1. **置换**

1.poj2154（m种颜色染正n边形）

#include <iostream>

#include <cstdio>

using namespace std;

typedef long long ll;

const int N = 100100;

int mod;

//m种颜色染正n边形的n个顶点的方法数

//只有旋转置换

int isNotPrime[N],Prime[N],num\_Prime;

void getPrime()

{

memset(isNotPrime,0,sizeof(isNotPrime));

num\_Prime = 0;

for( int i = 2 ; i < N ; ++i ){

if( !isNotPrime[i] ) Prime[num\_Prime++] = i;

for( int j = 0 ; j < num\_Prime && i\*Prime[j] < N ; ++i ){

isNotPrime[i\*Prime[j]] = 1;

if(i%Prime[j] == 0) break;

}

}

}

//求n的欧拉函数值

int Euler(int n)

{

int ret = n;

for( int i = 0 ; Prime[i]\*Prime[i] <= n ; ++i ){

if(n%Prime[i] == 0){

ret = ret/Prime[i]\*(Prime[i]-1);

while(n%Prime[i] == 0){

n /= Prime[i];

}

}

}

if( n > 1 ) ret = ret/n\*(n-1);

return ret%mod;

}

ll fast\_mi(int x,int n)

{

ll ret = 1;

x %= mod;

while(n){

if(n&1) ret = ret\*x%mod;

x = x\*x%mod;

n >>= 1;

}

return ret;

}

//m种颜色染正n边形

ll solve(int m,int n)

{

ll re = 0;

for( int i = 1 ; i\*i <= n ; ++i ){

if(n%i == 0){

re = (re+Euler(n/i)\*fast\_mi(m,i-1))%mod;

if(i\*i == n) continue;

re = (re+Euler(i)\*fast\_mi(m,n/i-1))%mod;

}

}

return re;

}

int main()

{

getPrime();

int T;

scanf("%d",&T);

while( T-- ){

int n;

scanf("%d%d",&n,&mod);

printf("%lld\n",solve(n,n));

}

return 0;

}

**2.hdu1812（用c种颜色对n\*n的正方形小格子染色）**

import java.math.\*;

import java.util.\*;

import java.io.\*;

public class Main {

/\* n为奇 n为偶

\* 旋转置换: 0 n\*n n\*n

\* 90 (n\*n-1)/4+1 (n\*n)/4

\* 180 (n\*n-1)/2+1 (n\*n)/2

\* 270 (n\*n-1)/4+1 (n\*n)/4

\*

\* 反射置换: 对边中点反射(2)

\* (n\*n-n)/2+n (n\*n)/2

\* 对角反射(2)

\* (n\*n-n)/2+n (n\*n-n)/2+n

\*/

//c为颜色种数，n为正方形的边长

static BigInteger isodd(int c,int n){

BigInteger sum = BigInteger.ZERO;

BigInteger p = BigInteger.valueOf(c);

BigInteger two = BigInteger.valueOf(2);

BigInteger re;

//旋转置换的轮换数

re = p.pow(n\*n);

sum = sum.add(re);

re = two.multiply(p.pow((n\*n-1)/4+1));

sum = sum.add(re);

re = p.pow((n\*n-1)/2+1);

sum = sum.add(re);

//反射置换的轮换数

re = two.multiply(p.pow((n\*n-n)/2+n));

sum = sum.add(re);

re = two.multiply(p.pow((n\*n-n)/2+n));

sum = sum.add(re);

return sum.divide(BigInteger.valueOf(8));

}

static BigInteger iseven(int c,int n){

BigInteger sum = BigInteger.ZERO;

BigInteger p = BigInteger.valueOf(c);

BigInteger two = BigInteger.valueOf(2);

BigInteger re;

//旋转置换的轮换数

re = p.pow(n\*n);

sum = sum.add(re);

re = two.multiply(p.pow((n\*n)/4));

sum = sum.add(re);

re = p.pow((n\*n)/2);

sum = sum.add(re);

//反射置换的轮换数

re = two.multiply(p.pow((n\*n)/2));

sum = sum.add(re);

re = two.multiply(p.pow((n\*n-n)/2+n));

sum = sum.add(re);

return sum.divide(BigInteger.valueOf(8));

