```cpp
#include <bits/stdc++.h>
#define EPS 1e-10
#define equals(a, b) (fabs((a) - (b)) < EPS)
typedef long long ll;
#define PI 3.141592653589793238
using namespace std;

// COUNTER CLOCKWISE
static const int CCW_COUNTER_CLOCKWISE = 1;
static const int CCW_CLOCKWISE = -1;
static const int CCW_ONLINE_BACK = 2;
static const int CCW_ONLINE_FRONT = -2;
static const int CCW_ON_SEGMENT = 0;

//Intercsect Circle & Circle
static const int ICC_SEPERATE = 4;
static const int ICC_CIRCUMSCRIBE = 3;
static const int ICC_INTERSECT = 2;
static const int ICC_INSCRIBE = 1;
static const int ICC_CONTAIN = 0;

struct Point{
    double x,y;
    Point(){}
    Point(double x,double y) :x(x),y(y){}
    Point operator+(Point p) {return Point(x+p.x,y+p.y);}
    Point operator-(Point p) {return Point(x-p.x,y-p.y);}
    Point operator*(double k){return Point(x*k,y*k);}
    Point operator/(double k){return Point(x/k,y/k);}
    double norm(){return x*x+y*y;}
    double abs(){return sqrt(norm());}

    bool operator < (const Point &p) const{
        return x!=p.x?x<p.x:y<p.y;
        //grid-point only
        //return !equals(x,p.x)?x<p.x:!equals(y,p.y)?y<p.y:0;
    }

    bool operator == (const Point &p) const{
        return fabs(x-p.x)<EPS && fabs(y-p.y)<EPS;
    }
};

struct EndPoint{
    Point p;
    int seg,st;
    EndPoint(){}
    EndPoint(Point p,int seg,int st):p(p),seg(seg),st(st){}
    bool operator<(const EndPoint &ep)const{
        if(p.y==ep.p.y) return st<ep.st;
        return p.y<ep.p.y;
    }
};

istream &operator >> (istream &is,Point &p){
    is>>p.x>>p.y;
    return is;
}

ostream &operator << (ostream &os,Point p){
```

```
61      os<<fixed<<setprecision(12)<<p.x<<" "<<p.y;
62      return os;
63  }
64
65  bool sort_x(Point a,Point b){
66      return a.x!=b.x?a.x<b.x:a.y<b.y;
67  }
68
69  bool sort_y(Point a,Point b){
70      return a.y!=b.y?a.y<b.y:a.x<b.x;
71  }
72
73  typedef Point Vector;
74  typedef vector<Point> Polygon;
75
76  istream &operator >> (istream &is,Polygon &p){
77      for(int i=0;i<(int)p.size();i++) is>>p[i];
78      return is;
79  }
80
81  struct Segment{
82      Point p1,p2;
83      Segment(){}
84      Segment(Point p1, Point p2):p1(p1),p2(p2){}
85  };
86  typedef Segment Line;
87
88  istream &operator >> (istream &is,Segment &s){
89      is>>s.p1>>s.p2;
90      return is;
91  }
92
93  struct Circle{
94      Point c;
95      double r;
96      Circle(){}
97      Circle(Point c,double r):c(c),r(r){}
98  };
99
100 istream &operator >> (istream &is,Circle &c){
101     is>>c.c>>c.r;
102     return is;
103 }
104
105 double norm(Vector a){
106     return a.x*a.x+a.y*a.y;
107 }
108 double abs(Vector a){
109     return sqrt(norm(a));
110 }
