ACM/ICPC Template

ZheJiang GongShang University

linbinwu

2020 年 12 月 7 日

目录

0 其他

0.1 BigInteger

```
struct BigInteger {
    typedef unsigned long long LL;

    static const int BASE = 100000000;
    static const int WIDTH = 8;
    vector<int> s;

    BigInteger& clean() { while (!s.back() && s.size()>1)s.pop_back(); return *this; }
    BigInteger(LL num = 0) { *this = num; }
    BigInteger(string s) { *this = s; }
    BigInteger& operator = (long long num) {
        s.clear();
        do {
            s.push_back(num % BASE);
            num /= BASE;
        } while (num > 0);
        return *this;
    }
    BigInteger& operator = (const string& str) {
        s.clear();
        int x, len = (str.length() - 1) / WIDTH + 1;
        for (int i = 0; i < len; i++) {
            int end = str.length() - i*WIDTH;
            int start = max(0, end - WIDTH);
            sscanf(str.substr(start, end - start).c_str(), "%d", &x);
            s.push_back(x);
        }
        return (*this).clean();
    }

    BigInteger operator + (const BigInteger& b) const {
        BigInteger c; c.s.clear();
        for (int i = 0, g = 0; ; i++) {
            if (g == 0 && i >= s.size() && i >= b.s.size()) break;
            int x = g;
            if (i < s.size()) x += s[i];
            if (i < b.s.size()) x += b.s[i];
            c.s.push_back(x % BASE);
            g = x / BASE;
        }
        return c;
    }
    BigInteger operator - (const BigInteger& b) const {
        assert(b <= *this); // 减数不能大于被减数
        BigInteger c; c.s.clear();
        for (int i = 0, g = 0; ; i++) {
            if (g == 0 && i >= s.size() && i >= b.s.size()) break;
            int x = s[i] + g;
            if (i < b.s.size()) x -= b.s[i];
            if (x < 0) { g = -1; x += BASE; }
```

```
        else g = 0;
        c.s.push_back(x);
    }
    return c.clean();
}
BigInteger operator * (const BigInteger& b) const {
    int i, j; LL g;
    vector<LL> v(s.size() + b.s.size(), 0);
    BigInteger c; c.s.clear();
    for (i = 0; i<s.size(); i++) for (j = 0; j<b.s.size(); j++) v[i + j] += LL(s[i])*b.s[j];
    for (i = 0, g = 0; ; i++) {
        if (g == 0 && i >= v.size()) break;
        LL x = v[i] + g;
        c.s.push_back(x % BASE);
        g = x / BASE;
    }
    return c.clean();
}
BigInteger operator / (const BigInteger& b) const {
    assert(b > 0);  // 除数必须大于0
    BigInteger c = *this;       // 商:主要是让c.s和(*this).s的vector一样大
    BigInteger m;               // 余数:初始化为0
    for (int i = s.size() - 1; i >= 0; i--) {
        m = m*BASE + s[i];
        c.s[i] = bsearch(b, m);
        m -= b*c.s[i];
    }
    return c.clean();
}
BigInteger operator % (const BigInteger& b) const { //方法与除法相同
    BigInteger c = *this;
    BigInteger m;
    for (int i = s.size() - 1; i >= 0; i--) {
        m = m*BASE + s[i];
        c.s[i] = bsearch(b, m);
        m -= b*c.s[i];
    }
    return m;
}
// 二分法找出满足bx<=m的最大的x
int bsearch(const BigInteger& b, const BigInteger& m) const {
    int L = 0, R = BASE - 1, x;
    while (1) {
        x = (L + R) >> 1;
        if (b*x <= m) { if (b*(x + 1)>m) return x; else L = x; }
        else R = x;
    }
}
BigInteger& operator += (const BigInteger& b) { *this = *this + b; return *this; }
BigInteger& operator -= (const BigInteger& b) { *this = *this - b; return *this; }
BigInteger& operator *= (const BigInteger& b) { *this = *this * b; return *this; }
BigInteger& operator /= (const BigInteger& b) { *this = *this / b; return *this; }
BigInteger& operator %= (const BigInteger& b) { *this = *this % b; return *this; }
```

```cpp

    bool operator < (const BigInteger& b) const {
        if (s.size() != b.s.size()) return s.size() < b.s.size();
        for (int i = s.size() - 1; i >= 0; i--)
            if (s[i] != b.s[i]) return s[i] < b.s[i];
        return false;
    }
    bool operator >(const BigInteger& b) const { return b < *this; }
    bool operator<=(const BigInteger& b) const { return !(b < *this); }
    bool operator>=(const BigInteger& b) const { return !(*this < b); }
    bool operator!=(const BigInteger& b) const { return b < *this || *this < b; }
    bool operator==(const BigInteger& b) const { return !(b < *this) && !(b > *this); }
};

ostream& operator << (ostream& out, const BigInteger& x) {
    out << x.s.back();
    for (int i = x.s.size() - 2; i >= 0; i--) {
        char buf[20];
        sprintf(buf, "%08d", x.s[i]);
        for (int j = 0; j < strlen(buf); j++) out << buf[j];
    }
    return out;
}

istream& operator >> (istream& in, BigInteger& x) {
    string s;
    if (!(in >> s)) return in;
    x = s;
    return in;
}

int main()
{
    int t;
    scanf("%d", &t);
    while (t--)
    {
        BigInteger a, b;
        cin >> a >> b;
        cout << a + b << endl;
    }
}
```

0.2 bound

```cpp

//get pos of first num >= x
int pos(int x) { return lower_bound(b + 1, b + 1 + N, x) - b; }

//get pos of first num > x
int pos(int x) { return upper_bound(b + 1, b + 1 + N, x) - b; }
```

0.3 Cantor

```
//主要应用为将N维的排列状态压缩成数字id
//然后需要知道具体状态时用逆Cantor得到

int N;
int a[MAX], c[MAX];

void upd(int p, int k) { for (; p <= N; p += lowbit(p)) c[p] += k; }
int query(int p) {
    int res = 0;
    for (; p; p -= lowbit(p)) res += c[p];
    return res;
}

int cantor() {
    //ans = 1 + \sum_{i = 1} ^ {N} fac[N - i] * (\sum_{j = i + 1} ^ {N} x[i] > x[j])
    int res = 0, fac = 1;
    for (int i = N; i >= 1; i--) {
        upd(a[i], 1);
        res = (res + 1ll * fac * query(a[i] - 1) % mod) % mod;
        fac = 1ll * fac * (N - i + 1) % mod;
    }
    return res + 1;
}

//逆Cantor
#define lc  u<<1
#define rc  u<<1|1
#define mid (l+r)/2
int sum[MAX << 4];
void push_up(int u) { sum[u] = sum[lc] + sum[rc]; }
void build(int u, int l, int r) {
    if (l == r) {
        sum[u] = 1;
        return;
    }
    build(lc, l, mid); build(rc, mid + 1, r);
    push_up(u);
}
int query(int u, int l, int r, int k) {//查找第k大并且删除该数
    sum[u]--;
    if (l == r) return l;
    if (k <= sum[lc]) return query(lc, l, mid, k);
    else return query(rc, mid + 1, r, k - sum[lc]);
}

vector<int> inCantor(int x, int n) {
    x--;
    vector<int> res;
    ll fac = 1;
    build(1, 1, n);
    for (int i = 1; i <= n; i++) fac = fac * i;
    for (int i = 1; i <= n; i++) {
```

```
        fac = fac / (n - i + 1);
        int k = x / fac + 1;//比当前这位大的有x / fac位
        res.push_back(query(1, 1, n, k));//找到没被选的第k大
        x %= fac;
    }
    return res;
}
```

0.4 Hash

```
struct Hash {
    int b[N], tot;
    void init() { tot = 0; }
    void insert(int x) { b[++tot] = x; }
    void build() {
        sort(b + 1, b + 1 + tot);
        tot = unique(b + 1, b + 1 + tot) - (b + 1);
    }
    int pos(int x) { return lower_bound(b + 1, b + 1 + tot, x) - b; }
};
```

0.5 RMQ

```
const int MAX_BIT = 20;

int dp[N][MAX_BIT];

void build(int siz) {
    for (int i = 1; i <= siz; i++) cin >> dp[i][0];
    for (int j = 1; j < MAX_BIT; j++)
        for (int i = 1; i + (1 << j) - 1 <= siz; i++)
            dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
}

int query(int ql, int qr) {
    int k = (int)log2(qr - ql + 1);
    return min(dp[ql][k], dp[qr - (1 << k) + 1][k]);
}
```

0.6 set

```
set<int> st;
set<int>::iterator now;

int lower(int x) {
    now = st.find(x);
    if (now == st.begin()) return -1;
    return *(--now);
}

int upper(int x) {
    now = st.find(x); now++;
```

```
    if (now != st.end()) return *now;
    else return -1;
}
```

0.7 四维偏序

```
//四维偏序查最大值
int N;
struct node {
    int a, b, c, d;
    inline bool operator < (const node &rhs) const {//四维排序掉一维
        return a < rhs.a || (a == rhs.a && (b < rhs.b || (b == rhs.b && (c < rhs.c || (c == rhs.c && d < rhs.d)))));
    }
} nod[MAX];

struct point {
    int x, y, z, val;
} p[MAX];

inline point max(const point &a, const point &b) {
    return point{max(a.x, b.x), max(a.y, b.y), max(a.z, b.z)};
}

inline point min(const point &a, const point &b) {
    return point{min(a.x, b.x), min(a.y, b.y), min(a.z, b.z)};
}

int cmptype;

inline bool operator < (const point &a, const point &b) {
    if (cmptype == 0) return a.x <  b.x;
    else if (cmptype == 1) return a.y < b.y;
    else return a.z < b.z;
}

struct K_D_Tree{
    point pos, lpos, rpos;
    int mx, siz;
    int ls, rs;
} t[MAX];

int root, tot, top, rub[MAX];

inline int newnode() {
    if (top) return rub[top--];
    else return ++tot;
}

inline void push_up(int u) {
    t[u].lpos = t[u].rpos = t[u].pos;
    t[u].mx = t[u].pos.val;
    t[u].siz = 1;
```

```
    if (lc) {
        t[u].siz += t[lc].siz;
        t[u].lpos = min(t[u].lpos, t[lc].lpos);
        t[u].rpos = max(t[u].rpos, t[lc].rpos);
        t[u].mx = max(t[u].mx, t[lc].mx);
    }
    if (rc) {
        t[u].siz += t[rc].siz;
        t[u].lpos = min(t[u].lpos, t[rc].lpos);
        t[u].rpos = max(t[u].rpos, t[rc].rpos);
        t[u].mx = max(t[u].mx, t[rc].mx);
    }
}

inline int build(int l, int r, int type) {//建树
    if (l > r) return 0;
    cmptype = type;
    nth_element(p + l, p + m, p + r + 1);
    int u = newnode();
    t[u].pos = p[m];
    t[u].ls = build(l, m - 1, (type + 1) % 3);
    t[u].rs = build(m + 1, r, (type + 1) % 3);
    push_up(u);
    return u;
}

inline void pia(int u, int num) {//拍扁回炉重做
    if (lc) pia(lc, num);
    p[t[lc].siz + num + 1] = t[u].pos, rub[++top] = u;
    if (rc) pia(rc, t[lc].siz + num + 1);
}

inline void check(int &u, int type) {//检查是否平衡，不平衡则需要重建
    if (t[u].siz * alpha < t[lc].siz || t[u].siz * alpha < t[rc].siz) pia(u, 0), u = build
    (1, t[u].siz, type);
}

inline void insert(int &u, int type, point tp) {
    if (!u) {//新点
        u = newnode();
        lc = rc = 0;
        t[u].pos = tp;
        push_up(u);
        return;
    }
    cmptype = type;
    if (tp < t[u].pos) insert(lc, (type + 1) % 3, tp);
    else insert(rc, (type + 1) % 3, tp);
    push_up(u);
    check(u, type);
}

inline bool out(const point &l, const point &r, const point &L, const point &R) {//完全在外
    面
```

```
    return l.x > R.x || l.y > R.y || l.z > R.z || r.x < L.x || r.y < L.y || r.z < L.z;
}

inline bool in(const point &l, const point &r, const point &L, const point &R) {//完全在里面
    return l.x <= L.x && R.x <= r.x && l.y <= L.y && R.y <= r.y && l.z <= L.z && R.z <= r.z
    ;
}

inline int query(int u, point ql, point qr) {//查询最大值
    if (!u || out(ql, qr, t[u].lpos, t[u].rpos)) return 0;
    if (in(ql, qr, t[u].lpos, t[u].rpos)) return t[u].mx;
    int res = 0;
    if (in(ql, qr, t[u].pos, t[u].pos)) res = max(res, t[u].pos.val);
    res = max(res, query(lc, ql, qr));
    res = max(res, query(rc, ql, qr));
    return res;
}

inline void init() {
    root = tot = top = 0;
}

int main() {
    init();
    scanf("%d", &N);
    for (int i = 1; i <= N; i++) scanf("%d%d%d%d", &nod[i].a, &nod[i].b, &nod[i].c, &nod[i
    ].d);
    sort(nod + 1, nod + 1 + N);
    for (int i = 1; i <= N; i++) p[i] = point{nod[i].b, nod[i].c, nod[i].d, 0};
    int ans = 0;
    for (int i = 1; i <= N; i++) {
        ans = max(ans, p[i].val = query(root, point{ -INF, -INF, -INF}, p[i]) + 1);
        insert(root, 0, p[i]);
    }
    printf("%d\n", ans);
    return 0;
}
```

