Notes for ACMICPC World Finals 2014

ACMICPC World Finals 2014 **参考资料**

Chinese Edition **中文版**

*Shanghai Jiao Tong University* : **Secret;Weapon**



***Coach* 教 练**

Yong YU 俞 勇

***Team member* 队 员**

Xiaoxu GUO 郭晓旭

Qi LIU 刘 奇

Haobin NI 倪昊斌

目录

[计算几何 2](#_Toc390781152)

[三维几何 3](#_Toc390781153)

[三维几何操作合并 5](#_Toc390781154)

[三维旋转操作 6](#_Toc390781155)

[圆的面积模板 7](#_Toc390781156)

[三维凸包随机增量 7](#_Toc390781157)

[三维凸包求重心 9](#_Toc390781158)

[Farmland 9](#_Toc390781159)

[KM 9](#_Toc390781160)

[费用流 10](#_Toc390781161)

[无向图最小割 11](#_Toc390781162)

[一般图最大匹配\_片段 11](#_Toc390781163)

[有向图最小生成树 12](#_Toc390781164)

[Hopcroft 13](#_Toc390781165)

[Manacher 13](#_Toc390781166)

[ExtKMP 13](#_Toc390781167)

[SA 14](#_Toc390781168)

[最大团搜索算法 15](#_Toc390781169)

[极大团的计数 15](#_Toc390781170)

[FFT 16](#_Toc390781171)

[Simpson 16](#_Toc390781172)

[字符串的最小表示 16](#_Toc390781173)

[二次剩余 16](#_Toc390781174)

[Pell方程求解 17](#_Toc390781175)

[Exact Cover 17](#_Toc390781176)

[Link-Cut-Tree 19](#_Toc390781177)

[后缀自动机 20](#_Toc390781178)

[差分序列 20](#_Toc390781179)

[弦图的完美消除序列 20](#_Toc390781180)

[双人零和矩阵游戏（公式） 21](#_Toc390781181)

[质数测试 21](#_Toc390781182)

[Pollard-Rho 21](#_Toc390781183)

[直线下有多少个格点 22](#_Toc390781184)

[综合 22](#_Toc390781185)

[java\_scl 22](#_Toc390781186)

[基本形 公式 22](#_Toc390781187)

[树的计数 23](#_Toc390781188)

[代数 24](#_Toc390781189)

[三角公式 24](#_Toc390781190)

[积分表 24](#_Toc390781191)

## 计算几何

**bool** Contain**(**const Point **&**curr**)** const **{**

int i**,** res **=** 0**;**

Point A**,** B**;**

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

A **=** list**[**i**];**

B **=** list**[(**i **+** 1**)** **%** n**];**

**if** **(**In\_The\_Seg**(**A**,** B**,** curr**))** **return** 1**;**

**if** **(**Sign**(**A**.**y **-** B**.**y**)** **<=** 0**)** swap**(**A**,** B**);**

**if** **(**Sign**(**curr**.**y **-** A**.**y**)** **>** 0**)** **continue;**

**if** **(**Sign**(**curr**.**y **-** B**.**y**)** **<=** 0**)** **continue;**

res **+=** Sign**(**Det**(**B **-** curr**,** A **-** curr**))** **>** 0**;**

**}**

**return** res **&** 1**;**

**}**

**void** CircleCenter(point p0 , point p1 , point p2 , point &cp ){

**double** a1=p1.x-p0.x , b1=p1.y-p0.y , c1=(sqr(a1)+sqr(b1)) / **2** ;

**double** a2=p2.x-p0.x , b2=p2.y-p0.y , c2=(sqr(a2)+sqr(b2)) / **2** ;

**double** d = a1\*b2 - a2\*b1 ;

cp.x = p0.x + ( c1\*b2 - c2\*b1 ) / d ;

cp.y = p0.y + ( a1\*c2 - a2\*c1 ) / d ;

}// 三角形内心

**double** Incenter(point A, point B, point C, point &cp ){

**double** s , p , r , a , b , c ;

a = dis(B, C) , b = dis(C, A) , c = dis(A, B) ; p = (a +b +c) / **2** ;

s = sqrt ( p \* ( p-a ) \* ( p-b ) \* ( p-c ) ) ; r = s / p ;

cp.x = ( a\*A.x + b\*B. x + c\*C.x ) / ( a + b + c ) ;

cp.y = ( a\*A.y + b\*B. y + c\*C.y ) / ( a + b + c ) ;

**ret**urn r ;

}// 三角形 垂心

void Orthocenter(point A, point B, point C, point &cp ){

CircleCenter(A, B, C, cp );

cp.x = A.x + B.x + C.x - 2 \* cp.x ;cp.y = A.y + B.y + C.y - 2 \* cp.y ;}

}//====两园面积交 dist = 是距离 ， dis是平方

**double** twoCircleAreaUnion(point a, point b , **double** r1, **double** r2){

**if** (r1+r2<=(a-b).dist()) **return** **0**;

**if** (r1+(a-b).dist()<=r2) **return** pi\*r1\*r1;

**if** (r2+(a-b).dist()<=r1) **return** pi\*r2\*r2;

**double** c1,c2 , ans=0;

c1=(r1\*r1-r2\*r2+(a-b).dis())/(a-b).dist()/r1/**2.0**;

c2=(r2\*r2-r1\*r1+(a-b).dis())/(a-b).dist()/r2/**2.0**;

**double** s1,s2; s1=acos(c1); s2=acos(c2);

ans+=s1\*r1\*r1-r1\*r1\*sin(s1)\*cos(s1);

ans+=s2\*r2\*r2-r2\*r2\*sin(s2)\*cos(s2);

**return** ans;

}//=====多边形和圆相交的面积用有向面积，划分成一个三角形和圆的面积的交

double area2(point pa, point pb) {

if (pa.len() < pb.len()) swap(pa, pb); if (pb.len() < eps) return 0;

double a, b, c, B, C, sinB, cosB, sinC, cosC, S, h, theta;

a = pb.len(); b = pa.len(); c = (pb-pa).len();

cosB=dot(pb,pb-pa)/a/c; sinB=fabs(det(pb,pb-pa)/a/c);

cosC=dot(pa, pb) / a / b; sinC=fabs(det(pa,pb)/a/b);

B=atan2(sinB , cosB); C=atan2(sinC, cosC);

if (a > r) { S = C/2\*r\*r; h = a\*b\*sinC/c;

if (h < r && B < PI/2) S -= (acos(h/r)\*r\*r - h\*sqrt(r\*r-h\*h));

}

else if (b > r) { theta = PI - B - asin(sinB/r\*a);

S = .5\*a\*r\*sin(theta) + (C-theta)/2\*r\*r; }

else S = .5\*sinC\*a\*b; return S; }// a, b, c, r fixed

double area(const point &o) {

double S = 0; point oa = a-o, ob = b-o, oc = c-o;

S += area2(oa, ob) \* sign(det(oa, ob));

S += area2(ob, oc) \* sign(det(ob, oc));

S += area2(oc, oa) \* sign(det(oc, oa)); return abs(S);

}//======半平面交

**void** rebuild(point a, point b){//逆时针 ,ab左侧

**int** i,t;**double** k1,k2;sol[m]=sol[**0**]; t=**0**;

**foru**(i,**1**,m){ k1=area(a,b,sol[i]); k2=area(a,b,sol[i-**1**]);

**if** (cmp(k1)\*cmp(k2)<**0**){

tmp[t].x=(sol[i].x\*k2-sol[i-**1**].x\*k1) / (k2-k1);

tmp[t].y=(sol[i].y\*k2-sol[i-**1**].y\*k1) / (k2-k1); t++;

} **if** (cmp(area(a,b,sol[i])) >=**0**){ tmp[t]=sol[i]; t++;}}

m=t; **rep**(i,m) sol[i]=tmp[i];

}//====nlogn半平面交

**bool** check**(const** Plane **&**u**,** **const** Plane **&**v**,** **const** Plane **&**w**)** **{**

**return** intersect**(**u**,** v**).**in**(**w**);**

**}**

**void** build**(**vector **<**Plane**>** planes**)** **{**

**int** head **=** 0, tail **=** 0**;**

**for** **(int** i **=** 0**;** i **<** **(int)**planes**.**size**();** **++** i**)** **{**

**while** **(**tail **-** head **>** 1 **&&** **!**check**(**queue**[**tail **-** 2**],** queue**[**tail **-** 1**],** planes**[**i**]))** **{**

tail **--;**

**}**

**while** **(**tail **-** head **>** 1 **&&** **!**check**(**queue**[**head **+** 1**],** queue**[**head**],** planes**[**i**]))** **{**

head **++;**

**}**

queue**[**tail **++]** **=** planes**[**i**];**

**}**

**while** **(**tail **-** head **>** 2 **&&** **!**check**(**queue**[**tail **-** 2**],** queue**[**tail **-** 1**],** queue**[**head**]))** **{**

tail **--;**

**}**

**while** **(**tail **-** head **>** 2 **&&** **!**check**(**queue**[**head **+** 1**],** queue**[**head**],** queue**[**tail **-** 1**]))** **{**

head **++;**

**}**

**}**

## 三维几何

**//vlen(point3 P):length of vector; zero(double x):if fabs(x)<eps) return true;**

double vlen(point3 p);

