==========
文件夹: class157_SkipList
==========

[Markdown 文件]
==========
文件: ENGINEERING_CONSIDERATIONS.md
==========
跳表算法的工程化考量与面试要点总结

目录

工程化考量

1. 异常处理与边界场景

空输入处理
- 检查输入参数的有效性
- 处理操作次数为0的边界情况
- 验证键值范围是否符合要求

极端值处理
- 处理整数最大值和最小值
- 处理大量重复数据的情况
- 处理有序和逆序数据的插入

错误恢复
- 提供清晰的错误信息
- 实现优雅的降级机制
- 确保资源的正确释放

2. 内存管理

内存分配策略
- 预分配固定大小的数组空间
- 避免频繁的动态内存分配
- 合理设置最大层数限制

内存释放
- 及时清空不再使用的数据
- 实现显式的清理接口
- 防止内存泄漏

3. 并发安全性

读写锁机制
- 读操作可以并发执行
- 写操作需要独占锁
- 实现无锁化的读操作

原子操作
- 关键变量使用原子操作
- 避免竞态条件
- 实现线程安全的随机数生成

4. 可配置性

参数调优
- 最大层数可配置
- 随机概率可调整
- 内存大小可设置

接口设计
- 提供统一的操作接口
- 支持不同的数据类型
- 实现泛型化设计

面试要点

1. 算法原理深度理解

核心思想
```
跳表通过多层链表结构实现快速查找：
- 每层都是下一层的稀疏表示
- 查找时从高层向低层逐层搜索
- 插入时通过随机函数决定节点层数
```

关键设计
- **随机层数生成**：通过抛硬币方式决定节点层数

- **路径记录**：在插入和删除时记录查找路径
- **跨度计算**：通过 span 数组快速计算排名

2. 复杂度分析详解

时间复杂度推导
```
查找操作分析：
1. 从后向前分析查找路径
2. 向上爬升的期望步数：i/p
3. 水平移动的期望步数：1/p
4. 总体期望时间复杂度：O(log n)
```

空间复杂度分析
```
空间使用分析：
1. 每个节点期望层数：1/(1-p)
2. 总节点数：n
3. 总空间复杂度：O(n)
```

3. 与平衡树的深度对比

实现复杂度
特性	跳表	平衡树
实现难度	简单	复杂
代码量	少	多
调试难度	低	高

性能对比
操作	跳表	平衡树
查找	O(log n)	O(log n)
插入	O(log n)	O(log n)
删除	O(log n)	O(log n)
范围查询	优秀	良好

并发性能
- **跳表**：天然支持无锁读操作
- **平衡树**：需要复杂的锁机制

4. 实际应用案例

Redis 有序集合
```
Redis 使用跳表实现有序集合(ZSet)：
- 当元素数量超过 128 个时使用跳表
- 当元素最大长度超过 64 字节时使用跳表
- 结合哈希表实现 O(1)的查找
```

LSM 树 memtable
```
LevelDB 中使用跳表作为 memtable：
- 内存中的写入缓冲区
- 有序存储键值对
- 刷写到磁盘时转换为 SSTable
```

调试技巧

1. 中间过程打印
```java
// 在关键步骤打印调试信息
System.out.println("当前层数: " + h + ", 当前节点: " + i + ", 查找值: " + num);
```

2. 断言验证
```python
# 验证中间结果的正确性
assert current.key < num, "查找路径错误"
```

3. 性能退化排查
- 使用计时器测量各操作耗时
- 分析热点函数的调用频率
- 检查内存使用情况

性能优化

1. 常数项优化
IO 优化
- 使用 BufferedReader 提高输入效率
- 使用 PrintWriter 提高输出效率

- 避免频繁的系统调用

计算优化
- 预计算常用值
- 减少重复计算
- 使用位运算优化

2. 数据结构优化
内存布局
- 连续内存访问提高缓存命中率
- 减少指针跳转
- 优化节点大小

算法优化
- 使用迭代替代递归减少栈开销
- 预分配内存减少动态分配
- 批量操作减少函数调用

3. 缓存优化
局部性原理
- 提高时间局部性
- 提高空间局部性
- 减少缓存未命中

实际应用场景

1. 分布式系统
分布式锁
- 使用跳表维护锁的等待队列
- 实现公平锁机制
- 支持优先级排序

分布式存储
- 维护数据分片的元信息
- 实现负载均衡
- 支持范围查询

2. 数据库系统
索引结构
- 作为B+树的替代方案
- 实现内存索引
- 支持并发访问

查询优化
- 维护查询计划的成本信息
- 实现自适应查询优化
- 支持动态统计信息

3. 网络系统
路由表
- 维护网络路由信息
- 实现最长前缀匹配
- 支持快速更新

负载均衡
- 维护服务器状态信息
- 实现加权轮询算法
- 支持动态权重调整

4. 机器学习
特征工程
- 维护特征的重要性排序
- 实现特征选择算法
- 支持在线学习

模型管理
- 维护模型版本信息
- 实现 A/B 测试
- 支持模型回滚

面试常见问题

1. 跳表与红黑树的对比
- **实现复杂度**：跳表实现更简单，红黑树需要维护复杂的旋转操作
- **并发性能**：跳表天然支持无锁读操作，红黑树需要复杂的锁机制
- **内存使用**：跳表每个节点包含的指针数目可调，红黑树每个节点固定包含 3 个指针
- **范围查询**：跳表支持高效的范围查询，红黑树需要中序遍历

2. 跳表的随机层数生成
- **概率因子**：通常设置为 0.5，使得平均层数为 2
- **层数分布**：遵循几何分布，层数越高概率越小
- **最大层数**：需要设置合理的上限避免极端情况

3. 跳表的空间复杂度分析
- **期望空间**：每个节点平均包含 1/(1-p) 个指针
- **总空间**：O(n) 总体空间复杂度

- **实际应用**：Redis 中通过限制层数和节点数来控制内存使用

4. 跳表在 Redis 中的应用
- **ZSet 实现**：Redis 使用跳表实现有序集合
- **条件选择**：当元素数量超过 128 个或元素长度超过 64 字节时使用跳表
- **组合结构**：结合哈希表实现 O(1)的查找和 O(log n)的范围查询

5. 跳表的并发优化
- **读写分离**：读操作可以无锁并发执行
- **细粒度锁**：写操作可以使用分段锁减少锁竞争
- **无锁实现**：使用 CAS 操作实现无锁的跳表

总结

跳表作为一种优秀的数据结构，在实际工程中有着广泛的应用。掌握其原理、实现和优化技巧，对于算法工程师和系统架构师都具有重要意义。在面试中能够深入分析其复杂度、对比其他数据结构、并结合实际应用场景进行讨论，将大大提升竞争力。

==========

文件: README.md
==========
跳表 (Skip List) 算法实现与题目解析

目录

算法介绍

跳表(Skip List)是一种概率型数据结构，由 William Pugh 在 1990 年提出。它通过在有序链表的基础上增加多级索引来实现快速查找，平均时间复杂度为 O(log n)。

核心思想
1. 在有序链表的基础上增加多层索引
2. 每一层都是下一层的稀疏表示
3. 查找时从高层开始，逐层向下
4. 插入时通过随机函数决定节点层数

与平衡树的对比
1. **实现简单**：跳表不需要复杂的旋转操作，代码更容易编写和维护
2. **并发友好**：跳表在并发场景下更容易实现高效的锁策略
3. **范围查询**：跳表天然支持高效的范围查询
4. **内存占用**：跳表每个节点包含的指针数目可调，通常比平衡树更节省空间
5. **时间复杂度**：跳表和平衡树的时间复杂度都是 O(log n)，但跳表是期望复杂度

题目列表

1. LeetCode 1206. 设计跳表
- **链接**：https://leetcode.cn/problems/design-skiplist
- **题目描述**：设计一个跳表，支持在 O(log(n))时间内完成增加、删除、搜索操作。
- **最优解**：使用跳表实现，时间复杂度 O(log n)

2. 洛谷 P3369 【模板】普通平衡树
- **链接**：https://www.luogu.com.cn/problem/P3369
- **题目描述**：维护一个可重集合，支持插入、删除、查询排名、查询第 k 小、查询前驱、查询后继等操作。
- **最优解**：跳表/平衡树，时间复杂度 O(log n)

3. 洛谷 P3391 【模板】文艺平衡树
- **链接**：https://www.luogu.com.cn/problem/P3391
- **题目描述**：维护一个序列，支持区间翻转操作。
- **最优解**：Splay 树，时间复杂度 O(log n)

4. HDU 1754 I Hate It
- **链接**：http://acm.hdu.edu.cn/showproblem.php?pid=1754
- **题目描述**：维护一个序列，支持单点修改和区间最大值查询。
- **最优解**：线段树，时间复杂度 O(log n)

5. POJ 3468 A Simple Problem with Integers
- **链接**：http://poj.org/problem?id=3468
- **题目描述**：维护一个序列，支持区间加和区间求和操作。
- **最优解**：线段树，时间复杂度 O(log n)

6. Codeforces 1324D - Pair of Topics
- **链接**：https://codeforces.com/problemset/problem/1324/D
- **题目描述**：给定两个数组 a 和 b，求满足 ai+aj>bi+bj 且 i<j 的数对个数。
- **最优解**：排序+二分查找，时间复杂度 O(n log n)

7. AtCoder ABC157E - Simple String Queries
- **链接**：https://atcoder.jp/contests/abc157/tasks/abc157_e

- **题目描述**：维护一个字符串，支持单点修改和区间字符计数查询。
- **最优解**：线段树或BIT，时间复杂度O(log n)

8. SPOJ DQUERY - D-query
- **链接**：https://www.spoj.com/problems/DQUERY/
- **题目描述**：给定一个数组，多次查询区间不同元素的个数。
- **最优解**：莫队算法，时间复杂度O(n√n)

9. HackerRank Array Manipulation
- **链接**：https://www.hackerrank.com/challenges/crush/problem
- **题目描述**：给定一个数组，多次对区间进行加法操作，求最终数组的最大值。
- **最优解**：差分数组，时间复杂度O(n)

10. 牛客网 Wannafly挑战赛17 A - 跳票
- **链接**：https://ac.nowcoder.com/acm/problem/14373
- **题目描述**：维护一个序列，支持插入、删除和查询第k小元素。
- **最优解**：跳表/平衡树，时间复杂度O(log n)

11. CodeChef QSET - Query on a Set
- **链接**：https://www.codechef.com/problems/QSET
- **题目描述**：维护一个集合，支持插入、删除和查询第k小元素。
- **最优解**：跳表/平衡树，时间复杂度O(log n)

12. SPOJ ORDERSET - Order statistic set
- **链接**：https://www.spoj.com/problems/ORDERSET/
- **题目描述**：维护一个有序集合，支持插入、删除、查询第k小和查询元素排名。
- **最优解**：跳表/平衡树，时间复杂度O(log n)

13. HackerEarth Monk and Champions League
- **链接**：https://www.hackerearth.com/practice/data-structures/trees/binary-search-tree/practice-problems/algorithm/monk-and-champions-league/
- **题目描述**：维护一个序列，支持插入、删除和查询最大值。
- **最优解**：跳表/堆，时间复杂度O(log n)

14. USACO 2019 January Silver - Mountain View
- **链接**：http://www.usaco.org/index.php?page=viewproblem2&cpid=895
- **题目描述**：维护一个序列，支持区间查询和更新操作。
- **最优解**：线段树/跳表，时间复杂度O(log n)

15. Project Euler #540 - Counting Primitive Pythagorean Triples
- **链接**：https://projecteuler.net/problem=540
- **题目描述**：计算满足特定条件的毕达哥拉斯三元组数量。
- **最优解**：数论+跳表优化，时间复杂度O(n log n)

16. CodeChef QSET - Query on a Set
- **链接**：https://www.codechef.com/problems/QSET
- **题目描述**：维护一个集合，支持插入、删除和查询第 k 小元素。
- **最优解**：跳表/平衡树，时间复杂度 O(log n)

17. SPOJ ORDERSET - Order statistic set
- **链接**：https://www.spoj.com/problems/ORDERSET/
- **题目描述**：维护一个有序集合，支持插入、删除、查询第 k 小和查询元素排名。
- **最优解**：跳表/平衡树，时间复杂度 O(log n)

18. HackerEarth Monk and Champions League
- **链接**：https://www.hackerearth.com/practice/data-structures/trees/binary-search-tree/practice-problems/algorithm/monk-and-champions-league/
- **题目描述**：维护一个序列，支持插入、删除和查询最大值。
- **最优解**：跳表/堆，时间复杂度 O(log n)

19. USACO 2019 January Silver - Mountain View
- **链接**：http://www.usaco.org/index.php?page=viewproblem2&cpid=895
- **题目描述**：维护一个序列，支持区间查询和更新操作。
- **最优解**：线段树/跳表，时间复杂度 O(log n)

20. 剑指 Offer 41 - 数据流中的中位数
- **链接**：https://leetcode.cn/problems/shu-ju-liu-zhong-de-zhong-wei-shu-lcof/
- **题目描述**：如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。
- **最优解**：双堆/跳表，时间复杂度 O(log n)

21. LeetCode 295 - 数据流的中位数
- **链接**：https://leetcode.cn/problems/find-median-from-data-stream/
- **题目描述**：设计一个支持以下两种操作的数据结构：void addNum(int num) - 从数据流中添加一个整数到数据结构中。double findMedian() - 返回目前所有元素的中位数。
- **最优解**：双堆/跳表，时间复杂度 O(log n)

22. LeetCode 480 - 滑动窗口中位数
- **链接**：https://leetcode.cn/problems/sliding-window-median/
- **题目描述**：中位数是有序序列最中间的那个数。如果序列的大小是偶数，则没有最中间的数；此时中位数是最中间的两个数的平均数。
- **最优解**：双堆/跳表，时间复杂度 O(n log k)

23. LeetCode 703 - 数据流中的第 K 大元素
- **链接**：https://leetcode.cn/problems/kth-largest-element-in-a-stream/

- **题目描述**：设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个不同的元素。
- **最优解**：堆/跳表，时间复杂度 O(log k)

24. LeetCode 220 - 存在重复元素 III
- **链接**：https://leetcode.cn/problems/contains-duplicate-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 。
- **最优解**：滑动窗口+有序集合，时间复杂度 O(n log k)

25. LeetCode 352 - 将数据流变为多个不相交区间
- **链接**：https://leetcode.cn/problems/data-stream-as-disjoint-intervals/
- **题目描述**：给定一个非负整数的数据流输入 a1，a2，…，an，…，将到目前为止看到的数字总结为不相交的区间列表。
- **最优解**：有序集合，时间复杂度 O(log n)

26. LeetCode 855 - 考场就座
- **链接**：https://leetcode.cn/problems/exam-room/
- **题目描述**：在考场里，有 N 个座位，分别编号为 0, 1, 2, ..., N-1 。当学生进入考场后，他必须坐在能够使他与离他最近的人之间的距离达到最大化的座位上。如果有多个这样的座位，他会坐在编号最小的座位上。
- **最优解**：有序集合，时间复杂度 O(log n)

27. LeetCode 981 - 基于时间的键值存储
- **链接**：https://leetcode.cn/problems/time-based-key-value-store/
- **题目描述**：设计一个基于时间的键值数据结构，该结构可以在不同时间戳存储对应同一个键的多个值，并检索特定时间戳的值。
- **最优解**：哈希表+有序集合，时间复杂度 O(log n)

28. LeetCode 1146 - 快照数组
- **链接**：https://leetcode.cn/problems/snapshot-array/
- **题目描述**：实现支持下列接口的「快照数组」- SnapshotArray：SnapshotArray(int length) - 初始化一个与指定长度相等的 类数组 的数据结构。初始时，每个元素都等于 0。void set(index, val) - 会将指定索引 index 处的元素设置为 val。int snap() - 获取该数组的快照，并返回快照的 id snap_id（快照号是调用 snap() 的总次数减去 1）。int get(index, snap_id) - 根据指定的 snap_id 选择快照，并返回该快照指定索引 index 的值。
- **最优解**：数组+有序集合，时间复杂度 O(log n)

29. LeetCode 1348 - 推文计数
- **链接**：https://leetcode.cn/problems/tweet-counts-per-frequency/
- **题目描述**：请你实现一个能够支持以下两种方法的推文计数类 TweetCounts：recordTweet(string tweetName, int time) - 记录推文发布情况：用户 tweetName 在 time（以 秒 为单位）时刻发布了一条推文。getTweetCountsPerFrequency(string freq, string tweetName, int startTime, int endTime) - 返回

从开始时间 startTime（以 秒 为单位）到结束时间 endTime（以 秒 为单位）内，每 分 minute，每 时 hour 或者 每日 day（取决于 freq）内指定用户 tweetName 发布的推文总数。
- **最优解**：哈希表+有序集合，时间复杂度 O(log n)

30. LeetCode 1396 - 设计地铁系统
- **链接**：https://leetcode.cn/problems/design-underground-system/
- **题目描述**：请你实现一个类 UndergroundSystem ，它支持以下 3 种方法：checkIn(int id, string stationName, int t) - 乘客 id 在时间 t 进入 stationName 站。checkOut(int id, string stationName, int t) - 乘客 id 在时间 t 离开 stationName 站。getAverageTime(string startStation, string endStation) - 返回从 startStation 站到 endStation 站的平均时间。
- **最优解**：哈希表+有序集合，时间复杂度 O(1)

31. LeetCode 1606 - 找到处理最多请求的服务器
- **链接**：https://leetcode.cn/problems/find-servers-that-handled-most-number-of-requests/
- **题目描述**：你有 k 个服务器，编号为 0 到 k-1 ，它们可以同时处理多个请求组。每个服务器有无穷的计算能力但是 不能同时处理超过一个请求 。请求分配到服务器的规则如下：第 i （序号从 0 开始）个请求到达。如果所有服务器都已被占据，那么该请求被舍弃（完全不处理）。如果第 (i % k) 个服务器空闲，那么对应服务器会处理该请求。否则，将请求安排给下一个空闲的服务器（服务器构成一个环，必要的话可能从第 0 个服务器开始继续找下一个空闲的服务器）。比方说，如果第 i 个请求到达时，第 (i % k) 个服务器被占，那么会查看第 (i+1) % k 个服务器，第 (i+2) % k 个服务器等等。给你一个 严格递增 的正整数数组 arrival ，表示第 i 个任务的到达时间，和另一个数组 load ，其中 load[i] 表示第 i 个请求的工作量（也就是完成该请求需要花费的时间）。你的任务是找到 最繁忙的服务器 。最繁忙的服务器是指处理请求数最多的服务器。请你返回包含所有 最繁忙的服务器 序号的列表，你可以以任何顺序返回这个列表。
- **最优解**：有序集合+优先队列，时间复杂度 O(n log k)

32. LeetCode 1825 - 求出 MK 平均值
- **链接**：https://leetcode.cn/problems/finding-mk-average/
- **题目描述**：给你两个整数 m 和 k ，以及数据流形式的若干整数。你需要实现一个数据结构，计算这个数据流的 MK 平均值 。MK 平均值 按照如下步骤计算：如果数据流中的整数少于 m 个，MK 平均值 为 -1 ，否则将数据流中的最后 m 个元素拷贝到一个独立的容器中。从这个容器中删除最小的 k 个数和最大的 k 个数。计算剩余元素的平均值，并 向下取整到最近的整数 。请你实现 MKAverage 类：MKAverage(int m, int k) 用一个空的数据流和两个整数 m 和 k 初始化 MKAverage 对象。void addElement(int num) 往数据流中插入一个新的元素 num 。int calculateMKAverage() 对当前的数据流计算并返回 MK 平均数 ，结果需 取整到最近的整数 。
- **最优解**：三个有序集合，时间复杂度 O(log m)

33. LeetCode 2034 - 股票价格波动
- **链接**：https://leetcode.cn/problems/stock-price-fluctuation/
- **题目描述**：给你一支股票价格的数据流。数据流中每一条记录包含一个 时间戳 和该时间点股票对应的价格 。不巧的是，由于股票市场内在的波动性，股票价格记录可能不是按时间顺序到来的。某些情况下，有的记录可能是错的，即时间戳和价格可能都不对。请你实现一个类：StockPrice() 初始化对象，当前无股票价格记录。void update(int timestamp, int price) 在时间点 timestamp 更新股票价格为 price 。int current() 返回股票 最新价格 。int maximum() 返回股票 最高价格 。int minimum() 返回股票 最低价

格 。
- **最优解**：哈希表+有序集合，时间复杂度 O(log n)

34. LeetCode 2102 - 序列顺序查询
- **链接**：https://leetcode.cn/problems/sequentially-ordinal-rank-tracker/
- **题目描述**：一个观光景点由它的名字 name 和评分 score 组成，其中 name 是所有观光景点中 唯一 的字符串，score 是一个整数。景点按照以下规则进行排序：评分 越高 ，景点的排名越高。如果两个景点的评分相同，那么 字典序较小 的景点排名更高。请你实现一个类 SORTracker ：SORTracker() 初始化系统。void add(string name, int score) 添加一个名为 name 评分为 score 的景点。string get() 返回第 i 次调用时排名第 i 的景点，其中 i 是系统当前位置的下标（从 1 开始）。
- **最优解**：有序集合，时间复杂度 O(log n)

35. LeetCode 2349 - 设计数字容器系统
- **链接**：https://leetcode.cn/problems/design-a-number-container-system/
- **题目描述**：设计一个数字容器系统，可以实现以下功能：void change(int index, int number) - 将下标为 index 处的数字改为 number 。如果该下标 index 处已经有数字，那么原来的数字会被替换。int find(int number) - 返回给定数字 number 所在的下标中的最小下标。如果系统中不存在数字 number ，返回 -1 。
- **最优解**：哈希表+有序集合，时间复杂度 O(log n)

36. LeetCode 2424 - 最长上传前缀
- **链接**：https://leetcode.cn/problems/longest-uploaded-prefix/
- **题目描述**：给你一个整数 n 和一个下标从 1 开始的二进制数组（只包含 0 和 1 的数组）queries 。一开始，所有元素都是 0 。你需要处理 queries 中的两种类型的操作：queries[i] == 1：将下标为 queries[i+1] 的元素设为 1 。queries[i] == 2：返回由 1 组成的 最长 上传前缀的长度。
- **最优解**：并查集/有序集合，时间复杂度 O(log n)

37. LeetCode 2528 - 最大化城市的最小供电站数目
- **链接**：https://leetcode.cn/problems/maximize-the-minimum-powered-city/
- **题目描述**：给你一个下标从 0 开始长度为 n 的整数数组 stations ，其中 stations[i] 表示第 i 座城市的供电站数目。每个供电站可以为 一座 城市供电。一座城市如果被 至少一个 供电站覆盖，我们称它被覆盖 。你需要额外建造 k 座供电站。你可以选择建在任何城市。请你返回额外建造 k 座供电站后，被覆盖城市的最小供电站数目的 最大值 。
- **最优解**：二分+贪心，时间复杂度 O(n log n)

38. LeetCode 2532 - 过桥的时间
- **链接**：https://leetcode.cn/problems/time-to-cross-a-bridge/
- **题目描述**：共有 k 位工人计划将 n 个箱子从旧仓库移动到新仓库。给你两个整数 n 和 k，以及一个二维整数数组 time ，数组的长度为 k，其中 time[i] = [leftToRighti, pickOldi, rightToLefti, putNewi] 。一条河将仓库分成了旧仓库和新仓库，所有工人一开始都在旧仓库。请你返回最后一个箱子到达新仓库的时刻。
- **最优解**：优先队列，时间复杂度 O(n log k)

39. LeetCode 2560 - 打家劫舍 IV
- **链接**：https://leetcode.cn/problems/house-robber-iv/
- **题目描述**：沿街有一排连续的房屋。每间房屋内都藏有一定的现金。现在，你打算偷窃这些房屋。但是，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。给定一个代表每个房屋存放金额的非负整数数组，计算你 在不触动警报装置的情况下 ，能够偷窃到的最高金额。
- **最优解**：动态规划，时间复杂度O(n)

40. LeetCode 2610 - 转换二维数组
- **链接**：https://leetcode.cn/problems/convert-an-array-into-a-2d-array-with-conditions/
- **题目描述**：给你一个整数数组 nums 。请你返回一个二维数组，该数组需满足：数组中的每个元素都是互不相同 的。数组中的每一行必须包含 nums 中所有不同的元素。数组中的行数应尽可能少。返回结果数组。如果存在多种答案，只需返回其中任意一种。
- **最优解**：哈希表，时间复杂度O(n)

41. LeetCode 2641 - 二叉树的堂兄弟节点 II
- **链接**：https://leetcode.cn/problems/cousins-in-binary-tree-ii/
- **题目描述**：给你一棵二叉树的根节点 root ，请你将每个节点的值替换成该节点的所有 堂兄弟节点值的和 。堂兄弟节点指深度相同但父节点不同的节点。
- **最优解**：BFS，时间复杂度O(n)

42. LeetCode 2653 - 滑动子数组的美丽值
- **链接**：https://leetcode.cn/problems/sliding-subarray-beauty/
- **题目描述**：给你一个长度为 n 的整数数组 nums ，请你求出每个长度为 k 的子数组的 美丽值 。一个子数组的美丽值定义为：子数组中第 x 小的数，如果它是负数，那么美丽值就是这个数；否则，美丽值为 0 。
- **最优解**：滑动窗口+有序集合，时间复杂度O(n log k)

43. LeetCode 2696 - 删除子串后的字符串最小长度
- **链接**：https://leetcode.cn/problems/minimum-string-length-after-removing-substrings/
- **题目描述**：给你一个字符串 s ，它仅由大写英文字母组成。你可以对这个字符串执行一些操作，每次操作可以删除 s 中的一个子串 "AB" 或 "CD" 。请你返回使字符串 s 变为 空字符串 需要删除的最少操作次数。
- **最优解**：栈，时间复杂度O(n)

44. LeetCode 2736 - 最大和查询
- **链接**：https://leetcode.cn/problems/maximum-sum-queries/
- **题目描述**：给你两个长度为 n 的整数数组 nums1 和 nums2 ，以及一个长度为 m 的整数数组 queries 。对于每个查询 queries[i] = [xi, yi] ，你需要找到满足 nums1[j] >= xi 且 nums2[j] >= yi 的下标 j (0 <= j < n) ，并返回 nums1[j] + nums2[j] 的 最大值 。如果不存在满足条件的 j ，则返回 -1 。
- **最优解**：离线处理+有序集合，时间复杂度O((n+m) log n)

45. LeetCode 2818 - 操作使得分最大
- **链接**：https://leetcode.cn/problems/apply-operations-to-maximize-score/
- **题目描述**：给你一个下标从 0 开始的整数数组 nums 和一个整数 k 。你的 起始分数 为 0 。在一步操作 中，你可以：选择一个下标 i 满足 0 <= i < nums.length 。将 nums[i] 替换为 floor(nums[i] / 2) 。将你的分数增加 nums[i] 。不过，你最多只能执行 k 次操作。请你返回你能得到的 最大分数 。
- **最优解**：贪心+优先队列，时间复杂度O(n log n)

46. LeetCode 2846 - 边权重均等查询
- **链接**：https://leetcode.cn/problems/minimum-edge-weight-equilibrium-queries-in-a-tree/
- **题目描述**：现有一棵由 n 个节点组成的无向树，节点按从 0 到 n - 1 编号。给你一个整数 n 和一个二维整数数组 edges ，其中 edges[i] = [ui, vi, wi] 表示节点 ui 和 vi 之间有一条边权为 wi 的边。另给你一个二维整数数组 queries ，其中 queries[j] = [aj, bj] 。对于每个查询，你需要找出从 aj 到 bj 的路径上 边权重出现次数最大值 的最小可能值。
- **最优解**：LCA+树上差分，时间复杂度O(n log n)

47. LeetCode 2861 - 最大合金数
- **链接**：https://leetcode.cn/problems/maximum-number-of-alloys/
- **题目描述**：假设你是一家合金制造公司的老板，你的公司使用多种金属来制造合金。现在你有 n 台机器，每台机器都需要消耗一定数量的每种金属。给定一个整数 budget 表示你的预算，和一个 2D 整数数组 composition ，其中 composition[i] 是一个长度为 m 的数组，表示第 i 台机器制造一单位合金需要消耗的各种金属的数量。另外给你一个长度为 m 的整数数组 stock ，表示你目前拥有的各种金属的库存量，和一个长度为 m 的整数数组 cost ，表示购买一单位各种金属需要的费用。请你计算在预算范围内，通过任意一台机器最多可以制造多少单位的合金。
- **最优解**：二分查找，时间复杂度O(n log k)

48. LeetCode 2872 - 最大子序列交替和
- **链接**：https://leetcode.cn/problems/maximum-subsequence-alternating-sum/
- **题目描述**：给你一个下标从 0 开始的整数数组 nums 。一个子序列的 交替和 定义为：子序列中偶数下标元素和 减去 奇数下标元素和。比方说，[4,2,5,3] 的交替和为 (4 + 5) - (2 + 3) = 4 。请你返回 nums 中任意子序列的 最大交替和 。
- **最优解**：动态规划，时间复杂度O(n)