0.8 模拟退火

```
const double DOWN = 0.996;
const double START_T = 5000;
double ansx, ansy, ansz, anse;

void initAns() {
    //初始化一个答案点(可以选任意点)
}

double getEnergy(double x, double y, double z) {
    //具体分析题目
}

void SA() {
```

```cpp
    double T = START_T;
    while (T > 1e-15) {
        double newx = ansx + (rand() * 2 - RAND_MAX) * T;
        double newy = ansy + (rand() * 2 - RAND_MAX) * T;
        double newz = ansz + (rand() * 2 - RAND_MAX) * T;
        double newe = getEnergy(newx, newy, newz);
        double delta = newe - anse;
        if (delta < 0) ansx = newx, ansy = newy, ansz = newz, anse = newe;
        else if (exp(-delta / T) * RAND_MAX > rand())
            ansx = newx, ansy = newy, ansz = newz;
        T *= DOWN;
    }
}

void solve() {
    initAns();
    while ((double) clock() / CLOCKS_PER_SEC < 2.0) SA();
}
```

0.9 莫队

0.9.1 回滚莫队

```cpp
//问题可以莫队（询问可以离线，不带修改）
//区间伸长的时候很好维护信息
//区间缩短的时候不太好维护信息（如最大值，删除以后不知道次大值是多少）

struct Hash {
    int b[N], tot;
    void init() { tot = 0; }
    void insert(int x) { b[++tot] = x; }
    void build() {
        sort(b + 1, b + 1 + tot);
        tot = unique(b + 1, b + 1 + tot) - (b + 1);
    }
    int pos(int x) { return lower_bound(b + 1, b + 1 + tot, x) - b; }
} ha;

int n, m;
int c[N], pos[N], cnt[N], cntt[N];
int belong[N], sizz;
ll ans[N], res;

struct Query {
    int l, r, id;
    bool operator < (const Query &rhs) const {
        return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] : r < rhs.r;
    }
} q[N];

ll bruteForce(int ql, int qr) {
    ll result = 0;
    for (int i = ql; i <= qr; i++) {
```

```
        cntt[pos[i]]++;
        result = max(result, 1ll * c[i] * cntt[pos[i]]);
    }
    for (int i = ql; i <= qr; i++) cntt[pos[i]]--;
    return result;
}

void add(int x) {
    cnt[pos[x]]++;
    res = max(res, 1ll * c[x] * cnt[pos[x]]);
}

void del(int x) {
    cnt[pos[x]]--;
}

int main() {


    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &c[i]), ha.insert(c[i]);
    ha.build();
    for (int i = 1; i <= n; i++) pos[i] = ha.pos(c[i]);

    sizz = sqrt(n); int num = ceil((long double)n / sizz);
    for (int i = 1, j = 1; i <= num; i++)
        while (j <= i * sizz && j <= n)
            belong[j++] = i;

    for (int i = 1; i <= m; i++) scanf("%d%d", &q[i].l, &q[i].r), q[i].id = i;
    sort(q + 1, q + 1 + m);

    for (int i = 1, j = 1; i <= num; i++) {
        memset(cnt, 0, sizeof(cnt));
        int right = min(i * sizz, n);
        res = 0;
        for (int l = right + 1, r = right; j <= m && belong[q[j].l] == i; j++, l = right + 1) {
            int ql = q[j].l, qr = q[j].r;
            if (qr - ql + 1 <= sizz) {
                ans[q[j].id] = bruteForce(ql, qr);
                continue;
            }
            while (r < qr) add(++r);
            ll tmp = res;
            while (l > ql) add(--l);
            ans[q[j].id] = res;
            res = tmp;
            while (l < right + 1) del(l++);
        }
    }

    for (int i = 1; i <= m; i++) printf("%lld\n", ans[i]);

```

```
    return 0;
}
```

0.9.2 带修莫队

```

//带修莫队模板题
//查询[ql, qr]间不同颜色数量，带修改

int n, m;
int c[N], cnt[N];
int belong[N], size, totq, totm;
int ans[N], res;

struct Query {
    int l, r, t, id;
    bool operator < (const Query &rhs) const {
        return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] :
                (belong[r] ^ belong[rhs.r] ? belong[r] < belong[rhs.r] : t < rhs.t);
    }
} q[N];

struct Modify {
    int pos, val;
} modify[N];

void add(int x) {
    if (!cnt[c[x]]) res++;
    cnt[c[x]]++;
}

void del(int x) {
    cnt[c[x]]--;
    if (!cnt[c[x]]) res--;
}

void upd(int x, int ql, int qr) {
    int pos = modify[x].pos;
    if (ql <= pos && pos <= qr) {
        cnt[c[pos]]--; if (!cnt[c[pos]]) res--;
        if (!cnt[modify[x].val]) res++; cnt[modify[x].val]++;
    }
    swap(modify[x].val, c[pos]);//��ϑĵ����´λ�Ļ�ʐ
}

int main() {
#ifdef ACM_LOCAL
    freopen("input.in", "r", stdin);
    freopen("output.out", "w", stdout);
#endif
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &c[i]);
    for (int i = 1; i <= m; i++) {
```

```
        char op[10]; scanf("%s", op);
        if (op[0] == 'Q') {
            int ql, qr; scanf("%d%d", &ql, &qr); totq++;
            q[totq] = Query{ql, qr, totm, totq};
        }
        else {
            int pos, val; scanf("%d%d", &pos, &val); totm++;
            modify[totm] = Modify{pos, val};
        }
    }

    //size = N ^ (2 / 3), (N * totm) ^ (1 / 3)
    size = ceil(pow(n, (long double)2.0 / 3)); int num = ceil((long double)n / size);
    for (int i = 1, j = 1; i <= num; i++)
        while (j <= i * size && j <= n)
            belong[j++] = i;

    sort(q + 1, q + 1 + totq);

    int l = 1, r = 0, t = 0;
    for (int i = 1; i <= totq; i++) {
        int ql = q[i].l, qr = q[i].r, qt = q[i].t;
        while (l < ql) del(l++);
        while (l > ql) add(--l);
        while (r < qr) add(++r);
        while (r > qr) del(r--);
        while (t < qt) upd(++t, ql, qr);
        while (t > qt) upd(t--, ql, qr);
        ans[q[i].id] = res;
    }

    for (int i = 1; i <= totq; i++) printf("%d\n", ans[i]);

    return 0;
}
```

0.9.3 普通莫队

```

int n, m;
int belong[N], size;
int ans[N], res;

struct node {
    int l, r, id;
    bool operator < (const node &rhs) const {//奇偶优化
        return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] :
                ((belong[l] & 1) ? r < rhs.r : r > rhs.r);
    }
} q[N];

void add(int x) {

```

```
}

void del(int x) {

}

int main() {

    size = sqrt(n); int num = ceil((long double)n / size);
    for (int i = 1, j = 1; i <= num; i++)
        while (j <= i * size && j <= n)
            belong[j++] = i;


    int l = 1, r = 0;
    for (int i = 1; i <= m; i++) {
        int ql = q[i].l, qr = q[i].r;
        while (l < ql) del(l++);
        while (l > ql) add(--l);
        while (r < qr) add(++r);
        while (r > qr) del(r--);
        ans[q[i].id] = res;
    }



}
```

0.9.4 树上莫队

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 10;

struct Hash {
    int b[N], tot;
    void init() { tot = 0; }
    void insert(int x) { b[++tot] = x; }
    void build() {
        sort(b + 1, b + 1 + tot);
        tot = unique(b + 1, b + 1 + tot) - (b + 1);
    }
    int pos(int x) { return lower_bound(b + 1, b + 1 + tot, x) - b; }
} ha;

int n, m;
int c[N], cnt[N];
vector<int> g[N];
int st[N], ed[N], dfnt, nodeOf[N << 1], tag[N];
int belong[N], sizz;
int ans[N], res;

```

```
struct Query {
    int l, r, id, k;
    bool operator < (const Query &rhs) const {
        return belong[l] ^ belong[rhs.l] ? belong[l] < belong[rhs.l] : r < rhs.r;
    }
} q[N];

int son[N], siz[N], top[N], fa[N], dep[N];
void dfs(int u, int par) {
    dep[u] = dep[fa[u] = par] + (siz[u] = 1);
    int max_son = -1; nodeOf[st[u] = ++dfnt] = u;
    for (auto &v: g[u])
        if (v != par) {
            dfs(v, u);
            siz[u] += siz[v];
            if (max_son < siz[v])
                son[u] = v, max_son = siz[v];
        }
    nodeOf[ed[u] = ++dfnt] = u;
}
void dfs2(int u, int topf) {
    top[u] = topf;
    if (!son[u]) return;
    dfs2(son[u], topf);
    for (auto &v: g[u])
        if (v != fa[u] && v != son[u]) dfs2(v, v);
}
int lca(int x, int y) {
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        x = fa[top[x]];
    }
    return dep[x] < dep[y] ? x : y;
}

void upd(int x) {
    x = nodeOf[x];
    if (tag[x]) {
        cnt[c[x]]--;
        if (!cnt[c[x]]) res--;
    }
    else {
        if (!cnt[c[x]]) res++;
        cnt[c[x]]++;
    }
    tag[x] ^= 1;
}

int main() {
#ifdef ACM_LOCAL
    freopen("input.in", "r", stdin);
    freopen("output.out", "w", stdout);
#endif
    scanf("%d%d", &n, &m);
```

```cpp
    for (int i = 1; i <= n; i++) scanf("%d", &c[i]), ha.insert(c[i]);
    ha.build();
    for (int i = 1; i <= n; i++) c[i] = ha.pos(c[i]);

    for (int i = 1; i < n; i++) {
        int u, v; scanf("%d%d", &u, &v);
        g[u].push_back(v); g[v].push_back(u);
    }
    int rt = 1; dfs(rt, 0); dfs2(rt, rt);

    sizz = sqrt(dfnt); int num = ceil((long double)dfnt / sizz);
    for (int i = 1, j = 1; i <= num; i++)
        while (j <= i * sizz && j <= dfnt)
            belong[j++] = i;
    for (int i = 1; i <= m; i++) {
        int u, v; scanf("%d%d", &u, &v);
        int tlca = lca(u, v);
        if (st[u] > st[v]) swap(u, v);
        if (u == tlca) q[i] = Query{st[u], st[v], i, 0};
        else q[i] = Query{ed[u], st[v], i, tlca};

    }
    sort(q + 1, q + 1 + m);

    int l = 1, r = 0;
    for (int i = 1; i <= m; i++) {
        int ql = q[i].l, qr = q[i].r;
        while (l < ql) upd(l++);
        while (l > ql) upd(--l);
        while (r < qr) upd(++r);
        while (r > qr) upd(r--);
        ans[q[i].id] = res + (q[i].k ? (cnt[c[q[i].k]] == 0) : 0);
    }

    for (int i = 1; i <= m; i++) printf("%d\n", ans[i]);

    return 0;
}
```

1 动态规划

1.1 四边形优化

```cpp

//四边形优化区间dp(n^3 -> n^2)
//a < b < c < d, f[l][r] = min(f[l][k] + f[k + 1][r] + cost(l, r))
//1. cost(b, c) <= cost(a, d)
//2. cost(a, c) + cost(b, d) <= cost(a, d) + cost(b, c), 即交叉小于包含

void solve() {
    for (int len = 2; len <= n; len++) {
        for (int l = 1, r; l + len - 1 <= n; l++) {
            r = l + len - 1;
```

```
            mn[l][r] = 0x3f3f3f3f;
            for (int k = m[l][r - 1]; k <= m[l + 1][r]; k++)
                if (mn[l][k] + mn[k + 1][r] + cost(l, r) < mn[l][r]) {
                    mn[l][r] = mn[l][k] + mn[k + 1][r] + cost(l, r);
                    m[l][r] = k;
                }
        }
    }
}
```

1.2 数位 DP

```
ll dfs(int pos, ... , bool lead, bool isMax) {//当前位pos, ...为省略条件, Lead判前导零, isMax判前几位数是否选的都是最大值
    if (!pos) return 1;//此处为越过一个数的最后一位(最小的一位), 如123, 越过3这里说明当前已经是123了, 所以只有一个数
    //有时候不一定都是返回1, 看条件
    if (!isMax && !lead && dp[pos][...][...] != -1) return dp[pos][...][...];//记忆化, 如果有直接返回
    int up = isMax ? a[pos] : 9; //如果一直是最大, 当前位最多也就是a[pos], 超过了就大于这个数了
    ll res = 0;
    for (int i = 0; i <= up; i++) {
        //按限制条件来
        //如, 判是否有前导0
        if (lead) res += dfs(pos - 1, ... , !i, isMax && i == a[pos]);
        else res += dfs(pos - 1, ... , 0, isMax && i == a[pos]);
    }
    return isMax || lead ? res : dp[pos][...][...] = res;//如果有前导0或者是前几位都是最大, 直接返回
    //否则dp[pos][...][...]记录值再返回
}

ll calc(ll x) {
    int pos = 0;
    while (x) a[++pos] = x % 10, x /= 10;
    memset(dp, -1, sz(dp));//约束是不确定的  跟数有关
    //约束确定才放外面, 比如不要62, 对于每一个数都是不要62
    return dfs(pos, ... , 1, 1);
}
```

1.3 树上背包

```
//加了剪枝后复杂度为O(NM)

void dfs(int u, int fa) {
    siz[u] = 1;
    for (auto &v: g[u])
        if (v != fa) {
            dfs(v, u);

            int now = min(siz[u] + siz[v] + 1, M);
            int t[MAX_M]; for (int i = 0; i <= M; i++) t[i] = INF/-INF;//初始化
```

```
            for (int i = 0; i <= siz[u]; i++)
                for (int j = 0; j <= siz[v] && i + j <= M; j++) {
                    //...转移方程
                }
            for (int i = 0; i <= now; i++) f[u][i] = min/max(f[u][i], t[i]);
            siz[u] = now;
        }
}
```

1.4 环基树 DP

