# 第 50 届 ICPC 亚洲区域赛 (成都站) 题解

## The 2025 ICPC Asia Chengdu Regional Contest - Tutorial

电子科技大学出题组

UESTC

2025 年 10 月 26 日

# 概况

- Easy: G, J
- Easy-Medium: A, B, D, L
- Medium: C, K, M
- Medium-Hard: H, I
- Hard: E, F

# Problem J - Judging Papers

## 题目大意

给出多个不同的评委对若干篇论文的评分, 你需要进行 Rebuttal 来调整每篇论文的分数, 使得能被接收的论文数量最多.

- *Shortest Solution: 367 bytes (Python)*

分数已经满足要求的不管. 对每一篇没满足分数要求的论文, 检查 rebuttal 之后是否满足. 满足的话就进行 rebuttal, 然后剩余次数减一. 直到所有的论文都检查完或者 rebuttal 次数用完为止.

# Problem G - GCD of Subsets

## 题目大意

给出 $n, k, m$. 定义集合 $S = \{1, 2, \ldots, n\}$. 每次可以选择若干个数组成一个新的集合 (要求新集合的 $\gcd = k$) , 然后从 $S$ 中删除这些数. 操作之前可以选择最多 $m$ 个数字修改成 $1 \sim n$ 内任意数字. 问最多可以进行多少次操作? $n, k, m \leq 10^{18}$.

- *Shortest Solution: 186 bytes (Python)*

- 不失一般性, 考虑 $k = 1$ 的情况 (当 $k > 1$ 时, 能够选择的数只有 $k$ 的倍数) .
- 考虑 $m = 0$ 的情况. 由数论中的经典结论 $\gcd(a, a + 1) = 1$ 可知, 此时最好的选择方案是: $\{1\}, \{2, 3\}, \{4, 5\}, \ldots$ 来选择, 那么集合个数就是 $\lceil n/2 \rceil$.
- 再考虑 $m > 0$. 如果 $k > 1$, 那么此时优先修改非 $k$ 的倍数以及没有用上的 $k$ 的倍数 (最多一个) , 将其修改成 $k$ 以形成单独一个数的集合. 然后再将大小为 2 的集合内的数进行修改 (此时每修改两个数才能多贡献一个集合) .
- 进行上述简单的分类讨论即可. 时间复杂度: $\mathcal{O}(1)$

# Problem A - Arrow Painting

## 题目大意

给定一个长度为 $n$ 的数组 $a$, 请尝试构造出一个由非负整数构成的 $b$ 数组, 使得 $\frac{b_i}{\mathrm{sum}(b)}$ 四舍五入到小数点后两位后的值等于 $a_i$, 如果不存在这样的 $b$ 数组, 请报告这一事实.

- *Shortest Solution: 827 bytes (Cpp)*

- 如果没有“四舍五入”这步运算, 就应该有 $S = \sum_{i=1}^{n} a_i = 1$ 成立.
- 考虑“四舍五入”本身对这个求和的影响, 它对一个非零数的提升不超过 $0.5\%$, 对一个小于 $1$ 数的削减会严格小于 $0.5\%$, 因此设 $k$ 是数组 $a$ 中非零数的个数, $l$ 是小于 $1$ 的数的个数, 则数组 $a$ 的实际总和 $S$ 应该满足如下两个条件:

$$\begin{cases} S - 0.005k \leq 1 \\ S + 0.005l > 1 \end{cases}$$

- 设 $T$ 为题目输入的 $a$ 数组的总和, 则上面的条件可以改写成:

$$\begin{cases} T - 0.5k \leq 100 \\ T + 0.5l > 100 \end{cases}$$

为了让式子只有整数运算, 同时便于处理, 可以证明这个条件和下面这个条件是等价的:

$$\begin{cases} 2T - k \le 200 \\ 2T + n > 200 \end{cases}$$

上面的式子就是验证存在性的判定条件. 下面通过构造来证明, 输入满足上面这个条件时存在数组 $b$. 先将 $b$ 数组赋值成输入的 $a$ 数组, 然后分 $T > 100$ 和 $T \le 100$ 两种情况进行讨论:

- 当 $T > 100$ 时将任意 $(2(T - 100))$ 个非零的 $b_i$ 减少 $0.5$, 从而将 $b$ 数组的和调整成 $100$ 的同时使得每个 $b_i$ 的占比经过四舍五入后为 $a_i$, 再将 $b$ 数组整体乘以 $10$ 使得数组为整数数组;
- 当 $T \le 100$ 时, 所有的 $b_i$ 都肯定不超过 $99$ , 此时给 $b$ 数组每个数都增加 $\frac{100-T}{n}$, 从而将 $b$ 数组的和调整成 $100$ 的同时使得每个 $b_i$ 的占比经过四舍五入后为 $a_i$, 再将 $b$ 数组整体乘以 $n$ 使得数组为整数数组.

可以证明经过上述操作得到的 $b$ 数组是题目所需数组. 最终复杂度为 $O(n)$, 常数比较小.

