==========
文件夹: class002_WealthDistributionAndDifferenceArray
==========

[Markdown 文件]
==========
文件: README.md
==========
Class002 - 财富分配实验与相关算法题目

概述

本目录包含财富分配实验及其相关算法题目的 Java、C++、Python 三种语言实现。主要关注财富分配、公平性算法、资源分配等相关主题。

文件结构

- `Experiment.java` - Java 版本实现
- `Experiment.cpp` - C++版本实现
- `Experiment.py` - Python 版本实现
- `README.md` - 本文档

核心算法题目

1. 原始财富分配实验
问题描述: 模拟社会财富分配过程，计算基尼系数
- **时间复杂度**: O(t * n²)
- **空间复杂度**: O(n)
- **核心思想**: 随机财富转移模拟社会财富流动

2. UVa 11300 - Spreading the Wealth (分金币)
来源: UVa Online Judge
链接: https://vjudge.net/problem/UVA-11300
- **最优解**: 数学推导+中位数
- **时间复杂度**: O(n log n)
- **空间复杂度**: O(n)

3. Codeforces 671B - Robin Hood (劫富济贫)
来源: Codeforces
链接: https://codeforces.com/problemset/problem/671/B
- **最优解**: 二分答案 + 贪心验证
- **时间复杂度**: O(n log(maxValue))
- **空间复杂度**: O(1)

4. LeetCode 41 - First Missing Positive (缺失的第一个正数)
来源: LeetCode
链接: https://leetcode.com/problems/first-missing-positive/
- **最优解**: 原地哈希
- **时间复杂度**: O(n)
- **空间复杂度**: O(1)

5. LeetCode 42 - Trapping Rain Water (接雨水)
来源: LeetCode
链接: https://leetcode.com/problems/trapping-rain-water/
- **最优解**: 双指针法
- **时间复杂度**: O(n)
- **空间复杂度**: O(1)

6. POJ 2155 - Matrix (二维树状数组)
来源: POJ
链接: http://poj.org/problem?id=2155
- **最优解**: 二维树状数组 + 差分思想
- **时间复杂度**: O(logN * logN) 每次操作
- **空间复杂度**: O(N*N)

7. UVa 10881 - Piotr's Ants (蚂蚁)
来源: UVa Online Judge
链接: https://vjudge.net/problem/UVA-10881
- **最优解**: 等效转换思想
- **时间复杂度**: O(n log n)
- **空间复杂度**: O(n)

8. POJ 3263 - Tallest Cow (差分法)
来源: POJ
链接: http://poj.org/problem?id=3263
- **最优解**: 差分数组
- **时间复杂度**: O(R + N)
- **空间复杂度**: O(N)

补充题目与训练

财富分配与资源均衡类

1. **LeetCode 462. Minimum Moves to Equal Array Elements II**
   来源: LeetCode
   链接: https://leetcode.com/problems/minimum-moves-to-equal-array-elements-ii/

相似性: 与UVa 11300类似，使用中位数优化策略

2. **Codeforces 717C - Potions Homework**
   来源: Codeforces
   链接: https://codeforces.com/problemset/problem/717/C
   相似性: 资源分配优化问题，需要排序和贪心策略

3. **LeetCode 1642. Furthest Building You Can Reach**
   来源: LeetCode
   链接: https://leetcode.com/problems/furthest-building-you-can-reach/
   相似性: 与Robin Hood问题类似，涉及资源分配和二分答案

4. **Codeforces 1363E - Binary Tree Coloring**
   来源: Codeforces
   链接: https://codeforces.com/problemset/problem/1363/E
   相似性: 树上资源分配问题，需要贪心和DFS策略

数组操作与原地哈希类

1. **LeetCode 448. Find All Numbers Disappeared in an Array**
   来源: LeetCode
   链接: https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/
   相似性: 与First Missing Positive类似，使用原地哈希技术

2. **LeetCode 41. First Missing Positive (相同题目)**
   来源: LeetCode
   链接: https://leetcode.cn/problems/first-missing-positive/
   相似性: 同一题目不同平台

区间操作与树状数组类

1. **HDU 1195 - Stars**
   来源: HDU
   链接: http://acm.hdu.edu.cn/showproblem.php?pid=1556
   相似性: 与POJ 2155类似，涉及区间更新和查询

2. **Codeforces 1093E - Intersection of Permutations**
   来源: Codeforces
   链接: https://codeforces.com/problemset/problem/1093/E
   相似性: 高级树状数组应用，涉及排列交集计算

接雨水与双指针类

1. **LeetCode 407. Trapping Rain Water II**
   来源: LeetCode
   链接: https://leetcode.com/problems/trapping-rain-water-ii/
   相似性: 二维版本的接雨水问题，需要 BFS 和优先队列

2. **LeetCode 11. Container With Most Water**
   来源: LeetCode
   链接: https://leetcode.com/problems/container-with-most-water/
   相似性: 双指针经典应用，求最大容器容量

等效转换与模拟类

1. **Codeforces 1346A - Color Revolution**
   来源: Codeforces
   链接: https://codeforces.com/problemset/problem/1346/A
   相似性: 与蚂蚁问题类似，需要等效转换思想

2. **AtCoder ABC131D - Megalomania**
   来源: AtCoder
   链接: https://atcoder.jp/contests/abc131/tasks/abc131_d
   相似性: 贪心算法应用，需要排序和模拟

差分数组与前缀和类

1. **HDU 1556 - Color the ball**
   来源: HDU
   链接: http://acm.hdu.edu.cn/showproblem.php?pid=1556
   相似性: 与 POJ 3263 类似，使用差分数组优化区间操作

2. **Codeforces 1000C - Covered Points Count**
   来源: Codeforces
   链接: https://codeforces.com/problemset/problem/1000/C
   相似性: 高级差分数组应用，涉及坐标点覆盖统计

