莫队算法1

普通莫队

算法模型

只有询问操作，共 m 个询问，每个询问操作都是一个区间 $[l_i,r_i]$，询问区间范围为 $[1,n]$。

可以 $O(1)$ 根据当前维护区间 $[l,r]$ 更新到 $[l-1,r],[l,r+1],[l+1,r],[l,r-1]$。

则利用莫队可以 $O(n\sqrt m)$ 离线处理所有询问。

算法实现

先对 $[1,n]$ 进行分块，假设每块长度为 S，先将 $kS\lt l_i\le (k+1)S$ 的询问丢到同一个块。

对同一个块，根据 r_i 排序，然后依次处理排完序的每个询问，同时用两个指针维护当前区间。

首先对每个块，r_i 单调，于是每个块移动右指针的复杂度为 $O(n)$，移动右指针的总复杂度为 $O\left(\frac {n^2}S\right)$。

同时每个左指针每次只能在所在块中移动，于是每个询问左指针的复杂度为 $O(S)$，移动左指针的总复杂度为 $O(mS)$。

于是总复杂度为 $O\left(\frac {n^2}S+mS\right)$，令 $S\sim O\left(\frac n{\sqrt m}\right)$，则总复杂度为 $O(n\sqrt m)$。

注意 每次移动指针时要先拓展指针对应的区间再缩减指针对应区间，否则对应区间长度可能会变成负数产生各种 bug。

算法例题

洛谷p1494

题意

给定一个长度为 n 的序列，每个询问给定一个区间 $[l_i,r_i]$，询问从该区间的序列中任意取两个数，这两个数相同的概率。

题解

双指针维护区间中的每个值的个数，同时维护当前所有使得两数相同的方案数即可。

```
const int MAXN=5e4+5;
```

```
int blk_sz,a[MAXN],col[MAXN];
struct query{
    int l,r,idx;
    bool operator < (const query &b)const{
        if(l/blk_sz!=b.l/blk_sz)return l<b.l;
        return r<b.r;
    }
}q[MAXN];
LL ans1[MAXN],ans2[MAXN],cur;
LL gcd(LL a,LL b){
    while(b){
        LL t=b;
        b=a%b;
        a=t;
    }
    return a;
}
void add(int v){
    cur+=col[v];
    col[v]++;
}
void del(int v){
    col[v]--;
    cur-=col[v];
}
int main()
{
    int n=read_int(),m=read_int();
    blk_sz=1.0*n/sqrt(m)+1;
    _rep(i,1,n)
    a[i]=read_int();
    _for(i,0,m)
    q[i].l=read_int(),q[i].r=read_int(),q[i].idx=i;
    sort(q,q+m);
    int lef=1,rig=0;
    _for(i,0,m){
        if(q[i].l==q[i].r){
            ans1[q[i].idx]=0;
            ans2[q[i].idx]=1;
            continue;
        }
        while(lef>q[i].l)add(a[--lef]);
        while(rig<q[i].r)add(a[++rig]);
        while(lef<q[i].l)del(a[lef++]);
        while(rig>q[i].r)del(a[rig--]);
        ans1[q[i].idx]=cur;
        ans2[q[i].idx]=1LL*(q[i].r-q[i].l+1)*(q[i].r-q[i].l)/2;
    }
    _for(i,0,m){
        if(ans1[i]==0)
```

```
        ans2[i]=1;
        else{
            LL g=gcd(ans1[i],ans2[i]);
            ans1[i]/=g;
            ans2[i]/=g;
        }
        printf("%lld/%lld\n",ans1[i],ans2[i]);
    }
    return 0;
}
```

算法优化

利用奇偶化排序，即奇数块按 r_i 从小到大排序，偶数块 r_i 从大到小排序。

于是可以减少从一个块转移到另一个块时 r_i 的移动次数，具体代码如下

```
struct query{
    int l,r,idx;
    bool operator < (const query &b)const{
        if(l/blk_sz!=b.l/blk_sz)return l<b.l;
        return ((l/blk_sz)&1)?(r<b.r):(r>b.r);
    }
};
```

带修莫队

算法模型

在普通莫队的基础上加上单点修改操作。

算法实现

增加一维时间，区间变为 $[l_i,r_i,t_i]$□先根据 l_i 进行分块，每个块再根据 r_i 进行分块，最后对 t_i 进行排序即可。

修改操作先将 $[l,r]$ 区间调整为对应区间，然后调整 t□对每个修改/还原操作，直接更新数组即可，注意需要记录每个操作修改前后的值。

一个技巧为每次修改/还原后调换改操作记录的修改前后的值，可以省去对修改/还原操作的分类讨论。

另外如果修改的位置属于区间 $[l,r]$□则需要对答案进行修改。

假设 n 和 m 同阶，首先对每个块□t_i 单调，于是每个块移动时间指针的复杂度为 $O(n)$□移动时间指针的总复杂度为 $O\left(\frac {n^3}{S^2}\right)$□

