可持久化数据结构

RainAir

Tsinghua University

July 15, 2023

前言

- 简单自我介绍。

前言

- 简单自我介绍。
- 为了活跃课堂气氛，回答一些问题会有奖励。

前言

- 简单自我介绍。
- 为了活跃课堂气氛，回答一些问题会有奖励。
- 默认大家拥有基础的（动态开点）线段树和字典树知识。

前言

- 简单自我介绍。
- 为了活跃课堂气氛，回答一些问题会有奖励。
- 默认大家拥有基础的（动态开点）线段树和字典树知识。
- 课前调查：决定是否讲代码。

可持久化数据结构

- 我们已经知道数据结构是用来维护一些数据的，而维护的过程中一般伴随着对数据的修改。
- 在有些时候，我们不仅需要获得当前的数据情况，还需要能获得历史的数据情况。
- 支持访问历史数据状态的数据结构可被称为可持久化数据结构。
- 今天我们会讲一些简单的可持久化数据结构和相应的例题。

引入问题：区间第 k 小

- 给定 n 个整数构成的序列 a，进行多次询问，每次询问给出闭区间 $[l, r]$ 要求查询其中的第 k 小值。强制在线。

引入问题：区间第 k 小

- 给定 n 个整数构成的序列 a，进行多次询问，每次询问给出闭区间 $[l, r]$ 要求查询其中的第 k 小值。强制在线。
- 如果是全局第 k 小，~~当然可以直接排序~~，可以建立一棵权值线段树，然后在线段树上二分来得到第 k 小。

引入问题：区间第 k 小

- 给定 n 个整数构成的序列 a，进行多次询问，每次询问给出闭区间 $[l, r]$ 要求查询其中的第 k 小值。强制在线。
- 如果是全局第 k 小，~~当然可以直接排序~~，可以建立一棵权值线段树，然后在线段树上二分来得到第 k 小。
- 在线段树上二分时，我们实际上是进行若干次询问，每次询问形如「查询 $\leq x$ 的数字的个数」。

可持久化线段树 I

- 如果是区间第 k 小呢？我们可以在只插入指定区间内的数的线段树上二分。

可持久化线段树 I

- 如果是区间第 k 小呢？我们可以在只插入指定区间内的数的线段树上二分。
- 而由于权值线段树维护的是「某个数字出现的次数」，这是可减信息，所以我们类比前缀和，只需要维护 $n+1$ 棵线段树 $T_0, T_1, \ldots, T_n$，其中 T_k 表示插入 $a_1, \ldots, a_k$ 后得到的权值线段树。

可持久化线段树 I

- 如果是区间第 k 小呢？我们可以在只插入指定区间内的数的线段树上二分。
- 而由于权值线段树维护的是「某个数字出现的次数」，这是可减信息，所以我们类比前缀和，只需要维护 $n+1$ 棵线段树 $T_0, T_1, \ldots, T_n$，其中 T_k 表示插入 $a_1, \ldots, a_k$ 后得到的权值线段树。
- 我们如果想要询问区间 $[l, r]$，只需要找出 T_{l-1}, T_r，然后使用线段树二分：此时如果想要查询 $\leq x$ 的数字的个数，只需要在 T_r, T_{l-1} 上做查询分别得到结果 $Q(T_r, x), Q(T_{l-1}, x)$，那么在区间 $[l, r]$ 内 $\leq x$ 的数字的个数就是 $Q(T_r, x) - Q(T_{l-1}, x)$。

可持久化线段树 II

- 如果你真的建 $n+1$ 棵线段树，不仅时间复杂度爆炸，空间也存不下。

可持久化线段树 II

- 如果你真的建 $n+1$ 棵线段树，不仅时间复杂度爆炸，空间也存不下。
- 我们考虑在建立 T_k 时，能否充分利用 T_{k-1} 的信息。

可持久化线段树 II

- 如果你真的建 $n+1$ 棵线段树，不仅时间复杂度爆炸，空间也存不下。
- 我们考虑在建立 T_k 时，能否充分利用 T_{k-1} 的信息。
- 注意 T_k 是在 T_{k-1} 的基础上经过 单点修改而得到的，而这只会改变 $O(\log n)$ 个线段树上的节点的值。

可持久化线段树 II

- 如果你真的建 $n+1$ 棵线段树，不仅时间复杂度爆炸，空间也存不下。
- 我们考虑在建立 T_k 时，能否充分利用 T_{k-1} 的信息。
- 注意 T_k 是在 T_{k-1} 的基础上经过 单点修改而得到的，而这只会改变 $O(\log n)$ 个线段树上的节点的值。
- 也就是说，大部分 T_{k-1} 的节点是完全不会被影响的！

可持久化线段树 II

- 如果你真的建 $n+1$ 棵线段树，不仅时间复杂度爆炸，空间也存不下。
- 我们考虑在建立 T_k 时，能否充分利用 T_{k-1} 的信息。
- 注意 T_k 是在 T_{k-1} 的基础上经过 单点修改而得到的，而这只会改变 $O(\log n)$ 个线段树上的节点的值。
- 也就是说，大部分 T_{k-1} 的节点是完全不会被影响的！
- 所以我们只需要在 T_{k-1} 的基础上，新建 $O(\log n)$ 个被影响的节点，然后就可以得到 T_k 了！

