面向大规模查询的数据结构算法

上海市育才中学 丁梓洋

摘　要

在信息学奥林匹克竞赛（NOI）及相关竞赛中，数据结构题型常以高效查询与优化计算为核心，考察选手对算法设计与实现的能力。其中，离线算法因其能够高效处理大规模查询问题，在竞赛中得到了广泛应用。本文以「求区间内不同元素个数」问题为例，分析了莫队算法、CDQ 分治、分块技术等典型离线算法，探讨了它们在竞赛环境下的适用性、复杂度优化策略及应对卡常技巧。通过对不同算法的深入比较，本文为竞赛选手提供了一套较为系统的方法论，以帮助读者理解数据结构题的核心思想，提高算法设计与优化能力。

在实际工程应用中，大规模数据查询广泛存在于数据库系统、搜索引擎、数据挖掘等领域。传统的暴力查询方法在数据量较大时难以满足性能需求，因此需要利用高效的数据结构。本文亦为实际工程中高效处理大规模查询的问题提供理论支撑和实践指导。

关键词　信息学奥赛　离线算法　数据结构　莫队算法　分治思想　分块　树状数组　线段树

目　　录

第一章　例题

1.1　题目大意

给定一个长为 $N\,(1 \le N \le 10^6)$ 的序列 $\{a_i\}\,(1 \le a_i \le 10^6)$，给定指令：

$Q\ l\ r$：询问区间 $[l, r]$ 内有多少种不同的数。

现有 $M\,(1 \le M \le 10^6)$ 条指令。要求对每条询问指令输出其答案。内存限制 512 MB，时间限制 2.0 s。[1]

1.2　问题分析

该问题属于典型的区间查询问题，要求高效地处理大量查询，序列长度和询问次数都在 10^6 数量级。一种暴力的思路是对每个询问单独统计，但这在最坏情况下时间复杂度会达到 $O(NM)$，显然无法满足时间限制。

由于题目明确是静态查询，也就是说输入的序列不会发生变化，我们可以设计一个高效的离线算法来处理所有询问。

第二章　分块

2.1　思路分析

分块是一种常见的优化方法，适用于区间查询问题。将序列分为若干块，每个块内的元素个数相等。先预处理每个块内不同元素的个数。对于每个询问区间 $[l, r]$，如果 l 和 r 在同一个块内，则直接暴力计算；如果 l 和 r 不在同一个块内，利用 $[l, r]$ 真包含的块的预处理结果，避免重复计算，减少时间复杂度。

2.2　实现

在读入时，预处理 a_i 上一次出现的位置 bef_i 和下一次出现的位置 nxt_i。这步预处理操作可以在 $O(N)$ 的时间复杂度内完成。

首先，我们需要将序列分为若干块。设序列长度为 N，块的大小 $S = \left\lfloor \sqrt{N} \right\rfloor$，共有 $D = \left\lceil \frac{N}{S} \right\rceil$ 块。对于每个块，需要暴力地计算块内不同元素的个数。记二维数组 $f_{i,j}$

[1] 来源：洛谷 P1972 [SDOI2009] HH 的项链，有修改

表示第 i 到第 j 个块内不同元素的个数。则第 i 个块内不同元素的个数即为 $f_{i,i}$。此操作可以借用一个布尔数组记录 a_i 是否出现过，也可以利用 bef 数组实现。

我们可以使用递推的方法，计算 f 数组的所有值。其递推公式为 $f_{i,j}=f_{i,j-1}+x$，其中 x 表示第 j 个块内新增的不同元素的个数，即在第 j 个块内出现，而没有在第 i 到第 $j-1$ 个块内出现的元素个数。此操作可利用 bef 数组实现。对于每个递推过程，我们可以在 $O(S)$ 的时间复杂度内完成转移。至此，我们完成了所有预处理操作。

对于每条询问 $[l_i,r_i]$，先判断 l_i 与 r_i 是否在同一个块内。如果在同一个块内，直接暴力计算，其时间复杂度为 $O(S)$；如果不在同一个块内，我们可以利用 f 数组，避免重复计算。设 l_i 所在的块为 b_l，r_i 所在的块为 b_r，则 $[l_i,r_i]$ 内的不同元素的个数即为 f_{b_l+1,b_r-1} 加上 b_l 和 b_r 内的不同元素的个数。此操作的时间复杂度为 $O(S)$。

