1001 Submission

显然的，对于每个人 x，其对应的优先级在大脑内的时间是一段区间，并且这个区间的左端点一定是 x 第一次出现的位置。由于我们只关心这个区间内 x 的出现次数，我们不妨钦定区间的右端点也是某个 x 出现的位置。这样所有可能的区间被减少到了 $O(n)$ 个。

于是题意变成了，有一堆区间，每个区间有一个代价，即能跳过的提交记录个数（区间中 x 的出现个数 -1）。你需要选择一些，使得每个 x 对应的区间恰好选一个，不存在一个 $x.5$ 的位置被覆盖 $> k$ 次，使得总代价最大。

这是一个较为经典的问题，有如下费用流做法：对于每个位置建一个点，每个点向下一个点连一条流量 k 费用 0 的边；对于每个区间，从 l 到 r 建一条流量 1 费用为相应代价的边；对于某些区间选一个的限制，由于它们的 l 相同，于是建一个辅助点，把原来从 l 连出来的点改连向该辅助点，然后从 l 往该辅助点连一条流量 1 费用 0 的边即可。跑最大费用最大流即可。

由于要对于每个 k 都求一遍，故可以将流量为 k 的边都定为 m，然后每次尝试找一条增广路流 1 的流量即可。

显然该网络的点数、边数、流量、费用均为 $O(n)$ 级别，故使用传统的费用流做法，时间复杂度 $O(n^3)$。交一下发现过了，~~甚至跑得比正解还快~~，原因是这个图很难把 SPFA 卡满，反正出题人不会卡，如果有人会卡不吝赐教。

考虑优化费用流的常见做法：原始对偶。复杂度变为 $O(n^2 \log n)$，瓶颈在于最短路。进一步的，由于费用也是 $O(n)$ 级别的，因此最短路的长度只有 $O(n)$，于是可以在 dijkstra 的时候不用堆而是开 $O(n)$ 个桶，使复杂度去掉这个 $\log$，~~但其实由于常熟原因跑的差不多快~~。最终总复杂度 $O(n^2)$，可以轻松通过。

1002 Multiple and Factor

设立阈值 B，考虑用一个长度为 n 的数组 $b_1, \ldots, b_n$ 和一个长度为 B 的数组 $c_1, \ldots, c_B$ 来描述 a 序列，其中 $a_i = b_i + \sum_{j \le B \wedge j \mid i} c_j$

令 $d(x)$ 表示 x 的因数个数，$D(x) = \max_{i \le x} d(i)$。考虑每次操作进行的修改：

- 对于操作 1，若 $x \le B$，则直接给 c_x 加上 k；否则枚举所有 x 的倍数，给对应位置的 b 加上 k。复杂度 $O(\frac{n}{B})$。
- 对于操作 2，直接枚举所有 x 的因数，给对应位置的 b 加上 k。复杂度 $O(d(x))$。
- 对于操作 4，分别考虑 b 序列和 c 序列对答案的贡献：
  - 对于 b 序列，直接枚举所有 x 的因数位置，加上对应的 b 序列的值。
  - 对于 c 序列，枚举每一项 c_y，则 c_y 对答案的贡献为 $[y \mid x] \cdot c_y \cdot d(x/y)$。

  复杂度 $O(d(x) + B)$。
- 对于操作 3 且 $x > B$ 的询问，同样考虑 b 和 c 的贡献：
  - 对于 b 序列，枚举所有 x 的倍数下标位置，加上对应 b 序列的值。
  - 对于 c 序列，枚举每一项 c_y，则 c_y 对答案的贡献为 $\lfloor n / \operatorname{lcm}(x, y) \rfloor \cdot c_y$。

  复杂度 $O(\frac{n}{B} + B \cdot L(B, x))$，其中 $L(x, y)$ 表示求一个 $\le x$ 的数和一个 $\le y$ 的数的 LCM 的复杂度。

- 对于操作 3 且 $x \le B$ 的询问，考虑直接维护这些答案，在修改时对每一种询问处理贡献，形式与之前类似。

复杂度为 $O(n + m \cdot (B \cdot L(B, n) + \frac{n}{B} + d(n)))$，取 $B = \sqrt{n}$，复杂度为 $O(n + m\sqrt{n} \cdot L(\sqrt{n}, n))$。

求一个 $O(\sqrt{n})$ 的数和一个 $O(n)$ 的数的 LCM，考虑做一次辗转相除，变成求两个 $O(\sqrt{n})$ 的数的 GCD 的复杂度，可以直接预处理 $O(\sqrt{n})$ 的所有数对的 GCD。至此可以做到 $O(n + m\sqrt{n})$。

关于本题的时限：开在了 std 时间的两倍以上，被卡常可能是你写了 $\log$ 求 LCM。如果你的复杂度正确并且仍然被卡常，出题人在此致歉。

1003 Matrix Equation

不知道为什么，卡常高手拿 $O(n^3 \log m)$ 卡过去了。

> The problem itself is trivial, but solving it requires not skipping lectures on linear algebra.