可持久化线段树 II

- 如果你真的建 $n+1$ 棵线段树，不仅时间复杂度爆炸，空间也存不下。
- 我们考虑在建立 T_k 时，能否充分利用 T_{k-1} 的信息。
- 注意 T_k 是在 T_{k-1} 的基础上经过 单点修改而得到的，而这只会改变 $O(\log n)$ 个线段树上的节点的值。
- 也就是说，大部分 T_{k-1} 的节点是完全不会被影响的！
- 所以我们只需要在 T_{k-1} 的基础上，新建 $O(\log n)$ 个被影响的节点，然后就可以得到 T_k 了！
- 这里应该有代码演示。

可持久化线段树 II

- 如果你真的建 $n+1$ 棵线段树，不仅时间复杂度爆炸，空间也存不下。
- 我们考虑在建立 T_k 时，能否充分利用 T_{k-1} 的信息。
- 注意 T_k 是在 T_{k-1} 的基础上经过 单点修改而得到的，而这只会改变 $O(\log n)$ 个线段树上的节点的值。
- 也就是说，大部分 T_{k-1} 的节点是完全不会被影响的！
- 所以我们只需要在 T_{k-1} 的基础上，新建 $O(\log n)$ 个被影响的节点，然后就可以得到 T_k 了！
- 这里应该有代码演示。
- 常见错误：空间开小了。

例 0: 树上两点间第 k 小

- 顾名思义。

例 0: 树上两点间第 k 小

- 顾名思义。
- 类似区间前缀和变成树上前缀和，每个点从根继承过来线段树然后单点修改。

例 0: 树上两点间第 k 小

- 顾名思义。
- 类似区间前缀和变成树上前缀和，每个点从根继承过来线段树然后单点修改。
- 查询找到 lca，用 $x + y - 2lca$ 这三个线段树的值来决定二分方向。

例 1: 二维数点

- 现在平面上有 n 个点，第 i 个点的坐标为 (x_i, y_i)。现在进行 q 次 在线的询问，每次询问平面上某个矩形中的点的个数。
- 要求复杂度单 log。

例 1

- 如果可以离线，那么只需要一个树状数组就好了。

例 1

- 如果可以离线，那么只需要一个树状数组就好了。
- 我们通过离散化可以假设 $1 \leq x_i, y_i \leq n$。这样我们建立 n 棵线段树，第 i 棵线段树 T_i 存储所有满足 $x_k \leq i$ 的点的 y_i 的权值信息。

例 1

- 如果可以离线，那么只需要一个树状数组就好了。
- 我们通过离散化可以假设 $1 \leq x_i, y_i \leq n$。这样我们建立 n 棵线段树，第 i 棵线段树 T_i 存储所有满足 $x_k \leq i$ 的点的 y_i 的权值信息。
- 每次询问假设要找子矩阵 $[l_1, r_1] \times [l_2, r_2]$，只需要在 T_{l_1-1}, T_{r_1} 分别做一下在区间 $[l_2, r_2]$ 上的区间询问，然后减一下即可。

例 2

- 最初输入一个长度为 n 的数组 a_i，然后有 m 次操作，每次操作有 3 个参数 l, r, v，表示将区间 $[l, r]$ 内的数统一加上 v
- 之后 q 次询问，每次询问如果只考虑在区间 $[L, R]$ 内的操作，区间 $[l, r]$ 内 a 的元素的和是多少。
- 复杂度要求单 log。

例 2

- 如果改为 m 次单点修改操作大家肯定都会：直接可持久化线段树。

例 2

- 如果改为 m 次单点修改操作大家肯定都会：直接可持久化线段树。
- 线段树处理区间加操作一般的方法是使用 lazytag，但是 lazytag 会有 pushdown 的问题，这破坏了单次只修改 $O(\log n)$ 的限制，导致复杂度退化。

例 2

- 我们使用叫做「标记永久化」的 trick 来解决这个问题。

例 2

- 我们使用叫做「标记永久化」的 trick 来解决这个问题。
- 顾名思义，标记永久化是尝试将标记直接打在对应的区间节点而不需要下放的一种方式。我们在到达这个点的时候，直接根据标记对答案进行一些操作而不是下放标记。

例 2

- 我们使用叫做「标记永久化」的 trick 来解决这个问题。
- 顾名思义，标记永久化是尝试将标记直接打在对应的区间节点而不需要下放的一种方式。我们在到达这个点的时候，直接根据标记对答案进行一些操作而不是下放标记。
- 在这里我们就在对应的区间节点打上 $+val$ 标记，查询的时候将路径上经过的所有标记乘上区间长度加起来就好了。

例 2

- 我们使用叫做「标记永久化」的 trick 来解决这个问题。
- 顾名思义，标记永久化是尝试将标记直接打在对应的区间节点而不需要下放的一种方式。我们在到达这个点的时候，直接根据标记对答案进行一些操作而不是下放标记。
- 在这里我们就在对应的区间节点打上 $+val$ 标记，查询的时候将路径上经过的所有标记乘上区间长度加起来就好了。
- 由于任何一个区间在线段树上定位出的节点是 $O(\log n)$ 的，从而复杂度还是一个 log 的。