# Problem B - Blood Memories

## 题目大意

$n$ 个人, 每个人可以选择是否操作花费 $c_i$ 的费用造成 $a_i$ 的伤害, 每回合费用上限为 $m$, 如果一个人本回合操作了, 下回合费用会临时增加 $k$, 问 $T$ 回合能造成的最大伤害之和. $n \leq 6, T \leq 10^9$

- *Shortest Solution: 1316 bytes (Cpp)*

- 因为 $n$ 很小, 所以可以直接暴力 $2^n$ 压缩每个人每回合是否进行操作.
- 设 $f_{i,s}$ 为前 $i$ 回合, 其中第 $i$ 回合选择的操作状态为 $s$ 可以造成的最大伤害之和.
- 设 $A_s = \sum_{i \in s} a_i$, $C_s = \sum_{i \in s} c_i$
- 转移条件即为 $f_{i,s} = \max_{C_s + k \times popcount(s \& t) \le m}(f_{i-1,t} + A_s)$
- 对于 $T$ 很大的情况, 直接 $O(4^n)$ 预处理所有状态 $s, t$ 之间是否能够转移, 然后矩阵快速幂优化 $dp$ 即可.
- 复杂度 $O(8^n \log T)$

# Problem D - Deductive Snooker Scoring

## 题目大意

基于简化的斯诺克规则, 要求从当前局面推断一条可能的击球过程, 即从开局出发, 通过一系列成功或失败的击球操作, 到达给定的局面.

- *Shortest Solution: 798 bytes (Cpp)*

状态定义:(scoreA, scoreB, remBalls, player, redPhase)

- scoreA:A 的当前得分
- scoreB:B 的当前得分
- remBalls: 当前桌上剩余的球数 (包括红球和彩球)
- player: 当前击球方 (0 表示 A, 1 表示 B)
- redPhase: 当前是否处于“刚打进红球、等待彩球”阶段 (0 否, 1 是)

状态转移建图:

- 任何时候击球失败
  $(a, b, n, p, x) \to (a, b, n, 1-p, 0)$
- 成功打进红球 (redPhase $= 0$ 且 remBalls $> 6$)
  $(a, b, n, p, 0) \to (a + [p = 0], b + [p = 1], n-1, p, 1)$
- 打进红球后成功打进彩球 (redPhase $= 1$)
  $(a, b, n, p, 1) \to (a + [p = 0] \times c, b + [p = 1] \times c, n, p, 0), \quad c = 2, 3, \ldots, 7$
- 清彩阶段 (redPhase $= 0$ 且 remBalls $\leq 6$)
  $(a, b, n, p, 0) \to (a+[p = 0]\times c, b+[p = 1]\times c, n-1, p, 0), \quad c = 8-n$

搜索路径:

- 题目转换成从 $(0, 0, 21, 1, 0)$ 到给定的 $(A, B, n, p, 0)$ 或 $(A, B, n, p, 1)$ 的任意一条路径
- 使用你喜爱的任何方法, 比如 DFS、BFS、for 循环递推等方式, 搜索出符合条件的路径
- 以 $(0, 0, 21, 1, 0)$ 为起点, 可以通过一次搜索预处理出各点的前驱点/边, 查询时从该点逆向找到回到起点的路径

时间复杂度:$O(S + T \cdot L)$, 其中 $S$ 为状态总数, $T$ 为数据组数, $L$ 为路径长度

# Problem L - Label Matching

## 题目大意

给定一棵有根树, 树上每个节点有 $a_i$ 和 $b_i$, 若为 $0$ 则表示通配符. 你需要对每棵子树独立回答: 若你能无限次交换该子树中两个点的 $a_i$, 是否存在一种操作序列使得 $a_i = b_i$ 对该子树内所有点成立.

$1 \leq n \leq 2 \times 10^5$, $0 \leq a_i, b_i \leq n$.

- *Shortest Solution: 1450 bytes (Cpp)*

- 对于一棵子树, 无限次交换相当于能够任意重排子树内的 $a_i$, 问题等价于判断两个含通配符的多重集是否相等. 同时, 最优策略肯定是: 非通配符先匹配, 再让通配符和剩下的非通配符匹配, 最后通配符之间匹配.
- 对当前子树 $T_i$, 维护 $ca[v]$ 表示 $T_i$ 中 $a_k = v$ 的节点数量, $cb[v]$ 表示 $T_i$ 中 $b_k = v$ 的节点数量.
- 由剩余的非通配符 $a$ 值总数, 不超过 $b$ 的通配符总数, 有

$$\sum_{v=1}^{n} \max(0, ca[v] - cb[v]) \leq cb[0]$$

- 由剩余的非通配符 $b$ 值总数, 不超过 $a$ 的通配符总数, 有

$$\sum_{v=1}^{n} \max(0, cb[v] - ca[v]) \leq ca[0]$$

事实上, 这两个条件是等价的, 我们只需 check 其中一个, 另一个自然满足.

