### sort(a, a+N); ### complex (a+N); ### complex (abuble PI = acos(-1.0); ### complex (double PI = acos(-1.0); ### comple	// ====== FFT 傅里叶======	}
using namespace std; const double PI = acos(-1.0); struct complex {	#include <algorithm></algorithm>	• • • • • • • • • • • • • • • • • • • •
len <<-1;	#include <cmath></cmath>	int len_tmp = a[N-1]+1, len = 1;
struct complex {	using namespace std;	
double r, i); complex(double _ r = 0.0,double _ i = 0.0) {	const double PI = acos(-1.0);	,
complex(double_r=0.0,double_i=0.0) {r=_r; =_j; =_j; complex operator +(const complex &b) {r=_tr; =_l; =_l; complex operator +(const complex &b) {return complex(r-b.r,i-b.i);} complex operator -(const complex &b) {return complex(r-b.r,i-b.i);} complex operator -(const complex &b) {return complex(r-b.r,i-b.i);} complex operator *(const complex &b) {return complex(r-b.r,i-b.i); for(int i = 0; < len-j.i+) {return complex(r-b.r,i-b.i); for(int i = 0; < len	struct complex {	
(r = '_r; i = _i;)	double r,i;	
Second	complex(double _r = 0.0,double _i = 0.0)	
{return complex(r+b,r,i+b.i);} complex operator -(const complex &b) freturn complex(r-b,r-b.b);} complex operator *(const complex &b) freturn complex(r*b,r-i*b.b);} complex operator *(const complex &b) freturn complex(r*b,r-i*b.i,r*b.i+i*b.r);} } void change(complex y[], int len) { int i, j, k; for (i = 1, j = len/2; < len-1; i++)		
complex operator -(const complex &b)		
(return complex(r-b,r,i-b,i); complex operator *(const complex &b) (return complex(r*b,r-i*b,i,r*b,i+i*b,r); } (return complex(r*b,r-i*b,i,r*b,i+i*b,r); } (rotint complex y[], int len) {		
complex operator *(const complex &b)		
{return complex(r*b.r-i*b.i,r*b.i+i*b.r); } ; } ; yide change(complex y[],int len) {		
//可能要:扣除 a[i]+a[i]的情况		
void change(complex y[],int len) { int i,j,k; for(i = 1, j = len/2;i < len-1; i++) { if(i < j)swap(y[i],y[j]); k = len/2; while(j >= k) {j -= k;k /= 2;} if(j < k) j += k; } } void fft(complex y[],int len,int on) //on==-1 IDFT { change(y,len); for(int h = 2; h <= len; h <<= 1) { complex wn(cos(-on*2*Pl/h),sin(-on*2*Pl/h)); for(int j = 0; j < len; j+h/2); v[k] = u+t; v[k+h/2] = u-t; w = w*wn; } if(on == -1) for(int i = 0; i < len; i++) y[j].r /= len; } } const int MAXN = 200011; complex x(MAXN * 4); th namin() { memset(num, 0, 0, sizeof(num)); for (int i = 0; i < ln; i++) { compliation = search(c);		
int i, j, k; for(i = 1, j = len/2;i < len-1; i++) {		
for(i = 1, j = len/2;i < len-1; i++) { if(i < j)swap(y[i],y[j]); k = len/2; while(j >= k; k /= 2;) if(j < k) j += k; } } void fft(complex y[],int len,int on) //on==-1 IDFT { change(y,len); for(int h = 2; h <= len; h <<= 1) { complex wn(cos(-on*2*PI/h),sin(-on*2*PI/h)); for(int k = j,k < j+h/2;k++) { complex u = y[k]; complex t = w*y[k+h/2]; y[k+h/2] = u-t; w = w*wn; } if(on ==-1) for(int i = 0;i < len;i++) y[i].r /= len; for (int i = 0;i < N; i++) { complex x[MAXN * 4]; lt. ln might(n), sizeof(num)); for (int i = 0; i < N; i++) { complex x[MAXN * 4]; lt. ln might(n), sizeof(num)); for (int i = 0; i < N; i++) {		
fif(i < j)swap(y[i],y[j]);		//可能要:扣除带 0 的特殊情况
