last_one 模板

目录

计算几何

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

凸包

凸包

```
/*时间复杂度 nlog(n)*/
struct point{
    double x, y;
    double dis;
}pnt[10010] ,res[10010];
//pnt 存输入点，res 存凸包点
int n;//点的个数
bool mult(point sp, point ep, point op){
    return (sp.x - op.x) * (ep.y - op.y) >=
        (ep.x - op.x) * (sp.y - op.y);
}
bool operator < (const point &l, const point
&r){
    return l.y < r.y ||
(l.y == r.y && l.x < r.x);
}
double dist(point a ,point b){
    return sqrt((a.y - b.y) * (a.y - b.y) +
(a.x - b.x) * (a.x - b.x));
}
int graham(){
    int i, len, k = 0, top = 1;
    sort(pnt, pnt + n);
    if (n == 0) return 0;
res[0] = pnt[0];
    if (n == 1) return 1;
res[1] = pnt[1];
    if (n == 2) return 2;
 res[2] = pnt[2];
    for (i = 2; i < n; i++) {
        while (top &&
mult(pnt[i], res[top], res[top-1]))
 top--;
        res[++top] = pnt[i];
    }
    len = top;
res[++top] = pnt[n - 2];
    for (i = n - 3; i >= 0; i--) {
        while (top!=len &&
mult(pnt[i], res[top], res[top-1]))
top--;
        res[++top] = pnt[i];
    }
    return top; // 返回凸包中点的个数
}
int main()
{
    int top;
    double ans;
    while(scanf("%d",&n),n){
        for(int i = 0;i < n;i++)

    scanf("%lf%lf",&pnt[i].x,&pnt[i].y);
        if(n==1)
            printf("0.00\n");
        else if(n==2)

    printf("%.2lf\n",dist(pnt[0],pnt[1]));
        else{
            top = graham();
            //ans 为凸包周长
            ans = 0;
            for(int i = 0;i < top - 1 ;i++)
                ans += dist(res[i],res[i +
1]);
            ans += dist(res[0],res[top - 1]);
            printf("%.2lf\n",ans);
        }
    }
    return 0;
}
```

三维凸包

```
#define PR 1e-8
#define N 510
struct TPoint{
    double x,y,z;
    TPoint(){}
    TPoint(double _x,double _y,double
_z):x(_x),y(_y),z(_z){}
    TPoint operator-(const TPoint p) {
        return TPoint(x-p.x,y-p.y,z-p.z);}
    TPoint operator*(const TPoint p) {//叉积
        return
TPoint(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y*p.x);}
    double operator^(const TPoint p) {//点积
        return x*p.x+y*p.y+z*p.z;}
};
struct fac{
    int a,b,c;//凸包一个面上的三个点的编号
    bool ok;//该面是否是最终凸包中的面
};
struct T3dhull{
    int n;//初始点数
    TPoint ply[N];//初始点
    int trianglecnt;//凸包上三角形数
    fac tri[N];//凸包三角形
    int vis[N][N];//点i到点j是属于哪个面
    double dist(TPoint a){ //两点长度
        return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);}
    double area(TPoint a,TPoint b,TPoint c){ //三
角形面积*2
        return dist((b-a)*(c-a));}
    double volume(TPoint a,TPoint b,TPoint
c,TPoint d){
        return (b-a)*(c-a)^(d-a);}//四面体有向体
积*6
    double ptoplane(TPoint &p,fac &f){//正：点在
面同向
        TPoint
m=ply[f.b]-ply[f.a],n=ply[f.c]-ply[f.a],t=p-ply[f.a];
        return (m*n)^t;
    }
    void deal(int p,int a,int b){
        int f=vis[a][b];
        fac add;
        if(tri[f].ok){
            if((ptoplane(ply[p],tri[f]))>PR)
dfs(p,f);
            else{

add.a=b,add.b=a,add.c=p,add.ok=1;

vis[p][b]=vis[a][p]=vis[b][a]=trianglecnt;
                tri[trianglecnt++]=add;
            }
        }
    }
    void dfs(int p,int cnt){
    //维护凸包，如果点p在凸包外更新凸包
        tri[cnt].ok=0;
        deal(p,tri[cnt].b,tri[cnt].a);
        deal(p,tri[cnt].c,tri[cnt].b);
        deal(p,tri[cnt].a,tri[cnt].c);
    }
    bool same(int s,int e){
    //判断两个面是否为同一面
        TPoint a=ply[tri[s].a];
        TPoint b=ply[tri[s].b];
        TPoint c=ply[tri[s].c];
        return
fabs(volume(a,b,c,ply[tri[e].a]))<PR

&&fabs(volume(a,b,c,ply[tri[e].b]))<PR

&&fabs(volume(a,b,c,ply[tri[e].c]))<PR;
    }
    void construct(){//构建凸包
        int i,j;
        trianglecnt=0;
        if(n<4) return ;
        bool tmp=true;
        for(i=1;i<n;i++){//前两点不共点
            if((dist(ply[0]-ply[i]))>PR){
                swap(ply[1],ply[i]);
                tmp=false; break;
            }
        }
```

```
        if(tmp) return;
        tmp=true;
        for(i=2;i<n;i++){//前三点不共线

if((dist((ply[0]-ply[1])*(ply[1]-ply[i])))>PR){
                swap(ply[2],ply[i]);
                tmp=false; break;
            }
        }
        if(tmp) return ;
        tmp=true;
        for(i=3;i<n;i++){//前四点不共面

if(fabs((ply[0]-ply[1])*(ply[1]-ply[2])^(ply[0]-ply[i
]))>PR){
                swap(ply[3],ply[i]);
                tmp=false; break;
            }
        }
        if(tmp) return ;
        fac add;
        for(i=0;i<4;i++){//构建初始四面体
            add.a=(i+1)%4,add.b=(i+2)%4;
            add.c=(i+3)%4,add.ok=1;
            if((ptoplane(ply[i],add))>0)
                swap(add.b,add.c);

vis[add.a][add.b]=vis[add.b][add.c]=vis[add.c][add
.a]=trianglecnt;
            tri[trianglecnt++]=add;
        }
        for(i=4;i<n;i++){//构建更新凸包
            for(j=0;j<trianglecnt;j++){

if(tri[j].ok&&(ptoplane(ply[i],tri[j]))>PR){
                    dfs(i,j); break;
                }
            }
        }
        int cnt=trianglecnt;
        trianglecnt=0;
        for(i=0;i<cnt;i++)
        {
            if(tri[i].ok)
                tri[trianglecnt++]=tri[i];
        }
    }
    double area(){//表面积
        double ret=0;
        for(int i=0;i<trianglecnt;i++)

ret+=area(ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]);
        return ret/2.0;
    }
    double volume(){//体积
        TPoint p(0,0,0);
        double ret=0;
        for(int i=0;i<trianglecnt;i++)

ret+=volume(p,ply[tri[i].a],ply[tri[i].b],ply[tri[i].c])
;
        return fabs(ret/6);
    }
    int facetri() {return trianglecnt;}//表面三角形
数
    int facepolygon(){//表面多边形数
        int ans=0,i,j,k;
        for(i=0;i<trianglecnt;i++){
            for(j=0,k=1;j<i;j++){
                if(same(i,j)) {
                    k=0;break;}
            }
            ans+=k;
        }
        return ans;
    }
     double res(){//三维凸包中的点到凸包的最
短距离
        double _min=1e300;
        for(int i=0;i<trianglecnt;i++){
            double
now=Dis(ply[tri[i].a],ply[tri[i].b],ply[tri[i].c],dd);
            if(_min>now) _min=now;
        }
        return _min;
    }
}hull;
int main(){
```

```
    while(~scanf("%d",&hull.n)){
        int i;
        for(i=0;i<hull.n;i++)

scanf("%lf%lf%lf",&hull.ply[i].x,&hull.ply[i].y,&h
ull.ply[i].z);
        hull.construct();
        printf("%.3lf\n",hull.area());
    }
    return 0;
}
```

判断线段相交

```
const double eps=1e-10;
struct point{double x, y;};
double min(double a, double b){
    return a<b?a:b;}
double max(double a, double b){
    return a>b?a:b;}
bool inter(point a, point b, point c, point d){
    if (min(a.x, b.x)>max(c.x, d.x)||
        min(a.y, b.y)>max(c.y, d.y)||
        min(c.x, d.x)>max(a.x, b.x)||
        min(c.y, d.y)>max(a.y, b.y) )
            return 0;
    double h, i, j, k;
    h=(b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x);
    i=(b.x-a.x)*(d.y-a.y)-(b.y-a.y)*(d.x-a.x);
    j=(d.x-c.x)*(a.y-c.y)-(d.y-c.y)*(a.x-c.x);
    k=(d.x-c.x)*(b.y-c.y)-(d.y-c.y)*(b.x-c.x);
    return h * i <= eps && j * k <= eps;
}
```

三角形几个重要的点

```
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;
}
//最小覆盖圆圆心
struct point circumcenter(struct point a,struct point
b,struct point c){
    struct line u,v;
    struct point o1,o2,o3,ans;
    double r1,r2,r3;
    bool p=0;
    o1.x=(b.x+c.x)/2;
    o1.y=(b.y+c.y)/2;
    r1=distance(b,c)/2;
    if(distance(o1,a)<=r1){
        ans=o1;
        p=1;
    }
    o2.x=(a.x+c.x)/2;
    o2.y=(a.y+c.y)/2;
    r2=distance(a,c)/2;
    if(distance(o2,b)<=r2){
        ans=o2;
        p=1;
    }
    o3.x=(b.x+a.x)/2;
    o3.y=(b.y+a.y)/2;
    r3=distance(b,a)/2;
    if(distance(o3,c)<=r3){
        ans=o3;
        p=1;
    }
    if(p==0){
        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);
        }
        else return ans;
}
//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;
}
```

多边形面积

```
struct point{ double x; double
y;}map[100005];
int main(){
    int i,j,m,n;
    double x,y,x1,y1,x2,y2,ans,a,b,c,d;
    while(scanf("%d",&n)!=EOF){
        ans=0;
        for(i=1;i<=n;i++)

    scanf("%lf%lf",&map[i].x,&map[i].y);
        x=map[1].x;
        y=map[1].y;
        for(i=2;i<=n-1;i++){
            x1=map[i].x;
            y1=map[i].y;
            x2=map[i+1].x;
            y2=map[i+1].y;
            a=x-x1;
            b=y-y1;
            c=x-x2;
            d=y-y2;
            ans+=(a*d-b*c)/2;
        }
        ans=fabs(ans);
        printf("%.2lf\n",ans);
    }
    return 0;
}
```

多边形重心

```
struct point{ double x; double
y;}map[100005];
int main(){
    int i,j,m,n;
    double x,y,ss,sx,sy,s;
    while(scanf("%d",&n)!=EOF){
        ss=0;
        sx=0;
        sy=0;
        for(i=1;i<=n;i++)
    scanf("%lf%lf",&map[i].x,&map[i].y);
        for(i=2;i<=n-1;i++){

    x=(map[1].x+map[i].x+map[i+1].x)/3;

    y=(map[1].y+map[i].y+map[i+1].y)/3;

    s=( (map[i].x-map[1].x)*(map[i+1].y-m
ap[1].y)-(map[i+1].x-map[1].x)*(map[i].y-
map[1].y) )/2;
            sx+=s*x;
            sy+=s*y;
            ss+=s;
        }

    printf("%.2lf %.2lf\n",sx/ss+0.000000
01,sy/ss+0.00000001);
    }
    return 0;
}
```

分治法求最小点对（二维）

```
/*分治算法求最小点对*/
#define MAXN 100005
struct Point{double x,y;};
struct Point
point[MAXN],*px[MAXN],*py[MAXN];
double get_dis(Point *p1,Point *p2){
    return
sqrt((p1->x-p2->x)*(p1->x-p2->x)+(p1->y-p
2->y)*(p1->y-p2->y));
}
bool cmpx(Point *p1,Point *p2) {return
p1->x<p2->x;}
bool cmpy(Point *p1,Point *p2) {return
p1->y<p2->y;}
double min(double a,double b){return
a<b?a:b;}
//-------核心代码----------//
double closest(int s,int e){
    if(s+1==e)
```

```
        return get_dis(px[s],px[e]);
    if(s+2==e)
        return
min(get_dis(px[s],px[s+1]),min(get_dis(px
[s+1],px[e]),get_dis(px[s],px[e])));
    int mid=(s+e)>>1;
    double
ans=min(closest(s,mid),closest(mid+1,e));
//递归求解
    int i,j,cnt=0;
    for(i=s;i<=e;i++){//把x坐标在
px[mid].x-ans~px[mid].x+ans范围内的点取出
来

    if(px[i]->x>=px[mid]->x-ans&&px[i]->x
<=px[mid]->x+ans)
            py[cnt++]=px[i];
    }
    sort(py,py+cnt,cmpy);//按y坐标排序
    for(i=0;i<cnt;i++){
        for(j=i+1;j<cnt;j++){//py数组中
的点是按照y坐标升序的
            if(py[j]->y-py[i]->y>=ans)
                break;

    ans=min(ans,get_dis(py[i],py[j]));
        }
    }
    return ans;
}
int main(){
    int i,n;
    while(scanf("%d",&n)!=EOF){
        if(n==0)
            break;
        for(i=0;i<n;i++){

    scanf("%lf%lf",&point[i].x,&point[i].
y);
            px[i]=&point[i];
        }
        sort(px,px+n,cmpx);
        double distance=closest(0,n-1);
        printf("%.2lf\n",distance);
    }
    return 0;
}
```

