Code Template for ACM-ICPC

LaiYiC

2021 年 9 月 18 日

目录

1 计算几何

1.1 头文件

1.1.1 头文件

```
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<string.h>
#include<math.h> //hypot
#include <iomanip>
//cout << fixed << setprecision(10) << x << endl;#include <iomanip>
/*#define double long double
scanf("%LF",&x);printf("%LF\n",x);
*/
using namespace std;
typedef long long ll;
const double eps = 1e-8;
const double PI = acos(-1);

const int N = 1e5 + 50;

int main() {
//	freopen("in.txt","r",stdin);
	int cc=1;
//	scanf("%d",&cc);
	for(int i=0; i<cc; i++) {
		work();
	}
	return 0;
}
```

1.2 凸包

1.2.1 1 凸包模板

```
const int N = 1e3 + 50;
const double eps = 1e-8;

//判断x是否为0;是的话return 0
int sgn(double x) {
	if(fabs(x) < eps)return 0;
	else return x<0?-1:1;
}
```

```
struct Point {
	double x,y;
	Point(double X=0,double Y =0) {
		x = X,y = Y;
	}
	Point operator+ (Point B) {return Point(x + B.x,y+B.y);}
	Point operator- (Point B) {return Point(x - B.x,y-B.y);}
	bool operator == (Point B) {return sgn(x-B.x) == 0 && sgn(y-B.y) == 0;}
	Point operator* (double k) {return Point(x*k,y*k);}
	Point operator/(double k) {return Point(x/k,y/k);}
	bool operator<(Point B) {
		return sgn(x-B.x)<0 || (sgn(x-B.x)==0 && sgn(y-B.y) < 0);
	}
};
typedef Point Vector;

double Cross(Vector A,Vector B) {return A.x*B.y-A.y*B.x;}
double Distance(Point A,Point B) {return hypot(A.x-B.x,A.y-B.y);}

struct Line {
	Point p1,p2;
	Line() {};
	Line(Point p1,Point p2):p1(p1),p2(p2) {}
};

int n;
Point p[N],ch[N]; //输入点,凸包顶点

int Convex_hull(Point *P,int n,Point *ch) {
	sort(p,p+n); //p点按照x升序,x一样y升序
	n = unique(p,p+n)-p;
	int v =0 ;
	//向下求凸包
	for(int i=0; i<n; i++) {
		while(v > 1 && sgn(Cross(ch[v-1]-ch[v-2],p[i]-ch[v-2])) <= 0)v--;
		ch[v++] = p[i];
	}
	int j = v;
	//向上求凸包
	for(int i=n-2; i>=0; i--) {
		while(v > j && sgn(Cross(ch[v-1]-ch[v-2],p[i]-ch[v-2])) <= 0)v--;
		ch[v++] = p[i];
	}
	if(n > 1)v--;
	return v;
```

```
}

void work() {
	while(~scanf("%d",&n) && n) {
		for(int i=0; i<n; i++) {
			scanf("%lf %lf",&p[i].x,&p[i].y);
		}
		int v = Convex_hull(p,n,ch); //凸包顶点数
		double ans = 0;
		if(v == 1)ans = 0;
		else if(v == 2)ans = Distance(ch[0],ch[1]);
		else {
			for(int i=0; i<v; i++) {
				ans+=Distance(ch[i],ch[(i+1)%v]);
			}
		}
		printf("%.2f\n",ans); //凸包的周长
	}
}
```

1.2.2 2 极角排序

```
1.相对p[pos]极角排序,用Cross判断方向
bool cmp(Point a,Point b) {
	double t=Cross(a-p[pos],b-p[pos]);
	if(sgn(t)==0)return Distance(p[pos],a)<Distance(p[pos],b);
	else if(sgn(t)<0)return false;
	return true;
}
```

```
2.相对原点用下面的:
p[i].angle = atan2(p[i].y,p[i].x);
if(p[i].angle < 0)p[i].angle += 2*PI; //0~2PI


bool cmp(Point a,Point b){ //排序结果是3 4 1 2象限
	if(atan2(a.y,a.x)!=atan2(b.y,b.x))
	  return atan2(a.y,a.x)<atan2(b.y,b.x);
	return a.x<b.x;
}
/*************************************/
题意:逆时针绕圈,问最长怎么走,依次对每个点进行一遍极角排序

int sgn(double x) {
```

```
        if(fabs(x) < eps)return 0;
        else return x<0?-1:1;
}
struct Point {
        double x,y;
        int id;
        Point() {};
        Point (double x,double y):x(x),y(y) {}
        Point operator+(Point B) {
                return Point(x+B.x,y+B.y);
        }
        Point operator-(Point B) {
                return Point(x-B.x,y-B.y);
        }
        Point operator/(double k) {
                return Point(x/k,y/k);
        }
        bool operator == (Point B) {
                return sgn(x-B.x)==0&&sgn(y-B.y)==0;
        }
};

struct Line {
        Point p1,p2;
        Line() {}
        Line(Point p1,Point p2):p1(p1),p2(p2) {}
};

typedef Point Vector;

double Dot(Vector A,Vector B) {
        return A.x*B.x+A.y*B.y;
}

double Cross(Vector A,Vector B) {
        return A.x*B.y-A.y*B.x;
}
double Distance(Point A,Point B) {
        double xx = A.x - B.x;
        double yy = A.y - B.y;
        return sqrt(xx*xx+yy*yy);
}
int n;
int pos;
Point p[N];
```

```
bool cmp(Point a,Point b) {
	double t=Cross(a-p[pos],b-p[pos]);
	if(sgn(t)==0)return Distance(p[pos],a)<Distance(p[pos],b);
	else if(sgn(t)<0)return false;
	return true;

}
void work() {
	scanf("%d",&n);
	double x,y;
	for(int i=0; i<n; i++) { //先找到左下角的点
		scanf("%d%lf%lf",&p[i].id,&p[i].x,&p[i].y);
//		if(p[i].x < p[0].x || (p[i].x == p[0].x
//				&& p[i].y < p[0].y))swap(p[0],p[i]);
	//注意,先判最下面再判最左边,不然wa
	if(p[i].y < p[0].y || (p[i].y == p[0].y && p[i].x < p[0].x))swap(p[i],p[0]);
	}
	pos =0 ;
	for(int i=1; i<n; i++) {
		sort(p+i,p+n,cmp);
		pos++;
	}
	printf("%d",n);
	for(int i=0;i<n;i++){
		printf(" %d",p[i].id);
	}
	printf("\n");
}
```

1.2.3 3 凸包内最大三角形

```
int sgn(double x) {
	if(fabs(x) < eps)return 0;
	return x<0?-1:1;
}

struct Point {
	double x,y;
	Point(double _x=0,double _y=0) {
		x=_x,y=_y;
	}
	Point operator +(const Point &b)const {return Point(x+b.x,y+b.y);}
	Point operator -(const Point &b)const {return Point(x-b.x,y-b.y);}
	double operator ^(const Point &b)const {return x*b.y-y*b.x;}
	double operator *(const Point &b)const {return x*b.x+y*b.y;}
```

```
	bool operator == (Point b)const {return sgn(x-b.x)==0&& sgn(y-b.y)==0;}

	bool operator <(Point b)const {return sgn(x-b.x)==0?sgn(y-b.y)<0:x<b.x;}
	double distance(Point p) {return hypot(x-p.x,y-p.y);}
	void input() {scanf("%lf%lf",&x,&y);}
};

int id1,id2,id3;
int n;
Point p[N],list[N];

struct polygon {
	int n;
	Point p[N];
	void input(int _n) {
		n = _n;
		for(int i=0; i<n; i++) {
			p[i].input();
		}
	}
};

polygon vv;

struct cmp {
	Point p;
	cmp(const Point &p0) {
		p = p0;
	}
	bool operator()(const Point &aa,const Point &bb) {
		Point a = aa,b = bb;
		int d = sgn((a-p)^(b-p));
		if(d == 0)return sgn(a.distance(p)-b.distance(p))<0;
		return d > 0;
	}
};
void norm() {
	Point mi= p[0];
	for(int i=1; i<n; i++)mi=min(mi,p[i]);
	sort(p,p+n,cmp(mi));
}

void Graham(polygon &convex) {
	norm();
	int &top=convex.n;
	top = 0;
```

```
        if(n == 1) {
                top=1;
                convex.p[0] = p[0];
                return ;
        }
        if(n == 2) {
                top=2;
                convex.p[0] = p[0];
                convex.p[1] = p[1];
                if(convex.p[0] == convex.p[1])top--;
                return ;
        }

        convex.p[0] = p[0];
        convex.p[1] = p[1];
        top = 2;

        for(int i=2; i<n; i++) {
                while(top>1&& sgn((convex.p[top-1]-convex.p[top-2])^(p[i]-convex.p[top-2]))
                    <=0)top--;
                convex.p[top++] = p[i];
        }
        if(convex.n == 2 && (convex.p[0]==convex.p[1]))convex.n--;
}

Point p1,p2,p3;

/******凸包内最大三角形*****/
double rotating(Point p[],int n) {
        double ans = 0;
        Point v;
        for(int i=0; i<n; i++) {
                int j=(i+1)%n;
                int k=(j+1)%n;
                while(j!=i&&k!=i) {
                        double res = fabs((p[i]-p[j])^(p[k]-p[i]));
                        if(res > ans) {
                                ans = res;
//                              p1=p[i],p2=p[j],p3=p[k]; //记录那个三角形
                        }
                        while(((p[i]-p[j])^(p[(k+1)%n]-p[k])) < 0) k=(k+1)%n;
                        j=(j+1)%n;
                }
        }
        return ans/2.0;
}
```

```
void work() {
	scanf("%d",&n);
	vv.n = n;
	for(int i=0; i<n; i++) {
		p[i].input();
		vv.p[i] = p[i];
	}
	Graham(vv);
	double ans = rotating(vv.p,vv.n); //最大三角形面积

}
```

1.2.4 4 凸包被直线切割面积

```
//给一个逆时针的凸包和一条线,问你线的左边的和凸包的交面积

int n;
Point p[N],ch[N];
Point last[N]; //最后存在的点
//两直线交点
Point Cross_point(Point a,Point b,Point c,Point d) { //Line1:ab, Line2:cd
	double s1 = Cross(b-a,c-a);
	double s2 = Cross(b-a,d-a); //叉积有正负
	return Point(c.x*s2-d.x*s1,c.y*s2-d.y*s1)/(s2-s1);
}

double area(int n,Point a[]) { //求面积[0,n]
	double res=0;
	n++;
	for(int i=0; i<n; i++) {
		res+=(a[i]-a[0])^(a[(i+1)%n]-a[0]);
	}
	return res/2.0;
}

double convex_cut(Point p1,Point p2) { //p1->p2,右边不要
	int al=-1;
	for(int i=0; i<n; i++) {
		int t1=sgn((p2-p1)^(p[i]-p1));
		int t2=sgn((p2-p1)^(p[(i+1)%n]-p1));
		if(t1 >=0)last[++al]=p[i]; //p1在线左边(逆时针方向)
		if(t1*t2<0)last[++al]=Cross_point(p[i],p[(i+1)%n],p1,p2); //直线穿过,取交点
	}
	double res=area(al,last);
```

```
        return res;
}

void work() {
        scanf("%d",&n);
        for(int i=0; i<n; i++) {//逆时针给出的凸包
                scanf("%lf%lf",&p[i].x,&p[i].y);
        }
        int q;scanf("%d",&q);
        while(q--) {
                Point p1,p2;
                scanf("%lf%lf%lf%lf",&p1.x,&p1.y,&p2.x,&p2.y);
                double ans = convex_cut(p1,p2);
                printf("%.8f\n",ans);
        }
}
```

