**上海市“星光计划”**

**第十二届职业院校职业技能大赛**

“Python程序设计”项目

中职组

**竞赛样题**

上海市星光计划组委会竞赛办公室

二〇二五年4月

**竞赛任务及要求**

**一、模块A：Python编程判断**

## **1.**两数之和

在给定的整数数组 nums 和整数目标值 target 的情况下，以下代码能够正确找到数组中两个数的和等于目标值 target，并返回它们的数组下标。请判断以下代码是否正确。

def solutionSubmit(nums, target):

 num\_to\_index = {}

 for index, num in enumerate(nums):

 complement = target - num

 if complement in num\_to\_index:

 return [num\_to\_index[complement], index]

 num\_to\_index[num] = index

**2.回文数**

给你一个整数 x ，如果 x 是一个回文整数，返回 true ；否则，返回 false 。

回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。

例如，121 是回文，而 123 不是。

def solutionSubmit(x):

 if x % 10 == 0 and x != 0:

 return False

 reversed\_half = 0

 while x > reversed\_half:

 reversed\_half = reversed\_half \* 10 + x % 10

 x //= 10

return x == reversed\_half or x == reversed\_half // 10

这段代码能够判断回文数

**3.罗马数字转整数**

罗马数字包含以下七种字符: I，V，X，L，C，D 和 M。

字符 数值

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

例如，罗马数字 2 写做 II ，即为两个并列的 1。12 写做 XII ，即为 X + II 。27 写做 XXVII, 即为 XX + V + II 。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 IIII，而是 IV。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4。同样地，数字 9 表示为 IX。这个特殊的规则只适用于以下六种情况：

I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。

X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。

C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。

def solutionSubmit(s):

 roman\_to\_int = {

 'I': 1,

 'V': 5,

 'X': 10,

 'L': 50,

 'C': 100,

 'D': 500,

 'M': 1000

 }

 total = 0

 prev\_value = 0

 for char in reversed(s):

 current\_value = roman\_to\_int[char]

 if current\_value < prev\_value:

 total -= current\_value

 else:

 total += current\_value

 prev\_value = current\_value

 return total

这段代码能将给定的罗马数字，将其转换成整数。

**4.最长公共前缀**

编写一个函数来查找字符串数组中的最长公共前缀。

如果不存在公共前缀，返回空字符串 ""。

def solutionSubmit(strs):

 if not strs:

 return ""

 # 找到最短的字符串

 min\_str = min(strs, key=len)

 for i in range(len(min\_str)):

 for s in strs:

 if s[i] != min\_str[i]:

 return min\_str[:i]

 return min\_str

这段代码能够达到要求

**5.有效的括号**

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

有效字符串需满足：

左括号必须用相同类型的右括号闭合。

左括号必须以正确的顺序闭合。

每个右括号都有一个对应的相同类型的左括号。

def solutionSubmit(s):

 stack = []

 bracket\_map = {')': '(', '}': '{', ']': '['}

 for char in s:

 if char in bracket\_map:

 top\_element = stack.pop() if stack else '#'

 if bracket\_map[char] != top\_element:

 return False

 else:

 stack.append(char)

 return not stack

这段代码能够达到要求

**6.删除有序数组中的重复项**

给你一个 非严格递增排列 的数组 nums ，请你原地删除重复出现的元素，使每个元素 只出现一次 ，返回删除后数组的新长度。元素的 相对顺序 应该保持 一致 。然后返回 nums 中唯一元素的个数。

考虑 nums 的唯一元素的数量为 k ，你需要做以下事情确保你的题解可以被通过：

更改数组 nums ，使 nums 的前 k 个元素包含唯一元素，并按照它们最初在 nums 中出现的顺序排列。nums 的其余元素与 nums 的大小不重要。

返回 k 。

def solutionSubmit(nums):

 k = 1

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

 if nums[i] != nums[k - 1]:

 nums[k] = nums[i]

 k += 1

 return k

这段代码能够达到要求

**7.找出字符串中第一个匹配项的下标**

给你两个字符串 haystack 和 needle ，请你在 haystack 字符串中找出 needle 字符串的第一个匹配项的下标（下标从 0 开始）。如果 needle 不是 haystack 的一部分，则返回 -1。

def solutionSubmit(haystack, needle):

 return haystack.find(needle)

这段代码能够达到要求

**8.爬楼梯**

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

def solutionSubmit(n):

 if n < 2:

 return n

 first, second = 1, 2

 for \_ in range(3, n + 1):

 first, second = second, first + second

 return second

这段代码能够达到要求

**9.只出现一次的数字**

给你一个 非空 整数数组 nums ，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

你必须设计并实现线性时间复杂度的算法来解决此问题，且该算法只使用常量额外空间。

def solutionSubmit(nums):

 result = 0

 for num in nums:

 result ^= num

 return result

这段代码能够达到要求

**10.验证回文串**

给你一个字符串 s ，最多 可以从中删除一个字符。

请你判断 s 是否能成为回文字符串：如果能，返回 true ；否则，返回 false 。s 由小写英文字母组成。

def solutionSubmit(s):

 def is\_palindrome\_range(left, right):

 while left < right:

 if s[left] != s[right]:

 return False

 left += 1

 right -= 1

 return True

 if not s:

 return False

 left, right = 0, len(s) - 1

 while left < right:

 if s[left] != s[right]:

 return is\_palindrome\_range(left + 1, right) or is\_palindrome\_range(left, right - 1)

 left += 1

 right -= 1

 return True

这段代码能够达到要求

**二、模块B：Python程序补全**

**1.寻找两个正序数组的中位数**

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

算法的时间复杂度应该为 O(log (m+n)) 。

在以下代码中，partition1 和 partition2 的作用是什么？

def solutionSubmit(nums1, nums2):

 # Ensure nums1 is the smaller array

 if len(nums1) > len(nums2):

 nums1, nums2 = nums2, nums1

 m, n = len(nums1), len(nums2)

 left, right = 0, m

 while left <= right:

 partition1 = (left + right) // 2

 partition2 = (m + n + 1) // 2 - partition1

 maxLeft1 = float('-inf') if partition1 == 0 else nums1[partition1 - 1]

 minRight1 = float('inf') if partition1 == m else nums1[partition1]

 maxLeft2 = float('-inf') if partition2 == 0 else nums2[partition2 - 1]

 minRight2 = float('inf') if partition2 == n else nums2[partition2]

 if maxLeft1 <= minRight2 and maxLeft2 <= minRight1:

 if (m + n) % 2 == 0:

 return (max(maxLeft1, maxLeft2) + min(minRight1, minRight2)) / 2

 else:

 return max(maxLeft1, maxLeft2)

 elif maxLeft1 > minRight2:

 right = partition1 - 1

 else:

 left = partition1 + 1

A. partition1 和 partition2 用于将 nums1 和 nums2 分别分成两部分，使得左半部分的最大值小于等于右半部分的最小值。

B. partition1 和 partition2 用于将 nums1 和 nums2 分别分成两部分，使得左半部分的最小值小于等于右半部分的最大值。

C. partition1 和 partition2 用于将 nums1 和 nums2 分别分成两部分，使得左半部分的最大值大于等于右半部分的最小值。

D. partition1 和 partition2 用于将 nums1 和 nums2 分别分成两部分，使得左半部分的最小值大于等于右半部分的最大值。

**2.最长回文子串**

给你一个字符串 s，找到 s 中最长的回文子串。s 仅由数字和英文字母组成。

阅读以下代码，选择正确的选项：

def solutionSubmit(s):

 n = len(s)

 if n <= 1:

 return s

 dp = [[False] \* n for \_ in range(n)]

 longest = s[0]

 for i in range(n):

 dp[i][i] = True

 for i in range(n - 1):

 if s[i] == s[i + 1]:

 dp[i][i + 1] = True

 longest = s[i:i + 2]

 for length in range(3, n + 1):

 for i in range(n - length + 1):

 j = i + length - 1

 if s[i] == s[j] and dp[i + 1][j - 1]:

 dp[i][j] = True

 if length > len(longest):

 longest = s[i:j + 1]

 return longest

A. 使用动态规划算法，时间复杂度为 O(n^2)，空间复杂度为 O(n^2)。

B. 使用暴力搜索算法，时间复杂度为 O(n^3)，空间复杂度为 O(1)。

C. 使用滑动窗口算法，时间复杂度为 O(n)，空间复杂度为 O(1)。

D. 使用贪心算法，时间复杂度为 O(n log n)，空间复杂度为 O(n)。

**3.Z 字形变换**

将一个给定字符串 s 根据给定的行数 numRows，以从上往下、从左到右进行 Z 字形排列。比如输入字符串为 "PAYPALISHIRING" 行数为 3 时，排列如下：

