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 $f_1 = {x_1}^2 + {y_1}^2 - {r_1}^2, f_2 = {x_2}^2 + {y_2}^2 - {r_2}^2$。

2 字符串

2.1 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;
}
```

2.2 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]]]--]=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||y[sa[i]]!=y[sa[i-
                ++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-1]]-1);
        if(rank[i]==1)continue;
        int j=sa[rank[i]-1];
        while(max(i,j)+height[rank[i]]<=n&&a[i+height[ra
            ++height[rank[i]];
    }
}
```

2.3 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[MaxPoint],r
    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;
}
```

2.4 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; last = last->parent)
                    last->to[x] = nq;

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

        return np;
    }
}

int main() {
```

```
    return 0;
}
```

2.5 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;
}
```

2.6 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
// 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
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));
    }
}
```

2.7 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
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;
}}
```

3 数据结构

3.1 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();
    }
}
```

3.2 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) { // a in 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
            ↪ * alpha) return true;
        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 Vec &p, int val, int dim) {
    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[p.d[dim] >=
    ↪ k->p.d[dim]], p, val, dim ^ 1);
    k->Update();
    if (k->Bad()) return std::pair<Node *&,
    ↪ int>(k, dim);
    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;

    bool cmp(Node *a, Node *b) {
        return a->p.d[_dim] < b->p.d[_dim];
    }

    void Build(Node *&k, int l, int r, int dim)
    ↪  {
        if (l > r) return;
        int mid = (l + r) >> 1;
        _dim = dim;
        std::nth_element(nodes + l, nodes +
        ↪  mid, nodes + r + 1, cmp);

        k = nodes[mid]; k->ch[0] = k->ch[1] =
        ↪  0;
        Build(k->ch[0], l, mid - 1, dim ^ 1);
        Build(k->ch[1], mid + 1, r, dim ^ 1);
        k->Update();
    }

    void Rebuild(Node *&k, int dim) {
        node_cnt = 0;
        Traverse(k);
        Build(k, 1, node_cnt, dim);
    }

    int Query(Node *k, const Rec &rec) {
        if (!k) return 0;
        if (Rec::Out(k->rec, rec)) return 0;
        if (Rec::In(k->rec, rec)) return
        ↪  k->sum;
        int res = 0;
        if (Rec::In(k->p, rec)) res += k->val;
        for (int i = 0; i < 2; ++i)
            res += Query(k->ch[i], rec);
        return res;
    }

    // --------

    void Init() {
        pool_pointer = pool;
        root = 0;
    }

    void Insert(int x, int y, int val) {
        std::pair<Node *&, int> p =
        ↪  Insert(root, Vec(x, y), val, 0);
        if (p.first != null) Rebuild(p.first,
        ↪  p.second);
    }

    int Query(int x1, int y1, int x2, int y2) {
        Rec rec = Rec::Merge(Vec(x1, y1),
        ↪  Vec(x2, y2));
        return Query(root, rec);
    }
}
```

3.3 莫队上树

```
Let dfn_s[u] <= dfn_s[v].
If u is v's ancient, query(dfn_s[u],
↪  dfn_s[v]).
Else query(dfn_t[u], dfn_s[v]) + lca(u, v).
```

4 图论

4.1 点双连通分量

```
/*
 * Point Bi-connected Component
 * Check: VALLA 5135
 */

typedef std::pair<int, int> pii;
#define mkpair std::make_pair

int n, m;
std::vector<int> G[MAXN];

int dfn[MAXN], low[MAXN], bcc_id[MAXN],
↪  bcc_cnt, stamp;
bool iscut[MAXN];

std::vector<int> bcc[MAXN]; // Unnecessary

pii stk[MAXN]; int stk_top;
// Use a handwritten structure to get higher efficiency

void Tarjan(int now, int fa) {
    int child = 0;
    dfn[now] = low[now] = ++stamp;
    for (int to: G[now]) {
        if (!dfn[to]) {
            stk[++stk_top] = mkpair(now, to);
            ↪  ++child;
            Tarjan(to, now);
            low[now] = std::min(low[now],
            ↪  low[to]);
            if (low[to] >= dfn[now]) {
                iscut[now] = 1;
                bcc[++bcc_cnt].clear();
                while (1) {
                    pii tmp = stk[stk_top--];
                    if (bcc_id[tmp.first] !=
                    ↪  bcc_cnt) {
                        bcc[bcc_cnt].push_back(tmp.first
                        bcc_id[tmp.first] =
                        ↪  bcc_cnt;
                    }
                    if (bcc_id[tmp.second] !=
                    ↪  bcc_cnt) {
                        bcc[bcc_cnt].push_back(tmp.secon
                        bcc_id[tmp.second] =
                        ↪  bcc_cnt;
                    }
                    if (tmp.first == now &&
                    ↪  tmp.second == to)
                        break;
                }
```

```
            }
        }
        else if (dfn[to] < dfn[now] && to !=
        ↪ fa) {
            stk[++stk_top] = mkpair(now, to);
            low[now] = std::min(low[now],
            ↪ dfn[to]);
        }
    }
    if (!fa && child == 1)
        iscut[now] = 0;
}

void PBCC() {
    memset(dfn, 0, sizeof dfn);
    memset(low, 0, sizeof low);
    memset(iscut, 0, sizeof iscut);
    memset(bcc_id, 0, sizeof bcc_id);
    stamp = bcc_cnt = stk_top = 0;

    for (int i = 1; i <= n; ++i)
        if (!dfn[i]) Tarjan(i, 0);
}
```

4.2 边双连通分量

```
/*
 * Edge Bi-connected Component
 * Check: hihoCoder 1184
 */

int n, m;
int head[MAXN], nxt[MAXM << 1], to[MAXM << 1],
↪ ed;
// Opposite edge exists, set head[] to -1.

int dfn[MAXN], low[MAXN], bcc_id[MAXN],
↪ bcc_cnt, stamp;
bool isbridge[MAXM << 1], vis[MAXN];

std::vector<int> bcc[MAXN];

void Tarjan(int now, int fa) {
    dfn[now] = low[now] = ++stamp;
    for (int i = head[now]; ~i; i = nxt[i]) {
        if (!dfn[to[i]]) {
            Tarjan(to[i], now);
            low[now] = std::min(low[now],
            ↪ low[to[i]]);
            if (low[to[i]] > dfn[now])
                isbridge[i] = isbridge[i ^ 1] =
                ↪ 1;
        }
        else if (dfn[to[i]] < dfn[now] && to[i]
        ↪ != fa)
            low[now] = std::min(low[now],
            ↪ dfn[to[i]]);
    }
}

void DFS(int now) {
    vis[now] = 1;
    bcc_id[now] = bcc_cnt;
    bcc[bcc_cnt].push_back(now);
    for (int i = head[now]; ~i; i = nxt[i]) {
        if (isbridge[i]) continue;
        if (!vis[to[i]]) DFS(to[i]);
    }
}

void EBCC() {
    memset(dfn, 0, sizeof dfn);
    memset(low, 0, sizeof low);
    memset(isbridge, 0, sizeof isbridge);
    memset(bcc_id, 0, sizeof bcc_id);
    bcc_cnt = stamp = 0;

    for (int i = 1; i <= n; ++i)
        if (!dfn[i]) Tarjan(i, 0);

    memset(vis, 0, sizeof vis);
    for (int i = 1; i <= n; ++i)
        if (!vis[i]) {
            ++bcc_cnt;
            DFS(i);
        }
}
```

4.3 有根树同构-Reshiram

```
const unsigned long long MAGIC = 4423;

unsigned long long magic[N];
std::pair<unsigned long long, int> hash[N];

void solve(int root) {
    magic[0] = 1;
    for (int i = 1; i <= n; ++i) {
        magic[i] = magic[i - 1] * MAGIC;
    }
    std::vector<int> queue;
    queue.push_back(root);
    for (int head = 0; head <
    ↪ (int)queue.size(); ++head) {
        int x = queue[head];
        for (int i = 0; i < (int)son[x].size();
        ↪ ++i) {
            int y = son[x][i];
            queue.push_back(y);
        }
    }
    for (int index = n - 1; index >= 0;
    ↪ --index) {
        int x = queue[index];
        hash[x] = std::make_pair(0, 0);

        std::vector<std::pair<unsigned long
        ↪ long, int> > value;
        for (int i = 0; i < (int)son[x].size();
        ↪ ++i) {
            int y = son[x][i];
            value.push_back(hash[y]);
        }
        std::sort(value.begin(), value.end());
```

