A. 红藕香残玉簟秋

我们先考虑 $k=1$ 的情形。如果往空格子 (x,y) 里填入 1 将会同时满足第 x 行和第 y 列的限制。

我们将每行看成一个左侧点，每列看成一个右侧点，将每个空格子看成是连接所在行与所在列的一条边。问题转化为二部图最小边覆盖。

我们知道在二部图中最小边覆盖就等于顶点的总数再减去最大匹配。直接跑一遍匈牙利即可，时间复杂度 $\mathcal{O}(n^3)$，可以获得 50 分。

对于一般情形，我们注意到不同的限定数之间是彼此无关的。因为一个空格子 (x,y) 如果 $a_x\neq b_y$ 只能满足 a_x 和 b_y 里较小的那一个，并不会节省一条边覆盖。因此对每个限定数分别跑一遍最小边覆盖，将答案直接加起来即可。时间复杂度 $\mathcal{O}(nc^2)$，可以获得 100 分。

判断是否存在合法方案也很简单：只需要往 (x,y) 里填入 $\min\{a_x,b_y\}$，然后 $\mathcal{O}(n^2)$ check 一遍即可。

B. 薄雾浓云愁永昼

背包是好做加入但是不好做删除的。因此我们首先将所有操作离线下来，翻转整个时间轴，这样每次删除一块金子就变成了每次加入一块金子。直接开一个背包暴力维护即可，时间复杂度 $\mathcal{O}(nk)$，可以通过子任务 1 获得 20 分。

接下来我们着眼于 $m=1$ 的情形。我们要在加入物品的顺序上下功夫。

注意到通过 $x_i\geq x$ 的金子至多只能获取 $\lfloor\dfrac{k}{x}\rfloor$ 的价值。记 F_w 表示想要获取恰好 w 的价值最少需要多少时间，我们按照 x_i 从大到小的顺序依次加入所有金子，则加入到 x 的时候下标只需要开到 $\lfloor\dfrac{k}{x}\rfloor$。状态总数是调和级数，转移是 $\mathcal{O}(1)$ 的，因此时间复杂度 $\mathcal{O}(k\log k)$，可以通过子任务 2 获得 40 分。

最后我们解决一般情形。考虑根号分治。我们设置一个阈值 B，把金子分成 $x_i\leq B$ 和 $x_i>B$ 两类，分别开一个背包。

- 第一类 $x_i\leq B$ 的金子至多只有 B 块。每次修改暴力维护即可，这部分复杂度 $\mathcal{O}(kB)$；
- 第二类 $x_i>B$ 的金子可能有 n 块。但是初始时不存在的金子至多只有 m 块。先 $\mathcal{O}(k\log k)$ 预处理出初始背包，每次修改暴力维护，由于下标只需要开到 $\lfloor\dfrac{k}{B}\rfloor$，这部分复杂度 $\mathcal{O}(k\log k+\dfrac{mk}{B})$。
- 回答询问需要二背包合并。我们规定 $x_i\leq B$ 的背包记拥有一定时间最多获取多少价值，$x_i>B$ 的背包记想要获取一定的价值最少需要多少时间，回答询问时枚举想要在 $x_i>B$ 的金子那里获得多少价值，这部分复杂度 $\mathcal{O}(\dfrac{mk}{B})$。

取 $B=\mathcal{O}(\sqrt{m})$ 根号平衡得最优时间复杂度 $\mathcal{O}(k\log k+k\sqrt{m})$，可以通过所有子任务获得 100 分。

C. 雪里已知春信至

首先着眼于每次修改均是 $[1,n]$ 的情形。

由卢卡斯定理我们可以知道：在经过 x 次全局修改后，新的 $a'_i=\bigoplus\limits_{y\subseteq x}a_{i-y}$，其中 $y\subseteq x$ 表示在 y 的每个为 1 的二进制位上 x 也为 1。

