目录

[计算几何 4](#_Toc305828884)

[计算几何\_main 4](#_Toc305828885)

[多边形和圆相交的面积 10](#_Toc305828886)

[半平面交n^2 12](#_Toc305828887)

[三维几何操作合并 13](#_Toc305828888)

[三维几何 15](#_Toc305828889)

[三维旋转操作 19](#_Toc305828890)

[三维凸包\_随机增量 19](#_Toc305828891)

[三维凸包求重心 23](#_Toc305828892)

[随机增量最小覆盖圆 28](#_Toc305828893)

[圆面积模板（新） 29](#_Toc305828894)

[圆的面积并（可以求交） 30](#_Toc305828895)

[圆的面积模板(n^2logn) 33](#_Toc305828896)

[直线和凸包交点（返回最近和最远点） 38](#_Toc305828897)

[最小覆盖球 39](#_Toc305828898)

[图论 41](#_Toc305828899)

[KM 41](#_Toc305828900)

[求最小上下界网络流 44](#_Toc305828901)

[求最小下界网络流\_反边（optional） 45](#_Toc305828902)

[无向图最小割 47](#_Toc305828903)

[Voronoi 47](#_Toc305828904)

[KD-TREE 55](#_Toc305828905)

[弦图的完美消除序列 57](#_Toc305828906)

[一般图最大匹配\_片段 60](#_Toc305828907)

[最小树形图(ElogE+V^2) 63](#_Toc305828908)

[最小树形图（V^3） 65](#_Toc305828909)

[Hopcroft 67](#_Toc305828910)

[割点缩块 69](#_Toc305828911)

[割边缩块 71](#_Toc305828912)

[字符串 73](#_Toc305828913)

[字符串最小表示 73](#_Toc305828914)

[Manacher-O(n)求每个位置为中心的最长回文串 73](#_Toc305828915)

[多个串求最长连续的子串 —— 后缀数组 o(n) 73](#_Toc305828916)

[扩展\_KMp 75](#_Toc305828917)

[最多的重复字串（optional） 76](#_Toc305828918)

[后缀自动机 80](#_Toc305828919)

[dc3 83](#_Toc305828920)

[杂 85](#_Toc305828921)

[最大团搜索算法 85](#_Toc305828922)

[极大团的计数 86](#_Toc305828923)

[Farmland 87](#_Toc305828924)

[FFT（crazyb0y） 89](#_Toc305828925)

[fft——速度一般 90](#_Toc305828926)

[FFt\_speed 91](#_Toc305828927)

[Romberg 94](#_Toc305828928)

[多项式求根（求导二分）（optional） 95](#_Toc305828929)

[强连通分量（一遍dfs) 97](#_Toc305828930)

[求区间第K大数\_不改变值的 98](#_Toc305828931)

[任意两点间的第K短路，可重复走 101](#_Toc305828932)

[长方体表面两点最短距离 103](#_Toc305828933)

[字符串的最小表示(正确的zy) 104](#_Toc305828934)

[最长公共子序列 104](#_Toc305828935)

[Splay-Tree（带split） 107](#_Toc305828936)

[动态树(ftiasch) 110](#_Toc305828937)

[动态树 113](#_Toc305828938)

[曼哈顿最小生成树 117](#_Toc305828939)

[表达式的计算 121](#_Toc305828940)

[二维树状数组 121](#_Toc305828941)

[双人零和矩阵游戏 122](#_Toc305828942)

[Exact Cover(crazyb0y) 124](#_Toc305828943)

[数独 Dancing Links 127](#_Toc305828944)

[线性规划 132](#_Toc305828945)

[线性规划单纯形法\_武汉网络赛5题 134](#_Toc305828946)

[高精度 137](#_Toc305828947)

[高精度开根号 137](#_Toc305828948)

[高精度类 139](#_Toc305828949)

[数学 144](#_Toc305828950)

[牛顿迭代开根号 144](#_Toc305828951)

[有多少个点在多边形内 145](#_Toc305828952)

[斜线下格点统计 145](#_Toc305828953)

[大整数相乘取摸 146](#_Toc305828954)

[素数判定 146](#_Toc305828955)

[Pollard-Rho 147](#_Toc305828956)

[O(p)求1..p-1的逆元 148](#_Toc305828957)

[广义离散对数（不需要互质） 148](#_Toc305828958)

[n次剩余 149](#_Toc305828959)

[求ax取模n同余b的所有解及中国剩余定理 154](#_Toc305828960)

[Pell方程求解 155](#_Toc305828961)

[莫比乌斯函数以及gcd=1的对数 156](#_Toc305828962)

[二次剩余 158](#_Toc305828963)

[Tips 160](#_Toc305828964)

[差分序列 160](#_Toc305828965)

[牛顿迭代 160](#_Toc305828966)

[求某年某月某日是星期几 160](#_Toc305828967)

[图同构hash 160](#_Toc305828968)

[综合 161](#_Toc305828969)

[Stl使用 164](#_Toc305828970)

[java\_scl 164](#_Toc305828971)

[cpp 166](#_Toc305828972)

[积分表 169](#_Toc305828973)

[基本形 公式 169](#_Toc305828974)

[平方剩余求解 170](#_Toc305828975)

[树的计数 171](#_Toc305828976)

[代数 171](#_Toc305828977)

[三角公式 172](#_Toc305828978)

[积分表 172](#_Toc305828979)

# 计算几何

## 计算几何\_main

#include <iostream>

#include <cstdio>

#include <vector>

#include <string>

#include <cstring>

#include <cmath>

#include <algorithm>

**using** **namespace** std;

#define rep(i,n) for(i=0;i<(n);i++)

#define foru(i,a,b) for(i=(a);i<=(b);i++)

#define ford(i,a,b) for(i=(a);i>=(b);i--)

**double** eps = **1e**-**8**;

**struct** line{ **double** a,b,c; };

**int** cmp(**double** x){

**if** (x>eps) **return** **1**;

**if** (x<-eps) **return** -**1**;

**return** **0**;

}

**class** point{

**public**:

**double** x,y;

point(){}

point(**double** x,**double** y) : x(x) , y(y) {}

**void** input(){scanf(**"%lf %lf"**,&x,&y);}

point **operator** -(point a){a.x=x-a.x;a.y=y-a.y;**return** a;}

point **operator** +(point a){a.x=x+a.x;a.y=y+a.y;**return** a;}

point **operator** /(**double** a){ **return** point(x/a,y/a);}

**bool** **operator** == (**const** point &b) {**return** !cmp(x - b.x) && !cmp(y - b.y);}

};

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

**return** (b.x-a.x)\*(c.y-a.y) - (b.y-a.y)\*(c.x-a.x);

}

**double** dot(point a, point b,point c){

**return** (b-a) ^ (c-a);

}

**double** dis(point a){**return** sqrt(a.x\*a.x+a.y\*a.y);}

**double** dis(point a,point b){**return** dis(b-a);}

//================两点求线

line point\_make\_line(point a, point b){

line h;

h.a=b.y-a.y;

h.b=-(b.x-a.x);

h.c=-a.x\*b.y + a.y\*b.x;

**return** h;

}

//===========旋转角度p的向量

point rotate\_point(point a, **double** p){

point h;

h.x= a.x\*cos(p) - a.y\*sin(p);

h.y= a.x\*sin(p) + a.y\*cos(p);

**return** h;

}

//================点P到线段st的距离 ======================

**double** dis\_point\_segment(point p,point s,point t){

**if** (((p-t)^(s-t))>**0**&& (((p-s)^(t-s))>**0**)) **return** fabs((p-s)\*(t-s))/dis(s-t);

**else** **return** min(dis(p-s),dis(p-t));

}

//========一个点关于直线作镜像

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

**double** t = dot( p1 , p2 , p0 )/ dot( p1 , p2 , p2 ) ;

cp.x=p1.x + t\*(p2.x-p1.x);

cp.y=p1.y + t\*(p2.y-p1.y);

}

//===!! 或者 ===

**double** PointToLine (**const** point &p0,**const** point &p1,**const** point &p2,point &cp)

{

**double** d=dis(p1,p2);

**double** s=area(p1,p2,p0)/d;

cp.x=p0.x+s\*(p2.y-p1.y)/d;

cp.y=p0.y-s\*(p2.x-p1.x)/d;

**return** s;

}

**void** ReflectPoint (**const** point &p0,**const** point &p1,**const** point &p2,point &cp)

{

point p3;

PointToLine(p0,p1,p2,p3); //PointProjLine(p0,p1,p2,p3); 都是求影射点

cp=p3+(p3-p0);

}

//=========判点是否在线段上

**bool** PointOnSegment (point p , point s , point t ){

**return** cmp(area(p,s,t))==**0** && cmp(dot(p,s,t))<=**0**;

}

//================两线交点

point line\_make\_point(line a, line b){

point h;

h.y=-(a.c\*b.a - b.c\*a.a) / (a.b\*b.a - b.b\*a.a); //=====makesure a and b aren't parallel

**if** (abs(a.a)<eps) h.x=(-b.c-h.y\*b.b)/b.a;

**else** h.x=(-a.c-h.y\*a.b)/a.a;

**return** h;

}

//==========线段平移 D 的长度

line move\_d(line a,**const** **double** d){

**return** (line){a.a,a.b,a.c+d\*sqrt(a.a\*a.a+a.b\*a.b)};

}

//==========判平行

**bool** parallel(line a,line b){

**if** (cmp(a.b\*b.a - b.b\*a.a)==**0**) **return** **true**;

**return** **false**;

}

//========================点与多边形 线段与多边形=================

**int** PointInPolygon(point cp, point a[], **int** n){

**int** i , k , d1 , d2 ,wn=**0**;

a[n]=a[**0**];

**rep**(i,n){

**if** ( PointOnSegment ( cp,a[i],a[i+**1**] ) ) **return** **2** ;

k = cmp ( area (a [ i ] , a [ i + **1** ] , cp ) ) ;

d1 = cmp ( a [ i +**0**] . y - cp . y ) ;

d2 = cmp ( a [ i +**1**] . y - cp . y ) ;

**if** (k>**0** && d1<=**0** && d2>**0**) wn++;

**if** (k<**0** && d2<=**0** && d1>**0**) wn--;

}

**return** wn!=**0**;

}

//========================================判断线段是否有在多边形内部=========

**bool** compareab(**const** point &a, **const** point &b){

**if** (a.x<b.x || (a.x==b.x && a.y<b.y)) **return** **true**;

**else** **return** **false**;

}

point stack[**11000**];

**bool** SegmentCrossPolygon(point s, point t , point a[],**int** n){

**int** i,j,k,m1,m2,closed;

line e1,e2;

point cross;

**if** (PointInPolygon(s ,a ,n)==**1** || PointInPolygon(t , a ,n)==**1**) **return** **true**;

closed=**1**; stack[closed]=s;

e1=point\_make\_line(s,t);

a[n]=a[**0**];

**rep**(i,n){

k=i+**1**;

e2=point\_make\_line(a[i],a[k]);

**if** (!parallel(e1,e2)){

cross=line\_make\_point(e1,e2);

**if** (PointOnSegment(cross,s,t) && PointOnSegment(cross,a[i],a[k])) {

closed++; stack[closed]=cross;

}

}

}

closed++; stack[closed]=t;

sort(stack+**1**,stack+closed+**1**,compareab);

**foru**(i,**1**,closed-**1**){

cross=(stack[i]+stack[i+**1**])/**2**;

**if** (PointInPolygon(cross , a , n)==**1**)

**return** **true**;

}

**return** **false**;

}

// 多边形的重心

**void** PolygonCentroids (point p [ ] , **int** n ,point &cp ){

// if 面积为0 需要特判

**double** sum=**0** , s =**0**; cp.x=**0**; cp.y=**0**;

**for** ( **int** i =**1**; i<n-**1**; i++,sum+=s ){

s= area( p[**0**] , p[i] , p[i+**1**] ) ;

cp.x += s \*( p[**0**].x + p[i].x + p[i+**1**].x ) ;

cp.y += s \*( p[**0**].y + p[i].y + p[i+**1**].y ) ;

}

cp.x/=sum\***3**; cp.y/=sum\***3** ;

}

point gravity(point \*p, **int** n){

// if 面积为0 需要特判

**double** area = **0**;

point center;

center.x = **0**;

center.y = **0**;

p[n]=p[**0**];

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

area += (p[i].x\*p[i+**1**].y - p[i+**1**].x\*p[i].y)/**2**;

center.x += (p[i].x\*p[i+**1**].y - p[i+**1**].x\*p[i].y) \* (p[i].x + p[i+**1**].x);

center.y += (p[i].x\*p[i+**1**].y - p[i+**1**].x\*p[i].y) \* (p[i].y + p[i+**1**].y);

}

area += (p[n-**1**].x\*p[**0**].y - p[**0**].x\*p[n-**1**].y)/**2**;

center.x += (p[n-**1**].x\*p[**0**].y - p[**0**].x\*p[n-**1**].y) \* (p[n-**1**].x + p[**0**].x);

center.y += (p[n-**1**].x\*p[**0**].y - p[**0**].x\*p[n-**1**].y) \* (p[n-**1**].y + p[**0**].y);

center.x /= **6**\*area;

center.y /= **6**\*area;

**return** center;

}

//===============================圆

**double** angle (point p0 , point p1 , point p2 ){

**double** cr = area(p0,p1,p2);

**double** dt = dot(p0,p1,p2);

**if** (cmp(cr)==**0**) cr=**0.0**;

**if** (cmp(dt)==**0**) dt=**0.0**;

**return** atan2(cr , dt); // -pi~pi

}

**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 INPUT: ( 2 4 2 , 8 9 ) , ( 2 1 2 , 1 8 5 ) , ( 7 1 , 1 2 8 ) , OUTPUT: ( 1 8 9 . 5 2 8 6 , 1 3 7 . 4 9 8 7 )

**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 ) ;

**return** r ;

}

// 三角形 外心 INPUT: ( 2 4 2 , 8 9 ) , ( 2 1 2 , 1 8 5 ) , ( 7 1 , 1 2 8 ) , OUTPUT: ( 2 0 8 . 8 2 2 9 , 1 7 1 . 0 6 9 7 )

**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 ;

}

// 园外一点p0 ,半径为r， 直线ax+by+c=0 的交点

**int** CircleLine(point p0 , **double** r , **double** a , **double** b , **double** c , point &cp1 , point &cp2 ) {

**double** aa = a\*a , bb = b\*b , s = aa + bb ;

**double** d = r\*r\*s - sqr ( a\*p0.x+b\*p0.y+c ) ;

**if** (d+eps<**0**) **return** **0** ;

**if** (d<eps) d=**0**; **else** d=sqrt(d);

**double** ab = a\*b , bd = b\*d , ad = a\*d ;

**double** xx = bb\*p0.x - ab\*p0.y - a\*c ;

**double** yy = aa\*p0.y - ab\*p0.x - b\*c ;

cp2.x = ( xx + bd ) / s ; cp2.y = ( yy - ad ) / s ;

cp1.x = ( xx - bd ) / s ; cp1.y = ( yy + ad ) / s ;

**if**( d>eps ) **return** **2** ; **else** **return** **1** ;

}

// 两园交线 Common Axis of |P - P1| = r1 and |P - P2| = r2 of the ax + by + c = 0 form

**void** CommonAxis (point p1 , **double** r1 , point p2 , **double** r2 , **double** &a , **double** &b , **double** &c ){

**double** sx = p2.x + p1.x , mx = p2.x - p1.x ;

**double** sy = p2.y + p1.y , my = p2.y - p1.y ;

a = **2**\*mx ; b = **2**\*my ; c = -sx\*mx - sy\*my - ( r1+r2 )\*( r1-r2 ) ;

}

// 两园交点 Crossing of |P - P1| = r1 and |P - P2| = r2

// 两个圆不能共圆心，请特判

**int** CircleCrossCircle( point p1 , **double** r1 , point p2 , **double** r2 , point &cp1 , point &cp2 ){

**double** mx = p2.x - p1.x , sx = p2.x+p1.x , mx2 = mx\*mx;

**double** my = p2.y - p1.y , sy = p2.y+p1.y , my2 = my\*my;

**double** sq = mx2 + my2 , d = -( sq - sqr ( r1-r2 ) ) \* ( sq - sqr ( r1+r2 ) ) ;

**if** ( d+eps <**0** ) **return** **0** ; **if** ( d<eps ) d=**0** ; **else** d = sqrt(d ) ;

**double** x = mx\* ( ( r1+r2 )\*( r1-r2 ) + mx\*sx ) + sx\*my2 ;

**double** y = my\* ( ( r1+r2 )\*( r1-r2 ) + my\*sy ) + sy\*mx2 ;

**double** dx = mx\*d , dy = my\*d ; sq \*= **2**;

cp1.x = ( x - dy ) / sq ; cp1.y = ( y + dx ) / sq ;

cp2.x = ( x + dy ) / sq ; cp2.y = ( y - dx ) / sq ;

**if** ( d>eps ) **return** **2** ; **else** **return** **1** ;

}

//====两园面积交 dist = 是距离 **dis是距离的平方**

**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;

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);

**double** ans=**0**;

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

**struct** point {

**double** x, y;

point() {}

point(**double** \_x, **double** \_y): x(\_x), y(\_y) {}

**double** len() {**return** sqrt(x\*x+y\*y);}

**void** output() {printf(**"%.15lf %.15lf\n"**, x, y);}

} a, b, c, o;

**const** **double** eps = **1e**-**8**;

**const** **double** PI = acos(-**1.**);

**double** r;

**inline** **int** sign(**double** x) {

**if** (x < eps) **return** -**1**; **else** **return** (x > eps);

}

point **operator**\*(**double** &a, **const** point &b) {

**return** point(a\*b.x, a\*b.y);

}

**double** dot(**const** point &a, **const** point &b) {

**return** a.x\*b.x + a.y\*b.y;

}

**double** det(**const** point &a, **const** point &b) {

**return** a.x\*b.y - a.y\*b.x;

}

//===========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();

//sinB = abs(det(pb, pb-pa)) / a / c;

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

B = acos(cosB);

//sinC = abs(det(pa, pb)) / a / b;

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

C = acos(cosC);

//printf("area2( %.4lf, %.4lf, %.4lf )\n", a, b, C/PI\*180);

**if** (a > r) {

S = C/**2**\*r\*r;

h = a\*b\*sin(C)/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(sin(B)/r\*a);

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

}

**else** S = **.5**\*sin(C)\*a\*b;

//printf("res = %.4lf\n", S);

**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;

//printf(" oa = "); oa.output();

//printf(" ob = "); ob.output();

//printf(" oc = "); oc.output();

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

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

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

//printf("\*\*\* S = %.4lf\n", abs(S));

**return** abs(S);

}

## 半平面交n^2

**const** **int** maxn=**200**;

**const** **double** eps=**1e**-**8**;

**const** **int** infinite=**10000**;

**struct** point{

**double** x,y;

**void** input(){

scanf(**"%lf%lf"**,&x,&y);

}

} sol[maxn],tmp[maxn];

**struct** Tline{

point a,b;

} line[maxn];

**int** n,m;

**void** rebuild(point a, point b){

**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];

}

**void** work(){

**int** i,j,k;

**double** ans;

point o;

sol[**0**].x = **0**; sol[**0**].y = **0**;

sol[**1**].x = infinite; sol[**1**].y = **0**;

sol[**2**].x = infinite; sol[**2**].y = infinite;

sol[**3**].x = **0**; sol[**3**].y = infinite;

m=**4**;

**rep**(i,n) rebuild(line[i].a,line[i].b); // 保留直线line[i].a,line[i+1].b左边的点

**if** (m>**0**) printf(**"1\n"**);

**else** printf(**"0\n"**);

}

## 三维几何操作合并

**const** **double** pi = **3.1415926535897932384626433832795**;

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

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

**return** a - b > zero;

}

**inline** **double** sqrt\_fix(**double** a)

{

**return** a <= **0** ? **0** : sqrt(a);

}

**inline** **double** sqr(**double** a)

{

**return** a\*a;

}

**struct** Point\_3 {

**double** x, y, z;

Point\_3() {

}

Point\_3(**double** x, **double** y, **double** z) : x(x), y(y), z(z) {

}

**double** Length() **const** {

**return** sqrt\_fix(sqr(x) + sqr(y) + sqr(z));

}

};

**double** a[**4**][**4**];

**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);

r = -r / **180** \* pi;

Rotate(p, r); //===========顺时针旋转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**)**

## 三维旋转操作

/====a点，绕Ob向量，逆时针旋转弧度angle，**如果sin和cos可以不用angle算，最好传进来**

point e1,e2,e3;

point Rotate( point a, point b, **double** angle ){

b.std(); **//std()是单位化，b不可以为(0,0,0)**

e3=b;

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

e1=a - e3\*lens;

**if** (e1.len()>(**1e**-**8**)) e1.std();

**else** **return** a;

e2=e1/e3; **// / 是det(e1,e3)**

**double** x1,y1,x,y;

y1=a\*e1;

x1=a\*e2;

//cout<<x1<<" "<<y1<<endl;

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

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

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

}

## 三维凸包\_随机增量

#define SIZE(X) (int(X.size()))

#define Eps 1E-8

#define PI 3.14159265358979323846264338327950288

**inline** **int** Sign(**double** x) {

**return** x < -Eps ? -**1** : (x > Eps ? **1** : **0**);

}

**inline** **double** Sqrt(**double** x) {

**return** x < **0** ? **0** : sqrt(x);

}

**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** {

**return** x < p.x || x == p.x && y < p.y || x == p.x && y == p.y && z < p.z;

}

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

**return** Sign(x - p.x) == **0** && Sign(y - p.y) == **0** && Sign(z - p.z) == **0**;

}

Point **operator** +(**const** Point &p) **const** {

**return** Point(x + p.x, y + p.y, z + p.z);

}

Point **operator** -(**const** Point &p) **const** {

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

}

Point **operator** \*(**const** **double** k) **const** {

**return** Point(x \* k, y \* k, z \* k);

}

Point **operator** /(**const** **double** k) **const** {

**return** Point(x / k, y / k, z / k);

}

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());

}

**void** read() {

scanf(**"%lf%lf%lf"**, &x, &y, &z);

}

**void** write() {

printf(**"(%.10f, %.10f, %.10f)\n"**, x, y, z);

}

};

**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

**const** **double** eps = **1e**-**8**;

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

**inline** **int** cmp(**double** a) {

**return** a < -eps ? -**1** : a > eps;

}

**inline** **double** Sqrt(**double** a) {

**return** a <= **0** ? **0** : sqrt(a);

}

**struct** Point\_3 {

**double** x, y, z;

Point\_3() {

}

Point\_3(**double** x, **double** y, **double** z) : x(x), y(y), z(z) {

}

**void** Input() {

scanf(**"%lf%lf%lf"**, &x, &y, &z);

}

**double** Length() **const** {

**return** Sqrt(Sqr(x) + Sqr(y) + Sqr(z));

}

Point\_3 Unit() **const**;

Point\_3 Rotate(**const** Point\_3 &a, **double** delta) **const**;

};

Point\_3 **operator** + (**const** Point\_3 &a, **const** Point\_3 &b) {

**return** Point\_3(a.x + b.x, a.y + b.y, a.z + b.z);

}

Point\_3 **operator** - (**const** Point\_3 &a, **const** Point\_3 &b) {

**return** Point\_3(a.x - b.x, a.y - b.y, a.z - b.z);

}

Point\_3 **operator** \* (**const** Point\_3 &a, **double** b) {

**return** Point\_3(a.x \* b, a.y \* b, a.z \* b);

}

Point\_3 **operator** / (**const** Point\_3 &a, **double** b) {

**return** Point\_3(a.x / b, a.y / b, a.z / b);

}

Point\_3 Point\_3::Unit() **const** { **//这里只返回一个单位化的向量，自身值不改变**

**return** \***this** / Length();

}

Point\_3 Det(**const** Point\_3 &a, **const** Point\_3 &b) {

**return** Point\_3(a.y \* b.z - a.z \* b.y, a.z \* b.x - a.x \* b.z, a.x \* b.y - a.y \* b.x);

}

**double** Dot(**const** Point\_3 &a, **const** Point\_3 &b) {

**return** a.x \* b.x + a.y \* b.y + a.z \* b.z;

}

**double** Mix(**const** Point\_3 &a, **const** Point\_3 &b, **const** Point\_3 &c) {

**return** Dot(a, Det(b, c));

}

**double** dis(**const** Point\_3 &a, **const** Point\_3 &b){

**return** Sqrt(Sqr(a.x-b.x) + Sqr(a.y-b.y) + Sqr(a.z-b.z));

}

**void** printed(vector<Point\_3> &a) {

**int** i;

printf(**"face: \n"**);

**rep**(i,a.size()) {

printf(**"%lf %lf %lf \n"**,a[i].x,a[i].y,a[i].z);

}

printf(**"\n\n"**);

}

vector<Point\_3> a,b;

**int** n,m;

**bool** have[**70**][**70**][**70**];

**class** Tface{

**public**:

vector<Point\_3> p;

Point\_3 regular;

};

vector<Tface> face;

**bool** check\_Inface(Point\_3 a1, Point\_3 a2, Point\_3 a3 , vector<Point\_3> &a) {

**int** i;

**double** tmp=**0**;

Point\_3 regular=Det(a2-a1,a3-a1);

**double** k;

**rep**(i,a.size()){

k=(Dot(regular, a[i]-a1));

**if** (k==**0**) **continue**;

**if** (tmp==**0**) tmp=k;

**if** (k\*tmp<**0**) **return** **false**;

}

**return** **true**;

}

**bool** compareab(**const** Point\_3 &a, **const** Point\_3 &b){

**if** (cmp(a.x-b.x)) **return** cmp(a.x-b.x)<**0**;

**if** (cmp(a.y-b.y)) **return** cmp(a.y-b.y)<**0**;

**return** cmp(a.z-b.z)<**0**;

}

**int** num[**70**],numtot;

Tface find\_face(Point\_3 a1, Point\_3 a2, Point\_3 a3 , vector<Point\_3> &a) {

**int** i;

**double** tmp=**0**;

Point\_3 regular=Det(a2-a1,a3-a1);

**double** k;

Tface now;

now.p.clear();

now.regular=regular;

numtot=**0**;

**rep**(i,a.size()){

k=(Dot(regular, a[i]-a1));

**if** (k==**0**) {

now.p.push\_back(a[i]);

numtot++;

num[numtot]=i;

}

}

**int** j,kk;

**foru**(i,**1**,numtot)

**foru**(j,i,numtot)

**foru**(kk,j,numtot) have[num[i]][num[j]][num[kk]]=**true**;

sort(now.p.begin() , now.p.end(), compareab);

vector<Point\_3> con;

con.clear();

**int** open,closed;

closed=-**1**;

**rep**(i,now.p.size()) {

con.push\_back(now.p[i]); closed++;

**while** (closed>=**2** && Mix( now.regular , con[closed-**1**]-con[closed-**2**], con[closed]-con[closed-**2**])<**0**) {

con[closed-**1**]=con[closed];

con.pop\_back();

closed--;

}

}

open=closed;

**ford**(i,now.p.size()-**2**,**0**) {

con.push\_back(now.p[i]); closed++;

**while** (closed>=open+**2** && Mix( now.regular , con[closed-**1**]-con[closed-**2**], con[closed]-con[closed-**2**])<**0**) {

con[closed-**1**]=con[closed];

con.pop\_back();

closed--;

