最大凸包(convex,1s,512MB)

【算法分析】

考虑钦定一个凸包上y坐标最小（相同情况下x坐标最小）的点O，并以O为原点把所有y坐标大于O（或y坐标等于O但x坐标大于O）的点极角排序，考虑点集P′合法的条件：

（1） 对于点集P′内任意极角序相邻的三个点$i,j,k \neq O$，满足$\overrightarrow{ij} \times \overrightarrow{ik} > 0$。

（2） 对于点集P′内极角序最小的两个点$i,j \neq O$满足$\overrightarrow{Oi} \times \overrightarrow{Oj} > 0$，最大的两个点同理。

（3） 对于点集P′内任意极角序相邻的两个点$i,j \neq O$，满足$\triangle Oij$的内部（不包括边界）不包含任何点。

（4） 对于点集P′内任意点$i \neq O$且i不为第一个或最后一个点，满足线段Oi上（不包括端点）不包含任何点。

限制（1）（2）保证了是一个凸包，（3）（4）保证了凸包内没有任何其他点。

于是利用这四个限制进行 DP：$f[i][j]$表示现在到了点i并且上一个插入的点为j的最大面积的2倍。

初始化：对于所有y坐标大于O（或y坐标等于O但x坐标大于O）的点i，$f[i][O] = 0$。

转移时枚举i,j。为了满足限制（4），这里如果$j \neq O$且线段Oi上有其他点，那么这次就不能转移。

枚举一个k满足k的极角序在i之后。判断如果$\overrightarrow{ji} \times \overrightarrow{jk} > 0$且$\triangle Oik$内没有其他任何点则可以转移，即令$f[k][i] \leftarrow \max(f[k][i], f[i][j] + \overrightarrow{Oi} \times \overrightarrow{Ok})$。

更新答案即枚举i,j之后如果$\overrightarrow{ji} \times \overrightarrow{jO} > 0$则$ans \leftarrow \max(ans, f[j][i]/2)$。

理论上复杂度为$O(Tn^4)$，但这个 DP 的常数很小，且钦定点O之后y坐标大于O（或y坐标等于O但x坐标大于O）的点数通常都会远小于n，所以实际运行效率很快。

【参考程序】

```cpp
// 陈栉旷
#include <bits/stdc++.h>

template <class T>
inline void read(T &res)
{
    res = 0; bool bo = 0; char c;
    while (((c = getchar()) < '0' || c > '9') && c != '-');
    if (c == '-') bo = 1; else res = c - 48;
    while ((c = getchar()) >= '0' && c <= '9')
        res = (res << 3) + (res << 1) + (c - 48);
    if (bo) res = ~res + 1;
}

template <class T>
inline T Max(const T &a, const T &b) {return a > b ? a : b;}

const int N = 105;

int n, m, f[N][N], ans;
bool is[N][N][N];
```

```
std::bitset<N> li[N][N];

struct point
{
    int x, y, id;

    friend inline point operator - (point a, point b)
    {
        return (point) {b.x - a.x, b.y - a.y, 0};
    }

    friend inline int operator * (point a, point b)
    {
        return a.x * b.y - a.y * b.x;
    }

    inline int len()
    {
        return x * x + y * y;
    }
} a[N], b[N], OC;

inline bool comp(point a, point b) // 极角排序，第二关键字为到原点的距离
{
    return (OC - a) * (OC - b) > 0
        || ((OC - a) * (OC - b) == 0 && (OC - a).len() < (OC - b).len());
}

void jiejuediao(int O) // DP
{
    m = 0;
    memset(f, -1, sizeof(f));
    for (int i = 1; i <= n; i++)
        if (i != O && (a[i].y > a[O].y || (a[i].y == a[O].y && a[i].x >
a[O].x)))
            b[++m] = a[i];
    b[0] = OC = a[O]; std::sort(b + 1, b + m + 1, comp);
    for (int i = 1; i <= m; i++) f[b[i].id][OC.id] = 0;
    for (int i = 1; i <= m; i++)
    {
        int u = b[i].id;
        for (int j = 0; j < i; j++)
        {
            ans = Max(ans, f[u][b[j].id]);
```

```
            if ((!j || (OC - b[i - 1]) * (OC - b[i]) > 0) && f[u][b[j].id] !=
-1) // 由于极角排序的第二关键字是到原点的距离，所以判断线段 Oi 上是否有其他点，
// 只需判断 O、i – 1、i 三点是否共线
                for (int k = i + 1; k <= m; k++)
                    if  ((b[k]  -  b[i])  *  (b[k]  -  b[j])  <  0  &&
(li[OC.id][b[k].id]
                        & li[b[k].id][u] & li[u][OC.id]).count() == 0)
                         f[b[k].id][u] = Max(f[b[k].id][u], f[u][b[j].id]
                             + (OC - b[i]) * (OC - b[k]));
        }
    }
}

void work()
{
    read(n); ans = 0;
    for (int i = 1; i <= n; i++) read(a[i].x), read(a[i].y), a[i].id = i;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++) if (i != j)
        {
            li[i][j].reset();
            for (int k = 1; k <= n; k++) if (i != j && i != k
                && (a[i] - a[j]) * (a[i] - a[k]) < 0)
                    li[i][j][k] = 1;
        }
// li[i][j]为用 bitset 储存的一个集合，表示向量 ij 右侧的点构成的集合
// 判断三角形 ijk（点按顺时针顺序），
// 只需判断 li[i][j] & li[j][k] & li[k][i] 是否为空
    for (int i = 1; i <= n; i++) jiejuediao(i);
    printf("%.1lf\n", 0.5 * ans);
}

int main()
{
    #ifdef nealchentxdy
    #else
        freopen("convex.in", "r", stdin);
        freopen("convex.out", "w", stdout);
    #endif

    int T; read(T);
    while (T--) work();
    return 0;
}
```