1.2.5 5 动态凸包

```
/****动态添加点,判断点是否在凸包内部****/
#define fi first #define se second
using namespace std;
typedef long long ll;
#define MP make_pair
typedef pair<int,int>PII;
map<int,int>convex[2];
map<int,int>::iterator p,q,it,it1,it2;

ll Cross(PII a,PII b,PII c) {
        return 1ll*(b.fi-a.fi)*(c.se-a.se)-1ll*(b.se-a.se)*(c.fi-a.fi);
}
//判断点是否在凸包内部
bool judge(map<int,int>&st,int x,int y) {
        if(!st.size())return false;
        if(st.find(x)!=st.end())return y>=st[x];
        if(x<st.begin()->fi || (--st.end())->fi < x)return false;
        p = st.lower_bound(x);
        q = p,q--;    //找到左右点
        return Cross(MP(x,y),*q,*p)>=0;
}

void insert(map<int,int>&st,int x,int y) {
        if(judge(st,x,y))return ;
        st[x] = y;
        p = st.upper_bound(x);
```

```
        it = p,it--;
        it1 = it,it1--;
        it2 = it1,it2--;
        if(p != st.end()) {
                q = p,q++;
                while(q != st.end() && Cross(MP(x,y),*q,*p)>=0) {
                        st.erase(p);
                        p = q,q ++;
                }
        }
        if(it == st.begin() || it1 == st.begin())return ;
        while(it1 != st.begin() && Cross(MP(x,y),*it1,*it2)>=0) {
                st.erase(it1),it1=it2,it2--;
        }
}

int n;
void work() {
        scanf("%d",&n);
        while(n--) {
                int op,x,y;
                scanf("%d%d%d",&op,&x,&y);
                if(op == 1) {
                        insert(convex[0],x,y);
                        insert(convex[1],x,-y);
                } else {
                        bool ans1=judge(convex[0],x,y);
                        bool ans2=judge(convex[1],x,-y);
                        if(ans1&&ans2)puts("YES");
                        else puts("NO");
                }
        }
}
```

1.2.6 6 最大空凸包

```
//最大空凸包,里面没有点.O(n^3)

const int N = 1e2+50;
int sgn(double x) { //判断x是否等于0
        if(fabs(x) < eps) return 0;
        else return x<0?-1:1;
}

struct Point {
```

```
        double x,y;
        Point() {}
        Point(double x,double y):x(x),y(y) {}
        Point operator + (Point B) {return Point(x+B.x,y+B.y);}
        Point operator - (Point B) {return Point(x-B.x,y-B.y);}
        Point operator / (double k) {return Point(x/k,y/k);}
        double dis() {return x*x+y*y;}
};

struct Line {
        Point p1,p2;
        Line() {}
        Line(Point p1,Point p2):p1(p1),p2(p2) {}
};

int n;
Point o;
int tot;
Point a[N],p[N];

double dp[N][N],ans;
//dp[i][j]表示组成的凸包以o,i,j为最后一个三角形的最大面积

typedef Point Vector;

double Cross(Vector A,Vector B) {return A.x*B.y - A.y*B.x; //叉积}

bool cmp(Point a,Point b) {
        double res=Cross(a-o,b-o);
        if(sgn(res)!=0)return res>0; //a在b的顺时针方向
        return (a-o).dis() < (b-o).dis();
}

void solve() {
        memset(dp,0,sizeof(dp));
        sort(p+1,p+1+tot,cmp);
        for(int i=1; i<=tot; i++) {
                int j=i-1;
                while(j&&!Cross(p[i]-o,p[j]-o))j--;
                bool bz=(j==i-1);
                while(j){
                        int k=j-1;
                        while(k&&Cross(p[i]-p[k],p[j]-p[k])>0)k--;
                        double area=fabs(Cross(p[i]-o,p[j]-o))/2;
                        if(k)area+=dp[j][k];
                        if(bz)dp[i][j]=area;
```

```
                    ans = max(ans,area);
                    j=k;
                }
                if(bz)for(int j=1;j<i;j++)dp[i][j]=max(dp[i][j],dp[i][j-1]);
        }
}

void work() {
        scanf("%d",&n);
        ans =0;
        for(int i=1; i<=n; i++) {
                scanf("%lf%lf",&a[i].x,&a[i].y);
        }
        for(int i=1; i<=n; i++) { //选择凸包左下角的点
                o=a[i];
                tot=0;
                for(int j=1; j<=n; j++) {
                        if(a[j].y>a[i].y || (a[i].y==a[j].y && a[j].x>a[i].x))p[++tot]=a[j];
                }
        solve();
        }
        printf("%.1f\n",ans);
}
```

1.2.7 7 求上凸包

求上凸包,输出字典序最小(在同一条线上)的方案 Hdu6325**
凸包，逆时针走叉积最大，顺时针走叉积最小。

```
Sample Input
1
3
0 0
3 0
4 0
Sample Output
1 2 3
int sgn(double x) {
        if(fabs(x) < eps)return 0;
        else return x<0?-1:1;
}

struct Point {
        ll x,y;
        int id;
        Point() {}
```

```
        Point(ll x,ll y,int id):x(x),y(y),id(id) {};
        Point operator + (Point B) {
                return Point(x + B.x,y+B.y,0);
        }
        Point operator - (Point B) {
                return Point(x - B.x,y - B.y,0);
        }
        bool operator == (Point B) {
                return sgn(x-B.x) == 0 && sgn(y-B.y) == 0;
        }
        bool operator < (Point B) {
                return sgn(x-B.x)<0 || (sgn(x-B.x) == 0 && sgn(y-B.y)>0) ||
                        (sgn(x-B.x) == 0 && sgn(y-B.y) == 0 && id < B.id);
        }
};

typedef Point Vector;
ll Cross(Vector A,Vector B) {
        return A.x * B.y - A.y * B.x;
}
int ok(Point A,Point B,Point C) {
        return Cross(B-A,C-A) == 0 ? 0:1;
}

Point p[N],ch[N];
int vis[N],ans[N];
int n;

void work() {
        n = rd();
        for(int i=1; i<=n; i++) {
                p[i].x = rd(),p[i].y = rd();
                p[i].id = i;
        }
        sort(p+1,p+1+n);
        int v =0 ;
        for(int i=1; i<=n; i++) { //求上凸包
                if(i > 1 && p[i].x == p[i-1].x)continue;
         //这里Cross方向要注意,等于0时在同一直线上.
                while(v > 1 && Cross(p[i]-ch[v-2],ch[v-1]-ch[v-2])<0)v--;
                ch[v++] = p[i];
        }
        rep(i,0,v)vis[i]=0;
        vis[0] = vis[v-1] = 1;
        for(int i=1; i<=v-2; i++) {
                vis[i] = ok(ch[i-1],ch[i],ch[i+1]); //若不在同一直线上,这个点必走
```

```
        }

        for(int i = v-1; i>=0; i--) {
                if(vis[i])ans[i] = ch[i].id;
                else ans[i] = min(ch[i].id,ans[i+1]); //如果这个点比后一个点小,字典序更小
        }

        for(int i=0; i<v; i++) {
                if(ans[i] == ch[i].id) {
                        printf("%d",ans[i]);
                        if(i != v-1)printf(" ");
                        else printf("\n");
                }
        }
}
```

1.2.8 8 凸包切线

题意：给点N棵树，前K棵是已经拥有的，现在可以再拥有一棵树，问形成的最大凸包面积。
思路：先求K棵树的凸包C，然后对于后面的N-K棵树，我们先判断是否在凸包内，如果不在，我们要求两个切线。
这里分类讨论，即可。
如果点在C的左边，那么两条切线分别一上一下； 如果在下边，两条切线一左一右。 然后去对应区间二分即可。

```
#include<bits/stdc++.h>
#define ll long long
#define rep(i,a,b) for(int i=a;i<=b;i++)
using namespace std;
const int maxn=200010;
struct point{
    ll x,y;
    point(){}
    point(ll xx,ll yy):x(xx),y(yy){}
};
bool cmp(point w,point v){
    if(w.x!=v.x) return w.x<v.x;
    return w.y<v.y;
}
ll det(point a,point b){ return a.x*b.y-a.y*b.x;}
ll dot(point a,point b){ return a.x*b.x+a.y*b.y;}
point operator +(point a,point b){ return point(a.x+b.x,a.y+b.y);}
point operator -(point a,point b){ return point(a.x-b.x,a.y-b.y);}
point a[maxn],ch[maxn]; int top,ttop;
void convexhull(int N)
{
    for(int i=1;i<=N;i++){
```

```
        while(top>1&&det(ch[top]-ch[top-1],a[i]-ch[top-1])<=0) top--;
        ch[++top]=a[i];
    }
    ttop=top;
    for(int i=N-1;i>=1;i--){
        while(top>ttop&&det(ch[top]-ch[top-1],a[i]-ch[top-1])<=0) top--;
        ch[++top]=a[i];
    }
}
int get(int L,int R,int i,int w)
{
    while(L<R){
        int Mid=(L+R)>>1;
        if(det(ch[Mid]-a[i],ch[Mid+1]-a[i])*w>0) R=Mid;
        else L=Mid+1;
    }
    return L;
}
int bord(int L,int R,int i,int w)
{
    while(L<R){
        int Mid=(L+R)>>1;
        if((ch[Mid].x-a[i].x)*w<0) L=Mid+1;
        else R=Mid;
    }
    return L;
}
ll ans,sum[maxn],tmp;
int main()
{
    int N,K;
    scanf("%d%d",&N,&K);
    rep(i,1,N) scanf("%lld%lld",&a[i].x,&a[i].y);
    sort(a+1,a+K+1,cmp); convexhull(K);
    rep(i,1,top-1) ans+=det(ch[i],ch[i+1]),sum[i+1]=ans;
    rep(i,K+1,N){
        if(a[i].x<ch[1].x){
            int L=get(1,ttop,i,1),R=get(ttop,top,i,-1);
            tmp=sum[R]-sum[L]+det(ch[R],a[i])+det(a[i],ch[L]);
        }
        else if(a[i].x>ch[ttop].x){
            int L=get(1,ttop,i,-1),R=get(ttop,top,i,1);
            tmp=sum[top]-sum[R]+sum[L]+det(ch[L],a[i])+det(a[i],ch[R]);
        }
        else if(det(ch[ttop]-a[1],a[i]-ch[1])>0){//shang
            int Mid=bord(ttop,top,i,-1);
```

```
            if(Mid>ttop&&det(ch[Mid]-ch[Mid-1],a[i]-ch[Mid-1])>0) continue;
            int L=Mid>ttop?get(ttop,Mid-1,i,-1):Mid;
            int R=get(Mid,top,i,1);
            tmp=sum[top]-sum[R]+sum[L]+det(ch[L],a[i])+det(a[i],ch[R]);
        }
        else {
            int Mid=bord(1,ttop,i,1);
            if(Mid>1&&det(ch[Mid]-ch[Mid-1],a[i]-ch[Mid-1])>0) continue;
            int L=Mid>1?get(1,Mid-1,i,-1):1;
            int R=get(Mid,ttop,i,1);
            tmp=sum[top]-sum[R]+sum[L]+det(ch[L],a[i])+det(a[i],ch[R]);
        }
        ans=max(ans,tmp);
    }
    printf("%lld.%lld\n",ans/2,ans%2*5);
    return 0;
}
/*
5 3
-5 -5
-5 5
5 -5
-4 6
5 5
->100*/
```

