AC-LIB

dc

Last build at 11/21/2014

Contents

Data-Structure

Binary Indexed Tree

```
struct BIT {
    LL C[N];
    int n;
    LL sum (int x) {
        LL r = 0;
        for (; x; x -= x & -x) r += C[x];
        return r;
    }
    LL sum (int l, int r) {
        if (l > r) return 0;
        return sum(r) - sum(l - 1);
    }
    void modify (int x, LL d) {
        for (; x <= n; x += x & -x) C[x] += d;
    }
    void clear () {
        fill(C + 1, C + 1 + n, 0LL);
    }
}
```

ChairTree

```
/*
 * Persistent segment tree set
 * Example: Kth value in range supporting modification
 * */

#include <cstdio>
#include <map>
#include <vector>
#include <algorithm>
using namespace std;

const int N = 10050;

struct node {
    int s;
    node *c[2];
    node (int _s = 0) : s(_s) { c[0] = c[1] = 0; }
};

inline int size (node *X) { return (X?(X->s):0); }

node *T[N], *que[N];
int ft[N], maxval, rear, n;
map<int, int> compr;
vector<int> revcmp;

node* insTree (node *t, int l, int r, int p, int de)
{
    node *re = new node();
    if (t) *re = *t; re->s += de;
    if (l != r)
    {
        int m = (l + r) >> 1;
        if (p <= m) re->l = insTree(lc(re), l, m, p, de);
        else re->r = insTree(rc(re), m + 1, r, p, de);
    }
    return re;
}

int leftsize ()
{
    int res(0);
    for (int i = 0; i < rear; ++i)
        if (que[i] && que[i]->c[0]) res += que[i]->c[0]->s * ft[i];
    return res;
}
void turn (int p) {
    for (int i = 0; i < rear; ++i)
        if (que[i]) que[i] = que[i]->c[p];
}
void expand_pre (int p, int f) {
    for (; p; p -= p & -p)
        ft[rear] = f, que[rear] = T[p], rear++;
}
int query (int l, int r, int k)
{
    rear = 0;
    expand_pre(r, +1), expand_pre(l - 1, -1);
    int tl = 1, tr = maxval;
    while (tl != tr)
    {
        int tm = (tl + tr) >> 1, ts = leftsize();
        if (ts >= k) turn(0), tr = tm;
        else turn(1), tl = tm + 1, k -= ts;
    }
    return revcmp[tl];
}
int dt[N], tdt[N];
void modify (int p, int de)
{
    int pd = tdt[p]; tdt[p] = de;
    for (; p <= n; p += p & -p)
    {
        T[p] = insTree(T[p], 1, maxval, pd, -1);
        T[p] = insTree(T[p], 1, maxval, de, +1);
    }
}

struct event { int ty, a, b, c; } ev[N];

int main ()
{
#ifdef LOCAL
    freopen("in", "r", stdin);
#endif
    int m;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)
    {
        scanf("%d", dt + i);
        compr[dt[i]] = 1;
    }
    for (int i = 1; i <= m; ++i)
    {
        char op[20];
        scanf("%s", op);
        if (op[0] == 'Q')
        {
            scanf("%d%d%d", &ev[i].a, &ev[i].b, &ev[i].c);
            ev[i].ty = 0;
        }
        else
        {
            scanf("%d%d", &ev[i].a, &ev[i].b);
            ev[i].ty = 1; compr[ev[i].b] = 1;
        }
    }
    maxval = 0;
    revcmp.resize(1 + compr.size());
    for (__typeof(compr.begin()) it = compr.begin(); it != compr.end(); ++it)
        revcmp[it->second = ++maxval] = it->first;
    for (int i = 1; i <= n; ++i)
        modify(i, compr[dt[i]]);
    for (int i = 1; i <= m; ++i)
    {
        if (ev[i].ty == 0)
            printf("%d\n", query(ev[i].a, ev[i].b, ev[i].c));
        else
            modify(ev[i].a, compr[ev[i].b]);
    }
    return 0;
}
```

Dynamic Convex Hull

```
/*
 * Without deletion
 * include: <set>, <algorithm>
 * */

typedef long double LD;
typedef long long LL;

const LL inf = ~0ULL >> 2;
const int N = 200050;

inline LL sqr (LL x) { return x * x; }

const int POINT = 0, QUERY = 1;

struct point { LL x, y; };
inline LL operator^ (point a, point b) { return a.x * b.x + a.y * b.y; }
inline LD operator* (point a, point b) {
    return (LD)a.x * b.y - (LD)b.x * a.y;// prevent overflow
}
inline point operator- (point a, point b) {
    return (point){a.x - b.x, a.y - b.y};
}
struct snode { point dt; const snode *n; int ty; };

bool operator< (snode a, snode b)
{
    if (a.ty == POINT && b.ty == POINT)
        return a.dt.x < b.dt.x;
    bool rev = false, res;
    if (b.ty == QUERY) swap(a, b), rev = true;
    // return true if the next one is worse.
    // according to the defination of upper_bound, we should stop in that case.
    if (! b.n) res = true;
    else
    {
        LL va = a.dt ^ b.dt, vb = a.dt ^ b.n->dt;
        res = bool(va > vb);
    }
    return res ^ rev;
}
template <class T> T prev (T x) { return --x; }
template <class T> T succ (T x) { return ++x; }

/*
 * Maintain the lower convex envelope of the points inserted.
 * Query (point p) : return max(p.x * i.x + p.y * i.y : i in S)
 * Note that p.y is expected to be negative.
 * To query the minimum, in the < operator of snode change
 * > res = bool(va > vb);
 * to
 * > res = bool(va < vb);
 * ; To get the upper convex envelope, negate the y coordinate of all points.
 */

struct point_set
{
    typedef set<snode>::iterator iter;
    set<snode> S;
```

```
void printSet ()
{
    for (iter it = S.begin(); it != S.end(); ++it)
        fprintf(stderr, "%llx (%lld %lld) %llx %d\n",
                reinterpret_cast<LL>(&*it),
                it->dt.x, it->dt.y, reinterpret_cast<LL>(it->n), it->ty);
}
void clear ()
{
    S.clear();
}
void insert (point x)
{
    snode sn = (snode){x, 0x0, POINT};
    iter it = S.find(sn);
    if (it != S.end())
    {
        if ((x.y - it->dt.y) >= 0) return;
        S.erase(it);
    }
    iter isuc = S.upper_bound(sn), ipre;
    if (isuc != S.begin() && isuc != S.end())
    {
        ipre = prev(isuc);
        if (((x - ipre->dt) * (isuc->dt - ipre->dt)) <= 0) return;
    }
    if (isuc != S.end()) // the right pointt
    {
        for (iter cr = isuc, nr = succ(cr); nr != S.end();
                cr = nr, nr = succ(nr))
            if (((nr->dt - x) * (cr->dt - x)) < 0) break;
            else S.erase(cr);
    }
    isuc = S.upper_bound(sn);
    if (isuc != S.begin()) // the left pointt
    {
        ipre = prev(isuc);
        for (iter cr = ipre, nr = prev(cr); cr != S.begin();
                cr = nr, nr = (nr == S.begin() ? nr : prev(nr)))
            if (((nr->dt - x) * (cr->dt - x)) > 0) break;
            else S.erase(cr);
    }
```

```
        // rebuild the bi-directional link list
        isuc = S.upper_bound(sn);
        sn.n = isuc == S.end() ? 0x0 : &*isuc;
        isuc = S.insert(sn).first;
        if (isuc != S.begin())
        {
            ipre = prev(isuc);
            snode *s = const_cast<snode*>(&*ipre);
            s->n = &*isuc;
        }
    }
    LL query (point qr)
    {
        iter it = S.upper_bound((snode){qr, 0x0, QUERY});
        return qr ^ it->dt;
    }
} s;
```

Persistent Treap

```
// Check geometry/planar-graph
```

Segment Tree

```
struct tree
{
    int l, r;
    tree *lc, *rc;
    info i;
    tree (int _l, int _r)
    {
        l = _l, r = _r;
        lc = rc = 0;
        if (l == r)
        {
            i = info(1, 1); // ...
            return;
        }
        int m = (l + r) >> 1;
        lc = new tree(l, m);
        rc = new tree(m + 1, r);
        i = lc->i + rc->i;
    }
    void modify (int p)
    {
        if (l == r)
        {
            i = info(0, 0); // ...
            return;
        }
        int m = (l + r) >> 1;
        if (p <= m) lc->modify(p);
        else rc->modify(p);
        i = lc->i + rc->i;
    }
    info query (int ql, int qr)
    {
        if (ql > qr) return info(0, 0);
        if (ql <= l && r <= qr) return i;
        int m = (l + r) >> 1;
        info res(0, 0); // ...
        if (ql <= m) res = lc->query(ql, qr);
        if (qr > m) res = res + rc->query(ql, qr);
        return res;
    }
} *root;
```

Segment Tree (Splittable)

```
struct node {
    static node NBf[LIM], *NCr, *nil;
    node *lc, *rc;
    int s, tag, maxt, mint;
    LL sumt, delt;
    static void init () {
        nil = NBf; NCr = NBf + 1;
        nil->lc = nil->rc = nil;
        nil->maxt = -inf, nil->mint = inf;
    }
    static node* get () {
        assert(NCr - NBf < LIM);
        NCr->lc = NCr->rc = nil;
        return NCr++;
    }
    static node* getLeaf (int l, int r, int p, int w) {
        node *re = get();
        if (l == r) {
            re->s = 1;
            re->tag = re->maxt = re->mint = re->sumt = w;
        }
        else {
            int m = (l + r) >> 1;
            if (p <= m) re->lc = getLeaf(l, m, p, w);
            else re->rc = getLeaf(m + 1, r, p, w);
            re->update();
        }
        return re;
    }
    void add (int d) {
        assert(this != nil);
        delt += d, sumt += (LL)s * d;
        maxt += d, mint += d;
    }
    void push () {
        assert(this != nil);
        if (delt) {
            if (lc != nil) lc->add(delt);
            if (rc != nil) rc->add(delt);
            delt = 0;
        }
    }
    void update () {
//      assert(!delt && this != nil);
        s = lc->s + rc->s;
        sumt = lc->sumt + rc->sumt;
        maxt = max(lc->maxt, rc->maxt);
        mint = min(lc->mint, rc->mint);
    }
    void print (int l, int r) {
        if (l == r) {
            assert(s);
            cerr << l << ' ';
        }
        else {
            int m = (l + r) >> 1;
            if (lc != nil) lc->print(l, m);
            if (rc != nil) rc->print(m + 1, r);
        }
    }
    void split (int l, int r, int v, node *&rl, node *&rr) {
        if (this == nil) {
            rl = rr = nil;
            return;
        }
        if (v >= r) {
            rl = this, rr = nil;
            return;
        }
        if (v < l) {
            rl = nil, rr = this;
            return;
        }
        push();
        int m = (l + r) >> 1;
        rl = node::get(), rr = node::get();
        lc->split(l, m, v, rl->lc, rr->lc);
        rc->split(m + 1, r, v, rl->rc, rr->rc);
        if (rl->lc == nil && rl->rc == nil) rl = nil; else rl->update();
        if (rr->lc == nil && rr->rc == nil) rr = nil; else rr->update();
        assert(rl->lc == nil && rl->rc == nil || rl->s);
        assert(rr->lc == nil && rr->rc == nil || rr->s);
    }
    static node* merge (int l, int r, node *a, node *b) {
        if (a == nil) return b;
        if (b == nil) return a;
        a->push(); b->push();
        node *re = node::get();
        int m = (l + r) >> 1;
        re->lc = merge(l, m, a->lc, b->lc);
        re->rc = merge(m + 1, r, a->rc, b->rc);
        re->update();
        assert(re->s);
        return re;
    }
} node::NBf[LIM], *node::NCr, *node::nil;
```

Skew Heap (Mergeable Heap)

```
struct skewHeap {
    skewHeap *lc, *rc;
    int val, dist;
    skewHeap ();
    void extract_min ();
} SBuff[MXN], *nil = SBuff, *St = nil + 1, *top[MXN];

skewHeap::skewHeap () : lc(nil), rc(nil), val(0), dist(0) {}

skewHeap* merge (skewHeap *x, skewHeap *y)
{
    if (x == nil) return y;
    if (y == nil) return x;
    if (x->val > y->val) swap(x, y);
    x->rc = merge(x->rc, y);
    /*uncomment the two lines below to get a leftish-tree*/
    //if (x->lc->dist > x->rc->dist)
        swap(x->lc, x->rc);
    //x->dist = x->rc == nil ? 0 : x->rc->dist + 1;
    return x;
}
```

Splay Tree

```
// Rev 2
struct node {
    static node *nil;
    node *f, *c[2];
    int s, dt;
    lint hval;
    node () : f(nil), s(0), hval(0) { c[0] = c[1] = nil; }
    int w () { return this == f->c[1]; }
    void setc (node *t, int k) { c[k] = t; t->f = this; }
    void update ()
    {
        hval = c[0]->hval * M + dt;
        hval = hval * power[c[1]->s] + c[1]->hval;
        s = c[0]->s + c[1]->s + 1;
    }
} T[N], *node::nil = T, *cur = T + 1, *root;

inline node* newNode (int dt = 0)
{
    cur->dt = dt, cur->s = 1;
    return cur++;
}

inline void zig (node *x)
{
    int w = x->w(); node *y = x->f;
    y->setc(x->c[!w], w);
    if (y->f != node::nil) y->f->setc(x, y->w());
    else x->f = y->f;
    x->setc(y, !w);
    y->update();
    if (y == root) root = x;
}

void splay (node *x, node *y)
{
    for (; x->f != y; zig(x)) if (x->f->f != y)
        if (x->w() == x->f->w()) zig(x->f); else zig(x);
    x->update();
}
node* select (node *x, int k) // 0-based
{
    for (; ; )
    {
        int r = x->c[0]->s;
        if (k == r) return x;
        x = x->c[r < k];
        if (r < k) k -= r + 1;
    }
}

node* expose (int l, int r) // (l, r), 1-based
{
    splay(select(root, l), node::nil);
    splay(select(root, r), root);
    return root->c[1]->c[0];
}
```

```
lint getHash (int l, int r) { return expose(l - 1, r + 1)->hval; }

void insert (int rk, int dt) {
    node *t = newNode(dt);
    expose(rk - 1, rk);
    root->c[1]->setc(t, 0);
    splay(t, node::nil);
}
void replace (int p, int dt) {
    node *x = expose(p - 1, p + 1);
    x->dt = dt; splay(x, node::nil);
}

int query (int a, int b)
{
    int res(0), n = root->s - 2;
    if (getHash(a, a) != getHash(b, b)) return 0;
    for (int i = 1 << 18; i; i >>= 1)
        if (a + res + i <= n && b + res + i <= n &&
            getHash(a, a + res + i) == getHash(b, b + res + i))
            res += i;
    return ++ res;
}

struct qNode { int l, r; node *t; } que[N];
void buildTree (char *s, int len)
{
    int l(0), r(0), m;
    que[++ r] = (qNode){0, len, root = newNode()};
    while (l < r)
    {
        qNode cr = que[++ l];
        cr.t->dt = s[m = (cr.l + cr.r) >> 1];
        if (cr.l < m)
        {
            cr.t->setc(newNode(), 0);
            que[++ r] = (qNode){cr.l, m - 1, cr.t->c[0]};
        }
        if (m < cr.r)
        {
            cr.t->setc(newNode(), 1);
            que[++ r] = (qNode){m + 1, cr.r, cr.t->c[1]};
        }
    }
    for (int i = r; i > 0; --i) que[i].t->update();
}

void printTree (node *x, int de)
{
    for (int i = 1; i <= de; ++i) fputc(' ', stderr);
    fprintf(stderr, "Node %d %d %I64u %d %d %d\n", x-T, x->dt, x->hval,
        x->s, x->c[0]-T, x->c[1]-T);
    if (x->c[0] != node::nil) printTree(x->c[0], de + 1);
    if (x->c[1] != node::nil) printTree(x->c[1], de + 1);
}
```