if(i < j)swap(y[i],y[j]); k = len/2; while(j >= k) {j -= k; k /= 2;} if(j < k) j += k; } void fft(complex y[],int len,int on) //on=-1 IDFT { change(y,len); for(int h = 2; h <= len; h <<= 1) { complex wn(cos(-on*2*Pl/h),sin(-on*2*Pl/h)); for(int j = 0; j < len; j += h) { complex w(1,0); for(int k = j;k < j +h/2;k++) { complex u = y[k];		Cnt -= (LL)Cnt0 * (N-1) * 2LL;// 0+ai=ai &&
k = len/2;		ai+0=ai
if(j < k) j += k; Catalan 数 Cat[n]=C[2*n][n]/(n+1) 组合性质 Cat[n]=sum(Cat[i]*Cat[n-i] for i from 0 to n) (/on=-1 IDFT {		printf("%lld\n", Cnt); }
if(j < k) j += k;	while($j \ge k$) { $j = k; k \ne 2;$ }	// ======== Catalan =======
Cat[n]=C[2*n][n]/(n+1) 组合性质 (Cat[n]=c[1*n][n]/(n+1) 组合性质 Cat[n]=c[1*n][n]/(n+1) 组合性质 Cat[n]=cat[n*1]=sum(Cat[i]*Cat[n-i] for i from 0 to n) Cat[0]=1, Cat[n+1]=sum(Cat[i]*Cat[n-i] for in for i for in for no not not no not not not not not not n	if(j < k) j += k;	
yoid fft(complex y[],int len,int on) //on==-1 IDFT { change(y,len); for(int h = 2; h <= len; h <<= 1) { complex wn(cos(-on*2*PI/h),sin(-on*2*PI/h)); for(int h = j, k < j+h/2;k++) { complex w (1,0); for(int k = j,k < j+h/2;k++) } { complex u = y[k]; complex u = w*y[k+h/2]; y[k] = u+t; y[k+h/2] = u-t; w = w*wn; } if(on == -1) for(int i = 0;i < len;i++) y[i].r /= len; } int MAXN = 200011; complex x {MAXN * 4}; int main() { memset(num, 0, sizeof(num)); for (int i = 0;i < N; i++) { scanf("%d", &a[i]); } int combination, table(int n, MOD){ memset(c), o, sizeof(C)); complex table canding and the properties of the	}	Cat[n]=C[2*n][n]/(n+1)
void fft(complex y[],int len,int on) //on==-1 IDFT { change(y,len); for(int h = 2; h <= len; h <<= 1) { complex wn(cos(-on*2*Pl/h),sin(-on*2*Pl/h)); for(int j = 0; j < len; j+=h) { complex w (1,0); for(int k = j; k < j+h/2;k++) { complex u = y[k]; complex t = w*y[k+h/2]; y[k] = u+t; y[k]+h/2] = u-t; w = w*wn; } if(on == -1) for(int i = 0;i < len;i++) y[i].r /= len; } const int MAXN = 200011; complex x[MAXN* 4]; int main() { memset(num, 0, sizeof(num)); for (int i = 0; i < N; i++) {	}	
(Annge(y,len); for(int h = 2; h <= len; h <<= 1) { complex wn(cos(-on*2*Pl/h),sin(-on*2*Pl/h)); for(int j = 0; j < len; j+=h) { complex w(1,0); for(int k = j; k < j+h/2;k++) { complex u = y[k]; complex t = w*y[k+h/2]; y[k] = u+t; y[k] = u+t; y[k] = w*w(n; } } if(on == -1) for(int i = 0; i < len;i++) y[i].r /= len; } const int MAXN = 200011; complex x[MAXN * 4]; tnt main() { memset(num, 0, sizeof(num)); for (int i = 0; i < N; i++) { memset(num, 0, sizeof(num)); for (int i = 0; i < N; i++) { memset(num, 0, sizeof(c)); conflex(y,len); din Max(y,len); don Myck 词总数(长度为 2*n 的 Dyck 词。		
change(y,len); for(int h = 2; h <= len; h <<= 1) { complex wn(cos(-on*2*Pl/h),sin(-on*2*Pl/h)); for(int j = 0; j < len; j+=h) { complex w(1,0); for(int k = j;k < j+h/2;k++) { complex t = w*y[k+h/2]; y[k] = u+t; y[k+h/2] = u-t; w = w*wn; } if(on == -1) for(int i = 0; i < len; i++) y[i].r /= len; for (int i = 0; i < len; i++) y[i].r /= len; for (int i = 0; i < len; i++) { memset(num, 0, sizeof(num)); for (int i = 0; i < N; i++) { memset(num, 0, sizeof(num)); for (int i = 0; i < N; i++) { memset(num, 0, sizeof(c)); conflex (a, li); lo Dyck 词总数(长度为 2*n 的 Dyck 词总数(长有)		
