7 计算几何

7.1 二维几何

```
// 计算几何模板
const double eps = 1e-8;
const double inf = 1e20;
const double pi = acos(-1.0);
const int maxp = 1010;
//Compares a double to zero
int sgn(double x){
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
//square of a double
inline double sqr(double x){return x*x;}
/*
 * Point
 * Point()                - Empty constructor
 * Point(double _x,double _y)  - constructor
 * input()              - double input
 * output()             - %.2f output
 * operator ==          - compares x and y
 * operator <           - compares first by x, then by y
 * operator -           - return new Point after subtracting
    curresponging x and y
 * operator ^           - cross product of 2d points
 * operator *           - dot product
 * len()                - gives length from origin
 * len2()               - gives square of length from origin
 * distance(Point p)    - gives distance from p
 * operator + Point b   - returns new Point after adding
    curresponging x and y
 * operator * double k - returns new Point after multiplieing x and
     y by k
 * operator / double k - returns new Point after divideing x and y
    by k
 * rad(Point a,Point b)- returns the angle of Point a and Point b
    from this Point
 * trunc(double r)      - return Point that if truncated the
    distance from center to r
 * rotleft()            - returns 90 degree ccw rotated point
 * rotright()           - returns 90 degree cw rotated point
 * rotate(Point p,double angle) - returns Point after rotateing the
     Point centering at p by angle radian ccw
 */
struct Point{
    double x,y;
    Point(){}
    Point(double _x,double _y){
```

```
41          x = _x;
42          y = _y;
43      }
44      void input(){
45          scanf("%lf%lf",&x,&y);
46      }
47      void output(){
48          printf("%.2f␣%.2f\n",x,y);
49      }
50      bool operator == (Point b)const{
51          return sgn(x-b.x) == 0 && sgn(y-b.y) == 0;
52      }
53      bool operator < (Point b)const{
54          return sgn(x-b.x)== 0?sgn(y-b.y)<0:x<b.x;
55      }
56      Point operator -(const Point &b)const{
57          return Point(x-b.x,y-b.y);
58      }
59      //叉积
60      double operator ^(const Point &b)const{
61          return x*b.y - y*b.x;
62      }
63      //点积
64      double operator *(const Point &b)const{
65          return x*b.x + y*b.y;
66      }
67      //返回长度
68      double len(){
69          return hypot(x,y);//库函数
70      }
71      //返回长度的平方
72      double len2(){
73          return x*x + y*y;
74      }
75      //返回两点的距离
76      double distance(Point p){
77          return hypot(x-p.x,y-p.y);
78      }
79      Point operator +(const Point &b)const{
80          return Point(x+b.x,y+b.y);
81      }
82      Point operator *(const double &k)const{
83          return Point(x*k,y*k);
84      }
85      Point operator /(const double &k)const{
86          return Point(x/k,y/k);
87      }
88      //计算 pa 和 pb 的夹角
89      //就是求这个点看 a,b 所成的夹角
90      //测试 LightOJ1203
91      double rad(Point a,Point b){
```

```
92          Point p = *this;
93          return fabs(atan2( fabs((a-p)^(b-p)),(a-p)*(b-p) ));
94      }
95      //化为长度为 r 的向量
96      Point trunc(double r){
97          double l = len();
98          if(!sgn(l))return *this;
99          r /= l;
100         return Point(x*r,y*r);
101     }
102     //逆时针旋转 90 度
103     Point rotleft(){
104         return Point(-y,x);
105     }
106     //顺时针旋转 90 度
107     Point rotright(){
108         return Point(y,-x);
109     }
110     //绕着 p 点逆时针旋转 angle
111     Point rotate(Point p,double angle){
112         Point v = (*this) - p;
113         double c = cos(angle), s = sin(angle);
114         return Point(p.x + v.x*c - v.y*s,p.y + v.x*s + v.y*c);
115     }
116 };
117 /*
118  * Stores two points
119  * Line()                          - Empty constructor
120  * Line(Point _s,Point _e)         - Line through _s and _e
121  * operator ==                     - checks if two points are same
122  * Line(Point p,double angle)      - one end p , another end at
        angle degree
123  * Line(double a,double b,double c) - Line of equation ax + by + c
        = 0
124  * input()                         - inputs s and e
125  * adjust()                        - orders in such a way that s < e
126  * length()                        - distance of se
127  * angle()                         - return 0 <= angle < pi
128  * relation(Point p)               - 3 if point is on line
129  *                                   1 if point on the left of line
130  *                                   2 if point on the right of line
131  * pointonseg(double p)            - return true if point on segment
132  * parallel(Line v)                - return true if they are
        parallel
133  * segcrossseg(Line v)             - returns 0 if does not intersect
134  *                                   returns 1 if non-standard
        intersection
135  *                                   returns 2 if intersects
136  * linecrossseg(Line v)            - line and seg
137  * linecrossline(Line v)           - 0 if parallel
138  *                                   1 if coincides
```

```
139  *                                    2 if intersects
140  * crosspoint(Line v)                 - returns intersection point
141  * dispointtoline(Point p)            - distance from point p to the
        line
142  * dispointtoseg(Point p)             - distance from p to the segment
143  * dissegtoseg(Line v)                - distance of two segment
144  * lineprog(Point p)                  - returns projected point p on se
        line
145  * symmetrypoint(Point p)            - returns reflection point of p
        over se
146  *
147  */
148 struct Line{
149     Point s,e;
150     Line(){}
151     Line(Point _s,Point _e){
152         s = _s;
153         e = _e;
154     }
155     bool operator ==(Line v){
156         return (s == v.s)&&(e == v.e);
157     }
158     //根据一个点和倾斜角 angle 确定直线,0<=angle<pi
159     Line(Point p,double angle){
160         s = p;
161         if(sgn(angle-pi/2) == 0){
162             e = (s + Point(0,1));
163         }
164         else{