Geometry

Half-plane Intersection

```
/*
 * Last Edit: May 2011
 * I don't know why it works either.
 * */

#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;

namespace HPI {

#define MXN 20050

const double eps = 1e-7;

inline int sgn(double a) {
    if (fabs(a) < eps) return 0;
    return (a < 0)? -1 : 1;
}

#define ZERO(X) (sgn(X)==0)
```

```
#define NEGA(X) (sgn(X)==-1)
#define POSI(X) (sgn(X)==1)

struct point {
    double x, y;
    point(): x(0), y(0) {}
    point(double a, double b): x(a), y(b) {}
    point operator-(const point &b) const {
        return point(x - b.x, y - b.y);
    }
    double operator*(const point &b) const {
        return x * b.y - y * b.x;
    }
    bool operator==(const point &b) const {
        return (ZERO(y - b.y) && ZERO(x - b.x));
    }
} P[MXN];

struct hvec {
    point s, t;
    double ang;
    point operator*(const hvec &b) const {
        double s1 = (b.t - s) * (b.s - s),
            s2 = (b.s - t) * (b.t - t);
        return point((s.x * s2 + t.x * s1) / (s2 + s1),
            (s.y * s2 + t.y * s1) / (s2 + s1));
    }
    bool operator<(const hvec &b) const {
        if (ZERO(b.ang - ang))
            return sgn((b.t - b.s) * (t - b.s)) >= 0;
        else return ang < b.ang;
    }
} H[MXN], Q[MXN];
int N;

inline bool isout(const hvec &h, const point &p) {
    return POSI((p - h.s) * (h.t - h.s));
}
inline bool parralel(const hvec &a, const hvec &b) {
    return ZERO((a.t - a.s) * (b.t - b.s));
}
```

```
point inte(hvec a, hvec b) { return a * b; }
bool lesstahn(hvec a, hvec b) {return a < b; }

inline int hvec_int() {
    int M = 0, C = 1, l = 0, r = 1;
    sort(H + 1, H + N + 1);
    for (int i = 2; i <= N; ++i)
        if (! ZERO(H[i].ang - H[i - 1].ang))
            H[++C] = H[i];
    Q[0] = H[1], Q[1] = H[2];
    for (int i = 3; i <= C; ++i) {
        while (l < r && isout(H[i], Q[r] * Q[r - 1])) --r;
        while (l < r && isout(H[i], Q[l] * Q[l + 1])) ++l;
        Q[++r] = H[i];
    }
    while (l < r && isout(Q[l], Q[r] * Q[r - 1])) --r;
    while (l < r && isout(Q[r], Q[l] * Q[l + 1])) ++l;
    if (r <= l + 1) return 0;
    for (int i = l; i < r; ++i) P[++M] = Q[i] * Q[i + 1];
    P[++M] = Q[l] * Q[r];
    M = unique(P + 1, P + M + 1) - P - 1;
    return M;
}

double area(int n) {
    if (n < 3) return 0;
    double s = 0;
    for (int i = 1; i <= n; ++i)
        s += P[i] * P[i % n + 1];
    return fabs(s / 2);
}

void add_hvec(point s, point t) {
    H[++N].s = s, H[N].t = t;
    point c = t - s;
    H[N].ang = atan2(c.y, c.x);
}

int main() {
    int m;
    N = 0;
    add_hvec(point(0, 0), point(10000, 0)),
    add_hvec(point(10000, 0), point(10000, 10000)),
    add_hvec(point(10000, 10000), point(0, 10000)),
    add_hvec(point(0, 10000), point(0, 0));
    scanf("%d", &m);
    for (int i = 1; i <= m; ++i) {
        double x1, y1, x2, y2;
        scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
        add_hvec(point(x1, y1), point(x2, y2));
    }
    m = hvec_int();
    for (int i = 1; i <= m; ++i)
        printf("%.3f %.3f\n", P[i].x, P[i].y);
    printf("%.1lf\n", area(m));
    return 0;
}

}

int main ()
{
#ifdef LOCAL
    freopen("in", "r", stdin);
#endif
    HPI::main();
}
```

Header for Geometry

```
/*
 * Basic 2D / 3D geometry & FP functions
 * include: <cmath>, <algorithm>, <numeric>, <iostream>[optional]
 * 5-30-2013 3D point class
 * 3-18-2013 Covex Hull, point::rotate, crdcomp
 * Jan-      Point class, Half Plane Intersection(Bruteforce)
 * */

#define Px first
#define Py second

// {{{ FP functions & constants ..
typedef long double LD;
```

```
const LD eps = 1e-10, infcmp = 1e200, infcrd = 1e9, pi = acos(-1);
// avoid possible errors, infcrd * eps < 1
template <class T> inline T abs (T x) { return x > 0 ? x : -x; }
template <class T> inline int sgn (T x) {
    return abs(x) < eps ? 0 : x > 0 ? 1 : -1;
}
template <class T> inline T sqr (T x) { return x * x; }
template <class T> inline T normalize1 (T x) { // [0, 2pi)
    while (sgn(x - pi * 2) >= 0) x -= 2 * pi;
    while (sgn(x) < 0) x += 2 * pi;
    return x;
}
template <class T> inline T normalize2 (T x) { // (-pi,pi]
    while (sgn(-pi - x) >= 0) x += 2 * pi;
    while (sgn(x - pi) > 0) x -= 2 * pi;
    return x;
}
// }}}

// {{{ 2D Point ..
struct point
{
    LD x, y;
    point (LD xx = 0, LD yy = 0) { x = xx, y = yy; }
    LD len () const { return hypot(x, y); }
    point oppo () const { return point(y, -x); }
    point rotate (LD t) const { // counterclockwise
        LD cost = cos(t), sint = sin(t);
        return point(x * cost - y * sint, x * sint + y * cost);
    }
};
inline int crdcomp (point s, point e) {
    int sx = sgn(s.x - e.x); return sx ? sx == -1 : sgn(s.y - e.y) == -1;
}
inline point operator+ (const point &a, const point &b) {
    return point(a.x + b.x, a.y + b.y);
}
inline point operator- (const point &a, const point &b) {
    return point(a.x - b.x, a.y - b.y);
}
inline LD operator* (const point &a, const point &b) { // det
    return a.x * b.y - b.x * a.y;
```

```
}
inline LD operator^ (const point &a, const point &b) { // dot
    return a.x * b.x + a.y * b.y;
}
inline point operator* (LD l, const point &a) {
    return point(a.x * l, a.y * l);
}
inline point operator* (const point &a, LD l) {
    return point(a.x * l, a.y * l);
}
inline point operator/ (const point &a, LD l) {
    return (LD(1) / l) * a;
}
#ifdef _GLIBCXX_IOSTREAM
inline istream& operator>> (istream &in, point &a) {
    return in >> a.x >> a.y;
}
inline ostream& operator<< (ostream &out, point &a) {
    return out << '(' << a.x << ',' << a.y << ')';
}
#endif
inline bool operator== (const point &a, const point &b) {
    return sgn(a.x - b.x) == 0 && sgn(a.y - b.y) == 0;
}

inline bool onSeg (point a, point s, point e) {
    return sgn((e - a).len() + (a - s).len() - (e - s).len()) == 0;
}

inline pair<int, point> getConj (point s1, point e1, point s2, point e2)
{
    LD a = (s2 - s1) * (e1 - s1), b = (e2 - s1) * (e1 - s1);
    if (sgn(a - b) == 0)
        return make_pair(0, point(0.0, 0.0));
    point r = (a * e2 - b * s2) / (a - b);
    // intersection of the two lines.
    if (!onSeg(r, s1, e1) || !onSeg(r, s2, e2)) return make_pair(-1, r);
    return make_pair(1, r); // intersection of the two segments.
}
inline int colinear (point s1, point e1, point s2, point e2)
{
    LD a = (s2 - s1) * (e1 - s1), b = (e2 - s1) * (e1 - s1);
    if (sgn(a - b) != 0 || sgn(a) != 0) return 0;
    if (onSeg(s2, s1, e1) && onSeg(e2, s1, e1)) return 1;
    if (onSeg(s2, s1, e1) || onSeg(e2, s1, e1)) return 2;
    return 3;
}
// }}}

// {{{ 2D Polygon Functions ..
inline int getCH (point P[], int n, point CH[]) {
    // 1-based, colinear points removed
    sort(P + 1, P + 1 + n, crdcomp);
    int m = 0; CH[m = 1] = P[1];
    for (int i = 1; i <= n; ++i) {
        while (m > 1 && sgn((P[i] - CH[m]) * (CH[m] - CH[m - 1])) <= 0) --m;
        CH[++m] = P[i];
    }
    int cm = m;
    for (int i = n - 1; i >= 1; --i) {
        while (m > cm && sgn((P[i] - CH[m]) * (CH[m] - CH[m - 1])) <= 0) --m;
        CH[++m] = P[i];
    }
    return m - 1;
}
inline LD calcArea (point *s, point *e)
// [s, e), sorted, position e accessable
{
    if (e - s < 3) return 0;
    *e = *s; LD res(0);
    for (point *i = s; i < e; ++i) res += i[0] * i[1];
    return abs(res) / 2.0;
}

struct hplane {
    point s, e;
    // feasible region = {P|(P-s)*(e-s)>=0}
    // (sP lies on the right hand side of se)
    hplane (point ss, point ee) : s(ss), e(ee) {}
    bool inside (point p) { return sgn((p - s) * (e - s)) >= 0; }
};

struct HPI_t
{
    static const int N = 1024;
    point P1[N], P2[N];
    int n, m, inside[N];
    point* begin() { return P1 + 1; }
    point* end() { return P1 + n + 1; }
    void clear () {
        n = 4; m = 0;
        P1[1] = point(infcrd, infcrd);
        P1[2] = point(-infcrd, infcrd);
        P1[3] = point(-infcrd, -infcrd);
        P1[4] = point(infcrd, -infcrd);
    }
    bool extend (hplane now)
    {
        P1[n + 1] = P1[1]; m = 0;
        for (int i = 1; i <= n + 1; ++i) inside[i] = now.inside(P1[i]);
        for (int i = 1; i <= n; ++i)
        {
            if (inside[i]) P2[++m] = P1[i];
            if (inside[i] ^ inside[i + 1])
                P2[++m] = getConj(now.s, now.e, P1[i], P1[i + 1]).Py;
        }
        copy(P2 + 1, P2 + 1 + (n = m), P1 + 1);
        return n;
    }
};

// }}}

// {{{ 3D Point ..
struct point3 {
    LD x, y, z;
    point3 () { x = y = z = 0; }
    point3 (LD _x, LD _y, LD _z) { x = _x, y = _y, z = _z; }
    LD len () const { return sqrt(len2()); }
    LD len2 () const { return sqr(x) + sqr(y) + sqr(z); }
    point3 normalize () const {
        LD l = len(); return point3(x / l, y / l, z / l);
    }
};

inline bool operator< (const point3 &a, const point3 &b)
```

```
{
    if (sgn(a.x - b.x)) return sgn(a.x - b.x) < 0;
    if (sgn(a.y - b.y)) return sgn(a.y - b.y) < 0;
    return sgn(a.z - b.z) < 0;
}
inline bool operator== (const point3 &a, const point3 &b) {
    return !sgn(a.x - b.x) && !sgn(a.y - b.y) && !sgn(a.z - b.z);
}

#ifdef _GLIBCXX_IOSTREAM
inline istream& operator>> (istream &in, point3 &a) {
    return in >> a.x >> a.y >> a.z;
}
inline ostream& operator<< (ostream &out, const point3 &a) {
    return out << "(" << a.x << ", " << a.y << ", " << a.z << ")";
}
#endif
inline LD operator^ (const point3 &a, const point3 &b) { // dot
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
inline point3 operator* (const point3 &a, const point3 &b) { // cross
    return point3(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z,
            a.x * b.y - a.y * b.x);
}
inline point3 operator+ (const point3 &a, const point3 &b) {
    return point3(a.x + b.x, a.y + b.y, a.z + b.z);
}
inline point3 operator- (const point3 &a, const point3 &b) {
    return point3(a.x - b.x, a.y - b.y, a.z - b.z);
}
inline point3 operator* (const point3 &a, const LD b) {
    return point3(a.x * b, a.y * b, a.z * b);
}
inline point3 operator* (const LD b, const point3 &a) {
    return point3(a.x * b, a.y * b, a.z * b);
}
inline point3 operator/ (const point3 &a, const LD b) {
    return a * (1.0 / b);
}

inline LD angle (const point3 &a, const point3 &b) { // [0, pi]
    return acos((a ^ b) / (a.len() * b.len()));
```

```
}
inline pair<int, point3> intersection (
        const point3 &s1, const point3 &e1,
        const point3 &s2, const point3 &e2)
{
    if (sgn(((e1 - s1) * (e2 - s2)).len()) == 0) // parallel
        return make_pair(0, point3());

    if (sgn((e1 - s1) ^ ((e2 - s1) * (s2 - s1)))) // non-coplanar
        return make_pair(-2, point3());

    point3 pa = (s2 - s1) * (e1 - s1), pb = (e2 - s1) * (e1 - s1);
    LD a = pa.len(), b = pb.len();
    if (sgn(pa ^ pb) < 0) b = -b;
    point3 r = (a * e2 - b * s2) / (a - b);
//  assert(sgn(((s1 - r) * (e1 - r)).len()) == 0);
//  assert(sgn(((s2 - r) * (e2 - r)).len()) == 0);
    return make_pair(1, r); // intersect
}

struct plane {
    point3 A, B, C, L;
    LD lenB, lenC;
    // three points A, B, C or {P|(P-A)*L=0}
    plane () {}
    plane (const point3 &a, const point3 &b, const point3 &c) {
        A = a; B = b; C = c;
        C = (B - A) * (C - A) + A;
        C = (B - A) * (C - A) * ((B - A).len2() / (C - A).len2()) + A;
//      exception may raised by fp error
//      assert(sgn((B - A) ^ (C - A)) == 0);
//      assert(sgn(((c - a) * (b - a)) ^ (C - A)) == 0);
        L = ((C - A) * (B - A)).normalize();
        lenB = (B - A).len(), lenC = (C - A).len();
    }
    static plane normalPlane (const point3 &s, const point3 &e) {
        // possible fp error
        point3 x(rand() % 100, rand() % 100, rand() % 100),
               y(rand() % 100, rand() % 100, rand() % 100);
        return plane(s, s + (e - s) * x * 0.01, s + (e - s) * y * 0.01);
    }
    // may return negative value
    LD dis (const point3 &p) const { return (p - A) ^ L; }
    point projection (const point3 &p) const { // relative coordinate
        return point((p - A) ^ (B - A) / lenB, (p - A) ^ (C - A) / lenC);
    }
    pair<int, point3> intersection (const point3 &s, const point3 &e) const {
    // line-plane intersection. 0 - parallel, 1 - intersect
        LD a = dis(s), b = dis(e);
        if (sgn(a - b) == 0)
            return make_pair(0, point3());
        return make_pair(1, (s * -b + e * a) / (a - b));
    }
    LD lAngle (const point3 &s, const point3 &e) const {
        // line-plane angle, [0, pi/2]
        pair<int, point3> ret = intersection(s, e);
        if (ret.Px == 0)
            return 0;
        LD t = angle(e - ret.Py, L);
        return pi / 2 - min(t, pi - t);
    }
};

// }}}
```

