计算几何

常用函数

```
atan(double x);          //给一个值y/x，返回对应的角度，值域(-π/2,π/2)
atan2(double y,double x);       //给一个坐标 (x,y) ,返回和x轴的夹角，值域 (-π,π)
```

二维基本运算

```
#define int long long
#define double long double
const double eps=1e-8;
const double pi=acos(-1);
int sgn(double x){
    if(abs(x)<=eps)return 0;
    return x<0?-1:1;
}
struct Point{
    double x,y;
    Point(){}
    Point(double x,double y):x(x),y(y){}

    bool operator==(const Point&a)const{
        return (sgn(x-a.x)==0&&sgn(y-a.y)==0);
    }
    bool operator<(const Point&a)const{
        if(sgn(x-a.x)==0)return sgn(y-a.y)==-1;
        return sgn(x-a.x)==-1;
    }
    bool operator>(const Point&a)const{return !((*this<a)||(*this==a));}

    Point operator+(const Point&a)const{return {x+a.x,y+a.y};}
    Point operator-(const Point&a)const{return {x-a.x,y-a.y};}
    Point operator-()const{return {-x,-y};}
    Point operator*(double t)const{return {x*t,y*t};}
    Point operator/(double a)const{return {x/a,y/a};}

    double dot(const Point&a)const{return x*a.x+y*a.y;}
    double cross(const Point&a)const{return x*a.y-y*a.x;}
    double len2()const{return dot(*this);}
    double len()const{return sqrt(len2());}
    int toleft(const Point&a)const{return sgn(cross(a));}    //当前向量到 a 是逆时针旋转

    double dis2(const Point&a)const{return ((*this)-a).len2();}
    double dis(const Point&a)const{return sqrt(dis2(a));}

    double ang(const Point&a)const{          //两向量夹角
        return acos(max(-1.0l,min((dot(a)/(len()*a.len())),1.0l)));
    }
    double ang()const{return atan2(y,x);}        // 极角
    Point rotate()const{return {-y,x};}      // 逆时针旋转 90°
```

```
    Point rot(const double&rad)const{       // 逆时针旋转给定角度
        return{x*cos(rad)-y*sin(rad),x*sin(rad)+y*cos(rad)};
    }
};
typedef Point Vector;
struct argcmp{      // 极角排序
    bool operator()(const Point&a,const Point&b) const{
        const auto quad=[](const Point&a){
            if(a.y<-eps)return 1;
            if(a.y>eps)return 4;
            if(a.x<-eps)return 5;
            if(a.x>eps)return 3;
            return 2;
        };
        const int qa=quad(a),qb=quad(b);
        if(qa!=qb)return qa<qb;
        const double t=a.cross(b);
        // if(abs(t)<=eps) return a*a<b*b-eps;  // 不同长度的向量需要分开
        return t>eps;
    }
};
bool cmp(Point x,Point y){
    return argcmp()(x,y);
}
struct Line{        //直线
    Point p;
    Vector v;
    Line(){}
    Line(Point p,Vector v):p(p),v(v){}
    double dis(Point a)const{return abs(v.cross(a-p)/v.len());}      //点到直线距
离
    Point inter(const Line&a)const{     //两直线交点
        return p+v*((a.v.cross(p-a.p))/(v.cross(a.v)));
    }
    bool operator<(const Line&a)const{      // 直线极角排序，极角相同时靠左的在前
        if(sgn(atan2(v.y,v.x)-atan2(a.v.y,a.v.x))==0)return (a.p-
p).cross(a.p+a.v-p)>0;
        else return atan2(v.y,v.x)<atan2(a.v.y,a.v.x);
    }
};
struct Segment{         //线段
    Point a,b;
    Segment(){}
    Segment(Point a,Point b):a(a),b(b){}
    bool operator<(const Segment &s)const{      //按左端点横坐标排序
        return make_pair(a,b)<make_pair(s.a,s.b);
    }
    double dis(const Point&p)const{     //点到线段距离
        if(sgn((p-a).dot(b-a))==-1||sgn((p-b).dot(a-b))==-1)return
min(p.dis(a),p.dis(b));
        Point p0=a,v=b-a;
        return abs(v.cross(p-p0)/v.len());
    }
    // 判断点是否在线段上
    // -1 点在线段端点 | 0 点不在线段上 | 1 点严格在线段上
```

```
    int is_on(const Point&p)const{
        if(p==a || p==b)return -1;
        return (p-a).toleft(p-b)==0 && (p-a).dot(p-b)<-eps;
    }
};
struct Circle{          //圆
    Point p;
    double r;
    Circle(){}
    Circle(Point p,double r):p(p),r(r){}
    Point getPoint(double alpha){
        return p+Point(r*cos(alpha),r*sin(alpha));
    }
