计算几何与FFT

计算几何

```
#include<bits/stdc++.h>
using namespace std;
#define inf 1e100
#define eps 1e-8
//用于浮点数正负判断，根据题目精度修改
const double pi = acos(-1.0);//圆周率
int sgn(double x){
    if(fabs(x)<eps)return 0;
    if(x<0)return -1;
    return 1;
}//判断浮点数正负
double sqr(double x){return x*x;}//距离等运算涉及大量平方，简便
//使用Point时注意部分函数是返回新Point而非修改本身值
struct Point{
    double x,y;
    /*构造函数*/
    Point(){}
    Point(double xx,double yy){
        x=xx;y=yy;
    }
    /*重载一些点的基础运算符*/
    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;
    }
    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);
    }
    Point operator *(const double &k)const{
        return Point(x*k,y*k);
    }
    Point operator /(const double &k)const{
        return Point(x/k,y/k);
    }
    //叉积
```

```
    double operator ^(const Point &b)const{
        return x*b.y - y*b.x;
    }
    //点积
    double operator *(const Point &b)const{
        return x*b.x + y*b.y;
    }
    /*当前点为p，求角apb大小*/
    double rad(Point a,Point b){
        Point p = *this;
        return fabs(atan2( fabs((a-p)^(b-p)),(a-p)*(b-p) ));
    }
    /*逆时针旋转90度*/
    Point rotleft(){
        return Point(-y,x);
    }
    /*顺时针旋转90度*/
    Point rotright(){
        return Point(y,-x);
    }
    //两点距离
    double dis(Point p){
        return sqrt(sqr(x-p.x)+sqr(y-p.y));
    }
    //原点距离
    double abs(){
        return sqrt(abs2());
    }
    double abs2(){
        return sqr(x)+sqr(y);
    }
    //改变向量长度
    Point trunc(double r){
        double l = abs();
        if(!sgn(l))return *this;
        r /= l;
        return Point(x*r,y*r);
    }
    //单位化
    Point unit() { return *this/abs(); }
    //IO
    void input(){
        scanf("%lf%lf",&x,&y);
    }
    void output(){
        printf("%.7f %.7f\n",x,y);
    }
    //绕着p点逆时针旋转angle
    Point rotate(Point p,double angle){
        Point v = (*this) - p;
        double c = cos(angle), s = sin(angle);
        return Point(p.x + v.x*c - v.y*s,p.y + v.x*s + v.y*c);
    }
};
struct Line{
    //两点确定直线
    Point s,e;
    Line(){}
```

```
    Line(Point ss,Point ee){
        s=ss;e=ee;
    }
    void input(){
        s.input();
        e.input();
    }
    //点在线段上
    bool checkPS(Point p){
        return sgn((p-s)^(e-s)) == 0 && sgn((p-s)*(p-e)) <= 0;
    }
    //直线平行
    bool parallel(Line v){
        return sgn((e-s)^(v.e-v.s)) == 0;
    }
    //点和直线关系
    //1  在左侧
    //2  在右侧
    //3  在直线上
    int relation(Point p){
        int c = sgn((p-s)^(e-s));
        if(c < 0)return 1;
        else if(c > 0)return 2;
        else return 3;
    }
    //线段相交
    //2 规范相交
    //1 非规范相交
    //0 不相交
    int checkSS(Line v){
        int d1 = sgn((e-s)^(v.s-s));
        int d2 = sgn((e-s)^(v.e-s));
        int d3 = sgn((v.e-v.s)^(s-v.s));
        int d4 = sgn((v.e-v.s)^(e-v.s));
        if( (d1^d2)==-2 && (d3^d4)==-2 )return 2;
        return (d1==0 && sgn((v.s-s)*(v.s-e))<=0) ||
               (d2==0 && sgn((v.e-s)*(v.e-e))<=0) ||
               (d3==0 && sgn((s-v.s)*(s-v.e))<=0) ||
               (d4==0 && sgn((e-v.s)*(e-v.e))<=0);
    }
    //直线和线段相交
    //2 规范相交
    //1 非规范相交
    //0 不相交
    int checkLS(Line v){
        int d1 = sgn((e-s)^(v.s-s));
        int d2 = sgn((e-s)^(v.e-s));
        if((d1^d2)==-2) return 2;
        return (d1==0||d2==0);
    }
    //两直线关系
    //0 平行
    //1 重合
    //2 相交
    int checkLL(Line v){
        if((*this).parallel(v))
            return v.relation(s)==3;
        return 2;
```

```
    }
    //直线交点
    Point isLL(Line v){
        double a1 = (v.e-v.s)^(s-v.s);
        double a2 = (v.e-v.s)^(e-v.s);
        return Point((s.x*a2-e.x*a1)/(a2-a1),(s.y*a2-e.y*a1)/(a2-a1));
    }
    //点到直线的距离
    double disPL(Point p){
        return fabs((p-s)^(e-s))/(s.dis(e));
    }
    //点到线段的距离
    double disPS(Point p){