**//平面法向量**

point3 pvec(point3 s1,point3 s2,point3 s3){return det((s1-s2),(s2-s3));}

**//check共线**

int dots\_inline(point3 p1,point3 p2,point3 p3){

return vlen(det(p1-p2,p2-p3))<eps;}

**//check共平面**

int dots\_onplane(point3 a,point3 b,point3 c,point3 d){

return zero(dot(pvec(a,b,c),d-a));}

**//check在线段上(end point inclusive)**

int dot\_online\_in(point3 p,line3 l)

int dot\_online\_in(point3 p,point3 l1,point3 l2){return zero(vlen(det(p-l1,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; }

**//check在线段上(end point exclusive)**

int dot\_online\_ex(point3 p,line3 l)

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));

}

**//check一个点是否在三角形里(inclusive)**

int dot\_inplane\_in(point3 p,plane3 s)

int dot\_inplane\_in(point3 p,point3 s1,point3 s2,point3 s3){

return zero(vlen(det(s1-s2,s1-s3))-vlen(det(p-s1,p-s2))-

vlen(det(p-s2,p-s3))-vlen(det(p-s3,p-s1)));

}

**//check一个点是否在三角形里(exclusive)**

int dot\_inplane\_ex(point3 p,plane3 s)

int dot\_inplane\_ex(point3 p,point3 s1,point3 s2,point3 s3){

return dot\_inplane\_in(p,s1,s2,s3)&&vlen(det(p-s1,p-s2))>eps&&

vlen(det(p-s2,p-s3))>eps&&vlen(det(p-s3,p-s1))>eps;

}

**//check if two point and a segment in one plane have the same side**

int same\_side(point3 p1,point3 p2,point3 l1,point3 l2)

int same\_side(point3 p1,point3 p2,line3 l){

return dot(det(l.a-l.b,p1-l.b),det(l.a-l.b,p2-l.b))>eps;

}

**//check if two point and a segment in one plane have the opposite side**

int opposite\_side(point3 p1,point3 p2,point3 l1,point3 l2)

int opposite\_side(point3 p1,point3 p2,line3 l){

return dot(det(l.a-l.b,p1-l.b), det(l.a-l.b,p2-l.b))<-eps;

}

**//check if two point is on the same side of a plane**

int same\_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3)

int same\_side(point3 p1,point3 p2,plane3 s){

return dot(pvec(s),p1-s.a)\*dot(pvec(s),p2-s.a)>eps;

}

**//check if two point is on the opposite side of a plane**

int opposite\_side(point3 p1,point3 p2,point3 s1,point3 s2,point3 s3)

int opposite\_side(point3 p1,point3 p2,plane3 s){

return dot(pvec(s),p1-s.a)\*dot(pvec(s),p2-s.a)<-eps;

}

**//check if two straight line is parallel**

int parallel(point3 u1,point3 u2,point3 v1,point3 v2)

int parallel(line3 u,line3 v){ return vlen(det(u.a-u.b,v.a-v.b))<eps; }

**//check if two plane is parallel**

int parallel(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3)

int parallel(plane3 u,plane3 v){return vlen(det(pvec(u),pvec(v)))<eps;}

**//check if a plane and a line is parallel**

int parallel(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3)

int parallel(line3 l,plane3 s){ return zero(dot(l.a-l.b,pvec(s))); }

**//check if two line is perpendicular**

int perpendicular(point3 u1,point3 u2,point3 v1,point3 v2)

int perpendicular(line3 u,line3 v){return zero(dot(u.a-u.b,v.a-v.b)); }

**//check if two plane is perpendicular**

int perpendicular(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3)

int perpendicular(plane3 u,plane3 v){ return zero(dot(pvec(u),pvec(v))); }

**//check if plane and line is perpendicular**

int perpendicular(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3)

int perpendicular(line3 l,plane3 s){return vlen(det(l.a-l.b,pvec(s)))<eps;}

**//check 两条线段是否有交点(end point inclusive)**

int intersect\_in(point3 u1,point3 u2,point3 v1,point3 v2)

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);

}

**//check 两条线段是否有交点(end point exclusive)**

int intersect\_ex(point3 u1,point3 u2,point3 v1,point3 v2)

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);

}

**//check线段和三角形是否有交点(end point and border inclusive)**

int intersect\_in(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3)

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);

}

**//check线段和三角形是否有交点(end point and border exclusive)**

int intersect\_ex(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3)

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);}

**//calculate the intersection of two line**

**//Must you should ensure they are co-plane and not parallel**

point3 intersection(point3 u1,point3 u2,point3 v1,point3 v2)

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+=(u.b-u.a)\*t; return ret;

}

**//calculate the intersection of plane and line**

point3 intersection(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3)

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=l.a + (l.b-l.a)\*t; return ret;

}

**//calculate the intersection of two plane**

bool intersection(plane3 pl1 , plane3 pl2 , line3 &li) {

if (parallel(pl1,pl2)) return false;

li.a=parallel(pl2.a,pl2.b, pl1) ? intersection(pl2.b,pl2.c, pl1.a,pl1.b,pl1.c) : intersection(pl2.a,pl2.b, pl1.a,pl1.b,pl1.c);

point3 fa; fa=det(pvec(pl1),pvec(pl2)); li.b=li.a+fa; return true;

}

**//distance from point to line**

double ptoline(point3 p,point3 l1,point3 l2)

double ptoline(point3 p,line3 l){

return vlen(det(p-l.a,l.b-l.a))/distance(l.a,l.b);}

**//distance from point to plane**

double ptoplane(point3 p,plane3 s){

return fabs(dot(pvec(s),p-s.a))/vlen(pvec(s));}

double ptoplane(point3 p,point3 s1,point3 s2,point3 s3)

**//distance between two line 当u,v平行时有问题**

double linetoline(line3 u,line3 v){

point3 n=det(u.a-u.b,v.a-v.b); return fabs(dot(u.a-v.a,n))/vlen(n);

}

double linetoline(point3 u1,point3 u2,point3 v1,point3 v2)

**//cosine value of the angle formed by two lines**

double angle\_cos(line3 u,line3 v){

return dot(u.a-u.b,v.a-v.b)/vlen(u.a-u.b)/vlen(v.a-v.b);

}

double angle\_cos(point3 u1,point3 u2,point3 v1,point3 v2)

**//cosine value of the angle formed by two planes**

double angle\_cos(plane3 u,plane3 v){

return dot(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)

**//cosine value of the angle formed by plane and line**

double angle\_sin(line3 l,plane3 s){

return dot(l.a-l.b,pvec(s))/vlen(l.a-l.b)/vlen(pvec(s));}

double angle\_sin(point3 l1,point3 l2,point3 s1,point3 s2,point3 s3)

## 三维几何操作合并

**const** **double** pi = **acos(-1.0)**; **double** a[**4**][**4**];

**int** dcmp(**const** **double** &a, **const** **double** &b = **0**, **const** **double** & zero = **1e**-**6**){

**if** (a - b < -zero) **return** -**1**; **return** a - b > zero;}

**void** multi(**const** **double** a[**4**][**4**],**const** **double** b[**4**][**4**],**double** c[**4**][**4**]){

**for**(**int** i=**0**;i<**4**;i++)**for**(**int** j=**0**;j<**4**;j++){

c[i][j]=a[i][**0**]\*b[**0**][j]; **for**(**int** k=**1**;k<**4**;k++) c[i][j]+=a[i][k]\*b[k][j];

}}

**void** multi(**double** a[**4**][**4**],**const** **double** b[**4**][**4**]){

**static** **double** c[**4**][**4**]; multi(a,b,c);memcpy(a,c,**sizeof**(a[**0**][**0**])\***16**);

}

**void** Macro(){

**double** b[**4**][**4**]={**1**, **0**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **0**, **1**};

memcpy(a,b,**sizeof**(a[**0**][**0**])\***16**);

}

**void** Translation(**const** Point\_3 &s){

**double** p[**4**][**4**]={**1**, **0**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **0**, **1**, **0**, s.x, s.y, s.z, **1**};

multi(a,p);

}

**void** Scaling(**const** Point\_3 &s){

**double** p[**4**][**4**]={s.x, **0**, **0**, **0**, **0**, s.y, **0**, **0**, **0**, **0**, s.z, **0**, **0**, **0**, **0**, **1**};

multi(a,p);

