凸包与半平面交

凸包

Graham算法

Andrew算法:

1.按照<横坐标,纵坐标>排序

2.从左到右维护下凸包部分，然后再从右向左维护上凸包部分

```
double andrew(){
    sort(p+1,p+1+n);
    top=0;
    rep(i,1,n){  //顺着扫维护下凸包
        while(top>=2 && sign(area(p[stk[top-1]], p[stk[top]], p[i]))<=0){
            if(sign(area(p[stk[top-1]], p[stk[top]], p[i]))<0)
                st[stk[top--]]=false;
            else top--;
        }
        stk[++top]=i;
        st[i]=true;
    }
    st[1]=false;
    per(i,n,1){ //逆着扫维护上凸包
        if(st[i]) continue;
        while(top>=2&&sign(area(p[stk[top-1]], p[stk[top]], p[i]))<=0){
            top --;
        }
        stk[++top]=i;
    }
    double res=0;
    for(int i=2;i<=top;++i){
        res+=dist(p[stk[i-1]], p[stk[i]]);
    }
    return res;
}
```

半平面交

半平面交就是一堆直线只保留某一个方向的部分最终得到的阴影部分的信息。

人为规定，保留左侧部分而不是右边部分

半平面交算法：

1.按照向量的角度排序(atan2(y,x)将坐标映射到$(-\pi,\pi]$)

2.按照顺序扫描所有向量,将不符合要的的首尾都pop出去

```
double half_plane_intersection(){
    sort(l+1,l+1+idx,cmp);
    hh=0,tt=-1;
    for(int i=1;i<=idx;++i){
        if(i>1&&sign(get_angle(l[i])-get_angle(l[i-1]))==0) continue; //如果角度相同，继续
        while(hh+1<=tt&&on_right(l[i], l[q[tt-1]], l[q[tt]])) tt--;
        while(hh+1<=tt&&on_right(l[i], l[q[hh]], l[q[hh+1]])) hh++;
        q[++tt]=i;
    }
    while(hh+1<=tt&&on_right(l[q[hh]], l[q[tt-1]], l[q[tt]])) tt--;
    while(hh+1<=tt&&on_right(l[q[tt]], l[q[hh]], l[q[hh+1]])) hh++;

    q[++tt]=q[hh]; //将队首加入队列
    int k=0;
    double res=0;
    for(int i=hh;i<tt;++i){
        ans[k++]=get_line_intersection(l[q[i]], l[q[i+1]]);
    }
    for(int i=1;i+1<k;++i)
        res += area(ans[0], ans[i], ans[i+1]);
    return res/2.0;
}
```

三角剖分

求一个圆和一个多边形的面积交.

前置知识：

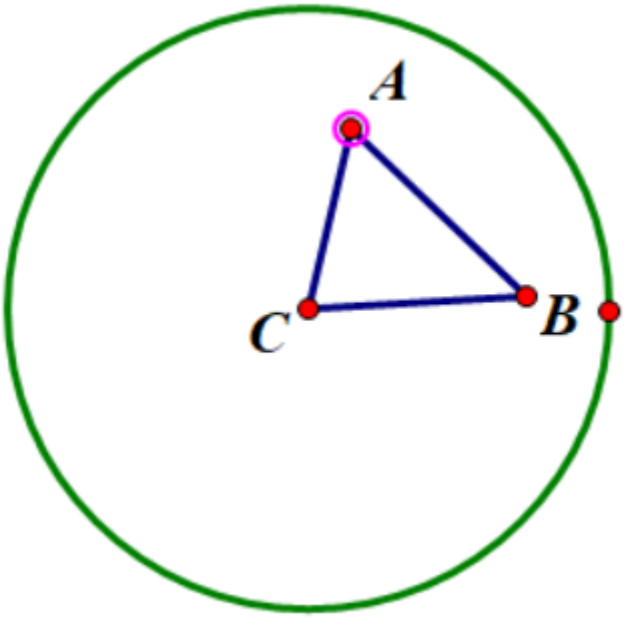
用向量求圆和直线的交点.

> POJ3675 望远镜

做法：

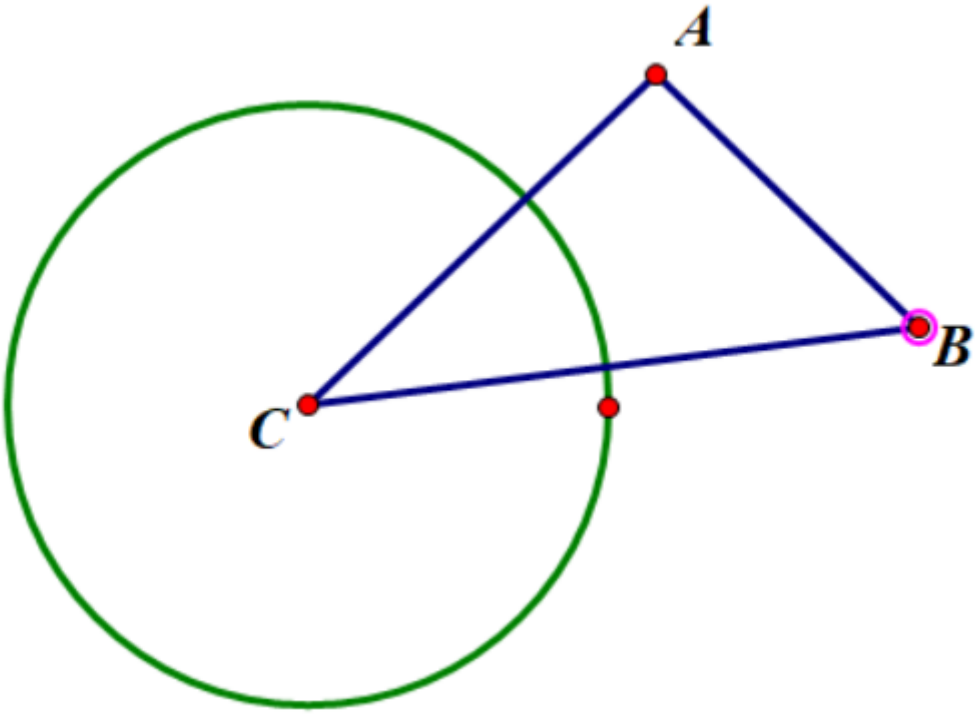
将多边形按照逆时针方向，圆心向所有顶点连边，剖分成多个三角形，然后分类讨论求三角形和圆的面积交即可。

