srgioreinier Notebook

Contents

1 Geometry 1
1.1 All Kinds of Distance 1
1.2 Intersection of Segments 3
1.3 Closest Point Approach 4
1.4 Number of Disjoint Triangles 6
1.5 Min/Max triangle(Rotating calipers) 7
1.6 Min triangle(Rotating calipers) / Max Triangle(Max Scalar Product) 9
1.7 Greatest Quadrilater 10
1.8 Smallest Quadrilater 12
1.9 Linear Translation 13
1.10 Maximize/Minimize Scalar Product 14
1.11 Maximize Scalar Product 16
1.12 Maximize Function (x*y = k) (Scalar Product) 17
1.13 Convex Hull/Point Inside Hull 18
1.14 Vectorial Space/Point Inside Hull 20
1.15 Polygon Tangent 21
1.16 Set of edges/Line Sweep 23
1.17 Shamos Hoey - Set of edges/Line Sweep 24
1.18 Shamos Hoey 26
1.19 Graham Scan (DP) 27
1.20 Union of Convex Hulls (O(n * log(n))) 28
1.21 Upward and Downward edges 30
1.22 Minimum Enclosing Circle 31

2 DSU 32
2.1 Bosses 32
2.2 DSU in Range 32
2.3 Nearest available 33
2.4 Nearest available right (circular) 33

3 Segment Tree 34
3.1 Element at least x (binary search) 34
3.2 Element at least x and j > l(binary search) 34
3.3 K-th one (binary search) 35
3.4 Intersecting segments 35
3.5 Inversion count 36
3.6 Recover answer from inversion count 36

4 Dynamic Segment Tree 37
4.1 BGSHOOT (Lazy) 37
4.2 Dynamic Segment Tree 37
4.3 Lazy Dynamic Segment Tree 38
4.4 Orderset 38
4.5 Pathwalks 39
4.6 Range Sum Query 40

5 Persistent Segment Tree 40
5.1 Persistent Segment Tree 40

6 DP Optimization 41
6.1 Divide And Conquer 41
6.2 CHT Example 41
6.3 Garçom (no opt) 42
6.4 Garçom (Divide and Conquer) 42
6.5 Garçom (Knuth) 42
6.6 Internet Trouble(Knuth) 43
6.7 Knuth Optimization 43

7 Stair Nim 44
7.1 Move coins (stair nim variation) 44
7.2 Classic Stair Nim 44

1 Geometry

1.1 All Kinds of Distance

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

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long double type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(point p) { return x == p.x and y == p.y; }
    bool operator !=(point p) { return x != p.x  or y != p.y; }
    bool operator <(const point p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
            );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y,p.x); }
point RotateCW90(point p)    { return point(p.y,-p.x); }

type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

point ProjectPointLine(point c, point a, point b) {
    ld r = dot(b - a,b - a);
    if (fabs(r) < EPS) return a;
    return a + (b - a)*dot(c - a, b - a)/dot(b - a, b - a);
}

point ProjectPointRay(point c, point a, point b) {
    ld r = dot(b - a, b - a);
    if (fabs(r) < EPS) return a;
    r = dot(c - a, b - a) / r;
    if (le(r, 0)) return a;
    return a + (b - a)*r;
}

point ProjectPointSegment(point c, point a, point b) {
    ld r = dot(b - a, b - a);
    if (fabs(r) < EPS) return a;
    r = dot(c - a, b - a)/r;
    if (le(r, 0)) return a;
    if (ge(r, 1)) return b;
    return a + (b - a)*r;
}

ld DistancePointLine(point c, point a, point b) {
    return c.dist2(ProjectPointLine(c, a, b));
}

ld DistancePointRay(point c, point a, point b) {
    return c.dist2(ProjectPointRay(c, a, b));
}

ld DistancePointSegment(point c, point a, point b) {
    return c.dist2(ProjectPointSegment(c, a, b));
}

ld DistancePointPlane(ld x, ld y, ld z,
                        ld a, ld b, ld c, ld d)
{
    return fabs(a*x + b*y + c*z - d)/sqrt(a*a + b*b + c*c);
}

bool LinesParallel(point a, point b, point c, point d) {
    return fabs(cross(b - a, d - c)) < EPS;
}

bool LinesCollinear(point a, point b, point c, point d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

point lines_intersect(point p, point q, point a, point b) {
    point r = q - p, s = b - a, c(p%q, a%b);
    if (eq(r%s,0)) return point(LINF, LINF);
    return point(point(r.x, s.x) % c, point(r.y, s.y) % c) / (r%s);
}

point ComputeLineIntersection(point a, point b, point c, point d) {
    b = b - a; d = c - d; c = c - a;
    assert(dot(b, b) > EPS && dot(d, d) > EPS);
    return a + b*cross(c, d)/cross(b, d);
}

bool LineLineIntersect(point a, point b, point c, point d) {
    if(!LinesParallel(a, b, c, d)) return true;
    if(LinesCollinear(a, b, c, d)) return true;
    return false;
}

bool RayRayIntersect(point a, point b, point c, point d){
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    if (LinesCollinear(a, b, c, d)) {
        if(ge(dot(b - a, d - c), 0)) return true;
        if(ge(dot(a - c, d - c), 0)) return true;
        return false;
    }
    if(!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(ge(dot(inters - c, d - c), 0) && ge(dot(inters - a, b - a), 0)) return true;
    return false;
}

bool SegmentSegmentIntersect(point a, point b, point c, point d) {
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    int d1, d2, d3, d4;
    d1 = direction(a, b, c);
    d2 = direction(a, b, d);
    d3 = direction(c, d, a);
    d4 = direction(c, d, b);
    if (d1*d2 < 0 and d3*d4 < 0) return 1;
    return a.on_seg(c, d) or b.on_seg(c, d) or
            c.on_seg(a, b) or b.on_seg(c, d);
}

bool SegmentLineIntersect(point a, point b, point c, point d){
    if(!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(inters.on_seg(a, b)) return true;
    return false;
}

bool SegmentRayIntersect(point a, point b, point c, point d){
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    if (LinesCollinear(a, b, c, d)) {
        if(c.on_seg(a, b)) return true;
        if(ge(dot(d - c, a - c), 0)) return true;
        return false;
    }
    if(!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(!inters.on_seg(a, b)) return false;
    if(ge(dot(inters - c, d - c), 0)) return true;
    return false;
}

bool RayLineIntersect(point a, point b, point c, point d){
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    if (!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(!LineLineIntersect(a, b, c, d)) return false;
    if(ge(dot(inters - a, b - a), 0)) return true;
    return false;
}

ld DistanceSegmentLine(point a, point b, point c, point d){
    if(SegmentLineIntersect(a, b, c, d)) return 0;
    return min(DistancePointLine(a, c, d), DistancePointLine(b, c, d));
}

ld DistanceSegmentRay(point a, point b, point c, point d){
    if(SegmentRayIntersect(a, b, c, d)) return 0;
    ld min1 = DistancePointSegment(c, a, b);
    ld min2 = min(DistancePointRay(a, c, d), DistancePointRay(b, c, d));
    return min(min1, min2);
}

ld DistanceSegmentSegment(point a, point b, point c, point d){
    if(SegmentSegmentIntersect(a, b, c, d)) return 0;
    ld min1 = min(DistancePointSegment(c, a, b), DistancePointSegment(d, a, b));
    ld min2 = min(DistancePointSegment(a, c, d), DistancePointSegment(b, c, d));
    return min(min1, min2);
}

ld DistanceRayLine(point a, point b, point c, point d){
    if(RayLineIntersect(a, b, c, d)) return 0;
    ld min1 = DistancePointLine(a, c, d);
    return min1;
}

ld DistanceRayRay(point a, point b, point c, point d){
    if(RayRayIntersect(a, b, c, d)) return 0;
    ld min1 = min(DistancePointRay(c, a, b), DistancePointRay(a, c, d));
    return min1;
}

ld DistanceLineLine(point a, point b, point c, point d){
    if(LineLineIntersect(a, b, c, d)) return 0;
    return DistancePointLine(a, c, d);
}

point pts[4];

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    for(int i = 0; i < 4; i++) cin >> pts[i].x >> pts[i].y;
    cout << setprecision(18) << fixed;
    // The distance from the point A to the point C.
        cout << pts[0].dist(pts[2]) << "\n";
    // The distance from the point A to the segment CD.
        cout << sqrt(DistancePointSegment(pts[0], pts[2], pts[3])) << "\n";
    // The distance from the point A to the half-infinite ray CD.
        cout << sqrt(DistancePointRay(pts[0], pts[2], pts[3])) << "\n";
    // The distance from the point A to the line CD.
        cout << sqrt(DistancePointLine(pts[0], pts[2], pts[3])) << "\n";
    // The distance from the segment AB to the point C.
        cout << sqrt(DistancePointSegment(pts[2], pts[0], pts[1])) << "\n";
    // The distance from the segment AB to the segment CD.
        cout << sqrt(DistanceSegmentSegment(pts[0], pts[1], pts[2], pts[3])) << "\n";
    // The distance from the segment AB to the half-infinite ray CD.
        cout << sqrt(DistanceSegmentRay(pts[0], pts[1], pts[2], pts[3])) << "\n";
    // The distance from the segment AB to the line CD.
        cout << sqrt(DistanceSegmentLine(pts[0], pts[1], pts[2], pts[3])) << "\n";
    // The distance from the half-infinite ray AB to the point C.
        cout << sqrt(DistancePointRay(pts[2], pts[0], pts[1])) << "\n";
    // The distance from the half-infinite ray AB to the segment CD.
        cout << sqrt(DistanceSegmentRay(pts[2], pts[3], pts[0], pts[1])) << "\n";
    // The distance from the half-infinite ray AB to the half-infinite ray CD.
        cout << sqrt(DistanceRayRay(pts[0], pts[1], pts[2], pts[3])) << "\n";
    // The distance from the half-infinite ray AB to the line CD.
        cout << sqrt(DistanceRayLine(pts[0], pts[1], pts[2], pts[3])) << "\n";
    // The distance from the line AB to the point C.
        cout << sqrt(DistancePointLine(pts[2], pts[0], pts[1])) << "\n";
    // The distance from the line AB to the segment CD.
        cout << sqrt(DistanceSegmentLine(pts[2], pts[3], pts[0], pts[1])) << "\n";
    // The distance from the line AB to the half-infinite ray CD.
        cout << sqrt(DistanceRayLine(pts[2], pts[3], pts[0], pts[1])) << "\n";
    // The distance from the line AB to the line CD.
        cout << sqrt(DistanceLineLine(pts[0], pts[1], pts[2], pts[3])) << "\n";
    return 0;
}
```

1.2 Intersection of Segments

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long double type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(point p) { return x == p.x and y == p.y; }
    bool operator !=(point p) { return x != p.x  or y != p.y; }
    bool operator <(const point p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y,p.x); }
point RotateCW90(point p)    { return point(p.y,-p.x); }

type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

point ProjectPointLine(point c, point a, point b) {
    ld r = dot(b - a,b - a);
    if (fabs(r) < EPS) return a;
    return a + (b - a)*dot(c - a, b - a)/dot(b - a, b - a);
}

point ProjectPointSegment(point c, point a, point b) {
    ld r = dot(b - a,b - a);
    if (fabs(r) < EPS) return a;
    r = dot(c - a, b - a)/r;
    if (!ge(r, 0)) return a;
    if (!le(r, 1)) return b;
    return a + (b - a)*r;
}

point ProjectPointRay(point c, point a, point b) {
    ld r = dot(b - a,b - a);
    if (fabs(r) < EPS) return a;
    r = dot(c - a, b - a) / r;
    if (!ge(r, 0)) return a;
    return a + (b - a)*r;
}

ld DistancePointSegment(point c, point a, point b) {
    return c.dist2(ProjectPointSegment(c, a, b));
}

ld DistancePointLine(point c, point a, point b) {
    return c.dist2(ProjectPointLine(c, a, b));
}

ld DistancePointRay(point c, point a, point b) {
    return c.dist2(ProjectPointRay(c, a, b));
}

ld DistancePointPlane(ld x, ld y, ld z,
                        ld a, ld b, ld c, ld d)
{
    return fabs(a*x + b*y + c*z - d)/sqrt(a*a + b*b + c*c);
}

bool LinesParallel(point a, point b, point c, point d) {
    return fabs(cross(b - a, c - d)) < EPS;
}

bool LinesCollinear(point a, point b, point c, point d) {
    return LinesParallel(a, b, c, d)
        && fabs(cross(a - b, a - c)) < EPS
        && fabs(cross(c - d, c - a)) < EPS;
}

point lines_intersect(point p, point q, point a, point b) {
    point r = q - p, s = b - a, c(p%q, a%b);
    if (eq(r%s,0)) return point(LINF, LINF);
    return point(point(r.x, s.x) % c, point(r.y, s.y) % c) / (r%s);
}

point ComputeLineIntersection(point a, point b, point c, point d) {
    b = b - a; d = c - d; c = c - a;
    assert(dot(b, b) > EPS && dot(d, d) > EPS);
    return a + b*cross(c, d)/cross(b, d);
}

bool LinesIntersect(point a, point b, point c, point d) {
    if(!LinesParallel(a, b, c, d)) return true;
    if(LinesCollinear(a, b, c, d)) return true;
    return false;
}

bool SegmentsIntersect(point p, point q, point a, point b) {
    int d1, d2, d3, d4;
    d1 = direction(p, q, a);
    d2 = direction(p, q, b);
    d3 = direction(a, b, p);
    d4 = direction(a, b, q);
    if (d1*d2 < 0 and d3*d4 < 0) return 1;
    return p.on_seg(a, b) or q.on_seg(a, b) or
            a.on_seg(p, q) or b.on_seg(p, q);
}

vector<point> CalcSegInter(point a, point b, point c, point d){
    vector<point> ans;
    if(!SegmentsIntersect(a, b, c, d)) return ans;
    if(c.on_seg(a, b)) ans.pb(c);
    if(d.on_seg(a, b)) ans.pb(d);
    if(a.on_seg(c, d)) ans.pb(a);
    if(b.on_seg(c, d)) ans.pb(b);
    if(!LinesParallel(a, b, c, d)){
        point inter = lines_intersect(a, b, c, d);
      if(inter.x + EPS < INF) ans.pb(inter);
    }
    return ans;
}

bool cmp(point a, point b){
  if(eq(a.x,b.x)) return le(a.y,b.y);
  return le(a.x,b.x);
}

point pts[4];

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout << setprecision(18) << fixed;
    for(int i = 0; i < 4; i++) cin >> pts[i].x >> pts[i].y;
    vector<point> ans = CalcSegInter(pts[0], pts[1], pts[2], pts[3]);
    if(!ans.size()) cout << "Empty\n";
    else{
        sort(ans.begin(), ans.end(), cmp);
        for(int i = 0; i < ans.size(); i++){
        point p = ans[i];
        if(i)
          if(eq(ans[i].x, ans[i-1].x) and eq(ans[i].y, ans[i-1].y)) continue;
            cout << p.x << " " << p.y << "\n";
        }
    }
    return 0;
}
```

1.3 Closest Point Approach

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long double type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(point p) { return x == p.x and y == p.y; }
    bool operator !=(point p) { return x != p.x  or y != p.y; }
    bool operator <(const point p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y,p.x); }
point RotateCW90(point p)    { return point(p.y,-p.x); }

type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

point ProjectPointLine(point c, point a, point b) {
    ld r = dot(b - a,b - a);
    if (fabs(r) < EPS) return a;
    return a + (b - a)*dot(c - a, b - a)/dot(b - a, b - a);
}

point ProjectPointRay(point c, point a, point b) {
    ld r = dot(b - a, b - a);
    if (fabs(r) < EPS) return a;
    r = dot(c - a, b - a) / r;
    if (le(r, 0)) return a;
    return a + (b - a)*r;
}

point ProjectPointSegment(point c, point a, point b) {
    ld r = dot(b - a, b - a);
    if (fabs(r) < EPS) return a;
    r = dot(c - a, b - a)/r;
    if (le(r, 0)) return a;
    if (ge(r, 1)) return b;
    return a + (b - a)*r;
}

ld DistancePointLine(point c, point a, point b) {
    return c.dist2(ProjectPointLine(c, a, b));
}
```

```
ld DistancePointRay(point c, point a, point b) {
    return c.dist2(ProjectPointRay(c, a, b));
}

ld DistancePointSegment(point c, point a, point b) {
    return c.dist2(ProjectPointSegment(c, a, b));
}

