

Contents

1 First Thing First

1.1	Header	1
1.2	55kai	1

2 Data Structure

2.1	RMQ	1
2.2	Segment Tree Beats	1
2.3	Segment Tree	2
2.4	K-D Tree	2
2.5	STL+	3
2.6	BIT	3
2.7	Trie	3
2.8	Treap	3
2.9	Cartesian Tree	4
2.10	LCT	5
2.11	Mo's Algorithm On Tree	5
2.12	CDQ's Divide and Conquer	5
2.13	Persistent Segment Tree	6
2.14	Persistent Union Find	6

3 Math

3.1	Multiplication, Powers	7
3.2	Matrix Power	7
3.3	Sieve	7
3.4	Prime Test	8
3.5	Pollard-Rho	8
3.6	Berlekamp-Massey	8
3.7	Extended Euclidean	8
3.8	Inverse	8
3.9	Binomial Numbers	8
3.10	NTT, FFT, FWT	9
3.11	Simpson's Numerical Integration	9
3.12	Gauss Elimination	9
3.13	Factor Decomposition	9
3.14	Primitive Root	9
3.15	Quadratic Residue	9
3.16	Chinese Remainder Theorem	10
3.17	Bernoulli Numbers	10
3.18	Simplex Method	10
3.19	BSGS	10

4 Graph Theory

4.1	LCA	10
4.2	Maximum Flow	10
4.3	Minimum Cost Maximum Flow	11
4.4	Path Intersection on Trees	11
4.5	Centroid Decomposition (Divide-Conquer)	11
4.6	Heavy-light Decomposition	12
4.7	Bipartite Matching	12
4.8	Virtual Tree	12
4.9	Euler Tour	12
4.10	SCC, 2-SAT	13
4.11	Topological Sort	13
4.12	General Matching	13
4.13	Tarjan	13
4.14	Bi-connected Components, Block-cut Tree	14
4.15	Minimum Directed Spanning Tree	14
4.16	Cycles	14
4.17	Dominator Tree	14
4.18	Global Minimum Cut	15

5 Geometry

5.1	2D Basics	15
5.2	Polar angle sort	15
5.3	Segments, lines	15
5.4	Polygons	15
5.5	Half-plane intersection	16

5.6	Circles	16
5.7	Circle Union	16
5.8	Minimum Covering Circle	17
5.9	Circle Inversion	17
5.10	3D Basics	17
5.11	3D Line, Face	18
5.12	3D Convex	18

6 String

6.1	Aho-Corasick Automation	1
6.2	Hash	1
6.3	KMP	1
6.4	Manacher	1
6.5	Palindrome Automation	1
6.6	Suffix Array	1
6.7	Suffix Automation	1

7 Miscellaneous

7.1	Date	21
7.2	Subset Enumeration	21
7.3	Digit DP	21
7.4	Simulated Annealing	21

1 First Thing First

1.1 Header

```
#include <bits/stdc++.h>
using namespace std;
using LL = long long;
#define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i < _##i; ++i)
#define FORD(i, x, y) for (decay<decltype(x)>::type i = (x), _##i = (y); i > _##i; --i)
#ifdef zero1
#define dbg(x...) do { cout << "\033[32;1m" << #x << " -> "; err(x); } while (0)
void err() { cout << "\033[39;0m" << endl; }
template<typename...> class T, typename t, typename... A>
void err(T<t> a, A... x) { for (auto v: a) cout << v << ' '; err(x...); }
template<typename T, typename... A>
void err(T a, A... x) { cout << a << ' '; err(x...); }
#else
#define dbg(...)
#endif
// for CLion
// set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O2 -Ddebug -Wall -DULTMASTER -DkBlack -Dzerol")
// for University of Porto
// setxkbmap us
```

1.2 55kai

```
inline char nc() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2) ? EOF : *p1++;
}
template <typename T>
bool rn(T& v) {
    static char ch;
    while (ch != EOF && !isdigit(ch)) ch = nc();
    if (ch == EOF) return false;
    for (v = 0; isdigit(ch); ch = nc())
        v = v * 10 + ch - '0';
    return true;
}
template <typename T>
void o(T p) {
    static int stk[70], tp;
    if (p == 0) { putchar('0'); return; }
    if (p < 0) { p = -p; putchar('-'); }
    while (p) stk[++tp] = p % 10, p /= 10;
    while (tp) putchar(stk[tp--] + '0');
}
```

```
if (p == 0) { putchar('0'); return; }
if (p < 0) { p = -p; putchar('-'); }
while (p) stk[++tp] = p % 10, p /= 10;
while (tp) putchar(stk[tp--] + '0');
}

2 Data Structure

2.1 RMQ

int f[maxn][maxn][10][10];
inline int highbit(int x) { return 31 - __builtin_clz(x); }
inline int calc(int x, int y, int xx, int yy, int p, int q) {
    return max(
        max(f[x][y][p][q], f[xx - (1 << p) + 1][yy - (1 << q) + 1][p][q]),
        max(f[xx - (1 << p) + 1][y][p][q], f[x][yy - (1 << q) + 1][p][q])
    );
}
void init() {
    FOR (x, 0, highbit(n) + 1)
        FOR (y, 0, highbit(m) + 1)
            FOR (i, 0, n - (1 << x) + 1)
                FOR (j, 0, m - (1 << y) + 1) {
                    if (!x && !y) { f[i][j][x][y] = a[i][j]; continue; }
                    f[i][j][x][y] = calc(
                        i, j,
                        i + (1 << x) - 1, j + (1 << y) - 1,
                        max(x - 1, 0), max(y - 1, 0)
                    );
                }
}
inline int get_max(int x, int y, int xx, int yy) {
    return calc(x, y, xx, yy, highbit(xx - x + 1), highbit(yy - y + 1));
}

struct RMQ {
    int f[22][M];
    inline int highbit(int x) { return 31 - __builtin_clz(x); }
    void init(int* v, int n) {
        FOR (i, 0, n) f[0][i] = v[i];
        FOR (x, 1, highbit(n) + 1)
            FOR (i, 0, n - (1 << x) + 1)
                f[x][i] = min(f[x - 1][i], f[x - 1][i + (1 << (x - 1))]);
    }
    int get_min(int l, int r) {
        assert(l <= r);
        int t = highbit(r - l + 1);
        return min(f[t][l], f[t][r - (1 << t) + 1]);
    }
} rmq;
```

2.2 Segment Tree Beats

```
namespace R {
#define lson o * 2, l, (l + r) / 2
#define rson o * 2 + 1, (l + r) / 2 + 1, r
int m1[N], m2[N], cml[N];
LL sum[N];
void up(int o) {
    int lc = o * 2, rc = lc + 1;
    m1[o] = max(m1[lc], m1[rc]);
    sum[o] = sum[lc] + sum[rc];
    if (m1[lc] == m1[rc]) {
        cml[o] = cml[lc] + cml[rc];
        m2[o] = max(m2[lc], m2[rc]);
    } else {
        cml[o] = m1[lc] > m1[rc] ? cml[lc] : cml[rc];
        m2[o] = max(min(m1[lc], m1[rc]), max(m2[lc], m2[rc]));
    }
}
void mod(int o, int x) {
    if (x == 0) return;
    if (x < 0) {
        if (m1[o] < -x) m1[o] = -x;
        else m2[o] += x;
    } else {
        if (m2[o] < x) m2[o] = x;
        else m1[o] += x;
    }
    up(o);
}
```

```

    if (x >= m1[o]) return;
    assert(x > m2[o]);
    sum[o] -= 1LL * (m1[o] - x) * cml[o];
    m1[o] = x;
}
void down(int o) {
    int lc = o * 2, rc = lc + 1;
    mod(lc, m1[o]); mod(rc, m1[o]);
}
void build(int o, int l, int r) {
    if (l == r) { int t; read(t); sum[o] = m1[o] = t; m2[o] = -INF; cml[o] = 1; }
    else { build(lson); build(rson); up(o); }
}
void update(int ql, int qr, int x, int o, int l, int r) {
    if (r < ql || qr < l || m1[o] <= x) return;
    if (ql <= l && r <= qr && m2[o] < x) { mod(o, x); return; }
    down(o);
    update(ql, qr, x, lson); update(ql, qr, x, rson);
    up(o);
}
int qmax(int ql, int qr, int o, int l, int r) {
    if (r < ql || qr < l) return -INF;
    if (ql <= l && r <= qr) return m1[o];
    down(o);
    return max(qmax(ql, qr, lson), qmax(ql, qr, rson));
}
LL qsum(int ql, int qr, int o, int l, int r) {
    if (r < ql || qr < l) return 0;
    if (ql <= l && r <= qr) return sum[o];
    down(o);
    return qsum(ql, qr, lson) + qsum(ql, qr, rson);
}
}
}

```

2.3 Segment Tree

// set + add

```

struct IntervalTree {
#define ls o * 2, l, m
#define rs o * 2 + 1, m + 1, r
    static const LL M = maxn * 4, RS = 1E18 - 1;
    LL addv[M], setv[M], minv[M], maxv[M], sumv[M];
    void init() {
        memset(addv, 0, sizeof addv);
        fill(setv, setv + M, RS);
        memset(minv, 0, sizeof minv);
        memset(maxv, 0, sizeof maxv);
        memset(sumv, 0, sizeof sumv);
    }
    void maintain(LL o, LL l, LL r) {
        if (l < r) {
            LL lc = o * 2, rc = o * 2 + 1;
            sumv[o] = sumv[lc] + sumv[rc];
            minv[o] = min(minv[lc], minv[rc]);
            maxv[o] = max(maxv[lc], maxv[rc]);
        } else sumv[o] = minv[o] = maxv[o] = 0;
        if (setv[o] != RS) { minv[o] = maxv[o] = setv[o]; sumv[o] = setv[o] * (r - l + 1); }
        if (addv[o]) { minv[o] += addv[o]; maxv[o] += addv[o]; sumv[o] += addv[o] * (r - l + 1); }
    }
    void build(LL o, LL l, LL r) {
        if (l == r) addv[o] = a[l];
        else {
            LL m = (l + r) / 2;
            build(ls); build(rs);
        }
        maintain(o, l, r);
    }
    void pushdown(LL o) {
        LL lc = o * 2, rc = o * 2 + 1;
        if (setv[o] != RS) {
            setv[lc] = setv[rc] = setv[o];

```

```

            addv[lc] = addv[rc] = 0;
            setv[o] = RS;
        }
        if (addv[o]) {
            addv[lc] += addv[o]; addv[rc] += addv[o];
            addv[o] = 0;
        }
    }
    void update(LL p, LL q, LL o, LL l, LL r, LL v, LL op) {
        if (p <= r && l <= q)
            if (p <= l && r <= q) {
                if (op == 2) { setv[o] = v; addv[o] = 0; }
                else addv[o] += v;
            }
            else {
                pushdown(o);
                LL m = (l + r) / 2;
                update(p, q, ls, v, op); update(p, q, rs, v, op);
            }
        maintain(o, l, r);
    }
    void query(LL p, LL q, LL o, LL l, LL r, LL add, LL& ssum, LL& smin, LL& smax) {
        if (p > r || l > q) return;
        if (setv[o] != RS) {
            LL v = setv[o] + add + addv[o];
            ssum += v * (min(r, q) - max(l, p) + 1);
            smin = min(smin, v);
            smax = max(smax, v);
        }
        else if (p <= l && r <= q) {
            ssum += sumv[o] + add * (r - l + 1);
            smin = min(smin, minv[o] + add);
            smax = max(smax, maxv[o] + add);
        }
        else {
            LL m = (l + r) / 2;
            query(p, q, ls, add + addv[o], ssum, smin, smax);
            query(p, q, rs, add + addv[o], ssum, smin, smax);
        }
    }
}
}
// persistent
namespace tree {
#define mid ((l + r) >> 1)
#define lson ql, qr, l, mid
#define rson ql, qr, mid + 1, r
    struct P {
        LL add, sum;
        int ls, rs;
    } tr[maxn * 45 * 2];
    int sz = 1;
    int N(LL add, int l, int r, int ls, int rs) {
        tr[sz] = {add, tr[ls].sum + tr[rs].sum + add * (len[r] - len[l - 1]), ls, rs};
        return sz++;
    }
    int update(int o, int ql, int qr, int l, int r, LL add) {
        if (ql > r || l > qr) return o;
        const P& t = tr[o];
        if (ql <= l && r <= qr) return N(add + t.add, l, r, t.ls, t.rs);
        return N(t.add, l, r, update(t.ls, lson, add), update(t.rs, rson, add));
    }
    LL query(int o, int ql, int qr, int l, int r, LL add = 0) {
        if (ql > r || l > qr) return 0;
        const P& t = tr[o];
        if (ql <= l && r <= qr) return add * (len[r] - len[l - 1]) + t.sum;
        return query(t.ls, lson, add + t.add) + query(t.rs, rson, add + t.add);
    }
}
}

```

2.4 K-D Tree

```

// global variable pruning
// visit L/R with more potential
namespace kd {
const int K = 2, inf = 1E9, M = N;
const double lim = 0.7;
struct P {
    int d[K], l[K], r[K], sz, val;
    LL sum;
    P *ls, *rs;
    P *up() {
        sz = ls->sz + rs->sz + 1;
        sum = ls->sum + rs->sum + val;
        FOR (i, 0, K) {
            l[i] = min(d[i], min(ls->l[i], rs->l[i]));
            r[i] = max(d[i], max(ls->r[i], rs->r[i]));
        }
        return this;
    }
} pool[M], *null = new P, *pit = pool;
static P *tmp[M], **pt;
void init() {
    null->ls = null->rs = null;
    FOR (i, 0, K) null->l[i] = inf, null->r[i] = -inf;
    null->sum = null->val = 0;
    null->sz = 0;
}
P* build(P** l, P** r, int d = 0) { // [l, r)
    if (d == K) d = 0;
    if (l >= r) return null;
    P** m = l + (r - l) / 2; assert(l <= m && m < r);
    nth_element(l, m, r, [&](const P* a, const P* b) {
        return a->d[d] < b->d[d];
    });
    P* o = *m;
    o->ls = build(l, m, d + 1); o->rs = build(m + 1, r, d + 1);
    return o->up();
}
P* Build() {
    pt = tmp; FOR (it, pool, pit) *pt++ = it;
    return build(tmp, pt);
}
inline bool inside(int p[], int q[], int l[], int r[]) {
    FOR (i, 0, K) if (r[i] < q[i] || p[i] < l[i]) return false;
    return true;
}
LL query(P* o, int l[], int r[]) {
    if (o == null) return 0;
    FOR (i, 0, K) if (o->r[i] < l[i] || r[i] < o->l[i]) return 0;
    if (inside(o->l, o->r, l, r)) return o->sum;
    return query(o->ls, l, r) + query(o->rs, l, r) + (inside(o->d, o->d, l, r) ? o->val : 0);
}
void dfs(P* o) {
    if (o == null) return;
    *pt++ = o; dfs(o->ls); dfs(o->rs);
}
P* ins(P* o, P* x, int d = 0) {
    if (d == K) d = 0;
    if (o == null) return x->up();
    P& oo = x->d[d] <= o->d[d] ? o->ls : o->rs;
    if (oo->sz > o->sz * lim) {
        pt = tmp; dfs(o); *pt++ = x;
        return build(tmp, pt, d);
    }
    oo = ins(oo, x, d + 1);
    return o->up();
}
}
}

```

2.5 STL+

```
// priority_queue
// binary_heap_tag
// pairing_heap_tag: support editing
// thin_heap_tag: fast when increasing, can't join
#include<ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;

typedef __gnu_pbds::priority_queue<LL, less<LL>,
    pairing_heap_tag> PQ;
__gnu_pbds::priority_queue<int, cmp, pairing_heap_tag>::
    point_iterator it;
PQ pq, pq2;

int main() {
    auto it = pq.push(2);
    pq.push(3);
    assert(pq.top() == 3);
    pq.modify(it, 4);
    assert(pq.top() == 4);
    pq2.push(5);
    pq.join(pq2);
    assert(pq.top() == 5);
}

// BBT

// ov_tree_tag
// rb_tree_tag
// splay_tree_tag

// mapped: null_type or null_mapped_type (old) is null
// Node_Update should be tree_order_statistics_node_update to
// use find_by_order & order_of_key
// find_by_order: find the element with order+1 (0-based)
// order_of_key: number of elements lt r_key
// support join & split

#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using Tree = tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update>;
Tree t;

// Persistent BBT

#include <ext/rope>
using namespace __gnu_cxx;
rope<int> s;

int main() {
    FOR (i, 0, 5) s.push_back(i); // 0 1 2 3 4
    s.replace(1, 2, s); // 0 (0 1 2 3 4) 3 4
    auto ss = s.substr(2, 2); // 1 2
    s.erase(2, 2); // 0 1 4
    s.insert(2, s); // equal to s.replace(2, 0, s)
    assert(s[2] == s.at(2)); // 2
}

// Hash Table

#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/hash_policy.hpp>
using namespace __gnu_pbds;

gp_hash_table<int, int> mp;
cc_hash_table<int, int> mp;
```

2.6 BIT

```
namespace bit {
    LL c[M];
    inline int lowbit(int x) { return x & -x; }
    void add(int x, LL v) {
        for (; x < M; x += lowbit(x))
```

```
        c[x] += v;
    }
    LL sum(int x) {
        LL ret = 0;
        for (; x > 0; x -= lowbit(x))
            ret += c[x];
        return ret;
    }
    int kth(LL k) {
        int p = 0;
        for (int lim = 1 << 20; lim; lim /= 2)
            if (p + lim < M && c[p + lim] < k) {
                p += lim;
                k -= c[p];
            }
        return p + 1;
    }
}

namespace bit {
    int c[maxn], cc[maxn];
    inline int lowbit(int x) { return x & -x; }
    void add(int x, int v) {
        for (int i = x; i <= n; i += lowbit(i)) {
            c[i] += v; cc[i] += x * v;
        }
    }
    void add(int l, int r, int v) { add(l, v); add(r + 1, -v); }
    int sum(int x) {
        int ret = 0;
        for (int i = x; i > 0; i -= lowbit(i))
            ret += (x + 1) * c[i] - cc[i];
        return ret;
    }
    int sum(int l, int r) { return sum(r) - sum(l - 1); }
}

namespace bit {
    LL c[N], cc[N], ccc[N];
    inline LL lowbit(LL x) { return x & -x; }
    void add(LL x, LL v) {
        for (LL i = x; i < N; i += lowbit(i)) {
            c[i] = (c[i] + v) % MOD;
            cc[i] = (cc[i] + x * v) % MOD;
            ccc[i] = (ccc[i] + x * x % MOD * v) % MOD;
        }
    }
    void add(LL l, LL r, LL v) { add(l, v); add(r + 1, -v); }
    LL sum(LL x) {
        static LL INV2 = (MOD + 1) / 2;
        LL ret = 0;
        for (LL i = x; i > 0; i -= lowbit(i))
            ret += (x + 1) * (x + 2) % MOD * c[i] % MOD
                - (2 * x + 3) * cc[i] % MOD
                + ccc[i];
        return ret % MOD * INV2 % MOD;
    }
    LL sum(LL l, LL r) { return sum(r) - sum(l - 1); }
}
```