1.A和B都在圆内



$S_{\cap} = S_{ABC}$

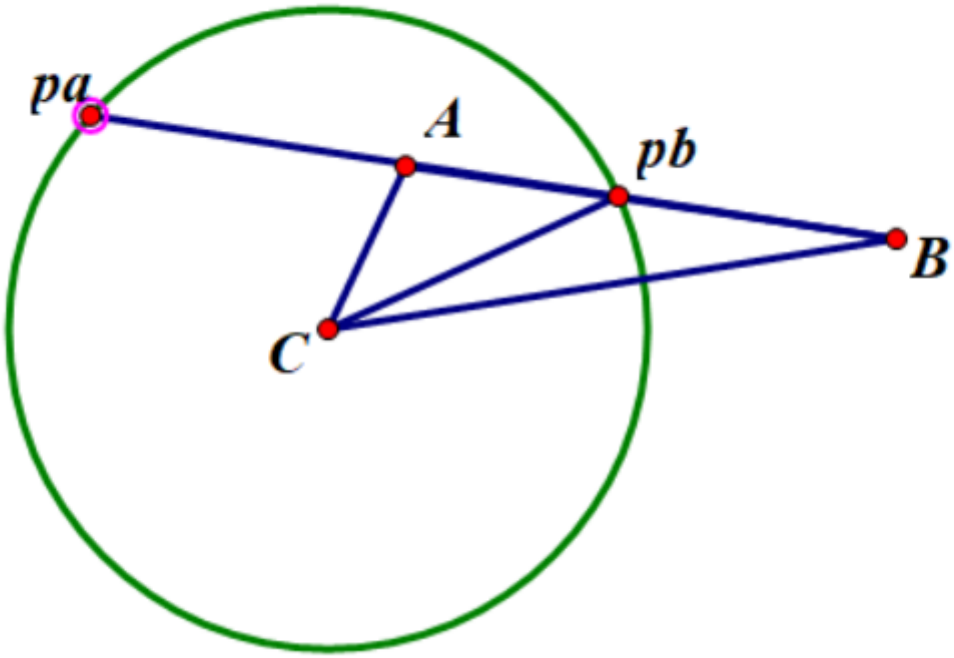
2.A和B都在圆外



$\theta = arccos(\frac{\vec{a}\cdot\vec{b}}{|\vec{a}|\cdot|\vec{b}|})$

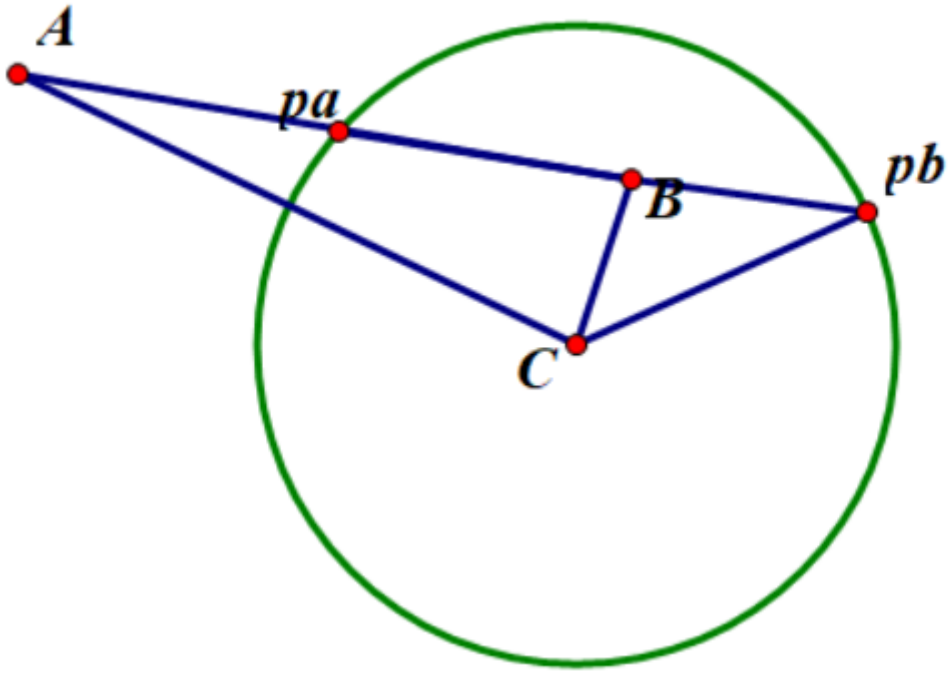
$S_{\cap} = \frac{\theta r^2}{2}$

3.A在圆内B在圆外



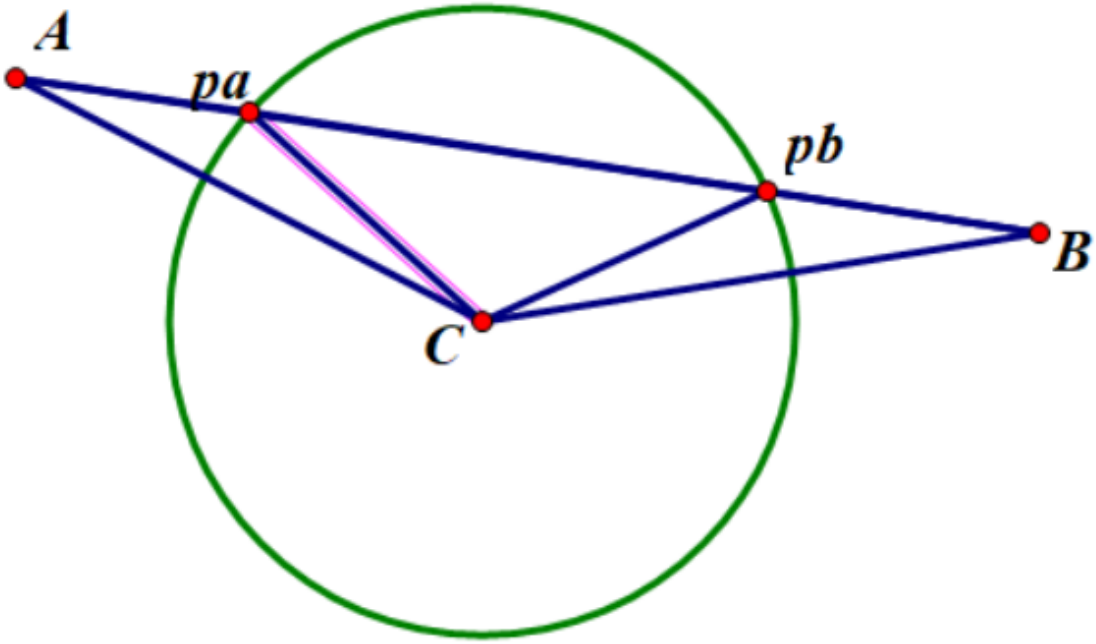
$S = 扇形(Cp_bB) + S_{\triangle ACp_b}$

4.A在圆外B在圆内



此情况与上一情况相同

5.A和B都在圆外的特殊情况



$S = S_{扇形ACp_a} + S_{扇形BCp_b} + S_{\triangle p_aCp_b}$

所以我们先求出直线AB与圆的两个交点和圆心到线段AB的最短距离，然后进行讨论。

> 1.如果a和b距离圆心的距离都小于r，则说明是第一种情况
>
> 2.如果a和b的距离都大于r且圆心到ab线段的距离大于r说明是第二种情况
>
> 3.然后依次讨论da和mind的长度来确定是否是第三种或者第四种情况
>
> 4否则为最后一种情况。

```
double circle_triangle(Poi a,Poi b,Cir c={{0,0},r}){ //求圆c和三角形的面积交
    auto da=dist(c.p,a),db=dist(c.p,b);
    if(sign(c.r-da)>=0&&sign(c.r-db)>=0) return a*b/2;
    if(!sign(a*b)) return  0.0;
     //直线ab和圆的交点
    auto mind=poi_to_segment(a,b,c.p);
    // debug(mind);
    vector<Poi> intersection=get_circle_line_intersection(a,b,c);
    Poi pa,pb;
    // if(intersection.size()<2) return 0.0;
    if(intersection.size()==2)
        pa=intersection[0],pb=intersection[1];
    if(sign(c.r-mind)<=0)   return sector_area(a,b,c);
    if(sign(c.r-da) >= 0) {//a在圆内，b在圆外
        return (a-c.p)*(pb-c.p)/2.0+sector_area(pb,b,c);
    }
    if(sign(c.r-db) >= 0)
        return sector_area(a,pa,c)+(pa-c.p)*(b-c.p)/2.0;
    return sector_area(a,pa,c)+sector_area(pb,b,c)+(pa-c.p)*(pb-c.p)/2.0;
}
```