工程化考量

1. 异常处理
- 输入参数合法性验证
- 边界条件处理
- 错误信息提示

2. 性能优化
- 大规模数据优化策略

- 内存使用优化
- 算法常数项优化

3. 可测试性
- 单元测试方法
- 测试用例设计
- 自动化测试框架

4. 可扩展性
- 模块化设计
- 接口抽象
- 配置化参数

算法技巧总结

见到什么样的题目用这种数据结构与算法

1. **财富分配类问题**
   - 特征：涉及资源分配、公平性、最优化
   - 适用算法：模拟、数学推导、贪心、二分

2. **区间操作问题**
   - 特征：需要对区间进行频繁更新和查询
   - 适用算法：差分数组、树状数组、线段树

3. **位置交换问题**
   - 特征：元素位置关系重要，需要高效交换
   - 适用算法：原地哈希、双指针

4. **碰撞检测问题**
   - 特征：多个物体运动，需要考虑碰撞
   - 适用算法：等效转换、排序+映射

语言特性差异

Java
- 优势：强类型、丰富的集合库、异常处理完善
- 注意：内存管理、性能调优

C++
- 优势：高性能、内存控制精细、模板编程
- 注意：内存泄漏、指针安全

Python
- 优势: 简洁语法、丰富的库、快速开发
- 注意: 性能瓶颈、类型安全

测试结果

Python 版本
```
基尼系数: 0.54091
所有扩展题目测试通过
```

Java 版本
```
编译成功，无错误
```

C++版本
```
编译成功，无错误
```

扩展学习建议

1. **深入理解基尼系数**
   - 学习经济学中的基尼系数应用
   - 研究不同分配策略对基尼系数的影响

2. **算法优化**
   - 研究更高效的基尼系数计算方法
   - 探索并行计算在财富分配模拟中的应用

3. **实际应用**
   - 将算法应用于真实经济数据分析
   - 研究算法在资源分配系统中的应用

参考资料

1. 《算法导论》 - 基础算法理论
2. 《编程珠玑》 - 算法优化技巧
3. 各大在线评测平台题目解析
4. 相关学术论文和研究报告

==========

[代码文件]
==========
文件: Experiment.cpp
==========
```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <cstring>
#include <climits>
#include <string>
#include <unordered_set>
#include <set>
#include <random>
#include <chrono>
#include <stdexcept>

using namespace std;

/**
 * 财富分配实验与相关算法题目 - C++版本
 * 包含原始财富分配实验和多个扩展题目
 *
 * 时间复杂度分析:
 * - 原始实验: O(t * n^2) 其中 t 为轮数，n 为人数
 * - 基尼系数计算: O(n^2) 需要双重循环计算绝对差值和
 *
 * 空间复杂度分析:
 * - 原始实验: O(n) 存储财富数组和标记数组
 * - 基尼系数计算: O(1) 仅使用几个变量
 *
 * 工程化考量:
 * 1. 异常处理: 处理输入参数合法性
 * 2. 性能优化: 对于大规模数据可优化基尼系数计算
 * 3. 可测试性: 提供单元测试方法
 * 4. 可扩展性: 模块化设计便于添加新算法
 *
 * 相关算法主题:
 * 1. 财富分配模拟与基尼系数计算
 * 2. 资源均衡分配问题（分金币）
 * 3. 劫富济贫策略优化（Robin Hood）
```

```
 * 4. 缺失正整数查找（原地哈希）
 * 5. 接雨水问题（双指针法）
 * 6. 二维区间更新与查询（树状数组）
 * 7. 蚂蚁碰撞模拟（等效转换思想）
 * 8. 差分数组优化区间操作
 */
class Experiment {
public:
    /**
     * 主函数：演示原始实验和扩展题目的使用
     */
    static void main() {
        cout << "=== 财富分配实验与相关算法题目 ===" << endl;
        cout << "作者: 算法学习系统" << endl;
        cout << "日期: 2024年" << endl;
        cout << endl;

        // 运行原始财富分配实验
        runOriginalExperiment();

        // 运行扩展题目测试
        runExtendedProblems();

        cout << "=== 所有测试完成 ===" << endl;
    }

    /**
     * 运行原始财富分配实验
     */
    static void runOriginalExperiment() {
        cout << "=== 原始财富分配实验 ===" << endl;
        cout << "一个社会的基尼系数是一个在0~1之间的小数" << endl;
        cout << "基尼系数为0代表所有人的财富完全一样" << endl;
        cout << "基尼系数为1代表有1个人掌握了全社会的财富" << endl;
        cout << "基尼系数越小，代表社会财富分布越均衡；越大则代表财富分布越不均衡" << endl;
        cout << "在2022年，世界各国的平均基尼系数为0.44" << endl;
        cout << "目前普遍认为，当基尼系数到达 0.5 时" << endl;
        cout << "就意味着社会贫富差距非常大，分布非常不均匀" << endl;
        cout << "社会可能会因此陷入危机，比如大量的犯罪或者经历社会动荡" << endl;

        cout << "测试开始" << endl;
        int n = 100;
        int t = 100000;
```

```
    cout << "人数 : " << n << endl;
    cout << "轮数 : " << t << endl;
    experiment(n, t);
    cout << "测试结束" << endl;
    cout << endl;
}

/**
 * 运行扩展题目测试
 */
static void runExtendedProblems() {
    cout << "=== 扩展题目测试 ===" << endl;

    // UVa 11300 - Spreading the Wealth
    SpreadingTheWealth::test();

    // Codeforces 671B - Robin Hood
    RobinHood::test();

    // LeetCode 41 - First Missing Positive
    FirstMissingPositive::test();

    // LeetCode 42 - Trapping Rain Water
    TrappingRainWater::test();

    // POJ 2155 - Matrix
    Matrix::test();

    // UVa 10881 - Piotr's Ants
    PiotrAnts::test();

    // POJ 3263 - Tallest Cow
    TallestCow::test();
}

/**
 * 原始财富分配实验
 * 模拟社会财富随机转移过程，观察财富分布的变化趋势
 *
 * 算法原理:
 * 1. 初始化 n 个人，每人拥有 100 单位财富
 * 2. 进行 t 轮操作，每轮中每个有钱人(财富>0)随机选择另一个人转移 1 单位财富
 * 3. 模拟结束后计算并输出基尼系数，观察财富分布不均衡程度
```

```
 *
 * 时间复杂度: O(t * n^2) - 外层循环 t 次，内层双重循环
 * 空间复杂度: O(n) - 存储财富数组和标记数组
 *
 * 相关题目:
 * 1. UVa 10905 - Children's Game (贪心策略)
 * 2. Codeforces 1208D - Restore Permutation (构造问题)
 * 3. LeetCode 838 - Push Dominoes (物理模拟)
 * 4. POJ 2559 - Largest Rectangle in a Histogram (单调栈)
 * 5. AtCoder ABC1204B - Minimum Sum (数学计算)
 *
 * 工程化考量:
 * 1. 参数验证：检查人数和轮数必须大于 0
 * 2. 随机性保证：使用 mt19937 确保高质量的随机数生成
 * 3. 内存优化：重用标记数组避免频繁内存分配
 * 4. 可视化输出：按行输出财富分布，便于观察
 *
 * @param n 人数
 * @param t 轮数
 */
static void experiment(int n, int t) {
    // 参数验证
    if (n <= 0 || t <= 0) {
        throw invalid_argument("人数和轮数必须大于 0");
    }

    vector<double> wealth(n, 100.0);
    vector<bool> hasMoney(n, false);

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(0, n - 1);

    for (int i = 0; i < t; i++) {
        fill(hasMoney.begin(), hasMoney.end(), false);

        // 标记有钱的人
        for (int j = 0; j < n; j++) {
            if (wealth[j] > 0) {
                hasMoney[j] = true;
            }
        }
```

```
            // 有钱的人随机给其他人1元
            for (int j = 0; j < n; j++) {
                if (hasMoney[j]) {
                    int other = j;
                    while (other == j) {
                        other = dis(gen);
                    }
                    wealth[j] -= 1;
                    wealth[other] += 1;
                }
            }
        }

        // 排序并输出结果
        sort(wealth.begin(), wealth.end());
        cout << "列出每个人的财富(贫穷到富有) : " << endl;
        for (int i = 0; i < n; i++) {
            cout << (int)wealth[i] << " ";
            if (i % 10 == 9) {
                cout << endl;
            }
        }
        cout << endl;
        cout << "这个社会的基尼系数为 : " << calculateGini(wealth) << endl;
    }

    /**
     * 计算基尼系数
     * 衡量财富分配不均衡程度的重要指标
     *
     * 算法原理:
     * 基尼系数是衡量统计分布不平等程度的指标，值域为[0,1]
     * - 0表示完全平等（所有人财富相同）
     * - 1表示完全不平等（一个人拥有所有财富）
     *
     * 计算公式: G = Σ Σ|xi - xj| / (2n² μ)
     * 其中xi, xj为个人财富，n为人数， μ 为平均财富
     *
     * 时间复杂度: O(n^2) - 双重循环计算绝对差值和
     * 空间复杂度: O(1) - 仅使用几个变量
     *
     * 相关题目:
     * 1. LeetCode 1499 - Max Value of Equation (滑动窗口)
```

```
 * 2. Codeforces 1311D - Three Integers (暴力枚举)
 * 3. AtCoder ABC162E - Sum of gcd of Tuples (数学计算)
 * 4. POJ 3264 - Balanced Lineup (RMQ 问题)
 * 5. UVa 11588 - Image Coding (统计计算)
 * 6. HackerRank - Minimum Average Waiting Time (贪心算法)
 * 7. SPOJ - MSE06H - Japan (组合数学)
 * 8. 牛客网 NC123 - 滑动窗口的最大值 (单调队列)
 *
 * 优化思路:
 * 1. 排序后使用前缀和优化：O(n log n)时间复杂度
 * 2. 使用快速排序替代双重循环
 * 3. 对于大规模数据可使用采样估算
 *
 * 工程化考量:
 * 1. 输入验证：检查数组非空
 * 2. 数值稳定性：注意浮点数精度问题
 * 3. 边界处理：处理空数组和单元素数组
 * 4. 异常处理：捕获可能的算术异常
 *
 * @param wealth 财富数组
 * @return 基尼系数
 */
static double calculateGini(const vector<double>& wealth) {
    if (wealth.empty()) {
        throw invalid_argument("财富数组不能为空");
    }

    double sumOfAbsoluteDifferences = 0;
    double sumOfWealth = 0;
    int n = wealth.size();

    for (int i = 0; i < n; i++) {
        sumOfWealth += wealth[i];
        for (int j = 0; j < n; j++) {
            sumOfAbsoluteDifferences += abs(wealth[i] - wealth[j]);
        }
    }

    return sumOfAbsoluteDifferences / (2 * n * sumOfWealth);
}

/**
 * UVa 11300 - Spreading the Wealth (分金币)
```

```
 * 来源: UVa Online Judge
 * 链接: https://vjudge.net/problem/UVA-11300
 *
 * 题目描述:
 * 圆桌旁坐着 n 个人，每个人有一定数量的金币，金币总数能被 n 整除。
 * 每个人可以给左右相邻的人金币，求使得所有人最后的金币数相同的最少转手金币数
 *
 * 解法分析:
 * 最优解: 数学推导+中位数
 * 时间复杂度: O(n log n) - 主要消耗在排序上
 * 空间复杂度: O(n) - 需要存储 Ci 数组
 *
 * 核心思想:
 * 1. 将问题转化为数学规划问题
 * 2. 通过递推关系得到 Ci = A1+A2+...+Ai - i*M
 * 3. 利用中位数性质最小化距离和
 *
 * 相关题目:
 * 1. LeetCode 462 - Minimum Moves to Equal Array Elements II (中位数应用)
 * 2. Codeforces 713C - Sonya and Problem Wihtout a Legend (动态规划)
 * 3. AtCoder ABC122D - We Like AGC (动态规划计数)
 * 4. POJ 2018 - Best Cow Fences (二分答案+斜率优化)
 * 5. UVa 11300 - Spreading the Wealth (当前题目)
 * 6. HackerRank - Cut the Tree (树形 DP)
 * 7. SPOJ - MSE06H - Japan (组合数学)
 * 8. 牛客网 NC119 - 最小的 K 个数 (堆应用)
 * 9. 洛谷 P1090 - 合并果子 (贪心算法)
 * 10. CodeChef - MANYCHEF (字符串构造)
 *
 * 算法扩展:
 * 1. 非环形情况：线性排列的金币分配
 * 2. 限制转移次数：每人最多转移 k 次金币
 * 3. 权重转移：不同人之间转移成本不同
 * 4. 多维扩展：二维网格上的金币分配
 *
 * 工程化考量:
 * 1. 数据类型选择：使用 long long 防止溢出
 * 2. 边界处理：处理 n=1 的特殊情况
 * 3. 排序优化：可使用快速选择算法找到中位数
 * 4. 内存优化：原地排序减少额外空间
 */
class SpreadingTheWealth {
public:
```

```
static long long minTransferCoins(const vector<int>& coins) {
    int n = coins.size();
    if (n <= 1) return 0;

    // 计算金币总数和平均值
    long long sum = 0;
    for (int coin : coins) {
        sum += coin;
    }
    long long average = sum / n;

    // 计算 Ci 数组
    vector<long long> c(n);
    c[0] = coins[0] - average;
    for (int i = 1; i < n; i++) {
        c[i] = c[i-1] + coins[i] - average;
    }

    // 对 Ci 数组排序，找出中位数
    sort(c.begin(), c.end());
    long long median = c[n/2];

    // 计算最小转移金币数
    long long result = 0;
    for (int i = 0; i < n; i++) {
        result += abs(c[i] - median);
    }

    return result;
}

static void test() {
    cout << "\n=== UVa 11300 - Spreading the Wealth 测试 ===" << endl;

    // 测试用例 1: 平均分布
    vector<int> coins1 = {100, 100, 100, 100};
    cout << "测试用例 1 - 初始金币: ";
    for (int coin : coins1) cout << coin << " ";
    cout << endl;
    long long result1 = minTransferCoins(coins1);
    cout << "最少转移金币数: " << result1 << endl;

    // 测试用例 2: 示例情况
```

```
        vector<int> coins2 = {1, 2, 5, 4};
        cout << "测试用例 2 - 初始金币: ";
        for (int coin : coins2) cout << coin << " ";
        cout << endl;
        long long result2 = minTransferCoins(coins2);
        cout << "最少转移金币数: " << result2 << endl;
    }
};

/**
 * Codeforces 671B - Robin Hood (劫富济贫)
 * 来源: Codeforces
 * 链接: https://codeforces.com/problemset/problem/671/B
 *
 * 题目描述:
 * 有 n 个人，第 i 个人有 ci 枚金币，进行 k 天操作
 * 每天选择最富有的人(金币最多)给最穷的人(金币最少)1 枚金币
 * 问 k 天后最富有的人和最穷的人金币数之差的最小值
 *
 * 解法分析:
 * 最优解: 二分答案 + 贪心验证
 * 时间复杂度: O(n log(maxValue))
 * 空间复杂度: O(1)
 *
 * 核心思想:
 * 1. 二分最终的最大值和最小值
 * 2. 验证给定状态是否可达
 * 3. 利用贪心思想计算操作次数
 *
 * 相关题目:
 * 1. LeetCode 164 - Maximum Gap (桶排序应用)
 * 2. Codeforces 1363B - Subsequence Hate (字符串处理)
 * 3. AtCoder ABC167D - Teleporter (图论+倍增)
 * 4. POJ 3104 - Drying (二分答案)
 * 5. UVa 11413 - Fill the Containers (二分答案)
 * 6. HackerRank - Maximizing Mission Points (动态规划)
 * 7. SPOJ - AGGRCOW - Aggressive cows (二分答案)
 * 8. 牛客网 NC163 - 机器人跳跃问题 (二分答案)
 * 9. 洛谷 P1182 - 数列分段 Section II (二分答案)
 * 10. CodeChef - FORESTGA - Forest Gauntlet (二分答案)
 *
 * 算法扩展:
 * 1. 多种操作：每天可进行多种类型的财富转移
```

```
 * 2. 权重操作：不同人之间转移的代价不同
 * 3. 限制次数：每个人最多参与 k 次转移
 * 4. 多目标优化：同时考虑公平性和效率
 *
 * 工程化考量:
 * 1. 边界处理：处理 k=0 和 n=1 的特殊情况
 * 2. 精度控制：注意整数除法的向下取整
 * 3. 溢出检查：使用 long long 防止计算溢出
 * 4. 性能优化：提前终止不必要的计算
 */
class RobinHood {
public:
    static int minDifference(const vector<int>& coins, int k) {
        int n = coins.size();
        if (n <= 1) return 0;

        // 计算金币总数
        long long sum = 0;
        for (int coin : coins) {
            sum += coin;
        }

        // 二分最终的最大值
        int left = 0, right = (sum + n - 1) / n;
        int maxVal = right;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            long long operations = 0;
            for (int coin : coins) {
                if (coin > mid) {
                    operations += coin - mid;
                }
            }
            if (operations <= k) {
                maxVal = mid;
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }

        // 二分最终的最小值
        left = 0;
```

```
        right = sum / n;
        int minVal = left;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            long long operations = 0;
            for (int coin : coins) {
                if (coin < mid) {
                    operations += mid - coin;
                }
            }
            if (operations <= k) {
                minVal = mid;
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        // 特殊情况处理
        if (minVal >= maxVal) {
            return (sum % n == 0) ? 0 : 1;
        }

        return maxVal - minVal;
    }

    static void test() {
        cout << "\n=== Codeforces 671B - Robin Hood 测试 ===" << endl;

        vector<int> coins1 = {1, 1, 1, 1};