以第一个条件为例, 代入 $cb[0] = |T_i| - \sum_{v=1}^{n} cb[v]$, 得到最终的判断条件

$$\sum_{v=1}^{n} \max(ca[v], cb[v]) \leq |T_i|$$

利用 dsu on tree 维护每棵子树的 $\sum_{v=1}^{n} \max(ca[v], cb[v])$, 时间复杂度 $O(n \log n)$.

# Problem C - Crossing River

## 题目大意

一条河上只有一艘船, 两侧各有 $n$ 和 $m$ 个人要到对岸 (每个人到达岸边时间可能不同) . 船一次性只能载一个人, 且船从某岸划到对岸所需时间固定为 $k$. 你可以自己决定船的起始岸, 求出一种载人方案, 使得最后一个人到达对岸的时间最小. $n, m \leq 10^5, k \leq 10^9$

- *Shortest Solution: 1069 bytes (Cpp)*

- 不妨将两侧按时间排序，并设 $n, m$ 同阶．策略一定先送到达时间早的．
- 考虑二分答案，如何判断能否在 $T$ 时刻送完．
- 顺着做我们发现有难度，考虑倒着做．此时我们枚举最后一次是送左边还是送右边，然后我们发现每次船往回开，到某一岸，我一定是送这一岸没有送过的最迟的，容易 $O(n)$ 判断．
- 此时我们发现二分答案也是不必要的，直接倒着做，此时每次到达某一岸有一个时间 $t$，如果 $t$ 小于这一岸没有送过的人的到达时间，我们直接弥补缺少的时间．这样最小时间和方案都可以 $O(n)$ 模拟得到．最后复杂度瓶颈在排序，复杂度 $O(n \log n)$．

# Problem K - K-Coverage

## 题目大意

给定数轴非负半轴上 $n$ 条长为 $L$ 的线段, 第 $i$ 条线段覆盖 $[l_i, l_i + L - 1]$. 你可以移动至多一条线段, 问最多有多少个整点被覆盖恰好 $k$ 次. $1 \leq n \leq 2 \cdot 10^5$, $1 \leq L, k \leq n$, $0 \leq l_i \leq 2 \times n$.

- *Shortest Solution: 2462 bytes (Cpp)*

- 利用差分和前缀和, 容易处理出 $d[i]$ 表示点 $i$ 初始被线段覆盖的次数.
- 当一条覆盖点 $i$ 的线段被移走时, 如果原先 $d[i] = k$, 答案减一; 如果原先 $d[i] = k + 1$, 答案加一.
- 于是定义 $x[i] = [d[i] = k + 1] - [d[i] = k]$, 表示点 $i$ 被覆盖次数减一时, 对答案的贡献.
- 同理定义 $y[i] = [d[i] = k - 1] - [d[i] = k]$, 表示点 $i$ 被覆盖次数加一时, 对答案的贡献.
- 现在分两种情况计算移动一条线段的贡献. (以下称移动前线段为 $I_{\text{old}}$, 移动后线段为 $I_{\text{new}}$)

# $I_{\text{old}}$ 和 $I_{\text{new}}$ 不相交

此时, 答案变化量可以表示为

$$\Delta_{\text{ans}} = \sum_{i \in I_{\text{old}}} x[i] + \sum_{i \in I_{\text{new}}} y[i]$$

我们枚举移动哪条线段, 因此 $I_{\text{old}}$ 已知, 预处理 $x[i]$ 的前缀和即可算出第一项.

至于第二项, 因为要求 $I_{\text{old}}$ 和 $I_{\text{new}}$ 不相交, 类似处理出" 结束位置 $\leq i$ 的所有长为 $L$ 的区间中 $\sum y[i]$ 的最大值", 和" 开始位置 $\geq i$ 的所有长为 $L$ 的区间中 $\sum y[i]$ 的最大值", 也能很快回答.

# $I_{\text{old}}$ 和 $I_{\text{new}}$ 部分相交

以线段向右平移为例，答案变化量为

$$\begin{aligned}\Delta_{\text{ans}} &= \sum_{i=l_{\text{old}}}^{l_{\text{new}}-1} x[i] + \sum_{i=l_{\text{old}}+L}^{l_{\text{new}}+L-1} y[i] \\ &= \sum_{i=l_{\text{old}}}^{l_{\text{new}}-1} (x[i] + y[i+L])\end{aligned}$$

对于固定的 $l_{\text{old}}$，我们要在 $l_{\text{new}} \in [l_{\text{old}}+1, l_{\text{old}}+L-1]$ 范围内最大化这个和. 将 $x[i]+y[i+L]$ 取前缀和，我们肯定是选择上述范围内让前缀和最大的 $l_{\text{new}}$. 注意到每次询问的区间长度固定，是滑动窗口问题，可以用单调队列线性求解.

线段向左平移同理. 总时间复杂度 $O(n)$.