例 3: Dynamic Rankings

- 在区间第 k 小这个题目中加入单点修改，强制在线。
- 要求两个 log。

例 3

- 之前做的是静态第 k 小，本质是将区间求和转化为前缀和的相减。

例 3

- 之前做的是静态第 k 小，本质是将区间求和转化为前缀和的相减。
- 现在可以修改了，回忆有什么数据结构可以支持动态维护前缀和？

例 3

- 之前做的是静态第 k 小，本质是将区间求和转化为前缀和的相减。
- 现在可以修改了，回忆有什么数据结构可以支持动态维护前缀和？
- 树状数组！

例 3

- 之前做的是静态第 k 小，本质是将区间求和转化为前缀和的相减。
- 现在可以修改了，回忆有什么数据结构可以支持动态维护前缀和？
- 树状数组！
- 我们设树状数组中下标 i 所记录的信息的下标集合为 S_i，那么我们定义线段树 T_i 是维护者 S_i 内信息的所有的点的集合。

例 3

- 之前做的是静态第 k 小，本质是将区间求和转化为前缀和的相减。
- 现在可以修改了，回忆有什么数据结构可以支持动态维护前缀和？
- 树状数组！
- 我们设树状数组中下标 i 所记录的信息的下标集合为 S_i，那么我们定义线段树 T_i 是维护者 S_i 内信息的所有的点的集合。
- 这样修改和树状数组一样，只需要将 p 不断加上 lowbit，然后再到达的每个线段树都修改，需要修改 $O(\log n)$ 棵线段树。

例 3

- 之前做的是静态第 k 小，本质是将区间求和转化为前缀和的相减。
- 现在可以修改了，回忆有什么数据结构可以支持动态维护前缀和？
- 树状数组！
- 我们设树状数组中下标 i 所记录的信息的下标集合为 S_i，那么我们定义线段树 T_i 是维护者 S_i 内信息的所有的点的集合。
- 这样修改和树状数组一样，只需要将 p 不断加上 lowbit，然后再到达的每个线段树都修改，需要修改 $O(\log n)$ 棵线段树。
- 查询 $[l, r]$ 拆成 $[\leq r] - [\leq l-1]$，分别不断减去 lowbit，得到 $O(\log n)$ 棵线段树。线段树二分时对这些线段树一起二分，每次的值就是将这些线段树的答案加/减起来。

例 3

- 之前做的是静态第 k 小，本质是将区间求和转化为前缀和的相减。
- 现在可以修改了，回忆有什么数据结构可以支持动态维护前缀和？
- 树状数组！
- 我们设树状数组中下标 i 所记录的信息的下标集合为 S_i，那么我们定义线段树 T_i 是维护者 S_i 内信息的所有的点的集合。
- 这样修改和树状数组一样，只需要将 p 不断加上 lowbit，然后再到达的每个线段树都修改，需要修改 $O(\log n)$ 棵线段树。
- 查询 $[l, r]$ 拆成 $[\leq r] - [\leq l-1]$，分别不断减去 lowbit，得到 $O(\log n)$ 棵线段树。线段树二分时对这些线段树一起二分，每次的值就是将这些线段树的答案加/减起来。
- 复杂度两个 log，可以看代码。

例 4: 区间颜色数

- 有一个长度为 n 的序列 a_i。
- 进行 q 次询问，每次询问给出 l, r，询问区间 $[l, r]$ 中有多少个不同的数字。
- 强制在线，要求单 log。

例 4

- 如果想要对于某个区间 $[l, r]$ 只将相同的数字统计一次的话，我们不妨只统计在区间内第一次出现的数的个数。

例 4

- 如果想要对于某个区间 $[l, r]$ 只将相同的数字统计一次的话，我们不妨只统计在区间内第一次出现的数的个数。
- 这个「第一次出现」如何表示？设 pre_i 表示在 a_i 前面且距离最近的相同的位置，那么 a_i 是 $[l, r]$ 中第一次出现的数的个数当且仅当 $pre_i < l$。

例 4

- 如果想要对于某个区间 $[l, r]$ 只将相同的数字统计一次的话，我们不妨只统计在区间内第一次出现的数的个数。
- 这个「第一次出现」如何表示？设 pre_i 表示在 a_i 前面且距离最近的相同的位置，那么 a_i 是 $[l, r]$ 中第一次出现的数的个数当且仅当 $pre_i < l$。
- 所以问题实际上是找 $i \in [l, r]$ 且 $pre_i \in [0, i-1]$ 的点的个数，二维数点！

例 5: Mishka and Interesting sum

- 给出一个长度为 n 的序列 a_i，$n \leq 10^6, a_i \leq 10^9$。
- 定义一个区间 $[l, r]$ 的价值是：只考虑所有出现偶数次的数，将这些数字去重后的异或和。
- q 次询问某个区间的价值。$q \leq 10^6$

