T1 - 竹竿

我们声明：取所有 b_i 和 $a_i - b_i$ 不去重的最大值和次大值的和即为答案。

证明：所有竹竿的标记在同一个点上，那么也就是说，对于一个竹竿，其被划分成为两部分，一个放于标记左边，一个放于标记右边。答案就是标记左边长度的最大值加标记右边长度的最大值。

考虑上述的最大值加次大值，这显然是答案的上界；如果它位于同一根竹竿上，那么已经满足条件，否则我们只需要调整这两根竹竿的摆放位置，即可达到目标。

时间复杂度 $\mathcal{O}(n)$。

T2 - 黑洞

暴力做法一

考虑 $n = 2$ 的时候，我们做水平和竖直铅垂线划分整个平面。对于每一部分，由于横纵坐标变化值相等，所以只有两个维度的长度都足够的情况下格子才会变为黑色，那么黑色格子的数量即两个维度长度的较小值。那么，答案为：
$\min(a_1 - 1, a_2 - 1) + \min(a_1 - 1, m_2 - a_2) + \min(m_1 - a_1, a_2 - 1) + \min(m_1 - a_1, m_2 - a_2) + 1$
。

对于 $n \geq 3$ 的时候也是同理，只不过我们需要枚举 2^n 个 $\min$，可以得到 $\mathcal{O}(2^n)$ 的做法。

暴力做法二

如果 $k = 0$，与黑洞重合，特判。考虑枚举 $k \geq 1$，对于第 i 个维度，在这个维度里，如果向一个方向上延伸，需要满足 $k \leq a_i - 1$，在对立的方向上，需要满足 $k \leq m_i - a_i$。如果两个都满足，那么这时的贡献是 $2\times$，因为两个方向都可以选择；如果仅满足一条，贡献是 $1\times$；否则贡献为 $0\times$。把所有贡献相乘即可。

这可以得到 $\mathcal{O}(nV)$ 的做法。

正解

我们可以从以上两个解法的任意一种入手进行优化。这里提供一个形式化一些的做法描述：对于一对 $(a_i - 1, m_i - a_i)$，设 p 为其中的较小值，q 为其中的较大值，看做维护一个长度充分长的初始全为 1 的数组 res，将 $i \geq q$ 的 res_i 设为 0，将 $i \leq p$ 的 res_i 乘以 2。最终要求 $\sum\limits_{k \geq 1} res_k + 1$。

可以将所有给出的 q 取最小值，这给出了 $res_k \neq 0$ 的 k 的上限。将给出的所有 p 排序，记为 $p_1 \dots p_n$，令 $p_0 = 0$，我们就知道在 $p_i < k \leq \min(p^{i+1}, q)$ 时，所有 res_k 均为 2^{n-i}，因为后面的所有 p 都对应于一个任意选择的维度。于是排序后可以很容易计算答案。

时间复杂度 $\mathcal{O}(n \log n)$。

T3 - 水星湖

因为本题的题解整理的时候较零碎，我们给出一些不同作者写的做法。大家可以自己选择较能让人理解的做法理解。

使用一个数组表示网格上的情况，对于整个过程中所有活的树，直接将其添加到网格中。

使用一个队列维护所有在上一秒长出树的生长时间等信息。每一秒，对队列中的树判断其上下左右四个格子是否会生长出树，若生长了，则将该位置入队列。

再使用一个队列维护所有刚种下时没有与其他树或湖相邻的树，对于第 i 棵树，如果其入了该队列，那么在 $t_i + k$ 秒时再次判断其是否与其他树或湖相邻，如果没有，那么删去这棵树。最后统计结果即可。

可以发现，只要一个格子中只种过一棵树，那么这个格子最多只会进入第一个队列一次。但是，如果一秒一秒枚举，可能会导致 TLE。但是，实际上过程中至多有 $nm + r$ 棵树入过第一个队列，于是我们可以直接把空队列的时刻直接跳过，一直跳到下一个 t_i 即可，因为在这期间第一个队列一定是空的。那么整体的时间复杂度即为 $\mathcal{O}(nm + r)$。

我们把一棵树种下去后，若其附近有湖且能因这棵树长出其他树的行为称为这棵树在湖边蔓延。

通过定义可以发现，如果一棵树不能在湖边蔓延，那么它也不可能被其他从湖边蔓延的树给相邻使得其存活。

那么我们便可以将所有的树分成两组：可以在湖边蔓延的、不可以在湖边蔓延的。对于可以在湖边蔓延的，我们先让这些树全部蔓延完毕直到不能再蔓延（即 dfs），然后对不可以在湖边蔓延的使用做法一中的第二个队列维护即可。

时间复杂度 $\mathcal{O}(nm + r)$。

一个很关键的性质是，一棵树如果周围有树，或者周围有湖，就不会死。

虽然这是题面告诉你的，但是稍微想一下就知道，每个位置只可能有过树然后死掉了，或者根本没有过树，或者种下的树一直活着。

再想一下就知道，如果树的生长蔓延了，被蔓延到的位置全都是永久存活的。这个蔓延会形成一个个绕着湖的**联通块**，中间靠一些零散的树连接起来，联通块内部一定是树挨着的，满足『周围有树』的生长条件。

