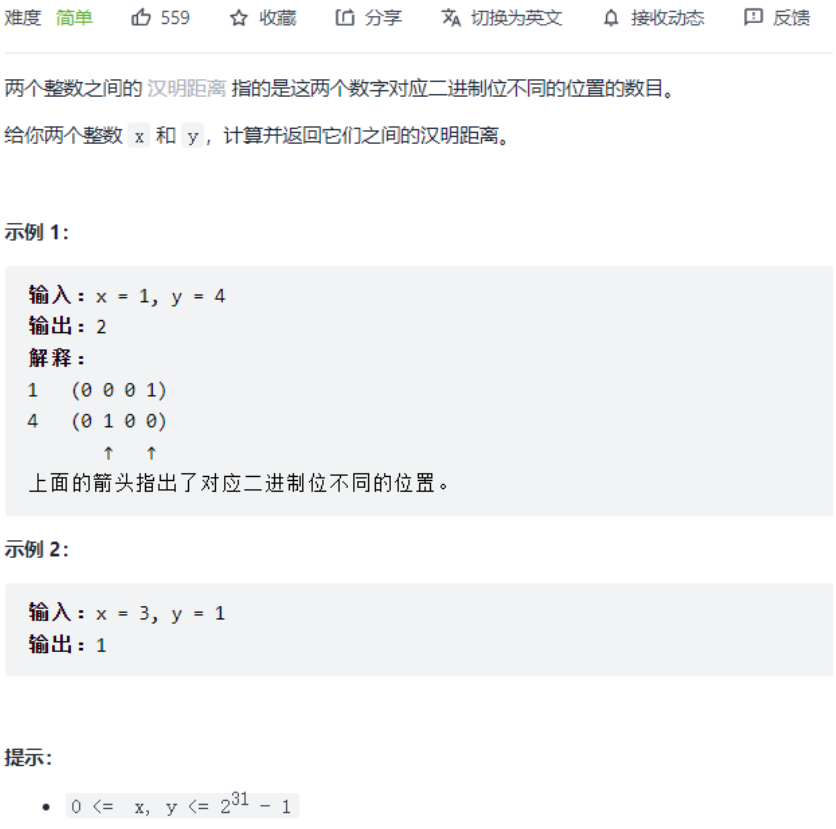
- `n == nums.length`
- `1 <= n <= 10^5`
- `1 <= nums[i] <= n`

进阶： 你能在不使用额外空间且时间复杂度为 `O(n)` 的情况下解决这个问题吗? 你可以假定返回的数组不算在额外空间内。

```python
class Solution:
    def findDisappearedNumbers(self, nums: List[int]) -> List[int]:
        # 原地哈希
        n = len(nums)
        idx = 0
        while idx < n:
```

```
            if nums[idx] == idx + 1:
                idx += 1
                continue
            # 如果nums[idx]不在正确的位置上，先看目标位上的数字，
            # 如果目标位的数字和idx的数字一样，说明idx的数字已经存在，idx右移，
            # 否则把idx和目标位的数字交换
            target_idx = nums[idx] - 1
            if nums[target_idx] == nums[idx]:
                idx += 1
                continue
            nums[idx], nums[target_idx] = nums[target_idx], nums[idx]

        res = []
        for i in range(n):
            if nums[i] != i + 1:
                res.append(i + 1)
        return res
```

461. 汉明距离



难度 简单 559 收藏 分享 切换为英文 接收动态 反馈

两个整数之间的 汉明距离 指的是这两个数字对应二进制位不同的位置的数目。

给你两个整数 `x` 和 `y`，计算并返回它们之间的汉明距离。

示例 1：

输入： x = 1, y = 4
输出： 2
解释：
1 (0 0 0 1)
4 (0 1 0 0)
↑ ↑
上面的箭头指出了对应二进制位不同的位置。

示例 2：

输入： x = 3, y = 1
输出： 1

提示：

- $0 <= x, y <= 2^{31} - 1$

```
class Solution:
    def hammingDistance(self, x: int, y: int) -> int:
        z = x ^ y
        cnt = 0
        while z > 0:
            cnt += z & 1
            z >>= 1
        return cnt
```

543. 二叉树的直径

难度 简单 900 收藏 分享 切换为英文 接收动态 反馈

给定一棵二叉树，你需要计算它的直径长度。一棵二叉树的直径长度是任意两个结点路径长度中的最大值。这条路径可能穿过也可能不穿过根结点。

示例：
给定二叉树

```
          1
         / \
        2   3
       / \
      4   5
```

返回 **3**, 它的长度是路径 [4,2,1,3] 或者 [5,2,1,3]。

注意：两结点之间的路径长度是以它们之间边的数目表示。

```python
class Solution:
    def diameterOfBinaryTree(self, root: TreeNode) -> int:
        self.d = 0
        def helper(root):
            if not root: return 0
            left = helper(root.left)
            right = helper(root.right)
            self.d = max(self.d, left + right + 1)
            return max(left, right) + 1
        helper(root)
        return self.d - 1 # 边的数目 = 点的数目 -1
```

617. 合并二叉树

难度 简单 866 收藏 分享 切换为英文 接收动态 反馈

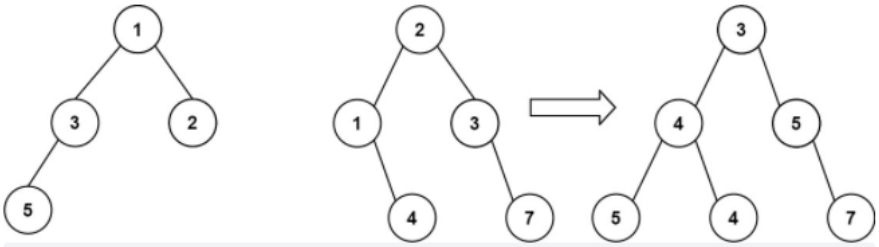
给你两棵二叉树： `root1` 和 `root2` 。

想象一下，当你将其中一棵覆盖到另一棵之上时，两棵树上的一些节点将会重叠（而另一些不会）。你需要将这两棵树合并成一棵新二叉树。合并的规则是：如果两个节点重叠，那么将这两个节点的值相加作为合并后节点的新值；否则，**不为** null 的节点将直接作为新二叉树的节点。

返回合并后的二叉树。

注意: 合并过程必须从两个树的根节点开始。

示例 1:




```
输入：root1 = [1,3,2,5], root2 = [2,1,3,null,4,null,7]
输出：[3,4,5,5,4,null,7]
```

示例 2:

```
输入：root1 = [1], root2 = [1,2]
输出：[2,2]
```

```python
class Solution:
    def mergeTrees(self, root1: TreeNode, root2: TreeNode) -> TreeNode:
        def helper(p, q):
            if not p and not q: return None
            if not p and q: return q
            if p and not q: return p
            node = TreeNode(p.val + q.val)
            node.left = helper(p.left, q.left)
            node.right = helper(p.right, q.right)
            return node
        return helper(root1, root2)
```

2. 两数相加

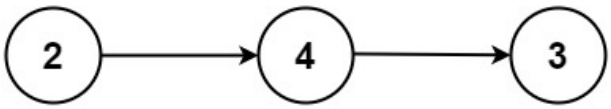
难度 中等 7422 收藏 分享 切换为英文 接收动态 反馈

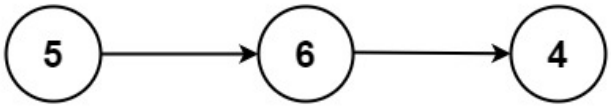
给你两个 **非空** 的链表，表示两个非负的整数。它们每位数字都是按照 **逆序** 的方式存储的，并且每个节点只能存储 **一位** 数字。

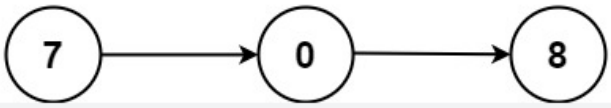
请你将两个数相加，并以相同形式返回一个表示和的链表。

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.

```python
# Definition for singly-linked list.
# 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:
        def list2int(head):
            res = 0
            i = 1
            while head:
                res += head.val * i
                i *= 10
                head = head.next
```

```
            return res

        def int2list(n):
            if n == 0: return ListNode(0)
            dummy = ListNode()
            p = dummy
            while n:
                v = n % 10
                n //= 10
                node = ListNode(v)
                p.next = node
                p = p.next
            return dummy.next

        a = list2int(l1)
        b = list2int(l2)
        res = a + b
        return int2list(res)
```

3. 无重复字符的最长子串

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 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

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        res = 0
        left = 0
        window = {}
        for right in range(len(s)):
            if s[right] not in window:
                window[s[right]] = 1
            else:
                window[s[right]] += 1
```

```
                while window[s[right]] > 1:
                    window[s[left]] -= 1
                    if window[s[left]] == 0:
                        window.pop(s[left])
                    left += 1
            res = max(res, right - left + 1)
        return res
```

####

4. 寻找两个正序数组的中位数☆☆☆☆

难度 困难 4932 收藏 分享 切换为英文 接收动态 反馈

给定两个大小分别为 `m` 和 `n` 的正序（从小到大）数组 `nums1` 和 `nums2`。请你找出并返回这两个正序数组的 **中位数**。

算法的时间复杂度应该为 `O(log (m+n))`。

示例 1：

```
输入：nums1 = [1,3], nums2 = [2]
输出：2.00000
解释：合并数组 = [1,2,3] ，中位数 2
```

示例 2：

```
输入：nums1 = [1,2], nums2 = [3,4]
输出：2.50000
解释：合并数组 = [1,2,3,4] ，中位数 (2 + 3) / 2 = 2.5
```

示例 3：

```
输入：nums1 = [0,0], nums2 = [0,0]
输出：0.00000
```

示例 4：

```
输入：nums1 = [], nums2 = [1]
输出：1.00000
```

从一般到特殊的方法，代码精简，边界清晰。 - 寻找两个正序数组的中位数 - 力扣（LeetCode） (leetcode-cn.com)

```
class Solution:
    def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) ->
float:
        # 本质是找第k小的数，k从1开始
        k1 = (len(nums1) + len(nums2) + 1) // 2
        k2 = (len(nums1) + len(nums2) + 2) // 2
        # 如果和是偶数，k1指向中间左侧的数，k2指向中间右侧的数
        # 如果和是奇数，k1 k2都指向中间
        def helper(nums1, nums2, k):
            # k是两个数组合并后的第k小的数
            # t是一刀切，找到两个数组分别的第k//2小的位置
            if len(nums1) < len(nums2):
                nums1, nums2 = nums2, nums1
            if len(nums2) == 0:
```

```
                return nums1[k-1]
            if k == 1:
                return min(nums1[0], nums2[0])
            t = min(len(nums2), k // 2)
            if nums1[t-1] >= nums2[t-1]:
                return helper(nums1, nums2[t:], k-t)
            else:
                return helper(nums1[t:], nums2, k-t)
        if k1 == k2:
            return helper(nums1, nums2, k1)
        else:
            return (helper(nums1, nums2, k1) + helper(nums1, nums2, k2)) / 2
```

5. 最长回文子串

难度 中等 4633 收藏 分享 切换为英文 接收动态 反馈

给你一个字符串 `s`，找到 `s` 中最长的回文子串。

示例 1：

```
输入：s = "babad"
输出："bab"
解释："aba" 同样是符合题意的答案。
```

示例 2：

```
输入：s = "cbbd"
输出："bb"
```

示例 3：

```
输入：s = "a"
输出："a"
```

示例 4：

```
输入：s = "ac"
输出："a"
```

```
class Solution:
    def longestPalindrome(self, s: str) -> str:
        if len(s) == 1: return s
        begin = 0
        max_len = 1
        dp = [[False] * len(s) for _ in range(len(s))]
        for i in range(len(s)):
            dp[i][i] = True
        for i in range(len(s) - 1, -1, -1):
            for j in range(i + 1, len(s)):
                if s[i] == s[j]:
                    if j - i == 1:
                        dp[i][j] = True
                    else:
                        dp[i][j] = dp[i+1][j-1]
                else:
                    dp[i][j] = False
                if dp[i][j] == True and (j - i + 1 > max_len):
                    max_len = j - i + 1
```

