多校联训 C 层

数据结构初探——堆, 树状数组及可持久化数据结构

何了

3-26-2025

西安交通大学附属中学

目录

堆

堆

堆是一棵树, 每个结点有一个权值 v_u, 根据其性质不同, 堆可分为**大根堆**和**小根堆**两种:

- 大根堆: 每个结点的权值都大于等于其子结点的权值的堆;
- 小根堆: 每个结点的权值都小于等于其子结点的权值的堆.

堆

堆是一棵树, 每个结点有一个权值 v_u, 根据其性质不同, 堆可分为**大根堆**和**小根堆**两种:

- 大根堆: 每个结点的权值都大于等于其子结点的权值的堆;
- 小根堆: 每个结点的权值都小于等于其子结点的权值的堆.

根据其形式不同, 堆可以分为二叉堆, 配对堆, 左偏树等. 若不加额外说明, 堆一般指二叉堆. 事实上, 在 OI 中, 单纯的二叉堆不太常用, 这是因为大部分需要用到堆的场合都可以用 STL 库中的 priority queue 替代. 不过, 堆作为基础数据结构之一依然是重要的, 可以提高我们党数据结构素养.

二叉堆

二叉堆, 顾名思义, 是一棵二叉树. 和线段树类似, 二叉堆是一棵完全二叉树. 传统意义上的二叉堆需要支持三种操作: 插入一个数, 删除序列中最小的数[1], 查询最小值. 接下来我们主要讲解这三种操作如何实现.

[1]以小根堆为例, 下同.

二叉堆

二叉堆, 顾名思义, 是一棵二叉树. 和线段树类似, 二叉堆是一棵完全二叉树. 传统意义上的二叉堆需要支持三种操作: 插入一个数, 删除序列中最小的数[1], 查询最小值. 接下来我们主要讲解这三种操作如何实现.

查询最小值: 这是最简单的操作, 只需要输出根节点的值即可.

[1]以小根堆为例, 下同.

二叉堆

二叉堆, 顾名思义, 是一棵二叉树. 和线段树类似, 二叉堆是一棵完全二叉树. 传统意义上的二叉堆需要支持三种操作: 插入一个数, 删除序列中最小的数[1], 查询最小值. 接下来我们主要讲解这三种操作如何实现.

查询最小值: 这是最简单的操作, 只需要输出根节点的值即可.

插入一个数: 像线段树一样, 假设某结点编号为 u, 则其左右子结点编号 (若存在), 则为 $2u$ 即 $2u+1$. 我们将这个数先按编号插入, 然后比较其和父亲的值, 若其值小于父亲的值, 就交换这两个结点, 直到满足堆的性质即可.

[1]以小根堆为例, 下同.

二叉堆

二叉堆, 顾名思义, 是一棵二叉树. 和线段树类似, 二叉堆是一棵完全二叉树. 传统意义上的二叉堆需要支持三种操作: 插入一个数, 删除序列中最小的数[1], 查询最小值. 接下来我们主要讲解这三种操作如何实现.

查询最小值: 这是最简单的操作, 只需要输出根节点的值即可.

插入一个数: 像线段树一样, 假设某结点编号为 u, 则其左右子结点编号 (若存在), 则为 $2u$ 即 $2u+1$. 我们将这个数先按编号插入, 然后比较其和父亲的值, 若其值小于父亲的值, 就交换这两个结点, 直到满足堆的性质即可.

删除最小值: 我们直接交换堆顶和堆底, 把堆底删除掉. 然后做插入操作的反向操作, 从堆顶开始往下, 若其值小于其子节点的值, 就交换这两个结点. 直到满足堆的性质即可.

[1]以小根堆为例, 下同.

二叉堆

二叉堆, 顾名思义, 是一棵二叉树. 和线段树类似, 二叉堆是一棵完全二叉树. 传统意义上的二叉堆需要支持三种操作: 插入一个数, 删除序列中最小的数[1], 查询最小值. 接下来我们主要讲解这三种操作如何实现.

查询最小值: 这是最简单的操作, 只需要输出根节点的值即可.

插入一个数: 像线段树一样, 假设某结点编号为 u, 则其左右子结点编号 (若存在), 则为 $2u$ 即 $2u+1$. 我们将这个数先按编号插入, 然后比较其和父亲的值, 若其值小于父亲的值, 就交换这两个结点, 直到满足堆的性质即可.