系数对 998244353 取模，相当于说矩阵的系数是大小为 $p = 998244353$ 的有限域内的元素。

回忆基础的线性代数知识：对于一个 m 次多项式 $f = \sum\limits_{i=0}^{m} a_i x^i$ 和一个 $n \times n$ 矩阵 A，定义 $f(A) = \sum\limits_{i=0}^{n} a_i A^i$。

这一运算有以下性质：对任意多项式 f, g 和 $n \times n$ 矩阵 A，$(fg)(A) = f(A)g(A)$。

我们希望找到最小的 $k \in [1, m]$ 使得 $(x^k - 1)(A) = 0$，即 $x^k - 1$ 是 A 的所谓 “零化多项式”。这里称一个多项式 f 是 A 的零化多项式当且仅当 $f(A) = 0$。

设 f, g 均是 A 的零化多项式，容易发现 $f + g$ 也是，且 xf 也是，于是辗转相除可以得到 $\gcd(f, g)$ 也是。

因此一定存在一个次数最低的非零零化多项式，使得剩余零化多项式均是它的倍式，这个零化多项式就是矩阵 A 的极小多项式。

假设我们已经算出 A 的极小多项式是 F，目标是找到最小的 k 使得 $F \mid x^k - 1$。如果 F 的常数项是 0 那么容易发现无解，否则相当于要解方程 $x^k = 1 \pmod F$ 是（Ex）BSGS 的形式，而由于 F 的常数项非 0 所以 x 与 F 互素，于是用 BSGS 就可以。

这里的 BSGS 相比一般用的对某个有限域内元素求离散对数的方法是基本一致的，无非是设 $B = \lceil \sqrt{m} \rceil$ 然后考虑设 $k = iB - j$ 其中 $i, j \in [0, B]$，那么 $x^{iB-j} = 1 \pmod F$ 即 $x^{iB} = x^j \pmod F$，注意是 x 与 F 互素保证了这两个条件等价。只需计算出所有 $\forall 0 \le i \le B, x^{iB} \bmod F$ 和 $\forall 0 \le j \le B, x^j \bmod F$ 并找到 $iB - j$ 最小的一对相等的 $x^{iB} \bmod F$ 与 $x^j \bmod F$ 即可。时间复杂度相当于做 $\Theta(\sqrt{m})$ 次次数 $O(n)$ 的多项式乘法与多项式取模，不使用 FFT 是 $\Theta(n^2\sqrt{m})$ 的，使用的话是 $\Theta(n \log n \sqrt{m})$ 的。

接下来考虑如何求一个矩阵的极小多项式，设极小多项式的次数为 d，这相当于找一系列系数 $c_0 \sim c_d$ 使得 $\sum\limits_{i=0}^{d} c_i A^i$ 是零矩阵。一个暴力的想法是依次计算出 $A^0, A^1 \ldots A^n$ 然后将其视作长度为 n^2 的向量并依次插入线性基，寻找最小的 d 使得 $A^0, A^1 \ldots A^d$ 存在非平凡线性关系（即一组系数不全为 0 的解），遗憾的是该算法是 $\Theta(n^4)$ 的。

考虑这样一个想法：随机长度为 n 的向量 v，并转而寻找 $A^0v, A^1v \dots A^nv$ 的线性关系，现在相当于找一系列系数 $c_0 \sim c_d$ 使得 $(\sum_{i=0}^{d} c_iA^i)v$ 是零向量，显然相比原来放松了一些限制。但假设 $\sum_{i=0}^{d} c_iA^i$ 并非零矩阵，看上去其将 v 映到 0 的概率也不高，我们先假定该算法有足够高的正确率，证明见后。

此时，计算 $A^0v, A^1v \dots A^nv$ 相当于对一个向量左乘 n 次矩阵，是 $\Theta(n^3)$ 的。而将 $n+1$ 个长度为 n 的向量插入线性基寻找线性关系，也是 $\Theta(n^3)$ 的。结合前面 BSGS 的部分，时间复杂度为 $\Theta(n^3 + n^2\sqrt{m})$ 或 $\Theta(n^3 + n\log n\sqrt{m})$。