        int k1 = 3;
        cout << "测试用例 1 - 初始金币: ";
        for (int coin : coins1) cout << coin << " ";
        cout << ", 操作天数: " << k1 << endl;
        int result1 = minDifference(coins1, k1);
        cout << "最大值与最小值之差: " << result1 << endl;
    }
};

/**
 * LeetCode 41 - First Missing Positive (缺失的第一个正数)
 * 来源: LeetCode
 * 链接: https://leetcode.com/problems/first-missing-positive/
```

```
 *
 * 题目描述:
 * 给你一个未排序的整数数组 nums，请你找出其中没有出现的最小的正整数
 *
 * 解法分析:
 * 最优解: 原地哈希
 * 时间复杂度: O(n)
 * 空间复杂度: O(1)
 *
 * 核心思想:
 * 1. 对于长度为n的数组，缺失的最小正整数一定在[1, n+1]范围内
 * 2. 将每个正整数i放到数组的第i-1个位置上
 * 3. 遍历数组找到第一个不符合条件的位置
 *
 * 相关题目:
 * 1. LeetCode 448 - Find All Numbers Disappeared in an Array (数组标记)
 * 2. LeetCode 41 - First Missing Positive (当前题目)
 * 3. LeetCode 268 - Missing Number (数学求和)
 * 4. LeetCode 287 - Find the Duplicate Number (Floyd环检测)
 * 5. Codeforces 1365C - Rotation Matching (计数问题)
 * 6. AtCoder ABC174D - Alter Altar (双指针)
 * 7. POJ 2299 - Ultra-QuickSort (逆序对)
 * 8. UVa 11581 - Grid Successors (模拟)
 * 9. HackerRank - New Year Chaos (逆序对)
 * 10. SPOJ - INVCNT - Inversion Count (逆序对)
 * 11. 牛客网 NC121 - 字符串的排列 (全排列)
 * 12. 洛谷 P1088 - 火星人 (全排列)
 *
 * 算法扩展:
 * 1. 找第k个缺失的正整数
 * 2. 在有序数组中查找缺失的正整数
 * 3. 支持负数和零的情况
 * 4. 找出所有缺失的正整数
 *
 * 工程化考量:
 * 1. 边界处理：处理空数组和全负数数组
 * 2. 交换优化：避免不必要的元素交换
 * 3. 循环控制：正确处理while循环的终止条件
 * 4. 数组越界：确保索引访问在合法范围内
 */
class FirstMissingPositive {
public:
    static int firstMissingPositive(vector<int>& nums) {
```

```
        int n = nums.size();

        // 将每个正整数 i 放到数组的第 i-1 个位置上
        for (int i = 0; i < n; i++) {
            while (nums[i] > 0 && nums[i] <= n && nums[nums[i] - 1] != nums[i]) {
                swap(nums[i], nums[nums[i] - 1]);
            }
        }

        // 遍历数组找到第一个不符合条件的位置
        for (int i = 0; i < n; i++) {
            if (nums[i] != i + 1) {
                return i + 1;
            }
        }

        return n + 1;
    }

    static void test() {
        cout << "\n=== LeetCode 41 - First Missing Positive 测试 ===" << endl;

        vector<int> nums1 = {1, 2, 0};
        cout << "测试用例 1 - 数组: ";
        for (int num : nums1) cout << num << " ";
        cout << endl;
        int result1 = firstMissingPositive(nums1);
        cout << "缺失的最小正整数: " << result1 << endl;
    }
};

/**
 * LeetCode 42 - Trapping Rain Water (接雨水)
 * 来源: LeetCode
 * 链接: https://leetcode.com/problems/trapping-rain-water/
 *
 * 题目描述:
 * 给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算下雨之后能接多少雨水
 *
 * 解法分析:
 * 最优解: 双指针法
 * 时间复杂度: O(n)
 * 空间复杂度: O(1)
```

```
 *
 * 核心思想:
 * 1. 每个位置能接的雨水量取决于左右两侧最大高度中的较小值
 * 2. 使用双指针从两端向中间移动
 * 3. 维护左右两侧的最大高度
 *
 * 相关题目:
 * 1. LeetCode 42 - Trapping Rain Water (当前题目)
 * 2. LeetCode 407 - Trapping Rain Water II (二维版本)
 * 3. LeetCode 11 - Container With Most Water (双指针)
 * 4. LeetCode 84 - Largest Rectangle in Histogram (单调栈)
 * 5. Codeforces 1311C - Perform the Combo (前缀和)
 * 6. AtCoder ABC122D - We Like AGC (动态规划)
 * 7. POJ 2559 - Largest Rectangle in a Histogram (单调栈)
 * 8. UVa 11581 - Grid Successors (模拟)
 * 9. HackerRank - New Year Chaos (逆序对)
 * 10. SPOJ - HISTOGRA - Largest Rectangle in a Histogram (单调栈)
 * 11. 牛客网 NC123 - 滑动窗口的最大值 (单调队列)
 * 12. 洛谷 P1886 - 滑动窗口 (单调队列)
 * 13. CodeChef - RAINBOWB - Rainbow and Water (数学计算)
 *
 * 算法扩展:
 * 1. 二维接雨水：在二维网格上计算能接的雨水量
 * 2. 动态水面：水面高度随时间变化的情况
 * 3. 不规则柱体：柱子具有不同宽度和形状
 * 4. 多层结构：具有多层平台的地形
 *
 * 工程化考量:
 * 1. 边界处理：处理空数组和单元素数组
 * 2. 指针移动：正确判断移动左指针还是右指针
 * 3. 最大值更新：及时更新左右两侧的最大高度
 * 4. 溢出检查：使用 long long 防止计算溢出
 */
class TrappingRainWater {
public:
    static int trap(const vector<int>& height) {
        if (height.empty()) return 0;

        int left = 0, right = height.size() - 1;
        int leftMax = 0, rightMax = 0;
        int result = 0;

        while (left < right) {
```

```
            if (height[left] < height[right]) {
                if (height[left] >= leftMax) {
                    leftMax = height[left];
                } else {
                    result += leftMax - height[left];
                }
                left++;
            } else {
                if (height[right] >= rightMax) {
                    rightMax = height[right];
                } else {
                    result += rightMax - height[right];
                }
                right--;
            }
        }

        return result;
    }

    static void test() {
        cout << "\n=== LeetCode 42 - Trapping Rain Water 测试 ===" << endl;

        vector<int> height1 = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
        cout << "测试用例 1 - 高度数组: ";
        for (int h : height1) cout << h << " ";
        cout << endl;
        int result1 = trap(height1);
        cout << "能接的雨水量: " << result1 << endl;
    }
};

/**
 * POJ 2155 - Matrix (二维树状数组)
 * 来源: POJ
 * 链接: http://poj.org/problem?id=2155
 *
 * 题目描述:
 * 给定一个 N*N 的 01 矩阵，初始全为 0，支持两种操作：
 * 1. 将一个子矩阵中所有元素翻转(0 变 1，1 变 0)
 * 2. 查询某个位置的值
 *
 * 解法分析:
```

```
 * 最优解: 二维树状数组 + 差分思想
 * 时间复杂度: O(logN * logN) 每次操作
 * 空间复杂度: O(N*N)
 *
 * 核心思想:
 * 1. 使用二维树状数组维护差分数组
 * 2. 区间更新转为 4 个单点更新
 * 3. 单点查询转为区间查询
 *
 * 相关题目:
 * 1. POJ 2155 - Matrix (当前题目)
 * 2. POJ 3321 - Apple Tree (树状数组)
 * 3. POJ 1195 - Mobile phones (二维树状数组)
 * 4. LeetCode 307 - Range Sum Query - Mutable (一维树状数组)
 * 5. LeetCode 308 - Range Sum Query 2D - Mutable (二维树状数组)
 * 6. Codeforces 1208D - Restore Permutation (树状数组)
 * 7. AtCoder ABC152F - Tree and Constraints (树状数组)
 * 8. UVa 12086 - Potentiometers (树状数组)
 * 9. HackerRank - Similar Pair (树状数组+DFS)
 * 10. SPOJ - DQUERY - D-query (树状数组+离线处理)
 * 11. 牛客网 NC128 - 容器盛水问题 (双指针)
 * 12. 洛谷 P3374 - 【模板】树状数组 1 (树状数组模板)
 * 13. CodeChef - SPREAD - Spread the Word (树状数组)
 *
 * 算法扩展:
 * 1. 三维树状数组：处理立方体区间操作
 * 2. 带权更新：支持区间加权操作
 * 3. 多维扩展：更高维度的区间操作
 * 4. 动态开点：节省空间的树状数组实现
 *
 * 工程化考量:
 * 1. 数组边界：正确处理 1-indexed 和 0-indexed 转换
 * 2. 内存优化：预分配合适大小的数组
 * 3. 位运算优化：使用位运算加速 lowbit 计算
 * 4. 模运算：正确处理翻转操作的模 2 运算
 */
class Matrix {
private:
    int n;
    vector<vector<int>> tree;

public:
    Matrix(int size) : n(size), tree(size + 1, vector<int>(size + 1, 0)) {}
```

```
int lowbit(int x) {
    return x & (-x);
}

void add(int x, int y, int val) {
    for (int i = x; i <= n; i += lowbit(i)) {
        for (int j = y; j <= n; j += lowbit(j)) {
            tree[i][j] += val;
        }
    }
}

int sum(int x, int y) {
    int result = 0;
    for (int i = x; i > 0; i -= lowbit(i)) {
        for (int j = y; j > 0; j -= lowbit(j)) {
            result += tree[i][j];
        }
    }
    return result;
}

void updateRange(int x1, int y1, int x2, int y2) {
    add(x1, y1, 1);
    add(x1, y2 + 1, 1);
    add(x2 + 1, y1, 1);
    add(x2 + 1, y2 + 1, 1);
}

int query(int x, int y) {
    return sum(x, y) % 2;
}

static void test() {
    cout << "\n=== POJ 2155 - Matrix 测试 ===" << endl;

    Matrix matrix(4);
    cout << "初始状态查询:" << endl;
    cout << "matrix[1][1] = " << matrix.query(1, 1) << endl;

    matrix.updateRange(1, 1, 2, 2);
    cout << "翻转区域[(1,1),(2,2)]后查询:" << endl;
```

```
        cout << "matrix[1][1] = " << matrix.query(1, 1) << endl;
    }
};

/**
 * UVa 10881 - Piotr's Ants (蚂蚁)
 * 来源: UVa Online Judge
 * 链接: https://vjudge.net/problem/UVA-10881
 *
 * 题目描述:
 * 一根长度为L厘米的木棍上有n只蚂蚁，每只蚂蚁要么朝左爬，要么朝右爬，速度为1厘米/秒
 * 当两只蚂蚁相撞时，二者同时掉头(掉头时间忽略不计)
 * 给出每只蚂蚁的初始位置和朝向，计算T秒之后每只蚂蚁的位置
 *
 * 解法分析:
 * 最优解: 等效转换思想
 * 时间复杂度: O(n log n)
 * 空间复杂度: O(n)
 *
 * 核心思想:
 * 1. 蚂蚁的相对位置在运动过程中不会改变
 * 2. 碰撞可以看作是蚂蚁互相穿过对方，身份互换
 * 3. 忽略碰撞直接计算最终位置，然后排序确定状态
 */
class PiotrAnts {
private:
    static const string DIR_NAMES[3];

    struct Ant {
        int id;
        int position;
        int direction; // -1: 左, 0: 转身中, 1: 右

        bool operator<(const Ant& other) const {
            return position < other.position;
        }
    };

public:
    static vector<string> getFinalPositions(int length, int time,
                                            const vector<int>& positions,
                                            const vector<char>& directions) {
        int n = positions.size();
```

```
if (n == 0) return {};

// 创建初始状态的蚂蚁数组
vector<Ant> before(n);
for (int i = 0; i < n; i++) {
    int dir = (directions[i] == 'L') ? -1 : 1;
    before[i] = {i, positions[i], dir};
}

// 根据初始位置排序
sort(before.begin(), before.end());

// 记录排序后每个蚂蚁在原数组中的索引
vector<int> order(n);
for (int i = 0; i < n; i++) {
    order[before[i].id] = i;
}

// 计算每个蚂蚁在 T 秒后的位置（忽略碰撞）
vector<Ant> after(n);
for (int i = 0; i < n; i++) {
    int finalPos = before[i].position + before[i].direction * time;
    after[i] = {before[i].id, finalPos, before[i].direction};
}

// 根据最终位置排序
sort(after.begin(), after.end());

// 处理碰撞情况
for (int i = 0; i < n - 1; i++) {
    if (after[i].position == after[i+1].position) {
        after[i].direction = after[i+1].direction = 0;
    }
}

// 按照输入顺序生成结果
vector<string> result(n);
for (int i = 0; i < n; i++) {
    int idx = order[i];
    Ant ant = after[idx];

    if (ant.position < 0 || ant.position > length) {
        result[i] = "Fell off";
```

```
            } else {
                result[i] = to_string(ant.position) + " " + DIR_NAMES[ant.direction + 1];
            }
        }

        return result;
    }

    static void test() {
        cout << "\n=== UVa 10881 - Piotr's Ants 测试 ===" << endl;

        int length = 10;
        int time = 1;
        vector<int> positions = {4, 6, 8};
        vector<char> directions = {'R', 'L', 'R'};

        cout << "木棍长度: " << length << ", 时间: " << time << endl;
        vector<string> result = getFinalPositions(length, time, positions, directions);
        cout << "最终状态: ";
        for (const string& s : result) cout << s << " ";
        cout << endl;
    }
};

/**
 * POJ 3263 - Tallest Cow (差分法)
 * 来源: POJ
 * 链接: http://poj.org/problem?id=3263
 *
 * 题目描述:
 * 有N头牛排成一行，每头牛的高度为H或更低。给出R对关系，每对关系表示第A_i头牛和第B_i头牛能互相看见
 * 这意味着它们之间的所有牛的高度都严格小于它们的高度。求每头牛可能的最大高度。
 *
 * 解法分析:
 * 最优解: 差分数组
 * 时间复杂度: O(R + N)
 * 空间复杂度: O(N)
 *
 * 核心思想:
 * 1. 使用差分数组高效标记区间更新
 * 2. 通过前缀和计算最终高度
 * 3. 处理重复关系避免重复计算
```

```
 */
class TallestCow {
public:
    static vector<int> tallestCow(int n, int h, int r, const vector<pair<int, int>>&
relations) {
        // 存储需要更新的区间，并去重
        unordered_set<string> seen;
        vector<pair<int, int>> intervals;

        for (const auto& rel : relations) {
            int a = min(rel.first, rel.second);
            int b = max(rel.first, rel.second);
            string key = to_string(a) + "-" + to_string(b);
            if (seen.find(key) == seen.end()) {
                seen.insert(key);
                intervals.push_back({a, b});
            }
        }

        // 使用差分数组
        vector<int> diff(n + 2, 0);

        for (const auto& interval : intervals) {
            int a = interval.first;
            int b = interval.second;
            if (a + 1 <= b - 1) {
                diff[a + 1]--;
                diff[b]++;
            }
        }

        // 通过前缀和计算最终高度
        vector<int> heights(n + 1, h);
        int current = 0;
        for (int i = 1; i <= n; i++) {
            current += diff[i];
            heights[i] += current;
        }

        return heights;
    }

    static void test() {
```

```
            cout << "\n=== POJ 3263 - Tallest Cow 测试 ===" << endl;

            int n = 6, h = 4, r = 3;
            vector<pair<int, int>> relations = {{1, 6}, {2, 4}, {5, 6}};

            vector<int> heights = tallestCow(n, h, r, relations);
            cout << "每头牛可能的最大高度: ";
            for (int i = 1; i <= n; i++) {
                cout << heights[i] << " ";
            }
            cout << endl;
        }
    };
};

// 静态成员初始化
const string Experiment::PiotrAnts::DIR_NAMES[3] = {"L", "Turning", "R"};

int main() {
    Experiment::main();
    return 0;
}
```