Planar Graph & Point Location

```
/*
 * Construction of dual graph for planar graph and point location
 * Work for connected graph only
 * Last Edit : Feb 2013
 * */

typedef long long LL;

struct point {
    int x, y;
    point (int _x = 0, int _y = 0) : x(_x), y(_y) {}
    point operator+ (const point &b) const { return point(x + b.x, y + b.y); }
    point operator- (const point &b) const { return point(x - b.x, y - b.y); }
    LL operator* (const point &b) const { return (LL)x * b.y - (LL)y * b.x; }
    LL operator^ (const point &b) const { return (LL)x * b.x + (LL)y * b.y; }
    bool operator== (const point &b) const { return x == b.x && y == b.y; }
    double abs () { return hypot(x, y); }
    double angle () { return atan2(y, x); }
};
inline int sgn (LL x) { return x > 0 ? 1 : x == 0 ? 0 : -1; }


namespace TREAP {
    template <class valtype>
    struct treap {
        static int tot_cnt;
        treap *lc, *rc;
        int s, w;
        const valtype &v;

        static int gsize (treap *x) { return x ? x->s : 0; }
        void update () { s = treap::gsize(lc) + treap::gsize(rc) + 1; }

        treap (const valtype &nv, int nw, treap *l, treap *r)
            : v(nv), w(nw), lc(l), rc(r) { update(); ++tot_cnt; }

        static treap* newLeaf (valtype &v) { return new treap(v, rand(), 0, 0); }
        static treap *splitL (treap *x, valtype &v) {
            if (!x) return x;
            if (x->v >= v) return splitL(x->lc, v);
            return new treap(x->v, x->w, x->lc, splitL(x->rc, v));
        }
        static treap *splitR (treap *x, valtype &v) {
            if (!x) return x;
            if ((x->v <= v) ^ (x->v < v || x->v == v))
                throw;
            if (x->v <= v) return splitR(x->rc, v);
            return new treap(x->v, x->w, splitR(x->lc, v), x->rc);
        }
        static treap *merge (treap *l, treap *r) { // max(l) < min(r)
            if (!l || !r) return l ? l : r;
            if (l->w < r->w)
                return new treap(l->v, l->w, l->lc, merge(l->rc, r));
            else
                return new treap(r->v, r->w, merge(l, r->lc), r->rc);
        }
        static treap *insert (treap *x, valtype &v) {
            treap *a = splitL(x, v), *b = splitR(x, v);
            return merge(a, merge(newLeaf(v), b));
            return merge(splitL(x, v), merge(newLeaf(v), splitR(x, v)));
        }
        static treap *erase (treap *x, valtype &v) {
            return merge(splitL(x, v), splitR(x, v));
        }
        const treap* lower_bound (valtype &x) const {
            if (v == x) return this;
            if (x < v) {
                const treap *res;
                if (lc) res = lc->lower_bound(x); else res = this;
                if (!res) res = this;
                return res;
            }
            else {
                if (rc) return rc->lower_bound(x);
                else return 0x0;
            }
        }
        void printTree (int dp, int ty) {
            for (int i = 0; i < dp; ++i) putchar(' ');
            printf("Ty %d %d %d ", ty, s, w);
            v.output(); puts("");
            if (lc) { lc->printTree(dp + 1, 0);  }
            if (rc) { rc->printTree(dp + 1, 1);  }
        }
    };
    template <class T> int treap<T>::tot_cnt = 0;
}

// Dual graph & Point locating. Intereact with STDIO
namespace GRAPH {
    const int N = 300050, inf = ~0u >> 2;
    point P[N];
    struct edge { int s, t, w, id; double ang; edge *d, *n, *p; } edges[N];
    void add_edge (int s, int t, int w) {
        static edge* ec = edges + 1;
        *ec = (edge){s, t, w, 0, (P[t] - P[s]).angle(), ec + 1, 0x0, 0x0}; ec++;
        *ec = (edge){t, s, w, 0, (P[s] - P[t]).angle(), ec - 1, 0x0, 0x0}; ec++;
    }

    int n, m;
    namespace DUAL { // dual graph construction
        vector<edge*> e[N];
        int outer, cid; // outer region; number of regions
        bool e_comp (const edge *a, const edge *b) { return a->ang < b->ang; }
        void init () {
            for (int i = 1, ie = m * 2; i <= ie; ++i)
                e[edges[i].s].push_back(edges + i);
            for (int i = 1; i <= n; ++i) {
                sort(e[i].begin(), e[i].end(), e_comp);
                for (__typeof(e[i].begin())it = e[i].begin(); it != e[i].end(); ++it) {
                    if (it == e[i].begin()) (*it)->p = *e[i].rbegin(); else (*it)->p = it[-1];
                    if (it + 1 == e[i].end()) (*it)->n = *e[i].begin(); else (*it)->n = it[1];
                }
            }
            for (int i = 1, ie = m * 2; i <= ie; ++i) {
                if (edges[i].id) continue;
                int nid = ++cid;
                LL area = 0;
                for (edge *e = edges + i; !e->id; e = e->d->p) {
//                  printf("%d->%d\t", e->s, e->t);
                    area += P[e->t] * P[e->s];
                    e->id = nid;
                }
//              puts("\n----------------------------");
                if (area > 0) {
                    if (outer > 0) throw;
                    outer = nid;
                }
            }
            for (int i = 1, ie = m * 2; i <= ie; ++i) {
                edge *cur = edges + i;
                if (cur->id == outer || cur->d->id == outer)
                    cur->w = inf;
                if (cur->d > cur)
                    MST::add_edge(cur->id, cur->d->id, cur->w);
            }
        }
    }
    namespace LOCATOR { // Solve the point location problem. guaranteed that no point lies on borders.
        using namespace TREAP;
        struct tnode {
            point s, e; int id;
            void output () { printf(" (%d,%d)->(%d,%d) %d ", s.x, s.y, e.x, e.y, id); }
            tnode (point a,  point b, int i) : s(a), e(b), id(i) {}
            tnode () {}
            bool operator< (const tnode &b) const {
                if (s == e) return !(b < *this); else
                {
                    int f1 = sgn((b.s - s) * (e - s)), f2 = sgn((b.e - s) * (e - s));
                    if (f1 == 0 && f2 == 0) return false;
                    if (f1 >= 0 && f2 >= 0) return true;
                    if (f1 <= 0 && f2 <= 0) return false;
                    return !(b < *this);
                }
            }
            bool operator== (const tnode &b) const {
                return s == b.s && e == b.e;
            }
            bool operator<= (const tnode &b) const {
                return !(b < *this);
            }
            bool operator>= (const tnode &b) const {
                return !(*this < b);
            }
            bool operator> (const tnode &b) const {
                return b < *this;
            }
        };
        struct tnode_bufferpool {
            tnode bb[2000000];
            tnode& new_tnode (point a, point b, int c) {
                static int top = 0;
                bb[top] = tnode(a, b, c);
                return bb[top++];
            }
        } tpool;
        typedef treap<tnode> tree;
        map<int, tree*> tMap;

        struct event {
            tnode x; int ti, ty;
            bool operator< (const event &b) const {
                if (ti == b.ti) return ty < b.ty; else return ti < b.ti;
            }
```

```
            event (tnode b, int i, int y) : x(b), ti(i), ty(y) {}
        };
        vector<event> tbuff;

        void init () {
            tbuff.reserve(m * 2);
            for (int i = 1, ie = m * 2; i <= ie; ++i) {
                edge &c = edges[i];
                if (P[c.s].x >= P[c.t].x) continue;
                tnode &tcur = tpool.new_tnode(P[c.s], P[c.t], c.id);
                tbuff.push_back(event(tcur, P[c.t].x, 0)); // REMOVE
                tbuff.push_back(event(tcur, P[c.s].x, 1)); // INSERT
            }
            sort(tbuff.begin(), tbuff.end());
            tree *cur = 0x0;
            for (__typeof(tbuff.begin())it = tbuff.begin(); it != tbuff.end(); ++it) {
                if (it->ty == 0) // remove
                    cur = tree::erase(cur, it->x);
                else // insert
                    cur = tree::insert(cur, it->x);
//              if (cur) fprintf(stderr, "%d\n", cur->s);
                if (it + 1 == tbuff.end() || it[1].ti != it[0].ti) // save status
                    tMap[it->ti] = cur;
            }
            /*for (__typeof(tMap.begin())it = tMap.begin(); it != tMap.end(); ++it) {
                printf("X Coordinate = %d\t", it->first);
                it->second ? it->second->printTree(0, 0) : void();
                puts("---------------------------------------");
            }*/
        }
        int query (point x) {
            __typeof(tMap.begin()) it = tMap.upper_bound(x.x);
            if (it == tMap.begin()) return -1;
            --it;
            const tree *xcur = it->second;
            if (!xcur) return -1;
            const tree *ycur = xcur->lower_bound(tpool.new_tnode(x, x, 0));
            return ycur ? ycur->v.id : -1;
        }
    }

    void solve () {
        scanf("%d%d", &n, &m);
        for (int i = 1; i <= n; ++i) {
            scanf("%d%d", &P[i].x, &P[i].y);
            P[i].x *= 2;
            P[i].y *= 2;
        }
        for (int i = 1; i <= m; ++i) {
            int a, b, w;
            scanf("%d%d%d", &a, &b, &w);
            add_edge(a, b, w);
        }
        DUAL::init();
        LOCATOR::init();
        MST::solve(DUAL::cid, m);
        int q;
        scanf("%d", &q);
        for (int i = 0; i < q; ++i) {
            double x1, x2, y1, y2;
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            int a = LOCATOR::query(point(int(x1 * 2), int(y1 * 2))),
                b = LOCATOR::query(point(int(x2 * 2), int(y2 * 2)));
//          fprintf(stderr, "Point 1 %d Point 2 %d\n", a, b);
            if (a == -1 || a == DUAL::outer || b == -1 || b == DUAL::outer) {
                puts("-1");
                continue;
            }
            printf("%d\n", TREE::getMax(a, b));
        }
    }
}
```

kd-Tree

```
/*
 * check http://en.wikipedia.org/wiki/K-d_tree
 * Orthogonal range search : O(N^{(d-1)/d})
 * Nearest neighbor query : O(N^{(d-1)/d})
 * */

const int K = 3, N = 100000;

struct point {
    long long dt[K], now;
    long long get (int d) { return dt[d]; }
} P[N];
bool bynow (point a, point b) { return a.now < b.now; }

int lmax[N], rmin[N];

void build (int t, int l, int r, int d)
{
//build the tree node of P[l .. r] in demension d
    if (l > r) return;
    int m = l + r >> 1;
    for (int i = l; i <= r; i++)
    {
        P[i].now = P[i].get(d);
        for (int j = 0; j < dmax; ++j) // for range query
            maxd[t][j] = max(maxd[t][j], P[i].get(j)),
            mind[t][j] = min(mind[t][j], P[i].get(j));
    }
    sort(P + l, P + r + 1, bynow);
    lmax[t] = m != l ? P[m - 1].get(d) : -inf;
    rmin[t] = m != r ? P[m + 1].get(d) : inf;
    build(t * 2, l, m - 1, (d + 1) % K);
    build(t * 2 + 1, m + 1, r, (d + 1) % K);
}
void update (long long A)
{
    if (A == 0) return;
    if (A < ans) { ans = A; num = 1;}
    else if (A == ans) ++num;
}

// nearest neighbor query
void query (int t, int l, int r, point pt, int d)
{
    if (l > r) return;
    update(dis(pt, P[l + r >> 1]));
    if (l == r) return;
    int m = l + r >> 1;
    if (pt.get(d) <= lmax[t]) {
        query(t * 2, l, m - 1, pt, (d + 1) % K);
        if ((long long)(pt.get(d) - rmin[t]) * (pt.get(d) - rmin[t]) <= ans) // improvable
```

```
            query(t * 2 + 1, m + 1, r, pt, (d + 1) % K);
    }
    else {
        query(t * 2 + 1, m + 1, r, pt, (d + 1) % K);
        if ((long long)(pt.get(d) - lmax[t]) * (pt.get(d) - lmax[t]) <= ans)
            query(t * 2, l, m - 1, pt, (d + 1) % K);
    }
}

// sketch of range query
void rquery (int t, int l, int r, int QR)
{
    if (l > r) return;
    if (included(t, QR))
    {
        // do something
        return;
    }
    if (disjoint(t, QR))
    {
        return;
    }
    int m = l + r >> 1;
    if (included(P[m], QR))
    {
        // do something
    }
    rquery(t * 2, l, m - 1, QR);
    rquery(t * 2 + 1, m + 1, r, QR);
}
```

Graph

2-SAT

```
/*
 * Construct a feasible solution for 2-SAT problem using tarjan
 * Counting the solutions is hard to solve.
 * Last Edit : Feb 2014
 * */
#include "scc-tarjan.h"
```

```
namespace SAT {
    const int AND = 0, OR = 1, SEL = 0, UNS = 1;
    int tlist[VT], cc[VT], color[VT], tmp[VT];
    void tsort ()
    {
        static int deg[VT], cur = 0;
        static queue<int> que;
        for (int i = 1; i <= cn; ++i)
            for_(it, modi[i]) ++ deg[*it];
        for (int i = 1; i <= cn; ++i)
            if (!deg[i]) que.push(i);
        while (!que.empty())
        {
            int x = que.front(); que.pop(); tlist[++cur] = x;
            for_(it, modi[x]) if (!--deg[*it]) que.push(*it);
        }
    }
    inline int dual (int x) { return x + ((x & 1) ? 1 : -1); }
    inline int sel (int x) { return x * 2 - 1; }
    inline int uns (int x) { return x * 2; }
    inline void add (int x, int y, int ty)
    { // edge(x,y): select(x) imp. select(y)
        if (ty == AND) // x&&y==0
        {
            esat[sel(x)].pback(uns(y));
            esat[sel(y)].pback(uns(x));
        }
        else // x||y==1
        {
            esat[uns(x)].pback(sel(y));
            esat[uns(y)].pback(sel(x));
        }
    }
    bool solve (int *res)
    {
        cn = tarjan::solve(cc);
        for (int i = 1; i <= on; ++i)
        {
            if (cc[i] == cc[dual(i)]) return false;
            conf[cc[i]].pback(cc[dual(i)]);
        }
        tsort();
        fill(color + 1, color + cn + 1, -1);
        for (int i = cn; i > 0; --i)
        {
            int x = tlist[i];
            if (color[x] != -1) continue;
            color[x] = SEL;
            for_(it, modi[x]) if (color[*it] == UNS)
                { color[x] = UNS; break; }
            for_(it, conf[x]) color[*it] = color[x] ^ 1;
        }
        for (int i = 1; i <= on; ++i) tmp[i] = color[cc[i]];
        for (int i = 1; i <= in; ++i) res[i] = tmp[sel(i)] == SEL;
        return true;
    }
}

/*
 * 建立 2N 个点分别表示变量 i 为 true / false 的情况。
 * 如果根据约束 (i, v) imp (j, v'), 则连边 ((i, v) -> (j, v'))
 * // v, v' \in {true, false}
 * 缩 SCC 之后, 如果 scc[(i, true)] == scc[(i, false)] 则无解
 * 否则如下构造方案: 将原图拓扑排序为 tlist[1 .. cn], 之后按拓扑序
 *      sel(scc[i]) = 0 iff (exists (i, j) in E, sel(j) = 0)
 *          // and sel(scc[i]) is not set
 *      sel(scc[conf[i]]) = not sel(scc[i])
 * 由于 2-SAT 构图有对称性, 这一做法的正确性可归纳证明。
 * */
```