//not tested
ld DistancePointPlane(ld x, ld y, ld z,
                      ld a, ld b, ld c, ld d)
{
    return fabs(a*x + b*y + c*z - d)/sqrt(a*a + b*b + c*c);
}

bool LinesParallel(point a, point b, point c, point d) {
    return fabs(cross(b - a, d - c)) < EPS;
}

bool LinesCollinear(point a, point b, point c, point d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

point lines_intersect(point p, point q, point a, point b) {
    point r = q - p, s = b - a, c(p%q, a%b);
    if (eq(r%s,0)) return point(LINF, LINF);
    return point(point(r.x, s.x) % c, point(r.y, s.y) % c) / (r%s);
}

point ComputeLineIntersection(point a, point b, point c, point d) {
    b = b - a; d = c - d; c = c - a;
    assert(dot(b, b) > EPS && dot(d, d) > EPS);
    return a + b*cross(c, d)/cross(b, d);
}

bool LineLineIntersect(point a, point b, point c, point d) {
    if(!LinesParallel(a, b, c, d)) return true;
    if(LinesCollinear(a, b, c, d)) return true;
    return false;
}

bool RayRayIntersect(point a, point b, point c, point d){
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    if (LinesCollinear(a, b, c, d)) {
        if(ge(dot(b - a, d - c), 0)) return true;
        if(ge(dot(a - c, d - c), 0)) return true;
        return false;
    }
    if(!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(ge(dot(inters - c, d - c), 0) && ge(dot(inters - a, b - a), 0)) return true;
    return false;
}

bool SegmentSegmentIntersect(point a, point b, point c, point d) {
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    int d1, d2, d3, d4;
    d1 = direction(a, b, c);
    d2 = direction(a, b, d);
    d3 = direction(c, d, a);
    d4 = direction(c, d, b);
    if (d1*d2 < 0 and d3*d4 < 0) return 1;
    return a.on_seg(c, d) or b.on_seg(c, d) or
            c.on_seg(a, b) or b.on_seg(c, d);
}

bool SegmentLineIntersect(point a, point b, point c, point d){
    if(!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(inters.on_seg(a, b)) return true;
    return false;
}

bool SegmentRayIntersect(point a, point b, point c, point d){
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    if (LinesCollinear(a, b, c, d)) {
        if(c.on_seg(a, b)) return true;
        if(ge(dot(d - c, a - c), 0)) return true;
        return false;
    }
    if(!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(!inters.on_seg(a, b)) return false;
    if(ge(dot(inters - c, d - c), 0)) return true;
    return false;
}

bool RayLineIntersect(point a, point b, point c, point d){
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    if (!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(!LineLineIntersect(a, b, c, d)) return false;
    if(ge(dot(inters - a, b - a), 0)) return true;
    return false;
}

ld DistanceSegmentLine(point a, point b, point c, point d){
    if(SegmentLineIntersect(a, b, c, d)) return 0;
    return min(DistancePointLine(a, c, d), DistancePointLine(b, c, d));
}

ld DistanceSegmentRay(point a, point b, point c, point d){
    if(SegmentRayIntersect(a, b, c, d)) return 0;
    ld min1 = DistancePointSegment(c, a, b);
    ld min2 = min(DistancePointRay(a, c, d), DistancePointRay(b, c, d));
    return min(min1, min2);
}

ld DistanceSegmentSegment(point a, point b, point c, point d){
    if(SegmentSegmentIntersect(a, b, c, d)) return 0;
    ld min1 = min(DistancePointSegment(c, a, b), DistancePointSegment(d, a, b));
    ld min2 = min(DistancePointSegment(a, c, d), DistancePointSegment(b, c, d));
    return min(min1, min2);
}

ld DistanceRayLine(point a, point b, point c, point d){
    if(RayLineIntersect(a, b, c, d)) return 0;
    ld min1 = DistancePointLine(a, c, d);
    return min1;
}

ld DistanceRayRay(point a, point b, point c, point d){
    if(RayRayIntersect(a, b, c, d)) return 0;
    ld min1 = min(DistancePointRay(c, a, b), DistancePointRay(a, c, d));
    return min1;
}

ld DistanceLineLine(point a, point b, point c, point d){
    if(LineLineIntersect(a, b, c, d)) return 0;
    return DistancePointLine(a, c, d);
}

//Closest Point Approach
ld CPA(point p, point u, point q, point v){
    point w = p - q;
    if(fabs(dot(u - v, u - v)) < EPS) return LINF;
    return -dot(w, u - v)/dot(u - v, u - v);
}

pair <bool, ld> time_intersects(point p, point a, point b, point v, point u){
    ld num = (p.x - a.x)*(b.y - a.y) - (p.y - a.y)*(b.x - a.x);
    ld den = (v.x - u.x)*(b.y - a.y) - (v.y - u.y)*(b.x - a.x);
    // db(num _ den);
    if(eq(abs(num), 0.0) and eq(abs(u%v), 0.0)){
        // db(num _ u%v);
        if(!ge((b - a)*(u), 0)) swap(b, a);
        if(!le((p - a)*(b - a), 0)){
            if(le(u * v,  0) or !le(v.abs2(), u.abs2())){
                return{true, p.dist(b)/(u - v).abs()};
            }
            else{
                return {false, LINF};
            }
        }
        else{
            if(ge(u * v, 0) and !le(u.abs2(), v.abs2())){
                return{true, p.dist(a)/(u - v).abs()};
            }
            else{
                return {false, LINF};
            }
        }
    }
    if(eq(abs(den), 0)) return {false, LINF};
    ld ans = -num/den;
    if(ge(ans, 0)) return {true, ans};
    return {false, LINF};
}

point p[2][2], v[2];
ld ans = LINF;
bool ok = false;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    for(int i = 0; i < 2; i++) for(int j = 0; j < 2; j++) cin >> p[i][j].x >> p[i][j].y;
    for(int i = 0; i < 2; i++) cin >> v[i].x >> v[i].y;
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 2; j++){
            pair <bool, ld> t = time_intersects(p[i][j], p[i^1][0], p[i^1][1], v[i], v[i^1]);
            // db(t.st _ t.nd);
            if(!t.st) continue;
            if((p[i][j] + v[i]*t.nd).on_seg((p[i^1][0] + v[i^1]*t.nd), (p[i^1][1] + v[i^1]*t.nd))) ans = min(
                ans, t.nd), ok = true;
        }
    }
    if(!ok) cout << "-1\n";
    else cout << setprecision(18) << fixed << ans << "\n";
    return 0;
}
```

1.4 Number of Disjoint Triangles

```
/*
    two pointers for max:
    idea: fix one point (i), than make two pointers (l, r) walk on the polygon:
            fixing l, walk with r until area decreases, than walk with l
            be careful with boundaries, suggestion to duplicate the polygon
            l cant become i, neither can r
*/
/*
    rotating calipers for min:
    idea:   sort points by x, than y
            sort all posible edges radially with respect to the edge perpendicular!
            start processing they on a sweep, every time you encounter one edge its time to process
            process means that the points from the edge will change places on the vector
            for min triangle: min triangle will be made with the current edge and adjacent points
            for max triangle: max triangle will be made with current edge and farthest points (0, n - 1)
*/
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long long type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0){}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }
    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return (x >= 0) - (x <= 0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)    { return point(-p.y, p.x); }
point RotateCW90(point p)     { return point(p.y, -p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

int angleLess(const point& a1, const point& b1, const point& a2, const point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot(   a1, b1), abs(cross(   a1, b1)));
    point p2(dot(   a2, b2), abs(cross(   a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

point origin;

int above(point p){
    if(p.y == origin.y) return p.x > origin.x;
    return p.y > origin.y;
}

bool cmp(pair<point, point> a, pair<point, point> b){
    point p = RotateCW90(a.nd - a.st);
    point q = RotateCW90(b.nd - b.st);
    int tmp = above(q) - above(p);
    if(tmp) return tmp > 0;
    return p.dir(origin,q) > 0;
}

int n;
map<point, int> id;
int main(){
    //freopen("in.txt", "r", stdin);
    //freopen("out2.txt", "w", stdout);
    scanf("%d", &n);
    vector<point> pts(n);
    for(int i = 0; i < n; i++){
        scanf("%lld%lld", &pts[i].x, &pts[i].y);
    }
    //sort points
    sort(pts.begin(), pts.end());
    for(int i = 0; i < n; i++){
        point p = pts[i];
        id[p] = i;
    }

    //create edges and sort perpendicular radially
    vector<pair<point, point>> edges;
    for(int i = 0; i < n; i++){
        for(int j = i + 1; j < n; j++){
            edges.pb({pts[i], pts[j]});
        }
    }
    sort(edges.begin(), edges.end(), cmp);

    ll ans = 0;

    //number of triangles
    //points will be adjacent if theres not 3 collinear points
    //a.nd - a.st => rotateCW
    //a.st - a.nd => rotateCCW
    for(auto e : edges){
        int l = id[e.st], r = id[e.nd];
        if(l > r) swap(l, r);
        ll a = ((l - 1) * l) / 2;
        ll b = ((n - 1 - r) * (n - 2 - r)) / 2;
        ans += 1ll * a * b;
        swap(pts[l], pts[r]);
        swap(id[e.nd], id[e.st]);
    }
    printf("%lld\n", ans);
    return 0;
}
```

1.5 Min/Max triangle(Rotating calipers)

```
//read triangles.cpp
//this code does not uses convex hull, bit faster but can fail for some tests
//todo: correct with input from http://serjudging.vanb.org/?p=561

#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef int type;
```

```
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0){}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y, p.x); }
point RotateCW90(point p)    { return point(p.y, p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

int angleLess(const point& a1, const point& b1, const point& a2, const point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot(   a1, b1), abs(cross(   a1, b1)));
    point p2(dot(   a2, b2), abs(cross(   a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

point origin, ini;

int above(point p){
    if(p.y == origin.y) return p.x > origin.x;
    return p.y > origin.y;
}

bool cmp(pair<point, point> a, pair<point, point> b){
    point p = RotateCW90(a.nd - a.st);
    point q = RotateCW90(b.nd - b.st);
    int tmp = above(q) - above(p);
    if(tmp) return tmp > 0;
    return p.dir(origin,q) > 0;
}

int n;
map<point, int> id;

int main(){
    freopen("in.txt", "r", stdin);
    freopen("out2.txt", "w", stdout);
    while(true){
        scanf("%d", &n);
        if(!n) return 0;
        vector<point> pts(n);
        for(int i = 0; i < n; i++){
            scanf("%d%d", &pts[i].x, &pts[i].y);
        }

        //area
        int mn_area = INF, mx_area = 0;
        vector<pair<point, point>> edges;

        sort(pts.begin(), pts.end());
        for(int i = 0; i < pts.size(); i++){
            point p = pts[i];
            id[p] = i;
        }

        //create edges and sort perpendicular radially
        for(int i = 0; i < n; i++){
            for(int j = i + 1; j < n; j++){
                edges.pb({pts[i], pts[j]});
            }
        }
        sort(edges.begin(), edges.end(), cmp);

        //smaller triangle
        for(auto e : edges){
            int tmp = INF;
            int l = id[e.st], r = id[e.nd];
            //bigger area
            if((n - 1 != r) and (n - 1 != l)){
                tmp = fabs(area2(pts[l], pts[r], pts[n - 1]));
                mx_area = max(tmp, mx_area);
            }
            if(0 != r and 0 != l){
                tmp = fabs(area2(pts[l], pts[r], pts[0]));
                mx_area = max(tmp, mx_area);
            }
            //smaller area
            if(l > 0 and l - 1 != r){
                tmp = fabs(area2(pts[l], pts[r], pts[l - 1]));
                mn_area = min(tmp, mn_area);
            }
            if(r > 0 and r - 1 != l){
                tmp = fabs(area2(pts[l], pts[r], pts[r - 1]));
                mn_area = min(tmp, mn_area);
            }
            if(l < (int)pts.size() - 1 and l + 1 != r){
                tmp = fabs(area2(pts[l], pts[r], pts[l + 1]));
                mn_area = min(tmp, mn_area);
            }
            if(r < (int)pts.size() - 1 and r + 1 != l){
                tmp = fabs(area2(pts[l], pts[r], pts[r + 1]));
                mn_area = min(tmp, mn_area);
            }
            swap(pts[l], pts[r]);
            swap(id[e.nd], id[e.st]);
        }
        printf("%d%s", mn_area/2, (mn_area % 2) ? ".5 " : ".0 ");
        printf("%d%s", mx_area/2, (mx_area % 2) ? ".5\n" : ".0\n");
    }
    return 0;
}
```

1.6 Min triangle(Rotating calipers) / Max Triangle(Max Scalar Product)

```
/*
    two pointers for max:
    idea: fix one point (i), than make two pointers (l, r) walk on the polygon:
            fixing l, walk with r until area decreases, than walk with l
            be careful with boundaries, suggestion to duplicate the polygon
            l cant become i, neither can r
*/
/*
    rotating calipers for min:
    idea:   sort points by x, than y
            sort all posible edges radially with respect to the edge perpendicular!
            start processing they on a sweep, every time you encounter one edge its time to process
            process means that the points from the edge will change places on the vector
            for min triangle: min triangle will be made with the current edge and adjacent points
            for max triangle: max triangle will be made with current edge and farthest points (0, n - 1)
*/
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef int type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0){}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right

    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y, p.x); }
point RotateCW90(point p)    { return point(p.y, -p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

int angleLess(const point& a1, const point& b1, const point& a2, const point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot(   a1, b1), abs(cross(   a1, b1)));
    point p2(dot(   a2, b2), abs(cross(   a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

//Monotone chain O(nlog(n))
#define REMOVE_REDUNDANT
#ifdef REMOVE_REDUNDANT
bool between(const point &a, const point &b, const point &c) {
    return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void ConvexHull(vector<point> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<point> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

    #ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
        if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
        dn[0] = dn.back();
        dn.pop_back();
    }
    pts = dn;
    #endif
}

point origin, ini;

int above(point p){
    if(p.y == origin.y) return p.x > origin.x;
    return p.y > origin.y;
}

bool cmp(pair<point, point> a, pair<point, point> b){
    point p = RotateCW90(a.nd - a.st);
    point q = RotateCW90(b.nd - b.st);
    int tmp = above(q) - above(p);
    if(tmp) return tmp > 0;
    return p.dir(origin,q) > 0;
}

int n;
map<point, int> id;

int main(){
    freopen("in.txt", "r", stdin);
    freopen("out2.txt", "w", stdout);
    while(true){
        scanf("%d", &n);
        if(!n) return 0;
        vector<point> pts(n), old(n);
        int oldn = n;
        for(int i = 0; i < n; i++){
            scanf("%d%d", &pts[i].x, &pts[i].y);
            old[i] = pts[i];
        }
        ConvexHull(pts);
        n = pts.size();
        pts.resize(2*n);
        for(int i = 0; i < n; i++){
            pts[i + n] = pts[i];
        }

        //greater area
        int mx_area = 0;
        for(int i = 0; i < n; i++){
            for(int l = i + 1, r = i + 2; (l < i + n) and (r < i + n); l++){
                int tmp = fabs(area2(pts[i], pts[l], pts[r]));
                while((r < i + n - 1) and fabs(area2(pts[i], pts[l], pts[r])) <= fabs(area2(pts[i], pts[l],
                      pts[r + 1]))){
                    r++;
                    tmp = fabs(area2(pts[i], pts[l], pts[r]));
                }
                mx_area = max(mx_area, tmp);
            }
        }

        //smaller area
        int mn_area = INF;
        vector<pair<point, point>> edges;

        sort(old.begin(), old.end());
        for(int i = 0; i < old.size(); i++){
            point p = old[i];
            id[p] = i;
        }

        //create edges and sort perpendicular radially
        for(int i = 0; i < oldn; i++){
            for(int j = i + 1; j < oldn; j++){
                edges.pb({old[i], old[j]});
            }
        }
        sort(edges.begin(), edges.end(), cmp);

        //smaller triangle
        for(auto e : edges){
            int tmp = INF;
            int l = id[e.st], r = id[e.nd];
            if(l > 0 and l - 1 != r){
                tmp = fabs(area2(old[l], old[r], old[l - 1]));
                mn_area = min(tmp, mn_area);
            }
            if(r > 0 and r - 1 != l){
                tmp = fabs(area2(old[l], old[r], old[r - 1]));
                mn_area = min(tmp, mn_area);
            }
            if(l < (int)old.size() - 1 and l + 1 != r){
                tmp = fabs(area2(old[l], old[r], old[l + 1]));
                mn_area = min(tmp, mn_area);
            }
            if(r < (int)old.size() - 1 and r + 1 != l){
                tmp = fabs(area2(old[l], old[r], old[r + 1]));
                mn_area = min(tmp, mn_area);
            }
            swap(old[l], old[r]);
            swap(id[e.nd], id[e.st]);
        }
        printf("%d%s", mn_area/2, (mn_area % 2) ? ".5 " : ".0 ");
        printf("%d%s", mx_area/2, (mx_area % 2) ? ".5\n" : ".0\n");
    }
    return 0;
}
```

1.7 Greatest Quadrilater

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long long type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point rotate_ccw90(point p)   { return point(-p.y,p.x); }
point rotate_cw90(point p)    { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area_2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

int angle_less(const point& a1, const point& b1, const point& a2, const point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot(   a1, b1), abs(cross(   a1, b1)));
    point p2(dot(   a2, b2), abs(cross(   a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

vector<point> pts;

ll ternary_search(int l, int r){
    int lm = l, rm = r;
    while(r - l > 3) {
        int m1 = l + (r - l) / 3;
        int m2 = r - (r - l) / 3;
        ll f1 = abs(area_2(pts[lm], pts[m1], pts[rm]));
        ll f2 = abs(area_2(pts[lm], pts[m2], pts[rm]));
        if (f1 < f2) l = m1;
        else r = m2;
    }
    ll ans = 0;
    for(int i = l; i <= r; i++){
        ll aux = abs(area_2(pts[lm], pts[i], pts[rm]));
        ans = max(ans, aux);
    }
    return ans;
}

//Monotone chain O(nlog(n))
```

```
// #define REMOVE_REDUNDANT
#ifdef REMOVE_REDUNDANT
bool between(const point &a, const point &b, const point &c) {
    return (abs(area_2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void monotone_hull(vector<point> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<point> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area_2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area_2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

    #ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
        if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
        dn[0] = dn.back();
        dn.pop_back();
    }
    pts = dn;
    #endif
}

int main(){
    int n;
    cin >> n;
    pts.resize(n);
    vector<point> old(n);
    for(int i = 0; i < n; i++){
        cin >> pts[i].x >> pts[i].y;
        old[i] = pts[i];
    }
    monotone_hull(pts);
    n = pts.size();
    if(n < 3){
        cout << "0.0\n";
        return 0;
    }
    // db(n);
    if(n == 3){
        ll ans = 0;
        for(int i = 0; i < old.size(); i++){
            if(old[i].on_seg(pts[0], pts[1]) or old[i].on_seg(pts[1], pts[1]) or old[i].on_seg(pts[0], pts
                  [2])) continue;
            ans = max(ans, abs(area_2(pts[0], pts[1], pts[2])) - abs(area_2(pts[0], pts[1], old[i])));
            ans = max(ans, abs(area_2(pts[0], pts[1], pts[2])) - abs(area_2(pts[2], pts[1], old[i])));
            ans = max(ans, abs(area_2(pts[0], pts[1], pts[2])) - abs(area_2(pts[0], pts[2], old[i])));
        }
        cout << ans/2;
        if(ans % 2) cout << ".5\n";
        else cout << ".0\n";
        return 0;
    }
    for(int i = 0; i < n; i++) pts.push_back(pts[i]);
    // ll ans = 0;
    // for(int l = 0; l < n; l++){
    //     for(int r = l + 2; r <= l + n - 2; r++){
    //         ll top_triangle = ternary_search(l, r);
    //         ll bot_triangle = ternary_search(r, l + n);
    //         ans = max(ans, top_triangle + bot_triangle);
    //     }
    // }
    ll ans = 0;
    for(int i = 0; i < n; i++){
        for(int l1 = i + 1, r = i + 2, l2 = r + 1; r < i + n - 1 and l1 < r and l2 < i + n; r++){
            ll top = abs(area_2(pts[i], pts[l1], pts[r]));
            while(l1 + 1 < r and abs(area_2(pts[i], pts[l1], pts[r])) <= abs(area_2(pts[i], pts[l1 + 1], pts[
                  r]))){
                l1++;
                top = abs(area_2(pts[i], pts[l1], pts[r]));
            }

            ll bot = abs(area_2(pts[i], pts[l2], pts[r]));
            while(l2 + 1 < i + n and abs(area_2(pts[i], pts[l2], pts[r])) <= abs(area_2(pts[i], pts[l2 + 1],
                  pts[r]))){
```

```
                l2++;
                bot = abs(area_2(pts[i], pts[l2], pts[r]));
            }
            // db(top _ bot);
            ans = max(ans, top + bot);
        }
    }
    cout << ans/2;
    if(ans % 2) cout << ".5\n";
    else cout << ".0\n";
    return 0;
}
```