同时每个左/右指针每次只能在所在块中移动，于是每个询问左/右指针的复杂度为 $O(S)$□移动左指针的总复杂度为 $O(nS)$□

于是总复杂度为 $O\left(\frac {n^3}{S^2}+nS\right)$取取块的大小为 $O\left(n^{\frac 23}\right)$则总时间复杂度为 $O\left(n^{\frac 53}\right)$

例题 1

洛谷p1903

题意

给定一个长度为 n 的序列，接下来两种操作。

操作 1 为询问区间 $[l,r]$ 的序列中的不同的值得数量。

操作 2 为单点修改。

题解

对区间 $[l,r]$维护每个值的个数和当前答案即可。

```
const int MAXN=1.5e5,MAXC=1e6+5;
int blk_sz,cc[MAXN],lc[MAXN],col[MAXC],cur_ans;
struct query{
    int l,r,t,idx;
    bool operator < (const query &b)const{
        if(l/blk_sz!=b.l/blk_sz)return l<b.l;
        if(r/blk_sz!=b.r/blk_sz)return ((l/blk_sz)&1)?(r<b.r):(r>b.r);
        return ((r/blk_sz)&1)?(t<b.t):(t>b.t);
    }
}q[MAXN];
struct opt{
    int pos,pre,now;
}p[MAXN];
int ans[MAXN],qn,pn,lef=1,rig=0;
void add(int c){
    if(!col[c])cur_ans++;
    col[c]++;
}
void del(int c){
    col[c]--;
    if(!col[c])cur_ans--;
}
void update(opt &p){
    if(lef<=p.pos&&p.pos<=rig){
        del(p.pre);
        add(p.now);
    }
```

```
    cc[p.pos]=p.now;
    swap(p.pre,p.now);
}
int main()
{
    int n=read_int(),m=read_int();
    blk_sz=pow(n,2.0/3);
    _rep(i,1,n)
    cc[i]=lc[i]=read_int();
    _for(i,0,m){
        char c=get_char();
        if(c=='Q'){
            qn++;
            q[qn].l=read_int(),q[qn].r=read_int(),q[qn].t=pn,q[qn].idx=qn;
        }
        else{
            pn++;
            int pos=read_int(),v=read_int();
            p[pn].pos=pos,p[pn].pre=lc[pos],p[pn].now=(lc[pos]=v);
        }
    }
    sort(q+1,q+qn+1);
    int tim=0;
    _rep(i,1,qn){
        while(lef>q[i].l)add(cc[--lef]);
        while(rig<q[i].r)add(cc[++rig]);
        while(lef<q[i].l)del(cc[lef++]);
        while(rig>q[i].r)del(cc[rig--]);
        while(tim<q[i].t)update(p[++tim]);
        while(tim>q[i].t)update(p[tim--]);
        ans[q[i].idx]=cur_ans;
    }
    _rep(i,1,qn)
    enter(ans[i]);
    return 0;
}
```

例题 2

CF940F

题意

给定一个长度为 n 的序列，接下来两种操作。

操作 1 给定询问区间 $[l,r]$，设 c_i 表示 i 在 $a_l\sim a_r$ 中出现的次数，s_i 表示 i 在 $c_0\sim c_{1e9}$ 中出现的次数，求 $\text{mex}(s)$。

其中 $\text{mex}(s)$ 表示没有出现在序列 s 中的最小非负整数。

操作 2 为单点修改。

题解

首先离散化一下原序列和修改操作的数值，这样数值范围变为 $O(n+q)$，同时也不会影响答案。

另外假设 $\text{mex}(s)=k+1$，则对每个 $i\in [1,k]$ 至少有一个 j 满足 $c_j=i$，于是这样询问区间长度至少为 $\sum_{i=1}^k i=\frac {k(k+1)}2$。

于是 $\text{mem}(s)\sim O(\sqrt n)$，带修莫队维护 c,s 序列，对每个询问暴力查询 s 数组，时间复杂度 $O(n^{\frac 53}+m\sqrt n)$。

