Geometry Code Library.

**// setOnLine**

double setOnLine(double A,double B, double C, double x, double y) {

//this function checks the rotation of a point depending on line

double value = A \* x + B \* y + C;

return value;

}

**//Line Formation**

bool lineFormation(double x1, double y1, double x2, double y2, double &A, double &B,double &C){

A = y1-y2;

B = x2-x1;

C = x1 \* y2 - x2 \* y1; //Ax+By+C=0

}

**//Distance Check**

double distance(double x1, double y1, double x2, double y2) {

double value;

value = (x1-x2)\*(x1-x2) + (y1-y2)\*(y1-y2);

value = sqrt(value);

return value;

}

**// Line Line Intersection**

double lineLineIntersection(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4, double &x, double &y) {

//First line x1,y1 and x2,y2

//second line x3,y3 and x4,y4

//lines formation

//line A1x+B1y+C1=0; (y1-y2)\*x + (x2-x1)\*y+ x1\*y2-x2\*y1=0

double A1 = y1-y2;

double B1 = x2-x1;

double C1 = x1 \* y2 - x2 \* y1;

C1 = -C1;

//line A2x+B2y+C2 = 0; (y3-y4)\*x+(x4-x3)\*y+x3\*y4-x4\*y3 = 0;

double A2 = y3-y4;

double B2 = x4-x3;

double C2 = x3 \* y4 - x4 \* y3;

C2 = -C2;

//line intersection point find through intersection

double hor = A1\*B2 - A2 \* B1; // common hor

//finding x

x = (C1\*B2 - B1 \* C2)/hor;

//finding y

y = (A1 \* C2 - A2 \* C1)/hor;

}

**//Closest Pair of Points (O(NlogNlogN) approach not satisfactory**

struct point

{

ll x;

double y;

} Point[MAX+5];

bool cmp(struct point p, struct point q)

{

if(p.y<q.y) return true;

if(p.y == q.y)

{

if(p.x<q.x) return true;

else return false;

}

else return false;

}

bool cmpx(struct point p, struct point q)

{

if(p.x<q.x) return true;

if(p.x == q.x) {

if(p.y>q.y) return true;

else return false;

}

else return false;

}

double bruteForce(point p, point q)

{

return sqrt(((double)(p.x-q.x)\*(p.x-q.x)+(p.y-q.y)\*(p.y-q.y)));

}

point strip[MAX+10];

int flag[MAX+5];

double closestPair(point px[], int st, int en)

{

if((en-st+1)>=3)

{

int mid = (st+en)/2;

/\*\*Two side distance calculation\*\*/

double dl = closestPair(px,st,mid);

double dr = closestPair(px,mid+1,en);

double d = min(dl,dr);

int j=0;

//take the points which are closer than d

for(int i=mid;i>=st;i--) //the left array breaking

{

if((px[mid].x-px[i].x)<d)

{

j++;

strip[j]=px[i];

}

else break;

}

for(int i=mid+1;i<=en;i++) //the right array breaking

{

if((px[i].x-px[mid].x)<d)

{

j++;

strip[j]=px[i];

}

}

//sort on the basis of y

sort(strip+1,strip+1+j,cmp);

//Now think the special looking like 0(n^2) but practically for each point 7 points

for(int i=1;i<=j;i++)

{

int shuru=0;

for(int k=i+1;k<=j;k++)

{

if((strip[k].y-strip[i].y)<d)

{

shuru++;

d = min(d,bruteForce(strip[i],strip[k]));

assert(shuru<=10);

}

else break;

}

}

return d;

}

else

{

double d = INF;

for(int i=st;i<=en;i++)

{

for(int k=i+1;k<=en;k++)

{

double v = bruteForce(px[i],px[k]);

if(d>v)

{

d = v;

}

}

}

return d;

}

}

// Rotating Callipers + Graham Scan + Largest Diagonal

#define MAX 100005

#define PI acos(-1.0)

#define EPS 1e-12

typedef long long int ll;

int zero;

//convex hull generation

struct point

{

ll x,y; //co ordinate

double ang; // angle based on base point

double dist; // distance from the base point

ll bx,by;

}Point[MAX+5];

point start;

ll euclidianDistance(point a, point b)

{

ll value = (a.x-b.x)\*(a.x-b.x) + (a.y-b.y)\*(a.y-b.y);

return value;

}

ll largestDiagonal(vector<point>p,int n)

{

ll maxDiagonal = -1.0;

int j=0;

for(int i=0;i<n;i++)

{

for(;;j++)

{

j = j % n;

ll dist1 = euclidianDistance(p[i],p[j]);

ll dist2 = euclidianDistance(p[i],p[(j+1)%n]);

if(dist2>dist1)

{

maxDiagonal = max(maxDiagonal,dist2);

continue;

}

else

{

maxDiagonal = max(maxDiagonal,dist1);

break;

