线段树理论

O. 前言

该讲义主要讲解偏理论上的线段树，意在为“如何做线段树题”提供类似于方法论的阐释。

该讲义不需要高深的抽象代数知识，只要理解概念即可。

如无特殊说明，所有变量默认取值范围为 $\mathbb{Z}$。

I. 半群线段树

定义

“半群线段树”是一种数据结构，可以维护一个序列 $a_1, a_2, \ldots, a_n$，其中 $a_i \in D$，$(D, *)$ 是一个**半群**，支持操作：

- 单点修改：给定 p, x $(1 \le p \le n,\ x \in D)$，令 $a_i \leftarrow x$；
- 区间查询：给定 l, r $(1 \le l \le r \le n)$，求 $a_l * a_{l+1} * \cdots * a_r$（或者右结合，下同）。

半群指的是一个元素集合和其上的二元乘法运算，乘法满足结合律。

实现

线段树的结构是大家熟知的，不多赘述。代表区间 $[l, r]$ 的节点 i 上记录了 $S_i = a_l * a_{l+1} * \cdots * a_r$。修改时从叶子自底向上，按照 $T_i = T_{\mathrm{lc}(i)} * T_{\mathrm{rc}(i)}$ 更新 T 值。查询的时候找到能合并出查询区间的极大节点 $i_1, i_2, \ldots, i_k$（按照从左到右排序），查询结果就是 $T_{i_1} * T_{i_2} * \cdots * T_{i_k}$。假设进行一次 $*$ 运算的复杂度是 $O(\tau)$，那么查询、修改复杂度均为 $O(\tau \log n)$。

应用

常见的单点修改，区间求和/求积/求矩阵积/求最大子段和都可以用“半群线段树”做。

区间求和/求积

> *我虽然不知道 2+3 等于多少，但是我知道 2+3=3+2，因为实数和加法构成阿贝尔群。*

由上面这个笑话可知，实数和加法也构成群，所以区间求和是可以做的。当然复数与加法、复数与乘法均构成阿贝尔群，所以一样是可以做的。

区间求矩阵积

矩阵和矩阵乘法是一个广为人知的群。相信大部分 OIer 学矩乘时都会听到“矩阵乘法不满足交换律，满足结合律”。所以区间求矩阵积可以做的。

绝大多数的所谓“线段树维护动态 DP”都是求矩阵积。

一般的可合并区间信息

什么叫做“可合并区间信息”？我们可以这样定义：设这些信息构成集合 D，那么有映射 $f:([1,n]\cap\mathbb{Z})^2\to D$，乘法运算 $*:D^2\to D$ 满足若 $f(l,m)=x$，$f(m+1,r)=y$，那么 $f(l,r)=x*y$。直观理解就是每个区间都只能对应唯一的值，并且乘法对应于相邻区间的合并。

注意到群的定义并未要求乘法要满足对应区间相邻，但是这点无伤大雅。

因为一个区间的信息无论怎么把它拆成子区间、按照什么顺序合并起来，只要左右顺序一致，最后的结果应当是一样的（否则这就不是区间信息）。这说明区间信息与合并操作是构成群的，可以用“半群线段树”做

例题

在做题的时候，需要考虑的大致分为以下几步：

1. 观察题目性质。
2. 我要维护什么?
3. 这种信息是不是某个常见类型?
4. 如果不是，我能否构造信息和操作，使其组成一个半群?
5. 那么，它可不可以用线段树做，做法是否高效?

而每道题不同的地方，也是较难的地方就在于观察性质、设计信息和思考如何合并。如果想要这一部分加速的话，就需要熟知一些常见的信息类型与合并方法。

[Ynoi2013] 文化课

首先发现这个信息就是典型的区间信息，因此只要构造出能合并的维护方式就可以随便线段树了。

观察一段区间，肯定是若干段连乘加起来，合并的时候中间是加号就可以直接拼接，中间是乘号就连接连乘段，特殊考虑一下整个区间就是一个连乘的情况。将上述信息打包一块，就可以轻松合并。

至于修改，改符号是很简单的，预先维护区间和、区间积即可。改数字的话，区间维护一个长为 $O(\sqrt{len})$ 的数组记录不同的连乘段长度和出现次数，这样就能做了。时间复杂度 $O(n\sqrt{n})$。