```
                    begin = i
        return s[begin: begin + max_len]
```

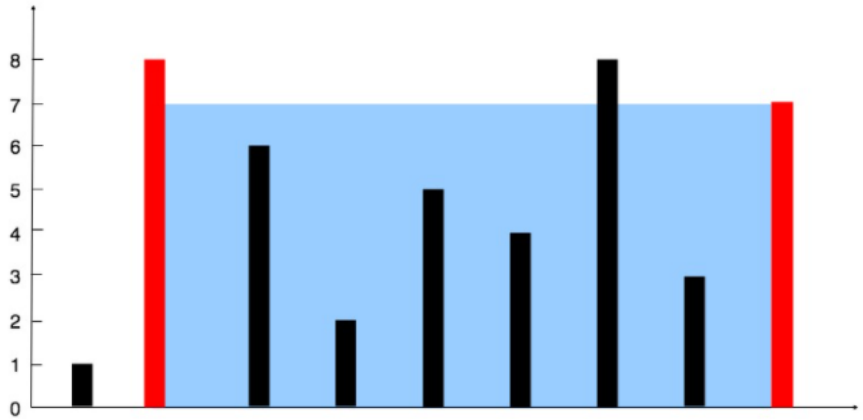
11. 盛最多水的容器☆☆☆

难度 中等 3158 收藏 分享 切换为英文 接收动态 反馈

给你 n 个非负整数 $a_1, a_2, ..., a_n$，每个数代表坐标中的一个点 (i, a_i) 。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, a_i) 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。

示例 1：



输入：[1,8,6,2,5,4,8,3,7]
输出：49
解释：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。

```
class Solution:
    def maxArea(self, height: List[int]) -> int:
        res = 0
        left, right = 0, len(height) - 1
        while left < right:
            res = max(res, min(height[left], height[right]) * (right - left))
            if height[left] < height[right]:
                left += 1
            else:
                right -= 1
        return res
```

15. 三数之和

难度 中等 4260 收藏 分享 切换为英文 接收动态 反馈

给你一个包含 `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 $a + b + c = 0$？请你找出所有和为 `0` 且不重复的三元组。

注意：答案中不可以包含重复的三元组。

示例 1：

```
输入：nums = [-1,0,1,2,-1,-4]
输出：[[-1,-1,2],[-1,0,1]]
```

示例 2：

```
输入：nums = []
输出：[]
```

示例 3：

```
输入：nums = [0]
输出：[]
```

```python
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        if len(nums) < 3: return []
        res = []
        nums.sort()

        # [-1, -1, 0, 1, 1]
        for i in range(len(nums)):
            if i > 0 and nums[i] == nums[i-1]: continue
            if nums[i] > 0: break

            j, k = i + 1, len(nums) - 1

            while j < k:

                s = nums[i] + nums[j] + nums[k]
                if s == 0:
                    res.append([nums[i], nums[j], nums[k]])
                    while j < k and nums[j] == nums[j + 1]: j += 1
                    while j < k and nums[k] == nums[k - 1]: k -= 1
                    j += 1
                    k -= 1
                elif s < 0:
                    j += 1
                else:
                    k -= 1
        return res
```

17. 电话号码的字母组合

难度 中等 1701 收藏 分享 切换为英文 接收动态 反馈

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。答案可以按 **任意顺序** 返回。

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 1:

```
输入：digits = "23"
输出：["ad","ae","af","bd","be","bf","cd","ce","cf"]
```

示例 2:

```
输入：digits = ""
输出：[]
```

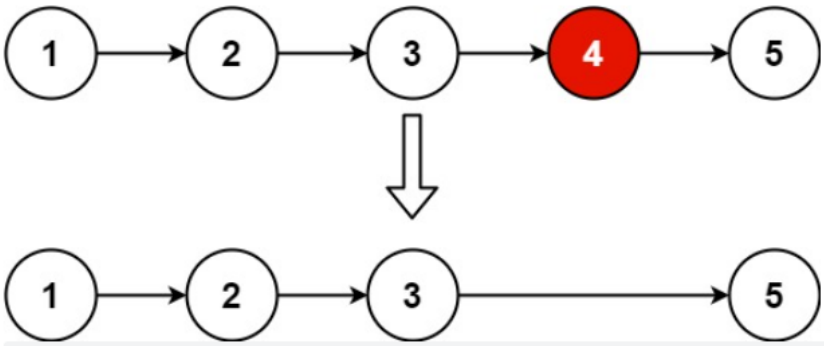
```python
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        dic = {
            '2': ['a', 'b', 'c'],
            '3': ['d', 'e', 'f'],
            '4': ['g', 'h', 'i'],
            '5': ['j', 'k', 'l'],
            '6': ['m', 'n', 'o'],
            '7': ['p', 'q', 'r', 's'],
            '8': ['t', 'u', 'v'],
            '9': ['w', 'x', 'y', 'z']
        }
        if len(digits) == 0: return []
        res = []
        path = []
        def dfs(path, cur_idx):
            if len(path) == len(digits):
                res.append(''.join(path))
                return
            for a in dic[digits[cur_idx]]:
                dfs(path + [a], cur_idx + 1)
        dfs(path, 0)
        return res
```

19. 删除链表的倒数第 N 个结点

难度 中等 1774 收藏 分享 切换为英文 接收动态 反馈

给你一个链表，删除链表的倒数第 `n` 个结点，并且返回链表的头结点。

示例 1：




```
输入：head = [1,2,3,4,5], n = 2
输出：[1,2,3,5]
```

示例 2：

```
输入：head = [1], n = 1
输出：[]
```

```python
class Solution:
    def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
        dummy = ListNode()
        dummy.next = head
        fast, slow = dummy, dummy
        for _ in range(n):
            fast = fast.next
        while fast.next:
            fast = fast.next
            slow = slow.next
        slow.next = slow.next.next
        return dummy.next
```

22. 括号生成

难度 中等 2338 收藏 分享 切换为英文 接收动态 反馈

数字 `n` 代表生成括号的对数，请你设计一个函数，用于能够生成所有可能的并且 **有效的** 括号组合。

示例 1：

```
输入：n = 3
输出：["((()))","(()())","(())()","()(())","()()()"]
```

示例 2：

```
输入：n = 1
输出：["()"]
```

提示：

- `1 <= n <= 8`

```python
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        res = []
        path = []
        left, right = n, n
        def backtrack(path, left, right):
            if left == right and left == 0:
                res.append(''.join(path))
                return

            if left > 0:
                backtrack(path + ['('], left - 1, right)
            if right > left:
                backtrack(path + [')'], left, right - 1)
        backtrack(path, left, right)
        return res
```

31. 下一个排列☆☆☆☆

下一个排列 - 下一个排列 - 力扣（LeetCode） (leetcode-cn.com)

难度 中等 1514 收藏 分享 切换为英文 接收动态 反馈

整数数组的一个 **排列** 就是将其所有成员以序列或线性顺序排列。

- 例如，`arr = [1,2,3]` ，以下这些都可以视作 `arr` 的排列：`[1,2,3]`、`[1,3,2]`、`[3,1,2]`、`[2,3,1]` 。

整数数组的 **下一个排列** 是指其整数的下一个字典序更大的排列。更正式地，如果数组的所有排列根据其字典顺序从小到大排列在一个容器中，那么数组的 **下一个排列** 就是在这个有序容器中排在它后面的那个排列。如果不存在下一个更大的排列，那么这个数组必须重排为字典序最小的排列（即，其元素按升序排列）。

- 例如，`arr = [1,2,3]` 的下一个排列是 `[1,3,2]` 。
- 类似地，`arr = [2,3,1]` 的下一个排列是 `[3,1,2]` 。
- 而 `arr = [3,2,1]` 的下一个排列是 `[1,2,3]` ，因为 `[3,2,1]` 不存在一个字典序更大的排列。

给你一个整数数组 `nums` ，找出 `nums` 的下一个排列。

必须 原地 修改，只允许使用额外常数空间。

示例 1：

```
输入：nums = [1,2,3]
输出：[1,3,2]
```

示例 2：

```
输入：nums = [3,2,1]
输出：[1,2,3]
```

```python
class Solution:
    def nextPermutation(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        def reverse(nums, i, j):
            while i < j:
                nums[i], nums[j] = nums[j], nums[i]
                i += 1
                j -= 1
```

```
        n = len(nums)
        first_idx = -1
        for i in range(n - 2, -1, -1):
            if nums[i] < nums[i + 1]:  # 如果要求上一个排列，这里改成>
                first_idx = i
                break
        if first_idx == -1:
            reverse(nums, 0, n - 1)
            return
        second_idx = -1
        for i in range(n-1, -1, -1):
            if nums[i] > nums[first_idx]:   # 如果要求上一个排列，这里改成<
                second_idx = i
                break
        nums[first_idx], nums[second_idx] = nums[second_idx], nums[first_idx]
        reverse(nums, first_idx + 1, n -1)
```

33. 搜索旋转排序数组

难度 中等 1812 收藏 分享 切换为英文 接收动态 反馈

整数数组 `nums` 按升序排列，数组中的值 **互不相同** 。

在传递给函数之前，`nums` 在预先未知的某个下标 `k`（`0 <= k < nums.length`）上进行了 **旋转**，使数组变为 `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]`（下标 **从 0 开始** 计数）。例如，`[0,1,2,4,5,6,7]` 在下标 `3` 处经旋转后可能变为 `[4,5,6,7,0,1,2]` 。

给你 **旋转后** 的数组 `nums` 和一个整数 `target` ，如果 `nums` 中存在这个目标值 `target` ，则返回它的下标，否则返回 `-1` 。

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
```

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        # 将数组一分为二，其中一定有一个是有序的，另一个可能是有序，也能是部分有序。此时有序部分用二分法查找。无序部分再一分为二，其中一个一定有序，另一个可能有序，可能无序。就这样循环.
        left, right = 0, len(nums) - 1
        while left <= right:
            mid = left + (right - left) // 2
            if nums[mid] == target: return mid
            if nums[mid] < nums[-1]:
                if nums[mid] < target <= nums[-1]:
                    left = mid + 1
                else:
                    right = mid - 1
            else:
                if nums[left] <= target < nums[mid]:
                    right = mid - 1
```

```
                else:
                    left = 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☆

难度 中等 1431 收藏 分享 切换为英文 接收动态 反馈

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。

如果数组中不存在目标值 `target`，返回 `[-1, -1]`。

进阶：

- 你可以设计并实现时间复杂度为 `O(log n)` 的算法解决此问题吗？

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]
```

示例 3：

```
输入：nums = [], target = 0
输出：[-1,-1]
```

```python
class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:
        """
        口诀
        求左边界：向下取整，等号归右，左加一
        求右边界：向上取整，等号归左，右减一
        总是右侧为所求
        """
        if not nums: return [-1, -1]
        left, right = 0, len(nums) - 1  # 右边界
        while left < right:
            mid = left + (right - left) // 2  # 向下取整
            if nums[mid] == target:
                right = mid  # 等号归右
            elif nums[mid] < target:
                left = mid + 1  # 左加一
            else:
                right = mid
        if nums[right] != target: return [-1, -1]
        begin = right  # 右侧为所求
        left, right = 0, len(nums) - 1  # 右边界
        while left < right:
            mid = left + (right - left + 1) // 2  # 向上取整
            if nums[mid] == target:
                left = mid
            elif nums[mid] < target:
                left = mid
```

```
            else:
                right = mid - 1
        end = right  # 右侧为所求
        return [begin, end]
```

39. 组合总和

难度 中等 1736 收藏 分享 切换为英文 接收动态 反馈

给你一个 **无重复元素** 的整数数组 `candidates` 和一个目标整数 `target` ，找出 `candidates` 中可以使数字和为目标数 `target` 的 *所有* **不同组合** ，并以列表形式返回。你可以按 **任意顺序** 返回这些组合。

`candidates` 中的 **同一个** 数字可以 **无限制重复被选取** 。如果至少一个数字的被选数量不同，则两种组合是不同的。

对于给定的输入，保证和为 `target` 的不同组合数少于 `150` 个。

示例 1：

输入： candidates = [2,3,6,7], target = 7
输出： [[2,2,3],[7]]
解释：
2 和 3 可以形成一组候选，2 + 2 + 3 = 7 。注意 2 可以使用多次。
7 也是一个候选， 7 = 7 。
仅有这两种组合。

示例 2：

输入： candidates = [2,3,5], target = 8
输出： [[2,2,2,2],[2,3,3],[3,5]]