```
int flag, S, E;//flag是否找到环，SE为环上两个点

void findCircle(int u, int fa) {
    vis[u] = 1;
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != fa) {
            if (vis[v]) flag = 1, S = u, E = v;
            else findCircle(v, u);
        }
}

void dp(int u, int fa) {
    //dp过程

    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != fa && v) {
            dp(v, u);

        }
}

ll calc(int u) {
    flag = 0;
    findCircle(u, 0);
    if (flag) {
        for (int i = head[S], v; i; i = e[i].nxt)
            if ((v = e[i].to) == E) {
                e[i].to = e[i ^ 1].to = 0;//删边操作，注意e[tot]中tot从2开始
                break;
            }
        ll res = 0;
        dp(S, 0); res = max(res, ...);
        dp(E, 0); res = max(res, ...);
        return res;
    }
    else {
        dp(u, 0);
        return ...;
    }
}
```

2 图论

2.1 BellmanFord

```
struct edge {
    int u, v, w;
} e[MAX];

bool BellmanFord(int s) {//判负环
    for (int i = 1; i <= n; i++) dis[i] = (i == s ? 0 : INF);
    for (int i = 1; i < n; i++)
        for (int j = 1; j <= m; j++)
            if (dis[e[j].u] + e[j].w < dis[e[j].v])
                dis[e[j].v] = dis[e[j].u] + e[j].w;
    for (int i = 1; i <= m; i++)
        if (dis[e[i].u] + e[i].w < dis[e[i].v])
            return false;
    return true;
}
```

2.2 Dijkstra

```
//O((N+M)LogM)
struct Node {
    int now, d;
    bool operator < (const Node &rhs) const {
        return d > rhs.d;
    }
};

priority_queue<Node> q;

void dijkstra(int s) {
    for (int i = 1; i <= n; i++) dis[i] = INF, vis[i] = 0;
    dis[s] = 0;
    q.push(Node{s, dis[s]});
    while (!q.empty()) {
        Node p = q.top(); q.pop();
        int u = p.now;
        if (vis[u]) continue;
        vis[u] = 1;
        for (int i = head[u], v; i; i = e[i].nxt)
            if (dis[u] + e[i].w < dis[v = e[i].to]) {
                dis[v] = dis[u] + e[i].w;
                if (!vis[v]) q.push(Node{v, dis[v]});
            }
    }
}
```

2.3 Kruskal

```cpp
int pre[N], m;

struct edge {
    int u, v, w;
    bool operator < (const edge &rhs) const {
        return w < rhs.w;
    }
} e[M * M];

int find(int x) { return x == pre[x] ? x : pre[x] = find(pre[x]); }

int kruskal() {
    int cnt = 0, ans = 0;
    for (int i = 1; i <= m; i++) {
        int u = find(e[i].u), v = find(e[i].v);
        if (u == v) continue;
        ans += e[i].w;
        pre[v] = u;
        cnt++;
        if (cnt == n - 1) break;
    }
    return ans;
}
```

2.4 Kruskal 重构树

```cpp
//用于解决图中两点间多条路中最大/小边权最小/大值问题
//u, v为原图上的点, 则val[lca(u, v)]为u->v间多条路中...
//由于u, v可能在原图中不连通, 所以需要find(u)和find(v)判断一下是不是在一棵树中
//经典问题, 如P4768 [NOI2018]归程, 求v->1路径中前半段路所有路径海拔大于给定h, 后半段路不管海拔高度的最短路
//一条路中最大边权小于等于给定值, 求....

struct Edge {
    int u, v, w;
    bool operator < (const Edge &rhs) const {
        return w > rhs.w;
        //升序为(u, v)间多条路中最大边权最小值
        //降序为(u, v)间多条路中最小边权最大值
    }
} E[M];
vector<int> g[N];
int pre[N], val[N], cnt;//cnt为kruskal重构树的点数, 点数最多为2N - 1
int son[N], siz[N], top[N], fa[N], dep[N];
void dfs(int u, int par) {
    dep[u] = dep[fa[u] = par] + (siz[u] = 1);
    int max_son = -1;
    for (auto &v: g[u])
        if (v != par) {
            dfs(v, u);
            siz[u] += siz[v];
            if (max_son < siz[v])
                son[u] = v, max_son = siz[v];
```

```
        }
}
void dfs2(int u, int topf) {
    top[u] = topf;
    if (!son[u]) return;
    dfs2(son[u], topf);
    for (auto &v: g[u])
        if (v != fa[u] && v != son[u]) dfs2(v, v);
}
int lca(int x, int y) {
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        x = fa[top[x]];
    }
    return dep[x] < dep[y] ? x : y;
}
int find(int x) { return x == pre[x] ? x : pre[x] = find(pre[x]); }
void exKruskal() {
    cnt = N; for (int i = 1; i < (N << 1); i++) pre[i] = i;
    sort(E + 1, E + 1 + M);
    for (int i = 1; i <= M; i++) {
        int u = find(E[i].u), v = find(E[i].v);
        if (u == v) continue;
        val[++cnt] = E[i].w;
        pre[u] = pre[v] = cnt;
        g[u].push_back(cnt), g[cnt].push_back(u);
        g[v].push_back(cnt), g[cnt].push_back(v);
        if (cnt == (N << 1) - 1) break;
    }
    //原图不连通的情况，那形成的就是森林
    for (int i = 1; i <= cnt; i++)
        if (!siz[i]) {//未访问过
            int rt = find(i);//下树剖lca
            dfs(rt, 0); dfs2(rt, rt);
        }
}
```

2.5 Prim

```
struct Node {
    int to, d;
    bool operator < (const Node &rhs) const {
        return d > rhs.d;
    }
};

ll prim(int n) {
    vector<int> vis(n + 1);
    priority_queue<Node> q; q.push({1, 0});
    ll ans = 0, cnt = 0;
    while (!q.empty() && cnt <= n) {
        Node now = q.top(); q.pop();
        int u = now.to, dis = now.d;
```

```
        if (vis[u]) continue;
        vis[u] = 1, cnt++, ans += dis;
        for (int i = head[u], v; i; i = e[i].nxt)
            if (!vis[v = e[i].to]) q.push({v, e[i].w});
    }
    return ans;
}
```

2.6 SPFA

```
bool spfa(int s) {
    for (int i = 1; i <= n; i++) dis[i] = (i == s ? 0 : INF), vis[i] = (i == s), cnt[i] =
    0;
    queue<int> q; q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = 0, cnt[u]++;
        if (cnt[u] >= n) return false;
        for (int i = head[u], v; i; i = e[i].nxt)
            if (dis[u] + e[i].w < dis[v = e[i].to]) {
                dis[v] = dis[u] + e[i].w;
                if (!vis[v]) q.push(v), vis[v] = 1;
            }
    }
    return true;
}

bool spfa(int u) {//dfs
    vis[u] = 1;
    for (int i = head[u]; i; i = e[i].nxt)
        if (dis[u] + e[i].w < dis[v = e[i].to]) {
            if (vis[v]) return false;
            else {
                dis[v] = dis[u] + e[i].w;
                if (!spfa(v)) return false;
            }
        }
    vis[u] = 0;
    return true;
}
```

2.7 严格次小生成树

```
const int N = 1e5 + 10;
const int M = 3e5 + 10;
const int INF = 0x3fffffff;
const ll INF64 = 0x3fffffffffffffffLL;

struct Edge {
    int u, v, w;
    bool operator < (const Edge &rhs) const { return w < rhs.w; }
```

```cpp
};

Edge e[M];
bool used[M];

int n, m;

class Tr {
private:
    struct Edge {
        int to, nxt, val;
    } e[M << 1];
    int tot, head[N];

    int par[N][22], dep[N];
    // 到祖先的路径上边权最大的边
    int maxx[N][22];
    // 到祖先的路径上边权次大的边，若不存在则为 -INF
    int minn[N][22];

public:
    void add(int u, int v, int val) {
        e[++tot] = (Edge){v, head[u], val};
        head[u] = tot;
    }

    void insedge(int u, int v, int val) {
        add(u, v, val);
        add(v, u, val);
    }

    void dfs(int u, int fa) {
        dep[u] = dep[fa] + 1;
        par[u][0] = fa;
        minn[u][0] = -INF;
        for (int i = 1; (1 << i) <= dep[u]; i++) {
            par[u][i] = par[par[u][i - 1]][i - 1];
            int kk[4] = {maxx[u][i - 1], maxx[par[u][i - 1]][i - 1],
                         minn[u][i - 1], minn[par[u][i - 1]][i - 1]};
            // 从四个值中取得最大值
            sort(kk, kk + 4);
            maxx[u][i] = kk[3];
            // 取得严格次大值
            int ptr = 2;
            while (ptr >= 0 && kk[ptr] == kk[3]) ptr--;
            minn[u][i] = (ptr == -1 ? -INF : kk[ptr]);
        }

        for (int i = head[u]; i; i = e[i].nxt)
            if (e[i].to != fa) {
                maxx[e[i].to][0] = e[i].val;
                dfs(e[i].to, u);
            }
    }
```

```

    int lca(int u, int v) {
        if (dep[u] < dep[v]) swap(u, v);
        for (int i = 21; i >= 0; i--)
            if (dep[par[u][i]] >= dep[v]) u = par[u][i];
        if (u == v) return u;
        for (int i = 21; i >= 0; i--)
            if (par[u][i] != par[v][i]) {
                u = par[u][i];
                v = par[v][i];
            }
        return par[u][0];
    }

    int query(int u, int v, int val) {
        int res = -INF;
        for (int i = 21; i >= 0; i--)
            if (dep[par[u][i]] >= dep[v]) {
                if (val != maxx[u][i]) res = max(res, maxx[u][i]);
                else res = max(res, minn[u][i]);
                u = par[u][i];
            }
        return res;
    }
} tr;

int pre[N];
int find(int x) { return pre[x] == x ? x : pre[x] = find(pre[x]); }

ll Kruskal() {
    int tot = 0; ll sum = 0;
    sort(e + 1, e + m + 1);
    for (int i = 1; i <= n; i++) pre[i] = i;

    for (int i = 1; i <= m; i++) {
        int a = find(e[i].u);
        int b = find(e[i].v);
        if (a != b) {
            pre[a] = b;
            tot++;
            tr.insedge(e[i].u, e[i].v, e[i].w);
            sum += e[i].w;
            used[i] = 1;
        }
        if (tot == n - 1) break;
    }
    return sum;
}

int main() {


    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
```

```
        int u, v, w; scanf("%d%d%d", &u, &v, &w);
        e[i] = {u, v, w};
    }

    ll sum = Kruskal();
    ll ans = INF64;
    tr.dfs(1, 0);

    for (int i = 1; i <= m; i++) {
        if (!used[i]) {
            int _lca = tr.lca(e[i].u, e[i].v);
            // 找到路径上不等于 e[i].val 的最大边权
            ll tmpa = tr.query(e[i].u, _lca, e[i].w);
            ll tmpb = tr.query(e[i].v, _lca, e[i].w);
            // 这样的边可能不存在，只在这样的边存在时更新答案
            if (max(tmpa, tmpb) > -INF)
                ans = min(ans, sum - max(tmpa, tmpb) + e[i].w);
        }
    }
    // 次小生成树不存在时输出 -1
    printf("%lld\n", ans == INF64 ? -1 : ans);
    return 0;
}
```

2.8 二分图判定

```
//不存在奇环即为二分图
int n, m;
int pre[N << 1], rk[N << 1];

int find(int x) { while (x ^ pre[x]) x = pre[x]; return x; }

void merge(int x, int y) {
    x = find(x), y = find(y);
    if (x == y) return;
    if (rk[x] < rk[y]) swap(x, y);
    rk[x] += rk[x] == rk[y];
    pre[y] = x;
}

int main() {

    for (int i = 1; i <= n << 1; i++) pre[i] = i, rk[i] = 0;

    int u, v; scanf("%d%d", &u, &v);
    if (find(u) == find(v)) cnt++;//flag = 0
    else {
        merge(u + n, v);
        merge(v + n, u);
    }



```

```
}
```

2.9 匹配问题

2.9.1 GaleShapley

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

const int N=4;

void GaleShapley(const int (&man)[MAX][MAX], const int (&woman)[MAX][MAX], int (&match)[MAX
    ]) {
    int wm[MAX][MAX];    // wm[i][j]: rank from girl i to boy j
    int choose[MAX];    // choose[i]: current boyfriend of girl i
    int manIndex[MAX]; //    manIndex[i]: how many girls that have rejected boy i
    int i, j;
    int w, m;
    for (i = 0; i < N; i++) {
        match[i] = -1;
        choose[i] = -1;
        manIndex[i] = 0;
        for (j = 0; j < N; j++)
            wm[i][woman[i][j]] = j;
    }

    bool bSingle = false;
    while (!bSingle) {
        bSingle = true;
        for (i = 0; i < N; i++) {
            if (match[i] != -1) // boy i already have a girlfriend
                continue;
            bSingle = false;
            j = manIndex[i]++; // the jth girl that boy i like most
            w = man[i][j];
            m = choose[w];    // current girl w's boyfriend
            if (m == -1 || wm[w][i] < wm[w][m]) { // if girl w prefer boy i
                match[i] = w;
                choose[w] = i;
                if (m != -1)
                    match[m] = -1;
            }
        }
    }
}


void Print(const int(&match)[MAX], int N) {
    for (int i = 0; i < N; i++)
        cout << i << " " << match[i] << endl;
}


