ACM 模板

JPVision Fighting!

To be or not to be, that is a question.

alpc48
2008-10-5

ACM Fighting!

ACM Fighting! *2*
1.计算几何 *5*
1.1 注意 **5**
1.2 几何公式 **6**
1.3 多边形 **8**
1.4 多边形切割 **11**
1.5 浮点函数 **12**
1.6 面积 **18**
1.7 球面 **18**
1.8 三角形 **19**
1.9 三维几何 **22**
1.10 凸包 **30**
1.11 网格 **32**
1.12 圆 **33**
1.13 矢量运算求几何模板 **35**
1.14 结构体表示几何图形 **47**
1.15 四城部分几何模板 **52**
1.16 一些代码 **54**
1.16.1 最小圆覆盖_zju1450 54
1.16.2 直线旋转_两凸包的最短距离(poj3608) 58
1.16.3 扇形的重心 62
1.16.4 根据经度纬度求球面距离 63
1.16.5 多边形的重心 64
1.16.6 存不存在一个平面把两堆点分开(poj3643) 66
1.16.7 pku_3335_判断多边形的核是否存在 67
1.16.8 pku_2600_二分+圆的参数方程 74
1.16.9 pku_1151_矩形相交的面积 76
1.16.10 pku_1118_共线最多的点的个数 78
1.16.11 pku2826_线段围成的区域可储水量 80
1.16.12 Pick公式 84
1.16.13 N点中三个点组成三角形面积最大 86
1.16.14 直线关于圆的反射 89
1.16.15 pku2002_3432_N个点最多组成多少个正方形(hao) 94

1.16.16 pku1981_单位圆覆盖最多点(poj1981)CircleandPoints 97
1.16.17 pku3668_GameofLine_N个点最多确定多少互不平行的直线(poj3668) 99
1.16.18 求凸多边形直径 100
2.组合 *102*
2.1 组合公式 **102**
2.2 排列组合生成 **102**
2.3 生成gray码 **104**
2.4 置换(polya) **104**
2.5 字典序全排列 **105**
2.6 字典序组合 **105**
2.7 一些原理及其例子 **106**
3.数论 *108*
3.1 阶乘最后非 0 位 **108**
3.2 模线性方程组 **108**
3.3 素数 **110**
3.4 欧拉函数 **114**
3.6 高精度 **116**
3.6.1 平方根 116
3.6.2 高精度乘幂 117
3.7 高斯消元回代法 **123**
3.8 数值计算 **124**
3.8.1 定积分计算 124
3.8.2 多项式求根(牛顿法) 125
3.8.3 周期性方程(追赶法) 127
4.排序 *128*
4.1 快速选择算法 **128**
4.2 归并排序+逆序数的求取 **129**
5.字符串 *130*
5.1 KMP应用 **130**
5.2 后缀数组 **131**
5.3 中缀表达式转后缀表达式 **134**
5.4 Firefighters 表达式求值 **135**
6.博弈 *139*
6.1 博弈的AB剪枝 **139**

6.1.1 取石子 139
6.2 博弈 SG函数 局势分割 141
7.数据结构 142
7.1 TRIE 142
7.2 线段树 147
7.3 并查集 151
7.4 树状数组 152
7.5 点树 154
7.6 STL 156
7.7 离散化 157
8.图论 158
8.0 2-SAT 158
8.2 寻找Euler回路 163
8.3 拓扑排序 163
8.4 差分约束系统 164
8.5 笛卡尔树 165
8.6 LCA和RMQ 167
8.7 割和桥 171
8.8 最小生成树(kruskal) 172
8.9 最短路径 173
8.10 最大网络流 175
8.11 最小费用流 180
8.12 最大团问题 182
8.13 二分图匹配 184
8.14 带权的最优二分图匹配 184
9.搜索算法概略 187
9.1 迭代深搜+IDA* 187
9.2 分之界限法（深搜） 189
9.3 A* 8 数码问题(pascal) 192
9.4 优先队列广搜 194
10.应用 197
10.1 Joseph问题 197

10.2 N皇后构造解 197
10.3 布尔母函数 198
10.4 第k元素 199
10.5 幻方构造 199
10.6 模式匹配(kmp) 201
10.7 逆序对数 201
10.8 字符串最小表示 202
10.9 最长公共单调子序列 202
10.10 最长子序列 204
10.11 最大子串匹配 204
10.12 最大子段和 205
10.13 最大子阵和 206
11.其它 207
11.1 大数(只能处理正数) 207
11.2 分数 212
11.3 矩阵 214
11.4 线性方程组 216
11. 5 线性相关 218
11.6 日期 219
11.7 读入 220
11.8 函数 220

1.计算几何

1.1 注意

1. 注意舍入方式(0.5 的舍入方向);防止输出-0.
2. 几何题注意多测试不对称数据.
3. 整数几何注意 xmult 和 dmult 是否会出界;
   符点几何注意 eps 的使用.
4. 避免使用斜率;注意除数是否会为 0.
5. 公式一定要化简后再代入.
6. 判断同一个 2*PI 域内两角度差应该是
   abs(a1-a2)<beta||abs(a1-a2)>pi+pi-beta;

相等应该是
abs(a1-a2)<eps||abs(a1-a2)>pi+pi-eps;
7. 需要的话尽量使用 atan2,注意:atan2(0,0)=0,
atan2(1,0)=pi/2,atan2(-1,0)=-pi/2,atan2(0,1)=0,atan2(0,-1)=pi.
8. cross product = |u|*|v|*sin(a)
dot product = |u|*|v|*cos(a)
9. (P1-P0)x(P2-P0)结果的意义:
正: <P0,P1>在<P0,P2>顺时针(0,pi)内
负: <P0,P1>在<P0,P2>逆时针(0,pi)内
0 : <P0,P1>,<P0,P2>共线,夹角为 0 或 pi
10. 误差限缺省使用 1e-8!

1.2 几何公式

三角形:
1. 半周长 P=(a+b+c)/2
2. 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
3. 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
4. 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
5. 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
6. 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)
=4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)
=Ptan(A/2)tan(B/2)tan(C/2)
7. 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))

四边形:
D1,D2 为对角线,M 对角线中点连线,A 为对角线夹角
1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
2. S=D1D2sin(A)/2
(以下对圆的内接四边形)
3. ac+bd=D1D2
4. S=sqrt((P-a)(P-b)(P-c)(P-d)),P 为半周长

正 n 边形:
R 为外接圆半径,r 为内切圆半径
1. 中心角 A=2PI/n
2. 内角 C=(n-2)PI/n
3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))

圆:
1. 弧长 l=rA
2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)
3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2

4. 扇形面积 S1=rl/2=r^2A/2
5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2

棱柱:
1. 体积 V=Ah,A 为底面积,h 为高
2. 侧面积 S=lp,l 为棱长,p 为直截面周长
3. 全面积 T=S+2A

棱锥:
1. 体积 V=Ah/3,A 为底面积,h 为高
(以下对正棱锥)
2. 侧面积 S=lp/2,l 为斜高,p 为底面周长
3. 全面积 T=S+A

棱台:
1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2 为上下底面积,h 为高
(以下为正棱台)
2. 侧面积 S=(p1+p2)l/2,p1.p2 为上下底面周长,l 为斜高
3. 全面积 T=S+A1+A2

圆柱:
1. 侧面积 S=2PIrh
2. 全面积 T=2PIr(h+r)
3. 体积 V=PIr^2h

圆锥:
1. 母线 l=sqrt(h^2+r^2)
2. 侧面积 S=PIrl
3. 全面积 T=PIr(l+r)
4. 体积 V=PIr^2h/3

圆台:
1. 母线 l=sqrt(h^2+(r1-r2)^2)
2. 侧面积 S=PI(r1+r2)l
3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)
4. 体积 V=PI(r1^2+r2^2+r1r2)h/3

球:
1. 全面积 T=4PIr^2
2. 体积 V=4PIr^3/3

球台:
1. 侧面积 S=2PIrh
2. 全面积 T=PI(2rh+r1^2+r2^2)

3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6

球扇形:

1. 全面积 T=PIr(2h+r0),h 为球冠高,r0 为球冠底面半径
2. 体积 V=2PIr^2h/3

1.3 多边形

```
#include <stdlib.h>
#include <math.h>
#define MAXN 1000
#define offset 10000
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
#define _sign(x) ((x)>eps?1:((x)<-eps?2:0))
struct point{double x,y;};
struct line{point a,b;};

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);
}

//判定凸多边形,顶点按顺时针或逆时针给出,允许相邻边共线
int is_convex(int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
    return s[1]|s[2];
}

//判定凸多边形,顶点按顺时针或逆时针给出,不允许相邻边共线
int is_convex_v2(int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
    return s[0]&&s[1]|s[2];
}

//判点在凸多边形内或多边形边上,顶点按顺时针或逆时针给出
int inside_convex(point q,int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
    return s[1]|s[2];
```

```
}

//判点在凸多边形内,顶点按顺时针或逆时针给出,在多边形边上返回 0
int inside_convex_v2(point q,int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
    return s[0]&&s[1]|s[2];
}

//判点在任意多边形内,顶点按顺时针或逆时针给出
//on_edge 表示点在多边形边上时的返回值,offset 为多边形坐标上限
int inside_polygon(point q,int n,point* p,int on_edge=1){
    point q2;
    int i=0,count;
    while (i<n)
        for (count=i=0,q2.x=rand()+offset,q2.y=rand()+offset;i<n;i++)
            if
(zero(xmult(q,p[i],p[(i+1)%n]))&&(p[i].x-q.x)*(p[(i+1)%n].x-q.x)<eps&&(p[i].y-q.y)*(p[(i+1)%n].y-q.y)<e
ps)
                return on_edge;
            else if (zero(xmult(q,q2,p[i])))
                break;
            else                                                                    if
(xmult(q,p[i],q2)*xmult(q,p[(i+1)%n],q2)<-eps&&xmult(p[i],q,p[(i+1)%n])*xmult(p[i],q2,p[(i+1)%n])<-ep
s)
                count++;
    return count&1;
}

inline int opposite_side(point p1,point p2,point l1,point l2){
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}

inline 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;
}

//判线段在任意多边形内,顶点按顺时针或逆时针给出,与边界相交返回 1
int inside_polygon(point l1,point l2,int n,point* p){
    point t[MAXN],tt;
    int i,j,k=0;
    if (!inside_polygon(l1,n,p)||!inside_polygon(l2,n,p))
        return 0;
```

```
    for (i=0;i<n;i++)
        if (opposite_side(l1,l2,p[i],p[(i+1)%n])&&opposite_side(p[i],p[(i+1)%n],l1,l2))
            return 0;
        else if (dot_online_in(l1,p[i],p[(i+1)%n]))
            t[k++]=l1;
        else if (dot_online_in(l2,p[i],p[(i+1)%n]))
            t[k++]=l2;
        else if (dot_online_in(p[i],l1,l2))
            t[k++]=p[i];
    for (i=0;i<k;i++)
        for (j=i+1;j<k;j++){
            tt.x=(t[i].x+t[j].x)/2;
            tt.y=(t[i].y+t[j].y)/2;
            if (!inside_polygon(tt,n,p))
                return 0;
        }
    return 1;
}

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 barycenter(point a,point b,point c){
    line u,v;
    u.a.x=(a.x+b.x)/2;
    u.a.y=(a.y+b.y)/2;
    u.b=c;
    v.a.x=(a.x+c.x)/2;
    v.a.y=(a.y+c.y)/2;
    v.b=b;
    return intersection(u,v);
}

//多边形重心
point barycenter(int n,point* p){
    point ret,t;
    double t1=0,t2;
    int i;
```

```
    ret.x=ret.y=0;
    for (i=1;i<n-1;i++)
        if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){
            t=barycenter(p[0],p[i],p[i+1]);
            ret.x+=t.x*t2;
            ret.y+=t.y*t2;
            t1+=t2;
        }
    if (fabs(t1)>eps)
        ret.x/=t1,ret.y/=t1;
    return ret;
}
```

1.4 多边形切割

```
//多边形切割
//可用于半平面交
#define MAXN 100
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point{double x,y;};

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);
}

int same_side(point p1,point p2,point l1,point l2){
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)>eps;
}

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;
}

//将多边形沿 l1,l2 确定的直线切割在 side 侧切割,保证 l1,l2,side 不共线
void polygon_cut(int& n,point* p,point l1,point l2,point side){
    point pp[100];
    int m=0,i;
    for (i=0;i<n;i++){
```

```
            if (same_side(p[i],side,l1,l2))
                pp[m++]=p[i];
            if (!same_side(p[i],p[(i+1)%n],l1,l2)&&!(zero(xmult(p[i],l1,l2))&&zero(xmult(p[(i+1)%n],l1,l2))))
                pp[m++]=intersection(p[i],p[(i+1)%n],l1,l2);
        }
        for (n=i=0;i<m;i++)
            if (!i||!zero(pp[i].x-pp[i-1].x)||!zero(pp[i].y-pp[i-1].y))
                p[n++]=pp[i];
        if (zero(p[n-1].x-p[0].x)&&zero(p[n-1].y-p[0].y))
            n--;
        if (n<3)
            n=0;
}
```

1.5 浮点函数

```
//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;};
struct line{point a,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 distance(point p1,point p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double distance(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_online_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);
}

//点到直线距离
double disptoline(point p,line l){
	return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
double disptoline(point p,point l1,point l2){
	return fabs(xmult(p,l1,l2))/distance(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))/distance(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 distance(p,l.a)<distance(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 distance(p,l1)<distance(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 distance(p,l.a)<distance(p,l.b)?distance(p,l.a):distance(p,l.b);
    return fabs(xmult(p,l.a,l.b))/distance(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 distance(p,l1)<distance(p,l2)?distance(p,l1):distance(p,l2);
    return fabs(xmult(p,l1,l2))/distance(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;
}
```

//p 点关于直线 L 的对称点

```
ponit symmetricalPointofLine(point p, line L)
{
     point p2;
     double d;
```

```
    d = L.a * L.a + L.b * L.b;
    p2.x = (L.b * L.b * p.x - L.a * L.a * p.x -
            2 * L.a * L.b * p.y - 2 * L.a * L.c) / d;
    p2.y = (L.a * L.a * p.y - L.b * L.b * p.y -
            2 * L.a * L.b * p.x - 2 * L.b * L.c) / d;
    return p2;
}

//求两点的平分线
line bisector(point& a, point& b) {
    line ab, ans;    ab.set(a, b);
    double midx = (a.x + b.x)/2.0, midy = (a.y + b.y)/2.0;
    ans.a = -ab.b, ans.b = -ab.a, ans.c = -ab.b * midx + ab.a * midy;
    return ans;
}

// 已知入射线、镜面，求反射线。
// a1,b1,c1 为镜面直线方程(a1 x + b1 y + c1 = 0 ,下同)系数;
a2,b2,c2 为入射光直线方程系数;
a,b,c 为反射光直线方程系数.
// 光是有方向的，使用时注意：入射光向量:<-b2,a2>；反射光向量:<b,-a>.
// 不要忘记结果中可能会有"negative zeros"

void reflect(double a1,double b1,double c1,double a2,double b2,double c2,double &a,double &b,double &c)
{
    double n,m;
    double tpb,tpa;
    tpb=b1*b2+a1*a2;
    tpa=a2*b1-a1*b2;
    m=(tpb*b1+tpa*a1)/(b1*b1+a1*a1);
    n=(tpa*b1-tpb*a1)/(b1*b1+a1*a1);
    if(fabs(a1*b2-a2*b1)<1e-20)
    {
        a=a2;b=b2;c=c2;
        return;
    }
    double xx,yy; //(xx,yy)是入射线与镜面的交点。
    xx=(b1*c2-b2*c1)/(a1*b2-a2*b1);
    yy=(a2*c1-a1*c2)/(a1*b2-a2*b1);
    a=n;
    b=-m;
    c=m*yy-xx*n;
}
```

1.6 面积

```
#include <math.h>
struct point{double x,y;};

//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);
}

//计算三角形面积,输入三顶点
double area_triangle(point p1,point p2,point p3){
    return fabs(xmult(p1,p2,p3))/2;
}
double area_triangle(double x1,double y1,double x2,double y2,double x3,double y3){
    return fabs(xmult(x1,y1,x2,y2,x3,y3))/2;
}

//计算三角形面积,输入三边长
double area_triangle(double a,double b,double c){
    double s=(a+b+c)/2;
    return sqrt(s*(s-a)*(s-b)*(s-c));
}

//计算多边形面积,顶点按顺时针或逆时针给出
double area_polygon(int n,point* p){
    double s1=0,s2=0;
    int i;
    for (i=0;i<n;i++)
        s1+=p[(i+1)%n].y*p[i].x,s2+=p[(i+1)%n].y*p[(i+2)%n].x;
    return fabs(s1-s2)/2;
}
```

1.7 球面

```
#include <math.h>
const double pi=acos(-1);

//计算圆心角 lat 表示纬度,-90<=w<=90,lng 表示经度
//返回两点所在大圆劣弧对应圆心角,0<=angle<=pi
```

```
double angle(double lng1,double lat1,double lng2,double lat2){
    double dlng=fabs(lng1-lng2)*pi/180;
    while (dlng>=pi+pi)
        dlng-=pi+pi;
    if (dlng>pi)
        dlng=pi+pi-dlng;
    lat1*=pi/180,lat2*=pi/180;
    return acos(cos(lat1)*cos(lat2)*cos(dlng)+sin(lat1)*sin(lat2));
}

//计算距离,r 为球半径
double line_dist(double r,double lng1,double lat1,double lng2,double lat2){
    double dlng=fabs(lng1-lng2)*pi/180;
    while (dlng>=pi+pi)
        dlng-=pi+pi;
    if (dlng>pi)
        dlng=pi+pi-dlng;
    lat1*=pi/180,lat2*=pi/180;
    return r*sqrt(2-2*(cos(lat1)*cos(lat2)*cos(dlng)+sin(lat1)*sin(lat2)));
}

//计算球面距离,r 为球半径
inline double sphere_dist(double r,double lng1,double lat1,double lng2,double lat2){
    return r*angle(lng1,lat1,lng2,lat2);
}
```

1.8 三角形

```
#include <math.h>
struct point{double x,y;};
struct line{point a,b;};

double distance(point p1,point p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}

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;
}
```

```
//外心
point circumcenter(point a,point b,point c){
    line u,v;
    u.a.x=(a.x+b.x)/2;
    u.a.y=(a.y+b.y)/2;
    u.b.x=u.a.x-a.y+b.y;
    u.b.y=u.a.y+a.x-b.x;
    v.a.x=(a.x+c.x)/2;
    v.a.y=(a.y+c.y)/2;
    v.b.x=v.a.x-a.y+c.y;
    v.b.y=v.a.y+a.x-c.x;
    return intersection(u,v);
}

//内心
point incenter(point a,point b,point c){
    line u,v;
    double m,n;
    u.a=a;
    m=atan2(b.y-a.y,b.x-a.x);
    n=atan2(c.y-a.y,c.x-a.x);
    u.b.x=u.a.x+cos((m+n)/2);
    u.b.y=u.a.y+sin((m+n)/2);
    v.a=b;
    m=atan2(a.y-b.y,a.x-b.x);
    n=atan2(c.y-b.y,c.x-b.x);
    v.b.x=v.a.x+cos((m+n)/2);
    v.b.y=v.a.y+sin((m+n)/2);
    return intersection(u,v);
}

//垂心
point perpencenter(point a,point b,point c){
    line u,v;
    u.a=c;
    u.b.x=u.a.x-a.y+b.y;
    u.b.y=u.a.y+a.x-b.x;
    v.a=b;
    v.b.x=v.a.x-a.y+c.y;
    v.b.y=v.a.y+a.x-c.x;
    return intersection(u,v);
}
```

```
//重心
//到三角形三顶点距离的平方和最小的点
//三角形内到三边距离之积最大的点
point barycenter(point a,point b,point c){
    line u,v;
    u.a.x=(a.x+b.x)/2;
    u.a.y=(a.y+b.y)/2;
    u.b=c;
    v.a.x=(a.x+c.x)/2;
    v.a.y=(a.y+c.y)/2;
    v.b=b;
    return intersection(u,v);
}

//费马点
//到三角形三顶点距离之和最小的点
point fermentpoint(point a,point b,point c){
    point u,v;
    double step=fabs(a.x)+fabs(a.y)+fabs(b.x)+fabs(b.y)+fabs(c.x)+fabs(c.y);
    int i,j,k;
    u.x=(a.x+b.x+c.x)/3;
    u.y=(a.y+b.y+c.y)/3;
    while (step>1e-10)
        for (k=0;k<10;step/=2,k++)
            for (i=-1;i<=1;i++)
                for (j=-1;j<=1;j++){
                    v.x=u.x+step*i;
                    v.y=u.y+step*j;
                    if
(distance(u,a)+distance(u,b)+distance(u,c)>distance(v,a)+distance(v,b)+distance(v,c))
                        u=v;
                }
    return u;
}

//求曲率半径 三角形内最大可围成面积
#include<iostream>
 #include<cmath>
 using namespace std;
 const double pi=3.14159265358979;
 int main()
 {
     double a,b,c,d,p,s,r,ans,R,x,l; int T=0;
    while(cin>>a>>b>>c>>d&&a+b+c+d)
```

```
    {
        T++;
        l=a+b+c;
        p=l/2;
        s=sqrt(p*(p-a)*(p-b)*(p-c));
        R= s /p;
        if (d >= l)    ans = s;
        else if(2*pi*R>=d) ans=d*d/(4*pi);
        else
        {
            r = (l-d)/((l/R)-(2*pi));
            x = r*r*s/(R*R);
            ans = s - x + pi * r * r;
        }
        printf("Case %d: %.2lf\n",T,ans);
    }
    return 0;
}
```

1.9 三维几何

```
//三维几何函数库
#include <math.h>
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point3{double x,y,z;};
struct line3{point3 a,b;};
struct plane3{point3 a,b,c;};

//计算 cross product U x V
point3 xmult(point3 u,point3 v){
    point3 ret;
    ret.x=u.y*v.z-v.y*u.z;
    ret.y=u.z*v.x-u.x*v.z;
    ret.z=u.x*v.y-u.y*v.x;
    return ret;
}

//计算 dot product U . V
double dmult(point3 u,point3 v){
    return u.x*v.x+u.y*v.y+u.z*v.z;
}

//矢量差 U - V
```

```
point3 subt(point3 u,point3 v){
    point3 ret;
    ret.x=u.x-v.x;
    ret.y=u.y-v.y;
    ret.z=u.z-v.z;
    return ret;
}

//取平面法向量
point3 pvec(plane3 s){
    return xmult(subt(s.a,s.b),subt(s.b,s.c));
}
point3 pvec(point3 s1,point3 s2,point3 s3){
    return xmult(subt(s1,s2),subt(s2,s3));
}

//两点距离,单参数取向量大小
double distance(point3 p1,point3 p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-p2.z)*(p1.z-p2.z));
}

//向量大小
double vlen(point3 p){
    return sqrt(p.x*p.x+p.y*p.y+p.z*p.z);
}

//判三点共线
int dots_inline(point3 p1,point3 p2,point3 p3){
    return vlen(xmult(subt(p1,p2),subt(p2,p3)))<eps;
}

//判四点共面
int dots_onplane(point3 a,point3 b,point3 c,point3 d){
    return zero(dmult(pvec(a,b,c),subt(d,a)));
}

//判点是否在线段上,包括端点和共线
int dot_online_in(point3 p,line3 l){
    return zero(vlen(xmult(subt(p,l.a),subt(p,l.b))))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&
        (l.a.y-p.y)*(l.b.y-p.y)<eps&&(l.a.z-p.z)*(l.b.z-p.z)<eps;
}
int dot_online_in(point3 p,point3 l1,point3 l2){
    return zero(vlen(xmult(subt(p,l1),subt(p,l2))))&&(l1.x-p.x)*(l2.x-p.x)<eps&&
        (l1.y-p.y)*(l2.y-p.y)<eps&&(l1.z-p.z)*(l2.z-p.z)<eps;
```

```
}

//判点是否在线段上,不包括端点
int dot_online_ex(point3 p,line3 l){
    return dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y)||!zero(p.z-l.a.z))&&
        (!zero(p.x-l.b.x)||!zero(p.y-l.b.y)||!zero(p.z-l.b.z));
}
int dot_online_ex(point3 p,point3 l1,point3 l2){
    return dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y)||!zero(p.z-l1.z))&&
        (!zero(p.x-l2.x)||!zero(p.y-l2.y)||!zero(p.z-l2.z));
}

//判点是否在空间三角形上,包括边界,三点共线无意义
int dot_inplane_in(point3 p,plane3 s){
    return zero(vlen(xmult(subt(s.a,s.b),subt(s.a,s.c)))-vlen(xmult(subt(p,s.a),subt(p,s.b)))-
        vlen(xmult(subt(p,s.b),subt(p,s.c)))-vlen(xmult(subt(p,s.c),subt(p,s.a))));
}
int dot_inplane_in(point3 p,point3 s1,point3 s2,point3 s3){
    return zero(vlen(xmult(subt(s1,s2),subt(s1,s3)))-vlen(xmult(subt(p,s1),subt(p,s2)))-
        vlen(xmult(subt(p,s2),subt(p,s3)))-vlen(xmult(subt(p,s3),subt(p,s1))));
}

//判点是否在空间三角形上,不包括边界,三点共线无意义
int dot_inplane_ex(point3 p,plane3 s){
    return dot_inplane_in(p,s)&&vlen(xmult(subt(p,s.a),subt(p,s.b)))>eps&&
        vlen(xmult(subt(p,s.b),subt(p,s.c)))>eps&&vlen(xmult(subt(p,s.c),subt(p,s.a)))>eps;
}
int dot_inplane_ex(point3 p,point3 s1,point3 s2,point3 s3){
    return dot_inplane_in(p,s1,s2,s3)&&vlen(xmult(subt(p,s1),subt(p,s2)))>eps&&
        vlen(xmult(subt(p,s2),subt(p,s3)))>eps&&vlen(xmult(subt(p,s3),subt(p,s1)))>eps;
}

//判两点在线段同侧,点在线段上返回 0,不共面无意义
int same_side(point3 p1,point3 p2,line3 l){
    return dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b)))>eps;
}
int same_side(point3 p1,point3 p2,point3 l1,point3 l2){
    return dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))>eps;
}

//判两点在线段异侧,点在线段上返回 0,不共面无意义
int opposite_side(point3 p1,point3 p2,line3 l){
    return dmult(xmult(subt(l.a,l.b),subt(p1,l.b)),xmult(subt(l.a,l.b),subt(p2,l.b)))<-eps;
}
```

```
int opposite_side(point3 p1,point3 p2,point3 l1,point3 l2){
    return dmult(xmult(subt(l1,l2),subt(p1,l2)),xmult(subt(l1,l2),subt(p2,l2)))<-eps;
}

//判两点在平面同侧,点在平面上返回 0
int same_side(point3 p1,point3 p2,plane3 s){
    return dmult(pvec(s),subt(p1,s.a))*dmult(pvec(s),subt(p2,s.a))>eps;
}
int same_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
    return dmult(pvec(s1,s2,s3),subt(p1,s1))*dmult(pvec(s1,s2,s3),subt(p2,s1))>eps;
}

//判两点在平面异侧,点在平面上返回 0
int opposite_side(point3 p1,point3 p2,plane3 s){
    return dmult(pvec(s),subt(p1,s.a))*dmult(pvec(s),subt(p2,s.a))<-eps;
}
int opposite_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3){
    return dmult(pvec(s1,s2,s3),subt(p1,s1))*dmult(pvec(s1,s2,s3),subt(p2,s1))<-eps;
}

//判两直线平行
int parallel(line3 u,line3 v){
    return vlen(xmult(subt(u.a,u.b),subt(v.a,v.b)))<eps;
}
int parallel(point3 u1,point3 u2,point3 v1,point3 v2){
    return vlen(xmult(subt(u1,u2),subt(v1,v2)))<eps;
}

//判两平面平行
int parallel(plane3 u,plane3 v){
    return vlen(xmult(pvec(u),pvec(v)))<eps;
}
int parallel(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    return vlen(xmult(pvec(u1,u2,u3),pvec(v1,v2,v3)))<eps;
}

//判直线与平面平行
int parallel(line3 l,plane3 s){
    return zero(dmult(subt(l.a,l.b),pvec(s)));
}
int parallel(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return zero(dmult(subt(l1,l2),pvec(s1,s2,s3)));
}
```

```
//判两直线垂直
int perpendicular(line3 u,line3 v){
    return zero(dmult(subt(u.a,u.b),subt(v.a,v.b)));
}
int perpendicular(point3 u1,point3 u2,point3 v1,point3 v2){
    return zero(dmult(subt(u1,u2),subt(v1,v2)));
}

//判两平面垂直
int perpendicular(plane3 u,plane3 v){
    return zero(dmult(pvec(u),pvec(v)));
}
int perpendicular(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    return zero(dmult(pvec(u1,u2,u3),pvec(v1,v2,v3)));
}

//判直线与平面平行
int perpendicular(line3 l,plane3 s){
    return vlen(xmult(subt(l.a,l.b),pvec(s)))<eps;
}
int perpendicular(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return vlen(xmult(subt(l1,l2),pvec(s1,s2,s3)))<eps;
}

//判两线段相交,包括端点和部分重合
int intersect_in(line3 u,line3 v){
    if (!dots_onplane(u.a,u.b,v.a,v.b))
        return 0;
    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(point3 u1,point3 u2,point3 v1,point3 v2){
    if (!dots_onplane(u1,u2,v1,v2))
        return 0;
    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_online_in(v2,u1,u2);
}

//判两线段相交,不包括端点和部分重合
int intersect_ex(line3 u,line3 v){
    return dots_onplane(u.a,u.b,v.a,v.b)&&opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
```

```
}
int intersect_ex(point3 u1,point3 u2,point3 v1,point3 v2){
    return dots_onplane(u1,u2,v1,v2)&&opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
}

//判线段与空间三角形相交,包括交于边界和(部分)包含
int intersect_in(line3 l,plane3 s){
    return !same_side(l.a,l.b,s)&&!same_side(s.a,s.b,l.a,l.b,s.c)&&
        !same_side(s.b,s.c,l.a,l.b,s.a)&&!same_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_in(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return !same_side(l1,l2,s1,s2,s3)&&!same_side(s1,s2,l1,l2,s3)&&
        !same_side(s2,s3,l1,l2,s1)&&!same_side(s3,s1,l1,l2,s2);
}

//判线段与空间三角形相交,不包括交于边界和(部分)包含
int intersect_ex(line3 l,plane3 s){
    return opposite_side(l.a,l.b,s)&&opposite_side(s.a,s.b,l.a,l.b,s.c)&&
        opposite_side(s.b,s.c,l.a,l.b,s.a)&&opposite_side(s.c,s.a,l.a,l.b,s.b);
}
int intersect_ex(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return opposite_side(l1,l2,s1,s2,s3)&&opposite_side(s1,s2,l1,l2,s3)&&
        opposite_side(s2,s3,l1,l2,s1)&&opposite_side(s3,s1,l1,l2,s2);
}

//计算两直线交点,注意事先判断直线是否共面和平行!
//线段交点请另外判线段相交(同时还是要判断是否平行!)
point3 intersection(line3 u,line3 v){
    point3 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;
    ret.z+=(u.b.z-u.a.z)*t;
    return ret;
}
point3 intersection(point3 u1,point3 u2,point3 v1,point3 v2){
    point3 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;
    ret.z+=(u2.z-u1.z)*t;
    return ret;
```

```
}

//计算直线与平面交点,注意事先判断是否平行,并保证三点不共线!
//线段和空间三角形交点请另外判断
point3 intersection(line3 l,plane3 s){
    point3 ret=pvec(s);
    double t=(ret.x*(s.a.x-l.a.x)+ret.y*(s.a.y-l.a.y)+ret.z*(s.a.z-l.a.z))/
        (ret.x*(l.b.x-l.a.x)+ret.y*(l.b.y-l.a.y)+ret.z*(l.b.z-l.a.z));
    ret.x=l.a.x+(l.b.x-l.a.x)*t;
    ret.y=l.a.y+(l.b.y-l.a.y)*t;
    ret.z=l.a.z+(l.b.z-l.a.z)*t;
    return ret;
}
point3 intersection(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    point3 ret=pvec(s1,s2,s3);
    double t=(ret.x*(s1.x-l1.x)+ret.y*(s1.y-l1.y)+ret.z*(s1.z-l1.z))/
        (ret.x*(l2.x-l1.x)+ret.y*(l2.y-l1.y)+ret.z*(l2.z-l1.z));
    ret.x=l1.x+(l2.x-l1.x)*t;
    ret.y=l1.y+(l2.y-l1.y)*t;
    ret.z=l1.z+(l2.z-l1.z)*t;
    return ret;
}

//计算两平面交线,注意事先判断是否平行,并保证三点不共线!
line3 intersection(plane3 u,plane3 v){
    line3 ret;
    ret.a=parallel(v.a,v.b,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):intersection(v.a,v.b,u.a,u.b,u.c);
    ret.b=parallel(v.c,v.a,u.a,u.b,u.c)?intersection(v.b,v.c,u.a,u.b,u.c):intersection(v.c,v.a,u.a,u.b,u.c);
    return ret;
}
line3 intersection(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    line3 ret;
    ret.a=parallel(v1,v2,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v1,v2,u1,u2,u3);
    ret.b=parallel(v3,v1,u1,u2,u3)?intersection(v2,v3,u1,u2,u3):intersection(v3,v1,u1,u2,u3);
    return ret;
}

//点到直线距离
double ptoline(point3 p,line3 l){
    return vlen(xmult(subt(p,l.a),subt(l.b,l.a)))/distance(l.a,l.b);
}
double ptoline(point3 p,point3 l1,point3 l2){
    return vlen(xmult(subt(p,l1),subt(l2,l1)))/distance(l1,l2);
}
```

```
//点到平面距离
double ptoplane(point3 p,plane3 s){
    return fabs(dmult(pvec(s),subt(p,s.a)))/vlen(pvec(s));
}
double ptoplane(point3 p,point3 s1,point3 s2,point3 s3){
    return fabs(dmult(pvec(s1,s2,s3),subt(p,s1)))/vlen(pvec(s1,s2,s3));
}

//直线到直线距离
double linetoline(line3 u,line3 v){
    point3 n=xmult(subt(u.a,u.b),subt(v.a,v.b));
    return fabs(dmult(subt(u.a,v.a),n))/vlen(n);
}
double linetoline(point3 u1,point3 u2,point3 v1,point3 v2){
    point3 n=xmult(subt(u1,u2),subt(v1,v2));
    return fabs(dmult(subt(u1,v1),n))/vlen(n);
}

//两直线夹角 cos 值
double angle_cos(line3 u,line3 v){
    return dmult(subt(u.a,u.b),subt(v.a,v.b))/vlen(subt(u.a,u.b))/vlen(subt(v.a,v.b));
}
double angle_cos(point3 u1,point3 u2,point3 v1,point3 v2){
    return dmult(subt(u1,u2),subt(v1,v2))/vlen(subt(u1,u2))/vlen(subt(v1,v2));
}

//两平面夹角 cos 值
double angle_cos(plane3 u,plane3 v){
    return dmult(pvec(u),pvec(v))/vlen(pvec(u))/vlen(pvec(v));
}
double angle_cos(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){
    return dmult(pvec(u1,u2,u3),pvec(v1,v2,v3))/vlen(pvec(u1,u2,u3))/vlen(pvec(v1,v2,v3));
}

//直线平面夹角 sin 值
double angle_sin(line3 l,plane3 s){
    return dmult(subt(l.a,l.b),pvec(s))/vlen(subt(l.a,l.b))/vlen(pvec(s));
}
double angle_sin(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3){
    return dmult(subt(l1,l2),pvec(s1,s2,s3))/vlen(subt(l1,l2))/vlen(pvec(s1,s2,s3));
}
```

