1 字符游戏

1.1 题目大意

Alice 和 Bob 正在玩游戏。初始时有一个字符串可重集 S，Alice 和 Bob 轮流操作，Alice 先手。每次可以选择一个 S 中的字符串 s，将其从 S 中删除，并选择一个在 s 中出现过的字符 c，将 s 中的所有字符 c 删除，并沿着这些删除的位置将 s 划分成若干子串，将这些子串中非空的加入进集合 S。不能进行操作的人输。

现在给定一棵树，节点数为 n，每个节点上有字符。记 $\text{path}(u,v)$ 表示将节点 u 到 v 的最短路径所经过的节点（包括 u 和 v）上的字符依次拼接而成的字符串。共 q 次询问，每次询问给定 u 和 v，求 $S=\{\text{path}(u,v)\}$ 时的胜者。若 Alice 获胜，同时输出第一步有多少种走法。

1.2 数据范围

保证对于所有测试点满足：$1\le n\le 5\times 10^4$，$1\le q\le 10^5$，$0\le s_i<10$。

子任务编号	$n\le$	$q\le$	$s_i<$	特殊性质	分数
1	10	10^3	10	无	7
2	500	2×10^4	10	A	13
3	$3,000$	2×10^4	10	A	12
4	5×10^4	10^5	10	A	19
5	2×10^4	2×10^4	5	无	24
6	5×10^4	10^5	10	无	25

特殊性质 A：保证树的形态为一条链。

1.3 解题过程

1.3.1 算法 1: 暴力 dp

对于每次询问暴力求出 $\text{path}(u,v)$。注意到任意时刻的集合 S 都是由 $\text{path}(u,v)$ 中若干个极长子段得到的，因此状态数不超过 $\mathcal{O}(2^l)$，暴力 dp 即可。

可以通过子任务 1。

1.3.2 算法 2: 区间 dp 维护 SG 函数

考虑计算 SG 函数，这样每个子问题都是求一段区间的 SG 值。

由于树的形态是一条链，可以对于每个区间预处理 SG 值，枚举该轮操作的字符并暴力拆分成子问题即可做到 $\mathcal{O}(n^3)$。

可以通过子任务 2。

1.3.3 算法 3: 加速区间 dp 的转移过程

考虑加速算法 2 中的转移过程。

记 $f_{i,j}$ 表示区间 $[i,j]$ 的子问题，辅助状态 $g_{i,j}$ 表示将区间 $[i,j]$ 按照字符 s_{i-1} 拆分成子问题后的 SG 值异或和。只需再预处理每个位置的下一个对应字符的位置即可 $\mathcal{O}(|\Sigma|)$ 转移 f 以及 $\mathcal{O}(1)$ 转移 g。

可以通过子任务 2, 3。

1.3.4 算法 4: 对区间 dp 有效状态数的观察

考虑有效的 dp 状态共有多少种。

记 $fl_{i,c}$ 表示从 i 之前第一个字符 c 之后到 i 的 SG 值，$fr_{i,c}$ 表示从 i 到 i 之后第一个字符 c 之前的 SG 值。

先进行一些预处理。记某个状态 $[i,j]$ 为关键状态当且仅当 $s_{i-1}=s_{j+1}$ 且 $[i,j]$ 中不存在字符 s_{i-1}。不难发现关键状态只有 $\mathcal{O}(n)$ 种。

接下来考虑暴力预处理所有关键状态的 SG 值。对于某个关键状态 $[i,j]$，枚举字符 c 从 $fr_{i,c}$ 和 $fl_{j,c}$ 进行转移，再暴力访问 $[i,j]$ 之间所有字符 c 的关键状态进行转移。注意 $fl_{i,c}$ 和 $fr_{i,c}$ 不必对于每个 i 和 c 求出，只需在求关键状态时进行记忆化即可。

考虑这样转移的时间复杂度。对于某个关键状态 $[i,j]$，其只有在左端点前是 i 之前第一次出现的字符并且右端点后是 j 之后第一次出现的字符时会转移过去。因此预处理的时间复杂度是 $\mathcal{O}(n|\Sigma|^2)$。