2.3 时间复杂度

该算法的时间复杂度为 $O\Big((N+M)\sqrt{N}\Big)$，其中 N 为序列长度，M 为询问区间的数量。预处理 bef 和 nxt 数组的时间复杂度为 $O(N)$；预处理 f 数组的时间复杂度为 $O(DS^2)$，即 $O\Big(N\sqrt{N}\Big)$；对于所有查询，时间复杂度为 $O(MS)$，即 $O\Big(M\sqrt{N}\Big)$。

2.4 衍生题目

分块是一种灵活且通用的思想。尤其在处理一些复杂的查询问题时，能充分利用其灵活性，处理那些普通树状数组或线段树难以做到的情况。对于区间查询和更新问题，分块可以在不需要过度复杂结构的情况下，提供较为高效的解决方案。

分块的一个关键优势在于它能够将问题的复杂度从一个大规模的结构中分解到若干小块中，这样能够更好地适应实际问题中的各种复杂约束。另外，分块不仅限于区间问题，还可以扩展到其他类型的问题，比如求最小公倍数、最大公约数、处理频率分布等。

分块的实现思路和其他数据结构不同，时间复杂度也不同。在应对不同题目时，读者可以根据自己擅长的解法、评估时间复杂度等因素，选择合适的方法。

以下推荐几道不同难度的题目，均可同时使用分块和数据结构完成，可供读者练习和比较两者的思路：

- 洛谷 P4145 上帝造题的七分钟 2 / 花神游历各国
- 洛谷 P1471 方差
- 洛谷 P1975 [国家集训队] 排队

对于难度更高的大分块题目，推荐 洛谷题单 - YNOI 大分块系列[2]。

第三章　普通莫队算法

3.1　思路分析

莫队算法是一种经典的离线算法，适用于多次的区间查询的问题。它可以利用上一次查询的结果，避免重复计算，从而提高效率。

假设已经计算好了区间 $[l,r]$ 内不同数的个数为 x，现询问区间 $[l,r+1]$ 内的不同数的个数。如果区间 $[l,r]$ 内没有 a_{r+1}，则答案即为 $x+1$，反之为 x。类似地，我们可以用这种方法暴力地从一个已经计算好的区间 $[l_i,r_i]$ 转移到 $[l_{i+1},r_{i+1}]$。

使用上述思路暴力地转移每个区间的时间复杂度为 $O(MN)$。莫队算法通过对询问离线，将询问按照更优的顺序排序，从而减少总转移距离。

3.2　实现

先对询问区间进行排序。莫队算法也使用了分块的思想，将区间分为多个块，所有左端点 l 在同一个块内的区间为一个整体。对于每个块，按照区间的右端点从小到大进行排序。最终的结果如 图 1 所示。这样离线的好处在后文可以体现。

用两个指针 p,q 表示现已计算好的区间 $[p,q]$，记该区间内不同元素的个数为 sum，区间内数 x 出现的次数为 cnt_x。在处理区间 $[l_i,r_i]$ 时，将左指针 p 逐位移动到 l_i。如果 p 需要向左移动 1 位，且 $cnt_{p-1}=0$，即a_{p-1} 在 $[p,q]$ 中没有出现过，则将 sum 增加 1；否则 sum 保持不变。同时，也要将 $cnt_{a_{p-1}}$ 增加 1，表示新的 $[p,q]$ 内该数的出现次数增加 1 次。如果 p 需要向右移动 1 位，则将 $cnt_{a_{p+1}}$ 减少 1。如果现在 $cnt_{a_{p+1}}=0$，即最后的一个 a_{p+1} 被移出了 $[p,q]$，则将 sum 减少 1；否则 sum 保持不变。右端点 q 的移动方式与左端点相反。

通过这样的暴力转移，我们可以在上一个区间的答案的基础上计算得到下一个区间的答案。经过离线的排序，如 图 1 所示，我们不难发现：左指针在同一个块内的移动距离较小，而右指针在每个块内都是单调增加的。这样的离线使总转移距离尽可能小，这便是莫队算法的核心思想。

