跑步

按照从上到下的顺序处理每一行，维护一个 $height$ 数组表示这一行的第 i 个格子向上遇到第一个水格子之前有多少个空地格子，每次可以 $O(m)$ 更新，然后从左往右处理每个格子，求出以该格子为右下角的最大周长矩形，对于第 i 个格子，假设矩形的左边界为 j，那么矩形的高度为 $\min height[j \cdots i]$，即周长为 $2(i-j+1+\min height[j \cdots i])$。

由于和后缀 $\min$ 有关，考虑用单调栈维护。在处理第 i 个格子之前先将 $height_i$ 插入单调栈，假设单调栈中元素在 $height$ 数组中的下标依次为 $s_1, s_2, \cdots, s_k$，那么最大周长为：

$$\max_{j=1}^{k} 2(i - s_{j-1} + height_{s_j}) = 2i + 2\max_{j=1}^{k}(height_{s_j} - s_{j-1})$$

记 $val_i = height_{s_i} - s_{i-1}$，在单调栈上的每个元素存 val 的前缀 $\max$，就可以 $O(1)$ 求出最大周长了。

复杂度 $O(nm)$。

网络

关于最小直径生成树有一个结论：最小直径生成树的直径中点（有可能在一条边上）是图的绝对中心，其中图的绝对中心可以存在于一条边上或某个点上，该中心到所有点的最短路的最大值最小。

下面来证明这个结论：记 $dist_u$ 表示 u 离所有点树上距离的最大值，首先树的直径有两种理解方式，一种理解方式是 $\max_u dist_u$，另一种是 $2\min_u dist_u$，其中 u 可以在边上。在第二种理解方式中，当 u 为直径中点的时候取到 $\min$，因此最小直径生成树需要最小化 $\min_u dist_u$，$dist_u$ 最理想的情况就是取到图的绝对中心到所有点的最远距离，不可能比这更短了，构造的方法就是以绝对中心为起点，生成一个最短路径树。

求绝对中心需要先求出全源最短路，由于边权全为 1，有复杂度为 $O(\frac{n^3}{\omega})$ 的算法，后面再介绍。记 $dist_{u,v}$ 表示 u, v 之间的最短路，$far_u = \max_v dist_{u,v}$，首先 far_u 最小的哪些点是绝对中心的候选点，绝对中心还可能是一条边的中点，枚举边 (u, v)，中点离所有点的最远距离为：

$$\max_{k=1}^{n} \min(dist_{u,k}, dist_{v,k}) + 0.5$$

由于 u, v 是相邻的，所以 $\forall k \in [1, n], |dist_{u,k} - dist_{v,k}| \le 1$，推出这个最短距离大于等于：

$$\max_{k=1}^{n} \max(dist_{u,k}, dist_{v,k}) - 0.5 = \max(far_u, far_v) - 0.5$$

所以只有当 u, v 同时为绝对中心候选点时 (u, v) 的中点才可能成为绝对中心，在此前提下这条边的中点是绝对中心当且仅当不存在 k 使得 $dist_{u,k} = far_u$ 和 $dist_{v,k} = far_v$ 同时成立。对每个点 u 预处理一个 `bitset` 表示哪些 v 满足 $dist_{u,v} = far_u$，判定 (u, v) 中点就把两个 `bitset` 按位与，再用 `bitset::none()` 检验是否全 0。

关于全源最短路，先对每个点 u 预处理一个 `bitset` 表示 u 和哪些点相邻，然后枚举起点进行 BFS，在扩展点 u 的时候，需要快速找到哪些点和 u 相邻且还没有入队，这可以用 `bitset` 实现。

以 $O(\frac{n}{\omega} + popcount)$ 的复杂度遍历一个 `bitset` $bits$ 中所有的 1 方法如下：

```
for(int i = bits._Find_first(); i != bits.size(); i = bits._Find_next(i))
```

复杂度 $O(\frac{n^3}{\omega})$。

游走

由于 q 很小，先考虑单个询问怎么做。

可以发现所有的相遇事件都是相邻的两个点相遇，如果能对每两个相邻的点求出它们在观测区间内相遇了多少次，那么问题就解决了。

处理相遇后反向有一个套路：如果所有点看起来是一样的，那么相遇其实和两个点对穿而过看起来是一样的，定义每个点独立移动的过程（不考虑相遇）为对照局面，考虑相遇的过程为真实局面。