如果 $\operatorname{popcount}(x)$ 特别大暴力枚举 y 不能承受。因此我们取最小的 $2^k\geq\sqrt{n}$，每经过 2^k 次修改操作我们就重构整个序列，这样回答询问只需要枚举 2^k-1 的子集，时间复杂度 $\mathcal{O}(q\sqrt{n})$，可以通过子任务 $4\sim6$ 获得 30 分。

对于一般情形，我们采用定期重构配合序列分块的办法。

不妨设 n 是完全平方数。我们把所有操作离线下来，每进来 $\sqrt{n}$ 个修改操作就重构整个序列。

将序列以 $\sqrt{n}$ 为块长分成 $\sqrt{n}$ 块。枚举每一个块，我们需要回答这个块内的询问，还要求出 $\sqrt{n}$ 个修改操作之后这个块内每个位置的值。由于修改操作的个数 $\leq\sqrt{n}$，我们只需要关心这个块内**以及前一个块内**每个值的变化情况。

- 如果进来一个修改操作完全覆盖了这两个块，我们就把全局修改的 tag 增加 1，暂时不进行处理；
- 如果进来一个修改操作只与这两个块的一部分有交，我们就要先利用卢卡斯定理清空 tag，之后再暴力处理。注意这里清空 tag 并不是暴力重构每个位置，而是类似高维前缀和的形式枚举当前累积 tag 的每一个二进制位分别处理，因此这里会带上一个 $\log$；
- 如果进来一个查询操作，只需要利用卢卡斯定理暴力得出当前这个位置上的值即可。

显然一个查询操作只会在一个块内被处理，因此复杂度是 $\mathcal{O}(q\sqrt{n})$ 的。而一个修改操作 $[l,r]$ 至多只会与 $\mathcal{O}(1)$ 个边界处的块部分有交，与内部 $\mathcal{O}(\sqrt{n})$ 个块都是完全覆盖的关系，因此复杂度是 $\mathcal{O}(q\sqrt{n}\log n)$ 的。时间复杂度 $\mathcal{O}\left(n+q\sqrt{n}\log n\right)$，可以获得 100 分。

D. 暗淡轻黄体性柔

解法一

我们考虑如何处理一对询问 (A,B)。我们把过程看作是先把 B 点删掉，如果一个点此时没有出度，那我们就继续把这个点也给删掉。一直这样进行下去。如果最后 A 被删掉了，那么 (A,B) 就能成为答案。

对于当前选择的根，如果删掉点 u 后会导致 v 点也被删除，我们称其为 u 支配点 v。我们可以构建出支配树：考虑一个点的所有出度，他们在支配树上的 LCA 就是它的父亲。

需要注意的是，根可能支配自己。根的所有出度如果都会被自己删掉那么他们在支配树上的 LCA 就会支配根。所以实际上，我们最后得到的关系是一个支配基环树。对于环上的每个点，我们如果删掉他就会让这个基环树被删掉；对于不在环上的点，删掉它会使得子树里的点被删掉。

可以发现，我们可以把一个图变成基环树森林再统计答案。考虑按 k 从 1 枚举到 n 动态的维护基环树森林的形态。最开始的图是 n 个自环，每次我们枚举到点 u 我们就把这个自环去掉，他的出边都加到图中。

可以发现 u 之前一定是某颗树的根。

- 如果当前这个点 u 的出度都在自己这颗树里，把他们的 LCA 往 u 连一条边就好了。
- 如果所有出度都在另一颗树中，同样求出 LCA，我们现在就是是把 u 的子树接到 LCA 下面。
- 如果出度分散在不同的联通块，此时 u 并不任何点支配。但是随着合并的进行，可能在某一刻，u 的出边全都在某一个联通块里了。所以我们在加入 u 的时候往每个联通块里扔一个标记，每次合并两个联通块的时候把两个联通块都有的标记的计数器减一，减到 0 就需要把 u 重新拿出来，把出度的 LCA 和它连边。我们可以考虑用线段树合并维护这个过程就可以做到单 log。每次在 merge 两个叶子的时候处理一下计数器。开一个队列记录当前需要考虑的 u，每次拿队首出来做。计数器变成 0 就 push 进去。