鸽子(lo,1s,512MB)

【算法分析】

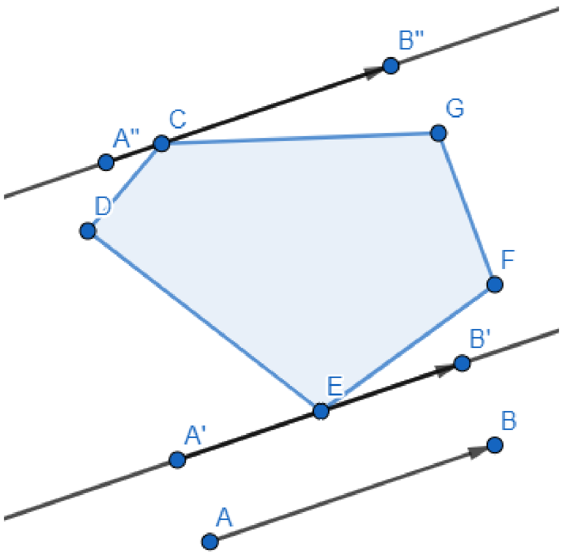
容易发现，题目所求的即为恰好能包含所有鸽子的多边形的最小点数。

新建一张图，每个监视器对应图中的一个点，点 A 向点 B 连边当且仅当任意一只鸽子 C 都满足 $\overrightarrow{AB}\times\overrightarrow{AC}>0$（即点 C 在 $\overrightarrow{AB}$ 的左侧），则多边形的最小点数即为新图中的最小环，可以暴力 Floyd 求出。

现在我们只需要快速求得这张有向图，即快速判断所有鸽子是否都在一个向量的左侧。

对于向量 $\overrightarrow{AB}$，我们只需要判断到直线 AB 距离最远的点是否满足条件，而这样的点一定在所有鸽子构成的凸包上。

进一步地，我们先作出直线 AB，然后将直线 AB 从无穷远处移向凸包，直线 AB 会恰好与凸包相切两次，每次相切我们都取出切点的坐标（如果相切的部分是一条线段则任取一个切点），而到直线 AB 距离最远的点一定恰好是两个切点中的一个，如下图所示。



如何求出切点的坐标呢？我们需要利用凸包的一些优秀性质。考虑取出凸包中横坐标最小和最大的两个点，把凸包分为上下两个部分，这两个部分一定恰好是上下凸壳，相邻两点所在直线的斜率满足单调性。以下凸壳为例，如上图所示，将直线的斜率用 k 表示，点 E 在凸壳上的上一个点为 D，下一个点为 F，直线 AB 在凸壳上的切点为 E 当且仅当 $k_{AB}\ge k_{DE}$且$k_{AB}\le k_{EF}$，因此切点可以直接在凸壳上二分得到，上凸壳的情况同理。