        if(sgn((p-s)*(e-s))<0 || sgn((p-e)*(s-e))<0)
            return min(p.dis(s),p.dis(e));
        return disPL(p);
    }
    //两线段距离
    double disSS(Line v){
        return min(min(disPS(v.s),disPS(v.e)),min(v.disPS(s),v.disPS(e)));
    }
    //点在直线上投影
    Point proj(Point p){
        return s + ( ((e-s)*((e-s)*(p-s)))/((e-s).abs2()) );
    }
    //向垂直有向直线的左侧移动x
    Line push(double x){
        Point tmp=e-s;
        tmp=tmp.rotleft().trunc(x);
        Point ss=s+tmp;
        Point ee=e+tmp;
        return {ss,ee};
    }
};

struct circle{
    Point p;//圆心
    double r;//半径
    circle(){}
    circle(Point pp,double rr){
        p = pp;
        r = rr;
    }
    void input(){
        p.input();
        scanf("%lf",&r);
    }
    bool operator == (circle v){
        return (p==v.p) && sgn(r-v.r)==0;
    }
    //面积
    double area(){
        return pi*r*r;
    }
    //周长
    double cir(){
        return 2*pi*r;
    }
```

```
    //点和圆的关系
    //0 圆外
    //1 圆上
    //2 圆内
    int relation(Point B){
        double dst = B.dis(p);
        if(sgn(dst-r) < 0)return 2;
        else if(sgn(dst-r)==0)return 1;
        return 0;
    }
    //两圆的关系
    //5 相离
    //4 外切
    //3 相交
    //2 内切
    //1 内含
    int checkCC(circle v){
        double d = p.dis(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;
    }
    //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
    int isCC(circle v,Point &p1,Point &p2){
        int rel = checkCC(v);
        if(rel == 1 || rel == 5)return 0;
        double d = p.dis(v.p);
        double l = (d*d+r*r-v.r*v.r)/(2*d);
        double h = sqrt(r*r-l*l);
        Point tmp = p + (v.p-p).trunc(l);
        p1 = tmp + ((v.p-p).rotleft().trunc(h));
        p2 = tmp + ((v.p-p).rotright().trunc(h));
        if(rel == 2 || rel == 4)
            return 1;
        return 2;
    }
    //求直线和圆的交点，返回交点个数
    int isCL(Line v,Point &p1,Point &p2){
        if(sgn(v.disPL(p)-r)>0)return 0;
        Point A = v.proj(p);
        double d = v.disPL(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;
    }
    //点到圆切线
    int tanCP(Point q,Line &u,Line &v){
        int x = relation(q);
        if(x == 2)return 0;
```

```
        if(x == 1){
            u = Line(q,q + (q-p).rotleft());
            v = u;
            return 1;
        }
        double d = p.dis(q);
        double l = r*r/d;
        double h = sqrt(r*r-l*l);
        u = Line(q,p + ((q-p).trunc(l) + (q-p).rotleft().trunc(h)));
        v = Line(q,p + ((q-p).trunc(l) + (q-p).rotright().trunc(h)));
        return 2;
    }
    //两圆相交面积
    double areaCC(circle v){
        int rel = checkCC(v);
        if(rel >= 4)return 0.0;
        if(rel <= 2)return min(area(),v.area());
        double d = p.dis(v.p);
        double hf = (r+v.r+d)/2.0;
        double ss = 2*sqrt(hf*(hf-r)*(hf-v.r)*(hf-d));
        double a1 = acos((r*r+d*d-v.r*v.r)/(2.0*r*d));
        a1 = a1*r*r;
        double a2 = acos((v.r*v.r+d*d-r*r)/(2.0*v.r*d));
        a2 = a2*v.r*v.r;
        return a1+a2-ss;
    }
    //求圆和三角形pab的相交面积
    double areaCT(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(isCL(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;
    }
};
//多边形面积，需保证A逆时针
double area(vector<Point> A) {
    double ans = 0;
    for (int i = 0; i < A.size(); i++) ans += (A[i]^A[(i + 1) % A.size()]);
    return ans / 2;
```

```
}
int contain(vector<Point>A, Point q) { // 2 内部 1 边界 0 外部
    int pd = 0; A.push_back(A[0]);
    for (int i = 1; i < A.size(); i++) {
        Point u = A[i - 1], v = A[i];
        if (Line(u,v).checkPS(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;
}
//凸包
vector<Point> ConvexHull(vector<Point>A, int flag = 1) { // flag=0 不严格 flag=1
严格
    int n = A.size(); vector<Point>ans(n * 2);
    sort(A.begin(), A.end()); int now = -1;