cnange(y,len); for(int h = 2; h <= len; h <<= 1) { complex wn(cos(-on*2*PI/h),sin(-on*2*PI/h)); for(int j = 0; j < len; j+=h) { complex w(1,0); for(int k = j; k < j+h/2; k++) { complex u = y[k]; complex t = w*y[k+h/2]; y[k] = u+t; y[k]+h/2] = u-t; w = w*wn; } if(on == -1) for(int i = 0; i < len; i++) y[i].r /= len; } const int MAXN = 200011; complex x[MAXN * 4]; int main() { memset(num, 0, sizeof(num)); for (int i = 0; i < N; i++) { scanf("%d", &a[i]); } b Dyck 词由 n 个'X'和 n 个'Y'组成, 对于其任意 前缀, 有 count('X')>=count('Y')) Cat[n]为给长度(n+1)的序列打上括号的不同方案 数 Cat[n]为有(n+1)片叶子的不同完全二叉树数 Cat[n]为用不相交直线将凸(n+2)边形划分为 n 个 三角形的方案数 Cat[n]为用不相交直线将凸(n+2)边形划分为 n 个 三角形的方案数 Cat[n]为有 n 个非叶子节点的不同二叉树数 Cat[n]为有 n 个非叶子节点的不同完全二叉树数 Cat[n]为有 n 个非叶子节点的不同完全三叉树数 Cat[n]为有 n 个非叶子节点的不同完全二叉树数 Cat[n]为有 n 个非叶子节点的不同完全三叉树数 Cat[n]为有 n 个非叶子节点的不同主义的表面和自由的不同工程序,由于是有的不同工程序,由于是有的不同类的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不可能的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的工程序,由于是有的工程序,由于是有的不同工程序,由于是有的不同工程序,由于是有的工程序,由于是有的工程序,由于是有的工程序,		
in (int in = 2; in <= left; in <<= 1)	= ''	
Complex wn(cos(-on*2*PI/h),sin(-on*2*PI/h)); for(int j = 0;j < len;j+=h) { complex w(1,0); for(int k = j;k < j+h/2;k++) { complex u = y[k]; complex t = w*y[k+h/2]; y[k] = u+t; y[k+h/2] = u-t; w = w*wn; } } const int MAXN = 200011; complex x[MAXN * 4]; complex x[MAXN * 4]; complex x[MAXN * 4]; complex t[m1, m1, n], complex x[m2, m2, m2]; complex x[m2, m2, m2]; complex x[m2, m2, m2]; complex x[m2, m2, m2]; for (int i = 0;i < len;i++) x[int ai[MAXN]; int main() { memset(num, 0, sizeof(num)); for (int i = 0;i < N; i++) { scanf("%d", &a[i]); } Cat[n]为有(n+1)片叶子的不同完全二叉树数 Cat[n]为用不相交直线将凸(n+2)边形划分为 n 个 三角形的方案数 Cat[n]为有 n 个非叶子节点的不同二叉树数 // ========== Combination ====================================		
for(int j = 0;j < len;j+=h) {	•	
Cat[n]为有(n+1)片叶子的不同完全二叉树数 complex w(1,0); for(int k = j;k < j+h/2;k++) { complex u = y[k]; complex t = w*y[k+h/2]; y[k] = u+t; y[k+h/2] = u-t; w = w*wn; } if(on == -1) for(int i = 0;i < len;i++) y[i].r /= len; } const int MAXN = 200011; complex x[MAXN * 4]; Lnum[MAXN * 4]; int a[MAXN]; int main() { memset(num, 0, sizeof(num)); for (int i = 0;i < N; i++) { scanf("%d", &a[i]); } Cat[n]为有(n+1)片叶子的不同完全二叉树数 Cat[n]为 n*n 网格线从左下角到右上角不经过左上部分的最短路径数 Cat[n]为用不相交直线将凸(n+2)边形划分为 n 个 三角形的方案数 Cat[n]为有 n 个非叶子节点的不同二叉树数 // ========== Combination ====================================		
Cat[n]为 n*n 网格线从左下角到右上角不经过左 L部分的最短路径数 Cat[n]为用不相交直线将凸(n+2)边形划分为 n 个 三角形的方案数 Cat[n]为有 n 个非叶子节点的不同二叉树数 Cat[n]为有 n 个非叶子可以和同二叉树数 Cat[n]为有 n 个非叶子可以和同二型的不同二型的不同二型的不同一型的不同一型的不同一型的不同一型的不同一型的不同一型的不同一型的不同一	for(int j = 0;j < len;j+=h)	
for(int k = j;k < j+h/2;k++) { complex u = y[k]; complex t = w*y[k+h/2]; y[k] = u+t; y[k+h/2] = u-t; w = w*wn; } if(on == -1) for(int i = 0;i < len;i++) y[i].r /= len; } const int MAXN = 200011; complex x[MAXN * 4]; LL num[MAXN * 4]; int a[MAXN]; int main() { memset(num, 0, sizeof(num)); for (int i = 0;i < N; i++) { scanf("%d", &a[i]);	{	
{		
complex u = y[k]; complex t = w*y[k+h/2]; y[k] = u+t; y[k+h/2] = u-t; w = w*wn; } If fast_pow(x, k, p); int Combination(int m, int n, int p){	for(int k = j;k < j+h/2;k++)	
complex t = w*y[k+h/2]; y[k] = u+t; y[k+h/2] = u-t; w = w*wn; } If fast_pow(ll x, ll k, ll p); int Combination (int m, int n, int p){ ll nom=1, den=1; for(int i = 0;i < len;i++) y[i].r /= len; } const int MAXN = 200011; complex x[MAXN * 4]; LL num[MAXN * 4]; int a[MAXN]; int main() { memset(num, 0, sizeof(num)); for (int i = 0;i < N; i++) { scanf("%d", &a[i]); cat[n]为有 n 个非叶子节点的不同二叉树数 // ========== Combination ====================================	{	Cat[n]为用不相交直线将凸(n+2)边形划分为 n 个
