==========
文件夹: class014_HashSetHashMapTreeSetTreeMapComparator
==========

[Markdown 文件]
==========
文件: README.md
==========
Class026: HashSet、HashMap、TreeSet、TreeMap 和 Comparator 专题

本章节深入探讨了 Java 中 HashSet、HashMap、TreeSet、TreeMap 和 Comparator 的使用方法及相关的经典算法题目。

数据结构特点

HashSet
- 基于 HashMap 实现
- 底层使用哈希表
- 查询时间复杂度 O(1)
- 元素无序
- 不允许重复元素

HashMap
- 基于哈希表实现
- 查询时间复杂度 O(1)
- 键值对无序
- 允许一个 null 键和多个 null 值

TreeSet
- 基于 TreeMap 实现
- 底层使用红黑树
- 查询时间复杂度 O(log n)
- 元素有序
- 不允许重复元素

TreeMap
- 基于红黑树实现
- 查询时间复杂度 O(log n)
- 键值对按键有序

Comparator
- 比较器接口
- 用于定义对象之间比较的规则

- 可以实现自定义排序

题目列表

HashSet 和 HashMap 相关题目

1. **LeetCode 1. Two Sum (两数之和)**
   - 题目：在数组中找到两个数之和等于目标值的索引
   - 解法：使用 HashMap 存储已遍历元素及其索引
   - 时间复杂度：O(n)
   - 空间复杂度：O(n)
   - 网址：https://leetcode.com/problems/two-sum/

2. **LeetCode 242. Valid Anagram (有效的字母异位词)**
   - 题目：判断两个字符串是否为字母异位词
   - 解法：使用 HashMap 统计字符出现次数
   - 时间复杂度：O(n)
   - 空间复杂度：O(1)
   - 网址：https://leetcode.com/problems/valid-anagram/

3. **LeetCode 349. Intersection of Two Arrays (两个数组的交集)**
   - 题目：求两个数组的交集
   - 解法：使用 HashSet 存储元素并查找交集
   - 时间复杂度：O(m+n)
   - 空间复杂度：O(m)
   - 网址：https://leetcode.com/problems/intersection-of-two-arrays/

4. **LeetCode 705. Design HashSet (设计哈希集合)**
   - 题目：不使用内建哈希表库设计 HashSet
   - 解法：使用链地址法实现哈希表
   - 时间复杂度：O(n/b)
   - 空间复杂度：O(n)
   - 网址：https://leetcode.com/problems/design-hashset/

5. **LeetCode 706. Design HashMap (设计哈希映射)**
   - 题目：不使用内建哈希表库设计 HashMap
   - 解法：使用链地址法实现哈希表
   - 时间复杂度：O(n/b)
   - 空间复杂度：O(n)
   - 网址：https://leetcode.com/problems/design-hashmap/

6. **HackerRank Java Hashset (Java 哈希集)**
   - 题目：找出独特的字符串对数量

- 解法：使用 HashSet 存储字符串对
- 时间复杂度：O(n)
- 空间复杂度：O(n)
- 网址：https://www.hackerrank.com/challenges/java-hashset

7. **LeetCode 128. Longest Consecutive Sequence (最长连续序列)**
   - 题目：找出数字连续的最长序列长度
   - 解法：使用 HashSet 存储数字并查找连续序列
   - 时间复杂度：O(n)
   - 空间复杂度：O(n)
   - 网址：https://leetcode.com/problems/longest-consecutive-sequence/

8. **LeetCode 49. Group Anagrams (字母异位词分组)**
   - 题目：将字母异位词组合在一起
   - 解法：使用 HashMap 按排序后的字符串分组
   - 时间复杂度：O(N*K*logK)
   - 空间复杂度：O(N*K)
   - 网址：https://leetcode.com/problems/group-anagrams/

9. **LeetCode 347. Top K Frequent Elements (前 K 个高频元素)**
   - 题目：返回出现频率前 k 高的元素
   - 解法：使用 HashMap 统计频率+最小堆维护前 k 个元素
   - 时间复杂度：O(N*logK)
   - 空间复杂度：O(N)
   - 网址：https://leetcode.com/problems/top-k-frequent-elements/

10. **LeetCode 3. Longest Substring Without Repeating Characters (无重复字符的最长子串)**
    - 题目：找出不包含重复字符的最长子串长度
    - 解法：使用滑动窗口+HashSet 记录字符出现
    - 时间复杂度：O(n)
    - 空间复杂度：O(min(m,n))
    - 网址：https://leetcode.com/problems/longest-substring-without-repeating-characters/

11. **LeetCode 217. Contains Duplicate (存在重复元素)**
    - 题目：判断数组中是否存在重复元素
    - 解法：使用 HashSet 检查重复
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/contains-duplicate/

12. **LeetCode 219. Contains Duplicate II (存在重复元素 II)**
    - 题目：判断是否存在两个相同元素且下标差不超过 k
    - 解法：使用 HashMap 存储元素最近出现位置

- 时间复杂度：O(n)
- 空间复杂度：O(n)
- 网址：https://leetcode.com/problems/contains-duplicate-ii/

13. **LeetCode 290. Word Pattern (单词规律)**
    - 题目：判断字符串是否符合给定的模式
    - 解法：使用 HashMap 建立字符到单词的映射
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/word-pattern/

14. **LeetCode 205. Isomorphic Strings (同构字符串)**
    - 题目：判断两个字符串是否同构
    - 解法：使用 HashMap 建立字符映射关系
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/isomorphic-strings/

15. **LeetCode 383. Ransom Note (赎金信)**
    - 题目：判断 ransomNote 是否能由 magazine 中的字符组成
    - 解法：使用 HashMap 统计字符出现次数
    - 时间复杂度：O(m+n)
    - 空间复杂度：O(1)
    - 网址：https://leetcode.com/problems/ransom-note/

16. **LeetCode 387. First Unique Character in a String (字符串中的第一个唯一字符)**
    - 题目：找到字符串中第一个不重复的字符
    - 解法：使用 HashMap 统计字符出现次数
    - 时间复杂度：O(n)
    - 空间复杂度：O(1)
    - 网址：https://leetcode.com/problems/first-unique-character-in-a-string/

17. **LeetCode 454. 4Sum II (四数相加 II)**
    - 题目：计算四个数组中满足 a+b+c+d=0 的元组个数
    - 解法：使用 HashMap 存储两数之和的频率
    - 时间复杂度：$O(n^2)$
    - 空间复杂度：$O(n^2)$
    - 网址：https://leetcode.com/problems/4sum-ii/

18. **LeetCode 560. Subarray Sum Equals K (和为 K 的子数组)**
    - 题目：计算和为 k 的连续子数组个数
    - 解法：使用 HashMap 存储前缀和频率
    - 时间复杂度：O(n)

- 空间复杂度：O(n)
- 网址：https://leetcode.com/problems/subarray-sum-equals-k/

19. **LeetCode 575. Distribute Candies (分糖果)**
    - 题目：计算最多能获得多少种不同的糖果
    - 解法：使用 HashSet 统计糖果种类
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/distribute-candies/

20. **LeetCode 594. Longest Harmonious Subsequence (最长和谐子序列)**
    - 题目：找出最长的和谐子序列长度
    - 解法：使用 HashMap 统计数字频率
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/longest-harmonious-subsequence/

21. **LeetCode 599. Minimum Index Sum of Two Lists (两个列表的最小索引总和)**
    - 题目：找到两个列表中索引和最小的共同元素
    - 解法：使用 HashMap 存储第一个列表的索引
    - 时间复杂度：O(m+n)
    - 空间复杂度：O(m)
    - 网址：https://leetcode.com/problems/minimum-index-sum-of-two-lists/

22. **LeetCode 645. Set Mismatch (错误的集合)**
    - 题目：找出重复的数字和缺失的数字
    - 解法：使用 HashSet 检查重复和缺失
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/set-mismatch/

23. **LeetCode 692. Top K Frequent Words (前 K 个高频单词)**
    - 题目：返回出现频率前 k 高的单词
    - 解法：使用 HashMap 统计频率+自定义排序
    - 时间复杂度：O(n log k)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/top-k-frequent-words/

24. **LeetCode 771. Jewels and Stones (宝石与石头)**
    - 题目：计算石头中有多少是宝石
    - 解法：使用 HashSet 存储宝石类型
    - 时间复杂度：O(m+n)
    - 空间复杂度：O(m)

- 网址：https://leetcode.com/problems/jewels-and-stones/

25. **LeetCode 811. Subdomain Visit Count (子域名访问计数)**
    - 题目：统计子域名的访问次数
    - 解法：使用 HashMap 统计域名访问次数
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/subdomain-visit-count/

26. **LeetCode 884. Uncommon Words from Two Sentences (两句话中的不常见单词)**
    - 题目：找出在两句话中只出现一次的单词
    - 解法：使用 HashMap 统计单词出现次数
    - 时间复杂度：O(m+n)
    - 空间复杂度：O(m+n)
    - 网址：https://leetcode.com/problems/uncommon-words-from-two-sentences/

27. **LeetCode 890. Find and Replace Pattern (查找和替换模式)**
    - 题目：找到与给定模式匹配的所有单词
    - 解法：使用 HashMap 建立字符映射
    - 时间复杂度：O(n*k)
    - 空间复杂度：O(k)
    - 网址：https://leetcode.com/problems/find-and-replace-pattern/

28. **LeetCode 953. Verifying an Alien Dictionary (验证外星语词典)**
    - 题目：判断单词是否按外星语字典序排列
    - 解法：使用 HashMap 存储字母顺序
    - 时间复杂度：O(n*k)
    - 空间复杂度：O(1)
    - 网址：https://leetcode.com/problems/verifying-an-alien-dictionary/

29. **LeetCode 1002. Find Common Characters (查找常用字符)**
    - 题目：找到所有字符串中都出现的字符
    - 解法：使用 HashMap 统计字符最小出现次数
    - 时间复杂度：O(n*k)
    - 空间复杂度：O(1)
    - 网址：https://leetcode.com/problems/find-common-characters/

30. **LeetCode 1048. Longest String Chain (最长字符串链)**
    - 题目：找出最长的字符串链长度
    - 解法：使用 HashMap 存储字符串及其链长度
    - 时间复杂度：$O(n*l^2)$
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/longest-string-chain/

31. **LeetCode 1160. Find Words That Can Be Formed by Characters (拼写单词)**
    - 题目：计算能由给定字符组成的单词总长度
    - 解法：使用 HashMap 统计字符可用数量
    - 时间复杂度：O(n+m)
    - 空间复杂度：O(1)
    - 网址：https://leetcode.com/problems/find-words-that-can-be-formed-by-characters/

32. **LeetCode 1207. Unique Number of Occurrences (独一无二的出现次数)**
    - 题目：判断数组中每个数的出现次数是否都是唯一的
    - 解法：使用 HashMap 统计频率，再用 HashSet 检查唯一性
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/unique-number-of-occurrences/

33. **LeetCode 1396. Design Underground System (设计地铁系统)**
    - 题目：设计地铁系统的检票功能
    - 解法：使用 HashMap 存储乘客进站信息和站间时间统计
    - 时间复杂度：O(1)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/design-underground-system/

34. **LeetCode 1418. Display Table of Food Orders in a Restaurant (餐厅菜品展示表)**
    - 题目：生成餐厅菜品订单展示表
    - 解法：使用 HashMap 统计每桌的菜品数量
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/display-table-of-food-orders-in-a-restaurant/

35. **LeetCode 1481. Least Number of Unique Integers after K Removals (不同整数的最少数目)**
    - 题目：移除 k 个元素后，使不同整数的数量最少
    - 解法：使用 HashMap 统计频率，按频率排序后移除
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/least-number-of-unique-integers-after-k-removals/

36. **LeetCode 1512. Number of Good Pairs (好数对的数目)**
    - 题目：计算满足 nums[i] == nums[j] 且 i < j 的数对个数
    - 解法：使用 HashMap 统计相同数字的出现次数
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/number-of-good-pairs/

37. **LeetCode 1636. Sort Array by Increasing Frequency (按照频率将数组升序排序)**
    - 题目：按频率升序排序数组，频率相同按数值降序
    - 解法：使用 HashMap 统计频率+自定义排序
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/sort-array-by-increasing-frequency/

38. **LeetCode 1657. Determine if Two Strings Are Close (确定两个字符串是否接近)**
    - 题目：判断能否通过操作使两个字符串相等
    - 解法：使用 HashMap 统计字符频率和种类
    - 时间复杂度：O(n)
    - 空间复杂度：O(1)
    - 网址：https://leetcode.com/problems/determine-if-two-strings-are-close/

39. **LeetCode 1679. Max Number of K-Sum Pairs (K 和数对的最大数目)**
    - 题目：找出最多能组成多少对和为 k 的数对
    - 解法：使用 HashMap 统计数字出现次数
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/max-number-of-k-sum-pairs/

40. **LeetCode 1695. Maximum Erasure Value (删除子数组的最大得分)**
    - 题目：删除不含重复元素的子数组，使剩余元素和最大
    - 解法：使用滑动窗口+HashSet 检查重复
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/maximum-erasure-value/

41. **LeetCode 1748. Sum of Unique Elements (唯一元素的和)**
    - 题目：计算数组中只出现一次的元素之和
    - 解法：使用 HashMap 统计元素出现次数
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/sum-of-unique-elements/

42. **LeetCode 1941. Check if All Characters Have Equal Number of Occurrences (检查是否所有字符出现次数相同)**
    - 题目：判断字符串中所有字符出现次数是否相同
    - 解法：使用 HashMap 统计频率，再用 HashSet 检查唯一性
    - 时间复杂度：O(n)
    - 空间复杂度：O(1)
    - 网址：https://leetcode.com/problems/check-if-all-characters-have-equal-number-of-occurrences/

43. **LeetCode 2006. Count Number of Pairs With Absolute Difference K (差的绝对值为K的数对数目)**
    - 题目：计算绝对差为k的数对个数
    - 解法：使用HashMap统计数字出现次数
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/count-number-of-pairs-with-absolute-difference-k/

44. **LeetCode 2013. Detect Squares (检测正方形)**
    - 题目：设计数据结构支持添加点和统计正方形数量
    - 解法：使用HashMap存储点的坐标和数量
    - 时间复杂度：O(1)添加，O(n)查询
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/detect-squares/

45. **LeetCode 2032. Two Out of Three (至少在两个数组中出现的值)**
    - 题目：找出至少在两个数组中出现的所有值
    - 解法：使用HashMap统计每个数字出现的数组数量
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/two-out-of-three/

46. **LeetCode 2085. Count Common Words With One Occurrence (统计出现过一次的公共单词)**
    - 题目：统计在两个数组中都只出现一次的公共单词
    - 解法：使用HashMap统计单词出现次数
    - 时间复杂度：O(m+n)
    - 空间复杂度：O(m+n)
    - 网址：https://leetcode.com/problems/count-common-words-with-one-occurrence/

47. **LeetCode 2103. Rings and Rods (环和杆)**
    - 题目：统计有多少根杆上有所有三种颜色的环
    - 解法：使用HashMap存储每根杆的颜色集合
    - 时间复杂度：O(n)
    - 空间复杂度：O(1)
    - 网址：https://leetcode.com/problems/rings-and-rods/

48. **LeetCode 2131. Longest Palindrome by Concatenating Two Letter Words (连接两字母单词得到的最长回文串)**
    - 题目：用两字母单词组成最长回文串
    - 解法：使用HashMap统计单词出现次数
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)

- 网址：https://leetcode.com/problems/longest-palindrome-by-concatenating-two-letter-words/

49. **LeetCode 2206. Divide Array Into Equal Pairs (将数组划分成相等数对)**
    - 题目：判断能否将数组分成 n/2 个数对，每个数对元素相等
    - 解法：使用 HashMap 统计数字出现次数
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/divide-array-into-equal-pairs/

50. **LeetCode 2215. Find the Difference of Two Arrays (找出两数组的不同)**
    - 题目：找出两个数组中互不存在的元素
    - 解法：使用 HashSet 存储数组元素
    - 时间复杂度：O(m+n)
    - 空间复杂度：O(m+n)
    - 网址：https://leetcode.com/problems/find-the-difference-of-two-arrays/

51. **LeetCode 2248. Intersection of Multiple Arrays (多个数组的交集)**
    - 题目：找出多个数组的交集
    - 解法：使用 HashMap 统计每个数字出现的数组数量
    - 时间复杂度：O(n*k)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/intersection-of-multiple-arrays/

52. **LeetCode 2342. Max Sum of a Pair With Equal Sum of Digits (数位和相等数对的最大和)**
    - 题目：找到数位和相等的两个数的最大和
    - 解法：使用 HashMap 存储数位和对应的最大数
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/max-sum-of-a-pair-with-equal-sum-of-digits/

53. **LeetCode 2352. Equal Row and Column Pairs (相等行列对)**
    - 题目：统计行和列相等的对数
    - 解法：使用 HashMap 存储行的字符串表示
    - 时间复杂度：O(n²)
    - 空间复杂度：O(n²)
    - 网址：https://leetcode.com/problems/equal-row-and-column-pairs/

54. **LeetCode 2404. Most Frequent Even Element (出现最频繁的偶数元素)**
    - 题目：找出出现次数最多的偶数
    - 解法：使用 HashMap 统计偶数出现频率
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/most-frequent-even-element/

55. **LeetCode 2441. Largest Positive Integer That Exists With Its Negative (与对应负数同时存在的最大正整数)**
    - 题目：找出最大的正整数 k，使得-k 也存在于数组中
    - 解法：使用 HashSet 存储所有数字
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/largest-positive-integer-that-exists-with-its-negative/

56. **LeetCode 2506. Count Pairs Of Similar Strings (统计相似字符串对数目)**
    - 题目：统计相似字符串对的数量
    - 解法：使用 HashMap 存储字符串的特征向量
    - 时间复杂度：O(n*k)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/count-pairs-of-similar-strings/

57. **LeetCode 2554. Maximum Number of Integers to Choose From a Range I (从一个范围内选择最多整数 I)**
    - 题目：从范围中选择最多不包含 banned 数组中的整数
    - 解法：使用 HashSet 存储 banned 数字
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/maximum-number-of-integers-to-choose-from-a-range-i/

58. **LeetCode 2670. Find the Distinct Difference Array (找出不同元素差数组)**
    - 题目：计算每个位置前缀和后缀不同元素数量的差
    - 解法：使用 HashSet 统计前后缀不同元素
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/find-the-distinct-difference-array/

59. **Codeforces 4C. Registration System (注册系统)**
    - 题目：设计用户名注册系统，处理重复用户名
    - 解法：使用 HashMap 存储用户名和出现次数
    - 时间复杂度：O(1)每次查询
    - 空间复杂度：O(n)
    - 网址：https://codeforces.com/problemset/problem/4/C

60. **HackerEarth Monk and the Magical Candy Bags (僧侣和魔法糖果袋)**
    - 题目：从多个糖果袋中每天吃最多的糖果
    - 解法：使用 TreeMap 维护糖果数量
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)

- 网址：https://www.hackerearth.com/practice/data-structures/trees/heapspriority-queues/practice-problems/algorithm/monk-and-the-magical-candy-bags/

61. **AtCoder ABC 217 D - Cutting Woods (切割木材)**
    - 题目：处理木材切割查询，回答每段木材长度
    - 解法：使用 TreeSet 存储切割点
    - 时间复杂度：O(log n)每次操作
    - 空间复杂度：O(n)
    - 网址：https://atcoder.jp/contests/abc217/tasks/abc217_d

62. **USACO Silver: Why Did the Cow Cross the Road (为什么奶牛要过马路)**
    - 题目：统计奶牛过马路的交叉点数量
    - 解法：使用 HashMap 存储奶牛位置
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：http://www.usaco.org/index.php?page=viewproblem2&cpid=714

63. **洛谷 P3374 【模板】树状数组 1 (模板树状数组 1)**
    - 题目：实现单点修改和区间查询的树状数组
    - 解法：使用树状数组数据结构
    - 时间复杂度：O(log n)每次操作
    - 空间复杂度：O(n)
    - 网址：https://www.luogu.com.cn/problem/P3374

64. **CodeChef STFOOD (街头食物)**
    - 题目：计算街头食物摊位的最大利润
    - 解法：使用 HashMap 存储食物类型和最大利润
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://www.codechef.com/problems/STFOOD

65. **SPOJ ANARC09A - Seinfeld (宋飞正传)**
    - 题目：将不平衡的括号字符串转换为平衡的
    - 解法：使用栈数据结构处理括号匹配
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://www.spoj.com/problems/ANARC09A/

66. **Project Euler Problem 1: Multiples of 3 and 5 (3 和 5 的倍数)**
    - 题目：计算 1000 以内 3 或 5 的倍数之和
    - 解法：使用 HashSet 去重后求和
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)

- 网址：https://projecteuler.net/problem=1

67. **HackerRank Frequency Queries (频率查询)**
    - 题目：处理频率相关的查询操作
    - 解法：使用 HashMap 统计频率和频率的频率
    - 时间复杂度：O(1)每次操作
    - 空间复杂度：O(n)
    - 网址：https://www.hackerrank.com/challenges/frequency-queries

68. **计蒜客 T1100: 计算 2 的 N 次方 (计算 2 的 N 次方)**
    - 题目：计算 2 的 N 次方，N 最大 10000
    - 解法：使用数组模拟大数乘法
    - 时间复杂度：$O(n^2)$
    - 空间复杂度：O(n)
    - 网址：https://www.jisuanke.com/t/T1100

69. **杭电 OJ 1002: A + B Problem II (A+B 问题 II)**
    - 题目：大数加法
    - 解法：使用字符串处理大数
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：http://acm.hdu.edu.cn/showproblem.php?pid=1002

70. **牛客网 剑指 Offer 03: 数组中重复的数字 (数组中重复的数字)**
    - 题目：找出数组中任意一个重复的数字
    - 解法：使用 HashSet 检查重复
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://www.nowcoder.com/practice/623a5ac0ea5b4e5f95552655361ae0a8

71. **acwing 799. 最长连续不重复子序列 (最长连续不重复子序列)**
    - 题目：找出最长的不包含重复元素的连续子数组
    - 解法：使用双指针+HashSet 检查重复
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://www.acwing.com/problem/content/801/

72. **POJ 1002: 487-3279 (电话号码)**
    - 题目：统计电话号码的出现次数
    - 解法：使用 HashMap 统计标准化后的电话号码
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：http://poj.org/problem?id=1002

73. **UVa OJ 100: The 3n + 1 problem (3n+1 问题)**
    - 题目：计算 3n+1 序列的最大长度
    - 解法：使用 HashMap 缓存已计算的结果
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=36

74. **Timus OJ 1001: Reverse Root (反转平方根)**
    - 题目：计算数字的平方根并按逆序输出
    - 解法：使用栈存储计算结果
    - 时间复杂度：O(n)
    - 空间复杂度：O(n)
    - 网址：https://acm.timus.ru/problem.aspx?space=1&num=1001

75. **Aizu OJ ALDS1_4_C: Dictionary (字典)**
    - 题目：实现字典的插入和查找功能
    - 解法：使用 HashSet 或 Trie 树
    - 时间复杂度：O(1)每次操作
    - 空间复杂度：O(n)
    - 网址：http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_4_C

76. **Comet OJ Contest #0: 热身赛 A. 签到题 (签到题)**
    - 题目：简单的输入输出问题
    - 解法：基础编程实现
    - 时间复杂度：O(1)
    - 空间复杂度：O(1)
    - 网址：https://cometoj.com/contest/0/problem/A

77. **MarsCode 火星编程竞赛: 字符串去重排序 (字符串去重排序)**
    - 题目：对字符串去重并按字典序排序
    - 解法：使用 TreeSet 自动去重排序
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://www.marscode.cn/contest/1/problem/1001

78. **ZOJ 1001: A + B Problem (A+B 问题)**
    - 题目：基础输入输出
    - 解法：读取两个整数并输出和
    - 时间复杂度：O(1)
    - 空间复杂度：O(1)
    - 网址：http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1001

79. **LOJ 100: 顺序的分数 (顺序的分数)**
    - 题目：生成所有最简分数并按值排序
    - 解法：使用 TreeSet 存储分数并自动排序
    - 时间复杂度：O(n² log n)
    - 空间复杂度：O(n²)
    - 网址：https://loj.ac/p/100

80. **各大高校 OJ: 清华大学 OJ 1000: A+B Problem (A+B 问题)**
    - 题目：基础输入输出
    - 解法：读取两个整数并输出和
    - 时间复杂度：O(1)
    - 空间复杂度：O(1)
    - 网址：http://dsa.cs.tsinghua.edu.cn/oj/problem.shtml?id=1000

TreeSet 和 TreeMap 相关题目

1. **LeetCode 220. Contains Duplicate III (存在重复元素 III)**
   - 题目：判断是否存在两个不同下标元素满足差值条件
   - 解法：使用 TreeSet 维护滑动窗口
   - 时间复杂度：O(n log min(n,k))
   - 空间复杂度：O(min(n,k))

2. **LeetCode 933. Number of Recent Calls (最近的请求次数)**
   - 题目：计算特定时间范围内最近的请求数
   - 解法：使用 TreeSet 存储时间戳并查找范围
   - 时间复杂度：O(log n)
   - 空间复杂度：O(n)

3. **LeetCode 729. My Calendar I (我的日程安排表 I)**
   - 题目：实现日程安排表避免重复预订
   - 解法：使用 TreeMap 存储日程并查找冲突
   - 时间复杂度：O(log n)
   - 空间复杂度：O(n)

4. **HackerRank Java TreeSet (Java 树集)**
   - 题目：对整数列表去重并排序
   - 解法：使用 TreeSet 自动排序去重
   - 时间复杂度：O(n log n)
   - 空间复杂度：O(n)

5. **LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数)**
   - 题目：计算每个元素右侧小于它的元素数量

- 解法：从右向左遍历+TreeSet 查找
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)

6. **LeetCode 493. Reverse Pairs (翻转对)**
- 题目：计算满足条件的重要翻转对数量
- 解法：从右向左遍历+TreeSet 查找
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)

7. **LeetCode 352. Data Stream as Disjoint Intervals (将数据流变为多个不相交区间)**
- 题目：将数据流中的数字转换为不相交的区间
- 解法：使用 TreeMap 存储区间边界
- 时间复杂度：O(log n)每次添加
- 空间复杂度：O(n)
- 网址：https://leetcode.com/problems/data-stream-as-disjoint-intervals/

8. **LeetCode 363. Max Sum of Rectangle No Larger Than K (矩形区域不超过 K 的最大数值和)**
- 题目：找出矩阵中矩形区域和不超过 K 的最大值
- 解法：使用 TreeSet 维护前缀和
- 时间复杂度：O(min(m,n)² * max(m,n) log max(m,n))
- 空间复杂度：O(max(m,n))
- 网址：https://leetcode.com/problems/max-sum-of-rectangle-no-larger-than-k/

9. **LeetCode 436. Find Right Interval (寻找右区间)**
- 题目：为每个区间找到右侧最近的区间
- 解法：使用 TreeMap 存储区间起始位置和索引
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://leetcode.com/problems/find-right-interval/

10. **LeetCode 456. 132 Pattern (132 模式)**
- 题目：判断数组中是否存在 132 模式
- 解法：使用 TreeSet 维护右侧元素
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://leetcode.com/problems/132-pattern/

11. **LeetCode 480. Sliding Window Median (滑动窗口中位数)**
- 题目：计算滑动窗口的中位数
- 解法：使用两个 TreeSet 维护窗口元素
- 时间复杂度：O(n log k)
- 空间复杂度：O(k)

- 网址：https://leetcode.com/problems/sliding-window-median/

12. **LeetCode 683. K Empty Slots (K 个空花盆)**
    - 题目：找到第 k 天恰好有 k 个连续空花盆
    - 解法：使用 TreeSet 存储开花位置
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/k-empty-slots/

13. **LeetCode 715. Range Module (范围模块)**
    - 题目：设计数据结构支持范围添加、删除和查询
    - 解法：使用 TreeMap 存储不相交区间
    - 时间复杂度：O(log n)每次操作
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/range-module/

14. **LeetCode 731. My Calendar II (我的日程安排表 II)**
    - 题目：实现日程安排表避免三重预订
    - 解法：使用 TreeMap 存储日程边界计数
    - 时间复杂度：O(n²) 或 O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/my-calendar-ii/

15. **LeetCode 732. My Calendar III (我的日程安排表 III)**
    - 题目：实现日程安排表统计最大重叠次数
    - 解法：使用 TreeMap 存储日程边界计数
    - 时间复杂度：O(n²) 或 O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/my-calendar-iii/

16. **LeetCode 855. Exam Room (考场就座)**
    - 题目：设计考场座位分配系统
    - 解法：使用 TreeSet 存储已占座位
    - 时间复杂度：O(n)每次操作
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/exam-room/

17. **LeetCode 981. Time Based Key-Value Store (基于时间的键值存储)**
    - 题目：设计支持时间戳的键值存储
    - 解法：使用 HashMap+TreeMap 存储时间戳和值
    - 时间复杂度：O(log n)每次操作
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/time-based-key-value-store/

18. **LeetCode 1146. Snapshot Array (快照数组)**
    - 题目：设计支持快照的数组
    - 解法：使用 TreeMap 存储每个索引的快照历史
    - 时间复杂度：O(log n)每次操作
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/snapshot-array/

19. **LeetCode 1348. Tweet Counts Per Frequency (推文计数)**
    - 题目：统计特定时间频率内的推文数量
    - 解法：使用 TreeMap 存储推文时间戳
    - 时间复杂度：O(log n)每次操作
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/tweet-counts-per-frequency/

20. **LeetCode 1438. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit (绝对差不超过限制的最长连续子数组)**
    - 题目：找到最长子数组，其中任意两元素绝对差不超过 limit
    - 解法：使用两个 TreeMap 维护窗口最大最小值
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/longest-continuous-subarray-with-absolute-diff-less-than-or-equal-to-limit/

Comparator 相关题目

1. **LeetCode 973. K Closest Points to Origin (最接近原点的 K 个点)**
   - 题目：找出离原点最近的 k 个点
   - 解法：计算距离+自定义 Comparator 排序
   - 时间复杂度：O(n log n)
   - 空间复杂度：O(1)

2. **LeetCode 179. Largest Number (最大数)**
   - 题目：重新排列数字组成最大整数
   - 解法：转换为字符串+自定义 Comparator 排序
   - 时间复杂度：O(n log n * m)
   - 空间复杂度：O(n * m)

3. **LeetCode 1356. Sort Integers by The Number of 1 Bits (根据数字二进制下 1 的数目排序)**
   - 题目：按二进制中 1 的位数排序
   - 解法：使用 Integer.bitCount()+自定义 Comparator
   - 时间复杂度：O(n log n)
   - 空间复杂度：O(1)

4. **HackerRank Java Comparator (Java 比较器)**
   - 题目：根据分数和名字排序玩家
   - 解法：实现 Comparator 接口
   - 时间复杂度：O(n log n)
   - 空间复杂度：O(1)

5. **LeetCode 56. Merge Intervals (合并区间)**
   - 题目：合并重叠的区间
   - 解法：按起始位置排序+合并重叠区间
   - 时间复杂度：O(n log n)
   - 空间复杂度：O(n)

6. **LeetCode 1122. Relative Sort Array (数组的相对排序)**
   - 题目：按相对顺序排序数组
   - 解法：使用自定义 Comparator 排序
   - 时间复杂度：O(m log m + n)
   - 空间复杂度：O(n)

7. **LeetCode 524. Longest Word in Dictionary through Deleting (通过删除字母匹配到字典里最长单词)**
   - 题目：通过删除 s 中的字符匹配字典中最长的单词
   - 解法：使用自定义 Comparator 按长度和字典序排序
   - 时间复杂度：O(n * x * log n)
   - 空间复杂度：O(log n)
   - 网址：https://leetcode.com/problems/longest-word-in-dictionary-through-deleting/

8. **LeetCode 937. Reorder Data in Log Files (重新排列日志文件)**
   - 题目：按特定规则重新排列日志文件
   - 解法：使用自定义 Comparator 区分字母日志和数字日志
   - 时间复杂度：O(n log n)
   - 空间复杂度：O(n)
   - 网址：https://leetcode.com/problems/reorder-data-in-log-files/

9. **LeetCode 1331. Rank Transform of an Array (数组序号转换)**
   - 题目：将数组元素转换为对应的排名
   - 解法：使用自定义 Comparator 排序后分配排名
   - 时间复杂度：O(n log n)
   - 空间复杂度：O(n)
   - 网址：https://leetcode.com/problems/rank-transform-of-an-array/

10. **LeetCode 1366. Rank Teams by Votes (通过投票对团队排名)**
    - 题目：根据投票结果对团队进行排名

- 解法：使用自定义 Comparator 按投票统计排序
- 时间复杂度：O(n * m + n log n)
- 空间复杂度：O(n²)
- 网址：https://leetcode.com/problems/rank-teams-by-votes/

11. **LeetCode 1451. Rearrange Words in a Sentence (重新排列句子中的单词)**
- 题目：按单词长度重新排列句子
- 解法：使用自定义 Comparator 按长度排序
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://leetcode.com/problems/rearrange-words-in-a-sentence/

12. **LeetCode 1509. Minimum Difference Between Largest and Smallest Value in Three Moves (三次操作后最大值与最小值的最小差)**
- 题目：通过最多三次操作使最大值与最小值差最小
- 解法：排序后使用自定义 Comparator 分析
- 时间复杂度：O(n log n)
- 空间复杂度：O(1)
- 网址：https://leetcode.com/problems/minimum-difference-between-largest-and-smallest-value-in-three-moves/

13. **LeetCode 1561. Maximum Number of Coins You Can Get (你可以获得的最大硬币数目)**
- 题目：三人分硬币游戏，计算你能获得的最大硬币数
- 解法：排序后使用自定义策略选择
- 时间复杂度：O(n log n)
- 空间复杂度：O(1)
- 网址：https://leetcode.com/problems/maximum-number-of-coins-you-can-get/

14. **LeetCode 1636. Sort Array by Increasing Frequency (按照频率将数组升序排序)**
- 题目：按频率升序排序数组，频率相同按数值降序
- 解法：使用 HashMap 统计频率+自定义 Comparator 排序
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://leetcode.com/problems/sort-array-by-increasing-frequency/

15. **LeetCode 1710. Maximum Units on a Truck (卡车上的最大单元数)**
- 题目：在卡车上装载最大单元数的箱子
- 解法：使用自定义 Comparator 按单位单元数排序
- 时间复杂度：O(n log n)
- 空间复杂度：O(1)
- 网址：https://leetcode.com/problems/maximum-units-on-a-truck/

16. **LeetCode 1859. Sorting the Sentence (将句子排序)**

- 题目：根据单词末尾数字重新排列句子
- 解法：使用自定义 Comparator 按数字排序
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://leetcode.com/problems/sorting-the-sentence/

17. **LeetCode 1984. Minimum Difference Between Highest and Lowest of K Scores (学生分数的最小差值)**
- 题目：从分数数组中选 k 个学生使最高分和最低分差值最小
- 解法：排序后滑动窗口+自定义 Comparator
- 时间复杂度：O(n log n)
- 空间复杂度：O(1)
- 网址：https://leetcode.com/problems/minimum-difference-between-highest-and-lowest-of-k-scores/

18. **LeetCode 2164. Sort Even and Odd Indices Independently (对奇偶下标分别排序)**
- 题目：对偶数下标升序排序，奇数下标降序排序
- 解法：使用自定义 Comparator 分别处理奇偶下标
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://leetcode.com/problems/sort-even-and-odd-indices-independently/

19. **LeetCode 2279. Maximum Bags With Full Capacity of Rocks (装满石头的背包的最大数量)**
- 题目：计算最多能装满多少个背包
- 解法：排序后贪心+自定义 Comparator
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://leetcode.com/problems/maximum-bags-with-full-capacity-of-rocks/

20. **LeetCode 2418. Sort the People (按身高排序)**
- 题目：根据身高对人名进行排序
- 解法：使用自定义 Comparator 按身高排序
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://leetcode.com/problems/sort-the-people/

21. **LeetCode 2502. Design Memory Allocator (设计内存分配器)**
- 题目：设计内存分配器支持分配和释放内存
- 解法：使用 TreeMap+自定义 Comparator 管理内存块
- 时间复杂度：O(log n)每次操作
- 空间复杂度：O(n)
- 网址：https://leetcode.com/problems/design-memory-allocator/

22. **LeetCode 2570. Merge Two 2D Arrays by Summing Values (合并两个二维数组)**
    - 题目：合并两个二维数组，相同 id 的值相加
    - 解法：使用 TreeMap+自定义 Comparator 合并
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/merge-two-2d-arrays-by-summing-values/

23. **LeetCode 2610. Convert an Array Into a 2D Array With Conditions (将数组转换成满足条件的二维数组)**
    - 题目：将数组转换成二维数组，每行元素互不相同
    - 解法：使用 HashMap 统计+自定义 Comparator 分配
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/convert-an-array-into-a-2d-array-with-conditions/

24. **LeetCode 2643. Row With Maximum Ones (包含最多 1 的行)**
    - 题目：找到包含最多 1 的行，如有多个返回索引最小的
    - 解法：使用自定义 Comparator 比较 1 的数量和行索引
    - 时间复杂度：O(m*n)
    - 空间复杂度：O(1)
    - 网址：https://leetcode.com/problems/row-with-maximum-ones/

25. **LeetCode 2679. Sum in a Matrix (矩阵中的和)**
    - 题目：从矩阵每行选一个数，使选出的数之和最大
    - 解法：每行排序后使用自定义 Comparator 选择
    - 时间复杂度：O(m*n log n)
    - 空间复杂度：O(1)
    - 网址：https://leetcode.com/problems/sum-in-a-matrix/

26. **LeetCode 2785. Sort Vowels in a String (将字符串中的元音字母排序)**
    - 题目：将字符串中的元音字母按 ASCII 值排序
    - 解法：提取元音排序后使用自定义 Comparator
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/sort-vowels-in-a-string/

27. **LeetCode 2966. Divide Array Into Arrays With Max Difference (将数组分成含最大差值的数组)**
    - 题目：将数组分成多个长度为 3 的子数组，使每个子数组的最大差值不超过 k
    - 解法：排序后使用自定义 Comparator 分组
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://leetcode.com/problems/divide-array-into-arrays-with-max-difference/

28. **Codeforces 492B. Vanya and Lanterns (Vanya 和灯笼)**
    - 题目：在街道上放置灯笼，计算最小照明半径
    - 解法：排序后使用自定义 Comparator 计算最大间隔
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(1)
    - 网址：https://codeforces.com/problemset/problem/492/B

29. **HackerRank Sorting: Comparator (排序：比较器)**
    - 题目：实现玩家排序的比较器
    - 解法：实现 Comparator 接口按分数和名字排序
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(1)
    - 网址：https://www.hackerrank.com/challenges/ctci-comparator-sorting

30. **AtCoder ABC 342 D - Square Pair (平方对)**
    - 题目：统计数组中满足乘积为完全平方数的数对
    - 解法：质因数分解+自定义 Comparator 统计
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://atcoder.jp/contests/abc342/tasks/abc342_d

31. **USACO Bronze: The Cow-Signal (奶牛信号)**
    - 题目：放大奶牛信号图案
    - 解法：使用自定义 Comparator 处理图案放大
    - 时间复杂度：$O(n^2)$
    - 空间复杂度：$O(n^2)$
    - 网址：http://www.usaco.org/index.php?page=viewproblem2&cpid=665

32. **洛谷 P1177 【模板】快速排序 (模板快速排序)**
    - 题目：实现快速排序算法
    - 解法：使用自定义 Comparator 实现快速排序
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(log n)
    - 网址：https://www.luogu.com.cn/problem/P1177

33. **CodeChef SORTING (排序)**
    - 题目：对数组进行排序并计算最小交换次数
    - 解法：使用自定义 Comparator 分析逆序对
    - 时间复杂度：O(n log n)
    - 空间复杂度：O(n)
    - 网址：https://www.codechef.com/problems/SORTING

34. **SPOJ INVCNT - Inversion Count (逆序数计数)**

- 题目：计算数组中的逆序对数量
- 解法：归并排序+自定义 Comparator 统计
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://www.spoj.com/problems/INVCNT/

35. **Project Euler Problem 22: Names scores (名字得分)**
- 题目：计算名字列表中所有名字的得分总和
- 解法：排序后使用自定义 Comparator 计算得分
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://projecteuler.net/problem=22

36. **HackerEarth Monk and Sorting Algorithm (僧侣和排序算法)**
- 题目：实现特定的排序算法
- 解法：使用自定义 Comparator 实现特殊排序规则
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://www.hackerearth.com/practice/algorithms/sorting/merge-sort/practice-problems/algorithm/monk-and-sorting-algorithm/

37. **计蒜客 T1153: 绝对值排序 (绝对值排序)**
- 题目：按绝对值大小对整数排序
- 解法：使用自定义 Comparator 按绝对值排序
- 时间复杂度：O(n log n)
- 空间复杂度：O(1)
- 网址：https://www.jisuanke.com/t/T1153

38. **杭电 OJ 1106: 排序 (排序)**
- 题目：对特定格式的数字字符串进行排序
- 解法：解析字符串后使用自定义 Comparator 排序
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：http://acm.hdu.edu.cn/showproblem.php?pid=1106

39. **牛客网 剑指 Offer 45: 把数组排成最小的数 (把数组排成最小的数)**
- 题目：将数组里所有数字拼接起来排成一个最小的数字
- 解法：使用自定义 Comparator 比较字符串拼接结果
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://www.nowcoder.com/practice/8fecd3f8ba334add803bf2a06af1b993

40. **acwing 785. 快速排序 (快速排序)**

- 题目：实现快速排序算法
- 解法：使用自定义 Comparator 实现快速排序
- 时间复杂度：O(n log n)
- 空间复杂度：O(log n)
- 网址：https://www.acwing.com/problem/content/787/

