莫队

张敬东

2023 年 12 月 6 日

目录

目录

今天的题单：点我
今天是我第一次上台，PPT 可能不是那么美观，如果有错误请大胆指出。
例题不会很难，请放心食用。

目录

莫队简介

什么是莫队？
莫队是由莫涛大神提出的一种~~暴力~~区间操作算法，它框架简单、板子好写、复杂度优秀。
然而由于莫队算法应用的毒瘤，很多莫队题难度评级都很高（蓝紫黑），令很多初学者望而却步。但如果你真正理解了莫队的算法原理，它用起来还是挺简单的。

莫队简介

什么是莫队？
莫队是由莫涛大神提出的一种~~暴力~~区间操作算法，它框架简单、板子好写、复杂度优秀。
然而由于莫队算法应用的毒瘤，很多莫队题难度评级都很高（蓝紫黑），令很多初学者望而却步。但如果你真正理解了莫队的算法原理，它用起来还是挺简单的。

前置知识

- 分块思想。
- `sort` 的用法（包括重载运算符或 `cmp` 函数，多关键字排序）。
- And so on.

使用莫队的情境

若 $m=O(n)$（即 m、n 同阶），且 $[l,r]$ **的答案能** $O(1)$ **地转换到** $[l-1,r],[l,r+1],[l+1,r],[l,r-1]$ **区间（即相邻区间）的答案**，那么莫队可以在 $O(n\sqrt{n})$ 的时间复杂度内**离线**求出所有询问的答案。

使用莫队的情境

若 $m = O(n)$（即 m、n 同阶），且 $[l, r]$ **的答案能** $O(1)$ **地转换到** $[l-1, r], [l, r+1], [l+1, r], [l, r-1]$ **区间（即相邻区间）的答案，**那么莫队可以在 $O(n\sqrt{n})$ 的时间复杂度内**离线**求出所有询问的答案。

注意

莫队是离线算法。如果题目强制在线，则不以可用莫队。

什么是离线、在线？

如果算法需要知道所有询问才能开始算法，则称此算法为离线算法。
读入一个询问，回答一个询问的算法叫在线算法。
强制在线就是要求你读入一个询问就立马回答。

莫队的基本思想

离线存下所有询问，借助分块按照一定的顺序处理这些询问，使得询问之间可以互相利用（一般情况下为了方便，只会是本次询问利用上次询问的答案），以减小时间复杂度。

目录

算法流程

1. 离线存下所有询问。

算法流程

1. 离线存下所有询问。
2. 以二元组 $(bel[l_i], r_i)$ 为关键字升序对所有询问排序。
   其中 i 表示当前询问编号，$bel[l_i]$（belong，属于）表示 l_i 所在的块的编号。

算法流程

1. 离线存下所有询问。
2. 以二元组 $(bel[l_i], r_i)$ 为关键字升序对所有询问排序。
   其中 i 表示当前询问编号，$bel[l_i]$（belong，属于）表示 l_i 所在的块的编号。
3. 遍历每个询问，维护两个指针 $[l, r]$ 表示当前区间，$tmpans$ 表示当前答案。
   初始时 $l = 1, r = 0$（如果 $l = 0$，那么我们还需要删除 a_0，导致一些奇怪的错误）。
   l, r 需区别于 l_i, r_i，它们一对是我们维护的指针（下标），一对是数据给出的询问。

算法流程

1. 离线存下所有询问。
2. 以二元组 $(bel[l_i], r_i)$ 为关键字升序对所有询问排序。
   其中 i 表示当前询问编号，$bel[l_i]$（belong，属于）表示 l_i 所在的块的编号。
3. 遍历每个询问，维护两个指针 $[l, r]$ 表示当前区间，$tmpans$ 表示当前答案。
   初始时 $l = 1, r = 0$（如果 $l = 0$，那么我们还需要删除 a_0，导致一些奇怪的错误）。
   l, r **需区别于** l_i, r_i**，它们一对是我们维护的指针（下标），一对是数据给出的询问。**
4. 移动区间 $[l, r] \to [l_i, r_i]$。途中维护区间 $[l, r]$ 的答案 $tmpans$。