在这里顺便放一下我的计算几何模板:

Code:

```
#include<unordered_set>
#include<unordered_map>
#include<functional>
#include<algorithm>
#include<string.h>
#include<iostream>
#include<iterator>
#include<cstring>
#include<numeric>
#include<cstdio>
#include<vector>
#include<bitset>
#include<queue>
#include<stack>
#include<cmath>
#include<set>
#include<map>
#define x first
#define y second
using namespace std;
//================================DEBUG
#define FASTIO cin.tie(nullptr) -> sync_with_stdio(false)
#define debug(a) cout << #a": " << a << endl;
#define rep(i, ll, rr) for(int i = ll; i <= rr; ++ i)
#define per(i, rr, ll) for(int i = rr; i >= ll; -- i)
//================================IO
typedef long long LL; typedef unsigned long long ULL; typedef long double LD;
inline LL read(){LL s=0,w=1;char ch=getchar();for(;!isdigit(ch); ch = getchar())if(ch == '-') w
= -1; for (; isdigit(ch);ch = getchar())s=(s<<1)+(s<<3)+(ch^48);return s*w;}
inline void print(LL x,int op=10){if(!x){putchar('0');if(op)putchar(op);return;}char F[40];LL
tmp=x>0?x:-x;if(x<0)putchar('-');int cnt=0;while(tmp>0)
{F[cnt++]=tmp%10+'0';tmp/=10;}while(cnt>0)putchar(F[--cnt]);if(op)putchar(op);}
inline void print128(__int128_t x){if(x < 0) {putchar('-');x = -x;}if(x/10)
print128(x/10);putchar(x%10+'0');}
template <typename T>void read(T &x){x=0;int f=1;char ch=getchar();while(!isdigit(ch)){if(ch=='-
')f=-1;ch=getchar();}while(isdigit(ch)){x=x*10+(ch^48);ch=getchar();}x*=f;return;}
template <typename T>void write(T x){if(x<0){putchar('-');x=-
x;}if(x>9)write(x/10);putchar(x%10+'0');return;}
//================================HABIT
LL fpower(LL a,LL b,LL mod) {LL ans = 1; while(b){ if(b & 1) ans = ans * (a % mod) % mod; a = a
% mod * (a % mod) % mod; b >>= 1;} return ans; }
LL Mod(LL a,LL mod){return (a%mod+mod)%mod;}
LL gcd(LL a,LL b) {return b?gcd(b,a%b):a;}
int mov[8][2]={1,0,0,1,-1,0,0,-1,1,1,-1,-1,1,-1,-1,1};
//================================DEFINE
// #define int LL
#define double long double
typedef pair<int,int> pii;
typedef pair<double,double> pdd;
const double eps=1e-9,PI=acos(-1);
//================================GEOMETRY
int sign(double x) {if(fabs(x)<eps) return 0;return x>0?1:-1;}
```

```
struct Poi{
    double x,y;
    Poi operator-(Poi b){return {x-b.x,y-b.y};}
    Poi operator+(Poi b){return {x+b.x,y+b.y};}
    Poi operator*(double k){return {x*k,y*k};}
    Poi operator/(double k){return {x/k,y/k};}
    Poi norm(){double d=sqrt(x*x+y*y);return {x/d,y/d};}
    double operator*(Poi b){return x*b.y-y*b.x;}
    double operator&(Poi b){return x*b.x+y*b.y;}
    bool operator==(Poi b){return sign(x-b.x)==0&&sign(y-b.y)==0;}
    bool operator<(Poi b){return sign(x-b.x)<0||(sign(x-b.x)==0&&sign(y-b.y)<0);}
};
double cross(Poi a,Poi b){return a.x*b.y-a.y*b.x;}
double area(Poi a,Poi b,Poi c){return cross({b.x-a.x,b.y-a.y},{c.x-a.x,c.y-a.y});}
double dist(Poi a,Poi b){double dx=a.x-b.x;double dy=a.y-b.y;return sqrt(dx*dx+dy*dy);}
struct Cir{Poi p;double r;};
struct Line{Poi st,ed;};
double get_angle(const Line &a){ //获得直线的角度
    return atan2(a.ed.y-a.st.y,a.ed.x-a.st.x);
}
//直线按照角度的排序函数
bool cmp(Line &a,Line &b){  double A=get_angle(a),B=get_angle(b); if(sign(A-B)==0) return
sign(area(a.st,a.ed,b.ed))<0;return A<B;}
//求直线p+kv和直线q+kw的交点
Poi get_line_intersection(Poi p,Poi v,Poi q,Poi w){ auto u=p-q; double t=cross(w,u)/cross(v,w);
return {p.x+v.x*t,p.y+v.y*t};}
//两条线的交点
Poi get_line_intersection(Line a,Line b){ return get_line_intersection(a.st,a.ed-a.st,b.st,b.ed-
b.st);}
//bc的交点是否再a的右侧
bool on_right(Line a,Line b,Line c){ auto jiao=get_line_intersection(b,c);return
sign(area(a.st,a.ed,jiao))<=0;}
//将一个点顺时针旋转d度
Poi rotate(Poi a,double b){return {a.x*cos(b)+a.y*sin(b), -a.x*sin(b)+a.y*cos(b)};}
//获取中垂线
Line get_perpendicular_bisector(Poi a,Poi b){return {(a+b)/2,rotate(b-a,PI/2.0)};}
//三点确定圆
Cir get_cir(Poi a,Poi b,Poi c){auto
u=get_perpendicular_bisector(a,b),v=get_perpendicular_bisector(a,c);auto
p=get_line_intersection(u.st,u.ed,v.st,v.ed);return {p, dist(p,a)};}
/*random_suffle(p+1,p+1+n);  点随机化*/
//================================
const int N=200010,M=N*2,mod=1e9+7;
vector<Poi> p;
Poi m,c;int n;
double r;

double len(Poi a){return sqrt(a&a);}

bool on_segment(Poi p,Poi a,Poi b){ //判断c是否在线段ab上
    return !sign((p-a)*(p-b)) && sign((p-a)&(p-b))<=0;
}

vector<Poi> get_circle_line_intersection(Poi a,Poi b,Cir c={{0,0},r}){ //线段ab和圆c的交点
    vector<Poi> ans;
    auto e=get_line_intersection(a, b-a, c.p, rotate(b-a,PI/2)); //弦与中垂线的交点
    auto d=dist(c.p, e); //弦心距
    if(!on_segment(e,a,b)) d=min(dist(c.p,a), dist(c.p, b));
```

```
    if(sign(c.r-d)<=0) return ans;
    auto len=sqrt(c.r*c.r-dist(c.p, e)*dist(c.p, e));
    Poi pa=e+(a-b).norm()*len,pb=e+((b-a).norm()*len);
    ans.push_back(pa);
    ans.push_back(pb);
    return ans;
}

double poi_to_segment(Poi a,Poi b,Poi c={0,0}){ //点到线段的距离
    auto e=get_line_intersection(a, b-a, c, rotate(b-a,PI/2)); //弦与中垂线的交点
    auto d=dist(c, e); //弦心距
    if(!on_segment(e,a,b)) d=min(dist(c,a), dist(c, b));
    return d;
}

double sector_area(Poi a,Poi b,Cir c){ //c为圆，acb扇形面积
    auto angle=acos((a&b)/len(a)/len(b));
    if(sign(a*b)<0) angle=-angle;
    return c.r*c.r*angle/2.0;
}

double circle_triangle(Poi a,Poi b,Cir c={{0,0},r}){ //求圆c和三角形的面积交
    auto da=dist(c.p,a),db=dist(c.p,b);
    if(sign(c.r-da)>=0&&sign(c.r-db)>=0) return a*b/2;
    if(!sign(a*b)) return  0.0;
     //直线ab和圆的交点
    auto mind=poi_to_segment(a,b,c.p);
    // debug(mind);
    vector<Poi> intersection=get_circle_line_intersection(a,b,c);
    Poi pa,pb;
    // if(intersection.size()<2) return 0.0;
    if(intersection.size()==2)
        pa=intersection[0],pb=intersection[1];
    if(sign(c.r-mind)<=0)   return sector_area(a,b,c);
    if(sign(c.r-da) >= 0) {//a在圆内，b在圆外
        return (a-c.p)*(pb-c.p)/2.0+sector_area(pb,b,c);
    }
    if(sign(c.r-db) >= 0)
        return sector_area(a,pa,c)+(pa-c.p)*(b-c.p)/2.0;
    return sector_area(a,pa,c)+sector_area(pb,b,c)+(pa-c.p)*(pb-c.p)/2.0;
}

double area(){
    double res=0;
    for(int i=0;i<n;++i){
        res += circle_triangle(p[i], p[(i+1)%n]);
    }
    return fabs(res);
}

void solve(){
    p.clear();
    p.resize(n);
    for(auto &u:p){
        scanf("%Lf%Lf",&u.x,&u.y);
    }
    printf("%.2Lf\n",area());
}
```

```
//================================
signed main(){
    // int _=1;
    while(scanf("%Lf%d",&r,&n)!=-1)
        solve();
    return 0;
}
```