    int in(Point const&a){return sgn(r-p.dis(a));}      //点在圆内
    pair<Point,Point>inter(const Line&l)const{      // 点和直线交点
        // 保证有解
        Point o=Line(p,l.v.rotate()).inter(l);
        double dis2=p.dis2(o);
        double d=sqrt(r*r-dis2),len=l.v.len();
        return {o+l.v*d/len,o-l.v*d/len};
    }
};
struct Polygon{             //多边形
    vector<Point> p; //逆时针顺序存储
    int nxt(const int x)const{return x==(p.size()-1)?0:(x+1);}      //下个点
    int pre(const int x)const{return x==0?(p.size()-1):(x-1);}      //上个点
    double area()const{     //多边形面积，顺时针存点为负，逆时针存点为正
        double sum=0;
        for(int i=0;i<p.size();i++)
            sum+=p[i].cross(p[nxt(i)]);
        return sum/2;
    }
    double circ()const{     //多边形周长
        double sum=0;
        for(int i=0;i<p.size();i++)
            sum+=p[i].dis(p[nxt(i)]);
        return sum;
    }
};
struct Convex:Polygon{      // 凸多边形
    double Calipers()const{         // 凸包直径 (旋转卡壳)
        double ans=0;
        if(p.size()==2)return p[0].dis(p[1]);
        int now=0;
        for(int i=0;i<p.size();i++){
            Line l(p[i],p[i]-p[nxt(i)]);
            while(l.dis(p[now])<=l.dis(p[nxt(now)]))now=nxt(now);
            ans=max(ans,max(p[i].dis(p[now]),p[nxt(i)].dis(p[now])));
        }
        return ans;
    }
    // O(logn) 判断点是否在凸包内
    // -1 点在凸包边上 | 0 点在凸包外 | 1 点在凸包内
    int is_in(const Point a)const{
        if(p.size()==1)return a==p[0]?-1:0;
```

```
        if(p.size()==2)return Segment(p[0],p[1]).is_on(a)?-1:0;
        if(a==p[0])return -1;
        if((p[1]-p[0]).toleft(a-p[0])==-1 || (p.back()-p[0]).toleft(a-
p[0])==1)return 0;
        auto cmp=[&](const Point&u,const Point&v){
            return (u-p[0]).toleft(v-p[0])==1;
        };
        int pos=lower_bound(p.begin()+1,p.end(),a,cmp)-p.begin();
        if(pos==1)return Segment(p[0],p[1]).is_on(a)?-1:0;
        if(pos==p.size()-1 && Segment(p[0],p[pos]).is_on(a))return -1;
        if(Segment(p[pos-1],p[pos]).is_on(a))return -1;
        return (p[pos]-p[pos-1]).toleft(a-p[pos-1])>0;
    }
    int get(const Point a,int op)const{     // op=1  极角序最大的切点    |   op=-1
极角序最小的切点
        int l=0,r=p.size()-1;
        auto cmp=[&](const Point&x,const Point&y){
            return (x-a).toleft(y-a)==op;
        };
        if(cmp(p[0],p.back())){     // 极值在 p[back]
            while(l<r){
                int mid=(l+r)>>1;
                if(cmp(p[r],p[mid]) && cmp(p[mid+1],p[mid]))r=mid;
                else l=mid+1;
            }
        }
        else{       // 极值在 p[0]
            while(l<r){
                int mid=(l+r+1)>>1;
                if(cmp(p[l],p[mid]) && cmp(p[mid-1],p[mid]))l=mid;
                else r=mid-1;
            }
        }
        return l;
    }
    // O(logn) 点到凸包切线，返回两个切点在凸包中的下标
    pair<int,int>get_tan(const Point&a)const{
        return {get(a,1),get(a,-1)};
    }
    // O(logn) 点到凸包距离
    double dis(const Point&a)const{
        if(is_in(a)==-1 || is_in(a)==1)return 0;
        if(p.size()==1)return p[0].dis(a);
        if(p.size()==2)return Segment(p[0],p[1]).dis(a);
        auto[mx,mn]=get_tan(a);
        double
dis=min(Segment(p[mx],p[nxt(mx)]).dis(a),Segment(p[pre(mn)],p[mn]).dis(a));
        int op1=sgn((p[nxt(mx)]-p[mx]).dot(a-p[mx]));
        int op2=sgn((p[mn]-p[pre(mn)]).dot(a-p[pre(mn)]));
        if(mx!=mn && op1!=op2){
            int l=mx,r=mn-1;
            if(r<l)r+=p.size();
            while(l<r){
                int mid=(l+r+1)>>1,t=mid%p.size();
                int p1=sgn((p[nxt(t)]-p[t]).dot(a-p[t]));
```

```
                if(p1==op1)l=mid;
                else r=mid-1;
            }
            l%=p.size();
            dis=min(dis,Segment(p[l],p[nxt(l)]).dis(a));
        }
        return dis;
    }
};
```

