# Zhejiang University

# ICPC Team

#### **Routine Library**

**by WishingBone (Dec. 2002)**

Last Update (Nov. 2004) by Riveria

[1、 几何 25](#_Toc87858530)

[1.1 注意 25](#_Toc87858531)

[1.2 几何公式 25](#_Toc87858532)

[1.3 多边形 27](#_Toc87858533)

[1.4 多边形切割 30](#_Toc87858534)

[1.5 浮点函数 31](#_Toc87858535)

[1.6 面积 36](#_Toc87858536)

[1.7 球面 37](#_Toc87858537)

[1.8 三角形 38](#_Toc87858538)

[1.9 三维几何 40](#_Toc87858539)

[1.10 凸包 47](#_Toc87858540)

[1.11 网格 49](#_Toc87858541)

[1.12 圆 49](#_Toc87858542)

[1.13 整数函数 51](#_Toc87858543)

[2、 组合 54](#_Toc87858544)

[2.1 组合公式 54](#_Toc87858545)

[2.2 排列组合生成 54](#_Toc87858546)

[2.3 生成gray码 56](#_Toc87858547)

[2.4 置换(polya) 56](#_Toc87858548)

[2.5 字典序全排列 57](#_Toc87858549)

[2.6 字典序组合 57](#_Toc87858550)

[3、 结构 58](#_Toc87858551)

[3.1 并查集 58](#_Toc87858552)

[3.2 堆 59](#_Toc87858553)

[3.3 线段树 60](#_Toc87858554)

[3.4 子段和 65](#_Toc87858555)

[3.5 子阵和 65](#_Toc87858556)

[4、 数论 66](#_Toc87858557)

[4.1 阶乘最后非0位 66](#_Toc87858558)

[4.2 模线性方程组 67](#_Toc87858559)

[4.3 素数 68](#_Toc87858560)

[4.4 欧拉函数 69](#_Toc87858561)

[5、 数值计算 70](#_Toc87858562)

[5.1 定积分计算(Romberg) 70](#_Toc87858563)

[5.2 多项式求根(牛顿法) 72](#_Toc87858564)

[5.3 周期性方程(追赶法) 73](#_Toc87858565)

[6、 图论—NP搜索 74](#_Toc87858566)

[6.1 最大团 74](#_Toc87858567)

[6.2 最大团(n<64)(faster) 75](#_Toc87858568)

[7、 图论—连通性 77](#_Toc87858569)

[7.1 无向图关键点(dfs邻接阵) 77](#_Toc87858570)

[7.2 无向图关键边(dfs邻接阵) 78](#_Toc87858571)

[7.3 无向图的块(bfs邻接阵) 79](#_Toc87858572)

[7.4 无向图连通分支(dfs/bfs邻接阵) 80](#_Toc87858573)

[7.5 有向图强连通分支(dfs/bfs邻接阵) 81](#_Toc87858574)

[7.6 有向图最小点基(邻接阵) 82](#_Toc87858575)

[8、 图论—匹配 83](#_Toc87858576)

[8.1 二分图最大匹配(hungary邻接表) 83](#_Toc87858577)

[8.2 二分图最大匹配(hungary邻接阵) 84](#_Toc87858578)

[8.3 二分图最大匹配(hungary正向表) 84](#_Toc87858579)

[8.4二分图最佳匹配(kuhn\_munkras邻接阵) 85](#_Toc87858580)

[8.5 一般图匹配(邻接表) 86](#_Toc87858581)

[8.6 一般图匹配(邻接阵) 87](#_Toc87858582)

[8.7 一般图匹配(正向表) 87](#_Toc87858583)

[9、 图论—网络流 88](#_Toc87858584)

[9.1 最大流(邻接阵) 88](#_Toc87858585)

[9.2 上下界最大流(邻接阵) 89](#_Toc87858586)

[9.3 上下界最小流(邻接阵) 90](#_Toc87858587)

[9.4 最大流无流量(邻接阵) 91](#_Toc87858588)

[9.5 最小费用最大流(邻接阵) 91](#_Toc87858589)

[10、 图论—应用 92](#_Toc87858590)

[10.1 欧拉回路(邻接阵) 92](#_Toc87858591)

[10.2 树的前序表转化 93](#_Toc87858592)

[10.3 树的优化算法 94](#_Toc87858593)

[10.4 拓扑排序(邻接阵) 95](#_Toc87858594)

[10.5 最佳边割集 96](#_Toc87858595)

[10.6 最佳点割集 97](#_Toc87858596)

[10.7 最小边割集 98](#_Toc87858597)

[10.8 最小点割集 99](#_Toc87858598)

[10.9 最小路径覆盖 101](#_Toc87858599)

[11、 图论—支撑树 101](#_Toc87858600)

[11.1 最小生成树(kruskal邻接表) 101](#_Toc87858601)

[11.2 最小生成树(kruskal正向表) 103](#_Toc87858602)

[11.3 最小生成树(prim+binary\_heap邻接表) 104](#_Toc87858603)

[11.4 最小生成树(prim+binary\_heap正向表) 105](#_Toc87858604)

[11.5 最小生成树(prim+mapped\_heap邻接表) 106](#_Toc87858605)

[11.6 最小生成树(prim+mapped\_heap正向表) 108](#_Toc87858606)

[11.7 最小生成树(prim邻接阵) 109](#_Toc87858607)

[11.8 最小树形图(邻接阵) 109](#_Toc87858608)

[12、 图论—最短路径 111](#_Toc87858609)

[12.1 最短路径(单源bellman\_ford邻接阵) 111](#_Toc87858610)

[12.2 最短路径(单源dijkstra+bfs邻接表) 111](#_Toc87858611)

[12.3 最短路径(单源dijkstra+bfs正向表) 112](#_Toc87858612)

[12.4 最短路径(单源dijkstra+binary\_heap邻接表) 113](#_Toc87858613)

[12.5 最短路径(单源dijkstra+binary\_heap正向表) 114](#_Toc87858614)

[12.6 最短路径(单源dijkstra+mapped\_heap邻接表) 115](#_Toc87858615)

[12.7 最短路径(单源dijkstra+mapped\_heap正向表) 116](#_Toc87858616)

[12.8 最短路径(单源dijkstra邻接阵) 117](#_Toc87858617)

[12.9 最短路径(多源floyd\_warshall邻接阵) 118](#_Toc87858618)

[13、 应用 118](#_Toc87858619)

[13.1 Joseph问题 118](#_Toc87858620)

[13.2 N皇后构造解 119](#_Toc87858621)

[13.3 布尔母函数 120](#_Toc87858622)

[13.4 第k元素 120](#_Toc87858623)

[13.5 幻方构造 121](#_Toc87858624)

[13.6 模式匹配(kmp) 122](#_Toc87858625)

[13.7 逆序对数 123](#_Toc87858626)

[13.8 字符串最小表示 123](#_Toc87858627)

[13.9 最长公共单调子序列 124](#_Toc87858628)

[13.10 最长子序列 125](#_Toc87858629)

[13.11 最大子串匹配 126](#_Toc87858630)

[13.12 最大子段和 127](#_Toc87858631)

[13.13 最大子阵和 127](#_Toc87858632)

[14、 其它 128](#_Toc87858633)

[14.1 大数(只能处理正数) 128](#_Toc87858634)

[14.2 分数 134](#_Toc87858635)

[14.3 矩阵 136](#_Toc87858636)

[14.4 线性方程组 138](#_Toc87858637)

[14.5 线性相关 140](#_Toc87858638)

[14.6 日期 140](#_Toc87858639)

## 几何

### 注意

1. 注意舍入方式(0.5的舍入方向);防止输出-0.

2. 几何题注意多测试不对称数据.

3. 整数几何注意xmult和dmult是否会出界;

符点几何注意eps的使用.

4. 避免使用斜率;注意除数是否会为0.

5. 公式一定要化简后再代入.

6. 判断同一个2\*PI域内两角度差应该是

abs(a1-a2)<beta||abs(a1-a2)>pi+pi-beta;

相等应该是

abs(a1-a2)<eps||abs(a1-a2)>pi+pi-eps;

7. 需要的话尽量使用atan2,注意:atan2(0,0)=0,

atan2(1,0)=pi/2,atan2(-1,0)=-pi/2,atan2(0,1)=0,atan2(0,-1)=pi.

8. cross product = |u|\*|v|\*sin(a)

dot product = |u|\*|v|\*cos(a)

9. (P1-P0)x(P2-P0)结果的意义:

正: <P0,P1>在<P0,P2>顺时针(0,pi)内

负: <P0,P1>在<P0,P2>逆时针(0,pi)内

0 : <P0,P1>,<P0,P2>共线,夹角为0或pi

10. 误差限缺省使用1e-8!

### 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

### 多边形

#include <stdlib.h>

#include <math.h>

#define MAXN 1000

#define offset 10000

#define eps 1e-8

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

#define \_sign(x) ((x)>eps?1:((x)<-eps?2:0))

struct point{double x,y;};

struct line{point a,b;};

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

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

}

//判定凸多边形,顶点按顺时针或逆时针给出,允许相邻边共线

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

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

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

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

return s[1]|s[2];

}

//判定凸多边形,顶点按顺时针或逆时针给出,不允许相邻边共线

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

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

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

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

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

}

//判点在凸多边形内或多边形边上,顶点按顺时针或逆时针给出

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

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

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

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

return s[1]|s[2];

}

//判点在凸多边形内,顶点按顺时针或逆时针给出,在多边形边上返回0

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

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

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

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

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

}

//判点在任意多边形内,顶点按顺时针或逆时针给出

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

int inside\_polygon(point q,int n,point\* p,int on\_edge=1){

point q2;

int i=0,count;

while (i<n)

for (count=i=0,q2.x=rand()+offset,q2.y=rand()+offset;i<n;i++)

if (zero(xmult(q,p[i],p[(i+1)%n]))&&(p[i].x-q.x)\*(p[(i+1)%n].x-q.x)<eps&&(p[i].y-q.y)\*(p[(i+1)%n].y-q.y)<eps)

return on\_edge;

else if (zero(xmult(q,q2,p[i])))

break;

else if (xmult(q,p[i],q2)\*xmult(q,p[(i+1)%n],q2)<-eps&&xmult(p[i],q,p[(i+1)%n])\*xmult(p[i],q2,p[(i+1)%n])<-eps)

count++;

return count&1;

}

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

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

}

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

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

}

//判线段在任意多边形内,顶点按顺时针或逆时针给出,与边界相交返回1

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

point t[MAXN],tt;

int i,j,k=0;

if (!inside\_polygon(l1,n,p)||!inside\_polygon(l2,n,p))

return 0;

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

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

return 0;

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

t[k++]=l1;

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

t[k++]=l2;

else if (dot\_online\_in(p[i],l1,l2))

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

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

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

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

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

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

return 0;

}

return 1;

}

point intersection(line u,line v){

point ret=u.a;

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

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

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

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

return ret;

}

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

line u,v;

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

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

u.b=c;

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

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

v.b=b;

return intersection(u,v);

}

//多边形重心

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

point ret,t;

double t1=0,t2;

int i;

ret.x=ret.y=0;

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

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

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

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

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

t1+=t2;

}

if (fabs(t1)>eps)

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

return ret;

}

### 多边形切割

//多边形切割

//可用于半平面交

#define MAXN 100

#define eps 1e-8

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

struct point{double x,y;};

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

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

}

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

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

}

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

point ret=u1;

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

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

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

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

return ret;

}

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

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

point pp[100];

int m=0,i;

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

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

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

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

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

}

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

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

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

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

n--;

if (n<3)

n=0;

}

### 浮点函数

//浮点几何函数库

#include <math.h>

#define eps 1e-8

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

struct point{double x,y;};

struct line{point a,b;};

//计算cross product (P1-P0)x(P2-P0)

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

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

}

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

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

}

//计算dot product (P1-P0).(P2-P0)

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

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

}

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

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

}

//两点距离

double distance(point p1,point p2){

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

}

double distance(double x1,double y1,double x2,double y2){

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

}

//判三点共线

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

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

}

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

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

}

//判点是否在线段上,包括端点

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

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

}

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

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

}

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

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

}

//判点是否在线段上,不包括端点

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

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

}

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

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

}

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

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

}

//判两点在线段同侧,点在线段上返回0

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

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

}

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

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

}

//判两点在线段异侧,点在线段上返回0

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

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

}

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

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

}

//判两直线平行

int parallel(line u,line v){

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

}

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

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

}

//判两直线垂直

int perpendicular(line u,line v){

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

}

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

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

}

//判两线段相交,包括端点和部分重合

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

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

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

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

}

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

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

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

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

}

//判两线段相交,不包括端点和部分重合

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

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

}

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

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

}

//计算两直线交点,注意事先判断直线是否平行!

//线段交点请另外判线段相交(同时还是要判断是否平行!)

point intersection(line u,line v){

point ret=u.a;

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

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

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

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

return ret;

}

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

point ret=u1;

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

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

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

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

return ret;

}

//点到直线上的最近点

point ptoline(point p,line l){

point t=p;

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

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

}

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

point t=p;

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

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

}

//点到直线距离

double disptoline(point p,line l){

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

}

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

return fabs(xmult(p,l1,l2))/distance(l1,l2);

}

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

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

}

//点到线段上的最近点

point ptoseg(point p,line l){

point t=p;

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

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

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

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

}

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

point t=p;

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

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

return distance(p,l1)<distance(p,l2)?l1:l2;

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

}

//点到线段距离

double disptoseg(point p,line l){

point t=p;

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

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

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

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

}

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

point t=p;

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

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

return distance(p,l1)<distance(p,l2)?distance(p,l1):distance(p,l2);

return fabs(xmult(p,l1,l2))/distance(l1,l2);

}

//矢量V以P为顶点逆时针旋转angle并放大scale倍

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

point ret=p;

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

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

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

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

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

return ret;

}

### 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;

}

### 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 distance(point p1,point p2){

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

}

point intersection(line u,line v){

point ret=u.a;

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

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

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

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

return ret;

}

//外心

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

line u,v;

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

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

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

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

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

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

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

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

return intersection(u,v);

}

//内心

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

line u,v;

double m,n;

u.a=a;

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

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

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

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

v.a=b;

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

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

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

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

return intersection(u,v);

}

//垂心

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

line u,v;

u.a=c;

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

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

v.a=b;

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

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

return intersection(u,v);

}

//重心

//到三角形三顶点距离的平方和最小的点

//三角形内到三边距离之积最大的点

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

line u,v;

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

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

u.b=c;

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

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

v.b=b;

return intersection(u,v);

}

//费马点

//到三角形三顶点距离之和最小的点

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

point u,v;

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

int i,j,k;

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

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

while (step>1e-10)

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

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

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

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

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

if (distance(u,a)+distance(u,b)+distance(u,c)>distance(v,a)+distance(v,b)+distance(v,c))

u=v;

}

return u;

}

### 三维几何

//三维几何函数库

#include <math.h>

#define eps 1e-8

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

struct point3{double x,y,z;};

struct line3{point3 a,b;};

struct plane3{point3 a,b,c;};

//计算cross product U x V

point3 xmult(point3 u,point3 v){

point3 ret;

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

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

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

return ret;

}

//计算dot product U . V

double dmult(point3 u,point3 v){

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

}

//矢量差 U - V

point3 subt(point3 u,point3 v){

point3 ret;

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

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

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

return ret;

}

//取平面法向量

point3 pvec(plane3 s){

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

}

point3 pvec(point3 s1,point3 s2,point3 s3){

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

}

//两点距离,单参数取向量大小

double distance(point3 p1,point3 p2){

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

}

//向量大小

double vlen(point3 p){

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

}

//判三点共线

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

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

}

//判四点共面

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

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

}

//判点是否在线段上,包括端点和共线

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

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

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

}

int dot\_online\_in(point3 p,point3 l1,point3 l2){

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

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

}

//判点是否在线段上,不包括端点

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

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

(!zero(p.x-l.b.x)||!zero(p.y-l.b.y)||!zero(p.z-l.b.z));

}

int dot\_online\_ex(point3 p,point3 l1,point3 l2){

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

(!zero(p.x-l2.x)||!zero(p.y-l2.y)||!zero(p.z-l2.z));

}

//判点是否在空间三角形上,包括边界,三点共线无意义

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

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

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

}

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

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

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

}

//判点是否在空间三角形上,不包括边界,三点共线无意义

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

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

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

}

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

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

vlen(xmult(subt(p,s2),subt(p,s3)))>eps&&vlen(xmult(subt(p,s3),subt(p,s1)))>eps;

}

//判两点在线段同侧,点在线段上返回0,不共面无意义

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

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

}

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

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

}

//判两点在线段异侧,点在线段上返回0,不共面无意义

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

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

}

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

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

}

//判两点在平面同侧,点在平面上返回0

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

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

}

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

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

}

//判两点在平面异侧,点在平面上返回0

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

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

}

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

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

}

//判两直线平行

int parallel(line3 u,line3 v){

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

}

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

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

}

//判两平面平行

int parallel(plane3 u,plane3 v){

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

}

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

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

}

//判直线与平面平行

int parallel(line3 l,plane3 s){

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

}

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

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

}

//判两直线垂直

int perpendicular(line3 u,line3 v){

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

}

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

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

}

//判两平面垂直

int perpendicular(plane3 u,plane3 v){

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

}

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

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

}

//判直线与平面平行

int perpendicular(line3 l,plane3 s){

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

}

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

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

}

//判两线段相交,包括端点和部分重合

int intersect\_in(line3 u,line3 v){

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

return 0;

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

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

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

}

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

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

return 0;

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

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

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

}

//判两线段相交,不包括端点和部分重合

int intersect\_ex(line3 u,line3 v){

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

}

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

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

}

//判线段与空间三角形相交,包括交于边界和(部分)包含

int intersect\_in(line3 l,plane3 s){

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

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

}

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

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

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

}

//判线段与空间三角形相交,不包括交于边界和(部分)包含

int intersect\_ex(line3 l,plane3 s){

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

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

}

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

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

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

}

//计算两直线交点,注意事先判断直线是否共面和平行!