删除最小值: 我们直接交换堆顶和堆底, 把堆底删除掉. 然后做插入操作的反向操作, 从堆顶开始往下, 若其值小于其子节点的值, 就交换这两个结点. 直到满足堆的性质即可.

容易分析出这三个操作的复杂度分别为 $O(1)$, $O(\log n)$, $O(\log n)$.

[1]以小根堆为例, 下同.

优先队列

STL 提供了封装好的堆, 叫作 priority queue, 接下来介绍其用法. 定义一个优先队列需要三个参数: 类型 T, 容器 (一般为 vector), 以及一个比较方式 (默认为 less, 即大根堆).

优先队列

STL 提供了封装好的堆, 叫作 priority queue, 接下来介绍其用法. 定义一个优先队列需要三个参数: 类型 T, 容器 (一般为 vector), 以及一个比较方式 (默认为 less, 即大根堆).

priority queue 同样支持我们上述所提到的三种操作, 使用 pop 函数删除最值, top 函数查询最值, push/emplace 操作插入一个值. 同时可以使用 empty 和 size 函数来判断队列是否为空以及获取队列大小.

优先队列

STL 提供了封装好的堆, 叫作 priority queue, 接下来介绍其用法. 定义一个优先队列需要三个参数: 类型 T, 容器 (一般为 vector), 以及一个比较方式 (默认为 less, 即大根堆).

priority queue 同样支持我们上述所提到的三种操作, 使用 pop 函数删除最值, top 函数查询最值, push/emplace 操作插入一个值. 同时可以使用 empty 和 size 函数来判断队列是否为空以及获取队列大小.

priority queue 包含在 queue 库中, 除了上述介绍的 priority queue, 还有一种 pbds 库里的 priority queue, 提供了更多堆的实现, 感兴趣的同学可以下课自行查找资料学习.

Luogu P1631 序列合并

给定长度为 n 的**单调不降**的两个序列 a 和 b, 每两项相加产生 n^2 个和, 求这 n^2 个和的前 n 小值.

Restrictions: $1 \leq n \leq 10^5$.

Luogu P1631 序列合并

给定长度为 n 的**单调不降**的两个序列 a 和 b, 每两项相加产生 n^2 个和, 求这 n^2 个和的前 n 小值.

Restrictions: $1 \le n \le 10^5$.

维护 n 个有序序列, 第 i 个序列 $\{c_i\}$ 的第 j 项为 $c_{ij} = a_i + b_j$. 则答案产生在这 n 个序列的前缀中. 考虑将所有序列的第一项加入一个小根堆中, 每次弹出最小值作为答案, 并将最小值所来自的序列的下一项加入小根堆中, 做 n 次即可. 代码实现上可以使用 priority queue 和 pair. 时间复杂度: $O(n \log n)$.

对顶堆——动态维护全局 k 小 (Luogu P1801 黑匣子)

对于动态维护 k 小问题, 平衡树自可以解决, 但对于一种特殊情况, 我们可以使用对顶堆来解决.

维护一个初始为空的集合, 包含两种操作: 给集合里插入一个数; 查询集合里的 k 小值, k 为当前查询操作的次数.

维护一个小根堆和一个大根堆, 其中大根堆维护 $k-1$ 小元素, 小根堆维护剩下的 $n-k+1$ 个元素. 插入一个数, 先将其插入到大根堆, 然后将大根堆的根插入到小根堆以保证元素个数正确, 则对于一个查询, 小根堆的根既是答案.

树状数组

树状数组

树状数组同样是一棵二叉树, 通常用以维护序列上简单的带结合律的数据. 具有代码量小, 常数小等优点. 经常作为高级数据结构的基石出现在题目中, 或作为辅助工具出现在其他类型的题目中, 是最重要的数据结构之一.

树状数组

树状数组同样是一棵二叉树, 通常用以维护序列上简单的带结合律的数据. 具有代码量小, 常数小等优点. 经常作为高级数据结构的基石出现在题目中, 或作为辅助工具出现在其他类型的题目中, 是最重要的数据结构之一.

我们以单点修改, 区间求和为例来看树状数组如何工作. 基于前缀和的思想, 如果我们能够维护每一项前缀和, 则区间和可以 $O(1)$ 计算. 也就是说, 我们现在主要的任务是维护前缀和.