```
class Solution:
    def combinationSum(self, candidates: List[int], target: int) ->
List[List[int]]:
        res = []
        path = []

        def backtrack(path, cur, target):
            if target == 0:
                res.append(path[:])
                return
            if target < 0:
                return
            for i in range(cur, len(candidates)):
                backtrack(path + [candidates[i]], i, target - candidates[i])
        backtrack(path, 0, target)
        return res
```

46. 全排列

难度 中等 1752 收藏 分享 切换为英文 接收动态 反馈

给定一个不含重复数字的数组 `nums` ，返回其 *所有可能的全排列* 。你可以 **按任意顺序** 返回答案。

示例 1：

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

示例 2：

```
输入：nums = [0,1]
输出：[[0,1],[1,0]]
```

示例 3：

```
输入：nums = [1]
输出：[[1]]
```

提示：

- `1 <= nums.length <= 6`
- `-10 <= nums[i] <= 10`
- `nums` 中的所有整数 **互不相同**

```python
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        res = []
        path = []
        visited = [0] * (len(nums))
        def dfs(path):
            if len(path) == len(nums):
                res.append(path[:])
                return
            for i in range(len(nums)):
                if visited[i] == 0:
                    visited[i] = 1
                    dfs(path + [nums[i]])
                    visited[i] = 0
        dfs(path)
        return res
```

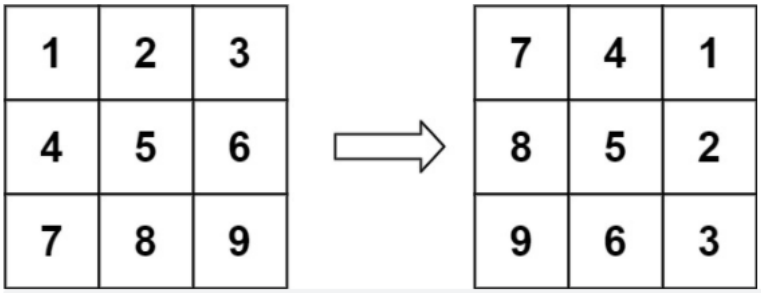
48. 旋转图像☆

难度 中等 1150 收藏 分享 切换为英文 接收动态 反馈

给定一个 $n \times n$ 的二维矩阵 `matrix` 表示一个图像。请你将图像顺时针旋转 90 度。

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：



输入： `matrix = [[1,2,3],[4,5,6],[7,8,9]]`
输出： `[[7,4,1],[8,5,2],[9,6,3]]`

```python
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        n = len(matrix)
        # 先沿对角线翻折
        for i in range(n):
            for j in range(i):
                matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

        # 沿垂直中线翻折
        for i in range(n):
            j, k = 0, n - 1
            while j < k:
                matrix[i][j], matrix[i][k] = matrix[i][k], matrix[i][j]
                j += 1
                k -= 1
```

49. 字母异位词分组 ☆

难度 中等 984 收藏 分享 切换为英文 接收动态 反馈

给你一个字符串数组，请你将 **字母异位词** 组合在一起。可以按任意顺序返回结果列表。

字母异位词 是由重新排列源单词的字母得到的一个新单词，所有源单词中的字母通常恰好只用一次。

示例 1:

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

示例 2:

```
输入: strs = [""]
输出: [[""]]
```

示例 3:

```
输入: strs = ["a"]
输出: [["a"]]
```

提示:

- `1 <= strs.length <= 10^4`
- `0 <= strs[i].length <= 100`
- `strs[i]` 仅包含小写字母

```python
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        dic = {}
        for s in strs:
            chars = [0] * 26
            for c in s:
                chars[ord(c) - ord('a')] += 1
            k = tuple(chars)
            if k not in dic:
                dic[k] = [s]
            else:
                dic[k].append(s)
        return list(dic.values())
```

55. 跳跃游戏 ☆

难度 中等 1622 收藏 分享 切换为英文 接收动态 反馈

给定一个非负整数数组 `nums` ，你最初位于数组的 **第一个下标** 。

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下标。

示例 1：

```
输入：nums = [2,3,1,1,4]
输出：true
解释：可以先跳 1 步，从下标 0 到达下标 1，然后再从下标 1 跳 3 步到达最后一个下标。
```

示例 2：

```
输入：nums = [3,2,1,0,4]
输出：false
解释：无论怎样，总会到达下标为 3 的位置。但该下标的最大跳跃长度是 0 ， 所以永远不可能到达最后一个下标。
```

提示：

- `1 <= nums.length <= 3 * 10^4`
- `0 <= nums[i] <= 10^5`

```python
# 贪心法。  dfs会超时
class Solution:
    def canJump(self, nums) :
        max_i = 0        #初始化当前能到达最远的位置
        for i, jump in enumerate(nums):   #i为当前位置，jump是当前位置的跳数
            if max_i>=i and i+jump>max_i:  #如果当前位置能到达，并且当前位置+跳数>最远位置
                max_i = i+jump  #更新最远能到达位置
        return max_i>=i
```

56. 合并区间

难度 中等 1290 收藏 分享 切换为英文 接收动态 反馈

以数组 `intervals` 表示若干个区间的集合，其中单个区间为 `intervals[i] = [start_i, end_i]`。请你合并所有重叠的区间，并返回 *一个不重叠的区间数组，该数组需恰好覆盖输入中的所有区间*。

示例 1：

> **输入：** intervals = [[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：** intervals = [[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。

提示：

- `1 <= intervals.length <= 10^4`
- `intervals[i].length == 2`
- `0 <= start_i <= end_i <= 10^4`

```python
# [[1,3],[2,6],[8,10],[15,18]]
# [[1,4],[2,3]]
class Solution:
    def merge(self, intervals: List[List[int]]) -> List[List[int]]:
        intervals.sort(key=lambda x: x[0])
        res = [intervals[0]]
        for i in range(1, len(intervals)):
            if intervals[i][0] <= res[-1][1]:
                pre = res.pop()
                res.append([pre[0], max(pre[1], intervals[i][1])])
            else:
                res.append(intervals[i])
        return res
```

62. 不同路径

难度 中等 1265 收藏 分享 切换为英文 接收动态 反馈

一个机器人位于一个 `m x n` 网格的左上角 （起始点在下图中标记为 “Start” ）。

机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为 “Finish” ）。

问总共有多少条不同的路径?

示例 1:

输入： `m = 3, n = 7`
输出： 28

```
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        dp = [[0] * n for _ in range(m)]
        for j in range(n):
            dp[0][j] = 1
        for i in range(m):
            dp[i][0] = 1
        for i in range(1, m):
            for j in range(1, n):
                dp[i][j] = dp[i-1][j] + dp[i][j-1]
        return dp[-1][-1]
```

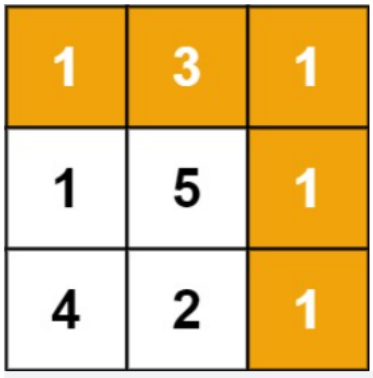
64. 最小路径和

难度 中等 1129 收藏 分享 切换为英文 接收动态 反馈

给定一个包含非负整数的 `m x n` 网格 `grid` ，请找出一条从左上角到右下角的路径，使得路径上的数字总和为最小。

说明： 每次只能向下或者向右移动一步。

示例 1:



输入： `grid = [[1,3,1],[1,5,1],[4,2,1]]`
输出： 7
解释： 因为路径 `1→3→1→1→1` 的总和最小。

```python
class Solution:
    def minPathSum(self, grid: List[List[int]]) -> int:
        m, n = len(grid), len(grid[0])
        dp = [[0] * (n) for _ in range(m)]
        s = 0
        for j in range(n):
            s += grid[0][j]
            dp[0][j] = s
        s = 0
        for i in range(m):
            s += grid[i][0]
            dp[i][0] = s
        for i in range(1, m):
            for j in range(1, n):
                dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j]
        return dp[-1][-1]
```

75. 颜色分类

难度 中等 1148 收藏 分享 切换为英文 接收动态 反馈

给定一个包含红色、白色和蓝色、共 `n` 个元素的数组 `nums` ，原地对它们进行排序，使得相同颜色的元素相邻，并按照红色、白色、蓝色顺序排列。

我们使用整数 `0` 、 `1` 和 `2` 分别表示红色、白色和蓝色。

必须在不使用库的sort函数的情况下解决这个问题。

示例 1：

```
输入：nums = [2,0,2,1,1,0]
输出：[0,0,1,1,2,2]
```

示例 2：

```
输入：nums = [2,0,1]
输出：[0,1,2]
```

提示：

- `n == nums.length`
- `1 <= n <= 300`
- `nums[i]` 为 `0` 、 `1` 或 `2`

进阶：

- 你可以不使用代码库中的排序函数来解决这道题吗？
- 你能想出一个仅使用常数空间的一趟扫描算法吗？

```python
class Solution:
    def sortColors(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        cnt = [0] * 3
        for c in nums:
            cnt[c] += 1
        c, n = 0, cnt[0]
        for i in range(len(nums)):
```

```
            while n == 0:
                c += 1
                n = cnt[c]
            nums[i] = c
            n -= 1
```

78. 子集

难度 中等 1468 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 `nums` ，数组中的元素 **互不相同** 。返回该数组所有可能的子集（幂集）。

解集 **不能** 包含重复的子集。你可以按 **任意顺序** 返回解集。

示例 1：

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

示例 2：

```
输入：nums = [0]
输出：[[],[0]]
```

提示：

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`
- `nums` 中的所有元素 **互不相同**

```
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        res = []
        path = []

        def dfs(path, cur):
            res.append(path[:])
            for i in range(cur, len(nums)):
                dfs(path + [nums[i]], i + 1)
        dfs(path, 0)
        return res
```

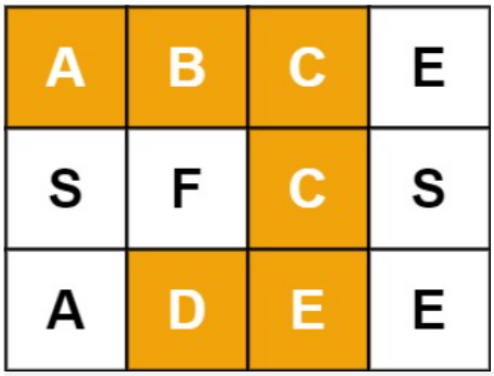
79. 单词搜索 ☆

难度 中等 1169 收藏 分享 切换为英文 接收动态 反馈

给定一个 `m x n` 二维字符网格 `board` 和一个字符串单词 `word` 。如果 `word` 存在于网格中，返回 `true` ；否则，返回 `false` 。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中"相邻"单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

示例 1：



输入： board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"
输出： true

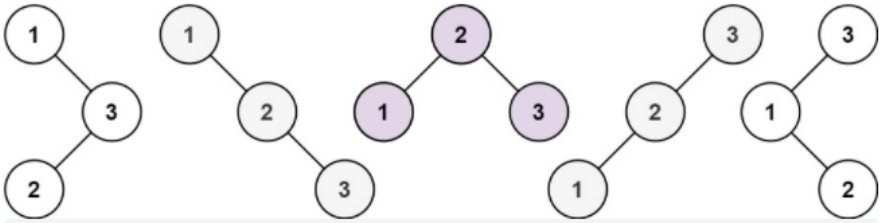
```python
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        m, n = len(board), len(board[0])
        visited = [[False] * n for _ in range(m)]
        path = ""
        def dfs(i, j, path, idx):
            if path == word:
                return True
            visited[i][j] = True
            for x, y in ((i, j+1), (i, j-1), (i-1, j), (i+1, j)):
                if x < 0 or x >= m or y < 0 or y >= n:
                    continue
                if visited[x][y]:
                    continue
                if board[x][y] == word[idx + 1]:
                    if dfs(x, y, path + board[x][y], idx+1):
                        return True
            visited[i][j] = False
            return False
        for i in range(m):
            for j in range(n):
                if board[i][j] == word[0]:
                    if dfs(i, j, path + board[i][j], 0):
                        return True
        return False
```

96. 不同的二叉搜索树☆

难度 中等 1525 收藏 分享 切换为英文 接收动态 反馈

给你一个整数 `n` ，求恰由 `n` 个节点组成且节点值从 `1` 到 `n` 互不相同的 **二叉搜索树** 有多少种？返回满足题意的二叉搜索树的种数。

示例 1：




