Algorithm Library

palayutm

September 25, 2018

Contents

1 计算几何

1.1 二维基础

```
const double INF = 1e60;
const double eps = 1e-8;
const double pi = acos(-1);

int sgn(double x) { return x < -eps ? -1 : x > eps; }
double Sqr(double x) { return x * x; }
double Sqrt(double x) { return x >= 0 ? std::sqrt(x) : 0; }

struct Vec {
    double x, y;

    Vec(double _x = 0, double _y = 0): x(_x), y(_y) {}

    Vec operator + (const Vec &oth) const { return Vec(x + oth.x, y + oth.y); }
    Vec operator - (const Vec &oth) const { return Vec(x - oth.x, y - oth.y); }
    Vec operator * (double t) const { return Vec(x * t, y * t); }
    Vec operator / (double t) const { return Vec(x / t, y / t); }

    double len2() const { return Sqr(x) + Sqr(y); }
    double len() const { return Sqrt(len2()); }

    Vec norm() const { return Vec(x / len(), y / len()); }
    Vec turn90() const { return Vec(-y, x); }
    Vec rotate(double rad) const { return Vec(x * cos(rad) - y * sin(rad), x * sin(rad) +
    ↪  y * cos(rad)); }
};

double Dot(Vec a, Vec b) { return a.x * b.x + a.y * b.y; }
double Cross(Vec a, Vec b) { return a.x * b.y - a.y * b.x; }
double Det(Vec a, Vec b, Vec c) { return Cross(b - a, c - a); }

double Angle(Vec a, Vec b) { return acos(Dot(a, b) / (a.len() * b.len())); }

struct Line {
    Vec a, b;
    double theta;

    void GetTheta() {
        theta = atan2(b.y - a.y, b.x - a.x);
    }

    Line() = default;
    Line(Vec _a, Vec _b): a(_a), b(_b) {
        GetTheta();
    }

    bool operator < (const Line &oth) const {
        return theta < oth.theta;
    }

    Vec v() const { return b - a; }
    double k() const { return !sgn(b.x - a.x) ? INF : (b.y - a.y) / (b.x - a.x); }
```

```
};

bool OnLine(Vec p, Line l) {
    return sgn(Cross(l.a - p, l.b - p)) == 0;
}

bool OnSeg(Vec p, Line l) {
    return OnLine(p, l) && sgn(Dot(l.b - l.a, p - l.a)) >= 0 && sgn(Dot(l.a - l.b, p -
    ↪  l.b)) >= 0;
}

bool Parallel(Line l1, Line l2) {
    return sgn(Cross(l1.v(), l2.v())) == 0;
}

Vec Intersect(Line l1, Line l2) {
    double s1 = Det(l1.a, l1.b, l2.a);
    double s2 = Det(l1.a, l1.b, l2.b);
    return (l2.a * s2 - l2.b * s1) / (s2 - s1);
}

Vec Project(Vec p, Line l) {
    return l.a + l.v() * (Dot(p - l.a, l.v())) / l.v().len2();
}

double DistToLine(Vec p, Line l) {
    return std::abs(Cross(p - l.a, l.v())) / l.v().len();
}

int Dir(Vec p, Line l) {
    return sgn(Cross(p - l.b, l.v()));
}

bool SegIntersect(Line l1, Line l2) { // Strictly
    return Dir(l2.a, l1) * Dir(l2.b, l1) < 0 && Dir(l1.a, l2) * Dir(l1.b, l2) < 0;
}

bool InTriangle(Vec p, std::vector<Vec> tri) {
    if (sgn(Cross(tri[1] - tri[0], tri[2] - tri[0])) < 0)
        std::reverse(tri.begin(), tri.end());
    for (int i = 0; i < 3; ++i)
        if (Dir(p, Line(tri[i], tri[(i + 1) % 3])) == 1)
            return false;
    return true;
}

std::vector<Vec> ConvexCut(const std::vector<Vec> &ps, Line l) { // Use the
↪  counterclockwise halfplane of l to cut a convex polygon
    std::vector<Vec> qs;
    for (int i = 0; i < (int)ps.size(); ++i) {
        Vec p1 = ps[i], p2 = ps[(i + 1) % ps.size()];
        int d1 = sgn(Cross(l.v(), p1 - l.a)), d2 = sgn(Cross(l.v(), p2 - l.a));
        if (d1 >= 0) qs.push_back(p1);
        if (d1 * d2 < 0) qs.push_back(Intersect(Line(p1, p2), l));
    }
    return qs;
```

```
}

struct Cir {
    Vec o;
    double r;

    Cir() = default;
    Cir(Vec _o, double _r): o(_o), r(_r) {}

    Vec PointOnCir(double rad) const { return Vec(o.x + cos(rad) * r, o.y + sin(rad) *
    ↪  r); }
};

bool Intersect(Cir c, Line l, Vec &p1, Vec &p2) {
    double x = Dot(l.a - c.o, l.b - l.a);
    double y = (l.b - l.a).len2();
    double d = Sqr(x) - y * ((l.a - c.o).len2() - Sqr(c.r));
    if (sgn(d) < 0) return false;
    d = std::max(d, 0.);
    Vec p = l.a - (l.v() * (x / y));
    Vec delta = l.v() * (Sqrt(d) / y);
    p1 = p + delta; p2 = p - delta;
    return true;
}

bool Intersect(Cir a, Cir b, Vec &p1, Vec &p2) { // Not suitable for coincident circles
    double s1 = (a.o - b.o).len();
    if (sgn(s1 - a.r - b.r) > 0 || sgn(s1 - std::abs(a.r - b.r)) < 0) return false;
    double s2 = (Sqr(a.r) - Sqr(b.r)) / s1;
    double aa = (s1 + s2) * 0.5, bb = (s1 - s2) * 0.5;
    Vec o = (b.o - a.o) * (aa / (aa + bb)) + a.o;
    Vec delta = (b.o - a.o).norm().turn90() * Sqrt(a.r * a.r - aa * aa);
    p1 = o + delta; p2 = o - delta;
    return true;
}

bool Tangent(Cir c, Vec p0, Vec &p1, Vec &p2) { // In clockwise order
    double x = (p0 - c.o).len2(), d = x - Sqr(c.r);
    if (sgn(d) <= 0) return false;
    Vec p = (p0 - c.o) * (Sqr(c.r) / x);
    Vec delta = ((p0 - c.o) * (-c.r * Sqrt(d) / x)).turn90();
    p1 = c.o + p + delta; p2 = c.o + p - delta;
    return true;
}

std::vector<Line> ExTangent(Cir c1, Cir c2) { // External tangent line
    std::vector<Line> res;
    if (sgn(c1.r - c2.r) == 0) {
        Vec dir = c2.o - c2.o;
        dir = (dir * (c1.r / dir.len())).turn90();
        res.push_back(Line(c1.o + dir, c2.o + dir));
        res.push_back(Line(c1.o - dir, c2.o - dir));
    } else {
        Vec p = (c1.o * -c2.r + c2.o * c1.r) / (c1.r - c2.r);
        Vec p1, p2, q1, q2;
        if (Tangent(c1, p, p1, p2) && Tangent(c2, p, q1, q2)) {
```

```
            res.push_back(Line(p1, q1));
            res.push_back(Line(p2, q2));
        }
    }
    return res;
}

std::vector<Line> InTangent(Cir c1, Cir c2) { // Internal tangent line
    std::vector<Line> res;
    Vec p = (c1.o * c2.r + c2.o * c1.r) / (c1.r + c2.r);
    Vec p1, p2, q1, q2;
    if (Tangent(c1, p, p1, p2) && Tangent(c2, p, q1, q2)) {
        res.push_back(Line(p1, q1));
        res.push_back(Line(p2, q2));
    }
    return res;
}

bool InPoly(Vec p, std::vector<Vec> poly) {
    int cnt = 0;
    for (int i = 0; i < (int)poly.size(); ++i) {
        Vec a = poly[i], b = poly[(i + 1) % poly.size()];
        if (OnSeg(p, Line(a, b)))
            return false;
        int x = sgn(Det(a, p, b));
        int y = sgn(a.y - p.y);
        int z = sgn(b.y - p.y);
        cnt += (x > 0 && y <= 0 && z > 0);
        cnt -= (x < 0 && z <= 0 && y > 0);
    }
    return cnt;
}
```

