回文

题目大意

给定一个只含小写字母的字符串 S，有 Q 次操作，操作有以下两种：

`1 i c`：将 S_i 修改为 c（$1 \le i \le |S|$，c 是小写字母）。

`2 l r`：查询 $S[l \dots r]$ 有多少回文后缀（$1 \le l \le r \le |S|$）。

本题**强制在线**，如果是操作 1，那么要将输入的 i 异或上 $lastans$，如果是操作 2，那么要将 l 和 r 都异或上 $lastans$。$lastans$ 是上一次操作 2 的答案，如果没有上一次操作 2，则 $lastans = 0$。

数据范围

对所有数据满足：

$1 \le |S|, Q \le 2 \times 10^5$。

详细子任务附加限制及分值如下表所示：

子任务	附加限制	分值	子任务依赖
1	$1 \le \|S\|, Q \le 5 \times 10^3$	10	
2	没有操作 1	10	
3	S_i 和 c 在 $\{'a','b'\}$ 中均匀独立随机，操作 1 中 i 在 $[1, \|S\|]$ 中均匀独立随机	10	
4	操作 2 中 $l = 1, r = \|S\|$	10	
5	$1 \le Q \le 5 \times 10^4$	30	1
6	无附加限制	30	$1, 2, 3, 4, 5$

解题过程

做法 1

对于一个询问，枚举 i 从 l 到 r，使用字符串 hash 检验 $S[i \dots r]$ 是否回文，累加答案。

时间复杂度 $O(Qn)$。可以通过子任务 1。

做法 2

如果没有操作 1，我们可以建出 S 的回文自动机，对于一个询问，转化为求回文树上的节点 u 有多少个祖先对应串长度不超过 $r - l + 1$，可以直接倍增二分找到对应祖先，求出答案。

时间复杂度 $O(n + Q \log n)$。可以通过子任务 2。

做法 3

如果数据随机，那么回文后缀长度很小，不超过小常数 K，直接枚举回文后缀长度，判断这个后缀是否回文即可。

时间复杂度 $O(n+QK)$。可以通过子任务 3。

做法 4

如果操作 2 中 $l=1,r=|S|$，那么可以根据加密后的 l 异或 1 得到上一个询问的答案，因此只有最后一个询问的答案需要回答，用做法 1 解决即可。

时间复杂度 $O(n+Q)$。可以通过子任务 4。

做法 5

有个经典结论，字符串 s 的回文后缀按照长度排序后，可以划分成 $\log|s|$ 段等差数列。证明：见参考资料1。

我们尝试增量维护这些等差数列：

考虑当前字符串 s，在末尾新增了一个字符 c，新串的回文后缀将会是：

1. 单独一个字符 c。
2. 原先的回文后缀在两端同时增加字符 c。

第一种情况新增的回文后缀就是单独一个字符 c。

第二种情况，考虑原先的一个极长的等差数列，定义 S^k 表示 k 个串 S 拼接形成的串，将这个等差数列表示的串表示成：$A,BA,BBA,B^3A,\dots B^kA$。此时如果 B 的最后一个字符是 c，那么这个等差数列除了 B^kA 都出现在新串的回文后缀中的；如果不是 c，那么这个等差数列除了 B^kA 都不出现在新串的回文后缀中的。而最后一项只需要检验它左边的字符是否是 c 即可判断它是否出现在新串回文后缀中。

做完这个，再把相邻的公差相同的等差数列合并成一个等差数列，以保证等差数列是 $O(\log|s|)$ 段。

如果没有操作 1，处理每个 $S[1\dots i]$ 的回文后缀形成的 $O(\log|s|)$ 段等差数列。对于一个询问，枚举一个 $S[1\dots r]$ 的回文后缀的等差数列，算这个等差数列对答案的贡献。

时间复杂度 $O((n+Q)\log n)$。可以通过子任务 2。

做法 6

沿用做法 5 的思路，显式的维护出 $O(\log|s|)$ 段等差数列。

做法 5 是每次增加一个字符 c，能不能一次增加多个？

对于一个串，用 $O(\log|s|)$ 段等差数列的形式维护出这个串的回文前缀和回文后缀，然后尝试合并两个串的信息。

回文后缀和回文前缀的维护方式是相同的，不妨假设现在只需要维护新串的回文后缀：

与做法 5 类似的，考虑当前字符串 s，在末尾加上字符串 t，新串的回文后缀将会是：

1. 不经过 s 的回文后缀。
2. 经过 s 的回文后缀，且中线在 t。
3. 经过 s 的回文后缀，且中线在 s 和 t 的中线。
4. 经过 s 的回文后缀，且中线在 s。

第一种情况，直接从 t 里继承即可。

第二种情况，发现这个新串回文后缀的中线肯定也是 t 的某个回文前缀的中线，也就是这种情况新串回文后缀肯定是从 t 中回文前缀扩展得到的。枚举 t 中的回文前缀，考虑现在有一个极长的回文串的等差数列：$A, AB, ABB, AB^3, \ldots, AB^k$。那么 $t = AB^kC$，C 是剩余部分，且 B 不是 C 的前缀。如果 C 是 B 的前缀，那么这个等差数列的一个后缀会对新串回文后缀贡献形成一个新的等差数列，这个等差数列的长度取决于 s，可以二分得到。否则 C 不是 B 的前缀，那么最多只有一项会成为新串的回文后缀。将 AB^k 往左扩展，扩展成 $B'^{k'}AB^k$ 的形式，这里 B' 是 B 的反串，k' 是尽可能大的向左扩展次数，此时只要判断以 $B'^{k'}AB^k$ 为回文中心，右端点为新串右端点的串是否是回文后缀，如果是，加入新串回文后缀中。

第三种情况最多只会新增一个回文后缀，直接判断有无新增即可。

第四种情况与做法 5 的第二种情况相似，新串的回文后缀会从 s 的回文后缀中扩展得到。原先是头尾增加单个字符 c，现在是头加 t 的反串，尾加字符串 t，解决思路基本相同。

如果精细实现，用 $O(\sqrt{n}) - O(1)$ 分块来处理动态修改的字符串 hash 问题，合并的时间复杂度是 $O(\log(|s| + |t|))$，因为当前等差数列会被完全包含在下一个等差数列的 A 中，且 B 的长度比 A 大，因此 k 之和是 $O(\log(|s| + |t|))$ 的，而第二种情况中 k' 是不会超过 k 的，因为中线在 t，k' 比 k 大中线不可能在 t。因此二分 k' 的复杂度也是 $O(\log(|s| + |t|))$ 的。

有了合并，使用线段树维护区间信息，单点修改和区间查询的问题迎刃而解，总的复杂度是 $O(n \log n + Q(\log^2 n + \sqrt{n}))$。可以通过整道题目。

参考资料

1. OI wiki
2. 2017 年 IOI 国家候选队论文集 回文树及其应用 翁文涛
3. 2019 年 IOI 国家候选队论文集 子串周期查询问题的相关算法及其应用 陈孙立
4. 字符串算法选讲 金策