对一类「求区间内不同元素个数」问题的分析

上海市育才中学 丁梓洋

摘　要

一类「求区间内不同元素个数」问题是大数据结构和离线算法入门的经典例题。

近年来，大数据结构题在信息学奥赛中常以压轴题出现。本文通过介绍一类「求区间内不同元素个数」的问题，分析比较了几种不同的算法，希望能够帮助读者更好地理解此类问题的解决方法，感受不同算法在数据结构问题中的应用。

关键词　离线算法　数据结构　莫队算法　CDQ 分治　分块　树状数组　线段树　珂朵莉树

目　　录

第一章　区间询问

1.1　题目大意

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

- $Q\ l\ r$：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。由于序列长度和查询数量都非常大，若采用暴力解法，直接遍历每个区间分别计算答案将会导致时间复杂度过高，超出了时间限制。

因此，采用高效的数据结构和离线的思想来优化查询过程是必要的。常见的方案包括树状数组、线段树、莫队算法等。这些数据结构都可以高效地支持区间查询和单点更新，因此它们成为了解决这类问题的基础。

1.3　分块

1.3.1　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。

1.3.2　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 $O(S)$。

1.3.3 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 $O\Big(M\sqrt{N}\Big)$。

1.3.4 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]。

1.4 普通莫队算法

1.4.1 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。

1.4.2 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

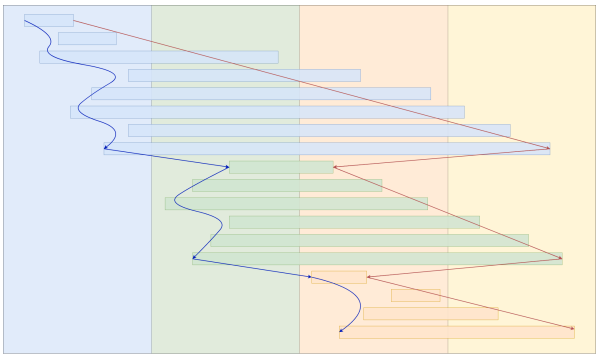

图 1　普通莫队算法

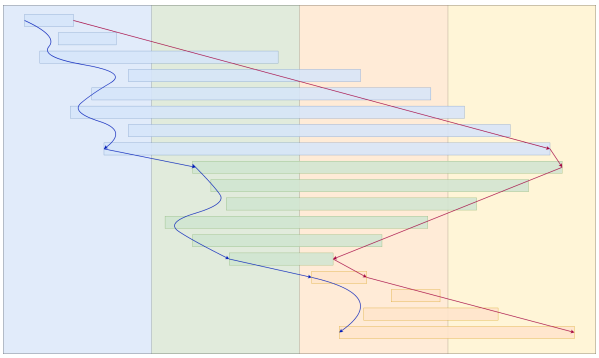

图 2　奇偶块交错排序优化的莫队算法

1.4.3　时间复杂度

在 n 与 m 同阶时，莫队算法的时间复杂度为 $O(n\sqrt{n})$。其中，排序的时间复杂度为 $O(n\log n)$；可以证明，在区间之间转移的总时间复杂度为 $O(n\sqrt{n})$[1]。

1.4.4　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：

与分块有关的算法，块的大小对莫队算法的常数时间复杂度有很大的影响。可以证明，块的大小为 $\frac{n}{\sqrt{m}}$ 时最优[1]。在本题中，n 与 m 同阶，因此块的大小取 $\sqrt{n}$ 即可。

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。

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]

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]