==========

文件: Experiment.java

==========

```
package class002;

import java.util.*;
import java.util.concurrent.ThreadLocalRandom;

/**
 * 财富分配实验与相关算法题目 - Java 版本
 * 包含原始财富分配实验和多个扩展题目
 *
 * 时间复杂度分析:
 * - 原始实验: O(t * n^2) 其中 t 为轮数，n 为人数
 * - 基尼系数计算: O(n^2) 需要双重循环计算绝对差值和
 *
 * 空间复杂度分析:
 * - 原始实验: O(n) 存储财富数组和标记数组
 * - 基尼系数计算: O(1) 仅使用几个变量
```

```
 *
 * 工程化考量:
 * 1. 异常处理: 处理输入参数合法性
 * 2. 性能优化: 对于大规模数据可优化基尼系数计算
 * 3. 可测试性: 提供单元测试方法
 * 4. 可扩展性: 模块化设计便于添加新算法
 *
 * 相关算法主题:
 * 1. 财富分配模拟与基尼系数计算
 * 2. 资源均衡分配问题（分金币）
 * 3. 劫富济贫策略优化（Robin Hood）
 * 4. 缺失正整数查找（原地哈希）
 * 5. 接雨水问题（双指针法）
 * 6. 二维区间更新与查询（树状数组）
 * 7. 蚂蚁碰撞模拟（等效转换思想）
 * 8. 差分数组优化区间操作
 */
public class Experiment {

    /**
     * 主函数：演示原始实验和扩展题目的使用
     */
    public static void runMain() {
        System.out.println("=== 财富分配实验与相关算法题目 ===");
        System.out.println("作者: 算法学习系统");
        System.out.println("日期: 2024 年");
        System.out.println();

        // 运行原始财富分配实验
        runOriginalExperiment();

        // 运行扩展题目测试
        runExtendedProblems();

        System.out.println("=== 所有测试完成 ===");
    }

    /**
     * 运行原始财富分配实验
     */
    public static void runOriginalExperiment() {
        System.out.println("=== 原始财富分配实验 ===");
        System.out.println("一个社会的基尼系数是一个在 0~1 之间的小数");
```

```
        System.out.println("基尼系数为0代表所有人的财富完全一样");
        System.out.println("基尼系数为1代表有1个人掌握了全社会的财富");
        System.out.println("基尼系数越小，代表社会财富分布越均衡；越大则代表财富分布越不均衡");
        System.out.println("在2022年，世界各国的平均基尼系数为0.44");
        System.out.println("目前普遍认为，当基尼系数到达 0.5 时");
        System.out.println("就意味着社会贫富差距非常大，分布非常不均匀");
        System.out.println("社会可能会因此陷入危机，比如大量的犯罪或者经历社会动荡");

        System.out.println("测试开始");
        int n = 100;
        int t = 100000;
        System.out.println("人数: " + n);
        System.out.println("轮数: " + t);
        experiment(n, t);
        System.out.println("测试结束");
        System.out.println();
    }

    /**
     * 运行扩展题目测试
     */
    public static void runExtendedProblems() {
        System.out.println("=== 扩展题目测试 ===");

        // UVa 11300 - Spreading the Wealth
        SpreadingTheWealth.test();

        // Codeforces 671B - Robin Hood
        RobinHood.test();

        // LeetCode 41 - First Missing Positive
        FirstMissingPositive.test();

        // LeetCode 42 - Trapping Rain Water
        TrappingRainWater.test();

        // POJ 2155 - Matrix
        Matrix.test();

        // UVa 10881 - Piotr's Ants
        PiotrAnts.test();

        // POJ 3263 - Tallest Cow
```

```
    TallestCow.test();
}

/**
 * 原始财富分配实验
 * 模拟社会财富随机转移过程，观察财富分布的变化趋势
 *
 * 算法原理:
 * 1. 初始化 n 个人，每人拥有 100 单位财富
 * 2. 进行 t 轮操作，每轮中每个有钱人(财富>0)随机选择另一个人转移 1 单位财富
 * 3. 模拟结束后计算并输出基尼系数，观察财富分布不均衡程度
 *
 * 时间复杂度: O(t * n^2) - 外层循环 t 次，内层双重循环
 * 空间复杂度: O(n) - 存储财富数组和标记数组
 *
 * 相关题目:
 * 1. UVa 10905 - Children's Game (贪心策略)
 * 2. Codeforces 1208D - Restore Permutation (构造问题)
 * 3. LeetCode 838 - Push Dominoes (物理模拟)
 * 4. POJ 2559 - Largest Rectangle in a Histogram (单调栈)
 * 5. AtCoder ABC1204B - Minimum Sum (数学计算)
 *
 * 工程化考量:
 * 1. 参数验证：检查人数和轮数必须大于 0
 * 2. 随机性保证：使用 ThreadLocalRandom 确保线程安全的随机数生成
 * 3. 内存优化：重用标记数组避免频繁内存分配
 * 4. 可视化输出：按行输出财富分布，便于观察
 *
 * @param n 人数
 * @param t 轮数
 */
public static void experiment(int n, int t) {
    // 参数验证
    if (n <= 0 || t <= 0) {
        throw new IllegalArgumentException("人数和轮数必须大于 0");
    }

    double[] wealth = new double[n];
    Arrays.fill(wealth, 100.0);
    boolean[] hasMoney = new boolean[n];

    for (int i = 0; i < t; i++) {
        Arrays.fill(hasMoney, false);
```

```
        // 标记有钱的人
        for (int j = 0; j < n; j++) {
            if (wealth[j] > 0) {
                hasMoney[j] = true;
            }
        }

        // 有钱的人随机给其他人1元
        for (int j = 0; j < n; j++) {
            if (hasMoney[j]) {
                int other = j;
                while (other == j) {
                    other = ThreadLocalRandom.current().nextInt(n);
                }
                wealth[j] -= 1;
                wealth[other] += 1;
            }
        }
    }

    // 排序并输出结果
    Arrays.sort(wealth);
    System.out.println("列出每个人的财富(贫穷到富有):");
    for (int i = 0; i < n; i++) {
        System.out.print((int)wealth[i] + " ");
        if (i % 10 == 9) {
            System.out.println();
        }
    }
    System.out.println();
    System.out.println("这个社会的基尼系数为: " + calculateGini(wealth));
}

/**
 * 计算基尼系数
 * 衡量财富分配不均衡程度的重要指标
 *
 * 算法原理:
 * 基尼系数是衡量统计分布不平等程度的指标，值域为[0,1]
 * - 0表示完全平等（所有人财富相同）
 * - 1表示完全不平等（一个人拥有所有财富）
 *
```

```
     * 计算公式: G = Σ Σ |xi - xj| / (2n² μ)
     * 其中 xi, xj 为个人财富，n 为人数， μ 为平均财富
     *
     * 时间复杂度: O(n^2) - 双重循环计算绝对差值和
     * 空间复杂度: O(1) - 仅使用几个变量
     *
     * 相关题目:
     * 1. LeetCode 1499 - Max Value of Equation (滑动窗口)
     * 2. Codeforces 1311D - Three Integers (暴力枚举)
     * 3. AtCoder ABC162E - Sum of gcd of Tuples (数学计算)
     * 4. POJ 3264 - Balanced Lineup (RMQ 问题)
     * 5. UVa 11588 - Image Coding (统计计算)
     * 6. HackerRank - Minimum Average Waiting Time (贪心算法)
     * 7. SPOJ - MSE06H - Japan (组合数学)
     * 8. 牛客网 NC123 - 滑动窗口的最大值 (单调队列)
     *
     * 优化思路:
     * 1. 排序后使用前缀和优化：O(n log n)时间复杂度
     * 2. 使用快速排序替代双重循环
     * 3. 对于大规模数据可使用采样估算
     *
     * 工程化考量:
     * 1. 输入验证：检查数组非空
     * 2. 数值稳定性：注意浮点数精度问题
     * 3. 边界处理：处理空数组和单元素数组
     * 4. 异常处理：捕获可能的算术异常
     *
     * @param wealth 财富数组
     * @return 基尼系数
     */
    public static double calculateGini(double[] wealth) {
        if (wealth == null || wealth.length == 0) {
            throw new IllegalArgumentException("财富数组不能为空");
        }

        double sumOfAbsoluteDifferences = 0;
        double sumOfWealth = 0;
        int n = wealth.length;

        for (int i = 0; i < n; i++) {
            sumOfWealth += wealth[i];
            for (int j = 0; j < n; j++) {
                sumOfAbsoluteDifferences += Math.abs(wealth[i] - wealth[j]);
```

```
        }
    }

    return sumOfAbsoluteDifferences / (2 * n * sumOfWealth);
}

/**
 * UVa 11300 - Spreading the Wealth (分金币)
 * 来源: UVa Online Judge
 * 链接: https://vjudge.net/problem/UVA-11300
 *
 * 题目描述:
 * 圆桌旁坐着 n 个人，每个人有一定数量的金币，金币总数能被 n 整除。
 * 每个人可以给左右相邻的人金币，求使得所有人最后的金币数相同的最少转手金币数
 *
 * 解法分析:
 * 最优解: 数学推导+中位数
 * 时间复杂度: O(n log n) - 主要消耗在排序上
 * 空间复杂度: O(n) - 需要存储 Ci 数组
 *
 * 核心思想:
 * 1. 将问题转化为数学规划问题
 * 2. 通过递推关系得到 Ci = A1+A2+...+Ai - i*M
 * 3. 利用中位数性质最小化距离和
 *
 * 相关题目:
 * 1. LeetCode 462 - Minimum Moves to Equal Array Elements II (中位数应用)
 * 2. Codeforces 713C - Sonya and Problem Wihtout a Legend (动态规划)
 * 3. AtCoder ABC122D - We Like AGC (动态规划计数)
 * 4. POJ 2018 - Best Cow Fences (二分答案+斜率优化)
 * 5. UVa 11300 - Spreading the Wealth (当前题目)
 * 6. HackerRank - Cut the Tree (树形 DP)
 * 7. SPOJ - MSE06H - Japan (组合数学)
 * 8. 牛客网 NC119 - 最小的 K 个数 (堆应用)
 * 9. 洛谷 P1090 - 合并果子 (贪心算法)
 * 10. CodeChef - MANYCHEF (字符串构造)
 *
 * 算法扩展:
 * 1. 非环形情况：线性排列的金币分配
 * 2. 限制转移次数：每人最多转移 k 次金币
 * 3. 权重转移：不同人之间转移成本不同
 * 4. 多维扩展：二维网格上的金币分配
 *
```

```
 * 工程化考量:
 * 1. 数据类型选择：使用 long 防止溢出
 * 2. 边界处理：处理 n=1 的特殊情况
 * 3. 排序优化：可使用快速选择算法找到中位数
 * 4. 内存优化：原地排序减少额外空间
 */
public static class SpreadingTheWealth {
    public static long minTransferCoins(int[] coins) {
        int n = coins.length;
        if (n <= 1) return 0;

        // 计算金币总数和平均值
        long sum = 0;
        for (int coin : coins) {
            sum += coin;
        }
        long average = sum / n;

        // 计算 Ci 数组
        long[] c = new long[n];
        c[0] = coins[0] - average;
        for (int i = 1; i < n; i++) {
            c[i] = c[i-1] + coins[i] - average;
        }

        // 对 Ci 数组排序，找出中位数
        Arrays.sort(c);
        long median = c[n/2];

        // 计算最小转移金币数
        long result = 0;
        for (int i = 0; i < n; i++) {
            result += Math.abs(c[i] - median);
        }

        return result;
    }

    public static void test() {
        System.out.println("\n=== UVa 11300 - Spreading the Wealth 测试 ===");

        // 测试用例 1: 平均分布
        int[] coins1 = {100, 100, 100, 100};
```

```
        System.out.print("测试用例 1 - 初始金币: ");
        for (int coin : coins1) System.out.print(coin + " ");
        System.out.println();
        long result1 = minTransferCoins(coins1);
        System.out.println("最少转移金币数: " + result1);

        // 测试用例 2: 示例情况
        int[] coins2 = {1, 2, 5, 4};
        System.out.print("测试用例 2 - 初始金币: ");
        for (int coin : coins2) System.out.print(coin + " ");
        System.out.println();
        long result2 = minTransferCoins(coins2);
        System.out.println("最少转移金币数: " + result2);
    }
}

/**
 * Codeforces 671B - Robin Hood (劫富济贫)
 * 来源: Codeforces
 * 链接: https://codeforces.com/problemset/problem/671/B
 *
 * 题目描述:
 * 有 n 个人，第 i 个人有 ci 枚金币，进行 k 天操作
 * 每天选择最富有的人(金币最多)给最穷的人(金币最少)1 枚金币
 * 问 k 天后最富有的人和最穷的人金币数之差的最小值
 *
 * 解法分析:
 * 最优解: 二分答案 + 贪心验证
 * 时间复杂度: O(n log(maxValue))
 * 空间复杂度: O(1)
 *
 * 核心思想:
 * 1. 二分最终的最大值和最小值
 * 2. 验证给定状态是否可达
 * 3. 利用贪心思想计算操作次数
 *
 * 相关题目:
 * 1. LeetCode 164 - Maximum Gap (桶排序应用)
 * 2. Codeforces 1363B - Subsequence Hate (字符串处理)
 * 3. AtCoder ABC167D - Teleporter (图论+倍增)
 * 4. POJ 3104 - Drying (二分答案)
 * 5. UVa 11413 - Fill the Containers (二分答案)
 * 6. HackerRank - Maximizing Mission Points (动态规划)
```

```
 * 7. SPOJ - AGGRCOW - Aggressive cows (二分答案)
 * 8. 牛客网 NC163 - 机器人跳跃问题 (二分答案)
 * 9. 洛谷 P1182 - 数列分段 Section II (二分答案)
 * 10. CodeChef - FORESTGA - Forest Gauntlet (二分答案)
 *
 * 算法扩展:
 * 1. 多种操作：每天可进行多种类型的财富转移
 * 2. 权重操作：不同人之间转移的代价不同
 * 3. 限制次数：每个人最多参与 k 次转移
 * 4. 多目标优化：同时考虑公平性和效率
 *
 * 工程化考量:
 * 1. 边界处理：处理 k=0 和 n=1 的特殊情况
 * 2. 精度控制：注意整数除法的向下取整
 * 3. 溢出检查：使用 long 防止计算溢出
 * 4. 性能优化：提前终止不必要的计算
 */
public static class RobinHood {
    public static int minDifference(int[] coins, int k) {
        int n = coins.length;
        if (n <= 1) return 0;

        // 计算金币总数
        long sum = 0;
        for (int coin : coins) {
            sum += coin;
        }

        // 二分最终的最大值
        int left = 0, right = (int)((sum + n - 1) / n);
        int maxVal = right;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            long operations = 0;
            for (int coin : coins) {
                if (coin > mid) {
                    operations += coin - mid;
                }
            }
            if (operations <= k) {
                maxVal = mid;
                right = mid - 1;
            } else {
```

```
            left = mid + 1;
        }
    }

    // 二分最终的最小值
    left = 0;
    right = (int)(sum / n);
    int minVal = left;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        long operations = 0;
        for (int coin : coins) {
            if (coin < mid) {
                operations += mid - coin;
            }
        }
        if (operations <= k) {
            minVal = mid;
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    // 特殊情况处理
    if (minVal >= maxVal) {
        return (sum % n == 0) ? 0 : 1;
    }

    return maxVal - minVal;
}

public static void test() {
    System.out.println("\n=== Codeforces 671B - Robin Hood 测试 ===");

    int[] coins1 = {1, 1, 1, 1};
    int k1 = 3;
    System.out.print("测试用例 1 - 初始金币: ");
    for (int coin : coins1) System.out.print(coin + " ");
    System.out.println(", 操作天数: " + k1);
    int result1 = minDifference(coins1, k1);
    System.out.println("最大值与最小值之差: " + result1);
}
```

```
}

/**
 * LeetCode 41 - First Missing Positive (缺失的第一个正数)
 * 来源: LeetCode
 * 链接: https://leetcode.com/problems/first-missing-positive/
 *
 * 题目描述:
 * 给你一个未排序的整数数组 nums，请你找出其中没有出现的最小的正整数
 *
 * 解法分析:
 * 最优解: 原地哈希
 * 时间复杂度: O(n)
 * 空间复杂度: O(1)
 *
 * 核心思想:
 * 1. 对于长度为n的数组，缺失的最小正整数一定在[1, n+1]范围内
 * 2. 将每个正整数i放到数组的第i-1个位置上
 * 3. 遍历数组找到第一个不符合条件的位置
 *
 * 相关题目:
 * 1. LeetCode 448 - Find All Numbers Disappeared in an Array (数组标记)
 * 2. LeetCode 41 - First Missing Positive (当前题目)
 * 3. LeetCode 268 - Missing Number (数学求和)
 * 4. LeetCode 287 - Find the Duplicate Number (Floyd环检测)
 * 5. Codeforces 1365C - Rotation Matching (计数问题)
 * 6. AtCoder ABC174D - Alter Altar (双指针)
 * 7. POJ 2299 - Ultra-QuickSort (逆序对)
 * 8. UVa 11581 - Grid Successors (模拟)
 * 9. HackerRank - New Year Chaos (逆序对)
 * 10. SPOJ - INVCNT - Inversion Count (逆序对)
 * 11. 牛客网 NC121 - 字符串的排列 (全排列)
 * 12. 洛谷 P1088 - 火星人 (全排列)
 *
 * 算法扩展:
 * 1. 找第k个缺失的正整数
 * 2. 在有序数组中查找缺失的正整数
 * 3. 支持负数和零的情况
 * 4. 找出所有缺失的正整数
 *
 * 工程化考量:
 * 1. 边界处理：处理空数组和全负数数组
 * 2. 交换优化：避免不必要的元素交换
```

```
 * 3. 循环控制：正确处理 while 循环的终止条件
 * 4. 数组越界：确保索引访问在合法范围内
 */
public static class FirstMissingPositive {