Blossom Algorithm

```
/*
 * 召唤友军支援 ... (
 * */

#include <cstdio>
#include <queue>

using namespace std;

const int maxn=230;

struct edge {
    int v;
    edge *next;
    edge(int _v,edge *_next)
    : v(_v),next(_next) {}
};

int n,spouse[maxn],fa[maxn],mark[maxn],link[maxn];
edge *E[maxn];
queue<int> Q;

int getfa(int i)
{
    return fa[i]==i? i: fa[i]=getfa(fa[i]);
}

void merge(int a,int b)
{
    a=getfa(a);  b=getfa(b);
    if (a!=b)  fa[a]=b;
}

void aug(int i)
{
    for(int j,k; i; i=k) {
        j=link[i], k=spouse[j];
        spouse[i]=j;  spouse[j]=i;
    }
}

int lca(int a,int b)
{
    static bool vis[maxn];
    for(int i=1; i<=n; vis[i++]=false);
    for(vis[a=getfa(a)]=true; spouse[a]; vis[a]=true)
        a=getfa(link[spouse[a]]);
    for(b=getfa(b); !vis[b]; b=getfa(link[spouse[b]]));
    return b;
}

void shrink(int a,int p)
{
    for(int b,c; getfa(a)!=p; a=c) {
        b=spouse[a];  c=link[b];
        if(getfa(c)!=p)  link[c]=b;
        mark[b]=0;  Q.push(b);
        merge(a,b);  merge(b,c);
    }
}

void findpath(int s)
{
    for(int i=1;i<=n;++i)
        fa[i]=i, mark[i]=-1, link[i]=0;
    while (!Q.empty())  Q.pop();
    for(Q.push(s); !Q.empty() && !spouse[s]; Q.pop()) {
        int a=Q.front();
        for(edge *e=E[a]; e; e=e->next) {
            int b=e->v;
            if (getfa(a)!=getfa(b) && mark[b]!=1) {
                if (mark[b]==-1) {
                    mark[b]=1;
                    link[b]=a;
                    if (!spouse[b]) {
                        aug(b);
                        break;
                    } else {
                        mark[spouse[b]]=0;
                        Q.push(spouse[b]);
                    }
                } else {
                    int p=lca(a,b);
                    if(getfa(a)!=p)  link[a]=b;
                    if(getfa(b)!=p)  link[b]=a;
                    shrink(a,p);  shrink(b,p);
                }
            }
        }
    }
}

int main()
{
    int a,b;
    scanf("%d",&n);
    while(scanf("%d%d",&a,&b)!=EOF) {
        E[a]=new edge(b,E[a]);
        E[b]=new edge(a,E[b]);
    }
    for(int i=1;i<=n;++i)
        if (!spouse[i]) findpath(i);

    int ret=0;
    for(int i=1;i<=n;++i)
        if(spouse[i])  ++ret;
    printf("%d\n",ret);
    for(int i=1;i<=n;++i)
        if(spouse[i]) {
            printf("%d %d\n",i, spouse[i]);
            spouse[spouse[i]]=0;
        }
    return 0;
}
```

Costflow with lower bound

```
/*
 * Calculate feasible flow with edges having lower bounds.
 * No sink or source included. add (sink, source, 0, inf) to fix.
 * The mcFlow part used zkw algorithm. Keep in mind its bugs.
 * Last Edit : before 2013
 * */
namespace mcFlow
{
    const int N = 2048, M = 1048576;
    struct edge {
        int t, of, f, c;
        edge *n, *op;
    } ebf[M], *ecur = ebf, *e[N];
    int dis[N], vis[N], vn, s, t, cost, flow;
    int modlabel ()
    {
        int delta = inf;
        for (int i = 1; i <= vn; ++i) if (vis[i])
            for (edge *it = e[i]; it; it = it->n)
                if (!vis[it->t] && it->f)
```

```
                    delta = min(delta, dis[it->t] + it->c - dis[i]);
        if (delta == inf) return 1;
        for (int i = 1; i <= vn; ++i)
            if (vis[i]) dis[i] += delta;
        return 0;
    }
    int augument (int x, int f)
    {
        if (x == t) { flow += f, cost += dis[s] * f; return f; }
        int r(f), de;
        vis[x] = true;
        for (edge *it = e[x]; it; it = it->n)
            if (!vis[it->t] && dis[x] == dis[it->t] + it->c && it->f)
            {
                de = augument(it->t, min(it->f, r));
                r -= de, it->f -= de, it->op->f += de;
                if (!r) break;
            }
        return f - r;
    }
    inline void solve ()
    {
        do
            do memset(vis, 0, sizeof(vis[0]) * (vn + 1));
            while (augument(s, inf));
        while (!modlabel());
    }
    inline edge* add (int s, int t, int f, int c)
    {
        //fprintf(stderr, "\t--MCFLOW--%d->%d F %d C %d\n", s, t, f, c);
        *ecur = (edge){t, f, f, c,  e[s], ecur + 1}; e[s] = ecur++;
        *ecur = (edge){s, 0, 0, -c, e[t], ecur - 1}; e[t] = ecur++;
        return ecur - 2;
    }
    inline int new_node () { return ++vn; }
    inline int set_src () { return s = new_node(); }
    inline int set_dst () { return t = new_node(); }
}

namespace acf
{
    const int N = 2048;
    int delt[N], preRes;
    inline int new_node () { return mcFlow::new_node(); }
    inline mcFlow::edge* add (int x, int y, int mn, int mx, int c)
    {
        delt[x] -= mn, delt[y] += mn;
        preRes += mn * c;
        return mcFlow::add(x, y, mx - mn, c);
    }
    inline int solve ()
    {
        int s = mcFlow::set_src(), t = mcFlow::set_dst();
        int supposed = 0;
        for (int i = 1; i <= mcFlow::vn - 2; ++i)
            if (delt[i] > 0) mcFlow::add(s, i, delt[i], 0, 0x0),
                supposed += delt[i];
            else if (delt[i] < 0) mcFlow::add(i, t, -delt[i], 0, 0x0);
        mcFlow::solve();
        if (mcFlow::flow != supposed) Throw("ACF Failed");
        return mcFlow::cost;
    }
}
```

Dijkstra

```
/*
 * SSSP - Dijkstra
 * Not able to handle negative-weighed edges.
 * */
namespace G
{
    using namespace CLR;
    const int N = 20050, M = 200050;
    struct edge { int t, w; edge *n; } ebf[M * 2], *ec = ebf, *e[N];
    int dis[N]; // 1-based

    inline void link (int u, int v, int w)
    {
        *ec = (edge){v, w, e[u]}; e[u] = ec++;
        *ec = (edge){u, w, e[v]}; e[v] = ec++;
    }
    inline void Dijkstra (int s)
    {
        priority_queue<pi> pque;
        fill(dis, dis + n + 1, inf);
        pque.push(pi(-(dis[s] = 0), s));
        while (!pque.empty())
        {
            int x = pque.top().second, w = -pque.top().first; pque.pop();
            if (dis[x] != w) continue;
            for (edge *i = e[x]; i; i = i->n)
                if (dis[x] + i->w < dis[i->t])
                    pque.push(pi(-(dis[i->t] = dis[x] + i->w), i->t));
        }
    }
}
```

Global Min-Cut

```
/*
 * O(N^3) Algorithm.
 * Last Edit: Feb 2011
 * */

#include <cstdio>
#include <cstring>
using namespace std;

#define N 505
const int inf = 100000000;
int n, g[N][N];

int min_cut ()
{
    int s, t, cut = inf, mx, u;
    static bool v[N], ex[N];
    static int w[N];
    memset(ex, 0, sizeof (int) * (n + 1));
    for (int l = 1; l < n; ++l)
    {
        memset(v, 0, sizeof (int) * (n + 1));
        memset(w, 0, sizeof (int) * (n + 1));
        for (int j = 1; j <= n; ++j)
        {
            mx = -inf, u = -1;
            for (int i = 1; i <= n; ++i)
                if (!v[i] && !ex[i] && w[i] > mx)
                {
                    mx = w[i];
                    u = i;
                }
            if (u == -1) break;
            s = t, t = u, v[u] = true;
            for (int i = 1; i <= n; ++i)
                if (!v[i]) w[i] += g[u][i];
        }
        if (w[t] < cut) cut = w[t];
        ex[s] = ex[t] = true;
        for (int i = 1; i <= n; ++i)
            if (!ex[i]) g[i][s] += g[i][t], g[s][i] += g[t][i];
        ex[s] = false;
    }
    return (cut < inf) ? cut : 0;
}

int v[N], tc;
void dfs (int x)
{
    v[x] = true;
    ++tc;
    for (int i = 1; i <= n; ++i)
        if (!v[i] && g[x][i] > 0)
            dfs(i);
}
bool connect ()
{
    if (n == 1) return false;
    memset(v, 0, sizeof v);
    tc = 0;
    dfs(1);
    return (tc == n);
}

int main ()
{
    int m, a, b, c;
    while (scanf("%d%d", &n, &m) == 2)
    {
        memset(g, 0, sizeof g);
        for (int i = 0; i < m; ++i)
        {
            scanf("%d%d%d", &a, &b, &c);
            ++a;
            ++b;
            g[a][b] += c;
            g[b][a] += c;
        }
        if (!connect()) printf("0\n");
        else printf("%d\n", min_cut());
    }
    return 0;
}
```

MST (Directed Graph)

```
/*
 * Minimal Spanning Tree on Direted Graph :
 * Improved version of Chu-liu/Edmonds Algorithm
 * Contract all circles in each iteration.
 * The time complexity is O(ElogV)
 * Tested; Last Edit Oct 2012
 * */
#include <cstdio>
#include <cstdlib>

#include <vector>
#include <algorithm>
#include <functional>
using namespace std;

const int N = 100050, M = 100050;

struct info;
vector<info*> infoList;

struct info {
    int c, ty, utime; info *pos, *neg;
    void inc () { ++utime; }
    void push () {
        if (!ty)
        {
            pos->utime += utime;
            neg->utime -= utime;
            utime = 0;
        }
    }
    info (int cost) : c(cost), ty(1), pos(0), neg(0), utime(0) {}
    info (info *p, info *n) : pos(p), neg(n),
        c(p->c - n->c), ty(0), utime(0) { infoList.push_back(this); }
    info () {}
};

struct edge {
    int u, v; info *i;
    edge (int x, int y, int w) : u(x), v(y) {
        i = new info(w);
    }
    edge () {}
} e[M];
info* origList[M];

edge* predge[N];
info* preInfo[N];
int vis[N], tmp[N], n, m;

void dMST ()
{
#define pre(i) predge[i]->u
    int res(0), root(1), n(::n);
    while (1)
    {
        fill(predge, predge + n + 1, reinterpret_cast<edge*>(0x0));
        fill(vis, vis + n + 1, 0);
        fill(tmp, tmp + n + 1, 0);
        for (int i = 1; i <= m; ++i) if (e[i].u != e[i].v)
            if (!predge[e[i].v] || predge[e[i].v]->i->c > e[i].i->c)
                predge[e[i].v] = e + i;
        int n_vert = 0;
        for (int i = 1, j; i <= n; ++i) if (i != root)
        {
            if (!predge[i])
            {
                res = -1;
                break;
            }
            predge[i]->i->inc();
            res += predge[i]->i->c;
            for (j = i; !vis[j] && tmp[j] != i && j != root; j = pre(j))
                tmp[j] = i;
            if (vis[j] || j == root)
                continue;
            ++n_vert;
            for (; !vis[j]; j = predge[j]->u) vis[j] = n_vert;
        }
        if (!n_vert) break;
        for (int i = 1; i <= n; ++i)
            if (!vis[i]) vis[i] = ++n_vert;
        for (int i = 1; i <= n; ++i)
            preInfo[i] = predge[i] ? predge[i]->i : 0x0;
        for (int i = 1; i <= m; ++i)
        {
            int v = e[i].v;
            e[i].u = vis[e[i].u]; e[i].v = vis[e[i].v];
            if (e[i].u != e[i].v)
                e[i].i = new info(e[i].i, preInfo[v]);
        }
        n = n_vert;
        root = vis[root];
    }
    if (res == -1) printf("-1\n");
    else
    {
        printf("%d\n", res);
        for (auto it = infoList.rbegin(); it != infoList.rend(); ++it)
            (*it)->push();
        for (int i = 1; i <= m; ++i)
        {
            if (origList[i]->utime < 0) throw;
            if (origList[i]->c && origList[i]->utime)
                printf("%d ", i);
        }
        puts("");
    }
```

```
}

int main ()
{
#ifdef LOCAL
    freopen("in", "r", stdin);
#else
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
    scanf("%d%d", &n, &m);
    for (int i = 1, a, b, c; i <= m; ++i)
    {
        scanf("%d%d%d", &a, &b, &c);
        e[i] = edge(a, b, c);
        origList[i] = e[i].i;
    }
    dMST();
    return 0;
}
```

MST (Directed Graph, V^3 Approach)

```
/*
 * Minimal Spanning Tree on Direted Graph : Chu-liu/Edmonds Algorithm
 * Time Complexity O(V^3)
 * Code checked
 * Last Edit : Before 2013
 * */
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;

const int N = 256;
const double inf = 1e50, eps = 1e-6;

double G[N][N];
int vis[N], n, pre[N], removed[N];

void dfs (int x) {
    vis[x] = 1;
    for (int i = 1; i <= n; ++i) if (G[x][i] < inf && !vis[i]) dfs(i);
}

void dmst ()
{
    double res(0.0);
    fill(removed, removed + n + 1, 0);
    while (1)
    {
    // For each node, let its precursor be the node with minimum weigh to it
        for (int i = 2; i <= n; ++i) if (!removed[i])
        {
            G[pre[i] = i][i] = inf;
            for (int j = 1; j <= n; ++j) if (!removed[j])
                if (G[j][i] < G[pre[i]][i]) pre[i] = j;
        }
        bool found = false;
        // Find a circuit and reduce it
        for (int i = 2; i <= n; ++i) if (!removed[i])
        {
            int j, k; fill(vis, vis + n + 1, 0);
            for (j = i; j != 1 && !vis[j]; j = pre[j]) vis[j] = 1;
            if (j == 1) continue;
            found = true; i = j;
            res += G[pre[i]][i];
            /*
             * Contract the circle :
             * (Temporarily) include all the edges on the circle,
             * and adjust the cost of edges adjacent to the circle,
             * so that to select one of them ((u, v) for instance)
             * is equivalent to remove the edge (pre<v>, v) on the circle.
             * Easy to figure out that in the last iteration,
             * when selecting all remaining |V'|-1 edges,
             * the subgraph selected will expand to a rooted tree.
             * */
            for (j = pre[i]; j != i; j = pre[j]) res += G[pre[j]][j], removed[j] = 1;
            for (k = 1; k <= n; ++k) if (!removed[k])
                if (G[k][i] < inf) G[k][i] -= G[pre[i]][i];
            // Adjust the cost of edges
            for (j = pre[i]; j != i; j = pre[j])
                for (k = 1; k <= n; ++k) if (!removed[k])
                {
                    if (G[k][j] < inf)
                        G[k][i] = min(G[k][i], G[k][j] - G[pre[j]][j]);
                    G[i][k] = min(G[i][k], G[j][k]);
                    // as i represents the the whole circle now.
                }
            break;
        }
        if (!found)
        {
            // select all |V'|-1 edges
            for (int i = 2; i <= n; ++i) if (!removed[i]) res += G[pre[i]][i];
            break;
        }
    }
    printf("%.2lf\n", res);
}

int x[N], y[N];

inline double sqr (double x) { return x * x; }
inline double edist (int a, int b) { return sqrt(sqr(double(x[a] - x[b])) + sqr(double(y[a] - y[b])));

void solve ()
{
    int m; scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)
        scanf("%d%d", x + i, y + i);
    for (int i = 0; i <= n; ++i)
        for (int j = 0; j <= n; ++j)
            G[i][j] = inf * 2.;
    while (m--)
    {
        int s, t; scanf("%d%d", &s, &t);
        G[s][t] = edist(s, t);
    }
    fill(vis, vis + n + 1, 0); dfs(1);
    bool able(true);
    for (int i = 1; i <= n; ++i) if (!vis[i]) { able = false; break; }
    if (!able) puts("poor snoopy"); else dmst();
}

int main ()
```

```
{
#ifdef LOCAL
    freopen("in", "r", stdin);
#else
    freopen("network.in", "r", stdin);
    freopen("network.out", "w", stdout);
#endif
    int _; scanf("%d", &_); while (_--) solve();
    return 0;
}
```

