# Template Library

NEW CODE!!

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

```cpp
#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

```cpp
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

```cpp
#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;
                }
```

```cpp
            }
        }
        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

```cpp
/*
 * 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

```cpp
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 Algorigghm 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» Ïõ¾  ¸
 */

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;
```

```cpp
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=_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 $\leq$ 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]);
```

```cpp
    }
    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

```cpp
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

```cpp
#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);
```

```cpp
    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

```cpp
#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

```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;
}
```

## 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 数学

## 6.1 质数

### 6.1.1 miller-rabin

```
const int BASE[12] = {2, 3, 5, 7, 11, 13, 17,
↪  19, 23, 29, 31, 37};
bool check(long long n,int base) {
    long long n2=n-1,res;
    int s=0;
    while(n2%2==0) n2>>=1,s++;
    res=pw(base,n2,n);
    if((res==1)||(res==n-1)) return 1;
    while(s--) {
        res=mul(res,res,n);
        if(res==n-1) return 1;
    }
    return 0; // n is not a strong pseudo prime
}
bool isprime(const long long &n) {
    if(n==2)
        return true;
    if(n<2 || n%2==0)
        return false;
    for(int i=0;i<12&&BASE[i]<n;i++){
        if(!check(n,BASE[i]))
            return false;
    }
```

```
    return true;
}
```

### 6.1.2 pollard-rho

```
LL prho(LL n,LL c){
    LL i=1,k=2,x=rand()%(n-1)+1,y=x;
    while(1){
        i++;x=(x*x%n+c)%n;
        LL d=__gcd((y-x+n)%n,n);
        if(d>1&&d<n)return d;
        if(y==x)return n;
        if(i==k)y=x,k<<=1;
    }
}
void factor(LL n,vector<LL>&fat){
    if(n==1)return;
    if(isprime(n)){
        fat.push_back(n);
        return;
    }LL p=n;
    while(p>=n)p=prho(p,rand()%(n-1)+1);
    factor(p,fat);
    factor(n/p,fat);
}
```

### 6.1.3 求原根

```
//51Nod - 1135
#include <iostream>
#include <string.h>
#include <algorithm>
#include <stdio.h>
#include <math.h>
#include <bitset>

using namespace std;
typedef long long LL;

const int N = 1000010;

bitset<N> prime;
int p[N],pri[N];
int k,cnt;

void isprime()
{
    prime.set();
    for(int i=2; i<N; i++)
    {
        if(prime[i])
        {
            p[k++] = i;
            for(int j=i+i; j<N; j+=i)
                prime[j] = false;
        }
    }
}

void Divide(int n)
{
    cnt = 0;
    int t = (int)sqrt(1.0*n);
    for(int i=0; p[i]<=t; i++)
    {
        if(n%p[i]==0)
        {
            pri[cnt++] = p[i];
            while(n%p[i]==0) n /= p[i];
        }
    }
    if(n > 1)
        pri[cnt++] = n;
}

LL quick_mod(LL a,LL b,LL m)
{
    LL ans = 1;
    a %= m;
    while(b)
    {
        if(b&1)
        {
            ans = ans * a % m;
            b--;
        }
        b >>= 1;
        a = a * a % m;
    }
    return ans;
}

int main()
{
    int P;
    isprime();
    while(cin>>P)
    {
        Divide(P-1);
        for(int g=2; g<P; g++)
        {
            bool flag = true;
            for(int i=0; i<cnt; i++)
            {
                int t = (P - 1) / pri[i];
                if(quick_mod(g,t,P) == 1)
                {
                    flag = false;
                    break;
                }
            }
            if(flag)
            {
                int root = g;
                cout<<root<<endl;
                break;
            }
        }
    }
    return 0;
}
```

## 6.2 多项式

### 6.2.1 快速傅里叶变换

```
#include<iostream>
#include<cstdio>
#include<cmath>
using namespace std;
const double eps=1e-8;
const double PI=acos(-1.0);
struct Complex
{
    double real,image;
    Complex(double _real,double _image)
    {
        real=_real;
        image=_image;
    }
    Complex(){real=0;image=0;}
};

Complex operator + (const Complex &c1, const Complex &c2)
{
    return Complex(c1.real + c2.real, c1.image + c2.image);
}

Complex operator - (const Complex &c1, const Complex &c2)
{
    return Complex(c1.real - c2.real, c1.image - c2.image);
}

Complex operator * (const Complex &c1, const Complex &c2)
{
    return Complex(c1.real*c2.real - c1.image*c2.image, c1.real*c2.image + c1.image*c2.real);
}

int rev(int id,int len)
{
    int ret=0;
    for(int i=0;(1<<i)<len;i++)
    {
        ret<<=1;
        if(id&(1<<i))
            ret|=1;
    }
    return ret;
}
Complex* IterativeFFT(Complex* a,int len,int DFT)
{
    Complex* A=new Complex[len];
    for(int i=0;i<len;i++)
        A[rev(i,len)]=a[i];
    for(int s=1;(1<<s)<=len;s++)
    {
        int m=(1<<s);
        Complex wm=Complex(cos(DFT*2*PI/m),sin(DFT*2*PI/m));
        for(int k=0;k<len;k+=m)
        {
            Complex w=Complex(1,0);
            for(int j=0;j<(m>>1);j++)
            {
                Complex t=w*A[k+j+(m>>1)];
                Complex u=A[k+j];
                A[k+j]=u+t;
                A[k+j+(m>>1)]=u-t;
                w=w*wm;
            }
        }
    }
    if(DFT==-1)
    for(int i=0;i<len;i++)
    {
        A[i].real/=len;
        A[i].image/=len;
    }
    return A;
}
char s[101010],t[101010];
Complex a[202020],b[202020],c[202020];
int pr[202020];
int main()
{
    int len;
    scanf("%d",&len);
    scanf("%s",s);
    scanf("%s",t);
    for(int i=0;i<len;i++)
        a[i]=Complex(s[len-i-1]-'0',0);
    for(int i=0;i<len;i++)
        b[i]=Complex(t[len-i-1]-'0',0);
    int tmp=1;
    while(tmp<=len)
        tmp*=2;
    len=tmp*2;
    Complex* aa=IterativeFFT(a,len,1);
    Complex* bb=IterativeFFT(b,len,1);
    for(int i=0;i<len;i++)
        c[i]=aa[i]*bb[i];
    Complex* ans=IterativeFFT(c,len,-1);
    for(int i=0;i<len;i++)
        pr[i]=round(ans[i].real);
    for(int i=0;i<=len;i++)
    {
        pr[i+1]+=pr[i]/10;
        pr[i]%=10;
    }
    bool flag=0;
    for(int i=len-1;i>=0;i--)
    {
        if(pr[i]>0)
            flag=1;
        if(flag)
            printf("%d",pr[i]);
    }
    printf("\n");
    return 0;
}
```

### 6.2.2 快速数论变换

```
#include<bits/stdc++.h>
using namespace std;
const int mod=1004535809;
int Pow(int a,int b)
{
    int ret=1;
    while(b)
    {
        if(b&1)
            ret=1ll*ret*a%mod;
        a=1ll*a*a%mod;
        b/=2;
    }
    return ret;
}

const int MAXN=(1<<18)+10;

struct NumberTheoreticTransform{
    int n,rev[MAXN];
    int g;
    void ini(int lim)
    {
        g=3;
        n=1;
        int k=0;
        while(n<=lim)
        {
            n<<=1;
            k++;
        }
        for(int i=0;i<n;i++)
            rev[i]=(rev[i>>1]>>1)|((i&1)<<(k-1));
    }
    void dft(int *a,int flag)
    {
        for(int i=0;i<n;i++)
        if(i<rev[i])
            swap(a[i],a[rev[i]]);
        for(int l=2;l<=n;l<<=1)
        {
            int m=l>>1;
            int
            ↪ wn=Pow(g,flag==1?((mod-1)/l):(mod-1-(mod-1)/l));
            for(int *p=a;p!=a+n;p+=l)
            {
                int w=1;
                for(int k=0;k<m;k++)
                {
                    int t=1ll*w*p[k+m]%mod;
                    p[k+m]=(p[k]-t+mod)%mod;
                    p[k]=(p[k]+t)%mod;
                    w=1ll*w*wn%mod;
                }
            }
        }
        if(flag==-1)
        {
            long long inv=Pow(n,mod-2);
            for(int i=0;i<n;i++)
                a[i]=1ll*a[i]*inv%mod;
        }
    }
    void mul(int *a,int *b,int m)
    {
        ini(m);
        dft(a,1);
        dft(b,1);
        for(int i=0;i<n;i++)
            a[i]=1ll*a[i]*b[i]%mod;
        dft(a,-1);
    }
}f;
int a[404040],b[404040];
int main()
{
    int n1,n2;
    scanf("%d%d",&n1,&n2);
    for(int i=0;i<=n1;i++)
        scanf("%d",&a[i]);
    for(int i=0;i<=n2;i++)
        scanf("%d",&b[i]);
    int m=n1+n2;
    f.mul(a,b,m);
    for(int i=0;i<=m;i++)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```

### 6.2.3 快速沃尔什变换

```
//Fast Walsh-Hadamard Transform 快速沃尔什变换 O(n\log n
//By ysf
//通过题目：COGS 上几道板子题

//注意 FWT 常数比较小，这点与 FFT/NTT 不同
//以下代码均以模质数情况为例，其中 n 为变换长度， tp 表示正

//按位或版本
void FWT_or(int *A,int n,int tp){
    for(int k=2;k<=n;k<<=1)
        for(int i=0;i<n;i+=k)
            for(int j=0;j<(k>>1);j++){
                if(tp>0)A[i+j+(k>>1)]=(A[i+j+(k>>1)]+A[i
                else
                ↪ A[i+j+(k>>1)]=(A[i+j+(k>>1)]-A[i+j]+
            }
}

//按位与版本
void FWT_and(int *A,int n,int tp){
    for(int k=2;k<=n;k<<=1)
        for(int i=0;i<n;i+=k)
            for(int j=0;j<(k>>1);j++){
                if(tp>0)A[i+j]=(A[i+j]+A[i+j+(k>>1)])%p;
                else
                ↪ A[i+j]=(A[i+j]-A[i+j+(k>>1)]+p)%p;
            }
}

