```cpp
using point_t=long double;  //全局数据类型，可修改为 long long 等

constexpr point_t eps=1e-8;
constexpr long double PI=3.14159265358979323841;

// 点与向量
template<typename T> struct point
{
    T x,y;

    bool operator==(const point &a) const {return (abs(x-a.x)<=eps && abs(y-a.y)<=eps);}
    bool operator<(const point &a) const {if (abs(x-a.x)<=eps) return y<a.y-eps; return x<a.x-eps;}
    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*(const T k) const {return {k*x,k*y};}
    point operator/(const T k) const {return {x/k,y/k};}
    T operator*(const point &a) const {return x*a.x+y*a.y;}  // 点积
    T operator^(const point &a) const {return x*a.y-y*a.x;}  // 叉积，注意优先级
    int toleft(const point &a) const {const auto t=(*this)^a; return (t>eps)-(t<-eps);}  // to-left 测试
    T len2() const {return (*this)*(*this);}  // 向量长度的平方
    T dis2(const point &a) const {return (a-(*this)).len2();}  // 两点距离的平方

    // 涉及浮点数
    long double len() const {return sqrtl(len2());}  // 向量长度
    long double dis(const point &a) const {return sqrtl(dis2(a));}  // 两点距离
    long double ang(const point &a) const {return acosl(max
```

```
        (-1.0l,min(1.0l,((*this)*a)/(len()*a.len())));}  // 向量夹角
29      point rot(const long double rad) const {return {x*cos(rad)-y*sin(rad),x*sin(rad)+y*cos(rad)};}  // 逆时针旋转（给定角度）
30      point rot(const long double cosr,const long double sinr) const {return {x*cosr-y*sinr,x*sinr+y*cosr};}  // 逆时针旋转（给定角度的正弦与余弦）
31  };
32
33  using Point=point<point_t>;
34
35  // 极角排序
36  struct argcmp
37  {
38      bool operator()(const Point &a,const Point &b) const
39      {
40          const auto quad=[](const Point &a)
41          {
42              if (a.y<-eps) return 1;
43              if (a.y>eps) return 4;
44              if (a.x<-eps) return 5;
45              if (a.x>eps) return 3;
46              return 2;
47          };
48          const int qa=quad(a),qb=quad(b);
49          if (qa!=qb) return qa<qb;
50          const auto t=a^b;
51          // if (abs(t)<=eps) return a*a<b*b-eps;  // 不同长度的向量需要分开
52          return t>eps;
53      }
54  };
55
56  // 直线
57  template<typename T> struct line
58  {
59      point<T> p,v;  // p 为直线上一点，v 为方向向量
60
61      bool operator==(const line &a) const {return v.toleft(a.v)==0 && v.toleft(p-a.p)==0;}
62      int toleft(const point<T> &a) const {return v.toleft(a-p);}  // to-left 测
```

```
63      bool operator<(const line &a) const  // 半平面交算法定义的排序
64      {
65          if (abs(v^a.v)<=eps && v*a.v>=-eps) return toleft(a.p)==-1;
66          return argcmp()(v,a.v);
67      }
68
69      // 涉及浮点数
70      point<T> inter(const line &a) const {return p+v*((a.v^(p-a.p))/(v^a.v));}  // 直线交点
71      long double dis(const point<T> &a) const {return abs(v^(a-p))/v.len();}  // 点到直线距离
72      point<T> proj(const point<T> &a) const {return p+v*((v*(a-p))/(v*v));}  // 点在直线上的投影
73  };
74
75  using Line=line<point_t>;
76
77  //线段
78  template<typename T> struct segment
79  {
80      point<T> a,b;
81
82      bool operator<(const segment &s) const {return make_pair(a,b)<make_pair(s.a,s.b);}
83
84      // 判定性函数建议在整数域使用
85
86      // 判断点是否在线段上
87      // -1 点在线段端点 | 0 点不在线段上 | 1 点严格在线段上
88      int is_on(const point<T> &p) const
89      {
90          if (p==a || p==b) return -1;
91          return (p-a).toleft(p-b)==0 && (p-a)*(p-b)<-eps;
92      }
93
94      // 判断线段直线是否相交
95      // -1 直线经过线段端点 | 0 线段和直线不相交 | 1 线段和直线严格相交
96      int is_inter(const line<T> &l) const
```

```
97      {
98          if (l.toleft(a)==0 || l.toleft(b)==0) return -1;
99          return l.toleft(a)!=l.toleft(b);
100     }
101
102     // 判断两线段是否相交
103     // -1 在某一线段端点处相交 | 0 两线段不相交 | 1 两线段严格相交
104     int is_inter(const segment<T> &s) const
105     {
106         if (is_on(s.a) || is_on(s.b) || s.is_on(a) || s.