```
        hash[x].first = hash[x].first *
        ↪ magic[1] + 37;
        hash[x].second++;
        for (int i = 0; i < (int)value.size();
        ↪ ++i) {
            hash[x].first = hash[x].first *
            ↪ magic[value[i].second] +
            ↪ value[i].first;
            hash[x].second += value[i].second;
        }
        hash[x].first = hash[x].first *
        ↪ magic[1] + 41;
        hash[x].second++;
    }
}
```

4.4 Hopcraft-Karp

```
int matchx[N], matchy[N], level[N];
vector<int> edge[N];
bool dfs(int x) {
    for (int i = 0; i < (int)edge[x].size();
    ↪ ++i) {
        int y = edge[x][i];
        int w = matchy[y];
        if (w == -1 || level[x] + 1 == level[w]
        ↪ && dfs(w)) {
            matchx[x] = y; matchy[y] = x;
            return true;
        }
    }
    level[x] = -1;
    return false;
}
int solve() {
    memset(matchx, -1, sizeof(*matchx) * n);
    memset(matchy, -1, sizeof(*matchy) * m);
    for (int ans = 0; ; ) {
        std::vector<int> q;
        for (int i = 0; i < n; ++i) {
            if (matchx[i] == -1) {
                level[i] = 0;
                q.push_back(i);
            } else level[i] = -1;
        }
        for (int head = 0; head <
        ↪ (int)q.size(); ++head) {
            int x = q[head];
            for (int i = 0; i <
            ↪ (int)edge[x].size(); ++i) {
                int y = edge[x][i];
                int w = matchy[y];
                if (w != -1 && level[w] < 0) {
                    level[w] = level[x] + 1;
                    q.push_back(w);
                }
            }
        }
        int delta = 0;
        for (int i = 0; i < n; ++i)
            if (matchx[i] == -1 && dfs(i))
            ↪ ++delta;
        if (delta == 0) return ans; else ans +=
        ↪ delta;
    }
}
```

4.5 ISAP

```
//Improved Shortest Augment Path Algorighm 最大流（ISAP
//By ysf
//注意 ISAP 适用于一般稀疏图，对于二分图或分层图情况 Dinic

//边的定义
//这里没有记录起点和反向边，因为反向边即为正向边 xor 1，起
struct edge{int to,cap,prev;}e[maxe<<1];

//全局变量和数组定义
int
↪ last[maxn],cnte=0,d[maxn],p[maxn],c[maxn],cur[maxn],
int n,m,s,t;//s,t 一定要开成全局变量

//重要!!!
//main 函数最前面一定要加上如下初始化
memset(last,-1,sizeof(last));

//加边函数 O(1)
//包装了加反向边的过程，方便调用
//需要调用 AddEdge
void addedge(int x,int y,int z){
    AddEdge(x,y,z);
    AddEdge(y,x,0);
}

//真·加边函数 O(1)
void AddEdge(int x,int y,int z){
    e[cnte].to=y;
    e[cnte].cap=z;
    e[cnte].prev=last[x];
    last[x]=cnte++;
}

//主过程 O(n^2 m)
//返回最大流的流量
//需要调用 bfs、augment
//注意这里的 n 是编号最大值，在这个值不为 n 的时候一定要开
//非递归
int ISAP(){
    bfs();
    memcpy(cur,last,sizeof(cur));
    int x=s,flow=0;
    while(d[s]<n){
        if(x==t){//如果走到了 t 就增广一次，并返回 s 重新
            flow+=augment();
            x=s;
        }
        bool ok=false;
        for(int &i=cur[x];~i;i=e[i].prev)
            if(e[i].cap&&d[x]==d[e[i].to]+1){
                p[e[i].to]=i;
                x=e[i].to;
                ok=true;
                break;
            }
        if(!ok){//修改距离标号
```

```
            int tmp=n-1;
            for(int i=last[x];~i;i=e[i].prev)
                if(e[i].cap)tmp=min(tmp,d[e[i].to]+1);
            if(!--c[d[x]])break;//gap 优化，一定要加上
            c[d[x]=tmp]++;
            cur[x]=last[x];
            if(x!=s)x=e[p[x]^1].to;
        }
    }
    return flow;
}

//bfs 函数 O(n+m)
//预处理到 t 的距离标号
//在测试数据组数较少时可以省略，把所有距离标号初始化为 0
void bfs(){
    memset(d,-1,sizeof(d));
    int head=0,tail=0;
    d[t]=0;
    q[tail++]=t;
    while(head!=tail){
        int x=q[head++];
        c[d[x]]++;
        for(int i=last[x];~i;i=e[i].prev)
            if(e[i^1].cap&&d[e[i].to]==-1){
                d[e[i].to]=d[x]+1;
                q[tail++]=e[i].to;
            }
    }
}

//augment 函数 O(n)
//沿增广路增广一次，返回增广的流量
int augment(){
    int a=(~0u)>>1;
    for(int
    ↪  x=t;x!=s;x=e[p[x]^1].to)a=min(a,e[p[x]].cap);
    for(int x=t;x!=s;x=e[p[x]^1].to){
        e[p[x]].cap-=a;
        e[p[x]^1].cap+=a;
    }
    return a;
}
```

4.6 zkw 费用流

```
int S, T, totFlow, totCost;

int dis[N], slack[N], visit[N];

int modlable () {
    int delta = INF;
    for (int i = 1; i <= T; i++) {
        if (!visit[i] && slack[i] < delta)
        ↪  delta = slack[i];
        slack[i] = INF;
    }
    if (delta == INF) return 1;
    for (int i = 1; i <= T; i++)
        if (visit[i]) dis[i] += delta;
    return 0;
}

int dfs (int x, int flow) {
    if (x == T) {
        totFlow += flow;
        totCost += flow * (dis[S] - dis[T]);
        return flow;
    }
    visit[x] = 1;
    int left = flow;
    for (int i = e.last[x]; ~i; i = e.succ[i])
        if (e.cap[i] > 0 && !visit[e.other[i]])
        ↪  {
            int y = e.other[i];
            if (dis[y] + e.cost[i] == dis[x]) {
                int delta = dfs (y, min (left,
                ↪  e.cap[i]));
                e.cap[i] -= delta;
                e.cap[i ^ 1] += delta;
                left -= delta;
                if (!left) { visit[x] = 0;
                ↪  return flow; }
            } else {
                slack[y] = min (slack[y],
                ↪  dis[y] + e.cost[i] -
                ↪  dis[x]);
            }
        }
    return flow - left;
}

pair <int, int> minCost () {
    totFlow = 0; totCost = 0;
    fill (dis + 1, dis + T + 1, 0);
    do {
        do {
            fill (visit + 1, visit + T + 1, 0);
        } while (dfs (S, INF));
} while (modlable ());
    return make_pair (totFlow, totCost);
}
```

4.7 无向图全局最小割

```
/*
 * Stoer Wagner õ¾ ¸ O(V ^ 3)
 * 1base, µ n, edge[MAXN][MAXN]
 * ·µ» Ï õ¾ ¸
 */

int StoerWagner() {
    static int v[MAXN], wage[MAXN];
    static bool vis[MAXN];

    for (int i = 1; i <= n; ++i) v[i] = i;

    int res = INF;

    for (int nn = n; nn > 1; --nn) {
        memset(vis, 0, sizeof(bool) * (nn +
        ↪  1));
        memset(wage, 0, sizeof(int) * (nn +
        ↪  1));

        int pre, last = 1; // vis[1] = 1;
```

```
        for (int i = 1; i < nn; ++i) {
            pre = last; last = 0;
            for (int j = 2; j <= nn; ++j) if
            ↪  (!vis[j]) {
                wage[j] += edge[v[pre]][v[j]];
                if (!last || wage[j] >
                ↪  wage[last]) last = j;
            }
            vis[last] = 1;
        }

        res = std::min(res, wage[last]);

        for (int i = 1; i <= nn; ++i) {
            edge[v[i]][v[pre]] +=
            ↪  edge[v[last]][v[i]];
            edge[v[pre]][v[i]] +=
            ↪  edge[v[last]][v[i]];
        }
        v[last] = v[nn];
    }
    return res;
}
```

4.8 KM