算法流程

1. 离线存下所有询问。
2. 以二元组 $(bel[l_i], r_i)$ 为关键字升序对所有询问排序。
   其中 i 表示当前询问编号，$bel[l_i]$（belong，属于）表示 l_i 所在的块的编号。
3. 遍历每个询问，维护两个指针 $[l, r]$ 表示当前区间，$tmpans$ 表示当前答案。
   初始时 $l = 1, r = 0$（如果 $l = 0$，那么我们还需要删除 a_0，导致一些奇怪的错误）。
   l, r 需区别于 l_i, r_i，它们一对是我们维护的指针（下标），一对是数据给出的询问。
4. 移动区间 $[l, r] \to [l_i, r_i]$。途中维护区间 $[l, r]$ 的答案 $tmpans$。
5. 移动结束后，记录区间 $[l_i, r_i]$ 的答案 ans_i。
   （$ans_i \leftarrow tmpans$）。

算法代码：洛谷 P3901 数列找不同（模板题）【有其他解法】

```
constexpr int N=114514;
int n,m,a[N],S;// S: 块长
// [1,S] 区间的块编号为 1，[S+1,2S] 区间的块编号为 2，以此类推。
inline int bel(int x){return (x-1)/S+1;}
struct query// 询问结构体
{
    int l,r,id;// 分别为每个询问区间的左端点、右端点、询问的编号。
    friend inline bool operator < (query x,query y)
    {return (bel(x.l)==bel(y.l)?x.r<y.r:bel(x.l)<bel(y.l));}
};
query q[N];// 查询数组
bitset<N> ans;// 答案数组
// cnt[i]: i 这个数在当前区间 [l,r] 出现次数，cf: 重复出现的数的数量。
// 如果 cf=0，[l,r] 中没有重复出现的数。
int cnt[N],cf=0;
// 移动区间
inline void add(int pos)// 添加 a[pos]
{
    cnt[a[pos]]++;// 将 a[pos] 的出现次数 +1。
    if(cnt[a[pos]]==2)cf++;// 如果已经出现两次，则重复了，cf++。
}
inline void del(int pos)// 删除 a[pos]
{
    cnt[a[pos]]--;// 将 a[pos] 的出现次数 -1。
    if(cnt[a[pos]]==1)cf--;// 如果当前只出现一次，则之前一定重复（出现两次），
    // 而现在不重复了，cf--。
}
```

算法代码：洛谷 P3901 数列找不同（模板题）【有其他解法】

```
void mt()
{
    S=int(ceil(pow(n,0.5)));// S=sqrt(n), 根号分块
    sort(q+1,q+m+1);// 结构体排序
    for(int i=1,l=1,r=0;i<=m;i++)// 遍历每个询问
    {
        #define q q[i]// 偷懒
        while(q.l<l)add(--l);// 向左扩展 l-1
        while(r<q.r)add(++r);// 向右扩展 r+1
        while(l<q.l)del(l++);// 向右删除 l
        while(q.r<r)del(r--);// 向左删除 r
        // 注意上面四句的顺序，需要先扩展在删除。
        // 同时注意自减自加运算符是前置还是后置。
        ans[q.id]=!cf;// 记录当前答案
        #undef q
    }
}

int main()
{
    // input
    mt();// 莫队
    for(int i=1;i<=m;i++)puts(ans[i]?"Yes":"No");// 输出
    return 0;
}
```

算法复杂度

下面的讨论中 $m = O(n)$。
单次移动 l, r 中的一个复杂度显然 $O(1)$。

算法复杂度

下面的讨论中 $m = O(n)$。
单次移动 l, r 中的一个复杂度显然 $O(1)$。
考虑 l, r 分别移动的次数。

算法复杂度

下面的讨论中 $m = O(n)$。
单次移动 l, r 中的一个复杂度显然 $O(1)$。
考虑 l, r 分别移动的次数。

- 考虑 l：设块 i 内的询问的左端点个数为 x_i，则块 i 中移动 l 的次数顶多 $x_i \times \sqrt{n}$。一共 $\sqrt{n}$ 个块，移动 l 的总时间复杂度为：

$$\begin{aligned}
&\sum_{i=1}^{\sqrt{n}} (x_i \times \sqrt{n}) \\
=&(\sqrt{n}) \times \sum_{i=1}^{\sqrt{n}} (x_i) \\
=&\sqrt{n} \times m \\
=&O(n\sqrt{n})
\end{aligned}$$

算法复杂度

- 考虑 r：每块内的 x_i 个 $l_j(1 \le j \le x_i)$ 肯定一一对应着 x_i 个 r_j。显然这 x_i 个 r_j 最多会使 r 移动 n 的长度（l_j 同一块时，按 r_j 升序，故不降）。
  一共 $\sqrt{n}$ 个块，移动 r 的总时间复杂度为：
  $\sqrt{n} \times n = O(n\sqrt{n})$。

