Algorithm template by WJMZBMR

1. AC

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <climits>
#include <numeric>
using namespace std;

const int CHARSET = 26;
const int MAX_N_NODES = int(3e5) + 10;

int pointer;
struct Node {
	Node*ch[CHARSET], *fail, *par;
	Node() {
		memset(ch, 0, sizeof ch);
		fail = 0;
	}
	Node*go(int w);
}*root;

Node nodePool[MAX_N_NODES], *cur;

Node*newNode() {
	Node*t = cur++;
	memset(t->ch, 0, sizeof t->ch);
	t->fail = 0;
	return t;
}

Node* Node::go(int w) {
	if (ch[w] == 0) {
		ch[w] = newNode();
		ch[w]->par = this;
	}
	return ch[w];
}

void init() {
	cur = nodePool;
	root = newNode();
	root->par = 0;
}

void build() {
	static Node*que[MAX_N_NODES];
	int qh = 0, qt = 0;
	que[qt++] = root;
	while (qh < qt) {
		Node*t = que[qh++];
		for (int c = 0; c < CHARSET; ++c) {
			Node*v = t->ch[c];
			if (!v)
				continue;
			Node*f = t->fail;
			while (f && f->ch[c] == 0)
				f = f->fail;
```

```
                        if (f == 0)
                                v->fail = root;
                        else
                                v->fail = f->ch[c];
                        que[qt++] = v;
                }
        }
}
```

2. Circle

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <climits>
#include <cstring>
#include <vector>
#include <cmath>
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
using namespace std;
struct Point {
        double x, y;
        Point() {
        }
        Point(double _x, double _y) :
                        x(_x), y(_y) {
        }
        Point operator+(const Point&p) const {
                return Point(x + p.x, y + p.y);
        }
        Point operator-(const Point&p) const {
                return Point(x - p.x, y - p.y);
        }
        Point operator*(double d) const {
                return Point(x * d, y * d);
        }
        Point operator/(double d) const {
                return Point(x / d, y / d);
        }
        double det(const Point&p) const {
                return x * p.y - y * p.x;
        }
        double dot(const Point&p) const {
                return x * p.x + y * p.y;
        }
        double alpha() const {
                return atan2(y, x);
        }
        Point rot90() const {
                return Point(-y, x);
        }
        void read() {
                scanf("%lf%lf", &x, &y);
        }
        void write() const {
                printf("(%lf,%lf)", x, y);
        }
        double abs() {
                return hypot(x, y);
```

```
	}
	double abs2() {
		return x * x + y * y;
	}
	Point unit() {
		return *this / abs();
	}
	double distTo(const Point&p) const {
		return hypot(x - p.x, y - p.y);
	}
};

#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))

const double EPS = 1e-8; //you should change it according to problem, nevertheless, it
mustn't be a constant value some times.
inline int sign(double a) {
	return a < -EPS ? -1 : a > EPS;
}

#define crossOp(p1,p2,p3) (sign(cross(p1,p2,p3)))

Point isSS(Point p1, Point p2, Point q1, Point q2) {
	double a1 = cross(q1,q2,p1), a2 = -cross(q1,q2,p2);
	return (p1 * a2 + p2 * a1) / (a1 + a2);
}

typedef pair<double, double> dpair;

vector<Point> make(Point a, Point b) {
	vector<Point> ret;
	ret.push_back(a);
	ret.push_back(b);
	return ret;
}

vector<Point> tanCP(Point c, double r, Point p) {
	double x2 = (p - c).abs2();
	double d2 = x2 - r * r;
	vector<Point> ret;
	if (d2 < -EPS)
		return ret;
	if (r <= EPS) {
		ret.push_back(c);
		ret.push_back(c);
		return ret;
	}
	d2 = max(d2, 0.);
	Point q1 = c + (p - c) * (r * r / x2);
	Point q2 = (p - c).rot90() * (-r * sqrt(d2) / x2);
	ret.push_back(q1 - q2);
	ret.push_back(q1 + q2);
	return ret;
}

vector<vector<Point> > tanCC(Point c1, double r1, Point c2, double r2) {
	vector<vector<Point> > ret;
	if (fabs(r1 - r2) <= EPS) {
```

```
            Point dir = (c2 - c1).unit().rot90() * r1;
            ret.push_back(make(c1 + dir, c2 + dir));
            ret.push_back(make(c1 - dir, c2 - dir));
        } else {
            Point p = (c2 * r1 - c1 * r2) / (r1 - r2);
            vector<Point> ps = tanCP(c1, r1, p);
            vector<Point> qs = tanCP(c2, r2, p);
            for (int i = 0; i < ps.size() && i < qs.size(); ++i) {
                ret.push_back(make(ps[i], qs[i]));
            }
        }
        return ret;
}
```

