//1.浮点数相关函数   
const int N = 5000;   
const double pi = 3.1415926535898;   
const double eps = 1e-10;   
inline int sgn(double x) {   
 if(fabs(x) < eps) return 0;   
 else return x > 0 ? 1 : -1;   
}   
inline int fcmp(double x, double y) {   
 if(fabs(x - y) < eps) return 0;   
 else return x > y ? 1 : -1;   
}   
inline double rd() { return (rand()) % 2 ? eps : -eps; }   
//2.平面向量类   
struct Point{   
 int x, y;   
 Point(){};   
 Point(int a, int b): x(a), y(b) {}   
 Point(Point a, Point b): x(b.x - a.x), y(b.y - a.y) {}   
 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 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);   
 }   
 friend int operator \* (const Point &a, const Point &b) {   
 return a.x \* b.y - a.y \* b.x;   
 }   
 friend int operator & (const Point &a, const Point &b) {   
 return a.x \* b.x + a.y \* b.y;   
 }   
 friend bool operator < (const Point &a, const Point &b) {   
 return fcmp(a.x, b.x ) == -1 ||   
 (fcmp(a.x, b.x)== 0 && fcmp(a.y, b.y) == -1);   
 }   
 friend bool operator == (const Point &a, const Point &b) {   
 return fcmp(a.x, b.x) == 0 && fcmp(a.y, b.y == 0);   
 }   
 inline double len() {   
 return sqrt(1.0 \* x \* x + 1.0 \* y \* y);   
 }   
 int sqr\_len() {   
 return x \* x + y \* y;   
 }   
 Point unit() {   
 return Point(x / len(), y / len());   
 }   
};   
typedef Point Vec;   
bool cmp1(Point a, Point b) {   
 return (fcmp(a.x, b.x) == -1) ||   
 (!fcmp(a.x, b.x) && fcmp(a.y, b.y) == -1);   
}   
bool cmp2(Point a, Point b) {   
 if(atan2(a.y, a.x) - atan2(b.y, b.x) == 0)   
 return a.x < b.x;   
 return atan2(a.y, a.x) < atan2(b.y, b.x);   
}   
inline double dis(Point &a, Point &b) {   
 return (a - b).len();   
}   
//平面凸包   
inline bool check(Point s1, Point s2, Point p) {   
 return Vec(s2, s1) \* Vec(s1, p) > 0;   
}   
int Convex\_hull\_2d(int n, Point \*p, Point \*ret) {   
 sort(p, p + n, cmp1);   
 int top = -1;   
 for (int i = 0; i < n; i++) {   
 while (top > 0 && !check(ret[top], ret[top - 1], p[i]))   
 top--;   
 ret[++top] = p[i];   
 }   
 int k = top;   
 for (int i = n - 2; i >= 0; i--) {   
 while (top > k && !check(ret[top], ret[top - 1], p[i]))   
 top--;   
 ret[++top] = p[i];   
 }   
 return top;   
}   
double Convex\_hull\_2d\_L(int n, Point \*p) {   
 Point convex[N];   
 int siz = Convex\_hull\_2d(n, p, convex);   
 double ans = dis(convex[0], convex[siz - 1]);   
 for (int i = 1; i < siz; i++)   
 ans += dis(convex[i - 1], convex[i]);   
 return ans;   
}   
double Convex\_hull\_2d\_S(int n, Point \*p) {   
 Point convex[N];   
 int siz = Convex\_hull\_2d(n, p, convex);   
 double ans = 0;   
 for (int i = 2; i < siz; i++)   
 ans += area(convex[0], convex[i - 1], convex[i]);   
 return ans;   
}   
//3.三维向量类   
struct Point3 {   
 double x, y, z;   
 Point3(){};   
 Point3(double a, double b, double c) : x(a), y(b), z(c) {}   