算法复杂度

- 考虑 r：每块内的 x_i 个 $l_j(1 \le j \le x_i)$ 肯定一一对应着 x_i 个 r_j。显然这 x_i 个 r_j 最多会使 r 移动 n 的长度（l_j 同一块时，按 r_j 升序，故不降）。
一共 $\sqrt{n}$ 个块，移动 r 的总时间复杂度为：
$\sqrt{n} \times n = O(n\sqrt{n})$。
- 则总时间复杂度为 $O(1) \times [O(n\sqrt{n}) + O(n\sqrt{n})] = O(n\sqrt{n})$。

奇偶性排序

刚才的复杂度分析中提出了一些极端情况：x_i 个 r_j 最多使 r 移动 n 的长度。而 $\sqrt{n}$ 个块都可能使 r 移动 n 的长度，例如下列询问（**已排序且块长为** 4）：

```
1 1
2 100
3 1
4 100
```

奇偶性排序

刚才的复杂度分析中提出了一些极端情况：x_i 个 r_j 最多使 r 移动 n 的长度。而 $\sqrt{n}$ 个块都可能使 r 移动 n 的长度，例如下列询问（**已排序且块长为** 4）：

```
1 1
2 100
3 1
4 100
```

按原先的排序策略，r 需要反复横跳，十分浪费时间。如果将处理顺序改为以下顺序将大大加速：

```
1 1
2 100
4 100
3 1
```

奇偶性排序

怎么修改排序策略？

依然以 $bel[l_i]$ 为第一关键字升序排序，若 $bel[l_i]$ 为奇数，以 r_i 为第二关键字升序排序，反之若 $bel[l_i]$ 为偶数，以 r_i 为第二关键字降序排序。

奇偶性排序

怎么修改排序策略？

依然以 $bel[l_i]$ 为第一关键字升序排序，若 $bel[l_i]$ 为奇数，以 r_i 为第二关键字升序排序，反之若 $bel[l_i]$ 为偶数，以 r_i 为第二关键字降序排序。

通俗来讲：即对于属于奇数块的询问，r 按从小到大排序，对于属于偶数块的排序，r 从大到小排序。

这样我们的 r 指针在处理完这个奇数块的问题后，将在返回的途中处理偶数块的问题，再向 n 移动处理下一个奇数块的问题，优化了 r 指针的移动次数，一般情况下，这种优化能让程序快 30% 左右。——OI-Wiki。

奇偶性排序

怎么修改排序策略？

依然以 $bel[l_i]$ 为第一关键字升序排序，若 $bel[l_i]$ 为奇数，以 r_i 为第二关键字升序排序，反之若 $bel[l_i]$ 为偶数，以 r_i 为第二关键字降序排序。

通俗来讲：即对于属于奇数块的询问，r 按从小到大排序，对于属于偶数块的排序，r 从大到小排序。

这样我们的 r 指针在处理完这个奇数块的问题后，将在返回的途中处理偶数块的问题，再向 n 移动处理下一个奇数块的问题，优化了 r 指针的移动次数，一般情况下，这种优化能让程序快 30% 左右。——OI-Wiki。

实测：810 ms $\rightarrow$ 622 ms。快约 23.2%。

奇偶性排序：代码

```
struct query
{
    int l,r,id;
    friend inline bool operator < (query x,query y)
    {
        if(bel(x.l)!=bel(y.l))return bel(x.l)<bel(y.l);
        if(bel(x.l)&1)return x.r<y.r;
        else return x.r>y.r;
    }
};
```

常数级展开

发现 `add()`，`del()` 两个函数可以压行并展开到 `mt()` 中。
这看似鸡肋的优化实测 572 ms——又优化了 50 ms。

常数级展开

发现 `add()`，`del()` 两个函数可以压行并展开到 `mt()` 中。
这看似鸡肋的优化实测 572 ms——又优化了 50 ms。
代码：

```
void mt()
{
    S=int(ceil(pow(n,0.5)));
    sort(q+1,q+m+1);
    for(int i=1,l=1,r=0;i<=m;i++)
    {
        #define q q[i]
        // 压行并展开:
        while(q.l<l)cf+=(++cnt[a[--l]]==2);// 与 add(--l) 等价
        while(r<q.r)cf+=(++cnt[a[++r]]==2);// 与 add(++r) 等价
        while(l<q.l)cf-=(--cnt[a[l++]]==1);// 与 del(l++) 等价
        while(q.r<r)cf-=(--cnt[a[r--]]==1);// 与 del(r--) 等价
        ans[q.id]=!cf;
        #undef q
    }
}
```