2.7 Trie

```
namespace trie {
    const int M = 31;
    int ch[N * M][2], sz;
    void init() { memset(ch, 0, sizeof ch); sz = 2; }
    void ins(LL x) {
        int u = 1;
        FOR (i, M, -1) {
            bool b = x & (1LL << i);
            if (!ch[u][b]) ch[u][b] = sz++;
            u = ch[u][b];
        }
    }
}

// persistent
// !!! sz = 1
```

```
struct P { int w, ls, rs; };
P tr[M] = {{0, 0, 0}};
int sz;

int _new(int w, int ls, int rs) { tr[sz] = {w, ls, rs}; return
    sz++; }
int ins(int oo, int v, int d = 30) {
    P& o = tr[oo];
    if (d == -1) return _new(o.w + 1, 0, 0);
    bool u = v & (1 << d);
    return _new(o.w + 1, u == 0 ? ins(o.ls, v, d - 1) : o.ls, u
        == 1 ? ins(o.rs, v, d - 1) : o.rs);
}
int query(int pp, int qq, int v, int d = 30) {
    if (d == -1) return 0;
    bool u = v & (1 << d);
    P& p = tr[pp], &q = tr[qq];
    int lw = tr[q.ls].w - tr[p.ls].w;
    int rw = tr[q.rs].w - tr[p.rs].w;

    int ret = 0;
    if (u == 0) {
        if (rw) { ret += 1 << d; ret += query(p.rs, q.rs, v, d -
            1); }
        else ret += query(p.ls, q.ls, v, d - 1);
    } else {
        if (lw) { ret += 1 << d; ret += query(p.ls, q.ls, v, d -
            1); }
        else ret += query(p.rs, q.rs, v, d - 1);
    }
    return ret;
}
```

2.8 Treap

```
// set
namespace treap {
    const int M = maxn * 17;
    extern struct P* const null;
    struct P {
        P *ls, *rs;
        int v, sz;
        unsigned rd;
        P(int v): ls(null), rs(null), v(v), sz(1), rd(rnd()) {}
        P(): sz(0) {}

        P* up() { sz = ls->sz + rs->sz + 1; return this; }
        int lower(int v) {
            if (this == null) return 0;
            return this->v >= v ? ls->lower(v) : rs->lower(v) +
                ls->sz + 1;
        }
        int upper(int v) {
            if (this == null) return 0;
            return this->v > v ? ls->upper(v) : rs->upper(v) +
                ls->sz + 1;
        }
    } *const null = new P, pool[M], *pit = pool;

    P* merge(P* l, P* r) {
        if (l == null) return r; if (r == null) return l;
        if (l->rd < r->rd) { l->rs = merge(l->rs, r); return l->
            up(); }
        else { r->ls = merge(l, r->ls); return r->up(); }
    }

    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        if (o->ls->sz >= rk) { split(o->ls, rk, l, o->ls); r = o
            ->up(); }
        else { split(o->rs, rk - o->ls->sz - 1, o->rs, r); l = o
            ->up(); }
    }
}

// persistent set
namespace treap {
```

```

const int M = maxn * 17 * 12;
extern struct P* const null, *pit;
struct P {
    P *ls, *rs;
    int v, sz;
    LL sum;
    P(P* ls, P* rs, int v): ls(ls), rs(rs), v(v), sz(ls->sz
        + rs->sz + 1),
        sum(ls->sum
            + rs
            ->sum
            + v)
    {}

    P() {}

    void* operator new(size_t _) { return pit++; }
    template<typename T>
    int rk(int v, T&& cmp) {
        if (this == null) return 0;
        return cmp(this->v, v) ? ls->rk(v, cmp) : rs->rk(v,
            cmp) + ls->sz + 1;
    }
    int lower(int v) { return rk(v, greater_equal<int>()); }
    int upper(int v) { return rk(v, greater<int>()); }
} pool[M], *pit = pool, *const null = new P;
P* merge(P* l, P* r) {
    if (l == null) return r; if (r == null) return l;
    if (rnd() % (l->sz + r->sz) < l->sz) return new P(l->ls,
        merge(l->rs, r), l->v);
    else return new P(merge(l, r->ls), r->rs, r->v);
}
void split(P* o, int rk, P*& l, P*& r) {
    if (o == null) { l = r = null; return; }
    if (o->ls->sz >= rk) { split(o->ls, rk, l, r); r = new P
        {r, o->rs, o->v}; }
    else { split(o->rs, rk - o->ls->sz - 1, l, r); l = new P
        {o->ls, l, o->v}; }
}
// persistent set with pushdown
int now;
namespace Treap {
    const int M = 10000000;
    extern struct P* const null, *pit;
    struct P {
        P *ls, *rs;
        int sz, time;
        LL cnt, sc, pos, add;
        bool rev;

        P* up() { sz = ls->sz + rs->sz + 1; sc = ls->sc + rs->sc
            + cnt; return this; } // MOD
        P* check() {
            if (time == now) return this;
            P* t = new(pit++) P; *t = *this; t->time = now;
            return t;
        };
        P* _do_rev() { rev ^= 1; add *= -1; pos *= -1; swap(ls,
            rs); return this; } // MOD
        P* _do_add(LL v) { add += v; pos += v; return this; } //
            MOD
        P* do_rev() { if (this == null) return this; return
            check()->_do_rev(); } // FIX & MOD
        P* do_add(LL v) { if (this == null) return this; return
            check()->_do_add(v); } // FIX & MOD
        P* _down() { // MOD
            if (rev) { ls = ls->do_rev(); rs = rs->do_rev(); rev
                = 0; }
            if (add) { ls = ls->do_add(add); rs = rs->do_add(add
                ); add = 0; }
            return this;
        }
        P* down() { return check()->_down(); } // FIX & MOD
        void _split(LL p, P*& l, P*& r) { // MOD
            if (pos >= p) { ls->split(p, l, r); ls = r; r = up()
                ; }
            else { rs->split(p, l, r); rs = l; l = up()
                ; }
        }
    }

    void split(LL p, P*& l, P*& r) { // FIX & MOD
        if (this == null) l = r = null;
        else down()->_split(p, l, r);
    }
} pool[M], *pit = pool, *const null = new P, *rt = null;

P* up() {
    assert(this != null);
    sz = ls->sz + rs->sz + 1;
    sum = ls->sum + rs->sum + val + add * sz;
    return this;
}
void upd(int v) {
    if (this == null) return;
    add += v;
    sum += sz * v;
}
P* down() {
    if (add) {
        ls->upd(add); rs->upd(add);
        val += add;
        add = 0;
    }
    return this;
}
P* select(int rk) {
    if (rk == ls->sz + 1) return this;
    return ls->sz >= rk ? ls->select(rk) : rs->select(rk
        - ls->sz - 1);
}
} pool[M], *pit = pool, *const null = new P, *rt = null;

P* merge(P* a, P* b) {
    if (a == null) return b;
    if (b == null) return a;
    if (rand() % (a->sz + b->sz) < a->sz) {
        return N(a->ls, merge(a->rs, b), a->v, a->fill);
    }
    else {
        return N(merge(a, b->ls), b->rs, b->v, b->fill);
    }
}
void go(P* o, int x, int y, P*& l, P*& m, P*& r) {
    o->split(y, l, r);
    l->split(x - 1, l, m);
}
}

split(rt, k - 1, l, r);
rt = merge(merge(l, new (pit++) P(v)), r);
}

inline void erase(int k) {
    P *l, *r, *_, *t;
    split(rt, k - 1, l, t);
    split(t, 1, _, r);
    rt = merge(l, r);
}

P* build(int l, int r, int* a) {
    if (l > r) return null;
    if (l == r) return new(pit++) P(a[l]);
    int m = (l + r) / 2;
    return (new(pit++) P(a[m], build(l, m - 1, a), build(m +
        1, r, a)))->up();
}
};
// persistent sequence
namespace treap {
    struct P;
    extern P*const null;
    P* N(P* ls, P* rs, LL v, bool fill);
    struct P {
        P *const ls, *const rs;
        const int sz, v;
        const LL sum;
        bool fill;
        int cnt;

        void split(int k, P*& l, P*& r) {
            if (this == null) { l = r = null; return; }
            if (ls->sz >= k) {
                ls->split(k, l, r);
                r = N(r, rs, v, fill);
            }
            else {
                rs->split(k - ls->sz - fill, l, r);
                l = N(ls, l, v, fill);
            }
        }

        P*const null = new P{0, 0, 0, 0, 0, 0, 1};

        P* N(P* ls, P* rs, LL v, bool fill) {
            ls->cnt++; rs->cnt++;
            return new P{ls, rs, ls->sz + rs->sz + fill, v, ls->sum
                + rs->sum + v, fill, 1};
        }

        P* merge(P* a, P* b) {
            if (a == null) return b;
            if (b == null) return a;
            if (rand() % (a->sz + b->sz) < a->sz)
                return N(a->ls, merge(a->rs, b), a->v, a->fill);
            else
                return N(merge(a, b->ls), b->rs, b->v, b->fill);
        }

        void go(P* o, int x, int y, P*& l, P*& m, P*& r) {
            o->split(y, l, r);
            l->split(x - 1, l, m);
        }
    }
}

```

2.9 Cartesian Tree

```

void build() {
    static int s[N], last;
    int p = 0;
    FOR (x, 1, n + 1) {
        last = 0;
        while (p && val[s[p - 1]] > val[x]) last = s[--p];
        if (p) G[s[p - 1]][1] = x;
        if (last) G[x][0] = last;
        s[p++] = x;
    }
}

```

```

    }
    rt = s[0];
}

```

2.10 LCT

// do not forget down when findint L/R most son
// make_root if not sure

```

namespace lct {
extern struct P *const null;
const int M = N;
struct P {
    P *fa, *ls, *rs;
    int v, maxv;
    bool rev;

    bool has_fa() { return fa->ls == this || fa->rs == this; }
    bool d() { return fa->ls == this; }
    P& c(bool x) { return x ? ls : rs; }
    void do_rev() {
        if (this == null) return;
        rev ^= 1;
        swap(ls, rs);
    }
    P* up() {
        maxv = max(v, max(ls->maxv, rs->maxv));
        return this;
    }
    void down() {
        if (rev) {
            rev = 0;
            ls->do_rev(); rs->do_rev();
        }
    }
    void all_down() { if (has_fa()) fa->all_down(); down(); }
} *const null = new P[0, 0, 0, 0, 0, 0], pool[M], *pit = pool;

void rot(P* o) {
    bool dd = o->d();
    P *f = o->fa, *t = o->c(!dd);
    if (f->has_fa()) f->fa->c(f->d()) = o; o->fa = f->fa;
    if (t != null) t->fa = f; f->c(dd) = t;
    o->c(!dd) = f->up(); f->fa = o;
}

void splay(P* o) {
    o->all_down();
    while (o->has_fa()) {
        if (o->fa->has_fa())
            rot(o->d() ^ o->fa->d() ? o : o->fa);
        rot(o);
    }
    o->up();
}

void access(P* u, P* v = null) {
    if (u == null) return;
    splay(u); u->rs = v;
    access(u->up()->fa, u);
}

void make_root(P* o) { access(o); splay(o); o->do_rev(); }
void split(P* u, P* v) { make_root(u); access(v); splay(v); }
bool linked(P* u, P* v) { split(u, v); return u == v || u->fa != null; }
void link(P* u, P* v) { make_root(u); u->fa = v; }
void cut(P* u, P* v) { split(u, v); u->fa = v->ls = null; v->up(); }

bool adj(P* u, P* v) {

```

```

    split(u, v);
    return v->ls == u && u->ls == null && u->rs == null;
}

bool linked(P* u, P* v) {
    split(u, v);
    return u == v || u->fa != null;
}

P* findrt(P* o) {
    access(o); splay(o);
    while (o->ls != null) o = o->ls;
    return o;
}

P* findfa(P* rt, P* u) {
    split(rt, u);
    u = u->ls;
    while (u->rs != null) {
        u = u->rs;
        u->down();
    }
    return u;
}

// maintain subtree size
P* up() {
    sz = ls->sz + rs->sz + _sz + 1;
    return this;
}

void access(P* u, P* v = null) {
    if (u == null) return;
    splay(u);
    u->_sz += u->rs->sz - v->sz;
    u->rs = v;
    access(u->up()->fa, u);
}

void link(P* u, P* v) {
    split(u, v);
    u->fa = v; v->_sz += u->sz;
    v->up();
}

//////////
// latest spanning tree
//////////
namespace lct {
extern struct P* null;
struct P {
    P *fa, *ls, *rs;
    int v;
    P *minp;
    bool rev;

    bool has_fa() { return fa->ls == this || fa->rs == this; }
    bool d() { return fa->ls == this; }
    P& c(bool x) { return x ? ls : rs; }
    void do_rev() { if (this == null) return; rev ^= 1; swap(ls, rs); }

    P* up() {
        minp = this;
        if (minp->v > ls->minp->v) minp = ls->minp;
        if (minp->v > rs->minp->v) minp = rs->minp;
        return this;
    }
    void down() { if (rev) { rev = 0; ls->do_rev(); rs->do_rev(); } }
    void all_down() { if (has_fa()) fa->all_down(); down(); }
} *null = new P[0, 0, 0, INF, 0, 0], pool[maxm], *pit = pool;

void rot(P* o) {
    bool dd = o->d();
    P *f = o->fa, *t = o->c(!dd);
    if (f->has_fa()) f->fa->c(f->d()) = o; o->fa = f->fa;
    if (t != null) t->fa = f; f->c(dd) = t;
    o->c(!dd) = f->up(); f->fa = o;
}

void splay(P* o) {

```

```

    o->all_down();
    while (o->has_fa()) {
        if (o->fa->has_fa()) rot(o->d() ^ o->fa->d() ? o : o->fa);
        rot(o);
    }
    o->up();
}

void access(P* u, P* v = null) {
    if (u == null) return;
    splay(u); u->rs = v;
    access(u->up()->fa, u);
}

void make_root(P* o) { access(o); splay(o); o->do_rev(); }
void split(P* u, P* v) { make_root(u); access(v); splay(v); }
bool linked(P* u, P* v) { split(u, v); return u == v || u->fa != null; }
void link(P* u, P* v) { make_root(u); u->fa = v; }
void cut(P* u, P* v) { split(u, v); u->fa = v->ls = null; v->up(); }
}

using namespace lct;
int n, m;
P *p[maxn];
struct Q {
    int tp, u, v, l, r;
};
vector<Q> q;

int main() {
    null->minp = null;
    cin >> n >> m;
    FOR (i, 1, n + 1) p[i] = new (pit++) P{null, null, null, INF, p[i], 0};
    int clk = 0;
    map<pair<int, int>, int> mp;
    FOR (_, 0, m) {
        int tp, u, v; scanf("%d%d%d", &tp, &u, &v);
        if (u > v) swap(u, v);
        if (tp == 0) mp.insert({{u, v}, clk});
        else if (tp == 1) {
            auto it = mp.find({u, v}); assert(it != mp.end());
            q.push_back({1, u, v, it->second, clk});
            mp.erase(it);
        } else q.push_back({0, u, v, clk, clk});
        ++clk;
    }
    for (auto& x: mp) q.push_back({1, x.first.first, x.first.second, x.second, clk});
    sort(q.begin(), q.end(), [](const Q& a, const Q& b)->bool { return a.l < b.l; });
    map<P*, int> mp2;
    FOR (i, 0, q.size()) {
        Q& cur = q[i];
        int u = cur.u, v = cur.v;
        if (cur.tp == 0) {
            if (!linked(p[u], p[v])) puts("N");
            else puts(p[v]->minp->v >= cur.r ? "Y" : "N");
            continue;
        }
        if (linked(p[u], p[v])) {
            P* t = p[v]->minp;
            if (t->v > cur.r) continue;
            Q& old = q[mp2[t]];
            cut(p[old.u], t); cut(p[old.v], t);
        }
        P* t = new (pit++) P{null, null, null, cur.r, t, 0};
        mp2[t] = i;
        link(t, p[u]); link(t, p[v]);
    }
}

```

2.11 Mo's Algorithm On Tree

```

struct Q {
    int u, v, idx;
    bool operator < (const Q& b) const {
        const Q& a = *this;
        return blk[a.u] < blk[b.u] || (blk[a.u] == blk[b.u] &&
            in[a.v] < in[b.v]);
    }
};

void dfs(int u = 1, int d = 0) {
    static int S[maxn], sz = 0, blk_cnt = 0, clk = 0;
    in[u] = clk++;
    dep[u] = d;
    int btm = sz;
    for (int v: G[u]) {
        if (v == fa[u]) continue;
        fa[v] = u;
        dfs(v, d + 1);
        if (sz - btm >= B) {
            while (sz > btm) blk[S[--sz]] = blk_cnt;
            ++blk_cnt;
        }
    }
    S[sz++] = u;
    if (u == 1) while (sz) blk[S[--sz]] = blk_cnt - 1;
}

void flip(int k) {
    dbg(k);
    if (vis[k]) {
        // ...
    } else {
        // ...
    }
    vis[k] ^= 1;
}

void go(int& k) {
    if (bug == -1) {
        if (vis[k] && !vis[fa[k]]) bug = k;
        if (!vis[k] && vis[fa[k]]) bug = fa[k];
    }
    flip(k);
    k = fa[k];
}

void mv(int a, int b) {
    bug = -1;
    if (vis[b]) bug = b;
    if (dep[a] < dep[b]) swap(a, b);
    while (dep[a] > dep[b]) go(a);
    while (a != b) {
        go(a); go(b);
    }
    go(a); go(bug);
}

for (Q& q: query) {
    mv(u, q.u); u = q.u;
    mv(v, q.v); v = q.v;
    ans[q.idx] = Ans;
}

```

2.12 CDQ's Divide and Conquer

```

const int maxn = 2E5 + 100;
struct P {
    int x, y;
    int* f;
    bool d1, d2;
} a[maxn], b[maxn], c[maxn];
int f[maxn];

void go2(int l, int r) {
    if (l + 1 == r) return;
    int m = (l + r) >> 1;

```

```

    go2(l, m); go2(m, r);
    FOR (i, l, m) b[i].d2 = 0;
    FOR (i, m, r) b[i].d2 = 1;
    merge(b + l, b + m, b + m, b + r, c + l, [](const P& a,
        const P& b)->bool {
        if (a.y != b.y) return a.y < b.y;
        return a.d2 > b.d2;
    });
    int mx = -1;
    FOR (i, l, r) {
        if (c[i].d1 && c[i].d2) *c[i].f = max(*c[i].f, mx + 1);
        if (!c[i].d1 && !c[i].d2) mx = max(mx, *c[i].f);
    }
    FOR (i, l, r) b[i] = c[i];

void go1(int l, int r) { // [l, r)
    if (l + 1 == r) return;
    int m = (l + r) >> 1;
    go1(l, m);
    FOR (i, l, m) a[i].d1 = 0;
    FOR (i, m, r) a[i].d1 = 1;
    copy(a + l, a + r, b + l);
    sort(b + l, b + r, [](const P& a, const P& b)->bool {
        if (a.x != b.x) return a.x < b.x;
        return a.d1 > b.d1;
    });
    go2(l, r);
    go1(m, r);
}