对照局面是很好求的，并且可以体现所有点的位置和方向，但不知道每个点的编号。另一个性质真实局面中所有点的相对顺序不变，即 1 号点永远是最左边的点，以此类推，所以对照局面排序后就是真实局面。

可以发现对照局面的周期为 $2L$，即经过 $2L$ 时间后，位置集合和初始位置集合相同，排序后也是一样的，所以真实局面的周期也是 $2L$，那么可以把 $[0,T]$ 的时间拆成若干个整周期和最后一个不完整的周期，问题就转化为了 $T\le 2L$ 的情况。

对照局面中两个点相遇说明真实局面中也有两个点相遇，只要找出相遇位置的大小排名就可以知道具体是哪两点相遇，于是就得到了一个做法：枚举两个点，求出它们在对照局面中所有的相遇事件并求出相遇位置的排名，就可以统计出在真实局面中相邻两个点相遇了多少次。

改进的方法就是只枚举一个点，考虑它在对照局面中所有的相遇事件，可以发现每一次相遇位置的排名只会 ± 1，取决于它当前移动的方向，所以每次相遇位置的排名构成若干个区间，同时相遇位置也是单调变化的，所以在观测区间内的相遇事件也是若干个区间，最终产生的贡献就若干次区间加，差分即可。

其实处理这个移动过程还是挺麻烦的，一种处理方法是把 $[0,L]$ 区间抽象成一个圆，向右移动的点在上半弧上顺时针走，向左移动的点在下半弧上顺时针走，相遇就是两个点移动到上下对称的位置。

直接求出这些区间需要二分，复杂度 $O(q(n+m)\log(n+m))$，无法通过。

考虑在圆上按顺时针考虑每个点，可以发现这些区间的端点都是单调变化的，并且一个周期中变化总量是 $O(n)$ 的，于是单次询问可以做到 $O(n)$。

由于需要一些排序，所以总复杂度为 $O((n+m)\log(n+m)+q(n+m))$。

移动

先做第二问，再做第一问。

如果这 n 条线段存在合法的移除方案，那么任意取一个子集也存在合法的移除方案，所以需要找到一条能立刻移除的线段。感性理解一下，如果有光从上往下照，那么存在一条线段没有被遮挡，这条线段就可以往上移走，严谨证明如下：先找到左端点最靠左再最靠上的一条线段，如果它被遮挡了，那么就找遮挡它的线段中左端点最靠左的一条，重复这个步骤，直到不被遮挡为止，由于每次找到的线段的左端点是不被遮挡的，所以下一条线段的左端点横坐标会更大，最终一定可以找到。

所以我们得到了只需要全部向上就可以移除所有线段这个结论，但每次找一个线段很难优化复杂度，考虑换一个思路，如果线段 u 遮挡了线段 v，那么 u 需要比 v 先移走，如果把线段当成结点，有遮挡关系的就连一条有向边，拓扑序就是合法的方案。考虑从左往右扫描线，把和扫描线接触的线段从上往下连成一条链，由于所有线段不相交，所以在扫描线移动的过程中，和扫描线接触的线段的上下顺序不会改变，可以用 `set` 维护所有和扫描线接触的线段，插入一条线段的时候和前驱后继连边，这样就可以得到遮挡关系的 DAG，第二问就解决了。

对于第一问，可以对四个方向的移动分别求出最早不合法的的移动，以向上移动的线段为例，考虑线段 u 在向上移动的过程中被线段 v 阻碍的条件是什么，在遮挡关系的 DAG 中 v 能到达 u 并不意味 v 会阻碍 u，因为遮挡关系不具有传递性，但再加上一个 u 和 v 的横坐标区间相交的条件就正确了，即：

- v 的拓扑序比 u 小。
- u 和 v 的横坐标区间有公共点。

不过仍然不好做，正难则反，把操作序列倒过来，删线段就变成了加线段，加入一条横坐标区间为 $[l, r]$ 的线段时，不合法当且仅当这个区间内有拓扑序更小的线段，离散化后线段树维护即可。

复杂度 $O(n \log n)$。