1.8 Smallest Quadrilater

```
/*
    rotating calipers (same problem as minimum triangle):
    idea:   sort points by x, than y
            sort all posible edges radially with respect to the edge perpendicular!
            start processing they on a sweep, every time you encounter one edge its time to process
            process means that the points from the edge will change places on the vector
            for min triangle: min triangle will be made with the current edge and adjacent points
            for max triangle: max triangle will be made with current edge and farthest points (0, n - 1)
            for this problem: pick first after r and first before l
*/
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long long type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0){}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return (x >= 0) - (x <= 0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y, p.x); }
point RotateCW90(point p)    { return point(p.y, -p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

point origin;

int above(point p){
    if(p.y == origin.y) return p.x > origin.x;
    return p.y > origin.y;
}

bool cmp(pair<point, point> a, pair<point, point> b){
    point p = RotateCW90(a.nd - a.st);
    point q = RotateCW90(b.nd - b.st);
    int tmp = above(q) - above(p);
    if(tmp) return tmp > 0;
    return p.dir(origin, q) > 0;
}

int n;
int main(){
    int t;
    cin >> t;
    for(int k = 1; k <= t; k++){
        scanf("%d", &n);
        vector<point> pts(n);
        for(int i = 0; i < n; i++){
            scanf("%lld%lld", &pts[i].x, &pts[i].y);
        }
        //sort points (base direction: x)
        map<point, int> id;
        sort(pts.begin(), pts.end());
        for(int i = 0; i < n; i++){
            point p = pts[i];
            id[p] = i;
        }

        //create edges and sort perpendicular radially
        vector<pair<point, point>> edges;
        for(int i = 0; i < n; i++){
            for(int j = i + 1; j < n; j++){
                edges.pb({pts[i], pts[j]});
            }
        }
        sort(edges.begin(), edges.end(), cmp);

        ll ans = LLONG_MAX;

        //min quad area
        //points will be adjacent if theres not 3 collinear points
        //a.nd - a.st => rotateCW
        //a.st - a.nd => rotateCCW
        for(auto e : edges){
            ll tmp = 0;
            int l = id[e.st], r = id[e.nd];
            //for not 3 collinear this never happens
            //if(l > r) swap(l, r);
            //choose first point above and first point below
            if(l > 0 and r < n - 1){
                tmp = abs(area2(pts[l - 1], pts[l], pts[r])) + abs(area2(pts[l], pts[r], pts[r + 1]));
                ans = min(ans, tmp);
            }
            swap(pts[l], pts[r]);
            swap(id[e.nd], id[e.st]);
        }
        printf("Case #%d: %lld\n", k, ans);
    }
    return 0;
}
```

1.9 Linear Translation

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e3+5;

typedef long double type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(point p) { return x == p.x and y == p.y; }
    bool operator !=(point p) { return x != p.x  or y != p.y; }
    bool operator <(const point p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
            );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y,p.x); }
point RotateCW90(point p)    { return point(p.y,-p.x); }

type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

point ProjectPointLine(point c, point a, point b) {
    ld r = dot(b - a,b - a);
    if (fabs(r) < EPS) return a;
    return a + (b - a)*dot(c - a, b - a)/dot(b - a, b - a);
}

point ProjectPointRay(point c, point a, point b) {
    ld r = dot(b - a, b - a);
    if (fabs(r) < EPS) return a;
    r = dot(c - a, b - a) / r;
    if (le(r, 0)) return a;
    return a + (b - a)*r;
}

point ProjectPointSegment(point c, point a, point b) {
    ld r = dot(b - a, b - a);
    if (fabs(r) < EPS) return a;
    r = dot(c - a, b - a)/r;
    if (le(r, 0)) return a;
    if (ge(r, 1)) return b;
    return a + (b - a)*r;
}

ld DistancePointLine(point c, point a, point b) {
    return c.dist2(ProjectPointLine(c, a, b));
}

ld DistancePointRay(point c, point a, point b) {
    return c.dist2(ProjectPointRay(c, a, b));
}
```

```
ld DistancePointSegment(point c, point a, point b) {
    return c.dist2(ProjectPointSegment(c, a, b));
}

//not tested
ld DistancePointPlane(ld x, ld y, ld z,
                      ld a, ld b, ld c, ld d)
{
    return fabs(a*x + b*y + c*z - d)/sqrt(a*a + b*b + c*c);
}

bool LinesParallel(point a, point b, point c, point d) {
    return fabs(cross(b - a, d - c)) < EPS;
}

bool LinesCollinear(point a, point b, point c, point d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

point lines_intersect(point p, point q, point a, point b) {
    point r = q - p, s = b - a, c(p%q, a%b);
    if (eq(r%s,0)) return point(LINF, LINF);
    return point(point(r.x, s.x) % c, point(r.y, s.y) % c) / (r%s);
}

point ComputeLineIntersection(point a, point b, point c, point d) {
    b = b - a; d = c - d; c = c - a;
    assert(dot(b, b) > EPS && dot(d, d) > EPS);
    return a + b*cross(c, d)/cross(b, d);
}

bool LineLineIntersect(point a, point b, point c, point d) {
    if(!LinesParallel(a, b, c, d)) return true;
    if(LinesCollinear(a, b, c, d)) return true;
    return false;
}

bool RayRayIntersect(point a, point b, point c, point d){
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    if (LinesCollinear(a, b, c, d)) {
        if(ge(dot(b - a, d - c), 0)) return true;
        if(ge(dot(a - c, d - c), 0)) return true;
        return false;
    }
    if(!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(ge(dot(inters - c, d - c), 0) && ge(dot(inters - a, b - a), 0)) return true;
    return false;
}

bool SegmentSegmentIntersect(point a, point b, point c, point d) {
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    int d1, d2, d3, d4;
    d1 = direction(a, b, c);
    d2 = direction(a, b, d);
    d3 = direction(c, d, a);
    d4 = direction(c, d, b);
    if (d1*d2 < 0 and d3*d4 < 0) return 1;
    return a.on_seg(c, d) or b.on_seg(c, d) or
            c.on_seg(a, b) or b.on_seg(c, d);
}

bool SegmentLineIntersect(point a, point b, point c, point d){
    if(!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(inters.on_seg(a, b)) return true;
    return false;
}

bool SegmentRayIntersect(point a, point b, point c, point d){
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    if (LinesCollinear(a, b, c, d)) {
        if(c.on_seg(a, b)) return true;
        if(ge(dot(d - c, a - c), 0)) return true;
        return false;
    }
    if(!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(!inters.on_seg(a, b)) return false;
    if(ge(dot(inters - c, d - c), 0)) return true;
    return false;
}

bool RayLineIntersect(point a, point b, point c, point d){
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    if (!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(!LineLineIntersect(a, b, c, d)) return false;
    if(ge(dot(inters - a, b - a), 0)) return true;
    return false;
}

ld DistanceSegmentLine(point a, point b, point c, point d){
    if(SegmentLineIntersect(a, b, c, d)) return 0;
    return min(DistancePointLine(a, c, d), DistancePointLine(b, c, d));
}

ld DistanceSegmentRay(point a, point b, point c, point d){
    if(SegmentRayIntersect(a, b, c, d)) return 0;
    ld min1 = DistancePointSegment(c, a, b);
    ld min2 = min(DistancePointRay(a, c, d), DistancePointRay(b, c, d));
    return min(min1, min2);
}

ld DistanceSegmentSegment(point a, point b, point c, point d){
    if(SegmentSegmentIntersect(a, b, c, d)) return 0;
    ld min1 = min(DistancePointSegment(c, a, b), DistancePointSegment(d, a, b));
    ld min2 = min(DistancePointSegment(a, c, d), DistancePointSegment(b, c, d));
    return min(min1, min2);
}

ld DistanceRayLine(point a, point b, point c, point d){
    if(RayLineIntersect(a, b, c, d)) return 0;
    ld min1 = DistancePointLine(a, c, d);
    return min1;
}

ld DistanceRayRay(point a, point b, point c, point d){
    if(RayRayIntersect(a, b, c, d)) return 0;
    ld min1 = min(DistancePointRay(c, a, b), DistancePointRay(a, c, d));
    return min1;
}

ld DistanceLineLine(point a, point b, point c, point d){
    if(LineLineIntersect(a, b, c, d)) return 0;
    return DistancePointLine(a, c, d);
}

int n[2];
point o[2], hull[2][N];

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    for(int i = 0; i < 2; i++){
        cin >> o[i].x >> o[i].y;
        cin >> n[i];
        for(int j = 0; j < n[i]; j++) cin >> hull[i][j].x >> hull[i][j].y;
    }
    point shift = o[1] - o[0];
    type d = shift.abs2();
    for(int i = 0; i < n[0]; i++) hull[0][i] = hull[0][i] + shift;
    for(int k = 0; k < 2; k++){
        for(int i = 0; i < n[k]; i++){
            for(int j = 0; j < n[k^1]; j++){
                type mn, mx;
                mn = DistancePointSegment(hull[k][i], hull[k^1][j], hull[k^1][(j + 1)%n[k^1]]);
                mx = max(hull[k][i].dist2(hull[k^1][j]), hull[k][i].dist2(hull[k^1][(j + 1)%n[k^1]]));
                if(ge(d, mn) and le(d, mx)){
                    cout << "YES\n";
                    return 0;
                }
            }
        }
    }
    cout << "NO\n";
    return 0;
}
```

1.10 Maximize/Minimize Scalar Product

```
#include <bits/stdc++.h>

using namespace std;

#define st first
```

```cpp
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long long type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }
};

int direction(point o, point p, point q) { return p.dir(o, q); }

point rotate_ccw90(point p)   { return point(-p.y,p.x); }
point rotate_cw90(point p)    { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area_2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

int angle_less(const point& a1, const point& b1, const point& a2, const point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot(   a1, b1), abs(cross(   a1, b1)));
    point p2(dot(   a2, b2), abs(cross(   a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

//Monotone chain O(nlog(n))
#define REMOVE_REDUNDANT
#ifdef REMOVE_REDUNDANT
bool between(const point &a, const point &b, const point &c) {
    return (abs(area_2(a,b,c)) == 0 && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void monotone_hull(vector<point> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<point> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area_2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area_2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

    #ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
        if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
        dn[0] = dn.back();
        dn.pop_back();
    }
    pts = dn;
    #endif
}

int maximizeScalarProduct(vector<point> &hull, point vec) {
        // this code assumes that there are no 3 colinear points
        int ans = 0;
        int n = hull.size();
        if(n < 20) {
                for(int i = 0; i < n; i++) {
                        if(hull[i] * vec > hull[ans] * vec) {
                                ans = i;
                        }
                }
        } else {
                if(hull[1] * vec > hull[ans] * vec) {
                        ans = 1;
                }
                for(int rep = 0; rep < 2; rep++) {
                        int l = 2, r = n - 1;
                        while(l != r) {
                                int mid = (l + r + 1) / 2;
                                bool flag = hull[mid] * vec >= hull[mid-1] * vec;
                                if(rep == 0) { flag = flag && hull[mid] * vec >= hull[0] * vec; }
                                else { flag = flag || hull[mid-1] * vec < hull[0] * vec; }
                                if(flag) {
                                        l = mid;
                                } else {
                                        r = mid - 1;
                                }
                        }
                        if(hull[ans] * vec < hull[l] * vec) {
                                ans = l;
                        }
                }
        }
        return ans;
}

struct line{
    type a, b, c;

    line (type aa = 0, type bb = 0, type cc = 0) : a(aa), b(bb), c(cc){}
};

int n, m;
vector<point> hull;
vector<line> h;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n >> m;
    hull.resize(m), h.resize(n);
    for(int i = 0; i < n; i++) cin >> h[i].a >> h[i].b >> h[i].c;
    for(int i = 0; i < m; i++) cin >> hull[i].x >> hull[i].y;

    monotone_hull(hull);
    vector<int> ans;
    for(int i = 0; i < n; i++){
        int mx = maximizeScalarProduct(hull, point(h[i].a, h[i].b));
        int mn = maximizeScalarProduct(hull, point(-h[i].a, -h[i].b));
        type mx_value = (hull[mx].x * h[i].a + hull[mx].y * h[i].b + h[i].c);
        type mn_value = (hull[mn].x * h[i].a + hull[mn].y * h[i].b + h[i].c);
        if(mx_value > 0) mx_value = 1;
        else if(mx_value < 0) mx_value = -1;
        if(mn_value > 0) mn_value = 1;
        else if(mn_value < 0) mn_value = -1;
        if(mx_value * mn_value <= 0) ans.push_back(i + 1);
    }
    cout << ans.size() << "\n";
    for(int i = 0; i < ans.size(); i++) cout << ans[i] << " ";
    cout << "\n";
    return 0;
}
```

1.11 Maximize Scalar Product

```
//using farthest point in direction
//function from: https://github.com/tfg50/Competitive-Programming/blob/master/Biblioteca/Math/2D%20Geometry/
      ConvexHull.cpp
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long double type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point rotate_ccw90(point p)   { return point(-p.y,p.x); }
point rotate_cw90(point p)    { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area_2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

int angle_less(const point& a1, const point& b1, const point& a2, const point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot(   a1, b1), abs(cross(   a1, b1)));
    point p2(dot(   a2, b2), abs(cross(   a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

//Monotone chain O(nlog(n))

void monotone_hull(vector<point> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<point> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area_2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area_2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);
}

point project_point_line(point c, point a, point b) {
    ld r = dot(b - a,b - a);
    if (fabs(r) < EPS) return a;
    return a + (b - a)*dot(c - a, b - a)/dot(b - a, b - a);
}

ld distance_point_line(point c, point a, point b) {
    return c.dist2(project_point_line(c, a, b));
}

int maximizeScalarProduct(vector<point> &hull, point vec) {
        // this code assumes that there are no 3 colinear points
        int ans = 0;
        int n = hull.size();
        if(n < 20) {
                for(int i = 0; i < n; i++) {
                        if(hull[i] * vec > hull[ans] * vec) {
                                ans = i;
                        }
                }
        } else {
                if(hull[1] * vec > hull[ans] * vec) {
                        ans = 1;
                }
                for(int rep = 0; rep < 2; rep++) {
                        int l = 2, r = n - 1;
                        while(l != r) {
                                int mid = (l + r + 1) / 2;
                                bool flag = hull[mid] * vec >= hull[mid-1] * vec;
                                if(rep == 0) { flag = flag && hull[mid] * vec >= hull[0] * vec; }
                                else { flag = flag || hull[mid-1] * vec < hull[0] * vec; }
                                if(flag) {
                                        l = mid;
                                } else {
                                        r = mid - 1;
                                }
                        }
                        if(hull[ans] * vec < hull[l] * vec) {
                                ans = l;
                        }
                }
        }
        return ans;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    //freopen("in.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);
    int k = 0;
    while(true){
        ++k;
        int n;
        cin >> n;
        if(!n) return 0;
        vector<point> pts(n);
        for(int i = 0; i < n; i++){
            cin >> pts[i].x >> pts[i].y;
        }
        monotone_hull(pts);
        n = pts.size();
        ld ans = LINF;
        for(int l = 0; l < n; l++){
            //maximize scalar product: hull ccw, rotate ccw / hull cw, rotate cw (if not sure test both)
            int r = maximizeScalarProduct(pts, rotate_ccw90(pts[(l + 1)%n] - pts[l]));
            ans = min(ans, distance_point_line(pts[r], pts[l], pts[(l + 1)%n]));
        }
        cout << "Case " << k << ": " << setprecision(2) << fixed << sqrt(ans) << "\n";
    }
    return 0;
}
```

1.12 Maximize Function (x*y = k) (Scalar Product)