}

}

closed--;

**while** (con.size()>closed+**1**)

con.pop\_back();

now.p=con;

**return** now;

}

**void** count\_center(Point\_3 o , Tface face , **double** &x, **double** &y, **double** &z , **double** &tot) {

**int** i,j,k;

Point\_3 o2;

o2=face.p[**0**];

**double** volume;

**double** xx,yy,zz;

**foru**(i,**1**,face.p.size()-**2**){

volume=fabs(Mix( o2-o,face.p[i]-o,face.p[i+**1**]-o))/**6**;

tot+=volume;

xx=(o.x+o2.x+face.p[i].x+face.p[i+**1**].x)/**4.0**;

yy=(o.y+o2.y+face.p[i].y+face.p[i+**1**].y)/**4.0**;

zz=(o.z+o2.z+face.p[i].z+face.p[i+**1**].z)/**4.0**;

x=x+xx\*volume;

y=y+yy\*volume;

z=z+zz\*volume;

}

}

**double** work(vector<Point\_3> &a) {

**int** n=a.size();

**int** i,j,k;

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

sort(a.begin(),a.end(),compareab);

face.clear();

**rep**(i,n)

**foru**(j,i+**1**,n-**1**)

**foru**(k,j+**1**,n-**1**) **if** (!have[i][j][k]) **if** (check\_Inface(a[i],a[j],a[k],a)){

face.push\_back(find\_face(a[i],a[j],a[k],a));

}

Point\_3 o;

Point\_3 ans;

**double** volume=**0**;

ans.x=ans.y=ans.z=**0**;

o=a[**0**];

**rep**(i,face.size()) {

count\_center(o,face[i],ans.x,ans.y,ans.z,volume);

}

ans=ans/volume;

**double** len=dis(ans,a[**0**]);

**rep**(i,face.size()) {

len=min(len, fabs(Dot(face[i].regular,ans-face[i].p[**0**]) / face[i].regular.Length()));

}

**return** len;

}

**int** main(){

**int** i,j,k,test;

**while** (scanf(**"%d"**,&n)==**1**) {

a.clear();

Point\_3 tmp;

**rep**(i,n) {

tmp.Input();

a.push\_back(tmp);

}

**double** ans1,ans2;

ans1=work(a);

printf(**"%.5lf\n"**,ans1);

}

**return** **0**;

}

## 随机增量最小覆盖圆

**const** **double** eps=**1e**-**7**;

**const** **int** maxn=**100000**;

**class** circle{

point o;

**double** r;

}

point a[maxn];

**int** n;

circle ans;

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

**return** ((b.x-a.x)\*(c.y-a.y)-(b.y-a.y)\*(c.x-a.x));

}

**double** dis(point a, point b){

**return** (a.x-b.x)\*(a.x-b.x) + (a.y-b.y)\*(a.y-b.y);

}

**void** init(){

**int** i,j,k;

scanf(**"%d"**,&n);

**rep**(i,n) scanf(**"%lf%lf"**,&a[i].x,&a[i].y);

}

**bool** check(**const** point &a){

**return** sqr(a.x-ans.o.x) + sqr(a.y-ans.o.y) <= ans.r + zero;

}

**void** Mincircle(){

**int** i,j,k;

ans.r=**0**; ans.x=**0**; ans.y=**0**;

**rep**(i,n) **if** (!check(a[i])) {

ans.o=a[i]; ans.r=**0**;

**rep**(j,i) **if** (!check(a[j])) {

CircleCenter(a[i],a[j],ans.o);

ans.r=dis(ans.o,a[i]);

**rep**(k,j) **if** (!check(a[k])) {

CircleCenter(a[i],a[j],a[k],ans.o);

ans.r=dis(ans.o,a[i]);

}

}

}

printf(**"%.4lf\n"**,sqrt(ans.r));

}

## 圆面积模板（新）

**const** **int** N **=** 22222**;**

**const** **double** EPS **=** 1e**-**8**;**

**const** **double** PI **=** acos**(-**1.0**);**

**typedef** complex**<double>** Point**;**

**int** n**,** m**;**

**double** r**[**N**],** result**[**N**];**

Point c**[**N**];**

pair**<double,** **int>** event**[**N**];**

**int** sgn **(double** x**)** **{return** x **<** **-**EPS**?** **-**1**:** x **<** EPS**?** 0**:** 1**;}**

**double** det **(const** Point **&**a**,** **const** Point **&**b**)** **{** **return** a**.**real**()** **\*** b**.**imag**()** **-** a**.**imag**()** **\*** b**.**real**();}**

**void** addEvent **(double** a**,** **int** v**)** **{** event**[**m **++]** **=** make\_pair**(**a**,** v**); }**

**void** addPair **(double** a**,** **double** b**)** **{**

**if** **(**sgn**(**a **-** b**)** **<=** 0**)** **{**

addEvent**(**a**,** **+**1**);**

addEvent**(**b**,** **-**1**);**

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

addPair**(**a**,** **+**PI**);**

addPair**(-**PI**,** b**);**

**}**

**}**

Point polar **(double** t**)** **{** **return** Point**(**cos**(**t**),** sin**(**t**)); }**

Point radius **(int** i**,** **double** t**)** **{** **return** c**[**i**]** **+** polar**(**t**)** **\*** r**[**i**]; }**

**void** solve **()** **{**

// result[k]: the total area covered no less than k times

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

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

m **=** 0**;**

addEvent**(-**PI**,** 0**);** addEvent**(+**PI**,** 0**);**

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

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

**if** **(**sgn**(**abs**(**c**[**i**]** **-** c**[**j**])** **-** abs**(**r**[**i**]** **-** r**[**j**]))** **<=** 0**)** **{**

**if** **(**sgn**(**r**[**i**]** **-** r**[**j**])** **<=** 0**)** **{**

addPair**(-**PI**,** **+**PI**);**

**}**

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

**if** **(**sgn**(**abs**(**c**[**i**]** **-** c**[**j**])** **-** **(**r**[**i**]** **+** r**[**j**]))** **>=** 0**)** **{**

**continue;**

**}**

**double** d **=** abs**(**c**[**j**]** **-** c**[**i**]);**

Point b **=** **(**c**[**j**]** **-** c**[**i**])** **/** d **\*** r**[**i**];**

**double** t **=** acos**((**r**[**i**]** **\*** r**[**i**]** **+** d **\*** d **-** r**[**j**]** **\*** r**[**j**])** **/** **(**2 **\*** r**[**i**]** **\*** d**));**

Point a **=** b **\*** polar**(-**t**);**

Point c **=** b **\*** polar**(+**t**);**

addPair**(**arg**(**a**),** arg**(**c**));**

**}**

**}**

**}**

sort**(**event**,** event **+** m**);**

**int** count **=** event**[**0**].**second**;**

**for** **(int** j **=** 1**;** j **<** m**;** **++** j**)** **{**

**double** delta **=** event**[**j**].**first **-** event**[**j **-** 1**].**first**;**

result**[**count**]** **+=** r**[**i**]** **\*** r**[**i**]** **\*** **(**delta **-** sin**(**delta**));**

result**[**count**]** **+=** det**(**radius**(**i**,** event**[**j **-** 1**].**first**),** radius**(**i**,** event**[**j**].**first**));**

count **+=** event**[**j**].**second**;**

**}**

**}**

**}**

## 圆的面积并（可以求交）

#define maxn 55

#define maxN (maxn\*maxn+3\*maxn)

#define eps 1e-8

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

**struct** Tpoint{

**double** x,y;

};

**struct** Tcir{

**double** r;

Tpoint o;

}a[maxn];

**struct** Tinterval{

**double** x,y,Area,mid;

**int** ID,type;

**inline** **void** area(**double** l,**double** r)

{

**double** len=sqrt(sqr(l-r) + sqr(x-y));

**double** d=sqrt(sqr(a[ID].r)-sqr(len)/**4.0**);

**double** angle=atan(len/**2.0**/d);

Area=fabs(angle\*sqr(a[ID].r)-d\*len/**2.0**);

}

}inter[maxn];

**double** x[maxN],l,r;

**int** n,N,Nn;

**inline** **bool** compR(**const** Tcir &a,**const** Tcir &b){ **return** a.r>b.r;}

**inline** **void** Get(**int** i,**double** x,**double** &l,**double** &r){

**double** y=fabs(a[i].o.x-x);

**double** d=sqrt(fabs(sqr(a[i].r) - sqr(y)));

l=a[i].o.y+d;

r=a[i].o.y-d;

}

**inline** **void** Get\_Interval(**int** i,**double** l,**double** r){

Get(i,l,inter[Nn].x,inter[Nn+**1**].x);

Get(i,r,inter[Nn].y,inter[Nn+**1**].y);

Get(i,(l+r)/**2.0**,inter[Nn].mid,inter[Nn+**1**].mid);

inter[Nn].ID=inter[Nn+**1**].ID=i;

inter[Nn].area(l,r);inter[Nn+**1**].area(l,r);

inter[Nn].type=**1**;inter[Nn+**1**].type=-**1**;

Nn+=**2**;

}

**inline** **bool** comp(**const** Tinterval &a,**const** Tinterval &b){

**return** a.mid>b.mid+eps;

}

**inline** **void** Add(**double** xx){ x[N++]=xx;}

**inline** **double** dist(**const** Tpoint &a,**const** Tpoint &b){

**return** sqr(a.x-b.x)+sqr(a.y-b.y);

}

**inline** **void** Get\_Intersect(**const** Tcir &a,**const** Tcir &b)

{

**double** l=dist(a.o,b.o);

**double** s=((a.r-b.r)\*(a.r+b.r)/l+**1**)/**2**;

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

**double** ux=b.o.x-a.o.x,uy=b.o.y-a.o.y;

**double** ix=a.o.x+s\*ux+t\*uy,iy=a.o.y+s\*uy-t\*ux;

**double** jx=a.o.x+s\*ux-t\*uy,jy=a.o.y+s\*uy+t\*ux;

Add(ix);

Add(jx);

}

**int** main(){

scanf(**"%d"**,&n);

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

scanf(**"%lf%lf%lf"**,&a[i].o.x,&a[i].o.y,&a[i].r);

**int** p=**1**;

sort(a,a+n,compR); //======消除被覆盖的圆，可以清除

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

{

**bool** fl=**true**;

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

**if** (dist(a[i].o,a[j].o)<=sqr(a[i].r-a[j].r)+**1e**-**12**)

{

fl=**false**;

**break**;

}

**if** (fl) a[p++]=a[i];

}

n=p;

N=**0**;

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

{

Add(a[i].o.x-a[i].r);

Add(a[i].o.x+a[i].r);

Add(a[i].o.x);

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

**if** (dist(a[i].o,a[j].o)<=sqr(a[i].r+a[j].r)+eps)

Get\_Intersect(a[i],a[j]);

}

sort(x,x+N);

p=**1**;

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

**if** (fabs(x[i]-x[i-**1**])>eps) x[p++]=x[i];

N=p;

**double** ans=**0**;

**for** (**int** i=**0**;i+**1**<N;++i)

{

l=x[i],r=x[i+**1**];

Nn=**0**;

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

**if** (fabs(a[j].o.x-l)<a[j].r+eps && fabs(a[j].o.x-r)<a[j].r+eps)

Get\_Interval(j,l,r);

**if** (Nn)

{

sort(inter,inter+Nn,comp);

**int** cnt=**0**;

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

{

**if** (cnt>**0**) //====cnt 被几个interval覆盖

{

ans+=(fabs(inter[i-**1**].x-inter[i].x)+fabs(inter[i-**1**].y-inter[i].y))\*(r-l)/**2.0**;

ans+=inter[i-**1**].type\*inter[i-**1**].Area;

ans-=inter[i].type\*inter[i].Area;

}

cnt+=inter[i].type;

}

}

}

printf(**"%.8f\n"**,ans);

**return** **0**;

}

## 圆的面积模板(n^2logn)

#define eps 1e-8

#define maxn 105

#define inf 0

**const** **long** **double** pi=acos(-**1.0**);

**struct** Tpoint

{

**long** **double** x,y;

Tpoint(){}

Tpoint(**long** **double** a,**long** **double** b){x=a,y=b;}

**inline** **void** read()

{

**double** a,b;

scanf(**"%lf%lf"**,&a,&b);

x=a;y=b;

}

};

**struct** Tcir

{

Tpoint o;

**long** **double** r;

**inline** **void** read()

{

o.read();

**double** x;

scanf(**"%lf"**,&x);

r=x;

}

}c[maxn];

**struct** Tevent

{

**long** **double** a;

**int** delta,sgn;

**long** **double** x,y;

Tevent(){}

Tevent(**int** sign,**long** **double** key,**int** d,**long** **double** A,**long** **double** B)

{

sgn=sign;

a=key;

delta=d;

x=A;y=B;

}

};

vector <Tevent> Event;

**long** **double** Sum[maxn];

**inline** **bool** compR(**const** Tcir &a,**const** Tcir &b)

{

**return** a.r>b.r+eps;

}

**inline** **bool** cmp(**const** Tevent &a,**const** Tevent &b)//时间点按照极角排序

{

**return** a.a+eps<b.a || fabs(a.a-b.a)<eps && a.delta>b.delta;

}

**inline** **long** **double** gong(**long** **double** A,**long** **double** r)//弓形面积

{

**return** r\*r\*A/**2.0**-r\*cos(A/**2.0**)\*r\*sin(A/**2.0**);

}

**inline** **void** Add(**long** **double** x1,**long** **double** y1,**long** **double** x2,**long** **double** y2,**const** Tcir &a)

{

//一个a.o+(x1,y1) --> a.o+(x2,y2) 逆时针绕向的圆弧

**long** **double** l=atan2(y1,x1),r=atan2(y2,x2);

**if** (l>r)

{ //第三个参数为第j个圆的cnt\_\_

Event.push\_back(Tevent(**1**,l,**1**,a.o.x+x1,a.o.y+y1));

Event.push\_back(Tevent(**1**,pi,-**1**,a.o.x-a.r,a.o.y));

Event.push\_back(Tevent(**1**,-pi,**1**,a.o.x-a.r,a.o.y));

Event.push\_back(Tevent(**1**,r,-**1**,a.o.x+x2,a.o.y+y2));

}**else**

{

Event.push\_back(Tevent(**1**,l,**1**,a.o.x+x1,a.o.y+y1));

Event.push\_back(Tevent(**1**,r,-**1**,a.o.x+x2,a.o.y+y2));

}

}

**inline** **long** **double** dist(**const** Tpoint &a,**const** Tpoint &b)

{

**return** sqr(a.x-b.x)+sqr(a.y-b.y);

}

**inline** **void** Get\_Intersect(**const** Tcir &a,**const** Tcir &b)

{

**long** **double** l=dist(a.o,b.o);

**long** **double** s=((a.r-b.r)\*(a.r+b.r)/l+**1**)/**2**;

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

**long** **double** ux=b.o.x-a.o.x,uy=b.o.y-a.o.y;

**long** **double** ix=a.o.x+s\*ux+t\*uy,iy=a.o.y+s\*uy-t\*ux;

**long** **double** jx=a.o.x+s\*ux-t\*uy,jy=a.o.y+s\*uy+t\*ux;

//求交点

**long** **double** dx1=jx-a.o.x,dy1=jy-a.o.y;

**long** **double** dx2=ix-a.o.x,dy2=iy-a.o.y;

**long** **double** x=(dx1+dx2),y=(dy1+dy2);

**long** **double** len=sqrt(sqr(x)+sqr(y));

**if** (fabs(len)<eps)

{

x=-dy1,y=dx1;

}**else**

{

x/=len;y/=len;

x\*=a.r;y\*=a.r;

}

//求得弧的一个中点

Tpoint tmp(a.o.x+x,a.o.y+y);

**if** (sqrt(dist(tmp,b.o))>b.r+eps) x=-x,y=-y;

//如果不在圆内，则一定是另一个点

**if** (dx2\*y-x\*dy2>-eps) Add(ix-a.o.x,iy-a.o.y,jx-a.o.x,jy-a.o.y,a);

**else** Add(jx-a.o.x,jy-a.o.y,ix-a.o.x,iy-a.o.y,a);

}

**int** main()

{

cout.precision(**5**);

cout.setf(ios::fixed);

**int** n;

scanf(**"%d"**,&n);

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

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

{

c[i].read();

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

{

**if** (fabs(c[i].r-c[j].r)<eps && fabs(c[i].o.x-c[j].o.x)<eps && fabs(c[i].o.y-c[j].o.y)<eps)

{

--i;

--n;

**break**;

}

}//去掉重复的圆，如需重复计算，则在struct增加一个记录圆的个数的域 cnt\_\_

}

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

{

Event.clear();

**int** cover=**1**; //===cover的初值赋为 cnt\_\_

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

**if** (i!=j)

{

**long** **double** d=sqrt(dist(c[i].o,c[j].o));

**if** (d>c[i].r+c[j].r+eps) **continue**;//相离

**if** (d<fabs(c[i].r-c[j].r)+eps)

{

**if** (c[i].r+eps<c[j].r) ++cover;//被包含

}**else** Get\_Intersect(c[i],c[j]);//相交

}

Event.push\_back(Tevent(**1**,-pi,cover,c[i].o.x-c[i].r,c[i].o.y));

Event.push\_back(Tevent(-**1**,**0**,**0**,c[i].o.x+c[i].r,c[i].o.y));

//过x轴后，下弧结束，开始处理上弧，所以sign\*=-1

Event.push\_back(Tevent(**1**,pi,-cover,c[i].o.x-c[i].r,c[i].o.y));

sort(Event.begin(),Event.end(),cmp);

**int** sign=-**1**,cnt=**0**;

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

{

**if** (j)

{

**long** **double** A=Event[j-**1**].a,B=Event[j].a;

**long** **double** x1=Event[j-**1**].x,y1=Event[j-**1**].y;

**long** **double** x2=Event[j].x,y2=Event[j].y;

**if** (sign==-**1**)

{

//下弧

Sum[cnt]-=(y1+inf+y2+inf)\*fabs(x1-x2)/**2.0**-gong(B-A,c[i].r);

Sum[cnt-**1**]+=(y1+inf+y2+inf)\*fabs(x1-x2)/**2.0**-gong(B-A,c[i].r);

}**else**

{

//上弧

Sum[cnt-**1**]-=(y1+inf+y2+inf)\*fabs(x1-x2)/**2.0**+gong(B-A,c[i].r);

Sum[cnt]+=(y1+inf+y2+inf)\*fabs(x1-x2)/**2.0**+gong(B-A,c[i].r);

}

}

sign\*=Event[j].sgn;

cnt+=Event[j].delta;

}

}

**long** **double** Ans1=**0**,Ans2=**0**;

**for** (**int** i=**1**;i<=n;i+=**2**)

Ans1+=Sum[i];

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

Ans2+=Sum[i];

cout << Ans1 << **" "** << Ans2 << endl;

**return** **0**;

}

## 直线和凸包交点（返回最近和最远点）

**double** calc**(**point a**,** point b**){**

**double** k**=**atan2**(**b**.**y**-**a**.**y **,** b**.**x**-**a**.**x**);** **if** **(**k**<**0**)** k**+=**2**\***pi**;return** k**;**

**}**//= the convex must compare y, then x£¬a[0] is the lower-right point

//======= three is no 3 points in line. a[] is convex 0~n-1

**void** prepare**(**point a**[]** **,double** w**[],int** **&**n**)** **{**

**int** i**;** rep**(**i**,**n**)** a**[**i**+**n**]=**a**[**i**];** a**[**2**\***n**]=**a**[**0**];**

rep**(**i**,**n**)** **{** w**[**i**]=**calc**(**a**[**i**],**a**[**i**+**1**]);**w**[**i**+**n**]=**w**[**i**];}**

**}**

**int** find**(double** k**,int** n **,** **double** w**[]){**

**if** **(**k**<=**w**[**0**]** **||** k**>**w**[**n**-**1**])** **return** 0**;** **int** l**,**r**,**mid**;** l**=**0**;** r**=**n**-**1**;**

**while** **(**l**<=**r**)** **{** mid**=(**l**+**r**)/**2**;if** **(**w**[**mid**]>=**k**)** r**=**mid**-**1**;** **else** l**=**mid**+**1**;**

**}return** r**+**1**;**

**}**

**int** dic**(const** point **&**a**,** **const** point **&**b **,** **int** l **,int** r **,** point c**[])** **{**

**int** s**;** **if** **(**area**(**a**,**b**,**c**[**l**])<**0**)** s**=-**1**;** **else** s**=**1**;** **int** mid**;**

**while** **(**l**<=**r**)** **{**

mid**=(**l**+**r**)/**2**;** **if** **(**area**(**a**,**b**,**c**[**mid**])\***s **<=** 0**)** r**=**mid**-**1**;** **else** l**=**mid**+**1**;**

**}return** r**+**1**;**

**}**

point get**(const** point **&**a**,** **const** point **&**b**,** point s1**,** point s2**)** **{**

**double** k1**,**k2**;** point tmp**;** k1**=**area**(**a**,**b**,**s1**);** k2**=**area**(**a**,**b**,**s2**);**

**if** **(**cmp**(**k1**)==**0**)** **return** s1**;** **if** **(**cmp**(**k2**)==**0**)** **return** s2**;**

tmp**=(**s1**\***k2 **-** s2**\***k1**)** **/** **(**k2**-**k1**);** **return** tmp**;**

**}**

**bool** line\_cross\_convex**(**point a**,** point b **,**point c**[]** **,** **int** n**,** point **&**cp1**,** point **&**cp2 **,** **double** w**[])** **{**

**int** i**,**j**;**

i**=**find**(**calc**(**a**,**b**),**n**,**w**);**

j**=**find**(**calc**(**b**,**a**),**n**,**w**);**

**double** k1**,**k2**;**

k1**=**area**(**a**,**b**,**c**[**i**]);** k2**=**area**(**a**,**b**,**c**[**j**]);**

**if** **(**cmp**(**k1**)\***cmp**(**k2**)>**0**)** **return** **false;** //no cross

**if** **(**cmp**(**k1**)==**0 **||** cmp**(**k2**)==**0**)** **{** //cross a point or a line in the convex

**if** **(**cmp**(**k1**)==**0**)** **{**

**if** **(**cmp**(**area**(**a**,**b**,**c**[**i**+**1**]))==**0**)** **{**cp1**=**c**[**i**];** cp2**=**c**[**i**+**1**];}**

**else** cp1**=**cp2**=**c**[**i**];** **return** **true;**

**}**

**if** **(**cmp**(**k2**)==**0**)** **{**

**if** **(**cmp**(**area**(**a**,**b**,**c**[**j**+**1**]))==**0**)** **{**cp1**=**c**[**j**];**cp2**=**c**[**j**+**1**];**

**}else** cp1**=**cp2**=**c**[**j**];**

**}return** **true;**

**}**

**if** **(**i**>**j**)** swap**(**i**,**j**);** **int** x**,**y**;** x**=**dic**(**a**,**b**,**i**,**j**,**c**);** y**=**dic**(**a**,**b**,**j**,**i**+**n**,**c**);**

cp1**=**get**(**a**,**b**,**c**[**x**-**1**],**c**[**x**]);** cp2**=**get**(**a**,**b**,**c**[**y**-**1**],**c**[**y**]);**

**return** **true;}**

## 最小覆盖球

**int** npoint, nouter;

Tpoint pt[**200000**], outer[**4**],res;

**double** radius,tmp;

**inline** **double** dist(Tpoint p1, Tpoint p2) {

**double** dx=p1.x-p2.x, dy=p1.y-p2.y, dz=p1.z-p2.z;

**return** ( dx\*dx + dy\*dy + dz\*dz );

}

**inline** **double** dot(Tpoint p1, Tpoint p2) {

**return** p1.x\*p2.x + p1.y\*p2.y + p1.z\*p2.z;

}

**void** ball() {

Tpoint q[**3**]; **double** m[**3**][**3**], sol[**3**], L[**3**], det;

**int** i,j;

res.x = res.y = res.z = radius = **0**;

**switch** ( nouter ) {

**case** **1**: res=outer[**0**]; **break**;

**case** **2**:

res.x=(outer[**0**].x+outer[**1**].x)/**2**;

res.y=(outer[**0**].y+outer[**1**].y)/**2**;

res.z=(outer[**0**].z+outer[**1**].z)/**2**;

radius=dist(res, outer[**0**]);

**break**;

**case** **3**:

**for** (i=**0**; i<**2**; ++i ) {

q[i].x=outer[i+**1**].x-outer[**0**].x;

q[i].y=outer[i+**1**].y-outer[**0**].y;

q[i].z=outer[i+**1**].z-outer[**0**].z;

}

**for** (i=**0**; i<**2**; ++i) **for**(j=**0**; j<**2**; ++j)

m[i][j]=dot(q[i], q[j])\***2**;

**for** (i=**0**; i<**2**; ++i ) sol[i]=dot(q[i], q[i]);

**if** (fabs(det=m[**0**][**0**]\*m[**1**][**1**]-m[**0**][**1**]\*m[**1**][**0**])<eps)

**return**;

L[**0**]=(sol[**0**]\*m[**1**][**1**]-sol[**1**]\*m[**0**][**1**])/det;

L[**1**]=(sol[**1**]\*m[**0**][**0**]-sol[**0**]\*m[**1**][**0**])/det;

res.x=outer[**0**].x+q[**0**].x\*L[**0**]+q[**1**].x\*L[**1**];

res.y=outer[**0**].y+q[**0**].y\*L[**0**]+q[**1**].y\*L[**1**];

res.z=outer[**0**].z+q[**0**].z\*L[**0**]+q[**1**].z\*L[**1**];

radius=dist(res, outer[**0**]);

**break**;

**case** **4**:

**for** (i=**0**; i<**3**; ++i) {

q[i].x=outer[i+**1**].x-outer[**0**].x;

q[i].y=outer[i+**1**].y-outer[**0**].y;

q[i].z=outer[i+**1**].z-outer[**0**].z;

sol[i]=dot(q[i], q[i]);

}

**for** (i=**0**;i<**3**;++i)

**for**(j=**0**;j<**3**;++j) m[i][j]=dot(q[i],q[j])\***2**;

det= m[**0**][**0**]\*m[**1**][**1**]\*m[**2**][**2**]

+ m[**0**][**1**]\*m[**1**][**2**]\*m[**2**][**0**]

+ m[**0**][**2**]\*m[**2**][**1**]\*m[**1**][**0**]

- m[**0**][**2**]\*m[**1**][**1**]\*m[**2**][**0**]

- m[**0**][**1**]\*m[**1**][**0**]\*m[**2**][**2**]

- m[**0**][**0**]\*m[**1**][**2**]\*m[**2**][**1**];

**if** ( fabs(det)<eps ) **return**;

**for** (j=**0**; j<**3**; ++j) {

**for** (i=**0**; i<**3**; ++i) m[i][j]=sol[i];

L[j]=( m[**0**][**0**]\*m[**1**][**1**]\*m[**2**][**2**]

+ m[**0**][**1**]\*m[**1**][**2**]\*m[**2**][**0**]

+ m[**0**][**2**]\*m[**2**][**1**]\*m[**1**][**0**]

- m[**0**][**2**]\*m[**1**][**1**]\*m[**2**][**0**]

- m[**0**][**1**]\*m[**1**][**0**]\*m[**2**][**2**]