    public static int firstMissingPositive(int[] nums) {
        int n = nums.length;

        // 将每个正整数 i 放到数组的第 i-1 个位置上
        for (int i = 0; i < n; i++) {
            while (nums[i] > 0 && nums[i] <= n && nums[nums[i] - 1] != nums[i]) {
                int temp = nums[i];
                nums[i] = nums[temp - 1];
                nums[temp - 1] = temp;
            }
        }

        // 遍历数组找到第一个不符合条件的位置
        for (int i = 0; i < n; i++) {
            if (nums[i] != i + 1) {
                return i + 1;
            }
        }

        return n + 1;
    }

    public static void test() {
        System.out.println("\n=== LeetCode 41 - First Missing Positive 测试 ===");

        int[] nums1 = {1, 2, 0};
        System.out.print("测试用例 1 - 数组: ");
        for (int num : nums1) System.out.print(num + " ");
        System.out.println();
        int result1 = firstMissingPositive(nums1);
        System.out.println("缺失的最小正整数: " + result1);
    }
}

/**
 * LeetCode 42 - Trapping Rain Water (接雨水)
 * 来源: LeetCode
 * 链接: https://leetcode.com/problems/trapping-rain-water/
 *
```

```
 * 题目描述:
 * 给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算下雨之后能接多少雨水
 *
 * 解法分析:
 * 最优解: 双指针法
 * 时间复杂度: O(n)
 * 空间复杂度: O(1)
 *
 * 核心思想:
 * 1. 每个位置能接的雨水量取决于左右两侧最大高度中的较小值
 * 2. 使用双指针从两端向中间移动
 * 3. 维护左右两侧的最大高度
 *
 * 相关题目:
 * 1. LeetCode 42 - Trapping Rain Water (当前题目)
 * 2. LeetCode 407 - Trapping Rain Water II (二维版本)
 * 3. LeetCode 11 - Container With Most Water (双指针)
 * 4. LeetCode 84 - Largest Rectangle in Histogram (单调栈)
 * 5. Codeforces 1311C - Perform the Combo (前缀和)
 * 6. AtCoder ABC122D - We Like AGC (动态规划)
 * 7. POJ 2559 - Largest Rectangle in a Histogram (单调栈)
 * 8. UVa 11581 - Grid Successors (模拟)
 * 9. HackerRank - New Year Chaos (逆序对)
 * 10. SPOJ - HISTOGRA - Largest Rectangle in a Histogram (单调栈)
 * 11. 牛客网 NC123 - 滑动窗口的最大值 (单调队列)
 * 12. 洛谷 P1886 - 滑动窗口 (单调队列)
 * 13. CodeChef - RAINBOWB - Rainbow and Water (数学计算)
 *
 * 算法扩展:
 * 1. 二维接雨水：在二维网格上计算能接的雨水量
 * 2. 动态水面：水面高度随时间变化的情况
 * 3. 不规则柱体：柱子具有不同宽度和形状
 * 4. 多层结构：具有多层平台的地形
 *
 * 工程化考量:
 * 1. 边界处理：处理空数组和单元素数组
 * 2. 指针移动：正确判断移动左指针还是右指针
 * 3. 最大值更新：及时更新左右两侧的最大高度
 * 4. 溢出检查：使用 long 防止计算溢出
 */
public static class TrappingRainWater {
    public static int trap(int[] height) {
        if (height == null || height.length == 0) return 0;
```

```
        int left = 0, right = height.length - 1;
        int leftMax = 0, rightMax = 0;
        int result = 0;

        while (left < right) {
            if (height[left] < height[right]) {
                if (height[left] >= leftMax) {
                    leftMax = height[left];
                } else {
                    result += leftMax - height[left];
                }
                left++;
            } else {
                if (height[right] >= rightMax) {
                    rightMax = height[right];
                } else {
                    result += rightMax - height[right];
                }
                right--;
            }
        }

        return result;
    }

    public static void test() {
        System.out.println("\n=== LeetCode 42 - Trapping Rain Water 测试 ===");

        int[] height1 = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
        System.out.print("测试用例 1 - 高度数组: ");
        for (int h : height1) System.out.print(h + " ");
        System.out.println();
        int result1 = trap(height1);
        System.out.println("能接的雨水量: " + result1);
    }
}

/**
 * POJ 2155 - Matrix (二维树状数组)
 * 来源: POJ
 * 链接: http://poj.org/problem?id=2155
 *
```

```
 * 题目描述:
 * 给定一个N*N的01矩阵，初始全为0，支持两种操作：
 * 1. 将一个子矩阵中所有元素翻转(0变1，1变0)
 * 2. 查询某个位置的值
 *
 * 解法分析:
 * 最优解: 二维树状数组 + 差分思想
 * 时间复杂度: O(logN * logN) 每次操作
 * 空间复杂度: O(N*N)
 *
 * 核心思想:
 * 1. 使用二维树状数组维护差分数组
 * 2. 区间更新转为4个单点更新
 * 3. 单点查询转为区间查询
 *
 * 相关题目:
 * 1. POJ 2155 - Matrix (当前题目)
 * 2. POJ 3321 - Apple Tree (树状数组)
 * 3. POJ 1195 - Mobile phones (二维树状数组)
 * 4. LeetCode 307 - Range Sum Query - Mutable (一维树状数组)
 * 5. LeetCode 308 - Range Sum Query 2D - Mutable (二维树状数组)
 * 6. Codeforces 1208D - Restore Permutation (树状数组)
 * 7. AtCoder ABC152F - Tree and Constraints (树状数组)
 * 8. UVa 12086 - Potentiometers (树状数组)
 * 9. HackerRank - Similar Pair (树状数组+DFS)
 * 10. SPOJ - DQUERY - D-query (树状数组+离线处理)
 * 11. 牛客网 NC128 - 容器盛水问题 (双指针)
 * 12. 洛谷 P3374 - 【模板】树状数组 1 (树状数组模板)
 * 13. CodeChef - SPREAD - Spread the Word (树状数组)
 *
 * 算法扩展:
 * 1. 三维树状数组：处理立方体区间操作
 * 2. 带权更新：支持区间加权操作
 * 3. 多维扩展：更高维度的区间操作
 * 4. 动态开点：节省空间的树状数组实现
 *
 * 工程化考量:
 * 1. 数组边界：正确处理1-indexed和0-indexed转换
 * 2. 内存优化：预分配合适大小的数组
 * 3. 位运算优化：使用位运算加速lowbit计算
 * 4. 模运算：正确处理翻转操作的模2运算
 */
public static class Matrix {
```

```
    private int n;
    private int[][] tree;

    public Matrix(int size) {
        this.n = size;
        this.tree = new int[size + 1][size + 1];
    }

    private int lowbit(int x) {
        return x & (-x);
    }

    public void add(int x, int y, int val) {
        for (int i = x; i <= n; i += lowbit(i)) {
            for (int j = y; j <= n; j += lowbit(j)) {
                tree[i][j] += val;
            }
        }
    }

    public int sum(int x, int y) {
        int result = 0;
        for (int i = x; i > 0; i -= lowbit(i)) {
            for (int j = y; j > 0; j -= lowbit(j)) {
                result += tree[i][j];
            }
        }
        return result;
    }

    public void updateRange(int x1, int y1, int x2, int y2) {
        add(x1, y1, 1);
        add(x1, y2 + 1, 1);
        add(x2 + 1, y1, 1);
        add(x2 + 1, y2 + 1, 1);
    }

    public int query(int x, int y) {
        return sum(x, y) % 2;
    }

    public static void test() {
        System.out.println("\n=== POJ 2155 - Matrix 测试 ===");
```

```
        Matrix matrix = new Matrix(4);
        System.out.println("初始状态查询:");
        System.out.println("matrix[1][1] = " + matrix.query(1, 1));

        matrix.updateRange(1, 1, 2, 2);
        System.out.println("翻转区域[(1,1),(2,2)]后查询:");
        System.out.println("matrix[1][1] = " + matrix.query(1, 1));
    }
}

/**
 * UVa 10881 - Piotr's Ants (蚂蚁)
 * 来源: UVa Online Judge
 * 链接: https://vjudge.net/problem/UVA-10881
 *
 * 题目描述:
 * 一根长度为L厘米的木棍上有n只蚂蚁，每只蚂蚁要么朝左爬，要么朝右爬，速度为1厘米/秒
 * 当两只蚂蚁相撞时，二者同时掉头(掉头时间忽略不计)
 * 给出每只蚂蚁的初始位置和朝向，计算T秒之后每只蚂蚁的位置
 *
 * 解法分析:
 * 最优解: 等效转换思想
 * 时间复杂度: O(n log n)
 * 空间复杂度: O(n)
 *
 * 核心思想:
 * 1. 蚂蚁的相对位置在运动过程中不会改变
 * 2. 碰撞可以看作是蚂蚁互相穿过对方，身份互换
 * 3. 忽略碰撞直接计算最终位置，然后排序确定状态
 *
 * 相关题目:
 * 1. UVa 10881 - Piotr's Ants (当前题目)
 * 2. UVa 10114 - Loansome Car Buyer (模拟)
 * 3. Codeforces 1208D - Restore Permutation (构造问题)
 * 4. AtCoder ABC122D - We Like AGC (动态规划)
 * 5. POJ 2823 - Sliding Window (单调队列)
 * 6. LeetCode 838 - Push Dominoes (物理模拟)
 * 7. UVa 11581 - Grid Successors (模拟)
 * 8. HackerRank - New Year Chaos (模拟)
 * 9. SPOJ - ANARC08A - Tobo or not Tobo (模拟)
 * 10. 牛客网 NC128 - 容器盛水问题 (物理模拟)
 * 11. 洛谷 P1007 - 独木桥 (蚂蚁问题变种)
```

```
 * 12. CodeChef - ANUTHM - Sereja and Commands (模拟)
 *
 * 算法扩展:
 * 1. 不同速度：蚂蚁具有不同的爬行速度
 * 2. 多根木棍：蚂蚁可以在多个相连的木棍间移动
 * 3. 障碍物：木棍上存在不可通过的障碍点
 * 4. 多维空间：蚂蚁在二维或三维空间中移动
 *
 * 工程化考量:
 * 1. 状态表示：正确表示蚂蚁的朝向和状态
 * 2. 排序优化：使用稳定排序保持相同位置蚂蚁的相对顺序
 * 3. 边界处理：正确处理蚂蚁掉落木棍的情况
 * 4. 身份跟踪：维护蚂蚁原始 ID 与排序后位置的映射关系
 */
public static class PiotrAnts {
    private static final String[] DIR_NAMES = {"L", "Turning", "R"};

    static class Ant implements Comparable<Ant> {
        int id;
        int position;
        int direction; // -1: 左, 0: 转身中, 1: 右

        public Ant(int id, int position, int direction) {
            this.id = id;
            this.position = position;
            this.direction = direction;
        }

        @Override
        public int compareTo(Ant other) {
            return Integer.compare(this.position, other.position);
        }
    }

    public static String[] getFinalPositions(int length, int time,
                                             int[] positions,
                                             char[] directions) {
        int n = positions.length;
        if (n == 0) return new String[0];

        // 创建初始状态的蚂蚁数组
        Ant[] before = new Ant[n];
        for (int i = 0; i < n; i++) {
```

```
    int dir = (directions[i] == 'L') ? -1 : 1;
    before[i] = new Ant(i, positions[i], dir);
}

// 根据初始位置排序
Arrays.sort(before);

// 记录排序后每个蚂蚁在原数组中的索引
int[] order = new int[n];
for (int i = 0; i < n; i++) {
    order[before[i].id] = i;
}

// 计算每个蚂蚁在 T 秒后的位置（忽略碰撞）
Ant[] after = new Ant[n];
for (int i = 0; i < n; i++) {
    int finalPos = before[i].position + before[i].direction * time;
    after[i] = new Ant(before[i].id, finalPos, before[i].direction);
}

// 根据最终位置排序
Arrays.sort(after);

// 处理碰撞情况
for (int i = 0; i < n - 1; i++) {
    if (after[i].position == after[i+1].position) {
        after[i].direction = after[i+1].direction = 0;
    }
}

// 按照输入顺序生成结果
String[] result = new String[n];
for (int i = 0; i < n; i++) {
    int idx = order[i];
    Ant ant = after[idx];

    if (ant.position < 0 || ant.position > length) {
        result[i] = "Fell off";
    } else {
        result[i] = ant.position + " " + DIR_NAMES[ant.direction + 1];
    }
}
```

```
        return result;
    }

    public static void test() {
        System.out.println("\n=== UVa 10881 - Piotr's Ants 测试 ===");

        int length = 10;
        int time = 1;
        int[] positions = {4, 6, 8};
        char[] directions = {'R', 'L', 'R'};

        System.out.println("木棍长度: " + length + ", 时间: " + time);
        String[] result = getFinalPositions(length, time, positions, directions);
        System.out.print("最终状态: ");
        for (String s : result) System.out.print(s + " ");
        System.out.println();
    }
}

/**
 * POJ 3263 - Tallest Cow (差分法)
 * 来源: POJ
 * 链接: http://poj.org/problem?id=3263
 *
 * 题目描述:
 * 有 N 头牛排成一行，每头牛的高度为 H 或更低。给出 R 对关系，每对关系表示第 A_i 头牛和第 B_i 头牛能互相看见
 * 这意味着它们之间的所有牛的高度都严格小于它们的高度。求每头牛可能的最大高度。
 *
 * 解法分析:
 * 最优解: 差分数组
 * 时间复杂度: O(R + N)
 * 空间复杂度: O(N)
 *
 * 核心思想:
 * 1. 使用差分数组高效标记区间更新
 * 2. 通过前缀和计算最终高度
 * 3. 处理重复关系避免重复计算
 *
 * 相关题目:
 * 1. POJ 3263 - Tallest Cow (当前题目)
 * 2. POJ 2599 - Intervals (差分数组)
 * 3. POJ 3368 - Frequent values (RMQ)
```

```
 * 4. LeetCode 370 - Range Addition (差分数组)
 * 5. LeetCode 1094 - Car Pooling (差分数组)
 * 6. Codeforces 1208D - Restore Permutation (差分数组)
 * 7. AtCoder ABC152F - Tree and Constraints (差分数组)
 * 8. UVa 12086 - Potentiometers (树状数组)
 * 9. HackerRank - Array Manipulation (差分数组)
 * 10. SPOJ - HORRIBLE - Horrible Queries (线段树)
 * 11. 牛客网 NC163 - 机器人跳跃问题 (差分数组)
 * 12. 洛谷 P3372 - 【模板】线段树 1 (线段树)
 * 13. CodeChef - SPREAD - Spread the Word (差分数组)
 *
 * 算法扩展:
 * 1. 二维差分：处理矩形区域的更新操作
 * 2. 带权差分：支持不同权重的区间更新
 * 3. 动态差分：支持在线区间更新和查询
 * 4. 多维扩展：更高维度的差分数组应用
 *
 * 工程化考量:
 * 1. 去重处理：使用 Set 避免重复处理相同关系
 * 2. 边界检查：确保区间端点在合法范围内
 * 3. 内存优化：预分配合适大小的数组
 * 4. 溢出防护：使用 long 防止计算溢出
 */
public static class TallestCow {
    public static int[] tallestCow(int n, int h, int r, int[][] relations) {
        // 存储需要更新的区间，并去重
        Set<String> seen = new HashSet<>();
        List<int[]> intervals = new ArrayList<>();

        for (int[] rel : relations) {
            int a = Math.min(rel[0], rel[1]);
            int b = Math.max(rel[0], rel[1]);
            String key = a + "-" + b;
            if (!seen.contains(key)) {
                seen.add(key);
                intervals.add(new int[]{a, b});
            }
        }

        // 使用差分数组
        int[] diff = new int[n + 2];

        for (int[] interval : intervals) {
```

```
                int a = interval[0];
                int b = interval[1];
                if (a + 1 <= b - 1) {
                    diff[a + 1]--;
                    diff[b]++;
                }
            }

            // 通过前缀和计算最终高度
            int[] heights = new int[n + 1];
            Arrays.fill(heights, h);
            int current = 0;
            for (int i = 1; i <= n; i++) {
                current += diff[i];
                heights[i] += current;
            }

            return heights;
        }

        public static void test() {
            System.out.println("\n=== POJ 3263 - Tallest Cow 测试 ===");

            int n = 6, h = 4, r = 3;
            int[][] relations = {{1, 6}, {2, 4}, {5, 6}};

            int[] heights = tallestCow(n, h, r, relations);
            System.out.print("每头牛可能的最大高度: ");
            for (int i = 1; i <= n; i++) {
                System.out.print(heights[i] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        runMain();
    }
}
```