y[k] = u+t; y[k+h/2] = u-t; w = w*wn; } if(on == -1) for(int i = 0;i < len;i++) y[i].r /= len; } const int MAXN = 200011; complex x[MAXN * 4]; LL num[MAXN * 4]; int a[MAXN]; int main() { memset(num, 0, sizeof(num)); for (int i = 0;i < N; i++) { scanf("%d", &a[i]);		三角形的方案数
<pre>y[k+h/2] = u-t; w = w*wn; } If fast_pow(x, k, p); int Combination(int m, int n, int p){ ll nom=1, den=1; for(int i = 0;i < len;i++) y[i].r /= len; } const int MAXN = 200011; complex x[MAXN * 4]; LL num[MAXN * 4]; int a[MAXN]; int main() { memset(num, 0, sizeof(num)); for (int i = 0;i < N; i++) { scanf("%d", &a[i]); combination ====================================</pre>		Cat[n]为有 n 个非叶子节点的不同二叉树数
<pre> W = W*Wn; </pre>	• • • • • • • • • • • • • • • • • • • •	
Il fast_pow(x, k, p);		// ====== Combination =======
int Combination(int m, int n, int p){ } int Combination(int m, int n, int p){ Il nom=1, den=1; for(int i=0;i < len;i++) y[i].r /= len; } const int MAXN = 200011; complex x[MAXN * 4]; LL num[MAXN * 4]; int a[MAXN]; int main() { memset(num, 0, sizeof(num)); for (int i = 0;i < N; i++) { scanf("%d", &a[i]); int Combination(int m, int n, int p){ Il nom=1, den=1; for(int i=m-n+1; i<=m; i++) {nom*=i; nom%=p; } for(int i=2; i<=n; i++) {den*=i; den%=p; } den=fast_pow(den, p-2, p); return (nom*den)%p; int Combination_table(int n, Il MOD){ memset(C, 0, sizeof(C)); constant int Combination_table(int n, Il MOD){ memset(C, 0, sizeof(C));	•	
Il nom=1, den=1; if(on == -1)		
<pre>for(int i = -1) for(int i = 0; i < len; i++) y[i].r /= len; } const int MAXN = 200011; complex x[MAXN * 4]; LL num[MAXN * 4]; int a[MAXN]; int main() { memset(num, 0, sizeof(num)); for (int i = 0; i < N; i++) { scanf("%d", &a[i]);</pre>	1	
for(int i = 0;i < len;i++) y[i].r /= len; for(int i = 0;i < len;i++) y[i].r /= len; for(int i = 0;i < len;i++) y[i].r /= len; for(int i = 2; i <= n; i++) {den*=i; den%=p; } den=fast_pow(den, p-2, p); return (nom*den)%p; Il C[maxn][maxn]; for (int i = 0;i < N; i++) { scanf("%d", &a[i]);	; if(on 1)	
<pre>for(int i=2; i<=n; i++)</pre>	· · ·	{nom*=i; nom%=p; }
<pre>const int MAXN = 200011;</pre>		for(int i=2; i<=n; i++)
<pre>complex x[MAXN * 4]; LL num[MAXN * 4]; int a[MAXN]; int main() { memset(num, 0, sizeof(num)); for (int i = 0;i < N; i++) { scanf("%d", &a[i]);</pre>		{den*=i; den%=p; }
LL num[MAXN * 4]; int a[MAXN]; int main() { memset(num, 0, sizeof(num)); for (int i = 0; i < N; i++) { scanf("%d", &a[i]); return (nom*den)%p; } Il C[maxn][maxn]; int Combination_table(int n, Il MOD){ memset(C, 0, sizeof(C)); C[o][o]=1;		den=fast_pow(den, p-2, p);
<pre>int main() { memset(num, 0, sizeof(num)); for (int i = 0;i < N; i++) { scanf("%d", &a[i]);</pre>	•	return (nom*den)%p;
memset(num, 0, sizeof(num)); for (int i = 0;i < N; i++) { scanf("%d", &a[i]); Il C[maxn][maxn]; int Combination_table(int n, ll MOD){ memset(C, 0, sizeof(C));		}
for (int $i = 0$; $i < N$; $i++$) { int Combination_table(int n , $ll MOD$) { memset(C , C , sizeof(C));		<pre>II C[maxn][maxn];</pre>
scanf("%d", &a[i]); memset(C, 0, sizeof(C));		
C[0][0]=1.		