- m[**0**][**0**]\*m[**1**][**2**]\*m[**2**][**1**]

) / det;

**for** (i=**0**; i<**3**; ++i)

m[i][j]=dot(q[i], q[j])\***2**;

}

res=outer[**0**];

**for** (i=**0**; i<**3**; ++i ) {

res.x += q[i].x \* L[i];

res.y += q[i].y \* L[i];

res.z += q[i].z \* L[i];

}

radius=dist(res, outer[**0**]);

}

}

**void** minball(**int** n) {

ball();

//printf("(%.3lf,%.3lf,%.3lf) %.3lf\n", res.x,res.y,res.z,radius);

**if** ( nouter<**4** )

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

**if** (dist(res, pt[i])-radius>eps) {

outer[nouter]=pt[i];

++nouter;

minball(i);

--nouter;

**if** (i>**0**) {

Tpoint Tt = pt[i];

memmove(&pt[**1**], &pt[**0**], **sizeof**(Tpoint)\*i);

pt[**0**]=Tt;

}

}

}

**int** main(){

scanf(**"%d"**,&npoint);

**for** (**int** i=**0**;i<npoint;i++) scanf(**"%lf%lf%lf"**,&pt[i].x,&pt[i].y,&pt[i].z);

radius=-**1**;

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

**if** (dist(res,pt[i])-radius>eps){

nouter=**1**;

outer[**0**]=pt[i];

minball(i);

}

}

printf(**"%.3lf\n"**,sqrt(radius));

}

# 图论

## 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],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){

**int** i;

**for** (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;

}

**int** main(){

**for** (;scanf(**"%d%d"**,&n0,&m0)==**2**;){

**int** i,j;

**if** (n0+m0==**0**) **break**;

na=nb=**0**;

**for** (i=**0**;i<n0;i++) {

scanf(**"%s"**,s[i]);

**for** (j=**0**;j<m0;j++) **if** (s[i][j]==**'H'**) pa[na][**0**]=i,pa[na++][**1**]=j;

**else** **if** (s[i][j]==**'m'**) pb[nb][**0**]=i,pb[nb++][**1**]=j;

}

n=na;

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

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

w[i][j]=**300**-abs(pa[i][**0**]-pb[j][**0**])-abs(pa[i][**1**]-pb[j][**1**]);

// printf("%d ",300-w[i][j]);

}

//printf("\n");

}

printf(**"%d\n"**,**300**\*n-km());

}

**return** **0**;

}

## 求最小上下界网络流

#define L 60

#define inf 300000

**int** c[L][L],mi[L];

**int** fa[L],Q[L];

**int** S,T,l,r;

**int** bfs(){

**int** ans=**0**,i,x,y;

**while** (**1**){

**for** (i=**0**;i<=T;++i) mi[i]=-**1**,fa[i]=-**1**;

l=r=**0**;

Q[r++]=S;

mi[S]=inf;

**while** (l<r){

x=Q[l++];

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

**if** (mi[i]==-**1** && c[x][i]>**0**){

mi[i]=min(c[x][i],mi[x]);

Q[r++]=i;

fa[i]=x;

}

}

**if** (fa[T]==-**1**) **return** ans;

ans+=mi[T];

x=T;

y=fa[T];

**while** (y!=-**1**){

c[x][y]+=mi[T];

c[y][x]-=mi[T];

x=y;

y=fa[y];

}

}

}

**int** n,m,x,y,i,j,tot,a,b,v;

**char** s[**100**];

**int** main(){

**while** (scanf(**"%d%d"**,&n,&m),n||m){

x=**0**;y=n+**1**;S=n+**2**;T=n+**3**;

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

**for** (j=**0**;j<=T;++j)

c[i][j]=**0**;

tot=**0**;

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

scanf(**" %s"**,s);

**if** (s[**0**]==**'+'**) a=x;

**else** **if** (s[**0**]==**'-'**) a=y;

**else** sscanf(s,**"%d"**,&a);

scanf(**" %s"**,s);

**if** (s[**0**]==**'+'**) b=x;

**else** **if** (s[**0**]==**'-'**) b=y;

**else** sscanf(s,**"%d"**,&b);

scanf(**"%d"**,&v);

c[a][b]=inf;

c[S][b]+=v;

c[a][T]+=v;

tot+=v;

}

**int** e=bfs();

c[y][x]=inf;

**int** d=bfs();

**if** (e+d!=tot) printf(**"impossible\n"**);

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

}

}

## 求最小下界网络流\_反边（optional）

**void** init(){

**int** i,j,k,t;

nn=**0**;

**foru**(i,**1**,m) {

scanf(“%d%d%d”,&j,&k,&t)

down[j][k]=t;

c[j][k]=**200**\***100**\***60**; //上界

inner[k]+=t;

outer[j]+=t;

}

**foru**(i,**1**,n+**2**) **if** (inner[i]>outer[i]) c[n+**3**][i]=inner[i]-outer[i];

**else** c[i][n+**4**]=-(inner[i]-outer[i]);

nn=**0**;

**foru**(i,**1**,n+**2**) **if** (visit[i]) nn++;

}

**int** main(){

**int** i,j,k,test;

**while** (**1**){

scanf(**"%d%d\n"**,&n,&m);

**if** (n==**0** && m==**0**) **break**;

**char** ch;

init();

bfs\_prepare\_forword(n+**1**);

**if** (closed!=nn) {printf(**"impossible\n"**); **continue**;}

bfs\_prepare\_back(n+**2**);

**if** (closed!=nn) {printf(**"impossible\n"**); **continue**;}

ans=**0**;

c[n+**2**][n+**1**]=**200**\***100**\***60**; //important

**while** (find(n+**3**,n+**4**)) {

improve();

}

**int** ans1=**0**;

c[n+**2**][n+**1**]=**0**;

**foru**(i,**1**,n+**2**)

**foru**(j,**1**,n+**2**) **if** (down[i][j]>**0**) {

c[j][i]=**200**\***100**\***60** - c[i][j]; // 200\*100\*60是上界， 这个式子表示正向边的流量，即反向边的容量

**if** (c[j][i]<**0**) c[j][i]=**0**;

c[i][j]=**0**;

}

**foru**(i,**1**,n+**2**) **if** (down[n+**1**][i]>**0**) {

ans1+=down[n+**1**][i] + c[i][n+**1**];

}

ans=**0**;

**while** (find(n+**2**,n+**1**)) { // 汇到源求个最大流

improve();

}

printf(**"%d\n"**,ans1 - ans);

}

**return** **0**;

}

## 无向图最小割

#define typec int // type of res **注意具体范围**

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

**const** typec maxw = **1000**; // maximum edge weight

typec g[V][V], w[V] ; **//g[i][j]=g[j][i]**

**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;

}

## Voronoi

#define Oi(e) ((e)->oi)

#define Dt(e) ((e)->dt)

#define On(e) ((e)->on)

#define Op(e) ((e)->op)

#define Dn(e) ((e)->dn)

#define Dp(e) ((e)->dp)

#define Other(e, p) ((e)->oi == p ? (e)->dt : (e)->oi)

#define Next(e, p) ((e)->oi == p ? (e)->on : (e)->dn)

#define Prev(e, p) ((e)->oi == p ? (e)->op : (e)->dp)

#define V(p1, p2, u, v) (u = p2->x - p1->x, v = p2->y - p1->y)

#define C2(u1, v1, u2, v2) (u1 \* v2 - v1 \* u2)

#define C3(p1, p2, p3) ((p2->x - p1->x) \* (p3->y - p1->y) - (p2->y - p1->y) \* (p3->x - p1->x))

#define Dot(u1, v1, u2, v2) (u1 \* u2 + v1 \* v2)

#define dis(a,b) (sqrt( (a->x - b->x) \* (a->x - b->x) + (a->y - b->y) \* (a->y - b->y) ))

**const** **int** maxn = **110024**;

**const** **double** eps=**1e**-**7**;

**const** **int** aix=**4**;

**int** n, M , k;

**struct** gEdge

{

**int** u, v;

**double** w;

**bool** **operator** < (**const** gEdge &e1) **const** {**return** w < e1.w-eps;}

}E[aix \* maxn], MST[maxn];

**int** b[maxn];

**int** Find(**int** x)

{

**while** (x!=b[x]) {

b[x]=b[b[x]];

x=b[x];

}

**return** x;

}

**void** Kruskal()

{

**int** m1,m2;

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

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

sort(E, E + M);

**for**(**int** i = **0**, kk = **0**; i < M && kk < n - **1**; i ++)

{

m1=Find(E[i].u);

m2=Find(E[i].v);

**if** (m1!=m2) {

b[m1]=m2; MST[kk++] = E[i];

}

}/\*

for(int i = 0; i < n - 1; i++)

printf("%d %d %.3lf\n", MST[i].u, MST[i].v, MST[i].w);

\*/

}

**struct** point

{

**double** x, y;

**int** index;

**struct** edge \*in;

**bool** **operator** < (**const** point &p1) **const**

{

**return** x < p1.x-eps || ( abs(x-p1.x)<=eps && y < p1.y-eps);

}

};

**struct** edge

{

point \*oi, \*dt;

edge \*on, \*op, \*dn, \*dp;

};

point p[maxn], \*Q[maxn];

edge mem[aix \* maxn], \*elist[aix \* maxn];

**int** nfree;

//memory

**void** Alloc\_memory()

{

nfree = aix \* n;

edge \*e = mem;

**for**(**int** i = **0**; i < nfree; i ++) elist[i] = e++;

}

//Add an edge to a ring of edges

**void** Splice(edge \*a, edge \*b, point \*v)

{

edge \*next;

**if**(Oi(a) == v) next = On(a), On(a) = b;

**else** next = Dn(a), Dn(a) = b;

**if**(Oi(next) == v) Op(next) = b;

**else** Dp(next) = b;

**if**(Oi(b) == v) On(b) = next, Op(b) = a;

**else** Dn(b) = next, Dp(b) = a;

}

//Initialise a new edge

edge \*Make\_edge(point \*u, point \*v)

{

edge \*e = elist[--nfree];

e->on = e->op = e->dn = e->dp = e; e->oi = u; e->dt = v;

**if**(!u->in) u->in = e; **if**(!v->in) v->in = e;

**return** e;

}

//Creates a new edge and adds it to two rings of edges.

edge \*Join(edge \*a, point \*u, edge \*b, point \*v, **int** side)

{

edge \*e = Make\_edge(u, v);

**if**(side == **1**)

{

**if**(Oi(a) == u) Splice(Op(a), e, u);

**else** Splice(Dp(a), e, u);

Splice(b, e, v);

}

**else**

{

Splice(a, e, u);

**if**(Oi(b) == v) Splice(Op(b), e, v);

**else** Splice(Dp(b), e, v);

}

**return** e;

}

//Remove an edge

**void** Remove(edge \*e)

{

point \*u = Oi(e), \*v = Dt(e);

**if**(u->in == e) u->in = e->on; **if**(v->in == e) v->in = e->dn;

**if**(Oi(e->on) == u) e->on->op = e->op;

**else** e->on->dp = e->op;

**if**(Oi(e->op) == u) e->op->on = e->on;

**else** e->op->dn = e->on;

**if**(Oi(e->dn) == v) e->dn->op = e->dp;

**else** e->dn->dp = e->dp;

**if**(Oi(e->dp) == v) e->dp->on = e->dn;

**else** e->dp->dn = e->dn;

elist[nfree++] = e;

}

//Determines the lower tangent of two triangulations

**void** Low\_tangent(edge \*e\_l, point \*o\_l, edge \*e\_r, point \*o\_r, edge \*\*l\_low, point \*\*OL, edge \*\*r\_low, point \*\*OR)

{

point \*d\_l = Other(e\_l, o\_l), \*d\_r = Other(e\_r, o\_r);

**while**(**1**)

{

**if**(C3(o\_l, o\_r, d\_l) < -eps)

{

e\_l = Prev(e\_l, d\_l);

o\_l = d\_l; d\_l = Other(e\_l, o\_l);

}

**else** **if**(C3(o\_l, o\_r, d\_r) < -eps)

{

e\_r = Next(e\_r, d\_r);

o\_r = d\_r; d\_r = Other(e\_r, o\_r);

}

**else** **break**;

}

\*OL = o\_l, \*OR = o\_r;

\*l\_low = e\_l, \*r\_low = e\_r;

}

**void** Merge(edge \*lr, point \*s, edge \*rl, point \*u, edge \*\*tangent)

{

**double** l1, l2, l3, l4, r1, r2, r3, r4, cot\_L, cot\_R, u1, v1, u2, v2, n1, cot\_n, P1, cot\_P;

point \*O, \*D, \*OR, \*OL;

edge \*B, \*L, \*R;

Low\_tangent(lr, s, rl, u, &L, &OL, &R, &OR);

\*tangent = B = Join(L, OL, R, OR, **0**);

O = OL, D = OR;

**do**

{

edge \*El = Next(B, O), \*Er = Prev(B, D), \*next, \*prev;

point \*l = Other(El, O), \*r = Other(Er, D);

V(l, O, l1, l2); V(l, D, l3, l4); V(r, O, r1, r2); V(r, D, r3, r4);

**double** cl = C2(l1, l2, l3, l4), cr = C2(r1, r2, r3, r4);

**bool** BL = cl > eps, BR = cr > eps;

**if**(!BL && !BR) **break**;

**if**(BL)

{

**double** dl = Dot(l1, l2, l3, l4);

cot\_L = dl / cl;

**do**

{

next = Next(El, O);

V(Other(next, O), O, u1, v1); V(Other(next, O), D, u2, v2);

n1 = C2(u1, v1, u2, v2);

**if**(!(n1 > eps)) **break**;

cot\_n = Dot(u1, v1, u2, v2) / n1;

**if**(cot\_n > cot\_L) **break**;

Remove(El);

El = next;

cot\_L = cot\_n;

}

**while**(**1**);

}

**if**(BR)

{

**double** dr = Dot(r1, r2, r3, r4);

cot\_R = dr / cr;

**do**

{

prev = Prev(Er, D);

V(Other(prev, D), O, u1, v1); V(Other(prev, D), D, u2, v2);

P1 = C2(u1, v1, u2, v2);

**if**(!(P1 > eps)) **break**;

cot\_P = Dot(u1, v1, u2, v2) / P1;

**if**(cot\_P > cot\_R) **break**;

Remove(Er);

Er = prev;

cot\_R = cot\_P;

}

**while**(**1**);

}

l = Other(El, O); r = Other(Er, D);

**if**(!BL || (BL && BR && cot\_R < cot\_L)) { B = Join(B, O, Er, r, **0**); D = r; }

**else** { B = Join(El, l, B, D, **0**); O = l; }

}

**while**(**1**);

}

**void** Divide(**int** s, **int** t, edge \*\*L, edge \*\*R)

{

edge \*a, \*b, \*c, \*ll, \*lr, \*rl, \*rr, \*tangent;

**int** n = t - s + **1**;

**if**(n == **2**) \*L = \*R = Make\_edge(Q[s], Q[t]);

**else** **if**(n == **3**)

{

a = Make\_edge(Q[s], Q[s + **1**]), b = Make\_edge(Q[s + **1**], Q[t]);

Splice(a, b, Q[s + **1**]);

**double** v = C3(Q[s], Q[s + **1**], Q[t]);

**if**(v > eps)

{

c = Join(a, Q[s], b, Q[t], **0**);

\*L = a; \*R = b;

}

**else** **if**(v < -eps)

{

c = Join(a, Q[s], b, Q[t], **1**);

\*L = c; \*R = c;

}

**else** { \*L = a; \*R = b; }

}

**else** **if**(n > **3**)

{

**int** split = (s + t) / **2**;

Divide(s, split, &ll, &lr); Divide(split + **1**, t, &rl, &rr);

Merge(lr, Q[split], rl, Q[split + **1**], &tangent);

**if**(Oi(tangent) == Q[s]) ll = tangent;

**if**(Dt(tangent) == Q[t]) rr = tangent;

\*L = ll; \*R = rr;

}

}

**void** OLE(){

**while** (**1**) {

printf(**"no\n"**);

}

}

**void** Make\_Graph()

{

edge \*start, \*e;

point \*u, \*v;

**int** i;

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

{

u = &p[i];

start = e = u->in;

**do**

{

v = Other(e, u);

**if**(u < v)

{

E[M].u = u - p, E[M].v = v - p;

E[M++].w = dis(u,v);

**if** (M>=aix\*maxn) OLE();

}

e = Next(e, u);

}

**while**(e != start);

}

}

**void** solve()

{

**int** i , test;

scanf(**"%d"**,&test);

**while** (test)

{

test--;

n=**0**;

**double** ans = -**1**;

scanf(**"%d"**, &n);

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

scanf(**"%lf%lf"**,&p[i].x,&p[i].y);

p[i].index=i;

p[i].in=**NULL**;

}

Alloc\_memory();

**if**(n == **1** || n==**0** ){ **continue**;} // else RE

sort(p, p + n);

//=========点不能有重点，有的话不满足voronoi图的性质了

**for**(i = **0**; i < n; i++) Q[i] = p + i;

edge \*L, \*R;

Divide(**0**, n - **1**, &L, &R);

M = **0**;

Make\_Graph();

Kruskal();

// puts("---------------------");

}

}

**int** main()

{

freopen(**"input.txt"**,**"r"**,stdin);

freopen(**"output.txt"**,**"w"**,stdout);

solve();

**return** **0**;

}

## KD-TREE

/\*

KD-Tree

求n个点中距离一个定点的最近距离是多少

单次查询O(sqrt(n))

\*/

#define maxn 100005

#define LL long long

#define inf 1000000000000000000LL

**long** **long** res;

**struct** Tpoint

{

**int** x,y;

}a[maxn],p,bak[maxn];

**inline** LL dist(Tpoint a,Tpoint b)

{

**return** sqr(a.x-b.x)+sqr(a.y-b.y);

}

**inline** **bool** cmpx(**const** Tpoint &a,**const** Tpoint &b)

{

**return** (a.x<b.x || a.x==b.x && a.y<b.y);

}

**inline** **bool** cmpy(**const** Tpoint &a,**const** Tpoint &b)

{

**return** (a.y<b.y || a.y==b.y && a.x<b.x);

}

**struct** Trect

{

**int** minx,maxx,miny,maxy;

**inline** **void** rect(Tpoint &a)

{

minx=maxx=a.x;

miny=maxy=a.y;

}

**inline** **void** merge(Trect &a)

{

minx=min(a.minx,minx);

maxx=max(a.maxx,maxx);

miny=min(a.miny,miny);

maxy=max(a.maxy,maxy);

}

**inline** LL dist(**const** Tpoint &P)

{

**if**(P.x<=minx && P.y<=miny) **return** sqr(P.x-minx)+sqr(P.y-miny);

**if**(P.x<=maxx && P.y<=miny) **return** sqr(P.y-miny);

**if**(P.x>=maxx && P.y<=miny) **return** sqr(P.x-maxx)+sqr(P.y-miny);

**if**(P.x>=maxx && P.y<=maxy) **return** sqr(P.x-maxx);

**if**(P.x>=maxx && P.y>=maxy) **return** sqr(P.x-maxx)+sqr(P.y-maxy);

**if**(P.x>=minx && P.y>=maxy) **return** sqr(P.y-maxy);

**if**(P.x<=minx && P.y>=maxy) **return** sqr(P.x-minx)+sqr(P.y-maxy);

**if**(P.x<=minx && P.y<=maxy) **return** sqr(P.x-minx);

**return** **0**;

}

};

**struct** TKDTree

{

Tpoint p;

Trect rt;

}Tree[(**1**<<**18**)+**5**];

**inline** **void** Build(**int** num,**int** l,**int** r,**int** dep)

{

**if** (l>=r) **return**;

**int** mid=(l+r)/**2**;

nth\_element(a+l,a+mid,a+r,dep?cmpx:cmpy);

Tree[num].p=a[mid];

Tree[num].rt.rect(a[mid]);

**if** (l==r) **return**;

Build(num\***2**,l,mid,!dep);

Build(num\***2**+**1**,mid+**1**,r,!dep);

**if** (l<mid) Tree[num].rt.merge(Tree[num\***2**].rt);

**if** (mid+**1**<r) Tree[num].rt.merge(Tree[num\***2**+**1**].rt);

}

**inline** **void** Query(**int** num,**int** l,**int** r,**int** dep)

{

**int** mid=(l+r)/**2**;

**if** (Tree[num].rt.dist(p)>=res) **return**;

LL dt=dist(p,Tree[num].p);

**if** (dt && dt<res) res=dt;

**if** (dep && cmpx(p,Tree[num].p) || !dep && cmpy(p,Tree[num].p))

{

**if** (l<mid) Query(num\***2**,l,mid,!dep);

**if** (mid+**1**<r) Query(num\***2**+**1**,mid+**1**,r,!dep);

}**else**

{

**if** (mid+**1**<r) Query(num\***2**+**1**,mid+**1**,r,!dep);

**if** (l<mid) Query(num\***2**,l,mid,!dep);

}

}

**int** main()

{

**int** T;

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

{

**int** n;

scanf(**"%d"**,&n);

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

{

scanf(**"%d%d"**,&a[i].x,&a[i].y);

bak[i]=a[i];

}

Build(**1**,**0**,n,**0**);

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

{

p=bak[i];

res=inf;

Query(**1**,**0**,n,**0**);

printf(**"%lld\n"**,res);

}

}

**return** **0**;

}

## 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个点与多少个已标号的点相邻，每次选择label[i]最大的未标号的点进行标号。

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。

弦图里的团数等于色数，色数（从后往前）和最大独立集（从前往后）都可以按完美消除序列的顺序贪心。

/\*

弦图的完美消除序列

O(mlogn) 可以做到 O(n+m)

\*/

#include <iostream>

**using** **namespace** std;

#define maxn 1005

#define maxm 2000005

**int** head[maxn],heap[maxn],l[maxn],hz,Link[maxn];

**int** vtx[maxm],next[maxm],tot,n,m,A[maxn];

**bool** map[maxn][maxn];

**inline** **void** Add(**int** a,**int** b)

{

vtx[tot]=b;

next[tot]=head[a];

head[a]=tot++;

}

**inline** **void** sink(**int** x)

{

**int** mid=x\***2**;

**while** (mid<=hz)

{

**if** (mid+**1**<=hz && l[heap[mid+**1**]]>l[heap[mid]]) ++mid;

**if** (l[heap[x]]<l[heap[mid]])

{

swap(Link[heap[x]],Link[heap[mid]]);

swap(heap[x],heap[mid]);

}**else** **break**;

x=mid;

mid=x\***2**;

}

}

**inline** **void** up(**int** x)

{

**for** (**int** mid=x/**2**;mid>**0**;mid=x/**2**)

{

**if** (l[heap[mid]]<l[heap[x]])

{

swap(Link[heap[x]],Link[heap[mid]]);

swap(heap[x],heap[mid]);

}**else** **break**;

x=mid;

}

}

**int** main()

{

**for** (;scanf(**"%d%d"**,&n,&m) && (m+n);)

{

tot=**2**;

memset(map,**false**,**sizeof**(map));

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

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

{

**int** a,b;

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

--a;--b;

map[a][b]=map[b][a]=**true**;

Add(a,b);

Add(b,a);

}

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

hz=**0**;

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

{

Link[i]=++hz;

heap[hz]=i;

}

**for** (**int** i=n;i>**0**;--i)

{

**int** v=-**1**;

**int** u=heap[**1**];

//序列的第i项就是u

Link[u]=-**1**;

Link[heap[hz]]=**1**;

heap[**1**]=heap[hz--];

sink(**1**);

**for** (**int** p=head[u];p;p=next[p])

**if** (Link[vtx[p]]!=-**1**)

{

++l[vtx[p]];

up(Link[vtx[p]]);

}**else**

{

**if** (v==-**1**) v=vtx[p];

**else**

{

**if** (!map[v][vtx[p]])

{

printf(**"Imperfect\n"**);

//判定不是弦图

**goto** answer;

}

}

}

}

printf(**"Perfect\n"**);

answer:;

printf(**"\n"**);

}

**return** **0**;

}

## 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();

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

memset(use,**1**,**sizeof**(use));

FindMaxMatching();

k=**0**;

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

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

**return** **0**;

}

## 最小树形图(ElogE+V^2)

**const** **int** N **=** 1111**;**

**const** **int** M **=** 1111111**;**

**int** n**,** m**,** a**,** b**,** c**,** x**[**N**],** y**[**N**],** z**[**N**],**

edgeCnt**,** firstEdge**[**N**],** from**[**M**],** length**[**M**],** nextEdge**[**M**],**

inEdge**[**N**],** key**[**M**],** delta**[**M**],** depth**[**M**],** child**[**M**][**2**],**

parent**[**N**],** choosen**[**N**],** degree**[**N**],** queue**[**N**];**

**void** pass **(int** x**)** **{**

**if** **(**delta**[**x**]** **!=** 0**)** **{**

key**[**child**[**x**][**0**]]** **+=** delta**[**x**];**

delta**[**child**[**x**][**0**]]** **+=** delta**[**x**];**

key**[**child**[**x**][**1**]]** **+=** delta**[**x**];**

delta**[**child**[**x**][**1**]]** **+=** delta**[**x**];**

delta**[**x**]** **=** 0**;**

**}**

**}**

**int** merge **(int** x**,** **int** y**)** **{**

**if** **(**x **==** 0 **or** y **==** 0**)** **{**

**return** x **^** y**;**

**}**

**if** **(**key**[**x**]** **>** key**[**y**])** **{**

swap**(**x**,** y**);**

**}**

pass**(**x**);**

child**[**x**][**1**]** **=** merge**(**child**[**x**][**1**],** y**);**

**if** **(**depth**[**child**[**x**][**0**]]** **<** depth**[**child**[**x**][**1**]])** **{**

swap**(**child**[**x**][**0**],** child**[**x**][**1**]);**

**}**

depth**[**x**]** **=** depth**[**child**[**x**][**1**]]** **+** 1**;**

**return** x**;**

**}**

**void** addEdge **(int** u**,** **int** v**,** **int** w**)** **{**

from**[++** edgeCnt**]** **=** u**;**

length**[**edgeCnt**]** **=** w**;**

nextEdge**[**edgeCnt**]** **=** firstEdge**[**v**];**

firstEdge**[**v**]** **=** edgeCnt**;**

key**[**edgeCnt**]** **=** w**;**

delta**[**edgeCnt**]** **=** 0**;**

depth**[**edgeCnt**]** **=** 0**;**

child**[**edgeCnt**][**0**]** **=** child**[**edgeCnt**][**1**]** **=** 0**;**

inEdge**[**v**]** **=** merge**(**inEdge**[**v**],** edgeCnt**);**

**}**

**void** deleteMin **(int** **&**r**)** **{**

pass**(**r**);**

r **=** merge**(**child**[**r**][**0**],** child**[**r**][**1**]);**

**}**

**int** findRoot **(int** u**)** **{**

**if** **(**parent**[**u**]** **!=** u**)** **{**

parent**[**u**]** **=** findRoot**(**parent**[**u**]);**

**}**

**return** parent**[**u**];**

**}**

**void** clear **()** **{**

edgeCnt **=** 0**;**

depth**[**0**]** **=** **-**1**;**

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

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

**}**

**int** solve **(int** root**)** **{**

**int** result **=** 0**;**

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

parent**[**i**]** **=** i**;**

**}**

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

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

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

**if** **(**i **==** root **or** parent**[**i**]** **!=** i**)** **{**

**continue;**

**}**

**while** **(**findRoot**(**from**[**inEdge**[**i**]])** **==** findRoot**(**i**))** **{**

deleteMin**(**inEdge**[**i**]);**

**}**

choosen**[**i**]** **=** inEdge**[**i**];**

degree**[**findRoot**(**from**[**choosen**[**i**]])]** **+=** 1**;**

**}**

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

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

**if** **(**i **!=** root **and** parent**[**i**]** **==** i **and** degree**[**i**]** **==** 0**)** **{**

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

**}**

**}**

**while** **(**head **<** tail**)** **{**

**if** **(--** degree**[**findRoot**(**from**[**choosen**[**queue**[**head**]]])]** **==** 0**)** **{**

queue**[**tail **++]** **=** findRoot**(**from**[**choosen**[**queue**[**head**]]]);**

