对信息学竞赛中二维平面处理问题的总结和再优化

福建省厦门双十中学 施良致

摘 要

本文总结了在信息学竞赛中对于二维平面处理问题的常见做法——KD-Tree 和区域树（线段树套线段树），并利用分散层叠算法对于区域树进行改进，优化其时间复杂度。在最后一节中，笔者通过例题，展示利用分散层叠算法改进的区域树在信息学竞赛中的应用。

1 前言

二维平面处理问题在信息学竞赛中是比较经典的问题。对于此类问题，在信息学竞赛中一般使用 KD-Tree，树套树，分块，CDQ 分治等算法解决。然而，在数据规模为 n ，询问次数为 m 时，这些算法的时间复杂度在一般情况下都不优于 $O(m\log^2 n)$ 和 $O(m\sqrt{n})$。

本文将介绍一种在信息学竞赛中不常出现的新处理方法，该算法在某些特定的情况下能将此类问题的时间复杂度优化至 $O(m\log n)$。此外，本文还会通过例题展示其在信息学竞赛中的应用，并且本文还会将其与选手熟知的算法进行对比，分析该算法的优劣。

2 经典在线的二维问题

问题 1. 给定平面上 n 个点，第 i 个点的坐标为 (x_i, y_i)，其权值为 w_i。接下来将进行 m 次查询，每次查询一个左下角为 (x_1, y_1) ，右上角为 (x_2, y_2) 的矩形中的点的权值和，查询强制在线。

上述问题为一个经典的二维平面处理问题。对于该问题，下文将介绍三种解决方法。前两种分别是较为常见的 KD-Tree 和区域树，第三种为分散层叠（Fractional Cascading）优化区域树算法。

2.1 KD-Tree

对于此类在线的矩形查询问题，有经典的解决方法，即 KD-Tree[1]。

[1]参考资料：https://oi-wiki.org/ds/kdt

KD-Tree 是一种可以高效处理 k 维空间信息的数据结构，其在处理二维问题上是非常有用的工具。

2.1.1 解决方案

KD-Tree 的构造方法为每次选择一个维度，并切分这个维度。

对于本问题，KD-Tree 的构造方法为交替选择维度（横向或纵向），然后按照此维度排序（按横坐标排序或按纵坐标排序）。如图所示[2]，排序后选取中点并按照中点的位置将平面切割开，此时所有点被分成两个集合，这两个集合分别递归构造子树。

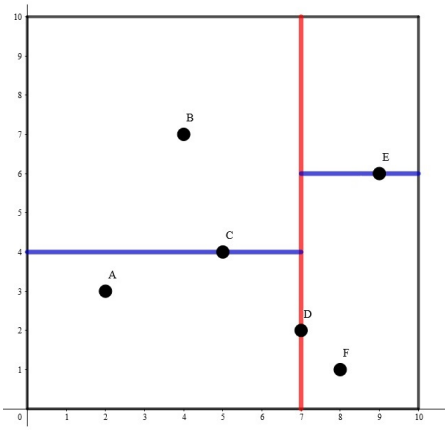


图 1: KD-Tree 的构造

按此方式构造后，KD-Tree 上的每个节点都对应平面上一个矩形区域。

查询时可以遍历整个 KD-Tree，对于每个节点的处理可以分为以下三种：

1. 当前矩形的边框或内部与查询矩形的边框相交：递归查询该节点的左子树和右子树。
2. 当前矩形被查询矩形相含：直接返回该节点对应子树中所有点的权值和。
3. 当前矩形与查询矩形相离：子树中所有点都不在查询矩形中，直接返回。