```

2.13 Persistent Segment Tree

```

namespace tree {
#define mid ((l + r) >> 1)
#define lson l, mid
#define rson mid + 1, r
    const int MAGIC = M * 30;
    struct P {
        int sum, ls, rs;
    } tr[MAGIC] = {{0, 0, 0}};
    int sz = 1;
    int N(int sum, int ls, int rs) {
        if (sz == MAGIC) assert(0);
        tr[sz] = {sum, ls, rs};
        return sz++;
    }
    int ins(int o, int x, int v, int l = 1, int r = ls) {
        if (x < l || x > r) return o;
        const P& t = tr[o];
        if (l == r) return N(t.sum + v, 0, 0);
        return N(t.sum + v, ins(t.ls, x, v, lson), ins(t.rs, x,
            v, rson));
    }
    int query(int o, int ql, int qr, int l = 1, int r = ls) {
        if (ql > r || l > qr) return 0;
        const P& t = tr[o];
        if (ql <= l && r <= qr) return t.sum;
        return query(t.ls, ql, qr, lson) + query(t.rs, ql, qr,
            rson);
    }
}

// kth
int query(int pp, int qq, int l, int r, int k) { // (pp, qq)
    if (l == r) return l;
    const P &p = tr[pp], &q = tr[qq];
    int w = tr[q.ls].w - tr[p.ls].w;
    if (k <= w) return query(p.ls, q.ls, lson, k);
    else return query(p.rs, q.rs, rson, k - w);
}

//////////
// with bit
//////////

typedef vector<int> VI;

```

```

struct TREE {
#define mid ((l + r) >> 1)
#define lson l, mid
#define rson mid + 1, r
    struct P {
        int w, ls, rs;
    } tr[maxn * 20 * 20];
    int sz = 1;
    TREE() { tr[0] = {0, 0, 0}; }
    int N(int w, int ls, int rs) {
        tr[sz] = {w, ls, rs};
        return sz++;
    }
    int add(int tt, int l, int r, int x, int d) {
        if (x < l || r < x) return tt;
        const P& t = tr[tt];
        if (l == r) return N(t.w + d, 0, 0);
        return N(t.w + d, add(t.ls, lson, x, d), add(t.rs, rson,
            x, d));
    }
    int ls_sum(const VI& rt) {
        int ret = 0;
        FOR (i, 0, rt.size())
            ret += tr[tr[rt[i]].ls].w;
        return ret;
    }
    inline void ls(VI& rt) { transform(rt.begin(), rt.end(), rt.
        begin(), [&](int x)->int{ return tr[x].ls; }); }
    inline void rs(VI& rt) { transform(rt.begin(), rt.end(), rt.
        begin(), [&](int x)->int{ return tr[x].rs; }); }
    int query(VI& p, VI& q, int l, int r, int k) {
        if (l == r) return l;
        int w = ls_sum(q) - ls_sum(p);
        if (k <= w) {
            ls(p); ls(q);
            return query(p, q, lson, k);
        }
        else {
            rs(p); rs(q);
            return query(p, q, rson, k - w);
        }
    }
} tree;

struct BIT {
    int root[maxn];
    void init() { memset(root, 0, sizeof root); }
    inline int lowbit(int x) { return x & -x; }
    void update(int p, int x, int d) {
        for (int i = p; i <= m; i += lowbit(i))
            root[i] = tree.add(root[i], 1, m, x, d);
    }
    int query(int l, int r, int k) {
        VI p, q;
        for (int i = l - 1; i > 0; i -= lowbit(i)) p.push_back(
            root[i]);
        for (int i = r; i > 0; i -= lowbit(i)) q.push_back(root[
            i]);
        return tree.query(p, q, 1, m, k);
    }
} bit;

void init() {
    m = 10000;
    tree.sz = 1;
    bit.init();
    FOR (i, 1, m + 1)
        bit.update(i, a[i], 1);
}

```

2.14 Persistent Union Find

```

namespace uf {
    int fa[maxn], sz[maxn];
    int undo[maxn], top;
    void init() { memset(fa, -1, sizeof fa); memset(sz, 0,
        sizeof sz); top = 0; }
}

```

```

int findset(int x) { while (fa[x] != -1) x = fa[x]; return x; }
bool join(int x, int y) {
    x = findset(x); y = findset(y);
    if (x == y) return false;
    if (sz[x] > sz[y]) swap(x, y);
    undo[top++] = x;
    fa[x] = y;
    sz[y] += sz[x] + 1;
    return true;
}
inline int checkpoint() { return top; }
void rewind(int t) {
    while (top > t) {
        int x = undo[--top];
        sz[fa[x]] -= sz[x] + 1;
        fa[x] = -1;
    }
}
}
}

```

3 Math

3.1 Multiplication, Powers

```

LL mul(LL u, LL v, LL p) {
    return (u * v - LL((long double) u * v / p) * p + p) % p;
}
LL mul(LL u, LL v, LL p) { // better constant
    LL t = u * v - LL((long double) u * v / p) * p;
    return t < 0 ? t + p : t;
}
LL bin(LL x, LL n, LL MOD) {
    n %= (MOD - 1); // if MOD is prime
    LL ret = MOD - 1;
    for (x %= MOD; n; n >>= 1, x = mul(x, x, MOD))
        if (n & 1) ret = mul(ret, x, MOD);
    return ret;
}
}

```

3.2 Matrix Power

```

struct Mat {
    static const LL M = 2;
    LL v[M][M];
    Mat() { memset(v, 0, sizeof v); }
    void eye() { FOR (i, 0, M) v[i][i] = 1; }
    LL* operator [] (LL x) { return v[x]; }
    const LL* operator [] (LL x) const { return v[x]; }
    Mat operator * (const Mat& B) {
        const Mat& A = *this;
        Mat ret;
        FOR (k, 0, M)
            FOR (i, 0, M) if (A[i][k])
                FOR (j, 0, M)
                    ret[i][j] = (ret[i][j] + A[i][k] * B[k][j]) % MOD;
        return ret;
    }
    Mat pow(LL n) const {
        Mat A = *this, ret; ret.eye();
        for (; n; n >>= 1, A = A * A)
            if (n & 1) ret = ret * A;
        return ret;
    }
    Mat operator + (const Mat& B) {
        const Mat& A = *this;
        Mat ret;
        FOR (i, 0, M)
            FOR (j, 0, M)
                ret[i][j] = (A[i][j] + B[i][j]) % MOD;
        return ret;
    }
    void prt() const {
        FOR (i, 0, M)

```

```

        FOR (j, 0, M)
            printf("%lld%c", (*this)[i][j], j == M - 1 ? '\n' : ' ');
    }
};

```

3.3 Sieve

```

const LL p_max = 1E5 + 100;
LL phi[p_max];
void get_phi() {
    phi[1] = 1;
    static bool vis[p_max];
    static LL prime[p_max], p_sz, d;
    FOR (i, 2, p_max) {
        if (!vis[i]) {
            prime[p_sz++] = i;
            phi[i] = i - 1;
        }
        for (LL j = 0; j < p_sz && (d = i * prime[j]) < p_max; ++j) {
            vis[d] = 1;
            if (i % prime[j] == 0) {
                phi[d] = phi[i] * prime[j];
                break;
            }
            else phi[d] = phi[i] * (prime[j] - 1);
        }
    }
}
// mobius
const LL p_max = 1E5 + 100;
LL mu[p_max];
void get_mu() {
    mu[1] = 1;
    static bool vis[p_max];
    static LL prime[p_max], p_sz, d;
    mu[1] = 1;
    FOR (i, 2, p_max) {
        if (!vis[i]) {
            prime[p_sz++] = i;
            mu[i] = -1;
        }
        for (LL j = 0; j < p_sz && (d = i * prime[j]) < p_max; ++j) {
            vis[d] = 1;
            if (i % prime[j] == 0) {
                mu[d] = 0;
                break;
            }
            else mu[d] = -mu[i];
        }
    }
}
// min_25
namespace min25 {
    const int M = 1E6 + 100;
    LL B, N;
    // g(x)
    inline LL pg(LL x) { return 1; }
    inline LL ph(LL x) { return x % MOD; }
    // Sum[g(i), {x, 2, x}]
    inline LL psg(LL x) { return x % MOD - 1; }
    inline LL psh(LL x) {
        static LL inv2 = (MOD + 1) / 2;
        x = x % MOD;
        return x * (x + 1) % MOD * inv2 % MOD - 1;
    }
    // f(pp=p^k)
    inline LL fpk(LL p, LL e, LL pp) { return (pp - pp / p) % MOD; }
    // f(p) = fgh(g(p), h(p))
    inline LL fgh(LL g, LL h) { return h - g; }
    LL pr[M], pc, sg[M], sh[M];
    void get_prime(LL n) {

```

```

        static bool vis[M]; pc = 0;
        FOR (i, 2, n + 1) {
            if (!vis[i]) {
                pr[pc++] = i;
                sg[pc] = (sg[pc - 1] + pg(i)) % MOD;
                sh[pc] = (sh[pc - 1] + ph(i)) % MOD;
            }
            FOR (j, 0, pc) {
                if (pr[j] * i > n) break;
                vis[pr[j] * i] = 1;
                if (i % pr[j] == 0) break;
            }
        }
        LL w[M];
        LL id1[M], id2[M], h[M], g[M];
        inline LL id(LL x) { return x <= B ? id1[x] : id2[N / x]; }
        LL go(LL x, LL k) {
            if (x <= 1 || (k >= 0 && pr[k] > x)) return 0;
            LL t = id(x);
            LL ans = fgh((g[t] - sg[k + 1]), (h[t] - sh[k + 1]));
            FOR (i, k + 1, pc) {
                LL p = pr[i];
                if (p * p > x) break;
                ans -= fgh(pg(p), ph(p));
                for (LL pp = p, e = 1; pp <= x; ++e, pp = pp * p)
                    ans += fpk(p, e, pp) * (1 + go(x / pp, i)) % MOD;
            }
            return ans % MOD;
        }
        LL solve(LL _N) {
            N = _N;
            B = sqrt(N + 0.5);
            get_prime(B);
            int sz = 0;
            for (LL l = 1, v, r; l <= N; l = r + 1) {
                v = N / l; r = N / v;
                w[sz] = v; g[sz] = psg(v); h[sz] = psh(v);
                if (v <= B) id1[v] = sz; else id2[r] = sz;
                sz++;
            }
            FOR (k, 0, pc) {
                LL p = pr[k];
                FOR (i, 0, sz) {
                    LL v = w[i]; if (p * p > v) break;
                    LL t = id(v / p);
                    g[i] = (g[i] - (g[t] - sg[k]) * pg(p)) % MOD;
                    h[i] = (h[i] - (h[t] - sh[k]) * ph(p)) % MOD;
                }
            }
            return (go(N, -1) % MOD + MOD + 1) % MOD;
        }
    }
}
// see cheatsheet for instructions
namespace dujiao {
    const int M = 5E6;
    LL f[M] = {0, 1};
    void init() {
        static bool vis[M];
        static LL pr[M], p_sz, d;
        FOR (i, 2, M) {
            if (!vis[i]) { pr[p_sz++] = i; f[i] = -1; }
            FOR (j, 0, p_sz) {
                if ((d = pr[j] * i) >= M) break;
                vis[d] = 1;
                if (i % pr[j] == 0) {
                    f[d] = 0;
                    break;
                }
                else f[d] = -f[i];
            }
        }
        FOR (i, 2, M) f[i] += f[i - 1];
    }
    inline LL s_fg(LL n) { return 1; }
    inline LL s_g(LL n) { return n; }
}

```



```

LL N, rd[M];
bool vis[M];
LL go(LL n) {
    if (n < M) return f[n];
    LL id = N / n;
    if (vis[id]) return rd[id];
    vis[id] = true;
    LL& ret = rd[id] = s_fg(n);
    for (LL l = 2, v, r; l <= n; l = r + 1) {
        v = n / l; r = n / v;
        ret -= (s_g(r) - s_g(l - 1)) * go(v);
    }
    return ret;
}
LL solve(LL n) {
    N = n;
    memset(vis, 0, sizeof vis);
    return go(n);
}
}

```

3.4 Prime Test

```

bool checkQ(LL a, LL n) {
    if (n == 2 || a >= n) return 1;
    if (n == 1 || !(n & 1)) return 0;
    LL d = n - 1;
    while (!(d & 1)) d >>= 1;
    LL t = bin(a, d, n); // usually needs mul-on-LL
    while (d != n - 1 && t != 1 && t != n - 1) {
        t = mul(t, t, n);
        d <<= 1;
    }
    return t == n - 1 || d & 1;
}

bool primeQ(LL n) {
    static vector<LL> t = {2, 325, 9375, 28178, 450775, 9780504,
        1795265022};
    if (n <= 1) return false;
    for (LL k: t) if (!checkQ(k, n)) return false;
    return true;
}

```

3.5 Pollard-Rho

```

mt19937 mt(time(0));
LL pollard_rho(LL n, LL c) {
    LL x = uniform_int_distribution<LL>(1, n - 1)(mt), y = x;
    auto f = [&](LL v) { LL t = mul(v, v, n) + c; return t < n ?
        t : t - n; };
    while (1) {
        x = f(x); y = f(f(y));
        if (x == y) return n;
        LL d = gcd(abs(x - y), n);
        if (d != 1) return d;
    }
}

LL fac[100], fcnt;
void get_fac(LL n, LL cc = 19260817) {
    if (n == 4) { fac[fcnt++] = 2; fac[fcnt++] = 2; return; }
    if (primeQ(n)) { fac[fcnt++] = n; return; }
    LL p = n;
    while (p == n) p = pollard_rho(n, --cc);
    get_fac(p); get_fac(n / p);
}

```

3.6 Berlekamp-Massey

```

namespace BerlekampMassey {
    inline void up(LL& a, LL b) { (a += b) %= MOD; }
    V mul(const V&a, const V& b, const V& m, int k) {
        V r; r.resize(2 * k - 1);
        FOR (i, 0, k) FOR (j, 0, k) up(r[i + j], a[i] * b[j]);
    }
}

```

```

FORD (i, k - 2, -1) {
    FOR (j, 0, k) up(r[i + j], r[i + k] * m[j]);
    r.pop_back();
}
return r;
}
V pow(LL n, const V& m) {
    int k = (int) m.size() - 1; assert (m[k] == -1 || m[k]
        == MOD - 1);
    V r(k), x(k); r[0] = x[1] = 1;
    for (; n; n >>= 1, x = mul(x, x, m, k))
        if (n & 1) r = mul(x, r, m, k);
    return r;
}
LL go(const V& a, const V& x, LL n) {
    // a: (-1, a1, a2, ..., ak).reverse
    // x: x1, x2, ..., xk
    // x[n] = sum[a[i]*x[n-i],{i,1,k}]
    int k = (int) a.size() - 1;
    if (n <= k) return x[n - 1];
    if (a.size() == 2) return x[0] * bin(a[0], n - 1, MOD) %
        MOD;
    V r = pow(n - 1, a);
    LL ans = 0;
    FOR (i, 0, k) up(ans, r[i] * x[i]);
    return (ans + MOD) % MOD;
}
V BM(const V& x) {
    V a = {-1}, b = {233}, t;
    FOR (i, 1, x.size()) {
        b.push_back(0);
        LL d = 0, la = a.size(), lb = b.size();
        FOR (j, 0, la) up(d, a[j] * x[i - la + 1 + j]);
        if (d == 0) continue;
        t.clear(); for (auto& v: b) t.push_back(d * v % MOD);
        FOR (_, 0, la - lb) t.push_back(0);
        lb = max(la, lb);
        FOR (j, 0, la) up(t[lb - 1 - j], a[la - 1 - j]);
        if (lb > la) {
            b.swap(a);
            LL inv = -get_inv(d, MOD);
            for (auto& v: b) v = v * inv % MOD;
        }
        a.swap(t);
    }
    for (auto& v: a) up(v, MOD);
    return a;
}
}

```

3.7 Extended Euclidean

```

LL ex_gcd(LL a, LL b, LL &x, LL &y) {
    if (b == 0) { x = 1; y = 0; return a; }
    LL ret = ex_gcd(b, a % b, y, x);
    y -= a / b * x;
    return ret;
}
///////////////////////////////////////////////////
inline int ctz(LL x) { return __builtin_ctzll(x); }
LL gcd(LL a, LL b) {
    if (!a) return b; if (!b) return a;
    int t = ctz(a | b);
    a >>= ctz(a);
    do {
        b >>= ctz(b);
        if (a > b) swap(a, b);
        b -= a;
    } while (b);
    return a << t;
}

```

3.8 Inverse

```
// if p is prime
```

```

inline LL get_inv(LL x, LL p) { return bin(x, p - 2, p); }
// if p is not prime
LL get_inv(LL a, LL M) {
    static LL x, y;
    assert(exgcd(a, M, x, y) == 1);
    return (x % M + M) % M;
}
///////////////////////////////////////////////////
LL inv[N];
void inv_init(LL n, LL p) {
    inv[1] = 1;
    FOR (i, 2, n)
        inv[i] = (p - p / i) * inv[p % i] % p;
}
///////////////////////////////////////////////////
LL invf[M], fac[M] = {1};
void fac_inv_init(LL n, LL p) {
    FOR (i, 1, n)
        fac[i] = i * fac[i - 1] % p;
    invf[n - 1] = bin(fac[n - 1], p - 2, p);
    FORD (i, n - 2, -1)
        invf[i] = invf[i + 1] * (i + 1) % p;
}

```

3.9 Binomial Numbers

```

inline LL C(LL n, LL m) { // n >= m >= 0
    return n < m || m < 0 ? 0 : fac[n] * invf[m] % MOD * invf[n
        - m] % MOD;
}
// The following code reverses n and m
LL C(LL n, LL m) { // m >= n >= 0
    if (m - n < n) n = m - n;
    if (n < 0) return 0;
    LL ret = 1;
    FOR (i, 1, n + 1)
        ret = ret * (m - n + i) % MOD * bin(i, MOD - 2, MOD) %
            MOD;
    return ret;
}
LL Lucas(LL n, LL m) { // m >= n >= 0
    return m ? C(n % MOD, m % MOD) * Lucas(n / MOD, m / MOD) %
        MOD : 1;
}
// precalculations
LL C[M][M];
void init_C(int n) {
    FOR (i, 0, n) {
        C[i][0] = C[i][i] = 1;
        FOR (j, 1, i)
            C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % MOD;
    }
}

```

3.10 NTT, FFT, FWT

```

// NTT
LL wn[N << 2], rev[N << 2];
int NTT_init(int n) {
    int step = 0; int n = 1;
    for (; n < n_; n <= 1) ++step;
    FOR (i, 1, n)
        rev[i] = (rev[i] >> 1) >> 1 | ((i & 1) << (step - 1));
    int g = bin(G, (MOD - 1) / n, MOD);
    wn[0] = 1;
    for (int i = 1; i <= n; ++i)
        wn[i] = wn[i - 1] * g % MOD;
    return n;
}
void NTT(LL a[], int n, int f) {
    FOR (i, 0, n) if (i < rev[i])
        std::swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k <= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            int t = n / (k << 1);