//按位异或版本
void FWT_xor(int *A,int n,int tp){
    for(int k=2;k<=n;k<<=1)
```

```cpp
        for(int i=0;i<n;i+=k)
            for(int j=0;j<(k>>1);j++){
                int a=A[i+j],b=A[i+j+(k>>1)];
                A[i+j]=(a+b)%p;
                A[i+j+(k>>1)]=(a-b+p)%p;
            }
    if(tp<0){
        int inv=qpow(n%p,p-2);//n 的逆元，在不取模时需要用每层除以 2 代替
        for(int i=0;i<n;i++)A[i]=A[i]*inv%p;
    }
}
```

### 6.2.4 线性递推求第 n 项

Given $a_0, a_1, \cdots, a_{m-1}$

$a_n = c_0 * a_{n-m} + \cdots + c_{m-1} * a_0$

$a_0$ *is the nth element,* $\cdots, a_{m-1}$ *is the* $n+m-1$*th element*

```cpp
void linear_recurrence(long long n, int m, int a[], int c[], int p) {
    long long v[M] = {1 % p}, u[M << 1], msk = !!n;
    for(long long i(n); i > 1; i >>= 1) {
        msk <<= 1;
    }
    for(long long x(0); msk; msk >>= 1, x <<= 1) {
        fill_n(u, m << 1, 0);
        int b(!!(n & msk));
        x |= b;
        if(x < m) {
            u[x] = 1 % p;
        }else {
            for(int i(0); i < m; i++) {
                for(int j(0), t(i + b); j < m; j++, t++) {
                    u[t] = (u[t] + v[i] * v[j]) % p;
                }
            }
            for(int i((m << 1) - 1); i >= m; i--) {
                for(int j(0), t(i - m); j < m; j++, t++) {
                    u[t] = (u[t] + c[j] * u[i]) % p;
                }
            }
        }
        copy(u, u + m, v);
    }
    //a[n] = v[0] * a[0] + v[1] * a[1] + ... + v[m - 1] * a[m - 1].
    for(int i(m); i < 2 * m; i++) {
        a[i] = 0;
        for(int j(0); j < m; j++) {
            a[i] = (a[i] + (long long)c[j] * a[i + j - m]) % p;
        }
    }
    for(int j(0); j < m; j++) {
        b[j] = 0;
        for(int i(0); i < m; i++) {
            b[j] = (b[j] + v[i] * a[i + j]) % p;
        }
    }
    for(int j(0); j < m; j++) {
        a[j] = b[j];
    }
```

## 6.3 膜

### 6.3.1 O(n) 求逆元

```cpp
//Mutiply Inversation 预处理乘法逆元 O(n)
//By ysf
//要求 p 为质数 (?)

inv[0]=inv[1]=1;
for(int i=2;i<=n;i++)
    inv[i]=(long long)(p-(p/i))*inv[p%i]%p;//p 为模数
//$i^{-1}\equiv-\left\lfloor\frac p i\right\rfloor\times
//i^-1 = -(p/i) * (p%i)^-1
```

### 6.3.2 非互质 CRT

```cpp
inline void fix(LL &x, LL y) {
    x = (x % y + y) % y;
}
bool solve(int n, std::pair<LL, LL> a[],
                  std::pair<LL, LL> &ans) {
    ans = std::make_pair(1, 1);
    for (int i = 0; i < n; ++i) {
        LL num, y;
        euclid(ans.second, a[i].second, num, y);
        LL divisor = std::__gcd(ans.second, a[i].second);
        if ((a[i].first - ans.first) % divisor) {
            return false;
        }
        num *= (a[i].first - ans.first) / divisor;
        fix(num, a[i].second);
        ans.first += ans.second * num;
        ans.second *= a[i].second / divisor;
        fix(ans.first, ans.second);
    }
    return true;
}
```

### 6.3.3 CRT

```cpp
// 51nod 1079
#include<iostream>
using namespace std;
int gcd(int x,int y)
{
    if(x==0)
        return y;
    if(y==0)
```

```cpp
        return x;
    return gcd(y,x%y);
}
long long exgcd(long long a,long long b,long long &x,long long &y)
{
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    long long ans=exgcd(b,a%b,x,y);
    long long temp=x;
    x=y;
    y=temp-a/b*y;
    return ans;
}
void fix(long long &x,long long &y)
{
    x%=y;
    if(x<0)
        x+=y;
}
bool solve(int n, std::pair<long long, long long> input[],std::pair<long long, long long> &output)
{
    output = std::make_pair(1, 1);
    for(int i = 0; i < n; ++i)
    {
        long long number, useless;
        exgcd(output.second, input[i].second, number, useless);
        long long divisor = gcd(output.second, input[i].second);
        if((input[i].first - output.first) % divisor)
        {
            return false;
        }
        number *= (input[i].first - output.first) / divisor;
        fix(number,input[i].second);
        output.first += output.second * number;
        output.second *= input[i].second / divisor;
        fix(output.first, output.second);
    }
    return true;
}
pair<long long,long long> input[101010],output;
int main()
{
    int n;
    cin>>n;
    for(int i=0;i<n;i++)
        cin>>input[i].second>>input[i].first;
    solve(n,input,output);
    cout<<output.first<<endl;
    return 0;
}
```

### 6.3.4 FactorialMod-NewMeta

```cpp
// Complexity is $ O(pq + q^2 \log_2 p) $
int calcsgn(LL x) { return (x % 8 <= 2 || x % 8 == 7) ? 1 : -1; } // 计算 mod 4 的答案
// $ 1 \leq n \leq 1000, p^q \leq 1000$ 测试通过, fastpo
LL f(LL n, LL p, LL q) {
    LL mod(fastpo(p, q, INT64_MAX));
    LL phi(mod / p * (p - 1));
    static LL pre[1111111];
    pre[0] = 1;
    for(int i(1); i <= p * (q + 1); i++) pre[i] = i % p == 0 ? pre[i - 1] : pre[i - 1] * i % mod;
    LL res(1);
    LL u(n / p), v(n % p);
    for(int j(1); j < q; j++) {
        __int128 alpha(1);
        for(int i(j + 1); i < q; i++) alpha = alpha * (u - i) / (j - i);
        for(int i(j - 1); i >= 0; i--) alpha = alpha * (u - i) / (j - i);
        alpha = (alpha % phi + phi) % phi;
        res = res * fastpo(pre[j * p + v] % mod * fastpo(pre[v], phi - 1, mod) % mod * fastpo(pre[j * p], phi - 1, mod) % mod, alpha, mod) % mod;
    }
    int sgn(calcsgn(u * 2));
    int r(max((LL)1, q / 2 + 1));
    for(int j(1); j <= r; j++) {
        __int128 beta(1);
        for(int i(j + 1); i <= r; i++) beta = beta * (u - i) / (j - i);
        for(int i(j - 1); i > -j; i--) beta = beta * (u - i) / (j - i);
        beta *= u + j;
        for(int i(-j - 1); i >= -r; i--) beta = beta * (u - i) / (j - i);
        assert(beta % (j + u) == 0);
        beta /= u + j;
        beta = (beta % phi + phi) % phi;
        if(beta % 2)
            sgn *= calcsgn(j * 2);
        res = res * fastpo(pre[j * p], beta, mod) % mod;
    }
    if(p == 2) res = (res * sgn + mod) % mod;
    res = res * pre[v] % mod;
    return res;
}
```

## 6.4 积分

### 6.4.1 自适应辛普森

```cpp
double area(const double &left, const double &right) {
    double mid = (left + right) / 2;
    return (right - left) * (calc(left) + 4 * calc(mid) + calc(right)) / 6;
}
```

```cpp
double simpson(const double &left, const double &right,
               const double &eps, const double &area_sum) {
    double mid = (left + right) / 2;
    double area_left = area(left, mid);
    double area_right = area(mid, right);
    double area_total = area_left + area_right;
    if (std::abs(area_total - area_sum) < 15 * eps) {
        return area_total + (area_total - area_sum) / 15;
    }
    return simpson(left, mid, eps / 2, area_left)
         + simpson(mid, right, eps / 2, area_right);
}

double simpson(const double &left, const double &right, const double &eps) {
    return simpson(left, right, eps, area(left, right));
}
```

### 6.4.2 Romberg-Dreadnought

```cpp
template<class T>
double romberg(const T&f,double a,double b,double eps=1e-8){
    std::vector<double>t; double h=b-a,last,curr; int k=1,i=1;
    t.push_back(h*(f(a)+f(b))/2); // 梯形
    do{ last=t.back(); curr=0; double x=a+h/2;
        for(int j=0;j<k;++j) curr+=f(x),x+=h;
        curr=(t[0]+h*curr)/2; double k1=4.0/3.0,k2=1.0/3.0;
        for(int j=0;j<i;j++){ double temp=k1*curr-k2*t[j];
            t[j]=curr; curr=temp; k2/=4*k1-k2; k1=k2+1; // 防止溢出
        } t.push_back(curr); k*=2; h/=2; i++;
    } while(std::fabs(last-curr)>eps);
    return t.back();
}
```

## 6.5 代数

### 6.5.1 ExGCD

```cpp
LL exgcd(LL a,LL b,LL &x,LL &y){
    if(!b){
        x=1;y=0;return a;
    }else{
        LL d=exgcd(b,a%b,x,y);
        LL t=x;x=y;y=t-a/b*y;
        return d;
    }
}
```

### 6.5.2 ExBSGS

```cpp
/*
 * EX_BSGS
 * a^x = b (mod p)
 * p may not be a prime
 */

ll qpow(ll a, ll x, ll Mod) {
    ll res = 1;
    for (; x; x >>= 1) {
        if (x & 1) res = res * a % Mod;
        a = a * a % Mod;
    }
    return res;
}

std::unordered_map<int, int> mp;

ll exbsgs(ll a, ll b, ll p) {
    if (b == 1) return 0;
    ll t, d = 1, k = 0;
    while ((t = std::__gcd(a, p)) != 1) {
        if (b % t) return -1;
        ++k, b /= t, p /= t, d = d * (a / t) % p;
        if (b == d) return k;
    }
    mp.clear();
    ll m = std::ceil(std::sqrt(p));
    ll a_m = qpow(a, m, p);
    ll mul = b;
    for (ll j = 1; j <= m; ++j) {
        mul = mul * a % p;
        mp[mul] = j;
    }
    for (ll i = 1; i <= m; ++i) {
        d = d * a_m % p;
        if (mp.count(d)) return i * m - mp[d] + k;
    }
    return -1;
}
```

### 6.5.3 线段下整点