1.10 凸包

```
#include<iostream>
#include<cmath>
#include<algorithm>
using namespace std;
const int maxn=500;
struct p
{
      double x,y;
};
int n,top;
p list[maxn];
int s[maxn];

double m(p p1,p p2,p p0)
{
 return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}

bool cmp(p p1,p p2)
{
 double t;
 t=m(p1,p2,list[0]);
if(t>0||(t==0&&pow(p1.x-list[0].x,2)+pow(p1.y-list[0].y,2)<pow(p2.x-list[0].x,2)+pow(p2.y-list[0].y,2)))
     return true;
 else return false;
}
void init()
{
 int i;    p t;
 cin>>n;
 for(i=0;i<n;i++)
      {
      cin>>list[i].x>>list[i].y;
      if ((list[i].y<list[0].y)||(list[i].y==list[0].y&&list[i].x<list[0].x)) {t=list[0];list[0]=list[i]; list[i]=t;}
      }
 sort(list+1,list+n,cmp);
}
 void graham()
 {
   int i;
   for(i=0;i<3;i++) s[i]=i;
```

```
    top=2;
    for(i=3;i<n;i++)
        {
          while (m(list[i],list[s[top]],list[s[top-1]])>=0) top--;
         s[++top]=i;
        }
    for(i=0;i<=top;i++)
      printf("( %.2lf , %.2lf )\n",list[s[i]].x,list[s[i]].y);
  }
  int main()
  {
  init();
  graham();
  return 0;
  }
// 卷包裹法求点集凸壳
void ConvexClosure(POINT PointSet[],POINT ch[],int n,int &len)
{
      int top=0,i,index,first;
      double curmax,curcos,curdis;
      POINT tmp;
      LINESEG l1,l2;
      bool use[MAXV];
      tmp=PointSet[0];
      index=0;
      // 选取 y 最小点，如果多于一个，则选取最左点
      for(i=1;i<n;i++)
      {
            if(PointSet[i].y<tmp.y||PointSet[i].y == tmp.y&&PointSet[i].x<tmp.x)
            {
                  index=i;
            }
            use[i]=false;
      }
      tmp=PointSet[index];
      first=index;
      use[index]=true;

      index=-1;
      ch[top++]=tmp;
      tmp.x-=100;
      l1.s=tmp;
      l1.e=ch[0];
      l2.s=ch[0];
```

```
    while(index!=first)
    {
        curmax=-100;
        curdis=0;
        // 选取与最后一条确定边夹角最小的点，即余弦值最大者
        for(i=0;i<n;i++)
        {
            if(use[i])continue;
            l2.e=PointSet[i];
            curcos=cosine(l1,l2); // 根据 cos 值求夹角余弦，范围在 （-1 -- 1 ）
            if(curcos>curmax || fabs(curcos-curmax)<1e-6 && dist(l2.s,l2.e)>curdis)
            {
                curmax=curcos;
                index=i;
                curdis=dist(l2.s,l2.e);
            }

        }
        use[first]=false;                    //清空第 first 个顶点标志，使最后能形成封闭的 hull
        use[index]=true;
        ch[top++]=PointSet[index];
        l1.s=ch[top-2];
        l1.e=ch[top-1];
        l2.s=ch[top-1];
    }
    len=top-1;
}
```

1.11 网格

```
#define abs(x) ((x)>0?(x):-(x))
struct point{int x,y;};

int gcd(int a,int b){return b?gcd(b,a%b):a;}

//多边形上的网格点个数
int grid_onedge(int n,point* p){
    int i,ret=0;
    for (i=0;i<n;i++)
        ret+=gcd(abs(p[i].x-p[(i+1)%n].x),abs(p[i].y-p[(i+1)%n].y));
    return ret;
}
```

```
//多边形内的网格点个数
int grid_inside(int n,point* p){
    int i,ret=0;
    for (i=0;i<n;i++)
        ret+=p[(i+1)%n].y*(p[i].x-p[(i+2)%n].x);
    return (abs(ret)-grid_onedge(n,p))/2+1;
}
```

1.12 圆

```
#include <math.h>
#define eps 1e-8
struct point{double x,y;};

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 distance(point p1,point p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}

double disptoline(point p,point l1,point l2){
    return fabs(xmult(p,l1,l2))/distance(l1,l2);
}

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;
}

//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=distance(c,l1)-r,t2=distance(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 distance(c1,c2)<r1+r2+eps&&distance(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 (distance(p,c)<eps)
        return p;
    u.x=c.x+r*fabs(c.x-p.x)/distance(c,p);
    u.y=c.y+r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
    v.x=c.x-r*fabs(c.x-p.x)/distance(c,p);
    v.y=c.y-r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
    return distance(u,p)<distance(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-distance(p,c)*distance(p,c))/distance(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)/distance(c1,c2)/distance(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);
}

//将向量 p 逆时针旋转 angle 角度
Point Rotate(Point p,double angle) {
    Point res;
    res.x=p.x*cos(angle)-p.y*sin(angle);
    res.y=p.x*sin(angle)+p.y*cos(angle);
    return res;
}
//求圆外一点对圆(o,r)的两个切点 result1 和 result2
void TangentPoint_PC(Point poi,Point o,double r,Point &result1,Point &result2) {
    double line=sqrt((poi.x-o.x)*(poi.x-o.x)+(poi.y-o.y)*(poi.y-o.y));
    double angle=acos(r/line);
    Point unitvector,lin;
    lin.x=poi.x-o.x;
    lin.y=poi.y-o.y;
    unitvector.x=lin.x/sqrt(lin.x*lin.x+lin.y*lin.y)*r;
    unitvector.y=lin.y/sqrt(lin.x*lin.x+lin.y*lin.y)*r;
    result1=Rotate(unitvector,-angle);
    result2=Rotate(unitvector,angle);
    result1.x+=o.x;
    result1.y+=o.y;
    result2.x+=o.x;
    result2.y+=o.y;
    return;
}
```

1.13 矢量运算求几何模板

```
#include <iostream>
#include <cmath>
#include <vector>
#include <algorithm>
#define MAX_N 100
using namespace std;

////////////////////////////////////////////////////////////////
//常量区
const double INF            = 1e10;      //  无穷大
```

```
const double EPS            = 1e-15;      // 计算精度
const int LEFT              = 0;          // 点在直线左边
const int RIGHT             = 1;          // 点在直线右边
const int ONLINE            = 2;          // 点在直线上
const int CROSS             = 0;          // 两直线相交
const int COLINE            = 1;          // 两直线共线
const int PARALLEL          = 2;          // 两直线平行
const int NOTCOPLANAR       = 3;          // 两直线不共面
const int INSIDE            = 1;          // 点在图形内部
const int OUTSIDE           = 2;          // 点在图形外部
const int BORDER            = 3;          // 点在图形边界
const int BAOHAN            = 1;          // 大圆包含小圆
const int NEIQIE            = 2;          // 内切
const int XIANJIAO          = 3;          // 相交
const int WAIQIE            = 4;          // 外切
const int XIANLI            = 5;          // 相离
////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////
//类型定义区
struct Point {                  // 二维点或矢量
    double x, y;
    double angle, dis;
    Point() {}
    Point(double x0, double y0): x(x0), y(y0) {}
};
struct Point3D {                //三维点或矢量
    double x, y, z;
    Point3D() {}
    Point3D(double x0, double y0, double z0): x(x0), y(y0), z(z0) {}
};
struct Line {                   // 二维的直线或线段
    Point p1, p2;
    Line() {}
    Line(Point p10, Point p20): p1(p10), p2(p20) {}
};
struct Line3D {                 // 三维的直线或线段
    Point3D p1, p2;
    Line3D() {}
    Line3D(Point3D p10, Point3D p20): p1(p10), p2(p20) {}
};
struct Rect {                   // 用长宽表示矩形的方法  w, h 分别表示宽度和高度
    double w, h;
  Rect() {}
```

```
  Rect(double _w,double _h) : w(_w),h(_h) {}
};
struct Rect_2 {                    // 表示矩形，左下角坐标是(xl, yl)，右上角坐标是(xh, yh)
      double xl, yl, xh, yh;
  Rect_2() {}
  Rect_2(double _xl,double _yl,double _xh,double _yh) : xl(_xl),yl(_yl),xh(_xh),yh(_yh) {}
};
struct Circle {                    //圆
  Point c;
  double r;
  Circle() {}
  Circle(Point _c,double _r) :c(_c),r(_r) {}
};
typedef vector<Point> Polygon;          // 二维多边形
typedef vector<Point> Points;           // 二维点集
typedef vector<Point3D> Points3D;     // 三维点集
//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////
//基本函数区
inline double max(double x,double y)
{
      return x > y ? x : y;
}
inline double min(double x, double y)
{
      return x > y ? y : x;
}
inline bool ZERO(double x)                    // x == 0
{
      return (fabs(x) < EPS);
}
inline bool ZERO(Point p)                     // p == 0
{
      return (ZERO(p.x) && ZERO(p.y));
}
inline bool ZERO(Point3D p)                     // p == 0
{
      return (ZERO(p.x) && ZERO(p.y) && ZERO(p.z));
}
inline bool EQ(double x, double y)          // eqaul, x == y
{
      return (fabs(x - y) < EPS);
}
```

```
inline bool NEQ(double x, double y)          // not equal, x != y
{
      return (fabs(x - y) >= EPS);
}
inline bool LT(double x, double y)          // less than, x < y
{
      return ( NEQ(x, y) && (x < y) );
}
inline bool GT(double x, double y)          // greater than, x > y
{
      return ( NEQ(x, y) && (x > y) );
}
inline bool LEQ(double x, double y)          // less equal, x <= y
{
      return ( EQ(x, y) || (x < y) );
}
inline bool GEQ(double x, double y)          // greater equal, x >= y
{
      return ( EQ(x, y) || (x > y) );
}
// 注意!!!
// 如果是一个很小的负的浮点数
// 保留有效位数输出的时候会出现-0.000 这样的形式，
// 前面多了一个负号
// 这就会导致错误!!!!!!
// 因此在输出浮点数之前，一定要调用次函数进行修正!
inline double FIX(double x)
{
      return (fabs(x) < EPS) ? 0 : x;
}
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
//二维矢量运算
bool operator==(Point p1, Point p2)
{
      return ( EQ(p1.x, p2.x) &&    EQ(p1.y, p2.y) );
}
bool operator!=(Point p1, Point p2)
{
      return ( NEQ(p1.x, p2.x) ||    NEQ(p1.y, p2.y) );
}
bool operator<(Point p1, Point p2)
{
```

```
        if (NEQ(p1.x, p2.x)) {
            return (p1.x < p2.x);
        } else {
            return (p1.y < p2.y);
        }
}
Point operator+(Point p1, Point p2)
{
        return Point(p1.x + p2.x, p1.y + p2.y);
}
Point operator-(Point p1, Point p2)
{
        return Point(p1.x - p2.x, p1.y - p2.y);
}
double operator*(Point p1, Point p2) // 计算叉乘 p1 × p2
{
        return (p1.x * p2.y - p2.x * p1.y);
}
double operator&(Point p1, Point p2) { // 计算点积 p1・p2
        return (p1.x * p2.x + p1.y * p2.y);
}
double Norm(Point p) // 计算矢量 p 的模
{
        return sqrt(p.x * p.x + p.y * p.y);
}
// 把矢量 p 旋转角度 angle (弧度表示)
// angle > 0 表示逆时针旋转
// angle < 0 表示顺时针旋转
Point Rotate(Point p, double angle)
{
        Point result;
        result.x = p.x * cos(angle) - p.y * sin(angle);
        result.y = p.x * sin(angle) + p.y * cos(angle);
        return result;
}
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
//三维矢量运算
bool operator==(Point3D p1, Point3D p2)
{
        return ( EQ(p1.x, p2.x) && EQ(p1.y, p2.y) && EQ(p1.z, p2.z) );
}
bool operator<(Point3D p1, Point3D p2)
```

```
{
	if (NEQ(p1.x, p2.x)) {
		return (p1.x < p2.x);
	} else if (NEQ(p1.y, p2.y)) {
		return (p1.y < p2.y);
	} else {
		return (p1.z < p2.z);
	}
}
Point3D operator+(Point3D p1, Point3D p2)
{
	return Point3D(p1.x + p2.x, p1.y + p2.y, p1.z + p2.z);
}
Point3D operator-(Point3D p1, Point3D p2)
{
	return Point3D(p1.x - p2.x, p1.y - p2.y, p1.z - p2.z);
}
Point3D operator*(Point3D p1, Point3D p2) //  计算叉乘  p1 x p2
{
	return Point3D(p1.y * p2.z - p1.z * p2.y,
		p1.z * p2.x - p1.x * p2.z,
		p1.x * p2.y - p1.y * p2.x );
}
double operator&(Point3D p1, Point3D p2) { //  计算点积  p1・p2
	return (p1.x * p2.x + p1.y * p2.y + p1.z * p2.z);
}
double Norm(Point3D p) //  计算矢量 p 的模
{
	return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
}

////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
//点.线段.直线问题
//
double Distance(Point p1, Point p2) //2 点间的距离
{
 return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double Distance(Point3D p1, Point3D p2) //2 点间的距离,三维
{
 return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-p2.z)*(p1.z-p2.z));
}
```

```
double Distance(Point p, Line L) // 求二维平面上点到直线的距离
{
    return ( fabs((p - L.p1) * (L.p2 - L.p1)) / Norm(L.p2 - L.p1) );
}
double Distance(Point3D p, Line3D L)// 求三维空间中点到直线的距离
{
    return ( Norm((p - L.p1) * (L.p2 - L.p1)) / Norm(L.p2 - L.p1) );
}
bool OnLine(Point p, Line L) // 判断二维平面上点 p 是否在直线 L 上
{
    return ZERO( (p - L.p1) * (L.p2 - L.p1) );
}
bool OnLine(Point3D p, Line3D L) // 判断三维空间中点 p 是否在直线 L 上
{
    return ZERO( (p - L.p1) * (L.p2 - L.p1) );
}
int Relation(Point p, Line L) // 计算点 p 与直线 L 的相对关系 ,返回 ONLINE,LEFT,RIGHT
{
    double res = (L.p2 - L.p1) * (p - L.p1);
    if (EQ(res, 0)) {
        return ONLINE;
    } else if (res > 0) {
        return LEFT;
    } else {
        return RIGHT;
    }
}
bool SameSide(Point p1, Point p2, Line L) // 判断点 p1, p2 是否在直线 L 的同侧
{
    double m1 = (p1 - L.p1) * (L.p2 - L.p1);
    double m2 = (p2 - L.p1) * (L.p2 - L.p1);
    return GT(m1 * m2, 0);
}
bool OnLineSeg(Point p, Line L) // 判断二维平面上点 p 是否在线段 l 上
{
    return ( ZERO( (L.p1 - p) * (L.p2 - p) ) &&
        LEQ((p.x - L.p1.x)*(p.x - L.p2.x), 0) &&
        LEQ((p.y - L.p1.y)*(p.y - L.p2.y), 0) );
}
bool OnLineSeg(Point3D p, Line3D L) // 判断三维空间中点 p 是否在线段 l 上
{
    return ( ZERO((L.p1 - p) * (L.p2 - p)) &&
        EQ( Norm(p - L.p1) + Norm(p - L.p2), Norm(L.p2 - L.p1)) );
}
```

```
Point SymPoint(Point p, Line L) // 求二维平面上点 p 关于直线 L 的对称点
{
    Point result;
    double a = L.p2.x - L.p1.x;
    double b = L.p2.y - L.p1.y;
    double t = ( (p.x - L.p1.x) * a + (p.y - L.p1.y) * b ) / (a*a + b*b);
    result.x = 2 * L.p1.x + 2 * a * t - p.x;
    result.y = 2 * L.p1.y + 2 * b * t - p.y;
    return result;
}
bool Coplanar(Points3D points) // 判断一个点集中的点是否全部共面
{
    int i;
    Point3D p;

    if (points.size() < 4) return true;
    p = (points[2] - points[0]) * (points[1] - points[0]);
    for (i = 3; i < points.size(); i++) {
        if (! ZERO(p & points[i]) ) return false;
    }
    return true;
}
bool LineIntersect(Line L1, Line L2) // 判断二维的两直线是否相交
{
    return (! ZERO((L1.p1 - L1.p2)*(L2.p1 - L2.p2)) );   // 是否平行
}
bool LineIntersect(Line3D L1, Line3D L2) // 判断三维的两直线是否相交
{
    Point3D p1 = L1.p1 - L1.p2;
    Point3D p2 = L2.p1 - L2.p2;
    Point3D p   = p1 * p2;
    if (ZERO(p)) return false;          // 是否平行
    p = (L2.p1 - L1.p2) * (L1.p1 - L1.p2);
    return ZERO(p & L2.p2);             // 是否共面
}
bool LineSegIntersect(Line L1, Line L2) // 判断二维的两条线段是否相交
{
    return ( GEQ( max(L1.p1.x, L1.p2.x), min(L2.p1.x, L2.p2.x) ) &&
        GEQ( max(L2.p1.x, L2.p2.x), min(L1.p1.x, L1.p2.x) ) &&
        GEQ( max(L1.p1.y, L1.p2.y), min(L2.p1.y, L2.p2.y) ) &&
        GEQ( max(L2.p1.y, L2.p2.y), min(L1.p1.y, L1.p2.y) ) &&
        LEQ( ((L2.p1 - L1.p1) * (L1.p2 - L1.p1)) * ((L2.p2 -   L1.p1) * (L1.p2 - L1.p1)), 0 ) &&
        LEQ( ((L1.p1 - L2.p1) * (L2.p2 - L2.p1)) * ((L1.p2 -   L2.p1) * (L2.p2 - L2.p1)), 0 ) );
}
```

```
bool LineSegIntersect(Line3D L1, Line3D L2) // 判断三维的两条线段是否相交
{
    // todo
    return true;
}
// 计算两条二维直线的交点，结果在参数 P 中返回
// 返回值说明了两条直线的位置关系: COLINE -- 共线 PARALLEL -- 平行 CROSS -- 相交
int CalCrossPoint(Line L1, Line L2, Point& P)
{
    double A1, B1, C1, A2, B2, C2;

    A1 = L1.p2.y - L1.p1.y;
    B1 = L1.p1.x - L1.p2.x;
    C1 = L1.p2.x * L1.p1.y - L1.p1.x * L1.p2.y;

    A2 = L2.p2.y - L2.p1.y;
    B2 = L2.p1.x - L2.p2.x;
    C2 = L2.p2.x * L2.p1.y - L2.p1.x * L2.p2.y;

    if (EQ(A1 * B2, B1 * A2))    {
        if (EQ( (A1 + B1) * C2, (A2 + B2) * C1 )) {
            return COLINE;
        } else {
            return PARALLEL;
        }
    } else {
        P.x = (B2 * C1 - B1 * C2) / (A2 * B1 - A1 * B2);
        P.y = (A1 * C2 - A2 * C1) / (A2 * B1 - A1 * B2);
        return CROSS;
    }
}
// 计算两条三维直线的交点，结果在参数 P 中返回
// 返回值说明了两条直线的位置关系 COLINE -- 共线 PARALLEL -- 平行 CROSS -- 相交 NONCOPLANAR -- 不公面
int CalCrossPoint(Line3D L1, Line3D L2, Point3D& P)
{
    // todo
    return 0;
}
// 计算点 P 到直线 L 的最近点
Point NearestPointToLine(Point P, Line L)
{
    Point result;
```

```
    double a, b, t;

    a = L.p2.x - L.p1.x;
    b = L.p2.y - L.p1.y;
    t = ( (P.x - L.p1.x) * a + (P.y - L.p1.y) * b ) / (a * a + b * b);

    result.x = L.p1.x + a * t;
    result.y = L.p1.y + b * t;
    return result;
}
// 计算点 P 到线段 L 的最近点
Point NearestPointToLineSeg(Point P, Line L)
{
    Point result;
    double a, b, t;

    a = L.p2.x - L.p1.x;
    b = L.p2.y - L.p1.y;
    t = ( (P.x - L.p1.x) * a + (P.y - L.p1.y) * b ) / (a * a + b * b);

    if ( GEQ(t, 0) && LEQ(t, 1) ) {
        result.x = L.p1.x + a * t;
        result.y = L.p1.y + b * t;
    } else {
        if ( Norm(P - L.p1) < Norm(P - L.p2) ) {
            result = L.p1;
        } else {
            result = L.p2;
        }
    }
    return result;
}
// 计算险段 L1 到线段 L2 的最短距离
double MinDistance(Line L1, Line L2)
{
    double d1, d2, d3, d4;

    if (LineSegIntersect(L1, L2)) {
        return 0;
    } else {
        d1 = Norm( NearestPointToLineSeg(L1.p1, L2) - L1.p1 );
        d2 = Norm( NearestPointToLineSeg(L1.p2, L2) - L1.p2 );
        d3 = Norm( NearestPointToLineSeg(L2.p1, L1) - L2.p1 );
        d4 = Norm( NearestPointToLineSeg(L2.p2, L1) - L2.p2 );
```

```
            return min( min(d1, d2), min(d3, d4) );
        }
}
// 求二维两直线的夹角，
// 返回值是 0~Pi 之间的弧度
double Inclination(Line L1, Line L2)
{
        Point u = L1.p2 - L1.p1;
        Point v = L2.p2 - L2.p1;
        return acos( (u & v) / (Norm(u)*Norm(v)) );
}
// 求三维两直线的夹角，
// 返回值是 0~Pi 之间的弧度
double Inclination(Line3D L1, Line3D L2)
{
        Point3D u = L1.p2 - L1.p1;
        Point3D v = L2.p2 - L2.p1;
        return acos( (u & v) / (Norm(u)*Norm(v)) );
}
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// 判断两个矩形是否相交
// 如果相邻不算相交
bool Intersect(Rect_2 r1, Rect_2 r2)
{
        return ( max(r1.xl, r2.xl) < min(r1.xh, r2.xh) &&
                        max(r1.yl, r2.yl) < min(r1.yh, r2.yh) );
}
// 判断矩形 r2 是否可以放置在矩形 r1 内
// r2 可以任意地旋转
//发现原来的给出的方法过不了 OJ 上的无归之室这题，
//所以用了自己的代码
bool IsContain(Rect r1, Rect r2)          //矩形的 w>h
  {
        if(r1.w >r2.w && r1.h > r2.h) return true;
        else
        {
            double r = sqrt(r2.w*r2.w + r2.h*r2.h) / 2.0;
            double alpha = atan2(r2.h,r2.w);
            double sita = asin((r1.h/2.0)/r);
            double x = r * cos(sita - 2*alpha);
            double y = r * sin(sita - 2*alpha);
```

```
            if(x < r1.w/2.0 && y < r1.h/2.0 && x > 0 && y > -r1.h/2.0) return true;
            else return false;
        }
}
////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////
//圆
Point Center(const Circle & C) //圆心
{
    return C.c;
}

double CommonArea(const Circle & A, const Circle & B) //两个圆的公共面积
{
    double area = 0.0;
    const Circle & M = (A.r > B.r) ? A : B;
    const Circle & N = (A.r > B.r) ? B : A;
    double D = Distance(Center(M), Center(N));
    if ((D < M.r + N.r) && (D > M.r - N.r))
    {
        double cosM = (M.r * M.r + D * D - N.r * N.r) / (2.0 * M.r * D);
        double cosN = (N.r * N.r + D * D - M.r * M.r) / (2.0 * N.r * D);
        double alpha = 2.0 * acos(cosM);
        double beta  = 2.0 * acos(cosN);
        double TM = 0.5 * M.r * M.r * sin(alpha);
        double TN = 0.5 * N.r * N.r * sin(beta);
        double FM = (alpha / 360.0) * Area(M);
        double FN = (beta / 360.0) * Area(N);
        area = FM + FN - TM - TN;
    }
    else if (D <= M.r - N.r)
    {
        area = Area(N);
    }
    return area;
}

bool IsInCircle(const Circle & C, const Rect_2 & rect)//判断圆是否在矩形内(不允许相切)
{
    return (GT(C.c.x - C.r, rect.xl)
  &&  LT(C.c.x + C.r, rect.xh)
  &&  GT(C.c.y - C.r, rect.yl)
  &&  LT(C.c.y + C.r, rect.yh));
```

```
}

//判断 2 圆的位置关系
//返回:
//BAOHAN     = 1;            // 大圆包含小圆
//NEIQIE     = 2;            // 内切
//XIANJIAO = 3;              // 相交
//WAIQIE     = 4;            // 外切
//XIANLI     = 5;            // 相离
int CirCir(const Circle &c1, const Circle &c2)//判断 2 圆的位置关系
{
  double dis = Distance(c1.c,c2.c);
  if(LT(dis,fabs(c1.r-c2.r))) return BAOHAN;
  if(EQ(dis,fabs(c1.r-c2.r))) return NEIQIE;
  if(LT(dis,c1.r+c2.r) && GT(dis,fabs(c1.r-c2.r))) return XIANJIAO;
  if(EQ(dis,c1.r+c2.r)) return WAIQIE;
  return XIANLI;
}
////////////////////////////////////////////////////////////////////

int main()
{
  return 0;
}
```

1.14 结构体表示几何图形

```
//计算几何(二维)
#include <cmath>
#include <cstdio>
#include <algorithm>
using namespace std;

typedef double TYPE;
#define Abs(x) (((x)>0)?(x):(-(x)))
#define Sgn(x) (((x)<0)?(-1):(1))
#define Max(a,b) (((a)>(b))?(a):(b))
#define Min(a,b) (((a)<(b))?(a):(b))
#define Epsilon 1e-8
#define Infinity 1e+10
#define PI acos(-1.0)//3.14159265358979323846
TYPE Deg2Rad(TYPE deg){return (deg * PI / 180.0);}
TYPE Rad2Deg(TYPE rad){return (rad * 180.0 / PI);}
```

```
TYPE Sin(TYPE deg){return sin(Deg2Rad(deg));}
TYPE Cos(TYPE deg){return cos(Deg2Rad(deg));}
TYPE ArcSin(TYPE val){return Rad2Deg(asin(val));}
TYPE ArcCos(TYPE val){return Rad2Deg(acos(val));}
TYPE Sqrt(TYPE val){return sqrt(val);}

//点
struct POINT
{
   TYPE x;
   TYPE y;
   POINT() : x(0), y(0) {};
   POINT(TYPE _x_, TYPE _y_) : x(_x_), y(_y_) {};
};
// 两个点的距离
TYPE Distance(const POINT & a, const POINT & b)
{
   return Sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
//线段
struct SEG
{
   POINT a; //起点
   POINT b; //终点
   SEG() {};
   SEG(POINT _a_, POINT _b_):a(_a_),b(_b_) {};
};
//直线(两点式)
struct LINE
{
   POINT a;
   POINT b;
   LINE() {};
   LINE(POINT _a_, POINT _b_) : a(_a_), b(_b_) {};
};
//直线(一般式)
struct LINE2
{
   TYPE A,B,C;
   LINE2() {};
   LINE2(TYPE _A_, TYPE _B_, TYPE _C_) : A(_A_), B(_B_), C(_C_) {};
};

//两点式化一般式
```

```
LINE2 Line2line(const LINE & L) // y=kx+c k=y/x
{
   LINE2 L2;
   L2.A = L.b.y - L.a.y;
   L2.B = L.a.x - L.b.x;
   L2.C = L.b.x * L.a.y - L.a.x * L.b.y;
   return L2;
}

// 引用返回直线 Ax + By + C =0 的系数
void Coefficient(const LINE & L, TYPE & A, TYPE & B, TYPE & C)
{
   A = L.b.y - L.a.y;
   B = L.a.x - L.b.x;
   C = L.b.x * L.a.y - L.a.x * L.b.y;
}
void Coefficient(const POINT & p,const TYPE a,TYPE & A,TYPE & B,TYPE & C)
{
   A = Cos(a);
   B = Sin(a);
   C = - (p.y * B + p.x * A);
}
/判等(值，点，直线)
bool IsEqual(TYPE a, TYPE b)
{
   return (Abs(a - b) <Epsilon);
}
bool IsEqual(const POINT & a, const POINT & b)
{
   return (IsEqual(a.x, b.x) && IsEqual(a.y, b.y));
}
bool IsEqual(const LINE & A, const LINE & B)
{
   TYPE A1, B1, C1;
   TYPE A2, B2, C2;
   Coefficient(A, A1, B1, C1);
   Coefficient(B, A2, B2, C2);
   return IsEqual(A1 * B2, A2 * B1) && IsEqual(A1 * C2, A2 * C1) && IsEqual(B1 * C2, B2 * C1);
}
// 矩形
struct RECT
{
   POINT a; // 左下点
   POINT b; // 右上点
```

```
    RECT() {};
    RECT(const POINT & _a_, const POINT & _b_) { a = _a_; b = _b_; }
};

//矩形化标准
RECT Stdrect(const RECT & q)
{
    TYPE t;
    RECT p=q;
    if(p.a.x > p.b.x) swap(p.a.x , p.b.x);
    if(p.a.y > p.b.y) swap(p.a.y , p.b.y);
    return p;
}

//根据下标返回矩形的边
SEG Edge(const RECT & rect, int idx)
{
    SEG edge;
    while (idx < 0) idx += 4;
    switch (idx % 4)
    {
    case 0: //下边
        edge.a = rect.a;
        edge.b = POINT(rect.b.x, rect.a.y);
        break;
    case 1: //右边
        edge.a = POINT(rect.b.x, rect.a.y);
        edge.b = rect.b;
        break;
    case 2: //上边
        edge.a = rect.b;
        edge.b = POINT(rect.a.x, rect.b.y);
        break;
    case 3: //左边
        edge.a = POINT(rect.a.x, rect.b.y);
        edge.b = rect.a;
        break;
    default:
        break;
    }
    return edge;
}

//矩形的面积
```

```
TYPE Area(const RECT & rect)
{
   return (rect.b.x - rect.a.x) * (rect.b.y - rect.a.y);
}

//两个矩形的公共面积
TYPE CommonArea(const RECT & A, const RECT & B)
{
   TYPE area = 0.0;
   POINT LL(Max(A.a.x, B.a.x), Max(A.a.y, B.a.y));
   POINT UR(Min(A.b.x, B.b.x), Min(A.b.y, B.b.y));
   if( (LL.x <= UR.x) && (LL.y <= UR.y) )
   {
      area = Area(RECT(LL, UR));
   }
   return area;
}
//判断圆是否在矩形内(不允许相切)
bool IsInCircle(const CIRCLE & circle, const RECT & rect)
{
   return (circle.x - circle.r > rect.a.x) &&
      (circle.x + circle.r < rect.b.x) &&
      (circle.y - circle.r > rect.a.y) &&
      (circle.y + circle.r < rect.b.y);
}

//判断矩形是否在圆内(不允许相切)
bool IsInRect(const CIRCLE & circle, const RECT & rect)
{
   POINT c,d;
   c.x=rect.a.x; c.y=rect.b.y;
   d.x=rect.b.x; d.y=rect.a.y;
   return (Distance( Center(circle) , rect.a ) < circle.r) &&
      (Distance( Center(circle) , rect.b ) < circle.r) &&
      (Distance( Center(circle) , c ) < circle.r) &&
      (Distance( Center(circle) , d ) < circle.r);
}

//判断矩形是否与圆相离(不允许相切)
bool Isoutside(const CIRCLE & circle, const RECT & rect)
{
   POINT c,d;
   c.x=rect.a.x; c.y=rect.b.y;
   d.x=rect.b.x; d.y=rect.a.y;
```

```
    return (Distance( Center(circle) , rect.a ) > circle.r) &&
        (Distance( Center(circle) , rect.b ) > circle.r) &&
        (Distance( Center(circle) , c ) > circle.r) &&
        (Distance( Center(circle) , d ) > circle.r) &&
        (rect.a.x > circle.x || circle.x > rect.b.x || rect.a.y > circle.y || circle.y > rect.b.y) ||
        ((circle.x - circle.r > rect.b.x) ||
        (circle.x + circle.r < rect.a.x) ||
        (circle.y - circle.r > rect.b.y) ||
        (circle.y + circle.r < rect.a.y));
}
```

1.15 四城部分几何模板

```
/*
1.注意实际运用的时候可以用 sqrd 代替 dist 提高精度，节省时间
*/
#include <iostream>
#include <math.h>
#include <algorithm>
using namespace std;

const double INF = 10e300;
const double EPS = 1e-8;
const double PI = acos(-1.0);

inline int dblcmp(double a, double b) {if(fabs(a-b) < EPS) return 0;if(a < b) return -1;return 1;}
inline double Max(double a, double b) { if(dblcmp(a, b) == 1) return a; return b; }
inline double Min(double a, double b) { if(dblcmp(a, b) == 1) return b; return a; }
inline double Agl(double deg) { return deg * PI / 180.0; }

struct Point { double x, y; void set(double a, double b) { x = a; y = b; } };
struct Vec { double x, y; void set(Point& a, Point& b) { x = b.x-a.x; y = b.y-a.y; } };
struct Line { double a, b, c; Point st, end;
void set(Point& u, Point& v) {a = v.y - u.y; b = u.x - v.x; c = a*u.x + b*u.y; st = u; end = v; } };

inline double dist(Point& a, Point& b) { return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y)); }
inline double sqrd(Point& a, Point& b) { return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y); }
inline double dot(Vec& a, Vec& b) { return a.x * b.x + a.y * b.y; }
inline double cross(Vec& a, Vec& b) { return a.x * b.y - a.y * b.x; }
inline double cross(Point& a, Point& b, Point& c) {Vec x, y; x.set(a, b); y.set(a, c); return cross(x, y); }
//返回 1 代表 a 在 bc 之间  0 代表在端点  -1 代表在外面
inline int between(Point& a, Point& b, Point& c) { Vec x, y; x.set(a,b); y.set(a,c); return dblcmp(dot(x,
y),0); }
```

//3 维坐标转换 输入是度数

```
void trans(double lat, double log, double& x, double& y, double& z, double radius) {
    x = radius * cos(lat) * cos(log);
    y = radius * cos(lat) * sin(log);
    z = radius * sin(lat);
}
```

//求两点的平分线

```
Line bisector(Point& a, Point& b) {
    Line ab, ans; ab.set(a, b);
    double midx = (a.x + b.x)/2.0, midy = (a.y + b.y)/2.0;
    ans.a = -ab.b, ans.b = -ab.a, ans.c = -ab.b * midx + ab.a * midy;
    return ans;
}
```

//线线相交 如果平行 返回-1, 重合返回-2

```
int line_line_intersect(Line& l1, Line& l2, Point& s) {
    double det = l1.a*l2.b - l2.a*l1.b;
     if(dblcmp(det, 0.0) == 0) { //平行或者重合
        if(dblcmp(point_line_dist(l1.st, l2.st, l2.end, 0), 0) == 0)
            return -2;
        return -1;
    }
     s.x = (l2.b*l1.c - l1.b*l2.c)/det;
     s.y = (l1.a*l2.c - l2.a*l1.c)/det;
    return 1;
}
```

//2 线段相交 ab, cd 交点是 s 平行返回-1, 重合返回-2, 不在线段上面返回 0 在线段中间返回 1 在线段两端返回 2

```
int seg_seg_intersect(Point& a, Point& b, Point& c, Point& d, Point& s) {
     Line l1, l2; l1.set(a, b); l2.set(c, d);
    int ans = line_line_intersect(l1, l2, s);
    if(ans == 1) {
        if(between(s, a, b) == 1 && between(s, c, d) == 1)
            return 1;
        if(between(s, a, b) == -1 && between(s, c, d) == -1)
            return 0;
        return 2;
    }
    return ans;
}
```

```
//求三点共圆 中心放在 center 中 返回半径
double center_3point(Point& a, Point& b, Point& c, Point& center) {
    Line x = bisector(a, b), y = bisector(b, c);
    line_line_intersect(x, y, center);
    return dist(center, a);
}
```