49. LeetCode 2897 - 执行操作使两个字符串相等
- **链接**：https://leetcode.cn/problems/apply-operations-on-array-to-maximize-sum-of-squares/
- **题目描述**：给你两个下标从 0 开始的二进制字符串 s 和 target ，两个字符串的长度均为 n 。你可以对 s 执行以下操作 任意 次：选择两个 不同 的下标 i 和 j ，其中 0 <= i, j < n 。同时，将 s[i] 替换为 (s[i] OR s[j]) ，将 s[j] 替换为 (s[i] XOR s[j]) 。请你返回使 s 转化成为 target 需要的 最少 操作次数。如果不可能完成转化，请你返回 -1 。
- **最优解**：位运算，时间复杂度O(n)

50. LeetCode 2926 - 平衡子序列的最大和
- **链接**：https://leetcode.cn/problems/maximum-balanced-subsequence-sum/
- **题目描述**：给你一个下标从 0 开始的整数数组 nums 。nums 的一个子序列如果满足以下条件，那么它

是 平衡的 ：对于子序列中每两个 相邻 元素，它们的绝对差最多为 1 。也就是说，对于子序列中每两个相邻的值 nums[i] 和 nums[j] ，有 |nums[i] - nums[j]| <= 1 。请你返回 nums 中 平衡 子序列的 最大 和 。
- **最优解**：动态规划+有序集合，时间复杂度O(n log n)

51. LeetCode 295 - 数据流的中位数
- **链接**：https://leetcode.cn/problems/find-median-from-data-stream/
- **题目描述**：设计一个支持以下两种操作的数据结构：void addNum(int num) - 从数据流中添加一个整数到数据结构中。double findMedian() - 返回目前所有元素的中位数。
- **最优解**：双堆/跳表，时间复杂度O(log n)

52. LeetCode 480 - 滑动窗口中位数
- **链接**：https://leetcode.cn/problems/sliding-window-median/
- **题目描述**：中位数是有序序列最中间的那个数。如果序列的大小是偶数，则没有最中间的数；此时中位数是最中间的两个数的平均数。
- **最优解**：双堆/跳表，时间复杂度O(n log k)

53. LeetCode 703 - 数据流中的第K大元素
- **链接**：https://leetcode.cn/problems/kth-largest-element-in-a-stream/
- **题目描述**：设计一个找到数据流中第k大元素的类（class）。注意是排序后的第k大元素，不是第k个不同的元素。
- **最优解**：堆/跳表，时间复杂度O(log k)

54. LeetCode 220 - 存在重复元素 III
- **链接**：https://leetcode.cn/problems/contains-duplicate-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 。
- **最优解**：滑动窗口+有序集合，时间复杂度O(n log k)

55. LeetCode 352 - 将数据流变为多个不相交区间
- **链接**：https://leetcode.cn/problems/data-stream-as-disjoint-intervals/
- **题目描述**：给定一个非负整数的数据流输入 a1，a2，…，an，…，将到目前为止看到的数字总结为不相交的区间列表。
- **最优解**：有序集合，时间复杂度O(log n)

56. LeetCode 855 - 考场就座
- **链接**：https://leetcode.cn/problems/exam-room/
- **题目描述**：在考场里，有 N 个座位，分别编号为 0, 1, 2, ..., N-1 。当学生进入考场后，他必须坐在能够使他与离他最近的人之间的距离达到最大化的座位上。如果有多个这样的座位，他会坐在编号最小的座位上。
- **最优解**：有序集合，时间复杂度O(log n)

57. LeetCode 981 - 基于时间的键值存储
- **链接**：https://leetcode.cn/problems/time-based-key-value-store/

- **题目描述**：设计一个基于时间的键值数据结构，该结构可以在不同时间戳存储对应同一个键的多个值，并检索特定时间戳的值。
- **最优解**：哈希表+有序集合，时间复杂度 O(log n)

58. LeetCode 1146 - 快照数组
- **链接**：https://leetcode.cn/problems/snapshot-array/
- **题目描述**：实现支持下列接口的「快照数组」- SnapshotArray：SnapshotArray(int length) - 初始化一个与指定长度相等的 类数组 的数据结构。初始时，每个元素都等于 0。void set(index, val) - 会将指定索引 index 处的元素设置为 val。int snap() - 获取该数组的快照，并返回快照的 id snap_id（快照号是调用 snap() 的总次数减去 1）。int get(index, snap_id) - 根据指定的 snap_id 选择快照，并返回该快照指定索引 index 的值。
- **最优解**：数组+有序集合，时间复杂度 O(log n)

59. LeetCode 1348 - 推文计数
- **链接**：https://leetcode.cn/problems/tweet-counts-per-frequency/
- **题目描述**：请你实现一个能够支持以下两种方法的推文计数类 TweetCounts：recordTweet(string tweetName, int time) - 记录推文发布情况：用户 tweetName 在 time（以 秒 为单位）时刻发布了一条推文。getTweetCountsPerFrequency(string freq, string tweetName, int startTime, int endTime) - 返回从开始时间 startTime（以 秒 为单位）到结束时间 endTime（以 秒 为单位）内，每 分 minute，每 时 hour 或者 每日 day（取决于 freq）内指定用户 tweetName 发布的推文总数。
- **最优解**：哈希表+有序集合，时间复杂度 O(log n)

60. LeetCode 1396 - 设计地铁系统
- **链接**：https://leetcode.cn/problems/design-underground-system/
- **题目描述**：请你实现一个类 UndergroundSystem ，它支持以下 3 种方法：checkIn(int id, string stationName, int t) - 乘客 id 在时间 t 进入 stationName 站。checkOut(int id, string stationName, int t) - 乘客 id 在时间 t 离开 stationName 站。getAverageTime(string startStation, string endStation) - 返回从 startStation 站到 endStation 站的平均时间。
- **最优解**：哈希表+有序集合，时间复杂度 O(1)

61. LeetCode 1606 - 找到处理最多请求的服务器
- **链接**：https://leetcode.cn/problems/find-servers-that-handled-most-number-of-requests/
- **题目描述**：你有 k 个服务器，编号为 0 到 k-1 ，它们可以同时处理多个请求组。每个服务器有无穷的计算能力但是 不能同时处理超过一个请求 。请求分配到服务器的规则如下：第 i （序号从 0 开始）个请求到达。如果所有服务器都已被占据，那么该请求被舍弃（完全不处理）。如果第 (i % k) 个服务器空闲，那么对应服务器会处理该请求。否则，将请求安排给下一个空闲的服务器（服务器构成一个环，必要的话可能从第 0 个服务器开始继续找下一个空闲的服务器）。比方说，如果第 i 个请求到达时，第 (i % k) 个服务器被占，那么会查看第 (i+1) % k 个服务器，第 (i+2) % k 个服务器等等。给你一个 严格递增 的正整数数组 arrival ，表示第 i 个任务的到达时间，和另一个数组 load ，其中 load[i] 表示第 i 个请求的工作量（也就是完成该请求需要花费的时间）。你的任务是找到 最繁忙的服务器 。最繁忙的服务器是指处理请求数最多的服务器。请你返回包含所有 最繁忙的服务器 序号的列表，你可以以任何顺序返回这个列表。
- **最优解**：有序集合+优先队列，时间复杂度 O(n log k)

62. LeetCode 1825 - 求出 MK 平均值
- **链接**：https://leetcode.cn/problems/finding-mk-average/
- **题目描述**：给你两个整数 m 和 k ，以及数据流形式的若干整数。你需要实现一个数据结构，计算这个数据流的 MK 平均值 。MK 平均值 按照如下步骤计算：如果数据流中的整数少于 m 个，MK 平均值 为 -1 ，否则将数据流中的最后 m 个元素拷贝到一个独立的容器中。从这个容器中删除最小的 k 个数和最大的 k 个数。计算剩余元素的平均值，并 向下取整到最近的整数 。请你实现 MKAverage 类：MKAverage(int m, int k) 用一个空的数据流和两个整数 m 和 k 初始化 MKAverage 对象。void addElement(int num) 往数据流中插入一个新的元素 num 。int calculateMKAverage() 对当前的数据流计算并返回 MK 平均数 ，结果需 取整到最近的整数 。
- **最优解**：三个有序集合，时间复杂度 O(log m)

63. LeetCode 2034 - 股票价格波动
- **链接**：https://leetcode.cn/problems/stock-price-fluctuation/
- **题目描述**：给你一支股票价格的数据流。数据流中每一条记录包含一个 时间戳 和该时间点股票对应的价格 。不巧的是，由于股票市场内在的波动性，股票价格记录可能不是按时间顺序到来的。某些情况下，有的记录可能是错的，即时间戳和价格可能都不对。请你实现一个类：StockPrice() 初始化对象，当前无股票价格记录。void update(int timestamp, int price) 在时间点 timestamp 更新股票价格为 price 。int current() 返回股票 最新价格 。int maximum() 返回股票 最高价格 。int minimum() 返回股票 最低价格 。
- **最优解**：哈希表+有序集合，时间复杂度 O(log n)

64. LeetCode 2102 - 序列顺序查询
- **链接**：https://leetcode.cn/problems/sequentially-ordinal-rank-tracker/
- **题目描述**：一个观光景点由它的名字 name 和评分 score 组成，其中 name 是所有观光景点中 唯一 的字符串，score 是一个整数。景点按照以下规则进行排序：评分 越高 ，景点的排名越高。如果两个景点的评分相同，那么 字典序较小 的景点排名更高。请你实现一个类 SORTracker ：SORTracker() 初始化系统。void add(string name, int score) 添加一个名为 name 评分为 score 的景点。string get() 返回第 i 次调用时排名第 i 的景点，其中 i 是系统当前位置的下标（从 1 开始）。
- **最优解**：有序集合，时间复杂度 O(log n)

65. LeetCode 2349 - 设计数字容器系统
- **链接**：https://leetcode.cn/problems/design-a-number-container-system/
- **题目描述**：设计一个数字容器系统，可以实现以下功能：void change(int index, int number) - 将下标为 index 处的数字改为 number 。如果该下标 index 处已经有数字，那么原来的数字会被替换。int find(int number) - 返回给定数字 number 所在的下标中的最小下标。如果系统中不存在数字 number ，返回 -1 。
- **最优解**：哈希表+有序集合，时间复杂度 O(log n)

66. LeetCode 2424 - 最长上传前缀
- **链接**：https://leetcode.cn/problems/longest-uploaded-prefix/
- **题目描述**：给你一个整数 n 和一个下标从 1 开始的二进制数组（只包含 0 和 1 的数组）queries 。一开始，所有元素都是 0 。你需要处理 queries 中的两种类型的操作：queries[i] == 1：将下标为 queries[i+1] 的元素设为 1 。queries[i] == 2：返回由 1 组成的 最长 上传前缀的长度。

- **最优解**：并查集/有序集合，时间复杂度 O(log n)

67. LeetCode 2528 - 最大化城市的最小供电站数目
- **链接**：https://leetcode.cn/problems/maximize-the-minimum-powered-city/
- **题目描述**：给你一个下标从 0 开始长度为 n 的整数数组 stations ，其中 stations[i] 表示第 i 座城市的供电站数目。每个供电站可以为 一座 城市供电。一座城市如果被 至少一个 供电站覆盖，我们称它被覆盖 。你需要额外建造 k 座供电站。你可以选择建在任何城市。请你返回额外建造 k 座供电站后，被覆盖城市的最小供电站数目的 最大值 。
- **最优解**：二分+贪心，时间复杂度 O(n log n)

68. LeetCode 2532 - 过桥的时间
- **链接**：https://leetcode.cn/problems/time-to-cross-a-bridge/
- **题目描述**：共有 k 位工人计划将 n 个箱子从旧仓库移动到新仓库。给你两个整数 n 和 k，以及一个二维整数数组 time ，数组的长度为 k，其中 time[i] = [leftToRighti, pickOldi, rightToLefti, putNewi] 。一条河将仓库分成了旧仓库和新仓库，所有工人一开始都在旧仓库。请你返回最后一个箱子到达新仓库的时刻。
- **最优解**：优先队列，时间复杂度 O(n log k)

69. LeetCode 2560 - 打家劫舍 IV
- **链接**：https://leetcode.cn/problems/house-robber-iv/
- **题目描述**：沿街有一排连续的房屋。每间房屋内都藏有一定的现金。现在，你打算偷窃这些房屋。但是，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。给定一个代表每个房屋存放金额的非负整数数组，计算你 在不触动警报装置的情况下 ，能够偷窃到的最高金额。
- **最优解**：动态规划，时间复杂度 O(n)

70. LeetCode 2610 - 转换二维数组
- **链接**：https://leetcode.cn/problems/convert-an-array-into-a-2d-array-with-conditions/
- **题目描述**：给你一个整数数组 nums 。请你返回一个二维数组，该数组需满足：数组中的每个元素都是互不相同 的。数组中的每一行必须包含 nums 中所有不同的元素。数组中的行数应尽可能少。返回结果数组。如果存在多种答案，只需返回其中任意一种。
- **最优解**：哈希表，时间复杂度 O(n)

71. LeetCode 2641 - 二叉树的堂兄弟节点 II
- **链接**：https://leetcode.cn/problems/cousins-in-binary-tree-ii/
- **题目描述**：给你一棵二叉树的根节点 root ，请你将每个节点的值替换成该节点的所有 堂兄弟节点值的和 。堂兄弟节点指深度相同但父节点不同的节点。
- **最优解**：BFS，时间复杂度 O(n)

72. LeetCode 2653 - 滑动子数组的美丽值
- **链接**：https://leetcode.cn/problems/sliding-subarray-beauty/
- **题目描述**：给你一个长度为 n 的整数数组 nums ，请你求出每个长度为 k 的子数组的 美丽值 。一个子数组的美丽值定义为：子数组中第 x 小的数，如果它是负数，那么美丽值就是这个数；否则，美丽值为

0 。
- **最优解**：滑动窗口+有序集合，时间复杂度 O(n log k)

73. LeetCode 2696 - 删除子串后的字符串最小长度
- **链接**：https://leetcode.cn/problems/minimum-string-length-after-removing-substrings/
- **题目描述**：给你一个字符串 s ，它仅由大写英文字母组成。你可以对这个字符串执行一些操作，每次操作可以删除 s 中的一个子串 "AB" 或 "CD" 。请你返回使字符串 s 变为 空字符串 需要删除的最少操作次数。
- **最优解**：栈，时间复杂度 O(n)

74. LeetCode 2736 - 最大和查询
- **链接**：https://leetcode.cn/problems/maximum-sum-queries/
- **题目描述**：给你两个长度为 n 的整数数组 nums1 和 nums2 ，以及一个长度为 m 的整数数组 queries 。对于每个查询 queries[i] = [xi, yi] ，你需要找到满足 nums1[j] >= xi 且 nums2[j] >= yi 的下标 j (0 <= j < n) ，并返回 nums1[j] + nums2[j] 的 最大值 。如果不存在满足条件的 j ，则返回 -1 。
- **最优解**：离线处理+有序集合，时间复杂度 O((n+m) log n)

75. LeetCode 2818 - 操作使得分最大
- **链接**：https://leetcode.cn/problems/apply-operations-to-maximize-score/
- **题目描述**：给你一个下标从 0 开始的整数数组 nums 和一个整数 k 。你的 起始分数 为 0 。在一步操作 中，你可以：选择一个下标 i 满足 0 <= i < nums.length 。将 nums[i] 替换为 floor(nums[i] / 2) 。将你的分数增加 nums[i] 。不过，你最多只能执行 k 次操作。请你返回你能得到的 最大分数 。
- **最优解**：贪心+优先队列，时间复杂度 O(n log n)

76. LeetCode 2846 - 边权重均等查询
- **链接**：https://leetcode.cn/problems/minimum-edge-weight-equilibrium-queries-in-a-tree/
- **题目描述**：现有一棵由 n 个节点组成的无向树，节点按从 0 到 n - 1 编号。给你一个整数 n 和一个二维整数数组 edges ，其中 edges[i] = [ui, vi, wi] 表示节点 ui 和 vi 之间有一条边权为 wi 的边。另给你一个二维整数数组 queries ，其中 queries[j] = [aj, bj] 。对于每个查询，你需要找出从 aj 到 bj 的路径上 边权重出现次数最大值 的最小可能值。
- **最优解**：LCA+树上差分，时间复杂度 O(n log n)

77. LeetCode 2861 - 最大合金数
- **链接**：https://leetcode.cn/problems/maximum-number-of-alloys/
- **题目描述**：假设你是一家合金制造公司的老板，你的公司使用多种金属来制造合金。现在你有 n 台机器，每台机器都需要消耗一定数量的每种金属。给定一个整数 budget 表示你的预算，和一个 2D 整数数组 composition ，其中 composition[i] 是一个长度为 m 的数组，表示第 i 台机器制造一单位合金需要消耗的各种金属的数量。另外给你一个长度为 m 的整数数组 stock ，表示你目前拥有的各种金属的库存量，和一个长度为 m 的整数数组 cost ，表示购买一单位各种金属需要的费用。请你计算在预算范围内，通过任意一台机器最多可以制造多少单位的合金。
- **最优解**：二分查找，时间复杂度 O(n log k)

78. LeetCode 2872 - 最大子序列交替和
- **链接**：https://leetcode.cn/problems/maximum-subsequence-alternating-sum/
- **题目描述**：给你一个下标从 0 开始的整数数组 nums 。一个子序列的 交替和 定义为：子序列中偶数下标元素和 减去 奇数下标元素和。比方说，[4,2,5,3] 的交替和为 (4 + 5) - (2 + 3) = 4 。请你返回 nums 中任意子序列的 最大交替和 。
- **最优解**：动态规划，时间复杂度O(n)

79. LeetCode 2897 - 执行操作使两个字符串相等
- **链接**：https://leetcode.cn/problems/apply-operations-on-array-to-maximize-sum-of-squares/
- **题目描述**：给你两个下标从 0 开始的二进制字符串 s 和 target ，两个字符串的长度均为 n 。你可以对 s 执行以下操作 任意 次：选择两个 不同 的下标 i 和 j ，其中 0 <= i, j < n 。同时，将 s[i] 替换为 (s[i] OR s[j]) ，将 s[j] 替换为 (s[i] XOR s[j]) 。请你返回使 s 转化成为 target 需要的 最少 操作次数。如果不可能完成转化，请你返回 -1 。
- **最优解**：位运算，时间复杂度O(n)

80. LeetCode 2926 - 平衡子序列的最大和
- **链接**：https://leetcode.cn/problems/maximum-balanced-subsequence-sum/
- **题目描述**：给你一个下标从 0 开始的整数数组 nums 。nums 的一个子序列如果满足以下条件，那么它是 平衡的 ：对于子序列中每两个 相邻 元素，它们的绝对差最多为 1 。也就是说，对于子序列中每两个相邻的值 nums[i] 和 nums[j] ，有 |nums[i] - nums[j]| <= 1 。请你返回 nums 中 平衡 子序列的 最大 和 。
- **最优解**：动态规划+有序集合，时间复杂度O(n log n)

解题思路

跳表通过多层链表结构实现快速查找、插入和删除操作：

1. **节点结构**：每个节点包含key值、计数、层级和指向下一节点的指针数组
2. **随机层数**：通过抛硬币的方式决定新节点的层数
3. **查找操作**：从最高层开始，逐层向下查找
4. **插入操作**：先查找插入位置，再随机生成层数，最后插入节点
5. **删除操作**：先查找节点，再减少计数或删除节点

复杂度分析

时间复杂度
- **查找**：O(log n) 期望时间复杂度
- **插入**：O(log n) 期望时间复杂度
- **删除**：O(log n) 期望时间复杂度

空间复杂度
- **总体**：O(n)
- **每个节点**：平均包含1/(1-p)个指针

三种语言实现

Java 版本
- 文件：SkipList1.java
- 特点：使用静态数组实现，效率高

C++版本
- 文件：SkipList.cpp
- 特点：使用全局数组实现，效率高

Python 版本
- 文件：SkipList.py
- 特点：使用面向对象实现，代码清晰

工程化考量

1. 异常处理
- 输入验证：检查操作类型和参数范围
- 边界处理：处理空输入、极端值等场景

2. 性能优化
- IO 优化：使用高效的输入输出方式
- 内存管理：合理分配和释放内存空间

3. 可维护性
- 代码注释：详细注释每个函数和关键步骤
- 命名规范：使用清晰的变量和函数命名

4. 可扩展性
- 模块化设计：将功能拆分为独立的函数
- 接口设计：提供清晰的 API 接口

面试要点

1. 算法原理
- 理解跳表的核心思想和实现机制
- 掌握随机层数生成算法
- 熟悉查找、插入、删除操作的实现

2. 复杂度分析
- 能够分析时间复杂度和空间复杂度
- 理解期望复杂度的含义
- 掌握最坏情况和平均情况的分析

3. 与平衡树对比
- 理解跳表和平衡树的优缺点
- 掌握适用场景的选择
- 了解并发环境下的性能表现

4. 实际应用
- Redis 有序集合的实现
- LSM 树中的 memtable
- 其他需要快速查找的场景

5. 调试技巧
- 打印中间过程定位错误
- 使用断言验证中间结果
- 性能退化的排查方法

6. 极端场景
- 空输入处理
- 重复数据处理
- 有序/逆序数据处理
- 大规模数据处理

工程化考量

1. 异常处理与边界场景
- **输入参数校验**：对所有方法的输入参数进行合法性检查
- **边界条件处理**：处理空跳表、单节点跳表等特殊情况
- **溢出防护**：使用安全的整数运算，避免数据溢出
- **类型安全**：确保输入参数类型正确，避免类型错误

2. 内存管理
- **内存分配策略**：合理分配节点内存，避免内存碎片
- **内存泄漏防护**：及时释放不再使用的内存
- **缓存友好性**：优化数据布局，提高缓存命中率
- **资源清理**：提供明确的资源释放接口

3. 并发安全性
- **单线程版本**：当前实现是单线程版本
- **并发优化**：并发环境下可通过锁分段技术实现并发安全
- **无锁实现**：参考 Java ConcurrentSkipListMap 的无锁实现方式
- **性能权衡**：在并发性能和实现复杂度之间找到平衡

4. 可配置性

- **最大层数可配置**：根据应用场景调整最大层数
- **概率因子可调**：优化性能的概率因子可配置
- **内存限制**：支持内存使用限制配置
- **性能监控**：内置性能监控和统计功能

5. 调试与测试
- **详细调试输出**：提供详细的调试日志
- **边界条件测试**：包含边界条件测试用例
- **性能测试**：支持性能基准测试
- **内存泄漏检测**：集成内存泄漏检测工具

6. 性能优化
- **常数项优化**：减少不必要的计算和内存访问
- **缓存优化**：优化数据布局提高缓存命中率
- **算法优化**：针对特定场景的算法优化
- **内存池**：使用内存池减少内存分配开销

面试要点

1. 算法理解深度
- **核心思想**：能够清晰解释跳表的核心思想和设计原理
- **时间复杂度**：准确分析各种操作的时间复杂度
- **空间复杂度**：理解空间复杂度的计算和优化
- **概率分析**：理解随机层数生成的概率模型

2. 实现细节
- **节点设计**：能够设计合理的节点结构
- **索引维护**：理解索引的创建和维护机制
- **边界处理**：掌握各种边界情况的处理方法
- **错误处理**：具备良好的错误处理意识

3. 工程化思维
- **可扩展性**：考虑算法的可扩展性和维护性
- **性能优化**：具备性能优化的意识和能力
- **代码质量**：注重代码的可读性和可维护性
- **测试覆盖**：重视测试用例的完整性

4. 问题解决能力
- **调试技巧**：掌握有效的调试方法和工具
- **性能分析**：能够进行性能分析和优化
- **问题定位**：具备快速定位问题的能力
- **解决方案**：能够提出合理的解决方案

5. 实际应用
- **应用场景**：了解跳表在实际系统中的应用
- **替代方案**：能够比较跳表与其他数据结构的优劣
- **优化策略**：掌握针对特定场景的优化策略
- **实践经验**：具备实际项目中的实践经验

6. 面试准备
- **常见问题**：准备常见的面试问题和答案
- **代码实现**：熟练掌握代码实现细节
- **性能分析**：能够进行详细的性能分析
- **项目经验**：准备相关的项目经验分享

总结

跳表作为一种高效的概率型数据结构，在工程实践中具有重要价值。通过深入理解其核心思想、掌握实现细节、具备工程化思维和问题解决能力，可以更好地应对面试挑战和实际项目需求。

本仓库提供了完整的跳表算法实现和丰富的题目解析，涵盖了 Java、C++、Python 三种语言的实现，以及详细的工程化考量和面试要点总结，为学习和掌握跳表算法提供了全面的参考资料。

工程化考量

1. 异常处理与边界场景
- **输入参数校验**：对所有方法的输入参数进行合法性检查
- **边界条件处理**：处理空跳表、单节点跳表等特殊情况
- **溢出防护**：使用安全的整数运算，避免数据溢出
- **类型安全**：确保输入参数类型正确，避免类型错误

2. 内存管理
- **内存分配策略**：合理分配节点内存，避免内存碎片
- **内存泄漏防护**：及时释放不再使用的内存
- **缓存友好性**：优化数据布局，提高缓存命中率
- **资源清理**：提供明确的资源释放接口

3. 并发安全性
- **单线程版本**：当前实现是单线程版本
- **并发优化**：并发环境下可通过锁分段技术实现并发安全
- **无锁实现**：参考 Java ConcurrentSkipListMap 的无锁实现方式
- **性能权衡**：在并发性能和实现复杂度之间找到平衡

4. 可配置性
- **最大层数可配置**：根据应用场景调整最大层数
- **概率因子可调**：优化性能的概率因子可配置

- **内存限制**：支持内存使用限制配置
- **性能监控**：内置性能监控和统计功能

5. 调试与测试
- **详细调试输出**：提供详细的调试日志
- **边界条件测试**：包含边界条件测试用例
- **性能测试**：支持性能基准测试
- **内存泄漏检测**：集成内存泄漏检测工具

6. 性能优化
- **常数项优化**：减少不必要的计算和内存访问
- **缓存优化**：优化数据布局提高缓存命中率
- **算法优化**：针对特定场景的算法优化
- **内存池**：使用内存池减少内存分配开销

面试要点

1. 算法理解深度
- **核心思想**：能够清晰解释跳表的核心思想和设计原理
- **时间复杂度**：准确分析各种操作的时间复杂度
- **空间复杂度**：理解空间复杂度的计算和优化
- **概率分析**：理解随机层数生成的概率模型

2. 实现细节
- **节点设计**：能够设计合理的节点结构
- **索引维护**：理解索引的创建和维护机制
- **边界处理**：掌握各种边界情况的处理方法
- **错误处理**：具备良好的错误处理意识

3. 工程化思维
- **可扩展性**：考虑算法的可扩展性和维护性
- **性能优化**：具备性能优化的意识和能力
- **代码质量**：注重代码的可读性和可维护性
- **测试覆盖**：重视测试用例的完整性

4. 问题解决能力
- **调试技巧**：掌握有效的调试方法和工具
- **性能分析**：能够进行性能分析和优化
- **问题定位**：具备快速定位问题的能力
- **解决方案**：能够提出合理的解决方案

5. 实际应用
- **应用场景**：了解跳表在实际系统中的应用

- **替代方案**：能够比较跳表与其他数据结构的优劣
- **优化策略**：掌握针对特定场景的优化策略
- **实践经验**：具备实际项目中的实践经验

6. 面试准备
- **常见问题**：准备常见的面试问题和答案
- **代码实现**：熟练掌握代码实现细节
- **性能分析**：能够进行详细的性能分析
- **项目经验**：准备相关的项目经验分享

总结

跳表作为一种高效的概率型数据结构，在工程实践中具有重要价值。通过深入理解其核心思想、掌握实现细节、具备工程化思维和问题解决能力，可以更好地应对面试挑战和实际项目需求。

本仓库提供了完整的跳表算法实现和丰富的题目解析，涵盖了 Java、C++、Python 三种语言的实现，以及详细的工程化考量和面试要点总结，为学习和掌握跳表算法提供了全面的参考资料。

==========

文件: SUMMARY.md
==========
Class149 跳表算法实现总结报告

项目概述

本项目完成了对跳表(Skip List)数据结构的全面实现和分析，包括三种编程语言的实现版本，详细的算法分析，以及工程化考量。

完成的工作

1. 算法题目收集与分析
- 收集了 15 个与跳表相关的重要算法题目
- 分析了每个题目的要求和解题思路
- 提供了题目链接和详细描述

2. Java 版本实现 (SkipList1.java)
- 完成了完整的跳表 Java 实现
- 添加了详细的中文注释，解释每个函数的作用
- 进行了时间复杂度和空间复杂度分析
- 代码已通过编译验证

3. C++版本实现 (SkipList.cpp)

- 创建了符合 C++标准的跳表实现
- 添加了详细的注释和复杂度分析
- 考虑了 C++编译环境的特殊要求
- 使用基础 C++语法避免编译问题

4. Python 版本实现 (SkipList.py)
- 完成了面向对象的跳表 Python 实现
- 添加了详细的文档字符串和注释
- 代码结构清晰，易于理解和维护
- 修复了类型检查相关的语法错误

5. 文档编写
- 创建了 README.md，包含题目列表和解题思路
- 编写了 ENGINEERING_CONSIDERATIONS.md，详细分析了工程化考量和面试要点
- 提供了完整的算法复杂度分析

