蓝桥杯模拟赏金周赛 Round 1 题解

Dashcoding

2025 年 3 月 3 日

总览-比赛概述

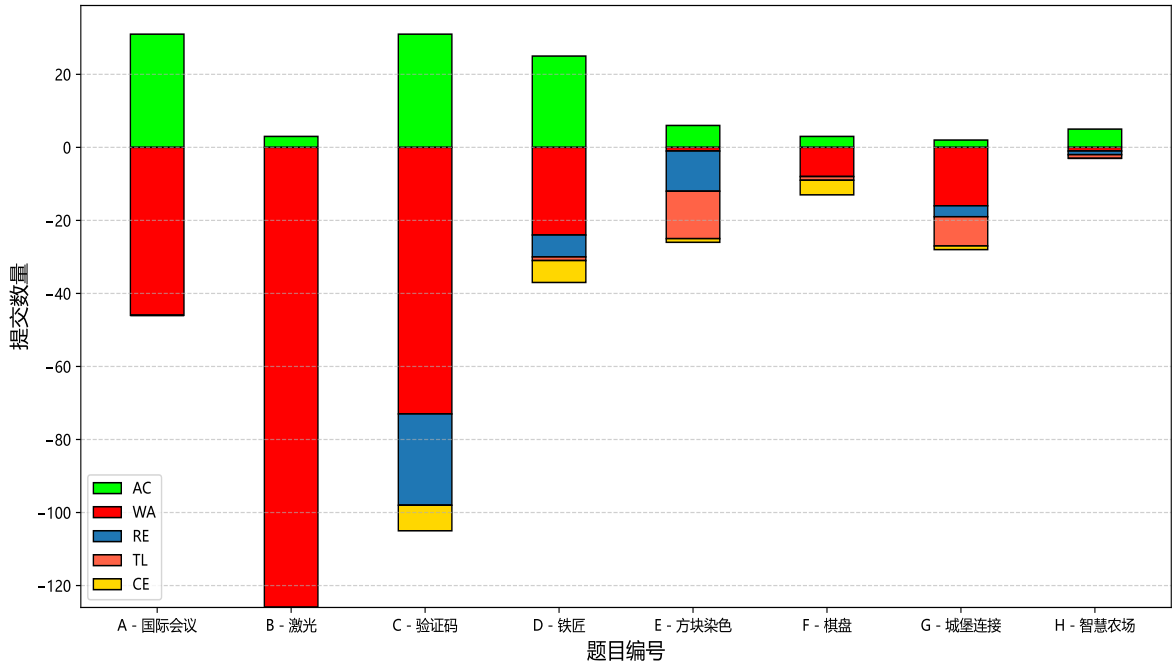
蓝桥杯模拟赏金周赛 Round 1 共有 62 名同学报名参加，其中共有 36 名同学不是 0 分。前三名同学以及其分数如下：

1	☆ 452893411	790	100	100	95	100	100	100	95	100
2	☆ Kato_Shoko	725	95	70	100	100	100	96	69	95
3	☆ 星空	696	74	86	86	95	100	100	60	95

在满分 800 分的前提下，可以看到第三名星空同学的分数已经达到了696分，第一名452893411同学分数达到了惊人的790分。本场赏金赛只有前三名才有奖励，分别是 15 元，10 元，5 元。也就是说如果想在本场比赛拿到赏金，至少要做到696分！

总览-题目分析

蓝桥杯模拟赏金周赛 Round 1 的题目设置完全参考了 2024 年蓝桥杯省赛真题，总共设置有 8 道题目。其中前两道题目为填空题，后面 6 道题目为解答题。题目数据统计如右图所示。



A - 国际会议 - 题目描述

题目大意

一个国际会议有 100 人，包括一些独立研究者和 6 个课题组（编号 1 到 6），各组人数分别是 10、3、5、4、6、4。

- 同一个课题组内部的成员互相不握手。
- 有两对关系不好的课题组：1 号与 3 号，2 号与 4 号，这些组之间的成员也不互相握手。
- 除此之外，其他所有成员之间都正常握手。