```



```

    FOR (j, 0, k) {
        LL w = f == 1 ? wn[t * j] : wn[n - t * j];
        LL x = a[i + j];
        LL y = a[i + j + k] * w % MOD;
        a[i + j] = (x + y) % MOD;
        a[i + j + k] = (x - y + MOD) % MOD;
    }
}
if (f == -1) {
    LL ninv = get_inv(n, MOD);
    FOR (i, 0, n)
        a[i] = a[i] * ninv % MOD;
}

// FFT
// n needs to be power of 2
typedef double LD;
const LD PI = acos(-1);
struct C {
    LD r, i;
    C(LD r = 0, LD i = 0): r(r), i(i) {}
};
C operator + (const C& a, const C& b) {
    return C(a.r + b.r, a.i + b.i);
}
C operator - (const C& a, const C& b) {
    return C(a.r - b.r, a.i - b.i);
}
C operator * (const C& a, const C& b) {
    return C(a.r * b.r - a.i * b.i, a.r * b.i + a.i * b.r);
}
void FFT(C x[], int n, int p) {
    for (int i = 0, t = 0; i < n; ++i) {
        if (i > t) swap(x[i], x[t]);
        for (int j = n >> 1; (t ^= j) < j; j >>= 1);
        for (int h = 2; h <= n; h <= 1) {
            C wn(cos(p * 2 * PI / h), sin(p * 2 * PI / h));
            for (int i = 0; i < n; i += h) {
                C w(1, 0), u;
                for (int j = i, k = h >> 1; j < i + k; ++j) {
                    u = x[j + k] * w;
                    x[j + k] = x[j] - u;
                    x[j] = x[j] + u;
                    w = w * wn;
                }
            }
        }
        if (p == -1)
            FOR (i, 0, n)
                x[i].r /= n;
    }
}
void conv(C a[], C b[], int n) {
    FFT(a, n, 1);
    FFT(b, n, 1);
    FOR (i, 0, n)
        a[i] = a[i] * b[i];
    FFT(a, n, -1);
}

// FWT
// C_k = \sum_{i \oplus j=k} A_i B_j
template<typename T>
void fwt(LL a[], int n, T f) {
    for (int d = 1; d < n; d *= 2)
        for (int i = 0, t = d * 2; i < n; i += t)
            FOR (j, 0, d)
                f(a[i + j], a[i + j + d]);
}

void AND(LL& a, LL& b) { a += b; }
void OR(LL& a, LL& b) { b += a; }
void XOR(LL& a, LL& b) {
    LL x = a, y = b;
    a = (x + y) % MOD;

```

```

    b = (x - y + MOD) % MOD;
}
void rAND(LL& a, LL& b) { a -= b; }
void rOR(LL& a, LL& b) { b -= a; }
void rXOR(LL& a, LL& b) {
    static LL INV2 = (MOD + 1) / 2;
    LL x = a, y = b;
    a = (x + y) * INV2 % MOD;
    b = (x - y + MOD) * INV2 % MOD;
}

/*
FWT subset convolution
a[popcount(x)][x] = A[x]
b[popcount(x)][x] = B[x]
fwt(a[i]) fwt(b[i])
c[i + j][x] += a[i][x] * b[j][x]
rfwt(c[i])
ans[x] = c[popcount(x)][x]
*/

```

3.11 Simpson's Numerical Integration

```

LD simpson(LD l, LD r) {
    LD c = (l + r) / 2;
    return (f(l) + 4 * f(c) + f(r)) * (r - l) / 6;
}

LD asr(LD l, LD r, LD eps, LD S) {
    LD m = (l + r) / 2;
    LD L = simpson(l, m), R = simpson(m, r);
    if (fabs(L + R - S) < 15 * eps) return L + R + (L + R - S) / 15;
    return asr(l, m, eps / 2, L) + asr(m, r, eps / 2, R);
}

LD asr(LD l, LD r, LD eps) { return asr(l, r, eps, simpson(l, r)); }

```

3.12 Gauss Elimination

```

// n equations, m variables
// a is an n x (m + 1) augmented matrix
// free is an indicator of free variable
// return the number of free variables, -1 for "404"
int n, m;
LD a[maxn][maxn], x[maxn];
bool free_x[maxn];
inline int sgn(LD x) { return (x > eps) - (x < -eps); }
int gauss(LD a[maxn][maxn], int n, int m) {
    memset(free_x, 1, sizeof free_x); memset(x, 0, sizeof x);
    int r = 0, c = 0;
    while (r < n && c < m) {
        int m_r = r;
        FOR (i, r + 1, n)
            if (fabs(a[i][c]) > fabs(a[m_r][c])) m_r = i;
        if (m_r != r)
            FOR (j, c, m + 1)
                swap(a[r][j], a[m_r][j]);
        if (!sgn(a[r][c])) {
            a[r][c] = 0; ++c;
            continue;
        }
        FOR (i, r + 1, n)
            if (a[i][c]) {
                LD t = a[i][c] / a[r][c];
                FOR (j, c, m + 1) a[i][j] -= a[r][j] * t;
            }
        ++r; ++c;
    }
    FOR (i, r, n)
        if (sgn(a[i][m])) return -1;
    if (r < m) {
        FORD (i, r - 1, -1) {
            int f_cnt = 0, k = -1;

```

```

        FOR (j, 0, m)
            if (sgn(a[i][j]) && free_x[j]) {
                ++f_cnt; k = j;
            }
        if (f_cnt > 0) continue;
        LD s = a[i][m];
        FOR (j, 0, m)
            if (j != k) s -= a[i][j] * x[j];
        x[k] = s / a[i][k];
        free_x[k] = 0;
    }
    return m - r;
}
FORD (i, m - 1, -1) {
    LD s = a[i][m];
    FOR (j, i + 1, m)
        s -= a[i][j] * x[j];
    x[i] = s / a[i][i];
}
return 0;
}

```

3.13 Factor Decomposition

```

LL factor[30], f_sz, factor_exp[30];
void get_factor(LL x) {
    f_sz = 0;
    LL t = sqrt(x + 0.5);
    for (LL i = 0; pr[i] <= t; ++i)
        if (x % pr[i] == 0) {
            factor_exp[f_sz] = 0;
            while (x % pr[i] == 0) {
                x /= pr[i];
                ++factor_exp[f_sz];
            }
            factor[f_sz++] = pr[i];
        }
    if (x > 1) {
        factor_exp[f_sz] = 1;
        factor[f_sz++] = x;
    }
}

```

3.14 Primitive Root

```

LL find_smallest_primitive_root(LL p) {
    // p should be a prime
    get_factor(p - 1);
    FOR (i, 2, p) {
        bool flag = true;
        FOR (j, 0, f_sz)
            if (bin(i, (p - 1) / factor[j], p) == 1) {
                flag = false;
                break;
            }
        if (flag) return i;
    }
    assert(0); return -1;
}

```

3.15 Quadratic Residue

```

LL q1, q2, w;
struct P { // x + y * sqrt(w)
    LL x, y;
};
P pmul(const P& a, const P& b, LL p) {
    P res;
    res.x = (a.x * b.x + a.y * b.y % p * w) % p;
    res.y = (a.x * b.y + a.y * b.x) % p;
    return res;
}
P bin(P x, LL n, LL MOD) {
    P ret = {1, 0};
    for (; n >>= 1, x = pmul(x, x, MOD))

```

```

        if (n & 1) ret = pmul(ret, x, MOD);
    return ret;
}
LL Legendre(LL a, LL p) { return bin(a, (p - 1) >> 1, p); }
LL equation_solve(LL b, LL p) {
    if (p == 2) return 1;
    if ((Legendre(b, p) + 1) % p == 0)
        return -1;
    LL a;
    while (true) {
        a = rand() % p;
        w = ((a * a - b) % p + p) % p;
        if ((Legendre(w, p) + 1) % p == 0)
            break;
    }
    return bin([a, 1], (p + 1) >> 1, p).x;
}
// Given a and prime p, find x such that x*x=a(mod p)
int main() {
    LL a, p; cin >> a >> p;
    a = a % p;
    LL x = equation_solve(a, p);
    if (x == -1) {
        puts("No root");
    } else {
        LL y = p - x;
        if (x == y) cout << x << endl;
        else cout << min(x, y) << " " << max(x, y) << endl;
    }
}

```

3.16 Chinese Remainder Theorem

```

LL CRT(LL *m, LL *r, LL n) {
    if (!n) return 0;
    LL M = m[0], R = r[0], x, y, d;
    FOR (i, 1, n) {
        d = ex_gcd(M, m[i], x, y);
        if ((r[i] - R) % d) return -1;
        x = (r[i] - R) / d * x % (m[i] / d);
        R += x * M;
        M = M / d * m[i];
        R %= M;
    }
    return R >= 0 ? R : R + M;
}

```

3.17 Bernoulli Numbers

```

namespace Bernoulli {
    LL inv[M] = {-1, 1};
    LL C[M][M];
    void init();
    LL B[M] = {1};
    void init() {
        inv_init(M, MOD);
        init_C(M);
        FOR (i, 1, M - 1) {
            LL& s = B[i] % MOD;
            FOR (j, 0, i)
                s += C[i + 1][j] * B[j] % MOD;
            s = (s % MOD * -inv[i + 1] % MOD + MOD) % MOD;
        }
    }
    LL p[M] = {1};
    LL go(LL n, LL k) {
        n %= MOD;
        if (k == 0) return n;
        FOR (i, 1, k + 2)
            p[i] = p[i - 1] * (n + 1) % MOD;
        LL ret = 0;
        FOR (i, 1, k + 2)
            ret += C[k + 1][i] * B[k + 1 - i] % MOD * p[i] % MOD;
        ret = ret % MOD * inv[k + 1] % MOD;
        return ret;
    }
}

```

```

}
3.18 Simplex Method
// x = 0 should satisfy the constraints
// initialize v to be 0
// n is dimension of vector, m is number of constraints
// min{ b x } / max { c x }
// A x >= c / A x <= b
// x >= 0
namespace lp {
    int n, m;
    double a[M][N], b[M], c[N], v;

    void pivot(int l, int e) {
        b[l] /= a[l][e];
        FOR (j, 0, n) if (j != e) a[l][j] /= a[l][e];
        a[l][e] = 1 / a[l][e];

        FOR (i, 0, m)
            if (i != l && fabs(a[i][e]) > 0) {
                b[i] -= a[i][e] * b[l];
                FOR (j, 0, n)
                    if (j != e) a[i][j] -= a[i][e] * a[l][j];
                a[i][e] = -a[i][e] * a[l][e];
            }
        v += c[e] * b[l];
        FOR (j, 0, n) if (j != e) c[j] -= c[e] * a[l][j];
        c[e] = -c[e] * a[l][e];
    }

    double simplex() {
        while (1) {
            v = 0;
            int e = -1, l = -1;
            FOR (i, 0, n) if (c[i] > eps) { e = i; break; }
            if (e == -1) return v;
            double t = INF;
            FOR (i, 0, m)
                if (a[i][e] > eps && t > b[i] / a[i][e]) {
                    t = b[i] / a[i][e];
                    l = i;
                }
            if (l == -1) return INF;
            pivot(l, e);
        }
    }
}

```

3.19 BSGS

```

// p is a prime
LL BSGS(LL a, LL b, LL p) { // a^x = b (mod p)
    a %= p;
    if (!a && !b) return 1;
    if (!a) return -1;
    static map<LL, LL> mp; mp.clear();
    LL m = sqrt(p + 1.5);
    LL v = 1;
    FOR (i, 1, m + 1) {
        v = v * a % p;
        mp[v * b % p] = i;
    }
    LL vv = v;
    FOR (i, 1, m + 1) {
        auto it = mp.find(vv);
        if (it != mp.end()) return i * m - it->second;
        vv = vv * v % p;
    }
    return -1;
}
// p can be not a prime
LL exBSGS(LL a, LL b, LL p) { // a^x = b (mod p)
    a %= p; b %= p;
    if (a == 0) return b > 1 ? -1 : b == 0 && p != 1;
    LL c = 0, q = 1;
    while (1) {

```

```

        LL g = __gcd(a, p);
        if (g == 1) break;
        if (b == 1) return c;
        if (b % g) return -1;
        ++c; b /= g; p /= g; q = a / g * q % p;
    }
    static map<LL, LL> mp; mp.clear();
    LL m = sqrt(p + 1.5);
    LL v = 1;
    FOR (i, 1, m + 1) {
        v = v * a % p;
        mp[v * b % p] = i;
    }
    FOR (i, 1, m + 1) {
        q = q * v % p;
        auto it = mp.find(q);
        if (it != mp.end()) return i * m - it->second + c;
    }
    return -1;
}

```

4 Graph Theory

4.1 LCA

```

void dfs(int u, int fa) {
    pa[u][0] = fa; dep[u] = dep[fa] + 1;
    FOR (i, 1, SP) pa[u][i] = pa[pa[u][i - 1]][i - 1];
    for (int& v: G[u]) {
        if (v == fa) continue;
        dfs(v, u);
    }
}
int lca(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    int t = dep[u] - dep[v];
    FOR (i, 0, SP) if (t & (1 << i)) u = pa[u][i];
    FOR (i, SP - 1, -1) {
        int uu = pa[u][i], vv = pa[v][i];
        if (uu != vv) { u = uu; v = vv; }
    }
    return u == v ? u : pa[u][0];
}

```

4.2 Maximum Flow

```

struct E {
    int to, cp;
    E(int to, int cp): to(to), cp(cp) {}
};

struct Dinic {
    static const int M = 1E5 * 5;
    int m, s, t;
    vector<E> edges;
    vector<int> G[M];
    int d[M];
    int cur[M];
    void init(int n, int s, int t) {
        this->s = s; this->t = t;
        for (int i = 0; i <= n; i++) G[i].clear();
        edges.clear(); m = 0;
    }
    void addedge(int u, int v, int cap) {
        edges.emplace_back(v, cap);
        edges.emplace_back(u, 0);
        G[u].push_back(m++);
        G[v].push_back(m++);
    }
    bool BFS() {
        memset(d, 0, sizeof d);
        queue<int> Q;
        Q.push(s); d[s] = 1;
        while (!Q.empty()) {

```

```

    int x = Q.front(); Q.pop();
    for (int& i: G[x]) {
        E& e = edges[i];
        if (!d[e.to] && e.cp > 0) {
            d[e.to] = d[x] + 1;
            Q.push(e.to);
        }
    }
    return d[t];
}

int DFS(int u, int cp) {
    if (u == t || !cp) return cp;
    int tmp = cp, f;
    for (int& i = cur[u]; i < G[u].size(); i++) {
        E& e = edges[G[u][i]];
        if (d[u] + 1 == d[e.to]) {
            f = DFS(e.to, min(cp, e.cp));
            e.cp -= f;
            edges[G[u][i] ^ 1].cp += f;
            cp -= f;
            if (!cp) break;
        }
    }
    return tmp - cp;
}

int go() {
    int flow = 0;
    while (BFS()) {
        memset(cur, 0, sizeof cur);
        flow += DFS(s, INF);
    }
    return flow;
}
} DC;

```

4.3 Minimum Cost Maximum Flow

```

struct E {
    int from, to, cp, v;
    E() {}
    E(int f, int t, int cp, int v) : from(f), to(t), cp(cp), v(v) {}
};

struct MCMF {
    int n, m, s, t;
    vector<E> edges;
    vector<int> G[maxn];
    bool inq[maxn];
    int d[maxn]; // shortest path
    int p[maxn]; // the last edge id of the path from s to i
    int a[maxn]; // least remaining capacity from s to i
    void init(int _n, int _s, int _t) {}
    void addedge(int from, int to, int cap, int cost) {
        edges.emplace_back(from, to, cap, cost);
        edges.emplace_back(to, from, 0, -cost);
        G[from].push_back(m++);
        G[to].push_back(m++);
    }

    bool BellmanFord(int &flow, int &cost) {
        FOR (i, 0, n + 1) d[i] = INF;
        memset(inq, 0, sizeof inq);
        d[s] = 0, a[s] = INF, inq[s] = true;
        queue<int> Q; Q.push(s);
        while (!Q.empty()) {
            int u = Q.front(); Q.pop();
            inq[u] = false;
            for (int& idx: G[u]) {
                E& e = edges[idx];
                if (e.cp && d[e.to] > d[u] + e.v) {
                    d[e.to] = d[u] + e.v;
                    p[e.to] = idx;
                    a[e.to] = min(a[u], e.cp);
                    if (!inq[e.to]) {
                        Q.push(e.to);
                        inq[e.to] = true;
                    }
                }
            }
        }
    }
};

```

```

    }
    }
    if (d[t] == INF) return false;
    flow += a[t];
    cost += a[t] * d[t];
    int u = t;
    while (u != s) {
        edges[p[u]].cp -= a[t];
        edges[p[u] ^ 1].cp += a[t];
        u = edges[p[u]].from;
    }
    return true;
}

int go() {
    int flow = 0, cost = 0;
    while (BellmanFord(flow, cost));
    return cost;
}
} MM;

```

4.4 Path Intersection on Trees

```

int intersection(int x, int y, int xx, int yy) {
    int t[4] = {lca(x, xx), lca(x, yy), lca(y, xx), lca(y, yy)};
    sort(t, t + 4);
    int r = lca(x, y), rr = lca(xx, yy);
    if (dep[t[0]] < min(dep[r], dep[rr]) || dep[t[2]] < max(dep[r], dep[rr]))
        return 0;
    return 0;
    int tt = lca(t[2], t[3]);
    int ret = 1 + dep[t[2]] + dep[t[3]] - dep[tt] * 2;
    return ret;
}

```

4.5 Centroid Decomposition (Divide-Conquer)

```

int get_rt(int u) {
    static int q[N], fa[N], sz[N], mx[N];
    int p = 0, cur = -1;
    q[p++] = u; fa[u] = -1;
    while (++cur < p) {
        u = q[cur]; mx[u] = 0; sz[u] = 1;
        for (int& v: G[u])
            if (!vis[v] && v != fa[u]) fa[q[p++] = v] = u;
    }
    FOR (i, p - 1, -1) {
        u = q[i];
        mx[u] = max(mx[u], p - sz[u]);
        if (mx[u] * 2 <= p) return u;
        sz[fa[u]] += sz[u];
        mx[fa[u]] = max(mx[fa[u]], sz[u]);
    }
    assert(0);
}

void dfs(int u) {
    u = get_rt(u);
    vis[u] = true;
    get_dep(u, -1, 0);
    // ...
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v]) continue;
        // ...
        dfs(v);
    }
}

// dynamic divide and conquer