树状数组

树状数组同样是一棵二叉树, 通常用以维护序列上简单的带结合律的数据. 具有代码量小, 常数小等优点. 经常作为高级数据结构的基石出现在题目中, 或作为辅助工具出现在其他类型的题目中, 是最重要的数据结构之一.

我们以单点修改, 区间求和为例来看树状数组如何工作. 基于前缀和的思想, 如果我们能够维护每一项前缀和, 则区间和可以 $O(1)$ 计算. 也就是说, 我们现在主要的任务是维护前缀和.

设树状数组为 bit_n, 则每一个元素都维护一段区间的信息, 假设 bit_x 维护的区间为 $[l, r]$, 则树状数组规定, 设: k 表示 x 的二进制表示里最低位的 1 的位数 (例如 $(1100)_2$ 的最低位的 1 的位数即为 2), 则

1. bit_x 维护的区间长度为 2^k.
2. bit_x 维护的区间右端点为 x.

由上两条规定, 我们可以得到: **树状数组上的元素 bit_x 维护的区间为** $[x - 2^k + 1, x]$, 其中 k 的定义如上所示, 通常将 2^k 记作 $\mathrm{lowbit}(x)$.

计算 $\text{lowbit}(x)$

结论: $\text{lowbit}(x) = x \ \& \ -x$.

计算 $\mathrm{lowbit}(x)$

结论: $\mathrm{lowbit}(x) = x \ \& \ -x$.

前置知识: $-x$ 的二进制编码为 x 取反后 $+1$.

证明: 设 x 的二进制编码为 $(\ldots 10 \ldots 00)_2$, 则取反变为 $(\ldots 01 \ldots 11)_2$, 加 1 后变为 $(\ldots 10 \ldots 00)$, 即为 $-x$ 的二进制编码. 两者相与, 由于前面的省略号代表的数全部相反, 相与为 0, 故运算结果为 $\mathrm{lowbit}(x)$.

区间查询

我们现在要查询 $[l, r]$ 的区间和, 用前缀和的思想, 我们转而求 $[1, r]$ 的和与 $[1, l-1]$ 的和.

区间查询

我们现在要查询 $[l, r]$ 的区间和, 用前缀和的思想, 我们转而求 $[1, r]$ 的和与 $[1, l-1]$ 的和.

一般来看, 假设我们所求的前缀和为 $[1, x]$, 我们希望将 $[1, x]$ 分解成若干个树状数组上的区间. 又因为右端点为 x 的区间已经确定, 故我们从 $\mathrm{bit}(x)$ 开始, 为了不遗漏且不重复, 我们下一个区间的右端点应为上一个区间的左端点的左侧, 以 x 为例, 下一个区间的右端点应为 $x - \mathrm{lowbit}(x)$. 逐步跳跃直到 $x = 0$ 停止.

区间查询

我们现在要查询 $[l, r]$ 的区间和, 用前缀和的思想, 我们转而求 $[1, r]$ 的和与 $[1, l-1]$ 的和.

一般来看, 假设我们所求的前缀和为 $[1, x]$, 我们希望将 $[1, x]$ 分解成若干个树状数组上的区间. 又因为右端点为 x 的区间已经确定, 故我们从 $\mathrm{bit}(x)$ 开始, 为了不遗漏且不重复, 我们下一个区间的右端点应为上一个区间的左端点的左侧, 以 x 为例, 下一个区间的右端点应为 $x - \mathrm{lowbit}(x)$. 逐步跳跃直到 $x = 0$ 停止.

容易发现, 我们跳跃的次数即为 $\mathrm{popcount}(x)$, 故时间复杂度为 $O(\log n)$.

树状数组的性质

我们刚刚考虑了树状数组的数组形态, 现在我们来考虑树状数组的树形态. 事实上, 树状数组即为 x 与 $x+\mathrm{lowbit}(x)$ 连边所形成的树. 若我们记 L_x 为 bit_x 的左端点, 即 $L_x = x - \mathrm{lowbit}(x) + 1$, 记 I_x 为 bit_x 所对应的区间, 即 $I_x = [L_x, x]$. 则树状数组有如下优秀的性质:

- 对 $x \le y$, 则 $I_x \subseteq I_y$ 或 $I_x \cap I_y = \varnothing$.