**}**

head **+=** 1**;**

**}**

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

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

**if** **(**i **!=** root **and** parent**[**i**]** **==** i **and** degree**[**i**]** **>** 0**)** **{**

found **=** **true;**

**int** j **=** i**,** temp **=** 0**;**

**do{**

j **=** findRoot**(**from**[**choosen**[**j**]]);**

parent**[**j**]** **=** i**;**

deleteMin**(**inEdge**[**j**]);**

result **+=** key**[**choosen**[**j**]];**

key**[**inEdge**[**j**]]** **-=** key**[**choosen**[**j**]];**

delta**[**inEdge**[**j**]]** **-=** key**[**choosen**[**j**]];**

temp **=** merge**(**temp**,** inEdge**[**j**]);**

**}** **while** **(**j **!=** i**);**

inEdge**[**i**]** **=** temp**;**

**}**

**}**

**if** **(not** found**)** **{**

**break;**

**}**

**}**

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

**if** **(**i **!=** root **and** parent**[**i**]** **==** i**)** **{**

result **+=** key**[**choosen**[**i**]];**

**}**

**}**

**return** result**;**

**}**

## 最小树形图（V^3）

**const** **int** maxn=**1100**;

**int** n,m , g[maxn][maxn] , used[maxn] , pass[maxn] , 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(){

freopen(**"input.txt"**,**"r"**,stdin);

freopen(**"output.txt"**,**"w"**,stdout);

**int** i,j,k,test,cases;

cases=**0**;

scanf(**"%d"**,&test);

**while** (test){

test--;

//if (n==0) break;

scanf(**"%d%d"**,&n,&m);

// memset(g,60,sizeof(g));

**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]);

}

cases++;

printf(**"Case #%d: "**,cases);

k=mdst(**1**);

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

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

}

**return** **0**;

}

## Hopcroft

#include <cstdio>

#include <cstring>

#define maxn 50005

#define maxm 150005

**int** cx[maxn],cy[maxn],mk[maxn],q[maxn],src[maxn],pre[maxn];

**int** head[maxn],vtx[maxm],next[maxm],tot,n,m;

**inline** **void** Add(**int** a,**int** b)

{

vtx[tot]=b;

next[tot]=head[a];

head[a]=tot++;

}

**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;

}

**int** main()

{

freopen(**"4206.in"**,**"r"**,stdin);

freopen(**"4206.out"**,**"w"**,stdout);

**int** P;

scanf(**"%d%d%d"**,&n,&m,&P);

tot=**2**;

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

{

**int** a,b;

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

--a;--b;

Add(a,b);

}

printf(**"%d\n"**,Maxmatch());

**return** **0**;

}

## 割点缩块

/\*

考虑割点的无向图缩块

\*/

#include<vector>

#include<cstdio>

#include<cstring>

**using** **namespace** std;

**const** **int** maxn = **100000**+**5**;

**const** **int** maxm = **200000**+**5**;

**int** e[maxm],prev[maxm];

**int** info[maxn];

**int** dfn[maxn],low[maxn],stack[maxn];

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

**int** cntB,cnt,top,tote;

**void** insertE( **int** x,**int** y )

{

++tote; e[tote]=y; prev[tote]=info[x]; info[x]=tote;

}

**void** Min( **int** &x,**int** y )

{

**if**(y < x) x = y;

}

**void** Dfs( **int** x,**int** father )

{

dfn[x] = low[x] = ++cnt;

stack[++top] = x;

**for**(**int** t=info[x];t;t=prev[t])

**if**(dfn[e[t]] == **0**)

{

**int** tmp = top;

Dfs(e[t],x);

Min(low[x],low[e[t]]);

**if**(low[e[t]] >= dfn[x])

{

Block[++cntB].clear();

**for**(**int** k=tmp+**1**;k<=top;++k) Block[cntB].push\_back(stack[k]);

Block[cntB].push\_back(x);

top=tmp;

}

}

**else**

**if**(e[t]!=father)

Min(low[x],dfn[e[t]]);

}

**int** main()

{

**int** n,m;

scanf(**"%d%d"**,&n,&m);

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

tote=**0**;

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

{

**int** x,y;

scanf(**"%d%d"**,&x,&y);

insertE(x,y);

insertE(y,x);

}

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

cnt=top=cntB=**0**;

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

printf(**"%d\n"**,cntB);

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

{

**for**(**int** j=**0**;j<Block[i].size();++j) printf(**"%d "**,Block[i][j]);

puts(**""**);

}

**return** **0**;

}

## 割边缩块

/\*

仅考虑割边的无向图缩块

\*/

#include <cstdio>

#include <cstring>

**const** **int** maxn = **10000**+**5**;

**const** **int** maxm = **200000**+**5**;

**int** color[maxn],low[maxn],stack[maxn],cnt,dep,N,n,m;

**bool** mark[maxn],vis[maxm];

**int** head[maxn],vtx[maxm],next[maxm],tot;

**inline** **int** min(**int** a,**int** b)

{

**if** (a<b) **return** a;

**return** b;

}

**inline** **void** Add(**int** a,**int** b)

{

vtx[tot]=b;

next[tot]=head[a];

head[a]=tot++;

}

**inline** **void** dfs(**int** u)

{

mark[u]=**true**;

low[u]=++cnt;

**int** Min=cnt;

stack[++dep]=u;

**for** (**int** p=head[u];p;p=next[p])

**if** (!vis[p>>**1**])

{

vis[p>>**1**]=**true**;

**if** (!mark[vtx[p]]) dfs(vtx[p]);

Min=min(Min,low[vtx[p]]);

}

**if** (Min==low[u])

{

**int** v;

++N;

**do**

{

v=stack[dep--];

low[v]=n+**1**;

color[v]=N;

}**while** (u!=v);

}**else** low[u]=Min;

}

**int** main()

{

**for** (**int** test=**1**;scanf(**"%d%d"**,&n,&m) && n;++test)

{

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

memset(vis,**false**,**sizeof**(vis));

tot=**2**;

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

{

**int** a,b;

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

Add(a,b);

Add(b,a);

}

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

memset(mark,**false**,**sizeof**(mark));

N=cnt=dep=**0**;

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

**if** (!mark[i])

dfs(i);

// printf(" %d\n",N);

**int** s,t;

scanf(**"%d%d"**,&s,&t);

printf(**"Case %d: "**,test);

**if** (color[s]==color[t]) puts(**"YES"**);

**else** puts(**"NO"**);

}

**return** **0**;

}

# 字符串

## 字符串最小表示

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);

## Manacher-O(n)求每个位置为中心的最长回文串

**void** manacher **(char** str**[],** **int** len**[],** **int** n**)** **{**

len**[**0**]** **=** 1**;**

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

**int** p **=** i **>>** 1**,**

q **=** i **-** p**,**

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

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

**while** **(**p **-** len**[**i**]** **>** **-**1 **and** q **+** len**[**i**]** **<** n **and** str**[**p **-** len**[**i**]]** **==** str**[**q **+** len**[**i**]])** **{**

len**[**i**]** **+=** 1**;**

**}**

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

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

**}**

**}**

**}**

## 多个串求最长连续的子串 —— 后缀数组 o(n)

#define maxn (200010)

**int** suffix[maxn],next[maxn][**26**],len[maxn],nodes;

**int** max[maxn],min[maxn];

**char** str[maxn];

**int** main(){

**int** i,last,p,q,r,c,sh,ans;

gets(str);

last=**1**;

nodes=**2**;

**for**(i=**0**;str[i];i++){

c=str[i]-**'a'**;

p=last;

last=nodes++;

len[last]=len[p]+**1**;

**while**(p&&!next[p][c]){

next[p][c]=last;

p=suffix[p];

}

**if**(!p)

suffix[last]=**1**;

**else** **if**(len[q=next[p][c]]==len[p]+**1**)

suffix[last]=q;

**else**{

r=nodes++;

len[r]=len[p]+**1**;

suffix[r]=suffix[q];

memcpy(next[r],next[q],**sizeof**(next[**0**]));

suffix[last]=suffix[q]=r;

**while**(p&&next[p][c]==q){

next[p][c]=r;

p=suffix[p];

}

}

}

**for**(i=**1**;i<nodes;i++)

min[i]=**0x7fffffff**;

**while**(gets(str)){

**for**(i=**1**;i<nodes;i++)

max[i]=**0**;

sh=**0**;

p=**1**;

**for**(i=**0**;str[i];i++){

c=str[i]-**'a'**;

**while**(p>**1**&&!next[p][c]){

sh=((len[p]+sh)-len[suffix[p]])<?**0**;

p=suffix[p];

}

**if**(next[p][c]){

sh=(len[p]+sh+**1**)-len[next[p][c]];

p=next[p][c];

}

max[p]>?=len[p]+sh;

}

**for**(i=**1**;i<nodes;i++)

min[i]<?=max[i];

}

ans=**0**;

**for**(i=**1**;i<nodes;i++)

ans>?=min[i];

printf(**"%d\n"**,ans);

**return** **0**;

}

## 扩展\_KMp

program peng;

var

s,t:string;

extend,next:array[**1..1000**]of longint;

procedure extendkmp(s,t:string);

var

i,j,k,a,l,len:longint;

lens,lent:longint;

**begin**

{================prepare==============}

lens:=length(s);

lent:=length(t);

s:=s+**'$'**;

t:=t+**'#'**;

j:=**0**;

**while** t[**1**+j]=t[**2**+j] **do** inc(j);

next[**2**]:=j;

a:=**2**;

**for** i:=**3** to lent **do**

**begin**

len:=next[a]-(i-a);

l:=next[i-a+**1**];

**if** l<len then next[i]:=l **else**

**begin**

j:=max(**0**,len);

**while** t[**1**+j]=t[i+j] **do** inc(j);

next[i]:=j;

a:=i;

**end**;

**end**;

{===================main================}

j:=**0**;

**while** s[**1**+j]=t[**1**+j] **do** inc(j);

extend[**1**]:=j;

a:=**1**;

**for** i:=**2** to lens **do**

**begin**

len:=extend[a]-(i-a);

l:=next[i-a+**1**];

**if** l<len then extend[i]:=l

**else** **begin**

j:=max(**0**,len);

**while** s[i+j]=t[**1**+j] **do** inc(j);

extend[i]:=j;

a:=i;

**end**;

**end**;

**for** i:=**1** to lens **do**

write(extend[i],**' '**);

**end**;

## 最多的重复字串（optional）

**const** **int** maxn=**110000**;

**char** s[maxn],ans\_s[maxn],a[maxn],b[maxn];

**int** n;

**int** start[maxn],next[maxn] , extended\_l[maxn] , extended\_r[maxn];

**int** ans;

**void** prepare\_extended\_kmp(**char** s[],**char** t[],**int** extended[],**int** s1,**int** t1,**int** limit){

**int** i,j,a,len,l;

**char** k1;

k1=s[limit+**1**]; s[limit+**1**]=**'#'**;

len=**0**;

**while** (t[t1+len]==t[t1+**1**+len]) len++;

next[t1+**1**]=len;

a=t1+**1**;

**foru**(i,t1+**2**,limit){

len=a+next[a] - i;

l=next[i-a+**1** +t1-**1**];

**if** (l<len) next[i]=l;

**else** {

j=max(**0**,len);

**while** (t[t1+j] == t[i+j]) j++;

next[i]=j;

a=i;

}

}

**if** (s1!=t1){

len=**0**;

**while** (s[s1+len]==t[t1+len]) len++;

extended[s1]=len;

a=s1;

}

**foru**(i,s1+**1**,t1-**1**){

len=a+extended[a] - i;

l=next[i-a+**1** +t1-**1**];

**if** (l<len) extended[i]=l;

**else**{

j=max(**0**,len);

**while** (t[t1+j] == s[i+j]) j++;

extended[i]=j;

a=i;

}

}

**foru**(i,t1+**1**,limit) extended[i]=next[i];

s[limit+**1**]=k1;

}

**bool** check(**int** start,**int** len){

**int** m1,i;

m1=strlen(ans\_s);

i=**0**;

**while** (i<m1 && i<len) {

**if** (a[start+i]!=ans\_s[i]) **return** a[start+i]<ans\_s[i];

i++;

}

**if** (len<m1) **return** **true**; **else** **return** **false**;

}

**void** push\_s(**int** start,**int** len){

**int** i;

**rep**(i,len) ans\_s[i]=a[start+i];

ans\_s[len]=**0**;

}

**int** make\_small(**int** s ,**int** t, **int** len , **int** num){

**int** i,j,k,u,n;

n=t-s+**1**;

j=s;

**bool** flag;

**foru**(i,**1**,n-len) {

k=s+i;

flag=**false**;

**rep**(u,num) **if** (a[j+u]!=a[k+u])

**if** (a[j+u]>a[k+u]) { flag=**true**; **break**;}

**else** { flag=**false**; **break**;}

**if** (flag) j=k;

}

**return** j;

}

**void** make\_ans(**int** left,**int** right){

**int** i,j,k,mid;

**if** (left>=right) **return**;

mid=(left+right)/**2**;

prepare\_extended\_kmp(a , a , extended\_r , left , mid , right);

prepare\_extended\_kmp(b , b , extended\_l , start[right] , start[mid] , start[left]);

//{====left=======}

**int** len,st;

**foru**(i,left,mid-**1**)

**if** (i+extended\_r[i]-**1**>=mid-**1**) {

len=extended\_r[i]+extended\_l[start[i]] + (mid-i) - **1**;

//len=extended\_r[i] + (mid-i) ;

k=len / (mid-i);

len=k\*(mid-i);

st=make\_small(i-extended\_l[start[i]]+**1**,mid+extended\_r[i]-**1**,len,k);

**if** (k>ans) {

ans=k;

push\_s(st, len);

// printf("%s\n",ans\_s);

}

**else**

**if** (k==ans)

**if** (check(st,len))

push\_s(st,len);

// printf("%s\n",ans\_s);

}

// {====right=======}

**ford**(i,right,mid+**1**)

**if** (i-extended\_l[start[i]]+**1**<=mid+**1**){

len=extended\_r[i]+extended\_l[start[i]] + (i-mid) - **1**;

//len=extended\_l[start[i]] + (i-mid) ;

k=len / (i-mid);

len=k\*(i-mid);

st=make\_small(mid-extended\_l[start[i]]+**1**,i+extended\_r[i]-**1**,len,k);

**if** (k>ans) {

ans=k;

push\_s(st, len);

// printf("%s\n",ans\_s);

}

**else**

**if** (k==ans)

**if** (check(st,len))

push\_s(st,len);

}

make\_ans(left,mid-**1**);

make\_ans(mid+**1**,right);

}

**void** work(){

**int** i,j,k;

**foru**(i,**1**,n) {

a[i]=s[i];

b[i]=s[n-i+**1**];

start[n-i+**1**]=i;

}

ans=**1**;

**rep**(i,n) ans\_s[i]=a[i+**1**];

ans\_s[n]=**0**;

make\_ans(**1**,n);

printf(**"%s\n"**,ans\_s);

}

**int** main(){

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

**while** (**1**){

scanf(**"%s"**,s);

**if** (s[**0**]==**'#'**) **break**;

test++;

printf(**"Case %d: "**,test);

n=strlen(s);

**ford**(i,n,**1**) s[i]=s[i-**1**];

work();

}

**return** **0**;

}

## 后缀自动机

**const** size\_t MAXL **=** 100005**;**

**const** size\_t POOLSIZE **=** MAXL **\*** 2**;**

**inline** **void** remax**(int** **&**a**,** **int** b**)** **{**

**if** **(**b **>** a**)**

a **=** b**;**

**}**

**inline** **void** remin**(int** **&**a**,** **int** b**)** **{**

**if** **(**b **<** a**)**

a **=** b**;**

**}**

**class** SuffixAutoman **{**

**public:**

**typedef** **char\*** ptr**;**

**struct** State **{**

**int** length**;**

State**\*** failure**;**

State**\*** transition**[**26**];**

**int** l**[**9**];**

**bool** inner**;**

**};**

**int** size**;**

State**\*** empty**;**

State**\*** last**;**

**static** State statePool**[**POOLSIZE**];**

**inline** State**\*** create**()** **{**

**return** **&**statePool**[**size**++];**

**}**

**void** build**(**ptr str**,** **int** len**)** **{**

size **=** 0**;**

empty **=** last **=** create**();**

last**->**length **=** 0**;**

last**->**failure **=** 0**;**

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

last **=** extend**(**last**,** str**[**i**]** **-** 'a'**);**

**}**

State**\*** extend**(**State**\*** p**,** **int** a**)** **{**

State **\***np **=** create**(),** **\***nq**,** **\***q**;**

np**->**length **=** p**->**length **+** 1**;**

**while** **(**p **&&** **!**p**->**transition**[**a**])**

p**->**transition**[**a**]** **=** np**,** p **=** p**->**failure**;**

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

np**->**failure **=** empty**;**

**else** **{**

q **=** p**->**transition**[**a**];**

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

np**->**failure **=** q**;**

**else** **{**

nq **=** create**();**

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

nq**->**length **=** p**->**length **+** 1**;**

nq**->**failure **=** q**->**failure**;**

q**->**failure **=** nq**;**

np**->**failure **=** nq**;**

**while** **(**p **&&** p**->**transition**[**a**]** **==** q**)**

p**->**transition**[**a**]** **=** nq**,** p **=** p**->**failure**;**

**}**

**}**

**return** np**;**

**}**

**void** match**(**ptr str**,** **int** len**,** **int** index**)** **{**

State**\*** p **=** empty**;**

**for** **(int** i **=** 0**,** l **=** 0**,** a**;** i **<** len**;** i**++)** **{**

a **=** str**[**i**]** **-** 'a'**;**

**while** **(**p **!=** empty **&&** **!**p**->**transition**[**a**])** **{**

p **=** p**->**failure**;**

l **=** p**->**length**;**

**}**

**if** **(**p **==** empty**)**

l **=** 0**;**

p **=** p**->**transition**[**a**];**

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

p **=** empty**;**

**else**

remax**(**p**->**l**[**index**],** **++**l**);**

**}**

**}**

**int** getLCS**(int** n**)** **{**

**int** fsize **=** 0**,** lcs **=** 0**;**

**static** State**\*** final**[**MAXL**];**

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

statePool**[**i**].**failure**->**inner **=** **true;**

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

**if** **(!**statePool**[**i**].**inner**)**

final**[**fsize**++]** **=** **&**statePool**[**i**];**

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

**for** **(**State**\*** p **=** final**[**i**];** p**->**failure**;** p **=** p**->**failure**)**

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

remax**(**p**->**failure**->**l**[**j**],** p**->**failure**->**length **<** p**->**l**[**j**]** **?** p**->**failure**->**length **:** p**->**l**[**j**]);**

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

**int** l **=** statePool**[**i**].**length**;**

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

remin**(**l**,** statePool**[**i**].**l**[**j**]);**

remax**(**lcs**,** l**);**

**}**

**return** lcs**;**

**}**

**};**

SuffixAutoman**::**State SuffixAutoman**::**statePool**[**POOLSIZE**];**

SuffixAutoman sa**;**

**int** n**;**

**char** str**[**MAXL**];**

**int** main**()** **{**

gets**(**str**);**

n **=** 0**;**

sa**.**build**(**str**,** strlen**(**str**));**

**while** **(**gets**(**str**))**

sa**.**match**(**str**,** strlen**(**str**),** n**++);**

printf**(**"%d\n"**,** sa**.**getLCS**(**n**));**

**return** 0**;**

**}**

## dc3

//DC3 待排序的字符串放在r 数组中，从r[0]到r[n-1]，长度为n，且最大值小于m。

//约定除r[n-1]外所有的r[i]都大于0, r[n-1]=0。

//函数结束后，结果放在sa 数组中，从sa[0]到sa[n-1]。

**//r必须开长度乘3**

#define maxn 10000

#define F(x) ((x)/3+((x)%3==1?0:tb))

#define G(x) ((x)<tb?(x)\*3+1:((x)-tb)\*3+2)

**int** wa[maxn],wb[maxn],wv[maxn],wss[maxn];

**int** s[maxn\***3**],sa[maxn\***3**];

**int** c0(**int** \*r,**int** a,**int** b)

{

**return** r[a]==r[b]&&r[a+**1**]==r[b+**1**]&&r[a+**2**]==r[b+**2**];

}

**int** c12(**int** k,**int** \*r,**int** a,**int** b)

{

**if**(k==**2**) **return** r[a]<r[b]||r[a]==r[b]&&c12(**1**,r,a+**1**,b+**1**);

**else** **return** r[a]<r[b]||r[a]==r[b]&&wv[a+**1**]<wv[b+**1**];

}

**void** sort(**int** \*r,**int** \*a,**int** \*b,**int** n,**int** m)

{

**int** i;

**for**(i=**0**;i<n;i++) wv[i]=r[a[i]];

**for**(i=**0**;i<m;i++) wss[i]=**0**;

**for**(i=**0**;i<n;i++) wss[wv[i]]++;

**for**(i=**1**;i<m;i++) wss[i]+=wss[i-**1**];

**for**(i=n-**1**;i>=**0**;i--) b[--wss[wv[i]]]=a[i];

}

**void** dc3(**int** \*r,**int** \*sa,**int** n,**int** m)

{

**int** i,j,\*rn=r+n,\*san=sa+n,ta=**0**,tb=(n+**1**)/**3**,tbc=**0**,p;

r[n]=r[n+**1**]=**0**;

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

**if**(i%**3**!=**0**) wa[tbc++]=i;

sort(r+**2**,wa,wb,tbc,m);

sort(r+**1**,wb,wa,tbc,m);

sort(r,wa,wb,tbc,m);

**for**(p=**1**,rn[F(wb[**0**])]=**0**,i=**1**;i<tbc;i++)

rn[F(wb[i])]=c0(r,wb[i-**1**],wb[i])?p-**1**:p++;

**if** (p<tbc) dc3(rn,san,tbc,p);

**else** **for** (i=**0**;i<tbc;i++) san[rn[i]]=i;

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

**if**(san[i]<tb) wb[ta++]=san[i]\***3**;

**if**(n%**3**==**1**) wb[ta++]=n-**1**;

sort(r,wb,wa,ta,m);

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

wv[wb[i]=G(san[i])]=i;

**for**(i=**0**,j=**0**,p=**0**;i<ta && j<tbc;p++)

sa[p]=c12(wb[j]%**3**,r,wa[i],wb[j])?wa[i++]:wb[j++];

**for**(;i<ta;p++) sa[p]=wa[i++];

**for**(;j<tbc;p++) sa[p]=wb[j++];

}

**int** main(){

**int** n,m=**0**;

scanf(**"%d"**,&n);

**for** (**int** i=**0**;i<n;i++) scanf(**"%d"**,&s[i]),s[i]++,m=max(s[i]+**1**,m);

printf(**"%d\n"**,m);

s[n++]=**0**;

dc3(s,sa,n,m);

**for** (**int** i=**0**;i<n;i++) printf(**"%d "**,sa[i]);printf(**"\n"**);

}

# 杂

## 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;

}

}

**void** print(){

printf(**"%d\n"**, 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。

【代码】

**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**);

}

## Farmland

**const** **int** mx = **210**;

**const** **double** eps = **1e**-**8**;

**struct** TPoint { **double** x, y;} p[mx];

**struct** TNode { **int** n, e[mx];} a[mx];

**bool** visit[mx][mx], valid[mx];

**int** l[mx\*mx][**2**], n, m, tp, ans, now, test;

**double** area;

**int** dcmp(**double** x) { **return** x < eps ? -**1** : x > eps; }

**int** cmp(**int** a, **int** b){

**return** dcmp(atan2(p[a].y - p[now].y, p[a].x - p[now].x) - atan2(p[b].y - p[now].y, p[b].x - p[now].x)) < **0**;

}

**double** cross(**const** TPoint&a, **const** TPoint&b){ **return** a.x \* b.y - b.x \* a.y;}

**void** init();

**void** work();

**bool** check(**int**, **int**);

**int** main()

{

scanf(**"%d"**, &test);

**while**(test--) {

init();

work();

}

**return** **0**;

}

**void** init()

{

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

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

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

scanf(**"%d"**, &n);

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

scanf(**"%d"**, &a[i].n);

scanf(**"%lf%lf"**, &p[i].x, &p[i].y);

scanf(**"%d"**, &a[i].n);

**for**(**int** j = **0**; j < a[i].n; j++) {

scanf(**"%d"**, &a[i].e[j]);

a[i].e[j]--;

}

}

scanf(**"%d"**, &m);

**for**(now = **0**; now < n; now++) sort(a[now].e, a[now].e + a[now].n, cmp);

}

**void** work()

{

ans = **0**;

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

**for**(**int** j = **0**; j < a[i].n; j++) **if**(!visit[i][a[i].e[j]])

**if**(check(i, a[i].e[j])) ans++;

printf(**"%d\n"**, ans);

}

**bool** check(**int** b1, **int** b2)

{

area = **0**;

l[**0**][**0**] = b1;

l[**0**][**1**] = b2;

**for**(tp = **1**; ; tp++) {

visit[l[tp - **1**][**0**]][l[tp - **1**][**1**]] = **1**;

area += cross(p[l[tp - **1**][**0**]], p[l[tp - **1**][**1**]]);

**int** k, r(l[tp][**0**] = l[tp - **1**][**1**]);

**for**(k = **0**; k < a[r].n; k++) **if**(a[r].e[k] == l[tp - **1**][**0**]) **break**;

l[tp][**1**] = a[r].e[(k + a[r].n - **1**) % a[r].n];

**if**(l[tp][**0**] == b1 && l[tp][**1**] == b2) **break**;

}

**if**(dcmp(area) < **0** || tp < **3** || tp != m) **return** **0**;

fill\_n(valid, n, **0**);

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

**if**(valid[l[i][**0**]]) **return** **0**;

valid[l[i][**0**]] = **1**;

}

**return** **1**;

}

## FFT（crazyb0y）

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

**const** **int** maxn **=** 1 **<<** 18**;**

**struct** Complex **{**

**double** x**,** y**;**

Complex **(double** real **=** 0**,** **double** imag **=** 0**)** **:** x**(**real**),** y**(**imag**)** **{}**

**double** **&**real**()** **{**

**return** x**;**

**}**

**double** **&**imag**()** **{**

**return** y**;**

**}**

**};**

Complex **operator+(const** Complex **&**a**,** **const** Complex **&**b**)** **{**

**return** Complex**(**a**.**x **+** b**.**x**,** a**.**y **+** b**.**y**);**

**}**

Complex **operator-(const** Complex **&**a**,** **const** Complex **&**b**)** **{**