P A H N

A P L S I I G

Y I R

之后，你的输出需要从左往右逐行读取，产生出一个新的字符串，比如："PAHNAPLSIIGYIR"。请你选择以下哪个选项是正确的实现代码？

A.

def convert(s, numRows):

 if numRows == 1 or numRows >= len(s):

 return s

 rows = [''] \* numRows

 current\_row = 0

 going\_down = True

 for char in s:

 rows[current\_row] += char

 if current\_row == 0:

 going\_down = True

 elif current\_row == numRows - 1:

 going\_down = False

 current\_row += 1 if going\_down else -1

 return ''.join(rows)

B.

def convert(s, numRows):

 if numRows == 1 or numRows >= len(s):

 return s

 rows = [''] \* numRows

 current\_row = 0

 going\_down = False

 for char in s:

 rows[current\_row] += char

 if current\_row == 0:

 going\_down = True

 elif current\_row == numRows - 1:

 going\_down = False

 current\_row += 1 if going\_down else -1

 return ''.join(rows)

C.

def convert(s, numRows):

 if numRows == 1 or numRows >= len(s):

 return s

 rows = [''] \* numRows

 current\_row = 0

 going\_down = True

 for char in s:

 rows[current\_row] += char

 if current\_row == 0:

 going\_down = False

 elif current\_row == numRows - 1:

 going\_down = True

 current\_row += 1 if going\_down else -1

 return ''.join(rows)

D.

def convert(s, numRows):

 if numRows == 1 or numRows >= len(s):

 return s

 rows = [''] \* numRows

 current\_row = 0

 going\_down = True

 for char in s:

 rows[current\_row] += char

 if current\_row == 0:

 going\_down = True

 elif current\_row == numRows - 1:

 going\_down = False

 current\_row += -1 if going\_down else 1

 return ''.join(rows)

**4.整数反转**

给定一个 32 位的有符号整数 x，返回将 x 中的数字部分反转后的结果。如果反转后整数超过 32 位的有符号整数的范围 [−2^31, 2^31 − 1]，就返回 0。假设环境不允许存储 64 位整数（有符号或无符号）。

以下哪个选项正确地实现了整数反转的功能？

A.

def solutionSubmit(x):

 sign = -1 if x < 0 else 1

 x \*= sign

 reversed\_x = 0

 while x != 0:

 digit = x % 10

 x //= 10

 if reversed\_x > (2\*\*31 - 1) // 10 or (reversed\_x == (2\*\*31 - 1) // 10 and digit > 7):

 return 0

 reversed\_x = reversed\_x \* 10 + digit

 return sign \* reversed\_x

B.

def solutionSubmit(x):

 sign = -1 if x < 0 else 1

 x \*= sign

 reversed\_x = 0

 while x != 0:

 digit = x % 10

 x //= 10

 if reversed\_x > (2\*\*31 - 1) // 10 or (reversed\_x == (2\*\*31 - 1) // 10 and digit > 8):

 return 0

 reversed\_x = reversed\_x \* 10 + digit

 return sign \* reversed\_x

C.

def solutionSubmit(x):

 sign = -1 if x < 0 else 1

 x \*= sign

 reversed\_x = 0

 while x != 0:

 digit = x % 10

 x //= 10

 if reversed\_x > (2\*\*31 - 1) // 10 or (reversed\_x == (2\*\*31 - 1) // 10 and digit > 6):

 return 0

 reversed\_x = reversed\_x \* 10 + digit

 return sign \* reversed\_x

D.

def solutionSubmit(x):

 sign = -1 if x < 0 else 1

 x \*= sign

 reversed\_x = 0

 while x != 0:

 digit = x % 10

 x //= 10

 if reversed\_x > (2\*\*31 - 1) // 10 or (reversed\_x == (2\*\*31 - 1) // 10 and digit > 9):

 return 0

 reversed\_x = reversed\_x \* 10 + digit

 return sign \* reversed\_x

**5.字符串转换整数**

请你来实现一个 myAtoi(string s) 函数，使其能将字符串转换成一个 32 位有符号整数。

函数 myAtoi(string s) 的算法如下：

空格：读入字符串并丢弃无用的前导空格（" "）。

符号：检查下一个字符（假设还未到字符末尾）为 '-' 还是 '+'。如果两者都不存在，则假定结果为正。

转换：通过跳过前置零来读取该整数，直到遇到非数字字符或到达字符串的结尾。如果没有读取数字，则结果为0。

舍入：如果整数数超过 32 位有符号整数范围 [−2^31, 2^31 − 1] ，需要截断这个整数，使其保持在这个范围内。具体来说，小于 −2^31 的整数应该被舍入为 −2^31 ，大于 2^31 − 1 的整数应该被舍入为 2^31 − 1 。

返回整数作为最终结果。

提示：

0 <= s.length <= 200

s 由英文字母（大写和小写）、数字（0-9）、' '、'+'、'-' 和 '.' 组成。

以下是代码：

def solutionSubmit(s):

 s = s.lstrip() # 去掉前导空格

 if not s:

 return 0

 sign = 1 # 默认正数

 index = 0

 result = 0

 INT\_MAX = 2\*\*31 - 1

 INT\_MIN = -2\*\*31

 # 处理符号

 if s[index] == '-':

 sign = -1

 index += 1

 elif s[index] == '+':

 index += 1

 # 处理数字

 while index < len(s) and s[index].isdigit():

 digit = int(s[index])

 # 检查是否会溢出

 if result > (INT\_MAX - digit) // 10:

 return INT\_MIN if sign == -1 else INT\_MAX

 result = result \* 10 + digit

 index += 1

 return sign \* result

那么，myAtoi(string s) 函数的以下哪个描述是正确的？

A. myAtoi 函数忽略前导空格，并将字符串的前导零去掉，返回剩余数字。

B. myAtoi 函数忽略前导空格，并正确处理字符串中的除字母外的所有字符。

C. myAtoi 函数会忽略前导空格，并转换字符串为整数，但不会处理溢出情况。

D. myAtoi 函数通过忽略前导空格，处理可能的正负号，转换数字部分，并截断整数到 32 位范围，来将字符串转换为整数。

**6.盛最多水的容器**

给定一个长度为 n 的整数数组 height。有 n 条垂线，第 i 条线的两个端点是 (i, 0) 和 (i, height[i])。

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。

返回容器可以储存的最大水量。

说明：你不能倾斜容器。

示例 1：



输入：[1,8,6,2,5,4,8,3,7]

输出：49

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

以下是代码：

def solutionSubmit(height):

 left = 0

 right = len(height) - 1

 max\_area = 0

 while left < right:

 width = right - left

 current\_height = min(height[left], height[right])

 current\_area = width \* current\_height

 max\_area = max(max\_area, current\_area)

 if height[left] < height[right]:

 left += 1

 else:

 right -= 1

 return max\_area

给定输入数组 [1,8,6,2,5,4,8,3,7]，最大水量是多少？

A. 36

B. 49

C. 42

D. 63

**7.三数之和**

给你一个整数数组 nums ，判断是否存在三元组 [nums[i], nums[j], nums[k]] 满足 i != j、i != k 且 j != k，同时还满足 nums[i] + nums[j] + nums[k] == 0 。请你返回所有和为 0 且不重复的三元组。

注意：答案中不可以包含重复的三元组。

以下是代码：

def solutionSubmit(nums):

 nums.sort()

 res = []

 n = len(nums)

 for i in range(n):

