*First edited in 2013/10/07*

*Second edited in 2014/2/20*

*Third edited in 2014/6/17*

*The Final Edition in 2014/10/14*

**ACM Template**

Written by asdw12345

Referred to SYL

**目录**

[几何 5](#_Toc405152827)

[多边形 5](#_Toc405152828)

[多边形切割（半平面交） 8](#_Toc405152829)

[浮点几何函数库 11](#_Toc405152830)

[几何公式 17](#_Toc405152831)

[面积 19](#_Toc405152832)

[面积并与面积交 20](#_Toc405152833)

[球面 21](#_Toc405152834)

[三角形 22](#_Toc405152835)

[三维几何 26](#_Toc405152836)

[体积并与体积交 35](#_Toc405152837)

[凸包(graham) 35](#_Toc405152838)

[三维凸包 39](#_Toc405152839)

[网格 45](#_Toc405152840)

[圆 46](#_Toc405152841)

[整数函数 50](#_Toc405152842)

[注意 52](#_Toc405152843)

[其他 53](#_Toc405152844)

[进制转换 53](#_Toc405152845)

[统计二进制数1的个数和 53](#_Toc405152846)

[分数 54](#_Toc405152847)

[矩阵 55](#_Toc405152848)

[日期 57](#_Toc405152849)

[线性方程组(gauss) 58](#_Toc405152850)

[归并排序求逆序对（O（n log n）） 61](#_Toc405152851)

[快速幂 62](#_Toc405152852)

[母函数（不限制个数，O( n^3 )） 63](#_Toc405152853)

[母函数（限制个数，O( n^4 )） 64](#_Toc405152854)

[稳定婚姻问题（Gale\_Shapley算法） 65](#_Toc405152855)

[数论 66](#_Toc405152856)

[阶乘最后非零位 66](#_Toc405152857)

[模线性方程(组) 66](#_Toc405152858)

[质数表 69](#_Toc405152859)

[质数随机判定(miller\_rabin) 69](#_Toc405152860)

[分解质因数 70](#_Toc405152861)

[最大公约数（GCD与exGCD） 71](#_Toc405152862)

[欧拉函数 72](#_Toc405152863)

[离散对数解方程A^x=B（mod C） 72](#_Toc405152864)

[康托展开 74](#_Toc405152865)

[数值计算 75](#_Toc405152866)

[定积分计算(Romberg) 75](#_Toc405152867)

[自适应Simpson求积分 76](#_Toc405152868)

[多项式求根(牛顿法) 77](#_Toc405152869)

[周期性方程(追赶法) 79](#_Toc405152870)

[组合 79](#_Toc405152871)

[排列组合生成 79](#_Toc405152872)

[生成gray码 81](#_Toc405152873)

[置换(polya定理) 81](#_Toc405152874)

[整数划分（五边形定义与其他） 82](#_Toc405152875)

[字典序全排列与序号的转换 83](#_Toc405152876)

[字典序组合与序号的转换 83](#_Toc405152877)

[组合公式 84](#_Toc405152878)

[求和公式（k = 1……n） 85](#_Toc405152879)

[大数高精度计算 86](#_Toc405152880)

[Java相关 93](#_Toc405152881)

[BigInteger输入输出外挂 93](#_Toc405152882)

[常用函数与其他 100](#_Toc405152883)

[常用函数 100](#_Toc405152884)

[格式化输入 101](#_Toc405152885)

[数位dp模板 101](#_Toc405152886)

[特殊算法汇总 103](#_Toc405152887)

[雅可比（Jacobi）迭代解线性方程组 103](#_Toc405152888)

[几何图形的反演变换 104](#_Toc405152889)

# 几何

## 多边形

#include <stdlib.h>

#include <math.h>

#define MAXN 1000

#define offset 10000

#define eps 1e-8

#define EE 2.718281828459

#define PI acos(-1.0)

#define zero(x) (( (x)>0 ? (x):-(x) )<eps)//|x|<eps，即为0

#define \_sign(x) ((x)>eps?1:((x)<-eps?2:0))//正为1，负为2，0为0

struct point

**{**

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

**};**

struct line

**{**

point a**,**b**;**

**};**

int sign**(**double x**)**

**{**

**if(**fabs**(**x**)<**eps**)** **return** 0**;**

**else** **return** x**>**0**?**1**:-**1**;**

**}**

//计算叉乘，正顺负逆0共线

double xmult**(**point p1**,**point p2**,**point p0**)**

**{**

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

**}**

//判定凸多边形,顶点按顺时针或逆时针给出,允许相邻边共线

int is\_convex**(**int n**,**point**\*** p**)**

**{**

int i**,**s**[**3**]=** **{**1**,**1**,**1**};**

**for** **(**i**=**0**;** i**<**n **&&** s**[**1**]|**s**[**2**];** i**++)** //全同拐向是为凸多边形

s**[**\_sign**(**xmult**(** p**[(**i**+**1**)%**n**]** **,** p**[(**i**+**2**)%**n**]** **,**p**[**i**]** **))]=**0**;**

**return** s**[**1**]|**s**[**2**];**

**}**

//判定凸多边形,顶点按顺时针或逆时针给出,不允许相邻边共线

int is\_convex\_v2**(**int n**,**point**\*** p**)**

**{**

int i**,**s**[**3**]=** **{**1**,**1**,**1**};**

**for** **(**i**=**0**;** i**<**n**&&**s**[**0**]&&**s**[**1**]|**s**[**2**];** i**++)**

s**[**\_sign**(**xmult**(**p**[(**i**+**1**)%**n**],**p**[(**i**+**2**)%**n**],**p**[**i**]))]=**0**;**

**return** s**[**0**]&&**s**[**1**]|**s**[**2**];**

**}**

//判点在凸多边形内或多边形边上,顶点按顺时针或逆时针给出

int inside\_convex**(**point q**,**int n**,**point**\*** p**)**

**{**

int i**,**s**[**3**]=** **{**1**,**1**,**1**};**

**for** **(**i**=**0**;** i**<**n**&&**s**[**1**]|**s**[**2**];** i**++)** //点q在多边形内时全同拐向

s**[**\_sign**(**xmult**(**p**[(**i**+**1**)%**n**],**q**,**p**[**i**]))]=**0**;**

**return** s**[**1**]|**s**[**2**];**

**}**

**//判点在凸多边形内,顶点按顺时针或逆时针给出,在多边形边上返回0**

//（可参见点在三角形内的判断方法）

int inside\_convex\_v2**(**point q**,**int n**,**point**\*** p**)**

**{**

int i**,**s**[**3**]=** **{**1**,**1**,**1**};**

**for** **(**i**=**0**;** i**<**n**&&**s**[**0**]&&**s**[**1**]|**s**[**2**];** i**++)**

s**[**\_sign**(**xmult**(**p**[(**i**+**1**)%**n**],**q**,**p**[**i**]))]=**0**;**

**return** s**[**0**]&&**s**[**1**]|**s**[**2**];**

**}**

**//判点在任意多边形内,顶点按顺时针或逆时针给出**

//on\_edge表示点在多边形边上时的返回值,offset为多边形坐标上限

#define zero(x) (((x)>0?(x):-(x))<eps)

int inside\_polygon(point p1,vector<point> poly,int on\_edge=1)

{

point p2;

int n=poly.size();

int i=0,j,k;

int cnt;

while(i<n)

{

p2.x=rand()+offset;

p2.y=rand()+offset;

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

{

if(zero(xmult(p1,poly[i],poly[(i+1)%n])) &&

(poly[i].x-p1.x)\*(poly[(i+1)%n].x-p1.x)<EPS &&

(poly[i].y-p1.y)\*(poly[(i+1)%n].y-p1.y)<EPS)

return on\_edge;

else if(zero(xmult(p1,p2,poly[i]))) break;

elseif(xmult(p1,poly[i],p2)\*xmult(p1,poly[(i+1)%n],p2)<-EPS && xmult(poly[i],p1,poly[(i+1)%n])\*xmult(poly[i],p2,poly[(i+1)%n])<-EPS)

cnt++;

}

}

return cnt&1;

}

inline int opposite\_side**(**point p1**,**point p2**,**point l1**,**point l2**)**

**{**

**return** xmult**(**l1**,**p1**,**l2**)\***xmult**(**l1**,**p2**,**l2**)<-**eps**;**//p1、p2在l1、l2的两旁

**}**

inline int dot\_online\_in**(**point p**,**point l1**,**point l2**)**

**{**

//点在线段l1、l2上（共线，x、y值介于l1、l2间）

**return** zero**(**xmult**(**p**,**l1**,**l2**))**

**&&** **(**l1**.**x**-**p**.**x**)\*(**l2**.**x**-**p**.**x**)<**eps

**&&** **(**l1**.**y**-**p**.**y**)\*(**l2**.**y**-**p**.**y**)<**eps**;**

**}**

//判线段在任意多边形内,顶点按顺时针或逆时针给出,与边界相交返回1

int inside\_polygon**(**point l1**,**point l2**,**int n**,**point**\*** p**)**

**{**

point t**[**MAXN**],**tt**;**

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

**if** **(!**inside\_polygon**(**l1**,**n**,**p**)||!**inside\_polygon**(**l2**,**n**,**p**))**//两点必须在多边形内

**return** 0**;**

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

**if** **(**opposite\_side**(**l1**,**l2**,**p**[**i**],**p**[(**i**+**1**)%**n**])** **&&**

opposite\_side**(**p**[**i**],**p**[(**i**+**1**)%**n**],**l1**,**l2**))**//不能与某边相交

**return** 0**;**

**else** **if** **(**dot\_online\_in**(**l1**,**p**[**i**],**p**[(**i**+**1**)%**n**]))**//l1在边上

t**[**k**++]=**l1**;**

**else** **if** **(**dot\_online\_in**(**l2**,**p**[**i**],**p**[(**i**+**1**)%**n**]))**//l2在边上

t**[**k**++]=**l2**;**

**else** **if** **(**dot\_online\_in**(**p**[**i**],**l1**,**l2**))**//p[i]在线段l1、l2上

t**[**k**++]=**p**[**i**];**

//要求t集合的任两点的中点在多边形内（凹多边形要考虑此部分）

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

**for** **(**j**=**i**+**1**;** j**<**k**;** j**++)**

**{**

tt**.**x**=(**t**[**i**].**x**+**t**[**j**].**x**)/**2**;**

tt**.**y**=(**t**[**i**].**y**+**t**[**j**].**y**)/**2**;**

**if** **(!**inside\_polygon**(**tt**,**n**,**p**))**

**return** 0**;**

**}**

**return** 1**;**

**}**

//求两直线的交点

point intersection**(**line u**,**line v**)**

**{**

point ret**=**u**.**a**;**

double t**=((**u**.**a**.**x**-**v**.**a**.**x**)\*(**v**.**a**.**y**-**v**.**b**.**y**)-(**u**.**a**.**y**-**v**.**a**.**y**)\*(**v**.**a**.**x**-**v**.**b**.**x**))**

**/((**u**.**a**.**x**-**u**.**b**.**x**)\*(**v**.**a**.**y**-**v**.**b**.**y**)-(**u**.**a**.**y**-**u**.**b**.**y**)\*(**v**.**a**.**x**-**v**.**b**.**x**));**

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

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

**return** ret**;**

**}**

//三角形的重心

point barycenter**(**point a**,**point b**,**point c**)**

**{**

line u**,**v**;**//两中线

u**.**a**.**x**=(**a**.**x**+**b**.**x**)/**2**;**

u**.**a**.**y**=(**a**.**y**+**b**.**y**)/**2**;**

u**.**b**=**c**;**

v**.**a**.**x**=(**a**.**x**+**c**.**x**)/**2**;**

v**.**a**.**y**=(**a**.**y**+**c**.**y**)/**2**;**

v**.**b**=**b**;**

**return** intersection**(**u**,**v**);**

**}**

//多边形重心，位置为sum(Si\*xi)/sum(Si),Si为各三角形面积，xi为个三角形重心

point barycenter**(**int n**,**point**\*** p**)**

**{**

point ret**,**t**;**

double t1**=**0**,**t2**;**

int i**;**

ret**.**x**=**ret**.**y**=**0**;**

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

//p[0]与其他点连线划分为多个三角形

**if** **(**fabs**(**t2**=**xmult**(**p**[**0**],**p**[**i**],**p**[**i**+**1**]))>**eps**)**

**{**

t**=**barycenter**(**p**[**0**],**p**[**i**],**p**[**i**+**1**]);**

ret**.**x**+=**t**.**x**\***t2**;**

ret**.**y**+=**t**.**y**\***t2**;**

t1**+=**t2**;**

**}**

**if** **(**fabs**(**t1**)>**eps**)**

ret**.**x**/=**t1**,**ret**.**y**/=**t1**;**

**return** ret**;**

**}**

## 多边形切割（半平面交）

//可用于半平面交

#define MAXN 100

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

struct point

**{**

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

**};**

double xmult**(**point p1**,**point p2**,**point p0**)**

**{**

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

**}**

int same\_side**(**point p1**,**point p2**,**point l1**,**point l2**)**

**{**

**return** xmult**(**l1**,**p1**,**l2**)\***xmult**(**l1**,**p2**,**l2**)>**eps**;**

**}**

point intersection**(**point u1**,**point u2**,**point v1**,**point v2**)**

**{**

point ret**=**u1**;**

double t**=((**u1**.**x**-**v1**.**x**)\*(**v1**.**y**-**v2**.**y**)-(**u1**.**y**-**v1**.**y**)\*(**v1**.**x**-**v2**.**x**))**

**/((**u1**.**x**-**u2**.**x**)\*(**v1**.**y**-**v2**.**y**)-(**u1**.**y**-**u2**.**y**)\*(**v1**.**x**-**v2**.**x**));**

ret**.**x**+=(**u2**.**x**-**u1**.**x**)\***t**;**

ret**.**y**+=(**u2**.**y**-**u1**.**y**)\***t**;**

**return** ret**;**

**}**

//将多边形沿l1,l2确定的直线切割在side侧切割,保证l1,l2,side不共线

void polygon\_cut**(**int**&** n**,**point**\*** p**,**point l1**,**point l2**,**point side**)**

**{**

point pp**[**100**];**

int m**=**0**,**i**;**

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

**{**

**if** **(**same\_side**(**p**[**i**],**side**,**l1**,**l2**))**

pp**[**m**++]=**p**[**i**];**

**if** **(!**same\_side**(**p**[**i**],**p**[(**i**+**1**)%**n**],**l1**,**l2**)&&!(**zero**(**xmult**(**p**[**i**],**l1**,**l2**))&&**zero**(**xmult**(**p**[(**i**+**1**)%**n**],**l1**,**l2**))))**

pp**[**m**++]=**intersection**(**p**[**i**],**p**[(**i**+**1**)%**n**],**l1**,**l2**);**

**}**

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

**if** **(!**i**||!**zero**(**pp**[**i**].**x**-**pp**[**i**-**1**].**x**)||!**zero**(**pp**[**i**].**y**-**pp**[**i**-**1**].**y**))**

p**[**n**++]=**pp**[**i**];**

**if** **(**zero**(**p**[**n**-**1**].**x**-**p**[**0**].**x**)&&**zero**(**p**[**n**-**1**].**y**-**p**[**0**].**y**))**

n**--;**

**if** **(**n**<**3**)**

n**=**0**;**

**}**

//半平面交 second temple

//初始化为整个平面，O(n^2)算法，返回的pol是半平面上的点#include<cmath>

#include<complex>

#define INF 0x3f3f3f3f

#define eps 1e-8

#define MAXN 1000+5

**typedef** long long LL**;**

**typedef** complex**<**double**>** point**;**

**typedef** vector**<**point**>** polygon**;**

**typedef** pair**<**point**,**point**>**line**;**

**typedef** point Vector**;**

#define x real()

#define y imag()

int sign**(**double xx**)**

**{**

**if(**fabs**(**xx**)<**eps**)** **return** 0**;**

**else** **return** xx**>**0**?**1**:-**1**;**

**}**

double xmult**(**Vector p1**,**Vector p2**)**

**{**

**return** p1**.**x**\***p2**.**y**-**p1**.**y**\***p2**.**x**;**

**}**

double xmult**(**point p0**,**point p1**,**point p2**)**

**{**

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

**}**

double Area**(**polygon convex**)**

**{**

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

double sum**=**0**;**

int i**;**

**for(**i**=**0**;** i**<**n**;** i**++)** sum**+=**xmult**(**convex**[**i**%**n**],**convex**[(**i**+**1**)%**n**]);**

**return** fabs**(**sum**)/**2.0**;**

**}**

point intersect**(**line L1**,**line L2**)**

**{**

point a1**=**L1**.**first**,** a2**=**L1**.**second**;**

point b1**=**L2**.**first**,** b2**=**L2**.**second**;**

Vector a**=**a2**-**a1**,** b**=**b2**-**b1**,** c**=**b1**-**a1**;**

**return** a1**+**a**\***xmult**(**b**,**c**)/**xmult**(**b**,**a**);**

**}**

polygon plane\_intersect**(**polygon p**,**line L**)**

**{**

polygon ret**;**

point p1**=**L**.**first**,** p2**=**L**.**second**;**

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

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

**{**

double a**=**xmult**(**p1**,**p**[**i**],**p2**);**

double b**=**xmult**(**p1**,**p**[(**i**+**1**)%**n**],**p2**);**

**if(**sign**(**a**)>=**0**)** ret**.**push\_back**(**p**[**i**]);**

//如果多边形的点是顺时针给出的，上一句中的>=0要改成<=0 **if(**sign**(**a**\***b**)<**0**)**

**{**

line W**=**make\_pair**(**p**[**i**],**p**[(**i**+**1**)%**n**]);**

ret**.**push\_back**(**intersect**(**L**,**W**));**

**}**

**}**

**return** ret**;**

**}**

line l**[**MAXN**];**

//void HPI() 这一部分一般可以不加

void HPI**()**

**{**

polygon pol**;**

pol**.**push\_back**(**point**(-**INF**,-**INF**));**

pol**.**push\_back**(**point**(-**INF**,+**INF**));**

pol**.**push\_back**(**point**(+**INF**,-**INF**));**

pol**.**push\_back**(**point**(+**INF**,+**INF**));**

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

**{**

pol**=**plane\_intersect**(**pol**,**l**[**i**]);**

**if(**sign**(**Area**(**pol**))==**0**)** **break;**

**}**

**}**

int main**()**

**{**

int T**;**

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

**while(**T**--)**

**{**

int n**;**

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

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

point pp**;**

polygon pol**;**

pol**.**clear**();**

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

**{**

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

pol**.**push\_back**(**pp**);**

**}**

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

**{**

l**[**i**]=**make\_pair**(**pol**[**i**],**pol**[(**i**+**1**)%**n**]);**

**}**

int flag**=**1**;**

**for(**i**=**0**;** i**<**n**;** i**++)** pol**=**plane\_intersect**(**pol**,**l**[**i**]);**

**}**

**return** 0**;**

**}**

## 浮点几何函数库

#include <math.h>

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

struct point

**{**

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

**};**

struct line

**{**

point a**,**b**;**

**};**

//计算cross product (P1-P0)x(P2-P0)

double xmult**(**point p1**,**point p2**,**point p0**)**

**{**

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

**}**

double xmult**(**double x1**,**double y1**,**double x2**,**double y2**,**double x0**,**double y0**)**

**{**

**return** **(**x1**-**x0**)\*(**y2**-**y0**)-(**x2**-**x0**)\*(**y1**-**y0**);**

**}**

//计算dot product (P1-P0).(P2-P0)

double dmult**(**point p1**,**point p2**,**point p0**)**

**{**

**return** **(**p1**.**x**-**p0**.**x**)\*(**p2**.**x**-**p0**.**x**)+(**p1**.**y**-**p0**.**y**)\*(**p2**.**y**-**p0**.**y**);**

**}**

double dmult**(**double x1**,**double y1**,**double x2**,**double y2**,**double x0**,**double y0**)**

**{**

**return** **(**x1**-**x0**)\*(**x2**-**x0**)+(**y1**-**y0**)\*(**y2**-**y0**);**

**}**

//两点距离

double veclen**(**point p1**,**point p2**)**

**{**

**return** sqrt**((**p1**.**x**-**p2**.**x**)\*(**p1**.**x**-**p2**.**x**)+(**p1**.**y**-**p2**.**y**)\*(**p1**.**y**-**p2**.**y**));**

**}**

double veclen**(**double x1**,**double y1**,**double x2**,**double y2**)**

**{**

**return** sqrt**((**x1**-**x2**)\*(**x1**-**x2**)+(**y1**-**y2**)\*(**y1**-**y2**));**

**}**

//判三点共线

int dots\_inline**(**point p1**,**point p2**,**point p3**)**

**{**

**return** zero**(**xmult**(**p1**,**p2**,**p3**));**

**}**

int dots\_inline**(**double x1**,**double y1**,**double x2**,**double y2**,**double x3**,**double y3**)**

**{**

**return** zero**(**xmult**(**x1**,**y1**,**x2**,**y2**,**x3**,**y3**));**

**}**

//判点是否在线段上,包括端点

int dot\_online\_in**(**point p**,**line l**)**

**{**

**return** zero**(**xmult**(**p**,**l**.**a**,**l**.**b**))&&(**l**.**a**.**x**-**p**.**x**)\*(**l**.**b**.**x**-**p**.**x**)<**eps**&&(**l**.**a**.**y**-**p**.**y**)\*(**l**.**b**.**y**-**p**.**y**)<**eps**;**

**}**

int dot\_online\_in**(**point p**,**point l1**,**point l2**)**

**{**

**return** zero**(**xmult**(**p**,**l1**,**l2**))&&(**l1**.**x**-**p**.**x**)\*(**l2**.**x**-**p**.**x**)<**eps**&&(**l1**.**y**-**p**.**y**)\*(**l2**.**y**-**p**.**y**)<**eps**;**

**}**

int dot\_online\_in**(**double x**,**double y**,**double x1**,**double y1**,**double x2**,**double y2**)**

**{**

**return** zero**(**xmult**(**x**,**y**,**x1**,**y1**,**x2**,**y2**))&&(**x1**-**x**)\*(**x2**-**x**)<**eps**&&(**y1**-**y**)\*(**y2**-**y**)<**eps**;**

**}**

//判点是否在线段上,不包括端点

int dot\_online\_ex**(**point p**,**line l**)**

**{**

**return** dot\_online\_in**(**p**,**l**)&&(!**zero**(**p**.**x**-**l**.**a**.**x**)||!**zero**(**p**.**y**-**l**.**a**.**y**))&&(!**zero**(**p**.**x**-**l**.**b**.**x**)||!**zero**(**p**.**y**-**l**.**b**.**y**));**

**}**

int dot\_online\_ex**(**point p**,**point l1**,**point l2**)**

**{**

**return** dot\_online\_in**(**p**,**l1**,**l2**)&&(!**zero**(**p**.**x**-**l1**.**x**)||!**zero**(**p**.**y**-**l1**.**y**))&&(!**zero**(**p**.**x**-**l2**.**x**)||!**zero**(**p**.**y**-**l2**.**y**));**

**}**

int dot\_online\_ex**(**double x**,**double y**,**double x1**,**double y1**,**double x2**,**double y2**)**

**{**

**return** dot\_online\_in**(**x**,**y**,**x1**,**y1**,**x2**,**y2**)&&(!**zero**(**x**-**x1**)||!**zero**(**y**-**y1**))&&(!**zero**(**x**-**x2**)||!**zero**(**y**-**y2**));**

**}**

//判两点在线段同侧,点在线段上返回0

int same\_side**(**point p1**,**point p2**,**line l**)**

**{**

**return** xmult**(**l**.**a**,**p1**,**l**.**b**)\***xmult**(**l**.**a**,**p2**,**l**.**b**)>**eps**;**

**}**

int same\_side**(**point p1**,**point p2**,**point l1**,**point l2**)**

**{**

**return** xmult**(**l1**,**p1**,**l2**)\***xmult**(**l1**,**p2**,**l2**)>**eps**;**

**}**

//判两点在线段异侧,点在线段上返回0

int opposite\_side**(**point p1**,**point p2**,**line l**)**

**{**

**return** xmult**(**l**.**a**,**p1**,**l**.**b**)\***xmult**(**l**.**a**,**p2**,**l**.**b**)<-**eps**;**

**}**

int opposite\_side**(**point p1**,**point p2**,**point l1**,**point l2**)**

**{**

**return** xmult**(**l1**,**p1**,**l2**)\***xmult**(**l1**,**p2**,**l2**)<-**eps**;**

**}**

// 点关于直线的对称点 ,缺点：用了斜率

// 也可以利用"点到直线上的最近点"来做，避免使用斜率。

point symmetric\_point**(**point p1**,** point l1**,** point l2**)**

**{**

point ret**;**

**if** **(**l1**.**x **>** l2**.**x **-** eps **&&** l1**.**x **<** l2**.**x **+** eps**)**

**{**

ret**.**x **=** **(**2 **\*** l1**.**x **-** p1**.**x**);**

ret**.**y **=** p1**.**y**;**

**}**

**else**

**{**

double k **=** **(**l1**.**y **-** l2**.**y **)** **/** **(**l1**.**x **-** l2**.**x**);**

ret**.**x **=** **(**2**\***k**\***k**\***l1**.**x **+** 2**\***k**\***p1**.**y **-** 2**\***k**\***l1**.**y **-** k**\***k**\***p1**.**x **+** p1**.**x**)** **/** **(**1 **+** k**\***k**);**

ret**.**y **=** p1**.**y **-** **(**ret**.**x **-** p1**.**x **)** **/** k**;**

**}**

**return** ret**;**

**}**

//判两直线平行

int parallel**(**line u**,**line v**)**

**{**

**return** zero**((**u**.**a**.**x**-**u**.**b**.**x**)\*(**v**.**a**.**y**-**v**.**b**.**y**)-(**v**.**a**.**x**-**v**.**b**.**x**)\*(**u**.**a**.**y**-**u**.**b**.**y**));**

**}**

int parallel**(**point u1**,**point u2**,**point v1**,**point v2**)**

**{**

**return** zero**((**u1**.**x**-**u2**.**x**)\*(**v1**.**y**-**v2**.**y**)-(**v1**.**x**-**v2**.**x**)\*(**u1**.**y**-**u2**.**y**));**

**}**

//判两直线垂直

int perpendicular**(**line u**,**line v**)**

**{**

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

**}**

int perpendicular**(**point u1**,**point u2**,**point v1**,**point v2**)**

**{**

**return** zero**((**u1**.**x**-**u2**.**x**)\*(**v1**.**x**-**v2**.**x**)+(**u1**.**y**-**u2**.**y**)\*(**v1**.**y**-**v2**.**y**));**

**}**

//判两线段相交,包括端点和部分重合

int intersect\_in**(**line u**,**line v**)**

**{**

**if** **(!**dots\_inline**(**u**.**a**,**u**.**b**,**v**.**a**)||!**dots\_inline**(**u**.**a**,**u**.**b**,**v**.**b**))**

**return** **!**same\_side**(**u**.**a**,**u**.**b**,**v**)&&!**same\_side**(**v**.**a**,**v**.**b**,**u**);**

**return** dot\_online\_in**(**u**.**a**,**v**)||**dot\_online\_in**(**u**.**b**,**v**)||**dot\_online\_in**(**v**.**a**,**u**)||**dot\_online\_in**(**v**.**b**,**u**);**

**}**

int intersect\_in**(**point u1**,**point u2**,**point v1**,**point v2**)**

**{**

**if** **(!**dots\_inline**(**u1**,**u2**,**v1**)||!**dots\_inline**(**u1**,**u2**,**v2**))**

**return** **!**same\_side**(**u1**,**u2**,**v1**,**v2**)&&!**same\_side**(**v1**,**v2**,**u1**,**u2**);**

**return** dot\_online\_in**(**u1**,**v1**,**v2**)||**dot\_online\_in**(**u2**,**v1**,**v2**)||**dot\_online\_in**(**v1**,**u1**,**u2**)||**dot\_online\_in**(**v2**,**u1**,**u2**);**

**}**

//判两线段相交,不包括端点和部分重合

int intersect\_ex**(**line u**,**line v**)**

**{**

**return** opposite\_side**(**u**.**a**,**u**.**b**,**v**)&&**opposite\_side**(**v**.**a**,**v**.**b**,**u**);**

**}**

int intersect\_ex**(**point u1**,**point u2**,**point v1**,**point v2**)**

**{**

**return** opposite\_side**(**u1**,**u2**,**v1**,**v2**)&&**opposite\_side**(**v1**,**v2**,**u1**,**u2**);**

**}**

//计算两直线交点,注意事先判断直线是否平行!

