BitofChairmanTree.cpp

```
#define mid (l + r >> 1)
using namespace std;
const int N = 101000;
int n, m, a[N], b[N], b_c, sav[N];
struct S {
    struct Q {
        Q *l, *r;
        int s, c;
    }key[N << 5];
    Q* root[N];Q* p;
    Q *a[N], *b[N];
    int t1, t2;
    int sav[N];
    void init () {
        p = key;
        memset (sav, 0, sizeof sav);
    }
    Q* getnew (int _c) {
        return p->s = 1, p->c = _c, p ++;
    }
    Q* getnew (Q* a, Q* b) {
        return p->l = a, p->r = b, p->s = a->s + b->s, p->c = a->c
+ b->c, p ++;
    }
    Q* build (int l, int r) {
        if (l == r) return getnew (0);
        return getnew (build (l, mid), build (mid + 1, r));
    }
    Q* inc (Q* t, int i) {
        if (t->s == 1) return getnew (t->c + 1);
        if (i <= t->l->s) return getnew (inc (t->l, i), t->r);
        else return getnew (t->l, inc (t->r, i - t->l->s));
    }
    Q* dec (Q* t, int i) {
        if (t->s == 1) return getnew (t->c - 1);
        if (i <= t->l->s) return getnew (dec (t->l, i), t->r);
        else return getnew (t->l, dec (t->r, i - t->l->s));
    }
    int query (int k) {
        if (b[1]->s == 1) return 1;
        int t (0);
        for (int i = 1; i <= t1; i ++)
            t -= a[i]->l->c;
        for (int i = 1; i <= t2; i ++)
            t += b[i]->l->c;
        if (k <= t)
        {
            for (int i = 1; i <= t1; i ++)
                a[i] = a[i]->l;
            for (int i = 1; i <= t2; i ++)
                b[i] = b[i]->l;
            return query (k);
        }
        else
        {
            int tmp = b[1]->l->s;
            for (int i = 1; i <= t1; i ++)
                a[i] = a[i]->r;
            for (int i = 1; i <= t2; i ++)
                b[i] = b[i]->r;
            return tmp + query (k - t);
        }
    }
    void INC (int x, int v)
```

```
    {
        for (int i = x; i <= n; i += i & -i)
            root[i] = inc (root[i], v);
    }
    void DEC (int x, int v)
    {
        for (int i = x; i <= n; i += i & -i)
            root[i] = dec (root[i], v);
    }
}seg;
int l[N], r[N], k[N], x[N], v[N];
int main ()
{
    seg.init ();
    scanf ("%d%d", &n, &m);
    for (int i = 1; i <= n; i ++)
        scanf ("%d", &a[i]), b[i] = a[i];
    b_c = n;
    char s[10];
    for (int i = 1; i <= m; i ++)
    {
        scanf ("%s", s);
        if (s[0] == 'Q')
            scanf ("%d%d%d", &l[i], &r[i], &k[i]);
        if (s[0] == 'C')
            scanf ("%d%d", &x[i], &v[i]), b[++ b_c] = v[i];
    }
    sort (b + 1, b + 1 + b_c);
    b_c = unique (b + 1, b + 1 + b_c) - (b + 1);
    for (int i = 1; i <= n; i ++)
        a[i] = lower_bound (b + 1, b + 1 + b_c, a[i]) - b;
    for (int i = 1; i <= m; i ++)
        if (x[i] != 0)
            v[i] = lower_bound (b + 1, b + 1 + b_c, v[i]) - b;

    seg.root[0] = seg.build (1, b_c);
    for (int i = 1; i <= n; i ++)
        seg.root[i] = seg.root[0];
    for (int i = 1; i <= n; i ++)
        seg.INC (i, a[i]);
    for (int j = 1; j <= m; j ++)
    {
        seg.t1 = seg.t2 = 0;
        if (l[j] != 0)
        {
            for (int i = l[j] - 1; i; i -= i & -i)
                seg.a[++ seg.t1] = seg.root[i];
            for (int i = r[j]; i; i -= i & -i)
                seg.b[++ seg.t2] = seg.root[i];
            printf ("%d\n", b[seg.query (k[j])]);
        }
        if (x[j] != 0)
        {
            seg.DEC (x[j], a[x[j]]);
            a[x[j]] = v[j];
            seg.INC (x[j], a[x[j]]);
        }
    }
    return 0;
}
```

ChairmanTree.cpp

```
#define mid (((l) + (r)) >> 1)
const int N = 301000;
struct S {
    struct Q {
        Q *l, *r;
        int s, c;
    }key[N << 4];
    Q *root[N];
    Q *p;
    inline void init (int n) {
        p = key;
        root[0] = build(1, n);
    }
    inline Q* getnew (int _c) {
        return p->s = 1, p->c = _c, p ++;
    }
    inline Q* getnew (Q* a, Q* b) {
        return p->l = a, p->r = b, p->s = a->s + b->s, p->c = a->c
+ b->c, p ++;
    }
    inline Q* build (int l, int r) {
        if (l == r) return getnew (0);
        return getnew (build (l, (l + r) >> 1), build (((l + r) >>
1) + 1, r));
    }
    inline Q* inc (Q* t, int i) {
        if (t->s == 1) return getnew (t->c + 1);
        if (i <= t->l->s) return getnew (inc (t->l, i), t->r);
        else return getnew (t->l, inc (t->r, i - t->l->s));
    }
    inline int query (Q* a, Q* b, int k) {
        if (a->s == 1) return 1;
        int t = b->l->c - a->l->c;
        if (k <= t) return query (a->l, b->l, k);
        else return a->l->s + query (a->r, b->r, k - t);
    }
}seg;
int n, m, a[N], b[N], c[N], b_c;
int main() {
    freopen("in", "r", stdin);
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i ++)
        scanf("%d", &a[i]), b[++ b_c] = a[i];
    sort(b + 1, b + 1 + b_c);
    b_c = unique(b + 1, b + 1 + b_c) - (b + 1);
    for (int i = 1; i <= n; i ++)
        c[i] = lower_bound(b + 1, b + 1 + b_c, a[i]) - b;
    seg.init(b_c);
    for (int i = 1; i <= n; i ++)
        seg.root[i] = seg.inc(seg.root[i - 1], c[i]);
    for (int i = 1, x, y, k; i <= m; i ++) {
        scanf("%d%d%d", &x, &y, &k);
        printf("%d\n", b[seg.query(seg.root[x - 1], seg.root[y],
k)]);
    }
    return 0;
}
```

