# leetcode-125:验证回文串

**题目链接**：[https://leetcode-cn.com/problems/valid-palindrome/](https://leetcode-cn.com/problems/valid-palindrome/" \t "_blank)

**题目描述**：给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。

说明**：**本题中，我们将空字符串定义为有效的回文串。

示例 1:

输入: "A man, a plan, a canal: Panama"

输出: true

示例 2:

输入: "race a car"

输出: false

**参考答案:**

class Solution:

    def isPalindrome(self, s: str) -> bool:

        n = len(s)

        left, right = 0, n - 1

        while left < right:

            while left < right and not s[left].isalnum():

                left += 1

            while left < right and not s[right].isalnum():

                right -= 1

            if left < right:

                if s[left].lower() != s[right].lower():

                    return False

                left, right = left + 1, right - 1

        return True

# leetcode-859: 亲密字符串

**题目链接**：[https://leetcode-cn.com/problems/buddy-strings/](https://leetcode-cn.com/problems/buddy-strings/" \t "_blank)

**题目描述**：题目给定两个由小写字母构成的字符串 A 和 B ，只要我们可以通过交换 A 中的两个字母得到与 B 相等的结果，就返回 true ；否则返回 false 。

示例 1：

输入： A = "ab", B = "ba"

输出： true

示例 2：

输入： A = "ab", B = "ab"

输出： false

示例 3:

输入： A = "aa", B = "aa"

输出： true

示例 4：

输入： A = "aaaaaaabc", B = "aaaaaaacb"

输出： true

示例 5：

输入： A = "", B = "aa"

输出： false

提示：

    0 <= A.length <= 20000

    0 <= B.length <= 20000

    A 和 B 仅由小写字母构成。

**参考答案：**

class Solution(object):

    def buddyStrings(self, A, B):

        if len(A) != len(B): return False

        if A == B:

            seen = set()

            for a in A:

                if a in seen:

                    return True

                seen.add(a)

            return False

        else:

            pairs = []

            for a, b in itertools.izip(A, B):

                if a != b:

                    pairs.append((a, b))

                if len(pairs) >= 3: return False

            return len(pairs) == 2 and pairs[0] == pairs[1][::-1]

# leetcode-557: 反转字符串中的单词 III

**题目链接：**[https://leetcode-cn.com/problems/reverse-words-in-a-string-iii/](https://leetcode-cn.com/problems/reverse-words-in-a-string-iii/" \t "_blank)

**题目描述：**

给定一个字符串，你需要反转字符串中每个单词的字符顺序，同时仍保留空格和单词的初始顺序。

**示例 1:**

输入: "Let's take LeetCode contest"

输出: "s'teL ekat edoCteeL tsetnoc"

**注意：**在字符串中，每个单词由单个空格分隔，并且字符串中不会有任何额外的空格。

**参考答案：**

class Solution(object):

    def reverseWords(self, s):

        return " ".join(word[::-1] for word in s.split(" "))

# 4.leetcode-15:三数之和

**题目链接：**[https://leetcode-cn.com/problems/3sum/](https://leetcode-cn.com/problems/3sum/" \t "_blank)

**题目描述：**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。

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]

]

**参考答案：**

class Solution:

    def threeSum(self, nums: List[int]) -> List[List[int]]:

        n = len(nums)

        nums.sort()

        ans = list()

        # 枚举 a

        for first in range(n):

            # 需要和上一次枚举的数不相同

            if first > 0 and nums[first] == nums[first - 1]:

                continue

            # c 对应的指针初始指向数组的最右端

            third = n - 1

            target = -nums[first]

            # 枚举 b

            for second in range(first + 1, n):

                # 需要和上一次枚举的数不相同

                if second > first + 1 and nums[second] == nums[second - 1]:

                    continue

                # 需要保证 b 的指针在 c 的指针的左侧

                while second < third and nums[second] + nums[third] > target:

                    third -= 1

                # 如果指针重合，随着 b 后续的增加

                # 就不会有满足 a+b+c=0 并且 b<c 的 c 了，可以退出循环

                if second == third:

                    break

                if nums[second] + nums[third] == target:

                    ans.append([nums[first], nums[second], nums[third]])

        return ans

# 5.leetcode-923:三数之和的多种可能

**题目链接：**[https://leetcode-cn.com/problems/3sum-with-multiplicity/](https://leetcode-cn.com/problems/3sum-with-multiplicity/" \t "_blank)

**题目描述：**给定一个整数数组 A，以及一个整数 target 作为目标值，返回满足 i < j < k 且 A[i] + A[j] + A[k] == target 的元组 i, j, k 的数量。

由于结果会非常大，请返回 结果除以 10^9 + 7 的余数。

示例 1：

输入：A = [1,1,2,2,3,3,4,4,5,5], target = 8

输出：20

解释：

按值枚举（A[i]，A[j]，A[k]）：

(1, 2, 5) 出现 8 次；

(1, 3, 4) 出现 8 次；

(2, 2, 4) 出现 2 次；

(2, 3, 3) 出现 2 次。

示例 2：

输入：A = [1,1,2,2,2,2], target = 5

输出：12

解释：

A[i] = 1，A[j] = A[k] = 2 出现 12 次：

我们从 [1,1] 中选择一个 1，有 2 种情况，

从 [2,2,2,2] 中选出两个 2，有 6 种情况。

提示：

    3 <= A.length <= 3000

    0 <= A[i] <= 100

    0 <= target <= 300

**参考答案：**

class Solution(object):

    def threeSumMulti(self, A, target):

        MOD = 10\*\*9 + 7

        count = collections.Counter(A)

        keys = sorted(count)

        ans = 0

        # Now, let's do a 3sum on "keys", for i <= j <= k.

        # We will use count to add the correct contribution to ans.

        for i, x in enumerate(keys):

            T = target - x

            j, k = i, len(keys) - 1

            while j <= k:

                y, z = keys[j], keys[k]

                if y + z < T:

                    j += 1

                elif y + z > T:

                    k -= 1

                else: # x+y+z == T, now calculate the size of the contribution

                    if i < j < k:

                        ans += count[x] \* count[y] \* count[z]

                    elif i == j < k:

                        ans += count[x] \* (count[x] - 1) / 2 \* count[z]

                    elif i < j == k:

                        ans += count[x] \* count[y] \* (count[y] - 1) / 2

                    else:  # i == j == k

                        ans += count[x] \* (count[x] - 1) \* (count[x] - 2) / 6

                    j += 1

                    k -= 1

        return ans % MOD

# leetcode-142:环形链表II

**题目链接：**[https://leetcode-cn.com/problems/linked-list-cycle-ii/](https://leetcode-cn.com/problems/linked-list-cycle-ii/" \t "_blank)

**题目描述：**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。

为了表示给定链表中的环，我们使用整数 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。

说明：不允许修改给定的链表。

例 1：

输入：head = [3,2,0,-4], pos = 1

输出：tail connects to node index 1

解释：链表中有一个环，其尾部连接到第二个节点。

示例 2：

输入：head = [1,2], pos = 0

输出：tail connects to node index 0

解释：链表中有一个环，其尾部连接到第一个节点。

**参考答案：**

class Solution(object):

    def detectCycle(self, head):

        visited = set()

        node = head

        while node is not None:

            if node in visited:

                return node

            else:

                visited.add(node)

                node = node.next

        return None

# 7.剑指offer24:翻转链表

**题目链接：**[https://leetcode-cn.com/problems/fan-zhuan-lian-biao-lcof/](https://leetcode-cn.com/problems/fan-zhuan-lian-biao-lcof/" \t "_blank)

**题目描述：**定义一个函数，输入一个链表的头节点，反转该链表并输出反转后链表的头节点。

示例:

输入: 1->2->3->4->5->NULL

输出: 5->4->3->2->1->NULL

限制：

0 <= 节点个数 <= 5000

**参考答案：**

# 实现一个链表类，只有一个值val和一个指向下一个节点的next'指针'

class ListNode:

    def \_\_init\_\_(self, x):

        self.val = x

        self.next = None

class Solution:

    # 返回从尾部到头部的列表值序列，例如[1,2,3]

    def printListFromTailToHead(self, listNode):

        # @listNode: 头结点

        # write code here

        l = []

        if listNode is None:

            return la.next = b

            b.next = c

        while listNode:

            l.append(listNode.val)

            listNode = listNode.next

        return l[::-1]

# 创建链表 a->b->c

a = ListNode(1)

b = ListNode(2)

c = ListNode(3)

a.next = b

b.next = c

# 实例化

demo = Solution()

print(demo.printListFromTailToHead(a))

# 8.剑指offer-27:二叉树的镜像

**题目链接：**[https://leetcode-cn.com/problems/er-cha-shu-de-jing-xiang-lcof/](https://leetcode-cn.com/problems/er-cha-shu-de-jing-xiang-lcof/" \t "_blank)

**题目描述：**请完成一个函数，输入一个二叉树，该函数输出它的镜像。

**示例 1：**

**输入：**root = [4,2,7,1,3,6,9]

**输出：**[4,7,2,9,6,3,1]

**参考答案：**

class Solution:

    def mirrorTree(self, root: TreeNode) -> TreeNode:

        if not root: return

        tmp = root.left

        root.left = self.mirrorTree(root.right)

        root.right = self.mirrorTree(tmp)

        return root

# 9.剑指offer26:树的子结构

**题目链接：**[https://leetcode-cn.com/problems/shu-de-zi-jie-gou-lcof/](https://leetcode-cn.com/problems/shu-de-zi-jie-gou-lcof/" \t "_blank)

**题目描述：**输入两棵二叉树A和B，判断B是不是A的子结构。(约定空树不是任意一个树的子结构)

B是A的子结构， 即 A中有出现和B相同的结构和节点值。

示例 1：

输入：A = [1,2,3], B = [3,1]

输出：false

示例 2：

输入：A = [3,4,5,1,2], B = [4,1]

输出：true

**参考答案：**

class Solution:

    def isSubStructure(self, A: TreeNode, B: TreeNode) -> bool:

        def recur(A, B):

            if not B: return True

            if not A or A.val != B.val: return False

            return recur(A.left, B.left) and recur(A.right, B.right)

        return bool(A and B) and (recur(A, B) or self.isSubStructure(A.left, B) or self.isSubStructure(A.right, B))

# 10.熟悉一下DFS

dfs算法，大家接触一下[https://juejin.im/post/5be661faf265da6116394c09](https://juejin.im/post/5be661faf265da6116394c09" \t "_blank)

深度优先搜索 [https://juejin.im/post/5be661faf265da6116394c09](https://juejin.im/post/5be661faf265da6116394c09" \t "_blank)

# 11.leetcode-3:无重复字符的最长子串

**题目链接：**[https://leetcode-cn.com/problems/longest-substring-without-repeating-characters/](https://leetcode-cn.com/problems/longest-substring-without-repeating-characters/" \t "_blank)

**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。

**参考答案：**

class Solution:

    def lengthOfLongestSubstring(self, s: str) -> int:

        # 哈希集合，记录每个字符是否出现过

        occ = set()

        n = len(s)

        # 右指针，初始值为 -1，相当于我们在字符串的左边界的左侧，还没有开始移动

        rk, ans = -1, 0

        for i in range(n):

            if i != 0:

                # 左指针向右移动一格，移除一个字符

                occ.remove(s[i - 1])

            while rk + 1 < n and s[rk + 1] not in occ:

                # 不断地移动右指针

                occ.add(s[rk + 1])

                rk += 1

            # 第 i 到 rk 个字符是一个极长的无重复字符子串

            ans = max(ans, rk - i + 1)

        return ans

# 12.剑指offer-51：数组中的逆序对

**题目链接：**[https://leetcode-cn.com/problems/shu-zu-zhong-de-ni-xu-dui-lcof/](https://leetcode-cn.com/problems/shu-zu-zhong-de-ni-xu-dui-lcof/" \t "_blank)

**题目描述：**在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组，求出这个数组中的逆序对的总数。

**示例 1:**

**输入**: [7,5,6,4]

**输出**: 5

**参考答案：**

class Solution:

    def mergeSort(self, nums, tmp, l, r):

        if l >= r:

            return 0

        mid = (l + r) // 2

        inv\_count = self.mergeSort(nums, tmp, l, mid) + self.mergeSort(nums, tmp, mid + 1, r)

        i, j, pos = l, mid + 1, l

        while i <= mid and j <= r:

            if nums[i] <= nums[j]:

                tmp[pos] = nums[i]

                i += 1

                inv\_count += (j - (mid + 1))

            else:

                tmp[pos] = nums[j]

                j += 1

            pos += 1

        for k in range(i, mid + 1):

            tmp[pos] = nums[k]

            inv\_count += (j - (mid + 1))

            pos += 1

        for k in range(j, r + 1):

            tmp[pos] = nums[k]

            pos += 1

        nums[l:r+1] = tmp[l:r+1]

        return inv\_count

    def reversePairs(self, nums: List[int]) -> int:

        n = len(nums)

        tmp = [0] \* n

        return self.mergeSort(nums, tmp, 0, n - 1)

# 13.leetcode-148:排序链表

**题目链接：**[https://leetcode-cn.com/problems/sort-list/](https://leetcode-cn.com/problems/sort-list/" \t "_blank)

**题目描述：**在 O(n log n) 时间复杂度和常数级空间复杂度下，对链表进行排序。

示例 1:

输入: 4->2->1->3

输出: 1->2->3->4

示例 2:

输入: -1->5->3->4->0

输出: -1->0->3->4->5

**参考答案：**

class Solution:

    def sortList(self, head: ListNode) -> ListNode:

        if not head or not head.next: return head # termination.

        # cut the LinkedList at the mid index.

        slow, fast = head, head.next

        while fast and fast.next:

            fast, slow = fast.next.next, slow.next

        mid, slow.next = slow.next, None # save and cut.

        # recursive for cutting.

        left, right = self.sortList(head), self.sortList(mid)

        # merge `left` and `right` linked list and return it.

        h = res = ListNode(0)

        while left and right:

            if left.val < right.val: h.next, left = left, left.next

            else: h.next, right = right, right.next

            h = h.next

        h.next = left if left else right

        return res.next

# 14.leetcode-23:合并K个排序链表

**题目链接：**[https://leetcode-cn.com/problems/merge-k-sorted-lists/](https://leetcode-cn.com/problems/merge-k-sorted-lists/" \t "_blank)

**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

**参考答案：**

class Solution:

    def mergeKLists(self, lists: List[ListNode]) -> ListNode:

        nodes = []

        head = point = ListNode(0)

        for l in lists:

            while l:

                nodes.append(l.val)

                l = l.next

        for x in sorted(nodes):

            point.next = ListNode(x)

            point = point.next

        return head.next

# 15.leetcode-82:删除排序链表中重复元素II

**题目链接：**[https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list-ii/](https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list-ii/" \t "_blank)

**题目描述：**给定一个排序链表，删除所有含有重复数字的节点，只保留原始链表中 没有重复出现 的数字。

示例 1:

输入: 1->2->3->3->4->4->5

输出: 1->2->5

示例 2:

输入: 1->1->1->2->3

输出: 2->3

**参考答案：**

class Solution:#感谢各位的更好思路或改进办法

    def deleteDuplicates(self, head: ListNode) -> ListNode:

        thead = ListNode('a')

        thead.next = head

        pre,cur = None,thead

        while cur:

            pre=cur

            cur=cur.next

            while cur and cur.next and cur.next.val == cur.val:

                t=cur.val

                while cur and cur.val==t:

                    cur=cur.next

            pre.next=cur

        return thead.next

# 16.leetcode-19:删除链表的倒数第N个节点

**题目链接：**[https://leetcode-cn.com/problems/remove-nth-node-from-end-of-list/](https://leetcode-cn.com/problems/remove-nth-node-from-end-of-list/" \t "_blank)

**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。

**参考答案：**

class Solution:

    def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:

        slow=head

        fast=head

        while n>0:

            fast=fast.next

            n-=1

        if fast:

            while fast.next:

                fast=fast.next

                slow=slow.next

            slow.next=slow.next.next

            return head

        else:

            return head.next

# 17.剑指offer-35:复杂链表的复制

**题目链接：**[https://leetcode-cn.com/problems/fu-za-lian-biao-de-fu-zhi-lcof/](https://leetcode-cn.com/problems/fu-za-lian-biao-de-fu-zhi-lcof/" \t "_blank)

**题目描述：**请实现 copyRandomList 函数，复制一个复杂链表。在复杂链表中，每个节点除了有一个 next 指针指向下一个节点，还有一个 random 指针指向链表中的任意节点或者 null。

**示例 1：**

输入：head = [[7,null],[13,0],[11,4],[10,2],[1,0]]

输出：[[7,null],[13,0],[11,4],[10,2],[1,0]]

**示例 2：**

**输入：**head = [[1,1],[2,1]]

**输出：**[[1,1],[2,1]]

**参考答案：**

class Solution:

    def copyRandomList(self, head: 'Node') -> 'Node':

        def dfs(head):

            if not head: return None

            if head in visited:

                return visited[head]

            # 创建新结点

            copy = Node(head.val, None, None)

            visited[head] = copy

            copy.next = dfs(head.next)

            copy.random = dfs(head.random)

            return copy

        visited = {}

        return dfs(head)

# 18.leetcode-138:复制带随机指针的链表

**题目链接：**[https://leetcode-cn.com/problems/copy-list-with-random-pointer/](https://leetcode-cn.com/problems/copy-list-with-random-pointer/" \t "_blank)

**题目描述：**给定一个链表，每个节点包含一个额外增加的随机指针，该指针可以指向链表中的任何节点或空节点。

要求返回这个链表的 深拷贝。

我们用一个由 n 个节点组成的链表来表示输入/输出中的链表。每个节点用一个 [val, random\_index] 表示：

    val：一个表示 Node.val 的整数。

    random\_index：随机指针指向的节点索引（范围从 0 到 n-1）；如果不指向任何节点，则为  null 。

**参考答案：**

class Solution(object):

    def copyRandomList(self, head):

        """

        :type head: Node

        :rtype: Node

        """

        if not head:

            return head

        # Creating a new weaved list of original and copied nodes.

        ptr = head

        while ptr:

            # Cloned node

            new\_node = Node(ptr.val, None, None)

            # Inserting the cloned node just next to the original node.

            # If A->B->C is the original linked list,

            # Linked list after weaving cloned nodes would be A->A'->B->B'->C->C'

            new\_node.next = ptr.next

            ptr.next = new\_node

            ptr = new\_node.next

        ptr = head

        # Now link the random pointers of the new nodes created.

        # Iterate the newly created list and use the original nodes random pointers,

        # to assign references to random pointers for cloned nodes.

        while ptr:

            ptr.next.random = ptr.random.next if ptr.random else None

            ptr = ptr.next.next

        # Unweave the linked list to get back the original linked list and the cloned list.

        # i.e. A->A'->B->B'->C->C' would be broken to A->B->C and A'->B'->C'

        ptr\_old\_list = head # A->B->C

        ptr\_new\_list = head.next # A'->B'->C'

        head\_old = head.next

        while ptr\_old\_list:

            ptr\_old\_list.next = ptr\_old\_list.next.next

            ptr\_new\_list.next = ptr\_new\_list.next.next if ptr\_new\_list.next else None

            ptr\_old\_list = ptr\_old\_list.next

            ptr\_new\_list = ptr\_new\_list.next

        return head\_old

# 19.leetcode-110:平衡二叉树

**题目链接：**[https://leetcode-cn.com/problems/balanced-binary-tree/](https://leetcode-cn.com/problems/balanced-binary-tree/" \t "_blank)

**题目描述：**给定一个二叉树，判断它是否是高度平衡的二叉树。

本题中，一棵高度平衡二叉树定义为：

    一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过1。

**示例 1:**

给定二叉树 [3,9,20,null,null,15,7]

返回 true 。

**示例 2:**

给定二叉树 [1,2,2,3,3,null,null,4,4]

返回 false 。

**参考答案：**

class Solution:

    # Compute the tree's height via recursion

    def height(self, root: TreeNode) -> int:

        # An empty tree has height -1

        if not root:

            return -1

        return 1 + max(self.height(root.left), self.height(root.right))

    def isBalanced(self, root: TreeNode) -> bool:

        # An empty tree satisfies the definition of a balanced tree

        if not root:

            return True

        # Check if subtrees have height within 1. If they do, check if the

        # subtrees are balanced

        return abs(self.height(root.left) - self.height(root.right)) < 2 \

            and self.isBalanced(root.left) \

            and self.isBalanced(root.right)

# 20.剑指offer-01:重建二叉树

**题目链接：**[https://leetcode-cn.com/problems/zhong-jian-er-cha-shu-lcof/](https://leetcode-cn.com/problems/zhong-jian-er-cha-shu-lcof/" \t "_blank)

**题目描述：**输入某二叉树的前序遍历和中序遍历的结果，请重建该二叉树。假设输入的前序遍历和中序遍历的结果中都不含重复的数字。

例如，给出

前序遍历 preorder = [3,9,20,15,7]

中序遍历 inorder = [9,3,15,20,7]

**参考答案：**

class Solution:

    def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:

        self.dic, self.po = {}, preorder

        for i in range(len(inorder)):

            self.dic[inorder[i]] = i

        return self.recur(0, 0, len(inorder) - 1)

    def recur(self, pre\_root, in\_left, in\_right):

        if in\_left > in\_right: return # 终止条件：中序遍历为空

        root = TreeNode(self.po[pre\_root]) # 建立当前子树的根节点

        i = self.dic[self.po[pre\_root]]    # 搜索根节点在中序遍历中的索引，从而可对根节点、左子树、右子树完成划分。

        root.left = self.recur(pre\_root + 1, in\_left, i - 1) # 开启左子树的下层递归

        root.right = self.recur(i - in\_left + pre\_root + 1, i + 1, in\_right) # 开启右子树的下层递归

        return root # 返回根节点，作为上层递归的左（右）子节点

# 21.剑指offer 32-1:从上到下打印二叉树

**题目链接：**[https://leetcode-cn.com/problems/cong-shang-dao-xia-da-yin-er-cha-shu-lcof/](https://leetcode-cn.com/problems/cong-shang-dao-xia-da-yin-er-cha-shu-lcof/" \t "_blank)

**题目描述：**从上到下打印出二叉树的每个节点，同一层的节点按照从左到右的顺序打印。

例如:

给定二叉树: [3,9,20,null,null,15,7],

返回：

[3,9,20,15,7]

**参考答案：**

class Solution:

    def levelOrder(self, root: TreeNode) -> List[int]:

        if not root: return []

        res, queue = [], collections.deque()

        queue.append(root)

        while queue:

            node = queue.popleft()

            res.append(node.val)

            if node.left: queue.append(node.left)

            if node.right: queue.append(node.right)

        return res

# 22.剑指offer 32-III:从上到下打印二叉树

**题目链接：**[https://leetcode-cn.com/problems/cong-shang-dao-xia-da-yin-er-cha-shu-iii-lcof/](https://leetcode-cn.com/problems/cong-shang-dao-xia-da-yin-er-cha-shu-iii-lcof/" \t "_blank)

**题目描述：**请实现一个函数按照之字形顺序打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右到左的顺序打印，第三行再按照从左到右的顺序打印，其他行以此类推。

返回其层次遍历结果：

[

[3],

[20,9],

[15,7]

]

**参考答案：**

class Solution:

    def levelOrder(self, root: TreeNode) -> List[List[int]]:

        if not root: return []

        res, deque = [], collections.deque([root])

        while deque:

            tmp = collections.deque()

            for \_ in range(len(deque)):

                node = deque.popleft()

                if len(res) % 2: tmp.appendleft(node.val) # 偶数层 -> 队列头部

                else: tmp.append(node.val) # 奇数层 -> 队列尾部

                if node.left: deque.append(node.left)

                if node.right: deque.append(node.right)

            res.append(list(tmp))

        return res

# 23.剑指offer-34:二叉树和为某一值的路径

**题目链接：**[https://leetcode-cn.com/problems/er-cha-shu-zhong-he-wei-mou-yi-zhi-de-lu-jing-lcof/](https://leetcode-cn.com/problems/er-cha-shu-zhong-he-wei-mou-yi-zhi-de-lu-jing-lcof/" \t "_blank)

**题目描述：**输入一棵二叉树和一个整数，打印出二叉树中节点值的和为输入整数的所有路径。从树的根节点开始往下一直到叶节点所经过的节点形成一条路径。

**参考答案：**

class Solution:

    def pathSum(self, root: TreeNode, sum: int) -> List[List[int]]:

        res, path = [], []

        def recur(root, tar):

            if not root: return

            path.append(root.val)

            tar -= root.val

            if tar == 0 and not root.left and not root.right:

                res.append(list(path))

            recur(root.left, tar)

            recur(root.right, tar)

            path.pop()

        recur(root, sum)

        return res

# leetcode-124:二叉树中的最大路径和

**题目链接：**[https://leetcode-cn.com/problems/binary-tree-maximum-path-sum/](https://leetcode-cn.com/problems/binary-tree-maximum-path-sum/" \t "_blank)

**题目描述：**给定一个**非空**二叉树，返回其最大路径和。

本题中，路径被定义为一条从树中任意节点出发，达到任意节点的序列。该路径**至少包含一个**节点，且不一定经过根节点。

**示例 1:**

**输入:** [1,2,3]

**1**

**/ \**

**2** **3**

**输出:** 6

示例 2:

输入: [-10,9,20,null,null,15,7]

   -10

   / \

  9  20

    /  \

   15   7

输出: 42

**参考答案：**

class Solution:

    def \_\_init\_\_(self):

        self.maxSum = float("-inf")

    def maxPathSum(self, root: TreeNode) -> int:

        def maxGain(node):

            if not node:

                return 0

            # 递归计算左右子节点的最大贡献值

            # 只有在最大贡献值大于 0 时，才会选取对应子节点

            leftGain = max(maxGain(node.left), 0)

            rightGain = max(maxGain(node.right), 0)

            # 节点的最大路径和取决于该节点的值与该节点的左右子节点的最大贡献值

            priceNewpath = node.val + leftGain + rightGain

            # 更新答案

            self.maxSum = max(self.maxSum, priceNewpath)

            # 返回节点的最大贡献值

            return node.val + max(leftGain, rightGain)

        maxGain(root)

        return self.maxSum

# 25.leetcode-200:岛屿数量

**题目链接：**[https://leetcode-cn.com/problems/number-of-islands/](https://leetcode-cn.com/problems/number-of-islands/" \t "_blank)

**题目描述：**

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:

[

['1','1','1','1','0'],

['1','1','0','1','0'],

['1','1','0','0','0'],

['0','0','0','0','0']

]

输出: 1

示例 2:

输入:

[

['1','1','0','0','0'],

['1','1','0','0','0'],

['0','0','1','0','0'],

['0','0','0','1','1']

]

输出: 3

解释: 每座岛屿只能由水平和/或竖直方向上相邻的陆地连接而成。

**参考答案：**

class Solution:

    def dfs(self, grid, r, c):

        grid[r][c] = 0

        nr, nc = len(grid), len(grid[0])

        for x, y in [(r - 1, c), (r + 1, c), (r, c - 1), (r, c + 1)]:

            if 0 <= x < nr and 0 <= y < nc and grid[x][y] == "1":

                self.dfs(grid, x, y)

    def numIslands(self, grid: List[List[str]]) -> int:

        nr = len(grid)

        if nr == 0:

            return 0

        nc = len(grid[0])

        num\_islands = 0

        for r in range(nr):

            for c in range(nc):

                if grid[r][c] == "1":

                    num\_islands += 1

                    self.dfs(grid, r, c)

        return num\_islands

# 26.leetcode-529:扫雷游戏

**题目链接：**[https://leetcode-cn.com/problems/minesweeper/](https://leetcode-cn.com/problems/minesweeper/" \t "_blank)

**题目描述：**

让我们一起来玩扫雷游戏！

给定一个代表游戏板的二维字符矩阵。 'M' 代表一个未挖出的地雷，'E' 代表一个未挖出的空方块，'B' 代表没有相邻（上，下，左，右，和所有4个对角线）地雷的已挖出的空白方块，数字（'1' 到 '8'）表示有多少地雷与这块已挖出的方块相邻，'X' 则表示一个已挖出的地雷。

现在给出在所有未挖出的方块中（'M'或者'E'）的下一个点击位置（行和列索引），根据以下规则，返回相应位置被点击后对应的面板：

    如果一个地雷（'M'）被挖出，游戏就结束了- 把它改为 'X'。

    如果一个没有相邻地雷的空方块（'E'）被挖出，修改它为（'B'），并且所有和其相邻的方块都应该被递归地揭露。

    如果一个至少与一个地雷相邻的空方块（'E'）被挖出，修改它为数字（'1'到'8'），表示相邻地雷的数量。

    如果在此次点击中，若无更多方块可被揭露，则返回面板。

示例 1：

输入:

[['E', 'E', 'E', 'E', 'E'],

 ['E', 'E', 'M', 'E', 'E'],

 ['E', 'E', 'E', 'E', 'E'],

 ['E', 'E', 'E', 'E', 'E']]

Click : [3,0]

输出:

[['B', '1', 'E', '1', 'B'],

 ['B', '1', 'M', '1', 'B'],

 ['B', '1', '1', '1', 'B'],

 ['B', 'B', 'B', 'B', 'B']]

示例 2：

输入:

[['B', '1', 'E', '1', 'B'],

 ['B', '1', 'M', '1', 'B'],

 ['B', '1', '1', '1', 'B'],

 ['B', 'B', 'B', 'B', 'B']]

Click : [1,2]

输出:

[['B', '1', 'E', '1', 'B'],

 ['B', '1', 'X', '1', 'B'],

 ['B', '1', '1', '1', 'B'],

 ['B', 'B', 'B', 'B', 'B']]

**参考答案：**

class Solution:

    def updateBoard(self, board: List[List[str]], click: List[int]) -> List[List[str]]:

        i, j = click

        row, col = len(board), len(board[0])

        if board[i][j] == "M":

            board[i][j] = "X"

            return board

        # 计算空白快周围的\*\*\*

        def cal(i, j):

            res = 0

            for x in [1, -1, 0]:

                for y in [1, -1, 0]:

                    if x == 0 and y == 0: continue

                    if 0 <= i + x < row and 0 <= j + y < col and board[i + x][j + y] == "M": res += 1

            return res

        def dfs(i, j):

            num = cal(i, j)

            if num > 0:

                board[i][j] = str(num)

                return

            board[i][j] = "B"

            for x in [1, -1, 0]:

                for y in [1, -1, 0]:

                    if x == 0 and y == 0: continue

                    nxt\_i, nxt\_j = i + x, j + y

                    if 0 <= nxt\_i < row and 0 <= nxt\_j < col and board[nxt\_i][nxt\_j] == "E": dfs(nxt\_i, nxt\_j)

        dfs(i, j)

        return board

# 27.leetcode-98: 验证二叉搜索树

**题目链接：**[https://leetcode-cn.com/problems/validate-binary-search-tree/](https://leetcode-cn.com/problems/validate-binary-search-tree/" \t "_blank)

**题目描述：**

给定一个二叉树，判断其是否是一个有效的二叉搜索树。

假设一个二叉搜索树具有如下特征：

    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:

    2

   / \

  1   3

输出: true

示例 2:

输入:

    5

   / \

  1   4

     / \

    3   6

输出: false

解释: 输入为: [5,1,4,null,null,3,6]。

     根节点的值为 5 ，但是其右子节点值为 4 。

**参考答案：**

class Solution:

    def isValidBST(self, root):

        """

        :type root: TreeNode

        :rtype: bool

        """

        def helper(node, lower = float('-inf'), upper = float('inf')):

            if not node:

                return True

            val = node.val

            if val <= lower or val >= upper:

                return False

            if not helper(node.right, val, upper):

                return False

            if not helper(node.left, lower, val):

                return False

            return True

        return helper(root)

# 28.剑指offer 36.二叉搜索树与双向链表

**题目链接：**[https://leetcode-cn.com/problems/er-cha-sou-suo-shu-yu-shuang-xiang-lian-biao-lcof/](https://leetcode-cn.com/problems/er-cha-sou-suo-shu-yu-shuang-xiang-lian-biao-lcof/" \t "_blank)

**题目描述：**输入一棵二叉搜索树，将该二叉搜索树转换成一个排序的循环双向链表。要求不能创建任何新的节点，只能调整树中节点指针的指向。

**参考答案：**

class Solution:

    def treeToDoublyList(self, root: 'Node') -> 'Node':

        def dfs(cur):

            if not cur: return

            dfs(cur.left) # 递归左子树

            if self.pre: # 修改节点引用

                self.pre.right, cur.left = cur, self.pre

            else: # 记录头节点

                self.head = cur

            self.pre = cur # 保存 cur

            dfs(cur.right) # 递归右子树

        if not root: return

        self.pre = None

        dfs(root)

        self.head.left, self.pre.right = self.pre, self.head

        return self.head

# 29.剑指offer 54:二叉搜索树的第K大节点

**题目链接：**[https://leetcode-cn.com/problems/er-cha-sou-suo-shu-de-di-kda-jie-dian-lcof/](https://leetcode-cn.com/problems/er-cha-sou-suo-shu-de-di-kda-jie-dian-lcof/" \t "_blank)

**题目描述：**给定一棵二叉搜索树，请找出其中第k大的节点。

示例 1:

输入: root = [3,1,4,null,2], k = 1

   3

  / \

 1   4

  \

   2

输出: 4

示例 2:

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

       5

      / \

     3   6

    / \

   2   4

  /

 1

输出: 4

**参考答案：**

class Solution:

    def kthLargest(self, root: TreeNode, k: int) -> int:

        def dfs(root):

            if not root: return

            dfs(root.right)

            if self.k == 0: return

            self.k -= 1

            if self.k == 0: self.res = root.val

            dfs(root.left)

        self.k = k

        dfs(root)

        return self.res

# 30.leetcode-98:验证二叉搜索树

**题目链接：**[https://leetcode-cn.com/problems/validate-binary-search-tree/](https://leetcode-cn.com/problems/validate-binary-search-tree/" \t "_blank)

**题目描述：**

给定一个二叉树，判断其是否是一个有效的二叉搜索树。

假设一个二叉搜索树具有如下特征：

    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:

    2

   / \

  1   3

输出: true

示例 2:

输入:

    5

   / \

  1   4

     / \

    3   6

输出: false

解释: 输入为: [5,1,4,null,null,3,6]。

     根节点的值为 5 ，但是其右子节点值为 4 。

**参考答案：**

class Solution:

    def isValidBST(self, root):

        """

        :type root: TreeNode

        :rtype: bool

        """

        def helper(node, lower = float('-inf'), upper = float('inf')):

            if not node:

                return True

            val = node.val

            if val <= lower or val >= upper:

                return False

            if not helper(node.right, val, upper):

                return False

            if not helper(node.left, lower, val):

                return False

            return True

        return helper(root)

# leetcode-111:二叉树的最小深度

**题目链接：**[https://leetcode-cn.com/problems/minimum-depth-of-binary-tree/](https://leetcode-cn.com/problems/minimum-depth-of-binary-tree/" \t "_blank)

**题目描述：**

给定一个二叉树，找出其最小深度。

最小深度是从根节点到最近叶子节点的最短路径上的节点数量。

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

返回它的最小深度  2.

**参考答案：**

class Solution:

    def minDepth(self, root):

        """

        :type root: TreeNode

        :rtype: int

        """

        if not root:

            return 0

        children = [root.left, root.right]

        # if we're at leaf node

        if not any(children):

            return 1

        min\_depth = float('inf')

        for c in children:

            if c:

                min\_depth = min(self.minDepth(c), min\_depth)

        return min\_depth + 1

# 32.leetcode-130:被围绕的区域

**题目链接：**[https://leetcode-cn.com/problems/surrounded-regions/](https://leetcode-cn.com/problems/surrounded-regions/" \t "_blank)

**题目描述：**

给定一个二维的矩阵，包含 'X' 和 'O'（字母 O）。

找到所有被 'X' 围绕的区域，并将这些区域里所有的 'O' 用 'X' 填充。

示例:

X X X X

X O O X

X X O X

X O X X

运行你的函数后，矩阵变为：

X X X X

X X X X

X X X X

X O X X

**参考答案：**

class Solution:

    def solve(self, board: List[List[str]]) -> None:

        """

        Do not return anything, modify board in-place instead.

        """

        if not board or not board[0]:

            return

        row = len(board)

        col = len(board[0])

        def dfs(i, j):

            board[i][j] = "B"

            for x, y in [(-1, 0), (1, 0), (0, -1), (0, 1)]:

                tmp\_i = i + x

                tmp\_j = j + y

                if 1 <= tmp\_i < row and 1 <= tmp\_j < col and board[tmp\_i][tmp\_j] == "O":

                    dfs(tmp\_i, tmp\_j)

        for j in range(col):

            # 第一行

            if board[0][j] == "O":

                dfs(0, j)

            # 最后一行

            if board[row - 1][j] == "O":

                dfs(row - 1, j)

        for i in range(row):

            # 第一列

            if board[i][0] == "O":

                dfs(i, 0)

            # 最后一列

            if board[i][col-1] == "O":

                dfs(i, col - 1)

        for i in range(row):

            for j in range(col):

                # O 变成 X

                if board[i][j] == "O":

                    board[i][j] = "X"

                # B 变成 O

                if board[i][j] == "B":

                    board[i][j] = "O"

# 33.leetcode-130: 被围绕的区域

**题目链接：**[https://leetcode-cn.com/problems/surrounded-regions/](https://leetcode-cn.com/problems/surrounded-regions/" \t "_blank)

**题目描述：**

给定一个二维的矩阵，包含 'X' 和 'O'（字母 O）。

找到所有被 'X' 围绕的区域，并将这些区域里所有的 'O' 用 'X' 填充。

示例:

X X X X

X O O X

X X O X

X O X X

运行你的函数后，矩阵变为：

X X X X

X X X X

X X X X

X O X X

解释:

被围绕的区间不会存在于边界上，换句话说，任何边界上的 'O' 都不会被填充为 'X'。 任何不在边界上，或不与边界上的 'O' 相连的 'O' 最终都会被填充为 'X'。如果两个元素在水平或垂直方向相邻，则称它们是“相连”的。

**参考答案：**

class Solution:

    def solve(self, board: List[List[str]]) -> None:

        """

        Do not return anything, modify board in-place instead.