1.3 半平面交

1.3.1 1 线性规划

```
//每次向平面增加一条线,每次询问加入后的平面交面积
typedef double ld;
const int N = 1e5 + 50;
const ld eps = 1e-18;
const ld PI = acos(-1.0);
int sgn(ld x) {
        if(fabs(x)<eps)return 0;
        return x<0?-1:1;
}
struct Point {
        ld x,y;
        Point() {}
        Point(ld _x,ld _y) {x=_x,y=_y;}
        Point operator -(const Point &b)const {return Point(x - b.x, y - b.y);}
        double operator ^(const Point &b)const {return x*b.y - y*b.x;}
```

```
        double operator *(const Point &b)const {return x*b.x + y*b.y;}
};
ld Area(Point p[],int n) {
        ld res=0;
        for(int i=0; i<n; i++) {
                res += (p[i]^p[(i+1)%n]);
        }
        return fabs(res/2.0);
}

int n;
Point p[N];
ld v[N],u[N],w[N];

//通过两点，确定直线方程
void Get_equation(Point p1,Point p2,ld &a,ld &b,ld &c) {
        a = p2.y - p1.y;
        b = p1.x - p2.x;
        c = p2.x*p1.y - p1.x*p2.y;
}
//求交点
Point Intersection(Point p1,Point p2,ld a,ld b,ld c) {
        ld u = fabs(a*p1.x + b*p1.y + c);
        ld v = fabs(a*p2.x + b*p2.y + c);
        Point t;
        t.x = (p1.x*v + p2.x*u)/(u+v);
        t.y = (p1.y*v + p2.y*u)/(u+v);
        return t;
}
Point tp[110];
void Cut(ld a,ld b,ld c,Point p[],int &cnt) {
        int tmp = 0;
        for(int i = 1; i <= cnt; i++) {
                //当前点在左侧，逆时针的点
                if(a*p[i].x + b*p[i].y + c < eps)tp[++tmp] = p[i];
                else {
                        if(a*p[i-1].x + b*p[i-1].y + c < -eps)
                                tp[++tmp] = Intersection(p[i-1],p[i],a,b,c);
                        if(a*p[i+1].x + b*p[i+1].y + c < -eps)
                                tp[++tmp] = Intersection(p[i],p[i+1],a,b,c);
                }
        }
        for(int i = 1; i <= tmp; i++)
                p[i] = tp[i];
        p[0] = p[tmp];
        p[tmp+1] = p[1];
```

```
        cnt = tmp;
}

bool solve(int id) {
        p[1] = Point(0,0);
        p[2] = Point(INF,0);
        p[3] = Point(INF,INF);
        p[4] = Point(0,INF);
        p[0] = p[4];
        p[5] = p[1];
        int cnt = 4;
        for(int i = 0; i < n; i++)
                if(i != id) { //所有ax+by+c<0围成的面积
                        double a = (v[i] - v[id])/(v[i]*v[id]);
                        double b = (u[i] - u[id])/(u[i]*u[id]);
                        double c = (w[i] - w[id])/(w[i]*w[id]);
                        if(sgn(a) == 0 && sgn(b) == 0) {
                                if(sgn(c) >= 0)return false;
                                else continue;
                        }
                        Cut(a,b,c,p,cnt);
                }
        if(sgn(Area(p,cnt)) == 0)return false;
        else return true;
}
//void test() { //测试板子打对了
//      p[1] = Point(0,0);
//      p[2] = Point(INF,0);
//      p[3] = Point(INF,INF);
//      p[4] = Point(0,INF);
//      p[0] = p[4];
//      p[5] = p[1];
//      int cnt = 4;
//      Cut(-2,1,0,p,cnt);
//      Cut(2,1,-4,p,cnt);
//      cout << Area(p,cnt) << endl; //output:2
//}
void work() {
        while(~scanf("%d",&n)) {
                for(int i=0; i<n; i++) {
                        scanf("%lf%lf%lf",&v[i],&u[i],&w[i]);
                }
                for(int i=0; i<n; i++) {
                        if(solve(i))printf("Yes\n");
                        else printf("No\n");
                }
```

```
        }
}
```

1.3.2 2 半平面交二分

给一个凸包,问凸包上距离边界最近的点最大化是多少

二分半径r,将凸包上的边向内平移r,若产生的半平面交存在,扩大r;

```
//由相似三角形得平移
void change(Point a,Point b,Point &c,Point &d,double p) {
        double len = dist(a,b);
        double dx = (a.y-b.y)*p/len;
        double dy = (b.x-a.x)*p/len;
        c.x=a.x+dx,c.y=a.y+dy;
        d.x=b.x+dx,d.y=b.y+dy;
}
//二分
double l=0,r=100000,ans =0;
while(r-l>=eps) {
        double mid =(l+r)/2;
        for(int i=0; i<n; i++) {
                Point p1,p2;
                change(p[i],p[(i+1)%n],p1,p2,mid);
                line[i]=Line(p1,p2);
        }
        int resn;
        HPI(line,n,pp,resn); //pp中间变量,存生成的半平面交
        if(resn == 0)r=mid-eps;
        else {
                ans=mid;l=mid+eps;
        }
}
printf("%.6f\n",ans);
```

1.3.3 3 半平面交判范围

题意:在10*10的范围,你每次站在一个点,会告诉你距离目标点变近了还是远了,每次输出可能的范围.

思路:对于当前p[i]和前一次p[i-1],他的回答等于告诉你在p[i] - >
p[i-1]这条线的中垂线的哪一边,先表示出这根中垂线,定义它的左边表示包含部分,然后用远近调整这条线的方向(经过Mid点),加入

```
int sgn(double x) {
        if(fabs(x) < eps) return 0; return x<0?-1:1;}
```

```
struct Point {
	double x,y;Point() {}
	Point(double _x,double _y) {x = _x;y = _y;}
	Point operator +(const Point &b)const {return Point(x+b.x,y+b.y);}
	Point operator -(const Point &b)const {return Point(x - b.x, y - b.y);}
	double operator ^(const Point &b)const {return x*b.y - y*b.x;}
	double operator *(const Point &b)const {return x*b.x + y*b.y;}
};
double Cross(Point a,Point b) {return a.x*b.y-a.y*b.x;}
typedef Point Vector;double ss;
struct Line {
	Point s,e;double k;Line() {}
	Line(Point _s,Point _e) {s = _s;e = _e;k = atan2(e.y - s.y,e.x - s.x);}
	Point operator &(const Line &b)const {
		Point res = s;
		double t = ((s - b.s)^(b.s - b.e))/((s - e)^(b.s - b.e));
		res.x += (e.x - s.x)*t;
		res.y += (e.y - s.y)*t;
		return res;
	}
};
//半平面交，直线的左边代表有效区域
bool HPIcmp(Line a,Line b) {
	if(fabs(a.k - b.k) > eps)return a.k < b.k;
	return ((a.s - b.s)^(b.e - b.s)) < 0;
}
Line Q[1010];
void HPI(Line line[], int n, Point res[], int &resn) {
	int tot = n;sort(line,line+n,HPIcmp);tot = 1;
	for(int i = 1; i < n; i++)
		if(fabs(line[i].k - line[i-1].k) > eps)
			line[tot++] = line[i];
	int head = 0, tail = 1;
	Q[0] = line[0];Q[1] = line[1];resn = 0;
	for(int i = 2; i < tot; i++) {
		if(fabs((Q[tail].e-Q[tail].s)^(Q[tail-1].e-Q[tail-1].s)) < eps ||
		    fabs((Q[head].e-Q[head].s)^(Q[head+1].e-Q[head+1].s)) < eps)
			return;
		while(head<tail && (((Q[tail]&Q[tail-1])-line[i].s)^(line[i].e-line[i].s)) > eps)
			tail--;
		while(head<tail && (((Q[head]&Q[head+1])-line[i].s)^(line[i].e-line[i].s)) > eps)
			head++;
		Q[++tail] = line[i];
	}
	while(head < tail && (((Q[tail]&Q[tail-1]) - Q[head].s)^(Q[head].e-Q[head].s)) > eps)
		tail--;
```

```
        while(head < tail && (((Q[head]&Q[head-1]) - Q[tail].s)^(Q[tail].e-Q[tail].e)) > eps)
                head++;
        if(tail <= head + 1)return;
        for(int i = head; i < tail; i++)
                res[resn++] = Q[i]&Q[i+1];
        if(head < tail - 1)
                res[resn++] = Q[head]&Q[tail];
        ss=0;
        for(int i=0; i<resn; i++) {
                int j=(i+1)%resn;
                ss += res[i]^res[j];
        }ss/=2;
}
Point p[1010];Line line[1010];
int n,cnt;char op[10];
void init() {n=0;
        line[n++]=Line(Point(0,0),Point(10,0));line[n++]=Line(Point(10,0),Point(10,10));
        line[n++]=Line(Point(10,10),Point(0,10));line[n++]=Line(Point(0,10),Point(0,0));}
Point getMid(Point a,Point b) {
        Point res;res.x = (a.x+b.x)/2;res.y = (a.y+b.y)/2;return res;
}
Point pp[N];
void work() {
        init();cnt=0;int flag=0;
        p[0].x=p[0].y=0;
        while(~scanf("%lf%lf%s",&p[cnt].x,&p[cnt].y,op)) {
                if(op[0]=='S')flag=1;
                Point v=p[cnt]-p[cnt-1];
                v=Point(v.y,-v.x);
                Point Mid = getMid(p[cnt],p[cnt-1]);
                if(sgn(Cross(v,p[cnt-1]-Mid))<0)v=Point(-v.x,-v.y); //p[cnt-1]在直线左边
                if(op[0] == 'H')v=Point(-v.x,-v.y);
                line[n++]=Line(Mid,Mid+v);
                if(flag)printf("0.00\n");
                else {//计算半平面交
                        int resn;ss=0;
                        HPI(line,n,pp,resn);printf("%.2f\n",ss);
                }
        }
```

