目录

计算几何

1. 基本

```
//浮点几何函数库
#include <math.h>
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)

struct point{
	double x,y;
	point(double x = 0, double y = 0): x(x), y(y) {};
	point operator +(const point &l) const {
		return point(x + l.x, y + l.y);
	}
	point operator -(const point &l) const {
		return point(x - l.x, y - l.y);
	}
	point operator /(const double &l) const {
		return point(x / l, y / l);
	}
	point operator *(const double &l) const {
		return point(x * l, y * l);
	}
	double len() {
		return sqrt(x * x + y * y);
	}
	bool operator <(const point &l)const {
		return x < l.x;
	}
};
struct line{
	point a,b;
	line() {};
	line(point a, point b): a(a), b(b) {};
};


//计算 cross product (P1-P0)x(P2-P0)
double xmult(point p1,point p2,point p0){
	return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
```

```
double xmult(double x1,double y1,double x2,double y2,double x0,double y0){
	return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
}

//计算 dot product (P1-P0).(P2-P0)
double dmult(point p1,point p2,point p0){
	return (p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y);
}
double dmult(double x1,double y1,double x2,double y2,double x0,double y0){
	return (x1-x0)*(x2-x0)+(y1-y0)*(y2-y0);
}

//两点距离
double dist2(point p1,point p2){
	return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double dist2(double x1,double y1,double x2,double y2){
	return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}

//判三点共线
int dots_inline(point p1,point p2,point p3){
	return zero(xmult(p1,p2,p3));
}
int dots_inline(double x1,double y1,double x2,double y2,double x3,double y3){
	return zero(xmult(x1,y1,x2,y2,x3,y3));
}

//判点是否在线段上,包括端点
int dot_online_in(point p,line l){
	return zero(xmult(p,l.a,l.b))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&(l.a.y-p.y)*(l.b.y-p.y)<eps;
}
int dot_online_in(point p,point l1,point l2){
	return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}
int dot_online_in(double x,double y,double x1,double y1,double x2,double y2){
	return zero(xmult(x,y,x1,y1,x2,y2))&&(x1-x)*(x2-x)<eps&&(y1-y)*(y2-y)<eps;
}

//判点是否在线段上,不包括端点
int dot_online_ex(point p,line l){
	return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y))&&(!zero(p.x-l.b.x)||!zero(p.y-l.b.y));
```

```
}

int dot_online_ex(point p,point l1,point l2){
	return	dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y))&&(!zero(p.x-l2.x)||!zero(p.y-l2.y));
}
int dot_online_ex(double x,double y,double x1,double y1,double x2,double y2){
	return	dot_online_in(x,y,x1,y1,x2,y2)&&(!zero(x-x1)||!zero(y-y1))&&(!zero(x-x2)||!zero(y-y2));
}

//判两点在线段同侧,点在线段上返回 0
int same_side(point p1,point p2,line l){
	return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)>eps;
}
int same_side(point p1,point p2,point l1,point l2){
	return xmult(l1,p1,l2)*xmult(l1,p2,l2)>eps;
}

//判两点在线段异侧,点在线段上返回 0
int opposite_side(point p1,point p2,line l){
	return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)<-eps;
}
int opposite_side(point p1,point p2,point l1,point l2){
	return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}

//判两直线平行
int parallel(line u,line v){
	return zero((u.a.x-u.b.x)*(v.a.y-v.b.y)-(v.a.x-v.b.x)*(u.a.y-u.b.y));
}
int parallel(point u1,point u2,point v1,point v2){
	return zero((u1.x-u2.x)*(v1.y-v2.y)-(v1.x-v2.x)*(u1.y-u2.y));
}

//判两直线垂直
int perpendicular(line u,line v){
	return zero((u.a.x-u.b.x)*(v.a.x-v.b.x)+(u.a.y-u.b.y)*(v.a.y-v.b.y));
}
int perpendicular(point u1,point u2,point v1,point v2){
	return zero((u1.x-u2.x)*(v1.x-v2.x)+(u1.y-u2.y)*(v1.y-v2.y));
}

//判两线段相交,包括端点和部分重合
```

```
int intersect_in(line u,line v){
	if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
		return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
	return
dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||dot_online_in(v.b
,u);
}
int intersect_in(point u1,point u2,point v1,point v2){
	if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
		return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
	return
dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||dot_online_in(v1,u1,u2)||dot_o
nline_in(v2,u1,u2);
}

//判两线段相交,不包括端点和部分重合
int intersect_ex(line u,line v){
	return opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
}
int intersect_ex(point u1,point u2,point v1,point v2){
	return opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
}

//计算两直线交点,注意事先判断直线是否平行!
//线段交点请另外判线段相交(同时还是要判断是否平行!)
point intersection(line u,line v){
	point ret=u.a;
	double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
			/((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
	ret.x+=(u.b.x-u.a.x)*t;
	ret.y+=(u.b.y-u.a.y)*t;
	return ret;
}
point intersection(point u1,point u2,point v1,point v2){
	point ret=u1;
	double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
			/((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
	ret.x+=(u2.x-u1.x)*t;
	ret.y+=(u2.y-u1.y)*t;
	return ret;
}

//点到直线上的最近点
point ptoline(point p,line l){
```

```
	point t=p;
	t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
	return intersection(p,t,l.a,l.b);
}
point ptoline(point p,point l1,point l2){
	point t=p;
	t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
	return intersection(p,t,l1,l2);
}

point symmetric_point(point p1, point l1, point l2) {
	 point ret = ptoline(p1, l1, l2);
	 ret.x = 2 * ret.x - p1.x;
	 ret.y = 2 * ret.y - p1.y;
	 return ret;
}

//点到直线距离
double disptoline(point p,line l){
	return fabs(xmult(p,l.a,l.b))/dist2(l.a,l.b);
}
double disptoline(point p,point l1,point l2){
	return fabs(xmult(p,l1,l2))/dist2(l1,l2);
}
double disptoline(double x,double y,double x1,double y1,double x2,double y2){
	return fabs(xmult(x,y,x1,y1,x2,y2))/dist2(x1,y1,x2,y2);
}

//点到线段上的最近点
point ptoseg(point p,line l){
	point t=p;
	t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
	if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
		return dist2(p,l.a)<dist2(p,l.b)?l.a:l.b;
	return intersection(p,t,l.a,l.b);
}
point ptoseg(point p,point l1,point l2){
	point t=p;
	t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
	if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
		return dist2(p,l1)<dist2(p,l2)?l1:l2;
	return intersection(p,t,l1,l2);
}
```

```
//点到线段距离
double disptoseg(point p,line l){
	point t=p;
	t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
	if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
		return dist2(p,l.a)<dist2(p,l.b)?dist2(p,l.a):dist2(p,l.b);
	return fabs(xmult(p,l.a,l.b))/dist2(l.a,l.b);
}
double disptoseg(point p,point l1,point l2){
	point t=p;
	t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
	if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
		return dist2(p,l1)<dist2(p,l2)?dist2(p,l1):dist2(p,l2);
	return fabs(xmult(p,l1,l2))/dist2(l1,l2);
}

//矢量 V 以 P 为顶点逆时针旋转 angle 并放大 scale 倍
point rotate(point v,point p,double angle,double scale){
	point ret=p;
	v.x-=p.x,v.y-=p.y;
	p.x=scale*cos(angle);
	p.y=scale*sin(angle);
	ret.x+=v.x*p.x-v.y*p.y;
	ret.y+=v.x*p.y+v.y*p.x;
	return ret;
}

/*----------------------------------------------------------*/


//判直线和圆相交,包括相切
int intersect_line_circle(point c,double r,point l1,point l2){
	return disptoline(c,l1,l2)<r+eps;
}

//判线段和圆相交,包括端点和相切
int intersect_seg_circle(point c,double r,point l1,point l2){
	double t1=dist2(c,l1)-r,t2=dist2(c,l2)-r;
	point t=c;
	if (t1<eps||t2<eps)
		return t1>-eps||t2>-eps;
	t.x+=l1.y-l2.y;
	t.y+=l2.x-l1.x;
	return xmult(l1,c,t)*xmult(l2,c,t)<eps&&disptoline(c,l1,l2)-r<eps;
```

```
}

//判圆和圆相交,包括相切
int intersect_circle_circle(point c1,double r1,point c2,double r2){
	return dist2(c1,c2)<r1+r2+eps&&dist2(c1,c2)>fabs(r1-r2)-eps;
}

//计算圆上到点 p 最近点,如 p 与圆心重合,返回 p 本身
point dot_to_circle(point c,double r,point p){
	point u,v;
	if (dist2(p,c)<eps)
		return p;
	u.x=c.x+r*fabs(c.x-p.x)/dist2(c,p);
	u.y=c.y+r*fabs(c.y-p.y)/dist2(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
	v.x=c.x-r*fabs(c.x-p.x)/dist2(c,p);
	v.y=c.y-r*fabs(c.y-p.y)/dist2(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
	return dist2(u,p)<dist2(v,p)?u:v;
}

//计算直线与圆的交点,保证直线与圆有交点
//计算线段与圆的交点可用这个函数后判点是否在线段上
void intersection_line_circle(point c,double r,point l1,point l2,point& p1,point& p2){
	point p=c;
	double t;
	p.x+=l1.y-l2.y;
	p.y+=l2.x-l1.x;
	p=intersection(p,c,l1,l2);
	t=sqrt(r*r-dist2(p,c)*dist2(p,c))/dist2(l1,l2);
	p1.x=p.x+(l2.x-l1.x)*t;
	p1.y=p.y+(l2.y-l1.y)*t;
	p2.x=p.x-(l2.x-l1.x)*t;
	p2.y=p.y-(l2.y-l1.y)*t;
}

//计算圆与圆的交点,保证圆与圆有交点,圆心不重合
void intersection_circle_circle(point c1,double r1,point c2,double r2,point& p1,point&
p2){
	point u,v;
	double t;
	t=(1+(r1*r1-r2*r2)/dist2(c1,c2)/dist2(c1,c2))/2;
	u.x=c1.x+(c2.x-c1.x)*t;
	u.y=c1.y+(c2.y-c1.y)*t;
	v.x=u.x+c1.y-c2.y;
	v.y=u.y-c1.x+c2.x;
```

```
    intersection_line_circle(c1,r1,u,v,p1,p2);
}

//求过三点的圆
point getcir3(point aa, point bb, point cc) {
    double x1 = aa.x, x2 = bb.x, x3 = cc.x;
    double y1 = aa.y, y2 = bb.y, y3 = cc.y;
    double a, b, c, d, e, f, x, y, r;
    a=2*(bb.x-aa.x);
    b=2*(bb.y-aa.y);
    c=x2*x2+y2*y2-x1*x1-y1*y1;
    d=2*(x3-x2);
    e=2*(y3-y2);
    f=x3*x3+y3*y3-x2*x2-y2*y2;
    x=(b*f-e*c)/(b*d-e*a);
    y=(d*c-a*f)/(b*d-e*a);
    r=sqrt((x-x1)*(x-x1)+(y-y1)*(y-y1));
    return point(x, y, r, 0);
}
```