```

```
int main(){
    int man[N][N]={
        {2,3,1,0},
        {2,1,3,0},
        {0,2,3,1},
        {1,3,2,0},
    };
    int woman[N][N]={
        {0,3,2,1},
        {0,1,2,3},
        {0,2,3,1},
        {1,0,3,2},
    };

    int match[N];
    GaleShapley(man,woman,match);
    Print(match,N);

    return 0;
}
```

2.9.2 Hungry

```
//不带权重的最大匹配，复杂度O(nm)
int used[N], match[N];
vector<int> g[N];

bool find(int u) {
    for (auto &v: g[u])
        if (!used[v]) {
            used[v] = 1;
            if (!match[v] || find(match[v])) {
                match[v] = u;
                return true;
            }
        }
    return false;
}

int hungry() {
    int res = 0;
    for (int i = 1; i <= n; i++) match[i] = 0;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) used[j] = 0;
        if (find(i)) res++;
    }
    return res;
}
```

2.9.3 KM

```
//https://ac.nowcoder.com/acm/contest/view-submission?submissionId=44654655

```

```cpp
struct KM {
#define type int
    //#define inf 0x3f3f3f3f
    static const int N = 505;
    static const int INF = 0x3f3f3f3f;
    int n, mx[N], my[N], prv[N];
    type slk[N], lx[N], ly[N], w[N][N];
    bool vx[N], vy[N];

    void init(int siz) {
        n = siz;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                w[i][j] = -505;
            }
        }
    }

    void add_edge(int x, int y, type val) { w[x][y] = val; }

    void match(int y) { while (y) swap(y, mx[my[y] = prv[y]]); }

    void bfs(int x) {
        int i, y;
        type d;
        for (i = 1; i <= n; i++) {
            vx[i] = vy[i] = 0;
            slk[i] = INF;
        }
        queue<int> q;
        q.push(x);
        vx[x] = 1;
        while (1) {
            while (!q.empty()) {
                x = q.front();
                q.pop();
                for (y = 1; y <= n; y++) {
                    d = lx[x] + ly[y] - w[x][y];
                    if (!vy[y] && d <= slk[y]) {
                        prv[y] = x;
                        if (!d) {
                            if (!my[y]) return match(y);
                            q.push(my[y]);
                            vx[my[y]] = 1;
                            vy[y] = 1;
                        } else slk[y] = d;
                    }
                }
            }
            d = INF + 1;
            for (i = 1; i <= n; i++) {
                if (!vy[i] && slk[i] < d) {
                    d = slk[i];
                    y = i;
```

```
                }
            }
            for (i = 1; i <= n; i++) {
                if (vx[i]) lx[i] -= d;
                if (vy[i]) ly[i] += d;
                else slk[i] -= d;
            }
            if (!my[y]) return match(y);
            q.push(my[y]);
            vx[my[y]] = 1;
            vy[y] = 1;
        }
    }

    type max_match() {
        int i;
        type res;
        for (i = 1; i <= n; i++) {
            mx[i] = my[i] = ly[i] = 0;
            lx[i] = *max_element(w[i] + 1, w[i] + n + 1);
        }
        for (i = 1; i <= n; i++) bfs(i);
        res = 0;
        for (i = 1; i <= n; i++) res += lx[i] + ly[i];
        return res;
    }

#undef type
};
```

2.9.4 一些结论

```
/*
最大匹配数：最大匹配的匹配边的数目

最小点覆盖数：选取最少的点，使任意一条边至少有一个端点被选择

最大独立数：选取最多的点，使任意所选两点均不相连

最小路径覆盖数：对于一个 DAG（有向无环图），选取最少条路径，使得每个顶点属于且仅属于一条路径。路径长可以为 0（即单个点）。

定理1：最大匹配数 = 最小点覆盖数（这是 Konig 定理）

定理2：最大匹配数 = 最大独立数

定理3：最小路径覆盖数 = 顶点数 - 最大匹配数
 */
```

2.10 同余最短路

```
/*
```

```
洛谷P3403: 给定x[0], x[1], x[2], ..., x[n - 1], 对于k <= h, 求有多少个k满足a[0]x[0] + a[1]x[1] + ... + a[n - 1]x[n - 1] = k
洛谷P2662: 最大的不能被x[0], x[1], ..., x[n - 1]表示的数(从小到大), 显然如果gcd(x[i]) = x[0], 无解, 否则跑同余最短路求出max(dis) - x[0]即为答案
解决形如上述类型的题目
 */

struct Node {
    ll now, d;
    bool operator < (const Node &rhs) const {
        return d > rhs.d;
    }
};

priority_queue<Node> q;

ll dis[N];

void dijkstra(int s, int n) {
    vector<int> vis(n);
    for (int i = 0; i < n; i++) dis[i] = INF;
    dis[s] = 0;
    q.push(Node{s, dis[s]});
    while (!q.empty()) {
        Node p = q.top(); q.pop();
        int u = p.now;
        if (vis[u]) continue;
        vis[u] = 1;
        for (int i = head[u], v; i; i = e[i].nxt)
            if (dis[u] + e[i].w < dis[v = e[i].to]) {
                dis[v] = dis[u] + e[i].w;
                if (!vis[v]) q.push(Node{v, dis[v]});
            }
    }
}

ll solve(ll *x, int n, ll h) {
    sort(x, x + n);
    if (x[0] == 1) return h;
    for (int i = 0; i < x[0]; i++)
        for (int j = 1; j < n; j++)
            add(i, (i + x[j]) % x[0], x[j]);
    dijkstra(1, x[0]);
    ll res = 0;
    for (int i = 0; i < x[0]; i++)
        if (dis[i] <= h) res += (h - dis[i] + x[0] - 1) / x[0];
    return res;
}

ll x[N];

int main() {
#ifdef ACM_LOCAL
    freopen("input.in", "r", stdin);
```

```
    freopen("output.out", "w", stdout);
#endif
    ll h, n; scanf("%lld", &h);
    for (int i = 0; i < n; i++) scanf("%lld", &x[i]);
    printf("%lld\n", solve(x, n, h));


    return 0;
}
```

2.11 差分约束

```
/*
差分约束是解决这样一类问题
给出n个形如x[j] - x[i] <= k的式子，求x[n] - x[1]的最大/最小值
最大值—>把所有式子整理为x[j] - x[i] <= k，从i向j连一条边权为k的边，跑最短路
最小值—>把所有式子整理为x[j] - x[i] >= k，从i向j连一条边权为k的边，跑最长路
注意初始化 有时候需要超级源点0
*/

bool spfa(int u) {//dfs跑差分约束最短路
    vis[u] = 1;
    for (int i = head[u], v; i; i = e[i].nxt)
        if (dis[u] + e[i].w < dis[v = e[i].to]) {
            if (vis[v]) return false;
            else {
                dis[v] = dis[u] + e[i].w;
                if (!spfa(v)) return false;
            }
        }
    vis[u] = 0;
    return true;
}
```

2.12 拓扑排序

```
void topo() {
    queue<int> q;
    for (int i = 1; i <= N; i++)
        if (!degree[i]) q.push(i);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        //
        for (auto &v: g[u]) {
            degree[v]--;
            if (!degree[v]) q.push(v);
        }
    }
    //
}
```

2.13 矩阵树

```
/*
计算生成树个数
即求\sum_{Tree} \prod_{e ∈ Tree} num(e)
*/

ll gauss(int n, ll K[][N]) {//求矩阵K的n-1阶顺序主子式
    ll res = 1;
    for (int i = 1; i <= n - 1; i++) {//枚举主对角线上第i个元素
        for (int j = i + 1; j <= n - 1; j++) {//枚举剩下的行
            while (K[j][i]) {//辗转相除
                int t = K[i][i] / K[j][i];
                for (int k = i; k <= n - 1; k++)//转为倒三角
                    K[i][k] = (K[i][k] - t * K[j][k] + mod) % mod;
                swap(K[i], K[j]);//交换i、j两行
                res = -res;//取负
            }
        }
        res = (res * K[i][i]) % mod;
    }
    return (res + mod) % mod;
}

int n, m;
int K[N][N];

int main() {

    for (int i = 1; i <= m; i++) {
        int u, v; ll w; scanf("%d%d%lld", &u, &v, &w);
        K[u][u]++, K[v][v]++, K[u][v]--, K[v][u]--;
    }

    ll ans = gauss(n, K);

    return 0;
}
```

2.14 网络流

2.14.1 Dinic

```
//理论复杂度O(n^2m), 求解二分图匹配问题时, 时间复杂度为O(m sqrt(n))
struct Dinic {
    static const int N = ...;//size
    struct Edge { int from, to, cap, flow; };
    int n, m, s, t;
    vector<Edge> edges;
    vector<int> G[N];
    int dep[N], cur[N];
    bool vis[N];

```

```cpp
    void init(int siz) { n = siz; for (int i = 0; i < siz; i++) G[i].clear(); edges.clear(); }

    void addEdge(int from, int to, int cap) {
        edges.push_back(Edge{from, to, cap, 0});
        edges.push_back(Edge{to, from, 0, 0});
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }

    bool bfs() {
        memset(vis, 0, sizeof(vis));
        queue<int> q; q.push(s); dep[s] = 0, vis[s] = 1;
        while (!q.empty()) {
            int x = q.front();
            q.pop();
            for (auto &v: G[x]) {
                Edge& e = edges[v];
                if (!vis[e.to] && e.cap > e.flow) {
                    dep[e.to] = dep[x] + (vis[e.to] = 1);
                    q.push(e.to);
                }
            }
        }
        return vis[t];
    }

    int dfs(int u, int a) {
        if (u == t || a == 0) return a;
        int flow = 0;
        for (int& i = cur[u], f; i < G[u].size(); i++) {
            Edge& e = edges[G[u][i]];
            if (dep[u] + 1 == dep[e.to] && (f = dfs(e.to, min(a, e.cap - e.flow))) > 0) {
                e.flow += f;
                edges[G[u][i] ^ 1].flow -= f;
                flow += f;
                a -= f;
                if (a == 0) break;
            }
        }
        return flow;
    }

    int maxFlow(int S, int T) {
        s = S, t = T;
        int flow = 0;
        while (bfs()) {
            memset(cur, 0, sizeof(cur));
            flow += dfs(S, INF);
        }
        return flow;
    }
} flow;
```

2.14.2 ISAP

```
struct ISAP {
    const static int N = ...;//node size
    struct Edge {
        int from, to, cap, flow;
        bool operator < (const Edge &rhs) const {
            return from < rhs.from || (from == rhs.from && to < rhs.to);
        }
    };
    int n, m, s, t;
    vector<Edge> edges;
    vector<int> g[N];
    bool vis[N];
    int dep[N], cur[N], p[N], num[N];

    void addEdge(int from, int to, int cap) {
        edges.push_back(Edge{from, to, cap, 0});
        edges.push_back(Edge{to, from, 0, 0});
        m = edges.size();
        g[from].push_back(m - 2);
        g[to].push_back(m - 1);
    }

    bool bfs() {
        memset(vis, 0, sizeof(vis));
        queue<int> q; q.push(t); vis[t] = 1, dep[t] = 0;
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (auto &v: g[u]) {
                Edge &e = edges[v ^ 1];
                if (!vis[e.from] && e.cap > e.flow) {
                    dep[e.from] = dep[u] + (vis[e.from] = 1);
                    q.push(e.from);
                }
            }
        }
        return vis[s];
    }

    void init(int siz) {
        n = siz;
        for (int i = 0; i < siz; i++) g[i].clear();
        edges.clear();
    }

    int augment() {
        int u = t, a = INF;
        while (u != s) {
            Edge &e = edges[p[u]];
            a = min(a, e.cap - e.flow);
            u = edges[p[u]].from;
        }
        u = t;
```

```
        while (u != s) {
            edges[p[u]].flow += a;
            edges[p[u] ^ 1].flow -= a;
            u = edges[p[u]].from;
        }
        return a;
    }

    int maxFlow(int S, int T) {
        s = S, t = T;
        int flow = 0; bfs();
        memset(num, 0, sizeof(num));
        for (int i = 0; i < n; i++) num[dep[i]]++;
        int u = S;
        memset(cur, 0, sizeof(cur));
        while (dep[S] < n) {
            if (u == T) {
                flow += augment();
                u = S;
            }
            int ok = 0;
            for (int i = cur[u]; i < g[u].size(); i++) {
                Edge &e = edges[g[u][i]];
                if (e.cap > e.flow && dep[u] == dep[e.to] + 1) {
                    ok = 1;
                    p[e.to] = g[u][i];
                    cur[u] = i;
                    u = e.to;
                    break;
                }
            }
            if (!ok) {
                int mn = n - 1;
                for (int i = 0; i < g[u].size(); i++) {
                    Edge &e = edges[g[u][i]];
                    if (e.cap > e.flow) mn = min(mn, dep[e.to]);
                }
                if (--num[dep[u]] == 0) break;
                num[dep[u] = mn + 1]++;
                cur[u] = 0;
                if (u != S) u = edges[p[u]].from;
            }
        }
        return flow;
    }

} flow;
```