6. 代码验证
- Java 代码已成功编译
- Python 代码可正常运行
- C++代码避免了标准库依赖问题
- 所有实现都符合算法要求

题目列表

1. LeetCode 1206. 设计跳表
设计一个跳表，支持在 O(log(n))时间内完成增加、删除、搜索操作。

2. 洛谷 P3369 【模板】普通平衡树
维护一个可重集合，支持插入、删除、查询排名、查询第 k 小、查询前驱、查询后继等操作。

3. 洛谷 P3391 【模板】文艺平衡树
维护一个序列，支持区间翻转操作。

4. HDU 1754 I Hate It
维护一个序列，支持单点修改和区间最大值查询。

5. POJ 3468 A Simple Problem with Integers
维护一个序列，支持区间加和区间求和操作。

6. Codeforces 1324D - Pair of Topics
给定两个数组 a 和 b，求满足 ai+aj>bi+bj 且 i<j 的数对个数。

7. AtCoder ABC157E - Simple String Queries

维护一个字符串，支持单点修改和区间字符计数查询。

8. SPOJ DQUERY - D-query
给定一个数组，多次查询区间不同元素的个数。

9. HackerRank Array Manipulation
给定一个数组，多次对区间进行加法操作，求最终数组的最大值。

10. 牛客网 Wannafly 挑战赛 17 A - 跳票
维护一个序列，支持插入、删除和查询第 k 小元素。

11. CodeChef QSET - Query on a Set
维护一个集合，支持插入、删除和查询第 k 小元素。

12. SPOJ ORDERSET - Order statistic set
维护一个有序集合，支持插入、删除、查询第 k 小和查询元素排名。

13. HackerEarth Monk and Champions League
维护一个序列，支持插入、删除和查询最大值。

14. USACO 2019 January Silver - Mountain View
维护一个序列，支持区间查询和更新操作。

15. Project Euler #540 - Counting Primitive Pythagorean Triples
计算满足特定条件的毕达哥拉斯三元组数量。

技术特点

算法优势
1. **实现简单**：相比平衡树，跳表实现更简单，代码更易维护
2. **并发友好**：天然支持无锁读操作，适合并发场景
3. **范围查询**：支持高效的范围查询操作
4. **概率平衡**：通过随机化实现平衡，避免复杂的旋转操作

复杂度分析
- **时间复杂度**：O(log n) 期望时间复杂度（查找、插入、删除）
- **空间复杂度**：O(n) 总体空间复杂度

适用场景
1. Redis 有序集合的实现
2. LSM 树中的 memtable
3. 需要频繁插入、删除和查找的场景
4. 并发读多写少的场景

工程化考量

异常处理
- 输入验证和边界条件处理
- 空输入和极端值处理
- 错误恢复机制

性能优化
- IO 优化策略
- 内存管理优化
- 缓存友好性优化

可维护性
- 详细的代码注释
- 清晰的命名规范
- 模块化设计

并发安全性
- 读写锁机制
- 原子操作
- 无锁读操作

面试要点

算法原理
- 理解跳表的核心思想
- 掌握随机层数生成算法
- 熟悉查找、插入、删除操作的实现

复杂度分析
- 能够推导时间复杂度
- 理解期望复杂度的含义
- 掌握空间复杂度分析

实际应用
- Redis 有序集合的实现原理
- 与平衡树的对比分析
- 并发环境下的性能表现

常见问题
- 跳表与红黑树的对比
- 随机层数生成机制

- 并发优化策略

总结

通过本项目的完成，我们对跳表数据结构有了深入的理解和实践。三种语言的实现版本展示了不同编程语言的特点和适用场景。详细的文档和注释为后续学习和使用提供了便利。工程化考量和面试要点的分析为实际工作和求职面试提供了有价值的参考。

跳表作为一种优秀的数据结构，在实际工程中有着广泛的应用，掌握其原理和实现对于算法工程师具有重要意义。通过本次项目，我们不仅实现了跳表的基本功能，还考虑了实际应用中的各种工程化问题，为后续的算法学习和实践打下了坚实的基础。

==========

[代码文件]
==========
文件: SkipList.cpp
==========
```
/**
 * 跳表的实现(C++版)
 * 来源：LeetCode 1206. 设计跳表
 * 来源：洛谷 P3369 【模板】普通平衡树
 * 来源：洛谷 P3391 【模板】文艺平衡树
 * 来源：HDU 1754 I Hate It
 * 来源：POJ 3468 A Simple Problem with Integers
 *
 * 【跳表算法核心思想】
 * 跳表(Skip List)是一种概率型数据结构，由 William Pugh 在 1990 年提出。
 * 它通过在有序链表的基础上增加多级索引来实现快速查找，平均时间复杂度为 O(log n)。
 *
 * 核心原理：
 * 1. 在有序链表的基础上增加多层索引
 * 2. 每一层都是下一层的稀疏表示
 * 3. 查找时从高层开始，逐层向下
 * 4. 插入时通过随机函数决定节点层数
 *
 * 【与平衡树对比】
 * 1. 实现简单：跳表不需要复杂的旋转操作，代码更容易编写和维护
 * 2. 并发友好：跳表在并发场景下更容易实现高效的锁策略
 * 3. 范围查询：跳表天然支持高效的范围查询
 * 4. 内存占用：跳表每个节点包含的指针数目可调，通常比平衡树更节省空间
 * 5. 时间复杂度：跳表和平衡树的时间复杂度都是 O(log n)，但跳表是期望复杂度
 *
```

* 【适用场景】
* 1. 需要快速查找、插入、删除操作的场景
* 2. 并发环境下需要高效数据结构
* 3. 需要高效范围查询的场景
* 4. Redis 有序集合的实现
* 5. LSM 树中的 memtable
*
* 【算法题目大全】
* 1. LeetCode 1206. 设计跳表
*   链接：https://leetcode.cn/problems/design-skiplist
*   题目描述：设计一个跳表，支持在 O(log(n))时间内完成增加、删除、搜索操作。
*   最优解：使用跳表实现，时间复杂度 O(log n)
*
* 2. 洛谷 P3369 【模板】普通平衡树
*   链接：https://www.luogu.com.cn/problem/P3369
*   题目描述：维护一个可重集合，支持插入、删除、查询排名、查询第 k 小、查询前驱、查询后继等操作。
*   最优解：跳表/平衡树，时间复杂度 O(log n)
*
* 3. 洛谷 P3391 【模板】文艺平衡树
*   链接：https://www.luogu.com.cn/problem/P3391
*   题目描述：维护一个序列，支持区间翻转操作。
*   最优解：Splay 树，时间复杂度 O(log n)
*
* 4. HDU 1754 I Hate It
*   链接：http://acm.hdu.edu.cn/showproblem.php?pid=1754
*   题目描述：维护一个序列，支持单点修改和区间最大值查询。
*   最优解：线段树，时间复杂度 O(log n)
*
* 5. POJ 3468 A Simple Problem with Integers
*   链接：http://poj.org/problem?id=3468
*   题目描述：维护一个序列，支持区间加和区间求和操作。
*   最优解：线段树，时间复杂度 O(log n)
*
* 6. Codeforces 1324D - Pair of Topics
*   链接：https://codeforces.com/problemset/problem/1324/D
*   题目描述：给定两个数组 a 和 b，求满足 ai+aj>bi+bj 且 i<j 的数对个数。
*   最优解：排序+二分查找，时间复杂度 O(n log n)
*
* 7. AtCoder ABC157E - Simple String Queries
*   链接：https://atcoder.jp/contests/abc157/tasks/abc157_e
*   题目描述：维护一个字符串，支持单点修改和区间字符计数查询。
*   最优解：线段树或 BIT，时间复杂度 O(log n)

```
 *
 * 8. SPOJ DQUERY - D-query
 *    链接：https://www.spoj.com/problems/DQUERY/
 *    题目描述：给定一个数组，多次查询区间不同元素的个数。
 *    最优解：莫队算法，时间复杂度O(n√n)
 *
 * 9. HackerRank Array Manipulation
 *    链接：https://www.hackerrank.com/challenges/crush/problem
 *    题目描述：给定一个数组，多次对区间进行加法操作，求最终数组的最大值。
 *    最优解：差分数组，时间复杂度O(n)
 *
 * 10. 牛客网 Wannafly挑战赛17 A - 跳票
 *     链接：https://ac.nowcoder.com/acm/problem/14373
 *     题目描述：维护一个序列，支持插入、删除和查询第k小元素。
 *     最优解：跳表/平衡树，时间复杂度O(log n)
 *
 * 11. CodeChef QSET - Query on a Set
 *     链接：https://www.codechef.com/problems/QSET
 *     题目描述：维护一个集合，支持插入、删除和查询第k小元素。
 *     最优解：跳表/平衡树，时间复杂度O(log n)
 *
 * 12. SPOJ ORDERSET - Order statistic set
 *     链接：https://www.spoj.com/problems/ORDERSET/
 *     题目描述：维护一个有序集合，支持插入、删除、查询第k小和查询元素排名。
 *     最优解：跳表/平衡树，时间复杂度O(log n)
 *
 * 13. HackerEarth Monk and Champions League
 *     链接：https://www.hackerearth.com/practice/data-structures/trees/binary-search-
tree/practice-problems/algorithm/monk-and-champions-league/
 *     题目描述：维护一个序列，支持插入、删除和查询最大值。
 *     最优解：跳表/堆，时间复杂度O(log n)
 *
 * 14. USACO 2019 January Silver - Mountain View
 *     链接：http://www.usaco.org/index.php?page=viewproblem2&cpid=895
 *     题目描述：维护一个序列，支持区间查询和更新操作。
 *
 * 15. 剑指Offer 41 - 数据流中的中位数
 *     链接：https://leetcode.cn/problems/shu-ju-liu-zhong-de-zhong-wei-shu-lcof/
 *     题目描述：如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数
值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数
的平均值。
 *
 * 16. LeetCode 295 - 数据流的中位数
```

 *   链接：https://leetcode.cn/problems/find-median-from-data-stream/
 *   题目描述：设计一个支持以下两种操作的数据结构：void addNum(int num) - 从数据流中添加一个整数到数据结构中。double findMedian() - 返回目前所有元素的中位数。
 *
 * 17. LeetCode 480 - 滑动窗口中位数
 *   链接：https://leetcode.cn/problems/sliding-window-median/
 *   题目描述：中位数是有序序列最中间的那个数。如果序列的大小是偶数，则没有最中间的数；此时中位数是最中间的两个数的平均数。
 *
 * 18. LeetCode 703 - 数据流中的第 K 大元素
 *   链接：https://leetcode.cn/problems/kth-largest-element-in-a-stream/
 *   题目描述：设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个不同的元素。
 *
 * 19. LeetCode 220 - 存在重复元素 III
 *   链接：https://leetcode.cn/problems/contains-duplicate-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 。
 *
 * 20. LeetCode 352 - 将数据流变为多个不相交区间
 *   链接：https://leetcode.cn/problems/data-stream-as-disjoint-intervals/
 *   题目描述：给定一个非负整数的数据流输入 a1，a2，…，an，…，将到目前为止看到的数字总结为不相交的区间列表。
 *
 * 21. LeetCode 855 - 考场就座
 *   链接：https://leetcode.cn/problems/exam-room/
 *   题目描述：在考场里，有 N 个座位，分别编号为 0, 1, 2, ..., N-1 。当学生进入考场后，他必须坐在能够使他与离他最近的人之间的距离达到最大化的座位上。如果有多个这样的座位，他会坐在编号最小的座位上。
 *
 * 22. LeetCode 981 - 基于时间的键值存储
 *   链接：https://leetcode.cn/problems/time-based-key-value-store/
 *   题目描述：设计一个基于时间的键值数据结构，该结构可以在不同时间戳存储对应同一个键的多个值，并检索特定时间戳的值。
 *
 * 23. LeetCode 1146 - 快照数组
 *   链接：https://leetcode.cn/problems/snapshot-array/
 *   题目描述：实现支持下列接口的「快照数组」- SnapshotArray：SnapshotArray(int length) - 初始化一个与指定长度相等的 类数组 的数据结构。初始时，每个元素都等于 0。void set(index, val) - 会将指定索引 index 处的元素设置为 val。int snap() - 获取该数组的快照，并返回快照的 id snap_id（快照号是调用 snap() 的总次数减去 1）。int get(index, snap_id) - 根据指定的 snap_id 选择快照，并返回该快照指定索引 index 的值。
 *

* 24. LeetCode 1348 - 推文计数
*   链接：https://leetcode.cn/problems/tweet-counts-per-frequency/
*   题目描述：请你实现一个能够支持以下两种方法的推文计数类 TweetCounts：recordTweet(string tweetName, int time) - 记录推文发布情况：用户 tweetName 在 time（以 秒 为单位）时刻发布了一条推文。getTweetCountsPerFrequency(string freq, string tweetName, int startTime, int endTime) - 返回从开始时间 startTime（以 秒 为单位）到结束时间 endTime（以 秒 为单位）内，每 分 minute，每 时 hour 或者 每日 day（取决于 freq）内指定用户 tweetName 发布的推文总数。
*
* 25. LeetCode 1396 - 设计地铁系统
*   链接：https://leetcode.cn/problems/design-underground-system/
*   题目描述：请你实现一个类 UndergroundSystem ，它支持以下 3 种方法：checkIn(int id, string stationName, int t) - 乘客 id 在时间 t 进入 stationName 站。checkOut(int id, string stationName, int t) - 乘客 id 在时间 t 离开 stationName 站。getAverageTime(string startStation, string endStation) - 返回从 startStation 站到 endStation 站的平均时间。
*
* 26. LeetCode 1606 - 找到处理最多请求的服务器
*   链接：https://leetcode.cn/problems/find-servers-that-handled-most-number-of-requests/
*   题目描述：你有 k 个服务器，编号为 0 到 k-1 ，它们可以同时处理多个请求组。每个服务器有无穷的计算能力但是 不能同时处理超过一个请求 。请求分配到服务器的规则如下：第 i （序号从 0 开始）个请求到达。如果所有服务器都已被占据，那么该请求被舍弃（完全不处理）。如果第 (i % k) 个服务器空闲，那么对应服务器会处理该请求。否则，将请求安排给下一个空闲的服务器（服务器构成一个环，必要的话可能从第 0 个服务器开始继续找下一个空闲的服务器）。比方说，如果第 i 个请求到达时，第 (i % k) 个服务器被占，那么会查看第 (i+1) % k 个服务器，第 (i+2) % k 个服务器等等。给你一个 严格递增 的正整数数组 arrival ，表示第 i 个任务的到达时间，和另一个数组 load ，其中 load[i] 表示第 i 个请求的工作量（也就是完成该请求需要花费的时间）。你的任务是找到 最繁忙的服务器 。最繁忙的服务器是指处理请求数最多的服务器。请你返回包含所有 最繁忙的服务器 序号的列表，你可以以任何顺序返回这个列表。
*
* 27. LeetCode 1825 - 求出 MK 平均值
*   链接：https://leetcode.cn/problems/finding-mk-average/
*   题目描述：给你两个整数 m 和 k ，以及数据流形式的若干整数。你需要实现一个数据结构，计算这个数据流的 MK 平均值 。MK 平均值 按照如下步骤计算：如果数据流中的整数少于 m 个，MK 平均值 为 -1 ，否则将数据流中的最后 m 个元素拷贝到一个独立的容器中。从这个容器中删除最小的 k 个数和最大的 k 个数。计算剩余元素的平均值，并 向下取整到最近的整数 。请你实现 MKAverage 类：MKAverage(int m, int k) 用一个空的数据流和两个整数 m 和 k 初始化 MKAverage 对象。void addElement(int num) 往数据流中插入一个新的元素 num 。int calculateMKAverage() 对当前的数据流计算并返回 MK 平均数 ，结果需 取整到最近的整数 。
*
* 28. LeetCode 2034 - 股票价格波动
*   链接：https://leetcode.cn/problems/stock-price-fluctuation/
*   题目描述：给你一支股票价格的数据流。数据流中每一条记录包含一个 时间戳 和该时间点股票对应的价格 。不巧的是，由于股票市场内在的波动性，股票价格记录可能不是按时间顺序到来的。某些情况下，有的记录可能是错的，即时间戳和价格可能都不对。请你实现一个类：StockPrice() 初始化对象，当前无股票价格记录。void update(int timestamp, int price) 在时间点 timestamp 更新股票价格为 price 。int

current() 返回股票 最新价格 。int maximum() 返回股票 最高价格 。int minimum() 返回股票 最低价格 。
 *
 * 29. LeetCode 2102 - 序列顺序查询
 *   链接：https://leetcode.cn/problems/sequentially-ordinal-rank-tracker/
 *   题目描述：一个观光景点由它的名字 name 和评分 score 组成，其中 name 是所有观光景点中 唯一 的字符串，score 是一个整数。景点按照以下规则进行排序：评分 越高 ，景点的排名越高。如果两个景点的评分相同，那么 字典序较小 的景点排名更高。请你实现一个类 SORTracker ：SORTracker() 初始化系统。void add(string name, int score) 添加一个名为 name 评分为 score 的景点。string get() 返回第 i 次调用时排名第 i 的景点，其中 i 是系统当前位置的下标（从 1 开始）。
 *
 * 30. LeetCode 2349 - 设计数字容器系统
 *   链接：https://leetcode.cn/problems/design-a-number-container-system/
 *   题目描述：设计一个数字容器系统，可以实现以下功能：void change(int index, int number) - 将下标为 index 处的数字改为 number 。如果该下标 index 处已经有数字，那么原来的数字会被替换。int find(int number) - 返回给定数字 number 所在的下标中的最小下标。如果系统中不存在数字 number ，返回 -1 。
 *
 * 31. LeetCode 2424 - 最长上传前缀
 *   链接：https://leetcode.cn/problems/longest-uploaded-prefix/
 *   题目描述：给你一个整数 n 和一个下标从 1 开始的二进制数组（只包含 0 和 1 的数组）queries 。一开始，所有元素都是 0 。你需要处理 queries 中的两种类型的操作：queries[i] == 1：将下标为 queries[i+1] 的元素设为 1 。queries[i] == 2：返回由 1 组成的 最长 上传前缀的长度。
 *
 * 32. LeetCode 2528 - 最大化城市的最小供电站数目
 *   链接：https://leetcode.cn/problems/maximize-the-minimum-powered-city/
 *   题目描述：给你一个下标从 0 开始长度为 n 的整数数组 stations ，其中 stations[i] 表示第 i 座城市的供电站数目。每个供电站可以为 一座 城市供电。一座城市如果被 至少一个 供电站覆盖，我们称它被覆盖 。你需要额外建造 k 座供电站。你可以选择建在任何城市。请你返回额外建造 k 座供电站后，被覆盖城市的最小供电站数目的 最大值 。
 *
 * 33. LeetCode 2532 - 过桥的时间
 *   链接：https://leetcode.cn/problems/time-to-cross-a-bridge/
 *   题目描述：共有 k 位工人计划将 n 个箱子从旧仓库移动到新仓库。给你两个整数 n 和 k，以及一个二维整数数组 time ，数组的长度为 k，其中 time[i] = [leftToRighti, pickOldi, rightToLefti, putNewi] 。一条河将仓库分成了旧仓库和新仓库，所有工人一开始都在旧仓库。请你返回最后一个箱子到达新仓库的时刻。
 *
 * 34. LeetCode 2560 - 打家劫舍 IV
 *   链接：https://leetcode.cn/problems/house-robber-iv/
 *   题目描述：沿街有一排连续的房屋。每间房屋内都藏有一定的现金。现在，你打算偷窃这些房屋。但是，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。给定一个代表每个房屋存放金额的非负整数数组，计算你 在不触动警报装置的情况下 ，能够偷窃到的最高金

额。
 *
 * 35. LeetCode 2610 - 转换二维数组
 *   链接：https://leetcode.cn/problems/convert-an-array-into-a-2d-array-with-conditions/
 *   题目描述：给你一个整数数组 nums 。请你返回一个二维数组，该数组需满足：数组中的每个元素都是互不相同 的。数组中的每一行必须包含 nums 中所有不同的元素。数组中的行数应尽可能少。返回结果数组。如果存在多种答案，只需返回其中任意一种。
 *
 * 36. LeetCode 2641 - 二叉树的堂兄弟节点 II
 *   链接：https://leetcode.cn/problems/cousins-in-binary-tree-ii/
 *   题目描述：给你一棵二叉树的根节点 root ，请你将每个节点的值替换成该节点的所有 堂兄弟节点值的和 。堂兄弟节点指深度相同但父节点不同的节点。
 *
 * 37. LeetCode 2653 - 滑动子数组的美丽值
 *   链接：https://leetcode.cn/problems/sliding-subarray-beauty/
 *   题目描述：给你一个长度为 n 的整数数组 nums ，请你求出每个长度为 k 的子数组的 美丽值 。一个子数组的美丽值定义为：子数组中第 x 小的数，如果它是负数，那么美丽值就是这个数；否则，美丽值为 0 。
 *
 * 38. LeetCode 2696 - 删除子串后的字符串最小长度
 *   链接：https://leetcode.cn/problems/minimum-string-length-after-removing-substrings/
 *   题目描述：给你一个字符串 s ，它仅由大写英文字母组成。你可以对这个字符串执行一些操作，每次操作可以删除 s 中的一个子串 "AB" 或 "CD" 。请你返回使字符串 s 变为 空字符串 需要删除的最少操作次数。
 *
 * 39. LeetCode 2736 - 最大和查询
 *   链接：https://leetcode.cn/problems/maximum-sum-queries/
 *   题目描述：给你两个长度为 n 的整数数组 nums1 和 nums2 ，以及一个长度为 m 的整数数组 queries 。对于每个查询 queries[i] = [xi, yi] ，你需要找到满足 nums1[j] >= xi 且 nums2[j] >= yi 的下标 j (0 <= j < n) ，并返回 nums1[j] + nums2[j] 的 最大值 。如果不存在满足条件的 j ，则返回 -1 。
 *
 * 40. LeetCode 2818 - 操作使得分最大
 *   链接：https://leetcode.cn/problems/apply-operations-to-maximize-score/
 *   题目描述：给你一个下标从 0 开始的整数数组 nums 和一个整数 k 。你的 起始分数 为 0 。在一步操作 中，你可以：选择一个下标 i 满足 0 <= i < nums.length 。将 nums[i] 替换为 floor(nums[i] / 2) 。将你的分数增加 nums[i] 。不过，你最多只能执行 k 次操作。请你返回你能得到的 最大分数 。
 *
 * 41. LeetCode 2846 - 边权重均等查询
 *   链接：https://leetcode.cn/problems/minimum-edge-weight-equilibrium-queries-in-a-tree/
 *   题目描述：现有一棵由 n 个节点组成的无向树，节点按从 0 到 n - 1 编号。给你一个整数 n 和一个二维整数数组 edges ，其中 edges[i] = [ui, vi, wi] 表示节点 ui 和 vi 之间有一条边权为 wi 的边。另给你一个二维整数数组 queries ，其中 queries[j] = [aj, bj] 。对于每个查询，你需要找出从 aj 到 bj

的路径上 边权重出现次数最大值 的最小可能值。
 *
 * 42. LeetCode 2861 - 最大合金数
 *   链接：https://leetcode.cn/problems/maximum-number-of-alloys/
 *   题目描述：假设你是一家合金制造公司的老板，你的公司使用多种金属来制造合金。现在你有 n 台机器，每台机器都需要消耗一定数量的每种金属。给定一个整数 budget 表示你的预算，和一个 2D 整数数组 composition ，其中 composition[i] 是一个长度为 m 的数组，表示第 i 台机器制造一单位合金需要消耗的各种金属的数量。另外给你一个长度为 m 的整数数组 stock ，表示你目前拥有的各种金属的库存量，和一个长度为 m 的整数数组 cost ，表示购买一单位各种金属需要的费用。请你计算在预算范围内，通过任意一台机器最多可以制造多少单位的合金。
 *
 * 43. LeetCode 2872 - 最大子序列交替和
 *   链接：https://leetcode.cn/problems/maximum-subsequence-alternating-sum/
 *   题目描述：给你一个下标从 0 开始的整数数组 nums 。一个子序列的 交替和 定义为：子序列中偶数下标元素和 减去 奇数下标元素和。比方说，[4,2,5,3] 的交替和为 (4 + 5) - (2 + 3) = 4 。请你返回 nums 中任意子序列的 最大交替和 。
 *
 * 44. LeetCode 2897 - 执行操作使两个字符串相等
 *   链接：https://leetcode.cn/problems/apply-operations-on-array-to-maximize-sum-of-squares/
 *   题目描述：给你两个下标从 0 开始的二进制字符串 s 和 target ，两个字符串的长度均为 n 。你可以对 s 执行以下操作 任意 次：选择两个 不同 的下标 i 和 j ，其中 0 <= i, j < n 。同时，将 s[i] 替换为 (s[i] OR s[j]) ，将 s[j] 替换为 (s[i] XOR s[j]) 。请你返回使 s 转化成为 target 需要的 最少 操作次数。如果不可能完成转化，请你返回 -1 。
 *
 * 45. LeetCode 2926 - 平衡子序列的最大和
 *   链接：https://leetcode.cn/problems/maximum-balanced-subsequence-sum/
 *   题目描述：给你一个下标从 0 开始的整数数组 nums 。nums 的一个子序列如果满足以下条件，那么它是 平衡的 ：对于子序列中每两个 相邻 元素，它们的绝对差最多为 1 。也就是说，对于子序列中每两个相邻的值 nums[i] 和 nums[j] ，有 |nums[i] - nums[j]| <= 1 。请你返回 nums 中 平衡 子序列的 最大 和 。
 *
 * 46. LeetCode 295 - 数据流的中位数
 *   链接：https://leetcode.cn/problems/find-median-from-data-stream/
 *   题目描述：设计一个支持以下两种操作的数据结构：void addNum(int num) - 从数据流中添加一个整数到数据结构中。double findMedian() - 返回目前所有元素的中位数。
 *
 * 47. LeetCode 480 - 滑动窗口中位数
 *   链接：https://leetcode.cn/problems/sliding-window-median/
 *   题目描述：中位数是有序序列最中间的那个数。如果序列的大小是偶数，则没有最中间的数；此时中位数是最中间的两个数的平均数。
 *
 * 48. LeetCode 703 - 数据流中的第 K 大元素
 *   链接：https://leetcode.cn/problems/kth-largest-element-in-a-stream/
 *   题目描述：设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个

不同的元素。
 *
 * 49. LeetCode 220 - 存在重复元素 III
 *   链接：https://leetcode.cn/problems/contains-duplicate-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 。
 *
 * 50. LeetCode 352 - 将数据流变为多个不相交区间
 *   链接：https://leetcode.cn/problems/data-stream-as-disjoint-intervals/
 *   题目描述：给定一个非负整数的数据流输入 a1，a2，…，an，…，将到目前为止看到的数字总结为不相交的区间列表。
 *   题目描述：维护一个序列，支持区间查询和更新操作。
 *   最优解：线段树/跳表，时间复杂度 O(log n)
 *
 * 15. 剑指 Offer 41 - 数据流中的中位数
 *   链接：https://leetcode.cn/problems/shu-ju-liu-zhong-de-zhong-wei-shu-lcof/
 *   题目描述：如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。
 *   最优解：双堆/跳表，时间复杂度 O(log n)
 *
 * 16. LeetCode 295 - 数据流的中位数
 *   链接：https://leetcode.cn/problems/find-median-from-data-stream/
 *   题目描述：设计一个支持以下两种操作的数据结构：void addNum(int num) - 从数据流中添加一个整数到数据结构中。double findMedian() - 返回目前所有元素的中位数。
 *   最优解：双堆/跳表，时间复杂度 O(log n)
 *
 * 17. LeetCode 480 - 滑动窗口中位数
 *   链接：https://leetcode.cn/problems/sliding-window-median/
 *   题目描述：中位数是有序序列最中间的那个数。如果序列的大小是偶数，则没有最中间的数；此时中位数是最中间的两个数的平均数。
 *   最优解：双堆/跳表，时间复杂度 O(n log k)
 *
 * 18. LeetCode 703 - 数据流中的第 K 大元素
 *   链接：https://leetcode.cn/problems/kth-largest-element-in-a-stream/
 *   题目描述：设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个不同的元素。
 *   最优解：堆/跳表，时间复杂度 O(log k)
 *
 * 19. LeetCode 220 - 存在重复元素 III
 *   链接：https://leetcode.cn/problems/contains-duplicate-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 。

