1 ComputationalGeometry

1.1 Plane Geometry

```
namespace Plane_Geometry{
const int Maxn = 100010;
const double eps = 1e-8;
const double PI = acos(-1.0);
#define sqr(x) ((x) * (x))
int dcmp(double x){ return (fabs(x) < eps) ? 0 : (x < 0 ? -1 : 1); }
struct point{
    double x, y;
    point(double x = 0, double y = 0) : x(x), y(y) {}
    bool operator < (const point &p) const{
        if(fabs(x - p.x) > eps) return x < p.x;
        return y < p.y - eps;
    }
    point operator + (const point &p) const{ return {x + p.x, y + p.y}; }
    point operator - (const point &p) const{ return {x - p.x, y - p.y}; }
    point operator -= (const point &p){ x -= p.x; y -= p.y; return *this; }
    double operator * (const point &p) const{ return x * p.x + y * p.y; }
    point operator * (const double &k) const{ return {x * k, y * k}; }
    point operator / (const double &k) const{ return {x / k, y / k}; }
    point operator /= (const double &k){ x /= k; y /= k; return *this; }
    double operator ^ (const point &p) const{ return x * p.y - y * p.x; }
    bool operator == (const point &p) const{ return fabs(x - p.x) < eps && fabs(y - p.y) < eps; }
    double module() const{ return hypot(x, y); }
    point norm() const{ return *this / module(); }
    point anticlockwise_orthogonal() const{ return {-y, x}; }
    point clockwise_orthogonal() const{ return {y, -x}; }
    double dist(const point &p) const{ return (*this - p).module(); }
    //判断是否在线段ab上
    bool on_segment(const point &a, const point &b) const{
        return fabs(((a - *this) ^ (b - *this))) < eps && ((a - *this) * (b - *this)) < eps;
    }
    //绕点p逆时针旋转a弧度
    void rotate(double a, point p = {0, 0}){
        double rx = (x - p.x) * cos(a) - (y - p.y) * sin(a) + p.x;
        double ry = (x - p.x) * sin(a) + (y - p.y) * cos(a) + p.y;
        x = rx, y = ry;
    }
    //到直线ab的距离
    double distance_to_line(const point &a, const point &b) const{
        return fabs((a - *this) ^ (b - *this)) / (a - b).module();
    }
    //到线段ab的最短距离
    double mindistance_to_segment(const point &a, const point &b) const{
        if((b - a) * (*this - a) < eps) return (*this - a).module();
        if((a - b) * (*this - b) < eps) return (*this - b).module();
        return fabs((b - a) ^ (*this - a)) / (a - b).module();
    }
    //到直线ab上的投影
    point get_line_projection(const point &a, const point &b) const{
        point v = b - a;
        return a + v * ((v * (*this - a)) / (v * v));
    }
    //极角排序
    void polar_angle_sort(point *p, const int &N) const{
        auto dy = [](const double &x, const double &y) -> bool { return x > y + eps; }; // x > y
        auto xy = [](const double &x, const double &y) -> bool { return x < y - eps; }; // x < y
        auto dyd = [](const double &x, const double &y) -> bool { return x > y - eps; }; // x >= y
        auto xyd = [](const double &x, const double &y) -> bool { return x < y + eps; }; // x <= y
        auto dd = [](const double &x, const double &y) -> bool { return fabs(x - y) < eps; };// x == y
        auto quad = [&](const point &a) -> int {
            if( dy(a.x, 0) && dyd(a.y, 0) ) return 1;
            if( xyd(a.x, 0) && dy(a.y, 0) ) return 2;
            if( xy(a.x, 0) && xyd(a.y, 0) ) return 3;
            if( dyd(a.x, 0) && xy(a.y, 0) ) return 4;
            return -1;
        };
        sort(p, p + N, [&](const point& a, const point& b){
            point p1 = a - *this, p2 = b - *this;
            int l1 = quad(p1), l2 = quad(p2);
            if( l1 == l2 ){
                LL c = p1 ^ p2;
                return dy(c, 0) || (dd(c, 0) && xy(abs(a.x), abs(b.x)));
            }
            return l1 < l2;
        });
    }
    int Read(){ return scanf("%lf %lf", &x, &y); }
    void Print(){ printf("%.10f %.10f\n", x, y); }
};
//判断直线ab和线段cd相交
bool line_intersect_segment(const point &a, const point &b, const point &c, const point &d){
    return ((b - a) ^ (c - a)) * ((b - a) ^ (d - a)) < eps;
}
//判断线段ab和线段cd严格相交
bool segments_intersect_strictly(const point &a, const point &b, const point &c, const point &d){
    double p = ((b - a) ^ (c - a)) * ((b - a) ^ (d - a)), q = ((d - c) ^ (a - c)) * ((d - c) ^ (b - c));
    return p < -eps && q < -eps;
}
//求直线ab和直线cd的交点
point lines_intersect(const point &a, const point &b, const point &c, const point &d){
    double p = (b - a) ^ (c - a), q = (b - a) ^ (d - a);
    return (c * q - d * p) / (q - p);
}
//多边形的有向面积
double polygon_area(point *p, const int &N){
    double area = 0;
    for(int i = 1; i < N - 1; ++i)
        area += (p[i] - p[0]) ^ (p[i + 1] - p[0]);
    return area / 2;
}
// 多边形重心
point masscenter(point *p, const int &N) {
    point ret = point(0, 0);
    double sum = polygon_area(p, N);
    if(dcmp(sum) == 0) return ret;
    p[N] = p[0];
    for(int i = 0; i < N; ++i)
        ret = ret + (p[i] + p[i + 1]) * (p[i + 1] ^ p[i]);
    return ret / sum / 6.0;
}
//将大小为N的点集p打凸包，convex保存被选中的凸包的点，cTotal是凸包中点的个数
void pack_convex_hull(point *p, const int &N, point *convex, int &cTotal){
    sort(p, p + N, [](const point &a, const point &b){
        if(fabs(a.x - b.x) > eps) return a.x < b.x;
        return a.y < b.y;
    });
    int i, Total = 0, tmp;
    for(i = 0; i < N; ++i){
        while(Total > 1 && ((convex[Total - 1] - convex[Total - 2]) ^ (p[i] - convex[Total - 1])) <= 0) Total--;
        convex[Total++] = p[i];
```

```
    }
    for(i = N - 2, tmp = Total; i >= 0; --i){
        while(Total > tmp && ((convex[Total - 1] - convex[Total - 2]) ^ (p[i] - convex[Total - 1])) <= 0) Total--;
        convex[Total++] = p[i];
    }
    cTotal = Total;
}
//判断点p是否在大小为N的凸包convex内
    （必须是严格凸包，不存在三点共线）
bool check_point_into_convex_hull(const point &p, point *convex, const int &N){
    for(int l = 1, r = N - 2, mid; l <= r;){
        mid = (l + r) >> 1;
        double a1 = (convex[mid] - convex[0]) ^ (p - convex[0]);
        double a2 = (convex[mid + 1] - convex[0]) ^ (p - convex[0]);
        if(a1 >= 0 && a2 <= 0){
            if(((convex[mid + 1] - convex[mid]) ^ (p - convex[mid])) >= 0)return true;
            return false;
        }
        else if(a1 < 0) r = mid - 1;
        else l = mid + 1;
    }
    return false;
}
//判断点p是否在大小为N的多边形poly内
int point_in_polygon(const point &p, point *poly, const int &N){
    int wn = 0; poly[N] = poly[0];
    for(int i = 0; i < N; ++i) {
        if (p.on_segment(poly[i], poly[i + 1])) return -1; //在边界上
        int k = dcmp((poly[i + 1] - poly[i]) ^ (p - poly[i]));
        int d1 = dcmp(poly[i].y - p.y);
        int d2 = dcmp(poly[i + 1].y - p.y);
        if (k > 0 && d1 <= 0 && d2 > 0) wn++;
        if (k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    if (wn != 0) return 1; //内部
    return 0;            //外部
}
struct circle {
    point o; double r, ang; int d;
    circle(point o = point(0, 0), double r = 0, double ang = 0, int d = 1) : o(o), r(r), ang(ang), d(d) {}
    //计算弧度为a在圆上的点
    point get_point(double a) const{
        return point(o.x + cos(a) * r, o.y + sin(a) * r);
    }
    //圆的公切线 返回切线条数
    int get_tangents(circle B, point *a, point *b) const{
        circle A = *this;
        int cnt = 0;
        if (A.r < B.r) swap(A, B), swap(a, b);
        double d2 = (A.o.x - B.o.x) * (A.o.x - B.o.x) + (A.o.y - B.o.y) * (A.o.y - B.o.y);
        double rcha = A.r - B.r;
        double rsum = A.r + B.r;
        if (dcmp(d2 - rcha * rcha) < 0) return 0; //内含
        double bas = atan2(B.o.y - A.o.y, B.o.x - A.o.x);
        if (dcmp(d2) == 0 && dcmp(A.r - B.r) == 0) return -1; //无数条切线
        if (dcmp(d2 - rcha * rcha) == 0) {
            //内含，一条切线
            a[cnt] = A.get_point(bas);
            b[cnt++] = B.get_point(bas);
            return 1;
        }
        double ang = acos((A.r - B.r) / sqrt(d2));
        a[cnt] = A.get_point(bas + ang); b[cnt++] = B.get_point(bas + ang);
        a[cnt] = A.get_point(bas - ang); b[cnt++] = B.get_point(bas - ang);
        if (dcmp(d2 - rsum * rsum) == 0) {
            a[cnt] = A.get_point(bas); b[cnt++] = B.get_point(bas + PI);
        }
        else if (dcmp(d2 - rsum * rsum) > 0) {
            double ang = acos((A.r + B.r) / sqrt(d2));
            a[cnt] = A.get_point(bas + ang); b[cnt++] = B.get_point(PI + bas + ang);
            a[cnt] = A.get_point(bas - ang); b[cnt++] = B.get_point(PI + bas - ang);
        }
        return cnt;
    }
    //圆与线段的交点
    void circle_cross_line(point a, point b, point *ret, int &num) const{
        double x0 = o.x, y0 = o.y, x1 = a.x, y1 = a.y, x2 = b.x, y2 = b.y;
        double dx = x2 - x1, dy = y2 - y1;
        double A = dx * dx + dy * dy, B = 2 * dx * (x1 - x0) + 2 * dy * (y1 - y0);
        double C = (x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 - y0) - r * r;
        double delta = B * B - 4 * A * C;
        num = 0;
        if (dcmp(delta) >= 0) {
            double t1 = (-B - sqrt(delta)) / (2 * A);
            double t2 = (-B + sqrt(delta)) / (2 * A);
            if (dcmp(t1 - 1) <= 0 && dcmp(t1) >= 0)
                ret[num++] = point(x1 + t1 * dx, y1 + t1 * dy);
            if (dcmp(t2 - 1) <= 0 && dcmp(t2) >= 0)
                ret[num++] = point(x1 + t2 * dx, y1 + t2 * dy);
        }
    }
    void Read(){ o.Read(); scanf("%lf", &r); }
};
//计算圆o1与圆o2的交点p1 p2
int circles_intersect(point o1, double r1, point o2, double r2, point &p1, point &p2){
    if (r1 < r2) swap(o1, o2), swap(r1, r2);
    double L = (o1 - o2).module();
    if(fabs(L) < eps) return fabs(r1 - r2) < eps ? -1 : 0;
    if (r1 + r2 < L - eps || L < r1 - r2 - eps) return 0;
    if (fabs(r1 + r2 - L) < eps || fabs(L - r1 + r2) < eps){
        p1 = p2 = o1 + (o2 - o1).norm() * r1;
        return 1;
    }
    double x = (r1 * r1 - r2 * r2 + L * L) / (2 * L), d = sqrt(r1 * r1 - x * x);
    point b = o1 + (o2 - o1).norm() * x, a = (o2 - o1).norm().anticlockwise_orthogonal();
    p1 = b + a * d; p2 = b - a * d;
    return 2;
}
// 两圆相交的面积
double cal_area_sec(const double &ra, const double &d, const double &rb){
    double cosh = (ra * ra + d - rb * rb) / (ra * sqrt(d) * 2);
    double sinh = sqrt(1 - cosh * cosh);
    double area_san = ra * (cosh * ra) * sinh;
    double ang = acos(cosh);
    double area_sec = ra * ra * ang;
    return area_sec - area_san;
}
double circles_cross_area(const point &a, const double &ra, const point &b, const double &rb){
    point p = b - a;
    double d = p.x * p.x + p.y * p.y;
    if (dcmp(sqrt(d) - ra - rb) >= 0) return 0; //相离
    if (dcmp(fabs(ra - rb) - sqrt(d)) >= 0 || dcmp(d) == 0) //内含或相切
        return PI * min(ra, rb) * min(ra, rb);
    return cal_area_sec(ra, d, rb) + cal_area_sec(rb, d, ra);
}
//求直线ab与圆o的交点p1 p2
double line_intersect_circle(const point &a, const point &b, const point &o, const double &r, point &p1, point
```

```
    &p2){
    point fp = lines_intersect(o, point(o.x + a.y - b.y,
        o.y + b.x - a.x), a, b);
    double rtol = o.dist(fp);
    double rtos = fp.on_segment(a, b) ? rtol :
        fmin(o.dist(a), o.dist(b));
    double atob = a.dist(b);
    double fptoe = sqrt(r * r - rtol * rtol) / atob;
    if(rtos > r - eps) return rtos;
    p1 = fp + (a - b) * fptoe;
    p2 = fp + (b - a) * fptoe;
    return rtos;
}
//不大于180度扇形面积，r->a->b逆时针
double sector_area(const point &r, const point &a, const
    point &b, double R){
    double A2 = (r - a) * (r - a), B2 = (r - b) * (r - b),
        C2 = (a - b) * (a - b);
    return R * R * acos((A2 + B2 - C2) * 0.5 / sqrt(A2) /
        sqrt(B2)) * 0.5;
}

//逆时针三角形与圆o交的面积
double triangle_and_circle_intersect_area(const point &o,
    const point &a, const point &b, const double &R){
    double adis = o.dist(a), bdis = o.dist(b);
    if(adis < R + eps && bdis < R + eps) return ((a - o) ^
        (b - o)) * 0.5;
    point ta, tb;
    if(fabs((a - o) ^ (b - o)) < eps) return 0.0;
    double rtos = line_intersect_circle(a, b, o, R, ta, tb);
    if(rtos > R - eps) return sector_area(o, a, b, R);
    if(adis < R + eps) return ((a - o) ^ (tb - o)) * 0.5 +
        sector_area(o, tb, b, R);
    if(bdis < R + eps) return ((ta - o) ^ (b - o)) * 0.5 +
        sector_area(o, a, ta, R);
    return ((ta - o) ^ (tb - o)) * 0.5 + sector_area(o, a,
        ta, R) + sector_area(o, tb, b, R);
}
//简单多边形与圆o交的面积
double simple_polygon_intersect_circle_area(point *p, const
    int &N, const point &o, const double &r){
    double res = 0; p[N] = p[0];
    for(int i = 1; i <= N; ++ i){
        if(((p[i - 1] - o) ^ (p[i] - o)) < -eps) res -=
            triangle_and_circle_intersect_area(o, p[i],
            p[i - 1], r);
        else res += triangle_and_circle_intersect_area(o,
            p[i - 1], p[i], r);
    }
    return fabs(res);
}
//求三点共圆的圆心
point triandcir(const point &a, const point &b, const point
    &c){
    double t = 2 * ((b - a) ^ (c - a)), u = ((a * a) - (b *
        b)) / t, v = ((a * a) - (c * c)) / t;
    return ((a - c) * u - (a - b) *
        v).clockwise_orthogonal();
}
// 三点求圆心
void circle_center(point p0, point p1, point p2, point &cp){
    double a1 = p1.x - p0.x, b1 = p1.y - p0.y, c1 = (a1 *
        a1 + b1 * b1) / 2;
    double a2 = p2.x - p0.x, b2 = p2.y - p0.y, c2 = (a2 *
        a2 + b2 * b2) / 2;
    double d = a1 * b2 - a2 * b1;
    cp.x = p0.x + (c1 * b2 - c2 * b1) / d;
    cp.y = p0.y + (a1 * c2 - a2 * c1) / d;
}
// 两点求圆心
void circle_center(point p0, point p1, point &cp){
    cp.x = (p0.x + p1.x) / 2;
    cp.y = (p0.y + p1.y) / 2;
}
// 点是否在圆内（包括边界）
bool point_in_circle(point p, point cp, double r){
    return dcmp((p - cp).module() - r) <= 0;
}
// 最小圆覆盖
void min_circle_cover(point *a, int n, point &cp, double
    &r){
    r = 0, cp = a[0];
    for (int i = 1; i < n; ++i)
        if (!point_in_circle(a[i], cp, r)) {
            cp = a[i], r = 0;
            for (int j = 0; j < i; ++j)
                if (!point_in_circle(a[j], cp, r)) {
                    circle_center(a[i], a[j], cp);
                    r = (a[j] - cp).module();
                    for (int k = 0; k < j; ++k)
                        if (!point_in_circle(a[k], cp, r)) {
                            circle_center(a[i], a[j], a[k], cp);
                            r = (a[k] - cp).module();
                        }
                }
        }
}
// 圆的k次面积并
circle tp[Maxn << 1];
double calc(const circle &o, const circle &a, const circle
    &b){
    return ((b.ang - a.ang) * sqr(o.r) - ((a.o - o.o) ^
        (b.o - o.o)) + (a.o ^ b.o)) / 2.;
}
bool circle_cross(point a, double ra, point b, double rb,
    point &v1, point &v2) {
    double d = (a - b).module();
    if(dcmp(d - ra - rb) >= 0 || dcmp(fabs(ra - rb) - d) >=
        0) return 1;
    double da = (ra * ra + d * d - rb * rb) / (2 * ra * d);
    double aa = atan2((b - a).y, (b - a).x);
    double rad = acos(da);
    v1 = point(a.x + cos(aa - rad) * ra, a.y + sin(aa -
        rad) * ra);
    v2 = point(a.x + cos(aa + rad) * ra, a.y + sin(aa +
        rad) * ra);
    return 0;
}
double area[Maxn]; //大于等于k次的面积并
void CirUnion(circle cir[], const int &N){
    circle res1, res2;
    sort(cir, cir + N, [](const circle &a, const circle
        &b){ return dcmp(a.r - b.r) < 0; });
    for(int i = 0; i < N; ++i)
        for(int j = i + 1; j < N; ++j)
            if (dcmp((cir[i].o - cir[j].o).module() +
                cir[i].r - cir[j].r) <= 0)
                cir[i].d++;
    for(int i = 0; i < N; ++i){
        int tn = 0, cnt = 0;
        for(int j = 0; j < N; ++j){
            if(i == j) continue;
            if(circle_cross(cir[i].o, cir[i].r, cir[j].o,
                cir[j].r, res1.o, res2.o))
                continue;
            res1.ang = atan2(res1.o.y - cir[i].o.y, res1.o.x
                - cir[i].o.x);
            res2.ang = atan2(res2.o.y - cir[i].o.y, res2.o.x
                - cir[i].o.x);