[CSP-S 2022] 数据传输

先观察性质，在最宽松的 $k=3$ 的情况下，我们可能跳到路径外至多距离 1 的点然后跳回路径上，否则不优。

因为树上查询可以用树链剖分简单处理为序列，我们现在就考虑一条重链怎么做。

因为跳跃距离 ≤ 3，所以很容易考虑设计信息 $I(l,r)$ 为一个 3×3 的矩阵，表示 $l,l+1,l+2$ 走到 $r,r-1,r-2$ 的最短距离（如果某个点超出 $[l,r]$ 范围直接将相应点值设为 $+\infty$ 即可。那么合并就是一个类似矩阵乘法的过程，预处理一下类似最小点权轻儿子之类的信息就可以轻松解决。

II. 环线段树

定义

“环线段树”是一种数据结构，可以维护一个序列 $a_1, a_2, \ldots, a_n$，其中 $a_i \in D$，$(D, *, +)$ 是一个**环**，支持操作：

- 区间修改：给定 l, r, x $(1 \le l \le r \le n,\ x \in D)$，对于所有 $l \le i \le r$，令 $a_i \leftarrow x * a_i$；
- 区间查询：给定 l, r $(1 \le l \le r \le n)$，求 $a_l + a_{l+1} + \cdots + a_r$。

环指的是一个元素集合和乘法、加法运算，乘法满足结合律，加法满足结合律、交换律，乘法与加法满足分配率。

实现

这种线段树也就是大家通常所说的 lazytag 线段树。相比于“半群线段树”，在“环线段树”的一个节点 i 上，除了需要维护 $S_i \in D$ 之外，还需要维护 $T_i \in D$ 代表所谓的“懒标记”，其意味着节点 i 的子树仍然有一些修改尚未下传，这些修改等效于左乘 T_i。通常而言，我们约定 S_i 表示的是**不考虑 i 的祖先（不包括自身）上的懒标记的修改时，i 区间内的 a_i 之和**。

修改、查询时可能需要进行标记下传。当要递归到 i 的左右儿子时进行下传，更新儿子的 S, T 值，然后将 T_i 重置为幺元。

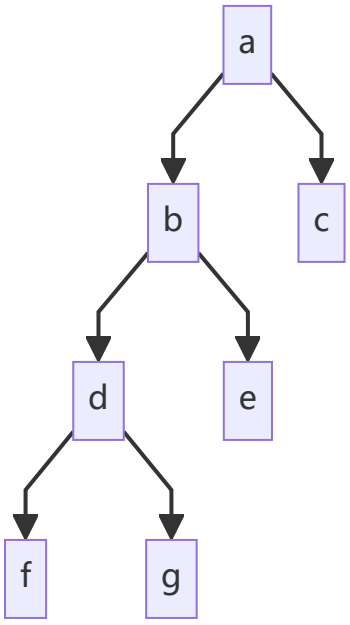
通常而言，若一次“乘法”运算时间复杂度为 $O(\tau)$，那么修改、查询复杂度均为 $O(\tau \log n)$。

标记永久化

在一些特殊的场景中（比如树套树），我们需要进行标记永久化。

标记永久化的其实就是标记不下传。为了做到这点，我们必须保证 $(D, *)$ 不仅仅是一个半群，而是一个**群**，即需要有乘法单位元和逆元。

大概做法就是，仍然维持 S_i, T_i 的含义与上文相同，但是始终不进行标记下传，这就要求我们要对传入的 x 进行一些处理。结合一个实例来讲，假设我们线段树结构如下，我们想给点 g 的区间整体左乘 x：



由于我们不能标记下传，所以肯定不是简单地 $S_g \leftarrow S_g * x$，$T_g \leftarrow T_g * x$，因为要考虑去除 a, b, d 的 tag 的影响。考虑 S_g 的含义是“不考虑祖先的 tag 时子树的和”，这意味着 g 区间的真实区间和 $S'_g = T_a * T_b * T_d * S_g$（这里钦定了标记计算顺序是自顶向下）。而修改过后就有 $S''_g = x * T_a * T_b * T_d * S_g$，改写一下就是 $S''_g = T_a * T_b * T_d * \left(\left(T_d^{-1} * T_b^{-1} * T_a^{-1} * x * T_a * T_b * T_d\right) * S_g\right)$。这个形式就和上面 S'_g 很像了。也就是说，我们只要令 $x' = T_d^{-1} * T_b^{-1} * T_a^{-1} * x * T_a * T_b * T_d$，然后就可以 $S_g \leftarrow S_g * x'$，$T_g \leftarrow T_g * x'$。