                is_on(b)) return -1;
107         const line<T> l{a,b-a},ls{s.a,s.b-s.a};
108         return l.toleft(s.a)*l.toleft(s.b)==-1 && ls.toleft(
                a)*ls.toleft(b)==-1;
109     }
110
111     // 点到线段距离
112     long double dis(const point<T> &p) const
113     {
114         if ((p-a)*(b-a)<-eps || (p-b)*(a-b)<-eps) return min
                (p.dis(a),p.dis(b));
115         const line<T> l{a,b-a};
116         return l.dis(p);
117     }
118
119     // 两线段间距离
120     long double dis(const segment<T> &s) const
121     {
122         if (is_inter(s)) return 0;
123         return min({dis(s.a),dis(s.b),s.dis(a),s.dis(b)});
124     }
125 };
126
127 using Segment=segment<point_t>;
128
129 // 多边形
130 template<typename T> struct polygon
131 {
132     vector<point<T>> p;  // 以逆时针顺序存储
133
134     size_t nxt(const size_t i) const {return i==p.size()
            -1?0:i+1;}
```

```
135     size_t pre(const size_t i) const {return i==0?p.size()
            -1:i-1;}
136
137     // 回转数
138     // 返回值第一项表示点是否在多边形边上
139     // 对于狭义多边形，回转数为 0 表示点在多边形外，否则点在多边形内
140     pair<bool,int> winding(const point<T> &a) const
141     {
142         int cnt=0;
143         for (size_t i=0;i<p.size();i++)
144         {
145             const point<T> u=p[i],v=p[nxt(i)];
146             if (abs((a-u)^(a-v))<=eps && (a-u)*(a-v)<=eps)
                    return {true,0};
147             if (abs(u.y-v.y)<=eps) continue;
148             const Line uv={u,v-u};
149             if (u.y<v.y-eps && uv.toleft(a)<=0) continue;
150             if (u.y>v.y+eps && uv.toleft(a)>=0) continue;
151             if (u.y<a.y-eps && v.y>=a.y-eps) cnt++;
152             if (u.y>=a.y-eps && v.y<a.y-eps) cnt--;
153         }
154         return {false,cnt};
155     }
156
157     // 多边形面积的两倍
158     // 可用于判断点的存储顺序是顺时针或逆时针
159     T area() const
160     {
161         T sum=0;
162         for (size_t i=0;i<p.size();i++) sum+=p[i]^p[nxt(i)];
163         return sum;
164     }
165
166     // 多边形的周长
167     long double circ() const
168     {
169         long double sum=0;
170         for (size_t i=0;i<p.size();i++) sum+=p[i].dis(p[nxt(
                i)]);
171         return sum;
172     }
173 };
```

```
174
175 using Polygon=polygon<point_t>;
176
177 //凸多边形
178 template<typename T> struct convex: polygon<T>
179 {
180     // 闵可夫斯基和
181     convex operator+(const convex &c) const
182     {
183         const auto &p=this->p;
184         vector<Segment> e1(p.size()),e2(c.p.size()),edge(p.
                size()+c.p.size());
185         vector<point<T>> res; res.reserve(p.size()+c.p.size
                ());
186         const auto cmp=[](const Segment &u,const Segment &v)
                 {return argcmp()(u.b-u.a,v.b-v.a);};
187         for (size_t i=0;i<p.size();i++) e1[i]={p[i],p[this->
                nxt(i)]};
188         for (size_t i=0;i<c.p.size();i++) e2[i]={c.p[i],c.p[
                c.nxt(i)]};
189         rotate(e1.begin(),min_element(e1.begin(),e1.end(),
                cmp),e1.end());
190         rotate(e2.begin(),min_element(e2.begin(),e2.end(),
                cmp),e2.end());
191         merge(e1.begin(),e1.end(),e2.begin(),e2.end(),edge.
                begin(),cmp);
192         const auto check=[](const vector<point<T>> &res,
                const point<T> &u)
193         {
194             const auto back1=res.back(),back2=*prev(res.end
                    (),2);
195             return (back1-back2).toleft(u-back1)==0 && (
                    back1-back2)*(u-back1)>=-eps;
196         };
197         auto u=e1[0].a+e2[0].a;
198         for (const auto &v:edge)
199         {
200             while (res.size()>1 && check(res,u)) res.