玄学剪枝

我考虑到有时候可能转移大半天还不如暴力重新算，所以想出了一个玄学剪枝：

```
// 如果转移代价大于重新算的代价
if(abs(q.l-l)+abs(q.r-r)>(r-l+1)+(q.r-q.l+1))
{
    while(l<=r)cf-=(--cnt[a[r--]]==1);// 清除
    r=(l=q.l)-1;// 直接跳转
}
```

（这段语句加在 `#define q q[i]` 后面。）
没想到只优化了 1 ms（悲。也许是每次都判断的代价太大，抵消了直接跳转的优化。

预处理 *bel*

我写好几题才发现其他大佬都预处理了 *bel*，于是赶紧改过来。快了 58 ms。

```
int S;
int bel[N];
inline void initbel()// 1~S: 1
{
    S=int(ceil(pow(n,0.5)));// S=sqrt(n)
    for(int i=1;i<=n;i++)bel[i]=(i-1)/S+1;
}
...
void mt()
{
    initbel();
    sort(q+1,q+m+1);
    ...
}
```

例题：DQUERY - D-query

简要题意：给出一个长度为 n 的数列 a，m 个询问，每次询问给出数对 l, r 表示查询区间 $[l, r]$ 中有多少不同的数。
数据范围：$n \leq 3 \times 10^5, m \leq 2 \times 10^5, a_i \leq 10^6$。

例题：DQUERY - D-query

简要题意：给出一个长度为 n 的数列 a，m 个询问，每次询问给出数对 l, r 表示查询区间 $[l, r]$ 中有多少不同的数。
数据范围：$n \leq 3 \times 10^5, m \leq 2 \times 10^5, a_i \leq 10^6$。
板子题，难度在于如何 $O(1)$ 转移答案。

例题：DQUERY - D-query

简要题意：给出一个长度为 n 的数列 a，m 个询问，每次询问给出数对 l, r 表示查询区间 $[l, r]$ 中有多少不同的数。
数据范围：$n \le 3 \times 10^5, m \le 2 \times 10^5, a_i \le 10^6$。
板子题，难度在于如何 $O(1)$ 转移答案。
考虑用数组 cnt_i 表示 $[l, r]$ 中 i 出现了几次，变量 bt 表示 $[l, r]$ 中有多少不同的数。
要添加 a_{pos}，那么 $cnt[a_{pos}] \leftarrow cnt[a_{pos}] + 1$。
此时若 $cnt[a_{pos}] = 1$，即多了一个不同的数，那么 $bt \leftarrow bt + 1$。
同理删除 a_{pos} 时 $cnt[a_{pos}] \leftarrow cnt[a_{pos}] - 1$，若 $cnt[a_{pos}] = 0$（少了一个数），$bt \leftarrow bt - 1$。
其他的正常地跑莫队即可。**但此题似乎卡常。**

例题：P2709 小 B 的询问

简要题意：给出一个长度为 n 的数列 a（值域 $[1,k]$），m 个询问，每次询问给出数对 l,r 表示查询：

$$\sum_{i=1}^{k} c_i^2$$

其中 c_i 表示数字 i 在 $[l,r]$ 中的出现次数。
数据范围：$1 \le n,m,k \le 5 \times 10^4$。

例题：P2709 小 B 的询问

简要题意：给出一个长度为 n 的数列 a（值域 $[1,k]$），m 个询问，每次询问给出数对 l,r 表示查询：

$$\sum_{i=1}^{k} c_i^2$$

其中 c_i 表示数字 i 在 $[l,r]$ 中的出现次数。
数据范围：$1 \le n,m,k \le 5 \times 10^4$。
难度依然在于如何 $O(1)$ 转移答案。

例题：P2709 小 B 的询问

简要题意：给出一个长度为 n 的数列 a（值域 $[1,k]$），m 个询问，每次询问给出数对 l,r 表示查询：

$$\sum_{i=1}^{k} c_i^2$$

其中 c_i 表示数字 i 在 $[l,r]$ 中的出现次数。
数据范围：$1 \le n,m,k \le 5\times 10^4$。
难度依然在于如何 $O(1)$ 转移答案。
c 数组很好维护，但答案（设它为 s）就不那么好维护了。
由于每次添加或删除数时只会改变 $c_{a[pos]}$，而且只会 ± 1。所以：