MST with Degree Limit

```
/*
 * Minimum Spanning Tree with degree limit on undirected graphs.
 * O(N^3) due to poor implementation.
 * Last Edit : Before 2013
 * */
#include <cstdio>
#include <iostream>
#include <map>
#include <queue>
#include <vector>
#include <utility>
#include <algorithm>
using namespace std;

#define F first
#define S second

typedef pair<int, int> pi;
const int N = 500;

struct sset {
    int f[N];
    sset () { for (int i = 0; i < N; ++i) f[i] = i; }
    int gf (int x) { return f[x] == x ? x : (f[x] = gf(f[x])); }
    void merge (int x, int y) { f[gf(x)] = gf(y); }
} st;
struct enode {
    int x, y, w;
    bool operator< (const enode &b) const
        { return w < b.w; }
} e[N * N];

int g[N][N], orig[N][N], n, m, res, deglimit;

inline void addEdge (int x, int y, int w) { res += w; g[x][y] = g[y][x] = w; }
inline void delEdge (int x, int y) { res -= g[x][y]; g[x][y] = g[y][x] = 0; }

void mst ()
{
    sort(e + 1, e + m + 1);
    for (int i = 1; i <= m; ++i)
        if (e[i].x != 1 && e[i].y != 1 &&
            st.gf(e[i].x) != st.gf(e[i].y))
        {
            st.merge(e[i].x, e[i].y);
            addEdge(e[i].x, e[i].y, e[i].w);
        }
}

enode dist[N];
bool direct[N];
void trav (int x, int ax, enode maxdist)
{
    dist[x] = maxdist;
    for (int y = 1; y <= n; ++y) if (g[x][y] && y != ax)
        trav(y, x, max(maxdist, (enode){x, y, g[x][y]}));
}

void dlmst ()
{
    map<int, pi> mindist;
    mst();
    for (int i = 2; i <= n; ++i)
        if (orig[i][1])
        {
            pi cur = pi(orig[i][1], i);
            if (!mindist.count(st.gf(i)) || mindist[st.gf(i)] > cur)
                mindist[st.gf(i)] = cur;
        }
    for (__typeof(mindist.begin()) it = mindist.begin(); it != mindist.end(); ++it)
        addEdge(it->S.S, 1, it->S.F), direct[it->S.S] = 1;
    int cdeg = mindist.size();
    for (; cdeg < deglimit; ++cdeg)
    {
        trav(1, 0, (enode){0, 0, 0});
        int bv = 0, ch = -1;
        for (int i = 2; i <= n; ++i)
            if (orig[1][i] && !direct[i])
                if (orig[1][i] - dist[i].w < bv)
                    bv = orig[1][i] - dist[i].w,
                    ch = i;
        if (bv == 0) return;
        delEdge(dist[ch].x, dist[ch].y);
        addEdge(1, ch, orig[1][ch]);
    }
}

map<string, int> idMap;
int get_id (string cur) {
    return idMap[cur] ? idMap[cur] : (idMap[cur] = ++n);
}

void init ()
{
    cin >> m;
    get_id("Park");
    for (int i = 1; i <= m; ++i)
    {
        string x, y;
        int w, a, b;
        cin >> x >> y >> w;
        a = get_id(x); b = get_id(y);
        e[i] = (enode){a, b, w};
        orig[a][b] = orig[b][a] = w;
    }
    cin >> deglimit;
}

int main ()
{
    freopen("in", "r", stdin);
    init ();
    dlmst();
    cout << "Total miles driven: " << res << endl;
    return 0;
}
```

Maxflow (Dinic)

```
/*
 * Dinic Algorithm for maximum flow.
 * Last Edit : Apr 2013
 * */

typedef long long LL;

namespace Dinic {
    const int N = 16384, M = N << 4;
    struct edge { int t; LL c; edge *op, *n; }
        ebf[M], *ec = ebf, *e[N], *er[N];
    int n, s, t, d[N];
    inline void add_edge (int s, int t, LL c)
    {
        *ec = (edge){t, c, ec + 1, er[s]}; er[s] = ec++;
        *ec = (edge){s, 0, ec - 1, er[t]}; er[t] = ec++;
    }
    inline void reset ()
    {
        memset(er, 0, sizeof(er[0]) * (n + 1));
        memset(d, 0, sizeof(d[0]) * (n + 1));
        n = 0; ec = ebf;
    }
    bool modlabel ()
    {
        static int que[M];
        int l(0), r(0), x, y;
        memcpy(e, er, sizeof(er[0]) * (n + 1));
        memset(d, 0xff, sizeof(d[0]) * (n + 1));
        d[que[++r] = s] = 0;
        while (l != r)
            for (edge *i = er[x = que[++l]]; i; i = i->n)
                if (i->c && d[y = i->t] < 0) d[que[++r] = y] = d[x] + 1;
        return d[t] >= 0;
    }
    LL augument (int x, LL c)
    {
        if (x == t) return c;
        LL r(c), de; int y; edge *i;
        for (i = e[x]; i; i = i->n)
            if (i->c && d[x] + 1 == d[y = i->t])
            {
                de = augument(y, min(r, i->c));
                i->c -= de, i->op->c += de, r -= de;
                if (!r) break;
            }
        e[x] = i;
        if (r) d[x] = -1; // CRUCIAL OPTIMIZATION
        return c - r;
    }
    LL solve (int _n, int _s, int _t)
    {
        LL res = 0, de; n = _n; s = _s; t = _t;
        while (modlabel()) while (de = augument(s, ~0ULL >> 4)) res += de;
        return res;
    }
}
```

MinCostMaxflow (SSP Approach)

```
/*
 * Naive Successive Shortest Path Algorithm for cost flow.
 * Last Edit : before 2013
 * */

namespace mcFlow {
    const int N = 6050, M = 8000000, inf = ~0U >> 1;
    struct edge { int s, t, fl, co; edge *n, *d; }
        ebf[M], *ec = ebf, *e[N], *pre[N];
    int n, s, t, cost, flow, dist[N], inque[N];
    inline void addEdge (int s, int t, int f, int c)
    {
        *ec = (edge){s, t, f, c, e[s], ec + 1}; e[s] = ec++;
        *ec = (edge){t, s, 0,-c, e[t], ec - 1}; e[t] = ec++;
    }
    inline int addNode () { return ++n; }
    inline int addSrc () { return s = ++n; }
    inline int addSnk () { return t = ++n; }
    void augument ()
    {
        int co(0), fl(inf);
        for (int cr = t; cr != s; cr = pre[cr]->s)
            co += pre[cr]->co, fl = min(fl, pre[cr]->fl);
        cost += co * fl, flow += fl;
        for (int cr = t; cr != s; cr = pre[cr]->s)
            pre[cr]->fl -= fl, pre[cr]->d->fl += fl;
    }
    bool modlabel ()
    {
        fill(dist + 1, dist + n + 1, -inf);
        fill(inque + 1, inque + n + 1, 0);
        queue<int> que;
        que.push(s); dist[s] = 0; inque[s] = true;
        while (!que.empty())
        {
            int x = que.front(), y;
            que.pop(); inque[x] = false;
            for (edge *i = e[x]; i; i = i->n)
                if (i->fl && dist[x] + i->co > dist[y = i->t])
                {
                    dist[y] = dist[x] + i->co, pre[y] = i;
                    if (!inque[y]) inque[y] = true, que.push(y);
                }
        }
        return dist[t] > -inf;
    }
    void solve () { while (modlabel()) augument(); }
}
```

MinCostMaxflow (Zkw Algorithm)

```
/*
 * Zkw's Algorithm for min-cost-max-flow, inspired by SAP and KM.
 * Keep in mind that when there are negative-weighed edges along with zero-
 * weighed edges, the algorithm will misbehave. It will try to augument along
 * zero-weighed edges first.
 * The fix is to force to augument along all negative edges before starting,
 * using the idea of excess flow.
 * Last Edit : before 2013
 * */

namespace Zkw {

    const int M = int(2e5) + 50, N = int(1e3) + 50, inf = ~0u >> 2;

    struct edge { int t, f, c; edge *n, *d; }
    eb[M << 1], *ec = eb, *e[N];

    inline void add_edge (int s, int t, int c, int f)
    {
        *ec = (edge){t, f, c, e[s], ec + 1}; e[s] = ec++;
        *ec = (edge){s, 0,-c, e[t], ec - 1}; e[t] = ec++;
    }

    int vis[N], pi[N], slack[N], n, s, t, flow, cost, arr[N];

    bool modlabel ()
    {
        int d = inf;
        for (int i = 1; i <= n; ++i)
            if (!vis[i] && slack[i] < d) d = slack[i];
        if (d == inf) return false;
        for (int i = 1; i <= n; ++i) {
            if (vis[i]) pi[i] += d; slack[i] = inf;
        }
        return true;
    }
    int augument (int x, int f)
    {
        if (x == t) {
            arr[flow + 1] = pi[s]; flow += f; cost += f * pi[s]; return f;
        }
        vis[x] = true; int r(f), y, d;
        for (edge *i = e[x]; i; i = i->n) if (i->f && !vis[y = i->t])
            if (pi[x] == i->c + pi[y])
            {
                d = augument(y, min(r, i->f));
                if (!(i->f -= d, i->d->f += d, r -= d)) break;
            }
            else slack[y] = min(slack[y], -pi[x] + pi[y] + i->c);
        return f - r;
    }

    void costflow ()
    {
        fill(slack + 1, slack + n + 1, inf);
        do
            do memset(vis, 0, sizeof(int) * (n + 1));
            while (augument(s, inf));
        while (modlabel());
    }

}
```

SPFA

```
namespace SPFA {
    const int V = 100000;
    struct enode { int t; lint w; };
    vector<enode> e[V];
    bool inque[V];
    queue<int> que;
    int ln, pre[V];

    inline void addEdge (int a, int b, int c)
    {
        e[a].push_back((enode){b, c});
        e[b].push_back((enode){a, c});
    }

    void reset ()
    { for (int i = 0; i <= ln; ++i) e[i].clear(); }

    void solve (int n, int s, int t)
    {
        ln = n;
        fill(dis + 1, dis + n + 1, inf64);
        fill(inque + 1, inque + n + 1, 0);
        dis[s] = 0; inque[s] = 1; que.push(s);
        while (!que.empty())
        {
            int x = que.front();
            inque[x] = false; que.pop();
            for (vector<enode>::iterator it = e[x].begin();
                    it != e[x].end(); ++it)
                if (dis[x] + it->w < dis[it->t])
                {
                    dis[it->t] = dis[x] + it->w; pre[it->t] = x;
                    if (!inque[it->t]) inque[it->t] = 1, que.push(it->t);
                }
        }
    }
}
```

Tarjan (BCC)

```
/*
 * Tarjan algorithm on undirected graph
 * Finding cut points, bridges, biconnected components &
 * 2-edge-connected components
 * (biconnected components refers to vertex-bcc;
 * 2-edge-connected components refers to maximal bridgeless subgraphs.)
 * Last Edit : 15-5-2013
 * */

int dfn[N], low[N], ts, n;
long long f[N];

void dfs (int x, int ax = 0) {
    int y, cc(0), ch(0);
    dfn[x] = low[x] = ++ts;
    for (edge *i = e[x]; i; i = i->n) if (i->v) {
        i->v = i->d->v = 0;
        if (!dfn[y = i->t]) {
            ++ch;
            dfs(y, x);
            // low[y] == dfn[x]:
            // dumplicated edge -> x is a cut-point, i is not a bridge
            if (ax != 0 && low[y] >= dfn[x]) ;//x is a cut point
            if (low[y] > dfn[x]) ;//i is a bridge
            low[x] = min(low[x], low[y]);
        } else low[x] = min(low[x], dfn[y]);
    }
    if (ax == 0 && ch >= 2) ; // x is a cut point
}
int solve () { for (int i = 1; i <= n; ++i) if (!dfn[i]) dfs(i); }

//in order to find edge biconnected components,
// just remove the bridges and do floodfill
//in order to get vertex bcc, use the code below

/*
 * UPDATE@20130515 : The previous version is wrong. Code fixed.
 * The time complexity is now linear.
 * further test may be needed.
 * */

struct edge { int s, t, i; edge *n; } ebf[M * 2], *ec = ebf, *e[N];
inline void add_edge (int u, int v, int i) {
    *ec = (edge){u, v, i, e[u]}; e[u] = ec++;
    *ec = (edge){v, u, i, e[v]}; e[v] = ec++;
}

bool ans[N];
int top, dfn[N], low[N], times, scnt, evis[M], sID[M];
edge* S[M];

void tarjan (int x, int ax = 0) {
#define edges(x) ebf[x*2-2]
    dfn[x] = low[x] = ++times;
    for (edge *i = e[x]; i; i = i->n) {
        int y = i->t;
        if (evis[i->i]) continue;
        S[++top] = i;
        evis[i->i] = 1;
        if (dfn[y]) {
            low[x] = min(low[x], dfn[y]);
            continue;
        }
        tarjan(y, x);
        if (low[y] == dfn[x]) { // a Vertex-BCC with cutpoint x
            for (++scnt; S[top]->s != x; sID[S[top--]->i] = scnt) ;
            //assert(S[top]->s == x && S[top]->t == y);
            sID[S[top--]->i] = scnt;
        }
        low[x] = min(low[x], low[y]);
    }
    if (dfn[x] == low[x]) {
        // there should be only one edge connecting its ancestor
        if (ax == 0) return;
        //assert(S[top]->s == ax && S[top]->t == x);
        sID[S[top--]->i] = ++scnt;
    }
}
```

Tarjan (SCC)

```
/*
 * Tarjan Algorithm on directed graph
 * Finding SCC
 * Last Edit : before 2013
 * */

#define pback push_back
#define for_(it,E) for(__typeof(E.begin())it=E.begin();it!=E.end();++it)

const int N = 5005, VT = N << 1;

vector<int> orig[N], esat[VT], modi[VT], conf[VT];
int on, cn, in;

namespace tarjan {
    int dfn[VT], low[VT], ins[VT], *cc, cnt, ctime;
    stack<int> s;
    void dfs (int x)
    {
        dfn[x] = low[x] = ++ctime; ins[x] = 1;
        s.push(x);
        for_(it, esat[x])
            if (!dfn[*it]) dfs(*it), low[x] = min(low[x], low[*it]);
            else if (ins[*it]) low[x] = min(low[x], dfn[*it]);
        if (dfn[x] == low[x])
        {
            cc[x] = ++ cnt; ins[x] = 0;
            while (s.top() != x) cc[s.top()] = cnt, ins[s.top()] = 0, s.pop();
            s.pop();
        }
    }
    int solve (int *res)
    {
        cc = res; cnt = 0;
        for (int i = 1; i <= on; ++i) if (!dfn[i]) dfs(i);
        for (int i = 1; i <= on; ++i)
            for_(it, esat[i]) if (cc[i] != cc[*it])
                modi[cc[i]].pback(cc[*it]);
        return cnt;
    }
}
```

Math

Factorial

```
/*
 * given p, P=p^k, calculate N = a(modP) * p^b
 * O(P) - O(logN)
 * */
struct mint { LL r, c; };

LL fact[N], p, P, cnt[N], n, m;

LL fpow (LL x, LL m, LL modu)
{
    if (!m) return 1;
    LL y = fpow(x, m >> 1, modu), r = y * y % modu;
    return (m & 1) ? r * x % modu : r;
}
LL exGCD (LL a, LL b, LL &x, LL &y)
{
    if (!b) { x = 1, y = 0; return a; }
    LL d = exGCD(b, a % b, y, x);
    y -= x * (a / b);
    return d;
}

inline mint getInv (mint p) {
    return (mint){fpow(p.r, P * (::p - 1) / ::p - 1, P), -p.c};
}
mint operator* (mint a, mint b) { return (mint){a.r * b.r % P, a.c + b.c}; }

mint gfact (LL n)
{
    if (n < p) return (mint){fact[n], 0};
    LL base = fpow(fact[P - 1], n / P, P) * fact[n % P] % P;
    return (mint){base, n / p} * gfact(n / p);
}
```