事实上，可以证明寻找极小多项式的部分至少有 $1 - \frac{n}{p}$ 的正确率。证明：设 F 在 $\mathbb{F}_p[x]$ 内的不可约分解是 $\prod_{i=1}^{l} p_i^{e_i}$ 其中 $p_i \in \mathbb{F}_p[x], e_i \in \mathbb{Z}_{\geq 1}$，$p_i$ 互不相同且不可约。假设这样的极小多项式不是 F，说明存在次数更低的非零多项式 G 使得 $G(A)v = 0$，不妨设 G 在这些多项式中次数最低，那么必然 $G \mid F$，否则考虑 $(F \bmod G)Av$ 不难发现其必然是 0，与 G 次数更低矛盾。既然这样，一定存在一个 $1 \leq i \leq l$ 使得 $(\frac{F}{p_i})(A)v = 0$。考虑 $(\frac{F}{p_i})(A)$ 这个矩阵将多少 v 映到 0，设 $\operatorname{rk}(\frac{F}{p_i})A = s$，当然是 p^{n-s}。而 s 不能为 n 否则 $\frac{F}{p_i}$ 是 A 的零化多项式与 F 是极小多项式矛盾，于是 $(\frac{F}{p_i})(A)$ 将至多 p^{n-1} 个 v 映到 0。而一共有 l 个 $(\frac{F}{p_i})(A)$，至多将 $lp^{n-1} \leq np^{n-1}$ 个 v 映到 0。随机取一个 v 落在其中的概率不超过 $\frac{n}{p}$，于是该算法的正确率至少是 $1 - \frac{n}{p}$。

[延申阅读：扩展上述算法，在一个足够大的有限域内以 $\Theta(n^3)$ 时间高概率计算出一个矩阵的有理标准形]。

未能解决的问题：能否使用分圆多项式相关的知识将复杂度做到与 m 无关?

1004 Stocks Trading

考虑一种暴力 DP，令 $f_{i,j}$ 表示已经经过了前 i 天，此时钱的数量是 j 且从未破产的概率。转移时就是将 f_{i-1} 和 p_i 做卷积即可转移至 f_i。但是这种 DP 方式 j 这一维的大小是 $O(nm)$ 的，无法通过。

考虑分治，定义 $solve(l, r, dp)$ 表示已经算出了 $f_{l-1} = dp$，要解决 $[l, r]$ 这一段的问题。令 $mid = \lfloor \frac{l+r}{2} \rfloor$。考虑条用左半边，如果当前 $j \geq (mid - l + 1) \cdot m$，那么即使 $[l, mid]$ 这一段全部 $-m$，也不会破产，因此可以令 dp' 为 dp 保留前 $(mid - l + 1) \cdot m$ 项的结果，可以先调用 $solve(l, mid, dp')$ 来解决左半边的问题，注意分治过程并不需要求出 f_{mid}。先通过分治卷积求出 $p_l, \dots, p_{mid}$ 这些多项式的乘积为 $prodl$，然后将 dp 和 $prodl$ 做卷积为 dq。同样注意到 dq 中 $j \geq (r - mid) \cdot m$ 的项在右半边一定也不会破产，因此只需要保留 dq 中前 $(r - mid) \cdot m$ 项为 dq'，然后调用 $solve(mid + 1, r, dq')$ 即可。

注意到每一次分治 $solve(l, r, dp)$ 时，传入的 dp 永远只有 $O((r - l + 1) \cdot m)$ 项。同时，在分治过程中我们也只需要做若干次长度为 $len = r - l + 1$ 的卷积。单次 $solve$ 复杂度为 $O(len \log len)$，容易分析出总复杂度为 $O(n \log^2 n)$。

1005 Range Convex Checker

注意到如果 S 是好的，那么 S 的任意非空子集都是好的。所以对于所有 l 一定的区间，只有一段 r 的前缀使 $[l, r]$ 满足要求。对于同一个 r 同理。

我们可以双指针维护极大的 l, r，每一次指针的移动需要检查插入某点后当前点集是否是凸包，如果不是就不插入。

假设我们找到了一个一定在凸包内的点 P，那么可以将 S 中点以 P 为中心进行极角排序。对于任意极角序相邻的三点 P_i, P_{i+1}, P_{i+2}（逆时针方向递增），若 $\overrightarrow{P_iP_{i+1}} \times \overrightarrow{P_{i+1}P_{i+2}} \geq 0$，则 S 就是好的集合。显然这是充要的。插入一个新的点时，找到他在极角序的前驱后继即可 $O(1)$ 检查是否合法。