```
输入：n = 3
输出：5
```

示例 2：

```
输入：n = 1
输出：1
```

提示：

- `1 <= n <= 19`

```
class Solution:
    def numTrees(self, n: int) -> int:
        dp = [0] * (n + 1)
        dp[0] = 1
        dp[1] = 1
        for i in range(2, n + 1):
            for j in range(i):
                dp[i] += dp[j] * dp[i - j - 1]
        return dp[n]
```

98. 验证二叉搜索树

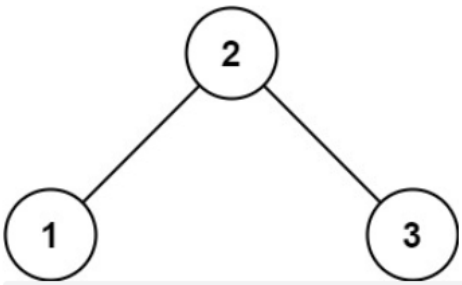
难度 中等 1404 收藏 分享 切换为英文 接收动态 反馈

给你一个二叉树的根节点 `root` ，判断其是否是一个有效的二叉搜索树。

有效 二叉搜索树定义如下：

- 节点的左子树只包含 **小于** 当前节点的数。
- 节点的右子树只包含 **大于** 当前节点的数。
- 所有左子树和右子树自身必须也是二叉搜索树。

示例 1：




```
输入：root = [2,1,3]
输出：true
```

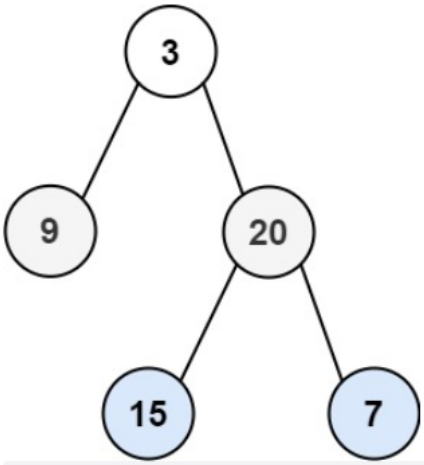
```python
class Solution:
    def isValidBST(self, root: TreeNode) -> bool:
        self.cur_max = -float('inf')
        def helper(root):
            if not root:
                return True
            if not helper(root.left):
                return False
            if self.cur_max < root.val:
                self.cur_max = root.val
            else:
                return False
            if not helper(root.right):
                return False
            return True
        return helper(root)
```

102. 二叉树的层序遍历

难度 中等 1168 收藏 分享 切换为英文 接收动态 反馈

给你二叉树的根节点 `root` ，返回其节点值的 **层序遍历** 。 （即逐层地，从左到右访问所有节点）。

示例 1：



输入： root = [3,9,20,null,null,15,7]
输出： [[3],[9,20],[15,7]]

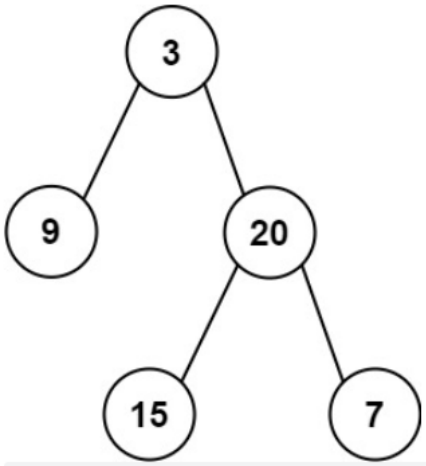
```python
class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root: return []
        q = [root]
        res = []
        while q:
            tmp = []
            for _ in range(len(q)):
                node = q.pop(0)
                tmp.append(node.val)
                if node.left:
                    q.append(node.left)
                if node.right:
                    q.append(node.right)
            res.append(tmp)
        return res
```

105. 从前序与中序遍历序列构造二叉树

难度 中等 1407 收藏 分享 切换为英文 接收动态 反馈

给定两个整数数组 `preorder` 和 `inorder` ，其中 `preorder` 是二叉树的**先序遍历**， `inorder` 是同一棵树的**中序遍历**，请构造二叉树并返回其根节点。

示例 1:




```
输入：preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
输出：[3,9,20,null,null,15,7]
```

```python
class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:
        if not preorder:
            return None
        root_val = preorder[0]
        root = TreeNode(root_val)
        idx = inorder.index(root_val)
        left_inorder = inorder[:idx]
        right_inorder = inorder[idx+1:]
        left_preorder = preorder[1: 1 + len(left_inorder)]
        right_preorder = preorder[1 + len(left_inorder): ]
        root.left = self.buildTree(left_preorder, left_inorder)
        root.right = self.buildTree(right_preorder, right_inorder)
        return root
```

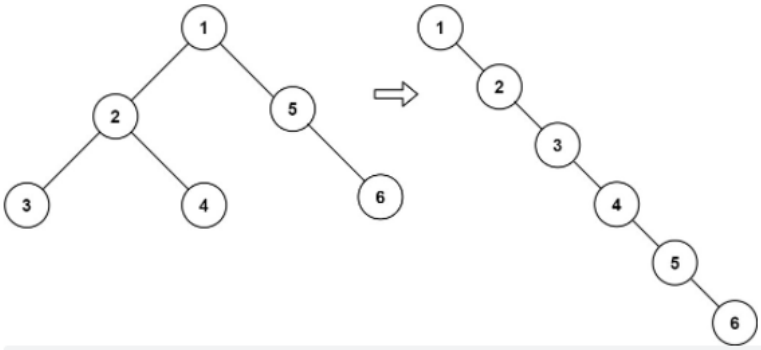
114. 二叉树展开为链表 ☆☆☆

难度 中等 1056 收藏 分享 切换为英文 接收动态 反馈

给你二叉树的根结点 `root` ，请你将它展开为一个单链表：

- 展开后的单链表应该同样使用 `TreeNode` ，其中 `right` 子指针指向链表中下一个结点，而左子指针始终为 `null` 。
- 展开后的单链表应该与二叉树 先序遍历 顺序相同。

示例 1：



输入： root = [1,2,5,3,4,null,6]
输出： [1,null,2,null,3,null,4,null,5,null,6]

```python
class Solution:
    def flatten(self, root: TreeNode) -> None:
        """
        Do not return anything, modify root in-place instead.
        """
        def helper(root):
            if not root: return None
            left = root.left
            right = root.right

            helper(left)
            helper(right)

            root.right = left
            root.left = None
            # root 移到left的末尾节点
            while root.right:
                root = root.right
            root.right = right

        helper(root)
```

128. 最长连续序列 ☆☆☆☆

【图解】遇到就深究——并查集 - 最长连续序列 - 力扣（LeetCode） (leetcode-cn.com)

难度 中等 1088 收藏 分享 切换为英文 接收动态 反馈

给定一个未排序的整数数组 `nums` ，找出数字连续的最长序列（不要求序列元素在原数组中连续）的长度。

请你设计并实现时间复杂度为 `O(n)` 的算法解决此问题。

示例 1：

输入： nums = [100,4,200,1,3,2]
输出： 4
解释： 最长数字连续序列是 [1, 2, 3, 4]。它的长度为 4。

示例 2：

输入： nums = [0,3,7,2,5,8,4,6,0,1]
输出： 9

提示：

- `0 <= nums.length <= 10^5`
- `-10^9 <= nums[i] <= 10^9`

```python
class UFS:
    def __init__(self, nums):
        self.parent = {num: num for num in nums}
        self.cnt = {num: 1 for num in nums}

    def find(self, x):
        while x != self.parent[x]:
            self.parent[x] = self.parent[self.parent[x]]
            x = self.parent[x]
        return x

    def union(self, x, y):
        if y not in self.parent:
            return 1
        root1, root2 = self.find(x), self.find(y)
        if root1 == root2:
            return self.cnt[root1]
        self.parent[root2] = root1
        self.cnt[root1] += self.cnt[root2]
        return self.cnt[root1]

class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:
        if len(nums) == 0: return 0
        ufs = UFS(nums)
        res = 1
        for num in nums:
            cnt = ufs.union(num, num + 1)
            res = max(res, cnt)
        return res
```

139. 单词拆分

难度 中等 1382 收藏 分享 切换为英文 接收动态 反馈

给你一个字符串 `s` 和一个字符串列表 `wordDict` 作为字典。请你判断是否可以利用字典中出现的单词拼接出 `s` 。

注意：不要求字典中出现的单词全部都使用，并且字典中的单词可以重复使用。

示例 1：

```
输入：s = "leetcode", wordDict = ["leet", "code"]
输出：true
解释：返回 true 因为 "leetcode" 可以由 "leet" 和 "code" 拼接成。
```

示例 2：

```
输入：s = "applepenapple", wordDict = ["apple", "pen"]
输出：true
解释：返回 true 因为 "applepenapple" 可以由 "apple" "pen" "apple" 拼接成。
     注意，你可以重复使用字典中的单词。
```

示例 3：

```
输入：s = "catsandog", wordDict = ["cats", "dog", "sand", "and", "cat"]
输出：false
```

```python
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        n = len(s)
        dp = [False] * (n + 1)
        dp[0] = True
        for i in range(1, n+1):
            for j in range(len(wordDict)):
                if len(wordDict[j]) > i:
                    continue
                dp[i] = dp[i] or (dp[i - len(wordDict[j])] and s[i -
len(wordDict[j]): i] == wordDict[j])
        return dp[-1]
```

142. 环形链表 II

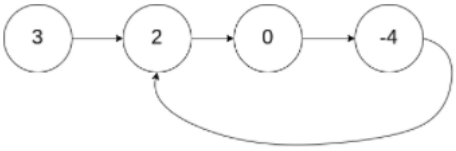
难度 中等 1359 收藏 分享 切换为英文 接收动态 反馈

给定一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 `null`。

如果链表中有某个节点，可以通过连续跟踪 `next` 指针再次到达，则链表中存在环。 为了表示给定链表中的环，评测系统内部使用整数 `pos` 来表示链表尾连接到链表中的位置（索引从 0 开始）。如果 `pos` 是 `-1`，则在该链表中没有环。**注意：`pos` 不作为参数进行传递**，仅仅是为了标识链表的实际情况。

不允许修改 链表。

示例 1：



输入： head = [3,2,0,-4], pos = 1
输出： 返回索引为 1 的链表节点
解释： 链表中有一个环，其尾部连接到第二个节点。

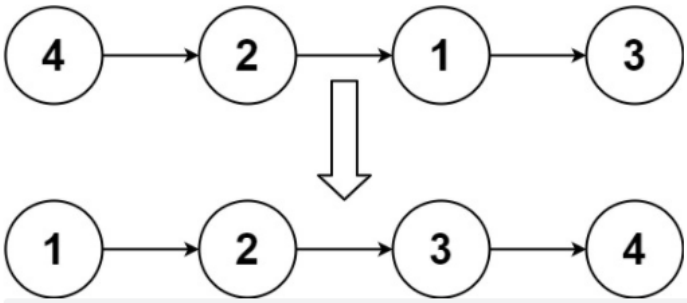
```python
class Solution:
    def detectCycle(self, head: ListNode) -> ListNode:
        fast, slow = head, head
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
            if fast == slow:
                p, q = fast, head
                while p != q:
                    p = p.next
                    q = q.next
                return p
        return None
```

148. 排序链表

难度 中等 1453 收藏 分享 切换为英文 接收动态 反馈

给你链表的头结点 `head` ，请将其按 **升序** 排列并返回 **排序后的链表** 。

示例 1：



输入： head = [4,2,1,3]
输出： [1,2,3,4]

```python
class Solution:
    def sortList(self, head: ListNode) -> ListNode:
        def helper(head):
            if not head or not head.next:
```