        """

        if not board or not board[0]:

            return

        row = len(board)

        col = len(board[0])

        def bfs(i, j):

            from collections import deque

            queue = deque()

            queue.appendleft((i, j))

            while queue:

                i, j = queue.pop()

                if 0 <= i < row and 0 <= j < col and board[i][j] == "O":

                    board[i][j] = "B"

                    for x, y in [(-1, 0), (1, 0), (0, -1), (0, 1)]:

                        queue.appendleft((i + x, j + y))

        for j in range(col):

            # 第一行

            if board[0][j] == "O":

                bfs(0, j)

            # 最后一行

            if board[row - 1][j] == "O":

                bfs(row - 1, j)

        for i in range(row):

            if board[i][0] == "O":

                bfs(i, 0)

            if board[i][col - 1] == "O":

                bfs(i, col - 1)

        for i in range(row):

            for j in range(col):

                if board[i][j] == "O":

                    board[i][j] = "X"

                if board[i][j] == "B":

                    board[i][j] = "O"

# 34.leetcode-200:岛屿数量

**题目链接：**[https://leetcode-cn.com/problems/surrounded-regions/](https://leetcode-cn.com/problems/surrounded-regions/" \t "_blank)

**题目描述：**

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:

[

['1','1','1','1','0'],

['1','1','0','1','0'],

['1','1','0','0','0'],

['0','0','0','0','0']

]

输出: 1

示例 2:

输入:

[

['1','1','0','0','0'],

['1','1','0','0','0'],

['0','0','1','0','0'],

['0','0','0','1','1']

]

输出: 3

解释: 每座岛屿只能由水平和/或竖直方向上相邻的陆地连接而成。

**参考答案：**

class Solution:

    def numIslands(self, grid: List[List[str]]) -> int:

        nr = len(grid)

        if nr == 0:

            return 0

        nc = len(grid[0])

        num\_islands = 0

        for r in range(nr):

            for c in range(nc):

                if grid[r][c] == "1":

                    num\_islands += 1

                    grid[r][c] = "0"

                    neighbors = collections.deque([(r, c)])

                    while neighbors:

                        row, col = neighbors.popleft()

                        for x, y in [(row - 1, col), (row + 1, col), (row, col - 1), (row, col + 1)]:

                            if 0 <= x < nr and 0 <= y < nc and grid[x][y] == "1":

                                neighbors.append((x, y))

                                grid[x][y] = "0"

        return num\_islands

# 35.leetcode-994:腐烂的橘子

**题目链接：**[https://leetcode-cn.com/problems/rotting-oranges/](https://leetcode-cn.com/problems/rotting-oranges/" \t "_blank)

**题目描述：**

在给定的网格中，每个单元格可以有以下三个值之一：

    值 0 代表空单元格；

    值 1 代表新鲜橘子；

    值 2 代表腐烂的橘子。

每分钟，任何与腐烂的橘子（在 4 个正方向上）相邻的新鲜橘子都会腐烂。

返回直到单元格中没有新鲜橘子为止所必须经过的最小分钟数。如果不可能，返回 -1。

示例 1：

输入：[[2,1,1],[1,1,0],[0,1,1]]

输出：4

示例 2：

输入：[[2,1,1],[0,1,1],[1,0,1]]

输出：-1

解释：左下角的橘子（第 2 行， 第 0 列）永远不会腐烂，因为腐烂只会发生在 4 个正向上。

示例 3：

输入：[[0,2]]

输出：0

解释：因为 0 分钟时已经没有新鲜橘子了，所以答案就是 0 。

**参考答案：**

class Solution(object):

    def orangesRotting(self, grid):

        R, C = len(grid), len(grid[0])

        # queue - all starting cells with rotting oranges

        queue = collections.deque()

        for r, row in enumerate(grid):

            for c, val in enumerate(row):

                if val == 2:

                    queue.append((r, c, 0))

        def neighbors(r, c):

            for nr, nc in ((r-1,c),(r,c-1),(r+1,c),(r,c+1)):

                if 0 <= nr < R and 0 <= nc < C:

                    yield nr, nc

        d = 0

        while queue:

            r, c, d = queue.popleft()

            for nr, nc in neighbors(r, c):

                if grid[nr][nc] == 1:

                    grid[nr][nc] = 2

                    queue.append((nr, nc, d+1))

        if any(1 in row for row in grid):

            return -1

        return d

# 36.leetcode-199:二叉树右视图

**题目链接：**[https://leetcode-cn.com/problems/binary-tree-right-side-view/](https://leetcode-cn.com/problems/binary-tree-right-side-view/" \t "_blank)

**题目描述：**

给定一棵二叉树，想象自己站在它的右侧，按照从顶部到底部的顺序，返回从右侧所能看到的节点值。

示例:

输入: [1,2,3,null,5,null,4]

输出: [1, 3, 4]

解释:

   1            <---

 /   \

2     3         <---

 \     \

  5     4       <---

**参考答案：**

class Solution(object):

    def rightSideView(self, root):

        rightmost\_value\_at\_depth = dict() # 深度为索引，存放节点的值

        max\_depth = -1

        stack = [(root, 0)]

        while stack:

            node, depth = stack.pop()

            if node is not None:

                # 维护二叉树的最大深度

                max\_depth = max(max\_depth, depth)

                # 如果不存在对应深度的节点我们才插入

                rightmost\_value\_at\_depth.setdefault(depth, node.val)

                stack.append((node.left, depth+1))

                stack.append((node.right, depth+1))

        return [rightmost\_value\_at\_depth[depth] for depth in range(max\_depth+1)]

# 37.leetcode-112:路径总和

**题目链接：**[https://leetcode-cn.com/problems/path-sum/](https://leetcode-cn.com/problems/path-sum/" \t "_blank)

**题目描述：**

给定一个二叉树和一个目标和，判断该树中是否存在根节点到叶子节点的路径，这条路径上所有节点值相加等于目标和。

说明: 叶子节点是指没有子节点的节点。

示例:

给定如下二叉树，以及目标和 sum = 22，

              5

             / \

            4   8

           /   / \

          11  13  4

         /  \      \

        7    2      1

返回 true, 因为存在目标和为 22 的根节点到叶子节点的路径 5->4->11->2。

**参考答案：**

class Solution:

    def hasPathSum(self, root: TreeNode, sum: int) -> bool:

        if not root:

            return False

        que\_node = collections.deque([root])

        que\_val = collections.deque([root.val])

        while que\_node:

            now = que\_node.popleft()

            temp = que\_val.popleft()

            if not now.left and not now.right:

                if temp == sum:

                    return True

                continue

            if now.left:

                que\_node.append(now.left)

                que\_val.append(now.left.val + temp)

            if now.right:

                que\_node.append(now.right)

                que\_val.append(now.right.val + temp)

        return False

# 38.leetcode-143:重排链表

**题目链接：**[https://leetcode-cn.com/problems/reorder-list/](https://leetcode-cn.com/problems/reorder-list/" \t "_blank)

**题目描述：**

给定一个单链表 L：L0→L1→…→Ln-1→Ln ，

将其重新排列后变为： L0→Ln→L1→Ln-1→L2→Ln-2→…

你不能只是单纯的改变节点内部的值，而是需要实际的进行节点交换。

示例 1:

给定链表 1->2->3->4, 重新排列为 1->4->2->3.

示例 2:

给定链表 1->2->3->4->5, 重新排列为 1->5->2->4->3.

**参考答案：**

class Solution:

    def reorderList(self, head: ListNode) -> None:

        """

        Do not return anything, modify head in-place instead.

        """

        if not head or not head.next:

            return head

        # 找中点

        slow,first = head,head

        while first.next and first.next.next:

            first = first.next.next

            slow = slow.next

        # 断开链表

        head2 = slow.next

        slow.next = None

        # 翻转中点后边的链表，

        # 因为单链表无前指针，所以定义双指针

        # 采用所谓的头插法

        cur\_node = head2

        pre\_node = None

        next\_node = None

        while cur\_node != None:

            next\_node = cur\_node.next

            cur\_node.next = pre\_node

            pre\_node = cur\_node

            cur\_node = next\_node

        # 连接两个链表

        # 用链表二往链表一里插

        head2 = pre\_node

        while head2:

            next1 = head.next

            next2 = head2.next

            head2.next = head.next

            head.next = head2

            head = next1

            head2 = next2

        return head

# 39.leetcode-508:出现次数最多的子树元素和

**题目链接：**[https://leetcode-cn.com/problems/most-frequent-subtree-sum/](https://leetcode-cn.com/problems/most-frequent-subtree-sum/" \t "_blank)

**题目描述：**

给你一个二叉树的根结点，请你找出出现次数最多的子树元素和。一个结点的「子树元素和」定义为以该结点为根的二叉树上所有结点的元素之和（包括结点本身）。

你需要返回出现次数最多的子树元素和。如果有多个元素出现的次数相同，返回所有出现次数最多的子树元素和（不限顺序）。

示例 1：

输入:

  5

 /  \

2   -3

返回 [2, -3, 4]，所有的值均只出现一次，以任意顺序返回所有值。

示例 2：

输入：

  5

 /  \

2   -5

返回 [2]，只有 2 出现两次，-5 只出现 1 次。

提示： 假设任意子树元素和均可以用 32 位有符号整数表示。

**参考答案：**

class Solution:

    def findFrequentTreeSum(self, root: TreeNode) -> List[int]:

        if not root:

            return []

        hashmap = collections.defaultdict(int)

        def helper(root):

            if not root.left and not root.right:

                hashmap[root.val] += 1

                return root.val

            left = helper(root.left) if root.left else 0

            right = helper(root.right) if root.right else 0

            tmp = left + right + root.val

            hashmap[tmp] += 1

            return tmp

        helper(root)

        result, tmp = [], 0

        for key, val in hashmap.items():

            if val == tmp:

                result.append(key)

            elif val > tmp:

                tmp = val

                result = [key]

        return result

# 40.leetcode-257:二叉树的所有路径

**题目链接：**[https://leetcode-cn.com/problems/binary-tree-paths/](https://leetcode-cn.com/problems/binary-tree-paths/" \t "_blank)

**题目描述：**

给定一个二叉树，返回所有从根节点到叶子节点的路径。

说明: 叶子节点是指没有子节点的节点。

示例:

输入:

   1

 /   \

2     3

 \

  5

输出: ["1->2->5", "1->3"]

解释: 所有根节点到叶子节点的路径为: 1->2->5, 1->3

**参考答案：**

class Solution:

    def binaryTreePaths(self, root):

        """

        :type root: TreeNode

        :rtype: List[str]

        """

        def construct\_paths(root, path):

            if root:

                path += str(root.val)

                if not root.left and not root.right:  # 当前节点是叶子节点

                    paths.append(path)  # 把路径加入到答案中

                else:

                    path += '->'  # 当前节点不是叶子节点，继续递归遍历

                    construct\_paths(root.left, path)

                    construct\_paths(root.right, path)

        paths = []

        construct\_paths(root, '')

        return paths

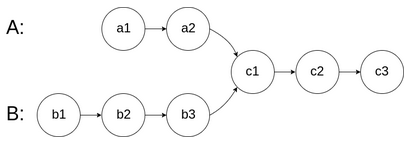
# 41.leetcode-160:相交链表

**题目链接：**[https://leetcode-cn.com/problems/intersection-of-two-linked-lists/](https://leetcode-cn.com/problems/intersection-of-two-linked-lists/" \t "_blank)

**题目描述：**

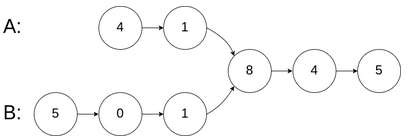
编写一个程序，找到两个单链表相交的起始节点。

如下面的两个链表：



在节点 c1 开始相交。

示例 1：

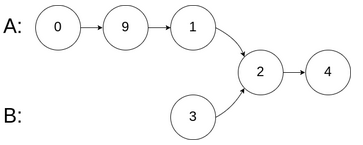


输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB = 3

输出：Reference of the node with value = 8

输入解释：相交节点的值为 8 （注意，如果两个链表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。

示例 2：

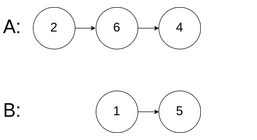


输入：intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1

输出：Reference of the node with value = 2

输入解释：相交节点的值为 2 （注意，如果两个链表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [0,9,1,2,4]，链表 B 为 [3,2,4]。在 A 中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。

示例 3：



输入：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2

输出：null

输入解释：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。

解释：这两个链表不相交，因此返回 null。

注意：

    如果两个链表没有交点，返回 null.

    在返回结果后，两个链表仍须保持原有的结构。

    可假定整个链表结构中没有循环。

    程序尽量满足 O(n) 时间复杂度，且仅用 O(1) 内存。

**参考答案：**

class Solution(object):

    def getIntersectionNode(self, headA, headB):

        ha, hb = headA, headB

        while ha != hb:

            ha = ha.next if ha else headB

            hb = hb.next if hb else headA

        return ha

# 42.leetcode-543：二叉树的直径

**题目链接：**[https://leetcode-cn.com/problems/diameter-of-binary-tree/](https://leetcode-cn.com/problems/diameter-of-binary-tree/" \t "_blank)

**题目描述：**

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。

**参考答案：**

class Solution(object):

    def diameterOfBinaryTree(self, root):

        self.ans = 1

        def depth(node):

            # 访问到空节点了，返回0

            if not node: return 0

            # 左儿子为根的子树的深度

            L = depth(node.left)

            # 右儿子为根的子树的深度

            R = depth(node.right)

            # 计算d\_node即L+R+1 并更新ans

            self.ans = max(self.ans, L+R+1)

            # 返回该节点为根的子树的深度

            return max(L, R) + 1

        depth(root)

        return self.ans - 1

# 43.leetcode-445:两数相加 II

**题目链接：**[https://leetcode-cn.com/problems/add-two-numbers-ii/](https://leetcode-cn.com/problems/add-two-numbers-ii/" \t "_blank)

**题目描述：**

给你两个 非空 链表来代表两个非负整数。数字最高位位于链表开始位置。它们的每个节点只存储一位数字。将这两数相加会返回一个新的链表。

你可以假设除了数字 0 之外，这两个数字都不会以零开头。

进阶：

如果输入链表不能修改该如何处理？换句话说，你不能对列表中的节点进行翻转。

示例：

输入：(7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)

输出：7 -> 8 -> 0 -> 7

**参考答案：**

class Solution:

    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:

        s1, s2 = [], []

        while l1:

            s1.append(l1.val)

            l1 = l1.next

        while l2:

            s2.append(l2.val)

            l2 = l2.next

        ans = None

        carry = 0

        while s1 or s2 or carry != 0:

            a = 0 if not s1 else s1.pop()

            b = 0 if not s2 else s2.pop()

            cur = a + b + carry

            carry = cur // 10

            cur %= 10

            curnode = ListNode(cur)

            curnode.next = ans

            ans = curnode

        return ans

# 44.leetcode-337:打家劫舍III

**题目链接：**[https://leetcode-cn.com/problems/house-robber-iii/](https://leetcode-cn.com/problems/house-robber-iii/" \t "_blank)

**题目描述：**

在上次打劫完一条街道之后和一圈房屋后，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“根”。 除了“根”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。

计算在不触动警报的情况下，小偷一晚能够盗取的最高金额。

示例 1:

输入: [3,2,3,null,3,null,1]

     3

    / \

   2   3

    \   \

     3   1

输出: 7

解释: 小偷一晚能够盗取的最高金额 = 3 + 3 + 1 = 7.

示例 2:

输入: [3,4,5,1,3,null,1]

     3

    / \

   4   5

  / \   \

 1   3   1

输出: 9

解释: 小偷一晚能够盗取的最高金额 = 4 + 5 = 9.

**参考答案：**

class Solution(object):

    def rob(self, root):

        if not root:

            return 0

        def dfs(root,status):

            if not root:

                return 0

            a,b,c = 0,0,0

            # 不管父节点选择偷/不偷，本次什么都不干

            a = dfs(root.left,status)+dfs(root.right,status)

            # 父节点选择偷，本次不偷

            if status:

                b = dfs(root.left,0)+dfs(root.right,0)

            # 父节点选择不偷，本次选择偷

            else:

                c = dfs(root.left,1)+dfs(root.right,1)+root.val

            # 返回三种状态的最大值

            return max(a,b,c)

        return dfs(root,0)

# 45.leetcode-147:对链表进行插入排序

**题目链接：**[https://leetcode-cn.com/problems/insertion-sort-list/](https://leetcode-cn.com/problems/insertion-sort-list/" \t "_blank)

**题目描述：**

插入排序算法：

    插入排序是迭代的，每次只移动一个元素，直到所有元素可以形成一个有序的输出列表。

    每次迭代中，插入排序只从输入数据中移除一个待排序的元素，找到它在序列中适当的位置，并将其插入。

    重复直到所有输入数据插入完为止。

示例 1：

输入: 4->2->1->3

输出: 1->2->3->4

示例 2：

输入: -1->5->3->4->0

输出: -1->0->3->4->5

**参考答案：**

# Definition for singly-linked list.

class ListNode(object):

    def \_\_init\_\_(self, x):

        self.val = x

        self.next = None

class Solution(object):

    def insertionSortList(self, head):

        """

        :type head: ListNode

        :rtype: ListNode

        """

        fhead = ListNode(float('-Inf') )

        fhead.next = head

        pcur = fhead

        cur = head

        while cur:

            if pcur.val <= cur.val:

                pcur = pcur.next

                cur = pcur.next

                continue

            pcur.next = cur.next

            cur.next = None

            p = fhead

            while p.next and p.next.val <= cur.val:

                p = p.next

            cur.next = p.next

            p.next = cur

            cur = pcur.next

        return fhead.next

# 46.leetcode-103:二叉树锯齿形层次遍历

**题目链接：**[https://leetcode-cn.com/problems/binary-tree-zigzag-level-order-traversal/](https://leetcode-cn.com/problems/binary-tree-zigzag-level-order-traversal/" \t "_blank)

**题目描述：**

给定一个二叉树，返回其节点值的锯齿形层次遍历。（即先从左往右，再从右往左进行下一层遍历，以此类推，层与层之间交替进行）。

例如：

给定二叉树 [3,9,20,null,null,15,7],

    3

   / \

  9  20

    /  \

   15   7

返回锯齿形层次遍历如下：

[

  [3],

  [20,9],

  [15,7]

]

**参考答案：**

# Definition for a binary tree node.

# class TreeNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.left = None

#         self.right = None

from collections import deque

class Solution:

    def zigzagLevelOrder(self, root):

        """

        :type root: TreeNode

        :rtype: List[List[int]]

        """

        ret = []

        level\_list = deque()

        if root is None:

            return []

        # start with the level 0 with a delimiter

        node\_queue = deque([root, None])

        is\_order\_left = True

        while len(node\_queue) > 0:

            curr\_node = node\_queue.popleft()

            if curr\_node:

                if is\_order\_left:

                    level\_list.append(curr\_node.val)

                else:

                    level\_list.appendleft(curr\_node.val)

                if curr\_node.left:

                    node\_queue.append(curr\_node.left)

                if curr\_node.right:

                    node\_queue.append(curr\_node.right)

            else:

                # we finish one level

                ret.append(level\_list)

                # add a delimiter to mark the level

                if len(node\_queue) > 0:

                    node\_queue.append(None)

                # prepare for the next level

                level\_list = deque()

                is\_order\_left = not is\_order\_left

        return ret

# 面试题02.06:回文链表

**题目链接：**[https://leetcode-cn.com/problems/palindrome-linked-list-lcci/](https://leetcode-cn.com/problems/palindrome-linked-list-lcci/" \t "_blank)

**题目描述：**

编写一个函数，检查输入的链表是否是回文的。

示例 1：

输入： 1->2

输出： false

示例 2：

输入： 1->2->2->1

输出： true

**参考答案：**

# Definition for singly-linked list.

# class ListNode(object):

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.next = None

class Solution(object):

    def isPalindrome(self, head):

        """

        :type head: ListNode

        :rtype: bool

        """

        if not head: return True

        slow = head

        fast = head

        # slow 遍历到中间，最后 slow 停的位置是 n/2+1

        while fast and fast.next:

            slow = slow.next

            fast = fast.next.next

        # 然后从 slow 开始反转链表

        # 如果是奇数，多出来的一个节点不用管了

        pre = slow

        while slow and slow.next:

            tmp = slow.next.next

            slow.next.next = pre

            pre = slow.next

            slow.next = tmp

        # 反转后的头结点是 pre，从 pre 和 head 开始比较 val

        while head and pre:

            if head.val != pre.val:

                return False

            head = head.next

            pre = pre.next

        return True

# 48.leetcode 654:最大二叉树

**题目链接：**[https://leetcode-cn.com/problems/maximum-binary-tree/](https://leetcode-cn.com/problems/maximum-binary-tree/" \t "_blank)

**题目描述：**

给定一个不含重复元素的整数数组。一个以此数组构建的最大二叉树定义如下：

    二叉树的根是数组中的最大元素。

    左子树是通过数组中最大值左边部分构造出的最大二叉树。

    右子树是通过数组中最大值右边部分构造出的最大二叉树。

通过给定的数组构建最大二叉树，并且输出这个树的根节点。

示例 ：

输入：[3,2,1,6,0,5]

输出：返回下面这棵树的根节点：

      6

    /   \

   3     5

    \    /

     2  0

       \

        1

提示：

    给定的数组的大小在 [1, 1000] 之间。

**参考答案：**

# Definition for a binary tree node.

# class TreeNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.left = None

#         self.right = None

class Solution:

    def constructMaximumBinaryTree(self, nums: List[int]) -> TreeNode:

        if not nums: return

        max\_num = max(nums)

        index = nums.index(max\_num) #找出最大值元素所在位置

        root = TreeNode(max\_num) #构建根节点

        root.left = self.constructMaximumBinaryTree(nums[:index]) #左边元素构建左子树

        root.right = self.constructMaximumBinaryTree(nums[index+1:]) #右边元素构建右子树

        return root

# 49.leetcode-1019:链表中的下一个更大节点

**题目链接：**[https://leetcode-cn.com/problems/next-greater-node-in-linked-list/](https://leetcode-cn.com/problems/next-greater-node-in-linked-list/" \t "_blank)