三角形的面积并

通过将所有交点竖直分割，然后使用扫描线法对于所有段的两边长度进行求和，用求梯形的方法来求。

```
#include<unordered_set>
#include<unordered_map>
#include<functional>
#include<algorithm>
#include<string.h>
#include<iostream>
#include<iterator>
#include<cstring>
#include<numeric>
#include<cstdio>
#include<vector>
#include<bitset>
#include<queue>
#include<stack>
#include<cmath>
#include<set>
#include<map>
#define x first
#define y second
using namespace std;
//================================DEBUG
#define FASTIO cin.tie(nullptr) -> sync_with_stdio(false)
#define debug(a) cout << #a": " << a << endl;
#define rep(i, ll, rr) for(int i = ll; i <= rr; ++ i)
#define per(i, rr, ll) for(int i = rr; i >= ll; -- i)
//================================IO
typedef long long LL; typedef unsigned long long ULL; typedef long double LD;
inline LL read(){LL s=0,w=1;char ch=getchar();for(;!isdigit(ch); ch = getchar())if(ch == '-') w
= -1; for (; isdigit(ch);ch = getchar())s=(s<<1)+(s<<3)+(ch^48);return s*w;}
inline void print(LL x,int op=10){if(!x){putchar('0');if(op)putchar(op);return;}char F[40];LL
tmp=x>0?x:-x;if(x<0)putchar('-');int cnt=0;while(tmp>0)
{F[cnt++]=tmp%10+'0';tmp/=10;}while(cnt>0)putchar(F[--cnt]);if(op)putchar(op);}
inline void print128(__int128_t x){if(x < 0) {putchar('-');x = -x;}if(x/10)
print128(x/10);putchar(x%10+'0');}
template <typename T>void read(T &x){x=0;int f=1;char ch=getchar();while(!isdigit(ch)){if(ch=='-
')f=-1;ch=getchar();}while(isdigit(ch)){x=x*10+(ch^48);ch=getchar();}x*=f;return;}
template <typename T>void write(T x){if(x<0){putchar('-');x=-
x;}if(x>9)write(x/10);putchar(x%10+'0');return;}
//================================HABIT
LL fpower(LL a,LL b,LL mod) {LL ans = 1; while(b){ if(b & 1) ans = ans * (a % mod) % mod; a = a
% mod * (a % mod) % mod; b >>= 1;} return ans; }
LL Mod(LL a,LL mod){return (a%mod+mod)%mod;}
LL gcd(LL a,LL b) {return b?gcd(b,a%b):a;}
```

```
int mov[8][2]={1,0,0,1,-1,0,0,-1,1,1,-1,-1,1,-1,-1,1};
//================================DEFINE
// #define int LL
// #define double long double
typedef pair<int,int> pii;
typedef pair<double,double> pdd;
const double eps=1e-18,PI=acos(-1),inf=1e9;
//================================GEOMETRY
int sign(double x) {if(fabs(x)<eps) return 0;return x>0?1:-1;}
struct Poi{
    double x,y;
    Poi operator-(Poi b){return {x-b.x,y-b.y};}
    Poi operator+(Poi b){return {x+b.x,y+b.y};}
    Poi operator*(double k){return {x*k,y*k};}
    Poi operator/(double k){return {x/k,y/k};}
    Poi norm(){double len=sqrt(x*x+y*y);return {x/len,y/len};}
    double operator*(Poi b){return x*b.y-y*b.x;}
    double operator&(Poi b){return x*b.x+y*b.y;}
    bool operator==(Poi b){return sign(x-b.x)==0&&sign(y-b.y)==0;}
    bool operator<(Poi b){return sign(x-b.x)<0||(sign(x-b.x)==0&&sign(y-b.y)<0);}
};
double cross(Poi a,Poi b){return a.x*b.y-a.y*b.x;}
double area(Poi a,Poi b,Poi c){return cross({b.x-a.x,b.y-a.y},{c.x-a.x,c.y-a.y});}
double dist(Poi a,Poi b){double dx=a.x-b.x;double dy=a.y-b.y;return sqrt(dx*dx+dy*dy);}
struct Cir{Poi p;double r;};
struct Line{Poi st,ed;};
double get_angle(const Line &a){ //获得直线的角度
    return atan2(a.ed.y-a.st.y,a.ed.x-a.st.x);
}
bool on_segment(Poi p,Poi a,Poi b){ return sign((p-a)&(p-b))<=0;}//判断c是否在线段ab上
//直线按照角度的排序函数
bool cmp(Line &a,Line &b){  double A=get_angle(a),B=get_angle(b); if(sign(A-B)==0) return
sign(area(a.st,a.ed,b.ed))<0;return A<B;}
//求直线p+kv和直线q+kw的交点
Poi get_line_intersection(Poi p,Poi v,Poi q,Poi w){
    if(!sign(v*w)) return {inf,inf};
    auto u=p-q; double t=cross(w,u)/cross(v,w);
    auto jiao=p+v*t;
    if(!on_segment(jiao,p,p+v) || !on_segment(jiao,q,q+w))
        return {inf,inf};
    return jiao;
}
//两条线的交点
Poi get_line_intersection(Line a,Line b){ return get_line_intersection(a.st,a.ed-a.st,b.st,b.ed-
b.st);}
//bc的交点是否再a的右侧
bool on_right(Line a,Line b,Line c){ auto jiao=get_line_intersection(b,c);return
sign(area(a.st,a.ed,jiao))<=0;}
//将一个点顺时针旋转d度
Poi rotate(Poi a,double b){return {a.x*cos(b)+a.y*sin(b), -a.x*sin(b)+a.y*cos(b)};}
//获取中垂线
Line get_perpendicular_bisector(Poi a,Poi b){return {(a+b)/2,rotate(b-a,PI/2.0)};}
//三点确定圆
Cir get_cir(Poi a,Poi b,Poi c){auto
u=get_perpendicular_bisector(a,b),v=get_perpendicular_bisector(a,c);auto
p=get_line_intersection(u.st,u.ed,v.st,v.ed);return {p, dist(p,a)};}
double len(Poi a){return sqrt(a&a);}
vector<Poi> get_circle_line_intersection(Poi a,Poi b,Cir c){ //线段ab和圆c的交点
```

```
    vector<Poi> ans;
    auto e=get_line_intersection(a, b-a, c.p, rotate(b-a,PI/2)); //弦与中垂线的交点
    auto d=dist(c.p, e); //弦心距
    if(!on_segment(e,a,b)) d=min(dist(c.p,a), dist(c.p, b));
    if(sign(c.r-d)<=0) return ans;