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

 continue # Skip duplicates for the first number

 left, right = i + 1, n - 1

 while left < right:

 total = nums[i] + nums[left] + nums[right]

 if total < 0:

 left += 1

 elif total > 0:

 right -= 1

 else:

 res.append([nums[i], nums[left], nums[right]])

 while left < right and nums[left] == nums[left + 1]:

 left += 1 # Skip duplicates for the second number

 while left < right and nums[right] == nums[right - 1]:

 right -= 1 # Skip duplicates for the third number

 left += 1

 right -= 1

 return res

给定输入数组为 [-1, 0, 1, 2, -1, -4]，以下哪个选项最符合该问题的正确输出？

A. [[-1, 0, 1], [-1, -1, 2]]

B. [[0, 0, 0], [1, -1, 0]]

C. [[2, -1, -1], [1, 0, -1]]

D. []

**8.最接近的三数之和**

给你一个长度为 n 的整数数组 nums 和一个目标值 target。请你从 nums 中选出三个整数，使它们的和与 target 最接近。

返回这三个数的和。

假定每组输入只存在恰好一个解。

以下是代码：
def solutionSubmit(nums, target):

 nums.sort()

 closest\_sum = float('inf')

 n = len(nums)

 for i in range(n - 2):

 left, right = i + 1, n - 1

 while left < right:

 current\_sum = nums[i] + nums[left] + nums[right]

 if abs(current\_sum - target) < abs(closest\_sum - target):

 closest\_sum = current\_sum

 if current\_sum < target:

 left += 1

 elif current\_sum > target:

 right -= 1

 else:

 return current\_sum

 return closest\_sum

输入：nums = [-1, 2, 1, -4], target = 1，输出的正确结果是？

A. 0

B. 1

C. 2

D. -1

**9.电话号码的字母组合**

给定一个仅包含数字 2-9 的字符串，返回所有它能表示的字母组合。答案可以按任意顺序返回。

给出数字到字母的映射如下（与电话按键相同）。



注意 1 不对应任何字母。

以下是代码：
 def backtrack(index, path):

 if index == len(digits):

 combinations.append("".join(path))

 return

 possible\_letters = phone\_map[digits[index]]

 for letter in possible\_letters:

 path.append(letter)

 backtrack(index + 1, path)

 path.pop()

 combinations = []

 backtrack(0, [])

 return combinations

哪一个是上述问题的正确输出？

A. 对于输入 "35"，返回

["dj","dk","dl","ej","ek","el","fj","fk","fl"]

B. 对于输入 "49"，返回

["gw","gx","gy","gz","hw","hx","hy","hz","iw","ix","iy","iz"]

C. 对于输入 "2"，返回 ["d","e","f"]

D. 对于输入 "5"，返回 []

**10.四数之和**

给你一个由 n 个整数组成的数组 nums ，和一个目标值 target 。请你找出并返回满足下述全部条件且不重复的四元组 [nums[a], nums[b], nums[c], nums[d]] （若两个四元组元素一一对应，则认为两个四元组重复）：

0 <= a, b, c, d < n

a、b、c 和 d 互不相同

nums[a] + nums[b] + nums[c] + nums[d] == target

你可以按 任意顺序 返回答案 。

代码如下：

def solutionSubmit(nums, target):

 nums.sort()

 result = []

 n = len(nums)

 for i in range(n):

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

 continue

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

 if j > i + 1 and nums[j] == nums[j - 1]:

 continue

 left, right = j + 1, n - 1

 while left < right:

 total = nums[i] + nums[j] + nums[left] + nums[right]

 if total < target:

 left += 1

 elif total > target:

 right -= 1

 else:

 result.append([nums[i], nums[j], nums[left], nums[right]])

 while left < right and nums[left] == nums[left + 1]:

 left += 1

 while left < right and nums[right] == nums[right - 1]:

 right -= 1

 left += 1

 right -= 1

 return result

请问，下面哪一个选项最符合上述例子的解题思路？

A.将数组排序后，遍历每个元素，并在确定第一个元素后用二分查找确定剩下三个元素的位置和组合。

B. 将数组中的数字分为两组，分别求和，再将两组的和进行配对，找出相对等于目标值的组合。

C. 遍历数组中的每个元素，如果它与前一个元素相同则跳过，继续向后遍历。如果遍历到的四个数的和等于目标值，则记录下来，否则调整两个指针的位置，继续判断。

D. 将数组作为键，将数组中每个数字出现的次数作为值，生成一个哈希表，最后判断四个数字的组合是否在哈希表中存在。

**三、模块C：Python程序开发**

**1.正则表达式匹配**

给你一个字符串 s 和一个字符规律 p，请你来实现一个支持 '.' 和 '\*' 的正则表达式匹配。

'.' 匹配任意单个字符

'\*' 匹配零个或多个前面的那一个元素

所谓匹配，是要涵盖整个字符串 s 的，而不是部分字符串。

示例 1：

输入：s = "aa", p = "a"

输出：false

解释："a" 无法匹配 "aa" 整个字符串。

示例 2:

输入：s = "aa", p = "a\*"

输出：true

解释：因为 '\*' 代表可以匹配零个或多个前面的那一个元素, 在这里前面的元素就是 'a'。因此，字符串 "aa" 可被视为 'a' 重复了一次。

示例 3：

输入：s = "ab", p = "."

输出：true

解释："." 表示可匹配零个或多个（'\*'）任意字符（'.'）。

提示：

1 <= s.length <= 20

1 <= p.length <= 20

s 只包含从 a-z 的小写字母。

p 只包含从 a-z 的小写字母，以及字符 . 和 \*。

保证每次出现字符 \* 时，前面都匹配到有效的字符。

**2.最长有效括号**

给你一个只包含 '(' 和 ')' 的字符串，找出最长有效（格式正确且连续）括号子串的长度。

示例 1：

输入：s = "(())"

输出：2

解释：最长有效括号子串是 "()"

示例 2：

输入：s = ")()())"

输出：4

解释：最长有效括号子串是 "()()"

示例 3：

输入：s = ""

输出：0

提示：

0 ≤ s.length ≤ 3 \* 10^4

s[i] 为 '(' 或 ')'

**3.接雨水**

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

示例 1：



输入：height = [0,1,0,2,1,0,1,3,2,1,2,1]

输出：6

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

示例 2：

输入：height = [4,2,0,3,2,5]

输出：9

提示：

n == height.length

1 <= n <= 2 \* 10^4

0 <= height[i] <= 10^5

**4.通配符匹配**

给你一个输入字符串 (s) 和一个字符模式 (p) ，请你实现一个支持 '?' 和 '\*' 匹配规则的通配符匹配：

'?' 可以匹配任何单个字符。

'\*' 可以匹配任意字符序列（包括空字符序列）。

判定匹配成功的充要条件是：字符模式必须能够 完全匹配 输入字符串（而不是部分匹配）。

示例 1：

输入：s = "aa", p = "a"

输出：false

解释："a" 无法匹配 "aa" 整个字符串。

示例 2：

输入：s = "aa", p = ""

输出：true

解释：'' 可以匹配任意字符串。

示例 3：

输入：s = "cb", p = "?a"

输出：false

解释：'?' 可以匹配 'c', 但第二个 'a' 无法匹配 'b'。

提示：

0 <= s.length, p.length <= 2000

s 仅由小写英文字母组成

p 仅由小写英文字母、'?' 或 '\*' 组成

**5.N皇后II**

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。

给你一个整数 n ，返回 n 皇后问题 不同的解决方案的数量。

示例 1：



输入：n = 4

输出：2

解释：如上图所示，4 皇后问题存在两个不同的解法。

示例 2：

输入：n = 1

输出：1

提示：

1 <= n <= 9

**6.地下城游戏**

恶魔们抓住了公主并将她关在了地下城 dungeon 的 右下角 。地下城是由 m x n 个房间组成的二维网格。我们英勇的骑士最初被安置在 左上角 的房间里，他必须穿过地下城并通过对抗恶魔来拯救公主。

骑士的初始健康点数为一个正整数。如果他的健康点数在某一时刻降至 0 或以下，他会立即死亡。

有些房间由恶魔守卫，因此骑士在进入这些房间时会失去健康点数（若房间里的值为负整数，则表示骑士将损失健康点数）；其他房间要么是空的（房间里的值为 0），要么包含增加骑士健康点数的魔法球（若房间里的值为正整数，则表示骑士将增加健康点数）。

为了尽快解救公主，骑士决定每次只 向右 或 向下 移动一步。

返回确保骑士能够拯救到公主所需的最低初始健康点数。

注意：任何房间都可能对骑士的健康点数造成威胁，也可能增加骑士的健康点数，包括骑士进入的左上角房间以及公主被监禁的右下角房间。

示例 1：



输入：dungeon = [[-2,-3,3],[-5,-10,1],[10,30,-5]]

输出：7

解释：如果骑士遵循最佳路径：右 -> 右 -> 下 -> 下 ，则骑士的初始健康点数至少为 7 。

示例 2：

输入：dungeon = [[0]]

输出：1

提示：

m == dungeon.length

n == dungeon[i].length

1 <= m, n <= 200

-1000 <= dungeon[i][j] <= 1000

**7.买卖股票的最佳时机IV**

给你一个整数数组 prices 和一个整数 k ，其中 prices[i] 是某支给定的股票在第 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 k 笔交易。也就是说，你最多可以买 k 次，卖 k 次。

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：k = 2, prices = [2,4,1]

输出：2

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

示例 2：

输入：k = 2, prices = [3,2,6,5,0,3]