[2] https://www.luogu.com.cn/training/44148

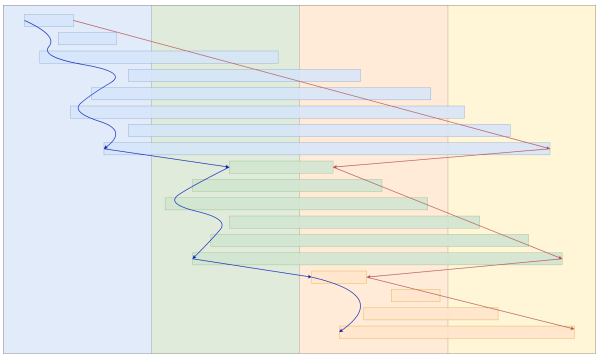

图 1　普通莫队算法

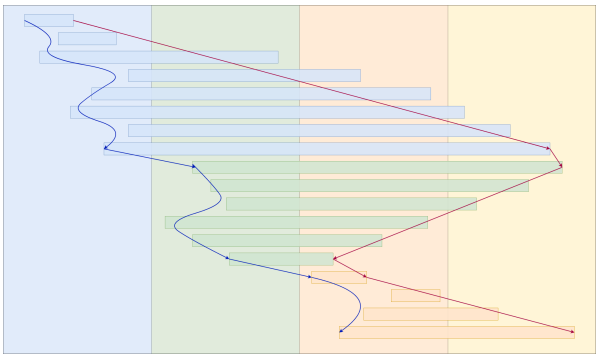

图 2　奇偶块交错排序优化的莫队算法

3.3　时间复杂度

在 N 与 m 同阶时，莫队算法的时间复杂度为 $O\left(N\sqrt{N}\right)$。其中，排序的时间复杂度为 $O(N\log N)$；区间之间转移的总时间复杂度为 $O\left(N\sqrt{N}\right)$。

对于区间之间转移的时间复杂度，我们可以通过以下方法计算得到：

令每一块中 L 的最大值为 $max_1, max_2, max_3, ..., max_{\lceil\sqrt{N}\rceil}$。由第一次排序可知，$max_1 \le max_2 \le ... \le max_{\lceil\sqrt{N}\rceil}$。显然，对于每一块暴力求出第一个询问的时间复杂度为 $O(N)$。考虑最坏的情况，在每一块中，R 的最大值均为 N，每次修改操作均要将 L 由 max_{i-1} 修改至 max_i 或由 max_i 修改至 max_{i-1}。考虑 R：因为 R 在块中已经排好序，所以在同一块修改完它的时间复杂度为 $O(N)$。对于所有块就是 $O\left(N\sqrt{N}\right)$。重点分析 L：因为每一次改变的时间复杂度都是 $O(max_i - max_{i-1})$ 的，所以在同一块中时间复杂度为 $O\left(\sqrt{N}\cdot(max_i - max_{i-1})\right)$。

将每一块 L 的时间复杂度合在一起，通过裂项求和可得：

$$O(L) = O\left(\sqrt{N}(max_1 - 1) + \sqrt{N}(max_2 - max_1) + ... + \sqrt{N}\left(max_{\lceil\sqrt{N}\rceil} - max_{\lceil\sqrt{N}\rceil} - 1\right)\right)$$

$$= O\left(\sqrt{N}\cdot\left(max_1 - 1 + max_2 - max_1 + ... + max_{\lceil\sqrt{N}\rceil} - max_{\lceil\sqrt{N}\rceil} - 1\right)\right)$$

$$= O\left(\sqrt{N}\cdot max_{\lceil\sqrt{N}\rceil} - 1\right)$$

又因为 $max_{\lceil\sqrt{N}\rceil}$ 最大为 N，因此 L 的总时间复杂度最坏情况下为 $O\left(N\sqrt{N}\right)$。[1]