            res1.d = 1, tp[tn++] = res1;
            res2.d = -1, tp[tn++] = res2;
            if (dcmp(res1.ang - res2.ang) > 0) cnt++;
        }
        tp[tn++] = circle(point(cir[i].o.x - cir[i].r,
            cir[i].o.y), 0, PI, -cnt);
        tp[tn++] = circle(point(cir[i].o.x - cir[i].r,
            cir[i].o.y), 0, -PI, cnt);
        sort(tp, tp + tn, [](const circle &a, const circle
            &b){
            return dcmp(a.ang - b.ang) < 0 || (dcmp(a.ang -
                b.ang) == 0 && a.d > b.d);
        });
        int p, s = cir[i].d + tp[0].d;
        for(int j = 1; j < tn; ++j){
            p = s; s += tp[j].d;
            area[p] += calc(cir[i], tp[j - 1], tp[j]);
        }
    }
```

```
}
struct line{
    double a, b, c;
    line(double a = 0, double b = 0, double c = 0) : a(a), b(b), c(c) {}
    //判断是否与直线l相交
    bool check_lines_intersect(const line &l) const{
        return fabs(point(a, b) ^ point(l.a, l.b)) < eps ? false : true;
    }
    //判断是否与直线l重合
    bool check_lines_coincidence(const line &l) const{
        return fabs(point(a, b) ^ point(l.a, l.b)) < eps && fabs(point(a, c) ^ point(l.a, l.c)) < eps && fabs(point(b, c) ^ point(l.b, l.c)) < eps;
    }
    //求与直线l的交点
    point lines_intersect(const line &l) const{
        double d = point(a, b) ^ point(l.a, l.b);
        return point((point(b, c) ^ point(l.b, l.c)) / d, (point(c, a) ^ point(l.c, l.a)) / d);
    }
    //求关于点p对称的直线
    line symmetric_line_about_a_point(const point &p) const{
        return line(a, b, -2 * a * p.x - 2 * b * p.y - c);
    }
};
//求圆o1与圆o2的公切线
inline void tangent_line(const point &o1, const double &r1, const point &o2, const double &r2, line &l){
    point o = o2 - o1; double d = o * o;
    point p(r2 - r1, sqrt(abs(d - (r2 - r1) * (r2 - r1))));
    l.a = (p * o) / d; l.b = (p ^ o) / d; l.c = r1 - l.a * o1.x - l.b * o1.y;
}
void circles_tangent_line(const point &o1, const double &r1, const point &o2, const double &r2, line &l1, line &l2, line &l3, line &l4){
    tangent_line(o1, r1, o2, r2, l1);
    tangent_line(o1, r1, o2, -r2, l2);
    tangent_line(o1, -r1, o2, r2, l3);
    tangent_line(o1, -r1, o2, -r2, l4);
}
struct Line{
    point p, v;
    double ang;
    Line() {}
    Line(point p, point v) : p(p), v(v) {
        ang = atan2(v.y, v.x);
    }
    bool operator < (const Line &b) const{
        return ang < b.ang;
    }
};
// 点p在有向直线L的左边（线上不算）
bool onleft(Line L, point p){
    return (L.v ^ (p - L.p)) > 0;
}
// 二直线交点
point get_line_intersection(Line a, Line b) {
    point u = a.p - b.p;
    double t = (b.v ^u) / (a.v ^ b.v);
    return a.p + a.v * t;
}
int half_plane_intersection(Line *L, int n, point *poly) {
    sort(L, L + n);
    int first, last;
    point *p = new point[n];
    Line *q = new Line[n];
    q[first = last = 0] = L[0];
    for (int i = 1; i < n; ++i) {
        while (first < last && !onleft(L[i], p[last - 1]))
            last--;
        while (first < last && !onleft(L[i], p[first]))
            first++;
        q[++last] = L[i];
        if (dcmp((q[last].v ^ q[last - 1].v)) == 0) {
            last--;
            if (onleft(q[last], L[i].p)) q[last] = L[i];
        }
        if (first < last)
            p[last - 1] = get_line_intersection(q[last - 1], q[last]);
    }
    while (first < last && !onleft(q[first], p[last - 1]))
        last--;
    if (last - first <= 1) return 0;
    p[last] = get_line_intersection(q[last], q[first]);
    int m = 0;
    for (int i = first; i <= last; ++i) poly[m++] = p[i];
    return m;
}
// 旋转卡壳求最远点对
double rotate_calipers(point *ch, int n) {
    int j = 1;
    double ret = 0;
    ch[n] = ch[0];
    for (int i = 0; i < n; ++i) {
        while (fabs((ch[i + 1] - ch[i]) ^ (ch[j + 1] - ch[i])) > fabs((ch[i + 1] - ch[i]) ^ (ch[j] - ch[i])))
            j = (j + 1) % n;
        ret = max(ret, max((ch[i] - ch[j]).module(), (ch[i + 1] - ch[j]).module()));
    }
    return ret;
}
// 求平面点集最小的三角形(先按照x轴排序)
point p[Maxn], q[Maxn];
bool cmpY(const point &a, const point &b) {
    return a.y < b.y || (a.y == b.y && a.x < b.x);
}
double solve1(int L, int R) {
    if (R - L + 1 <= 6) {
        double ret = 1e20;
        for (int i = L; i <= R; ++i) {
            for (int j = i + 1; j <= R; ++j) {
                double len1 = (p[i] - p[j]).module(), len2;
                if (len1 > ret / 2) continue;
                for (int k = j + 1; k <= R; ++k) {
                    len2 = (p[i] - p[k]).module() + (p[j] - p[k]).module();
                    ret = min(ret, len1 + len2);
                }
            }
        }
        return ret;
    }
    int m = (L + R) >> 1, cnt = 0;
    double d = min(solve1(L, m), solve1(m + 1, R));
    for (int i = L; i <= R; ++i)
        if (fabs(p[m].x - p[i].x) <= d)
            q[cnt++] = p[i];
    sort(q, q + cnt, cmpY);
    for (int i = 0; i < cnt; ++i) {
        for (int j = i + 1; j < cnt && j < i + 7; ++j) {
            double len1 = (q[i] - q[j]).module(), len2;
            if (len1 > d / 2) continue;
            for (int k = j + 1; k < cnt && k < j + 7; ++k) {
                len2 = (q[i] - q[k]).module() + (q[j] - q[k]).module();
                d = min(d, len1 + len2);
            }
        }
    }
    return d;
}
// 平面最近点对(先按照x轴排序)
double solve2(int L, int R) {
    if (R - L <= 3) {
        double ret = 1e20;
        for (int i = L; i <= R; ++i)
            for (int j = i + 1; j <= R; ++j)
                ret = min(ret, (p[i] - p[j]).module());
        return ret;
    }
    int m = (L + R) >> 1, cnt = 0;
    double ret = min(solve2(L, m), solve2(m + 1, R));
    for (int i = L; i <= R; ++i)
        if (fabs(p[m].x - p[i].x) <= ret)
```

```
            q[cnt++] = p[i];
    sort(q, q + cnt, cmpY);
    for (int i = 0; i < cnt; ++i)
        for (int j = i + 1; j < cnt && (q[j].y - q[i].y) <= ret; ++j)
            ret = min(ret, (q[j] - q[i]).module());
    return ret;
}
}
// 多边形面积并
struct polygon {
    point p[N];  //N比较小(N<=50)
    int sz;
    void input() {
        for (int i = 0; i < sz; ++i)
            p[i].input();
        if (dcmp(polygon_area(p, sz)) < 0)
            reverse(p, p + sz);
        p[sz] = p[0];
    }
}g[5];
pair<double, int> C[100020];
double segP(point a, point b, point c) {
    if (dcmp(b.x - c.x))
        return (a.x - b.x) / (c.x - b.x);
    return (a.y - b.y) / (c.y - b.y);
}
double polyUnion(int n) { //n是多边形的数目
    double sum = 0;
    for (int i = 0; i < n; ++i)
        for (int ii = 0; ii < g[i].sz; ++ii) {
            int tot = 0;
            C[tot++] = MP(0, 0);
            C[tot++] = MP(1, 0);
            for (int j = 0; j < n; ++j) if (i != j)
                for (int jj = 0; jj < g[j].sz; ++jj) {
                    int d1 = dcmp(cross(g[i].p[ii + 1] - g[i].p[ii], g[j].p[jj] - g[i].p[ii]));
                    int d2 = dcmp(cross(g[i].p[ii + 1] - g[i].p[ii], g[j].p[jj + 1] - g[i].p[ii]));
                    if (!d1 && !d2) {
                        point t1 = g[j].p[jj + 1] - g[j].p[jj];
                        point t2 = g[i].p[ii + 1] - g[i].p[ii];
                        if (dcmp(dot(t1, t2)) > 0 && j < i) {
                            C[tot++] = MP(segP(g[j].p[jj], g[i].p[ii], g[i].p[ii + 1]), 1);
                            C[tot++] = MP(segP(g[j].p[jj + 1], g[i].p[ii], g[i].p[ii + 1]), -1);
                        }
                    }
                    else if (d1 >= 0 && d2 < 0 || d1 < 0 && d2 >= 0) {
                        double d3 = cross(g[j].p[jj + 1] - g[j].p[jj], g[i].p[ii] - g[j].p[jj]);
                        double d4 = cross(g[j].p[jj + 1] - g[j].p[jj], g[i].p[ii + 1] - g[j].p[jj]);
                        if (d2 < 0)
                            C[tot++] = MP(d3 / (d3 - d4), 1);
                        else C[tot++] = MP(d3 / (d3 - d4), -1);
                    }
                }
            sort(C, C + tot);
            double cur = min(max(C[0].first, 0.0), 1.0);
            int sgn = C[0].second;
            double s = 0;
            for (int j = 1; j < tot; ++j) {
                double nxt = min(max(C[j].first, 0.0), 1.0);
                if (!sgn) s += nxt - cur;
                sgn += C[j].second;
                cur = nxt;
            }
            sum += cross(g[i].p[ii], g[i].p[ii + 1]) * s;
        }
    return fabs(sum) / 2;
}
// 判断射线与线段是否相交（s1，e1射线；s2，e2线段）
bool intersect(point s1, point e1, point s2, point e2) {
    double a, t1, t2;
    a = cross(s2 - s1, s2 - e1) - cross(e2 - s1, e2 - e1);
    if (fabs(a) < eps) return false;
    t1 = cross(s2 - s1, s2 - e1) / a;
    t2 = cross(s2 - s1, s2 - e2) / a;
    return (t1 > -eps && t1 < 1 + eps && t2 > -eps);
}


//三角剖分
struct point {
    double x, y;
    int id;
    struct Edge *e;
    bool operator < (const point & p) const {
        return dcmp(x - p.x) != 0 ? x < p.x : dcmp(y - p.y) < 0;
    }
    bool operator == (const point & p) const {
        return dcmp(x - p.x) == 0 && dcmp(y - p.y) == 0;
    }
    void input(int i) {
        id = i;
        scanf("%lf%lf", &x, &y);
    }
}pnt[N];
double cross(point & o, point & a, point & b) {
    return (a.x - o.x) * (b.y - o.y) - (b.x - o.x) * (a.y - o.y);
}
double dot(point & o, point & a, point & b) {
    return (a.x - o.x) * (b.x - o.x) + (a.y - o.y) * (b.y - o.y);
}
double dis(point a, point b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
struct Edge {
    point *o, *d;
    Edge *on, *op, *dn, *dp;
};
#define Op(e,p) ((e)->o==p?(e)->d:(e)->o)
#define Next(e,p) ((e)->o==p?(e)->on:(e)->dn)
#define Prev(e,p) ((e)->o==p?(e)->op:(e)->dp)
struct Delaunay {
    void solve(point * ps, int n) { //点集需要 sort 和 unique
        sort(ps, ps + n);
        edge_num = 0;
        rubb = NULL;
        for (int i = 0; i < n; i++) ps[i].e = NULL;
        Edge* l_cw, *r_ccw;
        divide(ps, 0, n, l_cw, r_ccw);
    }
    Edge es[M], *rubb;
    int edge_num;
    Edge *make_edge(point &u, point &v) {
        Edge * e;
        if (rubb == NULL) {
            e = es + edge_num++;
        }
        else {
            e = rubb;
            rubb = rubb->dn;
        }
        e->on = e->op = e->dn = e->dp = e;
        e->o = &u; e->d = &v;
        if (u.e == NULL) u.e = e;
        if (v.e == NULL) v.e = e;
        return e;
    }
    void delete_edge(Edge *e) {
        point *u = e->o, *v = e->d;
        if (u->e == e) u->e = e->on;
        if (v->e == e) v->e = e->dn;
        Prev(e->on, u) = e->op;
        Next(e->op, u) = e->on;
```

```
        Prev(e->dn, v) = e->dp;
        Next(e->dp, v) = e->dn;
        e->dn = rubb;
        rubb = e;
    }
    void splice(Edge *a, Edge *b, point *v) {
        Edge *n;
        n = Next(a, v); Next(a, v) = b;
        Prev(n, v) = b;
        Next(b, v) = n; Prev(b, v) = a;
    }
    Edge *join(Edge *a, point *u, Edge *b, point *v, int s) {
        Edge *e = make_edge(*u, *v);
        if (s == 0) {
            splice(Prev(a, u), e, u);
            splice(b, e, v);
        }
        else {
            splice(a, e, u);
            splice(Prev(b, v), e, v);
        }
        return e;
    }
    void lower_tangent(Edge * & l, Edge * & r, point * & s, point * & u) {
        point *dl = Op(l, s), *dr = Op(r, u);
        while (1) {
            if (dcmp(cross((*s), (*dl), (*u))) > 0) {
                l = Prev(l, dl); s = dl; dl = Op(l, s);
            }
            else if (dcmp(cross((*u), (*dr), (*s))) < 0) {
                r = Next(r, dr); u = dr; dr = Op(r, u);
            }
            else break;
        }
    }
    void merge(Edge *r_cw_l, point *s, Edge *l_ccw_r, point *u, Edge
        **l_tangent) {
        Edge *b, *lc, *rc;
        point *dlc, *drc;
        double crc, clc;
        lower_tangent(r_cw_l, l_ccw_r, s, u);
        b = join(r_cw_l, s, l_ccw_r, u, 1);
        *l_tangent = b;
        do {
            lc = Next(b, s); rc = Prev(b, u); dlc = Op(lc, s); drc = Op(rc, u);
            double cplc = cross(*dlc, *s, *u);
            double cprc = cross(*drc, *s, *u);
            bool alc = dcmp(cplc)>0, arc = dcmp(cprc)>0;
            if (!alc && !arc) break;
            if (alc) {
                clc = dot(*dlc, *s, *u) / cplc;
                do {
                    Edge * next = Next(lc, s);
                    point & dest = *Op(next, s);
                    double cpn = cross(dest, *s, *u);
                    if (dcmp(cpn) <= 0) break;
                    double cn = dot(dest, *s, *u) / cpn;
                    if (dcmp(cn - clc)>0) break;
                    delete_edge(lc);
                    lc = next;
                    clc = cn;
                } while (1);
            }
            if (arc) {
                crc = (double)dot(*drc, *s, *u) / cprc;
                do {
                    Edge * prev = Prev(rc, u);
                    point & dest = *Op(prev, u);
                    double cpp = cross(dest, *s, *u);
                    if (dcmp(cpp) <= 0) break;
                    double cp = dot(dest, *s, *u) / cpp;
                    if (dcmp(cp - crc) > 0) break;
                    delete_edge(rc);
                    rc = prev;
                    crc = cp;
                } while (1);
            }
            dlc = Op(lc, s); drc = Op(rc, u);
            if (!alc || (alc && arc && dcmp(crc - clc) < 0))
                {
                b = join(b, s, rc, drc, 1);
                u = drc;
            }
            else {
                b = join(lc, dlc, b, u, 1);
                s = dlc;
            }
        } while (1);
    }
    void divide(point *p, int l, int r, Edge * & l_ccw,
        Edge * & r_cw) {
        int n = r - l;
        Edge *l_ccw_l, *r_cw_l, *l_ccw_r, *r_cw_r,
            *l_tangent, *c;
        if (n == 2) {
            l_ccw = r_cw = make_edge(p[l], p[l + 1]);
        }
        else if (n == 3) {
            Edge * a = make_edge(p[l], p[l + 1]), *b =
                make_edge(p[l + 1], p[l + 2]);
            splice(a, b, &p[l + 1]);
            double c_p = cross(p[l], p[l + 1], p[l + 2]);
            if (dcmp(c_p)>0) {
                c = join(a, &p[l], b, &p[l + 2], 1); l_ccw =
                    a; r_cw = b;
            }
            else if (dcmp(c_p) < 0) {
                c = join(a, &p[l], b, &p[l + 2], 0); l_ccw =
                    c; r_cw = c;
            }
            else {
                l_ccw = a; r_cw = b;
            }
        }
        else if (n > 3) {
            int split = (l + r) / 2;
            divide(p, l, split, l_ccw_l, r_cw_l);
            divide(p, split, r, l_ccw_r, r_cw_r);
            merge(r_cw_l, &p[split - 1], l_ccw_r, &p[split],
                &l_tangent);
            if (l_tangent->o == &p[l]) l_ccw_l = l_tangent;
            if (l_tangent->d == &p[r - 1]) r_cw_r =
                l_tangent;
            l_ccw = l_ccw_l; r_cw = r_cw_r;
        }
    }
} de;
void getEdge(int &k, int n) {
    k = 0;
    Edge *st, *cur;
    point *u, *v;
    for (int i = 0; i < n; ++i) {
        u = &pnt[i];
        st = cur = u->e;
        do {
            v = Op(cur, u);
            if (u < v)
                addEdge(k, u->id, v->id, dis(*u, *v));
        } while ((cur = Next(cur, u)) != st);
    }

}
void enum_triangle(point *ps, int n) {
    Edge *e_start, *e, *nxt;
    point *u, *v, *w;
    for (int i = 0; i < n; i++) {
        u = &ps[i];
        e_start = e = u->e;
        do {
            v = Op(e, u);
            if (u < v) {
                nxt = Next(e, u);
                w = Op(nxt, u);
                if (u < w && Next(nxt, w) == Prev(e, v)) {
                    // now, (u v w) is a triangle!!!!!!
                    // 这时，uvw 的外接圆是空的（不包含 ps
                        中的其他点），如果要求最大空圆，则计算
```

```
                    uvw 的外接圆就可以!
            }
        }
    } while ((e = Next(e, u)) != e_start);
  }
}
```