```
/*maximize sum(aihi) * sum(aipi)
    the idea is to convert some variables to end on a 2d problem
    divide everyone by its cost, so each unit cost one, then multiply everyone by money (or divide by c / m)
    this way you have 1 coin, every troop costs 1 and you have to maximize h * p, among the possible troops
    creating a grid where the x axis is h and y axis is p, the possible combinations for this its the same
          from problem https://www.spoj.com/problems/PERFUME/en/
    this means the answer will be inside(edges included) the convex hull, the problem relies on maximizing: x
          *y
    doing some math and looking for the graph of y = k/x, its visible the answer will be on some edge, as a
          unimodal function
    just brute force edges with a ternary search for each one
*/
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long double type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }
    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point rotate_ccw90(point p)   { return point(-p.y,p.x); }
point rotate_cw90(point p)    { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area_2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

int angle_less(const point& a1, const point& b1, const point& a2, const point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot(   a1, b1), abs(cross(   a1, b1)));
    point p2(dot(   a2, b2), abs(cross(   a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

//Monotone chain O(nlog(n))
#define REMOVE_REDUNDANT
#ifdef REMOVE_REDUNDANT
bool between(const point &a, const point &b, const point &c) {
    return (fabs(area_2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void monotone_hull(vector<point> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<point> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area_2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area_2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

    #ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
        if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
        dn[0] = dn.back();
        dn.pop_back();
    }
    pts = dn;
    #endif
}

//Faster version - 300 iteratons up to 1e-6 precision
ld ternary_search(point p, point q, int No = 300){
    // y = m * x + n;
    if(eq(p.x, q.x)) return p.x * max(q.y, p.y);
    if(eq(p.y, q.y)) return p.y * max(p.x, q.x);
    if(p.x > q.x) swap(p, q);
    ld m = (q.y - p.y) / (q.x - p.x);
    ld n = p.y - m * p.x;
    ld l = p.x, r = q.x;
    // for(int i = 0; i < No; i++){
    while(r - l > EPS){
        //db(l _ r);
        ld m1 = l + (r - l) / 3;
        ld m2 = r - (r - l) / 3;
        // if (f(m1) > f(m2))
        if (m1 * (m * m1 + n) < m2  * (m * m2 + n))
            l = m1;
        else
            r = m2;
    }
    //db(l);
    return l * (m * l + n);
}

ld c[N], h[N], p[N];
int main(){
    freopen("Mobilization-1001.in", "r", stdin);
    freopen("out1.txt", "w", stdout);
    int n, m;
    scanf("%d%d", &n, &m);
    vector<point> pts(n);
    for(int i = 0; i < n; i++){
        scanf("%Lf%Lf%Lf", &c[i], &h[i], &p[i]);
        pts[i].x = h[i] / (c[i] / m);
        pts[i].y = p[i] / (c[i] / m);
    }
    monotone_hull(pts);
    n = pts.size();
    ld ans = -LINF;
    for(int i = 0; i < n; i++){
        ans = max(ans, ternary_search(pts[i], pts[(i + 1)%n]));
    }
    printf("%.2Lf\n", ans);
    return 0;
}
```

1.13 Convex Hull/Point Inside Hull

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef unsigned long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
```

```cpp
const int N = 1e5+5;

typedef long long type;
//for big coordinates change to long long

bool ge(type x, type y) { return x >= y; }
bool le(type x, type y) { return x <= y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y,p.x); }
point RotateCW90(point p)    { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

int angleLess(const point& a1, const point& b1, const point& a2, const point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot(   a1, b1), abs(cross(   a1, b1)));
    point p2(dot(   a2, b2), abs(cross(   a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

//Monotone chain O(nlog(n))
#define REMOVE_REDUNDANT
#ifdef REMOVE_REDUNDANT
bool between(const point &a, const point &b, const point &c) {
    return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void ConvexHull(vector<point> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<point> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

    #ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
        if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
        dn[0] = dn.back();
        dn.pop_back();
    }
    #endif
}

bool pointInTriangle(point a, point b, point c, point cur){
    ll s1 = 1ull * abs(cross(b - a, c - a));
    ll s2 = 1ull * abs(cross(a - cur, b - cur)) + 1ull * abs(cross(b - cur, c - cur)) + 1ull * abs(cross(c -
          cur, a - cur));
    return s1 == s2;
}

void sort_lex_hull(vector<point> &hull){
    int n = hull.size();
    //Sort hull by x
    int pos = 0;
    for(int i = 1; i < n; i++) if(hull[i] <  hull[pos]) pos = i;
    rotate(hull.begin(), hull.begin() + pos, hull.end());
}

//determine if point is inside or on the boundary of a polygon (O(logn))
bool pointInConvexPolygon(vector<point> &hull, point cur){
    int n = hull.size();
    //Corner cases: point outside most left and most right wedges
    if(cur.dir(hull[0], hull[1]) != 0 && cur.dir(hull[0], hull[1]) != hull[n - 1].dir(hull[0], hull[1]))
        return false;
    if(cur.dir(hull[0], hull[n - 1]) != 0 && cur.dir(hull[0], hull[n - 1]) != hull[1].dir(hull[0], hull[n -
         1]))
        return false;

    //Binary search to find which wedges it is between
    int l = 1, r = n - 1;
    while(r - l > 1){
        int mid = (l + r)/2;
        if(cur.dir(hull[0], hull[mid]) <= 0)l = mid;
        else r = mid;
    }
    return pointInTriangle(hull[l], hull[l + 1], hull[0], cur);
}

bool PointOnPolygon(vector<point> &p, point q) {
    for (int i = 0; i < p.size(); i++){
        if(p[i] == q or p[(i + 1)%p.size()] == q) return true;
        if(q.on_seg(p[i], p[(i + 1)%p.size()])) return true;
    }
    return false;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int t;
    cin >> t;
    while(t--){
        int n, q;
        cin >> n >> q;
        vector<point> tmp(n);
        set<point> pts;
        vector<vector<point>> hull;
        for(int i = 0; i < n; i++){
            point p;
            cin >> p.x >> p.y;
            pts.insert(p);
        }
        while(pts.size() > 2){
            vector<point> tmp, rem;
            for(auto p: pts) tmp.pb(p);
            hull.pb(tmp);
            ConvexHull(hull[ hull.size() - 1]);
            sort_lex_hull(hull[hull.size() - 1]);
            //db(hull[hull.size() - 1].size());
            for(auto p : pts){
                if(PointOnPolygon(hull[hull.size() - 1], p)){
                    rem.pb(p);
                }
            }
            for(auto p : rem) pts.erase(p);
        }
        for(int i = 0; i < q; i++){
            point p;
            cin >> p.x >> p.y;
            int ans = 0;
            for(int i = 0; i < hull.size(); i++){
                if(hull[i].size() > 2){
                    if(pointInConvexPolygon(hull[i], p) and !PointOnPolygon(hull[i], p)){
                        ans++;
                    }
                    else break;
                }
            }
            cout << ans << "\n";
        }
    }
    return 0;
}
```

1.14 Vectorial Space/Point Inside Hull

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-13, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long double type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y,p.x); }
point RotateCW90(point p)    { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

int angleLess(const point& a1, const point& b1, const point& a2, const point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot(   a1, b1), abs(cross(   a1, b1)));
    point p2(dot(   a2, b2), abs(cross(   a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}
```

```
ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

//Monotone chain O(nlog(n))
#define REMOVE_REDUNDANT
#ifdef REMOVE_REDUNDANT
bool between(const point &a, const point &b, const point &c) {
    return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void ConvexHull(vector<point> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<point> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

    #ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
        if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
        dn[0] = dn.back();
        dn.pop_back();
    }
    pts = dn;
    #endif
}

bool pointInTriangle(point a, point b, point c, point cur){
    ld s1 = abs(cross(b - a, c - a));
    ld s2 = abs(cross(a - cur, b - cur)) + abs(cross(b - cur, c - cur)) + abs(cross(c - cur, a - cur));
    return eq(s1, s2);
}

void sort_lex_hull(vector<point> &hull){
    int n = hull.size();

    //Sort hull by x
    int pos = 0;
    for(int i = 1; i < n; i++) if(!ge(hull[i].x,  hull[pos].x)) pos = i;
    rotate(hull.begin(), hull.begin() + pos, hull.end());
}

bool pointInConvexPolygon(vector<point> &hull, point cur){
    int n = hull.size();
    //Corner cases: point outside most left and most right wedges
    if(!eq(cur.dir(hull[0], hull[1]), 0) && cur.dir(hull[0], hull[1]) != hull[n - 1].dir(hull[0], hull[1]))
        return false;
    if(!eq(cur.dir(hull[0], hull[n - 1]), 0) && cur.dir(hull[0], hull[n - 1]) != hull[1].dir(hull[0], hull[n
        - 1]))
        return false;

    //Binary search to find which wedges it is between
    int l = 1, r = n - 1;
    while(r - l > 1){
        int mid = (l + r)/2;
        if(le(cur.dir(hull[0], hull[mid]), 0)) l = mid;
        else r = mid;
    }
    return pointInTriangle(hull[l], hull[l + 1], hull[0], cur);
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int t;
    cin >> t;
    while(t--){
        int n, q;
        cin >> n;
        vector<point> mix;
        for(int i = 0; i < n; i++){
            point p;
            cin >> p.x >> p.y;
```

```
            mix.pb({p.x, p.y});
        }
        ConvexHull(mix);
        sort_lex_hull(mix);
        cin >> q;
        for(int k = 0; k < q; k++){
            point p;
            cin >> p.x >> p.y;
            if(mix.size() == 1){
                if(p.dist(mix[0]) < EPS) cout << "Yes\n";
                else cout << "No\n";
            }
            else if(mix.size() == 2){
                if(p.on_seg(mix[0], mix[1])){
                    cout << "Yes\n";
                }
                else cout << "No\n";
            }
            else{
                if(pointInConvexPolygon(mix, p)) cout << "Yes\n";
                else cout << "No\n";
            }
        }
        if(t > 0) cout << "\n";
    }
    return 0;
}
```

1.15 Polygon Tangent

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 3e5+5;

typedef long long type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }
    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y,p.x); }
point RotateCW90(point p)    { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

int angleLess(const point& a1, const point& b1, const point& a2, const point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot(   a1, b1), abs(cross(   a1, b1)));
    point p2(dot(   a2, b2), abs(cross(   a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

point origin;

int above(point p){
    if(p.y == origin.y) return p.x > origin.x;
    return p.y > origin.y;
}

bool cmp(point p, point q){
    int tmp = above(q) - above(p);
    if(tmp) return tmp > 0;
    return p.dir(origin,q) > 0;
    //if(p.dir(origin,q) == 0) return p.abs2
}

//Monotone chain O(nlog(n))

void ConvexHull(vector<point> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<point> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);
}

bool pointInTriangle(point a, point b, point c, point cur){
    ll s1 = abs(cross(b - a, c - a));
    ll s2 = abs(cross(a - cur, b - cur)) + abs(cross(b - cur, c - cur)) + abs(cross(c - cur, a - cur));
    return s1 == s2;
}

void sort_lex_hull(vector<point> &hull){
    int n = hull.size();

    //Sort hull by x
    int pos = 0;
    for(int i = 1; i < n; i++) if(hull[i] <  hull[pos]) pos = i;
    rotate(hull.begin(), hull.begin() + pos, hull.end());
}

//determine if point is inside or on the boundary of a polygon (O(logn))
bool pointInConvexPolygon(vector<point> &hull, point cur){
    int n = hull.size();
    //Corner cases: point outside most left and most right wedges
    if(cur.dir(hull[0], hull[1]) != 0 && cur.dir(hull[0], hull[1]) != hull[n - 1].dir(hull[0], hull[1]))
        return false;
    if(cur.dir(hull[0], hull[n - 1]) != 0 && cur.dir(hull[0], hull[n - 1]) != hull[1].dir(hull[0], hull[n -
          1]))
        return false;

    //Binary search to find which wedges it is between
    int l = 1, r = n - 1;
    while(r - l > 1){
        int mid = (l + r)/2;
        if(cur.dir(hull[0], hull[mid]) <= 0)l = mid;
        else r = mid;
    }
    return pointInTriangle(hull[l], hull[l + 1], hull[0], cur);
}

int rtang(vector<point> &hull, point cur){
    int n = hull.size();
    int l = 0, r = n - 1;
    //borders
    if((hull[l].dir(cur, hull[(l + 1)%n]) > 0) and (hull[l].dir(cur, hull[(l - 1 + n)%n]) > 0)) return l;
    if((hull[r].dir(cur, hull[(r + 1)%n]) > 0) and (hull[r].dir(cur, hull[(r - 1 + n)%n]) > 0)) return r;
    l++, r--;
    while(l < r){
        int m = (l + r)/2;
        //db(l _ r _ m _ cur.dir(hull[m], hull[(m + 1)%n]) _ cur.dir(hull[m], hull[(m - 1 + n)%n]));
        if(hull[m].dir(cur, hull[(m + 1)%n]) < 0) l = (m + 1);
        else if(hull[m].dir(cur, hull[(m - 1 + n)%n]) < 0) r = m - 1;
        else return m;
    }
    return l;
}

int ltang(vector<point> &hull, point cur){
    int n = hull.size();
    int l = 0, r = n - 1;
    //borders
    if((hull[l].dir(cur, hull[(l + 1)%n]) < 0) and (hull[l].dir(cur, hull[(l - 1 + n)%n]) < 0)) return l;
    if((hull[r].dir(cur, hull[(r + 1)%n]) < 0) and (hull[r].dir(cur, hull[(r - 1 + n)%n]) < 0)) return r;
    l++, r--;
    while(l < r){
        int m = (l + r)/2;
        //db(l _ r _ m _ cur.dir(hull[m], hull[(m + 1)%n]) _ cur.dir(hull[m], hull[(m - 1 + n)%n]));
        if(hull[m].dir(cur, hull[(m + 1)%n]) > 0) l = (m + 1);
        else if(hull[m].dir(cur, hull[(m - 1 + n)%n]) > 0) r = m - 1;
        else return m;
    }
    return l;
}

int tangent(vector<point> &hull, point vec, int dir_flag) {
        // this code assumes that there are no 3 colinear points
    // -1 for right tangent
    //  1 for left tangent
        int ans = 0;
        int n = hull.size();
        if(n < 20) {
                for(int i = 0; i < n; i++) {
                        if(hull[ans].dir(vec, hull[i]) == dir_flag) {
                                ans = i;
                        }
                }
        } else {
                if(hull[ans].dir(vec, hull[1]) == dir_flag) {
                        ans = 1;
                }
                for(int rep = 0; rep < 2; rep++) {
                        int l = 2, r = n - 1;
                        while(l != r) {
                                int mid = (l + r + 1) / 2;
                                bool flag = hull[mid - 1].dir(vec, hull[mid]) == dir_flag;
                                if(rep == 0) { flag = flag && (hull[0].dir(vec, hull[mid]) == dir_flag); }
                                else { flag = flag || (hull[0].dir(vec, hull[mid - 1]) != dir_flag); }
                                if(flag) {
                                        l = mid;
                                } else {
                                        r = mid - 1;
                                }
                        }
                        if(hull[ans].dir(vec, hull[l]) == dir_flag) {
                                ans = l;
                        }
                }
        }
        return ans;
}

ll area[N];
point p[N];
//avoid using long double for comparisons, change type and remove division by 2
void ComputeSignedArea(const vector<point> &hull) {
    int n = (int)hull.size();
    for(int i = 0; i < n; i++){
        p[i] = p[i + n] = hull[i];
    }
    for(int i = 0; i < 2*n - 1; i++) {
        int j = (i+1);
        area[j] = area[i];
        area[j] += p[i].x*p[j].y - p[j].x*p[i].y;
    }
}

int n, k;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n >> k;
    vector <point> hull, pts, tmp;
    for(int i = 0; i < n; i++){
        point p;
        cin >> p.x >> p.y;
        if(i < k) hull.pb(p);
        else pts.pb(p);
    }
    ConvexHull(hull);
    //for(auto p : hull) db(p);
    sort_lex_hull(hull);
    for(int i = 0; i < pts.size(); i++){
        if(!pointInConvexPolygon(hull, pts[i])) tmp.push_back(pts[i]);
    }
    pts.clear();
    pts = tmp;
    //for(auto p : pts) db(p);
    ComputeSignedArea(hull);
    ll cur_area = abs(area[(int)hull.size()]);
    ll ans = cur_area;
    n = (int)hull.size();
    //db(ans);
    for(int i = 0; i < pts.size(); i++){
        int l, r;
        r = tangent(hull, pts[i], -1);
        l = tangent(hull, pts[i], 1);
        //db(pts[i] _ p[l] _ p[r] _ l _ r);
        //test points
        //if(r < l) swap(r, l);
        //db(l _ r);
        //db(hull[l].dir(pts[i], hull[r]));
        if(r < l) r += n;
        //db(l _ r);
        ll dif_area = abs((area[r] - area[l] + p[r].x*p[l].y - p[l].x*p[r].y));
        ll tot = cur_area + abs(area2(pts[i], p[l], p[r])) - abs(dif_area);
        ans = max(ans, tot);
    }
    cout << ans/2;
    if(ans%2) cout << ".5";
    else cout << ".0";
    cout << "\n";

    return 0;
}
```

1.16 Set of edges/Line Sweep

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long long type;
//for big coordinates change to long long