```
            return head
        dummy = ListNode()
        dummy.next = head
        fast, slow = dummy, dummy
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
        right_head = slow.next
        slow.next = None
        left = helper(dummy.next)
        right = helper(right_head)
        res = p = ListNode()
        while left and right:
            if left.val < right.val:
                p.next = left
                left = left.next
            else:
                p.next = right
                right = right.next
            p = p.next
        if left: p.next = left
        if right: p.next = right
        return res.next
    return helper(head)
```

152. 乘积最大子数组☆☆☆

难度 中等 1478 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 `nums` ，请你找出数组中乘积最大的连续子数组（该子数组中至少包含一个数字），并返回该子数组所对应的乘积。

示例 1:

输入： [2,3,-2,4]
输出： 6
解释： 子数组 [2,3] 有最大乘积 6。

示例 2:

输入： [-2,0,-1]
输出： 0
解释： 结果不能为 2, 因为 [-2,-1] 不是子数组。

```
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        n = len(nums)
        dp_max = [0] * n
        dp_max[0] = nums[0]
        dp_min = [0] * n
        dp_min[0] = nums[0]
        res = nums[0]

        for i in range(1, n):
```

```
            dp_max[i] = max(nums[i], dp_max[i-1] * nums[i], dp_min[i-1] *
nums[i])
            dp_min[i] = min(nums[i], dp_max[i-1] * nums[i], dp_min[i-1] *
nums[i])
            res = max(res, dp_max[i])
        return res
```

198. 打家劫舍

难度 中等 1869 收藏 分享 切换为英文 接收动态 反馈

你是一个专业的小偷，计划偷窃沿街的房屋。每间房内都藏有一定的现金，影响你偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统，**如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。**

给定一个代表每个房屋存放金额的非负整数数组，计算你 **不触动警报装置的情况下** ，一夜之内能够偷窃到的最高金额。

示例 1：

```
输入：[1,2,3,1]
输出：4
解释：偷窃 1 号房屋 (金额 = 1) ，然后偷窃 3 号房屋 (金额 = 3)。
     偷窃到的最高金额 = 1 + 3 = 4 。
```

示例 2：

```
输入：[2,7,9,3,1]
输出：12
解释：偷窃 1 号房屋 (金额 = 2), 偷窃 3 号房屋 (金额 = 9)，接着偷窃 5 号房屋
(金额 = 1)。
     偷窃到的最高金额 = 2 + 9 + 1 = 12 。
```

提示：

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 400`

```
class Solution:
    def rob(self, nums: List[int]) -> int:
        n = len(nums)
        if n <= 1: return nums[0]
        dp = [0] * n
        dp[0] = nums[0]
        dp[1] = max(nums[0], nums[1])
        for i in range(2, n):
            dp[i] = max(dp[i-2] + nums[i], dp[i-1])
        return dp[-1]
```

200. 岛屿数量

难度 中等 1532 收藏 分享 切换为英文 接收动态 反馈

给你一个由 '1'（陆地）和 '0'（水）组成的的二维网格，请你计算网格中岛屿的数量。

岛屿总是被水包围，并且每座岛屿只能由水平方向和/或竖直方向上相邻的陆地连接形成。

此外，你可以假设该网格的四条边均被水包围。

示例 1：

```
输入：grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
输出：1
```

```python
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        self.res = 0
        m, n = len(grid), len(grid[0])
        def dfs(i, j):
            if grid[i][j] == '#':
                return
            grid[i][j] = '#'
            for x, y in ((i, j-1), (i, j+1), (i+1, j), (i-1, j)):
                if x < 0 or x >= m or y < 0 or y >= n:
                    continue
                if grid[x][y] == '0':
                    continue
                dfs(x, y)

        for i in range(m):
            for j in range(n):
                if grid[i][j] == '1':
                    self.res += 1
                    dfs(i, j)
        return self.res
```

207. 课程表

难度 中等 1110 收藏 分享 切换为英文 接收动态 反馈

你这个学期必须选修 `numCourses` 门课程，记为 `0` 到 `numCourses - 1` 。

在选修某些课程之前需要一些先修课程。 先修课程按数组 `prerequisites` 给出，其中 `prerequisites[i] = [$a_i$, $b_i$]` ，表示如果要学习课程 `$a_i$` 则 **必须** 先学习课程 `$b_i$` 。

- 例如，先修课程对 `[0, 1]` 表示：想要学习课程 `0` ，你需要先完成课程 `1` 。

请你判断是否可能完成所有课程的学习？如果可以，返回 `true` ；否则，返回 `false` 。

示例 1：

```
输入：numCourses = 2, prerequisites = [[1,0]]
输出：true
解释：总共有 2 门课程。学习课程 1 之前，你需要完成课程 0 。这是可能的。
```

示例 2：

```
输入：numCourses = 2, prerequisites = [[1,0],[0,1]]
输出：false
解释：总共有 2 门课程。学习课程 1 之前，你需要先完成课程 0 ；并且学习课程 0 之前，你还应先完成课程 1 。这是不可能的。
```

提示：

- `1 <= numCourses <= $10^5$`
- `0 <= prerequisites.length <= 5000`
- `prerequisites[i].length == 2`
- `0 <= $a_i$, $b_i$ < numCourses`
- `prerequisites[i]` 中的所有课程对 **互不相同**

```python
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) ->
bool:
        graph = [[] for _ in range(numCourses)]
        in_degree = [0] * numCourses
        for pre in prerequisites:
            graph[pre[1]].append(pre[0])
            in_degree[pre[0]] += 1
        q = []
        for i in range(numCourses):
            if in_degree[i] == 0:
                q.append(i)
        while q:
            node = q.pop(0)
            for nxt in graph[node]:
                in_degree[nxt] -= 1
                if in_degree[nxt] == 0:  # ***入度为0才加入到队列中
                    q.append(nxt)
        return all(in_degree[i] == 0 for i in range(numCourses))
```

208. 实现 Trie (前缀树)☆

难度 中等 1051 收藏 分享 切换为英文 接收动态 反馈

Trie（发音类似 "try"）或者说 **前缀树** 是一种树形数据结构，用于高效地存储和检索字符串数据集中的键。这一数据结构有相当多的应用情景，例如自动补完和拼写检查。

请你实现 Trie 类：

- `Trie()` 初始化前缀树对象。
- `void insert(String word)` 向前缀树中插入字符串 `word` 。
- `boolean search(String word)` 如果字符串 `word` 在前缀树中，返回 `true` （即，在检索之前已经插入）；否则，返回 `false` 。
- `boolean startsWith(String prefix)` 如果之前已经插入的字符串 `word` 的前缀之一为 `prefix` ，返回 `true` ；否则，返回 `false` 。

示例：

```
输入
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
输出
[null, null, true, false, true, null, true]

解释
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // 返回 True
trie.search("app");     // 返回 False
trie.startsWith("app"); // 返回 True
trie.insert("app");
trie.search("app");     // 返回 True
```

```python
class Node:
    def __init__(self):
        self.is_word = False
        self.child = {}

class Trie:

    def __init__(self):
        self.root = Node()

    def insert(self, word: str) -> None:
        node = self.root
        for c in word:
            if c not in node.child:
                node.child[c] = Node()
            node = node.child[c]
        node.is_word = True

    def search(self, word: str) -> bool:
        node = self.root
        for c in word:
            if c not in node.child: return False
            else: node = node.child[c]
        return node.is_word

    def startsWith(self, prefix: str) -> bool:
        node = self.root
        for c in prefix:
            if c not in node.child: return False
            node = node.child[c]
```

```
        return True


# Your Trie object will be instantiated and called as such:
# obj = Trie()
# obj.insert(word)
# param_2 = obj.search(word)
# param_3 = obj.startsWith(prefix)
```

215. 数组中的第K个最大元素 ☆☆☆☆

难度 中等 1476 收藏 分享 切换为英文 接收动态 反馈

给定整数数组 `nums` 和整数 `k`，请返回数组中第 `k` 个最大的元素。

请注意，你需要找的是数组排序后的第 `k` 个最大的元素，而不是第 `k` 个不同的元素。

示例 1:

```
输入: [3,2,1,5,6,4] 和 k = 2
输出: 5
```

示例 2:

```
输入: [3,2,3,1,2,4,5,5,6] 和 k = 4
输出: 4
```

提示:

- `1 <= k <= nums.length <= $10^4$`
- `$-10^4$ <= nums[i] <= $10^4$`

```
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        heap = []  #  小根堆
        for num in nums:
            if len(heap) < k:  # 不足k个 直接进堆
                heapq.heappush(heap, num)
            else:
                # 数量大于等于k个时，当num大于堆顶元素，才需要进堆
                if num > heap[0]:
                    heapq.heappop(heap)
                    heapq.heappush(heap, num)
        return heap[0]
```

```
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        def check(mid):
            cnt = 0
            for n in nums:
                if n >= mid:
                    cnt += 1
            return cnt

        left, right = -int(1e4), int(1e4)
        while left <= right:
            mid = left + (right - left) // 2
```

```
            cnt = check(mid)
            if cnt >= k:
                left = mid + 1
            else:
                right = mid - 1
        return right
```

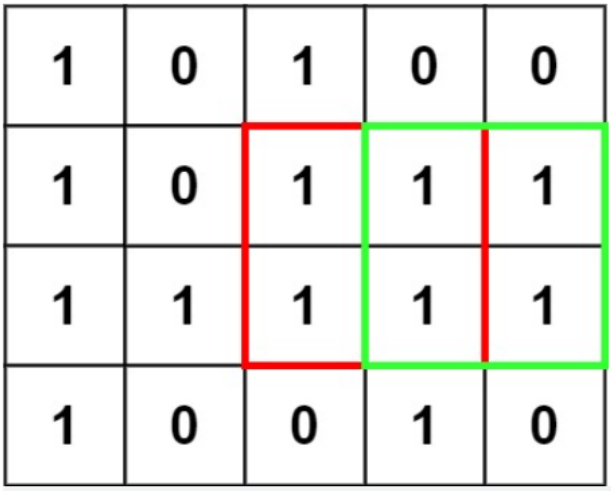
```
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        def randomized_partition(nums, left, right):
            index = random.randint(left, right)
            pivot = nums[index]
            nums[index], nums[left] = nums[left], nums[index]
            while left < right:
                while left < right and nums[right] >= pivot:
                    right -= 1
                nums[left] = nums[right]
                while left < right and nums[left] <= pivot:
                    left += 1
                nums[right] = nums[left]
            nums[left] = pivot
            return left

        def topk_split(nums, left, right, k):
            if left >= right:
                return
            # nums index左边是前k个小的数，index右边是n-k个大的数
            index = randomized_partition(nums, left, right)

            topk_split(nums, index + 1, right, k)
            topk_split(nums, left, index - 1, k)

        # 题目要第k大的数，我们转换为第len(nums)-k小的数
        topk_split(nums, 0, len(nums)-1, len(nums)-k)
        return nums[len(nums) - k]
```

221. 最大正方形☆

难度 中等 1019 收藏 分享 切换为英文 接收动态 反馈

在一个由 '0' 和 '1' 组成的二维矩阵内，找到只包含 '1' 的最大正方形，并返回其面积。

示例 1：



输入： matrix = [["1","0","1","0","0"],["1","0","1","1","1"],
["1","1","1","1","1"],["1","0","0","1","0"]]
输出： 4

```python
class Solution:
    def maximalSquare(self, matrix: List[List[str]]) -> int:
        rows = len(matrix)
        if rows < 1: return 0
        cols = len(matrix[0])
        dp = [[0] * cols for _ in range(rows)]
        maxside = 0
        for i in range(rows):
            for j in range(cols):
                if matrix[i][j] == '1':
                    if i == 0 or j == 0:
                        dp[i][j] = 1
                    else:
                        dp[i][j] = min(dp[i-1][j-1], dp[i-1][j], dp[i][j-1]) + 1

                    maxside = max(maxside, dp[i][j])
        return maxside * maxside
```

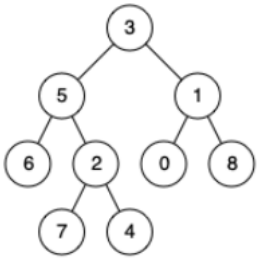
236. 二叉树的最近公共祖先

难度 中等 1526 收藏 分享 切换为英文 接收动态 反馈

给定一个二叉树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个节点 p、q，最近公共祖先表示为一个节点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（**一个节点也可以是它自己的祖先**）。”

示例 1：



输入： root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
输出： 3
解释： 节点 5 和节点 1 的最近公共祖先是节点 3 。