1.2 3D Convex

```
const int MAXN = 100;
const double EPS = 1e-8;
struct Point {
    double x, y, z;
    Point() {}
    Point(double xx, double yy, double zz): x(xx), y(yy), z(zz) {}
    Point operator -(const Point p1) {
        return Point(x - p1.x, y - p1.y, z - p1.z);
    }
    Point operator *(Point p) {
        return Point(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y * p.x);
    }
    double operator ^(Point p) {
        return (x * p.x + y * p.y + z * p.z);
    }
};
struct CH3D {
    struct face {
        int a, b, c;
        bool ok;
    };
    int n;
    Point P[MAXN];
    int num;
    face F[8 * MAXN];
    int g[MAXN][MAXN];
    double vlen(Point a) {
        return sqrt(a.x * a.x + a.y * a.y + a.z * a.z);
    }
    Point cross(const Point &a, const Point &b, const Point &c) {
        return Point((b.y - a.y) * (c.z - a.z) - (b.z - a.z) * (c.y - a.y), -((b.x - a.x) * (c.z - a.z) - (b.z - a.z) * (c.x - a.x)), (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x));
    }
    double area(Point a, Point b, Point c) {
                                //*2
        return vlen((b - a) * (c - a));
    }
    double volume(Point a, Point b, Point c, Point d) {
                //*6
        return (b - a) * (c - a) ^ (d - a);
    }
    double dblcmp(Point &p, face &f) {
        Point m = P[f.b] - P[f.a];
        Point n = P[f.c] - P[f.a];
        Point t = p - P[f.a];
        return (m * n)^t;
    }
    void deal(int p, int a, int b) {
        int f = g[a][b];
        face add;
        if(F[f].ok) {
            if(dblcmp(P[p], F[f]) > EPS)
                dfs(p, f);
            else {
                add.a = b;
                add.b = a;
                add.c = p;
                add.ok = 1;
                g[p][b] = g[a][p] = g[b][a] = num;
                F[num++] = add;
            }
        }
    }
    void dfs(int p, int now) {
        F[now].ok = 0;
        deal(p, F[now].b, F[now].a);
        deal(p, F[now].c, F[now].b);
        deal(p, F[now].a, F[now].c);
    }
    bool same(int s, int t) {
        Point &a = P[F[s].a];
        Point &b = P[F[s].b];
        Point &c = P[F[s].c];
        return fabs(volume(a, b, c, P[F[t].a])) < EPS &&
               fabs(volume(a, b, c, P[F[t].b])) < EPS
               && fabs(volume(a, b, c, P[F[t].c])) < EPS;
    }
    void solve() {
        int i, j, tmp;
        face add;
        bool flag = true;
        num = 0;
        if(n < 4)
            return;
        for(i = 1; i < n; i++) {
            if(vlen(P[0] - P[i]) > EPS) {
                swap(P[1], P[i]);
                flag = false;
                break;
            }
        }
        if(flag)
            return;
        flag = true;
        for(i = 2; i < n; i++) {
            if(vlen((P[0] - P[1]) * (P[1] - P[i])) > EPS) {
                swap(P[2], P[i]);
                flag = false;
                break;
            }
        }
        if(flag)
            return;
        flag = true;
        for(i = 3; i < n; i++) {
            if(fabs((P[0] - P[1]) * (P[1] - P[2]) ^ (P[0] - P[i])) > EPS) {
                swap(P[3], P[i]);
                flag = false;
                break;
            }
        }
        if(flag)
            return;
        for(i = 0; i < 4; i++) {
            add.a = (i + 1) % 4;
            add.b = (i + 2) % 4;
            add.c = (i + 3) % 4;
            add.ok = true;
            if(dblcmp(P[i], add) > 0)
                swap(add.b, add.c);
            g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] = num;
            F[num++] = add;
        }
        for(i = 4; i < n; i++) {
            for(j = 0; j < num; j++) {
                if(F[j].ok && dblcmp(P[i], F[j]) > EPS) {
                    dfs(i, j);
                    break;
                }
            }
        }
        tmp = num;
        for(i = num = 0; i < tmp; i++)
            if(F[i].ok) {
                F[num++] = F[i];
            }
    }
    double area() {
        double res = 0.0;
        if(n == 3) {
            Point p = cross(P[0], P[1], P[2]);
            res = vlen(p) / 2.0;
            return res;
```

```
        }
        for(int i = 0; i < num; i++)
            res += area(P[F[i].a], P[F[i].b], P[F[i].c]);
        return res / 2.0;
    }
    double volume() {
        double res = 0.0;
        Point tmp(0, 0, 0);
        for(int i = 0; i < num; i++)
            res += volume(tmp, P[F[i].a], P[F[i].b],
                  P[F[i].c]);
        return fabs(res / 6.0);
    }
    int triangle() {
        return num;
    }
    int polygon() {
        int i, j, res, flag;
        for(i = res = 0; i < num; i++) {
            flag = 1;
            for(j = 0; j < i; j++)
                if(same(i, j)) {
                    flag = 0;
                    break;
                }
            res += flag;
        }
        return res;
    }
    Point getcent() {
        Point ans(0, 0, 0), temp = P[F[0].a];
        double v = 0.0, t2;
        for(int i = 0; i < num; i++) {
            if(F[i].ok == true) {
                Point p1 = P[F[i].a], p2 = P[F[i].b], p3 =
                      P[F[i].c];
                t2 = volume(temp, p1, p2, p3) / 6.0;
                if(t2 > 0) {
                    ans.x += (p1.x + p2.x + p3.x + temp.x) *
                          t2;
                    ans.y += (p1.y + p2.y + p3.y + temp.y) *
                          t2;
                    ans.z += (p1.z + p2.z + p3.z + temp.z) *
                          t2;
                    v += t2;
                }
            }
        }
        ans.x /= (4 * v);
        ans.y /= (4 * v);
        ans.z /= (4 * v);
        return ans;
    }
    double function(Point fuck) {
        double min = 99999999;
        for(int i = 0; i < num; i++) {
            if(F[i].ok == true) {
                Point p1 = P[F[i].a], p2 = P[F[i].b], p3 =
                      P[F[i].c];
                double a = ( (p2.y - p1.y) * (p3.z - p1.z) -
                      (p2.z - p1.z) * (p3.y - p1.y) );
                double b = ( (p2.z - p1.z) * (p3.x - p1.x) -
                      (p2.x - p1.x) * (p3.z - p1.z) );
                double c = ( (p2.x - p1.x) * (p3.y - p1.y) -
                      (p2.y - p1.y) * (p3.x - p1.x) );
                double d = ( 0 - (a * p1.x + b * p1.y + c *
                      p1.z) );
                double temp = fabs(a * fuck.x + b * fuck.y +
                      c * fuck.z + d) / sqrt(a * a + b * b + c
                      * c);
                if(temp < min)min = temp;
            }
        }
        return min;
    }
};
```

1.3 Dynamic Convex

```
map < int , int > Convex[2] ;
inline long long Cross( int x1 , int y1 , int x2 , int y2 ){
    return 1LL * x1 * y2 - 1LL * x2 * y1;
}
bool Check( map < int , int > & cov , int x , int y ){
    if( cov.size() == 0 ) return false;
    if( cov.find( x ) != cov.end() ) return y >= cov[x];
    if( x < cov.begin()->first || x > (--cov.end())->first
          ) return false;
    map < int , int > :: iterator p = cov.lower_bound( x )
          , q = p ; -- q;
    return Cross( q->first - x , q->second - y , p->first -
          x , p ->second - y ) >= 0;
}
void Insert( map < int , int > & cov , int x , int y ){
    if( Check( cov , x , y ) ) return ;
    cov[x] = y; // cov[x] = min( cov[x] , y )
    map < int , int > :: iterator p = cov.upper_bound( x )
          , q;
    while( p != cov.end() ){
        q = p ; ++ q;
        if( q == cov.end() || Cross( p->first - x ,
              p->second - y , q ->first - p ->first ,
              q->second - p->second ) > 0 ) break;
        cov.erase( p );
        p = q;
    }
    p = cov.find( x );
    if( p == cov.begin() || --p == cov.begin() ) return;
    while( p != cov.begin() ){
        q = p ; -- q;
        if( Cross( p->first - q->first , p->second -
              q->second , x - p->first , y - p->second ) > 0
              ) break;
        cov.erase( p );
        p = q;
    }
}
```

2 DataStructure

2.1 BIT

```
int findkth(int k) {
    int idx = 0;
    for(int i = 20; i >= 0; --i) {
        idx ^= 1 << i;
        if(idx <= N && bit[idx] < k) k -= bit[idx];
        else idx ^= 1 << i;
    }
    return idx + 1;
}
```

2.2 Size Balanced Tree

```
namespace Size_Balanced_Tree{
#define lch(x) (x->ch[0])
#define rch(x) (x->ch[1])
const int Maxn = 100010, Inf = 0x3f3f3f3f;
struct Tree{
    int key, Size; LL sum;
    Tree *ch[2];
    Tree(){ key = -Inf; sum = Size = 0; ch[0] = ch[1] =
          NULL; }
}Aplay[Maxn], *Ap = &Aplay[0];
struct SBT{
    Tree *Root;
    SBT(){ Root = &Aplay[0]; lch(Root) = rch(Root) = Root; }
    inline void Update(Tree* &x){
        x->sum = lch(x)->sum + rch(x)->sum + x->key;
    }
    inline void Rotate(Tree* &x, int d){
        Tree *y = x->ch[d ^ 1];
        x->ch[d ^ 1] = y->ch[d];
        y->ch[d] = x;
        y->Size = x->Size;
```

```
        x->Size = lch(x)->Size + rch(x)->Size + 1;
        Update(x); Update(y);
        x = y;
    }
    void Maintain(Tree* &x, int d){
        if(x->ch[d]->ch[d]->Size > x->ch[d ^ 1]->Size)
            Rotate(x, d ^ 1);
        else if(x->ch[d]->ch[d ^ 1]->Size > x->ch[d ^ 1]->Size)
            Rotate(x->ch[d], d),Rotate(x, d ^ 1);
        else return;
        Maintain(x->ch[0], 0);
        Maintain(x->ch[1], 1);
        Maintain(x, 0);
        Maintain(x, 1);
    }
    void Insert(Tree* &x, int key){
        if(!x->Size){
            x = ++Ap; x->key = key; x->Size = 1; x->sum = x->key;
            lch(x) = rch(x) = &Aplay[0];
            return;
        }
        ++x->Size;
        if(key < x->key) Insert(lch(x), key);
        else Insert(rch(x), key);
        Update(x);
        Maintain(x,key >= x->key);
    }
    int Remove(Tree* &x, int key){
        int Dkey;
        --x->Size;
        if((key == x->key) || (key < x->key && !lch(x)->Size) || (key > x->key && !rch(x)->Size)){
            Dkey = x->key;
            if(lch(x)->Size && rch(x)->Size)
                x->key = Remove(lch(x), x->key + 1);
            else x = lch(x)->Size ? lch(x) : rch(x);
        }
        else if(key > x->key) Dkey = Remove(rch(x), key);
        else if(key < x->key) Dkey = Remove(lch(x), key);
        Update(x);
        return Dkey;
    }
    Tree* Pred(Tree* &x, Tree* y, int key){
        if(!x->Size) return y;
        if(x->key < key) return Pred(rch(x), x, key);
        else return Pred(lch(x), y, key);
    }//Pred(Root,&Aplay[0],key)
    Tree* Succ(Tree* &x, Tree* y, int key){
        if(!x->Size) return y;
        if(x->key > key) return Succ(lch(x), x, key);
        else return Succ(rch(x), y, key);
    }//Succ(Root,&Aplay[0],key)
    int Select(int k){
        int r;
        for(Tree *x = Root; x->Size;){
            r = lch(x)->Size + 1;
            if(r == k) return x->key;
            if(r < k) x = rch(x), k -= r;
            else x = lch(x);
        }
        return -1;
    }//k-th
    int Rank(int key){
        int res = 0;
        for(Tree *x = Root; x->Size;){
            if(x->key < key){
                res += lch(x)->Size + 1;
                x = rch(x);
            }
            else x = lch(x);
        }
        return res + 1;
    }
    LL Query(int key){
        LL res = 0;
        for(Tree *x = Root; x->Size;){
            if(x->key <= key){
                res += lch(x)->sum + x->key;
                x = rch(x);
            }
            else x = lch(x);
        }
        return res;
    }//sum
};
}
```