```
/*
 * Time: O(V ^ 3)
 * Condition: The perfect matching exists.
 * When finding minimum weight matching, change the weight to minus.
 */

bool e[MAXN][MAXN]; // whether the edge exists
// The array e[][] can be replaced by setting the absent edge's weight to -INF.
int val[MAXN][MAXN]; // the weight of the edge

int ex_A[MAXN], ex_B[MAXN];
bool vis_A[MAXN], vis_B[MAXN];
int match[MAXN];
int slack[MAXN];

bool DFS(int now) {
    vis_A[now] = 1;
    for (int i = 1; i <= n; ++i) {
        if (vis_B[i] || !e[now][i]) continue;

        int gap = ex_A[now] + ex_B[i] -
        ↪  val[now][i];

        if (gap == 0) {
            vis_B[i] = 1;
            if (!match[i] || DFS(match[i])) {
                match[i] = now;
                return 1;
            }
        }
        else slack[i] = std::min(slack[i],
        ↪  gap);
    }

    return 0;
}

int KM() {
    memset(match, 0, sizeof match);
    memset(ex_B, 0, sizeof ex_B);

    for (int i = 1; i <= n; ++i) {
        ex_A[i] = -INF;
        for (int j = 1; j <= n; ++j) if
        ↪  (e[i][j])
            ex_A[i] = std::max(ex_A[i],
            ↪  val[i][j]);
    }

    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) slack[j] =
        ↪  INF;
        while (1) {
            memset(vis_A, 0, sizeof vis_A);
            memset(vis_B, 0, sizeof vis_B);

            if (DFS(i)) break;

            int tmp = INF;
            for (int j = 1; j <= n; ++j) if
            ↪  (!vis_B[j])
                tmp = std::min(tmp, slack[j]);
            for (int j = 1; j <= n; ++j) {
                if (vis_A[j]) ex_A[j] -= tmp;
                if (vis_B[j]) ex_B[j] += tmp;
            }
        }
    }

    int res = 0;
    for (int i = 1; i <= n; ++i)
        res += val[match[i]][i];
    return res;
}
```

4.9 一般图最大权匹配

```
//maximum weight blossom,  change g[u][v].w to INF - g[u
//type of ans is long long
//replace all int to long long if weight of edge is long

struct WeightGraph {
    static const int INF = INT_MAX;
    static const int MAXN = 400;
    struct edge{
        int u, v, w;
        edge() {}
        edge(int u, int v, int w): u(u), v(v),
        ↪  w(w) {}
    };
    int n, n_x;
    edge g[MAXN * 2 + 1][MAXN * 2 + 1];
    int lab[MAXN * 2 + 1];
    int match[MAXN * 2 + 1], slack[MAXN * 2 +
    ↪  1], st[MAXN * 2 + 1], pa[MAXN * 2 + 1];
    int flower_from[MAXN * 2 + 1][MAXN+1],
    ↪  S[MAXN * 2 + 1], vis[MAXN * 2 + 1];
    vector<int> flower[MAXN * 2 + 1];
    queue<int> q;
```

```
inline int e_delta(const edge &e){
↪  // does not work inside blossoms
    return lab[e.u] + lab[e.v] -
    ↪  g[e.u][e.v].w * 2;
}
inline void update_slack(int u, int x){
    if(!slack[x] || e_delta(g[u][x]) <
    ↪  e_delta(g[slack[x]][x]))
        slack[x] = u;
}
inline void set_slack(int x){
    slack[x] = 0;
    for(int u = 1;u <= n; ++u)
        if(g[u][x].w > 0 && st[u] != x &&
        ↪  S[st[u]] == 0)
            update_slack(u, x);
}
void q_push(int x){
    if(x <= n)q.push(x);
    else for(size_t i = 0;i <
    ↪  flower[x].size(); i++)
        q_push(flower[x][i]);
}
inline void set_st(int x, int b){
    st[x]=b;
    if(x > n) for(size_t i = 0;i <
    ↪  flower[x].size(); ++i)
                set_st(flower[x][i], b);
}
inline int get_pr(int b, int xr){
    int pr = find(flower[b].begin(),
    ↪  flower[b].end(), xr) -
    ↪  flower[b].begin();
    if(pr % 2 == 1){
        reverse(flower[b].begin() + 1,
        ↪  flower[b].end());
        return (int)flower[b].size() - pr;
    } else return pr;
}
inline void set_match(int u, int v){
    match[u]=g[u][v].v;
    if(u > n){
        edge e=g[u][v];
        int xr = flower_from[u][e.u],
        ↪  pr=get_pr(u, xr);
        for(int i = 0;i < pr; ++i)
            set_match(flower[u][i],
            ↪  flower[u][i ^ 1]);
        set_match(xr, v);
        rotate(flower[u].begin(),
        ↪  flower[u].begin()+pr,
        ↪  flower[u].end());
    }
}
inline void augment(int u, int v){
    for(; ; ){
        int xnv=st[match[u]];
        set_match(u, v);
        if(!xnv)return;
        set_match(xnv, st[pa[xnv]]);
        u=st[pa[xnv]], v=xnv;
    }
}
inline int get_lca(int u, int v){
    static int t=0;
    for(++t; u || v; swap(u, v)){
        if(u == 0)continue;
        if(vis[u] == t)return u;
        vis[u] = t;
        u = st[match[u]];
        if(u) u = st[pa[u]];
    }
    return 0;
}
inline void add_blossom(int u, int lca, int
↪  v){
    int b = n + 1;
    while(b <= n_x && st[b]) ++b;
    if(b > n_x) ++n_x;
    lab[b] = 0, S[b] = 0;
    match[b] = match[lca];
    flower[b].clear();
    flower[b].push_back(lca);
    for(int x = u, y; x != lca; x =
    ↪  st[pa[y]]) {
        flower[b].push_back(x),
        flower[b].push_back(y =
        ↪  st[match[x]]),
        q_push(y);
    }
    reverse(flower[b].begin() + 1,
    ↪  flower[b].end());
    for(int x = v, y; x != lca; x =
    ↪  st[pa[y]]) {
        flower[b].push_back(x),
        flower[b].push_back(y =
        ↪  st[match[x]]),
        q_push(y);
    }
    set_st(b, b);
    for(int x = 1; x <= n_x; ++x) g[b][x].w
    ↪  = g[x][b].w = 0;
    for(int x = 1; x <= n; ++x)
    ↪  flower_from[b][x] = 0;
    for(size_t i = 0 ; i <
    ↪  flower[b].size(); ++i){
        int xs = flower[b][i];
        for(int x = 1; x <= n_x; ++x)
            if(g[b][x].w == 0 ||
            ↪  e_delta(g[xs][x]) <
            ↪  e_delta(g[b][x]))
                g[b][x] = g[xs][x], g[x][b]
                ↪  = g[x][xs];
        for(int x = 1;x <= n; ++x)
            if(flower_from[xs][x])
            ↪  flower_from[b][x] = xs;
    }
    set_slack(b);
}
inline void expand_blossom(int b){
↪  // S[b]  ==  1
    for(size_t i = 0; i < flower[b].size();
    ↪  ++i)
        set_st(flower[b][i], flower[b][i]);
    int xr = flower_from[b][g[b][pa[b]].u],
    ↪  pr = get_pr(b, xr);
```