分治法求最小点对（三维）

```
#define MAXN 100005
using namespace std;
struct Point{double x,y,z;};
struct Point
point[MAXN],*px[MAXN],*py[MAXN];
double get_dis(Point *p1,Point *p2){
    return
sqrt((p1->x-p2->x)*(p1->x-p2->x)+(p1->y-p
2->y)*(p1->y-p2->y)+(p1->z-p2->z)*(p1->z-
p2->z));
}
bool cmpx(Point *p1,Point *p2) {return
p1->x<p2->x;}
bool cmpy(Point *p1,Point *p2) {return
p1->y<p2->y;}
double min(double a,double b){return
a<b?a:b;}
//-------核心代码----------//
double closest(int s,int e){
    if(s+1==e)
        return get_dis(px[s],px[e]);
    if(s+2==e)
        return
min(get_dis(px[s],px[s+1]),min(get_dis(px
[s+1],px[e]),get_dis(px[s],px[e])));
    int mid=(s+e)>>1;
    double
ans=min(closest(s,mid),closest(mid+1,e));
//递归求解
    int i,j,cnt=0;
    for(i=s;i<=e;i++){//把x坐标在
px[mid].x-ans~px[mid].x+ans范围内的点取出
来

    if(px[i]->x>=px[mid]->x-ans&&px[i]->x
<=px[mid]->x+ans)
            py[cnt++]=px[i];
```

```
    }
    sort(py,py+cnt,cmpy);//按y坐标排序
    for(i=0;i<cnt;i++){
        for(j=i+1;j<cnt;j++){//py数组中的点是按照y坐标升序的
            if(py[j]->y-py[i]->y>=ans)
                break;

    ans=min(ans,get_dis(py[i],py[j]));
        }
    }
    return ans;
}
int main(){
    int i,n;
    while(scanf("%d",&n)!=EOF){
        if(n==0)
            break;
        for(i=0;i<n;i++){

    scanf("%lf%lf%lf",&point[i].x,&point[i].y,&point[i].z);
            px[i]=&point[i];
        }
        sort(px,px+n,cmpx);
        double distance=closest(0,n-1);
        printf("%.2lf\n",distance);
    }
    return 0;
}
```

最小覆盖圆

```
struct point{ double x;double y;}map[100005],o,o1,o2,o3;
struct line{struct point a,b;};
struct point intersection(struct line u,struct line v){
    struct 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;
}
double dis(struct point x,struct point y){
    return
sqrt( (x.x-y.x)*(x.x-y.x)+(x.y-y.y)*(x.y-y.y) );
}
struct point circumcenter(struct point a,struct point b,struct point c){
    struct line u,v;
    struct point o1,o2,o3,ans;
    double r1,r2,r3;
    bool p=0;
    o1.x=(b.x+c.x)/2;
    o1.y=(b.y+c.y)/2;
    r1=dis(b,c)/2;
    if(dis(o1,a)<=r1){
        ans=o1;
        p=1;
    }
    o2.x=(a.x+c.x)/2;
    o2.y=(a.y+c.y)/2;
    r2=dis(a,c)/2;
    if(dis(o2,b)<=r2){
        ans=o2;
        p=1;
    }
    o3.x=(b.x+a.x)/2;
    o3.y=(b.y+a.y)/2;
    r3=dis(b,a)/2;
    if(dis(o3,c)<=r3){
        ans=o3;
        p=1;
    }
    if(p==0){
        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);
    }
    else return ans;
}
double maxzhi(double x,double y){
    if(x>y) return x;
    else return y;
}
int main(){
    int i,j,m,n,p1,p2,p3,xuhao,xuanze;
    double maxdis,r,r1,r2,r3;
    while(scanf("%d",&n)!=EOF && n){
        for(i=1;i<=n;i++)

    scanf("%lf%lf",&map[i].x,&map[i].y);
        if(n>2){
            p1=1;
            p2=2;
            p3=3;

    o=circumcenter(map[p1],map[p2],map[p3
]);

    r=maxzhi(dis(o,map[p1]),dis(o,map[p2]
));
            while(1){
                maxdis=r;
                for(i=1;i<=n;i++){

    if(dis(o,map[i])>maxdis){

    maxdis=dis(o,map[i]);
                        xuhao=i;
                    }
                }
                xuanze=0;
                if(maxdis>r){

    o1=circumcenter(map[p2],map[p3],map[x
uhao]);

    r1=maxzhi( dis(o1,map[p2]),dis(o1,map
[p3]) );

    if(dis(map[p1],o1)<=r1){
                        o=o1;
                        r=r1;
                        xuanze=1;
                    }

    o2=circumcenter(map[p1],map[p3],map[x
uhao]);

    r2=maxzhi( dis(o2,map[p1]),dis(o2,map
[p3]) );

    if(dis(map[p2],o2)<=r2 &&
(r2<r||xuanze==0) ){
                        o=o2;
                        r=r2;
                        xuanze=2;
                    }

    o3=circumcenter(map[p1],map[p2],map[x
uhao]);

    r3=maxzhi( dis(o3,map[p1]),dis(o3,map
[p2]) );

    if(dis(map[p3],o3)<=r3 &&
(r3<r||xuanze==0) ){
                        o=o3;
                        r=r3;
                        xuanze=3;
                    }

    if(xuanze==1)p1=xuhao;
                    else
if(xuanze==2)p2=xuhao;
                    else p3=xuhao;
                }
                else break;
            }
        }
        else{
```

```
                r=dis(map[1],map[2])/2;
            }
            printf("%.2lf\n",r);
        }
}
```

最小覆盖矩阵

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <vector>
using namespace std;
const double eps=1e-6;
const double inf=1e10;
int dcmp(double x)
{
    if(fabs(x)<eps) return 0;
    else return x<0?-1:1;
}
struct point
{
    double x,y;
    point(double x=0,double y=0):x(x),y(y){}
};
point operator-(point a,point b){return
point(a.x-b.x,a.y-b.y);}
point operator+(point a,point b){return
point(a.x+b.x,a.y+b.y);}
point operator*(point a,double p){return
point(a.x*p,a.y*p);}
bool operator<(const point& a,const point& b)
{
    return a.x<b.x||(a.x==b.x&&a.y<b.y);
}
bool operator==(const point& a,const point& b)
{
    return
dcmp(a.x-b.x)==0&&dcmp(a.y-b.y)==0;
}
double cross(point a,point b){return
a.x*b.y-a.y*b.x;}
double dot(point a,point b){return
a.x*b.x+a.y*b.y;}
double length(point a){return sqrt(dot(a,a));}
point normal(point a)
{
    double l=length(a);
    return point(a.x/l,a.y/l);
}
double distoline(point p,point a,point b)
{
    point v1=b-a,v2=p-a;
    return fabs(cross(v1,v2))/length(v1);
}
vector<point> p;
double mina,minp;
vector<point> convex(vector<point>& p)
{
    sort(p.begin(),p.end());
    int n=p.size();
    vector<point> ch(n+1);
    int m=0;
    for(int i=0;i<n;i++)
    {

while(m>1&&cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])
<=0) m--;
        ch[m++]=p[i];
    }
    int k=m;
    for(int i=n-2;i>=0;i--)
    {

while(m>k&&cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])
<=0) m--;
        ch[m++]=p[i];
    }
    if(n>1) m--;
    ch.resize(m);
    return ch;
}
void rotating_calipers(vector<point>& points)
{
    vector<point> p=convex(points);
    int n=p.size();
```

```
        p.push_back(p[0]);
        mina=inf;minp=inf;
        int l=1,r=1,u=1;
        for(int i=0;i<n;i++)
        {
            point edge=normal(p[(i+1)%n]-p[i]);

while(dot(edge,p[r%n]-p[i])<dot(edge,p[(r+1)%n]-
p[i])) r++;

while(u<r||cross(edge,p[u%n]-p[i])<cross(edge,p[(
u+1)%n]-p[i])) u++;

while(l<u||dot(edge,p[l%n]-p[i])>dot(edge,p[(l+1)
%n]-p[i])) l++;
            double
w=dot(edge,p[r%n]-p[i])-dot(edge,p[l%n]-p[i]);
            double
h=distoline(p[u%n],p[i],p[(i+1)%n]);
            mina=min(mina,w*h);
            minp=min(minp,2*(w+h));
        }
}
int main()
{
        int n;
        while(~scanf("%d",&n))
        {
            p.clear();
            for(int i=0;i<n;i++)
            {
                double x,y;
                scanf("%lf%lf",&x,&y);
                p.push_back(point(x,y));
            }
            rotating_calipers(p);
            printf("%.6f\n",minp);
        }
        return 0;
}
```

数论

对于一个大于 1 正整数 n 可以分解质因数：n=p1^a1*p2^a2*p3^a3*…*pk^ak,
则 n 的正约数的个数就是(a1+1)(a2+1)（a3+1）…(ak+1).
斯特林公式：n!=sqrt(2*3.14*n)*(n/e)^n
整数的唯一分解定理：任意正整数都有且只有一种方式写出其素因子的乘积表达式。
A=(p1^k1)*(p2^k2)*(p3^k3)*....*(pn^kn)
其中 pi 均为素数
约数和公式：对于已经分解的整数 A=(p1^k1)*(p2^k2)*(p3^k3)*....*(pn^kn)
有 A 的所有因子之和为：S = (1+p1+p1^2+p1^3+...p1^k1) * (1+p2+p2^2+p2^3+….p2^k2) * (1+p3+p3^3+ … +p3^k3) * .... * (1+pn+pn^2+pn^3+...pn^kn)（展开理解）
同余模公式：(a+b)%m=(a%m+b%m)%m
(a*b)%m=(a%m*b%m)%m
n 的环排列的个数与 n-1 个元素的排列的个数相等。
组合数公式：c[i][j]=c[i-1][j-1]+c[i-1][j];(c[i][0]=1,c[i][1]=i)
stirling 数（将 n 个人分成 k 个组，每组内按特定方式围圈的分组方式）公式：s[i][j]=s[i-1][j-1]+(i-1)*s[i-1][j];（s[1][1]=1,s[i][0]=0）
M 个小球放入 N 个盒子中 C[N+M-1][N-1]
卡特兰数：
C[i+1]=(2*(2i+1)*C[i])/(i+2);
C[i]=for(j=0;j<=i-1;j++)
C[i]+=C[j]*C[i-1-j];
A^B mod C=(A mod C)^(B mod Phi(C)) mod C

求 A^B%C 1.0(A,B 为高精度，C 为 int)

```
/*a,b为高精度，c为int.公式：A^B%C = A^(B%phi(C)+phi(C))%C*/
#define LL long long
#define nnum 1000005
#define nmax 31625
int flag[nmax], prime[nmax] ,plen;
char cha[nnum], chb[nnum];
void mkprime() {
    int i, j;
    memset(flag, -1, sizeof(flag));
    for (i = 2, plen = 0; i < nmax; i++) {
        if (flag[i]) prime[plen++] = i;
        for (j = 0; (j < plen) && (i * prime[j] < nmax); j++){
            flag[i * prime[j]] = 0;
            if (i % prime[j] == 0) break;
        }
    }
}
int getPhi(int n){
    int i, te, phi;
    te = (int) sqrt(n * 1.0);
    for (i = 0, phi = n; (i < plen) && (prime[i] <= te); i++) {
        if (n % prime[i] == 0) {
            phi = phi / prime[i] * (prime[i] - 1);
            while (n % prime[i] == 0) n /= prime[i];
        }
    }
    if (n > 1) phi = phi / n * (n - 1);
    return phi;
}
int cmpBigNum(int p, char *ch)
{
    int i, len;
    LL res;
    len = strlen(ch);
    for (i = 0, res = 0; i < len; i++) {
        res = (res * 10 + (ch[i] - '0'));
```

```
        if (res > p) return 1;
    }
    return 0;
};
int getModBigNum(int p, char *ch) {
    int i, len;
    LL res;
    len = strlen(ch);
    for (i = 0, res = 0; i < len; i++)
        res = (res * 10 + (ch[i] - '0')) %
p;
    return (int) res;
};

int modular_exp(int a, int b, int c) {
    LL res, temp;
    res = 1 % c, temp = a % c;
    while (b) {
        if (b & 1)
            res = res * temp % c;
        temp = temp * temp % c;
        b >>= 1;
    }
    return (int) res;
};
void solve(int a, int c, char *ch) {
    int phi, res, b;
    phi = getPhi(c);
    if (cmpBigNum(phi, ch))
        b = getModBigNum(phi, ch) + phi;
    else
        b = atoi(ch);
    res = modular_exp(a, b, c);
    printf("%d\n", res);
};
int main() {
    int a, c;
    mkprime();
    while (~scanf("%s %s %d", cha, chb, &c))
{
        a = getModBigNum(c, cha);
        solve(a, c, chb);
    }
    return 0;
}
```

求 A^B%C 2.0(A,B,C 均为 long long)

```
//a,b,c小于2^63
// 二分求: (A*B)%C
unsigned long long Multi(unsigned long long
a,unsigned long long b,unsigned long long
n){
    unsigned long long back=0,temp = a%n;
    while (b){
        if ( (b&1) && ((back+=temp)>=n) )
back -= n;
        if ((temp<<=1) >= n) temp -= n;
        b>>=1;
    }
    return back;
};
//二分求: A^B%C
unsigned long long Pow(unsigned long long a,
unsigned long long b, unsigned long long c){
    unsigned long long ans = 1;
    while (b!=0){
        if (b&1) ans = Multi(ans, a, c);
        a = Multi(a, a, c);
        b >>= 1;
    }
    return ans;
};
int main(){
    unsigned long long a, b, c;
    while (scanf("%llu%llu%llu", &a, &b,
&c)!=EOF)
        printf("%llu\n", Pow(a, b, c));
    return 0;
}
```

整数分解（判断质数）