```

```

const int maxn = 15E4 + 100, INF = 1E9;
struct E {
    int to, d;
};
vector<E> G[maxn];
int n, Q, w[maxn];
LL A, ans;

bool vis[maxn];
int sz[maxn];

int get_rt(int u) {
    static int q[N], fa[N], sz[N], mx[N];
    int p = 0, cur = -1;
    q[p++] = u; fa[u] = -1;
    while (++cur < p) {
        u = q[cur]; mx[u] = 0; sz[u] = 1;
        for (int& v: G[u])
            if (!vis[v] && v != fa[u]) fa[q[p++] = v] = u;
    }
    FORD (i, p - 1, -1) {
        u = q[i];
        mx[u] = max(mx[u], p - sz[u]);
        if (mx[u] * 2 <= p) return u;
        sz[fa[u]] += sz[u];
        mx[fa[u]] = max(mx[fa[u]], sz[u]);
    }
    assert(0);
}

int dep[maxn], md[maxn];
void get_dep(int u, int fa, int d) {
    dep[u] = d; md[u] = 0;
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v] || v == fa) continue;
        get_dep(v, u, d + e.d);
        md[u] = max(md[u], md[v] + 1);
    }
}

struct P {
    int w;
    LL s;
};
using VP = vector<P>;
struct R {
    VP *rt, *rt2;
    int dep;
};
VP pool[maxn << 1], *pit = pool;
vector<R> tr[maxn];

void go(int u, int fa, VP* rt, VP* rt2) {
    tr[u].push_back({rt, rt2, dep[u]});
    for (E& e: G[u]) {
        int v = e.to;
        if (v == fa || vis[v]) continue;
        go(v, u, rt, rt2);
    }
}

void dfs(int u) {
    u = get_rt(u);
    vis[u] = true;
    get_dep(u, -1, 0);
    VP* rt = pit++; tr[u].push_back({rt, nullptr, 0});
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v]) continue;
        go(v, u, rt, pit++);
        dfs(v);
    }
}

bool cmp(const P& a, const P& b) { return a.w < b.w; }

LL query(VP& p, int d, int l, int r) {

```

```

l = lower_bound(p.begin(), p.end(), P[l, -1], cmp) - p.begin();
r = upper_bound(p.begin(), p.end(), P[r, -1], cmp) - p.begin();
return p[r].s - p[l - 1].s + 1LL * (r - l + 1) * d;
}

int main() {
    cin >> n >> Q >> A;
    FOR (i, 1, n + 1) scanf("%d", &w[i]);
    FOR (_, 1, n) {
        int u, v, d; scanf("%d%d%d", &u, &v, &d);
        G[u].push_back({v, d}); G[v].push_back({u, d});
    }
    dfs(1);
    FOR (i, 1, n + 1)
        for (R& x: tr[i]) {
            x.rt->push_back({w[i], x.dep});
            if (x.rt2) x.rt2->push_back({w[i], x.dep});
        }
    FOR (it, pool, pit) {
        it->push_back({-INF, 0});
        sort(it->begin(), it->end(), cmp);
        FOR (i, 1, it->size())
            (*it)[i].s += (*it)[i - 1].s;
    }
    while (Q--) {
        int u; LL a, b; scanf("%d%lld%lld", &u, &a, &b);
        a = (a + ans) % A; b = (b + ans) % A;
        int l = min(a, b), r = max(a, b);
        ans = 0;
        for (R& x: tr[u]) {
            ans += query(x.rt, x.dep, l, r);
            if (x.rt2) ans += query(x.rt2, x.dep, l, r);
        }
        printf("%lld\n", ans);
    }
}

```

4.6 Heavy-light Decomposition

```

// clear clk
// usage: hld::predfs(1, 1); hld::dfs(1, 1);
int fa[N], dep[N], idx[N], out[N], ridx[N];
namespace hld {
    int sz[N], son[N], top[N], clk;
    void predfs(int u, int d) {
        dep[u] = d; sz[u] = 1;
        int& maxs = son[u] = -1;
        for (int& v: G[u]) {
            if (v == fa[u]) continue;
            fa[v] = u;
            predfs(v, d + 1);
            sz[u] += sz[v];
            if (maxs == -1 || sz[v] > sz[maxs]) maxs = v;
        }
    }
    void dfs(int u, int tp) {
        top[u] = tp; idx[u] = ++clk; ridx[clk] = u;
        if (son[u] != -1) dfs(son[u], tp);
        for (int& v: G[u])
            if (v != fa[u] && v != son[u]) dfs(v, v);
        out[u] = clk;
    }
    template<typename T>
    int go(int u, int v, T&& f = [] (int, int) {}) {
        int uu = top[u], vv = top[v];
        while (uu != vv) {
            if (dep[uu] < dep[vv]) { swap(uu, vv); swap(u, v); }
            f(idx[uu], idx[vv]);
            u = fa[uu]; uu = top[u];
        }
        if (dep[u] < dep[v]) swap(u, v);
        // choose one
        // f(idx[v], idx[u]);
        // if (u != v) f(idx[v] + 1, idx[u]);
    }
}

```

```

return v;
}
int up(int u, int d) {
    while (d) {
        if (dep[u] - dep[top[u]] < d) {
            d -= dep[u] - dep[top[u]];
            u = top[u];
        } else return ridx[idx[u] - d];
        u = fa[u]; --d;
    }
    return u;
}
int finds(int u, int rt) { // find u in which sub-tree of rt
    while (top[u] != top[rt]) {
        u = top[u];
        if (fa[u] == rt) return u;
        u = fa[u];
    }
    return ridx[idx[rt] + 1];
}
}

```

4.7 Bipartite Matching

```

struct MaxMatch {
    int n;
    vector<int> G[maxn];
    int vis[maxn], left[maxn], clk;

    void init(int n) {
        this->n = n;
        FOR (i, 0, n + 1) G[i].clear();
        memset(left, -1, sizeof left);
        memset(vis, -1, sizeof vis);
    }

    bool dfs(int u) {
        for (int v: G[u])
            if (vis[v] != clk) {
                vis[v] = clk;
                if (left[v] == -1 || dfs(left[v])) {
                    left[v] = u;
                    return true;
                }
            }
        return false;
    }

    int match() {
        int ret = 0;
        for (clk = 0; clk <= n; ++clk)
            if (dfs(clk)) ++ret;
        return ret;
    }
} MM;

//////////
// max weight: KM
//////////
namespace R {
    const int maxn = 300 + 10;
    int n, m;
    int left[maxn], L[maxn], R[maxn];
    int w[maxn][maxn], slack[maxn];
    bool visL[maxn], visR[maxn];

    bool dfs(int u) {
        visL[u] = true;
        FOR (v, 0, m) {
            if (visR[v]) continue;
            int t = L[u] + R[v] - w[u][v];
            if (t == 0) {
                visR[v] = true;
                if (left[v] == -1 || dfs(left[v])) {
                    left[v] = u;
                    return true;
                }
            }
        }
    }
}

```

```

} else slack[v] = min(slack[v], t);
}
return false;
}

int go() {
    memset(left, -1, sizeof left);
    memset(R, 0, sizeof R);
    memset(L, 0, sizeof L);
    FOR (i, 0, n)
        FOR (j, 0, m)
            L[i] = max(L[i], w[i][j]);

    FOR (i, 0, n) {
        memset(slack, 0x3f, sizeof slack);
        while (1) {
            memset(visL, 0, sizeof visL); memset(visR, 0,
                sizeof visR);
            if (dfs(i)) break;
            int d = 0x3f3f3f3f;
            FOR (j, 0, m) if (!visR[j]) d = min(d, slack[j]);
            FOR (j, 0, n) if (visL[j]) L[j] -= d;
            FOR (j, 0, m) if (visR[j]) R[j] += d; else slack[j] -= d;
        }
    }
    int ret = 0;
    FOR (i, 0, m) if (left[i] != -1) ret += w[left[i]][i];
    return ret;
}
}

```

4.8 Virtual Tree

```

void go(vector<int>& V, int& k) {
    int u = V[k]; f[u] = 0;
    dbg(u, k);
    for (auto& e: G[u]) {
        int v = e.to;
        if (v == pa[u][0]) continue;
        while (k + 1 < V.size()) {
            int to = V[k + 1];
            if (in[to] <= out[v]) {
                go(V, ++k);
                if (key[to]) f[u] += w[to];
                else f[u] += min(f[to], (LL)w[to]);
            } else break;
        }
    }
    dbg(u, f[u]);
}

inline bool cmp(int a, int b) { return in[a] < in[b]; }
LL solve(vector<int>& V) {
    static vector<int> a; a.clear();
    for (int& x: V) a.push_back(x);
    sort(a.begin(), a.end(), cmp);
    FOR (i, 1, a.size())
        a.push_back(lca(a[i], a[i - 1]));
    a.push_back(1);
    sort(a.begin(), a.end(), cmp);
    a.erase(unique(a.begin(), a.end(), a.end()));
    dbg(a);
    int tmp; go(a, tmp = 0);
    return f[1];
}

```

4.9 Euler Tour

```

int S[N << 1], top;
Edge edges[N << 1];
set<int> G[N];

void DFS(int u) {
    S[top++] = u;
}

```

```

    for (int eid: G[u]) {
        int v = edges[eid].get_other(u);
        G[u].erase(eid);
        G[v].erase(eid);
        DFS(v);
        return;
    }
}

void fleury(int start) {
    int u = start;
    top = 0; path.clear();
    S[top++] = u;
    while (top) {
        u = S[--top];
        if (!G[u].empty())
            DFS(u);
        else path.push_back(u);
    }
}

4.10 SCC, 2-SAT

int n, m;
vector<int> G[N], rG[N], vs;
int used[N], cmp[N];

void add_edge(int from, int to) {
    G[from].push_back(to);
    rG[to].push_back(from);
}

void dfs(int v) {
    used[v] = true;
    for (int u: G[v]) {
        if (!used[u])
            dfs(u);
    }
    vs.push_back(v);
}

void rdfs(int v, int k) {
    used[v] = true;
    cmp[v] = k;
    for (int u: rG[v])
        if (!used[u])
            rdfs(u, k);
}

int scc() {
    memset(used, 0, sizeof(used));
    vs.clear();
    for (int v = 0; v < n; ++v)
        if (!used[v]) dfs(v);
    memset(used, 0, sizeof(used));
    int k = 0;
    for (int i = (int) vs.size() - 1; i >= 0; --i)
        if (!used[vs[i]]) rdfs(vs[i], k++);
    return k;
}

int main() {
    cin >> n >> m;
    n = 2;
    for (int i = 0; i < m; ++i) {
        int a, b; cin >> a >> b;
        add_edge(a - 1, (b - 1) ^ 1);
        add_edge(b - 1, (a - 1) ^ 1);
    }
    scc();
    for (int i = 0; i < n; i += 2) {
        if (cmp[i] == cmp[i + 1]) {
            puts("NIE");
            return 0;
        }
    }
    for (int i = 0; i < n; i += 2) {
        if (cmp[i] > cmp[i + 1]) printf("%d\n", i + 1);
    }
}

```

```

        else printf("%d\n", i + 2);
    }
}

4.11 Topological Sort

vector<int> toporder(int n) {
    vector<int> orders;
    queue<int> q;
    for (int i = 0; i < n; i++)
        if (!deg[i]) {
            q.push(i);
            orders.push_back(i);
        }
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v: G[u])
            if (!--deg[v]) {
                q.push(v);
                orders.push_back(v);
            }
    }
    return orders;
}

```

4.12 General Matching

```

// O(n^3)
vector<int> G[N];
int fa[N], mt[N], pre[N], mk[N];
int lca_clk, lca_mk[N];
pair<int, int> ce[N];
void connect(int u, int v) {
    mt[u] = v;
    mt[v] = u;
}

int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }
void flip(int s, int u) {
    if (s == u) return;
    if (mk[u] == 2) {
        int v1 = ce[u].first, v2 = ce[u].second;
        flip(mt[u], v1);
        flip(s, v2);
        connect(v1, v2);
    } else {
        flip(s, pre[mt[u]]);
        connect(pre[mt[u]], mt[u]);
    }
}

int get_lca(int u, int v) {
    lca_clk++;
    for (u = find(u), v = find(v); ; u = find(pre[u]), v = find(pre[v])) {
        if (u && lca_mk[u] == lca_clk) return u;
        lca_mk[u] = lca_clk;
        if (v && lca_mk[v] == lca_clk) return v;
        lca_mk[v] = lca_clk;
    }
}

void access(int u, int p, const pair<int, int>& c, vector<int>& q) {
    for (u = find(u); u != p; u = find(pre[u])) {
        if (mk[u] == 2) {
            ce[u] = c;
            q.push_back(u);
        }
        fa[find(u)] = find(p);
    }
}

bool aug(int s) {
    fill(mk, mk + n + 1, 0);
    fill(pre, pre + n + 1, 0);
    iota(fa, fa + n + 1, 0);
    vector<int> q = {s};
    mk[s] = 1;
}

```

```

int t = 0;
for (int t = 0; t < (int) q.size(); ++t) {
    // q size can be changed
    int u = q[t];
    for (int &v: G[u]) {
        if (find(v) == find(u)) continue;
        if (!mk[v] && !mt[v]) {
            flip(s, u);
            connect(u, v);
            return true;
        } else if (!mk[v]) {
            int w = mt[v];
            mk[v] = 2; mk[w] = 1;
            pre[w] = v; pre[v] = u;
            q.push_back(w);
        } else if (mk[find(v)] == 1) {
            int p = get_lca(u, v);
            access(u, p, {u, v}, q);
            access(v, p, {v, u}, q);
        }
    }
}

return false;
}

int match() {
    fill(mt + 1, mt + n + 1, 0);
    lca_clk = 0;
    int ans = 0;
    FOR (i, 1, n + 1)
        if (!mt[i]) ans += aug(i);
    return ans;
}

4.13 Tarjan

// articulation points
// note that the graph might be disconnected
int dfn[N], low[N], clk;
void init() { clk = 0; memset(dfn, 0, sizeof dfn); }
void tarjan(int u, int fa) {
    low[u] = dfn[u] = ++clk;
    int cc = fa != -1;
    for (int& v: G[u]) {
        if (v == fa) continue;
        if (!dfn[v]) {
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            cc += low[v] >= dfn[u];
        } else low[u] = min(low[u], dfn[v]);
    }
    if (cc > 1) // ...
}

// bridge
// note that the graph might have multiple edges or be disconnected
int dfn[N], low[N], clk;
void init() { memset(dfn, 0, sizeof dfn); clk = 0; }
void tarjan(int u, int fa) {
    low[u] = dfn[u] = ++clk;
    int _fst = 0;
    for (E& e: G[u]) {
        int v = e.to; if (v == fa && ++_fst == 1) continue;
        if (!dfn[v]) {
            tarjan(v, u);
            if (low[v] > dfn[u]) // ...
                low[u] = min(low[u], low[v]);
            else low[u] = min(low[u], dfn[v]);
        }
    }
}

// scc
int low[N], dfn[N], clk, B, bl[N];
vector<int> bcc[N];
void init() { B = clk = 0; memset(dfn, 0, sizeof dfn); }
}

```

```

void tarjan(int u) {
    static int st[N], p;
    static bool in[N];
    dfn[u] = low[u] = ++clk;
    st[p++] = u; in[u] = true;
    for (int& v: G[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (in[v]) low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]) {
        while (1) {
            int x = st[--p]; in[x] = false;
            bl[x] = B; bcc[B].push_back(x);
            if (x == u) break;
        }
        ++B;
    }
}

```

4.14 Bi-connected Components, Block-cut Tree

```

// Array size should be 2 * N
// Single edge also counts as bi-connected comp
// Use |V| <= |E| to filter
struct E { int to, nxt; } e[N];
int hd[N], ecnt;
void addedge(int u, int v) {
    e[ecnt] = {v, hd[u]};
    hd[u] = ecnt++;
}
int low[N], dfn[N], clk, B, bno[N];
vector<int> bc[N], be[N];
bool vise[N];
void init() {
    memset(vise, 0, sizeof vise);
    memset(hd, -1, sizeof hd);
    memset(dfn, 0, sizeof dfn);
    memset(bno, -1, sizeof bno);
    B = clk = ecnt = 0;
}

```

```

void tarjan(int u, int feid) {
    static int st[N], p;
    static auto add = [&](int x) {
        if (bno[x] != B) { bno[x] = B; bc[B].push_back(x); }
    };
    low[u] = dfn[u] = ++clk;
    for (int i = hd[u]; ~i; i = e[i].nxt) {
        if ((feid ^ i) == 1) continue;
        if (!vise[i]) { st[p++] = i; vise[i] = vise[i ^ 1] = true; }
        int v = e[i].to;
        if (!dfn[v]) {
            tarjan(v, i);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u]) {
                bc[B].clear(); be[B].clear();
                while (1) {
                    int eid = st[--p];
                    add(e[eid].to); add(e[eid ^ 1].to);
                    be[B].push_back(eid);
                    if ((eid ^ i) <= 1) break;
                }
                ++B;
            }
        } else low[u] = min(low[u], dfn[v]);
    }
}

```

```

//////////
// block-cut tree
// cactus -> block-cut tree

```

```

// N >= |E| * 2
//////////

vector<int> G[N];
int nn;

struct E { int to, nxt; };
namespace C {
    E e[N * 2];
    int hd[N], ecnt;
    void addedge(int u, int v) {
        e[ecnt] = {v, hd[u]};
        hd[u] = ecnt++;
    }
    int idx[N], clk, fa[N];
    bool ring[N];
    void init() { ecnt = 0; memset(hd, -1, sizeof hd); clk = 0; }
    void dfs(int u, int feid) {
        idx[u] = ++clk;
        for (int i = hd[u]; ~i; i = e[i].nxt) {
            if ((i ^ feid) == 1) continue;
            int v = e[i].to;
            if (!idx[v]) {
                fa[v] = u; ring[u] = false;
                dfs(v, i);
                if (!ring[u]) { G[u].push_back(v); G[v].push_back(u); }
            } else if (idx[v] < idx[u]) {
                ++nn;
                G[nn].push_back(v); G[v].push_back(nn); // put
                // the root of the cycle in the front
                for (int x = u; x != v; x = fa[x]) {
                    ring[x] = true;
                    G[nn].push_back(x); G[x].push_back(nn);
                }
                ring[v] = true;
            }
        }
    }
}

```

4.15 Minimum Directed Spanning Tree

```

// edges will be modified
vector<E> edges;
int in[N], id[N], pre[N], vis[N];
// a copy of n is needed
LL zl_tree(int rt, int n) {
    LL ans = 0;
    int v, _n = n;
    while (1) {
        fill(in, in + n, INF);
        for (E &e: edges) {
            if (e.u != e.v && e.w < in[e.v]) {
                pre[e.v] = e.u;
                in[e.v] = e.w;
            }
        }
        FOR (i, 0, n) if (i != rt && in[i] == INF) return -1;
        int tn = 0;
        fill(id, id + _n, -1); fill(vis, vis + _n, -1);
        in[rt] = 0;
        FOR (i, 0, n) {
            ans += in[v = i];
            while (vis[v] != i && id[v] == -1 && v != rt) {
                vis[v] = i; v = pre[v];
            }
            if (v != rt && id[v] == -1) {
                for (int u = pre[v]; u != v; u = pre[u]) id[u] = tn;
                id[v] = tn++;
            }
        }
        if (tn == 0) break;
        FOR (i, 0, n) if (id[i] == -1) id[i] = tn++;
    }
}

```

```

for (int i = 0; i < (int) edges.size(); ) {
    auto &e = edges[i];
    v = e.v;
    e.u = id[e.u]; e.v = id[e.v];
    if (e.u != e.v) { e.w -= in[v]; i++; }
    else { swap(e, edges.back()); edges.pop_back(); }
}
n = tn; rt = id[rt];
return ans;
}