**题目描述：**

给出一个以头节点 head 作为第一个节点的链表。链表中的节点分别编号为：node\_1, node\_2, node\_3, ... 。

每个节点都可能有下一个更大值（next larger value）：对于 node\_i，如果其 next\_larger(node\_i) 是 node\_j.val，那么就有 j > i 且  node\_j.val > node\_i.val，而 j 是可能的选项中最小的那个。如果不存在这样的 j，那么下一个更大值为 0 。

返回整数答案数组 answer，其中 answer[i] = next\_larger(node\_{i+1}) 。

注意：在下面的示例中，诸如 [2,1,5] 这样的输入（不是输出）是链表的序列化表示，其头节点的值为 2，第二个节点值为 1，第三个节点值为 5 。

示例 1：

输入：[2,1,5]

输出：[5,5,0]

示例 2：

输入：[2,7,4,3,5]

输出：[7,0,5,5,0]

示例 3：

输入：[1,7,5,1,9,2,5,1]

输出：[7,9,9,9,0,5,0,0]

提示：

    对于链表中的每个节点，1 <= node.val <= 10^9

    给定列表的长度在 [0, 10000] 范围内

**参考答案：**

# Definition for singly-linked list.

# class ListNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.next = None

class Solution:

    def nextLargerNodes(self, head: ListNode) -> List[int]:

        nums = []

        p = head

        while type(p) != type(None):

            nums.append(p.val)

            p = p.next

        ans = [0 for i in range(len(nums))]

        if (len(nums)) == 0:

            return ans

        dq = [[nums[0],0]]

        p1 = 0

        p2 = 1

        for i in range(1, len(nums)):

            while p2 -1 >= 0 and dq[p2 - 1][0] < nums[i]:

                p2 -= 1

                ans[dq[p2][1]] = nums[i]

            if len(dq) == p2:

                dq.append([nums[i], i])

                p2 += 1

            else:

                dq[p2] = [nums[i],i]

                p2 += 1

        return ans

# 50.leetcode-100:相同的树

**题目链接：**[https://leetcode-cn.com/problems/same-tree/](https://leetcode-cn.com/problems/same-tree/" \t "_blank)

**题目描述：**

给定两个二叉树，编写一个函数来检验它们是否相同。

如果两个树在结构上相同，并且节点具有相同的值，则认为它们是相同的。

示例 1:

输入:       1         1

          / \       / \

         2   3     2   3

        [1,2,3],   [1,2,3]

输出: true

示例 2:

输入:      1          1

          /           \

         2             2

        [1,2],     [1,null,2]

输出: false

示例 3:

输入:       1         1

          / \       / \

         2   1     1   2

        [1,2,1],   [1,1,2]

输出: false

**参考答案：**

class Solution:

    def isSameTree(self, p, q):

        """

        :type p: TreeNode

        :type q: TreeNode

        :rtype: bool

        """

        # p and q are both None

        if not p and not q:

            return True

        # one of p and q is None

        if not q or not p:

            return False

        if p.val != q.val:

            return False

        return self.isSameTree(p.right, q.right) and \

               self.isSameTree(p.left, q.left)

# 51.leetcode-1290:二进制链表转整数

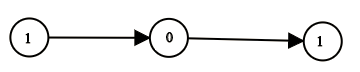
**题目链接：**[https://leetcode-cn.com/problems/convert-binary-number-in-a-linked-list-to-integer/](https://leetcode-cn.com/problems/convert-binary-number-in-a-linked-list-to-integer/" \t "_blank)

**题目描述：**

给你一个单链表的引用结点 head。链表中每个结点的值不是 0 就是 1。已知此链表是一个整数数字的二进制表示形式。

请你返回该链表所表示数字的 十进制值 。

示例1：



**输入：**head = [1,0,1]

**输出：**5

**解释：**二进制数 (101) 转化为十进制数 (5)

示例 2：

输入：head = [0]

输出：0

示例 3：

输入：head = [1]

输出：1

示例 4：

输入：head = [1,0,0,1,0,0,1,1,1,0,0,0,0,0,0]

输出：18880

示例 5：

输入：head = [0,0]

输出：0

提示：

    链表不为空。

    链表的结点总数不超过 30。

    每个结点的值不是 0 就是 1。

**参考答案：**

class Solution:

    def getDecimalValue(self, head: ListNode) -> int:

        cur = head

        ans = 0

        while cur:

            ans = ans \* 2 + cur.val

            cur = cur.next

        return ans

# 52.剑指offer 10-II:青蛙跳台阶问题

**题目链接：**[https://leetcode-cn.com/problems/qing-wa-tiao-tai-jie-wen-ti-lcof/](https://leetcode-cn.com/problems/qing-wa-tiao-tai-jie-wen-ti-lcof/" \t "_blank)

**题目描述：**

一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个 n 级的台阶总共有多少种跳法。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1：

输入：n = 2

输出：2

示例 2：

输入：n = 7

输出：21

提示：

    0 <= n <= 100

**参考答案：**

class Solution:

    def numWays(self, n: int) -> int:

        a, b = 1, 1

        for \_ in range(n):

            a, b = b, a + b

        return a % 1000000007

# 53.面试题 08.01.三步问题

**题目链接：**[https://leetcode-cn.com/problems/three-steps-problem-lcci/](https://leetcode-cn.com/problems/three-steps-problem-lcci/" \t "_blank)

**题目描述：**

三步问题。有个小孩正在上楼梯，楼梯有n阶台阶，小孩一次可以上1阶、2阶或3阶。实现一种方法，计算小孩有多少种上楼梯的方式。结果可能很大，你需要对结果模1000000007。

示例1:

 输入：n = 3

 输出：4

 说明: 有四种走法

示例2:

 输入：n = 5

 输出：13

提示:

    n范围在[1, 1000000]之间

**参考答案：**

import numpy as np

class Solution:

    def mat\_pow(self, A, n):

        m = A.shape[0]

        B = np.eye(m, dtype=np.int64)

        while n > 0:

            if (n&1)!=0:

                B = np.mod(np.matmul(B, A), self.p).astype(np.int64)

            A = np.mod(np.matmul(A, A), self.p).astype(np.int64)

            n >>= 1

        return B;

    def waysToStep(self, n: int) -> int:

        self.p = int(1e9+7)

        f = [1, 2, 4]

        if n <= 3: return f[n-1]

        A = np.array([[0, 0, 1], [1, 0, 1], [0, 1, 1]], dtype=np.int64)

        B = self.mat\_pow(A, n-3)

        res = 0

        for i in range(3):

            res += f[i] \* B[i][2]

        return int(res%self.p)

# 54.leetcode 144:二叉树的前序遍历

**题目链接：**[https://leetcode-cn.com/problems/binary-tree-preorder-traversal/](https://leetcode-cn.com/problems/binary-tree-preorder-traversal/" \t "_blank)

**题目描述：**

给定一个二叉树，返回它的 前序 遍历。

 示例:

输入: [1,null,2,3]

   1

    \

     2

    /

   3

输出: [1,2,3]

**参考答案：**

class Solution:

    def preorderTraversal(self, root: TreeNode) -> List[int]:

        r, stack = [], root and [root] or []

        while stack:

            root = stack.pop()

            r.append(root.val)

            stack += root.right and [root.right] or []

            stack += root.left and [root.left] or []

        return r

# 55.面试题08.05.:递归乘法

**题目链接：**[https://leetcode-cn.com/problems/recursive-mulitply-lcci/](https://leetcode-cn.com/problems/recursive-mulitply-lcci/" \t "_blank)

**题目描述：**

递归乘法。 写一个递归函数，不使用 \* 运算符， 实现两个正整数的相乘。可以使用加号、减号、位移，但要吝啬一些。

示例1:

 输入：A = 1, B = 10

 输出：10

示例2:

 输入：A = 3, B = 4

 输出：12

提示:

    保证乘法范围不会溢出

**参考答案：**

class Solution:

    def multiply(self, A: int, B: int) -> int:

        # 加一个优化处理

        if A < B: A, B = B, A

        return 0 if B <= 0 else (A + self.multiply(A, B - 1))

# 56.leetcode-122:买卖股票的最佳时机III

**题目链接：**[https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii/](https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii/" \t "_blank)

**题目描述：**

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。

设计一个算法来计算你所能获取的最大利润。你可以尽可能地完成更多的交易（多次买卖一支股票）。

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: [7,1,5,3,6,4]

输出: 7

解释: 在第 2 天（股票价格 = 1）的时候买入，在第 3 天（股票价格 = 5）的时候卖出, 这笔交易所能获得利润 = 5-1 = 4 。

     随后，在第 4 天（股票价格 = 3）的时候买入，在第 5 天（股票价格 = 6）的时候卖出, 这笔交易所能获得利润 = 6-3 = 3 。

示例 2:

输入: [1,2,3,4,5]

输出: 4

解释: 在第 1 天（股票价格 = 1）的时候买入，在第 5 天 （股票价格 = 5）的时候卖出, 这笔交易所能获得利润 = 5-1 = 4 。

     注意你不能在第 1 天和第 2 天接连购买股票，之后再将它们卖出。

     因为这样属于同时参与了多笔交易，你必须在再次购买前出售掉之前的股票。

示例 3:

输入: [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 <= 3 \* 10 ^ 4

    0 <= prices[i] <= 10 ^ 4

**参考答案：**

class Solution:

    def maxProfit(self, prices: List[int]) -> int:

        profit = 0

        for i in range(1, len(prices)):

            tmp = prices[i] - prices[i - 1]

            if tmp > 0: profit += tmp

        return profit

# 57.leetcode-123:买卖股票的最佳时机III

**题目链接：**[https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-iii/](https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-iii/" \t "_blank)

**题目描述：**

给定一个数组，它的第 i 个元素是一支给定的股票在第 i 天的价格。

设计一个算法来计算你所能获取的最大利润。你最多可以完成 两笔 交易。

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: [3,3,5,0,0,3,1,4]

输出: 6

解释: 在第 4 天（股票价格 = 0）的时候买入，在第 6 天（股票价格 = 3）的时候卖出，这笔交易所能获得利润 = 3-0 = 3 。

     随后，在第 7 天（股票价格 = 1）的时候买入，在第 8 天 （股票价格 = 4）的时候卖出，这笔交易所能获得利润 = 4-1 = 3 。

示例 2:

输入: [1,2,3,4,5]

输出: 4

解释: 在第 1 天（股票价格 = 1）的时候买入，在第 5 天 （股票价格 = 5）的时候卖出, 这笔交易所能获得利润 = 5-1 = 4 。

     注意你不能在第 1 天和第 2 天接连购买股票，之后再将它们卖出。

     因为这样属于同时参与了多笔交易，你必须在再次购买前出售掉之前的股票。

示例 3:

输入: [7,6,4,3,1]

输出: 0

解释: 在这个情况下, 没有交易完成, 所以最大利润为 0。

**参考答案：**

    def maxProfit(self, prices: List[int]) -> int:

        buy\_1 = buy\_2 = float('inf') # 第一二次买之前的最低价

        pro\_1 = pro\_2 = 0

        for p in prices:

            buy\_1 = min(buy\_1, p)

            pro\_1 = max(pro\_1, p - buy\_1)

            buy\_2 = min(buy\_2, p - pro\_1) # p - pro\_1 是用第一次的钱抵消了一部分第二次买的钱

            pro\_2 = max(pro\_2, p - buy\_2)

        return pro\_2

# 58.leetcode-455:分发饼干

**题目链接：**[https://leetcode-cn.com/problems/assign-cookies/](https://leetcode-cn.com/problems/assign-cookies/" \t "_blank)

**题目描述：**

假设你是一位很棒的家长，想要给你的孩子们一些小饼干。但是，每个孩子最多只能给一块饼干。对每个孩子 i ，都有一个胃口值 gi ，这是能让孩子们满足胃口的饼干的最小尺寸；并且每块饼干 j ，都有一个尺寸 sj 。如果 sj >= gi ，我们可以将这个饼干 j 分配给孩子 i ，这个孩子会得到满足。你的目标是尽可能满足越多数量的孩子，并输出这个最大数值。

注意：

你可以假设胃口值为正。

一个小朋友最多只能拥有一块饼干。

示例 1:

输入: [1,2,3], [1,1]

输出: 1

解释:

你有三个孩子和两块小饼干，3个孩子的胃口值分别是：1,2,3。

虽然你有两块小饼干，由于他们的尺寸都是1，你只能让胃口值是1的孩子满足。

所以你应该输出1。

示例 2:

输入: [1,2], [1,2,3]

输出: 2

解释:

你有两个孩子和三块小饼干，2个孩子的胃口值分别是1,2。

你拥有的饼干数量和尺寸都足以让所有孩子满足。

所以你应该输出2.

**参考答案：**

class Solution(object):

    def findContentChildren(self, g, s):

        """

        :type g: List[int]

        :type s: List[int]

        :rtype: int

        """

        res = 0

        g.sort()

        s.sort()

        g\_length = len(g)

        s\_length = len(s)

        i = 0

        j = 0

        while i < g\_length and j < s\_length:

            if g[i] <= s[j]:

                # 可以满足胃口，把小饼干喂给小朋友

                res += 1

                i += 1

                j += 1

            else:

                # 不满足胃口，查看下一块小饼干

                j += 1

        return res

# 59.leetcode-134:加油站

**题目链接：**[https://leetcode-cn.com/problems/gas-station/](https://leetcode-cn.com/problems/gas-station/" \t "_blank)

**题目描述：**

在一条环路上有 N 个加油站，其中第 i 个加油站有汽油 gas[i] 升。

你有一辆油箱容量无限的的汽车，从第 i 个加油站开往第 i+1 个加油站需要消耗汽油 cost[i] 升。你从其中的一个加油站出发，开始时油箱为空。

如果你可以绕环路行驶一周，则返回出发时加油站的编号，否则返回 -1。

说明:

    如果题目有解，该答案即为唯一答案。

    输入数组均为非空数组，且长度相同。

    输入数组中的元素均为非负数。

示例 1:

输入:

gas  = [1,2,3,4,5]

cost = [3,4,5,1,2]

输出: 3

解释:

从 3 号加油站(索引为 3 处)出发，可获得 4 升汽油。此时油箱有 = 0 + 4 = 4 升汽油

开往 4 号加油站，此时油箱有 4 - 1 + 5 = 8 升汽油

开往 0 号加油站，此时油箱有 8 - 2 + 1 = 7 升汽油

开往 1 号加油站，此时油箱有 7 - 3 + 2 = 6 升汽油

开往 2 号加油站，此时油箱有 6 - 4 + 3 = 5 升汽油

开往 3 号加油站，你需要消耗 5 升汽油，正好足够你返回到 3 号加油站。

因此，3 可为起始索引。

示例 2:

输入:

gas  = [2,3,4]

cost = [3,4,3]

输出: -1

解释:

你不能从 0 号或 1 号加油站出发，因为没有足够的汽油可以让你行驶到下一个加油站。

我们从 2 号加油站出发，可以获得 4 升汽油。 此时油箱有 = 0 + 4 = 4 升汽油

开往 0 号加油站，此时油箱有 4 - 3 + 2 = 3 升汽油

开往 1 号加油站，此时油箱有 3 - 3 + 3 = 3 升汽油

你无法返回 2 号加油站，因为返程需要消耗 4 升汽油，但是你的油箱只有 3 升汽油。

因此，无论怎样，你都不可能绕环路行驶一周。

**参考答案：**

class Solution:

    def canCompleteCircuit(self, gas, cost):

        """

        :type gas: List[int]

        :type cost: List[int]

        :rtype: int

        """

        n = len(gas)

        total\_tank, curr\_tank = 0, 0

        starting\_station = 0

        for i in range(n):

            total\_tank += gas[i] - cost[i]

            curr\_tank += gas[i] - cost[i]

            # If one couldn't get here,

            if curr\_tank < 0:

                # Pick up the next station as the starting one.

                starting\_station = i + 1

                # Start with an empty tank.

                curr\_tank = 0

        return starting\_station if total\_tank >= 0 else -1

# 60.leetcode-763:划分字母区间

**题目链接：**[https://leetcode-cn.com/problems/partition-labels/](https://leetcode-cn.com/problems/partition-labels/" \t "_blank)

**题目描述：**

字符串 S 由小写字母组成。我们要把这个字符串划分为尽可能多的片段，同一个字母只会出现在其中的一个片段。返回一个表示每个字符串片段的长度的列表。

示例 1：

输入：S = "ababcbacadefegdehijhklij"

输出：[9,7,8]

解释：

划分结果为 "ababcbaca", "defegde", "hijhklij"。

每个字母最多出现在一个片段中。

像 "ababcbacadefegde", "hijhklij" 的划分是错误的，因为划分的片段数较少。

提示：

    S的长度在[1, 500]之间。

    S只包含小写字母 'a' 到 'z' 。

**参考答案：**

class Solution(object):

    def partitionLabels(self, S):

        last = {c: i for i, c in enumerate(S)}

        j = anchor = 0

        ans = []

        for i, c in enumerate(S):

            j = max(j, last[c])

            if i == j:

                ans.append(i - anchor + 1)

                anchor = i + 1

        return ans

# 61.leetcode-435:无重叠区间

**题目链接：**[https://leetcode-cn.com/problems/non-overlapping-intervals/](https://leetcode-cn.com/problems/non-overlapping-intervals/" \t "_blank)

**题目描述：**

给定一个区间的集合，找到需要移除区间的最小数量，使剩余区间互不重叠。

注意:

    可以认为区间的终点总是大于它的起点。

    区间 [1,2] 和 [2,3] 的边界相互“接触”，但没有相互重叠。

示例 1:

输入: [ [1,2], [2,3], [3,4], [1,3]

输出: 1

解释: 移除 [1,3] 后，剩下的区间没有重叠。

示例 2:

输入: [ [1,2], [1,2], [1,2] ]

输出: 2

解释: 你需要移除两个 [1,2] 来使剩下的区间没有重叠。

示例 3:

输入: [ [1,2], [2,3] ]

输出: 0

解释: 你不需要移除任何区间，因为它们已经是无重叠的了。

**参考答案：**

class Solution:

    def eraseOverlapIntervals(self, intervals: List[List[int]]) -> int:

        if intervals == []: return 0

        ## 根据区间右端点排序

        intervals = sorted(intervals, key = lambda x: x[1])

        num = 0

        new\_interval = intervals[0]     # 被选的最新区间

        for interval in intervals[1:]:

            if interval[0] < new\_interval[1]:   # 如果重叠删除区间，删除数量+1

                num += 1

            else:   # 如果不重叠更新new\_interval

                new\_interval = interval

        return num

# 62.leetcode-784:字母大小写全排列

**题目链接：**[https://leetcode-cn.com/problems/letter-case-permutation/](https://leetcode-cn.com/problems/letter-case-permutation/" \t "_blank)

**题目描述：**

给定一个字符串S，通过将字符串S中的每个字母转变大小写，我们可以获得一个新的字符串。返回所有可能得到的字符串集合。

示例:

输入: S = "a1b2"

输出: ["a1b2", "a1B2", "A1b2", "A1B2"]

输入: S = "3z4"

输出: ["3z4", "3Z4"]

输入: S = "12345"

输出: ["12345"]

注意：

    S 的长度不超过12。

    S 仅由数字和字母组成。

**参考答案：**

class Solution(object):

    def letterCasePermutation(self, S):

        ans = [[]]

        for char in S:

            n = len(ans)

            if char.isalpha():

                for i in xrange(n):

                    ans.append(ans[i][:])

                    ans[i].append(char.lower())

                    ans[n+i].append(char.upper())

            else:

                for i in xrange(n):

                    ans[i].append(char)

        return map("".join, ans)