2.3 Splay

```
namespace Splay{
#define fat(x) (x->fa)
#define lch(x) (x->ch[0])
#define rch(x) (x->ch[1])
struct Tree{
    int key,point_size,tree_size;
    Tree *ch[2],*fa;
    Tree(){ key=tree_size=point_size=0;
        ch[0]=ch[1]=fa=NULL; }
}Aplay[MAXN],*Ap=&Aplay[0];

struct SPLAY{
    Tree *Root;
    SPLAY(){ Root=NULL; }
    void Update(Tree *x){
        x->tree_size=x->point_size;
        if(lch(x)) x->tree_size+=lch(x)->tree_size;
        if(rch(x)) x->tree_size+=rch(x)->tree_size;
    }
    void Rotate(Tree *x){
        Tree *y=fat(x);
        int d= x==lch(y);
        y->ch[d^1]=x->ch[d];
        if(x->ch[d]) fat(x->ch[d])=y;
        fat(x)=fat(y);
        if(y->fa) fat(y)->ch[rch(fat(y))==y]=x;
        fat(y)=x; x->ch[d]=y;
        Update(y);
    }
    void Splay(Tree *x,Tree *Stop){
        for(Tree *y=fat(x);y!=Stop;y=fat(x)){
            if(fat(y)!=Stop)
                Rotate((x==lch(y))==(y==lch(fat(y))) ?y:x);
            Rotate(x);
        }
        Update(x);
        if(!Stop) Root=x;
    }
    void Search(int key){
        Tree *x;
        for(x=Root;x;){
            if(x->key==key) break;
            if(x->key > key){ if(!lch(x)) break; x=lch(x); }
            else{ if(!rch(x)) break; x=rch(x); }
        }
        if(x) Splay(x,NULL);
    }
    void Insert(int key){
        Search(key);
        if(Root && Root->key == key) ++Root->point_size;
        else{
            Tree *x=++Ap; x->key=key; x->point_size=1;
            if(Root){
                int d=Root->key > key;
                x->ch[d]=Root; fat(Root)=x;
                x->ch[d^1]=Root->ch[d^1];
                if(Root->ch[d^1]) fat(Root->ch[d^1])=x;
                Root->ch[d^1]=NULL; Update(Root);
            }
            Root=x;
        }
        Update(Root);
    }
    void Delete(int key){
        Search(key);
        if(!(--Root->point_size)){
            int d= lch(Root)!=NULL;
            Tree *x=Root->ch[d];
```

```
            Root=Root->ch[d^1];
            if(!Root) return;
            fat(Root)=NULL; Search(key);
            Root->ch[d]=x; (x)&&(fat(x)=Root);
        }
        Update(Root);
    }
    int Kth(int k){
        int lcnt; Tree *x;
        for(x=Root;x;){
            lcnt=lch(x)?lch(x)->tree_size:0;
            if(k<=lcnt) x=lch(x);
            else if(k<=lcnt+x->point_size) return x->key;
            else k-=x->point_size+lcnt,x=rch(x);
        }
        return -1;
    }
    int Rank(int key){
        if(!Root) return 0;
        return Search(key),
              (lch(Root)?lch(Root)->tree_size:0)+1;
    }
    Tree *Pre(int key){
        if(!Root) return NULL;
        return Search(key), Root->key <
              key?Root:Find(lch(Root),1);
    }
    Tree *Sub(int key){
        if(!Root) return NULL;
        return Search(key), Root->key >
              key?Root:Find(rch(Root),0);
    }
    Tree* Find(Tree *x,int d){
        if(!x) return NULL;
        while(x->ch[d]) x=x->ch[d];
        return x;
    }
};
}
```

2.4 Treap

```
const int N = 100005;
struct Treap {
    int key, val, sz;
    Treap *lc, *rc;
} pool[N], *nill, *root;
int tot;
void init() {
    srand(0);
    root = nill = pool;
    nill->sz = 0;
    tot = 0;
}
Treap* newnode(int v) {
    Treap *t = pool + (++tot);
    t->val = v;
    t->sz = 1;
    t->key = (rand() << 16) | rand();
    t->lc = t->rc = nill;
    return t;
}
inline void push_up(Treap *p) {
    p->sz = p->lc->sz + p->rc->sz + 1;
}
Treap* Merge(Treap *a, Treap *b) {
    if(a == nill) return b;
    if(b == nill) return a;
    if(a->key < b->key) {
        a->rc = Merge(a->rc, b);
        push_up(a);
        return a;
    }
    else {
        b->lc = Merge(a, b->lc);
        push_up(b);
        return b;
    }
}
```

```
typedef pair <Treap*, Treap*> pii;
pii Split(Treap *a, int k) {
    if(!k) return make_pair(nill, a);
    int cnt = a->lc->sz;
    if(cnt >= k) {
        pii u = Split(a->lc, k);
        a->lc = u.SE;
        push_up(a);
        return make_pair(u.FI, a);
    }
    else {
        pii u = Split(a->rc, k - cnt - 1);
        a->rc = u.FI;
        push_up(a);
        return make_pair(a, u.SE);
    }
}
int get_rank(int k) {
    Treap *p = root;
    int res = 1;
    while(p != nill) {
        if(p->val < k) {
            res += p->lc->sz + 1;
            p = p->rc;
        }
        else p = p->lc;
    }
    return res;
}
int get_kth(int k) {
    Treap *p = root;
    while(p->lc->sz + 1 != k) {
        if(p->lc->sz + 1 < k) {
            k -= p->lc->sz + 1;
            p = p->rc;
        }
        else p = p->lc;
    }
    return p->val;
}
int get_pre(int k) {
    int res;
    Treap *p = root;
    while(p != nill) {
        if(p->val < k) {
            res = p->val;
            p = p->rc;
        }
        else p = p->lc;
    }
    return res;
}
int get_nxt(int k) {
    int res;
    Treap *p = root;
    while(p != nill) {
        if(p->val > k) {
            res = p->val;
            p = p->lc;
        }
        else p = p->rc;
    }
    return res;
}
void Insert(int k) {
    Treap *t = newnode(k);
    pii u = Split(root, get_rank(k) - 1);
    root = Merge(u.FI, t);
    root = Merge(root, u.SE);
}
void Delete(int k) {
    int p = get_rank(k);
    pii a = Split(root, p - 1);
    pii b = Split(a.SE, 1);
    root = Merge(a.FI, b.SE);
}
```

2.5 Persistent Treap

```
namespace Treap{
const int maxn = 3e6 + 50;
int ch[maxn][2] , key[maxn] , weight[maxn] , tot , sz[maxn];
void Init(){ tot = 0; }
int NewNode( int l , int r , int _key , int _weight ){
    int ret = ++ tot;
    ch[ret][0] = l , ch[ret][1] = r , key[ret] = _key ,
        weight[ret] = _weight;
    sz[ret] = 1 + sz[l] + sz[r];
    return ret;
}
int Merge( int x , int y ){
    if( !x || !y ) return x ? x : y;
    return weight[x] < weight[y] ? NewNode( ch[x][0] ,
        Merge( ch[x][1] , y ) , key[x] , weight[x] ) :
        NewNode( Merge( x , ch[y][0] ) , ch[y][1] , key[y]
        , weight[y] );
}
int Split_l( int x , int _key ){
    if( !x ) return 0;
    return key[x] <= _key ? NewNode( ch[x][0] , Split_l(
        ch[x][1] , _key ) , key[x] , weight[x] ) :
        Split_l( ch[x][0] , _key );
}
int Split_r( int x , int _key ){
    if( !x ) return 0;
    return key[x] > _key ? NewNode( Split_r( ch[x][0] ,
        _key ) , ch[x][1] , key[x] , weight[x] ) :
        Split_r( ch[x][1] , _key );
}
int Insert( int root , int y ){
    return Merge( Split_l( root , y ) , Merge( NewNode( 0 ,
        0 , y , rand() ) , Split_r( root , y ) ) );
}
}
```

2.6 Link Cut Tree

```
namespace Link_Cut_Tree{
#define fat(x) (x->fa)
#define FAT(x) (x->Fa)
#define lch(x) (x->ch[0])
#define rch(x) (x->ch[1])
#define lzy(x) (x->lazy)
const int Maxn = 100010;
struct Tree{
    int val, lazy;
    Tree *fa, *ch[2], *Fa, *Mpoint;
    Tree(){
        val = 0; lazy = 0;
        fa = Fa = ch[0] = ch[1] = NULL;
        Mpoint=this;
    }
}Aplay[Maxn];
inline void Update(Tree *x){
    x->Mpoint = x;
    if(lch(x)) x->Mpoint = x->Mpoint->val >
        lch(x)->Mpoint->val ? x->Mpoint : lch(x)->Mpoint;
    if(rch(x)) x->Mpoint = x->Mpoint->val >
        rch(x)->Mpoint->val ? x->Mpoint : rch(x)->Mpoint;
}
inline void Reverse(Tree *x){
    swap(lch(x), rch(x));
    if(lch(x)) lzy(lch(x)) ^= 1;
    if(rch(x)) lzy(rch(x)) ^= 1;
    lzy(x) = 0;
}
void Rotate(Tree *x){
    Tree *y = fat(x);
    if(lzy(y)) Reverse(y);
    if(lzy(x)) Reverse(x);
    int d = x == lch(y);
    y->ch[d ^ 1] = x->ch[d];
    if(x->ch[d]) fat(x->ch[d]) = y;
    fat(x) = fat(y); FAT(x) = FAT(y);
    if(y->fa) fat(y)->ch[rch(fat(y)) == y] = x;
    fat(y) = x; x->ch[d] = y;
    Update(y);
}
void Splay(Tree *x, Tree *Spot){
    if(lzy(x)) Reverse(x);
    for(Tree *y = fat(x); y != Spot; y = fat(x)){
        if(fat(y) != Spot)
            Rotate((x == lch(y)) == (y == lch(fat(y))) ? y :
                x);
        Rotate(x);
    }
    Update(x);
}
void Access(Tree *x){
    Tree *y; Splay(x, NULL);
    if(rch(x)){
        FAT(rch(x)) = x;
        rch(x) = fat(rch(x)) = NULL;
        Update(x);
    }
    for(y = FAT(x); y; y = FAT(x)){
        Splay(y, NULL);
        if(rch(y)) FAT(rch(y)) = y, fat(rch(y)) = NULL;
        rch(y) = x; fat(x) = y;
        Update(y); Splay(x, NULL);
    }
}
void Query(int u,int v){
    Access(&Aplay[u]); Aplay[u].lazy^=1;
    Access(&Aplay[v]);
    printf("%d\n",Aplay[v].Mpoint->val);
}
}
```

2.7 Leftist Tree

```
const int N = 100005;
struct LHeap {
    int dis, key;
    LHeap *lc, *rc;
} pool[N], *nill;
int tot;
inline void init() {
    tot = 0;
    nill = pool;
    nill->dis = -1;
}
inline LHeap* MakeTree(int v) {
    LHeap *t = pool + (++tot);
    t->lc = t->rc = nill;
    t->dis = 0;
    t->key = v;
    return t;
}
LHeap* Merge(LHeap *a, LHeap *b) {
    if(a == nill) return b;
    if(b == nill) return a;
    if(a->key > b->key) swap(a, b);
    a->rc = Merge(a->rc, b);
    if(a->rc->dis > a->lc->dis) swap(a->rc, a->lc);
    a->dis = a->rc->dis + 1;
    return a;
}
inline void Insert(LHeap* &a, int v) {
    LHeap *b = MakeTree(v);
    a = Merge(a, b);
}
inline int DeleteMin(LHeap* &a) {
    int t = a->key;
    a = Merge(a->lc, a->rc);
    return t;
}
```

2.8 Manhattan Distance MST

```
struct Point {
    int x, y, id;
} po[10005];
int data[10005], cc;
struct Edge {
    int u, v, l;
```

```
} ed[50005];
int ecnt = 0;
inline int Find( int x ) {
    return lower_bound( data, data + cc, x ) - data + 1;
}
inline bool cmp( Point a, Point b ) {
    return a.x > b.x || ( a.x == b.x && a.y > b.y );
}
inline int AB( int x ) {
    return x > 0 ? x : -x;
}
inline int Dis( Point a, Point b ) {
    return AB( a.x - b.x ) + AB( a.y - b.y );
}
inline void addedge( int u, int v, int l ) {
    ed[ecnt].u = u; ed[ecnt].v = v; ed[ecnt++].l = l;
}
int bitv[10005], bitid[10005];
inline void add( int x, int v, int id ) {
    x = cc - x + 1;
    for( ; x <= cc; x += x & -x ) if( bitv[x] > v ) {
            bitv[x] = v; bitid[x] = id;
    }
}
inline int read( int x ) {
    int v = INF, id = -1;
    x = cc - x + 1;
    for( ; x; x ^= x & -x ) if( bitv[x] < v ) {
            v = bitv[x]; id = bitid[x];
    }
    return id;
}
inline bool ecmp( Edge a, Edge b ) {
    return a.l < b.l;
}
int F[10005];
int findroot( int x ) {
    return F[x] == x ? x : F[x] = findroot( F[x] );
}
int main() {
    int n, K;
    while( ~scanf( "%d%d", &n, &K ) ) {
        for( int i = 0; i < n; ++i ) {
            scanf( "%d%d", &po[i].x, &po[i].y );
            po[i].id = i;
        }
        for( int dir = 0; dir < 4; ++dir ) {
            if( dir == 1 || dir == 3 ) {
                for( int i = 0; i < n; ++i ) swap( po[i].x,
                     po[i].y );
            } else if( dir == 2 ) {
                for( int i = 0; i < n; ++i ) po[i].x *= -1;
            }
            cc = 0;
            for( int i = 0; i < n; ++i ) data[cc++] =
                 po[i].y - po[i].x;
            sort( data, data + cc );
            cc = unique( data, data + cc ) - data;
            sort( po, po + n, cmp );
            memset( bitv, 0x3f, sizeof( bitv ) );
            for( int i = 0; i < n; ++i ) {
                int v = Find( po[i].y - po[i].x );
                int id = read( v );
                if( id != -1 ) addedge( po[i].id, po[id].id,
                     Dis( po[i], po[id] ) );
                add( v, po[i].x + po[i].y, i );
            }
        }
        sort( ed, ed + ecnt, ecmp );
        for( int i = 0; i < n; ++i ) F[i] = i;
        int cnt = 0;
        for( int i = 0; i < ecnt; ++i ) {
            int fu = findroot( ed[i].u ), fv = findroot(
                 ed[i].v );
            if( fu == fv ) continue;
            ++cnt;
            if( cnt == n - K ) {
                printf( "%d\n", ed[i].l );
                break;
            }
            F[fu] = fv;
        }
    }
    return 0;
}
```

2.9 Scanline Circle

```
const int maxn=210000;
int n,fa[maxn],deep[maxn];
int nowpos;
struct cir{
    int x,y,r,up,p;
    cir(){};
    cir(int x,int y,int r,int up=0,int
         p=0):x(x),y(y),r(r),up(up),p(p){};
    bool operator <(const cir a)const{
        if(x==a.x&&y==a.y&&r==a.r)return up<a.up;
        double cy=y + ((up)?1:-1) * sqrt(1ll*r*r - 1ll *
             (x-nowpos) * (x-nowpos));
        double ay=a.y + ((a.up)?1:-1) * sqrt(1ll*a.r*a.r -
             1ll * (a.x-nowpos) * (a.x-nowpos));
        return cy<ay;
    }
}c[maxn];
vector<pair<int,int>>ls;
set<cir>q;
void init(){
    ls.clear();
    scanf("%d",&n);
    for(int x=1;x<=n;x++){
        scanf("%d%d%d",&c[x].x,&c[x].y,&c[x].r);
        c[x].up=1;
        c[x].p=x;
        ls.push_back(make_pair(c[x].x-c[x].r,x));
        ls.push_back(make_pair(c[x].x+c[x].r,x));
    }
    sort(ls.begin(),ls.end());
    ls.erase(unique(ls.begin(),ls.end()),ls.end());
}
void build(){
    memset(fa,0,sizeof(fa));
    memset(deep,0,sizeof(deep));
    q.clear();
    nowpos=-100000;
    cir zs(0,0,100000,1,0);
    q.insert(zs);
    zs.up=0;
    q.insert(zs);
    for(auto && tt:ls){
        nowpos=tt.first;
        int pos=tt.second;
        if(nowpos==c[pos].x-c[pos].r){ //add
            auto ne=q.lower_bound(c[pos]);
            auto pr=prev(ne);
            if(ne->p==pr->p)
                fa[pos]=ne->p,deep[pos]=deep[ne->p]+1;
            else if(deep[ne->p]>deep[pr->p])
                fa[pos]=fa[ne->p],deep[pos]=deep[ne->p];
            else
                fa[pos]=fa[pr->p],deep[pos]=deep[pr->p];
            c[pos].up=0;
            q.insert(c[pos]);
            c[pos].up=1;
            q.insert(c[pos]);
        }
        else{         //del
            c[pos].up=1;
            auto ne=q.upper_bound(c[pos]);
            auto t1=prev(ne);
            assert(t1->p==pos);
            q.erase(t1);
            ne=q.upper_bound(c[pos]);
            auto t2=prev(ne);
            assert(t2->p==pos);
            q.erase(t2);
        }
    }
}
```