//线段交点请另外判线段相交(同时还是要判断是否平行!)

point intersection**(**line u**,**line v**)**

**{**

point ret**=**u**.**a**;**

double t**=((**u**.**a**.**x**-**v**.**a**.**x**)\*(**v**.**a**.**y**-**v**.**b**.**y**)-(**u**.**a**.**y**-**v**.**a**.**y**)\*(**v**.**a**.**x**-**v**.**b**.**x**))**

**/((**u**.**a**.**x**-**u**.**b**.**x**)\*(**v**.**a**.**y**-**v**.**b**.**y**)-(**u**.**a**.**y**-**u**.**b**.**y**)\*(**v**.**a**.**x**-**v**.**b**.**x**));**

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

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

**return** ret**;**

**}**

point intersection**(**point u1**,**point u2**,**point v1**,**point v2**)**

**{**

point ret**=**u1**;**

double t**=((**u1**.**x**-**v1**.**x**)\*(**v1**.**y**-**v2**.**y**)-(**u1**.**y**-**v1**.**y**)\*(**v1**.**x**-**v2**.**x**))**

**/((**u1**.**x**-**u2**.**x**)\*(**v1**.**y**-**v2**.**y**)-(**u1**.**y**-**u2**.**y**)\*(**v1**.**x**-**v2**.**x**));**

ret**.**x**+=(**u2**.**x**-**u1**.**x**)\***t**;**

ret**.**y**+=(**u2**.**y**-**u1**.**y**)\***t**;**

**return** ret**;**

**}**

//点到直线上的最近点

point ptoline**(**point p**,**line l**)**

**{**

point t**=**p**;**

t**.**x**+=**l**.**a**.**y**-**l**.**b**.**y**,**t**.**y**+=**l**.**b**.**x**-**l**.**a**.**x**;**

**return** intersection**(**p**,**t**,**l**.**a**,**l**.**b**);**

**}**

point ptoline**(**point p**,**point l1**,**point l2**)**

**{**

point t**=**p**;**

t**.**x**+=**l1**.**y**-**l2**.**y**,**t**.**y**+=**l2**.**x**-**l1**.**x**;**

**return** intersection**(**p**,**t**,**l1**,**l2**);**

**}**

//点到直线距离

double disptoline**(**point p**,**line l**)**

**{**

**return** fabs**(**xmult**(**p**,**l**.**a**,**l**.**b**))/**veclen**(**l**.**a**,**l**.**b**);**

**}**

double disptoline**(**point p**,**point l1**,**point l2**)**

**{**

**return** fabs**(**xmult**(**p**,**l1**,**l2**))/**veclen**(**l1**,**l2**);**

**}**

double disptoline**(**double x**,**double y**,**double x1**,**double y1**,**double x2**,**double y2**)**

**{**

**return** fabs**(**xmult**(**x**,**y**,**x1**,**y1**,**x2**,**y2**))/**veclen**(**x1**,**y1**,**x2**,**y2**);**

**}**

//点到线段上的最近点

point ptoseg**(**point p**,**line l**)**

**{**

point t**=**p**;**

t**.**x**+=**l**.**a**.**y**-**l**.**b**.**y**,**t**.**y**+=**l**.**b**.**x**-**l**.**a**.**x**;**

**if** **(**xmult**(**l**.**a**,**t**,**p**)\***xmult**(**l**.**b**,**t**,**p**)>**eps**)**

**return** veclen**(**p**,**l**.**a**)<**veclen**(**p**,**l**.**b**)?**l**.**a**:**l**.**b**;**

**return** intersection**(**p**,**t**,**l**.**a**,**l**.**b**);**

**}**

point ptoseg**(**point p**,**point l1**,**point l2**)**

**{**

point t**=**p**;**

t**.**x**+=**l1**.**y**-**l2**.**y**,**t**.**y**+=**l2**.**x**-**l1**.**x**;**

**if** **(**xmult**(**l1**,**t**,**p**)\***xmult**(**l2**,**t**,**p**)>**eps**)**

**return** veclen**(**p**,**l1**)<**veclen**(**p**,**l2**)?**l1**:**l2**;**

**return** intersection**(**p**,**t**,**l1**,**l2**);**

**}**

//点到线段距离

double disptoseg**(**point p**,**line l**)**

**{**

point t**=**p**;**

t**.**x**+=**l**.**a**.**y**-**l**.**b**.**y**,**t**.**y**+=**l**.**b**.**x**-**l**.**a**.**x**;**

**if** **(**xmult**(**l**.**a**,**t**,**p**)\***xmult**(**l**.**b**,**t**,**p**)>**eps**)**

**return** veclen**(**p**,**l**.**a**)<**veclen**(**p**,**l**.**b**)?**veclen**(**p**,**l**.**a**):**veclen**(**p**,**l**.**b**);**

**return** fabs**(**xmult**(**p**,**l**.**a**,**l**.**b**))/**veclen**(**l**.**a**,**l**.**b**);**

**}**

double disptoseg**(**point p**,**point l1**,**point l2**)**

**{**

point t**=**p**;**

t**.**x**+=**l1**.**y**-**l2**.**y**,**t**.**y**+=**l2**.**x**-**l1**.**x**;**

**if** **(**xmult**(**l1**,**t**,**p**)\***xmult**(**l2**,**t**,**p**)>**eps**)**

**return** veclen**(**p**,**l1**)<**veclen**(**p**,**l2**)?**veclen**(**p**,**l1**):**veclen**(**p**,**l2**);**

**return** fabs**(**xmult**(**p**,**l1**,**l2**))/**veclen**(**l1**,**l2**);**

**}**

//矢量V以P为顶点逆时针旋转angle并放大scale倍

point rotate**(**point v**,**point p**,**double angle**,**double scale**)**

**{**

point ret**=**p**;**

v**.**x**-=**p**.**x**,**v**.**y**-=**p**.**y**;**

p**.**x**=**scale**\***cos**(**angle**);**

p**.**y**=**scale**\***sin**(**angle**);**

ret**.**x**+=**v**.**x**\***p**.**x**-**v**.**y**\***p**.**y**;**

ret**.**y**+=**v**.**x**\***p**.**y**+**v**.**y**\***p**.**x**;**

**return** ret**;**

**}**

//极角排序（叉积版本）

//设点集为p[0]~[n-1],cnt初始化为p[0]，每一次排序后更新cnt

bool cmp\_angle**(**point a**,**point b**)**

**{**

int res **=** cross**(**cnt**,**a**,**b**);**

**if(**res **>** 0**)** **return** **true;** //点按顺时针排序

// if(res < 0) return true //点按逆时针排序

**else** **if(**res **<** 0**)** **return** **false;**

**else**

**{**

double dis1 **=** dis**(**a**,**cnt**);**

double dis2 **=** dis**(**b**,**cnt**);**

**if(**dis1 **<** dis2**)** **return** **true;**

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

**}**

**}**

cnt **=** p**[**0**];**

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

**{**

sort**(**p**+**i**,**p**+**n**,**cmp\_angle**);**

cnt **=** p**[**i**];**

**}**

## 几何公式

**三角形:**

1. 半周长 P=(a+b+c)/2

2. 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))

3. 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2

4. 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)

5. 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)

6. 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)

=4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)

=Ptan(A/2)tan(B/2)tan(C/2)

7. 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))

**四边形:**

D1,D2为对角线,M对角线中点连线,A为对角线夹角

1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2

2. S=D1D2sin(A)/2

(以下对圆的内接四边形)

3. ac+bd=D1D2

4. S=sqrt((P-a)(P-b)(P-c)(P-d)),P为半周长

**正n边形:（R为外接圆半径,r为内切圆半径**）

1. 中心角 A=2PI/n

2. 内角 C=(n-2)PI/n

3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)

4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))

**圆:**

1. 弧长 l=rA

2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)

3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2

4. 扇形面积 S1=rl/2=r^2A/2

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

**棱柱:**

1. 体积 V=Ah,A为底面积,h为高

2. 侧面积 S=lp,l为棱长,p为直截面周长

3. 全面积 T=S+2A

**棱锥:**

1. 体积 V=Ah/3,A为底面积,h为高

(以下对正棱锥)

2. 侧面积 S=lp/2,l为斜高,p为底面周长

3. 全面积 T=S+A

**棱台:**

1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2为上下底面积,h为高

(以下为正棱台)

2. 侧面积 S=(p1+p2)l/2,p1.p2为上下底面周长,l为斜高

3. 全面积 T=S+A1+A2

**圆柱:**

1. 侧面积 S=2PIrh

2. 全面积 T=2PIr(h+r)

3. 体积 V=PIr^2h

**圆锥:**

1. 母线 l=sqrt(h^2+r^2)

2. 侧面积 S=PIrl

3. 全面积 T=PIr(l+r)

4. 体积 V=PIr^2h/3

**圆台:**

1. 母线 l=sqrt(h^2+(r1-r2)^2)

2. 侧面积 S=PI(r1+r2)l

3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)

4. 体积 V=PI(r1^2+r2^2+r1r2)h/3

**球:**

1. 全面积 T=4PIr^2

2. 体积 V=4PIr^3/3

**球台:**

1. 侧面积 S=2PIrh

2. 全面积 T=PI(2rh+r1^2+r2^2)

3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6

**球扇形:**

1. 全面积 T=PIr(2h+r0),h为球冠高,r0为球冠底面半径

2. 体积 V=2PIr^2h/3

**四面体：**

//U,V,W,u,v,w是其六条棱，U,V,W构成三角形，大小写字母互为对棱

体积公式：

V=sqrt(ABCD)/(192uvw)

其中：

A=s-2a , B=s-2b , C=s-2c , D=s-2d;

a=sqrt(xYZ) , b=sqrt(XyZ) , c=sqrt(XYz) , d=sqrt(xyz);

s=a+b+c+d;

X=(w-U+v)(w+U+v) , x=(U-v+w)(v-w+U);

Y=(u-V+w)(V+w+u) , y=(V-w+u)(w-u+V);

Z=(v-W+u)(W+u+v) , z=(W-u+v)(u-v+W);

## 面积

#include <math.h>

struct point

**{**

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

**};**

//计算cross product (P1-P0)x(P2-P0)

double xmult**(**point p1**,**point p2**,**point p0**)**

**{**

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

**}**

double xmult**(**double x1**,**double y1**,**double x2**,**double y2**,**double x0**,**double y0**)**

**{**

**return** **(**x1**-**x0**)\*(**y2**-**y0**)-(**x2**-**x0**)\*(**y1**-**y0**);**

**}**

//计算三角形面积,输入三顶点

double area\_triangle**(**point p1**,**point p2**,**point p3**)**

**{**

**return** fabs**(**xmult**(**p1**,**p2**,**p3**))/**2**;**

**}**

double area\_triangle**(**double x1**,**double y1**,**double x2**,**double y2**,**double x3**,**double y3**)**

**{**

**return** fabs**(**xmult**(**x1**,**y1**,**x2**,**y2**,**x3**,**y3**))/**2**;**

**}**

//计算三角形面积,输入三边长

double area\_triangle**(**double a**,**double b**,**double c**)**

**{**

double s**=(**a**+**b**+**c**)/**2**;**

**return** sqrt**(**s**\*(**s**-**a**)\*(**s**-**b**)\*(**s**-**c**));**

**}**

//计算多边形面积,顶点按顺时针或逆时针给出

double area\_polygon**(**int n**,**point**\*** p**)**

**{**

double s1**=**0**,**s2**=**0**;**

int i**;**

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

s1**+=**p**[(**i**+**1**)%**n**].**y**\***p**[**i**].**x**,**s2**+=**p**[(**i**+**1**)%**n**].**y**\***p**[(**i**+**2**)%**n**].**x**;**

**return** fabs**(**s1**-**s2**)/**2**;**

**}**

## 面积并与面积交

//计算三角形与圆的（有向）面积交，简单多边形与圆的面积交可以讲多边形划分成多个三角形再求交（注意最后总面积要取fabs）

#define PI acos(-1.0)

#define EPS 1e-8

int sign**(**double x**)**

**{**

**if(**fabs**(**x**)<**EPS**)** **return** 0**;**

**else** **return** x**>**0**?**1**:-**1**;**

**}**

struct Point

**{**

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

Point**()** **{}**

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

**};**

double xmult**(**Point a**,** Point b**,** Point c**)**

**{**

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

**}**

double dot**(**Point a**,**Point b**,**Point c**)**

**{**

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

**}**

double veclen**(**Point a**,**Point b**)**

**{**

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

**}**

double squre**(**double x**)**

**{**

**return** x**\***x**;**

**}**

//三角形与圆的面积交，其中三角形三点为a、b、cir，cir为圆心，r为半径

double Triangle\_Circle\_Intersect\_Area**(**Point a**,**Point b**,**Point cir**,**double r**)**

**{**

double A**,**B**,**C**,**x**,**y**,**z**;**

A**=**veclen**(**b**,**cir**);**

B**=**veclen**(**a**,**cir**);**

C**=**veclen**(**b**,**a**);**

**if(**A**<**r**&&**B**<**r**)**

**return** xmult**(**a**,**b**,**cir**)/**2**;**

**else** **if(**A**<**r**&&**B**>=**r**)**

**{**

x**=(**dot**(**a**,**cir**,**b**)+**sqrt**(**r**\***r**\***C**\***C**-**squre**(**xmult**(**a**,**cir**,**b**))))/**C**;**

z**=**xmult**(**a**,**b**,**cir**)/**2**;**

**return** asin**(**z**\*(**1**-**x**/**C**)\***2**/**r**/**B**)\***r**\***r**/**2**+**z**\***x**/**C**;**

**}**

**else** **if(**A**>=**r**&&**B**<**r**)**

**{**

y**=(**dot**(**b**,**cir**,**a**)+**sqrt**(**r**\***r**\***C**\***C**-**suqre**(**xmult**(**b**,**cir**,**a**))))/**C**;**

z**=**xmult**(**a**,**b**,**cir**)/**2**;**

**return** asin**(**z**\*(**1**-**y**/**C**)\***2**/**r**/**A**)\***r**\***r**/**2**+**z**\***y**/**C**;**

**}**

**else** **if(**fabs**(**xmult**(**a**,**b**,**cir**))>=**r**\***C**||**dot**(**b**,**cir**,**a**)<=**0**||**dot**(**a**,**cir**,**b**)<=**0**)**

**{**

**if(**dot**(**a**,**b**,**cir**)<**0**)**

**{**

**if(**xmult**(**a**,**b**,**cir**)<**0**)**

**return** **(-**PI**-**asin**(**xmult**(**a**,**b**,**cir**)/**A**/**B**))\***r**\***r**/**2**;**

**else** **return** **(**PI**-**asin**(**xmult**(**a**,**b**,**cir**)/**A**/**B**))\***r**\***r**/**2**;**

**}**

**else** **return** asin**(**xmult**(**a**,**b**,**cir**)/**A**/**B**)\***r**\***r**/**2**;**

**}**

**else**

**{**

x**=(**dot**(**a**,**cir**,**b**)+**sqrt**(**r**\***r**\***C**\***C**-**squre**(**xmult**(**a**,**cir**,**b**))))/**C**;**

y**=(**dot**(**b**,**cir**,**a**)+**sqrt**(**r**\***r**\***C**\***C**-**squre**(**xmult**(**b**,**cir**,**a**))))/**C**;**

z**=**xmult**(**a**,**b**,**cir**)/**2**;**

**return** **(**asin**(**z**\*(**1**-**x**/**C**)\***2**/**r**/**B**\*(**1**-**EPS**))+**asin**(**z**\*(**1**-**y**/**C**)\***2**/**r**/**A**))\***r**\***r**/**2**+**z**\*((**y**+**x**)/**C**-**1**);**

**}**

**}**

//计算两个圆的面积交，两圆外离或相切时返回0

//验题 BNU 4067

double circleCrosscircleArea**(**Point c1**,**double r1**,**Point c2**,**double r2**)**

**{**

double len**=**veclen**(**c1**,**c2**);**

**if(**sign**(**len**-(**r1**+**r2**))>=**0**)** **return** 0**;**

**else** **if(**sign**(**len**-**fabs**(**r1**-**r2**))<=**0**)**

**{**

**return** min**(**r1**,**r2**)\***min**(**r1**,**r2**)\***PI**;**

**}**

**else**

**{**

double theta1,theta2;

theta1**=**2.0**\***acos**((**r1**\***r1**+**len**\***len**-**r2**\***r2**)/(**2.0**\***r1**\***len**));**

theta2**=**2.0**\***acos**((**r2**\***r2**+**len**\***len**-**r1**\***r1**)/(**2.0**\***r2**\***len**));**

double ret1**=**0.5**\***r1**\***r1**\*(**theta1**-**sin**(**theta1**));**

double ret2**=**0.5**\***r2**\***r2**\*(**theta2**-**sin**(**theta2**));**

**return** **(**ret1**+**ret2**);**

**}**

**}**

## 球面

#include <math.h>

const double pi**=**acos**(-**1**);**

//计算圆心角lat表示纬度,-90<=w<=90,lng表示经度

//返回两点所在大圆劣弧对应圆心角,0<=angle<=pi

double angle**(**double lng1**,**double lat1**,**double lng2**,**double lat2**)**

**{**

double dlng**=**fabs**(**lng1**-**lng2**)\***pi**/**180**;**

**while** **(**dlng**>=**pi**+**pi**)**

dlng**-=**pi**+**pi**;**

**if** **(**dlng**>**pi**)**

dlng**=**pi**+**pi**-**dlng**;**

lat1**\*=**pi**/**180**,**lat2**\*=**pi**/**180**;**

**return** acos**(**cos**(**lat1**)\***cos**(**lat2**)\***cos**(**dlng**)+**sin**(**lat1**)\***sin**(**lat2**));**

**}**

//计算距离,r为球半径

double line\_dist**(**double r**,**double lng1**,**double lat1**,**double lng2**,**double lat2**)**

**{**

double dlng**=**fabs**(**lng1**-**lng2**)\***pi**/**180**;**

**while** **(**dlng**>=**pi**+**pi**)**

dlng**-=**pi**+**pi**;**

**if** **(**dlng**>**pi**)**

dlng**=**pi**+**pi**-**dlng**;**

lat1**\*=**pi**/**180**,**lat2**\*=**pi**/**180**;**

**return** r**\***sqrt**(**2**-**2**\*(**cos**(**lat1**)\***cos**(**lat2**)\***cos**(**dlng**)+**sin**(**lat1**)\***sin**(**lat2**)));**

**}**

//计算球面距离,r为球半径

inline double sphere\_dist**(**double r**,**double lng1**,**double lat1**,**double lng2**,**double lat2**)**

**{**

**return** r**\***angle**(**lng1**,**lat1**,**lng2**,**lat2**);**

**}**

## 三角形

#include <math.h>

struct point

**{**

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

**};**

struct line

**{**

point a**,**b**;**

**};**

double veclen**(**point p1**,**point p2**)**

**{**

**return** sqrt**((**p1**.**x**-**p2**.**x**)\*(**p1**.**x**-**p2**.**x**)+(**p1**.**y**-**p2**.**y**)\*(**p1**.**y**-**p2**.**y**));**

**}**

point intersection**(**line u**,**line v**)**

**{**

point ret**=**u**.**a**;**

double t**=((**u**.**a**.**x**-**v**.**a**.**x**)\*(**v**.**a**.**y**-**v**.**b**.**y**)-(**u**.**a**.**y**-**v**.**a**.**y**)\*(**v**.**a**.**x**-**v**.**b**.**x**))**

**/((**u**.**a**.**x**-**u**.**b**.**x**)\*(**v**.**a**.**y**-**v**.**b**.**y**)-(**u**.**a**.**y**-**u**.**b**.**y**)\*(**v**.**a**.**x**-**v**.**b**.**x**));**

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

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

**return** ret**;**

**}**

//外心

point circumcenter**(**point a**,**point b**,**point c**)**

**{**

line u**,**v**;**

u**.**a**.**x**=(**a**.**x**+**b**.**x**)/**2**;**

u**.**a**.**y**=(**a**.**y**+**b**.**y**)/**2**;**

u**.**b**.**x**=**u**.**a**.**x**-**a**.**y**+**b**.**y**;**

u**.**b**.**y**=**u**.**a**.**y**+**a**.**x**-**b**.**x**;**

v**.**a**.**x**=(**a**.**x**+**c**.**x**)/**2**;**

v**.**a**.**y**=(**a**.**y**+**c**.**y**)/**2**;**

v**.**b**.**x**=**v**.**a**.**x**-**a**.**y**+**c**.**y**;**

v**.**b**.**y**=**v**.**a**.**y**+**a**.**x**-**c**.**x**;**

**return** intersection**(**u**,**v**);**

**}**

//内心

point incenter**(**point a**,**point b**,**point c**)**

**{**

line u**,**v**;**

double m**,**n**;**

u**.**a**=**a**;**

m**=**atan2**(**b**.**y**-**a**.**y**,**b**.**x**-**a**.**x**);**

n**=**atan2**(**c**.**y**-**a**.**y**,**c**.**x**-**a**.**x**);**

u**.**b**.**x**=**u**.**a**.**x**+**cos**((**m**+**n**)/**2**);**

u**.**b**.**y**=**u**.**a**.**y**+**sin**((**m**+**n**)/**2**);**

v**.**a**=**b**;**

m**=**atan2**(**a**.**y**-**b**.**y**,**a**.**x**-**b**.**x**);**

n**=**atan2**(**c**.**y**-**b**.**y**,**c**.**x**-**b**.**x**);**

v**.**b**.**x**=**v**.**a**.**x**+**cos**((**m**+**n**)/**2**);**

v**.**b**.**y**=**v**.**a**.**y**+**sin**((**m**+**n**)/**2**);**

**return** intersection**(**u**,**v**);**

**}**

//垂心

point perpencenter**(**point a**,**point b**,**point c**)**

**{**

line u**,**v**;**

u**.**a**=**c**;**

u**.**b**.**x**=**u**.**a**.**x**-**a**.**y**+**b**.**y**;**

u**.**b**.**y**=**u**.**a**.**y**+**a**.**x**-**b**.**x**;**

v**.**a**=**b**;**

v**.**b**.**x**=**v**.**a**.**x**-**a**.**y**+**c**.**y**;**

v**.**b**.**y**=**v**.**a**.**y**+**a**.**x**-**c**.**x**;**

**return** intersection**(**u**,**v**);**

**}**

//重心,到三角形三顶点距离的平方和最小的点

//三角形内到三边距离之积最大的点

point barycenter**(**point a**,**point b**,**point c**)**

**{**

line u**,**v**;**

u**.**a**.**x**=(**a**.**x**+**b**.**x**)/**2**;**

u**.**a**.**y**=(**a**.**y**+**b**.**y**)/**2**;**

u**.**b**=**c**;**

v**.**a**.**x**=(**a**.**x**+**c**.**x**)/**2**;**

v**.**a**.**y**=(**a**.**y**+**c**.**y**)/**2**;**

v**.**b**=**b**;**

**return** intersection**(**u**,**v**);**

**}**

//费马点，到三角形三顶点距离之和最小的点

//若三角形的三个内角均小于120度，那么该点连接三个顶点形成的三个角均为120度

//若三角形存在一个内角大于120度，则该顶点就是费马点

point fermentpoint**(**point a**,**point b**,**point c**)**

**{**

point u**,**v**;**

double step**=**fabs**(**a**.**x**)+**fabs**(**a**.**y**)+**fabs**(**b**.**x**)+**fabs**(**b**.**y**)+**fabs**(**c**.**x**)+**fabs**(**c**.**y**);**

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

u**.**x**=(**a**.**x**+**b**.**x**+**c**.**x**)/**3**;**

u**.**y**=(**a**.**y**+**b**.**y**+**c**.**y**)/**3**;**

**while** **(**step**>**1e-10**)**

**for** **(**k**=**0**;** k**<**10**;** step**/=**2**,**k**++)**

**for** **(**i**=-**1**;** i**<=**1**;** i**++)**

**for** **(**j**=-**1**;** j**<=**1**;** j**++)**

**{**

v**.**x**=**u**.**x**+**step**\***i**;**

v**.**y**=**u**.**y**+**step**\***j**;**

**if** **(**veclen**(**u**,**a**)+**veclen**(**u**,**b**)+**veclen**(**u**,**c**)**

**>**veclen**(**v**,**a**)+**veclen**(**v**,**b**)+**veclen**(**v**,**c**))**

u**=**v**;**

**}**

**return** u**;**

**}**

//求n个点的费马点，验题POJ 2420

// Fermat\_Point是求费马点的主函数，ptres返回费马点的位置，函数返回最短距离之和

#include <iostream>

#include <stdio.h>

#include <math.h>

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

#define MAXN 128

struct Point

**{**

double x**;**

double y**;**

**};**

double veclen**(**const Point **&**a**,** const Point **&**b**)**

**{**

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

**}**

double Fermat\_Point**(**Point pt**[],** int n**,** Point **&**ptres**)**

**{**

Point u**,** v**;**

double step **=** 0.0**,** curlen**,** explen**,** minlen**;**

int i**,** j**,** k**,** idx**;**

bool flag **=** **false;**

u**.**x **=** u**.**y **=** v**.**x **=** v**.**y **=** 0.0**;**

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

**{**

step **+=** fabs**(**pt**[**i**].**x**)** **+** fabs**(**pt**[**i**].**y**);**

u**.**x **+=** pt**[**i**].**x**;**

u**.**y **+=** pt**[**i**].**y**;**

**}**

u**.**x **/=** n**;**

u**.**y **/=** n**;**

**while(**step **>** 1e-5**)**

**{**

**for(**k **=** 0**;** k **<** 10**;** step **/=** 2**,** **++**k**)**

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

**for(**j **=** **-**1**;** j **<=** 1**;** **++**j**)**

**{**

v**.**x **=** u**.**x **+** step**\***i**;**

v**.**y **=** u**.**y **+** step**\***j**;**

curlen **=** explen **=** 0.0**;**

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

**{**

curlen **+=** veclen**(**u**,** pt**[**idx**]);**

explen **+=** veclen**(**v**,** pt**[**idx**]);**

**}**

**if(**curlen **>** explen**)**

**{**

u **=** v**;**

minlen **=** explen**;**

flag **=** 1**;**

**}**

**}**

**}**

ptres **=** u**;**

**return** flag**?** minlen**:** curlen**;**

**}**

int main**()**

**{**

int n**,** i**;**

double res**;**

Point pt**[**MAXN**],** ptres**;**

**while(**scanf**(**"%d"**,** **&**n**)!=**EOF**)**

**{**

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

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

res **=** Fermat\_Point**(**pt**,** n**,** ptres**);**

**if(**res **-** **(**int**)(**res**)** **>** 0.5**)**

printf**(**"%d\n"**,** **(**int**)(**res**+**1**));**

**else**

printf**(**"%d\n"**,** **(**int**)**res**);**

**}**

**return** 0**;**

**}**

//判断点p是否在三角形内与边上（只判断三角形内，把三个等号去掉即可；判断点p与多边形的关系亦可类似判断）

int onside\_inside\_triangle**(**point aa**,**point bb**,**point cc**,**point p**)**