```cpp
// \sum_{i=0}^{n-1}\lfloor\frac{a+bi}{m}\rfloor
// n, m, a, b > 0
LL solve(LL n,LL a,LL b,LL m){
    if(b==0) return n*(a/m);
    if(a>=m) return n*(a/m)+solve(n,a%m,b,m);
    if(b>=m) return (n-1)*n/2*(b/m)+solve(n,a,b%m,m);
    return solve((a+b*n)/m,(a+b*n)%m,m,b);
}
```

### 6.5.4 解一元三次方程

```cpp
double a(p[3]), b(p[2]), c(p[1]), d(p[0]);
double k(b / a), m(c / a), n(d / a);
double p(-k * k / 3. + m);
double q(2. * k * k * k / 27 - k * m / 3. + n);
```

```
Complex omega[3] = {Complex(1, 0), Complex(-0.5, 0.5 * sqrt(3)), Complex(-0.5, -0.5 * sqrt(3))};
Complex r1, r2;
double delta(q * q / 4 + p * p * p / 27);
if (delta > 0) {
    r1 = cubrt(-q / 2. + sqrt(delta));
    r2 = cubrt(-q / 2. - sqrt(delta));
} else {
    r1 = pow(-q / 2. + pow(Complex(delta), 0.5), 1. / 3);
    r2 = pow(-q / 2. - pow(Complex(delta), 0.5), 1. / 3);
}
for(int _(0); _ < 3; _++) {
    Complex x = -k / 3. + r1 * omega[_ * 1] + r2 * omega[_ * 2 % 3];
}
```

### 6.5.5 黑盒子代数-NewMeta

```
// Berlekamp-Massey Algorithm
// Complexity: O(n^2)
// Requirement: const MOD, inverse(int)
// Input: vector<int> - the first elements of the sequence
// Output: vector<int> - the recursive equation of the given sequence
// Example:  In: {1, 1, 2, 3}  Out: {1, 1000000006, 1000000006} (MOD = 1e9+7)
struct Poly {
    vector<int> a;
    Poly() { a.clear(); }
    Poly(vector<int> &a): a(a) {}
    int length() const { return a.size(); }
    Poly move(int d) {
        vector<int> na(d, 0);
        na.insert(na.end(), a.begin(), a.end());
        return Poly(na);
    }
    int calc(vector<int> &d, int pos) {
        int ret = 0;
        for (int i = 0; i < (int)a.size(); ++i) {
            if ((ret += (long long)d[pos - i] * a[i] % MOD) >= MOD) {
                ret -= MOD; }}
        return ret;
    }
    Poly operator - (const Poly &b) {
        vector<int> na(max(this->length(), b.length()));
        for (int i = 0; i < (int)na.size(); ++i) {
            int aa = i < this->length() ? this->a[i] : 0,
            bb = i < b.length() ? b.a[i] : 0;
            na[i] = (aa + MOD - bb) % MOD;
        }
        return Poly(na);
    }
};
Poly operator * (const int &c, const Poly &p) {
    vector<int> na(p.length());
    for (int i = 0; i < (int)na.size(); ++i) {
        na[i] = (long long)c * p.a[i] % MOD;
    }
    return na;
}
vector<int> solve(vector<int> a) {
    int n = a.size();
    Poly s, b;
    s.a.push_back(1), b.a.push_back(1);
    for (int i = 1, j = 0, ld = a[0]; i < n; ++i) {
        int d = s.calc(a, i);
        if (d) {
            if ((s.length() - 1) * 2 <= i) {
                Poly ob = b;
                b = s;
                s = s - (long long)d * inverse(ld) % MOD * ob.move(i - j);
                j = i;
                ld = d;
            } else {
                s = s - (long long)d * inverse(ld) % MOD * b.move(i - j);
            }
        }
    } //Caution: s.a might be shorter than expected
    return s.a;
}
/*
 如果要求行列式，只需要求出来特征多项式即可，
 而这个方法可以解出来最小多项式，如果最小多项式里面有 x 的
 否则我们让原矩阵乘以一个随机的对角阵，那么高概率最小多项式
 特征多项式从而容易求得行列式 .
 */
```

## 6.6 其他

### 6.6.1 O(1) 快速乘

```
//Quick Multiplication O(1) 快速乘
//By ysf
//在两数直接相乘会爆 long long 时才有必要使用
//常数比直接 long long 乘法 + 取模大很多，非必要时不建议使

long long mul(long long a,long long b,long long p){
    a%=p;b%=p;
    return ((a*b-p*(long long)((long double)a/p*b+0.5))%p+p)%p;
}
```

### 6.6.2 Pell 方程-Dreadnought

```
ULL A,B,p[maxn],q[maxn],a[maxn],g[maxn],h[maxn];
int main() {
    for (int test=1, n;scanf("%d",&n) && n;++test) {
        printf("Case %d: ",test);
```

```cpp
        if (fabs(sqrt(n)-floor(sqrt(n)+1e-7))<=1e-7) {
            int a=(int)(floor(sqrt(n)+1e-7)); printf("%d %d\n",a,1);
        } else {
            // 求 $x^2-ny^2=1$ 的最小正整数根，n 不是完全平方数
            p[1]=q[0]=h[1]=1;p[0]=q[1]=g[1]=0;
            a[2]=(int)(floor(sqrt(n)+1e-7));
            for (int i=2;i;++i) {
                g[i]=-g[i-1]+a[i]*h[i-1]; h[i]=(n-sqr(g[i]))/h[i-1];
                a[i+1]=(g[i]+a[2])/h[i]; p[i]=a[i]*p[i-1]+p[i-2];
                q[i]=a[i]*q[i-1]+q[i-2];
                if (sqr((ULL)(p[i]))-n*sqr((ULL)(q[i]))==1){
                    A=p[i];B=q[i];break;
                }
            }
            cout << A << ' ' << B <<endl;
        }
    }
}
```

### 6.6.3 单纯形

```cpp
namespace LP{
    const int maxn=233;
    double a[maxn][maxn];
    int Ans[maxn],pt[maxn];
    int n,m;
    void pivot(int l,int i){
        double t;
        swap(Ans[l+n],Ans[i]);
        t=-a[l][i];
        a[l][i]=-1;
        for(int j=0;j<=n;j++)a[l][j]/=t;
        for(int j=0;j<=m;j++){
            if(a[j][i]&&j!=l){
                t=a[j][i];
                a[j][i]=0;
                for(int k=0;k<=n;k++)a[j][k]+=t*a[l][k];
            }
        }
    }
    vector<double> solve(vector<vector<double> >A,vector<double>B,vector<double>C){
        n=C.size();
        m=B.size();
        for(int i=0;i<C.size();i++)
            a[0][i+1]=C[i];
        for(int i=0;i<B.size();i++)
            a[i+1][0]=B[i];

        for(int i=0;i<m;i++)
            for(int j=0;j<n;j++)
                a[i+1][j+1]=-A[i][j];

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

        double t;
        for(;;){
            int l=0;t=-eps;
            for(int j=1;j<=m;j++)if(a[j][0]<t)t=a[l=j][0];
            if(!l)break;
            int i=0;
            for(int j=1;j<=n;j++)if(a[l][j]>eps){i=j;break;}
            if(!i){
                puts("Infeasible");
                return vector<double>();
            }
            pivot(l,i);
        }
        for(;;){
            int i=0;t=eps;
            for(int j=1;j<=n;j++)if(a[0][j]>t)t=a[0][i=j];
            if(!i)break;
            int l=0;
            t=1e30;
            for(int j=1;j<=m;j++)if(a[j][i]<-eps){
                double tmp;
                tmp=-a[j][0]/a[j][i];
                if(t>tmp)t=tmp,l=j;
            }
            if(!l){
                puts("Unbounded");
                return vector<double>();
            }
            pivot(l,i);
        }
        vector<double>x;
        for(int i=n+1;i<=n+m;i++)pt[Ans[i]]=i-n;
        for(int i=1;i<=n;i++)x.push_back(pt[i]?a[pt[i]][0]:0
        return x;
    }
}
```

### 6.6.4 二次剩余-Dreadnought

```cpp
void calcH(int &t, int &h, const int p) {
    int tmp = p - 1; for (t = 0; (tmp & 1) == 0; tmp /= 2) t++; h = tmp;
}
// solve equation x^2 mod p = a
bool solve(int a, int p, int &x, int &y) {
    srand(19920225);
    if (p == 2) { x = y = 1; return true; }
    int p2 = p / 2, tmp = power(a, p2, p);
    if (tmp == p - 1) return false;
    if ((p + 1) % 4 == 0) {
        x = power(a, (p + 1) / 4, p); y = p - x; return true;
    } else {
        int t, h, b, pb; calcH(t, h, p);
        if (t >= 2) {
            do {b = rand() % (p - 2) + 2;
            } while (power(b, p / 2, p) != p - 1);
```

```
            pb = power(b, h, p);
        } int s = power(a, h / 2, p);
        for (int step = 2; step <= t; step++) {
            int ss = (((long long)(s * s) % p) * a) % p;
            for (int i = 0; i < t - step; i++) ss = ((long long)ss * ss) % p;
            if (ss + 1 == p) s = (s * pb) % p; pb = ((long long)pb * pb) % p;
        } x = ((long long)s * a) % p; y = p - x;
    } return true;
}
```

### 6.6.5 线性同余不等式-NewMeta

```
// Find the minimal non-negtive solutions for $ l \leq d \cdot x \bmod m \leq r $
// $0 \leq d, l, r < m; l \leq r, O(\log n)$
ll cal(ll m, ll d, ll l, ll r) {
    if (l == 0) return 0;
    if (d == 0) return MXL; // 无解
    if (d * 2 > m) return cal(m, m - d, m - r, m - l);
    if ((l - 1) / d < r / d) return (l - 1) / d + 1;
    ll k = cal(d, (-m % d + d) % d, l % d, r % d);
    return k == MXL ? MXL : (k * m + l - 1) / d + 1; // 无解 2
}