165             e = (s + Point(1,tan(angle)));
166         }
167     }
168     //ax+by+c=0
169     Line(double a,double b,double c){
170         if(sgn(a) == 0){
171             s = Point(0,-c/b);
172             e = Point(1,-c/b);
173         }
174         else if(sgn(b) == 0){
175             s = Point(-c/a,0);
176             e = Point(-c/a,1);
177         }
178         else{
179             s = Point(0,-c/b);
180             e = Point(1,(-c-a)/b);
181         }
182     }
183     void input(){
184         s.input();
185         e.input();
186     }
```

```
187     void adjust(){
188         if(e < s)swap(s,e);
189     }
190     //求线段长度
191     double length(){
192         return s.distance(e);
193     }
194     //返回直线倾斜角 0<=angle<pi
195     double angle(){
196         double k = atan2(e.y-s.y,e.x-s.x);
197         if(sgn(k) < 0)k += pi;
198         if(sgn(k-pi) == 0)k -= pi;
199         return k;
200     }
201     //点和直线关系
202     //1 在左侧
203     //2 在右侧
204     //3 在直线上
205     int relation(Point p){
206         int c = sgn((p-s)^(e-s));
207         if(c < 0)return 1;
208         else if(c > 0)return 2;
209         else return 3;
210     }
211     // 点在线段上的判断
212     bool pointonseg(Point p){
213         return sgn((p-s)^(e-s)) == 0 && sgn((p-s)*(p-e)) <= 0;
214     }
215     //两向量平行 (对应直线平行或重合)
216     bool parallel(Line v){
217         return sgn((e-s)^(v.e-v.s)) == 0;
218     }
219     //两线段相交判断
220     //2 规范相交
221     //1 非规范相交
222     //0 不相交
223     int segcrossseg(Line v){
224         int d1 = sgn((e-s)^(v.s-s));
225         int d2 = sgn((e-s)^(v.e-s));
226         int d3 = sgn((v.e-v.s)^(s-v.s));
227         int d4 = sgn((v.e-v.s)^(e-v.s));
228         if( (d1^d2)==-2 && (d3^d4)==-2 )return 2;
229         return (d1==0 && sgn((v.s-s)*(v.s-e))<=0) ||
230             (d2==0 && sgn((v.e-s)*(v.e-e))<=0) ||
231             (d3==0 && sgn((s-v.s)*(s-v.e))<=0) ||
232             (d4==0 && sgn((e-v.s)*(e-v.e))<=0);
233     }
234     //直线和线段相交判断
235     //-*this line -v seg
236     //2 规范相交
237     //1 非规范相交
```

```
238     //0 不相交
239     int linecrossseg(Line v){
240         int d1 = sgn((e-s)^(v.s-s));
241         int d2 = sgn((e-s)^(v.e-s));
242         if((d1^d2)==-2) return 2;
243         return (d1==0||d2==0);
244     }
245     //两直线关系
246     //0 平行
247     //1 重合
248     //2 相交
249     int linecrossline(Line v){
250         if((*this).parallel(v))
251             return v.relation(s)==3;
252         return 2;
253     }
254     //求两直线的交点
255     //要保证两直线不平行或重合
256     Point crosspoint(Line v){
257         double a1 = (v.e-v.s)^(s-v.s);
258         double a2 = (v.e-v.s)^(e-v.s);
259         return Point((s.x*a2-e.x*a1)/(a2-a1),(s.y*a2-e.y*a1)/(a2-a1
                ));
260     }
261     //点到直线的距离
262     double dispointtoline(Point p){
263         return fabs((p-s)^(e-s))/length();
264     }
265     //点到线段的距离
266     double dispointtoseg(Point p){
267         if(sgn((p-s)*(e-s))<0 || sgn((p-e)*(s-e))<0)
268             return min(p.distance(s),p.distance(e));
269         return dispointtoline(p);
270     }
271     //返回线段到线段的距离
272     //前提是两线段不相交，相交距离就是 0 了
273     double dissegtoseg(Line v){
274         return min(min(dispointtoseg(v.s),dispointtoseg(v.e)),min(v
                .dispointtoseg(s),v.dispointtoseg(e)));
275     }
276     //返回点 p 在直线上的投影
277     Point lineprog(Point p){
278         return s + ( ((e-s)*((e-s)*(p-s)))/((e-s).len2()) );
279     }
280     //返回点 p 关于直线的对称点
281     Point symmetrypoint(Point p){
282         Point q = lineprog(p);
283         return Point(2*q.x-p.x,2*q.y-p.y);
284     }
285 };
286 //圆
```

```
287 struct circle{
288     Point p;//圆心
289     double r;//半径
290     circle(){}
291     circle(Point _p,double _r){
292         p = _p;
293         r = _r;
294     }
295     circle(double x,double y,double _r){
296         p = Point(x,y);
297         r = _r;
298     }
299     //三角形的外接圆
300     //需要 Point 的 + / rotate() 以及 Line 的 crosspoint()
301     //利用两条边的中垂线得到圆心
302     //测试：UVA12304
303     circle(Point a,Point b,Point c){
304         Line u = Line((a+b)/2,((a+b)/2)+((b-a).rotleft()));
305         Line v = Line((b+c)/2,((b+c)/2)+((c-b).rotleft()));
306         p = u.crosspoint(v);
307         r = p.distance(a);
308     }
309     //三角形的内切圆
310     //参数 bool t 没有作用，只是为了和上面外接圆函数区别
311     //测试：UVA12304
312     circle(Point a,Point b,Point c,bool t){
313         Line u,v;
314         double m = atan2(b.y-a.y,b.x-a.x), n = atan2(c.y-a.y,c.x-a.
              x);
315         u.s = a;
316         u.e = u.s + Point(cos((n+m)/2),sin((n+m)/2));
317         v.s = b;
318         m = atan2(a.y-b.y,a.x-b.x) , n = atan2(c.y-b.y,c.x-b.x);
319         v.e = v.s + Point(cos((n+m)/2),sin((n+m)/2));
320         p = u.crosspoint(v);
321         r = Line(a,b).dispointtoseg(p);
322     }
323     //输入
324     void input(){
325         p.input();
326         scanf("%lf",&r);
327     }
328     //输出
329     void output(){
330         printf("%.2lf␣%.2lf␣%.2lf\n",p.x,p.y,r);
331     }
332     bool operator == (circle v){
333         return (p==v.p) && sgn(r-v.r)==0;
334     }
335     bool operator < (circle v)const{
336         return ((p<v.p)||((p==v.p)&&sgn(r-v.r)<0));
```

```
337     }
338     //面积
339     double area(){
340         return pi*r*r;
341     }
342     //周长
343     double circumference(){
344         return 2*pi*r;
345     }
346     //点和圆的关系
347     //0 圆外
348     //1 圆上
349     //2 圆内
350     int relation(Point b){
351         double dst = b.distance(p);
352         if(sgn(dst-r) < 0)return 2;
353         else if(sgn(dst-r)==0)return 1;
354         return 0;
355     }
356     //线段和圆的关系
357     //比较的是圆心到线段的距离和半径的关系
358     int relationseg(Line v){
359         double dst = v.dispointtoseg(p);
360         if(sgn(dst-r) < 0)return 2;
361         else if(sgn(dst-r) == 0)return 1;
362         return 0;
363     }
364     //直线和圆的关系
365     //比较的是圆心到直线的距离和半径的关系
366     int relationline(Line v){
367         double dst = v.dispointtoline(p);
368         if(sgn(dst-r) < 0)return 2;
369         else if(sgn(dst-r) == 0)return 1;
370         return 0;
371     }
372     //两圆的关系
373     //5 相离
374     //4 外切