```
    for(int i = 0; i < pr; i += 2){
        int xs = flower[b][i], xns =
        ↪ flower[b][i + 1];
        pa[xs] = g[xns][xs].u;
        S[xs] = 1, S[xns] = 0;
        slack[xs] = 0, set_slack(xns);
        q_push(xns);
    }
    S[xr] = 1, pa[xr] = pa[b];
    for(size_t i = pr + 1;i <
    ↪ flower[b].size(); ++i){
        int xs = flower[b][i];
        S[xs] = -1, set_slack(xs);
    }
    st[b] = 0;
}
inline bool on_found_edge(const edge &e){
    int u = st[e.u], v = st[e.v];
    if(S[v] == -1){
        pa[v] = e.u, S[v] = 1;
        int nu = st[match[v]];
        slack[v] = slack[nu] = 0;
        S[nu] = 0, q_push(nu);
    }else if(S[v] == 0){
        int lca = get_lca(u, v);
        if(!lca) return augment(u, v),
        ↪ augment(v, u), true;
        else add_blossom(u, lca, v);
    }
    return false;
}
inline bool matching(){
    memset(S + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) *
    ↪ n_x);
    q = queue<int>();
    for(int x = 1;x <= n_x; ++x)
        if(st[x] == x && !match[x])
        ↪ pa[x]=0, S[x]=0, q_push(x);
    if(q.empty())return false;
    for(;;){
        while(q.size()){
            int u = q.front();q.pop();
            if(S[st[u]] == 1)continue;
            for(int v = 1;v <= n; ++v)
                if(g[u][v].w > 0 && st[u]
                ↪ != st[v]){
                    if(e_delta(g[u][v]) ==
                    ↪ 0){
                        if(on_found_edge(g[u][v]))return
                        ↪ true;
                    }else update_slack(u,
                    ↪ st[v]);
                }
        }
        int d = INF;
        for(int b = n + 1; b <= n_x;++b)
            if(st[b] == b && S[b] == 1)d =
            ↪ min(d, lab[b]/2);
        for(int x = 1; x <= n_x; ++x)
            if(st[x] == x && slack[x]){
                if(S[x] == -1)d = min(d,
                ↪ e_delta(g[slack[x]][x]));
                else if(S[x] == 0)d =
                ↪ min(d,
                ↪ e_delta(g[slack[x]][x])/2);
            }
        for(int u = 1; u <= n; ++u){
            if(S[st[u]] == 0){
                if(lab[u] <= d)return 0;
                lab[u] -= d;
            }else if(S[st[u]] == 1)lab[u]
            ↪ += d;
        }
        for(int b = n+1; b <= n_x; ++b)
            if(st[b] == b){
                if(S[st[b]] == 0) lab[b] +=
                ↪ d * 2;
                else if(S[st[b]] == 1)
                ↪ lab[b] -= d * 2;
            }
        q=queue<int>();
        for(int x = 1; x <= n_x; ++x)
            if(st[x] == x && slack[x] &&
            ↪ st[slack[x]] != x &&
            ↪ e_delta(g[slack[x]][x]) ==
            ↪ 0)
                if(on_found_edge(g[slack[x]][x]))ret
                ↪ true;
        for(int b = n + 1; b <= n_x; ++b)
            if(st[b] == b && S[b] == 1 &&
            ↪ lab[b] ==
            ↪ 0)expand_blossom(b);
    }
    return false;
}
inline pair<long long, int> solve(){
    memset(match + 1, 0, sizeof(int) * n);
    n_x = n;
    int n_matches = 0;
    long long tot_weight = 0;
    for(int u = 0; u <= n; ++u) st[u] = u,
    ↪ flower[u].clear();
    int w_max = 0;
    for(int u = 1; u <= n; ++u)
        for(int v = 1; v <= n; ++v){
            flower_from[u][v] = (u == v ? u
            ↪ : 0);
            w_max = max(w_max, g[u][v].w);
        }
    for(int u = 1; u <= n; ++u) lab[u] =
    ↪ w_max;
    while(matching()) ++n_matches;
    for(int u = 1; u <= n; ++u)
        if(match[u] && match[u] < u)
            tot_weight += g[u][match[u]].w;
    return make_pair(tot_weight,
    ↪ n_matches);
}
inline void init(){
    for(int u = 1; u <= n; ++u)
        for(int v = 1; v <= n; ++v)
            g[u][v]=edge(u, v, 0);
}
};
```

4.10 最大团搜索

```
#include<iostream>
using namespace std;
int ans;
int num[1010];
int path[1010];
int a[1010][1010],n;
bool dfs(int *adj,int total,int cnt)
{
    int i,j,k;
    int t[1010];
    if(total==0)
    {
        if(ans<cnt)
        {
            ans=cnt;
            return 1;
        }
        return 0;
    }
    for(i=0;i<total;i++)
    {
        if(cnt+(total-i)<=ans)
            return 0;
        if(cnt+num[adj[i]]<=ans)
            return 0;
        for(k=0,j=i+1;j<total;j++)
        if(a[adj[i]][adj[j]])
            t[k++]=adj[j];
        if(dfs(t,k,cnt+1))
            return 1;
    }
    return 0;
}
int MaxClique()
{
    int i,j,k;
    int adj[1010];
    if(n<=0)
        return 0;
    ans=1;
    for(i=n-1;i>=0;i--)
    {
        for(k=0,j=i+1;j<n;j++)
        if(a[i][j])
            adj[k++]=j;
        dfs(adj,k,1);
        num[i]=ans;
    }
    return ans;
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    while(cin>>n)
    {
        if(n==0)
            break;
        for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            cin>>a[i][j];
        cout<<MaxClique()<<endl;
    }
    return 0;
}
```

4.11 极大团计数

```
#include<cstdio>
#include<cstring>
using namespace std;
const int N=130;
int ans,a[N][N],R[N][N],P[N][N],X[N][N];
bool Bron_Kerbosch(int d,int nr,int np,int nx)
{
    int i,j;
    if(np==0&&nx==0)
    {
        ans++;
        if(ans>1000)//
            return 1;
        return 0;
    }
    int u,max=0;
    u=P[d][1];
    for(i=1;i<=np;i++)
    {
        int cnt=0;
        for(j=1;j<=np;j++)
        {
            if(a[P[d][i]][P[d][j]])
                cnt++;
        }
        if(cnt>max)
        {
            max=cnt;
            u=P[d][i];
        }
    }
    for(i=1;i<=np;i++)
    {
        int v=P[d][i];
        if(a[v][u]) continue;
        for(j=1;j<=nr;j++)
            R[d+1][j]=R[d][j];
        R[d+1][nr+1]=v;
        int cnt1=0;
        for(j=1;j<=np;j++)
            if(P[d][j]&&a[P[d][j]][v])
                P[d+1][++cnt1]=P[d][j];
        int cnt2=0;
        for(j=1;j<=nx;j++)
            if(a[X[d][j]][v])
                X[d+1][++cnt2]=X[d][j];
        if(Bron_Kerbosch(d+1,nr+1,cnt1,cnt2))
            return 1;
        P[d][i]=0;
        X[d][++nx]=v;
    }
    return 0;
}
int main()
{
```

```
    int n,i,m,x,y;
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        memset(a,0,sizeof(a));
        while(m--)
        {
            scanf("%d%d",&x,&y);
            a[x][y]=a[y][x]=1;
        }
        ans=0;
        for(i=1;i<=n;i++)
            P[1][i]=i;
        Bron_Kerbosch(1,0,n,0);
        if(ans>1000)
            printf("Too many maximal sets of friends.\n");
        else
            printf("%d\n",ans);
    }
    return 0;
}
```

4.12 虚树-NewMeta

```
// 点集并的直径端点 $\subset$ 每个点集直径端点的并
// 可以用 dfs 序的 ST 表维护子树直径，建议使用 RMQLCA
void make(vi &poi) {
    //poi 要按 dfn 排序 需要清空边表 E 注意 V 无序
    //0 号点相当于一个虚拟的根，需要 lca(u,0)==0,h[0]=0
    V = {0}; vi st = {0};
    for (int v : poi) {
        V.pb(v);int w=lca(st.back(),v),
        sz=st.size();
        while (sz > 1 && h[st[sz - 2]] >= h[w])
            E[st[sz - 2]].pb(st[sz - 1]), sz
            --;
        st.resize(sz);
        if (st[sz - 1] != w)
            E[w].pb(st.back()), st.back() = w,
            V.pb(w);
        st.pb(v);
    }
    for (int i=1; i<st.size(); ++i)
    E[st[i-1]].pb(st[i]);
}
```

4.13 2-Sat

```
//清点清边要两倍
int stamp, comps, top;
int dfn[N], low[N], comp[N], stack[N];
void add(int x, int a, int y, int b) {
    edge[x << 1 | a].push_back(y << 1 | b);
}
void tarjan(int x) {
    dfn[x] = low[x] = ++stamp;
    stack[top++] = x;
    for (int i = 0; i < (int)edge[x].size();
    ++i) {
        int y = edge[x][i];
        if (!dfn[y]) {
            tarjan(y);
            low[x] = std::min(low[x], low[y]);
        } else if (!comp[y]) {
            low[x] = std::min(low[x], dfn[y]);
        }
    }
    if (low[x] == dfn[x]) {
        comps++;
        do {
            int y = stack[--top];
            comp[y] = comps;
        } while (stack[top] != x);
    }
}
bool solve() {
    int counter = n + n + 1;
    stamp = top = comps = 0;
    std::fill(dfn, dfn + counter, 0);
    std::fill(comp, comp + counter, 0);
    for (int i = 0; i < counter; ++i) {
        if (!dfn[i]) {
            tarjan(i);
        }
    }
    for (int i = 0; i < n; ++i) {
        if (comp[i << 1] == comp[i << 1 | 1]) {
            return false;
        }
        answer[i] = (comp[i << 1 | 1] < comp[i
        << 1]);
    }
    return true;
}
```

4.14 支配树