1.4.5 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

1.5 树状数组 / 线段树

1.5.1 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。

1.5.2 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。

1.5.3 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。

1.6 扫描线

1.6.1 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。

1.6.2 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

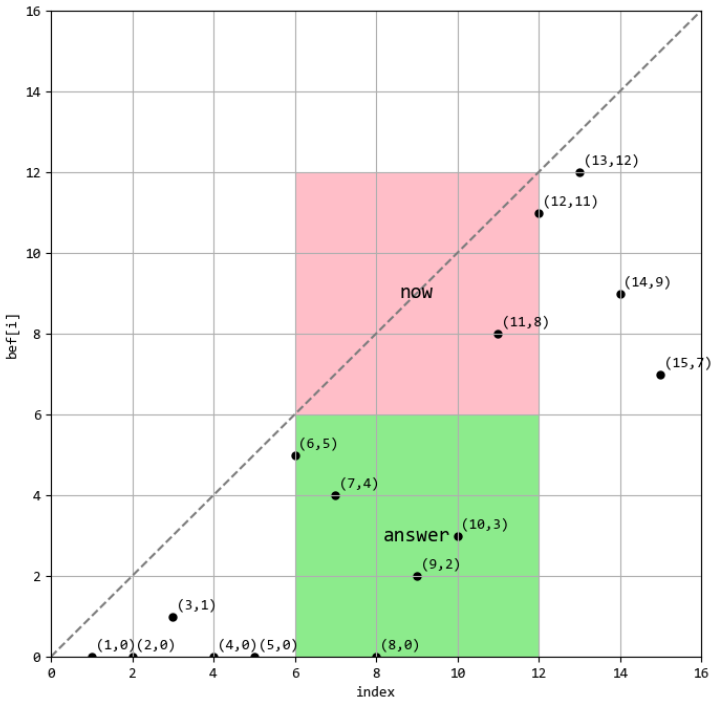

图 3　转化为二维数点问题

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。

1.6.3 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。

1.6.4 总结

不难发现，上述思路的代码实现与 小节 1.5 中的思路十分类似。两者均对询问离线，按照左/右端点进行排序，在同一个区间内出现多次的情况均只考虑最右/最左的元素，通过差分的方式求解。因此，扫描线的思路与直接使用树状数组/线段树的思路在某种程度上是等价的。扫描线的优势在于其更易思考，是求解这类问题的一种常见思路。

1.7 分治

1.8 归并树

1.9 珂朵莉树

1.10 总结

除了上文提供的解法，本题还有不少可以拿到部分分的解法。例如，可以使用树套树、主席树等暴力数据结构。这些数据结构在本题的数据范围内无法完全通过，但其思路简单，在一些特定的问题上能够提供更好的解法。

第二章 区间询问、单点修改

2.1 题目大意

给定一个长为 N $(1 \le N \le 10^5)$ 的序列 $\{a_i\}$ $(1 \le a_i \le 10^6)$，有如下两种指令：

- $Q\ l\ r$：询问区间 $[l,r]$ 内有多少种不同的数。
- $R\ i\ x$：将 a_i 替换为 x。

现有 $M\,(1 \le M \le 10^5)$ 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 2.5 s。[3]

2.2 分析

2.3 带修莫队

2.4 CDQ 分治

2.5 分块

2.6 ZKW 线段树

2.7 树套树

2.8 珂朵莉树

第三章 区间询问、区间修改

3.1 题目大意

给定一个长为 $N\,(1 \le N \le 10^5)$ 的序列 $\{a_i\}\,(1 \le a_i \le 10^9)$，有如下两种指令：

- $Q\ l\ r$：询问区间 $[l, r]$ 内有多少种不同的数。
- $R\ l\ r\ x$：将序列中 $[l, r]$ 内的所有数都替换为 x。

现有 $M\,(1 \le M \le 10^5)$ 条指令。要求对每条询问指令输出其答案。内存限制 64 MB，时间限制 1.5 s。[4]

[3]来源：洛谷 P1903 [国家集训队] 数颜色 / 维护队列，有修改
[4]来源：洛谷 P4690 [YNOI2016] 镜子里的昆虫，有修改

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。

附　　录

问题 1 - 分块解法

```cpp
#include <bits/stdc++.h>
using namespace std;

constexpr int MAX = 1e6 + 9, MAXn = 1e6 + 9, MAXsn = 1e3 + 9;
int a[MAXn], c[MAXn], p[MAXn], ans[MAXn];
int last[MAX], bef[MAXn], nxt[MAXn];
int d[MAXsn][MAXsn];
int n, sn, dn;

pair<int, int> border(const int & di) {
    int p = (di - 1) * sn + 1;
    int q = min(di * sn, n);
    return {p, q};
}

int piece(const int & p, const int & q) {
    bitset<MAX> ved;
    int ans = 0;
    for (int i = p; i <= q; ++i) {
        ans += !ved[a[i]];
        ved[a[i]] = true;
    }
    return ans;
}

int main() {
    scanf("%d", &n);
    sn = sqrt(n);
    dn = (n - 1) / sn + 1;
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
        bef[i] = last[a[i]];
        if (bef[i]) nxt[bef[i]] = i;
        nxt[i] = n + 1;
        last[a[i]] = i;
    }

    for (int di = 1; di <= dn; ++di) {
        int p, q;
        tie(p, q) = border(di);
        d[di][di] = piece(p, q);
    }
    for (int dis = 2; dis <= sn; ++dis) {
        for (int di = 1; ; ++di) {
            int dj = di + dis - 1;
            if (dj > dn) break;
            d[di][dj] = d[di][dj - 1];
            // [di+0][di+1][di+2][di+3][dj-1][dj-0]
            // [         d[di][dj-1]         ]
```

```
            // ^bp                              ^p   ^q
            int bp = border(di).first, p, q;
            tie(p, q) = border(dj);
            for (int i = p; i <= q; ++i)
                d[di][dj] += bef[i] < bp;
                //^ Do not use `if (bef[i] < bp) ++d[di][dj];`,
                //^ or its running time will be doubled.
        }
    }

    int m;
    scanf("%d", &m);
    for (int _ = 1; _ <= m; ++_) {
        int p, q, ans = 0;
        scanf("%d %d", &p, &q);
        int l = p / sn + 1, r = q / sn + 1;
        if (r - l <= 1) {
            ans = piece(p, q);
        } else {
            ans = d[l + 1][r - 1];
            // {p+0}{p+1}[l+1][l+2][r-2][r-1]{q-1}{q-0}
            //           ^m                 ^n
            int m = border(l + 1).first, n = border(r - 1).second;
            for (int i = p; i < m; ++i)
                ans += nxt[i] > n;
            for (int i = q; i > n; --i)
                ans += bef[i] < p;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

问题 1 - 普通莫队解法（无优化）