2. 二维凸包-水平序

```
// 求凸包_水平序

#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>

#define MAX 50010
#define EQ(x, y) (fabs((x) - (y)) < eps)

using namespace std;

const double eps = 1e-8;

struct Point {
    double x, y;

    Point() {}
    Point(const double _x, const double _y) : x(_x), y(_y) {}
```

```
	Point operator -(const Point &p) const {
		return Point(x - p.x, y - p.y);
	}

	double xMul(const Point &p) const {
		return x * p.y - y * p.x;
	}
} p[MAX];

int hull[MAX], cnt;

bool cmp(const Point &a, const Point &b) {
	return EQ(a.x, b.x) ? a.y < b.y : a.x < b.x;
}

bool check(const Point &a, const Point &b, const Point &c) {
	return (c - b).xMul(b - a) < 0;
}

void graham(const int n) {
	int i, m;

	sort(p, p + n, cmp);
	cnt = 0;
	for (i = 0; i < n; ++i) {
		while (cnt > 1 && !check(p[hull[cnt - 2]], p[hull[cnt - 1]], p[i])) --cnt;
		hull[cnt++] = i;
	}
	m = cnt;
	for (i = n - 1; ~i; --i) {
		while (cnt > m && !check(p[hull[cnt - 2]], p[hull[cnt - 1]], p[i])) --cnt;
		hull[cnt++] = i;
	}
	--cnt;
}
```

3. 点关于直线对称点

```
//求点 c 对于直线 ab 的对称点

node cal(node a, node b, node c) {
	node tmp;
	tmp.x = (b.x - a.x) * (b.x - a.x) * c.x + 2 * (b.y - a.y) * (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (b.y - a.y) * (c.x - 2 * a.x);
```

```
		tmp.x /= ((b.x - a.x) * (b.x - a.x) + (b.y - a.y) * (b.y - a.y));
		tmp.y = 2 * (b.x - a.x) * (b.y - a.y) * (c.x - a.x) + (b.y - a.y) * (b.y - a.y) * c.y + (b.x - a.x) * (b.x - a.x) * (2 * a.y - c.y);
		tmp.y /= ((b.x - a.x) * (b.x - a.x) + (b.y - a.y) * (b.y - a.y));
	return tmp;
}
```

4. 随机增量最小圆覆盖

```
//随机增量法求最小圆覆盖，近似 O(n)，使用前需随机播种
//center 为圆心，返回 ret 为半径

#define MAXN 1010

#include <cstdio>
#include <cmath>
#include <ctime>
#include <cstdlib>

struct node {
	double x, y;
} dots[MAXN], center;

double dis(node a, node b) {
	return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

node getpoint(node a, node b, node c) {
	node h, q, p;
	h.x = (a.x + b.x) / 2;
	h.y = (a.y + b.y) / 2;
	q.x = (b.x + c.x) / 2;
	q.y = (b.y + c.y) / 2;
	p.x = (b.y - a.y) * (h.y * (b.y - c.y) - q.x * (b.x - c.x)) + (b.y - c.y) * (h.x * (b.x - a.x) - q.y * (b.y - a.y));
	p.x /= ((b.x - a.x) * (b.y - c.y) - (b.y - a.y) * (b.x - c.x));
	p.y = (b.x - c.x) * (h.y * (b.y - a.y) - q.x * (b.x - a.x)) + (b.x - a.x) * (h.x * (b.x - c.x) - q.y * (b.y - c.y));
	p.y /= ((b.y - a.y) * (b.x - c.x) - (b.y - c.y) * (b.x - a.x));
	return p;
}

double solve(int n) {
```

```
int i, j, k;
double ret;

if (n == 1) {
	ret = 0;
	center = dots[0];
}
else if (n == 2) {
	ret = dis(dots[0], dots[1]) / 2;
	center.x = (dots[0].x + dots[1].x) / 2;
	center.y = (dots[0].y + dots[1].y) / 2;
}
else {
	for (i = 1; i < n; i++) {
		j = rand() % i;
		node tmp = dots[i];
		dots[i] = dots[j];
		dots[j] = tmp;
	}
	ret = dis(dots[0], dots[1]) / 2;
	center.x = (dots[0].x + dots[1].x) / 2;
	center.y = (dots[0].y + dots[1].y) / 2;
	for (i = 2; i < n; i++) {
		if (dis(dots[i], center) > ret) {
	    center.x = (dots[i].x + dots[i - 1].x) / 2;
			center.y = (dots[i].y + dots[i - 1].y) / 2;
			ret = dis(dots[i], dots[i - 1]) / 2;
			for (j = i - 2; j >= 0; j--) {
				if (dis(dots[j], center) > ret) {
					center.x = (dots[i].x + dots[j].x) / 2;
					center.y = (dots[i].y + dots[j].y) / 2;
					ret = dis(dots[i], dots[j]) / 2;
					for (k = j + 1; k < i; k++) {
						if (dis(dots[k], center) > ret) {
							center = getpoint(dots[i], dots[j], dots[k]);
							ret = dis(dots[i], center);
						}
					}
				}
			}
		}
	}
}
return ret;
```

```
}
```