```
struct prime{
    unsigned __int64 num;
    int sum;
}no[10010];
```

```
int sum;
unsigned __int64 factor[10010];
unsigned __int64 mul_mod(unsigned __int64
a,unsigned __int64 b,unsigned __int64 n){
    unsigned __int64 exp = a % n;
    unsigned __int64 res = 0;
    while(b){
        if(b&1){
            res += exp;
            if(res > n) res -= n;
        }
        exp <<= 1;
        if(exp>n) exp -= n;
        b >>= 1;
    }
    return res;
}
unsigned __int64 exp_mod(unsigned __int64
a,unsigned __int64 b,unsigned __int64 c){
    unsigned __int64 k = 1;
    while(b){
        if(b & 1) k = mul_mod(k, a, c);
        a = mul_mod(a, a, c);
        b >>= 1;
    }
    return k;
}

bool miller_rabbin(unsigned __int64 n,
unsigned __int64 times){
    if(n == 2) return 1;
    if(n < 2 || !(n & 1)) return 0;
    unsigned __int64 a, u = n - 1, x, y;
    int t = 0;
    while((u&1) == 0){
        ++ t;
        u >>= 1;
    }
    for(int i=0;i<times;i++){
        a = rand() % (n-1) + 1;
        x = exp_mod(a, u, n);
        for(int j=0;j<t;j++){
            y = mul_mod(x, x, n);
            if(y == 1 && x != 1 && x != n-1)
                return false;
            x = y;
        }
        if(y != 1) return false;
    }
    return true;
}
unsigned __int64 gcd(unsigned __int64
a,unsigned __int64 b){
    unsigned __int64 t;
    while(b){
        t = a % b;
        a = b;
        b = t;
    }
    return a;
};

unsigned __int64 Pollard_rho(unsigned
__int64 x,unsigned __int64 c){
    unsigned __int64 i = 1 ,k = 2;
    unsigned __int64 x0 = rand() % (x - 1)
+ 1;
    unsigned __int64 y = x0;
    while(1){
        i++;
        x0 = ( mul_mod(x0,x0,x) + c) % x;
        unsigned __int64 d = gcd((y - x0 +
x) % x,x);
        if(d > 1 && d < x) return d;
        if(y==x0) return x;
        if(i==k){
            y = x0;
            k <<= 1;
        }
    }
};
void findfac(unsigned __int64 n ,unsigned
__int64 c){
    if(n==1) return;
    if(miller_rabbin(n,6)){
        factor[sum++] = n;
        return;
    }
```

```
    unsigned __int64 p = n;
    unsigned __int64 k = c;
    while(p>=n)
        p = Pollard_rho(p,c--);
    findfac(p,k);
    findfac(n/p,k);
};
int main(){
    unsigned __int64 n,t;
    unsigned __int64 ans;
    scanf("%I64u",&t);
    while(t--){
        scanf("%I64u",&n);
        sum = 0;
        findfac(n,120);
        sort(factor,factor+sum);
        no[0].num = factor[0];
        no[0].sum = 1;
        int index = 0;
        for(int i = 1;i<sum;i++){
            if(factor[i]!=factor[i-1]){
                index++;
                no[index].num = factor[i];
                no[index].sum = 1;
            }
            else no[index].sum++;
        }
         if(sum==1)
             printf("Prime\n");
         else
             for(int i = 0;i<=index;i++)

    printf("%I64d %d\n",no[i].num,no[i].s
um);
    }
    return 0;
}
```

求单个数的欧拉值

```
unsigned euler(unsigned x){// 就是公式
    unsigned i, res=x;
    for (i = 2; i < (int)sqrt(x * 1.0) + 1;
i++)
        if(x%i==0) {
            res = res / i * (i - 1);
            while (x % i == 0) x /= i; //
保证i一定是素数
        }
        if (x > 1) res = res / x * (x - 1);
        return res;
}
```

快速 GCD

```
int gcd(int a, int b)
{
    while ( b ^= a ^= b ^= a %= b );
    return a;
}
```

快速幂

```
long long quickpow(long long m,long long
n,long long k){
    long long b = 1;
    while (n > 0){
        if (n & 1)
            b =( (b%k)*(m%k) )%k;
        n = n >> 1 ;
        m = ( (m%k)*(m%k) )%k;
    }
    return b;
}
```

错排公式

no[n]表示 n 个球放入 n 个盒子里，相同号不放入相同号箱子的方法

```
no[0]=0;
no[1]=0;
no[2]=1;
for (int i = 3; i <= 100;i++)
{
    no[i] = (i - 1) * (no[i-1] + no[i-2]) % MOD;
```

```
}
```

0 到 n 相异或值

```
int xorUpToK(int k) {
        switch (k % 4) {
                case 0: return k;
                case 1: return 1;
                case 2: return k + 1;
                case 3: return 0;
        }
}
```

组合数公式

```
for(int i = 0;i <= x;i++)
{
    c[i][0] = c[i][i] = 1;
    for(int j = 1;j < i;j++)
    {
        c[i][j] = c[i-1][j] + c[i-1][j-1];
    }
}
```

大组合数对素数取模

```
typedef __int64 ll;
const int maxn = 100005;
const ll mod = 1000000009;
ll factorial[2*maxn];

void init(){
    factorial[0] = 1;
    for(int i = 1;i< 2*maxn;i++)
        factorial[i] = (factorial[i-1] * i) %
mod;
}

ll powmod(ll a, ll m, ll n){
    ll ret = 1;
    a %= n;
    while(m){
        if(m&1) ret = (ret * a) % n;
        a = (a * a) % n;
        m >>= 1;
    }
    return ret;
}

ll calCmn(ll p, ll q){
    ll x = q, y = p - q;
    ll tmp = (factorial[p] % mod) *
powmod(factorial[y]*factorial[x], mod-2,
mod) % mod;
    return tmp;
}
```

线性筛素数

```
long long prime[N], flag[N];
long long count;//素数个数
void getprime(long long n){
    long long i,j;
    memset(flag,0,sizeof(flag));
    for(i=2;i<=n;i++){
        //未标记的就是素数了
        if(!flag[i]) prime[++count]=i;
        for(j = 1;j<=count&&i*prime[j]<=n;
j++){
            flag[i * prime[j]] = 1;//prime[j]
都是素数
            if(i % prime[j]==0)
                break;
        }
    }
}
```

容斥原理

```
#include<stdio.h>
#include<string.h>
int a[20], N, M, la ;
int hash[20];
__int64 gcd(__int64 a, __int64 b){
```

```
    __int64 c ;
    while(b){
        c = a % b ;
        a = b ;
        b = c ;
    }
    return a ;
}
//当前点已经加入容斥的个数，记录容斥的过程值结果
void dfs(int now, int count, __int64 lcm, __int64 &sum){
    lcm = a[now] / gcd(lcm, a[now]) * lcm ;
    if(count & 1)
        sum += (N - 1) / lcm ;
    else sum -= (N - 1) / lcm ;
    for(int i = now + 1; i < M; i++)
        dfs(i, count + 1, lcm, sum);
}
int main(){
    int b,i ;
    while(scanf("%d%d", &N, &M) != EOF){
        for(i = 0, la = 0; i < M; i++){
            scanf("%d", &b);
            if(b) a[la++] = b ;
        }
        M = la ;
        __int64 sum = 0 ;
        //memset(hash,0,sizeof(hash));
        for(i = 0; i < M; i++) //枚举起点
            dfs(i, 1, a[i], sum);
        printf("%I64d\n", sum);
    }
    return 0 ;
}
```

高斯消元

```
const int maxn = 105;
int equ, var; // 有 equ 个方程，var 个变元。增广阵行数为 equ，分别为 0 到 equ - 1，列数为 var + 1，分别为 0 到 var.
int a[maxn][maxn];
int x[maxn]; // 解集.
bool free_x[maxn]; // 判断是否是不确定的变元.
int free_num;
void Debug(void)
{
	int i, j;
	for (i = 0; i < equ; i++)
	{
		for (j = 0; j < var + 1; j++)
		{
			cout << a[i][j] << " ";
		}
		cout << endl;
	}
	cout << endl;
}
inline int gcd(int a, int b)
{
	int t;
	while (b != 0)
	{
		t = b;
		b = a % b;
		a = t;
	}
	return a;
}
inline int lcm(int a, int b)
{
	return a * b / gcd(a, b);
}
// 高斯消元法解方程组(Gauss-Jordan elimination).(-2 表示有浮点数解，但无整数解，-1 表示无解，0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数)
int Gauss(void)
{
	int i, j, k;
```

```
int max_r; // 当前这列绝对值最大的行.
int col; // 当前处理的列.
int ta, tb;
int LCM;
int temp;
int free_x_num;
int free_index;
// 转换为阶梯阵.
col = 0; // 当前处理的列.
for (k = 0; k < equ && col < var; k++, col++)
{ // 枚举当前处理的行.
    // 找到该 col 列元素绝对值最大的那行与第 k 行交换.(为了在除法时减小误差)
    max_r = k;
    for (i = k + 1; i < equ; i++)
    {
        if (abs(a[i][col]) > abs(a[max_r][col])) max_r = i;
    }
    if (max_r != k)
    { // 与第 k 行交换.
        for (j = k; j < var + 1; j++) swap(a[k][j], a[max_r][j]);
    }
    if (a[k][col] == 0)
    { // 说明该 col 列第 k 行以下全是 0 了，则处理当前行的下一列.
        k--; continue;
    }
    for (i = k + 1; i < equ; i++)
    { // 枚举要删去的行.
        if (a[i][col] != 0)
        {
            LCM = lcm(abs(a[i][col]), abs(a[k][col]));
            ta = LCM / abs(a[i][col]), tb = LCM / abs(a[k][col]);
            if (a[i][col] * a[k][col] < 0) tb = -tb; // 异号的情况是两个数相加.
            for (j = col; j < var + 1; j++)
            {
                a[i][j] = a[i][j] * ta - a[k][j] * tb;
            }
        }
    }
}
Debug();
// 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).
for (i = k; i < equ; i++)
{ // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换.
```

```
            if (a[i][col] != 0) return -1;
        }
        // 2. 无穷解的情况: 在 var * (var + 1)的增广阵中出现(0, 0, ..., 0)这样的行，即说明没有形成严格的上三角阵.
        // 且出现的行数即为自由变元的个数.
        if (k < var)
        {
            // 首先，自由变元有 var - k 个，即不确定的变元至少有 var - k 个.
            for (i = k - 1; i >= 0; i--)
            {
                // 第 i 行一定不会是(0, 0, ..., 0)的情况，因为这样的行是在第 k 行到第 equ 行.
                // 同样，第 i 行一定不会是(0, 0, ..., a), a != 0 的情况，这样的无解的.
                free_x_num = 0; // 用于判断该行中的不确定的变元的个数，如果超过 1 个，则无法求解，它们仍然为不确定的变元.
                for (j = 0; j < var; j++)
                {
                    if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
                }
                if (free_x_num > 1) continue; // 无法求解出确定的变元.
                // 说明就只有一个不确定的变元 free_index，那么可以求解出该变元，且该变元是确定的.
                temp = a[i][var];
                for (j = 0; j < var; j++)
                {
                    if (a[i][j] != 0 && j != free_index) temp -= a[i][j] * x[j];
                }
                x[free_index] = temp / a[i][free_index]; // 求出该变元.
                free_x[free_index] = 0; // 该变元是确定的.
            }
            return var - k; // 自由变元有 var - k 个.
        }
        // 3. 唯一解的情况: 在 var * (var + 1)的增广阵中形成严格的上三角阵.
        // 计算出 Xn-1, Xn-2 ... X0.
        for (i = var - 1; i >= 0; i--)
        {
            temp = a[i][var];
            for (j = i + 1; j < var; j++)
            {
                if (a[i][j] != 0) temp -= a[i][j] * x[j];
            }
            if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.
            x[i] = temp / a[i][i];
        }
    return 0;
```

```
}
int main(void)
{
    freopen("Input.txt", "r", stdin);
    int i, j;
    while (scanf("%d %d", &equ, &var) != EOF)
    {
        memset(a, 0, sizeof(a));
    memset(x, 0, sizeof(x));
    memset(free_x, 1, sizeof(free_x)); //  一开始全是不确定的变元.
        for (i = 0; i < equ; i++)
        {
            for (j = 0; j < var + 1; j++)
            {
                scanf("%d", &a[i][j]);
            }
        }
//          Debug();
        free_num = Gauss();
        if (free_num == -1) printf("无解!\n");
    else if (free_num == -2) printf("有浮点数解，无整数解!\n");
        else if (free_num > 0)
        {
            printf("无穷多解! 自由变元个数为%d\n", free_num);
            for (i = 0; i < var; i++)
            {
                if (free_x[i]) printf("x%d  是不确定的\n", i + 1);
                else printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        else
        {
            for (i = 0; i < var; i++)
            {
                printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        printf("\n");
    }
    return 0;
}
```

求和公式

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

图论

最短路

Dijkstra（带输出路径）

```
#define INF 100000000
#define maxn 1001
bool vis[maxn];
int
adj[maxn][maxn],dis[maxn],pre[maxn];//pre
[]记录前驱
int n, m;
void dijkstra(int v){
    int i, j, u , min;
    for(i=0;i<=n;i++){
        dis[i]=adj[v][i];
        vis[i]=0;
        //if(i!=v&&adj[v][i]!=INF)pre[i]
= v;
       // else pre[i] = -1;
    }
    vis[v]=1;dis[v]=0;
    for(i=1;i<n;i++){
        min = INF;
        for(j=1;j<=n;j++)
            if(!vis[j]&&min > dis[j]){
                min = dis[j];
                u = j;
            }
        if(min == INF)break;
        vis[u]=1;
        for(j=1;j<=n;j++)

if(!vis[j]&&adj[u][j]!=INF&&dis[u]+adj[u]
[j]<dis[j]){
                dis[j] = adj[u][j] +
dis[u];
              //  pre[j] = u;
            }
    }
}
int main()
{
    int i, j, x, y, w;
    while(~scanf("%d%d",&n,&m)&&n)
    {
        for(i=0;i<=n;i++)
        {
            for(j=0;j<=n;j++)
                if(i==j)adj[i][j]=0;
                else adj[i][j] = INF;
        }
        while(m--)
        {
            scanf("%d%d%d",&x,&y,&w);
            adj[x][y] = w;
            adj[y][x] = w;
        }
        dijkstra(0);
        printf("%d\n",dis[n]);  //以下为
输出路径
        /*int p, len=0, ans[maxn];
        p = n-1;
        while(p!=0)
        {
            ans[len++] = p;
            p = pre[p];