HeavyLightDecomposition.cpp

```
const int N = 500000;
int p[N], s[N], d[N], tid[N], top[N], son[N], key[N], next[N], len[N],
head[N], cnt, tid_c, w[N];
int a[N], b[N], c[N];
int n;
inline void add (int x, int y, int w) {
    key[cnt] = y;
    next[cnt] = head[x];
    len[cnt] = w;
    head[x] = cnt ++;
}
void D1 (int x, int fa) {
    p[x] = fa;
    d[x] = d[fa] + 1;
    s[x] = 1;
    int t1 (0), t2 (0);
    for (int i = head[x]; ~ i; i = next[i])
    {
        if (key[i] == fa) continue;
        D1 (key[i], x);
        s[x] += s[key[i]];
        if (s[key[i]] > t1)
            t1 = s[key[i]], t2 = key[i];
    }
    son[x] = t2;
}
void D2 (int x, int TOP) {
    tid[x] = ++ tid_c;
    top[x] = TOP;
    if (son[x]) D2 (son[x], TOP);
    for (int i = head[x]; ~ i; i = next[i])
    {
        if (key[i] == p[x] || key[i] == son[x]) continue;
        D2 (key[i], key[i]);
    }
}
void D3 (int x, int fa) {
    for (int i = head[x]; ~ i; i = next[i])
    {
        if (key[i] == fa) continue;
        D3 (key[i], x);
        w[tid[key[i]]] = len[i];
    }
}
int ask (int x, int y) {
    int z (0);
    while (top[x] != top[y])
    {
        if (d[top[x]] < d[top[y]])
            swap (x, y);
        z = max (z, seg.query (tid[top[x]], tid[x], 1, n, 1));
        x = p[top[x]];
    }
    if (d[x] > d[y])
        swap (x, y);
    return max (z, seg.query (tid[son[x]], tid[y], 1, n, 1));
}
void Cover(int x, int y, int v) {
    while(top[x] != top[y]) {
        if (d[top[x]] < d[top[y]])
            swap(x, y);
        seg.Cover(tid[top[x]], tid[x], v, 1, n, 1);
        x = p[top[x]];
    }
```

```
    if (d[x] > d[y])
        swap(x, y);
    seg.Cover(tid[son[x]], tid[y], v, 1, n, 1);
}
void Add(int x, int y, int v) {
    while(top[x] != top[y]) {
        if (d[top[x]] < d[top[y]])
            swap(x, y);
        seg.Add(tid[top[x]], tid[x], v, 1, n, 1);
        x = p[top[x]];
    }
    if (d[x] > d[y])
        swap(x, y);
    seg.Add(tid[son[x]], tid[y], v, 1, n, 1);
}
int main () {
    freopen("in", "r", stdin);
    tid_c = 0;cnt = 0;
    memset (head, -1, sizeof head);
    scanf ("%d", &n);
    for (int i = 1; i <= n - 1; i ++)
        scanf ("%d%d%d", &a[i], &b[i], &c[i]), add (a[i], b[i], c[i]),
add (b[i], a[i], c[i]);
    w[1] = 0;d[1] = 0;
    D1 (1, 1);D2 (1, 1);D3 (1, 1);
    seg.build(1, n, 1, w);
    char op[15];
    while(scanf("%s", op), op[0] != 'S') {
        int x, y, v;
        if (op[0] == 'M') {
            scanf("%d%d", &x, &y);
            printf("%d\n", ask(x, y));
        }
        if (op[0] == 'C' && op[1] == 'o') {
            scanf("%d%d%d", &x, &y, &v);
            Cover(x, y, v);
        }
        if (op[0] == 'A') {
            scanf("%d%d%d", &x, &y, &v);
            Add(x, y, v);
        }
    }
    return 0;
}
```

SegTreeSearch.cpp

```
#define lc idx << 1
#define rc idx << 1 | 1
#define lson l, mid, lc
#define rson mid + 1, r, rc
using namespace std;
const int N = 401000;
typedef long long LL;
LL sum[N << 2];
int cov[N << 2], mn[N << 2];
void pushup(int idx) {
    sum[idx] = sum[lc] + sum[rc];
    mn[idx] = mn[lc];
}
void pushdown(int l, int r, int mid, int idx) {
    if (cov[idx] != 0) {
        cov[lc] = cov[rc] = cov[idx];
        mn[lc] = cov[idx]; mn[rc] = cov[idx];
        sum[lc] = (LL)cov[idx] * (mid - l + 1);
        sum[rc] = (LL)cov[idx] * (r - mid);
        cov[idx] = 0;
    }
}
void build(int l, int r, int idx) {
    if (l == r) {
        sum[idx] = l; mn[idx] = l;
        return ;
    }
    int mid = (l + r) >> 1;
    build(lson);
    build(rson);
    pushup(idx);
}
void update(int L, int R, int val, int l, int r, int idx) {
    if (L > R) return ;
    if (L <= l && r <= R) {
        cov[idx] = val;
        mn[idx] = val;
        sum[idx] = (LL)val * (r - l + 1);
        return ;
    }
    int mid = (l + r) >> 1;
    pushdown(l, r, mid, idx);
    if (L <= mid) update(L, R, val, lson);
    if (R > mid) update(L, R, val, rson);
    pushup(idx);
}
int left(int L, int R, int val, int l, int r, int idx) {
    if (l == r) {
        if (mn[idx] < val) return l;
        else return 0;
    }
    int mid = (l + r) >> 1;
    pushdown(l, r, mid, idx);
    if (R <= mid) {
        return left(L, R, val, lson);
    }
    else if (L > mid) {
        return left(L, R, val, rson);
    }
    else {
        if (mn[rc] < val) return left(L, R, val, rson);
        else return left(L, R, val, lson);
    }
}
```

Splay-Recommand.cpp

```
#define rep(i,s,t) for(int i=s;i<=t;i++)
#define dwn(i,s,t) for(int i=s;i>=t;i--)
typedef long long LL;
struct Node {
    Node*ch[2], *p;
    int size, val;
    LL sum;
    Node() {
        size = 0;
        val = sum = 0;
    }
    bool d() {
        return this == p->ch[1];
    }
    void setc(Node*c, int d) {
        ch[d] = c;
        c->p = this;
    }
    void relax();
    void upd() {
        size = ch[0]->size + ch[1]->size + 1;
        sum = ch[0]->sum + ch[1]->sum + val;
    }
} Tnull, *null = &Tnull;
Node mem[1001000], *C = mem;
void Node::relax() {
}
Node*make(int v) {
    C->ch[0] = C->ch[1] = null;
    C->size = 1;C->val = v;
    C->sum = v;
    return C++;
}
Node*root;
Node*rot(Node*t) {
    Node*p = t->p;
    bool d = t->d();
    p->p->setc(t, p->d());
    p->setc(t->ch[!d], d);
    t->setc(p, !d);
    p->upd();
    if (p == root)
        root = t;
}
void splay(Node*t, Node*f = null) {
    while (t->p != f) {
        if (t->p->p == f)
            rot(t);
        else
            t->d() == t->p->d() ? (rot(t->p), rot(t)) : (rot(t),
rot(t));
    }
    t->upd();
}
void random_spaly() {
    if (C - mem < 10) return ;
    int t = rand() % (C - mem - 1) + 1;
    splay(mem + t);
}
Node* insert(Node* t, int val) {
    for (; ; ) {
        bool d = val > t->val;
        if (t->ch[d] == null) {
            Node* p = make(val);
```

```
            t->setc(p, d);
            return p;
        }
        t = t->ch[d];
    }
}
int select(LL k) {
    int z = 0;
    for (Node*t = root;;) {
        t->relax();
        if (t == null)
            return z;
        if (k >= t->sum)
            return z + t->size;
        LL c = t->ch[0]->sum + t->val;
        if (k == c)
            return z + t->ch[0]->size + 1;
        if (k > c)
            k -= c, z += t->ch[0]->size + 1, t = t->ch[1];
        else
            t = t->ch[0];
    }
}
void ins(int val) {
    if (root == null)
        root = make(val), root->p = null;
    else
        splay(insert(root, val));
}
int read()
{
    int x=0,f=1;char ch=getchar();
    while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}
    while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}
    return x*f;
}
    srand(time(0));
    root = null;
    random_spaly();
```

Splay-Remove.cpp

```
const int MAX_N = 1001000 + 10;
const int INF = ~0U >> 1;
struct Node {
    Node*ch[2], *p;
    int size, val, gcd;
    Node() {
        size = 0;
        val = gcd = 0;
    }
    bool d() {
        return this == p->ch[1];
    }
    void setc(Node*c, int d) {
        ch[d] = c;
        c->p = this;
    }
    void relax();
    void upd() {
        size = ch[0]->size + ch[1]->size + 1;
        gcd = __gcd(ch[0]->gcd, __gcd(ch[1]->gcd, val));
    }
} Tnull, *null = &Tnull;
Node mem[MAX_N], *C = mem;
void Node::relax() {
}
Node*make(int v) {
    C->ch[0] = C->ch[1] = null;
    C->size = 1;
    C->val = v;
    C->gcd = v;
    return C++;
}
Node*root;
Node*rot(Node*t) {
    Node*p = t->p;
    int d = t->d();
    p->p->setc(t, p->d());
    p->setc(t->ch[!d], d);
    t->setc(p, !d);
    p->upd();
    if (p == root)
        root = t;
}
void splay(Node*t, Node*f = null) {
    while (t->p != f) {
        if (t->p->p == f)
            rot(t);
        else
            t->d() == t->p->d() ? (rot(t->p), rot(t)) : (rot(t),
rot(t));
    }
    t->upd();
}
Node* insert(Node* t, int val) {
    for (; ; ) {
        bool d = val > t->val;
        if (t->ch[d] == null) {
            Node* p = make(val);
            t->setc(p, d);
            return p;
        }
        t = t->ch[d];
    }
}
```

```
Node* select(int k) {
    for (Node*t = root;;) {
        t->relax();
        int c = t->ch[0]->size;
        if (k == c)
            return t;
        if (k > c)
            k -= c + 1, t = t->ch[1];
        else
            t = t->ch[0];
    }
}
Node* find(Node* t, int val) {
    bool d = val > t->val;
    if (val == t->val) return t;
    return find(t->ch[d], val);
}
Node* mx(Node* t) {
    for (; ; t = t->ch[1])
        if (t->ch[1] == null) return t;
}
Node* mn(Node* t) {
    for (; ; t = t->ch[0])
        if (t->ch[0] == null) return t;
}
Node* remove(Node* t) {
    splay(t);
    if (root->ch[0] != null) {
        splay(mx(root->ch[0]), root);
        root->ch[0]->setc(root->ch[1], 1);
        root = root->ch[0];
        root->p = null;
        root->upd();
    } else {
        root = root->ch[1];
        root->p = null;
    }
}
Node*&get(int l, int r) { //[l,r)
    Node*L = select(l - 1);
    Node*R = select(r);
    splay(L);
    splay(R, L);
    return R->ch[0];
}
void travel(Node* t) {
    if (t == null) return ;
    travel(t->ch[0]);
    printf("%d ", t->val);
    travel(t->ch[1]);
}
void ins(int val) {
    if (root == null)
        root = make(val), root->p = null;
    else
        splay(insert(root, val));
}
void rm(int val) {
    remove(find(root, val));
}
```