Fast Fourier Transform

```
typedef long long lint;
const lint P = (15 << 27) + 1, X = 27, R = 137,
    MXN = 262144;
lint a[MXN + 1], b[MXN + 1], c[MXN + 1];
int n, mxa, mxb, mxc;

inline lint mult (lint a, lint b) { return a * b % P; }
lint fpow (lint a, lint b)
{
    if (b == 0) return 1;
    lint p = fpow(a, b >> 1), r = mult(p, p);
    return (b & 1) ? mult(r, a) : r;
}

inline void calc (lint a[], lint wt)
{
    for (lint m(2); m <= n; m <<= 1)
        for (lint k(0), wi(fpow(wt, n / m)); k < n; k += m) //~
            for (lint j(0), w(1); (j << 1) < m; ++j, w = mult(w, wi))
            {
                lint u(a[k + j]), v(mult(a[k + j + (m >> 1)], w));
                a[k + j] = (u + v) % P;
                a[k + j + (m >> 1)] = (u - v + P) % P;
            }
}

inline void bitrc (lint a[])
{
    int mx(0); while ((1 << mx) < n) ++mx;
    for (int i = 0; i <= n; ++i)
    {
        int x(i), y(0);
        for (int j(0); j < mx; ++j)
            y = (y << 1) | (x & 1), x >>= 1;
        if (i < y) swap(a[i], a[y]);
    }
}

inline void FFT (lint a[])
{
    bitrc(a);
    calc(a, fpow(R, (15 << X) / n));
}
//n is a power of 2
inline void IFFT (lint a[])
{
    bitrc(a);
    calc(a, fpow(R, (15 << X) / n * (P - 2)));
    lint inv_n = fpow(n, P - 2);
    for (int i = 0; i <= n; ++i)
        a[i] = mult(a[i], inv_n);
}
```

Gaussian Elimination

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <algorithm>
using namespace std;

const int N = 128;

double buff[N][N], *a[N], sol[N];
int n;

void gauss ()
{
    for (int i = 1; i <= n; ++i) a[i] = buff[i];
    for (int i = 1; i <= n; ++i)
    {
        double cur = 0; int k(i);
        for (int j = i; j <= n; ++j)
            if (fabs(a[j][i]) > cur)
                { cur = fabs(a[k = j][i]); break; }
        swap(a[k], a[i]);
        for (int j = i + 1; j <= n; ++j)
        {
            double d = -a[j][i] / a[i][i];
            for (int k = i; k <= n + 1; ++k)
                a[j][k] += d * a[i][k];
        }
    }
}
void evaluate ()
{
    for (int i = n; i > 0; --i)
    {
        sol[i] = a[i][n + 1] / a[i][i];
        for (int j = i - 1; j > 0; --j)
            a[j][n + 1] -= sol[i] * a[j][i], a[j][i] = 0.0;
    }
}

int main ()
{
    ios::sync_with_stdio(false);
    cin >> n;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n + 1; ++j)
            cin >> buff[i][j];
    gauss();
    evaluate();
    for (int i = 1; i <= n; ++i)
        cout << int(sol[i] + 0.5) << ' ';
    cout << endl;
    return 0;
}
```

Header for Number Theory

```
typedef long long LL;

// gcd
int gcd (int a, int b) { return b ? gcd(b, a % b) : a; }

LL getRoot (LL p) // Z*[p] 原根
{ // p is a prime
    static vector<LL> fac;
    decompose(p - 1, fac); // <-- phi(p)
    for (LL i = 2; i < p; ++i)
    {
        bool able = true;
        for_(it, fac) if (power(i, (p - 1) / *it) == 1)
            { able = false; break; }
        if (able) return i;
    }
    return -1;
}

LL babyStep (LL a, LL b, LL p) // 离散模对数
{
    LL sq = sqrt(p);
    static map<LL, int> id; id.clear();
    for (LL i = 0, cp = 1; i <= sq; ++i, cp = cp * a % p)
        if (!id.count(cp)) id[cp] = i;
    for (LL i = 0, cq = b, ivg = power(power(a, sq), p - 2);
        i <= sq; ++i, cq = cq * ivg % p)
        if (id.count(cq)) return i * sq + id[cq];
    return -1;
}

LL exGCD (LL a, LL b, LL &x, LL &y) // 扩展 GCD
{
    if (!b) { x = 1, y = 0; return a; }
    LL d = exGCD(b, a % b, y, x);
    y -= x * (a / b);
    return d;
}

void linerEqu (LL a, LL b, LL c, vector<LL> &res) // 模线性方程
```

```
{
    LL x, y, d = exGCD(a, c, x, y);
    res.clear();
    if (b % d) return;
    x = (x % c + c) * (b / d) % c;
    for (LL i = 0, t = c / d; i < d; ++i)
        res.push_back((x + i * t) % c);
}

LL modequ (vector< pair<LL, LL> > &a) // 模线性方程组
{
    LL n(1), res(0);
    for_(i, a) n *= i->second;
    for_(i, a)
    {
        LL m = n / i->second, x, y;
        if (exgcd(m % i->second, i->second, x, y) != 1) throw;
        res += x * m * i->first; res %= n;
    }
    while (res < 0) res += n;
    return res;
}

LL mult (LL a, LL b, LL modu) // multiply in case of overflow
{
#ifndef LOCAL
    LL res = 0;
    for (; b; b >>= 1)
    {
        if (b & 1)
        {
            res += a;
            while (res >= modu) res -= modu;
        }
        a <<= 1; // A
        while (a >= modu) a -= modu;
    } return res;
#else
    return (__int128)a * b % modu;
#endif
}
```

```
inline LL mult (LL a, LL b, LL c) // another approach with some precision error
{
    LL t = (LL)(a / (__float128)c * b + 1e-3);
    LL ret = a * b - c * t;
    return (ret % c + c) % c;
}

inline LL mult (LL x, LL y, LL z) { return x * y % z; }

LL fpow (LL x, LL m, LL modu) // iterative fast power
{
    LL res(1);
    for (; m; m >>= 1)
    {
        if (m & 1) res = mult(res, x, modu);
        x = mult(x, x, modu); // X
    }
    return res;
}

int rtest (LL x) // Rabin-Miller
{
    int P[10] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
    if (x < 100)
    {
        for (int i = 2; i < x; ++i)
            if (x % i == 0) return 0;
        return 1;
    }
    for (int i = 0; i < 10; ++i)
        if (fpow(P[i], x - 1, x) % x != 1) return 0;

    return 1;
}

inline LL C (LL n, LL m) // Using Lucas Theorem
{
    LL res(1);
    while (n) {
        int x = n % P, y = m % P;
        if (x >= y)
        {
            LL cur = (LL)fac[x] * invfac[y] % P * invfac[x - y] % P;
            res = res * cur % P;
        }
        else return 0;
        n /= P, m /= P;
    }
    return res;
}

template <int N>
struct primeTable // Euler Sieve
{
    int sdiv[N], primes[N], pcnt;
    primeTable ()
    {
        pcnt = 0;
        for (int i = 2; i < N; ++i)
        {
            if (!sdiv[i]) primes[++pcnt] = (sdiv[i] = i);
            for (int j = 1; j <= pcnt && primes[j] * i < N; ++j)
            {
                sdiv[primes[j] * i] = primes[j];
                if (i % primes[j] == 0) break;
            }
        }
    }
};

struct frac // Fraction Class
{
    LL num, den;
    frac (LL n, LL d) {
        if (n == 0 || d == 0) { num = 0; den = 1; }
        else { LL r = gcd(n, d); num = n / r; den = d / r; }
        if (den < 0) num = -num, den = -den;
    }
    frac operator+ (const frac &b) {
        return frac(num * b.den + den * b.num, den * b.den);
    }
    frac operator- (const frac &b) {
        return frac(num * b.den - den * b.num, den * b.den);
    }
    frac operator* (const frac &b) {
        return frac(num * b.num, den * b.den);
    }
    frac operator/ (const frac &b) {
        return frac(num * b.den, den * b.num);
    }
};
ostream& operator<< (ostream &a, const frac &b) {
    return a << b.num << ' ' << b.den;
}

// vim:ts=4:fdm=syntax
```

Karatsuba Algorithm

```
const int MXN = 262145;
typedef long long lint;
lint a[MXN], b[MXN], c[MXN], d[MXN];
int sa, sb, sc, t;
lint buff[MXN * 30], *cur = buff;

inline lint* newla (int size) { return cur += size, cur - size; }
inline void setcur (lint *a) { cur = a; }

void multiply (const lint a[], const lint b[], int n, lint c[])
{
    if (n == 1) c[0] = a[0] * b[0], c[1] = 0;
    else
    {
        int nn = n >> 1;
        lint *bak = cur,
            *A = newla(n), *B = newla(n), *C = newla(n),
            *D = newla(nn), *E = newla(nn);
        multiply (a, b, nn, A);
        multiply (a + nn, b + nn, nn, B);
        for (int i = 0; i < nn; ++i)
            D[i] = a[i] + a[nn + i], E[i] = b[i] + b[nn + i];
        multiply (D, E, nn, C);
        for (int i = 0; i < n; ++i) c[i] = A[i], c[i + n] = B[i];
        for (int i = 0; i < n; ++i) c[i + nn] += C[i] - A[i] - B[i];
        setcur(bak);
    }
}

inline void process (int csz)
{
    int sz(1); ++csz;
    while (sz < csz) sz <<= 1;
    multiply (a, b, sz, d);
    multiply (d, c, sz, a);
    sz <<= 1;
    for (int i = 1; i <= sz; ++i)
        a[i] += a[i - 1];
}
```

Rho Decomposition

```
typedef long long LL;
typedef long double LD;

int gcd (LL x, LL y) { return y ? gcd(y, x % y) : x; }

inline LL mult (LL a, LL b, LL modu)
{
    LL t = (LL)(a / (LD)modu * b + 1e-3), ret = a * b - t * modu;
    if(ret < 0) ret += modu; return ret;
}

inline LL fpow (LL x, LL m, LL modu)
{
    LL res(1);
    for (; m; m >>= 1)
    {
        x = mult(x, x, modu);
        if (m & 1) res = mult(res, x, modu);
    }
    return res;
}
inline bool primitive (LL x) // x should be large enough
{
    if (x < 300) throw;
    int p[10] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
    for (int i = 0; i < 10; ++i)
        if (fpow(p[i], x - 1, x) % x != 1) return 0;
    return 1;
}

map<LL, int> plist;

inline void naive_expand (LL x)
{
    for (int i = 2; i * i <= x; ++i)
        while (x % i == 0) x /= i, ++plist[i];
    if (x > 1) ++plist[x];
}

inline LL rho_iterate (LL a, LL ival)
{
    LL x = 2, y = 2, d = 1;
    while (d == 1)
    {
        x = (mult(x, x, a) + ival) % a;
        y = (mult(y, y, a) + ival) % a;
        y = (mult(y, y, a) + ival) % a;
        d = gcd(abs(y - x), a);
    }
    return d;
}
inline void expand (LL a, LL cival)
{
    if (a < 300) { naive_expand(a); return; }
    if (primitive(a)) { ++plist[a]; return; }
    LL d;
    while ((d = rho_iterate(a, cival)) == a) --cival;
    expand(d, cival); expand(a / d, cival);
}

// expand(a, 103);
```

Simplex Algorithm

```
#include <bits/stdc++.h>
using namespace std;
```

```
typedef long double LD;

const LD inf = 1e100, eps = 1e-9;
const int N = 505;

inline int sgn (LD x) { return fabs(x) < eps ? 0 : x > 0 ? 1 : -1; }

LD A[N][N];
int equ[N], var[N], n, m;

void pivot (int e, int v)
{
    static int que[N], f;

    swap(equ[e], var[v]);
    f = 0;
    for (int i = 0; i <= m; ++i)
        if (sgn(A[e][i])) que[++f] = i;

    A[e][v] = 1 / A[e][v];
    for (int i = 1; i <= f; ++i)
        if (que[i] != v) A[e][que[i]] *= A[e][v];

    for (int i = 0; i <= n + 1; ++i)
        if (i != e && sgn(A[i][v]))
        {
            LD t = A[i][v]; A[i][v] = 0;
            for (int j = 1; j <= f; ++j)
                A[i][que[j]] -= A[e][que[j]] * t;
        }
}

/*
   max(\sum A_{0,i}x_i)
   s.t. \sum A_{i,j}x_j \leq A_{i,0}
 */
LD base_simplex ()
{
    // assert A[0][i] >= 0
    while (1)
    {
        int v = 1, e = 0;
        for (int i = 2; i <= m; ++i)
            if (A[0][i] > A[0][v]) v = i;

        if (sgn(A[0][v]) <= 0) return -A[0][0];

        for (int i = 1; i <= n; ++i)
            if (sgn(A[i][v]) > 0 && (e == 0 || A[i][0] / A[i][v] < A[e][0] / A[e][v]))
                e = i;

        if (! e) return +inf; // unbounded
        pivot(e, v);
    }
}
LD simplex ()
{
    for (int i = 1; i <= n; ++i) equ[i] = -i;
    for (int i = 1; i <= m; ++i) var[i] = i;

    int v = 1;
    for (int i = 2; i <= n; ++i)
        if (A[i][0] < A[v][0]) v = i;

    if (A[v][0] < 0) // Initialization (CLRS)
    {
        // Let the original objective function get transformed with the matrix,
        // so that we don't need to restore the basic variables after init process.
        for (int i = 1; i <= m; ++i)
        {
            A[n + 1][i] = A[0][i];
            A[0][i] = 0;
        }
        ++m; var[m] = m;
        A[0][m] = -1;
        for (int i = 1; i <= n; ++i) A[i][m] = -1;
        pivot(v, m);
        if (sgn(base_simplex()) != 0)
            return -inf; // no solution

        for (int i = 0; i <= m; ++i)
        {
            A[0][i] = A[n + 1][i];
            A[n + 1][i] = 0;
        }

        // add constraint x_m <= 0
        int p = find(var + 1, var + 1 + m, m) - var;
        if (var[p] == m)
        {
            ++n;
            A[n][p] = 1;
        }
        else
        {
            p = find(equ + 1, equ + 1 + n, m) - equ;
            ++n;
            for (int i = 0; i <= m; ++i)
                A[n][i] = -A[p][i];
        }
    }
    return base_simplex();
}
```

Simpson Integration

```
/*
 * Last Edit : 09-05-2013
 */
double f (double t)
{
    return f(t);
}
 /*
    When the function is approximately piecewise linear
inline double eval (double l, double r, double fl, double fr, double fm) {
    return (fl + fr + fm) / 3.0 * (r - l);
}*/
inline double eval (double l, double r, double fl, double fr, double fm) {
    return (fl + fr + 4 * fm) / 6.0 * (r - l);
}
double rsimpson (double l, double r, double fl, double fm, double fr)
{
//  fprintf(stderr, "%.2lf %.2lf\n", l, r);
    double m = (l + r) / 2,
        ml = (l + m) / 2, mr = (m + r) / 2,
        fml = f(ml), fmr = f(mr),
        s = eval(l, r, fl, fr, fm), sl = eval(l, m, fl, fm, fml),
        sr = eval(m, r, fm, fr, fmr);
    if (fabs(sl + sr - s) < eps) return sl + sr;
//    if (r - l < 1e-3 && fabs(sl + sr - s) < eps) return sl + sr;
    return rsimpson(l, m, fl, fml, fm) + rsimpson(m, r, fm, fmr, fr);
}

double rsimpson (int l, int r) {
    return rsimpson(l, r, f(l), f((l + r) / 2), f(r));
}

//to prevent the loss of precision, do not call rsimpson(-inf,+inf) directly
```

String

Aho-Corasick Automaton