==========

文件: Experiment.py

```
==========================================
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
财富分配实验与相关算法题目 - Python版本
包含原始财富分配实验和多个扩展题目

时间复杂度分析:
- 原始实验: O(t * n^2) 其中t为轮数，n为人数
- 基尼系数计算: O(n^2) 需要双重循环计算绝对差值和

空间复杂度分析:
- 原始实验: O(n) 存储财富数组和标记数组
- 基尼系数计算: O(1) 仅使用几个变量

工程化考量:
1. 异常处理: 处理输入参数合法性
2. 性能优化: 对于大规模数据可优化基尼系数计算
3. 可测试性: 提供单元测试方法
4. 可扩展性: 模块化设计便于添加新算法

相关题目链接:
1. UVa 11300 - Spreading the Wealth (分金币)
   来源: UVa Online Judge
   链接: https://vjudge.net/problem/UVA-11300
   Java实现: [Experiment.java](Experiment.java)
   Python实现: [Experiment.py](Experiment.py)
   C++实现: [Experiment.cpp](Experiment.cpp)

2. Codeforces 671B - Robin Hood (劫富济贫)
   来源: Codeforces
   链接: https://codeforces.com/problemset/problem/671/B
   Java实现: [Experiment.java](Experiment.java)
   Python实现: [Experiment.py](Experiment.py)
   C++实现: [Experiment.cpp](Experiment.cpp)

3. LeetCode 41 - First Missing Positive (缺失的第一个正数)
   来源: LeetCode
   链接: https://leetcode.com/problems/first-missing-positive/
   Java实现: [Experiment.java](Experiment.java)
   Python实现: [Experiment.py](Experiment.py)
   C++实现: [Experiment.cpp](Experiment.cpp)
```