3.4　优化

该题的数据使用莫队算法可能会被卡常，在洛谷的测试结果仅为 22 分。以下是几种优化方法。

3.4.1 最优块的大小

与分块有关的算法，块的大小对常数时间复杂度有很大的影响。设块长度为 S，那么对于任意多个在同一块内的询问，指针的移动的距离为 N，共有 $\frac{N}{S}$ 个块，移动的总距离就是 $\frac{N^2}{S}$。由于移动可能跨越块，所以还要加上 $O(MS)$，总复杂度为 $O\left(\frac{N^2}{S}+MS\right)$。当 S 取 $\frac{N}{\sqrt{M}}$ 时时间复杂度最优，为 $O\left(N\sqrt{M}\right)$。

块的大小对莫队算法的时间复杂度影响非常显著。例如，M 与 $\sqrt{N}$ 同阶时，最优的块大小 $S=\frac{N}{\sqrt{M}}=N^{1-\frac{1}{2}\times\frac{1}{2}}=N^{\frac{3}{4}}$，时间复杂度可以达到 $O\left(N^{\frac{5}{4}}\right)$。如果块的大小取 $\sqrt{N}$，其时间复杂度为 $O\left(N\sqrt{N}\right)$。[1]

在本题中，N 与 M 同阶，因此块的大小取 $\sqrt{N}$ 即可。

3.4.2 奇偶块交错排序优化

同一个块内的询问区间的右端点 r 均是递增的。在两个块之间转移时，指针 q 的移动距离较远。因此，我们可以做如下优化：对于编号为奇数的块，块内的询问区间按照 r 递增排序；对于编号为偶数的块，块内的询问区间按照 r 递减排序。这样，指针 q 在两个块之间的移动距离将会减小，可以在一定程度上优化常数时间复杂度，见图 1 与 图 2。使用该优化，在洛谷的测试结果可以提高 8 分。

3.4.3 数据离散化

由于该题输入数据 $a_i\in[1,10^6]$，范围较大。因此，我们可以考虑将输入序列的值进行离散化。经过离散化后，对 cnt 数组的访问和修改更加连续，每次 p,q 指针的转移速度更快。由于莫队算法的瓶颈在于 p,q 指针的转移，因此经过离散化后莫队算法的常数时间复杂度得以有效减少。使用该优化，在洛谷的测试结果可以提高 48 分。[2]

3.4.4 优化 cnt 数组

通过上一种优化方法，我们可以发现，cnt 数组的访问和修改较为耗时。实际上，我们可以不记录 cnt 数组。考虑在读入时预处理 a_i 上一次出现的位置 bef_i 和下一次出现的位置 nxt_i。在转移指针 p,q 时，只需要访问 bef_i 或 nxt_i 中的一个元素便可知道数 a_i 是否在原区间 $[p,q]$ 出现过，且无需进行对 bef 或 nxt 的修改操作。另外，在使用此方法优化时，离散化输入序列的值就没有必要了。使用该优化，在洛谷测试结果可以提高 60 分，最大的数据甚至可以在 $1.01s$ 内通过。[3]

3.5 衍生题目

本题的数据对莫队算法进行了一定的卡常，不适合作为莫队算法的入门练习。但近年来，不少省选题可以通过一定的优化，使用莫队算法轻松解决。相比正解，莫队算法的思路和实现较为简便。因此，本题亦可作为莫队算法常数优化的练习题。

以下提供几道普通莫队算法的模板题，可供读者快速掌握莫队算法，更好地理解莫队算法的核心思想：

- 洛谷 P2709 小 B 的询问
- 洛谷 P1494 [国家集训队] 小 Z 的袜子
- 洛谷 P4462 [CQOI2018] 异或序列

第四章　树状数组 / 线段树

4.1 思路分析

使用树状数组或线段树这类数据结构可以高效地处理区间查询和单点更新。

对于每条区间询问，我们可以在 $O(\log N)$ 的时间复杂度内求出区间内所有元素的和、最大值、最小值等信息。令每个数都为 1，区间求和可以等价于求区间内元素的个数，但是会有重复的计算。如果区间内出现多个相同的数，其中只有一个数为 1，其他均为 0，我们便可以通过区间求和得出区间内不同元素的个数。

4.2 实现

类似莫队算法，我们也要考虑将询问离线。具体来说，将每个查询区间按照其右端点 r 升序排序。这样的离线方便维护上文所述的 01 树状数组。记排序后第 i 个询问为 $[l_i, r_i]$。