//线段交点请另外判线段相交(同时还是要判断是否平行!)

point3 intersection(line3 u,line3 v){

point3 ret=u.a;

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

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

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

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

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

return ret;

}

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

point3 ret=u1;

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

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

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

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

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

return ret;

}

//计算直线与平面交点,注意事先判断是否平行,并保证三点不共线!

//线段和空间三角形交点请另外判断

point3 intersection(line3 l,plane3 s){

point3 ret=pvec(s);

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

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

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

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

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

return ret;

}

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

point3 ret=pvec(s1,s2,s3);

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

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

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

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

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

return ret;

}

//计算两平面交线,注意事先判断是否平行,并保证三点不共线!

line3 intersection(plane3 u,plane3 v){

line3 ret;

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

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

return ret;

}

line3 intersection(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){

line3 ret;

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

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

return ret;

}

//点到直线距离

double ptoline(point3 p,line3 l){

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

}

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

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

}

//点到平面距离

double ptoplane(point3 p,plane3 s){

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

}

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

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

}

//直线到直线距离

double linetoline(line3 u,line3 v){

point3 n=xmult(subt(u.a,u.b),subt(v.a,v.b));

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

}

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

point3 n=xmult(subt(u1,u2),subt(v1,v2));

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

}

//两直线夹角cos值

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

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

}

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

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

}

//两平面夹角cos值

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

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

}

double angle\_cos(point3 u1,point3 u2,point3 u3,point3 v1,point3 v2,point3 v3){

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

}

//直线平面夹角sin值

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

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

}

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

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

}

### 凸包

#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;

}

### 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;

}

### 圆

#include <math.h>

#define eps 1e-8

struct point{double x,y;};

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

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

}

double distance(point p1,point p2){

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

}

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

return fabs(xmult(p,l1,l2))/distance(l1,l2);

}

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

point ret=u1;

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

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

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

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

return ret;

}

//判直线和圆相交,包括相切

int intersect\_line\_circle(point c,double r,point l1,point l2){

return disptoline(c,l1,l2)<r+eps;

}

//判线段和圆相交,包括端点和相切

int intersect\_seg\_circle(point c,double r,point l1,point l2){

double t1=distance(c,l1)-r,t2=distance(c,l2)-r;

point t=c;

if (t1<eps||t2<eps)

return t1>-eps||t2>-eps;

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

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

return xmult(l1,c,t)\*xmult(l2,c,t)<eps&&disptoline(c,l1,l2)-r<eps;

}

//判圆和圆相交,包括相切

int intersect\_circle\_circle(point c1,double r1,point c2,double r2){

return distance(c1,c2)<r1+r2+eps&&distance(c1,c2)>fabs(r1-r2)-eps;

}

//计算圆上到点p最近点,如p与圆心重合,返回p本身

point dot\_to\_circle(point c,double r,point p){

point u,v;

if (distance(p,c)<eps)

return p;

u.x=c.x+r\*fabs(c.x-p.x)/distance(c,p);

u.y=c.y+r\*fabs(c.y-p.y)/distance(c,p)\*((c.x-p.x)\*(c.y-p.y)<0?-1:1);

v.x=c.x-r\*fabs(c.x-p.x)/distance(c,p);

v.y=c.y-r\*fabs(c.y-p.y)/distance(c,p)\*((c.x-p.x)\*(c.y-p.y)<0?-1:1);

return distance(u,p)<distance(v,p)?u:v;

}

//计算直线与圆的交点,保证直线与圆有交点

//计算线段与圆的交点可用这个函数后判点是否在线段上

void intersection\_line\_circle(point c,double r,point l1,point l2,point& p1,point& p2){

point p=c;

double t;

p.x+=l1.y-l2.y;

p.y+=l2.x-l1.x;

p=intersection(p,c,l1,l2);

t=sqrt(r\*r-distance(p,c)\*distance(p,c))/distance(l1,l2);

p1.x=p.x+(l2.x-l1.x)\*t;

p1.y=p.y+(l2.y-l1.y)\*t;

p2.x=p.x-(l2.x-l1.x)\*t;

p2.y=p.y-(l2.y-l1.y)\*t;

}

//计算圆与圆的交点,保证圆与圆有交点,圆心不重合

void intersection\_circle\_circle(point c1,double r1,point c2,double r2,point& p1,point& p2){

point u,v;

double t;

t=(1+(r1\*r1-r2\*r2)/distance(c1,c2)/distance(c1,c2))/2;

u.x=c1.x+(c2.x-c1.x)\*t;

u.y=c1.y+(c2.y-c1.y)\*t;

v.x=u.x+c1.y-c2.y;

v.y=u.y-c1.x+c2.x;

intersection\_line\_circle(c1,r1,u,v,p1,p2);

}

### 整数函数

//整数几何函数库

//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);

}

## 组合

### 2.1 组合公式

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

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

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

= (n-1)(D(n-2) - D(n-1))

Q(n) = D(n) + D(n-1)

求和公式,k = 1..n

1. sum( k ) = n(n+1)/2

2. sum( 2k-1 ) = n^2

3. sum( k^2 ) = n(n+1)(2n+1)/6

4. sum( (2k-1)^2 ) = n(4n^2-1)/3

5. sum( k^3 ) = (n(n+1)/2)^2

6. sum( (2k-1)^3 ) = n^2(2n^2-1)

7. sum( k^4 ) = n(n+1)(2n+1)(3n^2+3n-1)/30

8. sum( k^5 ) = n^2(n+1)^2(2n^2+2n-1)/12

9. sum( k(k+1) ) = n(n+1)(n+2)/3

10. sum( k(k+1)(k+2) ) = n(n+1)(n+2)(n+3)/4

12. sum( k(k+1)(k+2)(k+3) ) = n(n+1)(n+2)(n+3)(n+4)/5

### 2.2 排列组合生成

//gen\_perm产生字典序排列P(n,m)

//gen\_comb产生字典序组合C(n,m)

//gen\_perm\_swap产生相邻位对换全排列P(n,n)

//产生元素用1..n表示

//dummy为产生后调用的函数,传入a[]和n,a[0]..a[n-1]为一次产生的结果

#define MAXN 100

int count;

#include <iostream.h>

void dummy(int\* a,int n){

int i;

cout<<count++<<": ";

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

cout<<a[i]<<' ';

cout<<a[n-1]<<endl;

}

void \_gen\_perm(int\* a,int n,int m,int l,int\* temp,int\* tag){

int i;

if (l==m)

dummy(temp,m);

else

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

if (!tag[i]){

temp[l]=a[i],tag[i]=1;

\_gen\_perm(a,n,m,l+1,temp,tag);

tag[i]=0;

}

}

void gen\_perm(int n,int m){

int a[MAXN],temp[MAXN],tag[MAXN]={0},i;

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

a[i]=i+1;

\_gen\_perm(a,n,m,0,temp,tag);

}

void \_gen\_comb(int\* a,int s,int e,int m,int& count,int\* temp){

int i;

if (!m)

dummy(temp,count);

else

for (i=s;i<=e-m+1;i++){

temp[count++]=a[i];

\_gen\_comb(a,i+1,e,m-1,count,temp);

count--;

}

}

void gen\_comb(int n,int m){

int a[MAXN],temp[MAXN],count=0,i;

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

a[i]=i+1;

\_gen\_comb(a,0,n-1,m,count,temp);

}

void \_gen\_perm\_swap(int\* a,int n,int l,int\* pos,int\* dir){

int i,p1,p2,t;

if (l==n)

dummy(a,n);

else{

\_gen\_perm\_swap(a,n,l+1,pos,dir);

for (i=0;i<l;i++){

p2=(p1=pos[l])+dir[l];

t=a[p1],a[p1]=a[p2],a[p2]=t;

pos[a[p1]-1]=p1,pos[a[p2]-1]=p2;

\_gen\_perm\_swap(a,n,l+1,pos,dir);

}

dir[l]=-dir[l];

}

}

void gen\_perm\_swap(int n){

int a[MAXN],pos[MAXN],dir[MAXN],i;

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

a[i]=i+1,pos[i]=i,dir[i]=-1;

\_gen\_perm\_swap(a,n,0,pos,dir);

}

### 2.3 生成gray码

//生成reflected gray code

//每次调用gray取得下一个码

//000...000是第一个码,100...000是最后一个码

void gray(int n,int \*code){

int t=0,i;

for (i=0;i<n;t+=code[i++]);

if (t&1)

for (n--;!code[n];n--);

code[n-1]=1-code[n-1];

}

### 2.4 置换(polya)

//求置换的循环节,polya原理

//perm[0..n-1]为0..n-1的一个置换(排列)

//返回置换最小周期,num返回循环节个数

#define MAXN 1000

int gcd(int a,int b){

return b?gcd(b,a%b):a;

}

int polya(int\* perm,int n,int& num){

int i,j,p,v[MAXN]={0},ret=1;

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

if (!v[i]){

for (num++,j=0,p=i;!v[p=perm[p]];j++)

v[p]=1;

ret\*=j/gcd(ret,j);

}

return ret;

}

### 2.5 字典序全排列

//字典序全排列与序号的转换

int perm2num(int n,int \*p){

int i,j,ret=0,k=1;

for (i=n-2;i>=0;k\*=n-(i--))

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

if (p[j]<p[i])

ret+=k;

return ret;

}

void num2perm(int n,int \*p,int t){

int i,j;

for (i=n-1;i>=0;i--)

p[i]=t%(n-i),t/=n-i;

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

for (j=i-1;j>=0;j--)

if (p[j]<=p[i])

p[i]++;

}

### 2.6 字典序组合

//字典序组合与序号的转换

//comb为组合数C(n,m),必要时换成大数,注意处理C(n,m)=0|n<m

int comb(int n,int m){

int ret=1,i;

m=m<(n-m)?m:(n-m);

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

for (i=1;i<=m;ret/=(i++));

return m<0?0:ret;

}

int comb2num(int n,int m,int \*c){

int ret=comb(n,m),i;

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

ret-=comb(n-c[i],m-i);

return ret;

}

void num2comb(int n,int m,int\* c,int t){

int i,j=1,k;

for (i=0;i<m;c[i++]=j++)

for (;t>(k=comb(n-j,m-i-1));t-=k,j++);

}

## 结构

### 3.1 并查集

//带路径压缩的并查集,用于动态维护查询等价类

//图论算法中动态判点集连通常用

//维护和查询复杂度略大于O(1)

//集合元素取值1..MAXN-1(注意0不能用!),默认不等价

#include <string.h>

#define MAXN 100000

#define \_ufind\_run(x) for(;p[t=x];x=p[x],p[t]=(p[x]?p[x]:x))

#define \_run\_both \_ufind\_run(i);\_ufind\_run(j)

struct ufind{

int p[MAXN],t;

void init(){memset(p,0,sizeof(p));}

void set\_friend(int i,int j){\_run\_both;p[i]=(i==j?0:j);}

int is\_friend(int i,int j){\_run\_both;return i==j&&i;}

};

//带路径压缩的并查集扩展形式

//用于动态维护查询friend-enemy型等价类

//维护和查询复杂度略大于O(1)

//集合元素取值1..MAXN-1(注意0不能用!),默认无关

#include <string.h>

#define MAXN 100000

#define sig(x) ((x)>0?1:-1)

#define abs(x) ((x)>0?(x):-(x))

#define \_ufind\_run(x) for(;p[t=abs(x)];x=sig(x)\*p[abs(x)],p[t]=sig(p[t])\*(p[abs(x)]?p[abs(x)]:abs(p[t])))

#define \_run\_both \_ufind\_run(i);\_ufind\_run(j)

#define \_set\_side(x) p[abs(i)]=sig(i)\*(abs(i)==abs(j)?0:(x)\*j)

#define \_judge\_side(x) (i==(x)\*j&&i)

struct ufind{

int p[MAXN],t;

void init(){memset(p,0,sizeof(p));}

int set\_friend(int i,int j){\_run\_both;\_set\_side(1);return !\_judge\_side(-1);}

int set\_enemy(int i,int j){\_run\_both;\_set\_side(-1);return !\_judge\_side(1);}

int is\_friend(int i,int j){\_run\_both;return \_judge\_side(1);}

int is\_enemy(int i,int j){\_run\_both;return \_judge\_side(-1);}

};

### 3.2 堆

//二分堆(binary)

//可插入,获取并删除最小(最大)元素,复杂度均O(logn)

//可更改元素类型,修改比较符号或换成比较函数

#define MAXN 10000

#define \_cp(a,b) ((a)<(b))

typedef int elem\_t;

struct heap{

elem\_t h[MAXN];

int n,p,c;

void init(){n=0;}

void ins(elem\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(elem\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

//映射二分堆(mapped)

//可插入,获取并删除任意元素,复杂度均O(logn)

//插入时提供一个索引值,删除时按该索引删除,获取并删除最小元素时一起获得该索引

//索引值范围0..MAXN-1,不能重复,不负责维护索引的唯一性,不在此返回请另外映射

//主要用于图论算法,该索引值可以是节点的下标

//可更改元素类型,修改比较符号或换成比较函数

#define MAXN 10000

#define \_cp(a,b) ((a)<(b))

typedef int elem\_t;

struct heap{

elem\_t h[MAXN];

int ind[MAXN],map[MAXN],n,p,c;

void init(){n=0;}

void ins(int i,elem\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

h[map[ind[p]=i]=p]=e;

}

int del(int i,elem\_t& e){

i=map[i];if (i<1||i>n) return 0;

for (e=h[p=i];p>1;h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

for (c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

int delmin(int& i,elem\_t& e){

if (n<1) return 0;i=ind[1];

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

};

### 3.3 线段树

线段树应用：

求面积:

1) 坐标离散化

2) 垂直边按x坐标排序

3) 从左往右用线段树处理垂直边

累计每个离散x区间长度和线段树长度的乘积

求周长:

1) 坐标离散化

2) 垂直边按x坐标排序, 第二关键字为入边优于出边

3) 从左往右用线段树处理垂直边

在每个离散点上先加入所有入边, 累计线段树长度变化值

再删除所有出边, 累计线段树长度变化值

4) 水平边按y坐标排序, 第二关键字为入边优于出边

5) 从上往下用线段树处理水平边

在每个离散点上先加入所有入边, 累计线段树长度变化值

再删除所有出边, 累计线段树长度变化值

//线段树

//可以处理加入边和删除边不同的情况

//inc\_seg和dec\_seg用于加入边

//seg\_len求长度

//t传根节点(一律为1)

//l0,r0传树的节点范围(一律为1..t)

//l,r传线段(端点)

#define MAXN 10000

struct segtree{

int n,cnt[MAXN],len[MAXN];

segtree(int t):n(t){

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

cnt[i]=len[i]=0;

};

void update(int t,int l,int r);

void inc\_seg(int t,int l0,int r0,int l,int r);

void dec\_seg(int t,int l0,int r0,int l,int r);

int seg\_len(int t,int l0,int r0,int l,int r);

};

int length(int l,int r){

return r-l;

}

void segtree::update(int t,int l,int r){

if (cnt[t]||r-l==1)

len[t]=length(l,r);

else

len[t]=len[t+t]+len[t+t+1];

}

void segtree::inc\_seg(int t,int l0,int r0,int l,int r){

if (l0==l&&r0==r)

cnt[t]++;

else{

int m0=(l0+r0)>>1;

if (l<m0)

inc\_seg(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

inc\_seg(t+t+1,m0,r0,m0>l?m0:l,r);

if (cnt[t+t]&&cnt[t+t+1]){

cnt[t+t]--;

update(t+t,l0,m0);

cnt[t+t+1]--;

update(t+t+1,m0,r0);

cnt[t]++;

}

}

update(t,l0,r0);

}

void segtree::dec\_seg(int t,int l0,int r0,int l,int r){

if (l0==l&&r0==r)

cnt[t]--;

else if (cnt[t]){

cnt[t]--;

if (l>l0)

inc\_seg(t,l0,r0,l0,l);

if (r<r0)

inc\_seg(t,l0,r0,r,r0);

}

else{

int m0=(l0+r0)>>1;

if (l<m0)

dec\_seg(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

dec\_seg(t+t+1,m0,r0,m0>l?m0:l,r);

}

update(t,l0,r0);

}

int segtree::seg\_len(int t,int l0,int r0,int l,int r){

if (cnt[t]||(l0==l&&r0==r))

return len[t];

else{

int m0=(l0+r0)>>1,ret=0;

if (l<m0)

ret+=seg\_len(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

ret+=seg\_len(t+t+1,m0,r0,m0>l?m0:l,r);

return ret;

}

}

//线段树扩展

//可以计算长度和线段数

//可以处理加入边和删除边不同的情况

//inc\_seg和dec\_seg用于加入边

//seg\_len求长度,seg\_cut求线段数

//t传根节点(一律为1)

//l0,r0传树的节点范围(一律为1..t)

//l,r传线段(端点)

#define MAXN 10000

struct segtree{

int n,cnt[MAXN],len[MAXN],cut[MAXN],bl[MAXN],br[MAXN];

segtree(int t):n(t){

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

cnt[i]=len[i]=cut[i]=bl[i]=br[i]=0;

};

void update(int t,int l,int r);

void inc\_seg(int t,int l0,int r0,int l,int r);

void dec\_seg(int t,int l0,int r0,int l,int r);

int seg\_len(int t,int l0,int r0,int l,int r);

int seg\_cut(int t,int l0,int r0,int l,int r);

};

int length(int l,int r){

return r-l;

}

void segtree::update(int t,int l,int r){

if (cnt[t]||r-l==1)

len[t]=length(l,r),cut[t]=bl[t]=br[t]=1;

else{

len[t]=len[t+t]+len[t+t+1];

cut[t]=cut[t+t]+cut[t+t+1];

if (br[t+t]&&bl[t+t+1])

cut[t]--;

bl[t]=bl[t+t],br[t]=br[t+t+1];