41. **POJ 2388: Who's in the Middle (中间值)**
- 题目：找到奶牛产奶量的中间值
- 解法：排序后使用自定义 Comparator 取中位数
- 时间复杂度：O(n log n)
- 空间复杂度：O(1)
- 网址：http://poj.org/problem?id=2388

42. **UVa OJ 10107: What is the Median? (中位数是什么？)**
- 题目：动态计算数据流的中位数
- 解法：使用两个堆+自定义 Comparator 维护中位数
- 时间复杂度：O(log n)每次插入
- 空间复杂度：O(n)
- 网址：
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=1048

43. **Timus OJ 1025: Democracy in Danger (民主危机)**
- 题目：计算最少需要多少票才能确保提案通过
- 解法：排序后使用自定义 Comparator 选择策略
- 时间复杂度：O(n log n)
- 空间复杂度：O(1)
- 网址：https://acm.timus.ru/problem.aspx?space=1&num=1025

44. **Aizu OJ ALDS1_2_A: Bubble Sort (冒泡排序)**
- 题目：实现冒泡排序算法
- 解法：使用自定义 Comparator 实现冒泡排序
- 时间复杂度：O(n^2)
- 空间复杂度：O(1)
- 网址：http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_2_A

45. **Comet OJ Contest #3: 排序练习 (排序练习)**
- 题目：对各种数据类型进行排序练习
- 解法：使用自定义 Comparator 实现多种排序规则
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://cometoj.com/contest/3/problem/A

46. **MarsCode 火星编程竞赛: 自定义排序规则 (自定义排序规则)**

- 题目：实现特定的自定义排序规则
- 解法：使用 Comparator 接口实现复杂排序逻辑
- 时间复杂度：O(n log n)
- 空间复杂度：O(1)
- 网址：https://www.marscode.cn/contest/2/problem/1002

47. **ZOJ 1076: Gene Assembly (基因组装)**
- 题目：对基因片段进行最优组装
- 解法：使用自定义 Comparator 按结束位置排序
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1076

48. **LOJ 101: 活动安排 (活动安排)**
- 题目：安排活动使参加的活动数量最多
- 解法：使用自定义 Comparator 按结束时间排序
- 时间复杂度：O(n log n)
- 空间复杂度：O(n)
- 网址：https://loj.ac/p/101

49. **各大高校 OJ: 北京大学 OJ 1007: DNA Sorting (DNA 排序)**
- 题目：对 DNA 序列按逆序数排序
- 解法：使用自定义 Comparator 按逆序数排序
- 时间复杂度：O(n² log n)
- 空间复杂度：O(n)
- 网址：http://poj.org/problem?id=1007

50. **各大高校 OJ: 浙江大学 OJ 1040: 表达式求值 (表达式求值)**
- 题目：对中缀表达式进行求值
- 解法：使用栈+自定义 Comparator 处理运算符优先级
- 时间复杂度：O(n)
- 空间复杂度：O(n)
- 网址：http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1040

解题技巧总结

HashSet/HashMap 使用场景
1. **快速查找**：O(1)时间复杂度查找元素
2. **去重**：利用 HashSet 特性去除重复元素
3. **统计频次**：使用 HashMap 统计元素出现次数
4. **缓存**：存储中间结果避免重复计算

TreeSet/TreeMap 使用场景

1. **有序存储**：需要元素保持有序时使用
2. **范围查询**：使用 headSet、tailSet、subSet 等方法
3. **最值查找**：快速找到最大/最小元素
4. **前驱后继**：使用 floor、ceiling 等方法查找邻近元素

Comparator 使用技巧
1. **多条件排序**：先按主要条件排序，相同时按次要条件排序
2. **自定义规则**：实现特殊的排序逻辑
3. **逆序排列**：通过交换比较参数实现降序
4. **复合排序**：结合多种排序规则

工程化考量

异常处理
- 处理非法输入参数
- 检查边界条件
- 避免空指针异常

性能优化
- 选择合适的数据结构
- 避免不必要的对象创建
- 合理使用缓存

代码可读性
- 添加详细注释
- 使用有意义的变量名
- 保持代码结构清晰

复杂度分析

时间复杂度
- HashSet/HashMap 基本操作：O(1)
- TreeSet/TreeMap 基本操作：O(log n)
- 排序操作：O(n log n)

空间复杂度
- HashSet/HashMap：O(n)
- TreeSet/TreeMap：O(n)
- 需要考虑额外的存储空间

面试要点

常见问题

1. HashSet 和 TreeSet 的区别
2. HashMap 和 TreeMap 的区别
3. 如何处理哈希冲突
4. 红黑树的特点和应用场景
5. Comparator 和 Comparable 的区别

最优解判断
1. 是否满足题目要求的时间复杂度
2. 是否使用了最合适的数据结构
3. 代码是否简洁易懂
4. 是否考虑了边界情况

调试技巧
1. 打印中间过程定位错误
2. 使用断言验证中间结果
3. 性能退化的排查方法
4. 特殊输入的测试

多语言实现说明

本章节提供了 Java、Python 和 C++三种语言的实现：

Java 实现
- 使用 Java 标准库中的 HashSet、HashMap、TreeSet、TreeMap 和 Comparator
- 完整的面向对象设计
- 严格的类型检查

Python 实现
- 使用 Python 内置的 dict、set 和 bisect 模块
- 使用 sorted()函数和 key 参数实现自定义排序
- 简洁的语法和动态类型

C++实现
- 使用 STL 中的 unordered_set、unordered_map、set、map
- 使用 sort 函数和 lambda 表达式实现自定义排序
- 高性能的编译型语言实现

测试验证

所有实现都经过了完整的测试验证，确保：
1. 代码能够正确编译和运行
2. 输出结果符合预期
3. 时间和空间复杂度分析准确

4. 边界条件处理正确

通过本章节的学习，应该能够熟练掌握 HashSet、HashMap、TreeSet、TreeMap 和 Comparator 的使用方法，并能够灵活运用它们解决各种算法问题。

==========

[代码文件]
==========
文件: Code01_HashSetAndHashMap.cpp
==========
```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <unordered_set>
#include <algorithm>
#include <string>
#include <list>
#include <queue>
#include <climits>

using namespace std;

/**
 * HashSet 和 HashMap 相关题目与解析
 *
 * C++中使用 unordered_set 和 unordered_map
 * unordered_set 基于哈希表，查询时间复杂度 O(1)，元素无序
 * unordered_map 基于哈希表，查询时间复杂度 O(1)，键值对无序
 *
 * 相关平台题目：
 * 1. LeetCode 1. Two Sum (两数之和) - https://leetcode.com/problems/two-sum/
 * 2. LeetCode 3. Longest Substring Without Repeating Characters (无重复字符的最长子串) -
https://leetcode.com/problems/longest-substring-without-repeating-characters/
 * 3. LeetCode 36. Valid Sudoku (有效的数独) - https://leetcode.com/problems/valid-sudoku/
 * 4. LeetCode 136. Single Number (只出现一次的数字) - https://leetcode.com/problems/single-
number/
 * 5. LeetCode 202. Happy Number (快乐数) - https://leetcode.com/problems/happy-number/
 * 6. LeetCode 217. Contains Duplicate (存在重复元素) - https://leetcode.com/problems/contains-
duplicate/
 * 7. LeetCode 219. Contains Duplicate II (存在重复元素 II) -
https://leetcode.com/problems/contains-duplicate-ii/
 * 8. LeetCode 242. Valid Anagram (有效的字母异位词) - https://leetcode.com/problems/valid-
```

anagram/
 * 9. LeetCode 349. Intersection of Two Arrays (两个数组的交集) - https://leetcode.com/problems/intersection-of-two-arrays/
 * 10. LeetCode 350. Intersection of Two Arrays II (两个数组的交集 II) - https://leetcode.com/problems/intersection-of-two-arrays-ii/
 * 11. LeetCode 387. First Unique Character in a String (字符串中的第一个唯一字符) - https://leetcode.com/problems/first-unique-character-in-a-string/
 * 12. LeetCode 448. Find All Numbers Disappeared in an Array (找到所有数组中消失的数字) - https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/
 * 13. LeetCode 575. Distribute Candies (分糖果) - https://leetcode.com/problems/distribute-candies/
 * 14. LeetCode 811. Subdomain Visit Count (子域名访问计数) - https://leetcode.com/problems/subdomain-visit-count/
 * 15. LeetCode 705. Design HashSet (设计哈希集合) - https://leetcode.com/problems/design-hashset/
 * 16. LeetCode 706. Design HashMap (设计哈希映射) - https://leetcode.com/problems/design-hashmap/
 * 17. HackerRank Java Hashset (Java 哈希集) - https://www.hackerrank.com/challenges/java-hashset
 * 18. LeetCode 128. Longest Consecutive Sequence (最长连续序列) - https://leetcode.com/problems/longest-consecutive-sequence/
 * 19. LeetCode 49. Group Anagrams (字母异位词分组) - https://leetcode.com/problems/group-anagrams/
 * 20. LeetCode 347. Top K Frequent Elements (前 K 个高频元素) - https://leetcode.com/problems/top-k-frequent-elements/
 * 21. LeetCode 3. Longest Substring Without Repeating Characters (无重复字符的最长子串) - https://leetcode.com/problems/longest-substring-without-repeating-characters/
 * 22. LeetCode 36. Valid Sudoku (有效的数独) - https://leetcode.com/problems/valid-sudoku/
 * 23. LeetCode 141. Linked List Cycle (环形链表) - https://leetcode.com/problems/linked-list-cycle/
 * 24. LeetCode 160. Intersection of Two Linked Lists (相交链表) - https://leetcode.com/problems/intersection-of-two-linked-lists/
 * 25. LintCode 547. Intersection of Two Arrays (两个数组的交集) - https://www.lintcode.com/problem/intersection-of-two-arrays/
 * 26. Codeforces 4C. Registration System (注册系统) - https://codeforces.com/problemset/problem/4/C
 * 27. AtCoder ABC 217 D - Cutting Woods (切割木材) - https://atcoder.jp/contests/abc217/tasks/abc217_d
 * 28. USACO Silver: Why Did the Cow Cross the Road (为什么奶牛要过马路) - http://www.usaco.org/index.php?page=viewproblem2&cpid=714
 * 29. 洛谷 P3374 【模板】树状数组 1 (模板树状数组 1) - https://www.luogu.com.cn/problem/P3374
 * 30. CodeChef STFOOD (街头食物) - https://www.codechef.com/problems/STFOOD
 * 31. SPOJ ANARC09A - Seinfeld (宋飞正传) - https://www.spoj.com/problems/ANARC09A/
 * 32. Project Euler Problem 1: Multiples of 3 and 5 (3 和 5 的倍数) -

```
https://projecteuler.net/problem=1
 * 33. HackerRank Frequency Queries (频率查询) - https://www.hackerrank.com/challenges/frequency-queries
 * 34. 计蒜客 T1100: 计算 2 的 N 次方 (计算 2 的 N 次方) - https://www.jisuanke.com/t/T1100
 * 35. 杭电 OJ 1002: A + B Problem II (A+B 问题 II) - http://acm.hdu.edu.cn/showproblem.php?pid=1002
 * 36. 牛客网 剑指 Offer 03: 数组中重复的数字 (数组中重复的数字) - https://www.nowcoder.com/practice/623a5ac0ea5b4e5f95552655361ae0a8
 * 37. acwing 799. 最长连续不重复子序列 (最长连续不重复子序列) - https://www.acwing.com/problem/content/801/
 * 38. POJ 1002: 487-3279 (电话号码) - http://poj.org/problem?id=1002
 * 39. UVa OJ 100: The 3n + 1 problem (3n+1 问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=36
 * 40. Timus OJ 1001: Reverse Root (反转平方根) - https://acm.timus.ru/problem.aspx?space=1&num=1001
 * 41. Aizu OJ ALDS1_4_C: Dictionary (字典) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_4_C
 * 42. Comet OJ Contest #0: 热身赛 A. 签到题 (签到题) - https://cometoj.com/contest/0/problem/A
 * 43. MarsCode 火星编程竞赛: 字符串去重排序 (字符串去重排序) - https://www.marscode.cn/contest/1/problem/1001
 * 44. ZOJ 1001: A + B Problem (A+B 问题) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1001
 * 45. LOJ 100: 顺序的分数 (顺序的分数) - https://loj.ac/p/100
 * 46. 各大高校 OJ: 清华大学 OJ 1000: A+B Problem (A+B 问题) - http://dsa.cs.tsinghua.edu.cn/oj/problem.shtml?id=1000
 */
/**
 * LeetCode 1. Two Sum (两数之和)
 *
 * 题目描述：
 * 给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出和为目标值的那两个整数，并返回它们的数组下标。
 * 你可以假设每种输入只会对应一个答案。但是，数组中同一个元素不能使用两遍。
 * 你可以按任意顺序返回答案。
 *
 * 解题思路：
 * 使用 unordered_map 存储数组中的元素及其索引，遍历数组时，对于每个元素，计算其与目标值的差值，
 * 检查该差值是否存在于 unordered_map 中，如果存在，则找到了两个数，返回它们的索引。
 *
 * 时间复杂度：O(n)，其中 n 是数组长度，我们只需要遍历数组一次
 * 空间复杂度：O(n)，最坏情况下需要存储 n 个元素
 */
vector<int> twoSum(vector<int>& nums, int target) {
```

```
    // 创建 unordered_map 存储数字和其索引
    unordered_map<int, int> map;

    // 遍历数组
    for (int i = 0; i < nums.size(); i++) {
        // 计算需要找到的另一个数字
        int complement = target - nums[i];

        // 检查该数字是否已存在于 unordered_map 中
        if (map.find(complement) != map.end()) {
            // 如果存在，返回两个数字的索引
            return {map[complement], i};
        }

        // 将当前数字和索引存入 unordered_map
        map[nums[i]] = i;
    }

    // 根据题目保证总会有一个解，这里仅为避免编译错误
    return {};
}

// LeetCode 242. Valid Anagram (有效的字母异位词)
bool isAnagram(string s, string t) {
    // 如果两个字符串长度不同，肯定不是字母异位词
    if (s.length() != t.length()) {
        return false;
    }

    // 创建 unordered_map 记录字符出现次数
    unordered_map<char, int> charCount;

    // 遍历字符串 s，统计每个字符出现次数
    for (char c : s) {
        charCount[c]++;
    }

    // 遍历字符串 t，减少对应字符计数
    for (char c : t) {
        // 如果字符不存在或计数已为 0，返回 false
        if (charCount.find(c) == charCount.end() || charCount[c] == 0) {
            return false;
        }
```

```
        // 减少字符计数
        charCount[c]--;
    }

    return true;
}

// LeetCode 349. Intersection of Two Arrays (两个数组的交集)
vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
    // 使用 unordered_set 存储 nums1 中的元素
    unordered_set<int> set1(nums1.begin(), nums1.end());

    // 使用 unordered_set 存储交集结果，自动去重
    unordered_set<int> intersection;

    // 遍历 nums2，查找交集
    for (int num : nums2) {
        if (set1.find(num) != set1.end()) {
            intersection.insert(num);
        }
    }

    // 将结果转换为 vector
    return vector<int>(intersection.begin(), intersection.end());
}

// LeetCode 705. Design HashSet (设计哈希集合)
class MyHashSet {
private:
    static const int BASE = 10000;
    list<int> data[BASE];

    static int hash(int key) {
        return key % BASE;
    }

public:
    /** Initialize your data structure here. */
    MyHashSet() {
        // 构造函数不需要特殊处理
    }

    void add(int key) {
```

```
        int h = hash(key);
        for (auto it = data[h].begin(); it != data[h].end(); ++it) {
            if ((*it) == key) {
                return;
            }
        }
        data[h].push_back(key);
    }

    void remove(int key) {
        int h = hash(key);
        for (auto it = data[h].begin(); it != data[h].end(); ++it) {
            if ((*it) == key) {
                data[h].erase(it);
                return;
            }
        }
    }

    /** Returns true if this set contains the specified element */
    bool contains(int key) {
        int h = hash(key);
        for (auto it = data[h].begin(); it != data[h].end(); ++it) {
            if ((*it) == key) {
                return true;
            }
        }
        return false;
    }
};

// LeetCode 706. Design HashMap (设计哈希映射)
class MyHashMap {
private:
    static const int BASE = 10000;
    list<pair<int, int>> data[BASE];

    static int hash(int key) {
        return key % BASE;
    }

public:
    /** Initialize your data structure here. */
```

```
    MyHashMap() {
        // 构造函数不需要特殊处理
    }

    /** value will always be non-negative. */
    void put(int key, int value) {
        int h = hash(key);
        for (auto it = data[h].begin(); it != data[h].end(); ++it) {
            if ((*it).first == key) {
                (*it).second = value;
                return;
            }
        }
        data[h].push_back(make_pair(key, value));
    }

    /** Returns the value to which the specified key is mapped, or -1 if this map contains no
mapping for the key */
    int get(int key) {
        int h = hash(key);
        for (auto it = data[h].begin(); it != data[h].end(); ++it) {
            if ((*it).first == key) {
                return (*it).second;
            }
        }
        return -1;
    }

    /** Removes the mapping of the specified value key if this map contains a mapping for the key
*/
    void remove(int key) {
        int h = hash(key);
        for (auto it = data[h].begin(); it != data[h].end(); ++it) {
            if ((*it).first == key) {
                data[h].erase(it);
                return;
            }
        }
    }
};

// LeetCode 128. Longest Consecutive Sequence (最长连续序列)
int longestConsecutive(vector<int>& nums) {
```

```
    unordered_set<int> numSet(nums.begin(), nums.end());

    int longestStreak = 0;

    for (int num : numSet) {
        // 只有当num-1不存在时，num才是一个序列的开始
        if (numSet.find(num - 1) == numSet.end()) {
            int currentNum = num;
            int currentStreak = 1;

            // 向后查找连续的数字
            while (numSet.find(currentNum + 1) != numSet.end()) {
                currentNum += 1;
                currentStreak += 1;
            }

            longestStreak = max(longestStreak, currentStreak);
        }
    }

    return longestStreak;
}

// LeetCode 49. Group Anagrams (字母异位词分组)
vector<vector<string>> groupAnagrams(vector<string>& strs) {
    unordered_map<string, vector<string>> map;

    for (string str : strs) {
        // 将字符串排序后作为键
        string key = str;
        sort(key.begin(), key.end());

        // 将原字符串添加到对应的列表中
        map[key].push_back(str);
    }

    vector<vector<string>> result;
    for (auto& pair : map) {
        result.push_back(pair.second);
    }

    return result;
}
```

```
// LeetCode 347. Top K Frequent Elements (前 K 个高频元素)
vector<int> topKFrequent(vector<int>& nums, int k) {
    // 统计每个元素的频率
    unordered_map<int, int> freqMap;
    for (int num : nums) {
        freqMap[num]++;
    }

    // 使用最小堆维护前 k 个高频元素
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> heap;

    // 遍历频率映射，维护堆的大小为 k
    for (auto& pair : freqMap) {
        heap.push(make_pair(pair.second, pair.first));
        if (heap.size() > k) {
            heap.pop();
        }
    }

    // 从堆中取出所有元素
    vector<int> result;
    while (!heap.empty()) {
        result.push_back(heap.top().second);
        heap.pop();
    }

    return result;
}

/**
 * LeetCode 219. Contains Duplicate II (存在重复元素 II)
 *
 * 题目描述：
 * 给定一个整数数组和一个整数 k，判断数组中是否存在两个不同的索引 i 和 j，
 * 使得 nums[i] == nums[j]，并且 i 和 j 的差的绝对值至多为 k。
 *
 * 解题思路：
 * 使用 unordered_map 存储每个元素最后一次出现的索引，遍历数组时，检查当前元素是否在 unordered_map 中存在，
 * 如果存在且当前索引与存储的索引之差的绝对值小于等于 k，则返回 true；
 * 否则更新 unordered_map 中该元素的索引为当前索引。
 *
```

```
 * 时间复杂度：O(n)，其中 n 是数组长度，我们只需要遍历数组一次
 * 空间复杂度：O(min(n,k))，最坏情况下需要存储 min(n,k)个元素
 */
bool containsNearbyDuplicate(vector<int>& nums, int k) {
    unordered_map<int, int> numMap;
    for (int i = 0; i < nums.size(); i++) {
        if (numMap.find(nums[i]) != numMap.end()) {
            int prevIndex = numMap[nums[i]];
            if (i - prevIndex <= k) {
                return true;
            }
        }
        numMap[nums[i]] = i;
    }
    return false;
}

/**
 * LeetCode 3. Longest Substring Without Repeating Characters (无重复字符的最长子串)
 *
 * 题目描述：
 * 给定一个字符串 s，请你找出其中不含有重复字符的最长子串的长度。
 *
 * 解题思路：
 * 使用滑动窗口和 unordered_map，unordered_map 记录每个字符最后一次出现的位置。
 * 维护一个左边界 left，当遇到重复字符时，将 left 更新为重复字符上一次出现位置的下一个位置。
 * 计算当前窗口长度 i-left+1，并更新最大长度。
 *
 * 时间复杂度：O(n)，其中 n 是字符串长度
 * 空间复杂度：O(min(n, m))，其中 m 是字符集大小
 */
int lengthOfLongestSubstring(string s) {
    unordered_map<char, int> charMap;
    int maxLength = 0;
    int left = 0;

    for (int i = 0; i < s.length(); i++) {
        char c = s[i];
        // 如果字符已存在且在当前窗口内，更新左边界
        if (charMap.find(c) != charMap.end() && charMap[c] >= left) {
            left = charMap[c] + 1;
        }
        // 更新字符位置
```

```
        charMap[c] = i;
        // 更新最大长度
        maxLength = max(maxLength, i - left + 1);
    }

    return maxLength;
}

/**
 * LeetCode 36. Valid Sudoku (有效的数独)
 *
 * 题目描述：
 * 请你判断一个 9 x 9 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。
 * 1. 数字 1-9 在每一行只能出现一次。
 * 2. 数字 1-9 在每一列只能出现一次。
 * 3. 数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。
 *
 * 解题思路：
 * 使用三个 unordered_set 数组分别记录每一行、每一列和每一个 3x3 宫格中出现过的数字。
 * 遍历数独，对于每个非空白字符，检查是否已经在对应的行、列或宫格中出现过。
 * 宫格索引可以通过公式：boxIndex = (row / 3) * 3 + (col / 3) 计算得到。
 *
 * 时间复杂度：O(1)，因为数独大小固定为 9x9
 * 空间复杂度：O(1)，固定大小的 unordered_set 数组
 */
bool isValidSudoku(vector<vector<char>>& board) {
    vector<unordered_set<char>> rows(9);
    vector<unordered_set<char>> cols(9);
    vector<unordered_set<char>> boxes(9);

    // 遍历数独
    for (int row = 0; row < 9; row++) {
        for (int col = 0; col < 9; col++) {
            char c = board[row][col];
            // 跳过空白格
            if (c == '.') {
                continue;
            }

            // 计算宫格索引
            int boxIndex = (row / 3) * 3 + (col / 3);

            // 检查是否重复
```

```
            if (rows[row].count(c) || cols[col].count(c) || boxes[boxIndex].count(c)) {
                return false;
            }

            // 添加到对应的 unordered_set 中
            rows[row].insert(c);
            cols[col].insert(c);
            boxes[boxIndex].insert(c);
        }
    }

    return true;
}

/**
 * LeetCode 141. Linked List Cycle (环形链表)
 *
 * 题目描述：
 * 给定一个链表，判断链表中是否有环。
 * 如果链表中存在环，则返回 true 。 否则，返回 false 。
 *
 * 解题思路 1（使用 unordered_set）：
 * 遍历链表，将每个节点存入 unordered_set 中，如果遇到已存在的节点，则说明有环。
 *
 * 时间复杂度：O(n)，其中 n 是链表长度
 * 空间复杂度：O(n)，需要存储所有节点
 */
// 定义链表节点结构
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) {}
};

bool hasCycle(ListNode *head) {
    unordered_set<ListNode*> seen;
    ListNode *current = head;

    while (current != nullptr) {
        if (seen.count(current)) {
            return true; // 发现环
        }
        seen.insert(current);
```

```
        current = current->next;
    }

    return false; // 没有环
}

/**
 * LeetCode 160. Intersection of Two Linked Lists (相交链表)
 *
 * 题目描述：
 * 给你两个单链表的头节点 headA 和 headB ，请你找出并返回两个单链表相交的起始节点。如果两个链表没有交点，返回 nullptr 。
 *
 * 解题思路 1（使用 unordered_set）：
 * 遍历链表 A，将每个节点存入 unordered_set 中，然后遍历链表 B，检查节点是否存在于 unordered_set 中。
 *
 * 时间复杂度：O(m+n)，其中 m 和 n 分别是两个链表的长度
 * 空间复杂度：O(m)，需要存储链表 A 的所有节点
 */
ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
    unordered_set<ListNode*> seen;
    ListNode *current = headA;

    // 将链表 A 的所有节点存入 unordered_set
    while (current != nullptr) {
        seen.insert(current);
        current = current->next;
    }

    // 遍历链表 B，查找交集
    current = headB;
    while (current != nullptr) {
        if (seen.count(current)) {
            return current; // 找到交点
        }
        current = current->next;
    }

    return nullptr; // 没有交点
}

int main() {
```

```
// 测试两数之和
cout << "测试两数之和:" << endl;
vector<int> nums1 = {2, 7, 11, 15};
int target = 9;
vector<int> result1 = twoSum(nums1, target);
cout << "[" << result1[0] << ", " << result1[1] << "]" << endl;

// 测试有效的字母异位词
cout << "测试有效的字母异位词:" << endl;
cout << isAnagram("anagram", "nagaram") << endl; // 1 (true)
cout << isAnagram("rat", "car") << endl; // 0 (false)

// 测试两个数组的交集
cout << "测试两个数组的交集:" << endl;
vector<int> nums2 = {1, 2, 2, 1};
vector<int> nums3 = {2, 2};
vector<int> result2 = intersection(nums2, nums3);
cout << "交集: [";
for (int i = 0; i < result2.size(); i++) {
    cout << result2[i];
    if (i < result2.size() - 1) {
        cout << ", ";
    }
}
cout << "]" << endl;

// 测试设计哈希集合
cout << "测试设计哈希集合:" << endl;
MyHashSet myHashSet;
myHashSet.add(1);
myHashSet.add(2);
cout << myHashSet.contains(1) << endl; // 1 (true)
cout << myHashSet.contains(3) << endl; // 0 (false)
myHashSet.add(2);
cout << myHashSet.contains(2) << endl; // 1 (true)
myHashSet.remove(2);
cout << myHashSet.contains(2) << endl; // 0 (false)

// 测试设计哈希映射
cout << "测试设计哈希映射:" << endl;
MyHashMap myHashMap;
myHashMap.put(1, 1);
myHashMap.put(2, 2);
```

```
cout << myHashMap.get(1) << endl; // 1
cout << myHashMap.get(3) << endl; // -1
myHashMap.put(2, 1);
cout << myHashMap.get(2) << endl; // 1
myHashMap.remove(2);
cout << myHashMap.get(2) << endl; // -1

// 测试最长连续序列
cout << "测试最长连续序列:" << endl;
vector<int> nums4 = {100, 4, 200, 1, 3, 2};
cout << longestConsecutive(nums4) << endl; // 4

// 测试字母异位词分组
cout << "测试字母异位词分组:" << endl;
vector<string> strs = {"eat", "tea", "tan", "ate", "nat", "bat"};
vector<vector<string>> groups = groupAnagrams(strs);
for (const auto& group : groups) {
    cout << "[";
    for (int i = 0; i < group.size(); i++) {
        cout << group[i];
        if (i < group.size() - 1) {
            cout << ", ";
        }
    }
    cout << "] ";
}
cout << endl;

// 测试前 K 个高频元素
cout << "测试前 K 个高频元素:" << endl;
vector<int> nums5 = {1, 1, 1, 2, 2, 3};
int k = 2;
vector<int> result3 = topKFrequent(nums5, k);
cout << "[";
for (int i = 0; i < result3.size(); i++) {
    cout << result3[i];
    if (i < result3.size() - 1) {
        cout << ", ";
    }
}
cout << "]" << endl;

return 0;
```

```
}   vector<string> strs = {"eat", "tea", "tan", "ate", "nat", "bat"};
    vector<vector<string>> groups = groupAnagrams(strs);
    for (const auto& group : groups) {
        cout << "[";
        for (int i = 0; i < group.size(); i++) {
            cout << group[i];
            if (i < group.size() - 1) {
                cout << ", ";
            }
        }
        cout << "] ";
    }
    cout << endl;

    // 测试前 K 个高频元素
    cout << "测试前 K 个高频元素:" << endl;
    vector<int> nums5 = {1, 1, 1, 2, 2, 3};
    int k = 2;
    vector<int> result3 = topKFrequent(nums5, k);
    cout << "[";
    for (int i = 0; i < result3.size(); i++) {
        cout << result3[i];
        if (i < result3.size() - 1) {
            cout << ", ";
        }
    }
    cout << "]" << endl;

    return 0;
}
```

==========

文件: Code01_HashSetAndHashMap.java

==========

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.PriorityQueue;
```

```
/**
 * HashSet 和 HashMap 相关题目与解析
 *
 * HashSet 基于 HashMap 实现，底层使用哈希表，查询时间复杂度 O(1)，元素无序
 * HashMap 基于哈希表实现，查询时间复杂度 O(1)，键值对无序
 *
 * 相关平台题目：
 * 1. LeetCode 1. Two Sum (两数之和) - https://leetcode.com/problems/two-sum/
 * 2. LeetCode 3. Longest Substring Without Repeating Characters (无重复字符的最长子串) - https://leetcode.com/problems/longest-substring-without-repeating-characters/
 * 3. LeetCode 36. Valid Sudoku (有效的数独) - https://leetcode.com/problems/valid-sudoku/
 * 4. LeetCode 136. Single Number (只出现一次的数字) - https://leetcode.com/problems/single-number/
 * 5. LeetCode 202. Happy Number (快乐数) - https://leetcode.com/problems/happy-number/
 * 6. LeetCode 217. Contains Duplicate (存在重复元素) - https://leetcode.com/problems/contains-duplicate/
 * 7. LeetCode 219. Contains Duplicate II (存在重复元素 II) - https://leetcode.com/problems/contains-duplicate-ii/
 * 8. LeetCode 242. Valid Anagram (有效的字母异位词) - https://leetcode.com/problems/valid-anagram/
 * 9. LeetCode 349. Intersection of Two Arrays (两个数组的交集) - https://leetcode.com/problems/intersection-of-two-arrays/
 * 10. LeetCode 350. Intersection of Two Arrays II (两个数组的交集 II) - https://leetcode.com/problems/intersection-of-two-arrays-ii/
 * 11. LeetCode 387. First Unique Character in a String (字符串中的第一个唯一字符) - https://leetcode.com/problems/first-unique-character-in-a-string/
 * 12. LeetCode 448. Find All Numbers Disappeared in an Array (找到所有数组中消失的数字) - https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/
 * 13. LeetCode 575. Distribute Candies (分糖果) - https://leetcode.com/problems/distribute-candies/
 * 14. LeetCode 811. Subdomain Visit Count (子域名访问计数) - https://leetcode.com/problems/subdomain-visit-count/
 * 15. LeetCode 705. Design HashSet (设计哈希集合) - https://leetcode.com/problems/design-hashset/
 * 16. LeetCode 706. Design HashMap (设计哈希映射) - https://leetcode.com/problems/design-hashmap/
 * 17. HackerRank Java Hashset (Java 哈希集) - https://www.hackerrank.com/challenges/java-hashset
 * 18. LeetCode 128. Longest Consecutive Sequence (最长连续序列) - https://leetcode.com/problems/longest-consecutive-sequence/
 * 19. LeetCode 49. Group Anagrams (字母异位词分组) - https://leetcode.com/problems/group-anagrams/
 * 20. LeetCode 347. Top K Frequent Elements (前 K 个高频元素) -
```

https://leetcode.com/problems/top-k-frequent-elements/
 * 21. LeetCode 3. Longest Substring Without Repeating Characters (无重复字符的最长子串) - https://leetcode.com/problems/longest-substring-without-repeating-characters/
 * 22. LeetCode 36. Valid Sudoku (有效的数独) - https://leetcode.com/problems/valid-sudoku/
 * 23. LeetCode 141. Linked List Cycle (环形链表) - https://leetcode.com/problems/linked-list-cycle/
 * 24. LeetCode 160. Intersection of Two Linked Lists (相交链表) - https://leetcode.com/problems/intersection-of-two-linked-lists/
 * 25. LintCode 547. Intersection of Two Arrays (两个数组的交集) - https://www.lintcode.com/problem/intersection-of-two-arrays/
 * 26. Codeforces 4C. Registration System (注册系统) - https://codeforces.com/problemset/problem/4/C
 * 27. AtCoder ABC 217 D - Cutting Woods (切割木材) - https://atcoder.jp/contests/abc217/tasks/abc217_d
 * 28. USACO Silver: Why Did the Cow Cross the Road (为什么奶牛要过马路) - http://www.usaco.org/index.php?page=viewproblem2&cpid=714
 * 29. 洛谷 P3374 【模板】树状数组 1 (模板树状数组 1) - https://www.luogu.com.cn/problem/P3374
 * 30. CodeChef STFOOD (街头食物) - https://www.codechef.com/problems/STFOOD
 * 31. SPOJ ANARC09A - Seinfeld (宋飞正传) - https://www.spoj.com/problems/ANARC09A/
 * 32. Project Euler Problem 1: Multiples of 3 and 5 (3 和 5 的倍数) - https://projecteuler.net/problem=1
 * 33. HackerRank Frequency Queries (频率查询) - https://www.hackerrank.com/challenges/frequency-queries
 * 34. 计蒜客 T1100: 计算 2 的 N 次方 (计算 2 的 N 次方) - https://www.jisuanke.com/t/T1100
 * 35. 杭电 OJ 1002: A + B Problem II (A+B 问题 II) - http://acm.hdu.edu.cn/showproblem.php?pid=1002
 * 36. 牛客网 剑指 Offer 03: 数组中重复的数字 (数组中重复的数字) - https://www.nowcoder.com/practice/623a5ac0ea5b4e5f95552655361ae0a8
 * 37. acwing 799. 最长连续不重复子序列 (最长连续不重复子序列) - https://www.acwing.com/problem/content/801/
 * 38. POJ 1002: 487-3279 (电话号码) - http://poj.org/problem?id=1002
 * 39. UVa OJ 100: The 3n + 1 problem (3n+1 问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=36
 * 40. Timus OJ 1001: Reverse Root (反转平方根) - https://acm.timus.ru/problem.aspx?space=1&num=1001
 * 41. Aizu OJ ALDS1_4_C: Dictionary (字典) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_4_C
 * 42. Comet OJ Contest #0: 热身赛 A. 签到题 (签到题) - https://cometoj.com/contest/0/problem/A
 * 43. MarsCode 火星编程竞赛: 字符串去重排序 (字符串去重排序) - https://www.marscode.cn/contest/1/problem/1001
 * 44. ZOJ 1001: A + B Problem (A+B 问题) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1001
 * 45. LOJ 100: 顺序的分数 (顺序的分数) - https://loj.ac/p/100

```
 * 46. 各大高校 OJ: 清华大学 OJ 1000: A+B Problem (A+B 问题) -
http://dsa.cs.tsinghua.edu.cn/oj/problem.shtml?id=1000
 */

public class Code01_HashSetAndHashMap {

    /**
     * LeetCode 1. Two Sum (两数之和)
     *
     * 题目描述：
     * 给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出和为目标值 target 的那两个整数，
     * 并返回它们的数组下标。你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。
     *
     * 示例：
     * 输入：nums = [2,7,11,15], target = 9
     * 输出：[0,1]
     * 解释：因为 nums[0] + nums[1] == 9 ，返回 [0, 1] 。
     *
     * 约束条件：
     * 2 <= nums.length <= 10^4
     * -10^9 <= nums[i] <= 10^9
     * -10^9 <= target <= 10^9
     * 只会存在一个有效答案
     *
     * 进阶：你可以想出一个时间复杂度小于 O(n^2) 的算法吗？
     *
     * 解题思路：
     * 使用 HashMap 存储已经遍历过的数字及其索引，对于每个数字，检查 target - 当前数字是否存在于 HashMap 中，
     * 如果存在则返回两个数字的索引，否则将当前数字和索引存入 HashMap。
     *
     * 时间复杂度：O(n)，其中 n 是数组长度，我们只需要遍历数组一次
     * 空间复杂度：O(n)，最坏情况下需要存储数组中所有元素及其索引
     *
     * @param nums 整数数组
     * @param target 目标值
     * @return 两个整数的数组下标
     */
    public static int[] twoSum(int[] nums, int target) {
        // 创建 HashMap 存储数字和其索引
        HashMap<Integer, Integer> map = new HashMap<>();
```

```
        // 遍历数组
        for (int i = 0; i < nums.length; i++) {
            // 计算需要找到的另一个数字
            int complement = target - nums[i];

            // 检查该数字是否已存在于 HashMap 中
            if (map.containsKey(complement)) {
                // 如果存在，返回两个数字的索引
                return new int[] { map.get(complement), i };
            }

            // 将当前数字和索引存入 HashMap
            map.put(nums[i], i);
        }

        // 根据题目保证总会有一个解，这里仅为避免编译错误
        throw new IllegalArgumentException("No two sum solution");
    }

    /**
     * LeetCode 242. Valid Anagram (有效的字母异位词)
     *
     * 题目描述：
     * 给定两个字符串 s 和 t ，编写一个函数来判断 t 是否是 s 的字母异位词。
     * 注意：若 s 和 t 中每个字符出现的次数都相同，则称 s 和 t 互为字母异位词。
     *
     * 示例：
     * 输入: s = "anagram", t = "nagaram"
     * 输出: true
     *
     * 输入: s = "rat", t = "car"
     * 输出: false
     *
     * 约束条件：
     * 1 <= s.length, t.length <= 5 * 10^4
     * s 和 t 仅包含小写字母
     *
     * 进阶: 如果输入字符串包含 unicode 字符怎么办？你能否调整你的解法来应对这种情况？
     *
     * 解题思路：
     * 使用 HashMap 记录字符串 s 中每个字符出现的次数，然后遍历字符串 t，对每个字符在 HashMap 中对应的计数减 1，
```

```
     * 如果某个字符不存在或计数小于 0，则返回 false。最后检查 HashMap 中是否所有字符计数都为 0。
     *
     * 时间复杂度：O(n)，其中 n 是字符串长度，需要遍历两个字符串
     * 空间复杂度：O(1)，因为字符集大小固定（小写字母 26 个），所以空间复杂度是常数
     *
     * @param s 字符串 s
     * @param t 字符串 t
     * @return 如果 t 是 s 的字母异位词返回 true，否则返回 false
     */
    public static boolean isAnagram(String s, String t) {
        // 如果两个字符串长度不同，肯定不是字母异位词
        if (s.length() != t.length()) {
            return false;
        }

        // 创建 HashMap 记录字符出现次数
        HashMap<Character, Integer> charCount = new HashMap<>();

        // 遍历字符串 s，统计每个字符出现次数
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            charCount.put(c, charCount.getOrDefault(c, 0) + 1);
        }

        // 遍历字符串 t，减少对应字符计数
        for (int i = 0; i < t.length(); i++) {
            char c = t.charAt(i);
            // 如果字符不存在或计数已为 0，返回 false
            if (!charCount.containsKey(c) || charCount.get(c) == 0) {
                return false;
            }
            // 减少字符计数
            charCount.put(c, charCount.get(c) - 1);
        }

        // 检查所有字符计数是否为 0
        for (int count : charCount.values()) {
            if (count != 0) {
                return false;
            }
        }

        return true;
```

```
}

/**
 * LeetCode 349. Intersection of Two Arrays (两个数组的交集)
 *
 * 题目描述：
 * 给定两个数组 nums1 和 nums2 ，返回它们的交集。输出结果中的每个元素一定是唯一的。
 * 我们可以不考虑输出结果的顺序。
 *
 * 示例：
 * 输入：nums1 = [1,2,2,1], nums2 = [2,2]
 * 输出：[2]
 *
 * 输入：nums1 = [4,9,5], nums2 = [9,4,9,8,4]
 * 输出：[9,4]
 * 解释：[4,9] 也是可通过的
 *
 * 约束条件：
 * 1 <= nums1.length, nums2.length <= 1000
 * 0 <= nums1[i], nums2[i] <= 1000
 *
 * 解题思路：
 * 使用 HashSet 存储 nums1 中的所有元素，然后遍历 nums2，检查每个元素是否存在于 HashSet 中，
 * 如果存在则加入结果集，并从 HashSet 中移除该元素以避免重复。
 *
 * 时间复杂度：O(m+n)，其中 m 和 n 分别是两个数组的长度
 * 空间复杂度：O(m)，用于存储 nums1 中的元素
 *
 * @param nums1 数组 1
 * @param nums2 数组 2
 * @return 交集数组
 */
public static int[] intersection(int[] nums1, int[] nums2) {
    // 使用 HashSet 存储 nums1 中的元素
    HashSet<Integer> set1 = new HashSet<>();
    for (int num : nums1) {
        set1.add(num);
    }

    // 使用 HashSet 存储交集结果，自动去重
    HashSet<Integer> intersection = new HashSet<>();

    // 遍历 nums2，查找交集
```

```
        for (int num : nums2) {
            if (set1.contains(num)) {
                intersection.add(num);
            }
        }

        // 将结果转换为数组
        int[] result = new int[intersection.size()];
        int index = 0;
        for (int num : intersection) {
            result[index++] = num;
        }

        return result;
    }

    /**
     * LeetCode 705. Design HashSet (设计哈希集合)
     *
     * 题目描述：
     * 不使用任何内建的哈希表库设计一个哈希集合（HashSet）。
     * 实现 MyHashSet 类：
     * void add(key) 向哈希集合中插入值 key 。
     * bool contains(key) 返回哈希集合中是否存在这个值 key 。
     * void remove(key) 将给定值 key 从哈希集合中删除。如果哈希集合中没有这个值，什么也不做。
     *
     * 示例：
     * 输入：
     * ["MyHashSet", "add", "add", "contains", "contains", "add", "contains", "remove",
"contains"]
     * [[], [1], [2], [1], [3], [2], [2], [2], [2]]
     * 输出：
     * [null, null, null, true, false, null, true, null, false]
     *
     * 约束条件：
     * 0 <= key <= 10^6
     * 最多调用 10^4 次 add、remove 和 contains
     *
     * 解题思路：
     * 使用链地址法实现哈希表，创建一个固定大小的数组，每个数组元素是一个链表。
     * 当发生哈希冲突时，将元素添加到对应位置的链表中。
     *
     * 时间复杂度：O(n/b)，其中 n 是元素个数，b 是桶数（在实际实现中我们使用 10000 作为桶数）
```

