浅谈函数最值的动态维护

北京大学附属中学 李白天

通过引入关于多个函数上包络线结构的讨论，本文主要对 OI 中的一些著名问题作出了如下改进：

- 将一个序列区间增加公差为正的等差数列，区间查询最值。本问题原本广为人知的做法为 $\Theta(\sqrt{n})$，而本文的做法目前有上界 $O(\log^2 n)$。
- 李超线段树可以在 $\Theta(\log^2 n)$ 内在平面中添加一条线段，并 $\Theta(\log n)$ 查询某一 x 位置上最大的 y 值。而本文通过完全不同的方法在保留询问复杂度的情况下将添加一条线段的复杂度改进为均摊 $\Theta(\alpha(n)\log n)$ 的理论复杂度，且该做法可在高次函数得到一般性的拓展。

1 概述

对于函数列 $f_i : \mathbb{R} \to \mathbb{R}$，我们希望支持对于函数列上进行一些修改，以及至少支持对于整体的最值查询，即给出 x，询问

$$\max_{j=1}^{n} f_j(x)$$

在接下来的分析中，我们需要引入 Davenport-Schinzel 序列的概念。

1.1 Davenport-Schinzel 序列

定义 1 ((n,s) Davenport-Schinzel 序列)**.** 记一个长度为 m 的序列 $\sigma_1, \sigma_2, \ldots, \sigma_m$ 是一个 (n,s) *Davenport-Schinzel* 序列（简记作 $DS(n,s)$ 序列），当且仅当 σ_i 为 1 至 n 中的整数，且满足：

- σ 中相邻两项值不同。
- 对于任意 $x \neq y$，任何 x,y 交替构成的序列如果是 σ 的子序列，则长度不超过 $s+1$。

接下来的一个定理尤为有效地刻画了 DS 序列：

定理 1. *记 $\lambda_s(n)$ 为 $DS(n,s)$ 序列可能的最长长度，有*[1]

$$\lambda_s(n)=\begin{cases} n, & s=1 \\ 2n-1, & s=2 \\ 2n\alpha(n)+O(n), & s=3 \\ \Theta(n2^{\alpha(n)}), & s=4 \\ \Theta(n\alpha(n)2^{\alpha(n)}), & s=5 \\ n2^{\alpha(n)^t/t!+O(\alpha(n)^{t-1})}, & s\geq 6, t=\left\lfloor\dfrac{s-2}{2}\right\rfloor \end{cases}$$

其中 $\alpha(n)$ 是反 Ackermann 函数。由其增长速度我们可知，$\lambda_s(n)$ 对于任意常数 s 都是近乎线性的。

接下来给出 s 较小的两个情况的证明。

1.1.1 $s=1$ 情况的证明

假设序列中存在相同元素 $\sigma_i=\sigma_j(i<j)$，则说明 $\sigma_{i+1}\neq\sigma_i$，而 $\sigma_i,\sigma_{i+1},\sigma_j$ 构成了一个长为 3 的交替子序列，矛盾。

因此我们得到，序列中不存在重复元素，又因为 σ 的元素均为 $1\sim n$ 的正整数，所以 $m\leq n$，任何一个 $1\sim n$ 的排列均取到等号，因此 $\lambda_1(n)=n$。

1.1.2 $s=2$ 情况的证明

我们考虑施第二数学归纳法证明。对于 $n=1$ 的情况，$\lambda_2(1)=1$ 显然成立。

考虑对于 $n>1$ ，此时对于 $<n$ 的 n_0 均有 $\lambda_2(n_0)=2n_0-1$，考虑一个 $DS(n,2)$ 序列，不妨设 $\sigma_1=1$。