LCT.cpp

```
#define rep(i, s, t) for (int i = s; i <= t; i ++)
typedef int int64;
const int LEN = 3001000;
namespace LCT {
    struct Node {
        Node*p, *ch[2];
        int64 add, val;
        int size;
        bool isRoot;
        Node*fa;
        Node() {
            add = val = 0;
            isRoot = 0;
            size = 0;
        }
        void sc(Node*c, int d) {
            ch[d] = c;
            c->p = this;
        }
        bool d() {
            return this == p->ch[1];
        }
        void pushup() {
            size = 1 + ch[0]->size + ch[1]->size;
        }
        void apply(int x) {
            add += x;
            val += x;
        }
        void pushdown();
        void setRoot(Node*f);
    } Tnull, *null = &Tnull;
    void Node::setRoot(Node*f) {
        fa = f;
        isRoot = true;
        p = null;
    }
    void Node::pushdown() {
        if (add) {
            rep(i, 0, 1)
                if (ch[i] != null)
                    ch[i]->apply(add);
            add = 0;
        }
    }
    Node*make(int v) {
        Node* C = new Node();
        C->val = v;
        C->add = 0;
        C->ch[0] = C->ch[1] = null;
        C->isRoot = true;
        C->p = null;
        C->fa = null;
        return C++;
    }
    void rot(Node*t) {
        Node*p = t->p;
        p->pushdown();
        t->pushdown();
        bool d = t->d();
        p->p->sc(t, p->d());
        p->sc(t->ch[!d], d);
        t->sc(p, !d);
        p->pushup();
```

```
        if (p->isRoot) {
            p->isRoot = false;
            t->isRoot = true;
            t->fa = p->fa;
        }
    }
    void pushTo(Node*t) {
        static Node*stk[LEN];
        int top = 0;
        while (t != null) {
            stk[top++] = t;
            t = t->p;
        }
        for (int i = top - 1; i >= 0; --i)
            stk[i]->pushdown();
    }
    void splay(Node*u, Node*f = null) {
        pushTo(u);
        while (u->p != f) {
            if (u->p->p == f)
                rot(u);
            else
                u->d() == u->p->d() ? (rot(u->p), rot(u)) : (rot(u),
rot(u));
        }
        u->pushup();
    }
    Node* expose(Node*u) {
        Node*v;
        for (v = null; u != null; v = u, u = u->fa) {
            splay(u);
            u->ch[1]->setRoot(u);
            u->sc(v, 1);
            v->fa = u;
        }
        return v;
    }
    void makeRoot(Node*u) {
        expose(u);
        splay(u);
    }
    void addEdge(Node*u, Node*v) {
        makeRoot(v);
        v->fa = u;
    }
    void delEdge(Node*u, Node*v) {
        makeRoot(u);
        expose(v);
        splay(u);
        u->sc(null, 1);
        u->pushup();
        v->fa = null;
        v->isRoot = true;
        v->p = null;
    }
    void markPath(Node*u, Node*v, int x) {
        makeRoot(u);
        expose(v);
        splay(v);
        v->apply(x);
    }
}

struct Q {
    Q *suf, *go[26], *nxt;
    int val;
```

```
    LCT::Node* tree;
    Q() :
            suf(0), val(0) {
        memset(go, 0, sizeof go);
        tree = LCT::make(0);
    }
}*root, *last;

void init() {
    root = last = new Q();
}

void extend(int w) {
    Q *p = last, *np = new Q();
    np->val = p->val + 1;
    while (p && !p->go[w])
        p->go[w] = np, p = p->suf;
    if (!p) {
        np->suf = root;
        LCT::addEdge(root->tree, np->tree);
    }
    else {
        Q *q = p->go[w];
        if (p->val + 1 == q->val) {
            np->suf = q;
            LCT::addEdge(q->tree, np->tree);
        } else {
            Q *nq = new Q();
            memcpy(nq->go, q->go, sizeof q->go);
            nq->val = p->val + 1;
            LCT::delEdge(q->suf->tree, q->tree);
            nq->suf = q->suf; LCT::addEdge(q->suf->tree, nq->tree);
            q->suf = nq;  LCT::addEdge(nq->tree, q->tree);
            np->suf = nq; LCT::addEdge(nq->tree, np->tree);
            LCT::pushTo(q->tree);
            nq->tree->val = q->tree->val;
            while (p && p->go[w] == q)
                p->go[w] = nq, p = p->suf;
        }
    }
    last = np;
    markPath(root->tree, np->tree, 1);
    //for (; np; np = np->suf)
    //  np->size++;
}
int query(char *s) {
    Q* now = root;
    for (; *s; s ++) {
        now = now->go[*s - 'A'];
        if (now == 0) return 0;
    }
    LCT::pushTo(now->tree);
    return now->tree->val;
}
    init();
```

MonotonousQueue.cpp

```
struct T {
    int pos, val;
    T(){}
    T(int pos, int val):pos(pos),val(val){}
    bool operator > (const T& a) const {
        return val > a.val;
    }
}q[6001000];
struct Q {
    int h, t;
    void init() {
        h = 1, t = 1;
    }
    void insert(const T& x) {
        while(t > h && x > q[t - 1]) t--;
        q[t++] = x;
    }
    void pop(int pos) {
        while(t > h && q[h].pos < pos) h++;
    }
    int get() {
        return q[h].val;
    }
}que;
```

Multipack.cpp

```
#include <cstdio>
#include <cstring>
const int N = 1001000, M = 1001000;
int n, m, v[N], c[N], f[M];
void init() {
    memset (f, 0, sizeof f);
}
void solve() {
    for (int i = 1; i <= n; i ++)
        scanf("%d", &v[i]);
    for (int i = 1; i <= n; i ++)
        scanf("%d", &c[i]);
    //w[i]
    f[0] = 1;
    for (int i = 1; i <= n; i ++)
        for (int d = 0; d < v[i]; d ++) {
            que.init();
            for (int k = 0; k <= (m - d) / v[i]; k ++) {
                que.insert(T(k, f[d + k * v[i]]));
                que.pop(k - c[i]);
                f[d + k * v[i]] = que.get();
            }
        }
    int ans = 0;
    for (int i = 1; i <= m; i ++)
        ans += f[i];
    printf("%d\n", ans);
}
```