    auto len=sqrt(c.r*c.r-dist(c.p, e)*dist(c.p, e));
    Poi pa=e+(a-b).norm()*len,pb=e+((b-a).norm()*len);
    ans.push_back(pa);
    ans.push_back(pb);
    return ans;
}
double poi_to_segment(Poi a,Poi b,Poi c={0,0}){ //点到线段的距离
    auto e=get_line_intersection(a, b-a, c, rotate(b-a,PI/2)); //弦与中垂线的交点
    auto d=dist(c, e); //弦心距
    if(!on_segment(e,a,b)) d=min(dist(c,a), dist(c, b));
    return d;
}
//c为圆，acb扇形面积
double sector_area(Poi a,Poi b,Cir c){ auto angle=acos((a&b)/len(a)/len(b));if(sign(a*b)<0)
angle=-angle;return c.r*c.r*angle/2.0;}
/*random_suffle(p+1,p+1+n);  点随机化*/

/* 3D-GEOMETRY
double rand_eps(){return ((double)rand()/RAND_MAX-0.5)*eps;}
struct Point{ //三维点 or 向量
    double x,y,z;
    void shake(){x+=rand_eps(),y+=rand_eps(),z+=rand_eps();}
    Point operator+(Point b){return {x+b.x,y+b.y,z+b.z};}
    Point operator-(Point b){return {x-b.x,y-b.y,z-b.z};}
    double operator&(Point t){return x*t.x+y*t.y+z*t.z;} //点积
    Point operator*(Point t){return {y*t.z-t.y*z, z*t.x-x*t.z, x*t.y-y*t.x};}
    double len(){return sqrt(x*x+y*y+z*z);}
};
struct Plane{ //平面
    int v[3];
    Point norm(){return (q[v[1]]-q[v[0]])*(q[v[2]]-q[v[0]]);}
    double area() { return norm().len() / 2;}
    bool above(Point a){return ((a-q[v[0]])&norm()) >=0 ;}
    double dist(Point W){return (norm()&(q[v[0]]-W))/(norm().len());} //点到平面的距离
};*/
//================================
const int N=200010,M=N*2,mod=1e9+7,INF=1e9;
int n,m,k,a[N];
Poi tr[N][3];
Poi q[N];

double line_area(double a,int tag){
    int idx=0;
    for(int i=0;i<n;++i){
        auto t=tr[i];
        if(sign(t[2].x-a)<0||sign(t[0].x-a)>0) continue;
        if(!sign(t[0].x-a) && !sign(t[1].x-a)){
            if(tag) {
                q[idx++]={t[0].y, t[1].y};
            }
        }
        else if(!sign(t[1].x-a)&&!sign(t[2].x-a)){
            if(!tag) q[idx++]={t[2].y,t[1].y};
```

```
        }
        else{
            int u=0;
            static double tmp[3];
            for(int j=0;j<3;++j){
                auto jiao=get_line_intersection(t[j],t[(j+1)%3]-t[j],{a,-inf},{0,inf*2});
                if(sign(jiao.x-inf))
                    tmp[u++]=jiao.y;
            }
            if(!u) continue;
            sort(tmp,tmp+u);
            q[idx++]={tmp[0],tmp[u-1]};
        }
    }
    if(!idx) return 0.0;
    for(int i=0;i<idx;++i)
        if(sign(q[i].x-q[i].y)>0) swap(q[i].x,q[i].y);
    sort(q,q+idx);
    double res=0;
    double st=q[0].x,ed=q[0].y;
    for(int i=0;i<idx;++i){
        if(sign(q[i].x-ed)<=0) ed=max(ed, q[i].y);
        else{
            res += ed-st;
            st=q[i].x,ed=q[i].y;
        }
    }
    res+=ed-st;
    return res;
}

double range_area(double a,double b){
    // cout << " " << line_area(a,1) << " " << line_area(b,0) << "\n";
    return (line_area(a,1)+line_area(b,0))*(b-a)/2;
}

void solve(){
    n=read();
    vector<double> xs;
    for(int i=0;i<n;++i){
        for(int j=0;j<3;++j){
            scanf("%lf%lf",&tr[i][j].x,&tr[i][j].y);
            xs.push_back(tr[i][j].x);
        }
        sort(tr[i],tr[i]+3);
    }
    for(int i=0;i<n;++i){
        for(int j=0;j<n;++j){
            for(int x=0;x<3;++x){
                for(int y=0;y<3;++y){
                    auto jiao=get_line_intersection(tr[i][x], tr[i][(x+1)%3]-tr[i][x],tr[j]
[y],tr[j][(y+1)%3]-tr[j][y]);
                    if(sign(jiao.x-inf))
                        xs.push_back(jiao.x);
                }
            }
        }
    }
```

```
    sort(xs.begin(), xs.end());

    double res=0;
    for(int i=0;i+1<xs.size();++i){
        if(sign(xs[i]-xs[i+1]))
        {
            // cout << xs[i] << " " << xs[i+1] ;
            res += range_area(xs[i], xs[i+1]);
        }
    }
    printf("%.2lf\n",res);
}
//==============================
signed main(){
    int _=1;
    // _=read();
    while(_--)  solve();
    return 0;
}
```

三维计算几何

1.以任意顺序存储每一个面，将每一个面分割成若干三角形。

2.使用增量法，对于新加入的点，对当前的三维凸包做一个光照，所有能够找到的平面的轮廓组成的平面与新的点组成新的三维凸包的边界。

```
const int N=200010,M=N*2,mod=1e9+7;
int n,m,k,a[N];
string p;
bool g[110][110];

double rand_eps(){return ((double)rand()/RAND_MAX-0.5)*eps;}

struct Point{
    double x,y,z;
    void shake(){
        x+=rand_eps(),y+=rand_eps(),z+=rand_eps();
    }
    Point operator+(Point b){
        return {x+b.x,y+b.y,z+b.z};
    }
    Point operator-(Point b){
        return {x-b.x,y-b.y,z-b.z};
    }
    double operator&(Point t){ //点积
        return x*t.x+y*t.y+z*t.z;
    }
    Point operator*(Point t){
        return {y*t.z-t.y*z, z*t.x-x*t.z, x*t.y-y*t.x};
    }
    double len(){
        return sqrt(x*x+y*y+z*z);
    }
}q[N],D;
```

