浅析一些维护二维点的算法

中山纪念中学 林立

摘 要

本文研究的是有矩形修改和矩形查询的维护二维点的经典问题，分析了三种不同的解法：$2-d$ 树、定期重构、平面分块，比较它们的理论复杂度以及实际测试中的耗时。并研究了该问题的强制在线动态加点版本，分析了两种不同的解法：$2-d$ 树、平面分块，比较它们的理论复杂度。虽然平面分块比 $2-d$ 树的理论时间复杂度要劣，但是实验中运行时间前者优于后者。

1 引言

本文第二节介绍了一道维护二维点的经典问题，第三、四、五节分别介绍了对于该经典问题的 $2-d$ 树解法、定期重构解法、平面分块解法，在第六节中比较这三种解法的优劣并将其拓展到 k 维，同时研究了有其他不同操作类型的维护二维点问题。第七节介绍了经典问题的强制在线动态加点版本及解法，并比较了几种方法的优劣。

2 经典问题

本节介绍一道经典维护二维点问题，在接下来的三节中将介绍三种不同的解法：$2-d$ 树、定期重构、平面分块。

2.1 问题描述

二维平面上有 n 个点，第 i 个点是 (i,p_i)，初始权值是 w_i，其中 p 是一个大小为 n 的排列。

一共要进行 m 次操作，操作有以下两种：

1 l r d u k b 表示将在左下角是 (l,d) 右上角是 (r,u) 这个矩形内的点的权值乘 k 加 b。

2 l r d u 表示询问左下角是 (l,d) 右上角是 (r,u) 这个矩形内的点的权值和对 2^{64} 取模。

$1 \le n,m \le 10^5$

3 $2-d$ 树做法

这道经典问题可以使用 $2-d$ 树来解决。

接下来就先介绍 $2-d$ 树，然后再讲述如何使用 $2-d$ 树解决本题。

3.1 $2-d$ 树简介

$2-d$ 树就是 $k-d$ 树在维护二维点时的名称。

$2-d$ 树就是一棵二叉树，其中每个叶子节点都是 2 维点。而每个非叶子节点视为隐式地将一个二维平面切分为两半。这个非叶节点的左子树表示左半平面上的点，而右子树表示右半平面上的点。

3.2 $2-d$ 树的创建方法

有许多种建 $2-d$ 树的方法，这里选取了一种相对常见的方法：

每次都是给出一个点集，然后建出这个点集的 $2-d$ 树，一开始将 n 个点的点集递归下去，每递归到一个点集，就按如下方法顺序判断：

- 当点集中没有点时，建出一棵空的 $2-d$ 树，直接返回。
- 当点集中只有一个点时，建出一棵只有该点的 $2-d$ 树，直接返回。
- 当点集有中超过一个点时，需要选取一维来切分点集，而维度是循环地选择来切分[1]，这里假定选择了用 x 坐标。

  取出这个点集中 x 坐标是中位数的点[2]，接着将这个点作为切分点，将点集中的点分为 x 坐标小于切分点的和大于的。然后将小于的和大于的点集分别递归下去建出对应的 $2-d$ 树，并且将这两棵 $2-d$ 树分别设为切分点的左子树与右子树。

而其中选择中位数的点中 $C++$ 中可以使用 *nth_element* 函数。

对于一段长度为 n 的数组使用 *nth_element* 函数是 $O(n)$ 的时间，所以创建 $2-d$ 树的时间复杂度是 $T(n)=2T(n/2)+O(n)=O(n\log n)$。

