题意：
中文题目就不解释了。
思路：
对于L,R的询问。设其中颜色为x,y,z....的袜子的个数为a,b,c。。。
那么答案即为(a*(a-1)/2+b*(b-1)/2+c*(c-1)/2....)/((R-L+1)*(R-L)/2)
化简得:(a^2+b^2+c^2+...x^2-(a+b+c+d+.....))/((R-L+1)*(R-L))
即：(a^2+b^2+c^2+...x^2-(R-L+1))/((R-L+1)*(R-L))
所以这道题目的关键是求一个区间内每种颜色数目的平方和。
但问题时怎么快速求解呢？
对于一般区间维护类问题一般想到用线段树。但是这题完全不知道线段树怎么做，百度了下。知道是莫队算法。
于是乎学习了下。写写学习的心得吧。
莫队算法是莫涛发明了。感觉这人蛮牛逼的。但是网上各种百度他的论文却找不到了。只好到别人的博客里学习学习。莫队算法是离线处理一类区间不修改查询类问题的算法。就是如果你知道了[L,R]的答案。你可以在O(1)的时间下得到[L,R-1]和[L,R+1]和[L-1,R]和[L+1,R]的答案的话。就可以使用莫队算法。
对于莫队算法我感觉就是暴力。只是预先知道了所有的询问。可以合理的组织计算每个询问的顺序以此来降低复杂度。要知道我们算完[L,R]的答案后现在要算[L',R']的答案。由于可以在O(1)的时间下得到[L,R-1]和[L,R+1]和[L-1,R]和[L+1,R]的答案.所以计算[L',R']的答案花的时间为|L-L'|+|R-R'|。如果把询问[L,R]看做平面上的点a(L,R).询问[L',R']看做点b(L',R')的话。那么时间开销就为两点的曼哈顿距离。所以对于每个询问看做一个点。我们要按一定顺序计算每个值。那开销就为曼哈顿距离的和。要计算到每个点。那么路径至少是一棵树。所以问题就变成了求二维平面的最小曼哈顿距离生成树。
关于二维平面最小曼哈顿距离生成树。感兴趣的可以参考点击打开链接
这样只要顺着树边计算一次就ok了。可以证明时间复杂度为n*sqrt(n)这个我不会证明。
但是这种方法编程复杂度稍微高了一点。所以有一个比较优雅的替代品。那就是先对序列分块。然后对于所有询问按照L所在块的大小排序。如果一样再按照R排序。然后按照排序后的顺序计算。为什么这样计算就可以降低复杂度呢。
一、i与i+1在同一块内，r单调递增，所以r是O(n)的。由于有n^0.5块,所以这一部分时间复杂度是n^1.5。
二、i与i+1跨越一块，r最多变化n，由于有n^0.5块，所以这一部分时间复杂度是n^1.5
三、i与i+1在同一块内时变化不超过n^0.5，跨越一块也不会超过2*n^0.5，不妨看作是n^0.5。由于有n个数，所以时间复杂度是n^1.5
于是就变成了O(n^1.5)了。

详细过程见代码：

[cpp]

```cpp
#include<algorithm>
#include<iostream>
#include<string.h>
#include<stdio.h>
#include<math.h>
using namespace std;
const int INF=0x3f3f3f3f;
const int maxn=50010;
typedef long long ll;
ll num[maxn],up[maxn],dw[maxn],ans,aa,bb,cc;
int col[maxn],pos[maxn];
struct qnode
{
    int l,r,id;
} qu[maxn];
bool cmp(qnode a,qnode b)
{
    if(pos[a.l]==pos[b.l])
        return a.r<b.r;
    return pos[a.l]<pos[b.l];
}
ll gcd(ll x,ll y)
{
    ll tp;
    while(tp=x%y)
    {
        x=y;
        y=tp;
    }
    return y;
}
void update(int x,int d)
{
    ans-=num[col[x]]*num[col[x]];
    num[col[x]]+=d;
    ans+=num[col[x]]*num[col[x]];
}
int main()
{
    int n,m,i,j,bk,pl,pr,id;

    freopen("in.txt","r",stdin);
    while(~scanf("%d%d",&n,&m))
    {
        memset(num,0,sizeof num);
        bk=ceil(sqrt(1.0*n));
        for(i=1;i<=n;i++)
        {
            scanf("%d",&col[i]);
            pos[i]=(i-1)/bk;
```

```
        }
        for(i=0;i<m;i++)
        {
            scanf("%d%d",&qu[i].l,&qu[i].r);
            qu[i].id=i;
        }
        sort(qu,qu+m,cmp);
        pl=1,pr=0;
        ans=0;
        for(i=0;i<m;i++)
        {
            id=qu[i].id;
            if(qu[i].l==qu[i].r)
            {
                up[id]=0,dw[id]=1;
                continue;
            }
            if(pr<qu[i].r)
            {
                for(j=pr+1;j<=qu[i].r;j++)
                    update(j,1);
            }
            else
            {
                for(j=pr;j>qu[i].r;j--)
                    update(j,-1);
            }
            pr=qu[i].r;
            if(pl<qu[i].l)
            {
                for(j=pl;j<qu[i].l;j++)
                    update(j,-1);
            }
            else
            {
                for(j=pl-1;j>=qu[i].l;j--)
                    update(j,1);
            }
            pl=qu[i].l;
            aa=ans-qu[i].r+qu[i].l-1;
            bb=(ll)(qu[i].r-qu[i].l+1)*(qu[i].r-qu[i].l);
            cc=gcd(aa,bb);
            aa/=cc,bb/=cc;
            up[id]=aa,dw[id]=bb;
        }
        for(i=0;i<m;i++)
            printf("%I64d/%I64d\n",up[i],dw[i]);
    }
    return 0;
}
```