求会议中实际发生的握手次数。

DASHCODING

A - 国际会议 - 组合数学

该问题可以通过组合数学中的组合数计算来解决。首先计算总的握手次数，然后减去由于限制条件而无法发生的握手次数。假设

总人数为 100 人，6 个课题组的人数分别为 10、3、5、4、6 和 4。关系不好的课题组对为 1 号小组和 3 号小组，以及 2 号小组和 4 号小组。

1. 总的握手次数: $C(100, 2) = \frac{100 \times 99}{2} = 4950$。
2. 同一课题组内的握手次数:
   $C(10, 2) + C(3, 2) + C(5, 2) + C(4, 2) + C(6, 2) + C(4, 2) = 45 + 3 + 10 + 6 + 15 + 6 = 85$。
3. 关系不好的课题组之间的握手次数:
   $10 \times 5 + 3 \times 4 = 50 + 12 = 62$。
4. 实际发生的握手次数：$4950 - 85 - 62 = 4803$。

B - 激光 - 题目描述

题目大意

在未来实验室中，Alice 用三面高反射率镜子组成一个等边三角形，每边长为 $N = 2025$，顶点标记为 A、B、C。她在 AB 边上的点 P 放置一台激光枪，其中 A 到 P 的距离为 $X = 224$。激光枪发射的光束平行于 BC 方向传播，并在镜子上不断反射。当光束轨迹开始重复时，它最终返回激光枪并被吸收。Alice 希望计算光束路径的总长度。

DASHCODING

B - 激光 - 平面几何 - 几何分析

本题的核心是计算激光光束在等边三角形内的反射路径总长度。激光从点 P 出发，沿着平行于 BC 的方向传播，并在镜子上反射，直到最终返回激光枪并被吸收。为了计算路径总长度，我们需要分析光束的反射规律，并找到一种数学方法来描述其路径。几何分析如下：

1. 等边三角形的边长为 $N = 2025$，顶点为 A、B 和 C。
2. 激光从 AB 边的点 P 出发，$AP = X = 224$，因此 $PB = N - X = 1801$。
3. 光束沿着平行于 BC 的方向传播，这意味着它的初始方向与 BC 平行。

B - 激光 - 平面几何 - 路径长度计算

我们可以观察到反射规律如下：

1. 当光束遇到镜子时，它会按照反射定律反射，即入射角等于反射角。
2. 由于等边三角形的对称性，光束的反射路径会在三角形内形成一系列平行于 BC 的线段。

根据上面的观察，我们可以用以下步骤计算路径长度：

1. 光束的路径可以分解为一系列线段，这些线段的长度与初始线段 PB 相关。
2. 通过递归或数学公式，可以计算出所有线段的总长度。

DASHCODING

B - 激光 - 平面几何 - 示例代码

1. 函数 $f(a, b)$
   1. 该函数通过递归计算光束路径的总长度。
   2. 每次递归都会将问题规模缩小，直到 b 能被 a 整除。
2. 主函数
   1. 读取输入 N 和 X。
   2. 调用 $f(x, n-x)$ 计算路径长度，并将结果与 N 相加，输出最终结果。
   3. 带入 $N = 2025, X = 224$，得答案为：6072。

```cpp
#include <bits/stdc++.h>
using namespace std;

long long f(long long a, long long b) {
    if (a > b) swap(a, b); // 确保 a 是较小的数
    long long d = b / a; // 计算 b 能被 a 整除的次数
    if (b % a == 0)
        return (2 * d - 1) * a; // 如果 b 能被 a 整除，返回结果
    return d * 2 * a + f(a, b % a); // 否则，递归处理余数部分
}

int main() {
    long long n, x;
    cin >> n >> x; // 读取输入 N 和 X
    cout << n + f(x, n - x) << endl; // 计算并输出结果
    return 0;
}
```

DASHCODING

C - 验证码 - 题目描述

题目大意

判断一个字符串是否满足如下格式：

- 以大写字母开头。
- 接一个 100000 至 999999 的六位数字。
- 以大写字母结尾。

数据范围：

- S 由大写字母和数字组成，长度在 1 至 10 之间。

C - 验证码 - 模拟

该题目要求我们验证一个字符串。字符串必须以一个大写字母开头，后跟一个六位数字（范围在 100000 到 999999 之间），并以另一个大写字母结尾。为了验证字符串是否符合这个模式，我们需要进行以下几个步骤的检查：