1.4 三角形

1.4.1 三角形四心

```
const int N = 1e5 + 50;
```

```
const ld PI = acos(-1.0);
const ld eps=1e-8;
int sgn(ld x) {
	if(fabs(x)<eps)return 0;
	return x<0?-1:1;
}
struct Point {
	ld x,y;
	Point() {}
	Point(ld x,ld y):x(x),y(y) {}
	Point operator +(Point B) {return Point(x+B.x,y+B.y);}
	Point operator -(Point B) {return Point(x-B.x,y-B.y);}
	Point operator *(double k) {return Point(x*k,y*k);}
	Point operator /(double k) {return Point(x/k,y/k);}
	bool operator ==(Point B) {return sgn(x-B.x)==0&&sgn(y-B.y)==0;}
};

struct Line {
	Point p1,p2;
	Line() {}
	Line(Point p1,Point p2):p1(p1),p2(p2) {}

};
typedef Point Vector;


/*************三角形的四心***********/
//外心,中垂线交点,外接圆
Point triangle_waixin(Point a,Point b,Point c) {
	Point s;
	ld a1=2*(b.x-a.x),b1=2*(b.y-a.y),a2=2*(c.x-a.x),b2=2*(c.y-a.y);
	ld c1=b.x*b.x-a.x*a.x+b.y*b.y-a.y*a.y;
	ld c2=c.x*c.x-a.x*a.x+c.y*c.y-a.y*a.y;
	s.x=(b1*c2-b2*c1)/(a2*b1-a1*b2);
	s.y=(a2*c1-a1*c2)/(a2*b1-a1*b2);
	return s;
}
//内心,角平分线交点,内接圆
ld Dis(Point a,Point b) {
	return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
Point triangle_neixin(Point a,Point b,Point c) {
	ld A = Dis(b,c),B=Dis(a,c),C=Dis(a,b),S = A+B+C;
	ld x = (A*a.x+B*b.x+C*c.x)/S;
	ld y = (A*a.y+B*b.y+C*c.y)/S;
	return Point(x,y);
```

```
}

//重心,中线交点
Point triangle_zhongxin(Point a,Point b,Point c) {
	return Point((a.x+b.x+c.x)/3.0,(a.y+b.y+c.y)/3.0);
}

//垂心
Point triangle_chuixinchuixin(Point a,Point b,Point c) {
	return triangle_zhongxin(a,b,c)*3.0-triangle_waixin(a,b,c)*2.0;
}
/*******************************************************/

//海伦公式求面积
ld triangle_hailun(Point a,Point b,Point c) {
	ld A = Dis(b,c),B=Dis(a,c),C=Dis(a,b),p=(A+B+C)/2.0;
	return sqrt(p*(p-A)*(p-B)*(p-C));
}
```

1.5 球

1.5.1 1 球面经纬最短路

```
const int N = 100 + 50,M = 1e5 + 50;
const double R = 6378.0;
const double PI = acos(-1);

int n,m,q;
struct Point {
	string id;
	double x,y,z;
	Point(double _x=0,double _y=0,double _z=0) {
		x=_x,y=_y,z=_z;
	}
} p[N];
map<string,int>mp;
int g[N][N];

//球面两点间的弧长
int distance(int i,int j) {
	double res = (p[i].x-p[j].x)*(p[i].x-p[j].x)+
		(p[i].y-p[j].y)*(p[i].y-p[j].y)+
		(p[i].z-p[j].z)*(p[i].z-p[j].z);
	double angle=acos((2*R*R-res)/(2*R*R));
	return (int)(angle*R+0.5); //注意
```

```
}

void floyd() {
	for(int k=1; k<=n; k++) {
		for(int i=1; i<=n; i++) {
			for(int j=1; j<=n; j++) {
				if(g[i][k] + g[k][j] < g[i][j]) {
					g[i][j] = g[i][k] + g[k][j];
				}
			}
		}
	}
}

void work() {
	int cnt=1;
	while(cin >> n >> m >> q && n) {
		if(cnt != 1)printf("\n");
		printf("Case #%d\n",cnt);
		cnt++;
		for(int i=1; i<=n; i++) {
			for(int j=1; j<=n; j++) {
				g[i][j] = 100000000;
			}
		}
		for(int i=1; i<=n; i++) {
			cin >> p[i].id,mp[p[i].id] = i;
			double a,b,cin >>a >> b;
			a = a*PI/180.0,b = b*PI/180.0;
			p[i].x = R*cos(a)*sin(b);
			p[i].y = R*cos(a)*cos(b);
			p[i].z = sin(a)*R;
		}
		while(m--) {
			string a,b;
			cin >> a >> b;
			int x = mp[a],y = mp[b]; //地名
			g[x][y] = distance(x,y);
		}
		floyd();
		for(int cc=1; cc<=q; cc++) {
			string a,b;
			cin >> a >> b;
			int x = mp[a],y = mp[b];
			if(g[x][y] == 100000000)printf("no route exists\n");
			else {
```

```
                        printf("%d km\n",g[x][y]);
                    }
                }

        }
}

//给一个经度和维度,先把x -> torad(x),然后angle_3d()算出两点间的弧度制距离.
double torad(double x){
        return x*pi/180;
}
double angle_3d(double lng1, double lat1, double lng2, double lat2) {
    //经度，纬度，经度，纬度
    return acos(cos(lat1)*cos(lat2)*cos(lng1 - lng2) + sin(lat1)*sin(lat2));
}
```

1.5.2 2 最小球覆盖

```
struct Point {
        double x,y,z;
} p[N];
double dist(Point a,Point b) {
        return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y)+(a.z-b.z)*(a.z-b.z));
}
int n;
void work() { //输出最小球的半径
        while(scanf("%d",&n) && n) {
                for(int i=1; i<=n; i++) {
                        scanf("%lf%lf%lf",&p[i].x,&p[i].y,&p[i].z);
                }
                double ans = 1e9;
                Point tmp= {0,0,0};
                double step = 100;
                int s = 1;
                while(step > eps) {
                        for(int i=1; i<=n; i++) {//找到距离tmp最远的点
                                if(dist(tmp,p[s]) < dist(tmp,p[i]))s=i;
                        }
                        double d = dist(tmp,p[s]);
                        ans = min(ans,d);
                        tmp.x+=(p[s].x-tmp.x)/d*step;
                        tmp.y+=(p[s].y-tmp.y)/d*step;
                        tmp.z+=(p[s].z-tmp.z)/d*step;
                        step*=0.99;
                }
```

```
                printf("%.5f\n",ans);
        }
}
```

1.6 扫描线

1.6.1 1 矩形面积交

```
/* Luogu5490
N开两倍,一个矩形两根线,线段树开8倍!
输入左下角和右上角坐标,输出矩形面积交
*/
const int N = 2e5 + 50; //注意N的大小,会RE

int n;
int v[N];

struct node{
        int l,r;
        int cover;
        ll len;
}tr[N<<3];

void pushup(int u){
        if(tr[u].cover)tr[u].len=tr[u].r-tr[u].l;
        else tr[u].len=tr[u<<1].len+tr[u<<1|1].len;
}

//数组记得全从1开始
struct L{
        int x;
        int y1,y2; //y1<y2;
        int state; //左边1,右边-1;
        bool operator<(L B){
                return x < B.x;
        }
}line[N];

void build(int u,int l,int r){
        tr[u].l=v[l],tr[u].r=v[r];
        if(r-l<=1)return ;
        int mid = (l + r) >> 1;
        build(u<<1,l,mid);
        build(u<<1|1,mid,r);
}
```

```
void modify(int u,int x,int y,int v){
	int l=tr[u].l,r=tr[u].r;
	if(x<=l&&y>=r){
		tr[u].cover += v;
		pushup(u);
		return ;
	}
	if(x<tr[u<<1].r)modify(u<<1,x,y,v);
	if(y>tr[u<<1|1].l)modify(u<<1|1,x,y,v);
	pushup(u);
}

void work(){
	scanf("%d",&n);
	for(int i=1;i<=n;i++){
		int a,b,c,d;
		scanf("%d%d%d%d",&a,&b,&c,&d);
		v[i]=b,v[n+i]=d; //y方向线段,建线段树
		line[i]=(L){a,b,d,1},line[i+n]=(L){c,b,d,-1};
	}
	sort(v+1,v+1+2*n);
	sort(line+1,line+1+2*n);
	build(1,1,2*n);

	ll ans = 0;
	for(int i=1;i<=2*n;i++){
		ans += tr[1].len*(line[i].x-line[i-1].x);
		modify(1,line[i].y1,line[i].y2,line[i].state);
	}
	printf("%lld\n",ans);
}
```

1.7 多边形

1.7.1 多边形

求多边形重心 Hdu1115

```
const int N = 1e6 + 50;
const double eps = 1e-8;

//判断x是否为0;是的话return 0
int sgn(double x){
	if(fabs(x) < eps)return 0;
```

```
        else return x<0?-1:1;
}


struct Point {
        double x,y;
        Point(double X=0,double Y =0) {
                x = X,y = Y;
        }
        Point operator+ (Point B) {
                return Point(x + B.x,y+B.y);
        }
        Point operator- (Point B) {
                return Point(x - B.x,y-B.y);
        }
        Point operator* (double k) {
                return Point(x*k,y*k);
        }
        Point operator/(double k) {
                return Point(x/k,y/k);
        }
};
typedef Point Vector;

double Cross(Vector A,Vector B){
        return A.x*B.y-A.y*B.x;
}


Point p[N];
Point center;


double Polygon_area(Point *p,int n){
        double res = 0;
        for(int i=0;i<n;i++){
                res += Cross(p[i],p[(i+1)%n]);
        }
        return res/2;
}


Point Polygon_center(Point *p,int n){
        Point ans(0,0);
        double s = Polygon_area(p,n);
        if(sgn(s) == 0)return ans; //面积等于0
```

```
        for(int i=0;i<n;i++){
                ans = ans + (p[i]+p[(i+1)%n]) * (Cross(p[i],p[(i+1)%n]));
        }
        return ans / s / 6.0;
}

int n;

//多边形的重心
void work() {
        scanf("%d",&n);
        for(int i=0;i<n;i++){
                scanf("%lf %lf",&p[i].x,&p[i].y);
        }
        center = Polygon_center(p,n);
        printf("%.2f %.2f\n",center.x,center.y);
}
```