注意凸包上横坐标最小和最大的两个点需要特殊处理，为了方便可以直接把它们单独拿出来和 $\overrightarrow{AB}$ 作判断。

时间复杂度 $O(n\log n+m^2\log n+m^3)$。

【参考程序】

```
//陈贤
#include <bits/stdc++.h>

template <class T>
inline void read(T &res)
{
```

```
    char ch; bool flag = false; res = 0;
    while (ch = getchar(), !isdigit(ch) && ch != '-');
    ch == '-' ? flag = true : res = ch ^ 48;
    while (ch = getchar(), isdigit(ch))
        res = res * 10 + ch - 48;
    flag ? res = -res : 0;
}

typedef long long ll;
const int N = 1e5 + 5;
const int Maxn = 2e9 / 3;
const int M = 505;
int cur[N], stk[N], cur1[N], cur2[N], f[M][M], g[M][M];
int ans = Maxn, top, cm, n, m, m1, m2;

template <class T>
inline void CkMin(T &x, T y) {x > y ? x = y : 0;}

struct point
{
    ll x, y;

    point() {}
    point(ll X, ll Y):
        x(X), y(Y) {}

    inline void scan()
    {
        read(x);
        read(y);
    }

    inline ll norm()
    {
        return x * x + y * y;
    }

    inline point operator + (const point &a) const
    {
        return point(x + a.x, y + a.y);
    }

    inline point operator - (const point &a) const
    {
```

```
        return point(x - a.x, y - a.y);
    }

    inline ll operator * (const point &a) const
    {
        return x * a.y - y * a.x;
    }

    inline bool operator < (const point &a) const
    {
        return x < a.x || x == a.x && y < a.y;
    }
}p[N], q[M];

inline bool cmp(const int &a, const int &b)
{
    point ta = p[a] - p[1],
          tb = p[b] - p[1];
    ll tmp = ta * tb;
    return tmp == 0 ? (ta.norm() < tb.norm()) : (tmp < 0);
}

inline bool check(const point &a, const point &b)
{ //在上下凸壳二分求切点，判断是否能够连边
    point c = b - a;
    if ((p[1] - a) * c > 0)
        return false;
    if ((p[cur1[m1]] - a) * c > 0)
        return false;
    int l = 2, r = m1 - 1, res1 = 0, res2 = 0;
    while (l <= r)
    {
        int mid = l + r >> 1;
        point tmp = p[cur1[mid + 1]] - p[cur1[mid]];
        if (c.y / (double)c.x >= tmp.y / (double)tmp.x)
            res1 = cur1[mid], r = mid - 1;
        else
            l = mid + 1;
    }
    if (res1 && (p[res1] - a) * c > 0)
        return false;

    l = 2, r = m2 - 1;
    while (l <= r)
```

```
    {
        int mid = l + r >> 1;
        point tmp = p[cur2[mid + 1]] - p[cur2[mid]];
        if (c.y / (double)c.x <= tmp.y / (double)tmp.x)
            res2 = cur2[mid], r = mid - 1;
        else
            l = mid + 1;
    }
    if (res2 && (p[res2] - a) * c > 0)
        return false;
    return true;
}

int main()
{
    freopen("lo.in", "r", stdin);
    freopen("lo.out", "w", stdout);

    read(n); read(m);
    for (int i = 1; i <= n; ++i)
        p[i].scan();
    for (int i = 1; i <= m; ++i)
        q[i].scan();
    int st = 1;
    for (int i = 2; i <= n; ++i)
        if (p[i] < p[st])
            st = i;
    if (st != 1)
        std::swap(p[st], p[1]);
    for (int i = 2; i <= n; ++i)
        cur[++cm] = i;
    std::sort(cur + 1, cur + cm + 1, cmp);
    stk[top = 1] = 1;
    for (int i = 1; i <= cm; ++i)
    { //求出所有鸽子的凸包
        while (top > 1 && (p[cur[i]] - p[stk[top - 1]]) * (p[stk[top]] -
p[stk[top - 1]]) <= 0)
            --top;
        stk[++top] = cur[i];
    }
    for (int i = 1; i <= m; ++i)
        for (int j = 1; j <= m; ++j)
            f[i][j] = Maxn;
```

```
    int ed = 1;
    for (int i = 2; i <= top; ++i)
        if (p[stk[ed]] < p[stk[i]])
            ed = i;
    for (int i = 1; i <= ed; ++i)
        cur1[++m1] = stk[i];
    cur2[++m2] = 1;
    for (int i = top; i >= ed; --i)
        cur2[++m2] = stk[i];

    for (int i = 1; i <= m; ++i) //构造有向图
        for (int j = 1; j <= m; ++j)
            if (i != j && check(q[i], q[j]))
                f[i][j] = 1;
    for (int i = 1; i <= m; ++i)
        for (int j = 1; j <= m; ++j)
            g[i][j] = f[i][j];
    for (int k = 1; k <= m; ++k)
    { //Floyd 求最小环
        for (int i = 1; i < k; ++i)
            for (int j = 1; j < k; ++j)
                if (i != j)
                    CkMin(ans, g[i][k] + g[k][j] + f[j][i]);
        for (int i = 1; i <= m; ++i)
            for (int j = 1; j <= m; ++j)
                CkMin(f[i][j], f[i][k] + f[k][j]);
    }
    printf("%d\n", ans == Maxn ? -1 : ans);

    fclose(stdin); fclose(stdout);
    return 0;
}
```