例 5

- 我们发现答案等价于：区间内所有出现过的数字的异或和（重复只算一次）异或上区间内所有出现过奇数次数字的异或和（重复只算一次）

例 5

- 我们发现答案等价于：区间内所有出现过的数字的异或和（重复只算一次）异或上区间内所有出现过奇数次数字的异或和（重复只算一次）
- 后一部分就是区间异或和，一个前缀和即可。

例 5

- 我们发现答案等价于：区间内所有出现过的数字的异或和（重复只算一次）异或上区间内所有出现过奇数次数字的异或和（重复只算一次）
- 后一部分就是区间异或和，一个前缀和即可。
- 而为了强制在线，类比上题只统计 $pre_i < l$ 的点之后也是一个二维数点问题，使用主席树即可。

例 6: 宝石

- 给出一个树，每个节点有一个种类为 a_i 的宝石。一开始给出一个长度为 c 的宝石种类的序列 P，保证 P 中元素互不相同。
- 有一个人有一个宝石收集器，这个宝石收集器要求必须先收集种类为 P_1 的宝石，才能收集 P_2 的，...
- 现在处理 q 次询问，每次询问给出两个点 u, v，求这个人从点 u 出发按照树上最短路径走到 v，最多能收集多少个宝石。
- (最好强制在线)
- $n, q \leq 2 \times 10^5$。

例 6

- 这是 21 年省选签到题，先考虑链上怎么做。

例 6

- 这是 21 年省选签到题，先考虑链上怎么做。
- 暴力想法是从头开始找到第一个 p_1 出现的位置，然后找到 p_2 的，...，什么时候离开这个区间就停止，并统计跳的次数。

例 6

- 这是 21 年省选签到题，先考虑链上怎么做。
- 暴力想法是从头开始找到第一个 p_1 出现的位置，然后找到 p_2 的，...，什么时候离开这个区间就停止，并统计跳的次数。
- 然后搬到树上，这样给出 u, v 后还是一条链。设 u, v 的 lca 为 l，这样将这个跳的步骤分成两部分：$u \to l$ 和 $l \to v$。

例 6

- 这是 21 年省选签到题，先考虑链上怎么做。
- 暴力想法是从头开始找到第一个 p_1 出现的位置，然后找到 p_2 的，...，什么时候离开这个区间就停止，并统计跳的次数。
- 然后搬到树上，这样给出 u, v 后还是一条链。设 u, v 的 lca 为 l，这样将这个跳的步骤分成两部分：$u \to l$ 和 $l \to v$。
- 可以用倍增快速实现向上跳的部分，特殊处理 l 后就变成了：从 P 序列的某个位置开始向下跳的问题。

例 6

- 这种问题向下跳都是不好处理的：因为方向不确定。考虑转化成向上跳的问题。现在考虑询问 $l \to v$，且序列从 P_k 开始。

例 6

- 这种问题向下跳都是不好处理的：因为方向不确定。考虑转化成向上跳的问题。现在考虑询问 $l \to v$，且序列从 P_k 开始。
- 我们假设知道答案是 x，如何判断答案是否合法？只要从 v 开始，找到 v 上面距离 v 最近的颜色为 P_x 的点，然后这样依次往上调，看跳到 P_k 后是否还在 l 的子树内就好了。

例 6

- 这种问题向下跳都是不好处理的：因为方向不确定。考虑转化成向上跳的问题。现在考虑询问 $l \to v$，且序列从 P_k 开始。
- 我们假设知道答案是 x，如何判断答案是否合法？只要从 v 开始，找到 v 上面距离 v 最近的颜色为 P_x 的点，然后这样依次往上调，看跳到 P_k 后是否还在 l 的子树内就好了。
- 并且答案显然具有单调性，这启发我们二分答案。

例 6

- 这种问题向下跳都是不好处理的：因为方向不确定。考虑转化成向上跳的问题。现在考虑询问 $l \to v$，且序列从 P_k 开始。
- 我们假设知道答案是 x，如何判断答案是否合法？只要从 v 开始，找到 v 上面距离 v 最近的颜色为 P_x 的点，然后这样依次往上调，看跳到 P_k 后是否还在 l 的子树内就好了。
- 并且答案显然具有单调性，这启发我们二分答案。
- 考虑如何加速判断答案是否合法这一部分：一个想法是也是用倍增，但是问题是此时我们可能任意指定第一个需要的颜色，你显然不能对每个颜色都处理一遍。

例 6

- 这时候你发现还没有使用「P 中元素互不相同」的性质。

例 6

- 这时候你发现还没有使用「P 中元素互不相同」的性质。
- 我们发现从 v 开始，只要跳了一步之后，设跳到 v_1，颜色是 P_{k_1}，那么下一次肯定是要求跳到颜色 P_{k_1-1}。

例 6

- 这时候你发现还没有使用「P 中元素互不相同」的性质。
- 我们发现从 v 开始，只要跳了一步之后，设跳到 v_1，颜色是 P_{k_1}，那么下一次肯定是要求跳到颜色 P_{k_1-1}。
- 也就是说我们只需要快速跳出第一步（找到 v 上面第一个颜色 c）的点，剩下的点的跳跃过程是固定的，这就可以倍增了。