1.2 半平面交

```
bool HalfPlaneIntersect(std::vector<Line> L, std::vector<Vec> &ch) {
        std::sort(L.begin(), L.end());
        int head = 0, tail = 0;
        Vec *p = new Vec[L.size()];
        Line *q = new Line[L.size()];
        q[0] = L[0];
        for (int i = 1; i < (int)L.size(); i++) {
                while (head < tail && Dir(p[tail - 1], L[i]) != 1) tail--;
                while (head < tail && Dir(p[head], L[i]) != 1) head++;
                q[++tail] = L[i];
                if (!sgn(Cross(q[tail].b - q[tail].a, q[tail - 1].b - q[tail - 1].a))) {
                        tail--;
                        if (Dir(L[i].a, q[tail]) == 1) q[tail] = L[i];
                }
                if (head < tail) p[tail - 1] = Intersect(q[tail - 1], q[tail]);
        }
        while (head < tail && Dir(p[tail - 1], q[head]) != 1) tail--;
        if (tail - head <= 1) return false;
        p[tail] = Intersect(q[head], q[tail]);
        for (int i = head; i <= tail; i++) ch.push_back(p[i]);
        delete[] p; delete[] q;
```

```
        return true;
}
```

1.3 二维最小圆覆盖

```
Vec ExCenter(Vec a, Vec b, Vec c) {
    if (a == b) return (a + c) / 2;
    if (a == c) return (a + b) / 2;
    if (b == c) return (a + b) / 2;
    Vec m1 = (a + b) / 2;
    Vec m2 = (b + c) / 2;
    return Insersect(Line(m1, m1 + (b - a).turn90()), Line(m2, m2 + (c - b).turn90()));
}

Cir Solve(std::vector<Vec> p) {
    std::random_shuffle(p.begin(), p.end());
    Vec o = p[0];
    double r = 0;
    for (int i = 1; i < (int)p.size(); ++i) {
        if (sgn((p[i] - o).len() - r) <= 0) continue;
        o = (p[0] + p[i]) / 2;
        r = (o - p[i]).len();
        for (int j = 0; j < i; ++j) {
            if (sgn((p[j] - o).len() - r) <= 0) continue;
            o = (p[i] + p[j]) / 2;
            r = (o - p[i]).len();
            for (int k = 0; k < j; ++k) {
                if (sgn((p[k] - o).len() - r) <= 0) continue;
                o = ExCenter(p[i], p[j], p[k]);
                r = (o - p[i]).len();
            }
        }
    }
    return Cir(o, r);
}
```

