浅谈利用分散层叠算法对经典分块问题的优化

成都七中 蒋明润

摘 要

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。本文介绍了这个算法对经典分块题的优化并展示了它的一些应用。

前言

分散层叠算法已经产生了 20 多年，但是尚未在算法竞赛界得到广泛应用。本文试图利用该算法对经典分块问题进行优化，希望在算法竞赛中引入分散层叠算法。本文第一节介绍了分散层叠算法及其意义，第二节介绍经典分块题及其常用解法，第三节则利用分散层叠算法对经典分块题进行了算法优化，并与传统算法进行了比较，同时对经典题进行了扩展。

1 分散层叠

本章介绍了分散层叠（Fractional Cascading）算法。并将通过一个具体的例子来说明该算法的含义。

1.1 简介

在计算机科学中，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，其中第一次查找是普通的二分查找，时间复杂度为 $O(\log n)$ ，后面的二分查找会比第一次更快。分散层叠最早由 Chazelle 和 Guibas 在 1986 年发表的两篇文章中[1, 2] 提出。顾名思义，Fractional 是指“零碎的、微小的”，“Cascade”则是一个比喻，指“像小瀑布一样层叠的”。 Chazelle 和 Guibas 原文给出了一幅图画，如图 1所示，它相当形象地揭示了这个名词背后的深刻含义：像瀑布一样从高到低逐渐分散成越来越细的支流，再层层叠叠地覆盖[3, 4]。

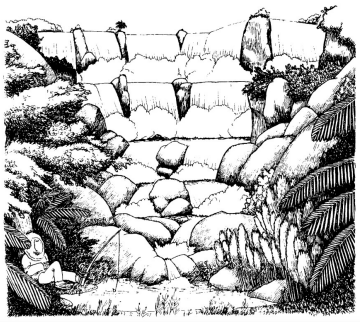

图 1: Chazelle和Guibas原文给出的图画

1.2 举例说明

我们将用例题的形式来描述这个算法。

考虑下面这个问题：给出长度和为 n 的 k 个有序数列，第 i 个数列记作 L_i，要求建立一个数据结构，支持对同一个数 q 在 k 个数列中分别进行二分查找（查询第一个大于等于 q 的数的位置），例如下面这个例子中， $k = 4$， $n = 17$。

$$L_1 = 24, 64, 65, 80, 93$$

$$L_2 = 23, 25, 26$$

$$L_3 = 13, 44, 62, 66$$

$$L_4 = 11, 35, 46, 79, 81$$

最简单的方法是把每个序列分开存放。如果我们这么做，我们只需要 $O(n)$ 的空间复杂度。但是查询时需要对每个序列分别进行一次二分查找，在最坏情况下，k 个序列长度相等，需要 $O(k \log(\frac{n}{k}))$ 的时间复杂度。

第二种方法可以支持更快的查询，但是需要消耗更多的空间：我们可以把这 k 个序列合并成一个大的序列 L，并对 L 中的每个元素记录下它在原来的 k 个序列中进行二分查找的结果。如果我们把合并后的序列中每个元素记作 $x[a, b, c, d]$，x 是数值，a, b, c, d 为 x 在原来的序列中的后继所在的位置（从 0 开始标号），那么：

$$\begin{aligned} L = & 11[0,0,0,0], 13[0,0,0,1], 23[0,0,1,1], 24[0,1,1,1], 25[1,1,1,1], 26[1,2,1,1], 35[1,3,1,1], \\ & 44[1,3,1,2], 46[1,3,2,2], 62[1,3,2,3], 64[1,3,3,3]65[2,3,3,3], 66[3,3,3,3], \\ & 79[3,3,4,3], 80[3,3,4,4], 81[4,3,4,4], 93[4,3,4,5] \end{aligned}$$

这种方法可以做到查询的时间复杂度为 $O(k + \log n)$：直接在 L 中进行二分查找，然后返回在元素 x 中记录的信息。但是它需要 $O(nk)$ 的空间。

分散层叠算法支持对同样的问题同时做到最优的时间复杂度和空间复杂度：查询的时间复杂度为 $O(k + \log n)$，空间复杂度为 $O(n)$。建立 k 个新的有序序列，第 i 个记作 M_i。其