使用树状数组维护每个位置的状态。每次根据 r 的升序计算询问区间的同时，更新树状数组。假设上一个处理的区间为 $[l_i, r_i]$，则现在处理的区间为 $[l_{i+1}, r_{i+1}]$，我们需要更新树状数组 $\left(c_{r_i}, c_{r_{i+1}}\right]$ 区间的每个值。

对于相同的值，我们只关心这个值在区间中出现的最右一个。如果 a_i 在 $[1, i)$ 之间没有出现过（即 $bef_i = 0$），则将 c_i 记为 1；如果 a_i 在 $[1, i)$ 之间出现过（即 $bef_i \neq 0$），则将 c_i 记为 1 的同时将 c_{bef_i} 记为 0（在维护树状数组时，通过将 c_i 加 1 或减 1 的

方式实现)。因此，我们就避免了重复计算同一个数值。$[l_i, r_i]$ 区间内的不同数的个数即为 $\sum_{j=l_i}^{r_i} c_j$（树状数组通过 $\text{query}(r_i) - \text{query}(l_i - 1)$ 实现)。

除了树状数组，我们还可以使用线段树来实现，实现方法与树状数组类似。线段树的每个节点维护区间内的不同数的个数。在更新和查询线段树时，我们可以通过递归地更新或查询左右子树来实现。其常数时间复杂度较树状数组更大，且代码较为复杂。

4.3 时间复杂度

该算法的时间复杂度为 $O((N + M) \log N)$，其中 N 为序列长度，M 为询问区间的数量。预处理 $\{bef_i\}$ 的时间复杂度为 $O(N)$。将查询按照右端点 r 排序的时间复杂度为 $O(M \log M)$。对于所有查询，更新树状数组或线段树的时间复杂度为 $O((N + M) \log N)$。

使用树状数组 / 线段树实现本题游刃有余，不需要过多的优化，其关键在于如何对询问离线，以及树状数组 / 线段树维护什么内容。相比莫队算法，其思路和代码实现略显复杂。

第五章　归并树

5.1 思路分析

归并树算法通过构建归并树维护序列的辅助信息，并结合二分查找快速回答每次询问。它结合了归并排序和线段树的思想。在构建时，它递归地将序列划分为左右子区间，对每个子区间进行排序，并将有序序列存储在对应的线段树节点上。对于询问 $[l, r]$，我们可以通过线段树的区间分解，将其拆分为 $O(\log N)$ 个节点区间，每个节点存储的 bef 值是有序的。利用二分查找，我们可以在 $O(\log N)$ 时间内统计每个节点中 $bef_i < l$ 的元素个数，从而计算答案。

5.2 实现

在读入序列 $\{a_i\}$ 时预处理每个元素 a_i 在序列中上一次出现的位置 bef_i。

利用归并树来组织 bef 数组的数据。归并树是一棵线段树，每个节点负责一段区间，存储该区间内 bef 值的有序序列。具体实现时，从根节点开始递归划分区间。对

于当前节点负责的区间 $[l,r]$，如果 $l=r$，说明是叶子节点，直接将 $bef[l]$ 存入该节点的向量中。如果 $l<r$，则将区间分为 $[l,mid]$ 和 $[mid+1,r]$ 两部分，分别递归构建左子树和右子树。子树构建完成后，将左右子树的有序 bef 序列归并到当前节点的序列中。归并的过程类似于归并排序，使用双指针依次比较并合并，确保当前节点存储的 bef 序列也是有序的。由于每个元素只会被归并 $O(\log N)$ 次，建树总时间复杂度为 $O(N\log N)$。

对于每次查询，给定区间 $[l,r]$，需要在归并树上计算答案。查询函数从根节点开始，判断当前节点负责的区间 $[x,y]$ 与查询区间 $[l,r]$ 的关系。如果 $[x,y]$ 完全被 $[l,r]$ 包含，则直接在该节点的有序 bef 序列中使用二分查找，统计小于 l 的 bef 值的个数，这些位置对应的元素在 $[l,r]$ 内是首次出现。如果 $[x,y]$ 与 $[l,r]$ 部分重叠，则递归查询左子树（如果 $l\le mid$）和右子树（如果 $mid<r$），将两部分的结果相加。