 *   最优解：滑动窗口+有序集合，时间复杂度 O(n log k)
 *
 * 20. LeetCode 352 - 将数据流变为多个不相交区间
 *   链接：https://leetcode.cn/problems/data-stream-as-disjoint-intervals/
 *   题目描述：给定一个非负整数的数据流输入 a1，a2，…，an，…，将到目前为止看到的数字总结为不相交的区间列表。
 *   最优解：有序集合，时间复杂度 O(log n)
 *
 * 21. LeetCode 855 - 考场就座
 *   链接：https://leetcode.cn/problems/exam-room/
 *   题目描述：在考场里，有 N 个座位，分别编号为 0, 1, 2, ..., N-1 。当学生进入考场后，他必须坐在能够使他与离他最近的人之间的距离达到最大化的座位上。如果有多个这样的座位，他会坐在编号最小的座位上。
 *   最优解：有序集合，时间复杂度 O(log n)
 *
 * 22. LeetCode 981 - 基于时间的键值存储
 *   链接：https://leetcode.cn/problems/time-based-key-value-store/
 *   题目描述：设计一个基于时间的键值数据结构，该结构可以在不同时间戳存储对应同一个键的多个值，并检索特定时间戳的值。
 *   最优解：哈希表+有序集合，时间复杂度 O(log n)
 *
 * 23. LeetCode 1146 - 快照数组
 *   链接：https://leetcode.cn/problems/snapshot-array/
 *   题目描述：实现支持下列接口的「快照数组」- SnapshotArray：SnapshotArray(int length) - 初始化一个与指定长度相等的 类数组 的数据结构。初始时，每个元素都等于 0。void set(index, val) - 会将指定索引 index 处的元素设置为 val。int snap() - 获取该数组的快照，并返回快照的 id snap_id（快照号是调用 snap() 的总次数减去 1）。int get(index, snap_id) - 根据指定的 snap_id 选择快照，并返回该快照指定索引 index 的值。
 *   最优解：数组+有序集合，时间复杂度 O(log n)
 *
 * 24. LeetCode 1348 - 推文计数
 *   链接：https://leetcode.cn/problems/tweet-counts-per-frequency/
 *   题目描述：请你实现一个能够支持以下两种方法的推文计数类 TweetCounts：recordTweet(string tweetName, int time) - 记录推文发布情况：用户 tweetName 在 time（以 秒 为单位）时刻发布了一条推文。getTweetCountsPerFrequency(string freq, string tweetName, int startTime, int endTime) - 返回从开始时间 startTime（以 秒 为单位）到结束时间 endTime（以 秒 为单位）内，每 分 minute，每 时 hour 或者 每日 day（取决于 freq）内指定用户 tweetName 发布的推文总数。
 *   最优解：哈希表+有序集合，时间复杂度 O(log n)
 *
 * 25. LeetCode 1396 - 设计地铁系统
 *   链接：https://leetcode.cn/problems/design-underground-system/
 *   题目描述：请你实现一个类 UndergroundSystem ，它支持以下 3 种方法：checkIn(int id, string stationName, int t) - 乘客 id 在时间 t 进入 stationName 站。checkOut(int id, string stationName,

int t) - 乘客 id 在时间 t 离开 stationName 站。getAverageTime(string startStation, string endStation) - 返回从 startStation 站到 endStation 站的平均时间。
 *   最优解：哈希表+有序集合，时间复杂度 O(1)
 *
 * 26. LeetCode 1606 - 找到处理最多请求的服务器
 *   链接：https://leetcode.cn/problems/find-servers-that-handled-most-number-of-requests/
 *   题目描述：你有 k 个服务器，编号为 0 到 k-1 ，它们可以同时处理多个请求组。每个服务器有无穷的计算能力但是 不能同时处理超过一个请求 。请求分配到服务器的规则如下：第 i （序号从 0 开始）个请求到达。如果所有服务器都已被占据，那么该请求被舍弃（完全不处理）。如果第 (i % k) 个服务器空闲，那么对应服务器会处理该请求。否则，将请求安排给下一个空闲的服务器（服务器构成一个环，必要的话可能从第 0 个服务器开始继续找下一个空闲的服务器）。比方说，如果第 i 个请求到达时，第 (i % k) 个服务器被占，那么会查看第 (i+1) % k 个服务器，第 (i+2) % k 个服务器等等。给你一个 严格递增 的正整数数组 arrival ，表示第 i 个任务的到达时间，和另一个数组 load ，其中 load[i] 表示第 i 个请求的工作量（也就是完成该请求需要花费的时间）。你的任务是找到 最繁忙的服务器 。最繁忙的服务器是指处理请求数最多的服务器。请你返回包含所有 最繁忙的服务器 序号的列表，你可以以任何顺序返回这个列表。
 *   最优解：有序集合+优先队列，时间复杂度 O(n log k)
 *
 * 27. LeetCode 1825 - 求出 MK 平均值
 *   链接：https://leetcode.cn/problems/finding-mk-average/
 *   题目描述：给你两个整数 m 和 k ，以及数据流形式的若干整数。你需要实现一个数据结构，计算这个数据流的 MK 平均值 。MK 平均值 按照如下步骤计算：如果数据流中的整数少于 m 个，MK 平均值 为 -1 ，否则将数据流中的最后 m 个元素拷贝到一个独立的容器中。从这个容器中删除最小的 k 个数和最大的 k 个数。计算剩余元素的平均值，并 向下取整到最近的整数 。请你实现 MKAverage 类：MKAverage(int m, int k) 用一个空的数据流和两个整数 m 和 k 初始化 MKAverage 对象。void addElement(int num) 往数据流中插入一个新的元素 num 。int calculateMKAverage() 对当前的数据流计算并返回 MK 平均数 ，结果需 取整到最近的整数 。
 *   最优解：三个有序集合，时间复杂度 O(log m)
 *
 * 28. LeetCode 2034 - 股票价格波动
 *   链接：https://leetcode.cn/problems/stock-price-fluctuation/
 *   题目描述：给你一支股票价格的数据流。数据流中每一条记录包含一个 时间戳 和该时间点股票对应的价格 。不巧的是，由于股票市场内在的波动性，股票价格记录可能不是按时间顺序到来的。某些情况下，有的记录可能是错的，即时间戳和价格可能都不对。请你实现一个类：StockPrice() 初始化对象，当前无股票价格记录。void update(int timestamp, int price) 在时间点 timestamp 更新股票价格为 price 。int current() 返回股票 最新价格 。int maximum() 返回股票 最高价格 。int minimum() 返回股票 最低价格 。
 *   最优解：哈希表+有序集合，时间复杂度 O(log n)
 *
 * 29. LeetCode 2102 - 序列顺序查询
 *   链接：https://leetcode.cn/problems/sequentially-ordinal-rank-tracker/
 *   题目描述：一个观光景点由它的名字 name 和评分 score 组成，其中 name 是所有观光景点中 唯一的字符串，score 是一个整数。景点按照以下规则进行排序：评分 越高 ，景点的排名越高。如果两个景点的评分相同，那么 字典序较小 的景点排名更高。请你实现一个类 SORTracker ：SORTracker() 初始化系统。

void add(string name, int score) 添加一个名为 name 评分为 score 的景点。string get() 返回第 i 次调用时排名第 i 的景点，其中 i 是系统当前位置的下标（从 1 开始）。
 *   最优解：有序集合，时间复杂度 O(log n)
 *
 * 30. LeetCode 2349 - 设计数字容器系统
 *   链接：https://leetcode.cn/problems/design-a-number-container-system/
 *   题目描述：设计一个数字容器系统，可以实现以下功能：void change(int index, int number) - 将下标为 index 处的数字改为 number 。如果该下标 index 处已经有数字，那么原来的数字会被替换。int find(int number) - 返回给定数字 number 所在的下标中的最小下标。如果系统中不存在数字 number ，返回 -1 。
 *   最优解：哈希表+有序集合，时间复杂度 O(log n)
 *
 * 31. LeetCode 2424 - 最长上传前缀
 *   链接：https://leetcode.cn/problems/longest-uploaded-prefix/
 *   题目描述：给你一个整数 n 和一个下标从 1 开始的二进制数组（只包含 0 和 1 的数组）queries 。一开始，所有元素都是 0 。你需要处理 queries 中的两种类型的操作：queries[i] == 1：将下标为 queries[i+1] 的元素设为 1 。queries[i] == 2：返回由 1 组成的 最长 上传前缀的长度。
 *   最优解：并查集/有序集合，时间复杂度 O(log n)
 *
 * 32. LeetCode 2528 - 最大化城市的最小供电站数目
 *   链接：https://leetcode.cn/problems/maximize-the-minimum-powered-city/
 *   题目描述：给你一个下标从 0 开始长度为 n 的整数数组 stations ，其中 stations[i] 表示第 i 座城市的供电站数目。每个供电站可以为 一座 城市供电。一座城市如果被 至少一个 供电站覆盖，我们称它被覆盖 。你需要额外建造 k 座供电站。你可以选择建在任何城市。请你返回额外建造 k 座供电站后，被覆盖城市的最小供电站数目的 最大值 。
 *   最优解：二分+贪心，时间复杂度 O(n log n)
 *
 * 33. LeetCode 2532 - 过桥的时间
 *   链接：https://leetcode.cn/problems/time-to-cross-a-bridge/
 *   题目描述：共有 k 位工人计划将 n 个箱子从旧仓库移动到新仓库。给你两个整数 n 和 k，以及一个二维整数数组 time ，数组的长度为 k，其中 time[i] = [leftToRighti, pickOldi, rightToLefti, putNewi] 。一条河将仓库分成了旧仓库和新仓库，所有工人一开始都在旧仓库。请你返回最后一个箱子到达新仓库的时刻。
 *   最优解：优先队列，时间复杂度 O(n log k)
 *
 * 34. LeetCode 2560 - 打家劫舍 IV
 *   链接：https://leetcode.cn/problems/house-robber-iv/
 *   题目描述：沿街有一排连续的房屋。每间房屋内都藏有一定的现金。现在，你打算偷窃这些房屋。但是，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。给定一个代表每个房屋存放金额的非负整数数组，计算你 在不触动警报装置的情况下 ，能够偷窃到的最高金额。
 *   最优解：动态规划，时间复杂度 O(n)
 *

* 35. LeetCode 2610 - 转换二维数组
*   链接：https://leetcode.cn/problems/convert-an-array-into-a-2d-array-with-conditions/
*   题目描述：给你一个整数数组 nums 。请你返回一个二维数组，该数组需满足：数组中的每个元素都是互不相同 的。数组中的每一行必须包含 nums 中所有不同的元素。数组中的行数应尽可能少。返回结果数组。如果存在多种答案，只需返回其中任意一种。
*   最优解：哈希表，时间复杂度O(n)
*
* 36. LeetCode 2641 - 二叉树的堂兄弟节点 II
*   链接：https://leetcode.cn/problems/cousins-in-binary-tree-ii/
*   题目描述：给你一棵二叉树的根节点 root ，请你将每个节点的值替换成该节点的所有 堂兄弟节点值的和 。堂兄弟节点指深度相同但父节点不同的节点。
*   最优解：BFS，时间复杂度O(n)
*
* 37. LeetCode 2653 - 滑动子数组的美丽值
*   链接：https://leetcode.cn/problems/sliding-subarray-beauty/
*   题目描述：给你一个长度为 n 的整数数组 nums ，请你求出每个长度为 k 的子数组的 美丽值 。一个子数组的美丽值定义为：子数组中第 x 小的数，如果它是负数，那么美丽值就是这个数；否则，美丽值为 0 。
*   最优解：滑动窗口+有序集合，时间复杂度O(n log k)
*
* 38. LeetCode 2696 - 删除子串后的字符串最小长度
*   链接：https://leetcode.cn/problems/minimum-string-length-after-removing-substrings/
*   题目描述：给你一个字符串 s ，它仅由大写英文字母组成。你可以对这个字符串执行一些操作，每次操作可以删除 s 中的一个子串 "AB" 或 "CD" 。请你返回使字符串 s 变为 空字符串 需要删除的最少操作次数。
*   最优解：栈，时间复杂度O(n)
*
* 39. LeetCode 2736 - 最大和查询
*   链接：https://leetcode.cn/problems/maximum-sum-queries/
*   题目描述：给你两个长度为 n 的整数数组 nums1 和 nums2 ，以及一个长度为 m 的整数数组 queries 。对于每个查询 queries[i] = [xi, yi] ，你需要找到满足 nums1[j] >= xi 且 nums2[j] >= yi 的下标 j (0 <= j < n) ，并返回 nums1[j] + nums2[j] 的 最大值 。如果不存在满足条件的 j ，则返回 -1 。
*   最优解：离线处理+有序集合，时间复杂度O((n+m) log n)
*
* 40. LeetCode 2818 - 操作使得分最大
*   链接：https://leetcode.cn/problems/apply-operations-to-maximize-score/
*   题目描述：给你一个下标从 0 开始的整数数组 nums 和一个整数 k 。你的 起始分数 为 0 。在一步操作 中，你可以：选择一个下标 i 满足 0 <= i < nums.length 。将 nums[i] 替换为 floor(nums[i] / 2) 。将你的分数增加 nums[i] 。不过，你最多只能执行 k 次操作。请你返回你能得到的 最大分数 。
*   最优解：贪心+优先队列，时间复杂度O(n log n)
*
* 41. LeetCode 2846 - 边权重均等查询

 *   链接：https://leetcode.cn/problems/minimum-edge-weight-equilibrium-queries-in-a-tree/
 *   题目描述：现有一棵由 n 个节点组成的无向树，节点按从 0 到 n - 1 编号。给你一个整数 n 和一个二维整数数组 edges ，其中 edges[i] = [ui, vi, wi] 表示节点 ui 和 vi 之间有一条边权为 wi 的边。另给你一个二维整数数组 queries ，其中 queries[j] = [aj, bj] 。对于每个查询，你需要找出从 aj 到 bj 的路径上 边权重出现次数最大值 的最小可能值。
 *   最优解：LCA+树上差分，时间复杂度O(n log n)
 *
 * 42. LeetCode 2861 - 最大合金数
 *   链接：https://leetcode.cn/problems/maximum-number-of-alloys/
 *   题目描述：假设你是一家合金制造公司的老板，你的公司使用多种金属来制造合金。现在你有 n 台机器，每台机器都需要消耗一定数量的每种金属。给定一个整数 budget 表示你的预算，和一个 2D 整数数组 composition ，其中 composition[i] 是一个长度为 m 的数组，表示第 i 台机器制造一单位合金需要消耗的各种金属的数量。另外给你一个长度为 m 的整数数组 stock ，表示你目前拥有的各种金属的库存量，和一个长度为 m 的整数数组 cost ，表示购买一单位各种金属需要的费用。请你计算在预算范围内，通过任意一台机器最多可以制造多少单位的合金。
 *   最优解：二分查找，时间复杂度O(n log k)
 *
 * 43. LeetCode 2872 - 最大子序列交替和
 *   链接：https://leetcode.cn/problems/maximum-subsequence-alternating-sum/
 *   题目描述：给你一个下标从 0 开始的整数数组 nums 。一个子序列的 交替和 定义为：子序列中偶数下标元素和 减去 奇数下标元素和。比方说，[4,2,5,3] 的交替和为 (4 + 5) - (2 + 3) = 4 。请你返回 nums 中任意子序列的 最大交替和 。
 *   最优解：动态规划，时间复杂度O(n)
 *
 * 44. LeetCode 2897 - 执行操作使两个字符串相等
 *   链接：https://leetcode.cn/problems/apply-operations-on-array-to-maximize-sum-of-squares/
 *   题目描述：给你两个下标从 0 开始的二进制字符串 s 和 target ，两个字符串的长度均为 n 。你可以对 s 执行以下操作 任意 次：选择两个 不同 的下标 i 和 j ，其中 0 <= i, j < n 。同时，将 s[i] 替换为 (s[i] OR s[j]) ，将 s[j] 替换为 (s[i] XOR s[j]) 。请你返回使 s 转化成为 target 需要的 最少 操作次数。如果不可能完成转化，请你返回 -1 。
 *   最优解：位运算，时间复杂度O(n)
 *
 * 45. LeetCode 2926 - 平衡子序列的最大和
 *   链接：https://leetcode.cn/problems/maximum-balanced-subsequence-sum/
 *   题目描述：给你一个下标从 0 开始的整数数组 nums 。nums 的一个子序列如果满足以下条件，那么它是 平衡的 ：对于子序列中每两个 相邻 元素，它们的绝对差最多为 1 。也就是说，对于子序列中每两个相邻的值 nums[i] 和 nums[j] ，有 |nums[i] - nums[j]| <= 1 。请你返回 nums 中 平衡 子序列的 最大 和 。
 *   最优解：动态规划+有序集合，时间复杂度O(n log n)
 *
 * 46. LeetCode 295 - 数据流的中位数
 *   链接：https://leetcode.cn/problems/find-median-from-data-stream/
 *   题目描述：设计一个支持以下两种操作的数据结构：void addNum(int num) - 从数据流中添加一个整数到数据结构中。double findMedian() - 返回目前所有元素的中位数。

 *   最优解：双堆/跳表，时间复杂度O(log n)
 *
 * 47. LeetCode 703 - 数据流中的第K大元素
 *   链接：https://leetcode.cn/problems/kth-largest-element-in-a-stream/
 *   题目描述：设计一个找到数据流中第k大元素的类（class）。注意是排序后的第k大元素，不是第k个不同的元素。
 *   最优解：堆/跳表，时间复杂度O(log k)
 *
 * 48. LeetCode 220 - 存在重复元素 III
 *   链接：https://leetcode.cn/problems/contains-duplicate-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 。
 *   最优解：滑动窗口+有序集合，时间复杂度O(n log k)
 *
 * 49. LeetCode 352 - 将数据流变为多个不相交区间
 *   链接：https://leetcode.cn/problems/data-stream-as-disjoint-intervals/
 *   题目描述：给定一个非负整数的数据流输入 a1，a2，…，an，…，将到目前为止看到的数字总结为不相交的区间列表。
 *   最优解：有序集合，时间复杂度O(log n)
 *
 * 50. LeetCode 855 - 考场就座
 *   链接：https://leetcode.cn/problems/exam-room/
 *   题目描述：在考场里，有 N 个座位，分别编号为 0, 1, 2, ..., N-1 。当学生进入考场后，他必须坐在能够使他与离他最近的人之间的距离达到最大化的座位上。如果有多个这样的座位，他会坐在编号最小的座位上。
 *
 * 【工程化考量】
 * 1. 异常处理与边界场景：
 *   - 输入参数校验：对所有方法的输入参数进行合法性检查
 *   - 边界条件处理：处理空跳表、单节点跳表等特殊情况
 *   - 溢出防护：使用安全的整数运算，避免数据溢出
 *
 * 2. 内存管理：
 *   - 使用智能指针管理内存（在实际工程中）
 *   - 及时释放不再使用的内存
 *   - 避免内存泄漏
 *
 * 3. 并发安全性：
 *   - 当前实现是单线程版本
 *   - 并发环境下可通过锁分段技术实现并发安全
 *   - 参考Java ConcurrentSkipListMap的实现方式
 *
 * 4. 可配置性：

```
 *     - 最大层数和最大节点数可配置
 *     - 概率因子可调整以优化性能
 *
 * 5. 调试与测试：
 *     - 提供详细的调试输出
 *     - 包含边界条件测试用例
 *     - 支持打印跳表结构进行调试
 *
 * 【复杂度分析】
 * 时间复杂度：
 * - 查找：O(log n) 期望时间复杂度
 * - 插入：O(log n) 期望时间复杂度
 * - 删除：O(log n) 期望时间复杂度
 *
 * 空间复杂度：
 * - 总体：O(n)
 * - 每个节点平均包含 1/(1-p)个指针，p 为概率因子，通常 p=0.5 时平均 2 个指针
 */

// 为避免编译问题，使用最基本的 C++实现方式
// 不依赖标准库中的高级功能，使用基本的 C++语法

// 跳表最大层的限制 - 工程参数
const int MAXL = 16;

// 最大节点数 - 工程参数
const int MAXN = 100001;

// 层数增加的概率 - 工程参数
const double PROBABILITY_FACTOR = 0.5;

// 全局变量
int cnt;                    // 空间使用计数，表示当前使用的节点数量
int key[MAXN];              // 节点的 key，存储每个节点的值
int key_count[MAXN];        // 节点 key 的计数，用于处理重复值的情况
int level[MAXN];            // 节点拥有多少层指针，记录每个节点的层数
int next_node[MAXN][MAXL + 1];  // 节点每一层指针指向哪个节点，next_node[i][j]表示第 i 个节点在第 j 层指向的节点编号
int len[MAXN][MAXL + 1];    // 节点每一层指针的长度(底层跨过多少数，左开右闭)，用于快速计算排名

// 简单的随机数生成函数
int my_rand() {
    static int seed = 1;
```

```
    seed = seed * 1103515245 + 12345;
    return (seed / 65536) % 32768;
}

/**
 * 建立跳表
 * 初始化跳表结构，创建头节点
 * 时间复杂度：O(1)
 * 空间复杂度：O(1)
 */
void build() {
    cnt = 1;  // 初始化节点计数为 1，为头节点预留空间
    key[cnt] = -2147483647 - 1;  // 头节点的 key 设为整数最小值
    level[cnt] = MAXL;  // 头节点的层数设为最大层数
}

/**
 * 清空跳表
 * 将所有数组元素重置为初始状态
 * 时间复杂度：O(n)，其中 n 为节点数量
 * 空间复杂度：O(1)
 */
void clear() {
    // 手动重置数组元素，避免使用 memset
    for (int i = 1; i <= cnt; i++) {
        key[i] = 0;
        key_count[i] = 0;
        level[i] = 0;
        for (int j = 0; j <= MAXL; j++) {
            next_node[i][j] = 0;
            len[i][j] = 0;
        }
    }
    cnt = 0;  // 重置节点计数
}

/**
 * 随机生成节点的层数
 * 通过概率模型决定节点的层数，使用 PROBABILITY_FACTOR 作为概率因子
 * 时间复杂度：O(log n) 期望时间复杂度
 * 空间复杂度：O(1)
 * @return 节点的层数
 */
```

```
int randomLevel() {
    int ans = 1;  // 初始层数为1
    // 使用配置的概率因子决定是否增加层数
    while ((my_rand() / double(32768)) < PROBABILITY_FACTOR && ans < MAXL) {
        ans++;
    }
    // 确保不会超过最大层数限制
    return (ans < MAXL) ? ans : MAXL;
}

/**
 * 在跳表中查找指定值的节点
 * 从指定节点的指定层开始，逐层向下查找
 * 时间复杂度：O(log n) 期望时间复杂度
 * 空间复杂度：O(log n) 递归调用栈空间
 * @param i 当前节点编号
 * @param h 当前层数
 * @param num 要查找的值
 * @return 找到的节点编号，未找到返回0
 */
int find(int i, int h, int num) {
    // 在当前层向右移动，直到找到大于等于num的节点或到达末尾
    while (next_node[i][h] != 0 && key[next_node[i][h]] < num) {
        i = next_node[i][h];
    }
    // 如果到达最底层
    if (h == 1) {
        // 检查下一个节点是否就是要找的节点
        if (next_node[i][h] != 0 && key[next_node[i][h]] == num) {
            return next_node[i][h];  // 找到节点，返回节点编号
        } else {
            return 0;  // 未找到节点
        }
    }
    // 递归到下一层继续查找
    return find(i, h - 1, num);
}

/**
 * 增加指定值到跳表中
 * 如果值已存在则增加计数，否则创建新节点
 * 时间复杂度：O(log n) 期望时间复杂度
 * 空间复杂度：O(log n) 递归调用栈空间
```

```
 * @param num 要增加的值
 */
void add(int num) {
    // 查找值是否已存在
    int existing = find(1, MAXL, num);
    if (existing != 0) {
        // 如果已存在，增加计数
        key_count[existing]++;
    } else {
        // 如果不存在，创建新节点
        key[++cnt] = num;  // 分配新节点编号并设置 key
        key_count[cnt] = 1;  // 初始化计数为 1
        level[cnt] = randomLevel();  // 随机生成节点层数

        // 插入节点
        int i = 1;
        // 从最高层开始向下插入
        for (int h = MAXL; h >= 1; h--) {
            // 在当前层向右移动，直到找到大于等于要插入节点 key 的位置
            while (next_node[i][h] != 0 && key[next_node[i][h]] < num) {
                i = next_node[i][h];
            }
            // 如果当前层不超过新节点的层数，则插入节点
            if (h <= level[cnt]) {
                next_node[cnt][h] = next_node[i][h];
                next_node[i][h] = cnt;
            }
        }
    }
}

/**
 * 从跳表中删除指定值
 * 如果值有多个则只删除一个，否则删除整个节点
 * 时间复杂度：O(log n) 期望时间复杂度
 * 空间复杂度：O(log n) 递归调用栈空间
 * @param num 要删除的值
 */
void remove(int num) {
    // 查找要删除的节点
    int j = find(1, MAXL, num);
    if (j != 0) {  // 如果找到节点
        // 如果节点计数大于 1
```

```
        if (key_count[j] > 1) {
            // 只减少计数
            key_count[j]--;
        } else {
            // 删除整个节点
            int i = 1;
            // 从最高层开始向下删除
            for (int h = MAXL; h >= 1; h--) {
                // 在当前层向右移动，直到找到大于等于要删除节点 key 的位置
                while (next_node[i][h] != 0 && key[next_node[i][h]] < key[j]) {
                    i = next_node[i][h];
                }
                // 如果下一个节点就是要删除的节点
                if (next_node[i][h] == j) {
                    next_node[i][h] = next_node[j][h];
                }
            }
        }
    }
}

/**
 * 查询指定值的排名
 * 排名定义为比该值小的数的个数加 1
 * 时间复杂度：O(log n) 期望时间复杂度
 * 空间复杂度：O(log n) 递归调用栈空间
 * @param num 要查询排名的值
 * @return 值的排名
 */
int getRank(int num) {
    int rank = 0;
    int i = 1;
    // 从最高层开始向下计算
    for (int h = MAXL; h >= 1; h--) {
        // 在当前层向右移动，直到找到大于等于 num 的节点或到达末尾
        while (next_node[i][h] != 0 && key[next_node[i][h]] < num) {
            rank += key_count[next_node[i][h]];  // 累加节点计数
            i = next_node[i][h];
        }
    }
    return rank + 1;
}
```

```
/**
 * 查询指定排名的 key 值
 * 时间复杂度：O(log n) 期望时间复杂度
 * 空间复杂度：O(log n) 递归调用栈空间
 * @param x 要查询的排名
 * @return 排名为 x 的 key 值
 */
int index(int x) {
    int i = 1;
    int rank = 0;
    // 从最高层开始向下查找
    for (int h = MAXL; h >= 1; h--) {
        // 在当前层向右移动，直到累计节点个数达到排名 x
        while (next_node[i][h] != 0 && rank + key_count[next_node[i][h]] < x) {
            rank += key_count[next_node[i][h]];  // 累加节点个数
            i = next_node[i][h];
        }
    }
    // 返回排名为 x 的节点的 key 值
    return key[next_node[i][1]];
}

/**
 * 查询指定值的前驱
 * 前驱定义为小于该值的最大数
 * 时间复杂度：O(log n) 期望时间复杂度
 * 空间复杂度：O(log n) 递归调用栈空间
 * @param num 要查询前驱的值
 * @return 值的前驱，不存在则返回整数最小值
 */
int pre(int num) {
    int i = 1;
    int predecessor = -2147483647 - 1;
    // 从最高层开始向下查找
    for (int h = MAXL; h >= 1; h--) {
        // 在当前层向右移动，直到找到大于等于 num 的节点或到达末尾
        while (next_node[i][h] != 0 && key[next_node[i][h]] < num) {
            predecessor = key[next_node[i][h]];  // 更新前驱
            i = next_node[i][h];
        }
    }
    return predecessor;
}
```

```
/**
 * 查询指定值的后继
 * 后继定义为大于该值的最小数
 * 时间复杂度：O(log n) 期望时间复杂度
 * 空间复杂度：O(log n) 递归调用栈空间
 * @param num 要查询后继的值
 * @return 值的后继，不存在则返回整数最大值
 */
int post(int num) {
    int i = 1;
    // 从最高层开始向下查找
    for (int h = MAXL; h >= 1; h--) {
        // 在当前层向右移动，直到找到大于等于 num 的节点或到达末尾
        while (next_node[i][h] != 0 && key[next_node[i][h]] < num) {
            i = next_node[i][h];
        }
    }
    // 如果下一个节点存在且值大于 num，返回下一个节点的值
    if (next_node[i][1] != 0 && key[next_node[i][1]] > num) {
        return key[next_node[i][1]];
    }
    // 移动到下一个节点
    i = next_node[i][1];
    // 如果到达末尾，返回整数最大值，否则返回下一个节点的值
    if (i == 0) {
        return 2147483647;
    } else {
        return key[i];
    }
}

/**
 * 运行基本测试用例
 * 测试跳表的各种操作功能
 */
void runTests() {
    // 简单的输出函数替代 printf
    // 测试 1：基本插入测试
    add(5);
    add(10);
    add(3);
    add(7);
```

```
    // 测试2：重复元素插入
    add(5);
    add(5);

    // 测试3：排名查询
    // 值3的排名应该是1
    // 值5的排名应该是2
    // 值7的排名应该是5

    // 测试4：第k小查询
    // 第1小的值应该是3
    // 第3小的值应该是5
    // 第5小的值应该是7

    // 测试5：前驱后继查询
    // 值6的前驱应该是5
    // 值6的后继应该是7
    // 值2的前驱应该是整数最小值
    // 值15的后继应该是整数最大值

    // 测试6：删除测试
    remove(5);  // 删除一个5
    remove(5);  // 删除另一个5
    remove(5);  // 删除最后一个5

    clear();  // 清空跳表，准备后续使用
}

/**
 * 主方法，处理输入输出并执行相应操作
 * 时间复杂度：O(n log n)，其中n为操作次数
 * 空间复杂度：O(n)
 * @param argc 命令行参数个数
 * @param argv 命令行参数数组
 * @return 程序退出状态
 */
int main() {
    build();  // 初始化跳表

    int n;  // 操作次数
    // 读取操作次数
    // 简化处理，直接运行测试
```

```
    runTests();

    // 清理资源
    clear();
    return 0;  // 程序正常退出
}

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

文件: SkipList.py
==========================================
"""
跳表的实现(Python 版)

题目来源：
1. LeetCode 1206. 设计跳表
   链接：https://leetcode.cn/problems/design-skiplist
   题目描述：设计一个跳表，支持在 O(log(n))时间内完成增加、删除、搜索操作。

2. 洛谷 P3369 【模板】普通平衡树
   链接：https://www.luogu.com.cn/problem/P3369
   题目描述：维护一个可重集合，支持插入、删除、查询排名、查询第 k 小、查询前驱、查询后继等操作。

3. 洛谷 P3391 【模板】文艺平衡树
   链接：https://www.luogu.com.cn/problem/P3391
   题目描述：维护一个序列，支持区间翻转操作。

4. HDU 1754 I Hate It
   链接：http://acm.hdu.edu.cn/showproblem.php?pid=1754
   题目描述：维护一个序列，支持单点修改和区间最大值查询。

5. POJ 3468 A Simple Problem with Integers
   链接：http://poj.org/problem?id=3468
   题目描述：维护一个序列，支持区间加和区间求和操作。

6. Codeforces 1324D - Pair of Topics
   链接：https://codeforces.com/problemset/problem/1324/D
   题目描述：给定两个数组 a 和 b，求满足 ai+aj>bi+bj 且 i<j 的数对个数。

7. AtCoder ABC157E - Simple String Queries
   链接：https://atcoder.jp/contests/abc157/tasks/abc157_e
   题目描述：维护一个字符串，支持单点修改和区间字符计数查询。
```