那么如果我们发现这棵树种下之后会永久存活，就从这棵树开始搜索，搜出一个全都能生长树的联通块，标记成永久存活的位置即可。

复杂度是正确的，因为如果已经被标记为永久存活，下一次搜到这里想给它打标记，就可以直接不管这棵树。这样一个位置只可能被标记一次，复杂度正确。

处理树的死亡可以开一个队列储存。每一次种树的时候看一下队首元素，如果生长了超过 k 秒就尝试让它死亡。只有标记成永久存活的位置的树才可以不死亡。

进行一些简单的预处理：

- 因为湖不会相交，我们可以对每个湖 $a_{i,1}, b_{i,1}, a_{i,2}, b_{i,2}$ 暴力标记其内部位置。时间复杂度不会超过 $\mathcal{O}(nm)$；
- 得到湖的位置之后，枚举每个陆地的相邻方块，可以得到哪些陆地是与湖相邻的；

- 由于在相同的 (x_i, y_i) 种树时，除最后一次以外，前若干次种下的树最终都已死亡，这说明前若干次都没有对整个地图产生影响。**如果一棵树的出现使其它格子里长出了树，那么它显然最终也不会死亡**。可以采取一些方式，忽视前若干次的 t_i。

  考虑如果没有 k 的限制，也就是说树不会死亡，怎么做。

  我们可以使用 BFS。如果一个位置 (x, y) 长出了一棵树，或被种下了一棵树，就把它加入队列中。对于一棵树 (x, y)，枚举所有和它相邻的 (x_0, y_0)，如果满足题目描述中的另外两点（与湖相邻的陆地；没有树），就将其标记为长出一棵树，加入队列。每个节点仅会被加入一次队列，时间复杂度为 $\mathcal{O}(nm)$。

  有了 k 的限制呢？同样采用 BFS，相当于求解一个不同时间依次加入一些源点的最短路问题，这可以得到，对于每个位置 (x, y)，如果不考虑所有树都会死亡，那么它最早长出的时间是多少。具体实现上，可以采用两个队列，其中一个队列存放给出的 (x_i, y_i, t_i)，另一个队列存放被扩展出的点。两个队列里的 t 是分别单调的。如果我们对一个相同的点只在第一次出队时更新，那么时间复杂度同样是 $\mathcal{O}(nm)$。

  由于一个位置 (x, y) 被自动生长出而不是被钦定种下时，其相邻的位置一定有树，而 **如果一棵树的出现使其它格子里长出了树，那么它显然最终也不会死亡**；所以，事实上，最终死亡的树的位置一定在给出的 (x_i, y_i) 之中。死亡的唯一可能性是：四联通格子里既没有水，又没有一颗长出的时间不超过 $t_i + k$ 的树。我们可以 **按照时间顺序，也即给出的顺序，依次判定**，如果均不存在，则视为此格无树。

  因为按照时间顺序进行了判定，所以不会有跨越时空的影响，也即：一棵树本来应该是死的，但是因为还没有判定它死，它救活了另一颗本来该死的树。由此可以保证该做法的正确性。

  时间复杂度 $\mathcal{O}(nm + r)$，如果在 nm 上带有小常数的 $\log$ 也有可能通过。

T4 - 2048

- 先把所有权值取 $\log_2$，那么失败状态里，所有数在 $[1, x - 1]$ 之间。
- 可以证明，最终的失败状态内得到的 n 个格子里的数 $a_1, \ldots, a_n$ 满足其严格单峰，也即存在 $1 \le k \le n$，使得 $a_1 < a_2 < \cdots < a_k > a_{k+1} > \cdots > a_n$；
  - 考虑归纳。先忽略棋盘上所有空位。不妨假设对于总共进行了 t 次操作后的 2048 状态，一定是非严格单峰的，也即存在 $1 \le k \le n$，使得 $a_1 \le a_2 \le \cdots \le a_k \ge a_{k+1} \ge \cdots \ge a_n$；我们证明对于 $t + 1$ 步的所有可能性，一定是非严格单峰的。具体证明从略。
  - 考虑如果最终状态存在 $a_i = a_{i+1}$，一定可以滑动消除，因此不是结束状态。那么结束状态一定是单峰的。由此我们可以知道，所有方块的出现时间里，也是最大的先出现，而左侧的方块和右侧的方块的出现时间 **分别具有单调性。**
  - 考虑一个数 i 被合成的时候，至少有一个瞬间出现了 $1, 1, 2, 3, 4 \ldots (i - 1)$，最后合并成一个 i。所以一个数 i 出现的时候，必须有 **至少 i 个剩余空位。**
  - 考虑从两端往中间进行动态规划（本质上，最终状态里的方块是从中间向两边产生的；我们只关心两端的状态，这告诉我们有多少个剩余空位，所以可以同时解决不同 n 时的答案）。
  - 设 $dp_{m,l,r}$ 表示当前左边确定了 m 个数，左边的最右端数是 l（如果不存在则为 0，下同），右边的最左端数是 r，**并且必须有** $\max(l, r) \le m$。初始状态为 $dp_{0,0,0} = 1$，转移为：

$$dp_{m,l,r} = \sum_{0 \le i \le l-1} dp_{m-1,i,r} + \sum_{0 \le j \le r-1} dp_{m-1,l,i}$$

- 两个求和分别对应上一个出现的数在左侧或在右侧。这个 dp 的计算方式可以同时完美的和出现时间的不同构成一一对应关系。
- 计算一对 (n, x) 的答案时，考虑枚举最大的数统计答案，答案是 $\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(x-\max(i,j)-1)dp_{n-1,i,j}$。
- n, x 必定同阶。使用前缀和优化 dp 转移和统计答案部分，即可以做到 $\mathcal{O}(n^3)$ 预处理，$\mathcal{O}(1)$ 回答。
- Bonus：能不能再给力一点啊？