3.3 $2-d$ 树上的修改和查询

由于有矩形修改与矩形查询，$2-d$ 树上每个点需要多维护一些信息。

每个点需要维护最小包含该点整个子树的矩形，子树内的信息和以及懒标记。

为了方便叙述，以下的“操作矩形”都是指当前的修改或询问的矩形，$2-d$ 树上的一个点的“最小包含矩形”是指最小的能够包含这个点子树内所有点的矩形。

[1]例如父亲节点选择了用 x 坐标，那么这次选择用 y 坐标，否则选择用 x 坐标

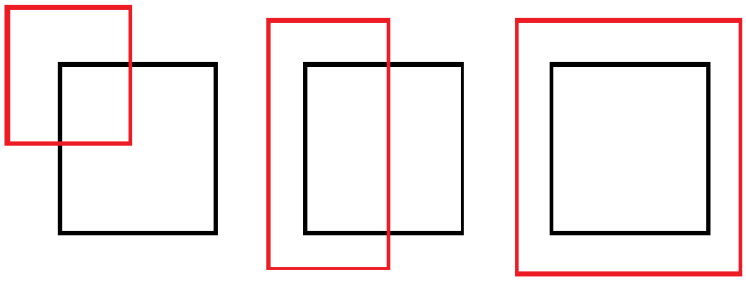
[2]如果有多个相同，将 y 坐标作为第二关键字

而矩形修改/询问时，从 $2-d$ 树的根节点开始递归，每递归到一个点，就按如下方法顺序判断：

- 如果最小包含矩形与操作矩形不交，直接退出
- 如果操作矩形包含最小包含矩形，那么直接打上标记/询问，然后退出
- 递归进左子树与右子树

注意需要下传标记，以及判断当前节点是否也在操作矩形内。

3.4 $2-d$ 树上修改和查询操作的时间复杂度分析



如图所示将对应的操作矩形如上命名，其中红色矩形代表操作矩形，而黑色矩形代表当前递归到的 $2-d$ 树上的点的最小包含矩形。

考虑创建 $2-d$ 树时是先用 x 坐标切分，最坏情况下操作矩形与第一次切分线和第二次切分线都有交，那么这样就会分成四个 1 号操作矩形。

1 号操作矩形，最坏情况下两次切分会分成一个 1 号操作矩形，一个 3 号操作矩形以及两个 2 号操作矩形。

2 号操作矩形，最坏情况下两次切分只会分成两个 2 号操作矩形和 3 号操作矩形。

3 号操作矩形，是可以直接 $O(1)$ 打标记/询问。

令 $T(n)$ 为对大小为 n 的 $2-d$ 树进行一次 1 号矩形操作时间复杂度，$T_2(n)$ 为进行一次 2 号矩形操作的时间复杂度。

根据上述最坏情况下的切分方法，有

$$\begin{aligned} T_2(n) &= 2T_2(n/4) + O(1) \\ &= \sum_{i=0}^{\lfloor \log_4(n) \rfloor} O(2^i) \\ &= O(\sqrt{n}) \end{aligned}$$

$$\begin{aligned} T(n) &= T(n/4) + 2T_2(n/4) + O(1) \\ &= T(n/4) + O(\sqrt{n}) \\ &= \sum_{i=0}^{\lfloor \log_4(n) \rfloor} O(\sqrt{\frac{n}{4^i}}) \\ &= \sum_{i=0}^{\lfloor \log_4(n) \rfloor} O(\frac{\sqrt{n}}{2^i}) \\ &= O(\sqrt{n}) \end{aligned}$$

综上所述一次矩形操作的时间复杂度为 $O(\sqrt{n})$。

3.5 $2-d$ 树做法

因为标记是可以被表示成 $kx+b$ 这样的形式，并且标记满足有结合律且能 $O(1)$ 计算，所以建出 $2-d$ 树后按照 3.2 节所述方法做即可。

时间复杂度 $O(n \log n + m\sqrt{n})$。

4 定期重构

设阈值为 B，考虑每 B 个操作就定期重构一次。

把 n 个点按照 B 个操作矩形的坐标分块，这样每个块对于这 B 个操作都是完全在操作矩形内或者完全不在操作矩形。对于每个块记录块内点的信息和以及懒标记，然后对于每个操作就直接打标记/询问其包含的所有块。由于块数最多为 $4B^2$，所以每个操作的复杂度为 $O(B^2)$。