        }
        printf("0->");
        for(i=len-1;i>=0;i--)
            printf("%d",ans[i]);
        puts("");  */
    }
    return 0;
}
```

Floyd

```
        //floyd求出了各点之间的距离
        for(k = 1;k<=n;k++){
            for(i = 1;i<=n;i++){
                for(j = 1;j<=n;j++){
```

```
    if(dis[i][j]>dis[i][k] + dis[k][j]){
                        dis[i][j] =
dis[i][k] + dis[k][j];
                    }
                }
            }
        }
```

SPFA(邻接表)

```
#define INF 9999999
using namespace std;
struct EDGE{
    int v ,w ,next;
}edge[10010];
int head[10010] ,k;
int n ,m ,s ,e;
int dis[1010] ,vis[1010] ,cnt[1010];
void build_edge(int u ,int v ,int w){//无
向图
    edge[k].v = v;
    edge[k].w = w;
    edge[k].next = head[u];
    head[u] = k++;
    edge[k].v = u;
    edge[k].w = w;
    edge[k].next = head[v];
    head[v] = k++;
};
void build_edge1(int u ,int v ,int w){//
有向图
    edge[k].v = v;
    edge[k].w = w;
    edge[k].next = head[u];
    head[u] = k++;
};
bool spfa(){
    memset(vis,0,sizeof(vis));
    memset(cnt,0,sizeof(cnt));
    for(int i = 0;i<=n;i++)
        dis[i] = INF;
    dis[s] = 0;
    priority_queue<pair<int ,int> > q;
    q.push(make_pair(0,s));
    vis[s] = 1;
    cnt[s]++;
    int temp;
    while(!q.empty()){
        temp = q.top().second;
        q.pop();
        vis[temp] = 0;
        for(int i = head[temp];i!=-1;i =
edge[i].next){
            if(dis[edge[i].v] - edge[i].w >
dis[temp]){
                dis[edge[i].v] = edge[i].w +
dis[temp];
                if(!vis[edge[i].v]){

    q.push(make_pair(-dis[edge[i].v],edge[i].v));
                    vis[edge[i].v] = 1;
                    cnt[edge[i].v]++;
                    if(cnt[edge[i].v]>=n)
                        return true;//有负
环
                }
            }
        }
    }
    return false;
}
```

最小生成树

Kruskal

```
struct point{
    int u,v;
    int w;
}num[5008];
int p[102], n,i,sum,x,y;;
int cmp(const void *a,const void *b){
    struct point *c,*d;
    c = (struct point *)a;
```

```
    d = (struct point *)b;
    return c->w - d->w;
};
int root(int x){
    if(x==p[x]) return x;
   return p[x]=root(p[x]);
};
int Kruskal(){
    int i;
    for(i=0;i<=n;i++)p[i]=i;

qsort(num,n*(n-1)/2,sizeof(num[0]),cmp);
    sum=0;
    for(i=0;i<n*(n-1)/2;i++){
        x = root(num[i].u);
        y = root(num[i].v);
        if(x!=y){
            sum+=num[i].w;
            p[x] = y;
        }
    }
    return sum;
}
int main(){
    while(scanf("%d",&n),n){
        for(i=0;i<n*(n-1)/2;i++)

scanf("%d%d%d",&num[i].u,&num[i].v,&num[i
].w);
        Kruskal();
        printf("%d\n",sum);
    }
    return 0;
}
```

强连通分量

Tarjan

```
#define MAX 100010
vector<int> p[MAX];
stack<int> s;
int dfn[MAX] ,low[MAX] ,vis[MAX] ,no[MAX];
//no表示每个点在哪个强连通分量中
int index ,flag ,num;//num表示强连通分量的个数

void Tarjan(int x){
    dfn[x] = low[x] = ++index;
    vis[x] = 1;
    s.push(x);
    for(int i = 0;i < p[x].size();i++){
        if(!dfn[p[x][i]]){
            Tarjan(p[x][i]);
            low[x] = min(low[x],low[p[x][i]]);
        }
        else if(vis[p[x][i]])
            low[x] = min(dfn[p[x][i]],low[x]);
    }
    if(dfn[x]==low[x]){
        num++;
        int v;
        while(1){
            v = s.top();
            no[v] = num;
            vis[v] = 0;
            s.pop();
            if(v==x)
                break;
        }
    }
};
int main(){
    int n ,m;
    while(scanf("%d",&n)!=EOF){
        scanf("%d",&m);
        for(int i = 0;i <= n;i++)
            p[i].clear();
        for(int i = 0;i<m;i++){
            int a ,b;
            scanf("%d%d",&a,&b);
            p[a].push_back(b);
        }
        memset(dfn,0,sizeof(dfn));
        memset(low,0,sizeof(low));
        memset(vis,0,sizeof(vis));
        index = 0;
```

```
            flag = 0;
            num = 0;
            for(int i = 1;i<=n;i++)
                if(!dfn[i])
                    Tarjan(i);
        }
        return 0;
}
```

网络流

Dinic

```
#define MAX 200010
#define INF 99999999999999999
#define min(a,b) (a<b?a:b)
struct Edge{
    int st, ed;
    int next;
    int flow;
} edge[MAX*10];
int head[MAX], out[MAX];
int stack[MAX], p[MAX];
int leve[MAX];
int  Count, s, t;
void build_edge ( int u, int v, long long flw ){
    edge[Count].st = u;
    edge[Count].ed = v;
    edge[Count].flow = flw;
    edge[Count].next = head[u];
    head[u] = Count++;
    edge[Count].st = v;
    edge[Count].ed = u;
    edge[Count].flow = 0;
    edge[Count].next = head[v];
    head[v] = Count++;
};
bool BFS(){
    memset(leve,-1,sizeof(leve));
    int front, rear, u, v, i;
    front = rear = 0;
    p[rear++] = s;
    leve[s] = 0;
    while(front!=rear){
        u = p[front++];
        for (i = head[u];i!=-1;i = edge[i].next ){
            v = edge[i].ed;
            if(edge[i].flow > 0&&leve[v]==-1){
                leve[v] = leve[u] + 1;
                p[rear++] = v;
            }
        }
    }
    return leve[t]!=-1;
};
int Dinic (){
    int maxFlow = 0;
    while (BFS()){
        int top = 0, u = s, i;
        for(i = s;i <= t;i++) //此处s为0,t为n+1
            out[i] = head[i];
        while(out[s] != -1){
            if(u==t){
                long long dd = INF;
                for (i = top-1;i >=0;i--)
                    dd = min (edge[stack[i]].flow,dd);
                for (i = top-1;i>=0;i--){
                    edge[stack[i]].flow -= dd;

    edge[stack[i]^1].flow += dd;
                    if (edge[stack[i]].flow==0)
                        top = i;
                }
                maxFlow += dd;
                u = edge[stack[top]].st;
            }
            else if (out[u]!=-1&&edge[out[u]].flow > 0&&leve[u] + 1==leve[edge[out[u]].ed]){
```

```
                stack[top++] = out[u];
                u = edge[out[u]].ed;
            }
            else{
                while
(top>0&&u!=s&&out[u]==-1)
                    u =
edge[stack[--top]].st;
                out[u] =
edge[out[u]].next;
            }
        }
    }
    return maxFlow;
};
int main(){
    int n, m, u, v, w ,a ,b;
    scanf("%d%d",&n,&m);
    s = 0;
    t = n + 1;
    Count = 0;
    memset(head,-1,sizeof(head));
    for(int i = 1;i <= n;i++){
        scanf("%d%d",&a,&b);
        build_edge (s,i,a);
        build_edge (i,t,b);
    }
    while(m--){
        scanf("%d%d%d",&u,&v,&w);
        build_edge (u,v,w);
        build_edge (v,u,w);
    }
    long long maxFlow = Dinic();
    printf("%lld\n",maxFlow);
    return 0;
}
```

费用流

最小费用最大流 MCMF

//建图前运行 init()

```
//节点下标从 0 开始
//加边时运行 add_edge(a,b,c,0,d)表示加一条 a 到 b 的流量为 c 花费为 d 的边（注意花费为单位流量花费）
//特别注意双向边，如果要加的是双向边的话，运行 add_edge(a,b,c,0,d),add_edge(b,a,c,0,d)较好，不要只运行一次 add_edge(a,b,c,c,d),费用会不对。
//求解时代入 MCMF(s,t,n,v1,v2)，表示起点为 s，终点为 t，点数为 n 的图中，最大流为 v1，最大花费为 v2
#define INF 1000000000
const int M = 4 * 10010;//边
const int N = 1010;//点
struct Node{//边，点f到点t，流量为c，费用为w
    int st, ed;
    int flow, cost;
    int next;
}edge[M];
int head[N], dis[N], q[N], pre[N], cnt;
//cnt为已添加的边数，head为邻接表,dis为花费，pre为父亲节点
bool vis[N];
void init(){
    memset(head, -1, sizeof(head));
    cnt = 0;
}

void add_edge(int f, int t, int d1, int d2, int w){
    //f到t的一条边，流量为d1,反向流量d2,花费w,反向边花费-w（可以反悔)
    edge[cnt].st = f;
    edge[cnt].ed = t;
    edge[cnt].flow = d1;
    edge[cnt].cost = w;
    edge[cnt].next = head[f];
    head[f] = cnt++;

    edge[cnt].st = t;
    edge[cnt].ed = f;
    edge[cnt].flow = d2;
    edge[cnt].cost = -w;
    edge[cnt].next = head[t];
```

```
        head[t] = cnt++;
}

bool spfa(int s, int t, int n){
        int i, tmp, l, r;
        memset(pre, -1, sizeof(pre));
        for(i = 0; i < n; ++i) dis[i] = INF;
        dis[s] = 0;
        q[0] = s;
        l = 0, r = 1;
        vis[s] = true;
        while(l != r){
                tmp = q[l];
                l = (l + 1) % (n + 1);
                vis[tmp] = false;
                for(i = head[tmp]; i!=-1; i =
edge[i].next){
                        if(edge[i].flow && dis[edge[i].ed] >
dis[tmp] + edge[i].cost){
                                dis[edge[i].ed] = dis[tmp] +
edge[i].cost;
                                pre[edge[i].ed] = i;
                                if(!vis[edge[i].ed]){
                                        vis[edge[i].ed] = true;
                                        q[r] = edge[i].ed;
                                        r = (r + 1) % (n + 1);
                                }
                        }
                }
        }
        if(pre[t] == -1) return false;
        return true;
}

void MCMF(int s, int t, int n, int &flow, int
&cost){
        //起点s，终点t，点数n，最大流flow，最小
花费cost
        int tmp, arg;
        flow = cost = 0;
        while(spfa(s, t, n)){
                arg = INF;
                tmp = t;
                while(tmp != s){
                        arg = min(arg,
edge[pre[tmp]].flow);
                        tmp = edge[pre[tmp]].st;
                }
                tmp = t;
                while(tmp != s){
                        edge[pre[tmp]].flow -= arg;
                        edge[pre[tmp] ^ 1].flow += arg;
                        tmp = edge[pre[tmp]].st;
                }
                flow += arg;
                cost += arg * dis[t];
        }
}
```

最大费用流

```
const int E = 50010;//边数
const int INF = 0x7fffffff;
const int N = 210;//点数

struct edge
{
        int next,v,flow,cost;
}e[E];
int head[N],cnt;
queue<int> q;
int S,T;//起点、终点
int dis[N],cc[N],visit[N],pre[N],dd[N];

void addedge(int u,int v,int flow,int cost){
        e[cnt].v = v;
        e[cnt].flow = flow;
        e[cnt].cost = cost;
        e[cnt].next = head[u];
        head[u] = cnt ++;
}

void addEdge(int u,int v,int flow,int cost){
        addedge(u,v,flow,cost);
        addedge(v,u,0, -cost);
}
```

```
void init(){
	cnt = 0;
	memset(head,-1,sizeof(head));
}

int spfa(){
	fill(dis,dis + T + 1, -INF);
	dis[S] = 0;
	pre[S] = -1;
	while(!q.empty()) q.pop();
	q.push(S);
	while(!q.empty()) {
		int u = q.front();
		q.pop();
		visit[u] = 0;
		for(int i = head[u]; i != -1; i = e[i].next) {
			if(e[i].flow > 0 && dis[e[i].v] < dis[u] + e[i].cost) {
				dis[e[i].v] = dis[u] + e[i].cost;
				pre[e[i].v] = u;
				cc[e[i].v] = i;
				dd[e[i].v] = e[i].cost;
				if(!visit[e[i].v]) {
					q.push(e[i].v);
					visit[e[i].v] = 1;
				}
			}
		}
	}
	return dis[T] >= 0;
}

int argument(){
	int aug = INF;
	int u,v;
	int ans = 0;
	for(u = pre[v = T]; v != S; v = u, u = pre[v])
		if(e[cc[v]].flow < aug) aug = e[cc[v]].flow;
	for(u = pre[v = T]; v != S; v = u, u = pre[v]) {
		e[cc[v]].flow -= aug;
		e[cc[v] ^ 1].flow += aug;
		ans += dd[v] * aug;
	}
	return ans;
}

void mcmf(){
	memset(visit,0,sizeof(visit));
	while(spfa()) {
		int cost = argument();
		if(ans < 0) break;
		ans -= cost;
	}
	printf("%d\n",ans);
}
```

树的直径