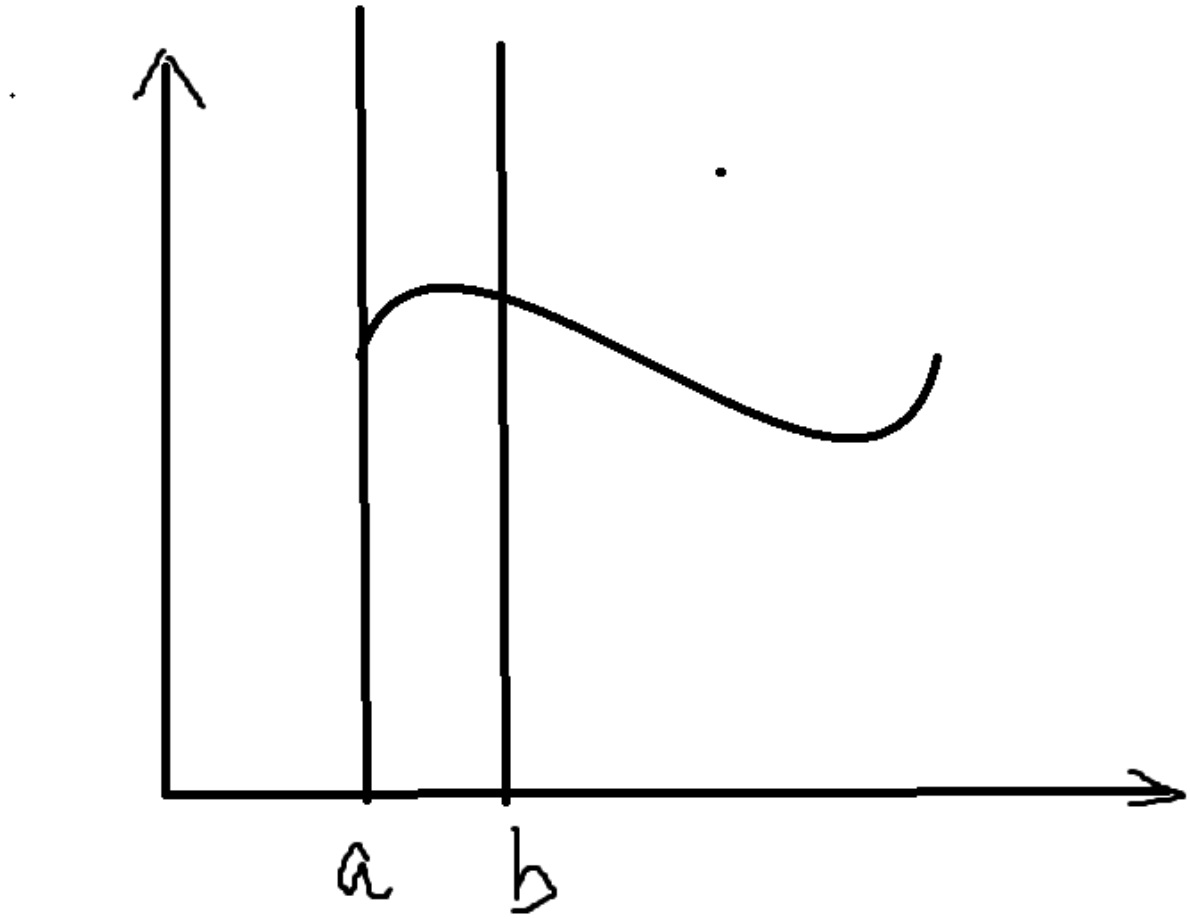
```
struct Plane{
    int v[3];
    Point norm(){
        return (q[v[1]]-q[v[0]])*(q[v[2]]-q[v[0]]);
    }
    double area() {
        return norm().len() / 2;
    }
    bool above(Point a){
        return ((a-q[v[0]])&norm()) >=0 ;
    }
    //点到平面的距离
    double dist(Point W){return (norm()&(q[v[0]]-W))/(norm().len());}
}plane[N],np[N];

void get_convex_3d(){
    plane[m ++]={0,1,2};
    plane[m ++]={2,1,0};
    for(int i=3;i<n;++i){
        int cnt=0;
        for(int j=0;j<m;++j){
            bool t=plane[j].above(q[i]);
            if(!t) np[cnt ++]=plane[j];
            for(int k=0;k<3;k++)
                g[plane[j].v[k]][plane[j].v[(k+1)%3]]=t;
        }
        for(int j=0;j<m;++j){
            for(int k=0;k<3;++k){
                int a=plane[j].v[k],b=plane[j].v[(k+1)%3];
                if(g[a][b]&&!g[b][a])
                    np[cnt++]={a,b,i};
            }
        }
        m=cnt;
        for(int j=0;j<m;++j) plane[j]=np[j];
    }
}
```

辛普森积分法

在求一段不规则图形的面积的时候，使用抛物线来近似曲线。

我们在学习积分的时候常常使用长方形来近似无限分割后的曲线的积分面积。但是如果在数值计算的时候，这样会造成不小的精度误差。因此辛普森积分使用抛物线上的曲线来近似$(a, f(a)), (b, f(b))$这一段曲线。

因此对于已知的函数方程



我们用抛物线来近似a,b之间的面积。即通过确定的$(a, f(a))$，$(\frac{a+b}{2}, f(\frac{a+b}{2}))$和$(b, f(b))$三个点确定的抛物线积分而得的面积$S = \frac{b-a}{6}(f(a) + 4f(\frac{a+b}{2}) + f(b))$

自适应辛普森积分：

对于一段面积，先用辛普森积分法求出整段的面积S。然后找到中点mid，分别求出左边的面积L和右边的面积R。

```
while(fabs(S-L-R)>eps){
    S=L+R;
    递归对于每个区间接着做。
}
```

模板:（只需要改动定义的f函数即可。）

```
double f(double x){

}

double simpson(double l,double r){
    return (r-l)*(f(l)+f((l+r)/2)*4+f(r))/6;
}

double dfs(double l,double r,double S){
    double mid=(l+r)/2;
```

```
    double L=simpson(l,mid),R=simpson(mid,r);
    if(fabs(L+R-S)<eps) return L+R;
    return dfs(l,mid,L)+dfs(mid,r,R);
}

void solve(){
    double L=2000,R=-2000;
    scanf("%d",&n);
    for(int i=0;i<n;++i){
        scanf("%lf%lf%lf",&c[i].p.x,&c[i].p.y,&c[i].r);
        L=min(L,c[i].p.x-c[i].r);
        R=max(R,c[i].p.x+c[i].r);
    }
    double S=simpson(L,R);
    printf("%.3lf",dfs(L-100,R+100,S));
}
```

旋转卡壳

给定一个二维平面，上面有很多点。求平面最远点对。

性质1：平面最远点对的两个点一定在凸包的边缘上。(缩小搜索范围)

2.用两条平行线进行旋转，两条平行线在旋转过程中的最大值就是最远点对距离

> 枚举的时候枚举一些离散的值，由于对踵点是成对出现的，所以对踵点距离的最大值就是最远点对

因此就是找出点集的凸包，然后对于凸包的每一条边，找到到这个边最远的点，用当前边的两个端点和最远点来更新最远点对距离。

由于随着凸包的边进行遍历的时候(角度逐渐变大)，最远点的角度也在不断增大。因此可以使用双指针算法。

```
void andrew(){
    sort(p+1,p+1+n);
    top=0;
    rep(i,1,n){  //顺着扫维护下凸包
        while(top>=2 && area(p[stk[top-1]], p[stk[top]], p[i])<=0){
            if(area(p[stk[top-1]], p[stk[top]], p[i])<0)
                st[stk[top--]]=false;
            else top--;
        }
        stk[++top]=i;
        st[i]=true;
    }
    st[1]=false;
    per(i,n,1){ //逆着扫维护上凸包
        if(st[i]) continue;
        while(top>=2&&area(p[stk[top-1]], p[stk[top]], p[i])<=0){
            top --;
        }
        stk[++top]=i;
    }
    --top;
}
```

