开头的东西

```
const double eps = 1e-8;
const double pi = acos(-1.0);
const int maxn = 1005;
inline int sgn(double x) { return (x > eps) - (x < -eps); }
int inmid(double k1, double k2, double k3) { return sgn(k1 - k3) * sgn(k2 - k3) <= 0; }
```

点

```
struct point {
    double x, y;
    point() { }
    point(double _x, double _y) : x(_x), y(_y) { }
    bool operator==(const point& b) const { return sgn(x - b.x) == 0 && sgn(y - b.y) ==
0; }
    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); }
    bool operator<(const point& b) const { return sgn(x - b.x) == 0 ? sgn(y - b.y) < 0
: x < b.x; }
    double operator*(const point& b) const { return x * b.x + y * b.y; } //点积
    double operator^(const point& b) const { return x * b.y - y * b.x; } //叉积
    double len() { return hypot(x, y); } //返回到原点长度
    double len2() { return x * x + y * y; } //返回到原点长度平方
    double dis(const point& b) const { return hypot(x - b.x, y - b.y); } //返回两点距离
    point operator*(const double& b) const { return point(x * b, y * b); }
    point operator/(const double& b) const { return point(x / b, y / b); }
    point left() { return point(-y, x); } //逆时针90度
    point right() { return point(y, -x); } //顺时针90度
    bool inPi() const { return sgn(y) == 1 || (sgn(y) == 0 && sgn(x) == -1); } // 0 ->
pi
    point unit(const double& r = 1.0) //默认为单位化向量，参数为化长度为r的向量
    {
        double l = len();
        return sgn(l) == 0 ? point(r, 0) : point(x * r / l, y * r / l);
    }
};
int inmid(point p1, point p2, point p3)
{
    return inmid(p1.x, p2.x, p3.x) && inmid(p1.y, p2.y, p3.y);
}
int cmpAng(const point& a, const point& b)
{
    return a.inPi() < b.inPi() || (a.inPi() == b.inPi() && sgn(a ^ b) > 0);
```

```
}
// 直线和直线是否有交点
bool checkLineLine(point p1, point p2, point p3, point p4)
{
    return sgn(((p3 - p1) ^ (p4 - p1)) - ((p3 - p2) ^ (p4 - p2))) != 0;
}
// 直线和直线的交点
point getLineLine(point p1, point p2, point p3, point p4)
{
    double w1 = (p1 - p3) ^ (p4 - p3), w2 = (p4 - p3) ^ (p2 - p3);
    return (p1 * w2 + p2 * w1) / (w1 + w2);
}
int intersect(double l1, double r1, double l2, double r2)
{
    if (l1 > r1) swap(l1, r1);
    if (l2 > r2) swap(l2, r2);
    return sgn(r1 - l2) != -1 && sgn(r2 - l1) != -1;
}
// 线段和线段是否有交点
int checkSegSeg(point k1, point k2, point k3, point k4)
{
    return intersect(k1.x, k2.x, k3.x, k4.x) && intersect(k1.y, k2.y, k3.y, k4.y)
        && sgn((k3 - k1) ^ (k4 - k1)) * sgn((k3 - k2) ^ (k4 - k2)) <= 0
        && sgn((k1 - k3) ^ (k2 - k3)) * sgn((k1 - k4) ^ (k2 - k4)) <= 0;
}
point proj(point p1, point p2, point p3)
{ // p3到直线p1,p2的投影
    point k = p2 - p1;
    return p1 + k * ((p3 - p1) * k) / k.len2();
}
point reflect(point p1, point p2, point p3)
{ // p3对直线p1,p2对称点
    return proj(p1, p2, p3) * 2 - p3;
}
int clockwise(point p1, point p2, point p3)
{ // p1p2p3顺时针1, 逆时针-1, other 0
    return sgn((p2 - p1) ^ (p3 - p1));
}
bool onS(point a, point b, point c)
{ // c 在 ab 之间
    return inmid(a, b, c) && sgn((a - c) ^ (b - a)) == 0;
}
// 点到线段的距离
double disSegPt(point p1, point p2, point q)
{
    point p3 = proj(p1, p2, q);
    if (inmid(p1, p2, p3)) return p3.dis(q);
    return min(p1.dis(q), p2.dis(q));
}
```

```
// 线段到线段的距离
double disSegSeg(point k1, point k2, point k3, point k4)
{
    if (checkSegSeg(k1, k2, k3, k4))
        return 0;
    else
        return min(min(disSegPt(k1, k2, k3), disSegPt(k1, k2, k4)),
            min(disSegPt(k3, k4, k1), disSegPt(k3, k4, k2)));
}
```