111 double dot(Vector a,Vector b){
112     return a.x*b.x+a.y*b.y;
113 }
114 double cross(Vector a,Vector b){
115     return a.x*b.y-a.y*b.x;
116 }
117
118 Point orth(Point p){return Point(-p.y,p.x);}
119
120 bool isOrthogonal(Vector a,Vector b){
```

```
    return equals(dot(a,b),0.0);
}

bool isOrthogonal(Point a1,Point a2,Point b1,Point b2){
    return isOrthogonal(a1-a2,b1-b2);
}

bool isOrthogonal(Segment s1,Segment s2){
    return equals(dot(s1.p2-s1.p1,s2.p2-s2.p1),0.0);
}

bool isParallel(Vector a,Vector b){
    return equals(cross(a,b),0.0);
}

bool isParallel(Point a1,Point a2,Point b1,Point b2){
  return isParallel(a1-a2,b1-b2);
}

bool isParallel(Segment s1,Segment s2){
  return equals(cross(s1.p2-s1.p1,s2.p2-s2.p1),0.0);
}

Point project(Segment s,Point p){
  Vector base=s.p2-s.p1;
  double r=dot(p-s.p1,base)/norm(base);
  return s.p1+base*r;
}

Point reflect(Segment s,Point p){
  return p+(project(s,p)-p)*2.0;
}

double arg(Vector p){
  return atan2(p.y,p.x);
}

Vector polar(double a,double r){
  return Point(cos(r)*a,sin(r)*a);
}

int ccw(Point p0,Point p1,Point p2);
bool intersectSS(Point p1,Point p2,Point p3,Point p4);
bool intersectSS(Segment s1,Segment s2);
bool intersectPS(Polygon p,Segment l);
int intersectCC(Circle c1,Circle c2);
bool intersectSC(Segment s,Circle c);
double getDistanceLP(Line l,Point p);
double getDistanceSP(Segment s,Point p);
double getDistanceSS(Segment s1,Segment s2);
Point getCrossPointSS(Segment s1,Segment s2);
Point getCrossPointLL(Line l1,Line l2);
Polygon getCrossPointCL(Circle c,Line l);
Polygon getCrossPointCC(Circle c1,Circle c2);
int contains(Polygon g,Point p);
Polygon andrewScan(Polygon s);
Polygon convex_hull(Polygon ps);
double diameter(Polygon s);
bool isConvex(Polygon p);
double area(Polygon s);
```

```
Polygon convexCut(Polygon p,Line l);
Line bisector(Point p1,Point p2);
Vector translate(Vector v,double theta);
vector<Line> corner(Line l1,Line l2);
vector< vector<int> >
segmentArrangement(vector<Segment> &ss, Polygon &ps);

int ccw(Point p0,Point p1,Point p2){
  Vector a = p1-p0;
  Vector b = p2-p0;
  if(cross(a,b) > EPS) return CCW_COUNTER_CLOCKWISE;
  if(cross(a,b) < -EPS) return CCW_CLOCKWISE;
  if(dot(a,b) < -EPS) return CCW_ONLINE_BACK;
  if(a.norm()<b.norm()) return CCW_ONLINE_FRONT;
  return CCW_ON_SEGMENT;
}

bool intersectSS(Point p1,Point p2,Point p3,Point p4){
  return (ccw(p1,p2,p3)*ccw(p1,p2,p4) <= 0 &&
          ccw(p3,p4,p1)*ccw(p3,p4,p2) <= 0 );
}

bool intersectSS(Segment s1,Segment s2){
  return intersectSS(s1.p1,s1.p2,s2.p1,s2.p2);
}

bool intersectPS(Polygon p,Segment l){
  int n=p.size();
  for(int i=0;i<n;i++)
    if(intersectSS(Segment(p[i],p[(i+1)%n]),l)) return 1;
  return 0;
}

