题解

T1 分组（group）

算法一

暴力枚举所有划分方式并检验，时间复杂度为贝尔数，期望得分 10 分。

算法二

定义 g_s 表示 S 中的人划分为一组是否可行（可行为 1，不可行为 0 ）

定义 f_S 表示集合 S 内部的答案，转移即

$$f_S=\sum_{T\subseteq S,g_T=1}f_{S-T}$$

用枚举子集优化，时间复杂度为 $O(3^n)$，期望得分 20 分。

算法三

搜索剪枝或高次多项式复杂度的算法，期望得分 40 分。

算法四

由于所有人是等价的，不妨按 a 从小到大排序。

限制仅由每组中 a 最小的人产生，即确定 a 最小的人后，此后的人仅需关心于数量。

定义 $f_{i,j}$，表示确定前 i 个人分组情况且之后有 j 个人所在分组在其中，转移即

$$\begin{cases}j\cdot f_{i-1,j}\to f_{i,j-1} & j\ge 1\\ \forall t\in[1,a_i],\frac{f_{i-1,j}}{(t-1!)}\to f_{i,j+t-1} & i+j+t-1\le n\end{cases}$$

时间复杂度为 $O(n^3)$，期望得分 70 分。

算法五

定义 $f_{i,j}$ 表示确定了人数 $\le i$ 的组且有 j 个 $a_t>i$ 的人加入，设有 x 个 $a_t=i$ 和 y 个 $a_t>i$，转移即

$$\forall t\in\left[\left\lceil\frac{max(x-j,0)}{i}\right\rceil,\left\lfloor\frac{x+y-j}{i}\right\rfloor\right],\frac{(j+it)!}{(j+it-x)!}\frac{f_{i-1,j}}{t!(i!)^t}\to f_{i,j+it-x}$$

根据调和级数，时间复杂度为 $O(n^21ogn)$，期望得分 100 分。

T2 奇迹（miracle）

电荷守恒只是必要条件。我们尝试用初始状态构造出终末状态，构造不出就是无解。

想象这样一个场景：你手头有一张借记卡（不能欠款），你会不停地收入一些钱，花掉一些钱，当一次消费的金额大于卡内余额的时候，这次消费就不能进行了。

到这个题里，我们也可以开一张「借记卡」。

我们将本题的一些概念用金融中的术语表达：m_i 可以认为是第 i 笔账单的数量，a_i 可以认为是每笔账单的支出（因此初态中的第 i 块金属可以变成 m_i 个单价为 a_i 的商品）；同理，b_i 可以认为是每笔账单的收入（因此终态中的第 i 块金属可以变成 m_i 张每张票面金额为 b_i 的支票）。

首先将初始状态的 n 块金属按比荷升序排序，终态的 n 块金属也按比荷升序排序。

接下来我们搞两个指针 p, q，刚开始 p 指向初始态的第一块金属，q 指向终态的第一块金属。

我们现在可以用 q 指向的支票去买 p 指向的商品。当然商家很刁钻，一张支票只能买一个商品。

如果一张支票的金额大于一个商品的钱，我们就可以将多余的钱（或者说是电荷）存进银行卡，如果一张支票的金额小于一个商品的钱，我们就需要从银行卡里取些钱（电荷）了。

（p 指向的商品数和 q 指向的支票数可能不一定相等，不过这不是问题，p 空的时候就将 p 向后移动，q 空的时候也将 q 向后移动就行了，别忘了总商品数和总支票数是相等的）

因为我们手里拿的是借记卡，所以任何时候余额都不能为负。

如果我们顺利地走完了上面的整个过程（也就是说没有赊账的情况发生），说明有解。否则无解。

T3 渔猎（fishing）

横纵坐标是独立的。

我们发现一对顶点的作用是把 X 和 Y 坐标划分成了两个部分，使得这两个部分不可能同时被覆盖到，且具体覆盖到哪个区域我们可以自己指定。于是开线段树，对值域进行哈希。以 X 坐标为例，若两个顶点的 X 坐标分别是 $p, q(p < q)$，则可知 $[p, q)$ 和 $[0, p) \vee [q, w)$ 必然不可能同时被覆盖，我们需要给它们不同的哈希值。

最后哈希值相同的部分大概率能被一起覆盖，那么分别取 X, Y 坐标里出现次数最多的哈希值，乘起来就行了。

这个大概率是多大呢？如果你用 32 位随机数，这个概率确实不够大，用 64 位随机数就可以了。

T4 影袭（strike）

猜猜这个题的人物为什么是桑格莉娅？

贴模型走路是一个很实用的技巧，可以在这类网格路径题中有效地去除一些重复冗余状态。

比如这道题，定义首末两行、两列为**边缘**部分，其他为**非边缘**部分，我们可以发现无论怎么在非边缘部分添加障碍，都存在至少两条合法路径，如下图：

```
ooooo
o.x.o
oxx.o
ox.xo
ooooo
```

`x` 代表障碍，`o` 代表边缘部分，`.` 是普通的空地。我们发现只要存在边缘部分，我们就不用管普通的空地。

但是边缘部分可能会改变。比如在一个空的 5×5 网格中添加一个障碍：

```
oooX.
o.ooo
o...o
o...o
ooooo
```

不难发现障碍的作用是“如果处在原来的边缘部分，就把这个边缘部分挪个地方到左下角/右上角”，或者说“如果处在原来的边缘部分，则令自己左下角/右上角不可能再为边缘部分”。到底哪个角就要看自己原先处在左下角的边缘还是右上角的边缘。如果同时处在两类边缘，那说明没有路径可以走了，此时输出 `Yes`，我们也顺便获得了是否有路径的充要条件。

边缘的修改是可以均摊的，因为每个点最多被一类边缘用一次，于是写一个事件驱动模拟。其中需要找最大最小值，set 常数大，换成懒惰删除的堆就可以了。