2.10 PBDS

```
#include <ext/pb_ds/priority_queue.hpp>
__gnu_pbds::priority_queue < int > Q;
__gnu_pbds::priority_queue < int , greater < int > , pairing_heap_tag > Q;
__gnu_pbds::priority_queue < int , greater < int > , pairing_heap_tag > :: point_iterator id[ maxn ];
id[x] = Q.push( 5 ) ;
Q.modify( id[x] , 6) ;

#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
tree < int , int , less < int > , rb_tree_tag , tree_order_statistics_node_update > rbt;
tree<int,null_type,less<int>,rb_tree_tag, tree_order_statistics_node_update> :: iterator it;
find_by_order(size_type order)
order_of_key(int val)

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/hash_policy.hpp>
__gnu_pbds::cc_hash_table < key , value > hs;
struct HASH{
  size_t operator()(const pair<int,int>&x)const{
    return ((long long)x.first)^(((long long)x.second)<<32);
  }
};
unordered_map<pair<int,int>,int,HASH>m;
//-------------
template < class Node_CItr , class Node_Itr , class Cmp_Fn , class _Alloc>
struct MyUpdate{
    virtual Node_CItr node_begin() const = 0;
    virtual Node_CItr node_end() const = 0;
    typedef int metadata_type;
    inline void operator()(Node_Itr it , Node_CItr end_it){
        Node_Itr l = it.get_l_child(), r = it.get_r_child();
        int res = 0;
        if(l != end_it) res = max( res , l.get_metadata());
        if(r != end_it) res = max( res , r.get_metadata());
        const_cast <metadata_type &>(it.get_metadata()) = max( res , (*it)->second );
    }
    inline int PrefixMax( int x ){
        int ret = 0;
        Node_CItr it = node_begin();
        while( it != node_end() ){
            Node_CItr l = it.get_l_child() , r = it.get_r_child();
            if(Cmp_Fn()(x,(*it)->first)) it=l;
            else{
                ret = max( ret , (*it)->second );
                if( l != node_end() ) ret = max( ret , l.get_metadata() );
                it = r;
            }
        }
        return ret;
    }
};
tree < int , int , less < int > , rb_tree_tag , MyUpdate > rbt;
```

2.11 kdTree

```
namespace Kdtree{
const static int Maxn = 1e5 + 50;
static int Dim = 5;
struct Point{
    int x[5];
}p[Maxn << 2];
int isok[Maxn << 2];
priority_queue < pair < long long , int > > pq;
long long Cal( const Point & x , const Point & y ){
    long long ret = 0;
    for(int i = 0 ; i < Dim ; ++ i)
        ret += 1LL * ( x.x[i] - y.x[i] ) * ( x.x[i] - y.x[i] );
    return ret;
}
void Build( int l , int r , Point s[] , int o = 1 , int k = 0 ){
    if( l > r ) return;
    int mid = l + r >> 1;
    isok[o] = 1 , isok[o << 1] = isok[o << 1 | 1] = 0;
    nth_element( s + l , s + mid , s + r + 1 , [ & ]( const Point & x , const Point & y ){
        return x.x[k] < y.x[k];
    } );
    p[o] = s[mid];
    int nexv = ( k + 1 == Dim ) ? 0 : k + 1;
    Build( l , mid - 1 , s , o << 1 , nexv );
    Build( mid + 1 , r , s , o << 1 | 1 , nexv );
}
void query( Point x , int m , int o = 1 , int k = 0 ){
    if( !isok[o] ) return;
    int lft = o << 1 , rht = o << 1 | 1 , nexv = ( k + 1 == Dim ) ? 0 : k + 1;
    if( x.x[k] > p[o].x[k] ) swap( lft , rht );
    query( x , m , lft , nexv );
    while( pq.size() >= m && Cal( p[o] , x ) < pq.top().first )
        pq.pop();
    if( pq.size() < m )
        pq.push( make_pair( Cal( p[o] , x ) , o ) );
    if( 1LL * ( x.x[k] - p[o].x[k] ) * ( x.x[k] - p[o].x[k] ) < pq.top().first )
        query( x , m , rht , nexv );
}
};
```

2.12 Virtual Tree

```
int n;
vector< int > adj[MAXN];
vector< int > L[MAXN];
namespace Least_Common_Ancestors{
    const int LOG = 20;
    int dep[MAXN], fa[MAXN][20];
    void DFS(int u, int f, int l){
        dep[u] = dep[f] + 1;
        fa[u][0] = f;
        L[l].push_back(u);
        for(auto &v : adj[u]){
            if(v == f) continue;
            DFS(v, u, l + 1);
        }
    }
    void Build(int root = 1, int root_dep = -1){
        dep[0] = root_dep; DFS(root, 0, 0);
        for(int j = 1; j < LOG; ++j)
            for(int i = 1; i <= n; ++i)
                fa[i][j] = fa[fa[i][j - 1]][j - 1];
    }
    int LCA(int x, int y){
        if(dep[x] < dep[y]) swap(x, y);
        for(int i = 19; ~i; --i)
            if(dep[x] - (1 << i) >= dep[y])
                x = fa[x][i];
        if(x == y) return x;
        for(int i = 19; ~i; --i)
            if(fa[x][i] != fa[y][i])
                x = fa[x][i], y = fa[y][i];
        return fa[x][0];
    }
}
void Init(){
    int i, fa;
    scanf("%d", &n); ++n;
    for(i = 2; i <= n; ++i){
        scanf("%d", &fa); ++fa;
        adj[fa].push_back(i);
    }
    Least_Common_Ancestors::Build();
}
int dp[MAXN][2], f[MAXN];
int stk[MAXN];
vector < int > tree[MAXN];
```

```
vector< int > s;
void Addedge(int u, int v){
    tree[u].push_back(v);
    tree[v].push_back(u);
    s.push_back(u); s.push_back(v);
}
void build(vector< int > p, int k){
    int sz = 0; stk[sz++] = 0;
    for(int i = 0; i < k; ++i) {
        int u = p[i], lca = Least_Common_Ancestors::LCA(u,
             stk[sz - 1]);
        if(lca == stk[sz - 1]) stk[sz++] = u;
        else {
            while (sz - 2 >= 0 &&
                 Least_Common_Ancestors::dep[stk[sz - 2]] >=
                 Least_Common_Ancestors::dep[lca]) {
                Addedge(stk[sz - 2], stk[sz - 1]);
                sz--;
            }
            if (stk[sz - 1] != lca) {
                Addedge(lca, stk[--sz]);
                stk[sz++] = lca;
            }
            stk[sz++] = u;
        }
    }
    for(int i = 0; i < sz - 1; ++i) Addedge(stk[i], stk[i +
         1]);
}
int POW(int a,int b){
    int res;
    for(res = 1; b; (b & 1) && (res = 1LL * res * a % mo),
         a = 1LL * a * a % mo, b >>= 1);
    return res;
}
void DFS(int u, int fa){
    int flag = false;
    dp[u][0] = dp[u][1] = 0;
    int res = 1, res2 = 1;
    for(auto &v: tree[u]){
        if(v == fa) continue;
        DFS(v, u); flag = true;
        res = 1LL * res * (dp[v][0] + dp[v][1]) % mo;
        res2 = 1LL * res2 * dp[v][0] % mo;
    }
    if(!flag) dp[u][0] = dp[u][1] = 1;
    else{
        for(auto &v: tree[u]){
            if(v == fa) continue;
            dp[u][1] = (dp[u][1] + 1LL * res2 *
                 POW(dp[v][0], mo - 2) % mo * dp[v][1]) % mo;
        }
        dp[u][0] = (0LL + res + mo - dp[u][1]) % mo;
    }
}
void Work(){
    int ans = 0;
    for(int i = 0; i <= n; ++i){
        if((int)L[i].size() == 0) continue;
        build(L[i], L[i].size());
        DFS(0, -1);
        ans = (ans + 1LL * dp[0][1] * POW(2, n -
             (int)L[i].size())) % mo;
        for(auto &it: s){
            tree[it].clear();
        }
        s.clear();
    }
    printf("%d\n", ans);
}
```

3 Graph

3.1 Directed MST

```
struct Directed_Mst{
typedef int Directed_MST_Type;
const static Directed_MST_Type inf=(1)<<30;
const static int M = 105;
struct Node{
    int u , v;
    Directed_MST_Type cost;
}E[M*M+5];
int pre[M],ID[M],vis[M],n,m;
Directed_MST_Type In[M];
void init(int n){
    this-> n = n;
    m = 0;
}
void link(int u,int v,Directed_MST_Type c){
    if(u != v){
        E[m].u = u , E[m].v = v , E[m++].cost = c;
    }
}
Directed_MST_Type Directed_MST(int root) {
    int NV = n , NE = m;
    Directed_MST_Type ret = 0;
    while(true) {
        for(int i=0;i<NV;i++) In[i] = inf;
        for(int i=0;i<NE;i++){
            int u = E[i].u;
            int v = E[i].v;
            if(E[i].cost < In[v] && u != v) {
                pre[v] = u;
                In[v] = E[i].cost;
            }
        }
        for(int i=0;i<NV;i++) {
            if(i == root) continue;
            if(In[i] == inf) return -1;
        }
        int cntnode = 0;
        memset(ID,-1,sizeof(ID));
        memset(vis,-1,sizeof(vis));
        In[root] = 0;
        for(int i=0;i<NV;i++) {
            ret += In[i];
            int v = i;
            while(vis[v] != i && ID[v] == -1 && v != root) {
                vis[v] = i;
                v = pre[v];
            }
            if(v != root && ID[v] == -1) {
                for(int u = pre[v] ; u != v ; u = pre[u]) {
                    ID[u] = cntnode;
                }
                ID[v] = cntnode ++;
            }
        }
        if(cntnode == 0)    break;
        for(int i=0;i<NV;i++) if(ID[i] == -1) {
            ID[i] = cntnode ++;
        }
        for(int i=0;i<NE;i++) {
            int v = E[i].v;
            E[i].u = ID[E[i].u];
            E[i].v = ID[E[i].v];
            if(E[i].u != E[i].v) {
                E[i].cost -= In[v];
            }
        }
        NV = cntnode;
        root = ID[root];
    }
    return ret;
}
}solver;
```

3.2 Global Minimum Cut

```
const int maxn = 510;
int G[maxn][maxn];
int n, m;
void contract(int x, int y) {
    for(int i = 0; i < n; ++i) if(i != x) G[x][i] +=
         G[y][i], G[i][x] += G[i][y];
```

```
    for(int i = y + 1; i < n; ++i) for(int j = 0; j < n; ++j) {
      G[i - 1][j] = G[i][j];
      G[j][i - 1] = G[j][i];
    }
  n--;
}
int w[maxn], c[maxn];
int sx, tx;
int mincut() {
  int t, k;
  memset(c, 0, sizeof(c));
  c[0] = 1;
  for(int i = 0; i < n; ++i) w[i] = G[0][i];
  for(int i = 1; i + 1 < n; ++i) {
    t = k = -1;
    for(int j = 0; j < n; ++j) if(c[j] == 0 && w[j] > k) k = w[t = j];
    c[sx = t] = 1;
    for(int j = 0; j < n; ++j) w[j] += G[t][j];
  }
  for(int i = 0; i < n; ++i) if(c[i] == 0) return w[tx = i];
}
int main() {
  while(~scanf("%d%d", &n, &m)) {
    memset(G, 0, sizeof(G));
    while(m--) {
      int u, v, c;
      scanf("%d%d%d", &u, &v, &c);
      G[u][v] += c;
      G[v][u] += c;
    }
    int mint = INF;
    while(n > 1) {
      int t = mincut();
      mint = min(mint, t);
      contract(sx, tx);
    }
    printf("%d\n", mint);
  }
  return 0;
}
```

3.3 Dominator Tree

```
/**
 * 应用:
 * 1. 求有向图的割顶: dominator tree上的非叶节点
 * 2. 有向图的必经边: 每条边上加一个点, 转化成必经点问题
 * 3. 求起点S到终点T的所有路径中最接近源的必经点: 求出必经点, 取最近的
 * 4.      求多少个(x,y)满足存在1->x的路径和1->y的路径只有1这个公共点:
 *    求出1为根的dominator tree, 算出不合法的, 总的减去即可.
 *    考虑1的每个儿子v, 同一颗子树的节点对都是非法的.
 *
 * succ是原图, pred是边反向后的图, dom是Dominator Tree
 * dom记录的是dfs序构成的树, G中节点u在dom树上的标号是dfn[u]
 * 相反dom中节点u在原图G中的标号是pt[u]
 * 调用build得到以s为根的Dominator Tree
 **/
#include <vector>
namespace DominatorTree {
  const static int N = 100000 + 10;
  typedef std::vector<int> VI;
  int dfn[N], pre[N], pt[N], sz;
  int semi[N], dsu[N], idom[N], best[N];
  int get(int x) {
    if (x == dsu[x]) return x;
    int y = get(dsu[x]);
    if (semi[best[x]] > semi[best[dsu[x]]]) best[x] = best[dsu[x]];
    return dsu[x] = y;
  }
  void dfs(int u, const VI succ[]) {
    dfn[u] = sz; pt[sz ++] = u;
    for (auto &v: succ[u]) if (!~dfn[v]) {
      dfs(v, succ); pre[dfn[v]] = dfn[u];
    }
  }
  void tarjan(const VI pred[], VI dom[]) {
    for (int j = sz - 1, u; u = pt[j], j > 0; -- j) {
      for (auto &tv: pred[u]) if (~dfn[tv]) {
        int v = dfn[tv]; get(v);
        if (semi[best[v]] < semi[j]) semi[j] = semi[best[v]];
      }
      dom[semi[j]].push_back(j);
      int x = dsu[j] = pre[j];
      for (auto &z: dom[x]) {
        get(z);
        if (semi[best[z]] < x) idom[z] = best[z];
        else idom[z] = x;
      }
      dom[x].clear();
    }
    for (int i = 1; i < sz; ++ i) {
      if (semi[i] != idom[i]) idom[i] = idom[idom[i]];
      dom[idom[i]].push_back(i);
    }
  }
  void build(int n, int s, const VI succ[], const VI pred[], VI dom[]) {
    for (int i = 0; i < n; ++ i) {
      dfn[i] = -1; dom[i].clear();
      dsu[i] = best[i] = semi[i] = i;
    }
    sz = 0; dfs(s, succ); tarjan(pred, dom);
  }
}
```

3.4 KM

```
/*******************************************************
Bipartite Graph Maximum Weighted Matching
(kuhn munkras algorithm O(m*m*n))
adjacent matrix: mat
notice: m <= n
init: for(i=0;i<MAXN;i++)
          for(j=0;j<MAXN;j++) mat[i][j]=-inf;
for existing edges: mat[i][j]=val;
*******************************************************/
#define MAXN 310
#define inf 1000000000
#define _clr(x) memset(x,-1,sizeof(int)*MAXN)
int KM(int m, int n, int mat[][MAXN], int *match1, int *match2) {
  int s[MAXN], t[MAXN], l1[MAXN], l2[MAXN];
  int p, q, i, j, k, ret = 0;
  for(i = 0; i < m; i++) {
    l1[i] = -inf;
    for(j = 0; j < n; j++)
      l1[i] = mat[i][j] > l1[i] ? mat[i][j] : l1[i];
    if(l1[i] == -inf) return -1;
  }
  for(i = 0; i < n; i++) l2[i] = 0;
  _clr(match1); _clr(match2);
  for(i = 0; i < m; i++) {
    _clr(t); p = 0; q = 0;
    for(s[0] = i; p <= q && match1[i] < 0; p++)
      for(k = s[p], j = 0; j < n && match1[i] < 0; j++)
        if(l1[k] + l2[j] == mat[k][j] && t[j] < 0) {
          s[++q] = match2[j]; t[j] = k;
          if(s[q] < 0) {
            for(p = j; p >= 0; j = p) {
              match2[j] = k = t[j];
              p = match1[k];
              match1[k] = j;
            }
          }
        }
    if(match1[i] < 0) {
      i--; p = inf;
      for(k = 0; k <= q; k++)
        for(j = 0; j < n; j++)
          if(t[j] < 0 && l1[s[k]] + l2[j] - mat[s[k]][j] < p)
```

```
                    p = l1[s[k]] + l2[j] - mat[s[k]][j];
            for(j = 0; j < n; j++)
                l2[j] += t[j] < 0 ? 0 : p;
            for(k = 0; k <= q; k++)
                l1[s[k]] -= p;
        }
    }
    for(i = 0; i < m; i++)
        ret += mat[i][match1[i]];
    return ret;
}
```