5. 随机增量最近点对

```
//随机增量法求平面最近点对
//ans 为最近两点距离，O(n)

#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <cstdlib>
#include <ctime>

#define MAX 100100
#define MAXP 100003
#define P 359

using namespace std;

const int stepx[] = {-1, -1, -1, 0, 0, 0, 1, 1, 1};
const int stepy[] = {-1, 0, 1, -1, 0, 1, -1, 0, 1};

struct Point {
	double x, y;
} p[MAX];

struct HashNode {
	int x, y, idx, next;
} hash[MAX];

int head[MAX], tot;

double ans;

int getKey(const int x, const int y) {
	return abs((x * P) + y) % MAXP;
}

void init() {
	memset(head, -1, sizeof(head));
	tot = 0;
}
```

```
double dis(const Point &a, const Point &b) {
	return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

void sherwood(const int n) {
	int i, r;

	for (i = 0; i < n; ++i) {
		r = rand() % n;
		if (i != r) swap(p[i], p[r]);
	}
}

void getGrid(const Point &a, int &x, int &y) {
	x = (int)(a.x / ans);
	y = (int)(a.y / ans);
}

void insert(const int k) {
	int x, y, key;

	getGrid(p[k], x, y);
	key = getKey(x, y);
	hash[tot].x = x;
	hash[tot].y = y;
	hash[tot].idx = k;
	hash[tot].next = head[key];
	head[key] = tot++;
}

bool update(const int k) {
	int i, j, key, x, y, tx, ty;
	double nowLen;
	bool tag = false;

	getGrid(p[k], x, y);
	for (i = 0; i < 9; ++i) {
		tx = x + stepx[i];
		ty = y + stepy[i];
		key = getKey(tx, ty);
		for (j = head[key]; ~j; j = hash[j].next) {
			if (tx == hash[j].x && ty == hash[j].y) {
				nowLen = dis(p[k], p[hash[j].idx]);
				if (ans > nowLen) {
```

```
                    ans = nowLen;
                    tag = true;
                }
            }
        }
    }

    return tag;
}

void rebuild(const int n) {
    int i, x, y;
    init();
    for (i = 0; i < n; ++i) insert(i);
}

void solve(const int n) {
    int i;

    sherwood(n);
    init();
    ans = dis(p[0], p[1]);
    if (ans < 1e-8) return;
    rebuild(2);
    for (i = 2; i < n; ++i) {
        if (update(i)) {
            if (ans < 1e-8) return;
            rebuild(i + 1);
        } else {
            insert(i);
        }
    }
}

int main() {
    int n, i;

    srand((unsigned int)(time(0)));
    while (scanf("%d", &n) && n) {
        for (i = 0; i < n; ++i) scanf("%lf %lf", &p[i].x, &p[i].y);
        solve(n);
        printf("%.2lf\n", ans);
    }
```

```
	return 0;
}
```

6. 半平面交排序增量

```
// 半平面交_排序增量法_O(nlogn)
// 返回 true 表示 <= 不等式存在解，< 需要用可行域面积 > 判定
// init 函数根据需要修改，加入可行域外边框，默认为加入 INF
// hull 记录了结果点集，cnt 为总点数，hull[cnt] = hull[0]

#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>

#define MAX 10010
#define INF 1000000000
#define ZERO(x) (fabs(x) < eps)
#define EQ(x, y) (fabs((x) - (y)) < eps)

using namespace std;

const double eps = 1e-8;

struct Point {
	double x, y;

	Point() {}
	Point(const double _x, const double _y) : x(_x), y(_y) {}

	bool operator ==(const Point &p) const {
		return EQ(x, p.x) && EQ(y, p.y);
	}

	Point operator -(const Point &p) const {
		return Point(x - p.x, y - p.y);
	}

	double xMul(const Point &p) const {
		return x * p.y - y * p.x;
	}
} hull[MAX];
```

```
int cnt;

struct Hvec {
	Point u, v;
	double ang;
} h[MAX];

int que[MAX], front, tail, nHvec;

double u[MAX], v[MAX], w[MAX];

bool cmp(const Hvec &a, const Hvec &b) {
	return EQ(a.ang, b.ang) ? (b.v - b.u).xMul(a.v - b.u) >= 0 : a.ang < b.ang;
}

bool isOut(const Hvec &h, const Point &p) {
	return (p - h.u).xMul(h.v - h.u) > 0;
}

bool parallel(const Hvec &a, const Hvec &b) {
	return ZERO((a.v - a.u).xMul(b.v - b.u));
}

Point inter(const Hvec &a, const Hvec &b) {
	double s1 = (b.v - a.u).xMul(b.u - a.u), s2 = (b.u - a.v).xMul(b.v - a.v);
	return Point((a.u.x * s2 + a.v.x * s1) / (s2 + s1), (a.u.y * s2 + a.v.y * s1) / (s2 + s1));
}

// 根据两点添加约束，可行域在 a->b 左侧
void addHvec(const Point &a, const Point &b) {
	h[nHvec].u = a;
	h[nHvec].v = b;
	h[nHvec++].ang = atan2((b - a).y, (b - a).x);
}

// 根据不等式添加约束 ax + by + c < 0，其中 a b 不能同时为 0
void addHvec(const double a, const double b, const double c) {
	if (ZERO(a)) {
		addHvec(Point(b > 0, -c / b), Point(b < 0, -c / b));
	} else if (ZERO(b)) {
		addHvec(Point(-c / a, a < 0), Point(-c / a, a > 0));
	} else if (ZERO(c)) {
		addHvec(Point(0, 0), Point(-b, a));
	} else {
```

```
			if ((a > 0) ^ (b > 0) ^ (c > 0)) addHvec(Point(0, -c / b), Point(-c / a, 0));
			else addHvec(Point(-c / a, 0), Point(0, -c / b));
		}
}

void init() {
		nHvec = 0;
		addHvec(Point(-INF, -INF), Point(INF, -INF)),
		addHvec(Point(INF, -INF), Point(INF, INF)),
		addHvec(Point(INF, INF), Point(-INF, INF)),
		addHvec(Point(-INF, INF), Point(-INF, -INF));
}

bool hvecInt() {
		int i, s;

		sort(h, h + nHvec, cmp);
		for (s = i = 1; i < nHvec; ++i) {
			if (!EQ(h[i].ang, h[i - 1].ang)) h[s++] = h[i];
		}
		cnt = front = tail = 0;
		que[tail++] = 0;
		que[tail++] = 1;
		for (i = 2; i < s; ++i) {
			while (tail - front > 1 && isOut(h[i], inter(h[que[tail - 1]], h[que[tail - 2]]))) --tail;
			while (tail - front > 1 && isOut(h[i], inter(h[que[front]], h[que[front + 1]]))) ++front;
			if (parallel(h[i], h[que[tail - 1]])) return false;
			que[tail++] = i;
		}
		while (tail - front > 1 && isOut(h[que[front]], inter(h[que[tail - 1]], h[que[tail - 2]]))) --tail;
		while (tail - front > 1 && isOut(h[que[tail - 1]], inter(h[que[front]], h[que[front + 1]]))) ++front;

		if (tail - front < 3) return false;
		for (i = front; i < tail - 1; ++i) hull[cnt++] = inter(h[que[i]], h[que[i + 1]]);
		hull[cnt++] = inter(h[que[front]], h[que[tail - 1]]);
		cnt = unique(hull, hull + cnt) - hull;
		if (cnt > 1 && hull[0] == hull[cnt - 1]) --cnt;
		else hull[cnt] = hull[0];

		return true;
```

```
}
```