线

```
struct line {
    point s, e;
    line() { }
    line(point _s, point _e) : s(_s), e(_e) { }
    // 点和倾斜角构造直线
    line(point p, double ang)
    {
        s = p;
        if (sgn(ang - pi / 2) == 0)
            e = (s + point(0, 1));
        else
            e = (s + point(1, tan(ang)));
    }
    // 构造ax + by = c
    line(double a, double b, double c)
    {
        if (sgn(a) == 0) {
            s = point(0, -c / b);
            e = point(1, -c / b);
        } else if (sgn(b) == 0) {
            s = point(-c / a, 0);
            e = point(-c / a, 1);
        } else {
            s = point(0, -c / b);
            e = point(1, (-c - a) / b);
        }
    }
    bool operator==(const line& v) { return (s == v.s) && (e == v.e); }
    double len() { return s.dis(e); } // 返回长度
    double ang() // 返回倾斜角
    {
        double k = atan2(e.y - s.y, e.x - s.x);
        if (sgn(k) < 0) k += pi;
        if (sgn(k - pi) == 0) k -= pi;
        return k;
    }
```

```
    point dir() { return e - s; }
    int relation(const point& p) //点和直线关系 1 左 0 上 -1 右
    {
        int c = sgn((e - s) ^ (p - s));
        if (c < 0) return 1;
        else if (c > 0) return -1;
        else return 0;
    }
    bool pointonseg(const point& p) // 点在线段上
    {
        return sgn((p - s) ^ (e - s)) == 0 && sgn((p - s) * (p - e)) <= 0;
    }
    bool parallel(line v) { return sgn((e - s) ^ (v.e - v.s)) == 0; } // 直线平行 (重合也是平行)
    // 判重合就parallel 然后 relation = 0
};
bool sameDir(line l1, line l2) { return l1.parallel(l2) && sgn(l1.dir() * l2.dir()) == 1; }
bool operator<(line l1, line l2)
{
    if (sameDir(l1, l2)) return l2.relation(l1.s);
    return cmpAng(l1.dir(), l2.dir());
}
// 直线和直线交点
point getLineLine(line l1, line l2) { return getLineLine(l1.s, l1.e, l2.s, l2.e); }
// l3关于l1和l2的交点关系
int checkpos(line l1, line l2, line l3) { return l3.relation(getLineLine(l1, l2)); }
```

圆

```
struct circle {
    point o;
    double r;
    circle() { }
    circle(point _o, double _r) : o(_o), r(_r) { }
    int inside(point k) { return sgn(r - o.dis(k)); }
};
circle getcircle(point k1, point k2, point k3)
{ // 三点求圆
    double a1 = k2.x - k1.x, b1 = k2.y - k1.y, c1 = (a1 * a1 + b1 * b1) / 2;
    double a2 = k3.x - k1.x, b2 = k3.y - k1.y, c2 = (a2 * a2 + b2 * b2) / 2;
    double d = a1 * b2 - a2 * b1;
    point o = point(k1.x + (c1 * b2 - c2 * b1) / d, k1.y + (a1 * c2 - a2 * c1) / d);
    return circle(o, k1.dis(o));
}
```

```
circle getScircle(vector<point> A)
{ // 随机增量法 求最小圆覆盖
    random_shuffle(A.begin(), A.end());
    circle ans = circle(A[0], 0);
    for (int i = 1; i < A.size(); i++) {
        if (ans.inside(A[i]) == -1) {
            ans = circle({ A[i], 0 });
            for (int j = 0; j < i; j++) {
                if (ans.inside(A[j]) == -1) {
                    ans.o = (A[i] + A[j]) / 2.0;
                    ans.r = ans.o.dis(A[i]);
                    for (int k = 0; k < j; k++) {
                        if (ans.inside(A[k]) == -1) { ans = getcircle(A[i], A[j],
A[k]); }
                    }
                }
            }
        }
    }
    return ans;
}

// 模拟退火 究极版
#include <bits/stdc++.h>
using namespace std;
const double eps = 1e-10;
const double PI = acos(-1.0);
const int maxn = 205;
int n, k;
mt19937 rnd(time(0));

struct Point{
    double x,y;
}p[maxn];
double dist(Point a,Point b){
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
double Rand()
{
    return (rand() % 1000 + 1.0) / 1000.0;
}
double cal(Point tmp)
{
    //cout << tmp.x << ' ' << tmp.y << ' ';
    double sum = 0;
    vector<double> d;
    for(int i = 0; i < n; i++) d.push_back(dist(tmp, p[i]));
    sort(d.begin(), d.end());
    for (int i = n - 1; i >= n - k; i--) {
```

```
        sum += d[i];
    }
    //cout << sum << endl;
    return sum;
}
double solve(int n){
    double ans=1e30;
    Point tmp,nxt;
    tmp.x=tmp.y=0;
    double step=100000;
    while(step>eps){
        double al = 2.0 * PI * Rand();
        nxt.x = tmp.x + cos(al) * step;
        nxt.y = tmp.y + sin(al) * step;
        double now = cal(nxt);
        double del = now - ans;
        if (del < eps) {
            tmp.x = nxt.x, tmp.y = nxt.y;
            ans = now;
        }
        //cout << ans << endl;
        step*=0.998;
    }
    return ans;
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> k;
    for (int i = 0; i < n; i++) cin >> p[i].x >> p[i].y;
    double ans = solve(n);
    cout << fixed << setprecision(10) << ans << endl;
}
```

