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　普通莫队算法

1.3.1　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。

[1] 来源：洛谷 P1972 [SDOI2009] HH 的项链，有修改

1.3.2 解法

先对询问区间进行排序。莫队算法使用了分块的思想，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。

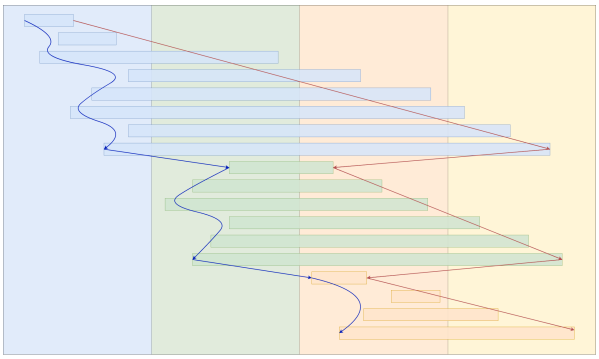

图 1　普通莫队算法

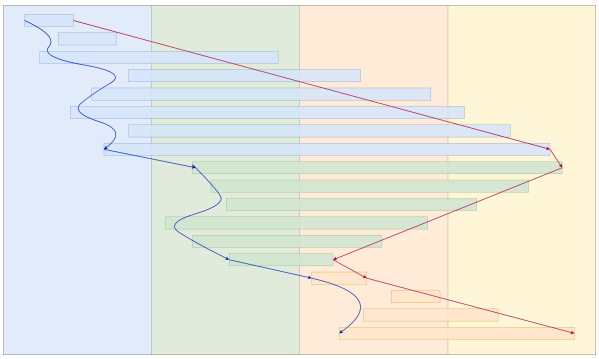

图 2　奇偶块交错排序优化的莫队算法

1.3.3 时间复杂度

在 n 与 m 同阶时，莫队算法的时间复杂度为 $O(n\sqrt{n})$。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]。

1.3.4 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：

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]

1.3.5 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

1.4 树状数组 / 线段树

1.4.1 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。

1.4.2 解法

首先，我们需要预处理序列中每个元素上一次出现的位置。记序列第 i 位的值为 a_i，a_i 上一次在序列中出现的位置为 p_i。这个预处理可以在读入序列时完成，时间复杂度为 $O(n)$。

我们可以考虑将询问离线。具体来说，将每个查询区间按照其右端点 r 升序排序。这样的离线可以在处理查询时，利用之前处理的结果，避免重复计算。记排序后第 i 个询问为 $[l_i, r_i]$。

我们可以使用树状数组来维护每个位置的状态。树状数组可以实现在 $O(\log N)$ 的时间复杂度内完成单点更新和区间查询的操作。

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 $p_i = 0$），则将 c_i 记为 1；如果 a_i 在 $[1, i)$ 之间出现过（即 $p_i \neq 0$），则将 c_i 记为 1 的同时将 c_{p_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，且代码实现较为复杂。

1.4.3 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 $\{p_i\}$ 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。但是，相比莫队算法，其代码实现略显复杂。

1.5 分块

https://www.luogu.com.cn/article/4vp3j5v4

1.6 分治

https://www.luogu.com.cn/article/9ajn0u1i

1.7 主席树

TODO

1.8 扫描线

https://www.luogu.com.cn/article/j7n2fv29

1.9 归并树

https://www.luogu.com.cn/article/rzuy3es9

1.10 总结

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。[2]

[2]来源：洛谷 P1903 [国家集训队] 数颜色 / 维护队列，有修改

2.2 分析

2.3 带修莫队

https://www.luogu.com.cn/article/tkzm8297

2.4 CDQ 分治

https://www.luogu.com.cn/article/hzoyp8rd

2.5 分块

https://www.luogu.com.cn/article/6o26r3fu

2.6 ZKW 线段树

https://www.luogu.com.cn/article/um3phhxf

2.7 树套树

https://www.luogu.com.cn/article/fp3jadsk

2.8 珂朵莉树

https://oi-wiki.org/misc/odt/

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。[3]

[3]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 - 普通莫队解法（无优化）

```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 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）

```
#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

```
#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;
}
```