例题 3

CF1476G

题意

给定一个长度为 n 的序列，接下来两种操作。

操作 1 给定询问区间 $[l,r]$ 和整数 k，设 c_i 表示 i 在 $a_l\sim a_r$ 中出现的次数，要求选择 k 个 $c_i\gt 0$，记为 $d_{1\sim k}$，最小化 $\max (d)-\min (d)$。

操作 2 为单点修改。

题解

设 s 表示将 c 从小到大排列的序列，于是答案变为 $\min (s_{i+k}-s_i)$。

有由于对于一个固定区间，c_i 最多只有 $O(\sqrt n)$ 种取值(原因见例题 2)，于是 s 最多仅有 $O(\sqrt n)$ 个值相同的段。

考虑维护 s 每个段的左端点和右端点，然后双指针遍历 s，即可 $O(\sqrt n)$ 得到答案。

接下来考虑如何维护 s，当 c_i 变为 c_i+1 时，只需要将 s 中的值等于 c_i 的段的右端点的 c_i 改为 c_i+1 即可。

当 c_i 变为 c_i-1 时，只需要将 s 中的值等于 c_i 的段的左端点的 c_i 改为 c_i-1 即可。

因为最多有 n 个非 0 的 c_i，故初始时用 n 个 0 填充 s 即可(用 0 填充可以将插入操作转化为修改操作)。

```
const int MAXN=1e5+5,MAXC=2e5+5,Inf=1e9;
int blk_sz,a[MAXN],lst_a[MAXN],cnt[MAXC],cl[MAXN],cr[MAXN],s[MAXN];
int ans[MAXN],qn,pn,lef=1,rig=0;
struct query{
```

```
    int l,r,k,t,idx;
    bool operator < (const query &b)const{
        if(l/blk_sz!=b.l/blk_sz)return l<b.l;
        if(r/blk_sz!=b.r/blk_sz)return ((l/blk_sz)&1)?(r<b.r):(r>b.r);
        return ((r/blk_sz)&1)?(t<b.t):(t>b.t);
    }
}q[MAXN];
struct opt{
    int pos,cur,next;
}p[MAXN];
void add(int v){
    int c=cnt[v];
    s[cr[c]]++;
    cl[c+1]=cr[c];
    if(!cr[c+1])cr[c+1]=cl[c+1];
    cr[c]--;
    if(cl[c]>cr[c])cl[c]=cr[c]=0;
    cnt[v]++;
}
void del(int v){
    int c=cnt[v];
    s[cl[c]]--;
    cr[c-1]=cl[c];
    if(!cl[c-1])cl[c-1]=cr[c-1];
    cl[c]++;
    if(cl[c]>cr[c])cl[c]=cr[c]=0;
    cnt[v]--;
}
void update(opt &p){
    if(lef<=p.pos&&p.pos<=rig){
        del(p.cur);
        add(p.next);
    }
    a[p.pos]=p.next;
    swap(p.cur,p.next);
}
int main()
{
    int n=read_int(),m=read_int();
    blk_sz=pow(n,2.0/3)+1;
    _rep(i,1,n)
    a[i]=lst_a[i]=read_int();
    _for(i,0,m){
        int op=read_int();
        if(op==1){
            qn++;
            int l=read_int(),r=read_int(),k=read_int();
            q[qn].l=l,q[qn].r=r,q[qn].k=k,q[qn].t=pn,q[qn].idx=qn;
        }
        else{
            pn++;
```

```
            int pos=read_int(),v=read_int();
            p[pn].pos=pos,p[pn].cur=lst_a[pos],p[pn].next=(lst_a[pos]=v);
        }
    }
    sort(q+1,q+qn+1);
    cl[0]=1,cr[0]=n;
    _rep(i,1,n)cl[i]=cr[i]=0;
    int tim=0;
    _rep(i,1,qn){
        while(lef>q[i].l)add(a[--lef]);
        while(rig<q[i].r)add(a[++rig]);
        while(lef<q[i].l)del(a[lef++]);
        while(rig>q[i].r)del(a[rig--]);
        while(tim<q[i].t)update(p[++tim]);
        while(tim>q[i].t)update(p[tim--]);
        int cur=Inf,lpos=cr[0]+1,rpos=cr[0];
        while(lpos<=n){
            while(rpos<n&&rpos-lpos+1<q[i].k)rpos=cr[s[rpos+1]];
            if(rpos-lpos+1>=q[i].k)cur=min(cur,s[rpos]-s[lpos]);
            lpos=cr[s[lpos]]+1;
        }
        ans[q[i].idx]=cur==Inf?-1:cur;
    }
    _rep(i,1,qn)
    enter(ans[i]);
    return 0;
}
```