```
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q:
'TreeNode') -> 'TreeNode':
        def helper(root, p, q):
            if not root:
                return None
            if root == p or root == q:
                return root
            left = helper(root.left, p, q)
            right = helper(root.right, p, q)
            if left and right:
                return root
            if not left:
                return right
            if not right:
                return left
        return helper(root, p, q)
```

238. 除自身以外数组的乘积

难度 中等 1052 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 `nums`，返回 *数组 `answer`，其中 `answer[i]` 等于 `nums` 中除 `nums[i]` 之外其余各元素的乘积*。

题目数据 **保证** 数组 `nums` 之中任意元素的全部前缀元素和后缀的乘积都在 **32 位** 整数范围内。

请**不要使用除法**，且在 `O(n)` 时间复杂度内完成此题。

示例 1:

```
输入: nums = [1,2,3,4]
输出: [24,12,8,6]
```

示例 2:

```
输入: nums = [-1,1,0,-3,3]
输出: [0,0,9,0,0]
```

提示:

- `2 <= nums.length <= 10^5`
- `-30 <= nums[i] <= 30`
- **保证** 数组 `nums` 之中任意元素的全部前缀元素和后缀的乘积都在 **32 位** 整数范围内

```python
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        # nums   [1 2 3 4]
        # prefix [1 1 2 6]
        # suffix [24 12 4 1]
        # 一个数的除自己以外其他数的乘积，等于该数前面所有数的乘积 x 该数后面所有数的乘积

        prefix = [1]
        for i in range(1, len(nums)):
            prefix.append(prefix[-1] * nums[i-1])
        suffix = [1]
        for i in range(len(nums)-1, 0, -1):
            suffix.append(suffix[-1] * nums[i])
        output = [0] * len(nums)
        for i in range(len(nums)):
            output[i] = prefix[i] * suffix[len(nums) -i -1]
        return output
```

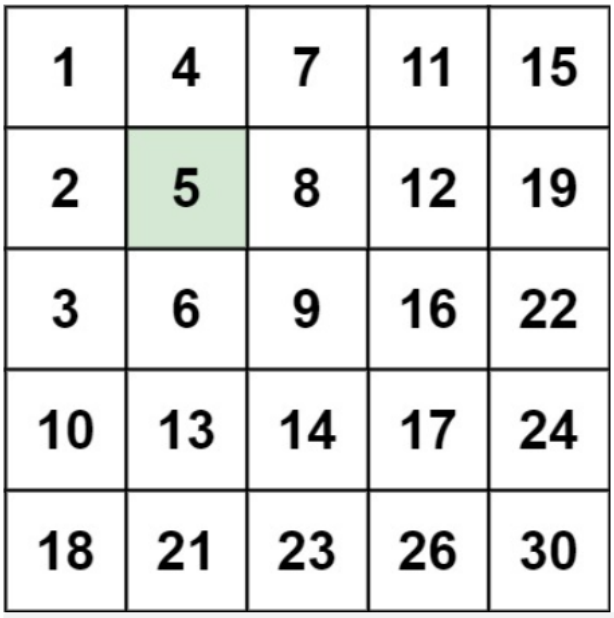
240. 搜索二维矩阵 II

难度 中等 922 收藏 分享 切换为英文 接收动态 反馈

编写一个高效的算法来搜索 $m \times n$ 矩阵 `matrix` 中的一个目标值 `target` 。该矩阵具有以下特性：

- 每行的元素从左到右升序排列。
- 每列的元素从上到下升序排列。

示例 1：




```
输入：matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],
[18,21,23,26,30]], target = 5
```

```python
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        m, n = len(matrix), len(matrix[0])
        x, y = m-1, 0
        while x >= 0 and y < n:
            if matrix[x][y] == target:
                return True
            elif matrix[x][y] > target:
                x -= 1
            else:
                y += 1
        return False
```

279. 完全平方数

难度 中等 1232 收藏 分享 切换为英文 接收动态 反馈

给你一个整数 `n` ，返回 *和为 `n` 的完全平方数的最少数量* 。

完全平方数 是一个整数，其值等于另一个整数的平方；换句话说，其值等于一个整数自乘的积。例如，`1` 、`4` 、`9` 和 `16` 都是完全平方数，而 `3` 和 `11` 不是。

示例 1：

```
输入：n = 12
输出：3
解释：12 = 4 + 4 + 4
```

示例 2：

```
输入：n = 13
输出：2
解释：13 = 4 + 9
```

提示：

- $1 <= n <= 10^4$

```python
class Solution:
    def numSquares(self, n: int) -> int:
        dp = [n] * (n + 1)
        dp[0] = 0
        dp[1] = 1
        for i in range(1, n + 1):
            s = i * i
            for j in range(s, n + 1):
                dp[j] = min(dp[j], dp[j-s] + 1)
        return dp[-1]
```

287. 寻找重复数

难度 中等 1569 收藏 分享 切换为英文 接收动态 反馈

给定一个包含 `n + 1` 个整数的数组 `nums` ，其数字都在 `[1, n]` 范围内（包括 `1` 和 `n`），可知至少存在一个重复的整数。

假设 `nums` 只有 **一个重复的整数** ，返回 **这个重复的数** 。

你设计的解决方案必须 **不修改** 数组 `nums` 且只用常量级 `O(1)` 的额外空间。

示例 1：

```
输入：nums = [1,3,4,2,2]
输出：2
```

示例 2：

```
输入：nums = [3,1,3,4,2]
输出：3
```

提示：

- $1 <= n <= 10^5$
- `nums.length == n + 1`
- `1 <= nums[i] <= n`
- `nums` 中 **只有一个整数** 出现 **两次或多次** ，其余整数均只出现 **一次**

```python
class Solution:
    def findDuplicate(self, nums: List[int]) -> int:
        # 环的入口
        """
        [1, 3, 4, 2, 2]  是value 也是next的索引
        [0, 1, 2, 3, 4]  索引 head
        """
        fast, slow = 0, 0
        while True:
            slow = nums[slow]
            fast = nums[nums[fast]]
            if fast == slow:
                break
        fast = 0
        while True:
            slow = nums[slow]
            fast = nums[fast]
            if fast == slow:
                return slow
```

```python
class Solution:
    def findDuplicate(self, nums: List[int]) -> int:
        def check(mid):
            cnt = 0
            for n in nums:
                if n <= mid:
                    cnt += 1
            return cnt

        left, right = 1, len(nums)-1
        while left <= right:
            mid = left + (right - left) // 2
            if check(mid) <= mid:
                left = mid + 1
            else:
                right = mid - 1
        return left
```

300. 最长递增子序列

难度 中等 2201 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 `nums` ，找到其中最长严格递增子序列的长度。

子序列 是由数组派生而来的序列，删除（或不删除）数组中的元素而不改变其余元素的顺序。例如，`[3,6,2,7]` 是数组 `[0,3,1,6,2,2,7]` 的子序列。

示例 1：

```
输入：nums = [10,9,2,5,3,7,101,18]
输出：4
解释：最长递增子序列是 [2,3,7,101]，因此长度为 4 。
```

示例 2：

```
输入：nums = [0,1,0,3,2,3]
输出：4
```

示例 3：

```
输入：nums = [7,7,7,7,7,7,7]
输出：1
```

提示：

- `1 <= nums.length <= 2500`
- `-10^4 <= nums[i] <= 10^4`

```python
class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:
        res = 1
        dp = [1] * len(nums)
        for i in range(1, len(nums)):
            for j in range(i):
                if nums[i] > nums[j]:
                    dp[i] = max(dp[i], dp[j] + 1)
                res = max(res, dp[i])
        return res
```

309. 最佳买卖股票时机含冷冻期 ☆☆☆☆

难度 中等 1052 收藏 分享 切换为英文 接收动态 反馈

给定一个整数数组 `prices`，其中第 `prices[i]` 表示第 `i` 天的股票价格 。

设计一个算法计算出最大利润。在满足以下约束条件下，你可以尽可能地完成更多的交易（多次买卖一支股票）：

- 卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。

注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

示例 1:

```
输入：prices = [1,2,3,0,2]
输出：3
解释：对应的交易状态为：[买入，卖出，冷冻期，买入，卖出]
```

示例 2:

```
输入：prices = [1]
输出：0
```

提示：

- `1 <= prices.length <= 5000`
- `0 <= prices[i] <= 1000`

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        """
        第i天的状态
        持有
        不持有
            处于冷冻期
            不在冷冻期
        """
        dp = [[0] * 3 for _ in range(len(prices))]
        dp[0][0] = -prices[0]
        for i in range(1, len(prices)):
            dp[i][0] = max(dp[i-1][0], dp[i-1][2] - prices[i])
            dp[i][1] = dp[i-1][0] + prices[i]  # dp[i][1]表示第i天结束，处于冷冻期，导致i+1天不能交易
            dp[i][2] = max(dp[i-1][1], dp[i-1][2])
        return max(dp[-1][1], dp[-1][2])
```

322. 零钱兑换

难度 中等 1714 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 `coins` ，表示不同面额的硬币；以及一个整数 `amount` ，表示总金额。

计算并返回可以凑成总金额所需的 **最少的硬币个数** 。如果没有任何一种硬币组合能组成总金额，返回 `-1` 。

你可以认为每种硬币的数量是无限的。

示例 1：

```
输入：coins = [1, 2, 5], amount = 11
输出：3
解释：11 = 5 + 5 + 1
```

示例 2：

```
输入：coins = [2], amount = 3
输出：-1
```

示例 3：

```
输入：coins = [1], amount = 0
输出：0
```

```python
class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        dp = [amount + 1] * (amount + 1)
        dp[0] = 0
        # 完全背包 组合问题，外层遍历物品
        for i in range(len(coins)):
            for j in range(coins[i], amount + 1):
                dp[j] = min(dp[j], dp[j - coins[i]] + 1)
        return dp[-1] if dp[-1] < amount + 1 else -1
```

337. 打家劫舍 III☆

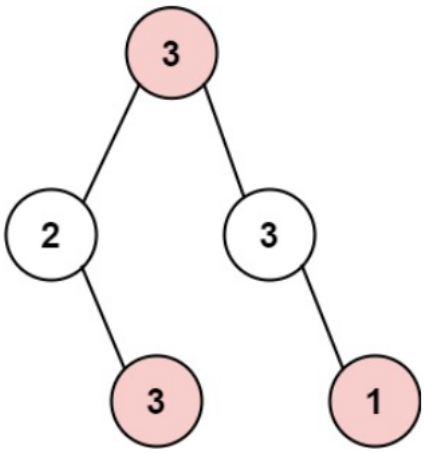
难度 中等 1134 收藏 分享 切换为英文 接收动态 反馈

小偷又发现了一个新的可行窃的地区。这个地区只有一个入口，我们称之为 `root` 。

除了 `root` 之外，每栋房子有且只有一个"父"房子与之相连。一番侦察之后，聪明的小偷意识到"这个地方的所有房屋的排列类似于一棵二叉树"。 如果 **两个直接相连的房子在同一天晚上被打劫** ，房屋将自动报警。

给定二叉树的 `root` 。返回 ***在不触动警报的情况下*** ，*小偷能够盗取的最高金额*。

示例 1:




```
输入：root = [3,2,3,null,3,null,1]
输出：7
解释：小偷一晚能够盗取的最高金额 3 + 3 + 1 = 7
```

```python
class Solution:
    def rob(self, root: TreeNode) -> int:
        def helper(root):
            if not root:
                return 0, 0
            left_rob, left_norob = helper(root.left)
            right_rob, right_norob = helper(root.right)

            root_norob = max(left_rob, left_norob) + max(right_rob, right_norob)
            root_rob = left_norob + right_norob + root.val
            return root_rob, root_norob
        return max(helper(root))
```

347. 前 K 个高频元素

难度 中等 1010 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 `nums` 和一个整数 `k` ，请你返回其中出现频率前 `k` 高的元素。你可以按 **任意顺序** 返回答案。

示例 1:

```
输入：nums = [1,1,1,2,2,3], k = 2
输出：[1,2]
```

示例 2:

```
输入：nums = [1], k = 1
输出：[1]
```

提示:

- `1 <= nums.length <= 10^5`
- `k` 的取值范围是 `[1, 数组中不相同的元素的个数]`
- 题目数据保证答案唯一，换句话说，数组中前 `k` 个高频元素的集合是唯一的