2.1.2 时间复杂度

定理 2.1. 对于任意一个查询矩形的边框，在 KD-Tree 上与之有交的矩形为 $O(\sqrt{n})$ 。

证明. 可以将矩形边框拆分成四条线段，而相交的矩形数不会超过分别和每条线段相交的矩形数目的总和。

接着将线段向两边延伸转换为直线，与线段相交的矩形数目不会超过与直线相交的矩形数目。

[2]图片来源：https://oi-wiki.org/ds/images/kdt1.jpg

每次考虑两层节点，即把当前矩形划分成四份，设 $F(n)$ 表示 n 个节点的 KD-Tree 与一条直线相交的矩形数。

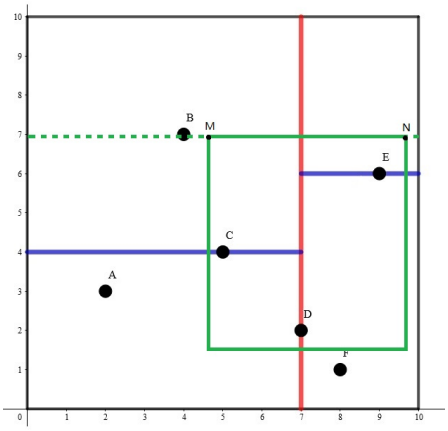


图 2: KD-Tree 复杂度分析

由于任意一条直线只会和四个矩形中的两个相交，因此得到: $F(n) = O(2) + 2F(n/4)$。

根据 Master Theorem[3] 得到，$F(n) = O(\sqrt{n})$ 。

□

定理 2.2. 对于点数为 n 的 KD-Tree，每次查询的时间复杂度为 $O(\sqrt{n})$。

证明. 根据构造，不难得知，当某个节点对应的矩形的边框或内部与查询矩形的边框不相交时，均不用继续递归。当某个节点对应的矩形的边框或内部与查询矩形的边框相交时，均会继续递归。

因为每个节点的子节点最多两个，设有 x 个节点对应的矩形与查询矩形的边框相交，那么被访问的节点数会严格小于 $3x$ 。根据定理 2.1 ，可知 x 为 $O(\sqrt{n})$ ，因此 KD-Tree 单次查询的时间复杂度为 $O(\sqrt{n})$。 □

2.2 区域树

区域树（Range Tree），在信息学竞赛中一般被称为线段树套线段树。这是一种常见的对二维问题处理的方法。本文后面将会使用分散层叠算法对区域树进行优化。

2.2.1 解决方案

区域树的构造方法为将两个维度分开考虑。先对第一个维度建立一棵静态线段树，线段树上每个节点对应一段数据。对于这些数据，再按照第二个维度建立一棵静态线段树。

[3]参考资料：https://oi-wiki.org/misc/complexity/

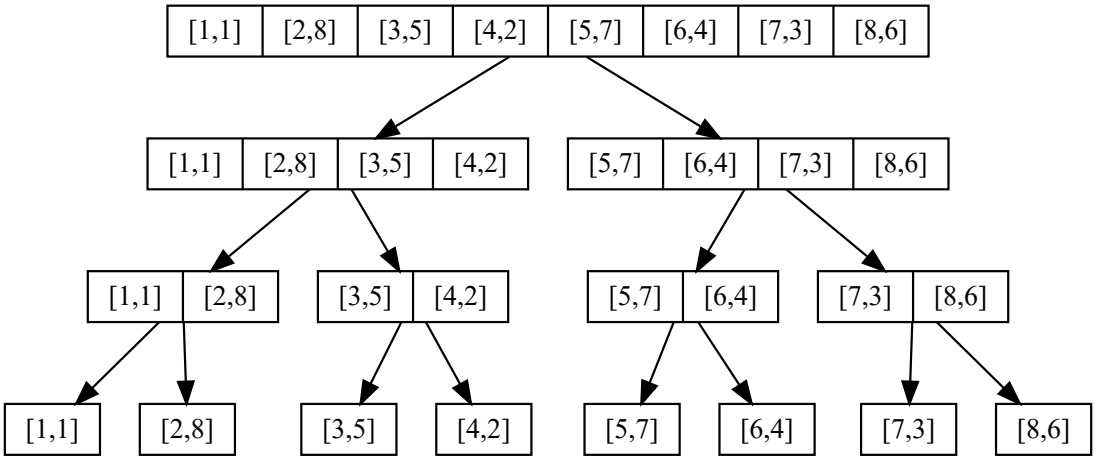


图 3: 区域树，每个节点上存有一定数据，每个节点上的数据独立建立线段树

在查询时，可以在外层线段树上查询出第一个维度对应的节点，再在对应节点的内层线段树上查询第二个维度的信息。

2.2.2 时间复杂度

不难证明，在外层线段树上会对应 $O(\log n)$ 个节点，每一个节点在内层线段树上会对应 $O(\log n)$ 个节点，因此时间复杂度为 $O(m \log^2 n)$。