- 若 1 仅出现了一次，则 $m\leq 1+\lambda_2(n-1)=2n-2\leq 2n-1$。
- 设 1 出现的下一个位置为 i，则对于 $1<j<i$ 的所有元素 σ_j，因为不存在长为 4 的交替子序列，其在 $>i$ 的位置均不会再次出现。设 $2\sim i-1$ 中总共出现了 k 种元素，则说明 $2\sim i-1$ 构成一个 $DS(k,2)$ 序列，$\geq i$ 的部分构成一个 $DS(n-k,2)$ 序列，因此 $m\leq 1+\lambda_2(k)+\lambda_2(n-k)=1+2k-1+2(n-k)-1=2n-1$。

而 $m=2n-1$ 容易在形如 $1,2,\ldots,n-1,n,n-1,\ldots,2,1$ 的序列取到。综上所属，$\lambda_2(n)=2n-1$ 归纳成立。

[1] Seth Pettie, *Sharp Bounds on Davenport-Schinzel Sequences of Every Order*, 2015

1.2 DS 序列与问题的联系

我们认为 s 可以某种程度上衡量我们维护的一族函数的“复杂程度”。我们注意到，对于我们维护的一族函数，如果任何两个函数 f,g，$\max(f(x),g(x))$ 的分段数量不超过 $s+1$，那么 n 个该族函数的上包络线的分段情况便等价于某个 $DS(n,s)$ 序列，我们称这族函数是 s 阶交替的。因此上包络线分段长度最长只有 $\lambda_s(n)$。

其中最为常见的便是最高 s 次多项式。由于两个 s 次多项式的交点最多只有 s 个，因此最值的分段数量不超过 $s+1$ 个。自然而然地，我们知道对于 n 个最高 s 次多项式，上包络线分段长度不超过 $\lambda_s(n)$。

另一个值得注意的是在一段区间上定义的 s 次多项式。我们不妨将分段函数定义为: 在原本定义域外的部分，函数值都为 $-\infty$。这样可以得到任意两个分段 s 次函数是 $s+2$ 阶交替的，因此 n 个分段 s 次函数的最值的上包络线分段长度不超过 $\lambda_{s+2}(n)$。

在之前 OI 研究的问题中，多为一次函数和分段一次函数，这两个问题分别有 $s=1$ 和 $s=3$。

在接下来的讨论中，我们将问题加以不同的特殊限制，并且在其上能够基于 $\lambda_s(n)$ 的上界设计出较为高效的算法。

1. 不对函数进行修改，仅作为一个集合来维护。这一限制下能够通过二进制分组的思路进行设计。

2. 询问的 x 保证是单调递增的。这一限制下能够通过类似 Segment Tree Beats 的思路进行设计。

2 函数集合维护

设有 n 个函数 $f_i:\mathbb{R}\to\mathbb{R}$，对于 $x_1,x_2,\ldots,x_m$ 中的每个 x_k，求

$$\max_{j=1}^{n} f_j(x_k)$$

当所有函数初始就被确定，我们可以进行分治，每次将当前函数分成两部分进行处理，然后给得到的两组分段函数合并。复杂度即为

$$T(n)=2T\left(\frac{n}{2}\right)+\Theta(\lambda_s(n))$$

根据主定理，解得 $T(n)=\Theta(\lambda_s(n)\log n)$。用于回答的时间为 $\Theta(m\log n)$。

对于函数被逐一确定的情况，我们可以使用二进制分组进行处理。

若插入 n 个最高 s 次函数，则可以做到 $\Theta(\lambda_s(n)\log n)$ 的总时间以及 $\Theta(\lambda_s(n))$ 的总空间进行维护，以及朴素实现可做到 $\Theta(\log^2 n)$ 时间处理每次询问。事实上，我们甚至可以做到 $\Theta(\log n)$ 时间处理每次询问。

对比 $s=3$ 的情况，本算法支持插入线段，均摊时间为 $\Theta(\alpha(n)\log n)$，而询问可以做到 $\Theta(\log n)$，相较于李超线段树的复杂度优秀，并且还有一个额外优点：本算法并不需要提前知道询问的 x 的值，它是完全在线的。