中最后一个序列 M_k 和 L_k 相同。前面的每一个序列 M_i 由 L_i 和 M_{i+1} 的从第二个元素开始隔一个采样一个得到的子序列归并得到，并对 M_i 中每个元素记录下它在 L_i 和 M_{i+1} 中二分查找的结果。例如，对于前面给出的 4 个序列，我们有：

$$M_1 = 24[0,1], 25[1,1], 35[1,3], 64[1,5], 65[2,5], 79[3,5], 80[3,6], 93[4,6]$$

$$M_2 = 23[0,1], 25[1,1], 26[2,1], 35[3,1], 62[3,3], 79[3,5]$$

$$M_3 = 13[0,1], 35[1,1], 44[1,2], 62[2,3], 66[3,3], 79[4,3]$$

$$M_4 = 11[0,0], 35[1,0], 46[2,0], 79[3,0], 81[4,0]$$

如果我们想要对 $q = 50$ 进行查询，我们先在 M_1 中进行二分查找，得到 $64[1,5]$。“1”表示在 L_1 中二分查找的结果是 $L_1[1] = 64$，“5”表示在 M_2 中进行二分查找的结果大致在下标 5。准确地说，是在下标 5 的 $79[3,5]$ 或者前面一个位置的 $62[3,3]$ 。将 q 和 62 进行比较，得知正确的查找结果为 $62[3,3]$。通过相同的方法，可以得到 q 在 L_2 ， M_3 ， L_3 ， M_4 中二分查找的结果。

更一般地说，对于任何一个这样的结构，我们可以先在 M_1 中进行二分查找，然后对于任何一个 i ，我们可以通过 q 在 M_i 中的位置定位出 q 在 L_i 中的位置和 q 在 M_{i+1} 中的位置。 q 在 M_i 中查询的结果指向的 M_{i+1} 中的位置要么是 q 在 M_{i+1} 中查询的结果，要么只相差一个位置，所以可以通过一次比较确定。总时间复杂度为 $O(k + \log n)$ 。

在前面的例子中，四个新序列的长度总和是 25，不超过 $2n$。一般来说，M_i 的长度不超过 $|L_i| + \frac{1}{2}|L_{i+1}| + \frac{1}{4}|L_{i+2}| + \cdots$ 。所有M_i 的长度总和不超过 $\sum |M_i| \le \sum |L_i|(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots) = 2n = O(n)$。

1.3 普遍情况

考虑给出一张有向无环图，每个节点的入度和出度都不超过一个常数 d，每个节点上存有一个有序序列 L_u。一次查询需要对一个数 q 在一条路径上的所有节点 u 上存有的序列 L_u 中进行二分查找。对于前面的例子，给出的有向图是一条长度为 4 的链。

对每一个节点 u 建立一个新序列 M_u ， M_u 由 L_u 和 u 的所有后继 v 上建立的序列 M_v 按一定比例均匀选取元素得到的序列归并得到（如果要做到 $O(n)$ 的空间复杂度，选取的比例必须小于 $\frac{1}{d}$ ），并对每个元素记录下它在 L_u 和每个 M_v 中二分查找的结果。对于前面的例子，$d = 1$ ，选取的比例为 $\frac{1}{2}$。

对于一次查询，我们先在路径的起点 s 上建立的序列 M_s 中进行一次 $O(\log n)$ 的二分查找。假设我们知道 q 在 M_u 中二分查找的结果，要对于 u 的一个后继 v 求出 q 在 M_v 中二分查找的结果。如果我们建立数据结构时选取的比例是 $\frac{1}{r}$ ，那么 q 在 M_u 中二分查找的结果指向的 M_v 中的位置和 q 在 M_v 中二分查找的正确结果的距离不会超过 r。因为 r 是一个常数，所以可以在 $O(1)$ 的时间复杂度内求出 q 在 M_v 中二分查找的正确结果。

设查询的路径长度为 k，则单次查询的时间复杂度为 $O(k + \log n)$。

另外分散层叠可以支持 $O(\log n)$ 在一个序列中加入一个元素，但查询复杂度会变为$O(\log n+k\log\log n)$，不过与本文关系不大，这里略去。