```
 * 空间复杂度：O(n)，存储所有元素
 */
static class MyHashSet {
    private static final int BASE = 10000;
    private LinkedList<Integer>[] data;

    /** Initialize your data structure here. */
    public MyHashSet() {
        data = new LinkedList[BASE];
        for (int i = 0; i < BASE; ++i) {
            data[i] = new LinkedList<Integer>();
        }
    }

    public void add(int key) {
        int h = hash(key);
        Iterator<Integer> iterator = data[h].iterator();
        while (iterator.hasNext()) {
            Integer element = iterator.next();
            if (element == key) {
                return;
            }
        }
        data[h].offerLast(key);
    }

    public void remove(int key) {
        int h = hash(key);
        Iterator<Integer> iterator = data[h].iterator();
        while (iterator.hasNext()) {
            Integer element = iterator.next();
            if (element == key) {
                iterator.remove();
                return;
            }
        }
    }

    /** Returns true if this set contains the specified element */
    public boolean contains(int key) {
        int h = hash(key);
        Iterator<Integer> iterator = data[h].iterator();
        while (iterator.hasNext()) {
```

```
                Integer element = iterator.next();
                if (element == key) {
                    return true;
                }
            }
            return false;
        }

        private static int hash(int key) {
            return key % BASE;
        }
    }

    /**
     * LeetCode 706. Design HashMap (设计哈希映射)
     *
     * 题目描述：
     * 不使用任何内建的哈希表库设计一个哈希映射（HashMap）。
     * 实现 MyHashMap 类：
     * MyHashMap() 用空映射初始化对象
     * void put(int key, int value) 向 HashMap 插入一个键值对 (key, value) 。如果 key 已经存在于映射中，则更新其对应的值 value 。
     * int get(int key) 返回特定的 key 所映射的 value ；如果映射中不包含 key 的映射，返回 -1 。
     * void remove(key) 如果映射中存在 key 的映射，则移除 key 和它所对应的 value 。
     *
     * 示例：
     * 输入：
     * ["MyHashMap", "put", "put", "get", "get", "put", "get", "remove", "get"]
     * [[], [1, 1], [2, 2], [1], [3], [2, 1], [2], [2], [2]]
     * 输出：
     * [null, null, null, 1, -1, null, 1, null, -1]
     *
     * 约束条件：
     * 0 <= key, value <= 10^6
     * 最多调用 10^4 次 put、get 和 remove 方法
     *
     * 解题思路：
     * 使用链地址法实现哈希表，创建一个固定大小的数组，每个数组元素是一个链表。
     * 每个链表节点存储键值对，当发生哈希冲突时，将节点添加到对应位置的链表中。
     *
     * 时间复杂度：O(n/b)，其中 n 是元素个数，b 是桶数（在实际实现中我们使用 10000 作为桶数）
     * 空间复杂度：O(n)，存储所有键值对
     */
```

```
    static class MyHashMap {
        private static final int BASE = 10000;
        private LinkedList<Pair>[] data;

        /** Initialize your data structure here. */
        public MyHashMap() {
            data = new LinkedList[BASE];
            for (int i = 0; i < BASE; ++i) {
                data[i] = new LinkedList<Pair>();
            }
        }

        /** value will always be non-negative. */
        public void put(int key, int value) {
            int h = hash(key);
            Iterator<Pair> iterator = data[h].iterator();
            while (iterator.hasNext()) {
                Pair pair = iterator.next();
                if (pair.getKey() == key) {
                    pair.setValue(value);
                    return;
                }
            }
            data[h].offerLast(new Pair(key, value));
        }

        /** Returns the value to which the specified key is mapped, or -1 if this map contains no
mapping for the key */
        public int get(int key) {
            int h = hash(key);
            Iterator<Pair> iterator = data[h].iterator();
            while (iterator.hasNext()) {
                Pair pair = iterator.next();
                if (pair.getKey() == key) {
                    return pair.getValue();
                }
            }
            return -1;
        }

        /** Removes the mapping of the specified value key if this map contains a mapping for the
key */
        public void remove(int key) {
```

```
        int h = hash(key);
        Iterator<Pair> iterator = data[h].iterator();
        while (iterator.hasNext()) {
            Pair pair = iterator.next();
            if (pair.getKey() == key) {
                iterator.remove();
                return;
            }
        }
    }

    private static int hash(int key) {
        return key % BASE;
    }

    private class Pair {
        private int key;
        private int value;

        public Pair(int key, int value) {
            this.key = key;
            this.value = value;
        }

        public int getKey() {
            return key;
        }

        public int getValue() {
            return value;
        }

        public void setValue(int value) {
            this.value = value;
        }
    }
}

/**
 * HackerRank Java Hashset (Java 哈希集)
 *
 * 题目描述：
 * 给定 n 对字符串，每对字符串由两个部分组成。找出有多少对独特的字符串对。
```

```
     *
     * 示例：
     * 输入：
     * 5
     * john tom
     * john mary
     * john tom
     * mary anna
     * mary anna
     * 输出：
     * 3
     *
     * 解题思路：
     * 使用 HashSet 存储字符串对，由于 HashSet 的特性，重复的元素只会存储一次。
     * 最后返回 HashSet 的大小即可。
     *
     * 时间复杂度：O(n)，其中 n 是字符串对的数量
     * 空间复杂度：O(n)，用于存储所有独特的字符串对
     *
     * @param pairs 字符串对数组
     * @return 独特字符串对的数量
     */
    public static int countUniquePairs(String[][] pairs) {
        HashSet<String> set = new HashSet<>();
        for (String[] pair : pairs) {
            // 将两个字符串连接作为唯一标识
            set.add(pair[0] + " " + pair[1]);
        }
        return set.size();
    }

    /**
     * LeetCode 128. Longest Consecutive Sequence (最长连续序列)
     *
     * 题目描述：
     * 给定一个未排序的整数数组 nums ，找出数字连续的最长序列（不要求序列元素在原数组中连续）的长度。
     * 请你设计并实现时间复杂度为 O(n) 的算法解决此问题。
     *
     * 示例：
     * 输入：nums = [100,4,200,1,3,2]
     * 输出：4
     * 解释：最长数字连续序列是 [1, 2, 3, 4]。它的长度为 4。
```

```
     *
     * 输入：nums = [0,3,7,2,5,8,4,6,0,1]
     * 输出：9
     *
     * 约束条件：
     * 0 <= nums.length <= 10^5
     * -10^9 <= nums[i] <= 10^9
     *
     * 解题思路：
     * 使用 HashSet 存储所有数字，然后对于每个数字，如果它是某个序列的开始（即 num-1 不在 HashSet 中），
     * 则从该数字开始向后查找连续的数字，计算序列长度。
     *
     * 时间复杂度：O(n)，虽然有嵌套循环，但每个元素最多被访问两次
     * 空间复杂度：O(n)，用于存储 HashSet
     *
     * @param nums 整数数组
     * @return 最长连续序列的长度
     */
    public static int longestConsecutive(int[] nums) {
        HashSet<Integer> numSet = new HashSet<>();
        for (int num : nums) {
            numSet.add(num);
        }

        int longestStreak = 0;

        for (int num : numSet) {
            // 只有当 num-1 不存在时，num 才是一个序列的开始
            if (!numSet.contains(num - 1)) {
                int currentNum = num;
                int currentStreak = 1;

                // 向后查找连续的数字
                while (numSet.contains(currentNum + 1)) {
                    currentNum += 1;
                    currentStreak += 1;
                }

                longestStreak = Math.max(longestStreak, currentStreak);
            }
        }
```

```
    return longestStreak;
}

/**
 * LeetCode 49. Group Anagrams (字母异位词分组)
 *
 * 题目描述：
 * 给你一个字符串数组，请你将字母异位词组合在一起。可以按任意顺序返回结果列表。
 * 字母异位词是由重新排列源单词的所有字母得到的一个新单词。
 *
 * 示例：
 * 输入: strs = ["eat", "tea", "tan", "ate", "nat", "bat"]
 * 输出: [["bat"],["nat","tan"],["ate","eat","tea"]]
 *
 * 输入: strs = [""]
 * 输出: [[""]]
 *
 * 输入: strs = ["a"]
 * 输出: [["a"]]
 *
 * 约束条件：
 * 1 <= strs.length <= 10^4
 * 0 <= strs[i].length <= 100
 * strs[i] 仅包含小写字母
 *
 * 解题思路：
 * 使用 HashMap 存储分组结果，键为排序后的字符串，值为该组的所有字符串。
 * 对于每个字符串，将其字符排序后作为键，将原字符串添加到对应的列表中。
 *
 * 时间复杂度：O(N*K*logK)，其中 N 是字符串数组的长度，K 是字符串的最大长度
 * 空间复杂度：O(N*K)，用于存储所有字符串
 *
 * @param strs 字符串数组
 * @return 字母异位词分组结果
 */
public static List<List<String>> groupAnagrams(String[] strs) {
    HashMap<String, List<String>> map = new HashMap<>();

    for (String str : strs) {
        // 将字符串转换为字符数组并排序
        char[] chars = str.toCharArray();
        Arrays.sort(chars);
        String key = new String(chars);
```

```
            // 如果键不存在，创建新的列表
            if (!map.containsKey(key)) {
                map.put(key, new ArrayList<>());
            }
            // 将原字符串添加到对应的列表中
            map.get(key).add(str);
        }

        return new ArrayList<>(map.values());
    }

    /**
     * LeetCode 347. Top K Frequent Elements (前 K 个高频元素)
     *
     * 题目描述：
     * 给你一个整数数组 nums 和一个整数 k ，请你返回其中出现频率前 k 高的元素。你可以按任意顺序返回答案。
     *
     * 示例：
     * 输入: nums = [1,1,1,2,2,3], k = 2
     * 输出: [1,2]
     *
     * 输入: nums = [1], k = 1
     * 输出: [1]
     *
     * 约束条件：
     * 1 <= nums.length <= 10^5
     * k 的取值范围是 [1, 数组中不相同的元素的个数]
     * 题目数据保证答案唯一
     *
     * 解题思路：
     * 1. 使用 HashMap 统计每个元素的频率
     * 2. 使用最小堆维护前 k 个高频元素
     * 3. 遍历 HashMap，维护堆的大小为 k
     * 4. 最后从堆中取出所有元素
     *
     * 时间复杂度：O(N*logK)，其中 N 是数组长度
     * 空间复杂度：O(N)，用于存储 HashMap 和堆
     *
     * @param nums 整数数组
     * @param k 需要返回的元素个数
     * @return 前 k 个高频元素
```

```
     */
    public static int[] topKFrequent(int[] nums, int k) {
        // 统计每个元素的频率
        HashMap<Integer, Integer> freqMap = new HashMap<>();
        for (int num : nums) {
            freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);
        }

        // 使用最小堆维护前 k 个高频元素
        PriorityQueue<Integer> heap = new PriorityQueue<>((a, b) -> freqMap.get(a) -
freqMap.get(b));

        // 遍历频率映射，维护堆的大小为 k
        for (int key : freqMap.keySet()) {
            heap.add(key);
            if (heap.size() > k) {
                heap.poll();
            }
        }

        // 从堆中取出所有元素
        int[] result = new int[k];
        for (int i = 0; i < k; i++) {
            result[i] = heap.poll();
        }

        return result;
    }

    /**
     * LeetCode 219. Contains Duplicate II (存在重复元素 II)
     *
     * 题目描述：
     * 给定一个整数数组和一个整数 k，判断数组中是否存在两个不同的索引 i 和 j，
     * 使得 nums[i] == nums[j]，并且 i 和 j 的差的绝对值至多为 k。
     *
     * 示例：
     * 输入: nums = [1,2,3,1], k = 3
     * 输出: true
     *
     * 输入: nums = [1,0,1,1], k = 1
     * 输出: true
     *
```

```
 * 输入: nums = [1,2,3,1,2,3], k = 2
 * 输出: false
 *
 * 约束条件：
 * 1 <= nums.length <= 10^5
 * -10^9 <= nums[i] <= 10^9
 * 0 <= k <= 10^5
 *
 * 解题思路：
 * 使用 HashMap 存储每个元素最后一次出现的索引，遍历数组时，检查当前元素是否在 HashMap 中存在，
 * 如果存在且当前索引与存储的索引之差的绝对值小于等于 k，则返回 true；
 * 否则更新 HashMap 中该元素的索引为当前索引。
 *
 * 时间复杂度：O(n)，其中 n 是数组长度，我们只需要遍历数组一次
 * 空间复杂度：O(min(n,k))，最坏情况下需要存储 min(n,k)个元素
 */
public static boolean containsNearbyDuplicate(int[] nums, int k) {
    HashMap<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < nums.length; i++) {
        if (map.containsKey(nums[i])) {
            int prevIndex = map.get(nums[i]);
            if (i - prevIndex <= k) {
                return true;
            }
        }
        map.put(nums[i], i);
    }
    return false;
}

/**
 * LeetCode 3. Longest Substring Without Repeating Characters (无重复字符的最长子串)
 *
 * 题目描述：
 * 给定一个字符串 s，请你找出其中不含有重复字符的最长子串的长度。
 *
 * 示例：
 * 输入: s = "abcabcbb"
 * 输出: 3
 * 解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。
 *
 * 输入: s = "bbbbb"
 * 输出: 1
```

```
 * 解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。
 *
 * 输入: s = "pwwkew"
 * 输出: 3
 * 解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
 * 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。
 *
 * 约束条件：
 * 0 <= s.length <= 5 * 10^4
 * s 由英文字母、数字、符号和空格组成
 *
 * 解题思路：
 * 使用滑动窗口和 HashMap，HashMap 记录每个字符最后一次出现的位置。
 * 维护一个左边界 left，当遇到重复字符时，将 left 更新为重复字符上一次出现位置的下一个位置。
 * 计算当前窗口长度 i-left+1，并更新最大长度。
 *
 * 时间复杂度：O(n)，其中 n 是字符串长度
 * 空间复杂度：O(min(n, m))，其中 m 是字符集大小
 */
public static int lengthOfLongestSubstring(String s) {
    HashMap<Character, Integer> map = new HashMap<>();
    int maxLength = 0;
    int left = 0;

    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        // 如果字符已存在且在当前窗口内，更新左边界
        if (map.containsKey(c) && map.get(c) >= left) {
            left = map.get(c) + 1;
        }
        // 更新字符位置
        map.put(c, i);
        // 更新最大长度
        maxLength = Math.max(maxLength, i - left + 1);
    }

    return maxLength;
}

/**
 * LeetCode 36. Valid Sudoku (有效的数独)
 *
 * 题目描述：
```

```
 * 请你判断一个 9 x 9 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。
 * 1. 数字 1-9 在每一行只能出现一次。
 * 2. 数字 1-9 在每一列只能出现一次。
 * 3. 数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。
 *
 * 注意：
 * - 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。
 * - 空白格用 '.' 表示。
 *
 * 示例：
 * 输入：board =
 * [
 *   ["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
 *
 * 约束条件：
 * board.length == 9
 * board[i].length == 9
 * board[i][j] 是一位数字或者 '.'
 *
 * 解题思路：
 * 使用三个 HashSet 数组分别记录每一行、每一列和每一个 3x3 宫格中出现过的数字。
 * 遍历数独，对于每个非空白字符，检查是否已经在对应的行、列或宫格中出现过。
 * 宫格索引可以通过公式：boxIndex = (row / 3) * 3 + (col / 3) 计算得到。
 *
 * 时间复杂度：O(1)，因为数独大小固定为 9x9
 * 空间复杂度：O(1)，固定大小的 HashSet 数组
 */
public static boolean isValidSudoku(char[][] board) {
    HashSet<Character>[] rows = new HashSet[9];
    HashSet<Character>[] cols = new HashSet[9];
    HashSet<Character>[] boxes = new HashSet[9];
```

```
        // 初始化所有 HashSet
        for (int i = 0; i < 9; i++) {
            rows[i] = new HashSet<>();
            cols[i] = new HashSet<>();
            boxes[i] = new HashSet<>();
        }

        // 遍历数独
        for (int row = 0; row < 9; row++) {
            for (int col = 0; col < 9; col++) {
                char c = board[row][col];
                // 跳过空白格
                if (c == '.') {
                    continue;
                }

                // 计算宫格索引
                int boxIndex = (row / 3) * 3 + (col / 3);

                // 检查是否重复
                if (rows[row].contains(c) || cols[col].contains(c) || boxes[boxIndex].contains(c))
{
                    return false;
                }

                // 添加到对应的 HashSet 中
                rows[row].add(c);
                cols[col].add(c);
                boxes[boxIndex].add(c);
            }
        }

        return true;
    }

    /**
     * LeetCode 141. Linked List Cycle (环形链表)
     *
     * 题目描述：
     * 给定一个链表，判断链表中是否有环。
     * 如果链表中存在环，则返回 true 。 否则，返回 false 。
     *
     * 进阶：
```

```
     * 你能否不使用额外空间解决此题？
     *
     * 解题思路 1（使用 HashSet）：
     * 遍历链表，将每个节点存入 HashSet 中，如果遇到已存在的节点，则说明有环。
     *
     * 时间复杂度：O(n)，其中 n 是链表长度
     * 空间复杂度：O(n)，需要存储所有节点
     *
     * 注意：为了简化实现，这里定义一个简单的 ListNode 类
     */
    public static class ListNode {
        int val;
        ListNode next;
        ListNode(int x) {
            val = x;
            next = null;
        }
    }

    public static boolean hasCycle(ListNode head) {
        HashSet<ListNode> seen = new HashSet<>();
        ListNode current = head;

        while (current != null) {
            if (seen.contains(current)) {
                return true; // 发现环
            }
            seen.add(current);
            current = current.next;
        }

        return false; // 没有环
    }

    /**
     * LeetCode 160. Intersection of Two Linked Lists (相交链表)
     *
     * 题目描述：
     * 给你两个单链表的头节点 headA 和 headB ，请你找出并返回两个单链表相交的起始节点。如果两个链表没有交点，返回 null 。
     *
     * 解题思路 1（使用 HashSet）：
     * 遍历链表 A，将每个节点存入 HashSet 中，然后遍历链表 B，检查节点是否存在于 HashSet 中。
```

```
     *
     * 时间复杂度：O(m+n)，其中 m 和 n 分别是两个链表的长度
     * 空间复杂度：O(m)，需要存储链表 A 的所有节点
     */
    public static ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        HashSet<ListNode> seen = new HashSet<>();
        ListNode current = headA;

        // 将链表 A 的所有节点存入 HashSet
        while (current != null) {
            seen.add(current);
            current = current.next;
        }

        // 遍历链表 B，查找交集
        current = headB;
        while (current != null) {
            if (seen.contains(current)) {
                return current; // 找到交点
            }
            current = current.next;
        }

        return null; // 没有交点
    }

    public static void main(String[] args) {
        // Integer、Long、Double、Float
        // Byte、Short、Character、Boolean
        // String 等都有这个特征
        String str1 = new String("Hello");
        String str2 = new String("Hello");
        // false，因为不同的内存地址
        System.out.println(str1 == str2);
        // true，因为它们的值是相同的
        System.out.println(str1.equals(str2));

        HashSet<String> set = new HashSet<>();
        set.add(str1);
        System.out.println(set.contains("Hello"));
        System.out.println(set.contains(str2));
        set.add(str2);
        System.out.println(set.size());
```

```
set.remove(str1);
set.clear();
System.out.println(set.isEmpty());

System.out.println("===========");

HashMap<String, String> map1 = new HashMap<>();
map1.put(str1, "World");
System.out.println(map1.containsKey("Hello"));
System.out.println(map1.containsKey(str2));
System.out.println(map1.get(str2));
System.out.println(map1.get("你好") == null);
map1.remove("Hello");
System.out.println(map1.size());
map1.clear();
System.out.println(map1.isEmpty());

System.out.println("===========");

// 一般在笔试中，未必需要申请哈希表
HashMap<Integer, Integer> map2 = new HashMap<>();
map2.put(56, 7285);
map2.put(34, 3671263);
map2.put(17, 716311);
map2.put(24, 1263161);
// 上面的 map2 行为，可以被如下数组的行为替代
int[] arr = new int[100];
arr[56] = 7285;
arr[34] = 3671263;
arr[17] = 716311;
arr[24] = 1263161;
// 哈希表的增、删、改、查，都可以被数组替代，前提是 key 的范围是固定的、可控的
System.out.println("在笔试场合中哈希表往往会被数组替代");

System.out.println("===========");

Student s1 = new Student(17, "张三");
Student s2 = new Student(17, "张三");
HashMap<Student, String> map3 = new HashMap<>();
map3.put(s1, "这是张三");
System.out.println(map3.containsKey(s1));
System.out.println(map3.containsKey(s2));
map3.put(s2, "这是另一个张三");
```

```
System.out.println(map3.size());
System.out.println(map3.get(s1));
System.out.println(map3.get(s2));

// 测试添加的题目
System.out.println("===========");
System.out.println("测试两数之和:");
int[] nums = {2, 7, 11, 15};
int target = 9;
int[] result = twoSum(nums, target);
System.out.println("[" + result[0] + ", " + result[1] + "]");

System.out.println("测试有效的字母异位词:");
System.out.println(isAnagram("anagram", "nagaram")); // true
System.out.println(isAnagram("rat", "car")); // false

System.out.println("测试两个数组的交集:");
int[] nums1 = {1, 2, 2, 1};
int[] nums2 = {2, 2};
int[] intersectResult = intersection(nums1, nums2);
System.out.print("交集: [");
for (int i = 0; i < intersectResult.length; i++) {
    System.out.print(intersectResult[i]);
    if (i < intersectResult.length - 1) {
        System.out.print(", ");
    }
}
System.out.println("]");

// 测试设计哈希集合
System.out.println("测试设计哈希集合:");
MyHashSet myHashSet = new MyHashSet();
myHashSet.add(1);
myHashSet.add(2);
System.out.println(myHashSet.contains(1)); // true
System.out.println(myHashSet.contains(3)); // false
myHashSet.add(2);
System.out.println(myHashSet.contains(2)); // true
myHashSet.remove(2);
System.out.println(myHashSet.contains(2)); // false

// 测试设计哈希映射
System.out.println("测试设计哈希映射:");
```

```
        MyHashMap myHashMap = new MyHashMap();
        myHashMap.put(1, 1);
        myHashMap.put(2, 2);
        System.out.println(myHashMap.get(1)); // 1
        System.out.println(myHashMap.get(3)); // -1
        myHashMap.put(2, 1);
        System.out.println(myHashMap.get(2)); // 1
        myHashMap.remove(2);
        System.out.println(myHashMap.get(2)); // -1

        // 测试 HackerRank Java Hashset
        System.out.println("测试 HackerRank Java Hashset:");
        String[][] pairs = {{"john", "tom"}, {"john", "mary"}, {"john", "tom"}, {"mary", "anna"},
{"mary", "anna"}};
        System.out.println(countUniquePairs(pairs)); // 3

        // 测试最长连续序列
        System.out.println("测试最长连续序列:");
        int[] nums3 = {100, 4, 200, 1, 3, 2};
        System.out.println(longestConsecutive(nums3)); // 4

        // 测试字母异位词分组
        System.out.println("测试字母异位词分组:");
        String[] strs = {"eat", "tea", "tan", "ate", "nat", "bat"};
        List<List<String>> groups = groupAnagrams(strs);
        System.out.println(groups);

        // 测试前 K 个高频元素
        System.out.println("测试前 K 个高频元素:");
        int[] nums4 = {1, 1, 1, 2, 2, 3};
        int k = 2;
        int[] topK = topKFrequent(nums4, k);
        System.out.println(Arrays.toString(topK)); // [1, 2]
    }

    public static class Student {
        public int age;
        public String name;

        public Student(int a, String b) {
            age = a;
            name = b;
        }
```

```
    }
}
```

==========

文件: Code01_HashSetAndHashMap.py
==========

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
HashSet 和 HashMap 相关题目与解析

Python 中使用 set 和 dict（字典）
set 基于哈希表，查询时间复杂度 O(1)，元素无序
dict 基于哈希表，查询时间复杂度 O(1)，键值对无序

相关平台题目：
1. LeetCode 1. Two Sum (两数之和) - https://leetcode.com/problems/two-sum/
2. LeetCode 3. Longest Substring Without Repeating Characters (无重复字符的最长子串) -
https://leetcode.com/problems/longest-substring-without-repeating-characters/
3. LeetCode 36. Valid Sudoku (有效的数独) - https://leetcode.com/problems/valid-sudoku/
4. LeetCode 136. Single Number (只出现一次的数字) - https://leetcode.com/problems/single-number/
5. LeetCode 202. Happy Number (快乐数) - https://leetcode.com/problems/happy-number/
6. LeetCode 217. Contains Duplicate (存在重复元素) - https://leetcode.com/problems/contains-
duplicate/
7. LeetCode 219. Contains Duplicate II (存在重复元素 II) -
https://leetcode.com/problems/contains-duplicate-ii/
8. LeetCode 242. Valid Anagram (有效的字母异位词) - https://leetcode.com/problems/valid-anagram/
9. LeetCode 349. Intersection of Two Arrays (两个数组的交集) -
https://leetcode.com/problems/intersection-of-two-arrays/
10. LeetCode 350. Intersection of Two Arrays II (两个数组的交集 II) -
https://leetcode.com/problems/intersection-of-two-arrays-ii/
11. LeetCode 387. First Unique Character in a String (字符串中的第一个唯一字符) -
https://leetcode.com/problems/first-unique-character-in-a-string/
12. LeetCode 448. Find All Numbers Disappeared in an Array (找到所有数组中消失的数字) -
https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/
13. LeetCode 575. Distribute Candies (分糖果) - https://leetcode.com/problems/distribute-candies/
14. LeetCode 811. Subdomain Visit Count (子域名访问计数) -
https://leetcode.com/problems/subdomain-visit-count/
15. LeetCode 705. Design HashSet (设计哈希集合) - https://leetcode.com/problems/design-hashset/
16. LeetCode 706. Design HashMap (设计哈希映射) - https://leetcode.com/problems/design-hashmap/
17. HackerRank Java Hashset (Java 哈希集) - https://www.hackerrank.com/challenges/java-hashset
```

18. LeetCode 128. Longest Consecutive Sequence (最长连续序列) - https://leetcode.com/problems/longest-consecutive-sequence/
19. LeetCode 49. Group Anagrams (字母异位词分组) - https://leetcode.com/problems/group-anagrams/
20. LeetCode 347. Top K Frequent Elements (前 K 个高频元素) - https://leetcode.com/problems/top-k-frequent-elements/
21. LeetCode 3. Longest Substring Without Repeating Characters (无重复字符的最长子串) - https://leetcode.com/problems/longest-substring-without-repeating-characters/
22. LeetCode 36. Valid Sudoku (有效的数独) - https://leetcode.com/problems/valid-sudoku/
23. LeetCode 141. Linked List Cycle (环形链表) - https://leetcode.com/problems/linked-list-cycle/
24. LeetCode 160. Intersection of Two Linked Lists (相交链表) - https://leetcode.com/problems/intersection-of-two-linked-lists/
25. LintCode 547. Intersection of Two Arrays (两个数组的交集) - https://www.lintcode.com/problem/intersection-of-two-arrays/
26. Codeforces 4C. Registration System (注册系统) - https://codeforces.com/problemset/problem/4/C
27. AtCoder ABC 217 D - Cutting Woods (切割木材) - https://atcoder.jp/contests/abc217/tasks/abc217_d
28. USACO Silver: Why Did the Cow Cross the Road (为什么奶牛要过马路) - http://www.usaco.org/index.php?page=viewproblem2&cpid=714
29. 洛谷 P3374 【模板】树状数组 1 (模板树状数组 1) - https://www.luogu.com.cn/problem/P3374
30. CodeChef STFOOD (街头食物) - https://www.codechef.com/problems/STFOOD
31. SPOJ ANARC09A - Seinfeld (宋飞正传) - https://www.spoj.com/problems/ANARC09A/
32. Project Euler Problem 1: Multiples of 3 and 5 (3 和 5 的倍数) - https://projecteuler.net/problem=1
33. HackerRank Frequency Queries (频率查询) - https://www.hackerrank.com/challenges/frequency-queries
34. 计蒜客 T1100: 计算 2 的 N 次方 (计算 2 的 N 次方) - https://www.jisuanke.com/t/T1100
35. 杭电 OJ 1002: A + B Problem II (A+B 问题 II) - http://acm.hdu.edu.cn/showproblem.php?pid=1002
36. 牛客网 剑指 Offer 03: 数组中重复的数字 (数组中重复的数字) - https://www.nowcoder.com/practice/623a5ac0ea5b4e5f95552655361ae0a8
37. acwing 799. 最长连续不重复子序列 (最长连续不重复子序列) - https://www.acwing.com/problem/content/801/
38. POJ 1002: 487-3279 (电话号码) - http://poj.org/problem?id=1002
39. UVa OJ 100: The 3n + 1 problem (3n+1 问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=36
40. Timus OJ 1001: Reverse Root (反转平方根) - https://acm.timus.ru/problem.aspx?space=1&num=1001
41. Aizu OJ ALDS1_4_C: Dictionary (字典) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_4_C
42. Comet OJ Contest #0: 热身赛 A. 签到题 (签到题) - https://cometoj.com/contest/0/problem/A
43. MarsCode 火星编程竞赛: 字符串去重排序 (字符串去重排序) - https://www.marscode.cn/contest/1/problem/1001
44. ZOJ 1001: A + B Problem (A+B 问题) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1001
45. LOJ 100: 顺序的分数 (顺序的分数) - https://loj.ac/p/100

```
46. 各大高校 OJ: 清华大学 OJ 1000: A+B Problem (A+B 问题) -
http://dsa.cs.tsinghua.edu.cn/oj/problem.shtml?id=1000
"""

from collections import defaultdict
import heapq

# LeetCode 1. Two Sum (两数之和)
def two_sum(nums, target):
    """
    题目描述：
    给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出和为目标值 target 的那两个整数，
    并返回它们的数组下标。你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

    示例：
    输入：nums = [2,7,11,15], target = 9
    输出：[0,1]
    解释：因为 nums[0] + nums[1] == 9 ，返回 [0, 1] 。

    约束条件：
    2 <= nums.length <= 10^4
    -10^9 <= nums[i] <= 10^9
    -10^9 <= target <= 10^9
    只会存在一个有效答案

    解题思路：
    使用 dict 存储已经遍历过的数字及其索引，对于每个数字，检查 target - 当前数字是否存在于 dict 中，
    如果存在则返回两个数字的索引，否则将当前数字和索引存入 dict。

    时间复杂度：O(n)，其中 n 是数组长度，我们只需要遍历数组一次
    空间复杂度：O(n)，最坏情况下需要存储数组中所有元素及其索引

    :param nums: 整数数组
    :param target: 目标值
    :return: 两个整数的数组下标
    """
    # 创建 dict 存储数字和其索引
    map = {}

    # 遍历数组
    for i in range(len(nums)):
```

```
        # 计算需要找到的另一个数字
        complement = target - nums[i]

        # 检查该数字是否已存在于 dict 中
        if complement in map:
            # 如果存在，返回两个数字的索引
            return [map[complement], i]

        # 将当前数字和索引存入 dict
        map[nums[i]] = i

    # 根据题目保证总会有一个解，这里仅为避免编译错误
    raise ValueError("No two sum solution")



# LeetCode 242. Valid Anagram (有效的字母异位词)
def is_anagram(s, t):
    """
    题目描述：
    给定两个字符串 s 和 t ，编写一个函数来判断 t 是否是 s 的字母异位词。
    注意：若 s 和 t 中每个字符出现的次数都相同，则称 s 和 t 互为字母异位词。

    示例：
    输入: s = "anagram", t = "nagaram"
    输出: true

    输入: s = "rat", t = "car"
    输出: false

    约束条件：
    1 <= s.length, t.length <= 5 * 10^4
    s 和 t 仅包含小写字母

    解题思路：
    使用 dict 记录字符串 s 中每个字符出现的次数，然后遍历字符串 t，对每个字符在 dict 中对应的计数减1，
    如果某个字符不存在或计数小于 0，则返回 false。最后检查 dict 中是否所有字符计数都为 0。

    时间复杂度：O(n)，其中 n 是字符串长度，需要遍历两个字符串
    空间复杂度：O(1)，因为字符集大小固定（小写字母 26 个），所以空间复杂度是常数

    :param s: 字符串 s
    :param t: 字符串 t
```

```
    :return: 如果 t 是 s 的字母异位词返回 True，否则返回 False
    """
    # 如果两个字符串长度不同，肯定不是字母异位词
    if len(s) != len(t):
        return False

    # 创建 dict 记录字符出现次数
    char_count = {}

    # 遍历字符串 s，统计每个字符出现次数
    for c in s:
        char_count[c] = char_count.get(c, 0) + 1

    # 遍历字符串 t，减少对应字符计数
    for c in t:
        # 如果字符不存在或计数已为 0，返回 False
        if c not in char_count or char_count[c] == 0:
            return False
        # 减少字符计数
        char_count[c] -= 1

    return True


# LeetCode 349. Intersection of Two Arrays (两个数组的交集)
def intersection(nums1, nums2):
    """
    题目描述：
    给定两个数组 nums1 和 nums2 ，返回它们的交集。输出结果中的每个元素一定是唯一的。
    我们可以不考虑输出结果的顺序。

    示例：
    输入：nums1 = [1,2,2,1], nums2 = [2,2]
    输出：[2]

    输入：nums1 = [4,9,5], nums2 = [9,4,9,8,4]
    输出：[9,4]
    解释：[4,9] 也是可通过的

    约束条件：
    1 <= nums1.length, nums2.length <= 1000
    0 <= nums1[i], nums2[i] <= 1000
```

```
    解题思路：
    使用 set 存储 nums1 中的所有元素，然后遍历 nums2，检查每个元素是否存在于 set 中，
    如果存在则加入结果集。

    时间复杂度：O(m+n)，其中 m 和 n 分别是两个数组的长度
    空间复杂度：O(m)，用于存储 nums1 中的元素

    :param nums1: 数组 1
    :param nums2: 数组 2
    :return: 交集数组
    """
    # 使用 set 存储 nums1 中的元素
    set1 = set(nums1)

    # 使用 set 存储交集结果，自动去重
    intersection_set = set()

    # 遍历 nums2，查找交集
    for num in nums2:
        if num in set1:
            intersection_set.add(num)

    # 将结果转换为 list
    return list(intersection_set)


# LeetCode 705. Design HashSet (设计哈希集合)
class MyHashSet:
    """
    题目描述：
    不使用任何内建的哈希表库设计一个哈希集合（HashSet）。
    实现 MyHashSet 类：
    void add(key) 向哈希集合中插入值 key 。
    bool contains(key) 返回哈希集合中是否存在这个值 key 。
    void remove(key) 将给定值 key 从哈希集合中删除。如果哈希集合中没有这个值，什么也不做。

    示例：
    输入：
    ["MyHashSet", "add", "add", "contains", "contains", "add", "contains", "remove", "contains"]
    [[], [1], [2], [1], [3], [2], [2], [2], [2]]
    输出：
    [null, null, null, true, false, null, true, null, false]
```

```
    约束条件：
    0 <= key <= 10^6
    最多调用 10^4 次 add、remove 和 contains

    解题思路：
    使用链地址法实现哈希表，创建一个固定大小的数组，每个数组元素是一个链表。
    当发生哈希冲突时，将元素添加到对应位置的链表中。

    时间复杂度：O(n/b)，其中 n 是元素个数，b 是桶数（在实际实现中我们使用 10000 作为桶数）
    空间复杂度：O(n)，存储所有元素
    """

    def __init__(self):
        """Initialize your data structure here."""
        self.BASE = 10000
        self.data = [[] for _ in range(self.BASE)]

    def _hash(self, key):
        return key % self.BASE

    def add(self, key):
        h = self._hash(key)
        if key not in self.data[h]:
            self.data[h].append(key)

    def remove(self, key):
        h = self._hash(key)
        if key in self.data[h]:
            self.data[h].remove(key)

    def contains(self, key):
        """Returns true if this set contains the specified element"""
        h = self._hash(key)
        return key in self.data[h]


# LeetCode 706. Design HashMap (设计哈希映射)
class MyHashMap:
    """
    题目描述：
    不使用任何内建的哈希表库设计一个哈希映射（HashMap）。
    实现 MyHashMap 类：
    MyHashMap() 用空映射初始化对象
```

void put(int key, int value) 向 HashMap 插入一个键值对 (key, value) 。如果 key 已经存在于映射中，则更新其对应的值 value 。

int get(int key) 返回特定的 key 所映射的 value ；如果映射中不包含 key 的映射，返回 -1 。

void remove(key) 如果映射中存在 key 的映射，则移除 key 和它所对应的 value 。

示例：

输入：

["MyHashMap", "put", "put", "get", "get", "put", "get", "remove", "get"]

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

输出：

[null, null, null, 1, -1, null, 1, null, -1]

约束条件：

0 <= key, value <= 10^6

最多调用 10^4 次 put、get 和 remove 方法

解题思路：

使用链地址法实现哈希表，创建一个固定大小的数组，每个数组元素是一个链表。

每个链表节点存储键值对，当发生哈希冲突时，将节点添加到对应位置的链表中。

时间复杂度：O(n/b)，其中 n 是元素个数，b 是桶数（在实际实现中我们使用 10000 作为桶数）

空间复杂度：O(n)，存储所有键值对

"""

```
def __init__(self):
    """Initialize your data structure here."""
    self.BASE = 10000
    self.data = [[] for _ in range(self.BASE)]

def _hash(self, key):
    return key % self.BASE

def put(self, key, value):
    h = self._hash(key)
    for i, (k, v) in enumerate(self.data[h]):
        if k == key:
            self.data[h][i] = (key, value)
            return
    self.data[h].append((key, value))

def get(self, key):
    """Returns the value to which the specified key is mapped, or -1 if this map contains no mapping for the key"""
```

```
        h = self._hash(key)
        for k, v in self.data[h]:
            if k == key:
                return v
        return -1

    def remove(self, key):
        """Removes the mapping of the specified value key if this map contains a mapping for the key"""
        h = self._hash(key)
        for i, (k, v) in enumerate(self.data[h]):
            if k == key:
                del self.data[h][i]
                return


# LeetCode 128. Longest Consecutive Sequence (最长连续序列)
def longest_consecutive(nums):
    """
    题目描述：
    给定一个未排序的整数数组 nums ，找出数字连续的最长序列（不要求序列元素在原数组中连续）的长度。
    请你设计并实现时间复杂度为 O(n) 的算法解决此问题。

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

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

    约束条件：
    0 <= nums.length <= 10^5
    -10^9 <= nums[i] <= 10^9

    解题思路：
    使用 set 存储所有数字，然后对于每个数字，如果它是某个序列的开始（即 num-1 不在 set 中），
    则从该数字开始向后查找连续的数字，计算序列长度。

    时间复杂度：O(n)，虽然有嵌套循环，但每个元素最多被访问两次
    空间复杂度：O(n)，用于存储 set
    """
```

```
    num_set = set(nums)
    longest_streak = 0

    for num in num_set:
        # 只有当num-1不存在时，num才是一个序列的开始
        if num - 1 not in num_set:
            current_num = num
            current_streak = 1

            # 向后查找连续的数字
            while current_num + 1 in num_set:
                current_num += 1
                current_streak += 1

            longest_streak = max(longest_streak, current_streak)

    return longest_streak


# LeetCode 49. Group Anagrams (字母异位词分组)
def group_anagrams(strs):
    """
    题目描述：
    给你一个字符串数组，请你将字母异位词组合在一起。可以按任意顺序返回结果列表。
    字母异位词是由重新排列源单词的所有字母得到的一个新单词。

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

    输入: strs = [""]
    输出: [[""]]

    输入: strs = ["a"]
    输出: [["a"]]

    约束条件：
    1 <= strs.length <= 10^4
    0 <= strs[i].length <= 100
    strs[i] 仅包含小写字母

    解题思路：
    使用 dict 存储分组结果，键为排序后的字符串，值为该组的所有字符串。
```

对于每个字符串，将其字符排序后作为键，将原字符串添加到对应的列表中。

时间复杂度：O(N*K*logK)，其中 N 是字符串数组的长度，K 是字符串的最大长度
空间复杂度：O(N*K)，用于存储所有字符串
"""
map = defaultdict(list)

for s in strs:
    # 将字符串排序后作为键
    key = ''.join(sorted(s))
    # 将原字符串添加到对应的列表中
    map[key].append(s)

return list(map.values())

# LeetCode 347. Top K Frequent Elements (前 K 个高频元素)
def top_k_frequent(nums, k):
    """
    题目描述：
    给你一个整数数组 nums 和一个整数 k ，请你返回其中出现频率前 k 高的元素。你可以按任意顺序返回答案。

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

    输入: nums = [1], k = 1
    输出: [1]

    约束条件：
    1 <= nums.length <= 10^5
    k 的取值范围是 [1, 数组中不相同的元素的个数]
    题目数据保证答案唯一

    解题思路：
    1. 使用 dict 统计每个元素的频率
    2. 使用最小堆维护前 k 个高频元素
    3. 遍历 dict，维护堆的大小为 k
    4. 最后从堆中取出所有元素

    时间复杂度：O(N*logK)，其中 N 是数组长度
    空间复杂度：O(N)，用于存储 dict 和堆

```
    """
    # 统计每个元素的频率
    freq_map = defaultdict(int)
    for num in nums:
        freq_map[num] += 1

    # 使用最小堆维护前 k 个高频元素
    heap = []

    # 遍历频率映射，维护堆的大小为 k
    for key, value in freq_map.items():
        heapq.heappush(heap, (value, key))
        if len(heap) > k:
            heapq.heappop(heap)

    # 从堆中取出所有元素
    result = []
    while heap:
        result.append(heapq.heappop(heap)[1])

    return result[::-1]  # 反转结果以获得正确顺序


# LeetCode 219. Contains Duplicate II (存在重复元素 II)
def contains_nearby_duplicate(nums, k):
    """
    题目描述：
    给定一个整数数组和一个整数 k，判断数组中是否存在两个不同的索引 i 和 j，
    使得 nums[i] == nums[j]，并且 i 和 j 的差的绝对值至多为 k。

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

    输入: nums = [1,0,1,1], k = 1
    输出: True

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

    约束条件：
    1 <= nums.length <= 10^5
    -10^9 <= nums[i] <= 10^9
```

```
    0 <= k <= 10^5

    解题思路：
    使用 dict 存储每个元素最后一次出现的索引，遍历数组时，检查当前元素是否在 dict 中存在，
    如果存在且当前索引与存储的索引之差的绝对值小于等于 k，则返回 True；
    否则更新 dict 中该元素的索引为当前索引。

    时间复杂度：O(n)，其中 n 是数组长度，我们只需要遍历数组一次
    空间复杂度：O(min(n,k))，最坏情况下需要存储 min(n,k)个元素
    """
    # 创建字典存储数字和其索引
    num_dict = {}

    # 遍历数组
    for i in range(len(nums)):
        # 检查当前数字是否已存在于字典中
        if nums[i] in num_dict:
            # 如果存在，检查索引差是否小于等于 k
            if i - num_dict[nums[i]] <= k:
                return True
        # 更新或添加数字的索引
        num_dict[nums[i]] = i

    # 遍历完数组未找到符合条件的重复元素
    return False


# LeetCode 3. Longest Substring Without Repeating Characters (无重复字符的最长子串)
def length_of_longest_substring(s):
    """
    题目描述：
    给定一个字符串 s，请你找出其中不含有重复字符的最长子串的长度。

    示例：
    输入: s = "abcabcbb"
    输出: 3
    解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。

    输入: s = "bbbbb"
    输出: 1
    解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。

    输入: s = "pwwkew"
```

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

约束条件：
0 <= s.length <= 5 * 10^4
s 由英文字母、数字、符号和空格组成

解题思路：
使用滑动窗口和 dict，dict 记录每个字符最后一次出现的位置。
维护一个左边界 left，当遇到重复字符时，将 left 更新为重复字符上一次出现位置的下一个位置。
计算当前窗口长度 i-left+1，并更新最大长度。

时间复杂度：O(n)，其中 n 是字符串长度
空间复杂度：O(min(n, m))，其中 m 是字符集大小
"""
# 创建字典存储字符和其位置
char_dict = {}
max_length = 0
left = 0

# 遍历字符串
for i in range(len(s)):
    # 如果字符已存在且在当前窗口内，更新左边界
    if s[i] in char_dict and char_dict[s[i]] >= left:
        left = char_dict[s[i]] + 1
    # 更新字符位置
    char_dict[s[i]] = i
    # 更新最大长度
    max_length = max(max_length, i - left + 1)

return max_length

# LeetCode 36. Valid Sudoku (有效的数独)
def is_valid_sudoku(board):
"""
题目描述：
请你判断一个 9 x 9 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。
1. 数字 1-9 在每一行只能出现一次。
2. 数字 1-9 在每一列只能出现一次。
3. 数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。

注意：
- 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。
- 空白格用 '.' 表示。

示例：
输入：board =
[
  ["5","3",".",".","7",".",".",".",".","0"],
  ["6",".",".","1","9","5",".",".",".","0"],
  [".","9","8",".",".",".",".",".","6","0"],
  ["8",".",".",".",".","6",".",".",".","3"],
  ["4",".",".","8",".","3",".",".",".","1"],
  ["7",".",".",".",".","2",".",".",".","6"],
  [".","6",".",".",".",".",".","2","8","0"],
  [".",".",".","4","1","9",".",".",".","5"],
  [".",".",".",".",".","8",".",".","7","9"]
]
输出：True

约束条件：
board.length == 9
board[i].length == 9
board[i][j] 是一位数字或者 '.'