输出：7

解释：在第 2 天 (股票价格 = 2) 的时候买入，在第 3 天 (股票价格 = 6) 的时候卖出，这笔交易所能获得利润 = 6-2 = 4 。

随后，在第 5 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 。

提示：

1 <= k <= 100

1 <= prices.length <= 1000

0 <= prices[i] <= 1000

**8.天际线问题**

城市的 天际线 是从远处观看该城市中所有建筑物形成的轮廓的外部轮廓。给你所有建筑物的位置和高度，请返回 由这些建筑物形成的 天际线 。

每个建筑物的几何信息由数组 buildings 表示，其中三元组 buildings[i] = [lefti, righti, heighti] 表示：

lefti 是第 i 座建筑物左边缘的 x 坐标。

righti 是第 i 座建筑物右边缘的 x 坐标。

heighti 是第 i 座建筑物的高度。

你可以假设所有的建筑都是完美的长方形，在高度为 0 的绝对平坦的表面上。

天际线应该表示为由 “关键点” 组成的列表，格式 [[x1,y1],[x2,y2],...] ，并按 x 坐标 进行 排序 。关键点是水平线段的左端点。列表中最后一个点是最右侧建筑物的终点，y 坐标始终为 0 ，仅用于标记天际线的终点。此外，任何两个相邻建筑物之间的地面都应被视为天际线轮廓的一部分。

注意：输出天际线中不得有连续的相同高度的水平线。例如 [...[2 3], [4 5], [7 5], [11 5], [12 7]...] 是不正确的答案；三条高度为 5 的线应该在最终输出中合并为一个：[...[2 3], [4 5], [12 7], ...]

示例 1：



输入：

buildings = [[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]

输出：

[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]

解释：

图 A 显示输入的所有建筑物的位置和高度，

图 B 显示由这些建筑物形成的天际线。图 B 中的红点表示输出列表中的关键点。

示例 2：

输入：

buildings = [[0,2,3],[2,5,3]]

输出：

[[0,3],[5,0]]

提示：

1 <= buildings.length <= 104

0 <= lefti < righti <= 231 - 1

1 <= heighti <= 231 - 1

buildings 按 lefti 非递减排序

**9.基本计算器**

给你一个字符串表达式 s ，请你实现一个基本计算器来计算并返回它的值。

注意:不允许使用任何将字符串作为数学表达式计算的内置函数，比如 eval() 。

示例 1：

输入：s = "1 + 1"

输出：2

示例 2：

输入：s = " 2-1 + 2 "

输出：3

示例 3：

输入：s = "(1+(4+5+2)-3)+(6+8)"

输出：23

提示：

1 <= s.length <= 3 \* 10^5

s 由数字、'+'、'-'、'('、')'、和 ' ' 组成

s 表示一个有效的表达式

'+' 不能用作一元运算(例如， "+1" 和 "(2 + 3)" 无效)

'-' 可以用作一元运算(即 "-1" 和 "(2 + 3)" 是有效的)

输入中不存在两个连续的操作符

每个数字和运行的计算将适合于一个有符号的 32位 整数

**10.最大矩形**

给定一个仅包含 0 和 1 、大小为 rows x cols 的二维二进制矩阵，找出只包含 1 的最大矩形，并返回其面积。

示例 1：



输入：

matrix =

[["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]

输出：6

解释：最大矩形如上图所示。

示例 2：

输入：matrix = [["0"]]

输出：0

示例 3：

输入：matrix = [["1"]]

输出：1

提示：

rows == matrix.length

cols == matrix[0].length

1 <= row, cols <= 200

matrix[i][j] 为 '0' 或 '1'

**11.戳气球**

有 n 个气球，编号为 0 到 n - 1，每个气球上都标有一个数字，这些数字存在数组 nums 中。

现在要求你戳破所有的气球。戳破第 i 个气球，你可以获得 nums[i - 1] \* nums[i] \* nums[i + 1] 枚硬币。这里的 i - 1 和 i + 1 代表和 i 相邻的两个气球的序号。如果 i - 1 或 i + 1 超出了数组的边界，那么就当它是一个数字为 1 的气球。

求所能获得硬币的最大数量。

示例 1：

输入：nums = [3,1,5,8]

输出：167

解释：

nums = [3,1,5,8] → [3,5,8] → [3,8] → [8] → []

coins = 315 + 358 + 138 + 181 = 167

示例 2：

输入：nums = [1,5]

输出：10

提示：

n == nums.length

1 ≤ n ≤ 300

0 ≤ nums[i] ≤ 100

**12.计算右侧小于当前元素的个数**

给你一个整数数组 nums，按要求返回一个新数组 counts。数组 counts 有该性质：counts[i] 的值是 nums[i] 右侧小于 nums[i] 的元素的数量。

示例 1：

输入： nums = [5,2,6,1]

输出： [2,1,1,0]

解释：

5 的右侧有 2 个更小的元素 (2 和 1)

2 的右侧仅有 1 个更小的元素 (1)

6 的右侧有 1 个更小的元素 (1)

1 的右侧有 0 个更小的元素

示例 2：

输入： nums = [-1]

输出： [0]

示例 3：

输入： nums = [-1,-1]

输出： [0,0]

提示：

1 <= nums.length <= 10⁵

-10⁴ <= nums[i] <= 10⁴

**13.去除重复字母**

给你一个字符串 s，请你去除字符串中重复的字母，使得每个字母只出现一次。需保证返回结果的字典序最小（要求不能打乱其他字符的相对位置）。

示例 1：

输入： s = "bcabc"

输出： "abc"

示例 2：

输入： s = "cbacdcbc"

输出： "acdb"

提示：

1 <= s.length <= 10^4

s 由小写英文字母组成

**14.拼接最大数**

给你两个整数数组 nums1 和 nums2，它们的长度分别为 m 和 n。数组 nums1 和 nums2 分别代表两个数个位上的数字。同时你也会得到一个整数 k。

请你利用这两个数组中的数字中创建一个长度为 k <= m + n 的最大数，在这个必须保留来自同一数组的数字的相对顺序。

返回代表答案的长度为 k 的数组。

示例 1：

输入：nums1 = [3,4,6,5], nums2 = [9,1,2,5,8,3], k = 5

输出：[9,8,6,5,3]

示例 2：

输入：nums1 = [6,7], nums2 = [6,0,4], k = 5

输出：[6,7,6,0,4]

示例 3：

输入：nums1 = [3,9], nums2 = [8,9], k = 3

输出：[9,8,9]

提示：

m == nums1.length

n == nums2.length

1 <= m, n <= 500

0 <= nums1[i], nums2[i] <= 9

1 <= k <= m + n

**15.零钱兑换**

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

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

你可以认为每种硬币的数量是无限的。

示例 1：

输入： coins = [1, 2, 5], amount = 11

输出： 3

解释： 11 = 5 + 5 + 1

示例 2：

输入： coins = [2], amount = 3

输出： -1

示例 3：

输入： coins = [1], amount = 0

输出： 0

提示：

1 <= coins.length <= 12

1 <= coins[i] <= 2^31 - 1

0 <= amount <= 10^4

**16.和等于k的最长子数组长度**

给定一个数组 nums 和一个目标值 k，找到和等于 k 的最长连续子数组长度。如果不存在任意一个符合要求的子数组，则返回 0。

示例 1:

输入: nums = [1,-1,5,-2,3], k = 3

输出: 4

解释: 子数组 [1, -1, 5, -2] 和等于 3，且长度最长。

示例 2:

输入: nums = [-2,-1,2,1], k = 1

输出: 2

解释: 子数组 [-1, 2] 和等于 1，且长度最长。

提示：

1 <= nums.length <= 2 \* 10^5

-10^4 <= nums[i] <= 10^4

-10^9 <= k <= 10^9

**17.区间和的个数**

给你一个整数数组 nums 以及两个整数 lower 和 upper 。求数组中，值位于范围 [lower, upper] （包含 lower 和 upper）之内的 区间和的个数 。

区间和 S(i, j) 表示在 nums 中，位置从 i 到 j 的元素之和，包含 i 和 j (i ≤ j)。

示例 1：

输入： nums = [-2,5,-1], lower = -2, upper = 2

输出： 3

解释： 存在三个区间：[0,0]、[2,2] 和 [0,2] ，对应的区间和分别是：-2 、-1 、2 。

示例 2：

输入： nums = [0], lower = 0, upper = 0

输出： 1

提示：

1 <= nums.length <= 10^5

-2^31 <= nums[i] <= 2^31 - 1

-10^5 <= lower <= upper <= 10^5

题目数据保证答案是一个 32 位 的整数

**18.按要求补齐数组**

给定一个已排序的正整数数组 nums，和一个正整数 n。从 [1, n] 区间内选取任意个数字补充到 nums 中，使得 [1, n] 区间内的任何数字都可以用 nums 中某几个数字的和来表示。