}

**void** Rotate(**const** Point\_3 &s, **double** r) {

**double** l=s.Length(); **double** x=s.x/l,y=s.y/l,z=s.z/l;

**double** SinA=sin(r),CosA=cos(r);

**double** p[**4**][**4**]={CosA + (**1** - CosA) \* x \* x, (**1** - CosA) \* x \* y - SinA \* z, (**1** - CosA) \* x \* z + SinA \* y, **0**,(**1** - CosA) \* y \* x + SinA \* z,

CosA + (**1** - CosA) \* y \* y, (**1** - CosA) \* y \* z - SinA \* x, **0**,

(**1** - CosA) \* z \* x - SinA \* y, (**1** - CosA) \* z \* y + SinA \* x, CosA + (**1** - CosA) \* z \* z, **0**, **0**, **0**, **0**, **1**};

multi(a,p);

}

Point\_3 opt(**const** Point\_3&s){

**double** x,y,z;

**return** Point\_3( s.x \* a[**0**][**0**] + s.y \* a[**1**][**0**] + s.z \* a[**2**][**0**] + a[**3**][**0**],

s.x \* a[**0**][**1**] + s.y \* a[**1**][**1**] + s.z \* a[**2**][**1**] + a[**3**][**1**],

s.x \* a[**0**][**2**] + s.y \* a[**1**][**2**] + s.z \* a[**2**][**2**] + a[**3**][**2**]);

}

**int** main(){

Macro();

**int** n;**for** (scanf(**"%d"**, &n); n; n--) {

**char** c; Point\_3 p;

scanf(**"\n%c%lf%lf%lf"**, &c, &p.x, &p.y, &p.z);

**if** (c == **'T'**) Translation(p); **if** (c == **'S'**) Scaling(p);

**if** (c == **'R'**) { **double** r;scanf(**"%lf\n"**, &r);

Rotate(p, r); //===========绕OP逆时针旋转r角度

}}

**for** (scanf(**"%d"**, &n); n; n--) {

Point\_3 p, p2; scanf(**"%lf%lf%lf"**, &p.x, &p.y, &p.z);

p2 = opt(p); printf(“%f %f %f\n”,p2.x,p2.y,p2.z);

}}

## 三维旋转操作

//a点绕Ob向量，逆时针旋转弧度angle, sin(angle),cos(angle)先求出来，减少精度问题。

point e1,e2,e3; point Rotate( point a, point b, **double** angle ){

b.std();//单位化，注意b不能为（0，0，0）

e3=b; **double** lens=a\*e3;//dot(a,e3)

e1=a - e3\*lens; **if** (e1.len()>(**1e**-**8**)) e1.std(); **else** **return** a;

e2=e1/e3; //det(e1,e3)

**double** x1,y1,x,y; y1=a\*e1; x1=a\*e2;

x=x1\*cos(angle) - y1\*sin(angle); y=x1\*sin(angle) + y1\*cos(angle);

**return** e3\*lens + e1\*y + e2\*x; }

## 圆的面积模板

struct Tevent {

point p;double ang;int add;

bool operator <(const Tevent &a) const {

return ang < a.ang;

}

} eve[maxn \* 2];

int E, cnt;

void circleCrossCircle(const Tcir &a, const Tcir &b) {

double l = (a.o - b.o).len2();

double s = ((a.r - b.r) \* (a.r + b.r) / l + 1) \* .5;

double t = sqrt(-(l - sqr(a.r - b.r)) \* (l - sqr(a.r + b.r)) / (l \* l \* 4.));

point dir = b.o - a.o;

point Ndir = point(-dir.y, dir.x);

point aa = a.o + dir \* s + Ndir \* t;

point bb = a.o + dir \* s - Ndir \* t;

double A = atan2(aa.y - a.o.y, aa.x - a.o.x);

double B = atan2(bb.y - a.o.y, bb.x - a.o.x);

eve[E++] = Tevent(bb, B, 1);

eve[E++] = Tevent(aa, A, -1);

if (B > A) {

cnt++;

}

} //必须去掉重复的圆 Overlap[i][j] i包含j g[i][j] i和j相交

bool g[maxn][maxn], Overlap[maxn][maxn];

double Area[maxn];

Tcir c[maxn];

int C;

int main() {

scanf("%d", &C);

for (int i = 0; i <= C; ++i) Area[i] = 0;

for (int i = 0; i < C; ++i) {

E = 0;

cnt = 1;

for (int j = 0; j < C; ++j) if (j != i && Overlap[j][i]) cnt++;

for (int j = 0; j < C; ++j) {

if (i != j && g[i][j]) {

circleCrossCircle(c[i], c[j]);

}

}

//cnt表示覆盖次数超过cnt

if (E == 0) {

Area[cnt] += PI \* c[i].r \* c[i].r;

} else {

double counts = 0;

sort(eve, eve + E);

eve[E] = eve[0];

for (int j = 0; j < E; ++j) {

cnt += eve[j].add;

Area[cnt] += cross(eve[j].p, eve[j + 1].p) \* .5;

double theta = eve[j + 1].ang - eve[j].ang;

if (theta < 0) theta += PI \* 2.;

Area[cnt] += theta \* c[i].r \* c[i].r \* .5 - sin(theta) \* c[i].r \* c[i].r \* .5;

}

}

}

}

## 三维凸包随机增量

**struct** Point { **double** x, y, z; Point() {x = y = z = **0**;}

Point(**double** x, **double** y, **double** z): x(x), y(y), z(z) {}

**bool** **operator** <(**const** Point &p) **const** {x,y,z}

**bool** **operator** ==(**const** Point &p) **const** {}

Point cross(**const** Point &p) **const** {

**return** Point(y \* p.z - z \* p.y, z \* p.x - x \* p.z, x \* p.y - y \* p.x);}

**double** dot(**const** Point &p) **const** { **return** x \* p.x + y \* p.y + z \* p.z; }

**double** norm() {**return** dot(\***this**);} **double** length() {**return** Sqrt(norm());}

};

**int** mark[**1005**][**1005**];Point info[**1005**];**int** n, cnt;

**double** mix(**const** Point &a, **const** Point &b, **const** Point &c) {

**return** a.dot(b.cross(c));}

**double** area(**int** a, **int** b, **int** c) {

**return** ((info[b] - info[a]).cross(info[c] - info[a])).length();}

**double** volume(**int** a, **int** b, **int** c, **int** d) {

**return** mix(info[b] - info[a], info[c] - info[a], info[d] - info[a]);}

**struct** Face { **int** a, b, c; Face() {}

Face(**int** a, **int** b, **int** c): a(a), b(b), c(c) {}

**int** &**operator** [](**int** k) {**if** (k == **0**) **return** a; **if** (k == **1**) **return** b; **return** c; }};

vector <Face> face;

**inline** **void** insert(**int** a, **int** b, **int** c) { face.push\_back(Face(a, b, c)); }

**void** add(**int** v) {

vector <Face> tmp; **int** a, b, c; cnt ++;

**for** (**int** i = **0**; i < SIZE(face); i ++) {

a = face[i][**0**]; b = face[i][**1**]; c = face[i][**2**];

**if** (Sign(volume(v, a, b, c)) < **0**)

mark[a][b]=mark[b][a]=mark[b][c] = mark[c][b] = mark[c][a] = mark[a][c] = cnt;

**else** tmp.push\_back(face[i]); }

face = tmp;

**for** (**int** i = **0**; i < SIZE(tmp); i ++) {

a = face[i][**0**]; b = face[i][**1**]; c = face[i][**2**];

**if** (mark[a][b] == cnt) insert(b, a, v);

**if** (mark[b][c] == cnt) insert(c, b, v);

**if** (mark[c][a] == cnt) insert(a, c, v);

}}

**int** Find() {

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

Point ndir = (info[**0**] - info[i]).cross(info[**1**] - info[i]);

**if** (ndir == Point()) **continue**;

swap(info[i], info[**2**]);

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

**if** (Sign(volume(**0**, **1**, **2**, j)) != **0**) {

swap(info[j], info[**3**]); insert(**0**, **1**, **2**); insert(**0**, **2**, **1**);

**return** **1**;

} } **return** **0**;}

**int** main() {

**double** ans, ret; **int** Case;

**for** (scanf(**"%d"**, &Case); Case; Case --) {

scanf(**"%d"**, &n); **for** (**int** i = **0**; i < n; i ++) info[i].read();

sort(info, info + n); n = unique(info, info + n) - info;

face.clear(); random\_shuffle(info, info + n);

ans = ret = **0**; **if** (Find()) {

memset(mark, **0**, **sizeof**(mark)); cnt = **0**;

**for** (**int** i = **3**; i < n; i ++) add(i);

**int** first = face[**0**][**0**];

**for** (**int** i = **0**; i < SIZE(face); i ++) {

ret += area(face[i][**0**], face[i][**1**], face[i][**2**]);

ans += fabs(volume(first, face[i][**0**], face[i][**1**], face[i][**2**]));

} ans /= **6**; ret /= **2**; }

printf(**"%.3f %.3f\n"**, ret, ans);