2 经典分块题

本章介绍一道OI经典分块题及其传统解法。下一章我们将讨论利用分散层叠算法对这道题进行优化。

2.1 题目描述

给出一个长度为 n 的序列，有 n 次操作，操作分为两种：

1 修改操作：给定 l, r, x，对于所有 $l \le i \le r$，将 a_i 增加 x。

2 查询操作：给定 l, r, x，求出在所有 $l \le i \le r$ 中，有多少个 i 满足 $a_i \le x$。

2.2 常用解法（算法1）

将序列中每相邻的 B 个数分成一块，总共 $O(\frac{n}{B})$ 块。每个块内存下这个块中所有元素排序后的结果。这样对于一个区间 $[l, r]$，最多只有 2 个块被不完全覆盖，其它块要么是完全覆盖，要么没有被覆盖。

对于修改操作，不完全覆盖的块需要暴力重构。此时需要重新求出排序后的结果，如果暴力排序，时间复杂度为 $O(B\log B)$, 不过注意到对于两个都没有被修改的数或者对于两个都被修改的数，它们的大小关系是不会改变的，所以如果我们排序时记录下元素在排序前的位置，就可以 $O(B)$ 将没有被修改的数和被修改的数分别排序，然后归并即可，时间复杂度为 $O(B)$。完全覆盖的块可以记录一个懒标记，表示将块内所有元素全部加上一个数。每个块的时间复杂度为 $O(1)$, 总共 $O(\frac{n}{B})$ 个块，总时间复杂度为 $O(\frac{n}{B})$。

对于查询操作，不完全覆盖的块可以直接枚举所有元素进行查询，时间复杂度为 $O(B)$，完全覆盖的块需要在排序后的数组上二分，每个块的时间复杂度为 $O(\log B)$, 总时间复杂度为 $O(\frac{n\log B}{B})$。

综上，单次操作的时间复杂度为 $O(B+\frac{n\log B}{B})$, 总时间复杂度为 $O(nB+\frac{n^2\log B}{B})$, 取 B 为 $\sqrt{n\log n}$ 时时间复杂度最优，为 $O(n\sqrt{n\log n})$。

2.3 离线算法（算法2）

本节讨论离线算法。

在本题中只有以下输入范围能使用本节算法：输入的所有数均为绝对值不超过 2^ω 的整数，且 $\omega = O(\log n)$。

注意到上一节中算法的瓶颈在二分查找，本节考虑对这部分进行优化。

然而优化单次二分查找是很困难的（可以使用y-fast tree等数据结构，但不实用）。因此，要优化算法，需要从优化多次二分查找的总时间复杂度入手。其中有两种方法，第一种方法是优化多次询问在一个块中进行的二分查找，第二种方法是优化单次询问在多个块中进行的二分查找。本节讨论第一种方法，第二种方法将在下一章中讨论。

考虑预先存下所有询问，逐块处理。在对一个块进行重构时求出之前所有在这个块中进行的二分查找的结果。这样问题转化成在一个长度为 B 的序列中进行 Q 次二分查找。

注意到如果每次二分查找的数单调递增，可以利用单调性做到 $O(B+Q)$ 的时间复杂度。所以如果可以快速地将 Q 个数排序，就可以快速地求出 Q 次二分查找的结果。

在 $\omega = O(\log B)$ 时，可以以 B 为底进行基数排序，时间复杂度为 $O(\frac{(B+Q)\omega}{\log B}) = O(B+Q)$ 。这样，原分块题的时间复杂度可以优化至 $O(nB + \frac{n^2}{B})$ ，取 $B = \sqrt{n}$ 时最优，为 $O(n\sqrt{n})$ 。

3 基于分散层叠的算法

上一章介绍了OI经典分块题及其传统解法。本章我们将讨论利用分散层叠算法对这道题进行优化。

基数排序的做法可以做到 $O(n\sqrt{n})$ 的时间复杂度，然而如果值域不是整数，或者强制在线（读入一个操作之前必须求出之前所有询问的结果，所以不能预先存下所有询问），就不适用了。