**{**

int a1**=**area\_triangle**(**aa**,**bb**,**p**);**

int a2**=**area\_triangle**(**aa**,**cc**,**p**);**

int a3**=**area\_triangle**(**bb**,**cc**,**p**);**

int a4**=**area\_triangle**(**aa**,**bb**,**cc**);**

**if(**a1**>=**0**&&**a2**>=**0**&&**a3**>=**0**&&**a1**+**a2**+**a3**<**a4**+**eps**)** **return** 1**;**

**else** **return** 0**;**

**}**

## 三维几何

#include <math.h>

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

struct Point3

**{**

double x**,**y**,**z**;**

**};**

struct Line3

**{**

Point3 a**,**b**;**

**};**

struct Plane3

**{**

Point3 a**,**b**,**c**;**

**};**

// 计 算 cross product U x V

Point3 xmult**(**Point3 u**,**Point3 v**)**

**{**

Point3 ret**;**

ret**.**x**=**u**.**y**\***v**.**z**-**v**.**y**\***u**.**z**;**

ret**.**y**=**u**.**z**\***v**.**x**-**u**.**x**\***v**.**z**;**

ret**.**z**=**u**.**x**\***v**.**y**-**u**.**y**\***v**.**x**;**

**return** ret**;**

**}**

//计算dot product U . V

double dmult**(**Point3 u**,**Point3 v**)**

**{**

**return** u**.**x**\***v**.**x**+**u**.**y**\***v**.**y**+**u**.**z**\***v**.**z**;**

**}**

//矢量差 U - V

Point3 subt**(**Point3 u**,**Point3 v**)**

**{**

Point3 ret**;**

ret**.**x**=**u**.**x**-**v**.**x**;**

ret**.**y**=**u**.**y**-**v**.**y**;**

ret**.**z**=**u**.**z**-**v**.**z**;**

**return** ret**;**

**}**

//取平面法向量

Point3 pvec**(**Plane3 s**)**

**{**

**return** xmult**(**subt**(**s**.**a**,**s**.**b**),**subt**(**s**.**b**,**s**.**c**));**

**}**

Point3 pvec**(**Point3 s1**,**Point3 s2**,**Point3 s3**)**

**{**

**return** xmult**(**subt**(**s1**,**s2**),**subt**(**s2**,**s3**));**

**}**

//两点距离,单参数取向量大小

double veclen**(**Point3 p1**,**Point3 p2**)**

**{**

**return** sqrt**((**p1**.**x**-**p2**.**x**)\*(**p1**.**x**-**p2**.**x**)+(**p1**.**y**-**p2**.**y**)\*(**p1**.**y**-**p2**.**y**)+(**p1**.**z**-**p2**.**z**)\*(**p1**.**z**-**p2**.**z**));**

**}**

//向量大小

double vlen**(**Point3 p**)**

**{**

**return** sqrt**(**p**.**x**\***p**.**x**+**p**.**y**\***p**.**y**+**p**.**z**\***p**.**z**);**

**}**

//判三点共线

int dots\_inline**(**Point3 p1**,**Point3 p2**,**Point3 p3**)**

**{**

**return** vlen**(**xmult**(**subt**(**p1**,**p2**),**subt**(**p2**,**p3**)))<**eps**;**

**}**

//判四点共面

int dots\_onplane**(**Point3 a**,**Point3 b**,**Point3 c**,**Point3 d**)**

**{**

**return** zero**(**dmult**(**pvec**(**a**,**b**,**c**),**subt**(**d**,**a**)));**

**}**

//判点是否在线段上,包括端点和共线

int dot\_online\_in**(**Point3 p**,**Line3 l**)**

**{**

**return** zero**(**vlen**(**xmult**(**subt**(**p**,**l**.**a**),**subt**(**p**,**l**.**b**))))&&(**l**.**a**.**x**-**p**.**x**)\*(**l**.**b**.**x**-**p**.**x**)**

**<**eps**&&(**l**.**a**.**y**-**p**.**y**)\*(**l**.**b**.**y**-**p**.**y**)<**eps**&&(**l**.**a**.**z**-**p**.**z**)\*(**l**.**b**.**z**-**p**.**z**)<**eps**;**

**}**

int dot\_online\_in**(**Point3 p**,**Point3 l1**,**Point3 l2**)**

**{**

**return** zero**(**vlen**(**xmult**(**subt**(**p**,**l1**),**subt**(**p**,**l2**))))&&(**l1**.**x**-**p**.**x**)\*(**l2**.**x**-**p**.**x**)<**eps

**&&(**l1**.**y**-**p**.**y**)\*(**l2**.**y**-**p**.**y**)<**eps**&&(**l1**.**z**-**p**.**z**)\*(**l2**.**z**-**p**.**z**)<**eps**;**

**}**

//判点是否在线段上,不包括端点

int dot\_online\_ex**(**Point3 p**,**Line3 l**)**

**{**

**return** dot\_online\_in**(**p**,**l**)&&(!**zero**(**p**.**x**-**l**.**a**.**x**)||!**zero**(**p**.**y**-**l**.**a**.**y**)||!**zero**(**p**.**z**-**l**.**a**.**z**))&&(!**zero**(**p**.**x**-**l**.**b**.**x**)||!**zero**(**p**.**y**-**l**.**b**.**y**)||!**zero**(**p**.**z**-**l**.**b**.**z**));**

**}**

int dot\_online\_ex**(**Point3 p**,**Point3 l1**,**Point3 l2**)**

**{**

**return** dot\_online\_in**(**p**,**l1**,**l2**)&&(!**zero**(**p**.**x**-**l1**.**x**)||!**zero**(**p**.**y**-**l1**.**y**)||!**zero**(**p**.**z**-**l1**.**z**))&&(!**zero**(**p**.**x**-**l2**.**x**)||!**zero**(**p**.**y**-**l2**.**y**)||!**zero**(**p**.**z**-**l2**.**z**));**

**}**

//判点是否在空间三角形上,包括边界,三点共线无意义

int dot\_inplane\_in**(**Point3 p**,**Plane3 s**)**

**{**

**return** zero**(**vlen**(**xmult**(**subt**(**s**.**a**,**s**.**b**),**subt**(**s**.**a**,**s**.**c**)))-**vlen**(**xmult**(**subt**(**p**,**s**.**a**),**subt**(**p**,**s**.**b**)))-**vlen**(**xmult**(**subt**(**p**,**s**.**b**),**subt**(**p**,**s**.**c**)))-**vlen**(**xmult**(**subt**(**p**,**s**.**c**),**subt**(**p**,**s**.**a**))));**

**}**

int dot\_inplane\_in**(**Point3 p**,**Point3 s1**,**Point3 s2**,**Point3 s3**)**

**{**

**return** zero**(**vlen**(**xmult**(**subt**(**s1**,**s2**),**subt**(**s1**,**s3**)))-**vlen**(**xmult**(**subt**(**p**,**s1**),**subt**(**p**,**s2**)))-**vlen**(**xmult**(**subt**(**p**,**s2**),**subt**(**p**,**s3**)))-**vlen**(**xmult**(**subt**(**p**,**s3**),**subt**(**p**,**s1**))));**

**}**

//判点是否在空间三角形上,不包括边界,三点共线无意义

int dot\_inplane\_ex**(**Point3 p**,**Plane3 s**)**

**{**

**return** dot\_inplane\_in**(**p**,**s**)&&**vlen**(**xmult**(**subt**(**p**,**s**.**a**),**subt**(**p**,**s**.**b**)))>**eps**&&**

vlen**(**xmult**(**subt**(**p**,**s**.**b**),**subt**(**p**,**s**.**c**)))>**eps**&&**vlen**(**xmult**(**subt**(**p**,**s**.**c**),**subt**(**p**,**s**.**a**)))>**eps**;**

**}**

int dot\_inplane\_ex**(**Point3 p**,**Point3 s1**,**Point3 s2**,**Point3 s3**)**

**{**

**return** dot\_inplane\_in**(**p**,**s1**,**s2**,**s3**)&&**vlen**(**xmult**(**subt**(**p**,**s1**),**subt**(**p**,**s2**)))>**eps**&&**vlen**(**xmult**(**subt**(**p**,**s2**),**subt**(**p**,**s3**)))>**eps**&&**vlen**(**xmult**(**subt**(**p**,**s3**),**subt**(**p**,**s1**)))>**eps**;**

**}**

//判两点在线段同侧,点在线段上返回0,不共面无意义

int same\_side**(**Point3 p1**,**Point3 p2**,**Line3 l**)**

**{**

**return** dmult**(**xmult**(**subt**(**l**.**a**,**l**.**b**),**subt**(**p1**,**l**.**b**)),**xmult**(**subt**(**l**.**a**,**l**.**b**),**subt**(**p2**,**l**.**b**)))>**eps**;**

**}**

int same\_side**(**Point3 p1**,**Point3 p2**,**Point3 l1**,**Point3 l2**)**

**{**

**return** dmult**(**xmult**(**subt**(**l1**,**l2**),**subt**(**p1**,**l2**)),**xmult**(**subt**(**l1**,**l2**),**subt**(**p2**,**l2**)))>**eps**;**

**}**

//判两点在线段异侧,点在线段上返回0,不共面无意义

int opposite\_side**(**Point3 p1**,**Point3 p2**,**Line3 l**)**

**{**

**return** dmult**(**xmult**(**subt**(**l**.**a**,**l**.**b**),**subt**(**p1**,**l**.**b**)),**xmult**(**subt**(**l**.**a**,**l**.**b**),**subt**(**p2**,**l**.**b**)))<-**eps**;**

**}**

int opposite\_side**(**Point3 p1**,**Point3 p2**,**Point3 l1**,**Point3 l2**)**

**{**

**return** dmult**(**xmult**(**subt**(**l1**,**l2**),**subt**(**p1**,**l2**)),**xmult**(**subt**(**l1**,**l2**),**subt**(**p2**,**l2**)))<-**eps**;**

**}**

//判两点在平面同侧,点在平面上返回0

int same\_side**(**Point3 p1**,**Point3 p2**,**Plane3 s**)**

**{**

**return** dmult**(**pvec**(**s**),**subt**(**p1**,**s**.**a**))\***dmult**(**pvec**(**s**),**subt**(**p2**,**s**.**a**))>**eps**;**

**}**

int same\_side**(**Point3 p1**,**Point3 p2**,**Point3 s1**,**Point3 s2**,**Point3 s3**)**

**{**

**return** dmult**(**pvec**(**s1**,**s2**,**s3**),**subt**(**p1**,**s1**))\***dmult**(**pvec**(**s1**,**s2**,**s3**),**subt**(**p2**,**s1**))>**eps**;**

**}**

//判两点在平面异侧,点在平面上返回0

int opposite\_side**(**Point3 p1**,**Point3 p2**,**Plane3 s**)**

**{**

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

**}**

int opposite\_side**(**Point3 p1**,**Point3 p2**,**Point3 s1**,**Point3 s2**,**Point3 s3**)**

**{**

**return** dmult**(**pvec**(**s1**,**s2**,**s3**),**subt**(**p1**,**s1**))\***dmult**(**pvec**(**s1**,**s2**,**s3**),**subt**(**p2**,**s1**))<-**eps**;**

**}**

//判两直线平行

int parallel**(**Line3 u**,**Line3 v**)**

**{**

**return** vlen**(**xmult**(**subt**(**u**.**a**,**u**.**b**),**subt**(**v**.**a**,**v**.**b**)))<**eps**;**

**}**

int parallel**(**Point3 u1**,**Point3 u2**,**Point3 v1**,**Point3 v2**)**

**{**

**return** vlen**(**xmult**(**subt**(**u1**,**u2**),**subt**(**v1**,**v2**)))<**eps**;**

**}**

//判两平面平行

int parallel**(**Plane3 u**,**Plane3 v**)**

**{**

**return** vlen**(**xmult**(**pvec**(**u**),**pvec**(**v**)))<**eps**;**

**}**

int parallel**(**Point3 u1**,**Point3 u2**,**Point3 u3**,**Point3 v1**,**Point3 v2**,**Point3 v3**)**

**{**

**return** vlen**(**xmult**(**pvec**(**u1**,**u2**,**u3**),**pvec**(**v1**,**v2**,**v3**)))<**eps**;**

**}**

//判直线与平面平行

int parallel**(**Line3 l**,**Plane3 s**)**

**{**

**return** zero**(**dmult**(**subt**(**l**.**a**,**l**.**b**),**pvec**(**s**)));**

**}**

int parallel**(**Point3 l1**,**Point3 l2**,**Point3 s1**,**Point3 s2**,**Point3 s3**)**

**{**

**return** zero**(**dmult**(**subt**(**l1**,**l2**),**pvec**(**s1**,**s2**,**s3**)));**

**}**

//判两直线垂直

int perpendicular**(**Line3 u**,**Line3 v**)**

**{**

**return** zero**(**dmult**(**subt**(**u**.**a**,**u**.**b**),**subt**(**v**.**a**,**v**.**b**)));**

**}**

int perpendicular**(**Point3 u1**,**Point3 u2**,**Point3 v1**,**Point3 v2**)**

**{**

**return** zero**(**dmult**(**subt**(**u1**,**u2**),**subt**(**v1**,**v2**)));**

**}**

//判两平面垂直

int perpendicular**(**Plane3 u**,**Plane3 v**)**

**{**

**return** zero**(**dmult**(**pvec**(**u**),**pvec**(**v**)));**

**}**

int perpendicular**(**Point3 u1**,**Point3 u2**,**Point3 u3**,**Point3 v1**,**Point3 v2**,**Point3 v3**)**

**{**

**return** zero**(**dmult**(**pvec**(**u1**,**u2**,**u3**),**pvec**(**v1**,**v2**,**v3**)));**

**}**

//判直线与平面平行

int perpendicular**(**Line3 l**,**Plane3 s**)**

**{**

**return** vlen**(**xmult**(**subt**(**l**.**a**,**l**.**b**),**pvec**(**s**)))<**eps**;**

**}**

int perpendicular**(**Point3 l1**,**Point3 l2**,**Point3 s1**,**Point3 s2**,**Point3 s3**)**

**{**

**return** vlen**(**xmult**(**subt**(**l1**,**l2**),**pvec**(**s1**,**s2**,**s3**)))<**eps**;**

**}**

//判两线段相交,包括端点和部分重合

int intersect\_in**(**Line3 u**,**Line3 v**)**

**{**

**if** **(!**dots\_onplane**(**u**.**a**,**u**.**b**,**v**.**a**,**v**.**b**))**

**return** 0**;**

**if** **(!**dots\_inline**(**u**.**a**,**u**.**b**,**v**.**a**)||!**dots\_inline**(**u**.**a**,**u**.**b**,**v**.**b**))**

**return** **!**same\_side**(**u**.**a**,**u**.**b**,**v**)&&!**same\_side**(**v**.**a**,**v**.**b**,**u**);**

**return** dot\_online\_in**(**u**.**a**,**v**)||**dot\_online\_in**(**u**.**b**,**v**)||**dot\_online\_in**(**v**.**a**,**u**)||**dot\_online\_in**(**v**.**b**,**u**);**

**}**

int intersect\_in**(**Point3 u1**,**Point3 u2**,**Point3 v1**,**Point3 v2**)**

**{**

**if** **(!**dots\_onplane**(**u1**,**u2**,**v1**,**v2**))**

**return** 0**;**

**if** **(!**dots\_inline**(**u1**,**u2**,**v1**)||!**dots\_inline**(**u1**,**u2**,**v2**))**

**return** **!**same\_side**(**u1**,**u2**,**v1**,**v2**)&&!**same\_side**(**v1**,**v2**,**u1**,**u2**);**

**return** dot\_online\_in**(**u1**,**v1**,**v2**)||**dot\_online\_in**(**u2**,**v1**,**v2**)||**dot\_online\_in**(**v1**,**u1**,**u2**)||**dot\_online\_in**(**v2**,**u1**,**u2**);**

**}**

//判两线段相交,不包括端点和部分重合

int intersect\_ex**(**Line3 u**,**Line3 v**)**

**{**

**return** dots\_onplane**(**u**.**a**,**u**.**b**,**v**.**a**,**v**.**b**)&&**opposite\_side**(**u**.**a**,**u**.**b**,**v**)&&**opposite\_side**(**v**.**a**,**v**.**b**,**u**);**

**}**

int intersect\_ex**(**Point3 u1**,**Point3 u2**,**Point3 v1**,**Point3 v2**)**

**{**

**return** dots\_onplane**(**u1**,**u2**,**v1**,**v2**)&&**opposite\_side**(**u1**,**u2**,**v1**,**v2**)**

**&&**opposite\_side**(**v1**,**v2**,**u1**,**u2**);**

**}**

//判线段与空间三角形相交,包括交于边界和(部分)包含

int intersect\_in**(**Line3 l**,**Plane3 s**)**

**{**

**return** **!**same\_side**(**l**.**a**,**l**.**b**,**s**)&&!**same\_side**(**s**.**a**,**s**.**b**,**l**.**a**,**l**.**b**,**s**.**c**)&&** **!**same\_side**(**s**.**b**,**s**.**c**,**l**.**a**,**l**.**b**,**s**.**a**)&&!**same\_side**(**s**.**c**,**s**.**a**,**l**.**a**,**l**.**b**,**s**.**b**);**

**}**

int intersect\_in**(**Point3 l1**,**Point3 l2**,**Point3 s1**,**Point3 s2**,**Point3 s3**)**

**{**

**return** **!**same\_side**(**l1**,**l2**,**s1**,**s2**,**s3**)&&!**same\_side**(**s1**,**s2**,**l1**,**l2**,**s3**)&&!**same\_side**(**s2**,**s3**,**l1**,**l2**,**s1**)&&!**same\_side**(**s3**,**s1**,**l1**,**l2**,**s2**);**

**}**

//判线段与空间三角形相交,不包括交于边界和(部分)包含

int intersect\_ex**(**Line3 l**,**Plane3 s**)**

**{**

**return** opposite\_side**(**l**.**a**,**l**.**b**,**s**)&&**opposite\_side**(**s**.**a**,**s**.**b**,**l**.**a**,**l**.**b**,**s**.**c**)&&**

opposite\_side**(**s**.**b**,**s**.**c**,**l**.**a**,**l**.**b**,**s**.**a**)&&**opposite\_side**(**s**.**c**,**s**.**a**,**l**.**a**,**l**.**b**,**s**.**b**);**

**}**

int intersect\_ex**(**Point3 l1**,**Point3 l2**,**Point3 s1**,**Point3 s2**,**Point3 s3**)**

**{**

**return** opposite\_side**(**l1**,**l2**,**s1**,**s2**,**s3**)&&**opposite\_side**(**s1**,**s2**,**l1**,**l2**,**s3**)&&** opposite\_side**(**s2**,**s3**,**l1**,**l2**,**s1**)&&**opposite\_side**(**s3**,**s1**,**l1**,**l2**,**s2**);**

**}**

//计算两直线交点,注意事先判断直线是否共面和平行!

//线段交点请另外判线段相交(同时还是要判断是否平行!)

Point3 intersection**(**Line3 u**,**Line3 v**)**

**{**

Point3 ret**=**u**.**a**;**

double t**=((**u**.**a**.**x**-**v**.**a**.**x**)\*(**v**.**a**.**y**-**v**.**b**.**y**)-(**u**.**a**.**y**-**v**.**a**.**y**)\*(**v**.**a**.**x**-**v**.**b**.**x**))**

**/((**u**.**a**.**x**-**u**.**b**.**x**)\*(**v**.**a**.**y**-**v**.**b**.**y**)-(**u**.**a**.**y**-**u**.**b**.**y**)\*(**v**.**a**.**x**-**v**.**b**.**x**));**

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

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

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

**return** ret**;**

**}**

Point3 intersection**(**Point3 u1**,**Point3 u2**,**Point3 v1**,**Point3 v2**)**

**{**

Point3 ret**=**u1**;**

double t**=((**u1**.**x**-**v1**.**x**)\*(**v1**.**y**-**v2**.**y**)-(**u1**.**y**-**v1**.**y**)\*(**v1**.**x**-**v2**.**x**))**

**/((**u1**.**x**-**u2**.**x**)\*(**v1**.**y**-**v2**.**y**)-(**u1**.**y**-**u2**.**y**)\*(**v1**.**x**-**v2**.**x**));**

ret**.**x**+=(**u2**.**x**-**u1**.**x**)\***t**;**

ret**.**y**+=(**u2**.**y**-**u1**.**y**)\***t**;**

ret**.**z**+=(**u2**.**z**-**u1**.**z**)\***t**;**

**return** ret**;**

**}**

//计算直线与平面交点,注意事先判断是否平行,并保证三点不共线!

//线段和空间三角形交点请另外判断

Point3 intersection**(**Line3 l**,**Plane3 s**)**

**{**

Point3 ret**=**pvec**(**s**);**

double t**=(**ret**.**x**\*(**s**.**a**.**x**-**l**.**a**.**x**)+**ret**.**y**\*(**s**.**a**.**y**-**l**.**a**.**y**)+**ret**.**z**\*(**s**.**a**.**z**-**l**.**a**.**z**))/**

**(**ret**.**x**\*(**l**.**b**.**x**-**l**.**a**.**x**)+**ret**.**y**\*(**l**.**b**.**y**-**l**.**a**.**y**)+**ret**.**z**\*(**l**.**b**.**z**-**l**.**a**.**z**));**

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

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

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

**return** ret**;**

**}**

Point3 intersection**(**Point3 l1**,**Point3 l2**,**Point3 s1**,**Point3 s2**,**Point3 s3**)**

**{**

Point3 ret**=**pvec**(**s1**,**s2**,**s3**);**

double t**=(**ret**.**x**\*(**s1**.**x**-**l1**.**x**)+**ret**.**y**\*(**s1**.**y**-**l1**.**y**)+**ret**.**z**\*(**s1**.**z**-**l1**.**z**))/**

**(**ret**.**x**\*(**l2**.**x**-**l1**.**x**)+**ret**.**y**\*(**l2**.**y**-**l1**.**y**)+**ret**.**z**\*(**l2**.**z**-**l1**.**z**));**

ret**.**x**=**l1**.**x**+(**l2**.**x**-**l1**.**x**)\***t**;**

ret**.**y**=**l1**.**y**+(**l2**.**y**-**l1**.**y**)\***t**;**

ret**.**z**=**l1**.**z**+(**l2**.**z**-**l1**.**z**)\***t**;**

**return** ret**;**

**}**

//计算两平面交线,注意事先判断是否平行,并保证三点不共线!

Line3 intersection**(**Plane3 u**,**Plane3 v**)**

**{**

Line3 ret**;**

ret**.**a**=**parallel**(**v**.**a**,**v**.**b**,**u**.**a**,**u**.**b**,**u**.**c**)?**intersection**(**v**.**b**,**v**.**c**,**u**.**a**,**u**.**b**,**u**.**c**):**intersection**(**v**.**a**,**v**.**b**,**u**.**a**,**u**.**b**,**u**.**c**);**

ret**.**b**=**parallel**(**v**.**c**,**v**.**a**,**u**.**a**,**u**.**b**,**u**.**c**)?**intersection**(**v**.**b**,**v**.**c**,**u**.**a**,**u**.**b**,**u**.**c**):**intersection**(**v**.**c**,**v**.**a**,**u**.**a**,**u**.**b**,**u**.**c**);**

**return** ret**;**

**}**

Line3 intersection**(**Point3 u1**,**Point3 u2**,**Point3 u3**,**Point3 v1**,**Point3 v2**,**Point3 v3**)**

**{**

Line3 ret**;**

ret**.**a**=**parallel**(**v1**,**v2**,**u1**,**u2**,**u3**)?**intersection**(**v2**,**v3**,**u1**,**u2**,**u3**):**intersection**(**v1**,**v2**,**u1**,**u2**,**u3**);**

ret**.**b**=**parallel**(**v3**,**v1**,**u1**,**u2**,**u3**)?**intersection**(**v2**,**v3**,**u1**,**u2**,**u3**):**intersection**(**v3**,**v1**,**u1**,**u2**,**u3**);**

**return** ret**;**

**}**

//点到直线距离

double ptoline**(**Point3 p**,**Line3 l**)**

**{**

**return** vlen**(**xmult**(**subt**(**p**,**l**.**a**),**subt**(**l**.**b**,**l**.**a**)))/**veclen**(**l**.**a**,**l**.**b**);**

**}**

double ptoline**(**Point3 p**,**Point3 l1**,**Point3 l2**)**

**{**

**return** vlen**(**xmult**(**subt**(**p**,**l1**),**subt**(**l2**,**l1**)))/**veclen**(**l1**,**l2**);**

**}**

//点到平面距离

double ptoplane**(**Point3 p**,**Plane3 s**)**

**{**

**return** fabs**(**dmult**(**pvec**(**s**),**subt**(**p**,**s**.**a**)))/**vlen**(**pvec**(**s**));**

**}**

double ptoplane**(**Point3 p**,**Point3 s1**,**Point3 s2**,**Point3 s3**)**

**{**

**return** fabs**(**dmult**(**pvec**(**s1**,**s2**,**s3**),**subt**(**p**,**s1**)))/**vlen**(**pvec**(**s1**,**s2**,**s3**));**

**}**

//直线到直线距离

double linetoline**(**Line3 u**,**Line3 v**)**

**{**

Point3 n**=**xmult**(**subt**(**u**.**a**,**u**.**b**),**subt**(**v**.**a**,**v**.**b**));**

**return** fabs**(**dmult**(**subt**(**u**.**a**,**v**.**a**),**n**))/**vlen**(**n**);**

**}**

double linetoline**(**Point3 u1**,**Point3 u2**,**Point3 v1**,**Point3 v2**)**

**{**

Point3 n**=**xmult**(**subt**(**u1**,**u2**),**subt**(**v1**,**v2**));**

**return** fabs**(**dmult**(**subt**(**u1**,**v1**),**n**))/**vlen**(**n**);**

**}**

//两直线夹角cos值

double angle\_cos**(**Line3 u**,**Line3 v**)**

**{**

**return** dmult**(**subt**(**u**.**a**,**u**.**b**),**subt**(**v**.**a**,**v**.**b**))/**

vlen**(**subt**(**u**.**a**,**u**.**b**))/**vlen**(**subt**(**v**.**a**,**v**.**b**));**

**}**

double angle\_cos**(**Point3 u1**,**Point3 u2**,**Point3 v1**,**Point3 v2**)**

**{**

**return** dmult**(**subt**(**u1**,**u2**),**subt**(**v1**,**v2**))/**vlen**(**subt**(**u1**,**u2**))/**vlen**(**subt**(**v1**,**v2**));**

**}**

//两平面夹角cos值

double angle\_cos**(**Plane3 u**,**Plane3 v**)**

**{**

**return** dmult**(**pvec**(**u**),**pvec**(**v**))/**vlen**(**pvec**(**u**))/**vlen**(**pvec**(**v**));**

**}**

double angle\_cos**(**Point3 u1**,**Point3 u2**,**Point3 u3**,**Point3 v1**,**Point3 v2**,**Point3 v3**)**

**{**

**return** dmult**(**pvec**(**u1**,**u2**,**u3**),**pvec**(**v1**,**v2**,**v3**))/**

vlen**(**pvec**(**u1**,**u2**,**u3**))/**vlen**(**pvec**(**v1**,**v2**,**v3**));**

**}**

//直线平面夹角sin值

double angle\_sin**(**Line3 l**,**Plane3 s**)**

**{**

**return** dmult**(**subt**(**l**.**a**,**l**.**b**),**pvec**(**s**))/**vlen**(**subt**(**l**.**a**,**l**.**b**))/**vlen**(**pvec**(**s**));**

**}**

double angle\_sin**(**Point3 l1**,**Point3 l2**,**Point3 s1**,**Point3 s2**,**Point3 s3**)**

**{**

**return** dmult**(**subt**(**l1**,**l2**),**pvec**(**s1**,**s2**,**s3**))/**

vlen**(**subt**(**l1**,**l2**))/**vlen**(**pvec**(**s1**,**s2**,**s3**));**

**}**

## 体积并与体积交

**//同心球体和圆柱体的体积交**

//球的方程 x^2+y^2+z^2<=r^2

//圆柱的坐标 x^2+y^2<=hr^2, |z|<=hz

//asr\_main()函数是simpson积分主函数，F(x)是积分函数

#define PI acos(-1.0)

double F**(**double x**)**

**{**

//R为球的半径

**return** PI**\*(**R**\***R**-**x**\***x**);**

**}**

double orb\_cross\_cylinder**(**double r**,**double hr**,**double hz**)**

**{**

**if(**hr**\***hr**+**hz**\***hz**<=**r**\***r**)** **return** 2**\***PI**\***hr**\***hr**\***hz**;**

**else** **if(**hr**>=**r**&&**hz**>=**r**)** **return** 4**\***PI**\***r**\***r**\***r**/**3**;**

**else** **if(**hr**<**r**&&**hz**>=**r**)**

**{**

double ans1**=**2**\***asr\_main**(**sqrt**(**r**\***r**-**hr**\***hr**),**r**);**

double ans2**=**2**\***sqrt**(**r**\***r**-**hr**\***hr**)\***PI**\***hr**\***hr**;**

**return** ans1**+**ans2**;**

**}**

**else** **if(**hr**>=**r**&&**hz**<**r**)** **return** 4**\***PI**\***r**\***r**\***r**/**3**-**2**\***asr\_main**(**hz**,**r**);**

**else**

**{**

double ans1**=**2**\***asr\_main**(**sqrt**(**r**\***r**-**hr**\***hr**),**hz**);**

double ans2**=**2**\***sqrt**(**r**\***r**-**hr**\***hr**)\***PI**\***hr**\***hr**;**

**return** ans1**+**ans2**;**

**}**

**}**

## 凸包(graham)

// CONVEX HULL I

// modified by rr 不能去掉点集中重合的点

#include <stdlib.h>

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

struct point

**{**

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

**};**

//计算cross product (P1-P0)x(P2-P0)

double xmult**(**point p1**,**point p2**,**point p0**)**

**{**

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

**}**

//graham算法顺时针构造包含所有共线点的凸包,O(nlogn)

point p1**,**p2**;**

int graham\_cp**(**const void**\*** a**,**const void**\*** b**)**

**{**

double ret**=**xmult**(\*((**point**\*)**a**),\*((**point**\*)**b**),**p1**);**

**return** zero**(**ret**)?(**xmult**(\*((**point**\*)**a**),\*((**point**\*)**b**),**p2**)>**0**?**1**:-**1**):(**ret**>**0**?**1**:-**1**);**

**}**

void \_graham**(**int n**,**point**\*** p**,**int**&** s**,**point**\*** ch**)**

**{**

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

**for** **(**p1**=**p2**=**p**[**0**],**i**=**1**;** i**<**n**;** p2**.**x**+=**p**[**i**].**x**,**p2**.**y**+=**p**[**i**].**y**,**i**++)**

**if** **(**p1**.**y**-**p**[**i**].**y**>**eps**||(**zero**(**p1**.**y**-**p**[**i**].**y**)&&**p1**.**x**>**p**[**i**].**x**))**

p1**=**p**[**k**=**i**];**

p2**.**x**/=**n**,**p2**.**y**/=**n**;**

p**[**k**]=**p**[**0**],**p**[**0**]=**p1**;**

qsort**(**p**+**1**,**n**-**1**,sizeof(**point**),**graham\_cp**);**

**for** **(**ch**[**0**]=**p**[**0**],**ch**[**1**]=**p**[**1**],**ch**[**2**]=**p**[**2**],**s**=**i**=**3**;** i**<**n**;** ch**[**s**++]=**p**[**i**++])**

**for** **(;** s**>**2**&&**xmult**(**ch**[**s**-**2**],**p**[**i**],**ch**[**s**-**1**])<-**eps**;** s**--);**

**}**

//构造凸包接口函数,传入原始点集大小n,点集p(p原有顺序被打乱!)