```
struct NODE{
	int v ,w;
	int next;
}nodes[1000010];
int head[1000010] ,cnt;
int n ,m ,maxnode;
long long ans;
int flag[1000010];
int dir[4][2] = {{1,0},{-1,0},{0,-1},{0,1}};
void addedge(int u ,int v ,int w){
	nodes[cnt].v = v;
	nodes[cnt].w = w;
	nodes[cnt].next = head[u];
	head[u] = cnt;
	cnt++;

	nodes[cnt].v = u;
	nodes[cnt].w = w;
	nodes[cnt].next = head[v];
	head[v] = cnt;
	cnt++;
}

void DFS(int x ,int sum){
```

```
    if(sum > ans)
    {
        ans = sum;
        maxnode = x;
    }
    int re;
    flag[x] = 1;
    for(int i = head[x];i!=-1;i = nodes[i].next)
    {
        re = nodes[i].v;
        if(!flag[re])
        {
            DFS(re,sum + nodes[i].w);
        }
    }
}

int main(){
    int u ,v ,w;
    while(~scanf("%d",&n)){
        memset(head,-1,sizeof(head));
        memset(flag,0,sizeof(flag));
        ans = 0;
        cnt = 0;
        for(int i = 1;i < n;i++) {
            scanf("%d%d%d",&u,&v,&w);
            addedge(u,v,w);
        }
        DFS(1,0);
        ans = 0;
        memset(flag,0,sizeof(flag));
        DFS(maxnode,0);
        printf("%lld\n",ans);
    }
    return 0;
}
```

LCA(Tarjan)

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>
#include<cmath>
#include<queue>
#define mem(a,b) memset(a,b,sizeof(a))
#define MAX 40010
using namespace std;
struct NODE
{
    int v ,w;
    int to;
}edges[2*MAX];
int head[MAX] ,cnt;//邻接表存边
struct QUERY
{
    int v;
    int to ,no;
}querys[MAX];
int he[MAX] ,cntq;//邻接表存询问
int n ,m;
bool flag[MAX];//是否遍历
int ancestor[MAX] ,pa[MAX] ,cntpa[MAX];
//祖先，并查集，集合中元素个数
int ans[MAX];
int dist[MAX] ,num[MAX][2];
void addedge(int u ,int v, int w){
    edges[cnt].v = v;
    edges[cnt].w = w;
    edges[cnt].to = head[u];
    head[u] = cnt++;
    edges[cnt].v = u;
    edges[cnt].w = w;
    edges[cnt].to = head[v];
    head[v] = cnt++;
}

void addquery(int u ,int v ,int no){
    querys[cntq].no = no;
    querys[cntq].v = v;
    querys[cntq].to = he[u];
    he[u] = cntq++;
    querys[cntq].no = no;
    querys[cntq].v = u;
    querys[cntq].to = he[v];
    he[v] = cntq++;
}
```

```
int find(int x){
    if(pa[x]==x) return x;
    return pa[x] = find(pa[x]);
}
void merge(int x ,int y){
    int xx = find(x);
    int yy = find(y);
    if(xx != yy){
        if(cntpa[xx] <= cntpa[yy]){
            pa[xx] = yy;
            cntpa[yy] += cntpa[xx];
        }
        else{
            pa[yy] = xx;
            cntpa[xx] += cntpa[yy];
        }
    }
}
void LCA(int u){
    flag[u] = 1;
    ancestor[u] = u;
    for(int i = head[u]; i != -1; i = edges[i].to){
        int re= edges[i].v;
        if(flag[re]) continue;
        dist[re] = dist[u] + edges[i].w;
        LCA(re);
        merge(u, re);
        ancestor[find(u)] = u;
    }
    for(int i = he[u]; i != -1; i =querys[i].to){
        int re = querys[i].v;
        if(flag[re])
            ans[querys[i].no]                    =
ancestor[find(re)];
    }
}

int main(){
    int t ,x ,y ,w;
    scanf("%d",&t);
    while(t--){
        mem(head,-1);
        mem(he,-1);
        cnt = cntq = 0;
        mem(flag,0);
        for(int i = 1; i <= n; i ++){
            pa[i] = i;
            cntpa[i] = 1;
        }
        scanf("%d%d",&n,&m);
        for(int i = 1; i < n; i ++){
            scanf("%d%d%d",&x,&y,&w);
            addedge(x,y,w);
        }
        for(int i = 1; i <= m;i++){
            scanf("%d%d",&x,&y);
            num[i][0] = x;
            num[i][1] = y;
            addquery(x,y,i);
        }
        dist[1] = 0;
        LCA(1);
        for(int i = 1; i <= m; i ++)
            printf("%d\n",dist[num[i][0]]  +
dist[num[i][1]] - 2 * dist[ans[i]]);
    }
    return 0;
}
```

LCA(RMQ)

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>
#include<cmath>
#define mem(a,b) memset(a,b,sizeof(a))
#define MAX 100010
using namespace std;
struct NODE
{
    int v, w;
    int to ,no;
}edges[2*MAX];
int head[MAX] ,cnt;//邻接表
bool vis[MAX];//是否到达
int first[MAX],last[MAX],times;
```

```
//每个点第一次访问时间，最后访问时间，记录时间
int pos[MAX];//访问的编号
int dep[2*MAX];//每个点的深度
int que[2*MAX] ,cou;//dfs 访问顺序
int n, s, q;
int val[MAX] ,path[MAX];
int no[30];
int dp[2*MAX][20];//rmq
void addedge(int u ,int v ,int w ,int no){
    edges[cnt].no = no;
    edges[cnt].v = v;
    edges[cnt].w = w;
    edges[cnt].to = head[u];
    head[u] = cnt;
    cnt ++;
    edges[cnt].no = no;
    edges[cnt].v = u;
    edges[cnt].w = w;
    edges[cnt].to = head[v];
    head[v] = cnt;
    cnt ++;
}

int _min(int i ,int j){
    if(dep[i] < dep[j])
        return i;
    return j;
}
void rmq(){
    int sta;
    for(int i = 0; i <= 20; i++){
        if(cou <= no[i]){
            sta = i;
            break;
        }
    }
    for(int j = 1; j <= sta; j ++){
        int re = 1<<(j - 1);
        //此处如果写 i <= cou，开 2 倍会RE
        for(int i = 1;i + re <= cou;i++){
            dp[i][j] = _min(dp[i][j - 1], dp[i + re][j - 1]);
        }
    }
}
int query(int x ,int y){
    int xx = pos[x];
    int yy = pos[y];
    if(xx > yy)
        swap(xx, yy);
    int len = yy - xx + 1;
    int k = log(len * 1.0) / log(2.0);
    return _min(dp[xx][k], dp[yy - (1<<k) + 1][k]);
}
void DFS(int u ,int depth){
    vis[u] = 1;
    que[++cou] = u;
    dep[cou] = depth;
    dp[cou][0] = pos[u] = cou;
    first[u] = ++times;
    for(int i = head[u]; i!= -1; i = edges[i].to){
        int re = edges[i].v;
        if(vis[re]) continue;
        path[edges[i].no] = re;
        DFS(re, depth + 1);
        que[++cou] = u;
        dep[cou] = depth;
        dp[cou][0] = cou;
    }
    last[u] = times;
}
int sum[MAX];
int lowbit(int x){
    return (x)&(-x);
}
void update(int pos, int x){
    while(pos <= times){
        sum[pos] += x;
        pos += lowbit(pos);
    }
}
int query1(int pos){
    int s = 0;
    while(pos > 0){
        s += sum[pos];
```

```
            pos -= lowbit(pos);
        }
        return s;
}
int main(){
        int x ,y ,w;
        no[0] = 1;
        for(int i = 1; i <= 20; i ++){
            no[i] = no[i - 1] * 2;
        }
        while(~scanf("%d%d%d",&n,&q,&s)){
            mem(head,-1);
            mem(vis,0);
            mem(sum,0);
            cnt = 0;
            cou = 0;
            times = 0;
            for(int i = 1;i < n;i++){
                scanf("%d%d%d",&x,&y,&w);
                addedge(x, y, w, i);
                val[i] = w;
            }
            DFS(s,1);
            rmq();
            for(int i = 1; i < n;i++){
                update(first[path[i]], val[i]);
                update(last[path[i]] + 1, -val[i]);
            }
            int op;
            for(int i = 1; i <= q; i ++){
                scanf("%d",&op);
                if(op == 1){
                    scanf("%d%d",&x,&w);
                    update(first[path[x]], w -
val[x]);
                    update(last[path[x]] + 1,
val[x] - w);
                    val[x] = w;
                }
                else if(op == 0){
                    scanf("%d",&x);
                    int d1 ,d2 ,d3;
                    d1 = query1(first[s]);
                    d2 = query1(first[x]);
                    d3 =
query1(first[que[query(s,x)]]);
                    s = x;
                    printf("%d\n", d1 + d2 - 2
* d3);
                }
            }
        }
        return 0;
}
```

哈密顿回路

```
/*保证有哈密顿回路的情况下，才能这样求。
Dirac 定理：设一个无向图中有 N 个节点，
若所有节点的度数都大于等于 N/2，
则汉密尔顿回路一定存在。
注意，“N/2” 中的除法不是整除，而是实
数除法。*/
void reverse(int s ,int t){
    int temp;
    while(s < t){
        temp = ans[s];
        ans[s] = ans[t];
        ans[t] = temp;
        s++;
        t--;
    }
}

void hamilton(){
    int s = 1 ,t;//s表示起点
    int cnt = 0;
    bool vis[410];
    mem(vis);
    for(int i = 1;i <= n;i++)
        if(map[s][i]){
            t = i;
            break;
        }
    vis[s] = vis[t] = true;
    ans[cnt++] = s;
    ans[cnt++] = t;
```

```
    while(true){
        while(true){
            int i;
            for(i = 1;i <= n;i++)
                if(map[t][i] && !vis[i]){
                    ans[cnt++] = i;
                    vis[i] = true;
                    t = i;
                    break;
                }
            if(i > n)
                break;
        }
        reverse(0,cnt-1);
        swap(s,t);
        while(true){
            int i;
            for(i = 1;i <= n;i++)
                if(map[t][i] && !vis[i]){
                    ans[cnt++] = i;
                    vis[i] = true;
                    t = i;
                    break;
                }
            if(i > n)
                break;
        }
        if(!map[s][t]){
            int i;
            for(i = 1;i < cnt - 2;i++)
                if(map[ans[i]][t] &&
map[s][ans[i+1]])
                    break;
            i++;
            t = ans[i];
            reverse(i,cnt - 1);
        }
        if(cnt==n) return;
        int j ,i;
        for(j = 1;j <= n;j++){
            if(vis[j]) continue;
            for(i = 1;i < cnt - 2;i++)
                if(map[ans[i]][j])
                    break;
            if(map[ans[i]][j])
                break;
        }
        s = ans[i-1];
        t = j;
        reverse(0,i-1);
        reverse(i,cnt - 1);
        ans[cnt++] = j;
        vis[j] = true;
    }
}
```

最小树形图（朱刘算法）

poj 3164 验

```
double map[110][110];
int flag[110] ,pre[110] ,vis[110];
int n;//点数
double zhu_liu(int s)//朱刘算法,s为树根节点
{
    int i ,j ,k;
    pre[s] = s;
    for(i = 0;i < n;i++){
        map[i][i] = INF;
        flag[i] = 0;
    }
    double sum = 0;
    while(1){
        //求最短弧集合E
        for(i = 0;i < n;i++){
            if(flag[i] || i==s) continue;
            pre[i] = i;
            for(j = 0;j < n;j++)
                if(!flag[j] && map[j][i] <
map[pre[i]][i]) pre[i] = j;
            if(pre[i]==i) return -1.0;//不能建
成树形图
        }
        //检查E
        for(i = 0;i < n;i++)
        {
            if(flag[i] || i==s) continue;
            //从当前点开始找环
```

```
                mem(vis,0);
                vis[s] = 1;
                j = i;
                do{
                    vis[j] = 1;
                    j = pre[j];
                }while(!vis[j]);
                //没有找到环
                if(j==s) continue;
                //收缩G中的有向环
                i = j;
                //将整个环的权值保存，累计入原图的最小树形图
                do{
                    sum += map[pre[j]][j];
                    j = pre[j];
                }while(j!=i);
                j = i;
                //对与环上的点有关的边，修改边权
                do{
                    for(k = 0;k < n;k++)
                        if(!flag[k] && map[k][j] < INF && k!=pre[j])
                            map[k][j] -= map[pre[j]][j];
                    j = pre[j];
                }while(j!=i);
                //缩点，将整个环缩成i号点，所有与环上的点有关的边转移到点i
                for(j = 0;j < n;j++){
                    if(j==i) continue;
                    for(k = pre[i];k!=i;k = pre[k]){
                        if(map[k][j] < map[i][j]) map[i][j] = map[k][j];
                        if(map[j][k] < map[j][i]) map[j][i] = map[j][k];
                    }
                }
                //标记环上其他的点为被缩掉
                for(j = pre[i];j!=i;j = pre[j]) flag[j] = 1;
                //当前环缩点结束，形成新的图G1，跳出继续求G1的最小树形图
                break;
            }
            if(i==n){
                for(i = 0;i < n;i++)
                    if(!flag[i] && i!=s) sum += map[pre[i]][i];
                break;
            }
        }
        return sum;
}
```

无向图求割边（含重边，邻接表）

```
#define MAX 10010//点
#define MAXN 100010//边
struct EDGE{
    int v;
    int next;
    int id;
}edges[2*MAXN];
int head[MAX] ,cnt;
int n ,m;
void addedge(int u ,int v ,int id){
    edges[cnt].v = v;
    edges[cnt].next = head[u];
    edges[cnt].id = id;
    head[u] = cnt++;
    edges[cnt].v = u;
    edges[cnt].next = head[v];
    edges[cnt].id = id;
    head[v] = cnt++;
}
bool vis[MAX];
int low[MAX] ,dfn[MAX] ,index1;
int ans[MAXN] ,cou;//割边的编号和个数
void tarjan(int u ,int id){
    low[u] = dfn[u] = ++index1;
    vis[u] = 1;
    for(int i = head[u];i != -1; i = edges[i].next){
        int re = edges[i].v;
```

