Algorithm Library

palayutm

September 25, 2018

Contents

1 计算几何 **3**
1.1 二维基础 . . . 3
1.2 半平面交 . . . 6
1.3 二维最小圆覆盖 . . . 7
1.4 凸包 . . . 7
1.5 凸包游戏 . . . 8
1.6 圆并 . . . 10
1.7 最远点对 . . . 13
1.8 根轴 . . . 13

2 字符串 **15**
2.1 manacher . . . 15
2.2 后缀数组 . . . 15
2.3 后缀自动机 . . . 15
2.4 广义后缀自动机 . . . 16
2.5 回文自动机 . . . 17
2.6 Lyndon Word Decomposition NewMeta . . . 18
2.7 EXKMP NewMeta . . . 18

3 数据结构 **18**
3.1 Link-Cut-Tree . . . 18
3.2 KDTree . . . 19
3.3 莫队上树 . . . 21

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); }
```

```cpp
};

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
                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]];
        }
}
```

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[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
}
*/
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;
```

```cpp
        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

```cpp
//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));
    }
}
```

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;
}}
```

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)
```

```cpp
            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.second);
                        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 图同构 hash

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

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

4.4 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++;
    }
}
```

4.5 Hopcraft-Karp

4.6 ISAP

```
//Improved Shortest Augment Path Algorighm
↪ 最大流（ISAP 版本） O(n^2 m)
//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 比较优，稠密图则 HLPP 更优

//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，起点即为反向边的终点
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],q[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
↪ 的时候一定要开个变量记录下来并修改代码
//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 重新找
        ↪ 增广路
            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].t
                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].pre
                if(e[i].cap)tmp=min
            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]==
                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]]
    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;
}
```

4.7 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);
}
```

4.8 无向图全局最小割

```
/*
 * Stoer Wagner ō¾   ¸O(V ^ 3)
 * 1base, µ  n,    edge[MAXN][MAXN]
 *  • µ» Ïō¾   ¸
 */

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[
                                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;
}
```

4.9 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;
}
```

4.10 一般图最大权匹配

```
//maximum weight blossom, change g[u][v].w to INF - g[u][v].w when minimum weight blossom is needed
//type of ans is long long
//replace all int to long long if weight of edge is long 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(flow 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] 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][
            ↪ flower_from[b][
            ↪ = 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(
                        ↪
                        ↪


                        }el
                        ↪
                        ↪
                }
}
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
                else
                ↪   if(S[x]
                ↪   ==
                ↪   0)d
                ↪   =
                ↪   min(d,
                ↪   e_delta
        }
for(int u = 1; u <=
↪   n; ++u){
        if(S[st[u]]
        ↪   == 0){
                if(lab[u]
                ↪   <=
                ↪   d)retur
                ↪   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]))return
                    ↪ 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]].
        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);
    }
};
```

4.11 曼哈顿最小生成树

/* ‘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 条无向边。‘ */ const int maxn = 100000+5; const int Inf = 1000000005; struct TreeEdge, int x,y,z; void make(int $_x, int_y, int_z)x =_x; y =_y; z =$
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[max
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$_X()for(inti=0;i<n;++i)val[i]=x[i];for(inti=0;i<n;++i)id[i]=i;sort(id,id+n,compare1);$

int test=0; while(scanf(" for(int i=0;i<n;++i) scanf("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],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]; 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;

4.12 哈密顿回路

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("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("for (int j = next[i]; j != i; j = next[j]) printf(" printf(" break; if (i == t) break;

4.13 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;
}
```

4.14 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;
}
```

4.15 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]);
}
```

4.16 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;
}
```

4.17 弦图

1. 团数 $\le$ 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 $|N(v)| + 1 \le |N(w)|$ 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 $\{p_1, p_2, \dots, p_t\}$, 则 $\{p_1 \cup N(p_1), \dots, p_t \cup N(p_t)\}$ 为最小团覆盖

4.18 支配树

```
//solve(s, n, raw_g): s is the root and
↪  base accords to base of raw_g
//idom[x] will be x if x does not have a
↪  dominator,and will be -1 if x is not
↪  reachable from s.
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])
                ↪  {
                    get
                    p
                    ↪
                    ↪
                }
                if
                ↪  (dfn[sd
                ↪  >
                ↪  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
            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]];
```

```
                }
        }
};
```