Dinic.cpp

```
int S, T;
const int N = 500, M = 501000, INF = 0x3f3f3f3f;
struct Flow {
    int key[M], next[M], head[N], f[M], cnt, q[N], d[N];
    void init() {
        cnt = 0;
        memset (head, -1, sizeof head);
    }
    inline void add (int x, int y, int F)
    {
        key[cnt] = y;
        next[cnt] = head[x];
        f[cnt] = F;
        head[x] = cnt ++;

        key[cnt] = x;
        next[cnt] = head[y];
        f[cnt] = 0;
        head[y] = cnt ++;
    }
    bool SPFA ()
    {
        memset (d, -1, sizeof d);
        int h = 1, t = 2;
        q[1] = S;
        d[S] = 0;
        while (h < t)
        {
            int u = q[h ++];
            for (int i = head[u]; ~ i; i = next[i])
                if (f[i] && d[key[i]] == -1)
                    d[key[i]] = d[u] + 1, q[t ++] = key[i];
        }
        return d[T] != -1;
    }
    int DFS (int a, int b)
    {
        if (a == T)
            return b;
        int t (0), r (0);
        for (int i = head[a]; ~ i && r < b; i = next[i])
            if (f[i] && d[key[i]] == d[a] + 1)
            {
                t = DFS (key[i], min (b - r, f[i]));
                f[i] -= t, r += t, f[i ^ 1] += t;
            }
        if (!r) d[a] = -1;
        return r;
    }
    int work() {
        int z(0);
        while(SPFA())
            z += DFS(S, INF);
        return z;
    }
}flow;
```

Hungary.cpp

```
//优化：随机一个匹配增广；改邻接表
const int N = 1010;
bool vis[N], map[N][N];
int n, m, t, x, lnk[N], left[N];
bool DFS (int u)
{
    for (int v = 1; v <= m; v ++)
        if (map[u][v] && !vis[v])
        {
            vis[v] = true;
            if (lnk[v] == -1 || DFS (lnk[v]))
            {
                lnk[v] = u;
                return true;
            }
        }
    return false;
}

int hungary ()
{
    int ans (0);
    memset (lnk, -1, sizeof lnk);
    for (int i = 1; i <= n; i ++)
    {
        memset (vis, 0, sizeof vis);
        if (DFS (i))
            ans ++;
    }
    return ans;
}
```

```
int k;
int main() {
    scanf("%d%d%d", &n, &m, &k);
    for (int i = 1, a, b; i <= k; i ++)
        scanf("%d%d", &a, &b), map[a][b] = true;
    int ans = hungary();
    printf("%d\n", n + m - hungary());
    return 0;
}
```

KM.cpp

```
const int N = 1010, INF = 0x3f3f3f3f;
int w[N][N], lx[N], ly[N], match[N], slack[N];
bool vx[N], vy[N];
bool dfs(int i) {
    vx[i] = true;
    for (int j = 0; j < n; j ++) {
        if (lx[i] + ly[j] > w[i][j]) {
            slack[j] = min(slack[j], lx[i] + ly[j] - w[i][j]);
        } else if (!vy[j]) {
            vy[j] = true;
            if (match[j] < 0 || dfs(match[j])) {
                match[j] = i;
                return true;
            }
        }
    }
    return false;
}
int km() {
    memset(match, -1, sizeof match);
    memset(ly, 0, sizeof ly);
    for (int i = 0; i < n; i ++) lx[i] = *max_element(w[i], w[i] +
n);
    for (int i = 0; i < n; i ++) {
        while(1) {
            memset(vx, 0, sizeof vx);
            memset(vy, 0, sizeof vy);
            memset(slack, 0x3f, sizeof slack);
            if (dfs(i)) break;
            int d = 0x3f3f3f3f;
            for (int i = 0; i < n; i ++) {
                if (!vy[i]) d = min(d, slack[i]);
            }
            for (int i = 0; i < n; i ++) {
                if (vx[i]) lx[i] -= d;
                if (vy[i]) ly[i] += d;
            }
        }
    }
    int z = 0;
    for (int i = 0; i < n; i ++) {
        if (w[match[i]][i] == -INF) return -1;
        z += w[match[i]][i];
    }
    return z;
}
```

MatrixPow.cpp

```
int mod(int x) { return x % P; }
LL mod(LL x) { return x % P; }
struct Q {
    int s[N][N];
    Q () {
        memset (s, 0, sizeof s);
    }
    Q operator * (const Q& a) {
        Q c;
        for (int i = 0; i < N; i ++)
            for (int j = 0; j < N; j ++)
                for (int k = 0; k < N; k ++)
                    c.s[i][j]  =  mod(c.s[i][j]  +  mod(s[i][k]  *
a.s[k][j]));
        return c;
    }
};
Q qk(Q& A, LL k) {
    Q z;z.s[0][0] = z.s[1][1] = 1;
    for (; k; k >>= 1) {
        if (k & 1)
            z = z * A;
        A = A * A;
    }
    return z;
}
```

LinearBound.cpp

```
    for(int i = n; i >= 1; --i) {
        r[i] = i + 1;
        while (r[i] <= n && a[i] > a[r[i]]) r[i] = r[r[i]];
        if (r[i] <= n && a[i] == a[r[i]]) {
            c[i] = c[r[i]] + 1;
            r[i] = r[r[i]];
        }
    }
    for(int i = 1; i <= n; i ++) {
        l[i] = i - 1;
        while (l[i] >= 1 && a[i] > a[l[i]]) l[i] = l[l[i]];
        if (l[i] >= 1 && a[i] == a[l[i]]) {
            l[i] = l[l[i]];
        }
    }
```

Discretize.cpp

```
b[++b_c] = a[i];
sort (b + 1, b + 1 + b_c);
b_c = unique (b + 1, b + 1 + b_c) - (b + 1);
for (int i = 1; i <= n; i ++)
    a[i] = lower_bound (b + 1, b + 1 + b_c, a[i]) - b;
```

Pb_ds_tree.cpp

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
using namespace std;
using namespace __gnu_pbds;
const int N = 101000;
tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics
_node_update> st[N], ed[N];
    st[x[i]].order_of_key(t[i])
```

CountingSort.cpp

```
for (int i = 1; i <= n; i ++)
    a[i] = (1LL * a[i - 1] * A + B) % p, s[a[i]]++;
for (int i = 1; i <= p; i ++)
    s[i] += s[i - 1];
for (int i = n; i >= 1; i --)
    id[s[a[i]]--] = i;
```

BitOp.cpp

```
bool test(int s, int i) {
    return (s >> i) & 1;
}
void set(int& s, int i) {
    s |= (1 << i);
}
void flip(int &s, int i) {
    s ^= (1 << i);
}
void clear(int& s, int i) {
    if (test(s, i))
        flip(s, i);
}
int count(int s) {
    int z = 0;
    for (int i = 0; i < 8 * sizeof(s); i ++)
        if (test(s, i))
            z ++;
    return z;
}
```

Mode.cpp

```
struct Mode {
    int mx, c[N], cnt[N];
    void init() {
        memset(c, 0, sizeof c);
        memset(cnt, 0, sizeof cnt);
        mx = 0;
        cnt[0] = 0x3f3f3f3f;
    }
    void inc(int x) {
        cnt[c[x]]--;
        c[x]++;
        cnt[c[x]]++;
        mx = max(mx, c[x]);
    }
    void dec(int x) {
        cnt[c[x]]--;
        c[x]--;
        cnt[c[x]]++;
        while(cnt[mx] == 0) mx--;
    }
    int get() {
        return mx;
    }
}mode;
```

CountingColors.cpp

```
int n, m, a[N], s[N], ans[N], last[N];
struct Q {
    int l, r, id;
    bool operator<(const Q& a) const {
        return r < a.r;
    }
}b[N];
void update(int x, int v) {
    for (; x <= n; x += x & -x) s[x] += v;
}
int query(int x) {
    int z = 0;
    for (; x; x -= x & -x) z += s[x];
    return z;
}
int main() {
    scanf("%d", &n); scanf("%d", &m);
    rep(i, 1, n) scanf("%d", &a[i]);
    rep(i, 1, m) scanf("%d%d", &b[i].l, &b[i].r), b[i].id = i;
    sort(b + 1, b + 1 + m);int k = 1;
    rep(i, 1, n) {
        update(i, 1);
        if (last[a[i]]) update(last[a[i]], -1);
        last[a[i]] = i;
        while(b[k].r == i) {
            ans[b[k].id] = query(b[k].r) - query(b[k].l - 1),k++;
        }
    }
    rep(i, 1, m) printf("%d\n", ans[i]);
    return 0;
}
```

PowerSum.cpp & Fraction.cpp

