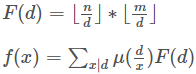
# HDU 4746 Mophues（莫比乌斯反演）

https://blog.csdn.net/u014610830/article/details/49382683

用F(d)表示满足d|gcd(x,y)且1<=x<=n,1<=y<=m的对数。可知：



设k是满足质因子个数小于p的数。枚举每一个k，就是所求的答案。



展开：



算一下现在的复杂度：

o(n)线筛出μ(x)，枚举每一个k，因为对于每一个k对应的满足条件的

d有n/k个。

一个事实：复杂度为o(logn)

所以每一个询问复杂度是o(n\*logn)，总的复杂度就为o(q\*n\*logn+n)，可以看出来跪了。

每次询问能接受的复杂度应该是o(logn)或者是级别的。

再来看表达式：



可以推出：



预处理出，

这里还有一个问题：k的质因子数要保证小于p个。

1...5\*105质因子数最多有log(5∗105)个，即不超过19个，这样原来的sum(d)再加上一维：sum(p,d)即可。

此时，在每次查找中的复杂度变成了o(1)，现在的问题变成了如何在合适的复杂度内找出的值？



这里用到分块加速的方法：

因为是向下取整，对于一个i我们一定能找到一个区间[i,i′]使得所有在该区间内取值都相同。

例：   
，   
当的取值都是p。   


枚举i只需要找到右端点即可，所有我们只需要j的右端点。即：



原式中是，每次右端点都取即可

这样在求和中不需要o(n)的遍历，看别人的博客说复杂度是，虽然我不会证。

此时的复杂度降到了。每次不再是一个完整区间的和，再维护一个前缀后即可在o(1)复杂度内搞定。sum(p,d)的值需要o(nogn)复杂度搞定。

所以总的复杂度就是

下面是代码部分：

#include <bits/stdc++.h>

#define LL long long

#define FOR(i,x,y) for(int i = x;i < y;++ i)

#define IFOR(i,x,y) for(int i = x;i > y;-- i)

using namespace std;

const int maxn = 500050;

int prime[maxn],mu[maxn],cnt[maxn];

int sum[20][maxn];

bool check[maxn];

void Mobius(){

memset(check,false,sizeof(check));

mu[1] = 1;

prime[0] = 0;

cnt[1] = 0;

FOR(i,2,maxn){

if(!check[i]){

prime[++prime[0]] = i;

mu[i] = -1;

cnt[i] = 1;

}

FOR(j,1,prime[0]+1){

if(i\*prime[j] >= maxn) break;

check[i\*prime[j]] = true;

cnt[i\*prime[j]] = cnt[i] + 1;

if(i % prime[j]){

mu[i\*prime[j]] = -mu[i];

}

else{

mu[i\*prime[j]] = 0;

break;

}

}

}

}

int n,m,p;

void init(){

Mobius();

memset(sum,0,sizeof(sum));

*//sum[i][j]表示所有能整除j的质因子个数为cnt[i]的(x,y)的对数*

FOR(i,1,maxn){

for(int j = i;j < maxn;j += i){

sum[cnt[i]][j] += mu[j/i];

}

}

*//sum[i][j]表示所有能整除j的质因子个数<=i的(x,y)的对数*

FOR(j,1,maxn){

FOR(i,1,20){

sum[i][j] += sum[i-1][j];

}

}

*//sum[i][j]表示sum[i][0]+sum[i][1]+...+sum[i][j]的前缀和*

FOR(j,1,maxn){

FOR(i,0,20){

sum[i][j] += sum[i][j-1];

}

}

}

void work(){

if(p >= 20) {printf("%I64d\n",(LL)n\*(LL)m);return;}

int mx = min(n,m);

LL ans = 0;

for(int i = 1;i <= mx;){//枚举i(即为d),也为块的左边界。

LL a = n/i,b = m/i;//a,b分别表示m,n中有多少个i。

LL now = min(n/a,m/b);//计算出当前块(相同因子数称为块)的右边界now

ans += a\*b\*(sum[p][now]-sum[p][i-1]);//计算当前块的值

i = now+1;//下一块的位置

}

printf("%I64d\n",ans);

}

int main(){

*//freopen("test.in","r",stdin);*

init();

int T; scanf("%d",&T);

while(T--){

scanf("%d%d%d",&n,&m,&p);

work();

}

return 0;

}