树状数组的性质

我们刚刚考虑了树状数组的数组形态, 现在我们来考虑树状数组的树形态. 事实上, 树状数组即为 x 与 $x+\mathrm{lowbit}(x)$ 连边所形成的树. 若我们记 L_x 为 bit_x 的左端点, 即 $L_x = x - \mathrm{lowbit}(x) + 1$, 记 I_x 为 bit_x 所对应的区间, 即 $I_x = [L_x, x]$. 则树状数组有如下优秀的性质:

- 对 $x \le y$, 则 $I_x \subseteq I_y$ 或 $I_x \cap I_y = \varnothing$.
- $I_x \subset I_{x+\mathrm{lowbit}(x)}$.

树状数组的性质

我们刚刚考虑了树状数组的数组形态, 现在我们来考虑树状数组的树形态. 事实上, 树状数组即为 x 与 $x+\mathrm{lowbit}(x)$ 连边所形成的树. 若我们记 L_x 为 bit_x 的左端点, 即 $L_x = x - \mathrm{lowbit}(x) + 1$, 记 I_x 为 bit_x 所对应的区间, 即 $I_x = [L_x, x]$. 则树状数组有如下优秀的性质:

- 对 $x \le y$, 则 $I_x \subseteq I_y$ 或 $I_x \cap I_y = \varnothing$.
- $I_x \subset I_{x+\mathrm{lowbit}(x)}$.
- $\forall y \in (x, x+\mathrm{lowbit}(x)), I_x \cap I_y = \varnothing$.

令 f_u 表示树上结点 u 的父亲, 并将这三条性质在树上考虑, 我们得到:

- $I_u \subset I_{f_u}$;
- $I_u \subset I_v$, 其中 v 是 u 的任一祖先;
- 对于任意的 $v > u$, 若 v 不是 u 的祖先, 则 $I_u \cap I_v = \varnothing$.

树状数组的性质

我们刚刚考虑了树状数组的数组形态, 现在我们来考虑树状数组的树形态. 事实上, 树状数组即为 x 与 $x+\mathrm{lowbit}(x)$ 连边所形成的树. 若我们记 L_x 为 bit_x 的左端点, 即 $L_x = x - \mathrm{lowbit}(x) + 1$, 记 I_x 为 bit_x 所对应的区间, 即 $I_x = [L_x, x]$. 则树状数组有如下优秀的性质:

- 对 $x \le y$, 则 $I_x \subseteq I_y$ 或 $I_x \cap I_y = \varnothing$.
- $I_x \subset I_{x+\mathrm{lowbit}(x)}$.
- $\forall y \in (x, x+\mathrm{lowbit}(x)), I_x \cap I_y = \varnothing$.

令 f_u 表示树上结点 u 的父亲, 并将这三条性质在树上考虑, 我们得到:

- $I_u \subset I_{f_u}$;
- $I_u \subset I_v$, 其中 v 是 u 的任一祖先;
- 对于任意的 $v > u$, 若 v 不是 u 的祖先, 则 $I_u \cap I_v = \varnothing$.

于是我们可以归纳出: **对任意的 $v > u$, 当且仅当 v 是 u 的祖先, $I_u \subset I_v$.**

单点修改 & 建树

为了实现单点修改, 我们需要快速的找到包含该点的所有区间, 而区间包含点 x 的结点必然在树状数组上为 x 的祖先, 故我们只需要从 x 开始逐步跳父亲, 直到超过 n 即可.

由于点 x 的高度为 $\log \operatorname{lowbit}(x)$, 故单点修改的复杂度为 $O(\log n)$.

单点修改 & 建树

为了实现单点修改, 我们需要快速的找到包含该点的所有区间, 而区间包含点 x 的结点必然在树状数组上为 x 的祖先, 故我们只需要从 x 开始逐步跳父亲, 直到超过 n 即可.

由于点 x 的高度为 $\log \operatorname{lowbit}(x)$, 故单点修改的复杂度为 $O(\log n)$.

关于建树, 朴素的想法是做 n 次单点修改, 这样的总复杂度依然是 $O(n \log n)$. 但为了使代码更高效, 此处介绍两种 $O(n)$ 建树的方法:

- 由于 u 的所有儿子结点的编号均 $< u$. 于是我们可以直接考虑用儿子考虑父亲结点的值 (本质上像 zkw 线段树的建树).
- 由于 $I_u = [u - \operatorname{lowbit}(u) + 1, u]$, 故可以维护一个前缀和数组去建树.

区间修改

同线段树一样, 树状数组也可以解决区间修改问题. 对区间修改单点查询, 只需要做一次差分即可. 对区间修改区间查询, 需要维护两个树状数组, 计算差分的贡献, 相对比较复杂.