    for (int i = 0; i < A.size(); i++) {
        while (now > 0 && sgn((ans[now] - ans[now - 1])^(A[i] - ans[now - 1])) <
flag) now--;
        ans[++now] = A[i];
    } int pre = now;
    for (int i = n - 2; i >= 0; i--) {
        while (now > pre && sgn((ans[now] - ans[now - 1])^(A[i] - ans[now - 1]))
< flag) now--;
        ans[++now] = A[i];
    } ans.resize(now); return ans;
}
//凸包周长
double convexC(vector<Point>A)
{
    double ans = 0;
    for(int i = 0;i<A.size()-1;i++)
    {
        ans+=A[i].dis(A[i+1]);
    }
    ans += A[A.size()-1].dis(A[0]);
    return ans;
}
//凸包面积
double convexS(vector<Point>A)
{
    int n = A.size();
    double ans = 0;
    for(int i=1;i<n;i++)
    {
        ans += (A[i] - A[0]) ^ (A[i+1] - A[0]);
    }
    return abs(ans) / 2;
}
//凸包直径
double convexDiameter(vector<Point>A) {
    int now = 0, n = A.size(); double ans = 0;
    for (int i = 0; i < A.size(); i++) {
        now = max(now, i);
        while (1) {
            double k1 = A[i].dis(A[now % n]), k2 = A[i].dis(A[(now + 1) % n]);
            ans = max(ans, max(k1, k2)); if (k2 > k1) now++; else break;
        }
```

```
    }
    return ans;
}
//多边形和圆交面积
double areaPC(vector<Point> p,circle c){
    double ans = 0;
    int n=p.size();
    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.areaCT(p[i],p[j]);
        else ans -= c.areaCT(p[i],p[j]);
    }
    return fabs(ans);
}
// 最近点对 ， 先要按照 x 坐标排序
double closepoint(vector<Point>&A, int l, int r) {
    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, A[i].dis(A[j]));
        return ans;
    }
    int mid = l + r >> 1; double ans = min(closepoint(A, l, mid), closepoint(A,
mid + 1, r));
    vector<Point>B; for (int i = l; i <= r; i++) if (abs(A[i].x - A[mid].x) <=
ans) B.push_back(A[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;
}
// 两个点的叉积
double Cross(Point A, Point B) { return A.x*B.y - B.x*A.y; }
// 四个点构成的两个线段的相交情况
bool SideCross(Point a1, Point a2, Point b1, Point b2)
{
    double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1, b2 - a1),
    c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2 - b1);
    return sgn(c1)*sgn(c2)<0 && sgn(c3)*sgn(c4)<0;
}
bool pInConvex(Point a, vector<Point> p)
{

    int i;
    int n = p.size();
    if(n==3)
    {
        if(a.x < min(p[0].x, p[1].x) || a.x > max(p[0].x, p[1].x))
        return 0;
        if(a.y < min(p[0].y, p[1].y) || a.y > max(p[0].y, p[1].y))
            return 0;
    }
    for(i=1;i<n;i++)
    {
        if( Cross(p[i]-p[i-1],a-p[i-1]) < 0.0 ) return 0;
    }
    return 1;
```

```
}
int n,m,cnt;
vector<Point> p,pp;
int solve()
{
    int i,j;
    if(n==1&&m==1) return 1;
    else if(n==1)
    {
        //nw=CH(white, m, cw);
        if(pInConvex(p[0], ConvexHull(pp,1))) return 0;
    }
    else if(m==1)
    {
        //nb=CH(black, n, cb);
        if(pInConvex(pp[0], ConvexHull(p,1))) return 0;
    }
    else
    {
        vector<Point> black = ConvexHull(p,1);
        vector<Point> white = ConvexHull(pp,1);
        //nb=CH(black, n, cb);
        //nw=CH(white, m, cw);
        for(i=1;i<black.size();i++)
        {
            for(j=1;j<white.size();j++)
                if(SideCross( black[i], black[i-1], white[j] ,white[j-1] ))
return 0;
        }
        for(i=0;i<n;i++)
            if(pInConvex(black[i],white)) return 0;
        for(i=0;i<m;i++)
            if(pInConvex(white[i],black)) return 0;
    }
    return 1;
}

int main(){
    scanf("%d %d",&n,&m);
    p.resize(n);
    pp.resize(m);
    for(int i = 0;i<n;i++)
    {
        p[i].input();
    }
    for(int i = 0;i<m;i++)
    {
        pp[i].input();
    }
    int ans=solve();
    if(ans)
        cout<<"YES"<<endl;
    else
        cout<<"NO"<<endl;
//  printf("%.2lf %.2lf",convexC(ConvexHull(p,1)),convexS(ConvexHull(p,1)));
}
```