```
            if(!vis[re]){
                tarjan(re, edges[i].id);
                low[u] = min(low[u],low[re]);
                if(low[re] > dfn[u])
                    ans[cou++] = edges[i].id;
            }
            else if(edges[i].id != id && vis[re] ==
1)
                low[u] = min(low[u], dfn[re]);
        }
}
int main(){
        int t , u ,v;
        scanf("%d",&t);
        while(t--){
            memset(head,-1,sizeof(head));
            cnt = 0;
            scanf("%d%d",&n,&m);
            for(int i = 1;i <= m;i++){
                scanf("%d%d",&u,&v);
                if(u == v) continue;
                addedge(u,v,i);
            }
            memset(vis,0,sizeof(vis));
            memset(dfn,0,sizeof(dfn));
            memset(low,0,sizeof(low));
            index1 = 0;
            cou = 0;
            tarjan(1,-1);
        }
        return 0;
}
```

无向图求割点

1) 如果 u 该点是根节点并且有两个或者两个以上儿子，那么 u 是一个割点
2) 如果 u 不是根节点并且存在它的一个儿子 v,使得 low[v] >= dep[u],那么 u 是割点

数据结构

线段树

线段树（一段数进行加减）

```
#define MAX 1000010
struct Tree{
    int left ,right;
    __int64 sum;
    __int64 lnc;
}tree[MAX];
__int64 num[100010];
void build_tree(int id,int l,int r){
    tree[id].left = l;
    tree[id].right = r;
    tree[id].lnc = 0;
    if (l==r){
        tree[id].sum = num[l];
        return;
    }
    int mid = (l + r)>>1;
    build_tree(id<<1,l,mid);
    build_tree(id<<1+1,mid+1,r);
    tree[id].sum = tree[id<<1].sum +
tree[id<<1+1].sum;
};
void update(int id,int l,int r, __int64
val){
    if
(tree[id].left==l&&tree[id].right==r){
        tree[id].lnc += val;
        return;
    }
    tree[id].sum += ((r - l + 1) * val);
    int mid = (tree[id].left +
tree[id].right)>>1;
    if (r<=mid){
        update(id<<1,l,r,val);
    }
    else if(l>mid){
        update(id<<1+1,l,r,val);
    }
    else{
        update(id<<1,l,mid,val);
        update(id<<1+1,mid+1,r,val);
    }
};
__int64 query(int id,int l,int r){
    if
(tree[id].left==l&&tree[id].right==r){
        return tree[id].sum + (r - l + 1)
* tree[id].lnc;//询问总和
    }
    else{
        tree[id<<1].lnc += tree[id].lnc;
        tree[id<<1+1].lnc +=
tree[id].lnc;
        tree[id].sum += (tree[id].lnc *
(tree[id].right - tree[id].left + 1));
        tree[id].lnc = 0;
    }
    int mid =
(tree[id].left+tree[id].right)>>1;
    if (r<=mid)return query(id<<1,l,r);
    else if(l>mid)return
query(id<<1+1,l,r);
    else return query(id<<1,l,mid) +
query(id<<1+1,mid+1,r);
};
int main(){
    int n ,m ,q;
    while(scanf("%d%d",&n,&m)!=EOF){//有n
个数，进行m次操作
        memset(num,0,sizeof(num));
        memset(tree,0,sizeof(tree));
        for(int i = 1;i<=n;i++){
            scanf("%I64d",&num[i]);
        }
        build_tree(1,1,n);
        for(int i = 0;i<m;i++){
            char c;
            getchar();
            scanf("%c",&c);
            if(c=='C'){//C表示增加
```

```
                int l ,r;
                __int64 val;

    scanf("%d %d %I64d",&l,&r,&val);
                update(1,l,r,val);
            }
            if(c=='Q'){//Q表示询问
                int l ,r;
                scanf("%d%d",&l,&r);

    printf("%I64d\n",query(1,l,r));//询问
l到r区间内的总和
            }
        }
    }
    return 0;
}
```

线段树求逆序数

（将一个数放在最后，逆序数改变量 n-2*a[i]-1（从 0 开始且 n 个数连续））

```
struct Tree{
    int lson ,rson;
    int no;
}tree[5010*3];
int num[5010];
void build_tree(int id ,int left ,int
right){
    tree[id].lson = left;
    tree[id].rson = right;
    tree[id].no = 0;
    if(left==right)
        return;
    int mid = (left + right) / 2;
    build_tree(2*id,left,mid);
    build_tree(2*id+1,mid+1,right);
};
void update(int id ,int pos){
    if(tree[id].lson==tree[id].rson){
        tree[id].no++;
        return;
    }
    int mid = (tree[id].lson +
tree[id].rson) / 2;
    if(pos <= mid) update(2*id,pos);
    else update(2*id+1,pos);
    tree[id].no = tree[2*id].no +
tree[2*id+1].no;
};
int query(int id ,int left ,int right){
    if(tree[id].lson==left &&
tree[id].rson==right)
        return tree[id].no;
    int mid = (tree[id].lson +
tree[id].rson) / 2;
    if(right<=mid)
        return query(2*id,left,right);
    else if(left > mid)
        return query(2*id+1,left,right);
    else
        return query(2*id,left,mid) +
query(2*id+1,mid+1,right);
};
int main(){
    int n;
    long long ans ,min;
    while(~scanf("%d",&n)){
        build_tree(1,0,n-1);
        ans = 0;
        for(int i = 0;i<n;i++){
            scanf("%d",&num[i]);
            ans += query(1,num[i],n-1);
            update(1,num[i]);
        }
        //求以此将最前面的数放到最后面，
逆序数最小值
        min = ans;
        for(int i = 0;i<n-1;i++){
            ans = ans + (n - 2 * num[i] -
1);
            if(ans < min)min = ans;
        }
        printf("%lld\n",min);
    }
    return 0;
}
```

归并求逆序数

```
long long num[500010];
long long c[500010];
long long cnt;
void Mergesort(int l ,int r){
    int mid ,i ,j ,tmp;
    if(r > l + 1){
        mid = (l + r) / 2;
        Mergesort(l,mid);
        Mergesort(mid,r);
        tmp = l;
        for(i = l,j = mid;i < mid&&j < r;){
            if(num[i] > num[j]){
                c[tmp++] = num[j++];
                cnt += mid - i;
            }
            else c[tmp++] = num[i++];
        }
        if(j < r)
            for(;j < r;j++)c[tmp++] =
num[j];
        else
            for(;i<mid;i++)c[tmp++] =
num[i];
        for(i = l;i < r;i++) num[i] = c[i];
    }
};
int main(){
    int n;
    while(scanf("%d",&n),n){
        cnt = 0;
        for(int i = 0;i<n;i++)
            scanf("%lld",&num[i]);
        Mergesort(0,n);
        printf("%lld\n",cnt);
    }
    return 0;
}

i<=n;i++) aa[in[i].order]=i;
        //树状数组求逆序
        memset(c,0,sizeof(c));
```

树状数组

树状数组求逆序数

```
const int maxn=500005;
int n ,aa[maxn]; //离散化后的数组
int c[maxn];    //树状数组
struct Node{
   int v;
   int order;
}in[maxn];
int lowbit(int x){
    return x&(-x);
}
void update(int t,int value){
    int i;
    for(i=t;i<=n;i+=lowbit(i))
c[i]+=value;
}
int getsum(int x){
    int i;
    int temp=0;
    for(i=x;i>=1;i-=lowbit(i))
temp+=c[i];
    return temp;
}
bool cmp(Node a ,Node b){
    return a.v<b.v;
}
int main(){
    int i,j;
    while(scanf("%d",&n)==1 && n){
        //离散化
        for(i=1;i<=n;i++){
            scanf("%d",&in[i].v);
            in[i].order=i;
        }
        sort(in+1,in+n+1,cmp);
        for(i=1;
        long long ans=0;
        for(i=1;i<=n;i++){
            update(aa[i],1);
```

```
            ans+=i-getsum(aa[i]);
        }
        cout<<ans<<endl;
    }
    return 0;
}
```

二维树状数组

```
#define MAX 100005
int N=1005,c[1005][1005];
int lowbit( int x ){
    return x & (-x);
}
void modify( int x, int y, int delta ){//a[x][y]增加
delta
    int i, j;
    for(i=x; i<=N; i+=lowbit(i)){
        for(j=y; j<=N; j+=lowbit(j)){
            c[i][j] += delta;
        }
    }
}
int sum( int x, int y ){
    int res = 0, i, j;
    for(i=x; i>0; i-=lowbit(i)){
        for(j=y; j>0; j-=lowbit(j)){
            res += c[i][j];
        }
    }
    return res;
}
```

扫描线

```
#define MAX 10010//点数
struct NODE{
    int lx ,rx ,y;
    int flag;
}nodes[2*MAX];
int n;//点数
int xx[2*MAX];
int cnt;//去重之后的个数
void deal(){
    cnt = 0;
    int cou = 2 * n;
    for(int i = 1;i < cou;i++){
        if(xx[cnt] != xx[i]){
            cou++;
            xx[cou] = xx[i];
        }
    }
}
int cmp(struct NODE a,struct NODE b){
    if(a.y == b.y)
        return a.flag > b.flag;
    return a.y < b.y;
}
int get_id(int no){
    int left ,right = cnt;
    while(left <= right){
        int mid = (left + right) >> 1;
        if(xx[mid] > no) right = mid + 1;
        else if(xx[mid] < no) left = mid + 1;
        else return mid;
    }
}
struct TREE{
    int left ,right;
    int sum;
}tree[8*MAX];
void build_tree(int id ,int left ,int right){
    tree[id].left = left;
    tree[id].right = right;
    tree[id].sum = 0;
    if(left == right)
        return ;
    int mid = (left + right) >> 1;
    build_tree(id<<1,left,mid);
    build_tree((id<<1)|1,mid + 1,right);
}
void update(int id ,int left ,int right ,int flag){}
int main(){
    int lx ,rx ,dy ,uy;
    while(~scanf("%d",&n)){
        for(int i = 0;i < n;i++){
```

```
scanf("%d%d%d%d",&lx,&dy,&rx,&uy);
                xx[2*i]    =    nodes[2*i].lx    =
nodes[2*i+1].lx = lx;
                xx[2*i+1]    =    nodes[2*i].rx    =
nodes[2*i+1].rx = rx;
                nodes[2*i].y = dy;
                nodes[2*i].flag = 1;
                nodes[2*i+1].y = uy;
                nodes[2*i+1].flag = -1;
            }
            int cou = 2 * n;
            sort(xx,xx+cou);
            deal();
            sort(nodes,nodes+cou,cmp);
            build_tree(1,0,cnt - 1);
            for(int i = 0;i < cou - 1;i++){
                int left = get_id(nodes[i].lx);
                int right = get_id(nodes[i].rx);

update(1,left,right,nodes[i].flag);
                ans    +=    tree[1].sum    *
(nodes[i+1].y - nodes[i].y);
            }
        }
        return 0;
}
```

哈希函数

```
// JS Hash Function
unsigned int JSHash(char *str){
    unsigned int hash = 1315423911;
    while (*str){
        hash ^= ((hash << 5) + (*str++) + (hash >>
2));
    }
    return (hash & 0x7FFFFFFF);
}

// BKDR Hash Function
unsigned int BKDRHash(char *str){
    unsigned int seed = 131;
    // 31 131 1313 13131 131313 etc..
    unsigned int hash = 0;
    while (*str)
        hash = hash * seed + (*str++);
    return (hash & 0x7FFFFFFF);
}
// AP Hash Function
unsigned int APHash(char *str){
    unsigned int hash = 0;
    int i;
    for (i=0; *str; i++){
        if ((i & 1) == 0)
            hash ^= ((hash << 7) ^ (*str++) ^
(hash >> 3));
        else
            hash ^= (~((hash << 11) ^ (*str++) ^
(hash >> 5)));
    }
    return (hash & 0x7FFFFFFF);
}
```

树链剖分

```
#pragma                    comment(linker,
"/STACK:200000000")
#define MAX 500010//点数
using namespace std;
struct EDGE{
    int v;
    int next;
}edges[2 * MAX];
int head[MAX] ,cou;//邻接表

int n ,m ,p;
int num[MAX];

void addedge(int x ,int y){
    edges[cou].v = y;
    edges[cou].next = head[x];
    head[x] = cou++;
    edges[cou].v = x;
    edges[cou].next = head[y];
    head[y] = cou++;
```

```
}
int size[MAX] ,son[MAX] ,fa[MAX] ,dep[MAX];
//儿子个数，重儿子编号，父亲节点编号，深度
int top[MAX] ,pos[MAX] ,no ,ppos[MAX];
/*所在链的顶点，与父亲所在边在线段树中的位置，
线段树大小，线段树编号对应的点*/
//第一次求出 fa,dep,size,son
int dfs1(int u ,int pre ,int depth){
    fa[u] = pre;
    dep[u] = depth;
    size[u] = 1;
    son[u] = -1;
    for(int i = head[u]; i != -1;i = edges[i].next){
        int re = edges[i].v;
        if(re == pre) continue;
        cnt = dfs1(re, u, depth + 1);
        if(son[u] == -1 || size[son[u]] < size[re])
            son[u] = re;
        size[u] += cnt;
    }
    return size[u];
}
//第二次求 top 和 pos
void dfs2(int u ,int tops){
    top[u] = tops;
    pos[u] = no;
    ppos[no] = u;
    no++;
    if(son[u] == -1)
        return;
    dfs2(son[u], tops);
    for(int i = head[u]; i != -1; i = edges[i].next){
        int re = edges[i].v;
        if(re == fa[u] || re == son[u])
            continue;
        dfs2(re, re);
    }
}
struct TREE{
    int sum;
    int left ,right;
    int lazy;
}tree[4*MAX];

void push_down(int id){
    if(tree[id].lazy != 0){
        tree[id<<1].lazy += tree[id].lazy;
        tree[(id<<1)+1].lazy += tree[id].lazy;
        tree[id].lazy = 0;
    }
}
void build_tree(int id ,int left ,int right){
    tree[id].left = left;
    tree[id].right = right;
    tree[id].sum = tree[id].lazy = 0;
    if(left == right){
        tree[id].sum = num[ppos[left]];
        return;
    }
    int mid = (left + right) >> 1;
    build_tree(id<<1, left, mid);
    build_tree((id<<1)|1, mid + 1,right);
}
void update(int id ,int le ,int ri ,int val){
    if(tree[id].left == le && tree[id].right == ri){
        if(le == ri){
            tree[id].sum += val;
            return;
        }
        tree[id].lazy += val;
        return;
    }
    int mid = (tree[id].left + tree[id].right) >> 1;
    push_down(id);
    if(ri <=  mid)
        update(id<<1, le, ri ,val);
    else if(le > mid)
        update((id<<1)|1, le, ri, val);
    else{
        update(id<<1, le, mid ,val);
        update((id<<1)|1, mid + 1, ri, val);
```