// return all x satisfying l1<=x<=r1 and l2<=(x*mul+add)%LIM<=r2
// here LIM = 2^32 so we use UI instead of "%".
// $O(\log p + #solutions)$
struct Jump {
    UI val, step;
    Jump(UI val, UI step) : val(val), step(step) { }
    Jump operator + (const Jump & b) const {
        return Jump(val + b.val, step + b.step); }
    Jump operator - (const Jump & b) const {
        return Jump(val - b.val, step + b.step);
}};
inline Jump operator * (UI x, const Jump & a) {
    return Jump(x * a.val, x * a.step);
}
vector<UI> solve(UI l1, UI r1, UI l2, UI r2, pair<UI, UI> muladd) {
    UI mul = muladd.first, add = muladd.second, w = r2 - l2;
    Jump up(mul, 1), dn(-mul, 1);
    UI s(l1 * mul + add);
    Jump lo(r2 - s, 0), hi(s - l2, 0);
    function<void(Jump &, Jump &)> sub = [&](Jump & a, Jump & b) {
        if (a.val > w) {
            UI t(((long long)a.val - max(0ll, w + 1ll - b.val)) / b.val);
            a = a - t * b;
        }
    };
    sub(lo, up), sub(hi, dn);
    while (up.val > w || dn.val > w) {
        sub(up, dn); sub(lo, up);
        sub(dn, up); sub(hi, dn); }
    assert(up.val + dn.val > w);
    vector<UI> res;
    Jump bg(s + mul * min(lo.step, hi.step), min(lo.step, hi.step));
    while (bg.step <= r1 - l1) {
        if (l2 <= bg.val && bg.val <= r2)
            res.push_back(bg.step + l1);
        if (l2 <= bg.val - dn.val && bg.val - dn.val <= r2) {
            bg = bg - dn;
        } else bg = bg + up;
    } return res;
}
```

# 7 杂项

## 7.1 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;
```

## 7.2 真正释放 STL 内存

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

## 7.3 梅森旋转算法

```
#include <random>

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

## 7.4 蔡勒公式

```
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;
}
```

## 7.5 开栈

```
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;
}
```

## 7.6 Size 为 k 的子集

```
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;
    }
}
```

## 7.7 长方体表面两点最短距离

```
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;
}
```

## 7.8 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 |

## 7.9 NTT 素数及其原根

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

## 7.10 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：

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

## 7.11 博弈游戏-Reshiram

### 7.11.1 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。

### 7.11.2 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：

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

### 7.11.3 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)

### 7.11.4 图上删边游戏

### 7.11.5 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）

### 7.11.6 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。

### 7.11.7 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。

## 7.12 Formulas

## 7.13 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 $n$.

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

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

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

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

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

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

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

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

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

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

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

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

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

## 7.14 Binomial Coefficients

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

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

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

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

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

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

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

## 7.15 Fibonacci Numbers

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

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

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

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

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

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

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

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

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

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

## 7.16 Stirling Cycle Numbers

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

## 7.17 Stirling Subset Numbers

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

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

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

## 7.18 Eulerian Numbers

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

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

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

## 7.19 Harmonic Numbers

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

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

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

## 7.20 Pentagonal Number Theorem

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

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

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

## 7.21 Bell Numbers

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

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

## 7.22 Bernoulli Numbers

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

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

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

## 7.23 Tetrahedron Volume

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

## 7.24 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 $G$.

## 7.25 重心

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

### 7.26 Others

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

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

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

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

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

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

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

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

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

### Integrals of Rational Functions

$$\int \frac{1}{1+x^2}dx = \tan^{-1} x \tag{1}$$

$$\int \frac{1}{a^2+x^2}dx = \frac{1}{a}\tan^{-1}\frac{x}{a} \tag{2}$$

$$\int \frac{x}{a^2+x^2}dx = \frac{1}{2}\ln|a^2+x^2| \tag{3}$$

$$\int \frac{x^2}{a^2+x^2}dx = x - a\tan^{-1}\frac{x}{a} \tag{4}$$

$$\int \frac{x^3}{a^2+x^2}dx = \frac{1}{2}x^2 - \frac{1}{2}a^2\ln|a^2+x^2| \tag{5}$$

$$\int \frac{1}{ax^2+bx+c}dx = \frac{2}{\sqrt{4ac-b^2}}\tan^{-1}\frac{2ax+b}{\sqrt{4ac-b^2}} \tag{6}$$

$$\int \frac{1}{(x+a)(x+b)}dx = \frac{1}{b-a}\ln\frac{a+x}{b+x},\ a\neq b \tag{7}$$

$$\int \frac{x}{(x+a)^2}dx = \frac{a}{a+x} + \ln|a+x| \tag{8}$$

$$\begin{aligned}\int \frac{x}{ax^2+bx+c}dx &= \frac{1}{2a}\ln|ax^2+bx+c| \\ &- \frac{b}{a\sqrt{4ac-b^2}}\tan^{-1}\frac{2ax+b}{\sqrt{4ac-b^2}}\end{aligned} \tag{9}$$

### Integrals with Roots

$$\int \frac{x}{\sqrt{x\pm a}}dx = \frac{2}{3}(x\mp 2a)\sqrt{x\pm a} \tag{10}$$

$$\int \sqrt{\frac{x}{a-x}}dx = -\sqrt{x(a-x)} - a\tan^{-1}\frac{\sqrt{x(a-x)}}{x-a} \tag{11}$$

$$\int \sqrt{\frac{x}{a+x}}dx = \sqrt{x(a+x)} - a\ln\left[\sqrt{x}+\sqrt{x+a}\right] \tag{12}$$

$$\int x\sqrt{ax+b}dx = \frac{2}{15a^2}(-2b^2+abx+3a^2x^2)\sqrt{ax+b} \tag{13}$$

$$\begin{aligned}\int \sqrt{x(ax+b)}dx &= \frac{1}{4a^{3/2}}\Big[(2ax+b)\sqrt{ax(ax+b)} \\ &-b^2\ln\left|a\sqrt{x}+\sqrt{a(ax+b)}\right|\Big]\end{aligned} \tag{14}$$

$$\begin{aligned}\int \sqrt{x^3(ax+b)}dx &= \left[\frac{b}{12a}-\frac{b^2}{8a^2x}+\frac{x}{3}\right]\sqrt{x^3(ax+b)} \\ &+ \frac{b^3}{8a^{5/2}}\ln\left|a\sqrt{x}+\sqrt{a(ax+b)}\right|\end{aligned} \tag{15}$$

$$\int \sqrt{x^2\pm a^2}dx = \frac{1}{2}x\sqrt{x^2\pm a^2} \pm \frac{1}{2}a^2\ln\left|x+\sqrt{x^2\pm a^2}\right| \tag{16}$$

$$\int \sqrt{a^2-x^2}dx = \frac{1}{2}x\sqrt{a^2-x^2} + \frac{1}{2}a^2\tan^{-1}\frac{x}{\sqrt{a^2-x^2}} \tag{17}$$

$$\int x\sqrt{x^2\pm a^2}dx = \frac{1}{3}\left(x^2\pm a^2\right)^{3/2} \tag{18}$$

$$\int \frac{1}{\sqrt{x^2\pm a^2}}dx = \ln\left|x+\sqrt{x^2\pm a^2}\right| \tag{19}$$

$$\int \frac{1}{\sqrt{a^2-x^2}}dx = \sin^{-1}\frac{x}{a} \tag{20}$$

$$\int \frac{x}{\sqrt{x^2\pm a^2}}dx = \sqrt{x^2\pm a^2} \tag{21}$$

$$\int \frac{x}{\sqrt{a^2-x^2}}dx = -\sqrt{a^2-x^2} \tag{22}$$

$$\int \frac{x^2}{\sqrt{x^2\pm a^2}}dx = \frac{1}{2}x\sqrt{x^2\pm a^2} \mp \frac{1}{2}a^2\ln\left|x+\sqrt{x^2\pm a^2}\right| \tag{23}$$

$$\begin{aligned}\int \sqrt{ax^2+bx+c}dx &= \frac{b+2ax}{4a}\sqrt{ax^2+bx+c} \\ &+ \frac{4ac-b^2}{8a^{3/2}}\ln\left|2ax+b+2\sqrt{a(ax^2+bx^+c)}\right|\end{aligned} \tag{24}$$

$$\begin{aligned}\int x\sqrt{ax^2+bx+c} &= \frac{1}{48a^{5/2}}\Big(2\sqrt{a}\sqrt{ax^2+bx+c} \\ &\times\left(-3b^2+2abx+8a(c+ax^2)\right) \\ &+3(b^3-4abc)\ln\left|b+2ax+2\sqrt{a}\sqrt{ax^2+bx+c}\right|\Big)\end{aligned} \tag{25}$$

$$\int \frac{1}{\sqrt{ax^2+bx+c}}dx = \frac{1}{\sqrt{a}}\ln\left|2ax+b+2\sqrt{a(ax^2+bx+c)}\right| \tag{26}$$

$$\begin{aligned}\int \frac{x}{\sqrt{ax^2+bx+c}}dx &= \frac{1}{a}\sqrt{ax^2+bx+c} \\ &- \frac{b}{2a^{3/2}}\ln\left|2ax+b+2\sqrt{a(ax^2+bx+c)}\right|\end{aligned} \tag{27}$$

$$\int \frac{dx}{(a^2+x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2+x^2}} \tag{28}$$

### Integrals with Logarithms

$$\int \frac{\ln ax}{x}dx = \frac{1}{2}(\ln ax)^2 \tag{29}$$

$$\int \ln(ax+b)dx = \left(x+\frac{b}{a}\right)\ln(ax+b) - x, a\neq 0 \tag{30}$$

$$\int \ln(x^2+a^2)\,\mathrm{dx} = x\ln(x^2+a^2) + 2a\tan^{-1}\frac{x}{a} - 2x \tag{31}$$

$$\int \ln(x^2-a^2)\,\mathrm{dx} = x\ln(x^2-a^2) + a\ln\frac{x+a}{x-a} - 2x \tag{32}$$

$$\begin{aligned}\int \ln\left(ax^2+bx+c\right)dx &= \frac{1}{a}\sqrt{4ac-b^2}\tan^{-1}\frac{2ax+b}{\sqrt{4ac-b^2}} \\ &-2x+\left(\frac{b}{2a}+x\right)\ln\left(ax^2+bx+c\right)\end{aligned} \tag{33}$$

$$\begin{aligned}\int x\ln(ax+b)dx &= \frac{bx}{2a} - \frac{1}{4}x^2 \\ &+ \frac{1}{2}\left(x^2-\frac{b^2}{a^2}\right)\ln(ax+b)\end{aligned} \tag{34}$$

$$\begin{aligned}\int x\ln\left(a^2-b^2x^2\right)dx &= -\frac{1}{2}x^2 + \\ &\frac{1}{2}\left(x^2-\frac{a^2}{b^2}\right)\ln\left(a^2-b^2x^2\right)\end{aligned} \tag{35}$$

### Integrals with Exponentials

$$\int x^n e^{ax}\,\mathrm{dx} = \frac{x^n e^{ax}}{a} - \frac{n}{a}\int x^{n-1}e^{ax}\,\mathrm{dx} \tag{36}$$

$$\int xe^{-ax^2}\,\mathrm{dx} = -\frac{1}{2a}e^{-ax^2} \tag{37}$$

### Integrals with Trigonometric Functions

$$\int \sin^3 ax dx = -\frac{3\cos ax}{4a} + \frac{\cos 3ax}{12a} \tag{38}$$

$$\int \cos^2 ax dx = \frac{x}{2} + \frac{\sin 2ax}{4a} \tag{39}$$

$$\int \cos^3 ax dx = \frac{3\sin ax}{4a} + \frac{\sin 3ax}{12a} \tag{40}$$

$$\int \cos ax\sin bx dx = \frac{\cos[(a-b)x]}{2(a-b)} - \frac{\cos[(a+b)x]}{2(a+b)}, a\neq b \tag{41}$$

$$\begin{aligned}\int \sin^2 ax\cos bx dx &= -\frac{\sin[(2a-b)x]}{4(2a-b)} \\ &+ \frac{\sin bx}{2b} - \frac{\sin[(2a+b)x]}{4(2a+b)}\end{aligned} \tag{42}$$

$$\int \sin^2 x\cos x dx = \frac{1}{3}\sin^3 x \tag{43}$$

$$\int \cos^2 ax \sin bx dx = \frac{\cos[(2a-b)x]}{4(2a-b)} - \frac{\cos bx}{2b} - \frac{\cos[(2a+b)x]}{4(2a+b)} \tag{44}$$

$$\int \cos^2 ax \sin ax dx = -\frac{1}{3a}\cos^3 ax \tag{45}$$

$$\int \sin^2 ax \cos^2 bx dx = \frac{x}{4} - \frac{\sin 2ax}{8a} - \frac{\sin[2(a-b)x]}{16(a-b)} + \frac{\sin 2bx}{8b} - \frac{\sin[2(a+b)x]}{16(a+b)} \tag{46}$$

$$\int \sin^2 ax \cos^2 ax dx = \frac{x}{8} - \frac{\sin 4ax}{32a} \tag{47}$$

$$\int \tan ax dx = -\frac{1}{a}\ln\cos ax \tag{48}$$

$$\int \tan^2 ax dx = -x + \frac{1}{a}\tan ax \tag{49}$$

$$\int \tan^3 ax dx = \frac{1}{a}\ln\cos ax + \frac{1}{2a}\sec^2 ax \tag{50}$$

$$\int \sec x dx = \ln|\sec x + \tan x| = 2\tanh^{-1}\left(\tan\frac{x}{2}\right) \tag{51}$$

$$\int \sec^2 ax dx = \frac{1}{a}\tan ax \tag{52}$$

$$\int \sec^3 x \, \mathrm{dx} = \frac{1}{2}\sec x\tan x + \frac{1}{2}\ln|\sec x + \tan x| \tag{53}$$

$$\int \sec x \tan x dx = \sec x \tag{54}$$

$$\int \sec^2 x \tan x dx = \frac{1}{2}\sec^2 x \tag{55}$$

$$\int \sec^n x \tan x dx = \frac{1}{n}\sec^n x, n \neq 0 \tag{56}$$

$$\int \csc x dx = \ln\left|\tan\frac{x}{2}\right| = \ln|\csc x - \cot x| + C \tag{57}$$

$$\int \csc^2 ax dx = -\frac{1}{a}\cot ax \tag{58}$$

$$\int \csc^3 x dx = -\frac{1}{2}\cot x\csc x + \frac{1}{2}\ln|\csc x - \cot x| \tag{59}$$

$$\int \csc^n x \cot x dx = -\frac{1}{n}\csc^n x, n \neq 0 \tag{60}$$

$$\int \sec x \csc x dx = \ln|\tan x| \tag{61}$$

**Products of Trigonometric Functions and Monomials**

$$\int x\cos x dx = \cos x + x\sin x \tag{62}$$

$$\int x\cos ax dx = \frac{1}{a^2}\cos ax + \frac{x}{a}\sin ax \tag{63}$$

$$\int x^2\cos x dx = 2x\cos x + \left(x^2-2\right)\sin x \tag{64}$$

$$\int x^2\cos ax dx = \frac{2x\cos ax}{a^2} + \frac{a^2x^2-2}{a^3}\sin ax \tag{65}$$

$$\int x\sin x dx = -x\cos x + \sin x \tag{66}$$

$$\int x\sin ax dx = -\frac{x\cos ax}{a} + \frac{\sin ax}{a^2} \tag{67}$$

$$\int x^2\sin x dx = \left(2-x^2\right)\cos x + 2x\sin x \tag{68}$$

$$\int x^2\sin ax dx = \frac{2-a^2x^2}{a^3}\cos ax + \frac{2x\sin ax}{a^2} \tag{69}$$

**Products of Trigonometric Functions and Exponentials**

$$\int e^x\sin x dx = \frac{1}{2}e^x(\sin x - \cos x) \tag{70}$$

$$\int e^{bx}\sin ax dx = \frac{1}{a^2+b^2}e^{bx}(b\sin ax - a\cos ax) \tag{71}$$

$$\int e^x\cos x dx = \frac{1}{2}e^x(\sin x + \cos x) \tag{72}$$

$$\int e^{bx}\cos ax dx = \frac{1}{a^2+b^2}e^{bx}(a\sin ax + b\cos ax) \tag{73}$$

$$\int xe^x\sin x dx = \frac{1}{2}e^x(\cos x - x\cos x + x\sin x) \tag{74}$$

$$\int xe^x\cos x dx = \frac{1}{2}e^x(x\cos x - \sin x + x\sin x) \tag{75}$$

### 7.27 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());
    }
}
```