375     //3 相交
376     //2 内切
377     //1 内含
378     //需要 Point 的 distance
379     //测试: UVA12304
380     int relationcircle(circle v){
381         double d = p.distance(v.p);
382         if(sgn(d-r-v.r) > 0)return 5;
383         if(sgn(d-r-v.r) == 0)return 4;
384         double l = fabs(r-v.r);
385         if(sgn(d-r-v.r)<0 && sgn(d-l)>0)return 3;
386         if(sgn(d-l)==0)return 2;
387         if(sgn(d-l)<0)return 1;
```

```
388     }
389     //求两个圆的交点，返回 0 表示没有交点，返回 1 是一个交点，2 是两个交点
390     //需要 relationcircle
391     //测试：UVA12304
392     int pointcrosscircle(circle v,Point &p1,Point &p2){
393         int rel = relationcircle(v);
394         if(rel == 1 || rel == 5)return 0;
395         double d = p.distance(v.p);
396         double l = (d*d+r*r-v.r*v.r)/(2*d);
397         double h = sqrt(r*r-l*l);
398         Point tmp = p + (v.p-p).trunc(l);
399         p1 = tmp + ((v.p-p).rotleft().trunc(h));
400         p2 = tmp + ((v.p-p).rotright().trunc(h));
401         if(rel == 2 || rel == 4)
402             return 1;
403         return 2;
404     }
405     //求直线和圆的交点，返回交点个数
406     int pointcrossline(Line v,Point &p1,Point &p2){
407         if(!(*this).relationline(v))return 0;
408         Point a = v.lineprog(p);
409         double d = v.dispointtoline(p);
410         d = sqrt(r*r-d*d);
411         if(sgn(d) == 0){
412             p1 = a;
413             p2 = a;
414             return 1;
415         }
416         p1 = a + (v.e-v.s).trunc(d);
417         p2 = a - (v.e-v.s).trunc(d);
418         return 2;
419     }
420     //得到过 a,b 两点，半径为 r1 的两个圆
421     int gercircle(Point a,Point b,double r1,circle &c1,circle &c2){
422         circle x(a,r1),y(b,r1);
423         int t = x.pointcrosscircle(y,c1.p,c2.p);
424         if(!t)return 0;
425         c1.r = c2.r = r;
426         return t;
427     }
428     //得到与直线 u 相切，过点 q，半径为 r1 的圆
429     //测试：UVA12304
430     int getcircle(Line u,Point q,double r1,circle &c1,circle &c2){
431         double dis = u.dispointtoline(q);
432         if(sgn(dis-r1*2)>0)return 0;
433         if(sgn(dis) == 0){
434             c1.p = q + ((u.e-u.s).rotleft().trunc(r1));
435             c2.p = q + ((u.e-u.s).rotright().trunc(r1));
436             c1.r = c2.r = r1;
437             return 2;
438         }
```

```
439         Line u1 = Line((u.s + (u.e-u.s).rotleft().trunc(r1)),(u.e +
                (u.e-u.s).rotleft().trunc(r1)));
440         Line u2 = Line((u.s + (u.e-u.s).rotright().trunc(r1)),(u.e
               + (u.e-u.s).rotright().trunc(r1)));
441         circle cc = circle(q,r1);
442         Point p1,p2;
443         if(!cc.pointcrossline(u1,p1,p2))cc.pointcrossline(u2,p1,p2)
               ;
444         c1 = circle(p1,r1);
445         if(p1 == p2){
446             c2 = c1;
447             return 1;
448         }
449         c2 = circle(p2,r1);
450         return 2;
451     }
452     //同时与直线 u,v 相切，半径为 r1 的圆
453     //测试：UVA12304
454     int getcircle(Line u,Line v,double r1,circle &c1,circle &c2,
           circle &c3,circle &c4){
455         if(u.parallel(v))return 0;//两直线平行
456         Line u1 = Line(u.s + (u.e-u.s).rotleft().trunc(r1),u.e + (u
               .e-u.s).rotleft().trunc(r1));
457         Line u2 = Line(u.s + (u.e-u.s).rotright().trunc(r1),u.e + (
               u.e-u.s).rotright().trunc(r1));
458         Line v1 = Line(v.s + (v.e-v.s).rotleft().trunc(r1),v.e + (v
               .e-v.s).rotleft().trunc(r1));
459         Line v2 = Line(v.s + (v.e-v.s).rotright().trunc(r1),v.e + (
               v.e-v.s).rotright().trunc(r1));
460         c1.r = c2.r = c3.r = c4.r = r1;
461         c1.p = u1.crosspoint(v1);
462         c2.p = u1.crosspoint(v2);
463         c3.p = u2.crosspoint(v1);
464         c4.p = u2.crosspoint(v2);
465         return 4;
466     }
467     //同时与不相交圆 cx,cy 相切，半径为 r1 的圆
468     //测试：UVA12304
469     int getcircle(circle cx,circle cy,double r1,circle &c1,circle &
           c2){
470         circle x(cx.p,r1+cx.r),y(cy.p,r1+cy.r);
471         int t = x.pointcrosscircle(y,c1.p,c2.p);
472         if(!t)return 0;
473         c1.r = c2.r = r1;
474         return t;
475     }
476
477     //过一点作圆的切线 (先判断点和圆的关系)
478     //测试：UVA12304
479     int tangentline(Point q,Line &u,Line &v){
480         int x = relation(q);
```

```
481         if(x == 2)return 0;
482         if(x == 1){
483             u = Line(q,q + (q-p).rotleft());
484             v = u;
485             return 1;
486         }
487         double d = p.distance(q);
488         double l = r*r/d;
489         double h = sqrt(r*r-l*l);
490         u = Line(q,p + ((q-p).trunc(l) + (q-p).rotleft().trunc(h)))
                ;
491         v = Line(q,p + ((q-p).trunc(l) + (q-p).rotright().trunc(h))
                );
492         return 2;
493     }
494     //求两圆相交的面积
495     double areacircle(circle v){
496         int rel = relationcircle(v);
497         if(rel >= 4)return 0.0;
498         if(rel <= 2)return min(area(),v.area());
499         double d = p.distance(v.p);
500         double hf = (r+v.r+d)/2.0;
501         double ss = 2*sqrt(hf*(hf-r)*(hf-v.r)*(hf-d));
502         double a1 = acos((r*r+d*d-v.r*v.r)/(2.0*r*d));
503         a1 = a1*r*r;
504         double a2 = acos((v.r*v.r+d*d-r*r)/(2.0*v.r*d));
505         a2 = a2*v.r*v.r;
506         return a1+a2-ss;
507     }
508     //求圆和三角形 pab 的相交面积
509     //测试：POJ3675 HDU3982 HDU2892
510     double areatriangle(Point a,Point b){
511         if(sgn((p-a)^(p-b)) == 0)return 0.0;
512         Point q[5];
513         int len = 0;
514         q[len++] = a;
515         Line l(a,b);
516         Point p1,p2;
517         if(pointcrossline(l,q[1],q[2])==2){
518             if(sgn((a-q[1])*(b-q[1]))<0)q[len++] = q[1];
519             if(sgn((a-q[2])*(b-q[2]))<0)q[len++] = q[2];
520         }
521         q[len++] = b;
522         if(len == 4 && sgn((q[0]-q[1])*(q[2]-q[1]))>0)swap(q[1],q
                [2]);
523         double res = 0;
524         for(int i = 0;i < len-1;i++){
525             if(relation(q[i])==0||relation(q[i+1])==0){
526                 double arg = p.rad(q[i],q[i+1]);
527                 res += r*r*arg/2.0;
528             }
```

```
529             else{