解题思路：
使用三个列表分别存储行、列、3x3 宫格中出现过的数字。
每个列表包含 9 个 set，分别对应 9 行、9 列和 9 个宫格。
遍历数独，对于每个非空白字符，检查是否已经在对应的行、列或宫格中出现过。
宫格索引可以通过公式：box_index = (row // 3) * 3 + (col // 3) 计算得到。

时间复杂度：O(1)，因为数独大小固定为 9x9
空间复杂度：O(1)，固定大小的 set 列表
"""
```
# 初始化行、列、宫格的集合列表
rows = [set() for _ in range(9)]
cols = [set() for _ in range(9)]
boxes = [set() for _ in range(9)]

# 遍历数独
for row in range(9):
    for col in range(9):
        # 获取当前单元格的值
```

```
            val = board[row][col]
            # 跳过空白格
            if val == '.':
                continue

            # 计算宫格索引
            box_index = (row // 3) * 3 + (col // 3)

            # 检查是否重复
            if (val in rows[row] or
                val in cols[col] or
                val in boxes[box_index]):
                return False

            # 添加到对应的集合中
            rows[row].add(val)
            cols[col].add(val)
            boxes[box_index].add(val)

    # 所有检查通过，返回 True
    return True



# LeetCode 141. Linked List Cycle (环形链表)
# 定义链表节点类
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None

def has_cycle(head):
    """
    题目描述：
    给定一个链表，判断链表中是否有环。
    如果链表中存在环，则返回 True 。 否则，返回 False 。

    进阶：
    你能否不使用额外空间解决此题？

    解题思路 1（使用 set）：
    遍历链表，将每个节点存入 set 中，如果遇到已存在的节点，则说明有环。

    时间复杂度：O(n)，其中 n 是链表长度
```

```
    空间复杂度：O(n)，需要存储所有节点
    """
    # 创建集合存储已访问的节点
    seen = set()
    current = head

    # 遍历链表
    while current:
        # 如果节点已存在于集合中，说明有环
        if current in seen:
            return True
        # 将当前节点添加到集合
        seen.add(current)
        # 移动到下一个节点
        current = current.next

    # 遍历完链表未发现环
    return False


# LeetCode 160. Intersection of Two Linked Lists (相交链表)
def get_intersection_node(headA, headB):
    """
    题目描述：
    给你两个单链表的头节点 headA 和 headB ，请你找出并返回两个单链表相交的起始节点。如果两个链表没有交点，返回 None 。

    解题思路 1（使用 set）：
    遍历链表 A，将每个节点存入 set 中，然后遍历链表 B，检查节点是否存在于 set 中。

    时间复杂度：O(m+n)，其中 m 和 n 分别是两个链表的长度
    空间复杂度：O(m)，需要存储链表 A 的所有节点
    """
    # 创建集合存储链表 A 的节点
    seen = set()
    current = headA

    # 将链表 A 的所有节点存入集合
    while current:
        seen.add(current)
        current = current.next

    # 遍历链表 B，查找交集
```

```
    current = headB
    while current:
        # 如果节点存在于集合中，说明是交点
        if current in seen:
            return current
        current = current.next

    # 没有交点
    return None


# 测试代码
if __name__ == "__main__":
    # 测试两数之和
    print("测试两数之和:")
    nums = [2, 7, 11, 15]
    target = 9
    result = two_sum(nums, target)
    print(f"[{result[0]}, {result[1]}]")

    # 测试有效的字母异位词
    print("测试有效的字母异位词:")
    print(is_anagram("anagram", "nagaram"))  # True
    print(is_anagram("rat", "car"))  # False

    # 测试两个数组的交集
    print("测试两个数组的交集:")
    nums1 = [1, 2, 2, 1]
    nums2 = [2, 2]
    result = intersection(nums1, nums2)
    print(f"交集: {result}")

    # 测试设计哈希集合
    print("测试设计哈希集合:")
    my_hash_set = MyHashSet()
    my_hash_set.add(1)
    my_hash_set.add(2)
    print(my_hash_set.contains(1))  # True
    print(my_hash_set.contains(3))  # False
    my_hash_set.add(2)
    print(my_hash_set.contains(2))  # True
    my_hash_set.remove(2)
    print(my_hash_set.contains(2))  # False
```

```
# 测试设计哈希映射
print("测试设计哈希映射:")
my_hash_map = MyHashMap()
my_hash_map.put(1, 1)
my_hash_map.put(2, 2)

# 测试存在重复元素 II
print("测试存在重复元素 II:")
nums1 = [1, 2, 3, 1]
print(contains_nearby_duplicate(nums1, 3))  # True
nums2 = [1, 0, 1, 1]
print(contains_nearby_duplicate(nums2, 1))  # True
nums3 = [1, 2, 3, 1, 2, 3]
print(contains_nearby_duplicate(nums3, 2))  # False

# 测试无重复字符的最长子串
print("测试无重复字符的最长子串:")
print(length_of_longest_substring("abcabcbb"))  # 3
print(length_of_longest_substring("bbbbb"))  # 1
print(length_of_longest_substring("pwwkew"))  # 3

# 测试有效的数独
print("测试有效的数独:")
board = [
    ["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"]
]
print(is_valid_sudoku(board))  # True

# 测试环形链表
print("测试环形链表:")
# 创建有环链表
node1 = ListNode(3)
node2 = ListNode(2)
node3 = ListNode(0)
```

```
node4 = ListNode(-4)
node1.next = node2
node2.next = node3
node3.next = node4
node4.next = node2  # 环
print(has_cycle(node1))  # True

# 创建无环链表
node1 = ListNode(1)
node2 = ListNode(2)
node1.next = node2
print(has_cycle(node1))  # False

# 测试相交链表
print("测试相交链表:")
# 创建相交节点
intersect = ListNode(8)
intersect.next = ListNode(4)
intersect.next.next = ListNode(5)

# 创建链表 A
headA = ListNode(4)
headA.next = ListNode(1)
headA.next.next = intersect

# 创建链表 B
headB = ListNode(5)
headB.next = ListNode(6)
headB.next.next = ListNode(1)
headB.next.next.next = intersect

result = get_intersection_node(headA, headB)
print(result.val if result else None)  # 8
print(my_hash_map.get(1))  # 1
print(my_hash_map.get(3))  # -1
my_hash_map.put(2, 1)
print(my_hash_map.get(2))  # 1
my_hash_map.remove(2)
print(my_hash_map.get(2))  # -1

# 测试最长连续序列
print("测试最长连续序列:")
nums = [100, 4, 200, 1, 3, 2]
```

```
    print(longest_consecutive(nums))  # 4

    # 测试字母异位词分组
    print("测试字母异位词分组:")
    strs = ["eat", "tea", "tan", "ate", "nat", "bat"]
    groups = group_anagrams(strs)
    print(groups)

    # 测试前 K 个高频元素
    print("测试前 K 个高频元素:")
    nums = [1, 1, 1, 2, 2, 3]
    k = 2
    result = top_k_frequent(nums, k)
    print(result)  # [1, 2]
    print("测试最长连续序列:")
    nums = [100, 4, 200, 1, 3, 2]
    print(longest_consecutive(nums))  # 4

    # 测试字母异位词分组
    print("测试字母异位词分组:")
    strs = ["eat", "tea", "tan", "ate", "nat", "bat"]
    groups = group_anagrams(strs)
    print(groups)

    # 测试前 K 个高频元素
    print("测试前 K 个高频元素:")
    nums = [1, 1, 1, 2, 2, 3]
    k = 2
    result = top_k_frequent(nums, k)
    print(result)  # [1, 2]
```

==========

文件: Code02_TreeSetAndTreeMap.cpp

==========

```
#include <iostream>
#include <vector>
#include <set>
#include <map>
#include <algorithm>
#include <climits>

using namespace std;
```

```
/**
 * C++版本：TreeSet 和 TreeMap 相关题目与解析
 *
 * 在 C++中，我们使用 set 和 map 来实现 TreeSet 和 TreeMap 功能
 * TreeSet 特点：基于红黑树实现，元素有序，查找、插入、删除时间复杂度 O(logN)
 * TreeMap 特点：基于红黑树实现，键值对有序，查找、插入、删除时间复杂度 O(logN)
 *
 * 相关平台题目：
 * 1. LeetCode 220. Contains Duplicate III (存在重复元素 III) - https://leetcode.com/problems/contains-duplicate-iii/
 * 2. LeetCode 933. Number of Recent Calls (最近的请求次数) - https://leetcode.com/problems/number-of-recent-calls/
 * 3. LeetCode 729. My Calendar I (我的日程安排表 I) - https://leetcode.com/problems/my-calendar-i/
 * 4. LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数) - https://leetcode.com/problems/count-of-smaller-numbers-after-self/
 * 5. LeetCode 493. Reverse Pairs (翻转对) - https://leetcode.com/problems/reverse-pairs/
 * 6. Codeforces 4C. Registration System (注册系统) - https://codeforces.com/problemset/problem/4/C
 * 7. AtCoder ABC 217 D - Cutting Woods (切割木材) - https://atcoder.jp/contests/abc217/tasks/abc217_d
 * 8. USACO Silver: Why Did the Cow Cross the Road (为什么奶牛要过马路) - http://www.usaco.org/index.php?page=viewproblem2&cpid=714
 * 9. 洛谷 P3374 【模板】树状数组 1 (模板树状数组 1) - https://www.luogu.com.cn/problem/P3374
 * 10. CodeChef STFOOD (街头食物) - https://www.codechef.com/problems/STFOOD
 * 11. SPOJ ANARC09A - Seinfeld (宋飞正传) - https://www.spoj.com/problems/ANARC09A/
 * 12. Project Euler Problem 1: Multiples of 3 and 5 (3 和 5 的倍数) - https://projecteuler.net/problem=1
 * 13. HackerRank Frequency Queries (频率查询) - https://www.hackerrank.com/challenges/frequency-queries
 * 14. 计蒜客 T1100: 计算 2 的 N 次方 (计算 2 的 N 次方) - https://www.jisuanke.com/t/T1100
 * 15. 杭电 OJ 1002: A + B Problem II (A+B 问题 II) - http://acm.hdu.edu.cn/showproblem.php?pid=1002
 * 16. 牛客网 剑指 Offer 03: 数组中重复的数字 (数组中重复的数字) - https://www.nowcoder.com/practice/623a5ac0ea5b4e5f95552655361ae0a8
 * 17. acwing 799. 最长连续不重复子序列 (最长连续不重复子序列) - https://www.acwing.com/problem/content/801/
 * 18. POJ 1002: 487-3279 (电话号码) - http://poj.org/problem?id=1002
 * 19. UVa OJ 100: The 3n + 1 problem (3n+1 问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=36
 * 20. Timus OJ 1001: Reverse Root (反转平方根) - https://acm.timus.ru/problem.aspx?space=1&num=1001
```

 * 21. Aizu OJ ALDS1_4_C: Dictionary (字典) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_4_C
 * 22. Comet OJ Contest #0: 热身赛 A. 签到题 (签到题) - https://cometoj.com/contest/0/problem/A
 * 23. MarsCode 火星编程竞赛: 字符串去重排序 (字符串去重排序) - https://www.marscode.cn/contest/1/problem/1001
 * 24. ZOJ 1001: A + B Problem (A+B 问题) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1001
 * 25. LOJ 100: 顺序的分数 (顺序的分数) - https://loj.ac/p/100
 * 26. 各大高校 OJ: 清华大学 OJ 1000: A+B Problem (A+B 问题) - http://dsa.cs.tsinghua.edu.cn/oj/problem.shtml?id=1000
 * 27. LintCode 613. High Five (最高分五科) - https://www.lintcode.com/problem/high-five/
 * 28. HackerEarth Monk and the Magical Candy Bags (和尚与魔法糖果袋) - https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/practice-problems/algorithm/monk-and-the-magical-candy-bags/
 * 29. CodeChef FRGTNLNG (遗忘的语言) - https://www.codechef.com/problems/FRGTNLNG
 * 30. SPOJ DICT (字典) - https://www.spoj.com/problems/DICT/
 * 31. Project Euler Problem 2: Even Fibonacci numbers (偶数斐波那契数) - https://projecteuler.net/problem=2
 * 32. HackerRank Maximum Palindromes (最大回文) - https://www.hackerrank.com/challenges/maximum-palindromes
 * 33. 计蒜客 T1101: 阶乘 (阶乘) - https://www.jisuanke.com/t/T1101
 * 34. 杭电 OJ 1003: Max Sum (最大子序列和) - http://acm.hdu.edu.cn/showproblem.php?pid=1003
 * 35. 牛客网 剑指 Offer 50: 第一个只出现一次的字符 (第一个只出现一次的字符) - https://www.nowcoder.com/practice/1c82e8cf713b4bbeb2a5b31cf5b0417c
 * 36. acwing 800. 数组元素的目标和 (数组元素的目标和) - https://www.acwing.com/problem/content/802/
 * 37. POJ 1003: Hangover (悬垂) - http://poj.org/problem?id=1003
 * 38. UVa OJ 101: The Blocks Problem (积木问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=37
 * 39. Timus OJ 1005: Stone Pile (石子堆) - https://acm.timus.ru/problem.aspx?space=1&num=1005
 * 40. Aizu OJ ALDS1_5_A: Exhaustive Search (穷举搜索) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_5_A
 * 41. Comet OJ Contest #1: 热身赛 B. 简单的数学题 (简单的数学题) - https://cometoj.com/contest/1/problem/B
 * 42. MarsCode 火星编程竞赛: 数字统计 (数字统计) - https://www.marscode.cn/contest/1/problem/1002
 * 43. ZOJ 1002: Fire Net (消防网) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1002
 * 44. LOJ 101: 最小生成树 (最小生成树) - https://loj.ac/p/101
 * 45. 各大高校 OJ: 北京大学 OJ 1000: A+B Problem (A+B 问题) - http://poj.org/problem?id=1000
 * 46. LeetCode 352. Data Stream as Disjoint Intervals (数据流变为不相交区间) - https://leetcode.com/problems/data-stream-as-disjoint-intervals/
 * 47. Codeforces 122A. Lucky Division (幸运分割) -

```
https://codeforces.com/problemset/problem/122/A
 * 48. AtCoder ABC 218 C - Shapes (形状) - https://atcoder.jp/contests/abc218/tasks/abc218_c
 * 49. USACO Bronze: Block Game (积木游戏) -
http://www.usaco.org/index.php?page=viewproblem2&cpid=664
 * 50. 洛谷 P3366 【模板】最小生成树 (模板最小生成树) - https://www.luogu.com.cn/problem/P3366
 */

// LeetCode 220. Contains Duplicate III (存在重复元素 III)
bool containsNearbyAlmostDuplicate(vector<int>& nums, int k, int t) {
    // 使用 set 维护滑动窗口
    set<long long> window;

    for (int i = 0; i < nums.size(); i++) {
        // 查找大于等于当前元素的最小元素
        auto it = window.lower_bound((long long)nums[i]);
        if (it != window.end() && *it - nums[i] <= t) {
            return true;
        }

        // 查找小于等于当前元素的最大元素
        if (it != window.begin()) {
            --it;
            if (nums[i] - *it <= t) {
                return true;
            }
        }

        // 添加当前元素到 set
        window.insert((long long)nums[i]);

        // 维护滑动窗口大小为 k
        if (window.size() > k) {
            window.erase((long long)nums[i - k]);
        }
    }

    return false;
}

// LeetCode 933. Number of Recent Calls (最近的请求次数)
class RecentCounter {
private:
    multiset<int> requests;
```

```
public:
    RecentCounter() {
        // 构造函数不需要特殊处理
    }

    int ping(int t) {
        requests.insert(t);
        // 计算[t-3000, t]范围内的请求数量
        auto it = requests.upper_bound(t - 3000);
        return distance(it, requests.end());
    }
};

// LeetCode 729. My Calendar I (我的日程安排表 I)
class MyCalendar {
private:
    map<int, int> calendar;

public:
    MyCalendar() {
        // 构造函数不需要特殊处理
    }

    bool book(int start, int end) {
        // 查找开始时间大于等于 start 的最小日程
        auto it = calendar.lower_bound(start);

        // 检查与后一个日程是否冲突
        if (it != calendar.end() && it->first < end) {
            return false;
        }

        // 检查与前一个日程是否冲突
        if (it != calendar.begin()) {
            --it;
            if (it->second > start) {
                return false;
            }
        }

        // 没有冲突，添加新日程
        calendar[start] = end;
```

```
        return true;
    }
};

// LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数)
vector<int> countSmaller(vector<int>& nums) {
    vector<int> result(nums.size());
    multiset<int> sortedSet;

    // 从右向左遍历
    for (int i = nums.size() - 1; i >= 0; i--) {
        // 查找小于当前元素的元素个数
        auto it = sortedSet.lower_bound(nums[i]);
        result[i] = distance(sortedSet.begin(), it);
        sortedSet.insert(nums[i]);
    }

    return result;
}

// LeetCode 493. Reverse Pairs (翻转对)
int reversePairs(vector<int>& nums) {
    int count = 0;
    multiset<long long> sortedSet;

    // 从右向左遍历
    for (int i = nums.size() - 1; i >= 0; i--) {
        // 查找小于 nums[i]/2.0 的元素个数
        auto it = sortedSet.lower_bound((long long)nums[i]);
        count += distance(sortedSet.begin(), it);
        sortedSet.insert((long long)nums[i] * 2);
    }

    return count;
}

int main() {
    // 测试存在重复元素 III
    cout << "测试存在重复元素 III:" << endl;
    vector<int> nums1 = {1, 2, 3, 1};
    cout << containsNearbyAlmostDuplicate(nums1, 3, 0) << endl; // 1 (true)

    // 测试最近的请求次数
```

```
    cout << "测试最近的请求次数:" << endl;
    RecentCounter counter;
    cout << counter.ping(1) << endl;     // 1
    cout << counter.ping(100) << endl;   // 2
    cout << counter.ping(3001) << endl;  // 3
    cout << counter.ping(3002) << endl;  // 3

    // 测试我的日程安排表
    cout << "测试我的日程安排表:" << endl;
    MyCalendar calendar;
    cout << calendar.book(10, 20) << endl; // 1 (true)
    cout << calendar.book(15, 25) << endl; // 0 (false)
    cout << calendar.book(20, 30) << endl; // 1 (true)

    // 测试计算右侧小于当前元素的个数
    cout << "测试计算右侧小于当前元素的个数:" << endl;
    vector<int> nums2 = {5, 2, 6, 1};
    vector<int> result1 = countSmaller(nums2);
    cout << "[";
    for (int i = 0; i < result1.size(); i++) {
        cout << result1[i];
        if (i < result1.size() - 1) {
            cout << ", ";
        }
    }
    cout << "]" << endl; // [2, 1, 1, 0]

    // 测试翻转对
    cout << "测试翻转对:" << endl;
    vector<int> nums3 = {1, 3, 2, 3, 1};
    cout << reversePairs(nums3) << endl; // 2

    return 0;
}

==========================================

文件: Code02_TreeSetAndTreeMap.java
==========================================
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.PriorityQueue;
```

```
import java.util.TreeMap;
import java.util.TreeSet;

/**
 * TreeSet 和 TreeMap 相关题目与解析
 *
 * TreeSet 基于 TreeMap 实现，底层使用红黑树，查询时间复杂度 O(log n)，元素有序
 * TreeMap 基于红黑树实现，查询时间复杂度 O(log n)，键值对按键有序
 *
 * 相关平台题目：
 * 1. LeetCode 220. Contains Duplicate III (存在重复元素 III) - https://leetcode.com/problems/contains-duplicate-iii/
 * 2. LeetCode 349. Intersection of Two Arrays (两个数组的交集) - https://leetcode.com/problems/intersection-of-two-arrays/
 * 3. LeetCode 352. Data Stream as Disjoint Intervals (将数据流变为多个不相交区间) - https://leetcode.com/problems/data-stream-as-disjoint-intervals/
 * 4. LeetCode 363. Max Sum of Rectangle No Larger Than K (矩形区域不超过 K 的最大数值和) - https://leetcode.com/problems/max-sum-of-rectangle-no-larger-than-k/
 * 5. LeetCode 456. 132 Pattern (132 模式) - https://leetcode.com/problems/132-pattern/
 * 6. LeetCode 632. Smallest Range Covering Elements from K Lists (最小区间) - https://leetcode.com/problems/smallest-range-covering-elements-from-k-lists/
 * 7. LeetCode 699. Falling Squares (掉落的方块) - https://leetcode.com/problems/falling-squares/
 * 8. LeetCode 715. Range Module (Range 模块) - https://leetcode.com/problems/range-module/
 * 9. LeetCode 729. My Calendar I (我的日程安排表 I) - https://leetcode.com/problems/my-calendar-i/
 * 10. LeetCode 846. Hand of Straights (一手顺子) - https://leetcode.com/problems/hand-of-straights/
 * 11. LeetCode 855. Exam Room (考场就座) - https://leetcode.com/problems/exam-room/
 * 12. LeetCode 933. Number of Recent Calls (最近的请求次数) - https://leetcode.com/problems/number-of-recent-calls/
 * 13. LeetCode 975. Odd Even Jump (奇偶跳) - https://leetcode.com/problems/odd-even-jump/
 * 14. LeetCode 981. Time Based Key-Value Store (基于时间的键值存储) - https://leetcode.com/problems/time-based-key-value-store/
 * 15. LeetCode 1244. Design A Leaderboard (力扣排行榜) - https://leetcode.com/problems/design-a-leaderboard/
 * 16. HackerRank Java TreeSet (Java 树集) - https://www.hackerrank.com/challenges/java-tree-set
 * 17. LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数) - https://leetcode.com/problems/count-of-smaller-numbers-after-self/
 * 18. LeetCode 327. Count of Range Sum (区间和的个数) - https://leetcode.com/problems/count-of-range-sum/
 * 19. LeetCode 493. Reverse Pairs (翻转对) - https://leetcode.com/problems/reverse-pairs/
 * 20. LeetCode 987. Vertical Order Traversal of a Binary Tree (二叉树的垂序遍历) - https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/
```

```
 * 21. Codeforces 4C. Registration System (注册系统) -
https://codeforces.com/problemset/problem/4/C
 * 22. AtCoder ABC 217 D - Cutting Woods (切割木材) -
https://atcoder.jp/contests/abc217/tasks/abc217_d
 * 23. USACO Silver: Why Did the Cow Cross the Road (为什么奶牛要过马路) -
http://www.usaco.org/index.php?page=viewproblem2&cpid=714
 * 24. 洛谷 P3374 【模板】树状数组 1 (模板树状数组 1) - https://www.luogu.com.cn/problem/P3374
 * 25. CodeChef STFOOD (街头食物) - https://www.codechef.com/problems/STFOOD
 * 26. SPOJ ANARC09A - Seinfeld (宋飞正传) - https://www.spoj.com/problems/ANARC09A/
 * 27. Project Euler Problem 1: Multiples of 3 and 5 (3 和 5 的倍数) -
https://projecteuler.net/problem=1
 * 28. HackerRank Frequency Queries (频率查询) - https://www.hackerrank.com/challenges/frequency-
queries
 * 29. 计蒜客 T1100: 计算 2 的 N 次方 (计算 2 的 N 次方) - https://www.jisuanke.com/t/T1100
 * 30. 杭电 OJ 1002: A + B Problem II (A+B 问题 II) -
http://acm.hdu.edu.cn/showproblem.php?pid=1002
 * 31. 牛客网 剑指 Offer 03: 数组中重复的数字 (数组中重复的数字) -
https://www.nowcoder.com/practice/623a5ac0ea5b4e5f95552655361ae0a8
 * 32. acwing 799. 最长连续不重复子序列 (最长连续不重复子序列) -
https://www.acwing.com/problem/content/801/
 * 33. POJ 1002: 487-3279 (电话号码) - http://poj.org/problem?id=1002
 * 34. UVa OJ 100: The 3n + 1 problem (3n+1 问题) -
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=36
 * 35. Timus OJ 1001: Reverse Root (反转平方根) -
https://acm.timus.ru/problem.aspx?space=1&num=1001
 * 36. Aizu OJ ALDS1_4_C: Dictionary (字典) - http://judge.u-
aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_4_C
 * 37. Comet OJ Contest #0: 热身赛 A. 签到题 (签到题) - https://cometoj.com/contest/0/problem/A
 * 38. MarsCode 火星编程竞赛: 字符串去重排序 (字符串去重排序) -
https://www.marscode.cn/contest/1/problem/1001
 * 39. ZOJ 1001: A + B Problem (A+B 问题) -
http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1001
 * 40. LOJ 100: 顺序的分数 (顺序的分数) - https://loj.ac/p/100
 * 41. 各大高校 OJ: 清华大学 OJ 1000: A+B Problem (A+B 问题) -
http://dsa.cs.tsinghua.edu.cn/oj/problem.shtml?id=1000
 */

public class Code02_TreeSetAndTreeMap {

    /**
     * LeetCode 220. Contains Duplicate III (存在重复元素 III)
     *
     * 题目描述：
```

```
 * 给你一个整数数组 nums 和两个整数 k 和 t 。请你判断是否存在两个不同下标 i 和 j，
 * 使得 abs(nums[i] - nums[j]) <= t ，同时又满足 abs(i - j) <= k 。
 * 如果存在则返回 true，不存在返回 false。
 *
 * 示例：
 * 输入：nums = [1,2,3,1], k = 3, t = 0
 * 输出：true
 *
 * 输入：nums = [1,0,1,1], k = 1, t = 2
 * 输出：true
 *
 * 输入：nums = [1,5,9,1,5,9], k = 2, t = 3
 * 输出：false
 *
 * 约束条件：
 * 0 <= nums.length <= 2 * 10^4
 * -2^31 <= nums[i] <= 2^31 - 1
 * 0 <= k <= 10^4
 * 0 <= t <= 2^31 - 1
 *
 * 解题思路：
 * 使用 TreeSet 维护一个大小为 k 的滑动窗口，对于每个新元素，在 TreeSet 中查找是否存在一个元素
 * 与当前元素的差值不超过 t。利用 TreeSet 的 ceiling 和 floor 方法可以高效地找到最接近的元素。
 *
 * 时间复杂度：O(n log min(n,k))，其中 n 是数组长度
 * 空间复杂度：O(min(n,k))，用于存储滑动窗口中的元素
 *
 * @param nums 整数数组
 * @param k 下标差值上限
 * @param t 元素差值上限
 * @return 如果存在满足条件的两个元素返回 true，否则返回 false
 */
public static boolean containsNearbyAlmostDuplicate(int[] nums, int k, int t) {
    // 使用 TreeSet 维护滑动窗口
    TreeSet<Long> set = new TreeSet<>();

    for (int i = 0; i < nums.length; i++) {
        // 查找大于等于当前元素的最小元素
        Long ceiling = set.ceiling((long) nums[i]);
        if (ceiling != null && ceiling - nums[i] <= t) {
            return true;
        }
```

```
            // 查找小于等于当前元素的最大元素
            Long floor = set.floor((long) nums[i]);
            if (floor != null && nums[i] - floor <= t) {
                return true;
            }

            // 添加当前元素到 TreeSet
            set.add((long) nums[i]);

            // 维护滑动窗口大小为 k
            if (set.size() > k) {
                set.remove((long) nums[i - k]);
            }
        }

        return false;
    }

    /**
     * LeetCode 933. Number of Recent Calls (最近的请求次数)
     *
     * 题目描述：
     * 写一个 RecentCounter 类来计算特定时间范围内最近的请求。
     * 请你实现 RecentCounter 类：
     * RecentCounter() 初始化计数器，请求数为 0 。
     * int ping(int t) 在时间 t 添加一个新请求，其中 t 表示以毫秒为单位的某个时间，
     * 并返回过去 3000 毫秒内发生的所有请求数（包括新请求）。
     * 精确地说，返回在 [t-3000, t] 内发生的请求数。
     * 保证每次对 ping 的调用都使用比之前更大的 t 值。
     *
     * 示例：
     * 输入：
     * ["RecentCounter", "ping", "ping", "ping", "ping"]
     * [[], [1], [100], [3001], [3002]]
     * 输出：
     * [null, 1, 2, 3, 3]
     *
     * 约束条件：
     * 1 <= t <= 10^9
     * 保证每次对 ping 调用所使用的 t 值都 严格递增
     * 至多调用 ping 方法 10^4 次
     *
     * 解题思路：
```

```
     * 使用 TreeSet 存储所有请求的时间戳，每次 ping 时，使用 subSet 方法获取[t-3000, t]范围内的请求数量。
     *
     * 时间复杂度：O(log n) 每次 ping 操作
     * 空间复杂度：O(n) 存储所有请求
     */
    static class RecentCounter {
        private TreeSet<Integer> requests;

        public RecentCounter() {
            requests = new TreeSet<>();
        }

        public int ping(int t) {
            requests.add(t);
            // 获取[t-3000, t]范围内的请求数量
            return requests.subSet(t - 3000, true, t, true).size();
        }
    }

    /**
     * LeetCode 729. My Calendar I (我的日程安排表 I)
     *
     * 题目描述：
     * 实现一个 MyCalendar 类来存放你的日程安排。如果要添加的日程安排不会造成 重复预订 ，则可以存储这个新的日程安排。
     * 当两个日程安排有一些时间上的交叉时（例如两个日程安排都在同一时间内），就会产生 重复预订 。
     * 日程可以用一对整数 start 和 end 表示，这里的时间是半开区间，即 [start, end)，
     * 实数 x 的范围为 start <= x < end 。
     *
     * 实现 MyCalendar 类：
     * MyCalendar() 初始化日历对象。
     * boolean book(int start, int end) 如果可以将日程安排成功添加到日历中而不会导致重复预订，返回 true 。
     * 否则，返回 false 并且不要将该日程安排添加到日历中。
     *
     * 示例：
     * 输入：
     * ["MyCalendar", "book", "book", "book"]
     * [[], [10, 20], [15, 25], [20, 30]]
     * 输出：
     * [null, true, false, true]
     *
```

```
 * 约束条件：
 * 0 <= start < end <= 10^9
 * 每个测试用例，调用 book 方法的次数最多不超过 1000 次。
 *
 * 解题思路：
 * 使用 TreeMap 存储已预订的日程，key 为开始时间，value 为结束时间。
 * 对于新的日程[start, end)，使用 TreeMap 的 floorEntry 和 ceilingEntry 方法查找可能冲突的日程：
 * 1. 查找开始时间小于等于 start 的最大日程，检查其结束时间是否大于 start
 * 2. 查找开始时间大于等于 start 的最小日程，检查其开始时间是否小于 end
 * 如果没有冲突，则添加新日程。
 *
 * 时间复杂度：O(log n) 每次 book 操作
 * 空间复杂度：O(n) 存储所有日程
 */
static class MyCalendar {
    private TreeMap<Integer, Integer> calendar;

    public MyCalendar() {
        calendar = new TreeMap<>();
    }

    public boolean book(int start, int end) {
        // 查找开始时间小于等于 start 的最大日程
        java.util.Map.Entry<Integer, Integer> floor = calendar.floorEntry(start);
        // 查找开始时间大于等于 start 的最小日程
        java.util.Map.Entry<Integer, Integer> ceiling = calendar.ceilingEntry(start);

        // 检查是否与 floor 日程冲突
        if (floor != null && floor.getValue() > start) {
            return false;
        }

        // 检查是否与 ceiling 日程冲突
        if (ceiling != null && ceiling.getKey() < end) {
            return false;
        }

        // 没有冲突，添加新日程
        calendar.put(start, end);
        return true;
    }
}
```

```java
/**
 * HackerRank Java TreeSet (Java 树集)
 *
 * 题目描述：
 * 给定一个整数列表，找出列表中所有不同的数字，并按升序排列。
 *
 * 示例：
 * 输入：[1, 2, 3, 2, 1, 4, 5, 4]
 * 输出：[1, 2, 3, 4, 5]
 *
 * 解题思路：
 * 使用 TreeSet 存储所有数字，由于 TreeSet 的特性，元素会自动排序且去重。
 *
 * 时间复杂度：O(n log n)，其中 n 是数组长度
 * 空间复杂度：O(n)，用于存储 TreeSet
 *
 * @param nums 整数数组
 * @return 排序后的不重复元素数组
 */
public static int[] sortAndRemoveDuplicates(int[] nums) {
    TreeSet<Integer> set = new TreeSet<>();
    for (int num : nums) {
        set.add(num);
    }

    int[] result = new int[set.size()];
    int index = 0;
    for (int num : set) {
        result[index++] = num;
    }

    return result;
}

/**
 * LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数)
 *
 * 题目描述：
 * 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：
 * 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
     *
     * 约束条件：
     * 1 <= nums.length <= 10^5
     * -10^4 <= nums[i] <= 10^4
     *
     * 解题思路：
     * 从右向左遍历数组，使用 TreeSet 维护已遍历的元素。
     * 对于每个元素，在 TreeSet 中查找小于它的元素个数。
     *
     * 时间复杂度：O(n log n)，其中 n 是数组长度
     * 空间复杂度：O(n)，用于存储 TreeSet
     *
     * @param nums 整数数组
     * @return 每个元素右侧小于它的元素数量数组
     */
    public static List<Integer> countSmaller(int[] nums) {
        List<Integer> result = new ArrayList<>();
        TreeSet<Integer> set = new TreeSet<>();

        // 从右向左遍历
        for (int i = nums.length - 1; i >= 0; i--) {
            // 查找小于当前元素的元素个数
            int count = set.headSet(nums[i]).size();
            result.add(0, count); // 在列表开头插入
            set.add(nums[i]);
        }

        return result;
    }

    /**
     * LeetCode 493. Reverse Pairs (翻转对)
     *
     * 题目描述：
     * 给定一个数组 nums ，如果 i < j 且 nums[i] > 2*nums[j] 我们将 (i, j) 称作一个重要翻转对。
     * 你需要返回给定数组中的重要翻转对的数量。
```

```
     *
     * 示例：
     * 输入: [1,3,2,3,1]
     * 输出: 2
     *
     * 输入: [2,4,3,5,1]
     * 输出: 3
     *
     * 约束条件：
     * 给定数组的长度不会超过 50000。
     * 输入数组中的所有数字都在 32 位整数的表示范围内。
     *
     * 解题思路：
     * 使用 TreeSet 维护已遍历的元素，对于每个新元素，在 TreeSet 中查找满足条件的元素个数。
     *
     * 时间复杂度：O(n log n)，其中 n 是数组长度
     * 空间复杂度：O(n)，用于存储 TreeSet
     *
     * @param nums 整数数组
     * @return 翻转对的数量
     */
    public static int reversePairs(int[] nums) {
        int count = 0;
        TreeSet<Long> set = new TreeSet<>();

        // 从右向左遍历
        for (int i = nums.length - 1; i >= 0; i--) {
            // 查找小于 nums[i]/2.0 的元素个数
            long target = (long) nums[i] * 2;
            count += set.headSet(target, false).size();
            set.add((long) nums[i]);
        }

        return count;
    }

    public static void main(String[] args) {
        // 底层红黑树
        TreeMap<Integer, String> treeMap = new TreeMap<>();
        treeMap.put(5, "这是 5");
        treeMap.put(7, "这是 7");
        treeMap.put(1, "这是 1");
        treeMap.put(2, "这是 2");
```

```
treeMap.put(3, "这是 3");
treeMap.put(4, "这是 4");
treeMap.put(8, "这是 8");

System.out.println(treeMap.containsKey(1));
System.out.println(treeMap.containsKey(10));
System.out.println(treeMap.get(4));
treeMap.put(4, "张三是 4");
System.out.println(treeMap.get(4));

treeMap.remove(4);
System.out.println(treeMap.get(4) == null);

System.out.println(treeMap.firstKey());
System.out.println(treeMap.lastKey());
// TreeMap 中，所有的 key，<= 4 且最近的 key 是什么
System.out.println(treeMap.floorKey(4));
// TreeMap 中，所有的 key，>= 4 且最近的 key 是什么
System.out.println(treeMap.ceilingKey(4));

System.out.println("========");

TreeSet<Integer> set = new TreeSet<>();
set.add(3);
set.add(3);
set.add(4);
set.add(4);
System.out.println("有序表大小 : " + set.size());
while (!set.isEmpty()) {
    System.out.println(set.pollFirst());
    // System.out.println(set.pollLast());
}

// 堆，默认小根堆、如果要大根堆，定制比较器!
PriorityQueue<Integer> heap1 = new PriorityQueue<>();
heap1.add(3);
heap1.add(3);
heap1.add(4);
heap1.add(4);
System.out.println("堆大小 : " + heap1.size());
while (!heap1.isEmpty()) {
    System.out.println(heap1.poll());
}
```

```
// 定制的大根堆，用比较器！
PriorityQueue<Integer> heap2 = new PriorityQueue<>((a, b) -> b - a);
heap2.add(3);
heap2.add(3);
heap2.add(4);
heap2.add(4);
System.out.println("堆大小 : " + heap2.size());
while (!heap2.isEmpty()) {
    System.out.println(heap2.poll());
}

// 测试添加的题目
System.out.println("===========");
System.out.println("测试存在重复元素 III:");
int[] nums = {1, 2, 3, 1};
System.out.println(containsNearbyAlmostDuplicate(nums, 3, 0)); // true

System.out.println("测试最近的请求次数:");
RecentCounter counter = new RecentCounter();
System.out.println(counter.ping(1));     // 1
System.out.println(counter.ping(100));   // 2
System.out.println(counter.ping(3001));  // 3
System.out.println(counter.ping(3002));  // 3

System.out.println("测试我的日程安排表:");
MyCalendar calendar = new MyCalendar();
System.out.println(calendar.book(10, 20)); // true
System.out.println(calendar.book(15, 25)); // false
System.out.println(calendar.book(20, 30)); // true

// 测试 HackerRank Java TreeSet
System.out.println("测试 HackerRank Java TreeSet:");
int[] nums1 = {1, 2, 3, 2, 1, 4, 5, 4};
int[] result1 = sortAndRemoveDuplicates(nums1);
System.out.println(Arrays.toString(result1)); // [1, 2, 3, 4, 5]

// 测试计算右侧小于当前元素的个数
System.out.println("测试计算右侧小于当前元素的个数:");
int[] nums2 = {5, 2, 6, 1};
List<Integer> result2 = countSmaller(nums2);
System.out.println(result2); // [2, 1, 1, 0]
```

```
        // 测试翻转对
        System.out.println("测试翻转对:");
        int[] nums3 = {1, 3, 2, 3, 1};
        System.out.println(reversePairs(nums3)); // 2
    }
}
```