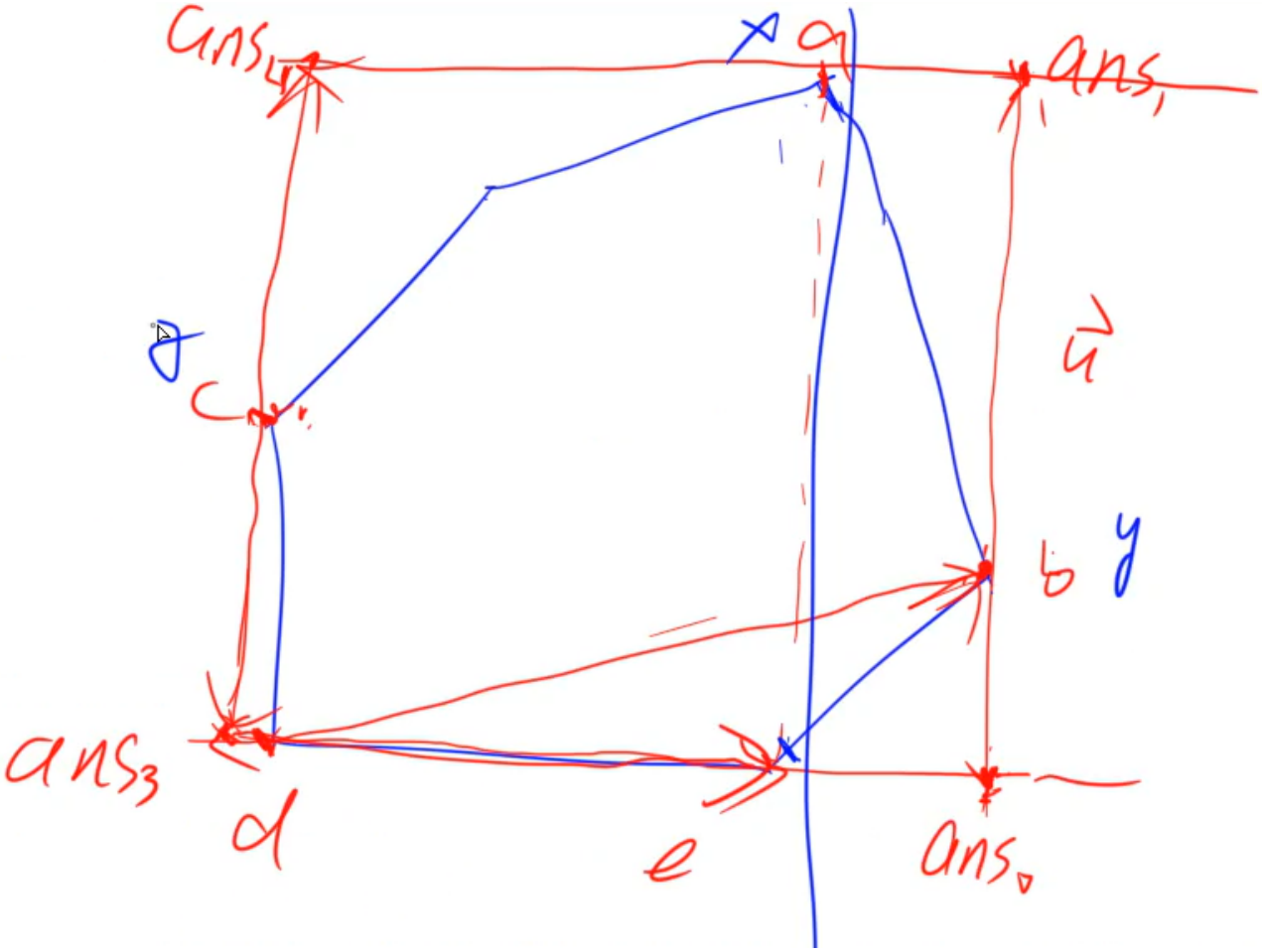
```
int rotating_calipers(){
    if(top<=2) return dist(p[1],p[n]);
    int res=0;
    for(int i=1,j=2;i<=top;++i){
        auto d=p[stk[i]],e=p[stk[i+1]];
        while(area(d,e,p[stk[j]])<area(d,e,p[stk[j+1]])) j=j%top+1;
        res=max(res, max(dist(d,p[stk[j]]),dist(e,p[stk[j]])));
    }
    return res;
}
```