1.16 一些代码

1.16.1 最小圆覆盖_zju1450

```
/*
包含点集所有点的最小圆的算法
最小圆覆盖

 http://acm.zju.edu.cn/show_problem.php?pid=1450

 相关题目最小球包含  http://acm.pku.edu.cn/JudgeOnline/problem?id=2069

平面上有 n 个点，给定 n 个点的坐标，试找一个半径最小的圆，将 n
个点全部包围，点可以在圆上。

1. 在点集中任取 3 点 A,B,C。
2. 作一个包含 A,B,C 三点的最小圆,圆周可能通过这 3 点，也可能只通过
其中两点,但包含第 3 点.后一种情况圆周上的两点一定是位于圆的一条直
径的两端。
3. 在点集中找出距离第 2 步所建圆圆心最远的 D 点，若 D 点已在圆内或圆周上，
则该圆即为所求的圆，算法结束.则，执行第 4 步。
4. 在 A,B,C,D 中选 3 个点,使由它们生成的一个包含这 4 个点的圆为最小，这 3
点成为新的 A,B,C，返回执行第 2 步。若在第 4 步生成的圆的圆周只通过 A,B,C,D
中的两点，则圆周上的两点取成新的 A 和 B,从另两点中任取一点作为新的 C。

程序设计题解上的解题报告：
对于一个给定的点集 A，记 MinCircle(A)为点集 A 的最小外接圆，显然，对于所
有的点集情况 A,MinCircle(A)都是存在且惟一的。需要特别说明的是，当 A 为空
集时，MinCircle(A)为空集，当 A={a}时，MinCircle(A)圆心坐标为 a，半径为 0；
    显然，MinCircle(A)可以有 A 边界上最多三个点确定(当点集 A 中点的个数大于
1 时，有可能两个点确定了 MinCircle(A))，也就是说存在着一个点集 B，|B|<=3
且 B 包含与 A，有 MinCircle(B)=MinCircle(A).所以，如果 a 不属于 B，则
MinCircle(A-{a})=MinCircle(A);如果 MinCircle(A-{a})不等于 MinCircle(A),则
a 属于 B。
     所以我们可以从一个空集 R 开始，不断的把题目中给定的点集中的点加入 R，同
时维护 R 的外接圆最小，这样就可以得到解决该题的算法。
```

```
pku2069

*/
#include <stdio.h>
#include <math.h>

const int maxn = 1005;
//const double eps = 1e-6;

struct TPoint
{
    double x, y;
    TPoint operator-(TPoint &a)
    {
        TPoint p1;
        p1.x = x - a.x;
        p1.y = y - a.y;
        return p1;
    }
};

struct TCircle
{
    double r;
    TPoint centre;
};

struct TTriangle
{
    TPoint t[3];
};

TCircle c;
TPoint a[maxn];

double distance(TPoint p1, TPoint p2)
{
    TPoint p3;
    p3.x = p2.x - p1.x;
    p3.y = p2.y - p1.y;
    return sqrt(p3.x * p3.x + p3.y * p3.y);
}
```

```
double triangleArea(TTriangle t)
{
    TPoint p1, p2;
    p1 = t.t[1] - t.t[0];
    p2 = t.t[2] - t.t[0];
    return fabs(p1.x * p2.y - p1.y * p2.x) / 2;
}

TCircle circumcircleOfTriangle(TTriangle t)
{
    //三角形的外接圆
    TCircle tmp;
    double a, b, c, c1, c2;
    double xA, yA, xB, yB, xC, yC;
    a = distance(t.t[0], t.t[1]);
    b = distance(t.t[1], t.t[2]);
    c = distance(t.t[2], t.t[0]);
    //根据 S = a * b * c / R / 4;求半径 R
    tmp.r = a * b * c / triangleArea(t) / 4;

    xA = t.t[0].x; yA = t.t[0].y;
    xB = t.t[1].x; yB = t.t[1].y;
    xC = t.t[2].x; yC = t.t[2].y;
    c1 = (xA * xA + yA * yA - xB * xB - yB * yB) / 2;
    c2 = (xA * xA + yA * yA - xC * xC - yC * yC) / 2;

    tmp.centre.x = (c1 * (yA - yC) - c2 * (yA - yB)) /
            ((xA - xB) * (yA - yC) - (xA - xC) * (yA - yB));
    tmp.centre.y = (c1 * (xA - xC) - c2 * (xA - xB)) /
            ((yA - yB) * (xA - xC) - (yA - yC) * (xA - xB));

    return tmp;
}

TCircle MinCircle2(int tce, TTriangle ce)
{
    TCircle tmp;
    if(tce == 0) tmp.r = -2;
    else if(tce == 1)
    {
        tmp.centre = ce.t[0];
        tmp.r = 0;
    }
    else if(tce == 2)
```

```
    {
        tmp.r = distance(ce.t[0], ce.t[1]) / 2;
        tmp.centre.x = (ce.t[0].x + ce.t[1].x) / 2;
        tmp.centre.y = (ce.t[0].y + ce.t[1].y) / 2;
    }
    else if(tce == 3) tmp = circumcircleOfTriangle(ce);
    return tmp;
}

void MinCircle(int t, int tce, TTriangle ce)
{
    int i, j;
    TPoint tmp;
    c = MinCircle2(tce, ce);
    if(tce == 3) return;
    for(i = 1;i <= t;i++)
    {
        if(distance(a[i], c.centre) > c.r)
        {
            ce.t[tce] = a[i];
            MinCircle(i - 1, tce + 1, ce);
            tmp = a[i];
            for(j = i;j >= 2;j--)
            {
                a[j] = a[j - 1];
            }
            a[1] = tmp;
        }
    }
}

void run(int n)
{
    TTriangle ce;
    int i;
    MinCircle(n, 0, ce);
    printf("%.2lf %.2lf %.2lf\n", c.centre.x, c.centre.y, c.r);
}

int main()
{
     freopen("circle.in", "r", stdin);
     freopen("out.txt", "w", stdout);
     int n;
```

```
    while(scanf("%d", &n) != EOF && n)
    {
        for(int i = 1;i <= n;i++)
            scanf("%lf%lf", &a[i].x, &a[i].y);
        run(n);
    }
    return 0;
}
```

1.16.2 直线旋转_两凸包的最短距离(poj3608)

```
#include <stdio.h>
#include <math.h>

#define pi acos(-1.0)
#define eps 1e-6
#define inf 1e250
#define Maxn 10005

typedef struct TPoint
{
    double x, y;
}TPoint;

typedef struct TPolygon
{
    TPoint p[Maxn];
    int n;
}TPolygon;

typedef struct TLine
{
    double a, b, c;
}TLine;

double max(double a, double b)
{
    if(a > b) return a;
    return b;
}

double min(double a, double b)
{
```

```
    if(a < b) return a;
    return b;
}

double distance(TPoint p1, TPoint p2)
{
    return sqrt((p1.x - p2.x) * (p1.x - p2.x)
     + (p1.y - p2.y) * (p1.y - p2.y));
}

TLine lineFromSegment(TPoint p1, TPoint p2)
{
     TLine tmp;
     tmp.a = p2.y - p1.y;
     tmp.b = p1.x - p2.x;
     tmp.c = p2.x * p1.y - p1.x * p2.y;
     return tmp;
}

double polygonArea(TPolygon p)
{
     int i, n;
     double area;
     n = p.n;
     area = 0;
     for(i = 1;i <= n;i++)
          area += (p.p[i - 1].x * p.p[i % n].y - p.p[i % n].x * p.p[i - 1].y);

    return area / 2;
}

void ChangeClockwise(TPolygon &polygon)
{
    TPoint tmp;
    int i;
    for(i = 0;i <= (polygon.n - 1) / 2;i++)
    {
         tmp = polygon.p[i];
         polygon.p[i] = polygon.p[polygon.n - 1 - i];
         polygon.p[polygon.n - 1 - i] = tmp;
    }
}

double disPointToSeg(TPoint p1, TPoint p2, TPoint p3)
```

```
{
    double a = distance(p1, p2);
    double b = distance(p1, p3);
    double c = distance(p2, p3);
    if(fabs(a + b - c) < eps) return 0;
    if(fabs(a + c - b) < eps || fabs(b + c - a) < eps) return min(a, b);
    double t1 = -a * a + b * b + c * c;
    double t2 = a * a - b * b + c * c;
    if(t1 <= 0 || t2 <= 0) return min(a, b);

    TLine l1 = lineFromSegment(p2, p3);
    return fabs(l1.a * p1.x + l1.b * p1.y + l1.c) / sqrt(l1.a * l1.a + l1.b * l1.b);
}

double disPallSeg(TPoint p1, TPoint p2, TPoint p3, TPoint p4)
{
    return min(min(disPointToSeg(p1, p3, p4), disPointToSeg(p2, p3, p4)),
      min(disPointToSeg(p3, p1, p2), disPointToSeg(p4, p1, p2)));
}

double angle(TPoint p1, TPoint p2, double SlewRate)
{
    double ang, tmp;
    TPoint p;
    p.x = p2.x - p1.x;
    p.y = p2.y - p1.y;
    if(fabs(p.x) < eps)
    {
        if(p.y > 0) ang = pi / 2;
        else ang = 3 * pi / 2;
    }
    else
    {
         ang = atan(p.y / p.x);
        if(p.x < 0) ang += pi;
    }
    while(ang < 0) ang += 2 * pi;
    if(ang >= pi) SlewRate += pi;
    if(ang > SlewRate) tmp = ang - SlewRate;
    else tmp = pi - (SlewRate - ang);
    while(tmp >= pi) tmp -= pi;
    if(fabs(tmp - pi) < eps) tmp = 0;
    return tmp;
}
```

```
int main()
{
    int n, m, i;
    TPolygon polygon1, polygon2;
    double ymin1, ymax2, ans, d;
    int k1, k2;
    while(scanf("%d%d", &n, &m) && n)
    {
        polygon1.n = n;
        polygon2.n = m;
        for(i = 0;i < n;i++)
            scanf("%lf%lf", &polygon1.p[i].x, &polygon1.p[i].y);
        for(i = 0;i < m;i++)
            scanf("%lf%lf", &polygon2.p[i].x, &polygon2.p[i].y);
        if(polygonArea(polygon1) < 0) ChangeClockwise(polygon1);
        if(polygonArea(polygon2) < 0) ChangeClockwise(polygon2);
        ymin1 = inf, ymax2 = -inf;
        for(i = 0;i < n;i++)
            if(polygon1.p[i].y < ymin1) ymin1 = polygon1.p[i].y , k1 = i;
        for(i = 0;i < m;i++)
            if(polygon2.p[i].y > ymax2) ymax2 = polygon2.p[i].y , k2 = i;
        double SlewRate = 0;
        double angle1, angle2;
        ans = inf;
        double Slope = 0;
        while(Slope <= 360)
        {
            while(SlewRate >= pi) SlewRate -= pi;
            if(fabs(pi - SlewRate) < eps) SlewRate = 0;
            angle1 = angle(polygon1.p[k1], polygon1.p[(k1 + 1) % n], SlewRate);
            angle2 = angle(polygon2.p[k2], polygon2.p[(k2 + 1) % m], SlewRate);
            if(fabs(angle1 - angle2) < eps)
            {
                d = disPallSeg(polygon1.p[k1], polygon1.p[(k1 + 1) % n], polygon2.p[k2], polygon2.p[(k2
+ 1) % m]);
                if(d < ans) ans = d;
                  k1++;
                k1 %= n;
                k2++;
                k2 %= m;
                SlewRate += angle1;
                Slope += angle1;
            }
```

```
            else if(angle1 < angle2)
            {
                d = disPointToSeg(polygon2.p[k2], polygon1.p[k1], polygon1.p[(k1 + 1) % n]);
                if(d < ans) ans = d;
                k1++;
                k1 %= n;
                SlewRate += angle1;
                Slope += angle1;
            }
            else
            {
                d = disPointToSeg(polygon1.p[k1], polygon2.p[k2], polygon2.p[(k2 + 1) % m]);
                if(d < ans) ans = d;
                k2++;
                k2 %= m;
                SlewRate += angle2;
                Slope += angle2;
            }
        }
        printf("%.5lf\n", ans);
    }
    return 0;
}
```

1.16.3 扇形的重心

```
//Xc = 2*R*sinA/3/A
//A 为圆心角的一半
#include <stdio.h>
#include <math.h>
int main()
{
    double r, angle;
    while(scanf("%lf%lf", &r, &angle) != EOF){
        angle /= 2;
        printf("%.6lf\n", 2 * r * sin(angle) / 3 / angle);
    }
    return 0;
}
```

1.16.4 根据经度纬度求球面距离

```
/*
假设地球是球体，
设地球上某点的经度为 lambda,纬度为 phi，
则这点的空间坐标是
x=cos(phi)*cos(lambda)
y=cos(phi)*sin(lambda)
z=sin(phi)
设地球上两点的空间坐标分别为(x1,y1,z1),(x2,y2,z2)
直线距离即为 R*sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1)+(z2-z1)*(z2-z1)),
则它们的夹角为
A = acos(x1 * x2 + y1 * y2 + z1 * z2)，
则两地距离为  A * R，其中 R 为地球平均半径 6371
*/

/*
这里坐标都要乘以半径 R，但由于是求角度，所以统一都没有乘
注意这里还要判断坐标的正负和经度纬度的规定有关

pku_3407
*/
#include <stdio.h>
#include <math.h>

const double pi = acos(-1.0);

struct TPoint
{
    double x, y, z;
};

int main()
{
      double w1, wm1, j1, jm1, wd1, wd2;
      double w2, wm2, j2, jm2, jd1, jd2;
      TPoint p1, p2;
      char chr1, chr2;
      while(scanf("%lf%lf ", &w1, &wm1) != EOF){
            scanf("%c ", &chr1);
            scanf("%lf %lf %c", &j1, &jm1, &chr2);
            wd1 = (w1 + wm1 / 60) * pi / 180;
            jd1 = (j1 + jm1 / 60) * pi / 180;
```

```
        if(chr1 == 'S') wd1 *= -1.0;
        if(chr2 == 'W') jd1 *= -1.0;
        p1.x = cos(wd1) * cos(jd1);
        p1.y = cos(wd1) * sin(jd1);
        p1.z = sin(wd1);
        scanf("%lf %lf %c %lf %lf %c", &w2, &wm2, &chr1, &j2, &jm2, &chr2);
        wd2 = (w2 + wm2 / 60) * pi / 180;
        jd2 = (j2 + jm2 / 60) * pi / 180;
        if(chr1 == 'S') wd2 *= -1.0;
        if(chr2 == 'W') jd2 *= -1.0;
        p2.x = cos(wd2) * cos(jd2);
        p2.y = cos(wd2) * sin(jd2);
        p2.z = sin(wd2);
        double a = acos(p1.x * p2.x + p1.y * p2.y + p1.z * p2.z);
        printf("%.3lf\n", a * 6370.0);
    }
    return 0;
}
```

1.16.5 多边形的重心

```
/*
题目描述：
有一个密度均匀的平面 N 多边形(3 <= N <= 1000000)，可能凹也可能凸，但没有边相交叉，
另外已知 N 个有序(顺时针或逆时针)顶点的坐标值，第 j 个顶点坐标为（Xi ， Yi ），且满
足 (|Xi|, |Yi| <= 20000)，求这个平面多边形的重心。

  解题过程：
  从第 1 个顶点出发，分别连接第 i, i+1 个顶点组成三角形 Ti,1 < i < n,
  一共 n-2 个三角形正好是多连形的一个划分，分别求出每个三角形的面积 Si，
  总面积为各个面积相加
  根据物理学知识得：n 个点(xi,yi)每个重量是 mi,则重心是
  X = (x1*M1+x2*M2+...+xn*Mn) / (M1+M2+....+Mn)
  Y = (y1*M1+y2*M2+...+yn*Mn) / (M1+M2+....+Mn)
  另个需要用的知识有：
  已知 3 点求三角形的面积，设三点分别为 p[0].x, p[0].y p[1].x, p[1].y p[1].x, p[1].y
  面积 s =[ p[0].x*p[1].y-p[1].x*p[0].y + p[1].x*p[2].y-p[2].x*p[1].y + p[2].x*p[0].y-p[0].x*p[2].y ] / 2 ,
  这是这 3 个点是逆时针的值，顺时针取负。
  已知 3 点求重心，x = (p[0].x+p[1].x+p[2].x)/3.0 , y = (p[0].y+p[1].y+p[2].y)/3.0
  另外在求解的过程中，不需要考虑点的输入顺序是顺时针还是逆时针，相除后就抵消了，
  还要注意 一点是不必在求每个小三角形的重心时都除以 3，可以在最后除一下

*/
```

```
/*fzu_1132*/
#include <stdio.h>
#include <math.h>

typedef struct TPoint
{
	double x;
	double y;
}TPoint;

double triangleArea(TPoint p0, TPoint p1, TPoint p2)
{
	//已知三角形三个顶点的坐标，求三角形的面积
	double k = p0.x * p1.y + p1.x * p2.y
		+ p2.x * p0.y - p1.x * p0.y
		- p2.x * p1.y - p0.x * p2.y;
	//if(k >= 0) return k / 2;
//	else return -k / 2;
		return k / 2;
}

int main()
{
	int i, n, test;
	TPoint p0, p1, p2, center;
	double area, sumarea, sumx, sumy;
	scanf("%d", &test);
	while(test--){
		scanf("%d", &n);
		scanf("%lf%lf", &p0.x, &p0.y);
		scanf("%lf%lf", &p1.x, &p1.y);
		  sumx = 0;
		  sumy = 0;
		  sumarea = 0;
		  for(i = 2;i < n;i++){
			scanf("%lf%lf", &p2.x, &p2.y);
			  center.x = p0.x + p1.x + p2.x;
			  center.y = p0.y + p1.y + p2.y;
			  area =    triangleArea(p0, p1, p2);
			  sumarea += area;
			sumx += center.x * area;
			sumy += center.y * area;
			p1 = p2;
		  }
```

```
            printf("%.2lf %.2lf\n", sumx / sumarea / 3, sumy / sumarea / 3);
        }
        return 0;
}
```

1.16.6 存不存在一个平面把两堆点分开(poj3643)

```
#include <stdio.h>
struct point
{
    double x, y, z;
}pa[201], pb[201];
int main()
{
    int n, m, i;
    while (scanf("%d", &n), n != -1)
    {
        for (i = 0; i < n; i++)
            scanf("%lf%lf%lf", &pa[i].x, &pa[i].y, &pa[i].z);
        scanf("%d", &m);
        for (i = 0; i < m; i++)
            scanf("%lf%lf%lf", &pb[i].x, &pb[i].y, &pb[i].z);
        int cnt = 0, finish = 0;
        double a = 0, b = 0, c = 0, d = 0;
        while (cnt < 100000 && !finish)
        {
            finish = 1;
            for (i = 0; i < n; i++)
                if (a * pa[i].x + b * pa[i].y + c * pa[i].z + d > 0)
                {
                    a -= pa[i].x;
                    b -= pa[i].y;
                    c -= pa[i].z;
                    d -= 3;
                    finish = 0;
                }
            for (i = 0; i < m; i++)
                if (a * pb[i].x + b * pb[i].y + c * pb[i].z + d <= 0)
                {
                    a += pb[i].x;
                    b += pb[i].y;
                    c += pb[i].z;
                    d += 3;
                    finish = 0;
```

```
                    }
                cnt++;
            }
            printf("%lf %lf %lf %lf\n", a, b, c, d);
        }
        return 0;
}
```

1.16.7 pku_3335_判断多边形的核是否存在

```
/*多边形的核*/
#include <stdio.h>
#include <math.h>

#define Maxn 3005
const double eps = 1e-10;

typedef struct TPodouble
{
    double x;
    double y;
}TPoint;

typedef struct TPolygon
{
    TPoint p[Maxn];
    int n;
};

typedef struct TLine
{
    double a, b, c;
}TLine;

bool same(TPoint p1, TPoint p2)
{
    if(p1.x != p2.x) return false;
    if(p1.y != p2.y) return false;
    return true;
}

double multi(TPoint p1, TPoint p2, TPoint p0)
```

```
{
    //求矢量[p0, p1], [p0, p2]的叉积
    //p0 是顶点
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
    //若结果等于 0，则这三点共线
    //若结果大于 0，则 p0p2 在 p0p1 的逆时针方向
    //若结果小于 0，则 p0p2 在 p0p1 的顺时针方向
}

TLine lineFromSegment(TPoint p1, TPoint p2)
{
    //线段所在直线,返回直线方程的三个系统
    TLine tmp;
    tmp.a = p2.y - p1.y;
    tmp.b = p1.x - p2.x;
    tmp.c = p2.x * p1.y - p1.x * p2.y;
    return tmp;
}

TPoint LineInter(TLine l1, TLine l2)
{
    //求两直线得交点坐标
    TPoint tmp;
    double a1 = l1.a;
    double b1 = l1.b;
    double c1 = l1.c;
    double a2 = l2.a;
    double b2 = l2.b;
    double c2 = l2.c;
    //注意这里 b1 = 0
    if(fabs(b1) < eps){
        tmp.x = -c1 / a1;
        tmp.y = (-c2 - a2 * tmp.x) / b2;
    }
    else{
        tmp.x = (c1 * b2 - b1 * c2) / (b1 * a2 - b2 * a1);
        tmp.y = (-c1 - a1 * tmp.x) / b1;
    }
   return tmp;
}

TPolygon Cut_polygon(TPoint p1, TPoint p2, TPolygon polygon)
{
    TPolygon new_polygon;
```

```
    TPoint interp;
    TLine l1, l2;
    int i, j;
    double t1, t2;
    new_polygon.n = 0;
    for(i = 0;i <= polygon.n - 1;i++){
        t1 = multi(p2, polygon.p[i], p1);
        t2 = multi(p2, polygon.p[i + 1], p1);
        if(fabs(t1) < eps || fabs(t2) < eps){
            if(fabs(t1) < eps) new_polygon.p[new_polygon.n++] = polygon.p[i];
            if(fabs(t2) < eps) new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
        }
        else if(t1 < 0 && t2 < 0){
            new_polygon.p[new_polygon.n++] = polygon.p[i];
            new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
        }
        else if(t1 * t2    < 0){
            l1 = lineFromSegment(p1, p2);
            l2 = lineFromSegment(polygon.p[i], polygon.p[i + 1]);
            interp = LineInter(l1, l2);
            if(t1 < 0) {
                new_polygon.p[new_polygon.n++] = polygon.p[i];
                new_polygon.p[new_polygon.n++] = interp;
            }
            else {
                new_polygon.p[new_polygon.n++] = interp;
                new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
            }
        }
    }
    polygon.n = 0;
    if(new_polygon.n == 0) return polygon;
    polygon.p[polygon.n++] = new_polygon.p[0];
    for(i = 1;i < new_polygon.n;i++){
        if(!same(new_polygon.p[i], new_polygon.p[i - 1])){
            polygon.p[polygon.n++] = new_polygon.p[i];
        }
    }
    if(polygon.n != 1 && same(polygon.p[polygon.n - 1], polygon.p[0])) polygon.n--;
    polygon.p[polygon.n] = polygon.p[0];
    return polygon;
}

double polygonArea(TPolygon p)
```

```
{
    //已知多边形各顶点的坐标，求其面积
    int i, n;
    double area;
    n = p.n;
    area = 0;
    for(i = 1;i <= n;i++){
        area += (p.p[i - 1].x * p.p[i % n].y - p.p[i % n].x * p.p[i - 1].y);
    }
    return area / 2;
}

void ChangeClockwise(TPolygon &polygon)
{
    TPoint tmp;
    int i;
    for(i = 0;i <= (polygon.n - 1) / 2;i++){
        tmp = polygon.p[i];
        polygon.p[i] = polygon.p[polygon.n - 1 - i];
        polygon.p[polygon.n - 1 - i] = tmp;
    }
}

int main()
{
    int test, i, j;
    double area;
    TPolygon polygon, new_polygon;
    scanf("%d", &test);
    while(test--){
        scanf("%d", &polygon.n);
        for(i = 0;i <= polygon.n - 1;i++){
            scanf("%lf%lf", &polygon.p[i].x, &polygon.p[i].y);
        }
        /*若是逆时针转化为顺时针*/
        if(polygonArea(polygon) > 0) ChangeClockwise(polygon);
        polygon.p[polygon.n] = polygon.p[0];
        new_polygon = polygon;
        for(i = 0;i <= polygon.n - 1;i++){
            new_polygon = Cut_polygon(polygon.p[i], polygon.p[i + 1], new_polygon);
        }
        area = polygonArea(new_polygon);
        if(area < 0) printf("%.2lf\n", -area);
        else printf("%.2lf\n", area);
```

```
    }
    return 0;
}

//是否存在

#include <stdio.h>
#include <math.h>

#define Maxn 3005
const double eps = 1e-10;

typedef struct TPodouble
{
    double x;
    double y;
}TPoint;

typedef struct TPolygon
{
    TPoint p[Maxn];
    int n;
};

typedef struct TLine
{
    double a, b, c;
}TLine;

bool same(TPoint p1, TPoint p2)
{
    if(p1.x != p2.x) return false;
    if(p1.y != p2.y) return false;
    return true;
}

double multi(TPoint p1, TPoint p2, TPoint p0)
{
     //求矢量[p0, p1], [p0, p2]的叉积
     //p0 是顶点
     return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
     //若结果等于 0，则这三点共线
     //若结果大于 0，则 p0p2 在 p0p1 的逆时针方向
     //若结果小于 0，则 p0p2 在 p0p1 的顺时针方向
```

```
}

TLine lineFromSegment(TPoint p1, TPoint p2)
{
	//线段所在直线,返回直线方程的三个系统
	TLine tmp;
	tmp.a = p2.y - p1.y;
	tmp.b = p1.x - p2.x;
	tmp.c = p2.x * p1.y - p1.x * p2.y;
	return tmp;
}

TPoint LineInter(TLine l1, TLine l2)
{
	//求两直线得交点坐标
	TPoint tmp;
	double a1 = l1.a;
	double b1 = l1.b;
	double c1 = l1.c;
	double a2 = l2.a;
	double b2 = l2.b;
	double c2 = l2.c;
	//注意这里 b1 = 0
	if(fabs(b1) < eps){
		tmp.x = -c1 / a1;
		tmp.y = (-c2 - a2 * tmp.x) / b2;
	}
	else{
		tmp.x = (c1 * b2 - b1 * c2) / (b1 * a2 - b2 * a1);
		tmp.y = (-c1 - a1 * tmp.x) / b1;
	}
	return tmp;
}

TPolygon Cut_polygon(TPoint p1, TPoint p2, TPolygon polygon)
{
	TPolygon new_polygon;
	TPoint interp;
	TLine l1, l2;
	int i, j;
	double t1, t2;
	new_polygon.n = 0;
	for(i = 0;i <= polygon.n - 1;i++){
		t1 = multi(p2, polygon.p[i], p1);
```

```
            t2 = multi(p2, polygon.p[i + 1], p1);
            if(fabs(t1) < eps || fabs(t2) < eps){
                if(fabs(t1) < eps) new_polygon.p[new_polygon.n++] = polygon.p[i];
                if(fabs(t2) < eps) new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
            }
            else if(t1 < 0 && t2 < 0){
                new_polygon.p[new_polygon.n++] = polygon.p[i];
                new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
            }
            else if(t1 * t2    < 0){
                l1 = lineFromSegment(p1, p2);
                l2 = lineFromSegment(polygon.p[i], polygon.p[i + 1]);
                interp = LineInter(l1, l2);
                if(t1 < 0) {
                    new_polygon.p[new_polygon.n++] = polygon.p[i];
                    new_polygon.p[new_polygon.n++] = interp;
                }
                else {
                    new_polygon.p[new_polygon.n++] = interp;
                    new_polygon.p[new_polygon.n++] = polygon.p[i + 1];
                }
            }
        }
        polygon.n = 0;
        if(new_polygon.n == 0) return polygon;
        polygon.p[polygon.n++] = new_polygon.p[0];
        for(i = 1;i < new_polygon.n;i++){
            if(!same(new_polygon.p[i], new_polygon.p[i - 1])){
                polygon.p[polygon.n++] = new_polygon.p[i];
            }
        }
        if(polygon.n != 1 && same(polygon.p[polygon.n - 1], polygon.p[0])) polygon.n--;
        polygon.p[polygon.n] = polygon.p[0];
        return polygon;
}

void ChangeClockwise(TPolygon &polygon)
{
        TPoint tmp;
        int i;
        for(i = 0;i <= (polygon.n - 1) / 2;i++){
            tmp = polygon.p[i];
            polygon.p[i] = polygon.p[polygon.n - 1 - i];
            polygon.p[polygon.n - 1 - i] = tmp;
```

```
    }
}

double polygonArea(TPolygon p)
{
    //已知多边形各顶点的坐标，求其面积
    double area;
    int i, n;
    n = p.n;
    area = 0;
    for(i = 1;i <= n;i++){
        area += (p.p[i - 1].x * p.p[i % n].y - p.p[i % n].x * p.p[i - 1].y);
    }
    return area / 2;
}

int main()
{
    int i, j;
    TPolygon polygon, new_polygon;
    while(scanf("%d", &polygon.n) && polygon.n){
        for(i = 0;i <= polygon.n - 1;i++){
            scanf("%lf%lf", &polygon.p[i].x, &polygon.p[i].y);
        }
        /*若是逆时针转化为顺时针*/
        if(polygonArea(polygon) > 0) ChangeClockwise(polygon);
        polygon.p[polygon.n] = polygon.p[0];
        new_polygon = polygon;
        for(i = 0;i <= polygon.n - 1;i++){
            new_polygon = Cut_polygon(polygon.p[i], polygon.p[i + 1], new_polygon);
        }
        if(new_polygon.n > 0) printf("1\n");
        else printf("0\n");
    }
    return 0;
}
```

1.16.8 pku_2600_二分+圆的参数方程

```
#include <stdio.h>
#include <math.h>

const double eps = 1e-4;
```

```
const double pi = acos(-1.0);

struct TPoint
{
    double x, y;
}p[60], a[60];
double angle[60];

double multi(TPoint p1, TPoint p2, TPoint p0)
{
     return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}

TPoint fine_a2(TPoint a1, TPoint m, double angle1)
{
    TPoint a2;
    double r, angle2, angle3;
    r = sqrt((a1.x - m.x) * (a1.x - m.x) + (a1.y - m.y) * (a1.y - m.y));
    angle2 = acos((a1.x - m.x) / r);
    if(a1.y < m.y) {
        if(angle2 <= pi / 2) angle2 = -angle2;
        if(angle2 > pi / 2) angle2 = 3 * pi / 2 - (angle2 - pi / 2);
    }
    angle3 = angle2 - angle1;
    a2.x = m.x + r * cos(angle3);
    a2.y = m.y + r * sin(angle3);
    if(multi(m, a2, a1) < 0) return a2;
     angle3 = angle2 + angle1;
    a2.x = m.x + r * cos(angle3);
    a2.y = m.y + r * sin(angle3);
    if(multi(m, a2, a1) < 0) return a2;
}

int main()
{
    int n, i, j;
    while(scanf("%d", &n) != EOF){
        for(i = 0;i < n;i++){
            scanf("%lf%lf", &p[i].x, &p[i].y);
        }
        for(i = 0;i < n;i++){
            scanf("%lf", &angle[i]);
            angle[i] = angle[i] * pi / 180;
        }
```

```
        a[0].x = 0;
        a[0].y = 0;
        while(1){
            for(i = 1;i <= n;i++){
                a[i] = fine_a2(a[i - 1], p[i - 1], angle[i - 1]);
            }
            if(fabs(a[n].x - a[0].x) <= eps
               && fabs(a[n].y - a[0].y) <= eps) break;
            else {
                a[0].x = (a[0].x + a[n].x) / 2;
                a[0].y = (a[0].y + a[n].y) / 2;
            }
        }
        for(i = 0;i < n;i++){
            printf("%.0lf %.0lf\n", a[i].x, a[i].y);
        }
    }
    return 0;
}
```

1.16.9 pku_1151_矩形相交的面积

```
/*
大牛的思想
题目给出 n 个矩形，要求它们的面积并。具体做法是离散化。
先把 2n 个 x 坐标排序去重，然后再把所有水平线段（
要记录是矩形上边还是下边）按 y 坐标排序。
最后对于每一小段区间 (x[i], x[i + 1]) 扫描所有的水平线段，
求出这些水平线段在小区间内覆盖的面积。总的时间复杂度是 O(n^2)。
利用线段树，可以优化到 O(nlogn)。
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define up    1
#define down -1


typedef struct TSeg
{
    double l, r;
    double y;
```

```
    int UpOrDown;
}TSeg;
TSeg seg[210];
int segn;
double x[210];
int xn;

int cmp1(const void *a, const void *b)
{
    if(*(double *)a < *(double *)b) return -1;
    else return 1;
}

int cmp2(const void *a, const void *b)
{
    TSeg *c = (TSeg *)a;
    TSeg *d = (TSeg *)b;
    if(c->y < d->y) return -1;
    else return 1;
}

void movex(int t, int &xn)
{
    int i;
    for(i = t;i <= xn - 1;i++){
        x[i] = x[i + 1];
    }
    xn--;
}

int main()
{
    //freopen("in.in", "r", stdin);
    //freopen("out.out", "w", stdout);
    int n, i, j, cnt, test = 1;
    double x1, y1, x2, y2, ylow, area;
    while(scanf("%d", &n) != EOF && n){
        xn = 0;
        segn = 0;
        for(i = 0;i < n;i++){
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            x[xn++] = x1;
            x[xn++] = x2;
            seg[segn].l = x1;
```

```
            seg[segn].r = x2;
            seg[segn].y = y1;
            seg[segn++].UpOrDown = up;
            seg[segn].l = x1;
            seg[segn].r = x2;
            seg[segn].y = y2;
            seg[segn++].UpOrDown = down;
        }
        qsort(x, xn, sizeof(x[0]), cmp1);
        /*除掉重复的 x*/
        for(i = 1;i < xn;){
            if(x[i] == x[i - 1]) movex(i, xn);
            else i++;
        }
        qsort(seg, segn, sizeof(seg[0]), cmp2);
        area = 0.0;
        for(i = 0;i < xn - 1;i++){
            cnt = 0;
            for(j = 0;j < segn;j++){
                if(seg[j].l <= x[i] && seg[j].r >= x[i + 1]){
                    if(cnt == 0) ylow = seg[j].y;
                    if(seg[j].UpOrDown == down) cnt++;
                    else cnt--;
                    if(cnt == 0) area += (x[i + 1] - x[i]) * (seg[j].y - ylow);
                }
            }
        }
        printf("Test case #%d\n", test++);
          printf("Total explored area: %.2lf\n", area);
    }
    return 0;
}
```

1.16.10 pku_1118_共线最多的点的个数

```
/*
2617120 chenhaifeng 1118 Accepted 512K 1890MS C++ 977B 2007-09-04 18:43:26
直接 O(n^3)超时，用一个标记数组，标记 i,j 所做直线已经查找过，可以跳过

大牛的思想
朴素做法是 O(n3) 的，超时。我的做法是枚举每个点，
然后求其它点和它连线的斜率，再排序。这样就得到经过
该点的直线最多能经过几个点。求个最大值就行了。复
杂度是 O(n2logn) 的。把排序换成 hash，
```

可以优化到 O(n2)。
2617134 chenhaifeng 1118 Accepted 276K 312MS G++ 1394B 2007-09-04 18:49:08
*/

```
#include <stdio.h>
#include <math.h>

bool f[705][705];
int a[705];

int main()
{
    int n, i, j, s, num, maxn;
    int x[705], y[705];
    int t, m;


    while(scanf("%d", &n) != EOF && n){
        for(i = 0;i <= n - 1;i++){
            scanf("%d%d", &x[i], &y[i]);
        }
        maxn = -1;
        for(i = 0;i <= n - 1;i++){
            for(j = i;j <= n - 1;j++){
                f[i][j] = false;
            }
        }
        for(i = 0;i <= n - 1;i++){
            for(j = i + 1;j <= n - 1;j++){
                if(f[i][j] == true) continue;
                if(n - j < maxn) break;
                num = 2;
                t = 2;
                a[0] = i;
                a[1] = j;
                f[i][j] = true;
                for(s = j + 1;s <= n - 1;s++){
                    if(f[i][s] == true || f[j][s] == true) continue;
                    if((y[i] - y[s]) * (x[j] - x[s]) == (x[i] - x[s]) * (y[j] - y[s])){
                        num++;
                        a[t] = s;
                        for(m = 0;m <= t - 1;m++){
                            f[m][s] = true;
                        }
                        t++;
```

```
                        }
                    }
                    if(num > maxn) maxn = num;
                }
            }
            printf("%d\n", maxn);
        }
        return 0;
}
```

1.16.11 pku2826_线段围成的区域可储水量

```
/*
两条线不相交，
左边或右边的口被遮住，
交点是某条线的那个纵坐标较高的那点
某条线段水平放置
*/
#include <stdio.h>
#include <math.h>

#define eps 1e-8

struct TPoint
{
    double x, y;
};
struct TLine
{
     double a, b, c;
};

int same(TPoint p1, TPoint p2)
{
    if(fabs(p1.x - p2.x) > eps) return 0;
    if(fabs(p1.y - p2.y) > eps) return 0;
    return 1;
}

double min(double x, double y)
{
     if(x < y) return x;
     else return y;
```

```
}

double max(double x, double y)
{
	if(x > y) return x;
	else return y;
}

double multi(TPoint p1, TPoint p2, TPoint p0)
{
	return (p1.x - p0.x) * (p2.y - p0.y)
		- (p2.x - p0.x) * (p1.y - p0.y);
}

bool isIntersected(TPoint s1, TPoint e1, TPoint s2, TPoint e2)
{
	if(
	(max(s1.x, e1.x) >= min(s2.x, e2.x)) &&
	(max(s2.x, e2.x) >= min(s1.x, e1.x)) &&
	(max(s1.y, e1.y) >= min(s2.y, e2.y)) &&
	(max(s2.y, e2.y) >= min(s1.y, e1.y)) &&
	(multi(s2, e1, s1) * multi(e1, e2, s1) >= 0) &&
	(multi(s1, e2, s2) * multi(e2, e1, s2) >= 0)
	)	return true;

	return false;
}

TLine lineFromSegment(TPoint p1, TPoint p2)
{
	TLine tmp;
	tmp.a = p2.y - p1.y;
	tmp.b = p1.x - p2.x;
	tmp.c = p2.x * p1.y - p1.x * p2.y;
	return tmp;
}

TPoint LineInter(TLine l1, TLine l2)
{
	TPoint tmp;
	double a1 = l1.a;
	double b1 = l1.b;
	double c1 = l1.c;
	double a2 = l2.a;
```

```
    double b2 = l2.b;
    double c2 = l2.c;
    if(fabs(b1) < eps){
        tmp.x = -c1 / a1;
        tmp.y = (-c2 - a2 * tmp.x) / b2;
    }
    else{
        tmp.x = (c1 * b2 - b1 * c2) / (b1 * a2 - b2 * a1);
        tmp.y = (-c1 - a1 * tmp.x) / b1;
    }
    return tmp;
}

double triangleArea(TPoint p1, TPoint p2, TPoint p3)
{
    TPoint p4, p5;
    p4.x = p2.x - p1.x;
    p4.y = p2.y - p1.y;
    p5.x = p3.x - p1.x;
    p5.y = p3.y - p1.y;
    return fabs(p5.x * p4.y - p5.y * p4.x) / 2;
}

double find_x(double y, TLine line)
{
    return (-line.c - line.b * y) / line.a;
}

double find_y(double x, TLine line)
{
    if(fabs(line.b) < eps)
    {
        return -1e250;
    }
    else
    {
        return (-line.c - line.a    * x) / line.b;
    }
}

int main()
{
    //freopen("in.in", "r", stdin);
    //freopen("out.out", "w", stdout);
```

```
int test;
double miny, y;
TLine l1, l2;
TPoint p1, p2, p3, p4, inter;
TPoint tp1, tp2;
scanf("%d", &test);
while(test--)
{
    scanf("%lf%lf%lf%lf%lf%lf%lf", &p1.x, &p1.y,
    &p2.x, &p2.y, &p3.x, &p3.y, &p4.x, &p4.y);
    if(same(p1, p2) || same(p3, p4)
        || !isIntersected(p1, p2, p3, p4)
        || fabs(p1.y - p2.y) < eps //平行与 x 轴
        || fabs(p3.y - p4.y) < eps
      )
    {
        printf("0.00\n");
        continue;
    }
    l1 = lineFromSegment(p1, p2);
    l2 = lineFromSegment(p3, p4);
    inter = LineInter(l1, l2);
    if(p1.y > p2.y) tp1 = p1;
    else tp1 = p2;
    if(p3.y > p4.y) tp2 = p3;
    else tp2 = p4;
    if(tp1.y < tp2.y)
    {
        if(tp1.x >= min(p4.x, p3.x) && tp1.x <= max(p4.x, p3.x))
        {
            y = find_y(tp1.x, l2);
            if(y >= tp1.y)
            {
                printf("0.00\n");
                continue;
            }
        }
        miny = tp1.y;
    }
    else
    {
        if(tp2.x >= min(p1.x, p2.x) && tp2.x <= max(p1.x, p2.x))
        {
            y = find_y(tp2.x, l1);
```

```
                    if(y >= tp2.y)
                    {
                        printf("0.00\n");
                        continue;
                    }
                }
                miny = tp2.y;
            }
            if(fabs(miny - inter.y) < eps)
            {
                printf("0.00\n");
                continue;
            }
            tp1.x = find_x(miny, l1);
            tp2.x = find_x(miny, l2);
            tp1.y = tp2.y = miny;
            printf("%.2lf\n", triangleArea(tp1, tp2, inter));
        }
        return 0;
}
```