计算几何（long long)

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define inf 1e100
#define eps 1e-8
//用于浮点数正负判断，根据题目精度修改
const double pi = acos(-1.0);//圆周率
long long sgn(long long x){
    if(fabs(x)<eps)return 0;
    if(x<0)return -1;
    return 1;
}//判断浮点数正负
long long sqr(long long x){return x*x;}//距离等运算涉及大量平方，简便
//使用Point时注意部分函数是返回新Point而非修改本身值
struct Point{
    long long x,y;
    /*构造函数*/
    Point(){}
    Point(long long xx,long long yy){
        x=xx;y=yy;
    }
    /*重载一些点的基础运算符*/
    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;
    }
    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);
    }
    Point operator *(const long long &k)const{
        return Point(x*k,y*k);
    }
    Point operator /(const long long &k)const{
        return Point(x/k,y/k);
    }
    //叉积
    long long operator ^(const Point &b)const{
        return x*b.y - y*b.x;
    }
    //点积
    long long operator *(const Point &b)const{
        return x*b.x + y*b.y;
    }
    /*当前点为p，求角apb大小*/
    long long rad(Point a,Point b){
        Point p = *this;
        return fabs(atan2( fabs((a-p)^(b-p)),(a-p)*(b-p) ));
    }
    /*逆时针旋转90度*/
    Point rotleft(){
```

```
        return Point(-y,x);
    }
    /*顺时针旋转90度*/
    Point rotright(){
        return Point(y,-x);
    }
    //两点距离
    long long dis(Point p){
        return sqrt(sqr(x-p.x)+sqr(y-p.y));
    }
    long long dis2(Point p){
        return sqr(x-p.x)+sqr(y-p.y);
    }
    //原点距离
    long long abs(){
        return sqrt(abs2());
    }
    long long abs2(){
        return sqr(x)+sqr(y);
    }
    //改变向量长度
    Point trunc(long long r){
        long long l = abs();
        if(!sgn(l))return *this;
        r /= l;
        return Point(x*r,y*r);
    }
    //单位化
    Point unit() { return *this/abs(); }
    //IO
    void input(){
        scanf("%lld%lld",&x,&y);
    }
    void output(){
        printf("%lld %lld\n",x,y);
    }
    //绕着p点逆时针旋转angle
    Point rotate(Point p,long long angle){
        Point v = (*this) - p;
        long long c = cos(angle), s = sin(angle);
        return Point(p.x + v.x*c - v.y*s,p.y + v.x*s + v.y*c);
    }
};
struct Line{
    //两点确定直线
    Point s,e;
    Line(){}
    Line(Point ss,Point ee){
        s=ss;e=ee;
    }
    void input(){
        s.input();
        e.input();
    }
    //点在线段上
    bool checkPS(Point p){
        return sgn((p-s)^(e-s)) == 0 && sgn((p-s)*(p-e)) <= 0;
    }
```