```
//solve(s, n, raw_g): s is the root and base accords to
//idom[x] will be x if x does not have a dominator,and u
struct dominator_tree {
    int base, dfn[N], sdom[N], idom[N], id[N],
    f[N], fa[N], smin[N], stamp;
    Graph *g;
    void predfs(int u) {
        id[dfn[u] = stamp++] = u;
        for (int i = g -> adj[u]; ~i; i = g ->
        nxt[i]) {
            int v = g -> v[i];
            if (dfn[v] < 0) f[v] = u,
            predfs(v);
        }
    }
    int getfa(int u) {
        if (fa[u] == u) return u;
        int ret = getfa(fa[u]);
        if (dfn[sdom[smin[fa[u]]]] <
        dfn[sdom[smin[u]]])
            smin[u] = smin[fa[u]];
        return fa[u] = ret;
    }
    void solve (int s, int n, Graph *raw_graph)
    {
        g = raw_graph;
        base = g -> base;
```

```
        memset(dfn + base, -1, sizeof(*dfn) *
        ↪ n);
        memset(idom + base, -1, sizeof(*idom) *
        ↪ n);
        static Graph pred, tmp;
        pred.init(base, n);
        for (int i = 0; i < n; ++i) {
            for (int p = g -> adj[i + base];
            ↪ ~p; p = g -> nxt[p])
                pred.ins(g -> v[p], i + base);
        }
        stamp = 0; tmp.init(base, n);
        ↪ predfs(s);
        for (int i = 0; i < stamp; ++i) {
            fa[id[i]] = smin[id[i]] = id[i];
        }
        for (int o = stamp - 1; o >= 0; --o) {
            int x = id[o];
            if (o) {
                sdom[x] = f[x];
                for (int i = pred.adj[x]; ~i; i
                ↪ = pred.nxt[i]) {
                    int p = pred.v[i];
                    if (dfn[p] < 0) continue;
                    if (dfn[p] > dfn[x]) {
                        getfa(p);
                        p = sdom[smin[p]];
                    }
                    if (dfn[sdom[x]] > dfn[p])
                    ↪ sdom[x] = p;
                }
                tmp.ins(sdom[x], x);
            }
            while (~tmp.adj[x]) {
                int y = tmp.v[tmp.adj[x]];
                tmp.adj[x] =
                ↪ tmp.nxt[tmp.adj[x]];
                getfa(y);
                if (x != sdom[smin[y]]) idom[y]
                ↪ = smin[y];
                else idom[y] = x;
            }
            for (int i = g -> adj[x]; ~i; i = g
            ↪ -> nxt[i])
                if (f[g -> v[i]] == x) fa[g ->
                ↪ v[i]] = x;
        }
        idom[s] = s;
        for (int i = 1; i < stamp; ++i) {
            int x = id[i];
            if (idom[x] != sdom[x]) idom[x] =
            ↪ idom[idom[x]];
        }
    }
};
```

4.15 哈密顿回路

```
\begin{lstlisting}
bool graph[N][N];
int n, l[N], r[N], next[N], last[N], s, t;
char buf[10010];
void cover(int x) { l[r[x]] = l[x]; r[l[x]] =
↪ r[x]; }
int adjacent(int x) {
    for (int i = r[0]; i <= n; i = r[i]) if
    ↪ (graph[x][i]) return i;
    return 0;
}
int main() {
    scanf("%d\n", &n);
    for (int i = 1; i <= n; ++i) {
        gets(buf);
        string str = buf;
        istringstream sin(str);
        int x;
        while (sin >> x) {
            graph[i][x] = true;
        }
        l[i] = i - 1;
        r[i] = i + 1;
    }
    for (int i = 2; i <= n; ++i)
        if (graph[1][i]) {
            s = 1;
            t = i;
            cover(s);
            cover(t);
            next[s] = t;
            break;
        }
    while (true) {
        int x;
        while (x = adjacent(s)) {
            next[x] = s;
            s = x;
            cover(s);
        }
        while (x = adjacent(t)) {
            next[t] = x;
            t = x;
            cover(t);
        }
        if (!graph[s][t]) {
            for (int i = s, j; i != t; i =
            ↪ next[i])
                if (graph[s][next[i]] &&
                ↪ graph[t][i]) {
                    for (j = s; j != i; j =
                    ↪ next[j])
                        last[next[j]] = j;
                    j = next[s];
                    next[s] = next[i];
                    next[t] = i;
                    t = j;
                    for (j = i; j != s; j =
                    ↪ last[j])
                        next[j] = last[j];
                    break;
                }
        }
        next[t] = s;
        if (r[0] > n)
            break;
        for (int i = s; i != t; i = next[i])
```

```
            if (adjacent(i)) {
                s = next[i];
                t = i;
                next[t] = 0;
                break;
            }
    }
    for (int i = s; ; i = next[i]) {
        if (i == 1) {
            printf("%d", i);
            for (int j = next[i]; j != i; j =
            ↪ next[j])
                printf(" %d", j);
            printf(" %d\n", i);
            break;
        }
        if (i == t)
            break;
    }
}
\end{lstlisting}
```

4.16 曼哈顿最小生成树

```
\begin{lstlisting}
/*
`只需要考虑每个点的 pi/4*k -- pi/4*(k+1) 的区间内的第一个点，这样只有 4n 条边。
*/
const int maxn = 100000+5;
const int Inf = 1000000005;
struct TreeEdge
{
    int x,y,z;
    void make( int _x,int _y,int _z ) { x=_x;
    ↪ y=_y; z=_z; }
} data[maxn*4];

inline bool operator < ( const TreeEdge&
↪ x,const TreeEdge& y ){
    return x.z<y.z;
}

int
↪ x[maxn],y[maxn],px[maxn],py[maxn],id[maxn],tree[maxn],node[maxn],val[maxn],fa[maxn];
int n;
inline bool compare1( const int a,const int b )
↪ { return x[a]<x[b]; }
inline bool compare2( const int a,const int b )
↪ { return y[a]<y[b]; }
inline bool compare3( const int a,const int b )
↪ { return (y[a]-x[a]<y[b]-x[b] ||
↪ y[a]-x[a]==y[b]-x[b] && y[a]>y[b]); }
inline bool compare4( const int a,const int b )
↪ { return (y[a]-x[a]>y[b]-x[b] ||
↪ y[a]-x[a]==y[b]-x[b] && x[a]>x[b]); }
inline bool compare5( const int a,const int b )
↪ { return (x[a]+y[a]>x[b]+y[b] ||
↪ x[a]+y[a]==x[b]+y[b] && x[a]<x[b]); }
inline bool compare6( const int a,const int b )
↪ { return (x[a]+y[a]<x[b]+y[b] ||
↪ x[a]+y[a]==x[b]+y[b] && y[a]>y[b]); }
void Change_X()
{
    for(int i=0;i<n;++i) val[i]=x[i];
    for(int i=0;i<n;++i) id[i]=i;
    sort(id,id+n,compare1);
    int cntM=1, last=val[id[0]]; px[id[0]]=1;
    for(int i=1;i<n;++i)
    {
        if(val[id[i]]>last)
        ↪ ++cntM,last=val[id[i]];
        px[id[i]]=cntM;
    }
}
void Change_Y()
{
    for(int i=0;i<n;++i) val[i]=y[i];
    for(int i=0;i<n;++i) id[i]=i;
    sort(id,id+n,compare2);
    int cntM=1, last=val[id[0]]; py[id[0]]=1;
    for(int i=1;i<n;++i)
    {
        if(val[id[i]]>last)
        ↪ ++cntM,last=val[id[i]];
        py[id[i]]=cntM;
    }
}
inline int absValue( int x ) { return
↪ (x<0)?-x:x; }
inline int Cost( int a,int b ) { return
↪ absValue(x[a]-x[b])+absValue(y[a]-y[b]); }
int find( int x ) { return
↪ (fa[x]==x)?x:(fa[x]=find(fa[x])); }
int main()
{
//    freopen("input.txt", "r", stdin);
//    freopen("output.txt", "w", stdout);

    int test=0;
    while( scanf("%d",&n)!=EOF && n )
    {
        for(int i=0;i<n;++i)
        ↪ scanf("%d%d",x+i,y+i);
        Change_X();
        Change_Y();

        int cntE = 0;
        for(int i=0;i<n;++i) id[i]=i;
        sort(id,id+n,compare3);
        for(int i=1;i<=n;++i)
        ↪ tree[i]=Inf,node[i]=-1;
        for(int i=0;i<n;++i)
        {
            int Min=Inf, Tnode=-1;
            for(int k=py[id[i]];k<=n;k+=k&(-k))
            ↪ if(tree[k]<Min)
            ↪ Min=tree[k],Tnode=node[k];
            if(Tnode>=0)
            ↪ data[cntE++].make(id[i],Tnode,Cost(id[i]
            int tmp=x[id[i]]+y[id[i]];
            for(int k=py[id[i]];k;k-=k&(-k))
            ↪ if(tmp<tree[k])
            ↪ tree[k]=tmp,node[k]=id[i];
        }
        sort(id,id+n,compare4);
```