1.16.12 Pick 公式

```
//A = b / 2 + i -1   其中 b 与 i 分别表示在边界上及内部的格子点之个數
//http://www.hwp.idv.tw/bbs1/htm/%A6V%B6q%B7L%BFn%A4%C0/%A6V%B6q%B7L%BFn%A4%C0.
htm
// http://acm.pku.edu.cn/JudgeOnline/problem?id=2954

#include <stdio.h>
#include <stdlib.h>

typedef struct TPoint
{
    int x;
    int y;
}TPoint;

typedef struct TLine
{
    int a, b, c;
}TLine;

int triangleArea(TPoint p1, TPoint p2, TPoint p3)
{
```

```
    //已知三角形三个顶点的坐标，求三角形的面积
    int k = p1.x * p2.y + p2.x * p3.y + p3.x * p1.y
      - p2.x * p1.y - p3.x * p2.y - p1.x * p3.y;
    if(k < 0) return -k;
    else return k;
}

TLine lineFromSegment(TPoint p1, TPoint p2)
{
    //线段所在直线,返回直线方程的三个系统
    TLine tmp;
    tmp.a = p2.y - p1.y;
    tmp.b = p1.x - p2.x;
    tmp.c = p2.x * p1.y - p1.x * p2.y;
    return tmp;
}

void swap(int &a, int &b)
{
    int t;
    t = a;
    a = b;
    b = t;
}

int Count(TPoint p1, TPoint p2)
{
    int i, sum = 0, y;
    TLine l1 =   lineFromSegment(p1, p2);
    if(l1.b == 0) return abs(p2.y - p1.y) + 1;
    if(p1.x > p2.x) swap(p1.x, p2.x); //这里没有交换 WA 两次
    for(i = p1.x;i <= p2.x;i++){
        y = -l1.c - l1.a * i;
        if(y % l1.b == 0) sum++;
    }
    return sum;
}

int main()
{
    //freopen("in.in", "r", stdin);
    //freopen("OUT.out", "w", stdout);
    TPoint p1, p2, p3;
    while(scanf("%d%d%d%d%d%d", &p1.x, &p1.y, &p2.x, &p2.y, &p3.x, &p3.y) != EOF){
```

```
        if(p1.x == 0 && p1.y == 0 && p2.x == 0 && p2.y == 0 && p3.x == 0 && p3.y == 0) break;
        int    A = triangleArea(p1, p2, p3);//A 为面积的两倍
        int b = 0;
        int i;
        b = Count(p1, p2) + Count(p1, p3) + Count(p3, p2) - 3;//3 个顶点多各多加了一次
        //i = A / 2- b / 2 + 1;
        i = (A - b) / 2 + 1;
        printf("%d\n", i);
    }
    return 0;
}
```

1.16.13 N 点中三个点组成三角形面积最大

```
//Rotating Calipers algorithm


#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MaxNode 50005

int stack[MaxNode];
int top;
double max;

typedef struct TPoint
{
    int x;
    int y;
}TPoint;
TPoint point[MaxNode];

void swap(TPoint point[], int i, int j)
{
    TPoint tmp;
    tmp = point[i];
    point[i] = point[j];
    point[j] = tmp;
}

double multi(TPoint p1, TPoint p2, TPoint p0)
{
```

```
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}

double distance(TPoint p1, TPoint p2)
{
    return (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y);
}

int cmp(const void *a, const void *b)
{
    TPoint *c = (TPoint *)a;
    TPoint *d = (TPoint *)b;
    double k = multi(*c, *d, point[0]);
    if(k< 0) return 1;
    else if(k == 0 && distance(*c, point[0]) >= distance(*d, point[0]))
            return 1;
    else return -1;
}

void grahamScan(int n)
{
    //Graham 扫描求凸包
    int i, u;

    //将最左下的点调整到 p[0]的位置
    u = 0;
    for(i = 1;i <= n - 1;i++){
        if((point[i].y < point[u].y) ||
            (point[i].y == point[u].y && point[i].x    < point[u].x))
            u = i;
    }
    swap(point, 0, u);

    //将平 p[1]到 p[n - 1]按按极角排序，可采用快速排序
    qsort(point + 1, n - 1, sizeof(point[0]), cmp);

    for(i = 0;i <= 2;i++) stack[i] = i;
    top = 2;
    for(i = 3;i <= n - 1;i++){
        while(multi(point[i], point[stack[top]], point[stack[top - 1]]) >= 0){
            top--;
            if(top == 0) break;
        }
        top++;
```

```
            stack[top] = i;
        }
}

int main()
{
        double triangleArea(int i, int j, int k);
        void PloygonTriangle();
        int i, n;
        while(scanf("%d", &n) && n != -1){
            for(i = 0;i < n;i++)
                scanf("%d%d", &point[i].x, &point[i].y);
            if(n <= 2){
                printf("0.00\n");
                  continue;
            }
            if(n == 3){
                  printf("%.2lf\n", triangleArea(0, 1, 2));
                  continue;
            }
            grahamScan(n);
            PloygonTriangle();
            printf("%.2lf\n", max);
        }
        return 0;
}

void PloygonTriangle()
{
        double triangleArea(int i, int j, int k);
        int i, j , k;
        double area, area1;
        max = -1;
       for(i = 0;i <= top - 2;i++){
            k = -1;
            for(j = i + 1; j <= top - 1;j++){
                  if(k <= j) k= j + 1;
                  area = triangleArea(stack[i], stack[j], stack[k]);
                  if(area > max) max = area;
                  while(k + 1 <= top){
                        area1= triangleArea(stack[i], stack[j], stack[k + 1]);
                        if(area1 < area) break;
                        if(area1 > max) max = area1;
                        area = area1;
```

```
                    k++;
                }
            }
        }
    }
}

double triangleArea(int i, int j, int k)
{
    //已知三角形三个顶点的坐标，求三角形的面积
    double l = fabs(point[i].x * point[j].y + point[j].x * point[k].y
        + point[k].x * point[i].y - point[j].x * point[i].y
        - point[k].x * point[j].y - point[i].x * point[k].y) / 2;
    return l;
}
```

1.16.14 直线关于圆的反射

```
/*
fzu_1035
1.直线和圆的交点
2.点关于线的对称点
3.点到线的距离
4.直线方程
*/
#include <iostream>

#include <cmath>

using namespace std;

#define INF 999999999
const double eps = 1e-6;

int up;

typedef struct TPoint
{
    double x;
    double y;
}TPoint;

typedef struct TCircle
{
```

```
    TPoint center;
    double r;
}TCircle;

typedef struct TLine
{
    //直线标准式中的系数
    double a, b, c;
}TLine;

void SloveLine(TLine &line, TPoint start, TPoint dir)
{
    //根据直线上一点和直线的方向求直线的方程
    if(dir.x == 0){
        line.a = 1;
        line.b = 0;
        line.c = start.x;
    }
    else {
        double k = dir.y / dir.x;
        line.a = k;
        line.b = -1;
        line.c = start.y - k * start.x;
    }
}

TLine lineFromSegment(TPoint p1, TPoint p2)
{
    //线段所在直线,返回直线方程的三个系统
    TLine tmp;
    tmp.a = p2.y - p1.y;
    tmp.b = p1.x - p2.x;
    tmp.c = p2.x * p1.y - p1.x * p2.y;
    return tmp;
}

TPoint symmetricalPointofLine(TPoint p, TLine L)
{
    //p 点关于直线 L 的对称点
    TPoint p2;
    double d;
    d = L.a * L.a + L.b * L.b;
    p2.x = (L.b * L.b * p.x - L.a * L.a * p.x -
            2 * L.a * L.b * p.y - 2 * L.a * L.c) / d;
```

```
    p2.y = (L.a * L.a * p.y - L.b * L.b * p.y -
                2 * L.a * L.b * p.x - 2 * L.b * L.c) / d;
    return p2;
}

double distanc(TPoint p1, TPoint p2)
{
    //计算平面上两个点之间的距离
   return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}

bool samedir(TPoint dir, TPoint start, TPoint point)
{
    //判断方向
    TPoint tmp;
    tmp.x = point.x - start.x;
    tmp.y = point.y - start.y;
    if(tmp.x != 0 && dir.x != 0){
        if(tmp.x / dir.x > 0) return true;
        else return false;
    }
    else if(tmp.y != 0 && dir.y != 0){
        if(tmp.y / dir.y > 0) return true;
        else return false;
    }
    return true;
}

bool Intersected(TPoint &point, TLine line, const TCircle circle[],
                 TPoint start, TPoint dir, int which)
{
    //如果圆与直线有(有效交点)交点就存放在变量 point 中
    double a = line.a, b = line.b, c = line.c;
    double x0 = circle[which].center.x, y0 = circle[which].center.y;
    double r =   circle[which].r;
    //有交点，求交点
    double x2front = b * b + a * a;
    double x1front = -2 * x0 * b * b + 2 * a * b * y0 + 2 * a * c;
    double front = x0 * x0 * b * b + y0 * y0 * b * b
        + c * c + 2 * c * y0 * b - b * b * r * r;
    double d = x1front * x1front - 4 * x2front * front;
    TPoint p1, p2;
    bool k1, k2;
    if(fabs(d) < eps){
```

```
            //x2front 不可能等于零
            point.x = -x1front / x2front / 2;
            point.y = (-c - a * point.x) / b;
            //判断方向
            if(samedir(dir, start, point)) return true;
            else return false;
        }
        else if(d < 0) return false;
        else {
            p1.x = (-x1front + sqrt(d)) / 2 / x2front;
            p1.y = (-c - a * p1.x) / b;
            p2.x = (-x1front - sqrt(d)) / 2 / x2front;
            p2.y = (-c - a * p2.x) / b;
            k1 = samedir(dir, start, p1);
            k2 = samedir(dir, start, p2);
            if(k1 == false && k2 == false) return false;
            if(k1 == true && k2 == true){
                double dis1 = distanc(p1, start);
                double dis2 = distanc(p2, start);
                if(dis1 < dis2) point = p1;
                else point = p2;
                return true;
            }
            else if(k1 == true) point = p1;
            else point = p2;
            return true;
        }
}

void Reflect(int &num, TCircle circle[], TPoint start, TPoint dir, int n)
{
        //反复反射
        int i;
        TLine line;
        TPoint interpoint, newstart;
        int u;
        SloveLine(line, start, dir);
        int tag = 0;
        double mindis = INF;
        for(i = 1;i <= n;i++){
            if(i != up && Intersected(interpoint, line, circle, start, dir, i)){
                double dis = distanc(start, interpoint);
                if(dis < mindis){
                    tag = 1;
```

```
                    u = i;
                    mindis = dis;
                    newstart = interpoint;
                }
            }
        }
        if(tag == 0){
            cout << "inf" << endl;
            return ;
        }
        else {
            if(num == 10){
                cout << "..." << endl;
               return ;
            }
            cout << u << " ";
            num++;
            //新的方向
            TLine line1;
            TPoint p;
            line1 = lineFromSegment(newstart, circle[u].center);
            if(fabs(line1.a * start.x + line1.b * start.y +line1.c) <= eps){
                dir.x = -dir.x;
                dir.y = -dir.y;
            }
            else {
                p = symmetricalPointofLine(start, line1);//start 的对称点
                dir.x = p.x - newstart.x;
                dir.y = p.y - newstart.y;
            }

            start = newstart;
            up = u;
            Reflect(num, circle, start, dir, n);
        }
}

int main()
{
    //freopen("fzu_1035.in", "r", stdin);
    //freopen("fzu_1035.out", "w", stdout);
    int n, i, j, num, test = 1;
    TCircle circle[30];
    TPoint start, dir;
```

```
    while(cin >> n && n){
        for(i = 1;i <= n;i++){
            cin >> circle[i].center.x >> circle[i].center.y >> circle[i].r;
        }
        cin >> start.x >> start.y >> dir.x >> dir.y;

        cout << "Scene " << test++ << endl;

        num = 0;
        up = -1;
        Reflect(num, circle, start, dir, n);
        cout << endl;
    }
    return 0;
}
```

1.16.15 pku2002_3432_N 个点最多组成多少个正方形(hao)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define eps 1e-6
#define pi acos(-1.0)

#define PRIME 9991

struct point
{
    int x, y;
}p[2201];
int n;

struct HASH
{
    int cnt;
    int next;
}hash[50000];
int hashl;

int Hash(int n)
{
    int i = n % PRIME;
    while(hash[i].next != -1){
```

```
            if(hash[hash[i].next].cnt == n) return 1;
            else if(hash[hash[i].next].cnt > n) break;
            i = hash[i].next;
        }
        hash[hashl].cnt = n;
        hash[hashl].next = hash[i].next;
        hash[i].next = hashl;
        hashl++;
        return 0;
}

int Hash2(int n)
{
        int i = n % PRIME;
        while(hash[i].next != -1){
            if(hash[hash[i].next].cnt == n) return 1;
            else if(hash[hash[i].next].cnt > n) return 0;
            i = hash[i].next;
        }
        return 0;
}

int check(double ax, double ay, int &x, int &y)
{
        int a0 = (int)ax;
        int b0 = (int)ay;
        int tag1 = 0, tag2 = 0;
        if(fabs(a0 - ax) < eps){
            tag1 = 1;
            x = a0;
        }
        else if(fabs(a0 + 1 - ax) < eps){
            tag1 = 1;
            x = a0 + 1;
        }
        if(fabs(b0 - ay) < eps){
            tag2 = 1;
            y = b0;
        }
        else if(fabs(b0 + 1 - ay) < eps){
            y = b0 + 1;
            tag2 = 1;
        }
        if(tag1 == 1 && tag2 == 1) return 1;
```

```
    else return 0;
}

int squares(point p1, point p2, point &p3, point &p4)
{
    double a = (double)p2.x - p1.x;
    double b = (double)p2.y - p1.y;
    double midx = ((double)p1.x + p2.x) / 2;
    double midy = ((double)p1.y + p2.y) / 2;
    double tmp = a * a + b * b;
    double x1 = sqrt(b * b) / 2;
    double y1;
    if(fabs(b) < eps) y1 = sqrt(a * a + b * b) / 2;
    else y1 = -a * x1 / b;
    x1 += midx;
    y1 += midy;
    if(check(x1, y1, p3.x, p3.y) == 0) return 0;
    x1 = 2 * midx - x1;
    y1 = 2 * midy - y1;
    if(check(x1, y1, p4.x, p4.y) == 0) return 0;
    return 1;
}

int main()
{
    int i, j, cnt;
    while(scanf("%d", &n) != EOF && n)
    {
        for(i = 0;i < PRIME;i++) hash[i].next = -1;
        hashl = PRIME;
        int x1, y1, x2, y2;
        for (i = 0; i < n; i++){
            scanf("%d%d", &p[i].x, &p[i].y);
            Hash((p[i].x + 100000) * 100000 + p[i].y + 100000);
        }
        cnt = 0;
        for (i = 0; i < n; i++){
            for (j = i + 1; j < n; j++)
            {
                point a, b;
                if(squares(p[i], p[j], a, b) == 0) continue;
                if(Hash2((a.x + 100000) * 100000 + a.y + 100000) == 0) continue;
                if(Hash2((b.x + 100000) * 100000 + b.y + 100000) == 0) continue;
                cnt++;
```

```
                }
            }
            printf("%d\n", cnt / 2);
        }
        return 0;
}
```

1.16.16 pku1981_单位圆覆盖最多点(poj1981)CircleandPoints

```
/*
平面上 N 个点，用一个半径 R 的圆去覆盖，最多能覆盖多少个点？

比较经典的题目。

对每个点以 R 为半径画圆，对 N 个圆两两求交。这一步 O(N^2)。问题转化为求被覆盖次数最多的弧。

对每一个圆，求其上的每段弧重叠次数。假如 A 圆与 B 圆相交。A 上[PI/3, PI/2]的区间被 B 覆盖(PI 为圆周率)。那么对于 A 圆，我们在 PI/3 处做一个+1 标记，在 PI/2 处做一个-1 标记。

对于[PI*5/3, PI*7/3]这样横跨 0 点的区间只要在 0 点处拆成两段即可。

将一个圆上的所有标记排序，从头开始扫描。初始 ans = 0，碰到+1 标记给 ans++，碰到-1 标记 ans--。扫描过程中 ans 的最大值就是圆上被覆盖最多的弧。求所有圆的 ans 的最大值就是答案。

总复杂度 O(N^2 * logN)

*/#include <stdio.h>
#include <math.h>

#define eps 1e-6

struct point
{
    double x, y;
};

double dis(point p1, point p2)
{
    point p3;
    p3.x = p2.x - p1.x;
    p3.y = p2.y - p1.y;
    return p3.x * p3.x + p3.y * p3.y;
```

```
}

point find_centre(point p1, point p2)
{
    point p3, mid, centre;
    double b, c, ang;
    p3.x = p2.x - p1.x;
    p3.y = p2.y - p1.y;
    mid.x = (p1.x + p2.x) / 2;
    mid.y = (p1.y + p2.y) / 2;
    b = dis(p1, mid);
    c = sqrt(1 - b);
    if(fabs(p3.y) < eps)//垂线的斜角 90 度
    {
        centre.x = mid.x;
        centre.y = mid.y + c;
    }
    else
    {
        ang = atan(-p3.x / p3.y);
        centre.x = mid.x + c * cos(ang);
        centre.y = mid.y + c * sin(ang);
    }
    return centre;
}

int main()
{
    int n, ans, tmpans, i, j, k;
    point p[305], centre;
    double tmp;
    while(scanf("%d", &n) && n)
    {
        for(i = 0;i < n;i++)
            scanf("%lf%lf", &p[i].x, &p[i].y);
        ans = 1;
        for(i = 0;i < n;i++)
            for(j = i + 1;j < n;j++)
            {
                if(dis(p[i], p[j]) > 4) continue;
                tmpans = 0;
                centre = find_centre(p[i], p[j]);
                for(k = 0;k < n;k++)
                {
```

```
                        //if(tmpans + n - k <= ans) break;
                        tmp = dis(centre, p[k]);
                        //if(tmp < 1.0 || fabs(tmp - 1.0) < eps) tmpans++;
                        if(tmp <= 1.000001) tmpans++;
                    }
                    if(ans < tmpans) ans = tmpans;
                }
            printf("%d\n", ans);
        }
        return 0;
}
```

1.16.17 pku3668_GameofLine_N 个点最多确定多少互不平行的直线(poj3668)

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define eps 1e-6
#define pi acos(-1)

struct point
{
    double x, y;
};

double FindSlewRate(point p1, point p2)
{
    point p;
    p.x = p2.x - p1.x;
    p.y = p2.y - p1.y;
    if(fabs(p.x) < eps) return pi / 2;
    double tmp = atan(p.y / p.x);
    if(tmp < 0) return pi + tmp;
    return tmp;
}

int cmp(const void *a, const void *b)
{
    double *c = (double *)a;
    double *d = (double *)b;
    if(*c < *d) return -1;
    return 1;
```

```
}

int main()
{
    int n, rt;
    point p[205];
    double rate[40005];
    while(scanf("%d", &n) != EOF)
    {
        for(int i = 0;i < n;i++)
            scanf("%lf%lf", &p[i].x ,&p[i].y);
        rt = 0;
        for(int i = 0;i < n;i++)
            for(int j = i + 1;j < n;j++)
                rate[rt++] = FindSlewRate(p[i], p[j]);
        qsort(rate, rt, sizeof(rate[0]), cmp);
        int ans = 1;
        for(int i = 1;i < rt;i++)
            if(rate[i] > rate[i - 1]) ans++;
        //注意这里写 fabs(rate[i] - rate[i - 1]) > eps Wrong Answer
        printf("%d\n", ans);
    }
    return 0;
}
```

1.16.18 求凸多边形直径

```
#include<stdio.h>
#include<math.h>

#define eps 1e-6
#define MaX 6000

/*-----------多边形结构------------*/
struct POLYGON{
    int n;                                          //多边形顶点数
    double x[MaX],y[MaX];           //顶点坐标
}poly;

int zd[100000][2],znum;         //跖对的集合和跖对的数量


/*------------辅助函数-----------*/
double dist(int a,int b,int c)
```

```
{
    double vx1,vx2,vy1,vy2;
    vx1=poly.x[b]-poly.x[a]; vy1=poly.y[b]-poly.y[a];
    vx2=poly.x[c]-poly.x[a]; vy2=poly.y[c]-poly.y[a];
    return fabs(vx1*vy2 - vy1*vx2);
}


/*-------------求凸多边形直径的函数-------------*/
double DIAMETER()
{
    znum=0;
    int i,j,k=1;
    double m,tmp;
    while(dist(poly.n-1,0,k+1) > dist(poly.n-1,0,k)+eps)
        k++;
    i=0; j=k;
    while(i<=k && j<poly.n)
    {
        zd[znum][0]=i; zd[znum++][1]=j;
        while(dist(i,i+1,j+1)>dist(i,i+1,j)-eps && j<poly.n-1)
        {
            zd[znum][0]=i; zd[znum++][1]=j;
            j++;
        }
        i++;
    }
    m=-1;
    for(i=0;i<znum;i++)
    {
        tmp =(poly.x[zd[i][0]]-poly.x[zd[i][1]]) * (poly.x[zd[i][0]]-poly.x[zd[i][1]]);
        tmp+=(poly.y[zd[i][0]]-poly.y[zd[i][1]]) * (poly.y[zd[i][0]]-poly.y[zd[i][1]]);
        if(m<tmp) m=tmp;
    }
    return sqrt(m);
}

/*----------主函数----------*/
int main()
{
    int i;
    while(scanf("%d",&poly.n)==1)
    {
        for(i=0;i<poly.n;i++)
```

```
            scanf("%lf %lf",&poly.x[i],&poly.y[i]);
        printf("%.3lf\n",DIAMETER());
    }
    return 0;
}
```

2.组合

2.1 组合公式

1. C(m,n)=C(m,m-n)
2. C(m,n)=C(m-1,n)+C(m-1,n-1)

derangement D(n) = n!(1 - 1/1! + 1/2! - 1/3! + ... + (-1)^n/n!)
= (n-1)(D(n-2) - D(n-1))
Q(n) = D(n) + D(n-1)

求和公式,k = 1..n
1. sum(k) = n(n+1)/2
2. sum(2k-1) = n^2
3. sum(k^2) = n(n+1)(2n+1)/6
4. sum((2k-1)^2) = n(4n^2-1)/3
5. sum(k^3) = (n(n+1)/2)^2
6. sum((2k-1)^3) = n^2(2n^2-1)
7. sum(k^4) = n(n+1)(2n+1)(3n^2+3n-1)/30
8. sum(k^5) = n^2(n+1)^2(2n^2+2n-1)/12
9. sum(k(k+1)) = n(n+1)(n+2)/3
10. sum(k(k+1)(k+2)) = n(n+1)(n+2)(n+3)/4
12. sum(k(k+1)(k+2)(k+3)) = n(n+1)(n+2)(n+3)(n+4)/5

2.2 排列组合生成

//gen_perm 产生字典序排列 P(n,m)
//gen_comb 产生字典序组合 C(n,m)
//gen_perm_swap 产生相邻位对换全排列 P(n,n)
//产生元素用 1..n 表示
//dummy 为产生后调用的函数,传入 a[]和 n,a[0]..a[n-1]为一次产生的结果

```
#define MAXN 100
int count;

#include <iostream.h>
```

```
void dummy(int* a,int n){
    int i;
    cout<<count++<<": ";
    for (i=0;i<n-1;i++)
        cout<<a[i]<<' ';
    cout<<a[n-1]<<endl;
}

void _gen_perm(int* a,int n,int m,int l,int* temp,int* tag){
    int i;
    if (l==m)
        dummy(temp,m);
    else
        for (i=0;i<n;i++)
            if (!tag[i]){
                temp[l]=a[i],tag[i]=1;
                _gen_perm(a,n,m,l+1,temp,tag);
                tag[i]=0;
            }
}

void gen_perm(int n,int m){
    int a[MAXN],temp[MAXN],tag[MAXN]={0},i;
    for (i=0;i<n;i++)
        a[i]=i+1;
    _gen_perm(a,n,m,0,temp,tag);
}

void _gen_comb(int* a,int s,int e,int m,int& count,int* temp){
    int i;
    if (!m)
        dummy(temp,count);
    else
        for (i=s;i<=e-m+1;i++){
            temp[count++]=a[i];
            _gen_comb(a,i+1,e,m-1,count,temp);
            count--;
        }
}

void gen_comb(int n,int m){
    int a[MAXN],temp[MAXN],count=0,i;
    for (i=0;i<n;i++)
        a[i]=i+1;
```

```
    _gen_comb(a,0,n-1,m,count,temp);
}

void _gen_perm_swap(int* a,int n,int l,int* pos,int* dir){
    int i,p1,p2,t;
    if (l==n)
        dummy(a,n);
    else{
        _gen_perm_swap(a,n,l+1,pos,dir);
        for (i=0;i<l;i++){
            p2=(p1=pos[l])+dir[l];
            t=a[p1],a[p1]=a[p2],a[p2]=t;
            pos[a[p1]-1]=p1,pos[a[p2]-1]=p2;
            _gen_perm_swap(a,n,l+1,pos,dir);
        }
        dir[l]=-dir[l];
    }
}

void gen_perm_swap(int n){
    int a[MAXN],pos[MAXN],dir[MAXN],i;
    for (i=0;i<n;i++)
        a[i]=i+1,pos[i]=i,dir[i]=-1;
    _gen_perm_swap(a,n,0,pos,dir);
}
```

2.3 生成 gray 码

```
//生成 reflected gray code
//每次调用 gray 取得下一个码
//000...000 是第一个码,100...000 是最后一个码
void gray(int n,int *code){
    int t=0,i;
    for (i=0;i<n;t+=code[i++]);
    if (t&1)
        for (n--;!code[n];n--);
    code[n-1]=1-code[n-1];
}
```

2.4 置换(polya)

```
//求置换的循环节,polya 原理
//perm[0..n-1]为 0..n-1 的一个置换(排列)
```

```
//返回置换最小周期,num 返回循环节个数
#define MAXN 1000

int gcd(int a,int b){
    return b?gcd(b,a%b):a;
}

int polya(int* perm,int n,int& num){
    int i,j,p,v[MAXN]={0},ret=1;
    for (num=i=0;i<n;i++)
        if (!v[i]){
            for (num++,j=0,p=i;!v[p=perm[p]];j++)
                v[p]=1;
            ret*=j/gcd(ret,j);
        }
    return ret;
}
```

2.5 字典序全排列

```
//字典序全排列与序号的转换
int perm2num(int n,int *p){
    int i,j,ret=0,k=1;
    for (i=n-2;i>=0;k*=n-(i--))
        for (j=i+1;j<n;j++)
            if (p[j]<p[i])
                ret+=k;
    return ret;
}

void num2perm(int n,int *p,int t){
    int i,j;
    for (i=n-1;i>=0;i--)
        p[i]=t%(n-i),t/=n-i;
    for (i=n-1;i;i--)
        for (j=i-1;j>=0;j--)
            if (p[j]<=p[i])
                p[i]++;
}
```

2.6 字典序组合

```
//字典序组合与序号的转换
```

```
//comb 为组合数 C(n,m),必要时换成大数,注意处理 C(n,m)=0|n<m
int comb(int n,int m){
    int ret=1,i;
    m=m<(n-m)?m:(n-m);
    for (i=n-m+1;i<=n;ret*=(i++));
    for (i=1;i<=m;ret/=(i++));
    return m<0?0:ret;
}

int comb2num(int n,int m,int *c){
    int ret=comb(n,m),i;
    for (i=0;i<m;i++)
        ret-=comb(n-c[i],m-i);
    return ret;
}

void num2comb(int n,int m,int* c,int t){
    int i,j=1,k;
    for (i=0;i<m;c[i++]=j++)
        for (;t>(k=comb(n-j,m-i-1));t-=k,j++);
}
```

2.7 一些原理及其例子

2.7.1 常见递推关系及应用

1.算术序列

每一项比前一项大一个常数 d；

若初始项为 h0：则递推关系为 $h_n=h_{n-1}+d=h_0+nd$；

对应的各项为：h_0，h_0+d，h_0+2d，....,h_0+nd;

前 n 项的和为(n+1)h0+dn(n+1)/2

例 5: 1,2,3,...

例 6: 1,3,5,7...等都是算术序列。

2.几何序列

每一项是前面一项的常数 q 倍

若初始项为 h0：则递推关系为 $h_n=h_0q^{n-1}q=h0q^n$；

对应的各项为: h_0，h_0q^1，h_0q^2，....,h_0q^n

例 7: 1,2,4,8,16,...

例 8: 5,15,45,135,...等都是几何序列;

前 n 项和为$((q^{n+1}-1)/(q-1))h_0$

3.Fibonacci 序列

除第一、第二项外每一项是它前两项的和；

若首项为 f_0 为 0,则序列为 0，1，1，2，3，5，8...递推关系为(n>=2)$f_n=f_{n-1}+f_{n-2}$

前 n 项的和 $S_n=f_0+f_1+f_2+...+f_n=f_{n+2}-1$

例 9:以下是 Fibonacci 的示例:

1.楼梯有 n 阶台阶,上楼可以一步上 1 阶,也可以一步上 2 阶,编一程序计算共有多少种不同的走法?

2.有一对雌雄兔,每两个月就繁殖雌雄各一对兔子.问 n 个月后共有多少对兔子?

3.有 n*2 的一个长方形方格,用一个 1*2 的骨牌铺满方格。求铺法总数?

4.错位排列

首先看例题:

例 10:在书架上放有编号为 1,2,....n 的 n 本书。现将 n 本书全部取下然后再放回去,当放回去时要求每本书都不能放在原来的位置上。

例如:n=3 时:

原来位置为:123

放回去时只能为:312 或 231 这两种

问题:求当 n=5 时满足以上条件的放法共有多少种?(不用列出每种放法) (44)

{1，2，3，....,n}错位排列是{1,2,3,..,n}的一个排列 $i_1i_2...i_n$,使得 $i_1<>1,i_2<>2,i_3<>3,...i_n<>n$

错位排列数列为

0,1,2,9,44,265,....

错位排列的递推公式是:$d_n=(n-1)(d_{n-2}+d_{n-1})(n>=3)$

$=nd_{n-1}+(-1)^{n-2}$

5.分平面的最大区域数

1.直线分平面的最大区域数的序列为:

2，4，7，11，....,

递推公式是: $f_n=f_{n-1}+n=n(n+1)/2+1$

2.折线分平面的最大区域数的序列为:

2, 7, 16,29, ...,

递推公式是:$f_n=(n-1)(2n-1)+2n$;

3.封闭曲线(如一般位置上的圆)分平面的最大区域数的序列为:

2, 4, 8, 14,...,

递推公式是:$f_n=f_{n-1}+2(n-1)=n^2-n+2$

6.Catalan 数列

先看下面两个例题:

例 11:将一个凸多边形区域分成三角形区域的方法数?

令 hn 表示具有 n+1 条边的凸多边形区域分成三角形的方法数。同时令 h1=1，则 hn 满足递推关系

$h_n=h_1h_{n-1}+h_2h_{n-2}+...+h_{n-1}h_1(n>=2)$(想一想,为什么?)

该递推关系的解为 $h_n=c(2n-2,n-1)/n$ (n=1,2,3,...)

其对应的序列为 1,1,2,5,14,42,132,....从第二项开始分别是三边形,四边形,...的分法数

即 k 边形分成三角形的方法数为 $h_k=c(2k-4,k-2)/(k-1)(k>=3)$

例 12:n 个+1 和 n 个-1 构成 2n 项 $a_1,a_2,...,a_{2n}$

其部分和满足 a1+a2+...+ak>=0(k=1,2,3,..,2n)对与 n 该数列为

Cn=c(2k,k)/(k+1) (k>=0) 对应的序列为 1,1,2,5,14,42,132,...

序列 1,1,2,5,14,42,132,....叫 Catalan 数列。

例 13：下列问题都是 Catalan 数列。

1.有 2n 个人排成一行进入剧场。入场费 5 元。其中只有 n 个人有一张 5 元钞票，另外 n 人只有 10 元钞票，剧院无其它钞票，问有多少中方法使得只要有 10 元的人买票，售票处就有 5 元的钞票找零?

2.一位大城市的律师在她住所以北 n 个街区和以东 n 个街区处工作。每天她走 2n 个街区去上班。如果他

从不穿越（但可以碰到）从家到办公室的对角线，那么有多少条可能的道路？

3.在圆上选择 2n 个点,将这些点成对连接起来使得所得到的 n 条线段不相交的方法数?

4.n 个结点可够造多少个不同的二叉树?

5.一个栈(无穷大)的进栈序列为 1,2,3,..n,有多少个不同的出栈序列?