```
const int N = 35;
struct Q {
    LL a, b;
    Q () { a = 0; b = 1; }
    Q (LL x) { a = x; b = 1; }
    Q (LL x, LL y) {
        a = x, b = y;
        uni();
    }
    void uni() {
        LL t = __gcd(a, b);
        a /= t;
        b /= t;
        if (b < 0) {
            a = -a;
            b = -b;
        }
    }
    Q operator + (const Q& x) {
        Q c;
        c.a = a * x.b + x.a * b;
        c.b = b * x.b;
        c.uni();
        return c;
    }
    Q operator * (const Q& x) {
        Q c;
        c.a = a * x.a;
        c.b = b * x.b;
        c.uni();
        return c;
    }
    void print() {
        printf("%lld/%lld ", a, b);
    }
}C[N][N], B[N];
int main() {
    for (int i = 1; i <= 31; i ++) {
        C[i][0] = C[i][i] = Q(1, 1);
        for (int j = 1; j < i; j ++)
            C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
    }
    B[0] = Q(1, 1);
    for (int i = 1; i <= 30; i ++) {
        B[i] = Q(0, 1);
        for (int j = 0; j < i; j ++)
            B[i] = B[i] + C[i + 1][j] * B[j];
        B[i] = B[i] * Q(-1, i + 1);
        //printf("%d ", i);B[i].print();

    }
    int m;
    scanf("%d", &m);
    for (int k = 0; k <= m; k ++) {
        Q t(1, m + 1);
        t = t * C[m + 1][k];
        t = t * B[k];
        if (k == 1) t = t + Q(1, 1);
        t.print();
    }
    return 0;
}
```

BigInt.cpp

```
const int LEN = 110, base = 10000;
struct Num {
    int s[LEN], len;
    Num() { len = 0; memset(s, 0, sizeof s); }
    Num(int x) {
        len = 1;
        memset (s, 0, sizeof s);
        s[1] = x;
    }
    int& operator[](int x) {
        return s[x];
    }
    int operator[](int x) const {
        return s[x];
    }
    void get() {
        LL x;
        cin >> x;
        while(x) {
            s[++len] = x % base;
            x /= base;
        }
    }
    void print() {
        printf("%d", s[len]);
        for (int i = len - 1; i >= 1; i --)
            printf("%04d", s[i]);
        printf("\n");
    }
};
Num operator+ (const Num& a, const Num& b) {
    Num c;
    c.len = max(a.len, b.len);
    for (int i = 1; i <= c.len; i ++) {
        c[i] += a[i] + b[i];
        c[i + 1] += c[i] / base;
        c[i] %= base;
    }
    if (c[c.len + 1]) c.len++;
    return c;
}
Num operator* (const Num& a, const Num& b) {
    Num c;
    c.len = a.len + b.len - 1;
    for (int i = 1; i <= a.len; i ++)
        for (int j = 1; j <= b.len; j ++) {
            c[i + j - 1] += a[i] * b[j];
            c[i + j] += c[i + j - 1] / base;
            c[i + j - 1] %= base;
        }
    if (c[c.len + 1]) c.len++;
    return c;
}
bool operator< (const Num& a, const Num& b) {
    if (a.len == b.len)
        for (int i = a.len; i >= 0; i --)
            if (a[i] != b[i])
                return a[i] < b[i];
    return a.len < b.len;
}
bool operator== (const Num& a, const Num& b) {
    if (a.len != b.len)
        return false;
    for (int i = a.len; i >= 0; i --)
```

```
        if (a[i] != b[i])
            return false;
    return true;
}
struct Q {
    Num x, y;
}a, b, A, B;
Q operator* (const Q& a, const Q& b) {
    Q c;
    c.x = a.x * b.x - a.y * b.y;
    c.y = a.x * b.y + a.y * b.x;
    return c;
}
int n, m;
int main() {
    freopen("a.in", "r", stdin);
    A.x.get();A.y.get();
    cin >> n;
    B.x.get();B.y.get();
    cin >> m;
    //(a + bi)(c + di) = (ac - bd) + (ad + bc)
    for (int i = 1; i <= m; i ++)
        A = A * a;
    for (int i = 1; i <= n; i ++)
        B = B * b;
    if (A.x == B.x && A.y == B.y)
        printf("%d\n", __gcd(n, m));
    else
        printf("0\n");
    return 0;
}
```

Hash.cpp

```
const int N = 400000, M = 5000000, HEAD = 399997, P = 1e9 + 7;
struct HASH {
    int cnt, head[N], next[M], len[M];
    LL key[M];
    HASH() {
        clear();
    }
    inline void clear() {
        memset (head, -1, sizeof head);cnt = 0;
    }
    inline void ADD(int x, LL y, int w) {
        key[cnt] = y;
        next[cnt] = head[x];
        len[cnt] = w;
        head[x] = cnt ++;
    }
    inline int GETHEAD(LL idx) {
        return idx % HEAD;
    }
    inline void add(LL idx, int w) {
        int h = GETHEAD(idx);
        ADD(h, idx, w);
    }
    bool find(LL idx, int w) {
        int h = GETHEAD(idx);
        for (int i = head[h]; ~ i; i = next[i])
            if (key[i] == idx && len[i] == w)
                return true;
        return false;
    }
}mp;
```

PAM.cpp

1.len[i]表示编号为 i 的节点表示的回文串的长度（一个节点表示一个回文串）

2.next[i][c]表示编号为 i 的节点表示的回文串在两边添加字符 c 以后变成的回文串的编号（和字典树类似）。

3.fail[i]表示节点 i 失配以后跳转不等于自身的节点 i 表示的回文串的最长后缀回文串（和 AC 自动机类似）。

4.cnt[i]表示节点 i 表示的本质不同的串的个数（建树时求出的不是完全的，最后 count()函数跑一遍以后才是正确的）

5.num[i]表示以节点 i 表示的最长回文串的最右端点为回文串结尾的回文串个数。

6.last 指向新添加一个字母后所形成的最长回文串表示的节点。

7.S[i]表示第 i 次添加的字符（一开始设 S[0] = -1（可以是任意一个在串 S 中不会出现的字符））。

8.p 表示添加的节点个数。

9.n 表示添加的字符个数。

```
const int N = 501000, M = 30;
struct PAM {
    int next[N][M], fail[N], cnt[N], len[N], S[N], num[N];
    int last, n, p;
    int newnode(int l) {
        for (int i = 0; i < M; i ++) next[p][i] = 0;
        cnt[p] = num[p] = 0;
        len[p] = l;
        return p++;
    }
    void init() {
        p = last = n = 0;
        newnode(0);newnode(-1);
        S[n] = -1;
        fail[0] = 1;
    }
    int get_fail(int x) {
        while(S[n - len[x] - 1] != S[n]) x = fail[x];
        return x;
    }
    void add(int c) {
        c -= 'a';
        S[++n] = c;
        int cur = get_fail(last);
        if (!next[cur][c]) {
            int now = newnode(len[cur] + 2);
            fail[now] = next[get_fail(fail[cur])][c];
            next[cur][c] = now;
            num[now] = num[fail[now]] + 1;
        }
        last = next[cur][c];
        cnt[last]++;
    }
    void count() {
        for (int i = p - 1; i >= 0; i --)
            cnt[fail[i]] += cnt[i];
    }

}pa, pb;
int main() {
    int T;
    scanf("%d", &T);
        scanf("%s", sa);
        int lb = strlen(sb);
        pa.init();
        for (int i = 0; i < la; i ++)
            pa.add(sa[i]);
        pa.count();
    }
    return 0;
}
```

SAM.cpp

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

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

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

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

TreeConquer.cpp