问题是不一定会有点一直在凸包内部。注意到若三点不共线，则这三点构成三角形的重心必然在三角形内部。则只需从当前区间内选取三点并以重心为中心进行操作，等到三点中任意一点被移出区间后再重新选取并重构。因为对于 $l > 1, r < n$ 的极大区间，对应 S 的凸包面积一定 > 0，所以这样的重心一定存在，只需在开头（或）结尾特判即可。

为了保证重构的复杂度，重构 $[l, r]$ 时选择最靠近 r 的不共线的连续三点求重心即可。若任意相邻三点不共线，则每次三点移动的距离和重构复杂度成正比，所以只需要进行 $O(n)$ 个点的重构。相邻三点共线也不影响复杂度。

实现时将所有坐标乘以三可以全整数运算。总时间复杂度 $O(n \log n)$。

关于输入坐标 10^7 级别时，极角计算用 `atan2` 会不会有精度问题：仔细分析的话发现精度误差和要求的非常接近，但是出题人拼尽全力没有卡掉。不放心的话用 `atan2l` 即可。如果有选手有卡掉 `atan2` 的方法麻烦不吝赐教。

1006 Oden VS Genshin Impact

我们发现对于一个 x 级的角色升级到 $x+1$ 级，要么要 $3^x - 3^{x-1}$ 张低级复制卡，要么要 2 张高级复制卡。

首先发现，一定可以找到一个最优解满足存在一个 x，$> x$ 的升级不用低级复制卡，$< x$ 的升级不用高级复制卡。

相当于除了 x 级以外，这个题目会被割裂为两部分。

所以我们先分别考虑 只有低级 和 只有高级 怎么做。

只有高级的话，我们发现是一个经典的背包问题，可以在 $O(nmB)$ 的复杂度内使用动态规划解决。

只有低级的话，我们逐位考虑，使用数位 dp 算法，可以参考 CF1290F 的做法，$f_{l,S,i}$ 表示现在 dp 到第 l 位，已经升级到这级的有 S，还要向高位借 i 个低级复制卡时最多总攻击力，转移就枚举每个是否升级，升级了就消耗复制卡，不升级就贡献到答案并且移出 S，这样 dp 的复杂度是 $O(2^n n^2 m)$。

ok 然后我们考虑怎么将两个做法拼接起来。

我们枚举这个层数 x，假设我们现在要让集合 S 升级到 $> x$ 的等级，我们就先计算最少要用几个高级复制卡可以让他们全部升级到 $x+1$ 的等级，然后就相当于问一个集合在 $x+1$ 级之后能用 T 张高级复制卡最多能有多少攻击力。

低位问题仍然使用数位 dp，高位问题使用背包 dp 来处理高位的问题，$g_{S,i,j}$ 表示 S 这个集合从 i 级开始升级，有 j 张高级复制卡能升到多少总战力，g 的转移可以使用背包 dp，每次枚举最低位用了多少张高级复制卡，相当于从 $g_{S-\{u\},i,k} + a_{u,i+j-k}$ 转移，这样对于一个 i 背包复杂度是 $O(2^n mB)$。

我们发现有效的 x 只有 $O(\log_3 n)$ 个，因为往低了蓝色复制卡用不完，往高了蓝色复制卡完全不够用。

所以复杂度为 $O(2^n m(B \log n + n^2))$。

std 实现和讲解有所出入。

1007 Message Spreading

两种操作相当于将一个 0 修改为 1 和将所有 1 连续段向两端扩展一次。

首先有一个简单的观察，一定会先进行赋值为 1 的操作，然后在进行扩展连续段操作。

考虑单次询问怎么做，首先特判全 0 和全 1 的情况。对于一般情况，求出所有 0 颜色的连续段长度，令所有 0 颜色连续段的长度集合为 S。枚举扩展连续段操作的次数为 t，那么对于一个长度为 len 的 0 连续段，至少需要放置 $\lceil \frac{len-2t}{2t+1} \rceil = \lfloor \frac{len}{2t+1} \rfloor$ 个 1，容易得到答案为：

$$\min_{t\geq 0}\left(t+\sum_{x\in S}\left\lfloor\frac{x}{2t+1}\right\rfloor\right)$$