请返回满足上述要求的最少需要补充的数字个数。

示例 1:

输入: nums = [1,3], n = 6

输出: 1

解释:

根据 nums 里现有的组合 [1], [3], [1,3]，可以得出 1, 3, 4。

现在如果我们将 2 添加到 nums 中，组合变为: [1], [2], [3], [1,3], [2,3], [1,2,3]。

其和可以表示数字 1, 2, 3, 4, 5, 6，能够覆盖 [1, 6] 区间里所有的数。

所以我们最少需要添加一个数字。

示例 2:

输入: nums = [1,5,10], n = 20

输出: 2

解释: 我们需要添加 [2,4]。

示例 3:

输入: nums = [1,2,2], n = 5

输出: 0

提示：

1 <= nums.length <= 1000

1 <= nums[i] <= 10^4

nums 按升序排列

1 <= n <= 2^31 - 1

**19.验证二叉树的前序序列化**

序列化二叉树的一种方法是使用 前序遍历 。当我们遇到一个非空节点时，我们可以记录下这个节点的值。如果它是一个空节点，我们可以使用一个标记值记录，例如 #。



例如，上面的二叉树可以被序列化为字符串 "9,3,4,#,#,1,#,#,2,#,6,#,#"，其中 # 代表一个空节点。

给定一串以逗号分隔的序列，验证它是否是正确的二叉树的前序序列化。编写一个在不重构树的条件下的可行算法。

保证 每个以逗号分隔的字符或为一个整数或为一个表示 null 指针的 '#' 。

你可以认为输入格式总是有效的

例如它永远不会包含两个连续的逗号，比如 "1,,3" 。

注意：不允许重建树。

示例 1:

输入: preorder = "9,3,4,#,#,1,#,#,2,#,6,#,#"

输出: true

示例 2:

输入: preorder = "1,#"

输出: false

示例 3:

输入: preorder = "9,#,#,1"

输出: false

提示:

1 <= preorder.length <= 104

preorder 由以逗号 “，” 分隔的 [0,100] 范围内的整数和 “#” 组成

**20.重新安排行程**

给你一份航线列表 tickets ，其中 tickets[i] = [fromi, toi] 表示飞机出发和降落的机场地点。请你对该行程进行重新规划排序。

所有这些机票都属于一个从 JFK（肯尼迪国际机场）出发的先生，所以该行程必须从 JFK 开始。如果存在多种有效的行程，请你按字典排序返回最小的行程组合。

例如，行程 ["JFK", "LGA"] 与 ["JFK", "LGB"] 相比就更小，排序更靠前。

假定所有机票至少存在一种合理的行程。且所有的机票 必须都用一次 且 只能用一次。

示例 1：



输入：

tickets =

[["MUC","LHR"],["JFK","MUC"],["SFO","SJC"],["LHR","SFO"]]

输出：

["JFK","MUC","LHR","SFO","SJC"]

示例 2：



输入：

tickets =

[["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK"],["ATL","SFO"]]

输出：

["JFK","ATL","JFK","SFO","ATL","SFO"]

解释：另一种有效的行程是 ["JFK","SFO","ATL","JFK","ATL","SFO"] ，但是它字典排序更大更靠后。

提示：

1 <= tickets.length <= 300

tickets[i].length == 2

fromi.length == 3

toi.length == 3

fromi 和 toi 由大写英文字母组成

fromi != toi

**21.递增的三元子序列**

给你一个整数数组 nums ，判断这个数组中是否存在长度为 3 的递增子序列。

如果存在这样的三元组下标 (i, j, k) 且满足 i < j < k ，使得 nums[i] < nums[j] < nums[k] ，返回 true ；否则，返回 false 。

示例 1：

输入： nums = [1,2,3,4,5]

输出： true

解释： 任何 i < j < k 的三元组都满足题意

示例 2：

输入： nums = [5,4,3,2,1]

输出： false

解释： 不存在满足题意的三元组

示例 3：

输入： nums = [2,1,5,0,4,6]

输出： true

解释： 三元组 (3, 4, 5) 满足题意，因为 nums[3] == 0 < nums[4] == 4 < nums[5] == 6

提示：

1 <= nums.length <= 5 \* 10^5

-2^31 <= nums[i] <= 2^31 - 1

**22.回文对**

给定一个由唯一字符串构成的 0 索引 数组 words 。

回文对 是一对整数 (i, j) ，满足以下条件：

0 <= i, j < words.length，

i != j ，并且

words[i] + words[j]（两个字符串的连接）是一个回文串。

返回一个数组，它包含 words 中所有满足 回文对 条件的字符串。

你必须设计一个时间复杂度为 O(sum of words[i].length) 的算法。

示例 1：

输入： words = ["abcd","dcba","lls","s","sssll"]

输出： [[0,1],[1,0],[3,2],[2,4]]

解释：

可拼接成的回文串为 ["dcbaabcd","abcddcba","slls","llssssll"]

示例 2：

输入： words = ["bat","tab","cat"]

输出： [[0,1],[1,0]]

解释： 可拼接成的回文串为 ["battab","tabbat"]

示例 3：

输入： words = ["a",""]

输出： [[0,1],[1,0]]

提示：

1 <= words.length <= 5000

0 <= words[i].length <= 300

words[i] 由小写英文字母组成

**23.至多包含K个不同字符的最长子串**

给你一个字符串 s 和一个整数 k ，请你找出 至多 包含 k 个 不同 字符的最长子串，并返回该子串的长度。

示例 1：

输入： s = "eceba", k = 2

输出： 3

解释： 满足题目要求的子串是 "ece" ，长度为 3 。

示例 2：

输入： s = "aa", k = 1

输出： 2

解释： 满足题目要求的子串是 "aa" ，长度为 2 。

提示：

1 <= s.length <= 5 \* 10^4

0 <= k <= 50

**24.俄罗斯套娃信封问题**

给你一个二维整数数组 envelopes ，其中 envelopes[i] = [wᵢ, hᵢ] ，表示第 i 个信封的宽度和高度。

当另一个信封的宽度和高度都比这个信封大的时候，这个信封就可以放进另一个信封里，如同俄罗斯套娃一样。

请计算 最多能有多少个 信封能组成一组“俄罗斯套娃”信封（即可以把一个信封放到另一个信封里面）。

注意：不允许旋转信封。

示例 1：

输入： envelopes = [[5,4],[6,4],[6,7],[2,3]]

输出： 3

解释： 最多信封的个数为 3, 组合为: [2,3] => [5,4] => [6,7]。

示例 2：

输入： envelopes = [[1,1],[1,1],[1,1]]

输出： 1

提示：

1 <= envelopes.length <= 10⁵

envelopes[i].length == 2

1 <= wᵢ, hᵢ <= 10⁵

**25.K距离间隔重排字符串**

给你一个非空的字符串 s 和一个整数 k，你要将这个字符串 s 中的字母进行重新排列，使得重排后的字符串中相同字母的位置间隔距离 至少 为 k。如果无法做到，请返回一个空字符串 ""。

示例 1：

输入: s = "aabbcc", k = 3

输出: "abcabc"

解释: 相同的字母在新的字符串中间隔至少 3 个单位距离。

示例 2:

输入: s = "aaabc", k = 3

输出: ""

解释: 没有办法找到可能的重排结果。

示例 3:

输入: s = "aaadbbcc", k = 2

输出: "abacabcd"

解释: 相同的字母在新的字符串中间隔至少 2 个单位距离。

提示：

1 <= s.length <= 3 \* 10^5

s 仅由小写英文字母组成

0 <= k <= s.length

**26.矩形区域不超过K的最大数值和**

给你一个 m x n 的矩阵 matrix 和一个整数 k ，找出并返回矩阵内部矩形区域的不超过 k 的最大数值和。

题目数据保证总会存在一个数值和不超过 k 的矩形区域。

示例 1：



输入：matrix = [[1,0,1],[0,-2,3]], k = 2

输出：2

解释：蓝色边框圈出来的矩形区域 [[0, 1], [-2, 3]] 的数值和是 2，且 2 是不超过 k 的最大数字（k = 2）。

示例 2：

输入：matrix = [[2,2,-1]], k = 3

输出：3

提示：

m == matrix.length

n == matrix[i].length

1 <= m, n <= 100

-100 <= matrix[i][j] <= 100

-105 <= k <= 105

**27.水壶问题**

有两个水壶，容量分别为 x 和 y 升。水的供应是无限的。确定是否有可能使用这两个壶准确得到 target 升。

你可以：

装满任意一个水壶

清空任意一个水壶

将水从一个水壶倒入另一个水壶，直到接水壶已满，或倒水壶已空。

示例 1:

输入: x = 3, y = 5, target = 4

输出: true

解释：

按照以下步骤操作，以达到总共 4 升水：

装满 5 升的水壶(0, 5)。

把 5 升的水壶倒进 3 升的水壶，留下 2 升(3, 2)。

倒空 3 升的水壶(0, 2)。

把 2 升水从 5 升的水壶转移到 3 升的水壶(2, 0)。

再次加满 5 升的水壶(2, 5)。

从 5 升的水壶向 3 升的水壶倒水直到 3 升的水壶倒满。5 升的水壶里留下了 4 升水(3, 4)。

倒空 3 升的水壶。现在，5 升的水壶里正好有 4 升水(0, 4)。

参考：来自著名的 "Die Hard"

示例 2:

输入: x = 2, y = 6, target = 5

输出: false

示例 3:

输入: x = 1, y = 2, target = 3

输出: true

解释：同时倒满两个水壶。现在两个水壶中水的总量等于 3。

提示:

1 <= x, y, target <= 10^3

**28.最大整除子集**

给你一个由 无重复 正整数组成的集合 nums ，请你找出并返回其中最大的整除子集 answer ，子集中每一元素对 (answer[i], answer[j]) 都应当满足：

answer[i] % answer[j] == 0 ，或

answer[j] % answer[i] == 0

如果存在多个有效解子集，返回其中任何一个均可。

示例 1：

输入： nums = [1,2,3]

输出： [1,2]

解释： [1,3] 也会被视为正确答案。

示例 2：

输入： nums = [1,2,4,8]

输出： [1,2,4,8]

提示：

1 <= nums.length <= 1000

1 <= nums[i] <= 2 \* 10^9

nums 中的所有整数 互不相同

**29.查找和最小的k对数字**

给定两个以 非递减顺序排列 的整数数组 nums1 和 nums2，以及一个整数 k。

定义一对值 (u,v)，其中第一个元素来自 nums1，第二个元素来自 nums2。

请找到和最小的 k 个数对 (u₁,v₁), (u₂,v₂) ... (uₖ,vₖ)。

示例 1:

输入: nums1 = [1,7,11], nums2 = [2,4,6], k = 3

输出: [1,2],[1,4],[1,6]

解释: 返回序列中的前 3 对数：

[1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]

示例 2:

输入: nums1 = [1,1,2], nums2 = [1,2,3], k = 2

输出: [1,1],[1,1]

解释: 返回序列中的前 2 对数：

[1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]

提示:

1 <= nums1.length, nums2.length <= 10⁵

-10⁹ <= nums1[i], nums2[i] <= 10⁹

nums1 和 nums2 均为 升序排列

1 <= k <= 10⁴

k <= nums1.length \* nums2.length

**30.猜数字大小lI**

我们正在玩一个猜数游戏，游戏规则如下：

我从 1 到 n 之间选择一个数字。

你来猜我选了哪个数字。

如果你猜到正确的数字，就会 赢得游戏 。

如果你猜错了，那么我会告诉你，我选的数字比你的 更大或者更小 ，并且你需要继续猜数。

每当你猜了数字 x 并且猜错了的时候，你需要支付金额为 x 的现金。如果你花光了钱，就会 输掉游戏 。

给你一个特定的数字 n ，返回能够 确保你获胜 的最小现金数，不管我选择那个数字 。

示例 1：



输入：n = 10

输出：16

解释：制胜策略如下：

- 数字范围是 [1,10] 。你先猜测数字为 7 。

 - 如果这是我选中的数字，你的总费用为 $0 。否则，你需要支付 $7 。

 - 如果我的数字更大，则下一步需要猜测的数字范围是 [8,10] 。你可以猜测数字为 9 。

 - 如果这是我选中的数字，你的总费用为 $7 。否则，你需要支付 $9 。

 - 如果我的数字更大，那么这个数字一定是 10 。你猜测数字为 10 并赢得游戏，总费用为 $7 + $9 = $16 。

 - 如果我的数字更小，那么这个数字一定是 8 。你猜测数字为 8 并赢得游戏，总费用为 $7 + $9 = $16 。

 - 如果我的数字更小，则下一步需要猜测的数字范围是 [1,6] 。你可以猜测数字为 3 。

 - 如果这是我选中的数字，你的总费用为 $7 。否则，你需要支付 $3 。

 - 如果我的数字更大，则下一步需要猜测的数字范围是 [4,6] 。你可以猜测数字为 5 。

 - 如果这是我选中的数字，你的总费用为 $7 + $3 = $10 。否则，你需要支付 $5 。

 - 如果我的数字更大，那么这个数字一定是 6 。你猜测数字为 6 并赢得游戏，总费用为 $7 + $3 + $5 = $15 。

 - 如果我的数字更小，那么这个数字一定是 4 。你猜测数字为 4 并赢得游戏，总费用为 $7 + $3 + $5 = $15 。

 - 如果我的数字更小，则下一步需要猜测的数字范围是 [1,2] 。你可以猜测数字为 1 。

 - 如果这是我选中的数字，你的总费用为 $7 + $3 = $10 。否则，你需要支付 $1 。

 - 如果我的数字更大，那么这个数字一定是 2 。你猜测数字为 2 并赢得游戏，总费用为 $7 + $3 + $1 = $11 。

在最糟糕的情况下，你需要支付 $16 。因此，你只需要 $16 就可以确保自己赢得游戏。

示例 2：

输入：n = 1

输出：0

解释：只有一个可能的数字，所以你可以直接猜 1 并赢得游戏，无需支付任何费用。

示例 3：

输入：n = 2

输出：1

解释：有两个可能的数字 1 和 2 。

- 你可以先猜 1 。

 - 如果这是我选中的数字，你的总费用为 $0 。否则，你需要支付 $1 。

 - 如果我的数字更大，那么这个数字一定是 2 。你猜测数字为 2 并赢得游戏，总费用为 $1 。

最糟糕的情况下，你需要支付 $1 。

提示：

1 <= n <= 200

**31.摆动序列**

如果连续数字之间的差严格地在正数和负数之间交替，则数字序列称为 摆动序列。第一个差（如果存在的话）可能是正数或负数。仅有一个元素或者含两个不等元素的序列也视作摆动序列。

例如，[1, 7, 4, 9, 2, 5] 是一个 摆动序列，因为差值 (6, -3, 5, -7, 3) 是正负交替出现的。

相反，[1, 4, 7, 2, 5] 和 [1, 7, 4, 5, 5] 不是摆动序列，第一个序列是因为它的前两个差值都是正数，第二个序列是因为它的最后一个差值为零。

子序列 可以通过从原始序列中删除一些（也可以不删除）元素来获得，剩下的元素保持其原始顺序。

给你一个整数数组 nums，返回 nums 中作为 摆动序列 的 最长子序列的长度。

示例 1：

输入： nums = [1,7,4,9,2,5]

输出： 6

解释： 整个序列均为摆动序列，各元素之间的差值为 (6, -3, 5, -7, 3)。

示例 2：

输入： nums = [1,17,5,10,13,15,10,5,16,8]

输出： 7

解释： 这个序列包含几个长度为 7 的摆动序列。

其中一个是 [1, 17, 10, 13, 10, 16, 8]，各元素之间的差值为 (16, -7, 3, -3, 6, -8)。

示例 3：

输入： nums = [1,2,3,4,5,6,7,8,9]

输出： 2

提示：

1 <= nums.length <= 1000

0 <= nums[i] <= 1000

**32.组合总和.Ⅳ**

给你一个由 不同 整数组成的数组 nums ，和一个目标整数 target 。请你从 nums 中找出并返回总和为 target 的元素组合的个数。

题目数据保证答案符合 32 位整数范围。

示例 1：

输入：nums = [1,2,3], target = 4

输出：7

解释：

所有可能的组合为：

(1, 1, 1, 1)

(1, 1, 2)

(1, 2, 1)

(1, 3)

(2, 1, 1)

(2, 2)

(3, 1)

请注意，顺序不同的序列被视作不同的组合。

示例 2：

输入：nums = [9], target = 3

输出：0

提示：

1 <= nums.length <= 200

1 <= nums[i] <= 1000

nums 中的所有元素 互不相同

1 <= target <= 1000

**33.有序矩阵中第k小的元素**

给你一个 n x n 矩阵 matrix，其中每行和每列元素均按升序排序，找到矩阵中第 k 小的元素。请注意，它是排序后的第 k 小元素，而不是第 k 个不同的元素。