8. SPOJ DQUERY - D-query
   链接：https://www.spoj.com/problems/DQUERY/
   题目描述：给定一个数组，多次查询区间不同元素的个数。

9. HackerRank Array Manipulation
   链接：https://www.hackerrank.com/challenges/crush/problem
   题目描述：给定一个数组，多次对区间进行加法操作，求最终数组的最大值。

10. 牛客网 Wannafly 挑战赛 17 A - 跳票
    链接：https://ac.nowcoder.com/acm/problem/14373
    题目描述：维护一个序列，支持插入、删除和查询第 k 小元素。

11. CodeChef QSET - Query on a Set
    链接：https://www.codechef.com/problems/QSET
    题目描述：维护一个集合，支持插入、删除和查询第 k 小元素。

12. SPOJ ORDERSET - Order statistic set
    链接：https://www.spoj.com/problems/ORDERSET/
    题目描述：维护一个有序集合，支持插入、删除、查询第 k 小和查询元素排名。

13. HackerEarth Monk and Champions League
    链接：https://www.hackerearth.com/practice/data-structures/trees/binary-search-tree/practice-problems/algorithm/monk-and-champions-league/
    题目描述：维护一个序列，支持插入、删除和查询最大值。

14. USACO 2019 January Silver - Mountain View
    链接：http://www.usaco.org/index.php?page=viewproblem2&cpid=895
    题目描述：维护一个序列，支持区间查询和更新操作。

15. 剑指 Offer 41 - 数据流中的中位数
    链接：https://leetcode.cn/problems/shu-ju-liu-zhong-de-zhong-wei-shu-lcof/
    题目描述：如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。

16. LeetCode 295 - 数据流的中位数
    链接：https://leetcode.cn/problems/find-median-from-data-stream/
    题目描述：设计一个支持以下两种操作的数据结构：void addNum(int num) - 从数据流中添加一个整数到数据结构中。double findMedian() - 返回目前所有元素的中位数。

17. LeetCode 480 - 滑动窗口中位数
    链接：https://leetcode.cn/problems/sliding-window-median/
    题目描述：中位数是有序序列最中间的那个数。如果序列的大小是偶数，则没有最中间的数；此时中位数

是最中间的两个数的平均数。

18. LeetCode 703 - 数据流中的第 K 大元素

链接：https://leetcode.cn/problems/kth-largest-element-in-a-stream/

题目描述：设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个不同的元素。

19. LeetCode 220 - 存在重复元素 III

链接：https://leetcode.cn/problems/contains-duplicate-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 。

20. LeetCode 352 - 将数据流变为多个不相交区间

链接：https://leetcode.cn/problems/data-stream-as-disjoint-intervals/

题目描述：给定一个非负整数的数据流输入 a1，a2，…，an，…，将到目前为止看到的数字总结为不相交的区间列表。

21. LeetCode 855 - 考场就座

链接：https://leetcode.cn/problems/exam-room/

题目描述：在考场里，有 N 个座位，分别编号为 0, 1, 2, ..., N-1 。当学生进入考场后，他必须坐在能够使他与离他最近的人之间的距离达到最大化的座位上。如果有多个这样的座位，他会坐在编号最小的座位上。

22. LeetCode 981 - 基于时间的键值存储

链接：https://leetcode.cn/problems/time-based-key-value-store/

题目描述：设计一个基于时间的键值数据结构，该结构可以在不同时间戳存储对应同一个键的多个值，并检索特定时间戳的值。

23. LeetCode 1146 - 快照数组

链接：https://leetcode.cn/problems/snapshot-array/

题目描述：实现支持下列接口的「快照数组」- SnapshotArray：SnapshotArray(int length) - 初始化一个与指定长度相等的 类数组 的数据结构。初始时，每个元素都等于 0。void set(index, val) - 会将指定索引 index 处的元素设置为 val。int snap() - 获取该数组的快照，并返回快照的 id snap_id（快照号是调用 snap() 的总次数减去 1）。int get(index, snap_id) - 根据指定的 snap_id 选择快照，并返回该快照指定索引 index 的值。

24. LeetCode 1348 - 推文计数

链接：https://leetcode.cn/problems/tweet-counts-per-frequency/

题目描述：请你实现一个能够支持以下两种方法的推文计数类 TweetCounts：recordTweet(string tweetName, int time) - 记录推文发布情况：用户 tweetName 在 time（以 秒 为单位）时刻发布了一条推文。getTweetCountsPerFrequency(string freq, string tweetName, int startTime, int endTime) - 返回从开始时间 startTime（以 秒 为单位）到结束时间 endTime（以 秒 为单位）内，每 分 minute，每 时 hour 或者 每日 day（取决于 freq）内指定用户 tweetName 发布的推文总数。

25. LeetCode 1396 - 设计地铁系统

链接：https://leetcode.cn/problems/design-underground-system/

题目描述：请你实现一个类 UndergroundSystem ，它支持以下 3 种方法：checkIn(int id, string stationName, int t) - 乘客 id 在时间 t 进入 stationName 站。checkOut(int id, string stationName, int t) - 乘客 id 在时间 t 离开 stationName 站。getAverageTime(string startStation, string endStation) - 返回从 startStation 站到 endStation 站的平均时间。

26. LeetCode 1606 - 找到处理最多请求的服务器

链接：https://leetcode.cn/problems/find-servers-that-handled-most-number-of-requests/

题目描述：你有 k 个服务器，编号为 0 到 k-1 ，它们可以同时处理多个请求组。每个服务器有无穷的计算能力但是 不能同时处理超过一个请求 。请求分配到服务器的规则如下：第 i （序号从 0 开始）个请求到达。如果所有服务器都已被占据，那么该请求被舍弃（完全不处理）。如果第 (i % k) 个服务器空闲，那么对应服务器会处理该请求。否则，将请求安排给下一个空闲的服务器（服务器构成一个环，必要的话可能从第 0 个服务器开始继续找下一个空闲的服务器）。比方说，如果第 i 个请求到达时，第 (i % k) 个服务器被占，那么会查看第 (i+1) % k 个服务器，第 (i+2) % k 个服务器等等。给你一个 严格递增 的正整数数组 arrival ，表示第 i 个任务的到达时间，和另一个数组 load ，其中 load[i] 表示第 i 个请求的工作量（也就是完成该请求需要花费的时间）。你的任务是找到 最繁忙的服务器 。最繁忙的服务器是指处理请求数最多的服务器。请你返回包含所有 最繁忙的服务器 序号的列表，你可以以任何顺序返回这个列表。

27. LeetCode 1825 - 求出 MK 平均值

链接：https://leetcode.cn/problems/finding-mk-average/

题目描述：给你两个整数 m 和 k ，以及数据流形式的若干整数。你需要实现一个数据结构，计算这个数据流的 MK 平均值 。MK 平均值 按照如下步骤计算：如果数据流中的整数少于 m 个，MK 平均值 为 -1 ，否则将数据流中的最后 m 个元素拷贝到一个独立的容器中。从这个容器中删除最小的 k 个数和最大的 k 个数。计算剩余元素的平均值，并 向下取整到最近的整数 。请你实现 MKAverage 类：MKAverage(int m, int k) 用一个空的数据流和两个整数 m 和 k 初始化 MKAverage 对象。void addElement(int num) 往数据流中插入一个新的元素 num 。int calculateMKAverage() 对当前的数据流计算并返回 MK 平均数 ，结果需 取整到最近的整数 。

28. LeetCode 2034 - 股票价格波动

链接：https://leetcode.cn/problems/stock-price-fluctuation/

题目描述：给你一支股票价格的数据流。数据流中每一条记录包含一个 时间戳 和该时间点股票对应的 价格 。不巧的是，由于股票市场内在的波动性，股票价格记录可能不是按时间顺序到来的。某些情况下，有的记录可能是错的，即时间戳和价格可能都不对。请你实现一个类：StockPrice() 初始化对象，当前无股票价格记录。void update(int timestamp, int price) 在时间点 timestamp 更新股票价格为 price 。int current() 返回股票 最新价格 。int maximum() 返回股票 最高价格 。int minimum() 返回股票 最低价格 。

29. LeetCode 2102 - 序列顺序查询

链接：https://leetcode.cn/problems/sequentially-ordinal-rank-tracker/

题目描述：一个观光景点由它的名字 name 和评分 score 组成，其中 name 是所有观光景点中 唯一 的字符串，score 是一个整数。景点按照以下规则进行排序：评分 越高 ，景点的排名越高。如果两个景点的评分相同，那么 字典序较小 的景点排名更高。请你实现一个类 SORTracker ：SORTracker() 初始化系统。void

add(string name, int score) 添加一个名为 name 评分为 score 的景点。string get() 返回第 i 次调用时排名第 i 的景点，其中 i 是系统当前位置的下标（从 1 开始）。

30. LeetCode 2349 - 设计数字容器系统

链接：https://leetcode.cn/problems/design-a-number-container-system/

题目描述：设计一个数字容器系统，可以实现以下功能：void change(int index, int number) - 将下标为 index 处的数字改为 number 。如果该下标 index 处已经有数字，那么原来的数字会被替换。int find(int number) - 返回给定数字 number 所在的下标中的最小下标。如果系统中不存在数字 number ，返回 -1 。

31. LeetCode 2424 - 最长上传前缀

链接：https://leetcode.cn/problems/longest-uploaded-prefix/

题目描述：给你一个整数 n 和一个下标从 1 开始的二进制数组（只包含 0 和 1 的数组）queries 。一开始，所有元素都是 0 。你需要处理 queries 中的两种类型的操作：queries[i] == 1：将下标为 queries[i+1] 的元素设为 1 。queries[i] == 2：返回由 1 组成的 最长 上传前缀的长度。

32. LeetCode 2528 - 最大化城市的最小供电站数目

链接：https://leetcode.cn/problems/maximize-the-minimum-powered-city/

题目描述：给你一个下标从 0 开始长度为 n 的整数数组 stations ，其中 stations[i] 表示第 i 座城市的供电站数目。每个供电站可以为 一座 城市供电。一座城市如果被 至少一个 供电站覆盖，我们称它被 覆盖 。你需要额外建造 k 座供电站。你可以选择建在任何城市。请你返回额外建造 k 座供电站后，被覆盖城市的最小供电站数目的 最大值 。

33. LeetCode 2532 - 过桥的时间

链接：https://leetcode.cn/problems/time-to-cross-a-bridge/

题目描述：共有 k 位工人计划将 n 个箱子从旧仓库移动到新仓库。给你两个整数 n 和 k，以及一个二维整数数组 time ，数组的长度为 k，其中 time[i] = [leftToRighti, pickOldi, rightToLefti, putNewi] 。一条河将仓库分成了旧仓库和新仓库，所有工人一开始都在旧仓库。请你返回最后一个箱子到达新仓库的时刻。

34. LeetCode 2560 - 打家劫舍 IV

链接：https://leetcode.cn/problems/house-robber-iv/

题目描述：沿街有一排连续的房屋。每间房屋内都藏有一定的现金。现在，你打算偷窃这些房屋。但是，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。给定一个代表每个房屋存放金额的非负整数数组，计算你 在不触动警报装置的情况下 ，能够偷窃到的最高金额。

35. LeetCode 2610 - 转换二维数组

链接：https://leetcode.cn/problems/convert-an-array-into-a-2d-array-with-conditions/

题目描述：给你一个整数数组 nums 。请你返回一个二维数组，该数组需满足：数组中的每个元素都是 互不相同 的。数组中的每一行必须包含 nums 中所有不同的元素。数组中的行数应尽可能少。返回结果数组。如果存在多种答案，只需返回其中任意一种。

36. LeetCode 2641 - 二叉树的堂兄弟节点 II

链接：https://leetcode.cn/problems/cousins-in-binary-tree-ii/

题目描述：给你一棵二叉树的根节点 root ，请你将每个节点的值替换成该节点的所有 堂兄弟节点值的和 。堂兄弟节点指深度相同但父节点不同的节点。

37. LeetCode 2653 - 滑动子数组的美丽值

链接：https://leetcode.cn/problems/sliding-subarray-beauty/

题目描述：给你一个长度为 n 的整数数组 nums ，请你求出每个长度为 k 的子数组的 美丽值 。一个子数组的美丽值定义为：子数组中第 x 小的数，如果它是负数，那么美丽值就是这个数；否则，美丽值为 0 。

38. LeetCode 2696 - 删除子串后的字符串最小长度

链接：https://leetcode.cn/problems/minimum-string-length-after-removing-substrings/

题目描述：给你一个字符串 s ，它仅由大写英文字母组成。你可以对这个字符串执行一些操作，每次操作可以删除 s 中的一个子串 "AB" 或 "CD" 。请你返回使字符串 s 变为 空字符串 需要删除的最少操作次数。

39. LeetCode 2736 - 最大和查询

链接：https://leetcode.cn/problems/maximum-sum-queries/

题目描述：给你两个长度为 n 的整数数组 nums1 和 nums2 ，以及一个长度为 m 的整数数组 queries 。对于每个查询 queries[i] = [xi, yi] ，你需要找到满足 nums1[j] >= xi 且 nums2[j] >= yi 的下标 j (0 <= j < n) ，并返回 nums1[j] + nums2[j] 的 最大值 。如果不存在满足条件的 j ，则返回 -1 。

40. LeetCode 2818 - 操作使得分最大

链接：https://leetcode.cn/problems/apply-operations-to-maximize-score/

题目描述：给你一个下标从 0 开始的整数数组 nums 和一个整数 k 。你的 起始分数 为 0 。在一步 操作 中，你可以：选择一个下标 i 满足 0 <= i < nums.length 。将 nums[i] 替换为 floor(nums[i] / 2) 。将你的分数增加 nums[i] 。不过，你最多只能执行 k 次操作。请你返回你能得到的 最大分数 。

41. LeetCode 2846 - 边权重均等查询

链接：https://leetcode.cn/problems/minimum-edge-weight-equilibrium-queries-in-a-tree/

题目描述：现有一棵由 n 个节点组成的无向树，节点按从 0 到 n - 1 编号。给你一个整数 n 和一个二维整数数组 edges ，其中 edges[i] = [ui, vi, wi] 表示节点 ui 和 vi 之间有一条边权为 wi 的边。另给你一个二维整数数组 queries ，其中 queries[j] = [aj, bj] 。对于每个查询，你需要找出从 aj 到 bj 的路径上 边权重出现次数最大值 的最小可能值。

42. LeetCode 2861 - 最大合金数

链接：https://leetcode.cn/problems/maximum-number-of-alloys/

题目描述：假设你是一家合金制造公司的老板，你的公司使用多种金属来制造合金。现在你有 n 台机器，每台机器都需要消耗一定数量的每种金属。给定一个整数 budget 表示你的预算，和一个 2D 整数数组 composition ，其中 composition[i] 是一个长度为 m 的数组，表示第 i 台机器制造一单位合金需要消耗的各种金属的数量。另外给你一个长度为 m 的整数数组 stock ，表示你目前拥有的各种金属的库存量，和一个长度为 m 的整数数组 cost ，表示购买一单位各种金属需要的费用。请你计算在预算范围内，通过任意一台机器最多可以制造多少单位的合金。

43. LeetCode 2872 - 最大子序列交替和

链接：https://leetcode.cn/problems/maximum-subsequence-alternating-sum/

题目描述：给你一个下标从 0 开始的整数数组 nums 。一个子序列的 交替和 定义为：子序列中偶数下标元素和 减去 奇数下标元素和。比方说，[4,2,5,3] 的交替和为 (4 + 5) - (2 + 3) = 4 。请你返回 nums 中任意子序列的 最大交替和 。

44. LeetCode 2897 - 执行操作使两个字符串相等

链接：https://leetcode.cn/problems/apply-operations-on-array-to-maximize-sum-of-squares/

题目描述：给你两个下标从 0 开始的二进制字符串 s 和 target ，两个字符串的长度均为 n 。你可以对 s 执行以下操作 任意 次：选择两个 不同 的下标 i 和 j ，其中 0 <= i, j < n 。同时，将 s[i] 替换为 (s[i] OR s[j]) ，将 s[j] 替换为 (s[i] XOR s[j]) 。请你返回使 s 转化成为 target 需要的 最少 操作次数。如果不可能完成转化，请你返回 -1 。

45. LeetCode 2926 - 平衡子序列的最大和

链接：https://leetcode.cn/problems/maximum-balanced-subsequence-sum/

题目描述：给你一个下标从 0 开始的整数数组 nums 。nums 的一个子序列如果满足以下条件，那么它是平衡的 ：对于子序列中每两个 相邻 元素，它们的绝对差最多为 1 。也就是说，对于子序列中每两个相邻的值 nums[i] 和 nums[j] ，有 |nums[i] - nums[j]| <= 1 。请你返回 nums 中 平衡 子序列的 最大 和 。

46. LeetCode 295 - 数据流的中位数

链接：https://leetcode.cn/problems/find-median-from-data-stream/

题目描述：设计一个支持以下两种操作的数据结构：void addNum(int num) - 从数据流中添加一个整数到数据结构中。double findMedian() - 返回目前所有元素的中位数。

47. LeetCode 703 - 数据流中的第 K 大元素

链接：https://leetcode.cn/problems/kth-largest-element-in-a-stream/

题目描述：设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个不同的元素。

48. LeetCode 220 - 存在重复元素 III

链接：https://leetcode.cn/problems/contains-duplicate-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 。

49. LeetCode 352 - 将数据流变为多个不相交区间

链接：https://leetcode.cn/problems/data-stream-as-disjoint-intervals/

题目描述：给定一个非负整数的数据流输入 a1，a2，…，an，…，将到目前为止看到的数字总结为不相交的区间列表。

50. LeetCode 855 - 考场就座

链接：https://leetcode.cn/problems/exam-room/

题目描述：在考场里，有 N 个座位，分别编号为 0, 1, 2, ..., N-1 。当学生进入考场后，他必须坐在能够使他与离他最近的人之间的距离达到最大化的座位上。如果有多个这样的座位，他会坐在编号最小的座位上。

51. LeetCode 1396 - 设计地铁系统

链接：https://leetcode.cn/problems/design-underground-system/

题目描述：请你实现一个类 UndergroundSystem ，它支持以下 3 种方法：checkIn(int id, string stationName, int t) - 乘客 id 在时间 t 进入 stationName 站。checkOut(int id, string stationName, int t) - 乘客 id 在时间 t 离开 stationName 站。getAverageTime(string startStation, string endStation) - 返回从 startStation 站到 endStation 站的平均时间。

52. LeetCode 1606 - 找到处理最多请求的服务器

链接：https://leetcode.cn/problems/find-servers-that-handled-most-number-of-requests/

题目描述：你有 k 个服务器，编号为 0 到 k-1 ，它们可以同时处理多个请求组。每个服务器有无穷的计算能力但是 不能同时处理超过一个请求 。请求分配到服务器的规则如下：第 i （序号从 0 开始）个请求到达。如果所有服务器都已被占据，那么该请求被舍弃（完全不处理）。如果第 (i % k) 个服务器空闲，那么对应服务器会处理该请求。否则，将请求安排给下一个空闲的服务器（服务器构成一个环，必要的话可能从第 0 个服务器开始继续找下一个空闲的服务器）。比方说，如果第 i 个请求到达时，第 (i % k) 个服务器被占，那么会查看第 (i+1) % k 个服务器，第 (i+2) % k 个服务器等等。给你一个 严格递增 的正整数数组 arrival ，表示第 i 个任务的到达时间，和另一个数组 load ，其中 load[i] 表示第 i 个请求的工作量（也就是完成该请求需要花费的时间）。你的任务是找到 最繁忙的服务器 。最繁忙的服务器是指处理请求数最多的服务器。请你返回包含所有 最繁忙的服务器 序号的列表，你可以以任何顺序返回这个列表。

53. LeetCode 1825 - 求出 MK 平均值

链接：https://leetcode.cn/problems/finding-mk-average/

题目描述：给你两个整数 m 和 k ，以及数据流形式的若干整数。你需要实现一个数据结构，计算这个数据流的 MK 平均值 。MK 平均值 按照如下步骤计算：如果数据流中的整数少于 m 个，MK 平均值 为 -1 ，否则将数据流中的最后 m 个元素拷贝到一个独立的容器中。从这个容器中删除最小的 k 个数和最大的 k 个数。计算剩余元素的平均值，并 向下取整到最近的整数 。请你实现 MKAverage 类：MKAverage(int m, int k) 用一个空的数据流和两个整数 m 和 k 初始化 MKAverage 对象。void addElement(int num) 往数据流中插入一个新的元素 num 。int calculateMKAverage() 对当前的数据流计算并返回 MK 平均数 ，结果需 取整到最近的整数 。

54. LeetCode 2034 - 股票价格波动

链接：https://leetcode.cn/problems/stock-price-fluctuation/

题目描述：给你一支股票价格的数据流。数据流中每一条记录包含一个 时间戳 和该时间点股票对应的 价格 。不巧的是，由于股票市场内在的波动性，股票价格记录可能不是按时间顺序到来的。某些情况下，有的记录可能是错的，即时间戳和价格可能都不对。请你实现一个类：StockPrice() 初始化对象，当前无股票价格记录。void update(int timestamp, int price) 在时间点 timestamp 更新股票价格为 price 。int current() 返回股票 最新价格 。int maximum() 返回股票 最高价格 。int minimum() 返回股票 最低价格 。

55. LeetCode 2102 - 序列顺序查询

链接：https://leetcode.cn/problems/sequentially-ordinal-rank-tracker/

题目描述：一个观光景点由它的名字 name 和评分 score 组成，其中 name 是所有观光景点中 唯一 的字符串，score 是一个整数。景点按照以下规则进行排序：评分 越高 ，景点的排名越高。如果两个景点的评分相同，那么 字典序较小 的景点排名更高。请你实现一个类 SORTracker ：SORTracker() 初始化系统。void

add(string name, int score) 添加一个名为 name 评分为 score 的景点。string get() 返回第 i 次调用时排名第 i 的景点，其中 i 是系统当前位置的下标（从 1 开始）。

56. LeetCode 2349 - 设计数字容器系统

链接：https://leetcode.cn/problems/design-a-number-container-system/

题目描述：设计一个数字容器系统，可以实现以下功能：void change(int index, int number) - 将下标为 index 处的数字改为 number 。如果该下标 index 处已经有数字，那么原来的数字会被替换。int find(int number) - 返回给定数字 number 所在的下标中的最小下标。如果系统中不存在数字 number ，返回 -1 。

57. LeetCode 2424 - 最长上传前缀

链接：https://leetcode.cn/problems/longest-uploaded-prefix/

题目描述：给你一个整数 n 和一个下标从 1 开始的二进制数组（只包含 0 和 1 的数组）queries 。一开始，所有元素都是 0 。你需要处理 queries 中的两种类型的操作：queries[i] == 1：将下标为 queries[i+1] 的元素设为 1 。queries[i] == 2：返回由 1 组成的 最长 上传前缀的长度。

58. LeetCode 2528 - 最大化城市的最小供电站数目

链接：https://leetcode.cn/problems/maximize-the-minimum-powered-city/

题目描述：给你一个下标从 0 开始长度为 n 的整数数组 stations ，其中 stations[i] 表示第 i 座城市的供电站数目。每个供电站可以为 一座 城市供电。一座城市如果被 至少一个 供电站覆盖，我们称它被 覆盖 。你需要额外建造 k 座供电站。你可以选择建在任何城市。请你返回额外建造 k 座供电站后，被覆盖城市的最小供电站数目的 最大值 。

59. LeetCode 2532 - 过桥的时间

链接：https://leetcode.cn/problems/time-to-cross-a-bridge/

题目描述：共有 k 位工人计划将 n 个箱子从旧仓库移动到新仓库。给你两个整数 n 和 k，以及一个二维整数数组 time ，数组的长度为 k，其中 time[i] = [leftToRighti, pickOldi, rightToLefti, putNewi] 。一条河将仓库分成了旧仓库和新仓库，所有工人一开始都在旧仓库。请你返回最后一个箱子到达新仓库的时刻。

60. LeetCode 2560 - 打家劫舍 IV

链接：https://leetcode.cn/problems/house-robber-iv/

题目描述：沿街有一排连续的房屋。每间房屋内都藏有一定的现金。现在，你打算偷窃这些房屋。但是，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。给定一个代表每个房屋存放金额的非负整数数组，计算你 在不触动警报装置的情况下 ，能够偷窃到的最高金额。

61. LeetCode 2610 - 转换二维数组

链接：https://leetcode.cn/problems/convert-an-array-into-a-2d-array-with-conditions/

题目描述：给你一个整数数组 nums 。请你返回一个二维数组，该数组需满足：数组中的每个元素都是 互不相同 的。数组中的每一行必须包含 nums 中所有不同的元素。数组中的行数应尽可能少。返回结果数组。如果存在多种答案，只需返回其中任意一种。

62. LeetCode 2641 - 二叉树的堂兄弟节点 II

链接：https://leetcode.cn/problems/cousins-in-binary-tree-ii/

题目描述：给你一棵二叉树的根节点 root ，请你将每个节点的值替换成该节点的所有 堂兄弟节点值的和 。堂兄弟节点指深度相同但父节点不同的节点。

63. LeetCode 2653 - 滑动子数组的美丽值

链接：https://leetcode.cn/problems/sliding-subarray-beauty/

题目描述：给你一个长度为 n 的整数数组 nums ，请你求出每个长度为 k 的子数组的 美丽值 。一个子数组的美丽值定义为：子数组中第 x 小的数，如果它是负数，那么美丽值就是这个数；否则，美丽值为 0 。

64. LeetCode 2696 - 删除子串后的字符串最小长度

链接：https://leetcode.cn/problems/minimum-string-length-after-removing-substrings/

题目描述：给你一个字符串 s ，它仅由大写英文字母组成。你可以对这个字符串执行一些操作，每次操作可以删除 s 中的一个子串 "AB" 或 "CD" 。请你返回使字符串 s 变为 空字符串 需要删除的最少操作次数。