//返回凸包大小,凸包的点在convex中

//参数maxsize为1包含共线点,为0不包含共线点,缺省为1

//参数clockwise为1顺时针构造,为0逆时针构造,缺省为1

//在输入仅有若干共线点时算法不稳定,可能有此类情况请另行处理!

//不能去掉点集中重合的点

int graham**(**int n**,**point**\*** p**,**point**\*** convex**,**int maxsize**=**1**,**int dir**=**1**)**

**{**

point**\*** temp**=new** point**[**n**];**

int s**,**i**;**

\_graham**(**n**,**p**,**s**,**temp**);**

**for** **(**convex**[**0**]=**temp**[**0**],**n**=**1**,**i**=(**dir**?**1**:(**s**-**1**));** dir**?(**i**<**s**):**i**;** i**+=(**dir**?**1**:-**1**))**

**if** **(**maxsize**||!**zero**(**xmult**(**temp**[**i**-**1**],**temp**[**i**],**temp**[(**i**+**1**)%**s**])))**

convex**[**n**++]=**temp**[**i**];**

**delete** **[]**temp**;**

**return** n**;**

**}**

// CONVEX HULL II

// modified by mgmg 去掉点集中重合的点

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

struct point

**{**

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

**};**

//计算cross product (P1-P0)x(P2-P0)

double xmult**(**point p1**,**point p2**,**point p0**)**

**{**

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

**}**

//graham算法顺时针构造包含所有共线点的凸包,O(nlogn)

point p1**,**p2**;**

int graham\_cp**(**const void**\*** a**,**const void**\*** b**)**

**{**

double ret**=**xmult**(\*((**point**\*)**a**),\*((**point**\*)**b**),**p1**);**

**return** zero**(**ret**)?(**xmult**(\*((**point**\*)**a**),\*((**point**\*)**b**),**p2**)>**0**?**1**:-**1**):(**ret**>**0**?**1**:-**1**);**

**}**

void \_graham**(**int n**,**point**\*** p**,**int**&** s**,**point**\*** ch**)**

**{**

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

**for** **(**p1**=**p2**=**p**[**0**],**i**=**1**;** i**<**n**;** p2**.**x**+=**p**[**i**].**x**,**p2**.**y**+=**p**[**i**].**y**,**i**++)**

**if** **(**p1**.**y**-**p**[**i**].**y**>**eps**||(**zero**(**p1**.**y**-**p**[**i**].**y**)&&**p1**.**x**>**p**[**i**].**x**))**

p1**=**p**[**k**=**i**];**

p2**.**x**/=**n**,**p2**.**y**/=**n**;**

p**[**k**]=**p**[**0**],**p**[**0**]=**p1**;**

qsort**(**p**+**1**,**n**-**1**,sizeof(**point**),**graham\_cp**);**

**for** **(**ch**[**0**]=**p**[**0**],**ch**[**1**]=**p**[**1**],**ch**[**2**]=**p**[**2**],**s**=**i**=**3**;** i**<**n**;** ch**[**s**++]=**p**[**i**++])**

**for** **(;** s**>**2**&&**xmult**(**ch**[**s**-**2**],**p**[**i**],**ch**[**s**-**1**])<-**eps**;** s**--);**

**}**

int wipesame\_cp**(**const void **\***a**,** const void **\***b**)**

**{**

**if** **((\*(**point **\*)**a**).**y **<** **(\*(**point **\*)**b**).**y **-** eps**)** **return** **-**1**;**

**else** **if** **((\*(**point **\*)**a**).**y **>** **(\*(**point **\*)**b**).**y **+** eps**)** **return** 1**;**

**else** **if** **((\*(**point **\*)**a**).**x **<** **(\*(**point **\*)**b**).**x **-** eps**)** **return** **-**1**;**

**else** **if** **((\*(**point **\*)**a**).**x **>** **(\*(**point **\*)**b**).**x **+** eps**)** **return** 1**;**

**else** **return** 0**;**

**}**

int \_wipesame**(**point **\*** p**,** int n**)**

**{**

int i**,** k**;**

qsort**(**p**,** n**,** **sizeof(**point**),** wipesame\_cp**);**

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

**if** **(**wipesame\_cp**(**p**+**i**,**p**+**i**-**1**)!=**0**)** p**[**k**++]=**p**[**i**];**

**return** k**;**

**}**

//构造凸包接口函数,传入原始点集大小n,点集p(p原有顺序被打乱!)

//返回凸包大小,凸包的点在convex中

//参数maxsize为1包含共线点,为0不包含共线点,缺省为1

//参数clockwise为1顺时针构造,为0逆时针构造,缺省为1

//在输入仅有若干共线点时算法不稳定,可能有此类情况请另行处理!

int graham**(**int n**,**point**\*** p**,**point**\*** convex**,**int maxsize**=**1**,**int dir**=**1**)**

**{**

point**\*** temp**=new** point**[**n**];**

int s**,**i**;**

n **=** \_wipesame**(**p**,**n**);**

\_graham**(**n**,**p**,**s**,**temp**);**

**for** **(**convex**[**0**]=**temp**[**0**],**n**=**1**,**i**=(**dir**?**1**:(**s**-**1**));** dir**?(**i**<**s**):**i**;** i**+=(**dir**?**1**:-**1**))**

**if** **(**maxsize**||!**zero**(**xmult**(**temp**[**i**-**1**],**temp**[**i**],**temp**[(**i**+**1**)%**s**])))**

convex**[**n**++]=**temp**[**i**];**

**delete** **[]**temp**;**

**return** n**;**

**}**

//CONVEX HULL III

//modified by asdw12345

int sign**(**double x**)** //三态函数

**{**

**if(**fabs**(**x**)<**eps**)** **return** 0**;**

**else** **return** x**>**0**?**1**:-**1**;**

**}**

bool graham\_cmp**(**point p1**,**point p2**)** //水平排序

**{**

**if(**sign**(**p1**.**y**-**p2**.**y**)<**0**)** **return** **true;**

**else** **if(**sign**(**p1**.**y**-**p2**.**y**)==**0**)** **return** sign**(**p1**.**x**-**p2**.**x**)<**0**;**

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

**}**

vector**<**point**>** polygon\_graham**(**vector**<**point**>** convex**)**

**{**

int s**=**convex**.**size**();**

vector**<**point**>** ret**;**

ret**.**resize**(**2**\***s**);**

int i**,**j**,**top**=**0**;**

**if(**s**<=**2**)**

**{**

**for(**i**=**0**;** i**<**s**;** i**++)** ret**[**i**]=**convex**[**i**];**

ret**.**resize**(**s**);**

**return** ret**;**

**}**

sort**(**convex**.**begin**(),**convex**.**end**(),**graham\_cmp**);**

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

**{**

**while(**top**>**1**&&**sign**(**xmult**(**ret**[**top**-**2**],**convex**[**i**],**ret**[**top**-**1**]))<=**0**)** top**--;**

ret**[**top**++]=**convex**[**i**];**

**}**

int len**=**top**;**

**for(**i**=**s**-**2**;** i**>=**0**;** i**--)**

**{**

**while(**top**>**len**&&**sign**(**xmult**(**ret**[**top**-**2**],**convex**[**i**],**ret**[**top**-**1**])<=**0**))** top**--;**

ret**[**top**++]=**convex**[**i**];**

**}**

**if(**s**>**1**)** top**--;**

ret**.**resize**(**top**);**

**return** ret**;**

**}**

//旋转卡壳法求凸包直径（最远对踵点对距离）

double xmult**(**point p1**,**point p2**,**point p0**)**

**{**

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

**}**

double veclen**(**point p**,**point q**)**

**{**

**return** sqrt**((**p**.**x**-**q**.**x**)\*(**p**.**x**-**q**.**x**)+(**p**.**y**-**q**.**y**)\*(**p**.**y**-**q**.**y**));**

**}**

//凸包上的点按逆时针方向给出

double rotating\_calipers**(**vector**<**point**>** convex**)**

**{**

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

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

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

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

**{**

double tp1**=**xmult**(**convex**[(**i**+**1**)%**n**],**convex**[(**j**+**1**)%**n**],**convex**[**i**]);**

double tp2**=**xmult**(**convex**[(**i**+**1**)%**n**],**convex**[**j**],**convex**[**i**]);**

//若给出顺序为顺时针，需要将下面的小于号改成大于号

**while(**tp1**<**tp2**)**

**{**

j**=(**j**+**1**)%**n**;**

tp1**=**xmult**(**convex**[(**i**+**1**)%**n**],**convex**[(**j**+**1**)%**n**],**convex**[**i**]);**

tp2**=**xmult**(**convex**[(**i**+**1**)%**n**],**convex**[**j**],**convex**[**i**]);**

**}**

double dis1**=**veclen**(**convex**[**i**],**convex**[**j**]);**

double dis2**=**veclen**(**convex**[(**i**+**1**)%**n**],**convex**[(**j**+**1**)%**n**]);**

ans**=**max**(**ans**,**max**(**dis1**,**dis2**));**

**}**

**return** ans**;**

**}**

## 三维凸包

#include <stdio.h>

#include <string.h>

#include <iostream>

#include <algorithm>

#include <vector>

#include <queue>

#include <set>

#include <map>

#include <string>

#include <math.h>

#include <stdlib.h>

#include <time.h>

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

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

const int MAXN **=** 550**;**

int sign**(**double x**)**

**{**

**if(**fabs**(**x**)** **<** eps**)return** 0**;**

**if(**x **<** 0**)return** **-**1**;**

**else** **return** 1**;**

**}**

struct Point3

**{**

double x**,**y**,**z**;**

Point3**(**double \_x **=** 0**,** double \_y **=** 0**,** double \_z **=** 0**)**

**{**

x **=** \_x**;**

y **=** \_y**;**

z **=** \_z**;**

**}**

void input**()**

**{**

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

**}**

bool **operator** **==(**const Point3 **&**b**)**const

**{**

**return** sign**(**x**-**b**.**x**)** **==** 0 **&&** sign**(**y**-**b**.**y**)** **==** 0 **&&** sign**(**z**-**b**.**z**)** **==** 0**;**

**}**

double len**()**

**{**

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

**}**

double len2**()**

**{**

**return** x**\***x**+**y**\***y**+**z**\***z**;**

**}**

double distance**(**const Point3 **&**b**)**const

**{**

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

**}**

Point3 **operator** **-(**const Point3 **&**b**)**const

**{**

**return** Point3**(**x**-**b**.**x**,**y**-**b**.**y**,**z**-**b**.**z**);**

**}**

Point3 **operator** **+(**const Point3 **&**b**)**const

**{**

**return** Point3**(**x**+**b**.**x**,**y**+**b**.**y**,**z**+**b**.**z**);**

**}**

Point3 **operator** **\*(**const double **&**k**)**const

**{**

**return** Point3**(**x**\***k**,**y**\***k**,**z**\***k**);**

**}**

Point3 **operator** **/(**const double **&**k**)**const

**{**

**return** Point3**(**x**/**k**,**y**/**k**,**z**/**k**);**

**}**

double **operator** **\*(**const Point3 **&**b**)**const //点乘

**{**

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

**}**

Point3 **operator** **^(**const Point3 **&**b**)**const //叉乘

**{**

**return** Point3**(**y**\***b**.**z**-**z**\***b**.**y**,**z**\***b**.**x**-**x**\***b**.**z**,**x**\***b**.**y**-**y**\***b**.**x**);**

**}**

**};**

struct CH3D

**{**

struct face

**{**

int a**,**b**,**c**;** //表示凸包一个面上的三个点的编号

bool ok**;** //表示该面是否属于最终的凸包上的面

**};**

int n**;** //初始顶点数

Point3 P**[**MAXN**]**；

int num**;** //凸包表面的三角形数

face F**[**8**\***MAXN**];** //凸包表面的三角形

int g**[**MAXN**][**MAXN**];**

Point3 cross**(**const Point3 **&**a**,**const Point3 **&**b**,**const Point3 **&**c**)** //叉乘

**{**

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

**}**

double area**(**Point3 a**,**Point3 b**,**Point3 c**)** //三角形面积\*2

**{**

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

**}**

//四面体有向面积\*6

double volume**(**Point3 a**,**Point3 b**,**Point3 c**,**Point3 d**)**

**{**

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

**}**

double dblcmp**(**Point3 **&**p**,**face **&**f**)** //正：点在面同向

**{**

Point3 p1 **=** P**[**f**.**b**]** **-** P**[**f**.**a**];**

Point3 p2 **=** P**[**f**.**c**]** **-** P**[**f**.**a**];**

Point3 p3 **=** p **-** P**[**f**.**a**];**

**return** **(**p1**^**p2**)\***p3**;**

**}**

void deal**(**int p**,**int a**,**int b**)**

**{**

int f **=** g**[**a**][**b**];**

face add**;**

**if(**F**[**f**].**ok**)**

**{**

**if(**dblcmp**(**P**[**p**],**F**[**f**])** **>** eps**)**

dfs**(**p**,**f**);**

**else**

**{**

add**.**a **=** b**;**

add**.**b **=** a**;**

add**.**c **=** p**;**

add**.**ok **=** **true;**

g**[**p**][**b**]** **=** g**[**a**][**p**]** **=** g**[**b**][**a**]** **=** num**;**

F**[**num**++]** **=** add**;**

**}**

**}**

**}**

//递归搜索所有应该从凸包内删除的面

void dfs**(**int p**,**int now**)**

**{**

F**[**now**].**ok **=** **false;**

deal**(**p**,**F**[**now**].**b**,**F**[**now**].**a**);**

deal**(**p**,**F**[**now**].**c**,**F**[**now**].**b**);**

deal**(**p**,**F**[**now**].**a**,**F**[**now**].**c**);**

**}**

bool same**(**int s**,**int t**)**

**{**

Point3 **&**a **=** P**[**F**[**s**].**a**];**

Point3 **&**b **=** P**[**F**[**s**].**b**];**

Point3 **&**c **=** P**[**F**[**s**].**c**];**

**return** fabs**(**volume**(**a**,**b**,**c**,**P**[**F**[**t**].**a**]))** **<** eps **&&**

fabs**(**volume**(**a**,**b**,**c**,**P**[**F**[**t**].**b**]))** **<** eps **&&**

fabs**(**volume**(**a**,**b**,**c**,**P**[**F**[**t**].**c**]))** **<** eps**;**

**}**

//构建三维凸包

void create**()**

**{**

num **=** 0**;**

face add**;**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//此段是为了保证前四个点不共面，若已经保证可以删去

bool flag **=** **true;**

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

**{**

**if(!(**P**[**0**]** **==** P**[**i**]))**

**{**

swap**(**P**[**1**],**P**[**i**]);**

flag **=** **false;**

**break;**

**}**

**}**

**if(**flag**)return;**

flag **=** **true;**

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

**{**

**if(** **((**P**[**1**]-**P**[**0**])^(**P**[**i**]-**P**[**0**])).**len**()** **>** eps **)**

**{**

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

flag **=** **false;**

**break;**

**}**

**}**

**if(**flag**)return;**

flag **=** **true;**

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

**{**

**if(**fabs**(** **((**P**[**1**]-**P**[**0**])^(**P**[**2**]-**P**[**0**]))\*(**P**[**i**]-**P**[**0**])** **)** **>** eps**)**

**{**

swap**(**P**[**3**],**P**[**i**]);**

flag **=** **false;**

**break;**

**}**

**}**

**if(**flag**)return;**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

**{**

add**.**a **=** **(**i**+**1**)%**4**;**

add**.**b **=** **(**i**+**2**)%**4**;**

add**.**c **=** **(**i**+**3**)%**4**;**

add**.**ok **=** **true;**

**if(**dblcmp**(**P**[**i**],**add**)** **>** 0**)**swap**(**add**.**b**,**add**.**c**);**

g**[**add**.**a**][**add**.**b**]** **=** g**[**add**.**b**][**add**.**c**]** **=** g**[**add**.**c**][**add**.**a**]** **=** num**;**

F**[**num**++]** **=** add**;**

**}**

**for(**int i **=** 4**;** i **<** n**;** i**++)**

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

**if(**F**[**j**].**ok **&&** dblcmp**(**P**[**i**],**F**[**j**])** **>** eps**)**

**{**

dfs**(**i**,**j**);**

**break;**

**}**

int tmp **=** num**;**

num **=** 0**;**

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

**if(**F**[**i**].**ok**)**

F**[**num**++]** **=** F**[**i**];**

**}**

//表面积，测试：HDU3528

double area**()**

**{**

double res **=** 0**;**

**if(**n **==** 3**)**

**{**

Point3 p **=** cross**(**P**[**0**],**P**[**1**],**P**[**2**]);**

**return** p**.**len**()/**2**;**

**}**

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

res **+=** area**(**P**[**F**[**i**].**a**],**P**[**F**[**i**].**b**],**P**[**F**[**i**].**c**]);**

**return** res**/**2.0**;**

**}**

double volume**()**

**{**

double res **=** 0**;**

Point3 tmp **=** Point3**(**0**,**0**,**0**);**

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

res **+=** volume**(**tmp**,**P**[**F**[**i**].**a**],**P**[**F**[**i**].**b**],**P**[**F**[**i**].**c**]);**

**return** fabs**(**res**/**6**);**

**}**

//表面三角形个数

int triangle**()**

**{**

**return** num**;**

**}**

//表面多边形个数，测试：HDU3662

int polygon**()**

**{**

int res **=** 0**;**

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

**{**

bool flag **=** **true;**

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

**if(**same**(**i**,**j**))**

**{**

flag **=** 0**;**

**break;**

**}**

res **+=** flag**;**

**}**

**return** res**;**

**}**

//重心，测试：HDU4273

Point3 barycenter**()**

**{**

Point3 ans **=** Point3**(**0**,**0**,**0**);**

Point3 o **=** Point3**(**0**,**0**,**0**);**

double all **=** 0**;**

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

**{**

double vol **=** volume**(**o**,**P**[**F**[**i**].**a**],**P**[**F**[**i**].**b**],**P**[**F**[**i**].**c**]);**

ans **=** ans **+** **(((**o**+**P**[**F**[**i**].**a**]+**P**[**F**[**i**].**b**]+**P**[**F**[**i**].**c**])/**4.0**)\***vol**);**

all **+=** vol**;**

**}**

ans **=** ans**/**all**;**

**return** ans**;**

**}**

//点到面的距离，测试：HDU4273

double ptoface**(**Point3 p**,**int i**)**

**{**

double tmp1 **=** fabs**(**volume**(**P**[**F**[**i**].**a**],**P**[**F**[**i**].**b**],**P**[**F**[**i**].**c**],**p**));**

double tmp2 **=** **((**P**[**F**[**i**].**b**]-**P**[**F**[**i**].**a**])^(**P**[**F**[**i**].**c**]-**P**[**F**[**i**].**a**])).**len**();**

**return** tmp1**/**tmp2**;**

**}**

**};**

CH3D hull**;**

int main**()**

**{**

**while(**scanf**(**"%d"**,&**hull**.**n**)!=**EOF**)**

**{**

**for(**int i **=** 0**;** i **<** hull**.**n**;** i**++)** hull**.**P**[**i**].**input**();**

hull**.**create**();**

printf**(**"%d\n"**,**hull**.**polygon**());**

**}**

**return** 0**;**

**}**

## 网格

#define abs(x) ((x)>0?(x):-(x))

struct point

**{**

int x**,**y**;**

**};**

int gcd**(**int a**,**int b**)**

**{**

**return** b**?**gcd**(**b**,**a**%**b**):**a**;**

**}**

//多边形上的网格点个数

int grid\_onedge**(**int n**,**point**\*** p**)**

**{**

int i**,**ret**=**0**;**

**for** **(**i**=**0**;** i**<**n**;** i**++)** ret**+=**gcd**(**abs**(**p**[**i**].**x**-**p**[(**i**+**1**)%**n**].**x**),**abs**(**p**[**i**].**y**-**p**[(**i**+**1**)%**n**].**y**));**

**return** ret**;**

**}**

//多边形内的网格点个数

int grid\_inside**(**int n**,**point**\*** p**)**

**{**

int i**,**ret**=**0**;**

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

ret**+=**p**[(**i**+**1**)%**n**].**y**\*(**p**[**i**].**x**-**p**[(**i**+**2**)%**n**].**x**);**

**return** **(**abs**(**ret**)-**grid\_onedge**(**n**,**p**))/**2**+**1**;**

**}**

//0≤x<z ，y≥0，且 y\*X≤x\*Y围成的三角形的格点个数（包括边界）

LL pointnum**(**LL x**,** LL y**,** LL z**)**

**{**

LL tp**=**0**;**

LL c**;**

c**=**z**/**x**;**

tp**+=**c**\*((**x**+**1**)\*(**y**+**1**)/**2**-**y**);**

tp**+=**c**\*(**c**-**1**)/**2**\*(**x**\***y**);**

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

tp**+=**c**\***y**\***z**;**

**if(**z**>**0**)**

**{**

**if(**x**<=**y**)**

**{**

c**=**y**/**x**;**

tp**+=**c**\***z**\*(**z**-**1**)/**2**;**

tp**+=**pointnum**(**x**,**y**%**x**,**z**);**

**}**

**else**

**{**

c**=**y**\*(**z**-**1**)/**x**+**1**;**

tp**+=**c**\***z**;**

tp**-=**pointnum**(**y**,**x**,**c**)-**1**;**

**}**

**}**

**return** tp**;**

**}**

## 圆

#include <math.h>

#define eps 1e-8

struct point

**{**

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

**};**

double xmult**(**point p1**,**point p2**,**point p0**)**

**{**

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

**}**

double veclen**(**point p1**,**point p2**)**

**{**

**return** sqrt**((**p1**.**x**-**p2**.**x**)\*(**p1**.**x**-**p2**.**x**)+(**p1**.**y**-**p2**.**y**)\*(**p1**.**y**-**p2**.**y**));**

**}**

double disptoline**(**point p**,**point l1**,**point l2**)**

**{**

**return** fabs**(**xmult**(**p**,**l1**,**l2**))/**veclen**(**l1**,**l2**);**

**}**

point intersection**(**point u1**,**point u2**,**point v1**,**point v2**)**

**{**

point ret**=**u1**;**

double t**=((**u1**.**x**-**v1**.**x**)\*(**v1**.**y**-**v2**.**y**)-(**u1**.**y**-**v1**.**y**)\*(**v1**.**x**-**v2**.**x**))**

**/((**u1**.**x**-**u2**.**x**)\*(**v1**.**y**-**v2**.**y**)-(**u1**.**y**-**u2**.**y**)\*(**v1**.**x**-**v2**.**x**));**

ret**.**x**+=(**u2**.**x**-**u1**.**x**)\***t**;**

ret**.**y**+=(**u2**.**y**-**u1**.**y**)\***t**;**

**return** ret**;**

**}**

//判直线和圆相交,包括相切

int intersect\_line\_circle**(**point c**,**double r**,**point l1**,**point l2**)**

**{**

**return** disptoline**(**c**,**l1**,**l2**)<**r**+**eps**;**

**}**

//判线段和圆相交,包括端点和相切

int intersect\_seg\_circle**(**point c**,**double r**,**point l1**,**point l2**)**

**{**

double t1**=**veclen**(**c**,**l1**)-**r**,**t2**=**veclen**(**c**,**l2**)-**r**;**

point t**=**c**;**

**if** **(**t1**<**eps**||**t2**<**eps**)**

**return** t1**>-**eps**||**t2**>-**eps**;**

t**.**x**+=**l1**.**y**-**l2**.**y**;**

t**.**y**+=**l2**.**x**-**l1**.**x**;**

**return** xmult**(**l1**,**c**,**t**)\***xmult**(**l2**,**c**,**t**)<**eps**&&**disptoline**(**c**,**l1**,**l2**)-**r**<**eps**;**

**}**

//判圆和圆相交,包括相切（推荐第二种）

int intersect\_circle\_circle**(**point c1**,**double r1**,**point c2**,**double r2**)**

**{**

**return** veclen**(**c1**,**c2**)<**r1**+**r2**+**eps**&&**veclen**(**c1**,**c2**)>**fabs**(**r1**-**r2**)-**eps**;**

**}**

bool circleCrosscircle**(**point p1**,**double r1**,**point p2**,**double r2**,**point **&**cp1**,**point **&**cp2**)**

**{**

double tx**=**p2**.**x**-**p1**.**x**,** sx**=**p2**.**x**+**p1**.**x**,** tx2**=**tx**\***tx**;**

double ty**=**p2**.**y**-**p1**.**y**,** sy**=**p2**.**y**+**p1**.**y**,** ty2**=**ty**\***ty**;**

double sq**=**tx2**+**ty2**,** 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**=**tx**\*(** **(**r1**+**r2**)\*(**r1**-**r2**)+**tx**\***sx **)** **+** sx**\***ty2**;**

double y**=**ty**\*(** **(**r1**+**r2**)\*(**r1**-**r2**)+**ty**\***sy **)** **+** sy**\***tx2**;**

double dx**=**tx**\***d**,**dy**=**ty**\***d**;**

sq**\*=**2**;**

cp1**.**x**=(**x**-**dy**)/**sq**;**

cp1**.**y**=(**y**+**dx**)/**sq**;**

cp2**.**x**=(**x**+**dy**)/**sq**;**

cp2**.**y**=(**y**-**dx**)/**sq**;**

**return** 1**;**

**}**

//计算圆上到点p最近点,如p与圆心重合,返回p本身

point dot\_to\_circle**(**point c**,**double r**,**point p**)**

**{**

point u**,**v**;**

**if** **(**veclen**(**p**,**c**)<**eps**)** **return** p**;**

u**.**x**=**c**.**x**+**r**\***fabs**(**c**.**x**-**p**.**x**)/**veclen**(**c**,**p**);**

u**.**y**=**c**.**y**+**r**\***fabs**(**c**.**y**-**p**.**y**)/**veclen**(**c**,**p**)\*((**c**.**x**-**p**.**x**)\*(**c**.**y**-**p**.**y**)<**0**?-**1**:**1**);**

v**.**x**=**c**.**x**-**r**\***fabs**(**c**.**x**-**p**.**x**)/**veclen**(**c**,**p**);**

v**.**y**=**c**.**y**-**r**\***fabs**(**c**.**y**-**p**.**y**)/**veclen**(**c**,**p**)\*((**c**.**x**-**p**.**x**)\*(**c**.**y**-**p**.**y**)<**0**?-**1**:**1**);**