3. Delaunay.java

```
class Delaunay{
    class Point implements Comparable<Point> {
        long x, y;

        public int compareTo(Point o) {
            if (x != o.x)
                return Long.signum(x - o.x);
            return Long.signum(y - o.y);
        }

        Point(long x, long y) {
            this.x = x;
            this.y = y;
        }

        Point3D get() {
            return new Point3D(x, y, x * x + y * y);
        }

        Point sub(Point p) {
            return new Point(x - p.x, y - p.y);
        }

        long det(Point p) {
            return x * p.y - y * p.x;
        }

        long abs2() {
            return x * x + y * y;
        }

        long distTo2(Point p) {
            return sub(p).abs2();
        }
    }

    long cross(Point p1, Point p2, Point p3) {
        return p2.sub(p1).det(p3.sub(p1));
    }

    int crossOp(Point p1, Point p2, Point p3) {
        return Long.signum(cross(p1, p2, p3));
    }
```

```
    boolean crsSS(Point p1, Point p2, Point q1, Point q2) {
        return crossOp(p1, p2, q1) * crossOp(p1, p2, q2) < 0 &&
                crossOp(q1, q2, p1) * crossOp(q1, q2, p2) < 0;
    }

    class Point3D {
        long x, y, z;

        Point3D(long x, long y, long z) {
            this.x = x;
            this.y = y;
            this.z = z;
        }

        long dot(Point3D p) {
            return x * p.x + y * p.y + z * p.z;
        }

        Point3D det(Point3D p) {
            return new Point3D(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y *
p.x);
        }

        Point3D sub(Point3D p) {
            return new Point3D(x - p.x, y - p.y, z - p.z);
        }
    }

    int inCircle(Point a, Point b, Point c, Point d) {
        b = b.sub(a);
        c = c.sub(a);
        d = d.sub(a);
        if (b.det(c) < 0) {
            Point t = b;
            b = c;
            c = t;
        }
        Point3D pb = b.get(), pc = c.get(), pd = d.get();
        Point3D o = pb.det(pc);
        return Long.signum(pd.dot(o));
    }

    class PointId extends Point {
        int id;
        List<PointId> edges = new LinkedList<PointId>();

        PointId(long x, long y, int id) {
            super(x, y);
            this.id = id;
        }

        void addEdge(PointId t) {
            edges.add(t);
            t.edges.add(this);
        }
    }
```

```
    int n;
    PointId[] ps;

    void construct(int l, int r) {//[l,r)
        if (r - l <= 3) {
            for (int i = l; i < r; i++) {
                for (int j = l + 1; j < r; j++) {
                    ps[i].addEdge(ps[j]);
                }
            }
            return;
        }
        int m = (l + r) / 2;
        construct(l, m);
        construct(m, r);

        //find the common tangent
        PointId pl = ps[l], pr = ps[r - 1];
        for (; ; ) {
            PointId next = null;
            for (PointId p : pl.edges) {
                int op = crossOp(pr, pl, p);
                if (op > 0 || (op == 0 && p.distTo2(pr) < pl.distTo2(pr))) {
                    next = p;
                    break;
                }
            }
            if (next != null)
                pl = next;
            else {
                next = null;
                for (PointId p : pr.edges) {
                    int op = crossOp(pr, pl, p);
                    if (op > 0 || (op == 0 && p.distTo2(pl) < pr.distTo2(pl))) {
                        next = p;
                        break;
                    }
                }
                if (next != null)
                    pr = next;
                else
                    break;
            }
        }

        //merge
        pl.addEdge(pr);
        pr.addEdge(pl);
        for (; ; ) {
            PointId next = null;
            boolean which = false;
            for (PointId p : pl.edges) {
                if (crossOp(pr, pl, p) < 0 && (next == null || inCircle(next, pl, pr,
p) == -1))
                    next = p;
            }
            for (PointId p : pr.edges) {
```

```
                if (crossOp(pl, pr, p) > 0 && (next == null || inCircle(next, pl, pr,
p) == -1)) {
                    next = p;
                    which = true;
                }
            }
            if (next == null)
                break;
            if (!which) {//pl
                List<PointId> nEdges = new ArrayList<PointId>();
                for (PointId p : pl.edges) {
                    if (!crsSS(next, pr, pl, p))
                        nEdges.add(p);
                }
                pl.edges = nEdges;
                pr.addEdge(next);
                pl = next;
            } else {//pr
                List<PointId> nEdges = new ArrayList<PointId>();
                for (PointId p : pr.edges) {
                    if (!crsSS(next, pl, pr, p))
                        nEdges.add(p);
                }
                pr.edges = nEdges;
                pl.addEdge(next);
                pr = next;
            }
        }
    }
}
```

4. FFT

```
#include <algorithm>
#include <complex>
#include <vector>
#include <cmath>
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
using namespace std;

typedef complex<double> Comp;

const Comp I(0, 1);

const int MAX_N = 1 << 20;
Comp tmp[MAX_N];
void DFT(Comp*a, int n, int rev) {
	if (n == 1)
		return;
	for (int i = 0; i < n; ++i) {
		tmp[i] = a[i];
	}
	for (int i = 0; i < n; ++i) {
		if (i & 1)
			a[n / 2 + i / 2] = tmp[i];
		else
			a[i / 2] = tmp[i];
	}
	Comp*a0 = a, *a1 = a + n / 2;
```

```
	DFT(a0, n / 2, rev);
	DFT(a1, n / 2, rev);
	Comp cur(1, 0);
	double alpha = 2 * M_PI / n * rev;
	Comp step = exp(I * alpha);
	for (int k = 0; k < n / 2; ++k) {
		tmp[k] = a0[k] + cur * a1[k];
		tmp[k + n / 2] = a0[k] - cur * a1[k];
		cur *= step;
	}
	for (int i = 0; i < n; ++i) {
		a[i] = tmp[i];
	}
}

int main() {
	static Comp a[1 << 20] = { }, b[1 << 20] = { };
	int n = 1 << 20;
	DFT(a, n, 1);
	DFT(b, n, 1);
	for (int i = 0; i < n; ++i) {
		a[i] *= b[i];
	}
	DFT(a, n, -1);
	for (int i = 0; i < n; ++i) {
		a[i] /= n;
	}
}
```

