专题：区间信息维护与查询——分块

一、分块原理

树状数组和线段树虽然非常方便，但维护的信息必须满足信息合并特性（如：区间可加、可减），若不满足此特性，则不可以使用树状数组和线段树。分块算法可以维护一些线段树维护不了的内容，它其实就是优化过后的暴力算法。分块可以解决几乎所有区间更新和区间查询问题，但效率相对于线段树等数据结构要差一些。

分块算法是将所有数据都分为若干块，维护块内信息，使得块内查询为 O(1)时间，而总询问可被看作若干块询问的总和。

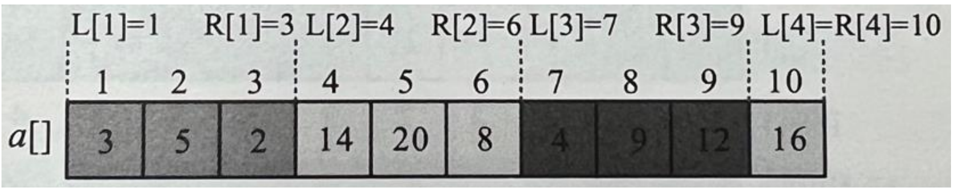
分块算法将长度为 n 的序列分成若干块，每一块都有 k 个元素，最后一块可能少于 k 个元素。为了使时间复杂度均摊，通常将块的大小设为 $k=\sqrt{n}$ ，用 pos[i] 表示第 i 个位置所属的块，对每个块都进行信息维护。分块可以解决以下问题。

- 单点更新：一般先将对应块的懒标记下传，再暴力更新块的状态，时间复杂度为 $O(\sqrt{n})$。
- 区间更新：若区间更新横跨若干块，则只需对完全覆盖的块打上懒标记，最多需要修改两端的两个块，对两端剩余的部分暴力更新块的状态。每次更新都最多遍历$\sqrt{n}$个块，遍历每个块的时间复杂度都是 O(1)，两端的两个块暴力更新$\sqrt{n}$次，总的时间复杂度是$O(\sqrt{n})$。
- 区间查询：和区间更新类似，对中间跨过的整个块直接利用块存储的信息统计答案，对两端剩余的部分可以暴力扫描统计。时间复杂度和区间修改一样，也是 $O(\sqrt{n})$。

将整个段分成多个块后进行修改或查询时，对完全覆盖的块直接进行修改，像线段树一样标记或累加；对两端剩余的部分进行暴力修改。分块算法遵循“大段维护、局部朴素”的原则。

1. 预处理

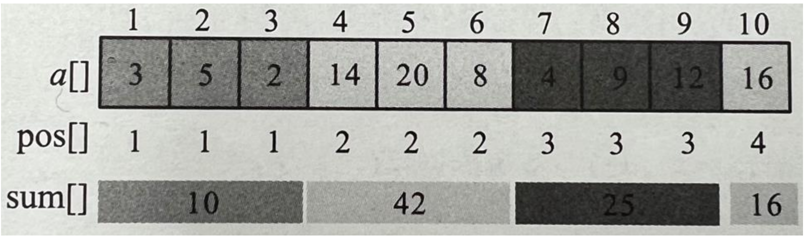
（1）将序列分块，然后将每个块都标记左右端点 L[i]和 R[i]，对最后一块需要特别处理。n=10，$t=\sqrt{n}=3$ ，每 3 个元素为一块，一共分为 4 块，最后一块只有一个元素。



算法代码：

```
t=sqrt(n*1.0);
int num=n/t;
if(n%t) num++;
for(int i=1;i<=num;i++)
{
    L[i]=(i-1)*t+1;  //每一块的左端点
    R[i]=i*t;          //每一块的右端点
}
R[num]=n;
```

（2）用 pos[]标记每个元素所属的块，用 sum[] 累加每一块的和值。



算法代码：

```
for(int i=1;i<=num;i++)
   for(int j=L[i];j<=R[i];j++)
   {
       pos[j]=i;       //表示属于哪个块
       sum[i]+=a[j] ; //计算每块的和值
   }
```