//BASICS

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x+p.x, y+p.y); }
    point operator -(point p) { return point(x-p.x, y-p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    type operator *(point p) { return x*p.x + y*p.y; }
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(point p) { return x == p.x and y == p.y; }
    bool operator !=(point p) { return x != p.x  or y != p.y; }
    bool operator <(const point p) const { return (x < p.x) or (x == p.x and y < p.y); }

    int dir(point o, point p) {
    type x = (*this - o) % (p - o);
    return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and
            ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y));
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
    return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x-sin*y, sin*x+cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }
    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }
};


point RotateCCW90(point p)   { return point(-p.y,p.x); }
point RotateCW90(point p)    { return point(p.y,-p.x); }

ld dot(point p, point q)     { return p.x*q.x+p.y*q.y; }
ld cross(point p, point q)   { return p.x*q.y-p.y*q.x; }

type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

int direction(point o, point p, point q) { return p.dir(o, q); }
point origin;

int above(point p){
    if(p.y == origin.y) return p.x > origin.x;
    return p.y > origin.y;
}

bool cmp(pair<point, pii> a, pair<point, pii> b){
    point p = a.st, q = b.st;
    int tmp = above(q) - above(p);
    if(tmp) return tmp > 0;
    return p.dir(origin,q) > 0;
    //Be Careful: p.dir(origin,q) == 0
}

bool SegmentSegmentIntersect(point a, point b, point c, point d) {
    if (a.dist2(c) < EPS || a.dist2(d) < EPS ||
        b.dist2(c) < EPS || b.dist2(d) < EPS) return true;
    int d1, d2, d3, d4;
    d1 = direction(a, b, c);
    d2 = direction(a, b, d);
    d3 = direction(c, d, a);
    d4 = direction(c, d, b);
    if (d1*d2 < 0 and d3*d4 < 0) return 1;
    return a.on_seg(c, d) or b.on_seg(c, d) or
            c.on_seg(a, b) or d.on_seg(a, b);
}

int s, k , w;
ll ans[N];
point kid[N];
pair <point, point> wall[N];

bool cmp2(int a, int b){
    point u = wall[a].st, v = wall[a].nd;
    point p = wall[b].st, q = wall[b].nd;
    //if u comes first (radially) than p, if u-v intersects origin - p than, u - v comes first, because its
          closer
    if (cross(u - origin, p - origin) > 0) return SegmentSegmentIntersect(u, v, origin, p);
    //else (p comes first than u), if p - q intersects u - origin, than p - q comes first, because its closer
    return !SegmentSegmentIntersect(u, origin, p, q);
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    //read
    while(cin >> s >> k >> w){
        for(int i = 0; i < k; i++){
            cin >> kid[i].x >> kid[i].y;
        }
        for(int i = 0; i < w; i++){
            cin >> wall[i].st.x >> wall[i].st.y;
            cin >> wall[i].nd.x >> wall[i].nd.y;
        }
        //sweep
        for(int i = 0; i < s; i++){
            //init
            origin = kid[i];
            ans[i] = 0;
            //point, type, id
            vector <pair<point, pii>> sweep;
            //2 for children in sweep
            for(int j = 0; j < k; j++){
                if(i != j) sweep.pb({kid[j], {2, j}});
            }
            //0 for opening wall, 1 for closing wall
            for(int j = 0; j < w; j++){
                //if order is reversed, swap it
                if(wall[j].st.dir(origin, wall[j].nd) < 0) swap(wall[j].st, wall[j].nd);
                sweep.pb({wall[j].st, {0, j}});
                sweep.pb({wall[j].nd, {1, j}});
            }
            //sort points radially with respect to the origin, the kid
            sort(sweep.begin(), sweep.end(), cmp);
            //order segments on set
            set<int,bool(*)(int,int)> ps(cmp2);
            //look for walls that are already closing view
            for(auto q : sweep){
                if(q.nd.st == 0) ps.insert(q.nd.nd);
                if(q.nd.st == 1) if(ps.count(q.nd.nd)) ps.erase(q.nd.nd);
            }
            //radial sweep: look for kids that are not being blocked by view
            for(auto q : sweep){
                if(q.nd.st == 2){
                    if(!ps.size()) ans[i]++;
                    //if the segment origin - kid does not intersect the closest wall, than the kid is being
                          seen
                    else if(!SegmentSegmentIntersect(origin, q.st, wall[*ps.begin()].st, wall[*ps.begin()].nd
                          )) ans[i]++;
                }
                else if(q.nd.st == 0) ps.insert(q.nd.nd);
                else if(ps.count(q.nd.nd)) ps.erase(q.nd.nd);
            }
        }
        for(int i = 0; i < s; i++) cout << ans[i] << "\n";
    }
    return 0;
}
```

1.17 Shamos Hoey - Set of edges/Line Sweep

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef long long type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }
    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y,p.x); }
point RotateCW90(point p)    { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions above:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

int angleLess(const point& a1, const point& b1, const point& a2, const point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot(   a1, b1), abs(cross(   a1, b1)));
    point p2(dot(   a2, b2), abs(cross(   a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

//Shamos - Hoey for test polygon simple in O(nlog(n))
struct edge{
    point ini, fim;
    edge(point ini = point(0,0), point fim = point(0,0)) : ini(ini), fim(fim) {}
};
```

```
bool operator < (const edge& a, const edge& b) {
    if (a.ini == b.ini) return direction(a.ini, a.fim, b.fim) < 0;
    if (a.ini.x < b.ini.x) return direction(a.ini, a.fim, b.ini) < 0;
    return direction(a.ini, b.fim, b.ini) < 0;
}

bool cmp(pair<point, pii> a, pair<point, pii> b){
    if(a.st.x == b.st.x){
        if(a.nd.st == b.nd.st){
            return a.st.y > b.st.y;
        }
        return a.nd.st < b.nd.st;
    }
    return a.st.x < b.st.x;
}

void left_sweep(const vector<edge> &pts, vector<set<int>> &par, vector<set<int>> &son, vector<ll> &water){
    vector <pair<point, pii>> eve;
    vector <pair<edge, int>> edgs;
    set <pair<edge, int>> sweep;
    int n = (int)pts.size();
    for(int i = 0; i < n; i++){
        edgs.pb(make_pair(pts[i], i));
        eve.pb({pts[i].ini, {0, i}});
        eve.pb({pts[i].fim, {1, i}});
    }
    sort(eve.begin(), eve.end(), cmp);
    int last = -INF;
    for(auto e : eve){
        if(!e.nd.st){
            if(!sweep.size()) last = e.st.x;
            else{
                auto cur = sweep.lower_bound(edgs[e.nd.nd]);
                if(cur == sweep.begin()){
                    water[cur -> nd] += (e.st.x - last);
                    last = e.st.x;
                }
            }
            sweep.insert(edgs[e.nd.nd]);
        }
        else{
            auto below = sweep.upper_bound(edgs[e.nd.nd]);
            auto cur = below, above = --cur;
            if(cur == sweep.begin()){
                water[cur -> nd] += (e.st.x - last);
                last = e.st.x;
            }
            if(edgs[e.nd.nd].st.ini.y > edgs[e.nd.nd].st.fim.y){
                if(below != sweep.end()){
                    son[e.nd.nd].insert(below->nd);
                    par[below->nd].insert(e.nd.nd);
                }
            }
            sweep.erase(cur);
        }
    }
}

void right_sweep(const vector<edge> &pts, vector<set<int>> &par, vector<set<int>> &son, vector<ll> &water){
    vector <pair<point, pii>> eve;
    vector <pair<edge, int>> edgs;
    set <pair<edge, int>> sweep;
    int n = (int)pts.size();
    for(int i = 0; i < n; i++){
        edgs.pb(make_pair(pts[i], i));
        eve.pb({pts[i].ini, {0, i}});
        eve.pb({pts[i].fim, {1, i}});
    }
    sort(eve.begin(), eve.end(), cmp);
    for(auto e : eve){
        if(!e.nd.st){
            sweep.insert(edgs[e.nd.nd]);
        }
        else{
            auto below = sweep.upper_bound(edgs[e.nd.nd]);
            auto cur = below, above = --cur;
            if(edgs[e.nd.nd].st.ini.y > edgs[e.nd.nd].st.fim.y){
                if(below != sweep.end()){
                    son[e.nd.nd].insert(below->nd);
                    par[below->nd].insert(e.nd.nd);
                }
            }
            sweep.erase(cur);
        }
    }
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int t;
    cin >> t;
    while(t--){
        int n;
        cin >> n;
        vector<edge> pts(n);
        vector<set<int>> son(n);
        vector<set<int>> par(n);
        vector<ll> water(n);
        for(int i = 0; i < n; i++){
            cin >> pts[i].ini.x >> pts[i].ini.y;
            cin >> pts[i].fim.x >> pts[i].fim.y;
        }
        left_sweep(pts, par, son, water);
        for(int i = 0; i < n; i++){
            swap(pts[i].ini, pts[i].fim);
            pts[i].ini.x = -pts[i].ini.x;
            pts[i].fim.x = -pts[i].fim.x;
        }
        right_sweep(pts, par, son, water);
        vector<int> upd;
        for(int i = 0; i < n; i++) if(par[i].empty()) upd.pb(i);

        while(!upd.empty()){
            int i = upd.back();
            upd.pop_back();
            for(auto x : son[i]){
                water[x] += water[i];
                par[x].erase(i);
                if(par[x].empty()) upd.pb(x);
            }
        }
        for(int i = 0; i < n; i++) cout << water[i] << "\n";
    }
    return 0;
}
```

1.18 Shamos Hoey

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 2e5+5, M = 30, K = 25;

typedef long long type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y,p.x); }
point RotateCW90(point p)    { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

void sort_lex_hull(vector<point> &hull){
    int n = hull.size();

    //Sort hull by x
    int pos = 0;
    for(int i = 1; i < n; i++) if(hull[i] <  hull[pos]) pos = i;
    rotate(hull.begin(), hull.begin() + pos, hull.end());
}

struct lower_hull{
    point ini, fim;
    int id_ini, id_fim;

    lower_hull(point ini = point(LINF, LINF), point fim = point(-LINF, -LINF)) : ini(ini), fim(fim) {
        id_ini = id_fim = -1;
    }
};

int n, k[N], p[N], a[N], root;
pii ans;
```

```
vector<int> par_upd[N], adj[N];
set<int> paired;
vector <point> hull[N];
pair<point, int> end_hull[N];
lower_hull low[N];

struct edge{
    point ini, fim;
    edge(point ini = point(0,0), point fim = point(0,0)) : ini(ini), fim(fim) {}
};

bool operator < (const edge& a, const edge& b) {
    if (a.ini == b.ini) return direction(a.ini, a.fim, b.fim) < 0;
    if (a.ini.x < b.ini.x) return direction(a.ini, a.fim, b.ini) < 0;
    return direction(a.ini, b.fim, b.ini) < 0;
}

vector <pair<point, piii>> eve;
vector <pair<edge, pii>> edgs[N];
set <pair<edge, pii>> sweep;

void is_simple_polygon(){
    int cnt_edge = 0;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < k[i]; j++){
            point l = min(hull[i][j], hull[i][(j + 1)%k[i]]);
            point r = max(hull[i][j], hull[i][(j + 1)%k[i]]);
            //if(l.x != r.x){
                eve.pb({l, {0, {i, j}}});
                eve.pb({r, {1, {i, j}}});
            //}
            edgs[i].pb(make_pair(edge(l, r), make_pair(i, j)));
            cnt_edge++;
        }
    }
    sort(eve.begin(), eve.end());
    for(auto e : eve){
        if(!e.nd.st){
            sweep.insert(edgs[e.nd.nd.st][e.nd.nd.nd]);
        }
        else{
            auto below = sweep.upper_bound(edgs[e.nd.nd.st][e.nd.nd.nd]);
            auto cur = below, above = --cur;
            if(above != sweep.begin() and end_hull[e.nd.nd.st].nd == e.nd.nd.nd){
                --above;
                //if below lower hull then its father is the father from the polygon with edge above
                if(above->nd.nd < low[above->nd.st].id_fim){
                    a[e.nd.nd.st] = above->nd.st;
                    par_upd[above->nd.st].pb(e.nd.nd.st);
                }
                //if below upper hull then it is inside the polygon with edge above
                else{
                    p[e.nd.nd.st] = above->nd.st;
                    paired.insert(e.nd.nd.st);
                }
            }
            sweep.erase(cur);
        }
    }
}

int vis[N], h[N], anc[N][M];


//LCA
void dfs (int u) {
    vis[u] = 1;
    for (auto v : adj[u]) if (!vis[v]) {
        h[v] = h[u]+1;
        anc[v][0] = u;
        dfs(v);
    }
    ans = max(ans, make_pair(h[u], u));
}

void build () {
    anc[n][0] = n;
    dfs(n);
    for (int j = 1; j <= K; j++) for (int i = 0; i <= n; i++)
        anc[i][j] = anc[anc[i][j-1]][j-1];
}

int lca (int u, int v) {
    if (h[u] < h[v]) swap(u, v);
    for (int j = K; j >= 0; j--) if (h[anc[u][j]] >= h[v]) {
        u = anc[u][j];
    }
    if (u == v) return u;
    for (int j = K; j >= 0; j--) if (anc[u][j] != anc[v][j]) {
        u = anc[u][j];
        v = anc[v][j];
    }
    return anc[u][0];
}

int dist (int u, int v) {
    return h[u] + h[v] - 2*h[lca(u, v)];
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    memset(p, -1, sizeof(p));
    memset(anc, -1, sizeof(anc));
    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> k[i];
        for(int j = 0; j < k[i]; j++){
            point u;
            cin >> u.x >> u.y;
            hull[i].pb(u);
        }
        //sort lex hull to make most left point to have index 0
        sort_lex_hull(hull[i]);
        //if(!((hull[i][1].x < hull[i][2].x) or (hull[i][1].x == hull[i][2].x and hull[i][1].y > hull[i][2].y
              ))) swap(hull[i][1], hull[i][2]);

        low[i].ini = hull[i][0];
        low[i].id_ini = 0;
        end_hull[i] = {point(-LINF, -LINF), -1};
        //search for point that ends lower hull
        //end_hull[i] = point that will mark the end of the hull so we can process the polygon in the sweep
              line
        for(int j = 0; j < k[i]; j++){
            point u = hull[i][j];
            if((u.x > low[i].fim.x) or (u.x == low[i].fim.x and u.y < low[i].fim.y)) low[i].fim = u, low[i].
                  id_fim = j;
            end_hull[i] = max(end_hull[i], {u, j});
        }
    }
    is_simple_polygon();
    //for all nodes with parent add parent to the nodes that depend on them
    while(!paired.empty()){
        auto cur = paired.begin();
        for(auto v : par_upd[*cur]){
            p[v] = p[*cur];
            paired.insert(v);
        }
        par_upd[*cur].clear();
        paired.erase(cur);
    }
    //generate graph
    //n = virtual node;
    for(int i = 0; i < n; i++){
        if(p[i] != -1){
            adj[p[i]].pb(i);
        }
        else{
            adj[n].pb(i);
        }
    }
    //generate lca
    build();
    // ans = longest path + longest distance between node from longest path and any other from graph
    int d = 0;
    for(int i = 0; i <= n; i++){
        d = max(d, dist(ans.nd, i));
    }
    cout << ans.st + d << "\n";
    return 0;
}
```

1.19 Graham Scan (DP)

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
```

```
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

typedef int type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point rotate_ccw90(point p)   { return point(-p.y,p.x); }
point rotate_cw90(point p)    { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area_2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int n;
    cin >> n;
    string d;
    vector<point> pts(n + 1);
    vector<int> ans(n + 1);
    pair<point, int> mn = {{INF, INF}, INF};
    for(int i = 1; i <= n; i++){
        cin >> pts[i].x >> pts[i].y;
        ans[i] = i;
        mn = min(mn, {pts[i], i});
    }
    swap(pts[1], pts[mn.nd]);
    swap(ans[1], ans[mn.nd]);
    cin >> d;
    for(int i = 2; i < n; i++){
        int cur = 1, to = i;
        if(d[i - 2] == ’L’) cur = -1;
        for(int j = i + 1; j <= n; j++){
            if(pts[to].dir(pts[i - 1], pts[j]) == cur) to = j;
        }
        swap(ans[i], ans[to]);
        swap(pts[i], pts[to]);
    }
    for(int i = 1; i <= n; i++) cout << ans[i] << " ";
    cout << "\n";
    return 0;
}
```

1.20 Union of Convex Hulls (O(n * log(n)))

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 2e5+5;

typedef long long type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;
    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
              );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point rotate_ccw90(point p)   { return point(-p.y,p.x); }
point rotate_cw90(point p)    { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area_2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

int angle_less(const point& a1, const point& b1, const point& a2, const point& b2) {
    //angle between (a1 and b1) vs angle between (a2 and b2)
    //1  : bigger
    //-1 : smaller
    //0  : equal
    point p1(dot(   a1, b1), abs(cross(   a1, b1)));
    point p2(dot(   a2, b2), abs(cross(   a2, b2)));
    if(cross(p1, p2) < 0) return 1;
    if(cross(p1, p2) > 0) return -1;
    return 0;
}

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}
```

```
//Monotone chain O(nlog(n))
#define REMOVE_REDUNDANT
#ifdef REMOVE_REDUNDANT
bool between(const point &a, const point &b, const point &c) {
    return (abs(area_2(a,b,c)) == 0 && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void monotone_hull(vector<point> &pts) {
    sort(pts.begin(), pts.end());
    pts.erase(unique(pts.begin(), pts.end()), pts.end());
    vector<point> up, dn;
    for (int i = 0; i < pts.size(); i++) {
        while (up.size() > 1 && area_2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
        while (dn.size() > 1 && area_2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
        up.push_back(pts[i]);
        dn.push_back(pts[i]);
    }
    pts = dn;
    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