1.8 圆

1.8.1 1 圆外一点到圆的切点

```
//点到圆的切点

int sgn(double x) {
        if(fabs(x)<eps)return 0;
        return x<0?-1:1;
}

struct Point { //定义点和基本运算
        double x,y;
        double ang;
        Point() {}
        Point(double x,double y):x(x),y(y) {}
        Point operator + (Point B) {
                return Point(x+B.x,y+B.y);
        }
        Point operator - (Point B) {
                return Point(x-B.x,y-B.y);
        }
        Point operator * (double k) {
                return Point(x*k,y*k); //长度增大k倍
        }
        Point operator / (double k) {
                return Point(x/k,y/k); //长度缩小k倍
```

```
        }
        bool operator == (Point B) {
                return sgn(x-B.x)==0 && sgn(y-B.y)==0;
        }
        double operator ^(Point B) {
                return x*B.y-y*B.x;
        }
        double distance(Point p) {
                return hypot(x-p.x,y-p.y);
        }
};

typedef Point Vector;
double Cross(Vector A,Vector B) {
        return A.x*B.y - A.y*B.x; //叉积
}

struct Line {
        Point p1,p2;//线上的两个点
        Line() {}
        Line(Point p1,Point p2):p1(p1),p2(p2) {}
};

struct Circle {
        Point c;//圆心
        double r;//半径
        Circle() {}
        Circle(Point c,double r):c(c),r(r) {}
        Circle(double x,double y,double _r) {
                c=Point(x,y);
                r = _r;
        }
};
/******************点到圆的切点***************/
double Distance(Point A, Point B) {
        return hypot(A.x-B.x,A.y-B.y);
}
Point rotate(Point base,Point a,double r) { //旋转
        Point b=a-base;
        a.x=b.x*cos(r)-b.y*sin(r);
        a.y=b.x*sin(r)+b.y*cos(r);
        a=a+base;
        return a;
}
void tangent_points(Circle c,Point p,Point &p1,Point &p2) {
        Vector v=p-c.c;
```

```
        double d = Distance(p,c.c);
        double r=acos(c.r/d);
        v=v*c.r/d;
        p1=rotate(c.c,c.c+v,r);
        p2=rotate(c.c,c.c+v,-r);
}/*********************************************/

void work() {
        Circle a;
        Point p;
        scanf("%lf%lf",&p.x,&p.y);
        scanf("%lf%lf%lf",&a.c.x,&a.c.y,&a.r);
        Point p1,p2;
        tangent_points(a,p,p1,p2);
        if(sgn(p1.x-p2.x)>0)swap(p1,p2);
        else if(sgn(p1.x-p2.x)==0 && sgn(p1.y-p2.y)>0)swap(p1,p2);

        printf("%lf %lf\n",p1.x,p1.y);
        printf("%lf %lf\n",p2.x,p2.y);
}
```

1.8.2 2 两个圆的切线与切点

两个圆的切线与切点

```
int sgn(double x) {
        if(fabs(x)<eps)return 0;
        return x<0?-1:1;
}
struct Point { //定义点和基本运算
        double x,y;
        double ang;
        Point() {}
        Point(double x,double y):x(x),y(y) {}
        Point operator + (Point B) {
                return Point(x+B.x,y+B.y);
        }
        Point operator - (Point B) {
                return Point(x-B.x,y-B.y);
        }
        Point operator * (double k) {
                return Point(x*k,y*k); //长度增大k倍
        }
        Point operator / (double k) {
                return Point(x/k,y/k); //长度缩小k倍
```

```
        }
        bool operator == (Point B) {
                return sgn(x-B.x)==0 && sgn(y-B.y)==0;
        }
        double operator ^(Point B) {
                return x*B.y-y*B.x;
        }
        double distance(Point p) {
                return hypot(x-p.x,y-p.y);
        }
};

typedef Point Vector;
double Cross(Vector A,Vector B) {
        return A.x*B.y - A.y*B.x; //叉积
}

struct Line {
        Point p1,p2;//线上的两个点
        Line() {}
        Line(Point p1,Point p2):p1(p1),p2(p2) {}
};

struct Circle {
        Point c;//圆心
        double r;//半径
        Circle() {}
        Circle(Point c,double r):c(c),r(r) {}
        Circle(double x,double y,double _r) {
                c=Point(x,y);
                r = _r;
        }
        Point point(double ang) { //圆上与圆心极坐标为ang的点-----------------
                return Point(c.x+cos(ang)*r,c.y+sin(ang)*r);
        }
};
//0,-1:没有切线 a[]是c1上的切点,b[]是c2
int getTangents(Circle A, Circle B, Point *a, Point *b) {
        int cnt = 0;      //存切点用
        if(sgn(A.r - B.r) < 0) {
                swap(A, B);
                swap(a, b);
        }
        double d = sqrt((A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) * (A.c.y - B.c.y));
            //圆心距
        double rdiff = A.r - B.r;  //两圆半径差
```

```
        double rsum = A.r + B.r;   //两圆半径和
        if(sgn(d - rdiff) < 0) return 0;  //1.内含
        double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x); //向量AB的极角
        if(sgn(d) == 0) return -1;   //2.重合
        if(sgn(d - rdiff) == 0) {  //3.内切
                a[cnt] = b[cnt] = A.point(base);
                cnt++;
                return 1;
        }
        double ang = acos((A.r - B.r) / d);
        a[cnt] = A.point(base + ang);
        b[cnt] = B.point(base + ang);
        cnt++;    //4.相交（外切、外离的外公切线也在此求出）
        a[cnt] = A.point(base - ang);
        b[cnt] = B.point(base - ang);
        cnt++;    //两条外公切线的切点
        if(sgn(d - rsum) == 0) {   //5.外切
                a[cnt] = b[cnt] = A.point(base);
                cnt++;
        } else if(sgn(d - rsum) > 0) { //6.外离
                double ang = acos((A.r + B.r) / d);
                a[cnt] = A.point(base + ang);
                b[cnt] = B.point(PI + base + ang);
                cnt++;
                a[cnt] = A.point(base - ang);
                b[cnt] = B.point(PI + base - ang);
                cnt++;
        }
        return cnt;

}

bool cmp(Point a,Point b) {
        if(sgn(a.x-b.x)!=0)return a.x < b.x;
        else if(sgn(a.x-b.x)==0)return a.y < b.y;
}

void work() {
        Circle a,b;
        scanf("%lf%lf%lf",&a.c.x,&a.c.y,&a.r);
        scanf("%lf%lf%lf",&b.c.x,&b.c.y,&b.r);
        Point p1[5],p2[5];
        int cnt = getTangents(a,b,p1,p2);
//      cout << cnt << endl;
        if(cnt == -1 ||cnt==0)return ;
        vector<Point>ans;
```

```
        for(int i=0; i<cnt; i++) {
                ans.push_back(p1[i]);
        }
        sort(ans.begin(),ans.end(),cmp);
        for(int i=0; i<cnt; i++) {
                printf("%.10f %.10f\n",ans[i].x,ans[i].y);
        }

}
```

1.8.3 3 两圆交点

```
//两圆交点

int sgn(double x) {
        if(fabs(x)<eps)return 0;
        return x<0?-1:1;
}

struct Point { //定义点和基本运算
        double x,y;
        double ang;
        Point() {}
        Point(double x,double y):x(x),y(y) {}
        Point operator + (Point B) {
                return Point(x+B.x,y+B.y);
        }
        Point operator - (Point B) {
                return Point(x-B.x,y-B.y);
        }
        Point operator * (double k) {
                return Point(x*k,y*k); //长度增大k倍
        }
        Point operator / (double k) {
                return Point(x/k,y/k); //长度缩小k倍
        }
        bool operator == (Point B) {
                return sgn(x-B.x)==0 && sgn(y-B.y)==0;
        }
        double operator ^(Point B) {
                return x*B.y-y*B.x;
        }
        double distance(Point p) {
                return hypot(x-p.x,y-p.y);
        }
```

```
};

typedef Point Vector;
double Cross(Vector A,Vector B) {
	return A.x*B.y - A.y*B.x; //叉积
}

struct Line {
	Point p1,p2;//线上的两个点
	Line() {}
	Line(Point p1,Point p2):p1(p1),p2(p2) {}
};

struct Circle {
	Point c;//圆心
	double r;//半径
	Circle() {}
	Circle(Point c,double r):c(c),r(r) {}
	Circle(double x,double y,double _r) {
		c=Point(x,y);
		r = _r;
	}
	Point point(double ang) { //圆上与圆心极坐标为ang的点
		return Point(c.x+cos(ang)*r,c.y+sin(ang)*r);
	}
};

double Distance(Point A, Point B) {
	return hypot(A.x-B.x,A.y-B.y);
}

//求向量v的极角
double angle(Vector v) {
	return atan2(v.y,v.x);
}

int getcirclecirclePoint(Circle c1,Circle c2,Point &p1,Point &p2) {
	double d = Distance(c1.c,c2.c);
	if(sgn(d)== 0) {
		if(sgn(c1.r-c2.r)==0)return -1; //重合
		return 0; //没有交点
	}
	if(sgn(c1.r+c2.r-d)<0)return 0; //相离
	if(sgn(fabs(c1.r-c2.r)-d)>0)return 0; //内含
	double a =angle(c2.c-c1.c); //c1->c2的极角
	double da=acos((c1.r*c1.r+d*d-c2.r*c2.r)/(2*c1.r*d));
```

```
        p1=c1.point(a-da),p2=c1.point(a+da);
        if(p1==p2)return 1;
        else return 2;
}


void work() {
        Circle a,b;
        scanf("%lf%lf%lf",&a.c.x,&a.c.y,&a.r);
        scanf("%lf%lf%lf",&b.c.x,&b.c.y,&b.r);
        Point p1,p2;
        int t= getcirclecirclePoint(a,b,p1,p2); //交点个数.-1重合,0,1,2
        if(sgn(p1.x-p2.x)>0)swap(p1,p2);
        else if(sgn(p1.x-p2.x)==0&&sgn(p1.y-p2.y)>0)swap(p1,p2);

        printf("%lf %lf ",p1.x,p1.y);
        printf("%lf %lf\n",p2.x,p2.y);

}
```