```

4.16 Cycles

```

// refer to cheatsheet for elaboration
LL cycle4() {
    LL ans = 0;
    iota(kth, kth + n + 1, 0);
    sort(kth, kth + n, [&](int x, int y) { return deg[x] < deg[y]; });
    FOR (i, 1, n + 1) rk[kth[i]] = i;
    FOR (u, 1, n + 1)
        for (int v: G[u])
            if (rk[v] > rk[u]) key[u].push_back(v);
    FOR (u, 1, n + 1) {
        for (int v: G[u])
            for (int w: key[v])
                if (rk[w] > rk[u]) ans += cnt[w]++;
            for (int v: G[u])
                for (int w: key[v])
                    if (rk[w] > rk[u]) --cnt[w];
    }
    return ans;
}
int cycle3() {
    int ans = 0;
    for (E &e: edges) { deg[e.u]++; deg[e.v]++; }
    for (E &e: edges) {
        if (deg[e.u] < deg[e.v] || (deg[e.u] == deg[e.v] && e.u < e.v))
            G[e.u].push_back(e.v);
        else G[e.v].push_back(e.u);
    }
    FOR (x, 1, n + 1) {
        for (int y: G[x]) p[y] = x;
        for (int y: G[x]) for (int z: G[y]) if (p[z] == x) ans++;
    }
    return ans;
}

```

4.17 Dominator Tree

```

vector<int> G[N], rG[N];
vector<int> dt[N];

namespace tl {
    int fa[N], idx[N], clk, ridx[N];
    int c[N], best[N], semi[N], idom[N];
    void init(int n) {
        clk = 0;
        fill(c, c + n + 1, -1);
        FOR (i, 1, n + 1) dt[i].clear();
        FOR (i, 1, n + 1) semi[i] = best[i] = i;
        fill(idx, idx + n + 1, 0);
    }
    void dfs(int u) {
        idx[u] = ++clk; ridx[clk] = u;
        for (int& v: G[u]) if (!idx[v]) { fa[v] = u; dfs(v); }
    }
    int fix(int x) {
        if (c[x] == -1) return x;
        int &f = c[x], rt = fix(f);
        if (idx[semi[best[x]]] > idx[semi[best[f]]]) best[x] = best[f];
        return f = rt;
    }
}

```

```

}
void go(int rt) {
    dfs(rt);
    FORD (i, clk, 1) {
        int x = ridx[i], mn = clk + 1;
        for (int& u: rG[X]) {
            if (!idx[u]) continue; // reaching all might
                                   // not be possible
            fix(u); mn = min(mn, idx[semi[best[u]]]);
        }
        c[x] = fa[x];
        dt[semi[x] = ridx[mn]].push_back(x);
        x = ridx[i - 1];
        for (int& u: dt[x]) {
            fix(u);
            if (semi[best[u]] != x) idom[u] = best[u];
            else idom[u] = x;
        }
        dt[x].clear();
    }

    FOR (i, 2, clk + 1) {
        int u = ridx[i];
        if (idom[u] != semi[u]) idom[u] = idom[idom[u]];
        dt[idom[u]].push_back(u);
    }
}
}

```

4.18 Global Minimum Cut

```

struct StoerWanger {
    LL n, vis[N];
    LL dist[N];
    LL g[N][N];

    void init(int nn, LL w[N][N]) {
        n = nn;
        FOR (i, 1, n + 1) FOR (j, 1, n + 1)
            g[i][j] = w[i][j];
        memset(dist, 0, sizeof(dist));
    }

    LL min_cut_phase(int clk, int &x, int &y) {
        int t;
        vis[t = 1] = clk;
        FOR (i, 1, n + 1) if (vis[i] != clk)
            dist[i] = g[1][i];
        FOR (i, 1, n) {
            x = t; t = 0;
            FOR (j, 1, n + 1)
                if (vis[j] != clk && (!t || dist[j] > dist[t]))
                    t = j;
            vis[t] = clk;
            FOR (j, 1, n + 1) if (vis[j] != clk)
                dist[j] += g[t][j];
        }
        y = t;
        return dist[t];
    }

    void merge(int x, int y) {
        if (x > y) swap(x, y);
        FOR (i, 1, n + 1)
            if (i != x && i != y) {
                g[i][x] += g[i][y];
                g[x][i] += g[i][y];
            }
        if (y == n) return;
        FOR (i, 1, n) if (i != y) {
            swap(g[i][y], g[i][n]);
            swap(g[y][i], g[n][i]);
        }
    }

    LL go() {

```

```

        LL ret = INF;
        memset(vis, 0, sizeof vis);
        for (int i = 1, x, y; n > 1; ++i, --n) {
            ret = min(ret, min_cut_phase(i, x, y));
            merge(x, y);
        }
        return ret;
    }
} SW;

```

5 Geometry

5.1 2D Basics

```

int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
struct L;
struct P;
typedef P V;
struct P {
    LD x, y;
    explicit P(LD x = 0, LD y = 0): x(x), y(y) {}
    explicit P(const L& l);
};
struct L {
    P s, t;
    L() {}
    L(P s, P t): s(s), t(t) {}
};

P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
P operator * (const P& a, LD k) { return P(a.x * k, a.y * k); }
P operator / (const P& a, LD k) { return P(a.x / k, a.y / k); }
inline bool operator < (const P& a, const P& b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
}

bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
P::P(const L& l) { *this = l.t - l.s; }
ostream& operator << (ostream& os, const P& p) {
    return (os << "(" << p.x << ", " << p.y << ")");
}
istream& operator >> (istream& is, P& p) {
    return (is >> p.x >> p.y);
}

LD dist(const P& p) { return sqrt(p.x * p.x + p.y * p.y); }
LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
LD det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
LD cross(const P& s, const P& t, const P& o = P()) { return det(s - o, t - o); }

```

5.2 Polar angle sort

```

int quad(P p) {
    int x = sgn(p.x), y = sgn(p.y);
    if (x > 0 && y >= 0) return 1;
    if (x <= 0 && y > 0) return 2;
    if (x < 0 && y <= 0) return 3;
    if (x >= 0 && y < 0) return 4;
    assert(0);
}

struct cmp_angle {
    P p;
    bool operator () (const P& a, const P& b) {
        int qa = quad(a - p), qb = quad(b - p);
        if (qa != qb) return qa < qb; // compare quad
        int d = sgn(cross(a, b, p));
        if (d) return d > 0;
        return dist(a - p) < dist(b - p);
    }
}

```

```

};

5.3 Segments, lines
bool parallel(const L& a, const L& b) {
    return !sgn(det(P(a), P(b)));
}
bool l_eq(const L& a, const L& b) {
    return parallel(a, b) && parallel(L(a.s, b.t), L(b.s, a.t));
}
// counter-clockwise r radius
P rotation(const P& p, const LD& r) { return P(p.x * cos(r) - p.y * sin(r), p.x * sin(r) + p.y * cos(r)); }
P RotateCCW90(const P& p) { return P(-p.y, p.x); }
P RotateCW90(const P& p) { return P(p.y, -p.x); }
V normal(const V& v) { return V(-v.y, v.x) / dist(v); }
// inclusive: <=0; exclusive: <0
bool p_on_seg(const P& p, const L& seg) {
    P a = seg.s, b = seg.t;
    return !sgn(det(p - a, b - a)) && sgn(dot(p - a, p - b)) <= 0;
}
LD dist_to_line(const P& p, const L& l) {
    return fabs(cross(l.s, l.t, p)) / dist(l);
}
LD dist_to_seg(const P& p, const L& l) {
    if (l.s == l.t) return dist(p - l);
    V vs = p - l.s, vt = p - l.t;
    if (sgn(dot(l, vs)) < 0) return dist(vs);
    else if (sgn(dot(l, vt)) > 0) return dist(vt);
    else return dist_to_line(p, l);
}
// make sure they have intersection in advance
P l_intersection(const L& a, const L& b) {
    LD s1 = det(P(a), b.s - a.s), s2 = det(P(a), b.t - a.s);
    return (b.s * s2 - b.t * s1) / (s2 - s1);
}
LD angle(const V& a, const V& b) {
    LD r = asin(fabs(det(a, b)) / dist(a) / dist(b));
    if (sgn(dot(a, b)) < 0) r = PI - r;
    return r;
}
// 1: proper; 2: improper
int s_l_cross(const L& seg, const L& line) {
    int d1 = sgn(cross(line.s, line.t, seg.s));
    int d2 = sgn(cross(line.s, line.t, seg.t));
    if ((d1 ^ d2) == -2) return 1; // proper
    if (d1 == 0 || d2 == 0) return 2;
    return 0;
}
// 1: proper; 2: improper
int s_cross(const L& a, const L& b, P& p) {
    int d1 = sgn(cross(a.t, b.s, a.s)), d2 = sgn(cross(a.t, b.t, a.s));
    int d3 = sgn(cross(b.t, a.s, b.s)), d4 = sgn(cross(b.t, a.t, b.s));
    if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) { p = l_intersection(a, b); return 1; }
    if (!d1 && p_on_seg(b.s, a)) { p = b.s; return 2; }
    if (!d2 && p_on_seg(b.t, a)) { p = b.t; return 2; }
    if (!d3 && p_on_seg(a.s, b)) { p = a.s; return 2; }
    if (!d4 && p_on_seg(a.t, b)) { p = a.t; return 2; }
    return 0;
}

```

5.4 Polygons

```

typedef vector<P> S;
// 0 = outside, 1 = inside, -1 = on border
int inside(const S& s, const P& p) {
    int cnt = 0;
    FOR (i, 0, s.size()) {
        P a = s[i], b = s[nxt(i)];
        if (p_on_seg(p, L(a, b))) return -1;
    }
}

```



```

    if (sgn(a.y - b.y) <= 0) swap(a, b);
    if (sgn(p.y - a.y) > 0) continue;
    if (sgn(p.y - b.y) <= 0) continue;
    cnt += sgn(cross(b, a, p)) > 0;
}
return bool(cnt & 1);
}
// can be negative
LD polygon_area(const S& s) {
    LD ret = 0;
    FOR (i, 1, (LL)s.size() - 1)
        ret += cross(s[i], s[i + 1], s[0]);
    return ret / 2;
}
// duplicate points are not allowed
// s is subject to change
const int MAX_N = 1000;
S convex_hull(S& s) {
    // assert(s.size() >= 3);
    sort(s.begin(), s.end());
    S ret(MAX_N * 2);
    int sz = 0;
    FOR (i, 0, s.size()) {
        while (sz > 1 && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) --sz;
        ret[sz++] = s[i];
    }
    int k = sz;
    FORD (i, (LL)s.size() - 2, -1) {
        while (sz > k && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) --sz;
        ret[sz++] = s[i];
    }
    ret.resize(sz - (s.size() > 1));
    return ret;
}
// centroid
P ComputeCentroid(const vector<P> &p) {
    P c(0, 0);
    LD scale = 6.0 * polygon_area(p);
    for (unsigned i = 0; i < p.size(); i++) {
        unsigned j = (i + 1) % p.size();
        c = c + (p[i] + p[j]) * (p[i].x * p[j].y - p[j].x * p[i].y);
    }
    return c / scale;
}
// Rotating Calipers, find convex hull first
LD rotatingCalipers(vector<P> &qs) {
    int n = qs.size();
    if (n == 2)
        return dist(qs[0] - qs[1]);
    int i = 0, j = 0;
    FOR (k, 0, n) {
        if (!qs[i] < qs[k]) i = k;
        if (qs[j] < qs[k]) j = k;
    }
    LD res = 0;
    int si = i, sj = j;
    while (i != sj || j != si) {
        res = max(res, dist(qs[i] - qs[j]));
        if (sgn(cross(qs[(i+1)%n] - qs[i], qs[(j+1)%n] - qs[j])) < 0)
            i = (i + 1) % n;
        else j = (j + 1) % n;
    }
    return res;
}
}

```

5.5 Half-plane intersection

```

struct LV {
    P p, v; LD ang;
    LV() {}
    LV(P s, P t): p(s), v(t - s) { ang = atan2(v.y, v.x); }
}; // 00000000

```

```

bool operator < (const LV &a, const LV& b) { return a.ang < b.ang; }
bool on_left(const LV& l, const P& p) { return sgn(cross(l.v, p - l.p)) >= 0; }
P l_intersection(const LV& a, const LV& b) {
    P u = a.p - b.p; LD t = cross(b.v, u) / cross(a.v, b.v);
    return a.p + a.v * t;
}
S half_plane_intersection(vector<LV>& L) {
    int n = L.size(), fi, la;
    sort(L.begin(), L.end());
    vector<P> p(n); vector<LV> q(n);
    q[fi = la = 0] = L[0];
    FOR (i, 1, n) {
        while (fi < la && !on_left(L[i], p[la - 1])) la--;
        while (fi < la && !on_left(L[i], p[fi])) fi++;
        q[++la] = L[i];
        if (sgn(cross(q[la].v, q[la - 1].v)) == 0) {
            la--;
            if (on_left(q[la], L[i].p)) q[la] = L[i];
        }
        if (fi < la) p[la - 1] = l_intersection(q[la - 1], q[la]);
    }
    while (fi < la && !on_left(q[fi], p[la - 1])) la--;
    if (la - fi <= 1) return vector<P>();
    p[la] = l_intersection(q[la], q[fi]);
    return vector<P>(p.begin() + fi, p.begin() + la + 1);
}
S convex_intersection(const vector<P> &v1, const vector<P> &v2) {
    vector<LV> h; int n = v1.size(), m = v2.size();
    FOR (i, 0, n) h.push_back(LV(v1[i], v1[(i + 1) % n]));
    FOR (i, 0, m) h.push_back(LV(v2[i], v2[(i + 1) % m]));
    return half_plane_intersection(h);
}

```

5.6 Circles

```

struct C {
    P p; LD r;
    C(LD x = 0, LD y = 0, LD r = 0): p(x, y), r(r) {}
    C(P p, LD r): p(p), r(r) {}
};
P compute_circle_center(P a, P b, P c) {
    b = (a + b) / 2;
    c = (a + c) / 2;
    return l_intersection({b, b + RotateCW90(a - b)}, {c, c + RotateCW90(a - c)});
}
// intersections are clockwise subject to center
vector<P> c_l_intersection(const L& l, const C& c) {
    vector<P> ret;
    P b(l), a = l.s - c.p;
    LD x = dot(b, b), y = dot(a, b), z = dot(a, a) - c.r * c.r;
    LD d = y * y - x * z;
    if (sgn(D) < 0) return ret;
    ret.push_back(c.p + a + b * (-y + sqrt(D + eps)) / x);
    if (sgn(D) > 0) ret.push_back(c.p + a + b * (-y - sqrt(D)) / x);
    return ret;
}
vector<P> c_c_intersection(C a, C b) {
    vector<P> ret;
    LD d = dist(a.p - b.p);
    if (sgn(d) == 0 || sgn(d - (a.r + b.r)) > 0 || sgn(d + min(a.r, b.r) - max(a.r, b.r)) < 0)
        return ret;
    LD x = (d * d - b.r * b.r + a.r * a.r) / (2 * d);
    LD y = sqrt(a.r * a.r - x * x);
    P v = (b.p - a.p) / d;
    ret.push_back(a.p + v * x + RotateCCW90(v) * y);
    if (sgn(y) > 0) ret.push_back(a.p + v * x - RotateCCW90(v) *

```

```

        y);
    return ret;
}
// 1: inside, 2: internally tangent
// 3: intersect, 4: ext tangent 5: outside
int c_c_relation(const C& a, const C& v) {
    LD d = dist(a.p - v.p);
    if (sgn(d - a.r - v.r) > 0) return 5;
    if (sgn(d - a.r - v.r) == 0) return 4;
    LD l = fabs(a.r - v.r);
    if (sgn(d - l) > 0) return 3;
    if (sgn(d - l) == 0) return 2;
    if (sgn(d - l) < 0) return 1;
}
// circle triangle intersection
// abs might be needed
LD sector_area(const P& a, const P& b, LD r) {
    LD th = atan2(a.y, a.x) - atan2(b.y, b.x);
    while (th <= 0) th += 2 * PI;
    while (th > 2 * PI) th -= 2 * PI;
    th = min(th, 2 * PI - th);
    return r * r * th / 2;
}
LD c_tri_area(P a, P b, P center, LD r) {
    a = a - center; b = b - center;
    int ina = sgn(dist(a) - r) < 0, inb = sgn(dist(b) - r) < 0;
    // dbg(a, b, ina, inb);
    if (ina && inb) {
        return fabs(cross(a, b)) / 2;
    } else {
        auto p = c_l_intersection(L(a, b), C(0, 0, r));
        if (ina ^ inb) {
            auto cr = p_on_seg(p[0], L(a, b)) ? p[0] : p[1];
            if (ina) return sector_area(b, cr, r) + fabs(cross(a, cr)) / 2;
            else return sector_area(a, cr, r) + fabs(cross(b, cr)) / 2;
        } else {
            if ((int) p.size() == 2 && p_on_seg(p[0], L(a, b))) {
                if (dist(p[0] - a) > dist(p[1] - a)) swap(p[0], p[1]);
                return sector_area(a, p[0], r) + sector_area(p[1], b, r) + fabs(cross(p[0], p[1])) / 2;
            } else return sector_area(a, b, r);
        }
    }
}
typedef vector<P> S;
LD c_poly_area(S poly, const C& c) {
    LD ret = 0; int n = poly.size();
    FOR (i, 0, n) {
        int t = sgn(cross(poly[i] - c.p, poly[(i + 1) % n] - c.p));
        if (t) ret += t * c_tri_area(poly[i], poly[(i + 1) % n], c.p, c.r);
    }
    return ret;
}

```

5.7 Circle Union

```

// version 1
// union O(n^3 log n)
struct CV {
    LD yl, yr, ym; C o; int type;
    CV() {}
    CV(LD yl, LD yr, LD ym, C c, int t)
        : yl(yl), yr(yr), ym(ym), type(t), o(c) {}
};
pair<LD, LD> c_point_eval(const C& c, LD x) {
    LD d = fabs(c.p.x - x), h = rt(sq(c.r) - sq(d));
    return {c.p.y - h, c.p.y + h};
}