一般来说, 对于区间修改问题, 使用线段树解决较为简单, 就算考虑到常数, 我们也可以使用 zkw 线段树, 故包含区间修改的树状数组问题仅作了解即可.

Luogu P1908 逆序对

给定长度为 n 的序列 a, 求序列中的逆序对数. 若 (i,j) 满足 $i<j$ 且 $a_i>a_j$, 则称 (i,j) 为一个逆序对. 形式化地, 题目要求

$$\sum_{i=1}^{n}\sum_{j=1}^{i-1}[a_i<a_j]$$

Restrictions: $1\le n\le 5\times 10^5$, $1\le a_i\le 10^9$.

Luogu P1908 逆序对

给定长度为 n 的序列 a, 求序列中的逆序对数. 若 (i, j) 满足 $i < j$ 且 $a_i > a_j$, 则称 (i, j) 为一个逆序对. 形式化地, 题目要求

$$\sum_{i=1}^{n} \sum_{j=1}^{i-1} [a_i < a_j]$$

Restrictions: $1 \le n \le 5 \times 10^5$, $1 \le a_i \le 10^9$.

维护一个**权值**树状数组, 区间 $[v_1, v_2]$ 表示 $v_1 \sim v_2$ 的值域区间上有多少数. 将序列 a 从 n 到 1 逐步插入树状数组中, 每次查询 $[1, a_i]$ 的区间和即可.

实现细节: 序列离散化, 注意答案的数据范围.

离线区间数颜色/Luogu P1972 [SDOI2009] HH的项链

给定长度为 n 的数列 $\{a_n\}$, m 次询问, 每次询问包含两个参数 l, r, 询问 $a_l \sim a_r$ 中有多少种不同的数.

Restrictions: $1 \leq n, m, a_i \leq 10^6$.

离线区间数颜色/Luogu P1972 [SDOI2009] HH的项链

给定长度为 n 的数列 $\{a_n\}$, m 次询问, 每次询问包含两个参数 l, r, 询问 $a_l \sim a_r$ 中有多少种不同的数.

Restrictions: $1 \leq n, m, a_i \leq 10^6$.

离线处理询问. 具体来说, 在此题中, 如果询问区间右端点 r 固定, 那么一个颜色的贡献仅由其最右边的存在产生.

于是我们存储所有询问并按右端点排序, 记录每个颜色出现的最晚位置并将这个位置的权值设为 1, 将这个颜色上一个位置出现的权值消去 (这需要我们记录每个颜色的前置位置). 对于一个询问 l, r, 我们只需要查询 l, r 的区间和即可. 这个操作可以通过很多方式实现, 上节课学的线段树, 本节课学的树状数组均可以.

需要注意虽然我们对值域/颜色考虑问题, 但我们的树状数组依然是对序列建的.

CF369E Valera and Queries

给定 n 条线段, 第 i 条线段的端点为 $[l_i, r_i]$, 询问 q 次, 每次询问包含如下参数: $(c; p_1, p_2, \ldots, p_c)$, 查询有多少线段至少覆盖 c 个点中的一个.

Restrictions: $1 \le n, q \le 3 \times 10^5$, $1 \le l_i \le r_i \le 10^6$, $1 \le \sum c \le 3 \times 10^5$.

[2]此处规定 $p_0 = 0, p_{c+1} = \max r_i$

CF369E Valera and Queries

给定 n 条线段, 第 i 条线段的端点为 $[l_i, r_i]$, 询问 q 次, 每次询问包含如下参数: $(c; p_1, p_2, \ldots, p_c)$, 查询有多少线段至少覆盖 c 个点中的一个.

Restrictions: $1 \le n, q \le 3 \times 10^5$, $1 \le l_i \le r_i \le 10^6$, $1 \le \sum c \le 3 \times 10^5$.

考虑反向回答询问: 计算有多少条线段没有包含点. 对每次询问, $\{p_c\}$ 将整条线段分成 $c+1$ 个小线段. 则每一次询问就是查询有多少 $[l_i, r_i] \subseteq [p_k, p_{k+1}]$[2].

[2]此处规定 $p_0 = 0, p_{c+1} = \max r_i$

CF369E Valera and Queries

给定 n 条线段, 第 i 条线段的端点为 $[l_i, r_i]$, 询问 q 次, 每次询问包含如下参数: $(c; p_1, p_2, \ldots, p_c)$, 查询有多少线段至少覆盖 c 个点中的一个.