2.3 分散层叠算法优化区域树

在信息学竞赛中，对于此问题的在线算法大多都停留在时间复杂度为 $O(m \log^2 n)$ 或 $O(m\sqrt{n})$ 的做法上。事实上，这类问题可以优化至时间复杂度为 $O(m \log n)$，且空间复杂度仍为 $O(n \log n)$。

学术界中存在一种名为分散层叠（Fractional Cascading）的算法，在信息学竞赛中，这个算法在 IOI2020 中国国家候选队论文《浅谈利用分散层叠算法对经典分块问题的优化》中出现过。该算法可以运用在对区域树的优化上，并且可以将此类问题的时间复杂度优化至 $O(m \log n)$ 。在下文中（除 2.3.1 外），文中分散层叠算法的意思均为**分散层叠算法优化区域树**。

2.3.1 分散层叠

分散层叠是一种在多个按一定组织结构的数列中对一个数进行加速二分查找的技术[4]。由于本文的重点并非介绍分散层叠算法，因此本文只讨论分散层叠对某一种特定结构的优化。

对于两个数集 A, B，保证 $B \subseteq A$，此时是否可以通过构造映射 $f : A \mapsto B$，使得若知道了 A 中比 x 大的第一个数时，可以在 $O(1)$ 的时间复杂度下得到 B 中比 x 大的第一个数。

不难想到，只需要让 A 中的每个数映射到 B 中第一个大于等于自己的数上。

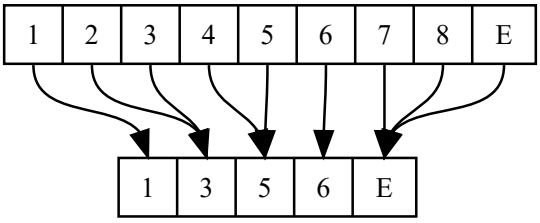


图 4: 分散层叠算法，E 为集合末尾的空节点

例如 $x = 4$，在 A 集合中大于等于 4 的第一个数是 4，并且由映射关系可以得知在集合 B 中大于等于 4 的第一个数是 5 。

2.3.2 改进区域树结构

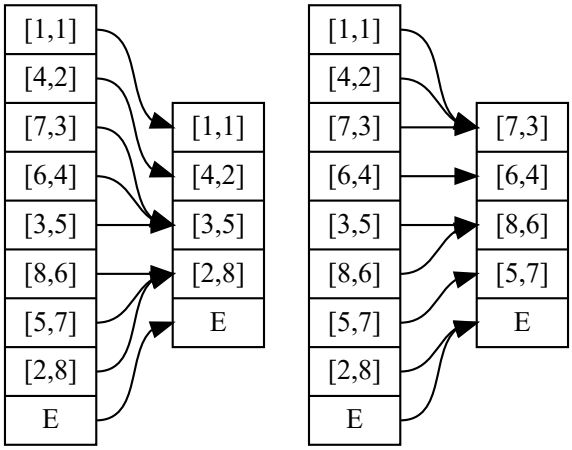
在 2.2 中有提到区域树的实现思想，即将数据的两个维度分开处理，先按第一个维度建立静态线段树，然后在每个节点上将对应的数据建立一棵线段树。

然而对于问题 1，在第二维上其实并不需要使用线段树。如果先将每个节点上的数据按照第二维从小到大排序，并且知道所查询节点上第二维符合询问的数据对应的区间，那么只需要一个前缀和就可以解决权值和的问题。在这里，分散层叠算法恰好可以解决定位区间的问题。