```
    //直线平行
    bool parallel(Line v){
        return sgn((e-s)^(v.e-v.s)) == 0;
    }
    //点和直线关系
    //1  在左侧
    //2  在右侧
    //3  在直线上
    long long relation(Point p){
        long long c = sgn((p-s)^(e-s));
        if(c < 0)return 1;
        else if(c > 0)return 2;
        else return 3;
    }
    //线段相交
    //2 规范相交
    //1 非规范相交
    //0 不相交
    long long checkSS(Line v){
        long long d1 = sgn((e-s)^(v.s-s));
        long long d2 = sgn((e-s)^(v.e-s));
        long long d3 = sgn((v.e-v.s)^(s-v.s));
        long long d4 = sgn((v.e-v.s)^(e-v.s));
        if( (d1^d2)==-2 && (d3^d4)==-2 )return 2;
        return (d1==0 && sgn((v.s-s)*(v.s-e))<=0) ||
               (d2==0 && sgn((v.e-s)*(v.e-e))<=0) ||
               (d3==0 && sgn((s-v.s)*(s-v.e))<=0) ||
               (d4==0 && sgn((e-v.s)*(e-v.e))<=0);
    }
    //直线和线段相交
    //2 规范相交
    //1 非规范相交
    //0 不相交
    long long checkLS(Line v){
        long long d1 = sgn((e-s)^(v.s-s));
        long long d2 = sgn((e-s)^(v.e-s));
        if((d1^d2)==-2) return 2;
        return (d1==0||d2==0);
    }
    //两直线关系
    //0 平行
    //1 重合
    //2 相交
    long long checkLL(Line v){
        if((*this).parallel(v))
            return v.relation(s)==3;
        return 2;
    }
    //直线交点
    Point isLL(Line v){
        long long a1 = (v.e-v.s)^(s-v.s);
        long long a2 = (v.e-v.s)^(e-v.s);
        return Point((s.x*a2-e.x*a1)/(a2-a1),(s.y*a2-e.y*a1)/(a2-a1));
    }
    //点到直线的距离
    long long disPL(Point p){
        return fabs((p-s)^(e-s))/(s.dis(e));
    }
```

```
    //点到线段的距离
    long long disPS(Point p){
        if(sgn((p-s)*(e-s))<0 || sgn((p-e)*(s-e))<0)
            return min(p.dis(s),p.dis(e));
        return disPL(p);
    }
    //两线段距离
    long long disSS(Line v){
        return min(min(disPS(v.s),disPS(v.e)),min(v.disPS(s),v.disPS(e)));
    }
    //点在直线上投影
    Point proj(Point p){
        return s + ( ((e-s)*((e-s)*(p-s)))/((e-s).abs2()) );
    }
    //向垂直有向直线的左侧移动x
    Line push(long long x){
        Point tmp=e-s;
        tmp=tmp.rotleft().trunc(x);
        Point ss=s+tmp;
        Point ee=e+tmp;
        return {ss,ee};
    }
};

struct circle{
    Point p;//圆心
    long long r;//半径
    circle(){}
    circle(Point pp,long long rr){
        p = pp;
        r = rr;
    }
    void input(){
        p.input();
        scanf("%lld",&r);
    }
    bool operator == (circle v){
        return (p==v.p) && sgn(r-v.r)==0;
    }
    //面积
    long long area(){
        return pi*r*r;
    }
    //周长
    long long cir(){
        return 2*pi*r;
    }
    //点和圆的关系
    //0 圆外
    //1 圆上
    //2 圆内
    long long relation(Point B){
        long long dst = B.dis(p);
        if(sgn(dst-r) < 0)return 2;
        else if(sgn(dst-r)==0)return 1;
        return 0;
    }
    //两圆的关系
```