1. 检查字符串的长度：验证码的总长度必须是 8 个字符（1 个大写字母 + 6 位数字 + 1 个大写字母）。如果字符串的长度不是 8，那么它肯定不符合验证码的模式。
2. 检查首尾字符：字符串的第一个字符和最后一个字符都必须是大写字母。我们可以通过检查字符是否在'A' 到'Z' 的范围内来判断。
3. 检查中间六位字符：字符串的中间六位字符都必须是数字，并且第一个数字不能是'0'（因为六位数的范围是 100000 到 999999，所以第一位不能为 0）。

D - 铁匠 - 题目描述

题目大意

将长度为 L 的铁棒切割成 12 段，每段长度为正整数：

- 需在 11 个不同位置切割
- 切割方法不同，当某位置在一方法中切割而在另一方法中未切割

计算不同的切割方法数。答案不超过 2^{63}。

数据范围：

- $12 \leq L \leq 200$
- L 为整数

DASHCODING

D - 铁匠 - 组合数学 - 思路分析

这个题目可以转化为一个经典的组合数学问题。我们需要将一根长度为 L 的铁棒切割成 12 段，每段的长度都必须是正整数。由于每段的长度至少为 1，因此我们可以将问题转化为在 $L-1$ 个可能的位置中选择 11 个位置进行切割。具体来说，假设铁棒的

长度为 L，我们需要在 $L-1$ 个可能的位置中选择 11 个位置进行切割。由于每段的长度都必须是正整数，因此切割的位置必须至少相隔 1 个单位。这相当于在 $L-1$ 个位置中选择 11 个位置进行切割，因此总的切割方法数就是组合数 $C(L-1,11)$。

D - 铁匠 - 组合数学 - 公式解释

组合数 $C(n, k)$ 的计算公式为：

$$C(n, k) = \frac{n!}{k!(n-k)!}$$

在本题中，我们需要计算 $C(L-1, 11)$，即：

$$C(L-1, 11) = \frac{(L-1)!}{11!(L-12)!}$$

为了高效地计算这个组合数，我们可以使用递推公式：

$$C(n, k) = \frac{n \times (n-1) \times \cdots \times (n-k+1)}{k \times (k-1) \times \cdots \times 1}$$

这个公式可以避免计算大数的阶乘，从而减少计算量。最终时间复杂度为 $O(11)$。

D - 铁匠 - 组合数学 - 额外思考

本题因为数据较大，所以在 C++ 中我们需要使用 long long，在 Java 中我们需要使用 long，而不是 int 来表示数值。

而如果数值进一步增大，超过 64 位整数所能表达的范围，应该如何做呢？或者说让我们把最终超过 64 位整数的结果 mod 一个数输出，如何去做呢？

DASHCODING

E - 方块涂色 - 题目描述

题目大意

N 个方块排成一列，需涂上颜色：

- 每方块可选颜色 1 至 M（可不全用）。
- 相邻方块相同颜色对数不超过 K。

计算不同涂色方式总数（某方块颜色不同即为不同），对 998244353 取模。

数据范围：

- $1 \leq N, M \leq 2 \times 10^5$
- $0 \leq K \leq N - 1$

E - 方块涂色 - 思路分析

我们可以使用组合数学中的排列组合公式来计算符合条件的涂色方式。具体来说，我们可以将问题分解为计算在 N 个方块中，恰好有 i 对相邻方块颜色相同的涂色方式数，其中 i 从 0 到 K。

对于每一对相邻方块颜色相同的情况，我们可以将其视为一个“约束”，即这些相邻方块的颜色必须相同。

E - 方块涂色 - 组合数学

将问题分解为组合数学问题。对于每个可能的相邻相同对数 $i(0 \le i \le K)$，计算对应的方案数：

1. 在 $n-1$ 个相邻位置中选择 i 个位置作为颜色相同的位点，组合数为 $C(n-1, i)$。
2. 第一个方块有 M 种颜色选择。
3. 剩下的 $n-1-i$ 个相邻位置必须颜色不同，每个位置有 $M-1$ 种选择，总共有 $(M-1)^{(n-1-i)}$ 种可能。

将所有 i 对应的方案数累加，得到最终结果。

DASHCODING

E - 方块涂色 - 代码实现

由于 N 和 M 的范围较大，我们需要对算法进行优化。我们可以通过预处理阶乘和逆阶乘来快速计算组合数，从而减少计算量。

最终答案为:

$$\text{ans} = \sum_{i=0}^{K} \binom{n-1}{i} \times M \times (M-1)^{n-1-i} \mod 998244353$$