                    pop_back();
201             res.push_back(u);
202             u=u+v.b-v.a;
203         }
```

```
204         if (res.size()>1 && check(res,res[0])) res.pop_back
                ();
205         return {res};
206     }
207
208     // 旋转卡壳
209     // func 为更新答案的函数，可以根据题目调整位置
210     template<typename F> void rotcaliper(const F &func)
            const
211     {
212         const auto &p=this->p;
213         const auto area=[](const point<T> &u,const point<T>
                &v,const point<T> &w){return (w-u)^(w-v);};
214         for (size_t i=0,j=1;i<p.size();i++)
215         {
216             const auto nxti=this->nxt(i);
217             //func(p[i],p[nxti],p[j]);
218             while (area(p[this->nxt(j)],p[i],p[nxti])>=area(
                    p[j],p[i],p[nxti]))
219             {
220                 j=this->nxt(j);
221                 //func(p[i],p[nxti],p[j]);
222             }
223             func(p[i],p[nxti],p[j]);
224         }
225     }
226
227     // 凸多边形的直径的平方
228     T diameter2() const
229     {
230         const auto &p=this->p;
231         if (p.size()==1) return 0;
232         if (p.size()==2) return p[0].dis2(p[1]);
233         T ans=0;
234         auto func=[&](const point<T> &u,const point<T> &v,
                const point<T> &w){ans=max({ans,w.dis2(u),w.dis2(
                v)});};
235         rotcaliper(func);
236         return ans;
237     }
238     // 凸包宽度
239         T get_width() const
```

```
240     {
241         T ans=INT_MAX;
242         auto func=[&](const point<T> &u,const point<T> &v,
                const point<T> &w){ans=min({ans,Line{u,v-u}.dis(w
                )});};
243         rotcaliper(func);
244         return ans;
245     }
246     // 最大三角形 n^2
247     T max_triangle() const
248     {
249         const auto &p=this->p;
250         if (p.size()==1) return 0;
251         if (p.size()==2) return 0;
252         T ans=0;
253         auto func=[&](const point<T> &u,const point<T> &v,
                const point<T> &w){ans=max({ans,(w-u)^(w-v)});};
254         rotcaliper(func);
255         return ans;
256     }
257
258     // 判断点是否在凸多边形内
259     // 复杂度 O(logn)
260     // -1 点在多边形边上 | 0 点在多边形外 | 1 点在多边形内
261     int is_in(const point<T> &a) const
262     {
263         const auto &p=this->p;
264         if (p.size()==1) return a==p[0]?-1:0;
265         if (p.size()==2) return segment<T>{p[0],p[1]}.is_on(
                a)?-1:0;
266         if (a==p[0]) return -1;
267         if ((p[1]-p[0]).toleft(a-p[0])==-1 || (p.back()-p
                [0]).toleft(a-p[0])==1) return 0;
268         const auto cmp=[&](const Point &u,const Point &v){
                return (u-p[0]).toleft(v-p[0])==1;};
269         const size_t i=lower_bound(p.begin()+1,p.end(),a,cmp
                )-p.begin();
270         if (i==1) return segment<T>{p[0],p[i]}.is_on(a)
                ?-1:0;
271         if (i==p.size()-1 && segment<T>{p[0],p[i]}.is_on(a))
                 return -1;
272         if (segment<T>{p[i-1],p[i]}.is_on(a)) return -1;
```

```
273         return (p[i]-p[i-1]).toleft(a-p[i-1])>0;
274     }
275
276     // 凸多边形关于某一方向的极点
277     // 复杂度 O(logn)
278     // 参考资料: https://codeforces.com/blog/entry/48868
279     template<typename F> size_t extreme(const F &dir) const
280     {
281         const auto &p=this->p;
282         const auto check=[&](const size_t i){return dir(p[i
                ]).toleft(p[this->nxt(i)]-p[i])>=0;};
283         const auto dir0=dir(p[0]); const auto check0=check
                (0);
284         if (!check0 && check(p.size()-1)) return 0;
285         const auto cmp=[&](const Point &v)
286         {
287             const size_t vi=&v-p.data();
288             if (vi==0) return 1;
289             const auto checkv=check(vi);
290             const auto t=dir0.toleft(v-p[0]);
291             if (vi==1 && checkv==check0 && t==0) return 1;
292             return checkv^(checkv==check0 && t<=0);
293         };
294         return partition_point(p.begin(),p.end(),cmp)-p.