**return** veclen**(**u**,**p**)<**veclen**(**v**,**p**)?**u**:**v**;**

**}**

//计算直线与圆的交点,保证直线与圆有交点

//计算线段与圆的交点可用这个函数后判点是否在线段上

void intersection\_line\_circle**(**point c**,**double r**,**point l1**,**point l2**,**point**&** p1**,**point**&** p2**)**

**{**

point p**=**c**;**

double t**;**

p**.**x**+=**l1**.**y**-**l2**.**y**;**

p**.**y**+=**l2**.**x**-**l1**.**x**;**

p**=**intersection**(**p**,**c**,**l1**,**l2**);**

t**=**sqrt**(**r**\***r**-**veclen**(**p**,**c**)\***veclen**(**p**,**c**))/**veclen**(**l1**,**l2**);**

p1**.**x**=**p**.**x**+(**l2**.**x**-**l1**.**x**)\***t**;**

p1**.**y**=**p**.**y**+(**l2**.**y**-**l1**.**y**)\***t**;**

p2**.**x**=**p**.**x**-(**l2**.**x**-**l1**.**x**)\***t**;**

p2**.**y**=**p**.**y**-(**l2**.**y**-**l1**.**y**)\***t**;**

**}**

//计算圆与圆的交点,保证圆与圆有交点,圆心不重合

void intersection\_circle\_circle**(**point c1**,**double r1**,**point c2**,**double r2**,**point**&** p1**,**point**&** p2**)**

**{**

point u**,**v**;**

double t**;**

t**=(**1**+(**r1**\***r1**-**r2**\***r2**)/**veclen**(**c1**,**c2**)/**veclen**(**c1**,**c2**))/**2**;**

u**.**x**=**c1**.**x**+(**c2**.**x**-**c1**.**x**)\***t**;**

u**.**y**=**c1**.**y**+(**c2**.**y**-**c1**.**y**)\***t**;**

v**.**x**=**u**.**x**+**c1**.**y**-**c2**.**y**;**

v**.**y**=**u**.**y**-**c1**.**x**+**c2**.**x**;**

intersection\_line\_circle**(**c1**,**r1**,**u**,**v**,**p1**,**p2**);**

**}**

//最小圆覆盖，求出半径最小的圆覆盖给出的所有的点

struct Point

**{**

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

**};**

double veclen**(**Point a**,**Point b**)**

**{**

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

**}**

//求线段交点

Point intersection**(**Point u1**,** Point u2**,** Point v1**,** Point v2**)**

**{**

Point ans**=**u1**;**

double t**=((**u1**.**x**-**v1**.**x**)\*(**v1**.**y**-**v2**.**y**)-(**u1**.**y**-**v1**.**y**)\*(**v1**.**x **-** v2**.**x**))/**

**((**u1**.**x**-**u2**.**x**)\*(**v1**.**y**-**v2**.**y**)-(**u1**.**y**-**u2**.**y**)\*(**v1**.**x **-** v2**.**x**));**

ans**.**x**+=(**u2**.**x**-**u1**.**x**)\***t**;**

ans**.**y**+=(**u2**.**y**-**u1**.**y**)\***t**;**

**return** ans**;**

**}**

//计算三角形外接圆圆心

Point circumcenter**(**Point a**,** Point b**,** Point c**)**

**{**

Point ua**,** ub**,** va**,** vb**;**

ua**.**x **=** **(** a**.**x **+** b**.**x **)** **/** 2**;**

ua**.**y **=** **(** a**.**y **+** b**.**y **)** **/** 2**;**

ub**.**x **=** ua**.**x **-** a**.**y **+** b**.**y**;**

ub**.**y **=** ua**.**y **+** a**.**x **-** b**.**x**;**

va**.**x **=** **(** a**.**x **+** c**.**x **)** **/** 2**;**

va**.**y **=** **(** a**.**y **+** c**.**y **)** **/** 2**;**

vb**.**x **=** va**.**x **-** a**.**y **+** c**.**y**;**

vb**.**y **=** va**.**y **+** a**.**x **-** c**.**x**;**

**return** intersection**(**ua**,** ub**,** va**,** vb**);**

**}**

//返回所求圆的圆心ret与半径ansr

Point min\_center**(**vector**<**Point**>** input**,**double **&**ansr**)**

**{**

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

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

Point ret**=**input**[**0**];**

ansr**=**0**;**

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

**{**

**if(**veclen**(**input**[**i**],**ret**)-**ansr**>**eps**)**

**{**

ret**=**input**[**i**];**

ansr**=**0**;**

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

**{**

**if(**veclen**(**input**[**j**],**ret**)-**ansr**>**eps**)**

**{**

ret**.**x**=(**input**[**i**].**x**+**input**[**j**].**x**)/**2.0**;**

ret**.**y**=(**input**[**i**].**y**+**input**[**j**].**y**)/**2.0**;**

ansr**=**veclen**(**ret**,**input**[**j**]);**

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

**{**

**if(**veclen**(**ret**,**input**[**k**])-**ansr**>**eps**)**

**{**

ret**=**circumcenter**(**input**[**i**],**input**[**j**],**input**[**k**]);**

ansr**=**veclen**(**ret**,**input**[**k**]);**

**}**

**}**

**}**

**}**

**}**

**}**

**return** ret**;**

**}**

## 整数函数

//注意某些情况下整数运算会出界!

#define sign(a) ((a)>0?1:(((a)<0?-1:0)))

struct point

**{**

int x**,**y**;**

**};**

struct line

**{**

point a**,**b**;**

**};**

//计算cross product (P1-P0)x(P2-P0)

int xmult**(**point p1**,**point p2**,**point p0**)**

**{**

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

**}**

int xmult**(**int x1**,**int y1**,**int x2**,**int y2**,**int x0**,**int y0**)**

**{**

**return** **(**x1**-**x0**)\*(**y2**-**y0**)-(**x2**-**x0**)\*(**y1**-**y0**);**

**}**

//计算dot product (P1-P0).(P2-P0)

int dmult**(**point p1**,**point p2**,**point p0**)**

**{**

**return** **(**p1**.**x**-**p0**.**x**)\*(**p2**.**x**-**p0**.**x**)+(**p1**.**y**-**p0**.**y**)\*(**p2**.**y**-**p0**.**y**);**

**}**

int dmult**(**int x1**,**int y1**,**int x2**,**int y2**,**int x0**,**int y0**)**

**{**

**return** **(**x1**-**x0**)\*(**x2**-**x0**)+(**y1**-**y0**)\*(**y2**-**y0**);**

**}**

//判三点共线

int dots\_inline**(**point p1**,**point p2**,**point p3**)**

**{**

**return** **!**xmult**(**p1**,**p2**,**p3**);**

**}**

int dots\_inline**(**int x1**,**int y1**,**int x2**,**int y2**,**int x3**,**int y3**)**

**{**

**return** **!**xmult**(**x1**,**y1**,**x2**,**y2**,**x3**,**y3**);**

**}**

//判点是否在线段上,包括端点和部分重合

int dot\_online\_in**(**point p**,**line l**)**

**{**

**return** **!**xmult**(**p**,**l**.**a**,**l**.**b**)&&(**l**.**a**.**x**-**p**.**x**)\*(**l**.**b**.**x**-**p**.**x**)<=**0**&&(**l**.**a**.**y**-**p**.**y**)\*(**l**.**b**.**y**-**p**.**y**)<=**0**;**

**}**

int dot\_online\_in**(**point p**,**point l1**,**point l2**)**

**{**

**return** **!**xmult**(**p**,**l1**,**l2**)&&(**l1**.**x**-**p**.**x**)\*(**l2**.**x**-**p**.**x**)<=**0**&&(**l1**.**y**-**p**.**y**)\*(**l2**.**y**-**p**.**y**)<=**0**;**

**}**

int dot\_online\_in**(**int x**,**int y**,**int x1**,**int y1**,**int x2**,**int y2**)**

**{**

**return** **!**xmult**(**x**,**y**,**x1**,**y1**,**x2**,**y2**)&&(**x1**-**x**)\*(**x2**-**x**)<=**0**&&(**y1**-**y**)\*(**y2**-**y**)<=**0**;**

**}**

//判点是否在线段上,不包括端点

int dot\_online\_ex**(**point p**,**line l**)**

**{**

**return** dot\_online\_in**(**p**,**l**)&&(**p**.**x**!=**l**.**a**.**x**||**p**.**y**!=**l**.**a**.**y**)&&(**p**.**x**!=**l**.**b**.**x**||**p**.**y**!=**l**.**b**.**y**);**

**}**

int dot\_online\_ex**(**point p**,**point l1**,**point l2**)**

**{**

**return** dot\_online\_in**(**p**,**l1**,**l2**)&&(**p**.**x**!=**l1**.**x**||**p**.**y**!=**l1**.**y**)**

**&&(**p**.**x**!=**l2**.**x**||**p**.**y**!=**l2**.**y**);**

**}**

int dot\_online\_ex**(**int x**,**int y**,**int x1**,**int y1**,**int x2**,**int y2**)**

**{**

**return** dot\_online\_in**(**x**,**y**,**x1**,**y1**,**x2**,**y2**)&&(**x**!=**x1**||**y**!=**y1**)&&(**x**!=**x2**||**y**!=**y2**);**

**}**

//判两点在直线同侧,点在直线上返回0

int same\_side**(**point p1**,**point p2**,**line l**)**

**{**

**return** sign**(**xmult**(**l**.**a**,**p1**,**l**.**b**))\***xmult**(**l**.**a**,**p2**,**l**.**b**)>**0**;**

**}**

int same\_side**(**point p1**,**point p2**,**point l1**,**point l2**)**

**{**

**return** sign**(**xmult**(**l1**,**p1**,**l2**))\***xmult**(**l1**,**p2**,**l2**)>**0**;**

**}**

//判两点在直线异侧,点在直线上返回0

int opposite\_side**(**point p1**,**point p2**,**line l**)**

**{**

**return** sign**(**xmult**(**l**.**a**,**p1**,**l**.**b**))\***xmult**(**l**.**a**,**p2**,**l**.**b**)<**0**;**

**}**

int opposite\_side**(**point p1**,**point p2**,**point l1**,**point l2**)**

**{**

**return** sign**(**xmult**(**l1**,**p1**,**l2**))\***xmult**(**l1**,**p2**,**l2**)<**0**;**

**}**

//判两直线平行

int parallel**(**line u**,**line v**)**

**{**

**return** **(**u**.**a**.**x**-**u**.**b**.**x**)\*(**v**.**a**.**y**-**v**.**b**.**y**)==(**v**.**a**.**x**-**v**.**b**.**x**)\*(**u**.**a**.**y**-**u**.**b**.**y**);**

**}**

int parallel**(**point u1**,**point u2**,**point v1**,**point v2**)**

**{**

**return** **(**u1**.**x**-**u2**.**x**)\*(**v1**.**y**-**v2**.**y**)==(**v1**.**x**-**v2**.**x**)\*(**u1**.**y**-**u2**.**y**);**

**}**

//判两直线垂直

int perpendicular**(**line u**,**line v**)**

**{**

**return** **(**u**.**a**.**x**-**u**.**b**.**x**)\*(**v**.**a**.**x**-**v**.**b**.**x**)==-(**u**.**a**.**y**-**u**.**b**.**y**)\*(**v**.**a**.**y**-**v**.**b**.**y**);**

**}**

int perpendicular**(**point u1**,**point u2**,**point v1**,**point v2**)**

**{**

**return** **(**u1**.**x**-**u2**.**x**)\*(**v1**.**x**-**v2**.**x**)==-(**u1**.**y**-**u2**.**y**)\*(**v1**.**y**-**v2**.**y**);**

**}**

//判两线段相交,包括端点和部分重合

int intersect\_in**(**line u**,**line v**)**

**{**

**if** **(!**dots\_inline**(**u**.**a**,**u**.**b**,**v**.**a**)||!**dots\_inline**(**u**.**a**,**u**.**b**,**v**.**b**))**

**return** **!**same\_side**(**u**.**a**,**u**.**b**,**v**)&&!**same\_side**(**v**.**a**,**v**.**b**,**u**);**

**return** dot\_online\_in**(**u**.**a**,**v**)||**dot\_online\_in**(**u**.**b**,**v**)||**dot\_online\_in**(**v**.**a**,**u**)||**dot\_online\_in**(**v**.**b**,**u**);**

**}**

int intersect\_in**(**point u1**,**point u2**,**point v1**,**point v2**)**

**{**

**if** **(!**dots\_inline**(**u1**,**u2**,**v1**)||!**dots\_inline**(**u1**,**u2**,**v2**))** **return** **!**same\_side**(**u1**,**u2**,**v1**,**v2**)&&!**same\_side**(**v1**,**v2**,**u1**,**u2**);**

**return** dot\_online\_in**(**u1**,**v1**,**v2**)||**dot\_online\_in**(**u2**,**v1**,**v2**)||**dot\_online\_in**(**v1**,**u1**,**u2**)||**dot\_online\_in**(**v2**,**u1**,**u2**);**

**}**

//判两线段相交,不包括端点和部分重合

int intersect\_ex**(**line u**,**line v**)**

**{**

**return** opposite\_side**(**u**.**a**,**u**.**b**,**v**)&&**opposite\_side**(**v**.**a**,**v**.**b**,**u**);**

**}**

int intersect\_ex**(**point u1**,**point u2**,**point v1**,**point v2**)**

**{**

**return** opposite\_side**(**u1**,**u2**,**v1**,**v2**)&&**opposite\_side**(**v1**,**v2**,**u1**,**u2**);**

**}**

## 注意

1. 注意舍入方式(0.5的舍入方向);防止输出—0的情况.
2. 注意在四舍五入的时候，考虑是否需要在输出的时候加减eps（四舍五入容易使结果略大，如四舍五入到2位小数有可能使得结果偏大0.01）
3. 四舍五入的位数尽量比要求的误差值多（如要求误差控制在1e-8，则四舍五入可以考虑精确到10位或更高）
4. 几何题注意多测试不对称数据.
5. 整数几何注意xmult和dmult是否会出界，符点几何注意eps的使用.
6. 避免使用斜率;注意除数是否会为0.
7. 公式一定要化简后再代入.
8. 判断同一个2\*PI域内两角度差应该是
   1. abs(a1-a2)<beta||abs(a1-a2)>pi+pi-beta;
   2. 相等应该是
   3. abs(a1-a2)<eps||abs(a1-a2)>pi+pi-eps;
9. 需要的话尽量使用atan2,注意:atan2(0,0)=0,
   1. atan2(1,0)=pi/2,atan2(-1,0)=-pi/2,atan2(0,1)=0,atan2(0,-1)=pi.
10. cross product = |u|\*|v|\*sin(a)

dot product = |u|\*|v|\*cos(a)

1. (P1-P0)x(P2-P0)结果的意义:
   1. 正: <P0,P1>在<P0,P2>顺时针(0,pi)内
   2. 负: <P0,P1>在<P0,P2>逆时针(0,pi)内
   3. 0 : <P0,P1>,<P0,P2>共线,夹角为0或pi
2. 误差限缺省使用1e-8!
3. 多边形的面积交可以转化为半平面交的问题

# 其他

## 进制转换

//将十进制s转换为b(>=2)进制，ans[pos]记录转换后第pos位的数（pos=0为最低位）

void base**(**int x**,**int b**)**

**{**

pos**=**0**;**

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

**{**

ans**[**pos**]=**x**%**b**;**

x**/=**b**;**

pos**++;**

**}**

**}**

//将十进制数x转换成-b进制（如转换成-2进制），ans[pos]记录转换后的的第pos位数（pos=0为最低位）

void base\_minus**(**int x**,**int b**)**

**{**

pos**=**0**;**

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

**{**

ans**[**pos**]=(**x**%**b**+**b**)%**b**;**

**if(**pos**%**2**==**0**)** x**=(**x**-**ans**[**pos**])/**b**;**

**else** x**=(**x**+**ans**[**pos**])/**b**;**

pos**++;**

**}**

**}**

## 统计二进制数1的个数和

//统计区间[a,b]内十进制数化成二进制数之后1的总个数

int Count**(**int n**,**int m**)**

**{**

**if((**n**>>**m**)==**0**)** **return** 0**;**

int d**=(**n**>>**m**)&**1**;**

**if(**d**==**1**)** **return** **(**n**>>(**m**+**1**))\*(**1**<<**m**)+(**n**&((**1**<<**m**)-**1**))+**1**+**Count**(**n**,**m**+**1**);**

**else** **return** **(**n**>>(**m**+**1**))\*(**1**<<**m**)+**Count**(**n**,**m**+**1**);**

**}**

int num\_one**(**int a**,**int b**)**

**{**

int n**=(**a**==**0**?**1**:**a**);**

**return** Count**(**b**,**0**)-**Count**(**n**-**1**,**0**);**

**}**

## 分数

struct frac

**{**

int num**,**den**;**

**};**

double fabs**(**double x**)**

**{**

**return** x**>**0**?**x**:-**x**;**

**}**

int gcd**(**int a**,**int b**)**

**{**

int t**;**

**if** **(**a**<**0**)** a**=-**a**;**

**if** **(**b**<**0**)** b**=-**b**;**

**if** **(!**b**)** **return** a**;**

**while** **(**t**=**a**%**b**)**a**=**b**,**b**=**t**;**

**return** b**;**

**}**

void simplify**(**frac**&** f**)**

**{**

int t**;**

**if** **(**t**=**gcd**(**f**.**num**,**f**.**den**))** f**.**num**/=**t**,**f**.**den**/=**t**;**

**else** f**.**den**=**1**;**

**}**

frac f**(**int n**,**int d**,**int s**=**1**)**

**{**

frac ret**;**

**if** **(**d**<**0**)** ret**.**num**=-**n**,**ret**.**den**=-**d**;**

**else** ret**.**num**=**n**,**ret**.**den**=**d**;**

**if** **(**s**)** simplify**(**ret**);**

**return** ret**;**

**}**

frac convert**(**double x**)**

**{**

frac ret**;**

**for** **(**ret**.**den**=**1**;** fabs**(**x**-**int**(**x**))>**1e-10**;** ret**.**den**\*=**10**,**x**\*=**10**);**

ret**.**num**=(**int**)**x**;**

simplify**(**ret**);**

**return** ret**;**

**}**

int fraqcmp**(**frac a**,**frac b**)**

**{**

int g1**=**gcd**(**a**.**den**,**b**.**den**),**g2**=**gcd**(**a**.**num**,**b**.**num**);**

**if** **(!**g1**||!**g2**)** **return** 0**;**

**return** b**.**den**/**g1**\*(**a**.**num**/**g2**)-**a**.**den**/**g1**\*(**b**.**num**/**g2**);**

**}**

frac add**(**frac a**,**frac b**)**

**{**

int g1**=**gcd**(**a**.**den**,**b**.**den**),**g2**,**t**;**

**if** **(!**g1**)** **return** f**(**1**,**0**,**0**);**

t**=**b**.**den**/**g1**\***a**.**num**+**a**.**den**/**g1**\***b**.**num**;**

g2**=**gcd**(**g1**,**t**);**

**return** f**(**t**/**g2**,**a**.**den**/**g1**\*(**b**.**den**/**g2**),**0**);**

**}**

frac sub**(**frac a**,**frac b**)**

**{**

**return** add**(**a**,**f**(-**b**.**num**,**b**.**den**,**0**));**

**}**

frac mul**(**frac a**,**frac b**)**

**{**

int t1**=**gcd**(**a**.**den**,**b**.**num**),**t2**=**gcd**(**a**.**num**,**b**.**den**);**

**if** **(!**t1**||!**t2**)** **return** f**(**1**,**1**,**0**);**

**return** f**(**a**.**num**/**t2**\*(**b**.**num**/**t1**),**a**.**den**/**t1**\*(**b**.**den**/**t2**),**0**);**

**}**

frac div**(**frac a**,**frac b**)**

**{**

**return** mul**(**a**,**f**(**b**.**den**,**b**.**num**,**0**));**

**}**

## 矩阵

#define MAXN 100

#define fabs(x) ((x)>0?(x):-(x))

#define zero(x) (fabs(x)<1e-10)

struct mat

**{**

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

double data**[**MAXN**][**MAXN**];**

**};**

int mul**(**mat**&** c**,**const mat**&** a**,**const mat**&** b**)**

**{**

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

**if** **(**a**.**m**!=**b**.**n**)**

**return** 0**;**

c**.**n**=**a**.**n**,**c**.**m**=**b**.**m**;**

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

**for** **(**j**=**0**;** j**<**c**.**m**;** j**++)**

**for** **(**c**.**data**[**i**][**j**]=**k**=**0**;** k**<**a**.**m**;** k**++)**

c**.**data**[**i**][**j**]+=**a**.**data**[**i**][**k**]\***b**.**data**[**k**][**j**];**

**return** 1**;**

**}**

int inv**(**mat**&** a**)**

**{**

int i**,**j**,**k**,**is**[**MAXN**],**js**[**MAXN**];**

double t**;**

**if** **(**a**.**n**!=**a**.**m**)**

**return** 0**;**

**for** **(**k**=**0**;** k**<**a**.**n**;** k**++)**

**{**

**for** **(**t**=**0**,**i**=**k**;** i**<**a**.**n**;** i**++)**

**for** **(**j**=**k**;** j**<**a**.**n**;** j**++)**

**if** **(**fabs**(**a**.**data**[**i**][**j**])>**t**)**

t**=**fabs**(**a**.**data**[**is**[**k**]=**i**][**js**[**k**]=**j**]);**

**if** **(**zero**(**t**))** **return** 0**;**

**if** **(**is**[**k**]!=**k**)**

**for** **(**j**=**0**;** j**<**a**.**n**;** j**++)**

t**=**a**.**data**[**k**][**j**],**a**.**data**[**k**][**j**]=**a**.**data**[**is**[**k**]][**j**] ,** a**.**data**[**is**[**k**]][**j**]=**t**;**

**if** **(**js**[**k**]!=**k**)**

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

t**=**a**.**data**[**i**][**k**],**a**.**data**[**i**][**k**]=**a**.**data**[**i**][**js**[**k**]] ,** a**.**data**[**i**][**js**[**k**]]=**t**;**

a**.**data**[**k**][**k**]=**1**/**a**.**data**[**k**][**k**];**

**for** **(**j**=**0**;** j**<**a**.**n**;** j**++)**

**if** **(**j**!=**k**)** a**.**data**[**k**][**j**]\*=**a**.**data**[**k**][**k**];**

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

**if** **(**i**!=**k**)**

**for** **(**j**=**0**;** j**<**a**.**n**;** j**++)**

**if** **(**j**!=**k**)**

a**.**data**[**i**][**j**]-=**a**.**data**[**i**][**k**]\***a**.**data**[**k**][**j**];**

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

**if** **(**i**!=**k**)**a**.**data**[**i**][**k**]\*=-**a**.**data**[**k**][**k**];**

**}**

**for** **(**k**=**a**.**n**-**1**;** k**>=**0**;** k**--)**

**{**

**for** **(**j**=**0**;** j**<**a**.**n**;** j**++)**

**if** **(**js**[**k**]!=**k**)**

t**=**a**.**data**[**k**][**j**],**a**.**data**[**k**][**j**]=**a**.**data**[**js**[**k**]][**j**],**a**.**data**[**js**[**k**]][**j**]=**t**;**

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

**if** **(**is**[**k**]!=**k**)**

t**=**a**.**data**[**i**][**k**],**a**.**data**[**i**][**k**]=**a**.**data**[**i**][**is**[**k**]],**a**.**data**[**i**][**is**[**k**]]=**t**;**

**}**

**return** 1**;**

**}**

double det**(**const mat**&** a**)**

**{**

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

double b**[**MAXN**][**MAXN**],**ret**=**1**,**t**;**

**if** **(**a**.**n**!=**a**.**m**)**

**return** 0**;**

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

**for** **(**j**=**0**;** j**<**a**.**m**;** j**++)**

b**[**i**][**j**]=**a**.**data**[**i**][**j**];**

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

**{**

**if** **(**zero**(**b**[**i**][**i**]))**

**{**

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

**if** **(!**zero**(**b**[**j**][**i**]))**

**break;**

**if** **(**j**==**a**.**n**)**

**return** 0**;**

**for** **(**k**=**i**;** k**<**a**.**n**;** k**++)**

t**=**b**[**i**][**k**],**b**[**i**][**k**]=**b**[**j**][**k**],**b**[**j**][**k**]=**t**;**

sign**++;**

**}**

ret**\*=**b**[**i**][**i**];**

**for** **(**k**=**i**+**1**;** k**<**a**.**n**;** k**++)**

b**[**i**][**k**]/=**b**[**i**][**i**];**

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

**for** **(**k**=**i**+**1**;** k**<**a**.**n**;** k**++)**

b**[**j**][**k**]-=**b**[**j**][**i**]\***b**[**i**][**k**];**

**}**

**if** **(**sign**&**1**)** ret**=-**ret**;**

**return** ret**;**

**}**

## 日期

//日期函数

int days**[**12**]=** **{**31**,**28**,**31**,**30**,**31**,**30**,**31**,**31**,**30**,**31**,**30**,**31**};**

struct date

**{**

int year**,**month**,**day**;**

**};**

//判闰年

inline int leap**(**int year**)**

**{**

**return** **(**year**%**4**==**0**&&**year**%**100**!=**0**)||**year**%**400**==**0**;**

**}**

//判合法性

inline int legal**(**date a**)**

**{**

**if** **(**a**.**month**<**0**||**a**.**month**>**12**)** **return** 0**;**

**if** **(**a**.**month**==**2**)**

**return** a**.**day**>**0**&&**a**.**day**<=**28**+**leap**(**a**.**year**);**

**return** a**.**day**>**0**&&**a**.**day**<=**days**[**a**.**month**-**1**];**

**}**

//比较日期大小

inline int datecmp**(**date a**,**date b**)**

**{**

**if** **(**a**.**year**!=**b**.**year**)**

**return** a**.**year**-**b**.**year**;**

**if** **(**a**.**month**!=**b**.**month**)**

**return** a**.**month**-**b**.**month**;**

**return** a**.**day**-**b**.**day**;**

**}**

//返回指定日期是星期几

int weekday**(**date a**)**

**{**

int tm**=**a**.**month**>=**3**?(**a**.**month**-**2**):(**a**.**month**+**10**);**

int ty**=**a**.**month**>=**3**?**a**.**year**:(**a**.**year**-**1**);**

**return** **(**ty**+**ty**/**4**-**ty**/**100**+**ty**/**400**+(**int**)(**2.6**\***tm**-**0.2**)+**a**.**day**)%**7**;**

**}**

//日期转天数偏移

int date2int**(**date a**)**

**{**

int ret**=**a**.**year**\***365**+(**a**.**year**-**1**)/**4**-(**a**.**year**-**1**)/**100**+(**a**.**year**-**1**)/**400**,**i**;**

days**[**1**]+=**leap**(**a**.**year**);**

**for** **(**i**=**0**;** i**<**a**.**month**-**1**;** ret**+=**days**[**i**++]);**

days**[**1**]=**28**;**

**return** ret**+**a**.**day**;**

**}**

//天数偏移转日期

date int2date**(**int a**)**

**{**

date ret**;**

ret**.**year**=**a**/**146097**\***400**;**

**for** **(**a**%=**146097**;** a**>=**365**+**leap**(**ret**.**year**);** a**-=**365**+**leap**(**ret**.**year**),**ret**.**year**++);**

days**[**1**]+=**leap**(**ret**.**year**);**

**for(**ret**.**month**=**1**;**a**>=**days**[**ret**.**month**-**1**];**a**-=**days**[**ret**.**month**-**1**],**ret**.**month**++);**

days**[**1**]=**28**;**

ret**.**day**=**a**+**1**;**

**return** ret**;**

**}**

## 线性方程组(gauss)

//仅适用于b[]非全零的情况

#define MAXN 100

#define fabs(x) ((x)>0?(x):-(x))