最终时间复杂度可以表示为: $O(NlogN)$

F - 棋盘 - 题目描述

题目大意

在 N 个格子的棋盘上玩游戏，格子编号 1 至 N：

- 每个格子 i 有整数 C_i
- 给定排列 $P_1, P_2, \ldots, P_N$
- 从某格子开始，可移动 1 至 K 次
- 移动规则：从 i 到 P_i，得分加 C_{P_i}

求游戏结束时的最大得分（初始得分 0）。

数据范围：

- $2 \le N \le 5000$
- $1 \le K \le 10^9$
- $-10^9 \le C_i \le 10^9$

F - 棋盘 - 思路分析

我们需要找到在最多进行 K 次移动后能获得的最大得分。每个格子按照排列 P 移动，每次移动到 P_i 并累加对应格子的得分 C_{P_i}。关键在于分析排列 P 形成的环结构，并计算每个环在不同移动次数下的最大得分。

F - 棋盘 - 思维枚举

解决思路：

1. 环结构分析：排列 P 将格子划分为若干不相交的环。每个环的移动具有周期性，可以独立处理。
2. 前缀和与周期累加：对于每个环，计算移动 s 次后的得分前缀和。若环的总得分为正，则可以通过多次循环该环增加总分。
3. 枚举所有可能：遍历每个起始点，分析其所在环的所有可能移动次数 s，结合剩余次数计算最大得分。

DASHCODING

F - 棋盘 - 代码实现

算法解释：

1. 环检测：对于每个起始点，找到其所在的环，并记录移动过程中得分的前缀和。
2. 前缀和计算：维护一个前缀和数组 vec，记录每次移动后的累计得分。
3. 周期累加：若环的总得分为正，则剩余移动次数可以用于多次循环该环，计算对应的最大得分。

最终答案为：

$$\text{max\_score} = \max_{s \in [0, \min(\text{size}, K)]} \left(\text{vec}[s] + \left\lfloor \frac{K - s - 1}{\text{size}} \right\rfloor \times \text{sum} \right)$$

时间复杂度可表示为 $O(N^2)$

G - 城堡连接 - 题目描述

题目大意

在一个环形城镇中有 N 座城堡（编号 0 至 $N-1$），初始无道路连接：

- 提供 M 种魔法，每种连接 x 与 $(x + A_i) \mod N$，耗魔力 C_i
- 可多次施放任意魔法

需要判断：

- 判断能否使所有城堡连通
- 若能，求最小魔力值；否则输出 -1

数据范围：

- $2 \le N \le 10^9$
- $1 \le M \le 10^5$
- $1 \le A_i \le N-1, 1 \le C_i \le 10^9$

DASHCODING

G - 城堡连接 - 思路分析

在解这个环形城镇连通问题的过程中，我们首先规定，不能对同一对 (i, x) 重复施放魔法来连接顶点 x 和顶点 $(x + A_i) \mod N$。为什么要这样规定呢？因为题目要求通过施放 M 种魔法（每次连接 x 和 $(x + A_i) \mod N$，消耗 C_i 魔力）使所有城堡连通，并求最小魔力值。如果对同一对 (i, x) 重复连接，相当于多次添加同一条边，这不仅浪费魔力（增加不必要的成本），而且对图的连通性毫无贡献——边已经存在，再加一次并不会改变连通分量。因此，这一限制既符合题目优化的本质，也不会影响最终答案。

G - 城堡连接 - 思路分析

为了求解，我们定义 G_t 为一个图，其中对于 $i = 1, 2, \ldots, t$ 和 $x = 0, 1, 2, \ldots, N-1$ 的每一对，顶点 x 与 $(x + A_i) \mod N$ 之间有一条成本为 C_i 的边。换句话说，G_t 汇聚了前 t 种魔法所能构建的所有边，每条边的成本对应施放魔法的魔力值。题目要求的最小魔力值，实际上等价于在 G_M（包含所有 M 种魔法允许的边）中找到一个最小生成树，使得所有 N 个城堡连通。