```
/*
 * trie graph
 * */

struct node { bool fl; node *f, *n[26]; }
    buff[N << 1], *rt = buff, *cr = buff + 1;
inline node* new_node () { return cr++; }
int alpha;

void expand (node *c, char *s)
{
    if (!*s) { c->fl = true; return; }
    int p = *s - 'a';
    expand(c->n[p] ? c->n[p] : (c->n[p] = new_node()), s + 1);
}

void build_fail ()
{
    queue<node*> que;
    que.push(rt); rt->f = rt;
    while (!que.empty())
    {
        node *t = que.front(); que.pop();
        for (int i = 0; i < alpha; ++i)
        {
            node *f = (t == rt) ? t : t->f->n[i];
            if (t->n[i])
            {
                t->n[i]->f = f;
                t->n[i]->fl |= f->fl || t->fl; // ,b - proof
                que.push(t->n[i]);
            }
            else t->n[i] = f;
        }
    }
}

//traditional implementation
//when |SIGMA| is big
struct node
{
    int id, fl;
    node *f;
    map<int, node*> n;
} Tbuf[L << 2], *Tcur = Tbuf + 1, *root = Tbuf;

vector<int> same[N];
int head[N];

inline node* newNode () { return Tcur++; }

void expand (node *t, int *s, int id)
{
    if (*s == -1)
    {
        if (!t->id) t->id = id, t->fl = 1;
        same[t->id].push_back(id);
        head[id] = t->id;
        return;
    }
    if (!t->n.count(*s)) t->n[*s] = newNode();
    expand(t->n[*s], s + 1, id);
}

node* next (node *t, int w)
{
    while (t != root && !t->n.count(w)) t = t->f;
    return t->n.count(w) ? t->n[w] : t;
}

void build_fail ()
{
    queue<node*> que;
    que.push(root); root->f = root;
    while (!que.empty())
    {
        node *t = que.front(); que.pop();
        for_(it, (t->n))
        {
            node *nt = (t == root) ? t : next(t->f, it->first);
            it->second->f = nt; it->second->fl |= nt->fl;
            que.push(it->second);
        }
    }
}
```

Compressed Trie

```
struct node {
    int cnt, len;
    const char *s;
    node *sib, *pos;
    node () { ++__c; }
    node (int c, int l, const char *ss) {
        cnt = c; len = l; s = ss; sib = pos = 0; ++__c;
    }
} *root;

inline void addString (const char *s, int len) {
    node *r = root, *nr;
    for (int j = 0, k; j < len; ) {
    //  printf("%d\n", j);
        for (nr = r->pos; nr && nr->s[0] != s[j]; nr = nr->sib) ;
        if (nr) {
            for (k = 0; k < nr->len && j < len && nr->s[k] == s[j]; ++k, ++j) ;
            if (k == nr->len) { ++nr->cnt; r = nr; continue; }
            node *lc = new node(nr->cnt, nr->len - k, nr->s + k);
            ++nr->cnt;
            nr->len = k;
            lc->pos = nr->pos;
            nr->pos = lc; r = nr;
        }
        if (j != len) {
            node *p = new node(1, len - j, s + j);
            p->sib = r->pos; r->pos = p;
        }
        break;
    }
}
```

Hash

```
const int N = 1000050, P = 31;

int hval[N], power[N], str[N], n;

int gethash (int l, int r) {
    return hval[r] - hval[l - 1] * power[r - l + 1];
}

void init ()
{
    srand(123);
    power[0] = 1;
    for (int i = 1; i <= n; i++) power[i] = power[i - 1] * P;
    hval[1] = str[1];
    for (int i = 2; i <= n; i++) hval[i] = hval[i - 1] * P + str[i];
}

int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i)
        scanf("%d", str + i);
    init();
    int m, a, b;
    for (scanf("%d", &m); m--; )
    {
        scanf("%d%d", &a, &b);
        printf("%d\n", gethash(a, b));
    }
}
```

Manacher Algorithm

```
int rad[N];

int main ()
{
    int m, n(0);
    scanf("%d\n", &m);
    str[++n] = '$';
    str[++n] = '#';
    while (m--)
    {
        char op; scanf("%c", &op);
        str[++n] = op;
        str[++n] = '#';
    } str[++n] = '%';
    int mx = 2, id = 1; rad[1] = 1;
    for (int i = 2; i <= n; ++i)
    {
        rad[i] = min(rad[id * 2 - i], mx - i);
        for (; i + rad[i] <= n && str[i + rad[i]] == str[i - rad[i]];
                ++rad[i]) ;
        if (rad[i] + i > mx) mx = rad[i] + i, id = i;
    }
}
```

Suffix Array (DA)

```
namespace SA
{

    int r[N], sa[N], rank[N], height[N];
    int wa[N], wb[N], ws[N], wv[N];
    inline int cmp (int *r, int x, int y, int l) {
        return r[x] == r[y] && r[x + l] == r[y + l];
    }
    void da (int *r, int n, int m)
    {
        int i, j, p, *x = wa, *y = wb;
        for (i = 0; i < m; ++i) ws[i] = 0;
        for (i = 0; i < n; ++i) ws[x[i] = r[i]]++;
        for (i = 1; i < m; ++i) ws[i] += ws[i - 1];
        //downto is for stability concern when elements duplicate
        for (i = n - 1; i >= 0; --i) sa[--ws[x[i]]] = i;
        for (j = 1, p = 1; p < n; j <<= 1, m = p)
        {
            //sort by (rank[i],rank[i+j])
            //second
            //second == 0, in arbitary order
            for (i = n - j, p = 0; i < n; ++i) y[p++] = i;
            // sa[i]-j : (*,i)
            for (i = 0; i < n; ++i) if (sa[i] >= j) y[p++] = sa[i] - j;
            //first
            for (i = 0; i < n; ++i) wv[i] = x[y[i]];
            for (i = 0; i < m; ++i) ws[i] = 0;
            for (i = 0; i < n; ++i) ws[wv[i]]++;
            for (i = 1; i < m; ++i) ws[i] += ws[i - 1];
            //rvalue is not i : stability
            for (i = n - 1; i >= 0; --i) sa[--ws[wv[i]]] = y[i];
            //store the current rank in x
            for (swap(x, y), p = i = 1, x[sa[0]] = 0; i < n; ++i)
                x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
        }
    }
    // LCP(i, SA[Rank[i] - 1]) >= LCP(i - 1, SA[Rank[i - 1] - 1]) - 1
    void calheight (int *r, int n)
    {
        int i, j, k = 0;
        for (i = 1; i <= n; ++i) rank[sa[i]] = i;
        for (i = 0; i < n; height[rank[i++]] = k)
            for (k ? --k : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; ++k) ;
    }

    struct SparseTable
    {
        static const int lgN = 22;
        int GP[lgN][N], n, lg[N];
        SparseTable ()
        {
            lg[1] = 0;
            for (int i = 2, j = 0; i < N; ++i)
                lg[i] = (j += ((1 << (j + 1)) == i));
        }
        void init (int S[], int n)
        {
            this->n = n;
            for (int j = 0; j < n; ++j)
                GP[0][j] = S[j];

            for (int i = 1; i <= lg[n]; ++i)
                for (int j = 0; j + (1 << i) <= n; ++j)
                    GP[i][j] = min(GP[i - 1][j], GP[i - 1][j + (1 << (i - 1))]);
        }
        inline int query (int l, int r)
        {
            assert(l >= 0 && l <= r && r < n);

            int z = lg[r - l + 1];
            int ret = min(GP[z][l], GP[z][r - (1 << z) + 1]);
            return ret;
        }
    } st;

    int n;
    // S[0 .. n)
    inline void init (int S[], int n)
    {
        SA::n = n;
        S[n] = 0;
        da(S, n + 1, 256);
        calheight(S, n);
        st.init(height, n + 1);
    }
    inline int LCP (int i, int j)
    {
        if (i == j) return n - i;
        int l = rank[i], r = rank[j];
        if (l > r) swap(l, r);
        return st.query(l + 1, r);
    }
}
```

Suffix Array (Hash)

```
const ULL F = 137;

ULL pow[N], hval[N];
char dt[N];
int sa[N], height[N], n;

inline ULL get_hash (int l, int r) {
    return hval[r] - hval[l - 1] * pow[r - l + 1];
}

int cmp_suffix (int a, int b)
{
    int cp = 0;
    for (int i = 22; i >= 0; --i)
        if (a + cp + (1 << i) <= n && b + cp + (1 << i) <= n &&
          get_hash(a, a + cp + (1 << i) - 1) ==
          get_hash(b, b + cp + (1 << i) - 1)) cp += 1 << i;
    return dt[a + cp] < dt[b + cp];
}

void SA ()
{
    pow[0] = 1; for (int i = 1; i < n; ++i) pow[i] = pow[i - 1] * F;
    for (int i = 0; i < n; ++i) hval[i] = hval[i - 1] * F + dt[i];
    for (int i = 0; i < n; ++i) sa[i] = i;
    stable_sort(sa, sa + n, cmp_suffix);
    for (int i = 1; i < n; ++i)
    {
        int a = sa[i - 1], b = sa[i], cp = 0;
        for (int i = 22; i >= 0; --i)
            if (a + cp + (1 << i) <= n && b + cp + (1 << i) <= n &&
              get_hash(a, a + cp + (1 << i) - 1) ==
              get_hash(b, b + cp + (1 << i) - 1)) cp += 1 << i;
        height[i] = cp;
    }
}
```

Suffix Automaton

```
const int SIGMA = 26, N = 500000;
struct node { node *f, *n[26]; int l, res, cur; }
  buff[N], *cur = buff, *rt, *last;
node *list[N];

//expand (ch) : add a new char ch to the string being maintained.
void expand (int ch)
{
    node *p = last, *q = cur++;
    q->l = q->res = last->l + 1, last = q;
    for (; p && !p->n[ch]; p = p->f) p->n[ch] = q;
    if (!p) q->f = rt;
    else // remove the conditional statements with 'q->f = p->n[ch]' and get a
         // Factor Oracle.
        if (p->n[ch]->l == p->l + 1) q->f = p->n[ch];
        else
        {
            node *u = p->n[ch], *v = cur++;
            memcpy(v->n, u->n, sizeof(u->n)); v->f = u->f;
            v->l = v->res = p->l + 1, q->f = u->f = v;
            for (; p && p->n[ch] == u; p = p->f) p->n[ch] = v;
        }
}
```

Tree

Edge Decomposition

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

const int MXN = 300500, MXD = 20, inf = 100000000;
struct edge { int u, v, w, rev; edge *next, *dual; }
    ebuff[MXN + MXN], *cur = ebuff, *e[MXN];
```

```
inline void add_edge (int u, int v, int w)
{
    *cur = (edge){u, v, w, 0, e[u], cur + 1}; e[u] = cur++;
    *cur = (edge){v, u, w, 0, e[v], cur - 1}; e[v] = cur++;
}

int diff, dis[MXN][MXD], root[MXN][MXD], stot, crt;
int log2[MXN];
int rid[MXN * 5], mxdis[MXN * 5], cwhite[MXN * 5], BITBuff[MXN * MXD];
edge *midedge[MXN * 5], *me;
bool white[MXN];
int *BIT[MXN * 5], *Bcur = BITBuff;
int tstamp[MXN], curstamp;

int det_edge (int u)
{
    int size(1), tm; tstamp[u] = curstamp;
    for (edge *i = e[u]; i; i = i->next)
        if (!i->rev && tstamp[i->v] != curstamp)
        {
            tm = det_edge(i->v);
            size += tm;
            if (abs(stot - tm - tm) < diff)
                diff = abs(stot - tm - tm), me = i;
        }
    return size;
}

int maxdis;
inline int update (int u, int au, int di, int de)
{
    int size(1); maxdis = max(maxdis, di);
    tstamp[u] = curstamp;
    dis[u][de] = di; root[u][de] = crt;
    for (edge *i = e[u]; i; i = i->next)
        if (!i->rev && tstamp[i->v] != curstamp)
            size += update(i->v, u, di + i->w, de);
    return size;
}
void divide (int u, int d)
{
    if (stot == 1) return;
    edge *tm = new edge, *be;
    int s1, s2;
    diff = inf; ++curstamp; det_edge(u);
    me->rev = me->dual->rev = 1;
    be = me;

    *tm = (edge){crt + 1, crt + 2, me->w, 0, 0x0, 0x0};
    rid[++crt] = be->u; maxdis = -inf; curstamp++;
    s1 = update(be->u, -1, 1, d); mxdis[crt] = maxdis,
    BIT[crt] = Bcur; Bcur += maxdis + 1; midedge[crt] = tm;
    rid[++crt] = be->v; maxdis = -inf;
    s2 = update(be->v, -1, 1, d); mxdis[crt] = maxdis,
    BIT[crt] = Bcur; Bcur += maxdis + 1; midedge[crt] = tm;

    stot = s1; divide(be->u, d + 1);
    stot = s2; divide(be->v, d + 1);
}

inline void print_edge (int u)
{
    for (edge *i = e[u]; i; i = i->next)
        fprintf(stderr, "%d %d %d %d\n", i->u, i->v, i->w, i->rev);
}

int Q[MXN], w[MXN], cv, vis[MXN];
inline void adjust (int u, int au)
{ vis[u] = true;
    for (edge *i = e[u]; i; i = i->next)
        if (i->v != au) adjust(i->v, u);
    int l(1), r(1);
    for (edge *i = e[u]; i; i = i->next)
        if (i->v != au) { ++r; Q[r] = i->v, w[r] = 1; }
    vis[u] = false;
    if (r - l <= 2) return;
    for (edge *i = e[u]; i; i = i->next)
        if (!vis[i->v]) i->rev = i->dual->rev = 1;
    while (r - l > 2)
    {
        ++l; add_edge(cv, Q[l], w[l]);
        ++l; add_edge(cv, Q[l], w[l]);
        ++r; Q[r] = cv++, w[r] = 0;
    }
    while (l < r) { ++l; add_edge(u, Q[l], w[l]); }
    //print_edge(u);
}

#define low(X) (X&(-X))
inline int getKth (int *B, int N, int K)
{
    int p(0), re(0);
    for (int i = log2[N - 1]; i >= 0; --i)
    {
        p += 1 << i;
        if (p > N || re + B[p] >= K)
            p -= 1 << i;
        else re += B[p];
    }
    return p + 1;
}
inline void modify (int *B, int N, int i, int delta)
{ for (; i <= N; i += low(i)) B[i] += delta; }

inline int query (int u)
{
    if (white[u]) return 0;
    int res(inf), a, b, tmp;
    for (int i = 0; i < 20; ++i)
    {
        if ((b = root[u][i]) == 0) break;
        a = (b == midedge[b]->u) ? midedge[b]->v : midedge[b]->u;
        if (!cwhite[a]) continue;
        tmp = midedge[b]->w + dis[u][i] + getKth(BIT[a], mxdis[a], 1) - 2;
        if (tmp < res) res = tmp;
    }
    return res == inf ? -1 : res;
}

inline void blacken (int u)
{
    int b; white[u] = false;
    for (int i = 0; i < 20; ++i)
    {
        if ((b = root[u][i]) == 0) break;
        --cwhite[b];
        modify(BIT[b], mxdis[b], dis[u][i], -1);
    }
}
inline void whiten (int u)
{
    int b; white[u] = true;
    for (int i = 0; i < 20; ++i)
    {
        if ((b = root[u][i]) == 0) break;
        ++cwhite[b];
        modify(BIT[b], mxdis[b], dis[u][i], +1);
    }
}
inline void flip (int u)
  { if (white[u]) blacken(u); else whiten(u); }

int main ()
{
    int N, tm(0);
#ifdef LOCAL
    freopen("qtree5.in", "r", stdin);
#endif
    scanf("%d", &N);
    for (int i = 1; i <= 300000; ++i)
    {
        if ((1 << tm) < i) ++tm;
        log2[i] = tm;
    }
    for (int i = 1; i < N; ++i)
    {
        int a, b; scanf("%d%d", &a, &b);
        add_edge(a, b, 1);
    }
    cv = N + 1; adjust(1, -1);
    stot = cv - 1; divide(1, 0);
    scanf("%d", &tm); while (tm--)
    {
        int a, b; scanf("%d%d", &a, &b);
        if (!a) flip(b);
        else printf("%d\n", query(b));
    }
    return 0;
```

```
}
```