# 63.leetcode-17:电话号码的字母组合

**题目链接：**[https://leetcode-cn.com/problems/letter-combinations-of-a-phone-number/](https://leetcode-cn.com/problems/letter-combinations-of-a-phone-number/" \t "_blank)

**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。

**参考答案：**

class Solution:

    def letterCombinations(self, digits):

        """

        :type digits: str

        :rtype: List[str]

        """

        phone = {'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']}

        def backtrack(combination, next\_digits):

            # if there is no more digits to check

            if len(next\_digits) == 0:

                # the combination is done

                output.append(combination)

            # if there are still digits to check

            else:

                # iterate over all letters which map

                # the next available digit

                for letter in phone[next\_digits[0]]:

                    # append the current letter to the combination

                    # and proceed to the next digits

                    backtrack(combination + letter, next\_digits[1:])

        output = []

        if digits:

            backtrack("", digits)

        return output

# 64.面试题08.07.:无重复字符串的排列组合

**题目链接：**[https://leetcode-cn.com/problems/permutation-i-lcci/](https://leetcode-cn.com/problems/permutation-i-lcci/" \t "_blank)

**题目描述：**

无重复字符串的排列组合。编写一种方法，计算某字符串的所有排列组合，字符串每个字符均不相同。

示例1:

 输入：S = "qwe"

 输出：["qwe", "qew", "wqe", "weq", "ewq", "eqw"]

示例2:

 输入：S = "ab"

 输出：["ab", "ba"]

提示:

    字符都是英文字母。

    字符串长度在[1, 9]之间。

**参考答案：**

class Solution:

    def permutation(self, S: str) -> List[str]:

        if S == '':return []

        res = []

        path = ''

        def backtrack(S, path, res):

            if S == '':

                res.append(path)

                return

            for i in range(len(S)):

                cur = S[i]

                backtrack(S[:i] + S[i+1:], path + cur, res)

        backtrack(S, path, res)

        return res

# 65.面试题08.09:括号

**题目链接：**[https://leetcode-cn.com/problems/bracket-lcci/](https://leetcode-cn.com/problems/bracket-lcci/" \t "_blank)

**题目描述：**

括号。设计一种算法，打印n对括号的所有合法的（例如，开闭一一对应）组合。

说明：解集不能包含重复的子集。

例如，给出 n = 3，生成结果为：

[

  "((()))",

  "(()())",

  "(())()",

  "()(())",

  "()()()"

]

**参考答案：**

class Solution:

    def generateParenthesis(self, n: int) -> List[str]:

        if n<0: return []

        if n==0: return ['']

        res=set()

        for s in self.generateParenthesis(n-1):

            for i in range(2\*n-1):

                if s[:i]+'()'+s[i:] in res:continue

                res.add(s[:i]+'()'+s[i:])

        return list(res)

# 66.leetcode-53:最大子序和

**题目链接：**[https://leetcode-cn.com/problems/maximum-subarray/](https://leetcode-cn.com/problems/maximum-subarray/" \t "_blank)

**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。

**参考答案：**

class Solution:

    def maxSubArray(self, nums: List[int]) -> int:

        for i in range(1,len(nums)):

            nums[i] = max(nums[i-1]+nums[i],nums[i])

        return max(nums)

# 67.leetcode-120:三角形最小路径和

**题目链接：**[https://leetcode-cn.com/problems/triangle/](https://leetcode-cn.com/problems/triangle/" \t "_blank)

**题目描述：**

给定一个三角形，找出自顶向下的最小路径和。每一步只能移动到下一行中相邻的结点上。

相邻的结点 在这里指的是 下标 与 上一层结点下标 相同或者等于 上一层结点下标 + 1 的两个结点。

例如，给定三角形：

[

     [2],

    [3,4],

   [6,5,7],

  [4,1,8,3]

]

自顶向下的最小路径和为 11（即，2 + 3 + 5 + 1 = 11）。

**参考答案：**

class Solution:

    def minimumTotal(self, triangle: List[List[int]]) -> int:

        n = len(triangle)

        f = [[0] \* n for \_ in range(n)]

        f[0][0] = triangle[0][0]

        for i in range(1, n):

            f[i][0] = f[i - 1][0] + triangle[i][0]

            for j in range(1, i):

                f[i][j] = min(f[i - 1][j - 1], f[i - 1][j]) + triangle[i][j]

            f[i][i] = f[i - 1][i - 1] + triangle[i][i]

        return min(f[n - 1])

# 68.leetcode-221:最大正方形

**题目链接：**[https://leetcode-cn.com/problems/maximal-square/](https://leetcode-cn.com/problems/maximal-square/" \t "_blank)

**题目描述：**

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:

输入:

1 0 1 0 0

1 0 1 1 1

1 1 1 1 1

1 0 0 1 0

输出: 4

**参考答案：**

class Solution:

    def maximalSquare(self, matrix: List[List[str]]) -> int:

        if len(matrix) == 0 or len(matrix[0]) == 0:

            return 0

        maxSide = 0

        rows, columns = len(matrix), len(matrix[0])

        for i in range(rows):

            for j in range(columns):

                if matrix[i][j] == '1':

                    # 遇到一个 1 作为正方形的左上角

                    maxSide = max(maxSide, 1)

                    # 计算可能的最大正方形边长

                    currentMaxSide = min(rows - i, columns - j)

                    for k in range(1, currentMaxSide):

                        # 判断新增的一行一列是否均为 1

                        flag = True

                        if matrix[i + k][j + k] == '0':

                            break

                        for m in range(k):

                            if matrix[i + k][j + m] == '0' or matrix[i + m][j + k] == '0':

                                flag = False

                                break

                        if flag:

                            maxSide = max(maxSide, k + 1)

                        else:

                            break

        maxSquare = maxSide \* maxSide

        return maxSquare

# 69.剑指offer 47:礼物的最大价值

**题目链接：**[https://leetcode-cn.com/problems/li-wu-de-zui-da-jie-zhi-lcof/](https://leetcode-cn.com/problems/li-wu-de-zui-da-jie-zhi-lcof/" \t "_blank)

**题目描述：**

在一个 m\*n 的棋盘的每一格都放有一个礼物，每个礼物都有一定的价值（价值大于 0）。你可以从棋盘的左上角开始拿格子里的礼物，并每次向右或者向下移动一格、直到到达棋盘的右下角。给定一个棋盘及其上面的礼物的价值，请计算你最多能拿到多少价值的礼物？

示例 1:

输入:

[

  [1,3,1],

  [1,5,1],

  [4,2,1]

]

输出: 12

解释: 路径 1→3→5→2→1 可以拿到最多价值的礼物

**参考答案：**

class Solution:

    def maxValue(self, grid: List[List[int]]) -> int:

        m, n = len(grid), len(grid[0])

        for j in range(1, n): # 初始化第一行

            grid[0][j] += grid[0][j - 1]

        for i in range(1, m): # 初始化第一列

            grid[i][0] += grid[i - 1][0]

        for i in range(1, m):

            for j in range(1, n):

                grid[i][j] += max(grid[i][j - 1], grid[i - 1][j])

        return grid[-1][-1]

# 70.剑指offer12:矩阵中的路径

**题目链接：**[https://leetcode-cn.com/problems/ju-zhen-zhong-de-lu-jing-lcof/](https://leetcode-cn.com/problems/ju-zhen-zhong-de-lu-jing-lcof/" \t "_blank)

**题目描述：**

请设计一个函数，用来判断在一个矩阵中是否存在一条包含某字符串所有字符的路径。路径可以从矩阵中的任意一格开始，每一步可以在矩阵中向左、右、上、下移动一格。如果一条路径经过了矩阵的某一格，那么该路径不能再次进入该格子。例如，在下面的3×4的矩阵中包含一条字符串“bfce”的路径（路径中的字母用加粗标出）。

[["a","b","c","e"],

["s","f","c","s"],

["a","d","e","e"]]

但矩阵中不包含字符串“abfb”的路径，因为字符串的第一个字符b占据了矩阵中的第一行第二个格子之后，路径不能再次进入这个格子。

示例 1：

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

输出：true

示例 2：

输入：board = [["a","b"],["c","d"]], word = "abcd"

输出：false

提示：

    1 <= board.length <= 200

    1 <= board[i].length <= 200

**参考答案：**

class Solution:

    def exist(self, board: List[List[str]], word: str) -> bool:

        def dfs(i, j, k):

            if not 0 <= i < len(board) or not 0 <= j < len(board[0]) or board[i][j] != word[k]: return False

            if k == len(word) - 1: return True

            tmp, board[i][j] = board[i][j], '/'

            res = dfs(i + 1, j, k + 1) or dfs(i - 1, j, k + 1) or dfs(i, j + 1, k + 1) or dfs(i, j - 1, k + 1)

            board[i][j] = tmp

            return res

        for i in range(len(board)):

            for j in range(len(board[0])):

                if dfs(i, j, 0): return True

        return False

# 71.面试题17.15.:按摩师

**题目链接：**[https://leetcode-cn.com/problems/the-masseuse-lcci/](https://leetcode-cn.com/problems/the-masseuse-lcci/" \t "_blank)

**题目描述：**

一个有名的按摩师会收到源源不断的预约请求，每个预约都可以选择接或不接。在每次预约服务之间要有休息时间，因此她不能接受相邻的预约。给定一个预约请求序列，替按摩师找到最优的预约集合（总预约时间最长），返回总的分钟数。

注意：本题相对原题稍作改动

示例 1：

输入： [1,2,3,1]

输出： 4

解释： 选择 1 号预约和 3 号预约，总时长 = 1 + 3 = 4。

示例 2：

输入： [2,7,9,3,1]

输出： 12

解释： 选择 1 号预约、 3 号预约和 5 号预约，总时长 = 2 + 9 + 1 = 12。

示例 3：

输入： [2,1,4,5,3,1,1,3]

输出： 12

解释： 选择 1 号预约、 3 号预约、 5 号预约和 8 号预约，总时长 = 2 + 4 + 3 + 3 = 12。

**参考答案：**

class Solution:

    def massage(self, nums: List[int]) -> int:

        n = len(nums)

        if n == 0:

            return 0

        dp0, dp1 = 0, nums[0]

        for i in range(1, n):

            tdp0 = max(dp0, dp1)   # 计算 dp[i][0]

            tdp1 = dp0 + nums[i]   # 计算 dp[i][1]

            dp0, dp1 = tdp0, tdp1

        return max(dp0, dp1)

# 72.leetcode-647:回文子串

**题目链接：**[https://leetcode-cn.com/problems/palindromic-substrings/](https://leetcode-cn.com/problems/palindromic-substrings/" \t "_blank)

**题目描述：**

给定一个字符串，你的任务是计算这个字符串中有多少个回文子串。

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，也会被计为是不同的子串。

示例 1:

输入: "abc"

输出: 3

解释: 三个回文子串: "a", "b", "c".

示例 2:

输入: "aaa"

输出: 6

说明: 6个回文子串: "a", "a", "a", "aa", "aa", "aaa".

注意:

    输入的字符串长度不会超过1000。

**参考答案：**

class Solution(object):

    def countSubstrings(self, S):

        N = len(S)

        ans = 0

        for center in xrange(2\*N - 1):

            left = center / 2

            right = left + center % 2

            while left >= 0 and right < N and S[left] == S[right]:

                ans += 1

                left -= 1

                right += 1

        return ans

# 73.leetcode-1046:最后一块石头的重量

**题目链接：**[https://leetcode-cn.com/problems/last-stone-weight/](https://leetcode-cn.com/problems/last-stone-weight/" \t "_blank)

**题目描述：**

有一堆石头，每块石头的重量都是正整数。

每一回合，从中选出两块 最重的 石头，然后将它们一起粉碎。假设石头的重量分别为 x 和 y，且 x <= y。那么粉碎的可能结果如下：

    如果 x == y，那么两块石头都会被完全粉碎；

    如果 x != y，那么重量为 x 的石头将会完全粉碎，而重量为 y 的石头新重量为 y-x。

最后，最多只会剩下一块石头。返回此石头的重量。如果没有石头剩下，就返回 0。

示例：

输入：[2,7,4,1,8,1]

输出：1

解释：

先选出 7 和 8，得到 1，所以数组转换为 [2,4,1,1,1]，

再选出 2 和 4，得到 2，所以数组转换为 [2,1,1,1]，

接着是 2 和 1，得到 1，所以数组转换为 [1,1,1]，

最后选出 1 和 1，得到 0，最终数组转换为 [1]，这就是最后剩下那块石头的重量。

提示：

    1 <= stones.length <= 30

    1 <= stones[i] <= 1000

**参考答案：**

class Solution:

    def lastStoneWeight(self, stones: List[int]) -> int:

        while len(stones)>=2:

            stones.sort()

            stones.append(stones.pop()-stones.pop())

        return stones[0]

# 74.leetcode-910:最小差值II

**题目链接：**[https://leetcode-cn.com/problems/smallest-range-ii/](https://leetcode-cn.com/problems/smallest-range-ii/" \t "_blank)

**题目描述：**

给定一个整数数组 A，对于每个整数 A[i]，我们可以选择 x = -K 或是 x = K，并将 x 加到 A[i] 中。

在此过程之后，我们得到一些数组 B。

返回 B 的最大值和 B 的最小值之间可能存在的最小差值。

示例 1：

输入：A = [1], K = 0

输出：0

解释：B = [1]

示例 2：

输入：A = [0,10], K = 2

输出：6

解释：B = [2,8]

示例 3：

输入：A = [1,3,6], K = 3

输出：3

解释：B = [4,6,3]

提示：

    1 <= A.length <= 10000

    0 <= A[i] <= 10000

    0 <= K <= 10000

**参考答案：**

class Solution(object):

    def smallestRangeII(self, A, K):

        A.sort()

        mi, ma = A[0], A[-1]

        ans = ma - mi

        for i in xrange(len(A) - 1):

            a, b = A[i], A[i+1]

            ans = min(ans, max(ma-K, a+K) - min(mi+K, b-K))

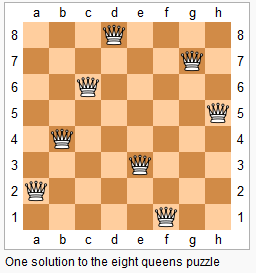
        return ans

# leetcode-51.N皇后

**题目链接：**[https://leetcode-cn.com/problems/n-queens/](https://leetcode-cn.com/problems/n-queens/" \t "_blank)

**题目描述：**

n 皇后问题研究的是如何将 n 个皇后放置在 n×n 的棋盘上，并且使皇后彼此之间不能相互攻击。



上图为 8 皇后问题的一种解法。

给定一个整数 n，返回所有不同的 n 皇后问题的解决方案。

每一种解法包含一个明确的 n 皇后问题的棋子放置方案，该方案中 'Q' 和 '.' 分别代表了皇后和空位。

示例:

输入: 4

输出: [

 [".Q..",  // 解法 1

  "...Q",

  "Q...",

  "..Q."],

 ["..Q.",  // 解法 2

  "Q...",

  "...Q",

  ".Q.."]

]

解释: 4 皇后问题存在两个不同的解法。

提示：

    皇后，是国际象棋中的棋子，意味着国王的妻子。皇后只做一件事，那就是“吃子”。当她遇见可以吃的棋子时，就迅速冲上去吃掉棋子。当然，她横、竖、斜都可走一到七步，可进可退。（引用自 百度百科 - 皇后 ）

**参考答案：**

class Solution:

    def solveNQueens(self, n: int) -> List[List[str]]:

        def could\_place(row, col):

            return not (cols[col] + hill\_diagonals[row - col] + dale\_diagonals[row + col])

        def place\_queen(row, col):

            queens.add((row, col))

            cols[col] = 1

            hill\_diagonals[row - col] = 1

            dale\_diagonals[row + col] = 1

        def remove\_queen(row, col):

            queens.remove((row, col))

            cols[col] = 0

            hill\_diagonals[row - col] = 0

            dale\_diagonals[row + col] = 0

        def add\_solution():

            solution = []

            for \_, col in sorted(queens):

                solution.append('.' \* col + 'Q' + '.' \* (n - col - 1))

            output.append(solution)

        def backtrack(row = 0):

            for col in range(n):

                if could\_place(row, col):

                    place\_queen(row, col)

                    if row + 1 == n:

                        add\_solution()

                    else:

                        backtrack(row + 1)

                    remove\_queen(row, col)

        cols = [0] \* n

        hill\_diagonals = [0] \* (2 \* n - 1)

        dale\_diagonals = [0] \* (2 \* n - 1)

        queens = set()

        output = []

        backtrack()

        return output

# 76.leetcode-90:子集II

**题目链接：**[https://leetcode-cn.com/problems/subsets-ii/](https://leetcode-cn.com/problems/subsets-ii/" \t "_blank)

**题目描述：**

给定一个可能包含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。

说明：解集不能包含重复的子集。

示例:

输入: [1,2,2]

输出:

[

  [2],

  [1],

  [1,2,2],

  [2,2],

  [1,2],

  []

]

**参考答案：**

# 刚开始我们只有空集一个答案，循环所有可能的数字，每次循环我们对当前答案的每一种情况考虑加入从1到上限次该数字并更新答案即可

    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:

        dic = {}

        for i in nums:

            dic[i] = dic.get(i, 0) + 1

        res = [[]]

        for i, v in dic.items():

            temp = res.copy()

            for j in res:

                temp.extend(j+[i]\*(k+1) for k in range(v))

            res = temp

        return res

# 77.面试题08.06.:汉诺塔问题

**题目链接：**[https://leetcode-cn.com/problems/hanota-lcci/](https://leetcode-cn.com/problems/hanota-lcci/" \t "_blank)

**题目描述：**

在经典汉诺塔问题中，有 3 根柱子及 N 个不同大小的穿孔圆盘，盘子可以滑入任意一根柱子。一开始，所有盘子自上而下按升序依次套在第一根柱子上(即每一个盘子只能放在更大的盘子上面)。移动圆盘时受到以下限制:

(1) 每次只能移动一个盘子;

(2) 盘子只能从柱子顶端滑出移到下一根柱子;

(3) 盘子只能叠在比它大的盘子上。

请编写程序，用栈将所有盘子从第一根柱子移到最后一根柱子。

你需要原地修改栈。

示例1:

 输入：A = [2, 1, 0], B = [], C = []

 输出：C = [2, 1, 0]

示例2:

 输入：A = [1, 0], B = [], C = []

 输出：C = [1, 0]

提示:

    A中盘子的数目不大于14个。

**参考答案：**

class Solution:

    def hanota(self, A: List[int], B: List[int], C: List[int]) -> None:

        n = len(A)

        self.move(n, A, B, C)

    # 定义move 函数移动汉诺塔

    def move(self,n, A, B, C):

        if n == 1:

            C.append(A[-1])

            A.pop()

            return

        else:

            self.move(n-1, A, C, B)  # 将A上面n-1个通过C移到B

            C.append(A[-1])          # 将A最后一个移到C

            A.pop()                  # 这时，A空了

            self.move(n-1,B, A, C)   # 将B上面n-1个通过空的A移到C

# 78.剑指offer 16:数值的整数次方

**题目链接：**[https://leetcode-cn.com/problems/shu-zhi-de-zheng-shu-ci-fang-lcof/](https://leetcode-cn.com/problems/shu-zhi-de-zheng-shu-ci-fang-lcof/" \t "_blank)

**题目描述：**

实现函数double Power(double base, int exponent)，求base的exponent次方。不得使用库函数，同时不需要考虑大数问题。

示例 1:

输入: 2.00000, 10

输出: 1024.00000

示例 2

输入: 2.10000, 3

输出: 9.26100

示例 3:

输入: 2.00000, -2

输出: 0.25000

解释: 2-2 = 1/22 = 1/4 = 0.25

说明:

    -100.0 < x < 100.0

    n 是 32 位有符号整数，其数值范围是 [−231, 231 − 1] 。

**参考答案：**

class Solution:

    def myPow(self, x: float, n: int) -> float:

        if x == 0: return 0

        res = 1

        if n < 0: x, n = 1 / x, -n

        while n:

            if n & 1: res \*= x

            x \*= x

            n >>= 1

        return res

# 79.leetcode-93:复原IP地址

**题目链接：**[https://leetcode-cn.com/problems/restore-ip-addresses/](https://leetcode-cn.com/problems/restore-ip-addresses/" \t "_blank)

