**计算几何**

2020.09.05

目录

[计算几何 2](#_Toc50205072)

[几何元素 2](#_Toc50205073)

[点 2](#_Toc50205074)

[线 2](#_Toc50205075)

[圆 3](#_Toc50205076)

[线与线 3](#_Toc50205077)

[直线求交点 3](#_Toc50205078)

[线段求交点 3](#_Toc50205079)

[圆与线 3](#_Toc50205080)

[直线与圆求交点 3](#_Toc50205081)

[线段与圆求交点 3](#_Toc50205082)

[多边形求面积 3](#_Toc50205083)

[判断点在多边形内部 4](#_Toc50205084)

[半平面交 4](#_Toc50205085)

[凸包(Gramham) 6](#_Toc50205086)

[极角排序 6](#_Toc50205087)

[普通排序 7](#_Toc50205088)

[动态凸壳 7](#_Toc50205089)

[旋转卡壳 9](#_Toc50205090)

[最小圆覆盖 10](#_Toc50205091)

[Simpson积分 12](#_Toc50205092)

[扫描线 12](#_Toc50205093)

[圆的并 12](#_Toc50205094)

[其他技巧 12](#_Toc50205095)

[点的旋转 12](#_Toc50205096)

[反演 13](#_Toc50205097)

# 计算几何

## 几何元素

### 点

class Point{

public:

db x , y;

Point() {x = y = 0;

}

Point (db \_x , db \_y){

x = \_x;

y = \_y;

}

db len(){

return sqrt(x \* x + y \* y);

}

Point operator+(const Point u){

return Point(x + u.x , y + u.y);

}

Point operator-(const Point u){

return Point(x - u.x , y - u.y);

}

Point operator\*(const db v){

return Point(x \* v , y \* v);

}

Point operator/(const db v){

return Point(x / v , y / v);

}

};

db dot(const Point u, const Point v){ **//点积**

return u.x \* v.y + u.y \* v.x;

}

db cross(const Point u, const Point v){ **//叉积**

return u.x \* v.y - u.y \* v.x;

}

### 线

class Line{

public:

Point P , v; **//P是直线上一个点，v是方向向量**

db ang; **//ang是直线的角度，范围**

Line () {P.x = P.y = v.x = v.y = 0; ang = 0;}

Line (Point \_p , Point \_v) { P = \_p; v = \_v; ang = atan2(v.y , v.x); }

void set(){ ang = atan2(v.y , v.x); }

};

### 圆

## 线与线

### 直线求交点

Point GetInt(Line &a , Line &b){

Point u = a.P - b.P;

db t = cross(b.v , u) / cross(a.v , b.v);

return a.P + a.v \* t;

}

### 线段求交点

## 圆与线

### 直线与圆求交点

### 线段与圆求交点

## 多边形求面积

db area(Point \*p , int s){ **//p数组存储多边形的s个点**

db ans = 0;

for(int i = 1 ; i <= s-1 ; ++ i)

ans += cross(p[i] , p[i + 1]);

ans += cross(p[s] , p[1]);

if(ans < 0) ans = -ans;

return ans / 2;

}

## 判断点在多边形内部

## 半平面交

typedef long double db; **//不建议直接使用long double**

const int MAXN = 1e5 + 10;

const db eps = 1e-9;

class Point{

public:

db x , y;

Point() {x = y = 0;

}

Point (db \_x , db \_y){

x = \_x;

y = \_y;

}

db len(){

return sqrt(x \* x + y \* y);

}

Point operator+(const Point u){

return Point(x + u.x , y + u.y);

}

Point operator-(const Point u){

return Point(x - u.x , y - u.y);

}

Point operator\*(const db v){

return Point(x \* v , y \* v);

}

Point operator/(const db v){

return Point(x / v , y / v);

}

};

db dot(const Point u, const Point v){ **//点积**

return u.x \* v.y + u.y \* v.x;

}

db cross(const Point u, const Point v){ **//叉积**

return u.x \* v.y - u.y \* v.x;