```

```

pair<CV, CV> pairwise_curves(const C& c, LD xl, LD xr) {
    LD yl1, yl2, yr1, yr2, ym1, ym2;
    tie(yl1, yl2) = c_point_eval(c, xl);
    tie(ym1, ym2) = c_point_eval(c, (xl + xr) / 2);
    tie(yr1, yr2) = c_point_eval(c, xr);
    return {CV(yl1, yr1, ym1, c, 1), CV(yl2, yr2, ym2, c, -1)};
}
bool operator < (const CV& a, const CV& b) { return a.ym < b.ym; }
LD cv_area(const CV& v, LD xl, LD xr) {
    LD l = rt(sq(xr - xl) + sq(v.yr - v.yl));
    LD d = rt(sq(v.o.r) - sq(l / 2));
    LD ang = atan(l / d / 2);
    return ang * sq(v.o.r) - d * l / 2;
}
LD circle_union(const vector<C>& cs) {
    int n = cs.size();
    vector<LD> xs;
    FOR (i, 0, n) {
        xs.push_back(cs[i].p.x - cs[i].r);
        xs.push_back(cs[i].p.x);
        xs.push_back(cs[i].p.x + cs[i].r);
        FOR (j, i + 1, n) {
            auto pts = c_c_intersection(cs[i], cs[j]);
            for (auto& p: pts) xs.push_back(p.x);
        }
    }
    sort(xs.begin(), xs.end());
    xs.erase(unique(xs.begin(), xs.end(), [](LD x, LD y) {
        return sgn(x - y) == 0; }), xs.end());
    LD ans = 0;
    FOR (i, 0, (int) xs.size() - 1) {
        LD xl = xs[i], xr = xs[i + 1];
        vector<CV> intv;
        FOR (k, 0, n) {
            auto& c = cs[k];
            if (sgn(c.p.x - c.r - xl) <= 0 && sgn(c.p.x + c.r -
                xr) >= 0) {
                auto t = pairwise_curves(c, xl, xr);
                intv.push_back(t.first); intv.push_back(t.second);
            }
        }
        sort(intv.begin(), intv.end());
        vector<LD> areas(intv.size());
        FOR (i, 0, intv.size()) areas[i] = cv_area(intv[i], xl,
            xr);
        int cc = 0;
        FOR (i, 0, intv.size()) {
            if (cc > 0) {
                ans += (intv[i].yl - intv[i - 1].yl + intv[i].yr
                    - intv[i - 1].yr) * (xr - xl) / 2;
                ans += intv[i - 1].type * areas[i - 1];
                ans -= intv[i].type * areas[i];
            }
            cc += intv[i].type;
        }
    }
    return ans;
}
// version 2 (k-cover, O(n^2 log n))
inline LD angle(const P &p) { return atan2(p.y, p.x); }
// Points on circle
// p is coordinates relative to c
struct CP {
    P p;
    LD a;
    int t;
    CP() {}
    CP(P p, LD a, int t) : p(p), a(a), t(t) {}
};
bool operator<(const CP &u, const CP &v) { return u.a < v.a; }
LD cv_area(LD r, const CP &q1, const CP &q2) {
    return (r * r * (q2.a - q1.a) - cross(q1.p, q2.p)) / 2;
}

```

```

}
LD ans[N];
void circle_union(const vector<C> &cs) {
    int n = cs.size();
    FOR (i, 0, n) {
        // same circle, only the first one counts
        bool ok = true;
        FOR (j, 0, i)
            if (sgn(cs[i].r - cs[j].r) == 0 && cs[i].p == cs[j].p) {
                ok = false;
                break;
            }
        if (!ok) continue;
        auto &c = cs[i];
        vector<CP> ev;
        int belong_to = 0;
        P bound = c.p + P(-c.r, 0);
        ev.emplace_back(bound, -PI, 0);
        ev.emplace_back(bound, PI, 0);
        FOR (j, 0, n) {
            if (i == j) continue;
            if (c_c_relation(c, cs[j]) <= 2) {
                if (sgn(cs[j].r - c.r) >= 0) // totally covered
                    belong_to++;
                continue;
            }
            auto its = c_c_intersection(c, cs[j]);
            if (its.size() == 2) {
                P p = its[1] - c.p, q = its[0] - c.p;
                LD a = angle(p), b = angle(q);
                if (sgn(a - b) > 0) {
                    ev.emplace_back(p, a, 1);
                    ev.emplace_back(bound, PI, -1);
                    ev.emplace_back(bound, -PI, 1);
                    ev.emplace_back(q, b, -1);
                } else {
                    ev.emplace_back(p, a, 1);
                    ev.emplace_back(q, b, -1);
                }
            }
        }
        sort(ev.begin(), ev.end());
        int cc = ev[0].t;
        FOR (j, 1, ev.size()) {
            int t = cc + belong_to;
            ans[t] += cross(ev[j - 1].p + c.p, ev[j].p + c.p) / 2;
            ans[t] += cv_area(c.r, ev[j - 1], ev[j]);
            cc += ev[j].t;
        }
    }
}

```

5.8 Minimum Covering Circle

```

P compute_circle_center(P a, P b) { return (a + b) / 2; }
bool p_in_circle(const P& p, const C& c) {
    return sgn(dist(p - c.p) - c.r) <= 0;
}
C min_circle_cover(const vector<P> &in) {
    vector<P> a(in.begin(), in.end());
    dbg(a.size());
    random_shuffle(a.begin(), a.end());
    P c = a[0]; LD r = 0; int n = a.size();
    FOR (i, 1, n) if (!p_in_circle(a[i], {c, r})) {
        c = a[i]; r = 0;
        FOR (j, 0, i) if (!p_in_circle(a[j], {c, r})) {
            c = compute_circle_center(a[i], a[j]);
            r = dist(a[j] - c);
            FOR (k, 0, j) if (!p_in_circle(a[k], {c, r})) {
                c = compute_circle_center(a[i], a[j], a[k]);
                r = dist(a[k] - c);
            }
        }
    }
}

```

```

}
return {c, r};
}

```

5.9 Circle Inversion

```

C inv(C c, const P& o) {
    LD d = dist(c.p - o);
    assert(sgn(d) != 0);
    LD a = 1 / (d - c.r);
    LD b = 1 / (d + c.r);
    c.r = (a - b) / 2 * R2;
    c.p = o + (c.p - o) * ((a + b) * R2 / 2 / d);
    return c;
}

```

5.10 3D Basics

```

struct P;
struct L;
typedef P V;
struct P {
    LD x, y, z;
    explicit P(LD x = 0, LD y = 0, LD z = 0): x(x), y(y), z(z) {}
    explicit P(const L& l);
};
struct L {
    P s, t;
    L() {}
    L(P s, P t): s(s), t(t) {}
};
struct F {
    P a, b, c;
    F() {}
    F(P a, P b, P c): a(a), b(b), c(c) {}
};
P operator + (const P& a, const P& b) {}
P operator - (const P& a, const P& b) {}
P operator * (const P& a, LD k) {}
P operator / (const P& a, LD k) {}
inline int operator < (const P& a, const P& b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && (sgn(a.y - b.y) < 0 ||
        (sgn(a.y - b.y) == 0 && sgn(a.z - b.z) < 0)));
}
bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y) && !sgn(a.z - b.z); }
P::P(const L& l) { *this = l.t - l.s; }
ostream &operator << (ostream &os, const P &p) {
    return (os << "(" << p.x << ", " << p.y << ", " << p.z << ")");
}
istream &operator >> (istream &is, P &p) {
    return (is >> p.x >> p.y >> p.z);
}
LD dist2(const P& p) { return p.x * p.x + p.y * p.y + p.z * p.z; }
LD dist(const P& p) { return sqrt(dist2(p)); }
LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y + a.z * b.z; }
P cross(const P& v, const P& w) {
    return P(v.y * w.z - v.z * w.y, v.z * w.x - v.x * w.z, v.x * w.y - v.y * w.x);
}
LD mix(const V& a, const V& b, const V& c) { return dot(a, cross(b, c)); }
// counter-clockwise r radius
// axis = 0 around axis x
// axis = 1 around axis y
// axis = 2 around axis z
P rotation(const P& p, const LD& r, int axis = 0) {
    if (axis == 0)
        return P(p.x, p.y * cos(r) - p.z * sin(r), p.y * sin(r) + p.z * cos(r));
}

```

```

    else if (axis == 1)
        return P(p.z * cos(r) - p.x * sin(r), p.y, p.z * sin(r)
            + p.x * cos(r));
    else if (axis == 2)
        return P(p.x * cos(r) - p.y * sin(r), p.x * sin(r) + p.y
            * cos(r), p.z);
}
// n is normal vector
// this is clockwise
P rotation(const P& p, const LD& r, const P& n) {
    LD c = cos(r), s = sin(r), x = n.x, y = n.y, z = n.z;
    return P((x * x * (1 - c) + c) * p.x + (x * y * (1 - c) + z
        * s) * p.y + (x * z * (1 - c) - y * s) * p.z,
        (x * y * (1 - c) - z * s) * p.x + (y * y * (1 - c)
            + c) * p.y + (y * z * (1 - c) + x * s) * p.z,
        (x * z * (1 - c) + y * s) * p.x + (y * z * (1 - c)
            - x * s) * p.y + (z * z * (1 - c) + c) * p.z);
}

```

5.11 3D Line, Face

```

// <= 0 improper, < 0 proper
bool p_on_seg(const P& p, const L& seg) {
    P a = seg.s, b = seg.t;
    return !sgn(dist2(cross(p - a, b - a))) && sgn(dot(p - a, p
        - b)) <= 0;
}
LD dist_to_line(const P& p, const L& l) {
    return dist(cross(l.s - p, l.t - p)) / dist(l);
}
LD dist_to_seg(const P& p, const L& l) {
    if (l.s == l.t) return dist(p - l.s);
    V vs = p - l.s, vt = p - l.t;
    if (sgn(dot(l, vs)) < 0) return dist(vs);
    else if (sgn(dot(l, vt)) > 0) return dist(vt);
    else return dist_to_line(p, l);
}

P norm(const F& f) { return cross(f.a - f.b, f.b - f.c); }
int p_on_plane(const F& f, const P& p) { return sgn(dot(norm(f),
    p - f.a)) == 0; }
// if two points are on the opposite side of a line
// return 0 if points is on the line
// makes no sense if points and line are not coplanar
int opposite_side(const P& u, const P& v, const L& l) {
    return sgn(dot(cross(P(l), u - l.s), cross(P(l), v - l.s))) < 0;
}

bool parallel(const L& a, const L& b) { return !sgn(dist2(cross(
    P(a), P(b)))); }
int s_intersect(const L& u, const L& v) {
    return p_on_plane(F(u.s, u.t, v.s), v.t) &&
        opposite_side(u.s, u.t, v) &&
        opposite_side(v.s, v.t, u);
}

```

5.12 3D Convex

```

struct FT {
    int a, b, c;
    FT() {}
    FT(int a, int b, int c) : a(a), b(b), c(c) {}
};

bool p_on_line(const P& p, const L& l) {
    return !sgn(dist2(cross(p - l.s, P(l))));
}

vector<F> convex_hull(vector<P> &p) {
    sort(p.begin(), p.end());
    p.erase(unique(p.begin(), p.end(), p.end()), p.end());
    random_shuffle(p.begin(), p.end());
    vector<FT> face;
    FOR (i, 2, p.size()) {

```

```

        if (p_on_line(p[i], L(p[0], p[1]))) continue;
        swap(p[i], p[2]);
        FOR (j, i + 1, p.size())
            if (sgn(mix(p[1] - p[0], p[2] - p[1], p[j] - p[0])))
                {
                    swap(p[j], p[3]);
                    face.emplace_back(0, 1, 2);
                    face.emplace_back(0, 2, 1);
                    goto found;
                }
        found:
        vector<vector<int>> mk(p.size(), vector<int>(p.size()));
        FOR (v, 3, p.size()) {
            vector<FT> tmp;
            FOR (i, 0, face.size()) {
                int a = face[i].a, b = face[i].b, c = face[i].c;
                if (sgn(mix(p[a] - p[v], p[b] - p[v], p[c] - p[v])))
                    < 0) {
                    mk[a][b] = mk[b][a] = v;
                    mk[b][c] = mk[c][b] = v;
                    mk[c][a] = mk[a][c] = v;
                } else tmp.push_back(face[i]);
            }
            face = tmp;
            FOR (i, 0, tmp.size()) {
                int a = face[i].a, b = face[i].b, c = face[i].c;
                if (mk[a][b] == v) face.emplace_back(b, a, v);
                if (mk[b][c] == v) face.emplace_back(c, b, v);
                if (mk[c][a] == v) face.emplace_back(a, c, v);
            }
        }
        vector<F> out;
        FOR (i, 0, face.size())
            out.emplace_back(p[face[i].a], p[face[i].b], p[face[i].c]);
        return out;
    }
}

```

6 String

6.1 Aho-Corasick Automaton

```

const int N = 1e6 + 100, M = 26;
int mp(char ch) { return ch - 'a'; }
struct ACA {
    int ch[N][M], danger[N], fail[N];
    int sz;
    void init() {
        sz = 1;
        memset(ch[0], 0, sizeof ch[0]);
        memset(danger, 0, sizeof danger);
    }
    void insert(const string &s, int m) {
        int n = s.size(); int u = 0, c;
        FOR (i, 0, n) {
            c = mp[s[i]];
            if (!ch[u][c]) {
                memset(ch[sz], 0, sizeof ch[sz]);
                danger[sz] = 0; ch[u][c] = sz++;
            }
            u = ch[u][c];
        }
        danger[u] |= 1 << m;
    }
    void build() {
        queue<int> Q;
        fail[0] = 0;
        for (int c = 0, u; c < M; c++) {
            u = ch[0][c];
            if (u) { Q.push(u); fail[u] = 0; }
        }
        while (!Q.empty()) {
            int r = Q.front(); Q.pop();
            danger[r] |= danger[fail[r]];

```

```

        for (int c = 0, u; c < M; c++) {
            u = ch[r][c];
            if (!u) {
                ch[r][c] = ch[fail[r]][c];
                continue;
            }
            fail[u] = ch[fail[r]][c];
            Q.push(u);
        }
    }
} ac;

char s[N];
int main() {
    int n; scanf("%d", &n);
    ac.init();
    while (n--) {
        scanf("%s", s);
        ac.insert(s, 0);
    }
    ac.build();
    scanf("%s", s);
    int u = 0; n = strlen(s);
    FOR (i, 0, n) {
        u = ac.ch[u][mp[s[i]]];
        if (ac.danger[u]) {
            puts("YES");
            return 0;
        }
    }
    puts("NO");
    return 0;
}

```

6.2 Hash

```

const int p1 = 1e9 + 7, p2 = 1e9 + 9;
ULL xp1[N], xp2[N], xp[N];
void init_xp() {
    xp1[0] = xp2[0] = xp[0] = 1;
    for (int i = 1; i < N; ++i) {
        xp1[i] = xp1[i - 1] * x % p1;
        xp2[i] = xp2[i - 1] * x % p2;
        xp[i] = xp[i - 1] * x;
    }
}

struct String {
    char s[N];
    int length, subsize;
    bool sorted;
    ULL h[N], hl[N];
    ULL hash() {
        length = strlen(s);
        ULL res1 = 0, res2 = 0;
        h[length] = 0; // ATTENTION!
        for (int j = length - 1; j >= 0; --j) {
            #ifdef ENABLE_DOUBLE_HASH
                res1 = (res1 * x + s[j]) % p1;
                res2 = (res2 * x + s[j]) % p2;
                h[j] = (res1 << 32) | res2;
            #else
                res1 = res1 * x + s[j];
                h[j] = res1;
            #endif
            // printf("%llu\n", h[j]);
        }
        return h[0];
    }
} // hash of [left, right)
ULL get_substring_hash(int left, int right) const {
    int len = right - left;
    #ifdef ENABLE_DOUBLE_HASH
        // get hash of s[left...right-1]
        unsigned int mask32 = ~(0u);
        ULL left1 = h[left] >> 32, right1 = h[right] >> 32;

```

```

ULL left2 = h[left] & mask32, right2 = h[right] & mask32
;
return (((left1 - right1 * xp1[len] % p1 + p1) % p1) <<
32) |
(((left2 - right2 * xp2[len] % p2 + p2) % p2));
#else
return h[left] - h[right] * xp[len];
#endif
}
void get_all_substring_hash(int sublen) {
    subsize = length - sublen + 1;
    for (int i = 0; i < subsize; ++i)
        hl[i] = get_substring_hash(i, i + sublen);
    sorted = 0;
}
void sort_substring_hash() {
    sort(hl, hl + subsize);
    sorted = 1;
}
bool match(ULL key) const {
    if (!sorted) assert (0);
    if (!subsize) return false;
    return binary_search(hl, hl + subsize, key);
}
void init(const char *t) {
    length = strlen(t);
    strcpy(s, t);
}
}
int LCP(const String &a, const String &b, int ai, int bi) {
    // Find LCP of a[ai...] and b[bi...]
    int l = 0, r = min(a.length - ai, b.length - bi);
    while (l < r) {
        int mid = (l + r + 1) / 2;
        if (a.get_substring_hash(ai, ai + mid) == b.
            get_substring_hash(bi, bi + mid))
            l = mid;
        else r = mid - 1;
    }
    return l;
}
}
}

```

6.3 KMP

```

void get_pi(int a[], char s[], int n) {
    int j = a[0] = 0;
    FOR (i, 1, n) {
        while (j && s[i] != s[j]) j = a[j - 1];
        a[i] = j += s[i] == s[j];
    }
}
void get_z(int a[], char s[], int n) {
    int l = 0, r = 0; a[0] = n;
    FOR (i, 1, n) {
        a[i] = i > r ? 0 : min(r - i + 1, a[i - l]);
        while (i + a[i] < n && s[a[i]] == s[i + a[i]]) ++a[i];
        if (i + a[i] - 1 > r) { l = i; r = i + a[i] - 1; }
    }
}
}

```

6.4 Manacher

```

int RL[N];
void manacher(int* a, int n) { // "abc" => "#a#b#a#"
    int r = 0, p = 0;
    FOR (i, 0, n) {
        if (i < r) RL[i] = min(RL[2 * p - i], r - i);
        else RL[i] = 1;
        while (i - RL[i] >= 0 && i + RL[i] < n && a[i - RL[i]]
            == a[i + RL[i]])
            RL[i]++;
        if (RL[i] + i - 1 > r) { r = RL[i] + i - 1; p = i; }
    }
    FOR (i, 0, n) --RL[i];
}
}

```

6.5 Palindrome Automation

```

// num: the number of palindrome suffixes of the prefix
// represented by the node
// cnt: the number of occurrences in string (should update to
// father before using)
namespace pam {
    int t[N][26], fa[N], len[N], rs[N], cnt[N], num[N];
    int sz, n, last;
    int _new(int l) {
        memset(t[sz], 0, sizeof t[0]);
        len[sz] = l; cnt[sz] = num[sz] = 0;
        return sz++;
    }
    void init() {
        rs[n = sz = 0] = -1;
        last = _new(0);
        fa[last] = _new(-1);
    }
    int get_fa(int x) {
        while (rs[n - 1 - len[x]] != rs[n]) x = fa[x];
        return x;
    }
    void ins(int ch) {
        rs[++n] = ch;
        int p = get_fa(last);
        if (!t[p][ch]) {
            int np = _new(len[p] + 2);
            num[np] = num[fa[np]] = t[get_fa(fa[p])][ch] + 1;
            t[p][ch] = np;
        }
        ++cnt[last = t[p][ch]];
    }
}
}

```

6.6 Suffix Array

```

struct SuffixArray {
    const int L;
    vector<vector<int>> > P;
    vector<pair<pair<int, int>, int> > M;
    int s[N], sa[N], rank[N], height[N];
    // s: raw string
    // sa[i]=k: s[k...L-1] ranks i (0 based)
    // rank[i]=k: the rank of s[i...L-1] is k (0 based)
    // height[i] = lcp(sa[i-1], sa[i])
    SuffixArray(const string &raw_s) : L(raw_s.length()), P(1,
        vector<int>(L, 0)), M(L) {
        for (int i = 0; i < L; i++)
            P[0][i] = this->s[i] = int(raw_s[i]);
        for (int skip = 1, level = 1; skip < L; skip *= 2, level
            ++){
            P.push_back(vector<int>(L, 0));
            for (int i = 0; i < L; i++)
                M[i] = make_pair(make_pair(P[level - 1][i], i +
                    skip < L ? P[level - 1][i + skip] : -1000),
                    i);
            sort(M.begin(), M.end());
            for (int i = 0; i < L; i++)
                P[level][M[i].second] = (i > 0 && M[i].first ==
                    M[i - 1].first) ? P[level][M[i - 1].second]
                    : i;
        }
        for (unsigned i = 0; i < P.back().size(); ++i) {
            rank[i] = P.back()[i];
            sa[rank[i]] = i;
        }
    }
    // This is a traditional way to calculate LCP
    void getHeight() {
        memset(height, 0, sizeof height);
        int k = 0;
        for (int i = 0; i < L; ++i) {
            if (rank[i] == 0) continue;