```
const int N = 501000, INF = 0x3f3f3f3f;
int key[N], next[N], len[N], head[N], cnt, f[N], sz[N], flag[N],
root[N], tot, K, n, t, ans;
vector<pi> son[N];
void add(int x, int y, int w) {
    key[cnt] = y;
    next[cnt] = head[x];
    len[cnt] = w;
    head[x] = cnt++;
}
void find(int u, int fa) {
    f[u] = 0, sz[u] = 1;
    edg(i, u) {
        int v = key[i];
        if (flag[v] || v == fa) continue;
        find(v, u);
        sz[u] += sz[v];
        f[u] = max(f[u], sz[v]);
    }
    f[u] = max(f[u], tot - sz[u]);
    if (f[u] < f[root[t]])
        root[t] = u;
}
bool cmp(pi x, pi y) {
    return sz[x.first] < sz[y.first];
}
void solve(int u) {
    flag[u] = t;
    if (tot == 1) return ;
    edg(i, u) {
        int v = key[i];
        if (flag[v]) continue;
        if (sz[v] > sz[u]) sz[v] = tot - sz[u];
        son[u].push_back(pi(v, len[i]));
    }
    sort(son[u].begin(), son[u].end(), cmp);
    edg(i, u) {
        int v = key[i];
        if (flag[v]) continue;
        ++t;tot = sz[v];
        find(v, 0);
        solve(root[t]);
    }
}
map<int, int> res, path;
map<int, int>::iterator it;
int dep1, dep2, clk;
void update(map<int, int>& mp, int key, int val) {
    if (mp.find(key) != mp.end()) mp[key] = min(mp[key], val);
    else mp[key] = val;
}
void DFS(int u, int fa, int dep, int dis) {
    update(path, dis, dep);
    edg(i, u) {
        int v = key[i];
        if (flag[v] <= clk || v == fa) continue;
        DFS(v, u, dep + 1, dis + len[i]);
    }
}
bool conquer() {
    for (clk = 1; clk <= n; clk++) {
        int u = root[clk];
        res[0] = 0;
        rep(i, 0, (int)son[u].size() - 1) {
```

```
            int v = son[u][i].first;
            int len = son[u][i].second;
            DFS(v, u, 1, len);
            ctn(it, path)
                if (res.find(K - it->first) != res.end())
                    ans = min(ans, it->second + res[K - it->first]);
            ctn(it, path)
                update(res, it->first, it->second);
            path.clear();
        }
        res.clear();
    }
    return 0;
}
int main() {
    memset (head, -1, sizeof head);
    n = read(), K = read();
    rep(i, 1, n - 1) {
        int a, b, c;
        a = read(), b = read(), c = read();
        ++a,++b;
        add(a, b, c);
        add(b, a, c);
    }
    ++t;
    tot = n;f[0] = INF;
    find(1, 0);
    solve(root[1]);
    ans = 0x3f3f3f3f;
    conquer();
    printf("%d\n", ans != 0x3f3f3f3f ? ans : -1);
    return 0;
}
```

LCA.cpp

```
const int N = 501000, M = 601000;
int key[M], nxt[M], head[N], cnt;
void add(int x, int y) {
    key[cnt] = y;
    nxt[cnt] = head[x];
    head[x] = cnt++;
}
struct LCA {
    int f[N][25], d[N], q[N];
    void BFS(int S) {
        int h = 1, t = 2, u;
        q[1] = S;
        while(h < t) {
            u = q[h++];
            for (int i = head[u]; ~ i; i = nxt[i]) {
                int v = key[i];
                if (d[v] != 0) continue;
                f[v][0] = u, d[v] = d[u] + 1;
                q[t++] = v;
            }
        }
    }
    void init(int n) {
        d[1] = 1;
        BFS(1);
        for (int j = 1; j <= 20; j ++)
            for (int i = 1; i <= n; i ++) {
                f[i][j] = f[f[i][j - 1]][j - 1];
            }
    }
    int get(int x, int y) {
        if (d[x] < d[y])
            swap(x, y);
        for (int j = 20; j >= 0; j --)
            if (d[f[x][j]] >= d[y])
                x = f[x][j];
        if (x == y)
            return x;                //return
        for (int j = 20; j >= 0; j --)
            if (f[x][j] != f[y][j])
                x = f[x][j], y = f[y][j];
        return f[x][0];              //return
    }
    int dis(int x, int y) {
        int t = get(x, y);
        return d[x] - d[t];
    }
}lca;
```

AC.cpp

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

int pointer;
struct Node {
    Node*ch[CHARSET], *fail, *par;
    Node() {
        memset(ch, 0, sizeof ch);
        fail = 0;
    }
    Node*go(int w);
}*root;
Node nodePool[MAX_N_NODES], *cur;
Node*newNode() {
    Node*t = cur++;
    memset(t->ch, 0, sizeof t->ch);
    t->fail = 0;
    return t;
}
Node* Node::go(int w) {
    if (ch[w] == 0) {
        ch[w] = newNode();
        ch[w]->par = this;
    }
    return ch[w];
}
void init() {
    cur = nodePool;
    root = newNode();
    root->par = 0;
}
void build() {
    static Node*que[MAX_N_NODES];
    int qh = 0, qt = 0;
    que[qt++] = root;
    while (qh < qt) {
        Node*t = que[qh++];
        for (int c = 0; c < CHARSET; ++c) {
            Node*v = t->ch[c];
            if (!v)
                continue;
            Node*f = t->fail;
            while (f && f->ch[c] == 0)
                f = f->fail;
            if (f == 0)
                v->fail = root;
            else
                v->fail = f->ch[c];
            que[qt++] = v;
        }
    }
}
```