数学

1. 求解三次函数

```
//求解三次函数

double solve(double a, double b, double c, double d) {
	complex<double> alpha = b * c / (6. * a * a) - b * b * b / (27. * a * a * a) - d / (2. * a);
	complex<double> beta = c / (3 * a) - b * b / (9 * a * a);
	complex<double> delta = alpha * alpha + beta * beta * beta;
	complex<double> u = complex<double>(-0.5, sqrt(3.0) / 2.0);
	complex<double> v = complex<double>(-0.5, -sqrt(3.0) / 2.0);
	complex<double> p = pow(alpha + sqrt(delta), 1.0 / 3.0);
	complex<double> q = pow(alpha - sqrt(delta), 1.0 / 3.0);
	complex<double> x1 = -b / (3. * a) + p + q;
	complex<double> x2 = -b / (3. * a) + u * p + v * q;
	complex<double> x3 = -b / (3. * a) + v * p + u * q;
	return max(x1.real(), max(x2.real(), x3.real()));
}
```

2. 高斯消元

```
//实数高斯消元：
double a[maxn][maxn],x[maxn],freex[maxn];

int gauss(int equ,int var){                //总共 equ 个等式，var 个变量，第 var+1 为等号右侧的值
	int i,j,k,maxr,col;
	for (k=col=1; k<=equ && col<=var; k++,col++){
		if (fabs(a[k][col])<eps){
			for (maxr=k,i=k+1; i<=equ; i++)
				if (fabs(a[i][col])>=eps){ maxr=i; break;}
			if (maxr==k) { freex[col]=1; continue;}
			for (i=col; i<=var+1; i++)    swap(a[k][i],a[maxr][i]);
		}                                      //经此处理后的矩阵形式：
		for (i=1; i<=equ; i++)  if (i!=k){           //1.0 1.0 0.0 0.0 0.0 . . .
```

```
3.0
            double tem=a[i][col]/a[k][col];                //0.0 0.0 1.1 0.0 0.0 . . .
1.3
              for (j=col; j<=var+1; j++) a[i][j]-=tem*a[k][j];//0.0 0.0 0.0 0.0 4.3 . . . 2.1
          }                                                           //0.0 0.0 0.0
0.0 0.0 . . . 0.0
    }                                                                 //0.0 0.0 0.0
0.0 0.0 . . . 0.0
    for (i=k; i<=equ; i++) if (fabs(a[i][var+1])>=eps) return -1;             //  无
解
    for (i=var; i>=1; i--)    if (!freex[i]){
         x[i]=a[i][var+1]/a[i][i];                              //判是否也是自由
变元的时候使用
         for (j=i+1; j<=var; j++) if (fabs(a[i][j])>eps && freex[j]) freex[i]=1;
     }
    return 1;
}

//异或高斯消元,每行数进行了压缩
int a[maxn];
lld gauss(int equ,int var){          //总共 equ 个等式，var 个变量(0..var-1)，第 var
为等号右侧的值
    lld i,j,k,tem;
    for (k=1,i=var-1; i>=0; i--){//从高位开始消
        for (j=k; j<=equ; j++)  if ((a[j]>>i)&1) break;
        if (j<=equ){
            swap(a[k],a[j]);
            for (j=1; j<=var; j++) if (j!=k && ((a[j]>>i)&1)) a[j]^=a[k];
            k++;                //整数位之间的运算，时间近似 O(1)
        }
    }
    return k-1;                       //返回多少个异或后的等式
}

//***********************************************************************
**************
//整数高斯消元，用最小公倍数进行处理，包括枚举自由变元
lld a[maxn][maxn],x[maxn];               //数组里存的是模 7 的结果，其他情况
改为模 p

    //深搜，从下向上，k 为第几个非 0 行，pos 当前处理到的变元 x[pos],num 总
数，var 总变量个数
void dfs(int k,int pos,int num,int var){  //与下方不统一,此为 0 和 1 的枚举
    int i,j,tem,fnum;
```

```
    if (k<=0){ upmin(ans,num); return ;}     //边界，统计结果,以所有变元有值为边
界
    if (flag[pos]){                                        //可由这一行算出
        x[pos]=a[k][var+1];
        for (j=pos+1; j<=var; j++)  if (a[k][j]) x[pos]^=x[j];     //当前行的计算
        if (x[pos]) dfs(k-1,pos-1,num+1,var);
        else dfs(k-1,pos-1,num,var);
        return;
    }
    x[pos]=0; dfs(k,pos-1,num,var);                //枚举 0,1
    x[pos]=1; dfs(k,pos-1,num+1,var);
}

int gauss(int equ,int var){                  //总共 equ 个等式，var 个变量，第 var+1 为
等号右侧的值
    int i,j,k,maxr,col;
    lld tem,g,ta,tb;
    for (k=col=1; k<=equ && col<=var; k++,col++){
        if (a[k][col]==0){
            for (maxr=k,i=k+1; i<=equ; i++)      if (a[i][col]) {maxr=i; break;}
            if (maxr==k) {k--; continue;}
            for (i=col; i<=var+1; i++)    swap(a[k][i],a[maxr][i]);
        }
        for (i=k+1; i<=equ; i++)
            if (a[i][col]){
                g=gcd(a[k][col],a[i][col]); ta=a[k][col]/g; tb=a[i][col]/g;
                for (j=col; j<=var+1; j++) a[i][j]=((a[i][j]*ta-a[k][j]*tb)%7+7)%7;
            }
    }
    for (i=k; i<=equ; i++) if (a[i][var+1]) return -1;          //无解的情况
    if (k-1<n){                                                  //多解，需要枚举自
由变元的时候
        mem(flag,0);                     //把每个等式中的第一个数标记为定的元素
        for (i=1; i<k; i++)  for (j=i; j<=var; j++)    if (a[i][j]){ flag[j]=1; break;}
        dfs(k-1,var,0,var);              //进行非主元素值的情况枚举
        return 1;
    }
    for (i=k-1; i>0; i--){                                       //唯一解，求解每个确
定的答案
        tem=a[i][var+1];
        for (j=i+1; j<=var; j++) if (a[i][j]) tem=((tem-a[i][j]*x[j])%7+7)%7;
        while (tem%a[i][i]!=0) tem+=7; x[i]=(tem/a[i][i])%7;
    }
    return 0;
```

```
}
```

3. 莫比乌斯反演

```
//产生上限为 up 的 mobious 系数
int prim[maxn],pnum,flag[maxn],mu[maxn];          //flag 标记最小素因子
void getMu(int up){
        int i,j,tem;
        mem(flag,0); pnum=0; mu[1]=1;
        for (i=2; i<=up; i++){
                if (flag[i]==0){ prim[++pnum]=i; flag[i]=i;}
                for (j=1; j<=pnum && i*prim[j]<=up; j++){
                        flag[i*prim[j]]=prim[j];
                        if (i%prim[j]==0) break;
                }
                if (i/flag[i]%flag[i]==0) mu[i]=0;    //出现偶数次则值为 0
                else mu[i]=-1*mu[i/flag[i]];
        }
}

//计算 f(n)的值（这里是以 n 为边长和的三边 gcd 为 1 的三角形）有多少个
lld cau(lld n){
        lld i,j,tem,ans=0;
        if (n<3) return ans;
        for (i=1; i*i<=n; i++){
                if (n%i) continue;
                ans=(ans+mu[i]*g[n/i])%mod;
                if (i*i!=n) ans=(ans+mu[n/i]*g[i])%mod;
        }
        return ans;
}
```

4. 波利亚计数法 p 色放 n 环上

```
lld prim[maxn],flag[maxn],pnum;
lld p,n,mod,ans;

void createPrim(){...}
lld eular(lld n){...}                                   //求单个数的 eular 函数
lld PowMod(lld a,lld b,lld p){...}                      //计算(a^b)%p;

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