```

```

            if (k) k--;
            int j = sa[rank[i] - 1];
            while (i + k < L && j + k < L && s[i + k] == s[j + k]
                ) ++k;
            height[rank[i]] = k;
        }
        rmq_init(height, L);
    }
    int f[N][Nlog];
    inline int highbit(int x) {
        return 31 - __builtin_clz(x);
    }
    int rmq_query(int x, int y) {
        int p = highbit(y - x + 1);
        return min(f[x][p], f[y - (1 << p) + 1][p]);
    }
    // arr has to be 0 based
    void rmq_init(int *arr, int length) {
        for (int x = 0; x <= highbit(length); ++x)
            for (int i = 0; i <= length - (1 << x); ++i) {
                if (!x) f[i][x] = arr[i];
                else f[i][x] = min(f[i][x - 1], f[i + (1 << (x -
                    1))][x - 1]);
            }
    }
    #ifdef NEW
    // returns the length of the longest common prefix of s[i...
    // L-1] and s[j...L-1]
    int LongestCommonPrefix(int i, int j) {
        int len = 0;
        if (i == j) return L - i;
        for (int k = (int) P.size() - 1; k >= 0 && i < L && j <
            L; k--) {
            if (P[k][i] == P[k][j]) {
                i += 1 << k;
                j += 1 << k;
                len += 1 << k;
            }
        }
        return len;
    }
    #else
    int LongestCommonPrefix(int i, int j) {
        // getHeight() must be called first
        if (i == j) return L - i;
        if (i > j) swap(i, j);
        return rmq_query(i + 1, j);
    }
    #endif
    int checkNonOverlappingSubstring(int K) {
        // check if there is two non-overlapping identical
        // substring of length K
        int minsa = 0, maxsa = 0;
        for (int i = 0; i < L; ++i) {
            if (height[i] < K) {
                minsa = sa[i]; maxsa = sa[i];
            } else {
                minsa = min(minsa, sa[i]);
                maxsa = max(maxsa, sa[i]);
                if (maxsa - minsa >= K) return 1;
            }
        }
        return 0;
    }
    int checkBelongToDifferentSubstring(int K, int split) {
        int minsa = 0, maxsa = 0;
        for (int i = 0; i < L; ++i) {
            if (height[i] < K) {
                minsa = sa[i]; maxsa = sa[i];
            } else {
                minsa = min(minsa, sa[i]);
                maxsa = max(maxsa, sa[i]);
                if (maxsa > split && minsa < split) return 1;
            }
        }
        return 0;
    }
}

```

```

} *S;
int main() {
    int sp = s.length();
    s += "x" + t;
    S = new SuffixArray(s);
    S->getHeight();
    int left = 0, right = sp;
    while (left < right) {
        // ...
        if (S->checkBelongToDifferentSubstring(mid, sp))
            // ...
    }
    printf("%d\n", left);
}

//////////
// rk [0..n-1] -> [1..n], sa/ht [1..n]
// s[i] > 0 && s[n] = 0
// b: normally as bucket
// c: normally as bucket1
// d: normally as bucket2
// f: normally as cntbuf
//////////
template<size_t size>
struct SuffixArray {
    bool t[size << 1];
    int b[size], c[size];
    int sa[size], rk[size], ht[size];
    inline bool isLMS(const int i, const bool *t) { return i > 0
        && t[i] && !t[i - 1]; }
    template<class T>
    inline void inducedSort(T s, int *sa, const int n, const int
        M, const int bs, bool *t, int *b, int *f, int *p) {
        fill(b, b + M, 0); fill(sa, sa + n, -1);
        FOR (i, 0, n) b[s[i]]++;
        f[0] = b[0];
        FOR (i, 1, M) f[i] = f[i - 1] + b[i];
        FORD (i, bs - 1, -1) sa[--f[s[p[i]]]] = p[i];
        FOR (i, 1, M) f[i] = f[i - 1] + b[i - 1];
        FOR (i, 0, n) if (sa[i] > 0 && !t[sa[i] - 1]) sa[f[s[sa[
            i] - 1]]++] = sa[i] - 1;
        f[0] = b[0];
        FOR (i, 1, M) f[i] = f[i - 1] + b[i];
        FORD (i, n - 1, -1) if (sa[i] > 0 && t[sa[i] - 1]) sa[--
            f[s[sa[i] - 1]]] = sa[i] - 1;
    }
    template<class T>
    inline void sais(T s, int *sa, int n, bool *t, int *b, int *
        c, int M) {
        int i, j, bs = 0, cnt = 0, p = -1, x, *r = b + M;
        t[n - 1] = 1;
        FORD (i, n - 2, -1) t[i] = s[i] < s[i + 1] || (s[i] == s
            [i + 1] && t[i + 1]);
        FOR (i, 1, n) if (t[i] && !t[i - 1]) c[bs++] = i;
        inducedSort(s, sa, n, M, bs, t, b, r, c);
        for (i = bs = 0; i < n; i++) if (isLMS(sa[i], t)) sa[bs
            ++] = sa[i];
        FOR (i, bs, n) sa[i] = -1;
        FOR (i, 0, bs) {
            x = sa[i];
            for (j = 0; j < n; j++) {
                if (p == -1 || s[x + j] != s[p + j] || t[x + j]
                    != t[p + j]) { cnt++, p = x; break; }
                else if (j > 0 && (isLMS(x + j, t) || isLMS(p +
                    j, t))) break;
            }
            x = (~x & 1 ? x >> 1 : x - 1 >> 1), sa[bs + x] = cnt
                - 1;
        }
        for (i = j = n - 1; i >= bs; i--) if (sa[i] >= 0) sa[j
            --] = sa[i];
        int *s1 = sa + n - bs, *d = c + bs;
        if (cnt < bs) sais(s1, sa, bs, t + n, b, c + bs, cnt);
        else FOR (i, 0, bs) sa[s1[i]] = i;
        FOR (i, 0, bs) d[i] = c[sa[i]];
        inducedSort(s, sa, n, M, bs, t, b, r, d);
    }
}

```

```

}
template<typename T>
inline void getHeight(T s, const int n, const int *sa) {
    for (int i = 0, k = 0; i < n; i++) {
        if (rk[i] == 0) k = 0;
        else {
            if (k > 0) k--;
            int j = sa[rk[i] - 1];
            while (i + k < n && j + k < n && s[i + k] == s[j
                + k]) k++;
        }
        ht[rk[i]] = k;
    }
}
template<class T>
inline void init(T s, int n, int M) {
    sais(s, sa, ++n, t, b, c, M);
    for (int i = 1; i < n; i++) rk[sa[i]] = i;
    getHeight(s, n, sa);
}
};
SuffixArray<N> sa;
int main() {
    int n = s.length();
    sa.init(s, n, 128);
    FOR (i, 1, n + 1) printf("%d%c", sa.sa[i] + 1, i == _i - 1 ?
        '\n' : ' ');
    FOR (i, 2, n + 1) printf("%d%c", sa.ht[i], i == _i - 1 ? '\n'
        : ' ');
}

```

6.7 Suffix Automation

```

namespace sam {
    const int M = N << 1;
    int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;
    void init() { memset(t, 0, (sz + 10) * sizeof t[0]); sz = 2;
        last = 1; }
    void ins(int ch) {
        int p = last, np = last = sz++;
        len[np] = len[p] + 1;
        for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
        if (!p) { fa[np] = 1; return; }
        int q = t[p][ch];
        if (len[p] + 1 == len[q]) fa[np] = q;
        else {
            int nq = sz++; len[nq] = len[p] + 1;
            memcpy(t[nq], t[q], sizeof t[0]);
            fa[nq] = fa[q];
            fa[np] = fa[q] = nq;
            for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
        }
    }
    int c[M] = {1}, a[M];
    void rsort() {
        FOR (i, 1, sz) c[i] = 0;
        FOR (i, 1, sz) c[len[i]]++;
        FOR (i, 1, sz) c[i] += c[i - 1];
        FOR (i, 1, sz) a[--c[len[i]]] = i;
    }
    // really-generalized sam
    int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;
    LL cnt[M][2];
    void ins(int ch, int id) {
        int p = last, np = 0, nq = 0, q = -1;
        if (!t[p][ch]) {
            np = sz++;
            len[np] = len[p] + 1;
            for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
        }
        if (!p) fa[np] = 1;
        else {
            q = t[p][ch];
            if (len[p] + 1 == len[q]) fa[np] = q;
            else {
                int nq = sz++; len[nq] = len[p] + 1;
                memcpy(t[nq], t[q], sizeof t[0]);
                fa[nq] = fa[q];
                fa[np] = fa[q] = nq;
                for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
            }
        }
    }
}

```

```

        nq = sz++; len[nq] = len[p] + 1;
        memcpy(t[nq], t[q], sizeof t[0]);
        fa[nq] = fa[q];
        fa[np] = fa[q] = nq;
        for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
    }
    last = np ? np : nq ? nq : q;
    cnt[last][id] = 1;
}
// lexicographical order
// rsort2 is not topo sort
void ins(int ch, int pp) {
    int p = last, np = last = sz++;
    len[np] = len[p] + 1; one[np] = pos[np] = pp;
    for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
    if (!p) { fa[np] = 1; return; }
    int q = t[p][ch];
    if (len[q] == len[p] + 1) fa[np] = q;
    else {
        int nq = sz++; len[nq] = len[p] + 1; one[nq] = one[q];
        memcpy(t[nq], t[q], sizeof t[0]);
        fa[nq] = fa[q];
        fa[q] = fa[np] = nq;
        for (; p && t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
    }
}
// lexicographical order
// generalized sam
int up[M], c[256] = {2}, a[M];
void rsort2() {
    FOR (i, 1, 256) c[i] = 0;
    FOR (i, 2, sz) up[i] = s[one[i] + len[fa[i]]];
    FOR (i, 2, sz) c[up[i]]++;
    FOR (i, 1, 256) c[i] += c[i - 1];
    FOR (i, 2, sz) a[--c[up[i]]] = i;
    FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]);
}
int t[M][26], len[M] = {0}, fa[M], sz = 2, last = 1;
char* one[M];
void ins(int ch, char* pp) {
    int p = last, np = 0, nq = 0, q = -1;
    if (!t[p][ch]) {
        np = sz++; one[np] = pp;
        len[np] = len[p] + 1;
        for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
    }
    if (!p) fa[np] = 1;
    else {
        q = t[p][ch];
        if (len[p] + 1 == len[q]) fa[np] = q;
        else {
            nq = sz++; len[nq] = len[p] + 1; one[nq] = one[q];
            memcpy(t[nq], t[q], sizeof t[0]);
            fa[nq] = fa[q];
            fa[np] = fa[q] = nq;
            for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
        }
    }
    last = np ? np : nq ? nq : q;
}
int up[M], c[256] = {2}, aa[M];
vector<int> G[M];
void rsort() {
    FOR (i, 1, 256) c[i] = 0;
    FOR (i, 2, sz) up[i] = *(one[i] + len[fa[i]]);
    FOR (i, 2, sz) c[up[i]]++;
    FOR (i, 1, 256) c[i] += c[i - 1];
    FOR (i, 2, sz) aa[--c[up[i]]] = i;
    FOR (i, 2, sz) G[fa[aa[i]]].push_back(aa[i]);
}
// match
int u = 1, l = 0;
FOR (i, 0, strlen(s)) {
    int ch = s[i] - 'a';
    while (u && !t[u][ch]) { u = fa[u]; l = len[u]; }
}

```

```

++l; u = t[u][ch];
if (!u) u = 1;
if (l) // do something...
}
// substring state
int get_state(int l, int r) {
    int u = rpos[r], s = r - l + 1;
    FORD (i, SP - 1, -1) if (len[pa[u][i]] >= s) u = pa[u][i];
    return u;
}

// LCT-SAM
namespace lct_sam {
    extern struct P *const null;
    const int M = N;
    struct P {
        P *fa, *ls, *rs;
        int last;

        bool has_fa() { return fa->ls == this || fa->rs == this; }
        bool d() { return fa->ls == this; }
        P& c(bool x) { return x ? ls : rs; }
        P* up() { return this; }
        void down() {
            if (ls != null) ls->last = last;
            if (rs != null) rs->last = last;
        }
        void all_down() { if (has_fa()) fa->all_down(); down(); }
    } *const null = new P[0, 0, 0, 0], pool[M], *pit = pool;
    P* G[N];
    int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;

    void rot(P* o) {
        bool dd = o->d();
        P *f = o->fa, *t = o->c(!dd);
        if (f->has_fa()) f->fa->c(f->d()) = o; o->fa = f->fa;
        if (t != null) t->fa = f; f->c(dd) = t;
        o->c(!dd) = f->up(); f->fa = o;
    }

    void splay(P* o) {
        o->all_down();
        while (o->has_fa()) {
            if (o->fa->has_fa())
                rot(o->d() ^ o->fa->d() ? o : o->fa);
            rot(o);
        }
        o->up();
    }

    void access(int last, P* u, P* v = null) {
        if (u == null) { v->last = last; return; }
        splay(u);
        P *t = u;
        while (t->ls != null) t = t->ls;
        int L = len[fa[t - pool]] + 1, R = len[u - pool];

        if (u->last) bit::add(u->last - R + 2, u->last - L + 2, 1);
        else bit::add(1, 1, R - L + 1);
        bit::add(last - R + 2, last - L + 2, -1);

        u->rs = v;
        access(last, u->up()->fa, u);
    }

    void insert(P* u, P* v, P* t) {
        if (v != null) { splay(v); v->rs = null; }
        splay(u);
        u->fa = t; t->fa = v;
    }

    void ins(int ch, int pp) {
        int p = last, np = last = sz++;
        len[np] = len[p] + 1;
        for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
        if (!p) fa[np] = 1;
        else {
            int q = t[p][ch];

```

```

        if (len[p] + 1 == len[q]) { fa[np] = q; G[np]->fa = G[q]; }
        else {
            int nq = sz++; len[nq] = len[p] + 1;
            memcpy(t[nq], t[q], sizeof t[0]);
            insert(G[q], G[fa[q]], G[nq]);
            G[nq]->last = G[q]->last;
            fa[nq] = fa[q];
            fa[np] = fa[q] = nq;
            G[np]->fa = G[nq];
            for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
        }
    }
    access(pp + 1, G[np]);
}

void init() {
    ++pit;
    FOR (i, 1, N) {
        G[i] = pit++;
        G[i]->ls = G[i]->rs = G[i]->fa = null;
    }
    G[1] = null;
}
}

```

7 Miscellaneous

7.1 Date

```

// Routines for performing computations on dates. In these
// routines, months are expressed as integers from 1 to 12, days
// are expressed as integers from 1 to 31, and
// years are expressed as 4-digit integers.
string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};
// converts Gregorian date to integer (Julian day number)
int DateToInt (int m, int d, int y){
    return
        1461 * (y + 4800 + (m - 14) / 12) / 4 +
        367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
        3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
        d - 32075;
}
// converts integer (Julian day number) to Gregorian date: month
//day/year
void IntToDate (int jd, int &m, int &d, int &y){
    int x, n, i, j;
    x = jd + 68569;
    n = 4 * x / 146097;
    x -= (146097 * n + 3) / 4;
    i = (4000 * (x + 1)) / 1461001;
    x -= 1461 * i / 4 - 31;
    j = 80 * x / 2447;
    d = x - 2447 * j / 80;
    x = j / 11;
    m = j + 2 - 12 * x;
    y = 100 * (n - 49) + i + x;
}
// converts integer (Julian day number) to day of week
string IntToDay (int jd){
    return dayOfWeek[jd % 7];
}

```

7.2 Subset Enumeration

```

// all proper subset
for (int s = (S - 1) & S; s; s = (s - 1) & S) {
    // ...
}

// subset of length k
template<typename T>
void subset(int k, int n, T&& f) {
    int t = (1 << k) - 1;
    while (t < 1 << n) {

```

```

        f(t);
        int x = t & -t, y = t + x;
        t = ((t & -y) / x >> 1) | y;
    }
}

7.3 Digit DP
LL dfs(LL base, LL pos, LL len, LL s, bool limit) {
    if (pos == -1) return s ? base : 1;
    if (!limit && dp[base][pos][len][s] != -1) return dp[base][pos][len][s];
    LL ret = 0;
    LL ed = limit ? a[pos] : base - 1;
    FOR (i, 0, ed + 1) {
        tmp[pos] = i;
        if (len == pos)
            ret += dfs(base, pos - 1, len - (i == 0), s, limit && i == a[pos]);
        else if (s && pos < (len + 1) / 2)
            ret += dfs(base, pos - 1, len, tmp[len - pos] == i, limit && i == a[pos]);
        else
            ret += dfs(base, pos - 1, len, s, limit && i == a[pos]);
    }
    if (!limit) dp[base][pos][len][s] = ret;
    return ret;
}

LL solve(LL x, LL base) {
    LL sz = 0;
    while (x) {
        a[sz++] = x % base;
        x /= base;
    }
    return dfs(base, sz - 1, sz - 1, 1, true);
}

7.4 Simulated Annealing
// Minimum Circle Cover
using LD = double;
const int N = 1E4 + 100;
int x[N], y[N], n;
LD eval(LD xx, LD yy) {
    LD r = 0;
    FOR (i, 0, n)
        r = max(r, sqrt(pow(xx - x[i], 2) + pow(yy - y[i], 2)));
    return r;
}

mt19937 mt(time(0));
auto rd = bind(uniform_real_distribution<LD>(-1, 1), mt);
int main() {
    int X, Y;
    while (cin >> X >> Y >> n) {
        FOR (i, 0, n) scanf("%d%d", &x[i], &y[i]);
        pair<LD, LD> ans;
        LD M = 1e9;
        FOR (_, 0, 100) {
            LD cur_x = X / 2.0, cur_y = Y / 2.0, T = max(X, Y);
            while (T > 1e-3) {
                LD best_ans = eval(cur_x, cur_y);
                LD best_x = cur_x, best_y = cur_y;
                FOR (____, 0, 20) {
                    LD nxt_x = cur_x + rd() * T, nxt_y = cur_y + rd() * T;
                    LD nxt_ans = eval(nxt_x, nxt_y);
                    if (nxt_ans < best_ans) {
                        best_x = nxt_x; best_y = nxt_y;
                        best_ans = nxt_ans;
                    }
                }
                cur_x = best_x; cur_y = best_y;
                T *= .9;
            }

```

```
if (eval(cur_x, cur_y) < M) {  
    ans = {cur_x, cur_y}; M = eval(cur_x, cur_y);  
}  
  
}  
printf("%.1f,%.1f).\n%.1f\n", ans.first, ans.second,  
    eval(ans.first, ans.second));  
}  
}
```