#define eps 1e-10

//列主元gauss消去求解a[][]x[]=b[]

//返回是否有唯一解,若有解在b[]中

int gauss\_cpivot**(**int n**,**double a**[][**MAXN**],**double b**[])**

**{**

int i**,**j**,**k**,**row**;**

double maxp**,**t**;**

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

**{**

**for** **(**maxp**=**0**,**i**=**k**;** i**<**n**;** i**++)**

**if** **(**fabs**(**a**[**i**][**k**])>**fabs**(**maxp**))**

maxp**=**a**[**row**=**i**][**k**];**

**if** **(**fabs**(**maxp**)<**eps**)** **return** 0**;**

**if** **(**row**!=**k**)**

**{**

**for** **(**j**=**k**;** j**<**n**;** j**++)**

t**=**a**[**k**][**j**],**a**[**k**][**j**]=**a**[**row**][**j**],**a**[**row**][**j**]=**t**;**

t**=**b**[**k**],**b**[**k**]=**b**[**row**],**b**[**row**]=**t**;**

**}**

**for** **(**j**=**k**+**1**;** j**<**n**;** j**++)**

**{**

a**[**k**][**j**]/=**maxp**;**

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

a**[**i**][**j**]-=**a**[**i**][**k**]\***a**[**k**][**j**];**

**}**

b**[**k**]/=**maxp**;**

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

b**[**i**]-=**b**[**k**]\***a**[**i**][**k**];**

**}**

**for** **(**i**=**n**-**1**;** i**>=**0**;** i**--)**

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

b**[**i**]-=**a**[**i**][**j**]\***b**[**j**];**

**return** 1**;**

**}**

//取模高斯消元

//返回是否有唯一解，唯一解在b[]中

#define MAXN 300

#define Mod 2

int idx**[**MAXN**];**

LL a**[**MAXN**][**MAXN**],**b**[**MAXN**];**

//m,n分别是a矩阵的行数与列数

int gaussTpivotMod**(**int m**,**int n**)**

**{**

int i**,**j**,**k**,**row**,**col**;**

LL maxp**,**t**;**

int ret**=**1**;**

**for(**i**=**0**;** i**<**m**;** i**++)**idx**[**i**]=**i**;**

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

**{**

**for(**maxp**=**0**,**i**=**k**;** i**<**m**;** i**++)**

**for(**j**=**k**;** j**<**n**;** j**++)**

**if(**a**[**i**][**j**]>**maxp**)**

maxp**=**a**[**row**=**i**][**col**=**j**];**

**if(**maxp**==**0**)**

**{**

bool fail**=**0**;**

**for(**i**=**k**;** i**<**m**;** i**++)if(**b**[**i**])**fail**=**1**;**

**if(**fail**)return** 0**;**

ret**=**k**-**n**;**

**break;**

**}**

**if(**col**!=**k**)for(**swap**(**idx**[**col**],**idx**[**k**]),**i**=**0**;** i**<**m**;** i**++)**

swap**(**a**[**i**][**col**],**a**[**i**][**k**]);**

**if(**row**!=**k**)for(**swap**(**b**[**k**],**b**[**row**]),**j**=**k**;** j**<**n**;** j**++)**

swap**(**a**[**k**][**j**],**a**[**row**][**j**]);**

LL inv**=**modInv**(**maxp**,**Mod**);**//逆元

**for(**i**=**k**+**1**;** i**<**m**;** i**++)**

**{**

LL mul**=**inv**\***a**[**i**][**k**];**

**for(**j**=**k**;** j**<**n**;** j**++)**

a**[**i**][**j**]-=**a**[**k**][**j**]\***mul**;**

b**[**i**]-=**mul**\***b**[**k**];**

**}**

**for(**i**=**k**;** i**<**m**;** i**++)**

**{**

**for(**j**=**k**;** j**<**n**;** j**++)**

**{**

LL **&**tmp**=**a**[**i**][**j**];**

**if(**tmp**>=**Mod**)**tmp**%=**Mod**;**

**if(**tmp**<**0**)**tmp**=**tmp**%**Mod**+**Mod**;**

**}**

**}**

**}**

**for(**i**=**n**-**1**;** i**>=**0**;** i**--)**

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

b**[**i**]-=**a**[**i**][**j**]\***b**[**j**];**

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

a**[**0**][**idx**[**k**]]=**b**[**k**];**

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

b**[**k**]=(**a**[**0**][**k**]%**Mod**+**Mod**)%**Mod**;**

**return** ret**;**

**}**

//全主元gauss消去解a[][]x[]=b[]

//返回是否有唯一解,若有解在b[]中

int gauss\_tpivot**(**int n**,**double a**[][**MAXN**],**double b**[])**

**{**

int i**,**j**,**k**,**row**,**col**,**index**[**MAXN**];**

double maxp**,**t**;**

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

index**[**i**]=**i**;**

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

**{**

**for** **(**maxp**=**0**,**i**=**k**;** i**<**n**;** i**++)**

**for** **(**j**=**k**;** j**<**n**;** j**++)**

**if** **(**fabs**(**a**[**i**][**j**])>**fabs**(**maxp**))**

maxp**=**a**[**row**=**i**][**col**=**j**];**

**if** **(**fabs**(**maxp**)<**eps**)**

**return** 0**;**

**if** **(**col**!=**k**)**

**{**

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

t**=**a**[**i**][**col**],**a**[**i**][**col**]=**a**[**i**][**k**],**a**[**i**][**k**]=**t**;**

j**=**index**[**col**],**index**[**col**]=**index**[**k**],**index**[**k**]=**j**;**

**}**

**if** **(**row**!=**k**)**

**{**

**for** **(**j**=**k**;** j**<**n**;** j**++)**

t**=**a**[**k**][**j**],**a**[**k**][**j**]=**a**[**row**][**j**],**a**[**row**][**j**]=**t**;**

t**=**b**[**k**],**b**[**k**]=**b**[**row**],**b**[**row**]=**t**;**

**}**

**for** **(**j**=**k**+**1**;** j**<**n**;** j**++)**

**{**

a**[**k**][**j**]/=**maxp**;**

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

a**[**i**][**j**]-=**a**[**i**][**k**]\***a**[**k**][**j**];**

**}**

b**[**k**]/=**maxp**;**

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

b**[**i**]-=**b**[**k**]\***a**[**i**][**k**];**

**}**

**for** **(**i**=**n**-**1**;** i**>=**0**;** i**--)**

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

b**[**i**]-=**a**[**i**][**j**]\***b**[**j**];**

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

a**[**0**][**index**[**k**]]=**b**[**k**];**

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

b**[**k**]=**a**[**0**][**k**];**

**return** 1**;**

**}**

## 归并排序求逆序对（O（n log n））

//对数组num中的[first,last]区间进行归并排序,函数mergesort返回的全局参数cnt为[first,last]区间中的逆序对数

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

void merge\_num**(**int num**[],**int p**,**int q**,**int r**)**

**{**

int begin1**,**end1**,**begin2**,**end2**,**k**,**temp**[**MAXN**];**

begin1**=**p**;**

end1**=**q**;**

begin2**=**q**+**1**;**

end2**=**r**;**

k**=**0**;**

**while(**begin1**<=**end1 **&&** begin2**<=**end2**)**

**{**

**if(**num**[**begin1**]** **<=** num**[**begin2**])**

**{**

temp**[**k**++]=**num**[**begin1**];**

begin1**++;**

cnt**+=**begin2**-(**q**+**1**);**

**}**

**else**

**{**

temp**[**k**++]=**num**[**begin2**];**

begin2**++;**

**}**

**}**

**while(**begin1**<=**end1**)**

**{**

temp**[**k**++]=**num**[**begin1**];**

begin1**++;**

cnt**+=**end2**-**q**;**

**}**

**while(**begin2**<=**end2**)**

**{**

temp**[**k**++]=**num**[**begin2**];**

begin2**++;**

**}**

**for(**int i**=**p**;** i**<=**r**;** i**++)**

num**[**i**]=**temp**[**i**-**p**];**

**}**

void mergesort**(**int num**[],**int first**,**int last**)**

**{**

int mid**=(**first**+**last**)/**2**;**

**if(**first**<**last**)**

**{**

mergesort**(**num**,**first**,**mid**);**

mergesort**(**num**,**mid**+**1**,**last**);**

merge\_num**(**num**,**first**,**mid**,**last**);**

**}**

**}**

## 快速幂

//(m^n)%k

int quickpow**(**int m**,**int n**,**int k**)**

**{**

int b **=** 1**;**

**while** **(**n **>** 0**)**

**{**

**if** **(**n **&** 1**)** b **=** **(**b**\***m**)%**k**;**

n **=** n **>>** 1 **;**

m **=** **(**m**\***m**)%**k**;**

**}**

**return** b**;**

**}**

//矩阵乘法，定义“ \* ”运算

//以结构体定义矩阵

struct Mat

**{**

double mat**[**N**][**N**];**

**};**

Mat **operator** **\*** **(**Mat a**,** Mat b**)**

**{**

Mat c**;**

memset**(**c**.**mat**,** 0**,** **sizeof(**c**.**mat**));**

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

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

**{**

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

**{**

**if(**a**.**mat**[**i**][**k**]** **<=** 0**)** **continue;**

//这两行continue最好不加,除非保证矩阵元素非负

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

**{**

**if(**b**.**mat**[**k**][**j**]** **<=** 0**)** **continue;** //剪枝

c**.**mat**[**i**][**j**]** **+=** a**.**mat**[**i**][**k**]** **\*** b**.**mat**[**k**][**j**];**

**}**

**}**

**}**

**return** c**;**

**}**

//矩阵快速幂，定义“ ^ ”运算

Mat **operator** **^** **(**Mat a**,** int k**)**

**{**

Mat c**;**

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

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

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

c**.**mat**[**i**][**j**]** **=** **(**i **==** j**);** //矩阵c初始化为单位矩阵

**for(;** k**;** k **>>=** 1**)**

**{**

**if(**k**&**1**)** c **=** c**\***a**;**

a **=** a**\***a**;**

**}**

**return** c**;**

**}**

## 母函数（不限制个数，O( n^3 )）

//验题 uva 674

//idx[i]为中间变量，coe[i]表示的是x^i的系数

//dir数组存储的是母函数中每一个乘法多项式的基数，如这里的母函数为 (1+x+…)（1+x^5+…）(1+x^10+…)（1+x^25+…）（1+x^50+…）

LL idx**[**MAXN**],**coe**[**MAXN**];**

int dir**[**5**]=** **{**1**,**5**,**10**,**25**,**50**};**

void init**()**

**{**

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

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

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

**{**

coe**[**i**]=**1**;**

idx**[**i**]=**0**;**

**}**

**for(**i**=**1**;** i**<**5**;** i**++)** //这里的5是dir的长度

**{**

int ii**=**dir**[**i**];**

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

**{**

**for(**k**=**0**;** k**+**j**<**MAXN**;** k**+=**ii**)**

**{**

idx**[**j**+**k**]+=**coe**[**j**];**

**}**

**}**

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

**{**

coe**[**j**]=**idx**[**j**];**

idx**[**j**]=**0**;**

**}**

**}**

**}**

## 母函数（限制个数，O( n^4 )）

//验题 hdu 2069

// coe[i][j]表示的是x^i且用不超过j个数表示系数

//100指的是限制数目，即总共只能用100个构成x^i

#define MAXN 500+5

#define MINN 100+5

LL idx**[**MAXN**][**MINN**],**coe**[**MAXN**][**MINN**],**ret**[**MAXN**];**

int dir**[**6**]=** **{**0**,**1**,**5**,**10**,**25**,**50**};**

void init**()**

**{**

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

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

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

int i**,**j**,**k**,**l**;**

coe**[**0**][**0**]=**1**;**

**for(**i**=**1**;** i**<**6**;** i**++)**

**{**

int ii**=**dir**[**i**];**

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

**{**

**for(**k**=**0**;** k**+**j**<**MAXN**;** k**+=**ii**)**

**{**

**for(**l**=**0**;** l**+**k**/**ii**<=**100**;** l**++)**

**{**

idx**[**k**+**j**][**l**+**k**/**ii**]+=**coe**[**j**][**l**];**

**}**

**}**

**}**

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

**{**

**for(**k**=**0**;** k**<=**100**;** k**++)**

**{**

coe**[**j**][**k**]=**idx**[**j**][**k**];**

idx**[**j**][**k**]=**0**;**

**}**

**}**

**}**

**for(**i**=**1**;** i**<=**MAXN**;** i**++)**

**{**

**for(**j**=**0**;** j**<=**100**;** j**++)**

**{**

ret**[**i**]+=**coe**[**i**][**j**];**

**}**

ret**[**0**]=**1**;**

**}**

**}**

## 稳定婚姻问题（Gale\_Shapley算法）

//n个男生的，n个女生的，已知每个男生中女生的喜欢程度，和女生中男生的喜欢程度，要求配成n对婚姻使其稳定

//rmw[i][j]代表i男对女生的喜欢排名

//lwm[i][j]代表i女对j男的喜欢程度

const int MAX **=** 510**;**

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

int rmw**[**MAX**][**MAX**];**

int lmw**[**MAX**][**MAX**],** lwm**[**MAX**][**MAX**];**

int couple**[**MAX**];**

char sman**[**MAX**][**110**],** swoman**[**MAX**][**110**];**

queue**<**int**>** SQ**;**

void gale\_shapley**()**

**{**

int i**,**man**,**woman**;**

**while(!**SQ**.**empty**())** SQ**.**pop**();**

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

**for(**i**=**1**;** i**<=**n**;** i**++)** SQ**.**push**(**i**);**

**while(!**SQ**.**empty**())**

**{**

man **=** SQ**.**front**();**

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

**{**

**if(**rmw**[**man**][**i**]** **!=** **-**1**)**

**{**

//选择为被拒绝且最喜欢的女生

woman **=** rmw**[**man**][**i**];**

rmw**[**man**][**i**]** **=** **-**1**;**

int pre **=** couple**[**woman**];**

**if(**pre **==** **-**1**)**

**{**

couple**[**woman**]** **=** man**;**

SQ**.**pop**();**

**break;**

**}**

**else**

**{**

**if(**lwm**[**woman**][**man**]** **>** lwm**[**woman**][**pre**])**

**{**

SQ**.**pop**();**

SQ**.**push**(**pre**);**

couple**[**woman**]** **=** man**;**

**break;**

**}**

**}**

**}**

**}**

**}**

**}**

# 数论

## 阶乘最后非零位

//求阶乘最后非零位,复杂度O(nlogn)

//返回该位,n以字符串方式传入

#include <string.h>

#define MAXN 10000

int lastdigit**(**char**\*** buf**)**

**{**

const int mod**[**20**]=** **{**1**,**1**,**2**,**6**,**4**,**2**,**2**,**4**,**2**,**8**,**4**,**4**,**8**,**4**,**6**,**8**,**8**,**6**,**8**,**2**};**

int len**=**strlen**(**buf**),**a**[**MAXN**],**i**,**c**,**ret**=**1**;**

**if** **(**len**==**1**)**

**return** mod**[**buf**[**0**]-**'0'**];**

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

a**[**i**]=**buf**[**len**-**1**-**i**]-**'0'**;**

**for** **(;** len**;** len**-=!**a**[**len**-**1**])**

**{**

ret**=**ret**\***mod**[**a**[**1**]%**2**\***10**+**a**[**0**]]%**5**;**

**for** **(**c**=**0**,**i**=**len**-**1**;** i**>=**0**;** i**--)**

c**=**c**\***10**+**a**[**i**],**a**[**i**]=**c**/**5**,**c**%=**5**;**

**}**

**return** ret**+**ret**%**2**\***5**;**

**}**

## 模线性方程(组)

#ifdef WIN32

**typedef** \_\_int64 i64**;**

#else

**typedef** long long i64**;**

#endif

//扩展Euclid求解gcd(a,b)=ax+by

int ext\_gcd**(**int a**,**int b**,**int**&** x**,**int**&** y**)**

**{**

int t**,**ret**;**

**if** **(!**b**)**

**{**

x**=**1**,**y**=**0**;**

**return** a**;**

**}**

ret**=**ext\_gcd**(**b**,**a**%**b**,**x**,**y**);**

t**=**x**,**x**=**y**,**y**=**t**-**a**/**b**\***y**;**

**return** ret**;**

**}**

//计算m^a, O(loga), 本身没什么用, 注意这个按位处理的方法 :-P

int exponent**(**int m**,**int a**)**

**{**

int ret**=**1**;**

**for** **(;** a**;** a**>>=**1**,**m**\*=**m**)**

**if** **(**a**&**1**)**

ret**\*=**m**;**

**return** ret**;**

**}**

// 求a的逆元x，ax ≡ 1 (mod n)

//若n为质数，由费马小定理(a^(n-1))%n=1，可得a^(n-2)是a关于n的逆元

int Inv**(**int a**,** int n**)**

**{**

int d**,** x**,** y**;**

d **=** exgcd**(**a**,** n**,** x**,** y**);**

**if(**d **==** 1**)** **return** **(**x**%**n **+** n**)** **%** n**;**

**else** **return** **-**1**;** // no solution

**}**

//计算幂取模a^b mod n, O(logb)

int modular\_exponent**(**int a**,**int b**,**int n**)** //a^b mod n

**{**

int ret**=**1**;**

**for** **(;** b**;** b**>>=**1**,**a**=(**int**)((**i64**)**a**)\***a**%**n**)**

**if** **(**b**&**1**)**

ret**=(**int**)((**i64**)**ret**)\***a**%**n**;**

**return** ret**;**

**}**

//求解模线性方程ax=b (mod n)

//返回解的个数,解保存在sol[]中

//要求n>0,解的范围0..n-1

int modular\_linear**(**int a**,**int b**,**int n**,**int**\*** sol**)**

**{**

int d**,**e**,**x**,**y**,**i**;**

d**=**ext\_gcd**(**a**,**n**,**x**,**y**);**

**if** **(**b**%**d**)**

**return** 0**;**

e**=(**x**\*(**b**/**d**)%**n**+**n**)%**n**;**

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

sol**[**i**]=(**e**+**i**\*(**n**/**d**))%**n**;**

**return** d**;**

**}**

//求解模线性方程组(中国余数定理)

// x = b[0] (mod w[0])

// x = b[1] (mod w[1])

// ...

// x = b[k-1] (mod w[k-1])

//要求w[i]>0,w[i]与w[j]互质,解的范围1..n,n=w[0]\*w[1]\*...\*w[k-1]

int modular\_linear\_system**(**int b**[],**int w**[],**int k**)**

**{**

int d**,**x**,**y**,**a**=**0**,**m**,**n**=**1**,**i**;**

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

n**\*=**w**[**i**];**

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

**{**

m**=**n**/**w**[**i**];**

d**=**ext\_gcd**(**w**[**i**],**m**,**x**,**y**);**

a**=(**a**+**y**\***m**\***b**[**i**])%**n**;**

**}**

**return** **(**a**+**n**)%**n**;**

**}**

//求解模线性方程组(中国余数定理)

// a[0]x = b[0] (mod w[0])

// a[1]x = b[1] (mod w[1])

// ...……

// a[k-1]x = b[k-1] (mod w[k-1])

//要求w[i]>0,w[i]与w[j]可以不互质

//返回x=b(mod m)

pair**<**LL**,**LL**>** modular\_linear\_system**(**vector**<**LL**>&**a**,**vector**<**LL**>&**b**,**vector**<**LL**>&**w **)**

**{**

LL x**=**0**,**m**=**1**;**

**for** **(** int i**=**0**;** i**<**a**.**sizeof()**;** i**++** **)**

**{**

LL t1**=**a**[**i**]\***m**;**

LL t2**=**b**[**i**]-**a**[**i**]\***x**;**

LL t3**=**\_\_gcd**(**w**[**i**],**t1**);**

**if** **(**t2**%**t3**)** **return** make\_pair**(**0**,-**1**);**

LL t**=**t2**/**t3**\***Inv**(**t1**/**t3**,**w**[**i**]/**t3**)%(**w**[**i**]/**t3**);**

x**+=**m**\***t**;**

m**\*=**w**[**i**]/**t3**;**

x**%=**m**;**

**}**

**return** make\_pair**(((**x**%**m**)+**m**)%**m**,**m**);**

**}**

## 质数表

//用素数表判定素数,先调用initprime

int plist**[**10000**],**pcount**=**0**;**

int prime**(**int n**)**

**{**

int i**;**

**if** **((**n**!=**2**&&!(**n**%**2**))||(**n**!=**3**&&!(**n**%**3**))||(**n**!=**5**&&!(**n**%**5**))||(**n**!=**7**&&!(**n**%**7**)))**

**return** 0**;**

**for** **(**i**=**0**;** plist**[**i**]\***plist**[**i**]<=**n**;** i**++)**

**if** **(!(**n**%**plist**[**i**]))**

**return** 0**;**

**return** n**>**1**;**

**}**

void initprime**()**

**{**

int i**;**

**for** **(**plist**[**pcount**++]=**2**,**i**=**3**;** i**<**50000**;** i**++)**

**if** **(**prime**(**i**))**

plist**[**pcount**++]=**i**;**

**}**

## 质数随机判定(miller\_rabin)

//miller rabin

//判断自然数n是否为素数

//time越高失败概率越低,一般取10到50

#include <stdlib.h>

#ifdef WIN32

**typedef** \_\_int64 i64**;**

#else

**typedef** long long i64**;**

#endif

int modular\_exponent**(**int a**,**int b**,**int n**)** //a^b mod n

**{**

int ret**;**

**for** **(;** b**;** b**>>=**1**,**a**=(**int**)((**i64**)**a**)\***a**%**n**)**

**if** **(**b**&**1**)**

ret**=(**int**)((**i64**)**ret**)\***a**%**n**;**

**return** ret**;**

**}**

// Carmicheal number: 561,41041,825265,321197185

// 这四个数是用来判断Miller\_Rabin程序的正确性的,输入Carmicheal数,程序应该判断为合数

int miller\_rabin**(**int n**,**int time**=**10**)**

**{**

**if** **(**n**==**1**||(**n**!=**2**&&!(**n**%**2**))||(**n**!=**3**&&!(**n**%**3**))||**

**(**n**!=**5**&&!(**n**%**5**))||(**n**!=**7**&&!(**n**%**7**)))**

**return** 0**;**

**while** **(**time**--)**

**if** **(**modular\_exponent**(((**rand**()&**0x7fff**<<**16**)**

**+**rand**()&**0x7fff**+**rand**()&**0x7fff**)%(**n**-**1**)+**1**,**n**-**1**,**n**)!=**1**)**

**return** 0**;**

**return** 1**;**

**}**

## 分解质因数

//prime\_factor()传入n, 返回不同质因数的个数

//f存放质因数，nf存放对应质因数的个数

//先调用initprime()，其中第二个initprime()更快

#include<iostream>

#include<cstdio>

#include<cmath>

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

#define MAXN 2001000

#define PSIZE 100000

int plist**[**PSIZE**],** pcount**=**0**;**

int prime**(**int n**)**

**{**

int i**;**

**if** **((**n**!=**2**&&!(**n**%**2**))||(**n**!=**3**&&!(**n**%**3**))||(**n**!=**5**&&!(**n**%**5**))||(**n**!=**7**&&!(**n**%**7**)))**

**return** 0**;**

**for** **(**i**=**0**;** plist**[**i**]\***plist**[**i**]<=**n**;** **++**i**)**

**if** **(!(**n**%**plist**[**i**]))**

**return** 0**;**

**return** n**>**1**;**

**}**

void initprime**()**

**{**

int i**;**

**for** **(**plist**[**pcount**++]=**2**,**i**=**3**;** i**<**100000**;** **++**i**)**

**if** **(**prime**(**i**))**

plist**[**pcount**++]=**i**;**

**}**

int prime\_factor**(**int n**,** int**\*** f**,** int **\***nf**)**

**{**

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

int n2 **=** sqrt**((**double**)**n**);**

**for(**int i **=** 0**;** n **>** 1 **&&** plist**[**i**]** **<=** n2**;** **++**i**)**

**if** **(**n **%** plist**[**i**]** **==** 0**)**

**{**

**for** **(**nf**[**cnt**]** **=** 0**;** n **%** plist**[**i**]** **==** 0**;** **++**nf**[**cnt**],** n **/=** plist**[**i**]);**

f**[**cnt**++]** **=** plist**[**i**];**

**}**

**if** **(**n **>** 1**)** nf**[**cnt**]** **=** 1**,** f**[**cnt**++]** **=** n**;**

**return** cnt**;**

**}**

/\*\*

//产生MAXN以内的所有素数

//note:2863311530就是10101010101010101010101010101010

//给所有2的倍数赋初值

#include <cmath>

#include <iostream>

using namespace std;

#define MAXN 100000000

unsigned int plist[6000000],pcount;

unsigned int isprime[(MAXN>>5)+1];

#define setbitzero(a) (isprime[(a)>>5]&=(~(1<<((a)&31))))

#define setbitone(a) (isprime[(a)>>5]|=(1<<((a)&31)))

#define ISPRIME(a) (isprime[(a)>>5]&(1<<((a)&31)))

void initprime()

{

int i,j,m;

int t=(MAXN>>5)+1;

for(i=0; i<t; ++i)isprime[i]=2863311530;

plist[0]=2;

setbitone(2);

setbitzero(1);

m=(int)sqrt(MAXN);

for(pcount=1,i=3; i<=m; i+=2)

if(ISPRIME(i))

for(plist[pcount++]=i,j=i<<1; j<=MAXN; j+=i)

setbitzero(j);

if(!(i&1))++i;

for(; i<=MAXN; i+=2)if(ISPRIME(i))plist[pcount++]=i;

}

\*\*/

## 最大公约数（GCD与exGCD）

//欧几里得算法

int gcd**(**int a**,**int b**)**

**{**

**return** b**?**gcd**(**b**,**a**%**b**):**a**;**

**}**

inline int lcm**(**int a**,**int b**)**

**{**

**return** a**/**gcd**(**a**,**b**)\***b**;**

**}**

// 求a的逆元x，ax ≡ 1 (mod n)

//若n为质数，由费马小定理(a^(n-1))%n=1，可得a^(n-2)是a关于n的逆元

int Inv**(**int a**,** int n**)**

**{**

int d**,** x**,** y**;**

d **=** exgcd**(**a**,** n**,** x**,** y**);**

**if(**d **==** 1**)** **return** **(**x**%**n **+** n**)** **%** n**;**

**else** **return** **-**1**;** // no solution

**}**

//扩展欧几里得算法

int exgcd**(**int a**,**int b**,**int **&**x**,**int **&**y**)**

**{**

**if(**b**==**0**)**

**{**

x**=**1**;**

y**=**0**;**

**return** a**;**

**}**

int r**=**exgcd**(**b**,**a**%**b**,**x**,**y**)**，t**=**x**;**

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

y**=**t**-**a**/**b**\***y**;**

**return** r**;**

**}**

## 欧拉函数

//求1..n-1中与n互质的数的个数

int eular**(**int n**)**

**{**

int ret**=**1**,**i**;**

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

**if** **(**n**%**i**==**0**)**

**{**

n**/=**i**,**ret**\*=**i**-**1**;**

**while** **(**n**%**i**==**0**)**

n**/=**i**,**ret**\*=**i**;**

**}**

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

ret**\*=**n**-**1**;**

**return** ret**;**

**}**

## 离散对数解方程A^x=B（mod C）

//扩展BabyStep-GaintStep解方程A^x=B（mod C）（B<C,否则无解）

//主函数extBabyStep无解返回-1,有解返回[0,C)区间内的解

//MAXN>ceil(sqrt(C))

