CRT

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
ll exgcd(ll a,ll b,ll &x,ll &y){//扩展欧几里得算法
	if(b==0){//递归边界
		x=1;y=0;
		return a;
	}
	ll ret=exgcd(b,a%b,x,y);
	ll tmp=y;//求解原 x,y
	y=x-a/b*y;
	x=tmp;
	return ret;//返回 gcd
}
ll crt(ll a[],ll b[],int n)//a[]为除数，b[]为余数
{
	ll M=1,y,x=0;
	for(int i=0;i<n;++i)    //算出它们累乘的结果
		M*=a[i];
	for(int i=0;i<n;++i)
	{
		ll w=M/a[i];
		ll tw=0;
		exgcd(w,a[i],tw,y);    //计算逆元
		ll t=w*b[i]%M*tw%M;
		x=(x+t)%M;
	}
	return (x+M)%M;
}
int n;
ll a[20],b[20];
int main(){
	cin>>n;
	for(int i=0;i<n;i++){
		cin>>a[i]>>b[i];
		//cout<<a[i]<<' '<<b[i]<<endl;
	}
	cout<<crt(a,b,n)<<endl;
	return 0;
}
```

excrt

```
// Code by KSkun, 2018/4
#include <cstdio>

typedef long long LL;

const int MAXN = 100005;

LL k, a[MAXN], r[MAXN];

inline LL exgcd(LL a, LL b, LL &x, LL &y) {
	if(!b) {
		x = 1; y = 0; return a;
	}
	LL res = exgcd(b, a % b, x, y);
	LL t = x; x = y; y = t - a / b * y;
	return res;
}

inline LL excrt() {
	LL A = a[1], R = r[1], x, y;
	for(int i = 2; i <= k; i++) {
		LL g = exgcd(A, a[i], x, y);
		if((r[i] - R) % g) return -1;
		x = (r[i] - R) / g * x; x = (x % (a[i] / g) + a[i] / g) % (a[i] / g);
		R = A * x + R;
		A = A / g * a[i]; R %= A;
	}
	return R;
}

int main() {
	while(scanf("%lld", &k) != EOF) {
		for(int i = 1; i <= k; i++) {
			scanf("%lld%lld", &a[i], &r[i]);
		}
		printf("%lld\n", excrt());
	}
	return 0;
}
```

exlucas

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define db double
ll ct;
ll A[1020], B[1020];
ll POW(ll a, ll b, ll p) {
	ll cur = a, ans = 1;
	while (b) {
		if (b & 1) ans = ans * cur % p;
		cur = cur * cur % p;
		b >>= 1;
	}
	return ans % p;
}
void exGCD(ll a, ll b, ll &x, ll &y) {
	if (!b) return (void)(x = 1, y = 0);
	exGCD(b, a % b, x, y);
	ll tmp = x;
	x = y;
	y = tmp - a / b * y;
}
inline ll INV(ll a, ll p) {
	ll x, y;
	exGCD(a, p, x, y);
	return (x % p + p) % p;
}
// n 个方程：x=a[i](mod m[i]) (0<=i<n)
ll CRT(int n, ll *a, ll *m) {
	ll M = 1, ret = 0;
	for (int i = 1; i <= n; i++) M *= m[i];
	for (int i = 1; i <= n; i++) {
		ll w = M / m[i];
		ret = (ret + w * INV(w, m[i]) * a[i]) % M;
	}
	return (ret + M) % M;
}

inline ll G(ll n, ll P) {
	if (n < P) return 0;
	return G(n / P, P) + (n / P);
```

```
}
inline ll F(ll n, ll p, ll pk) {
	if (n == 0) return 1;
	ll rou = 1;	//循环节
	ll rem = 1;	//余项
	for (ll i = 1; i <= pk; i++) {
		if (i % p) rou = rou * i % pk;
	}
	rou = POW(rou, n / pk, pk);
	for (ll i = pk * (n / pk); i <= n; i++) {
		if (i % p) rem = rem * (i % pk) % pk;
	}
	return F(n / p, p, pk) * rou % pk * rem % pk;
}
ll exC(ll n, ll m, ll p,ll pk) {
	ll fz=F(n,p,pk),fm1=INV(F(m,p,pk),pk),fm2=INV(F(n-m,p,pk),pk);
	ll mi=POW(p,G(n,p)-G(m,p)-G(n-m,p),pk);
	return fz*fm1%pk*fm2%pk*mi%pk;
}
ll exLucas(ll n, ll m, ll p) {
	ll tmp = p, ct = 0;
	for (ll i = 2; i * i <= tmp; i++) {
		if (tmp % i == 0) {
			ll s = 1;
			while (tmp % i == 0) {
				s *= i;tmp /= i;
			}
			B[++ct]=s;A[ct]=exC(n,m,i,s);
		}
	}
	if (tmp > 1){
		B[++ct]=tmp;
		A[ct]=exC(n,m,tmp,tmp);
	}
	return CRT(ct,A,B);
}

ll n, m, p;
int main() {
	scanf("%lld%lld%lld", &n, &m, &p);
	printf("%lld\n", exLucas(n, m, p));
	return 0;
}
```