2.1 朴素实现

记 $k=\lfloor \log_2 n \rfloor$，$n$ 的二进制分解为 $\sum_{j=0}^{k} n_j \cdot 2^j (n_j \in \{0,1\})$，对于 $n_j=1$ 的 j，我们维护块 L_j 为其中包含的 2^j 个函数的上包络线所组成的分段。每当加入一个函数的时候，如果存在两个包含同样多个函数的块，我们就将其合并，通过原来两块各自的分段计算出合并后的分段。当合并结束之后，可以得到剩下的各块大小即 n 的二进制分解。

整个过程可以看做一个 2^{k+1} 大小的分治的不完全执行，因此维护的复杂度为 $\Theta(\lambda_s(2^k)k)=\Theta(\lambda_s(n)\log n)$。

而对于查询，我们朴素地在每个块上二分查找到查询坐标所在的分段，时间复杂度为 $\Theta(\log^2 n)$。

2.2 分散层叠

通过 Fractional Cascading 的技术，我们可以优化询问的复杂度。

我们维护辅助数组 $T_k=L_k$，对于 $0\le j<k$，将 T_{j+1} 中所有 3 的倍数位置取出，与 L_j 归并得到 T_j。同时在 T_j 中记录每个元素的来源以及不同来源的前驱后继，便可以在 $\Theta(1)$ 时间内通过在 T_j 中的 lowerbound 找到在 L_j 和在 T_{j+1} 中的 lowerbound。因此查询的复杂度为 $\Theta(\log n)$。

由于 $|L_j|\le\lambda_s(2^j)$，且 $\lambda_s(n)=o(n^{1+\epsilon})$，我们可以得到 T_j 的长度不超过

$$
\begin{aligned}
|T_j| &\le \sum_{t=j}^{k} \frac{\lambda_s(2^t)}{3^{t-j}} \\
&= \sum_{t=0}^{k-j} \Theta\left(\lambda_s\left(\frac{2^{t+j}}{3^t}\right)\right) \\
&= \Theta\left(\lambda_s\left(\sum_{t=0}^{k-j} \frac{2^{t+j}}{3^t}\right)\right) \\
&= \Theta(\lambda_s(3\cdot 2^j)) \\
&= \Theta(\lambda_s(2^j))
\end{aligned}
$$

当一次进位至第 j 位时，重构的复杂度为 $\sum_{t=0}^{j}\Theta(\lambda_s(2^t))=\Theta(\lambda_s(2^j))$，因此可以得到维护的总体复杂度仍然为 $\Theta(\lambda_s(n)\log n)$。

2.3 应用

2.3.1 维护分段一次函数最值

当该数据结构用于维护分段一次函数的时候，我们带入 $s=1$，可得维护的总复杂度为 $\Theta(\lambda_{1+2}(n)\log n)=\Theta(n\alpha(n)\log n)$。这在理论上优于以往李超线段树的 $\Theta(\log^2 n)$ 加入一条线段的复杂度。值得一提的是，存在使 n 条线段的上包络线分段数为 $\Theta(n\alpha(n))$ 级别的构造。[2]因此上述复杂度的分析是紧的。

2.3.2 动态规划问题

对于形如下式的动态规划问题：

$$a_i=\max_{0\le j<i} a_j+w_{j,i}$$

若 $a_j+w_{j,i}$ 能够改写为 $f_j(x_i)$ 的形式，且函数族 f_i 交替的阶数较低，则我们可以通过上述结构进行优化转移。

以往的这类问题中，通常需要将问题改写为一次函数的形式，或者依赖决策单调性解决。

事实上可以说明，这两种情况均是函数为 $s=1$ 阶交替情况下的特例。对于能够改写为一次函数的情况显然，对于决策单调性，通常我们要求 w 满足四边形不等式，从而可以得到

$$\begin{gathered}
\forall i<j, w_{i,j+1}+w_{i+1,j}\le w_{i,j}+w_{i+1,j+1}\\
a_i+w_{i,j+1}+a_{i+1}+w_{i+1,j}\le a_i+w_{i,j}+a_{i+1}+w_{i+1,j+1}\\
f_i(j+1)+f_{i+1}(j)\le f_i(j)+f_{i+1}(j+1)\\
\Delta f_i\le \Delta f_{i+1}\\
0\le \Delta(f_{i+1}-f_i)
\end{gathered}$$