**题目描述：**

给定一个只包含数字的字符串，复原它并返回所有可能的 IP 地址格式。

有效的 IP 地址正好由四个整数（每个整数位于 0 到 255 之间组成），整数之间用 '.' 分隔。

示例:

输入: "25525511135"

输出: ["255.255.11.135", "255.255.111.35"]

**参考答案：**

class Solution:

    def restoreIpAddresses(self, s):

        """

        :type s: str

        :rtype: List[str]

        """

        def valid(segment):

            """

            Check if the current segment is valid :

            1. less or equal to 255

            2. the first character could be '0'

               only if the segment is equal to '0'

            """

            return int(segment) <= 255 if segment[0] != '0' else len(segment) == 1

        def update\_output(curr\_pos):

            """

            Append the current list of segments

            to the list of solutions

            """

            segment = s[curr\_pos + 1:n]

            if valid(segment):

                segments.append(segment)

                output.append('.'.join(segments))

                segments.pop()

        def backtrack(prev\_pos = -1, dots = 3):

            """

            prev\_pos : the position of the previously placed dot

            dots : number of dots to place

            """

            # The current dot curr\_pos could be placed

            # in a range from prev\_pos + 1 to prev\_pos + 4.

            # The dot couldn't be placed

            # after the last character in the string.

            for curr\_pos in range(prev\_pos + 1, min(n - 1, prev\_pos + 4)):

                segment = s[prev\_pos + 1:curr\_pos + 1]

                if valid(segment):

                    segments.append(segment)  # place dot

                    if dots - 1 == 0:  # if all 3 dots are placed

                        update\_output(curr\_pos)  # add the solution to output

                    else:

                        backtrack(curr\_pos, dots - 1)  # continue to place dots

                    segments.pop()  # remove the last placed dot

        n = len(s)

        output, segments = [], []

        backtrack()

# 80.leetcode-17：电话号码的字母组合

**题目链接：**[https://leetcode-cn.com/problems/letter-combinations-of-a-phone-number/](https://leetcode-cn.com/problems/letter-combinations-of-a-phone-number/" \t "_blank)

**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。

**参考答案：**

class Solution:

    def letterCombinations(self, digits):

        """

        :type digits: str

        :rtype: List[str]

        """

        phone = {'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']}

        def backtrack(combination, next\_digits):

            # if there is no more digits to check

            if len(next\_digits) == 0:

                # the combination is done

                output.append(combination)

            # if there are still digits to check

            else:

                # iterate over all letters which map

                # the next available digit

                for letter in phone[next\_digits[0]]:

                    # append the current letter to the combination

                    # and proceed to the next digits

                    backtrack(combination + letter, next\_digits[1:])

        output = []

        if digits:

            backtrack("", digits)

        return output

# 81.面试题 17.06.：2出现的次数

**题目链接：**[https://leetcode-cn.com/problems/number-of-2s-in-range-lcci/](https://leetcode-cn.com/problems/number-of-2s-in-range-lcci/" \t "_blank)

**题目描述：**

编写一个方法，计算从 0 到 n (含 n) 中数字 2 出现的次数。

示例:

输入: 25

输出: 9

解释: (2, 12, 20, 21, 22, 23, 24, 25)(注意 22 应该算作两次)

提示：

    n <= 10^9

**参考答案：**

class Solution:

    def numberOf2sInRange(self, n: int) -> int:

        s= str(n)

        x= 2

        count = 0

        for i in range(len(s)):

            current = int(s[i])

            high = 0 if s[:i]=='' else int(s[:i])

            low =0 if s[i+1:]=='' else int(s[i+1:])

            if current>x:

                count+=(high+1)\*(10\*\*len(s[i+1:]))

            elif current<x:

                count += (high) \* (10 \*\* len(s[i + 1:]))

            else:

                count +=(high) \* (10 \*\* len(s[i + 1:]))+low+1

        return count

# 82.面试题 08.11.:硬币

**题目链接：**[https://leetcode-cn.com/problems/coin-lcci/](https://leetcode-cn.com/problems/coin-lcci/" \t "_blank)

**题目描述：**

硬币。给定数量不限的硬币，币值为25分、10分、5分和1分，编写代码计算n分有几种表示法。(结果可能会很大，你需要将结果模上1000000007)

示例1:

 输入: n = 5

 输出：2

 解释: 有两种方式可以凑成总金额:

5=5

5=1+1+1+1+1

示例2:

 输入: n = 10

 输出：4

 解释: 有四种方式可以凑成总金额:

10=10

10=5+5

10=5+1+1+1+1+1

10=1+1+1+1+1+1+1+1+1+1

说明：

注意:

你可以假设：

    0 <= n (总金额) <= 1000000

**参考答案：**

class Solution:

    def waysToChange(self, n: int) -> int:

        mod = 10\*\*9 + 7

        coins = [25, 10, 5, 1]

        f = [1] + [0] \* n

        for coin in coins:

            for i in range(coin, n + 1):

                f[i] += f[i - coin]

        return f[n] % mod

# 83.leetcode-2:两数相加

**题目链接：**[https://leetcode-cn.com/problems/add-two-numbers/](https://leetcode-cn.com/problems/add-two-numbers/" \t "_blank)

**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

**参考答案：**

# Definition for singly-linked list.

# class ListNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.next = None

class Solution:

    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:

        dummy = p = ListNode(None)

        s = 0

        while l1 or l2 or s:

            s += (l1.val if l1 else 0) + (l2.val if l2 else 0)

            p.next = ListNode(s % 10)

            p = p.next

            s //= 10

            l1 = l1.next if l1 else None

            l2 = l2.next if l2 else None

        return dummy.next

# 84.有效的数独

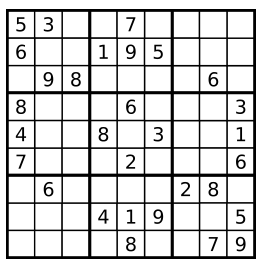
**题目链接：**[https://leetcode-cn.com/problems/valid-sudoku/](https://leetcode-cn.com/problems/valid-sudoku/" \t "_blank)

**题目描述：**判断一个 9x9 的数独是否有效。只需要根据以下规则，验证已经填入的数字是否有效即可。

数字 1-9 在每一行只能出现一次。

数字 1-9 在每一列只能出现一次。

数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。



上图是一个部分填充的有效的数独。

数独部分空格内已填入了数字，空白格用 '.' 表示。

示例 1:

输入:

[

  ["5","3",".",".","7",".",".",".","."],

  ["6",".",".","1","9","5",".",".","."],

  [".","9","8",".",".",".",".","6","."],

  ["8",".",".",".","6",".",".",".","3"],

  ["4",".",".","8",".","3",".",".","1"],

  ["7",".",".",".","2",".",".",".","6"],

  [".","6",".",".",".",".","2","8","."],

  [".",".",".","4","1","9",".",".","5"],

  [".",".",".",".","8",".",".","7","9"]

]

输出: true

示例 2:

输入:

[

  ["8","3",".",".","7",".",".",".","."],

  ["6",".",".","1","9","5",".",".","."],

  [".","9","8",".",".",".",".","6","."],

  ["8",".",".",".","6",".",".",".","3"],

  ["4",".",".","8",".","3",".",".","1"],

  ["7",".",".",".","2",".",".",".","6"],

  [".","6",".",".",".",".","2","8","."],

  [".",".",".","4","1","9",".",".","5"],

  [".",".",".",".","8",".",".","7","9"]

]

输出: false

解释: 除了第一行的第一个数字从 5 改为 8 以外，空格内其他数字均与 示例1 相同。

     但由于位于左上角的 3x3 宫内有两个 8 存在, 因此这个数独是无效的。

说明:

一个有效的数独（部分已被填充）不一定是可解的。

只需要根据以上规则，验证已经填入的数字是否有效即可。

给定数独序列只包含数字 1-9 和字符 '.' 。

给定数独永远是 9x9 形式的。

**参考答案：**

class Solution:

    def isValidSudoku(self, board: List[List[str]]) -> bool:

        hang\_dic=[[]for i in range(9)]

        lie\_dic=[[]for j in range(9)]#也可以构建一个9\*9的列表

        kuai\_dic=[[]for e in range(9)]

        for i in range (9):

            for j in range(9):

                num=board[i][j]

                kuai=i//3\*3+j//3

                if num!='.':

                    num=int(num)

                    if num not in hang\_dic[i]:

                        hang\_dic[i].append(num)

                    else:

                        return False

                    if num not in lie\_dic[j]:

                        lie\_dic[j].append(num)

                    else:

                        return False

                    if num not in kuai\_dic[kuai]:

                        kuai\_dic[kuai].append(num)

                    else:

                        return False

        return True

# 85.旋转图像

**题目链接：**[https://leetcode-cn.com/problems/rotate-image/](https://leetcode-cn.com/problems/rotate-image/" \t "_blank)

**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]

]

**参考答案：**

class Solution:

    def rotate(self, matrix: List[List[int]]) -> None:

        pos1,pos2 = 0,len(matrix)-1

        while   pos1<pos2:

            add = 0

            while   add < pos2-pos1:

                #左上角为0块，右上角为1块，右下角为2块，左下角为3块

                temp = matrix[pos2-add][pos1]

                matrix[pos2-add][pos1] = matrix[pos2][pos2-add]

                #3 = 2

                matrix[pos2][pos2-add] = matrix[pos1+add][pos2]

                #2 = 1

                matrix[pos1+add][pos2] = matrix[pos1][pos1+add]

                #1 = 0

                matrix[pos1][pos1+add] = temp

                #0 = temp

                add = add+1

            pos1 = pos1+1

            pos2 = pos2-1

# 86.搜索旋转排序数组

**题目链接：**[https://leetcode-cn.com/problems/search-in-rotated-sorted-array/](https://leetcode-cn.com/problems/search-in-rotated-sorted-array/" \t "_blank)

**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

**参考答案：**

class Solution:

    def search(self, nums: List[int], target: int) -> int:

        if nums==[]:

            return -1

        l=0

        r=len(nums)-1

        while l<r:

            mid=l+(r-l)//2

            if nums[mid]<nums[r]:

                if nums[mid]<target<=nums[r]:

                    l=mid+1

                else:

                    r=mid

            else:

                if nums[l]<=target<=nums[mid]:

                    r=mid

                else:

                    l=mid+1

        if nums[l]==target:

            return l

        else:

            return -1

# 87.数字范围按位与

**题目链接：**[https://leetcode-cn.com/problems/bitwise-and-of-numbers-range/](https://leetcode-cn.com/problems/bitwise-and-of-numbers-range/" \t "_blank)

**题目描述：**给定范围 [m, n]，其中 0 <= m <= n <= 2147483647，返回此范围内所有数字的按位与（包含 m, n 两端点）。

示例 1:

输入: [5,7]

输出: 4

示例 2:

输入: [0,1]

输出: 0

**参考答案：**

class Solution:

    def rangeBitwiseAnd(self, m: int, n: int) -> int:

        t = 0

        while m != n:

            m >>= 1

            n >>= 1

            t += 1

        return n << t

# 88.转置文件

**题目链接：**[https://leetcode-cn.com/problems/transpose-file/](https://leetcode-cn.com/problems/transpose-file/" \t "_blank)

**题目描述：**给定一个文件 file.txt，转置它的内容。

你可以假设每行列数相同，并且每个字段由 ' ' 分隔.

示例:

假设 file.txt 文件内容如下：

name age

alice 21

ryan 30

应当输出：

name alice ryan

age 21 30

**参考答案：**

data = []

i = 0

j = 0

with open('file.txt') as f:

    for line in f:

        i = len(line.split())

        j+=1

        data.append(line.split())

res = [[] for i in range(i)]

for a in range(j):

    for b in range(i):

        res[b].append(data[a][b])

st = ''

for item in res:

    st = st + ' '.join(item) + '\n'

print(st[:-1])

# 89.单词拆分

**题目链接：**[https://leetcode-cn.com/problems/word-break/](https://leetcode-cn.com/problems/word-break/" \t "_blank)

**题目描述：**给定一个非空字符串 s 和一个包含非空单词列表的字典 wordDict，判定 s 是否可以被空格拆分为一个或多个在字典中出现的单词。

说明：

拆分时可以重复使用字典中的单词。

你可以假设字典中没有重复的单词。

示例 1：

输入: s = "leetcode", wordDict = ["leet", "code"]

输出: true

解释: 返回 true 因为 "leetcode" 可以被拆分成 "leet code"。

示例 2：

输入: s = "applepenapple", wordDict = ["apple", "pen"]

输出: true

解释: 返回 true 因为 "applepenapple" 可以被拆分成 "apple pen apple"。

     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

**参考答案：**

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(n):

            for j in range(i+1,n+1):

                if(dp[i] and (s[i:j] in wordDict)):

                    dp[j]=True

        return dp[-1]

# 90.格雷编码

**题目链接：**[https://leetcode-cn.com/problems/gray-code/](https://leetcode-cn.com/problems/gray-code/" \t "_blank)

**题目描述：**格雷编码是一个二进制数字系统，在该系统中，两个连续的数值仅有一个位数的差异。

给定一个代表编码总位数的非负整数 n，打印其格雷编码序列。即使有多个不同答案，你也只需要返回其中一种。

格雷编码序列必须以 0 开头。

示例 1:

输入: 2

输出: [0,1,3,2]

解释:

00 - 0

01 - 1

11 - 3

10 - 2

对于给定的 n，其格雷编码序列并不唯一。

例如，[0,2,3,1] 也是一个有效的格雷编码序列。

00 - 0

10 - 2

11 - 3

01 - 1

示例 2:

输入: 0

输出: [0]

解释: 我们定义格雷编码序列必须以 0 开头。

     给定编码总位数为 n 的格雷编码序列，其长度为 2n。当 n = 0 时，长度为 20 = 1。

     因此，当 n = 0 时，其格雷编码序列为 [0]。

**参考答案：**

class Solution:

    def grayCode(self, n: int) -> List[int]:

        res, head = [0], 1

        for i in range(n):

            for j in range(len(res) - 1, -1, -1):

                res.append(head + res[j])

            head <<= 1

        return res

# 91.Pow(x, n)

**题目链接：**[https://leetcode-cn.com/problems/powx-n/](https://leetcode-cn.com/problems/powx-n/" \t "_blank)

**题目描述：**实现 pow(x, n) ，即计算 x 的 n 次幂函数。

示例 1:

输入: 2.00000, 10

输出: 1024.00000

示例 2:

输入: 2.10000, 3

输出: 9.26100

示例 3:

输入: 2.00000, -2

输出: 0.25000

解释: 2-2 = 1/22 = 1/4 = 0.25

说明:

-100.0 < x < 100.0

n 是 32 位有符号整数，其数值范围是 [−231, 231 − 1] 。

**参考答案：**

class Solution:

    def myPow(self, x: float, n: int) -> float:

        def quickMul(N):

            if N == 0:

                return 1.0

            y = quickMul(N // 2)

            return y \* y if N % 2 == 0 else y \* y \* x

        return quickMul(n) if n >= 0 else 1.0 / quickMul(-n)

# 92.网络延迟时间（最短路径）

**题目链接：**[https://leetcode-cn.com/problems/network-delay-time/](https://leetcode-cn.com/problems/network-delay-time/" \t "_blank)

**题目描述：**给定一个列表times，表示信号经过**有向**边的传递时间。

示例：

输入：times=[[2,1,1],[2,3,1],[3,4,1]],N=4,K=2

输出：2

**参考答案：**

class Solution:

    def networkDelayTime(self, times: List[List[int]], N: int, K: int) -> int:

        mp = {k: [] for k in range(1, N + 1)}

        for u, v, w in times:

            mp[u].append([v, w])

        arrived = [0 for \_ in range(N + 1)]

        distances = [sys.maxsize for \_ in range(N + 1)]

        arrived[K] = 0

        distances[K] = 0

        while True:

            min\_val = sys.maxsize

            idx = -1

            for i in range(1, N + 1):

                if distances[i] < min\_val and arrived[i] == 0:

                    min\_val = distances[i]

                    idx = i

            if idx == -1:

                break

            arrived[idx] = 1

            for v, w in mp[idx]:

                if arrived[v] == 0:

                    distances[v] = min(distances[idx] + w, distances[v])

        return max(distances[1:]) if all(i == 1 for i in arrived[1:]) else -1

# 93.相似度为K的字符串

**题目链接：**[https://leetcode-cn.com/problems/k-similar-strings/](https://leetcode-cn.com/problems/k-similar-strings/" \t "_blank)

**题目描述：**如果可以通过将 A 中的两个小写字母精确地交换位置 K 次得到与 B 相等的字符串，我们称字符串 A 和 B 的相似度为 K（K 为非负整数）。

给定两个字母异位词 A和 B ，返回 A 和 B 的相似度 K 的最小值。

示例1：

输入: A = "ab", B="ba"

输出：1

示例2：

输入：A = "abc", =“bca"

输出：2

**参考答案：**

class Solution:

    def kSimilarity(self, A: str, B: str) -> int:

        d = collections.defaultdict(list)       #生成图，用字典存邻接表，方便遍历

        c = collections.defaultdict(int)        #生成边，方便矛盾判断

        for i, j in zip(A, B):

            if i != j:

                d[i] += [j]

                c[(i, j)] += 1

        def dfs(i, dep, path):  #深搜函数，参数分别是：遍历到点，深度，当前路径

            if dep == depth:

                if i == start:  #如果到达深度且成环，就加入候选环生成队列

                    nonlocal dep\_loop

                    for i in range(depth + 1):

                        path = path[1: ] + path[: 1]

                        if tuple(path) in dep\_loop and start != path[0]: #不反复生成多余的环

                            return

                    dep\_loop += [tuple(path)]

            else:

                for j in d[i]:  #进一步深搜遍历

                    dfs(j, dep + 1, path + [i])

        def judgment(a, b):     #判断生成同样大小的环是否矛盾的函数

            tc = collections.defaultdict(int)

            for i in range(depth + 1):

                tc[(a[i - 1], a[i])] += 1

                tc[(b[i - 1], b[i])] += 1

            for i, j in tc:

                if tc[(i, j)] > c[(i, j)]:

                    return True

            return False

        ans = 0

        depth = 1

        while d:

            if depth == 5:      #最大深度剪枝

                ans += sum(c.values()) // 6 \* 5

                break

            dep\_loop = []       #同一深度生成候选环队列

            for i in d:

                start = i       #候选环深搜起点标记

                for j in d[i]:  #深搜生成环

                    dfs(j, 0, [i])

            n = len(dep\_loop)

            p = collections.defaultdict(int)

            for i in range(n):

                for j in range(i + 1, n):

                    if judgment(dep\_loop[i], dep\_loop[j]): #两两比较是否矛盾，矛盾多的环优先级靠后

                        p[dep\_loop[i]] += 1

                        p[dep\_loop[j]] += 1

            dep\_loop.sort(key = lambda x: p[x])     #按矛盾数对候选环进行升序排序

            for loop in dep\_loop:                   #在候选环队列里逐个枚举尝试生成环

                tc = collections.defaultdict(int)

                flag = False

                for i in range(depth + 1):          #检查图里面是否还有足够的边生成该候选环

                    tc[(loop[i - 1], loop[i])] += 1

                    if tc[(loop[i - 1], loop[i])] > c[(loop[i - 1], loop[i])]:

                        flag = True

                        break

                if flag:        #尝试生成环失败，继续试下一个候选环

                    continue

                ans += depth    #当前环生成成功，交换次数等于生成深度，加到总答案里面去

                for i in range(depth + 1):          #删去图中已生成环的有向边

                    d[loop[i - 1]].remove(loop[i])

                    if not d[loop[i - 1]]:

                        d.pop(loop[i - 1])

                    c[(loop[i - 1], loop[i])] -= 1

                if not d:       #如果图空了就跳出循环

                    break

            depth += 1

        return ans

# 94.翻转矩阵后的得分

**题目链接：**[h](https://leetcode-cn.com/problems/k-similar-strings/" \t "_blank)[ttps://leetcode-cn.com/problems/score-after-flipping-matrix/](https://leetcode-cn.com/problems/score-after-flipping-matrix/" \t "_blank)