反演变换

```
Point O;    // 反演圆心
double R;   // 反演半径
Circle rev(Circle a){       //反演圆-圆
    Circle ans;
    double dis1=a.p.dis(O);
    double x=1.0/(dis1-a.r),y=1.0/(dis1+a.r);
    ans.r=(x-y)/2.0;
    double dis2=(x+y)/2.0;
    ans.p=O+(a.p-O)*(dis2/dis1);
    return ans;
}
```

判断圆的交集是否非空

```
bool Cross(Circle c1,Circle c2,double&ans){
    double r1=c1.r,r2=c2.r;
    double dis=c1.p.dis2(c2.p);
    if(sgn(dis-(r1+r2)*(r1+r2))>=0)return false;       // 面积>0  >=    |  面积>=0
>
    double l=sqrt(dis);
    double cs0=(c2.p.x-c1.p.x)/l,sn0=(c2.p.y-c1.p.y)/l;
    double cs1=(r1*r1+dis-r2*r2)*0.5l/r1/l,sn1=sqrt(1.0l-cs1*cs1);
    ans=c1.p.x+r1*(cs0*cs1-sn0*sn1);
    return true;
}
bool check(vector<Circle>c){
    int n=c.size();
    double l=-1e10,r=1e10;
    for(int _=1;_<=200;_++){       // 二分出一条穿过交集的直线
        double mid=(l+r)/2;
        int flag=0;
        int a,b;
        double high=1e10,low=-1e10;
        for(int i=0;i<n;i++){
            if(sgn(c[i].p.x+c[i].r-mid)<=0)flag=1;
            if(sgn(c[i].p.x-c[i].r-mid)>=0)flag=2;
            if(flag)break;
            double del=sqrt(c[i].r*c[i].r-abs(mid-c[i].p.x)*abs(mid-c[i].p.x));
            if(c[i].p.y+del<high)high=c[i].p.y+del,a=i;     // 计算 min 上交点
            if(c[i].p.y-del>low)low=c[i].p.y-del,b=i;       // 计算 max 下交点
        }
        if(flag==1)r=mid;
```

```
        else if(flag==2)l=mid;
        else if(sgn(high-low)>0)return true;       // 面积>0  >   |  面积>=0  >=
        else{
            double ans;
            if(Cross(c[a],c[b],ans)==0)return false;
            if(ans<mid)r=mid;
            else l=mid;
        }
    }
    double high=1e10,low=-1e10;
    for(int i=0;i<n;i++){
        if(sgn(c[i].p.x+c[i].r-l)<=0)return false;       // 面积>0  <=   |  面积
>=0  <
        if(sgn(c[i].p.x-c[i].r-l)>=0)return false;       // 面积>0  >=   |  面积
>=0  >
        double del=sqrt(c[i].r*c[i].r-abs(l-c[i].p.x)*abs(l-c[i].p.x));
        high=min(high,c[i].p.y+del);
        low=max(low,c[i].p.y-del);
    }
    if(sgn(high-low)>0)return true;        // 面积>0  >   |  面积>=0  >=
    return false;
}
```