这说明对于可以通过四边形不等式进行决策单调性优化的问题，对于 $i\le i_0$ 的函数族 f 在 $j\ge i_0$ 的定义域上是 $s=1$ 阶交替的。

因此我们可以看到，上述数据结构在优化动态规划时能很好的兼容一些以往常见的条件。虽然在决策单调性问题上复杂度并不能达到最优，但上述数据结构却能处理性质略为复杂的转移代价函数。

[2] Ady Wiernik, Micha Sharir, *Planar realizations of nonlinear Davenport–Schinzel sequences by segments*, 1988

3 询问点单调递增

设有 n 个函数 $f_i:\mathbb{R}\to\mathbb{R}$，对于 $x_1<x_2<\cdots<x_m$ 中的每个 x_k，求

$$\max_{j=1}^{n} f_j(x_k)$$

接下来我们讨论中假设处理的函数族是 s 阶交替的，且可以在常数时间内求出交替位置。

3.1 Kinetic Tournament 树

我们考虑一颗线段树，每个叶子节点表示了一个函数，线段树的每个节点存储了当前时间下子树中叶子节点的函数在当前 x 下取到最大值的那个函数。在 x 增加的过程中，我们需要不断修改某些节点的取值。我们维护 x 变大后子树里使得某个 max 所取函数第一次发生改变的 x 的值，当下一次询问超过这一 x 时，我们递归下去将发生替换的部分重置。维护这种信息的线段树我们称为 Kinetic Tournament 树。之后我们简称为 KTT。

考虑线段树中每个节点被修改次数，这等价于子树内所有函数包络线的分段数，因此所有节点修改次数总和与分治同理为 $\Theta(\lambda_s(n)\log n)$。因此直接在线段树上进行维护，由于每次修改需要走到该叶子，本算法的复杂度为 $\Theta(\lambda_s(n)\log^2 n)$。这比直接分治要显得逊色，但是我们将要看到这一方法在更强问题上的潜力。

3.2 带修改函数序列最值问题

本问题的带修改版本即允许在过程中在线将某个函数重新赋值。总共修改 m 次，询问 q 次。

我们考虑通过 KTT 来对此进行维护。在将一个函数重新赋值的时候，考虑将线段树中对应节点修改，并且更新到根节点的一串节点。

这样维护的复杂度是什么呢？我们考虑将维护过程进行如下等价转化：每个叶子节点假设被修改了 k_i 次，我们将其转化为这个节点下面放置 k_i+1 个节点，第一个节点是初始函数，后面接着是剩下的 k_i 个函数，它们都假设是在其存在时间上的分段函数。那么一个管辖区间为 $[l,r]$ 的节点子树里的上包络线分段数不会超过 $\lambda_{s+2}(r-l+1+\sum_{i=l}^{r}k_i)$，因此同一层的包络线分段总和不会超过

$$\begin{aligned}\sum_{[l,r]}\lambda_{s+2}\left(r-l+1+\sum_{i=l}^{r}k_i\right)&\le\lambda_{s+2}\left(\sum_{[l,r]}\left(r-l+1+\sum_{i=l}^{r}k_i\right)\right)\\&=\lambda_{s+2}(n+m)\end{aligned}$$

经过以上分析，可知对于在线修改的最值查询中，KTT 可以做到 $\Theta(\lambda_{s+2}(n+m)\log^2 n+q)$ 的时间内完成计算。

我们将看到这一结构如何可以用于优化一个经典的有关一次函数的问题。

4 线性情况的扩展

由于线性函数的性质良好，我们可以在其上研究一些扩展问题，遗憾的是笔者暂时未能确认在更高阶情况下的类似问题维护是否有相近的复杂度。

4.1 包含两类区间修改的序列最值问题

对于数列 k_i, b_i，我们有如下两种修改操作：

- 给定 l, r, x，对于 $l \le i \le r$，使 $b_i \leftarrow k_i x + b_i$。
- 给定 l, r, c, k', b'，对于 $l \le i \le r$，使 $k_i \leftarrow ck_i + k', b_i \leftarrow cb_i + b'$。