Geo.cpp

```
const double eps = 1e-10;
int dcmp(double x) {
    return x < -eps ? -1 : x > eps;
}
const double pi = acos(-1.0);
inline double sqr(double x) {
    return x * x;
}
struct point {
    double x, y;
    point() : x(0), y (0) {}
    point(double a, double b) : x(a), y(b) {}
    void input() {
        scanf("%lf%lf", &x, &y);
    }
    void print() {
        printf("%lf %lf\n", x, y);
    }
    friend point operator +(const point &a, const point &b) {
        return point(a.x + b.x, a.y + b.y);
    }
    friend point operator -(const point &a, const point &b) {
        return point(a.x - b.x, a.y - b.y);
    }
    friend bool operator ==(const point &a, const point &b) {
        return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0;
    }
    friend point operator *(const double &a, const point &b) {
        return point(a * b.x, a * b.y);
    }
    friend point operator *(const point &b, const double &a) {
        return point(a * b.x, a * b.y);
    }
    friend point operator /(const point &a, const double &b) {
        return point(a.x / b, a.y / b);
    }
    double norm() const {
        return sqrt(sqr(x) + sqr(y));
    }
};
double det(const point &a, const point &b) {
    return a.x * b.y - a.y * b.x;
}
double dot(const point &a, const point &b) {
    return a.x * b.x + a.y * b.y;
}
double dis(const point &a, const point &b) {
    return (a - b).norm();
}
point rotate(const point &p, double A) {
    double tx = p.x, ty = p.y;
    return point(tx * cos(A) - ty * sin(A), tx * sin(A) + ty * cos(A));
}
struct line {
    point a, b;
    double ang;
    line() {}
    line(point x, point y) : a(x), b(y) {
        ang = atan2(b.y - a.y, b.x - a.x);
    }
    void input() {
        a.input();
        b.input();
    }
```

```
};
//line and seg are different
double dis(const point p, const point s, const point t) {
    if (dcmp(dot(p - s, t - s)) == -1) return (p - s).norm();
    if (dcmp(dot(p - t, s - t)) == -1) return (p - t).norm();
    return fabs(det(s - p, t - p) / dis(s, t));
}
void proj(const point p, const point s, const point t, point &cp)
{
    double r = dot((t - s), (p - s)) / dot(t - s, t - s);
    cp = s + r * (t - s);
}
bool onseg(point p, point s, point t) {
    return dcmp(det(p - s, t - s)) == 0 && dcmp(dot(p - s, p - t))
<= 0;
}
bool parallel(line a, line b) {
    return dcmp(det(a.a - a.b, b.a - b.b)) == 0;
}
bool inter(line a, line b) {
    double c1 = det(b.a - a.a, a.b - a.a), c2 = det(b.b - a.a, a.b
- a.a);
    double c3 = det(a.a - b.a, b.b - b.a), c4 = det(a.b - b.a, b.b
- b.a);
    return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}
point interpoint(line a, line b) {
    //if (inter(a, b) == false) return false;
    point u = a.a - b.a;
    point v = a.b - a.a;
    point w = b.b - b.a;
    double t = det(w, u) / det(v, w);
    return a.a + v * t;
}
line move_d(line a, const double &len) {
    point d = a.b - a.a;
    d = d / d.norm();
    d = rotate(d, pi / 2);
    return line(a.a + d * len, a.b + d * len);
}
bool cmpxy(const point &a, const point &b) {
    if (dcmp(a.x - b.x) == 0)
        return a.y < b.y;
    return a.x < b.x;
}
#define points vector<point>
#define lines vector<line>
#define next(x) ((x) + 1) % n
double area(points &p) {
    double z = 0;
    for (int i = 0; i < (int)p.size() - 1; i ++)
        z += det(p[i] - p[0], p[i + 1] - p[0]);
    return fabs(z) / 2;
}
void convex_hull(points &a, points &res) {
    res.resize(2 * a.size() + 10);
    sort(a.begin(), a.end(), cmpxy);
    a.erase(unique(a.begin(), a.end()), a.end());
    int m = 0;
    for (int i = 0; i < (int)a.size(); i ++) {
        while(m > 1 && dcmp(det(res[m - 1] - res[m - 2], a[i] - res[m
- 2])) <= 0) --m;
        res[m++] = a[i];
    }
    int k = m;
    for (int i = (int)a.size() - 2;  i >= 0; i --) {
```

```
        while(m > k && dcmp(det(res[m - 1] - res[m - 2], a[i] - res[m
- 2])) <= 0) --m;
        res[m++] = a[i];
    }
    res.resize(m);
    //if (a.size() > 1) res.resize(m - 1);
}
void cut(points &p, point b, point a, points &res) {
    res.clear();
    int n = p.size();
    for (int i = 0; i < n; i ++) {
        point c = p[i];
        point d = p[next(i)];
        if (dcmp(det(b - a, c - a)) >= 0) res.push_back(c);
        if (dcmp(det(b - a, c - d)) != 0) {
            point cp = interpoint(line(a, b), line(c, d));
            if (onseg(cp, c, d)) res.push_back(cp);
        }
    }
}
bool onleft(point a, line p) {
    return dcmp(det(a - p.a, p.b - p.a)) < 0;
}
bool cmpang(const line &a, const line &b) {
    if (dcmp(a.ang - b.ang) == 0)
        return onleft(a.a, b);
    return a.ang < b.ang;
}
int halfplane(lines &v, points &res) {
    sort(v.begin(), v.end(), cmpang);
    deque<line> q;
    deque<point> ans;
    q.push_back(v[0]);
    for (int i = 1; i < int(v.size()); i ++) {
        if (dcmp(v[i].ang - v[i - 1].ang) == 0) continue;
        while(ans.size() && !onleft(ans.back(), v[i])) {
            ans.pop_back();
            q.pop_back();
        }
        while(ans.size() && !onleft(ans.front(), v[i])) {
            ans.pop_front();
            q.pop_front();
        }
        ans.push_back(interpoint(q.back(), v[i]));
        q.push_back(v[i]);
    }
    while(ans.size() && !onleft(ans.back(), q.front())) {
        ans.pop_back();
        q.pop_back();
    }
    while(ans.size() && !onleft(ans.front(), q.back())) {
        ans.pop_front();
        q.pop_front();
    }
    ans.push_back(interpoint(q.back(), q.front()));
    res = points(ans.begin(), ans.end());
    return ans.size(); //you must use the size to assure an empty set,
area dont has the accuracy we need
}
const int N = 50010;
point p[N];
int n;
double r;
void init() {
}
double mysqrt(double x) {
```

```
    return sqrt(max(0.0, x));
}
void circle_inter_line(point a, point b, point o, double r, point
ret[], int &num) {
    point p = b - a;
    point q = a - o;
    double A = dot(p, p);
    double B = 2 * dot(p, q);
    double C = dot(q, q) - sqr(r);
    double delta = B * B - 4 * A * C;
    num = 0;
    if (dcmp(delta) >= 0) {
        double t1 = (-B - mysqrt(delta)) / (2 * A);
        double t2 = (-B + mysqrt(delta)) / (2 * A);
        if (t1 <= 1 && t1 >= 0) {
            ret[num++] = a + t1 * p;
        }
        if (t2 <= 1 && t2 >= 0) {
            ret[num++] = a + t2 * p;
        }
    }
}
double sector_area(const point &a, const point &b) {
    double theta = atan2(a.y, a.x) - atan2(b.y, b.x);
    while(theta <= 0) theta += 2 * pi;
    while(theta > 2 * pi) theta -= 2 * pi;
    theta = min(theta, 2 * pi - theta);
    return r * r * theta / 2;
}
double calc(const point &a, const point &b) {
    point p[2];
    int num = 0;
    int ina = dcmp(a.norm()- r) < 0;
    int inb = dcmp(b.norm() - r) < 0;
    if (ina) {
        if (inb) {
            return fabs(det(a, b)) / 2;
        } else {
            circle_inter_line(a, b, point(0, 0), r, p, num);
            return sector_area(b, p[0]) + fabs(det(a, p[0])) / 2;
        }
    } else {
        if (inb) {
            circle_inter_line(a, b, point(0, 0), r, p, num);
            return sector_area(p[0], a) + fabs(det(p[0], b)) / 2;
        } else {
            circle_inter_line(a, b, point(0, 0), r, p, num);
            if (num == 2) {
                return sector_area(a, p[0]) + sector_area(p[1], b) +
fabs(det(p[0], p[1])) / 2;
            } else {
                return sector_area(a, b);
            }
        }
    }
}
double area() {
    double ret = 0;
    for (int i = 0; i < n; i ++) {
        int sgn = dcmp(det(p[i], p[i + 1]));
        if (sgn) {
            ret += sgn * calc(p[i], p[i + 1]);
        }
    }
    return ret;
}
```

```
void solve() {
    scanf("%d", &n);
    for (int i = 0; i < n; i ++)
        p[i].input();
    p[n] = p[0];
    printf("%.2lf\n", fabs(area()) + eps);
}
```

CircleTracing.cpp

```
#include <bits/stdc++.h>
const int N = 201000;
double Time;
struct Q {
	int x, y, r;
	double getX(bool flag) {
		if (flag) return x + r;
		else return x - r;
	}
	double getY(bool flag) {
		double ret = sqrt(1.0 * r * r - (Time - x) * (Time - x));
		if (flag) return   y + ret;
		else return y - ret;
	}
	void scan() {
		scanf("%d%d%d", &x, &y, &r);
	}
}p[N];
struct E {
	int x, y, id;
	bool flag;
	E() {}
	E(int x, int y, int id, bool flag) : x(x), y(y), id(id), flag(flag) {}
	bool operator < (const E& a) const {
		return x == a.x ? y < a.y : x < a.x;
	}
}event[N];
struct P {
	int id;
	bool flag;
	P () {}
	P (int id, bool flag) : id(id), flag(flag) {}
	bool operator < (const P& a) const {
		double y1 = p[id].getY(flag);
		double y2 = p[a.id].getY(a.flag);
		return y1 < y2 || y1 == y2 && flag < a.flag;
	}
	bool operator == (const P& a) const {
		return id == a.id;
	}
};
set<P> ss;
set<P>::iterator up, dn;
int n, fa[N], key[N], nxt[N], head[N], cnt;
void add(int x, int y) {
	key[cnt] = y;
	nxt[cnt] = head[x];
	head[x] = cnt++;
	fa[y] = x;
}
void init() {
	ss.clear();
	cnt = 0;
	memset (head, -1, sizeof head);
}
int sg(int u) {
	if (head[u] == -1) return 0;
	int t = 0;
	for (int i = head[u]; ~ i; i = nxt[i]) {
		int v = key[i];
		t = t ^ (sg(v) + 1);
	}
```

```
	return t;
}
void solve() {
	scanf("%d", &n);
	for (int i = 1; i <= n; i ++) {
		p[i].scan();
		event[i] = E(p[i].getX(0), p[i].y, i, 0);
		event[n + i] = E(p[i].getX(1), p[i].y, i, 1);
	}
	sort(event + 1, event + 1 + n + n);
	for (int i = 1; i <= n; i ++)
		fa[i] = i;
	for (int i = 1; i <= n + n; i ++) {
		Time = event[i].x;
		if (event[i].flag == 0) {
			if (ss.empty()) {
				ss.insert(P(event[i].id, 0));
				ss.insert(P(event[i].id, 1));
				add(0, event[i].id);
				continue;
			}
			up = ss.upper_bound(P(event[i].id, 1));
			dn = ss.lower_bound(P(event[i].id, 0));
			if (dn == ss.begin() || up == ss.end())
				add(0, event[i].id);
			else {
				dn--;
				int t1 = up->id;
				int t2 = dn->id;
				if (t1 == t2)
					add(t1, event[i].id);
				else if (fa[t1] == fa[t2])
					add(fa[t1], event[i].id);
				else if (fa[t1] == t2)
					add(t2, event[i].id);
				else if (fa[t2] == t1)
					add(t1, event[i].id);
				else
					add(0, event[i].id);
			}
			ss.insert(P(event[i].id, 0));
			ss.insert(P(event[i].id, 1));
		} else {
			ss.erase(ss.find(P(event[i].id, 0)));
			ss.erase(ss.find(P(event[i].id, 1)));
		}
	}
	for (int i = 1; i <= n; i ++) {
		printf("%d: ", i);
		for (int j = head[i]; ~ j; j = nxt[j])
			printf("%d ", key[j]);
		printf("\n");
	}
	printf("%s\n", sg(0) ? "Alice" : "Bob");
}
int main()
{
	freopen("L.in", "r", stdin);
	int T;
	scanf("%d", &T);
	while(T --) {
		init();
		solve();
	}
	return 0;
}
```

