

莫队

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 $kS \leq l_i \leq (k+1)S$ 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);
    }
};
```

From:
<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E8%8E%AB%E9%98%9F%E7%AE%97%E6%B3%95_1&rev=1612181834

Last update: 2021/02/01 20:17