	num[a[i]]++;	C[0][0]=1;

```
for(int i=1; i<=n; i++){
                                                  d'Ocagne 性质:
        C[i][0]=1;
                                                  F[k*n+c]=sum
        for(int j=1; j<=i; j++)
                                                  (C[k][i]*F[c-i]*F[n]^i*F[n+1]^(k-i) for i=0 to k)
        C[i][j]=(C[i-1][j-1]+C[i-1][j])%MOD;
    }
                                                  母函数: s(x)=sum
}
                                                  (F[k]*x^k \text{ for } k=0 \text{ to infinity})=x/(1-x-x^2)
                                                  数论性质:
// ===== cont_frac 连分数逼近 =====
                                                  gcd(F[m], F[n])=F[gcd(m, n)]
Il a[maxn];
                                                  整数 N 为 Fibonacci 数的充要条件为 5*N^2+4 或
/* 连分数逼近由欧几里德算法求解
                                                  5*N^2-4 为完全平方数
n/d=a[0]+1/(a[1]+1/(a[2]+1/(a[3]+1/(...+1/a[len-
                                                  p|F[p-(5/p)] (此处括号为 Legendre 标记)
                                                  如果从 1 开始计数,则除了 F[4]=3 以外,若下标
int cont_frac(II n, II d){
                                                  n 为合数则 F[n]也为合数
    Il r;
                                                  除了 1 以外 Fibonacci 数中仅有 8 和 144 为整次
    int len=0;
    while(d){
                                                  除了 1, 8, 144, 所有 Fibonacci 数都有至少一个质
        a[len++]=n/d;
        r=n%d; n=d; d=r;
                                                  因数不在所有比其下标更小的 Fibonacci 数的质
    }
                                                  因数的集合中
    return len;
                                                  若构造一数列 A[i]=F[i]%n, n 为任意正整数,则数
                                                  列 A 有周期性且其周期不超过 6*n
// ====== Euler_Phi =======
int euler_phi(int n){
                                                  int m=int(sqrt(n+0.5)), res=n;
                                                  Legendre 符号 (p 为质数)
    for(int i=2; i<=m; i++) if(n\%i==0){
                                                  (a|p)=0 \text{ if } a\%p=0
        res=res/i*(i-1);
                                                  (a|p)=1 if a%p!=0 and 存在整数 x x^2=a (mod p)
        while(n%i==0) n/=i;
                                                  (a|p)=-1 otherwise
                                                  Euler 准则: 若 p 为奇质数且 p 不能整除 d 则
    if(n>1) res=res/n*(n-1);
                                                  d^{(p-1)/2}=(d|p) \pmod{p}
    return res;
                                                  Legendre 符号是完全积性函数
                                                  二次互反律: 若 p, q 为奇质数,则(q|p)=(p|q)*(-
int phi[maxn];
void phi table(int n){
                                                  1)^((p-1)/2*(q-1)/2)
    memset(phi, 0, sizeof(phi));
    phi[1]=1;
                                                  Mersenne 数
    for(int i=2; i<=n; i++) if(!phi[i])
                                                  M[n]=2^n-1
        for(int j=i; j <= n; j+=i){
                                                  Euclid-Euler 定理: 若 M[p]为素数,则(2^p-
            if(!phi[i]) phi[i]=i;
                                                  1)*2^(p-1)为完全数
            phi[j]=phi[j]/i*(i-1);
                                                  若 p 为奇质数,则 M[p]的所有质因子模 2*p 同
        }
                                                  余1
}
                                                  若 p 为奇质数,则 M[p]的所有质因子模 8 同余
                                                  +/-1
// ====== Fibonacci 数 ======
                                                  M[m]与 M[n]互质的充要条件为 m 与 n 互质
F[0]=F[1]=1;
                                                  若 p 与 2*p+1 皆为素数且 p 模 4 同余 3, 则
F[n]=F[n-1]+F[n-2];
                                                  (2*p+1)为 M[p]的因子
组合性质
F[n]=sum(C[n-k-1][k]  for k=0 to floor((n-1)/2)
                                                  Wilson 定理
C[i][j]表示组合数
                                                  大于 1 的自然数 n 为素数的充要条件为(n-1)!=-1
sum(F[i] for i=1 to n)=F[n+2]-1
                                                  (mod n)
sum(F[2*i+1] \text{ for } i=0 \text{ to } n-1)=F[2*n]
sum(F[2*i] for i=1 to n)=F[2*n+1]-1
                                                  Fermat 多边形数定理
sum(F[i]*F[i] for i=1 to n)=F[n]*F[n+1]
                                                  每一个正整数最多可以表示为 n 个 n 边形数之
Catalan 性质:
F[n]*F[n]-F[n-r]*F[n+r]=(-1)^{(n-r)*F[r]*F[r]}
                                                  和
Vajda 性质: F[n+i]*F[n+j]-F[n]*F[n+i+j]=
```

(-1)^n*F[i]*F[j]

```
Euler 引理
                                                          void print(){
                                                            for(int i=0; i<DIM.c; i++){
对于任意奇素数 p, 同余方程 x^2+y^2+1=0 (mod
                                                              for(int j=0; j<DIM.r; j++)</pre>
p) 必有一组正整数解(x, y)满足 0<=x<p/2,
                                                        cout<<matrix[i][j]<<'\t';
0 <= y < p/2
                                                              cout<<endl;
Lagrange 的四平方和定理
                                                            }
每个正整数均可以表示为 4 个整数的平方和
                                                          }
                                                        };
// ======= log mod ========
                                                        Matrix BigMatrixExpo(Matrix &A, long long n){
//解方程 a^x=b (mod n) n 为素数
                                                          Matrix B=A;
int shank(int a, int b, int n){
                                                          Matrix C(A.DIM.c, A.DIM.r);
    int m, v, e=1, i;
                                                          for(int i=0; i<C.DIM.c; i++)
    m=int(sqrt(n+0.5));//复杂度为
                                                            for(int j=0; j<C.DIM.r; j++)
O((m+n/m)logm) 所以 m==n/m 时最快
                                                              C.matrix[i][j]=i==j;
