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 $O\left(n^{\frac 53}\right)$

算法例题

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,a[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],q_cnt,p_cnt,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.now);
        add(p.pre);
    }
```

```
    a[p.pos]=p.pre;
    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)
    a[i]=read_int();
    _for(i,0,m){
        char c=get_char();
        if(c=='Q'){
q[q_cnt].l=read_int(),q[q_cnt].r=read_int(),q[q_cnt].t=p_cnt,q[q_cnt].idx=q
_cnt;
            q_cnt++;
        }
        else{
            int pos=read_int(),v=read_int();++p_cnt;
            p[p_cnt].pos=pos,p[p_cnt].pre=a[pos],p[p_cnt].now=v,a[pos]=v;
        }
    }
    sort(q,q+q_cnt);
    int tim=p_cnt;
    _for(i,0,q_cnt){
        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--]);
        ans[q[i].idx]=cur_ans;
    }
    _for(i,0,q_cnt)
    enter(ans[i]);
    return 0;
}
```