最小圆覆盖

```
// 随机增量法 O(n) 求最小圆
Circle MinCircle(vector<Point> p){
    mt19937 Rnd(random_device{}());
    shuffle(p.begin(), p.end(),Rnd);
    Circle ans(p[0],0);
    auto get=[](Point a,Point b,Point c){
        Circle ans;
        double a1=b.x-a.x,a2=c.x-a.x,b1=b.y-a.y,b2=c.y-a.y;
        double c1=b.x*b.x-a.x*a.x+b.y*b.y-a.y*a.y;
        double c2=c.x*c.x-a.x*a.x+c.y*c.y-a.y*a.y;
        ans.p=Point((b2*c1-b1*c2)/(b2*a1*2-b1*a2*2),(a2*c1-a1*c2)/(a2*b1*2-
a1*b2*2));
        ans.r=ans.p.dis(a);
        return ans;
    };
    for(int i=1;i<p.size();i++){
        if(ans.in(p[i])==-1){
            ans.p=(p[i]+p[1])*0.5;
            ans.r=p[i].dis(p[1])*0.5;
            for(int j=0;j<i;j++){
                if(ans.in(p[j])==-1){
                    ans.p=(p[i]+p[j])*0.5;
                    ans.r=p[i].dis(p[j])*0.5;
                    for(int k=0;k<j;k++){
                        if(ans.in(p[k])==-1){
                            ans=get(p[i],p[j],p[k]);
                        }
                    }
                }
            }
```

```
        }
    }
    return ans;
}
```

最小正方形覆盖

```
// 旋转卡壳实现 O(n)
// 返回最小正方形覆盖的边长
double MinSquare(vector<Point>&p){
    p=Andrew(p).p;          // 先求凸包
    if(p.size()<=1)return 0;
    if(p.size()==2)return p[0].dis(p[1])/sqrt(2);
    int n=p.size();
    auto nxt=[&](int x){return x==n-1?0:x+1;};
    vector<Point>vec;
    for(int i=0;i<n;i++){
        int j=nxt(i);
        vec.push_back(p[j]-p[i]);
        vec.push_back((p[j]-p[i]).rotate());
        vec.push_back(p[i]-p[j]);
        vec.push_back((p[i]-p[j]).rotate());
    }
    auto cal=[&](Point a){
        return a.y>0 || (sgn(a.y)==0 && a.x>0)?1:-1;
    };
    sort(vec.begin(), vec.end(),[&](const Point&a,const Point&b){
        if(cal(a)!=cal(b))return cal(a)==1;
        return a.cross(b)>0;
    });
    int i=n-1,j=n-1,k=n-1,l=n-1;
    Point v=vec[0];
    while(i>0 && sgn(v.cross(p[i-1]-p[i]))<=0)i--;
    while(j>0 && sgn(v.dot(p[j-1]-p[j]))>=0)j--;
    while(k>0 && sgn(v.cross(p[k-1]-p[k]))>=0)k--;
    while(l>0 && sgn(v.dot(p[l-1]-p[l]))<=0)l--;
    double ans=1e18;
    for(int u=0;u<vec.size();u++){
        Point t=vec[u];
        while(sgn(t.cross(p[i+1]-p[i]) )<=0)i=nxt(i);
        while(sgn(t.dot(p[j+1]-p[j]) )>=0)j=nxt(j);
        while(sgn(t.cross(p[k+1]-p[k]) )>=0)k=nxt(k);
        while(sgn(t.dot(p[l+1]-p[l]) )<=0)l=nxt(l);
        ans=min(ans,max(t.cross(p[k]-p[i]) , t.dot(p[j]-p[l])) / t.len());
        Point a=(p[i]-p[k]).rotate(),b=p[j]-p[l];
        Point c=(b-a).rotate();
        if(sgn(a.dot(c))<0)c=-c;
        Point v1=(u==vec.size()-1)?vec[0]:vec[u+1];
        if(sgn(c.dot(c))>0 && sgn(t.cross(c))>=0 && sgn(c.cross(v1))>=0){
            ans=min(ans,a.dot(c)/c.len());
        }
    }
    return ans;
}
```

```
// 三分实现 O(100*n)
double MinSquare(const vector<Point>&p){
    auto cal=[&](double rad){
        double mn1=1e10,mn2=1e10,mx1=-1e10,mx2=-1e10;
        for(auto&i:p){
            Point t=i.rot(rad);
            mn1=min(mn1,t.x);
            mx1=max(mx1,t.x);
            mn2=min(mn2,t.y);
            mx2=max(mx2,t.y);
        }
        return max(mx1-mn1,mx2-mn2);
    };
    // 在 (0,pi/2) 三分未必正确，可以尝试改变 r 为 pi, 3pi/4
    // 也可在 (0,pi/4) , (pi/4,pi/2) 分别三分
    double l=0,r=pi/2,ans=1e18;
    for(int i=1;i<=50;i++){
        double m1=(2*l+r)/3,m2=(l+2*r)/3;
        double cl=cal(m1),cr=cal(m2);
        ans=min({ans,cl,cr});
        if(cl<cr)r=m2;
        else l=m1;
    }
    return ans;
}
```

