大魔法师(magic)

都是区间操作，尝试使用线段树进行维护。

维护区间内 A, B, C 的和以及区间长度 len，令其构成一个向量 $\begin{bmatrix} A \\ B \\ C \\ len \end{bmatrix}$。

发现前六种操作都相当于将向量右乘上一个矩阵：

1. $\begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$
2. $\begin{bmatrix} 1 & & & \\ & 1 & & \\ & 1 & 1 & \\ & & & 1 \end{bmatrix}$
3. $\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$
4. $\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ v & & & 1 \end{bmatrix}$
5. $\begin{bmatrix} 1 & & & \\ & v & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$
6. $\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & & \\ & & v & 1 \end{bmatrix}$

使用懒标记维护该矩阵，时间复杂度 $O(n + m \log n)$。

盗道(thief)

考虑有足够多个窃贼，他们从第 1 家开始行窃，他们初始的库纳数量为：

$\cdots -4, -3, -2, -1, 0, 1, 2, 3, 4 \ldots$。

假设第一家的库纳数量为 2，那么在这一家进行行窃之后，它们的金币数量会变为：

$\cdots -3, -2, -1, 0, 1, 2, 2, 2, 3 \ldots$。

发现这个过程中，所有窃贼的库纳数量相对于初始而言是单调的，而在差分数组的意义上，就是将库纳树等于 2 的极长串左右两侧的 1 变为了 0。

如果有盗贼不是从 1 开始的，那么可以在上面那些盗贼扫描到 l 的时候，找到某一个当前库纳数等于 c 的盗贼（一定存在）与之对应。在扫描到 r 的时候查询这名盗贼的库纳数即可。

现在的问题变成了需要维护一个初始全为 1 的序列，支持如下操作：

1. 查询第 k 个 1 的位置。
2. 将某一个位置上的 1 改为 0。
3. 查询 $1 \sim k$ 中共有多少个 1。

使用树状数组维护，后两个操作就是单点修改和前缀和。而第一个操作可以通过树状数组上二分来实现。

时间复杂度 $O((n+m)\log n)$。

Alice 和 Bob 又在玩游戏(game)

这是一个公平组合游戏，所以我们考虑直接计算 SG 函数。

如果这一次决策选择删掉的点是 x，那么这个状态的 SG 值就是 x 所有儿子，以及 x 及其所有祖先的兄弟的 SG 值的异或和。

我们对于每一个节点维护其后继状态的 SG 值的集合，发现 u 的集合就是对于其每一个儿子 v 的集合，将每一个数异或上 v 所有兄弟的 SG 值之后，并在一起，然后加入所有儿子 SG 值的异或和这个状态。

这个过程可以使用 01Trie 合并来实现，朴素实现的时间复杂度为 $O(n\log n)$，空间复杂度为 $O(n\log n)$。

无法通过最后一档 128MB 的部分。

优化线段树合并形问题的空间的方法一般就两种：

1. 写压缩线段树。
2. 在优先遍历重儿子的情况下写节点回收。

这两种情况需要的节点数都可以证明是至多 $2n$ 的。

std 实现的是第二种方式。具体实现为：

先对原树进行树链剖分，dfs 的过程中优先遍历重儿子，然后遍历轻儿子。同时在线段树合并的过程中，被合并后不用的节点都要回收到当前的空节点集合中。

但是 dfs 有很大的空间常数，仍然可能会导致 MLE，所以通过手写栈来代替 dfs 即可。

简单数据结构(easy)

考虑如果初始有 $a_i = 0$，发现这时 1 操作和 2 操作都有很强的单调性。

具体的，如果序列保证单调不降，也就是 $a_i \le a_{i+1}$，那么显然有 $a_i + i \le a_{i+1} + i + 1$，所以 2 操作不会改变单调性，同时 1 操作会修改的就是一段后缀，仍然不影响单调性。

而 1 操作进行的修改，这也就意味着，如果 a_i 在某一时刻它进行了被 1 修改了，我们就可以认为 a_i 是一个初始从零开始的数。也就是**所有被 1 操作修改过的 a_i 是单调不降的。**

这也就意味着，我们如果知道每一个数被 1 操作修改的时刻，就可以分别维护没有被 1 操作修改的部分和已经被 1 操作修改过的部分。而这两个部分都是可以使用线段树维护做到 $O(q\log^2 n)$ 地。

现在的问题就变成如何求出每一个数被 1 操作修改的时刻了。

考虑对于一次 1 操作 v，如果它前面有 c 次 2 操作，它要修改 a_i 至少要有 $a_i + ci > v$，移项成 $a_i > v - ci$，那么我们就是要找到第一个 $v - ci < a_i$ 的位置，而前缀最小值 $\min(v - ci)$ 是可以进行二分的。

所以我们可以使用整体二分+李超线段树找到每一个 a_i 被修改的时间，时间复杂度 $O(n \log n \log q)$。