}

}

void segtree::inc\_seg(int t,int l0,int r0,int l,int r){

if (l0==l&&r0==r)

cnt[t]++;

else{

int m0=(l0+r0)>>1;

if (l<m0)

inc\_seg(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

inc\_seg(t+t+1,m0,r0,m0>l?m0:l,r);

if (cnt[t+t]&&cnt[t+t+1]){

cnt[t+t]--;

update(t+t,l0,m0);

cnt[t+t+1]--;

update(t+t+1,m0,r0);

cnt[t]++;

}

}

update(t,l0,r0);

}

void segtree::dec\_seg(int t,int l0,int r0,int l,int r){

if (l0==l&&r0==r)

cnt[t]--;

else if (cnt[t]){

cnt[t]--;

if (l>l0)

inc\_seg(t,l0,r0,l0,l);

if (r<r0)

inc\_seg(t,l0,r0,r,r0);

}

else{

int m0=(l0+r0)>>1;

if (l<m0)

dec\_seg(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

dec\_seg(t+t+1,m0,r0,m0>l?m0:l,r);

}

update(t,l0,r0);

}

int segtree::seg\_len(int t,int l0,int r0,int l,int r){

if (cnt[t]||(l0==l&&r0==r))

return len[t];

else{

int m0=(l0+r0)>>1,ret=0;

if (l<m0)

ret+=seg\_len(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

ret+=seg\_len(t+t+1,m0,r0,m0>l?m0:l,r);

return ret;

}

}

int segtree::seg\_cut(int t,int l0,int r0,int l,int r){

if (cnt[t])

return 1;

if (l0==l&&r0==r)

return cut[t];

else{

int m0=(l0+r0)>>1,ret=0;

if (l<m0)

ret+=seg\_cut(t+t,l0,m0,l,m0<r?m0:r);

if (r>m0)

ret+=seg\_cut(t+t+1,m0,r0,m0>l?m0:l,r);

if (l<m0&&r>m0&&br[t+t]&&bl[t+t+1])

ret--;

return ret;

}

}

### 3.4 子段和

//求sum{[0..n-1]}

//维护和查询复杂度均为O(logn)

//用于动态求子段和,数组内容保存在sum.a[]中

//可以改成其他数据类型

#include <string.h>

#define lowbit(x) ((x)&((x)^((x)-1)))

#define MAXN 10000

typedef int elem\_t;

struct sum{

elem\_t a[MAXN],c[MAXN],ret;

int n;

void init(int i){memset(a,0,sizeof(a));memset(c,0,sizeof(c));n=i;}

void update(int i,elem\_t v){for (v-=a[i],a[i++]+=v;i<=n;c[i-1]+=v,i+=lowbit(i));}

elem\_t query(int i){for (ret=0;i;ret+=c[i-1],i^=lowbit(i));return ret;}

};

### 3.5 子阵和

//求sum{a[0..m-1][0..n-1]}

//维护和查询复杂度均为O(logm\*logn)

//用于动态求子阵和,数组内容保存在sum.a[][]中

//可以改成其他数据类型

#include <string.h>

#define lowbit(x) ((x)&((x)^((x)-1)))

#define MAXN 100

typedef int elem\_t;

struct sum{

elem\_t a[MAXN][MAXN],c[MAXN][MAXN],ret;

int m,n,t;

void init(int i,int j){memset(a,0,sizeof(a));memset(c,0,sizeof(c));m=i,n=j;}

void update(int i,int j,elem\_t v){

for (v-=a[i][j],a[i++][j++]+=v,t=j;i<=m;i+=lowbit(i))

for (j=t;j<=n;c[i-1][j-1]+=v,j+=lowbit(j));

}

elem\_t query(int i,int j){

for (ret=0,t=j;i;i^=lowbit(i))

for (j=t;j;ret+=c[i-1][j-1],j^=lowbit(j));

return ret;

}

};

## 数论

### 4.1 阶乘最后非0位

//求阶乘最后非零位,复杂度O(nlogn)

//返回该位,n以字符串方式传入

#include <string.h>

#define MAXN 10000

int lastdigit(char\* buf){

const int mod[20]={1,1,2,6,4,2,2,4,2,8,4,4,8,4,6,8,8,6,8,2};

int len=strlen(buf),a[MAXN],i,c,ret=1;

if (len==1)

return mod[buf[0]-'0'];

for (i=0;i<len;i++)

a[i]=buf[len-1-i]-'0';

for (;len;len-=!a[len-1]){

ret=ret\*mod[a[1]%2\*10+a[0]]%5;

for (c=0,i=len-1;i>=0;i--)

c=c\*10+a[i],a[i]=c/5,c%=5;

}

return ret+ret%2\*5;

}

### 4.2 模线性方程组

#ifdef WIN32

typedef \_\_int64 i64;

#else

typedef long long i64;

#endif

//扩展Euclid求解gcd(a,b)=ax+by

int ext\_gcd(int a,int b,int& x,int& y){

int t,ret;

if (!b){

x=1,y=0;

return a;

}

ret=ext\_gcd(b,a%b,x,y);

t=x,x=y,y=t-a/b\*y;

return ret;

}

//计算m^a, O(loga), 本身没什么用, 注意这个按位处理的方法 :-P

int exponent(int m,int a){

int ret=1;

for (;a;a>>=1,m\*=m)

if (a&1)

ret\*=m;

return ret;

}

//计算幂取模a^b mod n, O(logb)

int modular\_exponent(int a,int b,int n){ //a^b mod n

int ret=1;

for (;b;b>>=1,a=(int)((i64)a)\*a%n)

if (b&1)

ret=(int)((i64)ret)\*a%n;

return ret;

}

//求解模线性方程ax=b (mod n)

//返回解的个数,解保存在sol[]中

//要求n>0,解的范围0..n-1

int modular\_linear(int a,int b,int n,int\* sol){

int d,e,x,y,i;

d=ext\_gcd(a,n,x,y);

if (b%d)

return 0;

e=(x\*(b/d)%n+n)%n;

for (i=0;i<d;i++)

sol[i]=(e+i\*(n/d))%n;

return d;

}

//求解模线性方程组(中国余数定理)

// x = b[0] (mod w[0])

// x = b[1] (mod w[1])

// ...

// x = b[k-1] (mod w[k-1])

//要求w[i]>0,w[i]与w[j]互质,解的范围1..n,n=w[0]\*w[1]\*...\*w[k-1]

int modular\_linear\_system(int b[],int w[],int k){

int d,x,y,a=0,m,n=1,i;

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

n\*=w[i];

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

m=n/w[i];

d=ext\_gcd(w[i],m,x,y);

a=(a+y\*m\*b[i])%n;

}

return (a+n)%n;

}

### 4.3 素数

//用素数表判定素数,先调用initprime

int plist[10000],pcount=0;

int prime(int n){

int i;

if ((n!=2&&!(n%2))||(n!=3&&!(n%3))||(n!=5&&!(n%5))||(n!=7&&!(n%7)))

return 0;

for (i=0;plist[i]\*plist[i]<=n;i++)

if (!(n%plist[i]))

return 0;

return n>1;

}

void initprime(){

int i;

for (plist[pcount++]=2,i=3;i<50000;i++)

if (prime(i))

plist[pcount++]=i;

}

//miller rabin

//判断自然数n是否为素数

//time越高失败概率越低,一般取10到50

#include <stdlib.h>

#ifdef WIN32

typedef \_\_int64 i64;

#else

typedef long long i64;

#endif

int modular\_exponent(int a,int b,int n){ //a^b mod n

int ret;

for (;b;b>>=1,a=(int)((i64)a)\*a%n)

if (b&1)

ret=(int)((i64)ret)\*a%n;

return ret;

}

// Carmicheal number: 561,41041,825265,321197185

int miller\_rabin(int n,int time=10){

if (n==1||(n!=2&&!(n%2))||(n!=3&&!(n%3))||(n!=5&&!(n%5))||(n!=7&&!(n%7)))

return 0;

while (time--)

if (modular\_exponent(((rand()&0x7fff<<16)+rand()&0x7fff+rand()&0x7fff)%(n-1)+1,n-1,n)!=1)

return 0;

return 1;

}

### 4.4 欧拉函数

int gcd(int a,int b){

return b?gcd(b,a%b):a;

}

inline int lcm(int a,int b){

return a/gcd(a,b)\*b;

}

//求1..n-1中与n互质的数的个数

int eular(int n){

int ret=1,i;

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

if (n%i==0){

n/=i,ret\*=i-1;

while (n%i==0)

n/=i,ret\*=i;

}

if (n>1)

ret\*=n-1;

return ret;

}

## 数值计算

### 5.1 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 Integral(double a, double b, double (\*f)(double x, double y, double z), double eps,

double l, double t)

double Romberg (double a, double b, double (\*f)(double x, double y, double z), double eps,

double l, double t)

{

#define MAX\_N 1000

int i, j, temp2, min;

double h, R[2][MAX\_N], temp4;

for (i=0; i<MAX\_N; i++) {

R[0][i] = 0.0;

R[1][i] = 0.0;

}

h = b-a;

min = (int)(log(h\*10.0)/log(2.0)); //h should be at most 0.1

R[0][0] = ((\*f)(a, l, t)+(\*f)(b, l, t))\*h\*0.50;

i = 1;

temp2 = 1;

while (i<MAX\_N){

i++;

R[1][0] = 0.0;

for (j=1; j<=temp2; j++)

R[1][0] += (\*f)(a+h\*((double)j-0.50), l, t);

R[1][0] = (R[0][0] + h\*R[1][0])\*0.50;

temp4 = 4.0;

for (j=1; j<i; j++) {

R[1][j] = R[1][j-1] + (R[1][j-1]-R[0][j-1])/(temp4-1.0);

temp4 \*= 4.0;

}

if ((fabs(R[1][i-1]-R[0][i-2])<eps)&&(i>min))

return R[1][i-1];

h \*= 0.50;

temp2 \*= 2;

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

R[0][j] = R[1][j];

}

return R[1][MAX\_N-1];

}

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

double l, double t)

{

#define pi 3.1415926535897932

int n;

double R, p, res;

n = (int)(floor)(b \* t \* 0.50 / pi);

p = 2.0 \* pi / t;

res = b - (double)n \* p;

if (n)

R = Romberg (a, p, f0, eps/(double)n, l, t);

R = R \* (double)n + Romberg( 0.0, res, f0, eps, l, t );

return R/100.0;

}

### 5.2 多项式求根(牛顿法)

/\* 牛顿法解多项式的根

输入：多项式系数c[]，多项式度数n，求在[a,b]间的根

输出：根

要求保证[a,b]间有根

\*/

double fabs( double x )

{

return (x<0)? -x : x;

}

double f(int m, double c[], double x)

{

int i;

double p = c[m];

for (i=m; i>0; i--)

p = p\*x + c[i-1];

return p;

}

int newton(double x0, double \*r,

double c[], double cp[], int n,

double a, double b, double eps)

{

int MAX\_ITERATION = 1000;

int i = 1;

double x1, x2, fp, eps2 = eps/10.0;

x1 = x0;

while (i < MAX\_ITERATION) {

x2 = f(n, c, x1);

fp = f(n-1, cp, x1);

if ((fabs(fp)<0.000000001) && (fabs(x2)>1.0))

return 0;

x2 = x1 - x2/fp;

if (fabs(x1-x2)<eps2) {

if (x2<a || x2>b)

return 0;

\*r = x2;

return 1;

}

x1 = x2;

i++;

}

return 0;

}

double Polynomial\_Root(double c[], int n, double a, double b, double eps)

{

double \*cp;

int i;

double root;

cp = (double \*)calloc(n, sizeof(double));

for (i=n-1; i>=0; i--) {

cp[i] = (i+1)\*c[i+1];

}

if (a>b) {

root = a; a = b; b = root;

}

if ((!newton(a, &root, c, cp, n, a, b, eps)) &&

(!newton(b, &root, c, cp, n, a, b, eps)))

newton((a+b)\*0.5, &root, c, cp, n, a, b, eps);

free(cp);

if (fabs(root)<eps)

return fabs(root);

else

return root;

}

### 5.3 周期性方程(追赶法)

/\* 追赶法解周期性方程

周期性方程定义：| a1 b1 c1 ... | = x1

| a2 b2 c2 ... | = x2

| ... | \* X = ...

| cn-1 ... an-1 bn-1 | = xn-1

| bn cn an | = xn

输入：a[],b[],c[],x[]

输出：求解结果X在x[]中

\*/

void run()

{

c[0] /= b[0]; a[0] /= b[0]; x[0] /= b[0];

for (int i = 1; i < N - 1; i ++) {

double temp = b[i] - a[i] \* c[i - 1];

c[i] /= temp;

x[i] = (x[i] - a[i] \* x[i - 1]) / temp;

a[i] = -a[i] \* a[i - 1] / temp;

}

a[N - 2] = -a[N - 2] - c[N - 2];

for (int i = N - 3; i >= 0; i --) {

a[i] = -a[i] - c[i] \* a[i + 1];

x[i] -= c[i] \* x[i + 1];

}

x[N - 1] -= (c[N - 1] \* x[0] + a[N - 1] \* x[N - 2]);

x[N - 1] /= (c[N - 1] \* a[0] + a[N - 1] \* a[N - 2] + b[N - 1]);

for (int i = N - 2; i >= 0; i --)

x[i] += a[i] \* x[N - 1];

}

## 图论—NP搜索

### 6.1 最大团

//最大团

//返回最大团大小和一个方案,传入图的大小n和邻接阵mat

//mat[i][j]为布尔量

#define MAXN 60

void clique(int n, int\* u, int mat[][MAXN], int size, int& max, int& bb, int\* res, int\* rr, int\* c) {

int i, j, vn, v[MAXN];

if (n) {

if (size + c[u[0]] <= max) return;

for (i = 0; i < n + size - max && i < n; ++ i) {

for (j = i + 1, vn = 0; j < n; ++ j)

if (mat[u[i]][u[j]])

v[vn ++] = u[j];

rr[size] = u[i];

clique(vn, v, mat, size + 1, max, bb, res, rr, c);

if (bb) return;

}

} else if (size > max) {

max = size;

for (i = 0; i < size; ++ i)

res[i] = rr[i];

bb = 1;

}

}

int maxclique(int n, int mat[][MAXN], int \*ret) {

int max = 0, bb, c[MAXN], i, j;

int vn, v[MAXN], rr[MAXN];

for (c[i = n - 1] = 0; i >= 0; -- i) {

for (vn = 0, j = i + 1; j < n; ++ j)

if (mat[i][j])

v[vn ++] = j;

bb = 0;

rr[0] = i;

clique(vn, v, mat, 1, max, bb, ret, rr, c);

c[i] = max;

}

return max;

}

### 6.2 最大团(n<64)(faster)

/\*\*

\* WishingBone's ACM/ICPC Routine Library

\*

\* maximum clique solver

\*/

#include <vector>

using std::vector;

// clique solver calculates both size and consitution of maximum clique

// uses bit operation to accelerate searching

// graph size limit is 63, the graph should be undirected

// can optimize to calculate on each component, and sort on vertex degrees

// can be used to solve maximum independent set

class clique {

public:

static const long long ONE = 1;

static const long long MASK = (1 << 21) - 1;

char\* bits;

int n, size, cmax[63];

long long mask[63], cons;

// initiate lookup table

clique() {

bits = new char[1 << 21];

bits[0] = 0;

for (int i = 1; i < 1 << 21; ++i) bits[i] = bits[i >> 1] + (i & 1);

}

~clique() {

delete bits;

}

// search routine

bool search(int step, int size, long long more, long long con);

// solve maximum clique and return size

int sizeClique(vector<vector<int> >& mat);

// solve maximum clique and return constitution

vector<int> consClique(vector<vector<int> >& mat);

};

// search routine

// step is node id, size is current solution, more is available mask, cons is

constitution mask

bool clique::search(int step, int size, long long more, long long cons) {

if (step >= n) {

// a new solution reached

this->size = size;

this->cons = cons;

return true;

}

long long now = ONE << step;

if ((now & more) > 0) {

long long next = more & mask[step];

if (size + bits[next & MASK] + bits[(next >> 21) & MASK] + bits[next >>

42] >= this->size

&& size + cmax[step] > this->size) {

// the current node is in the clique

if (search(step + 1, size + 1, next, cons | now)) return true;

}

}

long long next = more & ~now;

if (size + bits[next & MASK] + bits[(next >> 21) & MASK] + bits[next >> 42]

> this->size) {

// the current node is not in the clique

if (search(step + 1, size, next, cons)) return true;

}

return false;

}

// solve maximum clique and return size

int clique::sizeClique(vector<vector<int> >& mat) {

n = mat.size();

// generate mask vectors

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

mask[i] = 0;

for (int j = 0; j < n; ++j) if (mat[i][j] > 0) mask[i] |= ONE << j;

}

size = 0;

for (int i = n - 1; i >= 0; --i) {

search(i + 1, 1, mask[i], ONE << i);

cmax[i] = size;

}

return size;

}

// solve maximum clique and return constitution

// calls sizeClique and restore cons

vector<int> clique::consClique(vector<vector<int> >& mat) {

sizeClique(mat);

vector<int> ret;

for (int i = 0; i < n; ++i) if ((cons & (ONE << i)) > 0) ret.push\_back(i);

return ret;

}

## 图论—连通性

### 7.1 无向图关键点(dfs邻接阵)

//无向图的关键点,dfs邻接阵形式,O(n^2)

//返回关键点个数,key[]返回点集

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 110

void search(int n,int mat[][MAXN],int\* dfn,int\* low,int now,int& ret,int\* key,int& cnt,int root,int& rd,int\* bb){

int i;

dfn[now]=low[now]=++cnt;