}

class Line{

public:

Point P , v; **//P是直线上一个点，v是方向向量**

db ang; **//ang是直线的角度，范围**

Line () {P.x = P.y = v.x = v.y = 0; ang = 0;}

Line (Point \_p , Point \_v) { P = \_p; v = \_v; ang = atan2(v.y , v.x); }

void set(){ ang = atan2(v.y , v.x); }

};

bool equal(db n1 , db n2){ return fabs(n1 - n2) < eps;}

bool BothSide(Line &L , Point &A){ //判断A点是否在L的顺时针方向（合法区域）

return cross(L.v , (A - L.P)) > eps;

}

bool operator< (const Line &l1 , const Line &l2){

if(equal(l1.ang , l2.ang))

return BothSide(l2 , l1.P);

return l1.ang < l2.ang;

}

Point GetInt(Line &a , Line &b){ **//求两直线交点**

Point u = a.P - b.P;

db t = cross(b.v , u) / cross(a.v , b.v);

return a.P + a.v \* t;

}

Line q[MAXN];

Point p[MAXN];

int head , tail;

**// 半平面交，L是输入直线，U是输出半平面交的交点集合，返回值是是否能有剩余空间**

bool PIS(Line \*L , int length , Point \*U , int &m){

sort(L + 1 , L + length + 1);

q[head = tail = 0] = L[1];

Point g = GetInt(L[1] , L[2]);

for(int i = 2 ; i <= length ; ++ i){

if(equal(L[i].ang , L[i - 1].ang)) continue;

while(head < tail && !BothSide(L[i] , p[tail - 1]))

tail --;

while(head < tail && !BothSide(L[i] , p[head]))

head ++;

q[++ tail] = L[i];

if(head < tail)

p[tail - 1] = GetInt(q[tail] , q[tail - 1]);

}

while(head < tail && !BothSide(q[head] , p[tail - 1]))

tail --;

if(tail <= head + 1) return 0;

m = tail - head + 1;

for(int k = 1 ; k < m ; ++ k)

U[k] = p[k + head - 1];

U[m] = GetInt(q[head] , q[tail]);

return 1;

}

## 凸包(Gramham)

### 极角排序

找到右下角的点，所有点极角排序，单调栈处理。

struct POINT{

LL x, y;

POINT(LL a = 0, LL b = 0) {x = a; y = b;}

POINT operator - (const POINT & b){return POINT (x - b.x, y - b.y);}

}pt[N \* 2];

LL X(POINT a, POINT b) { return a.x \* b.y - a.y \* b.x;}

LL dist(POINT x, POINT y){

return (x.x - y.x) \* (x.x - y.x) + (x.y - y.y) \* (x.y - y.y);

}

int cmp(POINT x, POINT y){

return X(x - pt[1], y - pt[1]) > 0 || (X(x - pt[1], y - pt[1]) == 0 && dist(x, pt[1]) < dist(y, pt[1]));

}

void get\_convex()

{

int k = 1;

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

if(pt[i].y < pt[k].y || (pt[i].y == pt[k].y && pt[i].x < pt[k].x))

k = i;

swap(pt[k], pt[1]);

sort(pt + 2, pt + tot + 1, cmp);

que[1] = 1; top = 1;

for(int i = 2; i <= tot; i++)

{

while(top > 1 && X(pt[que[top]] - pt[que[top - 1]], pt[i] - pt[que[top - 1]]) <= 0) top--;

que[++top] = i;

}

que[top + 1] = 1;

mid\_top = 1;

for(; mid\_top <= top; mid\_top++)

if(pt[que[mid\_top]].y > pt[que[mid\_top + 1]].y)

break;

}

### 普通排序

## 动态凸壳

按照时间线建立线段树，在对应区间插入直线，每一个节点用vector存储所有直线。最后统一处理所有询问。求交点时可以二分。

const LL inf = 9e18;

vector <int> vec[N \* 4];

int S[N \* 2], E[N \* 2], st[N \* 2], que[N \* 2];