    #ifdef REMOVE_REDUNDANT
    if (pts.size() <= 2) return;
    dn.clear();
    dn.push_back(pts[0]);
    dn.push_back(pts[1]);
    for (int i = 2; i < pts.size(); i++) {
        if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
        dn.push_back(pts[i]);
    }
    if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
        dn[0] = dn.back();
        dn.pop_back();
    }
    pts = dn;
    #endif
}

int maximizeScalarProduct(vector<point> &hull, point vec) {
        // this code assumes that there are no 3 colinear points
        int ans = 0;
        int n = hull.size();
        if(n < 20) {
                for(int i = 0; i < n; i++) {
                        if(hull[i] * vec > hull[ans] * vec) {
                                ans = i;
                        }
                }
        } else {
                if(hull[1] * vec > hull[ans] * vec) {
                        ans = 1;
                }
                for(int rep = 0; rep < 2; rep++) {
                        int l = 2, r = n - 1;
                        while(l != r) {
                                int mid = (l + r + 1) / 2;
                                bool flag = hull[mid] * vec >= hull[mid-1] * vec;
                                if(rep == 0) { flag = flag && hull[mid] * vec >= hull[0] * vec; }
                                else { flag = flag || hull[mid-1] * vec < hull[0] * vec; }
                                if(flag) {
                                        l = mid;
                                } else {
                                        r = mid - 1;
                                }
                        }
                        if(hull[ans] * vec < hull[l] * vec) {
                                ans = l;
                        }
                }
        }
        return ans;
}

int n, m;
set<int> hull_sizes;
vector<point> hull[N];

void merge(vector<point>& cur, vector<point>& b){
    for(auto x : b) cur.push_back(x);
    monotone_hull(cur);
}

void add(point p){
    vector<point> cur{p};
    while(!hull_sizes.empty() and hull_sizes.count(cur.size())){
        int sz = cur.size();
        merge(cur, hull[sz]);
        hull_sizes.erase(sz);
    }
    hull[cur.size()] = cur;
    hull_sizes.insert(cur.size());
}

type calc(point p){
    type ans = -LINF;
    for(auto sz : hull_sizes){
        ans = max(ans, hull[sz][maximizeScalarProduct(hull[sz], p)] * p);
    }
    return ans;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n;
    point p;
    for(int i = 0; i < n; i++){
        cin >> p.x >> p.y;
        add(p);
    }
    cin >> m;
    for(int i = 0; i < m; i++){
        string s;
        cin >> s >> p.x >> p.y;
        if(s == "get") cout << calc(p) << "\n";
        else add(p);
    }
    return 0;
}
```

1.21 Upward and Downward edges

```
//line cutting a polygon
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e3+5;

typedef long double type;
//for big coordinates change to long long

bool ge(type x, type y) { return x + EPS > y; }
bool le(type x, type y) { return x - EPS < y; }
bool eq(type x, type y) { return ge(x, y) and le(x, y); }

struct point {
    type x, y;

    point() : x(0), y(0) {}
    point(type x, type y) : x(x), y(y) {}

    point operator -() { return point(-x, -y); }
    point operator +(point p) { return point(x + p.x, y + p.y); }
    point operator -(point p) { return point(x - p.x, y - p.y); }

    point operator *(type k) { return point(k*x, k*y); }
    point operator /(type k) { return point(x/k, y/k); }

    //inner product
    type operator *(point p) { return x*p.x + y*p.y; }
    //cross product
    type operator %(point p) { return x*p.y - y*p.x; }

    bool operator ==(const point &p) const{ return x == p.x and y == p.y; }
    bool operator !=(const point &p) const{ return x != p.x  or y != p.y; }
    bool operator <(const point &p) const { return (x < p.x) or (x == p.x and y < p.y); }

    // 0 => same direction
    // 1 => p is on the left
    //-1 => p is on the right
    int dir(point o, point p) {
        type x = (*this - o) % (p - o);
        return ge(x,0) - le(x,0);
    }

    bool on_seg(point p, point q) {
        if (this->dir(p, q)) return 0;
        return ge(x, min(p.x, q.x)) and le(x, max(p.x, q.x)) and ge(y, min(p.y, q.y)) and le(y, max(p.y, q.y)
             );
    }

    ld abs() { return sqrt(x*x + y*y); }
    type abs2() { return x*x + y*y; }
    ld dist(point q) { return (*this - q).abs(); }
    type dist2(point q) { return (*this - q).abs2(); }

    ld arg() { return atan2l(y, x); }

    // Project point on vector y
    point project(point y) { return y * ((*this * y) / (y * y)); }

    // Project point on line generated by points x and y
    point project(point x, point y) { return x + (*this - x).project(y-x); }

    ld dist_line(point x, point y) { return dist(project(x, y)); }

    ld dist_seg(point x, point y) {
        return project(x, y).on_seg(x, y) ? dist_line(x, y) :  min(dist(x), dist(y));
    }

    point rotate(ld sin, ld cos) { return point(cos*x - sin*y, sin*x + cos*y); }
    point rotate(ld a) { return rotate(sin(a), cos(a)); }

    // rotate around the argument of vector p
    point rotate(point p) { return rotate(p.x / p.abs(), p.y / p.abs()); }

};

int direction(point o, point p, point q) { return p.dir(o, q); }

point RotateCCW90(point p)   { return point(-p.y,p.x); }
point RotateCW90(point p)    { return point(p.y,-p.x); }

//for reading purposes avoid using * and % operators, use the functions below:
type dot(point p, point q)     { return p.x*q.x + p.y*q.y; }
type cross(point p, point q)   { return p.x*q.y - p.y*q.x; }

//double area
type area2(point a, point b, point c) { return cross(a,b) + cross(b,c) + cross(c,a); }

ostream &operator<<(ostream &os, const point &p) {
    os << "(" << p.x << "," << p.y << ")";
    return os;
}

bool LinesParallel(point a, point b, point c, point d) {
    return fabs(cross(b - a, d - c)) < EPS;
}

bool LinesCollinear(point a, point b, point c, point d) {
    // Degenerate case
    //if((a == c and b == d) || (a == d and b == c)) return true;
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

bool LineLineIntersect(point a, point b, point c, point d) {
    if(!LinesParallel(a, b, c, d)) return true;
    if(LinesCollinear(a, b, c, d)) return true;
    return false;
}

point lines_intersect(point p, point q, point a, point b) {
    point r = q - p, s = b - a, c(p%q, a%b);
    if (eq(r%s,0)) return point(LINF, LINF);
    return point(point(r.x, s.x) % c, point(r.y, s.y) % c) / (r%s);
}

bool SegmentLineIntersect(point a, point b, point c, point d){
    // Degenerate case
```

```cpp
    // if((a == c and b == d) || (a == d and b == c)) return true;
    if(!LineLineIntersect(a, b, c, d)) return false;
    point inters = lines_intersect(a, b, c, d);
    if(inters.on_seg(a, b)) return true;
    return false;
}

bool upward_edge(point a, point b, point c, point d){
    //Line: a - b
    //Edge: c - d
    //Edge who comes from bottom to top (or from right to left), but does not consider the final endpoint
    return (direction(a, b, c) < 1 and direction(a, b, d) == 1);
}

bool downward_edge(point a, point b, point c, point d){
    //Line: a - b
    //Edge: c - d
    //Edge who comes from top to bottom (or from left to right), but does not consider the initial endpoint
    return (direction(a, b, c) == 1 and direction(a, b, d) < 1);
}

int n, m;
vector <point> pts;
point lines[2][N];
vector<point> inters;

type calc(int i){
    type ans = 0;
    inters.clear();
    vector<pair<point, int>> sweep;
    //See for each edge if it intercepts:

    for(int j = 0; j < n; j++){
        //Check upward and downward for info
        //upward and downward disconsider "horizontal" edges
        if(upward_edge(lines[0][i], lines[1][i], pts[j], pts[(j + 1)%n]) || downward_edge(lines[0][i], lines
              [1][i], pts[j], pts[(j + 1)%n]))
            inters.push_back(lines_intersect(lines[0][i], lines[1][i], pts[j], pts[(j + 1)%n]));
        //if not upward or downward check if it is a collinear edge
        else if(LinesCollinear(lines[0][i], lines[1][i], pts[j], pts[(j + 1)%n])){
            point a = pts[j];
            point b = pts[(j + 1)%n];
            if(b < a) swap(a, b);
            sweep.push_back({a, -1});
            sweep.push_back({b, 1});
        }
    }
    //Add interceptions to the sweep:
    //even: line enters the polygon
    //odd:  line leaves the polygon
    sort(inters.begin(), inters.end());
    int mult = -1;
    for(int j = 0; j < inters.size(); j++){
        sweep.push_back({inters[j], mult});
        mult *= -1;
    }
    sort(sweep.begin(), sweep.end());
    int open = 0;
    point ini;
    for(int j = 0; j < sweep.size(); j++){
        pair <point, int> p = sweep[j];
        //-1: enters the polygon
        if(p.nd == -1){
            open++;
            //first interception
            if(open == 1) ini = p.st;
        }
        //1: leaves the polygon
        if(p.nd == 1) open--;
        if(open == 0) {
            //last interception, compute distance inside the polygon
            ans += sweep[j].st.dist(ini);
        }
    }
    return ans;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n >> m;
    pts.resize(n);
    for(int i = 0; i < n; i++){
        cin >> pts[i].x >> pts[i].y;
    }
    //REMOVE COLLINEAR POINTS: not necessary.
    // for(int i = 0; i < pts.size(); i++){
    //     if(direction(pts[i], pts[(i - 1 + (int)pts.size())%pts.size()], pts[(i + 1)%pts.size()]) == 0) db(
          i), pts.erase(pts.begin() + i), i--;
    // }
    for(int i = 0; i < m; i++){
        cin >> lines[0][i].x >> lines[0][i].y;
        cin >> lines[1][i].x >> lines[1][i].y;
    }
    for(int i = 0; i < m; i++){
        type ans = 0;
        ans = calc(i);
        cout << setprecision(15) << fixed << ans << "\n";
    }
    return 0;
}
```

1.22 Minimum Enclosing Circle

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

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

struct point {
        double x, y;
        point() { x = y = 0.0; }
        point(double _x, double _y) : x(_x), y(_y) {}
        point operator +(point other) const{
                return point(x + other.x, y + other.y);
        }
        point operator -(point other) const{
                return point(x - other.x, y - other.y);
        }
        point operator *(double k) const{
                return point(x*k, y*k);
        }
};

double dist(point p1, point p2) {
        return hypot(p1.x - p2.x, p1.y - p2.y);
}

double inner(point p1, point p2) {
        return p1.x*p2.x + p1.y*p2.y;
}

double cross(point p1, point p2) {
        return p1.x*p2.y - p1.y*p2.x;
}

point rotate(point p, double rad) {
        return point(p.x * cos(rad) - p.y * sin(rad),
        p.x * sin(rad) + p.y * cos(rad));
}

struct circle {
        point c;
        double r;
        circle() { c = point(); r = 0; }
        circle(point _c, double _r) : c(_c), r(_r) {}
        double area() { return acos(-1.0)*r*r; }
        double chord(double rad) { return  2*r*sin(rad/2.0); }
        double sector(double rad) { return 0.5*rad*area()/acos(-1.0); }
        bool intersects(circle other) {
                return dist(c, other.c) < r + other.r;
        }
        bool contains(point p) { return dist(c, p) <= r + EPS; }
        pair<point, point> getTangentPoint(point p) {
                double d1 = dist(p, c), theta = asin(r/d1);
                point p1 = rotate(c-p,-theta);
                point p2 = rotate(c-p,theta);
                p1 = p1*(sqrt(d1*d1-r*r)/d1)+p;
                p2 = p2*(sqrt(d1*d1-r*r)/d1)+p;
                return make_pair(p1,p2);
        }
};

circle circumcircle(point a, point b, point c) {
        circle ans;
        point u = point((b-a).y, -(b-a).x);
        point v = point((c-a).y, -(c-a).x);
        point n = (c-b)*0.5;
        double t = cross(u,n)/cross(v,u);
        ans.c = ((a+c)*0.5) + (v*t);
        ans.r = dist(ans.c, a);
        return ans;
}

int insideCircle(point p, circle c) {
        if (fabs(dist(p , c.c) - c.r)<EPS) return 1;
        else if (dist(p , c.c) < c.r) return 0;
        else return 2;
} //0 = inside/1 = border/2 = outside

circle minimumCircle(vector<point> p) {
        random_shuffle(p.begin(), p.end());
        circle C = circle(p[0], 0.0);
        for(int i = 0; i < (int)p.size(); i++) {
                if (C.contains(p[i])) continue;
                C = circle(p[i], 0.0);
                for(int j = 0; j < i; j++) {
                        if (C.contains(p[j])) continue;
                        C = circle((p[j] + p[i])*0.5, 0.5*dist(p[j], p[i]));
                        for(int k = 0; k < j; k++) {
                                if (C.contains(p[k])) continue;
                                C = circumcircle(p[j], p[i], p[k]);
                        }
                }
        }
        return C;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout << setprecision(10) << fixed;
    int n;
    cin >> n;
    vector <point> p(n);
    for(int i=0;i<n;i++){
        cin >> p[i].x >> p[i].y;
    }
    circle ans = minimumCircle(p);
    cout << ans.c.x << " " << ans.c.y << "\n" << ans.r << "\n";
    return 0;
}
```

2 DSU

2.1 Bosses

```
/*

There are n employees in a company, and in the current moment no one is a subordinate of any other one. That
      is, each employee is a boss of himself. We call a person a boss, if he is not a subordinate of anybody
      else.

You are to process two types of queries:

boss a becomes a subordinate of boss b (and no longer is a boss),
given an employee c, what is the number of his superiors we should pass to reach a boss?
In a query of the second type, if c is a boss, the answer is 0, otherwise it is some positive integer - the "
      depth" of the employee.

Write a program that processes the queries.

*/

#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 3e5+5;

int n, m, par[N], rk[N];

int find(int a){
    if(a == par[a]){
        rk[a] = 0;
        return a;
    }
    int up = find(par[a]);
    rk[a] += rk[par[a]];
    par[a] = up;
    return par[a] = up;
}

void unite(int a, int b){
    par[a] = b;
    rk[a]++;
}

int main(){
    scanf("%d %d", &n, &m);
    for(int i = 1; i <= n; i++) par[i] = i;
    for(int i = 0; i < m; i++){
        int op, a, b;
        scanf("%d", &op);
        if(op == 1){
            scanf("%d%d", &a, &b);
            unite(a, b);
        }
        if(op == 2){
            scanf("%d", &a);
            find(a);
            printf("%d\n", rk[a]);
        }
    }
    return 0;
}
```

2.2 DSU in Range

```
//dsu in range with color update


#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 2e5+5;

int n, q, par[N], sz[N], mn[N];

int find(int a){return par[a] == a ? a : par[a] = find(par[a]);}

void unite(int a, int b){
    if((a = find(a)) == (b = find(b))) return;
    if(sz[a] < sz[b]) swap(a, b);
    sz[a] += sz[b];
    par[b] = a;
    mn[a] = min(mn[b], mn[a]);
}

set<pii> comp;

void update(int l, int r){
    vector <pii> rem;
    int last = 0, add_l = l, add_r = r;
    auto it = comp.lower_bound({l, 0});
    if(it != comp.begin()) it--;
    for(; it != comp.end(); it++){
        int cur_l, cur_r;
        cur_l = (*it).st;
        cur_r = (*it).nd;
        if(cur_l > r) break;
        if(cur_r < l) continue;
        unite(mn[find(cur_l)], mn[find(l)]);
        if(cur_l  < l) add_l = cur_l;
        if(r < cur_r) add_r = cur_r;
        rem.pb(*it);
    }
    for(auto x : rem) comp.erase(x);
    comp.insert({add_l, add_r});
}

int main(){
    scanf(" %d%d", &n, &q);
    for(int i = 1; i <= n; i++){
        par[i] = mn[i] = i;
        sz[i] = 1;
        comp.insert({i,i});
    }
    for(int i = 0; i < q; i++){
        int t, x, y;
        scanf(" %d%d%d", &t, &x, &y);
        if(t == 1){
            unite(x, y);
        }
        if(t == 2){
            update(x, y);
        }
        if(t == 3){
            printf("%s\n", (find(x) == find(y) ? "YES" : "NO"));
        }
    }
    return 0;
}
```

2.3 Nearest available

```
/*

n persons are standing at positions 1 to n. You have to perform queries of two types:

"- x" - the person at position x leaves;
"? x" - find the nearest person to the right that is still standing.

*/

#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e6+5;

int par[N], sz[N], mx[N];

int find(int a){
    return (par[a] == a ? a : par[a] = find(par[a]));
}

void unite(int a, int b){
    if((a = find(a)) == (b = find(b))) return;
    if(sz[a] < sz[b]) swap(a, b);
    sz[a] += sz[b];
    par[b] = a;
    mx[a] = max(mx[b], mx[a]);
}

int main(){
    int n, m;
    scanf("%d %d", &n, &m);
    for(int i = 1; i <= n+1; i++) par[i] = mx[i] = i, sz[i] = 1;
    for(int i = 0; i < m; i++){
        char op;
        int x;
        scanf(" %c %d", &op, &x);
        if(op == '?'){
            int tp = mx[find(x)];
            printf("%d\n", (tp == n+1 ? -1 : tp));
        }
        else{
            unite(x, x+1);
        }
    }
    return 0;
}
```

2.4 Nearest available right (circular)

```
/*

There are n slots on a circular parking enumerated from 1 to n.
There are n cars that want to park in the natural order. i-th car wants to park at pi-th slot. If the car
      drives to a parking slot and her slot is occupied, it drives in a circular manner and parks on the
      first vacant slot.

*/

#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;
```

```
const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 6e5+5;

int par[N], sz[N], mx[N];

int find(int a){ return par[a] == a ? a : par[a] = find(par[a]);}

void unite(int a, int b){
    if((a = find(a)) == (b = find(b))) return;
    if(sz[a] < sz[b]) swap(a, b);
    sz[a] += sz[b];
    par[b] = a;
    mx[a] = max(mx[a], mx[b]);
}

int main(){
    int n;
    scanf("%d", &n);
    for(int i = 0; i < N; i++) par[i] = mx[i] = i, sz[i] = 1;
    for(int i = 0; i < n; i++){
        int x;
        scanf("%d", &x);
        x = mx[find(x-1)];
        printf("%d ", (x%n) + 1);
        if(x < n){
            unite(x, x+1);
            unite(x+n, x+n+1);
        }
        else{
            unite(x, x+1);
            unite(x-n, x-n+1);
        }
    }
    printf("\n");
    return 0;
}
```