} **return** **0**; }

## 三维凸包求重心

double calcDist(const Point &p, int a, int b, int c) {  
    return fabs(mix(info[a] - p, info[b] - p, info[c] - p) / area(a, b, c));  
}  
//compute the minimal distance of center of any faces  
double findDist() {  
    //compute center of mass  
    double totalWeight = 0;  
    Point center(.0, .0, .0);  
    Point first = info[face[0][0]];  
    for (int i = 0; i < SIZE(face); ++i) {  
        Point p = (info[face[i][0]] + info[face[i][1]] + info[face[i][2]] + first) \* .25;  
        double weight = mix(info[face[i][0]] - first, info[face[i][1]] - first, info[face[i][2]] - first);   
        totalWeight += weight;  
        center = center + p \* weight;  
    }  
    center = center / totalWeight;  
    //compute distance   
    double res = 1e100;  
    for (int i = 0; i < SIZE(face); ++i) {  
        res = min(res, calcDist(center, face[i][0], face[i][1], face[i][2]));  
    }  
    return res;  
}

## Farmland

int farmland()

{

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

{

vector <pair <double,int> > lq;

for (int j=b[i];j;j=a[j][1])

{

int x=a[j][0];

lq.push\_back(make\_pair(atan2(p[x].y-p[i].y,p[x].x-p[i].x),j));

}

sort(lq.begin(),lq.end());

for (int j=0;j<lq.size();j++)

to[lq[(j+1)%lq.size()].second^1]=lq[j].second;

}

memset(vis,0,sizeof(vis));

int ans=0;

for (int i=2;i<=tot;i++)

if (!vis[i])

{

ll area=0;

for (int j=i;!vis[j];j=to[j])

{

area+=det(p[a[j^1][0]],p[a[j][0]]);

vis[j]=true;

}

if (area>0)

ans++;

}

return(ans);

}

## KM

**const** **int** maxn=**200**;**const** **int** oo=**0x7fffffff**;

**int** w[maxn][maxn],x[maxn],y[maxn],px[maxn],py[maxn],sy[maxn],slack[maxn];

int par[maxn];**int** n;**int** pa[**200**][**2**],pb[**200**][**2**],n0,m0,na,nb;**char** s[**200**][**200**];

**void** adjust(**int** v){ sy[v]=py[v]; **if** (px[sy[v]]!=-**2**) adjust(px[sy[v]]);}

**bool** find(**int** v){**for** (int i=**0**;i<n;i++)

**if** (py[i]==-**1**){

**if** (slack[i]>x[v]+y[i]-w[v][i]){

slack[i]=x[v]+y[i]-w[v][i]; par[i]=v;}

**if** (x[v]+y[i]==w[v][i]){

py[i]=v; **if** (sy[i]==-**1**){adjust(i); **return** **1**;}

**if** (px[sy[i]]!=-**1**) **continue**; px[sy[i]]=i;

**if** (find(sy[i])) **return** **1**;

}}**return** **0**;}

**int** km(){**int** i,j,m;

**for** (i=**0**;i<n;i++) sy[i]=-**1**,y[i]=**0**;

**for** (i=**0**;i<n;i++) {x[i]=**0**; **for** (j=**0**;j<n;j++) x[i]=max(x[i],w[i][j]);}

**bool** flag;

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

**for** (j=**0**;j<n;j++) px[j]=py[j]=-**1**,slack[j]=oo;

px[i]=-**2**; **if** (find(i)) **continue**; flag=**false**;

**for** (;!flag;){

m=oo; **for** (j=**0**;j<n;j++) **if** (py[j]==-**1**) m=min(m,slack[j]);

**for** (j=**0**;j<n;j++){

**if** (px[j]!=-**1**) x[j]-=m;

**if** (py[j]!=-**1**) y[j]+=m;

**else** slack[j]-=m;}

**for** (j=**0**;j<n;j++){

**if** (py[j]==-**1**&&!slack[j]){

py[j]=par[j];

**if** (sy[j]==-**1**){ adjust(j); flag=**true**; **break**;}

px[sy[j]]=j; **if** (find(sy[j])){flag=**true**;**break**;}

}}}}

**int** ans=**0**; **for** (i=**0**;i<n;i++) ans+=w[sy[i]][i];**return** ans;}

## 费用流

int modlable**()** **{**

int delta **=** inf**;**

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

**if** **(!**visit**[**i**]** **&&** slack**[**i**]** **<** delta**)**

delta **=** slack**[**i**];**

slack**[**i**]** **=** inf**;**

**}**

**if** **(**delta **==** inf**)**

**return** 1**;**

**for(**int i **=** 1**;** i **<=** T**;** i**++)**

**if** **(**visit**[**i**])**

dis**[**i**]** **+=** delta**;**

**return** 0**;**

**}**

int dfs**(**int x**,** int flow**)** **{**

**if** **(**x **==** T**)** **{**

totFlow **+=** flow**;**

totCost **+=** flow **\*** **(**dis**[**S**]** **-** dis**[**T**]);**

**return** flow**;**

**}**

visit**[**x**]** **=** 1**;**

int left **=** flow**;**

**for(**int i **=** e**.**last**[**x**];** **~**i**;** i **=** e**.**succ**[**i**])**

**if** **(**e**.**cap**[**i**]** **>** 0 **&&** **!**visit**[**e**.**other**[**i**]])** **{**

int y **=** e**.**other**[**i**];**

**if** **(**dis**[**y**]** **+** e**.**cost**[**i**]** **==** dis**[**x**])** **{**

int delta **=** dfs**(**y**,** min**(**left**,** e**.**cap**[**i**]));**

e**.**cap**[**i**]** **-=** delta**;**

e**.**cap**[**i **^** 1**]** **+=** delta**;**

left **-=** delta**;**

**if** **(!**left**)** **{**

visit**[**x**]** **=** 0**;**

**return** flow**;**

**}**

**}** **else** **{**

slack**[**y**]** **=** min**(**slack**[**y**],** dis**[**y**]** **+** e**.**cost**[**i**]** **-** dis**[**x**]);**

**}**

**}**

**return** flow **-** left**;**

**}**

pair**<**int**,** int**>** minCost**()** **{**

totFlow **=** 0**,** totCost **=** 0**;**

fill**(**dis **+** 1**,** dis **+** T **+** 1**,** 0**);**

**do** **{**

**do** **{**

fill**(**visit **+** 1**,** visit **+** T **+** 1**,** 0**);**

**}** **while(**dfs**(**S**,** inf**));**

**}** **while(!**modlable**());**

**return** make\_pair**(**totFlow**,** totCost**);**

**}**

## 无向图最小割

#define typec int // type of res (or long long)

**const** typec inf = **0x3f3f3f3f**; // max of res

**const** typec maxw = **1000**; // maximum edge weight, g[i][j]=g[j][i]

typec g[V][V], w[V]; **int** a[V], v[V], na[V];

typec mincut(**int** n){

**int** i, j, pv, zj; typec best = maxw \* n \* n;

**for** (i = **0**; i < n; i++) v[i] = i; // vertex: 0 ~ n-1

**while** (n > **1**) {

**for** (a[v[**0**]] = **1**, i = **1**; i < n; i++) {

a[v[i]] = **0**; na[i - **1**] = i; w[i] = g[v[**0**]][v[i]];}

**for** (pv = v[**0**], i = **1**; i < n; i++ ) {

**for** (zj = -**1**, j = **1**; j < n; j++ )

**if** (!a[v[j]] && (zj < **0** || w[j] > w[zj])) zj = j;

a[v[zj]] = **1**;

**if** (i == n - **1**) {

**if** (best > w[zj]) best = w[zj];

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

g[v[i]][pv] = g[pv][v[i]]+=g[v[zj]][v[i]];

v[zj] = v[--n]; **break**;

} pv = v[zj];

**for** (j = **1**; j < n; j++) **if**(!a[v[j]]) w[j] += g[v[zj]][v[j]];

}} **return** best;}

## 一般图最大匹配\_片段

**const** **int** maxn=**310**;

vector<**int**> link[maxn];

**int** n; **int** match[maxn]; **int** Queue[maxn], head, tail; **int** pred[maxn], base[maxn];

**bool** InQueue[maxn], InBlossom[maxn]; **bool** use[maxn]; //===这个点是否有用

**int** start, finish; **int** newbase;

**void** push(**int** u) { Queue[tail++] = u; InQueue[u] = **true**; }

**int** pop() { **return** Queue[head++];}

**int** FindCommonAncestor(**int** u, **int** v) {

**bool** InPath[maxn]; **for** (**int** i = **0**; i < n; i++) InPath[i] = **0**;

**while**(**true**) {

u = base[u]; InPath[u] = **true**;