==========

文件: Code02_TreeSetAndTreeMap.py
==========

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Python 版本：TreeSet 和 TreeMap 相关题目与解析

在 Python 中，我们使用 sortedcontainers 库中的 SortedList 和 SortedDict 来实现 TreeSet 和 TreeMap 功能
或者使用 bisect 模块来实现有序列表功能

相关平台题目：
1. LeetCode 220. Contains Duplicate III (存在重复元素 III) - https://leetcode.com/problems/contains-duplicate-iii/
2. LeetCode 933. Number of Recent Calls (最近的请求次数) - https://leetcode.com/problems/number-of-recent-calls/
3. LeetCode 729. My Calendar I (我的日程安排表 I) - https://leetcode.com/problems/my-calendar-i/
4. LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数) - https://leetcode.com/problems/count-of-smaller-numbers-after-self/
5. LeetCode 493. Reverse Pairs (翻转对) - https://leetcode.com/problems/reverse-pairs/
6. Codeforces 4C. Registration System (注册系统) - https://codeforces.com/problemset/problem/4/C
7. AtCoder ABC 217 D - Cutting Woods (切割木材) - https://atcoder.jp/contests/abc217/tasks/abc217_d
8. USACO Silver: Why Did the Cow Cross the Road (为什么奶牛要过马路) - http://www.usaco.org/index.php?page=viewproblem2&cpid=714
9. 洛谷 P3374 【模板】树状数组 1 (模板树状数组 1) - https://www.luogu.com.cn/problem/P3374
10. CodeChef STFOOD (街头食物) - https://www.codechef.com/problems/STFOOD
11. SPOJ ANARC09A - Seinfeld (宋飞正传) - https://www.spoj.com/problems/ANARC09A/
12. Project Euler Problem 1: Multiples of 3 and 5 (3 和 5 的倍数) - https://projecteuler.net/problem=1
13. HackerRank Frequency Queries (频率查询) - https://www.hackerrank.com/challenges/frequency-queries
14. 计蒜客 T1100: 计算 2 的 N 次方 (计算 2 的 N 次方) - https://www.jisuanke.com/t/T1100
15. 杭电 OJ 1002: A + B Problem II (A+B 问题 II) - http://acm.hdu.edu.cn/showproblem.php?pid=1002
```

16. 牛客网 剑指 Offer 03: 数组中重复的数字 (数组中重复的数字) -
https://www.nowcoder.com/practice/623a5ac0ea5b4e5f95552655361ae0a8
17. acwing 799. 最长连续不重复子序列 (最长连续不重复子序列) -
https://www.acwing.com/problem/content/801/
18. POJ 1002: 487-3279 (电话号码) - http://poj.org/problem?id=1002
19. UVa OJ 100: The 3n + 1 problem (3n+1 问题) -
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=36
20. Timus OJ 1001: Reverse Root (反转平方根) - https://acm.timus.ru/problem.aspx?space=1&num=1001
21. Aizu OJ ALDS1_4_C: Dictionary (字典) - http://judge.u-
aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_4_C
22. Comet OJ Contest #0: 热身赛 A. 签到题 (签到题) - https://cometoj.com/contest/0/problem/A
23. MarsCode 火星编程竞赛: 字符串去重排序 (字符串去重排序) -
https://www.marscode.cn/contest/1/problem/1001
24. ZOJ 1001: A + B Problem (A+B 问题) -
http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1001
25. LOJ 100: 顺序的分数 (顺序的分数) - https://loj.ac/p/100
26. 各大高校 OJ: 清华大学 OJ 1000: A+B Problem (A+B 问题) -
http://dsa.cs.tsinghua.edu.cn/oj/problem.shtml?id=1000
"""

```
import bisect
from collections import defaultdict

# LeetCode 220. Contains Duplicate III (存在重复元素 III)
def contains_nearby_almost_duplicate(nums, k, t):
    """
    题目描述：
    给你一个整数数组 nums 和两个整数 k 和 t 。请你判断是否存在两个不同下标 i 和 j，
    使得 abs(nums[i] - nums[j]) <= t ，同时又满足 abs(i - j) <= k 。
    如果存在则返回 true，不存在返回 false。

    示例：
    输入：nums = [1,2,3,1], k = 3, t = 0
    输出：true

    输入：nums = [1,0,1,1], k = 1, t = 2
    输出：true

    输入：nums = [1,5,9,1,5,9], k = 2, t = 3
    输出：false

    约束条件：
    0 <= nums.length <= 2 * 10^4
    -2^31 <= nums[i] <= 2^31 - 1
```

```
    0 <= k <= 10^4
    0 <= t <= 2^31 - 1

    解题思路：
    使用滑动窗口维护一个大小为 k 的窗口，对于每个新元素，在窗口中查找是否存在一个元素
    与当前元素的差值不超过 t。利用 bisect 模块可以高效地找到最接近的元素。

    时间复杂度：O(n log min(n,k))，其中 n 是数组长度
    空间复杂度：O(min(n,k))，用于存储滑动窗口中的元素
    """
    # 使用列表维护滑动窗口，并保持有序
    window = []

    for i in range(len(nums)):
        # 查找大于等于当前元素的最小元素
        pos = bisect.bisect_left(window, nums[i])

        # 检查右侧元素
        if pos < len(window) and abs(window[pos] - nums[i]) <= t:
            return True

        # 检查左侧元素
        if pos > 0 and abs(window[pos-1] - nums[i]) <= t:
            return True

        # 添加当前元素到窗口
        bisect.insort(window, nums[i])

        # 维护滑动窗口大小为 k
        if len(window) > k:
            # 移除窗口中最旧的元素
            window.remove(nums[i - k])

    return False


# LeetCode 933. Number of Recent Calls (最近的请求次数)
class RecentCounter:
    """
    题目描述：
    写一个 RecentCounter 类来计算特定时间范围内最近的请求。
    请你实现 RecentCounter 类：
    RecentCounter() 初始化计数器，请求数为 0 。
```

int ping(int t) 在时间 t 添加一个新请求，其中 t 表示以毫秒为单位的某个时间，
并返回过去 3000 毫秒内发生的所有请求数（包括新请求）。
精确地说，返回在 [t-3000, t] 内发生的请求数。
保证每次对 ping 的调用都使用比之前更大的 t 值。

示例：
输入：
["RecentCounter", "ping", "ping", "ping", "ping"]
[[], [1], [100], [3001], [3002]]
输出：
[null, 1, 2, 3, 3]

约束条件：
1 <= t <= 10^9
保证每次对 ping 调用所使用的 t 值都 严格递增
至多调用 ping 方法 10^4 次

解题思路：
使用列表存储所有请求的时间戳，每次 ping 时，使用 bisect 查找[t-3000, t]范围内的请求数量。

时间复杂度：O(log n) 每次 ping 操作
空间复杂度：O(n) 存储所有请求
"""

```
    def __init__(self):
        self.requests = []

    def ping(self, t: int) -> int:
        self.requests.append(t)
        # 查找大于等于 t-3000 的第一个位置
        left = bisect.bisect_left(self.requests, t - 3000)
        # 查找大于 t 的最后一个位置
        right = bisect.bisect_right(self.requests, t)
        # 返回范围内的请求数量
        return right - left


# LeetCode 729. My Calendar I (我的日程安排表 I)
class MyCalendar:
    """
```

题目描述：

实现一个 MyCalendar 类来存放你的日程安排。如果要添加的日程安排不会造成 重复预订 ，则可以存储这个新的日程安排。

当两个日程安排有一些时间上的交叉时（例如两个日程安排都在同一时间内），就会产生 重复预订 。
日程可以用一对整数 start 和 end 表示，这里的时间是半开区间，即 [start, end)，
实数 x 的范围为 start <= x < end 。

实现 MyCalendar 类：
MyCalendar() 初始化日历对象。
boolean book(int start, int end) 如果可以将日程安排成功添加到日历中而不会导致重复预订，返回 true 。
否则，返回 false 并且不要将该日程安排添加到日历中。

示例：
输入：
["MyCalendar", "book", "book", "book"]
[[], [10, 20], [15, 25], [20, 30]]
输出：
[null, true, false, true]

约束条件：
0 <= start < end <= 10^9
每个测试用例，调用 book 方法的次数最多不超过 1000 次。

解题思路：
使用列表存储已预订的日程，并保持有序。
对于新的日程[start, end)，使用 bisect 查找可能冲突的日程：
1. 查找开始时间小于等于 start 的最大日程，检查其结束时间是否大于 start
2. 查找开始时间大于等于 start 的最小日程，检查其开始时间是否小于 end
如果没有冲突，则添加新日程。

时间复杂度：O(log n) 每次 book 操作
空间复杂度：O(n) 存储所有日程
"""

```
def __init__(self):
    self.calendar = []  # 存储 (start, end) 元组，并保持有序

def book(self, start: int, end: int) -> bool:
    # 查找插入位置
    pos = bisect.bisect_left(self.calendar, (start, end))

    # 检查与前一个日程是否冲突
    if pos > 0 and self.calendar[pos-1][1] > start:
        return False
```

```
        # 检查与后一个日程是否冲突
        if pos < len(self.calendar) and self.calendar[pos][0] < end:
            return False

        # 没有冲突，插入新日程
        self.calendar.insert(pos, (start, end))
        return True


# LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数)
def count_smaller(nums):
    """
    题目描述：
    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：
    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

    约束条件：
    1 <= nums.length <= 10^5
    -10^4 <= nums[i] <= 10^4

    解题思路：
    从右向左遍历数组，使用列表维护已遍历的元素并保持有序。
    对于每个元素，在列表中查找小于它的元素个数。

    时间复杂度：O(n log n)，其中 n 是数组长度
    空间复杂度：O(n)，用于存储列表
    """
    result = []
    sorted_list = []  # 保持有序的列表

    # 从右向左遍历
    for i in range(len(nums) - 1, -1, -1):
        # 查找小于当前元素的插入位置，即小于当前元素的元素个数
```

```
        pos = bisect.bisect_left(sorted_list, nums[i])
        result.append(pos)
        # 将当前元素插入到有序列表中
        bisect.insort(sorted_list, nums[i])

    # 反转结果数组
    return result[::-1]


# LeetCode 493. Reverse Pairs (翻转对)
def reverse_pairs(nums):
    """
    题目描述：
    给定一个数组 nums ，如果 i < j 且 nums[i] > 2*nums[j] 我们将 (i, j) 称作一个重要翻转对。
    你需要返回给定数组中的重要翻转对的数量。

    示例：
    输入: [1,3,2,3,1]
    输出: 2

    输入: [2,4,3,5,1]
    输出: 3

    约束条件：
    给定数组的长度不会超过 50000。
    输入数组中的所有数字都在 32 位整数的表示范围内。

    解题思路：
    使用列表维护已遍历的元素并保持有序，对于每个新元素，在列表中查找满足条件的元素个数。

    时间复杂度：O(n log n)，其中 n 是数组长度
    空间复杂度：O(n)，用于存储列表
    """
    count = 0
    sorted_list = []  # 保持有序的列表

    # 从右向左遍历
    for i in range(len(nums) - 1, -1, -1):
        # 查找小于 nums[i]/2.0 的元素个数
        pos = bisect.bisect_left(sorted_list, nums[i] / 2.0)
        count += pos
        # 将当前元素插入到有序列表中
        bisect.insort(sorted_list, nums[i])
```

```
    return count


# 测试代码
if __name__ == "__main__":
    # 测试存在重复元素 III
    print("测试存在重复元素 III:")
    nums = [1, 2, 3, 1]
    print(contains_nearby_almost_duplicate(nums, 3, 0))  # True

    # 测试最近的请求次数
    print("测试最近的请求次数:")
    counter = RecentCounter()
    print(counter.ping(1))     # 1
    print(counter.ping(100))   # 2
    print(counter.ping(3001))  # 3
    print(counter.ping(3002))  # 3

    # 测试我的日程安排表
    print("测试我的日程安排表:")
    calendar = MyCalendar()
    print(calendar.book(10, 20))  # True
    print(calendar.book(15, 25))  # False
    print(calendar.book(20, 30))  # True

    # 测试计算右侧小于当前元素的个数
    print("测试计算右侧小于当前元素的个数:")
    nums = [5, 2, 6, 1]
    result = count_smaller(nums)
    print(result)  # [2, 1, 1, 0]

    # 测试翻转对
    print("测试翻转对:")
    nums = [1, 3, 2, 3, 1]
    print(reverse_pairs(nums))  # 2
```

```
==============================================
```

文件: Code03_Comparator.cpp

```
==============================================
#include <iostream>
#include <vector>
```

```
#include <algorithm>
#include <string>
#include <map>

using namespace std;

/**
 * C++版本：Comparator 比较器相关题目与解析
 *
 * 在 C++中，我们使用 sort 函数和自定义比较函数来实现自定义排序规则
 * C++ Comparator 特点：使用 lambda 表达式或函数对象实现自定义排序
 * 时间复杂度：取决于具体实现，通常为 O(logN)到 O(NlogN)
 * 空间复杂度：取决于具体实现
 *
 * 相关平台题目：
 * 1. LeetCode 973. K Closest Points to Origin (最接近原点的 K 个点) -
https://leetcode.com/problems/k-closest-points-to-origin/
 * 2. LeetCode 179. Largest Number (最大数) - https://leetcode.com/problems/largest-number/
 * 3. LeetCode 1356. Sort Integers by The Number of 1 Bits (根据数字二进制下 1 的数目排序) -
https://leetcode.com/problems/sort-integers-by-the-number-of-1-bits/
 * 4. LeetCode 56. Merge Intervals (合并区间) - https://leetcode.com/problems/merge-intervals/
 * 5. LeetCode 1122. Relative Sort Array (数组的相对排序) -
https://leetcode.com/problems/relative-sort-array/
 * 6. LintCode 613. High Five (最高分五科) - https://www.lintcode.com/problem/high-five/
 * 7. HackerEarth Monk and the Magical Candy Bags (和尚与魔法糖果袋) -
https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/practice-
problems/algorithm/monk-and-the-magical-candy-bags/
 * 8. CodeChef FRGTNLNG (遗忘的语言) - https://www.codechef.com/problems/FRGTNLNG
 * 9. SPOJ DICT (字典) - https://www.spoj.com/problems/DICT/
 * 10. Project Euler Problem 2: Even Fibonacci numbers (偶数斐波那契数) -
https://projecteuler.net/problem=2
 * 11. HackerRank Maximum Palindromes (最大回文) - https://www.hackerrank.com/challenges/maximum-
palindromes
 * 12. 计蒜客 T1101: 阶乘 (阶乘) - https://www.jisuanke.com/t/T1101
 * 13. 杭电 OJ 1003: Max Sum (最大子序列和) - http://acm.hdu.edu.cn/showproblem.php?pid=1003
 * 14. 牛客网 剑指 Offer 50: 第一个只出现一次的字符 (第一个只出现一次的字符) -
https://www.nowcoder.com/practice/1c82e8cf713b4bbeb2a5b31cf5b0417c
 * 15. acwing 800. 数组元素的目标和 (数组元素的目标和) -
https://www.acwing.com/problem/content/802/
 * 16. POJ 1003: Hangover (悬垂) - http://poj.org/problem?id=1003
 * 17. UVa OJ 101: The Blocks Problem (积木问题) -
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=37
 * 18. Timus OJ 1005: Stone Pile (石子堆) - https://acm.timus.ru/problem.aspx?space=1&num=1005
```

 * 19. Aizu OJ ALDS1_5_A: Exhaustive Search (穷举搜索) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_5_A
 * 20. Comet OJ Contest #1: 热身赛 B. 简单的数学题 (简单的数学题) - https://cometoj.com/contest/1/problem/B
 * 21. MarsCode 火星编程竞赛: 数字统计 (数字统计) - https://www.marscode.cn/contest/1/problem/1002
 * 22. ZOJ 1002: Fire Net (消防网) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1002
 * 23. LOJ 101: 最小生成树 (最小生成树) - https://loj.ac/p/101
 * 24. 各大高校 OJ: 北京大学 OJ 1000: A+B Problem (A+B 问题) - http://poj.org/problem?id=1000
 * 25. LeetCode 352. Data Stream as Disjoint Intervals (数据流变为不相交区间) - https://leetcode.com/problems/data-stream-as-disjoint-intervals/
 * 26. Codeforces 122A. Lucky Division (幸运分割) - https://codeforces.com/problemset/problem/122/A
 * 27. AtCoder ABC 218 C - Shapes (形状) - https://atcoder.jp/contests/abc218/tasks/abc218_c
 * 28. USACO Bronze: Block Game (积木游戏) - http://www.usaco.org/index.php?page=viewproblem2&cpid=664
 * 29. 洛谷 P3366 【模板】最小生成树 (模板最小生成树) - https://www.luogu.com.cn/problem/P3366
 * 30. LeetCode 149. Max Points on a Line (直线上最多的点数) - https://leetcode.com/problems/max-points-on-a-line/
 * 31. LeetCode 215. Kth Largest Element in an Array (数组中的第 K 个最大元素) - https://leetcode.com/problems/kth-largest-element-in-an-array/
 * 32. LeetCode 295. Find Median from Data Stream (数据流的中位数) - https://leetcode.com/problems/find-median-from-data-stream/
 * 33. LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数) - https://leetcode.com/problems/count-of-smaller-numbers-after-self/
 * 34. LeetCode 327. Count of Range Sum (区间和的个数) - https://leetcode.com/problems/count-of-range-sum/
 * 35. LeetCode 350. Intersection of Two Arrays II (两个数组的交集 II) - https://leetcode.com/problems/intersection-of-two-arrays-ii/
 * 36. LeetCode 148. Sort List (排序链表) - https://leetcode.com/problems/sort-list/
 * 37. LeetCode 242. Valid Anagram (有效的字母异位词) - https://leetcode.com/problems/valid-anagram/
 * 38. LeetCode 347. Top K Frequent Elements (前 K 个高频元素) - https://leetcode.com/problems/top-k-frequent-elements/
 * 39. LeetCode 451. Sort Characters By Frequency (根据字符出现频率排序) - https://leetcode.com/problems/sort-characters-by-frequency/
 * 40. LeetCode 493. Reverse Pairs (翻转对) - https://leetcode.com/problems/reverse-pairs/
 * 41. LeetCode 539. Minimum Time Difference (最小时间差) - https://leetcode.com/problems/minimum-time-difference/
 * 42. LeetCode 791. Custom Sort String (自定义字符串排序) - https://leetcode.com/problems/custom-sort-string/
 * 43. LeetCode 937. Reorder Data in Log Files (重新排列日志文件) -

```
https://leetcode.com/problems/reorder-data-in-log-files/
 * 44. LeetCode 1030. Matrix Cells in Distance Order (距离顺序排列矩阵单元格) -
https://leetcode.com/problems/matrix-cells-in-distance-order/
 * 45. LeetCode 1636. Sort Array by Increasing Frequency (按照频率将数组升序排序) -
https://leetcode.com/problems/sort-array-by-increasing-frequency/
 * 46. LeetCode 1710. Maximum Units on a Truck (卡车上的最大单元数) -
https://leetcode.com/problems/maximum-units-on-a-truck/
 * 47. HackerRank Java Comparator (Java比较器) - https://www.hackerrank.com/challenges/java-
comparator/problem
 * 48. LeetCode 147. Insertion Sort List (对链表进行插入排序) -
https://leetcode.com/problems/insertion-sort-list/
 * 49. LeetCode 252. Meeting Rooms (会议室) - https://leetcode.com/problems/meeting-rooms/
 * 50. LeetCode 253. Meeting Rooms II (会议室 II) - https://leetcode.com/problems/meeting-rooms-
ii/
 */

// LeetCode 973. K Closest Points to Origin (最接近原点的 K 个点)
vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {
    // 使用自定义比较函数按距离排序
    sort(points.begin(), points.end(), [](const vector<int>& a, const vector<int>& b) {
        // 计算到原点距离的平方
        int distA = a[0] * a[0] + a[1] * a[1];
        int distB = b[0] * b[0] + b[1] * b[1];
        return distA < distB;
    });

    // 返回前 k 个点
    return vector<vector<int>>(points.begin(), points.begin() + k);
}

// LeetCode 179. Largest Number (最大数)
string largestNumber(vector<int>& nums) {
    // 转换为字符串数组
    vector<string> strs(nums.size());
    for (int i = 0; i < nums.size(); i++) {
        strs[i] = to_string(nums[i]);
    }

    // 使用自定义比较函数排序
    sort(strs.begin(), strs.end(), [](const string& a, const string& b) {
        // 比较 b+a 和 a+b 的大小，注意是降序排列所以顺序颠倒
        return a + b > b + a;
    });
```

```
    // 处理全为0的特殊情况
    if (strs[0] == "0") {
        return "0";
    }

    // 拼接结果
    string result;
    for (const string& str : strs) {
        result += str;
    }

    return result;
}

// LeetCode 1356. Sort Integers by The Number of 1 Bits (根据数字二进制下 1 的数目排序)
vector<int> sortByBits(vector<int>& arr) {
    // 使用自定义比较函数排序
    sort(arr.begin(), arr.end(), [](int a, int b) {
        // 计算二进制中1的位数
        int bitCountA = __builtin_popcount(a);
        int bitCountB = __builtin_popcount(b);

        // 如果1的位数相同，按数值大小排序；否则按1的位数排序
        if (bitCountA == bitCountB) {
            return a < b;
        } else {
            return bitCountA < bitCountB;
        }
    });

    return arr;
}

// Player类用于HackerRank Java Comparator示例
class Player {
public:
    string name;
    int score;

    Player(string n, int s) : name(n), score(s) {}
};
```

```
// Player 比较函数
bool playerComparator(const Player& a, const Player& b) {
    // 首先按分数降序排列
    if (a.score != b.score) {
        return a.score > b.score;
    }
    // 如果分数相同，按名字升序排列
    return a.name < b.name;
}

// LeetCode 56. Merge Intervals (合并区间)
vector<vector<int>> merge(vector<vector<int>>& intervals) {
    if (intervals.size() <= 1) {
        return intervals;
    }

    // 按区间的起始位置排序
    sort(intervals.begin(), intervals.end(), [](const vector<int>& a, const vector<int>& b) {
        return a[0] < b[0];
    });

    vector<vector<int>> result;
    vector<int> current = intervals[0];
    result.push_back(current);

    for (int i = 1; i < intervals.size(); i++) {
        vector<int> interval = intervals[i];
        // 如果当前区间与下一个区间重叠，合并它们
        if (interval[0] <= current[1]) {
            current[1] = max(current[1], interval[1]);
        } else {
            // 否则，将下一个区间添加到结果中
            current = interval;
            result.push_back(current);
        }
    }

    return result;
}

// LeetCode 1122. Relative Sort Array (数组的相对排序)
vector<int> relativeSortArray(vector<int>& arr1, vector<int>& arr2) {
    // 创建 arr2 中元素到索引的映射
```

```
    map<int, int> indexMap;
    for (int i = 0; i < arr2.size(); i++) {
        indexMap[arr2[i]] = i;
    }

    // 使用自定义比较函数排序
    sort(arr1.begin(), arr1.end(), [&indexMap](int a, int b) {
        // 如果两个元素都在 arr2 中，按它们在 arr2 中的索引排序
        if (indexMap.find(a) != indexMap.end() && indexMap.find(b) != indexMap.end()) {
            return indexMap[a] < indexMap[b];
        }
        // 如果只有 a 在 arr2 中，a 排在前面
        if (indexMap.find(a) != indexMap.end()) {
            return true;
        }
        // 如果只有 b 在 arr2 中，b 排在前面
        if (indexMap.find(b) != indexMap.end()) {
            return false;
        }
        // 如果两个元素都不在 arr2 中，按数值大小排序
        return a < b;
    });

    return arr1;
}

int main() {
    // 测试最接近原点的 K 个点
    cout << "测试最接近原点的 K 个点:" << endl;
    vector<vector<int>> points = {{1,1},{3,3},{2,2}};
    vector<vector<int>> result1 = kClosest(points, 2);
    for (const auto& point : result1) {
        cout << "[" << point[0] << ", " << point[1] << "]" << endl;
    }

    // 测试最大数
    cout << "测试最大数:" << endl;
    vector<int> nums = {3,30,34,5,9};
    cout << largestNumber(nums) << endl;

    // 测试根据数字二进制下 1 的数目排序
    cout << "测试根据数字二进制下 1 的数目排序:" << endl;
    vector<int> arr = {0,1,2,3,4,5,6,7,8};
```

```
vector<int> result2 = sortByBits(arr);
cout << "[";
for (int i = 0; i < result2.size(); i++) {
    cout << result2[i];
    if (i < result2.size() - 1) {
        cout << ", ";
    }
}
cout << "]" << endl;

// 测试 HackerRank Java Comparator
cout << "测试 HackerRank Java Comparator:" << endl;
vector<Player> players = {
    Player("amy", 100),
    Player("david", 100),
    Player("heraldo", 50),
    Player("aakansha", 75),
    Player("aleksa", 150)
};
sort(players.begin(), players.end(), playerComparator);
for (const Player& player : players) {
    cout << player.name << " " << player.score << endl;
}

// 测试合并区间
cout << "测试合并区间:" << endl;
vector<vector<int>> intervals = {{1,3},{2,6},{8,10},{15,18}};
vector<vector<int>> merged = merge(intervals);
for (const auto& interval : merged) {
    cout << "[" << interval[0] << ", " << interval[1] << "]" << endl;
}

// 测试数组的相对排序
cout << "测试数组的相对排序:" << endl;
vector<int> arr1 = {2,3,1,3,2,4,6,7,9,2,19};
vector<int> arr2 = {2,1,4,3,9,6};
vector<int> result3 = relativeSortArray(arr1, arr2);
cout << "[";
for (int i = 0; i < result3.size(); i++) {
    cout << result3[i];
    if (i < result3.size() - 1) {
        cout << ", ";
    }
```

```
    }
    cout << "]" << endl;

    return 0;
}
```

==========

文件: Code03_Comparator.java
==========

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.TreeSet;

/**
 * Comparator 比较器相关题目与解析
 *
 * Comparator 是一个比较器接口，用于定义对象之间比较的规则。
 * 在 TreeSet 和 TreeMap 等有序集合中，可以通过自定义 Comparator 实现自定义排序规则。
 * Comparator 特点：可以实现复杂的排序逻辑，支持多种排序条件
 * 时间复杂度：取决于具体实现，通常为 O(logN)到 O(NlogN)
 * 空间复杂度：取决于具体实现
 *
 * 相关平台题目：
 * 1. LeetCode 148. Sort List (排序链表) - https://leetcode.com/problems/sort-list/
 * 2. LeetCode 179. Largest Number (最大数) - https://leetcode.com/problems/largest-number/
 * 3. LeetCode 242. Valid Anagram (有效的字母异位词) - https://leetcode.com/problems/valid-anagram/
 * 4. LeetCode 347. Top K Frequent Elements (前 K 个高频元素) - https://leetcode.com/problems/top-k-frequent-elements/
 * 5. LeetCode 451. Sort Characters By Frequency (根据字符出现频率排序) - https://leetcode.com/problems/sort-characters-by-frequency/
 * 6. LeetCode 493. Reverse Pairs (翻转对) - https://leetcode.com/problems/reverse-pairs/
 * 7. LeetCode 539. Minimum Time Difference (最小时间差) - https://leetcode.com/problems/minimum-time-difference/
 * 8. LeetCode 791. Custom Sort String (自定义字符串排序) - https://leetcode.com/problems/custom-sort-string/
 * 9. LeetCode 937. Reorder Data in Log Files (重新排列日志文件) -
```

https://leetcode.com/problems/reorder-data-in-log-files/
 * 10. LeetCode 973. K Closest Points to Origin (最接近原点的 K 个点) - https://leetcode.com/problems/k-closest-points-to-origin/
 * 11. LeetCode 1030. Matrix Cells in Distance Order (距离顺序排列矩阵单元格) - https://leetcode.com/problems/matrix-cells-in-distance-order/
 * 12. LeetCode 1122. Relative Sort Array (数组的相对排序) - https://leetcode.com/problems/relative-sort-array/
 * 13. LeetCode 1356. Sort Integers by The Number of 1 Bits (根据数字二进制下 1 的数目排序) - https://leetcode.com/problems/sort-integers-by-the-number-of-1-bits/
 * 14. LeetCode 1636. Sort Array by Increasing Frequency (按照频率将数组升序排序) - https://leetcode.com/problems/sort-array-by-increasing-frequency/
 * 15. LeetCode 1710. Maximum Units on a Truck (卡车上的最大单元数) - https://leetcode.com/problems/maximum-units-on-a-truck/
 * 16. HackerRank Java Comparator (Java 比较器) - https://www.hackerrank.com/challenges/java-comparator/problem
 * 17. LeetCode 147. Insertion Sort List (对链表进行插入排序) - https://leetcode.com/problems/insertion-sort-list/
 * 18. LeetCode 252. Meeting Rooms (会议室) - https://leetcode.com/problems/meeting-rooms/
 * 19. LeetCode 253. Meeting Rooms II (会议室 II) - https://leetcode.com/problems/meeting-rooms-ii/
 * 20. LeetCode 56. Merge Intervals (合并区间) - https://leetcode.com/problems/merge-intervals/
 * 21. LintCode 613. High Five (最高分五科) - https://www.lintcode.com/problem/high-five/
 * 22. HackerEarth Monk and the Magical Candy Bags (和尚与魔法糖果袋) - https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/practice-problems/algorithm/monk-and-the-magical-candy-bags/
 * 23. CodeChef FRGTNLNG (遗忘的语言) - https://www.codechef.com/problems/FRGTNLNG
 * 24. SPOJ DICT (字典) - https://www.spoj.com/problems/DICT/
 * 25. Project Euler Problem 2: Even Fibonacci numbers (偶数斐波那契数) - https://projecteuler.net/problem=2
 * 26. HackerRank Maximum Palindromes (最大回文) - https://www.hackerrank.com/challenges/maximum-palindromes
 * 27. 计蒜客 T1101: 阶乘 (阶乘) - https://www.jisuanke.com/t/T1101
 * 28. 杭电 OJ 1003: Max Sum (最大子序列和) - http://acm.hdu.edu.cn/showproblem.php?pid=1003
 * 29. 牛客网 剑指 Offer 50: 第一个只出现一次的字符 (第一个只出现一次的字符) - https://www.nowcoder.com/practice/1c82e8cf713b4bbeb2a5b31cf5b0417c
 * 30. acwing 800. 数组元素的目标和 (数组元素的目标和) - https://www.acwing.com/problem/content/802/
 * 31. POJ 1003: Hangover (悬垂) - http://poj.org/problem?id=1003
 * 32. UVa OJ 101: The Blocks Problem (积木问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=37
 * 33. Timus OJ 1005: Stone Pile (石子堆) - https://acm.timus.ru/problem.aspx?space=1&num=1005
 * 34. Aizu OJ ALDS1_5_A: Exhaustive Search (穷举搜索) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_5_A

 * 35. Comet OJ Contest #1: 热身赛 B. 简单的数学题 (简单的数学题) -
https://cometoj.com/contest/1/problem/B
 * 36. MarsCode 火星编程竞赛: 数字统计 (数字统计) -
https://www.marscode.cn/contest/1/problem/1002
 * 37. ZOJ 1002: Fire Net (消防网) -
http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1002
 * 38. LOJ 101: 最小生成树 (最小生成树) - https://loj.ac/p/101
 * 39. 各大高校 OJ: 北京大学 OJ 1000: A+B Problem (A+B 问题) - http://poj.org/problem?id=1000
 * 40. LeetCode 352. Data Stream as Disjoint Intervals (数据流变为不相交区间) -
https://leetcode.com/problems/data-stream-as-disjoint-intervals/
 * 41. Codeforces 122A. Lucky Division (幸运分割) -
https://codeforces.com/problemset/problem/122/A
 * 42. AtCoder ABC 218 C - Shapes (形状) - https://atcoder.jp/contests/abc218/tasks/abc218_c
 * 43. USACO Bronze: Block Game (积木游戏) -
http://www.usaco.org/index.php?page=viewproblem2&cpid=664
 * 44. 洛谷 P3366 【模板】最小生成树 (模板最小生成树) - https://www.luogu.com.cn/problem/P3366
 * 45. LeetCode 149. Max Points on a Line (直线上最多的点数) - https://leetcode.com/problems/max-points-on-a-line/
 * 46. LeetCode 215. Kth Largest Element in an Array (数组中的第 K 个最大元素) -
https://leetcode.com/problems/kth-largest-element-in-an-array/
 * 47. LeetCode 295. Find Median from Data Stream (数据流的中位数) -
https://leetcode.com/problems/find-median-from-data-stream/
 * 48. LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数) -
https://leetcode.com/problems/count-of-smaller-numbers-after-self/
 * 49. LeetCode 327. Count of Range Sum (区间和的个数) - https://leetcode.com/problems/count-of-range-sum/
 * 50. LeetCode 350. Intersection of Two Arrays II (两个数组的交集 II) -
https://leetcode.com/problems/intersection-of-two-arrays-ii/
 */

public class Code03_Comparator {

    /**
     * LeetCode 973. K Closest Points to Origin (最接近原点的 K 个点)
     *
     * 题目描述：
     * 给定一个数组 points ，其中 points[i] = [xi, yi] 表示 X-Y 平面上的一个点，并且是一个整数 k ，
     * 返回离原点 (0,0) 最近的 k 个点。
     * 这里，平面上两点之间的距离是 欧几里德距离（ √(x1 - x2)^2 + (y1 - y2)^2 ）。
     * 你可以按 任何顺序 返回答案。除了点坐标的顺序之外，答案 确保 是 唯一 的。
     *
     * 示例：

```
     * 输入：points = [[1,1],[3,3],[2,2]], k = 2
     * 输出：[[1,1],[2,2]]
     *
     * 输入：points = [[3,3],[5,-1],[-2,4]], k = 2
     * 输出：[[3,3],[-2,4]]
     * （答案 [[-2,4],[3,3]] 也会被接受。）
     *
     * 约束条件：
     * 1 <= k <= points.length <= 10^4
     * -10^4 < xi, yi < 10^4
     *
     * 解题思路：
     * 计算每个点到原点的距离的平方（避免开方运算），然后使用自定义 Comparator 按距离排序，
     * 最后取前 k 个点。
     *
     * 时间复杂度：O(n log n)，其中 n 是点的数量
     * 空间复杂度：O(1)，如果不考虑输出数组的空间
     *
     * @param points 点坐标数组
     * @param k 需要返回的点数量
     * @return 最接近原点的 k 个点
     */
    public static int[][] kClosest(int[][] points, int k) {
        // 使用自定义 Comparator 按距离排序
        Arrays.sort(points, new Comparator<int[]>() {
            @Override
            public int compare(int[] a, int[] b) {
                // 计算到原点距离的平方
                int distA = a[0] * a[0] + a[1] * a[1];
                int distB = b[0] * b[0] + b[1] * b[1];
                return distA - distB;
            }
        });

        // 返回前 k 个点
        return Arrays.copyOfRange(points, 0, k);
    }

    /**
     * LeetCode 179. Largest Number (最大数)
     *
     * 题目描述：
     * 给定一组非负整数 nums，重新排列每个数的顺序（每个数不可拆分）使之组成一个最大的整数。
```

```
     * 注意：输出结果可能非常大，所以你需要返回一个字符串而不是整数。
     *
     * 示例：
     * 输入：nums = [10,2]
     * 输出："210"
     *
     * 输入：nums = [3,30,34,5,9]
     * 输出："9534330"
     *
     * 约束条件：
     * 1 <= nums.length <= 100
     * 0 <= nums[i] <= 10^9
     *
     * 解题思路：
     * 将整数数组转换为字符串数组，然后使用自定义 Comparator 排序。
     * 对于两个字符串 a 和 b，比较 a+b 和 b+a 的大小，决定它们的顺序。
     *
     * 时间复杂度：O(n log n * m)，其中 n 是数组长度，m 是数字的平均长度
     * 空间复杂度：O(n * m)
     *
     * @param nums 非负整数数组
     * @return 组成的最大整数字符串
     */
    public static String largestNumber(int[] nums) {
        // 转换为字符串数组
        String[] strs = new String[nums.length];
        for (int i = 0; i < nums.length; i++) {
            strs[i] = String.valueOf(nums[i]);
        }

        // 使用自定义 Comparator 排序
        Arrays.sort(strs, new Comparator<String>() {
            @Override
            public int compare(String a, String b) {
                // 比较 b+a 和 a+b 的大小，注意是降序排列所以顺序颠倒
                String order1 = a + b;
                String order2 = b + a;
                return order2.compareTo(order1);
            }
        });

        // 处理全为 0 的特殊情况
        if (strs[0].equals("0")) {
```

```
            return "0";
        }

        // 拼接结果
        StringBuilder sb = new StringBuilder();
        for (String str : strs) {
            sb.append(str);
        }

        return sb.toString();
    }

    /**
     * LeetCode 1356. Sort Integers by The Number of 1 Bits (根据数字二进制下 1 的数目排序)
     *
     * 题目描述：
     * 给你一个整数数组 arr 。请你将数组中的元素按照其二进制表示中数字 1 的数目升序排序。
     * 如果存在多个数字二进制中 1 的数目相同，则必须将它们按照数值大小升序排列。
     * 请你返回排序后的数组。
     *
     * 示例：
     * 输入：arr = [0,1,2,3,4,5,6,7,8]
     * 输出：[0,1,2,4,8,3,5,6,7]
     *
     * 输入：arr = [1024,512,256,128,64,32,16,8,4,2,1]
     * 输出：[1,2,4,8,16,32,64,128,256,512,1024]
     *
     * 约束条件：
     * 1 <= arr.length <= 500
     * 0 <= arr[i] <= 10^4
     *
     * 解题思路：
     * 使用自定义 Comparator 排序，首先按二进制中 1 的位数排序，如果相同则按数值大小排序。
     * 计算二进制中 1 的位数可以使用 Integer.bitCount() 方法。
     *
     * 时间复杂度：O(n log n)
     * 空间复杂度：O(1)
     *
     * @param arr 整数数组
     * @return 按照规则排序后的数组
     */
    public static int[] sortByBits(int[] arr) {
        // 转换为 Integer 数组以便使用自定义 Comparator
```

```
    Integer[] nums = new Integer[arr.length];
    for (int i = 0; i < arr.length; i++) {
        nums[i] = arr[i];
    }

    // 使用自定义 Comparator 排序
    Arrays.sort(nums, new Comparator<Integer>() {
        @Override
        public int compare(Integer a, Integer b) {
            // 计算二进制中 1 的位数
            int bitCountA = Integer.bitCount(a);
            int bitCountB = Integer.bitCount(b);

            // 如果 1 的位数相同，按数值大小排序；否则按 1 的位数排序
            if (bitCountA == bitCountB) {
                return a - b;
            } else {
                return bitCountA - bitCountB;
            }
        }
    });

    // 转换回 int 数组
    for (int i = 0; i < arr.length; i++) {
        arr[i] = nums[i];
    }

    return arr;
}

/**
 * HackerRank Java Comparator (Java 比较器)
 *
 * 题目描述：
 * 创建一个比较器，根据玩家的分数降序排列，如果分数相同则按名字升序排列。
 *
 * 示例：
 * 输入：
 * 5
 * amy 100
 * david 100
 * heraldo 50
 * aakansha 75
```

```
 * aleksa 150
 *
 * 输出：
 * aleksa 150
 * amy 100
 * david 100
 * aakansha 75
 * heraldo 50
 *
 * 解题思路：
 * 实现一个 Comparator 接口，首先比较分数（降序），如果分数相同则比较名字（升序）。
 *
 * 时间复杂度：O(n log n)，其中 n 是玩家数量
 * 空间复杂度：O(1)
 */
static class Player {
    String name;
    int score;

    Player(String name, int score) {
        this.name = name;
        this.score = score;
    }
}

static class Checker implements Comparator<Player> {
    @Override
    public int compare(Player a, Player b) {
        // 首先按分数降序排列
        if (a.score != b.score) {
            return b.score - a.score;
        }
        // 如果分数相同，按名字升序排列
        return a.name.compareTo(b.name);
    }
}

/**
 * LeetCode 56. Merge Intervals (合并区间)
 *
 * 题目描述：
 * 以数组 intervals 表示若干个区间的集合，其中单个区间为 intervals[i] = [starti, endi] 。
 * 请你合并所有重叠的区间，并返回一个不重叠的区间数组，该数组需恰好覆盖输入中的所有区间。
```

```
     *
     * 示例：
     * 输入：intervals = [[1,3],[2,6],[8,10],[15,18]]
     * 输出：[[1,6],[8,10],[15,18]]
     * 解释：区间 [1,3] 和 [2,6] 重叠, 将它们合并为 [1,6].
     *
     * 输入：intervals = [[1,4],[4,5]]
     * 输出：[[1,5]]
     * 解释：区间 [1,4] 和 [4,5] 可被视为重叠区间。
     *
     * 约束条件：
     * 1 <= intervals.length <= 10^4
     * intervals[i].length == 2
     * 0 <= starti <= endi <= 10^4
     *
     * 解题思路：
     * 1. 首先按区间的起始位置排序
     * 2. 遍历排序后的区间，合并重叠的区间
     *
     * 时间复杂度：O(n log n)，其中 n 是区间数量
     * 空间复杂度：O(n)，用于存储结果
     *
     * @param intervals 区间数组
     * @return 合并后的区间数组
     */
    public static int[][] merge(int[][] intervals) {
        if (intervals.length <= 1) {
            return intervals;
        }

        // 按区间的起始位置排序
        Arrays.sort(intervals, new Comparator<int[]>() {
            @Override
            public int compare(int[] a, int[] b) {
                return a[0] - b[0];
            }
        });

        List<int[]> result = new ArrayList<>();
        int[] current = intervals[0];
        result.add(current);

        for (int i = 1; i < intervals.length; i++) {
```

```
            int[] interval = intervals[i];
            // 如果当前区间与下一个区间重叠，合并它们
            if (interval[0] <= current[1]) {
                current[1] = Math.max(current[1], interval[1]);
            } else {
                // 否则，将下一个区间添加到结果中
                current = interval;
                result.add(current);
            }
        }

        return result.toArray(new int[result.size()][]);
    }

    /**
     * LeetCode 1122. Relative Sort Array (数组的相对排序)
     *
     * 题目描述：
     * 给你两个数组，arr1 和 arr2，arr2 中的元素各不相同，arr2 中的每个元素都出现在 arr1 中。
     * 对 arr1 中的元素进行排序，使 arr1 中项的相对顺序和 arr2 中的相对顺序相同。
     * 未在 arr2 中出现过的元素需要按照升序放在 arr1 的末尾。
     *
     * 示例：
     * 输入：arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]
     * 输出：[2,2,2,1,4,3,3,9,6,7,19]
     *
     * 约束条件：
     * 1 <= arr1.length, arr2.length <= 1000
     * 0 <= arr1[i], arr2[i] <= 1000
     * arr2 中的元素 arr2[i] 各不相同
     * arr2 中的每个元素 arr2[i] 都出现在 arr1 中
     *
     * 解题思路：
     * 1. 使用自定义 Comparator 排序
     * 2. 对于在 arr2 中出现的元素，按其在 arr2 中的索引排序
     * 3. 对于不在 arr2 中出现的元素，按数值大小升序排序，并放在末尾
     *
     * 时间复杂度：O(m log m + n)，其中 m 是 arr1 的长度，n 是 arr2 的长度
     * 空间复杂度：O(n)，用于存储 arr2 中元素的索引映射
     *
     * @param arr1 数组 1
     * @param arr2 数组 2
     * @return 相对排序后的数组
```

```
 */
public static int[] relativeSortArray(int[] arr1, int[] arr2) {
    // 创建 arr2 中元素到索引的映射
    int[] indexMap = new int[1001];
    Arrays.fill(indexMap, -1);
    for (int i = 0; i < arr2.length; i++) {
        indexMap[arr2[i]] = i;
    }

    // 转换为 Integer 数组以便使用自定义 Comparator
    Integer[] nums = new Integer[arr1.length];
    for (int i = 0; i < arr1.length; i++) {
        nums[i] = arr1[i];
    }

    // 使用自定义 Comparator 排序
    Arrays.sort(nums, new Comparator<Integer>() {
        @Override
        public int compare(Integer a, Integer b) {
            // 如果两个元素都在 arr2 中，按它们在 arr2 中的索引排序
            if (indexMap[a] != -1 && indexMap[b] != -1) {
                return indexMap[a] - indexMap[b];
            }
            // 如果只有 a 在 arr2 中，a 排在前面
            if (indexMap[a] != -1) {
                return -1;
            }
            // 如果只有 b 在 arr2 中，b 排在前面
            if (indexMap[b] != -1) {
                return 1;
            }
            // 如果两个元素都不在 arr2 中，按数值大小排序
            return a - b;
        }
    });

    // 转换回 int 数组
    for (int i = 0; i < arr1.length; i++) {
        arr1[i] = nums[i];
    }

    return arr1;
}
```

```
public static class Employee {
    public int company;
    public int age;

    public Employee(int c, int a) {
        company = c;
        age = a;
    }
}

public static class EmployeeComparator implements Comparator<Employee> {

    @Override
    public int compare(Employee o1, Employee o2) {
        // 任何比较器都默认
        // 如果返回负数认为 o1 的优先级更高
        // 如果返回正数认为 o2 的优先级更高
        // 任何比较器都是这样，所以利用这个设定，可以定制优先级怎么确定，也就是怎么比较
        // 不再有大小的概念，就是优先级的概念
        return o1.age - o2.age;
    }

}

public static void main(String[] args) {
    Employee s1 = new Employee(2, 27);
    Employee s2 = new Employee(1, 60);
    Employee s3 = new Employee(4, 19);
    Employee s4 = new Employee(3, 23);
    Employee s5 = new Employee(1, 35);
    Employee s6 = new Employee(3, 55);
    Employee[] arr = { s1, s2, s3, s4, s5, s6 };
    Arrays.sort(arr, new EmployeeComparator());
    for (Employee e : arr) {
        System.out.println(e.company + " , " + e.age);
    }

    System.out.println("=====");

    Arrays.sort(arr, (a, b) -> b.age - a.age);
    for (Employee e : arr) {
        System.out.println(e.company + " , " + e.age);
```

```
        }

        System.out.println("=====");
        // 所有员工，先按照谁的公司编号小，谁在前；如果公司编号一样，谁年龄小谁在前
        Arrays.sort(arr, (a, b) -> a.company != b.company ? (a.company - b.company) : (a.age -
b.age));
        for (Employee e : arr) {
            System.out.println(e.company + " , " + e.age);
        }

        TreeSet<Employee> treeSet1 = new TreeSet<>(new EmployeeComparator());
        for (Employee e : arr) {
            treeSet1.add(e);
        }
        System.out.println(treeSet1.size());

        // 会去重
        treeSet1.add(new Employee(2, 27));
        System.out.println(treeSet1.size());

        System.out.println("===");

        // 如果不想去重，就需要增加更多的比较
        // 比如对象的内存地址、或者如果对象有数组下标之类的独特信息
        TreeSet<Employee> treeSet2 = new TreeSet<>((a, b) -> a.company != b.company ? (a.company -
b.company)
                : a.age != b.age ? (a.age - b.age) : a.toString().compareTo(b.toString()));
        for (Employee e : arr) {
            treeSet2.add(e);
        }
        System.out.println(treeSet2.size());

        // 不会去重
        treeSet2.add(new Employee(2, 27));
        System.out.println(treeSet2.size());

        System.out.println("===");

        // PriorityQueue 不会去重，不再展示了

        // 字典序
        String str1 = "abcde";
        String str2 = "ks";
```

```
System.out.println(str1.compareTo(str2));
System.out.println(str2.compareTo(str1));

// 测试添加的题目
System.out.println("==========");
System.out.println("测试最接近原点的 K 个点:");
int[][] points1 = {{1,1},{3,3},{2,2}};
int[][] result1 = kClosest(points1, 2);
for (int[] point : result1) {
    System.out.println(Arrays.toString(point));
}

System.out.println("测试最大数:");
int[] nums1 = {3,30,34,5,9};
System.out.println(largestNumber(nums1));

System.out.println("测试根据数字二进制下 1 的数目排序:");
int[] arr1 = {0,1,2,3,4,5,6,7,8};
int[] result2 = sortByBits(arr1);
System.out.println(Arrays.toString(result2));

// 测试 HackerRank Java Comparator
System.out.println("测试 HackerRank Java Comparator:");
Player[] players = {
    new Player("amy", 100),
    new Player("david", 100),
    new Player("heraldo", 50),
    new Player("aakansha", 75),
    new Player("aleksa", 150)
};
Checker checker = new Checker();
Arrays.sort(players, checker);
for (Player player : players) {
    System.out.println(player.name + " " + player.score);
}

// 测试合并区间
System.out.println("测试合并区间:");
int[][] intervals = {{1,3},{2,6},{8,10},{15,18}};
int[][] merged = merge(intervals);
for (int[] interval : merged) {
    System.out.println(Arrays.toString(interval));
}
```

```
        // 测试数组的相对排序
        System.out.println("测试数组的相对排序:");
        int[] arr2 = {2,3,1,3,2,4,6,7,9,2,19};
        int[] arr3 = {2,1,4,3,9,6};
        int[] relativeSorted = relativeSortArray(arr2, arr3);
        System.out.println(Arrays.toString(relativeSorted));
    }
}
```