                begin();
295     }
296
297     // 过凸多边形外一点求凸多边形的切线, 返回切点下标
298     // 复杂度 O(logn)
299     // 必须保证点在多边形外
300     pair<size_t,size_t> tangent(const point<T> &a) const
301     {
302         const size_t i=extreme([&](const point<T> &u){return
                u-a;});
303         const size_t j=extreme([&](const point<T> &u){return
                a-u;});
304         return {i,j};
305     }
306
307     // 求平行于给定直线的凸多边形的切线, 返回切点下标
308     // 复杂度 O(logn)
309     pair<size_t,size_t> tangent(const line<T> &a) const
```

```
310     {
311         const size_t i=extreme([&](...){return a.v;});
312         const size_t j=extreme([&](...){return -a.v;});
313         return {i,j};
314     }
315 };
316 
317 using Convex=convex<point_t>;
318 
319 // 点集的凸包
320 // Andrew 算法, 复杂度 O(nlogn)
321 Convex convexhull(vector<Point> p)
322 {
323     vector<Point> st;
324     sort(p.begin(),p.end());
325     const auto check=[](const vector<Point> &st,const Point
        &u)
326     {
327         const auto back1=st.back(),back2=*prev(st.end(),2);
328         return (back1-back2).toleft(u-back2)<=0;
329     };
330     for (const Point &u:p)
331     {
332         while (st.size()>1 && check(st,u)) st.pop_back();
333         st.push_back(u);
334     }
335     size_t k=st.size();
336     p.pop_back(); reverse(p.begin(),p.end());
337     for (const Point &u:p)
338     {
339         while (st.size()>k && check(st,u)) st.pop_back();
340         st.push_back(u);
341     }
342     st.pop_back();
343     return Convex{st};
344 }
345 //最小面积矩形
346 double rotcaliper(Polygon &a)
347 {
348     double ans=LONG_LONG_MAX;
349     Polygon ansp;
350     for (int i=0,j=1,l=-1,r=-1;i<(int)a.p.size();i++)
```

```
351     {
352         while (((a.p[a.nxt(j)]-a.p[i])^(a.p[a.nxt(j)]-a.p[a.
                nxt(i)]))
353         >((a.p[j]-a.p[i])^(a.p[j]-a.p[a.nxt(i)]))) j=a.nxt(j
                );
354         if (l==-1) l=i,r=j;
355         Point v={a.p[a.nxt(i)]-a.p[i]};
356         v=Point{-v.y,v.x};
357         while (v.toleft(a.p[a.nxt(l)]-a.p[l])<=0) l=a.nxt(l)
                ;
358         while (v.toleft(a.p[a.nxt(r)]-a.p[r])>=0) r=a.nxt(r)
                ;
359         Line li={a.p[i],a.p[a.nxt(i)]-a.p[i]},lj={a.p[j],a.p
                [i]-a.p[a.nxt(i)]};
360         Line l1={a.p[l],v},lr={a.p[r],v};
361         vector<Point> t={li.inter(l1),l1.inter(lj),lj.inter(
                lr),lr.inter(li)};
362         Polygon pl={t};
363         double s=pl.area();
364         if (s<ans) ans=s,ansp=pl;
365     }
366     return ans;
367 }
368 // 圆
369 struct Circle
370 {
371     Point c;
372     long double r;
373 
374     bool operator==(const Circle &a) const {return c==a.c &&
             abs(r-a.r)<=eps;}
375     long double circ() const {return 2*PI*r;}  // 周长
376     long double area() const {return PI*r*r;}  // 面积
377 
378     // 点与圆的关系
379     // -1 圆上 | 0 圆外 | 1 圆内
380     int is_in(const Point &p) const {const long double d=p.
            dis(c); return abs(d-r)<=eps?-1:d<r-eps;}
381 
382     // 直线与圆关系
383     // 0 相离 | 1 相切 | 2 相交
384     int relation(const Line &l) const
```

```
385     {
386         const long double d=l.dis(c);
387         if (d>r+eps) return 0;
388         if (abs(d-r)<=eps) return 1;
389         return 2;
390     }
391 
392     // 圆与圆关系
393     // -1 相同 | 0 相离 | 1 外切 | 2 相交 | 3 内切 | 4 内含
394     int relation(const Circle &a) const
395     {
396         if (*this==a) return -1;
397         const long double d=c.dis(a.c);
398         if (d>r+a.r+eps) return 0;
399         if (abs(d-r-a.r)<=eps) return 1;
400         if (abs(d-abs(r-a.r))<=eps) return 3;
401         if (d<abs(r-a.r)-eps) return 4;
402         return 2;
403     }
404 
405     // 直线与圆的交点
406     vector<Point> inter(const Line &l) const
407     {
408         const long double d=l.dis(c);
409         const Point p=l.proj(c);
410         const int t=relation(l);
411         if (t==0) return vector<Point>();
412         if (t==1) return vector<Point>{p};
413         const long double k=sqrt(r*r-d*d);
414         return vector<Point>{p-(l.v/l.v.len())*k,p+(l.v/l.v.