compact1, compact2, compact3

java.math

# Class BigInteger

java.lang.Object
    java.lang.Number
        java.math.BigInteger

**All Implemented Interfaces:**

`Serializable, Comparable<BigInteger>`

---

```
public class BigInteger
extends Number
implements Comparable<BigInteger>
```

Immutable arbitrary-precision integers. All operations behave as if BigIntegers were represented in two's-complement notation (like Java's primitive integer types). BigInteger provides analogues to all of Java's primitive integer operators, and all relevant methods from java.lang.Math. Additionally, BigInteger provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a few other miscellaneous operations.

Semantics of arithmetic operations exactly mimic those of Java's integer arithmetic operators, as defined in *The Java Language Specification*. For example, division by zero throws an `ArithmeticException`, and division of a negative by a positive yields a negative (or zero) remainder. All of the details in the Spec concerning overflow are ignored, as BigIntegers are made as large as necessary to accommodate the results of an operation.

Semantics of shift operations extend those of Java's shift operators to allow for negative shift distances. A right-shift with a negative shift distance results in a left shift, and vice-versa. The unsigned right shift operator (>>>) is omitted, as this operation makes little sense in combination with the "infinite word size" abstraction provided by this class.

Semantics of bitwise logical operations exactly mimic those of Java's bitwise integer operators. The binary operators (`and`, `or`, `xor`) implicitly perform sign extension on the shorter of the two operands prior to performing the operation.

Comparison operations perform signed integer comparisons, analogous to those performed by Java's relational and equality operators.

Modular arithmetic operations are provided to compute residues, perform exponentiation, and compute multiplicative inverses. These methods always return a non-negative result, between `0` and `(modulus - 1)`, inclusive.

Bit operations operate on a single bit of the two's-complement representation of their operand. If necessary, the operand is sign- extended so that it contains the designated bit. None of the single-bit operations can produce a BigInteger with a different sign from the BigInteger being operated on, as they affect only a single bit, and the "infinite word size" abstraction provided by this class ensures that there are infinitely many "virtual sign bits" preceding each BigInteger.

For the sake of brevity and clarity, pseudo-code is used throughout the descriptions of BigInteger methods. The pseudo-code expression `(i + j)` is shorthand for "a BigInteger whose value is that of the BigInteger `i` plus that of the BigInteger `j`." The pseudo-code expression `(i == j)` is shorthand for "`true` if and only if the BigInteger `i` represents the same value as the BigInteger `j`." Other pseudo-code expressions are interpreted similarly.

All methods and constructors in this class throw `NullPointerException` when passed a null object reference for any input parameter. BigInteger must support values in the range $-2^{\texttt{Integer.MAX\_VALUE}}$ (exclusive) to $+2^{\texttt{Integer.MAX\_VALUE}}$ (exclusive) and may support values outside of that range. The range of probable prime values is limited and may be less than the full supported positive range of `BigInteger`. The range must be at least 1 to $2^{500000000}$.

**Implementation Note:**

BigInteger constructors and operations throw ArithmeticException when the result is out of the supported range of $-2^{\texttt{Integer.MAX\_VALUE}}$ (exclusive) to $+2^{\texttt{Integer.MAX\_VALUE}}$ (exclusive).