最小矩形覆盖

已知平面上不共线的一组点的坐标，求覆盖这组点的面积最小的矩形。输出矩形面积和四个顶点的坐标

性质：矩形上的边与凸包上的某条边共线

使用旋转卡壳算法，在凸包上维护转动线的同时双指针维护




```
#include<unordered_set>
#include<unordered_map>
#include<functional>
#include<algorithm>
```

```
#include<string.h>
#include<iostream>
#include<iterator>
#include<cstring>
#include<numeric>
#include<cstdio>
#include<vector>
#include<bitset>
#include<queue>
#include<stack>
#include<cmath>
#include<set>
#include<map>
#define x first
#define y second
using namespace std;
//================================DEBUG
#define FASTIO cin.tie(nullptr) -> sync_with_stdio(false)
#define debug(a) cout << #a": " << a << endl;
#define rep(i, ll, rr) for(int i = ll; i <= rr; ++ i)
#define per(i, rr, ll) for(int i = rr; i >= ll; -- i)
//================================IO
typedef long long LL; typedef unsigned long long ULL; typedef long double LD;
inline LL read(){LL s=0,w=1;char ch=getchar();for(;!isdigit(ch); ch = getchar())if(ch == '-') w
= -1; for (; isdigit(ch);ch = getchar())s=(s<<1)+(s<<3)+(ch^48);return s*w;}
inline void print(LL x,int op=10){if(!x){putchar('0');if(op)putchar(op);return;}char F[40];LL
tmp=x>0?x:-x;if(x<0)putchar('-');int cnt=0;while(tmp>0)
{F[cnt++]=tmp%10+'0';tmp/=10;}while(cnt>0)putchar(F[--cnt]);if(op)putchar(op);}
inline void print128(__int128_t x){if(x < 0) {putchar('-');x = -x;}if(x/10)
print128(x/10);putchar(x%10+'0');}
template <typename T>void read(T &x){x=0;int f=1;char ch=getchar();while(!isdigit(ch)){if(ch=='-
')f=-1;ch=getchar();}while(isdigit(ch)){x=x*10+(ch^48);ch=getchar();}x*=f;return;}
template <typename T>void write(T x){if(x<0){putchar('-');x=-
x;}if(x>9)write(x/10);putchar(x%10+'0');return;}
//================================HABIT
LL fpower(LL a,LL b,LL mod) {LL ans = 1; while(b){ if(b & 1) ans = ans * (a % mod) % mod; a = a
% mod * (a % mod) % mod; b >>= 1;} return ans; }
LL Mod(LL a,LL mod){return (a%mod+mod)%mod;}
LL gcd(LL a,LL b) {return b?gcd(b,a%b):a;}
int mov[8][2]={1,0,0,1,-1,0,0,-1,1,1,-1,-1,1,-1,-1,1};
//================================DEFINE
// #define int LL
// #define double long double
typedef pair<int,int> pii;
typedef pair<double,double> pdd;
const double eps=1e-9,PI=acos(-1);
//================================GEOMETRY
int sign(double x) {if(fabs(x)<eps) return 0;return x>0?1:-1;}
struct Poi{
    double x,y;
    Poi operator-(Poi b){return {x-b.x,y-b.y};}
    Poi operator+(Poi b){return {x+b.x,y+b.y};}
    Poi operator*(double k){return {x*k,y*k};}
    Poi operator/(double k){return {x/k,y/k};}
    double operator*(Poi b){return x*b.y-y*b.x;}
    double operator&(Poi b){return x*b.x+y*b.y;}
    bool operator==(Poi b){return sign(x-b.x)==0&&sign(y-b.y)==0;}
    bool operator<(Poi b){return sign(x-b.x)<0||(sign(x-b.x)==0&&sign(y-b.y)<0);}
```

```
    double len(){return sqrt(x*x+y*y);}
};
double cross(Poi a,Poi b){return a.x*b.y-a.y*b.x;}
double area(Poi a,Poi b,Poi c){return cross({b.x-a.x,b.y-a.y},{c.x-a.x,c.y-a.y});}
double dist(Poi a,Poi b){double dx=a.x-b.x;double dy=a.y-b.y;return sqrt(dx*dx+dy*dy);}
struct Cir{Poi p;double r;};
struct Line{Poi st,ed;};
double get_angle(const Line &a){ //获得直线的角度
    return atan2(a.ed.y-a.st.y,a.ed.x-a.st.x);
}
//直线按照角度的排序函数
bool cmp(Line &a,Line &b){  double A=get_angle(a),B=get_angle(b); if(sign(A-B)==0) return
sign(area(a.st,a.ed,b.ed))<0;return A<B;}
//求直线p+kv和直线q+kw的交点
Poi get_line_intersection(Poi p,Poi v,Poi q,Poi w){ auto u=p-q; double t=cross(w,u)/cross(v,w);
return {p.x+v.x*t,p.y+v.y*t};}
//两条线的交点
Poi get_line_intersection(Line a,Line b){ return get_line_intersection(a.st,a.ed-a.st,b.st,b.ed-
b.st);}
//bc的交点是否再a的右侧
bool on_right(Line a,Line b,Line c){ auto jiao=get_line_intersection(b,c);return
sign(area(a.st,a.ed,jiao))<=0;}
//将一个点顺时针旋转d度
Poi rotate(Poi a,double b){return {a.x*cos(b)+a.y*sin(b), -a.x*sin(b)+a.y*cos(b)};}
//获取中垂线
Line get_perpendicular_bisector(Poi a,Poi b){return {(a+b)/2,rotate(b-a,PI/2.0)};}
//三点确定圆
Cir get_cir(Poi a,Poi b,Poi c){auto
u=get_perpendicular_bisector(a,b),v=get_perpendicular_bisector(a,c);auto
p=get_line_intersection(u.st,u.ed,v.st,v.ed);return {p, dist(p,a)};}
/*random_suffle(p+1,p+1+n);  点随机化*/
//================================
const int N=200010,M=N*2,mod=1e9+7;
int n,m,k,a[N];
vector<Poi> p,ans;
double min_area=2e18;
bool vis[N];
int stk[N],top;
bool st[N];

double len(Poi a){return sqrt(a&a);}

//投影
double project(Poi a,Poi b,Poi c){
    return  ((b-a)&(c-a))/len(b-a);
}

Poi norm(Poi a){
    return a/len(a);
}

void get_convex(){
    sort(p.begin(), p.end());
    for(int i=0;i<p.size();++i){
        while(top>=2&&sign(area(p[stk[top-1]],p[stk[top]],p[i]))<=0){
            if(sign(area(p[stk[top-1]],p[stk[top]],p[i]))<0)
                st[stk[top]]=false;
            --top;
```

```
        }
        stk[++top]=i;
        st[i]=true;
    }
    st[0]=false;
    for(int i=p.size()-1;i>=0;--i){
        if(st[i]) continue;
        while(top>=2&&sign(area(p[stk[top-1]],p[stk[top]],p[i]))<=0)
            top --;
        stk[++top]=i;
    }
    --top;
}

void rotating_calipers(){
    for(int i=1,a=3,b=3,c=3;i<=top;++i){
        auto d=p[stk[i]],e=p[stk[i+1]];
        while(sign(area(d,e,p[stk[a]])-area(d,e,p[stk[a+1]])) < 0) a=a%top+1; //旋转卡壳，距离最远点
        while(sign(project(d,e,p[stk[b]])-project(d,e,p[stk[b+1]])) < 0) b=b%top+1; //正向投影最长的
        if(i==1) c=a;
        while(sign(project(d,e,p[stk[c]])-project(d,e,p[stk[c+1]])) > 0) c=c%top+1; //反向投影最长的
        auto x=p[stk[a]],y=p[stk[b]],z=p[stk[c]];
        auto h=area(d,e,x)/len(e-d);
        auto w=((y-z)&(e-d))/len(e-d);

        if(h*w<min_area){
            min_area=h*w;
            ans[0]=d+norm(e-d)*project(d,e,y);
            ans[3]=d+norm(e-d)*project(d,e,z);
            auto u=norm(rotate(e-d,-PI/2));
            ans[1]=ans[0]+u*h;
            ans[2]=ans[3]+u*h;
        }
    }
}

void solve(){
    scanf("%d",&n);
    p.resize(n); ans.resize(4);
    for(auto &u:p){
        scanf("%lf%lf",&u.x,&u.y);
    }
    get_convex();
    rotating_calipers();
    int k=0;
    for(int i=1;i<4;++i){
        if(sign(ans[i].y-ans[k].y)<0||sign(ans[i].y-ans[k].y)==0&&sign(ans[i].x-ans[k].x)<0){
            k=i;
        }
    }
    printf("%.5lf\n",min_area);
    for(int i=0;i<4;++i,k=(k+1)%4){
        double x=ans[k].x,y=ans[k].y;
        if(sign(x)==0) x=0.0;
        if(sign(y)==0) y=0.0;
```

```
        printf("%.51f %.51f\n",x,y);
    }
}
//================================
signed main(){
    int _=1;
    // _=read();
    while(_--)  solve();
    return 0;
}
```

最小圆覆盖

引出：找一个半径最小的圆将平面上的所有的点都盖住。

性质1:最小覆盖圆是唯一的

性质2:若p不在S的最小覆盖圆内部，则p一定在$\{p\} \cup S$的最小覆盖圆的边上

算法步骤：

1.将所有点随机化

2.`for(int i=2;i<=n;++i)`枚举每一个点，在点集中加入新点`p[i]`:

> 1.在当前的最小覆盖原内的话直接continue
>
> 2.如果不在最小覆盖圆内的话，必定在最小覆盖圆的边上
>
> 圆<-p[i]
>
> for(int j=1;j<i;++j)
>
> if j not in cir则j一定在1~j，i这j+1个点且i在cir边上的最小覆盖圆
>
> for(k=1;k<j;++k)
>
> if k not in cir则k一定在圆边上

```
random_shuffle(q,q+n);
Cir c{q[0], 0};
for(int i=1;i<n;++i){
    if(sign(c.r-dist(c.p,q[i]))<0){
        c={q[i], 0};
        for(int j=0;j<i;++j){
            if(sign(c.r-dist(c.p,q[j]))<0){
                c={(q[i]+q[j])/2.0, dist(q[i],q[j])/2.0};
                for(int k=0;k<j;++k){
                    if(sign(c.r-dist(c.p,q[k]))<0)
                    c=get_cir(q[i],q[j],q[k]);
                }
            }
        }
```

```
    }
}
```