以及进行询问一段区间上 b_i 的最值。

本问题的做法改进自[3]。

这一问题在一部分 OI 题中有所部分出现，而这些问题在之前基本都只给出了 $\Theta(n + (m+q)\sqrt{n})$ 复杂度的基于分块的做法，而我们将看到通过 KTT，我们可以在 $O(n\log^2 n + m\log^3 n + q\log n)$ 时间内完成操作和询问，需增加限制：**第一种修改操作中保证 $\mathbf{x} > \mathbf{0}$**，且为了叙述主要思想，在这里仅考虑 $c > 0$ 的情况。

具体的做法思路是简单的：考虑在线段树上记录惰性标记，表示操作 2 的累计变化效果以及操作 1 的 x 的累计。我们注意到操作 1 本质上就是对 KTT 上的某个节点的子树开始进行“x 增大”的操作，而不是像原始的 KTT 上只从根节点进行。

4.1.1 复杂度分析

我们考虑通过势能来分析这一算法的复杂度。

我们定义一个非叶子节点组成的集合 $\mathcal{P}$，对于一个非叶子节点 v，如果 v 在当前保留的取值较大的孩子所对应的直线斜率是严格小于另一个孩子的，那么 $v \in \mathcal{P}$。

我们定义势函数 $\Phi = \sum_{v\in\mathcal{P}} d(v)$，其中 $d(v)$ 是节点 v 在线段树上的深度，根节点的深度为 1。

[3]Daniel Zhang, https://codeforces.com/blog/entry/68534?#comment-530381, 2019

我们考虑 1 操作的最坏情况，即每个节点更新操作是单次进行的，那么我们定义一次更新操作的代价为 1，一次更新操作的均摊代价是 $1+\Delta\Phi$，其中对于被修改的节点 v，可以得知 v 必然从在 $\mathcal{P}$ 中变为不在，而 v 的父节点 p 有可能从不在 $\mathcal{P}$ 中变成在 $\mathcal{P}$ 中。故

$$\begin{aligned}\widehat{c} &= 1+\Delta\Phi \\ &\le 1-d(v)+d(p) \\ &= 1-d(v)+(d(v)-1) \\ &= 0\end{aligned}$$

我们这一定义使得原本无法确认进行次数的更新操作在均摊代价中不需要被考虑。接下来考虑操作 1 和操作 2，它们其实本质上是类似的。注意到当一个节点的惰性标记被修改或以其为子树的数被更新时，这一节点的父节点可能由不在 $\mathcal{P}$ 中变为在 $\mathcal{P}$ 中，这最多导致势能增加 $d(p)=\Theta(\log n)$。而一次操作会影响 $\Theta(\log n)$ 个节点，因此一次操作导致势能增加最多 $\Theta(\log^2 n)$。

还有一点需要注意的是由于 $\sum\widehat{c_i}=\sum c_i+\Phi_t-\Phi_s$，因此我们给总共的代价加上 $\Phi_s-\Phi_t=\Theta(n\log n)$，这是因为最坏情况是最初所有节点均在 $\mathcal{P}$ 中。

综上，操作过程中最多会引发 $\Theta(n\log n+m\log^2 n)$ 次更新操作，因此复杂度有上界 $O(n\log^2 n+m\log^3 n+q\log n)$。

4.1.2 特殊情况

在诸如“区间增加公差为正的等差数列”中，我们提炼出一个隐含条件：序列的 k 是单调递增的。因此引发的更新操作必然是一个节点所取到的 max 由在左子切换为在右子。