5. Geo

```
/*
 * Geo.cpp
 *
 *  Created on: 2012-11-2
 *      Author: mac
 */
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <climits>
#include <numeric>
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
#define REP(i,n) for(int i=0;i<n;++i)
using namespace std;

const double EPS = 1e-8;
inline int sign(double a) {
	return a < -EPS ? -1 : a > EPS;
}

struct Point {
	double x, y;
	Point() {
	}
	Point(double _x, double _y) :
			x(_x), y(_y) {
	}
```

```
	Point operator+(const Point&p) const {
		return Point(x + p.x, y + p.y);
	}
	Point operator-(const Point&p) const {
		return Point(x - p.x, y - p.y);
	}
	Point operator*(double d) const {
		return Point(x * d, y * d);
	}
	Point operator/(double d) const {
		return Point(x / d, y / d);
	}
	bool operator<(const Point&p) const {
		int c = sign(x - p.x);
		if (c)
			return c == -1;
		return sign(y - p.y) == -1;
	}
	double dot(const Point&p) const {
		return x * p.x + y * p.y;
	}
	double det(const Point&p) const {
		return x * p.y - y * p.x;
	}
	double alpha() const {
		return atan2(y, x);
	}
	double distTo(const Point&p) const {
		double dx = x - p.x, dy = y - p.y;
		return hypot(dx, dy);
	}
	double alphaTo(const Point&p) const {
		double dx = x - p.x, dy = y - p.y;
		return atan2(dy, dx);
	}
	void read() {
		scanf("%lf%lf", &x, &y);
	}
	double abs() {
		return hypot(x, y);
	}
	double abs2() {
		return x * x + y * y;
	}
	void write() {
		cout << "(" << x << "," << y << ")" << endl;
	}
};

#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))

#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))

Point isSS(Point p1, Point p2, Point q1, Point q2) {
	double a1 = cross(q1,q2,p1), a2 = -cross(q1,q2,p2);
	return (p1 * a2 + p2 * a1) / (a1 + a2);
}
```

```
vector<Point> convexCut(const vector<Point>&ps, Point q1, Point q2) {
	vector<Point> qs;
	int n = ps.size();
	for (int i = 0; i < n; ++i) {
		Point p1 = ps[i], p2 = ps[(i + 1) % n];
		int d1 = crossOp(q1,q2,p1), d2 = crossOp(q1,q2,p2);
		if (d1 >= 0)
			qs.push_back(p1);
		if (d1 * d2 < 0)
			qs.push_back(isSS(p1, p2, q1, q2));
	}
	return qs;
}

double calcArea(const vector<Point>&ps) {
	int n = ps.size();
	double ret = 0;
	for (int i = 0; i < n; ++i) {
		ret += ps[i].det(ps[(i + 1) % n]);
	}
	return ret / 2;
}

vector<Point> convexHull(vector<Point> ps) {
	int n = ps.size();
	if (n <= 1)
		return ps;
	sort(ps.begin(), ps.end());
	vector<Point> qs;
	for (int i = 0; i < n; qs.push_back(ps[i++])) {
		while (qs.size() > 1 && crossOp(qs[qs.size()-2],qs.back(),ps[i]) <= 0)
			qs.pop_back();
	}
	for (int i = n - 2, t = qs.size(); i >= 0; qs.push_back(ps[i--])) {
		while (qs.size() > t && crossOp(qs[qs.size()-2],qs.back(),ps[i]) <= 0)
			qs.pop_back();
	}
	qs.pop_back();
	return qs;
}

double convexDiameter(const vector<Point>&ps) {
	int n = ps.size();
	int is = 0, js = 0;
	for (int i = 1; i < n; ++i) {
		if (ps[i].x > ps[is].x)
			is = i;
		if (ps[i].x < ps[js].x)
			js = i;
	}
	double maxd = ps[is].distTo(ps[js]);
	int i = is, j = js;
	do {
		if ((ps[(i + 1) % n] - ps[i]).det(ps[(j + 1) % n] - ps[j]) >= 0)
			(++j) %= n;
		else
			(++i) %= n;
		maxd = max(maxd, ps[i].distTo(ps[j]));
```

```
	} while (i != is || j != js);
	return maxd;
}
```