```
    //5 相离
    //4 外切
    //3 相交
    //2 内切
    //1 内含
    long long checkCC(circle v){
        long long d = p.dis(v.p);
        if(sgn(d-r-v.r) > 0)return 5;
        if(sgn(d-r-v.r) == 0)return 4;
        long long 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;
    }
    //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
    long long isCC(circle v,Point &p1,Point &p2){
        long long rel = checkCC(v);
        if(rel == 1 || rel == 5)return 0;
        long long d = p.dis(v.p);
        long long l = (d*d+r*r-v.r*v.r)/(2*d);
        long long h = sqrt(r*r-l*l);
        Point tmp = p + (v.p-p).trunc(l);
        p1 = tmp + ((v.p-p).rotleft().trunc(h));
        p2 = tmp + ((v.p-p).rotright().trunc(h));
        if(rel == 2 || rel == 4)
            return 1;
        return 2;
    }
    //求直线和圆的交点，返回交点个数
    long long isCL(Line v,Point &p1,Point &p2){
        if(sgn(v.disPL(p)-r)>0)return 0;
        Point A = v.proj(p);
        long long d = v.disPL(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;
    }
    //点到圆切线
    long long tanCP(Point q,Line &u,Line &v){
        long long x = relation(q);
        if(x == 2)return 0;
        if(x == 1){
            u = Line(q,q + (q-p).rotleft());
            v = u;
            return 1;
        }
        long long d = p.dis(q);
        long long l = r*r/d;
        long long h = sqrt(r*r-l*l);
        u = Line(q,p + ((q-p).trunc(l) + (q-p).rotleft().trunc(h)));
        v = Line(q,p + ((q-p).trunc(l) + (q-p).rotright().trunc(h)));
        return 2;
```

```
    }
    //两圆相交面积
    long long areaCC(circle v){
        long long rel = checkCC(v);
        if(rel >= 4)return 0.0;
        if(rel <= 2)return min(area(),v.area());
        long long d = p.dis(v.p);
        long long hf = (r+v.r+d)/2.0;
        long long ss = 2*sqrt(hf*(hf-r)*(hf-v.r)*(hf-d));
        long long a1 = acos((r*r+d*d-v.r*v.r)/(2.0*r*d));
        a1 = a1*r*r;
        long long a2 = acos((v.r*v.r+d*d-r*r)/(2.0*v.r*d));
        a2 = a2*v.r*v.r;
        return a1+a2-ss;
    }
    //求圆和三角形pab的相交面积
    long long areaCT(Point a,Point b){
        if(sgn((p-a)^(p-b)) == 0)return 0.0;
        Point q[5];
        long long len = 0;
        q[len++] = a;
        Line l(a,b);
        Point p1,p2;
        if(isCL(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]);
        long long res = 0;
        for(int i = 0;i < len-1;i++){
            if(relation(q[i])==0||relation(q[i+1])==0){
                long long 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;
    }
};
//多边形面积，需保证A逆时针
long long area(vector<Point> A) {
    long long ans = 0;
    for (int i = 0; i < A.size(); i++) ans += (A[i]^A[(i + 1) % A.size()]);
    return ans / 2;
}
int contain(vector<Point>A, Point q) { // 2 内部 1 边界 0 外部
    long long pd = 0; A.push_back(A[0]);
    for (int i = 1; i < A.size(); i++) {
        Point u = A[i - 1], v = A[i];
        if (Line(u,v).checkPS(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;
}
```

```
//凸包
vector<Point> ConvexHull(vector<Point>A, int flag = 1) { // flag=0 不严格 flag=1
严格
    long long n = A.size(); vector<Point>ans(n * 2);
    sort(A.begin(), A.end()); int now = -1;
    for (int i = 0; i < A.size(); i++) {
        while (now > 0 && sgn((ans[now] - ans[now - 1])^(A[i] - ans[now - 1])) <
flag) now--;
        ans[++now] = A[i];
    } long long pre = now;
    for (int i = n - 2; i >= 0; i--) {
        while (now > pre && sgn((ans[now] - ans[now - 1])^(A[i] - ans[now - 1]))
< flag) now--;
        ans[++now] = A[i];
    } ans.resize(now); return ans;
}
//凸包周长
long long convexC(vector<Point>A)
{
    long long ans = 0;
    for(int i = 0;i<A.size()-1;i++)
    {
        ans+=A[i].dis(A[i+1]);
    }
    ans += A[A.size()-1].dis(A[0]);
    return ans;
}
//凸包面积
long long convexS(vector<Point>A)
{
    long long n = A.size();
    long long ans = 0;
    for(int i=1;i<n;i++)
    {
        ans += (A[i] - A[0]) ^ (A[i+1] - A[0]);
    }
    return abs(ans) / 2;
}
//凸包直径
long long convexDiameter(vector<Point>A) {
    int now = 0, n = A.size(); long long ans = 0;
    for (int i = 0; i < A.size(); i++) {
        now = max(now, i);
        while (1) {
            long long k1 = A[i].dis2(A[now % n]), k2 = A[i].dis2(A[(now + 1) %
n]);
            ans = max(ans, max(k1, k2)); if (k2 > k1) now++; else break;
        }
    }
    return ans;
}
//多边形和圆交面积
long long areaPC(vector<Point> p,circle c){
    long long ans = 0;
    long long n=p.size();
    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.areaCT(p[i],p[j]);
        else ans -= c.areaCT(p[i],p[j]);
    }
    return fabs(ans);
}
// 最近点对 , 先要按照 x 坐标排序
long long closepoint(vector<Point>&A, int l, int r) {
    if (r - l <= 5) {
        long long ans = 1e15;
        for (int i = l; i <= r; i++) for (int j = i + 1; j <= r; j++) ans =
min(ans, A[i].dis2(A[j]));
        return ans;
    }
    long long mid = l + r >> 1; long long ans = min(closepoint(A, l, mid),
closepoint(A, mid + 1, r));
    vector<Point>B; for (int i = l; i <= r; i++) if (abs(A[i].x - A[mid].x) <=
ans) B.push_back(A[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].dis2(B[j]));
    return ans;
}