```
        for(int i=1;i<=n;++i)
        ↪  tree[i]=Inf,node[i]=-1;
        for(int i=0;i<n;++i)
        {
            int Min=Inf, Tnode=-1;
            for(int k=px[id[i]];k<=n;k+=k&(-k))
            ↪  if(tree[k]<Min)
            ↪  Min=tree[k],Tnode=node[k];
            if(Tnode>=0)
            ↪  data[cntE++].make(id[i],Tnode,Cost(id[i],Tnode));
            int tmp=x[id[i]]+y[id[i]];
            for(int k=px[id[i]];k;k-=k&(-k))
            ↪  if(tmp<tree[k])
            ↪  tree[k]=tmp,node[k]=id[i];
        }
        sort(id,id+n,compare5);
        for(int i=1;i<=n;++i)
        ↪  tree[i]=Inf,node[i]=-1;
        for(int i=0;i<n;++i)
        {
            int Min=Inf, Tnode=-1;
            for(int k=px[id[i]];k;k-=k&(-k))
            ↪  if(tree[k]<Min)
            ↪  Min=tree[k],Tnode=node[k];
            if(Tnode>=0)
            ↪  data[cntE++].make(id[i],Tnode,Cost(id[i],Tnode));
            int tmp=-x[id[i]]+y[id[i]];
            for(int k=px[id[i]];k<=n;k+=k&(-k))
            ↪  if(tmp<tree[k])
            ↪  tree[k]=tmp,node[k]=id[i];
        }
        sort(id,id+n,compare6);
        for(int i=1;i<=n;++i)
        ↪  tree[i]=Inf,node[i]=-1;
        for(int i=0;i<n;++i)
        {
            int Min=Inf, Tnode=-1;
            for(int k=py[id[i]];k<=n;k+=k&(-k))
            ↪  if(tree[k]<Min)
            ↪  Min=tree[k],Tnode=node[k];
            if(Tnode>=0)
            ↪  data[cntE++].make(id[i],Tnode,Cost(id[i],Tnode));
            int tmp=-x[id[i]]+y[id[i]];
            for(int k=py[id[i]];k;k-=k&(-k))
            ↪  if(tmp<tree[k])
            ↪  tree[k]=tmp,node[k]=id[i];
        }

        long long Ans = 0;
        sort(data,data+cntE);
        for(int i=0;i<n;++i) fa[i]=i;
        for(int i=0;i<cntE;++i)
        ↪  if(find(data[i].x)!=find(data[i].y))
        {
            Ans += data[i].z;
            fa[fa[data[i].x]]=fa[data[i].y];
        }

        cout<<"Case "<<++test<<": "<<"Total Weight = "<<Ans<<endl;
    }
    return 0;
}
    \end{lstlisting}
```

4.17 弦图

1. 团数 ≤ 色数, 弦图团数 = 色数
2. 设 $next(v)$ 表示 $N(v)$ 中最前的点. 令 w* 表示所有满足 $A \in B$ 的 w 中最后的一个点, 判断 $v \cup N(v)$ 是否为极大团, 只需判断是否存在一个 w, 满足 $Next(w) = v$ 且 $|N(v)| + 1 \leq |N(w)|$ 即可.
3. 最小染色: 完美消除序列从后往前依次给每个点染色, 给每个点染上可以染的最小的颜色
4. 最大独立集: 完美消除序列从前往后能选就选
5. 弦图最大独立集数 = 最小团覆盖数, 最小团覆盖: 设最大独立集为 $\{p_1, p_2, \ldots, p_t\}$, 则 $\{p_1 \cup N(p_1), \ldots, p_t \cup N(p_t)\}$ 为最小团覆盖

4.18 图同构 hash

$$F_t(i) = (F_{t-1}(i) \times A + \sum_{i \to j} F_{t-1}(j) \times B + \sum_{j \to i} F_{t-1}(j) \times C + D \times (i =$$

枚举点 a , 迭代 K 次后求得的就是 a 点所对应的 hash 值.

其中 K , A , B , C , D , P 为 hash 参数, 可自选

5 字符串

5.1 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;
}
```

5.2 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]]]--]=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||y[sa[i]]!=y[sa[i-1]]||y[sa[i]+k]!=y[sa[i-1]+k])
                ++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-1]]-1);
        if(rank[i]==1)continue;
        int j=sa[rank[i]-1];
        while(max(i,j)+height[rank[i]]<=n&&a[i+height[rank[i]]]==a[j+height[rank[i]]])
            ++height[rank[i]];
    }
}
```

5.3 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[MaxPoint],r
    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[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;
}
```

5.4 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; last = last->parent)
                    last->to[x] = nq;

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

        return np;
    }
}

int main() {


    return 0;
}
```

5.5 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;
}
```

5.6 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
// 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
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));
    }
}
```

5.7 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
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;
}}
```

6 杂项

6.1 fread 读入优化

```
namespace Scanner {
    const int L = (1 << 15) + 5;
    char buffer[L], *S, *T;

    __advance __inline char GetChar() {
        if (S == T) {
            T = (S = buffer) + fread(buffer, 1,
            ↪ L, stdin);
            if (S == T)
                return -1;
        }
        return *S++;
    }

    template <class Type>
    __advance __inline void Scan(Type &x) {
        register char ch; x = 0;
        for (ch = GetChar(); ~ch && (ch < '0'
        ↪ || ch > '9'); ch = GetChar());
        for (; ch >= '0' && ch <= '9'; ch =
        ↪ GetChar()) x = x * 10 + ch - '0';
    }
} using Scanner::Scan;
```

6.2 真正释放 STL 内存

```cpp
template <typename T>
__inline void clear(T& container) {
    container.clear(); // 或者删除了一堆元素
    T(container).swap(container);
}
```

6.3 梅森旋转算法

```cpp
#include <random>

int main() {
    std::mt19937 g(seed);  // std::mt19937_64
    std::cout << g() << std::endl;
}
```

6.4 蔡勒公式

```cpp
int solve(int year, int month, int day) {
    int answer;
    if (month == 1 || month == 2) {
        month += 12;
        year--;
    }
    if ((year < 1752) || (year == 1752 && month
    ↪  < 9) ||
        (year == 1752 && month == 9 && day <
        ↪  3)) {
        answer = (day + 2 * month + 3 * (month
        ↪  + 1) / 5 + year + year / 4 + 5) %
        ↪  7;
    } else {
        answer = (day + 2 * month + 3 * (month
        ↪  + 1) / 5 + year + year / 4
                - year / 100 + year / 400) % 7;
    }
    return answer;
}
```

6.5 开栈

```cpp
register char *_sp __asm__("rsp");
int main() {
    const int size = 400 << 20;//400MB
    static char *sys, *mine(new char[size] +
    ↪  size - 4096);
    sys = _sp; _sp = mine; _main(); _sp = sys;
}
```

6.6 Size 为 k 的子集

```cpp
void solve(int n, int k) {
    for (int comb = (1 << k) - 1; comb < (1 <<
    ↪  n); ) {
        // ...
        int x = comb & -comb, y = comb + x;
        comb = (((comb & ~y) / x) >> 1) | y;
    }
}
```