1.8.4 4 圆与多边形的面积交

```
//圆与多边形的面积交

int sgn(double x) {
        if(fabs(x)<eps)return 0;
        return x<0?-1:1;
}
struct Point { //定义点和基本运算
        double x,y;
        double ang;
        Point() {}
        Point(double x,double y):x(x),y(y) {}
        Point operator + (Point B) {return Point(x+B.x,y+B.y);}
        Point operator - (Point B) {return Point(x-B.x,y-B.y);}
        Point operator * (double k) {return Point(x*k,y*k); //长度增大k倍}
        Point operator / (double k) {return Point(x/k,y/k); //长度缩小k倍}
        bool operator == (Point B) {return sgn(x-B.x)==0 && sgn(y-B.y)==0;}
        double operator ^(Point B) {return x*B.y-y*B.x;}
        double distance(Point p) {return hypot(x-p.x,y-p.y);}
        double len() {return hypot(x,y);//库函数}

        Point trunc(double r) {
                double l = len();
                if(!sgn(l))return *this;
```

```
            r /= l;
            return Point(x*r,y*r);
        }//点积
        double operator *(const Point &b)const {return x*b.x + y*b.y;}
        double len2() {return x*x + y*y;}
        double rad(Point a,Point b) {
            Point p = *this;
            return fabs(atan2( fabs((a-p)^(b-p)),(a-p)*(b-p) ));
        }
};

typedef Point Vector;
double Cross(Vector A,Vector B) {
        return A.x*B.y - A.y*B.x; //叉积
}

struct Line {
        //返回长度的平方
        Point s,e;
        Line() {}
        Line(Point _s,Point _e) {
            s = _s;
            e = _e;
        }
        double length() {
            return s.distance(e);
        }
        double dispointtoline(Point p) {
            return fabs((p-s)^(e-s))/length();
        }
        Point lineprog(Point p) {
            return s + ( ((e-s)*((e-s)*(p-s)))/((e-s).len2()) );
        }
};

struct Circle {
        Point p;//圆心
        double r;//半径
        Circle() {}
        Circle(Point p,double r):p(p),r(r) {}
        Circle(double x,double y,double _r) {
            p=Point(x,y);
            r = _r;
        }
        int relationcircle(Circle v) {
            double d = p.distance(v.p);
```

```
            if(sgn(d-r-v.r) > 0)return 5;
            if(sgn(d-r-v.r) == 0)return 4;
            double l = fabs(r-v.r);
            if(sgn(d-r-v.r)<0 && sgn(d-l)>0)return 3;
            if(sgn(d-l)==0)return 2;
            if(sgn(d-l)<0)return 1;
        }

        int relationline(Line v) {
            double dst = v.dispointtoline(p);
            if(sgn(dst-r) < 0)return 2;
            else if(sgn(dst-r) == 0)return 1;
            return 0;
        }
        //点和圆的关系
//0 圆外
//1 圆上
//2 圆内
        int relation(Point b) {
            double dst = b.distance(p);
            if(sgn(dst-r) < 0)return 2;
            else if(sgn(dst-r)==0)return 1;
            return 0;
        }

        //求直线和圆的交点，返回交点个数
        int pointcrossline(Line v,Point &p1,Point &p2) {
            if(!(*this).relationline(v))return 0;
            Point a = v.lineprog(p);
            double d = v.dispointtoline(p);
            d = sqrt(r*r-d*d);
            if(sgn(d) == 0) {
                p1 = a;
                p2 = a;
                return 1;
            }
            p1 = a + (v.e-v.s).trunc(d);
            p2 = a - (v.e-v.s).trunc(d);
            return 2;
        }

        double areatriangle(Point a,Point b) {
            if(sgn((p-a)^(p-b)) == 0)return 0.0;
            Point q[5];
            int len = 0;
            q[len++] = a;
```

```
                Line l(a,b);
                Point p1,p2;
                if(pointcrossline(l,q[1],q[2])==2) {
                        if(sgn((a-q[1])*(b-q[1]))<0)q[len++] = q[1];
                        if(sgn((a-q[2])*(b-q[2]))<0)q[len++] = q[2];
                }
                q[len++] = b;
                if(len == 4 && sgn((q[0]-q[1])*(q[2]-q[1]))>0)swap(q[1],q[2]);
                double res = 0;
                for(int i = 0; i < len-1; i++) {
                        if(relation(q[i])==0||relation(q[i+1])==0) {
                                double arg = p.rad(q[i],q[i+1]);
                                res += r*r*arg/2.0;
                        }


                        else {
                                res += fabs((q[i]-p)^(q[i+1]-p))/2.0;
                        }
                }
                return res;
        }
};

struct polypon {
        int n;
        Point p[N];
        Line l[N];

        double areacircle(Circle c) {
                double ans = 0;
                for(int i = 0; i < n; i++) {
                        int j = (i+1)%n;
                        if(sgn( (p[j]-c.p)^(p[i]-c.p) ) >= 0)
                                ans += c.areatriangle(p[i],p[j]);
                        else ans -= c.areatriangle(p[i],p[j]);
                }
                return fabs(ans);
        }

};

polypon pp;
Circle c;

void work() {
```

```
        scanf("%d",&pp.n);
        scanf("%lf",&c.r);
        c.p=Point(0,0);
        for(int i=0; i<pp.n; i++) {
                scanf("%lf%lf",&pp.p[i].x,&pp.p[i].y);
        }
        double ans=pp.areacircle(c);
        printf("%.6f\n",ans);

}
```

1.8.5 5 圆面积交并

测试TLE...没办法了

```
const int N = 1e3 + 50;
typedef double ld;
/*****************************************/
#define MP make_pair
#define pb push_back
int sgn(ld x) {
        if(fabs(x) < eps)return 0;
        return x<0?-1:1;
}
struct Point {
        ld x,y;
        Point() {}
        Point(ld _x,ld _y) {
                x=_x,y=_y;
        }
        bool operator == (Point b)const {
                return sgn(x-b.x)==0&&sgn(y-b.y)==0;
        }

        Point operator -(Point b)const {
                return Point(x-b.x,y-b.y);
        }
        Point operator +(Point b) {
                return Point(x+b.x,y+b.y);
        }
        Point operator *(ld k) {
                return Point(x*k,y*k);
        }
        //叉积
        double operator ^(const Point &b)const {
```

```
        return x*b.y-y*b.x;
    }

    bool operator <(Point b)const {
        return sgn(x-b.x)==0?sgn(y-b.y)<0:x<b.x;
    }
    double distance(Point p) {
        return hypot(x-p.x,y-p.y);
    }
    ld len() {
        return hypot(x,y);
    }

};

struct circle {
    Point p;
    double r;
    circle() {}
    circle(Point _p,ld _r) {
        p=_p,r=_r;
    }

    bool operator == (circle v) {
        return (p==v.p) && sgn(r-v.r)==0;
    }

    //5相离 4外切 3相交 2内切 1内含
    int relationcircle(circle v) {
        ld d = p.distance(v.p);
        if(sgn(d-r-v.r)>0)return 5;
        if(sgn(d-r-v.r)==0)return 4;
        ld l = fabs(r-v.r);
        if(sgn(d-r-v.r)<0&&sgn(d-l)>0)return 3;
        if(sgn(d-l)==0)return 2;
        if(sgn(d-l)<0)return 1;
    }
};

struct circles { //多圆
    circle c[N];
    ld ans[N]; //ans[i]表示被覆盖i次的面积
    ld pre[N];
    int n;
    circles() {}
    void add(circle cc) {
```

```
            c[n++]=cc;
        }
        //半径为 r 的圆，弧度为 th 对应的弓形的面积
        double areaarc(double th,double r) {
            return 0.5*r*r*(th-sin(th));
        }

        bool inner(circle x,circle y) { //x包含在y中
            if(x.relationcircle(y)!=1)return 0;
            return sgn(x.r-y.r)<=0?1:0;
        }
        void init_or() { //面积并
            bool mark[N]= {0};
            int i,j,k=0;
            for(i=0; i<n; i++) {
                for(j=0; j<n; j++) {
                    if(i!=j &&!mark[j]) {
                        if(c[i]==c[j] || inner(c[i],c[j]))break;
                    }

                }
                if(j < n)mark[i]=1;
                for(i=0; i<n; i++)
                    if(!mark[i])c[k++] = c[i];
            }
            n=k;
        }
        void init_add() { //面积交用这个
            bool mark[N]= {0};
            int i,j,k=0;
            for(i=0; i<n; i++) {
                for(j=0; j<n; j++) {
                    if(i!=j &&!mark[j]) {
                        if(c[i]==c[j] || inner(c[j],c[i]))break;
                    }
                }
                if(j < n)mark[i]=1;
                for(i=0; i<n; i++)
                    if(!mark[i])c[k++] = c[i];
            }
            n=k;
        }

        void getarea() { //多圆面积并
//          memset(ans,0,sizeof(ans));
            vector<pair<ld,int> >v;
```

```
for(int i=0; i<n; i++) {
	v.clear();
	v.pb(MP(-PI,1));
	v.pb(MP(PI,-1));
	for(int j=0; j<n; j++) {
		if(i!=j) {
			Point q=(c[j].p-c[i].p);
			ld ab=q.len(),ac=c[i].r,bc=c[j].r;
			if(sgn(ab+ac-bc)<=0) {
				v.pb(MP(-PI,1));
				v.pb(MP(PI,-1));
				continue;
			}
			if(sgn(ab+bc-ac)<=0)continue;
			if(sgn(ab-ac-bc)>0)continue;
			ld th=atan2(q.y,q.x),fai=acos((ac*ac+ab*ab-bc*bc)/(2.0*ac*ab));
			ld a0 = th-fai;
			if(sgn(a0+PI)<0)a0+=2*PI;
			ld a1 = th+fai;
			if(sgn(a1-PI)>0)a1-=2*PI;
			if(sgn(a0-a1)>0) {
				v.pb(MP(a0,1));
				v.pb(MP(PI,-1));
				v.pb(MP(-PI,1));
				v.pb(MP(a1,-1));
			} else {
				v.pb(MP(a0,1));
				v.pb(MP(a1,-1));
			}
		}
	}
	sort(v.begin(),v.end());
	int cur=0;
	for(int j = 0; j < v.size(); j++) {
		if(cur && sgn(v[j].first-pre[cur])) {
			ans[cur] += areaarc(v[j].first-pre[cur],c[i].r);
			ans[cur] +=
			    0.5*(Point(c[i].p.x+c[i].r*cos(pre[cur]),c[i].p.y+c[i].r*sin(pre[cur]))
				^Point(c[i].p.x+c[i].r*cos(v[j].first),c[i].p.y+c[i].r*sin(v[j].fi
		}
		cur += v[j].second;
		pre[cur] = v[j].first;
	}
}
for(int i=1; i<n; i++) {
	ans[i] -= ans[i+1];
```

```
                }
        }
};

int n;
circle in[N];
circles C;

void work() {
        scanf("%d",&n);
        for(int i=0; i<n; i++) {
                scanf("%lf%lf%lf",&in[i].p.x,&in[i].p.y,&in[i].r);
                C.add(in[i]);
        }
        C.init_or(); //注意改这个
        C.getarea();
        ld sum=0;
        for(int i=1; i<=n; i++) {
                sum += C.ans[i];
        }
        printf("%.3f\n",sum+eps);
//      printf("%.3f\n",C.ans[n]); //覆盖n次的就是交
}
```

1.8.6 6 最小圆覆盖

```
//最小圆覆盖
typedef double ld;
const int N = 1e3 + 50,M = 1e5 + 50;
const double PI = acos(-1);
const double eps = 1e-8;

int sgn(double x) {
        if(fabs(x) < eps)return 0;
        else return x<0?-1:1;
};
struct Point {double x,y;};
double Distance(Point A,Point B) {return hypot(A.x-B.x,A.y-B.y);}

//三角形外接圆圆心
Point circle_center(const Point a,const Point b,const Point c) {
        Point center;
        double a1=b.x-a.x,b1=b.y-a.y,c1=(a1*a1+b1*b1)/2;
        double a2=c.x-a.x,b2=c.y-a.y,c2=(a2*a2+b2*b2)/2;
        double d = a1*b2-a2*b1;
```

```
        center.x = a.x+(c1*b2-c2*b1)/d;
        center.y = a.y+(a1*c2-a2*c1)/d;
        return center;
}

void min_cover_circle(Point *p,int n,Point &c,double &r) {
        random_shuffle(p,p+n);
        c=p[0];
        r=0;
        for(int i=1; i<n; i++) {
                if(sgn(Distance(p[i],c)-r)>0) { //p[i]在圆外
                        c = p[i];
                        r=0;
                        for(int j=0; j<i; j++) {
                                if(sgn(Distance(p[j],c)-r)>0) {
                                        c.x = (p[i].x + p[j].x) / 2;
                                        c.y = (p[i].y + p[j].y) / 2;
                                        r = Distance(p[j],c);
                                        for(int k=0; k<j; k++) {
                                                if(sgn(Distance(p[k],c)-r ) > 0) {
                                                        c = circle_center(p[i],p[j],p[k]);
                                                        r=Distance(p[i],c);
                                                }
                                        }
                                }
                        }
                }
        }
}
int n;
Point p[N];
Point c;
double r;

void work() {
        while(~scanf("%d",&n) && n) {
                for(int i=0; i<n; i++) {
                        scanf("%lf%lf",&p[i].x,&p[i].y);
         }
                min_cover_circle(p,n,c,r);
                printf("%.2f %.2f %.2f\n",c.x,c.y,r);
        }
}
```