                len())*k};
415     }
416 
417     // 圆与圆交点
418     vector<Point> inter(const Circle &a) const
419     {
420         const long double d=c.dis(a.c);
421         const int t=relation(a);
422         if (t==-1 || t==0 || t==4) return vector<Point>();
423         Point e=a.c-c; e=e/e.len()*r;
424         if (t==1 || t==3)
425         {
```

```
426             if (r*r+d*d-a.r*a.r>=-eps) return vector<Point>{
                    c+e};
427             return vector<Point>{c-e};
428         }
429         const long double costh=(r*r+d*d-a.r*a.r)/(2*r*d),
                sinth=sqrt(1-costh*costh);
430         return vector<Point>{c+e.rot(costh,-sinth),c+e.rot(
                costh,sinth)};
431     }
432
433     // 圆与圆交面积
434     long double inter_area(const Circle &a) const
435     {
436         const long double d=c.dis(a.c);
437         const int t=relation(a);
438         if (t==-1) return area();
439         if (t<2) return 0;
440         if (t>2) return min(area(),a.area());
441         const long double costh1=(r*r+d*d-a.r*a.r)/(2*r*d),
                costh2=(a.r*a.r+d*d-r*r)/(2*a.r*d);
442         const long double sinth1=sqrt(1-costh1*costh1),
                sinth2=sqrt(1-costh2*costh2);
443         const long double th1=acos(costh1),th2=acos(costh2);
444         return r*r*(th1-costh1*sinth1)+a.r*a.r*(th2-costh2*
                sinth2);
445     }
446
447     // 过圆外一点圆的切线
448     vector<Line> tangent(const Point &a) const
449     {
450         const int t=is_in(a);
451         if (t==1) return vector<Line>();
452         if (t==-1)
453         {
454             const Point v={-(a-c).y,(a-c).x};
455             return vector<Line>{{a,v}};
456         }
457         Point e=a-c; e=e/e.len()*r;
458         const long double costh=r/c.dis(a),sinth=sqrt(1-
                costh*costh);
459         const Point t1=c+e.rot(costh,-sinth),t2=c+e.rot(
                costh,sinth);
```

```
460         return vector<Line>{{a,t1-a},{a,t2-a}};
461     }
462
463     // 两圆的公切线
464     vector<Line> tangent(const Circle &a) const
465     {
466         const int t=relation(a);
467         vector<Line> lines;
468         if (t==-1 || t==4) return lines;
469         if (t==1 || t==3)
470         {
471             const Point p=inter(a)[0],v={-(a.c-c).y,(a.c-c).
                x};
472             lines.push_back({p,v});
473         }
474         const long double d=c.dis(a.c);
475         const Point e=(a.c-c)/(a.c-c).len();
476         if (t<=2)
477         {
478             const long double costh=(r-a.r)/d,sinth=sqrt(1-
                costh*costh);
479             const Point d1=e.rot(costh,-sinth),d2=e.rot(
                costh,sinth);
480             const Point u1=c+d1*r,u2=c+d2*r,v1=a.c+d1*a.r,v2
                =a.c+d2*a.r;
481             lines.push_back({u1,v1-u1}); lines.push_back({u2
                ,v2-u2});
482         }
483         if (t==0)
484         {
485             const long double costh=(r+a.r)/d,sinth=sqrt(1-
                costh*costh);
486             const Point d1=e.rot(costh,-sinth),d2=e.rot(
                costh,sinth);
487             const Point u1=c+d1*r,u2=c+d2*r,v1=a.c-d1*a.r,v2
                =a.c-d2*a.r;
488             lines.push_back({u1,v1-u1}); lines.push_back({u2
                ,v2-u2});
489         }
490         return lines;
491     }
492 };
```

```
493
494 // 圆与多边形面积交
495 long double area_inter(const Circle &circ,const Polygon &
        poly)
496 {
497     const auto cal=[](const Circle &circ,const Point &a,
            const Point &b)
498     {
499         if ((a-circ.c).toleft(b-circ.c)==0) return 0.0l;
500         const auto ina=circ.is_in(a),inb=circ.is_in(b);
501         const Line ab={a,b-a};
502         if (ina && inb) return ((a-circ.c)^(b-circ.c))/2;
503         if (ina && !inb)
504         {
505             const auto t=circ.inter(ab);
506             const Point p=t.size()==1?t[0]:t[1];
507             const long double ans=((a-circ.c)^(p-circ.c))/2;
508             const long double th=(p-circ.c).ang(b-circ.c);
509             const long double d=circ.r*circ.r*th/2;
510             if ((a-circ.c).toleft(b-circ.c)==1) return ans+d
                    ;
511             return ans-d;
512         }
513         if (!ina && inb)
514         {
515             const Point p=circ.inter(ab)[0];
516             const long double ans=((p-circ.c)^(b-circ.c))/2;
517             const long double th=(a-circ.c).ang(p-circ.c);
518             const long double d=circ.r*circ.r*th/2;
519             if ((a-circ.c).toleft(b-circ.c)==1) return ans+d
                    ;
520             return ans-d;
521         }
522         const auto p=circ.inter(ab);
523         if (p.size()==2 && Segment{a,b}.dis(circ.c)<=circ.r+
                eps)
524         {
525             const long double ans=((p[0]-circ.c)^(p[1]-circ.