```
#include <bits/stdc++.h>
using namespace std;
#define st(x) get<0>(x)
#define nd(x) get<1>(x)
#define rd(x) get<2>(x)
typedef tuple<int, int, int> tup;

constexpr int MAX = 1e6 + 9;
constexpr int MAXn = 1e6 + 9;
int a[MAXn], ans[MAXn];
int cnt[MAX], sum = 0;
vector<tup> dq;
int n, m, unit;

bool cmp(const tup & x, const tup & y) {
    int bx = st(x) / unit, by = st(y) / unit;
    if (bx != by) return bx < by;
    return nd(x) < nd(y);
}
```

```
void add(int x) {
    if(!cnt[x]) ++sum;
    ++cnt[x];
}

void del(int x) {
    --cnt[x];
    if(!cnt[x]) --sum;
}

int main() {
    scanf("%d", &n);
    unit = sqrt(n);
    for (int i = 1; i <= n; ++i)
        scanf("%d", &a[i]);
    scanf("%d", &m);
    for (int i = 1; i <= m; ++i) {
        int l, r;
        scanf("%d %d", &l, &r);
        dq.emplace_back(l, r, i);
    }
    sort(dq.begin(), dq.end(), cmp);
    int p = 1, q = 0;
    for (const tup& tu : dq) {
        const int l = st(tu), r = nd(tu), k = rd(tu);
        while(p < l) del(a[p++]);
        while(p > l) add(a[--p]);
        while(q < r) add(a[++q]);
        while(q > r) del(a[q--]);
        ans[k] = sum;
    }
    for (int i = 1; i <= m; ++i)
        printf("%d\n", ans[i]);
    return 0;
}
```

问题 1 - 普通莫队解法（奇偶块交错排序优化）

```
#include <bits/stdc++.h>
using namespace std;
#define st(x) get<0>(x)
#define nd(x) get<1>(x)
#define rd(x) get<2>(x)
typedef tuple<int, int, int> tup;

constexpr int MAX = 1e6 + 9;
constexpr int MAXn = 1e6 + 9;
int a[MAXn], ans[MAXn];
int cnt[MAX], sum = 0;
vector<tup> dq;
int n, m, unit;

bool cmp(const tup & x, const tup & y) {
```

```
    int bx = st(x) / unit, by = st(y) / unit;
    if (bx != by) return bx < by;
    if (bx & 1) return nd(x) < nd(y);
    return nd(x) > nd(y);
}

void add(int x) {
    if(!cnt[x]) ++sum;
    ++cnt[x];
}

void del(int x) {
    --cnt[x];
    if(!cnt[x]) --sum;
}

int main() {
    scanf("%d", &n);
    unit = sqrt(n);
    for (int i = 1; i <= n; ++i)
        scanf("%d", &a[i]);
    scanf("%d", &m);
    for (int i = 1; i <= m; ++i) {
        int l, r;
        scanf("%d %d", &l, &r);
        dq.emplace_back(l, r, i);
    }
    sort(dq.begin(), dq.end(), cmp);
    int p = 1, q = 0;
    for (const tup& tu : dq) {
        const int l = st(tu), r = nd(tu), k = rd(tu);
        while(p < l) del(a[p++]);
        while(p > l) add(a[--p]);
        while(q < r) add(a[++q]);
        while(q > r) del(a[q--]);
        ans[k] = sum;
    }
    for (int i = 1; i <= m; ++i)
        printf("%d\n", ans[i]);
    return 0;
}
```

问题 1 - 普通莫队算法（离散化数值优化）

```
#include <bits/stdc++.h>
using namespace std;
#define st(x) get<0>(x)
#define nd(x) get<1>(x)
#define rd(x) get<2>(x)
typedef tuple<int, int, int> tup;

constexpr int MAX = 1e6 + 9;
constexpr int MAXn = 1e6 + 9;
int a[MAXn], ans[MAXn];
```