1.4 凸包

```
std::vector<Vec> ConvexHull(std::vector<Vec> p) {
    std::sort(p.begin(), p.end());
    std::vector<Vec> ans, S;
    for (int i = 0; i < (int)p.size(); ++i) {
        while (S.size() >= 2 && sgn(Det(S[S.size() - 2], S.back(), p[i])) <= 0)
            S.pop_back();
        S.push_back(p[i]);
    }
    ans = S;
    S.clear();
    for (int i = p.size() - 1; i >= 0; --i) {
        while (S.size() >= 2 && sgn(Det(S[S.size() - 2], S.back(), p[i])) <= 0)
            S.pop_back();
        S.push_back(p[i]);
    }
    for (int i = 1; i + 1 < (int)S.size(); ++i)
        ans.push_back(S[i]);
    return ans;
```

```
}
```

1.5 凸包游戏

```
/*
   给定凸包，$\log n$ 内完成各种询问，具体操作有 :
   1. 判定一个点是否在凸包内
   2. 询问凸包外的点到凸包的两个切点
   3. 询问一个向量关于凸包的切点
   4. 询问一条直线和凸包的交点
   INF 为坐标范围，需要定义点类大于号
   改成实数只需修改 sign 函数，以及把 long long 改为 double 即可
   构造函数时传入凸包要求无重点，面积非空，以及 pair(x,y) 的最小点放在第一个
*/
const int INF = 1000000000;
struct Convex
{
        int n;
        vector<Point> a, upper, lower;
        Convex(vector<Point> _a) : a(_a) {
                n = a.size();
                int ptr = 0;
                for(int i = 1; i < n; ++ i) if (a[ptr] < a[i]) ptr = i;
                for(int i = 0; i <= ptr; ++ i) lower.push_back(a[i]);
                for(int i = ptr; i < n; ++ i) upper.push_back(a[i]);
                upper.push_back(a[0]);
        }
        int sign(long long x) { return x < 0 ? -1 : x > 0; }
        pair<long long, int> get_tangent(vector<Point> &convex, Point vec) {
                int l = 0, r = (int)convex.size() - 2;
                for( ; l + 1 < r; ) {
                        int mid = (l + r) / 2;
                        if (sign((convex[mid + 1] - convex[mid]).det(vec)) > 0) r = mid;
                        else l = mid;
                }
                return max(make_pair(vec.det(convex[r]), r)
                        , make_pair(vec.det(convex[0]), 0));
        }
        void update_tangent(const Point &p, int id, int &i0, int &i1) {
                if ((a[i0] - p).det(a[id] - p) > 0) i0 = id;
                if ((a[i1] - p).det(a[id] - p) < 0) i1 = id;
        }
        void binary_search(int l, int r, Point p, int &i0, int &i1) {
                if (l == r) return;
                update_tangent(p, l % n, i0, i1);
                int sl = sign((a[l % n] - p).det(a[(l + 1) % n] - p));
                for( ; l + 1 < r; ) {
                        int mid = (l + r) / 2;
                        int smid = sign((a[mid % n] - p).det(a[(mid + 1) % n] - p));
                        if (smid == sl) l = mid;
                        else r = mid;
                }
                update_tangent(p, r % n, i0, i1);
        }
        int binary_search(Point u, Point v, int l, int r) {
                int sl = sign((v - u).det(a[l % n] - u));
```

```
        for( ; l + 1 < r; ) {
                int mid = (l + r) / 2;
                int smid = sign((v - u).det(a[mid % n] - u));
                if (smid == sl) l = mid;
                else r = mid;
        }
        return l % n;
}
// 判定点是否在凸包内，在边界返回 true
bool contain(Point p) {
        if (p.x < lower[0].x || p.x > lower.back().x) return false;
        int id = lower_bound(lower.begin(), lower.end()
                , Point(p.x, -INF)) - lower.begin();
        if (lower[id].x == p.x) {
                if (lower[id].y > p.y) return false;
        } else if ((lower[id - 1] - p).det(lower[id] - p) < 0) return false;
        id = lower_bound(upper.begin(), upper.end(), Point(p.x, INF)
                , greater<Point>()) - upper.begin();
        if (upper[id].x == p.x) {
                if (upper[id].y < p.y) return false;
        } else if ((upper[id - 1] - p).det(upper[id] - p) < 0) return false;
        return true;
}
// 求点 p 关于凸包的两个切点，如果在凸包外则有序返回编号
// 共线的多个切点返回任意一个，否则返回 false
bool get_tangent(Point p, int &i0, int &i1) {
        if (contain(p)) return false;
        i0 = i1 = 0;
        int id = lower_bound(lower.begin(), lower.end(), p) - lower.begin();
        binary_search(0, id, p, i0, i1);
        binary_search(id, (int)lower.size(), p, i0, i1);
        id = lower_bound(upper.begin(), upper.end(), p
                , greater<Point>()) - upper.begin();
        binary_search((int)lower.size() - 1, (int)lower.size() - 1 + id, p, i0,
        ↪  i1);
        binary_search((int)lower.size() - 1 + id
                , (int)lower.size() - 1 + (int)upper.size(), p, i0, i1);
        return true;
}
// 求凸包上和向量 vec 叉积最大的点，返回编号，共线的多个切点返回任意一个
int get_tangent(Point vec) {
        pair<long long, int> ret = get_tangent(upper, vec);
        ret.second = (ret.second + (int)lower.size() - 1) % n;
        ret = max(ret, get_tangent(lower, vec));
        return ret.second;
}
// 求凸包和直线 u,v 的交点，如果无严格相交返回 false.
//如果有则是和 (i,next(i)) 的交点，两个点无序，交在点上不确定返回前后两条线段其中之
↪  一
bool get_intersection(Point u, Point v, int &i0, int &i1) {
        int p0 = get_tangent(u - v), p1 = get_tangent(v - u);
        if (sign((v - u).det(a[p0] - u)) * sign((v - u).det(a[p1] - u)) < 0) {
                if (p0 > p1) swap(p0, p1);
                i0 = binary_search(u, v, p0, p1);
                i1 = binary_search(u, v, p1, p0 + n);
                return true;
```

```
                } else {
                        return false;
                }
        }
};
```