```
    p=n; ans=0;
    for (i=1; i*i<=n; i++){
        if (i*i==n) ans=(ans+eular(i)*PowMod(p,i-1,mod))%mod;
        else if (n%i==0)
            ans=(ans+eular(i)*PowMod(p,n/i-1,mod)%mod+eular(n/i)*PowMod(p,i-
1,mod)%mod)%mod;
    }
    printf("%d\n",ans);
```

5. 行列式

```
//求 det(A) mod p
lld mat[maxn][maxn];

lld det(lld a[maxn][maxn], int n, lld p) {
    lld i, j, k, t, ans = 1;
    for (i = 0; i < n; ++i) for (j = 0; j < n; ++j) a[i][j] %= p;
    for (i = 0; i < n; ++i) {
        for (j = i + 1; j < n; ++j) {
            for ( ; a[j][i]; ans = -ans) {
                for (t = a[i][i] / a[j][i], k = 0; k < n; ++k) {
                    a[i][k] = (a[i][k] - a[j][k] * t) % p;
                    swap(a[i][k], a[j][k]);
                }
            }
        }
        if (!a[i][i]) return 0;
        ans = ans * a[i][i] % p;
    }
    return (ans % p + p) % p;
}
```

6. 普通生成函数

```
#define maxn 251000
int num[110],val[110];
int f[maxn],t[maxn];
int n,up;

    while (scanf("%d",&n),n>0){
        up=0;
        for (i=1; i<=n; i++){scanf("%d%d",&val[i],&num[i]); up+=num[i]*val[i];}
```

```
        mem(f,0); f[0]=1;
        for (i=1; i<=n; i++){
            for (j=0; j<=up; j++) t[j]=0;
            for (j=0; j<=up; j++)
                for (int k=0; k<=num[i]*val[i]; k+=val[i])
                    if (j+k<=up && f[j]) t[j+k]=1;
            for (j=0; j<=up; j++) f[j]=t[j];
        }
        for (i=(up+1)/2; f[i]==0; i++);
        printf("%d %d\n",i,up-i);
```

7. FFT

```
const cp I=cp(0,1);
cp A[maxn],B[maxn],C[maxn];                    //A,B 用于计算，C 用于临时存储
void fft(cp A[], cp B[], int len, bool inv){//len 是 2 的幂次，从 0 开始表示长度,对 A 进行操作
    double arg = PI;                           //inv=false 进行 DFT，=true 进行 IDFT
    for (int n=len/2; n>=1; n/=2){
        cp pow = cp(1, 0);
        cp mul = exp((inv ? 1.0 : -1.0)*I*arg);
        for (int j=0; j<len; j+=n){
            for (int i=0; i<n; ++i)
                B[i+j] = A[i+j*2%len] + A[i+n+j*2%len] * pow;
            pow = pow * mul;
        }
        for (int i=0; i<len; ++i) A[i] = B[i];
        arg /= 2.0;
    }
}

void mul(cp a[], cp b[],int len){              //a 数组与 b 数组相乘,结果放入 ans 数组里
    mem(ans,0);                                // 注 意 len 的 长 度 需 要>=2*lena && >=2*lenb
    fft(a,C,len,false); fft(b,C,len,false);
    for (int i=0; i<len; i++) a[i]=a[i]*b[i];     //一定要从 0 开始，到 len-1 结束
    fft(a,C,len,true);
    for (int i=0; i<len; i++) ans[i]=(int)(a[i].real()/len+0.5);
}

//初始化    mem(A,0); A[tem]=1.0;
```

8. 寻找因子和素性测试

```
#define Times 10            //miller_rabin 测试次数
map<lld,lld>M;              //n 的所有素因子和其对应的个数
vector<pll>V;               //n 的所有素因子和其对应的个数
vector<lld>fac;             //n 的所有的约数

//random 产生 0 至 n-1 之间的随机数
lld random(lld n){ return ((double)rand()/RAND_MAX*n+0.49);}
lld gcd(lld a,lld b){ return !b ? a : gcd(b,a%b);}

//a*b%mod 的快速加的形式
lld fmul(lld a,lld b,lld mod){
    lld ans=0;
    for (; b; b/=2,a=(a+a)%mod)
        if (b&1) ans=(ans+a)%mod;
    return ans;
}

lld fpow(lld a,lld b,lld mod){
    lld ans=1;
    for (; b; b/=2,a=fmul(a,a,mod))
        if (b&1) ans=fmul(ans,a,mod);
    return ans;
}

bool witness(lld a,lld n){
    lld d=n-1;
    while (!(d&1)) d>>=1;
    lld t=fpow(a,d,n);
    while (d!=n-1 && t!=1 && t!=n-1) t=fmul(t,t,n),d*=2;
    return t==n-1 || (d&1);
}

//miller_rabin 测试，素数返回 true
bool miller_rabin(lld n){
    if (n==2) return true;
    if (n<2 || !(n&1)) return false;
    for (int i=1; i<=Times; i++){
        lld a=random(n-2)+1;
        if (!witness(a,n)) return false;
    }
    return true;
```

```
}

//找到 n 的一个因子,1<d<n
lld pollard_rho(lld n,lld c){
	lld x,y,d,i=1,k=2;
	x=random(n-2)+1; y=x;
	while (1){
		i++; x=(fmul(x,x,n)+c)%n; d=gcd(y-x,n);
		if (1<d && d<n) return d;
		if (y==x) return n;
		if (i==k) y=x,k*=2;
	}
}

//求出 n 的所有的素因子和其个数
void find(lld n,lld c){
	if (n==1) return ;
	if (miller_rabin(n)){ M[n]++; return;}
	lld p=n;
	while (p>=n) p=pollard_rho(p,c--);
	find(p,c); find(n/p,c);
}

//dfs 得到 n 的所有的约数
void dfs(lld dep,lld val){
	if (dep==V.size()){ fac.pb(val); return;}
	lld i,j=1,base=V[dep].first,up=V[dep].second;
	dfs(dep+1,val);
	for (i=1; i<=up; i++){
		j*=base;
		dfs(dep+1,val*j);
	}
}

//获得 n 的所有质因数和约数
void getDivider(lld n){
	M.clear(); V.clear(); fac.clear();
	find(n,12345);
	foreach(it,M) V.pb(mp(it->first,it->second));
	dfs(0,1);
	sort(fac.begin(),fac.end());      //对所有的约数进行排序
}
```

9. 小步大步攻击

```
struct node{
	lld val,sum,next;			//sum 记录 B 的多少次方，val 为其 B 的次方模 p
}hash[maxn];

lld adj[maxn],tot;
lld B,C,L,A,mod;			//B^x == C (%mod)的最小的 x;
lld flag,ans,num;

lld PowMod(lld a,lld b,lld p){}  //a^b%p,爆的话，用快速加
lld gcd(lld a,lld b){}

lld init(){
	lld i,j,g,tem,p;
	if (C>=mod) return -1;			//C 与 mod 之间的关系，依题意而定
	num=0; ans=-1;
	for (j=1,i=0; i<65 && j!=C; i++,j=(j*B)%mod);//枚举前 65 个
	if (i<65) return i;
	for (A=1; (g=gcd(B,mod))!=1; ){	//把 B 的 num 次方提出来当系数与 mod 进行约分
		if (C%g) return -1;
		A=B/g*A%mod; mod/=g; C/=g;
		num++;
	}
	mem(adj,-1);
	lld	d=(lld)sqrt(mod*1.0);
	for (tem=C,tot=i=0; i<d; i++,tem=(tem*B)%mod){//右端以 C*B^i 建立元素，放入 hash 表里
		p=tem%N;
		hash[tot].val=tem; hash[tot].sum=i; hash[tot].next=adj[p]; adj[p]=tot++;
	}
	return 0;
}

int main(){
	lld i,j,r,tem,x,y,p,xm;
	while (scanf("%I64d%I64d%I64d",&B,&mod,&C)!=EOF){
		flag=init();
		if (flag==-1)
			printf("Orz,I can’t find D!\n");
		else if (flag>0)
```