2．区间更新

区间更新，例如将[l,r]区间的元素都加上 d。

（1）求 l 和 r 所属的块，p=pos[l]，q=pos[r]。

（2）若属于同一块（p=q），则对该区间的元素进行暴力修改，同时更新该块的和值。

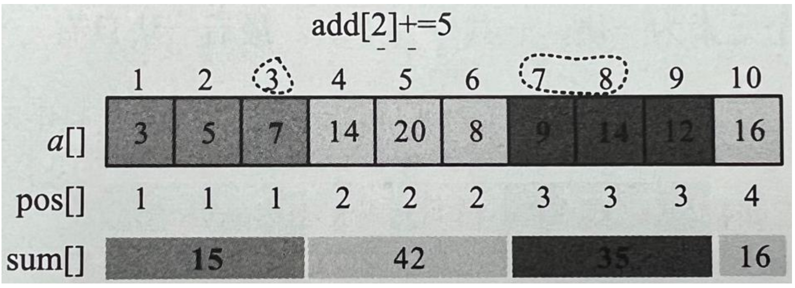
（3）若不属于同一块， 则对中间完全覆盖的块打上懒标记，add[i]+=d，对首尾两端的元素进行暴力修改。

进行暴力修改。

例如，将[3,8]区间的元素都加上 5，操作过程：

①读取 3 和 8 所属的块 p=pos[3]=1， q=pos[8]=3，不属于同一块，中间的完整块[p+1，q-1]为第 2 块， 为该块打上懒标记 add[2] +=5;

②对首尾两端的元素(下标 3、7、8)进行暴力修改，并修改和值。



算法代码：

```
void change(int l,int r,long long d) //[l,r]区间的元素加 d
{
    int p=pos[l],q=pos[r];           //读取所属的块
    if(p==q)                         //在同一块中
    {
        for(int i=l;i<=r;i++)        //暴力修改
            a[i]+=d;
        sum[p]+=d*(r-l+1);           //修改和值
    }
    else
    {
        for(int i=p+1;i<=q-1;i++)    //对中间完全覆盖的块打懒标记
            add[i]+=d;
        for(int i=l;i<=R[p];i++)     //左端暴力修改
            a[i]+=d;
        sum[p]+=d*(R[p]-l+1);        //修改和值
        for(int i=L[q];i<=r;i++)     //右端暴力修改
            a[i]+=d;
        sum[q]+=d*(r-L[q]+1);        //修改和值
    }
}
```

3. 区间查询

区间查询，例如查询[l,r]区间的元素和值。

（1）求 l 和 r 的所属块，p=pos[l]，q=pos[r]。

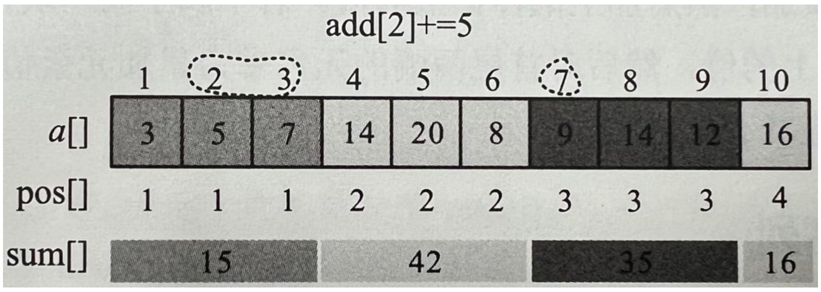
（2）若属于同一块（p=q），则对该区间的元素进行暴力累加，然后加上懒标记上的值。

（3）若不属于同一块，则对中间完全覆盖的块累加 sum[] 值和懒标记上的值，然后对首尾两端暴力累加元素值及懒标记值。

例如， 查询[2,7]区间的元素和值，操作过程：

①读 p=pos[2]=1，q=pos[7]=3，不属于同一块，则中间的完整块[p+1,q-1]为第 2 块，ans+=sum[2]+add[2]×(R[2]-L[2]+1)=42+5×3=57；