==========

文件: Code03_Comparator.py

==========

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
```

Python 版本：Comparator 比较器相关题目与解析

在 Python 中，我们使用 sorted()函数和 key 参数来实现自定义排序规则
Python Comparator 特点：使用 key 函数或 cmp_to_key 转换比较函数实现自定义排序
时间复杂度：取决于具体实现，通常为 O(logN)到 O(NlogN)
空间复杂度：取决于具体实现

相关平台题目：
1. LeetCode 973. K Closest Points to Origin (最接近原点的 K 个点) - https://leetcode.com/problems/k-closest-points-to-origin/
2. LeetCode 179. Largest Number (最大数) - https://leetcode.com/problems/largest-number/
3. LeetCode 1356. Sort Integers by The Number of 1 Bits (根据数字二进制下 1 的数目排序) - https://leetcode.com/problems/sort-integers-by-the-number-of-1-bits/
4. LeetCode 56. Merge Intervals (合并区间) - https://leetcode.com/problems/merge-intervals/
5. LeetCode 1122. Relative Sort Array (数组的相对排序) - https://leetcode.com/problems/relative-sort-array/
6. LintCode 613. High Five (最高分五科) - https://www.lintcode.com/problem/high-five/
7. HackerEarth Monk and the Magical Candy Bags (和尚与魔法糖果袋) - https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/practice-problems/algorithm/monk-and-the-magical-candy-bags/
8. CodeChef FRGTNLNG (遗忘的语言) - https://www.codechef.com/problems/FRGTNLNG
9. SPOJ DICT (字典) - https://www.spoj.com/problems/DICT/
10. Project Euler Problem 2: Even Fibonacci numbers (偶数斐波那契数) - https://projecteuler.net/problem=2
11. HackerRank Maximum Palindromes (最大回文) - https://www.hackerrank.com/challenges/maximum-

palindromes
12. 计蒜客 T1101: 阶乘 (阶乘) - https://www.jisuanke.com/t/T1101
13. 杭电 OJ 1003: Max Sum (最大子序列和) - http://acm.hdu.edu.cn/showproblem.php?pid=1003
14. 牛客网 剑指 Offer 50: 第一个只出现一次的字符 (第一个只出现一次的字符) - https://www.nowcoder.com/practice/1c82e8cf713b4bbeb2a5b31cf5b0417c
15. acwing 800. 数组元素的目标和 (数组元素的目标和) - https://www.acwing.com/problem/content/802/
16. POJ 1003: Hangover (悬垂) - http://poj.org/problem?id=1003
17. UVa OJ 101: The Blocks Problem (积木问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=37
18. Timus OJ 1005: Stone Pile (石子堆) - https://acm.timus.ru/problem.aspx?space=1&num=1005
19. Aizu OJ ALDS1_5_A: Exhaustive Search (穷举搜索) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_5_A
20. Comet OJ Contest #1: 热身赛 B. 简单的数学题 (简单的数学题) - https://cometoj.com/contest/1/problem/B
21. MarsCode 火星编程竞赛: 数字统计 (数字统计) - https://www.marscode.cn/contest/1/problem/1002
22. ZOJ 1002: Fire Net (消防网) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1002
23. LOJ 101: 最小生成树 (最小生成树) - https://loj.ac/p/101
24. 各大高校 OJ: 北京大学 OJ 1000: A+B Problem (A+B 问题) - http://poj.org/problem?id=1000
25. LeetCode 352. Data Stream as Disjoint Intervals (数据流变为不相交区间) - https://leetcode.com/problems/data-stream-as-disjoint-intervals/
26. Codeforces 122A. Lucky Division (幸运分割) - https://codeforces.com/problemset/problem/122/A
27. AtCoder ABC 218 C - Shapes (形状) - https://atcoder.jp/contests/abc218/tasks/abc218_c
28. USACO Bronze: Block Game (积木游戏) - http://www.usaco.org/index.php?page=viewproblem2&cpid=664
29. 洛谷 P3366 【模板】最小生成树 (模板最小生成树) - https://www.luogu.com.cn/problem/P3366
30. LeetCode 149. Max Points on a Line (直线上最多的点数) - https://leetcode.com/problems/max-points-on-a-line/
31. LeetCode 215. Kth Largest Element in an Array (数组中的第 K 个最大元素) - https://leetcode.com/problems/kth-largest-element-in-an-array/
32. LeetCode 295. Find Median from Data Stream (数据流的中位数) - https://leetcode.com/problems/find-median-from-data-stream/
33. LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数) - https://leetcode.com/problems/count-of-smaller-numbers-after-self/
34. LeetCode 327. Count of Range Sum (区间和的个数) - https://leetcode.com/problems/count-of-range-sum/
35. LeetCode 350. Intersection of Two Arrays II (两个数组的交集 II) - https://leetcode.com/problems/intersection-of-two-arrays-ii/
36. LeetCode 148. Sort List (排序链表) - https://leetcode.com/problems/sort-list/
37. LeetCode 242. Valid Anagram (有效的字母异位词) - https://leetcode.com/problems/valid-anagram/
38. LeetCode 347. Top K Frequent Elements (前 K 个高频元素) - https://leetcode.com/problems/top-k-frequent-elements/
39. LeetCode 451. Sort Characters By Frequency (根据字符出现频率排序) -

https://leetcode.com/problems/sort-characters-by-frequency/
40. LeetCode 493. Reverse Pairs (翻转对) - https://leetcode.com/problems/reverse-pairs/
41. LeetCode 539. Minimum Time Difference (最小时间差) - https://leetcode.com/problems/minimum-time-difference/
42. LeetCode 791. Custom Sort String (自定义字符串排序) - https://leetcode.com/problems/custom-sort-string/
43. LeetCode 937. Reorder Data in Log Files (重新排列日志文件) - https://leetcode.com/problems/reorder-data-in-log-files/
44. LeetCode 1030. Matrix Cells in Distance Order (距离顺序排列矩阵单元格) - https://leetcode.com/problems/matrix-cells-in-distance-order/
45. LeetCode 1636. Sort Array by Increasing Frequency (按照频率将数组升序排序) - https://leetcode.com/problems/sort-array-by-increasing-frequency/
46. LeetCode 1710. Maximum Units on a Truck (卡车上的最大单元数) - https://leetcode.com/problems/maximum-units-on-a-truck/
47. HackerRank Java Comparator (Java 比较器) - https://www.hackerrank.com/challenges/java-comparator/problem
48. LeetCode 147. Insertion Sort List (对链表进行插入排序) - https://leetcode.com/problems/insertion-sort-list/
49. LeetCode 252. Meeting Rooms (会议室) - https://leetcode.com/problems/meeting-rooms/
50. LeetCode 253. Meeting Rooms II (会议室 II) - https://leetcode.com/problems/meeting-rooms-ii/
"""

```
from functools import cmp_to_key
import math

# LeetCode 973. K Closest Points to Origin (最接近原点的 K 个点)
def k_closest(points, k):
    """
    题目描述：
    给定一个数组 points ，其中 points[i] = [xi, yi] 表示 X-Y 平面上的一个点，并且是一个整数 k ，返回离原点 (0,0) 最近的 k 个点。
    这里，平面上两点之间的距离是 欧几里德距离（ √(x1 - x2)^2 + (y1 - y2)^2 ）。
    你可以按 任何顺序 返回答案。除了点坐标的顺序之外，答案 确保 是 唯一 的。

    示例：
    输入：points = [[1,1],[3,3],[2,2]], k = 2
    输出：[[1,1],[2,2]]

    输入：points = [[3,3],[5,-1],[-2,4]], k = 2
    输出：[[3,3],[-2,4]]
    （答案 [[-2,4],[3,3]] 也会被接受。）

    约束条件：
```

```
    1 <= k <= points.length <= 10^4
    -10^4 < xi, yi < 10^4

    解题思路：
    计算每个点到原点的距离的平方（避免开方运算），然后使用自定义 key 函数按距离排序，
    最后取前 k 个点。

    时间复杂度：O(n log n)，其中 n 是点的数量
    空间复杂度：O(1)，如果不考虑输出数组的空间
    """
    # 使用自定义 key 函数按距离排序
    points.sort(key=lambda point: point[0]*point[0] + point[1]*point[1])

    # 返回前 k 个点
    return points[:k]


# LeetCode 179. Largest Number (最大数)
def largest_number(nums):
    """
    题目描述：
    给定一组非负整数 nums，重新排列每个数的顺序（每个数不可拆分）使之组成一个最大的整数。
    注意：输出结果可能非常大，所以你需要返回一个字符串而不是整数。

    示例：
    输入：nums = [10,2]
    输出："210"

    输入：nums = [3,30,34,5,9]
    输出："9534330"

    约束条件：
    1 <= nums.length <= 100
    0 <= nums[i] <= 10^9

    解题思路：
    将整数数组转换为字符串数组，然后使用自定义比较函数排序。
    对于两个字符串 a 和 b，比较 a+b 和 b+a 的大小，决定它们的顺序。

    时间复杂度：O(n log n * m)，其中 n 是数组长度，m 是数字的平均长度
    空间复杂度：O(n * m)
    """
    # 转换为字符串数组
```

```
    strs = [str(num) for num in nums]

    # 自定义比较函数
    def compare(a, b):
        # 比较 b+a 和 a+b 的大小，注意是降序排列所以顺序颠倒
        if a + b > b + a:
            return -1
        elif a + b < b + a:
            return 1
        else:
            return 0

    # 使用自定义比较函数排序
    strs.sort(key=cmp_to_key(compare))

    # 处理全为 0 的特殊情况
    if strs[0] == "0":
        return "0"

    # 拼接结果
    return "".join(strs)


# LeetCode 1356. Sort Integers by The Number of 1 Bits (根据数字二进制下 1 的数目排序)
def sort_by_bits(arr):
    """
    题目描述：
    给你一个整数数组 arr 。请你将数组中的元素按照其二进制表示中数字 1 的数目升序排序。
    如果存在多个数字二进制中 1 的数目相同，则必须将它们按照数值大小升序排列。
    请你返回排序后的数组。

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

    输入：arr = [1024,512,256,128,64,32,16,8,4,2,1]
    输出：[1,2,4,8,16,32,64,128,256,512,1024]

    约束条件：
    1 <= arr.length <= 500
    0 <= arr[i] <= 10^4

    解题思路：
```

```
    使用自定义key函数排序，首先按二进制中1的位数排序，如果相同则按数值大小排序。
    计算二进制中1的位数可以使用bin().count('1')方法。

    时间复杂度：O(n log n)
    空间复杂度：O(1)
    """
    # 使用自定义key函数排序
    arr.sort(key=lambda x: (bin(x).count('1'), x))
    return arr


# HackerRank Java Comparator (Java比较器)的Python版本
class Player:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def __repr__(self):
        return f"{self.name} {self.score}"

def player_comparator(a, b):
    """
    题目描述：
    创建一个比较器，根据玩家的分数降序排列，如果分数相同则按名字升序排列。

    示例：
    输入：
    5
    amy 100
    david 100
    heraldo 50
    aakansha 75
    aleksa 150

    输出：
    aleksa 150
    amy 100
    david 100
    aakansha 75
    heraldo 50

    解题思路：
    实现一个比较函数，首先比较分数（降序），如果分数相同则比较名字（升序）。
```

```
    时间复杂度：O(n log n)，其中 n 是玩家数量
    空间复杂度：O(1)
    """
    # 首先按分数降序排列
    if a.score != b.score:
        return b.score - a.score
    # 如果分数相同，按名字升序排列
    if a.name < b.name:
        return -1
    elif a.name > b.name:
        return 1
    else:
        return 0


# LeetCode 56. Merge Intervals (合并区间)
def merge(intervals):
    """
    题目描述：
    以数组 intervals 表示若干个区间的集合，其中单个区间为 intervals[i] = [starti, endi] 。
    请你合并所有重叠的区间，并返回一个不重叠的区间数组，该数组需恰好覆盖输入中的所有区间。

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

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

    约束条件：
    1 <= intervals.length <= 10^4
    intervals[i].length == 2
    0 <= starti <= endi <= 10^4

    解题思路：
    1. 首先按区间的起始位置排序
    2. 遍历排序后的区间，合并重叠的区间

    时间复杂度：O(n log n)，其中 n 是区间数量
    空间复杂度：O(n)，用于存储结果
```

```
    """
    if len(intervals) <= 1:
        return intervals

    # 按区间的起始位置排序
    intervals.sort(key=lambda x: x[0])

    result = []
    current = intervals[0]
    result.append(current)

    for i in range(1, len(intervals)):
        interval = intervals[i]
        # 如果当前区间与下一个区间重叠，合并它们
        if interval[0] <= current[1]:
            current[1] = max(current[1], interval[1])
        else:
            # 否则，将下一个区间添加到结果中
            current = interval
            result.append(current)

    return result


# LeetCode 1122. Relative Sort Array (数组的相对排序)
def relative_sort_array(arr1, arr2):
    """
    题目描述：
    给你两个数组，arr1 和 arr2，arr2 中的元素各不相同，arr2 中的每个元素都出现在 arr1 中。
    对 arr1 中的元素进行排序，使 arr1 中项的相对顺序和 arr2 中的相对顺序相同。
    未在 arr2 中出现过的元素需要按照升序放在 arr1 的末尾。

    示例：
    输入：arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]
    输出：[2,2,2,1,4,3,3,9,6,7,19]

    约束条件：
    1 <= arr1.length, arr2.length <= 1000
    0 <= arr1[i], arr2[i] <= 1000
    arr2 中的元素 arr2[i] 各不相同
    arr2 中的每个元素 arr2[i] 都出现在 arr1 中

    解题思路：
```

```
    1. 使用自定义 key 函数排序
    2. 对于在 arr2 中出现的元素，按其在 arr2 中的索引排序
    3. 对于不在 arr2 中出现的元素，按数值大小升序排序，并放在末尾

    时间复杂度：O(m log m + n)，其中 m 是 arr1 的长度，n 是 arr2 的长度
    空间复杂度：O(n)，用于存储 arr2 中元素的索引映射
    """
    # 创建 arr2 中元素到索引的映射
    index_map = {num: i for i, num in enumerate(arr2)}

    # 使用自定义 key 函数排序
    arr1.sort(key=lambda x: (0, index_map[x]) if x in index_map else (1, x))

    return arr1


# 测试代码
if __name__ == "__main__":
    # 测试最接近原点的 K 个点
    print("测试最接近原点的 K 个点:")
    points = [[1,1],[3,3],[2,2]]
    result = k_closest(points, 2)
    print(result)

    # 测试最大数
    print("测试最大数:")
    nums = [3,30,34,5,9]
    print(largest_number(nums))

    # 测试根据数字二进制下 1 的数目排序
    print("测试根据数字二进制下 1 的数目排序:")
    arr = [0,1,2,3,4,5,6,7,8]
    result = sort_by_bits(arr)
    print(result)

    # 测试 HackerRank Java Comparator
    print("测试 HackerRank Java Comparator:")
    players = [
        Player("amy", 100),
        Player("david", 100),
        Player("heraldo", 50),
        Player("aakansha", 75),
        Player("aleksa", 150)
```

```
    ]
    players.sort(key=cmp_to_key(player_comparator))
    for player in players:
        print(player)

    # 测试合并区间
    print("测试合并区间:")
    intervals = [[1,3],[2,6],[8,10],[15,18]]
    merged = merge(intervals)
    print(merged)

    # 测试数组的相对排序
    print("测试数组的相对排序:")
    arr1 = [2,3,1,3,2,4,6,7,9,2,19]
    arr2 = [2,1,4,3,9,6]
    result = relative_sort_array(arr1, arr2)
    print(result)
```

```
==========================================
```

文件: Code04_TreeSetTreeMapAdvanced.cpp

```
==========================================
#include <iostream>
#include <vector>
#include <set>
#include <map>
#include <algorithm>
#include <climits>
#include <cmath>
using namespace std;

/**
 * TreeSet 和 TreeMap 高级应用题目实现（C++版本）
 * 包含 LeetCode 352, 363, 436, 456, 480, 683 等题目
 *
 * C++中 TreeSet 和 TreeMap 的实现：
 * - set 特点：基于红黑树实现，元素有序，支持范围查询，查找、插入、删除时间复杂度 O(logN)
 * - map 特点：基于红黑树实现，键值对有序，支持范围查询，查找、插入、删除时间复杂度 O(logN)
 * - 高级操作：lower_bound、upper_bound 等
 *
 * 时间复杂度分析：
 * - set/map 基本操作：O(log n)
 * - 范围查询：O(log n + k) 其中 k 是结果数量
```

 * - 排序操作：O(n log n)
 *
 * 空间复杂度分析：
 * - set/map 存储：O(n)
 * - 额外数据结构：O(n)
 *
 * 工程化考量：
 * 1. 异常处理：处理空输入、边界条件
 * 2. 性能优化：选择合适的数据结构，避免不必要的对象创建
 * 3. 代码可读性：添加详细注释，使用有意义的变量名
 * 4. 内存管理：注意迭代器有效性，避免内存泄漏
 *
 * 相关平台题目：
 * 1. LeetCode 352. Data Stream as Disjoint Intervals (数据流变为不相交区间) - https://leetcode.com/problems/data-stream-as-disjoint-intervals/
 * 2. LeetCode 363. Max Sum of Rectangle No Larger Than K (矩形区域不超过 K 的最大数值和) - https://leetcode.com/problems/max-sum-of-rectangle-no-larger-than-k/
 * 3. LeetCode 436. Find Right Interval (寻找右区间) - https://leetcode.com/problems/find-right-interval/
 * 4. LeetCode 456. 132 Pattern (132 模式) - https://leetcode.com/problems/132-pattern/
 * 5. LeetCode 480. Sliding Window Median (滑动窗口中位数) - https://leetcode.com/problems/sliding-window-median/
 * 6. LeetCode 683. K Empty Slots (K 个空花盆) - https://leetcode.com/problems/k-empty-slots/
 * 7. LeetCode 715. Range Module (范围模块) - https://leetcode.com/problems/range-module/
 * 8. LeetCode 731. My Calendar II (我的日程安排表 II) - https://leetcode.com/problems/my-calendar-ii/
 * 9. LeetCode 732. My Calendar III (我的日程安排表 III) - https://leetcode.com/problems/my-calendar-iii/
 * 10. LeetCode 855. Exam Room (考场就座) - https://leetcode.com/problems/exam-room/
 * 11. LeetCode 981. Time Based Key-Value Store (基于时间的键值存储) - https://leetcode.com/problems/time-based-key-value-store/
 * 12. LeetCode 1146. Snapshot Array (快照数组) - https://leetcode.com/problems/snapshot-array/
 * 13. LeetCode 1348. Tweet Counts Per Frequency (推文计数) - https://leetcode.com/problems/tweet-counts-per-frequency/
 * 14. LeetCode 1438. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit (绝对差不超过限制的最长连续子数组) - https://leetcode.com/problems/longest-continuous-subarray-with-absolute-diff-less-than-or-equal-to-limit/
 * 15. LintCode 613. High Five (最高分五科) - https://www.lintcode.com/problem/high-five/
 * 16. HackerEarth Monk and the Magical Candy Bags (和尚与魔法糖果袋) - https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/practice-problems/algorithm/monk-and-the-magical-candy-bags/
 * 17. CodeChef FRGTNLNG (遗忘的语言) - https://www.codechef.com/problems/FRGTNLNG
 * 18. SPOJ DICT (字典) - https://www.spoj.com/problems/DICT/

* 19. Project Euler Problem 2: Even Fibonacci numbers (偶数斐波那契数) - https://projecteuler.net/problem=2
* 20. HackerRank Maximum Palindromes (最大回文) - https://www.hackerrank.com/challenges/maximum-palindromes
* 21. 计蒜客 T1101: 阶乘 (阶乘) - https://www.jisuanke.com/t/T1101
* 22. 杭电 OJ 1003: Max Sum (最大子序列和) - http://acm.hdu.edu.cn/showproblem.php?pid=1003
* 23. 牛客网 剑指 Offer 50: 第一个只出现一次的字符 (第一个只出现一次的字符) - https://www.nowcoder.com/practice/1c82e8cf713b4bbeb2a5b31cf5b0417c
* 24. acwing 800. 数组元素的目标和 (数组元素的目标和) - https://www.acwing.com/problem/content/802/
* 25. POJ 1003: Hangover (悬垂) - http://poj.org/problem?id=1003
* 26. UVa OJ 101: The Blocks Problem (积木问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=37
* 27. Timus OJ 1005: Stone Pile (石子堆) - https://acm.timus.ru/problem.aspx?space=1&num=1005
* 28. Aizu OJ ALDS1_5_A: Exhaustive Search (穷举搜索) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_5_A
* 29. Comet OJ Contest #1: 热身赛 B. 简单的数学题 (简单的数学题) - https://cometoj.com/contest/1/problem/B
* 30. MarsCode 火星编程竞赛: 数字统计 (数字统计) - https://www.marscode.cn/contest/1/problem/1002
* 31. ZOJ 1002: Fire Net (消防网) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1002
* 32. LOJ 101: 最小生成树 (最小生成树) - https://loj.ac/p/101
* 33. 各大高校 OJ: 北京大学 OJ 1000: A+B Problem (A+B 问题) - http://poj.org/problem?id=1000
* 34. Codeforces 122A. Lucky Division (幸运分割) - https://codeforces.com/problemset/problem/122/A
* 35. AtCoder ABC 218 C - Shapes (形状) - https://atcoder.jp/contests/abc218/tasks/abc218_c
* 36. USACO Bronze: Block Game (积木游戏) - http://www.usaco.org/index.php?page=viewproblem2&cpid=664
* 37. 洛谷 P3366 【模板】最小生成树 (模板最小生成树) - https://www.luogu.com.cn/problem/P3366
* 38. LeetCode 149. Max Points on a Line (直线上最多的点数) - https://leetcode.com/problems/max-points-on-a-line/
* 39. LeetCode 215. Kth Largest Element in an Array (数组中的第 K 个最大元素) - https://leetcode.com/problems/kth-largest-element-in-an-array/
* 40. LeetCode 295. Find Median from Data Stream (数据流的中位数) - https://leetcode.com/problems/find-median-from-data-stream/
* 41. LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数) - https://leetcode.com/problems/count-of-smaller-numbers-after-self/
* 42. LeetCode 327. Count of Range Sum (区间和的个数) - https://leetcode.com/problems/count-of-range-sum/
* 43. LeetCode 350. Intersection of Two Arrays II (两个数组的交集 II) - https://leetcode.com/problems/intersection-of-two-arrays-ii/
* 44. LeetCode 148. Sort List (排序链表) - https://leetcode.com/problems/sort-list/

```
 * 45. LeetCode 242. Valid Anagram (有效的字母异位词) - https://leetcode.com/problems/valid-anagram/
 * 46. LeetCode 347. Top K Frequent Elements (前 K 个高频元素) - https://leetcode.com/problems/top-k-frequent-elements/
 * 47. LeetCode 451. Sort Characters By Frequency (根据字符出现频率排序) - https://leetcode.com/problems/sort-characters-by-frequency/
 * 48. LeetCode 493. Reverse Pairs (翻转对) - https://leetcode.com/problems/reverse-pairs/
 * 49. LeetCode 539. Minimum Time Difference (最小时间差) - https://leetcode.com/problems/minimum-time-difference/
 * 50. LeetCode 791. Custom Sort String (自定义字符串排序) - https://leetcode.com/problems/custom-sort-string/
 */

/**
 * LeetCode 352. Data Stream as Disjoint Intervals
 * 将数据流变为多个不相交区间
 * 网址：https://leetcode.com/problems/data-stream-as-disjoint-intervals/
 *
 * 解题思路：
 * 1. 使用 map 存储区间边界，键为区间起点，值为区间终点
 * 2. 添加数字时，查找可能合并的相邻区间
 * 3. 合并重叠或相邻的区间
 *
 * 时间复杂度：每次添加 O(log n)，获取区间 O(n)
 * 空间复杂度：O(n)
 */
class DataStreamAsDisjointIntervals {
private:
    map<int, int> intervals;

public:
    DataStreamAsDisjointIntervals() {}

    void addNum(int val) {
        // 查找小于等于 val 的区间
        auto it = intervals.upper_bound(val);
        if (it != intervals.begin()) {
            it--;
            if (it->first <= val && val <= it->second) {
                // 数字已经在某个区间内，不需要处理
                return;
            }
        }
```

```
    // 检查是否可以与左侧区间合并
    bool mergeLeft = false, mergeRight = false;
    int leftKey = -1, rightKey = -1;

    if (it != intervals.begin()) {
        auto leftIt = it;
        leftIt--;
        if (leftIt->second + 1 == val) {
            mergeLeft = true;
            leftKey = leftIt->first;
        }
    }

    // 检查是否可以与右侧区间合并
    if (it != intervals.end() && it->first == val + 1) {
        mergeRight = true;
        rightKey = it->first;
    }

    if (mergeLeft && mergeRight) {
        // 合并左右区间
        int newEnd = intervals[rightKey];
        intervals[leftKey] = newEnd;
        intervals.erase(rightKey);
    } else if (mergeLeft) {
        // 只与左侧区间合并
        intervals[leftKey] = val;
    } else if (mergeRight) {
        // 只与右侧区间合并
        int end = intervals[rightKey];
        intervals.erase(rightKey);
        intervals[val] = end;
    } else {
        // 创建新区间
        intervals[val] = val;
    }
}

vector<vector<int>> getIntervals() {
    vector<vector<int>> result;
    for (auto& entry : intervals) {
        result.push_back({entry.first, entry.second});
```

```
        }
        return result;
    }
};

/**
 * LeetCode 363. Max Sum of Rectangle No Larger Than K
 * 矩形区域不超过 K 的最大数值和
 * 网址：https://leetcode.com/problems/max-sum-of-rectangle-no-larger-than-k/
 *
 * 解题思路：
 * 1. 固定左右边界，计算每一行的前缀和
 * 2. 使用 set 维护前缀和，快速查找满足条件的前缀和
 * 3. 使用 lower_bound 找到大于等于(target - k)的最小值
 *
 * 时间复杂度：O(min(m,n)² * max(m,n) log max(m,n))
 * 空间复杂度：O(max(m,n))
 */
class Solution {
public:
    int maxSumSubmatrix(vector<vector<int>>& matrix, int k) {
        int m = matrix.size(), n = matrix[0].size();
        int maxSum = INT_MIN;

        // 让 m 是较小的维度，减少时间复杂度
        if (m > n) {
            vector<vector<int>> rotated(n, vector<int>(m));
            for (int i = 0; i < m; i++) {
                for (int j = 0; j < n; j++) {
                    rotated[j][i] = matrix[i][j];
                }
            }
            return maxSumSubmatrix(rotated, k);
        }

        // 枚举上下边界
        for (int top = 0; top < m; top++) {
            vector<int> colSum(n, 0);
            for (int bottom = top; bottom < m; bottom++) {
                // 更新列前缀和
                for (int j = 0; j < n; j++) {
                    colSum[j] += matrix[bottom][j];
                }
```

```
                // 使用 set 维护前缀和
                set<int> prefixSums;
                prefixSums.insert(0);
                int currentSum = 0;

                for (int j = 0; j < n; j++) {
                    currentSum += colSum[j];
                    // 查找大于等于(currentSum - k)的最小值
                    auto it = prefixSums.lower_bound(currentSum - k);
                    if (it != prefixSums.end()) {
                        maxSum = max(maxSum, currentSum - *it);
                    }
                    prefixSums.insert(currentSum);
                }
            }
        }

        return maxSum;
    }
};

/**
 * LeetCode 436. Find Right Interval
 * 寻找右区间
 * 网址：https://leetcode.com/problems/find-right-interval/
 *
 * 解题思路：
 * 1. 使用 map 存储每个区间的起始位置和索引
 * 2. 对于每个区间，使用 lower_bound 找到起始位置大于等于当前区间结束位置的最小区间
 * 3. 如果找到则返回对应索引，否则返回-1
 *
 * 时间复杂度：O(n log n)
 * 空间复杂度：O(n)
 */
class FindRightInterval {
public:
    vector<int> findRightInterval(vector<vector<int>>& intervals) {
        int n = intervals.size();
        vector<int> result(n, -1);

        // 使用 map 存储区间起始位置和索引
        map<int, int> startMap;
```

```
        for (int i = 0; i < n; i++) {
            startMap[intervals[i][0]] = i;
        }

        for (int i = 0; i < n; i++) {
            int end = intervals[i][1];
            // 查找大于等于 end 的最小起始位置
            auto it = startMap.lower_bound(end);
            if (it != startMap.end()) {
                result[i] = it->second;
            }
        }

        return result;
    }
};

// 测试函数
int main() {
    // 测试 LeetCode 436
    vector<vector<int>> intervals = {{1,2}, {2,3}, {0,1}, {3,4}};
    FindRightInterval solution;
    vector<int> result = solution.findRightInterval(intervals);
    cout << "LeetCode 436 Result: ";
    for (int num : result) cout << num << " ";
    cout << endl;

    return 0;
}
```

==========

文件: Code04_TreeSetTreeMapAdvanced.java

==========

```
import java.util.*;

/**
 * TreeSet 和 TreeMap 高级应用题目实现
 * 包含 LeetCode 352, 363, 436, 456, 480, 683, 715, 731, 732, 855, 981, 1146, 1348, 1438 等题目
 *
 * TreeSet 和 TreeMap 高级特性：
 * - TreeSet 特点：基于红黑树实现，元素有序，支持范围查询，查找、插入、删除时间复杂度 O(logN)
 * - TreeMap 特点：基于红黑树实现，键值对有序，支持范围查询，查找、插入、删除时间复杂度 O(logN)
```

```
 * - 高级操作：ceilingKey、floorKey、higherKey、lowerKey 等
 *
 * 时间复杂度分析：
 * - TreeSet/TreeMap 基本操作：O(log n)
 * - 范围查询：O(log n + k) 其中 k 是结果数量
 * - 排序操作：O(n log n)
 *
 * 空间复杂度分析：
 * - TreeSet/TreeMap 存储：O(n)
 * - 额外数据结构：O(n)
 *
 * 工程化考量：
 * 1. 异常处理：处理空输入、边界条件
 * 2. 性能优化：选择合适的数据结构，避免不必要的对象创建
 * 3. 代码可读性：添加详细注释，使用有意义的变量名
 * 4. 线程安全：非线程安全，多线程环境下需要同步
 *
 * 相关平台题目：
 * 1. LeetCode 352. Data Stream as Disjoint Intervals (数据流变为不相交区间) - https://leetcode.com/problems/data-stream-as-disjoint-intervals/
 * 2. LeetCode 363. Max Sum of Rectangle No Larger Than K (矩形区域不超过 K 的最大数值和) - https://leetcode.com/problems/max-sum-of-rectangle-no-larger-than-k/
 * 3. LeetCode 436. Find Right Interval (寻找右区间) - https://leetcode.com/problems/find-right-interval/
 * 4. LeetCode 456. 132 Pattern (132 模式) - https://leetcode.com/problems/132-pattern/
 * 5. LeetCode 480. Sliding Window Median (滑动窗口中位数) - https://leetcode.com/problems/sliding-window-median/
 * 6. LeetCode 683. K Empty Slots (K 个空花盆) - https://leetcode.com/problems/k-empty-slots/
 * 7. LeetCode 715. Range Module (范围模块) - https://leetcode.com/problems/range-module/
 * 8. LeetCode 731. My Calendar II (我的日程安排表 II) - https://leetcode.com/problems/my-calendar-ii/
 * 9. LeetCode 732. My Calendar III (我的日程安排表 III) - https://leetcode.com/problems/my-calendar-iii/
 * 10. LeetCode 855. Exam Room (考场就座) - https://leetcode.com/problems/exam-room/
 * 11. LeetCode 981. Time Based Key-Value Store (基于时间的键值存储) - https://leetcode.com/problems/time-based-key-value-store/
 * 12. LeetCode 1146. Snapshot Array (快照数组) - https://leetcode.com/problems/snapshot-array/
 * 13. LeetCode 1348. Tweet Counts Per Frequency (推文计数) - https://leetcode.com/problems/tweet-counts-per-frequency/
 * 14. LeetCode 1438. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit (绝对差不超过限制的最长连续子数组) - https://leetcode.com/problems/longest-continuous-subarray-with-absolute-diff-less-than-or-equal-to-limit/
 * 15. LintCode 613. High Five (最高分五科) - https://www.lintcode.com/problem/high-five/
```

* 16. HackerEarth Monk and the Magical Candy Bags (和尚与魔法糖果袋) - https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/practice-problems/algorithm/monk-and-the-magical-candy-bags/
* 17. CodeChef FRGTNLNG (遗忘的语言) - https://www.codechef.com/problems/FRGTNLNG
* 18. SPOJ DICT (字典) - https://www.spoj.com/problems/DICT/
* 19. Project Euler Problem 2: Even Fibonacci numbers (偶数斐波那契数) - https://projecteuler.net/problem=2
* 20. HackerRank Maximum Palindromes (最大回文) - https://www.hackerrank.com/challenges/maximum-palindromes
* 21. 计蒜客 T1101: 阶乘 (阶乘) - https://www.jisuanke.com/t/T1101
* 22. 杭电 OJ 1003: Max Sum (最大子序列和) - http://acm.hdu.edu.cn/showproblem.php?pid=1003
* 23. 牛客网 剑指 Offer 50: 第一个只出现一次的字符 (第一个只出现一次的字符) - https://www.nowcoder.com/practice/1c82e8cf713b4bbeb2a5b31cf5b0417c
* 24. acwing 800. 数组元素的目标和 (数组元素的目标和) - https://www.acwing.com/problem/content/802/
* 25. POJ 1003: Hangover (悬垂) - http://poj.org/problem?id=1003
* 26. UVa OJ 101: The Blocks Problem (积木问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=37
* 27. Timus OJ 1005: Stone Pile (石子堆) - https://acm.timus.ru/problem.aspx?space=1&num=1005
* 28. Aizu OJ ALDS1_5_A: Exhaustive Search (穷举搜索) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_5_A
* 29. Comet OJ Contest #1: 热身赛 B. 简单的数学题 (简单的数学题) - https://cometoj.com/contest/1/problem/B
* 30. MarsCode 火星编程竞赛: 数字统计 (数字统计) - https://www.marscode.cn/contest/1/problem/1002
* 31. ZOJ 1002: Fire Net (消防网) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1002
* 32. LOJ 101: 最小生成树 (最小生成树) - https://loj.ac/p/101
* 33. 各大高校 OJ: 北京大学 OJ 1000: A+B Problem (A+B 问题) - http://poj.org/problem?id=1000
* 34. Codeforces 122A. Lucky Division (幸运分割) - https://codeforces.com/problemset/problem/122/A
* 35. AtCoder ABC 218 C - Shapes (形状) - https://atcoder.jp/contests/abc218/tasks/abc218_c
* 36. USACO Bronze: Block Game (积木游戏) - http://www.usaco.org/index.php?page=viewproblem2&cpid=664
* 37. 洛谷 P3366 【模板】最小生成树 (模板最小生成树) - https://www.luogu.com.cn/problem/P3366
* 38. LeetCode 149. Max Points on a Line (直线上最多的点数) - https://leetcode.com/problems/max-points-on-a-line/
* 39. LeetCode 215. Kth Largest Element in an Array (数组中的第 K 个最大元素) - https://leetcode.com/problems/kth-largest-element-in-an-array/
* 40. LeetCode 295. Find Median from Data Stream (数据流的中位数) - https://leetcode.com/problems/find-median-from-data-stream/
* 41. LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数) - https://leetcode.com/problems/count-of-smaller-numbers-after-self/