注意到取 $t=\sqrt{n}$ 时，该式子的值一定 $\leq 2\sqrt{n}$，因此该式的上界为 $2\sqrt{n}$，也就是说我们只需要维护 $t\leq 2\sqrt{n}$ 的值即可。考虑对于每一个 $t=0,\ldots,2\sqrt{n}$，分别维护 $\sum_{x\in S}\left\lfloor\frac{x}{2t+1}\right\rfloor$ 的值。根据颜色段均摊的结论，我们可以直接使用 set 维护所有全 0 连续段，同时维护长度集合 S，这样的更新只会有 $O(n+q)$ 次，每次更新直接枚举所有 t 来更新对应的右式和即可。复杂度 $O((n+q)\sqrt{n})$。

细节是这是一个环，可以直接按照链来维护，每次查询时删除 1 和 n 所在的连续段，然后加入新的一段，查询结束后复原即可。

1008 Repeater

仔细阅读题目，发现规则的本质是对于每一个 trust 操作，给自己 -1，给对方 $+3$。从 m 到 1 枚举 j，考虑每一个 j 这一列 `T` 字符的贡献。枚举到一个 j 时：

- 定义 f_i：
  - 若 $s_{i,j}$ 不是 trust，则 $f_i=0$。
  - 否则 f_i 为最大的 x，满足 $\forall 1\leq k<x$，$s_{i,j+2k}$ 都是 repeat。
- 定义 g_i 为最大的 x，满足 $1\leq k\leq x$，$s_{i,j+2k-1}$ 都是 repeat。

对于两个人 x,y，若 $s_{x,j}$ 为 trust，则这个 trust 对两人的贡献分别是 $3\cdot\min(f_x,g_y)-\min(f_x,g_y+1)$ 和 $3\cdot\min(f_x,g_y+1)-\min(f_x,g_y)$。

若我们对所有的 f_x 和 g_x 放在一起排序，那么可以分 f_x 和 g_y 的大小进行讨论，预处理前缀和快速计算一个人的贡献。复杂度 $O(nm\log n)$，瓶颈在于排序。

如何优化排序？f 和 g 的计算过程其实是每一列时，先将 f 和 g 交换，然后将所有的 f 加一，并将一些 f 赋值为 0。我们直接维护 f 和 g 分别排序的结果，每次 $+1$ 操作并不影响排序结果，将置为 0 的元素拉出来放到序列开头即可。最后在每一列将 f 和 g 进行归并排序。这样做复杂度就是 $O(nm)$ 的。

1009 PalindromemordnilaP

先考虑某个串是广义回文的充要条件，不难发现就是该串存在至少一个 border，证明不难。

先特判 $m=1$，此时每个长度 ≥ 2 的串都是广义回文的，$O(1)$ 计算即可。

当 $m\geq 2$ 的时候，若 s 存在两个相同的子串，由于数据随机，可以大致估算一下：任取两个不一样的起点求 lcp，后面的字符近似估计为独立的，那么 lcp 长度为 x 的概率只有 $\frac{1}{m^x}$。也就是说，长度是 $O(\log_m n)$ 级别的。这提示我们直接把所有出现的相同的子串找出来当作一对 border。这样的问题是一个串有多个 border 时会重复数，于是我们可以钦定每个串在最短的 border 处计数，这等价于我们枚举的 border 本身不含有另一个更小的 border，可以在枚举的时候直接判断。这部分时间复杂度是 $O(n\text{polylog}n)$ 的，完全可以承受。

于是问题转变成数据结构问题。做法较多，以下介绍一种。先考虑两个简单的暴力做法：

1. 对于每个询问，直接枚举每种不同的串，$O(1)$ 搞出区间内这种串出现的次数（用前缀和预处理）然后算一下加入答案。这样空间上不一定能承受，可以先枚举串再枚举询问，这样同时只要维护一个前缀和数组。
2. 对于每种串，直接把以它为 border 的所有串的左右端点 l', r' 拉出来，这样每个询问相当于询问 $l \le l', r' \le r$ 的个数。不难发现这就是二维数点，离线后可以使用树状数组完成。进一步的，考虑到 l', r' 会很多但询问的次数相对较少，可以使用 $O(1)$ 修改 $O(\sqrt{n})$ 查询的分块平衡一下复杂度，会更优秀。

考虑到，对于长度较小的串，由于字符集不大，串的种类数不多；对于长度较大的串，出现次数不会很多。于是考虑设置阈值 B，对于长度 $\le B$ 的串使用上面的算法 1；$> B$ 的串使用上面的算法 2。B 的取值大约取 $\frac{\log_m n}{2}$ 比较优秀，不难估计此时算法 1,2 的复杂度都在 $O(n\sqrt{n})$ 级别，可以通过。当然你也可以直接根据拉出来的串动态设一个阈值，std 就是这么写的。