1.6 圆并

```
double ans[2001];
struct Point {
        double x, y;
        Point(){}
        Point(const double & x, const double & y) : x(x), y(y) {}
        void scan() {scanf("%lf%lf", &x, &y);}
        double sqrlen() {return sqr(x) + sqr(y);}
        double len() {return sqrt(sqrlen());}
        Point rev() {return Point(y, -x);}
        void print() {printf("%f %f\n", x, y);}
        Point zoom(const double & d) {double lambda = d / len(); return Point(lambda * x,
        ↪  lambda * y);}
} dvd, a[2001];
Point centre[2001];
double atan2(const Point & x) {
        return atan2(x.y, x.x);
}
Point operator - (const Point & a, const Point & b) {
        return Point(a.x - b.x, a.y - b.y);
}
Point operator + (const Point & a, const Point & b) {
        return Point(a.x + b.x, a.y + b.y);
}
double operator * (const Point & a, const Point & b) {
        return a.x * b.y - a.y * b.x;
}
Point operator * (const double & a, const Point & b) {
        return Point(a * b.x, a * b.y);
}
double operator % (const Point & a, const Point & b) {
        return a.x * b.x + a.y * b.y;
}
struct circle {
        double r; Point o;
        circle() {}
        void scan() {
                o.scan();
                scanf("%lf", &r);
        }
} cir[2001];
struct arc {
        double theta;
        int delta;
        Point p;
        arc() {};
        arc(const double & theta, const Point & p, int d) : theta(theta), p(p), delta(d)
        ↪  {}
} vec[4444];
```

```
int nV;
inline bool operator < (const arc & a, const arc & b) {
        return a.theta + eps < b.theta;
}
int cnt;
inline void psh(const double t1, const Point p1, const double t2, const Point p2) {
        if(t2 + eps < t1)
                cnt++;
        vec[nV++] = arc(t1, p1, 1);
        vec[nV++] = arc(t2, p2, -1);
}
inline double cub(const double & x) {
        return x * x * x;
}
inline void combine(int d, const double & area, const Point & o) {
        if(sign(area) == 0) return;
        centre[d] = 1 / (ans[d] + area) * (ans[d] * centre[d] + area * o);
        ans[d] += area;
}
bool equal(const double & x, const double & y) {
        return x + eps>  y and y + eps > x;
}
bool equal(const Point & a, const Point & b) {
        return equal(a.x, b.x) and equal(a.y, b.y);
}
bool equal(const circle & a, const circle & b) {
        return equal(a.o, b.o) and equal(a.r, b.r);
}
bool f[2001];
int main() {
        //freopen("hdu4895.in", "r", stdin);
        int n, m, index;
        while(EOF != scanf("%d%d%d", &m, &n, &index)) {
                index--;
                for(int i(0); i < m; i++) {
                        a[i].scan();
                }
                for(int i(0); i < n; i++) {
                        cir[i].scan();//n 个圆
                }
                for(int i(0); i < n; i++) {//这一段在去重圆 能加速 删掉不会错
                        f[i] = true;
                        for(int j(0); j < n; j++) if(i != j) {
                                if(equal(cir[i], cir[j]) and i < j or !equal(cir[i],
                                ↪  cir[j]) and cir[i].r < cir[j].r + eps and (cir[i].o -
                                ↪  cir[j].o).sqrlen() < sqr(cir[i].r - cir[j].r) + eps)
                                ↪  {
                                        f[i] = false;
                                        break;
                                }
                        }
                }
                int n1(0);
                for(int i(0); i < n; i++)
                        if(f[i])
                                cir[n1++] = cir[i];
```

```
n = n1;//去重圆结束
fill(ans, ans + n + 1, 0);//ans[i] 表示被圆覆盖至少 i 次的面积
fill(centre, centre + n + 1, Point(0, 0));//centre[i] 表示上面 ans[i] 部
↪ 分的重心
for(int i(0); i < m; i++)
        combine(0, a[i] * a[(i + 1) % m] * 0.5, 1. / 3 * (a[i] + a[(i +
        ↪ 1) % m]));
for(int i(0); i < n; i++) {
        dvd = cir[i].o - Point(cir[i].r, 0);
        nV = 0;
        vec[nV++] = arc(-pi, dvd, 1);
        cnt = 0;
        for(int j(0); j < n; j++) if(j != i) {
                double d = (cir[j].o - cir[i].o).sqrlen();
                if(d < sqr(cir[j].r - cir[i].r) + eps) {
                        if(cir[i].r + i * eps < cir[j].r + j * eps)
                                psh(-pi, dvd, pi, dvd);
                }else if(d + eps < sqr(cir[j].r + cir[i].r)) {
                        double lambda = 0.5 * (1 + (sqr(cir[i].r) -
                        ↪ sqr(cir[j].r)) / d);
                        Point cp(cir[i].o + lambda * (cir[j].o -
                        ↪ cir[i].o));
                        Point nor((cir[j].o -
                        ↪ cir[i].o).rev().zoom(sqrt(sqr(cir[i].r) - (cp
                        ↪ - cir[i].o).sqrlen())));
                        Point frm(cp + nor);
                        Point to(cp - nor);
                        psh(atan2(frm - cir[i].o), frm, atan2(to -
                        ↪ cir[i].o), to);
                }
        }
        sort(vec + 1, vec + nV);
        vec[nV++] = arc(pi, dvd, -1);
        for(int j = 0; j + 1 < nV; j++) {
                cnt += vec[j].delta;
                //if(cnt == 1) {//如果只算 ans[1] 和 centre[1], 可以加这个
                ↪ if 加速.
                        double theta(vec[j + 1].theta - vec[j].theta);
                        double area(sqr(cir[i].r) * theta * 0.5);
                        combine(cnt, area, cir[i].o + 1. / area / 3 *
                        ↪ cub(cir[i].r) * Point(sin(vec[j + 1].theta) -
                        ↪ sin(vec[j].theta), cos(vec[j].theta) -
                        ↪ cos(vec[j + 1].theta)));
                        combine(cnt, -sqr(cir[i].r) * sin(theta) * 0.5,
                        ↪ 1. / 3 * (cir[i].o + vec[j].p + vec[j +
                        ↪ 1].p));
                        combine(cnt, vec[j].p * vec[j + 1].p * 0.5, 1. /
                        ↪ 3 * (vec[j].p + vec[j + 1].p));
                //}
        }
}//板子部分结束 下面是题目
combine(0, -ans[1], centre[1]);
for(int i = 0; i < m; i++) {
        if(i != index)
```

```
                                (a[index] - Point((a[i] - a[index]) * (centre[0] -
                                ↪  a[index]), (a[i] - a[index]) % (centre[0] -
                                ↪  a[index])).zoom((a[i] - a[index]).len())).print();
                        else
                                a[i].print();
                }
        }
        fclose(stdin);
        return 0;
}
```