}

}

}

return maxDiagonal;

}

bool cmpY(point a, point b) //compare function on the basis of y

{

if(a.y<b.y) return true;

if(a.y == b.y)

{

if(a.x<b.x) return true;

else return false;

}

return false;

}

double findAngle(point a, point b)

{

double lob = b.y-a.y;

double hor = b.x-a.x;

if(hor == 0)

{

//cout<<"dhuki"<<endl;

if(lob == 0) return 0;

double angle = PI/2.0;

return angle;

}

double angle = atan(lob/hor);

if(angle<0) angle += PI;

return angle;

}

ll dis(point a, point b)

{

ll value;

value = (a.x-b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y) ;

return value;

}

int orientation(point O, point A, point B)

{

ll value = (A.x-O.x)\*(B.y-O.y)-(A.y-O.y)\*(B.x-O.x);

/\*point a = O;

point b = A;

point c = B;

value = a.x \* (b.y-c.y) - a.y \* (b.x - c.x) + b.x \* c.y - c.x \* b.y;\*/

if(value == 0) return 0;

if(value>0) return 1;

else return -1;

}

bool cmpConvexHull(point a, point b)

{

point p;

p.x = a.bx;

p.y = a.by;

int dir = orientation(p,a,b);

if(dir == 1) return true;

if(dir == -1) return false;

if(dir == 0)

{

ll dist1 = euclidianDistance(p,a);

ll dist2 = euclidianDistance(p,b);

if(dist1<dist2) return true;

else return false;

}

}

vector<point> convexHull(point p[], int n)

{

ll kom = p[0].y;

int idx=0;

for(int i=1;i<n;i++)

{

if(p[i].y<kom)

{

kom = p[i].y;

idx = i;

}

else if(p[i].y == kom)

{

if(p[i].x < p[idx].x)

{

idx = i;

}

}

}

point m=p[0];

p[0] = p[idx];

p[idx] = m;

for(int i=1;i<n;i++)

{

p[i].bx = p[0].x;

p[i].by = p[0].y;

}

sort(p+1,p+n,cmpConvexHull);

point secondP[MAX+5];

int counter=0;

secondP[counter]=p[0];

for(int i=1;i<n;i++)

{

while((i+1)<n && orientation(p[0],p[i],p[i+1])==0)

{

i++;

}

counter++;

secondP[counter] = p[i];

}

counter++;

for(int i=0;i<counter;i++)

{

p[i] = secondP[i];

}

n=counter;

if(n==1)

{

vector<point>answer;

answer.push\_back(p[0]);

return answer;

}

else if(n == 2)

{

vector<point>answer;

answer.push\_back(p[0]);

answer.push\_back(p[1]);

return answer;

}

//cout<<"ase"<<endl;

stack<point>S;

S.push(p[0]);

S.push(p[1]);

S.push(p[2]);

for(int i=3;i<n;i++)

{

while(S.size()>=2)

{

point a = S.top();

S.pop();

point b = S.top();

int dir = orientation(b,a,p[i]);

if(dir == 1)

{

S.push(a);

S.push(p[i]);

break;

}

else if(dir == 0)

{

S.push(a);

S.push(p[i]);

break;

}

else if(dir<0)

{

continue;

}

}

}

//cout<<S.size()<<endl;

vector<point>answer;

while(S.empty() != true)

{

answer.push\_back(S.top());

S.pop();

}

return answer;

}

// structure of points

struct Point{ //structure to find point

int x,y,z;

};

//3D Distance square

//euclidian distance

//int data type

int \_distance(Point a,Point b){

int value=(a.x-b.x)\*(a.x-b.x) + (a.y-b.y)\*(a.y-b.y) + (a.z - b. z) \* (a.z - b.z);

return value;

}

//Segment segment Intersection

//orientation

//function to check orientation

/\* x

\* x1..................x2

\* \*/

int orientation(Point x1,Point x,Point x2) {

int value = x1.x \* (x.y - x2.y) - x1.y \* (x.x - x2.x) + (x.x \* x2.y) - x2.x \* x.y ;

if(value > 0) return 1;

if(value < 0) return -1;

else return value; // return 0

}

//this function says either segment(x1,x2) and segment(A,B) intersects or not

bool segmentSegmentIntersection(Point x1,Point x2, Point A, Point B){

int d1 = orientation(A,x1,B);

int d2 = orientation(A,x2,B);

int d3 = orientation(x1,A,x2);

int d4 = orientation(x1,B,x2);

if(d1\*d2 < 0) {

if(d3\*d4 < 0) {

return true; //intersects

}

}

if(d1 \* d2 > 0) return false; // doesn't intersect

if(d1 \* d2 == 0) { // might be a chance of intesect

if(d1 == 0) {

if(x1.x>=min(A.x,B.x) && x1.x<=max(A.x,B.x) && x1.y >= min(A.y,B.y) && x1.y <= max(A.y,B.y)) return true;

