函数

nq较小的情况，可以$O(nq)$遍历每个二次函数判断是否删除

对于m较小的情况，可以对于每个不同的m用一个堆按照k递增的顺序维护每个二次函数

每次弹掉的部分一定是最小的几个，每次遍历m个位置，一共会弹掉最多$n+q$个二次函数

因此总复杂度为均摊的$O((n+q)\log n+qm)$

对于m较大的情况，依然考虑用堆维护这些二次函数

根据$f(x)=(x-m)^2+k\le t$，发现最多只有x两边$\sqrt{t}$个位置的值可能存在需要弹掉的二次函数

因此用相同的方法维护因此复杂度为$O((n+q)\log n+q\sqrt{t})$

倍数

一个非常浅显的思路是枚举每个数a_i，找到两边最长的合法位置L_i, R_i

容易想到可以二分L_i, R_i，但是并不好直接检查是否合法，考虑将倍数转化一下

考虑$\forall [L,R], x|a_i \Leftrightarrow x|\gcd(a_L, \cdots, a_R)$

因此倍数问题可以转化为gcd问题

用倍增维护二分，预处理复杂度$O(n\log^2 n)$，依次查询复杂度为$O(n\log n)$

用线段树维护二分，预处理和查询复杂度均为$O(n\log n)$

然而最优的做法是用栈维护L_i, R_i

依次考虑每个数a_i，用一个栈维护前面出现的，且互相之间分离的每个$[L_j, j]$

插入时依次考虑栈内每个元素a_j，若$a_i|a_j$，则显然$\forall k \in [L_j, j], a_i|a_k$，因此可以弹掉这个元素，同时扩展$L_i$为$L_j$

同理地处理R_i，复杂度为$O(n)$

Tips:输出时注意不要输出重复

花园

下文中令 $d=D, n=N, m=M$。

不妨从部分分入手。由于子任务 1 和该题正解做法关联性较低，故不再赘述，我们从子任务 2 开始看起。

我们先偷看一眼样例解释，发现第一种纪念品是一堆相邻最近点对距离为 D 的点阵，第二种则是一个以若干条相邻之间距离为 D 的水平线和竖直线构成的网格，因此不难想到如果我们把平面划分为若干个大小为 $D\times D$ 的方形，每个方形内部都是一样的。故我们只需要考虑任意一个大小为 $2D\times 2D$ 的方形即可（不是 $D\times D$ 的原因是因为选超过边界的答案可能更优）。

因此我们有了一个 $O(d^4(n+m))$ 的暴力做法。具体而言是枚举每一个矩形和纪念品，判断其是否在这个矩形中即可。

然后我们想起来了我们在噗叽组的时候学过的最大子矩形之类的题目，我们现在试图去枚举三个边界，然后考虑每个纪念品是否已经被包含在当前矩形里。每个纪念品都会对下一个边界产生一个不等式的限制条件，对这些不等式求交即可获得一个 $O(d^3(n+m))$ 的做法。

我们试着再少扫一个边界，如果你像我一样只扫左上边界的话，你会获得一个没有前途的答辩 $O(d^2m)$ 的做法。因此我们考虑只扫左右边界。我们将纵轴看作是一个环，考虑此时的限制条件是什么。对于每一个一类纪念品，其相当于要求上下边界构成的弧包含其纵坐标。对于每一个二类纪念品，如果当前没有竖直线与其相交，则也是一个和一类纪念品一样的区间限制。这其实是一个环上的覆盖问题，可以以和环长线性相关的复杂度求出。具体而言，我们每次枚举相邻两个点，计算覆盖除去其中间一段的所有位置的答案。时间复杂度 $O(d^2(d+n+m))$。

再少扫一个边界感觉就彻底没救了。我们现在试图把固定左端点扫一轮右端点的复杂度降低为 $O(d+n+m)$，即把枚举并计算答案的复杂度均摊出去。不妨倒序枚举右端点，维护当前的每一个限制点。这其实相当于是加入若干关键点然后统计答案的问题。可以通过链表，每次删去一个点后用前驱后继的答案更新答案即可做到单次删除均摊 $O(1)$。总时间复杂度 $O(d(d+n+m))$。

那么最后的做法就呼之欲出了。注意到对于每一个位置其实只需要有一个未被覆盖的就需要从链表中被删除。而在所有这样的纪念品中对每一列保留行号最靠右的即可，那么每次均摊的东西由 $O(n+m)$ 变成了 $O(d)$，所以总时间复杂度是 $O(d^2)$。

期望

暴力做法就是直接dp然后两边累和，预处理和最后计数的复杂度为$O(nk)$

因为k很小，考虑用矩阵乘法优化

设无障碍的转移矩阵为A，有障碍的转移矩阵为B

则答案矩阵应为$Ans=\sum_{i=2}^{x-1}A^{i-1}\cdot B\cdot A^{n-i}$，我们需要知道$Ans[\frac{k}{2}][\frac{k}{2}]$，最后再乘上$n-2$的逆元

求出这个矩阵有两种做法

Solution1

考虑用一个矩阵维护两部分的答案

$$X=A^i,Y=\sum_{j=2}^{i}A^{j-1}BA^{i-j}$$

$$X'=X\cdot A,Y=Y\cdot A+X\cdot B$$

两部分分别转移按照递归关系转移即可，复杂度为$O((2k)^3\log n)$

(由于矩阵乘法的实现，可能实际复杂度还需要乘2，所以应当是$O(16k\log n)$，但是只要在矩阵乘法中合理优化，就能避免大部分空余复杂度)

Solution2

考虑倍增答案，每次倍增转移上面提到的两个矩阵，设其为X,Y

则扩展一倍之后，按照转移的式子应有$X'=XY+YX,Y'=Y\cdot Y$，这一步需要3次乘法

对于扩展之后仍然剩余的一个可以额外处理，这一步需要两次乘法

每次转移有$3-5$次乘法，复杂度上限为$O(5\cdot k^3\log n)$

选择不同的算法，根据常数优化能得到不同的分数

实际只需要优化取模次数，用unsigned long long 存储，每$16-18$次进行依次取模运算即可

分数

对于n,m较小的情况，可以考虑枚举路径起点，带回撤地遍历树就能得到所有情况的答案，复杂度为$O(n^2)$

对于$x_i=y_i$的情况，等价于求权值最大的路径，可以直接$O(n)dp$求出直径

对于$x_i=1$的情况，方案必然过1号节点，实际上直接从1号点开始累路径前缀和即可

一般的情况，考虑将每份答案的贡献描述为：当选择的路径包含x_i到达y_i的子段时，加上该权值

考虑一种比较简单的情况，$LCA(x_i,y_i)\neq x_i,y_i$，那么就是所有从x_i的子树到达y_i的子树中的路径都要加上权值

如果用dfs序描述一颗子树为一段连续的区间，那么这个问题可以转化为一个二维区间加法和查询全局最大值的问题

同理的x_i,y_i为祖先关系的情况也可以得到类似的处理

处理出所有询问后，依次扫描起始点的dfs序，用线段树维护二维加法和全局最大值，复杂度为$O(m\log n)$