以 2.2.1 中的图 3 的区域树举例，对于每个节点上的数据，可以先按第二维从小到大排序，然后对于上下两层按第二维的大小建立 2.3.1 中所描述的映射关系。

以此类推，每一层的节点中的数据都可以和下一层节点中的数据建立映射关系。这个映射关系的建立可以用归并算法实现。

[4]此为《浅谈利用分散层叠算法对经典分块问题的优化》的原话



(a) 左节点 (b) 右节点

图 5: 第一层和第二层的节点数据之间的映射关系

2.3.3 查询方式

与区域树查询数据的方法类似的，我们首先要在外层线段树上查询得到第一维符合条件的节点，而分散层叠算法可以在外层线段树节点移动的同时在 $O(1)$ 的时间复杂度下确定节点上第二维合法的数据所对应的区间。

假设查询第二维的范围为 $[l, r)$，则一个节点上合法的数据为第二维数值第一个大于等于 l 的数据到第一个大于等于 r 的数据之间的所有数据。而如果得到了这两个数据所在位置，其子节点上大于等于 l 和大于等于 r 的第一个数据所在位置也可以 $O(1)$ 得到。

以 2.2.1 中图 3 的区域树举例，我们希望查询第一维在 $[1, 6]$，第二维在 $[3, 8)$ 中的所有数据。

首先可以通过二分的方式在根节点中确定 $[3, 8)$ 的位置，即根节点中第二维大于等于 3 和 8 的第一个数据，在该例子中为 $[7, 3]$ 和 $[2, 8]$ 两个数据。

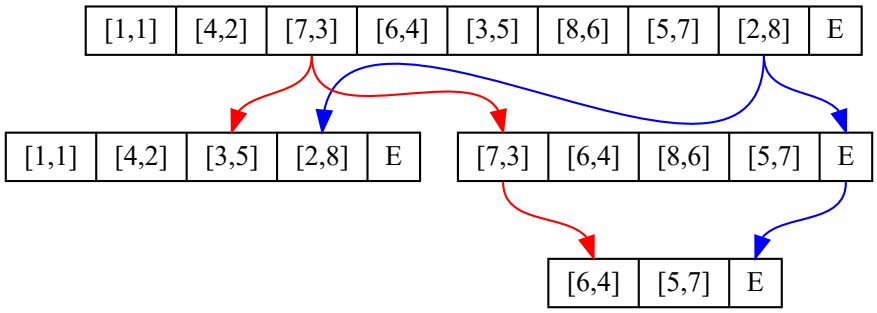


图 6: 分散层叠算法的查询方式，红色为 l 指针的移动方式，蓝色为 r 指针的移动方式

如图，在外层线段树查询时，总可以通过 $O(1)$ 的时间找到子节点中第二维在 $[3,8)$ 中的数据。

不难发现，对于每一个节点中的数据，某段区间的点的权值和可以用前缀和差分计算，这样是 $O(1)$ 的时间。因此，当外层线段树的搜索结束时，整个查询也就结束了。

2.3.4 空间复杂度分析

在 2.3.2 中介绍的这种算法的构造方法为归并算法，该算法总共的层数在 $O(\log n)$ 级别，每一层所有数据数目之和为 n 。

对于每一个数据，除了自身外，还需要多记入其在下层两个区间中对应的后继以及计算答案需要使用的前缀和，但空间复杂度仍为 $O(n\log n)$。

2.3.5 时间复杂度分析

第一维的查找和线段树的时间复杂度是一致的，单次为 $O(\log n)$。但在第二维查找上，该算法在定位区间和计算答案上都是 $O(1)$ 的。

综上所述，对于每一次查找，该算法的时间复杂度为 $O(\log n)$。

2.4 三种算法的比较

2.4.1 从空间上分析

对于 KD-Tree ，其空间复杂度为 $O(n)$，而区域树和分散层叠算法的空间复杂度均为 $O(n\log n)$。因此，KD-Tree 在处理此类问题时在空间复杂度上占有极大优势。

2.4.2 从时间上分析

对于 KD-Tree、区域树和分散层叠三种算法，其理论复杂度分别为 $O(m\sqrt{n})$、$O((n+m)\log^2 n)$ 和 $O((n+m)\log n)$ 的。从理论上看，分散层叠算法相对前两者在时间复杂度上占极大的优势。然而，在实际中并非如此。