6. Geo3D

```
/*
 * Geo3D.cpp
 *
 *  Created on: 2012-11-2
 *      Author: mac
 */
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <climits>
#include <numeric>
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
#define REP(i,n) for(int i=0;i<n;++i)
using namespace std;

const int MAX_N_POINTS = 10000 + 10;
const int MAX_N_FACES = 10000 + 10;

const double EPS = 1e-8;

int sign(double x) {
	return x < -EPS ? -1 : x > EPS;
}

struct Point {
	double x, y, z;
	void read() {
		scanf("%lf%lf%lf", &x, &y, &z);
	}
	Point() {
	}
	Point(double _x, double _y, double _z) :
			x(_x), y(_y), z(_z) {
	}
	Point operator+(Point p) {
		return Point(x + p.x, y + p.y, z + p.z);
	}
	Point operator-(Point p) {
		return Point(x - p.x, y - p.y, z - p.z);
	}
	Point operator*(double f) {
		return Point(x * f, y * f, z * f);
	}
	Point operator/(double f) {
		return Point(x / f, y / f, z / f);
	}
	double dot(Point p) {
		return x * p.x + y * p.y + z * p.z;
	}
	Point det(Point p) {
		return Point(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y * p.x);
	}
```

```
	double abs() {
		return sqrt(abs2());
	}
	double abs2() {
		return x * x + y * y + z * z;
	}
	Point norm() {
		return *this / abs();
	}
	void write() {
		cout << x << " " << y << " " << z << endl;
	}
};

//a plane (*-p) dot o ==0

//get a plane form p1,p2,p3
void set(Point p1, Point p2, Point p3, Point&p, Point&o) {
	o = (p2 - p1).det(p3 - p1).norm();
	p = p1;
}

double disFP(Point p, Point o, Point q) { //plane point
	return (q - p).dot(o);
}

double disLP(Point p1, Point p2, Point q) { //line point
	return (p2 - p1).det(q - p1).abs() / (p2 - p1).abs();
}

double disLL(Point p1, Point p2, Point q1, Point q2) {
	Point p = q1 - p1;
	Point u = p2 - p1;
	Point v = q2 - q1;
	double d = u.abs2() * v.abs2() - u.dot(v) * u.dot(v);
	if (abs(d) < EPS)
		return disLP(q1, q2, p1);
	double s = (p.dot(u) * v.abs2() - p.dot(v) * u.dot(v)) / d;
	return disLP(q1, q2, p1 + u * s);
}

Point isFL(Point p, Point o, Point q1, Point q2) {
	double a = o.dot(q2 - p);
	double b = o.dot(q1 - p);
	double d = a - b;
	if (abs(d) < EPS)
		throw "none";
	return (q1 * a - q2 * b) / d;
}

vector<Point> isFF(Point p1, Point o1, Point p2, Point o2) {
	Point e = o1.det(o2);
	Point v = o1.det(e);
	double d = o2.dot(v);
	if (abs(d) < EPS)
		throw "none";
	Point q = p1 + (v * (o2.dot(p2 - p1)) / d);
	vector<Point> ret;
```

```
	ret.push_back(q);
	ret.push_back(q + e);
	return ret;
}

int main() {
	return 0;
}
```

7. HalfPlaneIntersection

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <climits>
#include <cstring>
#include <cmath>
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
using namespace std;

struct Point {
	long double x, y;
	Point() {
	}
	Point(long double _x, long double _y) :
			x(_x), y(_y) {
	}
	Point operator+(const Point&p) const {
		return Point(x + p.x, y + p.y);
	}
	Point operator-(const Point&p) const {
		return Point(x - p.x, y - p.y);
	}
	Point operator*(long double d) const {
		return Point(x * d, y * d);
	}
	Point operator/(long double d) const {
		return Point(x / d, y / d);
	}
	long double det(const Point&p) const {
		return x * p.y - y * p.x;
	}
	long double dot(const Point&p) const {
		return x * p.x + y * p.y;
	}
	Point rot90() const {
		return Point(-y, x);
	}
	void read() {
		cin >> x >> y;
	}
	void write() const {
		printf("%lf %lf", x, y);
	}
};

#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))

const long double EPS = 1e-12;
```

```
inline int sign(long double a) {
	return a < -EPS ? -1 : a > EPS;
}

#define crossOp(p1,p2,p3) (sign(cross(p1,p2,p3)))

Point isSS(Point p1, Point p2, Point q1, Point q2) {
	long double a1 = cross(q1,q2,p1), a2 = -cross(q1,q2,p2);
	return (p1 * a2 + p2 * a1) / (a1 + a2);
}

struct Border {
	Point p1, p2;
	long double alpha;
	void setAlpha() {
		alpha = atan2(p2.y - p1.y, p2.x - p1.x);
	}
	void read() {
		p1.read();
		p2.read();
		setAlpha();
	}
};

int n;
const int MAX_N_BORDER = 20000 + 10;
Border border[MAX_N_BORDER];

bool operator<(const Border&a, const Border&b) {
	int c = sign(a.alpha - b.alpha);
	if (c != 0)
		return c == 1;
	return crossOp(b.p1,b.p2,a.p1) >= 0;
}

bool operator==(const Border&a, const Border&b) {
	return sign(a.alpha - b.alpha) == 0;
}

const long double LARGE = 10000;

void add(long double x, long double y, long double nx, long double ny) {
	border[n].p1 = Point(x, y);
	border[n].p2 = Point(nx, ny);
	border[n].setAlpha();
	n++;
}

Point isBorder(const Border&a, const Border&b) {
	return isSS(a.p1, a.p2, b.p1, b.p2);
}

Border que[MAX_N_BORDER];
int qh, qt;

bool check(const Border&a, const Border&b, const Border&me) {
	Point is = isBorder(a, b);
	return crossOp(me.p1,me.p2,is) > 0;
```