# Problem M - Meeting for Meals

## 题目大意

在一张无向带权图上 $k$ 个人到 $1$ 号节点会合. 在最早的约饭时间前提下, 对每个人分别求出最大的有人陪伴时间. $k \leq n \leq 3 \cdot 10^5, m \leq 10^6$

- *Shortest Solution: 1881 bytes (Cpp)*

- 先求出 1 号节点到每个点距离，从而确定会合的时间 $t_{\text{meeting}}$ 即为朋友位置到 1 号节点距离的最大值.
- 对于每个人来说，在第一次遇上别人之后可以一直走在一起，求最大陪伴时间等价于求最小碰面时间.

- 下面记 $\mathrm{dis}[i][j]$ 表示 $i, j$ 之间的距离.
- 考虑一个人 $i$ 到达了 $u$, 另外一人 $j$ 到达了 $v$，他俩在点边 $(u, v, w)$ 上碰面（在节点碰面看成特殊的在边上碰面）。

$$\mathrm{dis}[i][u] + w + dis[1][v] \leq t_{\mathrm{meeting}} \text{ (碰面之后返回 v)}$$

或者

$$\mathrm{dis}[j][v] + w + dis[1][u] \leq t_{\mathrm{meeting}} \text{ (碰面之后返回 u)}$$

- 对应的碰面所需要时间为 $\frac{\mathrm{dis}[j][v]+\mathrm{dis}[i][u]+w}{2}$, 对应的陪伴时间为 $t_{\max} - \frac{\mathrm{dis}[j][v]+\mathrm{dis}[i][u]+w}{2}$.

- 显然 dis$[j][v]$ 越小越好, 即只需要关心除了 $i$ 以外第一个到达节点 $v$ 的人所需要花费的时间.
- 而另一方面如果 $j$ 比 $i$ 先到达节点 $u$, $i$ 如果经过节点 $u$ 再去碰面肯定不如直接在此与 $j$ 碰面. 因此我们不需要考虑 $i$ 到达 $u$ 节点之后的答案更新.
- 综合上述两点, 对于每个节点, 我们只需要关注第一个到达这个节点的人就行.
- 通过跑多源最短路, 求出到达每个点的最小时间, 和对应是哪个人先到达该节点. 再通过之前的公式计算答案即可.
- 时间复杂度 $O((n+m)\log n)$

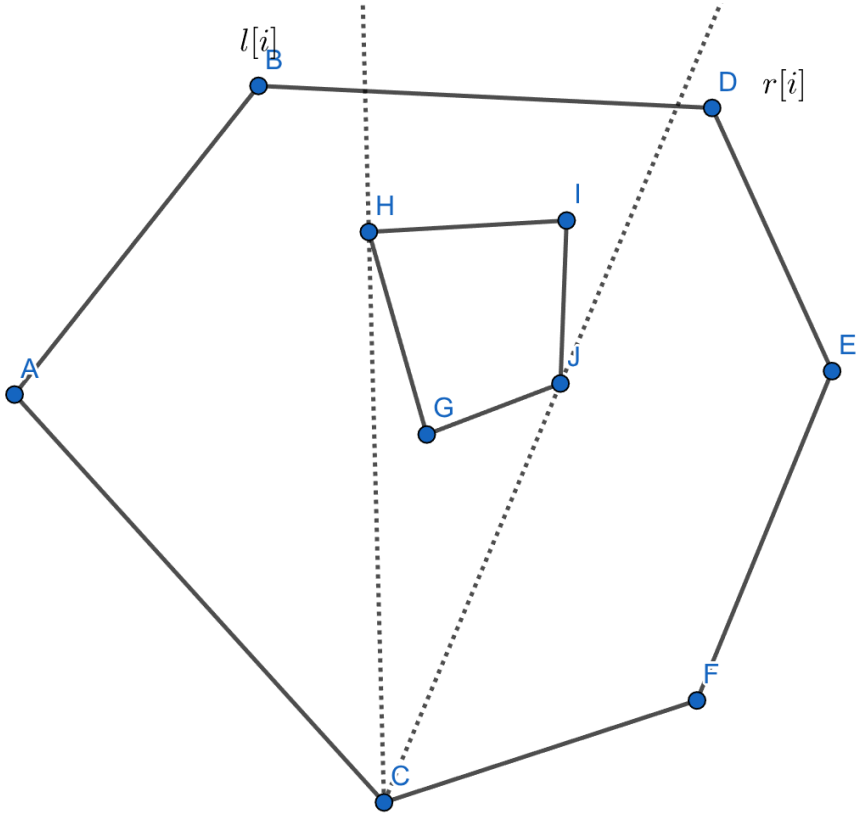
# Problem I - Inside Polygon

## 题目大意

给定两个凸包, 小凸包严格包含于大凸包内. 先需在大凸包上选三点, 三点构成三角形也包含小凸包. 问选点方案数. $n \leq 3 \cdot 10^5$

- *Shortest Solution: 1911 bytes (Cpp)*

对于大凸包上的每个点 $i$, 求出其往前往后连边的连边的范围 $l[i]$ 和 $r[i]$.