Hash (Tree)

```
const int N = 1000050, a = 7, p = 97, b = 31, P = 42961211;

set<int> cnt[N];
char buff[N << 2], *cur = buff;

int getHash (int &rdep)
{
    int res, tm, hv[3], deg(0);
    rdep = 0;
    while (*cur && *cur++ == '(') {
        hv[deg++] = getHash(tm);
        if (tm + 1 > rdep) rdep = tm + 1;
    }
    sort(hv, hv + deg);
    res = a;
    for (int i = 0; i < deg; ++i) res = (res * p ^ hv[i]) % P;
    res = res * b % P;
    cnt[rdep].insert(res);
    return res;
}
```

Heavy-Light Decomposition

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

const int N = 250050;

namespace CLR {
    int tID[N], chain[N], ccnt;
    int head[N];
}

namespace segTree {
```

```
struct node {
    int l, r, m, rs;
    node *lc, *rc;
} tbf[N << 2], *tc = tbf, *rt[N];
inline node* newNode (int l, int r) {
    *tc = (node){l, r, (l + r) >> 1, r - l + 1, 0, 0};
    return tc ++;
}
void build (node *x)
{
    if (x->l == x->r) return;
    build(x->lc = newNode(x->l, x->m));
    build(x->rc = newNode(x->m + 1, x->r));
}
int query (node *x, int l, int r)
{
    if (l <= x->l && r >= x->r) return x->rs;
    int res(0);
    if (l <= x->m) res += query(x->lc, l, r);
    if (r > x->m) res += query(x->rc, l, r);
    return res;
}
void modify (node *x, int l, int r)
{
    if (l <= x->l && r >= x->r) { x->rs = 0; return; }
    if (l <= x->m) modify(x->lc, l, r);
    if (r > x->m) modify(x->rc, l, r);
    x->rs = x->lc->rs + x->rc->rs;
}

using namespace CLR;

inline int queryTree (int u, int v)
{
    node *r = rt[chain[u]];
    if (tID[u] > tID[v]) swap(u, v);
    return query(r, tID[u], tID[v]);
}
inline void modifyTree (int u, int v)
{
    node *r = rt[chain[u]];
    if (tID[u] > tID[v]) swap(u, v);
```

```
        modify(r, tID[u], tID[v]);
    }

    inline void buildChain (int *a, int cnt)
    {
        build(rt[++ ccnt] = newNode(1, cnt));
        for (int i = 1; i <= cnt; ++i)
            chain[a[i]] = ccnt, tID[a[i]] = i;
    }
}

namespace tree {
    using namespace CLR;

    struct edge { int t; edge *n; }
    ebf[N << 1], *ecr = ebf, *e[N];
    inline void link (int s, int t) {
        *ecr = (edge){t, e[s]}; e[s] = ecr++;
    }
#define forEdges(it,u) for(edge*it=e[u];it;it=it->n)

    int dep[N], pre[N], next[N], size[N], status[N];

    void init (int n)
    {
        static int que[N];
        int l(0), r(0), x;
        que[++r] = 1; dep[1] = 1;
        while (l < r)
            forEdges(it, x = que[++l])
                if (it->t != pre[x])
                    pre[que[++r] = it->t] = x,
                        dep[it->t] = dep[x] + 1;
        for (int i = r; i > 0; --i)
            x = que[i], size[pre[x]] += ++size[x];
        for (int i = 1; i <= r; ++i)
        {
            int maxSize(0), id(0);
            forEdges(it, x = que[i]) if (it->t != pre[x])
                if (size[it->t] > maxSize)
                    maxSize = size[id = it->t];
            next[x] = id;
        }
        for (int i = 1; i <= n; ++i) if (next[pre[i]] != i)
        {
            int tc(0);
            for (int j = i; j; j = next[j])
                que[++tc] = j, head[j] = i;
            segTree::buildChain(que, tc);
        }
    }

    int query (int u)
    {
        int res(0);
        while (chain[u] != chain[1])
            res += segTree::queryTree(u, head[u]),
                u = pre[head[u]];
        if (u != 1)
            res += segTree::queryTree(next[1], u);
        return res;
    }

    void modify (int u, int v)
    {
        while (chain[u] != chain[v])
            if (dep[pre[head[u]]] < dep[pre[head[v]]])
                segTree::modifyTree(v, head[v]),
                    v = pre[head[v]];
            else
                segTree::modifyTree(u, head[u]),
                    u = pre[head[u]];
        if (dep[u] != dep[v])
        {
            if (dep[u] > dep[v]) swap(u, v);
            segTree::modifyTree(next[u], v);
        }
    }
}

int main ()
{
    using namespace tree;
    //freopen("in", "r", stdin);
    int n, a, b;
    scanf("%d", &n);
    for (int i = 1; i < n; ++i)
    {
        scanf("%d%d", &a, &b);
        link(a, b); link(b, a);
    }
    init(n);
    int m; scanf("%d", &m);
    m += n - 1;
    while (m--)
    {
        char op[20]; scanf("%s", op);
        if (op[0] == 'W')
        {
            scanf("%d", &a);
            printf("%d\n", query(a));
        }
        else
        {
            scanf("%d%d", &a, &b);
            modify(a, b);
        }
    }
    return 0;
}
```

LCA

```
void dfs (int x, int ax, int d)
{
    in[x] = ++ls; de[x] = d;
    ac[x][0] = ax;
    for (int k = 1; ac[x][k - 1]; ++k)
        ac[x][k] = ac[ac[x][k - 1]][k - 1];
    for (edge *it = e[x]; it; it = it->n)
        if (it->t != ax) dfs(it->t, x, d + 1);
    out[x] = ++ls;
}

int lg[Nm];
```

```
void lift (int &x, int d) {
    for (; de[x] > d; x = ac[x][lg[de[x] - d]]) ;
}
int LCA (int x, int y)
{
    if (de[x] < de[y]) lift(y, de[x]); else lift(x, de[y]);
    while (x != y)
    {
        int d;
        for (d = 0; ac[x][d] != ac[y][d]; ++d) ;
        d -= bool(d); x = ac[x][d], y = ac[y][d];
    }
    return x;
}
```

Link Cut Tree

```
/*
 * Suggested version.
 * */

struct node {
    static node *nil;
    node *f, *c[2];
    LL w, delt, maxw;
    int rev, cnt;
    inline void update () {
        assert(!delt);
        cnt = c[0]->cnt + c[1]->cnt + 1;
        maxw = max(max(c[0]->maxw, c[1]->maxw), w);
    }
    inline void inc (LL d) {
        if (this == nil) return;
        delt += d; maxw += d; w += d;
    }
    inline void flipRev () {
        if (this == nil) return;
        rev ^= 1; swap(c[0], c[1]);
    }
    inline void push () {
        if (rev) c[0]->flipRev(), c[1]->flipRev(), rev = 0;
        if (delt) c[0]->inc(delt), c[1]->inc(delt), delt = 0;
    }
    inline void sc (node *z, int w) { c[w] = z; z->f = this; }
    inline int ty () { return this == f->c[1]; }
    inline int rt () { return !(this == f->c[0] || this == f->c[1]); }
    inline void zig () {
        node *y = this->f; y->push(); push();
        int w = ty();
        f = y->f; if (!y->rt()) y->f->c[y->ty()] = this;
        y->sc(c[!w], w); sc(y, !w);
        y->update();
    }
    void init (LL _w) {
        c[0] = c[1] = f = nil;
        w = maxw = _w; cnt = 1; delt = 0; rev = 0;
    }
    static void init () {
        nil->f = nil->c[0] = nil->c[1] = nil;
        nil->w = nil->maxw = -inf;
    }
} T[N], *node::nil = &T[0];

inline void splay (node *x) { // zig only when reverse tag is added
    for (x->push(); !x->rt(); x->zig()) ; x->update();
}
inline void access (node *x) {
    for (node *y(x), *z(node::nil); y != node::nil; z = y, y = y->f)
        splay(y), y->sc(z, 1), y->update();
}
inline void increase (node *a, node *b, LL d) {
    access(a);
    for (node *y(b), *z(node::nil); y != node::nil; z = y, y = y->f) {
        splay(y);
        if (y->f == node::nil) {
            y->w += d; y->update();
            y->c[1]->inc(d); z->inc(d);
        }
        y->sc(z, 1); y->update();
    }
}
inline LL getMax (node *a, node *b) {
    access(a);
    LL res = 0;
    for (node *y(b), *z(node::nil); y != node::nil; z = y, y = y->f) {
        splay(y);
        if (y->f == node::nil) res = max(max(y->c[1]->maxw, z->maxw), y->w);
        y->sc(z, 1); y->update();
    }
    return res;
}
inline void evert (node *x) {
    access(x); splay(x); x->flipRev();
}
inline node* getRoot (node *x) {
    access(x); splay(x);
    for (; x->push(), x->c[0] != node::nil; x = x->c[0]) ;
    splay(x); return x;
}
inline node* getPre (node *x) {
    access(x); splay(x);
    for (x = x->c[0]; x->push(), x->c[1] != node::nil; x = x->c[1]) ;
    splay(x); return x;
}
inline void link (node *a, node *b) {
    evert(a);
    assert(a->f == node::nil);
    a->f = b; access(a);
}
inline void cut (node *a, node *b) {
    // cut b and its father in the tree rooted by a
    evert(a);
    access(b); splay(b);
    b->c[0]->f = node::nil; b->c[0] = node::nil; b->update();
}
```

Node Decomposition

```
#include <cstdio>
#include <cstring>
#include <vector>
using namespace std;

typedef long long lint;
```

```
const int N = 100050, M = 2000500;

struct edge { int t, w; edge *n; }
ebf[N << 2], *ec = ebf, *e[N];
#define forEdges(I,U) for(edge*I=e[U];I;I=I->n)
inline void addEdge (int u, int v, int w)
{
    *ec = (edge){v, w, e[u]}; e[u] = ec++;
    *ec = (edge){u, w, e[v]}; e[v] = ec++;
}

int root[N], size[N], que[M], dep[N], fa[N], n;
lint dis[N];
vector<lint> maxDis[M];
vector<pair<int, int> > child[M];
int start[N], L, U, f, r;
int gravity, cid;

void getGravity (int rt, int ID)
{
    int x, y, z;
    for (int i = r; i > 0; --i) size[que[i]] = 1;
    for (int i = r; i > 0; --i)
        size[fa[que[i]]] += size[que[i]];
    int mn(N), cur, id(-1);
    for (int i = r; i > 0; --i)
    {
        cur = 0;
        forEdges(it, x = que[i])
            if (root[y = it->t] == rt && y != fa[x])
                cur = max(cur, size[y]);
        cur = max(cur, size[rt] - size[x]);
        if (cur < mn) mn = cur, id = x;
    }
    if ((gravity = id) == -1) return;
    forEdges(it, gravity) if (root[it->t] == rt)
        child[ID].push_back(make_pair(it->t, it->w));
    root[gravity] = gravity;
    forEdges(it, gravity) if (root[it->t] == rt)
    {
        root[x = it->t] = x;
        f = r = 0; que[++r] = x; fa[x] = 0;
        while (f < r)
            forEdges(i, y = que[++f])
                if (root[z = i->t] == rt && z != fa[y])
                    fa[z] = y, root[z] = x, que[++r] = z;
    }
}

void calc (int rt, vector<lint> &res)
{
    int x, y; f = r = 0;
    que[++r] = rt; dep[rt] = 1, dis[rt] = fa[rt] = 0;
    while (f < r)
        forEdges(it, x = que[++f])
            if (root[y = it->t] == rt && y != fa[x])
            {
                dep[y] = dep[x] + 1, dis[y] = dis[x] + it->w;
                que[++r] = y; fa[y] = x;
            }
    x = 0;
    for (int i = r; i > 0; --i) if (dep[que[i]] > x) x = dep[que[i]];
    res.resize(x + 1);
    for (int i = r; i > 0; --i)
        res[dep[que[i]]] = max(res[dep[que[i]]], dis[que[i]]);
}

int solve (int x)
{
    int r = ++cid;
    calc(x, maxDis[r]);
    getGravity(x, r);
    if (gravity == -1) return r;
    for (int i = 0; i < child[r].size(); ++i)
    {
        int id = (child[r][i].first = solve(child[r][i].first));
        for (int j = 0; j < maxDis[id].size(); ++j)
            maxDis[id][j] += child[r][i].second;
    }
    return r;
}

bool getMax (vector<lint> &x,
        vector<lint> &y, double delt)
```

```
{
    int i;
    for (f = 1, r = 1, i = 1; i < y.size(); ++i)
    {
        if (L - i >= 1 && L - i < x.size())
        {
            for (; f < r; --r)
                if (x[L - i] - delt * (L - i) < x[que[r]] - delt * que[r]) break;
            que[++r] = L - i;
        }
        while (f < r && que[f + 1] + i > U) ++f;
        if (f < r && x[que[f + 1]] + y[i] >= delt * (que[f + 1] + i)) return true;
    }
    return false;
}

vector<lint> tmp;
bool check (double delt)
{
    for (int i = 1; i <= cid; ++i)
        for (int j = 1; j < maxDis[i].size(); ++j)
            if ((j - 1) >= L && (j - 1) <= U && (maxDis[i][j] - maxDis[i][0]) >= (j - 1) * delt)
                return true;
    for (int i = 1; i <= cid; ++i)
    {
        if (child[i].empty()) continue;
        tmp = maxDis[child[i][0].first];
        for (int j = 1; j < child[i].size(); ++j)
        {
            __typeof(tmp) &cur = maxDis[child[i][j].first];
            if (getMax(tmp, cur, delt) || getMax(cur, tmp, delt)) return true;
            for (int t = 0; t < tmp.size() && t < cur.size(); ++t)
                tmp[t] = max(tmp[t], cur[t]);
            for (int t = tmp.size(); t < cur.size(); ++t) tmp.push_back(cur[t]);
        }
    }
    return 0;
}

int main ()
{
    scanf("%d%d%d", &n, &L, &U);
    double s(0.0), e(0.0);
    for (int i = 1; i < n; ++i)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        addEdge(a, b, c);
        if (c > e) e = c;
    }
    fill(root + 1, root + n + 1, 1);
    solve(1);
    while (e - s > 1e-5)
    {
        double m = (e + s) / 2;
        if (check(m)) s = m; else e = m;
    }
    printf("%.3lf\n", s);
    return 0;
}
```

Unclassified

Simulated Annealing

```
const int N = 32, T = 1000;

//...

struct state { double a, b; } cur[N];

double P, Q, R;
inline double evaluate (state &s) { } // evalutae the state
inline bool legal (state &s) { } // if the state is legal
inline double rand1 () { return (double(rand() * 2) / RAND_MAX) - 1; }
inline double rand2 () { return double(rand()) / RAND_MAX; }

int main ()
{
    // Init
    for (int i = 0; i < N; ++i)
    {
        cur[i] = (state){rand2(), rand2()};
        res = min(res, evaluate(cur[i]));
    }
    for (double e = 1.0; e > 1e-10; e /= 2.0)
    {
        for (int i = 0; i < N; ++i)
        {
            double cv = evaluate(cur[i]), cr;
            for (int t = T; t > 0; --t)
            {
                state tmp = cur[i];
                tmp.a += rand1() * e, tmp.b += rand1() * e;
                if (legal(tmp))
                    if ((cr = evaluate(tmp)) < cv)
                        cv = cr, cur[i] = tmp;
                    else if (rand2() < 1e-4) cur[i] = tmp;
            }
            res = min(res, cv);
        }
    }
    printf("%.2lf\n", res);
    return 0;
}
```