```
/*
   Baricentro de un triangulo de puntos A,B,C
      G = (A+B+C) / 3

   Suma de Punto + Vector
      Punto A -------> Vector B
      Vector AB  = B - A
      A + AB = B

   Suma de Vectores
      A                    E
               C
          B         D

      AB + BC + CD + DE = AE
      AB = B - A
      AE = E - A

   Suma de vectores
      B-----------> C
      A
      |
      |
      A

      AB + BC = (B - A) + (C - B) = AC

   Producto Escalar (Point)
      u . v = u.x * v.x + u.y * v.y = |u||v|cos 0
         u . v = 0 => perpendiculares
      Conmutativo

   Producto Vectorial (cross)
      u x v = u.x*v.y - u.y*v.x = |u||v|sin 0

      NO conmutativo
         u x v = -v x u

      C----+
      A\   |   AC x BA > 0 Sentido Anti-Horario
      | \  |   AC x BA < 0 Sentido Horario
      |  \ |   AC x BA = 0 Colineales
      |   \|   AC x BA = 2 * Area Triangulo (con signo)
      A -->B

   Punto de Intersecci n
      A---P---------B

         D
        /
       /
      C

         P = A + AB K1 = C + CD K2
          K1 AB - K2 CD = C - A
          K1 AB - K2 CD = AC
   (K1 AB - K2 CD) x CD = AC x CD // x CD
K1 AB x CD - K2 CD x CD = AC x CD // CD x CD colineales
             k1 AB x CD = AC x CD
                     k1 = AC x CD / AB x CD

      ==> P = A + AB * (AC x CD / AB x CD)

   Proyeccion:
      P = point_intersection(A,B,X,X+orto(B-A));

   Circulo en base de 3 puntos A,B,C
      Sean: ABm y BCm los puntos medios de AB y BC
         El vectore R ABm es perpendicular a AB
         El vectore R BCm es perpendicular a BC

         R = Punto de interseccion entre:
                  R ---> orto(AB)
                  R ---> orto(BC)
      Circle Equation
         (x - h) ^ 2 + (y - k) ^ 2 = r ^ 2

   Teorema de  Pick:
      A = I + (.5B) -1

      A = Area
      I = # ptos en el interior del poligono
      B = # ptos en los bordes


   Matriz para girar
```

```

      |Cos0 -Sen0| |x|
      |Sen0  Cos0| |y|

      xx = xcos0 - ysen0
      yy = xsen0 + ycos0

   Trigonometr a
      sin A / A = sin B / B = sin C / C
      c^2 = a^2 + b^2 - 2ab cos 0 // 0 angulo entre a y b

   Centroid:
      The average of all the points.
      Properties:
         - This point minimizes the sum of squared Euclidean distances
            between itself and each point in the set.
*/


#define Vector Point
#define PP double
class Point{public:
   PP x,y;
   Point(){}
   Point(PP xx,PP yy){x = xx;y = yy;}
   double mod(){return hypot(x,y);}
   Point orto(){return Point(-y,x);}
   Point unit(){double k = mod();return Point(x/k,y/k);}
   void p(){cout << "::>␣" << x << "␣" << y << endl;}
};
Point operator + (const Point &A,const Point &B){return Point(A.x+B.x,A.y+B.y);}
Point operator - (const Point &A,const Point &B){return Point(A.x-B.x,A.y-B.y);}
Point operator * (const Point &A,const Point &B){return Point(A.x*B.x,A.y*B.y);}
Point operator / (const Point &A,double k){return Point(A.x/k,A.y/k);}
Point operator * (const Point &A,double k){return Point(A.x*k,A.y*k);}
bool  operator < (const Point &A,const Point &B){return pair<PP,PP>(A.x,A.y) < pair<PP,PP>(B.x,B.y);}
const double EPS = 0.0;
const double PI  = acos(-1);
const double oo = 1e18;
```

```

double cross(Point A,Point B){return A.x*B.y - A.y*B.x;}
double dot(Point A,Point B){return A.x*B.x + A.y*B.y;}
double dist(Point A,Point B){return hypot(A.x - B.x,A.y-B.y);}
double area2(Point A,Point B,Point C){return cross(B-A,C-A);}//For the triangle A,B,C  using A->B, A->C