2.14.3 MCMF

```
//spfa费用流
struct MCMF {
    static const int N = ...;
```

```
    static const int M = ...;
    struct Edge { int nxt, to, cap, cost; } e[M << 1];
    int head[N], tot;
    int n, m;
    int cur[N], dis[N], minCost;
    bool vis[N];

    void init(int siz) { n = siz, tot = 1; for (int i = 0; i < siz; i++) head[i] = 0; }

    void add(int u, int v, int w, int c) { e[++tot] = Edge{head[u], v, w, c}; head[u] = tot; }

    void addEdge(int u, int v, int w, int c) { add(u, v, w, c), add(v, u, 0, -c); }

    bool spfa(int s, int t) {
        for (int i = 0; i < n; i++) dis[i] = INF, cur[i] = head[i];
        queue<int> q; q.push(s), dis[s] = 0, vis[s] = 1;
        while (!q.empty()) {
            int u = q.front();
            q.pop(), vis[u] = 0;
            for (int i = head[u], v; i; i = e[i].nxt) {
                if (e[i].cap && dis[v = e[i].to] > dis[u] + e[i].cost) {
                    dis[v] = dis[u] + e[i].cost;
                    if (!vis[v]) q.push(v), vis[v] = 1;
                }
            }
        }
        return dis[t] != INF;
    }

    int dfs(int u, int t, int flow) {
        if (u == t) return flow;
        vis[u] = 1;
        int ans = 0;
        for (int &i = cur[u], v; i && ans < flow; i = e[i].nxt) {
            if (!vis[v = e[i].to] && e[i].cap && dis[v] == dis[u] + e[i].cost) {
                int x = dfs(v, t, min(e[i].cap, flow - ans));
                if (x) minCost += x * e[i].cost, e[i].cap -= x, e[i ^ 1].cap += x, ans += x;
            }
        }
        vis[u] = 0;
        return ans;
    }

    pair<int, int> mcmf(int s, int t) {
        int ans = 0; minCost = 0;
        while (spfa(s, t)) {
            int x;
            while ((x = dfs(s, t, INF))) ans += x;
        }
        return make_pair(ans, minCost);
    }
} flow;
```

2.15 连通分量

2.15.1 割点

```cpp

int dfn[N], low[N], cnt, tot;
bool cut[N];

void tarjan(int u, int topf) {//无向图割点
    dfn[u] = low[u] = ++cnt;
    int child = 0;
    for (auto &v: g[u]) {
        if (!dfn[v]) {
            tarjan(v, topf);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u] && u != topf) cut[u] = 1;
            if (u == topf) child++;
        }
        low[u] = min(low[u], dfn[v]);
    }
    if (child >= 2 && u == topf) cut[u] = 1;
}
```

2.15.2 桥

```cpp
vector<int> g[N];
int low[N], dfn[N], fa[N], dfnt, cnt_bridge;
bool isbridge[N];//(x, fa[x])为桥

void tarjan(int u, int par) {
    fa[u] = par;
    low[u] = dfn[u] = ++dfnt;
    for (auto &v: g[u]) {
        if (!dfn[v]) {
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] > dfn[u]) {
                isbridge[v] = true;
                cnt_bridge++;
            }
        }
        else if (dfn[v] < dfn[u] && v != par) {
            low[u] = min(low[u], dfn[v]);
        }
    }
}
```

2.15.3 连通分量

```cpp
vector<int> g[N];
vector<pii> edge;
int dfn[N], low[N], vis[N], dfnt;
int color[N], siz[N], col;
```

```
int st[N], top;

void tarjan(int u, int fa) {
    dfn[u] = low[u] = ++dfnt;
    st[++top] = u;
    vis[u] = 1;
    for (auto &v: g[u]) {
        if (v == fa) continue;//有向图去掉
        if (!dfn[v]) {
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
        }
        else if (vis[v]) low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]) {
        color[u] = ++col;
        vis[u] = 0;
        ++siz[col];
        while (st[top] != u) {
            ++siz[col];
            color[st[top]] = col;
            vis[st[top--]] = 0;
        }
        --top;
    }
}

void init() {
    dfnt = top = col = 0;
    for (int i = 1; i <= n; i++) g[i].clear();
}

int main() {


    for (int i = 1; i <= n; i++) if (!dfn[i]) tarjan(i, 0);
    for (int i = 1; i <= n; i++) g[i].clear();
    for (auto &i: edge) {
        int u = color[i.first], v = color[i.second];
        if (u != v) {
            g[u].push_back(v);
            g[v].push_back(u);
        }
    }

}
```

3 字符串

3.1 KMP

```
vector<int> getNext(string s) {
    int n = s.length();
```

```
    vector<int> nxt(n);
    for (int i = 1; i < n; i++) {
        int j = nxt[i - 1];
        while (j > 0 && s[i] != s[j]) j = nxt[j - 1];
        if (s[i] == s[j]) j++;
        nxt[i] = j;
    }
    return nxt;
}

int nxt[MAX];
void getNext(string str) {
    nxt[1] = 0;
    int j = 0, len = str.length();
    for (int i = 2; i <= len; i++) {
        while (j && str[j + 1] != str[i]) j = nxt[j];
        if (str[j + 1] == str[i]) j++;
        nxt[i] = j;
    }
}

int main() {
    int N = strlen(s + 1), M = strlen(t + 1);
    int j = 0;
    for (int i = 1; i <= N; i++) {
        while (j && t[j + 1] != s[i]) j = nxt[j];
        if (t[j + 1] == s[i]) j++;
        if (j == M) {
            ans.push_back(i - M + 1);
            j = nxt[j];
        }
    }
}
```

4 数学

4.1 数论

4.1.1 BSGS

```

struct HashTable {
    static const int MOD = 1e7 + 10;//此处sqrt(p)即可
    struct edge {
        int nxt;
        ll num, val;
    } e[MOD];
    int head[MOD], tot;
    void clear() { tot = 0; memset(head, 0, sizeof(head)); }
    void insert(ll u, ll w) { e[++tot] = edge{head[u % MOD], u, w }, head[u % MOD] = tot; }
    int find(ll u) {
        for (int i = head[u % MOD]; i; i = e[i].nxt)
            if (e[i].num == u) return e[i].val;
```

```
        return -1;
    }
} hs;

ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

ll BSGS(ll a, ll b, ll p) {//a ^ x = b (mod p)
    //令x = i * m - j
    //a ^ {i * m} = b * a ^ j (mod p) ， j ⍰ [0, m - 1]
    b %= p;
    if (a % p == 0 && b) return -1;
    if (b == 1) return 0;
    ll m = sqrt(p) + 1, base = qpow(a, m, p);
    hs.clear();
    //insert t = b * a ^ j to HashTable
    for (ll i = 0, t = b; i < m; i++, t = t * a % p) hs.insert(t, i);
    //t = a ^ {i * m}
    for (ll i = 1, t = base; i <= m + 1; i++, t = t * base % p) {
        ll j = hs.find(t);
        if (j != -1) return i * m - j;
    }
    return -1;
}
```

4.1.2 CRT

```
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    ll res = exgcd(b, a % b, x, y);
    ll t = y;
    y = x - a / b * y;
    x = t;
    return res;
}

ll inv(ll a, ll b) {
    ll x = 0, y = 0;
    exgcd(a, b, x, y);
    return x = (x % b + b) % b;
}

//r[]为余数，m为模数，其中模数互质
```

```
//M = pi(mi), Mi = M / mi, invMi = Mi % mi
//ni满足是除了mi之外的倍数, 且模mi为ri
//利用逆元性质, 即ri * Mi * invMi = ri (mod mi)
//res = (sigma(ri * Mi * invMi)) % M

ll china(ll r[], ll m[], int n) {
    ll M = 1, res = 0;
    for (int i = 1; i <= n; i++) M *= m[i];
    for (int i = 1; i <= n; i++) {
        ll Mi = M / m[i], invMi = inv(Mi, m[i]);
        res = (res + r[i] * Mi % M * invMi % M) % M;
        //res = (res + mul(mul(r[i], Mi, M), invMi, M)) % M;按位乘
    }
    return (res % M + M) % M;
}
```

4.1.3 exBSGS

```
ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    ll res = exgcd(b, a % b, x, y);
    ll t = y;
    y = x - a / b * y;
    x = t;
    return res;
}

ll inv(ll a, ll b) {
    ll x = 0, y = 0;
    exgcd(a, b, x, y);
    return x = (x % b + b) % b;
}

struct HashTable {
    static const int MOD = 1e5 + 10;
    struct edge {
        int nxt;
        ll num, val;
    } e[MOD];
    int head[MOD], tot;
```

```
    void clear() { tot = 0; memset(head, 0, sizeof(head)); }
    void insert(ll u, ll w) { e[++tot] = edge{head[u % MOD], u, w }, head[u % MOD] = tot; }
    int find(ll u) {
        for (int i = head[u % MOD]; i; i = e[i].nxt)
            if (e[i].num == u) return e[i].val;
        return -1;
    }
} hs;

ll BSGS(ll a, ll b, ll p) {//a ^ x = b (mod p)
    if (a % p == 0 && b) return -1;
    if (b == 1) return 0;
    ll m = sqrt(p) + 1, base = qpow(a, m, p);
    hs.clear();
    for (ll i = 0, t = b; i < m; i++, t = t * a % p) hs.insert(t, i);
    for (ll i = 1, t = base; i <= m + 1; i++, t = t * base % p) {
        ll j = hs.find(t);
        if (j != -1) return i * m - j;
    }
    return -1;
}

ll exBSGS(ll a, ll b, ll p) {
    if (b == 1 || p == 1) return 0;//b = 1 || (b = 0 && p = 1)的特殊情况
    //b % gcd(a, p) != 0 && b != 1时方程无解
    //   a ^ x = b (mod p)
    //=> a ^ (x - 1) * (a / G) = (b / G) (mod (p / G))
    //=> a ^ (x - 1) = (b / G * invg) (mod (p / G))
    //=> a ^ x' = b' (mod p')
    ll G = __gcd(a, p), k = 0, g = 1;
    while (G != 1) {
        if (b % G) return -1;
        k++, b /= G, p /= G, g = g * (a / G) % p;
        if (g == b) return k;//即a ^ x' = 1 (mod p')时, 返回k即可
        G = __gcd(a, p);
    }
    ll res = BSGS(a, b * inv(g, p) % p, p);
    return res == -1 ? -1 : res + k;
}
```

4.1.4 exCRT

```
ll mul(ll a, ll b, ll mod) {
    ll res = 0;
    while (b) {
        if (b & 1)
            res = (res + a) % mod;
        a = (a << 1) % mod;
        b >>= 1;
    }
    return res;
}

```

```
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    ll res = exgcd(b, a % b, x, y);
    ll t = y;
    y = x - a / b * y;
    x = t;
    return res;
}

ll excrt(ll r[], ll m[], int n) {
    //模数m[i]不互质时用excrt
    ll M = m[1], res = r[1];
    for (int i = 2; i <= n; i++) {
        ll a = M, b = m[i], c = (r[i] - res % m[i] + m[i]) % m[i], x = 0, y = 0;
        ll g = exgcd(a, b, x, y);
        if (c % g != 0) return -1;//c不能整除g那就无正整数解
        x = mul(x, c / g, b / g);
        res += x * M;
        M *= b / g;
        res = (res % M + M) % M;
    }
    return (res % M + M) % M;
}
```

4.1.5 exEuler

```
ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

ll getPhi(ll x) {
    ll res = 1;
    for (ll i = 2; i * i <= x; i++)
        if (x % i == 0) {
            x = x / i;
            res *= i - 1;
            while (x % i == 0) x = x / i, res *= i;
        }
    if (x > 1) res *= x - 1;
    return res;
}

ll exEuler(char *sa, char *sb, ll p) {//欧拉降幂求a ^ b % p
    //gcd(a, p) = 1        => a ^ b = a ^ (b % phi(p))
```

```
    //gcd(a, p) ≠ 1, b < p => a ^ b = a ^ b
    //gcd(a, p) ≠ 1, b ≥ p => a ^ b = a ^ (b % phi(p) + phi(p))
    int N = strlen(sa), M = strlen(sb), flag = 0;
    ll phi = getPhi(p), a = 0, b = 0;
    for (int i = 0; i < N; i++) a = (a * 10 + sa[i] - '0') % p;
    for (int i = 0; i < M; i++) {
        b = b * 10 + sb[i] - '0';
        if (b >= phi) flag = 1;
        b %= phi;
    }
    if (flag) b += phi;
    return qpow(a, b, p);
}
```

4.1.6 exGCD

```
int exgcd(int a, int b, int &x, int &y) {
    //算gcd的同时，得到ax + by = gcd(a, b)的解(x, y)
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    int res = exgcd(b, a % b, x, y);
    int t = y;
    y = x - a / b * y;
    x = t;
    return res;
}
```

4.1.7 exLucas

```
ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    ll res = exgcd(b, a % b, x, y);
    ll t = y;
    y = x - a / b * y;
    x = t;
    return res;
}
ll inv(ll a, ll b) {
    ll x = 0, y = 0;
```

```
    exgcd(a, b, x, y);
    return x = (x % b + b) % b;
}

ll f(ll n, ll p, ll pk) {
    if (n == 0) return 1;
    ll s = 1;
    for (ll i = 1; i <= pk; i++)
        if (i % p) s = s * i % pk;
    s = qpow(s, n / pk, pk);
    for (ll i = pk * (n / pk); i <= n; i++)
        if (i % p) s = i % pk * s % pk;
    return f(n / p, p, pk) * s % pk;
}

ll g(ll n, ll p) {
    if (n < p) return 0;
    return g(n / p, p) + (n / p);
}

ll c(ll n, ll m, ll p, ll pk) {
    ll frac1 = f(n, p, pk), frac2 = inv(f(m, p, pk) * f(n - m, p, pk) % pk, pk);
    ll s = qpow(p, g(n, p) - g(m, p) - g(n - m, p), pk);
    return frac1 * frac2 % pk * s % pk;
}

ll calc(ll r, ll m, ll M) {
    //CRT, M = p质因数分解后再全部相乘仍然是p
    ll Mi = M / m, invMi = inv(Mi, m);
    return r * Mi % M * invMi % M;
}

ll exLucas(ll n, ll m, ll p) {//O(plogp), 不要求p是质数
    ll t = p, res = 0;
    for (ll i = 2; i * i <= p; i++)
        if (t % i == 0) {//i为当前的质数
            ll pk = 1;//p^k
            while (t % i == 0) {
                t /= i, pk *= i;
            }
            res = (res + calc(c(n, m, i, pk), pk, p)) % p;
        }
    if (t > 1)
        res = (res + calc(c(n, m, t, t), t, p)) % p;
    return res;
}
```