例 6

- 这时候你发现还没有使用「P 中元素互不相同」的性质。
- 我们发现从 v 开始，只要跳了一步之后，设跳到 v_1，颜色是 P_{k_1}，那么下一次肯定是要求跳到颜色 P_{k_1-1}。
- 也就是说我们只需要快速跳出第一步（找到 v 上面第一个颜色 c）的点，剩下的点的跳跃过程是固定的，这就可以倍增了。
- 最后处理如何快速跳出第一步：这个你就使用可持久化数组直接维护就好了，儿子从父亲继承数组并单点修改。

例 6

- 这时候你发现还没有使用「P 中元素互不相同」的性质。
- 我们发现从 v 开始，只要跳了一步之后，设跳到 v_1，颜色是 P_{k_1}，那么下一次肯定是要求跳到颜色 P_{k_1-1}。
- 也就是说我们只需要快速跳出第一步（找到 v 上面第一个颜色 c）的点，剩下的点的跳跃过程是固定的，这就可以倍增了。
- 最后处理如何快速跳出第一步：这个你就使用可持久化数组直接维护就好了，儿子从父亲继承数组并单点修改。
- 二分 + 倍增，复杂度 $O(q \log^2 n)$。

例 6

- 当然这个题考场上是允许离线的，所以跳出第一步这个可以离线做，这样只需要 dfs 即可。

例 6

- 当然这个题考场上是允许离线的，所以跳出第一步这个可以离线做，这样只需要 dfs 即可。
- 大家也可以自行思考一下（离线的）点分治做法，这可以去掉一个 log。

例 7: The Classic Problem

- 输入一个 n 个点和 m 条边的无向图，其中第 i 条边的边权是 2^{c_i}。
- 要求找到 s 到 t 的最短路。只需要输出对 10^9+7 取模的结果。
- $n, m \leq 10^5, 0 \leq c_i \leq 10^5$

例 7

- ~~最短路啊，那跑个 Dijkstra 就完事了~~ 注意边权范围！

例 7

- ~~最短路啊，那跑个 Dijkstra 就完事了~~ 注意边权范围！
- 但最短路问题一定要用 Dijkstra 解决，所以我们需要实现一个高精度。

例 7

- ~~最短路啊，那跑个 Dijkstra 就完事子~~ 注意边权范围！
- 但最短路问题一定要用 Dijkstra 解决，所以我们需要实现一个高精度。
- Dijkstra 全过程需要支持三个操作：赋值，比较两个数的大小，和加上某个边权。

例 7

- ~~最短路啊，那跑个 Dijkstra 就完事子~~ 注意边权范围！
- 但最短路问题一定要用 Dijkstra 解决，所以我们需要实现一个高精度。
- Dijkstra 全过程需要支持三个操作：赋值，比较两个数的大小，和加上某个边权。
- 比较两个数的大小我们可以用 hash。因为我们要找到第一个不同的位，如果用线段树来维护区间 hash 值，就可以简单比较得出某两个线段树的某个对应区间节点是否是一样的，进而继续二分...

例 7

- 加边权操作只要注意到边权都是 2^c 的形式，如果用二进制数维护，也就是在某一位上 $+1$ 然后进位。这个用线段树二分往高位找到连续的 1，将其修改为 0，并将更高的一位修改为 1 即可。这需要一个区间修改和一个单点修改

例 7

- 加边权操作只要注意到边权都是 2^c 的形式，如果用二进制数维护，也就是在某一位上 $+1$ 然后进位。这个用线段树二分往高位找到连续的 1，将其修改为 0，并将更高的一位修改为 1 即可。这需要一个区间修改和一个单点修改
- 赋值操作直接把线段树改成主席树就好了，赋值就把对应的根的编号 copy 过去。

例 7

- 加边权操作只要注意到边权都是 2^c 的形式，如果用二进制数维护，也就是在某一位上 $+1$ 然后进位。这个用线段树二分往高位找到连续的 1，将其修改为 0，并将更高的一位修改为 1 即可。这需要一个区间修改和一个单点修改
- 赋值操作直接把线段树改成主席树就好了，赋值就把对应的根的编号 copy 过去。
- 在主席树上区间修改一般是要标记永久化的，但是这里是区间置为 0，所以直接将要置为 0 的对应的 $O(\log n)$ 个区间节点删了就行。(这等价于给他连了一个全 0 的子树)

例 7

- 加边权操作只要注意到边权都是 2^c 的形式，如果用二进制数维护，也就是在某一位上 $+1$ 然后进位。这个用线段树二分往高位找到连续的 1，将其修改为 0，并将更高的一位修改为 1 即可。这需要一个区间修改和一个单点修改
- 赋值操作直接把线段树改成主席树就好了，赋值就把对应的根的编号 copy 过去。
- 在主席树上区间修改一般是要标记永久化的，但是这里是区间置为 0，所以直接将要置为 0 的对应的 $O(\log n)$ 个区间节点删了就行。(这等价于给他连了一个全 0 的子树)
- 实际上这个题写暴力进位也能过 (思考如何卡)