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

if (mat[now][i]){

if (!dfn[i]){

search(n,mat,dfn,low,i,ret,key,cnt,root,rd,bb);

if (low[i]<low[now])

low[now]=low[i];

if (low[i]>=dfn[now]){

if (now!=root&&!bb[now])

key[ret++]=now,bb[now]=1;

else if(now==root)

rd++;

}

}

else if (dfn[i]<low[now])

low[now]=dfn[i];

}

}

int key\_vertex(int n,int mat[][MAXN],int\* key){

int ret=0,i,cnt,rd,dfn[MAXN],low[MAXN],bb[MAXN];

for (i=0;i<n;dfn[i++]=bb[i]=0);

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

if (!dfn[i]){

rd=0;

search(n,mat,dfn,low,i,ret,key,cnt,i,rd,bb);

if (rd>1&&!bb[i])

key[ret++]=i,bb[i]=1;

}

return ret;

}

### 7.2 无向图关键边(dfs邻接阵)

//无向图的关键边,dfs邻接阵形式,O(n^2)

//返回关键边条数,key[][2]返回边集

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

void search(int n,int mat[][MAXN],int\* dfn,int\* low,int now,int& cnt,int key[][2]){

int i;

for (low[now]=dfn[now],i=0;i<n;i++)

if (mat[now][i]){

if (!dfn[i]){

dfn[i]=dfn[now]+1;

search(n,mat,dfn,low,i,cnt,key);

if (low[i]>dfn[now])

key[cnt][0]=i,key[cnt++][1]=now;

if (low[i]<low[now])

low[now]=low[i];

}

else if (dfn[i]<dfn[now]-1&&dfn[i]<low[now])

low[now]=lev[i];

}

}

int key\_edge(int n,int mat[][MAXN],int key[][2]){

int ret=0,i,dfn[MAXN],low[MAXN];

for (i=0;i<n;dfn[i++]=0);

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

if (!dfn[i])

dfn[i]=1,bridge(n,mat,dfn,low,i,ret,key);

return ret;

}

### 7.3 无向图的块(bfs邻接阵)

//无向图的块,dfs邻接阵形式,O(n^2)

//每产生一个块调用dummy

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

#include <iostream.h>

void dummy(int n,int\* a){

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

cout<<a[i]<<' ';

cout<<endl;

}

void search(int n,int mat[][MAXN],int\* dfn,int\* low,int now,int& cnt,int\* st,int& sp){

int i,m,a[MAXN];

dfn[st[sp++]=now]=low[now]=++cnt;

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

if (mat[now][i]){

if (!dfn[i]){

search(n,mat,dfn,low,i,cnt,st,sp);

if (low[i]<low[now])

low[now]=low[i];

if (low[i]>=dfn[now]){

for (st[sp]=-1,a[0]=now,m=1;st[sp]!=i;a[m++]=st[--sp]);

dummy(m,a);

}

}

else if (dfn[i]<low[now])

low[now]=dfn[i];

}

}

void block(int n,int mat[][MAXN]){

int i,cnt,dfn[MAXN],low[MAXN],st[MAXN],sp=0;

for (i=0;i<n;dfn[i++]=0);

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

if (!dfn[i])

search(n,mat,dfn,low,i,cnt,st,sp);

}

### 7.4 无向图连通分支(dfs/bfs邻接阵)

//无向图连通分支,dfs邻接阵形式,O(n^2)

//返回分支数,id返回1..分支数的值

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

void floodfill(int n,int mat[][MAXN],int\* id,int now,int tag){

int i;

for (id[now]=tag,i=0;i<n;i++)

if (!id[i]&&mat[now][i])

floodfill(n,mat,id,i,tag);

}

int find\_components(int n,int mat[][MAXN],int\* id){

int ret,i;

for (i=0;i<n;id[i++]=0);

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

if (!id[i])

floodfill(n,mat,id,i,++ret);

return ret;

}

//无向图连通分支,bfs邻接阵形式,O(n^2)

//返回分支数,id返回1..分支数的值

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

int find\_components(int n,int mat[][MAXN],int\* id){

int ret,k,i,j,m;

for (k=0;k<n;id[k++]=0);

for (ret=k=0;k<n;k++)

if (!id[k])

for (id[k]=-1,ret++,m=1;m;)

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

if (id[i]==-1)

for (m++,id[i]=ret,j=0;j<n;j++)

if (!id[j]&&mat[i][j])

id[j]=-1;

return ret;

}

### 7.5 有向图强连通分支(dfs/bfs邻接阵)

//有向图强连通分支,dfs邻接阵形式,O(n^2)

//返回分支数,id返回1..分支数的值

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

void search(int n,int mat[][MAXN],int\* dfn,int\* low,int now,int& cnt,int& tag,int\* id,int\* st,int& sp){

int i,j;

dfn[st[sp++]=now]=low[now]=++cnt;

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

if (mat[now][i]){

if (!dfn[i]){

ssearch(n,mat,dfn,low,i,cnt,tag,id,st,sp);

if (low[i]<low[now])

low[now]=low[i];

}

else if (dfn[i]<dfn[now]){

for (j=0;j<sp&&st[j]!=i;j++);

if (j<cnt&&dfn[i]<low[now])

low[now]=dfn[i];

}

}

if (low[now]==dfn[now])

for (tag++;st[sp]!=now;id[st[--sp]]=tag);

}

int find\_components(int n,int mat[][MAXN],int\* id){

int ret=0,i,cnt,sp,st[MAXN],dfn[MAXN],low[MAXN];

for (i=0;i<n;dfn[i++]=0);

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

if (!dfn[i])

search(n,mat,dfn,low,i,cnt,ret,id,st,sp);

return ret;

}

//有向图强连通分支,bfs邻接阵形式,O(n^2)

//返回分支数,id返回1..分支数的值

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

int find\_components(int n,int mat[][MAXN],int\* id){

int ret=0,a[MAXN],b[MAXN],c[MAXN],d[MAXN],i,j,k,t;

for (k=0;k<n;id[k++]=0);

for (k=0;k<n;k++)

if (!id[k]){

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

a[i]=b[i]=c[i]=d[i]=0;

a[k]=b[k]=1;

for (t=1;t;)

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

if (a[i]&&!c[i])

for (c[i]=t=1,j=0;j<n;j++)

if (mat[i][j]&&!a[j])

a[j]=1;

if (b[i]&&!d[i])

for (d[i]=t=1,j=0;j<n;j++)

if (mat[j][i]&&!b[j])

b[j]=1;

}

for (ret++,i=0;i<n;i++)

if (a[i]&b[i])

id[i]=ret;

}

return ret;

}

### 7.6 有向图最小点基(邻接阵)

//有向图最小点基,邻接阵形式,O(n^2)

//返回电集大小和点集

//传入图的大小n和邻接阵mat,不相邻点边权0

//需要调用强连通分支

#define MAXN 100

int base\_vertex(int n,int mat[][MAXN],int\* sets){

int ret=0,id[MAXN],v[MAXN],i,j;

j=find\_components(n,mat,id);

for (i=0;i<j;v[i++]=1);

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

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

if (id[i]!=id[j]&&mat[i][j])

v[id[j]-1]=0;

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

if (v[id[i]-1])

v[id[sets[ret++]=i]-1]=0;

return ret;

}

## 图论—匹配

### 8.1 二分图最大匹配(hungary邻接表)

//二分图最大匹配,hungary算法,邻接表形式,复杂度O(m\*e)

//返回最大匹配数,传入二分图大小m,n和邻接表list(只需一边)

//match1,match2返回一个最大匹配,未匹配顶点match值为-1

#include <string.h>

#define MAXN 310

#define \_clr(x) memset(x,0xff,sizeof(int)\*MAXN)

struct edge\_t{

int from,to;

edge\_t\* next;

};

int hungary(int m,int n,edge\_t\* list[],int\* match1,int\* match2){

int s[MAXN],t[MAXN],p,q,ret=0,i,j,k;edge\_t\* e;

for (\_clr(match1),\_clr(match2),i=0;i<m;ret+=(match1[i++]>=0))

for (\_clr(t),s[p=q=0]=i;p<=q&&match1[i]<0;p++)

for (e=list[k=s[p]];e&&match1[i]<0;e=e->next)

if (t[j=e->to]<0){

s[++q]=match2[j],t[j]=k;

if (s[q]<0)

for (p=j;p>=0;j=p)

match2[j]=k=t[j],p=match1[k],match1[k]=j;

}

return ret;

}

### 8.2 二分图最大匹配(hungary邻接阵)

//二分图最大匹配,hungary算法,邻接阵形式,复杂度O(m\*m\*n)

//返回最大匹配数,传入二分图大小m,n和邻接阵mat,非零元素表示有边

//match1,match2返回一个最大匹配,未匹配顶点match值为-1

#include <string.h>

#define MAXN 310

#define \_clr(x) memset(x,0xff,sizeof(int)\*MAXN)

int hungary(int m,int n,int mat[][MAXN],int\* match1,int\* match2){

int s[MAXN],t[MAXN],p,q,ret=0,i,j,k;

for (\_clr(match1),\_clr(match2),i=0;i<m;ret+=(match1[i++]>=0))

for (\_clr(t),s[p=q=0]=i;p<=q&&match1[i]<0;p++)

for (k=s[p],j=0;j<n&&match1[i]<0;j++)

if (mat[k][j]&&t[j]<0){

s[++q]=match2[j],t[j]=k;

if (s[q]<0)

for (p=j;p>=0;j=p)

match2[j]=k=t[j],p=match1[k],match1[k]=j;

}

return ret;

}

### 8.3 二分图最大匹配(hungary正向表)

//二分图最大匹配,hungary算法,正向表形式,复杂度O(m\*e)

//返回最大匹配数,传入二分图大小m,n和正向表list,buf(只需一边)

//match1,match2返回一个最大匹配,未匹配顶点match值为-1

#include <string.h>

#define MAXN 310

#define \_clr(x) memset(x,0xff,sizeof(int)\*MAXN)

int hungary(int m,int n,int\* list,int\* buf,int\* match1,int\* match2){

int s[MAXN],t[MAXN],p,q,ret=0,i,j,k,l;

for (\_clr(match1),\_clr(match2),i=0;i<m;ret+=(match1[i++]>=0))

for (\_clr(t),s[p=q=0]=i;p<=q&&match1[i]<0;p++)

for (l=list[k=s[p]];l<list[k+1]&&match1[i]<0;l++)

if (t[j=buf[l]]<0){

s[++q]=match2[j],t[j]=k;

if (s[q]<0)

for (p=j;p>=0;j=p)

match2[j]=k=t[j],p=match1[k],match1[k]=j;

}

return ret;

}

### 8.4二分图最佳匹配(kuhn\_munkras邻接阵)

//二分图最佳匹配,kuhn munkras算法,邻接阵形式,复杂度O(m\*m\*n)

//返回最佳匹配值,传入二分图大小m,n和邻接阵mat,表示权值

//match1,match2返回一个最佳匹配,未匹配顶点match值为-1

//一定注意m<=n,否则循环无法终止

//最小权匹配可将权值取相反数

#include <string.h>

#define MAXN 310

#define inf 1000000000

#define \_clr(x) memset(x,0xff,sizeof(int)\*n)

int kuhn\_munkras(int m,int n,int mat[][MAXN],int\* match1,int\* match2){

int s[MAXN],t[MAXN],l1[MAXN],l2[MAXN],p,q,ret=0,i,j,k;

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

for (l1[i]=-inf,j=0;j<n;j++)

l1[i]=mat[i][j]>l1[i]?mat[i][j]:l1[i];

for (i=0;i<n;l2[i++]=0);

for (\_clr(match1),\_clr(match2),i=0;i<m;i++){

for (\_clr(t),s[p=q=0]=i;p<=q&&match1[i]<0;p++)

for (k=s[p],j=0;j<n&&match1[i]<0;j++)

if (l1[k]+l2[j]==mat[k][j]&&t[j]<0){

s[++q]=match2[j],t[j]=k;

if (s[q]<0)

for (p=j;p>=0;j=p)

match2[j]=k=t[j],p=match1[k],match1[k]=j;

}

if (match1[i]<0){

for (i--,p=inf,k=0;k<=q;k++)

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

if (t[j]<0&&l1[s[k]]+l2[j]-mat[s[k]][j]<p)

p=l1[s[k]]+l2[j]-mat[s[k]][j];

for (j=0;j<n;l2[j]+=t[j]<0?0:p,j++);

for (k=0;k<=q;l1[s[k++]]-=p);

}

}

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

ret+=mat[i][match1[i]];

return ret;

}

### 8.5 一般图匹配(邻接表)

//一般图最大匹配,邻接表形式,复杂度O(n\*e)

//返回匹配顶点对数,match返回匹配,未匹配顶点match值为-1

//传入图的顶点数n和邻接表list

#define MAXN 100

struct edge\_t{

int from,to;

edge\_t\* next;

};

int aug(int n,edge\_t\* list[],int\* match,int\* v,int now){

int t,ret=0;edge\_t\* e;

v[now]=1;

for (e=list[now];e;e=e->next)

if (!v[t=e->to]){

if (match[t]<0)

match[now]=t,match[t]=now,ret=1;

else{

v[t]=1;

if (aug(n,list,match,v,match[t]))

match[now]=t,match[t]=now,ret=1;

v[t]=0;

}

if (ret)

break;

}

v[now]=0;

return ret;

}

int graph\_match(int n,edge\_t\* list[],int\* match){

int v[MAXN],i,j;

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

v[i]=0,match[i]=-1;

for (i=0,j=n;i<n&&j>=2;)

if (match[i]<0&&aug(n,list,match,v,i))

i=0,j-=2;

else

i++;

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

j+=(match[i]>=0);

return j/2;

}

### 8.6 一般图匹配(邻接阵)

//一般图最大匹配,邻接阵形式,复杂度O(n^3)

//返回匹配顶点对数,match返回匹配,未匹配顶点match值为-1

//传入图的顶点数n和邻接阵mat

#define MAXN 100

int aug(int n,int mat[][MAXN],int\* match,int\* v,int now){

int i,ret=0;

v[now]=1;

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

if (!v[i]&&mat[now][i]){

if (match[i]<0)

match[now]=i,match[i]=now,ret=1;

else{

v[i]=1;

if (aug(n,mat,match,v,match[i]))

match[now]=i,match[i]=now,ret=1;

v[i]=0;

}

if (ret)

break;

}

v[now]=0;

return ret;

}

int graph\_match(int n,int mat[][MAXN],int\* match){

int v[MAXN],i,j;

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

v[i]=0,match[i]=-1;

for (i=0,j=n;i<n&&j>=2;)

if (match[i]<0&&aug(n,mat,match,v,i))

i=0,j-=2;

else

i++;

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

j+=(match[i]>=0);

return j/2;

}

### 8.7 一般图匹配(正向表)

//一般图最大匹配,正向表形式,复杂度O(n\*e)

//返回匹配顶点对数,match返回匹配,未匹配顶点match值为-1

//传入图的顶点数n和正向表list,buf

#define MAXN 100

int aug(int n,int\* list,int\* buf,int\* match,int\* v,int now){

int i,t,ret=0;

v[now]=1;

for (i=list[now];i<list[now+1];i++)

if (!v[t=buf[i]]){

if (match[t]<0)

match[now]=t,match[t]=now,ret=1;

else{

v[t]=1;

if (aug(n,list,buf,match,v,match[t]))

match[now]=t,match[t]=now,ret=1;

v[t]=0;

}

if (ret)

break;

}

v[now]=0;

return ret;

}

int graph\_match(int n,int\* list,int\* buf,int\* match){

int v[MAXN],i,j;

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

v[i]=0,match[i]=-1;

for (i=0,j=n;i<n&&j>=2;)

if (match[i]<0&&aug(n,list,buf,match,v,i))

i=0,j-=2;

else

i++;

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

j+=(match[i]>=0);

return j/2;

}

## 图论—网络流

### 9.1 最大流(邻接阵)

//求网络最大流,邻接阵形式

//返回最大流量,flow返回每条边的流量

//传入网络节点数n,容量mat,源点source,汇点sink

#define MAXN 100

#define inf 1000000000

int max\_flow(int n,int mat[][MAXN],int source,int sink,int flow[][MAXN]){

int pre[MAXN],que[MAXN],d[MAXN],p,q,t,i,j;

if (source==sink) return inf;

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

for (j=0;j<n;flow[i][j++]=0);

for (;;){

for (i=0;i<n;pre[i++]=0);

pre[t=source]=source+1,d[t]=inf;

for (p=q=0;p<=q&&!pre[sink];t=que[p++])

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

if (!pre[i]&&j=mat[t][i]-flow[t][i])

pre[que[q++]=i]=t+1,d[i]=d[t]<j?d[t]:j;

else if (!pre[i]&&j=flow[i][t])

pre[que[q++]=i]=-t-1,d[i]=d[t]<j?d[t]:j;

if (!pre[sink]) break;

for (i=sink;i!=source;)

if (pre[i]>0)

flow[pre[i]-1][i]+=d[sink],i=pre[i]-1;

else

flow[i][-pre[i]-1]-=d[sink],i=-pre[i]-1;

}

for (j=i=0;i<n;j+=flow[source][i++]);

return j;

}

### 9.2 上下界最大流(邻接阵)

//求上下界网络最大流,邻接阵形式

//返回最大流量,-1表示无可行流,flow返回每条边的流量

//传入网络节点数n,容量mat,流量下界bf,源点source,汇点sink

//MAXN应比最大结点数多2,无可行流返回-1时mat未复原!