#define MAXN 1000000

**typedef** long long LL**;**

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

struct Node

**{**

int idx**;**

int val**;**

**}** baby**[**MAXN**];**

bool cmp**(**Node n1**,**Node n2**)**

**{**

**return** n1**.**val**!=**n2**.**val**?**n1**.**val**<**n2**.**val**:**n1**.**idx**<**n2**.**idx**;**

**}**

int extend\_gcd**(**int a**,**int b**,**int **&**x**,**int **&**y**)**

**{**

**if(**b**==**0**)**

**{**

x**=**1**;**

y**=**0**;**

**return** a**;**

**}**

int gcd**=**extend\_gcd**(**b**,**a**%**b**,**x**,**y**);**

int t**=**x**;**

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

y**=**t**-**a**/**b**\***y**;**

**return** gcd**;**

**}**

int inval**(**int a**,**int b**,**int n**)**

**{**

int e**,**x**,**y**;**

extend\_gcd**(**a**,**n**,**x**,**y**);**

e**=((**LL**)**x**\***b**)%**n**;**

**return** e**<**0**?**e**+**n**:**e**;**

**}**

int PowMod**(**int a**,**int b**,**int MOD**)**

**{**

LL ret**=**1**%**MOD**,**t**=**a**%**MOD**;**

**while(**b**)**

**{**

**if(**b**&**1**)**

ret**=((**LL**)**ret**\***t**)%**MOD**;**

t**=((**LL**)**t**\***t**)%**MOD**;**

b**>>=**1**;**

**}**

**return** **(**int**)**ret**;**

**}**

int BinSearch**(**int num**,**int m**)**

**{**

int low**=**0**,**high**=**m**-**1**,**mid**;**

**while(**low**<=**high**)**

**{**

mid**=(**low**+**high**)>>**1**;**

**if(**baby**[**mid**].**val**==**num**)**

**return** baby**[**mid**].**idx**;**

**if(**baby**[**mid**].**val**<**num**)**

low**=**mid**+**1**;**

**else**

high**=**mid**-**1**;**

**}**

**return** **-**1**;**

**}**

int extBabyStep**(**int A**,**int B**,**int C**)**

**{**

LL tmp**,**D**=**1**%**C**;**

int temp**;**

**for(**int i**=**0**,**tmp**=**1**%**C**;** i**<**100**;** i**++,**tmp**=((**LL**)**tmp**\***A**)%**C**)**

**if(**tmp**==**B**)**

**return** i**;**

int d**=**0**;**

**while((**temp**=**\_\_gcd**(**A**,**C**))!=**1**)**

**{**

**if(**B**%**temp**)** **return** **-**1**;**

d**++;**

C**/=**temp**;**

B**/=**temp**;**

D**=((**A**/**temp**)\***D**)%**C**;**

**}**

int m**=(**int**)**ceil**(**sqrt**((**double**)**C**));**

**for(**int i**=**0**,**tmp**=**1**%**C**;** i**<=**m**;** i**++,**tmp**=((**LL**)**tmp**\***A**)%**C**)**

**{**

baby**[**i**].**idx**=**i**;**

baby**[**i**].**val**=**tmp**;**

**}**

sort**(**baby**,**baby**+**m**+**1**,**cmp**);**

int cnt**=**1**;**

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

**if(**baby**[**i**].**val**!=**baby**[**cnt**-**1**].**val**)**

baby**[**cnt**++]=**baby**[**i**];**

int am**=**PowMod**(**A**,**m**,**C**);**

**for(**int i**=**0**;** i**<=**m**;** i**++,**D**=((**LL**)(**D**\***am**))%**C**)**

**{**

int tmp**=**inval**(**D**,**B**,**C**);**

**if(**tmp**>=**0**)**

**{**

int pos**=**BinSearch**(**tmp**,**cnt**);**

**if(**pos**!=-**1**)**

**return** i**\***m**+**pos**+**d**;**

**}**

**}**

**return** **-**1**;**

**}**

## 康托展开

//一种哈希函数，参见8数码问题，详见维基百科

//把一个8位的不重复八进制数字串进行康托展开(或计算当前排列在所有由小到大全排列中的顺序)

LL cantor**(**LL S**)**

**{**

LL x**=**0**,**i**,**p**,**k**,**j**;**

bool hash**[**8**]=** **{false};**

**for** **(**i**=**8**;** i**>=**2**;** i**--)**

**{**

k**=**S**>>** 3**\*(**i**-**1**);**

S**-=**k**<<**3**\*(**i**-**1**);**

hash**[**k**]=true;**

p**=**k**;**

**for** **(**j**=**0**;** j**<=**k**-**1**;** j**++)**

**if** **(**hash**[**j**])** p**--;**

x**+=**fac**[**i**-**1**]\***p**;** //fac[i]=i的阶乘

**}**

**return** x**;**

**}**

# 数值计算

## 定积分计算(Romberg)

/\*\* Romberg求定积分

输入：积分区间[a,b]，被积函数f(x,y,z)

输出：积分结果

f(x,y,z)示例：

double f0( double x, double l, double t )

{

return sqrt(1.0+l\*l\*t\*t\*cos(t\*x)\*cos(t\*x));

}

\*\*/

double Romberg **(**double a**,** double b**,** double **(\***f**)(**double x**,** double y**,** double z**),** double eps**,**

double l**,** double t**)**

**{**

#define MAX\_N 1000

int i**,** j**,** temp2**,** min**;**

double h**,** R**[**2**][**MAX\_N**],** temp4**;**

**for** **(**i**=**0**;** i**<**MAX\_N**;** i**++)**

**{**

R**[**0**][**i**]** **=** 0.0**;**

R**[**1**][**i**]** **=** 0.0**;**

**}**

h **=** b**-**a**;**

min **=** **(**int**)(**log**(**h**\***10.0**)/**log**(**2.0**));** //h should be at most 0.1

R**[**0**][**0**]** **=** **((\***f**)(**a**,** l**,** t**)+(\***f**)(**b**,** l**,** t**))\***h**\***0.50**;**

i **=** 1**;**

temp2 **=** 1**;**

**while** **(**i**<**MAX\_N**)**

**{**

i**++;**

R**[**1**][**0**]** **=** 0.0**;**

**for** **(**j**=**1**;** j**<=**temp2**;** j**++)**

R**[**1**][**0**]** **+=** **(\***f**)(**a**+**h**\*((**double**)**j**-**0.50**),** l**,** t**);**

R**[**1**][**0**]** **=** **(**R**[**0**][**0**]** **+** h**\***R**[**1**][**0**])\***0.50**;**

temp4 **=** 4.0**;**

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

**{**

R**[**1**][**j**]** **=** R**[**1**][**j**-**1**]** **+** **(**R**[**1**][**j**-**1**]-**R**[**0**][**j**-**1**])/(**temp4**-**1.0**);**

temp4 **\*=** 4.0**;**

**}**

**if** **((**fabs**(**R**[**1**][**i**-**1**]-**R**[**0**][**i**-**2**])<**eps**)&&(**i**>**min**))**

**return** R**[**1**][**i**-**1**];**

h **\*=** 0.50**;**

temp2 **\*=** 2**;**

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

R**[**0**][**j**]** **=** R**[**1**][**j**];**

**}**

**return** R**[**1**][**MAX\_N**-**1**];**

**}**

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

**{**

#define pi 3.1415926535897932

int n**;**

double R**,** p**,** res**;**

n **=** **(**int**)(**floor**)(**b **\*** t **\*** 0.50 **/** pi**);**

p **=** 2.0 **\*** pi **/** t**;**

res **=** b **-** **(**double**)**n **\*** p**;**

**if** **(**n**)**

R **=** Romberg **(**a**,** p**,** f0**,** eps**/(**double**)**n**,** l**,** t**);**

R **=** R **\*** **(**double**)**n **+** Romberg**(** 0.0**,** res**,** f0**,** eps**,** l**,** t **);**

**return** R**/**100.0**;**

**}**

## 自适应Simpson求积分

**//积分上下限分别为a、b，积分函数为F(x)，积分结果通过函数asr\_main返回**

#include<cstdio>

#include<cmath>

#define EPS 1e-8

double F**(**double x**)**

**{**

**return** log10**(**x**);** //积分函数F(x)

**}**

//三点辛普森公式

double simpson**(**double width**,**double fa**,**double fb**,**double fc**)**

**{**

**return** **(**fb**+**fa**+**4**\***fc**)\***width**/**6**;**

**}**

//自适应simpson公式递归过程

double asr\_rec**(**double a**,**double b**,**double eps**,**double A**)**

**{**

double c**=(**a**+**b**)/**2**;**

double fa**,**fb**,**fc**,**L**,**R**;**

fa**=**F**(**a**);**

fb**=**F**(**b**);**

fc**=**F**(**c**);**

L**=**simpson**(**c**-**a**,**fa**,**fc**,**F**((**c**+**a**)/**2**));**

R**=**simpson**(**b**-**c**,**fc**,**fb**,**F**((**b**+**c**)/**2**));**

**if(**fabs**(**L**+**R**-**A**)<=**15**\***eps**)** **return** L**+**R**+(**L**+**R**-**A**)/**15**;**

**return** asr\_rec**(**a**,**c**,**eps**/**2**,**L**)+**asr\_rec**(**c**,**b**,**eps**/**2**,**R**);**

**}**

//自适应simpson公式主过程，a<b

double asr\_main**(**double a**,**double b**)**

**{**

**return** asr\_rec**(**a**,**b**,**EPS**,**simpson**(**b**-**a**,**F**(**a**),**F**(**b**),**F**((**b**+**a**)/**2**)));**

**}**

**//simpson积分公式另一类写法，在某些情况下更快一些**

double f**(**double x**)**

**{**

**return** PI**\*(**R**\***R**-**x**\***x**);**

**}**

double simpson**(**double a**,**double b**)**

**{**

**return** **(**b**-**a**)/**6.0**\*(**f**(**a**)+**4**\***f**((**a**+**b**)/**2**)+**f**(**b**));**

**}**

double asr\_main**(**double a**,**double b**)**

**{**

double sum**=**simpson**(**a**,**b**);**

double t**=**simpson**(**a**,(**a**+**b**)/**2**)+**simpson**((**a**+**b**)/**2**,**b**);**

**if(**fabs**(**sum**-**t**)<**EPS**)return** sum**;**

**return** simpson**(**a**,(**a**+**b**)/**2**)+**simpson**((**a**+**b**)/**2**,**b**);**

**}**

## 多项式求根(牛顿法)

/\*\* 牛顿法解多项式的根

输入：多项式系数c[]，多项式度数n，求在[a,b]间的根

输出：根

要求保证[a,b]间有根

\*\*/

double fabs**(** double x **)**

**{**

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

**}**

double f**(**int m**,** double c**[],** double x**)**

**{**

int i**;**

double p **=** c**[**m**];**

**for** **(**i**=**m**;** i**>**0**;** i**--)**

p **=** p**\***x **+** c**[**i**-**1**];**

**return** p**;**

**}**

int newton**(**double x0**,** double **\***r**,**

double c**[],** double cp**[],** int n**,**

double a**,** double b**,** double eps**)**

**{**

int MAX\_ITERATION **=** 1000**;**

int i **=** 1**;**

double x1**,** x2**,** fp**,** eps2 **=** eps**/**10.0**;**

x1 **=** x0**;**

**while** **(**i **<** MAX\_ITERATION**)**

**{**

x2 **=** f**(**n**,** c**,** x1**);**

fp **=** f**(**n**-**1**,** cp**,** x1**);**

**if** **((**fabs**(**fp**)<**0.000000001**)** **&&** **(**fabs**(**x2**)>**1.0**))**

**return** 0**;**

x2 **=** x1 **-** x2**/**fp**;**

**if** **(**fabs**(**x1**-**x2**)<**eps2**)**

**{**

**if** **(**x2**<**a **||** x2**>**b**)**

**return** 0**;**

**\***r **=** x2**;**

**return** 1**;**

**}**

x1 **=** x2**;**

i**++;**

**}**

**return** 0**;**

**}**

double Polynomial\_Root**(**double c**[],** int n**,** double a**,** double b**,** double eps**)**

**{**

double **\***cp**;**

int i**;**

double root**;**

cp **=** **(**double **\*)**calloc**(**n**,** **sizeof(**double**));**

**for** **(**i**=**n**-**1**;** i**>=**0**;** i**--)**

**{**

cp**[**i**]** **=** **(**i**+**1**)\***c**[**i**+**1**];**

**}**

**if** **(**a**>**b**)**

**{**

root **=** a**;**

a **=** b**;**

b **=** root**;**

**}**

**if** **((!**newton**(**a**,** **&**root**,** c**,** cp**,** n**,** a**,** b**,** eps**))** **&&**

**(!**newton**(**b**,** **&**root**,** c**,** cp**,** n**,** a**,** b**,** eps**)))**

newton**((**a**+**b**)\***0.5**,** **&**root**,** c**,** cp**,** n**,** a**,** b**,** eps**);**

free**(**cp**);**

**if** **(**fabs**(**root**)<**eps**)**

**return** fabs**(**root**);**

**else**

**return** root**;**

**}**

## 周期性方程(追赶法)

/\*\* 追赶法解周期性方程

周期性方程定义：| a1 b1 c1 ... | | | = x1

| a2 b2 c2 .... | | | = x2

| .......... | \* | X | = ...

| cn-1 ..... an-1 bn-1 | | | = xn-1

| bn cn an | | | = xn

输入：a[],b[],c[],x[]

输出：求解结果X在x[]中

\*\*/

void run**()**

**{**

c**[**0**]** **/=** b**[**0**];**

a**[**0**]** **/=** b**[**0**];**

x**[**0**]** **/=** b**[**0**];**

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

**{**

double temp **=** b**[**i**]** **-** a**[**i**]** **\*** c**[**i **-** 1**];**

c**[**i**]** **/=** temp**;**

x**[**i**]** **=** **(**x**[**i**]** **-** a**[**i**]** **\*** x**[**i **-** 1**])** **/** temp**;**

a**[**i**]** **=** **-**a**[**i**]** **\*** a**[**i **-** 1**]** **/** temp**;**

**}**

a**[**N **-** 2**]** **=** **-**a**[**N **-** 2**]** **-** c**[**N **-** 2**];**

**for** **(**int i **=** N **-** 3**;** i **>=** 0**;** i **--)**

**{**

a**[**i**]** **=** **-**a**[**i**]** **-** c**[**i**]** **\*** a**[**i **+** 1**];**

x**[**i**]** **-=** c**[**i**]** **\*** x**[**i **+** 1**];**

**}**

x**[**N **-** 1**]** **-=** **(**c**[**N **-** 1**]** **\*** x**[**0**]** **+** a**[**N **-** 1**]** **\*** x**[**N **-** 2**]);**

x**[**N **-** 1**]** **/=** **(**c**[**N **-** 1**]** **\*** a**[**0**]** **+** a**[**N **-** 1**]** **\*** a**[**N **-** 2**]** **+** b**[**N **-** 1**]);**

**for** **(**int i **=** N **-** 2**;** i **>=** 0**;** i **--)**

x**[**i**]** **+=** a**[**i**]** **\*** x**[**N **-** 1**];**

**}**

# 组合

## 排列组合生成

//gen\_perm产生字典序排列P(n,m)

//gen\_comb产生字典序组合C(n,m)

//gen\_perm\_swap产生相邻位对换全排列P(n,n)

//产生元素用1..n表示

//dummy为产生后调用的函数,传入a[]和n,a[0]..a[n-1]为一次产生的结果

#define MAXN 100

int count**;**

#include <iostream.h>

void dummy**(**int**\*** a**,**int n**)**

**{**

int i**;**

cout**<<**count**++<<**": "**;**

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

cout**<<**a**[**i**]<<**' '**;**

cout**<<**a**[**n**-**1**]<<**endl**;**

**}**

void \_gen\_perm**(**int**\*** a**,**int n**,**int m**,**int l**,**int**\*** temp**,**int**\*** tag**)**

**{**

int i**;**

**if** **(**l**==**m**)**

dummy**(**temp**,**m**);**

**else**

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

**if** **(!**tag**[**i**])**

**{**

temp**[**l**]=**a**[**i**],**tag**[**i**]=**1**;**

\_gen\_perm**(**a**,**n**,**m**,**l**+**1**,**temp**,**tag**);**

tag**[**i**]=**0**;**

**}**

**}**

void gen\_perm**(**int n**,**int m**)**

**{**

int a**[**MAXN**],**temp**[**MAXN**],**tag**[**MAXN**]=** **{**0**},**i**;**

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

a**[**i**]=**i**+**1**;**

\_gen\_perm**(**a**,**n**,**m**,**0**,**temp**,**tag**);**

**}**

void \_gen\_comb**(**int**\*** a**,**int s**,**int e**,**int m**,**int**&** count**,**int**\*** temp**)**

**{**

int i**;**

**if** **(!**m**)**

dummy**(**temp**,**count**);**

**else**

**for** **(**i**=**s**;** i**<=**e**-**m**+**1**;** i**++)**

**{**

temp**[**count**++]=**a**[**i**];**

\_gen\_comb**(**a**,**i**+**1**,**e**,**m**-**1**,**count**,**temp**);**

count**--;**

**}**

**}**

void gen\_comb**(**int n**,**int m**)**

**{**

int a**[**MAXN**],**temp**[**MAXN**],**count**=**0**,**i**;**

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

a**[**i**]=**i**+**1**;**

\_gen\_comb**(**a**,**0**,**n**-**1**,**m**,**count**,**temp**);**

**}**

void \_gen\_perm\_swap**(**int**\*** a**,**int n**,**int l**,**int**\*** pos**,**int**\*** dir**)**

**{**

int i**,**p1**,**p2**,**t**;**

**if** **(**l**==**n**)**

dummy**(**a**,**n**);**

**else**

**{**

\_gen\_perm\_swap**(**a**,**n**,**l**+**1**,**pos**,**dir**);**

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

**{**

p2**=(**p1**=**pos**[**l**])+**dir**[**l**];**

t**=**a**[**p1**],**a**[**p1**]=**a**[**p2**],**a**[**p2**]=**t**;**

pos**[**a**[**p1**]-**1**]=**p1**,**pos**[**a**[**p2**]-**1**]=**p2**;**

\_gen\_perm\_swap**(**a**,**n**,**l**+**1**,**pos**,**dir**);**

**}**

dir**[**l**]=-**dir**[**l**];**

**}**

**}**

void gen\_perm\_swap**(**int n**)**

**{**

int a**[**MAXN**],**pos**[**MAXN**],**dir**[**MAXN**],**i**;**

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

a**[**i**]=**i**+**1**,**pos**[**i**]=**i**,**dir**[**i**]=-**1**;**

\_gen\_perm\_swap**(**a**,**n**,**0**,**pos**,**dir**);**

**}**

## 生成gray码

//Gray(格雷)码是一种模数转换码字，每相邻码字之间只有一位不一样

//每次调用gray取得下一个码

//000...000是第一个码,100...000是最后一个码

void gray**(**int n**,**int **\***code**)**

**{**

int t**=**0**,**i**;**

**for** **(**i**=**0**;** i**<**n**;** t**+=**code**[**i**++]);**

**if** **(**t**&**1**)**

**for** **(**n**--;** **!**code**[**n**];** n**--);**

code**[**n**-**1**]=**1**-**code**[**n**-**1**];**

**}**

## 置换(polya定理)

//perm[0..n-1]为0..n-1的一个置换(排列)

//返回置换最小周期,num返回循环节个数

#define MAXN 1000

int gcd**(**int a**,**int b**)**

**{**

**return** b**?**gcd**(**b**,**a**%**b**):**a**;**

**}**

int polya**(**int**\*** perm**,**int n**,**int**&** num**)**

**{**

int i**,**j**,**p**,**v**[**MAXN**]=** **{**0**},**ret**=**1**;**

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

**if** **(!**v**[**i**])**

**{**

**for** **(**num**++,**j**=**0**,**p**=**i**;** **!**v**[**p**=**perm**[**p**]];** j**++)**

v**[**p**]=**1**;**

ret**\*=**j**/**gcd**(**ret**,**j**);**

**}**

**return** ret**;**

**}**

## 整数划分（五边形定理与其他）

..

**//将n划分成多个整数的和的形式，求划分的种数（五边形定理）**

//记录在dp[n]中，有取模

#define MOD 1000000007LL

#define MAXN 100000+5

LL dp**[**MAXN**];**

void full\_partition **()**

**{**

dp**[**0**]=**dp**[**1**]=**1**,**dp**[**2**]=**2**,**dp**[**3**]=**3**;**

**for(**int i**=**4**;** i**<=**MAXN**;** i**++)**

**{**

dp**[**i**]=**0**;**

int flag**=**1**;**

**for(**int j**=**1**;;** **++**j**)**

**{**

int a**=(**j**\***j**\***3**+**j**)/**2**,**b**=(**j**\***j**\***3**-**j**)/**2**;**

**if(**b**>**i**&&**a**>**i**)** **break;**

**if(**a**<=**i**)** dp**[**i**]=(**dp**[**i**]+**dp**[**i**-**a**]\***flag**+**Mod**)%**Mod**;**

**if(**b**<=**i**)** dp**[**i**]=(**dp**[**i**]+**dp**[**i**-**b**]\***flag**+**Mod**)%**Mod**;**

flag**=**flag**\*(-**1**);**

**}**

**}**

**}**

**//将n整数分拆，要求拆出来整数中任意整数出现次数小于k次**

**//等价于分拆出来的最大整数不超过k**

//dp[n]表示无限制分拆，在full\_partition()中初始化

int get**(** int n**,**int k **)**

**{**

LL ret**=**dp**[**n**];**

int flag**=-**1

**for** **(** int i**=**1**;** n**>=**k**\*(** 3**\***i**\***i**-**i **)/**2**;** i**++)**

**{**

ret**+=**dp**[**n**-**k**\*(** 3**\***i**\***i**-**i **)/**2**]\***flag**;**

**if** **(** n**>=**k**\*(** 3**\***i**\***i**+**i **)/**2 **)**

ret**+=**dp**[**n**-**k**\*(** 3**\***i**\***i**+**i **)/**2**]\***flag**;**

ret**=**ret**%**MOD**+**MOD**;**

flag**\*=-**1**;**

**}**

**return** ret**%**MOD**;**

**}**

**//将整数n分成k份, 且每份不能为空, 任意两种分法不能相同**

//(1)不考虑顺序

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

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

**for(**int j**=**k**;** j**>=**1 **;** j**--)**

dp**[**i**][**j**]** **+=** dp**[**i**-**p**][**j**-**1**];**

cout**<<** dp**[**n**][**k**]** **<<**endl**;**

//(2)考虑顺序

dp**[**i**][**j**]** **=** dp**[**i**-**k**][**j**-**1**] ;** **(**k**=**1…...i**)**

//(3)若分解出来的每个数均有一个上限m

dp**[**i**][**j**]** **=** dp**[**i**-**k**][**j**-**1**] ; (**k**=**1……m**)**

## 字典序全排列与序号的转换

int perm2num**(**int n**,**int **\***p**)**

**{**

int i**,**j**,**ret**=**0**,**k**=**1**;**

**for** **(**i**=**n**-**2**;** i**>=**0**;** k**\*=**n**-(**i**--))**

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

**if** **(**p**[**j**]<**p**[**i**])**

ret**+=**k**;**

**return** ret**;**

**}**

void num2perm**(**int n**,**int **\***p**,**int t**)**

**{**

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

**for** **(**i**=**n**-**1**;** i**>=**0**;** i**--)**

p**[**i**]=**t**%(**n**-**i**),**t**/=**n**-**i**;**

**for** **(**i**=**n**-**1**;** i**;** i**--)**

**for** **(**j**=**i**-**1**;** j**>=**0**;** j**--)**

**if** **(**p**[**j**]<=**p**[**i**])**

p**[**i**]++;**

**}**

## 字典序组合与序号的转换

//comb为组合数C(n,m),必要时换成大数,注意处理C(n,m)=0|n<m

int comb**(**int n**,**int m**)**

**{**

int ret**=**1**,**i**;**

m**=**m**<(**n**-**m**)?**m**:(**n**-**m**);**

**for** **(**i**=**n**-**m**+**1**;** i**<=**n**;** ret**\*=(**i**++));**

**for** **(**i**=**1**;** i**<=**m**;** ret**/=(**i**++));**

**return** m**<**0**?**0**:**ret**;**

**}**

int comb2num**(**int n**,**int m**,**int **\***c**)**

**{**

int ret**=**comb**(**n**,**m**),**i**;**

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

ret**-=**comb**(**n**-**c**[**i**],**m**-**i**);**

**return** ret**;**

**}**

void num2comb**(**int n**,**int m**,**int**\*** c**,**int t**)**

**{**

int i**,**j**=**1**,**k**;**

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

**for** **(;** t**>(**k**=**comb**(**n**-**j**,**m**-**i**-**1**));** t**-=**k**,**j**++);**

**}**

## 组合公式

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

3. 错排函数D(n) = n!(1 - 1/1! + 1/2! - 1/3! + ... + (-1)^n/n!)