3 Segment Tree

3.1 Element at least x (binary search)

```
//Find minimum index j such thad a[j] >= x

#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

int st[4*N], v[N];

void build (int p, int l, int r) {
    if (l == r) {
        st[p] = v[l];
        return;
    }
    build (2*p, l, (l+r)/2);
    build (2*p+1, (l+r)/2+1, r);
    st[p] = max(st[2*p], st[2*p+1]);
}
```

```
void update (int p, int l, int r, int x, int k) {
    if (x < l or r < x) return;
    if (l == r and l == x) {
        st[p] = k;
        return;
    }
    update (2*p, l, (l+r)/2, x, k);
    update (2*p+1, (l+r)/2+1, r, x, k);
    st[p] = max(st[2*p], st[2*p+1]);
}

int query (int p, int l, int r, int k) {
    if(st[p] < k) return INF;
    if(l == r) return l;
    int query_left = INF;
    query_left = query(2*p, l, (l+r)/2, k);
    if(query_left == INF) return query(2*p + 1, (l + r)/2 + 1, r, k);
    return query_left;
}

int main(){
    int n, m;
    scanf("%d %d", &n, &m);
    for(int i = 0; i < n; i++) scanf("%d", &v[i]);
    build(1, 0, n - 1);
    for(int i = 0; i < m; i++){
        int op, pos, val;
        scanf(" %d", &op);
        if(op == 1){
            scanf("%d %d", &pos, &val);
            update(1, 0, n-1, pos, val);
        }
        else{
            scanf("%d", &pos);
            printf("%d\n", query(1, 0, n - 1, pos));
        }
    }
    return 0;
}
```

3.2 Element at least x and j ¿ l(binary search)

```
//finding for the given x and l the minimum index j such that j>=l and a[j]>=x.
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

int st[4*N], v[N];

void build (int p, int l, int r) {
    if (l == r) {
        st[p] = v[l];
        return;
    }
    build (2*p, l, (l+r)/2);
    build (2*p+1, (l+r)/2+1, r);
    st[p] = max(st[2*p], st[2*p+1]);
}

void update (int p, int l, int r, int x, int k) {
    if (x < l or r < x) return;
    if (l == r and l == x) {
        st[p] = k;
        return;
    }
    update (2*p, l, (l+r)/2, x, k);
    update (2*p+1, (l+r)/2+1, r, x, k);
    st[p] = max(st[2*p], st[2*p+1]);
}

int query (int p, int l, int r, int k, int i) {
    if(r < i or st[p] < k) return INF;
    if(l == r){
        return l;
    }
    int query_left = INF, query_right = INF;
    query_left = query(2*p, l, (l+r)/2, k, i);
    // db(l _ r _ query_left);
    if(l < i or query_left == INF) query_right = query(2*p + 1, (l + r)/2 + 1, r, k, i);
    return min(query_left, query_right);
}

int main(){
    int n, m;
    scanf("%d %d", &n, &m);
    for(int i = 0; i < n; i++) scanf("%d", &v[i]);
    build(1, 0, n - 1);
    for(int i = 0; i < m; i++){
        int op, pos, val, l;
        scanf(" %d", &op);
        if(op == 1){
            scanf("%d %d", &pos, &val);
            update(1, 0, n-1, pos, val);
        }
        else{
            scanf("%d %d", &pos, &l);
            int ans = query(1, 0, n - 1, pos, l);
            printf("%d\n", (ans == INF) ? -1 : ans);
        }
    }
    return 0;
}
```

3.3 K-th one (binary search)

```
//K-TH ONE {0, 1}

#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

int st[4*N], v[N];

void build (int p, int l, int r) {
    if (l == r) {
        st[p] = v[l];
        return;
    }
    build (2*p, l, (l+r)/2);
    build (2*p+1, (l+r)/2+1, r);
    st[p] = st[2*p] + st[2*p+1];
}

void update (int p, int l, int r, int x) {
    if (x < l or r < x) return;
    if (l == r and l == x) {
        st[p] ^= 1;
        return;
    }
    update (2*p, l, (l+r)/2, x);
    update (2*p+1, (l+r)/2+1, r, x);
    st[p] = st[2*p] + st[2*p+1];
}

int query (int p, int l, int r, int k) {
    if(l == r){
        return l;
    }
    if(st[2*p] >= k){
        return query(2*p, l, (l+r)/2, k);
    }
    return query(2*p + 1, (l + r)/2 + 1, r, k - st[2*p]);
}

int main(){
    int n, m;
    scanf("%d %d", &n, &m);
    for(int i = 0; i < n; i++) scanf("%d", &v[i]);
    build(1, 0, n - 1);
    for(int i = 0; i < m; i++){
        int op, val;
        scanf("%d %d", &op, &val);
        if(op == 1){
            update(1, 0, n-1, val);
        }
        else{
            printf("%d\n", query(1, 0, n - 1, val + 1));
        }
    }
    return 0;
}
```

3.4 Intersecting segments

```
/*
Given an array of 2n numbers, each number from 1 to n in it occurs exactly twice.
We say that the segment y intersects the segment x if exactly one occurrence of the number y is between the
      occurrences of the number x.
Find for each segment i how many segments there are that intersect with it.
*/

#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 2e5+5;


int st[4*N], v[N], ans[N], q[N];

int query (int p, int l, int r, int i, int j) {
    if (r < i or j < l) return 0;
    if (i <= l and r <= j) return st[p];
    int x = query(2*p, l, (l+r)/2, i, j);
    int y = query(2*p+1, (l+r)/2+1, r, i, j);
    return x + y;
}
```

```
void update (int p, int l, int r, int x, int v) {
    if (x < l or r < x) return;
    if (l == r and l == x) {
        st[p] = v;
        return;
    }
    update (2*p, l, (l+r)/2, x, v);
    update (2*p+1, (l+r)/2+1, r, x, v);
    st[p] = st[2*p] + st[2*p+1];
}

int main(){
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= 2*n; i++){
        scanf("%d", &q[i]);
        if(v[q[i]]){
            update(1, 1, 2*n, v[q[i]], 0);
            ans[q[i]] += query(1, 1, 2*n, v[q[i]], i);
            v[q[i]] = 0;
        }
        else{
            v[q[i]] = i;
            update(1, 1, 2*n, v[q[i]], 1);
        }
    }
    for(int i = 2*n; i >= 1; i--){
        if(v[q[i]]){
            update(1, 1, 2*n, v[q[i]], 0);
            ans[q[i]] += query(1, 1, 2*n, i, v[q[i]]);
        }
        else{
            v[q[i]] = i;
            update(1, 1, 2*n, v[q[i]], 1);
        }
    }
    for(int i = 1; i <= n; i++) printf("%d ", ans[i]);
    return 0;
}
```

3.5 Inversion count

```
/*

Given a permutation pi of n elements, find for each i the number of j such that j<i and pj>pi.

*/
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

int st[4*N];

int query (int p, int l, int r, int i, int j) {
    if (r < i or j < l) return 0;
    if (i <= l and r <= j) return st[p];
    int x = query(2*p, l, (l+r)/2, i, j);
    int y = query(2*p+1, (l+r)/2+1, r, i, j);
    return x + y;
}

void update (int p, int l, int r, int x, int v) {
    if (x < l or r < x) return;
    if (l == r and l == x) {
        st[p] = v;
        return;
    }
    update (2*p, l, (l+r)/2, x, v);
    update (2*p+1, (l+r)/2+1, r, x, v);
    st[p] = st[2*p] + st[2*p+1];
}

int main(){
    int n;
    scanf("%d", &n);
    for(int i = 0; i < n; i++){
        int p;
        scanf("%d", &p);
        printf("%d ", query(1, 0, n, p, n));
        update(1, 0, n, p, 1);
    }
    printf("\n");
    return 0;
}
```

3.6 Recover answer from inversion count

```
/*

This problem is the reversed version of the previous one.
There was a permutation pi of n elements, for each i we wrote down the number ai, the number of j such that j<i and pj>pi.
Restore the original permutation for the given ai.

*/

#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

int st[4*N], v[N];

void build(int p, int l, int r){
    if(l == r){
        st[p] = 1;
        return;
    }
    int mid = (l+r)/2;
    build(2*p, l, mid);
    build(2*p + 1, mid + 1, r);
    st[p] = st[2*p] + st[2*p + 1];
}

int query (int p, int l, int r, int k) {
    if(st[p] < k) return -1;
    if(l == r) return l;
    int mid = (l + r)/2, query_right;
    query_right = query(2*p + 1, mid + 1, r, k);
    if(query_right != -1) return query_right;
    return query(2*p, l, mid, k - st[2*p + 1]);
}

void update (int p, int l, int r, int x, int k) {
    if (x < l or r < x) return;
    if (l == r and l == x) {
        st[p] = k;
        return;
    }
    update (2*p, l, (l+r)/2, x, k);
    update (2*p+1, (l+r)/2+1, r, x, k);
    st[p] = st[2*p] + st[2*p+1];
}

int main(){
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; i++){
        scanf("%d", &v[i]);
    }
    build(1, 1, n);
    for(int i = n; i >= 1; i--){
        v[i] = query(1, 1, n, v[i] + 1);
        update(1, 1, n, v[i], 0);
    }
    for(int i = 1; i <= n; i++) printf("%d ", v[i]);
    printf("\n");
    return 0;
}
```

4 Dynamic Segment Tree

4.1 BGSHOOT (Lazy)

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5, M = 1e9;

vector<int> e, d, mx, lazy;
//begin creating node 0, then start your segment tree creating node 1
int create(){
    mx.push_back(0);
    lazy.push_back(0);
    e.push_back(0);
    d.push_back(0);
    return mx.size() - 1;
}

void push(int pos, int ini, int fim){
    // if(pos == 0) return;
    if (lazy[pos]) {
        mx[pos] += lazy[pos];
        // RMQ (max/min)   -> update: = lazy[p],         incr: += lazy[p]
        // RSQ (sum)       -> update: = (r-l+1)*lazy[p], incr: += (r-l+1)*lazy[p]
        // Count lights on -> flip:   = (r-l+1)-st[p];
        if (ini != fim) {
            if(e[pos] == 0){
                int aux = create();
                e[pos] = aux;
            }
            if(d[pos] == 0){
                int aux = create();
                d[pos] = aux;
            }
            lazy[e[pos]] += lazy[pos];
            lazy[d[pos]] += lazy[pos];
            // update:    lazy[2*p] = lazy[p],  lazy[2*p+1] = lazy[p];
            // increment: lazy[2*p] += lazy[p], lazy[2*p+1] += lazy[p];
            // flip:      lazy[2*p] ^= 1,       lazy[2*p+1] ^= 1;
        }
        lazy[pos] = 0;
    }
}

void update(int pos, int ini, int fim, int p, int q, int val){
    // if(pos == 0) return;

    push(pos, ini, fim);

    if(q < ini || p > fim) return;

    if(p <= ini and fim <= q){
        lazy[pos] += val;
        // update:    lazy[p] = k;
        // increment: lazy[p] += k;
        // flip:      lazy[p] = 1;
        push(pos, ini, fim);
        return;
    }

    int m = (ini + fim) >> 1;
    if(e[pos] == 0){
        int aux = create();
        e[pos] = aux;
    }
    update(e[pos], ini, m, p, q, val);
    if(d[pos] == 0){
        int aux = create();
        d[pos] = aux;
    }
    update(d[pos], m + 1, fim, p, q, val);
    mx[pos] = max(mx[e[pos]], mx[d[pos]]);
}

int query(int pos, int ini, int fim, int p, int q){
    // if(pos == 0) return 0;

    push(pos, ini, fim);

    if(q < ini || p > fim) return 0;

    if(p <= ini and fim <= q) return mx[pos];

    int m = (ini + fim) >> 1;
    return max(query(e[pos], ini, m, p, q) , query(d[pos], m + 1, fim, p, q));
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    //init segtree
    create(), create();
    int n, q;
    cin >> n;
    for(int i = 0; i < n; i++){
        int x, y;
        cin >> x >> y;
        update(1, 1, M, x, y, 1);
    }
    cin >> q;
    for(int i = 0; i < q; i++){
        int l, r;
        cin >> l >> r;
        cout << query(1, 1, M, l, r) << "\n";
    }
    return 0;
}
```

4.2 Dynamic Segment Tree

```
#include <bits/stdc++.h>

/* tested:
    https://www.spoj.com/problems/ORDERSET/
    https://www.eolymp.com/en/contests/8463/problems/72212
    https://codeforces.com/contest/474/problem/E
    https://codeforces.com/problemset/problem/960/F
   ref:
    https://maratona.ic.unicamp.br/MaratonaVerao2022/slides/AulaSummer-SegmentTree-Aula2.pdf
*/
```

```
vector<int> e, d, mn;
//begin creating node 0, then start your segment tree creating node 1
int create(){
    mn.push_back(0);
    e.push_back(0);
    d.push_back(0);
    return mn.size() - 1;
}

void update(int pos, int ini, int fim, int id, int val){
    if(id < ini || id > fim) return;

    if(ini == fim){
        mn[pos] = val;
        return;
    }

    int m = (ini + fim) >> 1;
    if(id <= m){
        if(e[pos] == 0){
            int aux = create();
            e[pos] = aux;
        }
        update(e[pos], ini, m, id, val);
    }
    else{
        if(d[pos] == 0){
            int aux = create();
            d[pos] = aux;
        }
        update(d[pos], m + 1, fim, id, val);
    }

    mn[pos] = min(mn[e[pos]], mn[d[pos]]);
}

int query(int pos, int ini, int fim, int p, int q){
    if(q < ini || p > fim) return INT_MAX;

    if(pos == 0) return 0;

    if(p <= ini and fim <= q) return mn[pos];

    int m = (ini + fim) >> 1;
    return min(query(e[pos], ini, m, p, q), query(d[pos], m + 1, fim, p, q));
}
```

4.3 Lazy Dynamic Segment Tree

```
#include <bits/stdc++.h>

/* tested:
    https://www.spoj.com/problems/BGSHOOT/
   ref:
    https://maratona.ic.unicamp.br/MaratonaVerao2022/slides/AulaSummer-SegmentTree-Aula2.pdf
*/
vector<int> e, d, mx, lazy;
//begin creating node 0, then start your segment tree creating node 1
int create(){
    mx.push_back(0);
    lazy.push_back(0);
    e.push_back(0);
    d.push_back(0);
    return mx.size() - 1;
}

void push(int pos, int ini, int fim){
    if(pos == 0) return;
    if (lazy[pos]) {
        mx[pos] += lazy[pos];
        // RMQ (max/min)    -> update: = lazy[p],         incr: += lazy[p]
        // RSQ (sum)        -> update: = (r-l+1)*lazy[p], incr: += (r-l+1)*lazy[p]
        // Count lights on -> flip:   = (r-l+1)-st[p];
        if (ini != fim) {
            if(e[pos] == 0){
                int aux = create();
                e[pos] = aux;
            }
            if(d[pos] == 0){
                int aux = create();
                d[pos] = aux;
            }
            lazy[e[pos]] += lazy[pos];
            lazy[d[pos]] += lazy[pos];
            // update:    lazy[2*p] = lazy[p],  lazy[2*p+1] = lazy[p];
            // increment: lazy[2*p] += lazy[p], lazy[2*p+1] += lazy[p];
            // flip:      lazy[2*p] ^= 1,       lazy[2*p+1] ^= 1;
        }
        lazy[pos] = 0;
    }
}

void update(int pos, int ini, int fim, int p, int q, int val){
    if(pos == 0) return;

    push(pos, ini, fim);

    if(q < ini || p > fim) return;

    if(p <= ini and fim <= q){
        lazy[pos] += val;
        // update:    lazy[p] = k;
        // increment: lazy[p] += k;
        // flip:      lazy[p] = 1;
        push(pos, ini, fim);
        return;
    }

    int m = (ini + fim) >> 1;
    if(e[pos] == 0){
        int aux = create();
        e[pos] = aux;
    }
    update(e[pos], ini, m, p, q, val);
    if(d[pos] == 0){
        int aux = create();
        d[pos] = aux;
    }
    update(d[pos], m + 1, fim, p, q, val);
    mx[pos] = max(mx[e[pos]], mx[d[pos]]);
}

int query(int pos, int ini, int fim, int p, int q){
    if(pos == 0) return 0;

    push(pos, ini, fim);

    if(q < ini || p > fim) return 0;

    if(p <= ini and fim <= q) return mx[pos];

    int m = (ini + fim) >> 1;
    return max(query(e[pos], ini, m, p, q) , query(d[pos], m + 1, fim, p, q));
}
```

4.4 Orderset

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5, M = 1e9;

#include <bits/stdc++.h>

vector<int> e, d, sum, mn;
//begin creating node 0, then start your segment tree creating node 1
int create(){
    sum.push_back(0);
    mn.push_back(INF);
    e.push_back(0);
    d.push_back(0);
    return sum.size() - 1;
}

void update(int pos, int ini, int fim, int id, int val){
    if(id < ini || id > fim) return;

    if(ini == fim){
        sum[pos] = val;
        mn[pos] = ini;
        return;
    }

    int m = (ini + fim) >> 1;
    if(id <= m){
        if(e[pos] == 0){
            int aux = create();
            e[pos] = aux;
        }
        update(e[pos], ini, m, id, val);
    }
    else{
        if(d[pos] == 0){
            int aux = create();
            d[pos] = aux;
        }
        update(d[pos], m + 1, fim, id, val);
    }

    sum[pos] = sum[e[pos]] + sum[d[pos]];
    mn[pos] = min(mn[e[pos]], mn[d[pos]]);
}

int k_query(int pos, int ini, int fim, int k){
    //db(pos _ ini _ fim);
    //if(pos == 0) return 0;

    if(ini == fim) return ini;

    int m = (ini + fim) >> 1;

    if(sum[e[pos]] >= k)
        return k_query(e[pos], ini, m, k);

    return k_query(d[pos], m + 1, fim, k - sum[e[pos]]);
}

int c_query(int pos, int ini, int fim, int k){
    if(pos == 0) return 0;

    if(ini == fim){
        if(k == ini) return 0;
        return sum[pos];
    }

    int m = (ini + fim) >> 1;

    if(mn[d[pos]] <= k){
        return sum[e[pos]] + c_query(d[pos], m + 1, fim, k);
    }
    return c_query(e[pos], ini, m, k);
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    //init segtree
    create(), create();
    int q;
    cin >> q;
    for(int i = 0; i < q; i++){
        char c;
        int x;
        cin >> c >> x;
        if(c == 'I'){
            update(1, -M, M, x, 1);
            //db(sum[1]);
        }
        if(c == 'D'){
            update(1, -M, M, x, 0);
            //db(sum[1]);
        }
        if(c == 'C'){
            cout << c_query(1, -M, M, x) << "\n";
        }
        if(c == 'K'){
            if(sum[1] < x) cout << "invalid\n";
            else cout << k_query(1, -M, M, x) << "\n";
        }
    }
    return 0;
}
```