②对首尾两端的元素暴力累加元素值及懒标记值。此时懒标记 add[1]=add[3]=0，ans+=5+7+add[1]×(3-2+1)+9+add[3]×(7-7+1)=78。



算法代码：

```
ll ask(int l,int r)                                    //区间查询
{
    int p=pos[l],q=pos[r];
    ll ans=0;
    if(p==q)                                           //在同一块中
    {
        for(int i=l;i<=r;i++)                          //累加
            ans+=a[i];
        ans+=add[p]*(r-l+1);                           //计算懒标记
    }
    else
    {
        for(int i=p+1;i<=q-1;i++)                      //累加中间段落
            ans+=sum[i]+add[i]*(R[i]-L[i]+1);
        for(int i=l;i<=R[p];i++)                       //左端暴力累加
            ans+=a[i];
        ans+=add[p]*(R[p]-l+1);
        for(int i=L[q];i<=r;i++)                       //右端暴力累加
            ans+=a[i] ;
        ans+=add[q]*(r-L[q]+1);
    }
    return ans;
}
```

例题 1：简单的整数问题（POJ3468）

【题目描述】

有 N 个整数，$A_1,A_2,\cdots,A_N$,你需要对其进行两种类型的操作——

第一种类型的操作是：对给定区间中的每个数字都添加一个给定的数字。

第二种类型的操作是：查询给定区间中所有数字的总和。

【输入格式】

第 1 行包含两个整数 N 和 Q（$1\leqslant N,Q\leqslant 10^5$）。

第 2 行包含 N 个整数，为 $A_1,A_2,\cdots,A_N$,的初始值（$-10^9\leqslant A_i\leqslant 10^9$）。

接下来有 Q 行，每行表示一个操作：“C a b c”表示将 $A_a,A_{a+1},\cdots,A_b$ 中每一个数字都加 c（$-10^4\leqslant c\leqslant 10^4$）；“Q a b”表示查询 $A_a,A_{a+1},\cdots,A_b$ 中所有数字的总和。

【输出格式】

对每个查询，都单行输出区间和的值。

【输入输出样例】

输入样例	输出样例
10 5 1 2 3 4 5 6 7 8 9 10 Q 4 4 Q 1 10 Q 2 4 C 3 6 3 Q 2 4	4 55 9 15

【提示】

答案可能超过 32 位整数的范围。

【参考代码】

```
#include<cstdio>
#include<algorithm>
#include<cmath>
#define ll long long
#define N 100010
using namespace std;

ll a[N],sum[N],add[N];
int L[N],R[N],d;
int pos[N];
int n,m,t,l,r;
char op[3];
```

```
void build()
{
    t=sqrt(n*1.0); //注意没有 sqrt(int)，但是返回值可以为 int
                   //选择 G++提交,否则 int 型做参数会提示编译问题
    int num=n/t;
    if(n%t)  num++;
    for(int i=1;i<=num;i++)
    {
        L[i]=(i-1)*t+1;            //每块的左右
        R[i]=i*t;
    }
    R[num]=n;
    for(int i=1;i<=num;i++)
        for(int j=L[i];j<=R[i];j++)
        {
            pos[j]=i;              //表示属于哪个块
            sum[i]+=a[j];          //计算每块和值
        }
}

void change(int l,int r,long long d) //区间[l,r]加上 d
{
    int p=pos[l],q=pos[r];         //读所属块
    if(p==q)                       //在一块中
    {
        for(int i=l;i<=r;i++)      //暴力修改
            a[i]+=d;
        sum[p]+=d*(r-l+1);         //修改和值
    }
    else
    {
        for(int i=p+1;i<=q-1;i++)  //中间完全覆盖块打懒标记
            add[i]+=d;
        for(int i=l;i<=R[p];i++)   //左端暴力修改
            a[i]+=d;
        sum[p]+=d*(R[p]-l+1);
        for(int i=L[q];i<=r;i++)   //右端暴力修改
            a[i]+=d;
        sum[q]+=d*(r-L[q]+1);
    }
}
```

```
ll query(int l,int r)
{
    int p=pos[l],q=pos[r];
    ll ans=0;
    if(p==q)                          //在一块中
    {
        for(int i=l;i<=r;i++)         //累加
            ans+=a[i];
        ans+=add[p]*(r-l+1);          //计算懒标记
    }
    else
    {
        for(int i=p+1;i<=q-1;i++)              //累加中间块
            ans+=sum[i]+add[i]*(R[i]-L[i]+1);
        for(int i=l;i<=R[p];i++)               //左端暴力累加
            ans+=a[i];
        ans+=add[p]*(R[p]-l+1);
        for(int i=L[q];i<=r;i++)               //右端暴力累加
            ans+=a[i];
        ans+=add[q]*(r-L[q]+1);
    }
    return ans;
}

int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
        scanf("%lld",&a[i]);
    build();
    for(int i=1;i<=m;i++)
    {
        scanf("%s %d %d",op,&l,&r);
        if(op[0]=='C')
        {
            scanf("%d",&d);
            change(l,r,d);
        }
        else
            printf("%lld\n",query(l,r));
    }
    return 0;
}
```