LL A[N \* 2], B[N \* 2], ans[N \* 2], Q[N \* 2], pst[N \* 2];

void update(int x, int l, int r, int ll, int rr, int t)

{

if(r < ll || rr < l) return;

if(ll <= l && r <= rr) { vec[x].push\_back(t); return; }

int mid = (l + r) >> 1;

update(x \* 2, l, mid, ll, rr, t);

update(x \* 2 + 1, mid + 1, r, ll, rr, t);

}

LL get\_f(int id, LL x)

{

LL a = A[id], b = B[id];

return (x - a) \* (x - a) \* (x - a) \* (x - a) + b;

}

LL get\_point(int x, int y)

{

LL l = 0, r = 50001;

while(r - l > 1)

{

LL mid = (l + r) >> 1;

if(get\_f(x, mid) > get\_f(y, mid)) r = mid;

else l = mid;

}

return l;

}

LL bi\_search(int top, LL x)

{

int l = 1, r = top + 1;

while(r - l > 1)

{

int mid = (l + r) >> 1;

if(pst[mid] < x) l = mid;

else r = mid;

}

LL a = A[st[l]], b = B[st[l]];

return (x - a) \* (x - a) \* (x - a) \* (x - a) + b;

}

int cmp(int x, int y){ return A[x] < A[y];}

void pushdown(int x, int l, int r)

{

if(vec[x].size())

{

sort(vec[x].begin(), vec[x].end(), cmp);

st[1] = vec[x][0];

int top = 1;

for(int i = 1; i < vec[x].size(); i++)

{

while(top>1&&pst[top]>=get\_point(st[top],vec[x][i])) top--;

st[++top] = vec[x][i];

pst[top] = get\_point(st[top - 1], st[top]);

}

for(int i = l; i <= r; i++)

if(Q[i])

ans[i] = min(ans[i], bi\_search(top, Q[i]));

}

if(l == r) return;

int mid = (l + r) >> 1;

pushdown(x \* 2, l, mid);

pushdown(x \* 2 + 1, mid + 1, r);

}

int main()

{

int T, n, m;

scanf("%d", &T);

while(T)

{

T--;

scanf("%d%d", &n, &m);

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

scanf("%lld%lld", &A[i], &B[i]); S[i] = 1;

}

int cnt = 0;

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

{

Q[i] = 0;

int opt, t;

LL x;

scanf("%d", &opt);

switch(opt)

{

case 1: n++;scanf("%lld%lld",&A[n],&B[n]);S[n]=i;break;

case 2: scanf("%d", &t); E[t] = i; break;

case 3: scanf("%lld", &x);Q[i]=x; que[++cnt] = i; break;

}

}

for(int i = 1; i <= m \* 4; i++) vec[i].clear();

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

{

if(!E[i]) E[i] = m;

update(1, 1, m, S[i], E[i], i);

}

A[n + 1] = 0; B[n + 1] = 0;

for(int i = 1; i <= m; i++) ans[i] = inf;

pushdown(1, 1, m);

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

{

if(ans[que[i]] == inf) cout<<"-1"<<endl;

else cout<<ans[que[i]]<<endl;

}

for(int i = 1; i <= n; i++) E[i] = 0;

}

return 0;

}

## 旋转卡壳

先求凸包，然后对于每一个点找对踵点。注意判断凸包退化的情况。

int j=1;

for(int i=0;i<stop;i++)//旋转卡壳

{

while(abs(S(que[i],que[(i+1)%stop],que[j%stop]))<abs(S(que[i],que[(i+1)%stop],que[(j+1)%stop])))

j++;

ans=max(ans,cal(que[i],que[j%stop]));

}

## 最小圆覆盖

**//期望时间复杂度O(n)**

typedef long double db;

const int N = 1e5 + 10;

const db eps = 1e-9;

class Point{

public:

db x , y;

Point() {x = y = 0;

}

Point (db \_x , db \_y){

x = \_x;

y = \_y;