1010 Cut Check Bit

不妨先将 x 加上一（注意特判 x 的二进制是全 1 的情况），这样 $\le x$ 变成了 $< x$。

从高到低枚举哪一位结果是 0 而 x 是 1，不妨设为第 d 位（最低位为第 0 位）。此时只要对于每个 i 处理出，以 i 为左端点，让 $\ge d$ 位满足条件，右端点至少要到多远，记作 r_i。（当然也有很多等价的做法，比如往左拓展等。）于是答案容易贪心或 dp 计算。

考虑快速的求 r_i。初始时显然 $r_i = i$，以后只需要在高位往低位枚举的时候，若结果这一位强制为 0，就令 r_i 和 i 之后第一个这一位为 0 的位置取 $\max$ 即可。

时间复杂度 $O(nm)$。

1011 Worst Problem Of All Time

无解当且仅当存在自环。否则，考虑如下的贪心算法：

维护一个有向无环图 G，初始为空，并按顺序遍历每条边 u, v：

- 若 G 中 v 不能到 u，则向 G 加入边 u, v，这条边答案是 1；
- 否则，这条边答案是 2。

不难使用数学归纳法证明该贪心算法的正确性。

考虑优化复杂度。不难发现这个题其实类似于 https://qoj.ac/problem/4000，但是由于 G 任意时刻都是 DAG，故有较大的简化。以下给出做法。

类似于操作分块，设定阈值 B，对于每 B 条边一起做。记这 B 条边的所有端点为关键点。先不管这 B 条边，，由于 G 是 DAG 可以直接用拓扑排序 + bitset 求出关键点间的可达性。这样我们相当于把整个图的点数缩成了关键点个数，即 $O(B)$ 个。由于 G 是 DAG 的特殊性，加边时可以直接类似于传递闭包一样更新这些 bitset。当然直接在这上面 bitset 优化 bfs 并且额外维护新加的边也是可以打。总时间复杂度为 $O\left(\frac{m}{B}\left(\frac{nB}{w} + B\frac{B^2}{w}\right)\right) = O\left(\frac{nm}{w} + m\frac{B^2}{w}\right)$。

由此发现，只要取 B 为适当值，总复杂度即为 $O\left(\frac{nm}{w}\right)$。注意 B 不能太小，一方面是如果太小则没法除以 w；另一方面，每次拓扑排序等地方还有额外的 $O(n)$ 的开销，若 $\frac{m}{B}$ 太大则这些开销会成为瓶颈，不一定能通过。因此 B 需要适当取大，本题中取 $2^9 \sim 2^{10}$ 比较合适。

1012 Counting Colorful Sequence

首先容斥，统计对于每个 i，都不满足 a_i 为 $1 \sim i$ 的颜色数的序列个数。考虑从前往后 DP，DP 过程中记录当前 $[1, i)$ 的颜色数，则 a_i 的值并不会大幅影响 $[1, i]$ 的颜色数。具体来说：

考虑假设当前 $[1, i)$ 内的颜色数为 c，分类讨论 c 和 $c+1$ 是否在其中出现来确定 i 的值：

- c 出现，$c+1$ 出现，可以填任意不为 c 的颜色。
- c 出现，$c+1$ 未出现，可以填任意不为 c 和 $c+1$ 的数。
- c 未出现，$c+1$ 出现，可以填任意数。
- c 未出现，$c+1$ 未出现，可以填不为 $c+1$ 的任意数。

令 $f_{i,j,k,0/1,0/1}$ 表示考虑了 $[1, i]$ 这些位置的颜色，颜色数为 j，前缀中给的 $> j+1$ 的值一共有 k 种，j 这个值是否出现，$j+1$ 这个值是否出现。

转移时，若填了一个 $\le j+1$ 的数，那么我们直接确定到底填了什么数；若填了 $> j+1$ 的数，则我们只记录不同数的种类数，在确定某个值是否出现或最终统计答案时，再具体确定是哪个值。对于所有的 $1 \le n \le N$ 都可以共用这个 DP 数组，只是最后统计答案时略有差别。复杂度 $O(N^3)$。

1013 Spring River Flower Moon Night

容易发现到 (x, y) 时的代价一定为 $x \times y$，证明即为简单归纳。答案为 $n \times m$。时间复杂度 $O(q)$。