算法名称	KD-Tree	区域树	分散层叠
单次查询复杂度	$O(\sqrt{n})$	$O(\log^2 n)$	$O(\log n)$
递归次数	2×10^9	6×10^8	6×10^7
运行时间 (s)	45	15.7	6.7

表 1: 在 $n=2\times10^5, m=10^6$ 时三种算法时间复杂度的比较

通过实验分析[5]，分散层叠算法在递归次数上有很大的优化。当 $n = 2 \times 10^5, m = 10^6$ 时，区域树的函数递归次数约为 6×10^8 次，KD-Tree 的函数递归次数约为 2×10^9 次，而分散层叠算法的递归次数只有 6×10^7 次。然而，分散层叠算法在移动指针、计算答案时都需要进行繁琐的操作，该操作成本使得分散层叠算法有一个较大的常数，这使得分散层叠算法在时间效率上表现较劣。

不过尽管如此，在 n 和 m 较大时，分散层叠算法在运行时间上相对区域树和 KD-Tree 还是有较明显的优势。经过反复实验测试，对于问题 1，在 $n = 2 \times 10^5, m = 10^6$ 的随机数据下，KD-Tree 的运行时间最坏会达到 45s，区域树的平均运行时间为 15.7s，分散层叠算法的平均运行时间为 6.7s。由此可以看出，分散层叠算法虽然不能达到期望下比区域树快 10 ~ 20 倍的水平，但还是有明显的优势。

3 经典离线的二维问题

问题 2. 给定平面上 n 个点，第 i 个点的坐标为 (x_i, y_i)，其权值为 w_i。现在有 m 个询问，每个询问为查询一个以 (x_1, y_1) 为左下角，以 (x_2, y_2) 为右上角的矩形中的点的权值的最大值。

与问题 1 类似，该问题也可以使用 KD-Tree 和区域树解决，方法与问题 1 基本相同，时间复杂度也基本相同。而问题 1 中分散层叠算法利用了前缀和的性质，在这里失去了这个性质，所有没办法使用解决问题 1 的方法。因此这里将介绍另外一种方法。

3.1 问题转换

与问题 1 类似，在外层线段树查找时可以顺便得出每个节点合法数据的区间。而外层线段树上最多对应 $O(\log n)$ 个节点，此时每个节点中有一个询问，为询问这个节点的数据中某一段的最大值。

在线处理并不方便，因此考虑离线处理。不妨先把所有询问都记录下来，接着把这些询问按照其在外层线段树上对应的节点分组，把在同一个节点上的询问分成同一组。

分组后再按组回答询问。每一组询问都可以表示成下列形式：先给定一个长度为 k 的序列，再给定 q 个询问，每个询问为询问 $[l_i, r_i]$ 中数据权值的最大值。如果把所有组的 k 和 q 累加，可以得到 $O(\sum k) = O(n \log n), O(\sum q) = O(m \log n)$。

3.2 解决新问题

由于每一组问题的每个询问都可以表示为一个二元组 (l_i, r_i)，此时满足 $1 \le l_i \le r_i \le k$。因此，可以先使用桶排序将 q 个询问按照 r_i 升序排序。

[5]代码可在 https://github.com/shiliangzhi/ioi2021-thesis 查看

接着需要解决的问题是查询每个询问所对应的区间 $[l_i, r_i]$ 中数的最大值。这部分可以维护一个由 $[1, r_i]$ 中数据组成的单调递减队列，单调队列中下标大于等于 l_i 的第一个元素就是区间 $[l_i, r_i]$ 中的最大值。

如果使用二分查找则总时间复杂度为 $O(k + q \log k)$，这个时间复杂度和直接使用区域树并没有区别。不过注意到单调队列只有在删除元素和在末尾插入元素两种操作，所以该问题可以使用并查集来优化，这样一来复杂度就下降到 $O(k + q\alpha(k))$。

如果把所有问题叠加起来，总的时间复杂度为 $O(n \log n + m\alpha(n) \log n)$，该复杂度略小于 $O((n + m) \log^2 n)$。