例 8: K 个串

- 给出一个长度为 n 的数字序列，数字有正有负。
- 定义一个子串的价值为这个子串的每一种不同的数字之和，列出价值前 k 大的子串
- $n, k \leq 10^5$。

例 8

- 先假设相同的数字可以贡献多次怎么做。

例 8

- 先假设相同的数字可以贡献多次怎么做。
- 这其实就是「NOI 2010 超级钢琴」的弱化版

例 8

- 先假设相同的数字可以贡献多次怎么做。
- 这其实就是「NOI 2010 超级钢琴」的弱化版
- 我们用一个堆来做这个事情，堆中的每个元素形如 $(l, [r_1, r_2])$，表示这个元素考虑所有左端点是 l 且右端点在 $[r_1, r_2]$ 内的所有区间。

例 8

- 先假设相同的数字可以贡献多次怎么做。
- 这其实就是「NOI 2010 超级钢琴」的弱化版
- 我们用一个堆来做这个事情，堆中的每个元素形如 $(l, [r_1, r_2])$，表示这个元素考虑所有左端点是 l 且右端点在 $[r_1, r_2]$ 内的所有区间。
- 最初我们插入 $(i, [i+1, n]) \forall i \in [1, n-1]$。

例 8

- 先假设相同的数字可以贡献多次怎么做。
- 这其实就是「NOI 2010 超级钢琴」的弱化版
- 我们用一个堆来做这个事情，堆中的每个元素形如 $(l, [r_1, r_2])$，表示这个元素考虑所有左端点是 l 且右端点在 $[r_1, r_2]$ 内的所有区间。
- 最初我们插入 $(i, [i+1, n])\forall i \in [1, n-1]$。
- 然后对于每个元素 $(l, [r_1, r_2])$，定义它的权值是 $\min_{r\in[r_1,r_2]} w([l, r])$

例 8

- 先假设相同的数字可以贡献多次怎么做。
- 这其实就是「NOI 2010 超级钢琴」的弱化版
- 我们用一个堆来做这个事情，堆中的每个元素形如 $(l, [r_1, r_2])$，表示这个元素考虑所有左端点是 l 且右端点在 $[r_1, r_2]$ 内的所有区间。
- 最初我们插入 $(i, [i+1, n])\forall i \in [1, n-1]$。
- 然后对于每个元素 $(l, [r_1, r_2])$，定义它的权值是 $\min_{r\in[r_1,r_2]} w([l, r])$
- 这样每次找到堆中的权值最小的元素 $(l, [r_1, r_2])$，它对应的取到最小值的右端点设为 r，那么此时 $[l, r]$ 就是当前没考虑的区间中最小的。

例 8

- 先假设相同的数字可以贡献多次怎么做。
- 这其实就是「NOI 2010 超级钢琴」的弱化版
- 我们用一个堆来做这个事情，堆中的每个元素形如 $(l, [r_1, r_2])$，表示这个元素考虑所有左端点是 l 且右端点在 $[r_1, r_2]$ 内的所有区间。
- 最初我们插入 $(i, [i+1, n]) \forall i \in [1, n-1]$。
- 然后对于每个元素 $(l, [r_1, r_2])$，定义它的权值是 $\min_{r \in [r_1, r_2]} w([l, r])$
- 这样每次找到堆中的权值最小的元素 $(l, [r_1, r_2])$，它对应的取到最小值的右端点设为 r，那么此时 $[l, r]$ 就是当前没考虑的区间中最小的。
- 然后将元素分裂为 $(l, [r_1, r-1])$ 和 $(l, [r+1, r_2])$，继续做即可。

例 8

- 现在考虑加入相同的数字只贡献一次的限制。

例 8

- 现在考虑加入相同的数字只贡献一次的限制。
- 我们对于每个左端点 l，建立一棵线段树 T_l，其中第 r 个叶节点存储区间 $[l, r]$ 的权值。

例 8

- 现在考虑加入相同的数字只贡献一次的限制。
- 我们对于每个左端点 l，建立一棵线段树 T_l，其中第 r 个叶节点存储区间 $[l, r]$ 的权值。
- 这样从 T_{l+1} 继承 T_l，更新一下线段树上 $[l, nxt_l - 1]$ 的值就好了，其中 nxt_l 是 a_l 后面第一个和它相同的位置。打个区间加的永久化标记即可。

例 8

- 现在考虑加入相同的数字只贡献一次的限制。
- 我们对于每个左端点 l，建立一棵线段树 T_l，其中第 r 个叶节点存储区间 $[l, r]$ 的权值。
- 这样从 T_{l+1} 继承 T_l，更新一下线段树上 $[l, nxt_l - 1]$ 的值就好了，其中 nxt_l 是 a_l 后面第一个和它相同的位置。打个区间加的永久化标记即可。
- 每次查询某个元素的权值时就是线段树区间最小值。