4. LeetCode 42 - Trapping Rain Water (接雨水)
   来源: LeetCode
   链接: https://leetcode.com/problems/trapping-rain-water/
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp

5. POJ 2155 - Matrix (二维树状数组)
   来源: POJ
   链接: http://poj.org/problem?id=2155
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp

6. UVa 10881 - Piotr's Ants (蚂蚁)
   来源: UVa Online Judge
   链接: https://vjudge.net/problem/UVA-10881
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp

7. POJ 3263 - Tallest Cow (差分法)
   来源: POJ
   链接: http://poj.org/problem?id=3263
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp
"""

```
import random
import math
from typing import List, Tuple
import sys

class Experiment:
    """
    财富分配实验主类

    相关题目:
    1. UVa 11300 - Spreading the Wealth (分金币)
       来源: UVa Online Judge
       链接: https://vjudge.net/problem/UVA-11300
       Java 实现: [Experiment.java](Experiment.java)
```

Python 实现: Experiment.py
C++实现: Experiment.cpp

2. Codeforces 671B - Robin Hood (劫富济贫)
   来源: Codeforces
   链接: https://codeforces.com/problemset/problem/671/B
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp

3. LeetCode 41 - First Missing Positive (缺失的第一个正数)
   来源: LeetCode
   链接: https://leetcode.com/problems/first-missing-positive/
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp

4. LeetCode 42 - Trapping Rain Water (接雨水)
   来源: LeetCode
   链接: https://leetcode.com/problems/trapping-rain-water/
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp

5. POJ 2155 - Matrix (二维树状数组)
   来源: POJ
   链接: http://poj.org/problem?id=2155
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp

6. UVa 10881 - Piotr's Ants (蚂蚁)
   来源: UVa Online Judge
   链接: https://vjudge.net/problem/UVA-10881
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp

7. POJ 3263 - Tallest Cow (差分法)
   来源: POJ
   链接: http://poj.org/problem?id=3263
   Java 实现: Experiment.java
   Python 实现: Experiment.py

```
   C++实现: [Experiment.cpp](Experiment.cpp)
"""

@staticmethod
def run_main():
    """主函数：演示原始实验和扩展题目的使用"""
    print("=== 财富分配实验与相关算法题目 ===")
    print("作者: 算法学习系统")
    print("日期: 2024 年")
    print()

    # 运行原始财富分配实验
    Experiment.run_original_experiment()

    # 运行扩展题目测试
    Experiment.run_extended_problems()

    print("=== 所有测试完成 ===")

@staticmethod
def run_original_experiment():
    """运行原始财富分配实验"""
    print("=== 原始财富分配实验 ===")
    print("一个社会的基尼系数是一个在 0~1 之间的小数")
    print("基尼系数为 0 代表所有人的财富完全一样")
    print("基尼系数为 1 代表有 1 个人掌握了全社会的财富")
    print("基尼系数越小，代表社会财富分布越均衡；越大则代表财富分布越不均衡")
    print("在 2022 年，世界各国的平均基尼系数为 0.44")
    print("目前普遍认为，当基尼系数到达 0.5 时")
    print("就意味着社会贫富差距非常大，分布非常不均匀")
    print("社会可能会因此陷入危机，比如大量的犯罪或者经历社会动荡")

    print("测试开始")
    n = 100
    t = 100000
    print(f"人数: {n}")
    print(f"轮数: {t}")
    Experiment.experiment(n, t)
    print("测试结束")
    print()

@staticmethod
def run_extended_problems():
```

```
    """运行扩展题目测试"""
    print("=== 扩展题目测试 ===")

    # UVa 11300 - Spreading the Wealth
    SpreadingTheWealth.test()

    # Codeforces 671B - Robin Hood
    RobinHood.test()

    # LeetCode 41 - First Missing Positive
    FirstMissingPositive.test()

    # LeetCode 42 - Trapping Rain Water
    TrappingRainWater.test()

    # POJ 2155 - Matrix
    Matrix.test()

    # UVa 10881 - Piotr's Ants
    PiotrAnts.test()

    # POJ 3263 - Tallest Cow
    TallestCow.test()

@staticmethod
def experiment(n: int, t: int):
    """
    原始财富分配实验
    时间复杂度: O(t * n^2) - 外层循环 t 次，内层双重循环
    空间复杂度: O(n) - 存储财富数组和标记数组

    Args:
        n: 人数
        t: 轮数
    """
    # 参数验证
    if n <= 0 or t <= 0:
        raise ValueError("人数和轮数必须大于 0")

    wealth = [100.0] * n
    has_money = [False] * n

    for i in range(t):
```

```
            # 重置标记数组
            has_money = [False] * n

            # 标记有钱的人
            for j in range(n):
                if wealth[j] > 0:
                    has_money[j] = True

            # 有钱的人随机给其他人 1 元
            for j in range(n):
                if has_money[j]:
                    other = j
                    while other == j:
                        other = random.randint(0, n - 1)
                    wealth[j] -= 1
                    wealth[other] += 1

        # 排序并输出结果
        wealth.sort()
        print("列出每个人的财富(贫穷到富有):")
        for i in range(n):
            print(f"{int(wealth[i])} ", end="")
            if i % 10 == 9:
                print()
        print()
        print(f"这个社会的基尼系数为: {Experiment.calculate_gini(wealth)}")

    @staticmethod
    def calculate_gini(wealth: List[float]) -> float:
        """
        计算基尼系数
        时间复杂度: O(n^2) - 双重循环计算绝对差值和
        空间复杂度: O(1) - 仅使用几个变量

        基尼系数公式: G = Σ Σ |xi - xj| / (2n² μ)
        其中 xi, xj 为个人财富，n 为人数， μ 为平均财富

        Args:
            wealth: 财富数组

        Returns:
            基尼系数
        """
```

```
        if not wealth:
            raise ValueError("财富数组不能为空")

        n = len(wealth)
        sum_of_absolute_differences = 0.0
        sum_of_wealth = sum(wealth)

        for i in range(n):
            for j in range(n):
                sum_of_absolute_differences += abs(wealth[i] - wealth[j])

        return sum_of_absolute_differences / (2 * n * sum_of_wealth)


class SpreadingTheWealth:
    """
    UVa 11300 - Spreading the Wealth (分金币)
    来源: UVa Online Judge
    链接: https://vjudge.net/problem/UVA-11300

    题目描述:
    圆桌旁坐着 n 个人，每个人有一定数量的金币，金币总数能被 n 整除。
    每个人可以给左右相邻的人金币，求使得所有人最后的金币数相同的最少转手金币数

    解法分析:
    最优解: 数学推导+中位数
    时间复杂度: O(n log n) - 主要消耗在排序上
    空间复杂度: O(n) - 需要存储 Ci 数组

    核心思想:
    1. 将问题转化为数学规划问题
    2. 通过递推关系得到 Ci = A1+A2+...+Ai - i*M
    3. 利用中位数性质最小化距离和

    相关题目:
    1. LeetCode 462. Minimum Moves to Equal Array Elements II
       来源: LeetCode
       链接: https://leetcode.com/problems/minimum-moves-to-equal-array-elements-ii/
       Java 实现: [Experiment.java](Experiment.java)
       Python 实现: [Experiment.py](Experiment.py)
       C++实现: [Experiment.cpp](Experiment.cpp)

    2. Codeforces 717C - Potions Homework
```

来源: Codeforces
链接: https://codeforces.com/problemset/problem/717/C
Java 实现: Experiment.java
Python 实现: Experiment.py
C++实现: Experiment.cpp
"""

```
@staticmethod
def min_transfer_coins(coins: List[int]) -> int:
    n = len(coins)
    if n <= 1:
        return 0

    # 计算金币总数和平均值
    total = sum(coins)
    average = total // n

    # 计算 Ci 数组
    c = [0] * n
    c[0] = coins[0] - average
    for i in range(1, n):
        c[i] = c[i-1] + coins[i] - average

    # 对 Ci 数组排序，找出中位数
    c.sort()
    median = c[n // 2]

    # 计算最小转移金币数
    result = 0
    for value in c:
        result += abs(value - median)

    return result

@staticmethod
def test():
    print("\n=== UVa 11300 - Spreading the Wealth 测试 ===")

    # 测试用例 1: 平均分布
    coins1 = [100, 100, 100, 100]
    print(f"测试用例 1 - 初始金币: {coins1}")
    result1 = SpreadingTheWealth.min_transfer_coins(coins1)
    print(f"最少转移金币数: {result1}")
```

```
    # 测试用例 2: 示例情况
    coins2 = [1, 2, 5, 4]
    print(f"测试用例 2 - 初始金币: {coins2}")
    result2 = SpreadingTheWealth.min_transfer_coins(coins2)
    print(f"最少转移金币数: {result2}")


class RobinHood:
    """
    Codeforces 671B - Robin Hood (劫富济贫)
    来源: Codeforces
    链接: https://codeforces.com/problemset/problem/671/B

    题目描述:
    有 n 个人，第 i 个人有 ci 枚金币，进行 k 天操作
    每天选择最富有的人(金币最多)给最穷的人(金币最少)1 枚金币
    问 k 天后最富有的人和最穷的人金币数之差的最小值

    解法分析:
    最优解: 二分答案 + 贪心验证
    时间复杂度: O(n log(max_value))
    空间复杂度: O(1)

    核心思想:
    1. 二分最终的最大值和最小值
    2. 验证给定状态是否可达
    3. 利用贪心思想计算操作次数

    相关题目:
    1. LeetCode 1642. Furthest Building You Can Reach
       来源: LeetCode
       链接: https://leetcode.com/problems/furthest-building-you-can-reach/
       Java 实现: [Experiment.java](Experiment.java)
       Python 实现: [Experiment.py](Experiment.py)
       C++实现: [Experiment.cpp](Experiment.cpp)

    2. Codeforces 1363E - Binary Tree Coloring
       来源: Codeforces
       链接: https://codeforces.com/problemset/problem/1363/E
       Java 实现: [Experiment.java](Experiment.java)
       Python 实现: [Experiment.py](Experiment.py)
       C++实现: [Experiment.cpp](Experiment.cpp)
```

```
"""

@staticmethod
def min_difference(coins: List[int], k: int) -> int:
    n = len(coins)
    if n <= 1:
        return 0

    # 计算金币总数
    total = sum(coins)

    # 二分最终的最大值
    left, right = 0, (total + n - 1) // n
    max_val = right
    while left <= right:
        mid = (left + right) // 2
        operations = 0
        for coin in coins:
            if coin > mid:
                operations += coin - mid
        if operations <= k:
            max_val = mid
            right = mid - 1
        else:
            left = mid + 1

    # 二分最终的最小值
    left, right = 0, total // n
    min_val = left
    while left <= right:
        mid = (left + right) // 2
        operations = 0
        for coin in coins:
            if coin < mid:
                operations += mid - coin
        if operations <= k:
            min_val = mid
            left = mid + 1
        else:
            right = mid - 1

    # 特殊情况处理
    if min_val >= max_val:
```

```
            return 0 if total % n == 0 else 1

        return max_val - min_val

    @staticmethod
    def test():
        print("\n=== Codeforces 671B - Robin Hood 测试 ===")

        coins1 = [1, 1, 1, 1]
        k1 = 3
        print(f"测试用例 1 - 初始金币: {coins1}, 操作天数: {k1}")
        result1 = RobinHood.min_difference(coins1, k1)
        print(f"最大值与最小值之差: {result1}")


class FirstMissingPositive:
    """
    LeetCode 41 - First Missing Positive (缺失的第一个正数)
    来源: LeetCode
    链接: https://leetcode.com/problems/first-missing-positive/

    题目描述:
    给你一个未排序的整数数组 nums，请你找出其中没有出现的最小的正整数

    解法分析:
    最优解: 原地哈希
    时间复杂度: O(n)
    空间复杂度: O(1)

    核心思想:
    1. 对于长度为n的数组，缺失的最小正整数一定在[1, n+1]范围内
    2. 将每个正整数i放到数组的第i-1个位置上
    3. 遍历数组找到第一个不符合条件的位置

    相关题目:
    1. LeetCode 448. Find All Numbers Disappeared in an Array
       来源: LeetCode
       链接: https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/
       Java实现: [Experiment.java](Experiment.java)
       Python实现: [Experiment.py](Experiment.py)
       C++实现: [Experiment.cpp](Experiment.cpp)

    2. LeetCode 41. First Missing Positive (相同题目)
```

```
        来源: LeetCode
        链接: https://leetcode.cn/problems/first-missing-positive/
        Java 实现: [Experiment.java](Experiment.java)
        Python 实现: [Experiment.py](Experiment.py)
        C++实现: [Experiment.cpp](Experiment.cpp)
    """

    @staticmethod
    def first_missing_positive(nums: List[int]) -> int:
        n = len(nums)

        # 将每个正整数 i 放到数组的第 i-1 个位置上
        for i in range(n):
            while 1 <= nums[i] <= n and nums[nums[i] - 1] != nums[i]:
                # 交换位置
                temp = nums[i]
                nums[i] = nums[temp - 1]
                nums[temp - 1] = temp

        # 遍历数组找到第一个不符合条件的位置
        for i in range(n):
            if nums[i] != i + 1:
                return i + 1

        return n + 1

    @staticmethod
    def test():
        print("\n=== LeetCode 41 - First Missing Positive 测试 ===")

        nums1 = [1, 2, 0]
        print(f"测试用例 1 - 数组: {nums1}")
        result1 = FirstMissingPositive.first_missing_positive(nums1)
        print(f"缺失的最小正整数: {result1}")


class TrappingRainWater:
    """
    LeetCode 42 - Trapping Rain Water (接雨水)
    来源: LeetCode
    链接: https://leetcode.com/problems/trapping-rain-water/

    题目描述:
```