```
}

void convexIntersection() {
	qh = qt = 0;
	sort(border, border + n);
	n = unique(border, border + n) - border;
	for (int i = 0; i < n; ++i) {
		Border cur = border[i];
		while (qh + 1 < qt && !check(que[qt - 2], que[qt - 1], cur))
			--qt;
		while (qh + 1 < qt && !check(que[qh], que[qh + 1], cur))
			++qh;
		que[qt++] = cur;
	}
	while (qh + 1 < qt && !check(que[qt - 2], que[qt - 1], que[qh]))
		--qt;
	while (qh + 1 < qt && !check(que[qh], que[qh + 1], que[qt - 1]))
		++qh;
}

void calcArea() {
	static Point ps[MAX_N_BORDER];
	int cnt = 0;

	if (qt - qh <= 2) {
		puts("0.0");
		return;
	}

	for (int i = qh; i < qt; ++i) {
		int next = i + 1 == qt ? qh : i + 1;
		ps[cnt++] = isBorder(que[i], que[next]);
	}

	long double area = 0;
	for (int i = 0; i < cnt; ++i) {
		area += ps[i].det(ps[(i + 1) % cnt]);
	}
	area /= 2;
	area = fabsl(area);
	cout.setf(ios::fixed);
	cout.precision(1);
	cout << area << endl;
}

int main() {
	cin >> n;
	for (int i = 0; i < n; ++i) {
		border[i].read();
	}
	add(0, 0, LARGE, 0);
	add(LARGE, 0, LARGE, LARGE);
	add(LARGE, LARGE, 0, LARGE);
	add(0, LARGE, 0, 0);

	convexIntersection();
	calcArea();
}
```

8. LCT

```
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<vector>
#define REP(i,n) for(int i=0;i<n;++i)
using namespace std;

//Dynamic Tree

typedef long long int64;
const int MOD = 51061;
const int MAX_N = int(1e5) + 10;

struct Mark {
	int64 add, mul; //x*mul+add
	Mark(int64 add, int64 mul) {
		this->add = add;
		this->mul = mul;
	}
	Mark() {
		mul = 1;
		add = 0;
	}
	bool isId() {
		return mul == 1 && add == 0;
	}
};

Mark operator*(Mark a, Mark b) {
	return Mark((a.add * b.mul + b.add) % MOD, a.mul * b.mul % MOD);
}

struct Node {
	Node*p, *ch[2];
	bool rev;
	Mark m;
	int64 sum, val;
	int size;
	bool isRoot;
	Node*fa;
	Node() {
		sum = 0;
		isRoot = 0;
		size = 0;
	}
	void sc(Node*c, int d) {
		ch[d] = c;
		c->p = this;
	}
	bool d() {
		return this == p->ch[1];
	}
	void upd() {
		sum = (val + ch[0]->sum + ch[1]->sum) % MOD;
		size = 1 + ch[0]->size + ch[1]->size;
	}
```

```
	void apply(Mark a) {
		m = m * a;
		sum = (sum * a.mul + a.add * size) % MOD;
		val = (val * a.mul + a.add) % MOD;
	}
	void revIt() {
		rev ^= 1;
		swap(ch[0], ch[1]);
	}
	void relax();
	void setRoot(Node*f);
} Tnull, *null = &Tnull;

void Node::setRoot(Node*f) {
	fa = f;
	isRoot = true;
	p = null;
}

void Node::relax() {
	if (!m.isId()) {
		REP(i,2)
			if (ch[i] != null)
				ch[i]->apply(m);
		m = Mark();
	}
	if (rev) {
		REP(i,2)
			if (ch[i] != null)
				ch[i]->revIt();
		rev = 0;
	}
}

Node mem[MAX_N], *C = mem;

Node*make(int v) {
	C->sum = C->val = v;
	C->rev = 0;
	C->m = Mark();
	C->ch[0] = C->ch[1] = null;
	C->isRoot = true;
	C->p = null;
	C->fa = null;
	return C++;
}

void rot(Node*t) {
	Node*p = t->p;
	p->relax();
	t->relax();
	bool d = t->d();
	p->p->sc(t, p->d());
	p->sc(t->ch[!d], d);
	t->sc(p, !d);
	p->upd();
	if (p->isRoot) {
		p->isRoot = false;
```

```
		t->isRoot = true;
		t->fa = p->fa;
	}
}

void pushTo(Node*t) {
	static Node*stk[MAX_N];
	int top = 0;
	while (t != null) {
		stk[top++] = t;
		t = t->p;
	}
	for (int i = top - 1; i >= 0; --i)
		stk[i]->relax();
}

void splay(Node*u, Node*f = null) {
	pushTo(u);
	while (u->p != f) {
		if (u->p->p == f)
			rot(u);
		else
			u->d() == u->p->d() ? (rot(u->p), rot(u)) : (rot(u), rot(u));
	}
	u->upd();
}

Node*v[MAX_N];
vector<int> E[MAX_N];
int n, nQ;

int que[MAX_N], fa[MAX_N], qh = 0, qt = 0;

void bfs() {
	que[qt++] = 0;
	fa[0] = -1;
	while (qh < qt) {
		int u = que[qh++];
		for (vector<int>::iterator e = E[u].begin(); e != E[u].end(); ++e)
			if (*e != fa[u])
				fa[*e] = u, v[*e]->fa = v[u], que[qt++] = *e;
	}
}