**if**(u == start) **break**; u = pred[match[u]];}

**while**(**true**) {v = base[v]; **if**(InPath[v]) **break**; v = pred[match[v]]; }

**return** v;}

**void** ResetTrace(**int** u) {

**int** v;

**while**(base[u] != newbase) {

v = match[u]; InBlossom[base[u]] = InBlossom[base[v]] = **true**;

u = pred[v]; **if**(base[u] != newbase) pred[u] = v;}}

**void** BlossomContract(**int** u, **int** v) {

newbase = FindCommonAncestor(u, v);

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

ResetTrace(u); ResetTrace(v);

**if**(base[u] != newbase) pred[u]=v;**if**(base[v] != newbase) pred[v]=u;

**for**(**int** i = **0**; i < n; ++i)

**if**(InBlossom[base[i]]) {base[i]=newbase; **if**(!InQueue[i]) push(i);}}

**bool** FindAugmentingPath(**int** u) {

**bool** found = **false**;

**for**(**int** i = **0**; i < n; ++i) pred[i] = -**1**, base[i] = i;

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

start = u; finish = -**1**; head = tail = **0**; push(start);

**while**(head < tail) {

**int** u = pop();

**for**(**int** i = link[u].size() - **1**; i >= **0**; i--) {

**int** v = link[u][i];

**if**(use[u] && use[v] && base[u] != base[v] && match[u] != v)

**if**(v == start || (match[v] >= **0** && pred[match[v]] >= **0**))

BlossomContract(u, v);

**else** **if**(pred[v] == -**1**) {pred[v] = u;

**if**(match[v] >= **0**) push(match[v]);

**else** {finish = v; **return** **true**;}

}}} **return** found;}

**void** AugmentPath() {

**int** u, v, w; u = finish;

**while**(u >= **0**) { v = pred[u];w = match[v];match[v] = u; match[u] = v;u = w;}}

**void** FindMaxMatching() {

**for**(**int** i = **0**; i < n; ++i) match[i] = -**1**;

**for**(**int** i = **0**; i < n; ++i)

**if**(match[i] == -**1** && use[i])**if**(FindAugmentingPath(i)) AugmentPath();}

**int** main() {

**foru**(i,**0**,n) link[i].clear(); memset(use,**1**,**sizeof**(use));

//========编号从0~n-1 ， link[i] push\_back所有i号点连向的点。 双向边

FindMaxMatching(); k=**0**;**rep**(i,n) **if** (match[i]>=**0**) k++;

printf(**"%d\n"**,k/**2**); **return** **0**;

}

## 有向图最小生成树

**const** **int** maxn=**1100**; **int** n,m , g[maxn][maxn] , used[maxn] , pass[maxn] ;

**int** eg[maxn] , more , queue[maxn];

**void** combine (**int** id , **int** &sum ) {

**int** tot = **0** , from , i , j , k ;

**for** ( ; id!=**0** && !pass[ id ] ; id=eg[id] ) {

queue[tot++]=id ; pass[id]=**1**; }

**for** ( from=**0**; from<tot && queue[from]!=id ; from++);

**if** ( from==tot ) **return** ;

more = **1** ;

**for** ( i=from ; i<tot ; i++) {

sum+=g[eg[queue[i]]][queue[i]] ;

**if** ( i!=from ) {

used[queue[i]]=**1**;

**for** ( j = **1** ; j <= n ; j++) **if** ( !used[j] )

**if** ( g[queue[i]][j]<g[id][j] ) g[id][j]=g[queue[i]][j] ;}}

**for** ( i=**1**; i<=n ; i++) **if** ( !used[i] && i!=id ) {

**for** ( j=from ; j<tot ; j++){ k=queue[j];

**if** ( g[i][id]>g[i][k]-g[eg[k]][k] ) g[i][id]=g[i][k]-g[eg[k]][k]; }}}

**int** mdst( **int** root ) { // return the total length of MDST

**int** i , j , k , sum = **0** ;

memset ( used , **0** , **sizeof** ( used ) ) ;

**for** ( more =**1**; more ; ) {

more = **0** ; memset (eg,**0**,**sizeof**(eg)) ;

**for** ( i=**1** ; i <= n ; i ++) **if** ( !used[i] && i!=root ) {

**for** ( j=**1** , k=**0** ; j <= n ; j ++) **if** ( !used[j] && i!=j )

**if** ( k==**0** || g[j][i] < g[k][i] ) k=j ;

eg[i] = k ;

} memset(pass,**0**,**sizeof**(pass));

**for** ( i=**1**;i<=n;i++) **if** (!used[i] && !pass[i] && i!= root )combine(i,sum);

}

**for** ( i =**1**; i<=n ; i ++) **if** ( !used[i] && i!= root ) sum+=g[eg[i]][i];

**return** sum ; }

**int** main(){

**int** i,j,k,test,cases; cases=**0**; scanf(**"%d%d"**,&n,&m);

**foru**(i,**1**,n) **foru**(j,**1**,n) g[i][j]=**1000001**;

**foru**(i,**1**,m) {scanf(**"%d%d"**,&j,&k);j++;k++;scanf(**"%d"**,&g[j][k]);}

k=mdst(**1**); **if** (k>**1000000**) printf(**"Possums!\n"**); //===no

**else** printf(**"%d\n"**,k); **return** **0**;}

## Hopcroft

#define maxn 50005 #define maxm 150005

**inline** **int** Maxmatch(){

memset(mk,-**1**,**sizeof**(mk));memset(cx,-**1**,**sizeof**(cx));

memset(cy,-**1**,**sizeof**(cy));

**for** (**int** p=**1**,fl=**1**,h,tail;fl;++p){

fl=**0**; h=tail=**0**;

**for** (**int** i=**0**;i<n;++i) **if** (cx[i]==-**1**)

q[++tail]=i,pre[i]=-**1**,src[i]=i;

**for** (h=**1**;h<=tail;++h){

**int** u=q[h]; **if** (cx[src[u]]!=-**1**) **continue**;

**for** (**int** pp=head[u],v=vtx[pp];pp;pp=next[pp],v=vtx[pp])

**if** (mk[v]!=p) { mk[v]=p; q[++tail]=cy[v];

**if** (cy[v]>=**0**) {

pre[cy[v]]=u; src[cy[v]]=src[u];**continue**;

} **int** d,e,t;

**for** (--tail,fl=**1**,d=u,e=v;d!=-**1**;t=cx[d],cx[d]=e,cy[e]=d,e=t,d=pre[d]);

**break**; } } }

**int** res=**0**; **for** (**int** i=**0**;i<n;++i) res+=(cx[i]!=-**1**);**return** res;}

## Manacher

**void** manacher**(char** text**[],** **int** n**,** **int** palindrome**[])** **{**

palindrome**[**0**]** **=** 1**;**

**for** **(int** i **=** 1**,** j **=** 0**,** i **<** **(**n **<<** 1**)** **-** 1**;** **++** i**)** **{**

**int** p **=** i **>>** 1**;**

**int** q **=** i **-** p**;**

**int** r **=** **(**j **+** 1 **>>** 1**)** **+** palindrome**[**j**]** **-** 1**;**

palindrome**[**i**]** **=** r **<** q**?** 0**:** min**(**r **-** q **+** 1**,** palindrome**[(**j **<<** 1**)** **-** i**]);**

**while** **(**0 **<=** p **-** palindrome**[**i**]** **&&** q **+** palindrome**[**i**]** **<** n

**&&** text**[**p **-** palindrome**[**i**]]** **==** text**[**q **+** palindrome**[**i**]])** **{**

palindrome**[**i**]** **++;**

**}**

**if** **(**q **+** palindrome**[**i**]** **-** 1 **>** r**)** **{**

j **=** i**;**

**}**

**}**

**}**

## ExtKMP

void ExtendedKMP(char \*a, char \*b, int M, int N, int \*Next, int \*ret) {// a -> 模式串 b -> 匹配串  
    int i, j, k;  
    for (j = 0; 1 + j < M && a[j] == a[1 + j]; j++);  
    Next[1] = j;  
    k = 1;  
    for (i = 2; i < M; i++) {  
        int Len = k + Next[k], L = Next[i - k];  
        if (L < Len - i) {  
            Next[i] = L;  
        } else {  
            for (j = max(0, Len - i); i + j < M && a[j] == a[i + j]; j++);  
            Next[i] = j;  
            k = i;  
        }  
    }  
    for (j = 0; j < N && j < M && a[j] == b[j]; j++);  
    ret[0] = j;  
    k = 0;  
    for (i = 1; i < N; i++) {  
        int Len = k + ret[k], L = Next[i - k];  
        if (L < Len - i) {  
            ret[i] = L;  
        } else {  
            for (j = max(0, Len - i); j < M && i + j < N && a[j] == b[i + j]; j++);  
            ret[i] = j;  
            k = i;  
        }  
    }  
}

## SA

int n,a[20010],sa[20010],rank[20010],height[20010];

void build()

{

a[n+1]=-1;

void sort(int \*);

int count(int \*,int \*);

int b[20010],c[20010];