530                 res += fabs((q[i]-p)^(q[i+1]-p))/2.0;
531             }
532         }
533         return res;
534     }
535 };
536 
537 /*
538  * n,p  Line l for each side
539  * input(int _n)                          - inputs _n size polygon
540  * add(Point q)                           - adds a point at end of
       the list
541  * getline()                              - populates line array
542  * cmp                                    - comparision in
       convex_hull order
543  * norm()                                 - sorting in convex_hull
       order
544  * getconvex(polygon &convex)             - returns convex hull in
       convex
545  * Graham(polygon &convex)                - returns convex hull in
       convex
546  * isconvex()                             - checks if convex
547  * relationpoint(Point q)                 - returns 3 if q is a
       vertex
548  *                                                  2 if on a side
549  *                                                  1 if inside
550  *                                                  0 if outside
551  * convexcut(Line u,polygon &po)          - left side of u in po
552  * gercircumference()                     - returns side length
553  * getarea()                              - returns area
554  * getdir()                               - returns 0 for cw, 1 for
       ccw
555  * getbarycentre()                        - returns barycenter
556  *
557  */
558 struct polygon{
559     int n;
560     Point p[maxp];
561     Line l[maxp];
562     void input(int _n){
563         n = _n;
564         for(int i = 0;i < n;i++)
565             p[i].input();
566     }
567     void add(Point q){
568         p[n++] = q;
569     }
570     void getline(){
571         for(int i = 0;i < n;i++){
572             l[i] = Line(p[i],p[(i+1)%n]);
```

```
573         }
574     }
575     struct cmp{
576         Point p;
577         cmp(const Point &p0){p = p0;}
578         bool operator()(const Point &aa,const Point &bb){
579             Point a = aa, b = bb;
580             int d = sgn((a-p)^(b-p));
581             if(d == 0){
582                 return sgn(a.distance(p)-b.distance(p)) < 0;
583             }
584             return d > 0;
585         }
586     };
587     //进行极角排序
588     //首先需要找到最左下角的点
589     //需要重载号好 Point 的 < 操作符 (min 函数要用)
590     void norm(){
591         Point mi = p[0];
592         for(int i = 1;i < n;i++)mi = min(mi,p[i]);
593         sort(p,p+n,cmp(mi));
594     }
595     //得到凸包
596     //得到的凸包里面的点编号是 0~n-1 的
597     //两种凸包的方法
598     //注意如果有影响，要特判下所有点共点，或者共线的特殊情况
599     //测试 LightOJ1203 LightOJ1239
600     void getconvex(polygon &convex){
601         sort(p,p+n);
602         convex.n = n;
603         for(int i = 0;i < min(n,2);i++){
604             convex.p[i] = p[i];
605         }
606         if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n
                --;//特
                判
607         if(n <= 2)return;
608         int &top = convex.n;
609         top = 1;
610         for(int i = 2;i < n;i++){
611             while(top && sgn((convex.p[top]-p[i])^(convex.p[top-1]-
                    p[i])) <= 0)
612                 top--;
613             convex.p[++top] = p[i];
614         }
615         int temp = top;
616         convex.p[++top] = p[n-2];
617         for(int i = n-3;i >= 0;i--){
618             while(top != temp && sgn((convex.p[top]-p[i])^(convex.p
                    [top-1]-p[i])) <= 0)
619                 top--;
```

```
620             convex.p[++top] = p[i];
621         }
622         if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n
                --;//特
                判
623         convex.norm();//原来得到的是顺时针的点，排序后逆时针
624     }
625     //得到凸包的另外一种方法
626     //测试 LightOJ1203 LightOJ1239
627     void Graham(polygon &convex){
628         norm();
629         int &top = convex.n;
630         top = 0;
631         if(n == 1){
632             top = 1;
633             convex.p[0] = p[0];
634             return;
635         }
636         if(n == 2){
637             top = 2;
638             convex.p[0] = p[0];
639             convex.p[1] = p[1];
640             if(convex.p[0] == convex.p[1])top--;
641             return;
642         }
643         convex.p[0] = p[0];
644         convex.p[1] = p[1];
645         top = 2;
646         for(int i = 2;i < n;i++){
647             while( top > 1 && sgn((convex.p[top-1]-convex.p[top-2])
                    ^(p[i]-convex.p[top-2])) <= 0 )
648                 top--;
649             convex.p[top++] = p[i];
650         }
651         if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n
                --;//特
                判
652     }
653     //判断是不是凸的
654     bool isconvex(){
655         bool s[2];
656         memset(s,false,sizeof(s));
657         for(int i = 0;i < n;i++){
658             int j = (i+1)%n;
659             int k = (j+1)%n;
660             s[sgn((p[j]-p[i])^(p[k]-p[i]))+1] = true;
661             if(s[0] && s[2])return false;
662         }
663         return true;
664     }
665     //判断点和任意多边形的关系
```

```
666     // 3 点上
667     // 2 边上
668     // 1 内部
669     // 0 外部
670     int relationpoint(Point q){
671         for(int i = 0;i < n;i++){
672             if(p[i] == q)return 3;
673         }
674         getline();
675         for(int i = 0;i < n;i++){
676             if(l[i].pointonseg(q))return 2;
677         }
678         int cnt = 0;
679         for(int i = 0;i < n;i++){
680             int j = (i+1)%n;
681             int k = sgn((q-p[j])^(p[i]-p[j]));
682             int u = sgn(p[i].y-q.y);
683             int v = sgn(p[j].y-q.y);
684             if(k > 0 && u < 0 && v >= 0)cnt++;
685             if(k < 0 && v < 0 && u >= 0)cnt--;
686         }
687         return cnt != 0;
688     }
689     //直线 u 切割凸多边形左侧
690     //注意直线方向
691     //测试：HDU3982
692     void convexcut(Line u,polygon &po){
693         int &top = po.n;//注意引用
694         top = 0;
695         for(int i = 0;i < n;i++){
696             int d1 = sgn((u.e-u.s)^(p[i]-u.s));
697             int d2 = sgn((u.e-u.s)^(p[(i+1)%n]-u.s));
698             if(d1 >= 0)po.p[top++] = p[i];