int intersectCC(Circle c1,Circle c2){
  if(c1.r<c2.r) swap(c1,c2);
  double d=abs(c1.c-c2.c);
  double r=c1.r+c2.r;
  if(equals(d,r)) return ICC_CIRCUMSCRIBE;
  if(d>r) return ICC_SEPERATE;
  if(equals(d+c2.r,c1.r)) return ICC_INSCRIBE;
  if(d+c2.r<c1.r) return ICC_CONTAIN;
  return ICC_INTERSECT;
}

bool intersectSC(Segment s,Circle c){
  return getDistanceSP(s,c.c)<=c.r;
}

int intersectCS(Circle c,Segment s){
  if(norm(project(s,c.c)-c.c)-c.r*c.r>EPS) return 0;
  double d1=abs(c.c-s.p1),d2=abs(c.c-s.p2);
  if(d1<c.r+EPS&&d2<c.r+EPS) return 0;
  if((d1<c.r-EPS&&d2>c.r+EPS)||(d1>c.r+EPS&&d2<c.r-EPS)) return 1;
  Point h=project(s,c.c);
  if(dot(s.p1-h,s.p2-h)<0) return 2;
  return 0;
}

double getDistanceLP(Line l,Point p){
  return abs(cross(l.p2-l.p1,p-l.p1)/abs(l.p2-l.p1));
```

```
}

double getDistanceSP(Segment s,Point p){
  if(dot(s.p2-s.p1,p-s.p1) < 0.0 ) return abs(p-s.p1);
  if(dot(s.p1-s.p2,p-s.p2) < 0.0 ) return abs(p-s.p2);
  return getDistanceLP(s,p);
}

double getDistanceSS(Segment s1,Segment s2){
  if(intersectSS(s1,s2)) return 0.0;
  return min(min(getDistanceSP(s1,s2.p1),getDistanceSP(s1,s2.p2)),
             min(getDistanceSP(s2,s1.p1),getDistanceSP(s2,s1.p2)));
}

Point getCrossPointSS(Segment s1,Segment s2){
  for(int k=0;k<2;k++){
    if(getDistanceSP(s1,s2.p1)<EPS) return s2.p1;
    if(getDistanceSP(s1,s2.p2)<EPS) return s2.p2;
    swap(s1,s2);
  }
  Vector base=s2.p2-s2.p1;
  double d1=abs(cross(base,s1.p1-s2.p1));
  double d2=abs(cross(base,s1.p2-s2.p1));
  double t=d1/(d1+d2);
  return s1.p1+(s1.p2-s1.p1)*t;
}

Point getCrossPointLL(Line l1,Line l2){
  double a=cross(l1.p2-l1.p1,l2.p2-l2.p1);
  double b=cross(l1.p2-l1.p1,l1.p2-l2.p1);
  if(abs(a)<EPS&&abs(b)<EPS) return l2.p1;
  return l2.p1+(l2.p2-l2.p1)*(b/a);
}

Polygon getCrossPointCL(Circle c,Line l){
  Polygon ps;
  Point pr=project(l,c.c);
  Vector e=(l.p2-l.p1)/abs(l.p2-l.p1);
  if(equals(getDistanceLP(l,c.c),c.r)){
    ps.emplace_back(pr);
    return ps;
  }
  double base=sqrt(c.r*c.r-norm(pr-c.c));
  ps.emplace_back(pr+e*base);
  ps.emplace_back(pr-e*base);
  return ps;
}

Polygon getCrossPointCS(Circle c,Segment s){
  Line l(s);
  Polygon res=getCrossPointCL(c,l);
  if(intersectCS(c,s)==2) return res;
  if(res.size()>1u){
    if(dot(l.p1-res[0],l.p2-res[0])>0) swap(res[0],res[1]);
    res.pop_back();
  }
  return res;
}


```

```
301 Polygon getCrossPointCC(Circle c1,Circle c2){
302   Polygon p(2);
303   double d=abs(c1.c-c2.c);
304   double a=acos((c1.r*c1.r+d*d-c2.r*c2.r)/(2*c1.r*d));
305   double t=arg(c2.c-c1.c);
306   p[0]=c1.c+polar(c1.r,t+a);
307   p[1]=c1.c+polar(c1.r,t-a);
308   return p;
309 }
310
311 // IN:2 ON:1 OUT:0
312 int contains(Polygon g,Point p){