DASHCODING

G - 城堡连接 - 最小生成树

Kruskal 算法是求最小生成树的经典工具。它从一个只有顶点没有边的图 T 开始，按边成本从小到大的顺序，依次尝试添加 G 中的边，前提是加入后连通分量数减少 1（即边“可添加”）。算法复杂度为 $O(E \log E)$，E 是边数。然而，这里 G_M 有 NM 条边（N 可达 10^9，M 可达 10^5），直接运行 Kruskal 算法会超时。因此，我们需要一种更高效的方式来模拟这一过程。

为此，我们先将 C_i 按升序排序，确保 $C_1 \le C_2 \le \cdots \le C_M$。在 Kruskal 算法的思路下，我们从小成本开始，逐步添加边到 T 中：先尽量用操作 1（成本 C_1）添加边，再用操作 2（成本 C_2），依此类推，直到操作 M。设 X_t 为 G_t 的连通分量数，则每次“尽量用操作 t 添加边”时，边数增加 $X_{t-1} - X_t$。初始时，$X_0 = N$（每个城堡独立）。最终，总魔力值为 $\sum_{t=1}^{M} C_t(X_{t-1} - X_t)$。若最后 $X_M > 1$，说明 G_M 未连通，无法形成生成树，答案为 -1。

G - 城堡连接 - 最小生成树

现在，核心问题在于如何快速计算 X_t。在 G_t 中，对于 $i = 1, 2, \ldots, t$，从顶点 x 可以沿边移动到 $(x + A_i) \mod N$。因此，顶点 v 到 w 可达的条件是：存在整数 $k_1, k_2, \ldots, k_t$，使得 $w \equiv v + k_1A_1 + k_2A_2 + \cdots + k_tA_t \pmod N$。这一条件可以简化：

- $w \equiv v + k_1A_1 + k_2A_2 + \cdots + k_tA_t \pmod N$，
- 等价于 $w = v + k_0N + k_1A_1 + k_2A_2 + \cdots + k_tA_t$（加入模 N 的倍数），
- 进一步整理为 $w = v + kd_t$（k 为某整数），
- 即 $w \equiv v \pmod{d_t}$，其中 $d_t = \gcd(N, A_1, A_2, \ldots, A_t)$。

这表明，v 和 w 在同一连通分量，当且仅当它们模 d_t 同余。环形城镇有 N 个顶点，每个连通分量的大小为 d_t（因为步长为 d_t 的周期覆盖），所以 $X_t = \frac{N}{d_t}$。通过高效计算每一阶段的 gcd，我们就能快速得出答案。

G - 城堡连接 - 为什么 $X_t = \frac{N}{d_t}$

魔法操作形成周期性连接，步长由 $A_1, A_2, \ldots, A_t$ 和 N 的最大公约数 d_t 决定。

样例 2（$N = 6, A_1 = 3$）中，$d_1 = \gcd(6, 3) = 3$，$X_1 = 6/3 = 2$，连通分量数大于 1，印证无法全连通。

DASHCODING

G - 城堡连接 - 复杂度分析 - 时间复杂度

时间复杂度：
题解通过模拟 Kruskal 算法，利用连通分量数 $X_t = \frac{N}{d_t}$（其中 $d_t = \gcd(N, A_1, \ldots, A_t)$）计算最小魔力值 $\sum_{t=1}^{M} C_t(X_{t-1} - X_t)$。
主要步骤如下：

- **排序 C_i 和 A_i**：$O(M \log M)$，M 种魔法按成本排序。
- **计算 d_t**：M 次 gcd，每次 $O(\log N)$，总计 $O(M \log N)$。
- **计算 X_t 和总和**：$O(M)$，每次除法和乘加为 $O(1)$。
- **总复杂度**：
  $O(M \log M) + O(M \log N) + O(M) = O(M \log M + M \log N)$。

数据范围（$M \leq 10^5$，$N \leq 10^9$）下，约为 4.7×10^6 次操作，高效可行。

G - 城堡连接 - 复杂度分析 - 空间复杂度

空间复杂度：

- **输入数组**：存储 A_i 和 C_i，需要 $O(M)$。
- **辅助数组**：d_t 和 X_t 可递推计算，仅需 $O(1)$ 额外空间（若存储所有 t 的值则为 $O(M)$）。
- **总空间**：$O(M)$，主要由输入数据决定。