此时我们定义势能 Φ 为“max 位于左子”的节点数。由于操作过程最多引发 $\Theta(n+m\log n)$ 次更新操作。我们有复杂度上界 $O(n\log n+m\log^2 n+q\log n)$。

4.2 例题

例题 1. *Bear and Bowling*[4]

这里仅讨论与本文相关的一种解法，在该解法中我们需要按照一个顺序在一个序列 a_n 中每次拿走一个数，设该数左边已经被拿走的数有 k_i-1 个，右边已经被拿走的数的总和为 s_i，那么每次拿的数必须是 $k_i\cdot a_i+s_i$ 中最大的。[5]

如果我们使用上述的数据结构，那么只需最初令 $b_i=a_i$，每次拿走一个数后我们将 b_i 加上 $-\infty$，给 $[1,i-1]$ 这一段加上 a_i，给 $[i+1,n]$ 这一段执行 $b_j\leftarrow b_j+a_j$ 即可。

[4]Codeforces 573E，https://codeforces.com/problemset/problem/573/E

[5]徐翊轩, *IOI2020* 中国国家集训队第一阶段作业试题准备, 2019

例题 2. *Innophone*[6]

给若干个 x_i, y_i，选取 a, b 最大化

$$\sum_{i=1}^{n} [a \le x_i]a + [a > x_i][b \le y_i]b$$

我们按照 y 排序，扫描过程中在 KTT 上区间增加斜率，按照对应的 y 值查询就可以了。

4.2.1 最大连续子段和

题目大意 对于一个数列 a_n，支持将区间上整体加一个正数，询问一个区间上的最大子段和。

解法 这是一个较为复杂的问题，我们首先回顾没有修改操作的做法。我们在线段树的每个节点上可以维护四个值 $sum, lmax, rmax, totmax$，即可得到维护方法

$$\begin{aligned} sum &= ls.sum + rs.sum \\ lmax &= \max(ls.lmax, ls.sum + rs.lmax) \\ rmax &= \max(rs.rmax, rs.sum + ls.rmax) \\ totmax &= \max(ls.totmax, rs.totmax, ls.rmax + rs.lmax) \end{aligned}$$

我们考虑将现在维护的这 4 个信息均表示为一个一次函数，那么每个节点维护的击败时的 x 即上式中所有 max 所取函数发生变更时间的最小值。

我们记一个包含 max 比较的变量的 rank 值 $r(v, para)$:

- 对于 $r(v, lmax)$ 或 $r(v, rmax)$：对于该变量的决定式 $f(x) = \max(a(x), b(x))$，rank 值表示当前 a, b 中斜率大于 f 的斜率的直线的数量乘以 $d(v)^2$。
- 对于 $r(v, totmax)$：对于该变量的决定式 $f(x) = \max(a(x), b(x), c(x))$，rank 值表示当前 a, b, c 中斜率大于 f 的斜率的直线的数量乘以 $d(v)$。

我们直接定义势能 $\Phi = \sum_v \sum_{para} r(v, para)$。

- 当发生一次 $lmax$ 或 $rmax$ 的击败，有:

$$\Delta\Phi \le 1 - d^2 + (d-1)^2 + d - 1 = 1 - d \le 0$$

[6]ROI 2018，https://loj.ac/problem/2845

- 当发生一次 *totmax* 的击败，有：

$$\Delta\Phi \le 1 - d + d - 1 = 0$$

可以得到本算法的一个复杂度上界 $\Theta(n\log^3 n + m\log^4 n + q\log n)$，其中 m 为修改次数，q 为询问次数。

但类比之前对于斜率单调的特殊情况所进行的分析，我们考虑每个节点的 *lmax* 和 *rmax* 的决策发生变化的次数。注意到 *lmax* 的决策点只会递增，而在一个节点发生斜率切换的必要条件是决策位置从 $[l, mid]$ 切换到了 $[mid+1, r]$。每个节点最多触发一次，故 *lmax*, *rmax* 的总共额外复杂度是 $\Theta(n\log n)$。因此我们现在知道在这个问题上 KTT 维护 *lmax* 和 *rmax* 的总共 dfs 到的节点数是 $\Theta((n+q)\log n)$ 的。我们沿用 *totmax* 的势能分析，复杂度是 $O((n+m)\log^3 n + q\log n)$。