例题 2：超级马里奥（HDU4417 Super Mario）

https://acm.dingbacode.com/showproblem.php?pid=4417

【题目描述】

马里奥是世界著名的水管工。他的“魁梧”身材和惊人的跳跃能力让我们记忆犹新。现在可怜的公主又陷入了困境，马里奥需要拯救他的情人。我们将通往老板城堡的道路视为一条直线（长度为 N），在每个整数点 i 上都有一块砖，高度为 h_i。现在的问题是，如果马里奥能跳的最大高度是 H，他能在[L,R]区间击中多少块砖头。

【输入格式】

第一行是一个整数 T，为测试数据的组数。

对于每组测试数据：

第一行包含两个整数 N 和 M（$1 \leqslant N \leqslant 10^5$，$1 \leqslant M \leqslant 10^5$），N 是道路长度，M 是查询次数。

第二行包含 N 个整数，表示每块砖的高度，范围为[0,1000000000]。

接下来的 M 行，每行包含三个整数 L、R、H。（$0 \leqslant L \leqslant R < N$；$0 \leqslant H \leqslant 1000000000$。）

注意：询问中 L 和 R 是从 0 开始编号的！

【输出格式】

对于每组测试数据，第一行输出“Case X:”（X 是从 1 开始的测试数据组的编号）。

接下来有 M 行，每行包含一个整数，第 i 个整数是马里奥在第 i 个查询中可以击中的砖块数。

【输入输出样例】

输入样例	输出样例
1 10 10 0 5 2 7 5 4 3 8 7 7 2 8 6 3 5 0 1 3 1 1 9 4 0 1 0 3 5 5 5 5 1 4 6 3 1 5 7 5 7 3	Case 1: 4 0 0 3 1 2 0 1 5 1

【题目来源】

2012 ACM/ICPC Asia Regional Hangzhou Online

【算法分析】

本题为区间查询问题，查询[l,r]区间小于或等于 h 的元素个数，可以采用分块的方法解决。

1．算法步骤

（1）分块。划分块并对每一块进行非递减排序。在辅助数组 temp[]上排序，原数组不变。

（2）查询。查询[l,r]区间小于或等于 h 的元素个数。

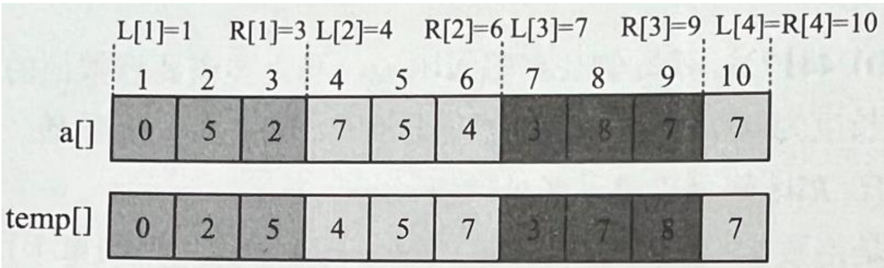
若该区间属于同一块，则暴力累加块内小于或等于 h 的元素个数。

若该区间包含多个块，则累加中间每一块小于或等于 h 的元素个数，此时可以用 upper_bound() 函数统计，然后暴力累加左端和右端小于或等于 h 的元素个数。