进阶：你所设计算法的时间复杂度 **必须** 优于 `O(n log n)` ，其中 `n` 是数组大小。

```python
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        dic = {}
        for n in nums:
            if n not in dic:
                dic[n] = 1
            else:
                dic[n] += 1
        heap = []
        for n, v in dic.items():
            if len(heap) < k:
                heapq.heappush(heap, (v, n))
            else:
                if v > heap[0][0]:
                    heapq.heappop(heap)
                    heapq.heappush(heap, (v, n))
        return [h[1] for h in heap]
```

406. 根据身高重建队列

难度 中等 1134 收藏 分享 切换为英文 接收动态 反馈

假设有打乱顺序的一群人站成一个队列，数组 `people` 表示队列中一些人的属性（不一定按顺序）。每个 `people[i] = [hi, ki]` 表示第 `i` 个人的身高为 `hi` ，前面 **正好** 有 `ki` 个身高大于或等于 `hi` 的人。

请你重新构造并返回输入数组 `people` 所表示的队列。返回的队列应该格式化为数组 `queue` ，其中 `queue[j] = [hj, kj]` 是队列中第 `j` 个人的属性（`queue[0]` 是排在队列前面的人）。

示例 1：

```
输入：people = [[7,0],[4,4],[7,1],[5,0],[6,1],[5,2]]
输出：[[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]]
解释：
编号为 0 的人身高为 5 ，没有身高更高或者相同的人排在他前面。
编号为 1 的人身高为 7 ，没有身高更高或者相同的人排在他前面。
编号为 2 的人身高为 5 ，有 2 个身高更高或者相同的人排在他前面，即编号为 0 和 1 的人。
编号为 3 的人身高为 6 ，有 1 个身高更高或者相同的人排在他前面，即编号为 1 的人。
编号为 4 的人身高为 4 ，有 4 个身高更高或者相同的人排在他前面，即编号为 0、1、2、3 的人。
编号为 5 的人身高为 7 ，有 1 个身高更高或者相同的人排在他前面，即编号为 1 的人。
因此 [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]] 是重新构造后的队列。
```

```python
class Solution:
    def reconstructQueue(self, people: List[List[int]]) -> List[List[int]]:
        heap = []
        for i in range(len(people)):
            people[i][0] = -people[i][0]
            heapq.heappush(heap, people[i])
        res = []
        for _ in range(len(heap)):
            p = heapq.heappop(heap)
            p[0] = -p[0]
            res.insert(p[1], p)
        return res
```

416. 分割等和子集

难度 中等 1133 收藏 分享 切换为英文 接收动态 反馈

给你一个 **只包含正整数** 的 **非空** 数组 `nums` 。请你判断是否可以将这个数组分割成两个子集，使得两个子集的元素和相等。

示例 1：

```
输入：nums = [1,5,11,5]
输出：true
解释：数组可以分割成 [1, 5, 5] 和 [11] 。
```

示例 2：

```
输入：nums = [1,2,3,5]
输出：false
解释：数组不能分割成两个元素和相等的子集。
```

提示：

- `1 <= nums.length <= 200`
- `1 <= nums[i] <= 100`

```python
class Solution:
    def canPartition(self, nums: List[int]) -> bool:
        s = sum(nums)
        if s % 2 == 1:
            return False
        s = s // 2
        dp = [False] * (s + 1)
        dp[0] = True
        for i in range(len(nums)):
            for j in range(s, nums[i] - 1, -1):
                dp[j] = dp[j] or dp[j-nums[i]]

        return dp[-1]
```

394. 字符串解码

难度 中等 1028 收藏 分享 切换为英文 接收动态 反馈

给定一个经过编码的字符串，返回它解码后的字符串。

编码规则为: `k[encoded_string]`，表示其中方括号内部的 `encoded_string` 正好重复 `k` 次。注意 `k` 保证为正整数。

你可以认为输入字符串总是有效的；输入字符串中没有额外的空格，且输入的方括号总是符合格式要求的。

此外，你可以认为原始数据不包含数字，所有的数字只表示重复的次数 `k` ，例如不会出现像 `3a` 或 `2[4]` 的输入。

示例 1:

```
输入：s = "3[a]2[bc]"
输出："aaabcbc"
```

示例 2:

```
输入：s = "3[a2[c]]"
输出："accaccacc"
```

示例 3:

```
输入：s = "2[abc]3[cd]ef"
输出："abcabccdcdcdef"
```

```python
class Solution:
    def decodeString(self, s: str) -> str:
        res = []
        stack = []
        for c in s:
            if c != ']':
                stack.append(c)
            else:
                temp = ""
                while stack[-1] != '[':
                    temp = stack.pop() + temp
                stack.pop()

                num = []
                cnt = 0
                while stack and stack[-1].isdigit():
                    num.append(int(stack.pop()))
                for i in range(len(num)):
                    cnt += num[i] * (10 ** i)
```

```
                stack.append(temp * cnt)

        return''.join(stack)
```

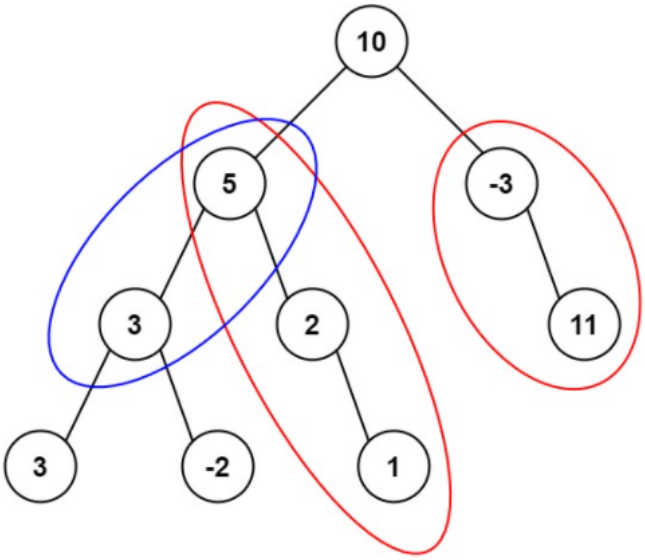
437. 路径总和 III

难度 中等 1222 收藏 分享 切换为英文 接收动态 反馈

给定一个二叉树的根节点 `root` ，和一个整数 `targetSum` ，求该二叉树里节点值之和等于 `targetSum` 的 **路径** 的数目。

路径 不需要从根节点开始，也不需要在叶子节点结束，但是路径方向必须是向下的（只能从父节点到子节点）。

示例 1：



输入： root = [10,5,-3,3,2,null,11,3,-2,null,1], targetSum = 8
输出： 3

```
class Solution:
    def pathSum(self, root: TreeNode, targetSum: int) -> int:
        self.res = 0
        def helper(root, target):
            if target == root.val:
                self.res += 1
            if root.left:
                helper(root.left, target - root.val)
            if root.right:
                helper(root.right, target - root.val)
        def dfs(root, target):
            if not root: return
            helper(root, target)
            if root.left:
                dfs(root.left, target)
            if root.right:
                dfs(root.right, target)
        dfs(root, targetSum)
        return self.res
```

399. 除法求值☆☆☆

难度 中等 679 收藏 分享 切换为英文 接收动态 反馈

给你一个变量对数组 `equations` 和一个实数值数组 `values` 作为已知条件，其中 `equations[i] = [Ai, Bi]` 和 `values[i]` 共同表示等式 $A_i / B_i = values[i]$ 。每个 A_i 或 B_i 是一个表示单个变量的字符串。

另有一些以数组 `queries` 表示的问题，其中 `queries[j] = [Cj, Dj]` 表示第 `j` 个问题，请你根据已知条件找出 $C_j / D_j = ?$ 的结果作为答案。

返回 **所有问题的答案** 。如果存在某个无法确定的答案，则用 `-1.0` 替代这个答案。如果问题中出现了给定的已知条件中没有出现的字符串，也需要用 `-1.0` 替代这个答案。

注意：输入总是有效的。你可以假设除法运算中不会出现除数为 0 的情况，且不存在任何矛盾的结果。

示例 1：

输入： equations = [["a","b"],["b","c"]], values = [2.0,3.0], queries = [["a","c"],["b","a"],["a","e"],["a","a"],["x","x"]]
输出： [6.00000,0.50000,-1.00000,1.00000,-1.00000]
解释：
条件：$a / b = 2.0, b / c = 3.0$
问题：$a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ?$
结果：[6.0, 0.5, -1.0, 1.0, -1.0]

示例 2：

输入： equations = [["a","b"],["b","c"],["bc","cd"]], values = [1.5,2.5,5.0], queries = [["a","c"],["c","b"],["bc","cd"],["cd","bc"]]
输出： [3.75000,0.40000,5.00000,0.20000]

```python
class Solution:
    def calcEquation(self, equations: List[List[str]], values: List[float], queries: List[List[str]]) -> List[float]:
        def bfs(start, end, graph):
            res = -1
            q = [(start, 1)]
            visited = set()
            visited.add(start)
            while q:
                node, weight = q.pop()  # 正常应该是pop(0)，但是此题和顺序无关，所以pop也可
                if node == end:
                    return weight
                for nxt, nxt_weight in graph[node].items():
                    if nxt not in visited:
                        visited.add(nxt)
                        q.append((nxt, weight * nxt_weight))
            return -1

        graph = {}
        for i, (start, end) in enumerate(equations):
            if start not in graph:
                graph[start] = {end: values[i]}
            else:
                graph[start][end] = values[i]
            if end not in graph:
                graph[end] = {start: 1 / values[i]}
            else:
                graph[end][start] = 1 / values[i]
```

```
        res = []
        for node in queries:
            if node[0] not in graph or node[1] not in graph:
                res.append(-1)
            else:
                res.append(bfs(node[0], node[1], graph))
        return res
```

494. 目标和

难度 中等 1047 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 `nums` 和一个整数 `target` 。

向数组中的每个整数前添加 `'+'` 或 `'-'` ，然后串联起所有整数，可以构造一个 **表达式** ：

- 例如，`nums = [2, 1]` ，可以在 `2` 之前添加 `'+'` ，在 `1` 之前添加 `'-'` ，然后串联起来得到表达式 `"+2-1"` 。

返回可以通过上述方法构造的、运算结果等于 `target` 的不同 **表达式** 的数目。

示例 1：

输入： nums = [1,1,1,1,1], target = 3
输出： 5
解释： 一共有 5 种方法让最终目标和为 3 。
-1 + 1 + 1 + 1 + 1 = 3
+1 - 1 + 1 + 1 + 1 = 3
+1 + 1 - 1 + 1 + 1 = 3
+1 + 1 + 1 - 1 + 1 = 3
+1 + 1 + 1 + 1 - 1 = 3

示例 2：

输入： nums = [1], target = 1
输出： 1

```
class Solution:
    def findTargetSumWays(self, nums: List[int], target: int) -> int:
        # 正数的和x，  负数的绝对值的和y，  x-y=target
        # x + y = sum(target) = s
        # y = s - x
        # x - (s  - x) = 2x-s=target
        # x = (target + s) / 2

        s = sum(nums)
        if s < target or (target + s) % 2 == 1: return 0
        x = abs((target + s) // 2)
        dp = [0] * (x + 1)
        dp[0] = 1
        for num in nums:
            for j in range(x, num-1, -1):
                dp[j] += dp[j-num]
        return dp[-1]
```

438. 找到字符串中所有字母异位词 ☆☆☆☆

难度 中等 770 收藏 分享 切换为英文 接收动态 反馈

给定两个字符串 `s` 和 `p`，找到 `s` 中所有 `p` 的 **异位词** 的子串，返回这些子串的起始索引。不考虑答案输出的顺序。

异位词 指由相同字母重排列形成的字符串（包括相同的字符串）。

示例 1:

```
输入: s = "cbaebabacd", p = "abc"
输出: [0,6]
解释:
起始索引等于 0 的子串是 "cba", 它是 "abc" 的异位词。
起始索引等于 6 的子串是 "bac", 它是 "abc" 的异位词。
```