二维凸包

```
Convex Andrew(vector<Point> p) {    //nlogn求凸包
    vector<Point> st;
    Convex ans;
    if(p.empty()){
        ans.p=st;
        return ans;
    }
    sort(p.begin(),p.end());
    auto check=[](const vector<Point>&st,Point u){
        Point tmp1=st.back(),tmp2=*prev(st.end(),2);
        return (tmp1-tmp2).toleft(u-tmp1)<=0;      // < 边上有点 , <= 边上没点
    };
    for(auto u:p){
        while(st.size()>1 && check(st,u))
            st.pop_back();
        st.push_back(u);
    }
    int k=st.size();
    p.pop_back();       // 下凸壳最后一点和上凸壳起始点重复
    reverse(p.begin(), p.end());
    for(auto u:p){
        while(st.size()>k && check(st,u))
            st.pop_back();
        st.push_back(u);
    }
    st.pop_back();      // 上凸壳最后一点和下凸壳起始点重复
```

```
    ans.p=st;
    return ans;
}
// 凸包判定: 先转换为逆时针存点, 判断所有内角为锐角, 并且内角和为 (n-2)*pi
bool is_Convex(vector<Point>p){
    int n=p.size();
    auto pre=[&](int x){return x==0?(n-1):(x-1);};
    auto nxt=[&](int x){return x==(n-1)?0:(x+1);};
    double area=0;
    for(int i=0;i<n;i++){
        if(p[i]==p[pre(i)])return false;
        area+=p[i].cross(p[nxt(i)]);
    }
    if(area<0)      // 顺时针转逆时针
        for(int i=0;i<n/2;i++)swap(p[i],p[n-i]);
    double sum=0;
    for(int i=0;i<n;i++){
        if((p[nxt(i)]-p[i]).cross(p[pre(i)]-p[i])<=0)return false;
        sum+=(p[nxt(i)]-p[i]).ang(p[pre(i)]-p[i]);
    }
    return fabs(sum-(n-2)*pi)<=1.0;
}
```

动态凸包

坐标排序维护凸壳

```
struct Point{
    int x,y;
    Point(){}
    Point(int x,int y):x(x),y(y){}
    Point operator-(const Point&a)const{return {x-a.x,y-a.y};}
    int cross(const Point&a)const{return x*a.y-y*a.x;}
    // 按坐标排序
    bool operator<(const Point&a)const{return x==a.x?y<a.y:x<a.x;}
    double dis(const Point&a)const{return sqrt(1.0l*(x-a.x)*(x-a.x)+(y-a.y)*(y-
a.y));}
};
double circ=0;  // 凸壳周长
set<Point>st;
void del(Point p){
    st.erase(p);
    auto it=st.lower_bound(p);
    circ-=p.dis(*it);
    circ-=p.dis(*prev(it));
    circ+=(*it).dis(*prev(it));
}
void ins(Point p){
    auto nxt=st.lower_bound(p);
    // if(nxt==st.end()){特判当前点为右端点}
    if(nxt->x==p.x)return;      // 上凸壳
    // if(nxt==st.begin()){特判当前点为左端点}
    auto pre=prev(nxt);
    // if(pre->x==p.x)return;   // 下凸壳
    while(pre!=st.begin()){
```

```
        auto it=prev(pre);
        if((*pre-*it).cross(p-*pre)>=0){    // 上凸壳 >=   |   下凸壳 <=
            del(*pre);
            pre=it;
        }
        else break;
    }
    while(next(nxt)!=st.end()){
        auto it=next(nxt);
        if((*nxt-p).cross(*it-*nxt)>=0){    // 上凸壳 >=   |   下凸壳 <=
            del(*nxt);
            nxt=it;
        }
        else break;
    }
    if((p-*pre).cross(*nxt-p)<0){    // 上凸壳 <   |   下凸壳 >
        circ+=p.dis(*pre);
        circ+=p.dis(*nxt);
        circ-=(*pre).dis(*nxt);
        st.insert(p);
    }
}
```