3.数论

3.1 阶乘最后非 0 位

```
//求阶乘最后非零位,复杂度 O(nlogn)
//返回该位,n 以字符串方式传入
#include <string.h>
#define MAXN 10000

int lastdigit(char* buf){
	const int mod[20]={1,1,2,6,4,2,2,4,2,8,4,4,8,4,6,8,8,6,8,2};
	int len=strlen(buf),a[MAXN],i,c,ret=1;
	if (len==1)
		return mod[buf[0]-'0'];
	for (i=0;i<len;i++)
		a[i]=buf[len-1-i]-'0';
	for (;len;len-=!a[len-1]){
		ret=ret*mod[a[1]%2*10+a[0]]%5;
		for (c=0,i=len-1;i>=0;i--)
			c=c*10+a[i],a[i]=c/5,c%=5;
	}
	return ret+ret%2*5;
}
```

3.2 模线性方程组

```
#ifdef WIN32
typedef __int64 i64;
#else
typedef long long i64;
#endif
//扩展 Euclid 求解 gcd(a,b)=ax+by
int ext_gcd(int a,int b,int& x,int& y){
	int t,ret;
	if (!b){
```

```
        x=1,y=0;
        return a;
    }
    ret=ext_gcd(b,a%b,x,y);
    t=x,x=y,y=t-a/b*y;
    return ret;
}

//计算 m^a, O(loga), 本身没什么用，注意这个按位处理的方法 :-P
int exponent(int m,int a){
    int ret=1;
    for (;a;a>>=1,m*=m)
        if (a&1)
            ret*=m;
    return ret;
}

//计算幂取模 a^b mod n, O(logb)
int modular_exponent(int a,int b,int n){ //a^b mod n
    int ret=1;
    for (;b;b>>=1,a=(int)((i64)a)*a%n)
        if (b&1)
            ret=(int)((i64)ret)*a%n;
    return ret;
}

//求解模线性方程 ax=b (mod n)
//返回解的个数,解保存在 sol[]中
//要求 n>0,解的范围 0..n-1
int modular_linear(int a,int b,int n,int* sol){
    int d,e,x,y,i;
    d=ext_gcd(a,n,x,y);
    if (b%d)
        return 0;
    e=(x*(b/d)%n+n)%n;
    for (i=0;i<d;i++)
        sol[i]=(e+i*(n/d))%n;
    return d;
}

//求解模线性方程组(中国余数定理)
//   x = b[0] (mod w[0])
//   x = b[1] (mod w[1])
//   ...
```

```
//    x = b[k-1] (mod w[k-1])
//要求 w[i]>0,w[i]与 w[j]互质,解的范围 1..n,n=w[0]*w[1]*...*w[k-1]
int modular_linear_system(int b[],int w[],int k){
    int d,x,y,a=0,m,n=1,i;
    for (i=0;i<k;i++)
        n*=w[i];
    for (i=0;i<k;i++){
        m=n/w[i];
        d=ext_gcd(w[i],m,x,y);
        a=(a+y*m*b[i])%n;
    }
    return (a+n)%n;
}
```

3.3 素数

```
//用素数表判定素数,先调用 initprime
int plist[10000],pcount=0;

int prime(int n){
    int i;
    if ((n!=2&&!(n%2))||(n!=3&&!(n%3))||(n!=5&&!(n%5))||(n!=7&&!(n%7)))
        return 0;
    for (i=0;plist[i]*plist[i]<=n;i++)
        if (!(n%plist[i]))
            return 0;
    return n>1;
}

void initprime(){
    int i;
    for (plist[pcount++]=2,i=3;i<50000;i++)
        if (prime(i))
            plist[pcount++]=i;
}

//miller rabin
//判断自然数 n 是否为素数
//time 越高失败概率越低,一般取 10 到 50
#include <stdlib.h>
#ifdef WIN32
typedef __int64 i64;
#else
typedef long long i64;
```

```
#endif

int modular_exponent(int a,int b,int n){ //a^b mod n
    int ret;
    for (;b;b>>=1,a=(int)((i64)a)*a%n)
        if (b&1)
            ret=(int)((i64)ret)*a%n;
    return ret;
}

// Carmicheal number: 561,41041,825265,321197185
int miller_rabin(int n,int time=10){
    if (n==1||(n!=2&&!(n%2))||(n!=3&&!(n%3))||(n!=5&&!(n%5))||(n!=7&&!(n%7)))
        return 0;
    while (time--)
        if (modular_exponent(((rand()&0x7fff<<16)+rand()&0x7fff+rand()&0x7fff)%(n-1)+1,n-1,n)!=1)
            return 0;
    return 1;
}
```

//1181 Miller-Rabbin ＆＆ Pallord

```
#include <ctime>
#include <cmath>
#include <cstdlib>
#include <algorithm>
using namespace std;

__int64 Mulmod(__int64 a, __int64 b, __int64 c) {
    if (b == 0)   return 0;
    __int64 ans = Mulmod(a, b/2, c);
    ans = (ans*2) % c;
    if (b % 2)
        ans = (ans + a) % c;
    return ans;
}

__int64 Witness(const __int64 a, const __int64 b, const __int64 c)
// Calculate a^b % c
// if find witness prove that c is not a prime then return 0 , else return a^b % c
{
    if (b == 0)   return 1;
    __int64 x = Witness(a, b/2, c);
    if (x == 0)   return 0;
    __int64 y = Mulmod(x, x, c); //y = (x*x)%c;
```

```
        if (y == 1 && x != 1 && x != c-1)
            return 0;                    //another witness prove n is not a pirme
        if (b % 2)
            y = Mulmod(y, a, c);       //y = (a*y) % c;
        return y;
}

bool IsPrime(const __int64 &n)
{
        if(n < 2)      return false;
        if(n == 2)    return true;
        int num = 10;
        while (num--)
            if (Witness(rand()%(n-2) + 2, n-1, n) != 1)
                return false;
        return true;
}

//Pollard 质因子分解
void Pollard(__int64);
__int64 Gcd(__int64 a, __int64 b) {
        __int64 t;
        while (b){
            t = a;a = b;b = t%b;
        }
        return a;
}

__int64 factor[110], cnt;

void Factor(__int64 n) {
        __int64 d = 2;
        while (true) {
            if(n%d == 0) {
                Pollard(d);
                Pollard(n/d);
                return;
            }
            d++;
        }
}

void Pollard(__int64 n)
{
```

```
    if(n <= 0)
        while(1)
        printf("error\n");
    if(n == 1)
        return;

    if(IsPrime(n)) {
        factor[cnt++] = n;
        return;
    }

    __int64 i = 0, k = 2, y, x, d;
    x = y = rand() % (n-1) + 1;
    while (i++ < 123456) {
        x = (Mulmod(x, x, n) + n-1) % n; //x = (x*x-1) % n
        d = Gcd((y-x+n) % n, n);
        if (d != 1) {
            Pollard(d);
            Pollard(n/d);
            return;
        }
        if (i==k) {
            y = x;
            k *= 2;
        }
    }
    Factor(n);
}
int main() {
    srand(time(0));
    __int64 n, nn;
    int i, Case;
    scanf("%d", &Case);
    while (Case--) {
        scanf("%I64d", &n);
        if ( IsPrime(n) )
            printf("Prime\n");
        else {
            cnt = 0;
            Pollard(n);
            sort(factor, factor + cnt);
            printf("%I64d\n", factor[0]);
        }
    }
```

```
    return 0;
}
```

3.4 欧拉函数

```
int gcd(int a,int b){
    return b?gcd(b,a%b):a;
}

inline int lcm(int a,int b){
    return a/gcd(a,b)*b;
}

//求 1..n-1 中与 n 互质的数的个数
int eular(int n){
    int ret=1,i;
    for (i=2;i*i<=n;i++)
        if (n%i==0){
            n/=i,ret*=i-1;
            while (n%i==0)
                n/=i,ret*=i;
        }
    if (n>1)
        ret*=n-1;
    return ret;
}
```

3.5 方程 ax+by=c 整数解的应用

有三个分别装有 a 升水、b 升水和 c 升水的量筒(gcd(a，b)＝1，c>b>a>0),现 c 筒装水，问能否在 c 筒个量出 d 升水(c>d>0)。

算法分析：

量水过程实际上就是倒来倒去，每次倒的时候总有如下几个持点：

1.总有一个筒中的水没有变动；

2．不是一个筒被倒满就是另一个筒被倒光；

3．c 筒仅起中转作用，而本身容积除了必须足够装下 a 简和 b 简的全部水外，别无其它限制。

```
program mw;
type
node=array[0..1] of longint;
var
a,b,c:node;
d,step,x,y:longint;
function exgcd(a,b:longint;var x,y:longint):longint;
var t:longint;
```

```
begin
  if b=0 then
    begin
      exgcd:=a;;x:=1;y:=0;
    end
    else
    begin
      exgcd:=exgcd(b,a mod b,x,y);
      t:=x;x:=y;y:=t-(a div b)*y
    end;
end;
procedure equation(a,b,c:longint;var x0,y0:longint);
var d,x,y:longint;
begin
  d:=exgcd(a,b,x,y);
  if c mod d>0 then
  begin
    writeln('no answer');
    halt;
  end else
  begin
    x0:=x*(c div d);
    y0:=y*(c div d);
  end;
end;
procedure fill(var a,b:node);
var t:longint;
begin
  if a[1]<b[0]-b[1] then t:=a[1]
                                else t:=b[0]-b[1];
  a[1]:=a[1]-t;
  b[1]:=b[1]+t
end;
begin
  write('a,b,c,d=');
  readln(a[0],b[0],c[0],d);
  equation(a[0],b[0],d,x,y);
  step:=0;
  a[1]:=0;b[1]:=0;c[1]:=c[0];
  writeln(step:5,':',a[1]:5,b[1]:5,c[1]:5);
  if x>0 then
    repeat
      if a[1]=0 then fill(c,a) else
                          if b[1]=b[0] then fill(b,c) else fill(a,b);
```

```
    inc(step);
    writeln(step:5,':',a[1]:5,b[1]:5,c[1]:5);
  until c[1]=d
  else
    repeat
      if b[1]=0 then fill(c,b) else
                    if a[1]=a[0] then fill(a,c) else fill(b,a);
      inc(step);
    writeln(step:5,':',a[1]:5,b[1]:5,c[1]:5);
    until c[1]=d;
end.
```

3.6 高精度

3.6.1 平方根

```
int    x[maxlen], y[maxlen], z[maxlen], bck[maxlen], lx, ly, lz;

int    IsSmaller()    // isz<=y ?
{ int I = ly;    while( I > 1 && z[I ]== y[i] ) i--;    return ( z[ i ]<=y[ I ]);    }

void Sovle()    // y^2=x;
{
   Int I , j , k;
lx = ( ly + 1) / 2; ly = lx * 2;
memset( x , 0 , sizeof( x ) ); memset( z , 0 , sizeof( z ) );
  for( i= lx; I > 0; i--){
    for( j= 1; j < 10; x[i] = j++){
          memcpy(bck,z,sizeof(z));
z[ 2*I – 1 ] ++;    for( k = I; k < lx; k ++)
{ z[ i -1 + k]+= 2 * x[ k ];    z[ i + k] + = z[ I -1 +k ]/ 10; z[ I -1 + k]%=10 ;}
for( k = lx + I; k<=ly; k++) { z[ k + 1 ]+= z [ k ] / 10;    z[ k ]% = 10; }
if( !IsSmaller()) break;
}
If(j<10) memcpy( z, bck, sizeof( bck ));
}
    for( i = lx ; I > 0; i--) cout<<x[i]; cout<<endl;
}

int main()
{
  char ch , s[ maxlen ]; int i , j;
  memset( y ,0, sizeof( y ));
```

```
  cin>>s;    ly= srtlen ( s );
  for( i = 0; I < ly ; i++) y[ i+1 ] = s [ ly – 1 – i ] – ‘0’;
Sovle();
return    0 ;
}
```

3.6.2 高精度乘幂

```
#include<iostream>
using namespace std;
void change(int m,char * a)    //数字转化为字符串存储
{
  int t=m;
  for(int i=0; t;i++){
    a[i]=t%10+48;
    t/=10;
  }
}
void reverse(char * str,int l) //字符串翻转
{
  char temp;
  for(int i=0;i<=l/2;i++){
    temp=str[i];
    str[i]=str[l-i];
    str[l-i]=temp;
  }
}
void mul(int m,int n,char * result)    //高精度 m^n 乘法
{
  char a[20]={0};
  int    c[5000]={0};
  int la,lr;
  if(n==0 || m==1){result[0]='1';return ;}
  change(m,a);          //将数字转化为字符
  la=strlen(a)-1;          //记录字符 a 的位数
  lr=la;
  strcpy(result,a);        //积初始化为 a*1
  int i,j,k,l;
  for(i=2;i<=n;i++){        //result*=a^(n-1)
    memset(c,0,sizeof(c));
    for(j=0;j<=la;j++)        //大数相乘
      for(k=0;k<=lr;k++){
        c[j+k]+=(a[j]-48)*(result[k]-48);
        c[j+k+1]+=c[j+k]/10;
```

```
      c[j+k]%=10;
     }
     l=k+j+1;     //记录当前可能的最大位数
   while(c[l]==0)l--;   //去除 la+lr+1 最高几位的的 0
   memset(result,0,sizeof(result));
   for(j=0;j<=l;j++)result[j]=c[j]+'0';//将临时变量 c 里的数字转化为字符存到 result 中
   lr=l;        //刷新 result 的字符个数
  }
  reverse(result,lr);     //字符串翻转，方便输出
}
main()
{

  int m,n;
  while(scanf("%d%d",&m,&n)!=EOF){
   char  result[5000]={0};   //这句必须放到循环体内，WA 得好苦 因为有这句 if(n==0 || m==1){result[0]='1';return ;}
   if(m==0&&n==0)break;
  mul(m,n,result);
  printf("%s\n",result);
  }
  return 0;
}
```

3.6.3 高精度除高精度

高精度除法

```
#i nclude <cstdio>
#i nclude <cstring>
#define K 4
#define N 1024
#define UP 10000
//将字符串转换为按四位存储的整数
void strToInt(char *x,int *y)
{
  int i,k,m,cnt,wgt;
  cnt = strlen(x);
  i = cnt - 1;
  m = y[0] = (cnt + K - 1) / K;
       while(m > 0)
  {
   k = 0;
   wgt = 1;
   y[m] = 0;
   while(i >= 0 && k < K)
   {
```

```
      y[m] += (x[i] - 48) * wgt;
      wgt *= 10;
      i --,k ++;
    }
    m --;
  }
}
//过滤数字前面的 0
void skipZero(int *x)
{
  int i,k;
  for(i = 1;i <= x[0];i ++)
    if(x[i])    break;
  if(i > x[0])    i --;
  for(k = i;k <= x[0];k ++)
    x[k - i + 1] = x[k];
  x[0] = x[0] - i + 1;
}
//将按四位存储的结果还原
void highToLow(int *high,bool flag)
{
  int i,j,k,n,p,cnt,sgn;
  sgn = high[high[0] + 1];
  cnt = high[0] * K;
  p = k = cnt;
       for(i = high[0];i > 0;i --)
  {
    j = k;
    n = high[i];
    while(n)
    {
      high[j --] = n%10;
      n /= 10;
    }
    while(j > k - K)
          high[j --] = 0;
    k -= K;
  }
  high[0] = p;

  if(flag)    skipZero(high);

  high[high[0] + 1] = sgn;
}
```

```
//比较两个大整数的大小关系
int compare(int *x,int *y)
{
  int i = 1;
  if(x[0] > y[0])    return 1;
  if(x[0] < y[0])    return -1;
       while(i <= x[0])
  {
    if(x[i] > y[i])    return 1;
    if(x[i] < y[i])    return -1;
    i ++;
  }
  return 0;
}
//两数相减
void subtract(int *x,int *y,int *ans)
{
  int tmp[N];
  int i,j,p,s,t,res,sub,sgn;
  sgn = 1;
  p = compare(x,y);
       if(p < 0)
       {
              sgn = 0;
    for(i = 0;i <= x[0];i ++)    tmp[i] = x[i];
              for(i = 0;i <= y[0];i ++)    x[i] = y[i];
              for(i = 0;i <= tmp[0];i ++)    y[i] = tmp[i];
       }
  sub = 0;
       i = ans[0] = x[0];
  j = y[0];
       while(i > 0)
       {
              t = 0,s = x[i];
              if(j > 0)    t = y[j --];
              res = s - t - sub;
              sub = 0;
              if(res < 0)    { res += UP; sub = 1; }
              ans[i --] = res;
       }
  skipZero(ans);
  ans[ans[0] + 1] = sgn;
}
void singleMultiply(int *x,int n,int *ans)
```

```
{
  int i,k,l,s,inc = 0;
       i = l = x[0];
       ans[0] = k = l + 1;
      while(i > 0)
       {
            s = x[i --] * n + inc;
            inc = s / UP;
            ans[k --] = s % UP;
       }
  ans[k] = inc;
  skipZero(ans);
}
//二分查找试除的结果
int finds(int *x,int *y)
{
  int res[N];
  int l,m,h,p,sgn;
  l = 0;
  h = UP - 1;
  while(l <= h)
  {
    m = (l + h) / 2;
    singleMultiply(y,m,res);
    sgn = compare(x,res);
    if(sgn >= 0)    { p = m; l = m + 1; }
    else    h = m - 1;
  }
  return p;
}
void divide(int *x,int *y,int prec,int *res,int *ans)
{
  int i,k;
  int tmp[N],tmp1[N],tmp2[N],tmp3[N];
  for(i = 0;i <= x[0];i ++)
    tmp1[i] = x[i];
  for(i = 0;i <= y[0] ;i ++)
    tmp2[i] = y[i];
  i = 1;
  while(i < tmp1[0] && i < tmp2[0])
  {
    tmp[i] = tmp1[i];
    i ++;
  }
```

```
  ans[0] = 0;
  tmp[0] = i - 1;
  while(i <= tmp1[0])
  {
    tmp[0] ++;
    tmp[tmp[0]] = tmp1[i ++];
    skipZero(tmp);
    k = 0;
    if(tmp[0] >= tmp2[0])
          k = finds(tmp,tmp2);
    ans[++ ans[0]] = k;
    if(k)
    {
          singleMultiply(y,k,tmp3);
          subtract(tmp,tmp3,tmp);
             }
       }
  for(i = 0;i <= tmp[0] + 1;i ++)
    res[i] = tmp[i];
  highToLow(res,true);
  ans[ans[0] + 1] = 1;
}
int ans[N];
int main()
{
  int i,w[N],x[N],y[N];
  char cx[N],cy[N];
  while(scanf("%s%s",cx,cy) == 2)
  {
    strToInt(cx,x);
    strToInt(cy,y);
    divide(x,y,-1,w,ans);
    highToLow(ans,true);
    if(ans[ans[0] + 1] == 0)
      printf("-");
    for(i = 1;i <= ans[0];i ++)
      printf("%d",ans[i]);
    printf("\n");
  }
       return 0;
}
```

3.7 高斯消元回代法

```
/*
 *       o(n^3)算法
 *  主程序要做的事情
 *       ok = 1;
 *     solve(0);
 */

const int N = 1010;
double mat[N][N];
double ans[N];
int n;
const double EPS = 1e-7;
bool ok;

int dblcmp(double a) { if(fabs(a) < EPS) return 0; if(a < 0) return -1; return 1; }

void solve(int x) {
    if(x == n-1) {
        if(dblcmp(mat[x][x]) == 0) ok = 0;
        else ans[x] = mat[x][x+1] / mat[x][x];
        return;
    }
    //消去系数
    int i, j;
    for(i = x; i < n && dblcmp(mat[i][x]) == 0; i++);
    if(i == n) { ok = 0; return; }
    if(i != x) {
        double tmp[N];
        memcpy(tmp, mat[x], (n+1) * sizeof(double));
        memcpy(mat[x], mat[i], (n+1) * sizeof(double));
        memcpy(mat[i], tmp, (n+1) * sizeof(double));
    }
    for(i = x+1; i < n; i++) {
        if(dblcmp(mat[i][x]) == 0) continue;
        double m = mat[x][x] / mat[i][x];
        for(j = x; j < n + 1; j++)
            mat[i][j] = mat[i][j] * m - mat[x][j] * m;
    }
    solve(x+1);
    //计算 ans[x];
    double sum = mat[x][n];
```

```
    for(i = x+1; i < n; i++) sum -= mat[x][i] * ans[i];
    ans[x] = sum / mat[x][x];
}
```

3.8 数值计算

3.8.1 定积分计算

```
/*  Romberg 求定积分
    输入：积分区间[a,b]，被积函数 f(x,y,z)
    输出：积分结果

    f(x,y,z)示例：
    double f0( double x, double l, double t )
    {
        return sqrt(1.0+l*l*t*t*cos(t*x)*cos(t*x));
    }
*/

double Integral(double a, double b, double (*f)(double x, double y, double z), double eps,
                double l, double t)

double Romberg (double a, double b, double (*f)(double x, double y, double z), double eps,
                double l, double t)
{
#define MAX_N   1000

    int i, j, temp2, min;
    double h, R[2][MAX_N], temp4;

    for (i=0; i<MAX_N; i++) {
        R[0][i] = 0.0;
        R[1][i] = 0.0;
    }
    h = b-a;
    min = (int)(log(h*10.0)/log(2.0)); //h should be at most 0.1
    R[0][0] = ((*f)(a, l, t)+(*f)(b, l, t))*h*0.50;
    i = 1;
    temp2 = 1;
    while (i<MAX_N){
        i++;
        R[1][0] = 0.0;
        for (j=1; j<=temp2; j++)
```

```
            R[1][0] += (*f)(a+h*((double)j-0.50), l, t);
        R[1][0] = (R[0][0] + h*R[1][0])*0.50;
        temp4 = 4.0;
        for (j=1; j<i; j++) {
            R[1][j] = R[1][j-1] + (R[1][j-1]-R[0][j-1])/(temp4-1.0);
            temp4 *= 4.0;
        }
        if ((fabs(R[1][i-1]-R[0][i-2])<eps)&&(i>min))
            return R[1][i-1];
        h *= 0.50;
        temp2 *= 2;
        for (j=0; j<i; j++)
            R[0][j] = R[1][j];
    }
    return R[1][MAX_N-1];
}

double Integral(double a, double b, double (*f)(double x, double y, double z), double eps,
                double l, double t)
{
#define pi 3.1415926535897932

    int n;
    double R, p, res;

    n = (int)(floor)(b * t * 0.50 / pi);
    p = 2.0 * pi / t;
    res = b - (double)n * p;
    if (n)
        R = Romberg (a, p, f0, eps/(double)n, l, t);
    R = R * (double)n + Romberg( 0.0, res, f0, eps, l, t );

    return R/100.0;
}
```

3.8.2 多项式求根(牛顿法)

```
/*  牛顿法解多项式的根
    输入：多项式系数 c[]，多项式度数 n，求在[a,b]间的根
    输出：根
    要求保证[a,b]间有根
*/

double fabs( double x )
```

```
{
    return (x<0)? -x : x;
}

double f(int m, double c[], double x)
{
    int i;
    double p = c[m];

    for (i=m; i>0; i--)
        p = p*x + c[i-1];
    return p;
}

int newton(double x0, double *r,
           double c[], double cp[], int n,
           double a, double b, double eps)
{
    int MAX_ITERATION = 1000;
    int i = 1;
    double x1, x2, fp, eps2 = eps/10.0;

    x1 = x0;
    while (i < MAX_ITERATION) {
        x2 = f(n, c, x1);
        fp = f(n-1, cp, x1);
        if ((fabs(fp)<0.000000001) && (fabs(x2)>1.0))
            return 0;
        x2 = x1 - x2/fp;
        if (fabs(x1-x2)<eps2) {
            if (x2<a || x2>b)
                return 0;
            *r = x2;
            return 1;
        }
        x1 = x2;
        i++;
    }
    return 0;
}

double Polynomial_Root(double c[], int n, double a, double b, double eps)
{
    double *cp;
```

```
    int i;
    double root;

    cp = (double *)calloc(n, sizeof(double));
    for (i=n-1; i>=0; i--) {
        cp[i] = (i+1)*c[i+1];
    }
    if (a>b) {
            root = a; a = b; b = root;
    }
    if ((!newton(a, &root, c, cp, n, a, b, eps)) &&
        (!newton(b, &root, c, cp, n, a, b, eps)))
            newton((a+b)*0.5, &root, c, cp, n, a, b, eps);
    free(cp);
    if (fabs(root)<eps)
        return fabs(root);
    else
        return root;
}
```

3.8.3 周期性方程(追赶法)

```
/*  追赶法解周期性方程
    周期性方程定义：| a1 b1 c1 ...                    |          =   x1
                    |     a2 b2 c2 ...                |          =   x2
                    |          ...                    | *    X   =   ...
                    | cn-1 ...          an-1 bn-1     |          =   xn-1
                    | bn    cn                  an    |          =   xn
    输入：a[],b[],c[],x[]
    输出：求解结果 X 在 x[]中
*/

void run()
{
    c[0] /= b[0]; a[0] /= b[0]; x[0] /= b[0];
    for (int i = 1; i < N - 1; i ++) {
        double temp = b[i] - a[i] * c[i - 1];
        c[i] /= temp;
        x[i] = (x[i] - a[i] * x[i - 1]) / temp;
        a[i] = -a[i] * a[i - 1] / temp;
    }
    a[N - 2] = -a[N - 2] - c[N - 2];
    for (int i = N - 3; i >= 0; i --) {
        a[i] = -a[i] - c[i] * a[i + 1];
```

```
            x[i] -= c[i] * x[i + 1];
        }
        x[N - 1] -= (c[N - 1] * x[0] + a[N - 1] * x[N - 2]);
        x[N - 1] /= (c[N - 1] * a[0] + a[N - 1] * a[N - 2] + b[N - 1]);
        for (int i = N - 2; i >= 0; i --)
            x[i] += a[i] * x[N - 1];
}
```

4.排序

4.1 快速选择算法

```
//a 是 key 数组，idx 是前 K 个最小值排序的序号 l 是左下标  0, r 是右下标 a.length-1
private static void beat(float[] a, int[] ind, int l, int r, int k) {
            while (l < r) {
                float x = a[l + (int)(Math.random() * (r - l + 1))];
                int i = l;
                int j = r;
                while (i <= j) {
                    while (a[i] < x) i++;
                    while (a[j] > x) j--;
                    if (i <= j) {
                        float t = a[i];
                        a[i] = a[j];
                        a[j] = t;
                        int tt = ind[i];
                        ind[i] = ind[j];
                        ind[j] = tt;
                        i++;
                        j--;
                    }
                }
                if (k > i) {
                    l = i;
                } else if (k <= j) {
                    r = j;
                } else return;
            }
}
```

4.2 归并排序+逆序数的求取

```
#include <stdio.h>
#include <string.h>
//利用归并求逆序是指在对子序列 s1 和 s2 在归并时，若 s1[i]>s2[j]（逆序状况），
//则逆序数加上 s1.length-i,因为 s1 中 i 后面的数字对于 s2[j]都是逆序的。

const int MAXINT = 2000000000;

//返回：逆序数  参数：原数组 首 中 尾 临时数组(长度比原数组要大 2 个)
int merge(int a[], int p, int q, int r, int b[])
{
    int n1 = q-p+1, n2 = r-q;
    memcpy(b, a+p, n1*(sizeof(a[0])));
    b[n1] = MAXINT;
    memcpy( b+n1+1, a+q+1, n2*(sizeof(a[0])));
    b[n1+1+n2] = MAXINT;
    int i = 0, j = n1+1, ans = 0, ai = 0;
    for(ai=p; ai<=r; ai++) {
        if(b[i]<=b[j])
            a[ai] = b[i++];
        else
            a[ai] = b[j++],ans += n1-i;
    }
    return ans;
}

//返回：逆序数  参数：原数组 首 尾 临时数组(长度比 a 数组要大 2 个)
int mergesort(int a[], int p, int r, int b[])
{
    if(p<r)
    {
        int q = (p+r)/2;
        return mergesort(a, p, q, b)+mergesort(a, q+1, r, b)+merge(a, p, q, r, b);
    }
    return 0;
}

int main()
{
    int a[11] = {-42, 23, 6, 28, -100, 65537};
    int b[13], i;
    printf("%d\n", mergesort(a, 0, 5, b));
```

```
    for(i=0; i<10; i++)
        printf("%d\n", a[i]);
    return 0;
}
```

5.字符串

5.1 KMP 应用

```
#include<iostream>
#include<string>
using namespace std;
int next[10001];
void getnext(char s[10001])
{
  int i=0,j=-1;
  next[0]=-1;
  while(s[i]!='\0')
      {
       if(j==-1||s[i]==s[j])
            { i++; j++;
              //修正  if(s[i]==s[j]) next[i] = next[j]; else
              next[i]=j;
            }
       else j=next[j];
      }
}
int main()
{
  int n,i;
  char s[10005];
  cin>>n;
  cin>>s;
  strcat(s,"&");
  getnext(s);

  for(i=1;i<=n;i++)
    if(i%(i-next[i])==0)
      {
        if(i/(i-next[i])!=1) cout<<i<<" "<<i/(i-next[i])<<endl;
      }
```

```
 return 0;
}
```

5.2 后缀数组

```
/*
复杂度:o(nlogn)建立  o(1)查询任意两个后缀后缀的最长公共前缀
主程序要做的事情:
初始化 N, set_size, f[?][N],其中  ? > log(n)  修改 str 的类型
运行 suffix(str, Len)
本程序  : pku3261
*/
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
using namespace std;

inline int Min(int a, int b) {return a < b ? a : b;}
inline int Max(int a, int b) {return a > b ? a : b;}

const int N = 20001; //数组最大长度
const int set_size = 1000001; //字符集大小(比如 ASCII,256 就够了)

int str[N * 2]; //输入的串
int len; //串长度
int SA[N], Rank[N]; //SA:排序后的后缀的起始位置  Rank:每个后缀的排名(相同值相同排名)
int bucket[set_size], D[N]; //bucket :  桶  D :  临时 SA
int h[N]; //h[i] : suffix(i)与 suffix(SA[Rank[i]-1])的最长公共前缀
int height[N]; //height[i] :  得到有序后缀中 SA[i]与 SA[i-1]的最长公共前缀
int f[20][N]; //RMQ 预处理

inline int strsuf(int *p1, int *p2) { //得到 p1 与 p2 子串的最长公共前缀
    int step = 0;
    for (; *p1 == *p2; step++, p1++, p2++);
    return step;
}

inline int RMQ(int lt, int rt) { //区间[lt,rt]的最小 height 值
    int det = rt - lt + 1, k;
    for(k = 0; (1<<k) <= det; ++k); k--;
    return Min(f[k][lt], f[k][rt - (1<<k) + 1]);
}
```

```
inline void clear() {
    memset(str, 0, sizeof(str));
    memset(SA, 0, sizeof(SA));
    memset(Rank, 0, sizeof(Rank));
}

inline int suff(int k) {return (k>=0 && k<len) ? SA[k] : -1;} //返回排名第 k 的后缀
inline int rank(int k) {return (k>=0 && k<len) ? Rank[k] : -1;} //返回第 k 个后缀的排名
inline int lcp_str(int p1, int p2) { //返回 suffix(p1)和(p2)的最长公共前缀
    if (p1 < 0 || p2 < 0 || p1 >= len || p2 >= len) return -1;
    if (p1 == p2) return len - p1;
    return RMQ(Min(Rank[p1], Rank[p2]) + 1, Max(Rank[p1], Rank[p2]));
}
int repeation(int k) { //返回最大 K 重子串
    int ret = 0, i;
    for(i = k-1; i < len; ++i) { //k-1 个 height 的最小值
        ret = Max(ret, RMQ(i-k+2, i)); //RMQ(i-k+2, i)返回 suffix(p)和 suffix([Rank[p]]-K)之间的 K 个串
的最长公共前缀
    }
    return ret;
}

void Predone() { //桶排序
    memset(bucket, 0, sizeof(bucket));
     for (int i=0;i<len;i++) bucket[str[i]] ++;
     for (i=1;i<set_size;i++) bucket[i] += bucket[i-1];
     for (i=0;i<len;i++) Rank[i] = bucket[str[i]] - 1; //每个后缀的排名
     for (i=len-1;i>=0;i--) SA[--bucket[str[i]]] = i; //排序后的后缀的起始位置
};

void process() {
    Predone();
     int i, L;
     for (L=1;L<len;L*=2) {
        memset(bucket, 0, sizeof(bucket));
        for (i=0;i<len;i++) bucket[Rank[i+L]] ++;
        for (i=1;i<len;i++) bucket[i] += bucket[i-1];
        for (i=len-1;i>=0;i--) D[--bucket[Rank[i+L]]] = i;
        //D:k 前缀意义下的 suffix(i+L)后缀数组序(对 i 排序)
        memset(bucket, 0, sizeof(bucket));
        for (i=0;i<len;i++) bucket[Rank[i]] ++;
        for (i=1;i<len;i++) bucket[i] += bucket[i-1];
        for (i=len-1;i>=0;i--) SA[--bucket[Rank[D[i]]]] = D[i];
```

```
        //SA:2K 前缀意义下的 suffix(i)后缀数组序
        memcpy(D, Rank, sizeof(Rank));
        //重写 Rank
        Rank[SA[0]] = 0;
        for (i=1;i<len;i++) {
            //如果 2K 前缀不相同
            if (D[SA[i]] != D[SA[i-1]] || D[SA[i]+L] != D[SA[i-1]+L])
                Rank[SA[i]] = i;
            else Rank[SA[i]] = Rank[SA[i-1]];
        }
    }
};

void make_height() {
    for (int i=0;i<len;i++) {
        //h[i]记录的是 suffix(i)与 suffix(SA[Rank[i]-1])的最长公共前缀
        if (Rank[i] == 0) h[i] = 0; //第一的后缀 h[i] = 0;
        else if (i == 0 || h[i-1] <= 1) //第一个计算的 h[0],或者 h[i]公共前缀为 0
            h[i] = strsuf(&str[i], &str[SA[Rank[i]-1]]);
        else h[i] = h[i-1] - 1 +
            strsuf(&str[i+h[i-1]-1], &str[SA[Rank[i]-1]+h[i-1]-1]);
        //并且&str[SA[Rank[i]-1]一定与&str[SA[Rank[i]-1]]存在 h[i-1]-1 的公共前缀
    }
    //得到有序后缀的 height
    for (i=0;i<len;i++)
        height[i] = h[SA[i]];
};

void make_RMQ() {
    int i, L, dep = 1;
    for (i=0;i<len;i++) f[0][i] = height[i];
    for (L=1;L*2<len;L*=2, dep++)
        for (i=0;i+L<len;i++)
            f[dep][i] = Min(f[dep-1][i], f[dep-1][i+L]);
};

void suffix(int* pt, int Len) {
    len = Len;
    clear();
    memcpy(str, pt, len*sizeof(str[0]));
    process();
    make_height();
    make_RMQ();
};
```

```
int s[N];

int main() {

	freopen("t.in", "r", stdin);

	int n, K, i;
	scanf(" %d %d", &n, &K);

	for(i = 0; i < n; ++i) {
		scanf("%d", &s[i]);
	}
	suffix(s, n);
	printf("%d\n", repeation(K));
	return 0;
}
```

5.3 中缀表达式转后缀表达式

```
#include <stdio.h>
#include <string.h>

char stack[110];              //操作符栈
char infix[101];              //中缀表达式
char postfix[101];            //后缀表达式
int top;

int main()
{
	int i, index;             //index for posfix subscript
	while(1)
	{
		i = 0;
		top = -1;
		index = 0;
		memset(postfix, 0, sizeof(postfix));       //initiate postfix to empty string
		printf("please input the infixed Expression:\n");
		scanf("%s", infix);
		for(i = 0; infix[i]; ++i)
		{
			//push when numbers occur
			if(infix[i] >= '0' && infix[i] <='9') postfix[index++] = infix[i];
			//when operators occur
```

```
            else
            {
                //push when stack is empty
                if(top == -1) stack[++top] = infix[i];
                //do as common calculating sequences
                else
                {
                    switch(infix[i])
                    {
                    case '(':
                        stack[++top] = infix[i];break;
                    case ')':
                        while(stack[top] != '(')
                            postfix[index++] = stack[top--];
                        top--;break;
                    case '+':
                    case '-':
                        while(stack[top] != '(' && top!=-1)
                            postfix[index++] = stack[top--];
                        stack[++top] = infix[i];break;
                    case '*':
                    case '/':
                        if(stack[top] == '*' || stack[top] == '/')
                            postfix[index++] = stack[top--];
                        stack[++top] = infix[i];break;
                    default:;
                    }
                }
            }
        }
        //Pop all when Stack is still not empty
        while(top!=-1)
            postfix[index++] = stack[top--];
        printf("the postfixed Expression is as follows:\n%s\n\n", postfix);
    }
    return 0;
}
```

5.4 Firefighters 表达式求值

```
#include <stack>
#include <algorithm>
using namespace std;
```

```
const int N = 110;
const int INF = 123456789;

char st[N];
char stemp[] = "?+-*/()";
int pri[] = {0, 1, 1, 2, 2, 0, 0};

int answer;
int n;
int v[N];
bool f[N];

int find(char c) {
	int i;
	for(i=0; ; i++)
		if(stemp[i] == c)
			return i;
}
void init() {
	n = 0;
	f[n] = 1;
	v[n] = 5;
	n++;

	int i, k;
	for(i=0; st[i]; n++)
		if(isdigit(st[i])) {
			k = 0;
			while(isdigit(st[i]))
				k = k * 10 + st[i++] - '0';
			f[n] = 0;
			v[n] = k;
		}
		else {
			f[n] = 1;
			v[n] = find(st[i]);
			i++;
		}

	f[n] = 1;
	v[n] = 6;
	n++;
}
```

```
int OP(int i, int j, int op) {
    switch(stemp[op]) {
        case '+': return i+j;
        case '-':  return i-j;
        case '*': return i*j;
        case '/':
            if(j == 0)      return -INF;
            return i/j;
    }
    return 0;
}
int calculate() {
    stack <int> s1, s2;
    int i, j, k, op, ans;
    for(i=0; i<n; i++)
        if(!f[i])
            s1.push(v[i]);
        else {
            if(stemp[v[i]] == '(')
                s2.push(v[i]);
            else if(stemp[v[i]] == ')') {
                while(stemp[s2.top()] != '(') {
                    j = s1.top();
                    s1.pop();
                    k = s1.top();
                    s1.pop();
                    op = s2.top();
                    s2.pop();
                    ans = OP(k,j,op);
                    if(ans == -INF)   return ans;
                    s1.push(ans);
                }
                s2.pop();
            }
            else {
                while(!s2.empty() && pri[v[i]] <= pri[s2.top()]) {
                    j = s1.top();
                    s1.pop();
                    k = s1.top();
                    s1.pop();
                    op = s2.top();
                    s2.pop();
                    ans = OP(k,j,op);
                    if(ans == -INF)   return ans;
```

```
                        s1.push(ans);
                    }
                    s2.push(v[i]);
                }
            }
        ans = s1.top();
        s1.pop();
        return ans;
}
bool dfs(int k) {
        if(k == n) {
            return (calculate() == answer);
        }
        if(f[k] && !v[k]) {
            int i;
            for(i=1; i<=4; i++) {
                v[k] = i;
                if(dfs(k+1))
                    return true;
            }
            v[k] = 0;
            return false;
        }
        return dfs(k+1);
}

int main() {
        int Case;
        scanf("%d", &Case);
        while(Case--) {
            scanf("%s", st);
            scanf("%d", &answer);
            init();
            if(dfs(0))      printf("yes\n");
            else        printf("no\n");
        }
        return 0;
}
```

6.博弈

6.1 博弈的 AB 剪枝

```
//注意 AB 剪枝不要与 DP 混用
int search(int p, int sta) { //假定现在在计算 S 的子局面 S2，sta 则是 S1 的状态值
    if(p == 0) { //假定 S 局面取小，如果 S2 比 S1 大 则无用 即只要 S2 的子局面有一个比 STA 大 S2 就无用了 退出
        int now = -MAXINT; //S2 局面取大 初始化为最小
        for( every possible action ) {
            //go this action;
            int next = search(1, now);
            now = Max(now, next);
            //cancel this cation
            if(now > sta) break; //S2 的某个子局面出现了比 sta 还小的 退出
        }
        return now;
    }
    else { //假定 S 局面取大，如果 S2 比 S1 小 则无用 即只要 S2 的子局面有一个比 STA 小 S2 就无用了 退出
        int now = MAXINT;
        for( every possible action ) {
            //go this action;
            int next = search(1, now);
            now = Min(now, next);
            //cancel this action
            if(now < sta) break;
        }
        return now;
    }
}
```