699             if(d1*d2 < 0)po.p[top++] = u.crosspoint(Line(p[i],p[(i
                +1)%n]));
700         }
701     }
702     //得到周长
703     //测试 LightOJ1239
704     double getcircumference(){
705         double sum = 0;
706         for(int i = 0;i < n;i++){
707             sum += p[i].distance(p[(i+1)%n]);
708         }
709         return sum;
710     }
711     //得到面积
712     double getarea(){
713         double sum = 0;
714         for(int i = 0;i < n;i++){
715             sum += (p[i]^p[(i+1)%n]);
```

```
716         }
717         return fabs(sum)/2;
718     }
719     //得到方向
720     // 1 表示逆时针，0 表示顺时针
721     bool getdir(){
722         double sum = 0;
723         for(int i = 0;i < n;i++)
724             sum += (p[i]^p[(i+1)%n]);
725         if(sgn(sum) > 0)return 1;
726         return 0;
727     }
728     //得到重心
729     Point getbarycentre(){
730         Point ret(0,0);
731         double area = 0;
732         for(int i = 1;i < n-1;i++){
733             double tmp = (p[i]-p[0])^(p[i+1]-p[0]);
734             if(sgn(tmp) == 0)continue;
735             area += tmp;
736             ret.x += (p[0].x+p[i].x+p[i+1].x)/3*tmp;
737             ret.y += (p[0].y+p[i].y+p[i+1].y)/3*tmp;
738         }
739         if(sgn(area)) ret = ret/area;
740         return ret;
741     }
742     //多边形和圆交的面积
743     //测试：POJ3675 HDU3982 HDU2892
744     double areacircle(circle c){
745         double ans = 0;
746         for(int i = 0;i < n;i++){
747             int j = (i+1)%n;
748             if(sgn( (p[j]-c.p)^(p[i]-c.p) ) >= 0)
749                 ans += c.areatriangle(p[i],p[j]);
750             else ans -= c.areatriangle(p[i],p[j]);
751         }
752         return fabs(ans);
753     }
754     //多边形和圆关系
755     // 2 圆完全在多边形内
756     // 1 圆在多边形里面，碰到了多边形边界
757     // 0 其它
758     int relationcircle(circle c){
759         getline();
760         int x = 2;
761         if(relationpoint(c.p) != 1)return 0;//圆心不在内部
762         for(int i = 0;i < n;i++){
763             if(c.relationseg(l[i])==2)return 0;
764             if(c.relationseg(l[i])==1)x = 1;
765         }
766         return x;
```

```
767     }
768 };
769 //AB X AC
770 double cross(Point A,Point B,Point C){
771     return (B-A)^(C-A);
772 }
773 //AB*AC
774 double dot(Point A,Point B,Point C){
775     return (B-A)*(C-A);
776 }
777 //最小矩形面积覆盖
778 // A 必须是凸包 (而且是逆时针顺序)
779 // 测试 UVA 10173
780 double minRectangleCover(polygon A){
781     //要特判 A.n < 3 的情况
782     if(A.n < 3)return 0.0;
783     A.p[A.n] = A.p[0];
784     double ans = -1;
785     int r = 1, p = 1, q;
786     for(int i = 0;i < A.n;i++){
787         //卡出离边 A.p[i] - A.p[i+1] 最远的点
788         while( sgn( cross(A.p[i],A.p[i+1],A.p[r+1]) - cross(A.p[i],
               A.p[i+1],A.p[r]) ) >= 0 )
789             r = (r+1)%A.n;
790         //卡出 A.p[i] - A.p[i+1] 方向上正向 n 最远的点
791         while(sgn( dot(A.p[i],A.p[i+1],A.p[p+1]) - dot(A.p[i],A.p[i
               +1],A.p[p]) ) >= 0 )
792             p = (p+1)%A.n;
793         if(i == 0)q = p;
794         //卡出 A.p[i] - A.p[i+1] 方向上负向最远的点
795         while(sgn(dot(A.p[i],A.p[i+1],A.p[q+1]) - dot(A.p[i],A.p[i
               +1],A.p[q])) <= 0)
796             q = (q+1)%A.n;
797         double d = (A.p[i] - A.p[i+1]).len2();
798         double tmp = cross(A.p[i],A.p[i+1],A.p[r]) *
799             (dot(A.p[i],A.p[i+1],A.p[p]) - dot(A.p[i],A.p[i+1],A.p[
                  q]))/d;
800         if(ans < 0 || ans > tmp)ans = tmp;
801     }
802     return ans;
803 }
804
805 //直线切凸多边形
806 //多边形是逆时针的，在 q1q2 的左侧
807 //测试:HDU3982
808 vector<Point> convexCut(const vector<Point> &ps,Point q1,Point q2){
809     vector<Point>qs;
810     int n = ps.size();
811     for(int i = 0;i < n;i++){
812         Point p1 = ps[i], p2 = ps[(i+1)%n];
813         int d1 = sgn((q2-q1)^(p1-q1)), d2 = sgn((q2-q1)^(p2-q1));
```

```
814         if(d1 >= 0)
815             qs.push_back(p1);
816         if(d1 * d2 < 0)
817             qs.push_back(Line(p1,p2).crosspoint(Line(q1,q2)));
818     }
819     return qs;
820 }
821 //半平面交
822 //测试 POJ3335 POJ1474 POJ1279
823 //**************************
824 struct halfplane:public Line{
825     double angle;
826     halfplane(){}
827     //表示向量 s->e 逆时针 (左侧) 的半平面
828     halfplane(Point _s,Point _e){
829         s = _s;
830         e = _e;
831     }
832     halfplane(Line v){
833         s = v.s;
834         e = v.e;
835     }
836     void calcangle(){
837         angle = atan2(e.y-s.y,e.x-s.x);
838     }
839     bool operator <(const halfplane &b)const{
840         return angle < b.angle;
841     }
842 };
843 struct halfplanes{
844     int n;
845     halfplane hp[maxp];
846     Point p[maxp];
847     int que[maxp];
848     int st,ed;
849     void push(halfplane tmp){
850         hp[n++] = tmp;
851     }
852     //去重
853     void unique(){
854         int m = 1;
855         for(int i = 1;i < n;i++){
856             if(sgn(hp[i].angle-hp[i-1].angle) != 0)
857                 hp[m++] = hp[i];
858             else if(sgn( (hp[m-1].e-hp[m-1].s)^(hp[i].s-hp[m-1].s)
                    ) > 0)
859                 hp[m-1] = hp[i];
860         }
861         n = m;
862     }
863     bool halfplaneinsert(){
```

```
        for(int i = 0;i < n;i++)hp[i].calcangle();
        sort(hp,hp+n);
        unique();
        que[st=0] = 0;
        que[ed=1] = 1;
        p[1] = hp[0].crosspoint(hp[1]);
        for(int i = 2;i < n;i++){
            while(st<ed && sgn((hp[i].e-hp[i].s)^(p[ed]-hp[i].s))<0)ed--;
            while(st<ed && sgn((hp[i].e-hp[i].s)^(p[st+1]-hp[i].s))<0)st++;
            que[++ed] = i;
            if(hp[i].parallel(hp[que[ed-1]]))return false;
            p[ed]=hp[i].crosspoint(hp[que[ed-1]]);
        }
        while(st<ed && sgn((hp[que[st]].e-hp[que[st]].s)^(p[ed]-hp[que[st]].s))<0)ed--;
        while(st<ed && sgn((hp[que[ed]].e-hp[que[ed]].s)^(p[st+1]-hp[que[ed]].s))<0)st++;
        if(st+1>=ed)return false;
        return true;
    }
    //得到最后半平面交得到的凸多边形
    //需要先调用 halfplaneinsert() 且返回 true
    void getconvex(polygon &con){
        p[st] = hp[que[st]].crosspoint(hp[que[ed]]);
        con.n = ed-st+1;
        for(int j = st,i = 0;j <= ed;i++,j++)
            con.p[i] = p[j];
    }
};