Node* expose(Node*u) {
	Node*v;
	for (v = null; u != null; v = u, u = u->fa) {
		splay(u);
		u->ch[1]->setRoot(u);
		u->sc(v, 1);
		v->fa = u;
	}
	return v;
}

void makeRoot(Node*u) {
	expose(u);
	splay(u);
```

```
	u->revIt();
}

void addEdge(Node*u, Node*v) {
	makeRoot(v);
	v->fa = u;
}

void delEdge(Node*u, Node*v) {
	makeRoot(u);
	expose(v);
	splay(u);
	u->sc(null, 1);
	u->upd();
	v->fa = null;
	v->isRoot = true;
	v->p = null;
}

void markPath(Node*u, Node*v, Mark m) {
	makeRoot(u);
	expose(v);
	splay(v);
	v->apply(m);
}

int queryPath(Node*u, Node*v) {
	makeRoot(u);
	expose(v);
	splay(v);
	return v->sum;
}

int main() {
	scanf("%d%d", &n, &nQ);
	REP(i,n-1) {
		int u, v;
		scanf("%d%d", &u, &v);
		--u, --v;
		E[u].push_back(v);
		E[v].push_back(u);
	}
	REP(i,n)
		v[i] = make(1);
	bfs();
	REP(i,nQ) {
		char cmd;
		scanf(" ");
		scanf("%c", &cmd);
		int i, j;
		scanf("%d%d", &i, &j);
		Node*u = ::v[--i], *v = ::v[--j];
		if (cmd == '+') {
			int c;
			scanf("%d", &c);
			markPath(u, v, Mark(c, 1));
		} else if (cmd == '*') {
			int c;
```

```
                scanf("%d", &c);
                markPath(u, v, Mark(0, c));
            } else if (cmd == '/') {
                printf("%d\n", queryPath(u, v));
            } else {
                int k, l;
                scanf("%d%d", &k, &l);
                delEdge(u, v);
                addEdge(::v[--k], ::v[--l]);
            }
        }
}
```

9. Manacher

```
#include <algorithm>
using namespace std;

void palindrome(char cs[], int len[], int n) { //len[i] means the max palindrome
length centered i/2
        for (int i = 0; i < n * 2; ++i) {
                len[i] = 0;
        }
        for (int i = 0, j = 0, k; i < n * 2; i += k, j = max(j - k, 0)) {
                while (i - j >= 0 && i + j + 1 < n * 2 && cs[(i - j) / 2] == cs[(i + j +
1) / 2])
                        j++;
                len[i] = j;
                for (k = 1; i - k >= 0 && j - k >= 0 && len[i - k] != j - k; k++) {
                        len[i + k] = min(len[i - k], j - k);
                }
        }
}
```

10. MinCostFlow

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <climits>
#include <numeric>
#include <vector>
#include <queue>
using namespace std;

template<class Flow = int, class Cost = int>
struct MinCostFlow {
        struct Edge {
                int t;
                Flow f;
                Cost c;
                Edge*next, *rev;
                Edge(int _t, Flow _f, Cost _c, Edge*_next) :
                                t(_t), f(_f), c(_c), next(_next) {
                }
        };

        vector<Edge*> E;
```

```
	int addV() {
		E.push_back((Edge*) 0);
		return E.size() - 1;
	}

	Edge* makeEdge(int s, int t, Flow f, Cost c) {
		return E[s] = new Edge(t, f, c, E[s]);
	}

	void addEdge(int s, int t, Flow f, Cost c) {
		Edge*e1 = makeEdge(s, t, f, c), *e2 = makeEdge(t, s, 0, -c);
		e1->rev = e2, e2->rev = e1;
	}

	pair<Flow, Cost> minCostFlow(int vs, int vt) { //flow,cost
		int n = E.size();
		Flow flow = 0;
		Cost cost = 0;
//		const Cost MAX_COST = numeric_limits<Cost>::max();
//		const Flow MAX_FLOW = numeric_limits<Flow>::max();
		const Cost MAX_COST = ~0U >> 1;
		const Flow MAX_FLOW = ~0U >> 1;
		for (;;) {
			vector<Cost> dist(n, MAX_COST);
			vector<Flow> am(n, 0);
			vector<Edge*> prev(n);
			vector<bool> inQ(n, false);
			queue<int> que;

			dist[vs] = 0;
			am[vs] = MAX_FLOW;
			que.push(vs);
			inQ[vs] = true;

			while (!que.empty()) {
				int u = que.front();
				Cost c = dist[u];
				que.pop();
				inQ[u] = false;
				for (Edge*e = E[u]; e; e = e->next)
					if (e->f > 0) {
						Cost nc = c + e->c;
						if (nc < dist[e->t]) {
							dist[e->t] = nc;
							prev[e->t] = e;
							am[e->t] = min(am[u], e->f);
							if (!inQ[e->t]) {
								que.push(e->t);
								inQ[e->t] = true;
							}
						}
					}
			}

			if (dist[vt] == MAX_COST)
				break;
```

```
                        Flow by = am[vt];
                        int u = vt;
                        flow += by;
                        cost += by * dist[vt];
                        while (u != vs) {
                                Edge*e = prev[u];
                                e->f -= by;
                                e->rev->f += by;
                                u = e->rev->t;
                        }
                }

                return make_pair(flow, cost);
        }
};

int main() {
        return 0;
}
```