Restrictions: $1 \le n, q \le 3 \times 10^5$, $1 \le l_i \le r_i \le 10^6$, $1 \le \sum c \le 3 \times 10^5$.

考虑反向回答询问: 计算有多少条线段没有包含点. 对每次询问, $\{p_c\}$ 将整条线段分成 $c+1$ 个小线段. 则每一次询问就是查询有多少 $[l_i, r_i] \subseteq [p_k, p_{k+1}]$[2].

离线处理询问, 按照 l 倒序处理. 遇到一个线段就将 r 端点加入树状数组. 遇到询问 (被点分成的子区间) 就查询右端点的前缀和. 因为我们是按 l 倒序处理的, 所以前缀和查询到的区间一定满足 $[l_i, r_i] \subseteq [p_k, p_{k+1}]$, 这实际上类似于偏序问题. 树状数组解决即可. 复杂度为 $O\left(\sum c + n \log |V|\right)$.

[2]此处规定 $p_0 = 0, p_{c+1} = \max r_i$

可持久化数据结构

什么是可持久化数据结构

可持久化数据结构 (Persistent Data Structure), 是一种保留其所有历史版本的数据结构, 并且操作不可变 (Immutable). 以可持久化的类型分类, 可分为如下几种:

- 部分可持久化 (Partially Persistent): 所有版本都可以访问, 但只有最新的版本可以做修改.
- 完全可持久化 (Fully Persistent): 所有版本都可以进行访问及修改.
- 汇集可持久化 (Confluently Persistent): 在 Fully Persistent 的基础上, 支持两个版本合并为一个新的版本.
- 函数式可持久化 (Functional Persistent): 无法修改结点, 只产生新的结点.
- 瞬息数据结构 (Ephemeral Data Structure): 非可持久化的数据结构.

什么是可持久化数据结构

可持久化数据结构 (Persistent Data Structure), 是一种保留其所有历史版本的数据结构, 并且操作不可变 (Immutable). 以可持久化的类型分类, 可分为如下几种:

- 部分可持久化 (Partially Persistent): 所有版本都可以访问, 但只有最新的版本可以做修改.
- 完全可持久化 (Fully Persistent): 所有版本都可以进行访问及修改.
- 汇集可持久化 (Confluently Persistent): 在 Fully Persistent 的基础上, 支持两个版本合并为一个新的版本.
- 函数式可持久化 (Functional Persistent): 无法修改结点, 只产生新的结点.
- 瞬息数据结构 (Ephemeral Data Structure): 非可持久化的数据结构.

在 OI 中, 可持久化线段树以及可持久化 trie 是最常用的两种可持久化数据结构. 不过需要知道的是, 并查集, 平衡树同样也可以持久化.

Persistent Segment Tree/P3919 [模板] 可持久化线段树 1 (可持久化数组)

在 OI 中的可持久化线段树的核心思想基于每次单点操作新建最多 $\log n$ 个结点的事实, 即每一个版本的可持久化线段树都与其前一个版本有大部分的重叠而仅有少部分的更新. 这让我们得以控制空间复杂度.

Persistent Segment Tree/P3919 [模板] 可持久化线段树 1 (可持久化数组)

在 OI 中的可持久化线段树的核心思想基于每次单点操作新建最多 $\log n$ 个结点的事实, 即每一个版本的可持久化线段树都与其前一个版本有大部分的重叠而仅有少部分的更新. 这让我们得以控制空间复杂度.

让我们直接以洛谷模板为例: 维护一个长度为 n 的数组 $\{a_n\}$. 共进行 m 次操作, 分为如下两种:

- 基于某个历史版本单点修改;
- 单点询问某个历史版本的值.

注意, 每次操作 (包括询问) 都会生成一个新的版本.

Restrictions: $1 \leq n, m \leq 10^6$.

Persistent Segment Tree/P3919 [模板] 可持久化线段树 1 (可持久化数组)

在 OI 中的可持久化线段树的核心思想基于每次单点操作新建最多 $\log n$ 个结点的事实, 即每一个版本的可持久化线段树都与其前一个版本有大部分的重叠而仅有少部分的更新. 这让我们得以控制空间复杂度.

让我们直接以洛谷模板为例: 维护一个长度为 n 的数组 $\{a_n\}$. 共进行 m 次操作, 分为如下两种:

- 基于某个历史版本单点修改;
- 单点询问某个历史版本的值.