4.5 Pathwalks

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

#include <bits/stdc++.h>

vector<int> e, d, mx;
//begin creating node 0, then start your segment tree creating node 1
int create(){
    mx.push_back(0);
    e.push_back(0);
    d.push_back(0);
    return mx.size() - 1;
}

void update(int pos, int ini, int fim, int id, int val){
    if(id < ini || id > fim) return;

    if(ini == fim){
        mx[pos] = max(val, mx[pos]);
        return;
    }

    int m = (ini + fim) >> 1;
    if(id <= m){
        if(e[pos] == 0){
            int aux = create();
            e[pos] = aux;
        }
        update(e[pos], ini, m, id, val);
    }
    else{
        if(d[pos] == 0){
            int aux = create();
            d[pos] = aux;
        }
        update(d[pos], m + 1, fim, id, val);
    }

    mx[pos] = max(mx[e[pos]], mx[d[pos]]);
}

int query(int pos, int ini, int fim, int l, int r){
    if(r < ini or l > fim) return 0;

    if(pos == 0) return 0;

    if(l <= ini and r >= fim) return mx[pos];

    int m = (ini + fim) >> 1;
    return max(query(e[pos], ini, m, l, r), query(d[pos], m + 1, fim, l, r));
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    //init segtree
    int n, m;
    cin >> n >> m;
    for(int i = 0; i <= n; i++) create();
    for(int i = 0; i < m; i++){
        int u, v, p;
        cin >> u >> v >> p;
        int cur = query(u, 0, N, 0, p - 1);
        //db(cur);
        update(v, 0, N, p, cur + 1);
    }
    int ans = -INF;
    for(int i = 1; i <= n; i++) ans = max(ans, mx[i]);
    cout << ans << "\n";
    return 0;
}
```

4.6 Range Sum Query

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

#include <bits/stdc++.h>

vector<ll> e, d, sum;
//begin creating node 0, then start your segment tree creating node 1
int create(){
    sum.push_back(0);
    e.push_back(0);
    d.push_back(0);
    return sum.size() - 1;
}

void update(int pos, int ini, int fim, int id, int val){
    if(id < ini || id > fim) return;

    if(ini == fim){
        sum[pos] = val;
        return;
    }

    int m = (ini + fim) >> 1;
    if(id <= m){
        if(e[pos] == 0){
            int aux = create();
            e[pos] = aux;
        }
        update(e[pos], ini, m, id, val);
    }
    else{
        if(d[pos] == 0){
            int aux = create();
            d[pos] = aux;
        }
        update(d[pos], m + 1, fim, id, val);
    }

    sum[pos] = sum[e[pos]] + sum[d[pos]];
}
```

```
ll query(int pos, int ini, int fim, int l, int r){
    if(r < ini or l > fim) return 0;

    if(pos == 0) return 0;

    if(l <= ini and r >= fim) return sum[pos];

    int m = (ini + fim) >> 1;
    return query(e[pos], ini, m, l, r) + query(d[pos], m + 1, fim, l, r);
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    //init segtree
    create(), create();
    int n, m;
    cin >> n >> m;
    ll P = 91, Q = 47;
    for(int i = 0; i < m; i++){
        char c;
        ll a, b;
        cin >> c >> a >> b;
        if(c == ’!’){
            update(1, 0, n, (a + P) % n, (b + Q) % MOD);
        }
        else{
            int l = (a + P) % n, r = (b + Q) % n;
            if(l > r) swap(l, r);
            ll z = query(1, 0, n, l, r);
            cout << z << "\n";
            P = (((P * 31)%MOD + z) % MOD);
            Q = (((Q * 29)%MOD + z) % MOD);
        }
    }
    return 0;
}
```

5 Persistent Segment Tree

5.1 Persistent Segment Tree

```
#include <bits/stdc++.h>

/* tested:
*/

vector<int> e, d, sum;
//begin creating node 0, then start your segment tree creating node 1
int create(){
    sum.push_back(0);
    e.push_back(0);
    d.push_back(0);
    return sum.size() - 1;
}

int update(int pos, int ini, int fim, int id, int val){
    int novo = create();

    sum[novo] = sum[pos];
    e[novo] = e[pos];
    d[novo] = d[pos];

    if(ini == fim){
        sum[pos] = val;
        return novo;
    }

    int m = (ini + fim) >> 1;
    if(id <= m){
        int aux = update(e[pos], ini, m, id, val);
        e[pos] = aux;
    }
    else{
        int aux = update(d[pos], m + 1, fim, id, val);
        d[pos] = aux;
    }

    sum[pos] = sum[e[pos]] + sum[d[pos]];
}
```

```
int query(int pos, int ini, int fim, int p, int q){
    if(q < ini || p > fim) return INT_MAX;

    if(pos == 0) return 0;

    if(p <= ini and fim <= q) return sum[pos];

    int m = (ini + fim) >> 1;
    return query(e[pos], ini, m, p, q) + query(d[pos], m + 1, fim, p, q);
}
```

6 DP Optimization

6.1 Divide And Conquer

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define mp make_pair
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

int n;

// cost function
ll C(ll i, ll j) {return 0;}
//dp vectors
vector <ll> dp_bef(n), dp_cur(n);

//iterate compute over k, in the end, shift cur to bef and clear cur

void compute(int l, int r, int optl, int optr){
    // stop condition
    if(l > r) return;
    int mid = (l+r)/2;
    //best : cost, pos
    pair<ll,ll> best = {INF,-1};

    //searchs best, lower bound to right, upper bound to left
    for(int k = optl; k <= min(mid, optr); k++){
        best = min(best, {dp_bef[k] + C(k,mid), k});
    }
    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid-1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    return 0;
}
```

6.2 CHT Example

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define mp make_pair
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e6+5;

typedef long long type;
struct line { type b, m; };

int nh, pos;
line hull[N];

bool check(line s, line t, line u) {
    return (t.b - s.b) / ld(s.m - t.m) < (u.b - s.b) / ld(s.m - u.m);
}

void update(line s) {
  while (nh >= 2 and !check(hull[nh-2], hull[nh-1], s)) nh--;
  pos = min(pos, nh);
  hull[nh++] = s;
}

type eval(int id, type x) { return hull[id].b + hull[id].m * x; }

type query(type x) {
  while (pos+1 < nh and eval(pos, x) < eval(pos+1, x)) pos++;
  return eval(pos, x);
}

struct rect{
    type x = 0,y = 0,a = 0;
};

bool cmp(rect a, rect b){
    if(a.x == b.x) return a.y < b.y;
    return a.x < b.x;
}

type n, dp[N];
vector <rect> r;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n;
    for(int i=0;i<n;i++){
        rect aux;
        r.pb(aux);
        cin >> r[i].x >> r[i].y >> r[i].a;
    }
    ll ans = -LINF;
    sort(r.begin(),r.end(),cmp);
    for(int i=0;i<n;i++){
        ll aux = 0;
        if(i) aux = max(query(-r[i].y),aux);
        dp[i] = (r[i].x*r[i].y - r[i].a) + aux;
        ans = max(ans,dp[i]);
        update({dp[i],r[i].x});
    }
    cout << ans << "\n";
    return 0;
}
```

6.3 Garçom (no opt)

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define mp make_pair
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 405;

ll n, k, q[N], qx[N], dp[N][N];

ll C(ll j, ll i){
    ll ans = 0;
    ll m = (i+j)/2;
    ans += (qx[m] - qx[j]) - j*(q[m] - q[j]) - (qx[i-1] - qx[m]) + i*(q[i-1] - q[m]);
    return ans;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n >> k;
    ll ans = LINF;
    for(ll i=1;i<=n;i++){
        ll x;
        cin >> x;
        q[i] = q[i-1] + x;
        qx[i] = qx[i-1] + x*i;
    }
    for(ll l=1;l<=k;l++){
        for(ll i=1;i<=n;i++){
            ans = LINF;
            if(l == 1){
                dp[i][1] = i*q[i-1] - qx[i-1];
                continue;
            }
            if(i < l){
                dp[i][k] = LINF;
                continue;
            }
            for(ll j=1;j<i;j++){
                ans = min(ans, dp[j][k-1] + C(j,i));
            }
            dp[i][k] = ans;
        }
    }
    ans = LINF;
    for(ll i=1;i<=n;i++){
        ll rest = (qx[n] - qx[i]) - i*(q[n] - q[i]);
        //db(rest);
        ans = min(ans,dp[i][k] + rest);
    }
    cout << ans << "\n";
    return 0;
}
```

6.4 Garçom (Divide and Conquer)

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define mp make_pair
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 405;

ll n, k, q[N], qx[N], dp[N][N];

ll C(ll j, ll i){
    ll ans = 0;
    ll m = (i+j)/2;
    ans += (qx[m] - qx[j]) - j*(q[m] - q[j]) - (qx[i-1] - qx[m]) + i*(q[i-1] - q[m]);
    return ans;
}

void compute(ll l, ll r, ll k, ll optl, ll optr){
    // stop condition
    if(l > r) return;
    ll mid = (l+r)/2;
    //best : cost, pos
    pair<ll,ll> best = {LINF,-1};

    //searchs best, lower bound to right, upper bound to left
    for(ll i = optl; i <= min(mid, optr); i++){
        best = min(best, {dp[i][k-1] + C(i,mid), i});
    }
    dp[mid][k] = best.first;
    ll opt = best.second;

    compute(l, mid-1, k, optl, opt);
    compute(mid + 1, r, k, opt, optr);
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n >> k;
    ll ans = LINF;
    for(ll i=1;i<=n;i++){
        ll x;
        cin >> x;
        q[i] = q[i-1] + x;
        qx[i] = qx[i-1] + x*i;
    }
    for(ll l = 1; l<=k; l++){
        if(l == 1){
            for(ll i=1;i<=n;i++){
                dp[i][1] = i*q[i-1] - qx[i-1];
            }
            continue;
        }
        compute(1,n,l,1,n);
    }
    ans = LINF;
    for(ll i=1;i<=n;i++){
        ll rest = (qx[n] - qx[i]) - i*(q[n] - q[i]);
        ans = min(ans,dp[i][k] + rest);
    }
    cout << ans << "\n";
    return 0;
}
```

6.5 Garçom (Knuth)

```
#include <bits/stdc++.h>
```

```
using namespace std;

#define st first
#define nd second
#define mp make_pair
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 405;

ll n, k, q[N], qx[N], dp[N][N], L[N][N];

ll C(ll j, ll i){
    ll ans = 0;
    ll m = (i+j)/2;
    ans += (qx[m] - qx[j]) - j*(q[m] - q[j]) - (qx[i-1] - qx[m]) + i*(q[i-1] - q[m]);
    return ans;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n >> k;
    ll ans = LINF;
    for(ll i=1;i<=n;i++){
        ll x;
        cin >> x;
        q[i] = q[i-1] + x;
        qx[i] = qx[i-1] + x*i;
    }
    //Knuth
    for(ll i=1;i<=n;i++){
        dp[i][1] = i*q[i-1] - qx[i-1];
        L[n+1][i] = n;
        //L[1][i] = 1;
    }
    for(ll l = 2; l<=k; l++){
        for(ll i=n; i>=1; i--){
            pair<ll,ll> best = {LINF, n};
            for(ll j = L[i][l-1]; j <= L[i+1][l]; j++){
                best = min(best, {dp[j][l-1] + C(j, i), j});
            }
            dp[i][l] = best.first;
            L[i][l] = best.second;
        }
    }
    ans = LINF;
    for(ll i=1;i<=n;i++){
        ll rest = (qx[n] - qx[i]) - i*(q[n] - q[i]);
        ans = min(ans,dp[i][k] + rest);
    }
    cout << ans << "\n";
    return 0;
}
```

6.6 Internet Trouble(Knuth)

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define mp make_pair
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<
```

```
typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 6005;

ll n, b, c, q[N], qx[N], dp_cur[N], dp_bef[N], L_cur[N], L_bef[N];

ll C(ll j, ll i){
    ll ans = 0;
    ll m = (i+j)/2;
    ans += c*((qx[m] - qx[j]) - j*(q[m] - q[j]) - (qx[i-1] - qx[m]) + i*(q[i-1] - q[m]));
    return ans;
}

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cin >> n >> b >> c;
    ll ans = LINF;
    for(ll i=1;i<=n;i++){
        ll x;
        cin >> x;
        q[i] = q[i-1] + x;
        qx[i] = qx[i-1] + x*i;
    }
    for(ll i=1;i<=n;i++){
        dp_cur[i] = c*(i*q[i-1] - qx[i-1]);
        L_cur[n+1] = L_bef[n+1] = n;
        L_cur[i] = 1;
        ans = LINF;
    }
    for(ll i=1;i<=n;i++){
        ll rest = c*((qx[n] - qx[i]) - i*(q[n] - q[i]));
        ans = min(ans,dp_cur[i] + rest + b);
        dp_bef[i] = dp_cur[i];
        L_bef[i] = L_cur[i];
    }
    cout << ans << " ";

    for(ll l = 2; l<=n; l++){
        for(ll i=n; i>=1; i--){
            pair<ll,ll> best = {LINF, n};
            for(ll j = L_bef[i]; j <= L_cur[i+1] and j < i; j++){
                best = min(best, {dp_bef[j] + C(j, i), j});
            }
            dp_cur[i] = best.first;
            L_cur[i] = best.second;
        }
        ans = LINF;
        for(ll i=1;i<=n;i++){
            ll rest = c*((qx[n] - qx[i]) - i*(q[n] - q[i]));
            ans = min(ans,dp_cur[i] + rest + l*b);
            dp_bef[i] = dp_cur[i];
            L_bef[i] = L_cur[i];
        }
        cout << ans << (l==n ? "\n" : " ");
    }
    return 0;
}
```

6.7 Knuth Optimization

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define mp make_pair
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<
```

```
typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    return 0;
}
```

7 Stair Nim

7.1 Move coins (stair nim variation)

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

int n, q, ans, c[N], id[N], mx[N], vis[N], pxor[N], par[N];
vi adj[N];

int dfs(int s, int cnt, int p){
    vis[s] = 1;
    mx[cnt] = cnt;
    id[s] = cnt;
    par[cnt] = p;
    pxor[cnt] = pxor[cnt-1]^c[s];
    if(p) ans ^= c[s];
    for(auto v: adj[s]){
        if(!vis[v]){
            mx[cnt] = dfs(v,mx[cnt]+1,p^1);
        }
    }
    return mx[cnt];
}

int main(){
    // ios_base::sync_with_stdio(false);
    // cin.tie(NULL);
    int n;
    scanf("%d",&n);
    for(int i=1;i<=n;i++) scanf("%d",&c[i]);
    for(int i=1;i<n;i++){
        int u,v;
        scanf("%d %d",&u, &v);
        adj[u].pb(v);
        adj[v].pb(u);
    }
    dfs(1,1,0);
    scanf("%d",&q);
    for(int i=0;i<q;i++){
        int partans = ans;
        int u,v;
        scanf("%d%d",&u,&v);
        if(id[v] >= id[u] and id[v] <= mx[id[u]]) printf("INVALID\n");
        else{
            if(par[id[u]] == par[id[v]])  partans = partans^pxor[mx[id[u]]]^pxor[id[u]-1];
            if(partans) printf("YES\n");
            else printf("NO\n");
        }
    }
    return 0;
}
```

7.2 Classic Stair Nim

```
#include <bits/stdc++.h>

using namespace std;

#define st first
#define nd second
#define pb push_back
#define cl(x,v) memset((x), (v), sizeof(x))
#define db(x) cerr << #x << " == " << x << endl
#define dbs(x) cerr << x << endl
#define _ << ", " <<

typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<int, pii> piii;
typedef pair<ll,ll> pll;
typedef pair<ll, pll> plll;
typedef vector<int> vi;
typedef vector <vi> vii;

const ld EPS = 1e-9, PI = acos(-1.);
const ll LINF = 0x3f3f3f3f3f3f3f3f;
const int INF = 0x3f3f3f3f, MOD = 1e9+7;
const int N = 1e5+5;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int t;
    cin >> t;
    while(t--){
        int n;
        cin >> n;
        ll ans;
        for(int i=0;i<n;i++){
            ll p;
            cin >> p;
            if(i==1) ans = p;
            else if(i%2) ans^=p;
        }
        cout << (ans ? "first" : "second") << "\n";
    }
    return 0;
}
```