**Since:**

JDK1.1

**See Also:**

BigDecimal, Serialized Form

## *Field Summary*

### Fields

| Modifier and Type | Field and Description |
|---|---|
| static **BigInteger** | **ONE**<br>The BigInteger constant one. |
| static **BigInteger** | **TEN**<br>The BigInteger constant ten. |
| static **BigInteger** | **ZERO**<br>The BigInteger constant zero. |

## *Constructor Summary*

### Constructors

| Constructor and Description |
|---|
| **BigInteger**(byte[] val)<br>Translates a byte array containing the two's-complement binary representation of a BigInteger into a BigInteger. |
| **BigInteger**(int signum, byte[] magnitude)<br>Translates the sign-magnitude representation of a BigInteger into a BigInteger. |

**BigInteger**(int bitLength, int certainty, **Random** rnd)
Constructs a randomly generated positive BigInteger that is probably prime, with the specified bitLength.

**BigInteger**(int numBits, **Random** rnd)
Constructs a randomly generated BigInteger, uniformly distributed over the range 0 to ($2^{numBits}$ - 1), inclusive.

**BigInteger**(**String** val)
Translates the decimal String representation of a BigInteger into a BigInteger.

**BigInteger**(**String** val, int radix)
Translates the String representation of a BigInteger in the specified radix into a BigInteger.

### *Method Summary*

**All Methods** Static Methods Instance Methods Concrete Methods

| Modifier and Type | Method and Description |
|---|---|
| **BigInteger** | **abs**()<br>Returns a BigInteger whose value is the absolute value of this BigInteger. |
| **BigInteger** | **add**(**BigInteger** val)<br>Returns a BigInteger whose value is `(this + val)`. |
| **BigInteger** | **and**(**BigInteger** val)<br>Returns a BigInteger whose value is `(this & val)`. |
| **BigInteger** | **andNot**(**BigInteger** val)<br>Returns a BigInteger whose value is `(this & ~val)`. |
| int | **bitCount**()<br>Returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit. |
| int | **bitLength**()<br>Returns the number of bits in the minimal two's-complement representation of this BigInteger, *excluding* a sign bit. |
| byte | **byteValueExact**()<br>Converts this `BigInteger` to a `byte`, checking for lost information. |
| **BigInteger** | **clearBit**(int n)<br>Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared. |
| int | **compareTo**(**BigInteger** val)<br>Compares this BigInteger with the specified BigInteger. |
| **BigInteger** | **divide**(**BigInteger** val) |

| | |
|---|---|
| | Returns a BigInteger whose value is `(this / val)`. |
| `BigInteger[]` | **`divideAndRemainder(BigInteger val)`**<br>Returns an array of two BigIntegers containing `(this / val)` followed by `(this % val)`. |
| `double` | **`doubleValue()`**<br>Converts this BigInteger to a `double`. |
| `boolean` | **`equals(Object x)`**<br>Compares this BigInteger with the specified Object for equality. |
| `BigInteger` | **`flipBit(int n)`**<br>Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit flipped. |
| `float` | **`floatValue()`**<br>Converts this BigInteger to a `float`. |
| `BigInteger` | **`gcd(BigInteger val)`**<br>Returns a BigInteger whose value is the greatest common divisor of `abs(this)` and `abs(val)`. |
| `int` | **`getLowestSetBit()`**<br>Returns the index of the rightmost (lowest-order) one bit in this BigInteger (the number of zero bits to the right of the rightmost one bit). |
| `int` | **`hashCode()`**<br>Returns the hash code for this BigInteger. |
| `int` | **`intValue()`**<br>Converts this BigInteger to an `int`. |
| `int` | **`intValueExact()`**<br>Converts this `BigInteger` to an `int`, checking for lost information. |
| `boolean` | **`isProbablePrime(int certainty)`**<br>Returns `true` if this BigInteger is probably prime, `false` if it's definitely composite. |
| `long` | **`longValue()`**<br>Converts this BigInteger to a `long`. |
| `long` | **`longValueExact()`**<br>Converts this `BigInteger` to a `long`, checking for lost information. |
| `BigInteger` | **`max(BigInteger val)`**<br>Returns the maximum of this BigInteger and `val`. |
| `BigInteger` | **`min(BigInteger val)`**<br>Returns the minimum of this BigInteger and `val`. |
| `BigInteger` | **`mod(BigInteger m)`**<br>Returns a BigInteger whose value is `(this mod m)`. |

| | |
|---|---|
| BigInteger | modInverse(BigInteger m)<br>Returns a BigInteger whose value is ($this^{-1}$ mod m). |
| BigInteger | modPow(BigInteger exponent, BigInteger m)<br>Returns a BigInteger whose value is ($this^{exponent}$ mod m). |
| BigInteger | multiply(BigInteger val)<br>Returns a BigInteger whose value is (this * val). |
| BigInteger | negate()<br>Returns a BigInteger whose value is (-this). |
| BigInteger | nextProbablePrime()<br>Returns the first integer greater than this BigInteger that is probably prime. |
| BigInteger | not()<br>Returns a BigInteger whose value is (~this). |
| BigInteger | or(BigInteger val)<br>Returns a BigInteger whose value is (this \| val). |
| BigInteger | pow(int exponent)<br>Returns a BigInteger whose value is ($this^{exponent}$). |
| static BigInteger | probablePrime(int bitLength, Random rnd)<br>Returns a positive BigInteger that is probably prime, with the specified bitLength. |
| BigInteger | remainder(BigInteger val)<br>Returns a BigInteger whose value is (this % val). |
| BigInteger | setBit(int n)<br>Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit set. |
| BigInteger | shiftLeft(int n)<br>Returns a BigInteger whose value is (this << n). |
| BigInteger | shiftRight(int n)<br>Returns a BigInteger whose value is (this >> n). |
| short | shortValueExact()<br>Converts this BigInteger to a short, checking for lost information. |
| int | signum()<br>Returns the signum function of this BigInteger. |
| BigInteger | subtract(BigInteger val)<br>Returns a BigInteger whose value is (this - val). |
| boolean | testBit(int n)<br>Returns true if and only if the designated bit is set. |
| byte[] | toByteArray() |

| | |
|---|---|
| | Returns a byte array containing the two's-complement representation of this BigInteger. |
| `String` | `toString()`<br>Returns the decimal String representation of this BigInteger. |
| `String` | `toString(int radix)`<br>Returns the String representation of this BigInteger in the given radix. |
| `static BigInteger` | `valueOf(long val)`<br>Returns a BigInteger whose value is equal to that of the specified `long`. |
| `BigInteger` | `xor(BigInteger val)`<br>Returns a BigInteger whose value is `(this ^ val)`. |

### Methods inherited from class java.lang.Number

`byteValue`, `shortValue`

### Methods inherited from class java.lang.Object

`clone`, `finalize`, `getClass`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

## *Field Detail*

### ZERO

`public static final BigInteger ZERO`

The BigInteger constant zero.

**Since:**

1.2

### ONE

`public static final BigInteger ONE`

The BigInteger constant one.

**Since:**

1.2

### TEN

`public static final BigInteger TEN`

The BigInteger constant ten.

Other methods may have slightly different rounding semantics. For example, the result of the pow method using the specified algorithm can occasionally differ from the rounded mathematical result by more than one unit in the last place, one *ulp*.

Two types of operations are provided for manipulating the scale of a BigDecimal: scaling/rounding operations and decimal point motion operations. Scaling/rounding operations (setScale and round) return a BigDecimal whose value is approximately (or exactly) equal to that of the operand, but whose scale or precision is the specified value; that is, they increase or decrease the precision of the stored number with minimal effect on its value. Decimal point motion operations (movePointLeft and movePointRight) return a BigDecimal created from the operand by moving the decimal point a specified distance in the specified direction.

For the sake of brevity and clarity, pseudo-code is used throughout the descriptions of BigDecimal methods. The pseudo-code expression (i + j) is shorthand for "a BigDecimal whose value is that of the BigDecimal i added to that of the BigDecimal j." The pseudo-code expression (i == j) is shorthand for "true if and only if the BigDecimal i represents the same value as the BigDecimal j." Other pseudo-code expressions are interpreted similarly. Square brackets are used to represent the particular BigInteger and scale pair defining a BigDecimal value; for example [19, 2] is the BigDecimal numerically equal to 0.19 having a scale of 2.

Note: care should be exercised if BigDecimal objects are used as keys in a SortedMap or elements in a SortedSet since BigDecimal's *natural ordering* is *inconsistent with equals*. See Comparable, SortedMap or SortedSet for more information.

All methods and constructors for this class throw NullPointerException when passed a null object reference for any input parameter.

**See Also:**

BigInteger, MathContext, RoundingMode, SortedMap, SortedSet, Serialized Form

## Field Summary

### Fields

| Modifier and Type | Field and Description |
|---|---|
| static BigDecimal | **ONE**<br>The value 1, with a scale of 0. |
| static int | **ROUND_CEILING**<br>Rounding mode to round towards positive infinity. |
| static int | **ROUND_DOWN**<br>Rounding mode to round towards zero. |
| static int | **ROUND_FLOOR**<br>Rounding mode to round towards negative infinity. |
| static int | **ROUND_HALF_DOWN**<br>Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round down. |
| static int | **ROUND_HALF_EVEN** |

| | |
|---|---|
| | Rounding mode to round towards the "nearest neighbor" unless both neighbors are equidistant, in which case, round towards the even neighbor. |
| static int | **ROUND_HALF_UP**<br>Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round up. |
| static int | **ROUND_UNNECESSARY**<br>Rounding mode to assert that the requested operation has an exact result, hence no rounding is necessary. |
| static int | **ROUND_UP**<br>Rounding mode to round away from zero. |
| static **BigDecimal** | **TEN**<br>The value 10, with a scale of 0. |
| static **BigDecimal** | **ZERO**<br>The value 0, with a scale of 0. |

## *Constructor Summary*

### Constructors

**Constructor and Description**

**BigDecimal**(**BigInteger** val)
Translates a BigInteger into a BigDecimal.

**BigDecimal**(**BigInteger** unscaledVal, int scale)
Translates a BigInteger unscaled value and an int scale into a BigDecimal.

**BigDecimal**(**BigInteger** unscaledVal, int scale, **MathContext** mc)
Translates a BigInteger unscaled value and an int scale into a BigDecimal, with rounding according to the context settings.