**return** Complex**(**a**.**x **-** b**.**x**,** a**.**y **-** b**.**y**);**

**}**

Complex **operator\*(const** Complex **&**a**,** **const** Complex **&**b**)** **{**

**return** Complex**(**a**.**x **\*** b**.**x **-** a**.**y **\*** b**.**y**,** a**.**x **\*** b**.**y **+** a**.**y **\*** b**.**x**);**

**}**

**void** build**(**Complex \_P**[],** Complex P**[],** **int** n**,** **int** m**,** **int** curr**,** **int** **&**cnt**)**

**{**

**if** **(**m **==** n**)**

\_P**[**curr**]** **=** P**[**cnt**++];**

**else** **{**

build**(**\_P**,** P**,** n**,** m **\*** 2**,** curr**,** cnt**);**

build**(**\_P**,** P**,** n**,** m **\*** 2**,** curr **+** m**,** cnt**);**

**}**

**}**

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

**{**

**static** Complex \_P**[**maxn**];**

**int** cnt **=** 0**;**

build**(**\_P**,** P**,** n**,** 1**,** 0**,** cnt**);**

copy**(**\_P**,** \_P **+** n**,** P**);**

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

**int** m **=** 1 **<<** d**;**

**int** m2 **=** m **\*** 2**;**

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

Complex unit\_p0 **=** Complex**(**cos**(**p0**),** sin**(**p0**));**

**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**;**

**}**

**}**

**}**

**}**

## fft——速度一般

**const** **int** maxn=**130000**+**10**;

**int** Z[maxn],X[maxn],Y[maxn],res[maxn];

**int** B[**18**][maxn];

**void** add(**int** n ,**int** x[], **int** y[] , **bool** flag) {

**int** i;

**rep**(i,n) x[i]+=flag?y[i] : - y[i];

}

**void** calc(**int** dep, **int** n , **int** x[], **int** y[] ,**int** res[]) {

**if** (n<=**100**) {

**int** i,j;

**rep**(i,n)

**rep**(j,n) res[i+j]+=x[i]\*y[j];

**return**;

}

**int** i;

**int** m=n/**2**;

**rep**(i, (n-m)\***2**-**1**) B[dep][i]=**0**;

calc(dep+**1**,n-m, x+m, y+m , B[dep]);

add( (n-m)\***2**-**1**, res + m\***2** , B[dep],**true**);

add( (n-m)\***2**-**1**, res + m , B[dep],**false**);

**rep**(i, m\***2**-**1**) B[dep][i]=**0**;

calc(dep+**1**,m,x,y,B[dep]);

add(m\***2**-**1**,res,B[dep],**true**);

add(m\***2**-**1**,res+m,B[dep],**false**);

add(m,x+m,x,**true**);

add(m,y+m,y,**true**);

calc(dep+**1**,n-m,x+m,y+m,res+m);

add(m,x+m,x,**false**);

add(m,y+m,y,**false**);

}

**class** CircularShifts{

**public**:

**int** maxScore(**int** N, **int** Z0, **int** A, **int** B, **int** M)

{

Z[**0**]=Z0%M;

**int** i,j,k;

**foru**(i,**1**,N\***2**) Z[i]=((**long** **long** ) Z[i-**1**]\*A + B) %M;

**rep**(i,N) X[i]=Z[i]%**100**;

**rep**(i,N) Y[i]=Z[i+N]%**100**;

**rep**(i,N>>**1**) swap(Y[i],Y[N-**1**-i]);

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

calc(**0**,N,X,Y,res);

**int** Max=**0**;

**rep**(i,N) Max=max(Max,res[i]+res[i+N]);

**return** Max;

}

}

## FFt\_speed

**typedef** **long** **long** int64;

#define two(X) (1<<(X))

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

**template**<**class** T> **inline** T lowbit(T n){**return** (n^(n-**1**))&n;}

**class** complex

{

**public**:

**double** a,b;

complex(){};

complex(**double** \_a,**double** \_b) {a=\_a;b=\_b;}

};

**const** **int** maxn=two(**19**)+**5**;

**int** L1,L2;

**int** s1[maxn],s2[maxn];

**int** n,id;

**int** A[maxn];

complex tmp[maxn],P[maxn],PB[maxn];

**int** lowbit(**int** n)

{

**return** (n^(n-**1**))&n;

}

**int** getnumber(**int** s[],**int** L,**int** id)

{

**if** (id>L)

**return** **0**;

**return** s[L-id]-**48**;

}

**void** Fill(**int** s[],**int** L,**int** m,**int** d)

{

**if** (m==n)

P[d]=complex(s[id++],**0**);

**else**

{

Fill(s,L,m\***2**,d);

Fill(s,L,m\***2**,d+m);

}

}

**void** Fill2(**int** m,**int** d)

{

**if** (m==n)

P[d]=tmp[id++];

**else**

{

Fill2(m\***2**,d);

Fill2(m\***2**,d+m);

}

}

**void** FFT(**int** oper)

{

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

{

**int** i,m=(**1**<<d);

**double** p0=**2**\*pi/**double**(m\***2**)\***double**(oper);

**double** sinp0=sin(p0);

**double** cosp0=cos(p0);

**for** (i=**0**;i<n;i+=(m\***2**))

{

**double** sinp=**0**;

**double** cosp=**1**;

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

{

**double** ta=cosp\*P[i+j+m].a-sinp\*P[i+j+m].b;

**double** tb=cosp\*P[i+j+m].b+sinp\*P[i+j+m].a;

P[i+j+m].a=P[i+j].a-ta;

P[i+j+m].b=P[i+j].b-tb;

P[i+j].a+=ta;

P[i+j].b+=tb;

**double** tsinp=sinp;

sinp=sinp\*cosp0+ cosp\*sinp0;

cosp=cosp\*cosp0-tsinp\*sinp0;

}

}

}

}

**class** CircularShifts

{

**public**:

**int** Z[maxn];

**int** maxScore(**int** L, **int** Z0, **int** A, **int** B, **int** M)

{

Z[**0**]=Z0%M;

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

Z[i]=(**int**)(((int64)Z[i-**1**]\*(int64)A+(int64)B)%M);

memset(s1,**0**,**sizeof**(s1));

memset(s2,**0**,**sizeof**(s2));

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

{

s1[i+L]=s1[i]=Z[i]%**100**;

s2[L-**1**-i]=Z[i+L]%**100**;

} **//s1[0]和s2[0]是两个高精度数的最低位**

n=L+L; **//n=LenA+LenB**

**for** (;n!=lowbit(n);n+=lowbit(n));

id=**0**;

Fill(s1,L,**1**,**0**);

FFT(**1**);

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

PB[i]=P[i];

id=**0**;

Fill(s2,L,**1**,**0**);

FFT(**1**);

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

{

tmp[i].a=P[i].a\*PB[i].a-P[i].b\*PB[i].b;

tmp[i].b=P[i].a\*PB[i].b+P[i].b\*PB[i].a;

}

id=**0**;

Fill2(**1**,**0**);

FFT(-**1**);

**double** result=-**1e100**;

**for** (**int** i=L-**1**;i<L+L-**1**;i++)

{

**double** t=P[i].a/(**double**)(n);

**if** (t>result)

result=t;

}

**return** (**int**)(result+**0.5**); **//这里需要分正负考虑取floor**

}

};

**int** main()

{

//这个程序中没有出现小写的L。

//这个程序是求s1[]\*s2[]平移后的矩阵的。倍长了各自的长度后，只需要截取中间的一段即可。

}

## Romberg

#include<vector>

#include<cmath>

**template**<**class** T>

**double** romberg(**const** T&f,**double** a,**double** b,**double** eps=**1e**-**8**){

std::vector<**double**>t;

**double** h=b-a,last,curr;

**int** k=**1**,i=**1**;

t.push\_back(h\*(f(a)+f(b))/**2**); // 梯形

**do**{

last=t.back();

curr=**0**;

**double** x=a+h/**2**;

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

curr+=f(x);

x+=h;

}

curr=(t[**0**]+h\*curr)/**2**;

**double** k1=**4.0**/**3.0**,k2=**1.0**/**3.0**;

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

**double** temp=k1\*curr-k2\*t[j];

t[j]=curr;

curr=temp;

k2/=**4**\*k1-k2; // 防止溢出

k1=k2+**1**;

}

t.push\_back(curr);

k\*=**2**;

h/=**2**;

i++;

}**while**(std::fabs(last-curr)>eps);

**return** t.back();

}

**template**<**class** T>

**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**;

}

#include<cstdio>

**double** test(**double** x){

**if**(x==**0**)**return** **1**;

**else** **return** sin(x)/x;

}

**int** main(){

printf(**"%lf\n"**,romberg(test,**0**,**1**));

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

}

## 多项式求根（求导二分）（optional）

**const** **double** error=**1e**-**12**;

**const** **double** infi=**1e**+**12**;

**double** a[**10**],x[**10**];

**int** n;

**int** sign(**double** x) {

**return** (x<-error)?(-**1**):(x>error);

}

**double** f(**double** a[],**int** n,**double** x) {

**double** tmp=**1**,sum=**0**;

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

sum=sum+a[i]\*tmp;

tmp=tmp\*x;

}

**return** sum;

}

**double** binary(**double** l,**double** r,**double** a[],**int** n) {

**int** sl=sign(f(a,n,l)),sr=sign(f(a,n,r));

**if** (sl==**0**) **return** l;

**if** (sr==**0**) **return** r;

**if** (sl\*sr>**0**) **return** infi;

**while** (r-l>error) {

**double** mid=(l+r)/**2**;

**int** ss=sign(f(a,n,mid));

**if** (ss==**0**) **return** mid;

**if** (ss\*sl>**0**) l=mid; **else** r=mid;

}

**return** l;

}

**void** solve(**int** n,**double** a[],**double** x[],**int** &nx) {

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

x[**1**]=-a[**0**]/a[**1**];

nx=**1**;

**return**;

}

**double** da[**10**],dx[**10**];

**int** ndx;

**for** (**int** i=n;i>=**1**;i--) da[i-**1**]=a[i]\*i;

solve(n-**1**,da,dx,ndx);

nx=**0**;

**if** (ndx==**0**) {

**double** tmp=binary(-infi,infi,a,n);

**if** (tmp<infi) x[++nx]=tmp;

**return**;

}

**double** tmp;

tmp=binary(-infi,dx[**1**],a,n);

**if** (tmp<infi) x[++nx]=tmp;

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

tmp=binary(dx[i],dx[i+**1**],a,n);

**if** (tmp<infi) x[++nx]=tmp;

}

tmp=binary(dx[ndx],infi,a,n);

**if** (tmp<infi) x[++nx]=tmp;

}

**int** main() {

scanf(**"%d"**,&n);

**for** (**int** i=n;i>=**0**;i--) scanf(**"%lf"**,&a[i]);

**int** nx;

solve(n,a,x,nx);

**for** (**int** i=**1**;i<=nx;i++) printf(**"%0.6lf\n"**,x[i]);

**return** **0**;

}

## 强连通分量（一遍dfs)

**int** deep,p,n,m;

**bool** inner[maxn];

**int** st[maxn];

**int** dfn[maxn],low[maxn];

**int** col[maxn];

**int** tobo[maxn],total;

**void** out(**int** v) {

tot++;

st[p+**1**]=-**1**;

**while** (st[p+**1**]!=v) {

col[st[p]]=tot;

inner[st[p]]=**false**;

total++;

tobo[total]=st[p];

p--;

}

}

**void** search(**int** v){

**int** j;

deep++;

dfn[v]=deep;

low[v]=deep;

p++;

st[p]=v;

inner[v]=**true**;

j=**0**;

j=d[v];

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

**if** (dfn[e[j]]==**0**) {

search(e[j]);

low[v]=min(low[v],low[e[j]]);

}

**else** {

**if** (dfn[e[j]]<dfn[v] && inner[e[j]]) low[v]=min(low[v],dfn[e[j]]);

}

j=next[j];

}

**if** (low[v]==dfn[v]) out(v);

}

**void** work\_graph() {

**int** x;

tot=**0**;

deep=**0**; p=**0**;

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

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

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

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

total=**0**;

**foru**(x,**1**,n) **if** (dfn[x]==**0**) search(x);

}

//========tobo他的逆序就是拓扑序，如果是两遍dfs，那么标号正序就是拓扑序

## 求区间第K大数\_不改变值的

**const** **int** D **=** 18**;**

**const** **int** N **=** 100000**;**

**int** n**,** value**[**N**],** rank**[**N**],** order**[**D**][**N**],** pos**[**D**][**N**];**

**long** **long** sum**[**D**][**N**];**

pair **<int,** **int>** backup**[**N**];**

**void** build **(int** d**,** **int** l**,** **int** r**)** **{**

**if** **(**r **-** l **>** 1**)** **{**

**int** m **=** **(**l **+** r**)** **>>** 1**,**

curLeft **=** l**,**

curRight **=** m**;**

**for** **(int** i **=** l**;** i **<** r**;** **++** i**)** **{**

**if** **(**rank**[**order**[**d**][**i**]]** **<** m**)** **{**

order**[**d **+** 1**][**curLeft **++]** **=** order**[**d**][**i**];**

**}else{**

order**[**d **+** 1**][**curRight **++]** **=** order**[**d**][**i**];**

**}**

pos**[**d**][**i**]** **=** curLeft**;**

**}**

build**(**d **+** 1**,** l**,** m**);**

build**(**d **+** 1**,** m**,** r**);**

**}**

sum**[**d**][**r **-** 1**]** **=** value**[**order**[**d**][**r **-** 1**]];**

**for** **(int** i **=** r **-** 2**;** i **>=** l**;** **--** i**)** **{**

sum**[**d**][**i**]** **=** value**[**order**[**d**][**i**]]** **+** sum**[**d**][**i **+** 1**];**

**}**

**}**

// [l, r) [a, b) k-th sum

**long** **long** query **(int** d**,** **int** l**,** **int** r**,** **int** a**,** **int** b**,** **int** k**)** **{**

**if** **(**k**)** **{**

**if** **(**r **-** l **==** 1**)** **{**

**return** sum**[**d**][**a**];**

**}**

**int** m **=** **(**l **+** r**)** **>>** 1**,**

posBegin **=** pos**[**d**][**a**];**

**if** **(**rank**[**order**[**d**][**a**]]** **<** m**)** **{**

posBegin **-=** 1**;**

**}**

**int** posEnd **=** pos**[**d**][**b **-** 1**],**

posCnt **=** posEnd **-** posBegin**;**

**if** **(**k **<** posCnt**)** **{**

**return** query**(**d **+** 1**,** l**,** m**,** posBegin**,** pos**[**d**][**b **-** 1**],** k**);**

**}**

#define RIGHT(i) m + i + 1 - pos[d][i]

**int** rightBegin **=** RIGHT**(**a**);**

**if** **(**rank**[**order**[**d**][**a**]]** **>=** m**)** **{**

rightBegin **-=** 1**;**

**}**

**long** **long** result **=** **(**posBegin **<** m**?** sum**[**d **+** 1**][**posBegin**]:** 0**)** **-** **(**posEnd **<** m**?** sum**[**d **+** 1**][**posEnd**]:** 0**);**

result **+=** query**(**d **+** 1**,** m**,** r**,** rightBegin**,** RIGHT**(**b **-** 1**),** k **-** posCnt**);**

#undef RIGHT

**return** result**;**

**}**

**return** 0**;**

**}**

**void** clear **()** **{**

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

order**[**0**][**i**]** **=** i**;**

**}**

build**(**0**,** 0**,** n**);**

**}**

**int** main**(){**

std**::**ios**::**sync\_with\_stdio**(false);**

**int** testCount**;**

scanf**(**"%d"**,** **&**testCount**);**

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

std**::**cout **<<** "Case #" **<<** t **<<** ":\n"**;**

scanf**(**"%d"**,** **&**n**);**

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

scanf**(**"%d"**,** value **+** i**);**

backup**[**i**]** **=** std**::**make\_pair**(**value**[**i**],** i**);**

**}**

std**::**sort**(**backup**,** backup **+** n**);**

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

rank**[**backup**[**i**].**second**]** **=** i**;**

**}**

**int** m**;**

scanf**(**"%d"**,** **&**m**);**

**while(**m **--){**

**int** a**,** b**;**

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

b **++;**

**int** length **=** b **-** a**;**

**long** **long** result **=** sum**[**0**][**a**]** **-** **(**b **<** n**?** sum**[**0**][**b**]:** 0**);**

result **-=** query**(**0**,** 0**,** n**,** a**,** b**,** **(**length **+** 1**)** **>>** 1**);**

result **-=** query**(**0**,** 0**,** n**,** a**,** b**,** length **>>** 1**);**

std**::**cout **<<** result **<<** "\n"**;**

**}**

std**::**cout **<<** "\n"**;**

**}**

**return** 0**;**

**}**

## 任意两点间的第K短路，可重复走

program peng;

**const**

maxn = **100**;

maxk = **101**;

none = **1000000000**;

type integer = longint;

Tdata = object

answer: array[**1** .. maxn, **1** .. maxk] of integer;

procedure calc(s: integer);

**end**;

var cost: array[**1** .. maxn, **1** .. maxn] of integer;

used: array[**1** .. maxn, **1** .. maxn] of integer;

data: array[**1** .. maxn] of Tdata;

dep, dis, pre: array[**1** .. maxn] of integer;

n, m: integer;

procedure init;

var u, v, l, i: integer;

**begin**

read(n, m);

**for** u := **1** to n **do** **for** v := **1** to n **do** cost[u, v] := none;

**for** i := **1** to m **do** **begin**

read(u, v, l);

cost[u, v] := l;

**end**;

**end**;

procedure Tdata.calc(s: integer);

var i, j, k, p, now, opt: integer;

**begin**

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

fillchar(pre,**sizeof**(pre),**0**);

**for** i:=**1** to n **do**

**begin**

dep[i]:=**1**; pre[i]:=s;

dis[i]:=cost[s,i];

used[i,i]:=**2**\*n;

**end**;

dis[s] := **0**;

**for** now:=**1** to n\*maxk + **1** **do**

**begin**

k:=**0**;

**for** i:=**1** to n **do**

**if** (dep[i]<=maxk)**and**( (k=**0**) **or** (dis[i]<dis[k]) ) then k:=i;

**for** i:=**1** to n **do**

**if** (used[i,k] < dep[k]) **and** (dis[k] + cost[k,i] < dis[i]) then

**begin**

dis[i]:=dis[k]+cost[k,i];

pre[i]:=k;

**end**;

**if** dep[k] > **0** then answer[k, dep[k]] := dis[k];

inc(used[k, pre[k]]); inc(dep[k]);

opt:=none; p:=k;

**for** i:=**1** to n **do**

**begin**

j:=used[k,i] + **1**; {===chose used[k,i]+**1** shortest road to add=====}

**if** (j<dep[i]) **and** (j<=maxk) **and** (answer[i,j] + cost[i,k] < opt) then

**begin**

opt:=answer[i,j]+cost[i,k];

p:=i;

**end**;

**if** (j=dep[i]) **and** (dep[i]<=maxk)**and**(dis[i] + cost[i,k] < opt) then

**begin**

opt:=dis[i] + cost[i,k];

p:=i;

**end**;

**end**;

pre[k] := p; dis[k] := opt;

**end**;

**end**;

procedure main;

var i, u, v, k, task: integer;

**begin**

**for** i := **1** to n **do** data[i].calc(i);

read(task);

**for** i:=**1** to task **do**

**begin**

read(u,v,k);

**if** u=v then inc(k);

**if** data[u].answer[v,k]<none then writeln(data[u].answer[v,k])

**else** writeln(**'-1'**);

**end**;

**end**;

**begin**

init;

main;

**end**.

## 长方体表面两点最短距离

**int** r;

**void** turn(**int** i, **int** j, **int** x, **int** y, **int** z,**int** x0, **int** y0, **int** L, **int** W, **int** H) {

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

**int** R = x\*x+y\*y;

**if** (R<r) r=R;

}

**else**{

**if**(i>=**0** && i< **2**)

turn(i+**1**, j, x0+L+z, y, x0+L-x, x0+L, y0, H, W, L);

**if**(j>=**0** && j< **2**)

turn(i, j+**1**, x, y0+W+z, y0+W-y, x0, y0+W, L, H, W);

**if**(i<=**0** && i>-**2**)

turn(i-**1**, j, x0-z, y, x-x0, x0-H, y0, H, W, L);

**if**(j<=**0** && j>-**2**)

turn(i, j-**1**, x, y0-z, y-y0, x0, y0-H, L, H, W);

}

}

**int** main(){

**int** L, H, W, x1, y1, z1, x2, y2, z2;

cin >> L >> W >> H >> x1 >> y1 >> z1 >> x2 >> y2 >> z2;

**if** (z1!=**0** && z1!=H)

**if** (y1==**0** || y1==W)

swap(y1,z1), std::swap(y2,z2), std::swap(W,H);

**else**

swap(x1,z1), std::swap(x2,z2), std::swap(L,H);

**if** (z1==H) z1=**0**, z2=H-z2;

r=**0x3fffffff**; turn(**0**,**0**,x2-x1,y2-y1,z2,-x1,-y1,L,W,H);

cout<<r<<endl;

**return** **0**;

}

## 字符串的最小表示(正确的zy)

program peng;

var

i,j,k,n,m,t:longint;

a:array[**1..2000000**]of **char**;

**begin**

readln(n);

**for** i:=**1** to n **do** read(a[i]);

**for** i:=n+**1** to n+n **do** a[i]:=a[i-n];

i:=**1**; j:=**2**; k:=**0**; t:=**0**;

repeat

k:=**0**;

**while** (a[i+k]=a[j+k]) **do** inc(k);

**if** a[i+k]>a[j+k] then i:=i+k+**1**

**else** j:=j+k+**1**;

**if** i=j then inc(j); {==============important======}

**if** i>j then **begin** t:=i; i:=j; j:=t; **end**;

until j>n ;

writeln(i);

**end**.

## 最长公共子序列

**const** **int** dx[]={**0**,-**1**,**0**,**1**};

**const** **int** dy[]={**1**,**0**,-**1**,**0**};

**const** string ds=**"ENWS"**;

**char** G[**52**][**52**];

**char** A[**22222**], B[**22222**], buf[**22222**];

**int** n, m;

**typedef** **unsigned** **long** **long** ll;

**const** **int** M = **62**;

**const** **int** maxn = **20010**;

**const** **int** maxt = **130**;

**const** **int** maxl = maxn / M + **10**;

**const** ll Top = ((ll) **1** << (M));

**const** ll Topless = Top - **1**;

**const** ll underTop = ((ll) **1** << (M - **1**));

**typedef** ll bitarr[maxl];

bitarr comp[maxt], row[**2**], X;

**void** get(**char** \*S){

**int** L,x,y,sz=**0**;

scanf(**"%d%d%d"**,&L,&x,&y),x--,y--;

//scanf(" %s",buf);

S[sz++]=G[x][y];

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

**char** ch;

scanf(**" %c"**, &ch);

**int** pos=ds.find(ch);

x+=dx[pos],y+=dy[pos];

**if** (x < **0** || y < **0** || x >= n || y >= m) **for**(;;);

S[sz++]=G[x][y];

}

S[sz]=**0**;

}

**bool** calc[maxt];

**void** prepare() {

**int** u, p;

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

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

u = B[i];

**if** (calc[u]) **continue**; //======仅对所有字符集 ，每次一次

calc[u] = **1**;

memset(comp[u], **0**, **sizeof**(comp[u]));

**for** (p = **0**; p < n; p++) **if** (u == A[p]) comp[u][p / M] ^= ((ll) **1** << (p % M));

}

}

**void** solve() {

prepare();

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

**int** prev, curt;

**int** i, u, p, c, cc;

**int** Ln = (n / M) + **1**;

prev = **0**;

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

curt = **1** - prev; u = B[i];

**for** (p = **0**; p < Ln; p++) X[p] = row[prev][p] | comp[u][p];

c = **0**;

**for** (p = **0**; p < Ln; p++) {

cc = (row[prev][p] & underTop) > **0**;

row[prev][p] = ((row[prev][p] & (underTop - **1**)) << **1**) + c;

c = cc;

}

**for** (p = **0**; p < Ln; p++) {

**if** (row[prev][p] != Topless) {

row[prev][p]++;

**break**;

}

row[prev][p] = **0**;

}

c = **0**;

**for** (p = **0**; p < Ln; p++) {

**if** (X[p] >= row[prev][p] + c)

row[prev][p] = X[p] - (row[prev][p] + c), c = **0**;

**else**

row[prev][p] = Top + X[p] - (row[prev][p] + c), c = **1**;

}

**for** (p = **0**; p < Ln; p++)

row[curt][p] = X[p] & (row[prev][p] ^ X[p]);

prev = curt;

}

**int** ret = **0**;

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

**if** (row[prev][i / M] & ((ll) **1** << (i % M))) ret++;

// printf("%d %d %d\n", n, m, ret);

//=========ret 就是最长公共子序列。

printf(**"%d %d\n"**, n - ret, m - ret);

}

**int** main(){

**int** tests=**0**,T;

scanf(**"%d"**,&T);

**while**(T--){

scanf(**"%d%d"**,&n,&m);

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

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

scanf(**" %c"**,&G[i][j]);

get(A),get(B);

printf(**"Case %d: "**, ++tests);

// printf("A = %s\n, B = %s\n", A, B);

n = strlen(A), m = strlen(B);

//n = 20000; m = 20000;

//for (int i = 0; i < m; i++) A[i] = B[i] = 'A';

//A[m] = B[m] = 0;

solve();

}

}

## Splay-Tree（带split）

**const** **int** maxNodeCnt **=** 111111**;**

**int** nodeCnt**,** root**,** type**[**maxNodeCnt**],** parent**[**maxNodeCnt**],** childs**[**maxNodeCnt**][**2**],** size**[**maxNodeCnt**],** stack**[**maxNodeCnt**],** reversed**[**maxNodeCnt**];**

// ...

**void** clear**()** **{**

root **=** 0**;**

size**[**0**]** **=** 0**;**

nodeCnt **=** 1**;**

**}**

**int** malloc**()** **{**

type**[**nodeCnt**]** **=** 2**;**

childs**[**nodeCnt**][**0**]** **=** childs**[**nodeCnt**][**1**]** **=** 0**;**

size**[**nodeCnt**]** **=** 1**;**

reversed**[**nodeCnt**]** **=** 0**;**

**return** nodeCnt **++;**

**}**

**void** update**(int** x**)** **{**

size**[**x**]** **=** size**[**childs**[**x**][**0**]]** **+** 1 **+** size**[**childs**[**x**][**1**]];**

// ...

**}**

**void** pass**(int** x**)** **{**

// NOTICE: childs[x][i] == 0

**if** **(**reversed**[**x**])** **{**

swap**(**childs**[**x**][**0**],** childs**[**x**][**1**]);**

type**[**childs**[**x**][**0**]]** **=** 0**;**

reversed**[**childs**[**x**][**0**]]** **^=** 1**;**

type**[**childs**[**x**][**1**]]** **=** 1**;**

reversed**[**childs**[**x**][**1**]]** **^=** 1**;**

reversed**[**x**]** **=** 0**;**

**}**

// ...