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

c[i]=a[i],b[i]=-1,sa[i]=i;

sort(c),count(c,b);

for (int k=1;;k<<=1)

{

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

c[i]=rank[i],b[i]=i+k<=n?rank[i+k]:0;

sort(b),sort(c);

if (count(c,b)>=n)

break;

}

k=0;

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

{

k=k?k-1:0;

if (rank[i]==1)

{

height[rank[i]]=0;

continue;

}

int p=sa[rank[i]-1],q=sa[rank[i]];

while (a[p+k]==a[q+k])

k++;

height[rank[i]]=k;

}

}

void sort(int \*a)

{

int f[20010],x[20010],t=0;

memset(f,0,sizeof(f));

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

f[a[i]]++,t=max(t,a[i]);

for (int i=1;i<=t;i++)

f[i]+=f[i-1];

for (int i=n;i>=1;i--)

x[f[a[sa[i]]]--]=sa[i];

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

sa[i]=x[i];

}

int count(int \*a,int \*b)

{

rank[sa[1]]=1;

int t=1;

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

{

if (a[sa[i]]!=a[sa[i-1]] || b[sa[i]]!=b[sa[i-1]])

t++;

rank[sa[i]]=t;

}

return(t);

}

## 最大团搜索算法

Int g[][]为图的邻接矩阵。 MC(V)表示点集V的最大团

令Si={vi, vi+**1**, ..., vn}, mc[i]表示MC(Si). 倒着算mc[i]，那么显然MC(V)=mc[**1**]

此外有mc[i]=mc[i+**1**] **or** mc[i]=mc[i+**1**]+**1**

**void** init(){

**int** i, j;**for** (i=**1**; i<=n; ++i) **for** (j=**1**; j<=n; ++j) scanf(**"%d"**, &g[i][j]);

}

**void** dfs(**int** size){

**int** i, j, k;

**if** (len[size]==**0**) { **if** (size>ans) { ans=size; found=**true**;} **return**;}

**for** (k=**0**; k<len[size] && !found; ++k) {

**if** (size+len[size]-k<=ans) **break**;

i=list[size][k]; **if** (size+mc[i]<=ans) **break**;

**for** (j=k+**1**, len[size+**1**]=**0**; j<len[size]; ++j)

**if** (g[i][list[size][j]]) list[size+**1**][len[size+**1**]++]=list[size][j];

dfs(size+**1**);}}

**void** work(){

**int** i, j; mc[n]=ans=**1**;

**for** (i=n-**1**; i; --i) { found=**false**; len[**1**]=**0**;

**for** (j=i+**1**; j<=n; ++j) **if** (g[i][j]) list[**1**][len[**1**]++]=j;

dfs(**1**); mc[i]=ans;}}

## 极大团的计数

Bool g[][] 为图的邻接矩阵，图点的标号由1至n。

**void** dfs(**int** size){

**int** i, j, k, t, cnt, best = **0**; **bool** bb;

**if** (ne[size]==ce[size]){**if** (ce[size]==**0**) ++ans;**return**;}

**for** (t=**0**, i=**1**; i<=ne[size]; ++i) {

**for** (cnt=**0**, j=ne[size]+**1**; j<=ce[size]; ++j)

**if** (!g[list[size][i]][list[size][j]]) ++cnt;

**if** (t==**0** || cnt<best) t=i, best=cnt;

}

**if** (t && best<=**0**) **return**;

**for** (k=ne[size]+**1**; k<=ce[size]; ++k) {

**if** (t>**0**){

**for** (i=k; i<=ce[size]; ++i)

**if** (!g[list[size][t]][list[size][i]]) **break**;

swap(list[size][k], list[size][i]);

}

i=list[size][k]; ne[size+**1**]=ce[size+**1**]=**0**;

**for** (j=**1**; j<k; ++j)**if** (g[i][list[size][j]])

list[size+**1**][++ne[size+**1**]]=list[size][j];

**for** (ce[size+**1**]=ne[size+**1**], j=k+**1**; j<=ce[size]; ++j)

**if** (g[i][list[size][j]]) list[size+**1**][++ce[size+**1**]]=list[size][j];

dfs(size+**1**); ++ne[size]; --best;

**for** (j=k+**1**, cnt=**0**; j<=ce[size]; ++j) **if** (!g[i][list[size][j]]) ++cnt;

**if** (t==**0** || cnt<best) t=k, best=cnt;

**if** (t && best<=**0**) **break**;

}}

**void** work(){

**int** i; ne[**0**]=**0**; ce[**0**]=**0**; **for** (i=**1**; i<=n; ++i) list[**0**][++ce[**0**]]=i;

ans=**0**; dfs(**0**);}

## FFT

**void** FFT**(**Complex P**[],** **int** n**,** **int** oper**)**

**{**

**for** **(int** i **=** 1**,** j **=** 0**;** i **<** n **-** 1**;** i**++)** **{**

**for** **(int** s **=** n**;** j **^=** s **>>=** 1**,** **~**j **&** s**;);**

**if** **(**i **<** j**)** **{**

swap**(**P**[**i**],** P**[**j**]);**

**}**

**}**

Complex unit\_p0**;**

**for** **(int** d **=** 0**;** **(**1 **<<** d**)** **<** n**;** d**++)** **{**

**int** m **=** 1 **<<** d**,** m2 **=** m **\*** 2**;**

**double** p0 **=** pi **/** m **\*** oper**;**

sincos**(**p0**,** **&**unit\_p0**.**y**,** **&**unit\_p0**.**x**);**

**for** **(int** i **=** 0**;** i **<** n**;** i **+=** m2**)** **{**

Complex unit **=** 1**;**

**for** **(int** j **=** 0**;** j **<** m**;** j**++)** **{**

Complex **&**P1 **=** P**[**i **+** j **+** m**],** **&**P2 **=** P**[**i **+** j**];**

Complex t **=** unit **\*** P1**;**

P1 **=** P2 **-** t**;**

P2 **=** P2 **+** t**;**

unit **=** unit **\*** unit\_p0**;**

**}**

**}**

**}**

**}**

## Simpson

**double** simpson(**const** T&f,**double** a,**double** b,**int** n){

**const** **double** h=(b-a)/n; **double** ans=f(a)+f(b);

**for**(**int** i=**1**;i<n;i+=**2**)ans+=**4**\*f(a+i\*h);

**for**(**int** i=**2**;i<n;i+=**2**)ans+=**2**\*f(a+i\*h);

**return** ans\*h/**3**;

}printf(**"%lf\n"**,simpson(test,**0**,**1**,(**int**)**1e6**)

## 字符串的最小表示

A[1..n]; A[n+1..n+n]=A[1..n]; i:=**1**; j:=**2**; k:=**0**; t:=**0**;

while (j<=n) { k=**0**; **while** (a[i+k]=a[j+k]) k++;

**if** (a[i+k]>a[j+k]) i=i+k+**1;** **else** j=j+k+**1**;

**if** (i==j) j++; **if** (i>j) swap(i,j);

} printf(“%d\n”,i);

## 二次剩余

/\*a\*x^2+b\*x+c==0 (mod P) 求0..P-1的根 \*/

**int** pDiv2,P,a,b,c,Pb,d;

**inline** **int** calc(**int** x,**int** Time){

**if** (!Time) **return** ***1***; **int** tmp=calc(x,Time/***2***);

tmp=(**long** **long**)tmp\*tmp%P;

**if** (Time&***1***) tmp=(**long** **long**)tmp\*x%P; **return** tmp;

}

**inline** **int** rev(**int** x){ **if** (!x) **return** ***0***; **return** calc(x,P-***2***);}

**inline** **void** Compute(){

**while** (***1***) { b=rand()%(P-***2***)+***2***; **if** (calc(b,pDiv2)+***1***==P) **return**; }

}

**int** main(){

srand(time(***0***)^***312314***); **int** T;

**for** (scanf(***"%d"***,&T);T;--T) {

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

**if** (P==***2***) {

**int** cnt=***0***; **for** (**int** i=***0***;i<***2***;++i) **if** ((a\*i\*i+b\*i+c)%P==***0***) ++cnt;

printf(***"%d"***,cnt);

**for** (**int** i=***0***;i<***2***;++i) **if** ((a\*i\*i+b\*i+c)%P==***0***) printf(***" %d"***,i);

puts(***""***);

}**else** {

**int** delta=(**long** **long**)b\*rev(a)\*rev(***2***)%P;

a=(**long** **long**)c\*rev(a)%P-sqr( (**long** **long**)delta )%P;

a%=P;a+=P;a%=P; a=P-a;a%=P; pDiv2=P/***2***;

**if** (calc(a,pDiv2)+***1***==P) puts(***"0"***);

**else** {

**int** t=***0***,h=pDiv2; **while** (!(h%***2***)) ++t,h/=***2***;

**int** root=calc(a,h/***2***);

**if** (t>***0***) { Compute(); Pb=calc(b,h); }

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

d=(**long** **long**)root\*root\*a%P;