4.1.8 Lucas

```
//求解C(n, m) (mod p), 其中p为素数且较小
//O(p+logp)?O(logn)

ll qpow(ll a, ll b, ll mod) {
```

```
    ll res = 1;
    while (b) {
        if (b & 1)
            res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

ll C(ll n, ll m, ll mod) {
    if (n < m) return 0;
    m = min(m, n - m);
    ll a = 1, b = 1;
    for (int i = 0; i < m; i++)
        a = a * (n - i) % mod, b = b * (i + 1) % mod;
    return a * qpow(b, mod - 2) % mod;
}

ll Lucas(ll n, ll m, ll mod) {
    if (m == 0) return 1;
    return Lucas(n / mod, m / mod, mod) * C(n % mod, m % mod, mod) % mod;
}
```

4.1.9 二次剩余

```
struct Complex {
    ll x, y;
};
ll w;

Complex mul(Complex a, Complex b, ll mod) {//复数乘法
    Complex ans = {0, 0};
    ans.x = ((a.x * b.x % mod + a.y * b.y % mod * w % mod) % mod + mod) % mod;
    ans.y = ((a.x * b.y % mod + a.y * b.x % mod) % mod + mod) % mod;
    return ans;
}

ll binpow_real(ll a, ll b, ll mod) {//实部快速幂
    ll ans = 1;
    while (b) {
        if (b & 1) ans = ans * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return ans % mod;
}

ll binpow_imag(Complex a, ll b, ll mod) {//虚部快速幂
    Complex ans = {1, 0};
    while (b) {
        if (b & 1) ans = mul(ans, a, mod);
        a = mul(a, a, mod);
```

```
        b >>= 1;
    }
    return ans.x % mod;
}

ll cipolla(ll n, ll mod) {//n = 0外面特判
    n %= mod;
    if (mod == 2) return n;
    if (binpow_real(n, (mod - 1) / 2, mod) == mod - 1) return -1;
    ll a;
    while (1) {//生成随机数再检验找到满足非二次剩余的a
        a = rand() % mod;
        w = ((a * a % mod - n) % mod + mod) % mod;
        if (binpow_real(w, (mod - 1) / 2, mod) == mod - 1) break;
    }
    Complex x = {a, 1};
    return binpow_imag(x, (mod + 1) / 2, mod);
}

int main() {

    ll n, mod; scanf("%lld%lld", &n, &mod);
    if (n == 0) {
        printf("0\n");
        continue;
    }
    ll ans1 = cipolla(n, mod);
    if (ans1 == -1) printf("-1\n");
    else {
        ll ans2 = mod - ans1;
        if (ans1 == ans2) printf("%lld\n", ans1);
        else {
            if (ans1 > ans2) swap(ans1, ans2);
            printf("%lld %lld\n", ans1, ans2);
        }
    }

    return 0;
}
```

4.1.10 反演相关

```
/*
莫比乌斯反演
g[n] = \sum_{d | n} f[d]
f[d] = \sum_{d | n} g[d] * mu[n / d]

二项式反演
g[n] = \sum{i = 1}^{n} C(n, i) * f[i]
f[n] = \sum{i = 1}^{n} C(n, i) * g[i] * (-1)^{n - i}

子集反演
f(S) = \sum_{T \belong S} g(T)
```

```
g(S) = \sum_{T \belong S} f(T) * (-1) ^ {|S| - |T|}
*/
```

4.1.11 常见积性函数

```
//phi
//phi[i * j] = phi[i] * phi[j] * gcd(i, j) / phi[gcd(i, j)]
```

4.1.12 整除分块

```

int calc(int n, int m) {
    //\sum_{i = 1} ^ {m} n / i
    //向下取整
    for (int l = 1, r; l <= m; l = r + 1) {
        if (n / l) r = min(m, n / (n / l));
        else r = m;
        //[l, r]之间的 n / l 都相等
    }

    //向上取整
    for (int l = 1, r; l <= m; l = r + 1) {
        int t = (n + l - 1) / l;
        if (t == 1) r = m;
        else r = min(m, (n - 1) / (t - 1));
        //[l, r]之间的 (n + l - 1) / l 都相等
    }

}
```

4.1.13 杜教筛

```



unordered_map<int, ll> smu, sphi;
bool isPrime[MAX];
int prime[MAX], num;
ll mu[MAX], phi[MAX];

void makeMobiusAndEuler(int siz) {
    mu[1] = phi[1] = 1;
    for (int i = 2; i <= siz; i++) {
        if (!isPrime[i]) prime[++num] = i, mu[i] = -1, phi[i] = i - 1;
        for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
            isPrime[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else {
```

```
                phi[i * prime[j]] = phi[prime[j]] * phi[i];
                mu[i * prime[j]] = -mu[i];
            }
        }
    }
    for (int i = 1; i <= siz; i++) mu[i] += mu[i - 1], phi[i] += phi[i - 1];
}

ll getSmu(int n) {
    if (n <= N) return mu[n];
    if (smu[n]) return smu[n];
    ll res = 1;
    for (unsigned int l = 2, r = 0; l <= n; l = r + 1) {
        r = n / (n / l);
        res -= 1ll * (r - l + 1) * getSmu(n / l);
    }
    return smu[n] = res;
}

ll getSphi(int n) {
    if (n <= N) return phi[n];
    if (sphi[n]) return sphi[n];
    ll res = 1ll * n * (n + 1) / 2;
    for (unsigned int l = 2, r = 0; l <= n; l = r + 1) {
        r = n / (n / l);
        res -= 1ll * (r - l + 1) * getSphi(n / l);
    }
    return sphi[n] = res;
}
```

4.1.14 筛 mobius

```
int vis[N], prime[N], num, mu[N];
void makeMobius(int siz) {
    mu[1] = 1, num = 0;
    for (int i = 2; i <= siz; i++) {
        if (!vis[i]) prime[++num] = i, mu[i] = -1;
        for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
            vis[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                break;
            }
            else mu[i * prime[j]] = mu[i] * mu[prime[j]];
        }
    }
}
```

4.1.15 筛 phi

```
int vis[N], prime[N], num, phi[N];
void makePhi(int siz) {
    phi[1] = 1, num = 0;
```

```
    for (int i = 2; i <= siz; i++) {
        if (!vis[i]) prime[++num] = i, phi[i] = i - 1;
        for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
            vis[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else phi[i * prime[j]] = phi[i] * phi[prime[j]];
        }
    }
}
```

4.1.16 筛积性函数

```
//只需要计算f(p ^ k)即可
//其余的都可以通过积性函数的性质来计算

int vis[N], prime[N], num;
int f[N], low[N];

void makeF(int siz) {//f为积性函数
    num = 0, low[1] = f[1] = 1;
    for (int i = 2; i <= siz; i++) {
        if (!vis[i]) prime[++num] = i, low[i] = i, f[i] = ...;//这里是f(p)的答案
        for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
            vis[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                low[i * prime[j]] = low[i] * prime[j];
                if(low[i] == i) {//i = prime[j] ^ k
                    //只需要这里算一下
                    //考虑 p ^ 1 , p ^ 2, p ^ 3...
                }
                else f[i * prime[j]] = 1ll * f[i / low[i]] * f[prime[j] * low[i]] % mod;
                break;
            }
            low[i * prime[j]] = prime[j];
            f[i * prime[j]] = 1ll * f[i] * f[prime[j]] % mod;
        }
    }

}
```

4.1.17 线性同余方程

```
int exgcd(int a, int b, int &x, int &y) {
    //算gcd的同时, 得到ax + by = gcd(a, b)的解(x, y)
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    int res = exgcd(b, a % b, x, y);
    int t = y;
```

```
    y = x - a / b * y;
    x = t;
    return res;
}

bool solve(int a, int b, int c, int &x, int &y) {
    //求 ax + by = c 的解(x, y), 有解的条件为c | (gcd(a, b))!!!!!!
    //等价于求 ax = b (mod c) 的整数解x
    int d = exgcd(a, b, x, y);
    if (c % d != 0) return false;
    int k = c / d;
    x *= k, y *= k;
    /*求ax = b (mod c)的最小正整数解
    int t = b / __gcd(a, b);
    x = (x % t + t) % t;*/
    return true;
}
```

4.1.18 质因子分解

```
int vis[N], prime[N], num;
void makePrime(int siz) {
    num = 0;
    for (int i = 2; i <= siz; i++) {
        if (!vis[i]) prime[++num] = i;
        for (int j = 1; j <= num && i * prime[j] <= siz; j++) {
            vis[i * prime[j]] = 1;
            if (i % prime[j] == 0) break;
        }
    }
}

void divide(ll x) {
    for (int i = 1; i <= num && 1ll * prime[i] * prime[i] <= x; i++)
        if (x % prime[i] == 0) {
            int cnt = 0;
            while (x % prime[i] == 0) x /= prime[i], cnt++;
            store.push_back({prime[i], cnt});
        }
    if (x > 1) store.push_back({x, 1});
}
```

4.1.19 逆元

```
//存在逆元的充要条件为gcd(a, mod) = 1
//小心那种mod = 2^k, 然后求inv(2^i), 这种只能迭代求(可能为0)
//mod为质数, 费马小定理求解
#define inv(x,y) qpow(x,y-2,y)

ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1)
```

```
            res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

//mod不是质数, exgcd求解
//exgcd(a, b x, y) -> b=mod, x为逆元
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    ll res = exgcd(b, a % b, x, y);
    ll t = y;
    y = x - a / b * y;
    x = t;
    return res;
}

ll inv(ll a, ll b = mod) {
    ll x = 0, y = 0;
    exgcd(a, b, x, y);
    return x = (x % b + b) % b;
}
```

4.1.20 龟速乘

```
//在a * a > ll, a * a % mod < ll下使用
ll mul(ll a, ll b, ll mod) {
    ll res = 0;
    while (b) {
        if (b & 1)
            res = (res + a) % mod;
        a = (a << 1) % mod;
        b >>= 1;
    }
    return res;
}

ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1)
            res = mul(res, a, mod);
        a = mul(a, a, mod);
        b >>= 1;
    }
    return res;
}
```

4.2 线性代数

4.2.1 多项式

FFT

```cpp
//F(x)=FL(x^2)+x*FR(x^2)
//F(W^k)=FL(w^k)+W^k*FR(w^k)
//F(W^{k+n/2})=FL(w^k)-W^k*FR(w^k)


const int N = 4e5 + 10;//4倍空间
const double PI = acos(-1);

struct Complex {
    double a, b;
    Complex(double a = 0, double b = 0): a(a), b(b) {}
    Complex operator * (const Complex &rhs) { return Complex(a * rhs.a - b * rhs.b, a * rhs
    .b + b * rhs.a); }
    Complex operator + (const Complex &rhs) { return Complex(a + rhs.a, b + rhs.b); }
    Complex operator - (const Complex &rhs) { return Complex(a - rhs.a, b - rhs.b); }
};

int tr[N];

void FFT(Complex *A, int len, int type) {
    for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
    for (int i = 2; i <= len; i <<= 1) {//区间长度
        int mid = i / 2;
        Complex Wn(cos(2 * PI / i), type * sin(2 * PI / i));//单位根
        for (int k = 0; k < len; k += i) {//每个子问题的起始点
            Complex w(1, 0);//omega
            for (int l = k; l < k + mid; l++) {
                Complex t = w * A[l + mid];
                A[l + mid] = A[l] - t;
                A[l] = A[l] + t;
                w = w * Wn;
            }
        }
    }
}

void mul(Complex *a, Complex *b, int n) {
    int len = 1; while (len <= n) len <<= 1;
    for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
    FFT(a, len, 1), FFT(b, len, 1);
    for (int i = 0; i < len; i++) a[i] = a[i] * b[i];
    FFT(a, len, -1);
    for (int i = 0; i < len; i++) a[i].a /= len;
}

Complex a[N], b[N];

int main() {

```

```
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 0; i <= n; i++) scanf("%lf", &a[i].a);
    for (int i = 0; i <= m; i++) scanf("%lf", &b[i].a);

    mul(a, b, n + m);
    for (int i = 0; i <= n + m; i++)
        printf("%d ", (int)(a[i].a + 0.5));


    return 0;
}
```

FWT

```
const int mod = ...;
const int inv2 = ...;

int n;
int a[MAX], b[MAX];

inline void get() {
    for (int i = 0; i < n; i++) a[i] = 1ll * a[i] * b[i] % mod;
}

inline void OR(int *f, int x = 1) {
    for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
        for (int i = 0; i < n; i += o)
            for (int j = 0; j < k; j++)
                f[i + j + k] = (f[i + j + k] + 1ll * f[i + j] * x % mod + mod) % mod;
}

inline void AND(int *f, int x = 1) {
    for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
        for (int i = 0; i < n; i += o)
            for (int j = 0; j < k; j++)
                f[i + j] = (f[i + j] + 1ll * f[i + j + k] * x % mod + mod) % mod;
}

inline void XOR(int *f, int x = 1) {
    for (int o = 2, k = 1; o <= n; o <<= 1, k <<= 1)
        for (int i = 0; i < n; i += o)
            for (int j = 0; j < k; j++) {
                ll a0 = f[i + j], a1 = f[i + j + k];
                f[i + j] = (a0 + a1) % mod * x % mod;
                f[i + j + k] = (a0 - a1 + mod) % mod * x % mod;
            }
}

int main() {

    FWT(a, 1); FWT(b, 1); get(); FWT(a, (OR + AND: -1, XOR: inv2));

}
```