单独的一次二分查找难以优化，不过注意到我们需要的不是单独的二分查找，而是对同一个数在 $O(\frac{n}{B})$ 个序列中进行二分查找，所以可以考虑使用分散层叠算法。然而分散层叠算法不支持修改，所以需要一些处理。

3.1 $O(n\sqrt{n\log\log n})$做法（算法3）

清华大学蔡承泽在笔者之前想出了时间复杂度为 $O(n\sqrt{n\log\log n})$ 的算法，但他并没有公开详细内容，本节中所述做法是笔者根据一些相关的讨论推断出的一种可能的做法，不知道是否与蔡学长做法一致。无论如何，在此对蔡承泽学长表示感谢。

在本算法中，建立分散层叠的方法为：在每一个块中均匀选取 $\frac{1}{D}$ 的元素建立分散层叠。这样在二分查找的时候可以根据分散层叠定位出一个距离相差不超过 D 的位置，从而对每个块分别做到 $O(\log D)$ 的时间复杂度。用此方法每连续 D 个块建立一组分散层叠，共 $O(\frac{n}{BD})$ 组。

执行修改操作的时候，至多只有 2 组分散层叠的结构会被破坏（别的分散层叠只可能对所有数加上同一个数，这不会改变分散层叠的结构）。由于每一个块只选取了 $O(\frac{B}{D})$ 个元素建立分散层叠，每组分散层叠只在 D 个块之间建立，所以重构一组分散层叠的时间复杂度为 $O(B)$ 。

对于查询操作，需要在每组分散层叠中进行一次查询，在一组分散层叠中查询的时间复杂度为 $O(D+\log B)$, 总共 $O(\frac{n}{BD})$ 组，总时间复杂度为 $O(\frac{n}{B}+\frac{n\log B}{BD})$, 另外对每一个块还要进行一次 $O(\log D)$ 的二分查找，总时间复杂度为 $O(\frac{n\log D}{B})$, 另外还要对不完全覆盖的块进行 $O(B)$ 的枚举。

综上，单次操作的时间复杂度可以优化至 $O(B+\frac{n\log D}{B}+\frac{n\log B}{BD})$，总时间复杂度为 $O(nB+\frac{n^2\log D}{B}+\frac{n^2\log B}{BD})$, 取 $B=\sqrt{n\log\log n}, D=\log n$ ，时间复杂度为 $O(n\sqrt{n\log\log n})$。

3.2 $O(n\sqrt{n})$ 做法（算法4）

如果我们采用1.2节中的合并方式，已经难以继续优化，接下来本小节将从另一种合并方式入手，对问题进行进一步的优化。

建立一棵有 $\frac{n}{B}$ 个叶节点的线段树，线段树的叶节点依次存放每个块中维护的排序后的序列，非叶节点存放的序列由两个子节点所存放的序列中分别按一定比例均匀选取元素得到的序列归并得到，并记录下每个元素在两个子节点所存的序列中二分查找的结果。（即在1.3节中，让线段树中每个非叶节点连向它的两个子节点形成有向无环图，叶节点的 L_u 为每个块中维护的序列，非叶节点的 L_u 为空序列得到的分散层叠结构）。

执行修改操作时，根据线段树的性质，只有 $O(\log\frac{n}{B})$ 个节点所对应的区间会被不完全覆盖。而如果一个节点对应的区间被完全覆盖，那么这个节点的子树中所有元素的数值会被加上一个数，但是结构不会改变。所以只有 $O(\log\frac{n}{B})$ 个节点需要重新计算存放的序列。当选取元素的比例为 $\frac{1}{2}$ 时，每个节点存放的序列长度都是 $O(B)$ ，总时间复杂度为 $O(B\log\frac{n}{B})$ ，不够优秀。但是，如果我们用更小的比例，比如 $\frac{1}{3}$ 时，线段树的一层中每个节点存放的序列长度都是下一层每个节点的 $\frac{2}{3}$, 而根据线段树的性质，每一层中只有 $O(1)$ 个节点会被不完全覆盖。所以总时间复杂度不超过 $O(B(1+\frac{2}{3}+\frac{4}{9}+\frac{8}{27}+\cdots))=O(B)$。