}

public static void main(String[]args){

Scanner cin = new Scanner(System.in);

int n,c;

while(cin.hasNext()){

n = cin.nextInt();

c = cin.nextInt();

if(n%2 == 0) System.out.println(iseven(c, n));

else System.out.println(isodd(c, n));

}

}

}

**3.高斯消元模板**

#include <algorithm>

#include <cstring>

#include <string>

#include <iostream>

#include <cmath>

using namespace std;

const int N = 100;

typedef double Matrix[N][N];

void gauss\_elimination(Matrix A,int n)

{

int i,j,k,r;

//消元过程

for( i = 0 ; i < n ; ++i ){

//选一行r并与第i行交换

r = i;

for( j = i+1 ; j < n ; ++j )

if(fabs(A[j][i]) > fabs(A[r][i])) r = j;

if(r != i) for( j = 0 ; j <= n ; ++j ) swap(A[r][j],A[i][j]);

//与第i+1~n行进行消元

for( k = i+1 ; k < n ; ++k ){

double f = A[k][i]/A[i][i];

for( j = i ; j <= n ; ++j ) A[k][j] -= f\*A[i][j];

}

// for( j = n ; j >= i ; --j )

// for( k = i+1 ; k < n ; ++k )

// A[k][j] -= A[k][i]/A[i][i]\*A[i][j];

}

//回带过程

for( i = n-1 ; i >= 0 ; --i ){

for( j = i+1 ; j < n ; ++j )

A[i][n] -= A[j][n]\*A[i][j];

A[i][n] /= A[i][i];

}

}

**4.容斥原理，一个数与一个区间内互质的数的个数**

#include <iostream>

#include <cstring>

#include <cstdio>

#include <vector>

using namespace std;

typedef long long ll;

const int MAXN = 515;

const int INF = 0x3f3f3f3f;

/\*

问一个区间内与数t互质的数的个数

我们可以求它的反面，与t不互质的数的个数

先求出t的质因子，即区间内有多少个数能被t的质因子整除

简单容斥

\*/

//数n与区间[1,r]互质的数的个数

ll solve (ll n, ll r){

vector<int>p;

for(int i=2; i\*i<=n; ++i)

if(n%i == 0){

p.push\_back (i);

while(n%i == 0)

n /= i;

}

if(n > 1)

p.push\_back(n);

int sz = p.size();

ll sum = 0;

for(int i=1; i<(1<<sz); ++i){

ll mult = 1,bits = 0;

for (int j=0; j<sz; ++j)

if (i&(1<<j)) {

++bits;

mult \*= (ll)p[j];

}

ll cur = r/mult;

if (bits % 2 == 1) sum += cur;

else sum -= cur;

}

return r - sum;

}

int main()

{

int T,cas=0;

scanf("%d",&T);

while(T--){

ll a,b,n;

scanf("%lld%lld%lld",&a,&b,&n);

printf("Case #%d: %lld\n",++cas,solve(n,b)-solve(n,a-1));

}

return 0;

}

**5.容斥原理，求两个区间内互质的数的对数**

#include <iostream>

#include <cstring>

#include <cstdio>

#include <vector>

using namespace std;

typedef long long ll;

const int N = 100300;

const int INF = 0x3f3f3f3f;

int oula[N];

void getLa( int n )

{

for( int i = 0 ; i < n ; ++i ) oula[i] = i;

for( int i = 2 ; i < n ; ++i ){

//i为质数

if( oula[i] == i ){

for( int j = 1 ; j\*i < n ; ++j ){

oula[j\*i] -= oula[j\*i]/i;

}

}

}

}

int solve(int n,int r){

vector<int>p;

for(int i=2; i\*i<=n; ++i)

if(n%i == 0){

p.push\_back (i);

while(n%i == 0)

n /= i;

}

if(n > 1)

p.push\_back(n);

int sz = p.size();

int sum = 0;

for(int i=1; i<(1<<sz); ++i){

int mult = 1,bits = 0;

for (int j=0; j<sz; ++j)

if (i&(1<<j)) {

++bits;

mult \*= (ll)p[j];

}

int cur = r/mult;

if (bits % 2 == 1) sum += cur;

else sum -= cur;