```
int mp[MAX], col;
int cnt[MAX], sum = 0;
vector<tup> dq;
int n, m, unit;

bool cmp(const tup & x, const tup & y) {
    int bx = st(x) / unit, by = st(y) / unit;
    if (bx != by) return bx < by;
    if (bx & 1) return nd(x) < nd(y);
    return nd(x) > nd(y);
}

void add(int x) {
    if(!cnt[x]) ++sum;
    ++cnt[x];
}

void del(int x) {
    --cnt[x];
    if(!cnt[x]) --sum;
}

int main() {
    scanf("%d", &n);
    unit = sqrt(n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
        if (!mp[a[i]]) mp[a[i]] = ++col;
        a[i] = mp[a[i]];
    }
    scanf("%d", &m);
    for (int i = 1; i <= m; ++i) {
        int l, r;
        scanf("%d %d", &l, &r);
        dq.emplace_back(l, r, i);
    }
    sort(dq.begin(), dq.end(), cmp);
    int p = 1, q = 0;
    for (const tup& tu : dq) {
        const int l = st(tu), r = nd(tu), k = rd(tu);
        while(p < l) del(a[p++]);
        while(p > l) add(a[--p]);
        while(q < r) add(a[++q]);
        while(q > r) del(a[q--]);
        ans[k] = sum;
    }
    for (int i = 1; i <= m; ++i)
        printf("%d\n", ans[i]);
    return 0;
}
```

问题 1 - 普通莫队算法（cnt 数组优化）

```cpp
#include <bits/stdc++.h>
using namespace std;
#define st(x) get<0>(x)
#define nd(x) get<1>(x)
#define rd(x) get<2>(x)
typedef tuple<int, int, int> tup;

constexpr int MAX = 1e6 + 9;
constexpr int MAXn = 1e6 + 9;
int a[MAXn], ans[MAXn];
int last[MAX], bef[MAXn], nxt[MAXn];
int cnt[MAX], sum = 0;
vector<tup> dq;
int n, m, unit;

bool cmp(const tup & x, const tup & y) {
    int bx = st(x) / unit, by = st(y) / unit;
    if (bx != by) return bx < by;
    if (bx & 1) return nd(x) < nd(y);
    return nd(x) > nd(y);
}

int main() {
    scanf("%d", &n);
    unit = sqrt(n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
        bef[i] = last[a[i]];
        if (bef[i]) nxt[bef[i]] = i;
        nxt[i] = n + 1;
        last[a[i]] = i;
    }
    scanf("%d", &m);
    for (int i = 1; i <= m; ++i) {
        int l, r;
        scanf("%d %d", &l, &r);
        dq.emplace_back(l, r, i);
    }
    sort(dq.begin(), dq.end(), cmp);
    int p = 1, q = 0, sum = 0;
    for (const tup& tu : dq) {
        const int l = st(tu), r = nd(tu), k = rd(tu);
        while (p < l) sum -= (q < nxt[p++]);
        while (p > l) sum += (q < nxt[--p]);
        while (q < r) sum += (p > bef[++q]);
        while (q > r) sum -= (p > bef[q--]);
        ans[k] = sum;
    }
    for (int i = 1; i <= m; ++i)
        printf("%d\n", ans[i]);
    return 0;
}
```

问题 1 - 树状数组解法

```cpp
#include <bits/stdc++.h>
using namespace std;
#define st(x) get<0>(x)
#define nd(x) get<1>(x)
#define rd(x) get<2>(x)
typedef tuple<int, int, int> tup;

constexpr int MAXn = 1e6 + 9;
int a[MAXn], c[MAXn], p[MAXn], ans[MAXn];
map<int, int> last;
vector<tup> dq;
int n, m;

bool cmp(const tup & x, const tup & y) {
    if (nd(x) == nd(y)) return st(x) < st(y);
    return nd(x) < nd(y);
}

int lowbit(int x) {
    return x & (-x);
}

void add(int x, int v) {
    for (int i = x; i <= n; i += lowbit(i))
        c[i] += v;
}

int query(int x) {
    int ans = 0;
    for (int i = x; i; i -= lowbit(i))
        ans += c[i];
    return ans;
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
        p[i] = last[a[i]];
        last[a[i]] = i;
    }
    scanf("%d", &m);
    for (int i = 1; i <= m; ++i) {
        int l, r;
        scanf("%d %d", &l, &r);
        dq.emplace_back(l, r, i);
    }
    sort(dq.begin(), dq.end(), cmp);
    int now = 1;
    for (const tup & tu : dq) {
        const int l = st(tu), r = nd(tu), k = rd(tu);
        for (; now <= r; ++now) {
            add(now, 1);
            if (p[now]) add(p[now], -1);
        }
        ans[k] = query(r) - query(l - 1);
```

```
    }
    for (int i = 1; i <= m; ++i)
        printf("%d\n", ans[i]);
    return 0;
}
```