}

db len(){

return sqrt(x \* x + y \* y);

}

Point operator+(const Point u){

return Point(x + u.x , y + u.y);

}

Point operator-(const Point u){

return Point(x - u.x , y - u.y);

}

Point operator\*(const db v){

return Point(x \* v , y \* v);

}

Point operator/(const db v){

return Point(x / v , y / v);

}

};

db dot(const Point u, const Point v){

return u.x \* v.y + u.y \* v.x;

}

db cross(const Point u, const Point v){

return u.x \* v.y - u.y \* v.x;

}

Point rotate\_90(const Point u){

return Point(u.y , -u.x);

}

class Line{

public:

Point P , v;

db ang;

Line () {P.x = P.y = v.x = v.y = 0; ang = 0;}

Line (Point \_p , Point \_v) { P = \_p; v = \_v; ang = atan2(v.y , v.x); }

void set(){ ang = atan2(v.y , v.x); }

};

bool equal(db n1 , db n2){ return fabs(n1 - n2) < eps;}

Point GetInt(Line &a , Line &b){ //ÇóÁ½Ö±Ïß½»µã

Point u = a.P - b.P;

db t = cross(b.v , u) / cross(a.v , b.v);

return a.P + a.v \* t;

}

int n;

Point p[N];

Point work(Point p0 , Point p1 , Point p2){**//三点成圆（非共线）**

Point t1 = p1 - p0 , t2 = p2 - p0;

Line l1 = Line((p0 + p1) / 2 , rotate\_90(t1)) , l2 = Line((p2 + p0) / 2 , rotate\_90(t2));

return GetInt(l1 , l2);

}

int main(){

ios::sync\_with\_stdio(0);

cin >> n;

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

cin >> p[i].x >> p[i].y;

random\_shuffle(p + 1 , p + n + 1);

Point ans = p[1]; db r = 0;

for(int i = 2 ; i <= n ; ++ i)

if((p[i] - ans).len() > r){

ans = p[i]; r = 0;

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

if((p[j] - ans).len() > r){

ans = (p[i] + p[j]) / 2; r = (p[i] - ans).len();

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

if((p[k] - ans).len() > r){

ans = work(p[i] , p[j] , p[k]); r = (p[i] - ans).len();

}

}

}

cout.precision(10);

cout << fixed << r << endl;

cout << fixed << ans.x << " " << ans.y << endl;

return 0;

}

## Simpson积分

Simpson积分对于**连续函数**来说有比较好的效果。

建议对于**突变点分段**进行Simpson积分。

typedef double db;

**//F(x)表示x点的上边界减下边界**

db simpson(db l, db r){

db mid = l + (r - l) / 2;

return (F(l) + 4 \* F(mid) + F(r)) \* (r - l) / 6;

}

db cal(db l, db r){

db mid = l + (r - l) / 2;

db L = simpson(l , mid) , R = simpson(mid , r) , A = simpson(l , r);

if(fabs(L + R - A) < 5 \* eps)

return L + R + fabs(L + R - A) / 5.0;

return cal(l , mid) + cal(mid , r);

}

## 扫描线

### 圆的并

## 其他技巧

### 点的旋转

Point rotate(const Point &A , db ang) {

return Point(A.x \* cos(ang) - A.y \* sin(ang) , A.x \* sin(ang) + A.y \* cos(ang));

}

有的时候可以旋转坐标系进行

### 反演

#### 定义

⊙O半径为r，线段PQ过圆心，，则P、Q为一对关于⊙O的反演点。将一个图形逐点反演后得到的图形为其反形。

#### 性质

1. **过**反演中心的圆，反形为一条直线。
2. 不过反演中心的直线，反形为一个过反演中心的圆。
3. 不过反演中心的圆，反形为一个不过反演中心的圆，反演中心为两个圆的位似中心。
4. 相切两圆反形仍相切。若切点是反演中心，那么得到两条平行线。

根据这些性质，可以将相切的圆/过一点的圆转化成圆的外切线/过一点的直线。