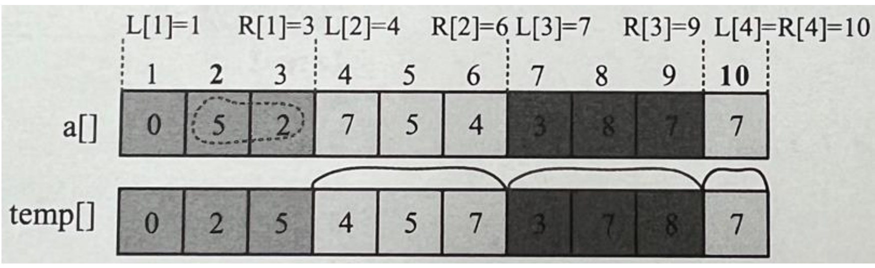
2．图解分析

根据测试用例的输入数据，分块算法的求解过程如下。

（1）分块。n=10，t=$\sqrt{n}$=3，每 3 个元素为一块，一共分为 4 块，最后一块只有一个元素。原数组 a[]和每一块排序后的辅助数组 temp[]如下图所示。



（2）查询。1 9 4：因为题目中的下标从 0 开始，上图中的下标从 1 开始，所以实际上是查询[2,10]区间高度小于或等于 4 的元素个数。[2,10]区间跨 4 个块，左端第 1 个块没有完全包含，需要暴力统计 a[2]、a[3]小于或等于 4 的元素。后面 3 个块是完整的块，对完整的块可以直接用 upper_bound()函数在 temp 数组中统计，该函数利用有序性进行二分查找， 效率较高。



upper_bound(begin,end,num)：从数组的 begin 位置到 end-1 位置二分查找第 1 个大于 num 的数字，若找到，则返回该数字的地址，否则返回 end。将返回的地址减去起始地址 begin，即可得到小于或等于 num 的元素个数。

区别：

lower_bound(begin,end,num)：从数组的 begin 位置到 end-1 位置二分查找第一个大于或等于 num 的数字，找到返回该数字的地址，不存在则返回 end。

upper_bound(begin,end,num)：从数组的 begin 位置到 end-1 位置二分查找第一个大于 num 的数字，找到返回该数字的地址，不存在则返回 end。

通过返回的地址减去起始地址 begin,得到找到数字在数组中的下标。

【参考代码】

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

const int maxn=1e5+10;
int L[maxn],R[maxn],belong[maxn];
int a[maxn],temp[maxn],n,m;

void build()
{
    int t=sqrt(n);
    int num=n/t;
    if(n%num) num++;
    for(int i=1;i<=num;i++)
        L[i]=(i-1)*t+1,R[i]=i*t;
    R[num]=n;
    for(int i=1;i<=n;i++)
        belong[i]=(i-1)/t+1;
    for(int i=1;i<=num;i++)
        sort(temp+L[i],temp+1+R[i]);                    //每块排序
}

int query(int l,int r,int h)
{
    int ans=0;
    if(belong[l]==belong[r])
    {
        for(int i=l;i<=r;i++)
            if(a[i]<=h) ans++;
    }
    else
    {
        for(int i=l;i<=R[belong[l]];i++)                //左端
            if(a[i]<=h) ans++;
        for(int i=belong[l]+1;i<belong[r];i++)          //中间
            ans+=upper_bound(temp+L[i],temp+R[i]+1,h)-temp-L[i];
        for(int i=L[belong[r]];i<=r;i++)                //右端
            if(a[i]<=h) ans++;
    }
    return ans;
}
```