3.3 总结

通过该问题的解决方法，可以得到分散层叠算法在处理问题的特点：对一个二维询问，将其分解为 $O(\log n)$ 个一维询问。对于每个一维询问，如果可以通过预处理或离线处理等方法使处理每一个问题的时间复杂度低于 $O(\log n)$ ，则使用分散层叠算法解决该问题可以使其时间复杂度小于 $O(m \log^2 n)$。

4 分散层叠在信息学竞赛中的应用

该算法在处理二维问题上是一个非常有利的工具，但是在信息学竞赛中，由于数据范围过小、算法常数过大、适用条件过于严格等因素，很少出现此类算法的题目。本文中将会选择三道与信息学竞赛联系较为密切的例题来说明分散层叠如何在信息学竞赛中应用。

4.1 例题一

题目大意

平面上有 n 个点，第 i 个点的坐标为 (x_i, y_i)。接下来会在线的提出 m 个询问，每个询问会给出一个坐标 (X, Y)，并询问 n 个点中与这个点距离最近的点的距离。本题中的距离为曼哈顿距离。

数据范围

$n, m \leq 5 \times 10^5$

限制

时间限制：4s　空间限制：192MB

解题思路

本题是经典的在线问题的一个变种。观察题目，注意到 $n, m \leq 5 \times 10^5$ 但时限只有 4s，因此应该需要一个 $O((n+m)\log n)$ 的算法。

每次询问可以按照询问点 (x, y) 为原点将整个平面分成四份，分别为询问点的左上区域、左下区域、右上区域和右下区域。由于:

$$\begin{aligned} \mathrm{dist}((x,y),(x_1,y_1)) &= |x-x_1|+|y-y_1| &(1)\\ &= \max\{x,x_1\}+\max\{y,y_1\}-\min\{x,x_1\}-\min\{y,y_1\} &(2)\end{aligned}$$

因此对于四个区域中的点可以分别设立权值 w_i 为 $-x+y, -x-y, x+y, x-y$，此时问题转换为求矩形内的点权值的最小值。

如果使用 KD-Tree 算法，最后的时间复杂度为 $O(m\sqrt{n})$，并不被允许。

如果使用分散层叠算法，则至少需要记 2 个前缀最小值，2 个后缀最小值，2 种指针，因此空间复杂度至少为 $O(6n\log n)$。此算法无法在题目给定的空间限制下通过。

然而，对于曼哈顿距离还有一种常见的处理方法，就是把所有点 (x, y) 都变化为 $(x+y, x-y)$，此时原来两点的曼哈顿距离和现在两点的切比雪夫距离相同。

定理 4.1. 点 (x_1, y_1) 与点 (x_2, y_2) 的曼哈顿距离和点 (x_1+y_1, x_1-y_1) 与点 (x_2+y_2, x_2-y_2) 的切比雪夫距离相等。

证明. 注意到：

$$\begin{aligned} |x_1-x_2|+|y_1-y_2| &= \max\{x_1-x_2, x_2-x_1\}+\max\{y_1-y_2, y_2-y_1\}\\ &= \max\{x_1-x_2+y_1-y_2, x_1-x_2+y_2-y_1, x_2-x_1+y_1-y_2, x_2-x_1+y_2-y_1\}\\ &= \max\{\max\{x_1+y_1-x_2-y_2, x_2+y_2-x_1-y_1\},\\ &\qquad \max\{x_1-y_1-x_2+y_2, x_2-y_2-x_1+y_1\}\}\\ &= \max\{|(x_1+y_1)-(x_2+y_2)|, |(x_1-y_1)-(x_2-y_2)|\}\end{aligned}$$

故定理 4.1 得证。 □

因此，问题转换为求切比雪夫距离的最小值。

如果采用二分加判断矩形内是否存在点的方法，即使使用分散层叠算法，复杂度也会退化到 $O(m\log^2 n)$，这是不被允许的。

不过事实上，可以在查询的同时二分。

已知询问点为 (x, y)，设在查找时第一维确定在 $[L, R]$ 的范围上，且满足 $x \notin [L, R]$，即 $x < L \leq R$ 或 $L \leq R < x$。为了方便，这里只讨论 $x < L \leq R$，对于 $L \leq R < x$ 的情况仅为对称情况，请读者自证。