#define MAXN 100

#define inf 1000000000

int limit\_max\_flow(int n,int mat[][MAXN],int bf[][MAXN],int source,int sink,int flow[][MAXN]){

int i,j,sk,ks;

if (source==sink) return inf;

for (mat[n][n+1]=mat[n+1][n]=mat[n][n]=mat[n+1][n+1]=i=0;i<n;i++)

for (mat[n][i]=mat[i][n]=mat[n+1][i]=mat[i][n+1]=j=0;j<n;j++)

mat[i][j]-=bf[i][j],mat[n][i]+=bf[j][i],mat[i][n+1]+=bf[i][j];

sk=mat[source][sink],ks=mat[sink][source],mat[source][sink]=mat[sink][source]=inf;

for (i=0;i<n+2;i++)

for (j=0;j<n+2;flow[i][j++]=0);

\_max\_flow(n+2,mat,n,n+1,flow);

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

if (flow[n][i]<mat[n][i]) return -1;

flow[source][sink]=flow[sink][source]=0,mat[source][sink]=sk,mat[sink][source]=ks;

\_max\_flow(n,mat,source,sink,flow);

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

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

mat[i][j]+=bf[i][j],flow[i][j]+=bf[i][j];

for (j=i=0;i<n;j+=flow[source][i++]);

return j;

}

### 9.3 上下界最小流(邻接阵)

//求上下界网络最小流,邻接阵形式

//返回最大流量,-1表示无可行流,flow返回每条边的流量

//传入网络节点数n,容量mat,流量下界bf,源点source,汇点sink

//MAXN应比最大结点数多2,无可行流返回-1时mat未复原!

#define MAXN 100

#define inf 1000000000

int limit\_min\_flow(int n,int mat[][MAXN],int bf[][MAXN],int source,int sink,int flow[][MAXN]){

int i,j,sk,ks;

if (source==sink) return inf;

for (mat[n][n+1]=mat[n+1][n]=mat[n][n]=mat[n+1][n+1]=i=0;i<n;i++)

for (mat[n][i]=mat[i][n]=mat[n+1][i]=mat[i][n+1]=j=0;j<n;j++)

mat[i][j]-=bf[i][j],mat[n][i]+=bf[j][i],mat[i][n+1]+=bf[i][j];

sk=mat[source][sink],ks=mat[sink][source],mat[source][sink]=mat[sink][source]=inf;

for (i=0;i<n+2;i++)

for (j=0;j<n+2;flow[i][j++]=0);

\_max\_flow(n+2,mat,n,n+1,flow);

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

if (flow[n][i]<mat[n][i]) return -1;

flow[source][sink]=flow[sink][source]=0,mat[source][sink]=sk,mat[sink][source]=ks;

\_max\_flow(n,mat,sink,source,flow);

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

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

mat[i][j]+=bf[i][j],flow[i][j]+=bf[i][j];

for (j=i=0;i<n;j+=flow[source][i++]);

return j;

}

### 9.4 最大流无流量(邻接阵)

//求网络最大流,邻接阵形式

//返回最大流量

//传入网络节点数n,容量mat,源点source,汇点sink

//注意mat矩阵被修改

#define MAXN 100

#define inf 1000000000

int max\_flow(int n,int mat[][MAXN],int source,int sink){

int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;

for (;;){

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

v[i]=c[i]=0;

for (c[source]=inf;;){

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

if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))

j=i;

if (j<0) return ret;

if (j==sink) break;

for (v[j]=1,i=0;i<n;i++)

if (mat[j][i]>c[i]&&c[j]>c[i])

c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;

}

for (ret+=j=c[i=sink];i!=source;i=p[i])

mat[p[i]][i]-=j,mat[i][p[i]]+=j;

}

}

### 9.5 最小费用最大流(邻接阵)

//求网络最小费用最大流,邻接阵形式

//返回最大流量,flow返回每条边的流量,netcost返回总费用

//传入网络节点数n,容量mat,单位费用cost,源点source,汇点sink

#define MAXN 100

#define inf 1000000000

int min\_cost\_max\_flow(int n,int mat[][MAXN],int cost[][MAXN],int source,int sink,int flow[][MAXN],int& netcost){

int pre[MAXN],min[MAXN],d[MAXN],i,j,t,tag;

if (source==sink) return inf;

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

for (j=0;j<n;flow[i][j++]=0);

for (netcost=0;;){

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

pre[i]=0,min[i]=inf;

for (pre[source]=source+1,min[source]=0,d[source]=inf,tag=1;tag;)

for (tag=t=0;t<n;t++)

if (d[t])

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

if (j=mat[t][i]-flow[t][i]&&min[t]+cost[t][i]<min[i])

tag=1,min[i]=min[t]+cost[t][i],pre[i]=t+1,d[i]=d[t]<j?d[t]:j;

else if (j=flow[i][t]&&min[t]<inf&&min[t]-cost[i][t]<min[i])

tag=1,min[i]=min[t]-cost[i][t],pre[i]=-t-1,d[i]=d[t]<j?d[t]:j;

if (!pre[sink]) break;

for (netcost+=min[sink]\*d[i=sink];i!=source;)

if (pre[i]>0)

flow[pre[i]-1][i]+=d[sink],i=pre[i]-1;

else

flow[i][-pre[i]-1]-=d[sink],i=-pre[i]-1;

}

for (j=i=0;i<n;j+=flow[source][i++]);

return j;

}

## 图论—应用

## 10.1 欧拉回路(邻接阵)

//求欧拉回路或欧拉路,邻接阵形式,复杂度O(n^2)

//返回路径长度,path返回路径(有向图时得到的是反向路径)

//传入图的大小n和邻接阵mat,不相邻点边权0

//可以有自环与重边,分为无向图和有向图

#define MAXN 100

void find\_path\_u(int n,int mat[][MAXN],int now,int& step,int\* path){

int i;

for (i=n-1;i>=0;i--)

while (mat[now][i]){

mat[now][i]--,mat[i][now]--;

find\_path\_u(n,mat,i,step,path);

}

path[step++]=now;

}

void find\_path\_d(int n,int mat[][MAXN],int now,int& step,int\* path){

int i;

for (i=n-1;i>=0;i--)

while (mat[now][i]){

mat[now][i]--;

find\_path\_d(n,mat,i,step,path);

}

path[step++]=now;

}

int euclid\_path(int n,int mat[][MAXN],int start,int\* path){

int ret=0;

find\_path\_u(n,mat,start,ret,path);

// find\_path\_d(n,mat,start,ret,path);

return ret;

}

## 10.2 树的前序表转化

//将用边表示的树转化为前序表示的树

//传入节点数n和邻接表list[],邻接表必须是双向的,会在函数中释放

//pre[]返回前序表,map[]返回前序表中的节点到原来节点的映射

#define MAXN 10000

struct node{

int to;

node\* next;

};

void prenode(int n,node\* list[],int\* pre,int\* map,int\* v,int now,int last,int& id){

node\* t;

int p=id++;

for (v[map[p]=now]=1,pre[p]=last;list[now];){

t=list[now],list[now]=t->next;

if (!v[t->to])

prenode(n,list,pre,map,v,t->to,p,id);

}

}

void makepre(int n,node\* list[],int\* pre,int\* map){

int v[MAXN],id=0,i;

for (i=0;i<n;v[i++]=0);

prenode(n,list,pre,map,v,0,-1,id);

}

## 10.3 树的优化算法

//最大顶点独立集

int max\_node\_independent(int n,int\* pre,int\* set){

int c[MAXN],i,ret=0;

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

c[i]=set[i]=0;

for (i=n-1;i>=0;i--)

if (!c[i]){

set[i]=1;

if (pre[i]!=-1)

c[pre[i]]=1;

ret++;

}

return ret;

}

//最大边独立集

int max\_edge\_independent(int n,int\* pre,int\* set){

int c[MAXN],i,ret=0;

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

c[i]=set[i]=0;

for (i=n-1;i>=0;i--)

if (!c[i]&&pre[i]!=-1&&!c[pre[i]]){

set[i]=1;

c[pre[i]]=1;

ret++;

}

return ret;

}

//最小顶点覆盖集

int min\_node\_cover(int n,int\* pre,int\* set){

int c[MAXN],i,ret=0;

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

c[i]=set[i]=0;

for (i=n-1;i>=0;i--)

if (!c[i]&&pre[i]!=-1&&!c[pre[i]]){

set[i]=1;

c[pre[i]]=1;

ret++;

}

return ret;

}

//最小顶点支配集

int min\_node\_dominant(int n,int\* pre,int\* set){

int c[MAXN],i,ret=0;

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

c[i]=set[i]=0;

for (i=n-1;i>=0;i--)

if (!c[i]&&(pre[i]==-1||!set[pre[i]])){

if (pre[i]!=-1){

set[pre[i]]=1;

c[pre[i]]=1;

if (pre[pre[i]]!=-1)

c[pre[pre[i]]]=1;

}

else

set[i]=1;

ret++;

}

return ret;

}

## 10.4 拓扑排序(邻接阵)

//拓扑排序,邻接阵形式,复杂度O(n^2)

//如果无法完成排序,返回0,否则返回1,ret返回有序点列

//传入图的大小n和邻接阵mat,不相邻点边权0

#define MAXN 100

int toposort(int n,int mat[][MAXN],int\* ret){

int d[MAXN],i,j,k;

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

for (d[i]=j=0;j<n;d[i]+=mat[j++][i]);

for (k=0;k<n;ret[k++]=i){

for (i=0;d[i]&&i<n;i++);

if (i==n)

return 0;

for (d[i]=-1,j=0;j<n;j++)

d[j]-=mat[i][j];

}

return 1;

}

## 10.5 最佳边割集

//最佳边割集

#define MAXN 100

#define inf 1000000000

int max\_flow(int n,int mat[][MAXN],int source,int sink){

int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;

for (;;){

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

v[i]=c[i]=0;

for (c[source]=inf;;){

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

if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))

j=i;

if (j<0) return ret;

if (j==sink) break;

for (v[j]=1,i=0;i<n;i++)

if (mat[j][i]>c[i]&&c[j]>c[i])

c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;

}

for (ret+=j=c[i=sink];i!=source;i=p[i])

mat[p[i]][i]-=j,mat[i][p[i]]+=j;

}

}

int best\_edge\_cut(int n,int mat[][MAXN],int source,int sink,int set[][2],int& mincost){

int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,l,ret=0,last;

if (source==sink)

return -1;

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

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

m0[i][j]=mat[i][j];

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

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

m[i][j]=m0[i][j];

mincost=last=max\_flow(n,m,source,sink);

for (k=0;k<n&&last;k++)

for (l=0;l<n&&last;l++)

if (m0[k][l]){

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

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

m[i][j]=m0[i][j];

m[k][l]=0;

if (max\_flow(n,m,source,sink)==last-mat[k][l]){

set[ret][0]=k;

set[ret++][1]=l;

m0[k][l]=0;

last-=mat[k][l];

}

}

return ret;

}

## 10.6 最佳点割集

//最佳顶点割集

#define MAXN 100

#define inf 1000000000

int max\_flow(int n,int mat[][MAXN],int source,int sink){

int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;

for (;;){

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

v[i]=c[i]=0;

for (c[source]=inf;;){

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

if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))

j=i;

if (j<0) return ret;

if (j==sink) break;

for (v[j]=1,i=0;i<n;i++)

if (mat[j][i]>c[i]&&c[j]>c[i])

c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;

}

for (ret+=j=c[i=sink];i!=source;i=p[i])

mat[p[i]][i]-=j,mat[i][p[i]]+=j;

}

}

int best\_vertex\_cut(int n,int mat[][MAXN],int\* cost,int source,int sink,int\* set,int& mincost){

int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,ret=0,last;

if (source==sink||mat[source][sink])

return -1;

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

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

m0[i][j]=0;

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

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

if (mat[i][j])

m0[i][n+j]=inf;

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

m0[n+i][i]=cost[i];

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

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

m[i][j]=m0[i][j];

mincost=last=max\_flow(n+n,m,source,n+sink);

for (k=0;k<n&&last;k++)

if (k!=source&&k!=sink){

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

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

m[i][j]=m0[i][j];

m[n+k][k]=0;

if (max\_flow(n+n,m,source,n+sink)==last-cost[k]){

set[ret++]=k;

m0[n+k][k]=0;

last-=cost[k];

}

}

return ret;

}

## 10.7 最小边割集

//最小边割集

#define MAXN 100

#define inf 1000000000

int max\_flow(int n,int mat[][MAXN],int source,int sink){

int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;

for (;;){

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

v[i]=c[i]=0;

for (c[source]=inf;;){

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

if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))

j=i;

if (j<0) return ret;

if (j==sink) break;

for (v[j]=1,i=0;i<n;i++)

if (mat[j][i]>c[i]&&c[j]>c[i])

c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;

}

for (ret+=j=c[i=sink];i!=source;i=p[i])

mat[p[i]][i]-=j,mat[i][p[i]]+=j;

}

}

int min\_edge\_cut(int n,int mat[][MAXN],int source,int sink,int set[][2]){

int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,l,ret=0,last;

if (source==sink)

return -1;

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

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

m0[i][j]=(mat[i][j]!=0);

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

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

m[i][j]=m0[i][j];

last=max\_flow(n,m,source,sink);

for (k=0;k<n&&last;k++)

for (l=0;l<n&&last;l++)

if (m0[k][l]){

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

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

m[i][j]=m0[i][j];

m[k][l]=0;

if (max\_flow(n,m,source,sink)<last){

set[ret][0]=k;

set[ret++][1]=l;

m0[k][l]=0;

last--;

}

}

return ret;

}

## 10.8 最小点割集

//最小顶点割集

#define MAXN 100

#define inf 1000000000

int max\_flow(int n,int mat[][MAXN],int source,int sink){

int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;

for (;;){

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

v[i]=c[i]=0;

for (c[source]=inf;;){

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

if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))

j=i;

if (j<0) return ret;

if (j==sink) break;

for (v[j]=1,i=0;i<n;i++)

if (mat[j][i]>c[i]&&c[j]>c[i])

c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;

}

for (ret+=j=c[i=sink];i!=source;i=p[i])

mat[p[i]][i]-=j,mat[i][p[i]]+=j;

}

}

int min\_vertex\_cut(int n,int mat[][MAXN],int source,int sink,int\* set){

int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,ret=0,last;

if (source==sink||mat[source][sink])

return -1;

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

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

m0[i][j]=0;

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

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

if (mat[i][j])

m0[i][n+j]=inf;

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

m0[n+i][i]=1;

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

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

m[i][j]=m0[i][j];

last=max\_flow(n+n,m,source,n+sink);

for (k=0;k<n&&last;k++)

if (k!=source&&k!=sink){

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

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

m[i][j]=m0[i][j];

m[n+k][k]=0;

if (max\_flow(n+n,m,source,n+sink)<last){

set[ret++]=k;

m0[n+k][k]=0;

last--;

}

}

return ret;

}

## 10.9 最小路径覆盖

//最小路径覆盖,O(n^3)

//求解最小的路径覆盖图中所有点,有向图无向图均适用

//注意此问题等价二分图最大匹配,可以用邻接表或正向表减小复杂度

//返回最小路径条数,pre返回前指针(起点-1),next返回后指针(终点-1)

#include <string.h>

#define MAXN 310

#define \_clr(x) memset(x,0xff,sizeof(int)\*n)

int hungary(int n,int mat[][MAXN],int\* match1,int\* match2){

int s[MAXN],t[MAXN],p,q,ret=0,i,j,k;

for (\_clr(match1),\_clr(match2),i=0;i<n;ret+=(match1[i++]>=0))

for (\_clr(t),s[p=q=0]=i;p<=q&&match1[i]<0;p++)

for (k=s[p],j=0;j<n&&match1[i]<0;j++)

if (mat[k][j]&&t[j]<0){

s[++q]=match2[j],t[j]=k;

if (s[q]<0)

for (p=j;p>=0;j=p)

match2[j]=k=t[j],p=match1[k],match1[k]=j;

}

return ret;

}

inline int path\_cover(int n,int mat[][MAXN],int\* pre,int\* next){

return n-hungary(n,mat,next,pre);

}

## 图论—支撑树

### 11.1 最小生成树(kruskal邻接表)

//无向图最小生成树,kruskal算法,邻接表形式,复杂度O(mlogm)

//返回最小生成树的长度,传入图的大小n和邻接表list

//可更改边权的类型,edge[][2]返回树的构造,用边集表示

//如果图不连通,则对各连通分支构造最小生成树,返回总长度

#include <string.h>

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

struct edge\_t{

int from,to;

elem\_t len;

edge\_t\* next;

};

#define \_ufind\_run(x) for(;p[t=x];x=p[x],p[t]=(p[x]?p[x]:x))

#define \_run\_both \_ufind\_run(i);\_ufind\_run(j)

struct ufind{

int p[MAXN],t;

void init(){memset(p,0,sizeof(p));}

void set\_friend(int i,int j){\_run\_both;p[i]=(i==j?0:j);}

int is\_friend(int i,int j){\_run\_both;return i==j&&i;}

};

#define \_cp(a,b) ((a).len<(b).len)

struct heap\_t{int a,b;elem\_t len;};

struct minheap{

heap\_t h[MAXN\*MAXN];

int n,p,c;

void init(){n=0;}

void ins(heap\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(heap\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

elem\_t kruskal(int n,edge\_t\* list[],int edge[][2]){

ufind u;minheap h;

edge\_t\* t;heap\_t e;

elem\_t ret=0;int i,m=0;

u.init(),h.init();

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

for (t=list[i];t;t=t->next)

if (i<t->to)

e.a=i,e.b=t->to,e.len=t->len,h.ins(e);

while (m<n-1&&h.del(e))

if (!u.is\_friend(e.a+1,e.b+1))

edge[m][0]=e.a,edge[m][1]=e.b,ret+=e.len,u.set\_friend(e.a+1,e.b+1);

return ret;

}

### 11.2 最小生成树(kruskal正向表)

//无向图最小生成树,kruskal算法,正向表形式,复杂度O(mlogm)

//返回最小生成树的长度,传入图的大小n和正向表list,buf

//可更改边权的类型,edge[][2]返回树的构造,用边集表示

//如果图不连通,则对各连通分支构造最小生成树,返回总长度

#include <string.h>

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

struct edge\_t{

int to;

elem\_t len;

};

#define \_ufind\_run(x) for(;p[t=x];x=p[x],p[t]=(p[x]?p[x]:x))

#define \_run\_both \_ufind\_run(i);\_ufind\_run(j)

struct ufind{

int p[MAXN],t;

void init(){memset(p,0,sizeof(p));}

void set\_friend(int i,int j){\_run\_both;p[i]=(i==j?0:j);}

int is\_friend(int i,int j){\_run\_both;return i==j&&i;}

};

#define \_cp(a,b) ((a).len<(b).len)

struct heap\_t{int a,b;elem\_t len;};

struct minheap{

heap\_t h[MAXN\*MAXN];

int n,p,c;

void init(){n=0;}

void ins(heap\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(heap\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

elem\_t kruskal(int n,int\* list,edge\_t\* buf,int edge[][2]){

ufind u;minheap h;

heap\_t e;elem\_t ret=0;

int i,j,m=0;

u.init(),h.init();

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

for (j=list[i];j<list[i+1];j++)

if (i<buf[j].to)

e.a=i,e.b=buf[j].to,e.len=buf[j].len,h.ins(e);

while (m<n-1&&h.del(e))

if (!u.is\_friend(e.a+1,e.b+1))

edge[m][0]=e.a,edge[m][1]=e.b,ret+=e.len,u.set\_friend(e.a+1,e.b+1);

return ret;

}

### 11.3 最小生成树(prim+binary\_heap邻接表)

//无向图最小生成树,prim算法+二分堆,邻接表形式,复杂度O(mlogm)

//返回最小生成树的长度,传入图的大小n和邻接表list

//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre值为-1

//必须保证图的连通的!