```
        }
}
int query(int id ,int pos){
        if(tree[id].left == tree[id].right){
                tree[id].sum += tree[id].lazy;
                tree[id].lazy = 0;
                return tree[id].sum;
        }
        int mid = (tree[id].left + tree[id].right) >>
1;
        push_down(id);
        if(pos <= mid)
                return query(id<<1, pos);
        else
                return query((id<<1)|1, pos);
}

void in_de(int x ,int y ,int val){
        int f1 = top[x] ,f2 = top[y];
        while(f1 != f2){
                if(dep[f1] < dep[f2]){
                        swap(f1, f2);
                        swap(x, y);
                }
                update(1, pos[f1], pos[x], val);
                x = fa[f1];
                f1 = top[x];
        }
        /*对边权更新
        if(x == y)
                return;
        if(dep[x] < dep[y])
                swap(x,y);
        update(1,pos[son[y]],pos[x],val);
        */
        对点权更新
        if(dep[x] < dep[y])
                swap(x, y);
        update(1, pos[y], pos[x], val);
}
int find_num(int x){
        int ans = pos[x];
        return query(1, ans);
}
int main(){
        int x ,y ,w;
        char str[10];
        while(~scanf("%d%d%d", &n, &m, &p)){
                mem(head,-1);
                no = cou = 0;
                for(int i = 1;i <= n; i ++)
                        scanf("%d", &num[i]);
                for(int i = 1;i <= m; i ++){
                        scanf("%d%d", &x, &y);
                        addedge(x, y);
                }
                addedge(n + 1,1);
                dfs1(n + 1, 0, 1);
                dfs2(n + 1, n + 1);
                build_tree(1,0,no - 1);
                for(int i = 0; i < p; i ++){
                        scanf("%s",str);
                        if(str[0] == 'Q'){
                                scanf("%d", &x);

        printf("%d\n",find_num(x));
                        }
                        else if(str[0] == 'D'){
                                scanf("%d%d%d",  &x,  &y,
&w);
                                in_de(x, y, -w);
                        }
                        else
                        {
                                scanf("%d%d%d",  &x,  &y,
&w);
                                in_de(x, y , w);
                        }
                }
        }
        return 0;
}
```

splay

```
#define KEY_VALUE ch[ch[root][1]][0]
#define MAX 200010
```

```
using namespace std;

int num[MAX];
int n;
int min1[MAX] ,val[MAX];
int add[MAX] ,flag[MAX];

int pre[MAX];//表示父亲节点
int ch[MAX][2];//表示左右儿子节点
int root;//根节点
int tot;//节点数
int size[MAX];//子树规模
int s[MAX] ,tot1;//内存池
void update_add(int r ,int ADD){
    if(!r) return;
    add[r] += ADD;
    min1[r] += ADD;
    val[r] += ADD;
}
void update_re(int r ,int fl){
    if(!r) return;
    swap(ch[r][0],ch[r][1]);
    flag[r] ^= fl;
}
//一般不会使用 push_down，只有在使用 lazy 标记的时候需要添加
void push_down(int u){
    if(add[u]){
        update_add(ch[u][0],add[u]);
        update_add(ch[u][1],add[u]);
        add[u] = 0;
    }
    if(flag[u]){
        update_re(ch[u][0],flag[u]);
        update_re(ch[u][1],flag[u]);
        flag[u] = 0;
    }
}
//将子树信息更新到父亲节点
void push_up(int u){
    size[u] = size[ch[u][0]] + size[ch[u][1]] + 1;
    //根据题目修改一下
    min1[u] = min(min(min1[ch[u][0]],min1[ch[u][1]]),val[u]);
}
//建立新节点
void newnode(int &r ,int father ,int k){
    if(tot1) r = s[tot1--];
    else r = ++tot;
    pre[r] = father;
    ch[r][0] = ch[r][1] = 0;
    size[r] = 1;
    //以下内容根据题目修改
    val[r] = k;
    min1[r] = k;
    flag[r] = 0;
    add[r] = 0;
}
//旋转操作
void rotate(int u ,int kind){//kind 为 1 是右旋，为 0 是左旋
    int y = pre[u];
    push_down(y);
    push_down(u);
    ch[y][!kind] = ch[u][kind];
    pre[ch[u][kind]] = y;
    if(pre[y])
        ch[pre[y]][ch[pre[y]][1]==y] = u;
    pre[u] = pre[y];
    pre[y] = u;
    ch[u][kind] = y;
    push_up(y);
}
void splay(int r ,int goal){
    push_down(r);
    while(pre[r] != goal){
        if(pre[pre[r]]==goal)
            rotate(r,ch[pre[r]][0]==r);
        else{
            int y = pre[r];
            int kind = ch[pre[y]][0]==y;//如果 y 是父亲节点的左子树为 1，否则为 0
            //判断两个方向是否相同,方向不同
            if(ch[y][kind]==r){
                rotate(r,!kind);
```

```
                rotate(r,kind);
            }
            else{
                rotate(y,kind);
                rotate(r,kind);
            }
        }
    }
    push_up(r);
    if(goal==0) root = r;
}
//把第 k 位的数移动到 goal 下面
void rotateto(int k ,int goal){
    int r = root;
    push_down(r);
    while(size[ch[r][0]] != k){
        if(k < size[ch[r][0]])
            r = ch[r][0];
        else{
            k -= (size[ch[r][0]] + 1);
            r = ch[r][1];
        }
        push_down(r);
    }
    splay(r,goal);
}
void adds(int left ,int right ,int x){
    rotateto(left-1,0);
    rotateto(right+1,root);
    update_add(KEY_VALUE,x);
    push_up(ch[root][1]);
    push_up(root);
}
int query(int left ,int right){
    rotateto(left-1,0);
    rotateto(right+1,root);
    return min1[KEY_VALUE];
}
//建树
void build_tree(int &x ,int left ,int right ,int
father){
    if(left > right) return;
    int mid = (left + right) >> 1;
    newnode(x,father,num[mid]);
    build_tree(ch[x][0],left,mid-1,x);
    build_tree(ch[x][1],mid+1,right,x);
    push_up(x);
}
//初始化
void init(){
    ch[0][0] = ch[0][1] = size[0] = pre[0] = 0;
    min1[0] = INF;
    root = tot = tot1 = 0;
    newnode(root,0,INF);
    newnode(ch[root][1],root,INF);
    size[root] = 2;
    build_tree(KEY_VALUE,0,n-1,ch[root][1]);
    push_up(ch[root][1]);
    push_up(root);
}
//插入操作
void insert(int k ,int x){
    rotateto(k,0);
    rotateto(k+1,root);
    newnode(KEY_VALUE,ch[root][1],x);
    push_up(ch[root][1]);
    push_up(root);
}
//删除操作，将第 k 位的数删除
void erase(int k){
    rotateto(k-1,0);
    rotateto(k+1,root);
    s[++tot1] = KEY_VALUE;
    KEY_VALUE = 0;
    push_up(ch[root][1]);
    push_up(root);
}
void reverse(int left ,int right){
    rotateto(left-1,0);
    rotateto(right+1,root);
    update_re(KEY_VALUE,1);
}
void revolve(int left ,int right ,int t){
    if(!t) return;
    rotateto(right-t,0);
    rotateto(right+1,root);
    push_down(KEY_VALUE);
    int re = KEY_VALUE;
```

```
        KEY_VALUE = 0;
        push_up(ch[root][1]);
        push_up(root);
        rotateto(left-1,0);
        rotateto(left,root);
        KEY_VALUE = re;
        if(re) pre[re] = ch[root][1];
        push_up(ch[root][1]);
        push_up(root);
}
char str[110];

int main()
{
        int m ,a ,b ,c;
        while(~scanf("%d",&n))
        {
                for(int i = 0;i < n;i++)
                        scanf("%d",&num[i]);
                init();
                scanf("%d",&m);
                while(m--){
                        scanf("%s",str);
                        if(str[0]=='M'){
                                scanf("%d%d",&a,&b);
                                printf("%d\n",query(a,b));
                        }
                        else if(str[0]=='A'){

scanf("%d%d%d",&a,&b,&c);
                                adds(a,b,c);
                        }
                        else if(str[0]=='D'){
                                scanf("%d",&a);
                                erase(a);
                        }
                        else if(str[0]=='I'){
                                scanf("%d%d",&a,&b);
                                insert(a,b);
                        }
                        else if(str[3]=='E'){
                                scanf("%d%d",&a,&b);
                                reverse(a,b);
                        }
                                else{

scanf("%d%d%d",&a,&b,&c);
                                        c %= (b - a + 1);
                                        revolve(a,b,c);
                                }
                        }
                }
                return 0;
}
```

字符串

KMP

KMP

```
int nextval[1000010];
vector<int> v;
void get_nextval(int len ,char ptrn[]){
    int i = 0 ,j = -1;
    nextval[i] = -1;
    while(i < len-1)  {
        if(j == -1||ptrn[i]==ptrn[j]){
            ++i;
            ++j;

    if(ptrn[i]!=ptrn[j])nextval[i] = j;
            else nextval[i] = nextval[j];
        }
        else j = nextval[j];
    }
};

void kmp_search(char str1[] ,char
str2[] ,int len1 ,int len2){
    int i = 0,  j = 0;
    while(i <= len1){
        if(j==len2){
            v.push_back(i - len2);
            j = nextval[j];
            if(i==len1)break;
        }
        else{
            if(j==-1 ||
str1[i]==str2[j]){
                i++;
                j++;
            }
            else j = nextval[j];
        }
    }
};

int main(){
    char str1[1000010] ,str2[1000010];
    int len1 ,len2;
    scanf("%s%s",str1,str2);
    len1 = strlen(str1);
    len2 = strlen(str2);
    //当需要处理如:在abababab中abab的个数
时，加上一下两行
    str2[len2] = '*';
    len2++;
    str2[len2] = '\0';
    get_nextval(len2 , str2);
    len2--;//对应上面的注释
    kmp_search(str1,str2,len1,len2);
    printf("%d",v.size());
    for(int i = 0;i<v.size();i++)
        printf(" %d",v[i]);
    printf("\n");
    return 0;
}
```

后缀数组

```
int
wa[MAX] ,wb[MAX] ,wv[MAX] ,ws1[MAX] ,sa[
MAX];
int rank1[MAX],height[MAX];
int cmp(int *r,int a,int b,int l)
    return r[a]==r[b]&&r[a+l]==r[b+l];

void da(int *r,int *sa,int n,int m){
    int i,j,p,*x=wa,*y=wb,*t;
    for(i=0;i<m;i++) ws1[i]=0;
    for(i=0;i<n;i++) ws1[x[i]=r[i]]++;
    for(i=1;i<m;i++) ws1[i]+=ws1[i-1];
    for(i=n-1;i>=0;i--) sa[--ws1[x[i]]]=i;
    for(j=1,p=1;p<n;j*=2,m=p){
        for(p=0,i=n-j;i<n;i++) y[p++]=i;
        for(i=0;i<n;i++) if(sa[i]>=j)
y[p++]=sa[i]-j;
        for(i=0;i<n;i++) wv[i]=x[y[i]];
```

```
            for(i=0;i<m;i++) ws1[i]=0;
            for(i=0;i<n;i++) ws1[wv[i]]++;
            for(i=1;i<m;i++) ws1[i]+=ws1[i-1];
            for(i=n-1;i>=0;i--)
sa[--ws1[wv[i]]]=y[i];

    for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)

    x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
    }
    return;
}
void calheight(int *r,int *sa,int n){
    int i,j,k=0;
    for(i=1;i<=n;i++)
        rank1[sa[i]]=i;
    for(i=0;i<n;height[rank1[i++]]=k)

    for(k?k--:0,j=sa[rank1[i]-1];r[i+k]==r[j+k];k+
+);
    return;
}
num[1...n-1];
num[n] = 0;
da(num,sa,n+1,1000001);
calheight(num,sa,n);
```

动态规划

最长非上升子序列

```
//时间复杂度O（nlogn）
int LDesS(int a[], int n) {
     int i, r, l, len, m ,order[1010];
      memset(order,0,sizeof(order));
     len = 1;
     for (i = 0; i < n; ++i) {
         l = 1;
         r = len;
         while (l <= r){
             m = (l + r) >> 1;
              if (order[m] >= a[i])//改为>,
则是最长下降子序列
                  l = m + 1;
             else r = m - 1;
         }
          if (order[l] < a[i]) order[l] =
a[i];
         len = max(len, l);
     }
     return len;
 };

int main(){
     int n ,num[1010] ,Max;
     scanf("%d",&n);//n个数
     for(int i = 0;i<n;i++)
         scanf("%d",&num[i]);
     Max = LDesS(num,n);
     printf("%d\n",Max);
}
```

最长非下降子序列

```
//时间复杂度O（nlogn）
#define MAX 1010
int a[MAX] ,k;
void find(int x,int y,int b){
     int mid;
     int f;
     while(x < y){
         mid = ( x + y ) / 2;
         if(a[mid] >= b){
             f = mid;
             y = mid;
         }
         else x = mid + 1;
     }
     if(a[x]>=b) f = x;
     a[f] = b;
}
int main(){
     int i,n,b;
     while(scanf("%d",&n)!=-1){
         memset(a,0,sizeof(a));
         k = 0;
         for(i = 1;i <= n;i++){
             scanf("%d",&b);
             if(b >= a[k])//改为>,则是最长
上升子序列
                 a[++k]=b;
             else find(1,k,b);
         }
         printf("%d\n",k);
     }
     return 0;
}
```

最长公共子序列