 Point3 operator + (const Point3 &b) {   
 return Point3(x + b.x, y + b.y, z + b.z);   
 }   
 Point3 operator - (const Point3 &b) {   
 return Point3(x - b.x, y - b.y, z - b.z);   
 }   
 Point3 operator \* (const Point3 &b) {   
 return Point3(y\*b.z - z\*b.y, z\*b.x - x\*b.z, x\*b.y - y\*b.x);   
 }   
 double operator & (const Point3 &b) {   
 return x \* b.x + y \* b.y + z \* b.z;   
 }   
 bool operator == (const Point3 &b) {   
 return fcmp(x, b.x) == 0 && fcmp(y, b.y) == 0 && fcmp(z, b.z) == 0;   
 }   
 double len() {   
 return sqrt(x \* x + y \* y + z \* z);   
 }   
 void shake() { x += rd(), y += rd(), z += rd(); }   
} p[N];   
//4.平面类   
struct plane{   
 int v[3];   
 plane(){};   
 plane(int a, int b, int c) { v[0] = a, v[1] = b, v[2] = c; }   
 Point3 normal() {   
 return (p[v[1]] - p[v[0]]) \* (p[v[2]] - p[v[0]]);   
 }   
 bool is\_above(Point3 A) {   
 return (normal() & (A - p[v[0]])) >= 0;   
 }   
 double area() {   
 return normal().len() / 2.0;   
 }   
};   
int Convex\_hull\_3d(int n, plane \*ret) {   
 plane tmp[N];   
 bool g[N][N];   
 for (int i = 0; i < n; i++) p[i].shake();   
 int top = -1;   
 ret[++top] = plane(0, 1, 2);   
 ret[++top] = plane(0, 2, 1);   
 for (int i = 3; i < n; i++)   
 {   
 int cnt = -1;   
 for (int j = 0; j <= top; j++)   
 {   
 bool flag = ret[j].is\_above(p[i]);   
 if (!flag)   
 tmp[++cnt] = ret[j];   
 for (int k = 0; k < 3; k++)   
 g[ret[j].v[k]][ret[j].v[(k + 1) % 3]] = flag;   
 }   
 for (int j = 0; j <= top; j++)   
 {   
 for (int k = 0; k < 3; k++)   
 {   
 int a = ret[j].v[k], b = ret[j].v[(k + 1) % 3];   
 if (g[a][b] && !g[b][a])   
 tmp[++cnt] = plane(a, b, i);   
 }   
 }   
 for (int j = 0; j <= cnt; j++) ret[j] = tmp[j];   
 top = cnt;   
 }   
 return (top + 1);   
}   
double Convex\_hull\_3d\_S(int n) {   
 plane convex[N];   
 int siz = Convex\_hull\_3d(n, convex);   
 double ret = 0;   
 for (int i = 0; i < siz; i++) ret += convex[i].area();   
 return ret;   
}   
   
//5.旋转卡壳   
//凸包直径   
inline int sqr\_dis(Point a, Point b) {   
 return (a - b).sqr\_len();   
}   
int Get\_Max(int n, Point \*ch) {//传入convex-hull   
 int ret = 0;   
 ch[n] = ch[0];   
 int j = 1;   
 for(int i = 0; i < n; i++) {   
 while((ch[i] - ch[j+1]) \* (ch[i+1] - ch[j+1]) >   
 (ch[i] - ch[j ]) \* (ch[i+1] - ch[j ]))   
 j = (j + 1) % n;   
 ret = max(ret, max(sqr\_dis(ch[i], ch[j]), sqr\_dis(ch[i+1], ch[j])));   
 }   
 return ret;   
}   
//最小矩形覆盖   
double Get\_Min(int n, Point \*ch, Point \*Ans) {   
 ch[n] = ch[0];   
 int u = 2, l, r = 2;   
 //u是距离AB最远的点；在AB为底时，l和r是两个最靠边的点   
 double ret = 1e100, H, L, R, S;   
 for(int i = 0; i < n; i++) {   
 Point A = ch[i], B = ch[i+1]; Vec AB = B - A, BA = A - B;   
 while((AB \* Vec(B, ch[u+1])) >= (AB \* Vec(B, ch[u])))   
 u = (u + 1) % n;   
 while((AB & Vec(B, ch[r+1])) >= (AB & Vec(B, ch[r])))   