                    c))/2;
526             const long double th1=(a-circ.c).ang(p[0]-circ.c
                    ),th2=(b-circ.c).ang(p[1]-circ.c);
527             const long double d1=circ.r*circ.r*th1/2,d2=circ
```

```
                .r*circ.r*th2/2;
528             if ((a-circ.c).toleft(b-circ.c)==1) return ans+
                    d1+d2;
529             return ans-d1-d2;
530         }
531         const long double th=(a-circ.c).ang(b-circ.c);
532         if ((a-circ.c).toleft(b-circ.c)==1) return circ.r*
                circ.r*th/2;
533         return -circ.r*circ.r*th/2;
534     };
535
536     long double ans=0;
537     for (size_t i=0;i<poly.p.size();i++)
538     {
539         const Point a=poly.p[i],b=poly.p[poly.nxt(i)];
540         ans+=cal(circ,a,b);
541     }
542     return ans;
543 }
544
545
546 // 半平面交
547 // 排序增量法, 复杂度 O(nlogn)
548 // 输入与返回值都是用直线表示的半平面集合
549 vector<Line> halfinter(vector<Line> l, const point_t lim=1e9
    )
550 {
551     const auto check=[](const Line &a,const Line &b,const
        Line &c){return a.toleft(b.inter(c))<0;};
552     // 无精度误差的方法, 但注意取值范围会扩大到三次方
553     /*const auto check=[](const Line &a,const Line &b,const
        Line &c)
554     {
555         const Point p=a.v*(b.v^c.v),q=b.p*(b.v^c.v)+b.v*(c.v
            ^(b.p-c.p))-a.p*(b.v^c.v);
556         return p.toleft(q)<0;
557     };*/
558     l.push_back({{-lim,0},{0,-1}}); l.push_back({{0,-lim
        },{1,0}});
559     l.push_back({{lim,0},{0,1}}); l.push_back({{0,lim
        },{-1,0}});
560     sort(l.begin(),l.end());
```

```
561     deque<Line> q;
562     for (size_t i=0;i<l.size();i++)
563     {
564         if (i>0 && l[i-1].v.toleft(l[i].v)==0 && l[i-1].v*l[
                i].v>eps) continue;
565         while (q.size()>1 && check(l[i],q.back(),q[q.size()
                -2])) q.pop_back();
566         while (q.size()>1 && check(l[i],q[0],q[1])) q.
                pop_front();
567         if (!q.empty() && q.back().v.toleft(l[i].v)<=0)
                return vector<Line>();
568         q.push_back(l[i]);
569     }
570     while (q.size()>1 && check(q[0],q.back(),q[q.size()-2]))
            q.pop_back();
571     while (q.size()>1 && check(q.back(),q[0],q[1])) q.
            pop_front();
572     return vector<Line>(q.begin(),q.end());
573 }
574
575 // 点集形成的最小最大三角形
576 // 极角序扫描线，复杂度 O(n^2logn)
577 // 最大三角形问题可以使用凸包与旋转卡壳做到 O(n^2)
578 pair<point_t,point_t> minmax_triangle(const vector<Point> &
        vec)
579 {
580     if (vec.size()<=2) return {0,0};
581     vector<pair<int,int>> evt;
582     evt.reserve(vec.size()*vec.size());
583     point_t maxans=0,minans=numeric_limits<point_t>::max();
584     for (size_t i=0;i<vec.size();i++)
585     {
586         for (size_t j=0;j<vec.size();j++)
587         {
588             if (i==j) continue;
589             if (vec[i]==vec[j]) minans=0;
590             else evt.push_back({i,j});
591         }
592     }
593     sort(evt.begin(),evt.end(),[&](const pair<int,int> &u,
            const pair<int,int> &v)
594     {
```

```
595         const Point du=vec[u.second]-vec[u.first],dv=vec[v.