至于一般的情况，也很容易举一反三，就不啰嗦了。

value 与 tag 的分离

其实很多时候，所谓“环线段树”并不能很好地描述我们要做的事情，而是得把 value 和 tag 认为是不同的东西。也就是说，我们有值集合 D 和标记集合 E，并且有运算：

- $* : E \times D \to D$，表示给某个值 apply 某个标记的结果；
- $* : E \times E \to E$，表示两个标记的复合，有结合律；
- $+ : D \times D \to D$，表示值的求和，与 E, D 间乘法有分配律。

仍然满足那套运算律，不多赘述。要直观理解的话，一个经典的例子就是，D 为 n 维向量的集合，E 为 $n \times n$ 矩阵的集合。

要做的修改为区间左乘 x $(x \in E)$，查询为求区间值和。

其实 value 和 tag 在数学上是完全可以统一起来的（比如向量可以认为是某种矩阵），分离仅仅为了思维上的方便性。

应用

区间加区间和、区间加乘区间和、区间加区间历史最大值、区间加区间历史版本和都可以“环线段树”做。

区间加乘区间和

考虑维护向量序列 $[a_i, 1]^\top$，那么每次修改都相当于乘上一个 2×2 矩阵。而矩阵与其乘法、加法是构成环的，所以可以用“环线段树”做。

区间加区间历史版本和

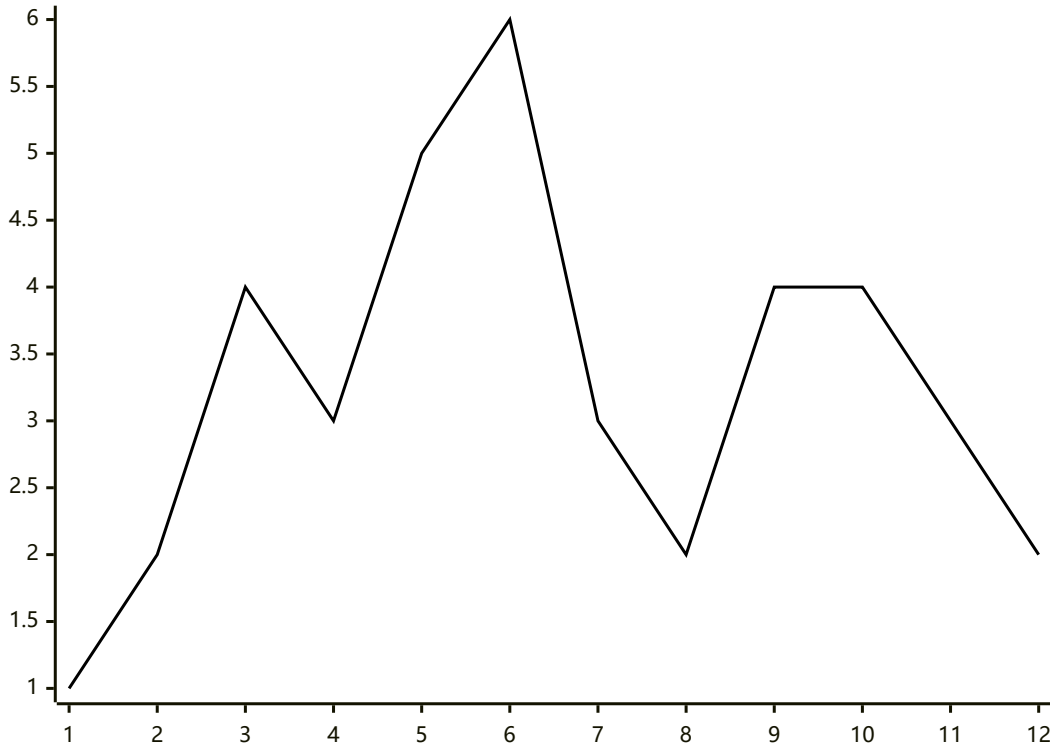
考虑维护向量序列 $[a_i, h_i, 1]^\top$，其中 a_i 是原序列，h_i 是 i 位置的历史版本和。那么各种操作就是：

- 加 x：$a_i \leftarrow a_i + x \cdot 1$，是一个线性变换；
- 累加一次当前版本：$h_i \leftarrow h_i + a_i$，是一个线性变换。