5.3 时间复杂度

归并树算法的时间复杂度为 $O((N+M)\log^2 N)$，其中 N 为序列长度，M 为询问区间的数量。构建归并树的时间复杂度为 $O(N\log N)$；对于每个询问，我们需要在 $O(\log N)$ 个节点上进行二分查找，每个节点内二分的时间复杂度为 $O(\log N)$，因此对于单个询问的时间复杂度为 $O(M\log^2 N)$。

5.4 优化

朴素的实现仍需在 $O(\log N)$ 个节点上进行二分查找，总复杂度为 $O(M\log^2 N)$，可能面临常数过大的风险，在洛谷的测试结果为 88 分。考虑应用离线处理和优化技巧：将所有询问按左端点 l 递增排序，并在每个节点记录上一次二分查找的结果 d。由于 l 单调递增，下次查询时可从 $d+1$ 开始二分，缩小查找范围，从而降低平均时间复杂度。通过这样的优化，可以有效提高性能。

第六章　扫描线

6.1 思路分析

扫描线不仅可以解决二维矩形的面积并、周长并等问题，还可以解决二维数点问题。如查询区间中值在 $[x,y]$ 内的元素个数。将该问题映射到二维坐标系中，通过扫描

线的思想，结合树状数组 / 线段树即可实现。对于本题，可采用类似的思路，其关键在于如何将题目转化为二维数点问题。

6.2 实现

在一个区间 $[l,r]$ 中，如果有多个相同的数，此时我们考虑在 $[l,r]$ 中第一次出现的元素，其余不产生贡献。那么一个数是否产生贡献可以通过 bef 数组求出。若 $bef_i < l$，则表明 a_i 在 $[l,r]$ 中第一次出现，产生贡献；否则不产生贡献。因此，该区间中不同元素的个数即为 $\sum_{i=l}^{r}[bef_i < l]$。

把 bef 数组映射到一个二维平面中，其中 bef_i 为点 (i, bef_i)。至此，我们把原问题转化为：求二维平面中，以 $(l,0)$ 为左下角、$(r,l-1)$ 为右上角的矩形中的点的个数。参考 图 3，左侧表示的是输入的序列 $\{a\}$ 以及预处理的数组 $\{bef\}$，询问的区间为 $[6,12]$，我们可以把 bef 数组如图映射。绿色矩形 ans 内点的数量就是该区间内不同元素的数量，而红色矩形 now 表示的是在该区间内出现次数超过一次的元素。

i	a_i	bef_i
1	1	0
2	9	0
3	1	1
4	5	0
5	4	0
6	4	5
7	5	4
8	6	0
9	9	2
10	1	3
11	6	8
12	6	11
13	6	12
14	9	9
15	5	7