                second]-vec[v.first];
596         return argcmp()({du.y,-du.x},{dv.y,-dv.x});
597     });
598     vector<size_t> vx(vec.size()),pos(vec.size());
599     for (size_t i=0;i<vec.size();i++) vx[i]=i;
600     sort(vx.begin(),vx.end(),[&](int x,int y){return vec[x]<
            vec[y];});
601     for (size_t i=0;i<vx.size();i++) pos[vx[i]]=i;
602     for (auto [u,v]:evt)
603     {
604         const size_t i=pos[u],j=pos[v];
605         const size_t l=min(i,j),r=max(i,j);
606         const Point vecu=vec[u],vecv=vec[v];
607         if (l>0) minans=min(minans,abs((vec[vx[l-1]]-vecu)^(
                vec[vx[l-1]]-vecv)));
608         if (r<vx.size()-1) minans=min(minans,abs((vec[vx[r
                +1]]-vecu)^(vec[vx[r+1]]-vecv)));
609         maxans=max({maxans,abs((vec[vx[0]]-vecu)^(vec[vx
                [0]]-vecv)),abs((vec[vx.back()]-vecu)^(vec[vx.
                back()]-vecv))});
610         if (i<j) swap(vx[i],vx[j]),pos[u]=j,pos[v]=i;
611     }
612     return {minans,maxans};
613 }
614
615 // 判断多条线段是否有交点
616 // 扫描线，复杂度 O(nlogn)
617 bool segs_inter(const vector<Segment> &segs)
618 {
619     if (segs.empty()) return false;
620     using seq_t=tuple<point_t,int,Segment>;
621     const auto seqcmp=[](const seq_t &u, const seq_t &v)
622     {
623         const auto [u0,u1,u2]=u;
624         const auto [v0,v1,v2]=v;
625         if (abs(u0-v0)<=eps) return make_pair(u1,u2)<
                make_pair(v1,v2);
626         return u0<v0-eps;
627     };
628     vector<seq_t> seq;
629     for (auto seg:segs)
```

```
630     {
631         if (seg.a.x>seg.b.x+eps) swap(seg.a,seg.b);
632         seq.push_back({seg.a.x,0,seg});
633         seq.push_back({seg.b.x,1,seg});
634     }
635     sort(seq.begin(),seq.end(),seqcmp);
636     point_t x_now;
637     auto cmp=[&](const Segment &u, const Segment &v)
638     {
639         if (abs(u.a.x-u.b.x)<=eps || abs(v.a.x-v.b.x)<=eps)
                return u.a.y<v.a.y-eps;
640         return ((x_now-u.a.x)*(u.b.y-u.a.y)+u.a.y*(u.b.x-u.a
                .x))*(v.b.x-v.a.x)<((x_now-v.a.x)*(v.b.y-v.a.y)+v
                .a.y*(v.b.x-v.a.x))*(u.b.x-u.a.x)-eps;
641     };
642     multiset<Segment,decltype(cmp)> s{cmp};
643     for (const auto [x,o,seg]:seq)
644     {
645         x_now=x;
646         const auto it=s.lower_bound(seg);
647         if (o==0)
648         {
649             if (it!=s.end() && seg.is_inter(*it)) return
                    true;
650             if (it!=s.begin() && seg.is_inter(*prev(it)))
                    return true;
651             s.insert(seg);
652         }
653         else
654         {
655             if (next(it)!=s.end() && it!=s.begin() && (*prev
                    (it)).is_inter(*next(it))) return true;
656             s.erase(it);
657         }
658     }
659     return false;
660 }
661
662 // 多边形面积并
663 // 轮廓积分，复杂度约 O边数(^2)
664 // ans[i] 表示被至少覆盖了 i+1 次的区域的面积
665 vector<long double> area_union(const vector<Polygon> &polys)
```

```
666 {
667     const size_t siz=polys.size();
668     vector<vector<pair<Point,Point>>> segs(siz);
669     const auto check=[](const Point &u,const Segment &e){
            return !((u<e.a && u<e.b) || (u>e.a && u>e.b));};
670     auto cut_edge=[&](const Segment &e,const size_t i)
671     {
672         const Line le{e.a,e.b-e.a};
673         const auto cmp=[&](const Point &u,const Point &v){
                return e.a<e.b?u<v:u>v;};
674         map<Point,int,decltype(cmp)> cnt(cmp);
675         cnt[e.a]; cnt[e.b];
676         for (size_t j=0;j<polys.size();j++)
677         {
678             if (i==j) continue;
679             const auto &pj=polys[j];
680             for (size_t k=0;k<pj.p.size();k++)
681             {
682                 const Segment s={pj.p[k],pj.p[pj.nxt(k)]};
683                 if (le.toleft(s.a)==0 && le.toleft(s.b)==0)
                        cnt[s.a],cnt[s.b];
684                 else if (s.is_inter(le))
685                 {
686                     const Line ls{s.a,s.b-s.a};
687                     const Point u=le.inter(ls);
688                     if (le.toleft(s.a)<0 && le.toleft(s.b)
                            >=0) cnt[u]--;
689                     else if (le.toleft(s.a)>=0 && le.toleft(
                            s.b)<0) cnt[u]++;
690                 }
691             }
692         }
693         int sum=cnt.begin()->second;
694         for (auto it=cnt.begin();next(it)!=cnt.end();it++)
695         {
696             const Point u=it->first,v=next(it)->first;
697             if (check(u,e) && check(v,e)) segs[sum].