1.7 最远点对

```
point conv[100000];
int totco, n;
//凸包
void convex( point p[], int n ){
        sort( p, p+n, cmp );
        conv[0]=p[0]; conv[1]=p[1]; totco=2;
        for ( int i=2; i<n; i++ ){
                while ( totco>1 && (conv[totco-1]-conv[totco-2])/(p[i]-conv[totco-2])<=0
                ↪  ) totco--;
                conv[totco++]=p[i];
        }
        int limit=totco;
        for ( int i=n-1; i>=0; i-- ){
                while ( totco>limit &&
                ↪  (conv[totco-1]-conv[totco-2])/(p[i]-conv[totco-2])<=0 ) totco--;
                conv[totco++]=p[i];
        }
}
point pp[100000];
int main(){
        scanf("%d", &n);
        for ( int i=0; i<n; i++ )
        scanf("%d %d", &pp[i].x, &pp[i].y);
        convex( pp, n );
        n=totco;
        for ( int i=0; i<n; i++ ) pp[i]=conv[i];
        n--;
        int ans=0;
        for ( int i=0; i<n; i++ )
        pp[n+i]=pp[i];
        int now=1;
        for ( int i=0; i<n; i++ ){
                point tt=point( pp[i+1]-pp[i] );
                while ( now<2*n-2 && tt/(pp[now+1]-pp[now])>0 ) now++;
                if ( dist( pp[i], pp[now] )>ans ) ans=dist( pp[i], pp[now] );
                if ( dist( pp[i+1], pp[now] )>ans ) ans=dist( pp[i+1], pp[now] );
        }
        printf("%d\n", ans);
}
```

1.8 根轴

根轴定义：到两圆圆幂相等的点形成的直线

两圆 $\{(x_1, y_1), r_1\}$ 和 $\{(x_2, y_2), r_2\}$ 的根轴方程:
$2(x_2 - x_1)x + 2(y_2 - y_1)y + f_1 - f_2 = 0$，其中 $f_1 = {x_1}^2 + {y_1}^2 - {r_1}^2, f_2 = {x_2}^2 + {y_2}^2 - {r_2}^2$。