**for** (**int** j=***1***;j<=t-i;++j) d=(**long** **long**)d\*d%P;

**if** (d+***1***==P) root=(**long** **long**)root\*Pb%P;

Pb=(**long** **long**)Pb\*Pb%P;

}

root=(**long** **long**)a\*root%P;

**int** root1=P-root; root-=delta;

root%=P; **if** (root<***0***) root+=P;

root1-=delta; root1%=P; **if** (root1<***0***) root1+=P;

**if** (root>root1) { t=root;root=root1;root1=t; }

**if** (root==root1) printf(***"1 %d\n"***,root);

**else** printf(***"2 %d %d\n"***,root,root1);

}}}**return** ***0***; }

## Pell方程求解

//求x^2-ny^2=1的最小正整数根,n不是完全平方数

p[**1**]=**1**;p[**0**]=**0**; q[**1**]=**0**;q[**0**]=**1**; a[**2**]=(**int**)(floor(sqrt(n)+**1e**-**7**));

g[**1**]=**0**;h[**1**]=**1**;

**for** (**int** i=**2**;i;++i) {

g[i]=-g[i-**1**]+a[i]\*h[i-**1**]; h[i]=(n-sqr(g[i]))/h[i-**1**];

a[i+**1**]=(g[i]+a[**2**])/h[i]; p[i]=a[i]\*p[i-**1**]+p[i-**2**];

q[i]=a[i]\*q[i-**1**]+q[i-**2**]; 检查p[i],q[i]是否为解，如果是，则退出

}

## Exact Cover

void del(int x)

{

a[a[x].l].r=a[x].r;

a[a[x].r].l=a[x].l;

for (int i=a[x].d;i!=x;i=a[i].d)

for (int j=a[i].r;j!=i;j=a[j].r)

{

sum[a[j].y]--;

a[a[j].u].d=a[j].d;

a[a[j].d].u=a[j].u;

}

}

void renew(int x)

{

a[a[x].l].r=x;

a[a[x].r].l=x;

for (int i=a[x].u;i!=x;i=a[i].u)

for (int j=a[i].l;j!=i;j=a[j].l)

{

sum[a[j].y]++;

a[a[j].u].d=j;

a[a[j].d].u=j;

}

}

bool search()

{

if (a[0].r==0)

return(true);

int k,min=20000000;

for (int i=a[0].r;i!=0;i=a[i].r)

if (sum[i]<min)

min=sum[k=i];

del(k);

for (int i=a[k].d;i!=k;i=a[i].d)

{

for (int j=a[i].r;j!=i;j=a[j].r)

del(a[j].y);

if (search())

return(true);

for (int j=a[i].l;j!=i;j=a[j].l)

renew(a[j].y);

}

renew(k);

return(false);

}

void del(int x)

{

for (int i=a[x].d;i!=x;i=a[i].d)

{

sum[a[i].y]--;

a[a[i].l].r=a[i].r;

a[a[i].r].l=a[i].l;

}

}

void renew(int x)

{

for (int i=a[x].u;i!=x;i=a[i].u)

{

sum[a[i].y]++;

a[a[i].l].r=i;

a[a[i].r].l=i;

}

}

bool vis[60];

int best()

{

memset(vis,0,sizeof(vis));

int ans=0;

for (int i=a[0].r;i!=0;i=a[i].r)

{

if (vis[i])

continue;

ans++;

for (int j=a[i].d;j!=i;j=a[j].d)

for (int k=a[j].r;k!=j;k=a[k].r)

vis[a[k].y]=true;

}

return(ans);

}

bool DLX(int dep)

{

if (a[0].r==0)

return(true);

if (dep+best()>m)

return(false);

int k,mi=1<<30;

for (int i=a[0].r;i!=0;i=a[i].r)

if (sum[i]<mi)

mi=sum[k=i];

for (int i=a[k].d;i!=k;i=a[i].d)

{

del(i);

for (int j=a[i].r;j!=i;j=a[j].r)

del(j);

if (DLX(dep+1))

return(true);

for (int j=a[i].l;j!=i;j=a[j].l)

renew(j);

renew(i);

}

return(false);

}

## Link-Cut-Tree

**void** rotate**(int** x**)** **{**

**int** t **=** type**[**x**];**

**int** y **=** parent**[**x**];**

**int** z **=** children**[**x**][**1 **^** t**];**

type**[**x**]** **=** type**[**y**];**

parent**[**x**]** **=** parent**[**y**];**

**if** **(**type**[**x**]** **!=** 2**)** **{**

children**[**parent**[**x**]][**type**[**x**]]** **=** x**;**

**}**

type**[**y**]** **=** 1 **^** t**;**

parent**[**y**]** **=** x**;**

children**[**x**][**1 **^** t**]** **=** y**;**

**if** **(**z **!=** 0**)** **{**

type**[**z**]** **=** t**;**

parent**[**z**]** **=** y**;**

**}**

children**[**y**][**t**]** **=** z**;**

update**(**y**);**

**}**

**void** splay**(int** x**)** **{**

vector **<int>** stack**(**1**,** x**);**

**for** **(int** i **=** x**;** type**[**i**]** **!=** 2**;** i **=** parent**[**i**])** **{**

stack**.**push\_back**(**parent**[**i**]);**

**}**

**while** **(!**stack**.**empty**())** **{**

push**(**stack**.**back**());**

stack**.**pop\_back**();**

**}**

**while** **(**type**[**x**]** **!=** 2**)** **{**

**int** y **=** parent**[**x**];**

**if** **(**type**[**x**]** **==** type**[**y**])** **{**

rotate**(**y**);**

**}** **else** **{**

rotate**(**x**);**

**}**

**if** **(**type**[**x**]** **==** 2**)** **{**

**break;**

**}**

rotate**(**x**);**

**}**

update**(**x**);**

**}**

**void** access**(int** x**)** **{**

**int** z **=** 0**;**

**while** **(**x **!=** 0**)** **{**

splay**(**x**);**

type**[**children**[**x**][**1**]]** **=** 2**;**

children**[**x**][**1**]** **=** z**;**

type**[**z**]** **=** 1**;**

update**(**x**);**

z **=** x**;**

x **=** parent**[**x**];**

**}**

**}**

## 后缀自动机

**struct** State **{**

**static** vector **<**State**\*>** states**;**

**int** id**,** length**;**

State **\***parent**;**

State**\*** go**[**C**];**

State**(int** length**)** **:** id**((int)**states**.**size**()),** length**(**length**),** parent**(NULL)** **{**

memset**(**go**,** **NULL,** **sizeof(**go**));**

states**.**push\_back**(this);**

**}**

State**\*** extend**(**State**\*** start**,** **int** token**)** **{**

State **\***p **=** **this;**

State **\***np **=** **new** State**(**length **+** 1**);**

**while** **(**p **&&** **!**p**->**go**[**token**])** **{**

p**->**go**[**token**]** **=** np**;**

p **=** p**->**parent**;**

**}**

**if** **(!**p**)** **{**

np**->**parent **=** start**;**

**}** **else** **{**

State **\***q **=** p**->**go**[**token**];**

**if** **(**p**->**length **+** 1 **==** q**->**length**)** **{**

np**->**parent **=** q**;**

**}** **else** **{**

State **\***nq **=** **new** State**(**p**->**length **+** 1**);**

memcpy**(**nq**->**go**,** q**->**go**,** **sizeof(**q**->**go**));**

nq**->**parent **=** q**->**parent**;**

np**->**parent **=** q**->**parent **=** nq**;**

**while** **(**p **&&** p**->**go**[**token**]** **==** q**)** **{**

p**->**go**[**token**]** **=** nq**;**

p **=** p**->**parent**;**

**}**

**}**

**}**

**return** np**;**

**}**

**};**

## 差分序列

F(n) = c0 \* C(n, 0) + c1 \* C(n, 1) + ... + cp \* C(n, p)

S(n) = F(0) + F(1) + ... + F(n)

= c0 \* C(n + 1, 1) + c1 \* (n + 1, 2) + ... + cp \* C(n + 1, p + 1)

## 弦图的完美消除序列

从n到1的顺序依次给点标号（标号为i的点出现在完美消除序列的第i个）

设lable[i]表示第i个点与多少已标号的点相邻，每次选择label[i]最大的未标号点进行标号。

任取一个已标号的与当前新标号的点相邻的点，如果与其他的已标号的且与当前点相邻的点之间没有边，则无解。

1.团数 ≤ 色数

2.最大独立集数 ≤ 最小团覆盖数

3.任何一个弦图都至少有一个单纯点，不是完全图的弦图至少有两个不相邻的单纯点。

4.设第i个点在弦图的完美消除序列第p(i)个。令N(v) = {w | w与v相邻且p(w) > p(v)}弦图的极大团一定是v∪N(v)的形式。

5.弦图最多有n个极大团。

6.设next(v) 表示N(v)中最前的点。令w\*表示所有满足A∈B的w中最后的一个点。判断v∪N(v)是否为极大团,只需判断是否存在一个w，满足Next(w) = v且|N(v)| + 1 ≤ |N(w)|即可。

7.最小染色：完美消除序列从后往前依次给每个点染色，给每个点染上可以染的最小的颜色。//团数=色数

8.最大独立集：完美消除序列从前往后能选就选。

9.最小团覆盖：设最大独立集为{p1 , p2 , …, pt}，则{p1∪N(p1), …, pt∪N(pt)}为最小团覆盖。 //最大独立集数 = 最小团覆盖数!!!