    v=inv(fast_pow(a, m, n), n);//fast_pow(a, m,
                                                          while(n){
n)=(a^m)%n
                                                            if(n&1) C=C*B;
    map<int, int> x;//x[j]=min(i|e[i]==j)
                                                            B=B*B;
    x[1]=0;
                                                            n>>=1;
    for(int i=1; i<m; i++){
                                                          }
         e=a*e%n;//e=(a^i)%n
                                                          return C;
         if(!x.count(e)) x[e]=i;
                                                        //定义新矩阵 Matrix a(3, 5); a.matrix={{},{},{}};
    for(int i=0; i<m; i++){
                                                        //乘法 c=a*b; 注意 a 的第一个 parametre 等于 b
         //a^(im)到 a^(im+m-1)
                                                        的第二个 parametre;
         if(x.count(b)) return i*m+x[b];
                                                        //加法 c=a+b; //输出 c.print();
         b=b*v%n;//递推更新 b
                                                        //完全积性函数 mo[i*j]=mo[i]*mo[j]
    return -1;//无解
                                                        //sum(mo[d] for d|n)=(n==1)
                                                        //反演公式
// ======= matrix ========
                                                        //若 f(n)=sum(g(d) for d|n) 则
struct parametre{int c, r;};
                                                        g(n)=sum(mo[n/d]*f(d) for d|n)=sum(mo[d]*f(n/d)
struct Matrix{
  long long matrix[maxn][maxn];
                                                        //If f(i)=sum(g(d*i) for d from 1 to floor(n/i)) then
  parametre DIM;
                                                        g(i)=sum(f(d*i)*mo[d] for d from 1 to floor(n/i))
  Matrix(){}
                                                        bool vis[maxn+123];
  Matrix(int c, int r){
                                                        int mo[maxn+123], primes[maxn+123],
    DIM=\{c, r\};
                                                        a[maxn+123], pcnt, N;
    memset(matrix, 0, sizeof(matrix));
                                                        void mobius(){//预处理
                                                            mo[1]=1;
  Matrix operator*(Matrix &A){
                                                            for(int i=2; i <= maxn; i++){
    Matrix C(DIM.c, A.DIM.r);
                                                                 if(!vis[i])
    memset(C.matrix, 0, sizeof(C.matrix));
                                                                 { mo[i]=-1; primes[pcnt++]=i; }
    for(int i=0; i<DIM.c; i++)
                                                                 for(int j=0;
      for(int j=0; j<A.DIM.r; j++)
                                                        j<pcnt&&ll(i)*primes[j]<=maxn; j++){
        for(int k=0; k<DIM.r; k++)
                                                                      vis[i*primes[j]]=true;
C.matrix[i][j]=(C.matrix[i][j]+A.matrix[i][k]*
                                                                      if(i%primes[j]) mo[i*primes[j]]=-
matrix[k][j])%MOD;
                                                        mo[i];
    return C;
                                                                          mo[i*primes[j]]=false;
  Matrix operator+(Matrix &A){
                                                                          break;
    Matrix C(A.DIM.c, A.DIM.r);
                                                                     }
    for(int i=0; i<DIM.c; i++)</pre>
                                                                 }
      for(int j=0; j<DIM.r; j++)
      C.matrix[i][j]=A.matrix[i][j]+matrix[i][j];
                                                            for(int i=2; i<=maxn; i++) mo[i]+=mo[i-1];//mo
    return C;
                                                        记录前缀和
  }
                                                        }
```

```
//O(sqrt(n)+sqrt(m))
                                                             //解存在 A[k][n]里面
Il cnt_gcd(Il n, Il m, Il k){//for i from 1 to n for j
                                                             return 1;
from 1 to m cnt gcd(i, j)=k
                                                        }
    if(n>m) swap(n, m);
    II res=0;
                                                         // ====== pell equation 佩尔方程=====
    n/=k, m/=k;
                                                         //用于求解标准型 Pell 方程的第(k+1)组非平凡
    for(int i=1, j=1; i<=n; i=j+1){
                                                         解 (x^2-n*y^2=1)
         j=min(n/(n/i), m/(m/i));
                                                         //输入 n, k 和 MOD
         res+=ll(mo[j]-mo[i-1])*(n/i)*(m/i);//前缀
                                                         //递推关系为 x[i+1]=x[0]*x[i]+n*y[0]*y[i];
和 Mobius
                                                         //y[i+1]=y[0]*x[i]+x[0]*y[i];
                                                         //上述递推关系可由 sqrt(n)的连分数表示推出
    return res;
                                                         typedef pair<II, II> pii;
}
                                                         pii res;//(xk, yk)
                                                         II MOD;//模<必须是全局变量>
// ========= 高斯消元 ========
                                                         void Find(II n, II& x, II& y){
typedef int Matrix[maxn][maxn];
                                                             //暴力寻找特解(x0, y0)
void exgcd(int a, int b, int& d, int& x, int& y){
                                                             y=1;
    !b?(d=a, x=1, y=0):(exgcd(b, a%b, d, y, x), y-
                                                             while(true){
=x*(a/b));