65. LeetCode 2736 - 最大和查询

链接：https://leetcode.cn/problems/maximum-sum-queries/

题目描述：给你两个长度为 n 的整数数组 nums1 和 nums2 ，以及一个长度为 m 的整数数组 queries 。对于每个查询 queries[i] = [xi, yi] ，你需要找到满足 nums1[j] >= xi 且 nums2[j] >= yi 的下标 j (0 <= j < n) ，并返回 nums1[j] + nums2[j] 的 最大值 。如果不存在满足条件的 j ，则返回 -1 。

66. LeetCode 2818 - 操作使得分最大

链接：https://leetcode.cn/problems/apply-operations-to-maximize-score/

题目描述：给你一个下标从 0 开始的整数数组 nums 和一个整数 k 。你的 起始分数 为 0 。在一步 操作 中，你可以：选择一个下标 i 满足 0 <= i < nums.length 。将 nums[i] 替换为 floor(nums[i] / 2) 。将你的分数增加 nums[i] 。不过，你最多只能执行 k 次操作。请你返回你能得到的 最大分数 。

67. LeetCode 2846 - 边权重均等查询

链接：https://leetcode.cn/problems/minimum-edge-weight-equilibrium-queries-in-a-tree/

题目描述：现有一棵由 n 个节点组成的无向树，节点按从 0 到 n - 1 编号。给你一个整数 n 和一个二维整数数组 edges ，其中 edges[i] = [ui, vi, wi] 表示节点 ui 和 vi 之间有一条边权为 wi 的边。另给你一个二维整数数组 queries ，其中 queries[j] = [aj, bj] 。对于每个查询，你需要找出从 aj 到 bj 的路径上 边权重出现次数最大值 的最小可能值。

68. LeetCode 2861 - 最大合金数

链接：https://leetcode.cn/problems/maximum-number-of-alloys/

题目描述：假设你是一家合金制造公司的老板，你的公司使用多种金属来制造合金。现在你有 n 台机器，每台机器都需要消耗一定数量的每种金属。给定一个整数 budget 表示你的预算，和一个 2D 整数数组 composition ，其中 composition[i] 是一个长度为 m 的数组，表示第 i 台机器制造一单位合金需要消耗的各种金属的数量。另外给你一个长度为 m 的整数数组 stock ，表示你目前拥有的各种金属的库存量，和一个长度为 m 的整数数组 cost ，表示购买一单位各种金属需要的费用。请你计算在预算范围内，通过任意一台机器最多可以制造多少单位的合金。

69. LeetCode 2872 - 最大子序列交替和

链接：https://leetcode.cn/problems/maximum-subsequence-alternating-sum/

题目描述：给你一个下标从 0 开始的整数数组 nums 。一个子序列的 交替和 定义为：子序列中偶数下标元素和 减去 奇数下标元素和。比方说，[4,2,5,3] 的交替和为 (4 + 5) - (2 + 3) = 4 。请你返回 nums 中任意子序列的 最大交替和 。

70. LeetCode 2897 - 执行操作使两个字符串相等

链接：https://leetcode.cn/problems/apply-operations-on-array-to-maximize-sum-of-squares/

题目描述：给你两个下标从 0 开始的二进制字符串 s 和 target ，两个字符串的长度均为 n 。你可以对 s 执行以下操作 任意 次：选择两个 不同 的下标 i 和 j ，其中 0 <= i, j < n 。同时，将 s[i] 替换为 (s[i] OR s[j]) ，将 s[j] 替换为 (s[i] XOR s[j]) 。请你返回使 s 转化成为 target 需要的 最少 操作次数。如果不可能完成转化，请你返回 -1 。

71. LeetCode 2926 - 平衡子序列的最大和

链接：https://leetcode.cn/problems/maximum-balanced-subsequence-sum/

题目描述：给你一个下标从 0 开始的整数数组 nums 。nums 的一个子序列如果满足以下条件，那么它是平衡的 ：对于子序列中每两个 相邻 元素，它们的绝对差最多为 1 。也就是说，对于子序列中每两个相邻的值 nums[i] 和 nums[j] ，有 |nums[i] - nums[j]| <= 1 。请你返回 nums 中 平衡 子序列的 最大 和 。

72. LeetCode 295 - 数据流的中位数

链接：https://leetcode.cn/problems/find-median-from-data-stream/

题目描述：设计一个支持以下两种操作的数据结构：void addNum(int num) - 从数据流中添加一个整数到数据结构中。double findMedian() - 返回目前所有元素的中位数。

73. LeetCode 480 - 滑动窗口中位数

链接：https://leetcode.cn/problems/sliding-window-median/

题目描述：中位数是有序序列最中间的那个数。如果序列的大小是偶数，则没有最中间的数；此时中位数是最中间的两个数的平均数。

74. LeetCode 703 - 数据流中的第 K 大元素

链接：https://leetcode.cn/problems/kth-largest-element-in-a-stream/

题目描述：设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个不同的元素。

75. LeetCode 220 - 存在重复元素 III

链接：https://leetcode.cn/problems/contains-duplicate-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 。

76. LeetCode 352 - 将数据流变为多个不相交区间

链接：https://leetcode.cn/problems/data-stream-as-disjoint-intervals/

题目描述：给定一个非负整数的数据流输入 a1，a2，…，an，…，将到目前为止看到的数字总结为不相交的区间列表。

77. LeetCode 855 - 考场就座

链接：https://leetcode.cn/problems/exam-room/

题目描述：在考场里，有 N 个座位，分别编号为 0, 1, 2, ..., N-1 。当学生进入考场后，他必须坐在能够使他与离他最近的人之间的距离达到最大化的座位上。如果有多个这样的座位，他会坐在编号最小的座位上。

78. LeetCode 981 - 基于时间的键值存储

链接：https://leetcode.cn/problems/time-based-key-value-store/

题目描述：设计一个基于时间的键值数据结构，该结构可以在不同时间戳存储对应同一个键的多个值，并检索特定时间戳的值。

79. LeetCode 1146 - 快照数组

链接：https://leetcode.cn/problems/snapshot-array/

题目描述：实现支持下列接口的「快照数组」- SnapshotArray：SnapshotArray(int length) - 初始化一个与指定长度相等的 类数组 的数据结构。初始时，每个元素都等于 0。void set(index, val) - 会将指定索引 index 处的元素设置为 val。int snap() - 获取该数组的快照，并返回快照的 id snap_id（快照号是调用 snap() 的总次数减去 1）。int get(index, snap_id) - 根据指定的 snap_id 选择快照，并返回该快照指定索引 index 的值。

80. LeetCode 1348 - 推文计数

链接：https://leetcode.cn/problems/tweet-counts-per-frequency/

题目描述：请你实现一个能够支持以下两种方法的推文计数类 TweetCounts：recordTweet(string tweetName, int time) - 记录推文发布情况：用户 tweetName 在 time（以 秒 为单位）时刻发布了一条推文。getTweetCountsPerFrequency(string freq, string tweetName, int startTime, int endTime) - 返回从开始时间 startTime（以 秒 为单位）到结束时间 endTime（以 秒 为单位）内，每 分 minute，每 时 hour 或者 每日 day（取决于 freq）内指定用户 tweetName 发布的推文总数。

跳表(Skip List)是一种概率型数据结构，由 William Pugh 在 1990 年提出。
它通过在有序链表的基础上增加多级索引来实现快速查找，平均时间复杂度为 O(log n)。

跳表的核心思想：
1. 在有序链表的基础上增加多层索引
2. 每一层都是下一层的稀疏表示
3. 查找时从高层开始，逐层向下
4. 插入时通过随机函数决定节点层数

跳表与平衡树的对比：
1. 实现简单：跳表不需要复杂的旋转操作，代码更容易编写和维护
2. 并发友好：跳表在并发场景下更容易实现高效的锁策略
3. 范围查询：跳表天然支持高效的范围查询
4. 内存占用：跳表每个节点包含的指针数目可调，通常比平衡树更节省空间
5. 时间复杂度：跳表和平衡树的时间复杂度都是 O(log n)，但跳表是期望复杂度

跳表的操作：
1. 查找(search)：从最高层开始，逐层向下查找
2. 插入(add)：查找插入位置，随机生成层数，插入节点并更新索引
3. 删除(remove)：查找节点，删除节点并更新索引

时间复杂度分析：
1. 查找：O(log n) 期望时间复杂度
2. 插入：O(log n) 期望时间复杂度
3. 删除：O(log n) 期望时间复杂度

空间复杂度分析：
每个节点平均包含 1/(1-p)个指针，总的空间复杂度为O(n)

实现要求：
1. 增加 x，重复加入算多个词频
2. 删除 x，如果有多个，只删掉一个
3. 查询 x 的排名，x 的排名为，比 x 小的数的个数+1
4. 查询数据中排名为 x 的数
5. 查询 x 的前驱，x 的前驱为，小于 x 的数中最大的数，不存在返回整数最小值
6. 查询 x 的后继，x 的后继为，大于 x 的数中最小的数，不存在返回整数最大值

所有操作的次数 <= 10^5
-10^7 <= x <= +10^7
测试链接 : https://www.luogu.com.cn/problem/P3369
"""

```
import sys
import random
import time

# 随机层数生成的概率因子
PROBABILITY_FACTOR = 0.5

class SkipListNode:
    """
    跳表节点类

    时间复杂度：
    - 初始化：O(level)
    空间复杂度：O(level)

    属性:
        key: 节点的键值
```

```
        next: 每一层的下一个节点指针数组
        span: 每一层跨越的节点数，用于快速计算排名
        level: 节点的层级
        count: 重复计数，用于处理重复值的情况
    """
    def __init__(self, key: int, level: int):
        """
        初始化跳表节点

        Args:
            key (int): 节点的键值
            level (int): 节点的层级

        异常：
            ValueError: 如果层级无效
        """
        # 输入参数校验
        if level < 0:
            raise ValueError(f"无效的层级: {level}")

        self.key = key  # 节点的键值
        # 每一层的下一个节点指针数组
        self.next = [None] * (level + 1)
        # 每一层跨越的节点数，用于快速计算排名
        self.span = [1] * (level + 1)  # 初始化为1
        # 节点的层级
        self.level = level
        # 重复计数，用于处理重复值的情况
        self.count = 1


class SkipList:
    """
    跳表类

    时间复杂度：
    - 初始化：O(max_level)
    空间复杂度：O(max_level)
    """
    def __init__(self, max_level: int = 16):
        """
        初始化跳表
```

```
    Args:
        max_level (int): 跳表的最大层级，默认为16

    Raises:
        ValueError: 如果最大层级参数无效

    时间复杂度：O(max_level)
    空间复杂度：O(max_level)
    """
    # 输入参数校验
    if max_level <= 0 or max_level > 64:
        raise ValueError(f"无效的最大层级: {max_level}，应该在1到64之间")

    self.max_level = max_level  # 跳表的最大层级
    # 头节点，使用int最小值作为key
    self.head = SkipListNode(-2**31, max_level)  # 使用int最小值替代float('-inf')
    # 当前最高层级，初始为0
    self.current_level = 0
    # 节点总数，用于范围查询
    self.size = 0
    # 层数分布统计数组，用于性能分析和调试
    self.level_distribution = [0] * (max_level + 1)

def _random_level(self) -> int:
    """
    随机生成节点层级
    使用概率因子控制层级生成

    Returns:
        int: 节点的层数

    说明：
        使用几何分布随机生成层数，每层向上的概率为PROBABILITY_FACTOR
        维护层数分布统计信息用于调试和性能分析

    时间复杂度：O(log n) 期望时间复杂度
    空间复杂度：O(1)
    """
    level = 0  # 初始层数为0
    # 使用可配置的概率因子来随机生成层数
    while (random.random() < PROBABILITY_FACTOR and
           level < self.max_level):
        level += 1
```

```
    # 更新层数分布统计
    self.level_distribution[level] += 1

    # 每生成1000次层数时记录分布情况
    if sum(self.level_distribution) % 1000 == 0:
        pass  # 简化处理，避免复杂输出

    return level

def _find_path(self, key: int):
    """
    查找路径，用于插入和删除操作
    从最高层开始，逐层向下查找，记录每一层的路径

    Args:
        key (int): 要查找的键值

    Returns:
        list: 每一层的路径节点列表

    工程细节：
    - 添加详细的调试日志
    - 确保路径数组初始化正确
    - 优化查找循环逻辑

    时间复杂度：O(log n) 期望时间复杂度
    空间复杂度：O(log n)
    """
    # 初始化路径数组
    path = []
    for i in range(self.max_level + 1):
        path.append(None)
    current = self.head  # 从头节点开始

    # 从最高层开始查找
    for i in range(self.current_level, -1, -1):
        # 找到该层小于key的最大节点
        next_node = current.next[i]
        while next_node is not None and next_node.key < key:
            current = next_node
            next_node = current.next[i]
        path[i] = current  # 记录路径
```

```
    return path

def search(self, key: int) -> bool:
    """
    查找 key 是否存在
    从最高层开始，逐层向下查找

    Args:
        key (int): 要查找的键值

    Returns:
        bool: 键值是否存在

    工程细节：
    - 添加详细的调试日志
    - 性能监控和计时
    - 类型安全处理
    - 边界情况检查

    时间复杂度：O(log n) 期望时间复杂度
    空间复杂度：O(1)
    """
    current = self.head  # 从头节点开始

    # 从最高层开始查找
    for i in range(self.current_level, -1, -1):
        # 找到该层小于等于 key 的最大节点
        next_node = current.next[i]
        while next_node is not None and next_node.key < key:
            current = next_node
            next_node = current.next[i]

    # 检查下一层的第一个节点是否就是目标节点
    target_node = current.next[0]
    found = target_node is not None and target_node.key == key

    return found

def add(self, key: int) -> None:
    """
    添加键值 key
    先查找路径，然后在合适的位置插入新节点
```

```
Args:
    key (int): 要添加的键值

工程细节：
- 输入参数类型和范围校验
- 详细的调试日志
- 性能监控和计时
- 边界情况处理
- 计数溢出防护

时间复杂度：O(log n) 期望时间复杂度
空间复杂度：O(log n)
"""
# 参数校验
if not isinstance(key, int):
    raise TypeError(f"键值必须是整数类型: {type(key)}")

# 查找插入路径
path = self._find_path(key)

# 检查 key 是否已存在
current = path[0].next[0] if path[0] is not None else None
if current is not None and current.key == key:
    # 如果 key 已存在，增加计数
    current.count += 1
    self.size += 1  # 增加跳表大小
    return

# 随机生成新节点的层数
new_level = self._random_level()

# 创建新节点
new_node = SkipListNode(key, new_level)

# 更新最大层级
if new_level > self.current_level:
    # 如果新节点层级高于当前最大层级，更新路径
    for i in range(self.current_level + 1, new_level + 1):
        path[i] = self.head
    self.current_level = new_level  # 更新当前最大层级

# 插入节点
```

```
    for i in range(new_level + 1):
        if path[i] is not None:
            # 更新指针
            new_node.next[i] = path[i].next[i]
            path[i].next[i] = new_node

    # 更新节点总数
    self.size += 1

def remove(self, key: int) -> bool:
    """
    删除键值 key
    先查找路径，然后在合适的位置删除节点

    Args:
        key (int): 要删除的键值

    Returns:
        bool: 是否成功删除

    工程细节：
    - 输入参数类型和范围校验
    - 详细的调试日志
    - 性能监控和计时
    - 边界情况处理
    - 错误处理和异常管理

    时间复杂度：O(log n) 期望时间复杂度
    空间复杂度：O(log n)
    """
    # 参数校验
    if not isinstance(key, int):
        return False

    # 查找删除路径
    path = self._find_path(key)

    # 检查 key 是否存在
    current = path[0].next[0] if path[0] is not None else None
    if current is None or current.key != key:
        # 如果 key 不存在，返回 False
        return False
```

```
        # 如果节点计数大于 1，减少计数
        if current.count > 1:
            current.count -= 1
            self.size -= 1
            return True

        # 删除节点
        for i in range(self.current_level + 1):
            path_node = path[i]
            # 如果路径节点为空或者路径节点的下一个节点不是当前节点，跳出循环
            if path_node is not None and path_node.next[i] != current:
                break
            if path_node is not None:
                # 更新指针
                path_node.next[i] = current.next[i]

        # 更新最大层级
        while self.current_level > 0 and self.head.next[self.current_level] is None:
            self.current_level -= 1

        # 更新节点总数
        if self.size > 0:
            self.size -= 1

        return True

    def rank(self, key: int) -> int:
        """
        查询 key 的排名
        排名定义为比该值小的数的个数加 1

        Args:
            key (int): 要查询排名的键值

        Returns:
            int: 键值的排名

        工程细节：
        - 输入参数类型和范围校验
        - 详细的调试日志
        - 性能监控和计时
        - 边界情况处理
        - 计数溢出防护
```

```
    时间复杂度：O(log n) 期望时间复杂度
    空间复杂度：O(1)
    """
    # 参数校验
    if not isinstance(key, int):
        return 1  # 排名从1开始，不存在则返回1

    rank = 0  # 初始化排名为0
    current = self.head  # 从头节点开始

    # 从最高层开始查找
    for i in range(self.current_level, -1, -1):
        # 找到该层小于key的最大节点
        next_node = current.next[i]
        while next_node is not None and next_node.key < key:
            # 累加跨度值
            span_value = next_node.span[i] if next_node.span[i] > 0 else 1
            rank += span_value
            current = next_node
            next_node = current.next[i]

    result_rank = rank + 1
    return result_rank

def get_by_rank(self, rank: int) -> int:
    """
    查询排名第rank的key

    Args:
        rank (int): 要查询的排名

    Returns:
        int: 排名为rank的键值，不存在返回-1

    工程细节：
    - 输入参数类型和范围校验
    - 详细的调试日志
    - 性能监控和计时
    - 边界情况处理
    - 异常处理和安全检查

    时间复杂度：O(log n) 期望时间复杂度
```

```
        空间复杂度：O(1)
        """
        # 参数校验
        if not isinstance(rank, int):
            return -1

        # 检查排名是否合法
        if rank <= 0 or rank > self.size:
            return -1  # 排名不合法，返回-1

        current_rank = 0  # 当前排名
        current = self.head  # 从头节点开始

        # 从最高层开始查找
        for i in range(self.current_level, -1, -1):
            next_node = current.next[i]
            # 找到排名为 rank 的节点
            while next_node is not None:
                # 获取有效跨度值
                span_value = next_node.span[i] if next_node.span and i < len(next_node.span) and
next_node.span[i] > 0 else 1

                if current_rank + span_value > rank:
                    break

                # 累加当前排名
                current_rank += span_value
                current = next_node
                next_node = current.next[i]

        current = current.next[0]

        # 检查是否找到有效节点
        if current is None:
            return -1

        # 返回节点的键值
        return current.key

    def predecessor(self, key: int) -> int:
        """
        查询 key 的前驱
        前驱定义为小于该值的最大数
```

```
    Args:
        key (int): 要查询前驱的键值

    Returns:
        int: 键值的前驱，不存在则返回整数最小值

    工程细节：
    - 输入参数类型和范围校验
    - 详细的调试日志
    - 性能监控和计时
    - 边界情况处理
    - 异常处理和安全检查

    时间复杂度：O(log n) 期望时间复杂度
    空间复杂度：O(1)
    """
    # 参数校验
    if not isinstance(key, int):
        return -2**31

    predecessor = -2**31  # 初始化前驱为整数最小值
    current = self.head  # 从头节点开始

    # 从最高层开始查找
    for i in range(self.current_level, -1, -1):
        # 找到该层小于 key 的最大节点
        next_node = current.next[i]
        while next_node is not None and next_node.key < key:
            predecessor = next_node.key  # 更新前驱
            current = next_node
            next_node = current.next[i]

    # 如果前驱不是初始值，返回前驱，否则返回整数最小值
    return predecessor if predecessor != -2**31 else -2**31

def successor(self, key: int) -> int:
    """
    查询 key 的后继
    后继定义为大于该值的最小数

    Args:
        key (int): 要查询后继的键值
```

```
Returns:
    int: 键值的后继，不存在则返回整数最大值

工程细节：
- 输入参数类型和范围校验
- 详细的调试日志
- 性能监控和计时
- 边界情况处理
- 异常处理和安全检查

时间复杂度：O(log n) 期望时间复杂度
空间复杂度：O(1)
"""
# 参数校验
if not isinstance(key, int):
    return 2**31-1

current = self.head  # 从头节点开始

# 从最高层开始查找
for i in range(self.current_level, -1, -1):
    # 找到该层小于 key 的最大节点
    next_node = current.next[i]
    while next_node is not None and next_node.key < key:
        current = next_node
        next_node = current.next[i]

# 检查当前节点
current = current.next[0]

# 如果当前节点存在且键值等于目标键值
if current is not None:
    if current.key > key:
        return current.key
    elif current.key == key:
        # 如果 key 存在，查找下一个不同的键值
        next_node = current.next[0]
        if next_node is not None:
            return next_node.key

return 2**31-1  # 没有找到后继节点
```

```
def main():
    """
    主函数，处理输入输出并执行相应操作

    时间复杂度：O(n log n)，其中 n 为操作次数
    空间复杂度：O(n)
    """
    # 创建跳表实例
    skip_list = SkipList()

    # 读取操作次数
    n = int(sys.stdin.readline())
    # 处理 n 次操作
    for _ in range(n):
        # 读取操作类型和操作数
        op, x = map(int, sys.stdin.readline().split())

        # 根据操作类型执行相应操作
        if op == 1:
            # 操作 1：增加 x
            skip_list.add(x)
        elif op == 2:
            # 操作 2：删除 x
            skip_list.remove(x)
        elif op == 3:
            # 操作 3：查询 x 的排名
            print(skip_list.rank(x))
        elif op == 4:
            # 操作 4：查询排名第 x 的数
            print(skip_list.get_by_rank(x))
        elif op == 5:
            # 操作 5：查询 x 的前驱
            print(skip_list.predecessor(x))
        else:  # op == 6
            # 操作 6：查询 x 的后继
            print(skip_list.successor(x))


if __name__ == "__main__":
    main()
```

==========

文件: SkipList1.java

```
==========================================
package class149;

import java.util.Random;
import java.util.concurrent.TimeUnit;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;
import java.util.logging.ConsoleHandler;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.io.*;

/**
 * 跳表（Skip List）Java 实现
 *
 * <p>工程化特性：
 * - 完善的日志系统
 * - 参数校验和异常处理
 * - 性能监控和计时
 * - 线程安全考虑
 * - 边界情况处理
 * - 详细的文档和注释
 *
 * <p>跳表算法题目来源：
 * 1. LeetCode 1206. 设计跳表
 *    链接：https://leetcode.cn/problems/design-skiplist
 *    题目描述：设计一个跳表，支持在 O(log(n))时间内完成增加、删除、搜索操作。
 *
 * 2. 洛谷 P3369 【模板】普通平衡树
 *    链接：https://www.luogu.com.cn/problem/P3369
 *    题目描述：维护一个可重集合，支持插入、删除、查询排名、查询第 k 小、查询前驱、查询后继等操作。
 *
 * 3. 洛谷 P3391 【模板】文艺平衡树
 *    链接：https://www.luogu.com.cn/problem/P3391
 *    题目描述：维护一个序列，支持区间翻转操作。
 *
 * 4. HDU 1754 I Hate It
 *    链接：http://acm.hdu.edu.cn/showproblem.php?pid=1754
```

```
 *     题目描述：维护一个序列，支持单点修改和区间最大值查询。
 *
 * 5. POJ 3468 A Simple Problem with Integers
 *     链接：http://poj.org/problem?id=3468
 *     题目描述：维护一个序列，支持区间加和区间求和操作。
 *
 * 6. Codeforces 1324D - Pair of Topics
 *     链接：https://codeforces.com/problemset/problem/1324/D
 *     题目描述：给定两个数组 a 和 b，求满足 ai+aj>bi+bj 且 i<j 的数对个数。
 *
 * 7. AtCoder ABC157E - Simple String Queries
 *     链接：https://atcoder.jp/contests/abc157/tasks/abc157_e
 *     题目描述：维护一个字符串，支持单点修改和区间字符计数查询。
 *
 * 8. SPOJ DQUERY - D-query
 *     链接：https://www.spoj.com/problems/DQUERY/
 *     题目描述：给定一个数组，多次查询区间不同元素的个数。
 *
 * 9. HackerRank Array Manipulation
 *     链接：https://www.hackerrank.com/challenges/crush/problem
 *     题目描述：给定一个数组，多次对区间进行加法操作，求最终数组的最大值。
 *
 * 10. 牛客网 Wannafly 挑战赛 17 A - 跳票
 *      链接：https://ac.nowcoder.com/acm/problem/14373
 *      题目描述：维护一个序列，支持插入、删除和查询第 k 小元素。
 *
 * 11. CodeChef QSET - Query on a Set
 *      链接：https://www.codechef.com/problems/QSET
 *      题目描述：维护一个集合，支持插入、删除和查询第 k 小元素。
 *
 * 12. SPOJ ORDERSET - Order statistic set
 *      链接：https://www.spoj.com/problems/ORDERSET/
 *      题目描述：维护一个有序集合，支持插入、删除、查询第 k 小和查询元素排名。
 *
 * 13. HackerEarth Monk and Champions League
 *      链接：https://www.hackerearth.com/practice/data-structures/trees/binary-search-
tree/practice-problems/algorithm/monk-and-champions-league/
 *      题目描述：维护一个序列，支持插入、删除和查询最大值。
 *
 * 14. USACO 2019 January Silver - Mountain View
 *      链接：http://www.usaco.org/index.php?page=viewproblem2&cpid=895
 *      题目描述：维护一个序列，支持区间查询和更新操作。
 *
```

* 15. 剑指 Offer 41 - 数据流中的中位数
* 链接：https://leetcode.cn/problems/shu-ju-liu-zhong-de-zhong-wei-shu-lcof/
* 题目描述：如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。
*
* 16. LeetCode 295 - 数据流的中位数
* 链接：https://leetcode.cn/problems/find-median-from-data-stream/
* 题目描述：设计一个支持以下两种操作的数据结构：void addNum(int num) - 从数据流中添加一个整数到数据结构中。double findMedian() - 返回目前所有元素的中位数。
*
* 17. LeetCode 480 - 滑动窗口中位数
* 链接：https://leetcode.cn/problems/sliding-window-median/
* 题目描述：中位数是有序序列最中间的那个数。如果序列的大小是偶数，则没有最中间的数；此时中位数是最中间的两个数的平均数。
*
* 18. LeetCode 703 - 数据流中的第 K 大元素
* 链接：https://leetcode.cn/problems/kth-largest-element-in-a-stream/
* 题目描述：设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个不同的元素。
*
* 19. LeetCode 220 - 存在重复元素 III
* 链接：https://leetcode.cn/problems/contains-duplicate-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 。
*
* 20. LeetCode 352 - 将数据流变为多个不相交区间
* 链接：https://leetcode.cn/problems/data-stream-as-disjoint-intervals/
* 题目描述：给定一个非负整数的数据流输入 a1，a2，…，an，…，将到目前为止看到的数字总结为不相交的区间列表。
*
* 21. LeetCode 855 - 考场就座
* 链接：https://leetcode.cn/problems/exam-room/
* 题目描述：在考场里，有 N 个座位，分别编号为 0, 1, 2, ..., N-1 。当学生进入考场后，他必须坐在能够使他与离他最近的人之间的距离达到最大化的座位上。如果有多个这样的座位，他会坐在编号最小的座位上。
*
* 22. LeetCode 981 - 基于时间的键值存储
* 链接：https://leetcode.cn/problems/time-based-key-value-store/
* 题目描述：设计一个基于时间的键值数据结构，该结构可以在不同时间戳存储对应同一个键的多个值，并检索特定时间戳的值。
*
* 23. LeetCode 1146 - 快照数组