你必须找到一个内存复杂度优于 O(n²) 的解决方案。

示例 1：

输入：matrix = [[1,5,9],[10,11,13],[12,13,15]], k = 8

输出：13

解释：矩阵中的元素为 [1,5,9,10,11,12,13,13,15]，第 8 小元素是 13

示例 2：

输入：matrix = [[-5]], k = 1

输出：-5

提示：

n == matrix.length

n == matrix[i].length

1 <= n <= 300

-10⁹ <= matrix[i][j] <= 10⁹

题目数据保证 matrix 中的所有行和列都按非递减顺序排列

1 <= k <= n²

**34.赎金信**

给你两个字符串：ransomNote 和 magazine ，判断 ransomNote 能不能由 magazine 里面的字符构成。

如果可以，返回 true ；否则返回 false 。

magazine 中的每个字符只能在 ransomNote 中使用一次。

示例 1：

输入：ransomNote = "a", magazine = "b"

输出：false

示例 2：

输入：ransomNote = "aa", magazine = "ab"

输出：false

示例 3：

输入：ransomNote = "aa", magazine = "aab"

输出：true

提示：

1 ≤ ransomNote.length, magazine.length ≤ 10^5

ransomNote 和 magazine 由小写英文字母组成

**35.字典序排数**

给你一个整数 n ，按字典序返回范围 [1, n] 内所有整数。

你必须设计一个时间复杂度为 O(n) 且使用 O(1) 额外空间的算法。

示例 1：

输入：n = 13

输出：[1,10,11,12,13,2,3,4,5,6,7,8,9]

示例 2：

输入：n = 2

输出：[1,2]

提示：

1 <= n <= 5 \* 10^4

**36.字符串中的第一个唯一字符**

给定一个字符串 s，找到它的第一个不重复的字符，并返回它的索引。如果不存在，则返回 -1。

示例 1：

输入: s = "leetcode"

输出: 0

示例 2：

输入: s = "loveleetcode"

输出: 2

示例 3：

输入: s = "aabb"

输出: -1

提示：

1 <= s.length <= 10^5

s 只包含小写字母

**37.找不同**

给定两个字符串 s 和 t，它们只包含小写字母。

字符串 t 由字符串 s 随机重排，然后在随机位置添加一个字母。

请找出在 t 中被添加的字母。

示例 1：

输入：s = "abcd", t = "abcde"

输出："e"

解释：'e' 是那个被添加的字母。

示例 2：

输入：s = "", t = "y"

输出："y"

提示：

0 <= s.length <= 1000

t.length == s.length + 1

s 和 t 只包含小写字母

**38.判断子序列**

给定字符串 s 和 t ，判断 s 是否为 t 的子序列。

字符串的一个子序列是原始字符串删除一些（也可以不删除）字符而不改变剩余字符相对位置形成的新字符串。（例如，"ace"是"abcde"的一个子序列，而"aec"不是）。

进阶：

如果有大量输入的 S，称作 S1, S2, ... , Sk 其中 k >= 10亿，你需要依次检查它们是否为 T 的子序列。在这种情况下，你会怎样改变代码？

示例 1：

输入：s = "abc", t = "ahbgdc"

输出：true

示例 2：

输入：s = "axc", t = "ahbgdc"

输出：false

提示：

0 <= s.length <= 100

0 <= t.length <= 10^4

两个字符串都只由小写字符组成。

**39.UTF-8编码验证**

给定一个表示数据的整数数组 data ，返回它是否为有效的 UTF-8 编码。

UTF-8 中的一个字符可能的长度为 1 到 4 字节，遵循以下的规则：

对于 1 字节 的字符，字节的第一位设为 0 ，后面 7 位为这个符号的 unicode 码。

对于 n 字节 的字符 (n > 1)，第一个字节的前 n 位都设为1，第 n+1 位设为 0 ，后面字节的前两位一律设为 10 。剩下的没有提及的二进制位，全部为这个符号的 unicode 码。

这是 UTF-8 编码的工作方式：

Number of Bytes | UTF-8 octet sequence (binary)

--------------------+---------------------------------------------

1 | 0xxxxxxx

2 | 110xxxxx 10xxxxxx

3 | 1110xxxx 10xxxxxx 10xxxxxx

4 | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

x 表示二进制形式的一位，可以是 0 或 1。

注意：输入是整数数组。只有每个整数的 最低 8 个有效位 用来存储数据。这意味着每个整数只表示 1 字节的数据。

示例 1：

输入：data = [197,130,1]

输出：true

解释：数据表示字节序列: 11000101 10000010 00000001。

这是有效的 utf-8 编码，为一个 2 字节字符，跟着一个 1 字节字符。

示例 2：

输入：data = [235,140,4]

输出：false

解释：数据表示 8 位的序列: 11101011 10001100 00000100。

前 3 位都是 1 ，第 4 位为 0 表示它是一个 3 字节字符。

下一个字节是开头为 10 的延续字节，这是正确的。

但第二个延续字节不以 10 开头，所以是不符合规则的。

提示：

1 <= data.length <= 2 \* 10^4

0 <= data[i] <= 255

**40.字符串解码**

给定一个经过编码的字符串，返回它解码后的字符串。

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

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

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

示例 1：

输入：s = "3[a]2[bc]"

输出："aaabcbc"

示例 2：

输入：s = "3[a2[c]]"

输出："accaccacc"

示例 3：

输入：s = "2[abc]3[cd]ef"

输出："abcabccdcdcdef"

示例 4：

输入：s = "abc3[cd]xyz"

输出："abccdcdcdxyz"

提示：

1 <= s.length <= 30

s 由小写英文字母、数字和方括号组成

s 保证是一个 有效 的输入。

s 中所有整数的取值范围为 [1, 300]

**41.旋转函数**

给定一个长度为 n 的整数数组 nums 。

假设 arrk 是数组 nums 顺时针旋转 k 个位置后的数组，我们定义 nums 的 旋转函数 F 为：

F(k) = 0 \* arrk[0] + 1 \* arrk[1] + ... + (n - 1) \* arrk[n - 1]

返回 F(0), F(1), ..., F(n-1)中的最大值 。

生成的测试用例让答案符合 32 位 整数。

示例 1:

输入: nums = [4,3,2,6]

输出: 26

解释:

F(0) = (0 \* 4) + (1 \* 3) + (2 \* 2) + (3 \* 6) = 0 + 3 + 4 + 18 = 25

F(1) = (0 \* 6) + (1 \* 4) + (2 \* 3) + (3 \* 2) = 0 + 4 + 6 + 6 = 16

F(2) = (0 \* 2) + (1 \* 6) + (2 \* 4) + (3 \* 3) = 0 + 6 + 8 + 9 = 23

F(3) = (0 \* 3) + (1 \* 2) + (2 \* 6) + (3 \* 4) = 0 + 2 + 12 + 12 = 26

所以 F(0), F(1), F(2), F(3) 中的最大值是 F(3) = 26 。

示例 2:

输入: nums = [100]

输出: 0

提示:

n == nums.length

1 ≤ n ≤ 105

-100 ≤ nums[i] ≤ 100

**42.整数替换**

给定一个正整数 n，你可以做如下操作：

如果 n 是偶数，则用 n / 2 替换 n。

如果 n 是奇数，则可以用 n + 1 或 n - 1 替换 n。

返回 n 变为 1 所需的最小替换次数。

示例 1：

输入：n = 8

输出：3

解释：8 -> 4 -> 2 -> 1

示例 2：

输入：n = 7

输出：4

解释：7 -> 8 -> 4 -> 2 -> 1

或 7 -> 6 -> 3 -> 2 -> 1

示例 3：

输入：n = 4

输出：2

提示：

1 <= n <= 2^31 - 1

**43.第N位数字**

给你一个整数 n，请你在无限的整数序列 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...] 中找出并返回第 n 位上的数字。

示例 1：

输入：n = 3

输出：3

示例 2：

输入：n = 11

输出：0

解释：第 11 位数字在序列 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ... 里是 0，它是 10 的一部分。

提示：

1 <= n <= 2^31 - 1

**44.二进制手表**

二进制手表顶部有 4 个 LED 代表 小时（0-11），底部的 6 个 LED 代表 分钟（0-59）。每个 LED 代表一个 0 或 1，最低位在右侧。

例如，下面的二进制手表读取 "4:51" 。



给你一个整数 turnedOn ，表示当前亮着的 LED 的数量，返回二进制手表可以表示的所有可能时间。你可以 按任意顺序 返回答案。

小时不会以零开头：

例如，"01:00" 是无效的时间，正确的写法应该是 "1:00" 。

分钟必须由两位数组成，可能会以零开头：

例如，"10:2" 是无效的时间，正确的写法应该是 "10:02" 。

示例 1：

输入：turnedOn = 1

输出：["0:01","0:02","0:04","0:08","0:16","0:32","1:00","2:00","4:00","8:00"]