**题目描述：**有一个二维矩阵 A 其中每个元素的值为 0 或 1 。

移动是指选择任一行或列，并转换该行或列中的每一个值：将所有 0 都更改为 1，将所有 1 都更改为 0。

在做出任意次数的移动后，将该矩阵的每一行都按照二进制数来解释，矩阵的得分就是这些数字的总和。

返回尽可能高的分数。

示例：

输入：[[0,0,1,1],[1,0,1,0][1,1,0,0]]

输出：39

**参考答案：**

class Solution:

    def matrixScore(self, A: List[List[int]]) -> int:

        R,C = len(A),len(A[0])

        #行变换，只要第一行不为1就转换

        for r in range(R):

            if A[r][0] == 0:

                A[r]=[1-i for i in A[r]]

        #列变换，只要每列0的个数大于一半

        for c in range(1,C):

            cnt = 0

            for r in range(R):

                if A[r][c] == 1:

                    cnt +=1

            if cnt < R/2:

                for r in range(R):

                    A[r][c] = 1-A[r][c]

        res = 0

        for r in range(R):

            row = A[r][::-1]

            res +=sum([row[c]\*2\*\*c for c in range(C)])

        return res

# 95.优势洗牌

**题目链接：**[https://leetcode-cn.com/problems/advantage-shuffle/](https://leetcode-cn.com/problems/advantage-shuffle/" \t "_blank)

**题目描述：**给定两个大小相等的数组 A 和 B，A 相对于 B 的优势可以用满足 A[i] > B[i] 的索引 i 的数目来描述。

返回 A 的任意排列，使其相对于 B 的优势最大化。

示例1：

输入：A = [2,7,11,15], B=[1,10,4,11]

输出：[2,11,7,15]

示例2：

输入：A = [12,24,8,32], B = [13,25,32,11]

输出：[24,32,8,12]

**参考答案：**

class Solution:

    def advantageCount(self, A: List[int], B: List[int]) -> List[int]:

        b = [(i, B[i]) for i in range(len(B))]#可以用字典来实现  只是排序需要自己实现

        b.sort(key = lambda x : x[1])

        min\_pos, max\_pos = 0, len(B)-1#取得B的最小值下标和最大值下标

        ans = [0 for \_ in range(len(A))]#结果

        A.sort()

        for val in A:#依次去取出最小值

            if val > b[min\_pos][1]:#如果刚好A的最小值>B的最小值，就直接贪心的做   这匹马可以带来收益

                ans[b[min\_pos][0]] = val#将这个最小值放在与B对应最小值的位置  即可获得收益

                min\_pos += 1#最小值位置后移

            else:#如果相等或者小于B的最小值，就将A中对应B最大值的位置改为当前值，    这匹马不能带来收益

                ans[b[max\_pos][0]] = val##将这个最小值放在与B对应最大值的位置  会获得损失

                max\_pos -= 1

        return ans

# 96.石子游戏

**题目链接：**[https://leetcode-cn.com/problems/stone-game/](https://leetcode-cn.com/problems/stone-game/" \t "_blank)

**题目描述：**亚历克斯和李用几堆石子在做游戏。偶数堆石子排成一行，每堆都有正整数颗石子 piles[i] 。

游戏以谁手中的石子最多来决出胜负。石子的总数是奇数，所以没有平局。

亚历克斯和李轮流进行，亚历克斯先开始。 每回合，玩家从行的开始或结束处取走整堆石头。 这种情况一直持续到没有更多的石子堆为止，此时手中石子最多的玩家获胜。

假设亚历克斯和李都发挥出最佳水平，当亚历克斯赢得比赛时返回 true ，当李赢得比赛时返回 false 。

示例：

输入:[5,3,4,5]

输出：true

**参考答案：**

 class Solution:

    def stoneGame(self, piles: List[int]) -> bool:

        #定义状态f(i)当前手中的石子的数量

        #状态转移：f(i) = f(i-1)+max(piles[0],piles[-1])

        #状态初始化i=1开始，f(0) = max(piles[0],piles[-1])

        #输出值：f[-1]

        if not piles:

            return True

        f = [0]\*(len(piles))

        if piles[0]>piles[-1]:

            f[0] = piles[0]

            del piles[0]

        else:

            f[0]=piles[-1]

            del piles[-1]

        for i in range(1,len(f)):

            if piles[0]>piles[-1]:index = 0

            else:index = -1

            if i%2==0:f[i] = f[i-2]+piles[index]

            else:f[i] = f[i-1]+piles[index]

            del piles[index]

        print(f)

        return True if f[-1]>f[-2] else False

# 97.一手顺子

**题目链接：**[https://leetcode-cn.com/problems/hand-of-straights/](https://leetcode-cn.com/problems/hand-of-straights/" \t "_blank)

**题目描述：**爱丽丝有一手（hand）由整数数组给定的牌。

现在她想把牌重新排列成组，使得每个组的大小都是 W，且由 W 张连续的牌组成。

如果她可以完成分组就返回 true，否则返回 false。

示例1：

输入：hand=[1,2,3,6,2,3,4,7,8], w=3

输出：true

示例2：

输入：hand = [1,2,3,4,5], w=4

输出：false

**参考答案：**

from collections import Counter

class Solution:

    def isNStraightHand(self, hand: List[int], W: int) -> bool:

        hand\_counter = Counter(hand)

        # 按 key 值排序结果

        sort\_hand = sorted(hand\_counter)

        for h in sort\_hand:

            # 牌不够取了

            if hand\_counter[h] < 0:

                return False

            elif hand\_counter[h] > 0:

                for i in range(1, W):

                    if hand\_counter[h + i] == 0:

                        return False

                    else:

                        hand\_counter[h + i] -= hand\_counter[h]

            else:

                # == 0 时跳过

                continue

        return True

# 98.情侣牵手

**题目链接：**[https://leetcode-cn.com/problems/couples-holding-hands/](https://leetcode-cn.com/problems/couples-holding-hands/" \t "_blank)

**题目描述：**N 对情侣坐在连续排列的 2N 个座位上，想要牵到对方的手。 计算最少交换座位的次数，以便每对情侣可以并肩坐在一起。 一次交换可选择任意两人，让他们站起来交换座位。

人和座位用 0 到 2N-1 的整数表示，情侣们按顺序编号，第一对是 (0, 1)，第二对是 (2, 3)，以此类推，最后一对是 (2N-2, 2N-1)。

这些情侣的初始座位  row[i] 是由最初始坐在第 i 个座位上的人决定的。

示例1：

输入：row = [0,2,1,3]

输出：1

示例2：

输入：row = [3,2,0,1]

输出：0

**参考答案：**

class Solution:

    def minSwapsCouples(self, row: List[int]) -> int:

        res = 0

        seatmap= [0 for \_ in range(len(row))]

        for j in range(len(row)):

            seatmap[row[j]] = j#序号为row[j]的人的座位号为j

        for i in range(0,len(row),2):

            x = row[i]^1

            if x == row[i+1]:

                continue

            else:

                index = seatmap[x]#找到序号为x的人对应的座位号

                row[i+1],row[index] = row[index],row[i+1]#交换座位使情侣坐一起

                #此时需要更新seatmap的信息

                seatmap[x] = i+1#序号为x的人现在在座位i+1上

                seatmap[row[index]] = index#序号为row[index]的人现在在做为index上

                res += 1

        return res

# 99.森林中的兔子

**题目链接：**[https://leetcode-cn.com/problems/rabbits-in-forest/](https://leetcode-cn.com/problems/rabbits-in-forest/" \t "_blank)

**题目描述：**森林中，每个兔子都有颜色。其中一些兔子（可能是全部）告诉你还有多少其他的兔子和自己有相同的颜色。我们将这些回答放在 answers 数组里。

返回森林中兔子的最少数量。

示例1：

输入：answers = [1,1,2]

输出：5

示例2：

输入：answers = [10,10,10]

输出：10

**参考答案：**

class Solution:

    def numRabbits(self, answers: List[int]) -> int:

        if len(answers)==0:

            return 0

        hashmap ={}

        count = 0

        for i in range(len(answers)):

            if answers[i] not in hashmap:

                hashmap[answers[i]]=1

            else:

                hashmap[answers[i]]+=1

        print(hashmap)

        for i in hashmap.keys():

            count += (hashmap[i]//(i+1))\*(i+1)

            if hashmap[i]%(i+1)!=0:

                count += i+1

        return count

# 100.多米诺和托米诺平铺

**题目链接：**[https://leetcode-cn.com/problems/domino-and-tromino-tiling/](https://leetcode-cn.com/problems/domino-and-tromino-tiling/" \t "_blank)

**题目描述：**有两种形状的瓷砖：一种是 2x1 的多米诺形，另一种是形如 "L" 的托米诺形。两种形状都可以旋转。

示例：

输入：3

输出：5

**参考答案：**

class Solution:

    def numTilings(self, N: int) -> int:

        dp = [0] \* (2 \* 1000 + 1)#后面的这几行是初始的前几个方法数量

        dp[2] = 1

        dp[3] = 2

        dp[4] = 2

        dp[5] = 4

        for i in range(6,2 \* N + 1):

            dp[i] = dp[i - 2] + dp[i - 3] + (dp[i - 4] if i % 2 == 0 else dp[i - 3])#这一行其实没这么简

#单，虽然写出来只有一行，但是得分开理解，解析如下：

'''

   奇数：dp[i] = dp[i - 2] + dp[i - 3] + dp[i - 4]，简单说来就是i - 2就一种，你最后两格只能竖着放多米诺

，i - 3你也没得选，只能放一个托米诺，i - 4其实有两种，要么两个横着多米诺，要么两个竖着多米诺，但其实两个

竖着的多米诺，是和i - 2中的一种情况重复，故只能加上一个，所以答案为此

'''

'''

   偶数：dp[i] = dp[i - 2] + 2 \* dp[i - 3]，i - 2没得选，只能放多米诺，i - 3的话你可以想象，只有放托米诺

，多出来的那一格在上，要么在下，故两种

'''

        return dp[2 \* N] % (10 \*\* 9 + 7)

# 101.最大黑方阵

**题目链接：**[https://leetcode-cn.com/problems/max-black-square-lcci/](https://leetcode-cn.com/problems/max-black-square-lcci/" \t "_blank)

**题目描述：**给定一个方阵，其中每个单元(像素)非黑即白。设计一个算法，找出 4 条边皆为黑色像素的最大子方阵。

返回一个数组 [r, c, size] ，其中 r, c 分别代表子方阵左上角的行号和列号，size 是子方阵的边长。若有多个满足条件的子方阵，返回 r 最小的，若 r 相同，返回 c 最小的子方阵。若无满足条件的子方阵，返回空数组。

示例1：

输入：[[1,0,1],[0,0,1],[0,0,1]]

输出：[1,0,2]

示例2：

输入：[[0,1,1],[1,0,1],[1,1,0]]

输出：[0,0,1]

**参考答案：**

class Solution:

    def findSquare(self, matrix: List[List[int]]) -> List[int]:

        if not matrix:

            return []

        rows,cols=len(matrix),len(matrix[0])

        # dp[i][j]=[a,b] a is the right line, b is the bottom line

        # dp[i][j]表示以i,j为右下角的方阵，上右方a和左下方b的连续0长度

        dp=[[[0,0] for \_ in range(cols+1)] for \_ in range(rows+1)] #边界处理

        # dp数组初始化

        for i in range(1,rows+1):

            for j in range(1,cols+1):

                if matrix[i-1][j-1]==0:

                    dp[i][j][0]=dp[i-1][j][0]+1

                    dp[i][j][1]=dp[i][j-1][1]+1

        r,c,size=-1,-1,0

        for i in range(1,rows+1):

            for j in range(1,cols+1):

                if matrix[i-1][j-1]==0: # 其实个人认为这行无所谓要不要，看别的题解都有这个判断，无非就是优化了一点时间

                    t\_size=min(dp[i][j]) # 因为是方阵 所以取最短边

                    while t\_size>size:

                        #判断两边长度能否满足这个最短边的长度

                        if dp[i-t\_size+1][j][1]>=t\_size and dp[i][j-t\_size+1][0]>=t\_size:

                            r,c=i-t\_size,j-t\_size

                            size=t\_size

                            break

                        t\_size-=1

        if size==0:

            return []

        return [r,c,size]

# 102.最大层内元素和

**题目链接：**[https://leetcode-cn.com/problems/maximum-level-sum-of-a-binary-tree/](https://leetcode-cn.com/problems/maximum-level-sum-of-a-binary-tree/" \t "_blank)

**题目描述：**给你一个二叉树的根节点 root。设根节点位于二叉树的第 1 层，而根节点的子节点位于第 2 层，依此类推。

请你找出层内元素之和 最大 的那几层（可能只有一层）的层号，并返回其中 最小 的那个

示例：

输入：[1,7,0,7,-8,null,null]

输出：2

**参考答案：**

class Solution:

    def maxLevelSum(self, root: TreeNode) -> int:

        # 存储每层结点

        stack = [root]

        # 记录层数

        depth = 0

        # 记录最大和层数

        res = 1

        # 记录最大和

        h = 0

        while stack:

            depth += 1

            # 叠加记录当前层数前缀和

            s = 0

            l = len(stack)

            i = 0

            while i < l:

                pre = stack.pop(0)

                s += pre.val

                if pre.left:

                    stack.append(pre.left)

                if pre.right:

                    stack.append(pre.right)

                i += 1

            if s > h:

                h = s

                res = depth

        return res

# 103.掷骰子模拟

**题目链接：**[https://leetcode-cn.com/problems/dice-roll-simulation/](https://leetcode-cn.com/problems/dice-roll-simulation/" \t "_blank)

**题目描述：**有一个骰子模拟器会每次投掷的时候生成一个 1 到 6 的随机数。

不过我们在使用它时有个约束，就是使得投掷骰子时，连续 掷出数字 i 的次数不能超过 rollMax[i]（i 从 1 开始编号）。

现在，给你一个整数数组 rollMax 和一个整数 n，请你来计算掷 n 次骰子可得到的不同点数序列的数量。

假如两个序列中至少存在一个元素不同，就认为这两个序列是不同的。由于答案可能很大，所以请返回 模 10^9 + 7 之后的结果

示例1：

输入：n = 2, rollMax = [1,1,2,2,2,3]

输出：34

示例2：

输入：n = 2, rollMax = [1,1,1,1,1,1]

输出：30

**参考答案：**

class Solution:

    def dieSimulator(self, n: int, rollMax: List[int]) -> int:

        dp = [[[0 for \_ in range(16)] for \_ in range(7)] for \_ in range(n + 1)]

        mod = 10\*\*9 + 7

        for i in range(1, n + 1):

            # 投掷的数

            for j in range(1, 7):

                # 第一次投掷

                if i == 1:

                    dp[i][j][1] = 1

                    continue

                # 数字 j 连续出现 k 次

                for k in range(2, rollMax[j - 1] + 1):

                    dp[i][j][k] = dp[i - 1][j][k - 1]

                # 前一次投出的数不是 j

                s = 0

                for l in range(1, 7):

                    if l == j:

                        continue

                    for k in range(1, 16):

                        s += dp[i - 1][l][k]

                        s %= mod

                dp[i][j][1] = s

        res = 0

        for j in range(1, 7):

            for k in range(1, 16):

                # 求投掷 n 次时所有组合总和

                res += dp[n][j][k]

                res %= mod

        return res

# 104最大相等频率

**题目链接：**[https://leetcode-cn.com/problems/maximum-equal-frequency/](https://leetcode-cn.com/problems/maximum-equal-frequency/" \t "_blank)

**题目描述：**给出一个正整数数组 nums，请你帮忙从该数组中找出能满足下面要求的 最长 前缀，并返回其长度：

从前缀中 删除一个 元素后，使得所剩下的每个数字的出现次数相同。

如果删除这个元素后没有剩余元素存在，仍可认为每个数字都具有相同的出现次数（也就是 0 次）。

示例1：

输入：nums = [2,2,1,1,5,3,3,5]

输出：7

示例2：

输入：nums = [1,1,1,2,2,2,3,3,3,4,4,4,5]

输出：13

**参考答案：**

class Solution:

    def maxEqualFreq(self, nums: List[int]) -> int:

        times=collections.Counter(nums)     # 元素出现次数

        freq=collections.Counter(times.values())    # 该次数下的元素数目

        for i in range(len(nums),0,-1):

            # print(times,freq)

            if len(times)<2:return i    # 只有一种元素，符合，返回长度。

            if len(freq)<2 and 1 in freq.keys():return i    # 所有元素都只出现1次，返回长度。

            if len(freq)==2 and freq.get(1,0)==1:return i   # 某个元素出现1次，其它元素出现次数相同。

            # 其它所有元素出现次数相同，只有某个元素恰好多一次。符合，返回长度。

            if len(freq)==2 and max(freq.keys())-min(freq.keys())==1 and freq[max(freq.keys())]==1:return i

            # 更新字典。

            freq[times[nums[i-1]]]-=1   # 末位元素出现次数下的元素数目减1

            if freq[times[nums[i-1]]]==0:

                del freq[times[nums[i-1]]]  # 为零值删除条目

            times[nums[i-1]]-=1         # 末位元素出现次数减1

            if times[nums[i-1]]==0:

                del times[nums[i-1]]    # 为零值删除条目

            if times[nums[i-1]]>0:      # 新的次数下的元素数目加1

                freq[times[nums[i-1]]]=freq.get(times[nums[i-1]],0)+1

# 105.除以自身以外数组的乘积

**题目链接：**[https://leetcode-cn.com/problems/product-of-array-except-self/](https://leetcode-cn.com/problems/product-of-array-except-self/" \t "_blank)

**题目描述：**给你一个长度为 n 的整数数组 nums，其中 n > 1，返回输出数组 output ，其中 output[i] 等于 nums 中除 nums[i] 之外其余各元素的乘积。

示例：

输入：[1,2,3,4]

输出：[24,12,8,6]

**参考答案：**

class Solution:

    def productExceptSelf(self, nums: List[int]) -> List[int]:

        n = len(nums); ans = [0]\*n

        left, right = [1]\*n, [1]\*n

        for i in range(1, n):

            left[i] = left[i-1]\*nums[i-1]

        for i in range(n-2, -1, -1):

            right[i] = right[i+1]\*nums[i+1]

        for i in range(n):

            ans[i] = left[i]\*right[i]

        return ans

# 106.寻找两个正序数组的中位数

**题目链接：**[https://leetcode-cn.com/problems/median-of-two-sorted-arrays/](https://leetcode-cn.com/problems/median-of-two-sorted-arrays/" \t "_blank)

**题目描述：**给定两个大小为 m 和 n 的正序（从小到大）数组 nums1 和 nums2。

请你找出这两个正序数组的中位数，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

**参考答案：**

class Solution(object):

    def findMedianSortedArrays(self, nums1, nums2):

        """

        :type nums1: List[int]

        :type nums2: List[int]

        :rtype: float

        """

        numsum=nums1+nums2 #将2个序列拼接

        numsum.sort() #排序

        if len(numsum)%2==0:

            themidnum  = float(numsum[len(numsum)//2-1]+numsum[len(numsum)//2])/2

        else:

            themidnum  = float(numsum[len(numsum)//2])

        return themidnum

# 107.盛最多水的容器

**题目链接：**[https://leetcode-cn.com/problems/container-with-most-water/](https://leetcode-cn.com/problems/container-with-most-water/" \t "_blank)

**题目描述：**给你 n 个非负整数 a1，a2，...，an，每个数代表坐标中的一个点 (i, ai) 。在坐标内画 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。

示例：

输入：[1,8,6,2,5,4,8,3,7]

输出：49

**参考答案：**

class Solution:

    def maxArea(self, height: List[int]) -> int:

        i, j, res = 0, len(height) - 1, 0

        while i < j:

            if height[i] < height[j]:

                res = max(res, height[i] \* (j - i))

                i += 1

            else:

                res = max(res, height[j] \* (j - i))

                j -= 1

        return res

# 108.括号生成

**题目链接：**[https://leetcode-cn.com/problems/generate-parentheses/](https://leetcode-cn.com/problems/generate-parentheses/" \t "_blank)