```
int main()
{
    int T;
    scanf("%d",&T);
    for(int cas=1;cas<=T;cas++)
    {
        scanf("%d%d",&n,&m);
        for(int i=1;i<=n;i++)
        {
            scanf("%d",&a[i]);
            temp[i]=a[i];
        }
        build();
        printf("Case %d:\n",cas);
        while(m--)
        {
            int l,r,h;
            scanf("%d%d%d",&l,&r,&h);
            printf("%d\n",query(++l,++r,h));
        }
    }
    return 0;
}
```

例题 3：蒲公英（洛谷 P4168）

https://www.luogu.com.cn/problem/P4168

【题目描述】

在乡下的小路旁种着许多蒲公英，而我们的问题正是与这些蒲公英有关。

为了简化起见，我们把所有的蒲公英看成一个长度为 n 的序列 $\{a_1,a_2,\cdots,a_n\}$，其中 a_i 为一个正整数，表示第 i 棵蒲公英的种类编号。

而每次询问一个区间 $[l,r]$，你需要回答区间里出现次数最多的是哪种蒲公英，如果有若干种蒲公英出现次数相同，则输出种类编号最小的那个。

注意，你的算法必须是在线的。

【输入格式】

第一行有两个整数，分别表示蒲公英的数量 n 和询问次数 m。

第二行有 n 个整数，第 i 个整数表示第 i 棵蒲公英的种类 a_i。

接下来 m 行，每行两个整数 l_0 和 r_0，表示一次询问。输入是加密的，解密方法如下：

令上次询问的结果为 x（如果这是第一次询问，则 x=0），设 $l=((l_0+x-1) \bmod n)+1$，$r=((r_0+x-1) \bmod n)+1$。如果 $l>r$，则交换 l 和 r。最终的询问区间为计算后的 $[l,r]$。

【输出格式】

对于每次询问，输出一行一个整数表示答案。

【输入输出样例】

输入样例	输出样例
6 3 1 2 3 2 1 2 1 5 3 6 1 5	1 2 1

【数据规模与约定】

对于 20%的数据，保证 $n,m\leqslant 3000$。

对于 100% 的数据，$1\leqslant n\leqslant 40000$，$1\leqslant m\leqslant 50000$，$1\leqslant a_i\leqslant 10^9$，$1\leqslant l_0,r_0\leqslant n$。

【算法分析】

本题目是询问区间众数且强制在线。

若题目是询问区间是否有过半众数，就是主席树，按值域建树，不断判断左右子树子节点数量大于(r-l+1)/2，如果一直可以到叶子节点，则 return true，否则 return false。

若题目是询问区间是否有过半众数且带修改，就是树套树。

若题目是询问区间众数且可以离线，就可以用莫队做，信息只增不删，开一个桶（离散化）维护各数出现次数，时间复杂度：$O(n\sqrt{n})$

然而，这道题是询问区间众数且强制在线。

那我们可以考虑分块做法，n≤40000，是符合$O(n\sqrt{n})$的。

思考：对于一个区间，这个区间存在两种情况

1. [L,R]都由整块构成
2. [L,R]由整块和一些零散的点构成

我们可以预处理出这样的两个数组

sum[i][j]从最开始到第 i 块 j 这种颜色出现的次数，p[i][j]第 i 块到第 j 块的众数

处理 p 数组时有一个小技巧，保证时间。可以这样处理：

```
for(int i=1;i<=num;++i)
{
    int tot=0,col=INF;
    for(int j=l[i];j<=n;++j)
    {
        int pos=id[j];
        tong[a[j]]++;  //开桶处理 p 数组
        if(tong[a[j]]>tot||(tong[a[j]]==tot&&col>a[j]))
           tot=tong[a[j]],col=a[j];
        p[i][pos]=col;
    }
    for(int j=l[i];j<=n;++j)tong[a[j]]=0;
}
```

每一次在统计的时候，先找到整块的众数，前后的残余的点开一个桶暴力判断即可（每一次桶记得清空）记得离散化

【参考代码】