**}**

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

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

y **=** parent**[**x**],**

z **=** childs**[**x**][**1 **-** t**];**

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

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

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

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

**}**

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

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

childs**[**x**][**1 **-** t**]** **=** y**;**

**if** **(**z**)** **{**

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

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

**}**

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

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

**}**

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

**int** stackCnt **=** 0**;**

stack**[**stackCnt **++]** **=** x**;**

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

stack**[**stackCnt **++]** **=** parent**[**i**];**

**}**

**for** **(int** i **=** stackCnt **-** 1**;** i **>** **-**1**;** **--** i**)** **{**

pass**(**stack**[**i**]);**

**}**

**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**);**

**}**

**int** find**(int** x**,** **int** rank**)** **{**

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

pass**(**x**);**

**if** **(**size**[**childs**[**x**][**0**]]** **+** 1 **==** rank**)** **{**

**break;**

**}**

**if** **(**rank **<=** size**[**childs**[**x**][**0**]])** **{**

x **=** childs**[**x**][**0**];**

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

rank **-=** size**[**childs**[**x**][**0**]]** **+** 1**;**

x **=** childs**[**x**][**1**];**

**}**

**}**

**return** x**;**

**}**

**void** split**(int** **&**x**,** **int** **&**y**,** **int** a**)** **{**

// NOTICE: x, y != 0

y **=** find**(**x**,** a **+** 1**);**

splay**(**y**);**

x **=** childs**[**y**][**0**];**

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

childs**[**y**][**0**]** **=** 0**;**

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

**}**

**void** split3**(int** **&**x**,** **int** **&**y**,** **int** **&**z**,** **int** a**,** **int** b**)** **{**

split**(**x**,** z**,** b**);**

split**(**x**,** y**,** a **-** 1**);**

**}**

**void** join**(int** **&**x**,** **int** y**)** **{**

// NOTICE x, y != 0

x **=** find**(**x**,** size**[**x**]);**

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

childs**[**x**][**1**]** **=** y**;**

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

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

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

**}**

**void** join3**(int** **&**x**,** **int** y**,** **int** z**)** **{**

join**(**y**,** z**);**

join**(**x**,** y**);**

**}**

**int** getRank**(int** x**)** **{**

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

root **=** x**;**

**return** size**[**childs**[**x**][**0**]];**

**}**

**void** reverse**(int** a**,** **int** b**)** **{**

**int** x**,** y**;**

split3**(**root**,** x**,** y**,** a **+** 1**,** b **+** 1**);**

reversed**[**x**]** **^=** 1**;**

join3**(**root**,** x**,** y**);**

**}**

## 动态树(ftiasch)

**const** **int** N **=** 666666**;**

**int** n**,**

edgeCnt**,** firstEdge**[**N**],** to**[**N**],** nextEdge**[**N**],**

type**[**N**],** parent**[**N**],** childs**[**N**][**2**],**

top**,** stack**[**N**],**

reversed**[**N**],** key**[**N**],** delta**[**N**],** maximum**[**N**];**

**void** passDelta **(int** x**,** **int** d**)** **{**

**if** **(**x**)** **{**

key**[**x**]** **+=** d**;**

delta**[**x**]** **+=** d**;**

maximum**[**x**]** **+=** d**;**

**}**

**}**

**void** pass **(int** x**)** **{**

**if** **(**reversed**[**x**])** **{**

swap**(**childs**[**x**][**0**],** childs**[**x**][**1**]);**

type**[**childs**[**x**][**0**]]** **=** 0**;**

type**[**childs**[**x**][**1**]]** **=** 1**;**

reversed**[**childs**[**x**][**0**]]** **^=** 1**;**

reversed**[**childs**[**x**][**1**]]** **^=** 1**;**

reversed**[**x**]** **^=** 1**;**

**}**

**if** **(**delta**[**x**])** **{**

passDelta**(**childs**[**x**][**0**],** delta**[**x**]);**

passDelta**(**childs**[**x**][**1**],** delta**[**x**]);**

delta**[**x**]** **=** 0**;**

**}**

**}**

**void** update **(int** x**)** **{**

maximum**[**x**]** **=** max**(**key**[**x**],**

max**(**maximum**[**childs**[**x**][**0**]],** maximum**[**childs**[**x**][**1**]]));**

**}**

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

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

y **=** parent**[**x**],**

z **=** childs**[**x**][**1 **-** t**];**

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

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

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

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

**}**

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

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

childs**[**x**][**1 **-** t**]** **=** y**;**

**if** **(**z**)** **{**

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

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

**}**

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

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

**}**

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

top **=** 0**;**

stack**[**top **++]** **=** x**;**

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

stack**[**top **++]** **=** parent**[**i**];**

**}**

**for** **(int** i **=** top **-** 1**;** i **>** **-**1**;** **--** i**)** **{**

pass**(**stack**[**i**]);**

**}**

**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**)** **{**

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

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

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

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

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

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

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

**}**

**}**

**void** setRoot **(int** x**)** **{**

access**(**x**);**

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

reversed**[**x**]** **^=** 1**;**

**}**

**int** findRoot **(int** x**)** **{**

access**(**x**);**

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

**while** **(**childs**[**x**][**0**])** **{**

x **=** childs**[**x**][**0**];**

pass**(**x**);**

**}**

**return** x**;**

**}**

## 动态树

#define maxn 10005

**struct** Tsplay

{

**int** p,l,r,s[**2**],c;

**bool** reverse;

}Tree[maxn];

**int** n,m;

**inline** **void** swap(**int** &a,**int** &b)

{

**int** t=a;a=b;b=t;

}

**inline** **void** Pass(**int** u)

{

**if** (!Tree[u].reverse) **return**;

Tree[Tree[u].l].reverse^=**1**;

Tree[Tree[u].r].reverse^=**1**;

swap(Tree[u].l,Tree[u].r);

Tree[u].reverse=**0**;

}

**inline** **void** Update(**int** x)

{

Tree[x].s[**0**]=Tree[Tree[x].l].s[**0**]+Tree[Tree[x].r].s[**0**];

Tree[x].s[**1**]=Tree[Tree[x].l].s[**1**]+Tree[Tree[x].r].s[**1**];

++Tree[x].s[Tree[x].c];

}

**inline** **void** zig(**int** x)

{

**int** y=Tree[x].p;

**int** z=Tree[y].p;

**if** (Tree[z].l==y) Tree[z].l=x;

**else** **if** (Tree[z].r==y) Tree[z].r=x;

Tree[x].p=z;Tree[y].p=x;

Tree[y].l=Tree[x].r;

Tree[x].r=y;

Tree[Tree[y].l].p=y;

Update(y);

Update(x);

}

**inline** **void** zag(**int** x)

{

**int** y=Tree[x].p;

**int** z=Tree[y].p;

**if** (Tree[z].l==y) Tree[z].l=x;

**else** **if** (Tree[z].r==y) Tree[z].r=x;

Tree[x].p=z;Tree[y].p=x;

Tree[y].r=Tree[x].l;

Tree[x].l=y;

Tree[Tree[y].r].p=y;

Update(y);

Update(x);

}

**inline** **bool** Root(**int** t)

{

**return** Tree[Tree[t].p].l!=t && Tree[Tree[t].p].r!=t;

}

**int** stack[maxn],top;

**inline** **void** Splay(**int** x)

{

stack[top=**1**]=x;

**for** (**int** t=x;!Root(t);t=Tree[t].p)

stack[++top]=Tree[t].p;

**for** (;top;--top)

Pass(stack[top]);

**for** (;!Root(x);)

{

**int** y=Tree[x].p,z=Tree[y].p;

**if** (Root(y))

{

**if** (Tree[y].l==x) zig(x);

**else** zag(x);

}**else**

{

**if** (Tree[z].l==y)

**if** (Tree[y].l==x) zig(y),zig(x);

**else** zag(x),zig(x);

**else**

**if** (Tree[y].r==x) zag(y),zag(x);

**else** zig(x),zag(x);

}

}

Update(x);

}

//以上是Splay部分

**inline** **void** Expose(**int** u)

{

**for** (**int** v=**0**;u;u=Tree[u].p)

{

Splay(u);

Tree[u].r=v;

Update(v=u);

}

}

//把u到根的边都变成实边

**inline** **int** FindRoot(**int** x)

{

Expose(x);Splay(x);

**for** (;Pass(x),Tree[x].l;x=Tree[x].l);

**return** x;

}

//找x所在树的真正的根

**inline** **int** LCA(**int** x,**int** y)

{

**int** ret=**0**;

Expose(x);

**for** (**int** u=y,v=**0**;u;u=Tree[u].p)

{

Splay(u);

**if** (!Tree[u].p) ret=u;

Tree[u].r=v;

Update(v=u);

}

**return** ret;

}

//求x,y的最近公共祖先

**int** x,y;

**inline** **void** Add()

{

scanf(**"%d%d"**,&x,&y);

Expose(x);Splay(x);

Expose(y);Splay(y);

Tree[x].r=**0**;

Tree[x].reverse=**1**;

Tree[x].p=y;

Tree[y].r=x;

Update(y);

}

//添加一条x,y的边

**inline** **void** Del()

{

scanf(**"%d%d"**,&x,&y);

**int** z=LCA(x,y);

**if** (z==y) swap(x,y);

Expose(y);Splay(y);

Tree[Tree[y].l].p=**0**;

Tree[y].l=**0**;

Update(y);

}

//删除一条x,y的边

**inline** **void** Set()

{

**char** st[**10**];

scanf(**"%d%s"**,&x,st);

Splay(x);

Tree[x].c=(st[**0**]==**'W'**);

Update(x);

}

//修改x的有关属性值

**inline** **void** Query()

{

scanf(**"%d%d"**,&x,&y);

**if** (FindRoot(x)!=FindRoot(y))

{

puts(**"-1"**);

**return**;

}

**int** ret[**2**];

ret[**0**]=ret[**1**]=**0**;

Expose(x);

**for** (**int** u=y,v=**0**;u;u=Tree[u].p)

{

Splay(u);

**if** (!Tree[u].p)

{

ret[**0**]=Tree[v].s[**0**]+Tree[Tree[u].r].s[**0**];

ret[**1**]=Tree[v].s[**1**]+Tree[Tree[u].r].s[**1**];

++ret[Tree[u].c];

}

Tree[u].r=v;

Update(v=u);

}

printf(**"%d %d\n"**,ret[**0**],ret[**1**]);

}

//完成x->y路径上的一些查询

**int** main()

{

freopen(**"G.in"**,**"r"**,stdin);

freopen(**"G.out"**,**"w"**,stdout);

**for** (;scanf(**"%d%d"**,&n,&m) && (n+m);)

{

**char** st[**10**];

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

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

{

scanf(**"%s"**,st);

Tree[i].c=(st[**0**]==**'W'**);

Update(i);

}

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

{

scanf(**"%s"**,st);

**if** (st[**0**]==**'a'**) Add();

**else** **if** (st[**0**]==**'d'**) Del();

**else** **if** (st[**0**]==**'s'**) Set();

**else** Query();

}

}

**return** **0**;

}

## 曼哈顿最小生成树

只需要考虑每个点的 pi/**4**\*k -- pi/**4**\*(k+**1**)的区间内的第一个点，这样只有4n条无向边。

\***/**

**const** **int** maxn = **100000**+**5**;

**const** **int** Inf = **1000000005**;

**struct** TreeEdge

{

**int** x,y,z;

**void** make( **int** \_x,**int** \_y,**int** \_z ) { x=\_x; y=\_y; z=\_z; }

} data[maxn\***4**];

**inline** **bool** **operator** < ( **const** TreeEdge& x,**const** TreeEdge& y ){

**return** x.z<y.z;

}

**int** x[maxn],y[maxn],px[maxn],py[maxn],id[maxn],tree[maxn],node[maxn],val[maxn],fa[maxn];

**int** n;

**inline** **bool** compare1( **const** **int** a,**const** **int** b ) { **return** x[a]<x[b]; }

**inline** **bool** compare2( **const** **int** a,**const** **int** b ) { **return** y[a]<y[b]; }

**inline** **bool** compare3( **const** **int** a,**const** **int** b ) { **return** (y[a]-x[a]<y[b]-x[b] || y[a]-x[a]==y[b]-x[b] && y[a]>y[b]); }

**inline** **bool** compare4( **const** **int** a,**const** **int** b ) { **return** (y[a]-x[a]>y[b]-x[b] || y[a]-x[a]==y[b]-x[b] && x[a]>x[b]); }

**inline** **bool** compare5( **const** **int** a,**const** **int** b ) { **return** (x[a]+y[a]>x[b]+y[b] || x[a]+y[a]==x[b]+y[b] && x[a]<x[b]); }

**inline** **bool** compare6( **const** **int** a,**const** **int** b ) { **return** (x[a]+y[a]<x[b]+y[b] || x[a]+y[a]==x[b]+y[b] && y[a]>y[b]); }

**void** Change\_X()

{

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

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

sort(id,id+n,compare1);

**int** cntM=**1**, last=val[id[**0**]]; px[id[**0**]]=**1**;

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

{

**if**(val[id[i]]>last) ++cntM,last=val[id[i]];

px[id[i]]=cntM;

}

}

**void** Change\_Y()

{

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

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

sort(id,id+n,compare2);

**int** cntM=**1**, last=val[id[**0**]]; py[id[**0**]]=**1**;

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

{

**if**(val[id[i]]>last) ++cntM,last=val[id[i]];

py[id[i]]=cntM;

}

}

**inline** **int** absValue( **int** x ) { **return** (x<**0**)?-x:x; }

**inline** **int** Cost( **int** a,**int** b ) { **return** absValue(x[a]-x[b])+absValue(y[a]-y[b]); }

**int** find( **int** x ) { **return** (fa[x]==x)?x:(fa[x]=find(fa[x])); }

**int** main()

{

// freopen("input.txt", "r", stdin);

// freopen("output.txt", "w", stdout);

**int** test=**0**;

**while**( scanf(**"%d"**,&n)!=EOF && n )

{

**for**(**int** i=**0**;i<n;++i) scanf(**"%d%d"**,x+i,y+i);

Change\_X();

Change\_Y();

**int** cntE = **0**;

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

sort(id,id+n,compare3);

**for**(**int** i=**1**;i<=n;++i) tree[i]=Inf,node[i]=-**1**;

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

{

**int** Min=Inf, Tnode=-**1**;

**for**(**int** k=py[id[i]];k<=n;k+=k&(-k)) **if**(tree[k]<Min) Min=tree[k],Tnode=node[k];

**if**(Tnode>=**0**) data[cntE++].make(id[i],Tnode,Cost(id[i],Tnode));

**int** tmp=x[id[i]]+y[id[i]];

**for**(**int** k=py[id[i]];k;k-=k&(-k)) **if**(tmp<tree[k]) tree[k]=tmp,node[k]=id[i];

}

sort(id,id+n,compare4);

**for**(**int** i=**1**;i<=n;++i) tree[i]=Inf,node[i]=-**1**;

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

{

**int** Min=Inf, Tnode=-**1**;

**for**(**int** k=px[id[i]];k<=n;k+=k&(-k)) **if**(tree[k]<Min) Min=tree[k],Tnode=node[k];

**if**(Tnode>=**0**) data[cntE++].make(id[i],Tnode,Cost(id[i],Tnode));

**int** tmp=x[id[i]]+y[id[i]];

**for**(**int** k=px[id[i]];k;k-=k&(-k)) **if**(tmp<tree[k]) tree[k]=tmp,node[k]=id[i];

}

sort(id,id+n,compare5);

**for**(**int** i=**1**;i<=n;++i) tree[i]=Inf,node[i]=-**1**;

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

{

**int** Min=Inf, Tnode=-**1**;

**for**(**int** k=px[id[i]];k;k-=k&(-k)) **if**(tree[k]<Min) Min=tree[k],Tnode=node[k];

**if**(Tnode>=**0**) data[cntE++].make(id[i],Tnode,Cost(id[i],Tnode));

**int** tmp=-x[id[i]]+y[id[i]];

**for**(**int** k=px[id[i]];k<=n;k+=k&(-k)) **if**(tmp<tree[k]) tree[k]=tmp,node[k]=id[i];

}

sort(id,id+n,compare6);

**for**(**int** i=**1**;i<=n;++i) tree[i]=Inf,node[i]=-**1**;

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

{

**int** Min=Inf, Tnode=-**1**;

**for**(**int** k=py[id[i]];k<=n;k+=k&(-k)) **if**(tree[k]<Min) Min=tree[k],Tnode=node[k];

**if**(Tnode>=**0**) data[cntE++].make(id[i],Tnode,Cost(id[i],Tnode));

**int** tmp=-x[id[i]]+y[id[i]];

**for**(**int** k=py[id[i]];k;k-=k&(-k)) **if**(tmp<tree[k]) tree[k]=tmp,node[k]=id[i];

}

**long** **long** Ans = **0**;

sort(data,data+cntE);

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

**for**(**int** i=**0**;i<cntE;++i) **if**(find(data[i].x)!=find(data[i].y))

{

Ans += data[i].z;

fa[fa[data[i].x]]=fa[data[i].y];

}

cout<<**"Case "**<<++test<<**": "**<<**"Total Weight = "**<<Ans<<endl;

}

**return** **0**;

}

## 表达式的计算

#include <cctype>

**const** **int** maxl = **1000**;**const** **int** maxt = **100**;

**const** **double** eps = **1e**-**8**;**int** value[**26**];**char** str1[maxl], str2[maxl];

**inline** **int** Level(**char** ch) {

**switch** (ch) {

**case** **'+'** :

**case** **'-'** : **return** **0**;

**case** **'\*'** : **return** **1**;

}

**return** -**1**;

}

**int** Calc(**const** **char** \*&p, **int** level) {

**int** res;

**if** (level == **2**) {

**if** (\*p == **'('**) { p++; res = Calc(p, **0**); p++; }

**else** { res = isdigit(\*p) ? \*p - **'0'** : value[\*p - **'a'**]; p++; }

**return** res;

}

res = Calc(p, level + **1**); **char** ch; **int** next;

**while** (\*p && Level(\*p) == level) {

ch = \*p++; next = Calc(p, level + **1**);

**switch** (ch) {

**case** **'+'** : res += next; **break**;

**case** **'-'** : res -= next; **break**;

**case** **'\*'** : res \*= next; **break**;

}

}

**return** res;

}

**int** Evaluate(**const** **char** \*str) { **const** **char** \*p = str; **return** Calc(p, **0**);}

**void** Work() {

**int** i, j; **for** (j = **0**; j < **26**; j++) value[j] = rand();

ans=Evaluate(str1)

}

## 二维树状数组

**template** <**class** T, **int** N>

**struct** radixtree {

T dat[N+**1**];

**int** lowbit(**int** t) { **return** t & (-t); }

radixtree() { }

**void** init() { memset( dat, **0**, **sizeof**(dat)); }

**void** add(**int** x, T v) {

**for** ( ; x <= N; dat[x] += v, x = lowbit(x));

}

T sum(**int** x) {

T s = **0**;

**for** ( ; x >= **1**; s += dat[x], x -= lowbit(x));

**return** s;

}

};

**template** <**class** T, **int** N, **int** M>

**struct** radixtree2 {

T dat[N+**1**][M+**1**];

**int** lowbit(**int** t) { **return** t & (-t); }

radixtree2() { }

**void** init() { memset(dat, **0**, **sizeof**(dat)); }

**void** add(**int** x, **int** y, T v) {

**int** yy = y;

**while** ( x <= N ) {

y = yy;

**while** (y <= M) {

dat[x][y] += v;

y += lowbit(y);

}

x += lowbit(x);

}

}

T sum(**int** x, **int** y) {

**int** yy = y; T s = **0**;

**while** ( x > **0** ) {

y = yy;

**while** ( y > **0** ) {

s += dat[x][y];

y -= lowbit(y);

}

x -= lowbit(x);

}

**return** s;

}

};

## 双人零和矩阵游戏

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

//双人零和矩阵游戏，矩阵的大小为n\*n，该解法中矩阵必须非奇异，如果是奇异矩阵，则给每个元素加上一个数字即可。

#define maxn 205

#define eps 1e-8

**double** map[maxn][maxn],ans[maxn];

**int** n;

**inline** **void** swap(**double** &a,**double** &b)

{

**double** t=a;a=b;b=t;

}

**int** main()

{

scanf(**"%d"**,&n);

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

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

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

{

scanf(**"%lf"**,&map[j][i]);

map[j][i]=-map[j][i]+**20**;

}

//map[][]是那个矩阵

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

map[i][i+n]=**1**;

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

{

**int** p;

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

**if** (fabs(map[i][j])>eps)

{

p=j;

**break**;

}

**for** (**int** j=**1**;j<=**2**\*n;++j)

swap(map[i][j],map[p][j]);

**double** delta=**1.0**/map[i][i];

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

map[i][j]\*=delta;

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

**if** (i!=j && fabs(map[j][i])>eps)

{

delta=map[j][i];

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

map[j][k]-=map[i][k]\*delta;

}

}

**double** s=**0**;

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

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

{

map[i][j]=map[i][j+n];

s+=map[i][j];

}

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

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

ans[j]+=map[i][j];

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

ans[i]/=s;

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

printf(**"%.5lf\n"**,ans[i]);

//ans[i]为第一个人选择第i行的概率

**return** **0**;

}

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

用线性规划解决

## Exact Cover(crazyb0y)

**class** ExactCover**{**

**private:**

vector**<int>** u**,**d**,**l**,**r**,**C**,**R**,**head**,**tail**;**

**int** head0**,**tail0**,**seed**;**

**void** cover**(int** x**){**

**int** i**=**x**,**j**;**

r**[**l**[**x**]]=**r**[**x**];**

l**[**r**[**x**]]=**l**[**x**];**

**while((**i**=**d**[**i**])!=**x**){**

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

**while((**j**=**l**[**j**])!=**i**){**

u**[**d**[**j**]]=**u**[**j**];**

d**[**u**[**j**]]=**d**[**j**];**

R**[**C**[**j**]]--;**

**}**

**}**

**}**

**void** uncover**(int** x**){**

**int** i**=**x**,**j**;**

**while((**i**=**u**[**i**])!=**x**){**

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

**while((**j**=**r**[**j**])!=**i**){**

u**[**d**[**j**]]=**j**;**

d**[**u**[**j**]]=**j**;**

R**[**C**[**j**]]++;**

**}**

**}**

r**[**l**[**x**]]=**x**;**

l**[**r**[**x**]]=**x**;**

**}**

**public:**

vector**<int>** ans**;**

**void** resize**(int** n**){**

u**.**resize**(**1**,**0**);**

d**.**resize**(**1**,**0**);**

l**.**resize**(**1**,**0**);**

r**.**resize**(**1**,**0**);**

C**.**resize**(**1**,-**1**);**

R**.**resize**(**1**,-**1**);**

head**.**resize**(**n**,-**1**);**

tail**.**resize**(**n**,-**1**);**

ans**.**resize**(**n**,**0**);**

head0**=**tail0**=**0**;**

**}**

**void** add**(**vector**<int>** a**,bool** must**=true){**

u**.**push\_back**(**u**.**size**()+**a**.**size**());**

**if(**must**){**

l**.**push\_back**(**tail0**);**

r**.**push\_back**(**head0**);**

tail0**=**l**[**r**[**d**.**size**()]]=**r**[**l**[**d**.**size**()]]=**d**.**size**();**

**}else{**

l**.**push\_back**(**l**.**size**());**

r**.**push\_back**(**r**.**size**());**

**}**

C**.**push\_back**(**C**.**size**());**

R**.**push\_back**(**a**.**size**());**

**int** n**=**u**.**size**(),**m**=**a**.**size**(),**i**,**j**;**

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

j**=**a**[**i**];**

**if(**head**[**j**]==-**1**){**

l**.**push\_back**(**n**+**i**);**

r**.**push\_back**(**n**+**i**);**

head**[**j**]=**n**+**i**;**

tail**[**j**]=**n**+**i**;**

**}else{**

l**.**push\_back**(**tail**[**j**]);**

r**.**push\_back**(**head**[**j**]);**

tail**[**j**]=**r**[**l**[**n**+**i**]]=**l**[**r**[**n**+**i**]]=**n**+**i**;**

**}**

u**.**push\_back**(**n**+**i**-**1**);**

d**.**push\_back**(**n**+**i**);**

C**.**push\_back**(**C**.**back**());**

R**.**push\_back**(**j**);**

**}**

d**.**push\_back**(**n**-**1**);**

**}**

**void** select**(int** a**){**

ans**[**a**]=**1**;**

a**=**head**[**a**];**

**if(**a**==-**1**)**

**return;**

**int** x**=**a**;**

**while((**x**=**r**[**x**])!=**a**)**

cover**(**C**[**x**]);**

cover**(**C**[**a**]);**

**}**

**bool** search**(){**

**if(**r**[**0**]==**0**)**

**return** **true;**

**int** x**,**i**,**j**,**min**=**0x7fffffff**;**

i**=**0**;**

**while((**i**=**r**[**i**])!=**0**)**

**if(**R**[**i**]<**min**||!(++**seed**&**3**)&&**R**[**i**]==**min**)**

min**=**R**[**x**=**i**];**

cover**(**i**=**x**);**

**while((**i**=**d**[**i**])!=**x**){**

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

**while((**j**=**r**[**j**])!=**i**)**

cover**(**C**[**j**]);**

ans**[**R**[**i**]]=**1**;**

**if(**search**())**

**return** **true;**

ans**[**R**[**i**]]=**0**;**

**while((**j**=**l**[**j**])!=**i**)**

uncover**(**C**[**j**]);**

**}**

uncover**(**x**);**

**return** **false;**

**}**

**};**

## 数独 Dancing Links

//数独 Dancing Links

**const** **int** Row=**16**\***16**\***16**+**5**;

**const** **int** Col=**16**\***16**\***4**+**5**;

**const** **int** size=**20000**;

**int** L[size],R[size],U[size],D[size],col[size],x[size],y[size],c[size];

**int** X[**17**][**17**][**17**],Y[**17**][**17**][**17**],grid[**17**][**17**][**17**],sub[**17**][**17**][**17**],first[**17**][**17**][**17**];;

**int** info[Row];

**int** now[Col],sum[Col];

**char** map[**17**][**17**];

**int** h,t,tot,need;

**void** init()

{

**int** i;

**for** (i=**1**;i<=**16**;++i)

{

**char** ch=getchar();

**while** (ch!=**'\n'** && ch!=**' '**) ch=getchar();

scanf(**"%s"**,map[i]+**1**);

}

}

**void** prepare()

{

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

**for** (i=**1**;i<=**16**;++i)

**for** (j=**1**;j<=**16**;++j)

{

++cnt;

**for** (k=**1**;k<=**16**;++k)

{

X[i][j][k]=(i-**1**)\***16**+k;

Y[i][j][k]=**16**\***16**+(j-**1**)\***16**+k;

sub[i][j][k]=**16**\***16**\***2**+((i-**1**)/**4**\***4**+(j-**1**)/**4**)\***16**+k;

grid[i][j][k]=**16**\***16**\***3**+cnt;

}

}

}

**void** insert(**int** x)

{

col[++tot]=x;

++sum[x];

R[info[t]]=tot;L[tot]=info[t];info[t]=tot;

D[now[x]]=tot;U[tot]=now[x];now[x]=tot;

}

**void** DLX()

{

**int** i,j,k;

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

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

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

tot=**16**\***16**\***4**;