```
            printf("%I64d\n",flag);//此地的 flag 即为前面的 0..64 的枚举结果
        else {
            xm=PowMod(B,d,mod);                                    //提前计算，优
化速度
            for (x=(A*xm)%mod,i=1; i<=d+1 && ans==-1; i++,x=(x*xm)%mod)
                for (p=x%N,j=adj[p]; j!=-1; j=hash[j].next)   //因为从大到小插入，
故找到即是答案
                    if (hash[j].val==x){
                        ans=num+i*d-hash[j].sum;
                        break;
                    }
            if (ans!=-1) printf("%I64d\n",ans);
            else printf("Orz,I can't find D!\n");
        }
    }
    return 0;
}
```

10. 中国剩余定理

```
//a 为需要模的数，r 为其对应余数
struct nCRT{
    lld a,r;
}list[maxn];

lld CRT(lld n){
    lld i,j,tem,g,xi,yi,a1,r1,a2,r2,x,max;
    r1=0; a1=1;
    for (i=1; i<=n; i++){
        a2=list[i].a; r2=list[i].r;
        e_gcd(a1,a2,g,xi,yi);
        if ((r2-r1)%g) return -1;
        max=a1/g*a2;
        x=(((r2-r1)/g*xi%a2*a1+r1)%max+max)%max;
        r1=x; a1=max;
    }
    return x;           //return x==0 ? max : x;
    //次行注意 x 为 0 时是否为符合题意的解，否则进行注释
}
```

11. 基础数学

```
//求整数 x 和 y，使得 ax+by=d，且|x|+|y|最小。其中 d=gcd(a,b)
void e_gcd(lld a,lld b,lld &d,lld &x,lld &y){
	if (!b){d=a; x=1; y=0;}
	else {e_gcd(b,a%b,d,y,x); y-=x*(a/b);}
}

//求快速幂 a^b%mod
lld fpow(lld a,lld b){
	lld ans=1;
	for (; b; b/=2,a=(a*a)%mod)
		if (b&1) ans=(ans*a)%mod;
	return ans;
}

//求 n 关于 p 的逆元，p 为素数
lld cau(lld n,lld p){
	return n==1 ? 1 : (p-p/n*cau(p%n,p)%p)%p;
}

//lucas 定理，需要自己补充 C(n,m,p)的结果，可能是打表或用费马小定理
lld lucas(lld n,lld m,lld p){
	if (m==0) return 1;
	return lucas(n/p,m/p,p)*C(n%p,m%p,p)%p;
}

//欧拉函数打表，范围<=up
lld prim[maxn],flag[maxn],phi[maxn],pnum;
void getPHI(lld up){
	mem(flag,0); pnum=0; phi[1]=1;
	for (lld i=2; i<=up; i++){
		if (!flag[i]) prim[++pnum]=i,phi[i]=i-1;
		for (lld j=1; j<=pnum && i*prim[j]<=up; j++){
			flag[i*prim[j]]=1;
			if (i%prim[j]!=0) phi[i*prim[j]]=phi[i]*(prim[j]-1);
			else { phi[i*prim[j]]=phi[i]*prim[j]; break;}
		}
	}
}

//求单个数的欧拉函数
lld eular(lld n){
```

```
	lld i,j,tem,ans=1;
	for (i=1; prim[i]*prim[i]<=n; i++) if (n%prim[i]==0){
		ans*=prim[i]-1;
		for (n/=prim[i]; n%prim[i]==0; n/=prim[i]) ans*=prim[i];
	}
	if (n>1) ans*=n-1;
	return ans;
}

//整数开平方，下区整
lld lldSqr(lld n) {
	lld x, y=n;
	do { x=y; y=(x+n/x) >> 1; }while (y < x);
	return x;
}
```

12. Simpson 数值积分

```
double F(double x){}

double simpson(double aa , double bb) {
	double cc = aa + (bb-aa)/2;
	return (F(aa)+4*F(cc)+F(bb))*(bb-aa)/6;
}

double asr(double aa, double bb , double epx , double A) {
	double cc = aa+(bb-aa)/2;
	double L = simpson(aa,cc),R = simpson(cc,bb);
	if(fabs(L+R-A) <=epx) return L+R+(L+R-A);
	return asr(aa,cc,epx/2,L)+asr(cc,bb,epx/2,R);
}

//simpson 积分，aa 到 bb，精度不会超过 eps
double asr(double aa , double bb) {
	return asr(aa,bb,eps,simpson(aa,bb));
}
```

13. 素数筛选与欧拉函数

```
#define MAX 100010

int a[MAX], p[MAX], phi[MAX];
```

```
int prime() {
        int i, j, n = 0;

        phi[1] = 1;
        for (i = 2; i < MAX; ++i) {
                if (!a[i]) {
                        a[i] = p[n++] = i; phi[i] = i - 1;
                }
                for (j = 0; j < n && i * p[j] < MAX; ++j) {
                        a[i * p[j]] = p[j];
                        if (i % p[j]) phi[i * p[j]] = phi[i] * (p[j] - 1);
                        else {
                                phi[i * p[j]] = phi[i] * p[j];
                                break;
                        }
                }
        }

       return n;
}

int eula(int n) {
       int i, ret = 1;

       for (i = 0; p[i] * p[i] <= n; ++i) {
              if (!(n % p[i])) {
                     ret *= p[i] - 1;
                     for (n /= p[i]; !(n % p[i]); n /= p[i]) ret *= p[i];
              }
       }
       if (n > 1) ret *= n - 1;

       return ret;
}
```

14. QR 分解

```
#include<iostream>
#include<fstream>
#include<iomanip>
using namespace std;
#include<math.h>
```

```
#define N 3 //矩阵的维数
#define NUM 1000 //QR 分解次数

void QR(long double A[N][N],long double Q[N][N],long double R[N][N]); //QR 分解
void Multiplicate(long double A[N][N],long double R[N][N],long double Q[N][N]); //迭代，获得下一次矩阵 A=QR

long double A[N][N];
long double Q[N][N]={0};
long double R[N][N]={0};
bool domain()
{
	int i,j;
	double a, b, c, d, e, f;
	double ret = -1, cur;

	if(scanf("%lf%lf%lf%lf%lf%lf",&a,&b,&c,&d,&e,&f)==EOF)
		return false;

	A[0][0] = a;
	A[1][1] = b;
	A[2][2] = c;
	A[0][1] = A[1][0] = f / 2.0;
	A[0][2] = A[2][0] = e / 2.0;
	A[1][2] = A[2][1] = d / 2.0;
	memset(Q, 0, sizeof(Q));
	memset(R, 0, sizeof(R));
	for(i=1;i<=NUM;i++)
	{
		QR(A,Q,R);
		Multiplicate(A,R,Q);
	}
	for(i=0;i<N;i++) //输出特征值
	{
		if(R[i][i] > 0)
		{
			cur = 1.0 / sqrt(R[i][i]);
			if(ret < 0 || cur <= ret)
				ret = cur;
		}
	}
	if(ret < 0)
		ret = *(int*)NULL;
```

```
	printf("%.6lf\n",ret);

	return true;
}

void QR(long double A[N][N],long double Q[N][N],long double R[N][N])
{
	int i,j,k,m;
	long double temp;
	long double a[N],b[N];
	for(j=0;j<N;j++)
	{
		for(i=0;i<N;i++)
		{
			a[i]=A[i][j];
			b[i]=A[i][j];
		}
		for(k=0;k<j;k++)
		{
			R[k][j]=0;
			for(m=0;m<N;m++)
				R[k][j]+=a[m]*Q[m][k];
			for(m=0;m<N;m++)
				b[m]-=R[k][j]*Q[m][k];
		}
		temp=0;
		for(i=0;i<N;i++)
			temp+=b[i]*b[i];
		R[j][j]=sqrt(temp);
		for(i=0;i<N;i++)
			Q[i][j]=b[i]/sqrt(temp);
	}
}

void Multiplicate(long double A[N][N],long double R[N][N],long double Q[N][N])
{
	int i,j,k;
	long double temp;
	for(i=0;i<N;i++)
		for(j=0;j<N;j++)
		{
			temp=0;
			for(k=0;k<N;k++)
				temp+=R[i][k]*Q[k][j];
```