注意, 每次操作 (包括询问) 都会生成一个新的版本.

Restrictions: $1 \leq n, m \leq 10^6$.

对数组建立可持久化线段树, 动态开点 (所有的可持久化线段树都需要动态开点). 维护每个版本的根, 注意到一次操作最多修改 $(\log n)$ 级别的结点. 故空间复杂度与时间复杂度都是 $O(m \log n)$.

P3834 [模板] 可持久化线段树 2 (静态区间查询 k 小)

给定长度为 n 的数组 $\{a_n\}$, q 次查询, 每次查询包含三个参数 l, r, k, 查询 $a_l \sim a_r$ 中第 k 小值.

Restrictions: $1 \leq n, m \leq 2 \times 10^5, 0 \leq a_i \leq 10^9$.

P3834 [模板] 可持久化线段树 2 (静态区间查询 k 小)

给定长度为 n 的数组 $\{a_n\}$, q 次查询, 每次查询包含三个参数 l, r, k, 查询 $a_l \sim a_r$ 中第 k 小值.

Restrictions: $1 \leq n, m \leq 2 \times 10^5, 0 \leq a_i \leq 10^9$.

建立可持久化权值线段树, 发现 $a_l \sim a_r$ 组成的权值线段树即为 $a_1 \sim a_r$ 组成的权值线段树减去 $a_1 \sim a_{l-1}$ 的权值线段树. 于是维护每个版本的权值线段树, 询问则将权值线段树相减, 然后转为全局求 k 小问题. 上节课已经讲过. 时间复杂度: $O((n+q)\log n)$.

P3834 [模板] 可持久化线段树 2 (静态区间查询 k 小)

给定长度为 n 的数组 $\{a_n\}$, q 次查询, 每次查询包含三个参数 l, r, k, 查询 $a_l \sim a_r$ 中第 k 小值.

Restrictions: $1 \le n, m \le 2 \times 10^5, 0 \le a_i \le 10^9$.

建立可持久化权值线段树, 发现 $a_l \sim a_r$ 组成的权值线段树即为 $a_1 \sim a_r$ 组成的权值线段树减去 $a_1 \sim a_{l-1}$ 的权值线段树. 于是维护每个版本的权值线段树, 询问则将权值线段树相减, 然后转为全局求 k 小问题. 上节课已经讲过. 时间复杂度: $O((n+q)\log n)$.

实现细节: 值域较大, 需要离散化. 不需要实际上计算出一棵新线段树, 只需要在沿路路径上计算对应的值即可.

Persistent 01-Trie/Luogu P4735 最大异或和

在 OI 中另一个比较常用的可持久化数据结构即为可持久化 01-trie. 其基本思想和可持久化线段树并无二致, 都是使用新建结点的方式进行版本更新. 不同之处在于树高, 可持久化 01-trie 的树高是 $\log V$.

Persistent 01-Trie/Luogu P4735 最大异或和

在 OI 中另一个比较常用的可持久化数据结构即为可持久化 01-trie. 其基本思想和可持久化线段树并无二致, 都是使用新建结点的方式进行版本更新. 不同之处在于树高, 可持久化 01-trie 的树高是 $\log V$.

接下来看可持久化 01-trie 模板题: 给定序列 a, 长度为 n, m 次操作, 分为如下两种:

- 在序列的末尾添加一个数;
- 给定 l, r, x, 查询位置 p 满足 x 与 $\bigoplus_{i=p}^{n} a_i$ 异或值最大, 求这个最大异或和.

Restrictions: $1 \le n, m \le 3 \times 10^5, 0 \le a_i \le 10^7$.

Persistent 01-Trie/Luogu P4735 最大异或和

在 OI 中另一个比较常用的可持久化数据结构即为可持久化 01-trie. 其基本思想和可持久化线段树并无二致, 都是使用新建结点的方式进行版本更新. 不同之处在于树高, 可持久化 01-trie 的树高是 $\log V$.

接下来看可持久化 01-trie 模板题: 给定序列 a, 长度为 n, m 次操作, 分为如下两种:

- 在序列的末尾添加一个数;
- 给定 l, r, x, 查询位置 p 满足 x 与 $\bigoplus_{i=p}^{n} a_i$ 异或值最大, 求这个最大异或和.