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

struct edge\_t{

int from,to;

elem\_t len;

edge\_t\* next;

};

#define \_cp(a,b) ((a).d<(b).d)

struct heap\_t{elem\_t d;int v;};

struct heap{

heap\_t h[MAXN\*MAXN];

int n,p,c;

void init(){n=0;}

void ins(heap\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(heap\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

elem\_t prim(int n,edge\_t\* list[],int\* pre){

heap h;

elem\_t min[MAXN],ret=0;

edge\_t\* t;heap\_t e;

int v[MAXN],i;

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

min[i]=inf,v[i]=0,pre[i]=-1;

h.init();e.v=0,e.d=0,h.ins(e);

while (h.del(e))

if (!v[e.v])

for (v[e.v]=1,ret+=e.d,t=list[e.v];t;t=t->next)

if (!v[t->to]&&t->len<min[t->to])

pre[t->to]=t->from,min[e.v=t->to]=e.d=t->len,h.ins(e);

return ret;

}

### 11.4 最小生成树(prim+binary\_heap正向表)

//无向图最小生成树,prim算法+二分堆,正向表形式,复杂度O(mlogm)

//返回最小生成树的长度,传入图的大小n和正向表list,buf

//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre值为-1

//必须保证图的连通的!

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

struct edge\_t{

int to;

elem\_t len;

};

#define \_cp(a,b) ((a).d<(b).d)

struct heap\_t{elem\_t d;int v;};

struct heap{

heap\_t h[MAXN\*MAXN];

int n,p,c;

void init(){n=0;}

void ins(heap\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(heap\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

elem\_t prim(int n,int\* list,edge\_t\* buf,int\* pre){

heap h;heap\_t e;

elem\_t min[MAXN],ret=0;

int v[MAXN],i,j;

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

min[i]=inf,v[i]=0,pre[i]=-1;

h.init();e.v=0,e.d=0,h.ins(e);

while (h.del(e))

if (!v[i=e.v])

for (v[i]=1,ret+=e.d,j=list[i];j<list[i+1];j++)

if (!v[buf[j].to]&&buf[j].len<min[buf[j].to])

pre[buf[j].to]=i,min[e.v=buf[j].to]=e.d=buf[j].len,h.ins(e);

return ret;

}

### 11.5 最小生成树(prim+mapped\_heap邻接表)

//无向图最小生成树,prim算法+映射二分堆,邻接表形式,复杂度O(mlogn)

//返回最小生成树的长度,传入图的大小n和邻接表list

//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre值为-1

//必须保证图的连通的!

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

struct edge\_t{

int from,to;

elem\_t len;

edge\_t\* next;

};

#define \_cp(a,b) ((a)<(b))

struct heap{

elem\_t h[MAXN+1];

int ind[MAXN+1],map[MAXN+1],n,p,c;

void init(){n=0;}

void ins(int i,elem\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

h[map[ind[p]=i]=p]=e;

}

int del(int i,elem\_t& e){

i=map[i];if (i<1||i>n) return 0;

for (e=h[p=i];p>1;h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

for (c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

int delmin(int& i,elem\_t& e){

if (n<1) return 0;i=ind[1];

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

};

elem\_t prim(int n,edge\_t\* list[],int\* pre){

heap h;

elem\_t min[MAXN],ret=0,e;

edge\_t\* t;

int v[MAXN],i;

for (h.init(),i=0;i<n;i++)

min[i]=(i?inf:0),v[i]=0,pre[i]=-1,h.ins(i,min[i]);

while (h.delmin(i,e))

for (v[i]=1,ret+=e,t=list[i];t;t=t->next)

if (!v[t->to]&&t->len<min[t->to])

pre[t->to]=t->from,h.del(t->to,e),h.ins(t->to,min[t->to]=t->len);

return ret;

}

### 11.6 最小生成树(prim+mapped\_heap正向表)

//无向图最小生成树,prim算法+映射二分堆,正向表形式,复杂度O(mlogn)

//返回最小生成树的长度,传入图的大小n和正向表list,buf

//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre值为-1

//必须保证图的连通的!

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

struct edge\_t{

int to;

elem\_t len;

};

#define \_cp(a,b) ((a)<(b))

struct heap{

elem\_t h[MAXN+1];

int ind[MAXN+1],map[MAXN+1],n,p,c;

void init(){n=0;}

void ins(int i,elem\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

h[map[ind[p]=i]=p]=e;

}

int del(int i,elem\_t& e){

i=map[i];if (i<1||i>n) return 0;

for (e=h[p=i];p>1;h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

for (c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

int delmin(int& i,elem\_t& e){

if (n<1) return 0;i=ind[1];

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

};

elem\_t prim(int n,int\* list,edge\_t\* buf,int\* pre){

heap h;

elem\_t min[MAXN],ret=0,e;

int v[MAXN],i,j;

for (h.init(),i=0;i<n;i++)

min[i]=(i?inf:0),v[i]=0,pre[i]=-1,h.ins(i,min[i]);

while (h.delmin(i,e))

for (v[i]=1,ret+=e,j=list[i];j<list[i+1];j++)

if (!v[buf[j].to]&&buf[j].len<min[buf[j].to])

pre[buf[j].to]=i,h.del(buf[j].to,e),h.ins(buf[j].to,min[buf[j].to]=buf[j].len);

return ret;

}

### 11.7 最小生成树(prim邻接阵)

//无向图最小生成树,prim算法,邻接阵形式,复杂度O(n^2)

//返回最小生成树的长度,传入图的大小n和邻接阵mat,不相邻点边权inf

//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre值为-1

//必须保证图的连通的!

#define MAXN 200

#define inf 1000000000

typedef double elem\_t;

elem\_t prim(int n,elem\_t mat[][MAXN],int\* pre){

elem\_t min[MAXN],ret=0;

int v[MAXN],i,j,k;

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

min[i]=inf,v[i]=0,pre[i]=-1;

for (min[j=0]=0;j<n;j++){

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

if (!v[i]&&(k==-1||min[i]<min[k]))

k=i;

for (v[k]=1,ret+=min[k],i=0;i<n;i++)

if (!v[i]&&mat[k][i]<min[i])

min[i]=mat[pre[i]=k][i];

}

return ret;

}

### 11.8 最小树形图(邻接阵)

//多源最小树形图,edmonds算法,邻接阵形式,复杂度O(n^3)

//返回最小生成树的长度,构造失败返回负值

//传入图的大小n和邻接阵mat,不相邻点边权inf

//可更改边权的类型,pre[]返回树的构造,用父结点表示

//传入时pre[]数组清零,用-1标出源点

#include <string.h>

#define MAXN 120

#define inf 1000000000

typedef int elem\_t;

elem\_t edmonds(int n,elem\_t mat[][MAXN\*2],int\* pre){

elem\_t ret=0;

int c[MAXN\*2][MAXN\*2],l[MAXN\*2],p[MAXN\*2],m=n,t,i,j,k;

for (i=0;i<n;l[i]=i,i++);

do{

memset(c,0,sizeof(c)),memset(p,0xff,sizeof(p));

for (t=m,i=0;i<m;c[i][i]=1,i++);

for (i=0;i<t;i++)

if (l[i]==i&&pre[i]!=-1){

for (j=0;j<m;j++)

if (l[j]==j&&i!=j&&mat[j][i]<inf&&(p[i]==-1||mat[j][i]<mat[p[i]][i]))

p[i]=j;

if ((pre[i]=p[i])==-1)

return -1;

if (c[i][p[i]]){

for (j=0;j<=m;mat[j][m]=mat[m][j]=inf,j++);

for (k=i;l[k]!=m;l[k]=m,k=p[k])

for (j=0;j<m;j++)

if (l[j]==j){

if (mat[j][k]-mat[p[k]][k]<mat[j][m])

mat[j][m]=mat[j][k]-mat[p[k]][k];

if (mat[k][j]<mat[m][j])

mat[m][j]=mat[k][j];

}

c[m][m]=1,l[m]=m,m++;

}

for (j=0;j<m;j++)

if (c[i][j])

for (k=p[i];k!=-1&&l[k]==k;c[k][j]=1,k=p[k]);

}

}

while (t<m);

for (;m-->n;pre[k]=pre[m])

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

if (l[i]==m){

for (j=0;j<m;j++)

if (pre[j]==m&&mat[i][j]==mat[m][j])

pre[j]=i;

if (mat[pre[m]][m]==mat[pre[m]][i]-mat[pre[i]][i])

k=i;

}

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

if (pre[i]!=-1)

ret+=mat[pre[i]][i];

return ret;

}

## 图论—最短路径

### 12.1 最短路径(单源bellman\_ford邻接阵)

//单源最短路径,bellman\_ford算法,邻接阵形式,复杂度O(n^3)

//求出源s到所有点的最短路经,传入图的大小n和邻接阵mat

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,路权可为负,若图包含负环则求解失败,返回0

//优化:先删去负边使用dijkstra求出上界,加速迭代过程

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

int bellman\_ford(int n,elem\_t mat[][MAXN],int s,elem\_t\* min,int\* pre){

int v[MAXN],i,j,k,tag;

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

min[i]=inf,v[i]=0,pre[i]=-1;

for (min[s]=0,j=0;j<n;j++){

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

if (!v[i]&&(k==-1||min[i]<min[k]))

k=i;

for (v[k]=1,i=0;i<n;i++)

if (!v[i]&&mat[k][i]>=0&&min[k]+mat[k][i]<min[i])

min[i]=min[k]+mat[pre[i]=k][i];

}

for (tag=1,j=0;tag&&j<=n;j++)

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

for (k=0;k<n;k++)

if (min[k]+mat[k][i]<min[i])

min[i]=min[k]+mat[pre[i]=k][i],tag=1;

return j<=n;

}

### 12.2 最短路径(单源dijkstra+bfs邻接表)

//单源最短路径,用于路权相等的情况,dijkstra优化为bfs,邻接表形式,复杂度O(m)

//求出源s到所有点的最短路经,传入图的大小n和邻接表list,边权值len

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负且相等!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

struct edge\_t{

int from,to;

edge\_t\* next;

};

void dijkstra(int n,edge\_t\* list[],elem\_t len,int s,elem\_t\* min,int\* pre){

edge\_t\* t;

int i,que[MAXN],f=0,r=0,p=1,l=1;

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

min[i]=inf;

min[que[0]=s]=0,pre[s]=-1;

for (;r<=f;l++,r=f+1,f=p-1)

for (i=r;i<=f;i++)

for (t=list[que[i]];t;t=t->next)

if (min[t->to]==inf)

min[que[p++]=t->to]=len\*l,pre[t->to]=que[i];

}

### 12.3 最短路径(单源dijkstra+bfs正向表)

//单源最短路径,用于路权相等的情况,dijkstra优化为bfs,正向表形式,复杂度O(m)

//求出源s到所有点的最短路经,传入图的大小n和正向表list,buf,边权值len

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负且相等!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

void dijkstra(int n,int\* list,int\* buf,elem\_t len,int s,elem\_t\* min,int\* pre){

int i,que[MAXN],f=0,r=0,p=1,l=1,t;

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

min[i]=inf;

min[que[0]=s]=0,pre[s]=-1;

for (;r<=f;l++,r=f+1,f=p-1)

for (i=r;i<=f;i++)

for (t=list[que[i]];t<list[que[i]+1];t++)

if (min[buf[t]]==inf)

min[que[p++]=buf[t]]=len\*l,pre[buf[t]]=que[i];

}

### 12.4 最短路径(单源dijkstra+binary\_heap邻接表)

//单源最短路径,dijkstra算法+二分堆,邻接表形式,复杂度O(mlogm)

//求出源s到所有点的最短路经,传入图的大小n和邻接表list

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

struct edge\_t{

int from,to;

elem\_t len;

edge\_t\* next;

};

#define \_cp(a,b) ((a).d<(b).d)

struct heap\_t{elem\_t d;int v;};

struct heap{

heap\_t h[MAXN\*MAXN];

int n,p,c;

void init(){n=0;}

void ins(heap\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(heap\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

void dijkstra(int n,edge\_t\* list[],int s,elem\_t\* min,int\* pre){

heap h;

edge\_t\* t;heap\_t e;

int v[MAXN],i;

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

min[i]=inf,v[i]=0,pre[i]=-1;

h.init();min[e.v=s]=e.d=0,h.ins(e);

while (h.del(e))

if (!v[e.v])

for (v[e.v]=1,t=list[e.v];t;t=t->next)

if (!v[t->to]&&min[t->from]+t->len<min[t->to])

pre[t->to]=t->from,min[e.v=t->to]=e.d=min[t->from]+t->len,h.ins(e);

}

### 12.5 最短路径(单源dijkstra+binary\_heap正向表)

//单源最短路径,dijkstra算法+二分堆,正向表形式,复杂度O(mlogm)

//求出源s到所有点的最短路经,传入图的大小n和正向表list,buf

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

struct edge\_t{

int to;

elem\_t len;

};

#define \_cp(a,b) ((a).d<(b).d)

struct heap\_t{elem\_t d;int v;};

struct heap{

heap\_t h[MAXN\*MAXN];

int n,p,c;

void init(){n=0;}

void ins(heap\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);

h[p]=e;

}

int del(heap\_t& e){

if (!n) return 0;

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);

h[p]=h[n--];return 1;

}

};

void dijkstra(int n,int\* list,edge\_t\* buf,int s,elem\_t\* min,int\* pre){

heap h;heap\_t e;

int v[MAXN],i,t,f;

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

min[i]=inf,v[i]=0,pre[i]=-1;

h.init();min[e.v=s]=e.d=0,h.ins(e);

while (h.del(e))

if (!v[e.v])

for (v[f=e.v]=1,t=list[f];t<list[f+1];t++)

if (!v[buf[t].to]&&min[f]+buf[t].len<min[buf[t].to])

pre[buf[t].to]=f,min[e.v=buf[t].to]=e.d=min[f]+buf[t].len,h.ins(e);

}

### 12.6 最短路径(单源dijkstra+mapped\_heap邻接表)

//单源最短路径,dijkstra算法+映射二分堆,邻接表形式,复杂度O(mlogn)

//求出源s到所有点的最短路经,传入图的大小n和邻接表list

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

struct edge\_t{

int from,to;

elem\_t len;

edge\_t\* next;

};

#define \_cp(a,b) ((a)<(b))

struct heap{

elem\_t h[MAXN+1];

int ind[MAXN+1],map[MAXN+1],n,p,c;

void init(){n=0;}

void ins(int i,elem\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

h[map[ind[p]=i]=p]=e;

}

int del(int i,elem\_t& e){

i=map[i];if (i<1||i>n) return 0;

for (e=h[p=i];p>1;h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

for (c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

int delmin(int& i,elem\_t& e){

if (n<1) return 0;i=ind[1];

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

};

void dijkstra(int n,edge\_t\* list[],int s,elem\_t\* min,int\* pre){

heap h;

edge\_t\* t;elem\_t e;

int v[MAXN],i;

for (h.init(),i=0;i<n;i++)

min[i]=((i==s)?0:inf),v[i]=0,pre[i]=-1,h.ins(i,min[i]);

while (h.delmin(i,e))

for (v[i]=1,t=list[i];t;t=t->next)

if (!v[t->to]&&min[i]+t->len<min[t->to])

pre[t->to]=i,h.del(t->to,e),min[t->to]=e=min[i]+t->len,h.ins(t->to,e);

}

### 12.7 最短路径(单源dijkstra+mapped\_heap正向表)

//单源最短路径,dijkstra算法+映射二分堆,正向表形式,复杂度O(mlogn)

//求出源s到所有点的最短路经,传入图的大小n和正向表list,buf

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

struct edge\_t{

int to;

elem\_t len;

};

#define \_cp(a,b) ((a)<(b))

struct heap{

elem\_t h[MAXN+1];

int ind[MAXN+1],map[MAXN+1],n,p,c;

void init(){n=0;}

void ins(int i,elem\_t e){

for (p=++n;p>1&&\_cp(e,h[p>>1]);h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

h[map[ind[p]=i]=p]=e;

}

int del(int i,elem\_t& e){

i=map[i];if (i<1||i>n) return 0;

for (e=h[p=i];p>1;h[map[ind[p]=ind[p>>1]]=p]=h[p>>1],p>>=1);

for (c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

int delmin(int& i,elem\_t& e){

if (n<1) return 0;i=ind[1];

for (e=h[p=1],c=2;c<n&&\_cp(h[c+=(c<n-1&&\_cp(h[c+1],h[c]))],h[n]);h[map[ind[p]=ind[c]]=p]=h[c],p=c,c<<=1);

h[map[ind[p]=ind[n]]=p]=h[n];n--;return 1;

}

};

void dijkstra(int n,int\* list,edge\_t\* buf,int s,elem\_t\* min,int\* pre){

heap h;elem\_t e;

int v[MAXN],i,t;

for (h.init(),i=0;i<n;i++)

min[i]=((i==s)?0:inf),v[i]=0,pre[i]=-1,h.ins(i,min[i]);

while (h.delmin(i,e))

for (v[i]=1,t=list[i];t<list[i+1];t++)

if (!v[buf[t].to]&&min[i]+buf[t].len<min[buf[t].to])

pre[buf[t].to]=i,h.del(buf[t].to,e),min[buf[t].to]=e=min[i]+buf[t].len,h.ins(buf[t].to,e);

}

### 12.8 最短路径(单源dijkstra邻接阵)

//单源最短路径,dijkstra算法,邻接阵形式,复杂度O(n^2)

//求出源s到所有点的最短路经,传入图的顶点数n,(有向)邻接矩阵mat

//返回到各点最短距离min[]和路径pre[],pre[i]记录s到i路径上i的父结点,pre[s]=-1

//可更改路权类型,但必须非负!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

void dijkstra(int n,elem\_t mat[][MAXN],int s,elem\_t\* min,int\* pre){

int v[MAXN],i,j,k;

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

min[i]=inf,v[i]=0,pre[i]=-1;

for (min[s]=0,j=0;j<n;j++){

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

if (!v[i]&&(k==-1||min[i]<min[k]))

k=i;

for (v[k]=1,i=0;i<n;i++)

if (!v[i]&&min[k]+mat[k][i]<min[i])

min[i]=min[k]+mat[pre[i]=k][i];

}

}

### 12.9 最短路径(多源floyd\_warshall邻接阵)

//多源最短路径,floyd\_warshall算法,复杂度O(n^3)

//求出所有点对之间的最短路经,传入图的大小和邻接阵

//返回各点间最短距离min[]和路径pre[],pre[i][j]记录i到j最短路径上j的父结点

//可更改路权类型,路权必须非负!