6.1.1 取石子

If gi is the Sprague-Grundy function of Gi ,
i = 1, . . . , n, then
G = G1 +· · · + Gn has Sprague-Grundy function
g(x1, . . . , xn) = g1(x1) ⊕ · · · ⊕ gn(xn).

```
#include<stdio.h>
```

```
#include<string.h>
#include<algorithm>
using namespace std;
int k,a[100],f[10001];
int mex(int p)
{   int i,t;
    bool g[101]={0};
    for(i=0;i<k;i++)
    {
        t=p-a[i];
        if(t<0)
            break;
        if(f[t]==-1)
            f[t]=mex(t);
        g[f[t]]=1;
    }
    for(i=0;;i++)
    {
        if(!g[i])
            return i;
    }
}
int main()
{   int n,i,m,t,s;
    while(scanf("%d",&k),k)
    {   for(i=0;i<k;i++)
            scanf("%d",&a[i]);
        sort(a,a+k);
        memset(f,-1,sizeof(f));
        f[0]=0;
        scanf("%d",&n);
        while(n--)
        { scanf("%d",&m);
            s=0;
            while(m--)
            {
                scanf("%d",&t);
                if(f[t]==-1)
                    f[t]=mex(t);
                s=s^f[t];
            }
            if(s==0)
                printf("L");
            else
```

```
                printf("W");
            }
            printf("\n");
        }
}
```

6.2 博弈 SG 函数 局势分割

```
#include <algorithm>
using namespace std;

const int N = 1010;

int n;
int adj[N][N], top[N];
int sg[N];

void input() {
    int i, j;
    for(i=0; i<n; i++) {
        scanf("%d", top+i);
        for(j=0; j<top[i]; j++)
            scanf("%d", adj[i]+j);
    }
}
void calSg(int k) {
    if(sg[k] != -1) return;
    int i;
    bool f[N] = {0};
    for(i=0; i<top[k]; i++) {
        calSg(adj[k][i]);
        f[sg[adj[k][i]]] = true;
    }
    for(sg[k]=0; f[sg[k]]; sg[k]++);
}
void init() {
    int i;
    memset(sg, -1, sizeof(sg));
    for(i=0; i<n; i++)
        calSg(i);
}
int main() {
    freopen("in", "r", stdin);
    int k, i, ans;
```

```
    while(scanf("%d", &n) != EOF) {
        input();
        init();
        while(true) {
            scanf("%d", &k);
            if(k == 0)    break;
            ans = 0;
            while(k--) {
                scanf("%d", &i);
                ans ^= sg[i];
            }
            printf("%s\n", ans ? "WIN":"LOSE");
        }
    }
    return 0;
```

7.数据结构

7.1 TRIE

```
/*
主程序要做的事:
调用 trieBuild();
初始化 N;
*/
const int MAX = 500000;

int up;
struct Node {
    int c[26]; //26 仅限 26 个小写字幕(或大写)
    bool isword;
    Node() {
        up++;
        isword = false;
        memset(c, 0, sizeof(c));
    }
}trie[MAX];

void trieBuild() {
    up = 0;
    trie[0] = Node(false);
```

```
}

void trieAdd(char * s) {
	int i, now = 0;
	for(i = 0; s[i]; ++i) {
		int p = s[i]-'a';
		if(trie[now].c[p] == 0) {
			trie[now].c[p] = up;
			trie[up] = Node(false);
		}
		now = trie[now].c[p];
	}
	trie[now].isword = true;
}

bool trieFind(char * s) {
	int i, now = 0;
	for(i = 0; s[i]; ++i) {
		int p = s[i]-'a';
		if(trie[now].c[p] == 0)
			return false;
		now = trie[now].c[p];
	}
	return trie[now].isword;
}

/*
//PKU2513 并查集判联通+Trie 判字符串
#include <string.h>
#include <stdio.h>
const int N = 500010;
int p[N], deg[N];
int up, top;
int Root(int x) {
	if (p[x] == x)
		return x;
	return p[x] = Root(p[x]);
}
void Union(int a,int b) {
	if (Root(a) != Root(b)) {
		if(a < b) p[b] = a;
		else p[a] = b;
	}
}
```

```
struct Node {
	int c[26];
	bool isword;
	int idx;
}trie[N*26];

void trieBuild() {
	trie[0].idx = -1;
	trie[0].isword = false;
	memset(trie[0].c, 0, sizeof(trie[0].c));
	up = 1;
}

void trieAdd(char * s) {
	int i, now = 0;
	for(i = 0; s[i]; ++i) {
		int p = s[i]-'a';
		if(trie[now].c[p] == 0) {
			memset(trie[up].c, 0, sizeof(trie[up].c));
			trie[up].isword = false;
			trie[now].c[p] = up++;
		}
		now = trie[now].c[p];
		if(!s[i+1]) {
			trie[now].isword = true;
			trie[now].idx = top++;
		}
	}
}

int trieFind(char * s) {
	int i, now = 0;
	for(i = 0; s[i]; ++i) {
		int p = s[i]-'a';
		if(trie[now].c[p] == 0)
			return -1;
		now = trie[now].c[p];
		if(!s[i+1]) return trie[now].idx;
	}
	while(1);
}

int go(char * s) {
	int ans;
```

```
    if((ans = trieFind(s)) == -1) {
        trieAdd(s);
        ans = top-1;
        p[ans] = ans;
        deg[ans] = 0;
    }
    return ans;
}

int main() {
//  freopen("t.in", "r", stdin);
    char a[12], b[12];
    top = 0; up = 1;
    while(scanf("%s %s", a, b) != EOF) {
        int ida = go(a);
        int idb = go(b);
        Union(ida, idb);
        deg[ida]++;
        deg[idb]++;
    }
    int i, ori = Root(0);
    for(i = 0; i < top; ++i) {
        if(Root(i) != ori)
            break;
    }
    if(i < top) { printf("Impossible\n"); return 0; }
    int cnt = 0;
    for(i = 0; i < top; ++i) if(deg[i] % 2)
        cnt++;
    if(cnt == 0 || cnt == 2) printf("Possible\n");
    else printf("Impossible\n");

    return 0;
}
*/
/*PKU1204 Word Puzzle
#include <stdio.h>
#include <string.h>
const int MAX = 100000;
const int N = 1010;
int kx[] = {-1, -1, 0, 1, 1, 1, 0, -1};
int ky[] = {0, 1, 1, 1, 0, -1, -1, -1};

int up;
```

```
struct Node {
    int c[26]; //26 仅限 26 个小写字幕(或大写)
    bool isword;
    Node() {
        isword = false;
        memset(c, 0, sizeof(c));
    }
}trie[MAX];
void trieBuild() {
    up = 0;
    trie[up++] = Node();
}
void trieAdd(char * s) {
    int i, now = 0;
    for(i = 0; s[i]; ++i) {
        int p = s[i]-'A';
        if(trie[now].c[p] == 0) {
            trie[now].c[p] = up;
            trie[up++] = Node();
        }
        now = trie[now].c[p];
    }
    trie[now].isword = true;
}
char brd[N][N];
char w[N][N];
int n, m, K;
int X[N], Y[N], D[N];
void Go(int x, int y, int d) {
    if(x == 0 && y == 1 && d == 2)
        d = 2;
    char s[N];
    int xx = x, yy = y, i, now = 0, j;
    for(i = 0; xx >= 0 && yy >= 0 && xx < n && yy < m; ++i) {
        s[i] = brd[xx][yy];
        int p = s[i]-'A';
        if(trie[now].c[p] == 0)
            return;
        now = trie[now].c[p];
        if(trie[now].isword) {
            s[i+1] = 0;
            for(j = 0; j < K; ++j) {
                if(strcmp(s, w[j]) == 0) {
                    X[j] = x, Y[j] = y, D[j] = d;
```

```
                        break;
                    }
                }
            }
            xx += kx[d], yy += ky[d];
        }
    }
    int main() {
        //freopen("t.in", "r", stdin);
        //freopen("t6.out", "w", stdout);

        int i, j, k;
        scanf("%d%d%d", &n, &m, &K);
        for(i = 0; i < n; ++i) scanf("%s", brd[i]);
        trieBuild();
        for(i = 0; i < K; ++i) {
            scanf("%s", w[i]);
            if(strlen(w[i]) <= 1000)
                trieAdd(w[i]);
        }
        for(i = 0; i < n; ++i)
            for(j = 0; j < m; ++j)
                for(k = 0; k < 8; ++k)
                    Go(i, j, k);
        for(i = 0; i < K; ++i) {
            printf("%d %d %c\n", X[i], Y[i], D[i]+'A');
        }
        return 0;
    }
    */
```

7.2 线段树

```
/*
主程序中设置一个全局变量 up = 0;
注意 这个线段树只能查询一段区间的总覆盖情况,不能得知一段上覆盖多少线段
*/
const int N = 50010;

struct ST {int i,j,m,l,r,c;} st[2*N]; //区间宽度的 2 倍
int up;

void bd(int d, int x, int y) {
    st[d].i = x, st[d].j = y, st[d].m = (x+y)/2, st[d].c = 0;
```

```
    if(x < y-1) {
        st[d].l = ++up; bd(up, x, st[d].m);
        st[d].r = ++up; bd(up, st[d].m, y);
    }
}

void ins(int d, int x, int y) { //不能插入 x == y
    if(st[d].c) return;
    if(x <= st[d].i && y>=st[d].j) st[d].c++; //注意  这里并没有实时更新下面的子节点
    else {
        if(x < st[d].m) ins(st[d].l, x, y);
        if(y > st[d].m) ins(st[d].r, x, y);
    }
}

void del(int d, int x, int y) {
    if(x <= st[d].i && y >= st[d].j) st[d].c--;
    else {
        if(x < st[d].m) del(st[d].l, x, y);
        if(y > st[d].m) del(st[d].r, x, y);
    }
}

int get(int d, int x, int y) {
    if(st[d].c > 0)      return st[d].j - st[d].i;
    if(st[d].i == st[d].j-1) return 0;
    int ret = 0;
    if(x < st[d].m) ret += get(st[d].l, x, y);
    if(y > st[d].m) ret += get(st[d].r, x, y);
    return ret;
}
/*
#include <algorithm>
#include <vector>
#include <set>
using namespace std;

const int N = 40010;
int a[N], b[N], h[N];
int nb;
struct E { int x, h, f; } line[2 * N];
vector<int> mapy;
int ny;
int up;
```

```
struct Node {
  int i, j, cnt;
  Node * lcd, * rcd;
  void Build(int, int);
  void Insert(int, int);
  void Delete(int, int);
  int GetLen();
}node[N * 4];

void Node::Build(int l, int r) {
  i = l;
  j = r;
  cnt = 0;
  if(r - l > 1) {
    lcd = &node[++up];
    lcd ->Build(l, (l+r)>>1);
    rcd = &node[++up];
    rcd->Build((l+r)>>1, r);
  }
  else {
    lcd = NULL;
    rcd = NULL;
  }
}

void Node::Insert(int l, int r) {
  if(l <= mapy[i] && r >= mapy[j])
    cnt++;
  else {
    if(l < mapy[(i+j)>>1])
          lcd->Insert(l, r);
    if(r > mapy[(i+j)>>1])
          rcd->Insert(l, r);
  }
}

void Node::Delete(int l, int r) {
  if(l <= mapy[i] && r >= mapy[j]) cnt--;
  else {
    if(l < mapy[(i+j)>>1]) lcd->Delete(l, r);
    if(r > mapy[(i+j)>>1]) rcd->Delete(l, r);
  }
}
```

```
int Node::GetLen() {
  if(cnt > 0) return mapy[j] - mapy[i];
  if(lcd == NULL) return 0;
  return lcd->GetLen() + rcd->GetLen();
}

void solve() {
	up = 0;
	node[0].Build(0, ny-1);
	int i;
	__int64 sum = 0;
	for(i = 0; i < 2 * nb; ++i) {
		int x = line[i].x, h = line[i].h, f = line[i].f;
		if(f == 0)	{
			node[0].Insert(0, h);
		}	else {
			node[0].Delete(0, h);
		}

		int len = node[0].GetLen();
		if(i < 2 * nb -1)
			sum += (__int64)len * (__int64)( line[i + 1].x - line[i].x );
	}
	printf("%I64d\n", sum);
}

bool operator<(const E& a, const E& b) {
	return a.x < b.x;
}

int main() {
	scanf("%d", &nb);
	set<int> sx, sy;
	int i;
	for(i = 0; i < nb; ++i) {
		scanf("%d %d %d", &a[i], &b[i], &h[i]);
		sx.insert(a[i]);
		sx.insert(b[i]);
		sy.insert(h[i]);
		line[i * 2].x = a[i];
		line[i * 2].h = h[i];
		line[i * 2].f = 0;
		line[i * 2 + 1].x = b[i];
```

```
            line[i * 2 + 1].h = h[i];
            line[i * 2 + 1].f = 1;
        }
        ny = sy.size() + 1;
        mapy.resize(ny);
        set<int>::iterator it;
        mapy[0] = 0;
        for(it = sy.begin(), i = 1; it != sy.end(); ++it, ++i)
            mapy[i] = *it;

        sort(line, line + 2 * nb);

        solve();

        return 0;
}
*/
```

7.3 并查集

```
#include<iostream>
using namespace std;
int rank[1001],x,y;
int v[1001];
//初始化 x 集合
void make_set(int x)
{
    v[x]=x;
    rank[x]=0;
}

//查找 x 所在的集合
int find_set(int x)
{
    if(v[x]!=x) v[x]=find_set(v[x]);
    return v[x];
}

//合并 x，y 所在的集合，用到路径压缩，按秩合并
void Union(int x,int y)
{
    x=find_set(x);
    y=find_set(y);
    if(rank[x]>rank[y])
```

```
            v[y]=x;
        else if(rank[x]<rank[y])
            v[x]=y;
        else if(rank[x]==rank[y])
        {
            v[x]=y;
            rank[y]++;
        }
}

int main()
{

  return 0;
}
```

7.4 树状数组

```
class TA
{
public:
    int c[N],n;
    void clear()
    {
        memset(this,0,sizeof(*this));
    }
    int lowbit(int x) //返回最低一位的权
    {
        return x&(x^(x-1));
    }
    void change(int i,int d) //n 是点的最大数目
    {
        for (;i<=n;i+=lowbit(i))            c[i]+=d;
    }
    int sum(int i) //sum 代表从 0->p 的加合  如果需要(a,b)  则运行 sum(b) - sum(a-1)
    {
        int t;
        for (t=0;i>0;i-=lowbit(i))            t+=c[i];
        return t;
    }
};

/*
#include <iostream>
```

```
#include <vector>
using namespace std;

const int N = 100100;
vector<int> head[N];
int f[N], b[N], n, C[2*N];
bool chk[N], has[N];
int Time;

inline int lowbit(int x) { //返回最低一位的权
    return x&(x^(x-1));
}

void add(int p, int x) {
    while(p < Time) {
        C[p] += x;
        p += lowbit(p);
    }
}

int sum(int p) {
    int ret = 0;
    while(p > 0) {
        ret += C[p];
        p -= lowbit(p);
    }
    return ret;
}

void DFS(int x) {
    chk[x] = true;
    b[x] = Time++;
    int i, j;
    for(i = 0; i < head[x].size(); ++i) {
        j = head[x][i];
        if(!chk[j])
            DFS(j);
    }
    f[x] = Time++;
}

int main() {
    int i, u, v, nq;
    char ch[2];
```

```
    scanf("%d", &n);
    for(i = 0; i < n-1; ++i) {
        scanf("%d %d", &u, &v);
        u--; v--;
        head[u].push_back(v);
        head[v].push_back(u);
    }

    memset(chk, 0, sizeof(chk));
    Time = 0;
    DFS(0);

    memset(C, 0, sizeof(C));
    for(i = 0; i < n; ++i) {
        has[i] = true;
        add(f[i], 1);
    }

    scanf("%d", &nq);
    while(nq--) {
        scanf("%s %d", ch, &u); --u;
        if(ch[0] == 'Q') {
            printf("%d\n", sum(f[u]) - sum(b[u]-1));
        }
        else if( has[u] == 0) {
            add(f[u], 1);
            has[u] = 1;
        }
        else {
            add(f[u], -1);
            has[u] = 0;
        }
    }

    return 0;
}

*/
```

7.5 点树

```
//pku1769
/*
  * trival DP n^2
```

```
 * 考虑到转移的时候选择的是一段内的最小 dp 值，运用点树可以解决
 */
#include <string.h>
#include <stdio.h>

const int N = 50010;
const int MAXINT = 1000000000;

int n, l;

struct ST {int i,j,m,l,r,c;} st[2*N];
int up, cnt;

void bd(int d, int x, int y) {
    st[d].i = x, st[d].j = y, st[d].m = (x+y)/2, st[d].c = MAXINT;
    if(x < y) {
        st[d].l = ++up; bd(up, x, st[d].m);
        st[d].r = ++up; bd(up, st[d].m+1, y);
    }
}

void ins(int d, int x, int c) {
    if(c < st[d].c)
        st[d].c = c;
    if(st[d].i != st[d].j) {
        if(x <= st[d].m)
            ins(st[d].l, x, c);
        else
            ins(st[d].r, x, c);
    }
}

int getmin(int d, int x, int y) {
    if(x <= st[d].i && y >= st[d].j)
        return st[d].c;
    int min = MAXINT;
    if(x <= st[d].m) {
        int now = getmin(st[d].l, x, y);
        if(now < min) min = now;
    }
    if(y > st[d].m) {
        int now = getmin(st[d].r, x, y);
        if(now < min) min = now;
    }
```

```
    return min;
}

int main() {
    int i, a, b;
    up = 0;
    scanf("%d %d ", &l, &n);
    bd(0, 1, l);
    ins(0, 1, 0);
    int max = 0;
    for(i = 0; i < n; ++i) {
        scanf("%d%d", &a, &b);
        if(a < b) {
            int min = getmin(0, a, b-1);
            ins(0, b, min+1);
        }
    }
    printf("%d\n", getmin(0, l, l));
    return 0;
}
```

7.6 STL

栈(Stack)

栈可以用向量(vector)、线性表(list)或双向队列(deque)来实现：

```
stack<vector<int>> s1;
stack<list<int>    >    s2;
stack<deque<int>> s3;
```

其成员函数有“判空(empty)”、“尺寸(Size)”、“栈顶元素(top)”、“压栈(push)”、“弹栈(pop)”等。

队列(Queue)

队列可以用线性表(list)或双向队列(deque)来实现(注意 vector container 不能用来实现 queue，因为 vector 没有成员函数 pop_front!)：

```
queue<list<int>>    q1;
queue<deque<int>>    q2;
```

其成员函数有“判空(empty)”、“尺寸(Size)”、“首元(front)”、“尾元(backt)”、“加入队列(push)”、“弹出队列(pop)”等操作。

优先级队列(Priority Queue)

优先级队列可以用向量(vector)或双向队列(deque)来实现(注意 list container 不能用来实现 queue，因为 list 的迭代器不是任意存取 iterator，而 pop 中用到堆排序时是要求 random access iterator 的!)：

```
priority_queue<vector<int>, less<int>>      pq1;      // 使用递增 less<int>函数对象排序
priority_queue<deque<int>, greater<int>>    pq2;      // 使用递减 greater<int>函数对象排序
```

其成员函数有“判空(empty)”、“尺寸(Size)”、“栈顶元素(top)”、“压栈(push)”、“弹栈(pop)”等。

```
vector<int> v(3);
    v[0] = 5;
    v[1] = 2;
    v[2] = 7;

    vector<int>::iterator first = v.begin();
    vector<int>::iterator last = v.end();

    while (first != last)
    cout << *first++ << " ";
```

这四个 vector 对象是相等的，可以用 operator==来判断。

其余常用的 vector 成员函数有：

empty()：判断 vector 是否为空
front()：取得 vector 的第一个元素
back()：取得 vector 的最后一个元素
pop_back()：去掉最后一个元素
erase()：去掉某个 iterator 或者 iterator 区间指定的元素

7.7 离散化

```
#include <stdio.h>

struct SEG
{
    int x1, x2;
};

int n, flag;
SEG seg[10005];

void lisan(int l, int r, int j)
{
    if(j == n)
    {
        if(l < r) flag = 1;
        return;
    }
    if(l >= seg[j].x1 && r <= seg[j].x2) return;
    else
    {
```

```
            if(l >= seg[j].x2) lisan(l, r, j + 1);
            else if(r <= seg[j].x1) lisan(l, r, j + 1);
            else
            {
                if(l < seg[j].x1) lisan(l, seg[j].x1, j + 1);
                if(r > seg[j].x2) lisan(seg[j].x2, r, j + 1);
            }
        }
}

int main()
{
    int ca, ans;
    scanf("%d", &ca);
    while(ca--)
    {
        scanf("%d", &n);
        for(int i = 0;i < n;i++)
        {
            scanf("%d%d", &seg[i].x1, &seg[i].x2);
            seg[i].x1--;
        }
        ans = 0;
        for(int i = n - 1;i >= 0;i--)
        {
            flag = 0;
            lisan(seg[i].x1, seg[i].x2, i + 1);
            if(flag == 1) ans++;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

8.图论

8.0 2-SAT

```
//pku3683
#include<stdio.h>
#include<algorithm>
```

```
#define max(a,b) ( a>b ? a:b )
#define min(a,b) ( a<b ? a:b )
using namespace std;

inline int rev ( int x ) { return x ^ 1 ; };

const int maxn = 2000;

int n;
int st[maxn], ed[maxn];

int mk[maxn], low[maxn], depth[maxn], choice [ maxn ];

int find( int x ) { if( x != low[x] ) return low[x] = find(low[x]); return low[x]; };

void dfs ( int u, int d = 0 )
{
	mk[u] = -1; depth[u] = d; low[u] = u;
	for( int v = 0; v < n; ++ v )
		if ( ( u >> 1 ) != ( v >> 1 ) && max( st[u], st[rev(v)] ) < min( ed[u], ed[rev(v)] ) )
		{
		if ( !mk[v] ) dfs(v, d+1);
		if( mk[low[v]] == -1 && depth[low[u]] > depth[low[v]] ) low[u] = low[v];
		}
	if( !choice[rev(u)] ) choice[u] = 1;
	if( mk[rev(u)] == -1 ) mk[u] = 2; else mk[u] = 1;
}

void solve ()
{
	int i;
	memset( mk, 0, sizeof( mk ) );
	memset( choice, 0, sizeof( choice ) );
	for( i = 0; i < n; ++ i ) if( !mk[i] ) dfs( i );
	for( i = 0; i < n; ++ i ) if( mk[i] == 2 && depth[ find ( i ) ] < depth[ rev(i) ] )
	{
		printf("NO\n"); return ;
	}
	printf("YES\n");
	for( i = 0; i < n; ++ i ) if( choice[i] )
		printf("%02d:%02d %02d:%02d\n", st[i]/60, st[i]%60, ed[i]/60, ed[i]%60);
}

int main ()
```

```
{
    scanf("%d", &n);
    for( int i = 0; i < n; ++ i )
    {
        int a, b, v;
        scanf("%d:%d", &a, &b);
        st[i*2] = a * 60 + b;
        scanf("%d:%d", &a, &b);
        ed[i*2+1] = a * 60 + b;
        scanf("%d", &v);
        ed[i*2] = st[i*2] + v;
        st[i*2+1] = ed[i*2+1] - v;
    }
    n *= 2;
    solve ();
    return 0;
}
```

8.1 有向图强连通分量

```
#include <string.h>
#include <stdio.h>
#include <algorithm>
using namespace std;
const int MAXINT = 2000000000;
const int N = 10000;
const int M = 50000;
int head[N], set[N], post[N], head2[N];
int no[M], next[M], no2[M], next2[M];
bool vst[N]; //可多次重复利用
int n, m, cnt;    //cnt  也是这样

inline void read(int &data) {
    char ch = getchar();
    while (ch < '0' || ch > '9') ch = getchar();
    data = 0;
    do{
        data = data*10 + ch-'0';
        ch = getchar();
    }while (ch >= '0' && ch <= '9');
}

void init() {
    int i, a, b;
    //input
```

```
    scanf("%d%d", &n, &m);
    memset(head, -1, sizeof(head));
    memset(head2, -1, sizeof(head2));
    for(i = 0; i < m; i++) {
        read(a); read(b);
        a--; b--;
        no[i] = b;
        next[i] = head[a];
        head[a] = i;
    }
    //pretreatment
}
//有向图的 DFS 要分成多步进行 dfs 这里没有判连通 因为不连通一定有 2 个以上的极大强连通子图
void DFS(int x) {
    vst[x] = 1;
    int p = head[x];
    for(; p >= 0; p = next[p])
        if(!vst[no[p]])
            DFS(no[p]);
    post[cnt] = x; //我们只需要按照完成时间排序即可 并不需要严格按照结束时间
    cnt++;
}

void reverse() {
    int i, t = 0, p, j = 0;
    for(i = 0; i<n; i++) {
        for(p = head[i]; p >= 0; p = next[p]) { //i->no[p]
            no2[j] = i;
            next2[j] = head2[no[p] ];
            head2[no[p] ] = j++;
        }
    }
}

void DFS2(int x) {
    vst[x] = 1;
    set[x] = cnt;
    int p;
    for(p = head2[x]; p >= 0; p = next2[p])
        if(!vst[no2[p] ])
            DFS2(no2[p]);
}

void work() {
```

```
    int i, j;
    memset(vst, 0, sizeof(vst));
    cnt = 0; //cnt 充当了记录时间的作用
    for(i = 0; i<n; i++)
        if(!vst[i])
            DFS(i);
    reverse();
    memset(vst, 0, sizeof(vst));
    cnt = 0;
    for(i = n-1; i>=0; i--)
        if(!vst[post[i]]) {
            DFS2(post[i]);
            cnt++; //此处作为 scc 的 id 值使用
        }
    //去除错误的 scc
    memset(vst, 0, sizeof(vst));
    int p;
    for(i = 0; i<n; i++)
        if(!vst[set[i]])
            for(p = head[i]; p >= 0; p = next[p])
                if(set[no[p] ] != set[i]) {
                    vst[set[i] ] = true;
                    break; //有引申出去的点 则当前 scc 错误!
                }

    j = -1;
    for(i = 0; i<cnt; i++)
        if(!vst[i]) {
            if(j >= 0) {
                printf("0\n"); //2 连通分量
                return;
            }
            j = i;
        }
    cnt = 0;
    for(i = 0; i<n; i++)
        if(set[i] == j)
            cnt++;
    printf("%d\n", cnt);
}

int main() {
//    freopen("t.in", "r", stdin);
    init();
```

```
    work();
    return 0;
    //数据测试充分？去掉调试语句,freopen,还原数据大小了吗？__int64 to long long?
}
```

8.2 寻找 Euler 回路

```
//注意 这里的 Euler 回路寻找方法是记录边形成一个序列 path
void find_path(int loc, int e) {
    for(int lv = 0; lv < node[loc].total; lv++) {
        int ee = node[loc].ed[lv].e;
        if(!touched[ee]) {
            touched[ee] = 1;
            find_path(node[loc].ed[lv].v, ee);
        }
    }
    path[plen++] = e;
}
```

8.3 拓扑排序

```
//要求 有 ind 即入度数组
void TOPO() {
    int idx[N], stack[N], top = 0, i, cnt = 0;
    memset(chk, 0, sizeof(chk));
    while(1) {
        if(top == 0) {
            for(i = 0; i < nv; ++i) {
                if(!chk[i] && ind[i] == 0)
                    break;
            }
            if(i >= nv) break;
            stack[top++] = i;
        }
        idx[cnt++] = i;
        int cur = i;
        chk[i] = 1;
        for(i = 0; i < to[cur].size(); ++i) {
            ind[to[cur][i]]--;
            if(ind[to[cur][i]] == 0)
                stack[top++] = to[cur][i];
```

```
        }
    }
}
```

8.4 差分约束系统

```
//差分约束系统 一部分奶牛最多隔开 D，一部分最少隔开 D
//无解输出-1
//1 和 N 奶牛可以隔任意远 输出-2
//其他清除输出 1 和 N 的最大距离
//求出来的标号具有最大的性质 如果要求最小 则必须二分枚举 run 一次是否有解

#include <string.h>
#include <stdio.h>
#include <iostream.h>
#include <math.h>
const int MAXINT = 1000000000;
const int N = 1000;
const int M = 10000;
struct node{int u, v, w; void set(int a, int b, int c) { u = a;v = b; w = c;}}e[M*4];
int dist[N];
int nv, s1, s2, ne;

void init() {
    int i, u, v, w, j = 0;
    scanf("%d%d%d", &nv, &s1, &s2);
    for(i = 0; i < s1; i++) {
        scanf("%d%d%d", &u, &v, &w);
        u--; v--;
        if(u>v)e[j++].set(v, u, w);
        else e[j++].set(u, v, w);
    }
    for(; i<s1+s2; i++) {
        scanf("%d%d%d", &u, &v, &w);
        u--; v--;
        if(u>v)  e[j++].set(u, v, -w);
        else e[j++].set(v, u, -w);
    }
    for(i = 1; i<nv; i++)   e[j++].set(i, i-1, 0);
    ne = j;
}

void work() {
```

```
    int i, j, u, v, w;
    dist[0] = -MAXINT; //放到最大的情况
    for(i = 1; i < nv; i++)
        dist[i] = MAXINT;
    for(i = 0; i<nv; i++)
    {
        bool change = false;
        for(j = 0; j<ne; j++) //F(v)-F(u) < w
        {
            u = e[j].u;
            v = e[j].v;
            w = e[j].w;
            if(dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                change = true;
            }
        }
        if(!change) break;
    }
    if(i == nv) {printf("-1\n"); return;}
    else if(abs(dist[nv-1]-dist[0]) == 2*MAXINT) {printf("-2\n"); return;}
    printf("%d\n", abs(dist[nv-1]-dist[0]));
}

int main() {
    init();
    work();
    return 0;
}
```

8.5 笛卡尔树

```
/*
 * 笛卡儿树
 *
 * 1.定义:    每个节点有 2 的属性 a,b  树的性质是满足 a 的 2 叉排序树规则,b 的堆规则(现拟定小根堆)
 * 2.实现:
 *    1.组织一个栈,按照 a 下标从小到大到节点依次压栈
 *    2.当压入节点 A 的 b 值大于栈顶元素 B 的 b 值得时候,A 作为 B 的右儿子.(满足堆规则)
 *    3.否则,将 B 弹出,继续判断,最后一个弹出的节点作为 A 的左儿子(a 下标小,满足二叉树规则)
 * 3.思想与应用:
 *    1.任何时候,栈中的第一个元素都是该树的根.
 *    2.RMQ->LCA:  将 RMQ 的数组下标作为 a 值,将其 key 作为 b 值,不难看出,对 RMQ 的数组转化
```

成 LCA 是相等的
 * 4.题目:
 *   PKU2201,1785
 */

//PKU2201

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
using std::sort;

const int N = 50010;

struct Node {
    int a, b, idx;
}A[N], st[N];

int L[N], R[N], T[N];

bool operator<(const Node& a, const Node& b) {
    return a.a < b.a;
}

int main() {

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

    int n, i;
    scanf("%d", &n);

    memset(L, -1, sizeof(L));
    memset(R, -1, sizeof(R));
    memset(T, -1, sizeof(T));

    for(i = 0; i < n; ++i) {
        scanf("%d%d", &A[i].a, &A[i].b);
        A[i].idx = i;
    }

    sort(A, A+n);

    int top = 0;
    for(i = 0; i < n; ++i) {
```

```
            int k = top-1;
            while(k >= 0 && A[i].b < st[k].b) {
                k--;
            }
            if(k >= 0) {
                R[st[k].idx] = A[i].idx;
                T[A[i].idx] = st[k].idx;
            }
            if(k != top-1) {
                L[A[i].idx] = st[k+1].idx;
                T[st[k+1].idx] = A[i].idx;
            }
            st[k+1] = A[i];
            top = k+2;
        }

        printf("YES\n");
        for(i = 0; i < n; ++i) {
            printf("%d %d %d\n", T[i]+1, L[i]+1, R[i]+1);
        }

        return 0;
}
```

8.6 LCA 和 RMQ

```
//LCA 的 tarjan 算法
memset(color, 0, sizeof(color));
void LCA(int u) {
    int i;
    p[u] = u;
    ancestor[u] = u;
    for(i = 0; i < v[u].size(); i++) {
        LCA(v[u][i]);
        int y = Find_set(v[u][i]); //是否可以改成 p[v[u][i]] = u;
        if(u != y) p[y] = u;
    }
    color[u] = 1;
    for(i = 0; i<q[u].size(); i++) {
        if(color[q[u][i]] == 1)
            //如果 q[u][i]在 u 的子树下面  则其公共祖先是 u  否则是 ancestor[Find_set(q[u][i])];
            num[ancestor[Find_set(q[u][i])] ]++;
```

```
    }
}

//LCA 的 ST 算法 <O(NlogN), O(logN)>
/*
 1.输入 记录父子 找出根节点 做 DFS 确定高度
 2.DP 做 P[i][j]
 3.查询 a,b 的公共祖先
   1.若 l[a] < l[b] 交换
   2.若 l[a] > l[b] bs 找出和 l[b]相等的 c
   3.BS 求出 LCA(c,b)
*/

#include <stdio.h>
#include <string.h>
#include <vector>
using namespace std;

const int N = 10001;
int p[N], l[N];
int height, nv;
int P[N][20];
vector<int> child[N];

void DFS(int x, int h) {
    int i;
    l[x] = h;
    if(h > height) height = h;
    for(i = 0; i < child[x].size(); ++i)
        DFS(child[x][i], h + 1);
}

void DP() {
    int i, j;

    memset(P, -1, sizeof(P));

    for(i = 1; i <= nv; ++i)
        P[i][0] = p[i];
    for(j = 1; 1 << j <= nv; ++j) {
        for(i = 1; i <= nv; ++i) if(P[i][j-1] != -1) {
            P[i][j] = P[P[i][j-1]][j-1];
        }
    }
```

```
}

int LCA(int x, int y) {
    int i, tmp, logk;
    if(l[x] < l[y])
        tmp = x, x = y, y = tmp;

    for(logk = 0; 1 << logk <= l[x]; ++logk);
    logk--;

    if(l[x] > l[y]) {
        for(i = logk; i >= 0; --i)
            if(l[x] - (1 << i) >= l[y])
                x = P[x][i];
    }

    if(x == y)    return x;

    for(i = logk; i >= 0; --i) {
        if(P[x][i] != P[y][i] && P[x][i] != -1)
            x = P[x][i], y = P[y][i];
    }
    return p[x];
}

int main() {
//    freopen("t.in", "r", stdin);
    int ntc;
    int a, b, i, j;
    scanf("%d",&ntc);
    while(ntc--) {
        scanf("%d", &nv);
        memset(p, -1, sizeof(p));
        for(i = 0; i < nv-1; ++i) {
            scanf("%d %d", &a, &b);
            p[b] = a;
            child[a].push_back(b);
        }
        for(i = 1; p[i] != -1; ++i);
        DFS(i, 0);
        DP();
        scanf("%d %d", &a, &b);
        printf("%d\n", LCA(a, b));
```

```
        for(i = 1; i <= nv; ++i)
            child[i].clear();
    }
    return 0;
}
```

一、最近公共祖先(Least Common Ancestors)

对于有根树 T 的两个结点 u、v，最近公共祖先 LCA(T,u,v)表示一个结点 x，满足 x 是 u、v 的祖先且 x 的深度尽可能大。另一种理解方式是把 T 理解为一个无向无环图，而 LCA(T,u,v)即 u 到 v 的最短路上深度最小的点。

这里给出一个 LCA 的例子：

例一

对于 T=<V,E>
V={1,2,3,4,5}
E={(1,2),(1,3),(3,4),(3,5)}

则有：

LCA(T,5,2)=1
LCA(T,3,4)=3
LCA(T,4,5)=3

二、RMQ 问题(Range Minimum Query)

RMQ 问题是指：对于长度为 n 的数列 A，回答若干询问 RMQ(A,i,j)(i,j<=n)，返回数列 A 中下标在[i,j]里的最小值下标。这时一个 RMQ 问题的例子：

例二

对数列：5,8,1,3,6,4,9,5,7 有：

RMQ(2,4)=3
RMQ(6,9)=6

RMQ 问题与 LCA 问题的关系紧密，可以相互转换，相应的求解算法也有异曲同工之妙。

下面给出 LCA 问题向 RMQ 问题的转化方法。

对树进行深度优先遍历，每当“进入”或回溯到某个结点时，将这个结点的深度存入数组 E 最后一

位。同时记录结点 i 在数组中第一次出现的位置(事实上就是进入结点 i 时记录的位置)，记做 R[i]。如果结点 E[i]的深度记做 D[i]，易见，这时求 LCA(T,u,v)，就等价于求 E[RMQ(D,R[u],R[v])]，(R[u]<R[v])。例如，对于第一节的例一，求解步骤如下：