}

if(d2 == 0) {

if(x2.x>=min(A.x,B.x) && x2.x<=max(A.x,B.x) && x2.y >= min(A.y,B.y) && x2.y <= max(A.y,B.y)) return true;

}

if(d3 == 0) {

if(A.x>=min(x1.x,x2.x) && A.x<=max(x1.x,x2.x) && A.y >= min(x1.y,x2.y) && A.y <= max(x1.y,x2.y)) return true;

}

if(d4 == 0) {

if(B.x>=min(x1.x,x2.x) && B.x<=max(x1.x,x2.x) && B.y >= min(x1.y,x2.y) && B.y <= max(x1.y,x2.y)) return true;

}

}

if(d3 \* d4 == 0) {

if(d1 == 0) {

if(x1.x>=min(A.x,B.x) && x1.x<=max(A.x,B.x) && x1.y >= min(A.y,B.y) && x1.y <= max(A.y,B.y)) return true;

}

if(d2 == 0) {

if(x2.x>=min(A.x,B.x) && x2.x<=max(A.x,B.x) && x2.y >= min(A.y,B.y) && x2.y <= max(A.y,B.y)) return true;

}

if(d3 == 0) {

if(A.x>=min(x1.x,x2.x) && A.x<=max(x1.x,x2.x) && A.y >= min(x1.y,x2.y) && A.y <= max(x1.y,x2.y)) return true;

}

if(d4 == 0) {

if(B.x>=min(x1.x,x2.x) && B.x<=max(x1.x,x2.x) && B.y >= min(x1.y,x2.y) && B.y <= max(x1.y,x2.y)) return true;

}

}

return false;

}

//Point inside a convex polygon

//ologn

struct Point{

ll x,y;

};

Point point[MAX+1];

//orientation

//function to check orientation

/\* x

\* x1..................x2

\* \*/

int orientation(Point x1,Point x,Point x2) {

ll value = x1.x \* (x.y - x2.y) - x1.y \* (x.x - x2.x) + (x.x \* x2.y) - x2.x \* x.y ;

if(value > 0) return 1;

if(value < 0) return -1;

else return value; // return 0

}

//this function checkes if the point is on the line or not

//line (A,B) point C

bool inSameSegment(Point A,Point B, Point C){

//already the tri area is zero calculated

if(C.x>=min(A.x,B.x) && C.y <= max(A.x,B.x) && C.y>=min(A.y,B.y) && C.y <= max(A.y,B.y)) {

return true;

}

return false;

}

//this function finds triangle area

/\* x

\* x1..................x2

\* \*/

long long int triArea(Point x1,Point x,Point x2){

long long int value = x1.x \* (x.y - x2.y) - x1.y \* (x.x - x2.x) + (x.x \* x2.y) - x2.x \* x.y ;

return abs(value);

}

/\*this function make binary search in polygon

\* finds a point remains in polygon or not

\* in logn

\*/

bool pointInsideConvexPolygon(Point p,int tot){

int low=1,high=tot-1;

while(low<=high){

int mid=(low+high)/2;

if(abs(low-high)==1) {

//cout<<"low = " << low <<" "<<high<<endl;

ll area = triArea(point[0],point[low],point[high]);

ll area1 = triArea(point[0],p,point[low]);

ll area2 = triArea(point[0],p,point[high]);

ll area3 = triArea(p,point[low],point[high]);

//cout<<area1<<" "<<area2<<" "<<area3<<endl;

if(area == (area1+area2+area3)) return true;

if(area1 == 0) {

if(inSameSegment(point[0],point[low],p) == true) return true;

}

if(area2 == 0) {

if(inSameSegment(point[0],point[high],p) == true) return true;

}

if(area3 == 0) {

if(inSameSegment(point[low],point[high],p) == true) return true;

}

else return false;

}

int d1 = orientation(point[0],p,point[mid]);

int d2 = orientation(point[0],p,point[high]);

if(d1\*d2>0) { //points are in same side of (0,mid) and (0,high) line reduce high

high=mid;

}

if(d1\*d2<0) {

low=mid;

}

if(d1\*d2 == 0) {

if(d1 == 0) {

bool v = inSameSegment(point[0],point[mid],p);

if(v == true) return true;

else return false;

}

if(d2 == 0) {

int v = inSameSegment(point[0],point[high],p);

if(v == true) return true;

else return false;

}

}

/\*if(abs(low-high)==1) {

cout<<"low = " << low <<" "<<high<<endl;

int area = triArea(point[0],point[low],point[high]);

int area1 = triArea(point[0],p,point[low]);

int area2 = triArea(point[0],p,point[high]);

int area3 = triArea(p,point[low],point[high]);

if(area == (area1+area2+area3)) return true;

else return false;

}\*/

}

return false;

}