3.5 ISAP

```
namespace ISAP{
const int Maxn = 1e5 + 10, Maxm = 1e6 + 10, Inf =
      0x3f3f3f3f;
int s, t, cnt, Maxd, d[Maxn], vd[Maxn], adj[Maxn];
struct node{
    int v, f, next;
}edge[Maxm * 2];
void Init(int S, int T){
    cnt = 1; s = S; t = T; Maxd = t + 5;
    memset(adj, 0, sizeof(adj));
}
void Addedge(int u, int v, int f){
    ++cnt;
    edge[cnt].v = v; edge[cnt].f = f;
    edge[cnt].next = adj[u]; adj[u] = cnt;
    ++cnt;
    edge[cnt].v = u; edge[cnt].f = 0;
    edge[cnt].next = adj[v]; adj[v] = cnt;
}
int Aug(int u, int lim){
    if(u == t) return lim;
    int sum = 0, delta, Mind = Maxd - 1;
    for(int p = adj[u]; p; p = edge[p].next){
        if(!edge[p].f) continue;
        if(d[u] == d[edge[p].v] + 1){
            delta = Aug(edge[p].v, min((lim - sum),
                  edge[p].f));
            edge[p].f -= delta; edge[p ^ 1].f += delta;
            sum += delta;
            if(d[s] >= Maxd) return sum;
            if(sum == lim) break;
        }
        Mind = min(Mind, d[edge[p].v]);
    }
    if(!sum){
        if(!(--vd[d[u]])) d[s] = Maxd;
        ++vd[d[u] = Mind + 1];
    }
    return sum;
}
int Maxflow(){
    memset(d, 0, sizeof(d));
    memset(vd, 0, sizeof(vd));
    int flow = 0; vd[0] = Inf;
    while(d[s] < Maxd) flow += Aug(s, Inf);
    return flow;
}
}
```

3.6 Dinic

```
namespace NetFlow{
const int MAXN=100000,MAXM=100000,inf=1e9;
struct Edge{
    int v,c,f,nx;
    Edge() {}
    Edge(int v,int c,int f,int nx):v(v),c(c),f(f),nx(nx) {}
} E[MAXM];
int G[MAXN],cur[MAXN],pre[MAXN],dis[MAXN],gap[MAXN],N,sz;
void init(int _n){
    N=_n,sz=0; memset(G,-1,sizeof(G[0])*N);
}
void link(int u,int v,int c){
    E[sz]=Edge(v,c,0,G[u]); G[u]=sz++;
    E[sz]=Edge(u,0,0,G[v]); G[v]=sz++;
}
bool bfs(int S,int T){
    static int Q[MAXN]; memset(dis,-1,sizeof(dis[0])*N);
    dis[S]=0; Q[0]=S;
    for (int h=0,t=1,u,v,it;h<t;++h){
        for (u=Q[h],it=G[u];~it;it=E[it].nx){
            if (dis[v=E[it].v]==-1&&E[it].c>E[it].f){
                dis[v]=dis[u]+1; Q[t++]=v;
            }
        }
    }
    return dis[T]!=-1;
}
int dfs(int u,int T,int low){
    if (u==T) return low;
    int ret=0,tmp,v;
    for (int &it=cur[u];~it&&ret<low;it=E[it].nx){
        if (dis[v=E[it].v]==dis[u]+1&&E[it].c>E[it].f){
            if (tmp=dfs(v,T,min(low-ret,E[it].c-E[it].f))){
                ret+=tmp; E[it].f+=tmp; E[it^1].f-=tmp;
            }
        }
    }
    if (!ret) dis[u]=-1; return ret;
}
int dinic(int S,int T){
    int maxflow=0,tmp;
    while (bfs(S,T)){
        memcpy(cur,G,sizeof(G[0])*N);
        while (tmp=dfs(S,T,inf)) maxflow+=tmp;
    }
    return maxflow;
}
}
```

3.7 MCMF

```
struct Mcmf{
const static int MAXN = 10000;
const static int MAXM = 100000;
int INF = 0x3f3f3f3f;
struct Edge{
    int to, next, cap, flow, cost;
    int x, y;
} edge[MAXM],HH[MAXN],MM[MAXN];
int head[MAXN],tol;
int pre[MAXN],dis[MAXN];
bool vis[MAXN];
int N, M;
void init(){
    N = MAXN; tol = 0;
    memset(head, -1, sizeof(head));
}
void addedge(int u, int v, int cap, int cost){
    edge[tol]. to = v;
    edge[tol]. cap = cap;
    edge[tol]. cost = cost;
    edge[tol]. flow = 0;
    edge[tol]. next = head[u];
    head[u] = tol++;
    edge[tol]. to = u;
    edge[tol]. cap = 0;
    edge[tol]. cost = -cost;
    edge[tol]. flow = 0;
    edge[tol]. next = head[v];
    head[v] = tol++;
}
bool spfa(int s, int t){
    queue<int>q;
    for(int i = 0; i < N; i++){
        dis[i] = INF;
        vis[i] = false;
        pre[i] = -1;
    }
    dis[s] = 0;
    vis[s] = true;
    q.push(s);
    while(!q.empty()){
```

```
        int u = q.front();
        q.pop();
        vis[u] = false;
        for(int i = head[u]; i != -1; i = edge[i]. next){
            int v = edge[i]. to;
            if(edge[i]. cap > edge[i]. flow &&
                    dis[v] > dis[u] + edge[i]. cost ){
                dis[v] = dis[u] + edge[i]. cost;
                pre[v] = i;
                if(!vis[v]){
                    vis[v] = true;
                    q.push(v);
                }
            }
        }
    }
    if(pre[t] == -1) return false;
    else return true;
}
int minCostMaxflow(int s, int t, int &cost){
    int flow = 0;
    cost = 0;
    while(spfa(s,t)){
        int Min = INF;
        for(int i = pre[t]; i != -1; i = pre[edge[i^1]. to]){
            if(Min > edge[i]. cap - edge[i]. flow)
                Min = edge[i]. cap - edge[i]. flow;
        }
        for(int i = pre[t]; i != -1; i = pre[edge[i^1]. to]){
            edge[i]. flow += Min;
            edge[i^1]. flow -= Min;
            cost += edge[i]. cost * Min;
        }
        flow += Min;
    }
    return flow;
}
};
```

3.8 ZKW MCMF

```
namespace ZKE_MCMF{
#define V(m) (e[(m)].v)
#define F(m) (e[(m )].f)
#define C(m) (e[(m)].c)
#define NX(m) (e[(m)].next)
const int Maxn = 510, Inf = 0x3f3f3f3f;
int s, t, ans, ecnt=1, adj[Maxn],dis[Maxn];
bool vis[Maxn];
struct node{
    int v,f,c,next;
}e[Maxn*Maxn];
void Addedge(int u,int v,int f,int c){
    ++ecnt;
    V(ecnt)=v; F(ecnt)=f; C(ecnt)=c;
    NX(ecnt)=adj[u]; adj[u]=ecnt;
    ++ecnt;
    V(ecnt)=u; F(ecnt)=0; C(ecnt)=-c;
    NX(ecnt)=adj[v]; adj[v]=ecnt;
}
int Aug(int u, int lim){
    if(u == t){
        ans+=lim*dis[1];
        return lim;
    }
    vis[u]=true;
    int p,v,f,c,delta,tot,sum=0;
    for(p=adj[u];p;p=NX(p)){
        v=V(p); f=F(p); c=C(p);
        if(vis[v] || !f || dis[v]+c!=dis[u]) continue;
        tot=min(f,(lim-sum));
        delta=Aug(v,tot);
        F(p)-=delta; F(p^1)+=delta;
        sum+=delta;
        if(sum==lim) break;
    }
    return sum;
}
bool Update(){
    int i,p,v,c,f,Min=Inf;
    for(i=s;i<=t;++i){
        if(!vis[i]) continue;
        for(p=adj[i];p;p=NX(p)){
            v=V(p); f=F(p); c=C(p);
            if(!f || vis[v]) continue;
            Min=min(Min,(dis[v]+c-dis[i]));
        }
    }
    if(Min==Inf) return false;
    for(i=s;i<=t;++i) if(vis[i]) dis[i]+=Min;
    return true;
}
void ZKW(){
    do{ do{ memset(vis,0,sizeof(vis));
            }while(Aug(s,Inf));
    }while(Update());
    printf("%d\n",ans);
}
};
```

3.9 TarjanSCC

```
vector<int> G[maxn]; // 邻接表
int dfn[maxn]; // dfs时间戳
int low[maxn]; // u以及u的后代所能追溯到的最早的祖先
int sccno[maxn]; // sccno[u] 为u所属SCC的编号，从1开始
int dfs_clock; // dfs时间戳变量
int scc_cnt;   // SCC数量
stack<int> S;  // DFS储存访问的结点
void dfs(int u) {
    dfn[u] = low[u] = ++dfs_clock;
    S.push(u);
    for (int i = 0; i < G[u].size(); i++) {
        int v = G[u][i];
        if (!dfn[v]) { // 这条边是树边 没有走过
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
        else if (!sccno[v]) { // 这条边是反向边 走过
              但sccno[v]==0，为祖先
            low[u] = min(low[u], dfn[v]);
        }
    }

    if (low[u] == dfn[u]) { // 判断是否是SCC中的最早访问结点
        scc_cnt++;
        for (;;) {
            int x = S.top(); S.pop();
            sccno[x] = scc_cnt;
            if (x == u) break;
        }
    }
}
void find_scc(int n) {
    dfs_clock = scc_cnt = 0;
    memset(sccno, 0, sizeof(sccno));
    memset(dfn, 0, sizeof(dfn));
    for (int i = 0; i < n; i++) {
        if (!dfn[i]) dfs(i);
    }
}
```

3.10 TarjanBCC

```
int tarjan_dfs(int u,int fa){
    dfn[u] = low[u] = ++T;
    for(int i = head[u] ; ~i ; i = e[i].nxt){
        int v = e[i].v;
        if(v == fa) continue;
        if(!dfn[v]){
            s.push(make_pair(u,v));
            int lowv = tarjan_dfs(v,u);
            low[u] = min(low[u],lowv);
            if(lowv >= dfn[u]){
                bcc_cnt++;bcc[bcc_cnt].clear();
```

```
            while(1){
                dl ss = s.top();s.pop();
                int x = ss.first , y = ss.second;
                if(belong[x] != bcc_cnt) {
                    bcc[bcc_cnt].push_back(x);
                    belong[x] = bcc_cnt;
                }
                if(belong[y] != bcc_cnt) {
                    bcc[bcc_cnt].push_back(y);
                    belong[y] = bcc_cnt;
                }
                if(x==u&&y==v) break;
            }
        }
    }
    else if(dfn[v] < dfn[u]){
        s.push(make_pair(u,v));
        low[u] = min(low[u] , dfn[v]);
    }
  }
  return low[u];
}
```

3.11 Blossom Tree

```
const int N = 250;
int belong[N];
int findb(int x) {
    return belong[x] == x ? x : belong[x] =
         findb(belong[x]);
}
void unit(int a, int b) {
    a = findb(a);
    b = findb(b);
    if (a != b) belong[a] = b;
}
int n, match[N];
vector<int> e[N];
int Q[N], rear;
int next[N], mark[N], vis[N];
int LCA(int x, int y) {
    static int t = 0;
    t++;
    while (true) {
        if (x != -1) {
            x = findb(x);
            if (vis[x] == t) return x;
            vis[x] = t;
            if (match[x] != -1) x = next[match[x]];
            else x = -1;
        }
        swap(x, y);
    }
}
void group(int a, int p) {
    while (a != p) {
        int b = match[a], c = next[b];
        if (findb(c) != p) next[c] = b;
        if (mark[b] == 2) mark[Q[rear++] = b] = 1;
        if (mark[c] == 2) mark[Q[rear++] = c] = 1;
        unit(a, b);
        unit(b, c);
        a = c;
    }
}
void aug(int s) {
    for (int i = 0; i < n; i++)
        next[i] = -1, belong[i] = i, mark[i] = 0, vis[i] =
             -1;
    mark[s] = 1;
    Q[0] = s;
    rear = 1;
    for (int front = 0; match[s] == -1 && front < rear;
         front++) {
        int x = Q[front];
        for (int i = 0; i < (int)e[x].size(); i++) {
            int y = e[x][i];
            if (match[x] == y) continue;
            if (findb(x) == findb(y)) continue;
            if (mark[y] == 2) continue;
            if (mark[y] == 1) {
                int r = LCA(x, y);
                if (findb(x) != r) next[x] = y;
                if (findb(y) != r) next[y] = x;

                group(x, r);
                group(y, r);
            } else if (match[y] == -1) {
                next[y] = x;
                for (int u = y; u != -1; ) {
                    int v = next[u];
                    int mv = match[v];
                    match[v] = u, match[u] = v;
                    u = mv;
                }
                break;
            } else {
                next[y] = x;
                mark[Q[rear++] = match[y]] = 1;
                mark[y] = 2;
            }
        }
    }
}
bool g[N][N];
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) g[i][j] = false;

    int x, y;
    while (scanf("%d%d", &x, &y) != EOF) {
        x--, y--;
        if (x != y && !g[x][y])
            e[x].push_back(y), e[y].push_back(x);
        g[x][y] = g[y][x] = true;
    }

    for (int i = 0; i < n; i++) match[i] = -1;
    for (int i = 0; i < n; i++) if (match[i] == -1) aug(i);

    int tot = 0;
    for (int i = 0; i < n; i++) if (match[i] != -1) tot++;
    printf("%d\n", tot);
    for (int i = 0; i < n; i++) if (match[i] > i)
            printf("%d %d\n", i + 1, match[i] + 1);
    return 0;
}
```

4 Math

4.1 EX GCD

```
//ax+by=gcd(a,b):
LL ex(LL a,LL b,LL &x,LL &y){
    if(a==0&&b==0)return -1;
    if(b==0){x=1;y=0;return a;}
    LL d=ex(b,a%b,y,x);
    y-=a/b*x;
    return d;
}
//-1 no solution
//x=x0+(b/g)t, y=y0-(a/g)t
//ax+by=c:******
//g=gcd(a,b)
//ax+by=g*(c/g) //gcd(a,b) |c not no solution
//ax0+by0=g
//x=(c/g)x0+(b/g)t, y=(c/g)y0-(a/g)t
//ax=b%n:******
//ax+n(-y)=b,g=gcd(a,n) //gcd(a,n)|b not no
//x=(b/g)x0+(n/g)t %n   t=[0,g)
```