Restrictions: $1 \le n, m \le 3 \times 10^5, 0 \le a_i \le 10^7$.

设前缀异或和为 op_i, 则问题转化为求 p 使得 $\mathrm{op}_{p-1} \oplus \mathrm{op}_n \oplus x$ 最大. 注意到后两项在查询时是常数, 如果是全局询问明显很好做, 按路径贪心的走不一样的一位即可. 放到区间上也一样, 只需要建可持久化 01-trie, 对区间 $[l, r]$ 的询问, 查询 $[l-1, r-1]$ 区间的 01-trie 与 $\mathrm{op}_n \oplus x$ 的异或最大值即可. 实现上可以对每个结点记录其子树里的数的序列位置最大值 (记作 last_u), 这样只需要每次在 $r-1$ 的版本上查询所有结点均满足 $\mathrm{last}_u \ge l-1$ 的异或最大值.

P4587 [FJOI2016] 神秘数

给定长度为 n 的序列 a, m 次查询, 每次查询包含两个参数 l, r, 查询最小的不能由 $a_l \sim a_r$ 中的数之和表示出来的数. 例如 $1, 2, 5$, 则最小不能被表示出的数为 4.

Restrictions: $1 \leq n, m \leq 10^5, 0 \leq \sum a \leq 10^9$.

P4587 [FJOI2016] 神秘数

给定长度为 n 的序列 a, m 次查询, 每次查询包含两个参数 l, r, 查询最小的不能由 $a_l \sim a_r$ 中的数之和表示出来的数. 例如 $1, 2, 5$, 则最小不能被表示出的数为 4.

Restrictions: $1 \leq n, m \leq 10^5, 0 \leq \sum a \leq 10^9$.

发现这个询问有一个性质: 对一个**升序**的序列 $\{b_k\}$, 将 $b_1 \sim b_k$ 逐步插入集合. 若在某一瞬间 $[1, x]$ 都能被表示出来, 若待加入的数 z 满足 $z \leq x + 1$ 时, 则 $[1, z + x]$ 都能被表示出来. 否则, 则说明 $x + 1$ 不能被表示出来. 所以我们有一个很暴力的想法, 即对每个询问排序, 然后逐渐将数插入到集合中即可. 这样的复杂度是 $O(nm \log n)$.

P4587 [FJOI2016] 神秘数

给定长度为 n 的序列 a, m 次查询, 每次查询包含两个参数 l, r, 查询最小的不能由 $a_l \sim a_r$ 中的数之和表示出来的数. 例如 $1, 2, 5$, 则最小不能被表示出的数为 4.

Restrictions: $1 \le n, m \le 10^5, 0 \le \sum a \le 10^9$.

发现这个询问有一个性质: 对一个**升序**的序列 $\{b_k\}$, 将 $b_1 \sim b_k$ 逐步插入集合. 若在某一瞬间 $[1, x]$ 都能被表示出来, 若待加入的数 z 满足 $z \le x + 1$ 时, 则 $[1, z + x]$ 都能被表示出来. 否则, 则说明 $x + 1$ 不能被表示出来. 所以我们有一个很暴力的想法, 即对每个询问排序, 然后逐渐将数插入到集合中即可. 这样的复杂度是 $O(nm \log n)$.

考虑优化: 当可以表示出的区间为 $[1, x]$ 时, 在值域上查询 $[1, x]$ 的区间和, 记为 S, 若 $S \ge x$, 则说明在 $[1, x]$ 这个区间内还有小于等于 x 且并没有被插入到集合中的数. 于是将 x 设为 S, 继续查询, 直至没有这样的数为止. 容易证明, 如果答案可以更新, 则区间至少扩大一倍. 故复杂度为 $O\left(m \log n \log\left(\sum a_i\right)\right)$.

Luogu P2617 Dynamic Rankings

单点修改区间 k 小问题. $1 \leq n, m \leq 10^5$.

Luogu P2617 Dynamic Rankings

单点修改区间 k 小问题. $1 \leq n, m \leq 10^5$.

发现静态可持久化线段树实际上是前缀和相减, 所以我们需要一个能够动态维护前缀和的数据结构来维护可持久化线段树, 即树状数组. 故本题对可持久化线段树建树状数组, 修改的时候需要修改一整条树状数组上路径的可持久化线段树, 查询时也要查询树状数组上一条路径上所有的可持久化线段树. 故复杂度多一个 $\log$, 为 $O(m \log^2 n)$.