PrimitiveRoot.cpp

```
typedef long long LL;
const int N = 4001000;
int P, n, a_c;
int a[N];
LL pw(LL a, int k) {
    LL z(1);
    for (; k; k >>= 1) {
        if (k & 1) z = z * a % P;
        a = a * a % P;
    }
    return z;
}
bool vis[N];
int pr[N];
void getpr() {
    const int N = 4000000;
    memset (vis, 0, sizeof vis);
    int cnt = 0;
    for (int i = 2; i <= N; i ++) {
        if (!vis[i])
            pr[++ cnt] = i;
        for (int j = 1; j <= cnt; j ++) {
            if (i * pr[j] > N) break;
            vis[i * pr[j]] = true;
            if (i % pr[j] == 0) break;
        }
    }
}
void divide(int n) {
    for (int i = 1; pr[i] * pr[i] <= n; i ++) {
        if (n % pr[i] != 0) continue;
        a[++ a_c] = pr[i];
        while(n % pr[i] == 0) n /= pr[i];
    }
    if (n != 1) a[++ a_c] = n;
}
bool ck(int x) {
    for (int i = 1; i <= a_c; i ++)
        if (pw(x, n / a[i]) == 1)
            return false;
    return true;
}
int main() {
    scanf("%d", &n);
    getpr();
    divide(n - 1);
    P = n;
    int i;
    for (i = 2; ; i ++) {
        if (ck(i))
            break;
    }
    printf("%d\n", i);
    return 0;
}
```

Log.cpp

```
typedef long long LL;
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}
int pw(LL x, int k, LL p) {
    LL z = 1;
    for (; k; k >>= 1) {
        if (k & 1) z = z * x % p;
        x = x * x % p;
    }
    return z;
}
const int N = 40000, M = 100000, HEAD = 39997;
struct HASH {
    int cnt, head[N], next[M], len[M], key[M];
    HASH() {
        clear();
    }
    inline void clear() {
        memset (head, -1, sizeof head);
        cnt = 0;
    }
    inline void ADD(int x, int y, int w) {
        key[cnt] = y;
        next[cnt] = head[x];
        len[cnt] = w;
        head[x] = cnt ++;
    }
    inline int GETHEAD(int idx) {
        return idx % HEAD;
    }
    inline void add(int idx, int val) {
        int h = GETHEAD(idx);
        ADD(h, idx, val);
    }
    bool find(int idx) {
        int h = GETHEAD(idx);
        for (int i = head[h]; ~ i; i = next[i])
            if (key[i] == idx)
                return true;
        return false;
    }
    int get(int idx) {
        int h = GETHEAD(idx);
        for (int i = head[h]; ~ i; i = next[i])
            if (key[i] == idx)
                return len[i];
    }
};
struct HASH hash;
int BSGS(int a, int b, int p) {
    a %= p, b %= p;
    if (b == 1) return 0;
    int cnt = 0;
    LL t = 1;
    for (int g = gcd(a, p); g != 1; g = gcd(a, p)) {
        if (b % g) return -1;
        p /= g, b /= g, t = t * a / g % p;
        ++cnt;
        if (b == t) return cnt;
    }
    hash.clear();
    int m = int(sqrt(1.0 * p) + 0.5);
    LL base = b;
```

```
    for (int i = 0; i < m; i ++) {
        hash.add(base, i);
        base = base * a % p;
    }
    base = pw(a, m, p);
    LL now = t;
    for (int i = 1; i <= m + 1; ++i) {
        now = now * base % p;
        if (hash.find(now))
            return i * m - hash.get(now) + cnt;
    }
    return -1;
}
int main() {
    freopen("a.in", "r", stdin);
    int a, b, p;
    while(scanf("%d%d%d", &p, &a, &b) != EOF) {
        int tmp = BSGS(a, b, p);
        if (tmp == -1) printf("no solution\n");
        else printf("%d\n", tmp);
    }
    return 0;
}
```

Binomial.cpp

```
const int N = 1001000, P = 1e9 + 7;
LL inv[N], fac[N], faci[N];
LL C(int n, int m) {
    if (n < 0 || m < 0 || m > n) return 0;
    return fac[n] * faci[n - m] % P * faci[m] % P;
}
void pre() {
    const int P = 1e9 + 7, N = 1000000;
    inv[1] = 1;
    rep(i, 2, N) inv[i] = (P - P / i) * inv[P % i] % P;
    fac[0] = 1;
    rep(i, 1, N) fac[i] = fac[i - 1] * i % P;
    faci[0] = 1;
    rep(i, 1, N) faci[i] = faci[i - 1] * inv[i] % P;
}
```

Quadratic.cpp

```
LL D, P;
struct Q{
    LL a, b;
    Q(LL _a = 0, LL _b = 0) : a(_a), b(_b) {}
    Q operator* (const Q& p) {
        return Q((a * p.a % P + b * p.b % P * D % P) % P, (a * p.b % P + p.a * b % P) % P);
    }
};
LL qk(LL x, LL k) {
    LL z(1);
    for (; k; k >>= 1) {
        if (k & 1) z = z * x % P;
        x = x * x % P;
    }
    return z;
}
LL qk(Q x, LL k) {
    Q z(1, 0);
    for (; k; k >>= 1) {
        if (k & 1) z = z * x;
        x = x * x;
    }
    return z.a;
}
LL L(LL a) {
    return qk(a, (P - 1) / 2) == 1;
}
LL solve(LL n) {
    //P == 2 special judge
    if (P == 2) {
        if (n == 1) return 1;
        else return -1;
    }
    if (!L(n)) return -1;
    LL a;
    while(1) {
        a = rand() % P;
        D = ((a * a - n) % P + P) % P;
        if (!L(D)) break;
    }
    return qk(Q(a, 1), (P + 1) / 2);
}
int main() {
    srand(time(0));
    int T;
    scanf("%d", &T);
    while(T --) {
        int a, n, t;
        scanf("%d%d", &a, &n);
        a %= n;
        P = n;
        t = solve(a);
        if (t == -1)
            puts("No root");
        else {
            if (t == n - t)
                printf("%d\n", t);
            else
                printf("%d %d\n", min(t, n - t), max(t, n - t));
        }
    }
    return 0;
}
```

Initial.cpp

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i, s, t) for (int i = s; i <= t; i++)
#define dwn(i, s, t) for (int i = s; i >= t; i--)
#define edg(i, x) for (int i = head[x]; ~ i; i = next[i])
#define ctn(i, x) for (i = x.begin(); i != x.end(); i++)
#define clr(x) memset ((x), 0, sizeof (x))
#define size(x) (int)x.size()
typedef long long LL;
int read()
{
    int x=0,f=1;char ch=getchar();
    while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}
    while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}
    return x*f;
}
void print(LL x) {
    static int a[24];int n = 0;
    while(x > 0) {
        a[n++] = x % 10;
        x /= 10;
    }
    if (n == 0) a[n++] = 0;
    while(n--) putchar('0' + a[n]);
    putchar('\n');
}
void from(const char *s) {
    freopen(s, "r", stdin);
}
```