对于查询操作，与1.3节中类似，我们可以先在根节点中进行二分查找，然后根据在一个节点中二分查找的结果 $O(1)$ 确定出在两个子节点中二分查找的结果，总时间复杂度为 $O(\frac{n}{B}+\log n)$ ，另外还要对不完全覆盖的块进行 $O(B)$ 的枚举。

综上，单次操作的时间复杂度为 $O(B+\frac{n}{B}+\log n)$，总时间复杂度为 $O(nB+\frac{n^2}{B}+n\log n)$，取 $B=\sqrt{n}$ 时，时间复杂度最优，为 $O(n\sqrt{n})$ 。

3.3 算法小结

综上所述，各种算法的时间复杂度如表1所示，其中算法1为传统算法。算法2虽然也达到了优化的目的，但是只适用于部分输入情况。算法3和算法4都是利用分散层叠优化的算法，使用都不受限制，其中算法4在已知算法中最优。

表 1: 各种算法的比较

算法	时间复杂度	备注
算法 1	$O(n\sqrt{n\log n})$	
算法 2	$O(n\sqrt{n})$	只适用于部分输入情况
算法 3	$O(n\sqrt{n\log\log n})$	
算法 4	$O(n\sqrt{n})$	

3.4 算法应用扩展

分散层叠算法不仅可以应用于本道经典题，还可以扩展应用到一些其它题目中。

3.4.1 题目 1

给出一个长度为 n 的序列，有 n 次操作，操作分为两种：

1 修改操作：给定 l, r, x，对于所有 $l \le i \le r$，将 a_i 增加 x。

2 查询操作：给定 l, r, x，求出区间 $[l, r]$ 有多少个子区间 $[l', r']$，满足 $[l', r']$ 中所有数均不超过 x。[1]

注意到对于两个区间 $[l, mid]$ 和 $[mid + 1, r]$ ，如果我们知道它们的答案（这里的“答案”包括查询操作的答案，区间中第一个大于 x 的数的位置，区间中最后一个大于 x 的数的位置，下同），我们可以求出区间 $[l, r]$ 的答案。

将每相邻的 B 个数分成一块，共 $O(\frac{n}{B})$ 块。对于一次查询，可以将区间 $[l, r]$ 拆分成 $O(\frac{n}{B})$ 个长度不超过 B 的区间，其中只有 $O(1)$ 个区间不是完整的一块。对于不是完整的一块的区间，可以拆分成 $O(B)$ 个长度为 1 的区间处理，对于完整的一块，求出块中所有元素排序后的序列，并求出块中每一个元素作为 x 时这个块的答案。后者可以通过以下方法得到：

考虑按递增的顺序处理，将所有大于 x 的数按出现的位置顺序加入链表中（为方便处理，可以在链表两端加入特殊节点）。随着 x 的增大，链表中的元素将逐渐被删除。求这个块的答案时，求出区间中第一个大于 x 的数的位置和区间中最后一个大于 x 的数的位置可以直接通过特殊节点的指针得到，求查询操作的答案时需要考虑删除一个节点对答案的影响，显然所有包含这个节点而不包含这个节点的前驱和后继的所有区间将被加入答案，而不会有其它影响。（这一步的时间复杂度为 $O(B)$ ，所以不会增加对块进行重构的时间复杂度）

求出这个之后，剩下的部分和第二章中的经典分块题做法基本相同，可以用3.2节中的方法进行优化。

[1]本题改编自Comet OJ Contest #7 F https://cometoj.com/contest/52/problem/F?problem_id=2426

3.4.2 题目 2

给出一个长度为 n 的序列，有 n 次操作，操作分为两种：

1 修改操作：给定 l, r, x，对于所有 $l \le i \le r$，将 a_i 增加 x。

2 查询操作：给定 l, r, x，求出区间 $[l, r]$ 的所有子区间中，区间中所有元素的和最大是多少。

下面先讲述官方题解的做法。

注意到与上一题类似，对于两个区间 $[l, mid]$ 和 $[mid + 1, r]$, 如果我们知道它们的答案（此处“答案”包括区间最大子段和（即查询操作的答案），区间最大前缀和，区间最大后缀和），我们可以求出区间 $[l, r]$ 的答案。