bool pointInBox(Point P,Point A,Point B){//Point P inside box A,B
   return P.x >= min(A.x,B.x) and P.x <= max(A.x,B.x) and
          P.y >= min(A.y,B.y) and P.y <= max(A.y,B.y);
}
bool pointOverSegment(Point P,Point A,Point B){//p over AB
   return fabs(area2(A,B,P)) <= EPS and pointInBox(P,A,B);
}
//NO
double pseudoangulo(Point a,Point b){   //Da un pseudo angulo, solo para comparaciones
   if(a.x==b.x&&a.y==b.y)return 0.0;
   int dx=b.x-a.x,dy=b.y-a.y;
   double res=(double)dy/(abs(dx)+abs(dy));
   if(dx<0)res=2-res;
   else if(dy<0)res=4+res;
   return res*90.0;
}

// ==================== Lines and segments ============================

// UVA = {191,378,866,11665}
bool segmentsIntersect(Point A,Point B,Point C,Point D){//AB, CD
   double A1 = area2(C, D, A);
   double A2 = area2(C, D, B);
   double A3 = area2(A, B, C);
   double A4 = area2(A, B, D);

   if( ((A1 > 0 and A2 < 0) or (A1 < 0 and A2 > 0)) and
       ((A3 > 0 and A4 < 0) or (A3 < 0 and A4 > 0)))
           return true;

   else if(A1 == 0 and pointOverSegment(A, C, D)) return true;
   else if(A2 == 0 and pointOverSegment(B, C, D)) return true;
   else if(A3 == 0 and pointOverSegment(C, A, B)) return true;
   else if(A4 == 0 and pointOverSegment(D, A, B)) return true;
   else return false;
}
// UVA = {191,378,866,11665}
bool intersectionPoint(Point A,Point B,Point C,Point D){// AB y CD
   if(cross(B-A,D-C) == 0)//Parallels
      return pointOverSegment(C,A,B) or pointOverSegment(D,A,B);
   Point p = A + (B - A) * (cross(C - A, D - C) / cross(B - A, D - C));

   return pointInBox(p,A,B) and pointInBox(p,C,D);//If segments
   //return true; // If lines
}
// UVA = {10263}
double distToSegment(Point A,Point B,Point P){//dist from P to AB
   Point D = P + (B-A).orto();//perpendicular to AB
   Point p_int = A + (B - A) * (cross(P - A, D - P) / cross(B - A, D - P));
   if(pointInBox(p_int,A,B))
      return dist(P,p_int);
   else{//The answer is some Point
      double da = dist(A,P);
      double db = dist(B,P);
      p_int = da < db?A:B;
      return min(da,db);
   }
}
// UVA = {634,11665}
bool pointInPoly(vector<Point> pol,Point p){
   int cont=0,len=pol.size();
   Point act,sig;

   for(int i=0;i<len;i++){
      if (pointOverSegment(p,pol[i],pol[(i+1) %len]))
         return true;
      act = pol[i] - p;
      sig = pol[(i+1) %len] - p;
      if (act.y>sig.y)
         swap(act,sig);
      if (act.y<0 and sig.y>=0 and cross(sig,act)>=0)
         cont++;
   }
   return cont %2==1;
}

// ==================== Polygons ============================


// O(log n)
bool pointInConvexPoly(const vector <Point> &A, const Point &P){
   int n = A.size(), lo = 1, hi = A.size() - 1;

   if(area2(A[0], A[1], P) <= 0) return false;
   if(area2(A[n-1], A[0], P) <= 0) return false;

   while(hi - lo > 1){
      int mid = (lo + hi) / 2;

      if(area2(A[0], A[mid], P) > 0) lo = mid;
      else hi = mid;
   }

   return area2(A[lo], A[hi], P) > 0;
}

// LA = {4187}
double areaPolygon(const vector <Point> &P){
   int n = P.size();
   double A = 0;
   for(int i=1; i<=n-2; i++)
      A += area2(P[0], P[i], P[i+1]);
   return fabs(A/2);
}

// First Point != Last Point
// First Point bottom lefmost
// UVA = {UVA_10002}
void centerOfMass(vector<Point> ch){
   double x=0.0,y=0.0,tmp=0.0,area;
   for(int i=2;i<ch.size();i++){
      area = fabs(area2(ch[0],ch[i-1],ch[i]) / 2.0);
      x += area * (ch[0].x+ch[i-1].x+ch[i].x)/3.0;
      y += area * (ch[0].y+ch[i-1].y+ch[i].y)/3.0;