```
vector<char> common;
int lcs(string A,string B){
     vector<vector<int> > len;
     len.resize(A.size()+1);
     for(int i = 0;i <= A.size();i++)
         len[i].resize(B.size()+1,0);
     for (int i = 1; i <= A.size(); ++i){
         for (int j = 1; j <= B.size(); ++j){
             if (A[i-1] == B[j-1])
                 len[i][j] = len[i-1][j-1]
+ 1;
```

```
            else if (len[i-1][j] >=
len[i][j-1])
                len[i][j] = len[i-1][j];
            else
                len[i][j] = len[i][j-1];
        }
    }
    int apos = A.size();
    int bpos = B.size();
    int commonlen = len[apos][bpos];
    int k = commonlen;
    common.resize(commonlen);
    while(apos && bpos){
        if(len[apos][bpos] ==
len[apos-1][bpos] + 1){
            common[--k] = A[--apos];
            --bpos;
        }
        else if (len[apos-1][bpos] >=
len[apos][bpos-1])
            --apos;
        else
            --bpos;
    }
    for(int i = 0;i < commonlen;i++)
        cout<<common[i];
    cout<<endl;
    return commonlen;
}
int main()
{
    string A = "abcdss";
    string B = "asbda";
    cout<<lcs(A,B);
    return 0;
}
```

最长公共子串

```
//  查找公共子串
//  lcs记录公共子串
//  return   公共子串长度
int LCS(const char *str1  , int len1 , const
char *str2 , int len2 , char *&lcs){
    if(NULL == str1 || NULL == str2)
        return -1;   //空参数
    //  压缩后的最长子串记录向量
    int *c = new int[len2+1];
    for(int i = 0 ; i < len2 ; ++i)
        c[i] = 0;
    int max_len = 0;  //匹配的长度
    int pos = 0;      //在str2上的匹配最
末位置
    for(int i = 0 ; i < len1 ; ++i){
        for(int j = len2 ; j > 0 ; --j){//
更新时从后往前遍历
            if(str1[i] == str2[j-1]){
                c[j] = c[j-1] + 1;
                if(c[j] > max_len){
                    max_len = c[j];
                    pos = j-1;
                }
            }
            else
                c[j] = 0;
        }
    }
    if(0 == max_len)
        return 0;
    //  得到公共子串
    lcs = new char[max_len];
    for(int i = 0 ; i < max_len ; ++i)
        lcs[i] = str2[pos-max_len+1+i];
    cout<<"pos = "<<pos<<endl;
    delete [] c;
    return max_len;
}
//  test
int main(){
    const char *str1 = "abacaba";
    const char *str2 = "caba";
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    char *lcs;
    int len = LCS(str1 , len1 , str2 , len2 ,
lcs);
    cout<<"max length = "<<len<<endl;
```

```
    for(int i = 0 ; i < len ; ++i)
        cout<<lcs[i]<<" ";
```

多重背包

```
struct Cash{
    int n, d;
};
typedef struct Cash C;
C num[11];
int f[100010];
int main(){
    int N, cash;
    int i, j ,k;
    scanf("%d",&cash);
    scanf("%d",&N);
    for(i = 0;i<N;i++)
      scanf("%d%d",&num[i].n,&num[i].d);
    int min = 100000 ,all = 0;
    for(i = 0;i<N;i++){
      if(min>num[i].d)
        min = num[i].d;
      all += num[i].d * num[i].n;
    }
    if(min > cash){
      printf("0\n");
      continue;
    }
    if(all < cash){
      printf("%d\n",all);
      continue;
    }
    memset(f,0,sizeof(f));
    for(i = 0;i<N;i++){
      if(num[i].d * num[i].n==cash)
        f[cash] = cash;
      else if(num[i].d*num[i].n>cash)
        for (k=num[i].d;k<=cash;k++)
          if (f[k-num[i].d]+ num[i].d>f[k])
            f[k]=f[k-num[i].d]+num[i].d;
          else{
            int m = num[i].n;
            for(j=1;j<=num[i].n/2;j*=2){
              for( k=cash;k>=j*num[i].d;k--)
                if (f[k-j*num[i].d]+
                    j*num[i].d>f[k])
                f[k]=f[k-j*num[i].d]+j*num[i].d;
                m -= j;
              }
            for (k=cash;k>=m*num[i].d;k--)
              if(f[k-m*num[i].d]+
                 m*num[i].d>f[k])
              f[k]=f[k-m*num[i].d]+m*num[i].d;
            }
    }
    printf("%d\n",f[cash]);
    return 0;
}
```

泛化背包

```
int n , dp[510][10010] ,max1;
int c[100010] ,u[100010];
vector<int> v[100010];
void solve(int x ,int spend){
    int re;
    for(int i = 0;i < v[x].size();i++){
        re = v[x][i];
        //对于子节点为叶子节点的点，直接进行dp操作合并。
        if(v[re].empty()){
            for(int j = spend;j>=c[re];j--){
                dp[x][j] = max(dp[x][j] ,
                dp[x][j-c[re]] + u[re]);
                if(dp[x][j] > max1)
                    max1 = dp[x][j];
            }
        }
        else{
            for(int j = 0;j<=spend - c[re];j++)
                dp[re][j] = dp[x][j];
            //对dp[re][0~spend-c[re]]更行，算出其子树的一个泛化背包。即花费 j=1~spend-c[re]能获得的最大战斗力
            solve(re,spend-c[re]);
            for(int j = spend;j>=c[re];j--){
                dp[x][j] = max(dp[x][j],
                dp[re][j-c[re]] + u[re]);
                if(dp[x][j] > max1)
                    max1 = dp[x][j];
            }
        }
    }
```

双调 TSP

```
for(int i = 1;i<=n;i++)
{
    for(int j = 1;j<=i;j++)
    {
        dp[i][j] = dp[j][i] = 999999999;
    }
}

dp[2][1] = dp[1][2] = dist[1][2];
for(int j = 3;j<=n;j++)
{
    dp[j][1] = dp[1][j] = dp[1][j-1] + dist[j-1][j];
}
for(int i = 2;i<=n;i++)
{
    double ans = 99999999;
    for(int k = 1;k<i;k++)
    {
        f(ans > dp[i][k] + dist[k][i+1])
        {
            ans = dp[i][k] + dist[k][i+1];
        }
    }
    dp[i+1][i] = dp[i][i+1] = ans;
    for(int j = i + 2;j<=n;j++)
    {
        dp[j][i] = dp[i][j] = dp[i][j-1] + dist[j-1][j];
    }
}
dp[n][n] = dp[n][n-1] + dist[n-1][n];
```

高精度

C 语言实现

```
#define maxsize 100
struct hp{
    int len;
    int s[maxsize+1];
};
//输入
void input(hp &a,string str){
    int i;
    while(str[0]=='0' && str.size()!=1)
        str.erase(0,1);
    a.len=(int)str.size();
    for(i=1;i<=a.len;++i)
        a.s[i]=str[a.len-i]-48;
    for (i=a.len+1;i<=maxsize;++i)
        a.s[i]=0;
}
//输出
void print(const hp &y)
{
    int i;
    for(i=y.len;i>=1;i--)
        cout<<y.s[i];
    cout<<endl;
}
//比较函数 a > b返回正数，a==b返回0，a < b返回负数
int compare(const hp &a,const hp &b){
    int len;
    if(a.len>b.len)
        len=a.len;
    else
        len=b.len;
    while(len>0 && a.s[len]==b.s[len])
len--;
    if(len==0)
        return 0;
    else
        return a.s[len]-b.s[len];
}
//加法，结果保存在c中
void plus(const hp &a,const hp &b,hp &c)
{
    int i,len;
    for(i=1;i<=maxsize;i++) c.s[i]=0;
    if(a.len>b.len) len=a.len;
    else len=b.len;
    for(i=1;i<=len;i++)
    {
        c.s[i]+=a.s[i]+b.s[i];
        if(c.s[i]>=10)
        {
            c.s[i]-=10;
            c.s[i+1]++;
        }
    }
    if(c.s[len+1]>0) len++;
    c.len=len;
}
//减法 结果保存在c中，a是被减数，b是减数，不支持负数运算
void subtract(const hp &a,const hp &b,hp &c)
{
    int i,len;
    for(i=1;i<=maxsize;i++) c.s[i]=0;
    if(a.len>b.len) len=a.len;
    else len=b.len;
    for(i=1;i<=len;i++)
    {
        c.s[i]+=a.s[i]-b.s[i];
        if(c.s[i]<0)
        {
            c.s[i]+=10;
            c.s[i+1]--;
        }
    }
    while(len>1&&c.s[len]==0) len--;
    c.len=len;
}
//高精度乘10
void multiply10(hp &a)
{
```

```
    int i;
    for(i=a.len;i>=1;i--)
        a.s[i+1]=a.s[i];
    a.s[1]=0;
    a.len++;
    while(a.len>1&&a.s[a.len]==0)
a.len--;
}
//高精度乘单精度，a是高精度，b是单精度
void multiply(const hp &a,int b,hp &c){
    int i,len;
    for(i=1;i<=maxsize;i++) c.s[i]=0;
    len=a.len;
    for(i=1;i<=len;i++){
        c.s[i]+=a.s[i]*b;
        c.s[i+1]+=c.s[i]/10;
        c.s[i]%=10;
    }
    len++;
    while(c.s[len]>=10){
        c.s[len+1]+=c.s[len]/10;
        c.s[len]%=10;
        len++;
    }
    while(len>1&&c.s[len]==0) len--;
    c.len=len;
}
//高精度乘高精度
void multiplyh(const hp &a,const hp &b,hp
&c)
{
    int i,j,len;
    for(i=1;i<=maxsize;i++) c.s[i]=0;
    for(i=1;i<=a.len;i++)
        for(j=1;j<=b.len;j++)
        {
            c.s[i+j-1]+=a.s[i]*b.s[j];
            c.s[i+j]+=c.s[i+j-1]/10;
            c.s[i+j-1]%=10;
  d.s[1]=a.s[i];
  while(compare(d,b)>=0)
  {
      subtract(d,b,e);
      d=e;
        }
        len=a.len+b.len+1;
        while(len>1&&c.s[len]==0) len--;
        c.len=len;
}
//高精度除单精度，a是高精度，c为商，d为余
数
void divide(const hp &a,int b,hp &c,int &d)
{
    int i,len;
    for(i=1;i<=maxsize;i++) c.s[i]=0;
    len=a.len;
    d=0;
    for(i=len;i>=1;i--)
    {
        d=d*10+a.s[i];
        c.s[i]=d/b;
        d%=b;
    }
    while(len>1&&c.s[len]==0) len--;
    c.len=len;
}
//高精度除高精度
void divideh(const hp &a,const hp &b,hp
&c,hp &d)
{
    hp e;
    int i,len;
    for(i=1;i<=maxsize;i++)
    {
        c.s[i]=0;
        d.s[i]=0;
    }
    len=a.len;
    d.len=1;
    for(i=len;i>=1;i--)
    {
        multiply10(d);

            c.s[i]++;
        }
    }
    while(len>1&&c.s[len]==0) len--;
    c.len=len;
```

```
}
```

JAVA 实现

```
package Animal;

import java.util.*;
import java.math.*;

class order{
    int num ,id;
    static class orderComparator implements
Comparator{
        public int compare(Object o1,Object
o2){
                order s1 = (order)o1;
                order s2 = (order)o2;
                if(s1.num==s2.num)
                {
                    return s1.id > s2.id ? 1 :
-1;
                }
                return s1.num > s2.num ? 1 :
-1;
        }
    }
    order(int num ,int id){
        this.num = num;
        this.id = id;
    }
}

class orderComparator implements Comparator
{//从小到大排序
    public final int compare(Object pFirst,
Object pSecond) {
        order a = (order) pFirst;
        order b = (order) pSecond;
        if(a.num==b.num){
            return a.id > b.id ? 1 : -1;
        }
        return a.num > b.num ? 1 : -1;
    }
}
```

```
public class Cat {
    static int n;
    public static void main(String[] args){
        BigInteger[] ans = new
BigInteger[1010];
        order[] no = new order[1010];
        BigInteger a ,b ,c;
        Scanner cin = new Scanner(System.in);
        while(cin.hasNext()){
            n = cin.nextInt();
            for(int i = 0;i < n;i++){
                no[i] = new
order(cin.nextInt(),i);
            }
            Arrays.sort(no,0,n,new
orderComparator());
            a = BigInteger.ONE;
            b = BigInteger.ONE;
            int s = 2 ,cnt = 0;
            while(cnt < n){

                while(s < no[cnt].num){
                    c = a;
                    a = a.add(b);
                    b = c;
                    s++;
                }
                ans[no[cnt].id] = a;
                cnt++;
            }
            for(int i = 0;i < n;i++){
                System.out.println(ans[i]);
            }
        }
    }
}
```

BigInteger a ,b,c;
BigInteger[] d = new BigInteger[40];
Int x;
a = BigInteger.valueof(3232);将一个 int 型赋值给一个大数。

d = a. divideAndRemainder(b);除法并取余，返回一个数组，第一个为商，第二个为余数
c = a.pow(x);a 的 x 次方，x 为 int 型
//10 进制->2 进制
String a = "121";//输入数值
BigInteger src = new BigInteger(a);//转换为 BigInteger 类型
System.out.println(src.toString(2));//转换为 2 进制并输出
//2 进制->10 进制
String b = "1111001";//输入数值
BigInteger src1= new BigInteger(b,2);// 转换为 BigInteger 类型
System.out.println(src1.toString());//转换为 10 进制并输出结果

杂

去重

```
v.erase(unique(v.begin(),v.end()),v.end());
struct NODE{
    int x ,y;
    bool operator < (const struct NODE &a) const {
        if(x==a.x)
            return y > a.y;
        return x > a.x;
    }
};//按 x 从大到小排序，优先队列时按 x 从小到大
```

手动扩栈

```
#pragma comment(linker, "/STACK:102400000,102400000")
```

输入加速

```
void read(int &x) {
    char c;
    while((c=getchar())<'0' || c>'9');
    x=c-'0';
    while((c=getchar())>='0' && c<='9')
x=(x<<3)+(x<<1)+c-'0';
}
```