仍然可以考虑每相邻的 B 个数分成一块，共 $O(\frac{n}{B})$ 块。关键在于如何处理完整的一块的情况。

对完整的一块的处理需要在整块中所有数全部加上同一个数的情况下维护答案，首先对这种情况进行处理。

考虑使用线段树进行维护。对于一个节点，需要维护这个节点所代表的区间中所有数加上同一个数 x 时这个区间的答案。不难证明这是一个关于 x 的分段一次函数，且段数与区间的长度同阶。一个节点上维护的信息可以由它的两个子节点合并得到，合并只需要将分段函数相加或者取 max ，可以在 $O(len)$（len 是分段函数的段数，与区间长度同阶）的时间复杂度内完成。这样建立线段树的时间复杂度为线段树中所有节点对应的区间长度之和，为 $O(B \log B)$。

但是对于原问题，我们这么做会有一些问题：第一个问题是执行修改操作时，需要对不完全覆盖的块进行重构。如果暴力重建整个线段树，则时间复杂度为 $O(B \log B)$ ，不够优秀。

注意到对一个块进行重构时，对这个块进行的修改仅仅是区间加，根据线段树的性质，在每一层中分别只有 $O(1)$ 个节点会被不完全覆盖，而被完全覆盖的节点所进行的修改仅仅是对分段函数进行平移，可以用懒标记维护。这样每一层中分别只有 $O(1)$ 个节点的信息需要重新计算，时间复杂度为 $O(B + \frac{B}{2} + \frac{B}{4} + \frac{B}{8} + \cdots) = O(B)$ 。

另外一个问题是查询一个块的答案时，需要通过二分查找求出答案在分段函数的哪一段。如果直接进行二分查找，则单次操作的复杂度为 $O(B + \frac{n \log B}{B})$，不够优秀。官方题解给出的做法是类似 2.3节中的方法进行优化，即基数排序然后利用单调性。时间复杂度为 $O(n\sqrt{n})$ 。由于进行了逐块处理，可以只在处理该块时建立该块的线段树，所以空间复杂度为 $O(B \log B + n) = O(n)$，而不是将每一块的线段树全部建立出来时的 $O(n \log n)$ 。

事实上，我们可以使用 3.2节中算法4 分散层叠法，其中叶节点所存储的序列为分段函数的分界点构成的序列。对于被完全覆盖的块，修改操作进行的修改仅仅是对分段函数进行平移，即将分界点全部加上同一个数。所以如果线段树中的节点对应的区间被完全覆

盖，这个节点的子树中所有节点维护的序列也只会被全部加上一个数，不会改变结构。所以 3.2节中做法仍然适用，时间复杂度为 $O(n\sqrt{n})$ ，空间复杂度为 $O(n\log n)$ 。

4 总结

分散层叠算法虽然已经有 20 多年的历史，是一种常见的算法，但目前在信息学竞赛中应用还不广泛。本文介绍了分散层叠算法，并将该算法和其他算法进行了对比，应用于信息学竞赛中的一些题目，特别是在解决这类经典分块问题中进行了算法优化。希望对大家有所帮助。

但这一领域研究者不多，限于笔者水平有限，很多问题尚未解决，希望本文能抛砖引玉，吸引更多的同仁来研究这个算法。

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。
感谢蔺洋老师、张君亮老师、胡凡老师、李植武老师、林鸿老师对我的培养与教导。
感谢蔡承泽学长、李欣隆学长等在讨论中给予我的灵感。
感谢王修涵、王思齐、袁方舟等学长的指导。
感谢成都七中高18级信息组全体同学两年来的陪伴。
感谢父母对我的理解与支持。

参考文献

[1] Chazelle, Bernard;Guibas, Leonidas J. Fractional cascading: I. A data structuring technique. 1986

[2] Chazelle, Bernard;Guibas, Leonidas J. Fractional cascading: II. Applications. 1986

[3] Wikipedia contributors. Fractional cascading. Wikipedia.(https://en.wikipedia.org/wiki/Fractional_cascading)

[4] 白马负金羁CSDN博客(https://blog.csdn.net/baimafujinji/article/details/52956605)