313   int n=g.size();
314   bool x=false;
315   for(int i=0;i<n;i++){
316     Point a=g[i]-p,b=g[(i+1)%n]-p;
317     if(fabs(cross(a,b)) < EPS && dot(a,b) < EPS) return 1;
318     if(a.y>b.y) swap(a,b);
319     if(a.y < EPS && EPS < b.y && cross(a,b) > EPS ) x = !x;
320   }
321   return (x?2:0);
322 }
323
324 Polygon andrewScan(Polygon s){
325   Polygon u,l;
326   if(s.size()<3) return s;
327   sort(s.begin(),s.end());
328   u.push_back(s[0]);
329   u.push_back(s[1]);
330   l.push_back(s[s.size()-1]);
331   l.push_back(s[s.size()-2]);
332   for(int i=2;i<(int)s.size();i++){
333     for(int n=u.size();n>=2&&ccw(u[n-2],u[n-1],s[i])!=CCW_CLOCKWISE;n--){
334       u.pop_back();
335     }
336     u.push_back(s[i]);
337   }
338   for(int i=s.size()-3;i>=0;i--){
339     for(int n=l.size();n>=2&&ccw(l[n-2],l[n-1],s[i])!=CCW_CLOCKWISE;n--){
340       l.pop_back();
341     }
342     l.push_back(s[i]);
343   }
344   reverse(l.begin(),l.end());
345   for(int i=u.size()-2;i>=1;i--) l.push_back(u[i]);
346   return l;
347 }
348
349 Polygon convex_hull(Polygon ps){
350   int n=ps.size();
351   sort(ps.begin(),ps.end(),sort_y);
352   int k=0;
353   Polygon qs(n*2);
354   for(int i=0;i<n;i++){
355     while(k>1&&cross(qs[k-1]-qs[k-2],ps[i]-qs[k-1])<0) k--;
356     qs[k++]=ps[i];
357   }
358   for(int i=n-2,t=k;i>=0;i--){
359     while(k>t&&cross(qs[k-1]-qs[k-2],ps[i]-qs[k-1])<0) k--;
360     qs[k++]=ps[i];
```

```
  }
  qs.resize(k-1);
  return qs;
}

double diameter(Polygon s){
  Polygon p=s;
  int n=p.size();
  if(n==2) return abs(p[0]-p[1]);
  int i=0,j=0;
  for(int k=0;k<n;k++){
    if(p[i]<p[k]) i=k;
    if(!(p[j]<p[k])) j=k;
  }
  double res=0;
  int si=i,sj=j;
  while(i!=sj||j!=si){
    res=max(res,abs(p[i]-p[j]));
    if(cross(p[(i+1)%n]-p[i],p[(j+1)%n]-p[j])<0.0){
      i=(i+1)%n;
    }else{
      j=(j+1)%n;
    }
  }
  return res;
}

bool isConvex(Polygon p){
  bool f=1;
  int n=p.size();
  for(int i=0;i<n;i++){
    int t=ccw(p[(i+n-1)%n],p[i],p[(i+1)%n]);
    f&=t!=CCW_CLOCKWISE;
  }
  return f;
}

double area(Polygon s){
  double res=0;
  for(int i=0;i<(int)s.size();i++){
    res+=cross(s[i],s[(i+1)%s.size()])/2.0;
  }
  return res;
}

double area(Circle c1,Circle c2){
  double d=abs(c1.c-c2.c);
  if(c1.r+c2.r<=d+EPS) return 0;
  if(d<=abs(c1.r-c2.r)){
    double r=min(c1.r,c2.r);
    return PI*r*r;
  }
  double rc=(d*d+c1.r*c1.r-c2.r*c2.r)/(2*d);
  double th=acos(rc/c1.r);
  double ph=acos((d-rc)/c2.r);
  return c1.r*c1.r*th+c2.r*c2.r*ph-d*c1.r*sin(th);
}

Polygon convexCut(Polygon p,Line l){
  Polygon q;
```

```
421   for(int i=0;i<(int)p.size();i++){
422     Point a=p[i],b=p[(i+1)%p.size()];
423     if(ccw(l.p1,l.p2,a)!=-1) q.push_back(a);
424     if(ccw(l.p1,l.p2,a)*ccw(l.p1,l.p2,b)<0)