- 当逆时针遍历大凸包上的点 $i$ 时, 其在小凸包上的切点也在顺时针旋转, 于是被切线限制的 $l[i]$ 和 $r[i]$ 对应点也在不断顺时针旋转, 于是我们可以通过双指针求出 $l$ 和 $r$ 数组.
- 后续进入计数过程.
- 一个 $\triangle UVW$ (三点逆时针顺序) 符合条件等价于小凸包在直线 $UV$, 直线 $VW$, 直线 $WU$ 的左侧.
- 对于每个点 $i$, 以其为顶点能构成的三角形数量即为区间 $[l[i], i-1]$ 中选一点 $V$ , 区间 $[i+1, r[i]]$ 中取一点 $U$, 满足小凸包在直线 $UV$ 左侧的方案数. 记该数量为两个区间的配对数量.

- 考虑逆时针遍历大凸包上的点 $i$, 我们动态维护区间 $A = [l[i], i-1]$ 与区间 $B = [i+1, r[i]]$ 点对的配对数量.
- 从点 $i$ 转移到点 $i+1$ 时, 区间变化为
  - $A : [l[i], i-1] \to [l[i+1], i]$
  - $B : [i+1, r[i]] \to [i+2, r[i+1]]$

  对应的操作分别是
  - 往 $A$ 中加入点 $i$, 删除点 $l[i], ..., l[i+1]-1$
  - 往 $B$ 中删除点 $i+1$, 加入点 $r[i]+1, r[i]+2, ..., r[i]$
- 例如考虑往区间 $B$ 加入一点 $u$, 配对数量的增加量为区间 $[u+1, r[u]]$ 与区间 $A$ 重合点的数量. 区间重合点数量可以分类讨论 $O(1)$ 计算.
- 所有点能构成三角形之和即为答案的三倍.
- 复杂度 $O(n+m)$

# Problem H - Heuristic Knapsack

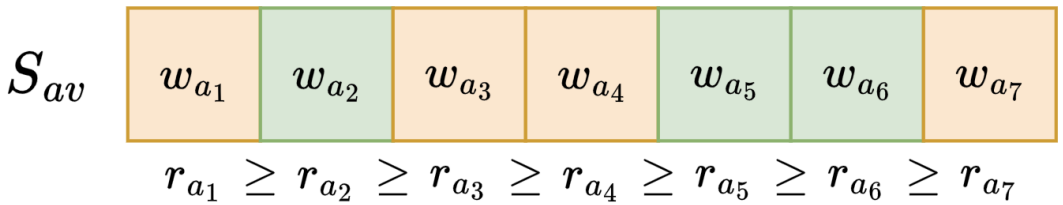
## 题目大意

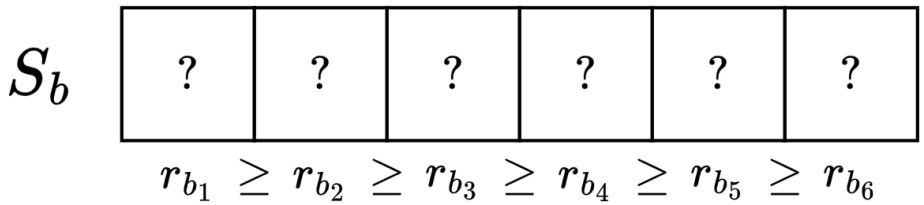
给出一个有 $n$ 个物品以及背包上限的 0-1 背包问题实例, 但是每个物品的价值 $(r_i)$ 和重量 $(w_i)$ 中最多有一个值缺失. 需要给缺失的值构造方案, 使得 Alice 按照“重量最低优先”与 Bob 按照“价值最高优先”两种启发式算法得到的物品集合完全相同. $n \leq 3000, w_i \leq 10^9, r_i \leq 10^9$.

- *Shortest Solution: 4653 bytes (Cpp)*

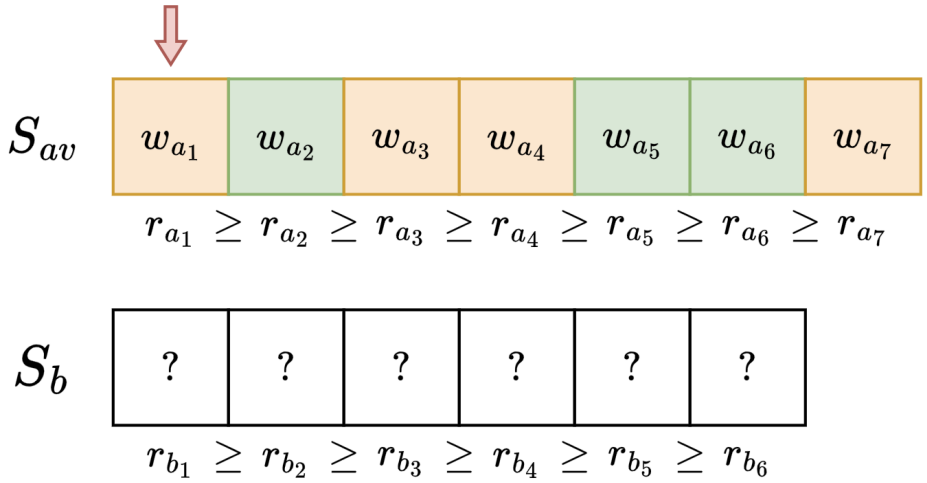
- 为了方便, 下文中所有的比较都默认带上了下标.
- 我们将物品分成两个集合 $S_a, S_b$, 其中 $S_a$ 是 $w_i$ 已知的物品的集合, $S_b$ 是 $w_i$ 未知物品的集合.
- 另外, $S_a$ 中的物品都按照 $w_i$ 从小到大排序, $S_b$ 中的物品都按照 $r_i$ 从大到小排序.
- 那么对于 Alice 来说, 她选择的物品集合一定是 $S_a$ 的一个前缀.
- 这样可以启发我们: 枚举 $S_a$ 的前缀, 作为 Alice 在 $S_a$ 中恰好要选择的物品集合, 然后去模拟 Bob 的选择, 检查剩下的物品.