NTT

```
//mod比最终答案要大
//如果中间乘法计算会爆ll，换成按位乘
//精度高，比3次FFT要快(复数运算=>整数运算)
//4倍空间

ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

const ll mod = 998244353;
const ll G = 3;
const ll invG = qpow(G, mod - 2, mod);
int tr[N];

void NTT(ll *A, int len, int type) {
    for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
    for (int i = 2; i <= len; i <<= 1) {
        int mid = i / 2;
        ll Wn = qpow(type == 1 ? G : invG, (mod - 1) / i, mod);
        for (int k = 0; k < len; k += i) {
            ll w = 1;
            for (int l = k; l < k + mid; l++) {
                ll t = w * A[l + mid] % mod;
                A[l + mid] = (A[l] - t + mod) % mod;
                A[l] = (A[l] + t) % mod;
                w = w * Wn % mod;
            }
        }
    }
    if (type == -1) {
        ll invn = qpow(len, mod - 2, mod);
        for (int i = 0; i < len; i++)
            A[i] = A[i] * invn % mod;
    }
}

void mul(ll *a, ll *b, int n) {
    int len = 1; while (len <= n) len <<= 1;
    for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
    NTT(a, len, 1), NTT(b, len, 1);
    for (int i = 0; i < len; i++) a[i] = a[i] * b[i] % mod;
    NTT(a, len, -1);
}

int n, m;
ll a[N], b[N];

```

```
int main() {

    scanf("%d%d", &n, &m);
    for (int i = 0; i <= n; i++) scanf("%lld", &a[i]);
    for (int i = 0; i <= m; i++) scanf("%lld", &b[i]);

    mul(a, b, n + m);
    for (int i = 0; i <= n + m; i++)
        printf("%lld ", a[i]);

    return 0;
}
```

任意模数 **MTT**

```
//将多项式拆成(a1 * mod + a2) * (b1 * mod + b2)的形式
//=>a1 * b1 * mod ^ 2 + (a2 * b1 + a1 * b2) * mod + a2 * b2
//在利用DFT合并、IDFT合并，最终只需要4次DFT即可
//精度10^14
//4倍空间


const double PI = acos(-1);

struct Complex {
    double x, y;
    Complex(double a = 0, double b = 0): x(a), y(b) {}
    Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
    Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
    Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x * rhs
    .y + y * rhs.x); }
    Complex conj() { return Complex(x, -y); }
} w[N];

ll mod;
int tr[N];
int a[N], b[N], ans[N];

void FFT(Complex *A, int len) {
    for (int i = 0; i < len; i++) if(i < tr[i]) swap(A[i], A[tr[i]]);
    for (int i = 2, lyc = len >> 1; i <= len; i <<= 1, lyc >>= 1)
        for (int j = 0; j < len; j += i) {
            Complex *l = A + j, *r = A + j + (i >> 1), *p = w;
            for (int k = 0; k < i >> 1; k++) {
                Complex tmp = *r * *p;
                *r = *l - tmp, *l = *l + tmp;
                ++l, ++r, p += lyc;
            }
        }
}

inline void MTT(int *x, int *y, int *z, int n) {
    int len = 1; while (len <= n) len <<= 1;
    for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
```

```
    for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin(2 * PI *
     i / len));

    for (int i = 0; i < len; i++) (x[i] += mod) %= mod, (y[i] += mod) %= mod;
    static Complex a[N], b[N];
    static Complex dfta[N], dftb[N], dftc[N], dftd[N];

    for (int i = 0; i < len; i++) a[i] = Complex(x[i] & 32767, x[i] >> 15);
    for (int i = 0; i < len; i++) b[i] = Complex(y[i] & 32767, y[i] >> 15);
    FFT(a, len), FFT(b, len);
    for (int i = 0; i < len; i++) {
        int j = (len - i) & (len - 1);
        static Complex da, db, dc, dd;
        da = (a[i] + a[j].conj()) * Complex(0.5, 0);
        db = (a[i] - a[j].conj()) * Complex(0, -0.5);
        dc = (b[i] + b[j].conj()) * Complex(0.5, 0);
        dd = (b[i] - b[j].conj()) * Complex(0, -0.5);
        dfta[j] = da * dc;
        dftb[j] = da * dd;
        dftc[j] = db * dc;
        dftd[j] = db * dd;
    }
    for (int i = 0; i < len; i++) a[i] = dfta[i] + dftb[i] * Complex(0, 1);
    for (int i = 0; i < len; i++) b[i] = dftc[i] + dftd[i] * Complex(0, 1);
    FFT(a, len), FFT(b, len);
    for (int i = 0; i < len; i++) {
        int da = (ll)(a[i].x / len + 0.5) % mod;
        int db = (ll)(a[i].y / len + 0.5) % mod;
        int dc = (ll)(b[i].x / len + 0.5) % mod;
        int dd = (ll)(b[i].y / len + 0.5) % mod;
        z[i] = (da + ((ll)(db + dc) << 15) + ((ll)dd << 30)) % mod;
    }
}



int main() {

    int n, m;
    scanf("%d%d%lld", &n, &m, &mod);
    for (int i = 0; i <= n; i++) scanf("%d", &a[i]);
    for (int i = 0; i <= m; i++) scanf("%d", &b[i]);

    MTT(a, b, ans, n + m);
    for (int i = 0; i <= n + m; i++)
        printf("%s%d", i == 0 ? "" : " ", (ans[i] + mod) % mod);

    return 0;
}
```

任意模数 NTT

```
//要求选取的三个模数mod1 * mod2 * mod3 >= p^2*n
//优点是精度高，可达10^26
```

```
//缺点是常数大(9次NTT)，并且还使用了龟速乘
//4倍空间

ll qmul(ll a, ll b, ll mod) {
    ll res = 0;
    while (b) {
        if (b & 1)
            res = (res + a) % mod;
        a = (a << 1) % mod;
        b >>= 1;
    }
    return res;
}

ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = qmul(res, a, mod);
        a = qmul(a, a, mod);
        b >>= 1;
    }
    return res;
}

const ll mod1 = 998244353, mod2 = 1004535809, mod3 = 469762049, mod4 = mod1 * mod2;
const ll G = 3;
ll a[3][MAX], b[3][MAX], ans[MAX], p;
int tr[MAX];

void NTT(ll *A, int len, int type, ll mod) {
    for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
    for (int i = 2; i <= len; i <<= 1) {
        int mid = i / 2;
        ll Wn = qpow(type == 1 ? G : qpow(G, mod - 2, mod), (mod - 1) / i, mod);
        for (int k = 0; k < len; k += i) {
            ll w = 1;
            for (int l = k; l < k + mid; l++) {
                ll t = w * A[l + mid] % mod;
                A[l + mid] = (A[l] - t + mod) % mod;
                A[l] = (A[l] + t) % mod;
                w = w * Wn % mod;
            }
        }
    }
    if (type != 1) {
        ll invn = qpow(len, mod - 2, mod);
        for (int i = 0; i < len; i++) A[i] = A[i] * invn % mod;
    }
}

void mul(int i, int len, ll mod) {
    NTT(a[i], len, 1, mod), NTT(b[i], len, 1, mod);
    for (int j = 0; j < len; j++) a[i][j] = a[i][j] * b[i][j] % mod;
    NTT(a[i], len, -1, mod);
```

```
}

void CRT(int len) {
    ll inv1 = qpow(mod2, mod1 - 2, mod1);
    ll inv2 = qpow(mod1, mod2 - 2, mod2);
    ll inv3 = qpow(mod4 % mod3, mod3 - 2, mod3);
    for (int i = 0; i < len; i++) {
        ll t = 0;
        t = (t + qmul(a[0][i] * mod2 % mod4, inv1, mod4)) % mod4;
        t = (t + qmul(a[1][i] * mod1 % mod4, inv2, mod4)) % mod4;
        a[1][i] = t;
        t = (a[2][i] - a[1][i] % mod3 + mod3) % mod3 * inv3 % mod3;
        ans[i] = (mod4 % p * t % p + a[1][i] % p) % p;
    }
}

void doNTT(int n) {
    int len = 1; while (len <= n) len <<= 1;
    for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
    mul(0, len, mod1), mul(1, len, mod2), mul(2, len, mod3);
    CRT(len);
}

int main() {

    int n, m;
    scanf("%d%d%lld", &n, &m, &p);
    for (int i = 0; i <= n; i++) {
        ll x; scanf("%lld", &x);
        a[0][i] = a[1][i] = a[2][i] = x % p;
    }
    for (int i = 0; i <= m; i++) {
        ll x; scanf("%lld", &x);
        b[0][i] = b[1][i] = b[2][i] = x % p;
    }
    doNTT(n + m);
    for (int i = 0; i <= n + m; i++) printf("%lld ", ans[i]);

    return 0;
}}
```

任意模数多项式

```
const double PI = acos(-1);

ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}
```

```

struct Complex {
    double x, y;
    Complex(double a = 0, double b = 0): x(a), y(b) {}
    Complex operator + (const Complex &rhs) { return Complex(x + rhs.x, y + rhs.y); }
    Complex operator - (const Complex &rhs) { return Complex(x - rhs.x, y - rhs.y); }
    Complex operator * (const Complex &rhs) { return Complex(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x); }
    Complex conj() { return Complex(x, -y); }
} w[N];

ll mod;
int tr[N];

int getLen(int n) {
    int len = 1; while (len < (n << 1)) len <<= 1;
    for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
    for (int i = 0; i < len; i++) w[i] = w[i] = Complex(cos(2 * PI * i / len), sin(2 * PI * i / len));
    return len;
}

void FFT(Complex *A, int len) {
    for (int i = 0; i < len; i++) if(i < tr[i]) swap(A[i], A[tr[i]]);
    for (int i = 2, lyc = len >> 1; i <= len; i <<= 1, lyc >>= 1)
        for (int j = 0; j < len; j += i) {
            Complex *l = A + j, *r = A + j + (i >> 1), *p = w;
            for (int k = 0; k < i >> 1; k++) {
                Complex tmp = *r * *p;
                *r = *l - tmp, *l = *l + tmp;
                ++l, ++r, p += lyc;
            }
        }
}

inline void MTT(ll *x, ll *y, ll *z, int len) {

    for (int i = 0; i < len; i++) (x[i] += mod) %= mod, (y[i] += mod) %= mod;
    static Complex a[N], b[N];
    static Complex dfta[N], dftb[N], dftc[N], dftd[N];

    for (int i = 0; i < len; i++) a[i] = Complex(x[i] & 32767, x[i] >> 15);
    for (int i = 0; i < len; i++) b[i] = Complex(y[i] & 32767, y[i] >> 15);
    FFT(a, len), FFT(b, len);
    for (int i = 0; i < len; i++) {
        int j = (len - i) & (len - 1);
        static Complex da, db, dc, dd;
        da = (a[i] + a[j].conj()) * Complex(0.5, 0);
        db = (a[i] - a[j].conj()) * Complex(0, -0.5);
        dc = (b[i] + b[j].conj()) * Complex(0.5, 0);
        dd = (b[i] - b[j].conj()) * Complex(0, -0.5);
        dfta[j] = da * dc;
        dftb[j] = da * dd;
        dftc[j] = db * dc;
```

```
        dftd[j] = db * dd;
    }
    for (int i = 0; i < len; i++) a[i] = dfta[i] + dftb[i] * Complex(0, 1);
    for (int i = 0; i < len; i++) b[i] = dftc[i] + dftd[i] * Complex(0, 1);
    FFT(a, len), FFT(b, len);
    for (int i = 0; i < len; i++) {
        ll da = (ll)(a[i].x / len + 0.5) % mod;
        ll db = (ll)(a[i].y / len + 0.5) % mod;
        ll dc = (ll)(b[i].x / len + 0.5) % mod;
        ll dd = (ll)(b[i].y / len + 0.5) % mod;
        z[i] = (da + ((ll)(db + dc) << 15) + ((ll)dd << 30)) % mod;
    }
}

void getInv(ll *f, ll *g, int n) {
    if (n == 1) { g[0] = qpow(f[0], mod - 2, mod); return; }
    getInv(f, g, (n + 1) >> 1);
    int len = getLen(n);
    static ll c[N];
    for (int i = 0; i < len; ++i) c[i] = i < n ? f[i] : 0;
    MTT(c, g, c, len), MTT(c, g, c, len);
    for (int i = 0; i < n; i++) g[i] = (2ll * g[i] - c[i] + mod) % mod;
    for (int i = n; i < len; ++i) g[i] = 0;
    for (int i = 0; i < len; i++) c[i] = 0;
}

void getDer(ll *f, ll *g, int len) { for (int i = 1; i < len; i++) g[i - 1] = f[i] * i % mod; g[len - 1] = 0; }

void getInt(ll *f, ll *g, int len) { for (int i = 1; i < len; i++) g[i] = f[i - 1] * qpow(i, mod - 2, mod) % mod; g[0] = 0; }

void getLn(ll *f, ll *g, int n) {
    static ll a[N], b[N];
    getDer(f, a, n);
    getInv(f, b, n);
    int len = getLen(n);
    MTT(a, b, a, len);
    getInt(a, g, len);
    for (int i = n; i < len; i++) g[i] = 0;
    for (int i = 0; i < len; i++) a[i] = b[i] = 0;
}


void getExp(ll *f, ll *g, int n) {
    if (n == 1) return (void) (g[0] = 1);
    getExp(f, g, (n + 1) >> 1);
    static ll a[N];
    getLn(g, a, n);
    a[0] = (f[0] + 1 - a[0] + mod) % mod;
    for (int i = 1; i < n; i++) a[i] = (f[i] - a[i] + mod) % mod;
    int len = getLen(n);
    MTT(a, g, g, len);
    for (int i = n; i < len; i++) g[i] = 0;
```

```
    for (int i = 0; i < len; i++) a[i] = 0;
}

void getPow(ll *f, ll *g, int n, ll k) {
    static ll a[N];
    getLn(f, a, n);
    for (int i = 0; i < n; i++) a[i] = a[i] * k % mod;
    getExp(a, g, n);
    for (int i = 0, len = getLen(n); i < len; i++) a[i] = 0;
}

void fenzhiFFT(ll *f, ll *g, int n) {
    //计算g[i] = \sum_{j = 1}^{i} g[i - j] * f[j]
    static ll a[N];
    for (int i = 1; i < n; i++) a[i] = (mod - f[i]) % mod;
    a[0] = 1;
    ll g0 = g[0];
    getInv(a, g, n);
    for (int i = 0; i < n; i++) g[i] = g[i] * g0 % mod, a[i] = 0;
}
```