B 个操作操作完之后，再将每个块的懒标记下传给对应的点上。

时间复杂度 $O(n + \frac{mn}{B} + mB^2)$。

当 B 取 $n^{1/3}$ 时最优，此时时间复杂度为 $O(n + mn^{2/3})$。

5 平面分块

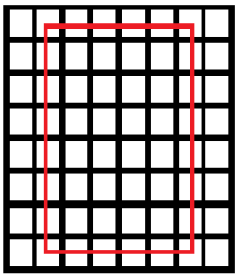


图 1. 一次矩形操作示意图

设阈值为 B，考虑直接将 $n \times n$ 的平面分成 $B \times B$ 块，每个块都维护该块内的点的信息和以及懒标记。

如图 1 所示，其中红色矩形为操作矩形，黑色小矩形为平面分块后的每一块，每次操作都会包含一些完整的块以及一些不完整的块，对应的做法如下：

- 对于被完整包含的块，直接打标记/询问，由于总块数是 B^2，所以这里最多对 B^2 个块进行操作，这里复杂度为 $O(B^2)$。
- 对于不被完整包含的块，枚举每个块中的点，如果在操作矩形内, 那么就做对应的操作。从图 1 可以看出这些块最多同属于两行整块与两列整块，又因为一行或者一列只有一个点，所以最多有 $4\frac{n}{B}$ 个点在这些块中，这里的复杂度为 $O(\frac{n}{B})$。

综上所述，总时间复杂度 $O(n + m(B^2 + \frac{n}{B}))$。

当 B 取 $n^{1/3}$ 时最优，此时时间复杂度为 $O(n + mn^{2/3})$。

6 算法小结

6.1 实验结果

对于 $2-d$ 树、定期重构、平面分块三种不同的解法，构造了不同类型的数据，测试了它们在这些数据下的运行时间。

	随机数据	针对 $2-d$ 树	针对平面分块	理论复杂度
$2-d$ 树	$1.809s$	$4.125s$	$4.001s$	$O(n \log n + m\sqrt{n})$
定期重构	$1.834s$	$2.028s$	$1.993s$	$O(n + mn^{2/3})$
平面分块	$0.628s$	$1.734s$	$1.853s$	$O(n + mn^{2/3})$

不同数据下不同做法的运行时间（数据规模：$n = m = 10^5$）

在随机数据下平面分块算法远优于 $2-d$ 树，而定期重构因为重构的时间消耗稳定且较多导致随机数据下与 $2-d$ 树的运行时间差不多。

在针对 $2-d$ 树的数据下，$2-d$ 树的极大常数的显现出来，尽管 $2-d$ 树的理论时间复杂度是 $O(n\log n+m\sqrt{n})$，其在实际测试中需要足足 $4s$，远远大于定期重构和平面分块解法的运行时间。

根据表格可以发现在 $n=m=10^5$ 的绝大多数情况下，平面分块算法都比 $2-d$ 树要优。

6.2 拓展性

6.2.1 拓展到 k 维

拓展到 k 维，三种解法都可行。

用 $2-d$ 树变成用 $k-d$ 树，而 $k-d$ 树就是在选择切分维度时从两维轮换变成 k 维轮换，维护方法和时间复杂度分析差不多，时间复杂度为 $O(n\log n+mn^{1-1/k})$。

定期重构就是每 $n^{1/(k+1)}$ 操作重构一次，维护方法和时间复杂度分析与二维差不多，时间复杂度为 $O(n+mn^{1-1/(k+1)})$。

平面分块就变成 k 维空间分块，就是每一位都分成 $n^{1/(k+1)}$ 块，维护方法和时间复杂度分析与二维差不多，时间复杂度为 $O(n+mn^{1-1/(k+1)})$。