查询时就可以对每个字符的关键状态做前缀和，然后前后缀非关键状态的部分要么是已经预处理过的 $fl_{i,c}$ 或 $fr_{i,c}$，要么仍是查询串的前缀并且右端点后是 l 之后第一次出现的字符，或者类似的后缀。总复杂度 $\mathcal{O}(n|\Sigma|^2+q|\Sigma|)$。

可以通过子任务 2, 3, 4。

1.3.5 算法 5: 标准算法

上述序列上的做法无法拓展到树的情况。

我们希望使用一些树上数据结构来维护 SG 值的信息，至少需要支持前后添加字符以及信息的合并。

考虑在当前串 s 的末尾添加一个字符 c，会对 SG 值造成什么变化。

我们不太容易直接描述整串的 SG 值 f 的变化，因此考虑记 f_i 表示对字符 i 进行操作后，得到的子串集合的 SG 值异或和。len_i 表示 i 上次出现的位置之后有多少个字符，未出现则为 -1。显然 $f=\operatorname{mex}_{len_i\neq -1}\{f_i\}$。

len_i 容易维护。那么考虑加入字符 c 后，f_i 会发生的变化：

- $i=c$，若 $len_i\neq -1$ 则 $f_i'\leftarrow f_i$ 无变化。否则，$f_i'\leftarrow f$，即修改前整串的 SG 值。
- $i\neq c$，若 $len_i=-1$ 则未定义。否则，这涉及到 s 长度为 len_i 的后缀添加字符 c 后如何变化。

记 s_i 表示 s 长度为 len_i 的后缀。记 g_i 表示 s_i 的 SG 值。类似地，记 g_{ij} 表示在 s_i 对字符 j 进行操作后，得到的子串集合的 SG 值异或和。

当 $i \neq c$ 时，直接修改 s_i 的异或和显然是正确的。即 $f'_i \leftarrow f_i \oplus g_i \oplus g'_i$。

那么 g_i 的修改也是和 f 类似的。注意到只有当 $len_j \neq -1$ 且 $len_j < len_i$ 时，g_{ij} 才有定义，因此 g_i 的修改只依赖于满足上述条件的 g_j，转移不会成环。

这样，我们就在 $\mathcal{O}(|\Sigma|^2)$ 的时间复杂度内解决了末尾添加字符的问题。在开头添加字符也是类似的，并且对上述算法简单修改，对前后缀分别维护 g 和 len，就能做到同时在开头和末尾添加字符。

方便起见，记上标 $^{[p]}$ 表示前缀的信息，上标 $^{[s]}$ 表示后缀的信息。

现在考虑如何合并两个串 s 和 t。记 h_{ij} 表示 $s_i^{[s]} + t_j^{[p]}$ 的 SG 值，计算时枚举这一步操作的字符 k，发现与 $sg_{ik}^{[s]} \oplus tg_{jk}^{[p]}$ 不同的只有跨过 s 和 t 分界点的那一段 h_{kk} 或 h_{ik} 或 h_{kj}，分析发现转移同样不会成环。计算出 h_{ij} 后，容易计算出新的 f' 和 g'。这部分复杂度显然是 $\mathcal{O}(|\Sigma|^3)$。

考虑最终答案的计算。使用点分治求出 $u \to p$ 和 $p \to v$ 的信息，其中 p 为 u 到 v 在点分树上的 LCA。最后再合并两段信息，时间复杂度容易做到 $\mathcal{O}(n \log n|\Sigma|^2 + q|\Sigma|^3)$，空间复杂度 $\mathcal{O}(n+q|\Sigma|^2)$。

至于求出 $u \to v$ 的信息后如何计算第一步有多少种走法，只需要计算有多少个 $f_i = 0$ 就可以了。

可以通过所有子任务。如果维护的信息较劣或者添加字符/合并的复杂度较高则应该也可以通过子任务 1, 2, 3, 5。

1.4 参考资料

无