A

f_i 表示只管前 i 个彩灯的答案，最后输出 f_n。有 DP：

$$f_i = \sum_{j=1}^{i} f_{j-1} \cdot \operatorname{mex}_{k=j}^{i} a_k$$

容易发现 i 固定时 $\operatorname{mex}_{k=j}^{i} a_k$ 关于 j 一定是不增的，考虑维护每个 mex 的值对应的 j 的区间和 f_i 的前缀和来快速转移。

当在序列末尾增加了一个数 a_{i+1} 时，只有 $\operatorname{mex} = a_{i+1}$ 的区间 $[l, r]$ 的值会改变。先计算出 $[r, i]$ 的 mex 值 w，那么区间 $[\max(l, las_w + 1), r]$ 的 mex 就是 w，其中 las_w 表示 w 在 $[1, i]$ 中最后一次出现的位置，然后接着处理剩下的区间。

计算区间的 mex 值，用权值线段树维护 las，然后在线段树上二分第一个 las 小于询问位置的地方。

每次会删除一个区间，增加一些区间，但增加的区间除了最后一个区间外，mex 值原来是不存在的。由于只可能有 n 个不同的 mex 值，每次只会删除一个 mex 值，所以总共只有 $O(n)$ 次询问 mex 值和找对应的区间。

mex 值对应的区间变化后，记得维护一下对转移的贡献。

时间复杂度 $O(n \log n)$。

B

发现 1：

若 $a \le b \le c$ 一定有 $a \oplus c \ge \min(a \oplus b, b \oplus c)$，那么集合 S 中只有相邻的 (i, j) 才会对答案有影响。证明考虑 $a \oplus c$ 中第一位不为 0 的地方，$a \oplus b$ 或 $b \oplus c$ 中一定有一个这一位为 0。

发现 2：

若 $a \le b$ 一定有 $b \oplus a \ge b - a$。

发现 3:

对于一对相邻的 (i, j)，枚举 x，只保留有用的 x，即异或值是前缀 $\min$，发现只有 $O(V)$ 个。且一定是让 $i + x$ 或 $j + x$ 的二进制下后 $k, 0 \le k \le V$ 位全是 0 的时候。证明考虑 $i + x$ 的后 k 位是 0，继续让 x 变大。若 $j + x$ 向 $k + 1$ 位进位了，那么情况变成 $j + x$ 的后一些位为 0 了；否则不进位的情况下，只考虑后 k 位，根据发现 2 肯定不可能更优。

有前两个发现后，设 k 为 $2^k \ge |i - j|$ 的最小值，那么 $x \ge k$ 时一定可以让较小数后 k 位位变成 0，达到异或的最小值 $|i - j|$。记 $mn = \min(|i - j|)$ 即所有相邻对差的最小值，那么 $x \ge mn$ 时 $ans = mn$，否则枚举答案。由于 $mn \le \frac{2^V}{|S|}$，所以 sub2 时间复杂度 $O(2^V)$，sub3 时间复杂度为调和级数 $O(V2^V)$。

有前三个发现后，用 set 维护一下有用的二元组 $(x, (x + i) \oplus (x + j))$ 即可。时间复杂度 $O(nV \log n)$。

C

先考虑 sub4，分别维护一下节点 $[l, 0]$ 和 $[k + 1, r]$ 连出去的所有边形成的生成树（此生成树可能包含 $[1, k]$ 的点），再维护一下边的两个端点全在 $[1, k]$ 的生成树（此生成树只包含 $[1, k]$ 的点且不会改变）。求答案时将 3 棵生成树的边拿出来重新求最小生成树即可。

容易发现 $[l, 0]$ 的生成树中只有 $[1, k]$ 路径上的最大的边才可能最终不在答案的生成树上，由于需要插入，所以维护 $[l, l + k - 1] \cup [1, k]$ 点集的虚树，边权为缩掉的边的 $\max$。这样插入时就对新增的 $k - 1$ 条边和原来虚树上的 $O(k)$ 条边一起建最小生成树然后建虚树，把所有后缀的生成树 $[i, 0], l \le i \le 0$ 全部记录下来，这样方便删除。

$[k + 1, r]$ 的生成树也同理维护，这样求答案时只需对 $O(k)$ 条边求最小生成树，时间复杂度 $O(nk \log k)$。

正解的话，还是一样的维护，只是中间的 k 个点就不一定是 $[1,k]$ 了。当点数 $\le k$ 时就 prim 暴力，否则一样维护 3 棵生成树，若最左或者最右的生成树被删除完了，并且开始删中间 k 个点时就重构这个结构。重新选现在区间中中间的 k 个点 $[ol,or]$，然后两边重建生成树。

将势能设置为 $\max(ol-L,R-or)$，插入会让势能 +1，重构会让势能减半，但重构复杂度为 $O(\text{势能}\cdot k\log k)$。

因为有 prim，时间复杂度 $O(nk^2)$，但运行时瓶颈在插入的 $O(nk\log k)$。