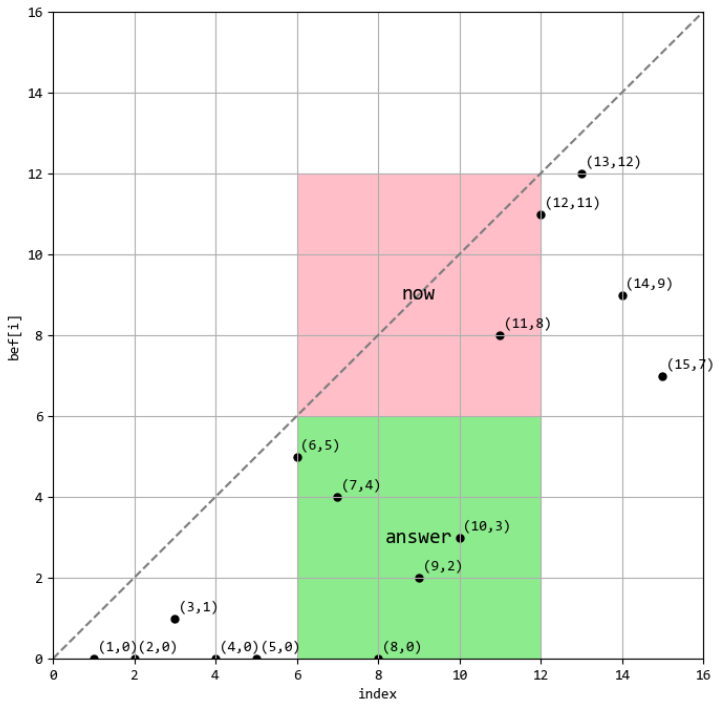

图 3　转化为二维数点问题

求该矩形内的点的数量可以通过差分的方式实现。由于该矩形左下角顶点的纵坐标一定为 0，矩形 $(l,0) \to (r,l-1)$ 内点的数量即为矩形 $(0,0) \to (r,l-1)$ 内的点的数

量减去矩形 $(0,0) \to (l-1, l-1)$ 内的点的数量。这样，我们便可以通过扫描线分别求出这两个差分矩形内的点的数量。

扫描线将直线 $x = 0$ 从左向右进行扫描，扫描的过程中用树状数组 c 维护纵坐标对应范围内点的数量。树状数组共有 2 种操作：单点增加和区间求和。对于每个元素 a_i，需要插入点 (i, bef_i)。因此，当扫描到 $x = i$ 时，c_{bef_i} 增加 1。对于每个询问区间 $[l_i, r_i]$，当扫描到 $x = l - 1$ 时，查询 $\sum_{i=0}^{l-1} c_i$ 的值；当扫描到 $x = r$ 时，查询 $\sum_{i=0}^{l-1} c_i$ 的值。两次查询的差值即为 $[l_i, r_i]$ 区间内不同元素的个数。使用线段树操作类似树状数组，常数略大。

6.3 时间复杂度

该算法的时间复杂度为 $O((N+M)\log(N+M))$。预处理 bef 数组时间复杂度 $O(N)$；将询问操作和加点操作排序时间复杂度 $O((2M+N)\log(2M+N))$，每次树状数组 / 线段树的修改和查询操作的时间复杂度均为 $O(\log N)$，共有 N 次加点操作和 $2M$ 次查询操作。

第七章　CDQ 分治

7.1 实现

上文已经证明，原问题可以转化为：对于每个询问 $[l, r]$，统计满足 $i \in [l, r]$ 且 $bef[i] \le l - 1$ 的 i 的个数。我们可以用差分的思想，将区间 $[l, r]$ 的统计拆成 $[1, r]$ 减去 $[1, l-1]$ 的统计。这样，每个询问被拆成了两个子问题，我们可以用分治来统一高效处理这些子问题。

定义一个结构体 node，包含三个字段 (i, x, opt) 表示一个记录。其中，i 是位置或区间的右端点，x 是约束条件，opt 表示操作类型。对于输入序列中的每个元素 a_i，我们将其转化为 node 元素 $(i, bef_i, 0)$，表示一个元素记录，其中 bef_i 是 a_i 上一次出现的位置（若首次出现则为某个无效值，如 -1）。对于每个询问 $[l_j, r_j]$，将其转化为两个 node 元素：一个是 $(l_j - 1, l_j - 1, -j)$，表示减去 $[1, l_j - 1]$ 的贡献；另一个是 $(r_j, l_j - 1, j)$，表示加上 $[1, r_j]$ 的贡献。这里的 j 是询问编号，opt 的正负区分加减操作。

将所有 node 元素放入数组 t 中，并将 t 按照 i 从小到大排序。若 i 值相等，则需保证询问记录排在元素记录之后，因为询问的统计依赖于元素贡献已计算完成。排序

后，对于任意 $i = x$ 的询问，其左侧所有 $i \le x$ 的元素记录必然先出现，这为后续分治中统计贡献提供了正确性基础。

现在进入 CDQ 分治的核心部分。分治的目标是计算每个询问的答案。我们将数组 t 分成左右两半，先递归处理子问题，然后合并结果。合并时，重点在于右半边的询问如何接收左半边元素的贡献。根据问题定义，询问统计的条件是 $bef[i] \le l-1$，即元素记录的 $x_i \le l-1$，因此在归并时我们按 x 值从小到大排序处理。过程中维护一个变量 s，表示左半边已处理过的元素个数（即 $opt = 0$ 的记录数量）。当遇到右半边的询问记录时，根据其 opt 的正负，将当前的 s 累加到对应询问的答案上（$opt > 0$ 表示加，$opt < 0$ 表示减）。通过这种方式，右半边的每个询问都能正确接收左半边满足 $x_i \le l-1$ 条件的元素贡献。