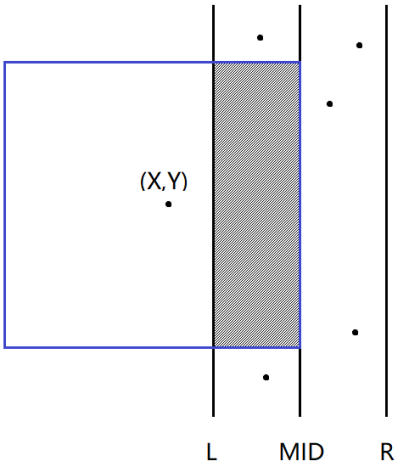


图 7: 例题一

如图，图中蓝色边框是以 (x, y) 为中心，$2(\text{MID} - x)$ 为边长的正方形。先考虑第一维在 $[\text{L}, \text{MID}]$ 中的点。不难得出，第一维在 $[\text{L}, \text{MID}]$ 并且第二维不在 $[x + y - \text{MID}, y + \text{MID} - x]$ 中的点（即位于阴影区域外的点）的切比雪夫距离都是由第二维决定的。因此，如果阴影区域内不存在点，$[\text{L}, \text{MID}]$ 就没有递归的必要了，直接计算完答案后递归 $[\text{MID}, \text{R}]$ 。

反之，如果阴影区域内存在点，这个点的切比雪夫距离距离一定小于等于 $\text{MID} - x$，这就意味着 $[\text{MID}, \text{R}]$ 中的点都不可能贡献答案（$[\text{MID}, \text{R}]$ 中的点的切比雪夫距离都大于等于 $\text{MID} - x$），此时直接递归 $[\text{L}, \text{MID}]$ 即可。

因此问题的关键就是快速判断阴影中是否存在点，以及如果阴影中不存在点，在阴影部分上下第二维离 y 最近的点是什么。

显然此时可以直接用分散层叠算法找到第二维坐标比 y 大的第一个点，这样可以解决上述两个问题。

这种算法除了指针和点的编号外其他都不需要存储，空间复杂度为 $O(3n \log n)$，时间复杂度为 $O(m \log n)$，可以通过本题。

4.2 例题二

题目大意

给定一棵大小为 n 的有根树，点从 $1 \sim n$ 编号，树根为 1 号点。二维平面上有 m 个点，第 i 个点的坐标为 (x_i, y_i) ，其代表树上的一个点 s_i。接下来将有 q 个询问，每个询问为查询一个左下角为 (x_1, y_1) ，右上角为 (x_2, y_2) 的矩形中所有点的最近公共祖先。

数据范围

$n, q \le 5 \times 10^5, m \le 5 \times 10^4$

限制

时间限制：5s 空间限制：1024MB

解题思路

本题没有强制在线，因此本题既可以使用在线的方法解决，也可以使用离线的方法解决。

对于在线算法，本题如果直接使用分散层叠，这样不方便直接求出区间 $[L, R]$ 中所有点的最近公共祖先。然而 m 只有 5×10^4 ，可以支持 $O(m \log^2 m)$ 的算法。因此可以先对外层线段树对应的每一个节点中的数据做一个 RMQ 预处理。这样在询问时就可以做到 $O(1)$ 了。时间复杂度为 $O((q+n)\log n + m \log^2 n)$。

对于离线算法，可以直接套用问题 2 的解决方案。先将询问离线，然后将询问按照 r_i 排序，最后使用单调队列加上并查集，时间复杂度为 $O((q+n)\alpha(m)\log n)$。不过该算法的理论复杂度并不如在线算法。

4.3 例题三：2020 北大集训 树数术

题目大意

给定一棵大小为 n 的有根树，点从 $1 \sim n$ 编号，树根为 1 号点。有一个长度为 m 的整数序列 a，$1 \le a_i \le n$。接下来有 q 个询问，每个询问会给出基于序列 a 的 k 段区间，并将这 k 段区间拼接成一个新的序列 b ，然后查询序列 b 中有多少个位置 i 满足对于所有 $1 \le j \le i$，都有 $\mathrm{lca}(b_j, b_i) = b_i$。