考虑怎么算答案，即对于每个时刻的支配森林求出有多少个点对 (u,v) 满足 u 能到达 v。

答案就是每个点的 siz 和也就是 dep 和，我们要维护的操作是把一个树根接到某个点的儿子处。考虑在 LCT 上维护 Link，求某个点的 dep，求 LCA，求联通块 siz 的操作就可以做了。

如果过程中出现了一个新的基环树，需要特殊处理一下。由于每个联通块只会变成一次基环树，我们可以直接暴力的通过某些手段 $\mathcal{O}(\mathrm{siz})$ 的处理这颗基环树的答案。

但是要注意的是，一颗基环树虽然不会再往外连出出度，但是里面的点之后有可能作为其他的点的出度。为了方便后续的答案计算，需要把基环树上的环点的 dep 人为的修改一下。具体的，把根的 dep 改为环长，其他环点的 dep 改成 0 就好啦。

以及，每颗树的树根有可能编号大于当前枚举的 k。可以最开始每个点的 dep 设成 0，加入这个点的时候加上对答案的贡献，再把 dep 改成 1，这样算出来的 dep 才是对的。

总体的时间复杂度为 $\mathcal{O}(m\log n)$。可以发现这颗 LCT 由于 Link 操作都是把一个树根接到另一个儿子上所以只需要 access 而不需要维护懒标记进行 makeroot，常数是很美丽的。

解法二

我们可以先直接求出 $k=n$ 时的基环森林。再考虑在最后的状态上统计答案。

首先应该怎么求出 $k=n$ 时的状态呢？如果我们现在有一个根，我们可以直接在图上模拟一遍求 LCA 加边的过程，用倍增维护就好。但是我们不能暴力的枚举一个根然后做一遍这样的事情。因为一个点为根的支配树可能是另一个点为根的支配树的子树。所以问题就变为了找到那些最厉害的根。

我们考虑不断的缩图。考虑一个点 u，如果他只有一条出边连向 v，那么我们就可以把 u 扔了，因为 v 的支配树一定包含他的支配树。此时我们把 u,v 缩成一个点，并且把之前连向 u 的边全部接到 v 上。重复这个过程，直到图中每个点的出度都 ≤ 2。此时图中的每个点都可以成为一个根/一个基环树环里的某个点。

维护缩图的过程可以使用线段树合并，可以发现和第一个做法中维护的信息大同小异，每个联通块的根处维护有那些点连向他，每次合并的时候相当于把两者都有的点的出度减一，每次找到出度变为 1 的那些点执行合并操作即可。

之后我们考虑怎么求出每个 k 的答案。首先可以发现每个时刻的图都是最后的图的子图，并且每条边都是在 k 逐渐增大到某个值的时候出现。这个值可以在求出基环树的过程中维护出来。具体来说，考虑构建基环树的过程中考虑到了当前点 u 以及他的出度的 LCA，u 连向 LCA 的这条边就是 u 到 LCA 上的每一条链上的所有边的权值的 max。于是我们在倍增的时候多维护一个链上权值的 max 即可。

之后考虑统计答案。我们拎一个环上的点当根搞一颗外向树。这个图会存在至多一条返祖边。我们把答案分为直上直下和必须经过返祖边的两部分进行统计。

- 对于直上直下的部分，可以并查集维护 siz 和最浅的那个点，很好做。
- 对于经过返祖边的部分，我们可以巧妙地把最开始钦定的根设置成使得那个返祖边是环上边权最大的边的那个点。这样我们就能保证在加入返祖边的时候，整个环就完整的形成了。
    - 加入返祖边之前只有直上直下的贡献。
    - 加入返祖边时，由于只会加入一次，我们可以暴力的 DFS 一遍算贡献。
    - 加入返祖边后，可以发现如果一对 (u, v) 满足从 u 走到 v 必然要经过返祖边那么一定是形如 u 处于 v 在环上的投影以下一直到返祖边端点的那一条链上。于是也可以很方便通过并查集的 siz 信息计算贡献。

最后整体的时间复杂度依然是 $\mathcal{O}(m \log n)$，来源于线段树合并以及倍增求基环树。