极角排序维护凸包

```
const double eps=1e-8;
int sgn(double x){
    if(abs(x)<=eps)return 0;
    return x<0?-1:1;
}
struct Point{
    double x,y,angle;
    Point(){}
    Point(double x,double y):x(x),y(y){}
    Point operator+(const Point&a)const{return {x+a.x,y+a.y};}
    Point operator-(const Point&a)const{return {x-a.x,y-a.y};}
    Point operator/(double a)const{return {x/a,y/a};}
    double cross(const Point&a)const{return x*a.y-y*a.x;}
    double ang()const{return atan2(y,x);}
    // 按和凸包内一点连线的极角排序，取初始三点的重心
    bool operator<(const Point&a)const{return angle<a.angle;}
}P[3];
set<Point>st;
set<Point>::iterator nxt,pre,nnxt,ppre;
void ins(Point p){
    nxt=st.lower_bound(p);
    if(nxt==st.begin())pre=prev(st.end());
    else pre=prev(nxt);
    if(nxt==st.end())nxt=st.begin();
    while(true){
        nnxt=next(nxt);
        if(nnxt==st.end())nnxt=st.begin();
        if(sgn((*nxt-p).cross(*nnxt-*nxt))<=0){
            st.erase(nxt);
            nxt=nnxt;
```

```
        }
        else break;
    }
    while(true){
        if(pre==st.begin())ppre=prev(st.end());
        else ppre=prev(pre);
        if(sgn((*pre-*ppre).cross(p-*pre))<=0){
            st.erase(pre);
            pre=ppre;
        }
        else break;
    }
    if(sgn((p-*pre).cross(*nxt-p))>0)st.insert(p);
}
// 查询点 p 是否在凸包内
bool query(Point p){
    nxt=st.lower_bound(p);
    if(nxt==st.begin())pre=prev(st.end());
    else pre=prev(nxt);
    if(nxt==st.end())nxt=st.begin();
    if(sgn((p-*pre).cross(*nxt-p))>0)return false;
    return true;
}
void solve(){
    Point o=(P[0]+P[1]+P[2])/3.0;
    for(int i=0;i<3;i++){
        P[i].angle=(P[i]-o).ang();
        st.insert(P[i]);
    }
}
```

半平面交

```
Polygon Cross(vector<Line>vec){     // nlogn 返回 n 个直线的半平面交形成的凸包
    sort(vec.begin(),vec.end());
    deque<Line>dq;
    vector<Line>v;
    for(auto line:vec){     // 按极角去重
        if(v.empty() || sgn(atan2(v.back().v.y,v.back().v.x)-
atan2(line.v.y,line.v.x))!=0)v.push_back(line);
    }
    // 判断是否点是否在直线右侧，是则非法
    auto illegal=[&](Point p,Line l){return (l.p-p).cross(l.p+l.v-p)<0;};
    for(auto line:v){
        while(dq.size()>=2 &&
illegal(dq.back().inter(dq.at(dq.size()-2)),line))dq.pop_back();
        while(dq.size()>=2 &&
illegal(dq.front().inter(dq.at(1)),line))dq.pop_front();
        dq.push_back(line);
    }
    // 去除多余的点
    while(dq.size()>=2 &&
illegal(dq.back().inter(dq.at(dq.size()-2)),dq.front()))dq.pop_back();
    while(dq.size()>=2 &&
illegal(dq.front().inter(dq.at(1)),dq.back()))dq.pop_front();
```

```
    vector<Point>ans;
    for(int i=0;i<dq.size();i++){
        ans.push_back(dq.at(i).inter(dq.at((i+1)%dq.size())));
    }
    // dq.size()>=3 表示交集非空
    // 注意特判：交为空，交为无穷
    // 根据加入边界的框，判断和框有无交点，可以判断无穷
    Polygon res;
    res.p=ans;
    return res;
}

Polygon Cross(const vector<Line> vec,Point L,Point R){     // n^2 半平面交
    vector<Point>p;
    // 加边框
    p.emplace_back(L.x,L.y);
    p.emplace_back(R.x,L.y);
    p.emplace_back(R.x,R.y);
    p.emplace_back(L.x,R.y);
    for(auto&l:vec){
        vector<Point>t;
        int n=p.size();
        for(int i=0;i<n;i++){
            if(l.v.toleft(p[i]-l.p)>=0)t.push_back(p[i]);
            if(sgn(l.v.cross(p[i]-l.p)) * sgn(l.v.cross(p[(i+1)%n]-l.p)) < 0)
                t.push_back(l.inter(Line(p[i],p[(i+1)%n]-p[i])));
        }
        p=t;
    }
    Polygon poly;
    poly.p=p;
    return poly;
}
```