**BigDecimal**(**BigInteger** val, **MathContext** mc)
Translates a BigInteger into a BigDecimal rounding according to the context settings.

**BigDecimal**(char[] in)
Translates a character array representation of a BigDecimal into a BigDecimal, accepting the same sequence of characters as the **BigDecimal(String)** constructor.

**BigDecimal**(char[] in, int offset, int len)
Translates a character array representation of a BigDecimal into a BigDecimal, accepting the same sequence of characters as the **BigDecimal(String)** constructor, while allowing a sub-array to be specified.

**BigDecimal**(char[] in, int offset, int len, **MathContext** mc)
Translates a character array representation of a BigDecimal into a BigDecimal, accepting the same sequence of characters as the **BigDecimal(String)** constructor, while allowing a sub-array to be specified and with rounding according to the context settings.

**BigDecimal**(char[] in, **MathContext** mc)
Translates a character array representation of a BigDecimal into a BigDecimal, accepting the same sequence of characters as the **BigDecimal(String)** constructor and with rounding according to the context settings.

**BigDecimal**(double val)
Translates a double into a BigDecimal which is the exact decimal representation of the double's binary floating-point value.

**BigDecimal**(double val, **MathContext** mc)
Translates a double into a BigDecimal, with rounding according to the context settings.

**BigDecimal**(int val)
Translates an int into a BigDecimal.

**BigDecimal**(int val, **MathContext** mc)
Translates an int into a BigDecimal, with rounding according to the context settings.

**BigDecimal**(long val)
Translates a long into a BigDecimal.

**BigDecimal**(long val, **MathContext** mc)
Translates a long into a BigDecimal, with rounding according to the context settings.

**BigDecimal**(**String** val)
Translates the string representation of a BigDecimal into a BigDecimal.

**BigDecimal**(**String** val, **MathContext** mc)
Translates the string representation of a BigDecimal into a BigDecimal, accepting the same strings as the **BigDecimal(String)** constructor, with rounding according to the context settings.

## *Method Summary*

**All Methods** Static Methods Instance Methods Concrete Methods

| Modifier and Type | Method and Description |
|---|---|
| **BigDecimal** | **abs**()<br>Returns a BigDecimal whose value is the absolute value of this BigDecimal, and whose scale is this.scale(). |
| **BigDecimal** | **abs**(**MathContext** mc)<br>Returns a BigDecimal whose value is the absolute value of this BigDecimal, with rounding according to the context settings. |
| **BigDecimal** | **add**(**BigDecimal** augend)<br>Returns a BigDecimal whose value is (this + augend), and whose scale is max(this.scale(), augend.scale()). |
| **BigDecimal** | **add**(**BigDecimal** augend, **MathContext** mc)<br>Returns a BigDecimal whose value is (this + augend), with rounding according to the context settings. |

| | |
|---|---|
| byte | **byteValueExact**()<br>Converts this BigDecimal to a byte, checking for lost information. |
| int | **compareTo**(**BigDecimal** val)<br>Compares this BigDecimal with the specified BigDecimal. |
| **BigDecimal** | **divide**(**BigDecimal** divisor)<br>Returns a BigDecimal whose value is (this / divisor), and whose preferred scale is (this.scale() - divisor.scale()); if the exact quotient cannot be represented (because it has a non-terminating decimal expansion) an ArithmeticException is thrown. |
| **BigDecimal** | **divide**(**BigDecimal** divisor, int roundingMode)<br>Returns a BigDecimal whose value is (this / divisor), and whose scale is this.scale(). |
| **BigDecimal** | **divide**(**BigDecimal** divisor, int scale, int roundingMode)<br>Returns a BigDecimal whose value is (this / divisor), and whose scale is as specified. |
| **BigDecimal** | **divide**(**BigDecimal** divisor, int scale, **RoundingMode** roundingMode)<br>Returns a BigDecimal whose value is (this / divisor), and whose scale is as specified. |
| **BigDecimal** | **divide**(**BigDecimal** divisor, **MathContext** mc)<br>Returns a BigDecimal whose value is (this / divisor), with rounding according to the context settings. |
| **BigDecimal** | **divide**(**BigDecimal** divisor, **RoundingMode** roundingMode)<br>Returns a BigDecimal whose value is (this / divisor), and whose scale is this.scale(). |
| **BigDecimal**[] | **divideAndRemainder**(**BigDecimal** divisor)<br>Returns a two-element BigDecimal array containing the result of divideToIntegralValue followed by the result of remainder on the two operands. |
| **BigDecimal**[] | **divideAndRemainder**(**BigDecimal** divisor, **MathContext** mc)<br>Returns a two-element BigDecimal array containing the result of divideToIntegralValue followed by the result of remainder on the two operands calculated with rounding according to the context settings. |
| **BigDecimal** | **divideToIntegralValue**(**BigDecimal** divisor)<br>Returns a BigDecimal whose value is the integer part of the quotient (this / divisor) rounded down. |
| **BigDecimal** | **divideToIntegralValue**(**BigDecimal** divisor, **MathContext** mc)<br>Returns a BigDecimal whose value is the integer part of (this / divisor). |
| double | **doubleValue**() |

| | |
|---|---|
| | Converts this `BigDecimal` to a `double`. |
| `boolean` | **`equals`**`(`**`Object`**` x)`<br>Compares this `BigDecimal` with the specified `Object` for equality. |
| `float` | **`floatValue`**`()`<br>Converts this `BigDecimal` to a `float`. |
| `int` | **`hashCode`**`()`<br>Returns the hash code for this `BigDecimal`. |
| `int` | **`intValue`**`()`<br>Converts this `BigDecimal` to an `int`. |
| `int` | **`intValueExact`**`()`<br>Converts this `BigDecimal` to an `int`, checking for lost information. |
| `long` | **`longValue`**`()`<br>Converts this `BigDecimal` to a `long`. |
| `long` | **`longValueExact`**`()`<br>Converts this `BigDecimal` to a `long`, checking for lost information. |
| **`BigDecimal`** | **`max`**`(`**`BigDecimal`**` val)`<br>Returns the maximum of this `BigDecimal` and `val`. |
| **`BigDecimal`** | **`min`**`(`**`BigDecimal`**` val)`<br>Returns the minimum of this `BigDecimal` and `val`. |
| **`BigDecimal`** | **`movePointLeft`**`(int n)`<br>Returns a `BigDecimal` which is equivalent to this one with the decimal point moved `n` places to the left. |
| **`BigDecimal`** | **`movePointRight`**`(int n)`<br>Returns a `BigDecimal` which is equivalent to this one with the decimal point moved `n` places to the right. |
| **`BigDecimal`** | **`multiply`**`(`**`BigDecimal`**` multiplicand)`<br>Returns a `BigDecimal` whose value is `(this × multiplicand)`, and whose scale is `(this.scale() + multiplicand.scale())`. |
| **`BigDecimal`** | **`multiply`**`(`**`BigDecimal`**` multiplicand, `**`MathContext`**` mc)`<br>Returns a `BigDecimal` whose value is `(this × multiplicand)`, with rounding according to the context settings. |
| **`BigDecimal`** | **`negate`**`()`<br>Returns a `BigDecimal` whose value is `(-this)`, and whose scale is `this.scale()`. |
| **`BigDecimal`** | **`negate`**`(`**`MathContext`**` mc)`<br>Returns a `BigDecimal` whose value is `(-this)`, with rounding according to the context settings. |
| **`BigDecimal`** | **`plus`**`()` |

| | |
|---|---|
| | Returns a BigDecimal whose value is (+this), and whose scale is this.scale(). |
| BigDecimal | **plus**(MathContext mc)<br>Returns a BigDecimal whose value is (+this), with rounding according to the context settings. |
| BigDecimal | **pow**(int n)<br>Returns a BigDecimal whose value is ($this^n$), The power is computed exactly, to unlimited precision. |
| BigDecimal | **pow**(int n, MathContext mc)<br>Returns a BigDecimal whose value is ($this^n$). |
| int | **precision**()<br>Returns the *precision* of this BigDecimal. |
| BigDecimal | **remainder**(BigDecimal divisor)<br>Returns a BigDecimal whose value is (this % divisor). |
| BigDecimal | **remainder**(BigDecimal divisor, MathContext mc)<br>Returns a BigDecimal whose value is (this % divisor), with rounding according to the context settings. |
| BigDecimal | **round**(MathContext mc)<br>Returns a BigDecimal rounded according to the MathContext settings. |
| int | **scale**()<br>Returns the *scale* of this BigDecimal. |
| BigDecimal | **scaleByPowerOfTen**(int n)<br>Returns a BigDecimal whose numerical value is equal to (this * $10^n$). |
| BigDecimal | **setScale**(int newScale)<br>Returns a BigDecimal whose scale is the specified value, and whose value is numerically equal to this BigDecimal's. |
| BigDecimal | **setScale**(int newScale, int roundingMode)<br>Returns a BigDecimal whose scale is the specified value, and whose unscaled value is determined by multiplying or dividing this BigDecimal's unscaled value by the appropriate power of ten to maintain its overall value. |
| BigDecimal | **setScale**(int newScale, RoundingMode roundingMode)<br>Returns a BigDecimal whose scale is the specified value, and whose unscaled value is determined by multiplying or dividing this BigDecimal's unscaled value by the appropriate power of ten to maintain its overall value. |
| short | **shortValueExact**()<br>Converts this BigDecimal to a short, checking for lost information. |
| int | **signum**() |

| | |
|---|---|
| | Returns the signum function of this `BigDecimal`. |
| **BigDecimal** | **stripTrailingZeros**()<br>Returns a `BigDecimal` which is numerically equal to this one but with any trailing zeros removed from the representation. |
| **BigDecimal** | **subtract**(**BigDecimal** subtrahend)<br>Returns a `BigDecimal` whose value is `(this - subtrahend)`, and whose scale is `max(this.scale(), subtrahend.scale())`. |
| **BigDecimal** | **subtract**(**BigDecimal** subtrahend, **MathContext** mc)<br>Returns a `BigDecimal` whose value is `(this - subtrahend)`, with rounding according to the context settings. |
| **BigInteger** | **toBigInteger**()<br>Converts this `BigDecimal` to a `BigInteger`. |
| **BigInteger** | **toBigIntegerExact**()<br>Converts this `BigDecimal` to a `BigInteger`, checking for lost information. |
| **String** | **toEngineeringString**()<br>Returns a string representation of this `BigDecimal`, using engineering notation if an exponent is needed. |
| **String** | **toPlainString**()<br>Returns a string representation of this `BigDecimal` without an exponent field. |
| **String** | **toString**()<br>Returns the string representation of this `BigDecimal`, using scientific notation if an exponent is needed. |
| **BigDecimal** | **ulp**()<br>Returns the size of an ulp, a unit in the last place, of this `BigDecimal`. |
| **BigInteger** | **unscaledValue**()<br>Returns a `BigInteger` whose value is the *unscaled value* of this `BigDecimal`. |
| static **BigDecimal** | **valueOf**(double val)<br>Translates a `double` into a `BigDecimal`, using the `double`'s canonical string representation provided by the **Double.toString(double)** method. |
| static **BigDecimal** | **valueOf**(long val)<br>Translates a `long` value into a `BigDecimal` with a scale of zero. |
| static **BigDecimal** | **valueOf**(long unscaledVal, int scale)<br>Translates a `long` unscaled value and an `int` scale into a `BigDecimal`. |