**题目描述：**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 ：

输入：n = 3

输出：["((()))","(()())","(())()","()(())","()()()"]

**参考答案：**

from typing import List

class Solution:

    def generateParenthesis(self, n: int) -> List[str]:

        if n == 0:

            return []

        dp = [None for \_ in range(n + 1)]

        dp[0] = [""]

        for i in range(1, n + 1):

            cur = []

            for j in range(i):

                left = dp[j]

                right = dp[i - j - 1]

                for s1 in left:

                    for s2 in right:

                        cur.append("(" + s1 + ")" + s2)

            dp[i] = cur

        return dp[n]

# 109.全排列

**题目链接：**[https://leetcode-cn.com/problems/permutations/](https://leetcode-cn.com/problems/permutations/" \t "_blank)

**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]]

**参考答案：**

class Solution:

    def permute(self, nums: List[int]) -> List[List[int]]:

        if len(nums) <= 1:  # 递归终止条件

            return [nums]

        res = []

        for idx, num in enumerate(nums):

            res\_nums = nums[:idx] + nums[idx + 1:]  # 确定剩余元素

            for j in self.permute(res\_nums):

                res.append([num] + j)

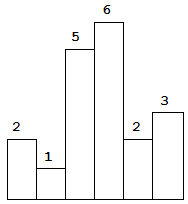
        return res

# 110.柱状图中最大的矩阵

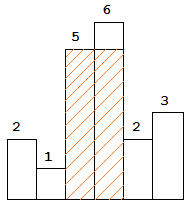
**题目连接：**[https://leetcode-cn.com/problems/largest-rectangle-in-histogram/](https://leetcode-cn.com/problems/largest-rectangle-in-histogram/" \t "_blank)

**题目描述：**给定 n 个非负整数，用来表示柱状图中各个柱子的高度。每个柱子彼此相邻，且宽度为 1 。

求在该柱状图中，能够勾勒出来的矩形的最大面积。



以上是柱状图的示例，其中每个柱子的宽度为 1，给定的高度为 [2,1,5,6,2,3]。



图中阴影部分为所能勾勒出的最大矩形面积，其面积为 10 个单位。

示例：

输入：[2,1,5,6,2,3]

输出：10

**参考答案：**

class Solution:

    def largestRectangleArea(self, heights: List[int]) -> int:

        stack = []

        heights = [0] + heights + [0]

        res = 0

        for i in range(len(heights)):

            #print(stack)

            while stack and heights[stack[-1]] > heights[i]:

                tmp = stack.pop()

                res = max(res, (i - stack[-1] - 1) \* heights[tmp])

            stack.append(i)

        return res

# 111.覆盖

**题目链接：**[https://leetcode-cn.com/problems/broken-board-dominoes/](https://leetcode-cn.com/problems/broken-board-dominoes/" \t "_blank)

**题目描述：**你有一块棋盘，棋盘上有一些格子已经坏掉了。你还有无穷块大小为1 \* 2的多米诺骨牌，你想把这些骨牌不重叠地覆盖在完好的格子上，请找出你最多能在棋盘上放多少块骨牌？这些骨牌可以横着或者竖着放。

输入：n, m代表棋盘的大小；broken是一个b \* 2的二维数组，其中每个元素代表棋盘上每一个坏掉的格子的位置。

输出：一个整数，代表最多能在棋盘上放的骨牌数。

示例1：

输入：n = 2，m = 3，broken = [[1,0][1,1]]

输出：2

示例2：

输入：n = 3, m = 3, broken = []

输出：4

**参考答案：**

class Solution:

    def domino(self, n: int, m: int, broken: List[List[int]]) -> int:

        def code(i, j):

            return 1 << (i \* m + j)

        # state 记录当前grid的填充状态，为什么不用二维矩阵，因为用一个整数可以更方便的做cache，而且这个问题的数据量较小可以用整数

        # 64bit的整数可以包括所有的可能。

        state = 0

        for b in broken:

            state |= code(b[0], b[1])

        @lru\_cache(None)

        def backtrace(i, j, state):

            if i == n:

                # 递归到头了

                return 0

            result = 0

            if j == m:

                # 到达最右，下一行

                result = max(result, backtrace(i + 1, 0, state))

            else:

                skip\_cur\_node = True

                if state & code(i, j) == 0:

                    if i < n - 1 and (state & code(i + 1, j) == 0):

                        # 向下

                        skip\_cur\_node = False

                        state |= code(i, j)

                        state |= code(i + 1, j)

                        result = max(result, backtrace(i, j + 1, state) + 1)

                        state &= ~code(i, j)

                        state &= ~code(i + 1, j)

                    if j < m - 1 and (state & code(i, j + 1) == 0):

                        # 向右

                        skip\_cur\_node = False

                        state |= code(i, j)

                        state |= code(i, j + 1)

                        result = max(result, backtrace(i, j + 2, state) + 1)

                        state &= ~code(i, j)

                        state &= ~code(i + 1, j)

                if skip\_cur\_node:

                    # 当前点不能加牌，跳过

                    result = max(result, backtrace(i, j + 1, state))

            # 返回各种可能中的最大值

            return result

        return backtrace(0, 0, state)

# 112.切分数组

**题目链接**：[https://leetcode-cn.com/problems/qie-fen-shu-zu/](https://leetcode-cn.com/problems/qie-fen-shu-zu/" \t "_blank)

**题目描述：**给定一个整数数组 nums ，小李想将 nums 切割成若干个非空子数组，使得每个子数组最左边的数和最右边的数的最大公约数大于 1 。为了减少他的工作量，请求出最少可以切成多少个子数组。

示例1：

输入：nums = [2,3,3,2,3,3]

输出：2

示例2：

输入：[2,3,5,7]

输出：4

**参考答案：**

import math

class Solution:

    def splitArray(self, nums: [int]) -> int:

        n = len(nums)

        maxn = max(nums)

        c = [0]\*(maxn+1)

        tot = 0

        t = 0

        for i in range(2, int(math.sqrt(maxn))+1):  //注意无需循环结束，到sqrtN即可

            if not c[i]:

                for j in range(i\*i, maxn+1, i):

                    if not c[j]:

                        c[j] = i

        for i in range(2, maxn+1):     //再枚举整个范围，处理所有数字的最小质因数

            if not c[i]:

                c[i] = i

        dp = [n]\*(maxn+1)  //数以因为是最小值，初始值要尽量大

        f = [n]\*(n+1)

        f[-1] = 0  //初值位0，方便直接转移

        d = 0

        for i in nums:

            while i>1:

                tmp = c[i]

                if f[d-1]<dp[tmp]:dp[tmp] = f[d-1]

                if dp[tmp]+1<f[d]:f[d] = dp[tmp]+1

                i //= tmp

            d += 1

        return f[n-1]

113.寻找右区间

**题目链接**：[https://leetcode-cn.com/problems/find-right-interval/](https://leetcode-cn.com/problems/find-right-interval/" \t "_blank)

**题目描述：**给定一组区间，对于每一个区间 i，检查是否存在一个区间 j，它的起始点大于或等于区间 i 的终点，这可以称为 j 在 i 的“右侧”。

对于任何区间，你需要存储的满足条件的区间 j 的最小索引，这意味着区间 j 有最小的起始点可以使其成为“右侧”区间。如果区间 j 不存在，则将区间 i 存储为 -1。最后，你需要输出一个值为存储的区间值的数组。

示例1：

输入：[[1,2]]

输出：[-1]

示例2：

输入：[[3,4],[2,3],[1,2]]

输出：[-1,0,1]

**参考答案：**

    # 二分法——把左边界提取出来排序，这样通过二分法找第一个大于等于右边界的数。

    def findRightInterval(self, intervals: List[List[int]]) -> List[int]:

        res = []

        loc = []

        for idx, val in enumerate(intervals):

            loc.append([val[0], idx])

        loc.sort()                                       # 把左边界提取出来排序，并记录对应index.

        print(loc)

        loc\_left = list(map(lambda x:x[0], loc))         # 取出已排序的左边界。

        for \_, right in intervals:

            index = self.searchInsert(loc\_left, right)   # 二分查找：第一个大于等于右边界的数。

            if index >= len(loc):

                res.append(-1)

            else:

                res.append(loc[index][1])                # 返回查找到的左边界的index，结束。

        return res

    def searchInsert(self, nums, target):

        # 35. 搜索插入位置（题解）

        length = len(nums)

        left = 0

        right = length - 1

        if target > nums[length - 1]:

            return length

        while left <= right:

            mid = (right + left) >> 1

            if nums[mid] == target:

                return mid

            if nums[mid] < target:

                left = mid + 1

            else:

                right = mid - 1

        return left

# 113.不相交的线

**题目链接：**[https://leetcode-cn.com/problems/uncrossed-lines/](https://leetcode-cn.com/problems/uncrossed-lines/" \t "_blank)

**题目描述：**我们在两条独立的水平线上按给定的顺序写下 A 和 B 中的整数。

现在，我们可以绘制一些连接两个数字 A[i] 和 B[j] 的直线，只要 A[i] == B[j]，且我们绘制的直线不与任何其他连线（非水平线）相交。

以这种方法绘制线条，并返回我们可以绘制的最大连线数。

示例1：

输入：A = [1,4,2], B = [1,2,4]

输出：2

示例2：

输入：A = [2,5,1,2,5], B = [10,5,2,1,5,2]

输出：3

**参考答案:**

class Solution:

    def findLength(self, A, B):

        m, n = len(A), len(B)

        ans = 0

        dp = [[0 for \_ in range(n + 1)] for \_ in range(m + 1)]

        for i in range(1, m + 1):

            for j in range(1, n + 1):

                if A[i - 1] == B[j - 1]:

                    dp[i][j] = dp[i - 1][j - 1] + 1

                    ans = max(ans, dp[i][j])

        return ans

# 114.多边三角形剖分的最低得分

**题目链接：**[https://leetcode-cn.com/problems/minimum-score-triangulation-of-polygon/](https://leetcode-cn.com/problems/minimum-score-triangulation-of-polygon/" \t "_blank)

**题目描述：**给定 N，想象一个凸 N 边多边形，其顶点按顺时针顺序依次标记为 A[0], A[i], ..., A[N-1]。

假设您将多边形剖分为 N-2 个三角形。对于每个三角形，该三角形的值是顶点标记的乘积，三角剖分的分数是进行三角剖分后所有 N-2 个三角形的值之和。

返回多边形进行三角剖分后可以得到的最低分。

示例1：

输入：[1,2,3]

输出：6

示例2：

输入：[3,7,4,5]

输出：144

**参考答案：**

    class Solution:

        def minScoreTriangulation(self, A: List[int]) -> int:

            n = len(A)

            dp = [[0] \* n for \_ in range(n)]

            """

            类似1030

            思路:

            (1)在多边形中,有0<=i < j < n, i,j和i,j间的点k可以构成一个三角形,其中i < k < j

            (2)以i为起始点,以j为终点的多边形三角形剖分的最低得分为dp[i][j]

            dp[i][j] = min(dp[i][k]+dp[k][j]+A[i]\*A[k]\*A[j]) k in [i+1, j)

            (3)当j==i+1时,dp[i][j] = 0(相邻的两个点不能构成三角形)

            假设n==4

            i==0, j==3, k in [1,3)

            k == 1, dp[0][3] = min(dp[0][3], dp[0][1] + (dp[1][3]待算) + A[0]\*A[1]\*A[3])

            k == 2, dp[0][3] = min(dp[0][3], (dp[0][2]待算) + dp[2][3] + A[0]\*A[2]\*A[3])

            i==1, j==3, k in [2,3)

            k == 2, dp[1][3] = min(dp[1][3], dp[1][2] + dp[2][3] + A[1]\*A[2]\*A[3])

            i==0, j==2, k in [1,2)

            k == 2, dp[0][2] = min(dp[0][2], dp[0][1] + dp[1][2] + A[0]\*A[1]\*A[2])

            计算方式

            j in [2,n)

                固定j时,i=j-2,i--

            对于每一个n >= 3,有: i == 0, j == 2, k == 1构成一个三角形,所以j的开始位置始终是2

            """

            for j in range(2, n):

                for i in range(j-2, -1, -1):

                    dp[i][j] = float("inf")

                    for k in range(i+1, j):

                        dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j] + A[i]\*A[k]\*A[j])

            return dp[0][n-1]

# 115.分隔数组以得到最大和

**题目链接：**[https://leetcode-cn.com/problems/partition-array-for-maximum-sum/](https://leetcode-cn.com/problems/partition-array-for-maximum-sum/" \t "_blank)

**题目描述：**给出整数数组 A，将该数组分隔为长度最多为 K 的几个（连续）子数组。分隔完成后，每个子数组的中的值都会变为该子数组中的最大值。

返回给定数组完成分隔后的最大和。

示例：

输入：A = [1,15,7,9,2,5,10], K=3

输出：84

**参考答案：**

class Solution:

    def \_\_init\_\_(self):

        self.memo = {}

    def maxSumAfterPartitioning(self, A: List[int], K: int) -> int:

        if len(A) not in self.memo:

            if len(A) <= K:

                self.memo[len(A)] = max(A) \* len(A)

            else:

                self.memo[len(A)] = max(self.maxSumAfterPartitioning(A[:-k], K) + max(A[-k:]) \* k for k in range(1, K+1))

        return self.memo[len(A)]

# 116.使括号有效的最少添加

**题目链接：**[https://leetcode-cn.com/problems/minimum-add-to-make-parentheses-valid/](https://leetcode-cn.com/problems/minimum-add-to-make-parentheses-valid/" \t "_blank)

**题目描述：**给定一个由 '(' 和 ')' 括号组成的字符串 S，我们需要添加最少的括号（ '(' 或是 ')'，可以在任何位置），以使得到的括号字符串有效。

从形式上讲，只有满足下面几点之一，括号字符串才是有效的：

它是一个空字符串，或者

它可以被写成 AB （A 与 B 连接）, 其中 A 和 B 都是有效字符串，或者

它可以被写作 (A)，其中 A 是有效字符串。

给定一个括号字符串，返回为使结果字符串有效而必须添加的最少括号数。

示例1：

输入："())"

输出：1

示例2：

输入："((("

输出：3

**参考答案：**

class Solution:

    def minAddToMakeValid(self, S: str) -> int:

        stack = []

        for i in S:

            if stack:

                if stack[-1] == '(' and i == ')':

                    stack.pop()

                else:

                    stack.append(i)

            else:

                stack.append(i)

        return (len(stack))

# 117.最佳观光组合

**题目链接：**[https://leetcode-cn.com/problems/best-sightseeing-pair/](https://leetcode-cn.com/problems/best-sightseeing-pair/" \t "_blank)

**题目描述：**给定正整数数组 A，A[i] 表示第 i 个观光景点的评分，并且两个景点 i 和 j 之间的距离为 j - i。

一对景点（i < j）组成的观光组合的得分为（A[i] + A[j] + i - j）：景点的评分之和减去它们两者之间的距离。

返回一对观光景点能取得的最高分。

示例：

输入：[8,1,5,2,6]

输出：11

**参考答案：**

class Solution:

    def maxScoreSightseeingPair(self, A: List[int]) -> int:

        dp = A[0] # 初始状态，选择第一个观光点

        res = 0 # 结果变量

        for j in range(1, len(A)):

            res = max(res, dp+A[j]-j) # 选择第二个观光点，计算得分，更新最高分

            dp = max(dp, A[j]+j) # 更新第一个观光点

        return res

# 118.交换一次的先前排列

**题目链接：**[https://leetcode-cn.com/problems/previous-permutation-with-one-swap/](https://leetcode-cn.com/problems/previous-permutation-with-one-swap/" \t "_blank)

**题目描述：**给你一个正整数的数组 A（其中的元素不一定完全不同），请你返回可在 一次交换（交换两数字 A[i] 和 A[j] 的位置）后得到的、按字典序排列小于 A 的最大可能排列。

如果无法这么操作，就请返回原数组。

示例1：

输入：[3,2,1]

输出：[3,1,2]

示例2：

输入：[1,1,5]

输入：[1,1,5]

**参考答案：**

class Solution:

    def prevPermOpt1(self, A: List[int]) -> List[int]:

        lenth = len(A)

        for i in range(lenth-1, 0, -1):

            # 第一次升序的位置，左侧元素A[i-1]大于当前元素A[i]

            if A[i-1] > A[i]:

                # 逆序从结尾向第i-1个索引查找A[j]

                for j in range(lenth-1, i-1, -1):

                    # 元素要满足小于A[i-1]且与左侧元素不相等就是要找的最大元素

                    if A[j] < A[i-1] and A[j] != A[j-1]:

                        A[i-1],A[j] = A[j],A[i-1]

                        return A

        return A

# 119.最短公共超序列

**题目链接：**[https://leetcode-cn.com/problems/shortest-common-supersequence/](https://leetcode-cn.com/problems/shortest-common-supersequence/" \t "_blank)

**题目描述：**给出两个字符串 str1 和 str2，返回同时以 str1 和 str2 作为子序列的最短字符串。如果答案不止一个，则可以返回满足条件的任意一个答案。

（如果从字符串 T 中删除一些字符（也可能不删除，并且选出的这些字符可以位于 T 中的 任意位置），可以得到字符串 S，那么 S 就是 T 的子序列）

示例：

输入：str1 = "abac", str2 = "cab"

输出："cabac"

**参考答案：**

from functools import lru\_cache

class Solution:

    @lru\_cache(typed=False, maxsize=128000000)

    def lcs(self, i, j):

        s1, s2 = self.s1, self.s2

        if i < 0 or j < 0:

            return ''

        if s1[i] == s2[j]:

            ret = self.lcs(i-1, j-1)

            return ret + s1[i]

        ret1, ret2 = self.lcs(i - 1, j), self.lcs(i, j-1)

        return ret1 if len(ret1) > len(ret2) else ret2

    def shortestCommonSupersequence(self, str1: str, str2: str) -> str:

        self.s1, self.s2 = str1, str2

        comm\_str = self.lcs(len(str1)-1, len(str2)-1)

        pos1, pos2 = [], []     # 两个字符串和最长公共子序列的匹配位置

        i, j = 0, 0

        while i < len(comm\_str):

            while str1[j] != comm\_str[i]:

                j += 1

            pos1.append(j)

            i, j = i+1, j+1

        i, j = 0, 0

        while i < len(comm\_str):

            while str2[j] != comm\_str[i]:

                j += 1

            pos2.append(j)

            i, j = i + 1, j + 1

        pos1.append(len(str1))

        pos2.append(len(str2))

        prev\_pos1, prev\_pos2 = -1, -1

        ans = ''

        for p1, p2 in zip(pos1, pos2):

            ans += str1[prev\_pos1+1: p1]

            ans += str2[prev\_pos2+1: p2]

            if p1 == pos1[-1]:

                break

            ans += str1[p1]

            prev\_pos1, prev\_pos2 = p1, p2

        return ans

# 120.从链表中删去总和值为零的连续节点

**题目链接：**[https://leetcode-cn.com/problems/remove-zero-sum-consecutive-nodes-from-linked-list/](https://leetcode-cn.com/problems/remove-zero-sum-consecutive-nodes-from-linked-list/" \t "_blank)

**题目描述：**给你一个链表的头节点 head，请你编写代码，反复删去链表中由 总和 值为 0 的连续节点组成的序列，直到不存在这样的序列为止。

删除完毕后，请你返回最终结果链表的头节点。

你可以返回任何满足题目要求的答案。

示例1：

输入：head = [1,2,-3,3,1]

输出：[3,1]

示例2：

输入：[1,2,3,-3,4]

输出：[1,2,4]

**参考答案：**

class Solution:

    def removeZeroSumSublists(self, head: ListNode) -> ListNode:

        l = ListNode(0)

        l.next = head

        d = {0:l}

        s = 0

        while head:

            s+=head.val

            if s in d:

                d[s].next = head.next

                return self.removeZeroSumSublists(l.next)

            else:

                d[s] = head

                head = head.next

        return l.next