数列 E[i]为：1,2,1,3,4,3,5,3,1
R[i]为：1,2,4,5,7
D[i]为：0,1,0,1,2,1,2,1,0

于是有：

LCA(T,5,2) = E[RMQ(D,R[2],R[5])] = E[RMQ(D,2,7)] = E[3] = 1
LCA(T,3,4) = E[RMQ(D,R[3],R[4])] = E[RMQ(D,4,5)] = E[4] = 3
LCA(T,4,5) = E[RMQ(D,R[4],R[5])] = E[RMQ(D,5,7)] = E[6] = 3

易知，转化后得到的数列长度为树的结点数的两倍加一，所以转化后的 RMQ 问题与 LCA 问题的规模同次

8.7 割和桥

```
/*Cut & Brige
求割点和桥
判定规则 1: 如果 root 节点又多于一个 1 子节点 则 root 是割点
判定规则 2: 如果一个节点 u 有某一个子节点 v 不含到 u 的祖先节点的后向边 则 u 为割点
 即对于 u 的子节点 v, u 是割点的条件 (p[u] == 0 && b[v] > 1) || (p[u] > 0 && l[v] >= d[u])
桥: 不属于任何简单回路的边 "一牵动全身" l[v] > d[u]即是桥
之所以不能等于 实际上等于的情况是存在 2 条以上的边 自然就不是桥了~
 (注意加上割点表 以防重复输出)
*/

void DFS(int k, int father, int deep) { //father
    int i, tot = 0;
     chk[k] = 1; //visited
    D[k] = Ance[k] = deep; //init
    for(i = 0; i<n; i++) {
        if(i==father || i == k) continue;
        if(adj[i][k] == 1 && chk[i])    Ance[k] = Min(Ance[k], D[i]);
        if(adj[i][k] == 1 && !chk[i]) {
            DFS(i, k, deep + 1);
            tot++;
            Ance[k] = Min(Ance[k], Ance[i]);
            if(k == Root && tot > 1 || k!=Root && Ance[i] >= D[k])
                Cut[k] = true;
            if(Ance[i] > D[k]) Brige[k][i] = true;
        }
```

```
    }
}
```

8.8 最小生成树(kruskal)

```
#include<iostream>
#include<algorithm>
using namespace std;

int rank[101];
int v[101];
void make_set(int x)
{
    v[x]=x;
    rank[x]=0;
}
int find_set(int x)
{
    if(v[x]!=x) v[x]=find_set(v[x]);
    return v[x];
}

void Union(int x,int y)
{
    if(rank[x]>rank[y])
        v[y]=x;
    else if(rank[x]<rank[y])
        v[x]=y;
    else if(rank[x]==rank[y])
    {
        v[x]=y;
        rank[y]++;
    }
}

struct Edge
{
  int x,y,w;
}e[1001];
bool cmp(Edge e1,Edge e2)
{if(e1.w<e2.w) return true; else return false;}
int main()
{
  int n,m,s1,s2; int i,j,ans;
```

```
  cin>>n>>m;
  for(i=0;i<m;i++)
        cin>>e[i].x>>e[i].y>>e[i].w;

  sort(e,e+m,cmp);

  for(i=1;i<=n;i++) v[i]=i;
  ans=0;
  for(i=0;i<m;i++)
      {
          s1=find_set(e[i].x); s2=find_set(e[i].y);
         if(s1!=s2)
           {
              ans+=e[i].w;
              Union(s1,s2);
            }

       }
  cout<<ans<<endl;
return 0;
}
```

8.9 最短路径

```
#include<iostream>
using namespace std;
int n,c[100][100],s,t;

void dijkstra()
{
      int f[100],k,i,j,k1; bool p[100]={0}; int min;
  p[s]=1; min=999999;
  for(i=1;i<=n;i++)
      {f[i]=c[s][i]; }
  for(j=1;j<=n;j++)
    {
    min=999999; k1=0;
    for(i=1;i<=n;i++)
          if(!p[i]&&f[i]!=0&&f[i]<min) {min=f[i], k1=i;}
    p[k1]=1;
    if(k1==0) break;
    for(i=1;i<=n;i++)
      if(!p[i]&&c[k1][i]>0)
      if(f[k1]+c[k1][i]<f[i]||f[i]==0)
```

```
        {
        f[i]=f[k1]+c[k1][i];
        }
   }
   if(f[t]==0)cout<<"no way!"<<endl;
   else
   cout<<f[t]<<endl;
}
```

void bellford()

```
{
  int i,j; bool p=0; int f[100];
  for(i=1;i<=n;i++) f[i]=999999;
  f[s]=0;
  while(!p)
    {
     p=1;
     for(i=1;i<=n;i++)
          if(f[i]!=999999)
             for(j=1;j<=n;j++)
               if(c[i][j]>0&&f[i]+c[i][j]<f[j])
                 {
                 f[j]=f[i]+c[i][j];
                 p=0;
                 }
   }
  if(f[t]==999999) cout<<"no way!"<<endl;
  else cout<<f[t]<<endl;
}
```

void floyed()

```
{
   int f[100][100]; int i,j,k;
   for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
           if(c[i][j]>0||i==j) f[i][j]=c[i][j];
            else f[i][j]=999999;

   for(k=1;k<=n;k++)
     for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
          if(f[i][k]+f[k][j]<f[i][j])
              f[i][j]=f[i][k]+f[k][j];
   if(f[s][t]==999999) cout<<"no way!"<<endl;
```

```
  else cout<<f[s][t]<<endl;
}

int main()
{
  int i,j;
  cin>>n;
  for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
        cin>>c[i][j];
  while(cin>>s>>t)
      {
        dijkstra();
         bellford();
        floyed();
      }
  return 0;
}
```

8.10 最大网络流

```
//ford-folkson
#include<iostream>
#include<cmath>
#include<memory>
using namespace std;
struct Node
{
  int l,p;
}list[100];

struct Net
{
  int f,c;
}g[100][100];

int s,t,n,ans,del;

int find()
{
  int i=1;
  while(i<=n&&!(list[i].l!=0&&list[i].p==0)) i++;
  if(i>n) return 0; else return i;
}
```

```
bool ford()
{
  int i,j,m,x;
  memset(list,0,sizeof(list));
  list[s].l=s;
  while(list[t].l==0)
  {
    i=find();
    if(i==0) return true;
    for(j=1;j<=n;j++)
      if(list[j].l==0&&(g[i][j].c!=0||g[j][i].c!=0))
        {
          if(g[i][j].f<g[i][j].c) list[j].l=i;
         if(g[j][i].f>0) list[j].l=-i;
        }
  list[i].p=1;
  }

  m=t; del=999999;
  while(m!=s)
  {
    j=m; m=abs(list[j].l);
    if(list[j].l>0) x=g[m][j].c-g[m][j].f;
    if(list[j].l<0) x=g[j][m].f;
    if(x<del) del=x;
  }
  ans+=del;
  return false;
}

void change(int a)
{
  int j,m;
  m=t;
  while(m!=s)
  {
    j=m; m=abs(list[j].l);
    if(list[j].l<0) g[j][m].f-=a;
    if(list[j].l>0) g[m][j].f+=a;
  }
}

void work()
{
```

```
  bool p=0;
  s=1; t=n;
  while(1)
  {
    p=ford();
    if(p) {cout<<ans<<endl; break;}
    else change(del);
  }
}
int main()
{
    int i,j;
    cin>>n;
    memset(g,0,sizeof(g));
    for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
          cin>>g[i][j].c;
    ans=0;
  work();
  return 0;
}
```

//最短增广路

```
#include <stdio.h>
#include <string.h>

#define Min(a, b) ((a) < (b) ? (a) : (b))

const int MAXINT = 2000000000;
const int N = 110;
int nv, ne, np, nc;
int G[N][N];
int pre[N];
int cur[N];
int q[N];
int d[N];
int num[N];
int S, T;

void BFS()
{
      memset(num, 0, sizeof(num));
      int qs = 0, qe = 1, i;
      for(i = 1; i <= nv; ++i)
```

```
            d[i] = nv;
        num[nv] = nv-1;
        d[T] = 0;
        q[0] = T;
        num[0]++;
        while(qs < qe) {
            int cur = q[qs++];
            for(i = 1; i <= nv; ++i) if(d[i] == nv && G[i][cur] > 0) {
                q[qe++] = i;
                d[i] = d[cur]+1;
                num[nv]--;
                num[d[i]]++;
            }
        }
}

int augment() {
        int i, j, min = MAXINT;
        for(i = T, j = pre[i]; i != S; i = j, j = pre[i])
            min = Min(min, G[j][i]);
        for(i = T, j = pre[i]; i != S; i = j, j = pre[i]) {
            G[j][i] -= min; G[i][j] += min;
//          F[j][i] += min; F[i][j] -= min;
        }
        return min;
}

int trackback(int &i)
        {
        int tmp, j, mind = nv-1;
        for(j = 1; j <= nv; ++j)
            if(G[i][j] > 0 && d[j] < mind)
                mind = d[j];
        tmp = d[i];
        num[d[i]]--;
        d[i] = 1 + mind;
        if(d[i] == nv) cur[i] = nv+1;
        num[d[i]]++;
        if(i != S)
            i = pre[i];
        return num[tmp];
}

int maxflow() {
```

```
    int i, j;
    int flow = 0;
    BFS();
    for(i = 1; i <= nv; ++i) cur[i] = 1;

    i = S;
    while(d[S] < nv) {
        for(j = cur[i]; j <= nv; ++j)
            if(G[i][j] > 0 && d[i] == d[j]+1)
                break;
        if(j <= nv) {
            cur[i] = j+1;
            pre[j] = i;
            i = j;
            if(i == T) { flow += augment(); i = S; }
        } else {
            cur[i] = 1;
            if(trackback(i) == 0)
                break;
        }
    }
    return flow;
}

void init()
    {
    memset(G, 0, sizeof(G));
    S = nv+1, T = nv+2;
    nv += 2;
    int i, u, v, w;
    for(i = 0; i < ne; ++i)
    {
        while(getchar()!='(');
        scanf("%d,%d)%d", &u, &v, &w);
        G[u+1][v+1] = w;
    }
    for(i = 0; i < np; ++i)
    {
        while(getchar()!='(');
        scanf("%d)%d", &u, &w);
        G[S][u+1] = w;
    }
    for(i = 0; i < nc; ++i)
    {
```

```
            while(getchar()!='(');
            scanf("%d)%d", &u, &w);
            G[u+1][T] = w;
        }
}

int main()
{
//    freopen("t.in", "r", stdin);
      while(scanf("%d %d %d %d", &nv, &np, &nc, &ne) == 4)
      {
            init();
            printf("%d\n", maxflow());
      }
      return 0;
}
```

8.11 最小费用流

```
#include<iostream>
#include<cmath>
#include<memory>
using namespace std;

struct Node
{
  int cost,father;
}list[100];

struct Net
{
  int f,c,w;
}g[100][100];

int Min_cost,s,t,n;

bool find()
{
  bool p=1; int i,j;
  for(i=1;i<=n;i++)
      { list[i].cost=999999; list[i].father=0;}
  list[s].cost=0;    list[s].father=s;
  while(p)
  {
```

```
    p=0;
    for(i=1;i<=n;i++)
          if(list[i].cost!=999999)
            for(j=1;j<=n;j++)
              if(g[i][j].f<g[i][j].c&&list[i].cost+g[i][j].w<list[j].cost)
              {
               p=1;
              list[j].cost=list[i].cost+g[i][j].w;
                list[j].father=i;
              }
  }
  if(list[t].cost==999999) return false;
  else return true;
}

void change()
{
  int j,m,maxflow=999999;
  m=t;
  while(m!=s)
  {
    j=m; m=list[j].father;
    maxflow=g[m][j].c-g[m][j].f<maxflow?g[m][j].c-g[m][j].f:maxflow;
  }

  m=t;
  while(m!=s)
  {
    j=m; m=list[j].father;
    g[m][j].f+=maxflow;
    g[j][m].f=-g[m][j].f;
    Min_cost+=(maxflow*g[m][j].w);
  }
}

int main()
{
  int i,x,y,z1,z2;
  cin>>n;
  memset(g,0,sizeof(g));
  while(cin>>x>>y>>z1>>z2&&x+y+z1+z2>0)
      {
        g[x][y].c=z1;
        g[y][x].c=0;
```

```
        g[x][y].w=z2;
        g[y][x].w=-z2;
      }
  Min_cost=0;
  s=1; t=n;
  while(find()) change();
  cout<<Min_cost<<endl;
  return 0;
}
```

8.12 最大团问题

```
#include<stdio.h>
#include<string.h>
#define N 56
int map[N][N];
int dp[N];
int visit[N];
int Max=0;
int finded;
int n;
int find(int start ,int flag[])
{
	int i;
	for (i=start;i<n;++i)
		if (flag[i])
			return i;
	return -1;
}
void dfs(int start ,int visit[],int depth)
{
	int i;
	int set[N];
	int flag[N];
	memcpy(set,visit,4*n);
	memcpy(flag,visit,4*n);
	int first;
	first=find(start,flag);
	if (first==-1)
	{
		if (depth>Max)
		{
			Max=depth;
			finded=1;
```

```
            }
            return ;
        }
        while (first!=-1)
        {
            if (depth+n-start<=Max) return ;
            if (depth+dp[first]<=Max) return ;
            set[first]=0;
            flag[first]=0;
            for (i=first+1;i<n;++i)
            {
                if (flag[i]&&map[first][i])
                    set[i]=1;
                else set[i]=0;
            }
            dfs(first,set,depth+1);
            if (finded)
                return ;
            first=find(first,flag);
        }
}
int main()
{
        while (scanf("%d",&n),n)
        {
            int i,j;
            for (i=0;i<n;++i)
            {
                for (j=0;j<n;++j)
                    scanf("%d",&map[i][j]);
                dp[i]=0;
            }
            Max=0;
            for (i=n-1;i>=0;--i)
            {
                finded=0;
                memcpy(visit,map[i],4*n);
                dfs(i,visit,1);
                dp[i]=Max;
            }
            printf("%d\n",dp[0]);
        }
        return 0;
}
```

8.13 二分图匹配

```
#include<iostream>
using namespace std;
int p[100];
int f[100][100];
bool b[100];
int n,m;
bool path(int x)
{
  int i;
  for(i=1;i<=m;i++)
     if(!b[i]&&f[x][i])
     {
       b[i]=1;
      if(p[i]==0||path(p[i]))
           {p[i]=x; return true;}
     }
  return false;
}
int main()
{
  int ans,x,y,i;
  cin>>n>>m;
  memset(f,0,sizeof(f));
  while(cin>>x>>y&&x+y>0) f[x][y]=1;

  ans=0;
  for(i=1;i<=n;i++)
    {
      memset(b,0,sizeof(b));
      if(path(i)) ans++;
    }
    cout<<ans<<endl;
    return 0;
}
```

8.14 带权的最优二分图匹配

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <algorithm>
```

```
using namespace std;

const int N=110, MAX=1234567;

int n;
int map[N][N];
int lx[N], ly[N];
int match[N];
bool fx[N],fy[N];

int TheMin(int a,int b){return a<b?a:b;}
int TheMax(int a,int b){return a>b?a:b;}

bool DFS(int k)
{
    fx[k] = true;
    int i,t;
    for (i=0;i<n;++i)
        if ( !fy[i] && map[k][i] == lx[k] + ly[i] )
        {
            fy[i] = true;
            t = match[i];
            match[i] = k;
            if (t == -1 || DFS(t))
                return true;
            match[i] = t;
        }
    return false;
}

int KM_Match()
{
    int i,j,k,min;
    memset(ly, 0, sizeof(ly));
    for (i=0;i<n;++i)
    {
        lx[i] = map[i][0];
        for (j=1;j<n;++j)
            lx[i] = TheMax(lx[i], map[i][j]);
    }
    memset(match, -1, sizeof(match));
    k = 0;
    while (k<n)
    {
```

```
            memset(fx, false, sizeof(fx));
            memset(fy, false, sizeof(fy));
            if (DFS(k))
                ++k;
            else
            {
                min = MAX;
                for (i=0;i<n;++i)
                    if (fx[i])
                        for (j=0;j<n;++j)
                            if (!fy[j])
                                min = TheMin(min, lx[i] + ly[j] - map[i][j]);
                for (i=0;i<n;++i)
                {
                    if (fx[i])
                        lx[i] -= min;
                    if (fy[i])
                        ly[i] += min;
                }
            }
        }
        int ans=0;
        for (i=0;i<n;++i)
            ans += lx[i] + ly[i];
        return ans;
}

int xm[N], ym[N], xh[N], yh[N];
char ch[N][N];

int main()
{
        int a,b,i,j,min_max;
        while (scanf("%d%d", &a, &b), !(a == 0 && b == 0))
        {
            for (i=0;i<a;++i)
                scanf("%s", ch[i]);
            n = 0;
            for (i=0;i<a;++i)
                for (j=0;j<b;++j)
                    if (ch[i][j] == 'm')
                    {
                        xm[n] = i;
                        ym[n++] = j;
```

```
                }
        n = 0;
        for (i=0;i<a;++i)
            for (j=0;j<b;++j)
                if (ch[i][j] == 'H')
                {
                    xh[n] = i;
                    yh[n++] = j;
                }
        min_max = 0;
        for (i=0;i<n;++i)
            for (j=0;j<n;++j)
            {
                map[i][j] = abs( xm[i] - xh[j] ) + abs( ym[i] - yh[j] );
                min_max = TheMax(min_max, map[i][j]);
            }
        for (i=0;i<n;++i)
            for (j=0;j<n;++j)
                map[i][j] = min_max - map[i][j];
        printf("%d\n", n * min_max - KM_Match());
    }
    return 0;
}
```

9.搜索算法概略

9.1 迭代深搜+IDA*

```
//埃及分数
#include <iostream>
#include <algorithm>
#include <cstdlib>
#include <cmath>
#include <iomanip>
using namespace std;
const double zero=1e-20;
const double maxn=1e20;
int num;
bool bb;
double tempa[1000],tempb[1000],best[1000],v[1000];
int g_cd(int a,int b)
```

```
{
    int ta,tb;
    ta=a;tb=b;
    if(ta<tb)swap(ta,tb);
    if(!(ta%tb))return tb;
    else return g_cd(tb,ta%tb);
}
void solve(int x)
{
    int i,next,temp,prev;
    if(x>num)
    {
        if(fabs(tempa[num])<zero)
        {
                bb=1;
                if(v[num]<best[num])
                  for(i=1;i<=num;i++)
                      best[i]=v[i];
        }
    }else
    {
        prev=(int(tempb[x-1]/tempa[x-1])>int(v[x-1]+1))?int(tempb[x-1]/tempa[x-1]):int(v[x-1]+1);
        next=int((num-x+1)*tempb[x-1]/tempa[x-1]+zero);
        if(tempb[x-1]/tempa[x-1]+num-x+1>best[num])return;
        for(i=prev;i<=next;i++)
        {
            v[x]=i;
            tempa[x]=tempa[x-1]*i-tempb[x-1];
            if(tempa[x]<0)continue;
            tempb[x]=tempb[x-1]*i;
            if(tempa[x]>-zero && int(tempa[x]*v[x]/tempb[x])<=num-x)
               solve(x+1);
        }
    }
}
int main()
{
    int i,j,n,temp,a,b;
    cin>>n;
    for(j=0;j<n;j++)
    {
        cin>>a>>b;
        temp=g_cd(a,b);
        a/=temp;b/=temp;
```

```
        if(a==1)
                cout<<"1/"<<b<<endl;
        else
        {
                v[0]=b/a;num=1;
                bb=0;best[1]=maxn;
                while(1)
                {
                    num++;
                    best[num]=maxn;
                    tempa[0]=a;tempb[0]=b;
                    solve(1);
                    if(bb)break;
                }
                for(i=1;i<num;i++)
                    cout<<"1/"<<setiosflags(ios::fixed)<<setprecision(0)<<best[i]<<"+";
                cout<<"1/"<<best[num]<<endl;
        }
    }
    //system("PAUSE");
    return 0;
}
```

9.2 分之界限法（深搜）

```
//program jobs;
#include<iostream>
using namespace std;
#define maxn 101;
#define maxm 101;

int t[maxm];
int l[maxn],bestl[maxn];
int a[maxn],besta[maxn][maxm];
int time[maxn];
bool done[maxm];
int least,i,j,k,n,m,rest,min;
bool pp;

void print()
{
int i,j;
  for(i=1;i<=n;i++)
     {
```

```
          for(j=1;j<=bestl[i];j++)
        cout<<besta[i][j]<<" ";
        cout<<endl;
       }
cout<<"Time="<<time[0]+1<<endl;
}

void init()
{
  int i,j,k;
  cin>>n>>m;
  rest=0;
  for(i=1;i<=m;i++)
       {
          cin>>t[i]; resr+=t[i];
       }

least=rest/n+1; //{确定下界}

for(i=1;i<=m-1;i++)
  for(j=i+1;j<=m;j++)
    if (t[i]<t[j])
       {
        k=t[i];t[i]=t[j];t[j]=k
       }
}

void find(int p,int q) //{从 p..m 中选取作业放在处理机 q 上}
{
  int i;
  if (pp) return;
  for(i=p;i<=m;i++)
    if (done[i] && time[q]+t[i]<=time[q-1])
       {
        done[i]=false;
        l[q]++;
        a[q][l[q]]=t[i];
        time[q]+=t[i];
        rest-=t[i];
        find(i+1,q);
        done[i]=true;
        l[q]--;
        time[q]-=t[i];
        rest-=t[i];
```

```
      }
  if(rest<=(n-q)*time[q])
     if(rest==0)
      {
        memcpy(bestl,l);besta=a;

       time[0]=time[1]-1;
       if(time[1]==least)
            {
           print();
             pp=1;
              return;
            }
      }
      else
       if (q<n) then find(1,q+1);
}

int main()
{
  init();
  memset(done,1,sizeof(done));
  memset(time,0,sizeof(time));
  memset(a,0,sizeof(a));
  memset(besta,0,sizeof(besta));
  memset(l,0,sizeof(l));
  memset(bestl,0,sizeof(bestl));

  for(i=1;i<=m;i++) //{确定上界}
  {
     k=1;

     for(j=2;j<=n;j++)
       if (time[j]<time[k]) k=j;

     time[k]=time[k]+t[i];
     bestl[k]=bestl[k]+1;
     besta[k][bestl[k]]=t[i];
  }

    min=time[1];
    for(i=2;i<=n;i++)
      if (time[i]>min) min=time[i];
    time[0]=min-1;
```

```
  if(min==least){ print(); return 0; }
  memset(time,0,sizeof(time));
  time[0]=min-1;
  pp=0;
  find(1,1);
  if(!pp) print();
  return 0;
  }
```

9.3 A* 8 数码问题(pascal)

```
type
       a33=array[1..3,1..3] of 0..8;
       a4=array[1..4] of -1..1;
    node=record
      ch:a33;
      si,sj:1..3;
      f:byte;
      pnt,dep,next:byte;
      end;
 const goal:a33=((1,2,3),(8,0,4),(7,6,5));
           start:a33=((2,8,3),(1,6,4),(7,0,5));
           di:a4=(0,-1,0,1);
           dj:a4=(-1,0,1,0);
 var data:array[0..100] of node;
  temp:node;
  r,k,ni,nj,head,tail,depth:integer;
 function check(k:integer):boolean;
  begin
    ni:=temp.si+di[k];nj:=temp.sj+dj[k];
    if (ni in [1..3]) and (nj in [1..3]) then check:=true else check:=false;
  end;
  function dupe:boolean;
  var i,j,k:integer;
       buf:boolean;
  begin
    buf:=false;i:=0;
    repeat
      inc(i);buf:=true;
      for j:=1 to 3 do
        for k:=1 to 3 do
          if data[i].ch[j,k]<>data[tail].ch[j,k] then buf:=false;
    until buf or (i>=tail-1);
    dupe:=buf;
```

```
    end;
    function goals:boolean;
      var i,j:byte;
      begin
        goals:=true;
        for i:=1 to 3 do
          for j:=1 to 3 do
            if data[tail].ch[i,j]<>goal[i,j] then goals:=false;
      end;
      procedure print;
        var buf:array[1..100] of integer;
        i,j,k,n:integer;
        begin
          n:=1;
          i:=tail;buf[1]:=i;
          repeat
            inc(n);buf[n]:=data[i].pnt;
            i:=data[i].pnt;
          until i=0;
          writeln('steps=',depth+1);
          for i:=1 to 3 do
            begin
              for k:=n-1 downto 1 do
              begin
                for j:=1 to 3 do write(data[buf[k]].ch[i,j]);
                  if (i=2) and (k<>1) then write('->') else write('    ');
              end;
              writeln;
            end;
          readln;halt
      end;
function calc_f(a:a33):byte;
var i,j,temp:byte;
begin
  temp:=0;
  for i:=1 to 3 do
    for j:=1 to 3 do
      if (a[i,j]<>goal[i,j]) and (a[i,j]>0) then inc(temp);
calc_f:=temp+depth+1;
end;
procedure sort(num:integer);
var x,y:word;
begin
  y:=head;
```

```
  repeat
    x:=y;y:=data[x].next;
  until (y=0) or (data[y].f>data[num].f);
data[x].next:=num;data[num].next:=y;
end;
begin
  head:=0;tail:=1; data[0].next:=1;
  with data[1] do
    begin
      ch:=start;si:=3;sj:=2;
      pnt:=0;dep:=0;next:=0;
      f:=calc_f(ch);
    end;
  repeat
  head:=data[head].next;temp:=data[head];
  depth:=temp.dep;
  for r:=1 to 4 do
    if check(r) then
      begin
      inc(tail);data[tail]:=temp;
      with data[tail] do
        begin
          ch[si,sj]:=ch[ni,nj];
          ch[ni,nj]:=0;si:=ni;sj:=nj;
          pnt:=head;
          dep:=depth+1;
          f:=calc_f(ch);
        end;
      if dupe then dec(tail) else if goals then print else sort(tail);
      end;
  until data[head].next=0;
  writeln('no solution');readln;
end.
```

9.4 优先队列广搜

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <queue>
using namespace std;

const int N = 30;
```

```
const int INF = 1000000000;
const int LCM = 2520; //避免精度误差  漂亮!

int np,n,m,sx,ex;
int piao[8];
int dp[N][1<<8];
int adj[N][N];

struct Node
{
    int x, y;
    int s;
    Node() {}
    Node(int xx, int yy, int ss)
    {
        x = xx; //当前所在点
        y = yy; //当前取票状态
        s = ss; //当前值
    }
};

bool operator<(const Node&a ,const Node& b)
{
    return a.s > b.s;
}

void solve()
{
    int i, j, k, x, y;
    int s, ss;
    Node now;
    for(i=0;i<n;++i)
    {
        for(j=0;j < (1<<np); ++j)
        {
            dp[i][j] = INF;
        }
    }

    priority_queue<Node> Q;

    Q.push(Node(sx, 0, 0));
    dp[sx][0] = 0;
    while(Q.size())
```

```
    {
        x = Q.top().x,    y = Q.top().y;
        s = Q.top().s;
        Q.pop();
        if(s > dp[x][y]) continue;
        if(x == ex)
        {
            printf("%lf\n", (double)s/LCM);
            return;
        }
        for(i=0;i<n;++i)
            if(adj[x][i])
            {
            for(j=0;j<np;++j)
                if(!(y&(1<<j)))
                {
                ss = s + adj[x][i] * (LCM/ piao[j]);
                if(ss < dp[i][y|(1<<j)])
                    {
                    dp[i][y|(1<<j)] = ss;
                    /*now.x = i;
                    now.y = y|(1<<j);
                    now.s = ss; */
                    Q.push(Node(i,y|(1<<j),ss));
                    }
                }
            }
    }
    printf("Impossible\n");
}

int main()
{
    int i, j, x, y, z;
    while(scanf("%d%d%d%d%d", &np, &n, &m, &sx, &ex), np+n+m+sx+ex)
    {
        --sx, --ex;
        memset(adj,0,sizeof(adj));
        for(i=0;i<np;++i) scanf("%d", &piao[i]);
        for(i=0;i<m;++i)
        {
            scanf("%d%d%d",&x,&y,&z);
            adj[x-1][y-1] = z;
            adj[y-1][x-1] = z;
```

```
            }
            solve();
        }
        return 0;
}
```

10.应用

10.1 Joseph 问题

```
// Joseph's Problem
// input: n,m            -- the number of persons, the inteval between persons
// output:         -- return the reference of last person

int josephus0(int n, int m)
{
        if (n == 2) return (m%2) ? 2 : 1;
        int v = (m+josephus0(n-1,m)) % n;
        if (v == 0) v = n;
        return v;
}
int josephus(int n, int m)
{
        if (m == 1) return n;
        if (n == 1) return 1;
        if (m >=n) return josephus0(n,m);
        int l = (n/m)*m;
        int j = josephus(n - (n/m), m);
        if (j <= n-l) return l+j;
        j -= n-l;
        int t = (j/(m-1))*m;
        if ((j % (m-1)) == 0) return t-1;
        return t + (j % (m-1));
}
```

10.2 N 皇后构造解

```
//N 皇后构造解,n>=4

void even1(int n,int *p){
        int i;
```

```
    for (i=1;i<=n/2;i++)
        p[i-1]=2*i;
    for (i=n/2+1;i<=n;i++)
        p[i-1]=2*i-n-1;
}

void even2(int n,int *p){
    int i;
    for (i=1;i<=n/2;i++)
        p[i-1]=(2*i+n/2-3)%n+1;
    for (i=n/2+1;i<=n;i++)
        p[i-1]=n-(2*(n-i+1)+n/2-3)%n;
}

void generate(int,int*);
void odd(int n,int *p){
    generate(n-1,p),p[n-1]=n;
}

void generate(int n,int *p){
    if (n&1)
        odd(n,p);
    else if (n%6!=2)
        even1(n,p);
    else
        even2(n,p);
}
```

10.3 布尔母函数

```
//布尔母函数
//判 m[]个价值为 w[]的货币能否构成 value
//适合 m[]较大 w[]较小的情况
//返回布尔量
//传入货币种数 n,个数 m[],价值 w[]和目标值 value
#define MAXV 100000

int genfunc(int n,int* m,int* w,int value){
    int i,j,k,c;
    char r[MAXV];
    for (r[0]=i=1;i<=value;r[i++]=0);
    for (i=0;i<n;i++){
        for (j=0;j<w[i];j++){
            c=m[i]*r[k=j];
```

```
                while ((k+=w[i])<=value)
                    if (r[k])
                        c=m[i];
                    else if (c)
                        r[k]=1,c--;
                if (r[value])
                    return 1;
            }
        }
        return 0;
}
```

10.4 第 k 元素

```
//取第 k 个元素,k=0..n-1
//平均复杂度 O(n)
//注意 a[]中的顺序被改变

#define _cp(a,b) ((a)<(b))
typedef int elem_t;
elem_t kth_element(int n,elem_t* a,int k){
        elem_t t,key;
        int l=0,r=n-1,i,j;
        while (l<r){
                for (key=a[((i=l-1)+(j=r+1))>>1];i<j;){
                        for (j--;_cp(key,a[j]);j--);
                        for (i++;_cp(a[i],key);i++);
                        if (i<j) t=a[i],a[i]=a[j],a[j]=t;
                }
                if (k>j) l=j+1;
                else r=j;
        }
        return a[k];
}
```

10.5 幻方构造

```
//幻方构造(l!=2)
#define MAXN 100

void dllb(int l,int si,int sj,int sn,int d[][MAXN]){
        int n,i=0,j=l/2;
        for (n=1;n<=l*l;n++){
```

```
            d[i+si][j+sj]=n+sn;
            if (n%l){
                i=(i)?(i-1):(l-1);
                j=(j==l-1)?0:(j+1);
            }
            else
                i=(i==l-1)?0:(i+1);
        }
}

void magic_odd(int l,int d[][MAXN]){
        dllb(l,0,0,0,d);
}

void magic_4k(int l,int d[][MAXN]){
        int i,j;
        for (i=0;i<l;i++)
            for (j=0;j<l;j++)

        d[i][j]=((i%4==0||i%4==3)&&(j%4==0||j%4==3)||(i%4==1||i%4==2)&&(j%4==1||j%4==2))?(l*l-(i*l+j)):(i*l+j+1);
}

void magic_other(int l,int d[][MAXN]){
        int i,j,t;
        dllb(l/2,0,0,0,d);
        dllb(l/2,l/2,l/2,l*l/4,d);
        dllb(l/2,0,l/2,l*l/2,d);
        dllb(l/2,l/2,0,l*l/4*3,d);
        for (i=0;i<l/2;i++)
            for (j=0;j<l/4;j++)
                if (i!=l/4||j)
                    t=d[i][j],d[i][j]=d[i+l/2][j],d[i+l/2][j]=t;
        t=d[l/4][l/4],d[l/4][l/4]=d[l/4+l/2][l/4],d[l/4+l/2][l/4]=t;
        for (i=0;i<l/2;i++)
            for (j=l-l/4+1;j<l;j++)
                t=d[i][j],d[i][j]=d[i+l/2][j],d[i+l/2][j]=t;
}

void generate(int l,int d[][MAXN]){
        if (l%2)
            magic_odd(l,d);
        else if (l%4==0)
            magic_4k(l,d);
        else
```

```
        magic_other(l,d);
}
```

10.6 模式匹配(kmp)

```
//模式匹配,kmp 算法,复杂度 O(m+n)
//返回匹配位置,-1 表示匹配失败,传入匹配串和模式串和长度
//可更改元素类型,更换匹配函数
#define MAXN 10000
#define _match(a,b) ((a)==(b))
typedef char elem_t;

int pat_match(int ls,elem_t* str,int lp,elem_t* pat){
    int fail[MAXN]={-1},i=0,j;
    for (j=1;j<lp;j++){
        for (i=fail[j-1];i>=0&&!_match(pat[i+1],pat[j]);i=fail[i]);
        fail[j]=(_match(pat[i+1],pat[j])?i+1:-1);
    }
    for (i=j=0;i<ls&&j<lp;i++)
        if (_match(str[i],pat[j]))
            j++;
        else if (j)
            j=fail[j-1]+1,i--;
    return j==lp?(i-lp):-1;
}
```

10.7 逆序对数

```
//序列逆序对数,复杂度 O(nlogn)
//传入序列长度和内容,返回逆序对数
//可更改元素类型和比较函数
#include <string.h>
#define MAXN 1000000
#define _cp(a,b) ((a)<=(b))
typedef int elem_t;
elem_t _tmp[MAXN];

int inv(int n,elem_t* a){
    int l=n>>1,r=n-l,i,j;
    int ret=(r>1?(inv(l,a)+inv(r,a+l)):0);
    for (i=j=0;i<=l;_tmp[i+j]=a[i],i++)
        for (ret+=j;j<r&&(i==l||!_cp(a[i],a[l+j]));_tmp[i+j]=a[l+j],j++);
    memcpy(a,_tmp,sizeof(elem_t)*n);
```

```
    return ret;
}
```

10.8 字符串最小表示

```
/*
    求字符串的最小表示
    输入：字符串
    返回：字符串最小表示的首字母位置(0...size-1)
*/
template <class T>
int MinString(vector <T> &str)
{
    int i, j, k;
    vector <T> ss(str.size() << 1);
    for (i = 0; i < str.size(); i ++) ss[i] = ss[i + str.size()] = str[i];
    for (i = k = 0, j = 1; k < str.size() && i < str.size() && j < str.size(); ) {
        for (k = 0; k < str.size() && ss[i + k] == ss[j + k]; k ++);
        if (k < str.size()) {
            if (ss[i + k] > ss[j + k])
                i += k + 1;
            else j += k + 1;
            if (i == j) j ++;
        }
    }
    return i < j ? i : j;
}
```

10.9 最长公共单调子序列

```
// 最长公共递增子序列， 时间复杂度 O(n^2 * logn)，空间 O(n^2)

/**
 * n 为 a 的大小, m 为 b 的大小
 *  结果在 ans 中
 * "define _cp(a,b) ((a)<(b))"求解最长严格递增序列
 */
#define MAXN 1000
#define _cp(a,b) ((a)<(b))
typedef int elem_t;

elem_t DP[MAXN][MAXN];
int num[MAXN], p[1<<20];
```

```
int LIS(int n, elem_t *a, int m, elem_t *b, elem_t *ans){
    int i, j, l, r, k;

    DP[0][0] = 0;
    num[0] = (b[0] == a[0]);
    for(i = 1; i < m; i++) {
        num[i] = (b[i] == a[0]) || num[i-1];
        DP[i][0] = 0;
    }
    for(i = 1; i < n; i++){
        if(b[0] == a[i] && !num[0]) {
            num[0] = 1;
            DP[0][0] = i<<10;
        }

        for(j = 1; j < m; j++){
            for(k=((l=0)+(r=num[j-1]-1))>>1; l<=r; k=(l+r)>>1)
                if(_cp(a[DP[j-1][k]>>10], a[i]))
                    l=k+1;
                else
                    r=k-1;

            if(l < num[j-1] && i == (DP[j-1][l]>>10) ){
                if(l >= num[j]) DP[j][num[j]++] = DP[j-1][l];
                else DP[j][l] = _cp(a[DP[j][l]>>10],a[i]) ? DP[j][l] : DP[j-1][l];
            }
            if(b[j] == a[i]){
                for(k=((l=0)+(r=num[j]-1))>>1; l<=r; k=(l+r)>>1)
                    if(_cp(a[DP[j][k]>>10], a[i]))
                        l=k+1;
                    else
                        r=k-1;

                DP[j][l] = (i<<10) + j;
                num[j] += (l>=num[j]);
                p[DP[j][l]] = l ? DP[j][l-1] : -1;
            }
        }
    }

    for (k=DP[m-1][i=num[m-1]-1];i>=0;ans[i--]=a[k>>10],k=p[k]);
    return num[m-1];
}
```