构造线段(disanti,1s,512MB)

【算法分析】

先求出这 $n+m$ 个点的凸包。

对于凸包上相邻的两个点 A,B，若 A,B 同类则连接 A,B。此时凸包上的同类点必定处于同一个连通块中，否则无解。

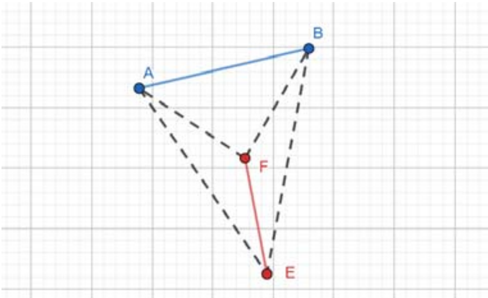
记凸包上的点的集合为 C，其它点的集合为 D，接下来对点的种类讨论：

1.C 的种类数为 1

(1). D 的种类数为 1

在 C 中任意取一点 E，把 D 中的所有点都和 E 连边即可。

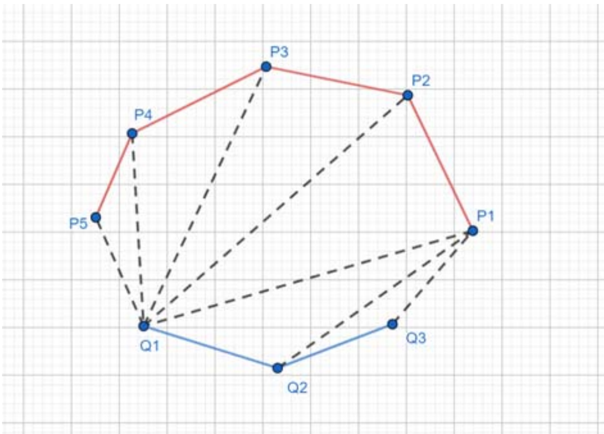
(2). D 的种类数为 2



如图，在 D 中任意取一和 C 中的点不同类的点 E。接下来，对于凸包上的每对相邻点 A,B，对 $\triangle ABE$ 执行如下过程：

考虑 $\triangle ABE$ 内部的点，如果种类全部相同，就都和 A,B,E 中的某个点连边。否则，在 $\triangle ABE$ 中找到一个和 E 同类的点 F，连接 E,F，然后递归处理 $\triangle ABF$,$\triangle AEF$,$\triangle BEF$。

2.C 的种类数为 2



如图，记 C 中的点 A 类点按逆时针顺序分别为 $P_1,P_2,\ldots,P_m$，C 中的B 类点按逆时针顺序分别为 $Q_1,Q_2,\ldots,Q_k$。那么把 $\triangle P_1P_2Q_1,\triangle P_2P_3Q_1,\ldots,\triangle P_{m-1}P_mQ_1,\triangle Q_1Q_2P_1,\triangle Q_2Q_3P_1,\ldots,\triangle Q_{k-1}Q_kP_1$ 分别调用上述过程即可。