由于 $k-d$ 树的大常数，在 k 维问题上应该还是 k 维空间分块算法实际时间上优秀。

6.2.2 强制在线

$2-d$ 树和平面分块解法都可行，定期重构就无法解决强制在线的一类问题。

在第七节介绍了该问题的强制在线动态加点的版本，$2-d$ 树和平面分块都可以解决该版本的问题，并且主要复杂度没有变化。

6.2.3 矩形最值操作

问题描述如下：

二维平面上有 n 个点，第 i 个点是 (i,p_i)，初始权值是 w_i，其中 p 是一个大小为 n 的排列。

一共要进行 m 次操作，操作有以下两种：

1 l r d u x 表示将在左下角是 (l,d) 右上角是 (r,u) 这个矩形内的点的权值与 x 取 max。

2 l r d u 表示询问左下角是 (l,d) 右上角是 (r,u) 这个矩形内的点的权值和对 2^{64} 取模。

$2-d$ 树做法：由于 $2-d$ 树的形态与线段树差不多，所以可以使用参考文献 [2] 中的方法，时间复杂度 $O(n\log n+m\sqrt{n})$。

定期重构做法：

- 重构后每个块内的点需要按照权值大小排序，要维护每个块的最小值，最小值的个数和非最小值总和。
- 每次矩形最值时，对于每个涉及到的块，修改其的最小值，当操作的 x 大于等于该块的严格次小值时，该块的最小值个数和非最小值总和就会改变，因为块内的点已经按权值排好序了，所以可以每个块用个指针指向严格次小值的位置，每次修改就判断是否要移动该指针。指针只会朝一个方向移动，每个块的变化量最多为块内点数，那么变化总量最多为 n。
- 询问矩形和就直接枚举被包含的块求和即可。

其中排序部分可以使用以下方法：

- 使用快排，那么时间复杂度为 $O(n+m(n\log n)^{2/3})$。
- 由于总共最多只有 $n+m$ 不同的值，可以一开始离散，然后排序时就可以使用多关键字桶排，时间复杂度是 $O(n+m(n\log_n(n+m))^{2/3})$。
- 直接使用多关键字桶排，时间复杂度是 $O(n+m(n\log_n maxw)^{2/3})$。

平面分块做法：每块维护的东西都与定期重构做法一样，所以只用考虑不被修改矩形完整包含的块。

对于这种块只用考虑一次修改的影响对于权值顺序的影响，因为其他的东西都可以在块内点数常数倍时间内解决。由于修改只是取 max，所以可以将被修改的点在权值顺序去掉，然后再加入进去，这样可以在块内点数常数倍时间内解决。

每次修改不完整的块总共能变化量只会多出 $O(n^{2/3})$，所以时间复杂度不变，仍是 $O(n+mn^{2/3})$。

6.2.4 矩形最值 + 矩形加

问题描述如下：

二维平面上有 n 个点，第 i 个点是 (i,p_i)，初始权值是 w_i，其中 p 是一个大小为 n 的排列。

一共要进行 m 次操作，操作有以下三种：

1 l r d u x 表示将在左下角是 (l,d) 右上角是 (r,u) 这个矩形内的点的权值与 x 取 max。

2 l r d u x 表示将在左下角是 (l,d) 右上角是 (r,u) 这个矩形内的点的权值加上 x。

3 l r d u 表示询问左下角是 (l,d) 右上角是 (r,u) 这个矩形内的点的权值和对 2^{64} 取模。

$2-d$ **树**做法：由于 $2-d$ 树的形态与线段树差不多，所以可以使用参考文献 [2] 中的方法，时间复杂度 $O(n\log n+m\sqrt{n}\log n)$，但是根据参考文献 [2] 中所述暂时无法构造能够达到该复杂度的数据。

定期重构做法：同 6.2.3 中所述，不过由于多了矩形加，所以排列只能用多关键字桶排，时间复杂度 $O(n+m(n\log_n maxw)^{2/3})$。

平面分块做法：同 6.2.3 中所述，不过需要多处理矩形加时不被完整包含的块，由于本身每个块都维护了该块内点的权值顺序，每次矩形加就将被修改的点按权值顺序提出来，那么两个有序的序列就能直接归并起来就能得到矩形加后的权值顺序序列。

时间复杂度不变，仍是 $O(n+mn^{2/3})$。

7 经典问题 2

比经典问题 1 多了动态加点操作以及强制在线。

7.1 问题描述

一共要进行 n 次加点和 n 次操作，第 i 次操作在第 i 次加点后，强制在线。

操作有以下两种：

1 l r d u k b 表示将在左下角是 (l,d) 右上角是 (r,u) 这个矩形内的点的权值乘 k 加 b。

2 l r d u 表示询问左下角是 (l,d) 右上角是 (r,u) 这个矩形内的点的权值和。

$1\le n\le 10^5$

7.2 $2-d$ 树 + 二进制分组

由于多了强制在线，就不能够直接预先建好 $2-d$ 树，此时考虑使用二进制分组。

我们将一个数拆分成 2 的次幂从大到小的形式。示例：

$$\begin{aligned}21 &= 16+4+1\\22 &= 16+4+2\\23 &= 16+4+2+1\\24 &= 16+8\end{aligned}$$