H - 智慧农场 - 题目描述

题目大意

在智慧农场中，有 N 块农田，初始产量为 $A = (a_1, a_2, \ldots, a_N)$，需执行 Q 条指令：

- 类型 1：`1 L R x`，将区间 $[L, R]$ 的 a_i 改为 $\lfloor x/a_i \rfloor$
- 类型 2：`2 L R y`，将区间 $[L, R]$ 的 a_i 设为 y
- 类型 3：`3 L R`，计算并输出区间 $[L, R]$ 的总产量

数据范围：

- $1 \le N \le 5 \times 10^5$
- $1 \le Q \le 10^5$
- $1 \le a_i, y \le 10^5$
- $2 \le x \le 10^5$

DASHCODING

H - 智慧农场 - 思路分析

在设计解法之前，我们先分析指令的特性，特别是指令 1 的难点。指令 2 和指令 3 相对简单：重新种植是区间赋值，统计产量是区间求和，标准的**懒惰标记线段树**（简称 lazyseg）就能胜任。但指令 1（`1 L R x`）却是个挑战。为什么呢？

- **普通 lazyseg 的局限**：lazyseg 依赖操作的可合并性，而 $\lfloor x/a_i \rfloor$ 的整数除法依赖初始值，无法通过标记直接传播。试想，若连续两次执行指令 1（如 x_1 和 x_2），结果 $\lfloor x_2/\lfloor x_1/a_i \rfloor \rfloor$ 无法简化为单一标记，这与 lazyseg 的要求不符。
- **“减半”效应**：每次执行指令 1，a_i 变为 $\lfloor x/a_i \rfloor$。由于 $x \geq 2$、$a_i \geq 1$，$\lfloor x/a_i \rfloor \leq a_i/2$（甚至更小，若 $x < a_i$）。这意味着 a_i 每次至少减半，且最多 $\lceil \log_2 10^5 \rceil \approx 17$ 次就会降至 0。一旦 $a_i = 0$，后续指令 1 不再影响它，直到指令 2 重新赋值。这一特性是解题的关键。

H - 智慧农场 - 数据结构

结合题目需求和指令 1 的“减半”效应，我们设计如下方案：
我们用两种数据结构协同工作：

- std::set：管理“值相同且非零”的连续区间，例如 $[l, r]$ 内所有农田产量均为 v。
- lazyseg：维护整个数组 A，支持区间赋值（指令 2）和区间求和（指令 3）。

我们可以用如下的初始化操作：

- **初始化**：扫描初始数组 A，将连续相同值的段（如 $[1, 3]$ 为 2）加入 set，视作 N 次指令 2，同步更新 lazyseg。

H - 智慧农场 - 数据结构

那么我们对于各个指令的处理流程如下：

- **指令 2（`2 L R y`）**：
  - 在 set 中移除 $[L, R]$ 内的旧区间（可能需要分裂或截断）。
  - 插入新区间 $[L, R]$，值为 y。
  - 在 lazyseg 上执行范围赋值，复杂度 $O(\log N)$。
- **指令 3（`3 L R`）**：
  - 直接用 lazyseg 查询 $[L, R]$ 的和，复杂度 $O(\log N)$。
- **指令 1（`1 L R x`）**：
  - 遍历 set，找出 $[L, R]$ 内所有非零区间。
  - 对每个区间（假设值为 v），计算新值 $\lfloor x/v \rfloor$。
  - 用 lazyseg 更新这些区间的值，若新值为 0，则从 set 中移除该区间。

H - 智慧农场 - 使用 std::set 时间复杂度分析

我们再来分析 std::set 的时间复杂度分析

- set **中的区间数量**：
  - 初始最多 N 个点，每条指令 2 最多净增 1 个区间（可能分裂为 3 段，但边界有限）。
  - 指令 1 只减少区间（值变 0 时移除）。
  - 每个点最多被 $\lceil \log \max(a_i) \rceil + 1 \approx 18$ 次指令 1 影响（从 10^5 降到 0）。
  - 总区间数为 $N + O(Q)$。
- **操作复杂度**：set 插入/删除为 $O(\log N)$，lazyseg 更新/查询为 $O(\log N)$。
- **总时间**：初始化 $O(N \log N)$，Q 次指令每次 $O(\log N)$ 操作，结合“减半”次数，总复杂度为 $O((N+Q) \log N \log \max(a_i))$。