时间复杂度 $O((n+m)^2)$。

【参考程序】

```
// 陈予菲
#include <bits/stdc++.h>

using namespace std;
```

```
#define ll long long

template <class T>
inline void read(T & res)
{
    char ch;
    while (ch = getchar(), !isdigit(ch));
    res = ch ^ 48;
    while (ch = getchar(), isdigit(ch))
    res = res * 10 + (ch ^ 48);
}

template <class T>
inline void print(T x)
{
    if (x > 9) print(x / 10);
    putchar(x % 10 + 48);
}

const int N = 1e4 + 5;

int n, m, cnt, top, stk[N], b[N], id[N], src, c[N], flag, d[N], tot, mid;
bool in[N];

struct line
{
    int x, y;
    line(){}
    line(int _x, int _y) :
        x(_x), y(_y) {}
}a[N];

struct point
{
    ll x, y;
    point(){}
    point(ll _x, ll _y) :
        x(_x), y(_y) {}
}p[N];

inline point operator - (point a, point b)
{
    return point(a.x - b.x, a.y - b.y);
}
```

```
inline ll operator * (point a, point b)
{
    return a.x * b.y - a.y * b.x;
}

inline bool operator < (point a, point b)
{
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

inline bool check_in(point a, point b, point c, point u) // 判断 u 是否在
三角形 abc 中
{
    if ((b - a) * (c - a) < 0) swap(b, c);
    return (b - a) * (u - a) > 0 && (u - a) * (c - a) > 0 && (c - b) * (u
- b) > 0;
}

inline bool which(int x) // 判断 x 是 A 类点还是 B 类点
{
    return x > n;
}

inline bool cmp(int x, int y)
{
    return (p[x] - p[src]) * (p[y] - p[src]) > 0;
}

inline point del(int x, int y)
{
    return p[x] - p[y];
}

inline void build() // 求所有点的凸包
{
    int i; src = 1;
    for (i = 2; i <= n + m; i++)
        if (p[i] < p[src]) src = i;

    for (i = 1; i <= n + m; i++)
        if (src != i) id[++id[0]] = i;

    sort(id + 1, id + n + m, cmp);
```

```
    stk[top = 1] = src;
    for (i = 1; i < n + m; i++)
    {
        while (top >= 2 && del(stk[top], stk[top - 1]) * del(id[i], stk[top
- 1]) <= 0)
            top--;
        stk[++top] = id[i];
    }

    for (i = 1; i <= top; i++)
        b[++cnt] = stk[i], in[stk[i]] = 1;

    for (i = 1; i <= n + m; i++)
        if (!in[i]) d[++tot] = i;
}

inline void nie()
{
    puts("GG!");
    fclose(stdin);
    fclose(stdout);
    exit(0);
}

inline int init() // 判断凸包上是否只有一种点
{
    int i, res = 1, t = 0, s, ed;
    for (i = 2; i <= cnt; i++)
        if (which(b[i]) != which(b[1])) res = 2;

    if (res == 1) return 1;

    for (i = 1; i < cnt; i++)
        if (which(b[i]) != which(b[i + 1])) t++;

    if (t >= 3) nie();
    if (t == 1)
    {
        for (i = 1; i < cnt; i++)
            if (which(b[i]) != which(b[i + 1])) mid = i;
        return 2;
    }
```

```
    for (i = 1; i < cnt; i++)
        if (which(b[i]) != which(b[i + 1]))
        {
            s = i + 1;
            break;
        }

    ed = cnt;
    for (i = s; i <= cnt; i++)
        if (which(b[i]) == which(b[s])) c[++c[0]] = b[i];
        else
        {
            ed = i - 1;
            mid = ed - s + 1;
            break;
        }

    for (i = ed + 1; i <= cnt; i++)
        c[++c[0]] = b[i];

    for (i = 1; i < s; i++)
        c[++c[0]] = b[i];

    for (i = 1; i <= cnt; i++)
        b[i] = c[i];

    return 2;
}

inline void link(int x, int y)
{
    a[++a[0].x] = line(x, y);
}

inline void solve(int x, int y, int z, vector<int>g) // 三角形 xyz，g 中的点在三角形内部
{
    if (!g.size()) return;

    int num[2] = {0, 0}, len = g.size(), count[2] = {0, 0}, i;