```
            A[i][j]=temp;
        }
}
```

15. 牛顿科特斯表

科特斯系数表

n	$C_i^{(n)}$								
1	$\frac{1}{2}$	$\frac{1}{2}$							
2	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$						
3	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$					
4	$\frac{7}{90}$	$\frac{16}{45}$	$\frac{2}{15}$	$\frac{16}{45}$	$\frac{7}{90}$				
5	$\frac{19}{288}$	$\frac{25}{96}$	$\frac{25}{144}$	$\frac{25}{144}$	$\frac{25}{96}$	$\frac{19}{288}$			
6	$\frac{41}{840}$	$\frac{9}{35}$	$\frac{9}{280}$	$\frac{34}{105}$	$\frac{9}{280}$	$\frac{9}{35}$	$\frac{41}{840}$		
7	$\frac{751}{17280}$	$\frac{3577}{17280}$	$\frac{1323}{17280}$	$\frac{2989}{17280}$	$\frac{2989}{17280}$	$\frac{1323}{17280}$	$\frac{3577}{17280}$	$\frac{751}{17280}$	
8	$\frac{989}{28350}$	$\frac{5888}{28350}$	$\frac{-928}{28350}$	$\frac{10496}{28350}$	$\frac{-4540}{28350}$	$\frac{10496}{28350}$	$\frac{-928}{28350}$	$\frac{5888}{28350}$	$\frac{989}{28350}$

字符串

1. AC 自动机

```
const int maxn = 110;
const int nchar = 26;
struct AC {
    int next[maxn][nchar];
    int fail[maxn];
    int status[maxn];
    int cc;
    void init() {
        mem(next, 0);
        mem(fail, 0);
        mem(status, 0);
        cc = 1;
    }
```

```
int hash(char a) {
	return a - 'a';
}
void insert_trie(char *a, int c) {
	int i = 0;
	int p = 0;
	while(a[i]) {
		int index = hash(a[i]);
		if(next[p][index] == 0) {
			next[p][index] = cc++;
		}
		p = next[p][index];
		i++;
	}
	status[p]  |= 1 << c;
}

void acinit() {
	queue<int> q;
	while(!q.empty()) q.pop();
	for(int i = 0; i < nchar; i++) {
		if(next[0][i]) q.push(next[0][i]);
	}
	while(!q.empty()) {
		int a = q.front();
		status[a]  |= status[fail[a]];
		q.pop();
		for(int i = 0; i < nchar; i++) {
			if(next[a][i] == 0) continue;
			q.push(next[a][i]);
			int tp = fail[a];
			while(tp && next[tp][i] == 0) {
				tp = fail[tp];
			}
			fail[next[a][i]] = next[tp][i];
		}
	}
}

int solve() {
	for(int i = 0; i <= n; i++) {
		for(int j = 0; j < cc; j++) {
			for(int k = 0; k < (1 << m); k++) {
				dp[i][j][k] = 0;
```

```
                }
            }
        }
        dp[0][0][0] = 1;
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < cc; j++) {
                for(int k = 0; k < (1 << m); k++) {
                    if(dp[i][j][k]) {
                        for(int l = 0; l < 26; l++) {
                            int tp = j;
                            while(tp && next[tp][l] == 0) tp = fail[tp];
                            int nj = next[tp][l];
                            int nk = k | status[nj];
                            dp[i + 1][nj][nk] += dp[i][j][k];
                            dp[i + 1][nj][nk] %= mod;
                        }
                    }
                }
            }
        }
        int ans = 0;
        for(int i = 0; i < (1 << m); i++) {
            if(count(i) < kk) continue;
            for(int j = 0; j < cc; j++) {
                ans += dp[n][j][i];
                ans %= mod;
            }
        }
        return ans;
    }
} ac;
```

JAVA

1. Bigdecimal

Java 中的 Bigdecimal 类型运算

双精度浮点型变量 double 可以处理 16 位有效数。在实际应用中，需要对更大或者更小的数进行运算和处理。Java 在 java.math 包中提 供的 API 类 BigDecimal，用来对超过 16 位有效位的数进行精确的运算。表 5.7 中列出了 BigDecimal 类的

主要构造器和方法。

构造器　描 述
BigDecimal(int)创建一个具有参数所指定整数值的对象。
BigDecimal(double)创建一个具有参数所指定双精度值的对象。
BigDecimal(long)创建一个具有参数所指定长整数值的对象。
BigDecimal(String)创建一个具有参数所指定以字符串表示的数值的对象。

方 法描 述
add(BigDecimal)BigDecimal 对象中的值相加，然后返回这个对象。
subtract(BigDecimal)BigDecimal 对象中的值相减，然后返回这个对象。
multiply(BigDecimal)BigDecimal 对象中的值相乘，然后返回这个对象。
divide(BigDecimal)BigDecimal 对象中的值相除，然后返回这个对象。
toString()将 BigDecimal 对象的数值转换成字符串。
doubleValue()将 BigDecimal 对象中的值以双精度数返回。
floatValue()将 BigDecimal 对象中的值以单精度数返回。
longValue()将 BigDecimal 对象中的值以长整数返回。
intValue()将 BigDecimal 对象中的值以整数返回。
注意，由于一般数值类型，例如 double，不能准确地代表 16 位有效数以上的数字，在使用 BigDecimal 时，应用 BigDecimal(String)构造器创建对象才有意义。另外，BigDecimal 所创建的是对象，我们不能使用传统的+、-、*、/等算术运算 符直接对其对象进行数学运算，而必须调用其相对应的方法。方法中的参数也必须是 BigDecimal 的对象。

eg:
两个 BigDecimal 类型的数据相乘：

```
BigDecimal   a   =   new   BigDecimal(15124);
             BigDecimal   b   =   new   BigDecimal(15124);
             BigDecimal   c   =   a.multiply(b);
```

java BigDecimal 比较大小

```
BigDecimal a=BigDecimal.valueOf(1.0);
BigDecimal b=BigDecimal.valueOf(1.000);
 if(a.compareTo(b)==0)  结果是 true
 a.compareTo(b)  返回值     -1  小于    0  等于     1  大于
```

```
public static void main(String[] argc) {
            //下面都以保留 2 位小数为例

            //ROUND_UP
            //只要第 2 位后面存在大于 0 的小数，则第 2 位就+1
```

```
System.out.println(round(12.3401,2,BigDecimal.ROUND_UP));//12.35
System.out.println(round(-12.3401,2,BigDecimal.ROUND_UP));//-12.35
//ROUND_DOWN
//与 ROUND_UP 相反
//直接舍弃第 2 位后面的所有小数
System.out.println(round(12.349,2,BigDecimal.ROUND_DOWN));//12.34
System.out.println(round(-12.349,2,BigDecimal.ROUND_DOWN));//-12.34
//ROUND_CEILING
//如果数字>0 则和 ROUND_UP 作用一样
//如果数字<0 则和 ROUND_DOWN 作用一样
System.out.println(round(12.3401,2,BigDecimal.ROUND_CEILING));//12.35
System.out.println(round(-12.349,2,BigDecimal.ROUND_CEILING));//-12.34
//ROUND_FLOOR
//如果数字>0 则和 ROUND_DOWN 作用一样
//如果数字<0 则和 ROUND_UP 作用一样
System.out.println(round(12.349,2,BigDecimal.ROUND_FLOOR));//12.34
System.out.println(round(-12.3401,2,BigDecimal.ROUND_FLOOR));//-12.35
//ROUND_HALF_UP [这种方法最常用]
//如果第 3 位数字>=5,则第 2 位数字+1
//备注:只看第 3 位数字的值,不会考虑第 3 位之后的小数的
System.out.println(round(12.345,2,BigDecimal.ROUND_HALF_UP));//12.35