示例 2:

```
输入: s = "abab", p = "ab"
输出: [0,1,2]
解释:
起始索引等于 0 的子串是 "ab", 它是 "ab" 的异位词。
起始索引等于 1 的子串是 "ba", 它是 "ab" 的异位词。
起始索引等于 2 的子串是 "ab", 它是 "ab" 的异位词。
```

```python
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        left = 0
        window = {}
        need = {}
        match = 0
        res = []
        for c in p:
            if c not in need:
                need[c] = 1
            else:
                need[c] += 1
        for right in range(len(s)):
            if s[right] in need:
                if s[right] not in window: window[s[right]] = 1
                else: window[s[right]] += 1
                if window[s[right]] == need[s[right]]: match += 1
            while match == len(need):
                if right - left + 1 == len(p):
                    res.append(left)
                if s[left] in need:
                    window[s[left]] -= 1
                    if window[s[left]] < need[s[left]]:
                        match -= 1
                left += 1
        return res
```

538. 把二叉搜索树转换为累加树

难度 中等 649 收藏 分享 切换为英文 接收动态 反馈

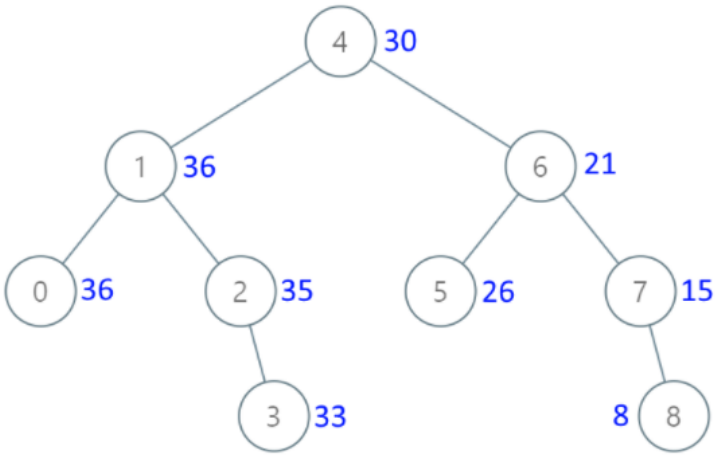
给出二叉 **搜索** 树的根节点，该树的节点值各不相同，请你将其转换为累加树（Greater Sum Tree），使每个节点 `node` 的新值等于原树中大于或等于 `node.val` 的值之和。

提醒一下，二叉搜索树满足下列约束条件：

- 节点的左子树仅包含键 **小于** 节点键的节点。
- 节点的右子树仅包含键 **大于** 节点键的节点。
- 左右子树也必须是二叉搜索树。

注意：本题和 1038: https://leetcode-cn.com/problems/binary-search-tree-to-greater-sum-tree/ 相同

示例 1：




```
class Solution:
    def convertBST(self, root: Optional[TreeNode]) -> Optional[TreeNode]:
        self.prev = 0
        def helper(root):
            if not root:
                return
            helper(root.right)
            root.val += self.prev
            self.prev = root.val
            helper(root.left)
        helper(root)
        return root
```

560. 和为 K 的子数组☆☆

难度 中等 1315 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 `nums` 和一个整数 `k` ，请你统计并返回该数组中和为 `k` 的连续子数组的个数。

示例 1：

```
输入：nums = [1,1,1], k = 2
输出：2
```

示例 2：

```
输入：nums = [1,2,3], k = 3
输出：2
```

提示：

- $1 <= nums.length <= 2 * 10^4$
- $-1000 <= nums[i] <= 1000$
- $-10^7 <= k <= 10^7$

```python
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        """
        [1,2,3,4,-1] k=3
      [0,1,3,6,10,9]
      [1,-1,0] k=0
      [0,1,0,0]
        """
        pre_sum = 0
        dic = collections.defaultdict(int)
        dic[0] = 1
        res = 0
        for i, num in enumerate(nums):
            pre_sum += num
            if pre_sum - k in dic:
                res += dic[pre_sum - k]
            dic[pre_sum] += 1

        return res
```

581. 最短无序连续子数组 ☆☆

难度 中等 780 收藏 分享 切换为英文 接收动态 反馈

给你一个整数数组 `nums` ，你需要找出一个 **连续子数组** ，如果对这个子数组进行升序排序，那么整个数组都会变为升序排序。

请你找出符合题意的 **最短** 子数组，并输出它的长度。

示例 1：

```
输入：nums = [2,6,4,8,10,9,15]
输出：5
解释：你只需要对 [6, 4, 8, 10, 9] 进行升序排序，那么整个表都会变为升序排序。
```

示例 2：

```
输入：nums = [1,2,3,4]
输出：0
```

示例 3：

```
输入：nums = [1]
输出：0
```

提示：

- `1 <= nums.length <= 10^4`
- `-10^5 <= nums[i] <= 10^5`

进阶：你可以设计一个时间复杂度为 `O(n)` 的解决方案吗？

```python
class Solution:
    def findUnsortedSubarray(self, nums: List[int]) -> int:
        left = -1
        right = -1
        maxn = -float('inf')
        minn = float('inf')
        n = len(nums)
        for i in range(n):
            if maxn <= nums[i]:
                maxn = nums[i]
            else:
                right = i

            if minn >= nums[n-1-i]:
                minn = nums[n-1-i]
            else:
                left = n-1-i
        return 0 if right == -1 else right-left+1
```

647. 回文子串☆

难度 中等 776 收藏 分享 切换为英文 接收动态 反馈

给你一个字符串 `s` ，请你统计并返回这个字符串中 **回文子串** 的数目。

回文字符串 是正着读和倒过来读一样的字符串。

子字符串 是字符串中的由连续字符组成的一个序列。

具有不同开始位置或结束位置的子串，即使是由相同的字符组成，也会被视作不同的子串。

示例 1：

```
输入：s = "abc"
输出：3
解释：三个回文子串: "a", "b", "c"
```

示例 2：

```
输入：s = "aaa"
输出：6
解释：6个回文子串: "a", "a", "a", "aa", "aa", "aaa"
```

提示：

- `1 <= s.length <= 1000`
- `s` 由小写英文字母组成

```python
class Solution:
    def countSubstrings(self, s: str) -> int:
        n = len(s)
        dp = [[False] * n for _ in range(n)]
        res = 0
        for i in range(n-1, -1, -1):
            for j in range(i, n):
                if s[i] == s[j]:
                    if j - i <= 1:
                        res += 1
                        dp[i][j] = True
                    elif dp[i+1][j-1]:
                        res += 1
                        dp[i][j] = True
        return res
```

739. 每日温度

难度 中等 1022 收藏 分享 切换为英文 接收动态 反馈

给定一个整数数组 `temperatures` ，表示每天的温度，返回一个数组 `answer` ，其中 `answer[i]` 是指在第 `i` 天之后，才会有更高的温度。如果气温在这之后都不会升高，请在该位置用 `0` 来代替。

示例 1:

```
输入: temperatures = [73,74,75,71,69,72,76,73]
输出: [1,1,4,2,1,1,0,0]
```

示例 2:

```
输入: temperatures = [30,40,50,60]
输出: [1,1,1,0]
```

示例 3:

```
输入: temperatures = [30,60,90]
输出: [1,1,0]
```

提示:

- `1 <= temperatures.length <= 10^5`
- `30 <= temperatures[i] <= 100`

```python
class Solution:
    def dailyTemperatures(self, temperatures: List[int]) -> List[int]:
        # 找到下一个更大的数，单调栈是存 栈底到栈顶递减的数
        n = len(temperatures)
        res = [0] * n
        stack = []
        for i, t in enumerate(temperatures):
            if not stack or t <= temperatures[stack[-1]]:
                stack.append(i)
            else:
                while stack and t > temperatures[stack[-1]]:
                    res[stack[-1]] = i - stack[-1]
                    stack.pop()
                stack.append(i)
        return res
```

621. 任务调度器

贪心 图解 代码简洁 - 任务调度器 - 力扣（LeetCode） (leetcode-cn.com)

难度 中等 824 收藏 分享 切换为英文 接收动态 反馈

给你一个用字符数组 `tasks` 表示的 CPU 需要执行的任务列表。其中每个字母表示一种不同种类的任务。任务可以以任意顺序执行，并且每个任务都可以在 1 个单位时间内执行完。在任何一个单位时间，CPU 可以完成一个任务，或者处于待命状态。

然而，两个 **相同种类** 的任务之间必须有长度为整数 `n` 的冷却时间，因此至少有连续 `n` 个单位时间内 CPU 在执行不同的任务，或者在待命状态。

你需要计算完成所有任务所需要的 **最短时间** 。

示例 1：

输入： `tasks = ["A","A","A","B","B","B"], n = 2`
输出： `8`
解释： `A -> B -> (待命) -> A -> B -> (待命) -> A -> B`
在本示例中，两个相同类型任务之间必须间隔长度为 `n = 2` 的冷却时间，而执行一个任务只需要一个单位时间，所以中间出现了（待命）状态。

示例 2：

输入： `tasks = ["A","A","A","B","B","B"], n = 0`
输出： `6`
解释： 在这种情况下，任何大小为 `6` 的排列都可以满足要求，因为 `n = 0`
`["A","A","A","B","B","B"]`
`["A","B","A","B","A","B"]`
`["B","B","B","A","A","A"]`
`...`
诸如此类

```python
class Solution:
    def leastInterval(self, tasks: List[str], n: int) -> int:
        dic = collections.defaultdict(int)
        max_cnt = 0
        for c in tasks:
            dic[c] += 1
            max_cnt = max(max_cnt, dic[c])
        res = (max_cnt - 1) * (n + 1)
        for c in dic.keys():
            if dic[c] == max_cnt:
                res += 1
        return max(res, len(tasks))
```

146. LRU 缓存

难度 中等 1924 收藏 分享 切换为英文 接收动态 反馈

请你设计并实现一个满足 LRU (最近最少使用) 缓存 约束的数据结构。
实现 `LRUCache` 类：

- `LRUCache(int capacity)` 以 **正整数** 作为容量 `capacity` 初始化 LRU 缓存
- `int get(int key)` 如果关键字 `key` 存在于缓存中，则返回关键字的值，否则返回 `-1` 。
- `void put(int key, int value)` 如果关键字 `key` 已经存在，则变更其数据值 `value` ；如果不存在，则向缓存中插入该组 `key-value` 。如果插入操作导致关键字数量超过 `capacity` ，则应该 **逐出** 最久未使用的关键字。

函数 `get` 和 `put` 必须以 `O(1)` 的平均时间复杂度运行。

示例：

输入
```
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
```
输出
```
[null, null, null, 1, null, -1, null, -1, 3, 4]
```

解释
```
LRUCache lRUCache = new LRUCache(2);
lRUCache.put(1, 1); // 缓存是 {1=1}
lRUCache.put(2, 2); // 缓存是 {1=1, 2=2}
lRUCache.get(1);    // 返回 1
lRUCache.put(3, 3); // 该操作会使得关键字 2 作废，缓存是 {1=1, 3=3}
lRUCache.get(2);    // 返回 -1 (未找到)
lRUCache.put(4, 4); // 该操作会使得关键字 1 作废，缓存是 {4=4, 3=3}
lRUCache.get(1);    // 返回 -1 (未找到)
lRUCache.get(3);    // 返回 3
lRUCache.get(4);    // 返回 4
```

```python
class ListNode:
    def __init__(self, key, value, prev=None, next=None):
        self.key = key
        self.value = value
        self.prev = prev
        self.next = None


class LRUCache:

    def __init__(self, capacity: int):
        self.cap = capacity
        self.size = 0
        self.map = {}
        self.head = ListNode(-1, -1)
        self.tail = ListNode(-1, -1)
        self.head.next = self.tail
        self.tail.prev = self.head

    def get(self, key: int) -> int:
        if key in self.map:
            self.move_to_tail(key, self.map[key].value)
            return self.map[key].value
        else:
            return -1

    def put(self, key: int, value: int) -> None:
        if key in self.map:
            self.move_to_tail(key, value)
```

```
            else:
                node = ListNode(key, value)
                self.add_last(node)

    def remove_cur_node(self, node):
        node.prev.next = node.next
        node.next.prev = node.prev
        self.map.pop(node.key)
        self.size -= 1

    def add_last(self, node):
        if self.size == self.cap:
            self.remove_cur_node(self.head.next)
        self.tail.prev.next = node
        node.next = self.tail
        node.prev = self.tail.prev
        self.tail.prev = node
        self.map[node.key] = node
        self.size += 1

    def move_to_tail(self, key, value):
        node = self.map[key]
        self.remove_cur_node(node)
        node = ListNode(key, value)
        self.add_last(node)


# Your LRUCache object will be instantiated and called as such:
# obj = LRUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)
```