10.10 最长子序列

```
//最长单调子序列,复杂度 O(nlogn)
//注意最小序列覆盖和最长序列的对应关系,例如
//"define _cp(a,b) ((a)>(b))"求解最长严格递减序列,则
//"define _cp(a,b) (!((a)>(b)))"求解最小严格递减序列覆盖
//可更改元素类型和比较函数
#define MAXN 10000
#define _cp(a,b) ((a)>(b))
typedef int elem_t;

int subseq(int n,elem_t* a){
	int b[MAXN],i,l,r,m,ret=0;
	for (i=0;i<n;b[l]=i++,ret+=(l>ret))
		for (m=((l=1)+(r=ret))>>1;l<=r;m=(l+r)>>1)
			if (_cp(a[b[m]],a[i]))
				l=m+1;
			else
				r=m-1;
	return ret;
}

int subseq(int n,elem_t* a,elem_t* ans){
	int b[MAXN],p[MAXN],i,l,r,m,ret=0;
	for (i=0;i<n;p[b[l]=i++]=b[l-1],ret+=(l>ret))
		for (m=((l=1)+(r=ret))>>1;l<=r;m=(l+r)>>1)
			if (_cp(a[b[m]],a[i]))
				l=m+1;
			else
				r=m-1;
	for (m=b[i=ret];i;ans[--i]=a[m],m=p[m]);
	return ret;
}
```

10.11 最大子串匹配

```
//最大子串匹配,复杂度 O(mn)
//返回最大匹配值,传入两个串和串的长度,重载返回一个最大匹配
//注意做字符串匹配是串末的'\0'没有置!
//可更改元素类型,更换匹配函数和匹配价值函数
#include <string.h>
#define MAXN 100
#define max(a,b) ((a)>(b)?(a):(b))
```

```
#define _match(a,b) ((a)==(b))
#define _value(a,b) 1
typedef char elem_t;

int str_match(int m,elem_t* a,int n,elem_t* b){
	int match[MAXN+1][MAXN+1],i,j;
	memset(match,0,sizeof(match));
	for (i=0;i<m;i++)
		for (j=0;j<n;j++)
			match[i+1][j+1]=max(max(match[i][j+1],match[i+1][j]),
					(match[i][j]+_value(a[i],b[i]))*_match(a[i],b[j]));
	return match[m][n];
}

int str_match(int m,elem_t* a,int n,elem_t* b,elem_t* ret){
	int match[MAXN+1][MAXN+1],last[MAXN+1][MAXN+1],i,j,t;
	memset(match,0,sizeof(match));
	for (i=0;i<m;i++)
		for (j=0;j<n;j++){
			match[i+1][j+1]=(match[i][j+1]>match[i+1][j]?match[i][j+1]:match[i+1][j]);
			last[i+1][j+1]=(match[i][j+1]>match[i+1][j]?3:1);
			if ((t=(match[i][j]+_value(a[i],b[i]))*_match(a[i],b[j]))>match[i+1][j+1])
				match[i+1][j+1]=t,last[i+1][j+1]=2;
		}
	for (;match[i][j];i-=(last[t=i][j]>1),j-=(last[t][j]<3))
		ret[match[i][j]-1]=(last[i][j]<3?a[i-1]:b[j-1]);
	return match[m][n];
}
```

10.12 最大子段和

```
//求最大子段和,复杂度 O(n)
//传入串长 n 和内容 list[]
//返回最大子段和,重载返回子段位置(maxsum=list[start]+...+list[end])
//可更改元素类型
typedef int elem_t;

elem_t maxsum(int n,elem_t* list){
	elem_t ret,sum=0;
	int i;
	for (ret=list[i=0];i<n;i++)
		sum=(sum>0?sum:0)+list[i],ret=(sum>ret?sum:ret);
	return ret;
}
```

```
elem_t maxsum(int n,elem_t* list,int& start,int& end){
	elem_t ret,sum=0;
	int s,i;
	for (ret=list[start=end=s=i=0];i<n;i++,s=(sum>0?s:i))
		if ((sum=(sum>0?sum:0)+list[i])>ret)
			ret=sum,start=s,end=i;
	return ret;
}
```

10.13 最大子阵和

```
//求最大子阵和,复杂度 O(n^3)
//传入阵的大小 m,n 和内容 mat[][]
//返回最大子阵和,重载返回子阵位置(maxsum=list[s1][s2]+...+list[e1][e2])
//可更改元素类型
#define MAXN 100
typedef int elem_t;

elem_t maxsum(int m,int n,elem_t mat[][MAXN]){
	elem_t matsum[MAXN][MAXN+1],ret,sum;
	int i,j,k;
	for (i=0;i<m;i++)
		for (matsum[i][j=0]=0;j<n;j++)
			matsum[i][j+1]=matsum[i][j]+mat[i][j];
	for (ret=mat[0][j=0];j<n;j++)
		for (k=j;k<n;k++)
			for (sum=0,i=0;i<m;i++)
				sum=(sum>0?sum:0)+matsum[i][k+1]-matsum[i][j],ret=(sum>ret?sum:ret);
	return ret;
}

elem_t maxsum(int m,int n,elem_t mat[][MAXN],int& s1,int& s2,int& e1,int& e2){
	elem_t matsum[MAXN][MAXN+1],ret,sum;
	int i,j,k,s;
	for (i=0;i<m;i++)
		for (matsum[i][j=0]=0;j<n;j++)
			matsum[i][j+1]=matsum[i][j]+mat[i][j];
	for (ret=mat[s1=e1=0][s2=e2=j=0];j<n;j++)
		for (k=j;k<n;k++)
			for (sum=0,s=i=0;i<m;i++,s=(sum>0?s:i))
				if ((sum=(sum>0?sum:0)+matsum[i][k+1]-matsum[i][j])>ret)
					ret=sum,s1=s,s2=i,e1=j,e2=k;
	return ret;
```

```
}
```

11.其它

11.1 大数(只能处理正数)

```
#include <iostream.h>
#include <string.h>

#define DIGIT4
#define DEPTH      10000
#define MAX        100
typedef int bignum_t[MAX+1];

int read(bignum_t a,istream& is=cin){
    char buf[MAX*DIGIT+1],ch;
    int i,j;
    memset((void*)a,0,sizeof(bignum_t));
    if (!(is>>buf)) return 0;
    for (a[0]=strlen(buf),i=a[0]/2-1;i>=0;i--)
        ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;
    for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf);j<a[0]*DIGIT;buf[j++]='0');
    for (i=1;i<=a[0];i++)
        for (a[i]=0,j=0;j<DIGIT;j++)
            a[i]=a[i]*10+buf[i*DIGIT-1-j]-'0';
    for (;!a[a[0]]&&a[0]>1;a[0]--);
    return 1;
}

void write(const bignum_t a,ostream& os=cout){
    int i,j;
    for (os<<a[i=a[0]],i--;i;i--)
        for (j=DEPTH/10;j;j/=10)
            os<<a[i]/j%10;
}

int comp(const bignum_t a,const bignum_t b){
    int i;
    if (a[0]!=b[0])
        return a[0]-b[0];
    for (i=a[0];i;i--)
```

```
        if (a[i]!=b[i])
            return a[i]-b[i];
    return 0;
}

int comp(const bignum_t a,const int b){
    int c[12]={1};
    for (c[1]=b;c[c[0]]>=DEPTH;c[c[0]+1]=c[c[0]]/DEPTH,c[c[0]]%=DEPTH,c[0]++);
    return comp(a,c);
}

int comp(const bignum_t a,const int c,const int d,const bignum_t b){
    int i,t=0,O=-DEPTH*2;
    if (b[0]-a[0]<d&&c)
        return 1;
    for (i=b[0];i>d;i--){
        t=t*DEPTH+a[i-d]*c-b[i];
        if (t>0) return 1;
        if (t<O) return 0;
    }
    for (i=d;i;i--){
        t=t*DEPTH-b[i];
        if (t>0) return 1;
        if (t<O) return 0;
    }
    return t>0;
}

void add(bignum_t a,const bignum_t b){
    int i;
    for (i=1;i<=b[0];i++)
        if ((a[i]+=b[i])>=DEPTH)
            a[i]-=DEPTH,a[i+1]++;
    if (b[0]>=a[0])
        a[0]=b[0];
    else
        for (;a[i]>=DEPTH&&i<a[0];a[i]-=DEPTH,i++,a[i]++);
    a[0]+=(a[a[0]+1]>0);
}

void add(bignum_t a,const int b){
    int i=1;
    for (a[1]+=b;a[i]>=DEPTH&&i<a[0];a[i+1]+=a[i]/DEPTH,a[i]%=DEPTH,i++);
    for (;a[a[0]]>=DEPTH;a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);
```

```
}

void sub(bignum_t a,const bignum_t b){
	int i;
	for (i=1;i<=b[0];i++)
		if ((a[i]-=b[i])<0)
			a[i+1]--,a[i]+=DEPTH;
	for (;a[i]<0;a[i]+=DEPTH,i++,a[i]--);
	for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void sub(bignum_t a,const int b){
	int i=1;
	for (a[1]-=b;a[i]<0;a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH,i++);
	for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void sub(bignum_t a,const bignum_t b,const int c,const int d){
	int i,O=b[0]+d;
	for (i=1+d;i<=O;i++)
		if ((a[i]-=b[i-d]*c)<0)
			a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH;
	for (;a[i]<0;a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH,i++);
	for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void mul(bignum_t c,const bignum_t a,const bignum_t b){
	int i,j;
	memset((void*)c,0,sizeof(bignum_t));
	for (c[0]=a[0]+b[0]-1,i=1;i<=a[0];i++)
		for (j=1;j<=b[0];j++)
			if ((c[i+j-1]+=a[i]*b[j])>=DEPTH)
				c[i+j]+=c[i+j-1]/DEPTH,c[i+j-1]%=DEPTH;
	for (c[0]+=(c[c[0]+1]>0);!c[c[0]]&&c[0]>1;c[0]--);
}

void mul(bignum_t a,const int b){
	int i;
	for (a[1]*=b,i=2;i<=a[0];i++){
		a[i]*=b;
		if (a[i-1]>=DEPTH)
			a[i]+=a[i-1]/DEPTH,a[i-1]%=DEPTH;
	}
	for (;a[a[0]]>=DEPTH;a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);
```

```
    for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void mul(bignum_t b,const bignum_t a,const int c,const int d){
    int i;
    memset((void*)b,0,sizeof(bignum_t));
    for (b[0]=a[0]+d,i=d+1;i<=b[0];i++)
        if ((b[i]+=a[i-d]*c)>=DEPTH)
            b[i+1]+=b[i]/DEPTH,b[i]%=DEPTH;
    for (;b[b[0]+1];b[0]++,b[b[0]+1]=b[b[0]]/DEPTH,b[b[0]]%=DEPTH);
    for (;!b[b[0]]&&b[0]>1;b[0]--);
}

void div(bignum_t c,bignum_t a,const bignum_t b){
    int h,l,m,i;
    memset((void*)c,0,sizeof(bignum_t));
    c[0]=(b[0]<a[0]+1)?(a[0]-b[0]+2):1;
    for (i=c[0];i;sub(a,b,c[i]=m,i-1),i--)
        for (h=DEPTH-1,l=0,m=(h+l+1)>>1;h>l;m=(h+l+1)>>1)
            if (comp(b,m,i-1,a)) h=m-1;
            else l=m;
    for (;!c[c[0]]&&c[0]>1;c[0]--);
    c[0]=c[0]>1?c[0]:1;
}

void div(bignum_t a,const int b,int& c){
    int i;
    for (c=0,i=a[0];i;c=c*DEPTH+a[i],a[i]=c/b,c%=b,i--);
    for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void sqrt(bignum_t b,bignum_t a){
    int h,l,m,i;
    memset((void*)b,0,sizeof(bignum_t));
    for (i=b[0]=(a[0]+1)>>1;i;sub(a,b,m,i-1),b[i]+=m,i--)
        for (h=DEPTH-1,l=0,b[i]=m=(h+l+1)>>1;h>l;b[i]=m=(h+l+1)>>1)
            if (comp(b,m,i-1,a)) h=m-1;
            else l=m;
    for (;!b[b[0]]&&b[0]>1;b[0]--);
    for (i=1;i<=b[0];b[i++]>>=1);
}

int length(const bignum_t a){
    int t,ret;
```

```
    for (ret=(a[0]-1)*DIGIT,t=a[a[0]];t;t/=10,ret++);
    return ret>0?ret:1;
}

int digit(const bignum_t a,const int b){
    int i,ret;
    for (ret=a[(b-1)/DIGIT+1],i=(b-1)%DIGIT;i;ret/=10,i--);
    return ret%10;
}

int zeronum(const bignum_t a){
    int ret,t;
    for (ret=0;!a[ret+1];ret++);
    for (t=a[ret+1],ret*=DIGIT;!(t%10);t/=10,ret++);
    return ret;
}

void comp(int* a,const int l,const int h,const int d){
    int i,j,t;
    for (i=l;i<=h;i++)
        for (t=i,j=2;t>1;j++)
            while (!(t%j))
                a[j]+=d,t/=j;
}

void convert(int* a,const int h,bignum_t b){
    int i,j,t=1;
    memset(b,0,sizeof(bignum_t));
    for (b[0]=b[1]=1,i=2;i<=h;i++)
        if (a[i])
            for (j=a[i];j;t*=i,j--)
                if (t*i>DEPTH)
                    mul(b,t),t=1;
    mul(b,t);
}

void combination(bignum_t a,int m,int n){
    int* t=new int[m+1];
    memset((void*)t,0,sizeof(int)*(m+1));
    comp(t,n+1,m,1);
    comp(t,2,m-n,-1);
    convert(t,m,a);
    delete []t;
}
```

```
void permutation(bignum_t a,int m,int n){
	int i,t=1;
	memset(a,0,sizeof(bignum_t));
	a[0]=a[1]=1;
	for (i=m-n+1;i<=m;t*=i++)
		if (t*i>DEPTH)
			mul(a,t),t=1;
	mul(a,t);
}

#define SGN(x) ((x)>0?1:((x)<0?-1:0))
#define ABS(x) ((x)>0?(x):-(x))

int read(bignum_t a,int &sgn,istream& is=cin){
	char str[MAX*DIGIT+2],ch,*buf;
	int i,j;
	memset((void*)a,0,sizeof(bignum_t));
	if (!(is>>str)) return 0;
	buf=str,sgn=1;
	if (*buf=='-') sgn=-1,buf++;
	for (a[0]=strlen(buf),i=a[0]/2-1;i>=0;i--)
		ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;
	for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf);j<a[0]*DIGIT;buf[j++]='0');
	for (i=1;i<=a[0];i++)
		for (a[i]=0,j=0;j<DIGIT;j++)
			a[i]=a[i]*10+buf[i*DIGIT-1-j]-'0';
	for (;!a[a[0]]&&a[0]>1;a[0]--);
	if (a[0]==1&&!a[1]) sgn=0;
	return 1;
}
```

11.2 分数

```
struct frac{
	int num,den;
};

double fabs(double x){
	return x>0?x:-x;
}

int gcd(int a,int b){
	int t;
```

```
    if (a<0)
        a=-a;
    if (b<0)
        b=-b;
    if (!b)
        return a;
    while (t=a%b)
        a=b,b=t;
    return b;
}

void simplify(frac& f){
    int t;
    if (t=gcd(f.num,f.den))
        f.num/=t,f.den/=t;
    else
        f.den=1;
}

frac f(int n,int d,int s=1){
    frac ret;
    if (d<0)
        ret.num=-n,ret.den=-d;
    else
        ret.num=n,ret.den=d;
    if (s)
        simplify(ret);
    return ret;
}

frac convert(double x){
    frac ret;
    for (ret.den=1;fabs(x-int(x))>1e-10;ret.den*=10,x*=10);
    ret.num=(int)x;
    simplify(ret);
    return ret;
}

int fraqcmp(frac a,frac b){
    int g1=gcd(a.den,b.den),g2=gcd(a.num,b.num);
    if (!g1||!g2)
        return 0;
    return b.den/g1*(a.num/g2)-a.den/g1*(b.num/g2);
}
```

```
frac add(frac a,frac b){
	int g1=gcd(a.den,b.den),g2,t;
	if (!g1)
		return f(1,0,0);
	t=b.den/g1*a.num+a.den/g1*b.num;
	g2=gcd(g1,t);
	return f(t/g2,a.den/g1*(b.den/g2),0);
}

frac sub(frac a,frac b){
	return add(a,f(-b.num,b.den,0));
}

frac mul(frac a,frac b){
	int t1=gcd(a.den,b.num),t2=gcd(a.num,b.den);
	if (!t1||!t2)
		return f(1,1,0);
	return f(a.num/t2*(b.num/t1),a.den/t1*(b.den/t2),0);
}

frac div(frac a,frac b){
	return mul(a,f(b.den,b.num,0));
}
```

11.3 矩阵

```
define MAXN 100

#define fabs(x) ((x)>0?(x):-(x))
#define zero(x) (fabs(x)<1e-10)

struct mat{
	int n,m;
	double data[MAXN][MAXN];
};

int mul(mat& c,const mat& a,const mat& b){
	int i,j,k;
	if (a.m!=b.n)
		return 0;
	c.n=a.n,c.m=b.m;
	for (i=0;i<c.n;i++)
		for (j=0;j<c.m;j++)
```

```
                for (c.data[i][j]=k=0;k<a.m;k++)
                    c.data[i][j]+=a.data[i][k]*b.data[k][j];
    return 1;
}

int inv(mat& a){
    int i,j,k,is[MAXN],js[MAXN];
    double t;
    if (a.n!=a.m)
        return 0;
    for (k=0;k<a.n;k++){
        for (t=0,i=k;i<a.n;i++)
            for (j=k;j<a.n;j++)
                if (fabs(a.data[i][j])>t)
                    t=fabs(a.data[is[k]=i][js[k]=j]);
        if (zero(t))
            return 0;
        if (is[k]!=k)
            for (j=0;j<a.n;j++)
                t=a.data[k][j],a.data[k][j]=a.data[is[k]][j],a.data[is[k]][j]=t;
        if (js[k]!=k)
            for (i=0;i<a.n;i++)
                t=a.data[i][k],a.data[i][k]=a.data[i][js[k]],a.data[i][js[k]]=t;
        a.data[k][k]=1/a.data[k][k];
        for (j=0;j<a.n;j++)
            if (j!=k)
                a.data[k][j]*=a.data[k][k];
        for (i=0;i<a.n;i++)
            if (i!=k)
                for (j=0;j<a.n;j++)
                    if (j!=k)
                        a.data[i][j]-=a.data[i][k]*a.data[k][j];
        for (i=0;i<a.n;i++)
            if (i!=k)
                a.data[i][k]*=-a.data[k][k];
    }
    for (k=a.n-1;k>=0;k--){
        for (j=0;j<a.n;j++)
            if (js[k]!=k)
                t=a.data[k][j],a.data[k][j]=a.data[js[k]][j],a.data[js[k]][j]=t;
        for (i=0;i<a.n;i++)
            if (is[k]!=k)
                t=a.data[i][k],a.data[i][k]=a.data[i][is[k]],a.data[i][is[k]]=t;
    }
```

```
    return 1;
}

double det(const mat& a){
    int i,j,k,sign=0;
    double b[MAXN][MAXN],ret=1,t;
    if (a.n!=a.m)
        return 0;
    for (i=0;i<a.n;i++)
        for (j=0;j<a.m;j++)
            b[i][j]=a.data[i][j];
    for (i=0;i<a.n;i++){
        if (zero(b[i][i])){
            for (j=i+1;j<a.n;j++)
                if (!zero(b[j][i]))
                    break;
            if (j==a.n)
                return 0;
            for (k=i;k<a.n;k++)
                t=b[i][k],b[i][k]=b[j][k],b[j][k]=t;
            sign++;
        }
        ret*=b[i][i];
        for (k=i+1;k<a.n;k++)
            b[i][k]/=b[i][i];
        for (j=i+1;j<a.n;j++)
            for (k=i+1;k<a.n;k++)
                b[j][k]-=b[j][i]*b[i][k];
    }
    if (sign&1)
        ret=-ret;
    return ret;
}
```

11.4 线性方程组

```
#define MAXN 100
#define fabs(x) ((x)>0?(x):-(x))
#define eps 1e-10

//列主元 gauss 消去求解 a[][]x[]=b[]
//返回是否有唯一解,若有解在 b[]中
int gauss_cpivot(int n,double a[][MAXN],double b[]){
    int i,j,k,row;
```

```
    double maxp,t;
    for (k=0;k<n;k++){
        for (maxp=0,i=k;i<n;i++)
            if (fabs(a[i][k])>fabs(maxp))
                maxp=a[row=i][k];
        if (fabs(maxp)<eps)
            return 0;
        if (row!=k){
            for (j=k;j<n;j++)
                t=a[k][j],a[k][j]=a[row][j],a[row][j]=t;
            t=b[k],b[k]=b[row],b[row]=t;
        }
        for (j=k+1;j<n;j++){
            a[k][j]/=maxp;
            for (i=k+1;i<n;i++)
                a[i][j]-=a[i][k]*a[k][j];
        }
        b[k]/=maxp;
        for (i=k+1;i<n;i++)
            b[i]-=b[k]*a[i][k];
    }
    for (i=n-1;i>=0;i--)
        for (j=i+1;j<n;j++)
            b[i]-=a[i][j]*b[j];
    return 1;
}

//全主元 gauss 消去解 a[][]x[]=b[]
//返回是否有唯一解,若有解在 b[]中
int gauss_tpivot(int n,double a[][MAXN],double b[]){
    int i,j,k,row,col,index[MAXN];
    double maxp,t;
    for (i=0;i<n;i++)
        index[i]=i;
    for (k=0;k<n;k++){
        for (maxp=0,i=k;i<n;i++)
            for (j=k;j<n;j++)
                if (fabs(a[i][j])>fabs(maxp))
                    maxp=a[row=i][col=j];
        if (fabs(maxp)<eps)
            return 0;
        if (col!=k){
            for (i=0;i<n;i++)
                t=a[i][col],a[i][col]=a[i][k],a[i][k]=t;
```

```
                j=index[col],index[col]=index[k],index[k]=j;
            }
            if (row!=k){
                for (j=k;j<n;j++)
                    t=a[k][j],a[k][j]=a[row][j],a[row][j]=t;
                t=b[k],b[k]=b[row],b[row]=t;
            }
            for (j=k+1;j<n;j++){
                a[k][j]/=maxp;
                for (i=k+1;i<n;i++)
                    a[i][j]-=a[i][k]*a[k][j];
            }
            b[k]/=maxp;
            for (i=k+1;i<n;i++)
                b[i]-=b[k]*a[i][k];
        }
        for (i=n-1;i>=0;i--)
            for (j=i+1;j<n;j++)
                b[i]-=a[i][j]*b[j];
        for (k=0;k<n;k++)
            a[0][index[k]]=b[k];
        for (k=0;k<n;k++)
            b[k]=a[0][k];
        return 1;
}
```

11.5 线性相关

```
//判线性相关(正交化)
//传入 m 个 n 维向量
#include <math.h>
#define MAXN 100
#define eps 1e-10

int linear_dependent(int m,int n,double vec[][MAXN]){
        double ort[MAXN][MAXN],e;
        int i,j,k;
        if (m>n)
            return 1;
        for (i=0;i<m;i++){
            for (j=0;j<n;j++)
                ort[i][j]=vec[i][j];
            for (k=0;k<i;k++){
                for (e=j=0;j<n;j++)
```

```
                e+=ort[i][j]*ort[k][j];
            for (j=0;j<n;j++)
                ort[i][j]-=e*ort[k][j];
            for (e=j=0;j<n;j++)
                e+=ort[i][j]*ort[i][j];
            if (fabs(e=sqrt(e))<eps)
                return 1;
            for (j=0;j<n;j++)
                ort[i][j]/=e;
        }
    }
    return 0;
}
```

11.6 日期

```
//日期函数
int days[12]={31,28,31,30,31,30,31,31,30,31,30,31};
struct Date{
   int year, month, day;
};

//判闰年
inline int leap(int year){
   return (year%4==0&&year%100!=0)||year%400==0;
}

//判合法性
inline int legal(Date a){
   if(a.month<0||a.month>12)
      return 0;
   if(a.month==2)
      return a.day>0 && a.day<=28+leap(a.year);
   return a.day>0 && a.day<=days[a.month-1];
}

//比较日期大小
inline int datecmp(Date a, Date b){
    if(a.year != b.year)
       return a.year - b.year;
    if(a.month != b.month)
       return a.month - b.month;
    return a.day - b.day;
}

//返回指定日期是星期几
int weekday(Date a){
   int tm = a.month>=3 ? (a.month-2) : (a.month+10);
   int ty = a.month>=3 ? a.year : (a.year-1);
   return (ty+ty/4-ty/100+ty/400+(int)(2.6*tm-0.2)+a.day)%7;
}
```

```
//日期转天数偏移
int date2int(Date a){
    int ret=a.year*365+(a.year-1)/4-(a.year-1)/100+(a.year-1)/400;
    days[1]+=leap(a.year);
    for(int i=0; i<a.month-1; ret+=days[i++]);
    days[1]=28;
    return ret+a.day;
}

//天数偏移转日期
Date int2date(int a){
    Date ret;
    ret.year = a/146097*400;
    for(a%=146097; a>=365+leap(ret.year); a-=365+leap(ret.year),ret.year++);
    days[1] += leap(ret.year);
    for(ret.month=1; a>=days[ret.month-1]; a-=days[ret.month-1],ret.month++);
    days[1]=28;
    ret.day=a+1;
    return ret;
}
```

11.7 读入

```
#include<sstream>
inline void read(int &data) {
      char ch = getchar();
      while (ch < '0' || ch > '9') ch = getchar();
      data = 0;
      do{
            data = data*10 + ch-'0';
            ch = getchar();
      }while (ch >= '0' && ch <= '9');
}
int main
{
    gets(line);
    istringstream is(line);
    string s;
  while(is >> s)
                {}
}
```

11.8 函数

分类函数,所在函数库为 ctype.h

int isalpha(int ch) 若 ch 是字母('A'-'Z','a'-'z')返回非 0 值,否则返回 0

int isalnum(int ch) 若 ch 是字母('A'-'Z','a'-'z')或数字('0'-'9'),返回非 0 值,否则返回 0

int isascii(int ch) 若 ch 是字符(ASCII 码中的 0-127)返回非 0 值,否则返回 0

int iscntrl(int ch) 若 ch 是作废字符(0x7F)或普通控制字符(0x00-0x1F),返回非 0 值,否则返回 0

int isdigit(int ch) 若 ch 是数字('0'-'9')返回非 0 值,否则返回 0

int isgraph(int ch) 若 ch 是可打印字符(不含空格)(0x21-0x7E)返回非 0 值,否则返回 0
int islower(int ch) 若 ch 是小写字母('a'-'z')返回非 0 值,否则返回 0
int isprint(int ch) 若 ch 是可打印字符(含空格)(0x20-0x7E)返回非 0 值,否则返回 0
int ispunct(int ch) 若 ch 是标点字符(0x00-0x1F)返回非 0 值,否则返回 0
int isspace(int ch) 若 ch 是空格(' '),水平制表符('\t'),回车符('\r'), 走纸换行('\f'),垂直制表符('\v'),换行符('\n'), 返回非 0 值,否则返回 0
int isupper(int ch) 若 ch 是大写字母('A'-'Z')返回非 0 值,否则返回 0
int isxdigit(int ch) 若 ch 是 16 进制数('0'-'9','A'-'F','a'-'f')返回非 0 值, 否则返回 0
int tolower(int ch) 若 ch 是大写字母('A'-'Z')返回相应的小写字母('a'-'z')
int toupper(int ch) 若 ch 是小写字母('a'-'z')返回相应的大写字母('A'-'Z')

数学函数,所在函数库为 math.h、stdlib.h、string.h、float.h
int abs(int i) 返回整型参数 i 的绝对值
double cabs(struct complex znum) 返回复数 znum 的绝对值
double fabs(double x) 返回双精度参数 x 的绝对值
long labs(long n) 返回长整型参数 n 的绝对值
double exp(double x) 返回指数函数 ex 的值
double frexp(double value,int *eptr) 返回 value=x*2n 中 x 的值,n 存贮在 eptr 中
double ldexp(double value,int exp); 返回 value*2exp 的值
double log(double x) 返回 logex 的值
double log10(double x) 返回 log10x 的值
double pow(double x,double y) 返回 xy 的值
double pow10(int p) 返回 10p 的值
double sqrt(double x) 返回 x 的开方
double acos(double x) 返回 x 的反余弦 cos-1(x)值,x 为弧度
double asin(double x) 返回 x 的反正弦 sin-1(x)值,x 为弧度
double atan(double x) 返回 x 的反正切 tan-1(x)值,x 为弧度
double atan2(double y,double x) 返回 y/x 的反正切 tan-1(x)值,y 的 x 为弧度
double cos(double x) 返回 x 的余弦 cos(x)值,x 为弧度
double sin(double x) 返回 x 的正弦 sin(x)值,x 为弧度
double tan(double x) 返回 x 的正切 tan(x)值,x 为弧度
double cosh(double x) 返回 x 的双曲余弦 cosh(x)值,x 为弧度
double sinh(double x) 返回 x 的双曲正弦 sinh(x)值,x 为弧度
double tanh(double x) 返回 x 的双曲正切 tanh(x)值,x 为弧度
double hypot(double x,double y) 返回直角三角形斜边的长度(z), x 和 y 为直角边的长度,z2=x2+y2
double ceil(double x) 返回不小于 x 的最小整数
double floor(double x) 返回不大于 x 的最大整数
void srand(unsigned seed) 初始化随机数发生器
int rand() 产生一个随机数并返回这个数
double poly(double x,int n,double c[]) 从参数产生一个多项式
double modf(double value,double *iptr) 将双精度数 value 分解成尾数和阶
double fmod(double x,double y) 返回 x/y 的余数
double frexp(double value,int *eptr) 将双精度数 value 分成尾数和阶
double atof(char *nptr) 将字符串 nptr 转换成浮点数并返回这个浮点数

double atoi(char *nptr) 将字符串 nptr 转换成整数并返回这个整数
double atol(char *nptr) 将字符串 nptr 转换成长整数并返回这个整数
char *ecvt(double value,int ndigit,int *decpt,int *sign)
将浮点数 value 转换成字符串并返回该字符串
char *fcvt(double value,int ndigit,int *decpt,int *sign)
将浮点数 value 转换成字符串并返回该字符串
char *gcvt(double value,int ndigit,char *buf)
将数 value 转换成字符串并存于 buf 中,并返回 buf 的指针
char *ultoa(unsigned long value,char *string,int radix)
将无符号整型数 value 转换成字符串并返回该字符串,radix 为转换时所用基数
char *ltoa(long value,char *string,int radix)
将长整型数 value 转换成字符串并返回该字符串,radix 为转换时所用基数
char *itoa(int value,char *string,int radix)
将整数 value 转换成字符串存入 string,radix 为转换时所用基数
double atof(char *nptr) 将字符串 nptr 转换成双精度数,并返回这个数,错误返回 0
int atoi(char *nptr) 将字符串 nptr 转换成整型数, 并返回这个数,错误返回 0
long atol(char *nptr) 将字符串 nptr 转换成长整型数,并返回这个数,错误返回 0
double strtod(char *str,char **endptr)将字符串 str 转换成双精度数,并返回这个数,
long strtol(char *str,char **endptr,int base)将字符串 str 转换成长整型数, 并返回这个数,
int matherr(struct exception *e) 用户修改数学错误返回信息函数(没有必要使用)
double _matherr(_mexcep why,char *fun,double *arg1p, double *arg2p,double retval)
用户修改数学错误返回信息函数(没有必要使用)
unsigned int _clear87() 清除浮点状态字并返回原来的浮点状态
void _fpreset() 重新初使化浮点数学程序包
unsigned int _status87() 返回浮点状态字

操作函数,所在函数库为 string.h、mem.h
mem…操作存贮数组
void *memccpy(void *destin,void *source,unsigned char ch,unsigned n)
void *memchr(void *s,char ch,unsigned n)
void *memcmp(void *s1,void *s2,unsigned n)
int memicmp(void *s1,void *s2,unsigned n)
void *memmove(void *destin,void *source,unsigned n)
void *memcpy(void *destin,void *source,unsigned n)
void *memset(void *s,char ch,unsigned n)

这些函数,mem…系列的所有成员均操作存贮数组.在所有这些函数中,数组是 n 字节长.
memcpy 从 source 复制一个 n 字节的块到 destin.如果源块和目标块重迭,则选择复制方向, 以例正确地复制覆盖的字节.
memmove 与 memcpy 相同. memset 将 s 的所有字节置于字节 ch 中.s 数组的长度由 n 给出.
memcmp 比较正好是 n 字节长的两个字符串 s1 和 s2.些函数按无符号字符比较字节,因此,
memcmp("0xFF","\x7F",1)返回值大于 0. memicmp 比较 s1 和 s2 的前 n 个字节,不管字符大写或小写.
memccpy 从 source 复制字节到 destin.复制一结束就发生下列任一情况:
(1)字符 ch 首选复制到 destin.

(2)n 个字节已复制到 destin.
memchr 对字符 ch 检索 s 数组的前 n 个字节.
返回值:memmove 和 memcpy 返回 destin
memset 返回 s 的值
memcmp 和 memicmp┬┬若 s1<s2 返回值小于 0
├─若 s1=s2 返回值等于 0
└─若 s1>s2 返回值大于 0
memccpy 若复制了 ch,则返回直接跟随 ch 的在 destin 中的字节的一个指针;

否则返回 NULL
memchr 返回在 s 中首先出现 ch 的一个指针;如果在 s 数组中不出现 ch,就返回 NULL.

void movedata(int segsrc,int offsrc, int segdest,int offdest, unsigned numbytes)
本函数将源地址(segsrcffsrc)处的 numbytes 个字节复制到目标地址(segdestffdest)
void movemem(void *source,void *destin,unsigned len)
本函数从 source 处复制一块长 len 字节的数据到 destin.若源地址和目标地址字符串重迭,则选择复制方向,以便正确的复制数据.
void setmem(void *addr,int len,char value)
本函数把 addr 所指的块的第一个字节置于字节 value 中.

str…字符串操作函数
char stpcpy(char *dest,const char *src) 将字符串 src 复制到 dest
char strcat(char *dest,const char *src) 将字符串 src 添加到 dest 末尾
char strchr(const char *s,int c) 检索并返回字符 c 在字符串 s 中第一次出现的位置
int strcmp(const char *s1,const char *s2) 比较字符串 s1 与 s2 的大小,并返回 s1-s2
char strcpy(char *dest,const char *src) 将字符串 src 复制到 dest
size_t strcspn(const char *s1,const char *s2) 扫描 s1,返回在 s1 中有,在 s2 中也有的字符个数
char strdup(const char *s) 将字符串 s 复制到最近建立的单元
int stricmp(const char *s1,const char *s2) 比较字符串 s1 和 s2,并返回 s1-s2
size_t strlen(const char *s) 返回字符串 s 的长度
char strlwr(char *s)
将字符串 s 中的大写字母全部转换成小写字母,并返回转换后的字符串
char strncat(char *dest,const char *src,size_t maxlen)
将字符串 src 中最多 maxlen 个字符复制到字符串 dest 中
int strncmp(const char *s1,const char *s2,size_t maxlen)
比较字符串 s1 与 s2 中的前 maxlen 个字符
char strncpy(char *dest,const char *src,size_t maxlen)
复制 src 中的前 maxlen 个字符到 dest 中
int strnicmp(const char *s1,const char *s2,size_t maxlen)
比较字符串 s1 与 s2 中的前 maxlen 个字符
char strnset(char *s,int ch,size_t n)
将字符串 s 的前 n 个字符置于 ch 中
char strpbrk(const char *s1,const char *s2)
扫描字符串 s1,并返回在 s1 和 s2 中均有的字符个数

char strrchr(const char *s,int c)
扫描最后出现一个给定字符 c 的一个字符串 s
char strrev(char *s)
将字符串 s 中的字符全部颠倒顺序重新排列,并返回排列后的字符串
char strset(char *s,int ch)
将一个字符串 s 中的所有字符置于一个给定的字符 ch
size_t strspn(const char *s1,const char *s2)
扫描字符串 s1,并返回在 s1 和 s2 中均有的字符个数
char strstr(const char *s1,const char *s2)
扫描字符串 s2,并返回第一次出现 s1 的位置
char strtok(char *s1,const char *s2)
检索字符串 s1,该字符串 s1 是由字符串 s2 中定义的定界符所分隔
char strupr(char *s)
将字符串 s 中的小写字母全部转换成大写字母,并返回转换后的字符串