}

return r - sum;

}

int main()

{

getLa(N);

int T,cas = 0;

scanf("%d",&T);

while( T-- ){

int a,b,c,d,k;

scanf("%d%d%d%d%d",&a,&b,&c,&d,&k);

if(k == 0) {printf("Case %d: %d\n",++cas,0);continue;}

int t1 = b/k;

int t2 = d/k;

if( t1 > t2 ) swap(t1,t2);

ll sum = 0;

for( int i = 1 ; i <= t1 ; ++i ) sum += (ll)oula[i];

for( int i = t1+1 ; i <= t2 ; ++i ) sum += (ll)solve(i,t1);

printf("Case %d: %lld\n",++cas,sum);

}

return 0;

}

**6.容斥原理，求两个区间内某两个数的乘积是某个数的倍数的对数**

首先把a分解质因数，然后通过质因数dfs找到a的所有因数。

假设a分解质因数后是a=k(1)^p(1)+k(2)^p(2)+...+k(n)^p(n)。

对于a的每一个因数x，假设x分解质因数后是x=k(1)^s(1)+k(2)^s(2)+...+k(n)^s(n)，找到1-b中所有x的倍数共有b1个，

并且列举出所有s(i)<p(i)(1≤i≤n)对应的质因数k(t(1)),k(t(2)),...k(t(m))。

把找出的m个质因数用容斥原理去掉b1个数字中x\*k(t(j))的倍数(1≤j≤m)剩下b2个数字，

计算1-c中a/x的倍数的个数c1，统计b2\*c1的和就是结果。

另外，因为数字范围较大，要用类似快速幂的方法计算乘法。

#include <iostream>

#include <cstdio>

using namespace std;

typedef long long ll;

typedef pair<ll,ll>PA;

const ll N = 100900;

const ll M = 60;

const ll mod = 333333333333333331ll;

vector<ll>factor;

vector<ll>prime\_factor;

ll gcd(ll a,ll b)

{

return b==0?a:gcd(b,a%b);

}

ll lcm(ll a,ll b)

{

return a/gcd(a,b)\*b;

}

void getDivisor( ll n )

{

factor.clear();

for( ll i = 1 ; i\*i <= n ; ++i ){

if( n%i == 0 ){

factor.push\_back(i);

if( n/i != i )

factor.push\_back(n/i);

}

}

prime\_factor.clear();

for( ll i = 2 ; i\*i <= n ; ++i ){

if( n%i == 0 ){

prime\_factor.push\_back(i);

while( n%i == 0 ){

n /= i;

}

}

}

if( n > 1 ) prime\_factor.push\_back(n);

}

ll fast\_mult(ll x,ll n)

{

ll ret = 0;

x = x%mod;

while(n){

if(n&1) ret = (ret+x)%mod;

x = (x+x)%mod;

n >>= 1;

}

return ret;

}

ll solve(ll b,int x,ll a)

{

ll s = a/x;

vector<ll>p;

for( ll i = 0 ; i < prime\_factor.size() ; ++i ){

if( s%prime\_factor[i] == 0 ) p.push\_back(prime\_factor[i]\*x);

}

ll sz = p.size();

ll sum = b/x;

for( ll i = 1 ; i < (1<<sz) ; ++i ){

ll mult = 1,bits = 0;

for( ll j = 0 ; j < sz ; ++j ){

if( i&(1<<j) ){

++bits;

//求最小公倍数

mult = lcm(mult,p[j])%mod;

}

}

ll cur = b/mult;

if(bits%2 == 1) sum = (sum-cur)%mod;

else sum = (sum+cur)%mod;

}

return sum;

}

int main()

{

// freopen("in.txt","r",stdin);

// freopen("yuan.txt","w",stdout);

ll a,b,c;

while( ~scanf("%lld%lld%lld",&a,&b,&c) ){

getDivisor((ll)a);

ll sum = 0;

for( ll i = 0 ; i < factor.size() ; ++i ){

ll t = solve(b,factor[i],a);

ll s = c/(a/factor[i]);

sum = (sum+fast\_mult(t,s))%mod;

}

printf("%lld\n",sum);

}

return 0;

}