示例 2：

输入：turnedOn = 9

输出：[]

提示：

0 <= turnedOn <= 10

**45.移掉k位数字**

给你一个以字符串表示的非负整数 num 和一个整数 k ，移除这个数中的 k 位数字，使得剩下的数字最小。请你以字符串形式返回这个最小的数字。

示例 1 ：

输入：num = "1432219", k = 3

输出："1219"

解释：移除掉三个数字 4, 3, 和 2 形成一个新的最小的数字 1219 。

示例 2 ：

输入：num = "10200", k = 1

输出："200"

解释：移掉首位的 1 剩下的数字为 200. 注意输出不能有任何前导零。

示例 3 ：

输入：num = "10", k = 2

输出："0"

解释：从原数字移除所有的数字，剩余为空就是 0 。

提示：

1 <= k <= num.length <= 10^5

num 仅由若干位数字（0 - 9）组成

除了 0 本身之外，num 不含任何前导零

**46.青蛙过河**

一只青蛙想要过河。假定河流被等分为若干个单元格，并且在每一个单元格内都有可能放有一块石子（也有可能没有）。青蛙可以跳上石子，但是不可以跳入水中。

给你石子的位置列表 stones（用单元格序号升序表示），请判定青蛙能否成功过河（即能否在最后一步跳至最后一块石子上）。开始时，青蛙默认已站在第一块石子上，并可以假定它第一步只能跳跃 1 个单位（即只能从单元格 1 跳至单元格 2）。

如果青蛙上一步跳跃了 k 个单位，那么它接下来的跳跃距离只能选择为 k - 1、k 或 k + 1 个单位。另请注意，青蛙只能向前方（终点的方向）跳跃。

示例 1：

输入：stones = [0,1,3,5,6,8,12,17]

输出：true

解释：青蛙可以成功过河，按照如下方案跳跃：跳 1 个单位到第 2 块石子，然后跳 2 个单位到第 3 块石子，接着跳 2 个单位到第 4 块石子，然后跳 3 个单位到第 6 块石子，跳 4 个单位到第 7 块石子，最后，跳 5 个单位到第 8 个石子（即最后一块石子）。

示例 2：

输入：stones = [0,1,2,3,4,8,9,11]

输出：false

解释：这是因为第 5 和第 6 个石子之间的间距太大，没有可选的方案供青蛙跳跃过去。

提示：

2 <= stones.length <= 2000

0 <= stones[i] <= 2^31 - 1

stones[0] == 0

stones 按严格升序排列

**47.数字转换为十六进制数**

给定一个整数，编写一个算法将这个数转换为十六进制数。对于负整数，我们通常使用补码运算方法。

注意：

十六进制中所有字母(a-f)都必须是小写。

十六进制字符串中不能包含多余的前导零。如果要转化的数为0，那么以单个字符'0'来表示；对于其他情况，十六进制字符串中的第一个字符将不会是0字符。

给定的数确保在32位有符号整数范围内。

不能使用任何由库提供的将数字直接转换或格式化为十六进制的方法。

示例 1：

输入:

26

输出:

"1a"

示例 2：

输入:

-1

输出:

"ffffffff"

**48.根据身高重建队列**

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

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

示例 1：

输入：people = [[7,0],[4,4],[7,1],[5,0],[6,1],[5,2]]

输出：[[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]]

解释：

编号为 0 的人身高为 5 ，没有身高更高或者相同的人排在他前面。

编号为 1 的人身高为 7 ，没有身高更高或者相同的人排在他前面。

编号为 2 的人身高为 5 ，有 2 个身高更高或者相同的人排在他前面，即编号为 0 和 1 的人。

编号为 3 的人身高为 6 ，有 1 个身高更高或者相同的人排在他前面，即编号为 1 的人。

编号为 4 的人身高为 4 ，有 4 个身高更高或者相同的人排在他前面，即编号为 0、1、2、3 的人。

编号为 5 的人身高为 7 ，有 1 个身高更高或者相同的人排在他前面，即编号为 1 的人。

因此 [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]] 是重新构造后的队列。

示例 2：

输入：people = [[6,0],[5,0],[4,0],[3,2],[2,2],[1,4]]

输出：[[4,0],[5,0],[2,2],[3,2],[1,4],[6,0]]

提示：

1 <= people.length <= 2000

0 <= hi <= 106

0 <= ki < people.length

题目数据确保队列可以被重建

**49.接雨水lI**

给你一个 m x n 的矩阵，其中的值均为非负整数，代表二维高度图每个单元的高度，请计算图中形状最多能接多少体积的雨水。

示例 1:



输入: heightMap = [[1,4,3,1,3,2],[3,2,1,3,2,4],[2,3,3,2,3,1]]

输出: 4

解释: 下雨后，雨水将会被上图蓝色的方块中。总的接雨水量为1+2+1=4。

示例 2:



输入:

heightMap =

[[3,3,3,3,3],[3,2,2,2,3],[3,2,1,2,3],[3,2,2,2,3],[3,3,3,3,3]]

输出: 10

提示:

m == heightMap.length

n == heightMap[i].length

1 <= m, n <= 200

0 <= heightMap[i][j] <= 2 \* 104

**50.最长回文串**

给定一个包含大写字母和小写字母的字符串 s，返回通过这些字母构造成的最长的回文串。

在构造过程中，请注意区分大小写。比如 "Aa" 不能当作一个回文字符串。

示例 1:

输入: s = "abccccdd"

输出: 7

解释:

我们可以构造的最长的回文串是"dccaccd", 它的长度是 7。

示例 2:

输入: s = "a"

输出: 1

示例 3：

输入: s = "aaaaaccc"

输出: 7

提示:

1 <= s.length <= 2000

s 只由小写和/或大写英文字母组成

**四、模块D：Python进阶开发**

**1.不同的平均值数目**

给你一个下标从 0 开始长度为 偶数 的整数数组 nums 。

只要 nums 不是 空数组，你就重复执行以下步骤：

找到 nums 中的最小值，并删除它。

找到 nums 中的最大值，并删除它。

计算删除两数的平均值。

两数 a 和 b 的 平均值 为 (a + b) / 2 。

比方说，2 和 3 的平均值是 (2 + 3) / 2 = 2.5 。

返回上述过程能得到的 不同 平均值的数目。

注意 ，如果最小值或者最大值有重复元素，可以删除任意一个。

示例 1：

输入：nums = [4,1,4,0,3,5]

输出：2

解释：

1. 删除 0 和 5 ，平均值是 (0 + 5) / 2 = 2.5 ，现在 nums = [4,1,4,3] 。

2. 删除 1 和 4 ，平均值是 (1 + 4) / 2 = 2.5 ，现在 nums = [4,3] 。

3. 删除 3 和 4 ，平均值是 (3 + 4) / 2 = 3.5 。

2.5 ，2.5 和 3.5 之中总共有 2 个不同的数，我们返回 2 。

示例 2：

输入：nums = [1,100]

输出：1

解释：

删除 1 和 100 后只有一个平均值，所以我们返回 1 。

提示：

2 <= nums.length <= 100

nums.length 是偶数。

0 <= nums[i] <= 100。

**2.执行子串操作后的字典序最小字符串**

给你一个仅由小写英文字母组成的字符串 s 。在一步操作中，你可以完成以下行为：

选择 s 的任一非空子字符串，可能是整个字符串，接着将字符串中的每一个字符替换为英文字母表中的前一个字符。例如，'b' 用 'a' 替换，'a' 用 'z' 替换。

返回执行上述操作 恰好一次 后可以获得的 字典序最小 的字符串。

子字符串 是字符串中的一个连续字符序列。

现有长度相同的两个字符串 x 和 字符串 y ，在满足 x[i] != y[i] 的第一个位置 i 上，如果 x[i] 在字母表中先于 y[i] 出现，则认为字符串 x 比字符串 y 字典序更小 。

示例 1：

输入：s = "cbabc"

输出："baabc"

解释：我们选择从下标 0 开始、到下标 1 结束的子字符串执行操作。

可以证明最终得到的字符串是字典序最小的。

示例 2：

输入：s = "acbbc"

输出："abaab"

解释：我们选择从下标 1 开始、到下标 4 结束的子字符串执行操作。

可以证明最终得到的字符串是字典序最小的。

示例 3：

输入：s = "leetcode"

输出："kddsbncd"

解释：我们选择整个字符串执行操作。

可以证明最终得到的字符串是字典序最小的。

提示：

1 <= s.length <= 3 \* 105

s 仅由小写英文字母组成。