字符串

2 manacher

```
#include<iostream>
#include<cstring>
using namespace std;
char Mana[202020];
int cher[202020];
int Manacher(char *S)
{
        int len=strlen(S),id=0,mx=0,ret=0;
        Mana[0]='$';
        Mana[1]='#';
        for(int i=0;i<len;i++)
        {
                Mana[2*i+2]=S[i];
                Mana[2*i+3]='#';
        }
        Mana[2*len+2]=0;
        for(int i=1;i<=2*len+1;i++)
        {
                if(i<mx)
                        cher[i]=min(cher[2*id-i],mx-i);
                else
                        cher[i]=0;
                while(Mana[i+cher[i]+1]==Mana[i-cher[i]-1])
                        cher[i]++;
                if(cher[i]+i>mx)
                {
                        mx=cher[i]+i;
                         id=i;
                }
                ret=max(ret,cher[i]);
        }
        return ret;
}
char S[101010];
int main()
{
        ios::sync_with_stdio(false);
        cin.tie(0);
        cout.tie(0);
        cin>>S;
        cout<<Manacher(S)<<endl;
        return 0;
}
```

3 后缀数组

```
const int maxl=1e5+1e4+5;
const int maxn=maxl*2;
int a[maxn],x[maxn],y[maxn],c[maxn],sa[maxn],rank[maxn],height[maxn];
void calc_sa(int n){
        int m=alphabet,k=1;
        memset(c,0,sizeof(*c)*(m+1));
        for(int i=1;i<=n;i++)c[x[i]=a[i]]++;
        for(int i=1;i<=m;i++)c[i]+=c[i-1];
        for(int i=1;i<=n;i++)sa[c[x[i]]--]=i;
        for(;k<=n;k<<=1){
                int tot=k;
                for(int i=n-k+1;i<=n;i++)y[i-n+k]=i
                for(int i=1;i<=n;i++)
                        if(sa[i]>k)y[++tot]=sa[i]-k
                memset(c,0,sizeof(*c)*(m+1));
                for(int i=1;i<=n;i++)c[x[i]]++;
                for(int i=1;i<=m;i++)c[i]+=c[i-1];
                for(int i=n;i>=1;i--)sa[c[x[y[i]]]-
                for(int i=1;i<=n;i++)y[i]=x[i];
                tot=1;x[sa[1]]=1;
                for(int i=2;i<=n;i++){
                        if(max(sa[i],sa[i-1])+k>n||
                                ++tot;
                        x[sa[i]]=tot;
                }
                if(tot==n)break;else m=tot;
        }
}
void calc_height(int n){
        for(int i=1;i<=n;i++)rank[sa[i]]=i;
        for(int i=1;i<=n;i++){
                height[rank[i]]=max(0,height[rank[i
                if(rank[i]==1)continue;
                int j=sa[rank[i]-1];
                while(max(i,j)+height[rank[i]]<=n&&
                        ++height[rank[i]];
        }
}
```

4 后缀自动机

```
#include<iostream>
#include<cstring>
using namespace std;
const int MaxPoint=1010101;
struct Suffix_AutoMachine{
        int son[MaxPoint][27],pre[MaxPoint],step[Ma
        int NewNode(int stp)
        {
                num++;
                memset(son[num],0,sizeof(son[num]))
                pre[num]=0;
                step[num]=stp;
                return num;
        }
        Suffix_AutoMachine()
        {
                num=0;
                root=last=NewNode(0);
        }
```

```
        void push_back(int ch)
        {
                int np=NewNode(step[last]+1);
                right[np]=1;
                step[np]=step[last]+1;
                int p=last;
                while(p&&!son[p][ch])
                {
                        son[p][ch]=np;
                        p=pre[p];
                }
                if(!p)
                        pre[np]=root;
                else
                {
                        int q=son[p][ch];
                        if(step[q]==step[p]+1)
                                pre[np]=q;
                        else
                        {
                                int nq=NewNode(step[p]+1);
                                memcpy(son[nq],son[q],sizeof(son[q]));
                                step[nq]=step[p]+1;
                                pre[nq]=pre[q];
                                pre[q]=pre[np]=nq;
                                while(p&&son[p][ch]==q)
                                {
                                        son[p][ch]=nq;
                                        p=pre[p];
                                }
                        }
                }
                last=np;
        }
};
/*

int arr[1010101];
bool Step_Cmp(int x,int y)
{
        return S.step[x]<S.step[y];
}
void Get_Right()
{
        for(int i=1;i<=S.num;i++)
                arr[i]=i;
        sort(arr+1,arr+S.num+1,Step_Cmp);
        for(int i=S.num;i>=2;i--)
                S.right[S.pre[arr[i]]]+=S.right[arr[i]];
}
*/
int main()
{

        return 0;
}
```

5 广义后缀自动机

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

const int MAXL = 1e5 + 5;

namespace GSAM {
    struct Node *pool_pointer;
    struct Node {
        Node *to[26], *parent;
        int step;

