可持久化数据结构

李淳风

长郡中学

2024 年 7 月 15 日

介绍

到目前为止，我们学习的数据结构都是维护“数据集的最新状态”。如果想知道数据集在任意时间的历史状态 ($\forall i \in [1, m]$，执行完操作序列中的第 i 项操作后数据集的状态)，一种朴素的做法是，在执行完第 i 项操作之后，把整个数据结构拷贝一遍，花费 m 倍的空间。
“可持久化”提供了一种思想，在每项操作结束后，仅仅创建数据结构中发生改变的部分的副本，不拷贝其它部分。这样一来，维护数据结构的时间复杂度没有增加，空间复杂度仅增长为与时间同级的规模。换言之，可持久化数据结构能够高效地记录一个数据结构的所有历史状态。

可持久化 Trie

与 Trie 的节点一样，可持久化 Trie 同样使用 $trie[x,c]$ 保存，从节点 x 出发，经过字符 c，到达的子节点编号。可持久化 Trie 按照以下步骤插入一个新的字符串 s：

- 设当前可持久化 Trie 的根节点为 $root$，令 $p = root, i = 0$。
- 建立一个新的节点 q，令 $root' = q$。
- 若 $p \neq 0$，则对于每种字符 c，令 $trie[q,c] = trie[p,c]$。
- 建立一个新的节点 q'，令 $trie[q,s_i] = q'$。换言之，除了经过字符 s_i 到达的子节点不同，q 与 p 的其它信息完全相同。
- 令 $p = trie[p,s_i], q = trie[q,s_i], i = i+1$。
- 重复步骤 $3 \sim 5$，直到 i 达到字符串末尾。

这样一来，从 $root$ 出发能到达的节点构成原本的 Trie，从 $root'$ 出发到达的节点构成插入 s 之后的 Trie。我们可以使用 $root$ 数组来记录这些根节点，$root[i]$ 表示插入 $s_1 \sim s_i$ 后的根节点。
由于每次新建的节点数只有 $O(|s_i|)$ 个，因此构建可持久化 Trie 所需的空间和时间复杂度都和字符串总长度线性相关。

例题

最大异或和

给定一个非负整数序列 a，初始长度为 n。有 m 个操作，每个操作是以下两种之一：

- "A x"，添加操作，表示在序列末尾插入一个数 x，序列的长度 n 增大 1。
- "Q l r x"，询问操作，求一个位置 p 满足 $l \le p \le r$，使得 $a_p\ xor\ a_{p+1}\ xor\ \cdots\ xor\ a_n\ xor\ x$ 最大。输出这个最大值。

$n, m \le 3 * 10^5, 0 \le a_i \le 10^7$。

设 s_i 表示序列 a 的前 i 个数异或起来得到的结果。特别地，$s_0 = 0$。
根据异或的性质，我们可以得到：

$$a_p\ xor\ a_{p+1}\ xor\ \cdots\ xor\ a_n\ xor\ x = s_{p-1}\ xor\ s_n\ xor\ x$$

最大异或和

对于添加操作，序列 s 很容易维护。对于询问操作，问题变为：已知一个数 $val = s_n \ xor \ x$，求一个位置 p，满足 $l-1 \le p \le r-1$，使得 $s_p \ xor \ val$ 最大。
如果去掉 $l-1 \le p \le r-1$ 的限制，我们只需要把每个数都看作二进制数，以字符串的形式 (高位在前) 插入 Trie 中。询问时按位考虑，在 Trie 中优先尝试访问与 val 的每一位相反的指针即可。
如果只有 $p \le r-1$ 的限制，可以直接用可持久化 Trie 代替上述解法中的 Trie。因为可持久化 Trie 保留了 $\forall i \in [0, n], s_0 \sim s_i$ 构成的 Trie 树，所以每次询问从 $root[r-1]$ 出发，同样操作即可。
回到原题，再来考虑 $p \ge l-1$ 的限制如何满足。我们可以在 Trie 树上多记录一下信息。设 $end[x]$ 表示节点 x 是序列 s 中第几个二进制数的末尾节点（若不是结尾，则 $end[x]$=-1），$last[x]$ 表示在以 x 为根的子树中 end 的最大值，表示从 x 出发能到达的最靠右的二进制数编号。
从 $root[r-1]$ 出发按位考虑时，只考虑 $last$ 值不小于 $l-1$ 的节点即可。

可持久化

基于同样的思想，也就是每次对于会被改变信息的节点，我们复制一份并进行修改，我们可以设计出许多数据结构的可持久化版本。但有一个前提是，数据结构的“内部结构”在操作过程中不发生变化（例如平衡树的旋转就改变了“内部结构”）。一旦数据结构的“内部结构”发生了变化，我们就需要把受到影响的节点也全部复制一份并进行修改，有可能会导致空间或者时间复杂度的爆炸。

可持久化线段树，又称函数式线段树，是最重要的可持久化数据结构之一，又被称为主席树。

由于线段树的“单点修改”只会使 $O(\log n)$ 个节点的信息更新，那么我们对于每个被更新的节点 p，我们创建该节点的副本 p'。我们同样先把 p 的所有信息（包括区间和、区间最大值）复制给 p'，然后再递归修改 p 和 p' 的子节点。

同样地，由于需要新建很多节点，使用与动态开节点相同的方式进行存储。开一个 $root$ 数组，$root[i]$ 表示进行完第 $1 \sim i$ 次操作后的线段树根节点。

例题

K-th Number

给定一个长度为 n 的整数序列 a，执行 m 次操作，其中第 i 次操作给出三个整数 l_i, r_i, k_i，求 $a_{l_i}, a_{l_i+1}, \cdots, a_{r_i}$ 中第 k_i 小的数是多少。
$n \leq 10^5, m \leq 10^4, |a_i| \leq 10^9$。

这道题的整体分治做法之前已经讲过了，现在我们来看看使用可持久化线段树的做法。
假设先不考虑 l_i, r_i 的限制，我们是可以通过建立值域线段树的方式，来求区间第 k 小的。现在有了 l_i, r_i，我们就考虑建立 n 棵值域线段树，$root[i]$ 表示把 $a_1 \sim a_i$ 都加入后的值域线段树的根节点。这样一来，如果我们在第 i 棵线段树上二分，那么我们就可以知道 $a_1 \sim a_i$ 在值域 $[L, R]$ 中的出现次数了。
那么，如果我们同时进入第 r 棵和第 $l-1$ 棵线段树，把二者统计的，在值域 $[L, R]$ 中的出现次数相减，是不是就能得到 $a_l \sim a_r$ 在 $[L, R]$ 中的出现次数了？因此我们同样对于每个值域 $[L, R]$，计算 $[L, mid]$ 数的出现次数 c_i，根据 c_i 和 k_i 的大小关系来决定进入左儿子还是右儿子。

可持久化线段树

通过这道题我们可以知道，我们不止可以对操作序列建立可持久化线段树，还可以原本对 n 个数建立可持久化值域线段树。所以如何建立，对什么建立可持久化线段树也是我们在做题时需要思考的问题。

另外需要注意的是，可持久化线段树难以支持大部分“区间修改”。因为一个节点下传延迟标记时，一旦我们创建它的左右儿子 lc, rc 的副本 lc', rc' 并更新标记，那么所有依赖 lc, rc 的线段树都要新建一份副本，甚至还需要自底向上重新更新某些信息。

因此，只有我们使用“标记永久化”来代替标记的下传，才能进行可持久化操作。但是，不是所有的标记都可以永久化，思考题目的时候需要注意。