- 既然要模拟 Bob 的选择, 那么就需要考虑 $S_a$ 中的物品选择.
- 构造一个新的集合 $S_{av}$, 其中物品与 $S_a$ 相同, 但是按照 $r$ 从大到小排序.
- 那么举个例子. 如下图, 绿色表示枚举的前缀的物品, 剩下的黄色的这些物品是不能被选择的.




- 对于绿色物品, 如果 $r_i$ 缺失, 那么将其设为 $10^9$ 即可；对于黄色物品, 如果 $r_i$ 缺失, 那么将其设为 $1$ 即可.




- 如图, 从 $S_{av}$ 第一个物品 $a_1$ 开始选择.
- 此时这个物品是不能被选进集合的, 那么就需要由 $S_b$ 中的物品来把 $a_1$ 抵消.
- 此处可以贪心, 即 $w_{b_1}$ 选得越大越好, 这样可以尽快占满背包, 后续的物品就设为上限 $(10^9)$ , 让双方都不能拿到即可.

- 那么接下来就需要考虑 $w_{b_1}$ 的上限. 分为若干种情况:
  1. $w_{b_1} \le W_r$, 其中 $W_r$ 表示剩余背包大小. 注意一开始 $W_r$ 的值为 $W - \sum_{i \in \text{green}} w_i$, 即总体积减去枚举的前缀体积.
  2. $w_{b_1} < w_{\text{nxt}(a_{\text{pre}})}$, 其中 pre 表示目前枚举的前缀个数, 那么 $\text{nxt}(a_{\text{pre}})$ 就表示枚举的前缀的后一个位置. 这是因为前缀后一个位置是黄色物品中 $w_i$ 最小的, 那么为了防止 Alice 去选择这个物品, $S_b$ 中要被选择的物品的 $w$ 就不能大于这个值.
  3. 题目中给出的构造上限 $10^9$. 但此处存在一些 case (后续提到) , 这里构造上限可以设为 $10^9 - 1$.
- 这样, 如果 $w_{b_1}$ 的上限 $= 0$, 说明该物品不能被选择, 将其 $w_{b_1}$ 设为构造上限 $10^9$ 即可.
- 另外注意此处 $w_{b_1}$ 如果不能选择的话不能直接判定无解, $S_b$ 中后面的物品可能是合法的.
  - 具体原因在于, $b_1$ 虽然 $r_{b_1}$ 比后面的大, 但是被上面计算的上限所限制, 再加上 $b_1$ 位置靠后, 故可能优先级还不如后面的物品.

由于本题构造上限与输入上限相同, 因此会存在一些 case 需要注意, 下面列举一些可能会遇到的情况:

- 背包容量为 $10^9$ 时, 即使把重量设置为 $10^9$ 也有可能被取到 (见样例 3) . 此时一个 trick 是在 check 的时候将重量上限设置为 $10^9 - 1$, 这样比起 $w = 10^9$ 而言 Alice 会更优先选择 $w = 10^9 - 1$ 的物品.
- 若 $S_a = \varnothing$, 此时可以将 $S_b$ 中价值最大的物品的 $w$ 设为 $\leq W$ 的任意数, 其他的物品的体积设置则需要严格大于该数字.
- 上述过程构造完毕之后可以使用 $\mathcal{O}(n \log n)$ 直接做一遍题面所给的判定, 可能可以少讨论其他一些构造上的 case.

枚举前缀需要 $\mathcal{O}(n)$, 后续的检查需要 $\mathcal{O}(n)$ 或 $\mathcal{O}(n \log n)$, 故总时间复杂度: $\mathcal{O}(n^2)$ 或 $\mathcal{O}(n^2 \log n)$.