数据范围

$n, q, \sum k \le 7 \times 10^5$

限制

时间限制：4s 空间限制：1024MB

解题思路

本道题是经典离线问题的变种。

首先可以对问题进行一个简单的转换，不难发现对于每一个区间都相当于给定 l, r 和树上的一个点 x，求区间中有多少个 i，满足对于任意 $l \le j \le i$，都有 $\mathrm{lca}(a_j, a_i) = a_i$ 并且 $\mathrm{lca}(x, a_i) = a_i$。

先对序列中的每一个 i 都求一个数 c_i，c_i 为满足对于任意的 $L \le j \le i$ 都有 $\mathrm{lca}(a_j, a_i) = a_i$ 中最小的 L。不妨把每个 i 都变成一个点 (i, c_i)，这样一来就变成求有多少个点满足 $l \le i \le r, c_i \le l, \mathrm{lca}(a_i, x) = a_i$。

定理 4.2. 假设 h 为满足 $l \le i \le r, c_i \le l$ 的所有 i 从小到大的序列，则一定满足 $\mathrm{lca}(a_{h_i}, a_{h_{i+1}}) = a_{h_{i+1}}$ 和 $c_{h_{i+1}} \le c_{h_i}$。

证明比较显然，请读者自证。

因此，如果把所有点 (i, c_i) 构建区域树（每个节点上的数据按照 c_i 从小到大排序），在第一维对应的某个合法节点上（即该节点对应的区间被 $[l, r]$ 包含），其中合法的点（即 $c_i \le l$）一定满足前一个是后一个的祖先。同时，在这些点中满足是 x 的祖先的点一定是这些点的一个前缀。

此时如果使用分散层叠算法，可以得到一个 $O(\sum k \log^2 n)$ 的算法。即对于每一个询问，查出对应节点和该节点中对应的 $c_i \le l$ 的点后，再在这些点上二分一下满足是 x 祖先的点。

事实上这个算法还可以优化。因为这些合法点构成的虚树一定是一条链，不妨记链的叶子为 y，可以先把 x 变成 $\mathrm{lca}(x, y)$，这样并不影响查询的正确性。然后把所有询问离线，使用桶排序让这些 $\mathrm{lca}(x, y)$ 按照深度从深到浅排序，此时再询问就不需要二分，直接在每一个节点上维护一个指针表示在该节点上满足深度小于等于 $\mathrm{lca}(x, y)$ 所对应的前缀。由于 $\mathrm{lca}(x, y)$ 只会从深到浅，所以前缀只会缩短，指针只会向前移动，因此这部分复杂度是线性的。这样一来总时间复杂度就变成 $O((\sum k + n) \log n)$ 的了。

4.4 总结

在本节中，笔者通过三道例题，总结了分散层叠算法优化区域树在信息学竞赛中出现的特点。

笔者认为，由于分散层叠算法本身对查询方式由极为特殊的要求，因此大多数问题都可以转化为经典的在线问题和经典的离线问题。即该问题的数据有可以预处理的特点，或该问题允许离线询问，并且在离线后总能按照某种顺序，用极短的时间回答每一个询问。

不过，因为笔者水平有限，不知道此类问题能否有其他出现方式。因此，笔者希望本篇论文能起到抛砖引玉的作用，引发更多人来研究此类算法。

5 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢国家集训队高闻远教练的指导。
感谢曾艺卿老师对我的培养与教导。
感谢父母对我的理解与支持。
感谢林荣恩同学、林弘扬同学、曾行周同学、任舍予同学对本文的帮助。

参考文献

[1] 邓俊辉, 《数据结构（C++ 语言版）（第 3 版）》, 清华大学出版社。

[2] 蒋明润, 《浅谈利用分散层叠算法对经典分块问题的优化》, IOI2020 中国国家候选队论文集。

[3] Bernard Chazelle and Leonidas J. Guibas, “Fractional Cascading: II. Applications”.