425       q.push_back(getCrossPointLL(Line(a,b),l));
426   }
427   return q;
428 }
429
430 Line bisector(Point p1,Point p2){
431   Circle c1=Circle(p1,abs(p1-p2)),c2=Circle(p2,abs(p1-p2));
432   Polygon p=getCrossPointCC(c1,c2);
433   if(cross(p2-p1,p[0]-p1)>0) swap(p[0],p[1]);
434   return Line(p[0],p[1]);
435 }
436
437 Vector translate(Vector v,double theta){
438   Vector res;
439   res.x=cos(theta)*v.x-sin(theta)*v.y;
440   res.y=sin(theta)*v.x+cos(theta)*v.y;
441   return res;
442 }
443
444 vector<Line> corner(Line l1,Line l2){
445   vector<Line> res;
446   if(isParallel(l1,l2)){
447     double d=getDistanceLP(l1,l2.p1)/2.0;
448     Vector v1=l1.p2-l1.p1;
449     v1=v1/v1.abs()*d;
450     Point p=l2.p1+translate(v1,90.0*(PI/180.0));
451     double d1=getDistanceLP(l1,p);
452     double d2=getDistanceLP(l2,p);
453     if(abs(d1-d2)>d){
454       p=l2.p1+translate(v1,-90.0*(PI/180.0));
455     }
456     res.push_back(Line(p,p+v1));
457   }else{
458     Point p=getCrossPointLL(l1,l2);
459     Vector v1=l1.p2-l1.p1,v2=l2.p2-l2.p1;
460     v1=v1/v1.abs();
461     v2=v2/v2.abs();
462     res.push_back(Line(p,p+(v1+v2)));
463     res.push_back(Line(p,p+translate(v1+v2,90.0*(PI/180.0))));
464   }
465   return res;
466 }
467
468 Polygon tangent(Circle c1,Point p2){
469   Circle c2=Circle(p2,sqrt(norm(c1.c-p2)-c1.r*c1.r));
470   Polygon p=getCrossPointCC(c1,c2);
471   sort(p.begin(),p.end());
472   return p;
473 }
474
475 vector<Line> tangent(Circle c1,Circle c2){
476   vector<Line> ls;
477   if(c1.r<c2.r) swap(c1,c2);
478   double g=norm(c1.c-c2.c);
479   if(equals(g,0)) return ls;
480   Point u=(c2.c-c1.c)/sqrt(g);
```

```
  Point v=orth(u);
  for(int s=1;s>=-1;s-=2){
    double h=(c1.r+s*c2.r)/sqrt(g);
    if(equals(1-h*h,0)){
      ls.emplace_back(c1.c+u*c1.r,c1.c+(u+v)*c1.r);
    }else if(1-h*h>0){
      Point uu=u*h,vv=v*sqrt(1-h*h);
      ls.emplace_back(c1.c+(uu+vv)*c1.r,c2.c-(uu+vv)*c2.r*s);
      ls.emplace_back(c1.c+(uu-vv)*c1.r,c2.c-(uu-vv)*c2.r*s);
    }
  }

  return ls;
}

double closest_pair(Polygon &a,int l=0,int r=-1){
  if(r<0){
    r=a.size();
    sort(a.begin(),a.end(),sort_x);
  }
  if(r-l<=1) return abs(a[0]-a[1]);
  int m=(l+r)>>1;
  double x=a[m].x;
  double d=min(closest_pair(a,l,m),closest_pair(a,m,r));
  inplace_merge(a.begin()+l,a.begin()+m,a.begin()+r,sort_y);

  Polygon b;
  for(int i=l;i<r;i++){
    if(fabs(a[i].x-x)>=d) continue;
    for(int j=0;j<(int)b.size();j++){
      double dy=a[i].y-next(b.rbegin(),j)->y;
      if(dy>=d) break;
      d=min(d,abs(a[i]-*next(b.rbegin(),j)));
    }
    b.emplace_back(a[i]);
  }
  return d;
}

vector<vector<int> >
segmentArrangement(vector<Segment> &ss, Polygon &ps){