math

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<cmath>
//#define io_opt ios::sync_with_stdio(false);cin.tie(0);cout.tie(0)
#ifdef io_opt
#define scanf sb
#define printf sb
#endif
typedef long long ll;
#define lll __int128
typedef double db;
using namespace std;
#define eps 0.000000001
int mod=1e9+7;
const int PN = 100020;
int pn;
bool ipr[PN + 10];
int pri[PN / 5];
int phi[PN + 10];
void prime() {
	memset(ipr, true, sizeof(ipr));
	ipr[1] = false;
	int N = sqrt(PN) + 0.5;
	for (int i = 2; i <= N; i++) {
		if (ipr[i]) {
			int d = i == 2 ? i : 2 * i;
			for (int j = i * i; j <= PN; j += d) {
				ipr[j] = false;
			}
		}
	}
	for (int i = 1; i <= PN; i++) {
		if (ipr[i]) pri[++pn] = i;
	}
}
int ct[PN/5];
void fj(int t){
	for(int i=1;i<=pn&&pri[i]*pri[i]<=t;i++){
		int cnt=0;
		while(t%pri[i]==0){
			t/=pri[i];
```

```
			cnt++;
		}
		ct[pri[i]]=max(ct[pri[i]],cnt);
	}
	if(t>1) ct[t]=max(ct[t],1);
}
void eprime() { //not verify
	memset(ipr, true, sizeof(ipr));
	ipr[1] = false;
	phi[1] = 1;
	for (int i = 2; i <= PN; i++) {
		if (ipr[i]) pri[++pn] = i, phi[i] = i - 1;
		for (int j = 1; j <= pn && pri[j] * i <= PN; j++) {
			ipr[pri[j] * i] = false;
			if (i % pri[j]) phi[i * pri[j]] = phi[i] * phi[pri[j]];
			else {
				phi[i * pri[j]] = phi[i] * pri[j];
				break;
			}
		}
	}
}

int euler_phi(int n) {
	int ans = n;
	for (int i = 2; i * i <= n; i++)
		if (n % i == 0) {
			ans = ans / i * (i - 1);
			while (n % i == 0) n /= i;
		}
	if (n > 1) ans = ans / n * (n - 1);
	return ans;
}

ll speed(ll a, ll b, ll p) {
	ll cur = a, ans = 1;
	while (b) {
		if (b & 1) ans = ans * cur % p;
		cur = cur * cur % p;
		b >>= 1;
	}
	return ans % p;
}
inline ll mm(ll k,ll p){
```

```
	return k<p?k:k%p;
	return k>=p?k%p:(k>=0?k:k%p+p);
}
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;
}
ll exgcd(ll &x,ll &y,ll a,ll b){
	if(!b)
	{
		x=1;
		y=0;
		return a;
	}
	ll gd=exgcd(x,y,b,a%b);
	ll t=x;
	x=y;
	y=t-a/b*y;
	return gd;
}
ll inv(ll a,ll b){
	ll x,y;
	exgcd(x,y,a,b);
	return mm(x,b);
}
const int MatrixSize=20;
struct Mat{
	int m,n;
	ll a[MatrixSize][MatrixSize];
	Mat(int mm=0,int nn=0,int init=-1):m(mm),n(nn){
		if(mm==0||nn==0){
			m=n=2;
			a[0][0]=1;
			a[0][1]=1;
			a[1][0]=1;
			a[1][1]=0;
			return;
		}
		if(init==-1) return;
		else if(init==0){
			for(int i=0;i<m;i++){
				for(int j=0;j<n;j++){
```

```
                    a[i][j]=0;
                }
            }
        }
        else if(m==n){
            for(int i=0;i<m;i++){
                for(int j=0;j<n;j++){
                    a[i][j]=i==j?1:0;
                }
            }
        }
    }

};
Mat operator+(Mat x,Mat y){
    for(int i=0;i<x.m;i++){
        for(int j=0;j<x.n;j++){
            x.a[i][j]=mm(x.a[i][j]+y.a[i][j],mod);
        }
    }
    return x;
}
Mat operator*(Mat x,Mat y){
    Mat ret(x.m,y.n);
    ll t;
    for(int i=0;i<x.m;i++){
        for(int j=0;j<y.n;j++){
            t=0;
            for(int k=0;k<x.n;k++){
                t=mm(t+x.a[i][k]*y.a[k][j],mod);
            }
            ret.a[i][j]=t;
        }
    }
    return ret;
}
Mat speed(Mat cur,ll b,ll p){
    Mat ans(cur.m,cur.n,1);
    while (b) {
        if (b & 1) ans = ans*cur;
        cur = cur*cur;
        b >>= 1;
    }
    return ans;
```