#define MAXN 200

#define inf 1000000000

typedef int elem\_t;

void floyd\_warshall(int n,elem\_t mat[][MAXN],elem\_t min[][MAXN],int pre[][MAXN]){

int i,j,k;

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

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

min[i][j]=mat[i][j],pre[i][j]=(i==j)?-1:i;

for (k=0;k<n;k++)

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

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

if (min[i][k]+min[k][j]<min[i][j])

min[i][j]=min[i][k]+min[k][j],pre[i][j]=pre[k][j];

}

## 应用

### 13.1 Joseph问题

// Joseph's Problem

// input: n,m -- the number of persons, the inteval between persons

// output: -- return the reference of last person

int josephus0(int n, int m)

{

if (n == 2) return (m%2) ? 2 : 1;

int v = (m+josephus0(n-1,m)) % n;

if (v == 0) v = n;

return v;

}

int josephus(int n, int m)

{

if (m == 1) return n;

if (n == 1) return 1;

if (m >=n) return josephus0(n,m);

int l = (n/m)\*m;

int j = josephus(n - (n/m), m);

if (j <= n-l) return l+j;

j -= n-l;

int t = (j/(m-1))\*m;

if ((j % (m-1)) == 0) return t-1;

return t + (j % (m-1));

}

### 13.2 N皇后构造解

//N皇后构造解,n>=4

void even1(int n,int \*p){

int i;

for (i=1;i<=n/2;i++)

p[i-1]=2\*i;

for (i=n/2+1;i<=n;i++)

p[i-1]=2\*i-n-1;

}

void even2(int n,int \*p){

int i;

for (i=1;i<=n/2;i++)

p[i-1]=(2\*i+n/2-3)%n+1;

for (i=n/2+1;i<=n;i++)

p[i-1]=n-(2\*(n-i+1)+n/2-3)%n;

}

void generate(int,int\*);

void odd(int n,int \*p){

generate(n-1,p),p[n-1]=n;

}

void generate(int n,int \*p){

if (n&1)

odd(n,p);

else if (n%6!=2)

even1(n,p);

else

even2(n,p);

}

### 13.3 布尔母函数

//布尔母函数

//判m[]个价值为w[]的货币能否构成value

//适合m[]较大w[]较小的情况

//返回布尔量

//传入货币种数n,个数m[],价值w[]和目标值value

#define MAXV 100000

int genfunc(int n,int\* m,int\* w,int value){

int i,j,k,c;

char r[MAXV];

for (r[0]=i=1;i<=value;r[i++]=0);

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

for (j=0;j<w[i];j++){

c=m[i]\*r[k=j];

while ((k+=w[i])<=value)

if (r[k])

c=m[i];

else if (c)

r[k]=1,c--;

if (r[value])

return 1;

}

}

return 0;

}

### 13.4 第k元素

//取第k个元素,k=0..n-1

//平均复杂度O(n)

//注意a[]中的顺序被改变

#define \_cp(a,b) ((a)<(b))

typedef int elem\_t;

elem\_t kth\_element(int n,elem\_t\* a,int k){

elem\_t t,key;

int l=0,r=n-1,i,j;

while (l<r){

for (key=a[((i=l-1)+(j=r+1))>>1];i<j;){

for (j--;\_cp(key,a[j]);j--);

for (i++;\_cp(a[i],key);i++);

if (i<j) t=a[i],a[i]=a[j],a[j]=t;

}

if (k>j) l=j+1;

else r=j;

}

return a[k];

}

### 13.5 幻方构造

//幻方构造(l!=2)

#define MAXN 100

void dllb(int l,int si,int sj,int sn,int d[][MAXN]){

int n,i=0,j=l/2;

for (n=1;n<=l\*l;n++){

d[i+si][j+sj]=n+sn;

if (n%l){

i=(i)?(i-1):(l-1);

j=(j==l-1)?0:(j+1);

}

else

i=(i==l-1)?0:(i+1);

}

}

void magic\_odd(int l,int d[][MAXN]){

dllb(l,0,0,0,d);

}

void magic\_4k(int l,int d[][MAXN]){

int i,j;

for (i=0;i<l;i++)

for (j=0;j<l;j++)

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

}

void magic\_other(int l,int d[][MAXN]){

int i,j,t;

dllb(l/2,0,0,0,d);

dllb(l/2,l/2,l/2,l\*l/4,d);

dllb(l/2,0,l/2,l\*l/2,d);

dllb(l/2,l/2,0,l\*l/4\*3,d);

for (i=0;i<l/2;i++)

for (j=0;j<l/4;j++)

if (i!=l/4||j)

t=d[i][j],d[i][j]=d[i+l/2][j],d[i+l/2][j]=t;

t=d[l/4][l/4],d[l/4][l/4]=d[l/4+l/2][l/4],d[l/4+l/2][l/4]=t;

for (i=0;i<l/2;i++)

for (j=l-l/4+1;j<l;j++)

t=d[i][j],d[i][j]=d[i+l/2][j],d[i+l/2][j]=t;

}

void generate(int l,int d[][MAXN]){

if (l%2)

magic\_odd(l,d);

else if (l%4==0)

magic\_4k(l,d);

else

magic\_other(l,d);

}

### 13.6 模式匹配(kmp)

//模式匹配,kmp算法,复杂度O(m+n)

//返回匹配位置,-1表示匹配失败,传入匹配串和模式串和长度

//可更改元素类型,更换匹配函数

#define MAXN 10000

#define \_match(a,b) ((a)==(b))

typedef char elem\_t;

int pat\_match(int ls,elem\_t\* str,int lp,elem\_t\* pat){

int fail[MAXN]={-1},i=0,j;

for (j=1;j<lp;j++){

for (i=fail[j-1];i>=0&&!\_match(pat[i+1],pat[j]);i=fail[i]);

fail[j]=(\_match(pat[i+1],pat[j])?i+1:-1);

}

for (i=j=0;i<ls&&j<lp;i++)

if (\_match(str[i],pat[j]))

j++;

else if (j)

j=fail[j-1]+1,i--;

return j==lp?(i-lp):-1;

}

### 13.7 逆序对数

//序列逆序对数,复杂度O(nlogn)

//传入序列长度和内容,返回逆序对数

//可更改元素类型和比较函数

#include <string.h>

#define MAXN 1000000

#define \_cp(a,b) ((a)<=(b))

typedef int elem\_t;

elem\_t \_tmp[MAXN];

int inv(int n,elem\_t\* a){

int l=n>>1,r=n-l,i,j;

int ret=(r>1?(inv(l,a)+inv(r,a+l)):0);

for (i=j=0;i<=l;\_tmp[i+j]=a[i],i++)

for (ret+=j;j<r&&(i==l||!\_cp(a[i],a[l+j]));\_tmp[i+j]=a[l+j],j++);

memcpy(a,\_tmp,sizeof(elem\_t)\*n);

return ret;

}

### 13.8 字符串最小表示

/\*

求字符串的最小表示

输入：字符串

返回：字符串最小表示的首字母位置(0...size-1)

\*/

template <class T>

int MinString(vector <T> &str)

{

int i, j, k;

vector <T> ss(str.size() << 1);

for (i = 0; i < str.size(); i ++) ss[i] = ss[i + str.size()] = str[i];

for (i = k = 0, j = 1; k < str.size() && i < str.size() && j < str.size(); ) {

for (k = 0; k < str.size() && ss[i + k] == ss[j + k]; k ++);

if (k < str.size()) {

if (ss[i + k] > ss[j + k])

i += k + 1;

else j += k + 1;

if (i == j) j ++;

}

}

return i < j ? i : j;

}

### 13.9 最长公共单调子序列

// 最长公共递增子序列， 时间复杂度O(n^2 \* logn)，空间 O(n^2)

/\*\*

\* n为a的大小, m为b的大小

\* 结果在ans中

\* "define \_cp(a,b) ((a)<(b))"求解最长严格递增序列

\*/

#define MAXN 1000

#define \_cp(a,b) ((a)<(b))

typedef int elem\_t;

elem\_t DP[MAXN][MAXN];

int num[MAXN], p[1<<20];

int LIS(int n, elem\_t \*a, int m, elem\_t \*b, elem\_t \*ans){

int i, j, l, r, k;

DP[0][0] = 0;

num[0] = (b[0] == a[0]);

for(i = 1; i < m; i++) {

num[i] = (b[i] == a[0]) || num[i-1];

DP[i][0] = 0;

}

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

if(b[0] == a[i] && !num[0]) {

num[0] = 1;

DP[0][0] = i<<10;

}

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

for(k=((l=0)+(r=num[j-1]-1))>>1; l<=r; k=(l+r)>>1)

if(\_cp(a[DP[j-1][k]>>10], a[i]))

l=k+1;

else

r=k-1;

if(l < num[j-1] && i == (DP[j-1][l]>>10) ){

if(l >= num[j]) DP[j][num[j]++] = DP[j-1][l];

else DP[j][l] = \_cp(a[DP[j][l]>>10],a[i]) ? DP[j][l] : DP[j-1][l];

}

if(b[j] == a[i]){

for(k=((l=0)+(r=num[j]-1))>>1; l<=r; k=(l+r)>>1)

if(\_cp(a[DP[j][k]>>10], a[i]))

l=k+1;

else

r=k-1;

DP[j][l] = (i<<10) + j;

num[j] += (l>=num[j]);

p[DP[j][l]] = l ? DP[j][l-1] : -1;

}

}

}

for (k=DP[m-1][i=num[m-1]-1];i>=0;ans[i--]=a[k>>10],k=p[k]);

return num[m-1];

}

### 13.10 最长子序列

//最长单调子序列,复杂度O(nlogn)

//注意最小序列覆盖和最长序列的对应关系,例如

//"define \_cp(a,b) ((a)>(b))"求解最长严格递减序列,则

//"define \_cp(a,b) (!((a)>(b)))"求解最小严格递减序列覆盖

//可更改元素类型和比较函数

#define MAXN 10000

#define \_cp(a,b) ((a)>(b))

typedef int elem\_t;

int subseq(int n,elem\_t\* a){

int b[MAXN],i,l,r,m,ret=0;

for (i=0;i<n;b[l]=i++,ret+=(l>ret))

for (m=((l=1)+(r=ret))>>1;l<=r;m=(l+r)>>1)

if (\_cp(a[b[m]],a[i]))

l=m+1;

else

r=m-1;

return ret;

}

int subseq(int n,elem\_t\* a,elem\_t\* ans){

int b[MAXN],p[MAXN],i,l,r,m,ret=0;

for (i=0;i<n;p[b[l]=i++]=b[l-1],ret+=(l>ret))

for (m=((l=1)+(r=ret))>>1;l<=r;m=(l+r)>>1)

if (\_cp(a[b[m]],a[i]))

l=m+1;

else

r=m-1;

for (m=b[i=ret];i;ans[--i]=a[m],m=p[m]);

return ret;

}

### 13.11 最大子串匹配

//最大子串匹配,复杂度O(mn)

//返回最大匹配值,传入两个串和串的长度,重载返回一个最大匹配

//注意做字符串匹配是串末的'\0'没有置!

//可更改元素类型,更换匹配函数和匹配价值函数

#include <string.h>

#define MAXN 100

#define max(a,b) ((a)>(b)?(a):(b))

#define \_match(a,b) ((a)==(b))

#define \_value(a,b) 1

typedef char elem\_t;

int str\_match(int m,elem\_t\* a,int n,elem\_t\* b){

int match[MAXN+1][MAXN+1],i,j;

memset(match,0,sizeof(match));

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

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

match[i+1][j+1]=max(max(match[i][j+1],match[i+1][j]),

(match[i][j]+\_value(a[i],b[i]))\*\_match(a[i],b[j]));

return match[m][n];

}

int str\_match(int m,elem\_t\* a,int n,elem\_t\* b,elem\_t\* ret){

int match[MAXN+1][MAXN+1],last[MAXN+1][MAXN+1],i,j,t;

memset(match,0,sizeof(match));

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

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

match[i+1][j+1]=(match[i][j+1]>match[i+1][j]?match[i][j+1]:match[i+1][j]);

last[i+1][j+1]=(match[i][j+1]>match[i+1][j]?3:1);

if ((t=(match[i][j]+\_value(a[i],b[i]))\*\_match(a[i],b[j]))>match[i+1][j+1])

match[i+1][j+1]=t,last[i+1][j+1]=2;

}

for (;match[i][j];i-=(last[t=i][j]>1),j-=(last[t][j]<3))

ret[match[i][j]-1]=(last[i][j]<3?a[i-1]:b[j-1]);

return match[m][n];

}

### 13.12 最大子段和

//求最大子段和,复杂度O(n)

//传入串长n和内容list[]

//返回最大子段和,重载返回子段位置(maxsum=list[start]+...+list[end])

//可更改元素类型

typedef int elem\_t;

elem\_t maxsum(int n,elem\_t\* list){

elem\_t ret,sum=0;

int i;

for (ret=list[i=0];i<n;i++)

sum=(sum>0?sum:0)+list[i],ret=(sum>ret?sum:ret);

return ret;

}

elem\_t maxsum(int n,elem\_t\* list,int& start,int& end){

elem\_t ret,sum=0;

int s,i;

for (ret=list[start=end=s=i=0];i<n;i++,s=(sum>0?s:i))

if ((sum=(sum>0?sum:0)+list[i])>ret)

ret=sum,start=s,end=i;

return ret;

}

### 13.13 最大子阵和

//求最大子阵和,复杂度O(n^3)

//传入阵的大小m,n和内容mat[][]

//返回最大子阵和,重载返回子阵位置(maxsum=list[s1][s2]+...+list[e1][e2])

//可更改元素类型

#define MAXN 100

typedef int elem\_t;

elem\_t maxsum(int m,int n,elem\_t mat[][MAXN]){

elem\_t matsum[MAXN][MAXN+1],ret,sum;

int i,j,k;

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

for (matsum[i][j=0]=0;j<n;j++)

matsum[i][j+1]=matsum[i][j]+mat[i][j];

for (ret=mat[0][j=0];j<n;j++)

for (k=j;k<n;k++)

for (sum=0,i=0;i<m;i++)

sum=(sum>0?sum:0)+matsum[i][k+1]-matsum[i][j],ret=(sum>ret?sum:ret);

return ret;

}

elem\_t maxsum(int m,int n,elem\_t mat[][MAXN],int& s1,int& s2,int& e1,int& e2){

elem\_t matsum[MAXN][MAXN+1],ret,sum;

int i,j,k,s;

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

for (matsum[i][j=0]=0;j<n;j++)

matsum[i][j+1]=matsum[i][j]+mat[i][j];

for (ret=mat[s1=e1=0][s2=e2=j=0];j<n;j++)

for (k=j;k<n;k++)

for (sum=0,s=i=0;i<m;i++,s=(sum>0?s:i))

if ((sum=(sum>0?sum:0)+matsum[i][k+1]-matsum[i][j])>ret)

ret=sum,s1=s,s2=i,e1=j,e2=k;

return ret;

}

## 其它

### 14.1 大数(只能处理正数)

#include <iostream.h>

#include <string.h>

#define DIGIT 4

#define DEPTH 10000

#define MAX 100

typedef int bignum\_t[MAX+1];

int read(bignum\_t a,istream& is=cin){

char buf[MAX\*DIGIT+1],ch;

int i,j;

memset((void\*)a,0,sizeof(bignum\_t));

if (!(is>>buf)) return 0;

for (a[0]=strlen(buf),i=a[0]/2-1;i>=0;i--)

ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;

for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf);j<a[0]\*DIGIT;buf[j++]='0');

for (i=1;i<=a[0];i++)

for (a[i]=0,j=0;j<DIGIT;j++)

a[i]=a[i]\*10+buf[i\*DIGIT-1-j]-'0';

for (;!a[a[0]]&&a[0]>1;a[0]--);

return 1;

}

void write(const bignum\_t a,ostream& os=cout){

int i,j;

for (os<<a[i=a[0]],i--;i;i--)

for (j=DEPTH/10;j;j/=10)

os<<a[i]/j%10;

}

int comp(const bignum\_t a,const bignum\_t b){

int i;

if (a[0]!=b[0])

return a[0]-b[0];

for (i=a[0];i;i--)

if (a[i]!=b[i])

return a[i]-b[i];

return 0;

}

int comp(const bignum\_t a,const int b){

int c[12]={1};

for (c[1]=b;c[c[0]]>=DEPTH;c[c[0]+1]=c[c[0]]/DEPTH,c[c[0]]%=DEPTH,c[0]++);

return comp(a,c);

}

int comp(const bignum\_t a,const int c,const int d,const bignum\_t b){

int i,t=0,O=-DEPTH\*2;

if (b[0]-a[0]<d&&c)

return 1;

for (i=b[0];i>d;i--){

t=t\*DEPTH+a[i-d]\*c-b[i];

if (t>0) return 1;

if (t<O) return 0;

}

for (i=d;i;i--){

t=t\*DEPTH-b[i];

if (t>0) return 1;

if (t<O) return 0;

}

return t>0;

}

void add(bignum\_t a,const bignum\_t b){

int i;

for (i=1;i<=b[0];i++)

if ((a[i]+=b[i])>=DEPTH)

a[i]-=DEPTH,a[i+1]++;

if (b[0]>=a[0])

a[0]=b[0];

else

for (;a[i]>=DEPTH&&i<a[0];a[i]-=DEPTH,i++,a[i]++);

a[0]+=(a[a[0]+1]>0);

}

void add(bignum\_t a,const int b){

int i=1;

for (a[1]+=b;a[i]>=DEPTH&&i<a[0];a[i+1]+=a[i]/DEPTH,a[i]%=DEPTH,i++);

for (;a[a[0]]>=DEPTH;a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);

}

void sub(bignum\_t a,const bignum\_t b){

int i;

for (i=1;i<=b[0];i++)

if ((a[i]-=b[i])<0)

a[i+1]--,a[i]+=DEPTH;

for (;a[i]<0;a[i]+=DEPTH,i++,a[i]--);

for (;!a[a[0]]&&a[0]>1;a[0]--);

}

void sub(bignum\_t a,const int b){

int i=1;

for (a[1]-=b;a[i]<0;a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH\*DEPTH,i++);

for (;!a[a[0]]&&a[0]>1;a[0]--);

}

void sub(bignum\_t a,const bignum\_t b,const int c,const int d){

int i,O=b[0]+d;

for (i=1+d;i<=O;i++)

if ((a[i]-=b[i-d]\*c)<0)

a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH\*DEPTH;

for (;a[i]<0;a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH\*DEPTH,i++);

for (;!a[a[0]]&&a[0]>1;a[0]--);

}

void mul(bignum\_t c,const bignum\_t a,const bignum\_t b){

int i,j;

memset((void\*)c,0,sizeof(bignum\_t));

for (c[0]=a[0]+b[0]-1,i=1;i<=a[0];i++)

for (j=1;j<=b[0];j++)

if ((c[i+j-1]+=a[i]\*b[j])>=DEPTH)

c[i+j]+=c[i+j-1]/DEPTH,c[i+j-1]%=DEPTH;

for (c[0]+=(c[c[0]+1]>0);!c[c[0]]&&c[0]>1;c[0]--);

}

void mul(bignum\_t a,const int b){

int i;

for (a[1]\*=b,i=2;i<=a[0];i++){

a[i]\*=b;

if (a[i-1]>=DEPTH)

a[i]+=a[i-1]/DEPTH,a[i-1]%=DEPTH;

}

for (;a[a[0]]>=DEPTH;a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);

for (;!a[a[0]]&&a[0]>1;a[0]--);

}

void mul(bignum\_t b,const bignum\_t a,const int c,const int d){

int i;

memset((void\*)b,0,sizeof(bignum\_t));

for (b[0]=a[0]+d,i=d+1;i<=b[0];i++)

if ((b[i]+=a[i-d]\*c)>=DEPTH)

b[i+1]+=b[i]/DEPTH,b[i]%=DEPTH;

for (;b[b[0]+1];b[0]++,b[b[0]+1]=b[b[0]]/DEPTH,b[b[0]]%=DEPTH);

for (;!b[b[0]]&&b[0]>1;b[0]--);

}

void div(bignum\_t c,bignum\_t a,const bignum\_t b){

int h,l,m,i;

memset((void\*)c,0,sizeof(bignum\_t));

c[0]=(b[0]<a[0]+1)?(a[0]-b[0]+2):1;

for (i=c[0];i;sub(a,b,c[i]=m,i-1),i--)

for (h=DEPTH-1,l=0,m=(h+l+1)>>1;h>l;m=(h+l+1)>>1)

if (comp(b,m,i-1,a)) h=m-1;

else l=m;

for (;!c[c[0]]&&c[0]>1;c[0]--);

c[0]=c[0]>1?c[0]:1;

}

void div(bignum\_t a,const int b,int& c){

int i;

for (c=0,i=a[0];i;c=c\*DEPTH+a[i],a[i]=c/b,c%=b,i--);

for (;!a[a[0]]&&a[0]>1;a[0]--);

}

void sqrt(bignum\_t b,bignum\_t a){

int h,l,m,i;

memset((void\*)b,0,sizeof(bignum\_t));

for (i=b[0]=(a[0]+1)>>1;i;sub(a,b,m,i-1),b[i]+=m,i--)

for (h=DEPTH-1,l=0,b[i]=m=(h+l+1)>>1;h>l;b[i]=m=(h+l+1)>>1)

if (comp(b,m,i-1,a)) h=m-1;

else l=m;

for (;!b[b[0]]&&b[0]>1;b[0]--);

for (i=1;i<=b[0];b[i++]>>=1);

}

int length(const bignum\_t a){

int t,ret;

for (ret=(a[0]-1)\*DIGIT,t=a[a[0]];t;t/=10,ret++);

return ret>0?ret:1;

}

int digit(const bignum\_t a,const int b){

int i,ret;

for (ret=a[(b-1)/DIGIT+1],i=(b-1)%DIGIT;i;ret/=10,i--);

return ret%10;

}

int zeronum(const bignum\_t a){

int ret,t;

for (ret=0;!a[ret+1];ret++);

for (t=a[ret+1],ret\*=DIGIT;!(t%10);t/=10,ret++);

return ret;

}

void comp(int\* a,const int l,const int h,const int d){

int i,j,t;

for (i=l;i<=h;i++)

for (t=i,j=2;t>1;j++)

while (!(t%j))

a[j]+=d,t/=j;

}

void convert(int\* a,const int h,bignum\_t b){

int i,j,t=1;

memset(b,0,sizeof(bignum\_t));

for (b[0]=b[1]=1,i=2;i<=h;i++)

if (a[i])

for (j=a[i];j;t\*=i,j--)

if (t\*i>DEPTH)

mul(b,t),t=1;

mul(b,t);

}

void combination(bignum\_t a,int m,int n){

int\* t=new int[m+1];

memset((void\*)t,0,sizeof(int)\*(m+1));

comp(t,n+1,m,1);

comp(t,2,m-n,-1);

convert(t,m,a);

delete []t;

}

void permutation(bignum\_t a,int m,int n){

int i,t=1;

memset(a,0,sizeof(bignum\_t));

a[0]=a[1]=1;

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

if (t\*i>DEPTH)

mul(a,t),t=1;

mul(a,t);

}

#define SGN(x) ((x)>0?1:((x)<0?-1:0))

#define ABS(x) ((x)>0?(x):-(x))

int read(bignum\_t a,int &sgn,istream& is=cin){

char str[MAX\*DIGIT+2],ch,\*buf;

int i,j;

memset((void\*)a,0,sizeof(bignum\_t));

if (!(is>>str)) return 0;

buf=str,sgn=1;

if (\*buf=='-') sgn=-1,buf++;

for (a[0]=strlen(buf),i=a[0]/2-1;i>=0;i--)

ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;

for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf);j<a[0]\*DIGIT;buf[j++]='0');

for (i=1;i<=a[0];i++)

for (a[i]=0,j=0;j<DIGIT;j++)

a[i]=a[i]\*10+buf[i\*DIGIT-1-j]-'0';

for (;!a[a[0]]&&a[0]>1;a[0]--);

if (a[0]==1&&!a[1]) sgn=0;

return 1;

}

### 14.2 分数

struct frac{

int num,den;

};

double fabs(double x){

return x>0?x:-x;

}

int gcd(int a,int b){

int t;

if (a<0)

a=-a;

if (b<0)

b=-b;

if (!b)

return a;

while (t=a%b)

a=b,b=t;

return b;

}

void simplify(frac& f){

int t;

if (t=gcd(f.num,f.den))

f.num/=t,f.den/=t;

else

f.den=1;

}

frac f(int n,int d,int s=1){

frac ret;

if (d<0)

ret.num=-n,ret.den=-d;

else

ret.num=n,ret.den=d;

if (s)

simplify(ret);

return ret;

}

frac convert(double x){

frac ret;

for (ret.den=1;fabs(x-int(x))>1e-10;ret.den\*=10,x\*=10);

ret.num=(int)x;

simplify(ret);

return ret;

}

int fraqcmp(frac a,frac b){

int g1=gcd(a.den,b.den),g2=gcd(a.num,b.num);

if (!g1||!g2)

return 0;

return b.den/g1\*(a.num/g2)-a.den/g1\*(b.num/g2);

}

frac add(frac a,frac b){

int g1=gcd(a.den,b.den),g2,t;

if (!g1)

return f(1,0,0);

t=b.den/g1\*a.num+a.den/g1\*b.num;

g2=gcd(g1,t);

return f(t/g2,a.den/g1\*(b.den/g2),0);

}

frac sub(frac a,frac b){

return add(a,f(-b.num,b.den,0));

}

frac mul(frac a,frac b){

int t1=gcd(a.den,b.num),t2=gcd(a.num,b.den);

if (!t1||!t2)

return f(1,1,0);

return f(a.num/t2\*(b.num/t1),a.den/t1\*(b.den/t2),0);

}

frac div(frac a,frac b){

return mul(a,f(b.den,b.num,0));

}

### 14.3 矩阵

define MAXN 100

#define fabs(x) ((x)>0?(x):-(x))

#define zero(x) (fabs(x)<1e-10)

struct mat{

int n,m;

double data[MAXN][MAXN];

};

int mul(mat& c,const mat& a,const mat& b){

int i,j,k;

if (a.m!=b.n)

return 0;

c.n=a.n,c.m=b.m;

for (i=0;i<c.n;i++)

for (j=0;j<c.m;j++)

for (c.data[i][j]=k=0;k<a.m;k++)

c.data[i][j]+=a.data[i][k]\*b.data[k][j];

return 1;

}

int inv(mat& a){

int i,j,k,is[MAXN],js[MAXN];

double t;

if (a.n!=a.m)

return 0;

for (k=0;k<a.n;k++){

for (t=0,i=k;i<a.n;i++)

for (j=k;j<a.n;j++)

if (fabs(a.data[i][j])>t)

t=fabs(a.data[is[k]=i][js[k]=j]);

if (zero(t))

return 0;

if (is[k]!=k)

for (j=0;j<a.n;j++)

t=a.data[k][j],a.data[k][j]=a.data[is[k]][j],a.data[is[k]][j]=t;

if (js[k]!=k)

for (i=0;i<a.n;i++)

t=a.data[i][k],a.data[i][k]=a.data[i][js[k]],a.data[i][js[k]]=t;

a.data[k][k]=1/a.data[k][k];

for (j=0;j<a.n;j++)

if (j!=k)

a.data[k][j]\*=a.data[k][k];

for (i=0;i<a.n;i++)

if (i!=k)

for (j=0;j<a.n;j++)

if (j!=k)

a.data[i][j]-=a.data[i][k]\*a.data[k][j];

for (i=0;i<a.n;i++)

if (i!=k)

a.data[i][k]\*=-a.data[k][k];

}

for (k=a.n-1;k>=0;k--){

for (j=0;j<a.n;j++)

if (js[k]!=k)

t=a.data[k][j],a.data[k][j]=a.data[js[k]][j],a.data[js[k]][j]=t;

for (i=0;i<a.n;i++)

if (is[k]!=k)

t=a.data[i][k],a.data[i][k]=a.data[i][is[k]],a.data[i][is[k]]=t;

}

return 1;

}

double det(const mat& a){

int i,j,k,sign=0;

double b[MAXN][MAXN],ret=1,t;

if (a.n!=a.m)

return 0;

for (i=0;i<a.n;i++)

for (j=0;j<a.m;j++)

b[i][j]=a.data[i][j];

for (i=0;i<a.n;i++){

if (zero(b[i][i])){

for (j=i+1;j<a.n;j++)

if (!zero(b[j][i]))

break;

if (j==a.n)

return 0;

for (k=i;k<a.n;k++)

t=b[i][k],b[i][k]=b[j][k],b[j][k]=t;

sign++;

}

ret\*=b[i][i];

for (k=i+1;k<a.n;k++)

b[i][k]/=b[i][i];

for (j=i+1;j<a.n;j++)

for (k=i+1;k<a.n;k++)

b[j][k]-=b[j][i]\*b[i][k];

}

if (sign&1)

ret=-ret;

return ret;

}

### 14.4 线性方程组

#define MAXN 100

#define fabs(x) ((x)>0?(x):-(x))

#define eps 1e-10

//列主元gauss消去求解a[][]x[]=b[]

//返回是否有唯一解,若有解在b[]中

int gauss\_cpivot(int n,double a[][MAXN],double b[]){

int i,j,k,row;

double maxp,t;

for (k=0;k<n;k++){

for (maxp=0,i=k;i<n;i++)

if (fabs(a[i][k])>fabs(maxp))

maxp=a[row=i][k];

if (fabs(maxp)<eps)

return 0;

if (row!=k){

for (j=k;j<n;j++)

t=a[k][j],a[k][j]=a[row][j],a[row][j]=t;

t=b[k],b[k]=b[row],b[row]=t;

}

for (j=k+1;j<n;j++){

a[k][j]/=maxp;

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

a[i][j]-=a[i][k]\*a[k][j];

}

b[k]/=maxp;

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

b[i]-=b[k]\*a[i][k];

}

for (i=n-1;i>=0;i--)

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

b[i]-=a[i][j]\*b[j];

return 1;

}

//全主元gauss消去解a[][]x[]=b[]

//返回是否有唯一解,若有解在b[]中

int gauss\_tpivot(int n,double a[][MAXN],double b[]){

int i,j,k,row,col,index[MAXN];

double maxp,t;

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

index[i]=i;

for (k=0;k<n;k++){

for (maxp=0,i=k;i<n;i++)

for (j=k;j<n;j++)

if (fabs(a[i][j])>fabs(maxp))

maxp=a[row=i][col=j];

if (fabs(maxp)<eps)

return 0;

if (col!=k){

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

t=a[i][col],a[i][col]=a[i][k],a[i][k]=t;

j=index[col],index[col]=index[k],index[k]=j;

}

if (row!=k){

for (j=k;j<n;j++)

t=a[k][j],a[k][j]=a[row][j],a[row][j]=t;

t=b[k],b[k]=b[row],b[row]=t;

}

for (j=k+1;j<n;j++){

a[k][j]/=maxp;

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

a[i][j]-=a[i][k]\*a[k][j];

}

b[k]/=maxp;

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

b[i]-=b[k]\*a[i][k];

}

for (i=n-1;i>=0;i--)

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

b[i]-=a[i][j]\*b[j];

for (k=0;k<n;k++)

a[0][index[k]]=b[k];

for (k=0;k<n;k++)

b[k]=a[0][k];

return 1;

}

### 14.5 线性相关

//判线性相关(正交化)

//传入m个n维向量

#include <math.h>

#define MAXN 100

#define eps 1e-10

int linear\_dependent(int m,int n,double vec[][MAXN]){

double ort[MAXN][MAXN],e;

int i,j,k;

if (m>n)

return 1;

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

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

ort[i][j]=vec[i][j];

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

for (e=j=0;j<n;j++)

e+=ort[i][j]\*ort[k][j];

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

ort[i][j]-=e\*ort[k][j];

for (e=j=0;j<n;j++)

e+=ort[i][j]\*ort[i][j];

if (fabs(e=sqrt(e))<eps)

return 1;

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

ort[i][j]/=e;

}

}

return 0;

}

### 14.6 日期

//日期函数

int days[12]={31,28,31,30,31,30,31,31,30,31,30,31};

struct date{

int year,month,day;

};

//判闰年

inline int leap(int year){

return (year%4==0&&year%100!=0)||year%400==0;

}

//判合法性

inline int legal(date a){

if (a.month<0||a.month>12)

return 0;

if (a.month==2)

return a.day>0&&a.day<=28+leap(a.year);

return a.day>0&&a.day<=days[a.month-1];

}

//比较日期大小

inline int datecmp(date a,date b){

if (a.year!=b.year)

return a.year-b.year;

if (a.month!=b.month)

return a.month-b.month;

return a.day-b.day;

}

//返回指定日期是星期几

int weekday(date a){

int tm=a.month>=3?(a.month-2):(a.month+10);

int ty=a.month>=3?a.year:(a.year-1);

return (ty+ty/4-ty/100+ty/400+(int)(2.6\*tm-0.2)+a.day)%7;

}

//日期转天数偏移

int date2int(date a){

int ret=a.year\*365+(a.year-1)/4-(a.year-1)/100+(a.year-1)/400,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;

}