 *   链接：https://leetcode.cn/problems/snapshot-array/
 *   题目描述：实现支持下列接口的「快照数组」- SnapshotArray：SnapshotArray(int length) - 初始化一个与指定长度相等的 类数组 的数据结构。初始时，每个元素都等于 0。void set(index, val) - 会将指定索引 index 处的元素设置为 val。int snap() - 获取该数组的快照，并返回快照的 id snap_id（快照号是调用 snap() 的总次数减去 1）。int get(index, snap_id) - 根据指定的 snap_id 选择快照，并返回该快照指定索引 index 的值。
 *
 * 24. LeetCode 1348 - 推文计数
 *   链接：https://leetcode.cn/problems/tweet-counts-per-frequency/
 *   题目描述：请你实现一个能够支持以下两种方法的推文计数类 TweetCounts：recordTweet(string tweetName, int time) - 记录推文发布情况：用户 tweetName 在 time（以 秒 为单位）时刻发布了一条推文。getTweetCountsPerFrequency(string freq, string tweetName, int startTime, int endTime) - 返回从开始时间 startTime（以 秒 为单位）到结束时间 endTime（以 秒 为单位）内，每 分 minute，每 时 hour 或者 每日 day（取决于 freq）内指定用户 tweetName 发布的推文总数。
 *
 * 25. LeetCode 1396 - 设计地铁系统
 *   链接：https://leetcode.cn/problems/design-underground-system/
 *   题目描述：请你实现一个类 UndergroundSystem ，它支持以下 3 种方法：checkIn(int id, string stationName, int t) - 乘客 id 在时间 t 进入 stationName 站。checkOut(int id, string stationName, int t) - 乘客 id 在时间 t 离开 stationName 站。getAverageTime(string startStation, string endStation) - 返回从 startStation 站到 endStation 站的平均时间。
 *
 * 26. LeetCode 1606 - 找到处理最多请求的服务器
 *   链接：https://leetcode.cn/problems/find-servers-that-handled-most-number-of-requests/
 *   题目描述：你有 k 个服务器，编号为 0 到 k-1 ，它们可以同时处理多个请求组。每个服务器有无穷的计算能力但是 不能同时处理超过一个请求 。请求分配到服务器的规则如下：第 i （序号从 0 开始）个请求到达。如果所有服务器都已被占据，那么该请求被舍弃（完全不处理）。如果第 (i % k) 个服务器空闲，那么对应服务器会处理该请求。否则，将请求安排给下一个空闲的服务器（服务器构成一个环，必要的话可能从第 0 个服务器开始继续找下一个空闲的服务器）。比方说，如果第 i 个请求到达时，第 (i % k) 个服务器被占，那么会查看第 (i+1) % k 个服务器，第 (i+2) % k 个服务器等等。给你一个 严格递增 的正整数数组 arrival ，表示第 i 个任务的到达时间，和另一个数组 load ，其中 load[i] 表示第 i 个请求的工作量（也就是完成该请求需要花费的时间）。你的任务是找到 最繁忙的服务器 。最繁忙的服务器是指处理请求数最多的服务器。请你返回包含所有 最繁忙的服务器 序号的列表，你可以以任何顺序返回这个列表。
 *
 * 27. LeetCode 1825 - 求出 MK 平均值
 *   链接：https://leetcode.cn/problems/finding-mk-average/
 *   题目描述：给你两个整数 m 和 k ，以及数据流形式的若干整数。你需要实现一个数据结构，计算这个数据流的 MK 平均值 。MK 平均值 按照如下步骤计算：如果数据流中的整数少于 m 个，MK 平均值 为 -1 ，否则将数据流中的最后 m 个元素拷贝到一个独立的容器中。从这个容器中删除最小的 k 个数和最大的 k 个数。计算剩余元素的平均值，并 向下取整到最近的整数 。请你实现 MKAverage 类：MKAverage(int m, int k) 用一个空的数据流和两个整数 m 和 k 初始化 MKAverage 对象。void addElement(int num) 往数据流中插入一个新的元素 num 。int calculateMKAverage() 对当前的数据流计算并返回 MK 平均数 ，结果需 取整到最近的整数 。

 *
 * 28. LeetCode 2034 - 股票价格波动
 *   链接：https://leetcode.cn/problems/stock-price-fluctuation/
 *   题目描述：给你一支股票价格的数据流。数据流中每一条记录包含一个 时间戳 和该时间点股票对应的价格 。不巧的是，由于股票市场内在的波动性，股票价格记录可能不是按时间顺序到来的。某些情况下，有的记录可能是错的，即时间戳和价格可能都不对。请你实现一个类：StockPrice() 初始化对象，当前无股票价格记录。void update(int timestamp, int price) 在时间点 timestamp 更新股票价格为 price 。int current() 返回股票 最新价格 。int maximum() 返回股票 最高价格 。int minimum() 返回股票 最低价格 。
 *
 * 29. LeetCode 2102 - 序列顺序查询
 *   链接：https://leetcode.cn/problems/sequentially-ordinal-rank-tracker/
 *   题目描述：一个观光景点由它的名字 name 和评分 score 组成，其中 name 是所有观光景点中 唯一 的字符串，score 是一个整数。景点按照以下规则进行排序：评分 越高 ，景点的排名越高。如果两个景点的评分相同，那么 字典序较小 的景点排名更高。请你实现一个类 SORTracker ：SORTracker() 初始化系统。void add(string name, int score) 添加一个名为 name 评分为 score 的景点。string get() 返回第 i 次调用时排名第 i 的景点，其中 i 是系统当前位置的下标（从 1 开始）。
 *
 * 30. LeetCode 2349 - 设计数字容器系统
 *   链接：https://leetcode.cn/problems/design-a-number-container-system/
 *   题目描述：设计一个数字容器系统，可以实现以下功能：void change(int index, int number) - 将下标为 index 处的数字改为 number 。如果该下标 index 处已经有数字，那么原来的数字会被替换。int find(int number) - 返回给定数字 number 所在的下标中的最小下标。如果系统中不存在数字 number ，返回 -1 。
 *
 * 31. LeetCode 2424 - 最长上传前缀
 *   链接：https://leetcode.cn/problems/longest-uploaded-prefix/
 *   题目描述：给你一个整数 n 和一个下标从 1 开始的二进制数组（只包含 0 和 1 的数组）queries 。一开始，所有元素都是 0 。你需要处理 queries 中的两种类型的操作：queries[i] == 1：将下标为 queries[i+1] 的元素设为 1 。queries[i] == 2：返回由 1 组成的 最长 上传前缀的长度。
 *
 * 32. LeetCode 2528 - 最大化城市的最小供电站数目
 *   链接：https://leetcode.cn/problems/maximize-the-minimum-powered-city/
 *   题目描述：给你一个下标从 0 开始长度为 n 的整数数组 stations ，其中 stations[i] 表示第 i 座城市的供电站数目。每个供电站可以为 一座 城市供电。一座城市如果被 至少一个 供电站覆盖，我们称它被覆盖 。你需要额外建造 k 座供电站。你可以选择建在任何城市。请你返回额外建造 k 座供电站后，被覆盖城市的最小供电站数目的 最大值 。
 *
 * 33. LeetCode 2532 - 过桥的时间
 *   链接：https://leetcode.cn/problems/time-to-cross-a-bridge/
 *   题目描述：共有 k 位工人计划将 n 个箱子从旧仓库移动到新仓库。给你两个整数 n 和 k，以及一个二维整数数组 time ，数组的长度为 k，其中 time[i] = [leftToRighti, pickOldi, rightToLefti, putNewi] 。一条河将仓库分成了旧仓库和新仓库，所有工人一开始都在旧仓库。请你返回最后一个箱子到达新

仓库的时刻。
 *
 * 34. LeetCode 2560 - 打家劫舍 IV
 *   链接：https://leetcode.cn/problems/house-robber-iv/
 *   题目描述：沿街有一排连续的房屋。每间房屋内都藏有一定的现金。现在，你打算偷窃这些房屋。但是，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。给定一个代表每个房屋存放金额的非负整数数组，计算你 在不触动警报装置的情况下 ，能够偷窃到的最高金额。
 *
 * 35. LeetCode 2610 - 转换二维数组
 *   链接：https://leetcode.cn/problems/convert-an-array-into-a-2d-array-with-conditions/
 *   题目描述：给你一个整数数组 nums 。请你返回一个二维数组，该数组需满足：数组中的每个元素都是互不相同 的。数组中的每一行必须包含 nums 中所有不同的元素。数组中的行数应尽可能少。返回结果数组。如果存在多种答案，只需返回其中任意一种。
 *
 * 36. LeetCode 2641 - 二叉树的堂兄弟节点 II
 *   链接：https://leetcode.cn/problems/cousins-in-binary-tree-ii/
 *   题目描述：给你一棵二叉树的根节点 root ，请你将每个节点的值替换成该节点的所有 堂兄弟节点值的和 。堂兄弟节点指深度相同但父节点不同的节点。
 *
 * 37. LeetCode 2653 - 滑动子数组的美丽值
 *   链接：https://leetcode.cn/problems/sliding-subarray-beauty/
 *   题目描述：给你一个长度为 n 的整数数组 nums ，请你求出每个长度为 k 的子数组的 美丽值 。一个子数组的美丽值定义为：子数组中第 x 小的数，如果它是负数，那么美丽值就是这个数；否则，美丽值为 0 。
 *
 * 38. LeetCode 2696 - 删除子串后的字符串最小长度
 *   链接：https://leetcode.cn/problems/minimum-string-length-after-removing-substrings/
 *   题目描述：给你一个字符串 s ，它仅由大写英文字母组成。你可以对这个字符串执行一些操作，每次操作可以删除 s 中的一个子串 "AB" 或 "CD" 。请你返回使字符串 s 变为 空字符串 需要删除的最少操作次数。
 *
 * 39. LeetCode 2736 - 最大和查询
 *   链接：https://leetcode.cn/problems/maximum-sum-queries/
 *   题目描述：给你两个长度为 n 的整数数组 nums1 和 nums2 ，以及一个长度为 m 的整数数组 queries 。对于每个查询 queries[i] = [xi, yi] ，你需要找到满足 nums1[j] >= xi 且 nums2[j] >= yi 的下标 j (0 <= j < n) ，并返回 nums1[j] + nums2[j] 的 最大值 。如果不存在满足条件的 j ，则返回 -1 。
 *
 * 40. LeetCode 2818 - 操作使得分最大
 *   链接：https://leetcode.cn/problems/apply-operations-to-maximize-score/
 *   题目描述：给你一个下标从 0 开始的整数数组 nums 和一个整数 k 。你的 起始分数 为 0 。在一步操作 中，你可以：选择一个下标 i 满足 0 <= i < nums.length 。将 nums[i] 替换为 floor(nums[i] /

2) 。将你的分数增加 nums[i] 。不过，你最多只能执行 k 次操作。请你返回你能得到的 最大分数 。
 *
 * 41. LeetCode 2846 - 边权重均等查询
 *   链接：https://leetcode.cn/problems/minimum-edge-weight-equilibrium-queries-in-a-tree/
 *   题目描述：现有一棵由 n 个节点组成的无向树，节点按从 0 到 n - 1 编号。给你一个整数 n 和一个二维整数数组 edges ，其中 edges[i] = [ui, vi, wi] 表示节点 ui 和 vi 之间有一条边权为 wi 的边。另给你一个二维整数数组 queries ，其中 queries[j] = [aj, bj] 。对于每个查询，你需要找出从 aj 到 bj 的路径上 边权重出现次数最大值 的最小可能值。
 *
 * 42. LeetCode 2861 - 最大合金数
 *   链接：https://leetcode.cn/problems/maximum-number-of-alloys/
 *   题目描述：假设你是一家合金制造公司的老板，你的公司使用多种金属来制造合金。现在你有 n 台机器，每台机器都需要消耗一定数量的每种金属。给定一个整数 budget 表示你的预算，和一个 2D 整数数组 composition ，其中 composition[i] 是一个长度为 m 的数组，表示第 i 台机器制造一单位合金需要消耗的各种金属的数量。另外给你一个长度为 m 的整数数组 stock ，表示你目前拥有的各种金属的库存量，和一个长度为 m 的整数数组 cost ，表示购买一单位各种金属需要的费用。请你计算在预算范围内，通过任意一台机器最多可以制造多少单位的合金。
 *
 * 43. LeetCode 2872 - 最大子序列交替和
 *   链接：https://leetcode.cn/problems/maximum-subsequence-alternating-sum/
 *   题目描述：给你一个下标从 0 开始的整数数组 nums 。一个子序列的 交替和 定义为：子序列中偶数下标元素和 减去 奇数下标元素和。比方说，[4,2,5,3] 的交替和为 (4 + 5) - (2 + 3) = 4 。请你返回 nums 中任意子序列的 最大交替和 。
 *
 * 44. LeetCode 2897 - 执行操作使两个字符串相等
 *   链接：https://leetcode.cn/problems/apply-operations-on-array-to-maximize-sum-of-squares/
 *   题目描述：给你两个下标从 0 开始的二进制字符串 s 和 target ，两个字符串的长度均为 n 。你可以对 s 执行以下操作 任意 次：选择两个 不同 的下标 i 和 j ，其中 0 <= i, j < n 。同时，将 s[i] 替换为 (s[i] OR s[j]) ，将 s[j] 替换为 (s[i] XOR s[j]) 。请你返回使 s 转化成为 target 需要的 最少 操作次数。如果不可能完成转化，请你返回 -1 。
 *
 * 45. LeetCode 2926 - 平衡子序列的最大和
 *   链接：https://leetcode.cn/problems/maximum-balanced-subsequence-sum/
 *   题目描述：给你一个下标从 0 开始的整数数组 nums 。nums 的一个子序列如果满足以下条件，那么它是 平衡的 ：对于子序列中每两个 相邻 元素，它们的绝对差最多为 1 。也就是说，对于子序列中每两个相邻的值 nums[i] 和 nums[j] ，有 |nums[i] - nums[j]| <= 1 。请你返回 nums 中 平衡 子序列的 最大 和 。
 *
 * 46. LeetCode 295 - 数据流的中位数
 *   链接：https://leetcode.cn/problems/find-median-from-data-stream/
 *   题目描述：设计一个支持以下两种操作的数据结构：void addNum(int num) - 从数据流中添加一个整数到数据结构中。double findMedian() - 返回目前所有元素的中位数。
 *
 * 47. LeetCode 703 - 数据流中的第 K 大元素

 *   链接：https://leetcode.cn/problems/kth-largest-element-in-a-stream/
 *   题目描述：设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个不同的元素。
 *
 * 48. LeetCode 220 - 存在重复元素 III
 *   链接：https://leetcode.cn/problems/contains-duplicate-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 。
 *
 * 49. LeetCode 352 - 将数据流变为多个不相交区间
 *   链接：https://leetcode.cn/problems/data-stream-as-disjoint-intervals/
 *   题目描述：给定一个非负整数的数据流输入 a1，a2，…，an，…，将到目前为止看到的数字总结为不相交的区间列表。
 *
 * 50. LeetCode 855 - 考场就座
 *   链接：https://leetcode.cn/problems/exam-room/
 *   题目描述：在考场里，有 N 个座位，分别编号为 0, 1, 2, ..., N-1 。当学生进入考场后，他必须坐在能够使他与离他最近的人之间的距离达到最大化的座位上。如果有多个这样的座位，他会坐在编号最小的座位上。
 *
 * 51. LeetCode 981 - 基于时间的键值存储
 *   链接：https://leetcode.cn/problems/time-based-key-value-store/
 *   题目描述：设计一个基于时间的键值数据结构，该结构可以在不同时间戳存储对应同一个键的多个值，并检索特定时间戳的值。
 *
 * 52. LeetCode 1146 - 快照数组
 *   链接：https://leetcode.cn/problems/snapshot-array/
 *   题目描述：实现支持下列接口的「快照数组」- SnapshotArray：SnapshotArray(int length) - 初始化一个与指定长度相等的 类数组 的数据结构。初始时，每个元素都等于 0。void set(index, val) - 会将指定索引 index 处的元素设置为 val。int snap() - 获取该数组的快照，并返回快照的 id snap_id（快照号是调用 snap() 的总次数减去 1）。int get(index, snap_id) - 根据指定的 snap_id 选择快照，并返回该快照指定索引 index 的值。
 *
 * 53. LeetCode 1348 - 推文计数
 *   链接：https://leetcode.cn/problems/tweet-counts-per-frequency/
 *   题目描述：请你实现一个能够支持以下两种方法的推文计数类 TweetCounts：recordTweet(string tweetName, int time) - 记录推文发布情况：用户 tweetName 在 time（以 秒 为单位）时刻发布了一条推文。getTweetCountsPerFrequency(string freq, string tweetName, int startTime, int endTime) - 返回从开始时间 startTime（以 秒 为单位）到结束时间 endTime（以 秒 为单位）内，每 分 minute，每 时 hour 或者 每日 day（取决于 freq）内指定用户 tweetName 发布的推文总数。
 *
 * 54. LeetCode 1396 - 设计地铁系统
 *   链接：https://leetcode.cn/problems/design-underground-system/

* 题目描述：请你实现一个类 UndergroundSystem ，它支持以下 3 种方法：checkIn(int id, string stationName, int t) - 乘客 id 在时间 t 进入 stationName 站。checkOut(int id, string stationName, int t) - 乘客 id 在时间 t 离开 stationName 站。getAverageTime(string startStation, string endStation) - 返回从 startStation 站到 endStation 站的平均时间。
*
* 55. LeetCode 1606 - 找到处理最多请求的服务器
* 链接：https://leetcode.cn/problems/find-servers-that-handled-most-number-of-requests/
* 题目描述：你有 k 个服务器，编号为 0 到 k-1 ，它们可以同时处理多个请求组。每个服务器有无穷的计算能力但是 不能同时处理超过一个请求 。请求分配到服务器的规则如下：第 i （序号从 0 开始）个请求到达。如果所有服务器都已被占据，那么该请求被舍弃（完全不处理）。如果第 (i % k) 个服务器空闲，那么对应服务器会处理该请求。否则，将请求安排给下一个空闲的服务器（服务器构成一个环，必要的话可能从第 0 个服务器开始继续找下一个空闲的服务器）。比方说，如果第 i 个请求到达时，第 (i % k) 个服务器被占，那么会查看第 (i+1) % k 个服务器，第 (i+2) % k 个服务器等等。给你一个 严格递增 的正整数数组 arrival ，表示第 i 个任务的到达时间，和另一个数组 load ，其中 load[i] 表示第 i 个请求的工作量（也就是完成该请求需要花费的时间）。你的任务是找到 最繁忙的服务器 。最繁忙的服务器是指处理请求数最多的服务器。请你返回包含所有 最繁忙的服务器 序号的列表，你可以以任何顺序返回这个列表。
*
* 56. LeetCode 1825 - 求出 MK 平均值
* 链接：https://leetcode.cn/problems/finding-mk-average/
* 题目描述：给你两个整数 m 和 k ，以及数据流形式的若干整数。你需要实现一个数据结构，计算这个数据流的 MK 平均值 。MK 平均值 按照如下步骤计算：如果数据流中的整数少于 m 个，MK 平均值 为 -1 ，否则将数据流中的最后 m 个元素拷贝到一个独立的容器中。从这个容器中删除最小的 k 个数和最大的 k 个数。计算剩余元素的平均值，并 向下取整到最近的整数 。请你实现 MKAverage 类：MKAverage(int m, int k) 用一个空的数据流和两个整数 m 和 k 初始化 MKAverage 对象。void addElement(int num) 往数据流中插入一个新的元素 num 。int calculateMKAverage() 对当前的数据流计算并返回 MK 平均数 ，结果需 取整到最近的整数 。
*
* 57. LeetCode 2034 - 股票价格波动
* 链接：https://leetcode.cn/problems/stock-price-fluctuation/
* 题目描述：给你一支股票价格的数据流。数据流中每一条记录包含一个 时间戳 和该时间点股票对应的价格 。不巧的是，由于股票市场内在的波动性，股票价格记录可能不是按时间顺序到来的。某些情况下，有的记录可能是错的，即时间戳和价格可能都不对。请你实现一个类：StockPrice() 初始化对象，当前无股票价格记录。void update(int timestamp, int price) 在时间点 timestamp 更新股票价格为 price 。int current() 返回股票 最新价格 。int maximum() 返回股票 最高价格 。int minimum() 返回股票 最低价格 。
*
* 58. LeetCode 2102 - 序列顺序查询
* 链接：https://leetcode.cn/problems/sequentially-ordinal-rank-tracker/
* 题目描述：一个观光景点由它的名字 name 和评分 score 组成，其中 name 是所有观光景点中 唯一的字符串，score 是一个整数。景点按照以下规则进行排序：评分 越高 ，景点的排名越高。如果两个景点的评分相同，那么 字典序较小 的景点排名更高。请你实现一个类 SORTracker ：SORTracker() 初始化系统。void add(string name, int score) 添加一个名为 name 评分为 score 的景点。string get() 返回第 i 次调用时排名第 i 的景点，其中 i 是系统当前位置的下标（从 1 开始）。

 *
 * 59. LeetCode 2349 - 设计数字容器系统
 *   链接：https://leetcode.cn/problems/design-a-number-container-system/
 *   题目描述：设计一个数字容器系统，可以实现以下功能：void change(int index, int number) - 将下标为 index 处的数字改为 number 。如果该下标 index 处已经有数字，那么原来的数字会被替换。int find(int number) - 返回给定数字 number 所在的下标中的最小下标。如果系统中不存在数字 number ，返回 -1 。
 *
 * 60. LeetCode 2424 - 最长上传前缀
 *   链接：https://leetcode.cn/problems/longest-uploaded-prefix/
 *   题目描述：给你一个整数 n 和一个下标从 1 开始的二进制数组（只包含 0 和 1 的数组）queries 。一开始，所有元素都是 0 。你需要处理 queries 中的两种类型的操作：queries[i] == 1：将下标为 queries[i+1] 的元素设为 1 。queries[i] == 2：返回由 1 组成的 最长 上传前缀的长度。
 *
 * 61. LeetCode 2528 - 最大化城市的最小供电站数目
 *   链接：https://leetcode.cn/problems/maximize-the-minimum-powered-city/
 *   题目描述：给你一个下标从 0 开始长度为 n 的整数数组 stations ，其中 stations[i] 表示第 i 座城市的供电站数目。每个供电站可以为 一座 城市供电。一座城市如果被 至少一个 供电站覆盖，我们称它被覆盖 。你需要额外建造 k 座供电站。你可以选择建在任何城市。请你返回额外建造 k 座供电站后，被覆盖城市的最小供电站数目的 最大值 。
 *
 * 62. LeetCode 2532 - 过桥的时间
 *   链接：https://leetcode.cn/problems/time-to-cross-a-bridge/
 *   题目描述：共有 k 位工人计划将 n 个箱子从旧仓库移动到新仓库。给你两个整数 n 和 k，以及一个二维整数数组 time ，数组的长度为 k，其中 time[i] = [leftToRighti, pickOldi, rightToLefti, putNewi] 。一条河将仓库分成了旧仓库和新仓库，所有工人一开始都在旧仓库。请你返回最后一个箱子到达新仓库的时刻。
 *
 * 63. LeetCode 2560 - 打家劫舍 IV
 *   链接：https://leetcode.cn/problems/house-robber-iv/
 *   题目描述：沿街有一排连续的房屋。每间房屋内都藏有一定的现金。现在，你打算偷窃这些房屋。但是，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。给定一个代表每个房屋存放金额的非负整数数组，计算你 在不触动警报装置的情况下 ，能够偷窃到的最高金额。
 *
 * 64. LeetCode 2610 - 转换二维数组
 *   链接：https://leetcode.cn/problems/convert-an-array-into-a-2d-array-with-conditions/
 *   题目描述：给你一个整数数组 nums 。请你返回一个二维数组，该数组需满足：数组中的每个元素都是互不相同 的。数组中的每一行必须包含 nums 中所有不同的元素。数组中的行数应尽可能少。返回结果数组。如果存在多种答案，只需返回其中任意一种。
 *
 * 65. LeetCode 2641 - 二叉树的堂兄弟节点 II
 *   链接：https://leetcode.cn/problems/cousins-in-binary-tree-ii/

 * 题目描述：给你一棵二叉树的根节点 root ，请你将每个节点的值替换成该节点的所有 堂兄弟节点值的和 。堂兄弟节点指深度相同但父节点不同的节点。
 *
 * 66. LeetCode 2653 - 滑动子数组的美丽值
 * 链接：https://leetcode.cn/problems/sliding-subarray-beauty/
 * 题目描述：给你一个长度为 n 的整数数组 nums ，请你求出每个长度为 k 的子数组的 美丽值 。一个子数组的美丽值定义为：子数组中第 x 小的数，如果它是负数，那么美丽值就是这个数；否则，美丽值为 0 。
 *
 * 67. LeetCode 2696 - 删除子串后的字符串最小长度
 * 链接：https://leetcode.cn/problems/minimum-string-length-after-removing-substrings/
 * 题目描述：给你一个字符串 s ，它仅由大写英文字母组成。你可以对这个字符串执行一些操作，每次操作可以删除 s 中的一个子串 "AB" 或 "CD" 。请你返回使字符串 s 变为 空字符串 需要删除的最少操作次数。
 *
 * 68. LeetCode 2736 - 最大和查询
 * 链接：https://leetcode.cn/problems/maximum-sum-queries/
 * 题目描述：给你两个长度为 n 的整数数组 nums1 和 nums2 ，以及一个长度为 m 的整数数组 queries 。对于每个查询 queries[i] = [xi, yi] ，你需要找到满足 nums1[j] >= xi 且 nums2[j] >= yi 的下标 j (0 <= j < n) ，并返回 nums1[j] + nums2[j] 的 最大值 。如果不存在满足条件的 j ，则返回 -1 。
 *
 * 69. LeetCode 2818 - 操作使得分最大
 * 链接：https://leetcode.cn/problems/apply-operations-to-maximize-score/
 * 题目描述：给你一个下标从 0 开始的整数数组 nums 和一个整数 k 。你的 起始分数 为 0 。在一步操作 中，你可以：选择一个下标 i 满足 0 <= i < nums.length 。将 nums[i] 替换为 floor(nums[i] / 2) 。将你的分数增加 nums[i] 。不过，你最多只能执行 k 次操作。请你返回你能得到的 最大分数 。
 *
 * 70. LeetCode 2846 - 边权重均等查询
 * 链接：https://leetcode.cn/problems/minimum-edge-weight-equilibrium-queries-in-a-tree/
 * 题目描述：现有一棵由 n 个节点组成的无向树，节点按从 0 到 n - 1 编号。给你一个整数 n 和一个二维整数数组 edges ，其中 edges[i] = [ui, vi, wi] 表示节点 ui 和 vi 之间有一条边权为 wi 的边。另给你一个二维整数数组 queries ，其中 queries[j] = [aj, bj] 。对于每个查询，你需要找出从 aj 到 bj 的路径上 边权重出现次数最大值 的最小可能值。
 *
 * 71. LeetCode 2861 - 最大合金数
 * 链接：https://leetcode.cn/problems/maximum-number-of-alloys/
 * 题目描述：假设你是一家合金制造公司的老板，你的公司使用多种金属来制造合金。现在你有 n 台机器，每台机器都需要消耗一定数量的每种金属。给定一个整数 budget 表示你的预算，和一个 2D 整数数组 composition ，其中 composition[i] 是一个长度为 m 的数组，表示第 i 台机器制造一单位合金需要消耗的各种金属的数量。另外给你一个长度为 m 的整数数组 stock ，表示你目前拥有的各种金属的库存量，和一个长度为 m 的整数数组 cost ，表示购买一单位各种金属需要的费用。请你计算在预算范围内，通过任意一台机器最多可以制造多少单位的合金。

*
* 72. LeetCode 2872 - 最大子序列交替和
*   链接：https://leetcode.cn/problems/maximum-subsequence-alternating-sum/
*   题目描述：给你一个下标从 0 开始的整数数组 nums 。一个子序列的 交替和 定义为：子序列中偶数下标元素和 减去 奇数下标元素和。比方说，[4,2,5,3] 的交替和为 (4 + 5) - (2 + 3) = 4 。请你返回 nums 中任意子序列的 最大交替和 。
*
* 73. LeetCode 2897 - 执行操作使两个字符串相等
*   链接：https://leetcode.cn/problems/apply-operations-on-array-to-maximize-sum-of-squares/
*   题目描述：给你两个下标从 0 开始的二进制字符串 s 和 target ，两个字符串的长度均为 n 。你可以对 s 执行以下操作 任意 次：选择两个 不同 的下标 i 和 j ，其中 0 <= i, j < n 。同时，将 s[i] 替换为 (s[i] OR s[j]) ，将 s[j] 替换为 (s[i] XOR s[j]) 。请你返回使 s 转化成为 target 需要的 最少 操作次数。如果不可能完成转化，请你返回 -1 。
*
* 74. LeetCode 2926 - 平衡子序列的最大和
*   链接：https://leetcode.cn/problems/maximum-balanced-subsequence-sum/
*   题目描述：给你一个下标从 0 开始的整数数组 nums 。nums 的一个子序列如果满足以下条件，那么它是 平衡的 ：对于子序列中每两个 相邻 元素，它们的绝对差最多为 1 。也就是说，对于子序列中每两个相邻的值 nums[i] 和 nums[j] ，有 |nums[i] - nums[j]| <= 1 。请你返回 nums 中 平衡 子序列的 最大 和 。
*
* 75. LeetCode 295 - 数据流的中位数
*   链接：https://leetcode.cn/problems/find-median-from-data-stream/
*   题目描述：设计一个支持以下两种操作的数据结构：void addNum(int num) - 从数据流中添加一个整数到数据结构中。double findMedian() - 返回目前所有元素的中位数。
*
* 76. LeetCode 480 - 滑动窗口中位数
*   链接：https://leetcode.cn/problems/sliding-window-median/
*   题目描述：中位数是有序序列最中间的那个数。如果序列的大小是偶数，则没有最中间的数；此时中位数是最中间的两个数的平均数。
*
* 77. LeetCode 703 - 数据流中的第 K 大元素
*   链接：https://leetcode.cn/problems/kth-largest-element-in-a-stream/
*   题目描述：设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个不同的元素。
*
* 78. LeetCode 220 - 存在重复元素 III
*   链接：https://leetcode.cn/problems/contains-duplicate-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 。
*
* 79. LeetCode 352 - 将数据流变为多个不相交区间
*   链接：https://leetcode.cn/problems/data-stream-as-disjoint-intervals/
*   题目描述：给定一个非负整数的数据流输入 a1，a2，…，an，…，将到目前为止看到的数字总结为不