= (n-1)(D(n-2) - D(n-1))

= floor(n!/*e*+0.5)

其中*e*是自然常数2.718281828459

4. 斐波那契数列（0，1，1，2，3，5，8，13，21，…….）

F[0]=0， F[1]=1， F[i]=F[i-1]+F[i-2]



5. Lucas数列（2，1，3，4，7，11，18，29……）

L[0]=2， L[1]=1， L[i]=L[i-1]+L[i-2]



6. Catalan数（1，2，5，14，42，132，429……）



//应用1：将 n+2边形沿弦切割成n个三角形的不同切割数

//应用2：n+1个数相乘，给每两个元素加上括号的不同方法数

//应用3：n个节点的不同形状的二叉树数

//应用4：从n\*n方格的左上角移动到右下角非升路径数

7. 第一类Stirling数

//s(n, m)将n个物体排成m个非空循环排列的方法数

//或是n个有标号的球摆成m个圆环，要求每个圆环非空，并且每个环的的不同排列顺序算一种



特例：



8. 第二类Stirling数

//S(n, m)表示含n个元素的集合划分为m个集合的情况数

//或是n个有标号的球放到m个无标号的盒子中, 要求无一为空, 其不同的方案数

//将n个有标号的球放到m个有标号的盒子中，要求无一为空,其不同的方案数为m! S(n, m)



特例：



9. Stirling阶乘逼近公式



10. 数的位数的计算

int ans**=(**int**)(**log**(**x**)+**1.0**)**

11．调和级数的部分和



其中γ=0.5772 1566 4901 5328 6060 6512 0901称作欧拉常数

## 求和公式（k = 1……n）

1. sum( k ) = n(n+1)/2

2. sum( 2k-1 ) = n^2

3. sum( k^2 ) = n(n+1)(2n+1)/6

4. sum( (2k-1)^2 ) = n(4n^2-1)/3

5. sum( k^3 ) = (n(n+1)/2)^2

6. sum( (2k-1)^3 ) = n^2(2n^2-1)

7. sum( k^4 ) = n(n+1)(2n+1)(3n^2+3n-1)/30

8. sum( k^5 ) = n^2(n+1)^2(2n^2+2n-1)/12

9. sum( k(k+1) ) = n(n+1)(n+2)/3

10. sum( k(k+1)(k+2) ) = n(n+1)(n+2)(n+3)/4

12. sum( k(k+1)(k+2)(k+3) ) = n(n+1)(n+2)(n+3)(n+4)/5

# 大数高精度计算

**//头文件与库函数**

#include<iostream>

#include<string>

#include<iomanip>

#include<algorithm>

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

#define MAXN 9999

#define MAXSIZE 10

#define DLEN 4 //每四位进行处理，万进制运算

**//大数BigNum类**

class BigNum

**{**

private**:**

int a**[**500**];** //可以控制大数的位数

int len**;** //大数长度

public**:**

BigNum**()**

**{**

len **=** 1**;** //构造函数

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

**}**

BigNum**(**const int**);** //将一个int类型的变量转化为大数

BigNum**(**const char**\*);** //将一个字符串类型的变量转化为大数

BigNum**(**const BigNum **&);** //拷贝构造函数

BigNum **&operator=(**const BigNum **&);**

//重载赋值运算符，大数之间进行赋值运算

friend istream**&** **operator>>(**istream**&,** BigNum**&);**

//重载输入运算符

friend ostream**&** **operator<<(**ostream**&,** BigNum**&);**

//重载输出运算符

BigNum **operator+(**const BigNum **&)** const**;**

//重载加法运算符，两个大数之间的相加运算

BigNum **operator-(**const BigNum **&)** const**;**

//重载减法运算符，两个大数之间的相减运算

BigNum **operator\*(**const BigNum **&)** const**;**

//重载乘法运算符，两个大数之间的相乘运算

BigNum **operator/(**const int **&)** const**;**

//重载除法运算符，大数对一个整数进行相除运算

BigNum **operator^(**const int **&)** const**;** //大数的n次方运算

int **operator%(**const int **&)** const**;**

//大数对一个int类型的变量进行取模运算

bool **operator>(**const BigNum **&** T**)**const**;**

//大数和另一个大数的大小比较

bool **operator>(**const int **&** t**)**const**;**

//大数和一个int类型的变量的大小比较

void print**();** //输出大数

**};**

**//其他类型变量转换为大数**

BigNum**::**BigNum**(**const int b**)** //将一个int类型的变量转化为大数

**{**

int c**,**d **=** b**;**

len **=** 0**;**

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

**while(**d **>** MAXN**)**

**{**

c **=** d **-** **(**d **/** **(**MAXN **+** 1**))** **\*** **(**MAXN **+** 1**);**

d **=** d **/** **(**MAXN **+** 1**);**

a**[**len**++]** **=** c**;**

**}**

a**[**len**++]** **=** d**;**

**}**

BigNum**::**BigNum**(**const char**\***s**)** //将一个字符串类型的变量转化为大数

**{**

int t**,**k**,**index**,**l**,**i**;**

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

l**=**strlen**(**s**);**

len**=**l**/**DLEN**;**

**if(**l**%**DLEN**)**

len**++;**

index**=**0**;**

**for(**i**=**l**-**1**;** i**>=**0**;** i**-=**DLEN**)**

**{**

t**=**0**;**

k**=**i**-**DLEN**+**1**;**

**if(**k**<**0**)**

k**=**0**;**

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

t**=**t**\***10**+**s**[**j**]-**'0'**;**

a**[**index**++]=**t**;**

**}**

**}**

BigNum**::**BigNum**(**const BigNum **&** T**)** **:** len**(**T**.**len**)** //拷贝构造函数

**{**

int i**;**

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

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

a**[**i**]** **=** T**.**a**[**i**];**

**}**

**//运算符的重载**

BigNum **&** BigNum**::operator=(**const BigNum **&** n**)** //重载赋值运算符，大数之间进行赋值运算

**{**

int i**;**

len **=** n**.**len**;**

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

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

a**[**i**]** **=** n**.**a**[**i**];**

**return** **\*this;**

**}**

istream**&** **operator>>(**istream **&** in**,** BigNum **&** b**)** //重载输入运算符

**{**

char ch**[**MAXSIZE**\***4**];**

int i **=** **-**1**;**

in**>>**ch**;**

int l**=**strlen**(**ch**);**

int count**=**0**,**sum**=**0**;**

**for(**i**=**l**-**1**;** i**>=**0**;)**

**{**

sum **=** 0**;**

int t**=**1**;**

**for(**int j**=**0**;** j**<**4**&&**i**>=**0**;** j**++,**i**--,**t**\*=**10**)**

**{**

sum**+=(**ch**[**i**]-**'0'**)\***t**;**

**}**

b**.**a**[**count**]=**sum**;**

count**++;**

**}**

b**.**len **=**count**++;**

**return** in**;**

**}**

ostream**&** **operator<<(**ostream**&** out**,** BigNum**&** b**)** //重载输出运算符

**{**

int i**;**

cout **<<** b**.**a**[**b**.**len **-** 1**];**

**for(**i **=** b**.**len **-** 2 **;** i **>=** 0 **;** i**--)**

**{**

cout**.**width**(**DLEN**);**

cout**.**fill**(**'0'**);**

cout **<<** b**.**a**[**i**];**

**}**

**return** out**;**

**}**

**//大数的四则运算**

BigNum BigNum**::operator+(**const BigNum **&** T**)** const

//两个大数之间的相加运算

**{**

BigNum t**(\*this);**

int i**,**big**;** //位数

big **=** T**.**len **>** len **?** T**.**len **:** len**;**

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

**{**

t**.**a**[**i**]** **+=**T**.**a**[**i**];**

**if(**t**.**a**[**i**]** **>** MAXN**)**

**{**

t**.**a**[**i **+** 1**]++;**

t**.**a**[**i**]** **-=**MAXN**+**1**;**

**}**

**}**

**if(**t**.**a**[**big**]** **!=** 0**)**

t**.**len **=** big **+** 1**;**

**else**

t**.**len **=** big**;**

**return** t**;**

**}**

BigNum BigNum**::operator-(**const BigNum **&** T**)** const

//两个大数之间的相减运算

**{**

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

bool flag**;**

BigNum t1**,**t2**;**

**if(\*this>**T**)**

**{**

t1**=\*this;**

t2**=**T**;**

flag**=**0**;**

**}**

**else**

**{**

t1**=**T**;**

t2**=\*this;**

flag**=**1**;**

**}**

big**=**t1**.**len**;**

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

**{**

**if(**t1**.**a**[**i**]** **<** t2**.**a**[**i**])**

**{**

j **=** i **+** 1**;**

**while(**t1**.**a**[**j**]** **==** 0**)**

j**++;**

t1**.**a**[**j**--]--;**

**while(**j **>** i**)**

t1**.**a**[**j**--]** **+=** MAXN**;**

t1**.**a**[**i**]** **+=** MAXN **+** 1 **-** t2**.**a**[**i**];**

**}**

**else**

t1**.**a**[**i**]** **-=** t2**.**a**[**i**];**

**}**

t1**.**len **=** big**;**

**while(**t1**.**a**[**len **-** 1**]** **==** 0 **&&** t1**.**len **>** 1**)**

**{**

t1**.**len**--;**

big**--;**

**}**

**if(**flag**)**

t1**.**a**[**big**-**1**]=**0**-**t1**.**a**[**big**-**1**];**

**return** t1**;**

**}**

BigNum BigNum**::operator\*(**const BigNum **&** T**)** const

//两个大数之间的相乘运算

**{**

BigNum ret**;**

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

int temp**,**temp1**;**

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

**{**

up **=** 0**;**

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

**{**

temp **=** a**[**i**]** **\*** T**.**a**[**j**]** **+** ret**.**a**[**i **+** j**]** **+** up**;**

**if(**temp **>** MAXN**)**

**{**

temp1 **=** temp **-** temp **/** **(**MAXN **+** 1**)** **\*** **(**MAXN **+** 1**);**

up **=** temp **/** **(**MAXN **+** 1**);**

ret**.**a**[**i **+** j**]** **=** temp1**;**

**}**

**else**

**{**

up **=** 0**;**

ret**.**a**[**i **+** j**]** **=** temp**;**

**}**

**}**

**if(**up **!=** 0**)**

ret**.**a**[**i **+** j**]** **=** up**;**

**}**

ret**.**len **=** i **+** j**;**

**while(**ret**.**a**[**ret**.**len **-** 1**]** **==** 0 **&&** ret**.**len **>** 1**)**

ret**.**len**--;**

**return** ret**;**

**}**

BigNum BigNum**::operator/(**const int **&** b**)** const

//大数对一个整数进行相除运算

**{**

BigNum ret**;**

int i**,**down **=** 0**;**

**for(**i **=** len **-** 1 **;** i **>=** 0 **;** i**--)**

**{**

ret**.**a**[**i**]** **=** **(**a**[**i**]** **+** down **\*** **(**MAXN **+** 1**))** **/** b**;**

down **=** a**[**i**]** **+** down **\*** **(**MAXN **+** 1**)** **-** ret**.**a**[**i**]** **\*** b**;**

**}**

ret**.**len **=** len**;**

**while(**ret**.**a**[**ret**.**len **-** 1**]** **==** 0 **&&** ret**.**len **>** 1**)**

ret**.**len**--;**

**return** ret**;**

**}**

**//取模，乘方，开方与比较大小**

int BigNum**::operator** **%(**const int **&** b**)** const //大数对一个int类型的变量进行取模运算

**{**

int i**,**d**=**0**;**

**for** **(**i **=** len**-**1**;** i**>=**0**;** i**--)**

**{**

d **=** **((**d **\*** **(**MAXN**+**1**))%** b **+** a**[**i**])%** b**;**

**}**

**return** d**;**

**}**

BigNum BigNum**::operator^(**const int **&** n**)** const //大数的n次方运算

**{**

BigNum t**,**ret**(**1**);**

int i**;**

**if(**n**<**0**)**

exit**(-**1**);**

**if(**n**==**0**)**

**return** 1**;**

**if(**n**==**1**)**

**return** **\*this;**

int m**=**n**;**

**while(**m**>**1**)**

**{**

t**=\*this;**

**for(** i**=**1**;** i**<<**1**<=**m**;** i**<<=**1**)**

**{**

t**=**t**\***t**;**

**}**

m**-=**i**;**

ret**=**ret**\***t**;**

**if(**m**==**1**)**

ret**=**ret**\*(\*this);**

**}**

**return** ret**;**

**}**

bool BigNum**::operator>(**const BigNum **&** T**)** const //大数和另一个大数的大小比较

**{**

int ln**;**

**if(**len **>** T**.**len**)**

**return** **true;**

**else** **if(**len **==** T**.**len**)**

**{**

ln **=** len **-** 1**;**

**while(**a**[**ln**]** **==** T**.**a**[**ln**]** **&&** ln **>=** 0**)**

ln**--;**

**if(**ln **>=** 0 **&&** a**[**ln**]** **>** T**.**a**[**ln**])**

**return** **true;**

**else**

**return** **false;**

**}**

**else**

**return** **false;**

**}**

bool BigNum**::operator** **>(**const int **&** t**)** const //大数和一个int类型的变量的大小比较

**{**

BigNum b**(**t**);**

**return** **\*this>**b**;**

**}**

void BigNum**::**print**()** //输出大数

**{**

int i**;**

cout **<<** a**[**len **-** 1**];**

**for(**i **=** len **-** 2 **;** i **>=** 0 **;** i**--)**

**{**

cout**.**width**(**DLEN**);**

cout**.**fill**(**'0'**);**

cout **<<** a**[**i**];**

**}**

cout **<<** endl**;**

**}**

int main**()**

**{**

int i**,**n**;**

BigNum x**[**101**];** //定义大数的对象数组

x**[**0**]=**1**;**

**for(**i**=**1**;** i**<**101**;** i**++)**

x**[**i**]=**x**[**i**-**1**]\*(**4**\***i**-**2**)/(**i**+**1**);**

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

**{**

x**[**n**].**print**();**

**}**

**}**

# Java相关

## BigInteger输入输出外挂

//验题：hdu5047

//注意Java在用Scanner作输入时效率会降低，如果出题者卡Scanner，可以用BufferedReader

**import** java**.**math**.**BigInteger**;**

**import** java**.**util**.\*;**

**import** java**.**io**.\*;**

class InputReader

**{**

private InputStream stream**;**

private byte**[]** buf **=** **new** byte**[**1000**];**

private int curChar**;**

private int numChars**;**

public InputReader**(**InputStream stream**)**

**{**

**this.**stream **=** stream**;**

**}**

private int read**()**

**{**

**if** **(**numChars **==** **-**1**)**

**throw** **new** UnknownError**();**

**if** **(**curChar **>=** numChars**)**

**{**

curChar **=** 0**;**

**try**

**{**

numChars **=** stream**.**read**(**buf**);**

**}**

**catch** **(**IOException e**)**

**{**

**throw** **new** UnknownError**();**

**}**

**if** **(**numChars **<=** 0**)**

**return** **-**1**;**

**}**

**return** buf**[**curChar**++];**

**}**

public int readInt**()**

**{**

int c **=** read**();**

**while** **(**isSpaceChar**(**c**))**

c **=** read**();**

int sgn **=** 1**;**

**if** **(**c **==** '-'**)**

**{**

sgn **=** **-**1**;**

c **=** read**();**

**}**

int res **=** 0**;**

**do**

**{**

**if** **(**c**<**'0'**||**c**>** '9'**)** **throw** **new** InputMismatchException**();**

res **\*=** 10**;**

res **+=** c **-** '0'**;**

c **=** read**();**

**}**

**while** **(!**isSpaceChar**(**c**));**

**return** res **\*** sgn**;**

**}**

public int**[]** readIntArray**(**int length**)**

**{**

int**[]** res **=** **new** int**[**length**];**

**for** **(**int i **=** 0**;** i **<** length**;** i **++)** res**[**i**]** **=** readInt**();**

**return** res**;**

**}**

public int**[][]** readIntArray**(**int n**,** int m**)**

**{**

int**[][]** res **=** **new** int**[**n**][**m**];**

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

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

res**[**i**][**j**]** **=** readInt**();**

**return** res**;**

**}**

public long readLong**()**

**{**

int c **=** read**();**

**while** **(**isSpaceChar**(**c**))**

c **=** read**();**

int sgn **=** 1**;**

**if** **(**c **==** '-'**)**

**{**

sgn **=** **-**1**;**

c **=** read**();**

**}**

long res **=** 0**;**

**do**

**{**

**if** **(**c**<**'0'**||**c**>**'9'**)** **throw** **new** InputMismatchException**();**

res **\*=** 10**;**

res **+=** c **-** '0'**;**

c **=** read**();**

**}**

**while** **(!**isSpaceChar**(**c**));**

**return** res **\*** sgn**;**

**}**

public long**[]** readLongArray**(**int length**)**

**{**

long**[]** res **=** **new** long**[**length**];**

**for** **(**int i **=** 0**;** i **<** length**;** i **++)** res**[**i**]** **=** readLong**();**

**return** res**;**

**}**

public long**[][]** readLongArray**(**int n**,** int m**)**

**{**

long**[][]** res **=** **new** long**[**n**][**m**];**

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

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

res**[**i**][**j**]** **=** readLong**();**

**return** res**;**

**}**

public String readString**()**

**{**

int c **=** read**();**

**while** **(**isSpaceChar**(**c**))**

c **=** read**();**

StringBuffer res **=** **new** StringBuffer**();**

**do**

**{**

res**.**appendCodePoint**(**c**);**

c **=** read**();**

**}**

**while** **(!**isSpaceChar**(**c**));**

**return** res**.**toString**();**

**}**

public String**[]** readStringArray**(**int length**)**

**{**

String**[]** res **=** **new** String**[**length**];**

**for** **(**int i **=** 0**;** i **<** length**;** i **++)** res**[**i**]** **=** readString**();**

**return** res**;**

**}**

public String next**()**

**{**

**return** readString**();**

**}**

private String readLine0**()**

**{**

StringBuffer buf **=** **new** StringBuffer**();**

int c **=** read**();**

**while** **(**c **!=** '\n' **&&** c **!=** **-**1**)**

**{**

buf**.**appendCodePoint**(**c**);**

c **=** read**();**

**}**

**return** buf**.**toString**();**

**}**

public String readLine**()**

**{**

String s **=** readLine0**();**

**while** **(**s**.**trim**().**length**()** **==** 0**)**

s **=** readLine0**();**

**return** s**;**

**}**

public String readLine**(**boolean ignoreEmptyLines**)**

**{**

**if** **(**ignoreEmptyLines**)**

**return** readLine**();**

**else**

**return** readLine0**();**

**}**

public BigInteger readBigInteger**()**

**{**

**try**

**{**

**return** **new** BigInteger**(**readString**());**

**}**

**catch** **(**NumberFormatException e**)**

**{**

**throw** **new** InputMismatchException**();**

**}**

**}**

public char readCharacter**()**

**{**

int c **=** read**();**

**while** **(**isSpaceChar**(**c**))**

c **=** read**();**

**return** **(**char**)** c**;**

**}**

public char**[]** readCharArray**(**int length**)**

**{**

char**[]** res **=** **new** char**[**length**];**

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

res**[**i**]** **=** readCharacter**();**

**return** res**;**

**}**

public char**[][]** readCharArray**(**int n**,** int m**)**

**{**

char**[][]** res **=** **new** char**[**n**][**m**];**

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

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

res**[**i**][**j**]** **=** readCharacter**();**

**return** res**;**

**}**

public double readDouble**()**

**{**

int c **=** read**();**

**while** **(**isSpaceChar**(**c**))**

c **=** read**();**

int sgn **=** 1**;**

**if** **(**c **==** '-'**)**

**{**

sgn **=** **-**1**;**

c **=** read**();**

**}**

double res **=** 0**;**

**while** **(!**isSpaceChar**(**c**)** **&&** c **!=** '.'**)**

**{**

**if** **(**c **<** '0' **||** c **>** '9'**)**

**throw** **new** InputMismatchException**();**

res **\*=** 10**;**

res **+=** c **-** '0'**;**

c **=** read**();**

**}**

**if** **(**c **==** '.'**)**

**{**

c **=** read**();**

double m **=** 1**;**

**while** **(!**isSpaceChar**(**c**))**

**{**

**if** **(**c **<** '0' **||** c **>** '9'**)**

**throw** **new** InputMismatchException**();**

m **/=** 10**;**

res **+=** **(**c **-** '0'**)** **\*** m**;**

c **=** read**();**

**}**

**}**

**return** res **\*** sgn**;**

**}**

public double**[]** readDoubleArray**(**int length**)**

**{**

double**[]** res **=** **new** double**[**length**];**

**for** **(**int i **=** 0**;** i **<** length**;** i **++)** res**[**i**]** **=** readDouble**();**

**return** res**;**

**}**

public double**[][]** readDoubleArray**(**int n**,** int m**)**

**{**

double**[][]** res **=** **new** double**[**n**][**m**];**

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

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

res**[**i**][**j**]** **=** readDouble**();**

**return** res**;**

**}**

private boolean isSpaceChar**(**int c**)**

**{**

**return** c **==** ' ' **||** c **==** '\n' **||** c **==** '\r' **||** c **==** '\t' **||** c **==** **-**1**;**

**}**

**}**

class OutputWriter

**{**

private final PrintWriter writer**;**

public OutputWriter**(**OutputStream outputStream**)**

**{**

writer **=** **new** PrintWriter**(new** BufferedWriter**(new** OutputStreamWriter**(**outputStream**)));**

**}**

public OutputWriter**(**Writer writer**)**

**{**

**this.**writer **=** **new** PrintWriter**(**writer**);**

**}**

public void print**(**Object**...**objects**)**

**{**

**for** **(**int i **=** 0**;** i **<** objects**.**length**;** i**++)**

**{**

**if** **(**i **!=** 0**)**

writer**.**print**(**' '**);**

writer**.**print**(**objects**[**i**]);**

**}**

**}**

public void printDouble**(**double x**,** int precision**)**

**{**

writer**.**printf**(**"%." **+** precision **+** "f"**,** x**);**

**}**

public void printLineDouble**(**double x**,** int precision**)**

**{**

printDouble**(**x**,** precision**);**

printLine**();**

**}**

public void printLine**(**Object**...**objects**)**

**{**

print**(**objects**);**

writer**.**println**();**

**}**

public void printIntArray**(**int**[]** data**)**

**{**

**for** **(**int i **=** 0**;** i **<** data**.**length**;** i **++)**

**if** **(**i **<** data**.**length **-** 1**)**

**{**

print**(**data**[**i**]** **+** " "**);**

**}**

**else**

**{**

print**(**data**[**i**]);**

**}**

**}**

public void printLongArray**(**long**[]** data**)**

**{**

**for** **(**int i **=** 0**;** i **<** data**.**length**;** i **++)**

**if** **(**i **<** data**.**length **-** 1**)**

**{**

print**(**data**[**i**]** **+** " "**);**

**}**

**else**

**{**

print**(**data**[**i**]);**

**}**

**}**

public void close**()**

**{**

writer**.**close**();**

**}**

**}**

public class Main

**{**

public static void main**(**String**[]** args**)**

**{**

InputReader s **=** **new** InputReader**(**System**.**in**);**

OutputWriter out **=** **new** OutputWriter**(**System**.**out**);**

int T **=** s**.**readInt**();**

**for** **(**int caseNo **=** 0**;** caseNo **<** T**;** caseNo **++)**

**{**

BigInteger n **=** s**.**readBigInteger**();**

BigInteger ans **=** BigInteger**.**valueOf**(**8**).**multiply**(**n**);**

ans**=** ans**.**multiply**(**n**.**subtract**(**BigInteger**.**ONE**));**

ans **=** ans**.**add**(**n**);**

ans **=** ans**.**add**(**BigInteger**.**ONE**);**

out**.**printLine**(**"Case #" **+** **(**caseNo **+** 1**)** **+** ": " **+** ans**);**

**}**

out**.**close**();**

**}**

**}**

# 常用函数与其他

## 常用函数

#include<cstdlib>

int getchar**(**void**)** //读取单个字符，常用来去掉无用字符

char **\***gets**(**char **\***str**)** //读取一行字符串，以回车结束

#include<cstdlib>

void **\***malloc**(**size\_t size**)** //动态分配大小为size的内存

#include<cmath>

double asin**(**double arg**)** //反正弦，返回值 -PI/2 ~ PI/2

double exp**(**double x**)** //求e^x

double log**(**double x**)** //求ln(x)

#include<cstring>

int sprintf**(**char **\***str**,** const char **\***format**,** **...)**

//将某一个数字格式化为字符串str

//例如sprintf(str,"%d”,12345)

int sscanf**(**const char **\***str**,** const char **\***format**,** **...)**

//从字符串str中提取数字和其他类型的数据

//例如sscanf(str,"%d %s",num,s)

int strcmp**(**const char **\***str1**,** const char **\***str2**)**

//比较两个字符串的字典序大小，返回0表示str1=str2，返回<0表示str1<str2,返回>0表示str1>str2

#include<algorithm>

Iterator nth\_element**(**start**,** start**+**n**,** end**)**

//使第n大元素处于第n位置（从0开始,其位置是下标为n的元素），并且比这个元素小的元素都排在这个元素之前，比这个元素大的元素都排在这个元素之后，但不能保证他们是有序的。

bool next\_permutation**(**start**,** end**)**

//输入升序排列的序列的头指针和尾指针，返回该序列按字典序的下一个排列

bool prev\_permutation**(**start**,** end**)**

//输入升序排列的序列的头指针和尾指针，返回该序列按字典序的下一个排列

bool binary\_search**(**first**,** last**,** const **&**val**,** cmp**)**

//按照cmp的规则在区间[first, last)二分查找是否存在val

int lower\_bound**(**start**,** end**,** int key**)**

//给定一个升序数组，在[start,end)区间内进行二分查找，返回>=key的第一个元素的位置；如果所有元素都小于key，返回end的位置

int upper\_bound**(**start**,** end**,** int key**)**

//给定一个升序数组，在[start,end)区间内进行二分查找，返回>key的第一个元素的位置；如果所有元素都小于key，返回end的位置

void merge**(**first1**,** last1**,** first2**,** last2**,** result**)**

//将两个升序数组按照升序合并存入result中，两个数组范围分别为[first1, last1]和[first2, last2]

bool includes**(**first1**,** end1**,** first2**,** end2**)**

//对于有序序列[first1, last1)和[first2, last2),判断第二个序列是否是第一个序列的子集

void set\_union**(**first1**,** last1**,** first2**,** last2**,** result**.**begin**())**

//求两个升序集合的集合并，存在result中

void set\_intersection**(**first1**,** last1**,** first2**,** last2**,** result**.**begin**())**

//求两个升序集合的集合差

iterator max\_element**(**first**,** last**,** cmp**)**

//按照cmp的规则找出[first, last)区间内的的最小值，返回最小值所在的指针

## 格式化输入

%c 读入一个字符

%d 读入十进制整数

%i 读入十进制，八进制，十六进制整数

%o 读入八进制整数

%x,%X 读入十六进制整数

%c 读入一个字符

%s 读入一个字符串，遇空格、制表符或换行符结束。

%p 读入一个指针

%u 读入一个无符号十进制整数

%n 至此已读入值的等价字符数

%[] 扫描字符集合

%% 读%符号

## 数位dp模板

//MAX\_DIGITS表示最大位数

//MAX\_STATUS表示status的不同状态数，通常跟题目要求相关

//dp函数fun中，依据题意要求的多种状态，可以适当增加维度

const int MAX\_DIGITS**,** MAX\_STATUS**;**

LL fun**[**MAX\_DIGITS**][**MAX\_STATUS**],** bits**[**MAX\_DIGITS**];**

//position表示递归到达的位数

//status表明是当前递归时的状态

//limit=true表明position这一位上的数在MAXN\_DIGITS以内

//first通常用来判断前导零的情况（如windy数问题）,可以省略

LL dfs**(**int position**,** int status**,** bool limit**,** bool first**)**

**{**

//s==target\_status表明的是初始值，是递归出口

**if** **(**position **==** **-**1**)**

**return** s **==** target\_status**;**

**if** **(!**limit **&&** **!**first **&&** **~**fun**[**position**][**status**])**

**return** fun**[**position**][**status**];**

//MAX\_ BITS表明每一数位的最大可能取值，如10进制下MAX\_ BITS=9，2进制下MAX\_ BITS=1

int u **=** limit **?** bits**[**position**]** **:** MAX\_BITS**;**

LL ret **=** 0**;**

//一般i从0开始到u，但也有其他情况

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

**{**

//next\_status由当前状态status和i转化得到

//这里可能会有下一个状态是否满足要求的判断

ret **+=** dfs**(**position **-** 1**,** next\_status**(**status**,** i**),** limit **&&** **(**i **==** u**),** first **&&** **!**i**);**

**}**

**return** limit **||** first **?** ret **:** fun**[**pos**][**status**]** **=** ret**;**

**}**

//将n按10进制存入数组，可以改成其他进制形式

//数组bits是按低位到高位存入，但是dfs是从高位到低位递归

LL calc**(**LL n**)**

**{**

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

int len **=** 0**;**

**while** **(**n**)**

**{**

bits**[**len**++]** **=** n **%** 10**;**

n **/=** 10**;**

**}**

**return** dfs**(**len **-** 1**,** 0**,** **true,** **true);**

**}**

int main**()**

**{**

//freopen("0.txt", "r", stdin);

LL a**,** b**;**

**while** **(**cin **>>** a **>>** b**)**

cout **<<** calc**(**b**)** **-** calc**(**a **-** 1**)** **<<** endl**;**

**return** 0**;**

**}**

**//更一般的情况，适用于没有区间可加性的问题**

//此外，当需要维护多个与数位相关的状态时，可以将fun数组定义为结构体类型，如HDU 4507

const int MAX\_DIGITS**,**MAX\_STATUS**;**

LL fun**[**MAX\_DIGITS**][**MAX\_STATUS**];**

LL bitm**[**MAX\_DIGITS**],**bitn**[**MAX\_DIGITS**];**

LL dfs**(**int pos**,**int st**,**bool limm**,**bool limn**)**

**{**

**if(**pos**==-**1**)** **return** s**==**target\_status**;**

**if(!**limm**&&!**limn**&&~**fun**[**pos**][**st**])** **return** fun**[**pos**][**st**];**

int um**=**limm**?**bitm**[**pos**]:**MIN\_BITS**;**

int un**=**limn**?**bitn**[**pos**]:**MAX\_BITS**;**

LL ret**=**0**;**

**for(**int i**=**um**;** i**<=**un**;** i**++)**

**{**

ret **+=** dfs**(**position **-** 1**,** next\_status**(**status**,** i**),** limm **&&** **(**i **==** um**),** limn **&&** **(**i **==** un**) );**

**}**

**return** **(**limm**||**limn**)?**ret**:**fun**[**pos**][**st**]=**ret**;**

**}**

//函数calc要求m<=n

LL calc**(**LL m**,**LL n**)**

**{**

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

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

int len**=**0**;**

**while(**n**)**

**{**

bitm**[**len**++]=**m**%**10**;**

bitn**[**len**-**1**]=**n**%**10**;**

m**/=**10**;**

n**/=**10**;**

**}**

**return** dfs**(**len**-**1**,**0**,true,true);**

**}**

int main**()**

**{**

int T**;**

**while(**scanf**(**"%d"**,&**T**)!=**EOF**)**

**{**

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

**while(**T**--)**

**{**

LL left**,**right**;**

scanf**(**"%I64d%I64d"**,&**left**,&**right**);**

printf**(**"%I64d\n"**,**calc**(**left**,**right**));**

**}**

**}**

**return** 0**;**

**}**

# 特殊算法汇总

## 雅可比（Jacobi）迭代解线性方程组

//例题 HDU 5097

在的线性方程组中，定义矩阵

，，

，，

那么就有



因此可以将线性方程组化成



雅可比迭代法就是利用上式来进迭代的



对于每一个元素来说



朴素的算法思想就是，以某一个***x***向量为初始向量***x***(1)（通常为全1向量），经过多次迭代求出***x***(*k*+1)和***x***(*k*)，迭代的结束条件通常为两次相邻结果的向量差的模小于某个特定的值



这种算法常适用于求解的非零解情况，也适用于求一般的线性方程组情况。

## 几何图形的反演变换

//例题 HDU 4773

## 向量旋转矩阵