System.out.println(round(12.3449,2,BigDecimal.ROUND_HALF_UP));//12.34
System.out.println(round(-12.345,2,BigDecimal.ROUND_HALF_UP));//-
12.35
System.out.println(round(-12.3449,2,BigDecimal.ROUND_HALF_UP));//-
12.34
//ROUND_HALF_DOWN
//如果第 3 位数字>=5,则做 ROUND_UP
//如果第 3 位数字<5,则做 ROUND_DOWN

System.out.println(round(12.345,2,BigDecimal.ROUND_HALF_DOWN));//12.35

System.out.println(round(12.3449,2,BigDecimal.ROUND_HALF_DOWN));//12.34
System.out.println(round(-12.345,2,BigDecimal.ROUND_HALF_DOWN));//-
12.35
System.out.println(round(-
12.3449,2,BigDecimal.ROUND_HALF_DOWN));//-12.34
//ROUND_HALF_EVEN
//如果第 3 位是偶数,则做 ROUND_HALF_DOWN
//如果第 3 位是奇数,则做 ROUND_HALF_UP

System.out.println(round(12.346,2,BigDecimal.ROUND_HALF_EVEN));//12.35
```

```
System.out.println(round(12.345,2,BigDecimal.ROUND_HALF_EVEN));//12.35
        }
}
```

2. BigInteger

JAVA 之 BigInteger(转)【转】【很好用啊】
用 Java 来处理高精度问题，相信对很多 ACMer 来说都是一件很 happy 的事，简单易懂。用 Java 刷了一些题，感觉 Java 还不错，在处理高精度和进制转换中，调用库函数的来处理。下面是写的一些 Java 中一些基本的函数的及其……
头文件：

```
 import java.io.*;
 import java.util.*;
 import java.math.*;
```

读入： Scanner cin = Scanner (System.in);

```
 while(cin.hasNext())//等价于!=EOF
 n=cin.nextInt();//读入一个 int 型的数
 n=cin.nextBigInteger();//读入一个大整数
```

输出： System.out.print(n);//打印 n

```
 System.out.println();//换行
 System.out.printf("%d\n",n);//也可以类似 c++里的输出方式
```

定义： int i,j,k,a[];

```
 a = new int[100];
 BigInteger n,m;
 BigDecimal n;
 String s;
```

数据类型：
数据类型 类型名 位长 取值范围 默认值
布尔型 boolean 1 true,false false
字节型 byte 8 -128-127 0
字符型 char 16 ‘\u000’-\uffff ‘\u0000’
短整型 short 16 -32768-32767 0
整型 int 32 -2147483648,2147483647 0
长整型 long 64 -9.22E18,9.22E18 0
浮点型 float 32 1.4E-45-3.4028E+38 0.0
双精度型 double 64 4.9E-324,1.7977E+308 0.0

这里特别要提出出的两种类型：
BigInteger 任意大的整数，原则上是，只要你的计算机的内存足够大，可以有无

限位的
BigInteger 任意大的实数，可以处理小数精度问题。
BigInteger 中一些常见的函数：
A=BigInteger.ONE
B=BigInteger.TEN
C=BigInteger.ZERO

一些常见的数的赋初值。将 int 型的数赋值给 BigInteger，BigInteger.valueOf(k);
基本的函数：
valueOf:赋初值
add:+ a.add(b);
subtract:-
multiply:*
divide:/

remainder：this % val
divideAndRemainder：a[0]=this / val; a[1]=this % val
pow：a.pow(b)=a^b
gcd,abs:公约数，绝对值
negate：取负数
signum：符号函数
mod：a.mod(b)=a%b;

shiftLeft:左移，this << n ，this*2^n;
shiftRight:右移，this >> n，this/2^n;
and:等同于 c++的&&,且；
or：||，或;
xor:异或，BigInteger xor(BigInteger val),this^val
not:!,非；

bitLength：返回该数的最小二进制补码表示的位的个数，即 *不包括* 符号位 (ceil(log2(this <0 ? -this : this + 1)))。对正数来说，这等价于普通二进制表示的位的个数。
bitCount：返回该数的二进制补码表示中不包扩符号位在内的位的个数。该方法在 BigIntegers 之上实现位向量风格的集合时很有用。
isProbablePrime：如果该 BigInteger 可能是素数，则返回 true ；如果它很明确是一个合数，则返回 false 。 参数 certainty 是对调用者愿意忍受的不确定性的度量：如果该数是素数的概率超过了 1 - 1/2**certainty 方法，则该方法返回 true 。执行时间正比于参数确定性的值。
compareTo：根据该数值是小于、等于、或大于 val 返回 -1、0 或 1；
equals：判断两数是否相等，也可以用 compareTo 来代替；
min，max：取两个数的较小、大者；
intValue，longValue，floatValue，doublue：把该数转换为该类型的数的值。

今天参考课本写了一个关于二进制与十进制转换的程序，程序算法不难，但写完后测试发现不论是二转十还是十转二，对于大于 21 亿即超过整数范围的数不能很好的转换。都会变成 0.
参考书籍发现使用使用 BigInteger 可以解决这个问题。
于是查找了下 JDK,然后测试几次终于写成功了！
使用心得如下：

1，BigInteger 属于 java.math.BigInteger,因此在每次使用前都要 import 这个类。偶开始就忘记 import 了，于是总提示找不到提示符。
2，其构造方法有很多，但现在偶用到的有： BigInteger(String val)
将 BigInteger 的十进制字符串表示形式转换为 BigInteger。
BigInteger(String val, int radix)
将指定基数的 BigInteger 的字符串表示形式转换为 BigInteger。
如要将 int 型的 2 转换为 BigInteger 型，要写为 BigInteger two=new BigInteger("2"); //注意 2 双引号不能省略
3，BigInteger 类模拟了所有的 int 型数学操作，如 add()==“+”,divide()==“-”等，但注意其内容进行数学运算时不能直接使用数学运算符进行运算，必须使用其内部方法。而且其操作数也必须为 BigInteger 型。
如：two.add(2)就是一种错误的操作，因为 2 没有变为 BigInteger 型。
4，当要把计算结果输出时应该使用.toString 方法将其转换为 10 进制的字符串，详细说明如下：
String toString()
返回此 BigInteger 的十进制字符串表示形式。
输出方法：System.out.print(two.toString());
5,另外说明三个个用到的函数。 BigInteger remainder(BigInteger val)
返回其值为 (this % val) 的 BigInteger。
BigInteger negate()
返回其值是 (-this) 的 BigInteger。
int compareTo(BigInteger val)
将此 BigInteger 与指定的 BigInteger 进行比较。
remainder 用来求余数。
negate 将操作数变为相反数。
compare 的详解如下：

compareTo
public int compareTo(BigInteger val)将此 BigInteger 与指定的 BigInteger 进行比较。对于针对六个布尔比较运算符 (<, ==, >, >=, !=, <=) 中的每一个运算符的各个方法，优先提供此方法。执行这些比较的建议语句是：(x.compareTo(y) <op> 0)，其中 <op> 是六个比较运算符之一。
指定者：
接口 Comparable<BigInteger> 中的 compareTo
参数：
val - 将此 BigInteger 与之比较的 BigInteger。

返回：

将 BigInteger 的数转为 2 进制：

```
public class TestChange {
public static void main(String[] args) {
System.out.println(change("3",10,2));
}
//num  要转换的数  from 源数的进制  to 要转换成的进制
private static String change(String num,int from, int to){
return new java.math.BigInteger(num, from).toString(to);
}
}
```