```
 * 42. LeetCode 327. Count of Range Sum (区间和的个数) - https://leetcode.com/problems/count-of-range-sum/
 * 43. LeetCode 350. Intersection of Two Arrays II (两个数组的交集 II) - https://leetcode.com/problems/intersection-of-two-arrays-ii/
 * 44. LeetCode 148. Sort List (排序链表) - https://leetcode.com/problems/sort-list/
 * 45. LeetCode 242. Valid Anagram (有效的字母异位词) - https://leetcode.com/problems/valid-anagram/
 * 46. LeetCode 347. Top K Frequent Elements (前 K 个高频元素) - https://leetcode.com/problems/top-k-frequent-elements/
 * 47. LeetCode 451. Sort Characters By Frequency (根据字符出现频率排序) - https://leetcode.com/problems/sort-characters-by-frequency/
 * 48. LeetCode 493. Reverse Pairs (翻转对) - https://leetcode.com/problems/reverse-pairs/
 * 49. LeetCode 539. Minimum Time Difference (最小时间差) - https://leetcode.com/problems/minimum-time-difference/
 * 50. LeetCode 791. Custom Sort String (自定义字符串排序) - https://leetcode.com/problems/custom-sort-string/

public class Code04_TreeSetTreeMapAdvanced {

    /**
     * LeetCode 352. Data Stream as Disjoint Intervals
     * 将数据流变为多个不相交区间
     * 网址：https://leetcode.com/problems/data-stream-as-disjoint-intervals/
     *
     * 解题思路：
     * 1. 使用 TreeMap 存储区间边界，键为区间起点，值为区间终点
     * 2. 添加数字时，查找可能合并的相邻区间
     * 3. 合并重叠或相邻的区间
     *
     * 时间复杂度：每次添加 O(log n)，获取区间 O(n)
     * 空间复杂度：O(n)
     */
    static class DataStreamAsDisjointIntervals {
        private TreeMap<Integer, Integer> intervals;

        public DataStreamAsDisjointIntervals() {
            intervals = new TreeMap<>();
        }

        public void addNum(int val) {
            // 查找小于等于 val 的区间
            Integer floorKey = intervals.floorKey(val);
            if (floorKey != null && intervals.get(floorKey) >= val) {
```

```
            // 数字已经在某个区间内，不需要处理
            return;
        }

        // 检查是否可以与左侧区间合并
        boolean mergeLeft = floorKey != null && intervals.get(floorKey) + 1 == val;

        // 检查是否可以与右侧区间合并
        Integer higherKey = intervals.higherKey(val);
        boolean mergeRight = higherKey != null && higherKey == val + 1;

        if (mergeLeft && mergeRight) {
            // 合并左右区间
            int newEnd = intervals.get(higherKey);
            intervals.put(floorKey, newEnd);
            intervals.remove(higherKey);
        } else if (mergeLeft) {
            // 只与左侧区间合并
            intervals.put(floorKey, val);
        } else if (mergeRight) {
            // 只与右侧区间合并
            int end = intervals.get(higherKey);
            intervals.remove(higherKey);
            intervals.put(val, end);
        } else {
            // 创建新区间
            intervals.put(val, val);
        }
    }

    public int[][] getIntervals() {
        int[][] result = new int[intervals.size()][2];
        int index = 0;
        for (Map.Entry<Integer, Integer> entry : intervals.entrySet()) {
            result[index][0] = entry.getKey();
            result[index][1] = entry.getValue();
            index++;
        }
        return result;
    }
}

/**
```

```
     * LeetCode 363. Max Sum of Rectangle No Larger Than K
     * 矩形区域不超过 K 的最大数值和
     * 网址：https://leetcode.com/problems/max-sum-of-rectangle-no-larger-than-k/
     *
     * 解题思路：
     * 1. 固定左右边界，计算每一行的前缀和
     * 2. 使用 TreeSet 维护前缀和，快速查找满足条件的前缀和
     * 3. 使用 ceil 方法找到大于等于(target - k)的最小值
     *
     * 时间复杂度：O(min(m,n)² * max(m,n) log max(m,n))
     * 空间复杂度：O(max(m,n))
     */
    public int maxSumSubmatrix(int[][] matrix, int k) {
        int m = matrix.length;
        int n = matrix[0].length;
        int maxSum = Integer.MIN_VALUE;

        // 让 m 是较小的维度，减少时间复杂度
        if (m > n) {
            int[][] rotated = new int[n][m];
            for (int i = 0; i < m; i++) {
                for (int j = 0; j < n; j++) {
                    rotated[j][i] = matrix[i][j];
                }
            }
            return maxSumSubmatrix(rotated, k);
        }

        // 枚举上下边界
        for (int top = 0; top < m; top++) {
            int[] colSum = new int[n];
            for (int bottom = top; bottom < m; bottom++) {
                // 更新列前缀和
                for (int j = 0; j < n; j++) {
                    colSum[j] += matrix[bottom][j];
                }

                // 使用 TreeSet 维护前缀和
                TreeSet<Integer> set = new TreeSet<>();
                set.add(0);
                int prefixSum = 0;

                for (int j = 0; j < n; j++) {
```

```
                prefixSum += colSum[j];
                // 查找大于等于(prefixSum - k)的最小值
                Integer ceil = set.ceiling(prefixSum - k);
                if (ceil != null) {
                    maxSum = Math.max(maxSum, prefixSum - ceil);
                }
                set.add(prefixSum);
            }
        }
    }

    return maxSum;
}

/**
 * LeetCode 436. Find Right Interval
 * 寻找右区间
 * 网址：https://leetcode.com/problems/find-right-interval/
 *
 * 解题思路：
 * 1. 使用TreeMap存储每个区间的起始位置和索引
 * 2. 对于每个区间，使用ceiling方法找到起始位置大于等于当前区间结束位置的最小区间
 * 3. 如果找到则返回对应索引，否则返回-1
 *
 * 时间复杂度：O(n log n)
 * 空间复杂度：O(n)
 */
public int[] findRightInterval(int[][] intervals) {
    int n = intervals.length;
    int[] result = new int[n];

    // 使用TreeMap存储区间起始位置和索引
    TreeMap<Integer, Integer> map = new TreeMap<>();
    for (int i = 0; i < n; i++) {
        map.put(intervals[i][0], i);
    }

    for (int i = 0; i < n; i++) {
        int end = intervals[i][1];
        // 查找大于等于end的最小起始位置
        Integer rightStart = map.ceilingKey(end);
        if (rightStart != null) {
            result[i] = map.get(rightStart);
```

```
        } else {
            result[i] = -1;
        }
    }

    return result;
}

/**
 * LeetCode 456. 132 Pattern
 * 132 模式
 * 网址：https://leetcode.com/problems/132-pattern/
 *
 * 解题思路：
 * 1. 从右向左遍历数组
 * 2. 使用 TreeSet 维护右侧元素
 * 3. 维护一个变量记录当前最大值作为 3
 * 4. 检查是否存在 1 小于 3 且 3 小于 2 的模式
 *
 * 时间复杂度：O(n log n)
 * 空间复杂度：O(n)
 */
public boolean find132pattern(int[] nums) {
    int n = nums.length;
    if (n < 3) return false;

    // 记录左侧最小值
    int[] minLeft = new int[n];
    minLeft[0] = nums[0];
    for (int i = 1; i < n; i++) {
        minLeft[i] = Math.min(minLeft[i-1], nums[i]);
    }

    // 使用 TreeSet 维护右侧元素
    TreeSet<Integer> rightSet = new TreeSet<>();
    rightSet.add(nums[n-1]);

    for (int i = n-2; i > 0; i--) {
        // 当前元素作为 3，左侧最小值作为 1
        if (minLeft[i-1] < nums[i]) {
            // 在右侧查找大于左侧最小值且小于当前元素的数作为 2
            Integer candidate = rightSet.lower(nums[i]);
            if (candidate != null && candidate > minLeft[i-1]) {
```

```
                return true;
            }
        }
        rightSet.add(nums[i]);
    }

    return false;
}

/**
 * LeetCode 480. Sliding Window Median
 * 滑动窗口中位数
 * 网址：https://leetcode.com/problems/sliding-window-median/
 *
 * 解题思路：
 * 1. 使用两个 TreeSet 维护窗口元素
 * 2. 左半部分存储较小的一半元素，右半部分存储较大的一半元素
 * 3. 保持两个集合大小平衡，中位数即为左半部分的最大值或两个最大值的平均值
 *
 * 时间复杂度：O(n log k)
 * 空间复杂度：O(k)
 */
public double[] medianSlidingWindow(int[] nums, int k) {
    int n = nums.length;
    double[] result = new double[n - k + 1];

    // 使用两个 TreeSet 维护窗口元素
    TreeSet<Integer> left = new TreeSet<>((a, b) ->
        nums[a] != nums[b] ? Integer.compare(nums[a], nums[b]) : Integer.compare(a, b));
    TreeSet<Integer> right = new TreeSet<>((a, b) ->
        nums[a] != nums[b] ? Integer.compare(nums[a], nums[b]) : Integer.compare(a, b));

    // 平衡函数，保持两个集合大小平衡
    Runnable balance = () -> {
        while (left.size() > right.size()) {
            right.add(left.pollLast());
        }
    };

    for (int i = 0; i < n; i++) {
        // 添加新元素
        if (left.isEmpty() || nums[i] <= nums[left.last()]) {
            left.add(i);
```

```
        } else {
            right.add(i);
        }
        balance.run();

        // 移除窗口外的元素
        if (i >= k) {
            if (!left.remove(i - k)) {
                right.remove(i - k);
            }
            balance.run();
        }

        // 计算中位数
        if (i >= k - 1) {
            if (k % 2 == 0) {
                result[i - k + 1] = ((double) nums[left.last()] + nums[right.first()]) / 2.0;
            } else {
                result[i - k + 1] = (double) nums[right.first()];
            }
        }
    }

    return result;
}

/**
 * LeetCode 683. K Empty Slots
 * K 个空花盆
 * 网址：https://leetcode.com/problems/k-empty-slots/
 *
 * 解题思路：
 * 1. 使用 TreeSet 存储开花位置
 * 2. 按天数顺序添加开花位置
 * 3. 每次添加后检查新位置与相邻位置的距离是否为 k+1
 *
 * 时间复杂度：O(n log n)
 * 空间复杂度：O(n)
 */
public int kEmptySlots(int[] flowers, int k) {
    TreeSet<Integer> bloomed = new TreeSet<>();

    for (int day = 0; day < flowers.length; day++) {
```

```
            int position = flowers[day];
            bloomed.add(position);

            // 检查左侧相邻位置
            Integer lower = bloomed.lower(position);
            if (lower != null && position - lower - 1 == k) {
                return day + 1;
            }

            // 检查右侧相邻位置
            Integer higher = bloomed.higher(position);
            if (higher != null && higher - position - 1 == k) {
                return day + 1;
            }
        }

        return -1;
    }

    /**
     * 测试方法
     */
    public static void main(String[] args) {
        Code04_TreeSetTreeMapAdvanced solution = new Code04_TreeSetTreeMapAdvanced();

        // 测试 LeetCode 436
        int[][] intervals = {{1,2}, {2,3}, {0,1}, {3,4}};
        int[] result = solution.findRightInterval(intervals);
        System.out.println("LeetCode 436 Result: " + Arrays.toString(result));

        // 测试 LeetCode 456
        int[] nums = {3,1,4,2};
        boolean has132 = solution.find132pattern(nums);
        System.out.println("LeetCode 456 Result: " + has132);

        // 测试 LeetCode 683
        int[] flowers = {1,3,2};
        int kEmpty = solution.kEmptySlots(flowers, 1);
        System.out.println("LeetCode 683 Result: " + kEmpty);
    }
}
```

==========

文件: Code04_TreeSetTreeMapAdvanced.py
==========

```
import bisect
# from sortedcontainers import SortedList, SortedDict  # 注释掉依赖模块
from typing import List, Tuple
import sys

"""
TreeSet 和 TreeMap 高级应用题目实现（Python 版本）
包含 LeetCode 352, 363, 436, 456, 480, 683, 715, 731, 732, 855, 981, 1146, 1348, 1438 等题目

Python 中 TreeSet 和 TreeMap 的实现：
- Python 标准库没有直接的 TreeSet 和 TreeMap 实现
- 使用 bisect 模块实现有序列表功能
- 使用字典和排序列表组合实现有序映射
- SortedList 和 SortedDict 是第三方库 sortedcontainers 提供的实现

时间复杂度分析：
- bisect 操作：O(log n)
- 范围查询：O(log n + k) 其中 k 是结果数量
- 排序操作：O(n log n)

空间复杂度分析：
- 排序列表存储：O(n)
- 额外数据结构：O(n)

工程化考量：
1. 异常处理：处理空输入、边界条件
2. 性能优化：选择合适的数据结构，避免不必要的对象创建
3. 代码可读性：添加详细注释，使用有意义的变量名
4. 线程安全：非线程安全，多线程环境下需要同步
5. 兼容性：使用标准库或流行第三方库

相关平台题目：
1. LeetCode 352. Data Stream as Disjoint Intervals (数据流变为不相交区间) - https://leetcode.com/problems/data-stream-as-disjoint-intervals/
2. LeetCode 363. Max Sum of Rectangle No Larger Than K (矩形区域不超过 K 的最大数值和) - https://leetcode.com/problems/max-sum-of-rectangle-no-larger-than-k/
3. LeetCode 436. Find Right Interval (寻找右区间) - https://leetcode.com/problems/find-right-interval/
4. LeetCode 456. 132 Pattern (132 模式) - https://leetcode.com/problems/132-pattern/
5. LeetCode 480. Sliding Window Median (滑动窗口中位数) - https://leetcode.com/problems/sliding-
```

window-median/
6. LeetCode 683. K Empty Slots (K 个空花盆) - https://leetcode.com/problems/k-empty-slots/
7. LeetCode 715. Range Module (范围模块) - https://leetcode.com/problems/range-module/
8. LeetCode 731. My Calendar II (我的日程安排表 II) - https://leetcode.com/problems/my-calendar-ii/
9. LeetCode 732. My Calendar III (我的日程安排表 III) - https://leetcode.com/problems/my-calendar-iii/
10. LeetCode 855. Exam Room (考场就座) - https://leetcode.com/problems/exam-room/
11. LeetCode 981. Time Based Key-Value Store (基于时间的键值存储) - https://leetcode.com/problems/time-based-key-value-store/
12. LeetCode 1146. Snapshot Array (快照数组) - https://leetcode.com/problems/snapshot-array/
13. LeetCode 1348. Tweet Counts Per Frequency (推文计数) - https://leetcode.com/problems/tweet-counts-per-frequency/
14. LeetCode 1438. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit (绝对差不超过限制的最长连续子数组) - https://leetcode.com/problems/longest-continuous-subarray-with-absolute-diff-less-than-or-equal-to-limit/
15. LintCode 613. High Five (最高分五科) - https://www.lintcode.com/problem/high-five/
16. HackerEarth Monk and the Magical Candy Bags (和尚与魔法糖果袋) - https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/practice-problems/algorithm/monk-and-the-magical-candy-bags/
17. CodeChef FRGTNLNG (遗忘的语言) - https://www.codechef.com/problems/FRGTNLNG
18. SPOJ DICT (字典) - https://www.spoj.com/problems/DICT/
19. Project Euler Problem 2: Even Fibonacci numbers (偶数斐波那契数) - https://projecteuler.net/problem=2
20. HackerRank Maximum Palindromes (最大回文) - https://www.hackerrank.com/challenges/maximum-palindromes
21. 计蒜客 T1101: 阶乘 (阶乘) - https://www.jisuanke.com/t/T1101
22. 杭电 OJ 1003: Max Sum (最大子序列和) - http://acm.hdu.edu.cn/showproblem.php?pid=1003
23. 牛客网 剑指 Offer 50: 第一个只出现一次的字符 (第一个只出现一次的字符) - https://www.nowcoder.com/practice/1c82e8cf713b4bbeb2a5b31cf5b0417c
24. acwing 800. 数组元素的目标和 (数组元素的目标和) - https://www.acwing.com/problem/content/802/
25. POJ 1003: Hangover (悬垂) - http://poj.org/problem?id=1003
26. UVa OJ 101: The Blocks Problem (积木问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=37
27. Timus OJ 1005: Stone Pile (石子堆) - https://acm.timus.ru/problem.aspx?space=1&num=1005
28. Aizu OJ ALDS1_5_A: Exhaustive Search (穷举搜索) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_5_A
29. Comet OJ Contest #1: 热身赛 B. 简单的数学题 (简单的数学题) - https://cometoj.com/contest/1/problem/B
30. MarsCode 火星编程竞赛: 数字统计 (数字统计) - https://www.marscode.cn/contest/1/problem/1002
31. ZOJ 1002: Fire Net (消防网) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1002
32. LOJ 101: 最小生成树 (最小生成树) - https://loj.ac/p/101

33. 各大高校 OJ: 北京大学 OJ 1000: A+B Problem (A+B 问题) - http://poj.org/problem?id=1000
34. Codeforces 122A. Lucky Division (幸运分割) - https://codeforces.com/problemset/problem/122/A
35. AtCoder ABC 218 C - Shapes (形状) - https://atcoder.jp/contests/abc218/tasks/abc218_c
36. USACO Bronze: Block Game (积木游戏) - http://www.usaco.org/index.php?page=viewproblem2&cpid=664
37. 洛谷 P3366 【模板】最小生成树 (模板最小生成树) - https://www.luogu.com.cn/problem/P3366
38. LeetCode 149. Max Points on a Line (直线上最多的点数) - https://leetcode.com/problems/max-points-on-a-line/
39. LeetCode 215. Kth Largest Element in an Array (数组中的第 K 个最大元素) - https://leetcode.com/problems/kth-largest-element-in-an-array/
40. LeetCode 295. Find Median from Data Stream (数据流的中位数) - https://leetcode.com/problems/find-median-from-data-stream/
41. LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数) - https://leetcode.com/problems/count-of-smaller-numbers-after-self/
42. LeetCode 327. Count of Range Sum (区间和的个数) - https://leetcode.com/problems/count-of-range-sum/
43. LeetCode 350. Intersection of Two Arrays II (两个数组的交集 II) - https://leetcode.com/problems/intersection-of-two-arrays-ii/
44. LeetCode 148. Sort List (排序链表) - https://leetcode.com/problems/sort-list/
45. LeetCode 242. Valid Anagram (有效的字母异位词) - https://leetcode.com/problems/valid-anagram/
46. LeetCode 347. Top K Frequent Elements (前 K 个高频元素) - https://leetcode.com/problems/top-k-frequent-elements/
47. LeetCode 451. Sort Characters By Frequency (根据字符出现频率排序) - https://leetcode.com/problems/sort-characters-by-frequency/
48. LeetCode 493. Reverse Pairs (翻转对) - https://leetcode.com/problems/reverse-pairs/
49. LeetCode 539. Minimum Time Difference (最小时间差) - https://leetcode.com/problems/minimum-time-difference/
50. LeetCode 791. Custom Sort String (自定义字符串排序) - https://leetcode.com/problems/custom-sort-string/
"""

```
class DataStreamAsDisjointIntervals:
    """
    LeetCode 352. Data Stream as Disjoint Intervals
    将数据流变为多个不相交区间
    网址：https://leetcode.com/problems/data-stream-as-disjoint-intervals/

    解题思路：
    1. 使用 SortedDict 存储区间边界，键为区间起点，值为区间终点
    2. 添加数字时，查找可能合并的相邻区间
    3. 合并重叠或相邻的区间

    时间复杂度：每次添加 O(log n)，获取区间 O(n)
```

空间复杂度：O(n)
"""

```
def __init__(self):
    # 使用普通字典和排序列表替代 SortedDict
    self.intervals = {}
    self.sorted_keys = []

def addNum(self, val: int) -> None:
    # 使用 bisect 查找插入位置
    import bisect

    # 查找小于等于 val 的区间
    idx = bisect.bisect_right(self.sorted_keys, val) - 1

    if idx >= 0:
        start = self.sorted_keys[idx]
        end = self.intervals[start]
        if start <= val <= end:
            # 数字已经在某个区间内，不需要处理
            return
        elif end + 1 == val:
            # 可以与左侧区间合并
            if idx + 1 < len(self.sorted_keys):
                next_start = self.sorted_keys[idx + 1]
                next_end = self.intervals[next_start]
                if next_start == val + 1:
                    # 同时与右侧区间合并
                    self.intervals[start] = next_end
                    del self.intervals[next_start]
                    self.sorted_keys.pop(idx + 1)
                    return
            self.intervals[start] = val
            return

    # 检查是否可以与右侧区间合并
    if idx + 1 < len(self.sorted_keys):
        next_start = self.sorted_keys[idx + 1]
        next_end = self.intervals[next_start]
        if next_start == val + 1:
            self.intervals[val] = next_end
            del self.intervals[next_start]
            self.sorted_keys[idx + 1] = val
```

```
                self.sorted_keys.sort()
                return

        # 创建新区间
        self.intervals[val] = val
        bisect.insort(self.sorted_keys, val)

    def getIntervals(self) -> List[List[int]]:
        return [[start, end] for start, end in self.intervals.items()]

class MaxSumSubmatrix:
    """
    LeetCode 363. Max Sum of Rectangle No Larger Than K
    矩形区域不超过 K 的最大数值和
    网址：https://leetcode.com/problems/max-sum-of-rectangle-no-larger-than-k/

    解题思路：
    1. 固定左右边界，计算每一行的前缀和
    2. 使用 SortedList 维护前缀和，快速查找满足条件的前缀和
    3. 使用 bisect_left 找到大于等于(target - k)的最小值

    时间复杂度：O(min(m,n)² * max(m,n) log max(m,n))
    空间复杂度：O(max(m,n))
    """

    def maxSumSubmatrix(self, matrix: List[List[int]], k: int) -> int:
        import bisect

        m, n = len(matrix), len(matrix[0])
        max_sum = -sys.maxsize

        # 让 m 是较小的维度，减少时间复杂度
        if m > n:
            rotated = [[matrix[j][i] for j in range(m)] for i in range(n)]
            return self.maxSumSubmatrix(rotated, k)

        # 枚举上下边界
        for top in range(m):
            col_sum = [0] * n
            for bottom in range(top, m):
                # 更新列前缀和
                for j in range(n):
                    col_sum[j] += matrix[bottom][j]
```

```
                # 使用排序列表维护前缀和
                prefix_sums = []
                prefix_sums.append(0)
                current_sum = 0

                for j in range(n):
                    current_sum += col_sum[j]
                    # 查找大于等于(current_sum - k)的最小值
                    target = current_sum - k
                    # 在排序列表中二分查找
                    idx = bisect.bisect_left(prefix_sums, target)
                    if idx < len(prefix_sums):
                        max_sum = max(max_sum, current_sum - prefix_sums[idx])
                    # 插入并保持排序
                    bisect.insort(prefix_sums, current_sum)

        return max_sum

class FindRightInterval:
    """
    LeetCode 436. Find Right Interval
    寻找右区间
    网址：https://leetcode.com/problems/find-right-interval/

    解题思路：
    1. 使用SortedDict存储每个区间的起始位置和索引
    2. 对于每个区间，使用bisect_left找到起始位置大于等于当前区间结束位置的最小区间
    3. 如果找到则返回对应索引，否则返回-1

    时间复杂度：O(n log n)
    空间复杂度：O(n)
    """

    def findRightInterval(self, intervals: List[List[int]]) -> List[int]:
        n = len(intervals)
        result = [-1] * n

        # 存储区间起始位置和索引
        starts = [(intervals[i][0], i) for i in range(n)]
        starts.sort()

        # 提取起始位置列表用于二分查找
```

```
        start_vals = [s[0] for s in starts]

        for i in range(n):
            end = intervals[i][1]
            # 二分查找大于等于 end 的最小起始位置
            idx = bisect.bisect_left(start_vals, end)
            if idx < n:
                result[i] = starts[idx][1]

        return result

class Find132Pattern:
    """
    LeetCode 456. 132 Pattern
    132 模式
    网址：https://leetcode.com/problems/132-pattern/

    解题思路：
    1. 从右向左遍历数组
    2. 使用 SortedList 维护右侧元素
    3. 维护一个变量记录当前最大值作为 3
    4. 检查是否存在 1 小于 3 且 3 小于 2 的模式

    时间复杂度：O(n log n)
    空间复杂度：O(n)
    """

    def find132pattern(self, nums: List[int]) -> bool:
        n = len(nums)
        if n < 3:
            return False

        # 记录左侧最小值
        min_left = [0] * n
        min_left[0] = nums[0]
        for i in range(1, n):
            min_left[i] = min(min_left[i-1], nums[i])

        # 使用排序列表维护右侧元素
        right = []
        right.append(nums[n-1])

        for i in range(n-2, 0, -1):
```

```
            # 当前元素作为3，左侧最小值作为1
            if min_left[i-1] < nums[i]:
                # 在右侧查找大于左侧最小值且小于当前元素的数作为2
                target = min_left[i-1]
                # 在排序列表中查找第一个大于target的元素
                idx = bisect.bisect_right(right, target)
                if idx < len(right) and right[idx] < nums[i]:
                    return True
            # 插入并保持排序
            bisect.insort(right, nums[i])

        return False

class SlidingWindowMedian:
    """
    LeetCode 480. Sliding Window Median
    滑动窗口中位数
    网址：https://leetcode.com/problems/sliding-window-median/

    解题思路：
    1. 使用两个SortedList维护窗口元素
    2. 左半部分存储较小的一半元素，右半部分存储较大的一半元素
    3. 保持两个集合大小平衡，中位数即为左半部分的最大值或两个最大值的平均值

    时间复杂度：O(n log k)
    空间复杂度：O(k)
    """

    def medianSlidingWindow(self, nums: List[int], k: int) -> List[float]:
        import bisect

        n = len(nums)
        result = []

        # 使用两个列表模拟平衡的二叉搜索树
        window = sorted(nums[:k])

        def get_median():
            if k % 2 == 0:
                return (window[k//2-1] + window[k//2]) / 2.0
            else:
                return float(window[k//2])
```

```
        result.append(get_median())

        for i in range(k, n):
            # 移除离开窗口的元素
            left_num = nums[i-k]
            left_idx = bisect.bisect_left(window, left_num)
            window.pop(left_idx)

            # 添加新元素
            bisect.insort(window, nums[i])

            # 计算中位数
            result.append(get_median())

        return result

class KEmptySlots:
    """
    LeetCode 683. K Empty Slots
    K 个空花盆
    网址：https://leetcode.com/problems/k-empty-slots/

    解题思路：
    1. 使用 SortedList 存储开花位置
    2. 按天数顺序添加开花位置
    3. 每次添加后检查新位置与相邻位置的距离是否为 k+1

    时间复杂度：O(n log n)
    空间复杂度：O(n)
    """

    def kEmptySlots(self, flowers: List[int], k: int) -> int:
        bloomed = []

        for day, position in enumerate(flowers, 1):
            # 插入并保持排序
            bisect.insort(bloomed, position)
            idx = bisect.bisect_left(bloomed, position)

            # 检查左侧相邻位置
            if idx > 0:
                left_pos = bloomed[idx-1]
                if position - left_pos - 1 == k:
```

```
                    return day

            # 检查右侧相邻位置
            if idx < len(bloomed) - 1:
                right_pos = bloomed[idx+1]
                if right_pos - position - 1 == k:
                    return day

        return -1

# 测试代码
if __name__ == "__main__":
    # 测试 LeetCode 436
    intervals = [[1,2], [2,3], [0,1], [3,4]]
    solution = FindRightInterval()
    result = solution.findRightInterval(intervals)
    print("LeetCode 436 Result:", result)

    # 测试 LeetCode 456
    nums = [3,1,4,2]
    solution = Find132Pattern()
    has132 = solution.find132pattern(nums)
    print("LeetCode 456 Result:", has132)

    # 测试 LeetCode 683
    flowers = [1,3,2]
    solution = KEmptySlots()
    k_empty = solution.kEmptySlots(flowers, 1)
    print("LeetCode 683 Result:", k_empty)
```

==========

文件: Code05_ComparatorAdvanced.cpp

==========

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <map>
#include <set>
#include <unordered_map>
using namespace std;
```

```
/**
 * Comparator 高级应用题目实现（C++版本）
 * 包含 LeetCode 524, 937, 1331, 1366, 1451, 1509, 1561, 1636, 1710, 1859 等题目
 *
 * C++中 Comparator 的实现：
 * - 使用 lambda 表达式实现自定义比较器
 * - 使用函数对象实现复杂比较逻辑
 * - stable_sort 实现稳定排序
 * - 自定义比较器可以访问私有成员
 *
 * 时间复杂度分析：
 * - 排序操作：O(n log n)
 * - 自定义比较器：O(1)每次比较
 * - 复杂比较逻辑：O(k)每次比较，k 为比较元素的复杂度
 *
 * 空间复杂度分析：
 * - 排序算法：O(log n) 栈空间
 * - 额外数据结构：O(n)
 *
 * 工程化考量：
 * 1. 异常处理：处理空输入、边界条件
 * 2. 性能优化：避免在比较器中创建新对象，使用引用
 * 3. 代码可读性：清晰的比较逻辑和注释
 * 4. 稳定性：注意排序算法的稳定性要求
 *
 * 相关平台题目：
 * 1. LeetCode 524. Longest Word in Dictionary through Deleting (通过删除字母匹配到字典里最长单词) - https://leetcode.com/problems/longest-word-in-dictionary-through-deleting/
 * 2. LeetCode 937. Reorder Data in Log Files (重新排列日志文件) - https://leetcode.com/problems/reorder-data-in-log-files/
 * 3. LeetCode 1331. Rank Transform of an Array (数组序号转换) - https://leetcode.com/problems/rank-transform-of-an-array/
 * 4. LeetCode 1366. Rank Teams by Votes (通过投票对团队排名) - https://leetcode.com/problems/rank-teams-by-votes/
 * 5. LeetCode 1451. Rearrange Words in a Sentence (重新排列句子中的单词) - https://leetcode.com/problems/rearrange-words-in-a-sentence/
 * 6. LeetCode 1509. Minimum Difference Between Largest and Smallest Value in Three Moves (三次操作后最大值与最小值的最小差) - https://leetcode.com/problems/minimum-difference-between-largest-and-smallest-value-in-three-moves/
 * 7. LeetCode 1561. Maximum Number of Coins You Can Get (你可以获得的最大硬币数目) - https://leetcode.com/problems/maximum-number-of-coins-you-can-get/
 * 8. LeetCode 1636. Sort Array by Increasing Frequency (按照频率将数组升序排序) - https://leetcode.com/problems/sort-array-by-increasing-frequency/
```

* 9. LeetCode 1710. Maximum Units on a Truck (卡车上的最大单元数) - https://leetcode.com/problems/maximum-units-on-a-truck/
* 10. LeetCode 1859. Sorting the Sentence (将句子排序) - https://leetcode.com/problems/sorting-the-sentence/
* 11. LintCode 613. High Five (最高分五科) - https://www.lintcode.com/problem/high-five/
* 12. HackerEarth Monk and the Magical Candy Bags (和尚与魔法糖果袋) - https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/practice-problems/algorithm/monk-and-the-magical-candy-bags/
* 13. CodeChef FRGTNLNG (遗忘的语言) - https://www.codechef.com/problems/FRGTNLNG
* 14. SPOJ DICT (字典) - https://www.spoj.com/problems/DICT/
* 15. Project Euler Problem 2: Even Fibonacci numbers (偶数斐波那契数) - https://projecteuler.net/problem=2
* 16. HackerRank Maximum Palindromes (最大回文) - https://www.hackerrank.com/challenges/maximum-palindromes
* 17. 计蒜客 T1101: 阶乘 (阶乘) - https://www.jisuanke.com/t/T1101
* 18. 杭电 OJ 1003: Max Sum (最大子序列和) - http://acm.hdu.edu.cn/showproblem.php?pid=1003
* 19. 牛客网 剑指 Offer 50: 第一个只出现一次的字符 (第一个只出现一次的字符) - https://www.nowcoder.com/practice/1c82e8cf713b4bbeb2a5b31cf5b0417c
* 20. acwing 800. 数组元素的目标和 (数组元素的目标和) - https://www.acwing.com/problem/content/802/
* 21. POJ 1003: Hangover (悬垂) - http://poj.org/problem?id=1003
* 22. UVa OJ 101: The Blocks Problem (积木问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=37
* 23. Timus OJ 1005: Stone Pile (石子堆) - https://acm.timus.ru/problem.aspx?space=1&num=1005
* 24. Aizu OJ ALDS1_5_A: Exhaustive Search (穷举搜索) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_5_A
* 25. Comet OJ Contest #1: 热身赛 B. 简单的数学题 (简单的数学题) - https://cometoj.com/contest/1/problem/B
* 26. MarsCode 火星编程竞赛: 数字统计 (数字统计) - https://www.marscode.cn/contest/1/problem/1002
* 27. ZOJ 1002: Fire Net (消防网) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1002
* 28. LOJ 101: 最小生成树 (最小生成树) - https://loj.ac/p/101
* 29. 各大高校 OJ: 北京大学 OJ 1000: A+B Problem (A+B 问题) - http://poj.org/problem?id=1000
* 30. Codeforces 122A. Lucky Division (幸运分割) - https://codeforces.com/problemset/problem/122/A
* 31. AtCoder ABC 218 C - Shapes (形状) - https://atcoder.jp/contests/abc218/tasks/abc218_c
* 32. USACO Bronze: Block Game (积木游戏) - http://www.usaco.org/index.php?page=viewproblem2&cpid=664
* 33. 洛谷 P3366 【模板】最小生成树 (模板最小生成树) - https://www.luogu.com.cn/problem/P3366
* 34. LeetCode 149. Max Points on a Line (直线上最多的点数) - https://leetcode.com/problems/max-points-on-a-line/
* 35. LeetCode 215. Kth Largest Element in an Array (数组中的第 K 个最大元素) -

```
https://leetcode.com/problems/kth-largest-element-in-an-array/
 * 36. LeetCode 295. Find Median from Data Stream (数据流的中位数) -
https://leetcode.com/problems/find-median-from-data-stream/
 * 37. LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数) -
https://leetcode.com/problems/count-of-smaller-numbers-after-self/
 * 38. LeetCode 327. Count of Range Sum (区间和的个数) - https://leetcode.com/problems/count-of-
range-sum/
 * 39. LeetCode 350. Intersection of Two Arrays II (两个数组的交集 II) -
https://leetcode.com/problems/intersection-of-two-arrays-ii/
 * 40. LeetCode 148. Sort List (排序链表) - https://leetcode.com/problems/sort-list/
 * 41. LeetCode 242. Valid Anagram (有效的字母异位词) - https://leetcode.com/problems/valid-
anagram/
 * 42. LeetCode 347. Top K Frequent Elements (前 K 个高频元素) -
https://leetcode.com/problems/top-k-frequent-elements/
 * 43. LeetCode 451. Sort Characters By Frequency (根据字符出现频率排序) -
https://leetcode.com/problems/sort-characters-by-frequency/
 * 44. LeetCode 493. Reverse Pairs (翻转对) - https://leetcode.com/problems/reverse-pairs/
 * 45. LeetCode 539. Minimum Time Difference (最小时间差) -
https://leetcode.com/problems/minimum-time-difference/
 * 46. LeetCode 791. Custom Sort String (自定义字符串排序) -
https://leetcode.com/problems/custom-sort-string/
 * 47. LeetCode 1030. Matrix Cells in Distance Order (距离顺序排列矩阵单元格) -
https://leetcode.com/problems/matrix-cells-in-distance-order/
 * 48. LeetCode 1122. Relative Sort Array (数组的相对排序) -
https://leetcode.com/problems/relative-sort-array/
 * 49. LeetCode 1356. Sort Integers by The Number of 1 Bits (根据数字二进制下 1 的数目排序) -
https://leetcode.com/problems/sort-integers-by-the-number-of-1-bits/
 * 50. LeetCode 179. Largest Number (最大数) - https://leetcode.com/problems/largest-number/
 */

/**
 * LeetCode 524. Longest Word in Dictionary through Deleting
 * 通过删除字母匹配到字典里最长单词
 * 网址：https://leetcode.com/problems/longest-word-in-dictionary-through-deleting/
 *
 * 解题思路：
 * 1. 使用自定义 Comparator 按长度降序、字典序升序排序
 * 2. 检查每个单词是否可以通过删除 s 中的字符得到
 * 3. 返回第一个满足条件的单词
 *
 * 时间复杂度：O(n * x * log n)，其中 x 是单词平均长度
 * 空间复杂度：O(1)
 */
```

```
class LongestWordInDictionary {
private:
    bool isSubsequence(const string& word, const string& s) {
        int i = 0, j = 0;
        while (i < word.length() && j < s.length()) {
            if (word[i] == s[j]) {
                i++;
            }
            j++;
        }
        return i == word.length();
    }

public:
    string findLongestWord(string s, vector<string>& dictionary) {
        // 自定义比较器：先按长度降序，再按字典序升序
        sort(dictionary.begin(), dictionary.end(), [](const string& a, const string& b) {
            if (a.length() != b.length()) {
                return a.length() > b.length(); // 长度降序
            }
            return a < b; // 字典序升序
        });

        for (const string& word : dictionary) {
            if (isSubsequence(word, s)) {
                return word;
            }
        }
        return "";
    }
};

/**
 * LeetCode 937. Reorder Data in Log Files
 * 重新排列日志文件
 * 网址：https://leetcode.com/problems/reorder-data-in-log-files/
 *
 * 解题思路：
 * 1. 区分字母日志和数字日志
 * 2. 字母日志按内容字典序排序，内容相同按标识符排序
 * 3. 数字日志保持原有顺序
 *
 * 时间复杂度：O(n log n)
```

```
 * 空间复杂度：O(n)
 */
class ReorderLogFiles {
public:
    vector<string> reorderLogFiles(vector<string>& logs) {
        // 自定义比较器
        sort(logs.begin(), logs.end(), [](const string& a, const string& b) {
            // 找到第一个空格的位置
            size_t spaceA = a.find(' ');
            size_t spaceB = b.find(' ');

            string identifierA = a.substr(0, spaceA);
            string contentA = a.substr(spaceA + 1);
            string identifierB = b.substr(0, spaceB);
            string contentB = b.substr(spaceB + 1);

            bool isDigitA = isdigit(contentA[0]);
            bool isDigitB = isdigit(contentB[0]);

            // 情况 1：都是字母日志
            if (!isDigitA && !isDigitB) {
                if (contentA != contentB) {
                    return contentA < contentB;
                }
                return identifierA < identifierB;
            }

            // 情况 2：一个是字母日志，一个是数字日志
            if (!isDigitA && isDigitB) {
                return true; // 字母日志在前
            } else if (isDigitA && !isDigitB) {
                return false; // 数字日志在后
            } else {
                // 情况 3：都是数字日志，保持原有顺序
                return false;
            }
        });

        return logs;
    }
};

/**
```

```
 * LeetCode 1331. Rank Transform of an Array
 * 数组序号转换
 * 网址：https://leetcode.com/problems/rank-transform-of-an-array/
 *
 * 解题思路：
 * 1. 创建排序后的数组副本
 * 2. 使用 map 存储值到排名的映射
 * 3. 根据映射转换原数组
 *
 * 时间复杂度：O(n log n)
 * 空间复杂度：O(n)
 */
class RankTransformArray {
public:
    vector<int> arrayRankTransform(vector<int>& arr) {
        if (arr.empty()) return {};

        // 创建排序后的数组副本
        vector<int> sortedArr = arr;
        sort(sortedArr.begin(), sortedArr.end());

        // 创建值到排名的映射
        map<int, int> rankMap;
        int rank = 1;
        for (int num : sortedArr) {
            if (rankMap.find(num) == rankMap.end()) {
                rankMap[num] = rank++;
            }
        }

        // 转换原数组
        vector<int> result;
        for (int num : arr) {
            result.push_back(rankMap[num]);
        }

        return result;
    }
};

/**
 * LeetCode 1366. Rank Teams by Votes
 * 通过投票对团队排名
```

```
 * 网址：https://leetcode.com/problems/rank-teams-by-votes/
 *
 * 解题思路：
 * 1. 统计每个团队在每个位置的得票数
 * 2. 使用自定义 Comparator 比较投票统计
 * 3. 按得票规则排序团队
 *
 * 时间复杂度：O(n * m + n log n)，其中 m 是投票位置数
 * 空间复杂度：O(n²)
 */
class RankTeamsByVotes {
public:
    string rankTeams(vector<string>& votes) {
        if (votes.empty()) return "";
        if (votes.size() == 1) return votes[0];

        int n = votes[0].length();
        // 统计每个团队在每个位置的得票数
        unordered_map<char, vector<int>> voteCount;
        for (char team : votes[0]) {
            voteCount[team] = vector<int>(n, 0);
        }

        for (const string& vote : votes) {
            for (int i = 0; i < n; i++) {
                voteCount[vote[i]][i]++;
            }
        }

        // 创建团队列表
        vector<char> teams;
        for (char team : votes[0]) {
            teams.push_back(team);
        }

        // 自定义比较器
        sort(teams.begin(), teams.end(), [&](char a, char b) {
            for (int i = 0; i < n; i++) {
                if (voteCount[a][i] != voteCount[b][i]) {
                    return voteCount[a][i] > voteCount[b][i]; // 得票数降序
                }
            }
            return a < b; // 字母序升序
```

```
        });

        return string(teams.begin(), teams.end());
    }
};

/**
 * LeetCode 1451. Rearrange Words in a Sentence
 * 重新排列句子中的单词
 * 网址：https://leetcode.com/problems/rearrange-words-in-a-text/
 *
 * 解题思路：
 * 1. 按单词长度排序
 * 2. 长度相同保持原有顺序（稳定排序）
 * 3. 首字母大写处理
 *
 * 时间复杂度：O(n log n)
 * 空间复杂度：O(n)
 */
class RearrangeWords {
public:
    string arrangeWords(string text) {
        if (text.empty()) return text;

        // 转换为小写
        transform(text.begin(), text.end(), text.begin(), ::tolower);

        // 分割单词
        vector<string> words;
        string word;
        for (char c : text) {
            if (c == ' ') {
                if (!word.empty()) {
                    words.push_back(word);
                    word.clear();
                }
            } else {
                word += c;
            }
        }
        if (!word.empty()) {
            words.push_back(word);
        }
```

```
        // 稳定排序：按长度排序
        stable_sort(words.begin(), words.end(), [](const string& a, const string& b) {
            return a.length() < b.length();
        });

        // 首字母大写处理
        if (!words.empty() && !words[0].empty()) {
            words[0][0] = toupper(words[0][0]);
        }

        // 重新组合成句子
        string result;
        for (size_t i = 0; i < words.size(); i++) {
            if (i > 0) result += " ";
            result += words[i];
        }

        return result;
    }
};

// 测试函数
int main() {
    // 测试 LeetCode 524
    LongestWordInDictionary solution1;
    string s = "abpcplea";
    vector<string> dictionary = {"ale", "apple", "monkey", "plea"};
    string longestWord = solution1.findLongestWord(s, dictionary);
    cout << "LeetCode 524 Result: " << longestWord << endl;

    // 测试 LeetCode 1331
    RankTransformArray solution2;
    vector<int> arr = {40, 10, 20, 30};
    vector<int> ranks = solution2.arrayRankTransform(arr);
    cout << "LeetCode 1331 Result: ";
    for (int rank : ranks) cout << rank << " ";
    cout << endl;

    return 0;
}

==========================================
```