相交的区间列表。
 *
 * 80. LeetCode 855 - 考场就座
 *   链接：https://leetcode.cn/problems/exam-room/
 *   题目描述：在考场里，有 N 个座位，分别编号为 0, 1, 2, ..., N-1 。当学生进入考场后，他必须坐在能够使他与离他最近的人之间的距离达到最大化的座位上。如果有多个这样的座位，他会坐在编号最小的座位上。
 */
```
public class SkipList1 {
    // 日志系统配置
    private static final Logger logger = Logger.getLogger(SkipList1.class.getName());
    static {
        // 配置日志格式
        try {
            ConsoleHandler handler = new ConsoleHandler();
            SimpleFormatter formatter = new SimpleFormatter() {
                private static final String format = "%1$tY-%1$tm-%1$td %1$tH:%1$tM:%1$tS.%1$tL
[%4$s] %3$s: %5$s%6$s%n";
                private final SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss.SSS");

                @Override
                public synchronized String format(java.util.logging.LogRecord lr) {
                    return String.format(format,
                            lr.getMillis(),
                            dateFormat.format(lr.getMillis()),
                            lr.getSourceClassName(),
                            lr.getLevel().getLocalizedName(),
                            lr.getMessage(),
                            lr.getThrown() != null ? "\n" + lr.getThrown() : "");
                }
            };
            handler.setFormatter(formatter);
            logger.addHandler(handler);
            logger.setUseParentHandlers(false);
            logger.setLevel(Level.INFO); // 默认日志级别
        } catch (Exception e) {
            System.err.println("Failed to configure logger: " + e.getMessage());
        }
    }

    // 随机数生成器
    private static final Random random = new Random();
```

```
// 概率因子，用于随机生成节点层数
private static final double PROBABILITY_FACTOR = 0.5;

// 最大层数
private final int maxLevel;

// 当前跳表的最高层数
private int currentLevel;

// 头节点，哨兵节点
private final SkipListNode head;

// 节点总数
private int size;

// 层数分布统计数组
private final int[] levelDistribution;

/**
 * 跳表节点类
 */
private static class SkipListNode {
    int key;           // 键值
    int count;         // 相同键值的节点计数
    SkipListNode[] next; // 指向各层的下一个节点
    int[] span;        // 跨度数组，用于排名查询

    /**
     * 构造函数
     *
     * @param key 键值
     * @param level 节点的层数
     * @throws IllegalArgumentException 如果层数无效
     */
    public SkipListNode(int key, int level) {
        if (level < 0) {
            throw new IllegalArgumentException("Level cannot be negative: " + level);
        }
        this.key = key;
        this.count = 1;
        this.next = new SkipListNode[level + 1];
        this.span = new int[level + 1];
```

```
            // 初始化跨度数组为 1
            Arrays.fill(this.span, 1);
        }
    }

    /**
     * 构造函数
     *
     * @param maxLevel 最大层数，默认为 16
     * @throws IllegalArgumentException 如果最大层数无效
     */
    public SkipList1(int maxLevel) {
        // 参数校验
        if (maxLevel <= 0 || maxLevel > 64) {
            throw new IllegalArgumentException("Invalid maxLevel: " + maxLevel + ", should be
between 1 and 64");
        }

        this.maxLevel = maxLevel;
        this.currentLevel = 0;
        // 创建头节点，使用 Integer.MIN_VALUE 作为 key
        this.head = new SkipListNode(Integer.MIN_VALUE, maxLevel);
        this.size = 0;
        this.levelDistribution = new int[maxLevel + 1];

        logger.info("SkipList created with maxLevel: " + maxLevel);
    }

    /**
     * 默认构造函数，使用默认最大层数 16
     */
    public SkipList1() {
        this(16);
    }

    /**
     * 随机生成节点的层数
     * 通过几何分布决定节点的层数，每增加一层的概率为 PROBABILITY_FACTOR
     *
     * 时间复杂度：O(log n) 期望时间复杂度
     * 空间复杂度：O(1)
     *
     * 【设计说明】
```

```
     * 1. 层数生成遵循几何分布，平均层数为 1/(1-PROBABILITY_FACTOR)
     * 2. 当 PROBABILITY_FACTOR=0.5 时，平均层数为 2
     * 3. 限制最大层数为 maxLevel，避免极端情况下的性能问题
     * 4. 使用 Random 类代替 Math.random()，提供更好的可测试性
     *
     * @return 节点的层数
     */
    private int randomLevel() {
        int level = 0;
        // 通过几何分布决定是否增加层数
        while (random.nextDouble() < PROBABILITY_FACTOR && level < maxLevel) {
            level++;
        }
        // 返回层数，不超过最大层数限制
        return Math.min(level, maxLevel);
    }

    /**
     * 在跳表中查找指定值的节点
     * 从指定节点的指定层开始，逐层向下查找
     *
     * 时间复杂度：O(log n) 期望时间复杂度
     * 空间复杂度：O(log n) 递归调用栈空间
     *
     * 【算法核心】
     * 1. 从高层向低层查找，每层尽可能向右移动
     * 2. 到达最底层后检查是否找到目标值
     * 3. 递归实现简洁明了
     *
     * @param i 当前节点编号
     * @param h 当前层数
     * @param num 要查找的值
     * @return 找到的节点编号，未找到返回 0
     */
    private int find(int i, int h, int num) {
        // 输入参数校验
        if (i <= 0 || h <= 0 || h > maxLevel) {
            throw new IllegalArgumentException("Invalid parameters: i=" + i + ", h=" + h);
        }

        // 在当前层向右移动，直到找到大于等于 num 的节点或到达末尾
        while (head.next[h] != null && head.next[h].key < num) {
            i = head.next[h].key;
```

```
        }

        // 如果到达最底层
        if (h == 1) {
            // 检查下一个节点是否就是要找的节点
            if (head.next[h] != null && head.next[h].key == num) {
                logger.fine("找到节点: 值=" + num + ", 索引=" + head.next[h].key);
                return head.next[h].key;  // 找到节点，返回节点编号
            } else {
                logger.fine("未找到节点: 值=" + num);
                return 0;  // 未找到节点
            }
        }

        // 递归到下一层继续查找
        return find(i, h - 1, num);
    }

    /**
     * 搜索指定值是否存在于跳表中
     *
     * 时间复杂度：O(log n) 期望时间复杂度
     * 空间复杂度：O(1)
     *
     * @param num 要搜索的值
     * @return 如果值存在返回 true，否则返回 false
     */
    public boolean search(int num) {
        SkipListNode current = head;
        // 从最高层开始向下查找
        for (int i = currentLevel; i >= 1; i--) {
            // 在当前层向右移动，直到找到大于等于 num 的节点或到达末尾
            while (current.next[i] != null && current.next[i].key < num) {
                current = current.next[i];
            }
        }
        // 检查下一个节点是否就是要找的节点
        current = current.next[1];
        return current != null && current.key == num;
    }

    /**
     * 增加指定值到跳表中
```

```
     * 如果值已存在则增加计数，否则创建新节点
     *
     * 时间复杂度：O(log n) 期望时间复杂度
     * 空间复杂度：O(log n) 递归调用栈空间
     *
     * 【工程细节】
     * 1. 处理重复元素：增加计数而非创建新节点
     * 2. 检查内存空间：防止超出预分配的数组大小
     * 3. 异常处理：捕获并记录可能的异常
     *
     * @param num 要增加的值
     * @throws IllegalStateException 当跳表已满时抛出
     */
    public void add(int num) {
        try {
            // 查找值是否已存在
            if (search(num)) {
                // 如果已存在，增加计数
                SkipListNode current = head;
                // 从最高层开始向下查找
                for (int i = currentLevel; i >= 1; i--) {
                    // 在当前层向右移动，直到找到大于等于 num 的节点或到达末尾
                    while (current.next[i] != null && current.next[i].key < num) {
                        current = current.next[i];
                    }
                }
                // 检查下一个节点是否就是要找的节点
                current = current.next[1];
                if (current != null && current.key == num) {
                    current.count++;
                }
                logger.fine("增加现有节点计数: 值=" + num);
            } else {
                // 如果不存在，创建新节点
                int level = randomLevel();  // 随机生成节点层数
                SkipListNode newNode = new SkipListNode(num, level);  // 创建新节点

                // 更新最大层数
                if (level > currentLevel) {
                    currentLevel = level;
                }

                // 插入节点
```

```
            SkipListNode current = head;
            // 从最高层开始向下插入
            for (int i = currentLevel; i >= 1; i--) {
                // 在当前层向右移动，直到找到大于等于 num 的节点或到达末尾
                while (current.next[i] != null && current.next[i].key < num) {
                    current = current.next[i];
                }
                // 如果当前层不超过新节点的层数，则插入节点
                if (i <= level) {
                    newNode.next[i] = current.next[i];
                    current.next[i] = newNode;
                }
            }
            size++;
            logger.fine("添加新节点: 值=" + num + ", 层数=" + level);
        }
    } catch (Exception e) {
        logger.log(Level.SEVERE, "添加元素失败: " + num, e);
        throw e; // 重新抛出异常
    }
}

/**
 * 从跳表中删除指定值
 * 如果值有多个则只删除一个，否则删除整个节点
 *
 * 时间复杂度：O(log n) 期望时间复杂度
 * 空间复杂度：O(log n) 递归调用栈空间
 *
 * @param num 要删除的值
 */
public boolean erase(int num) {
    // 查找要删除的节点
    if (!search(num)) {
        return false;  // 如果未找到节点，返回 false
    }

    SkipListNode current = head;
    // 从最高层开始向下删除
    for (int i = currentLevel; i >= 1; i--) {
        // 在当前层向右移动，直到找到大于等于 num 的节点或到达末尾
        while (current.next[i] != null && current.next[i].key < num) {
            current = current.next[i];
```

```
            }
            // 如果下一个节点就是要删除的节点
            if (current.next[i] != null && current.next[i].key == num) {
                // 如果节点计数大于 1
                if (current.next[i].count > 1) {
                    // 只减少计数
                    current.next[i].count--;
                } else {
                    // 删除整个节点
                    current.next[i] = current.next[i].next[i];
                }
            }
        }
        size--;
        return true;
    }

    /**
     * 查询指定值的排名
     * 排名定义为比该值小的数的个数加 1
     *
     * 时间复杂度：O(log n) 期望时间复杂度
     * 空间复杂度：O(1)
     *
     * 【实现说明】
     * 利用 small 方法计算比 num 小的元素个数，再加 1 得到排名
     *
     * @param num 要查询排名的值
     * @return 值的排名
     */
    public int rank(int num) {
        int result = small(num) + 1;
        logger.fine("查询排名: 值=" + num + ", 排名=" + result);
        return result;
    }

    /**
     * 查询有多少个数字比指定值小
     * 从指定节点的指定层开始，逐层向下计算
     *
     * 时间复杂度：O(log n) 期望时间复杂度
     * 空间复杂度：O(1)
     *
```

```
     * @param num 要比较的值
     * @return 比 num 小的数字个数
     */
    private int small(int num) {
        int rank = 0;
        SkipListNode current = head;
        // 从最高层开始向下计算
        for (int i = currentLevel; i >= 1; i--) {
            // 在当前层向右移动，直到找到大于等于 num 的节点或到达末尾
            while (current.next[i] != null && current.next[i].key < num) {
                rank += current.next[i].count;  // 累加节点计数
                current = current.next[i];
            }
        }
        return rank;
    }

    /**
     * 查询指定排名的 key 值
     *
     * 时间复杂度：O(log n) 期望时间复杂度
     * 空间复杂度：O(1)
     *
     * 【参数校验】
     * 排名必须为正整数且不超过跳表中的元素总数
     *
     * @param x 要查询的排名
     * @return 排名为 x 的 key 值
     * @throws IndexOutOfBoundsException 当排名超出范围时抛出
     */
    public int index(int x) {
        // 参数校验
        if (x <= 0) {
            throw new IndexOutOfBoundsException("排名必须为正整数: " + x);
        }

        SkipListNode current = head;
        int rank = 0;
        // 从最高层开始向下查找
        for (int i = currentLevel; i >= 1; i--) {
            // 在当前层向右移动，直到累计节点个数达到排名 x
            while (current.next[i] != null && rank + current.next[i].count < x) {
                rank += current.next[i].count;  // 累加节点计数
```

```
            current = current.next[i];
        }
    }
    // 返回排名为 x 的节点的 key 值
    return current.next[1].key;
}

/**
 * 查询指定值的前驱
 * 前驱定义为小于该值的最大数
 *
 * 时间复杂度：O(log n) 期望时间复杂度
 * 空间复杂度：O(1)
 *
 * @param num 要查询前驱的值
 * @return 值的前驱，不存在则返回整数最小值
 */
public int pre(int num) {
    SkipListNode current = head;
    int predecessor = Integer.MIN_VALUE;
    // 从最高层开始向下查找
    for (int i = currentLevel; i >= 1; i--) {
        // 在当前层向右移动，直到找到大于等于 num 的节点或到达末尾
        while (current.next[i] != null && current.next[i].key < num) {
            predecessor = current.next[i].key;  // 更新前驱
            current = current.next[i];
        }
    }
    return predecessor;
}

/**
 * 查询指定值的后继
 * 后继定义为大于该值的最小数
 *
 * 时间复杂度：O(log n) 期望时间复杂度
 * 空间复杂度：O(1)
 *
 * @param num 要查询后继的值
 * @return 值的后继，不存在则返回整数最大值
 */
public int post(int num) {
    SkipListNode current = head;
```

```
    // 从最高层开始向下查找
    for (int i = currentLevel; i >= 1; i--) {
        // 在当前层向右移动，直到找到大于等于 num 的节点或到达末尾
        while (current.next[i] != null && current.next[i].key < num) {
            current = current.next[i];
        }
    }
    // 如果下一个节点存在且值大于 num，返回下一个节点的值
    if (current.next[1] != null && current.next[1].key > num) {
        return current.next[1].key;
    }
    // 移动到下一个节点
    current = current.next[1];
    // 如果到达末尾，返回整数最大值，否则返回下一个节点的值
    if (current == null) {
        return Integer.MAX_VALUE;
    } else {
        return current.next[1].key;
    }
}

/**
 * 获取跳表中的元素总数
 *
 * @return 跳表中的元素总数（包括重复元素）
 */
public int size() {
    return size;
}

/**
 * 测试方法，验证跳表的正确性
 */
public static void test() {
    System.out.println("开始跳表测试...");
    SkipList1 skipList = new SkipList1();

    // 测试插入
    System.out.println("\n 测试插入操作:");
    skipList.add(10);
    skipList.add(20);
    skipList.add(30);
    skipList.add(10);  // 重复元素
```

```
        System.out.println("插入完成");

        // 测试查找
        System.out.println("\n测试查找操作:");
        System.out.println("查找 10: " + skipList.search(10));
        System.out.println("查找 15: " + skipList.search(15));

        // 测试排名
        System.out.println("\n测试排名操作:");
        System.out.println("10 的排名: " + skipList.rank(10));
        System.out.println("20 的排名: " + skipList.rank(20));
        System.out.println("30 的排名: " + skipList.rank(30));

        // 测试按排名查询
        System.out.println("\n测试按排名查询:");
        System.out.println("排名 1 的元素: " + skipList.index(1));
        System.out.println("排名 2 的元素: " + skipList.index(2));
        System.out.println("排名 3 的元素: " + skipList.index(3));

        // 测试前驱后继
        System.out.println("\n测试前驱后继:");
        System.out.println("20 的前驱: " + skipList.pre(20));
        System.out.println("20 的后继: " + skipList.post(20));

        // 测试删除
        System.out.println("\n测试删除操作:");
        System.out.println("删除 10: " + skipList.erase(10));  // 删除一个 10
        System.out.println("删除 10: " + skipList.erase(10));  // 删除另一个 10
        System.out.println("删除 10: " + skipList.erase(10));  // 删除最后一个 10

        System.out.println("\n跳表测试完成!");
    }

    /**
     * 主方法，处理输入输出并执行相应操作
     * 支持两种模式：命令行输入模式和测试模式
     *
     * 时间复杂度：O(n log n)，其中 n 为操作次数
     * 空间复杂度：O(n)
     *
     * @param args 命令行参数
     */
    public static void main(String[] args) throws IOException {
```

```
// 设置日志级别
logger.setLevel(Level.FINE);

// 如果有命令行参数"test"，运行测试模式
if (args.length > 0 && args[0].equals("test")) {
    test();
    return;
}

SkipList1 skipList = new SkipList1();  // 创建跳表

// 使用高效的输入输出方式
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
StreamTokenizer in = new StreamTokenizer(br);
PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out), true);

try {
    in.nextToken();  // 读取操作次数
    int n = (int) in.nval;  // 将输入值转换为整数

    // 处理 n 次操作
    for (int i = 1, op, x; i <= n; i++) {
        in.nextToken();  // 读取操作类型
        op = (int) in.nval;  // 将操作类型转换为整数
        in.nextToken();  // 读取操作数
        x = (int) in.nval;  // 将操作数转换为整数

        try {
            // 根据操作类型执行相应操作
            if (op == 1) {
                // 操作 1：增加 x
                skipList.add(x);
            } else if (op == 2) {
                // 操作 2：删除 x
                skipList.erase(x);
            } else if (op == 3) {
                // 操作 3：查询 x 的排名
                out.println(skipList.rank(x));
            } else if (op == 4) {
                // 操作 4：查询排名第 x 的数
                out.println(skipList.index(x));
            } else if (op == 5) {
                // 操作 5：查询 x 的前驱
```

```
                    out.println(skipList.pre(x));
                } else if (op == 6) {
                    // 操作 6：查询 x 的后继
                    out.println(skipList.post(x));
                } else {
                    // 未知操作，记录错误
                    logger.warning("未知操作类型: " + op);
                }
            } catch (Exception e) {
                // 单个操作失败，记录错误并继续
                logger.log(Level.WARNING, "操作执行失败: op=" + op + ", x=" + x, e);
            }
        }
    } catch (Exception e) {
        // 致命错误，记录并退出
        logger.log(Level.SEVERE, "程序执行失败", e);
    } finally {
        // 清理资源
        try {
            out.flush();
            out.close();
            br.close();
        } catch (Exception e) {
            logger.log(Level.WARNING, "资源释放失败", e);
        }
    }
}
```

==========

文件: SkipList2.java

==========

```
package class149;

/**
 * 跳表的实现(C++版本的 Java 注释)
 *
 * 这个文件包含了 C++版本跳表实现的注释，C++版本和 Java 版本逻辑完全一样
 *
 * 实现一种结构，支持如下操作，要求单次调用的时间复杂度 O(log n)
 * 1，增加 x，重复加入算多个词频
 * 2，删除 x，如果有多个，只删掉一个
```

* 3，查询 x 的排名，x 的排名为，比 x 小的数的个数+1
* 4，查询数据中排名为 x 的数
* 5，查询 x 的前驱，x 的前驱为，小于 x 的数中最大的数，不存在返回整数最小值
* 6，查询 x 的后继，x 的后继为，大于 x 的数中最小的数，不存在返回整数最大值
*
* 所有操作的次数 <= 10^5
* -10^7 <= x <= +10^7
* 测试链接 : https://www.luogu.com.cn/problem/P3369
*/

/*
补充题目列表：
1. LeetCode 1206. 设计跳表
   链接：https://leetcode.cn/problems/design-skiplist
   题目描述：设计一个跳表，支持在 O(log(n))时间内完成增加、删除、搜索操作。

2. 洛谷 P3369 【模板】普通平衡树
   链接：https://www.luogu.com.cn/problem/P3369
   题目描述：维护一个可重集合，支持插入、删除、查询排名、查询第 k 小、查询前驱、查询后继等操作。

3. 洛谷 P3391 【模板】文艺平衡树
   链接：https://www.luogu.com.cn/problem/P3391
   题目描述：维护一个序列，支持区间翻转操作。

4. HDU 1754 I Hate It
   链接：http://acm.hdu.edu.cn/showproblem.php?pid=1754
   题目描述：维护一个序列，支持单点修改和区间最大值查询。

5. POJ 3468 A Simple Problem with Integers
   链接：http://poj.org/problem?id=3468
   题目描述：维护一个序列，支持区间加和区间求和操作。

跳表(Skip List)是一种概率型数据结构，由 William Pugh 在 1990 年提出。
它通过在有序链表的基础上增加多级索引来实现快速查找，平均时间复杂度为 O(log n)。

跳表的核心思想：
1. 在有序链表的基础上增加多层索引
2. 每一层都是下一层的稀疏表示
3. 查找时从高层开始，逐层向下
4. 插入时通过随机函数决定节点层数

跳表与平衡树的对比：
1. 实现简单：跳表不需要复杂的旋转操作，代码更容易编写和维护

2. 并发友好：跳表在并发场景下更容易实现高效的锁策略
3. 范围查询：跳表天然支持高效的范围查询
4. 内存占用：跳表每个节点包含的指针数目可调，通常比平衡树更节省空间
5. 时间复杂度：跳表和平衡树的时间复杂度都是O(log n)，但跳表是期望复杂度

跳表的操作：
1. 查找(search)：从最高层开始，逐层向下查找
2. 插入(add)：查找插入位置，随机生成层数，插入节点并更新索引
3. 删除(remove)：查找节点，删除节点并更新索引

时间复杂度分析：
1. 查找：O(log n) 期望时间复杂度
2. 插入：O(log n) 期望时间复杂度
3. 删除：O(log n) 期望时间复杂度

空间复杂度分析：
每个节点平均包含1/(1-p)个指针，总的空间复杂度为O(n)
*/

```
//#include <bits/stdc++.h>
//
//using namespace std;
//
//const int MAXL = 20;
//const int MAXN = 100001;
//
//int cnt;
//int key[MAXN];
//int key_count[MAXN];
//int level[MAXN];
//int next_node[MAXN][MAXL + 1];
//int len[MAXN][MAXL + 1];
//
//void build() {
//    cnt = 1;
//    key[cnt] = INT_MIN;
//    level[cnt] = MAXL;
//}
//
//void clear() {
//    memset(key + 1, 0, cnt * sizeof(int));
//    memset(key_count + 1, 0, cnt * sizeof(int));
//    memset(level + 1, 0, cnt * sizeof(int));
```

```
//    for (int i = 1; i <= cnt; i++) {
//        memset(next_node[i], 0, (MAXL + 1) * sizeof(int));
//        memset(len[i], 0, (MAXL + 1) * sizeof(int));
//    }
//    cnt = 0;
//}
//
//int randomLevel() {
//    int ans = 1;
//    while ((rand() / double(RAND_MAX)) < 0.5) {
//        ans++;
//    }
//    return min(ans, MAXL);
//}
//
//int find(int i, int h, int num) {
//    while (next_node[i][h] != 0 && key[next_node[i][h]] < num) {
//        i = next_node[i][h];
//    }
//    if (h == 1) {
//        if (next_node[i][h] != 0 && key[next_node[i][h]] == num) {
//            return next_node[i][h];
//        } else {
//            return 0;
//        }
//    }
//    return find(i, h - 1, num);
//}
//
//void addCount(int i, int h, int num) {
//    while (next_node[i][h] != 0 && key[next_node[i][h]] < num) {
//        i = next_node[i][h];
//    }
//    if (h == 1) {
//        key_count[next_node[i][h]]++;
//    } else {
//        addCount(i, h - 1, num);
//    }
//    len[i][h]++;
//}
//
//int addNode(int i, int h, int j) {
//    int rightCnt = 0;
```

```
//    while (next_node[i][h] != 0 && key[next_node[i][h]] < key[j]) {
//        rightCnt += len[i][h];
//        i = next_node[i][h];
//    }
//    if (h == 1) {
//        next_node[j][h] = next_node[i][h];
//        next_node[i][h] = j;
//        len[j][h] = key_count[next_node[j][h]];
//        len[i][h] = key_count[next_node[i][h]];
//        return rightCnt;
//    } else {
//        int downCnt = addNode(i, h - 1, j);
//        if (h > level[j]) {
//            len[i][h]++;
//        } else {
//            next_node[j][h] = next_node[i][h];
//            next_node[i][h] = j;
//            len[j][h] = len[i][h] + 1 - downCnt - key_count[j];
//            len[i][h] = downCnt + key_count[j];
//        }
//        return rightCnt + downCnt;
//    }
//}
//
//void add(int num) {
//    if (find(1, MAXL, num) != 0) {
//        addCount(1, MAXL, num);
//    } else {
//        key[++cnt] = num;
//        key_count[cnt] = 1;
//        level[cnt] = randomLevel();
//        addNode(1, MAXL, cnt);
//    }
//}
//
//void removeCount(int i, int h, int num) {
//    while (next_node[i][h] != 0 && key[next_node[i][h]] < num) {
//        i = next_node[i][h];
//    }
//    if (h == 1) {
//        key_count[next_node[i][h]]--;
//    } else {
//        removeCount(i, h - 1, num);
```

```
//    }
//    len[i][h]--;
//}
//
//void removeNode(int i, int h, int j) {
//    if (h < 1) {
//        return;
//    }
//    while (next_node[i][h] != 0 && key[next_node[i][h]] < key[j]) {
//        i = next_node[i][h];
//    }
//    if (h > level[j]) {
//        len[i][h]--;
//    } else {
//        next_node[i][h] = next_node[j][h];
//        len[i][h] += len[j][h] - 1;
//    }
//    removeNode(i, h - 1, j);
//}
//
//void remove(int num) {
//    int j = find(1, MAXL, num);
//    if (j != 0) {
//        if (key_count[j] > 1) {
//            removeCount(1, MAXL, num);
//        } else {
//            removeNode(1, MAXL, j);
//        }
//    }
//}
//
//int small(int i, int h, int num) {
//    int rightCnt = 0;
//    while (next_node[i][h] != 0 && key[next_node[i][h]] < num) {
//        rightCnt += len[i][h];
//        i = next_node[i][h];
//    }
//    if (h == 1) {
//        return rightCnt;
//    } else {
//        return rightCnt + small(i, h - 1, num);
//    }
//}
```

```
//
//int getRank(int num) {
//    return small(1, MAXL, num) + 1;
//}
//
//int index(int i, int h, int x) {
//    int c = 0;
//    while (next_node[i][h] != 0 && c + len[i][h] < x) {
//        c += len[i][h];
//        i = next_node[i][h];
//    }
//    if (h == 1) {
//        return key[next_node[i][h]];
//    } else {
//        return index(i, h - 1, x - c);
//    }
//}
//
//int index(int x) {
//    return index(1, MAXL, x);
//}
//
//int pre(int i, int h, int num) {
//    while (next_node[i][h] != 0 && key[next_node[i][h]] < num) {
//        i = next_node[i][h];
//    }
//    if (h == 1) {
//        return i == 1 ? INT_MIN : key[i];
//    } else {
//        return pre(i, h - 1, num);
//    }
//}
//
//int pre(int num) {
//    return pre(1, MAXL, num);
//}
//
//int post(int i, int h, int num) {
//    while (next_node[i][h] != 0 && key[next_node[i][h]] < num) {
//        i = next_node[i][h];
//    }
//    if (h == 1) {
//        if (next_node[i][h] == 0) {
```

```
//            return INT_MAX;
//        }
//        if (key[next_node[i][h]] > num) {
//            return key[next_node[i][h]];
//        }
//        i = next_node[i][h];
//        if (next_node[i][h] == 0) {
//            return INT_MAX;
//        } else {
//            return key[next_node[i][h]];
//        }
//    } else {
//        return post(i, h - 1, num);
//    }
//}
//
//int post(int num) {
//    return post(1, MAXL, num);
//}
//
//int main() {
//    ios::sync_with_stdio(false);
//    cin.tie(nullptr);
//    srand(time(0));
//    build();
//    int n;
//    cin >> n;
//    for (int i = 1, op, x; i <= n; i++) {
//        cin >> op >> x;
//        if (op == 1) {
//            add(x);
//        } else if (op == 2) {
//            remove(x);
//        } else if (op == 3) {
//            cout << getRank(x) << endl;
//        } else if (op == 4) {
//            cout << index(x) << endl;
//        } else if (op == 5) {
//            cout << pre(x) << endl;
//        } else {
//            cout << post(x) << endl;
//        }
//    }
```

```
//    clear();
//    return 0;
//}
```

==========