例题：P2709 小 B 的询问

由

$$s=\sum_{i=1}^{k} c_i^2 = c_1^2+\cdots+c_{a[pos]}^2+\cdots+c_k^2$$

可得

$$\begin{aligned} s' &= c_1^2+\cdots+(c_{a[pos]}+1)^2+\cdots+c_k^2 \\ &= c_1^2+\cdots+c_{a[pos]}^2 \pm 2\times c_{a[pos]}+1+\cdots+c_k^2 \\ &= s \pm 2\times c_{a[pos]}+1 \end{aligned}$$

转移时修改即可。

目录

如何实现带修莫队？

发现普通莫队不支持修改，那么如何使它支持修改操作呢？
考虑存询问时加一个变量 t_i 表示**进行此询问时前面修改了几次。**
同时存下每一个修改操作（无需排序）。
再新增一个指针 t 表示当前区间所在的时间位置。那么移动方向就从 4 个变为 6 个：
$[l-1,r,t],[l,r+1,t],[l+1,r,t],[l,r-1,t],[l,r,t-1],[l,r,t+1]$。
新增的两个为时间轴上的移动。

例题：P1903 [国家集训队] 数颜色 / 维护队列

简要题意：给出一个长度为 n 的数列，m 个操作，要求支持两种操作：查询区间有多少不同的数、单点修改。
数据范围：$n, m \leq 1.33333 \times 10^5, a_i \leq 10^6$。

例题：P1903 [国家集训队] 数颜色 / 维护队列

简要题意：给出一个长度为 n 的数列，m 个操作，要求支持两种操作：查询区间有多少不同的数、单点修改。
数据范围：$n, m \le 1.33333 \times 10^5, a_i \le 10^6$。
板子题，直接上代码：

```
constexpr int N=214514,A=1145141;// A: a 的值域
int n,m,S,qm,a[N];// qm: 询问的个数
inline int bel(int x){return (x-1)/S+1;}// 分块
struct query
{
    int l,r,t,id;// 额外记录时间
    friend inline bool operator < (query x,query y)
    {//       若 l 所在块不同    按 l 的块的编号 否则 若 r 所在块不同 按 r 的块的编号 否则按时间排
        return (bel(x.l)^bel(y.l)?bel(x.l)<bel(y.l):(bel(x.r)^bel(y.r)?bel(x.r)<bel(y.r):x.t<y.t));
    }
};query q[N];
struct modify// 新增：修改操作
{int p,v;};modify mo[N];
```

例题：P1903 [国家集训队] 数颜色 / 维护队列

```
int cnt[A],bt,ans[N];// 类似于普通莫队
void mt()
{
    S=int(ceil(pow(n,0.66)));// 这里块长需要调整，具体可以可以看
    // https://oi-wiki.org/misc/modifiable-mo-algo/ 中的证明
    sort(q+1,q+qm+1);
    for(int i=1,l=1,r=0,t=0;i<=qm;i++)
    {
        #define q q[i]
        #define p mo[t].p
        #define v mo[t].v
        while(q.l<l)bt+=(!(cnt[a[--l]]++));// 类似
        while(r<q.r)bt+=(!(cnt[a[++r]]++));
        while(l<q.l)bt-=(!(--cnt[a[l++]]));
        while(q.r<r)bt-=(!(--cnt[a[r--]]));
        // 可怕的压行：【需要当场解释】
        while(t<q.t){t++;if(l<=p&&p<=r)bt-=(!(--cnt[a[p]])-!(cnt[v]++));swap(a[p],v);}
        while(q.t<t){if(l<=p&&p<=r)bt-=(!(--cnt[a[p]])-!(cnt[v]++));swap(a[p],v);t--;}
        ans[q.id]=bt;
        #undef q
        #undef p
        #undef v
    }
}
```

目录

简介

对于某些题目，普通莫队可能难以解决。如：AT_joisc2014_c 歴史の研究。

此题添加数很方便，但删除数却很麻烦。因为当最大值改变（如变小）时，我们无法确定新的最大值。

这时就要用到回滚莫队了。当删除和添加只有一个可实现时，就只用一个操作，剩下的就回滚解决。

简介

对于某些题目，普通莫队可能难以解决。如：AT_joisc2014_c 歴史の研究。
此题添加数很方便，但删除数却很麻烦。因为当最大值改变（如变小）时，我们无法确定新的最大值。
这时就要用到回滚莫队了。当删除和添加只有一个可实现时，就只用一个操作，剩下的就回滚解决。
当然还有树上莫队、二维莫队，由于难度过高，~~我自己也不会，~~就不多做介绍了。可以通过 OI-Wiki 自学。

目录

Thanks!

目录

此 PPT 是本人一边学习莫队一边写的，肯定有诸多不足，还请包容。

树上莫队和二维莫队