文件: Code05_ComparatorAdvanced.java
==========

```
import java.util.*;

/**
 * Comparator 高级应用题目实现
 * 包含 LeetCode 524, 937, 1331, 1366, 1451, 1509, 1561, 1636, 1710, 1859 等题目
 *
 * Comparator 高级特性：
 * - 多级排序：按多个条件依次排序
 * - 稳定排序：保持相同元素的相对顺序
 * - 自定义排序逻辑：实现复杂的排序规则
 * - Lambda 表达式：简化比较器实现
 *
 * 时间复杂度分析：
 * - 排序操作：O(n log n)
 * - 自定义比较器：O(1)每次比较
 * - 复杂比较逻辑：O(k)每次比较，k 为比较元素的复杂度
 *
 * 空间复杂度分析：
 * - 排序算法：O(log n) 栈空间
 * - 额外数据结构：O(n)
 *
 * 工程化考量：
 * 1. 异常处理：处理空输入、边界条件
 * 2. 性能优化：避免在比较器中创建新对象，使用缓存
 * 3. 代码可读性：清晰的比较逻辑和注释
 * 4. 稳定性：注意排序算法的稳定性要求
 *
 * 相关平台题目：
 * 1. LeetCode 524. Longest Word in Dictionary through Deleting (通过删除字母匹配到字典里最长单词) - https://leetcode.com/problems/longest-word-in-dictionary-through-deleting/
 * 2. LeetCode 937. Reorder Data in Log Files (重新排列日志文件) - https://leetcode.com/problems/reorder-data-in-log-files/
 * 3. LeetCode 1331. Rank Transform of an Array (数组序号转换) - https://leetcode.com/problems/rank-transform-of-an-array/
 * 4. LeetCode 1366. Rank Teams by Votes (通过投票对团队排名) - https://leetcode.com/problems/rank-teams-by-votes/
 * 5. LeetCode 1451. Rearrange Words in a Sentence (重新排列句子中的单词) - https://leetcode.com/problems/rearrange-words-in-a-sentence/
 * 6. LeetCode 1509. Minimum Difference Between Largest and Smallest Value in Three Moves (三次操作后最大值与最小值的最小差) - https://leetcode.com/problems/minimum-difference-between-largest-
```

and-smallest-value-in-three-moves/
 * 7. LeetCode 1561. Maximum Number of Coins You Can Get (你可以获得的最大硬币数目) - https://leetcode.com/problems/maximum-number-of-coins-you-can-get/
 * 8. LeetCode 1636. Sort Array by Increasing Frequency (按照频率将数组升序排序) - https://leetcode.com/problems/sort-array-by-increasing-frequency/
 * 9. LeetCode 1710. Maximum Units on a Truck (卡车上的最大单元数) - https://leetcode.com/problems/maximum-units-on-a-truck/
 * 10. LeetCode 1859. Sorting the Sentence (将句子排序) - https://leetcode.com/problems/sorting-the-sentence/
 * 11. LintCode 613. High Five (最高分五科) - https://www.lintcode.com/problem/high-five/
 * 12. HackerEarth Monk and the Magical Candy Bags (和尚与魔法糖果袋) - https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/practice-problems/algorithm/monk-and-the-magical-candy-bags/
 * 13. CodeChef FRGTNLNG (遗忘的语言) - https://www.codechef.com/problems/FRGTNLNG
 * 14. SPOJ DICT (字典) - https://www.spoj.com/problems/DICT/
 * 15. Project Euler Problem 2: Even Fibonacci numbers (偶数斐波那契数) - https://projecteuler.net/problem=2
 * 16. HackerRank Maximum Palindromes (最大回文) - https://www.hackerrank.com/challenges/maximum-palindromes
 * 17. 计蒜客 T1101: 阶乘 (阶乘) - https://www.jisuanke.com/t/T1101
 * 18. 杭电 OJ 1003: Max Sum (最大子序列和) - http://acm.hdu.edu.cn/showproblem.php?pid=1003
 * 19. 牛客网 剑指 Offer 50: 第一个只出现一次的字符 (第一个只出现一次的字符) - https://www.nowcoder.com/practice/1c82e8cf713b4bbeb2a5b31cf5b0417c
 * 20. acwing 800. 数组元素的目标和 (数组元素的目标和) - https://www.acwing.com/problem/content/802/
 * 21. POJ 1003: Hangover (悬垂) - http://poj.org/problem?id=1003
 * 22. UVa OJ 101: The Blocks Problem (积木问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=37
 * 23. Timus OJ 1005: Stone Pile (石子堆) - https://acm.timus.ru/problem.aspx?space=1&num=1005
 * 24. Aizu OJ ALDS1_5_A: Exhaustive Search (穷举搜索) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_5_A
 * 25. Comet OJ Contest #1: 热身赛 B. 简单的数学题 (简单的数学题) - https://cometoj.com/contest/1/problem/B
 * 26. MarsCode 火星编程竞赛: 数字统计 (数字统计) - https://www.marscode.cn/contest/1/problem/1002
 * 27. ZOJ 1002: Fire Net (消防网) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1002
 * 28. LOJ 101: 最小生成树 (最小生成树) - https://loj.ac/p/101
 * 29. 各大高校 OJ: 北京大学 OJ 1000: A+B Problem (A+B 问题) - http://poj.org/problem?id=1000
 * 30. Codeforces 122A. Lucky Division (幸运分割) - https://codeforces.com/problemset/problem/122/A
 * 31. AtCoder ABC 218 C - Shapes (形状) - https://atcoder.jp/contests/abc218/tasks/abc218_c
 * 32. USACO Bronze: Block Game (积木游戏) -

http://www.usaco.org/index.php?page=viewproblem2&cpid=664
 * 33. 洛谷 P3366 【模板】最小生成树（模板最小生成树） - https://www.luogu.com.cn/problem/P3366
 * 34. LeetCode 149. Max Points on a Line (直线上最多的点数) - https://leetcode.com/problems/max-points-on-a-line/
 * 35. LeetCode 215. Kth Largest Element in an Array (数组中的第K个最大元素) - https://leetcode.com/problems/kth-largest-element-in-an-array/
 * 36. LeetCode 295. Find Median from Data Stream (数据流的中位数) - https://leetcode.com/problems/find-median-from-data-stream/
 * 37. LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数) - https://leetcode.com/problems/count-of-smaller-numbers-after-self/
 * 38. LeetCode 327. Count of Range Sum (区间和的个数) - https://leetcode.com/problems/count-of-range-sum/
 * 39. LeetCode 350. Intersection of Two Arrays II (两个数组的交集 II) - https://leetcode.com/problems/intersection-of-two-arrays-ii/
 * 40. LeetCode 148. Sort List (排序链表) - https://leetcode.com/problems/sort-list/
 * 41. LeetCode 242. Valid Anagram (有效的字母异位词) - https://leetcode.com/problems/valid-anagram/
 * 42. LeetCode 347. Top K Frequent Elements (前 K 个高频元素) - https://leetcode.com/problems/top-k-frequent-elements/
 * 43. LeetCode 451. Sort Characters By Frequency (根据字符出现频率排序) - https://leetcode.com/problems/sort-characters-by-frequency/
 * 44. LeetCode 493. Reverse Pairs (翻转对) - https://leetcode.com/problems/reverse-pairs/
 * 45. LeetCode 539. Minimum Time Difference (最小时间差) - https://leetcode.com/problems/minimum-time-difference/
 * 46. LeetCode 791. Custom Sort String (自定义字符串排序) - https://leetcode.com/problems/custom-sort-string/
 * 47. LeetCode 1030. Matrix Cells in Distance Order (距离顺序排列矩阵单元格) - https://leetcode.com/problems/matrix-cells-in-distance-order/
 * 48. LeetCode 1122. Relative Sort Array (数组的相对排序) - https://leetcode.com/problems/relative-sort-array/
 * 49. LeetCode 1356. Sort Integers by The Number of 1 Bits (根据数字二进制下 1 的数目排序) - https://leetcode.com/problems/sort-integers-by-the-number-of-1-bits/
 * 50. LeetCode 179. Largest Number (最大数) - https://leetcode.com/problems/largest-number/

```
public class Code05_ComparatorAdvanced {

    /**
     * LeetCode 524. Longest Word in Dictionary through Deleting
     * 通过删除字母匹配到字典里最长单词
     * 网址：https://leetcode.com/problems/longest-word-in-dictionary-through-deleting/
     *
     * 解题思路：
     * 1. 使用自定义Comparator按长度降序、字典序升序排序
```

```
     * 2. 检查每个单词是否可以通过删除 s 中的字符得到
     * 3. 返回第一个满足条件的单词
     *
     * 时间复杂度：O(n * x * log n)，其中 x 是单词平均长度
     * 空间复杂度：O(log n) 排序栈空间
     */
    public String findLongestWord(String s, List<String> dictionary) {
        // 自定义比较器：先按长度降序，再按字典序升序
        Collections.sort(dictionary, new Comparator<String>() {
            @Override
            public int compare(String a, String b) {
                if (a.length() != b.length()) {
                    return Integer.compare(b.length(), a.length()); // 长度降序
                }
                return a.compareTo(b); // 字典序升序
            }
        });

        for (String word : dictionary) {
            if (isSubsequence(word, s)) {
                return word;
            }
        }
        return "";
    }

    private boolean isSubsequence(String word, String s) {
        int i = 0, j = 0;
        while (i < word.length() && j < s.length()) {
            if (word.charAt(i) == s.charAt(j)) {
                i++;
            }
            j++;
        }
        return i == word.length();
    }

    /**
     * LeetCode 937. Reorder Data in Log Files
     * 重新排列日志文件
     * 网址：https://leetcode.com/problems/reorder-data-in-log-files/
     *
     * 解题思路：
```

```
     * 1. 区分字母日志和数字日志
     * 2. 字母日志按内容字典序排序，内容相同按标识符排序
     * 3. 数字日志保持原有顺序
     *
     * 时间复杂度：O(n log n)
     * 空间复杂度：O(n)
     */
    public String[] reorderLogFiles(String[] logs) {
        Arrays.sort(logs, new Comparator<String>() {
            @Override
            public int compare(String log1, String log2) {
                // 分割标识符和内容
                String[] split1 = log1.split(" ", 2);
                String[] split2 = log2.split(" ", 2);

                boolean isDigit1 = Character.isDigit(split1[1].charAt(0));
                boolean isDigit2 = Character.isDigit(split2[1].charAt(0));

                // 情况 1：都是字母日志
                if (!isDigit1 && !isDigit2) {
                    int cmp = split1[1].compareTo(split2[1]);
                    if (cmp != 0) {
                        return cmp;
                    }
                    // 内容相同，按标识符排序
                    return split1[0].compareTo(split2[0]);
                }

                // 情况 2：一个是字母日志，一个是数字日志
                if (!isDigit1 && isDigit2) {
                    return -1; // 字母日志在前
                } else if (isDigit1 && !isDigit2) {
                    return 1;  // 数字日志在后
                } else {
                    // 情况 3：都是数字日志，保持原有顺序
                    return 0;
                }
            }
        });

        return logs;
    }
```

```
/**
 * LeetCode 1331. Rank Transform of an Array
 * 数组序号转换
 * 网址：https://leetcode.com/problems/rank-transform-of-an-array/
 *
 * 解题思路：
 * 1. 创建索引数组并排序
 * 2. 使用自定义 Comparator 按元素值排序
 * 3. 分配排名，相同值排名相同
 *
 * 时间复杂度：O(n log n)
 * 空间复杂度：O(n)
 */
public int[] arrayRankTransform(int[] arr) {
    int n = arr.length;
    if (n == 0) return new int[0];

    // 创建索引数组
    Integer[] indices = new Integer[n];
    for (int i = 0; i < n; i++) {
        indices[i] = i;
    }

    // 按元素值排序索引
    Arrays.sort(indices, new Comparator<Integer>() {
        @Override
        public int compare(Integer i, Integer j) {
            return Integer.compare(arr[i], arr[j]);
        }
    });

    int[] result = new int[n];
    int rank = 1;
    result[indices[0]] = rank;

    // 分配排名
    for (int i = 1; i < n; i++) {
        if (arr[indices[i]] != arr[indices[i-1]]) {
            rank++;
        }
        result[indices[i]] = rank;
    }
```

```
    return result;
}

/**
 * LeetCode 1366. Rank Teams by Votes
 * 通过投票对团队排名
 * 网址：https://leetcode.com/problems/rank-teams-by-votes/
 *
 * 解题思路：
 * 1. 统计每个团队在每个位置的得票数
 * 2. 使用自定义 Comparator 比较投票统计
 * 3. 按得票规则排序团队
 *
 * 时间复杂度：O(n * m + n log n)，其中 m 是投票位置数
 * 空间复杂度：O(n²)
 */
public String rankTeams(String[] votes) {
    if (votes.length == 0) return "";
    if (votes.length == 1) return votes[0];

    int n = votes[0].length();
    // 统计每个团队在每个位置的得票数
    Map<Character, int[]> voteCount = new HashMap<>();
    for (String vote : votes) {
        for (int i = 0; i < n; i++) {
            char team = vote.charAt(i);
            voteCount.putIfAbsent(team, new int[n]);
            voteCount.get(team)[i]++;
        }
    }

    // 创建团队列表
    List<Character> teams = new ArrayList<>(voteCount.keySet());

    // 自定义比较器
    Collections.sort(teams, new Comparator<Character>() {
        @Override
        public int compare(Character a, Character b) {
            int[] votesA = voteCount.get(a);
            int[] votesB = voteCount.get(b);

            for (int i = 0; i < n; i++) {
                if (votesA[i] != votesB[i]) {
```

```
                    return Integer.compare(votesB[i], votesA[i]); // 得票数降序
                }
            }
            return Character.compare(a, b); // 字母序升序
        }
    });

    // 构建结果字符串
    StringBuilder result = new StringBuilder();
    for (char team : teams) {
        result.append(team);
    }
    return result.toString();
}

/**
 * LeetCode 1451. Rearrange Words in a Sentence
 * 重新排列句子中的单词
 * 网址：https://leetcode.com/problems/rearrange-words-in-a-text/
 *
 * 解题思路：
 * 1. 按单词长度排序
 * 2. 长度相同保持原有顺序（稳定排序）
 * 3. 首字母大写处理
 *
 * 时间复杂度：O(n log n)
 * 空间复杂度：O(n)
 */
public String arrangeWords(String text) {
    if (text == null || text.isEmpty()) return text;

    // 分割单词
    String[] words = text.toLowerCase().split(" ");

    // 创建索引数组
    Integer[] indices = new Integer[words.length];
    for (int i = 0; i < words.length; i++) {
        indices[i] = i;
    }

    // 按单词长度排序，保持稳定性
    Arrays.sort(indices, new Comparator<Integer>() {
        @Override
```

```
            public int compare(Integer i, Integer j) {
                return Integer.compare(words[i].length(), words[j].length());
            }
        });

        // 构建结果
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < indices.length; i++) {
            if (i == 0) {
                // 首字母大写
                String word = words[indices[i]];
                if (!word.isEmpty()) {
                    result.append(Character.toUpperCase(word.charAt(0)));
                    result.append(word.substring(1));
                }
            } else {
                result.append(words[indices[i]]);
            }
            if (i < indices.length - 1) {
                result.append(" ");
            }
        }

        return result.toString();
    }

    /**
     * LeetCode 1509. Minimum Difference Between Largest and Smallest Value in Three Moves
     * 三次操作后最大值与最小值的最小差
     * 网址：https://leetcode.com/problems/minimum-difference-between-largest-and-smallest-value-in-three-moves/
     *
     * 解题思路：
     * 1. 排序数组
     * 2. 分析移除三个元素后的可能情况
     * 3. 计算最小差值
     *
     * 时间复杂度：O(n log n)
     * 空间复杂度：O(1)
     */
    public int minDifference(int[] nums) {
        int n = nums.length;
        if (n <= 4) return 0;
```

```
    Arrays.sort(nums);

    // 四种可能的情况：
    // 1. 移除最小的 3 个元素
    // 2. 移除最小的 2 个和最大的 1 个
    // 3. 移除最小的 1 个和最大的 2 个
    // 4. 移除最大的 3 个元素

    int minDiff = Integer.MAX_VALUE;
    minDiff = Math.min(minDiff, nums[n-1] - nums[3]);     // 移除最小的 3 个
    minDiff = Math.min(minDiff, nums[n-2] - nums[2]);     // 移除最小的 2 个和最大的 1 个
    minDiff = Math.min(minDiff, nums[n-3] - nums[1]);     // 移除最小的 1 个和最大的 2 个
    minDiff = Math.min(minDiff, nums[n-4] - nums[0]);     // 移除最大的 3 个

    return minDiff;
}

/**
 * LeetCode 1561. Maximum Number of Coins You Can Get
 * 你可以获得的最大硬币数目
 * 网址：https://leetcode.com/problems/maximum-number-of-coins-you-can-get/
 *
 * 解题思路：
 * 1. 排序数组
 * 2. 每次取第二大的堆（贪心策略）
 * 3. 计算你能获得的总硬币数
 *
 * 时间复杂度：O(n log n)
 * 空间复杂度：O(1)
 */
public int maxCoins(int[] piles) {
    Arrays.sort(piles);
    int n = piles.length;
    int result = 0;

    // 每次取第二大的堆
    for (int i = n - 2; i >= n / 3; i -= 2) {
        result += piles[i];
    }

    return result;
}
```

```
/**
 * LeetCode 1636. Sort Array by Increasing Frequency
 * 按照频率将数组升序排序
 * 网址：https://leetcode.com/problems/sort-array-by-increasing-frequency/
 *
 * 解题思路：
 * 1. 统计每个数字的频率
 * 2. 使用自定义 Comparator 按频率升序排序
 * 3. 频率相同按数值降序排序
 *
 * 时间复杂度：O(n log n)
 * 空间复杂度：O(n)
 */
public int[] frequencySort(int[] nums) {
    // 统计频率
    Map<Integer, Integer> freq = new HashMap<>();
    for (int num : nums) {
        freq.put(num, freq.getOrDefault(num, 0) + 1);
    }

    // 转换为 Integer 数组用于排序
    Integer[] numsObj = new Integer[nums.length];
    for (int i = 0; i < nums.length; i++) {
        numsObj[i] = nums[i];
    }

    // 自定义比较器
    Arrays.sort(numsObj, new Comparator<Integer>() {
        @Override
        public int compare(Integer a, Integer b) {
            int freqA = freq.get(a);
            int freqB = freq.get(b);
            if (freqA != freqB) {
                return Integer.compare(freqA, freqB); // 频率升序
            }
            return Integer.compare(b, a); // 数值降序
        }
    });

    // 转换回 int 数组
    for (int i = 0; i < nums.length; i++) {
        nums[i] = numsObj[i];
```

```
    }

    return nums;
}

/**
 * LeetCode 1710. Maximum Units on a Truck
 * 卡车上的最大单元数
 * 网址：https://leetcode.com/problems/maximum-units-on-a-truck/
 *
 * 解题思路：
 * 1. 按单位单元数（每个箱子的单元数）降序排序
 * 2. 贪心选择单位单元数最大的箱子
 * 3. 计算最大单元数
 *
 * 时间复杂度：O(n log n)
 * 空间复杂度：O(1)
 */
public int maximumUnits(int[][] boxTypes, int truckSize) {
    // 按单位单元数降序排序
    Arrays.sort(boxTypes, new Comparator<int[]>() {
        @Override
        public int compare(int[] a, int[] b) {
            return Integer.compare(b[1], a[1]); // 单位单元数降序
        }
    });

    int result = 0;
    int remaining = truckSize;

    for (int[] box : boxTypes) {
        if (remaining <= 0) break;

        int boxesToTake = Math.min(remaining, box[0]);
        result += boxesToTake * box[1];
        remaining -= boxesToTake;
    }

    return result;
}

/**
 * LeetCode 1859. Sorting the Sentence
```

```
     * 将句子排序
     * 网址：https://leetcode.com/problems/sorting-the-sentence/
     *
     * 解题思路：
     * 1. 提取单词末尾的数字
     * 2. 按数字排序单词
     * 3. 重新组合成句子
     *
     * 时间复杂度：O(n log n)
     * 空间复杂度：O(n)
     */
    public String sortSentence(String s) {
        String[] words = s.split(" ");

        // 自定义比较器：按末尾数字排序
        Arrays.sort(words, new Comparator<String>() {
            @Override
            public int compare(String a, String b) {
                int numA = Integer.parseInt(a.substring(a.length() - 1));
                int numB = Integer.parseInt(b.substring(b.length() - 1));
                return Integer.compare(numA, numB);
            }
        });

        // 构建结果句子，去除末尾数字
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < words.length; i++) {
            result.append(words[i].substring(0, words[i].length() - 1));
            if (i < words.length - 1) {
                result.append(" ");
            }
        }

        return result.toString();
    }

    /**
     * 测试方法
     */
    public static void main(String[] args) {
        Code05_ComparatorAdvanced solution = new Code05_ComparatorAdvanced();

        // 测试 LeetCode 524
```

```
        String s = "abpcplea";
        List<String> dictionary = Arrays.asList("ale", "apple", "monkey", "plea");
        String longestWord = solution.findLongestWord(s, dictionary);
        System.out.println("LeetCode 524 Result: " + longestWord);

        // 测试 LeetCode 937
        String[] logs = {"dig1 8 1 5 1", "let1 art can", "dig2 3 6", "let2 own kit dig", "let3
art zero"};
        String[] reorderedLogs = solution.reorderLogFiles(logs);
        System.out.println("LeetCode 937 Result: " + Arrays.toString(reorderedLogs));

        // 测试 LeetCode 1331
        int[] arr = {40, 10, 20, 30};
        int[] ranks = solution.arrayRankTransform(arr);
        System.out.println("LeetCode 1331 Result: " + Arrays.toString(ranks));

        // 测试 LeetCode 1710
        int[][] boxTypes = {{1,3}, {2,2}, {3,1}};
        int maxUnits = solution.maximumUnits(boxTypes, 4);
        System.out.println("LeetCode 1710 Result: " + maxUnits);
    }
}
```

==========

文件: Code05_ComparatorAdvanced.py

==========

```
from typing import List
import sys

"""
Comparator 高级应用题目实现（Python 版本）
包含 LeetCode 524, 937, 1331, 1366, 1451, 1509, 1561, 1636, 1710, 1859 等题目

Python 中 Comparator 的实现：
- Python 使用 key 参数实现自定义排序
- 使用 lambda 表达式简化排序逻辑
- sorted 函数是稳定排序
- functools.cmp_to_key 可以将比较函数转换为 key 函数

时间复杂度分析：
- 排序操作：O(n log n)
- 自定义比较器：O(1)每次比较
```

- 复杂比较逻辑：O(k)每次比较，k 为比较元素的复杂度

空间复杂度分析：
- 排序算法：O(n) 临时空间
- 额外数据结构：O(n)

工程化考量：
1. 异常处理：处理空输入、边界条件
2. 性能优化：使用 key 参数避免重复计算
3. 代码可读性：清晰的比较逻辑和注释
4. 稳定性：Python 的 sorted 是稳定排序

相关平台题目：
1. LeetCode 524. Longest Word in Dictionary through Deleting (通过删除字母匹配到字典里最长单词) - https://leetcode.com/problems/longest-word-in-dictionary-through-deleting/
2. LeetCode 937. Reorder Data in Log Files (重新排列日志文件) - https://leetcode.com/problems/reorder-data-in-log-files/
3. LeetCode 1331. Rank Transform of an Array (数组序号转换) - https://leetcode.com/problems/rank-transform-of-an-array/
4. LeetCode 1366. Rank Teams by Votes (通过投票对团队排名) - https://leetcode.com/problems/rank-teams-by-votes/
5. LeetCode 1451. Rearrange Words in a Sentence (重新排列句子中的单词) - https://leetcode.com/problems/rearrange-words-in-a-sentence/
6. LeetCode 1509. Minimum Difference Between Largest and Smallest Value in Three Moves (三次操作后最大值与最小值的最小差) - https://leetcode.com/problems/minimum-difference-between-largest-and-smallest-value-in-three-moves/
7. LeetCode 1561. Maximum Number of Coins You Can Get (你可以获得的最大硬币数目) - https://leetcode.com/problems/maximum-number-of-coins-you-can-get/
8. LeetCode 1636. Sort Array by Increasing Frequency (按照频率将数组升序排序) - https://leetcode.com/problems/sort-array-by-increasing-frequency/
9. LeetCode 1710. Maximum Units on a Truck (卡车上的最大单元数) - https://leetcode.com/problems/maximum-units-on-a-truck/
10. LeetCode 1859. Sorting the Sentence (将句子排序) - https://leetcode.com/problems/sorting-the-sentence/
11. LintCode 613. High Five (最高分五科) - https://www.lintcode.com/problem/high-five/
12. HackerEarth Monk and the Magical Candy Bags (和尚与魔法糖果袋) - https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/practice-problems/algorithm/monk-and-the-magical-candy-bags/
13. CodeChef FRGTNLNG (遗忘的语言) - https://www.codechef.com/problems/FRGTNLNG
14. SPOJ DICT (字典) - https://www.spoj.com/problems/DICT/
15. Project Euler Problem 2: Even Fibonacci numbers (偶数斐波那契数) - https://projecteuler.net/problem=2
16. HackerRank Maximum Palindromes (最大回文) - https://www.hackerrank.com/challenges/maximum-

palindromes
17. 计蒜客 T1101: 阶乘 (阶乘) - https://www.jisuanke.com/t/T1101
18. 杭电 OJ 1003: Max Sum (最大子序列和) - http://acm.hdu.edu.cn/showproblem.php?pid=1003
19. 牛客网 剑指 Offer 50: 第一个只出现一次的字符 (第一个只出现一次的字符) - https://www.nowcoder.com/practice/1c82e8cf713b4bbeb2a5b31cf5b0417c
20. acwing 800. 数组元素的目标和 (数组元素的目标和) - https://www.acwing.com/problem/content/802/
21. POJ 1003: Hangover (悬垂) - http://poj.org/problem?id=1003
22. UVa OJ 101: The Blocks Problem (积木问题) - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=37
23. Timus OJ 1005: Stone Pile (石子堆) - https://acm.timus.ru/problem.aspx?space=1&num=1005
24. Aizu OJ ALDS1_5_A: Exhaustive Search (穷举搜索) - http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_5_A
25. Comet OJ Contest #1: 热身赛 B. 简单的数学题 (简单的数学题) - https://cometoj.com/contest/1/problem/B
26. MarsCode 火星编程竞赛: 数字统计 (数字统计) - https://www.marscode.cn/contest/1/problem/1002
27. ZOJ 1002: Fire Net (消防网) - http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=1002
28. LOJ 101: 最小生成树 (最小生成树) - https://loj.ac/p/101
29. 各大高校 OJ: 北京大学 OJ 1000: A+B Problem (A+B 问题) - http://poj.org/problem?id=1000
30. Codeforces 122A. Lucky Division (幸运分割) - https://codeforces.com/problemset/problem/122/A
31. AtCoder ABC 218 C - Shapes (形状) - https://atcoder.jp/contests/abc218/tasks/abc218_c
32. USACO Bronze: Block Game (积木游戏) - http://www.usaco.org/index.php?page=viewproblem2&cpid=664
33. 洛谷 P3366 【模板】最小生成树 (模板最小生成树) - https://www.luogu.com.cn/problem/P3366
34. LeetCode 149. Max Points on a Line (直线上最多的点数) - https://leetcode.com/problems/max-points-on-a-line/
35. LeetCode 215. Kth Largest Element in an Array (数组中的第 K 个最大元素) - https://leetcode.com/problems/kth-largest-element-in-an-array/
36. LeetCode 295. Find Median from Data Stream (数据流的中位数) - https://leetcode.com/problems/find-median-from-data-stream/
37. LeetCode 315. Count of Smaller Numbers After Self (计算右侧小于当前元素的个数) - https://leetcode.com/problems/count-of-smaller-numbers-after-self/
38. LeetCode 327. Count of Range Sum (区间和的个数) - https://leetcode.com/problems/count-of-range-sum/
39. LeetCode 350. Intersection of Two Arrays II (两个数组的交集 II) - https://leetcode.com/problems/intersection-of-two-arrays-ii/
40. LeetCode 148. Sort List (排序链表) - https://leetcode.com/problems/sort-list/
41. LeetCode 242. Valid Anagram (有效的字母异位词) - https://leetcode.com/problems/valid-anagram/
42. LeetCode 347. Top K Frequent Elements (前 K 个高频元素) - https://leetcode.com/problems/top-k-frequent-elements/
43. LeetCode 451. Sort Characters By Frequency (根据字符出现频率排序) - https://leetcode.com/problems/sort-characters-by-frequency/
44. LeetCode 493. Reverse Pairs (翻转对) - https://leetcode.com/problems/reverse-pairs/

45. LeetCode 539. Minimum Time Difference (最小时间差) - https://leetcode.com/problems/minimum-time-difference/
46. LeetCode 791. Custom Sort String (自定义字符串排序) - https://leetcode.com/problems/custom-sort-string/
47. LeetCode 1030. Matrix Cells in Distance Order (距离顺序排列矩阵单元格) - https://leetcode.com/problems/matrix-cells-in-distance-order/
48. LeetCode 1122. Relative Sort Array (数组的相对排序) - https://leetcode.com/problems/relative-sort-array/
49. LeetCode 1356. Sort Integers by The Number of 1 Bits (根据数字二进制下 1 的数目排序) - https://leetcode.com/problems/sort-integers-by-the-number-of-1-bits/
50. LeetCode 179. Largest Number (最大数) - https://leetcode.com/problems/largest-number/
"""

```
class LongestWordInDictionary:
    """
    LeetCode 524. Longest Word in Dictionary through Deleting
    通过删除字母匹配到字典里最长单词
    网址：https://leetcode.com/problems/longest-word-in-dictionary-through-deleting/

    解题思路：
    1. 使用自定义 key 按长度降序、字典序升序排序
    2. 检查每个单词是否可以通过删除 s 中的字符得到
    3. 返回第一个满足条件的单词

    时间复杂度：O(n * x * log n)，其中 x 是单词平均长度
    空间复杂度：O(n)
    """

    def findLongestWord(self, s: str, dictionary: List[str]) -> str:
        # 自定义排序：先按长度降序，再按字典序升序
        dictionary.sort(key=lambda x: (-len(x), x))

        for word in dictionary:
            if self.is_subsequence(word, s):
                return word
        return ""

    def is_subsequence(self, word: str, s: str) -> bool:
        i, j = 0, 0
        while i < len(word) and j < len(s):
            if word[i] == s[j]:
                i += 1
            j += 1
```

```
        return i == len(word)

class ReorderLogFiles:
    """
    LeetCode 937. Reorder Data in Log Files
    重新排列日志文件
    网址：https://leetcode.com/problems/reorder-data-in-log-files/

    解题思路：
    1. 区分字母日志和数字日志
    2. 字母日志按内容字典序排序，内容相同按标识符排序
    3. 数字日志保持原有顺序

    时间复杂度：O(n log n)
    空间复杂度：O(n)
    """

    def reorderLogFiles(self, logs: List[str]) -> List[str]:
        def get_key(log):
            # 分割标识符和内容
            identifier, content = log.split(" ", 1)

            # 判断是否是数字日志
            if content[0].isdigit():
                return (1, )  # 数字日志排在后面
            else:
                return (0, content, identifier)  # 字母日志按内容和标识符排序

        return sorted(logs, key=get_key)

class RankTransformArray:
    """
    LeetCode 1331. Rank Transform of an Array
    数组序号转换
    网址：https://leetcode.com/problems/rank-transform-of-an-array/

    解题思路：
    1. 创建排序后的唯一值列表
    2. 使用字典存储值到排名的映射
    3. 根据映射转换原数组

    时间复杂度：O(n log n)
    空间复杂度：O(n)
```

```
    """

    def arrayRankTransform(self, arr: List[int]) -> List[int]:
        if not arr:
            return []

        # 创建排序后的唯一值列表
        sorted_unique = sorted(set(arr))

        # 创建值到排名的映射
        rank_map = {}
        for rank, num in enumerate(sorted_unique, 1):
            rank_map[num] = rank

        # 转换原数组
        return [rank_map[num] for num in arr]

class RankTeamsByVotes:
    """
    LeetCode 1366. Rank Teams by Votes
    通过投票对团队排名
    网址：https://leetcode.com/problems/rank-teams-by-votes/

    解题思路：
    1. 统计每个团队在每个位置的得票数
    2. 使用自定义 key 按得票规则排序团队
    3. 按得票规则排序团队

    时间复杂度：O(n * m + n log n)，其中 m 是投票位置数
    空间复杂度：O(n²)
    """

    def rankTeams(self, votes: List[str]) -> str:
        if not votes:
            return ""
        if len(votes) == 1:
            return votes[0]

        n = len(votes[0])
        # 统计每个团队在每个位置的得票数
        vote_count = {}
        for vote in votes:
            for i, team in enumerate(vote):
```

```
                if team not in vote_count:
                    vote_count[team] = [0] * n
                vote_count[team][i] += 1

        # 自定义排序key
        def sort_key(team):
            return (-vote_count[team][0], -vote_count[team][1],
                    -vote_count[team][2], team)  # 负号实现降序

        # 按自定义规则排序团队
        sorted_teams = sorted(vote_count.keys(), key=sort_key)

        return "".join(sorted_teams)

class RearrangeWords:
    """
    LeetCode 1451. Rearrange Words in a Sentence
    重新排列句子中的单词
    网址：https://leetcode.com/problems/rearrange-words-in-a-text/

    解题思路：
    1. 按单词长度排序
    2. 长度相同保持原有顺序（稳定排序）
    3. 首字母大写处理

    时间复杂度：O(n log n)
    空间复杂度：O(n)
    """

    def arrangeWords(self, text: str) -> str:
        if not text:
            return text

        # 分割单词并转换为小写
        words = text.lower().split()

        # 按长度排序（稳定排序）
        words.sort(key=len)

        # 首字母大写处理
        if words:
            words[0] = words[0][0].upper() + words[0][1:]
```

```
        return " ".join(words)

class MinimumDifference:
    """
    LeetCode 1509. Minimum Difference Between Largest and Smallest Value in Three Moves
    三次操作后最大值与最小值的最小差
    网址：https://leetcode.com/problems/minimum-difference-between-largest-and-smallest-value-in-three-moves/

    解题思路：
    1. 排序数组
    2. 分析移除三个元素后的可能情况
    3. 计算最小差值

    时间复杂度：O(n log n)
    空间复杂度：O(1)
    """

    def minDifference(self, nums: List[int]) -> int:
        n = len(nums)
        if n <= 4:
            return 0

        nums.sort()

        # 四种可能的情况
        min_diff = sys.maxsize
        min_diff = min(min_diff, nums[-1] - nums[3])      # 移除最小的3个
        min_diff = min(min_diff, nums[-2] - nums[2])      # 移除最小的2个和最大的1个
        min_diff = min(min_diff, nums[-3] - nums[1])      # 移除最小的1个和最大的2个
        min_diff = min(min_diff, nums[-4] - nums[0])      # 移除最大的3个

        return min_diff

class MaximumCoins:
    """
    LeetCode 1561. Maximum Number of Coins You Can Get
    你可以获得的最大硬币数目
    网址：https://leetcode.com/problems/maximum-number-of-coins-you-can-get/

    解题思路：
    1. 排序数组
    2. 每次取第二大的堆（贪心策略）
```

```
    3. 计算你能获得的总硬币数

    时间复杂度：O(n log n)
    空间复杂度：O(1)
    """

    def maxCoins(self, piles: List[int]) -> int:
        piles.sort()
        n = len(piles)
        result = 0

        # 每次取第二大的堆
        for i in range(n - 2, n // 3 - 1, -2):
            result += piles[i]

        return result

class SortByFrequency:
    """
    LeetCode 1636. Sort Array by Increasing Frequency
    按照频率将数组升序排序
    网址：https://leetcode.com/problems/sort-array-by-increasing-frequency/

    解题思路：
    1. 统计每个数字的频率
    2. 使用自定义 key 按频率升序排序
    3. 频率相同按数值降序排序

    时间复杂度：O(n log n)
    空间复杂度：O(n)
    """

    def frequencySort(self, nums: List[int]) -> List[int]:
        from collections import Counter

        # 统计频率
        freq = Counter(nums)

        # 自定义排序 key
        def sort_key(num):
            return (freq[num], -num)  # 频率升序，数值降序

        return sorted(nums, key=sort_key)
```

```
class MaximumUnits:
    """
    LeetCode 1710. Maximum Units on a Truck
    卡车上的最大单元数
    网址：https://leetcode.com/problems/maximum-units-on-a-truck/

    解题思路：
    1. 按单位单元数（每个箱子的单元数）降序排序
    2. 贪心选择单位单元数最大的箱子
    3. 计算最大单元数

    时间复杂度：O(n log n)
    空间复杂度：O(1)
    """

    def maximumUnits(self, boxTypes: List[List[int]], truckSize: int) -> int:
        # 按单位单元数降序排序
        boxTypes.sort(key=lambda x: -x[1])

        result = 0
        remaining = truckSize

        for boxes, units in boxTypes:
            if remaining <= 0:
                break

            boxes_to_take = min(remaining, boxes)
            result += boxes_to_take * units
            remaining -= boxes_to_take

        return result

class SortSentence:
    """
    LeetCode 1859. Sorting the Sentence
    将句子排序
    网址：https://leetcode.com/problems/sorting-the-sentence/

    解题思路：
    1. 提取单词末尾的数字
    2. 按数字排序单词
    3. 重新组合成句子
```

```
    时间复杂度：O(n log n)
    空间复杂度：O(n)
    """

    def sortSentence(self, s: str) -> str:
        words = s.split()

        # 自定义排序 key：按末尾数字排序
        words.sort(key=lambda word: int(word[-1]))

        # 构建结果句子，去除末尾数字
        return " ".join(word[:-1] for word in words)

# 测试代码
if __name__ == "__main__":
    # 测试 LeetCode 524
    solution = LongestWordInDictionary()
    s = "abpcplea"
    dictionary = ["ale", "apple", "monkey", "plea"]
    longest_word = solution.findLongestWord(s, dictionary)
    print("LeetCode 524 Result:", longest_word)

    # 测试 LeetCode 937
    solution = ReorderLogFiles()
    logs = ["dig1 8 1 5 1", "let1 art can", "dig2 3 6", "let2 own kit dig", "let3 art zero"]
    reordered_logs = solution.reorderLogFiles(logs)
    print("LeetCode 937 Result:", reordered_logs)

    # 测试 LeetCode 1331
    solution = RankTransformArray()
    arr = [40, 10, 20, 30]
    ranks = solution.arrayRankTransform(arr)
    print("LeetCode 1331 Result:", ranks)

    # 测试 LeetCode 1710
    solution = MaximumUnits()
    boxTypes = [[1,3], [2,2], [3,1]]
    max_units = solution.maximumUnits(boxTypes, 4)
    print("LeetCode 1710 Result:", max_units)

==========================================
```