6.7 长方体表面两点最短距离

```cpp
int r;
void turn(int i, int j, int x, int y, int z,int
↪  x0, int y0, int L, int W, int H) {
    if (z==0) { int R = x*x+y*y; if (R<r) r=R;
    } else {
        if(i>=0 && i< 2) turn(i+1, j, x0+L+z,
        ↪  y, x0+L-x, x0+L, y0, H, W, L);
        if(j>=0 && j< 2) turn(i, j+1, x,
        ↪  y0+W+z, y0+W-y, x0, y0+W, L, H, W);
        if(i<=0 && i>-2) turn(i-1, j, x0-z, y,
        ↪  x-x0, x0-H, y0, H, W, L);
        if(j<=0 && j>-2) turn(i, j-1, x, y0-z,
        ↪  y-y0, x0, y0-H, L, H, W);
    }
}
int main(){
    int L, H, W, x1, y1, z1, x2, y2, z2;
    cin >> L >> W >> H >> x1 >> y1 >> z1 >> x2
    ↪  >> y2 >> z2;
    if (z1!=0 && z1!=H) if (y1==0 || y1==W)
         swap(y1,z1), std::swap(y2,z2),
         ↪  std::swap(W,H);
    else swap(x1,z1), std::swap(x2,z2),
    ↪  std::swap(L,H);
    if (z1==H) z1=0, z2=H-z2;
    r=0x3fffffff;
    turn(0,0,x2-x1,y2-y1,z2,-x1,-y1,L,W,H);
    cout<<r<<endl;
}
```

6.8 32-bit/64-bit 随机素数

32-bit	64-bit
73550053	1249292846855685773
148898719	1701750434419805569
189560747	3605499878424114901
459874703	5648316673387803781
1202316001	6125342570814357977
1431183547	6215155308775851301
1438011109	6294606778040623451
1538762023	6347330550446020547
1557944263	7429632924303725207
1981315913	8524720079480389849

6.9 NTT 素数及其原根

Prime	Primitive root
1053818881	7
1051721729	6
1045430273	3
1012924417	5
1007681537	3
1000000000622593	5

6.10 伯努利数-Reshiram

1. 初始化：$B_0(n) = 1$

2. 递推公式:

$$B_m(n) = n^m - \sum_{k=0}^{m-1} mk \frac{B_k(n)}{m-k+1}$$

3. 应用:

$$\sum_{k=1}^{n} k^m = \frac{1}{m+1} \sum_{k=0}^{m} m+1 k n^{m+1-k}$$

6.11 博弈游戏-Reshiram

6.11.1 巴什博奕

1. 只有一堆 n 个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取 m 个。最后取光者得胜。

2. 显然，如果 $n = m + 1$，那么由于一次最多只能取 m 个，所以，无论先取者拿走多少个，后取者都能够一次拿走剩余的物品，后者取胜。因此我们发现了如何取胜的法则：如果 $n = m + 1\, r + s$，(r 为任意自然数，$s \le m$), 那么先取者要拿走 s 个物品，如果后取者拿走 $k(k \le m)$ 个，那么先取者再拿走 $m+1-k$ 个，结果剩下 $(m+1)(r-1)$ 个，以后保持这样的取法，那么先取者肯定获胜。总之，要保持给对手留下 $(m+1)$ 的倍数，就能最后获胜。

6.11.2 威佐夫博弈

1. 有两堆各若干个物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

2. 判断一个局势 (a, b) 为奇异局势 (必败态) 的方法:

$$a_k = [k(1+\sqrt{5})/2]\ b_k = a_k + k$$

6.11.3 阶梯博奕

1. 博弈在一列阶梯上进行，每个阶梯上放着自然数个点，两个人进行阶梯博弈，每一步则是将一个阶梯上的若干个点（至少一个）移到前面去，最后没有点可以移动的人输。

2. 解决方法：把所有奇数阶梯看成 N 堆石子，做 NIM。(把石子从奇数堆移动到偶数堆可以理解为拿走石子，就相当于几个奇数堆的石子在做 Nim)

6.11.4 图上删边游戏

6.11.5 链的删边游戏

1. 游戏规则: 对于一条链，其中一个端点是根，两人轮流删边，脱离根的部分也算被删去，最后没边可删的人输。

2. 做法: $sg[i] = n - dist(i) - 1$ (其中 n 表示总点数，$dist(i)$ 表示离根的距离)

6.11.6 树的删边游戏

1. 游戏规则: 对于一棵有根树，两人轮流删边，脱离根的部分也算被删去，没边可删的人输。

2. 做法: 叶子结点的 $sg = 0$，其他节点的 sg 等于儿子结点的 $sg + 1$ 的异或和。

6.11.7 局部连通图的删边游戏

1. 游戏规则：在一个局部连通图上，两人轮流删边，脱离根的部分也算被删去，没边可删的人输。局部连通图的构图规则是，在一棵基础树上加边得到，所有形成的环保证不共用边，且只与基础树有一个公共点。

2. 做法：去掉所有的偶环，将所有的奇环变为长度为 1 的链，然后做树的删边游戏。

6.12 Formulas

6.13 Arithmetic Function

$$\sigma_k(n) = \sum_{d|n} d^k = \prod_{i=1}^{\omega(n)} \frac{p_i^{(a_i+1)k} - 1}{p_i^k - 1}$$

$$J_k(n) = n^k \prod_{p|n} (1 - \frac{1}{p^k})$$

$J_k(n)$ is the number of k-tuples of positive integers all less than or equal to n that form a coprime $(k+1)$-tuple together with n.

$$\sum_{\delta|n} J_k(\delta) = n^k$$

$$\sum_{\delta|n} \delta^s J_r(\delta) J_s(\frac{n}{\delta}) = J_{r+s}(n)$$

$\sum_{\delta|n} \varphi(\delta) d(\frac{n}{\delta}) = \sigma(n),\ \sum_{\delta|n} |\mu(\delta)| = 2^{\omega(n)}$
$\sum_{\delta|n} 2^{\omega(\delta)} = d(n^2),\ \sum_{\delta|n} d(\delta^2) = d^2(n)$
$\sum_{\delta|n} d(\frac{n}{\delta}) 2^{\omega(\delta)} = d^2(n),\ \sum_{\delta|n} \frac{\mu(\delta)}{\delta} = \frac{\varphi(n)}{n}$
$\sum_{\delta|n} \frac{\mu(\delta)}{\varphi(\delta)} = d(n),\ \sum_{\delta|n} \frac{\mu^2(\delta)}{\varphi(\delta)} = \frac{n}{\varphi(n)}$

$$n | \varphi(a^n - 1)$$

$$\sum_{1 \le k \le n \atop \gcd(k,n)=1} f(\gcd(k-1, n)) = \varphi(n) \sum_{d|n} \frac{(\mu * f)(d)}{\varphi(d)}$$

$$\varphi(\mathrm{lcm}(m,n))\varphi(\gcd(m,n)) = \varphi(m)\varphi(n)$$

$$\sum_{\delta|n} d^3(\delta) = (\sum_{\delta|n} d(\delta))^2$$

$$d(uv) = \sum_{\delta|\gcd(u,v)} \mu(\delta) d(\frac{u}{\delta}) d(\frac{v}{\delta})$$

$$\sigma_k(u)\sigma_k(v) = \sum_{\delta|\gcd(u,v)} \delta^k \sigma_k(\frac{uv}{\delta^2})$$

$$\mu(n) = \sum_{k=1}^{n} [\gcd(k,n) = 1] \cos 2\pi \frac{k}{n}$$

$$\varphi(n) = \sum_{k=1}^{n} [\gcd(k,n) = 1] = \sum_{k=1}^{n} \gcd(k,n) \cos 2\pi \frac{k}{n}$$

$$\begin{cases} S(n) = \sum_{k=1}^{n} (f*g)(k) \sum_{k=1}^{n} S(\lfloor \frac{n}{k} \rfloor) = \sum_{i=1}^{n} f(i) \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} (g*1)(j) \\ S(n) = \sum_{k=1}^{n} (f \cdot g)(k), g completely multiplicative \sum_{k=1}^{n} S(\lfloor \frac{n}{k} \rfloor) g(k) = \sum_{k=1}^{n} (f*1)(k) g(k) \end{cases}$$