具体实现中，归并过程可通过双指针完成。设左指针遍历左半边，右指针遍历右半边。比较当前左右记录的 x 值：若左边的 x 值小于等于右边的 x 值，则将左边记录加入临时结果，若该记录是元素（$opt = 0$），则 s 加 1；否则将右边记录加入结果，若该记录是询问（$opt\ n = 0$），则根据 opt 的正负更新对应答案。归并完成后，将临时数组复制回原数组 t，继续递归处理左右子区间。

7.2 时间复杂度

初始排序需要 $O((M+N)\log(M+N))$，其中 M 是询问个数，N 是序列长度。CDQ 分治的每次归并复杂度为 $O(N)$（双指针线性扫描），递归层数为 $O(\log(M+N))$，因此分治部分总复杂度为 $O((M+N)\log(M+N))$。整体时间复杂度为 $O((M+N)\log(M+N))$。

第八章　总结

本文系统分析了分块、莫队算法、树状数组/线段树、归并树、扫描线及 CDQ 分治六类算法在区间查询问题中的表现。从时间复杂度来看，分块和莫队算法的时间复杂度均为 $O\left(N\sqrt{N}\right)$，实现灵活但理论复杂度较高；树状数组/线段树和扫描线算法的时间复杂度为 $O(N \log N)$，理论效率更优；归并树和 CDQ 分治的时间复杂度分别为 $O(N \log^2 N)$ 和 $O(N \log N)$，但前者常数较大，后者对离线处理要求较高。从实现难度来看，分块和莫队算法实现较为简单，适合竞赛场景快速编码；树状数组和 CDQ 分

治代码逻辑复杂，需较高抽象能力；归并树和扫描线则需要结合特定数据结构与问题转化技巧。

在信息学竞赛中，分块和莫队算法因灵活性与低代码量备受青睐。例如，分块可处理复杂约束的区间问题（如区间众数），莫队算法通过一定的排序优化和离散化可轻松通过不少题目。树状数组/线段树作为通用数据结构，适用于需动态维护的查询（如带修改的区间问题）。扫描线和 CDQ 分治则在高维问题中展现优势，如二维数点或离线统计。在实际竞赛中，需根据数据规模、内存限制和常数优化灵活选择算法。例如，莫队算法在数据量较大时易受常数影响，而树状数组的稳定 $O(N \log N)$ 复杂度更可靠。

在实际工程中，算法需兼顾效率与可维护性。树状数组和线段树因高效且易于扩展，广泛应用于数据库索引、实时监控系统；扫描线算法在图形处理和大规模数据聚合中表现优异；分块技术则适合流式计算或分布式场景，通过数据分片降低单点负载。相比之下，莫队算法和 CDQ 分治因依赖离线处理，工程适用性受限，但在特定批处理任务（如日志分析）中仍有价值。总体而言，算法选择需权衡问题特性、资源约束与应用场景，竞赛中追求极限优化，而工程中更注重鲁棒性与可扩展性。

参考文献

[1] KONNYAKUXZY. OI Wiki - 莫队算法[EB/OL]. (2018-07-30). https://oi-wiki.org/misc/mo-algo/.

[2] OOJ_BAI. 重新编号颜色优化莫队算法[EB/OL]. (2024-09-19). https://www.luogu.com.cn/discuss/929927.

[3] PHANTOM_LANDLER. 通过优化桶优化莫队算法[EB/OL]. (2024-11-25). https://www.luogu.com.cn/discuss/1005310.

致　　谢

- 感谢「沪疆教育信息化 算法小论文」提供的学习和交流的平台。
- 感谢张卫国老师、诸峰老师对我的指导和帮助。
- 感谢王泽华同学、程佳润同学对我的支持。
- 感谢洛谷平台提供的题目和测试。