Voronoi 图

```
// O(n^2logn),;
// 随机打乱半平面后，再使用 O(n^2) 半平面交的期望复杂度为 O(n^2)
// 注：不能有重点，跑之前要先去重
struct Line{        //直线
    Point p;
    Vector v;
    int id;
    Line(){}
    Line(Point p,Vector v):p(p),v(v){}
    Line(Point p,Vector v,int id):p(p),v(v),id(id){}
};
deque<Line> Cross(vector<Line>vec){}    // 半平面交
const int N=1e3+5;
Point p[N];
deque<Line> vor[N];    // voronoi 图
int vis[N];     // 标记是否和边框相邻
vector<int>G[N];    // 相邻的点集
Line bisector(Point a,Point b){     // 两点中垂线
    Vector v=b-a;
```

```
    return {(a+b)*0.5,Vector(-v.y,v.x)};
}
void Voronoi(int n,Point t){    // n 为点个数，(0,0),t 是左下/右上角
    for(int i=1;i<=n;i++){
        vector<Line>vec;
        // 加边框
        vec.emplace_back(Point(0,0),Vector(t.x,0),0);
        vec.emplace_back(Point(t.x,0),Vector(0,t.y),0);
        vec.emplace_back(Point(t.x,t.y),Vector(-t.x,0),0);
        vec.emplace_back(Point(0,t.y),Vector(0,-t.y),0);
        for(int j=1;j<=n;j++){
            if(i==j)continue;
            Line l=bisector(p[i],p[j]);
            vec.emplace_back(l.p,l.v,j);
        }
        // shuffle(vec.begin(), vec.end(), mt19937(random_device{}()));
        vor[i]=Cross(vec);
        int m=vor[i].size();
        for(int j=0;j<m;j++){
            Line l1,l2,l=vor[i].at(j);
            if(j==0)l1=vor[i].back();
            else l1=vor[i].at(j-1);
            if(j==m-1)l2=vor[i].at(0);
            else l2=vor[i].at(j+1);
            if(l1.inter(l)==l2.inter(l))continue;    // 交集为一个点，根据题意判断是否
保留
            if(l.id==0)vis[i]=1;
            else G[i].push_back(l.id);
        }
    }
}
```

平面最近点对

```
// O(nlogn) 平面分治
bool cmp_y(Point a,Point b){return a.y<b.y;}
double cal(int l,int r,vector<Point>&v){
    if(r-l<=2){     // 最多三个点
        double ans=1e18;
        for(int i=l;i<=r;i++)
            for(int j=i+1;j<=r;j++)
                ans=min(ans,v[i].dis(v[j]));
        sort(v.begin()+l,v.begin()+r+1,cmp_y);
        return ans;
    }
    int mid=(l+r)>>1;
    double mid_x=v[mid].x;
    double dis=min(cal(l,mid,v),cal(mid+1,r,v));
    vector<Point>vec;       // 按 y 进行归并排序
    int pos1=l,pos2=mid+1;
    while(pos1<=mid && pos2<=r){
        if(v[pos1].y<v[pos2].y)vec.push_back(v[pos1++]);
        else vec.push_back(v[pos2++]);
    }
    while(pos1<=mid)vec.push_back(v[pos1++]);
```

```
    while(pos2<=r)vec.push_back(v[pos2++]);
    int pos=l;
    for(auto i:vec)v[pos++]=i;
    for(int i=l;i<=r;i++){
        if(fabs(v[i].x-mid_x)<dis){          // 按 x 划分区域
            for(int j=i-1;j>=l;j--){
                if(v[i].y-v[j].y>dis)break;      // 按 y 划分区域
                dis=min(dis,v[i].dis(v[j]));
            }
        }
    }
    return dis;
}
double min_dis(vector<Point>&v){        // 平面最近点对
    sort(v.begin(),v.end());
    return cal(0,v.size()-1,v);
}
```