# Problem E - Escaping from Trap

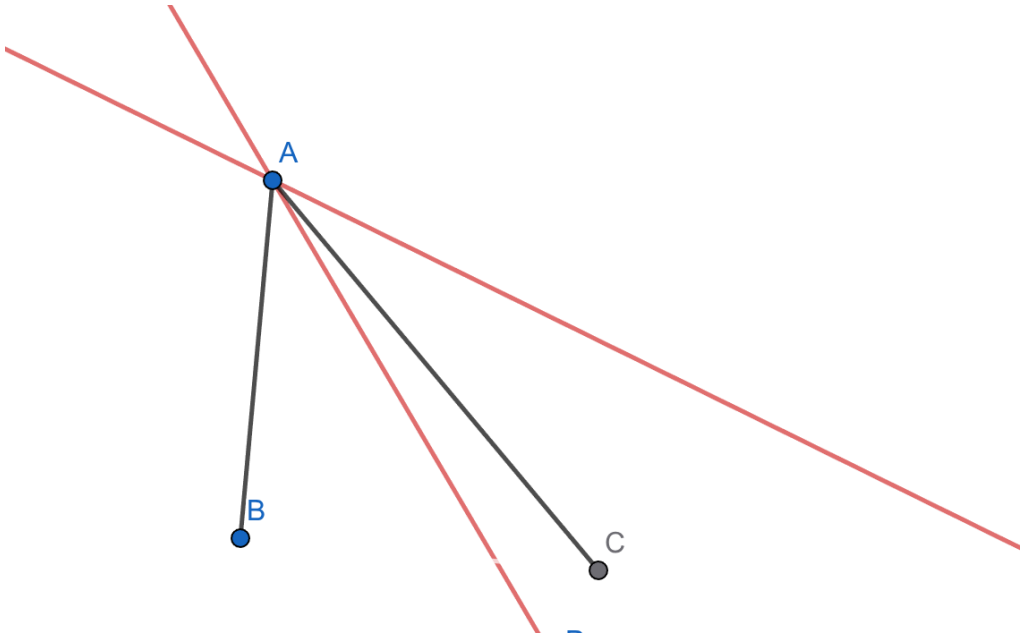
## 题目大意

给定正 $n$ 边形, 大小未知, 和一个位置位置点 $P$ . 5 次交互, 每次查询多边形上两点和 $P$ 组成的三角形面积. 求 $P$ 到多边形中心距离.

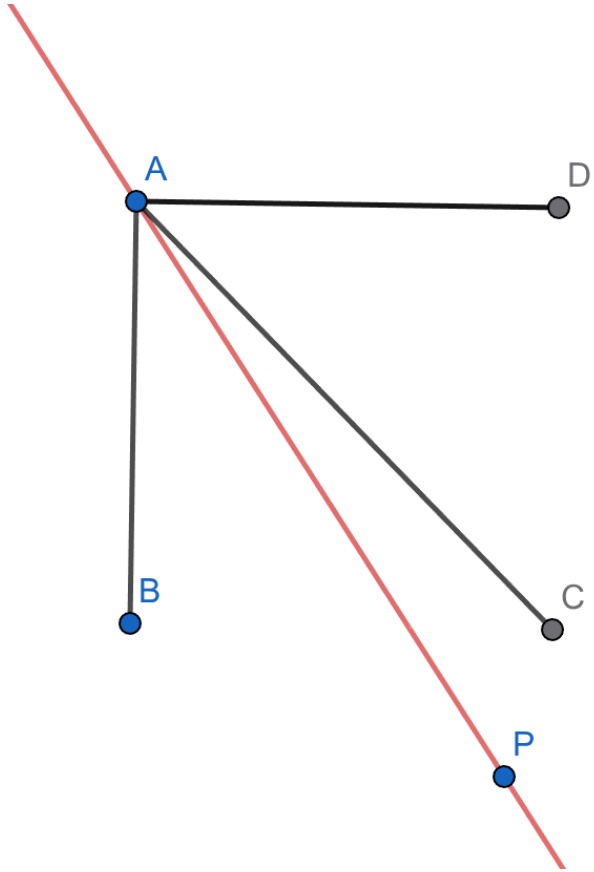
- *Shortest Solution: 2349 bytes (Cpp)*

先假设正 $n$ 边形外接圆的半径为 1，只需要求出 $P$ 点位置，根据缩放比例确定答案.