我们不妨用上述拆分方法对于加点操作分组，比如说，对于前 21 个加的点，我们会将前 16 个点分为第一组，第 17~20 的点分为第二组，第 21 个点单独作

为第 3 组。我们对于每组都使用 $2-d$ 树来维护。显然只会分出最多 $O(\log n)$ 组，修改和询问就直接对于每一组都做一遍，而最坏情况下是每一个 2 的次幂都有，此时时间复杂度为 $O(\sum_{i=0}^{\lfloor \log_2 n \rfloor} O(\sqrt{2^i}) = O(\sqrt{n})$。

而多加一个点时怎么做呢？事实上，我们只用考虑增加一个点之后原来的分组与新的分组的区别，而显然是一些分组和新加入的点合并成了一个新的分组。那么我们只需要将那一些不存在了的分组下传 $2-d$ 树上的标记，得到每个点当前真实的权值，然后将这些点和新的的点再建出一棵 $2-d$ 树。

我们来分析一下时间复杂度。我们考虑我们重建的所有组的元素总数。可以发现，当加入第 i 个点时，重建的元素个数是 $lowbit(i)$，也就是 i 的二进制表示下最右侧的 1 所代表的权值[3]。

可以推出重建需要的时间为：

$$\begin{aligned} & \sum_{i=1}^{n} O(lowbit(i) \log lowbit(i)) \\ = & \sum_{i=0}^{\lfloor \log_2 n \rfloor} \left\lfloor \frac{n}{2^{i+1}+0.5} \right\rfloor \times O(2^i \times i) \\ = & \sum_{i=0}^{\lfloor \log_2 n \rfloor} O(n \times i) = O(n \log^2 n) \end{aligned}$$

最后总的时间复杂度为 $O(n \log^2 n + n\sqrt{n})$。

7.3 平面分块

由于多了强制在线和动态加点，不能够直接将平面切分好，所以考虑动态维护平面的切分。

我们考虑每一次加入一个点，就是在两维坐标各加入当前点的坐标的两维，然后才能将这个点加入到其对应的整块中。那么两维各维护一个块状链表，每次将一个点加入到对应的块上，如果当前块的大小等于阈值，那么将这个块均匀分裂成两半，对应的将平面上的整块分裂成两块。

如果我们将阈值设为 $O(n^{2/3})$，那么只会发生 $O(n^{1/3})$ 次分裂，每次分裂操作只会涉及该块内 $O(n^{2/3})$ 个点以及 $O(n^{1/3})$ 个平面上整块，所以分裂的总复杂度是 $O(n)$。而修改与查询操作与沿用第五节的做法即可。

总时间复杂度为 $O(n^{5/3})$。

7.4 算法小结

即使是在强制在线 + 动态加点的问题上，平面分块算法依旧比使用 $2-d$ 树时间上优秀。

[3]示例：$24 = (11000)_2$, 所以 $lowbit(24) = (1000)_2 = 8$

8 总结

本文围绕一个维护二维点的经典问题，分析三种不同的解法：$2-d$ 树、定期重构、平面分块，比较了它们的理论时间复杂度和在试验中运行时间。

$2-d$ 树虽然是一种常见的数据结构，但是因为其极大的常数为人诟病。而平面分块算法能够在维护二维点集上优于 $2-d$ 树，即使理论复杂度更大，但因为常数更小也不失为一种好的做法。希望对读者有所帮助。

9 感谢

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。
感谢中山纪念中学的宋新波老师，熊超老师多年来给予的关心与帮助。
感谢父母对我的理解与支持。
感谢高嘉煊同学对本文的帮助。

参考文献

[1] k-d tree 的维基百科，https://en.wikipedia.org/wiki/K-d_tree

[2] 吉如一，《区间最值操作与历史最值问题》，2016 年信息学奥林匹克中国国家队候选队员论文集

[3] 许昊然，《浅谈数据结构题的几个非经典解法》，2013 年信息学奥林匹克中国国家队候选队员论文集