long long n,cnt;
vector<Point> p,pp;
int main(){
    scanf("%lld",&n);
    p.resize(n);
    for(int i = 0;i<n;i++)
    {
        p[i].input();
    }
    printf("%lld ",closepoint(p,0,n-1));
    printf("%lld\n",convexDiameter(ConvexHull(p,1)));

}
```

FFT

```
#include<iostream>
#include<algorithm>
#include<cmath>
#include<cstring>
#include<string>
#define MAXN 200010
//数据上限是多少，就开到2.5倍左右，准不会越界
#define RADIX 8
using namespace std;
//本代码为多项式与高精FFT的模板，包括正负0
const double Pi = acos(-1.0);
bool isPositive;
string A, B;
struct complex {
```

```
    double x, y;
    complex(double xx = 0, double yy = 0) {
        x = xx, y = yy;
    }
}a[MAXN],b[MAXN];
complex operator + (complex a, complex b) {
    return complex(a.x + b.x, a.y + b.y);
}
complex operator - (complex a, complex b) {
    return complex(a.x - b.x, a.y - b.y);
}
complex operator * (complex a, complex b) {
    return complex(a.x*b.x-a.y*b.y, a.x*b.y+b.x*a.y);
}
int result[2 * MAXN];
int N, M;
int l, r[MAXN];
int limit;
int counterA, counterB;
inline void init() {
    memset(r, 0, sizeof(r));
    memset(result, 0, sizeof(result));
    limit = 1;
    l = 0;
    counterA = counterB = 0;
    isPositive = true;
    for (int i = 0; i < MAXN; ++i) a[i].x = a[i].y = b[i].x = b[i].y = 0.0;
}
inline void FFT_Iteration(complex* C, double type) {
    for (int i = 0; i < limit; ++i)
        if (i < r[i])swap(C[i], C[r[i]]);
    for (int mi = 1; mi < limit; mi <<= 1) {
        int len = mi << 1;
        complex Wn(cos(Pi / mi), type * sin(Pi / mi));
        for (int j = 0; j < limit; j += len) {
            complex w(1, 0);
            for (int k = 0; k < mi; ++k, w = w * Wn) {
                complex x = C[j + k], y = w * C[j + mi + k];
                C[j + k] = x + y;
                C[j + mi + k] = x - y;
            }
        }
    }
}
inline void getLimitRev() {
    //while (limit <= N + M)limit <<= 1, ++l;//此处的小于还是小于等于要看n和m的意义
    while (limit < N + M)limit <<= 1, ++l;//高精专用，N，M仅指位数
    for (int i = 0; i < limit; ++i) {
        r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
    }
}
inline void input_Poly() {
    while (cin >> N >> M) {
        init();
        for (int i = 0; i <= N; ++i) cin >> a[i].x;
        for (int i = 0; i <= M; ++i) cin >> b[i].x;
        getLimitRev();
        FFT_Iteration(a, 1); FFT_Iteration(b, 1);
```