                    push_back({u,v});
698             sum+=next(it)->second;
699         }
700     };
701     for (size_t i=0;i<polys.size();i++)
```

```
    {
        const auto &pi=polys[i];
        for (size_t k=0;k<pi.p.size();k++)
        {
            const Segment ei={pi.p[k],pi.p[pi.nxt(k)]};
            cut_edge(ei,i);
        }
    }
    vector<long double> ans(siz);
    for (size_t i=0;i<siz;i++)
    {
        long double sum=0;
        sort(segs[i].begin(),segs[i].end());
        int cnt=0;
        for (size_t j=0;j<segs[i].size();j++)
        {
            if (j>0 && segs[i][j]==segs[i][j-1]) segs[i+(++
                cnt)].push_back(segs[i][j]);
            else cnt=0,sum+=segs[i][j].first^segs[i][j].
                second;
        }
        ans[i]=sum/2;
    }
    return ans;
}

// 圆面积并
// 轮廓积分，复杂度约 O(n^2)
// ans[i] 表示被至少覆盖了 i+1 次的区域的面积
vector<long double> area_union(const vector<Circle> &circs)
{
    const size_t siz=circs.size();
    using arc_t=tuple<Point,long double,long double,long
        double>;
    vector<vector<arc_t>> arcs(siz);

    auto cut_circ=[&](const Circle &ci,const size_t i)
    {
        auto cmp=[](const long double x,const long double y)
            {return x<y-eps;};
        map<long double,int,decltype(cmp)> cnt{cmp}; cnt[-PI
            ]; cnt[PI];
```

```
739         int init=0;
740         for (size_t j=0;j<circs.size();j++)
741         {
742             if (i==j) continue;
743             const Circle &cj=circs[j];
744             if (ci.r<cj.r-eps && ci.relation(cj)>=3) init++;
745             const auto inters=ci.inter(cj);
746             if (inters.size()==1) cnt[atan2l((inters[0]-ci.c
                    ).y,(inters[0]-ci.c).x)];
747             if (inters.size()==2)
748             {
749                 const Point dl=inters[0]-ci.c,dr=inters[1]-
                        ci.c;
750                 long double argl=atan2l(dl.y,dl.x),argr=
                        atan2l(dr.y,dr.x);
751                 if (abs(argl+PI)<=eps) argl=PI;
752                 if (abs(argr+PI)<=eps) argr=PI;
753                 if (argl>argr+eps) cnt[argl]++,cnt[PI]--,cnt
                        [-PI]++,cnt[argr]--,init++;
754                 else cnt[argl]++,cnt[argr]--;
755             }
756         }
757         if (cnt.empty()) arcs[init].push_back({ci.c,ci.r,-PI
                ,PI});
758         else
759         {
760             int sum=init;
761             for (auto it=cnt.begin();next(it)!=cnt.end();it
                    ++)
762             {
763                 arcs[sum].push_back({ci.c,ci.r,it->first,
                        next(it)->first});
764                 sum+=next(it)->second;
765             }
766         }
767     };
768
769     for (size_t i=0;i<circs.size();i++)
770     {
771         const auto &ci=circs[i];
772         cut_circ(ci,i);
773     }
```

```
774     vector<long double> ans(siz);
775     const auto oint=[](const arc_t &arc)
776     {
777         const auto [cc,cr,l,r]=arc;
778         if (abs(r-l-PI-PI)<=eps) return 2.0l*PI*cr*cr;
779         return cr*cr*(r-l)+cc.x*cr*(sin(r)-sin(l))-cc.y*cr*(
                cos(r)-cos(l));
780     };
781     for (size_t i=0;i<siz;i++)
782     {
783         long double sum=0;
784         sort(arcs[i].begin(),arcs[i].end());
785         int cnt=0;
786         for (size_t j=0;j<arcs[i].size();j++)
787         {
788             if (j>0 && arcs[i][j]==arcs[i][j-1]) arcs[i+(++
                    cnt)].push_back(arcs[i][j]);
789             else cnt=0,sum+=oint(arcs[i][j]);
790         }
791         ans[i]=sum/2;
792     }
793     return ans;
794 }
```