4.2 FFT

```
using namespace std;
```

```
const int mod = 1e9 + 7;
const int max0 = 1 << 17;
struct comp {
    double x, y;
    comp() : x(0), y(0) {}
    comp(const double &_x, const double &_y) : x(_x), y(_y) {}
};
inline comp operator+(const comp &a, const comp &b) {
    return comp(a.x + b.x, a.y + b.y);
}
inline comp operator-(const comp &a, const comp &b) {
    return comp(a.x - b.x, a.y - b.y);
}
inline comp operator*(const comp &a, const comp &b) {
    return comp(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x);
}
inline comp conj(const comp &a) {
    return comp(a.x, -a.y);
}
const double PI = acos(-1);
int N, L;
comp w[max0 + 5];
int bitrev[max0 + 5];
void fft(comp *a, const int &n) {
    for (int i = 0; i < n; ++i)
        if (i < bitrev[i])
            swap(a[i], a[bitrev[i]]);
    for (int i = 2, lyc = n >> 1; i <= n; i <<= 1, lyc >>= 1)
        for (int j = 0; j < n; j += i) {
            comp *l = a + j, *r = a + j + (i >> 1), *p = w;
            for (int k = 0; k < i >> 1; ++k) {
                comp tmp = *r * *p;
                *r = *l - tmp, *l = *l + tmp;
                ++l, ++r, p += lyc;
            }
        }
}
inline void fft_prepare() {
    for (int i = 0; i < N; ++i)
        bitrev[i] = bitrev[i >> 1] >> 1 | ((i & 1) << (L - 1));
    for (int i = 0; i < N; ++i)
        w[i] = comp(cos(2 * PI * i / N), sin(2 * PI * i / N));
}
inline vector<int> conv(const vector<int> &x, const vector<int> &y) {
    static comp a[max0 + 5], b[max0 + 5];
    static comp dfta[max0 + 5], dftb[max0 + 5], dftc[max0 + 5], dftd[max0 + 5];
    L = 0;
    while ((1 << L) < x.size() + y.size() - 1)
        ++L;
    N = 1 << L;
    fft_prepare();
    for (int i = 0; i < N; ++i)
        a[i] = b[i] = comp(0, 0);
    for (int i = 0; i < x.size(); ++i)
        a[i] = comp(x[i] & 32767, x[i] >> 15);
    for (int i = 0; i < y.size(); ++i)
        b[i] = comp(y[i] & 32767, y[i] >> 15);
    fft(a, N), fft(b, N);
    for (int i = 0; i < N; ++i) {
        int j = (N - i) & (N - 1);
        static comp da, db, dc, dd;
        da = (a[i] + conj(a[j])) * comp(0.5, 0);
        db = (a[i] - conj(a[j])) * comp(0, -0.5);
        dc = (b[i] + conj(b[j])) * comp(0.5, 0);
        dd = (b[i] - conj(b[j])) * comp(0, -0.5);
        dfta[j] = da * dc;
        dftb[j] = da * dd;
        dftc[j] = db * dc;
        dftd[j] = db * dd;
    }
    for (int i = 0; i < N; ++i)
        a[i] = dfta[i] + dftb[i] * comp(0, 1);
    for (int i = 0; i < N; ++i)
        b[i] = dftc[i] + dftd[i] * comp(0, 1);
    fft(a, N), fft(b, N);
    vector<int> z(x.size() + y.size() - 1);
    for (int i = 0; i < x.size() + y.size() - 1; ++i) {
        int da = (long long)(a[i].x / N + 0.5) % mod;
        int db = (long long)(a[i].y / N + 0.5) % mod;
        int dc = (long long)(b[i].x / N + 0.5) % mod;
        int dd = (long long)(b[i].y / N + 0.5) % mod;
        z[i] = (da + ((long long)(db + dc) << 15) + ((long long)dd << 30)) % mod;
    }
    return z;
}
```

4.3 FFT Normal

```
const double PI = acos(-1.0);
struct Virt {
    double r, i;
    Virt(double r = 0.0, double i = 0.0) {
        this->r = r;
        this->i = i;
    }
    Virt operator + (const Virt &x) {
        return Virt(r + x.r, i + x.i);
    }
    Virt operator - (const Virt &x) {
        return Virt(r - x.r, i - x.i);
    }
    Virt operator * (const Virt &x) {
        return Virt(r * x.r - i * x.i, i * x.r + r * x.i);
    }
};
void Rader(Virt F[], int len) {
    int i, j, k;
    for(i = 1, j = len / 2; i < len - 1; i++) {
        if(i < j) swap(F[i], F[j]);
        k = len / 2;
        while(j >= k) {
            j -= k;
            k >>= 1;
        }
        if(j < k) j += k;
    }
}
void FFT(Virt F[], int len, int on) {
    Rader(F, len);
    for(int h = 2; h <= len; h <<= 1) {
        Virt wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
        for(int j = 0; j < len; j += h) {
            Virt w(1, 0);
            for(int k = j; k < j + h / 2; k++) {
                Virt u = F[k];
                Virt t = w * F[k + h / 2];
                F[k] = u + t;
                F[k + h / 2] = u - t;
                w = w * wn;
            }
        }
    }
    if(on == -1)
        for(int i = 0; i < len; i++)
            F[i].r /= len;
}
```

4.4 NTT

```
const int NUM = 1<<20;
int wn[NUM],bitrev[NUM];
    // p                | deg | g
    // 469762049          26    3
    // 998244353          23    3
    // 1004535809         21    3
    // 1107296257         24    10
    // 10000093151233     26    5
    // 1000000523862017   26    3
    // 1000000000949747713 26  2
LL mu(LL a,LL b,LL P){
```

```
    LL ans=1;
    while(b){
        if(b&1) ans=ans*a%P;
        a=a*a%P;
        b>>=1;
    }
    return ans;
}
void GetWn(int G,int P,int len){
    wn[0] = 1, wn[1] = mu(G, (P - 1) / len, P);
    for(int i = 2; i < len; i++)
        wn[i] = 1LL * wn[i - 1] * wn[1] % P;
    int L=__builtin_ctz(len);
    for(int i=0;i<len;i++) bitrev[i] = bitrev[i >> 1] >> 1
         | ((i & 1) << (L - 1));
}
void NTT(int a[], int len, int on,int P){
    for(int i=0;i<len;i++) if (i < bitrev[i]) swap(a[i],
        a[bitrev[i]]);
    for(int h = 2; h <= len; h <<= 1) {
        int unit = on == -1 ? len - len / h : len / h;
        int hf = h >> 1;
        for(int i = 0; i < len; i += h) {
            int w = 0;
            for(int j = i; j < i + hf; j++) {
                int u = a[j], t = 1LL * wn[w] * a[j + hf] % P;
                if((a[j] = u + t) >= P) a[j] -= P;
                if((a[j + hf] = u - t) < 0) a[j + hf] += P;
                if((w += unit) >= len) w -= len;
            }
        }
    }
    if(on == -1) {
        int inv = mu(len, P - 2, P);
        for(int i = 0; i < len; i++) a[i] = 1LL * a[i] * inv
            % P;
    }
}
/*
    LL N=1ll<<20,g=3,P=469762049ll;
    GetWn(g,P,N);
    NTT(a, N, 1,P);
    NTT(b, N, 1,P);
    for(int i = 0; i < N; i++)
        a[i] = a[i] * b[i] % P;
    NTT(a, N, -1,P);
*/
int callen(int len1,int len2){
    int len=1;
    while(len < (len1<<1) || len < (len2<<1))len<<=1;
    return len;
}
```

4.5 FWT

```
void tfand(int a[], int n) {
    if(n == 1) return;
    int x = n >> 1;
    tfand(a, x); tfand(a + x, x);
    for(int i = 0; i < x; ++ i)
        a[i] += a[i + x];
}
void utfand(long long a[], int n) {
    if(n == 1) return;
    int x = n >> 1;
    for(int i = 0; i < x; ++ i)
        a[i] -= a[i + x];
    utfand(a, x); utfand(a + x, x);
}
void tfor(int a[], int n) {
    if(n == 1) return;
    int x = n >> 1;
    tfor(a, x); tfor(a + x, x);
    for(int i = 0; i < x; ++ i)
        a[i + x] += a[i];
}
void utfor(long long a[], int n) {
    if(n == 1) return;
    int x = n >> 1;
    for(int i = 0; i < x; ++ i)
        a[i + x] -= a[i];
    utfor(a, x); utfor(a + x, x);
}
void utfxor(long long a[], int n) {
    if(n == 1) return;
    int x = n >> 1;
    for(int i = 0; i < x; ++ i) {
        t[i] = (a[i] + a[i + x]) >> 1;
        t[i + x] = (a[i + x] - a[i]) >> 1;
    }
    memcpy(a, t, n * sizeof(long long));
    utfxor(a, x); utfxor(a + x, x);
}
void tfxor(int a[], int n) {
    if(n == 1) return;
    int x = n >> 1;
    tfxor(a, x); tfxor(a + x, x);
    for(int i = 0; i < x; ++ i) {
        tmp[i] = a[i] - a[i + x];
        tmp[i + x] = a[i] + a[i + x];
    }
    memcpy(a, tmp, n * sizeof(int));
}
```

4.6 Miller Rabin-Pollard rho

```
const int S = 20;
long long mult_mod(long long a, long long b, long long c) {
    a %= c;
    b %= c;
    long long ret = 0;
    while(b) {
        if(b & 1) {
            ret += a;
            ret %= c;
        }
        a <<= 1;
        if(a >= c)a %= c;
        b >>= 1;
    }
    return ret;
}
long long pow_mod(long long x, long long n, long long mod)
     { //x^n%c
    if(n == 1)return x % mod;
    x %= mod;
    long long tmp = x;
    long long ret = 1;
    while(n) {
        if(n & 1) ret = mult_mod(ret, tmp, mod);
        tmp = mult_mod(tmp, tmp, mod);
        n >>= 1;
    }
    return ret;
}
bool check(long long a, long long n, long long x, long long
     t) {
    long long ret = pow_mod(a, x, n);
    long long last = ret;
    for(int i = 1; i <= t; i++) {
        ret = mult_mod(ret, ret, n);
        if(ret == 1 && last != 1 && last != n - 1) return
             true;
        last = ret;
    }
    if(ret != 1) return true;
    return false;
}
bool Miller_Rabin(long long n) {
    if(n < 2)return false;
    if(n == 2)return true;
    if((n & 1) == 0) return false; // even number
    long long x = n - 1;
    long long t = 0;
    while((x & 1) == 0) {
        x >>= 1;
        t++;
    }
```

```
    for(int i = 0; i < S; i++) {
        long long a = rand() % (n - 1) + 1;
        if(check(a, n, x, t))
            return false;
    }
    return true;
}
long long factor[100];
int tol;
long long gcd(long long a, long long b) {
    if(a == 0)return 1;
    if(a < 0) return gcd(-a, b);
    while(b) {
        long long t = a % b;
        a = b;
        b = t;
    }
    return a;
}
long long Pollard_rho(long long x, long long c) {
    long long i = 1, k = 2;
    long long x0 = rand() % x;
    long long y = x0;
    while(1) {
        i++;
        x0 = (mult_mod(x0, x0, x) + c) % x;
        long long d = gcd(y - x0, x);
        if(d != 1 && d != x) return d;
        if(y == x0) return x;
        if(i == k) {
            y = x0;
            k += k;
        }
    }
}
void findfac(long long n) {
    if(n == 1) return;
    if(Miller_Rabin(n)) {
        factor[tol++] = n;
        return;
    }
    long long p = n;
    while(p >= n)p = Pollard_rho(p, rand() % (n - 1) + 1);
    findfac(p);
    findfac(n / p);
}
int main() {
    srand(time(NULL));
    long long n;
    while(scanf("%I64d", &n) != EOF) {
        tol = 0;
        findfac(n);
        for(int i = 0; i < tol; i++)printf("%I64d ",
             factor[i]);
        printf("\n");
        if(Miller_Rabin(n))printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}
```

4.7 Romberg

```
const int MAX = 18;
double f(double x) {

}
double Romberg (double a, double b) {
    #define MAX_N 18
    int i, j, temp2, min;
    double h, R[2][MAX_N], temp4;
    for (i = 0; i < MAX_N; i++) {
        R[0][i] = 0.0;
        R[1][i] = 0.0;
    }
    h = b - a;
    min = (int)(log(h * 10.0) / log(2.0)); //h should be at
         most 0.1
    R[0][0] = (f(a) + f(b)) * h * 0.50;
    i = 1;
    temp2 = 1;
    while (i < MAX_N) {
        i++;
        R[1][0] = 0.0;
        for (j = 1; j <= temp2; j++)
            R[1][0] += f(a + h * ((double)j - 0.50));
        R[1][0] = (R[0][0] + h * R[1][0]) * 0.50;
        temp4 = 4.0;
        for (j = 1; j < i; j++) {
            R[1][j] = R[1][j - 1] + (R[1][j - 1] - R[0][j -
                 1]) / (temp4 - 1.0);
            temp4 *= 4.0;
        }
        if ((fabs(R[1][i - 1] - R[0][i - 2]) < eps) && (i >
             min))
            return R[1][i - 1];
        h *= 0.50;
        temp2 *= 2;
        for (j = 0; j < i; j++)
            R[0][j] = R[1][j];
    }
    return R[1][MAX_N - 1];
}
```

4.8 Simplex

```
//a[i][0]*x[0] + a[i][1]*x[1] + ... <= a[i][n]
//max(a[m][0]*x[0] + a[m][1]*x[1] + ... + a[m][n-1]*x[n-1]
     - a[m][n])
//x[i] >= 0
const int maxm = 500; // st
const int maxn = 500; // var
const double INF = 1e100;
const double eps = 1e-10;

struct Simplex {
  int n; // var
  int m; // st
  double a[maxm][maxn]; // input matrix
  int B[maxm], N[maxn];

  void pivot(int r, int c) {
    swap(N[c], B[r]);
    a[r][c] = 1 / a[r][c];
    for(int j = 0; j <= n; j++) if(j != c) a[r][j] *=
         a[r][c];
    for(int i = 0; i <= m; i++) if(i != r) {
      for(int j = 0; j <= n; j++) if(j != c) a[i][j] -=
           a[i][c] * a[r][j];
      a[i][c] = -a[i][c] * a[r][c];
    }
  }

  bool feasible() {
    for(;;) {
      int r, c;
      double p = INF;
      for(int i = 0; i < m; i++) if(a[i][n] < p) p = a[r =
           i][n];
      if(p > -eps) return true;
      p = 0;
      for(int i = 0; i < n; i++) if(a[r][i] < p) p = a[r][c
           = i];
      if(p > -eps) return false;
      p = a[r][n] / a[r][c];
      for(int i = r+1; i < m; i++) if(a[i][c] > eps) {
        double v = a[i][n] / a[i][c];
        if(v < p) { r = i; p = v; }
      }
      pivot(r, c);
    }
  }

  // 0: no solution, -1: unlimit solution, 1: limit solution
  // b[i] = x[i], ret: opt value
  int simplex(int n, int m, double x[maxn], double& ret) {
    this->n = n;
    this->m = m;
```

```
    for(int i = 0; i < n; i++) N[i] = i;
    for(int i = 0; i < m; i++) B[i] = n+i;
    if(!feasible()) return 0;
    for(;;) {
      int r, c;
      double p = 0;
      for(int i = 0; i < n; i++) if(a[m][i] > p) p = a[m][c
          = i];
      if(p < eps) {
        for(int i = 0; i < n; i++) if(N[i] < n) x[N[i]] = 0;
        for(int i = 0; i < m; i++) if(B[i] < n) x[B[i]] =
            a[i][n];
        ret = -a[m][n];
        return 1;
      }
      p = INF;
      for(int i = 0; i < m; i++) if(a[i][c] > eps) {
        double v = a[i][n] / a[i][c];
        if(v < p) { r = i; p = v; }
      }
      if(p == INF) return -1;
      pivot(r, c);
    }
  }
};
```

4.9 Primitive Root

```
inline LL findRoot(LL p) { //p is odd prime
    LL sign, t;
    for(LL i = 1; i < p; ++i) {
        if(mu(i, p - 1, p) != 1ll) continue;
        t = p - 1; sign = 1;
        for(LL j = 2; j * j <= t; ++j){
            if(t % j != 0) continue;
            if(mu(i, (p - 1) / j, p) == 1) {
                sign = 0;
                break;
            }
            while(t % j == 0)t /= j;
        }
        if(t > 1 && mu(i, (p - 1) / t, p) == 1)sign = 0;
        if(sign)return i;
    }
}
```

4.10 Inv

```
int inv[maxn];
void initCab(int N){
    inv[0]=inv[1]=1;
    for(int x=2;x<=N;x++)
        inv[x]=(LL)(MOD-MOD/x)*inv[MOD%x]%MOD;
}
```

4.11 Berlekamp Massey

```
const int mod = 1e9 + 7;
vector<int> s, C, B;
int pow_mod(int a, int k){
    int s = 1;
    while (k){
        if (k & 1) s = 1ll * s * a % mod;
        a = 1ll * a * a % mod;
        k >>= 1;
    }
    return s;
}
int main(){
    C.push_back(1); B.push_back(1);
    int L = 0, m = 1, b = 1, n = 0, _s;
    while (scanf("%d", &_s) == 1){
        s.push_back(_s);
        int d = _s;
        for (int i = 1; i <= L; ++i)
            d = (d + 1ll * C[i] * s[n - i] % mod) % mod;
        if (d == 0) ++m;
        else if (2 * L <= n){
            vector<int> T = C;
            C.resize(n + 1 - L + 1);
            for (int i = L + 1; i <= n + 1 - L; ++i)
                C[i] = 0;
            for (int i = 0; i < B.size(); ++i)
                C[i + m] = (C[i + m] + mod - 1ll * d *
                    pow_mod(b, mod - 2) % mod * B[i] % mod)
                    % mod;
            L = n + 1 - L; B = T; b = d; m = 1;
        }
        else{
            for (int i = 0; i < B.size(); ++i)
                C[i + m] = (C[i + m] + mod - 1ll * d *
                    pow_mod(b, mod - 2) % mod * B[i] % mod)
                    % mod;
            ++m;
        }
        ++n;
        printf("F(n)=");
        for (int i = 1; i < C.size(); ++i){
            int output = (mod - C[i]) % mod;
            if (output > mod / 2)
                output -= mod;
            printf("%s%d*F(n-%d)", (output < 0 || i == 1) ?
                "" : "+", output, i);
        }
        puts("");
    }
    return 0;
}
```