//***************************

const int maxn = 1010;
struct circles{
    circle c[maxn];
    double ans[maxn];//ans[i] 表示被覆盖了 i 次的面积
    double pre[maxn];
    int n;
    circles(){}
    void add(circle cc){
        c[n++] = cc;
    }
    //x 包含在 y 中
    bool inner(circle x,circle y){
        if(x.relationcircle(y) != 1)return 0;
        return sgn(x.r-y.r)<=0?1:0;
    }
    //圆的面积并去掉内含的圆
    void init_or(){
        bool mark[maxn] = {0};
```

```
911         int i,j,k=0;
912         for(i = 0;i < n;i++){
913             for(j = 0;j < n;j++)
914                 if(i != j && !mark[j]){
915                     if( (c[i]==c[j])||inner(c[i],c[j]) )break;
916                 }
917             if(j < n)mark[i] = 1;
918         }
919         for(i = 0;i < n;i++)
920             if(!mark[i])
921                 c[k++] = c[i];
922         n = k;
923     }
924     //圆的面积交去掉内含的圆
925     void init_add(){
926         int i,j,k;
927         bool mark[maxn] = {0};
928         for(i = 0;i < n;i++){
929             for(j = 0;j < n;j++)
930                 if(i != j && !mark[j]){
931                     if( (c[i]==c[j])||inner(c[j],c[i]) )break;
932                 }
933             if(j < n)mark[i] = 1;
934         }
935         for(i = 0;i < n;i++)
936             if(!mark[i])
937                 c[k++] = c[i];
938         n = k;
939     }
940     //半径为 r 的圆，弧度为 th 对应的弓形的面积
941     double areaarc(double th,double r){
942         return 0.5*r*r*(th-sin(th));
943     }
944     //测试 SPOJVCIRCLES SPOJCIRUT
945     //SPOJVCIRCLES 求 n 个圆并的面积，需要加上 init_or() 去掉重复圆（否则
        WA）
946     //SPOJCIRUT 是求被覆盖 k 次的面积，不能加 init_or()
947     //对于求覆盖多少次面积的问题，不能解决相同圆，而且不能 init_or()
948     //求多圆面积并，需要 init_or，其中一个目的就是去掉相同圆
949     void getarea(){
950         memset(ans,0,sizeof(ans));
951         vector<pair<double,int> >v;
952         for(int i = 0;i < n;i++){
953             v.clear();
954             v.push_back(make_pair(-pi,1));
955             v.push_back(make_pair(pi,-1));
956             for(int j = 0;j < n;j++)
957                 if(i != j){
958                     Point q = (c[j].p - c[i].p);
959                     double ab = q.len(),ac = c[i].r, bc = c[j].r;
960                     if(sgn(ab+ac-bc)<=0){
```

```
961                         v.push_back(make_pair(-pi,1));
962                         v.push_back(make_pair(pi,-1));
963                         continue;
964                     }
965                     if(sgn(ab+bc-ac)<=0)continue;
966                     if(sgn(ab-ac-bc)>0)continue;
967                     double th = atan2(q.y,q.x), fai = acos((ac*ac+
                            ab*ab-bc*bc)/(2.0*ac*ab));
968                     double a0 = th-fai;
969                     if(sgn(a0+pi)<0)a0+=2*pi;
970                     double a1 = th+fai;
971                     if(sgn(a1-pi)>0)a1-=2*pi;
972                     if(sgn(a0-a1)>0){
973                         v.push_back(make_pair(a0,1));
974                         v.push_back(make_pair(pi,-1));
975                         v.push_back(make_pair(-pi,1));
976                         v.push_back(make_pair(a1,-1));
977                     }
978                     else{
979                         v.push_back(make_pair(a0,1));
980                         v.push_back(make_pair(a1,-1));
981                     }
982                 }
983             sort(v.begin(),v.end());
984             int cur = 0;
985             for(int j = 0;j < v.size();j++){
986                 if(cur && sgn(v[j].first-pre[cur])){
987                     ans[cur] += areaarc(v[j].first-pre[cur],c[i].r)
                            ;
988                     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].first)));
989                 }
990                 cur += v[j].second;
991                 pre[cur] = v[j].first;
992             }
993         }
994         for(int i = 1;i < n;i++)
995             ans[i] -= ans[i+1];
996     }
997 };
```

7.2 三维几何

```
1 const double eps = 1e-8;
2 int sgn(double x){
3     if(fabs(x) < eps)return 0;
4     if(x < 0)return -1;
5     else return 1;
6 }
7 struct Point3{
```

```
 8      double x,y,z;
 9      Point3(double _x = 0,double _y = 0,double _z = 0){
10          x = _x;
11          y = _y;
12          z = _z;
13      }
14      void input(){
15          scanf("%lf%lf%lf",&x,&y,&z);
16      }
17      void output(){
18          scanf("%.2lf␣%.2lf␣%.2lf\n",x,y,z);
19      }
20      bool operator ==(const Point3 &b)const{
21          return sgn(x-b.x) == 0 && sgn(y-b.y) == 0 && sgn(z-b.z) ==
               0;
22      }
23      bool operator <(const Point3 &b)const{
24          return sgn(x-b.x)==0?(sgn(y-b.y)==0?sgn(z-b.z)<0:y<b.y):x<b
               .x;
25      }
26      double len(){
27          return sqrt(x*x+y*y+z*z);
28      }
29      double len2(){
30          return x*x+y*y+z*z;
31      }
32      double distance(const Point3 &b)const{
33          return sqrt((x-b.x)*(x-b.x)+(y-b.y)*(y-b.y)+(z-b.z)*(z-b.z)
               );
34      }
35      Point3 operator -(const Point3 &b)const{
36          return Point3(x-b.x,y-b.y,z-b.z);
37      }
38      Point3 operator +(const Point3 &b)const{
39          return Point3(x+b.x,y+b.y,z+b.z);
40      }
41      Point3 operator *(const double &k)const{
42          return Point3(x*k,y*k,z*k);
43      }
44      Point3 operator /(const double &k)const{
45          return Point3(x/k,y/k,z/k);
46      }
47      //点乘
48      double operator *(const Point3 &b)const{
49          return x*b.x+y*b.y+z*b.z;
50      }
51      //叉乘
52      Point3 operator ^(const Point3 &b)const{
53          return Point3(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
54      }
55      double rad(Point3 a,Point3 b){
```

```
56          Point3 p = (*this);
57          return acos( ( (a-p)*(b-p) )/ (a.distance(p)*b.distance(p))
                ;
58      }
59      //变换长度
60      Point3 trunc(double r){
61          double l = len();
62          if(!sgn(l))return *this;
63          r /= l;
64          return Point3(x*r,y*r,z*r);
65      }
66  };
67  struct Line3
68  {
69      Point3 s,e;
70      Line3(){}
71      Line3(Point3 _s,Point3 _e)
72      {