1.8.7 7k 次圆覆盖

求被覆盖k次的圆的面积

```
const int N = 1e3 + 50;
typedef double ld;
/*****************************************/
#define MP make_pair
#define pb push_back
int sgn(ld x) {
	if(fabs(x) < eps)return 0;
	return x<0?-1:1;
}

struct Point {
	ld x,y;
	Point() {}
	Point(ld _x,ld _y) {
		x=_x,y=_y;
	}
	bool operator == (Point b)const {return sgn(x-b.x)==0&&sgn(y-b.y)==0;}
	Point operator -(Point b)const {return Point(x-b.x,y-b.y);}
	Point operator +(Point b) {return Point(x+b.x,y+b.y);}
	Point operator *(ld k) {return Point(x*k,y*k);}
	//叉积
	double operator ^(const Point &b)const {return x*b.y-y*b.x;}
	//点积
	bool operator <(Point b)const {
		return sgn(x-b.x)==0?sgn(y-b.y)<0:x<b.x;
	}
	double distance(Point p) {return hypot(x-p.x,y-p.y);}
	ld len() {return hypot(x,y);}
};

struct circle {
	Point p;
	double r;
	circle() {}
	circle(Point _p,ld _r) {
		p=_p,r=_r;
	}
	//5相离 4外切 3相交 2内切 1内含
	int relationcircle(circle v) {
		ld d = p.distance(v.p);
		if(sgn(d-r-v.r)>0)return 5;
		if(sgn(d-r-v.r)==0)return 4;
```

```
            ld l = fabs(r-v.r);
            if(sgn(d-r-v.r)<0&&sgn(d-l)>0)return 3;
            if(sgn(d-l)==0)return 2;
            if(sgn(d-l)<0)return 1;
        }
};

struct circles { //多圆
        circle c[N];
        ld ans[N]; //ans[i]表示被覆盖i次的面积
        ld pre[N];
        int n;
        circles() {}
        void add(circle cc) {
                c[n++]=cc;
        }
        //半径为 r 的圆，弧度为 th 对应的弓形的面积
        double areaarc(double th,double r) {
                return 0.5*r*r*(th-sin(th));
        }

        bool inner(circle x,circle y) { //x包含在y中
                if(x.relationcircle((y))!=1)return 0;
                return sgn(x.r-y.r)<=0?1:0;
        }
//      void init_or() {
//              bool mark[N]= {0};
//              int i,j,k=0;
//              for(i=0; i<n; i++) {
//                      for(j=0; j<n; j+) {
//                              if(i!=j &&!mark[j]) {
//                                      if(c[i]==c[j] || inner(c[i],c[j]))break;
//                              }
//                              if(j < n)mark[i]=1;
//                      }
//                      for(i=0; i<n; i++)
//                              if(!mark[i])c[k++] = c[i];
//              }
//              n=k;
//      }
        void getarea() { //多圆面积并
                memset(ans,0,sizeof(ans));
                vector<pair<ld,int> >v;
                for(int i=0; i<n; i++) {
                        v.clear();
                        v.pb(MP(-PI,1));
```

```
                v.pb(MP(PI,-1));
                for(int j=0; j<n; j++) {
                        if(i!=j) {
                                Point q=(c[j].p-c[i].p);
                                ld ab=q.len(),ac=c[i].r,bc=c[j].r;
                                if(sgn(ab+ac-bc)<=0) {
                                        v.pb(MP(-PI,1));
                                        v.pb(MP(PI,-1));
                                        continue;
                                }
                                if(sgn(ab+bc-ac)<=0)continue;
                                if(sgn(ab-ac-bc)>0)continue;
                                ld th=atan2(q.y,q.x),fai=acos((ac*ac+ab*ab-bc*bc)/(2.0*ac*ab));
                                ld a0 = th-fai;
                                if(sgn(a0+PI)<0)a0+=2*PI;
                                ld a1 = th+fai;
                                if(sgn(a1-PI)>0)a1-=2*PI;
                                if(sgn(a0-a1)>0) {
                                        v.pb(MP(a0,1));
                                        v.pb(MP(PI,-1));
                                        v.pb(MP(-PI,1));
                                        v.pb(MP(a1,-1));
                                } else {
                                        v.pb(MP(a0,1));
                                        v.pb(MP(a1,-1));
                                }
                        }
                }
                sort(v.begin(),v.end());
                int cur=0;
                for(int j = 0; j < v.size(); j++) {
                        if(cur && sgn(v[j].first-pre[cur])) {
                                ans[cur] += areaarc(v[j].first-pre[cur],c[i].r);
                                ans[cur] +=
                                    0.5*(Point(c[i].p.x+c[i].r*cos(pre[cur]),c[i].p.y+c[i].r*sin(pre[cur]))
                                              ^Point(c[i].p.x+c[i].r*cos(v[j].first),c[i].p.y+c[i].r*sin(v[j].fi
                        }
                        cur += v[j].second;
                        pre[cur] = v[j].first;
                }
        }
        for(int i=1; i<n; i++) {
                ans[i] -= ans[i+1];
        }
    }
};
```

```
int n;
circle in[N];
circles C;

void work() {
	scanf("%d",&n);
	for(int i=0; i<n; i++) {
		scanf("%lf%lf%lf",&in[i].p.x,&in[i].p.y,&in[i].r);
		C.add(in[i]);
	}
	C.getarea();
	for(int i=1; i<=n; i++) {
		printf("[%d] = %.3f\n",i,C.ans[i]);
	}

}
```

1.8.8 8 圆并

```
#include<complex> //.norn
using namespace std;
const double EPS=1e-9,PI=acos(-1.0);

int cmp(double k) {
	return k<-EPS ? -1:k>EPS ? 1:0;
}

inline double sqr(double x) {
	return x*x;
}

struct point {
	double x,y;
	point () {}
	point (double a,double b):x(a),y(b) {}
	bool input() {
		return scanf("%lf%lf",&x,&y)!=EOF;
	}
	friend point operator +(const point &a,const point &b) {
		return point(a.x+b.x,a.y+b.y);
	}
	friend point operator -(const point &a,const point &b) {
		return point(a.x-b.x,a.y-b.y);
```

```
        }
        friend bool operator ==(const point &a,const point &b) {
                return cmp(a.x-b.x)==0&&cmp(a.y-b.y)==0;
        }
        friend point operator *(const point &a,const double &b) {
                return point(a.x*b,a.y*b);
        }
        friend point operator*(const double &a,const point &b) {
                return point(a*b.x,a*b.y);
        }
        friend point operator /(const point &a,const double &b) {
                return point(a.x/b,a.y/b);
        }
        double norm() {
                return sqrt(sqr(x)+sqr(y));
        }
};


double cross(const point &a,const point &b) {
        return a.x*b.y-a.y*b.x;
}

struct Circle {
        point p;
        double r;
        bool operator <(const Circle &o)const {
                if(cmp(r-o.r)!=0)return cmp(r-o.r)==-1;
                if(cmp(p.x-o.p.x)!=0)return cmp(p.x-o.p.x)==-1;
                return cmp(p.y-o.p.y)==-1;
        }
        bool operator ==(const Circle &o)const {
                return cmp(r-o.r)==0&&cmp(p.x-o.p.x)==0&&cmp(p.y-o.p.y)==0;
        }
};

point rotate(const point &p,double cost,double sint) {
        double x=p.x,y=p.y;
        return point(x*cost-y*sint,x*sint+y*cost);
}

pair<point,point> crosspoint(point ap,double ar,point bp,double br) {
        double d=(ap-bp).norm();
        double cost=(ar*ar+d*d-br*br)/(2*ar*d);
        double sint=sqrt(1.-cost*cost);
        point v=(bp-ap)/(bp-ap).norm()*ar;
```

```
	return make_pair(ap+rotate(v,cost,-sint),ap+rotate(v,cost,sint));
}

inline pair<point,point> crosspoint(const Circle &a,const Circle &b) {
	return crosspoint(a.p,a.r,b.p,b.r);
}

const int maxn=2000;
struct Node {
	point p;
	double a;
	int d;
	Node(const point &p,double a,int d):p(p),a(a),d(d) {}
	bool operator <(const Node &o)const {
		return a<o.a;
	}
};

double arg(point p) {
	return arg(complex<double>(p.x,p.y));
}

double solve(int m,Circle tc[],Circle c[]) {
	int n=0;
	sort(tc,tc+m);
	m=unique(tc,tc+m)-tc;//unique返回去重后的尾地址
	for(int i=m-1; i>=0; i--) {
		bool ok=true;
		for(int j=i+1; j<m; ++j) {
			double d=(tc[i].p-tc[j].p).norm();
			if(cmp(d-abs(tc[i].r-tc[j].r))<=0) {
				ok=false;
				break;
			}
		}
		if(ok)c[n++]=tc[i];
	}
	double ans=0;
	for(int i=0; i<n; ++i) {
		vector<Node> event;
		point boundary=c[i].p+point(-c[i].r,0);
		event.push_back(Node(boundary,-PI,0));
		event.push_back(Node(boundary,PI,0));
		for(int j=0; j<n; ++j) {
			if(i==j)continue;
			double d=(c[i].p-c[j].p).norm();
```

```
                if(cmp(d-(c[i].r+c[j].r))<0) {
                        pair<point,point> ret=crosspoint(c[i],c[j]);
                        double x=arg(ret.first-c[i].p);
                        double y=arg(ret.second-c[i].p);
                        if(cmp(x-y)>0) {
                                event.push_back(Node(ret.first,x,1));
                                event.push_back(Node(boundary,PI,-1));
                                event.push_back(Node(boundary,-PI,1));
                                event.push_back(Node(ret.second,y,-1));
                        } else {
                                event.push_back(Node(ret.first,x,1));
                                event.push_back(Node(ret.second,y,-1));
                        }
                }
        }
        sort(event.begin(),event.end());
        int sum=event[0].d;
        for(int j=1; j<(int)event.size(); ++j) {
                if(sum==0) {
                        ans+=cross(event[j-1].p,event[j].p)/2.;
                        double x=event[j-1].a;
                        double y=event[j].a;
                        double area=c[i].r*c[i].r*(y-x)/2;
                        point v1=event[j-1].p-c[i].p;
                        point v2=event[j].p-c[i].p;
                        area-=cross(v1,v2)/2.;
                        ans+=area;
                }
                sum+=event[j].d;
        }
    }
    return ans;
}

Circle c[maxn],tc[maxn];
int m;
int main() {
    scanf("%d",&m);
    for(int i=0; i<m; i++)
        tc[i].p.input(),scanf("%lf",&tc[i].r);
    printf("%.5lf\n",solve(m,tc,c));
    return 0;
}
```

1.8.9 9 弧度转换

```
double a = asin(sina)*180/PI; //sina 对应的度数
asin(1/2) -> 1/2对应的弧度
```