    num[which(x)]++;
    num[which(y)]++;
    num[which(z)]++;
```

```
    if (num[0] == 2) // z 与 x,y 种类不同
    {
        if (which(x)) swap(x, z);
        if (which(y)) swap(y, z);
    }
    else
    {
        if (!which(x)) swap(x, z);
        if (!which(y)) swap(y, z);
    }

    for (i = 0; i < len; i++)
        count[which(g[i])]++;

    if (!count[0] || !count[1]) // g 中的点类别相同
    {
        int rt;
        if (which(x) == which(g[0])) rt = x;
        else rt = z;
        for (i = 0; i < len; i++)
            link(rt, g[i]);
        return;
    }

    int u;
    for (i = 0; i < len; i++)
        if (which(g[i]) == which(z))
        {
            u = g[i];
            break;
        }

    vector<int>g1(0), g2(0), g3(0);

    for (i = 0; i < len; i++) // 分成 3 个三角形
        if (g[i] != u)
        {
            if (check_in(p[x], p[y], p[u], p[g[i]])) g1.push_back(g[i]);
            else if (check_in(p[x], p[z], p[u], p[g[i]])) g2.push_back(g[i]);
            else g3.push_back(g[i]);
        }

    link(u, z);
```

```
    solve(x, y, u, g1);
    solve(x, z, u, g2);
    solve(y, z, u, g3);
}

inline void solveA() // 凸包上只有一种点
{
    int i, x = 0, j, k;
    for (i = 1; i < cnt; i++)
        link(b[i], b[i + 1]);

    if (!tot) return;
    for (i = 1; i <= tot; i++)
        if (which(d[i]) != which(b[1]))
        {
            x = d[i];
            break;
        }

    if (!x)
    {
        for (i = 1; i <= tot; i++)
            link(d[i], b[1]);
        return;
    }

    static vector<int>g[N];

    for (i = 1; i <= cnt; i++)
        g[i].clear();

    for (i = 1; i <= tot; i++)
        if (d[i] != x)
            for (j = 1; j <= cnt; j++)
            {
                int u = b[j], v = b[j + 1];
                if (j == cnt) v = b[1];
                if      (check_in(p[u],      p[v],      p[x],      p[d[i]]))
g[j].push_back(d[i]);
            }

    for (i = 1; i <= cnt; i++)
    {
        int u = b[i], v = b[i + 1];
```

```
        if (i == cnt) v = b[1];
        solve(u, v, x, g[i]);
    }
}

inline void solveB() // 凸包上有两种点
{
    int i, j, rt = b[mid + 1];
    for (i = 1; i < cnt; i++)
        if (i != mid) link(b[i], b[i + 1]);

    for (i = 1; i < mid; i++)
    {
        int u = b[i], v = b[i + 1];
        vector<int>g(0);
        for (j = 1; j <= tot; j++)
            if (check_in(p[u], p[v], p[rt], p[d[j]])) g.push_back(d[j]);
        solve(u, v, rt, g);
    }

    rt = b[1];
    for (i = mid + 1; i < cnt; i++)
    {
        int u = b[i], v = b[i + 1];
        vector<int>g(0);
        for (j = 1; j <= tot; j++)
            if (check_in(p[u], p[v], p[rt], p[d[j]])) g.push_back(d[j]);
        solve(u, v, rt, g);
    }
}

int main()
{
    freopen("disanti.in", "r", stdin);
    freopen("disanti.out", "w", stdout);

    read(n); read(m);
    int i, j;
    for (i = 1; i <= n + m; i++)
        read(p[i].x), read(p[i].y);

    build();
    flag = init();
```

```
    if (flag == 1) solveA();
    else solveB();

    for (i = 1; i <= n + m - 2; i++)
        if (!which(a[i].x))
        print(a[i].x), putchar(' '), print(a[i].y), putchar('\n');

    for (i = 1; i <= n + m - 2; i++)
        if (which(a[i].x))
        print(a[i].x - n), putchar(' '), print(a[i].y - n), putchar('\n');

    fclose(stdin);
    fclose(stdout);
    return 0;
}
```