4.12 Quadratic Residue

```
//二次域乘法
LL quick_mod(LL a, LL b, LL m) {
    LL ans = 1; a %= m;
    for(; b; (b & 1) && (ans = ans * a % m), a = a * a % m,
        b >>= 1);
    return ans;
}
struct T { LL p, d; };
LL w;
T multi_er(T a, T b, LL m) {
    T ans;
    ans.p = (a.p * b.p % m + a.d * b.d % m * w % m) % m;
    ans.d = (a.p * b.d % m + a.d * b.p % m) % m;
    return ans;
}
T power(T a, LL b, LL m) {
    T ans;
    ans.p = 1; ans.d = 0;
    for(; b; (b & 1) && (ans = multi_er(ans, a, m)), a =
        multi_er(a, a, m), b >>= 1);
    return ans;
}
LL Legendre(LL a, LL p) {
    return quick_mod(a, (p - 1) >> 1, p);
}
LL mod(LL a, LL m) {
    a %= m;
    if(a < 0) a += m;
    return a;
}
LL Solve(LL n, LL p) {
    n %= p;
    if(n == 0)return 0;
    if(p == 2) return 1;
    if (Legendre(n, p) + 1 == p)
        return -1;
    LL a = -1, t;
    while(true) {
        a = rand() % p;
        t = a * a - n;
        w = mod(t, p);
        if(Legendre(w, p) + 1 == p) break;
    }
    T tmp;
    tmp.p = a;
```

```
    tmp.d = 1;
    T ans = power(tmp, (p + 1) >> 1, p);
    return ans.p;
}
/*x ^ 2 = n % p
求的x, p - x为另一解
求的 -1为无解
所以 x = sqrt(n) % p  用x代替sqrt(n)
有解 称n为p的二次剩余
二次剩余*二次剩余 = 二次剩余
二次剩余*非二次剩余 = 非二次剩余
非二次剩余*非二次剩余 = 二次剩余*/
```

5 String

5.1 KMP

```
void KMP(){
    int i, j; fail[0] = fail[1] = 0;
    for(i=1;i<m;++i){
        for(j=fail[i];j && P[j]!=P[i];j=fail[j]);
        fail[i+1]= P[j]==P[i] ?j+1:0;
    }
    /* 最小循环节循环次数
        m%(m-fail[m])?1:m/(m-fail[m]); */
    for(j=0,i=0;i<n;++i){
        while(j && P[j]!=T[i]) j=fail[j];
        if(P[j]==T[i]) ++j;
        if(j==m) break;
    }
}
```

5.2 Suffix Array

```
struct SuffixArray{
    const static int maxn = 1e5 + 50;
    int sa[maxn], c[maxn], sq[maxn], sw[maxn], rank[maxn],
        height[maxn];
    void Build(int *s,int n,int m){
        ++ n;
        int * x = sq , *y = sw , p = 1 ;
        for(int i = 0 ; i < m ; ++ i) c[i] = 0;
        for(int i = 0 ; i < n ; ++ i) c[x[i]=s[i]]++;
        for(int i = 1 ; i < m ; ++ i) c[i] += c[i - 1];
        for(int i = n - 1 ; i >= 0 ; -- i) sa[--c[x[i]]] = i;
        for(int k = 1 ; p < n ; k <<= 1){
            p = 0;
            for(int i = n - k ; i < n ; ++ i) y[p ++] = i;
            for(int i = 0 ; i < n ; ++ i) if( sa[i] >= k )
                y[p ++] = sa[i] - k;
            for(int i = 0 ; i < m ; ++ i) c[i] = 0;
            for(int i = 0 ; i < n ; ++ i) c[x[y[i]]]++;
            for(int i = 1 ; i < m ; ++ i) c[i] += c[i-1];
            for(int i = n - 1 ; i >= 0 ; -- i)
                sa[--c[x[y[i]]]]=y[i];
            swap( x , y ) ; p = 1 , x[sa[0]] = 0;
            for(int i = 1 ; i < n ; ++ i) x[sa[i]] = y[sa[i
                - 1]] == y[sa[i]] && y[sa[i - 1] + k] ==
                y[sa[i] + k] ? p - 1 : p++;
            m = p;
        }
        -- n;
        for(int i = 0 ; i < n ; ++ i) sa[i] = sa[i + 1] ,
            rank[sa[i]] = i;
        for(int i = 0 , k = 0 ; i < n ; ++ i ){
            if( !rank[i] ) continue;
            int j = sa[rank[i] - 1];
            if( k ) -- k;
            while( s[i + k] == s[j + k] ) ++ k;
            height[rank[i]] = k;
        }
    }
}Sa;
```

5.3 Suffix Auto

```
struct Suffix_Auto{
    const static int maxn = ( 1e5 + 50 ) * 2;
    const static int lettersz = 26;
    int tot , last , link[maxn] , nxt[maxn][lettersz] ,
        len[maxn];
    int gethash( char c ){ return c -'a' ;}
    void init( int idx ){ link[idx] = len[idx] = 0 ,
        memset( nxt[idx] , 0 , sizeof( nxt[idx] ) ); }
    void init(){ memset(nxt[0],0,sizeof(nxt[0])) ; last =
        tot = len[0] = 0 , link[0] = -1 ; }
    void extend( char c ){
        int cur = ++ tot , p , q , id = gethash( c ) ;
        init( cur ) ;
        len[cur] = len[last] + 1;
        for( p = last ; ~p && !nxt[p][id] ; p = link[p] )
            nxt[p][id] = cur;
        if( ~p ){
            q = nxt[p][id];
            if( len[p] + 1 == len[q] ) link[cur] = q;
            else{
                // Pay attention to init for clone , such as
                    cnt array
                int clone = ++ tot ;
                len[clone] = len[p] + 1 , memcpy( nxt[clone]
                    , nxt[q] , sizeof(int)*lettersz ) ,
                    link[clone] = link[q];
                for( ; ~p && nxt[p][id] == q ; p = link[p] )
                    nxt[p][id] = clone;
                link[q] = link[cur] = clone;
            }
        }
        last = cur;
    }
}Sam;
```

5.4 Aho-Corasick Automaton

```
// Made by xiper
// Last Updata : 2016 / 2 / 02
// test status : ✓
struct AC_Auto{
    const static int LetterSize = 26; // 字符集大小
    const static int TrieSize = 26 * ( 1e5 + 50); //
        可能的所有节点总数量
    int tot; // 节点总数
    int fail[TrieSize]; // 失配函数
    int suffixlink[TrieSize]; // 后缀链接
    struct node{
        int ptr[LetterSize]; // 节点指针
        int val; // 结尾标记
    }tree[TrieSize];
    inline int GetLetterIdx(char c){ //
        获取字符哈希,在[0,LetterSize)内
        return c - 'a';
    }
    void init_node(node & s){
        memset( s.ptr , 0 , sizeof( s.ptr ) );
        s.val = 0;
    }
    void find(const char * str){
        int len = strlen( str );
        int j = 0;
        for(int i = 0 ; i < len ; ++ i){
            int idx = GetLetterIdx( str[i] );
            while(j && !tree[j].ptr[idx]) j = fail[j];
            j = tree[j].ptr[idx];
            if(tree[j].val){
                //Find
            }else if( suffixlink[j] ){
                //Find
            }
        }
    }
    void insert(const char * str){
        int len = strlen( str );
        int cur = 0;
```

```
        for(int i = 0 ; i < len ; ++ i){
            int idx = GetLetterIdx( str[i] );
            if(!tree[cur].ptr[idx]){
                tree[cur].ptr[idx] = tot;
                init_node( tree[tot++] );
            }
            cur = tree[cur].ptr[idx];
        }
        tree[cur].val ++ ;
    }
    void build_fail(){
        queue < int > Q; // 开在栈中 , 如果节点数较多 ,
             可设为全局变量
        suffixlink[0] = fail[0] = 0;
        for(int i = 0 ; i < LetterSize ; ++ i)
            if(tree[0].ptr[i]){
                int index = tree[0].ptr[i];
                fail[index] = suffixlink[index] = 0;
                Q.push( index );
            }
        while(!Q.empty()){
            int x = Q.front() ; Q.pop();
            for(int i = 0 ; i < LetterSize ; ++ i)
                if(tree[x].ptr[i]){
                    int v = tree[x].ptr[i];
                    int j = fail[x];
                    while( j && !tree[j].ptr[i] ) j = fail[j];
                    fail[v] = tree[j].ptr[i];
                    suffixlink[v] = tree[fail[v]].val ?
                         fail[v] : suffixlink[fail[v]];
                    Q.push( v );
                }
        }
    }
    void init(){ tot = 1 ; init_node( tree[0] );}
}ac_auto;
```

5.5 Minimum Representation

```
int MinimumRepresentation(char *s, int l)
{
    int i = 0, j = 1, k = 0, t;
    while(i < l && j < l && k < l) {
        t = s[(i + k) >= l ? i + k - l : i + k] - s[(j + k)
             >= l ? j + k - l : j + k];
        if(!t) k++;
        else{
            if(t > 0) i = i + k + 1;
            else j = j + k + 1;
            if(i == j) ++ j;
            k = 0;
        }
    }
    return (i < j ? i : j);
}
```

5.6 Manacher

```
void manacher(char s[],int l)
{
   int i,j,k,ans=0;
   for(i=1;i<=l;++i)str[i<<1]=s[i],str[(i<<1)+1]='#';
   str[1]='#';str[l*2+1]='#';str[0]='&';str[l*2+2]='$';
   l=l*2+1;j=0;
   for(i=1;i<=l;)
   {
      while(str[i-j-1]==str[i+j+1])++j;
      p[i]=j;if(j>ans)ans=j;
      for(k=1;k<=j&&p[i]-k!=p[i-k];++k)
           p[i+k]=min(p[i-k],p[i]-k);
      i+=k;j=max(j-k,0);
   }
}
```

5.7 Palindromic Auto

```
struct Palindromic_Auto{
    const static int maxn = 2e5 + 15;
    const static int lettersz = 26;
    int link[maxn] , len[maxn] , last , tot ,
         nxt[maxn][lettersz] , s[maxn] , n ;
    void init( int id , int l ){ memset( nxt[id] , 0 ,
         sizeof( nxt[id] ) ); len[id] = l; }
    void init(){ n = last = 0 , s[0] = -1 , link[0] = tot =
         1; init( 0 , 0 ) , init( 1 , -1 ); }
    int gethash( char c ){ return c -'a' ;}
    int extend( char c ){
        s[ ++ n ] = gethash( c );
        int cur = last ;
        while( s[n - len[cur] - 1] != s[n] ) cur = link[cur];
        if( !nxt[cur][s[n]] ){
            int id = ++ tot , t = link[cur] ; init( id ,
                 len[cur] + 2 );
            while( s[n - len[t] - 1] != s[n] ) t = link[t];
            link[id] = nxt[t][s[n]] , nxt[cur][s[n]] = id;
        }
        last = nxt[cur][s[n]];
        return len[last];
    }
}pa_auto;
```

5.8 Z Algorithm

```
int L = 0, R = 0;
for (int i = 1; i < n; i++) {
  if (i > R) {
    L = R = i;
    while (R < n && s[R-L] == s[R]) R++;
    z[i] = R-L; R--;
  } else {
    int k = i-L;
    if (z[k] < R-i+1) z[i] = z[k];
    else {
      L = i;
      while (R < n && s[R-L] == s[R]) R++;
      z[i] = R-L; R--;
    }
  }
}
```

5.9 BigInteger to Binary

```
string BigINTtoBinary(string n){
    string result="", temp="temp";
    while(temp.length()>0) {
        temp=""; int i=0; char ch;
        while(i<n.length()){
            ch=n[i]-'0';
            if(ch>=2) temp+=static_cast<char>(ch/2+'0');
            else if(ch==0||ch==1){
                if(temp.length()>0)
                    temp+='0';
            }
            if(ch%2==1&&i<n.length()-1) n[i+1]+=10;
            i++;
        }
        result=static_cast<char>(ch%2+'0')+result;
        n=temp;
    }
    return result;
}
```

6 Others

6.1 Long Long Mul

```
long long Mul(long long x, long long y) {
    return (x * y - (long long)(x / (long double)P * y +
         1e-3) * P + P) % P;
}

```

```
LL mul_mod(LL x, LL y, LL n) {// x*y % n
       LL T = floor(sqrt(n) + 0.5);
       LL t = T * T - n;
       LL a = x / T, b = x % T;
       LL c = y / T, d = y % T;
       LL e = a * c / T, f = a * c % T;
       LL v = ((a * d + b * c) % n + e * t) % n;
       LL g = v / T, h = v % T;
       LL ret = (((f + g) * t % n + b * d) % n + h * T) % n;
       return (ret % n + n) % n;
}
```

6.2 Somethings

```
c++:#pragma comment(linker, "/STACK:1024000000,1024000000");
g++:32
int size = 256 << 20; // 256MB
char *p = (char*)malloc(size) + size;
__asm__("movl %0, %%esp\n" :: "r"(p));
64
int size = 256 << 20; // 256MB
char *p = (char*)malloc(size) + size;
__asm__("movq %0, %%rsp\n" :: "r"(p));
high32
extern int main2(void) __asm__ ("_main2");
int main2() {
   //code
   exit(0);
}
int main() {
   int size = 256 << 20; // 256Mb
   char *p = (char *)malloc(size) + size;
   __asm__ __volatile__(
       "movl %0, %%esp\n"
       "pushl $_exit\n"
       "jmp _main2\n"
       :: "r"(p));
}
high64
extern int main2(void) __asm__ ("main2");
int main2() {
   //code
   exit(0);
}
int main() {
   int size = 256 << 20; // 256Mb
   char *p = (char *)malloc(size) + size;
   __asm__ __volatile__(
       "movq %0, %%rsp\n"
       "pushq $exit\n"
       "jmp main2\n"
       :: "r"(p));
}//////////////////////////////////////////////////
#include <bits/stdc++.h>
using namespace std;
extern int main2(void) __asm__("main2");
int main2(){
   // Code
   exit( 0 );
}
int main(){
   int size = 256 << 20;
   char *p = (char*)malloc(size) + size;
   __asm__ __volatile__("movq %0,%%rsp\n""pushq
        $exit\n""jmp main2\n"::"r"(p));
   return 0;
}
vector.shrink_to_fit()/***********/
default_random_engine generator((int)time(0));
uniform_int_distribution<int> dis(1,100);
//use dis(generator) []/**********************/
subset:
for (i = mask ; i ; i = (i - 1) & mask) ;
subset n,k:int k=4,n=8;
for(int comb=(1<<k)-1, temp1, temp2; comb<1<<n; temp1 =
     comb&-comb, temp2 = comb + temp1, comb =
     ((comb&~temp2) /
     temp1>>1)|temp2)/**********************/
```

6.3 Java

```
import java.io.*;
import java.util.*;
import java.math.*;

public class Main
{
   public static void main(String args[]) throws Exception
   {
       Scanner in = new Scanner(new File("in.txt"));
       PrintWriter out = new PrintWriter(new
            FileWriter("out.txt",true));
       //Scanner in = new Scanner(System.in);
       //PrintWriter out = new PrintWriter(System.out);
       int n = in.nextInt();
       out.println(ans[n]);
       out.close();
   }
}

.divide(BigDecimal.valueOf(k),100,BigDecimal.ROUND_DOWN)
charAt
BigInteger
```

6.4 Heap from Tree

```
//选择最大的结点集合使得任意两个结点i j，若i是j的祖先则a[i] >
     a[j]
void DFS(int u, int fa)
{
   for(auto &v: adj[u]){
       if(v == fa) continue;
       DFS(v, u);
       if((int)s[v].size() > (int)s[u].size())
           s[u].swap(s[v]);
       s[u].insert(s[v].begin(), s[v].end());
   }
   auto it = s[u].upper_bound(a[u]); //严格 lower 不严格
        upper
   if(it == s[u].end())
       s[u].insert(a[u]);
   else{
       s[u].erase(it);
       s[u].insert(a[u]);
   }
}
```