      tmp += area;
   }
   x/=tmp;
   y/=tmp;
}
```

```
//============= Algorithms ===================
// UVA = {218}
vector<Point> monotoneChainConvexHull(vector<Point> vc){
   int k=0;
   int n = vc.size();
   sort(vc.begin(),vc.end());

   Point CH[n];


   for(int i=0;i<n;i++){
      while(k>=2 and area2(CH[k-2],CH[k-1],vc[i])<=0.0)k--;//Cero si es colineal
      CH[k++]=vc[i];
   }
   int b=k+1;
   for(int i=n-2;i>=0;i--){
      while(k>=b and area2(CH[k-2],CH[k-1],vc[i])<=0.0) k--;//Cero si es colineal
      CH[k++]=vc[i];
   }

   assert(CH[0].x == CH[k-1].x and CH[0].y == CH[k-1].y);//first == last

   return vector<Point>(CH,CH+k);
}

//SPOJ = {TFOSS}
void rotatingCallipers(vector<Point> &P){//P is a convex hull
   int N = P.size();
   for(int i=0, j=2; i<N; i++){
      // P[j] debe ser el punto mas lejano a la linea P[i], P[(i+1)%n]:
      while(area2(P[i], P[(i+1)%N], P[(j+1)%N]) > area2(P[i], P[(i+1)%N],P[j])) j = (j+1)%N;

      // Antipodal Pairs: {(i, j),(i+1%N, j)}
      // the {(i, j+1%N),(i+1%N, j+1%N)} are found when j+1%n is evaluated
   }
}

// UVA = {10245}
int bb(vector<Point> &vc,int a,int b,double x){
   int mid;
   while(a<b){mid=(a+b)/2;
      if(vc[mid].x<x) a=mid+1;
      else            b=mid;
   }
   return b;
}
//Receive a range [start,end)
double closest_pair(int start,int end,vector<Point> &vc){
   if(start+1 == end) return oo;
   int mid=(start+end)/2;
   double delta=min(closest_pair(start,mid,vc),closest_pair(mid,end,vc));
   double lim_left  = vc[mid].x - delta;
   double lim_right = vc[mid].x + delta;

   int a=bb(vc, start,mid, lim_left );
   int b=bb(vc, mid  ,end, lim_right);

   for(int i=a;i<b;i++)
      for(int j=i+1;j<b;j++)
         delta= min(delta,dist(vc[i],vc[j]));
   return delta;
}
#include <set>
double closest_pair2(vector<Point> vc){
   sort(vc.begin(),vc.end());//sort by x
   set<Point> st;
   double res = oo;
   foreach(it,vc){
      set<Point>::iterator p = st.begin();
      while(p != st.end()){
         if(it->x - p->x >= res)//This point always be too far
            st.erase(p++);
         else{
            res = min(res,dist(*it,*p));
            p++;
         }
      }
      st.insert(*it);
   }
}
```

```
// SPOJ = {NKMARS}
void push(int x,int a,int d){
   if(tree[x] == 0) acum[x] = acum[2*x] + acum[2*x+1];
   else  acum[x] = yes[d+1] - yes[a];//yes[] are the y-coordinates compressed
}
void update(int x,int la,int ld,int a,int d,int add_val){
   if(a == la and d == ld){
      tree[x]+=add_val;
   }else{
      int lb = (la + ld) / 2;
      int lc = lb + 1;

      if(d <= lb)
         update(2*x,la,lb,a,d,add_val);
      else if(a >= lc)
         update(2*x+1,lc,ld,a,d,add_val);
      else{
         update(2*x,la,lb,a,lb,add_val);
         update(2*x+1,lc,ld,lc,d,add_val);
      }
}
push(x,la,ld);
}
void update(int a,int b,int add_val){
    update(1,0,y_segments - 1,a,b,add_val);
}
/*
   Lines is all the vertical lines ordered by X
   If the coordinates are too large, mp compress
   the coordinates
*/
long long overlapping_area(vector<line> lines){
   memset(tree,0,sizeof(tree));
   memset(acum,0,sizeof(acum));

   long long area = 0;
   long long pre_x = lines[0].x;
   foreach(ln,lines){
      w = ln->x - pre_x;

      if(w > 0)
         area += w * acum[1];

      a = mp[ln->a];b = mp[ln->b];

      update(a,b-1,ln->is_start?1:-1);//Add / Remove
      pre_x = ln->x;
  }
}
/*
   TODO:
      Geometric Properties
      Geometric Formulas
*/

int main(){}
```