**for** (i=**1**;i<=tot;++i)

{

L[i]=i-**1**;R[i]=i+**1**;sum[i]=**0**;now[i]=i;

}

h=++tot;

L[h]=tot-**1**;R[h]=**1**;R[tot-**1**]=h;L[**1**]=h;

t=**0**;

**for** (i=**1**;i<=**16**;++i)

**for** (j=**1**;j<=**16**;++j)

**for** (k=**1**;k<=**16**;++k)

{

first[i][j][k]=tot+**1**;

++t;

insert(X[i][j][k]);

insert(Y[i][j][k]);

insert(sub[i][j][k]);

insert(grid[i][j][k]);

x[tot]=x[tot-**1**]=x[tot-**2**]=x[tot-**3**]=i;

y[tot]=y[tot-**1**]=y[tot-**2**]=y[tot-**3**]=j;

c[tot]=c[tot-**1**]=c[tot-**2**]=c[tot-**3**]=k;

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

**while** (L[x]) x=L[x];

L[x]=info[t];R[info[t]]=x;

}

**for** (i=**1**;i<=**16**\***16**\***4**;++i)

{

U[i]=now[i];D[now[i]]=i;

}

}

**void** cover(**int** x)

{

**if** (R[L[x]]!=x) **return**;

**int** i,j;

R[L[x]]=R[x];L[R[x]]=L[x];

**for** (i=D[x];i!=x;i=D[i])

**for** (j=R[i];j!=i;j=R[j])

{

U[D[j]]=U[j];D[U[j]]=D[j];--sum[col[j]];

}

}

**void** recover(**int** x)

{

**int** i,j;

**for** (i=U[x];i!=x;i=U[i])

**for** (j=L[i];j!=i;j=L[j])

{

D[U[j]]=j;U[D[j]]=j;++sum[col[j]];

}

R[L[x]]=x;L[R[x]]=x;

}

**int** dfs(**int** dep)

{

**int** i,j,k=-**1**,min=**1000000000**;

**if** (dep==need) **return** **1**;

**for** (i=L[h];i!=h;i=L[i])

**if** (sum[i]<min)

{

min=sum[i];

k=i;

}

**if** (k==-**1**) **return** **1**;

**if** (!sum[k]) **return** **0**;

cover(k);

**for** (i=D[k];i!=k;i=D[i])

{

**for** (j=R[i];j!=i;j=R[j]) cover(col[j]);

map[x[i]][y[i]]=**'A'**+c[i]-**1**;

**if** (dfs(dep+**1**)) **return** **1**;

**for** (j=L[i];j!=i;j=L[j]) recover(col[j]);

}

recover(k);

**return** **0**;

}

**void** work()

{

**int** i,j;

DLX();

need=**16**\***16**;

**for** (i=**1**;i<=**16**;++i)

**for** (j=**1**;j<=**16**;++j)

**if** (map[i][j]!=**'-'**)

{

**int** k=first[i][j][map[i][j]-**'A'**+**1**];

**while** (**1**)

{

cover(col[k]);

k=R[k];

**if** (k==first[i][j][map[i][j]-**'A'**+**1**]) **break**;

}

--need;

}

dfs(**0**);

}

**void** print()

{

**int** i,j;

**for** (i=**1**;i<=**16**;++i)

{

**for** (j=**1**;j<=**16**;++j) putchar(map[i][j]);

putchar(**'\n'**);

}

}

**int** main()

{

**int** T;

prepare();

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

{

init();

work();

print();

**if** (T) putchar(**'\n'**);

}

**return** **0**;

}

Procedure Algorithm\_X(Dep)

如果矩阵中所有的列均被删除, 找到一组合法解, 退出．

任意选择一个未被删除的列c,

枚举一个未被删除的行r, 且Matrix[r][c] = **1**, 将(r, c)加入Ans．

枚举所有的列j, Matrix[r][j] = **1**, 将第j列删除．

枚举所有的行i, Matrix[i][j] = **1**, 将第i行删除．

Algorithm\_X(Dep + **1**)

Procedure Algorithm\_X(Dep)

如果h^.right = h(即所有的列均被删除), 找到一组解, 退出．

利用h和right指针找到一个c, 满足size[c]最小．

如果size[c] = **0**(当前列无法被覆盖), 无解, 退出．

Cover(c)

**for** (i = c^.down; i != c; i ← i^.down)

**for** (j = i^.right; j != i; j ← j^.right) Cover(j^.col)

将i结点加入Ans, Algorithm\_X(Dep + **1**)

**for** (j = i^.left; j != i; j ← j^.left) Recover(j^.col)

Recover(c)

Soduku问题可以转化一个Exact Cover Problem：16 \* 16 \* 16行, (i, j, k)表示(i, j)这个格子填上字母k．16 \* 16 \* 4列分别表示第i行中的字母k, 第i列中的字母k, 第i个子矩阵中的字母k, 以及(i, j)这个格子．对于每个集合(i, j, k), 它包含了4个元素：Line(i, k), Col(j, k), Sub(P[i][j], k), Grid(i, j), 其中P[i][j]表示(i, j)这个格子所属的子矩阵．本题转化为一个4096行, 1024列, 且1的个数为16384个的矩阵．下面介绍解决一般的Exact Cover Problem的Algorithm X．

N皇后问题：关键是构建Exact Cover问题的矩阵：N \* N行对应了N \* N个格子, 6N-2列对应了N行, N列, 2N-1条主对角线, 2N-1条副对角线．第i行共4个1, 分别对应(i, j)这个格子所处的行, 列, 主对角线和副对角线．直接对这个矩阵作Algorithm X是错误的, 虽然每行, 每列都恰好被覆盖一次, 但是对角线是最多覆盖一次, 它可以不被覆盖, 这与Exact Cover问题的定义是不同的．

有两种处理的方法：

1) 新增4N-2行, 每行只有一个1, 分别对应了2N-1条主对角线和2N-1条副对角线, 这样就可以保证某个对角线不被覆盖的时候, 可以使用新增行来覆盖．

2) 每次选择一个size[]值最小的列c进行覆盖, 而这一步, 我们忽略掉所有的对角线列, 只考虑c为行和列的情况．

事实证明, 第2)种方法的效果好很多, 因此这个问题可以使用Algorithm X轻松得到解决．

## 线性规划

有m种资源和n个项目，每个资源都是有限的，设它们的上限为bj(**1** <= j <= m)。假设第i个项目做出xi的成果量，可以获得ci\*xi的收益，同时会消耗第j种资源aij\*xi。求最大收益。

标准形式：

目标函数是最大化的，所有的线性约束都是小于等于的不等式，所有的变量都有非负的限制。

**const** **double** eps = **1e**-**10**;

**const** **int** myMAXSIZE = **200**;

**const** **int** oo = **19890709**;

**double** myA[myMAXSIZE+**1**][myMAXSIZE+**1**], mytA[myMAXSIZE+**1**][myMAXSIZE+**1**];

**double** myb[myMAXSIZE+**1**], mytb[myMAXSIZE+**1**], myc[myMAXSIZE+**1**], mytc[myMAXSIZE+**1**];

**int** myN[myMAXSIZE+**1**+**1**], myB[myMAXSIZE+**1**+**1**];

**int** n, m;

**double** myV;

**bool** read()

{

**if** (scanf(**"%d%d"**, &n, &m) == EOF) **return** **false**;

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

scanf(**"%lf"**, &myc[i]);

//每种项目的利润

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

{

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

scanf(**"%lf"**, &myA[n+i][j]);

//第j个项目需要的第i种资源的量

scanf(**"%lf"**, &myb[n+i]);

//第i种资源的总量

}

**return** **true**;

}

**void** pivot(**int** l, **int** e)

{

mytb[e] = myb[l]/myA[l][e];

mytA[e][l] = **1**/myA[l][e];

**for**(**int** i=**1**; i<=myN[**0**]; i++)

**if** (myN[i] != e)

mytA[e][myN[i]] = myA[l][myN[i]]/myA[l][e];

**for**(**int** i=**1**; i<=myB[**0**]; i++)

{

mytb[myB[i]] = myb[myB[i]]-myA[myB[i]][e]\*mytb[e];

mytA[myB[i]][l] = -myA[myB[i]][e]\*mytA[e][l];

**for**(**int** j=**1**; j<=myN[**0**]; j++)

**if** (myN[j] != e)

mytA[myB[i]][myN[j]] = myA[myB[i]][myN[j]]-mytA[e][myN[j]]\*myA[myB[i]][e];

}

myV += mytb[e]\*myc[e];

mytc[l] = -mytA[e][l]\*myc[e];

**for**(**int** i=**1**; i<=myN[**0**]; i++)

**if** (myN[i] != e)

mytc[myN[i]] = myc[myN[i]]-mytA[e][myN[i]]\*myc[e];

**for**(**int** i=**1**; i<=myN[**0**]; i++)

**if** (myN[i] == e) myN[i] = l;

**for**(**int** i=**1**; i<=myB[**0**]; i++)

**if** (myB[i] == l) myB[i] = e;

**for**(**int** i=**1**; i<=myB[**0**]; i++)

{

**for**(**int** j=**1**; j<=myN[**0**]; j++)

myA[myB[i]][myN[j]] = mytA[myB[i]][myN[j]];

myb[myB[i]] = mytb[myB[i]];

}

**for**(**int** i=**1**; i<=myN[**0**]; i++)

myc[myN[i]] = mytc[myN[i]];

}

**bool** opt()//false stands for unbounded

{

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

{

**int** e = myMAXSIZE+**1**;

**for**(**int** i=**1**; i<=myN[**0**]; i++)

**if** (myc[myN[i]] > eps && myN[i] < e) e = myN[i];//eps or 0???????????

**if** (e == myMAXSIZE+**1**) **break**;

**double** delta = oo;

**int** l = myMAXSIZE+**1**;

**for**(**int** i=**1**; i<=myB[**0**]; i++)

**if** (myA[myB[i]][e] > eps)//eps or 0????????????????????????????????

{

**double** temp = myb[myB[i]]/myA[myB[i]][e];

**if** (delta == oo || temp < delta || temp == delta && myB[i] < l)

{

delta = temp;

l = myB[i];

}

}

**if** (l == myMAXSIZE+**1**) **return** **false**;

pivot(l, e);

}

**return** **true**;

}

**bool** initialize()

{

myN[**0**] = myB[**0**] = **0**;

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

myN[++myN[**0**]] = i;

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

myB[++myB[**0**]] = n+i;

myV = **0**;

**return** **true**;

}

**int** main()

{

freopen(**"p10498.in"**, **"r"**, stdin);

freopen(**"a.out"**, **"w"**, stdout);

**while** (read())

{

initialize();

opt();

printf(**"Nasa can spend %d taka.\n"**, (**int**)ceil(myV\*m));

}

}

## 线性规划单纯形法\_武汉网络赛5题

#define REP(i,n) for(int i = 0; i < (int)(n); i++)

#define FOR(i,c) for(\_\_typeof((c).begin()) i = (c).begin(); i != (c).end(); ++i)

#define ALLOF(c) ((c).begin()), ((c).end())

**const** **double** EPS = **1e**-**6**;

**double** eps = **1e**-**6**;

**const** **double** INF = numeric\_limits<**double**>::infinity();

**int** cmp(**double** a) {

**return** a < -eps ? -**1** : a > eps;

}

**typedef** vector<**double**> vector\_t;

**typedef** vector<vector\_t> matrix\_t;

//========A \* x = b , 求 c\*x最小，保证未知数全>=0。 x>=5 等价于 x-y=5,x,y>=0。一些基本变换需要自己保证

//===返回如果.size()==0则没有找到解，否则为一个全>=0的解

vector\_t simplex(matrix\_t A, vector\_t b, vector\_t c) {

**const** **int** n = c.size(), m = b.size();

// modify b to non-negative

REP(i, m) **if** (b[i] < **0**) {

REP(j, n)

A[i][j] \*= -**1**;

b[i] \*= -**1**;

}

// list of base/independent variable ids

vector<**int**> bx(m), nx(n);

REP(i, m)

bx[i] = n+i;

REP(i, n)

nx[i] = i;

// extend A, b

A.resize(m+**2**);

REP(i, m+**2**)

A[i].resize(n+m, **0**);

REP(i, m)

A[i][n+i] = **1**;

REP(i, m) REP(j, n)

A[m][j] += A[i][j];

b.push\_back(accumulate(ALLOF(b), (**double**)**0.0**));

REP(j, n)

A[m+**1**][j] = -c[j];

REP(i, m)

A[m+**1**][n+i] = -INF;

b.push\_back(**0**);

// main optimization

REP(phase, **2**) {

**for**(;;) {

// select an independent variable

**int** ni = -**1**;

REP(i, n)

**if** (A[m][nx[i]] > EPS && (ni < **0** || nx[i] < nx[ni]))

ni = i;

**if** (ni < **0**)

**break**;

**int** nv = nx[ni];

// select a base variable

vector\_t bound(m);

REP(i, m)

bound[i] = (A[i][nv] < EPS ? INF : b[i] / A[i][nv]);

**if** (!(\*min\_element(ALLOF(bound)) < INF))

**return** vector\_t(); // -infinity

**int** bi = **0**;

REP(i, m)

**if** (bound[i] < bound[bi]-EPS || (bound[i] < bound[bi]+EPS && bx[i] < bx[bi]))

bi = i;

// pivot

**double** pd = A[bi][nv];

REP(j, n+m)

A[bi][j] /= pd;

b[bi] /= pd;

REP(i, m+**2**) **if** (i != bi) {

**double** pn = A[i][nv];

REP(j, n+m)

A[i][j] -= A[bi][j] \* pn;

b[i] -= b[bi] \* pn;

}

swap(nx[ni], bx[bi]);

}

**if** (phase == **0** && abs(b[m]) > EPS)

**return** vector\_t(); // no solution

A[m].swap(A[m+**1**]);

swap(b[m], b[m+**1**]);

}

vector\_t x(n+m, **0**);

REP(i, m)

x[bx[i]] = b[i];

x.resize(n);

**return** x;

}

**typedef** vector<**double**> vector\_t;

**typedef** vector<vector\_t> matrix\_t;

matrix\_t A;

vector\_t b;

vector\_t c;

**int** n;

**int** main(){

vector\_t now;

A.clear();

b.clear();

**foru**(i,**1**,n) {

now.clear();

**rep**(j,tot) **if** (dis(a[i].o,d[j])<=r+eps){

now.push\_back(**1**);

}

**else** now.push\_back(**0**);

**foru**(j,**1**,n) **if** (j!=i) now.push\_back(**0**);

**else** now.push\_back(-**1**);

b.push\_back(a[i].times);

A.push\_back(now);

}

c.clear();

**rep**(i,tot) c.push\_back(**1**);

**rep**(i,n) c.push\_back(**0**);

now=simplex(A, b, c);

**double** ans=**0**;

**rep**(i,tot) ans+=now[i];

printf(**"%.2lf\n"**,ans);

}

**return** **0**;

}

# 高精度

## 高精度开根号

**int** l,ans[**5**],cnt;

**bool** flag;

**int** work(**int** o,**char** \*O,**int** I)

{

**char** c,\*D=O ;

**if**(flag)

**return** **0**;

**if**(o>**0**)

{

**if**(flag)

**return** **0**;

**for**(l=**0**;D[l];D[l++]-=**10**)

{

**if**(flag)

**return** **0**;

D[l++]-=**120**;

D[l]-=**110**;

**while**(!work(**0**,O,l))

D[l]+=**20**;

cnt++;

ans[cnt%**3**]=((D[l]+**1032**)/**20**-**'0'**);

**if**(ans[**0**]==ans[**1**]&&ans[**1**]==ans[**2**])

{

printf(**"%d %d\n"**,cnt-**3**,ans[**0**]);

flag=**true**;

**return** **0**;

}

}

}

**else**

{

**if**(flag)

**return** **0**;

c=o+(D[I]+**82**)%**10**-(I>l/**2**)\*(D[I-l+I]+**72**)/**10**-**9**;

D[I]+=I<**0**?**0**:!(o=work(c/**10**,O,I-**1**))\*((c+**999**)%**10**-(D[I]+**92**)%**10**);

}

**return** o;

}

**char** s[**3111**];

**int** t,p[**100**],num;

**int** main()

{

**while**(**1**)

{

flag=**false**;

num=l=cnt=**0**;

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

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

scanf(**"%d"**,&t);

**if**(!t)

**break**;

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

**while**(t)

{

p[++num]=t%**10**;

t/=**10**;

}

s[**0**]=**'0'**;

**for**(**int** i=num;i;i--)

s[num+**1**-i]=p[i]+**'0'**;

**for**(**int** i=**1**;i<=**3000**;i++)

s[num+i]=**'0'**;

**if**(strlen(s)%**2** == **1**)

work(**2**,s+**1**,**0**);

**else**

work(**2**,s,**0**);

}

**return** **0**;

}

## 高精度类

/\*

无符号压位高精度类

要维护长度看规则是否需要判断0

Debug:Yes

\*/

#include<string.h>

#include<stdio.h>

**const** **int** maxleng=**500**;

**class** BigInt//高精度类

{

**private**:

**int** leng;//长度

**int** num[maxleng];//数字

**public**:

BigInt()

{

leng=**1**;

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

}

BigInt(**int** x)

{

leng=**0**;

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

**while**(x)

{

num[leng++]=x%**10000**;

x/=**10000**;

}

**if**(leng==**0**)leng=**1**;

}

**operator** **int**()

{

**int** x=**0**,l=leng-**1**;

**while**(l>=**0**)

{

x=x\***10000**+num[l];

l--;

}

**return** x;

}

**operator** **int**\*()

{

**return** num;

}

**int** length()

{

**return** leng;

}

**void** read()

{

**char** s[maxleng+**1**];

scanf(**"%s"**,s);

**int** l=strlen(s);

leng=**0**;

**for**(**int** i=l-**1**;i>=**0**;)

{

**if**(i>=**0**)num[leng]+=(s[i--]-**'0'**);

**if**(i>=**0**)num[leng]+=(s[i--]-**'0'**)\***10**;

**if**(i>=**0**)num[leng]+=(s[i--]-**'0'**)\***100**;

**if**(i>=**0**)num[leng]+=(s[i--]-**'0'**)\***1000**;

leng++;

}

**if**(leng==**0**)leng=**1**;

}

**void** write()

{

**int** i=leng-**1**;

printf(**"%d"**,num[i]);i--;

**while**(i>=**0**)printf(**"%04d"**,num[i--]);

}

**void** writeln()

{

write();

printf(**"\n"**);

}

**void** getlength()

{

leng=maxleng-**1**;

**while**(num[leng]==**0**&&leng>**0**)leng--;

leng++;

}

**friend** BigInt **operator**+(BigInt a,BigInt b);

**friend** BigInt **operator**+(BigInt a,**int** b);

**friend** BigInt **operator**-(BigInt a,BigInt b);

**friend** BigInt **operator**\*(BigInt a,BigInt b);

**friend** BigInt **operator**\*(BigInt a,**int** b);

**friend** BigInt **operator**/(BigInt a,BigInt b);

**friend** **bool** **operator**<=(BigInt a,BigInt b);

};

BigInt **operator**+(BigInt a,BigInt b)

{

**int** l=a.leng>b.leng?a.leng:b.leng,t=**0**;

BigInt ans;

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

{

ans[i]=(a[i]+b[i]+t)%**10000**;

t=(a[i]+b[i]+t)/**10000**;

}

**while**(t)

{

ans[l++]=t%**10000**;

t/=**10000**;

}

ans.leng=l;

**return** ans;

}

BigInt **operator**+(BigInt a,**int** b)

{

**int** t=**0**;

BigInt ans;

memcpy(ans.num,a.num,**sizeof**(a.num));

ans[t]+=b;

**while**(a[t]>=**10000**)

{

ans[t+**1**]+=ans[t]/**10000**;

ans[t]%=**10000**;

}

ans.getlength();

**return** ans;

}

//a必须大于等于b

BigInt **operator**-(BigInt a,BigInt b)

{

**int** l=a.leng;

BigInt ans;

memcpy(ans.num,a.num,**sizeof**(a.num));

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

{

ans[i]-=b[i];

**if**(ans[i]<**0**)

{

ans[i]+=**10000**;

ans[i+**1**]--;

}

}

ans.getlength();

**return** ans;

}

//一种看起来快点的乘法

BigInt **operator**\*(BigInt a,BigInt b)

{

**int** la=a.leng,lb=b.leng,t,p;

BigInt ans;

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

{

t=**0**;

**for**(**int** j=**0**;j<lb;j++)

{

p=(ans[i+j]+a[i]\*b[j]+t)/**10000**;

ans[i+j]=(ans[i+j]+a[i]\*b[j]+t)%**10000**;

t=p;

}

p=i+lb;

**if**(t)

{

ans[p]+=t;

**while**(ans[p]>=**10000**)

{

ans[p+**1**]+=ans[p]/**10000**;

ans[p]%=**10000**;

p++;

}

}

}

ans.getlength();

**return** ans;

}

//单精度乘以高精度

//本来不想写的 但是由于更不想写恶心除法

BigInt **operator**\*(BigInt a,**int** b)

{

**int** t=**0**,p=a.leng;

BigInt ans;

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

{

ans[i]=(a[i]\*b+t)%**10000**;

t=(a[i]\*b+t)/**10000**;

}

**while**(t)

{

ans[p++]=t%**10000**;

t/=**10000**;

}

ans.getlength();

**return** ans;

}

**bool** **operator**<=(BigInt a,BigInt b)

{

**if**(a.leng!=b.leng)**return** a.leng<b.leng;

**for**(**int** i=a.leng-**1**;i>=**0**;i--)

**if**(a[i]!=b[i])**return** a[i]<b[i];

**return** **true**;

}

//除法就只有乱写了,很少写除法的

BigInt **operator**/(BigInt a,BigInt b)

{

**int** la=a.leng,lb=b.leng;

BigInt ans,p;

**for**(**int** i=la-**1**;i>=**0**;i--)

{

p=p\***10000**+a[i];

**for**(**int** j=**13**;j>=**0**;j--)

{

**if**(b\*(**1**<<j)<=p)

{

p=(p-b\*(**1**<<j));

ans[i]+=(**1**<<j);

}

}

}

ans.getlength();

**return** ans;

}

BigInt a,b;

**int** main()

{

a.read();b.read();

(a/b).writeln();

}

# 数学

## 牛顿迭代开根号

**unsigned** **long** **long** sqrtll**(unsigned** **long** **long** n**)**

**{**

**if** **(**n **==** 0**)**

**return** 0**;**

**unsigned** **long** **long** x **=** 1LLU **<<** **(**63 **-** \_\_builtin\_clzll**(**n**)** **>>** 1**);**

**unsigned** **long** **long** xx **=** **-**1**;**

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

**unsigned** **long** **long** nx **=** x **+** n **/** x **>>** 1**;**

**if** **(**nx **==** xx**)**

**return** min**(**x**,** nx**);**

xx **=** x**;**

x **=** nx**;**

**}**

**}**

## 有多少个点在多边形内

//rn中的标号必须逆时针给出。一开始要旋转坐标，保证同一个x值上只有一个点。正向减点，//反向加点。num[i][j]=num[j][i]=严格在这根线下方的点。 on[i][j]=on[j][i]=严格//在线段上的点，包括两个端点。若有回边的话注意计算onit的方法，不要多算了线段上的点。

int ans=0,z,onit=0,lows=0;

rep(z,t) {

i=rn[z]; j=rn[z+1]; onit+=on[i][j]-1;

if (a[j].x>a[i].x){ans-=num[i][j];lows+=on[i][j]-1;}

else ans+=num[i][j];

}//ans-lows+1 is inside. 只会多算一次正向上的点（除去最左和最右的点）。Lows只算了除开最左边的点，但会多算最右边的点，所以要再加上1.

printf("%d\n",ans-lows+1 + onit);

## 斜线下格点统计

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

**//计算for (int i=0;i<n;++i) s+=floor((a+b\*i)/m)**

**//n,m,a,b>0**

**{**

// printf("%lld %lld %lld %lld\n", n, a, b, m);

**if(**b **==** 0**){**

**return** n **\*** **(**a **/** m**);**

**}**

**if(**a **>=** m**){**

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

**}**

**if(**b **>=** m**){**

**return** **(**n **-** 1**)** **\*** n **/** 2 **\*** **(**b **/** m**)** **+** solve**(**n**,** a**,** b **%** m**,** m**);**

**}**

LL q **=** **(**a **+** b **\*** n**)** **/** m**;**

**return** solve**(**q**,** **(**a **+** b **\*** n**)** **%** m**,** m**,** b**);**

**}**

## 大整数相乘取摸

**typedef** **long** **long** Int64;

Int64 mod\_mul(Int64 x,Int64 y,Int64 n){

Int64 d = (Int64)((**long** **double**)x \* y / n);

d = x \* y - n \* d;

**while** (d < **0**) d += n;

**while** (d >= n) d -= n;

**return** d;

}

## 素数判定

**int** strong\_pseudo\_primetest(**long** **long** n,**int** base) {

**long** **long** n2=n-**1**,res;

**int** s; s=**0**;

**while**(n2%**2**==**0**) n2>>=**1**,s++;

res=powmod(base,n2,n);

**if**((res==**1**)||(res==n-**1**)) **return** **1**;

s--;

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

res=mulmod(res,res,n);

**if**(res==n-**1**) **return** **1**;

s--;

}

**return** **0**; // n is not a strong pseudo prime

}

**int** isprime(**long** **long** n) {

**if**(n<**2**) **return** **0**; **if**(n<**4**) **return** **1**;

**if**(strong\_pseudo\_primetest(n,**2**)==**0**) **return** **0**;

**if**(strong\_pseudo\_primetest(n,**3**)==**0**) **return** **0**;

**if**(n<**1373653LL**) **return** **1**;

**if**(strong\_pseudo\_primetest(n,**5**)==**0**) **return** **0**;

**if**(n<**25326001LL**) **return** **1**;

**if**(strong\_pseudo\_primetest(n,**7**)==**0**) **return** **0**;

**if**(n==**3215031751LL**) **return** **0**;

**if**(n<**25000000000LL**) **return** **1**;

**if**(strong\_pseudo\_primetest(n,**11**)==**0**) **return** **0**;

**if**(n<**2152302898747LL**) **return** **1**;

**if**(strong\_pseudo\_primetest(n,**13**)==**0**) **return** **0**;

**if**(n<**3474749660383LL**) **return** **1**;

**if**(strong\_pseudo\_primetest(n,**17**)==**0**) **return** **0**;

**if**(n<**341550071728321LL**) **return** **1**;

**if**(strong\_pseudo\_primetest(n,**19**)==**0**) **return** **0**;

**if**(strong\_pseudo\_primetest(n,**23**)==**0**) **return** **0**;

**if**(strong\_pseudo\_primetest(n,**29**)==**0**) **return** **0**;

**if**(strong\_pseudo\_primetest(n,**31**)==**0**) **return** **0**;

**if**(strong\_pseudo\_primetest(n,**37**)==**0**) **return** **0**;

**return** **1**;

}

## Pollard-Rho

**inline** LL pollardRho**(**LL n**,**LL c**)**//return a non-trival factor of n, otherwise return n

**{**

//if (n-1==0) while(1);

