odekeke

算法 1

考虑 $c=0$ 的情况。此时 R 的限制可以忽略，只需要和 P,Q 取 $\min$，显然有一个背包 DP：设 $f(i,j,k)$ 表示前 i 个小球放入后 A 玩家的袋子的重量为 j 方洞的重量为 k 的方案数。

转移的时候枚举每一个小球是放哪个洞里就可以了，因为有类型的限制，可以先枚举这一个类型是选择哪一个玩家进行转移。

时间复杂度 $O(nm^2)$，期望得分 $10\ pts$。

算法 2

实际上把小球先划分给 A/B 玩家还是先把小球划分给方洞和圆洞都是没有影响的，把这两维独立考虑。

设 $f(i,j)$ 表示前 i 个球 A 玩家袋子重量为 j 的方案数，$g(i,j)$ 表示前 i 个球方洞的重量为 j 的方案数。

最后合并一下答案就为：

$$\sum_{i=S-P_2}^{P_1}\sum_{j=S-Q_2}^{Q_1} f(n,i)g(n,j)$$

分别背包的时间复杂度为 $O(nm)$，对 g 做一个前缀和优化，合并复杂度是 $O(m)$ 的，期望得分 $30\ pts$。

算法 3

现在有 $c=1$ 的情况，暴力枚举哪一个掉进魔法洞，剩下的小球就转换成 $c=0$ 的情况，直接套用算法 2。

时间复杂度 $O(n^2m)$，结合前面期望得分 $40\ pts$。

算法 4

方案数背包是支持撤回的，整体求出算法 2 的 DP 之后枚举哪一个掉进魔法洞直接把 for 循环倒过来做一遍就可以了，相当于新定义两个 DP f',g' 分别表示排除这个小球的方案数。

时间复杂度 $O(nm)$，期望得分 $100\ pts$。

lcp

考虑求 $\min$，即 $\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n}\min\limits_{k=1}^{M}\mathrm{LCP}(s_{i,k},s_{j,k})$。

考虑 $M=1$：建出 trie，LCP 即两个串 LCA 深度，也就是统计树上以某个点为 LCA 的点对个数问题。

现在 $M > 1$。考虑对每个 s_i，先将 s_i 中的 M 个字符串都只保留其中最短那个的长度，再将 $s_{i,j}$ 一位一位地依次串起来，得到字符串 r_i（例如：$s_i = \{\text{abcde}, \text{fghi}, \text{xxxxxx}\}$, 先将它们长度都变成 4，$\{\text{abcd}, \text{fghi}, \text{xxxx}\}$，然后串起来，$r_i = \text{afxbgxchxdix}$）。然后将 r_i 放进 trie 中，和 $M = 1$ 类似的，对树上每个点计算出以它为 LCA 的点对个数，它对答案的贡献是 $\lfloor \frac{depth}{M} \rfloor$（$depth$ 为该点深度）。

其它带 $\log$ 的常数不大的做法也都可以通过。

matrix

考虑先将修改操作和查询操作分开考虑。

修改操作：

可以将原来的矩阵视为 n^2 个四元组：$(i, j, a_{i,j}, b_{i,j})$。

那么将矩阵 a, b 每行向右循环移位 x 格，等价于将四元组变为 $(i, (j + x) \mod n, a_{i,j}, b_{i,j})$。每列向下循环移位同理。

将每行逆排列等价于：$b'_{i,a_{i,j}} = b_{i,j}$ ，$a'_{i,a_{i,j}} = j$，那么四元组会变为 $(i, a_{i,j}, j, b_{i,j})$，也就是交换 2,3 项。

将每列逆排列，同理，四元组会变为 $(a_{i,j}, j, i, b_{i,j})$，也就是交换 1,3 项。

令 x, y, z 是 $i, j, a_{i,j}$ 任意排列后的结果，那么这个四元组一定形如 $(x + \Delta x, y + \Delta y, z + \Delta z, b_{i,j})$ ，实时维护其表达形式。 查询操作： 考虑没有修改操作时怎么做。

给每行赋予一个随机值 w_i，用于哈希，预处理 $s_{i,j}$ 表示 b 矩阵第 i 列，等于 j 的行的随机值的和。

然后枚举第 i 行，判断是否满足条件， $\sum_{j=1}^{n} s_{j,a_{i,j}} - n \times w_i = (\sum_{k=1}^{n} w_k - w_i) \times K$ ，就可以得到所有满足条件的行。

如果四元组变为了 $(x + \Delta x, y + \Delta y, z + \Delta z, b_{i,j})$，那应该怎么做？$z, \Delta z$ 不影响答案，因为只关心 b 矩阵的变化，Δy 不影响答案，循环向右移位不影响对位相等的数量，Δx 只影响行的边编号。对于每个 $i, j, a_{i,j}$ 在 x, y, z 中的排列顺序，预处理出一个 set 表示用 $(x, y, z, b_{i,j})$ 四元组表示矩阵，所有满足的行的编号，再利用 Δx 在 set 中二分查找最小的满足条件的行编号。

时间复杂度 $O(n^2 + q \log n)$

elevator

先不管修改操作，考虑离线：

考虑这个求 LIS 的算法：维护 a_i, b_i, c_i，表示 i 到当前位置，长为 $1, 2, 3$ 的上升序列里结尾的最值。

加入当前的数 x 时，对每个 i 会进行下列操作：

- 若$x \le a_i$，$a_i \leftarrow x$
- 否则，若 $x \le b_i$，$b_i \leftarrow x$
- 否则，若 $x \le c_i$，$c_i \leftarrow x$
- 否则，当前位置是询问 $l = i$ 的答案。

a_i 的更新就是吉司机线段树，实际上 b_i, c_i 也同理。

线段树上维护 $mx(a), smx(a)$，表示 a 最大、次大值，考虑更新线段树上的结点：

- 若 $x > mx(a)$：a 没变化，更新 b, c。
- 若 $smx(a) < x \le smx(a)$：令$mx(a) = x$，对 $smx(a)$ 更新 b 和 c。
- 否则，递归进左右子树操作。

b, c 也需要维护 mx 和 smx，实现起来比较繁琐，但还是能维护。

考虑修改操作：

把每个询问按时间顺序放到线段树上，每个操作按 i 分类，并记录时间。

依次枚举下标 i：把 $l = i$ 的询问加入线段树。修改操作直接做，再把已经找到答案的询问从线段树上删掉。

复杂度：$O((n+q)\log n)$ 。