可持久化字典树

- 你直接对着模仿一下就好了。每次插入的时候将「更改节点信息」这个操作改为新建节点。

可持久化字典树

- 你直接对着模仿一下就好了。每次插入的时候将「更改节点信息」这个操作改为新建节点。
- 下面放几个例题。

例 1: 最大异或和

- 维护一个长度为 n 的非负整数序列，支持以下操作：
- 添加，在序列末尾添加一个数 x
- 查询，给出 l, r, x，找到一个位置 p 满足 $p \in [l, r]$ 且 $a[p] \oplus a[p+1] \oplus \ldots \oplus a[N] \oplus x$ 最大，输出最大值。这里 N 是当前情况下序列的长度。
- $n, m \leq 3 \times 10^5$

例 1

- 记 $s_p = \bigoplus_{k=1}^{p} a[p]$，注意到 $\bigoplus_{k=p}^{N} a[p] = s_N \oplus s_{p-1}$，所以问题变为找到 $p \in [l-1, r-1]$ 使得 $s_p \oplus (s_N \oplus x)$ 尽量大。

例 1

- 记 $s_p = \bigoplus_{k=1}^{p} a[p]$，注意到 $\bigoplus_{k=p}^{N} a[p] = s_N \oplus s_{p-1}$，所以问题变为找到 $p \in [l-1, r-1]$ 使得 $s_p \oplus (s_N \oplus x)$ 尽量大。
- 实际上就是区间最大异或和。

例 1

- 记 $s_p = \bigoplus_{k=1}^{p} a[p]$，注意到 $\bigoplus_{k=p}^{N} a[p] = s_N \oplus s_{p-1}$，所以问题变为找到 $p \in [l-1, r-1]$ 使得 $s_p \oplus (s_N \oplus x)$ 尽量大。
- 实际上就是区间最大异或和。
- 类比使用可持久化线段树处理区间问题，我们可持久化地建立 n 个 Trie，其中第 i 个 Trie T_i 表示将 $1..i$ 中的数字的二进制从高到低插入得到的 Trie。每个点维护它的子树中有多少终止节点。

例 1

- 记 $s_p = \bigoplus_{k=1}^{p} a[p]$，注意到 $\bigoplus_{k=p}^{N} a[p] = s_N \oplus s_{p-1}$，所以问题变为找到 $p \in [l-1, r-1]$ 使得 $s_p \oplus (s_N \oplus x)$ 尽量大。
- 实际上就是区间最大异或和。
- 类比使用可持久化线段树处理区间问题，我们可持久化地建立 n 个 Trie，其中第 i 个 Trie T_i 表示将 $1..i$ 中的数字的二进制从高到低插入得到的 Trie。每个点维护它的子树中有多少终止节点。
- 查询在 $l-1, r$ 两个 Trie 上走，通过终止节点个数的差即可判断某个方向是否在这个区间内存在。

例 1

- 记 $s_p = \bigoplus_{k=1}^{p} a[p]$，注意到 $\bigoplus_{k=p}^{N} a[p] = s_N \oplus s_{p-1}$，所以问题变为找到 $p \in [l-1, r-1]$ 使得 $s_p \oplus (s_N \oplus x)$ 尽量大。
- 实际上就是区间最大异或和。
- 类比使用可持久化线段树处理区间问题，我们可持久化地建立 n 个 Trie，其中第 i 个 Trie T_i 表示将 $1..i$ 中的数字的二进制从高到低插入得到的 Trie。每个点维护它的子树中有多少终止节点。
- 查询在 $l-1, r$ 两个 Trie 上走，通过终止节点个数的差即可判断某个方向是否在这个区间内存在。
- 这样也解决了树上问题和带修改的问题。

例 2: 异或粽子

- 19 省选签到题。
- 有一个长度为 n 的序列，第 i 个元素为 a_i。
- 定义区间的价值为元素的异或和，求价值前 k 大的区间。
- $n \leq 5 \times 10^5, k \leq 2 \times 10^5$

例 2

- 超级钢琴 + 可持久化 Trie

例 3

- 给出一个长度为 n 序列 a_i，处理 q 次询问。
- 每次询问给出一个区间 $[l, r]$，询问 $[l, r]$ 的所有子区间中异或值最大的那个。

例 3

- 做前缀和后转化为区间中任选两个数异或的最大值。

例 3

- 做前缀和后转化为区间中任选两个数异或的最大值。
- 考虑分块。将序列分为 $O(\sqrt{n})$ 块，对于每个块 i，设它的开头为 l_i，结尾为 r_i，预处理 $g_{i,j}$ 表示 $[l_i, r_i] \times [l_i, j]$ 的答案，这个可以先建立 $[l_i, r_i]$ 的 Trie，然后随着枚举一个个询问并更新最小值。

例 3

- 做前缀和后转化为区间中任选两个数异或的最大值。
- 考虑分块。将序列分为 $O(\sqrt{n})$ 块，对于每个块 i，设它的开头为 l_i，结尾为 r_i，预处理 $g_{i,j}$ 表示 $[l_i, r_i] \times [l_i, j]$ 的答案，这个可以先建立 $[l_i, r_i]$ 的 Trie，然后随着枚举一个个询问并更新最小值。
- 询问的时候整块直接得到答案，散块使用可持久化 Trie 暴力询问。