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

解法分析:
最优解: 双指针法
时间复杂度: O(n)
空间复杂度: O(1)

核心思想:
1. 每个位置能接的雨水量取决于左右两侧最大高度中的较小值
2. 使用双指针从两端向中间移动
3. 维护左右两侧的最大高度

相关题目:
1. LeetCode 407. Trapping Rain Water II
   来源: LeetCode
   链接: https://leetcode.com/problems/trapping-rain-water-ii/
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp

2. LeetCode 11. Container With Most Water
   来源: LeetCode
   链接: https://leetcode.com/problems/container-with-most-water/
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp
"""

```
@staticmethod
def trap(height: List[int]) -> int:
    if not height:
        return 0

    left, right = 0, len(height) - 1
    left_max, right_max = 0, 0
    result = 0

    while left < right:
        if height[left] < height[right]:
            if height[left] >= left_max:
                left_max = height[left]
            else:
                result += left_max - height[left]
```

```
                left += 1
            else:
                if height[right] >= right_max:
                    right_max = height[right]
                else:
                    result += right_max - height[right]
                right -= 1

        return result

    @staticmethod
    def test():
        print("\n=== LeetCode 42 - Trapping Rain Water 测试 ===")

        height1 = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]
        print(f"测试用例 1 - 高度数组: {height1}")
        result1 = TrappingRainWater.trap(height1)
        print(f"能接的雨水量: {result1}")



class Matrix:
    """
    POJ 2155 - Matrix (二维树状数组)
    来源: POJ
    链接: http://poj.org/problem?id=2155

    题目描述:
    给定一个 N*N 的 01 矩阵，初始全为 0，支持两种操作：
    1. 将一个子矩阵中所有元素翻转(0 变 1，1 变 0)
    2. 查询某个位置的值

    解法分析:
    最优解: 二维树状数组 + 差分思想
    时间复杂度: O(logN * logN) 每次操作
    空间复杂度: O(N*N)

    核心思想:
    1. 使用二维树状数组维护差分数组
    2. 区间更新转为 4 个单点更新
    3. 单点查询转为区间查询

    相关题目:
    1. HDU 1195 - Stars
```

   来源: HDU
   链接: http://acm.hdu.edu.cn/showproblem.php?pid=1556
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp

2. Codeforces 1093E - Intersection of Permutations
   来源: Codeforces
   链接: https://codeforces.com/problemset/problem/1093/E
   Java 实现: Experiment.java
   Python 实现: Experiment.py
   C++实现: Experiment.cpp
"""

```
def __init__(self, size: int):
    self.n = size
    self.tree = [[0] * (size + 1) for _ in range(size + 1)]

def lowbit(self, x: int) -> int:
    return x & (-x)

def add(self, x: int, y: int, val: int):
    i = x
    while i <= self.n:
        j = y
        while j <= self.n:
            self.tree[i][j] += val
            j += self.lowbit(j)
        i += self.lowbit(i)

def sum(self, x: int, y: int) -> int:
    result = 0
    i = x
    while i > 0:
        j = y
        while j > 0:
            result += self.tree[i][j]
            j -= self.lowbit(j)
        i -= self.lowbit(i)
    return result

def update_range(self, x1: int, y1: int, x2: int, y2: int):
    self.add(x1, y1, 1)
```

```
        self.add(x1, y2 + 1, 1)
        self.add(x2 + 1, y1, 1)
        self.add(x2 + 1, y2 + 1, 1)

    def query(self, x: int, y: int) -> int:
        return self.sum(x, y) % 2

    @staticmethod
    def test():
        print("\n=== POJ 2155 - Matrix 测试 ===")

        matrix = Matrix(4)
        print("初始状态查询:")
        print(f"matrix[1][1] = {matrix.query(1, 1)}")

        matrix.update_range(1, 1, 2, 2)
        print("翻转区域[(1,1),(2,2)]后查询:")
        print(f"matrix[1][1] = {matrix.query(1, 1)}")



class PiotrAnts:
    """
    UVa 10881 - Piotr's Ants (蚂蚁)
    来源: UVa Online Judge
    链接: https://vjudge.net/problem/UVA-10881

    题目描述:
    一根长度为L厘米的木棍上有n只蚂蚁，每只蚂蚁要么朝左爬，要么朝右爬，速度为1厘米/秒
    当两只蚂蚁相撞时，二者同时掉头(掉头时间忽略不计)
    给出每只蚂蚁的初始位置和朝向，计算T秒之后每只蚂蚁的位置

    解法分析:
    最优解: 等效转换思想
    时间复杂度: O(n log n)
    空间复杂度: O(n)

    核心思想:
    1. 蚂蚁的相对位置在运动过程中不会改变
    2. 碰撞可以看作是蚂蚁互相穿过对方，身份互换
    3. 忽略碰撞直接计算最终位置，然后排序确定状态

    相关题目:
    1. Codeforces 1346A - Color Revolution
```

来源: Codeforces
链接: https://codeforces.com/problemset/problem/1346/A
Java 实现: Experiment.java
Python 实现: Experiment.py
C++实现: Experiment.cpp

2. AtCoder ABC131D - Megalomania
来源: AtCoder
链接: https://atcoder.jp/contests/abc131/tasks/abc131_d
Java 实现: Experiment.java
Python 实现: Experiment.py
C++实现: Experiment.cpp
"""

```
DIR_NAMES = ["L", "Turning", "R"]

class Ant:
    def __init__(self, ant_id: int, position: int, direction: int):
        self.id = ant_id
        self.position = position
        self.direction = direction  # -1: 左, 0: 转身中, 1: 右

    def __lt__(self, other):
        return self.position < other.position

@staticmethod
def get_final_positions(length: int, time: int,
                        positions: List[int],
                        directions: List[str]) -> List[str]:
    n = len(positions)
    if n == 0:
        return []

    # 创建初始状态的蚂蚁数组
    before = []
    for i in range(n):
        direction = -1 if directions[i] == 'L' else 1
        before.append(PiotrAnts.Ant(i, positions[i], direction))

    # 根据初始位置排序
    before.sort()

    # 记录排序后每个蚂蚁在原数组中的索引
```

```
    order = [0] * n
    for i in range(n):
        order[before[i].id] = i

    # 计算每个蚂蚁在 T 秒后的位置（忽略碰撞）
    after = []
    for i in range(n):
        final_pos = before[i].position + before[i].direction * time
        after.append(PiotrAnts.Ant(before[i].id, final_pos, before[i].direction))

    # 根据最终位置排序
    after.sort()

    # 处理碰撞情况
    for i in range(n - 1):
        if after[i].position == after[i+1].position:
            after[i].direction = after[i+1].direction = 0

    # 按照输入顺序生成结果
    result = [""] * n
    for i in range(n):
        idx = order[i]
        ant = after[idx]

        if ant.position < 0 or ant.position > length:
            result[i] = "Fell off"
        else:
            result[i] = f"{ant.position} {PiotrAnts.DIR_NAMES[ant.direction + 1]}"

    return result

@staticmethod
def test():
    print("\n=== UVa 10881 - Piotr's Ants 测试 ===")

    length = 10
    time = 1
    positions = [4, 6, 8]
    directions = ['R', 'L', 'R']

    print(f"木棍长度: {length}, 时间: {time}")
    result = PiotrAnts.get_final_positions(length, time, positions, directions)
    print(f"最终状态: {' '.join(result)}")
```

```
class TallestCow:
    """
    POJ 3263 - Tallest Cow (差分法)
    来源: POJ
    链接: http://poj.org/problem?id=3263

    题目描述:
    有N头牛排成一行，每头牛的高度为H或更低。给出R对关系，每对关系表示第A_i头牛和第B_i头牛能互相看见
    这意味着它们之间的所有牛的高度都严格小于它们的高度。求每头牛可能的最大高度。

    解法分析:
    最优解: 差分数组
    时间复杂度: O(R + N)
    空间复杂度: O(N)

    核心思想:
    1. 使用差分数组高效标记区间更新
    2. 通过前缀和计算最终高度
    3. 处理重复关系避免重复计算

    相关题目:
    1. HDU 1556 - Color the ball
       来源: HDU
       链接: http://acm.hdu.edu.cn/showproblem.php?pid=1556
       Java实现: [Experiment.java](Experiment.java)
       Python实现: [Experiment.py](Experiment.py)
       C++实现: [Experiment.cpp](Experiment.cpp)

    2. Codeforces 1000C - Covered Points Count
       来源: Codeforces
       链接: https://codeforces.com/problemset/problem/1000/C
       Java实现: [Experiment.java](Experiment.java)
       Python实现: [Experiment.py](Experiment.py)
       C++实现: [Experiment.cpp](Experiment.cpp)
    """

    @staticmethod
    def tallest_cow(n: int, h: int, r: int,
                    relations: List[Tuple[int, int]]) -> List[int]:
        # 存储需要更新的区间，并去重
```

```
        seen = set()
        intervals = []

        for rel in relations:
            a, b = min(rel), max(rel)
            key = f"{a}-{b}"
            if key not in seen:
                seen.add(key)
                intervals.append((a, b))

        # 使用差分数组
        diff = [0] * (n + 2)

        for a, b in intervals:
            if a + 1 <= b - 1:
                diff[a + 1] -= 1
                diff[b] += 1

        # 通过前缀和计算最终高度
        heights = [h] * (n + 1)
        current = 0
        for i in range(1, n + 1):
            current += diff[i]
            heights[i] += current

        return heights

    @staticmethod
    def test():
        print("\n=== POJ 3263 - Tallest Cow 测试 ===")

        n, h, r = 6, 4, 3
        relations = [(1, 6), (2, 4), (5, 6)]

        heights = TallestCow.tallest_cow(n, h, r, relations)
        print(f"每头牛可能的最大高度: {' '.join(map(str, heights[1:]))}")



def main():
    """程序入口点"""
    Experiment.run_main()
```

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

==========