### Methods inherited from class java.lang.Number

byteValue, shortValue

compact1, compact2, compact3

java.util

# Class TreeMap<K,V>

java.lang.Object
    java.util.AbstractMap<K,V>
        java.util.TreeMap<K,V>

**Type Parameters:**

`K` - the type of keys maintained by this map

`V` - the type of mapped values

**All Implemented Interfaces:**

`Serializable`, `Cloneable`, `Map<K,V>`, `NavigableMap<K,V>`, `SortedMap<K,V>`

---

```
public class TreeMap<K,V>
extends AbstractMap<K,V>
implements NavigableMap<K,V>, Cloneable, Serializable
```

A Red-Black tree based `NavigableMap` implementation. The map is sorted according to the natural ordering of its keys, or by a `Comparator` provided at map creation time, depending on which constructor is used.

This implementation provides guaranteed log(n) time cost for the `containsKey`, `get`, `put` and `remove` operations. Algorithms are adaptations of those in Cormen, Leiserson, and Rivest's *Introduction to Algorithms*.

Note that the ordering maintained by a tree map, like any sorted map, and whether or not an explicit comparator is provided, must be *consistent with* `equals` if this sorted map is to correctly implement the `Map` interface. (See `Comparable` or `Comparator` for a precise definition of *consistent with equals*.) This is so because the `Map` interface is defined in terms of the `equals` operation, but a sorted map performs all key comparisons using its `compareTo` (or `compare`) method, so two keys that are deemed equal by this method are, from the standpoint of the sorted map, equal. The behavior of a sorted map *is* well-defined even if its ordering is inconsistent with `equals`; it just fails to obey the general contract of the `Map` interface.

**Note that this implementation is not synchronized.** If multiple threads access a map concurrently, and at least one of the threads modifies the map structurally, it *must* be synchronized externally. (A structural modification is any operation that adds or deletes one or more mappings; merely changing the value associated with an existing key is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the map. If no such object exists, the map should be "wrapped" using the `Collections.synchronizedSortedMap` method. This is best done at creation time, to prevent accidental unsynchronized access to the map:

```
SortedMap m = Collections.synchronizedSortedMap(new TreeMap(...));
```

The iterators returned by the `iterator` method of the collections returned by all of this class's "collection view methods" are *fail-fast*: if the map is structurally modified at any time after the iterator is created, in any way except through the iterator's own `remove` method, the iterator will throw a `ConcurrentModificationException`. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw `ConcurrentModificationException` on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: *the fail-fast behavior of iterators should be used only to detect bugs.*

All `Map.Entry` pairs returned by methods in this class and its views represent snapshots of mappings at the time they were produced. They do **not** support the `Entry.setValue` method. (Note however that it is possible to change mappings in the associated map using `put`.)

This class is a member of the Java Collections Framework.

**Since:**

1.2

**See Also:**

`Map`, `HashMap`, `Hashtable`, `Comparable`, `Comparator`, `Collection`, `Serialized Form`

### *Nested Class Summary*

#### Nested classes/interfaces inherited from class java.util.AbstractMap

`AbstractMap.SimpleEntry<K,V>`, `AbstractMap.SimpleImmutableEntry<K,V>`

### *Constructor Summary*

#### Constructors

**Constructor and Description**

`TreeMap()`
Constructs a new, empty tree map, using the natural ordering of its keys.

`TreeMap(Comparator<? super K> comparator)`
Constructs a new, empty tree map, ordered according to the given comparator.

`TreeMap(Map<? extends K,? extends V> m)`
Constructs a new tree map containing the same mappings as the given map, ordered according to the *natural ordering* of its keys.

`TreeMap(SortedMap<K,? extends V> m)`
Constructs a new tree map containing the same mappings and using the same ordering as the specified sorted map.

### *Method Summary*

**All Methods** Instance Methods Concrete Methods

| Modifier and Type | Method and Description |
|---|---|
| Map.Entry<K,V> | **ceilingEntry**(K key)<br>Returns a key-value mapping associated with the least key greater than or equal to the given key, or `null` if there is no such key. |
| K | **ceilingKey**(K key)<br>Returns the least key greater than or equal to the given key, or `null` if there is no such key. |
| void | **clear**()<br>Removes all of the mappings from this map. |
| Object | **clone**()<br>Returns a shallow copy of this `TreeMap` instance. |
| Comparator<? super K> | **comparator**()<br>Returns the comparator used to order the keys in this map, or `null` if this map uses the **natural ordering** of its keys. |
| boolean | **containsKey**(Object key)<br>Returns `true` if this map contains a mapping for the specified key. |
| boolean | **containsValue**(Object value)<br>Returns `true` if this map maps one or more keys to the specified value. |
| NavigableSet<K> | **descendingKeySet**()<br>Returns a reverse order `NavigableSet` view of the keys contained in this map. |
| NavigableMap<K,V> | **descendingMap**()<br>Returns a reverse order view of the mappings contained in this map. |
| Set<Map.Entry<K,V>> | **entrySet**()<br>Returns a `Set` view of the mappings contained in this map. |
| Map.Entry<K,V> | **firstEntry**()<br>Returns a key-value mapping associated with the least key in this map, or `null` if the map is empty. |
| K | **firstKey**()<br>Returns the first (lowest) key currently in this map. |
| Map.Entry<K,V> | **floorEntry**(K key)<br>Returns a key-value mapping associated with the greatest key less than or equal to the given key, or `null` if there is no such key. |
| K | **floorKey**(K key)<br>Returns the greatest key less than or equal to the given key, or `null` if there is no such key. |

| | |
|---|---|
| | or null if there is no such key. |
| void | **forEach**(**BiConsumer**<? super **K**,? super **V**> action)<br>Performs the given action for each entry in this map until all entries have been processed or the action throws an exception. |
| **V** | **get**(**Object** key)<br>Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| **SortedMap**<**K**,**V**> | **headMap**(**K** toKey)<br>Returns a view of the portion of this map whose keys are strictly less than toKey. |
| **NavigableMap**<**K**,**V**> | **headMap**(**K** toKey, boolean inclusive)<br>Returns a view of the portion of this map whose keys are less than (or equal to, if inclusive is true) toKey. |
| **Map.Entry**<**K**,**V**> | **higherEntry**(**K** key)<br>Returns a key-value mapping associated with the least key strictly greater than the given key, or null if there is no such key. |
| **K** | **higherKey**(**K** key)<br>Returns the least key strictly greater than the given key, or null if there is no such key. |
| **Set**<**K**> | **keySet**()<br>Returns a **Set** view of the keys contained in this map. |
| **Map.Entry**<**K**,**V**> | **lastEntry**()<br>Returns a key-value mapping associated with the greatest key in this map, or null if the map is empty. |
| **K** | **lastKey**()<br>Returns the last (highest) key currently in this map. |
| **Map.Entry**<**K**,**V**> | **lowerEntry**(**K** key)<br>Returns a key-value mapping associated with the greatest key strictly less than the given key, or null if there is no such key. |
| **K** | **lowerKey**(**K** key)<br>Returns the greatest key strictly less than the given key, or null if there is no such key. |
| **NavigableSet**<**K**> | **navigableKeySet**()<br>Returns a **NavigableSet** view of the keys contained in this map. |
| **Map.Entry**<**K**,**V**> | **pollFirstEntry**()<br>Removes and returns a key-value mapping associated with the least key in this map, or null if the map is empty. |
| **Map.Entry**<**K**,**V**> | **pollLastEntry**()<br>Removes and returns a key-value mapping associated with the greatest key in this map, or null if the map is empty. |

| | |
|---|---|
| | the greatest key in this map, or null if the map is empty. |
| V | put(K key, V value)<br>Associates the specified value with the specified key in this map. |
| void | putAll(Map<? extends K,? extends V> map)<br>Copies all of the mappings from the specified map to this map. |
| V | remove(Object key)<br>Removes the mapping for this key from this TreeMap if present. |
| V | replace(K key, V value)<br>Replaces the entry for the specified key only if it is currently mapped to some value. |
| boolean | replace(K key, V oldValue, V newValue)<br>Replaces the entry for the specified key only if currently mapped to the specified value. |
| void | replaceAll(BiFunction<? super K,? super V,? extends V> function)<br>Replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception. |
| int | size()<br>Returns the number of key-value mappings in this map. |
| NavigableMap<K,V> | subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive)<br>Returns a view of the portion of this map whose keys range from fromKey to toKey. |
| SortedMap<K,V> | subMap(K fromKey, K toKey)<br>Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive. |
| SortedMap<K,V> | tailMap(K fromKey)<br>Returns a view of the portion of this map whose keys are greater than or equal to fromKey. |
| NavigableMap<K,V> | tailMap(K fromKey, boolean inclusive)<br>Returns a view of the portion of this map whose keys are greater than (or equal to, if inclusive is true) fromKey. |
| Collection<V> | values()<br>Returns a Collection view of the values contained in this map. |

### Methods inherited from class java.util.AbstractMap

equals, hashCode, isEmpty, toString