例 3

- 做前缀和后转化为区间中任选两个数异或的最大值。
- 考虑分块。将序列分为 $O(\sqrt{n})$ 块，对于每个块 i，设它的开头为 l_i，结尾为 r_i，预处理 $g_{i,j}$ 表示 $[l_i, r_i] \times [l_i, j]$ 的答案，这个可以先建立 $[l_i, r_i]$ 的 Trie，然后随着枚举一个个询问并更新最小值。
- 询问的时候整块直接得到答案，散块使用可持久化 Trie 暴力询问。
- 预处理的复杂度就是 $O(n\sqrt{n}\log V)$，询问的复杂度 $O(q\sqrt{n}\log n)$

可持久化并查集

- 就是维护一个并查集，支持合并以及查询任何一个历史状态。
- 如果你只使用按秩合并，并查集复杂度退化到最差 $O(n \log n)$，但是这保证一次只会修改 $O(\log n)$ 个元素（注意：这里不是均摊分析）
- 所以我们实际上需要实现的是可持久化数组，用支持单点修改的可持久化线段树即可。
- 不过复杂度是两个 log。

例 1：归程

- 给出一张 n 个点 m 条边的图，每条边有两个值：海拔和距离。
- 接下来 q 组询问，每次给出水位线和起点。求从起点出发到 1 号节点的最小步行距离。
- 从起点出发时可以驾车，驾车不算做步行距离。但车不能经过海拔低于水位线的边，在那之前需要弃车。
- 强制在线。
- $n \leq 2 \times 10^5, m, q \leq 4 \times 10^5$。

例 1

- 比较简单的做法：Kruskal 重构树，但这不能提现我们数据结构学傻了的优越性。

例 1

- 比较简单的做法：Kruskal 重构树，但这不能提现我们数据结构学傻了的优越性。
- 考虑单组询问怎么做：首先我们跑一遍 Dij 求出 d_i 表示 $i \to 1$ 的最短距离，然后我们只保留车能经过的边（海拔不低于水位线），问题就是求起点 s 所在的连通块中 d_i 的最小值。

例 1

- 比较简单的做法：Kruskal 重构树，但这不能提现我们数据结构学傻了的优越性。
- 考虑单组询问怎么做：首先我们跑一遍 Dij 求出 d_i 表示 $i \to 1$ 的最短距离，然后我们只保留车能经过的边（海拔不低于水位线），问题就是求起点 s 所在的连通块中 d_i 的最小值。
- 如果允许离线：直接将询问和边都按照水位线/海拔排序，用个并查集维护连通块 min 就好了。

例 1

- 比较简单的做法：Kruskal 重构树，但这不能提现我们数据结构学傻了的优越性。
- 考虑单组询问怎么做：首先我们跑一遍 Dij 求出 d_i 表示 $i \to 1$ 的最短距离，然后我们只保留车能经过的边（海拔不低于水位线），问题就是求起点 s 所在的连通块中 d_i 的最小值。
- 如果允许离线：直接将询问和边都按照水位线/海拔排序，用个并查集维护连通块 min 就好了。
- 强制在线：只好使用可持久化并查集。我们将边按照海拔从大到小排序，并设 B_i 表示只保留所有海拔 $\geq i$ 的边的并查集，构造的时候从 B_{i-1} 转移过来并修改，这个修改都是可以可持久化的。

例 1

- 比较简单的做法：Kruskal 重构树，但这不能提现我们数据结构学傻了的优越性。
- 考虑单组询问怎么做：首先我们跑一遍 Dij 求出 d_i 表示 $i \to 1$ 的最短距离，然后我们只保留车能经过的边（海拔不低于水位线），问题就是求起点 s 所在的连通块中 d_i 的最小值。
- 如果允许离线：直接将询问和边都按照水位线/海拔排序，用个并查集维护连通块 min 就好了。
- 强制在线：只好使用可持久化并查集。我们将边按照海拔从大到小排序，并设 B_i 表示只保留所有海拔 $\geq i$ 的边的并查集，构造的时候从 B_{i-1} 转移过来并修改，这个修改都是可以可持久化的。
- 然后询问就直接在对应的并查集上询问即可。

例 1

- 比较简单的做法：Kruskal 重构树，但这不能提现我们数据结构学傻了的优越性。
- 考虑单组询问怎么做：首先我们跑一遍 Dij 求出 d_i 表示 $i \to 1$ 的最短距离，然后我们只保留车能经过的边（海拔不低于水位线），问题就是求起点 s 所在的连通块中 d_i 的最小值。
- 如果允许离线：直接将询问和边都按照水位线/海拔排序，用个并查集维护连通块 min 就好了。
- 强制在线：只好使用可持久化并查集。我们将边按照海拔从大到小排序，并设 B_i 表示只保留所有海拔 $\geq i$ 的边的并查集，构造的时候从 B_{i-1} 转移过来并修改，这个修改都是可以可持久化的。
- 然后询问就直接在对应的并查集上询问即可。
- 这个做法的复杂度凭空 log，可以看做是数据结构学傻的经典样例。