6.14 Binomial Coefficients

$$\binom{n}{k} = (-1)^k \binom{k-n-1}{k}$$

$$\sum_{k \le n} \binom{r+k}{k} = \binom{r+n+1}{n}$$

$$\sum_{k=0}^{n} \binom{k}{m} = \binom{n+1}{m+1}$$

$$\sqrt{1+z} = 1 + \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k \times 2^{2k-1}} \binom{2k-2}{k-1} z^k$$

$$\sum_{k=0}^{r} \binom{r-k}{m} \binom{s+k}{n} = \binom{r+s+1}{m+n+1}$$

$$C_{n,m} = \binom{n+m}{m} - \binom{n+m}{m-1}, n \ge m$$

$$\binom{n}{k} \equiv [n \& k = k] \pmod 2$$

6.15 Fibonacci Numbers

$$F(z) = \frac{z}{1-z-z^2}$$

$$f_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}, \phi = \frac{1+\sqrt{5}}{2}, \hat{\phi} = \frac{1-\sqrt{5}}{2}$$

$$\sum_{k=1}^{n} f_k = f_{n+2} - 1$$

$$\sum_{k=1}^{n} f_k^2 = f_n f_{n+1}$$

$$\sum_{k=0}^{n} f_k f_{n-k} = \frac{1}{5}(n-1) f_n + \frac{2}{5} n f_{n-1}$$

$$f_n^2 + (-1)^n = f_{n+1} f_{n-1}$$

$$f_{n+k} = f_n f_{k+1} + f_{n-1} f_k$$

$$f_{2n+1} = f_n^2 + f_{n+1}^2$$

$$(-1)^k f_{n-k} = f_n f_{k-1} - f_{n-1} f_k$$

$$Modulo f_n, f_{mn+r} \equiv \{ f_r, m \bmod 4 = 0; (-1)^{r+1} f_{n-r}, m \bmod 4 = 1; (-1)^n f_r, m \bmod 4 = 2; (-1)^{r+1+n} f_{n-r}, m \bmod 4 = 3.$$

6.16 Stirling Cycle Numbers

n+1 $\left[_{k=n\left[{n \atop k}\right]+\left[{n \atop k-1}\right]}, \ \left[{n+1 \atop 2}\right]=n! H_n x^{\underline{n}}=\sum_k \left[{n \atop k}\right](-1)^{n-k} x^k, \ x^{\overline{n}}=\sum_k \left[{n \atop k}\right] x^k\right]$

6.17 Stirling Subset Numbers

$$\left\{ {n+1 \atop k} \right\} = k \left\{ {n \atop k} \right\} + \left\{ {n \atop k-1} \right\}$$

$$x^n = \sum_k \left\{ {n \atop k} \right\} x^{\underline{k}} = \sum_k \left\{ {n \atop k} \right\} (-1)^{n-k} x^{\overline{k}}$$

$$m! \left\{ {n \atop m} \right\} = \sum_k \binom{m}{k} k^n (-1)^{m-k}$$

6.18 Eulerian Numbers

$$\left\langle {n \atop k} \right\rangle = (k+1) \left\langle {n-1 \atop k} \right\rangle + (n-k) \left\langle {n-1 \atop k-1} \right\rangle$$

$$x^n = \sum_k \left\langle {n \atop k} \right\rangle \binom{x+k}{n}$$

$$\left\langle {n \atop m} \right\rangle = \sum_{k=0}^{m} \binom{n+1}{k} (m+1-k)^n (-1)^k$$

6.19 Harmonic Numbers

$$\sum_{k=1}^{n} H_k = (n+1) H_n - n$$

$$\sum_{k=1}^{n} k H_k = \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}$$

$$\sum_{k=1}^{n} \binom{k}{m} H_k = \binom{n+1}{m+1} (H_{n+1} - \frac{1}{m+1})$$

6.20 Pentagonal Number Theorem

$$\prod_{n=1}^{\infty} (1-x^n) = \sum_{n=-\infty}^{\infty} (-1)^k x^{k(3k-1)/2}$$

$$p(n) = p(n-1) + p(n-2) - p(n-5) - p(n-7) + \cdots$$

$$f(n,k) = p(n) - p(n-k) - p(n-2k) + p(n-5k) + p(n-7k) - \cdots$$

6.21 Bell Numbers

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

$$B_{p^m+n} \equiv m B_n + B_{n+1} \pmod p$$

6.22 Bernoulli Numbers

$$B_n = 1 - \sum_{k=0}^{n-1} \binom{n}{k} \frac{B_k}{n-k+1}$$

$$G(x) = \sum_{k=0}^{\infty} \frac{B_k}{k!} x^k = \frac{1}{\sum_{k=0}^{\infty} \frac{x^k}{(k+1)!}}$$

$$S_m(n) = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} B_k n^{m-k+1}$$

6.23 Tetrahedron Volume

$$V=\frac{\sqrt{4u^2v^2w^2-\sum_{cyc}u^2(v^2+w^2-U^2)^2+\prod_{cyc}(v^2+w^2-U^2)}}{12}$$

6.24 BEST Thoerem

Counting the number of different Eulerian circuits in directed graphs.

$$\mathrm{ec}(G)=t_w(G)\prod_{v\in V}(\deg(v)-1)!$$

When calculating $t_w(G)$ for directed multigraphs, the entry $q_{i,j}$ for distinct i and j equals $-m$, where m is the number of edges from i to j, and the entry $q_{i,i}$ equals the indegree of i minus the number of loops at i. It is a property of Eulerian graphs that $\mathrm{tv}(G)=\mathrm{tw}(G)$ for every two vertices v and w in a connected Eulerian graph G.

6.25 重心

半径为 r , 圆心角为 θ 的扇形重心与圆心的距离为 $\frac{4r\sin(\theta/2)}{3\theta}$

半径为 r , 圆心角为 θ 的圆弧重心与圆心的距离为 $\frac{4r\sin^3(\theta/2)}{3(\theta-\sin(\theta))}$

6.26 Others

$$S_j=\sum_{k=1}^{n}x_k^j$$

$$h_m=\sum_{1\le j_1<\cdots<j_m\le n}x_{j_1}\cdots x_{j_m}$$

$$H_m=\sum_{1\le j_1\le\cdots\le j_m\le n}x_{j_1}\cdots x_{j_m}$$

$$h_n=\frac{1}{n}\sum_{k=1}^{n}(-1)^{k+1}S_kh_{n-k}$$

$$H_n=\frac{1}{n}\sum_{k=1}^{n}S_kH_{n-k}$$

$$\sum_{k=0}^{n}kc^k=\frac{nc^{n+2}-(n+1)c^{n+1}+c}{(c-1)^2}$$

$$n!=\sqrt{2\pi n}(\frac{n}{e})^n(1+\frac{1}{12n}+\frac{1}{288n^2}+O(\frac{1}{n^3}))$$

$$\max\{x_a-x_b,y_a-y_b,z_a-z_b\}-\min\{x_a-x_b,y_a-y_b,z_a-z_b\}=\frac{1}{2}\sum_{cyc}|(x_a-y_a)-(x_b-y_b)|$$

$$(a+b)(b+c)(c+a)=\frac{(a+b+c)^3-a^3-b^3-c^3}{3}$$

[pages=1-2]./hint/Integral.pdf

6.27 Java

```java
import java.io.*;
import java.util.*;
import java.math.*;
public class Main {
    public static void main(String[] args)
    ↪ {
        InputStream inputStream =
        ↪ System.in;
        OutputStream outputStream =
        ↪ System.out;
        InputReader in = new
        ↪ InputReader(inputStream);
        PrintWriter out = new
        ↪ PrintWriter(outputStream);
    }
}
public static class edge implements
↪ Comparable<edge>{
        public int u,v,w;
        public int compareTo(edge e){
                return w-e.w;
        }
}
public static class cmp implements
↪ Comparator<edge>{
        public int compare(edge a,edge b){
                if(a.w<b.w)return 1;
                if(a.w>b.w)return -1;
                return 0;
        }
}
class InputReader {
    public BufferedReader reader;
    public StringTokenizer tokenizer;

    public InputReader(InputStream stream)
    ↪ {
        reader = new BufferedReader(new
        ↪ InputStreamReader(stream),
        ↪ 32768);
        tokenizer = null;
    }
    public String next() {
        while (tokenizer == null ||
        ↪ !tokenizer.hasMoreTokens()) {
            try {
                tokenizer = new
                ↪ StringTokenizer(reader.readLine
            } catch (IOException e) {
                throw new
                ↪ RuntimeException(e);
            }
        }
```

```
        return tokenizer.nextToken();
    }

    public int nextInt() {
        return Integer.parseInt(next());
    }

    public long nextLong() {
        return Long.parseLong(next());
    }
}
```

[pages=1-6]./hint/biginteger.pdf [pages=1-7]./hint/BigDecimal.pdf
[pages=1-5]./hint/treemap.pdf