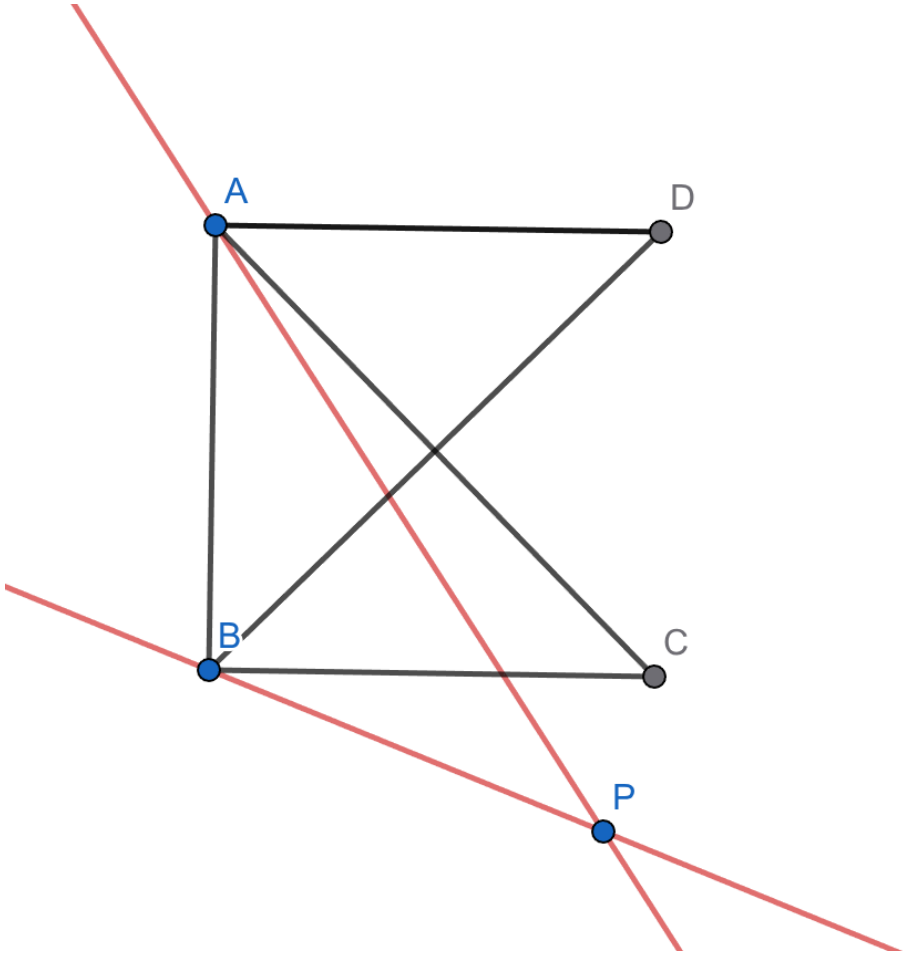
确定 $\triangle PAB$ ，$\triangle PAC$ 面积之后，根据面积比例可以把 $P$ 限制在两条直线上，如下图.



再多知道 $\triangle PAD$ 的面积后, 即可确定直线 $PA$.



如下图询问, 可以得到直线 $PA$, $PB$, 此时如果 $P$ 不在直线 $AB$ 上, 即可唯一确定 $P$ 点位置.



- 先分别询问 $\triangle PAB$, $\triangle PAC$, $\triangle PBC$ 的面积, 肯定有一者不为 $0$. 例如 $\triangle PAB$ 不为 $0$, 接下来就可以询问 $\triangle PDA$ , $\triangle PDB$ 的面积, 从而唯一确定点 $P$.
- 上述过程描述的是确定询问的过程, 而根据询问确定点 $P$ 可以使用公式

$$S_a\overrightarrow{PA} + S_b\overrightarrow{PB} + S_c\overrightarrow{PC} = \mathbf{0}$$

其中

$$S_a = S_{\triangle PBC}, \quad S_b = S_{\triangle PCA}, \quad S_c = S_{\triangle PAB}$$

- 注意上述公式的面积是有向面积, 需要枚举正负号再检查是否符合条件.
- 上述做法可以任取多边形上四个, 但建议不要选取较近的点, 这样在 $n$ 比较大的时候精度会比较糟糕.

# Problem F - Following Arrows

## 题目大意

机器人从左上角 $(1,1)$ 出发, 每一步先按当前格子箭头方向移动 (若会出界则不动), 然后将移动前所在格子的箭头翻转为相反方向.
输入 $k$, 目标是构造一个迷宫, 使得机器人恰好在 $k$ 步后首次到达右下角 $(N,M)$. 要求 $N,M \leq 8$.

- *Shortest Solution: 5147 bytes (Cpp)*

# Problem F - Following Arrows

这题我们分为几个步骤

1. 我们先构造一个网格使得步数能够达到 $10^6$ 级别，这个可以考虑先构造一条尽量长的从左上到右下的路径，然后路径上每个格子如果翻转，都会走到路径更之前的格子上，这样路径上每一个点的第一次到达时间会以极快的速度增长，我们将所有的格子翻转，就可以得到一个 $10^6$ 规模的网格，这里给一个 std 的例子

```
RRRRRRRD
DDLLLLLL
DRRRRRRD
DDLLLLLL
DRRRRRRD
DDLLLLLL
DRRRRRRD
DDLLLLLL
```

# Problem F - Following Arrows

2. 再观察我们如果将路径上某个格子的方向翻转后，对答案的影响，我们会发现，第一次走到这个格子的时候，路径前面的格子的方向是确定的，翻转这个格子到之前的格子后，经过一定步数又回到这个格子时，路径前面的格子的方向跟第一次没有区别，因此路径上格子翻转对步数的影响是独立的
3. 有了 2 的结论后，我们可以计算出每个格子翻转对步数的影响，然后通过背包的方式去组合出所有可能的步数
4.3 的方式凑步数会发现，不一定能组合出 1 到 $10^6$ 所有的数，但是可以通过构造多个初始网格来覆盖所有的数