1.9 椭圆

1.9.1 椭圆

```
a是长轴一半(最宽),b是短轴一半(最高)

S=PI * a * b;
C=2* PI * b + 4(a-b)
```

1.10 最近点对

1.10.1 最近点对板子

```
最近点对 http://acm.hdu.edu.cn/showproblem.php?pid=1007**

const int N = 1e5 + 50;
const double eps = 1e-8;

int sgn(double x) {
    if(fabs(x) < eps)return 0;
    else return x < 0?-1:1;
}

struct Point {double x,y;};

double Distance(Point A,Point B) {return hypot(A.x-B.x,A.y-B.y);}
bool cmpxy(Point A,Point B) {
    return sgn(A.x - B.x) < 0 || (sgn(A.x-B.x) == 0 && sgn(A.y - B.y) < 0);
}

bool cmpy(Point A,Point B) {return sgn(A.y - B.y) < 0;}

Point p[N],tmp[N];

double Close_pair(int l,int r) {
    double dis = INF;
    if(l == r)return dis;
    if(l + 1 == r)return Distance(p[l],p[r]);
    int mid = (l + r) >> 1;
```

```
    double d1 = Close_pair(l,mid);
    double d2 = Close_pair(mid+1,r);
    dis = min(d1,d2);
    int k =0 ;
    for(int i=l; i<=r; i++) { //找两个点在s1,s2的情况
        if(fabs(p[mid].x - p[i].x) <= dis)tmp[k++] = p[i];
    }
    sort(tmp,tmp+k,cmpy); //根据y小排序,剪枝
    for(int i=0; i<k; i++) {
        for(int j=i+1; j<k; j++) {
            if(tmp[j].y - tmp[i].y >= dis)break;
            dis = min(dis,Distance(tmp[i],tmp[j]));
        }
    }return dis;
}
int n;
void work() {
    while(~scanf("%d",&n) && n) {
        for(int i=0; i<n; i++) {
            scanf("%lf%lf",&p[i].x,&p[i].y);
        }
               sort(p,p+n,cmpxy);
        printf("%.2f\n",Close_pair(0,n-1) / 2.0); //不用/2.0
    }
}
```

1.11 旋转卡壳

1.11.1 旋转卡壳

旋转卡壳的应用,凸包直径,凸包外接矩形

一).计算距离

1.凸包直径 [模板 P1452](https://www.luogu.com.cn/problem/P1452)

```
const int N = 5e4 + 50;
const double eps = 1e-8;
int sgn(double x) {
       if(fabs(x) < eps)return 0;
       else return x<0?-1:1;
}
struct Point {
       double x,y;
       Point() {}
```

```
        Point(double x,double y):x(x),y(y) {}
        Point operator + (Point B) {return Point(x+B.x,y+B.y);}
        Point operator - (Point B) {return Point(x-B.x,y-B.y);}
        bool operator == (Point B) {return sgn(x-B.x) == 0 && sgn(y-B.y) == 0;}
        bool operator < (Point B) {
        return sgn(x-B.x) < 0 || (sgn(x-B.x) == 0 && sgn(y-B.y)< 0);
        }
};

Point p[N],ch[N];
typedef Point Vector;

ll Distance2(Point A,Point B) {return (A.x-B.x)*(A.x-B.x) +(A.y-B.y)*(A.y-B.y);}
ll Cross(Vector A,Vector B) {return A.x*B.y -A.y*B.x;}

ll Cross2(Point A,Point B,Point C) {
        Point x = B-A; Point y = C-A;
        return x.x*y.y - x.y*y.x;
}

ll find(int n,Point *p) { //凸包直径
        int ymax = -1e5, ymin = 1e5;
        int ymaxid,yminid;
        for(int i=0; i<n; i++) {
                if(p[i].y > ymax) {
                        ymax = p[i].y;
                        ymaxid = i;
                }
                if(p[i].y < ymin) {
                        ymin = p[i].y;
                        yminid = i;
                }
        }
        ll ans = Distance2(p[ymaxid],p[yminid]);

        for(int i=0; i<n; i++,yminid=(yminid+1)%n) {
                while(Cross2(p[yminid+1],p[ymaxid+1],p[yminid]) >
                    Cross2(p[yminid+1],p[ymaxid],p[yminid]))
                        ymaxid = (ymaxid + 1) % n;
                ans = max(ans,Distance2(p[ymaxid],p[yminid]));
                ans = max(ans,Distance2(p[ymaxid],p[yminid+1]));
        }
        return ans;
}

int n;
```

```
int Convex_hull(Point *p,int n,Point *ch) {
	sort(p,p+n);
	n = unique(p,p+n)-p;
	int v = 0;
	for(int i=0; i<n; i++) {
		while(v > 1 && sgn(Cross(ch[v-1]-ch[v-2],p[i]-ch[v-2])) <= 0)
			v--;
		ch[v++] = p[i];
	}
	int j = v;
	for(int i=n-2; i>=0; i--) {
		while(v > j && sgn(Cross(ch[v-1]-ch[v-2],p[i]-ch[v-2])) <= 0)
			v--;
		ch[v++] = p[i];
	}
	if(n > 1)v--;
	return v;
}

void work() {
	scanf("%d",&n);
	for(int i=0; i<n; i++) {
		scanf("%lf%lf",&p[i].x,&p[i].y);
	}
	int v = Convex_hull(p,n,ch); //求凸包
	ll ans = find(v,ch); //找最长直径
	printf("%lld\n",ans);
}
```

二).外接矩形

1.最小面积外接矩形 [模板](https://www.luogu.com.cn/problem/P3187)

```cpp
const double INF = 1e80+7; //hdu5251,P3187
const int N = 5e4 + 50;
const double eps = 1e-8;

int sgn(double x) {
	if(fabs(x) < eps)return 0;
	else return x<0?-1:1;
}

int dcmp(double x,double y) {
```

```
        if(fabs(x-y)<eps)return 0;
        else return x<y?-1:1;
}
struct Point {
        double x,y;
        Point() {}
        Point(double x,double y):x(x),y(y) {}
        Point operator + (Point B) {return Point(x+B.x,y+B.y);}
        Point operator - (Point B) {return Point(x-B.x,y-B.y);}
        bool operator == (Point B) {return sgn(x-B.x) == 0 && sgn(y-B.y) == 0;}
        Point operator * (double k) {return Point(x*k,y*k);}
        friend bool operator < (Point A,Point B) {
                return fabs(A.y-B.y)<eps?A.x<B.x:A.y<B.y;
        }
};

Point p[N],ch[N],res[5];
int n,v;
double ans;
typedef Point Vector;

double dis(Point x) {return sqrt(x.x*x.x+x.y*x.y);}

double Dot(Vector A,Vector B) { //判断角度
        return A.x*B.x+A.y*B.y;
}

double Cross(Point A,Point B) {return A.x*B.y-A.y*B.x;}

bool cmp(Point A,Point B) { //按逆时针排序
        double t = Cross(A-p[1],B-p[1]);
        if(fabs(t) < eps)return dis(p[1]-A)-dis(p[1]-B) < 0;//三点共线
        return t > 0;
}

void graham() { //求凸包
        for(int i=2; i<=n; i++) { //左下角点
                if(p[i] < p[1])swap(p[i],p[1]);
        }
        sort(p+2,p+1+n,cmp);
        v = 0;
        ch[++v] = p[1];
        for(int i=2; i<=n; i++) {
                while(v>1 && sgn(Cross(ch[v]-ch[v-1],p[i]-ch[v])) <= eps)v--;
                ch[++v] = p[i];
        }
```

```
        ch[0] = ch[v];
}

void findit() {
        int l=1,r=1,p=1; //p是(i,i+1)为底的最高点,l,r是此时的最左点和最右点,用Dot投影判断
        double L,R,D,H;
        for(int i=0; i<v; i++) {
                D = dis(ch[i+1]-ch[i]);
                while(Cross(ch[i+1]-ch[i],ch[p+1]-ch[i])-Cross(ch[i+1]-ch[i],ch[p]-ch[i])>-eps)p=(p+1)%v;
                while(Dot(ch[i+1]-ch[i],ch[r+1]-ch[i])-Dot(ch[i+1]-ch[i],ch[r]-ch[i])>-eps)r=(r+1)%v;
                if(i==0)l=r;
                while(Dot(ch[i+1]-ch[i],ch[l+1]-ch[i])-Dot(ch[i+1]-ch[i],ch[l]-ch[i])<eps)l=(l+1)%v;
                //确定四个端点
                L=Dot(ch[i+1]-ch[i],ch[l]-ch[i])/D; //左端点在底部的投影(负号)
                R=Dot(ch[i+1]-ch[i],ch[r]-ch[i])/D;
                H=Cross(ch[i+1]-ch[i],ch[p]-ch[i])/D;
                if(H < 0)H = -H;
                double tmp=(R-L)*H;
                if(tmp < ans) {
                        ans=tmp;
                        res[0]=ch[i]+(ch[i+1]-ch[i])*(R/D);//右下
                        res[1]=res[0]+(ch[r]-res[0])*(H/dis(res[0]-ch[r]));
                        res[2]=res[1]+(ch[i]-res[0])*((R-L)/dis(ch[i]-res[0]));
                        res[3]=res[2]+(res[0]-res[1]);
                }
        }
}

void work() {
        scanf("%d",&n);
        for(int i=1; i<=n; i++) {
                scanf("%lf%lf",&p[i].x,&p[i].y);
        }
        ans = INF;
        graham(); //凸包
        findit();
        printf("%.5f\n",ans);
        int mini = 0;
        rep(i,1,3) {
                if(res[i]<res[mini])mini = i;
        }
        rep(i,0,3) {
                int t = (mini + i) % 4;
                if(sgn(res[t].x) == 0)res[t].x = 0;
                if(sgn(res[t].y) == 0)res[t].y = 0;
                printf("%.5f %.5f\n",res[t].x,res[t].y);
```

```
	}
}
```

2.最小周长外接矩形,如上改一下(没测试)

1.12 其他

1.12.1 n 条直线交点

问n条平行于x,y的直线交点个数

```cpp
struct Line {
	int st,ed,pos;
	Line() {}
	Line(int a,int b,int c) {
		st=a,ed=b,pos=c;
	}
	bool operator<(const Line&B)const {
		return pos < B.pos;
	}
};
int n;
Point p[N],tmp[N];
vector<Line>vx,vy;

void work() {
	vx.clear(),vy.clear();
	scanf("%d",&n);
	for(int i=0; i<n; i++) {
		double x1,y1,x2,y2;
		scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
		if(x1!=x2) {
			vx.push_back(Line(min(x1,x2),max(x1,x2),y1));
		} else {
			vy.push_back(Line(min(y1,y2),max(y1,y2),x1));
		}
	}
	sort(vy.begin(),vy.end());//根据x排序
	vector<Line>::iterator it;

	int ans =0;
	for(it=vx.begin(); it!=vx.end(); it++) {
		Line now = *it;
		Line s=Line(0,0,now.st);
```

```
                Line e=Line(0,0,now.ed+1);
                int l = lower_bound(vy.begin(),vy.end(),s)-vy.begin();
                int r = lower_bound(vy.begin(),vy.end(),e)-vy.begin();
                for(int i=l; i<r; i++) {
                        if(vy[i].st<=now.pos && vy[i].ed>=now.pos)ans++;
                }
        }
        printf("%d\n",ans);

}
```