闵可夫斯基和

```
// 点集 A 取两个点的距离，可以转化为 Minkowski(A,-A) 上 (0,0) 到 (x,y) 的距离
// 推一下式子，转化为凸包的和/差，然后利用闵可夫斯基和优化
Convex Minkowski(Convex a,Convex b){
    vector<Vector>v1,v2;
    for(int i=0;i<a.p.size()-1;i++)v1.push_back(a.p[i+1]-a.p[i]);
    v1.push_back(a.p[0]-a.p.back());
    for(int i=0;i<b.p.size()-1;i++)v2.push_back(b.p[i+1]-b.p[i]);
    v2.push_back(b.p[0]-b.p.back());
    vector<Point>ans;
    ans.push_back(a.p[0]+b.p[0]);
    int p1=0,p2=0;
    while(p1<v1.size() && p2<v2.size()){
        ans.push_back(ans.back() + (v1[p1].cross(v2[p2])>=0?v1[p1++]:v2[p2++])
);
    }
    while(p1<v1.size())ans.push_back(ans.back()+v1[p1++]);
    while(p2<v2.size())ans.push_back(ans.back()+v2[p2++]);
    return Andrew(ans);
}
```

三维计算几何

```
#define int long long
#define double long double
const double eps=1e-8;
const double pi=acos(-1);
int sgn(double x){      //符号
    if(abs(x)<=eps)return 0;
    if(x>0)return 1;
    return -1;
}
mt19937_64 Rnd(random_device{}());
uniform_real_distribution<double>dist(-1e-2,1e-2);
double reps(){return dist(Rnd)*eps;}    // 随机扰动
```

```cpp
struct Point{       // 点
    double x,y,z;
    Point(){}
    Point(double x,double y,double z):x(x),y(y),z(z){}
    Point operator+(const Point&a)const{return {x+a.x,y+a.y,z+a.z};}
    Point operator-(const Point&a)const{return {x-a.x,y-a.y,z-a.z};}
    Point operator*(const double&a)const{return {x*a,y*a,z/a};}
    bool operator==(const Point&a)const{return sgn(x-a.x)==0 && sgn(y-a.y)==0 &&
sgn(z-a.z)==0;}

    double dot(const Point&a)const{return x*a.x+y*a.y+z*a.z;}       // 点积
    Point cross(const Point&a)const{return {y*a.z-z*a.y, z*a.x-x*a.z, x*a.y-
y*a.x};}      // 叉积
    double len()const{return sqrt(dot(*this));}     // 向量模长

    void shake(){x+=reps(),y+=reps(),z+=reps();}    // 随机扰动，避免三点共线/四点共面
};
typedef Point Vector;
bool coinline(const Point&a,const Point&b,const Point&c){
    Point t=(b-a).cross(c-a);
    return !sgn(t.x) && !sgn(t.y) && !sgn(t.z);
}

struct Plane{       // 平面
    Point p[3];     // 三点确定平面
    int id[3];      // 求凸包，记录平面上点的标号
    Vector normal()const{return (p[1]-p[0]).cross(p[2]-p[0]);}      // 法向量
    double area()const{return normal().len()/2.0;}      // 三角形面积
};

struct Convex{      // 三维凸包
    vector<Plane>p;
    double area(){      // 凸包表面积
        double ans=0;
        for(auto i:p)
            ans+=i.area();
        return ans;
    }
};
Convex Convex_3d(vector<Point>v){       // 增量法，O(n^2)
    int n=v.size();
    vector<vector<int>>vis(n,vector<int>(n,0));      // 记录每条棱是否可见
    for(auto&i:v)i.shake();     // 求精确解时不可以随机扰动
    vector<Plane>ans,tmp;
    ans.push_back({{v[0],v[1],v[2]},{0,1,2}});
    ans.push_back({{v[2],v[1],v[0]},{2,1,0}});
    if(n==3)return {ans};
    auto see=[&](Point p,Plane plane){     // 从一个点能否看到一个面
        return (p-plane.p[0]).dot(plane.normal())>0;
    };
    for(int i=3;i<n;i++){
        for(auto plane:ans){
            bool ok=see(v[i],plane);
            if(!ok)tmp.push_back(plane);
            for(int j=0;j<3;j++){
```

```
                vis[plane.id[j]][plane.id[(j+1)%3]]=ok;
            }
        }
        for(auto plane:ans){
            for(int j=0;j<3;j++){
                int x=plane.id[j],y=plane.id[(j+1)%3];
                if(vis[x][y] && !vis[y][x]){    // 关键棱，构成新面
                    tmp.push_back({{plane.p[j],plane.p[(j+1)%3],v[i]},{x,y,i}});
                }
            }
        }
        ans=tmp;
        tmp.clear();
    }
    return {ans};
};
```