                                                                  x=sqrt(y*y*n+1);
                                                                  if(x*x-n*y*y==1) break;
int inv(int a){
                                                                  y++;
    int d, x, y;
                                                             }
    exgcd(a, MOD, d, x, y);
    return d==1?(x+MOD)%MOD:-1;
                                                         struct parameter{int c, r;};
                                                         struct Matrix{
int gauss_jordan(Matrix A, int n, int m){//A 是增广
                                                           II matrix[maxn][maxn];
矩阵, n 个未知数, m 个方程, MOD 是模, 如果
                                                           parameter DIM;
MOD 不是质数的话每次 inv 完先检测是否是-1
                                                           Matrix(){}
    int i=0, j=0;
                                                           Matrix(int c, int r);
    while(i<m&&j<n){
                                                           Matrix operator*(Matrix &A);//带模乘法
         int row=i;
                                                           Matrix operator+(Matrix &A);
         for(int k=i; k<m; k++){
                                                           void print();
              if(A[k][j]){
                                                         };
                  row=k:
                                                         Matrix BigMatrixExpo(Matrix &A, II n);//带模快速
                  break;
              }
                                                         bool Pell(|| n, || k){//k 为第 k 组解, 从 0 开始数
         }
                                                             II t=sqrt(n)+0.5, x, y;
         if(row!=i) for(int k=0; k<=n; k++)
                                                             if(t*t==n) return false;//仅有平凡解 (1, 0) 和
              swap(A[i][k], A[row][k]);
         if(!A[i][j]){
                                                         (-1, 0)
                                                             Matrix A(2, 2);
             j++; continue;
                                                             Find(n, x, y);
         }
                                                             A.matrix[0][0]=A.matrix[1][1]=x;
         for(int k=0; k< m; k++){
                                                             A.matrix[0][1]=n*y;
              if(!A[k][j]||i==k) continue;
                                                             A.matrix[1][0]=y;
              int cur=A[k][j]*inv(A[i][j])%MOD;
                                                             A=BigMatrixExpo(A, k-1);
              for(int t=j; t<=n; t++)
                                                             res=make_pair((A.matrix[0][0]*x+A.matrix[0][
              A[k][t]=((A[k][t]-cur*A[i][t])
                                                         1]*y)%MOD,
%MOD+MOD)%MOD;
                                                         (A.matrix[1][0]*x+A.matrix[1][1]*y)%MOD);
         }
                                                             return true;
         i++;
    for(int k=i; k<m; k++)
         if(A[k][n]) return -1;//无解
    if(i<n) return 0;//无限解
    for(int k=0; k<n; k++)
         A[k][n]=A[k][n]*inv(A[k][k])%MOD;
```

// ======== CRT========	{return *this=*this-rhs;}
typedef long long ll;	fraction operator - (const int &rhs) const
//n 个方程为 x=a[i] (mod m[i])	{fraction r(rhs, 1);return *this-r;}
Il china(int n, int* a, int* m){	fraction operator -= (const int &rhs)
Ⅱ M=1, d, y, x=0;//M 是等价以后的模	{return *this=*this-rhs;}
for(int i=0; i <n; i++)="" m*="m[i];</td"><td>fraction operator * (const fraction &rhs) const</td></n;>	fraction operator * (const fraction &rhs) const
for(int i=0; i <n; i++){<="" td=""><td>fraction res;</td></n;>	fraction res;
II w=M/m[i];	res.num=num*rhs.num;
exgcd(m[i], w, d, d, y);	res.den=den*rhs.den;
x=(x+y*w*a[i])%M;	res.simplify();
}	return res;
return (x+M)%M;	}
}	fraction operator *= (const fraction &rhs)
// ====== Fraction ======	{return *this=(*this)*rhs;}
<pre>Il gcd(a, b){ return !b?a:gcd(b, a%b); }</pre>	fraction operator * (const int &rhs) const
struct fraction{	{fraction r(rhs, 1); return (*this)*r;}
Il num, den;	fraction operator *= (const int &rhs)
fraction(){ num=0; den=1; }	{return *this=(*this)*rhs;}
fraction(II a, II b)	fraction operator / (const fraction &rhs) const
{num=a; den=b; simplify();}	fraction res;
inline void reset(){ num=0; den=1;}	res.num=num*rhs.den;
void simplify(){	res.den=den*rhs.num;
<pre>II d=gcd(num, den);</pre>	res.simplify();
num/=d;	return res;
den/=d;	}
if(den<0){num=-num;den=-den;}	fraction operator /= (const fraction &rhs)
}	{return *this=(*this)/rhs;}
inline Il convert(){return num/den;}	fraction operator / (const int &rhs) const
fraction& operator = (int rhs){	{ fraction r(rhs, 1); return (*this)/r; }
(*this).num=rhs;	fraction operator /= (const int &rhs)
(*this).den=1;	{return *this=(*this)/rhs;}
return *this;	bool operator == (const fraction &rhs) const
}	{return num*rhs.den==den*rhs.num;}
fraction operator + (const fraction &rhs) const{	bool operator == (const int &rhs) const
fraction res;	{return num==den*rhs;}
res.den=lcm(den, rhs.den);	bool operator != (const fraction &rhs) const
res.num=res.den/den*num+res.den/rhs.den*rhs.n	{return !(*this==rhs);}
um;	bool operator != (const int &rhs) const
res.simplify();	{return !(*this==rhs);}
return res;	bool operator < (const fraction &rhs) const
}	{return num*rhs.den <den*rhs.num;}< td=""></den*rhs.num;}<>
fraction operator += (const fraction &rhs)	bool operator < (const int &rhs) const
{return *this=*this+rhs;}	{return num <den*rhs;}< td=""></den*rhs;}<>
fraction operator + (const int &rhs) const{	bool operator > (const fraction &rhs) const
fraction r(rhs, 1);	{return num*rhs.den>den*rhs.num;}
return *this+r;	bool operator > (const int& rhs) const