  int n=ss.size();
  for(int i=0;i<n;i++){
    ps.emplace_back(ss[i].p1);
    ps.emplace_back(ss[i].p2);
    for(int j=i+1;j<n;j++)
      if(intersectSS(ss[i],ss[j]))
        ps.emplace_back(getCrossPointSS(ss[i],ss[j]));
  }
  sort(ps.begin(),ps.end());
  ps.erase(unique(ps.begin(),ps.end()),ps.end());

  vector<vector<int> > G(ps.size());
  for(int i=0;i<n;i++){
    vector<pair<double,int> > ls;
    for(int j=0;j<(int)ps.size();j++)
      if(getDistanceSP(ss[i],ps[j])<EPS)
        ls.emplace_back(make_pair(norm(ss[i].p1-ps[j]),j));

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

```
    for(int j=0;j+1<(int)ls.size();j++){
      int a=ls[j].second,b=ls[j+1].second;
      G[a].emplace_back(b);
      G[b].emplace_back(a);
    }
  }
  for(auto &v:G){
    sort(v.begin(),v.end());
    v.erase(unique(v.begin(),v.end()),v.end());
  }
  return G;
}

int manhattanIntersection(vector<Segment> ss,const int INF){
  const int BTM = 0;
  const int LFT = 1;
  const int RGH = 2;
  const int TOP = 3;

  int n=ss.size();
  vector<EndPoint> ep;
  for(int i=0;i<n;i++){
    if(ss[i].p1.y==ss[i].p2.y){
      if(ss[i].p1.x>ss[i].p2.x) swap(ss[i].p1,ss[i].p2);
      ep.emplace_back(ss[i].p1,i,LFT);
      ep.emplace_back(ss[i].p2,i,RGH);
    }else{
      if(ss[i].p1.y>ss[i].p2.y) swap(ss[i].p1,ss[i].p2);
      ep.emplace_back(ss[i].p1,i,BTM);
      ep.emplace_back(ss[i].p2,i,TOP);
    }
  }
  sort(ep.begin(),ep.end());

  set<int> bt;
  bt.insert(INF);

  int cnt=0;
  for(int i=0;i<n*2;i++){
    if(ep[i].st==TOP){
      bt.erase(ep[i].p.x);
    }else if(ep[i].st==BTM){
      bt.emplace(ep[i].p.x);
    }else if(ep[i].st==LFT){
      auto b=bt.lower_bound(ss[ep[i].seg].p1.x);
      auto e=bt.upper_bound(ss[ep[i].seg].p2.x);
      cnt+=distance(b,e);
    }
  }

  return cnt;
}

double area(Polygon ps,Circle c){
  if(ps.size()<3u) return 0;
  function<double(Circle, Point, Point)> dfs=
    [&](Circle c,Point a,Point b){
      Vector va=c.c-a,vb=c.c-b;
      double f=cross(va,vb),res=0;
      if(equals(f,0.0)) return res;
```

```
601       if(max(abs(va),abs(vb))<c.r+EPS) return f;
602       Vector d(dot(va,vb),cross(va,vb));
603       if(getDistanceSP(Segment(a,b),c.c)>c.r-EPS)
604         return c.r*c.r*atan2(d.y,d.x);
605       auto u=getCrossPointCS(c,Segment(a,b));
606       if(u.empty()) return res;
607       if(u.size()>1u&&dot(u[1]-u[0],a-u[0])>0) swap(u[0],u[1]);
608       u.emplace(u.begin(),a);
609       u.emplace_back(b);
610       for(int i=1;i<(int)u.size();i++)
611         res+=dfs(c,u[i-1],u[i]);
612       return res;
613     };
614   double res=0;
615   for(int i=0;i<(int)ps.size();i++)
616     res+=dfs(c,ps[i],ps[(i+1)%ps.size()]);
617   return res/2;
618 }
```