多边形

```
double area(vector<point> p)  //多边形面积
{
    double ans = 0;
    for (int i = 0; i < p.size(); i++) ans += p[i] ^ p[(i + 1) % p.size()];
    return ans / 2;
}

double perimeter(vector<point> p)  // 多边形周长
{
```

```
    double ans = 0;
    for (int i = 0; i < p.size(); i++) ans += p[i].dis(p[(i + 1) % p.size()]);
    return ans;
}

bool checkConvex(vector<point> p)  // 判断凸
{
    int n = p.size();
    p.push_back(p[0]);
    p.push_back(p[1]);
    for (int i = 0; i < n; i++)
        if (sgn((p[i + 1] - p[i]) * (p[i + 2] - p[i])) == -1) return 0;
    return 1;
}

int contain(vector<point> p, point q)
{  // 0 外部 1 边界 2 内部
    int pd = 0;
    p.push_back(p[0]);
    for (int i = 1; i < p.size(); i++) {
        point u = p[i - 1], v = p[i];
        line l(u, v);
        if (l.pointonseg(q)) return 1;
        if (sgn(u.y - v.y) > 0) swap(u, v);
        if (sgn(u.y - q.y) >= 0 || sgn(v.y - q.y) < 0) continue;
        if (sgn((u - v) ^ (q - v)) < 0) pd ^= 1;
    }
    return pd << 1;
}

vector<point> getConvex(vector<point> p, int flag = 1)  // 凸包
{                                                       // flag = 1严格
    int n = p.size();
    vector<point> ans(2 * n);
    sort(p.begin(), p.end());
    int now = -1;
    for (int i = 0; i < p.size(); i++) {
        while (now > 0 && sgn((ans[now] - ans[now - 1]) ^ (p[i] - ans[now - 1])) <
flag) now--;
        ans[++now] = p[i];
    }
    int pre = now;
    for (int i = n - 2; i >= 0; i--) {
        while (now > pre && sgn((ans[now] - ans[now - 1]) ^ (p[i] - ans[now - 1])) <
flag) now--;
        ans[++now] = p[i];
    }
    ans.resize(now);
    return ans;
```

```
}

double convexDiameter(vector<point> p)  //凸包直径
{
    int now = 0, n = p.size();
    double ans = 0;
    for (int i = 0; i < p.size(); i++) {
        now = max(now, i);
        while (1) {
            double k1 = p[i].dis(p[now % n]), k2 = p[i].dis(p[(now + 1) % n]);
            ans = max(ans, max(k1, k2));
            if (k2 > k1)
                now++;
            else
                break;
        }
    }
    return ans;
}
```

其他

```
vector<line> getHPI(vector<line> &L)
{  // 半平面交 按逆时针输出
    sort(L.begin(), L.end());
    deque<line> q;
    for (int i = 0; i < (int)L.size(); i++) {
        if (i && sameDir(L[i], L[i - 1])) continue;
        while (q.size() > 1 && !checkpos(q[q.size() - 2], q[q.size() - 1], L[i]))
q.pop_back();
        while (q.size() > 1 && !checkpos(q[1], q[0], L[i])) q.pop_front();
        q.push_back(L[i]);
    }
    while (q.size() > 2 && !checkpos(q[q.size() - 2], q[q.size() - 1], q[0]))
q.pop_back();
    while (q.size() > 2 && !checkpos(q[1], q[0], q[q.size() - 1])) q.pop_front();
    vector<line> ans;
    for (int i = 0; i < q.size(); i++) ans.push_back(q[i]);
    return ans;
}

double closepoint(vector<point> &p, int l, int r)
{  // 最近点对, 先按x排序
    if (r - l <= 5) {
        double ans = 1e20;
        for (int i = l; i <= r; i++) {
            for (int j = i + 1; j <= r; j++) {
                ans = min(ans, p[i].dis(p[j]));
```

```
            }
        }
        return ans;
    }
    int mid = (l + r) >> 1;
    double ans = min(closepoint(p, l, mid), closepoint(p, mid + 1, r));
    vector<point> B;
    for (int i = l; i <= r; i++) {
        if (abs(p[i].x - p[mid].x) <= ans) B.push_back(p[i]);
    }
    sort(B.begin(), B.end(), [](point k1, point k2) { return k1.y < k2.y; });
    for (int i = 0; i < B.size(); i++) {
        for (int j = i + 1; j < B.size() && B[j].y - B[i].y < ans; j++) {
            ans = min(ans, B[i].dis(B[j]));
        }
    }
    return ans;
}
```