但这其实并没有说明之前的势能分析完全冗余，刚刚所作的 $\log^4 n$ 的分析可以认为是在考虑一个更加一般性的操作，即加法是令 $a_i \leftarrow a_i + k_i x$，而原问题中隐含了 $k_i = 1$，即 $k_i > 0$。

4.2.2 henry_y 的数列

题目大意 给一个长为 n 的数列 A_i，支持修改操作：

给定 l, r, a, b, c，对于 $l \le i \le r$，使 $A_i \leftarrow A_i + ai^2 + bi + c$，保证 $a, b \ge 0$，以及询问一段区间上 A_i 的最小值。[7]

解法 这一问题中看似有二次项，但二次项中的 i^2 是只与下标相关的，因此实际上更接近于维护若干个二元一次函数的最值。

首先还是类似 KTT 的思想，考虑用线段树先维护当前最值，以及加了 (ai^2+bi) 到什么时候会使得子树的某个比较关系发生变更。由于 $a, b \ge 0$ 且 $i \ge 1$，所以最小值只可能从右子树切换到左子树。

假设左子树的最值在 A_i 处，右子树的最值在 A_j 处，若 $A_i > A_j$，则有可能未来 A_j 反超 A_i，这需要

$$\begin{aligned} A_i + ai^2 + bi &\le A_j + aj^2 + bj \\ A_i - A_j &\le a(j^2 - i^2) + b(j - i) \\ \frac{A_i - A_j}{j - i} &\le a(i + j) + b \end{aligned}$$

[7]改自「hyOI2020」，https://loj.ac/problem/6727

由此可知，当 (a,b) 进入一个半平面时，子树的最值发生切换。反过来说，若在该半平面的补集中，则还未发生最值切换。因此若当前的 (a,b) 修改不发生子树的切换，则需要该点落在子树的所有限制之内，即所有半平面的交集之内。由于半平面的直线斜率为 $i+j$，所以左子树中的所有斜率均小于右子树中的所有斜率，我们可以使用可持久化平衡树来维护半平面交，并且在平衡树上进行二分从而在 $\Theta(\log n)$ 的时间得到两个半平面交合并的结果。

由于本问题具有与前文问题相似的单调性，我们可以设势能 Φ 为“min 位于右子”的节点数。由于操作过程最多引发 $\Theta(n+m\log n)$ 次更新操作，每次操作消耗 $\Theta(\log^2 n)$ 复杂度用于更新信息。我们有复杂度上界 $O((n+m\log n)\log^2 n+q\log n)$。

5 总结

本文主要围绕 $DS(n,s)$ 序列本身长度的一个优秀上界，通过“减少维护的必要交点数”的方法进行设计，得到了一些复杂度比较优秀的方法用于维护函数的最值。希望本文上述的一些思路可以对 OI 中维护最值有关的问题产生更多的启发。

致谢

感谢中国计算机学会提供学习和交流的平台。
感谢北大附中肖然老师的关心和指导。
感谢家人、朋友对我的支持与鼓励。
感谢戴江齐同学和刘承奥学长与我讨论以及给予的启发。
感谢房励行同学为本文刊误。

参考文献

[1] Micha Sharir, Pankaj K. Agarwal, *Davenport-Schinzel Sequences and their Geometric Applications*, 1995

[2] Seth Pettie, *Sharp Bounds on Davenport-Schinzel Sequences of Every Order*, 2015

[3] Ady Wiernik, Micha Sharir, *Planar realizations of nonlinear Davenport–Schinzel sequences by segments*, 1988

[4] 徐翊轩, *IOI2020 中国国家集训队第一阶段作业试题准备*, 2019

[5] Daniel Zhang, https://codeforces.com/blog/entry/68534?#comment-530381, 2019