## 双人零和矩阵游戏（公式）

N\*N的方阵A，选行的玩家的最优策略是p，选列的是q,则

q = A逆 \* e / (e转置 \* A逆 \*ｅ)

　　 p转置 = e转置 \* A逆 / (e转置 \* A逆 \*ｅ) e是全为1的列向量

当A不可逆时，每个元素加上一个值就可以了。

当矩阵是m行,n列的时候：

P[**1**]+P[**2**]+……+P[m]=**1**; P[i]>=**0**

V<=sigma(P[i]\*Matrix[i][j])

最大化V

## 质数测试

**bool** primeTest**(**LL n**,** LL b**)** **{**

LL m **=** n **-** 1**;**

LL counter **=** 0**;**

**while** **((**m **&** 1**)** **==** 0**)** **{**

m **>>=** 1**;**

counter **++;**

**}**

LL ret **=** powMod**(**b**,** m**,** n**);**

**if** **(**ret **==** 1 **||** ret **==** n **-** 1**)** **{**

**return** **true;**

**}**

counter **--;**

**while** **(**counter **>=** 0**)** **{**

ret **=** multiplyMod**(**ret**,** ret**,** n**);**

**if** **(**ret **==** n **-** 1**)** **{**

**return** **true;**

**}**

counter **--;**

**}**

**return** **false;**

**}**

**const** **int** BASIC**[**12**]** **=** **{**2**,** 3**,** 5**,** 7**,** 11**,** 13**,** 17**,** 19**,** 23**,** 29**,** 31**,** 37**};**

**bool** isPrime**(**LL n**)** **{**

**if** **(**n **<** 2**)** **return** **false;**

**if** **(**n **<** 4**)** **return** **true;**

**if** **(**n **==** 3215031751LL**) return** **false;**

**for** **(int** i **=** 0**;** i **<** 12 **&&** BASIC**[**i**]** **<** n**;** **++** i**)**

**if** **(!**primeTest**(**n**,** BASIC**[**i**]))** **return** **false;**

**return** **true;**

**}**

## Pollard-Rho

LL pollardRho**(**LL n**,** LL seed**)** **{**

LL x**,** y**;**

x **=** y **=** rand**()** **%** **(**n **-** 1**)** **+** 1**;**

LL head **=** 1**，**tail **=** 2**;**

**while** **(true)** **{**

x **=** multiplyMod**(**x**,** x**,** n**);**

x **=** addMod**(**x**,** seed**,** n**);**

**if** **(**x **==** y**)** **return** n**;**

LL d **=** gcd**(**abs**(**x **-** y**),** n**);**

**if** **(**1 **<** d **&&** d **<** n**)** **return** d**;**

head **++;**

**if** **(**head **==** tail**)** **{**

y **=** x**;**

tail **<<=** 1**;**

**}**

**}**

**}**

vector **<**LL**>** divisors**;**

**void** factorize**(**LL n**)** **{**

**if** **(**n **>** 1**)** **{**

**if** **(**isPrime**(**n**))** **{**

divisors**.**push\_back**(**n**);**

**}** **else** **{**

LL d **=** n**;**

**while** **(**d **>=** n**)** **{**

d **=** pollardRho**(**n**,** rand**()** **%** **(**n **-** 1**)** **+** 1**);**

**}**

factorize**(**n **/** d**);** factorize**(**d**);**

**}**

**}**

**}**

## 直线下有多少个格点

求

LL count**(**LL n**,** LL a**,** LL b**,** LL m**)** **{**

**if** **(**b**==**0**)** **return** n **\*** **(**a **/** m**);**

**if** **(**a**>=**m**)** **return** n **\*** **(**a **/** m**)** **+** count**(**n**,** a **%** m**,** b**,** m**);**

**if** **(**b**>=**m**)** **return** **(**n **-** 1**)** **\*** n **/** 2 **\*** **(**b **/** m**)** **+** count**(**n**,** a**,** b **%** m**,** m**);**

**return** count**((**a **+** b **\*** n**)** **/** m**,** **(**a **+** b **\*** n**)** **%** m**,** m**,** b**);**

**}**

## 综合

设正整数n的质因数分解为n = ∏pi^ai,则x^2+y^2=n有整数解的充要条件是n中不存在形如pi≡3(mod 4) &(and) 指数ai为奇数的质因数pi

Pick定理：简单多边形，不自交。(严格在多边形内部的整点数\*2 +在边上的整点数– 2)/2 =面积

有上下界网络流，可行流增广的流量不是实际流量。若要求实际流量应该强算一遍源点出去的流量。

求最小下届网络流： 方法一：加t-s的无穷大流，求可行流，然后把边反向后（减去下届网络流），在残留网络中从汇到源做最大流。

方法二：在求可行流的时候，不加从汇到源的无穷大边，得到最大流X， 加上从汇到源无穷大边后，再求最大流得到Y。那么Y即是答案最小下届网络流。

二次剩余：p为奇素数，若(a,p)=1，a为p的二次剩余必要充分条件为a^((p-1)/2) mod p=1.

(否则为p-1)

p为奇素数，x^b = a(mod p),x为p的b次剩余的必要充分条件为 若x^（(p-1)/ (p-1 和 b的最大公约数)） mod p=1.

## java\_scl

**public** **static** BigInteger getsqrt(BigInteger n){

**if** (n.compareTo(BigInteger.ZERO)<=**0**) **return** n;

BigInteger x,xx,txx; xx=x=BigInteger.ZERO;

**for** (**int** t=n.bitLength()/**2**;t>=**0**;t--){

txx=xx.add(x.shiftLeft(t+**1**)).add(BigInteger.ONE.shiftLeft(t+t));

**if** (txx.compareTo(n)<=**0**){

x=x.add(BigInteger.ONE.shiftLeft(t)); xx=txx;

}}**return** x;

}

## 基本形 公式

**椭圆：**

椭圆，其中离心率焦点参数

椭圆上(x,y)点处的曲率半径为 ,其中分别为(x,y)与两焦点的距离。设点A和点M的坐标分别为(a,0)和(x,y)，则AM的弧长为

椭圆的周长为 ，其中

设椭圆上点M(x,y),N(x,-y),x,y>0,A(a,0),原点O(0,0)。

扇形OAM的面积 弓形MAN的面积

方程，5个点确定一个圆锥曲线。

为(x,y)点关于椭圆中心的极角，r为(x,y)到椭圆中心的距离，椭圆极坐标方程:

**抛物线**

标准方程 曲率半径

弧长：设M(x,y)是抛物线上一点，则]

弓形面积：设M，D是抛物线上两点，且分居一、四象限。作一条平行于MD且与抛物线相切的直线L。若M到L的距离为h。则有

**重心**

半径为r、圆心角为的扇形的重心与圆心的距离为

半径为r、圆心角为的圆弧的重心与圆心的距离为

椭圆上半部分的重心与圆心的距离为

抛物线中弓形MOD的重心满足 , P是直线L与抛物线的切点，Q在MD上且PQ平行x轴。C是重心。

**内心**

**三重积公式**

**额外的公式**

***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

**算法**

## 树的计数

**有根树的计数**

令

于是，n+1个结点的有根树的总数为

附：

**无根树的计数**

当n是奇数时，则有 种不同的无根树。

当n是偶数时，则有这么多种不同的无根树。

## 代数

**Burnside引理**

**三次方程求根公式**

其中 j=0,1,2,

当求解时， 令 再求解y，即转化成的形式

**组合公式**

错排：

## 三角公式

## 积分表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | | | | |  |
|  | |  | | | | | |  |
|  | |  | | | | | |  |
|  | |  | | | | | |  |
|  | |  | | | | | |  |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | |  | | |
|  | | | | | | | | |
|  | | | |  | | | | |
|  | | | |  | | | | |
|  | | | |  | | | | |
|  | | | | | | | | |
|  | | | |  | | | | |
|  | | | |  | | | | |
|  | | | |  | | | | |
|  | | | | | | | | |
|  | | | | |  | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | |  |
|  | | | | | | | |
|  |  | | | | | | |  |
|  |  | | | | | | |  |
|  | | |  | | | | | |
|  | |  | | | | |  | |
|  | | |  | | | | | |
|  | | |  | | | | | |
|  | | |  | | | | | |