 r = (r + 1) % n;   
 if(i == 0) l = r;   
 while((AB & Vec(B, ch[l+1])) <= (AB & Vec(B, ch[l])))   
 l = (l + 1) % n;   
 H = (AB \* Vec(B, ch[u])) / AB.len(); //以AB所在直线为底边，矩形的高   
 L = (BA & Vec(A, ch[l])) / BA.len(); //A距离左侧顶点的距离   
 R = (AB & Vec(B, ch[r])) / AB.len(); //B距离右侧顶点的距离   
 S = H \* (L + AB.len() + R); //矩形面积   
 if(S < ret) { //求矩形顶点坐标   
 ret = S;   
 Ans[0] = A + L \* BA.unit();   
 Ans[1] = B + R \* AB.unit();   
 Ans[2] = Ans[1] + H \* (ch[r]-Ans[1]).unit();   
 Ans[3] = Ans[0] + H \* (ch[l]-Ans[0]).unit();   
 }   
 }   
 return ret;   
}   
//6.直线类   
struct Line{   
 Point s, t;   
 Line() {};   
 Line(Point a, Point b) : s(a), t(b) {}   
 double ang() { return atan2((t - s).y, (t - s).x); };   
 friend bool parallel(const Line &A, const Line &B) {   
 return sgn((A.s - A.t) \* (B.s - B.t)) == 0;   
 }   
 friend bool Calc\_intersection(Line &A, Line &B, Point &res) {   
 if(parallel(A, B)) return false;   
 double s1 = (B.t - B.s) \* (B.s - A.s);   
 double s2 = (B.t - B.s) \* (A.t - A.s);   
 res = A.s + (A.t - A.s) \* (s1 / s2);   
 return true;   
 }   
} L[N];   
inline bool cmp\_Line(Line A, Line B) {   
 //极角相等时，位置靠右的排在前面   
 if(!sgn(A.ang() - B.ang())) return (A.t - A.s) \* (B.t - A.s) > 0;   
 return A.ang() < B.ang();   
}   
void Halfplane\_unique(int &n, Line \*hp) {   
 int m = 0;   
 for(int i = 0; i < n - 1; i++) {   
 if(sgn(hp[i].ang() - hp[i + 1].ang()) == 0) continue;   
 hp[m++] = hp[i];   
 }   
 hp[m++] = hp[n - 1];   
 n = m;   
}   
int que[N], st, ed;   
bool Halfplane\_intersection(int n, Line \*hp, Point \*p) {   
 if(n < 3) return false;   
 sort(hp, hp + n, cmp\_Line);   
 Halfplane\_unique(n, hp);   
 st = 0; ed = 1;   
 que[0] = 0; que[1] = 1;   
 if(parallel(hp[0], hp[1])) return false;   
 Calc\_intersection(hp[0], hp[1], p[1]);   
 for(int i = 2; i < n; i++) {   
 while(st < ed &&   
 sgn((hp[i].t - hp[i].s) \* (p[ed] - hp[i].s)) < 0)   
 ed--;   
 while(st < ed &&   
 sgn((hp[i].t - hp[i].s) \* (p[st + 1] - hp[i].s)) < 0)   
 st++;   
 que[++ed] = i;   
 if(parallel(hp[i], hp[que[ed - 1]])) return false;   
 Calc\_intersection(hp[i], hp[que[ed - 1]], p[ed]);   
 }   
 while(st < ed &&   
 sgn((hp[que[st]].t - hp[que[st]].s) \* (p[ed] - hp[que[st]].s)) < 0)   
 ed--;   
 while(st < ed &&   
 sgn((hp[que[ed]].t - hp[que[ed]].s) \* (p[st + 1] - hp[que[ed]].s)) < 0)   
 st++;   
 if(st + 1 >= ed) return false;   
 return true;   
}   
int Get\_convex\_hull(Line \*hp, Point \*p, Point \*ch) {   
 Calc\_intersection(hp[que[st]], hp[que[ed]], p[st]);   
 for(int i = 0, j = st; j <= ed; i++, j++) ch[i] = p[j];   
 return ed - st + 1;   
}   
double area(Point &a, Point &b, Point &c) {   
 return (b - a) \* (c - a) / 2.0;   
}   
double Calc\_area(int n, Point \*ch) {   
 double ans = 0;   
 for (int i = 2; i < n; i++)   
 ans += area(ch[0], ch[i - 1], ch[i]);   
 return ans;   
}