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

LL head**=**1**,**tail**=**2**;**

**while** **(**1**){**

x**=**mod\_mul**(**x**,**x**,**n**);**

x**+=**c**;**

**if** **(**x**>=**n**)** x**-=**n**;**

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

LL d**=**\_\_gcd**(**myAbs**(**x**-**y**),**n**);**

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

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

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

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

**}**

**}**

**}**

**inline** **void** factor**(**LL n**)**//factorize n

**{**

**if** **(**n**<=**1**)** **return;**

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

**if** **(**N**>**100**)** **while** **(**1**);**

fac**[**N**++]=**n**;**

**return;**

**}**

//if (n-1==0) while(1);

LL p**=**n**;**

**while** **(**p**>=**n**)** p**=**pollardRho**(**n**,**rand**()%(**n**-**1**)+**1**);**

factor**(**n**/**p**);**

factor**(**p**);**

**}**

## O(p)求1..p-1的逆元

**void** solve **(int** m**)** **{**

**int** inv**[**m**];**

inv**[**1**]** **=** 1**;**

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

inv**[**i**]** **=** **((long** **long)(**m **-** m **/** i**)** **\*** inv**[**m **%** i**])** **%** m**;**

**}**

**}**

## 广义离散对数（不需要互质）

**void** extendedGcd **(int** a**,** **int** b**,** **long** **long** **&**x**,** **long** **long** y**)** **{**

**if** **(**b**)** **{**

extendedGcd**(**b**,** a **%** b**,** y**,** x**);**

y **-=** a **/** b **\*** x**;**

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

x **=** a**;**

y **=** 0**;**

**}**

**}**

**int** inverse **(int** a**,** **int** m**)** **{**

**long** **long** x**,** y**;**

extendedGcd**(**a**,** m**,** x**,** y**);**

**return** **(**x **%** m **+** m**)** **%** m**;**

**}**

// a ^ x = b (mod m)

**int** solve **(int** a**,** **int** b**,** **int** m**)** **{**

**int** tmp **=** 1 **%** m**,** c**;**

map**<int,** **int>** s**;**

**if** **(**tmp **==** b**)** **{**

**return** 0**;**

**}**

**for** **(int** i **=** 1**;** i **<=** 50**;** **++** i**)** **{**

tmp **=** **((long** **long)**tmp **\*** a**)** **%** m**;**

**if** **(**tmp **==** b**)** **{**

**return** i**;**

**}**

**}**

**int** x\_0 **=** 0**,** d **=** 1 **%** m**;**

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

tmp **=** gcd**(**a**,** m**);**

**if** **(**tmp **==** 1**)** **{**

**break;**

**}**

x\_0 **++;**

d **=** **((long** **long)**d **\*** **(**a **/** tmp**))** **%** m**;**

**if** **(**b **%** tmp**)** **{**

**return** **-**1**;**

**}**

b **/=** tmp**;**

m **/=** tmp**;**

**}**

b **=** **((long** **long)**b **\*** inverse**(**d**,** m**))** **%** m**;**

c **=** **int(**ceil**(**sqrt**(**m**)));**

s**.**clear**();**

tmp **=** b**;**

**int** tmpInv **=** intverse**(**a**,** m**);**

**for** **(int** i **=** 0**;** i **!=** c**;** **++** i**)** **{**

**if** **(**s**.**find**(**tmp**)** **==** s**.**end**())** **{**

s**[**tmp**]** **=** i**;**

**}**

tmp **=** **((long** **long)**tmp **\*** tmpInv**)** **%** m**;**

**}**

tmp **=** 1**;**

**for** **(int** i **=** 0**;** i **!=** c**;** **++** i**)** **{**

tmp **=** **((long** **long)**tmp **\*** a**)** **%** m**;**

**}**

**int** ans **=** 1**;**

**for** **(int** i **=** 0**;** i **!=** c**;** **++** i**)** **{**

**if** **(**s**.**find**(**ans**)** **!=** s**.**end**())** **{**

**return** x\_0 **+** i **\*** c **+** s**.**find**(**ans**)->**second**;**

**}**

ans **=** **((long** **long)**ans **\*** tmp**)** **%** m**;**

**}**

**return** **-**1**;**

**}**

## n次剩余

**const** **int** LimitSave=**100000**;

**long** **long** P,K,A;

vector<**long** **long**>ans;

**struct** tp{

**long** **long** expo,res;

}data[LimitSave+**100**];

**long** **long** \_mod(**long** **long** a, **long** **long** mo) {

a=a%mo;

**if** (a<**0**) a+=mo;

**return** a;

}

**long** **long** powers(**long** **long** a , **long** **long** K , **long** **long** modular) {

**long** **long** res;

res=**1**;

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

**if** (K & **1**) res=\_mod(res\*a,modular);

K=K>>**1**;

a=\_mod(a\*a , modular);

}

**return** res;

}

**long** **long** get\_originroot(**long** **long** p) {

**long** **long** primes[**100**];

**long** **long** tot,i,tp,j;

i=**2**; tp=P-**1**; tot=**0**;

**while** (i\*i<=P-**1**) {

**if** (\_mod(tp,i)==**0**) {

tot++;

primes[tot]=i;

**while** (\_mod(tp,i)==**0**) tp/=i;

}

i++;

}

**if** (tp!=**1**) {tot++; primes[tot]=tp;}

i=**2**;

**bool** ok;

**while** (**1**) {

ok=**true**;

**foru**(j,**1**,tot) {

**if** (powers(i, (P-**1**)/primes[j] , P)==**1**) {

ok=**false**;

**break**;

}

}

**if** (ok) **break**;

i++;

}

**return** i;

}

**bool** euclid\_extend(**long** **long** A ,**long** **long** B ,**long** **long** C ,**long** **long** &x, **long** **long** &y, **long** **long** &gcdnum) {

**long** **long** t;

**if** (A==**0**) {

gcdnum = B;

**if** (\_mod(C , B) ==**0**) {

x=**0**; y=C/B;

**return** **true**;

}

**else** **return** **false**;

}

**else** **if** (euclid\_extend(\_mod(B , A) , A , C , y , t , gcdnum)) {

x = t - **int**(B / A) \* y;

**return** **true**;

}

**else** **return** **false**;

}

**long** **long** Division(**long** **long** A, **long** **long** B, **long** **long** modular) {

**long** **long** gcdnum,K,Y;

euclid\_extend(modular, B,A,K,Y,gcdnum);

Y=\_mod(Y,modular);

**if** (Y<**0**) Y+=modular;

**return** Y;

}

**bool** Binary\_Search(**long** **long** key, **long** **long** &position) {

**long** **long** start,stop;

start=**1**; stop=LimitSave;

**bool** flag=**true**;

**while** (start<=stop) {

position=(start+stop)/**2**;

**if** (data[position].res==key) **return** **true**;

**else**

**if** (data[position].res<key) start=position+**1**;

**else** stop=position-**1**;

}

**return** **false**;

}

**bool** compareab(**const** tp &a, **const** tp &b) {

**return** a.res<b.res;

}

**long** **long** get\_log(**long** **long** root, **long** **long** A, **long** **long** modular) {

**long** **long** i,j,times,XD,XT,position;

**if** (modular-**1**<LimitSave) {

**long** **long** now=**1**;

**foru**(i,**0**,modular-**1**) {

**if** (now==A) {

**return** i;

}

now=\_mod(now \* root , modular);

}

}

data[**1**].expo=**0**; data[**1**].res=**1**;

**foru**(i,**1**,LimitSave-**1**) {

data[i+**1**].expo=i;

data[i+**1**].res=\_mod(data[i].res\*root,modular);

}

sort(data+**1**,data+LimitSave+**1**,compareab);

times=powers(root,LimitSave,modular);

j=**0**;

XD=**1**;

**while** (**1**) {

XT=Division(A,XD,modular);

**if** (Binary\_Search(XT,position)) {

**return** j+data[position].expo;

}

j=j+LimitSave;

XD=\_mod(XD\*times,modular);

}

}

**void** work\_ans() {

ans.clear();

**if** (A==**0**) {

ans.push\_back(**0**);

**return**;

}

**long** **long** root,logs,delta,deltapower,now,gcdnum, i,x,y;

root=get\_originroot(P);

logs=get\_log(root,A,P);

**if** (euclid\_extend(K,P-**1**,logs,x,y,gcdnum)) {

delta=(P-**1**)/gcdnum;

x=\_mod(x,delta);

**if** (x<**0**) x+=delta;

now=powers(root,x,P);

deltapower=powers(root,delta,P);

**while** (x<P-**1**) {

ans.push\_back(now);

now=\_mod(now\*deltapower,P);

x=x+delta;

}

}

**if** (ans.size()>**1**)

sort(ans.**begin**(),ans.**end**());

}

**int** main(){

freopen(**"in.txt"**,**"r"**,stdin);

// freopen("output.txt","w",stdout);

**int** i,j,k,test,cases=**0**;

scanf(**"%d"**,&test);

prepare();

**while** (test) {

test--;

cin>>P>>K>>A;

A=A % P; **//x^K mod P = A**

cases++;

printf(**"Case #%d:\n"**,cases);

work\_ans();

}

**return** **0**;

}

## 求ax取模n同余b的所有解及中国剩余定理

i,d,x,y,a,b,t,e,n:longint;

ans:array[**1..1000**]of longint;

function extended\_gcd(a,b:longint;var x,y:longint):longint;

**begin**

**if** b=**0** then **begin**

extended\_gcd:=a;

x:=**1**;

y:=**0**;

**end**

**else** **begin**

extended\_gcd:=extended\_gcd(b,a mod b,x,y);

t:=x;

x:=y;

y:=t-(a div b)\*y;

**end**;

**end**;

**begin**

readln(a,b,n);

d:=extended\_gcd(a,n,x,y);

**if** b mod d<>**0** then **begin** writeln(**'NO answer'**); exit;**end**;

e:=x\*(b div d) mod n;

**if** e<**0** then e:=e+n;

**for** i:=**0** to d-**1** **do**

ans[i+**1**]:=(e+i\*n div d) mod n;

**for** i:=**0** to d-**1** **do**

write(ans[i+**1**],**' '**);

**end**.

**void** mod\_equation\_solver(T a, T b, T n, T ans[], **int** &ansl) {

T d, x, y, e, i;

d = extgcd(a, n, x, y);

**if** (b % d != **0**) { ansl = **0**; **return**; }

e = x \* (b / d) % n;

**for** (T i = **0**; i < d; i ++) {

ans[i] = (e + i \* (n / d)) % n;

**if** (ans[i] < **0**) ans[i] += n;

}

ansl = d;

}

T china(T b[], T w[], **int** len) {

T d, ans, x, y, m, n;

ans = **0**; n = **1**; **for** (**int** i=**0**;i<len;i++) n \*= w[i];

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

m = n / w[i]; d = extgcd(w[i], m, x, y); ans = (ans + y \* m \* b[i]) % n;

}

**return** (n + ans % n) % n;

}

## Pell方程求解

#define sqr(x) ((x)\*(x))

#define maxn 50

#define UL unsigned long long

UL A,B,p[maxn],q[maxn],a[maxn],g[maxn],h[maxn];

**int** main()

{

**int** n;

**for** (**int** test=**1**;scanf(**"%d"**,&n) && n;++test)

{

printf(**"Case %d: "**,test);

n\*=**2**;

**if** (fabs(sqrt(n)-floor(sqrt(n)+**1e**-**7**))<=**1e**-**7**)

{

**int** a=(**int**)(floor(sqrt(n)+**1e**-**7**));

printf(**"%d %d\n"**,a,**1**);

}**else**

{

//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**];

**if** (sqr((UL)(p[i]))-n\*sqr((UL)(q[i]))==**1**)

{

A=p[i];B=q[i];

**break**;

}

}

cout << A << **' '** << B <<endl;

}

}

**return** **0**;

}

## 莫比乌斯函数以及gcd=1的对数

#define maxn 10000000

**int** div[maxn+**5**],sum[maxn+**5**],p[**1000000**],len;

**long** **long** ans;

**inline** **void** prepare()

{

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

**for** (**int** i=**2**;i<=maxn;++i)

**if** (!div[i])

{

div[i]=i;

p[len++]=i;

**if** (i>maxn/i) **continue**;

**for** (**int** j=i\*i;j<=maxn;j+=i)

**if** (!div[j]) div[j]=i;

}

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

{

**int** cnt=**0**,last=**0**;

**for** (**int** j=i;j>**1**;last=div[j],j/=div[j])

{

**if** (div[j]==last)

{

sum[i]=**0**;

**goto** Break;

}

cnt^=**1**;

}

**if** (cnt) sum[i]=-**1**;

**else** sum[i]=**1**;

Break:;

sum[i]+=sum[i-**1**];

}

}

//计算莫比乌斯函数，及其前缀和

//复杂度O(nlogn)

**inline** **void** calc(**int** a,**int** b)

{

**for** (**int** i=**1**,j,p,q;i<=a;i=j+**1**)

{

p=a/i;

q=b/i;

j=b/q;

**if** (a<p\*j) j=a/p;

ans+=(**long** **long**)(sum[j]-sum[i-**1**])\*p\*q;

}

}

//求1..a和1..b中有多少对的gcd=1

//复杂度O(sqrt(a+b))

**int** main()

{

prepare();

**int** T;

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

{

**int** a,b;

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

**if** (a>b) { **int** t=a;a=b;b=t; }

**int** limit=a;

**if** (b<limit) limit=b;

ans=**0**;

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

{

**if** (p[i]>limit) **break**;

calc(a/p[i],b/p[i]);

}

printf(**"%lld\n"**,ans);

}

**return** **0**;

}

## 二次剩余

/\*

a\*x^2+b\*x+c==0 (mod P)

求0..P-1的根

\*/

#include <cstdio>

#include <cstdlib>

#include <ctime>

#define sqr(x) ((x)\*(x))

**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***;

}

# Tips

## 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)

## 牛顿迭代

　x1=x0-func(x0)/func1(x0);//进行牛顿迭代计算

我们要求f(x)=0的解。func(x)为原方程，func1为原方程的导数方程

## 求某年某月某日是星期几

**int** whatday(**int** d, **int** m, **int** y) { //day month year

**int** ans;

**if** (m == **1** || m == **2**) { m += **12**; y --; }

**if** ((y < **1752**) || (y == **1752** && m < **9**)||(y == **1752** && m == **9** && d < **3**))

ans = (d + **2**\*m + **3**\*(m+**1**)/**5** + y + y/**4** +**5**) % **7**;

**else** ans = (d + **2**\*m + **3**\*(m+**1**)/**5** + y + y/**4** - y/**100** + y/**400**)%**7**;

**return** ans;

}

## 图同构hash

枚举点a，迭代K次后求得的就是a点所对应的hash值。

其中K、A、B、C、D、P为hash参数，可自选。

## 综合

lucas 定理 C(n,m) % p(p是素数） = n和m都化成p进制数， 累乘C(ni,mi)

Lucas(n,m,p)=c(n%p,m%p)\* Lucas(n/p,m/p,p) ,Lucas(x,0,p)=1;

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

定理1：最小覆盖数 = 最大匹配数

定理2：最大独立集S 与 最小覆盖集T 互补。

算法：

1. 做最大匹配，没有匹配的空闲点∈S

2. 如果u∈S那么u的临点必然属于T

3. 如果一对匹配的点中有一个属于T那么另外一个属于S

4. 还不能确定的，把左子图的放入S，右子图放入T

算法结束

p是素数且2^p－1的是素数,n不超过258的全部梅森素数终于确定，是

　　n=2,3,5,7,13,17,19,31,61,89,107,127

有上下界网络流，求可行流部分，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。

原因：感觉上是在第一遍已经把内部都消耗光了，第二遍是必须的流量。

路径剖分，取节点数最多的子树伸出来的路径。

序列差分表由它的第0行确定，也就是原序列，但同时也可以由第0条对角线上的元素确定。

换句话说，由差分表的第0条对角线就可以确定原序列。有这样两个公式：

原序列为h\_i，第0条对角线为c\_o,c\_1,…,c\_p,0,0,0,…

则h\_n = c\_0\*C(n,0)+c\_1\*C(n,1)+…+c\_p\*C(n,p)，

Σh\_k(k=0..n) = c\_0\*C(n+1,1)+c\_2\*(n+1,2)+…+c\_p\*C(n+1,p+1)。

记住这两个公式，差分表（的第0条对角线）就变得非常有用了。

平面图一定存在一个度小于等于5的点,且可以四染色

（ 欧拉公式 ） 设 G 是连通的平面图，n , m, r 分别是其顶点数、边数和面数，n-m+r=2

极大平面图 m≤3n-6

gcd(2^(a)-1,2^(b)-1)=(2^gcd(a,b))-1.

中国剩余定理：(牛书,P230)

m1,m2......mk两两互素.则下面的同余方程:

x=a1(mod m1)

x=a2(mod m2)

x=a3(mod m3)

.....

在0<=x<=M=m1\*m2\*m3..\*mk内有唯一解.

公式=e1\*a1+e2\*a2+e3\*a3+e4\*a4....就是方程组的一个解.

(附注:x mod 3=a1, x mod 5=a2 , x mod 7=a3.的做法是

x=(5\*7\*a1)+(3\*7\*a2)+(3\*5\*a3)

x= x mod 105.

这个是这个公式的特殊情况,因为ei=大Mi=大M/小mi).

Fibonacci数

gcd(Fn，Fm)=Fgcd(n,m) (牛书，P228)

即是说，两个fibonacci数的最大公约数，肯定是个fibonacci数

Fibonacci质数（和前面所有的Fibonacci数互质）（大多已经是质数了，可能有BUG吧，不确定）

定理：如果a是b的倍数，那么Fa是Fb的倍数。

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

平方数的和是平方数的问题。

a[0] := 0;

s := 0;

for i := 1 to n - 2 do

begin

a[i] := a[i - 1] + 1;

s := s + sqr(a[i]);

end;

{======s + sqr(a[n-1]) + sqr(a[n]) = k^2=======}

a[n - 1] := a[n - 2];

repeat

a[n - 1] := a[n - 1] + 1;

until odd(s + sqr(a[n - 1])) and (a[n - 1] > 2);

a[n] := (s + sqr(a[n - 1]) - 1) shr 1;

知道s和a[n-1]后，直接求了a[n].神奇了点。

其实。有当n为奇数：n^2 + ((n^2 - 1) div 2)^2 = ((n^2 + 1) div 2)^2

所以有3 4-- 5 12 -- 7 24 -- 9 40 -- 11 60 ....

a=k\*(s^2 - t^2);

b=2\*k\*s\*t

c=k(s^2 + t^2);

则c^2=a^2+b^2 完全的公式

定义：一颗树T的质心m，就是将m及m连出的边都删除之后，剩下的森林中，每颗树的节点数<=|V(T)|/2。任何树都有质心，并且可以在O(N)的时间内求出。

求的方法如下：以任意一个节点作为T的根，作后序遍历。对于节点v，若是叶子节点，令C(v)=1，否则C(v)=子树和 。遍历过程中，第一次出现C(v)>=|V(T)|/2，那么v就是质心。

质心是个好东西，也许以后对不是二叉树的树进行分治之类的算法，考虑强行把令质心作为根，可以得到二分法一样的时间复杂度。

重加权的方法如下：增加人工结点s，直接到所有点连一条弧，权均为0，然

后以s为起点运行bellman-ford，求出dist(v)。如果有负权圈则退出，否则对于原图

中的每个条边(u,v)，设新权w'(u,v)=dist(u)+w(u,v)-dist(v)，则它是非负的

k-连通(k-connected) ：对于任意一对结点都至少存在结点各不相同的k条

路。

点连通度(vertex connectivity) ：把图变成非连通图所需删除的最少点

数。

这两个定义是互通的，因为我们有：

Whitney定理：一个图是k-连通的当且仅当它的点连通度至少为k。

Fermat分解算法从t = n^1/2开始，依次检查t2-n; (t+1)2-n; (t+2)2-n … ，直到

出现一个平方数y，由于t2-y2 = n，因此分解得n = (t-y)(t+y)。显然，当两个因数很

5.1 数论基础243

接近时这个方法能很快找到结果，但如果遇到一个素数，则需要检查(n + 1)/2 - n^1/2个

整数，比试除法还慢得多。虽然方法并不是很有效，但是为我们提供了一个思路。Lehman算法

# Stl使用

## java\_scl

import java.io.\*;

import java.util.\*;

import java.math.\*;

import **static** java.lang.Math.\*;

**public** **class** main{

**public** **static** StringTokenizer st;

**public** **static** DataInputStream in;

**public** **static** PrintStream out;

**private** **static** **int** nextInt() throws Exception{

**for** (;st.countTokens()==**0**;) st=**new** StringTokenizer(in.readLine());

**return** Integer.valueOf(st.nextToken());

}

**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;

}

**public** **static** **void** main(String args[]) throws Exception{

in=**new** DataInputStream(System.in);

out=**new** PrintStream(**new** BufferedOutputStream(System.out));

st=**new** StringTokenizer(in.readLine());

out.close();

}

}

//Arrays

**int** a[]=**new** **int**[**10**];

Arrays.fill(a,**0**);

Arrays.sort(a);

//String

String s;

s.charAt(**int** i);

s.compareTo(String b);

s.compareToIgnoreCase();

s.contains(String b);

s.length();

s.substring(**int** l,**int** r);

//BigInteger

BigInteger a;

a.abs();

a.add(b);

a.bitLength();

a.subtract(b);

a.divide(b);

a.remainder(b);

a.divideAndRemainder(b);

a.modPow(b,c);//a^b mod c;

a.pow(**int**);

a.multiply(b);

a.compareTo(b);

a.gcd(b);

a.intValue();

a.longValue();

a.isProbablePrime(**int** certainty);//(1 - 1/2^certainty).

a.nextProbablePrime();

a.shiftLeft(**int**);

a.valueOf();

//BigDecimal

**static** **int** ROUND\_CEILING,ROUND\_DOWN,ROUND\_FLOOR,

ROUND\_HALF\_DOWN,ROUND\_HALF\_EVEN,ROUND\_HALF\_UP,ROUND\_UP;

a.divide(BigDecimal b,**int** scale,**int** round\_mode);

a.doubleValue();

a.movePointLeft(**int** i);

a.pow(**int**);

a.setScale(**int** scale,**int** round\_mode);

a.stripTrailingZeros();

//StringBuilder

StringBuilder sb=**new** StringBuilder();

sb.append(elem);

out.println(sb);

//StringTokenizer

StringTokenizer st=**new** StringTokenizer(in.readLine());

st.countTokens();

st.hasMoreTokens();

st.nextToken();

//Vector

a.add(elem);

a.add(index,elem);

a.clear();

a.elementAt(index);

a.isEmpty();

a.remove(index);

a.set(index,elem);

a.size();

//Queue

a.add(elem);

a.peek();//front

a.poll();//pop

//Integer Double Long

## cpp

ch1=cin.peek(); // 查看下一个字符，单不在流中剔除

cin.putback(**char**)

next\_permutation(p,p+n); // 求 p 的下一个排列，如果得到头排列则 false , 否则 true

prev\_permutation(p,p+n); // 求 p 的前一个排列，如果得到尾排列则 false，否则 true

#include <iomanip>

#include <iostream>

cout.setf(ios::fixed,ios::floatfield);

cout.precision(**3**);

cout<<**double**(u)<<endl;

**int** x=**30**, y=**300**, z=**1024**;

cout<<x<<**' '**<<y<<**' '**<<z<<endl; //按十进制输出

cout.setf(ios::showbase | ios::uppercase); //设置基指示符输出和数值中的字母大写输出

cout<<x<<**' '**<<y<<**' '**<<z<<endl;

cout.unsetf(ios::showbase | ios::uppercase); //取消基指示符输出和数值中的字母大写输出

cout.setf(ios::oct); //设置为八进制输出,此设置不取消一直有效

cout<<x<<**' '**<<y<<**' '**<<z<<endl; //按八进制输出

cout.setf(ios::showbase | ios::uppercase); //设置基指示符输出和数值中的字母大写输出

cout<<x<<**' '**<<y<<**' '**<<z<<endl;

cout.unsetf(ios::showbase | ios::uppercase); //取消基指示符输出和数值中的字母大写输出

cout.unsetf(ios::oct); //取消八进制输出设置，恢复按十进制输出

cout.setf(ios::hex); //设置为十六进制输出

cout<<x<<**' '**<<y<<**' '**<<z<<endl;

cout.setf(ios::showbase | ios::uppercase); //设置基指示符输出和数值中的字母大写输出

cout<<x<<**' '**<<y<<**' '**<<z<<endl;

cout.unsetf(ios::showbase | ios::uppercase); //取消基指示符输出和数值中的字母大写输出

cout.unsetf(ios::hex); //取消十六进制输出设置，恢复按十进制输出

cout<<x<<**' '**<<y<<**' '**<<z<<endl;

// cin>>hex>>i>>j;

//unique ly is size

**bool** Cmp\_Dbl(**double** a, **double** b) {

**return** Sign(a - b) < **0**;

}

**bool** Equ\_Dbl(**double** a, **double** b) {

**return** !Sign(a - b); //=== （a = b） return true;

}

sort(yar, yar + ly, Cmp\_Dbl);

ly = unique(yar, yar + ly, Equ\_Dbl) - yar;

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

scanf(**"%d%d%d%d"**,x1+i,y1+i,x2+i,y2+i);

XS.push\_back(x1[i]);

XS.push\_back(x2[i]);

}

sort(XS.**begin**(),XS.**end**()),XS.erase(unique(XS.**begin**(),XS.**end**()),XS.**end**());

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

x1[i]=lower\_bound(XS.**begin**(),XS.**end**(),x1[i])-XS.**begin**();

x2[i]=lower\_bound(XS.**begin**(),XS.**end**(),x2[i])-XS.**begin**();

}

//priority\_queue<int> h;

#include <queue>

priority\_queue<**int**> h;

**while** (!h.empty()) h.pop();

**if** (!h.empty()) **if** (now<h.top()) {bt=**false**; **break**;}

h.push(x);

priority\_queue<string , vector<string> , greater< vector<string>::value\_type> > hmin; //===小根堆

priority\_queue<string , vector<string> , less< vector<string>::value\_type> > hmax; //===大根堆

**struct** Tmax{

string data;

**int** num;

**bool** **operator**<(**const** Tmax &a) **const** {

**return** data>a.data;

}

};

**struct** Tmin{

string data;

**bool** **operator**<(**const** Tmin &a) **const**{ //must have "const" 用来保证不会改变本身的值

**return** data<a.data;

}

};

priority\_queue<Tmin > hmax;

priority\_queue<Tmax > hmin;

# 积分表

## 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**

## 平方剩余求解

给定a和素数p，求所有的满足.

Legendre符号：

Legendre符号是积性函数，即 ,若p为奇素数，

即当且仅当

若p为奇素数，则 即当且仅当 时，2为模p的二次剩余。

若pq为奇素数，且，则 。

**引理** [1,p-1]区间中最多有两个根,且满足

求解步骤如下：

1. 若 p=2, 则x=a；否则，转2.

2. 若 则转3；否则，无解。

3. 若 否则，转4

4.1 找一个最小的使得 。

4.2 令

4.3 反复做4.3.A和4.3.B，直到i为奇数。

4.3.A 。

4.3.B 若 。

4.4 最后 。

## 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。

**生成树的计数**

完全图的生成树个数

任意图的生成树个数： 生成树计数行列式tab[i][i] = Di，Di为i的度数tab[i][j] = −k, k为i和j之间的边数。任去一行一列之后的行列式。

## 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

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