```
        for (int i = 0; i <= limit; ++i)a[i] = a[i] * b[i];
        FFT_Iteration(a, -1);
        for (int i = 0; i <= N + M; ++i) {
            cout << (int)(a[i].x / limit + 0.5) << " ";
        }
        cout << endl;
    }
}
inline void input_Number() {
    while (cin >> A >> B) {
        if (A == "0" || A == "-0" || B == "0" || B == "-0") {
            cout << 0 << endl; continue;
        }
        init();
        if ((A[0] == '-' && B[0] != '-') || (B[0] == '-' && A[0] != '-'))
            isPositive = false;

        int lastA = (A[0] == '-'), lastB = (B[0] == '-');
        N = A.length(), M = B.length();
        for (int i = N - 1; i >= lastA; --i) a[counterA++].x = A[i] - '0';
        for (int i = M - 1; i >= lastB; --i) b[counterB++].x = B[i] - '0';
        N -= lastA; M -= lastB;//真实的位数
        getLimitRev();
        FFT_Iteration(a, 1); FFT_Iteration(b, 1);
        for (int i = 0; i <= limit; ++i)a[i] = a[i] * b[i];
        FFT_Iteration(a, -1);

        for (int i = 0; i <= limit; i++) {
            result[i] += (int)(a[i].x / limit + 0.5);
            if (result[i] >= RADIX) {
                result[i + 1] += result[i] / RADIX;
                result[i] %= RADIX;
                limit += (i == limit);
            }
        }

        if (!isPositive)cout << "-";
        while (!result[limit] && limit >= 1)limit--;
        limit++;

        while (--limit >= 0)cout << result[limit];
        cout << endl;
    }
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    input_Poly();
    return 0;
}
```

FFT(大整数相乘)

```
#include <stdio.h>
#include <string.h>
```

```
#include <set>
#include <map>
#include <algorithm>
#include<complex>
#include<iostream>
#define pi acos(-1)
using namespace std ;
typedef complex<double> E;

const int maxn = 100000+10;

int R[maxn];
char ch[maxn];
E a[maxn],b[maxn];
int  c[maxn];
int n,m,L;

void fft(E *a,int f){
    for(int i=0;i<n;i++) if(i<R[i]) swap(a[i],a[R[i]]);
    for(int i=1;i<n;i<<=1){
        E wn(cos(pi/i),f*sin(pi/i));
        for(int p=i<<1,j=0;j<n;j+=p){
            E w(1,0);
            for(int k=0;k<i;k++,w*=wn){
                E x=a[j+k],y=w*a[j+k+i];
                a[j+k]=x+y; a[j+k+i]=x-y;
            }
        }
    }
    if(f==-1) for(int i=0;i<n;i++) a[i]/=n;
}

int main(){
    int sgn = 1,len1,len2;
    scanf("%s",ch);
    if(ch[0]=='-')
    {
        len1=strlen(ch);
        for(int i=1;i<len1;i++) a[i-1]=ch[len1-i]-'0';
        sgn*=(-1);
        //len1--;
    }
    else
    {
        len1=strlen(ch);
        for(int i=0;i<len1;i++) a[i]=ch[len1-1-i]-'0';
    }
    //    int len1=strlen(ch);
    //    for(int i=0;i<len1;i++) a[i]=ch[len1-1-i]-'0';
    scanf("%s",ch);
    if(ch[0]=='-')
    {
        len2=strlen(ch);
        for(int i=1;i<len2;i++) b[i-1]=ch[len2-i]-'0';
        sgn*=(-1);
        len2--;
    }
    else
```

```
    {
        len2=strlen(ch);
        for(int i=0;i<len2;i++) b[i]=ch[len2-1-i]-'0';
    }
    //    int len2=strlen(ch);
    //    for(int i=0;i<len2;i++) b[i]=ch[len2-1-i]-'0';
    n=max(len1,len2); n--; L=0; m=2*n;
    for(n=1;n<=m;n<<=1) L++;
    for(int i=0;i<n;i++) R[i]=(R[i>>1]>>1)|((i&1)<<(L-1));
    fft(a,1),fft(b,1);
    for(int i=0;i<=n;i++) a[i]=a[i]*b[i];
    fft(a,-1);
    for(int i=0;i<=m;i++) c[i]=(int)(a[i].real()+0.1);
    for(int i=0;i<=m;i++){
        if(c[i]>=10){
            c[i+1]+=c[i]/10,c[i]%=10;
            if(i==m) m++;
        }
    }
    bool flag=false;
    if(sgn==-1) putchar('-');
    for(int i=m;i>=0;i--){
        if(flag||c[i]){
            printf("%d",c[i]);
            flag=true;
        }
    }
}
```