73          s = _s;
74          e = _e;
75      }
76      bool operator ==(const Line3 v)
77      {
78          return (s==v.s)&&(e==v.e);
79      }
80      void input()
81      {
82          s.input();
83          e.input();
84      }
85      double length()
86      {
87          return s.distance(e);
88      }
89      //点到直线距离
90      double dispointtoline(Point3 p)
91      {
92          return ((e-s)^(p-s)).len()/s.distance(e);
93      }
94      //点到线段距离
95      double dispointtoseg(Point3 p)
96      {
97          if(sgn((p-s)*(e-s)) < 0 || sgn((p-e)*(s-e)) < 0)
98              return min(p.distance(s),e.distance(p));
99          return dispointtoline(p);
100     }
101     //返回点 p 在直线上的投影
102     Point3 lineprog(Point3 p)
103     {
104         return s + ( ((e-s)*((e-s)*(p-s)))/((e-s).len2()) );
105     }
```

```
106     //p 绕此向量逆时针 arg 角度
107     Point3 rotate(Point3 p,double ang)
108     {
109         if(sgn(((s-p)^(e-p)).len()) == 0)return p;
110         Point3 f1 = (e-s)^(p-s);
111         Point3 f2 = (e-s)^(f1);
112         double len = ((s-p)^(e-p)).len()/s.distance(e);
113         f1 = f1.trunc(len); f2 = f2.trunc(len);
114         Point3 h = p+f2;
115         Point3 pp = h+f1;
116         return h + ((p-h)*cos(ang)) + ((pp-h)*sin(ang));
117     }
118     //点在直线上
119     bool pointonseg(Point3 p)
120     {
121         return sgn( ((s-p)^(e-p)).len() ) == 0 && sgn((s-p)*(e-p))
               == 0;
122     }
123 };
124 struct Plane
125 {
126     Point3 a,b,c,o;//平面上的三个点，以及法向量
127     Plane(){}
128     Plane(Point3 _a,Point3 _b,Point3 _c)
129     {
130         a = _a;
131         b = _b;
132         c = _c;
133         o = pvec();
134     }
135     Point3 pvec()
136     {
137         return (b-a)^(c-a);
138     }
139     //ax+by+cz+d = 0
140     Plane(double _a,double _b,double _c,double _d)
141     {
142         o = Point3(_a,_b,_c);
143         if(sgn(_a) != 0)
144             a = Point3((-_d-_c-_b)/_a,1,1);
145         else if(sgn(_b) != 0)
146             a = Point3(1,(-_d-_c-_a)/_b,1);
147         else if(sgn(_c) != 0)
148             a = Point3(1,1,(-_d-_a-_b)/_c);
149     }
150     //点在平面上的判断
151     bool pointonplane(Point3 p)
152     {
153         return sgn((p-a)*o) == 0;
154     }
155     //两平面夹角
```

```
156     double angleplane(Plane f)
157     {
158         return acos(o*f.o)/(o.len()*f.o.len());
159     }
160     //平面和直线的交点，返回值是交点个数
161     int crossline(Line3 u,Point3 &p)
162     {
163         double x = o*(u.e-a);
164         double y = o*(u.s-a);
165         double d = x-y;
166         if(sgn(d) == 0)return 0;
167         p = ((u.s*x)-(u.e*y))/d;
168         return 1;
169     }
170     //点到平面最近点 (也就是投影)
171     Point3 pointtoplane(Point3 p)
172     {
173         Line3 u = Line3(p,p+o);
174         crossline(u,p);
175         return p;
176     }
177     //平面和平面的交线
178     int crossplane(Plane f,Line3 &u)
179     {
180         Point3 oo = o^f.o;
181         Point3 v = o^oo;
182         double d = fabs(f.o*v);
183         if(sgn(d) == 0)return 0;
184         Point3 q = a + (v*(f.o*(f.a-a))/d);
185         u = Line3(q,q+oo);
186         return 1;
187     }
188 };
```

7.3 平面最近点对

HDU1007/ZOJ2107

```
 1 const int MAXN = 100010;
 2 const double eps = 1e-8;
 3 const double INF = 1e20;
 4 struct Point{
 5     double x,y;
 6     void input(){
 7         scanf("%lf%lf",&x,&y);
 8     }
 9 };
10 double dist(Point a,Point b){
11     return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
12 }
13 Point p[MAXN];
14 Point tmpt[MAXN];
15 bool cmpx(Point a,Point b){
```

```
16      return a.x < b.x || (a.x == b.x && a.y < b.y);
17  }
18  bool cmpy(Point a,Point b){
19      return a.y < b.y || (a.y == b.y && a.x < b.x);
20  }
21  double Closest_Pair(int left,int right){
22      double d = INF;
23      if(left == right)return d;
24      if(left+1 == right)return dist(p[left],p[right]);
25      int mid = (left+right)/2;
26      double d1 = Closest_Pair(left,mid);
27      double d2 = Closest_Pair(mid+1,right);
28      d = min(d1,d2);
29      int cnt = 0;
30      for(int i = left;i <= right;i++){
31          if(fabs(p[mid].x - p[i].x) <= d)
32              tmpt[cnt++] = p[i];
33      }
34      sort(tmpt,tmpt+cnt,cmpy);
35      for(int i = 0;i < cnt;i++){
36          for(int j = i+1;j < cnt && tmpt[j].y - tmpt[i].y < d;j++)
37              d = min(d,dist(tmpt[i],tmpt[j]));
38      }
39      return d;
40  }
41  int main(){
42      int n;
43      while(scanf("%d",&n) == 1 && n){
44          for(int i = 0;i < n;i++)p[i].input();
45          sort(p,p+n,cmpx);
46          printf("%.2lf\n",Closest_Pair(0,n-1));
47      }
48      return 0;
49  }
```

7.4 三维凸包

7.4.1 HDU4273

HDU 4273 给一个三维凸包，求重心到表面的最短距离。

```
1   const double eps = 1e-8;
2   const int MAXN = 550;
3   int sgn(double x){
4       if(fabs(x) < eps)return 0;
5       if(x < 0)return -1;
6       else return 1;
7   }
8   struct Point3{
9       double x,y,z;
10      Point3(double _x = 0, double _y = 0, double _z = 0){
11          x = _x;
```

```
12          y = _y;
13          z = _z;
14      }
15      void input(){
16          scanf("%lf%lf%lf",&x,&y,&z);
17      }
18      bool operator ==(const Point3 &b)const{
19          return sgn(x-b.x) == 0 && sgn(y-b.y) == 0 && sgn(z-b.z) ==
              0;
20      }
21      double len(){
22          return sqrt(x*x+y*y+z*z);
23      }
24      double len2(){
25          return x*x+y*y+z*z;
26      }
27      double distance(const Point3 &b)const{
28          return sqrt((x-b.x)*(x-b.x)+(y-b.y)*(y-b.y)+(z-b.z)*(z-b.z)
              );
29      }
30      Point3 operator -(const Point3 &b)const{
31          return Point3(x-b.x,y-b.y,z-b.z);
32      }
33      Point3 operator +(const Point3 &b)const{
34          return Point3(x+b.x,y+b.y,z+b.z);
35      }
36      Point3 operator *(const double &k)const{
37          return Point3(x*k,y*k,z*k);
38      }
39      Point3 operator /(const double &k)const{
40          return Point3(x/k,y/k,z/k);
41      }
42      //点乘
43      double operator *(const Point3 &b)const{
44          return x*b.x + y*b.y + z*b.z;
45      }
46      //叉乘
47      Point3 operator ^(const Point3 &b)const{