所以也可以用“环线段树”做。

区间加区间历史最大值

这个稍微复杂一些。一个 tag 相当于维护了某个操作序列的信息，那么其能够影响历史最大值的“关键信息”是什么？我们把加法操作过程写成折线图，比如下面这张：



不难发现，有影响的只有折线的末尾高度和最高点高度，所以我们把**标记**记作二元组 (Δ, m)。比如上面这个折线就是 $(6, 2)$。那么两个折线合并得到的新折线是什么呢？显然有：

$$(\Delta_2, m_2) * (\Delta_1, m_1) = (\Delta_1 + \Delta_2, \max\{m_1, \Delta_1 + m_2\})$$

而这种标记的“乘法”类似于前文的区间信息，所以是有结合律的。

对应的“加法”就可以如下定义：

$$(\Delta_1, m_1) + (\Delta_2, m_2) = (\max\{\Delta_1, \Delta_2\}, \max\{m_1, m_2\})$$

显然这个“乘法”关于“加法”是有分配律的。那么就可以直接用“环线段树”做了。

例题

做题时的重心仍然要放在观察性质与构造信息上，相比于“半群线段树”，现在需要考虑的更多一些。

[NOIP2022] 比赛

首先题目没有修改，这是好事，这意味着少了一维限制。

由于直接求解过于逆天，我们可以考虑反过来，计算每个 a_i, b_i 会在哪些区间作为最大值。以 a_i 为例，找到 $L = \min\{j : j \le i,\ a_j \le a_i\}$，$R = \max\{j \ge i,\ a_j \le a_i\}$，那么所有满足 $L \le l \le i \le r \le R$ 的区间都有 $m_a = a_i$。对于 b 也类似。

我们把区间看成平面上的一个点。每个点上维护向量 $[m_a, m_b, v, 1]^\top$，其中 $v = m_a \cdot m_b$，那么修改就是：

- 矩形范围内 m_a 加 x：$m_a \leftarrow m_a + x \cdot 1$，$v \leftarrow v + x \cdot m_b$；
- 矩形范围内 m_b 加 x：$m_n \leftarrow m_b + x \cdot 1$，$v \leftarrow v + x \cdot m_a$。

这些都是线性变换。查询就是求矩形范围内 v 的和。

常用技巧：矩形范围求和可以对其中一维差分，转换为两个区间历史版本和查询。

这里就要查询 v 的历史版本和。相当于向量变为 $[m_a, m_b, v, h, 1]^\top$，矩形范围操作变为区间修改，还要追加：

- 全局累加当前版本：$h \leftarrow h + v$。

但是现在仍然是线性变换，查询仍然是简单求和。所以信息还是一个环，直接上“环线段树”就可做了。

III. 特殊线段树

有一类线段树并不能简单归类到半群线段树或者环线段树，但是通常其底层的信息维护是异曲同工的。

平衡树

平衡树就是支持插入的线段树。

Segment Tree Beats

Segment Tree Beats 意在维护区间 check min/max 操作，也就是 $a_i \leftarrow \max\{a_i, x\}$ 或 $a_i \leftarrow \min\{a_i, x\}$。下面以 check max 为例。

核心思想是势能均摊。感性理解一下，进行一次 check max 很可能会让区间内的颜色数少很多。这样的话我们将所有线段树节点下颜色数之和作为势能函数就能证出暴力的正确性。理性思考一下，什么时候区间内颜色数不会减少呢？发现只能是区间最小值被更新，但是仍然比区间次小值小。

所以做法呼之欲出：每个节点维护最小值 m，次小值 s，以及其它因题目而异的附加信息。修改时到一个节点上时，如果 $s < x < m$，就给节点打上一个**最小值** $+(x - s)$ 的标记并返回，否则直接暴力递归下去。这样子每次递归下去必然伴随势能函数减少，复杂度就是 $O(\log n)$。

如果同时有 check min 和 check max 的话也是可以做的，但是需要更多分讨。

这个和“环线段树”的相通之处在哪呢？其实也就相当于环结构需要多支持“仅仅给最小值加 x”的操作，然后寻找修改区间这部分的算法有所改变。所以很多“环线段树”的方法还是可用的。

底层分块

底层分块是一种线段树空间优化的技巧。大致思想是：线段树的叶子节点长度是 $O(\log n)$ 而不是 1，那么节点数量就减少到了 $O(n/\log n)$。而修改、查询时到叶子直接暴力，复杂度仍然是 $O(\log n)$ 的。

这种修改在有很多个线段树，但是储存的底层下标不交的时候有很好的优化效果，比如 [Ynoi2007] rgxsxrs。

同样，这种线段树可以认为只是结构改变了，信息维护的方式还是和“半群线段树”和“环线段树”一样的。