```
}
ll getFib(ll x,ll p){
	Mat t;
	return speed(t,x,p).a[0][1];
}
int T,n;
const int CTN=1000000;
ll f[2*CTN+10]={1};
ll ivs[CTN+10]={1};
ll iv[CTN+10]={1};
ll h[CTN+10]={1};
void CTL(){
	for(int i=1;i<=CTN+1;i++){
		f[i]=mm(f[i-1]*i,mod);
		iv[i]=speed(i,mod-2,mod);
		ivs[i]=mm(ivs[i-1]*iv[i],mod);
	}
	for(int i=CTN+2;i<=CTN*2;i++){
		f[i]=mm(f[i-1]*i,mod);
	}
	for(int i=1;i<=CTN;i++){
		h[i]=mm(f[2*i]*mm(ivs[i]*mm(ivs[i]*iv[i+1],mod),mod),mod);
	}
}
int main(){
	mod=1000000007;
	CTL();
	scanf("%d",&T);
	for(int I=1;I<=T;I++){
		scanf("%d",&n);
		ll ans=0;

		for(int i=0;i<=n/2;i++){
			ans=mm(ans+mm(h[i]*mm(f[n]*mm(ivs[2*i]*ivs[n-2*i],mod),mod),mod),mod);
		}
		printf("%lld\n",ans);
	}
	return 0;
}
```

逆元

```
void exgcd(ll a,ll b,ll &x,ll &y){
	if (!b) return (void)(x=1,y=0);
	exgcd(b,a%b,x,y);
	ll tmp=x;x=y;y=tmp-a/b*y;
}
inline ll INV(ll a,ll p){
	ll x,y;
	exgcd(a,p,x,y);
	return (x%p+p)%p;
}
```

非素数模数的计算&&Lucas 定理

非素数模数的普遍计算

常规方法

以组合数为例。

对模数 m 进行因数分解，每个出现的因数存下来，这样最多十几个因数。

然后每次乘一个数之前，就检验是否包含得到的因数，包含就计数并且除掉，不计算到逆元和阶乘的前缀里。

设数组 C[i][j]表示阶乘乘到 i 时，对于前面得到的第 j 个质数，能分解出 C[i][j]个。

这样在计算 fac[n]×inv[n-m]×inv[m]之后，只需要再计算出上下因子个数的差，计算出没有乘上的因子，乘上去即可。

中国剩余定理解法

感觉有点 fake。

分解模数为互质的数，分别计算，再用中国剩余定理求解，不过有相应因子的还是要先除掉，最后再乘。

Lucas 定理(p 是素数)

Lucas(n,m)=C(n,m)%p

Lucas(n,m)=C(n%p,m%p)*Lucas(n/p,m/p)%p

```
ll Lucas(ll n,ll m,ll p){
    if(m==0)return 1;
    return C(n%p,m%p)*Lucas(n/p,m/p)%p;
```

```
}
```

优读

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int n,a[1010];
struct ios{
	inline char gc(){
		static const int IN_LEN=1<<18|1;
		static char buf[IN_LEN],*s,*t;
		return (s==t)&&(t=(s=buf)+fread(buf,1,IN_LEN,stdin)),s==t?-1:*s++;
	}
	template <typename _Tp> inline ios & operator >> (_Tp&x){
		static char ch,sgn; ch = gc(), sgn = 0;
		for(;!isdigit(ch);ch=gc()){if(ch==-1)return *this;sgn|=ch=='-';}
		for(x=0;isdigit(ch);ch=gc())x=x*10+(ch^'0');
		sgn&&(x=-x); return *this;
	}
}io;
int inline read(){
	int num=0;
	char c;
	bool plus=true;
	while((c=getchar())==' '||c=='\n'||c=='\r');
	if(c=='-') plus=false;
	else num=c-'0';
	while(isdigit(c=getchar()))
		num=num*10+c-'0';
	return    num*(plus?1:-1);
}
inline char nc(){
	static char buf[100000],*p1=buf,*p2=buf;
	return p1==p2&&(p2=(p1=buf)+fread(buf,1,100000,stdin),p1==p2)?EOF:*p1++;
}
inline int _read(){
	char ch=nc();int sum=0;
	while(!(ch>='0'&&ch<='9'))ch=nc();
	while(ch>='0'&&ch<='9')sum=sum*10+ch-48,ch=nc();
	return sum;
}
int main()
{
```

```
    n=read();
    for(int i=1;i<=n;i++)
        a[i]=read();
    printf("%d\n",n);
    for(int i=1;i<=n;i++)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```