        Node(int STEP = 0): step(STEP) {
            memset(to, 0, sizeof to);
            parent = 0;
        }

        void *operator new (size_t) {
            return pool_pointer++;
        }
    } pool[MAXL << 1], *root;

    void init() {
        pool_pointer = pool;
        root = new Node();
    }

    Node *Extend(Node *np, char ch) {
        static Node *last, *q, *nq;

        int x = ch - 'a';

        if (np->to[x]) {
            last = np;
            q = last->to[x];
            if (q->step == last->step + 1) np = q;
            else {
                nq = new Node(last->step + 1);
                memcpy(nq->to, q->to, sizeof q->to)
                nq->parent = q->parent;
                q->parent = np->parent = nq;
                for (; last && last->to[x] == q; la
                    last->to[x] = nq;

                np = nq;
            }
        } else {
            last = np; np = new Node(last->step + 1
            for (; last && !last->to[x]; last = las
                last->to[x] = np;
            if (!last) np->parent = last;
            else {
                q = last->to[x];
                if (q->step == last->step + 1) np->
                else {
                    nq = new Node(last->step + 1);
```

```
                memcpy(nq->to, q->to, sizeof q->to);
                nq->parent = q->parent;
                q->parent = np->parent = nq;
                for (; last && last->to[x] == q; last = last->parent)
                    last->to[x] = nq;
            }
        }

        return np;
    }
}

int main() {


    return 0;
}
```

6 回文自动机

```
//Tsinsen A1280 最长双回文串
#include<iostream>
#include<cstring>
using namespace std;

const int maxn = 100005;// n(空间复杂度 o(n*ALP)), 实际开 n 即可
const int ALP = 26;

struct PAM{ // 每个节点代表一个回文串
int next[maxn][ALP]; // next 指针, 参照 Trie 树
int fail[maxn]; // fail 失配后缀链接
int cnt[maxn]; // 此回文串出现个数
int num[maxn];
int len[maxn]; // 回文串长度
int s[maxn]; // 存放添加的字符
int last; //指向上一个字符所在的节点, 方便下一次 add
int n; // 已添加字符个数
int p; // 节点个数

int newnode(int w)
{// 初始化节点, w= 长度
        for(int i=0;i<ALP;i++)
        next[p][i] = 0;
        cnt[p] = 0;
        num[p] = 0;
        len[p] = w;
        return p++;
}
void init()
{
p = 0;
newnode(0);
newnode(-1);
last = 0;
n = 0;
s[n] = -1; // 开头放一个字符集中没有的字符, 减少特判
fail[0] = 1;
}
int get_fail(int x)
{ // 和 KMP 一样, 失配后找一个尽量最长的
while(s[n-len[x]-1] != s[n]) x = fail[x];
return x;
}
int add(int c)
{
c -= 'a';
s[++n] = c;
int cur = get_fail(last);
if(!next[cur][c])
{
int now = newnode(len[cur]+2);
fail[now] = next[get_fail(fail[cur])][c];
next[cur][c] = now;
num[now] = num[fail[now]] + 1;
}
last = next[cur][c];
cnt[last]++;
return len[last];
}
void count()
{
// 最后统计一遍每个节点出现个数
// 父亲累加儿子的 cnt, 类似 SAM 中 parent 树
// 满足 parent 拓扑关系
for(int i=p-1;i>=0;i--)
cnt[fail[i]] += cnt[i];
}
}pam;
char S[101010];
int l[101010],r[101010];
int main()
{
cin>>S;
int len=strlen(S);
pam.init();
for(int i=0;i<len;i++)
l[i]=pam.add(S[i]);
pam.init();
for(int i=len-1;i>=0;i--)
r[i]=pam.add(S[i]);
pam.init();
int ans=0;
for(int i=0;i<len-1;i++)
ans=max(ans,l[i]+r[i+1]);
cout<<ans<<endl;
return 0;
}
```

7 Lyndon Word Decomposition NewMeta

```
// 把串 s 划分成 lyndon words, s1, s2, s3, ..., sk
// 每个串都严格小于他们的每个后缀，且串大小不增
// 如果求每个前缀的最小后缀，取最后一次 k 经过这个前缀的右边界时的信息更新
// 如果求每个前缀的最大后缀，更改大小于号，并且取第一次 k 经过这个前缀的信息更新
void lynDecomp() {
        vector<string> ss;
        for (int i = 0; i < n; ) {
                int j = i, k = i + 1; //mnsuf[i] = i;
                for (; k < n && s[k] >= s[j]; k++) {
                        if (s[k] == s[j]) j++; // mnsuf[k] = mnsuf[j] + k - j;
                        else j = i; // mnsuf[k] = i;
                }
                for (; i <= j; i += k - j) ss.push_back(s.substr(i, k - j));
        }
}
```

8 EXKMP NewMeta

```
// 如果想求一个字符串相对另外一个字符串的最长公共前缀，可以把他们拼接起来从而求得
void exkmp(char *s, int *a, int n) {
        a[0] = n; int p = 0, r = 0;
        for (int i = 1; i < n; ++i) {
                a[i] = (r > i) ? min(r - i, a[i - p]) : 0;
                while (i + a[i] < n && s[i + a[i]] == s[a[i]]) ++a[i];
                if (r < i + a[i]) r = i + a[i], p = i;
}}
```

数据结构

9 Link-Cut-Tree