11. SA

```
#include <algorithm>
#include <numeric>
#include <cassert>
const int MAX_LEN = 100000;
using namespace std;
struct SuffixArray {
        int n;
        int m[2][MAX_LEN];
        int sa[MAX_LEN];

        void indexSort(int sa[], int ord[], int id[], int nId) { //ord is the ordering
get from prev stage
                static int cnt[MAX_LEN];
                memset(cnt, 0, sizeof(0) * nId);

                for (int i = 0; i < n; ++i) {
                        cnt[id[i]]++;
                }
                partial_sum(cnt, cnt + nId, cnt);
                for (int i = n - 1; i >= 0; --i) {
                        sa[--cnt[id[ord[i]]]] = ord[i];
                }
        }

        int*id, *oId;
        void init(int s[], int _n) { //s[n] == 0
                n = _n;
                assert(s[n - 1] == *min_element(s, s + n));
                static int w[MAX_LEN];
                memcpy(w, s, sizeof(int) * n);
                sort(w, w + n);
                int nId = unique(w, w + n) - w;
                id = m[0], oId = m[1];
                for (int i = 0; i < n; ++i) {
                        id[i] = lower_bound(w, w + nId, s[i]) - w;
                }
                static int ord[MAX_LEN];
```

```
		for (int i = 0; i < n; ++i) {
			ord[i] = i;
		}
		indexSort(sa, ord, id, nId);
		for (int k = 1; k <= n && nId < n; k <<= 1) {
			//get the prev order
			// k -> k*2
			int cur = 0;
			for (int i = n - k; i < n; ++i) {
				ord[cur++] = i;
			}
			for (int i = 0; i < n; ++i) {
				if (sa[i] >= k)
					ord[cur++] = sa[i] - k;
			}
			indexSort(sa, ord, id, nId);
			//get new id
			cur = 0;
			swap(oId, id);
			for (int i = 0; i < n; ++i) {
				int c = sa[i], p = i ? sa[i - 1] : 0;
				id[c] = (i == 0 || oId[c] != oId[p] || oId[c + k] != oId[p +
k]) ? cur++ : cur - 1;
			}
			nId = cur;
		}
	}
} sa;
```

12. SAM

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <climits>
#include <numeric>
#include <vector>
using namespace std;

const int MAX_N = 1000000 + 10;
struct State {
	State*suf, *go[26], *nxt;
	int val, cnt;
	State() :
			suf(0), val(0) {
		memset(go, 0, sizeof go);
	}
}*root, *last;
State statePool[MAX_N * 2], *cur;
State*first[MAX_N] = { };

void init() {
	cur = statePool;
	root = last = cur++;
}

void extend(int w) {
	State*p = last, *np = cur++;
```

```
	np->val = p->val + 1;
	np->cnt = 1;
	while (p && !p->go[w])
		p->go[w] = np, p = p->suf;
	if (!p)
		np->suf = root;
	else {
		State*q = p->go[w];
		if (p->val + 1 == q->val) {
			np->suf = q;
		} else {
			State*nq = cur++;
			memcpy(nq->go, q->go, sizeof q->go);
			nq->val = p->val + 1;
			nq->suf = q->suf;
			q->suf = nq;
			np->suf = nq;
			while (p && p->go[w] == q)
				p->go[w] = nq, p = p->suf;
		}
	}
	last = np;
}

int main() {
	string str;
	cin >> str;
	init();
	int L = str.size();
	for (int i = 0; i < L; ++i) {
		extend(str[i] - 'a');
	}
	for (State*i = statePool; i != cur; ++i)
		i->nxt = first[i->val], first[i->val] = i;
	for (int it = L; it >= 0; --it) {
		for (State*i = first[it]; i; i = i->nxt)
			if (i->suf)
				i->suf->cnt += i->cnt;
	}
//	cout << root->go[0]->go[0]->cnt << endl;
	return 0;
}
```

13. SAP

```
/*
 * SAP.cpp
 *
 *  Created on: 2012-10-28
 *      Author: mac
 */
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <climits>
#include <numeric>
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
#define REP(i,n) for(int i=0;i<n;++i)
```

```
using namespace std;

template<class Flow>
struct Maxflow {
	static const Flow INF = ~0U >> 1; //should change with type
//	static const Flow INF = numeric_limits<Flow>::max();
	struct Edge {
		int t;
		Flow c;
		Edge*n, *r;
		Edge(int _t, Flow _c, Edge*_n) :
				t(_t), c(_c), n(_n) {
		}
	};
	vector<Edge*> E;

	int addV() {
		E.push_back((Edge*) 0);
		return E.size() - 1;
	}

	void clear() {
		E.clear();
	}

	Edge* makeEdge(int s, int t, Flow c) {
		return E[s] = new Edge(t, c, E[s]);
	}

	void addEdge(int s, int t, Flow c) {
		Edge*e1 = makeEdge(s, t, c), *e2 = makeEdge(t, s, 0);
		e1->r = e2, e2->r = e1;
	}

	int calcMaxFlow(int vs, int vt) {
		int nV = E.size();
		Flow totalFlow = 0;

		vector<Flow> am(nV, 0);
		vector<int> h(nV, 0), cnt(nV + 1, 0);
		vector<Edge*> prev(nV, (Edge*) 0), cur(nV, (Edge*) 0);
		cnt[0] = nV;

		int u = vs;
		Edge*e;
		am[u] = INF;
		while (h[vs] < nV) {
			for (e = cur[u]; e; e = e->n)
				if (e->c > 0 && h[u] == h[e->t] + 1)
					break;
			if (e) {
				int v = e->t;
				cur[u] = prev[v] = e;
				am[v] = min(am[u], e->c);
				u = v;
				if (u == vt) {
					Flow by = am[u];
					while (u != vs) {
```