```
#include<bits/stdc++.h>
#define INF 2100000001
#define N 40003
#define LL long long
using namespace std;

int n,m,num,pp;
int a[N],b[N],l[202],r[202],id[N],color[N],tong[N],ans[50003];
int ge[202][202],p[202][202],sum[202][N];
            //sum[i][j]从最开始到第 i 块 j 这种颜色出现的次数，p[i][j]第 i 块到第 j 块的众数，id[i]i 所在的块

int read()
{
    int x=0,f=1;char s=getchar();
    while(s<'0'||s>'9'){if(s=='-')f=-1;s=getchar();}
    while(s>='0'&&s<='9'){x=x*10+s-'0';s=getchar();}
    return x*f;
}

void init()
{
    int base=sqrt(1.0*n);
    for(int i=1;i*base<n;++i)
    {
        int tot=0,col=INF;
        l[i]=base*(i-1)+1;
        r[i]=base*i;
        for(int j=l[i];j<=r[i];++j)
        {
            id[j]=i;
            sum[i][a[j]]++;//在第 i 块 a[j]颜色的个数
        }
        num=i;
    }
    num++;
    l[num]=base*(num-1)+1;r[num]=n;
    for(int j=l[num];j<=r[num];++j)id[j]=num,sum[num][a[j]]++;/单纯分块，最后一块单独处理
    for(int i=1;i<=num;++i)
      for(int j=1;j<=pp;++j)
        sum[i][j]+=sum[i-1][j];                                    //前缀和处理 sum
```

```
    for(int i=1;i<=num;++i)
    {
        int tot=0,col=INF;
        for(int j=l[i];j<=n;++j)
        {
            int pos=id[j];
            tong[a[j]]++;                                   //开桶处理 p 数组
            if(tong[a[j]]>tot||(tong[a[j]]==tot&&col>a[j]))
               tot=tong[a[j]],col=a[j];
            p[i][pos]=col;
        }
        for(int j=l[i];j<=n;++j)tong[a[j]]=0;
    }
}

int query(int L,int R)
{
    memset(tong,0,sizeof(tong));
    int pos1=id[L],pos2=id[R];
    if(pos2-pos1<=1)                                    //两块及以下暴力处理
    {
        int answer=INF,tot=0;
        for(int i=L;i<=R;++i)
        {
            ++tong[a[i]];
            if(tong[a[i]]>tot||(tong[a[i]]==tot&&answer>a[i]))
              tot=tong[a[i]],answer=a[i];
        }
        return answer;
    }
    int answer=p[pos1+1][pos2-1];                       //先取出整块的众数
    int tot=sum[pos2-1][answer]-sum[pos1][answer];
    int x=L,y=l[pos1+1]-1;                              //前面残余的
    for(int i=x;i<=y;++i)
    {
        ++tong[a[i]];
        if(tong[a[i]]+sum[pos2-1][a[i]]-sum[pos1][a[i]]>tot||(tong[a[i]]+sum[pos2-1][a[i]]-sum[pos1][a[i]]==tot&&answer>a[i]))
          tot=tong[a[i]]+sum[pos2-1][a[i]]-sum[pos1][a[i]],answer=a[i];
    }
    x=l[pos2],y=R;                                      //后面残余的
```

```
    for(int i=x;i<=y;++i)
    {
        ++tong[a[i]];
        if(tong[a[i]]+sum[pos2-1][a[i]]-sum[pos1][a[i]]>tot||(tong[a[i]]+sum[pos2-1][a[i]]-sum[pos1][a[i]]==tot&&answer>a[i]))
            tot=tong[a[i]]+sum[pos2-1][a[i]]-sum[pos1][a[i]],answer=a[i];
    }
    return answer;
}

int main()
{
    n=read(),m=read();
    for(int i=1;i<=n;++i)
        a[i]=b[i]=read();
    sort(b+1,b+1+n);
    pp=unique(b+1,b+1+n)-b-1;
    for(int i=1;i<=n;++i)
    {
        int col=a[i];
        a[i]=lower_bound(b+1,b+1+pp,a[i])-b;
        color[a[i]]=col;                                    //离散化
    }
    init();
    for(int i=1;i<=m;++i)
    {
        int L=read(),R=read();
        int x=(L+ans[i-1]-1)%n+1;
        int y=(R+ans[i-1]-1)%n+1;
        if(x>y)swap(x,y);
        ans[i]=color[query(x,y)];
        printf("%d\n",ans[i]);
    }
    return 0;
}
```