}	{return num>den*rhs;}
fraction operator += (const int &rhs)	bool operator <= (const fraction &rhs) const
{return *this=*this+rhs;}	{return *this==rhs *this <rhs;}< td=""></rhs;}<>
fraction operator - (const fraction &rhs) const{	bool operator <= (const int& rhs) const
fraction res;	{return *this==rhs *this <rhs;}< td=""></rhs;}<>
res=*this+fraction(-1, 1)*rhs;	bool operator >= (const fraction &rhs) const
res.simplify();	{return *this>rhs *this==rhs;}
return res;	bool operator >= (const int &rhs) const
}	{return *this>rhs *this==rhs;}
fraction operator -= (const fraction &rhs)	};

int convexhull(Point* p, int n, Point* ch) {

// ======== 辛普森积分 =======

```
sort(p, p+n);
double simpson(double a, double b) {
                                                               int m = 0;
     double c = (a + b) / 2.0;
     return (F(a)+4*F(c)+F(b)) * (b-a) / 6.0;
                                                               for(int i = 0; i < n; i++) {
                                                                 while (m>1 \&\& cross(ch[m-1]-ch[m-2], p[i]-
}// 这里 F 为自定义函数
                                                             ch[m-2]) <= 0) m--;
//given A as the simpson Value for the whole
                                                                 ch[m++] = p[i];
interval [a,b]
                                                               }
double asr(double a, double b, double eps, double
                                                               int k = m;
A) {
                                                               for(int i = n-2; i >= 0; i--) {
     double c = (a + b) / 2.0;
                                                                 while(m > k \&\& cross(ch[m-1]-ch[m-2], p[i]-
     double L = simpson(a, c);
                                                             ch[m-2]) <= 0) m--;
     double R = simpson(c, b);
                                                                 ch[m++] = p[i];
     if (fabs(L+R-A) \le 15*eps)
          return L + R + (L+R-A)/15.0;
                                                               if(n > 1) m--; return m;
return asr(a, c, eps/2, L) + asr(c, b, eps/2, R);
                                                             Vector Rotate(Vector A, double rad){
double asr(double a, double b, double eps)
                                                                 // 这里 rad 是逆时针旋转的角度
{ return asr(a, b, eps, simpson(a, b)); } //接口
                                                                 return Vector(A.x*cos(rad)-A.y*sin(rad),
// int main(): 调用 asr(left, right, 1e-5)
                                                                 A.x*sin(rad)+A.y*cos(rad));
// 得到 F(x) 在[left, right]上的积分 eps 也可改为
                                                             }
1e-6
                                                             // int main()
                                                             Point o(tmpx, tmpy);
// ========= 凸包 ========
                                                             p[point_cnt++]=o; ... ...
const double eps=1e-10;
                                                             int m=convexhull(p, point_cnt, ch);
const double PI=acos(-1);
                                                             double convex_area=PolygonArea(ch, m);
struct Point{ double x, y;
                                                             // Rotate vector(10,10) clockwise by 90 degree
  Point(double x=0, double y=0):x(x), y(y){}
                                                             // new o = o + Rotate(Vector(10.0,10.0),-
} p[maxn], ch[maxn];
                                                             torad(90.0));
typedef Point Vector;
Vector operator + (Vector A, Vector B)
                                                             // ======= 点在多边形内 =======
     { return Vector(A.x+B.x, A.y+B.y); }
                                                             bool PNPoly(int u, int deg) {
Vector operator - (Vector A, Vector B)
                                                                 if (! (vertxmin \leq x[u] \leq vertxmax) || !
     { return Vector(A.x-B.x, A.y-B.y); }
                                                             ( vertymin <= y[u] <= vertymax ) ) return 0;
Vector operator * (Vector A, double p)
                                                                 bool is_in = 0; int i,j;
     { return Vector(A.x*p, A.y*p); }
                                                                 for(i=0;i<deg;i++) {
Vector operator / (Vector A, double p)
                                                                 if(!i) j= deg-1;
     { return Vector(A.x/p, A.y/p); }
                                                                  else j= i-1;
int dcmp(double x) {
                                                                 if ((poy[i] > y[u]) != (poy[j] > y[u])) && (x[u] <
  if(fabs(x) < eps) return 0;
                                                             (pox[j] - pox[i]) * (y[u] - poy[i]) / (poy[j] - poy[i]) +
  else return x < 0 ? -1 : 1; }
                                                             pox[i]) )
bool operator == (const Point& a, const Point& b)
                                                                           is_in = ! is_in;
\{ return dcmp(a.x-b.x) == 0 \&\& dcmp(a.y-b.y) == 
                                                                 }
                                                                 return is_in;
bool operator < (const Point& a, const Point& b)
                                                             }
{ return a.x < b.x | | (a.x == b.x \&\& a.y < b.y); }
double cross(Vector A, Vector B)
     { return A.x*B.y - A.y*B.x; }
double torad(double deg)
     { return deg / 180 * PI; }
double PolygonArea(Point* p, int n){
  double area=0:
  for(int i=1; i<n-1; i++)
    area+=cross(p[i]-p[0], p[i+1]-p[0]);
  return area/2;
}
```