```
                                        prev[u]->c -= by;
                                        prev[u]->r->c += by;
                                        u = prev[u]->r->t;
                                }
                                totalFlow += by;
                                am[u] = INF;
                        }
                } else {
                        if (!--cnt[h[u]])
                                break;
                        h[u] = nV;
                        for (e = E[u]; e; e = e->n)
                                if (e->c > 0 && h[e->t] + 1 < h[u]) {
                                        h[u] = h[e->t] + 1;
                                        cur[u] = e;
                                }
                        ++cnt[h[u]];
                        if (u != vs)
                                u = prev[u]->r->t;
                }
        }

        return totalFlow;
    }

    ~Maxflow() {
        for (int i = 0; i < E.size(); ++i) {
            for (Edge*e = E[i]; e;) {
                Edge*ne = e->n;
                delete e;
                e = ne;
            }
        }
    }
};

int main() {
    return 0;
}
```

14. Splay

```
#include<cstdio>
#include<iostream>
#include<algorithm>
using namespace std;
const int MAX_N = 50000 + 10;
const int INF = ~0U >> 1;
struct Node {
    Node*ch[2], *p;
    int size, val, mx;
    int add;
    bool rev;
    Node() {
        size = 0;
        val = mx = -INF;
        add = 0;
    }
    bool d() {
```

```
		return this == p->ch[1];
	}
	void setc(Node*c, int d) {
		ch[d] = c;
		c->p = this;
	}
	void addIt(int ad) {
		add += ad;
		mx += ad;
		val += ad;
	}
	void revIt() {
		rev ^= 1;
	}
	void relax();
	void upd() {
		size = ch[0]->size + ch[1]->size + 1;
		mx = max(val, max(ch[0]->mx, ch[1]->mx));
	}
} Tnull, *null = &Tnull;
Node mem[MAX_N], *C = mem;

void Node::relax() {
	if (add != 0) {
		for (int i = 0; i < 2; ++i) {
			if (ch[i] != null)
				ch[i]->addIt(add);
		}
		add = 0;
	}
	if (rev) {
		swap(ch[0], ch[1]);
		for (int i = 0; i < 2; ++i) {
			if (ch[i] != null)
				ch[i]->revIt();
		}
		rev = 0;
	}
}

Node*make(int v) {
	C->ch[0] = C->ch[1] = null;
	C->size = 1;
	C->val = v;
	C->mx = v;
	C->add = 0;
	C->rev = 0;
	return C++;
}

Node*build(int l, int r) {
	if (l >= r)
		return null;
	int m = (l + r) >> 1;
	Node*t = make(0);
	t->setc(build(l, m), 0);
	t->setc(build(m + 1, r), 1);
	t->upd();
```

```
	return t;
}

Node*root;

Node*rot(Node*t) {
	Node*p = t->p;
	p->relax();
	t->relax();
	int d = t->d();
	p->p->setc(t, p->d());
	p->setc(t->ch[!d], d);
	t->setc(p, !d);
	p->upd();
	if (p == root)
		root = t;
}

void splay(Node*t, Node*f = null) {
	while (t->p != f) {
		if (t->p->p == f)
			rot(t);
		else
			t->d() == t->p->d() ? (rot(t->p), rot(t)) : (rot(t), rot(t));
	}
	t->upd();
}

Node* select(int k) {
	for (Node*t = root;;) {
		t->relax();
		int c = t->ch[0]->size;
		if (k == c)
			return t;
		if (k > c)
			k -= c + 1, t = t->ch[1];
		else
			t = t->ch[0];
	}
}

Node*&get(int l, int r) { //[l,r)
	Node*L = select(l - 1);
	Node*R = select(r);
	splay(L);
	splay(R, L);
	return R->ch[0];
}

int n, m;

int main() {
	cin >> n >> m;
	root = build(0, n + 2);
	root->p = null;
	for (int i = 0; i < m; ++i) {
		int k, l, r, v;
		scanf("%d%d%d", &k, &l, &r);
```

```
            Node*&t = get(l, r + 1);
            if (k == 1) {
                scanf("%d", &v);
                t->addIt(v);
                splay(t);
            } else if (k == 2) {
                t->revIt();
                splay(t);
            } else {
                printf("%d\n", t->mx);
            }
        }
}
```

15. TWOSAT.java

```
class TWOSAT {
    int n;
    V[] vs;

    class V extends ArrayList<V> {
        boolean visit = false;
        List<V> rs = new ArrayList<V>();
        int comp = -1;

        void addEdge(V v) {
            add(v);
            v.rs.add(this);
        }
    }

    TWOSAT(int n) {
        this.n = n;
        vs = new V[n * 2];
        for (int i = 0; i < vs.length; i++) {
            vs[i] = new V();
        }
    }

    void add(int a, int ai, int b, int bi) {//they contradict
        vs[a * 2 + ai].addEdge(vs[b * 2 + 1 - bi]);

        vs[b * 2 + bi].addEdge(vs[a * 2 + 1 - ai]);
    }

    V[] us;
    int cur;

    void dfs(V u) {
        u.visit = true;
        for (V v : u) {
            if (!v.visit)
                dfs(v);
        }
        us[--cur] = u;
    }

    void dfsrev(V u) {
        u.comp = cur;
```

```
            for (V r : u.rs) {
                if (r.comp == -1)
                    dfsrev(r);
            }
        }

        int[] solve() {//null if no solution exists
            //scc
            us = new V[vs.length];
            cur = vs.length;
            for (V v : vs) {
                if (!v.visit)
                    dfs(v);
            }
            cur = 0;
            for (V u : us) {
                if (u.comp == -1) {
                    dfsrev(u);
                    ++cur;
                }
            }

            int[] ret = new int[n];

            for (int i = 0; i < n; i++) {
                V a = vs[i * 2], b = vs[i * 2 + 1];
                if (a.comp == b.comp) {
                    return null;
                }
                ret[i] = a.comp > b.comp ? 0 : 1;
            }

            return ret;
        }
    }
```