```
namespace LinkCutTree {
	struct Node {
		Node *ch[2], *fa;
		int sz; bool rev;
		Node() {
			ch[0] = ch[1] = fa = NULL;
			sz = 1; rev = 0;
		}

		void reverse() { if (this) rev ^= 1; }

		void down() {
			if (rev) {
				std::swap(ch[0], ch[1]);
				for (int i = 0; i < 2; i++) ch[i]->reverse();
				rev = 0;
			}
		}

		int size() { return this ? sz : 0; }

		void update() {
			sz = 1 + ch[0]->size() + ch[1]->size();
		}

		int which() {
			if (!fa || (this != fa->ch[0] && this != fa->ch[1])) return -1;
			return this == fa->ch[1];
		}
	} *pos[100005];

	void rotate(Node *k) {
		Node *p = k->fa;
		int l = k->which(), r = l ^ 1;
		k->fa = p->fa;
		if (p->which() != -1) p->fa->ch[p->which()] = k;
		p->ch[l] = k->ch[r];
		if (k->ch[r]) k->ch[r]->fa = p;
		k->ch[r] = p; p->fa = k;
		p->update(); k->update();
	}

	void splay(Node *k) {
		static stack<Node *> stk;
		Node *p = k;
		while (true) {
			stk.push(p);
			if (p->which() == -1) break;
			p = p->fa;
		}
		while (!stk.empty()) {
			stk.top()->down(); stk.pop();
		}

		while (k->which() != -1) {
			p = k->fa;
			if (p->which() != -1) {
				if (p->which() ^ k->which()) rotate(k);
				else rotate(p);
			}
			rotate(k);
		}
	}

	void access(Node *k) {
		Node *p = NULL;
		while (k) {
			splay(k);
			k->ch[1] = p;
			(p = k)->update();
			k = k->fa;
		}
	}

	void evert(Node *k) {
		access(k);
		splay(k);
		k->reverse();
	}

	Node *get_root(Node *k) {
		access(k);
		splay(k);
		while (k->ch[0]) k = k->ch[0];
		return k;
	}

	void link(Node *u, Node *v) {
		evert(u);
		u->fa = v;
	}

	void cut(Node *u, Node *v) {
		evert(u);
		access(v);
		splay(v);
		// if (v->ch[0] != u) return;
		v->ch[0] = u->fa = NULL;
		v->update();
	}
}
```

10 KDTree

```
namespace KDTree {
    struct Vec {
```

```
    int d[2];

    Vec() = default;
    Vec(int x, int y) {
        d[0] = x; d[1] = y;
    }

    bool operator == (const Vec &oth) const {
        for (int i = 0; i < 2; ++i)
            if (d[i] != oth.d[i]) return false;
        return true;
    }
};

struct Rec {
    int mn[2], mx[2];

    Rec() = default;
    Rec(const Vec &p) {
        for (int i = 0; i < 2; ++i)
            mn[i] = mx[i] = p.d[i];
    }

    static Rec Merge(const Rec &a, const Rec &b) {
        Rec res;
        for (int i = 0; i < 2; ++i) {
            res.mn[i] = std::min(a.mn[i], b.mn[i]);
            res.mx[i] = std::max(a.mx[i], b.mx[i]);
        }
        return res;
    }

    static bool In(const Rec &a, const Rec &b) {
        for (int i = 0; i < 2; ++i)
            if (a.mn[i] < b.mn[i] || a.mx[i] > b.mx[i]) return false;
        return true;
    }

    static bool Out(const Rec &a, const Rec &b) {
        for (int i = 0; i < 2; ++i)
            if (a.mx[i] < b.mn[i] || a.mn[i] > b.mx[i]) return true;
        return false;
    }
};

struct Node *pool_pointer;
struct Node {
    Node *ch[2];
    Vec p;
    Rec rec;
    int sum, val;
    int size;

    Node() = default;
    Node(const Vec &_p, int _v): p(_p), rec(_p), sum(_v), val(_v) {
        ch[0] = ch[1] = 0;
        size = 1;
    }

    bool Bad() {
        const double alpha = 0.75;

        for (int i = 0; i < 2; ++i)
            if (ch[i] && ch[i]->size > size * a
        return false;
    }

    void Update() {
        sum = val;
        size = 1;
        rec = Rec(p);
        for (int i = 0; i < 2; ++i) if (ch[i])
            sum += ch[i]->sum;
            size += ch[i]->size;
            rec = Rec::Merge(rec, ch[i]->rec);
        }
    }

    void *operator new (size_t) {
        return pool_pointer++;
    }
} pool[MAXN], *root;
Node *null = 0;

std::pair<Node *&, int> Insert(Node *&k, const
    if (!k) {
        k = new Node(p, val);
        return std::pair<Node *&, int>(null, -1
    }
    if (k->p == p) {
        k->sum += val;
        k->val += val;
        return std::pair<Node *&, int>(null, -1
    }
    std::pair<Node *&, int> res = Insert(k->ch[
    k->Update();
    if (k->Bad()) return std::pair<Node *&, int
    return res;
}

Node *nodes[MAXN];
int node_cnt;

void Traverse(Node *k) {
    if (!k) return;
    Traverse(k->ch[0]);
    nodes[++node_cnt] = k;
    Traverse(k->ch[1]);
}
int _dim;
```