48          return Point3(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
49      }
50  };
51  struct CH3D{
52      struct face{
53          //表示凸包一个面上的三个点的编号
54          int a,b,c;
55          //表示该面是否属于最终的凸包上的面
56          bool ok;
57      };
58      //初始顶点数
59      int n;
60      Point3 P[MAXN];
```

```
61      //凸包表面的三角形数
62      int num;
63      //凸包表面的三角形
64      face F[8*MAXN];
65      int g[MAXN][MAXN];
66      //叉乘
67      Point3 cross(const Point3 &a,const Point3 &b,const Point3 &c){
68          return (b-a)^(c-a);
69      }
70      //三角形面积 *2
71      double area(Point3 a,Point3 b,Point3 c){
72          return ((b-a)^(c-a)).len();
73      }
74      //四面体有向面积 *6
75      double volume(Point3 a,Point3 b,Point3 c,Point3 d){
76          return ((b-a)^(c-a))*(d-a);
77      }
78      //正：点在面同向
79      double dblcmp(Point3 &p,face &f){
80          Point3 p1 = P[f.b] - P[f.a];
81          Point3 p2 = P[f.c] - P[f.a];
82          Point3 p3 = p - P[f.a];
83          return (p1^p2)*p3;
84      }
85      void deal(int p,int a,int b){
86          int f = g[a][b];
87          face add;
88          if(F[f].ok){
89              if(dblcmp(P[p],F[f]) > eps)
90                  dfs(p,f);
91              else {
92                  add.a = b;
93                  add.b = a;
94                  add.c = p;
95                  add.ok = true;
96                  g[p][b] = g[a][p] = g[b][a] = num;
97                  F[num++] = add;
98              }
99          }
100     }
101     //递归搜索所有应该从凸包内删除的面
102     void dfs(int p,int now){
103         F[now].ok = false;
104         deal(p,F[now].b,F[now].a);
105         deal(p,F[now].c,F[now].b);
106         deal(p,F[now].a,F[now].c);
107     }
108     bool same(int s,int t){
109         Point3 &a = P[F[s].a];
110         Point3 &b = P[F[s].b];
111         Point3 &c = P[F[s].c];
```

```
112         return fabs(volume(a,b,c,P[F[t].a])) < eps &&
113             fabs(volume(a,b,c,P[F[t].b])) < eps &&
114             fabs(volume(a,b,c,P[F[t].c])) < eps;
115     }
116     //构建三维凸包
117     void create(){
118         num = 0;
119         face add;
120 
121         //*********************************
122         //此段是为了保证前四个点不共面
123         bool flag = true;
124         for(int i = 1;i < n;i++){
125             if(!(P[0] == P[i])){
126                 swap(P[1],P[i]);
127                 flag = false;
128                 break;
129             }
130         }
131         if(flag)return;
132         flag = true;
133         for(int i = 2;i < n;i++){
134             if( ((P[1]-P[0])^(P[i]-P[0])).len() > eps ){
135                 swap(P[2],P[i]);
136                 flag = false;
137                 break;
138             }
139         }
140         if(flag)return;
141         flag = true;
142         for(int i = 3;i < n;i++){
143             if(fabs( ((P[1]-P[0])^(P[2]-P[0]))*(P[i]-P[0]) ) > eps)
                    {
144                 swap(P[3],P[i]);
145                 flag = false;
146                 break;
147             }
148         }
149         if(flag)return;
150         //*********************************
151 
152         for(int i = 0;i < 4;i++){
153             add.a = (i+1)%4;
154             add.b = (i+2)%4;
155             add.c = (i+3)%4;
156             add.ok = true;
157             if(dblcmp(P[i],add) > 0)swap(add.b,add.c);
158             g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] =
                    num;
159             F[num++] = add;
160         }
```

```
161         for(int i = 4;i < n;i++)
162             for(int j = 0;j < num;j++)
163                 if(F[j].ok && dblcmp(P[i],F[j]) > eps){
164                     dfs(i,j);
165                     break;
166                 }
167         int tmp = num;
168         num = 0;
169         for(int i = 0;i < tmp;i++)
170             if(F[i].ok)
171                 F[num++] = F[i];
172     }
173     //表面积
174     //测试：HDU3528
175     double area(){
176         double res = 0;
177         if(n == 3){
178             Point3 p = cross(P[0],P[1],P[2]);
179             return p.len()/2;
180         }
181         for(int i = 0;i < num;i++)
182             res += area(P[F[i].a],P[F[i].b],P[F[i].c]);
183         return res/2.0;
184     }
185     double volume(){
186         double res = 0;
187         Point3 tmp = Point3(0,0,0);
188         for(int i = 0;i < num;i++)
189             res += volume(tmp,P[F[i].a],P[F[i].b],P[F[i].c]);
190         return fabs(res/6);
191     }
192     //表面三角形个数
193     int triangle(){
194         return num;
195     }
196     //表面多边形个数
197     //测试：HDU3662
198     int polygon(){
199         int res = 0;
200         for(int i = 0;i < num;i++){
201             bool flag = true;
202             for(int j = 0;j < i;j++)
203                 if(same(i,j)){
204                     flag = 0;
205                     break;
206                 }
207             res += flag;
208         }
209         return res;
210     }
211     //重心
```

```
212     //测试：HDU4273
213     Point3 barycenter(){
214         Point3 ans = Point3(0,0,0);
215         Point3 o = Point3(0,0,0);
216         double all = 0;
217         for(int i = 0;i < num;i++){
218             double vol = volume(o,P[F[i].a],P[F[i].b],P[F[i].c]);
219             ans = ans + (((o+P[F[i].a]+P[F[i].b]+P[F[i].c])/4.0)*
                    vol);
220             all += vol;
221         }
222         ans = ans/all;
223         return ans;
224     }
225     //点到面的距离
226     //测试：HDU4273
227     double ptoface(Point3 p,int i){
228         double tmp1 = fabs(volume(P[F[i].a],P[F[i].b],P[F[i].c],p))
                ;
229         double tmp2 = ((P[F[i].b]-P[F[i].a])^(P[F[i].c]-P[F[i].a]))
                .len();
230         return tmp1/tmp2;
231     }
232 };
233 CH3D hull;
234 int main()
235 {
236     while(scanf("%d",&hull.n) == 1){
237         for(int i = 0;i < hull.n;i++)hull.P[i].input();
238         hull.create();
239         Point3 p = hull.barycenter();
240         double ans = 1e20;
241         for(int i = 0;i < hull.num;i++)
242             ans = min(ans,hull.ptoface(p,i));
243         printf("%.3lf\n",ans);
244     }
245     return 0;
246 }
```