其他

两次 FFT

```
//原理
//记P(x) = F(x) + G(x)i
//P(x)^2 = F(x)^2 - G(x)^2 + 2F(x)G(x)i
//故虚部/2就可得到F(x)G(x)

//快但是会丢失精度
//比如说F(x)的系数均在[1e6, 1e5]之间, G(x)的系数均在[1e5, 1e6]之间
//直接做FFT ,涉及的精度跨度上限是1e12
//假如使用三次变两次优化,由于平方项的存在,涉及的精度跨度上限是1e24, 严重掉精度

const int MAX = 4e6 + 10;
const double PI = acos(-1);

struct Complex {
    double a, b;
    Complex(double a = 0, double b = 0): a(a), b(b) {}
    Complex operator * (const Complex &rhs) { return Complex(a * rhs.a - b * rhs.b, a * rhs.b + b * rhs.a); }
    Complex operator + (const Complex &rhs) { return Complex(a + rhs.a, b + rhs.b); }
    Complex operator - (const Complex &rhs) { return Complex(a - rhs.a, b - rhs.b); }
};

int N, M;
int n, tr[MAX];
Complex a[MAX];

void FFT(Complex *A, int type) {
    for (int i = 0; i < n; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
    for (int len = 2; len <= n; len <<= 1) {//区间长度
```

```
        int mid = len / 2;
        Complex Wn(cos(2 * PI / len), type * sin(2 * PI / len));//单位根
        for (int k = 0; k < n; k += len) {//每个子问题的起始点
            Complex w(1, 0);//omega
            for (int l = k; l < k + mid; l++) {
                Complex t = w * A[l + mid];
                A[l + mid] = A[l] - t;
                A[l] = A[l] + t;
                w = w * Wn;
            }
        }
    }

}

int main() {
    scanf("%d%d", &N, &M);
    for (int i = 0; i <= N; i++) scanf("%lf", &a[i].a);
    for (int i = 0; i <= M; i++) scanf("%lf", &a[i].b);//虚部

    n = 1; while (n <= N + M) n <<= 1;
    for (int i = 0; i < n; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? n >> 1 : 0);

    FFT(a, 1);
    for (int i = 0; i < n; i++) a[i] = a[i] * a[i];

    FFT(a, -1);

    for (int i = 0; i <= N + M; i++)
        printf("%d ", (int)(a[i].b / n / 2 + 0.5));


    return 0;
}
```

递归无优化 FFT

```
struct Complex {
    double a, b;
    Complex(double a = 0, double b = 0): a(a), b(b) {}
    Complex operator * (const Complex &rhs) { return Complex(a * rhs.a - b * rhs.b, a * rhs
    .b + b * rhs.a); }
    Complex operator + (const Complex &rhs) { return Complex(a + rhs.a, b + rhs.b); }
    Complex operator - (const Complex &rhs) { return Complex(a - rhs.a, b - rhs.b); }
} store[MAX << 1];

//递归版
void FFT(Complex *A, int len, int type) {//type(1, -1) = (DFT, IDFT)
    if (len == 1) return;
    Complex *fl = A, *fr = A + len / 2;
    for (int k = 0; k < len; k++) store[k] = A[k];
    for (int k = 0; k < len / 2; k++) fl[k] = store[k << 1], fr[k] = store[k << 1 | 1];
    FFT(fl, len / 2, type), FFT(fr, len / 2, type);
    Complex Wn = Complex(cos(2 * PI / len), type * sin(2 * PI / len));
```

```
    Complex w = Complex(1, 0);
    //Wn -> 单位根, w * I -> 下一个单位根
    for (int k = 0; k < len / 2; k++) {
        store[k] = fl[k] + w * fr[k];
        store[k + len / 2] = fl[k] - w * fr[k];
        w = w * Wn;
    }
    for (int k = 0; k < len; k++) A[k] = store[k];
}
```

原根表

```
mod                                         原根
r*2^k+1   r               k             g
3   1   1   2
5   1   2   2
17  1   4   3
97  3   5   5
193 3   6   5
257 1   8   3
7681    15  9   17
12289   3   12  11
40961   5   13  3
65537   1   16  3
786433  3   18  10
5767169 11  19  3
7340033 7   20  3
23068673    11  21  3
104857601   25  22  3
167772161   5   25  3
469762049   7   26  3
998244353   119 23  3   这个数常用
1004535809  479 21  3   加起来不会爆int
2013265921  15  27  31
2281701377  17  27  3    这个数平方刚好不会爆ll
3221225473  3   30  5
75161927681 35  31  3
77309411329 9   33  7
206158430209    3   36  22
2061584302081   15  37  7
2748779069441   5   39  3
6597069766657   3   41  5
39582418599937  9   42  5
79164837199873  9   43  5
263882790666241 15  44  7
1231453023109121    35  45  3
1337006139375617    19  46  3
3799912185593857    27  47  5
4222124650659841    15  48  19
7881299347898369    7   50  6
31525197391593473   7   52  3
180143985094819841  5   55  6
1945555039024054273 27  56  5
4179340454199820289 29  57  3
```

多项式

```cpp

typedef long long ll;
const int N = 1e6 + 10;

ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

const ll mod = 998244353;
const ll G = 3;
const ll invG = qpow(G, mod - 2, mod);
const ll inv2 = qpow(2ll, mod - 2, mod);
int tr[N];

int getLen(int n) {
    int len = 1;
    while (len < (n << 1)) len <<= 1;
    for (int i = 0; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1 ? len >> 1 : 0);
    return len;
}

void NTT(ll *A, int len, int type) {
    for (int i = 0; i < len; i++) if (i < tr[i]) swap(A[i], A[tr[i]]);
    for (int i = 2; i <= len; i <<= 1) {
        int mid = i / 2;
        ll Wn = qpow(type == 1 ? G : invG, (mod - 1) / i, mod);
        for (int k = 0; k < len; k += i) {
            ll w = 1;
            for (int l = k; l < k + mid; l++) {
                ll t = w * A[l + mid] % mod;
                A[l + mid] = (A[l] - t + mod) % mod;
                A[l] = (A[l] + t) % mod;
                w = w * Wn % mod;
            }
        }
    }
    if (type == -1) {
        ll invn = qpow(len, mod - 2, mod);
        for (int i = 0; i < len; i++)
            A[i] = A[i] * invn % mod;
    }
}

void getDer(ll *f, ll *g, int len) { for (int i = 1; i < len; i++) g[i - 1] = f[i] * i % mod; g[len - 1] = 0; }

```

```cpp
void getInt(ll *f, ll *g, int len) { for (int i = 1; i < len; i++) g[i] = f[i - 1] * qpow(i, mod - 2, mod) % mod; g[0] = 0; }

void getInv(ll *f, ll *g, int n) {
    if (n == 1) return (void) (g[0] = qpow(f[0], mod - 2, mod));
    getInv(f, g, (n + 1) >> 1);
    int len = getLen(n);
    static ll a[N];
    for (int i = 0; i < len; ++i) a[i] = i < n ? f[i] : 0;
    NTT(a, len, 1), NTT(g, len, 1);
    for (int i = 0; i < len; i++)
        g[i] = 1LL * (2 - 1LL * a[i] * g[i] % mod + mod) % mod * g[i] % mod;
    NTT(g, len, -1);
    for (int i = n; i < len; i++) g[i] = 0;
}

void getLn(ll *f, ll *g, int n) {
    static ll a[N], b[N];
    getDer(f, a, n);
    getInv(f, b, n);
    int len = getLen(n);
    NTT(a, len, 1), NTT(b, len, 1);
    for (int i = 0; i < len; i++) a[i] = a[i] * b[i] % mod;
    NTT(a, len, -1);
    getInt(a, g, len);
    for (int i = n; i < len; i++) g[i] = 0;
    for (int i = 0; i < len; i++) a[i] = b[i] = 0;
}

void getExp(ll *f, ll *g, int n) {
    if (n == 1) return (void) (g[0] = 1);
    getExp(f, g, (n + 1) >> 1);
    static ll a[N];
    getLn(g, a, n);
    a[0] = (f[0] + 1 - a[0] + mod) % mod;
    for (int i = 1; i < n; i++) a[i] = (f[i] - a[i] + mod) % mod;
    int len = getLen(n);
    NTT(a, len, 1), NTT(g, len, 1);
    for (int i = 0; i < len; i++) g[i] = g[i] * a[i] % mod;
    NTT(g, len, -1);
    for (int i = n; i < len; i++) g[i] = 0;
    for (int i = 0; i < len; i++) a[i] = 0;
}

void getPow(ll *f, ll *g, int n, ll k) {
    static ll a[N];
    getLn(f, a, n);
    for (int i = 0; i < n; i++) a[i] = a[i] * k % mod;
    getExp(a, g, n);
    for (int i = 0, len = getLen(n); i < len; i++) a[i] = 0;
}

void getPower(ll *f, ll *g, int n, ll k1, ll k2) {//k1为原始模数, k2为模phi(mod - 1)
    int pos = 0; while (pos < n && !f[pos]) pos++;
```

```cpp
    if (k1 * pos >= n) { for (int i = 0; i < n; i++) g[i] = 0; return; }
    static ll a[N], b[N];
    int m = n - pos, inv = qpow(f[pos], mod - 2, mod), t = qpow(f[pos], k2, mod);
    for (int i = 0; i < m; i++) a[i] = f[i + pos] * inv % mod;
    getLn(a, b, m);
    for (int i = 0; i < m; i++) b[i] = b[i] * k1 % mod;
    getExp(b, g, m);
    for (int i = 0; i < m; i++) g[i] = g[i] * t % mod;
    pos = min(1ll * pos * k1, 1ll * n);
    for (int i = n - 1; i >= pos; i--) g[i] = g[i - pos];
    for (int i = pos - 1; i >= 0; i--) g[i] = 0;
    for (int i = 0, len = getLen(m); i < len; i++) a[i] = b[i] = 0;
}

void fenzhiFFT(ll *f, ll *g, int n) {
    //计算g[i] = \sum_{j = 1}^{i} g[i - j] * f[j]
    static ll a[N];
    for (int i = 1; i < n; i++) a[i] = (mod - f[i]) % mod;
    a[0] = 1;
    ll g0 = g[0];
    getInv(a, g, n);
    for (int i = 0; i < n; i++) g[i] = g[i] * g0 % mod, a[i] = 0;
}

struct Complex { ll x, y; }; ll w;
Complex mul(Complex a, Complex b, ll mod) {
    Complex ans = {0, 0};
    ans.x = ((a.x * b.x % mod + a.y * b.y % mod * w % mod) % mod + mod) % mod;
    ans.y = ((a.x * b.y % mod + a.y * b.x % mod) % mod + mod) % mod;
    return ans;
}
ll binpow_imag(Complex a, ll b, ll mod) {
    Complex ans = {1, 0};
    while (b) {
        if (b & 1) ans = mul(ans, a, mod);
        a = mul(a, a, mod);
        b >>= 1;
    }
    return ans.x % mod;
}
ll cipolla(ll n, ll mod) {
    srand(time(0));
    n %= mod;
    if (mod == 2) return n;
    if (qpow(n, (mod - 1) / 2, mod) == mod - 1) return -1;
    ll a;
    while (1) {
        a = rand() % mod;
        w = ((a * a % mod - n) % mod + mod) % mod;
        if (qpow(w, (mod - 1) / 2, mod) == mod - 1) break;
    }
    Complex x = {a, 1};
    return binpow_imag(x, (mod + 1) / 2, mod);
}
```

```
void getSqrt(ll *f, ll *g, int n) {
    if (n == 1) {
        if (f[0] == 0) g[0] = f[0];
        else {
            ll t = cipolla(f[0], mod);
            g[0] = min(t, mod - t);
        }
        return;
    }
    getSqrt(f, g, (n + 1) >> 1);
    int len = getLen(n);
    static ll a[N], b[N];
    for (int i = 0; i < len; ++i) a[i] = i < n ? f[i] : 0;
    getInv(g, b, n);
    NTT(a, len, 1), NTT(b, len, 1);
    for (int i = 0; i < len; i++) a[i] = 1ll * a[i] * b[i] % mod;
    NTT(a, len, -1);
    for (int i = 0; i < n; i++) g[i] = (g[i] + a[i]) % mod * inv2 % mod;
    for (int i = n; i < len; i++) g[i] = 0;
    for (int i = 0; i < len; i++) a[i] = b[i] = 0;
}

//分治乘法
void solve(ll *f1, ll *f2, ll *g, int l, int r) {
    if (l == r) return (void) (g[(l - 1) * 2] = f1[l], g[(l - 1) * 2 + 1] = f2[l]);
    int mid = (l + r) / 2;
    solve(f1, f2, g, l, mid);
    solve(f1, f2, g, mid + 1, r);
    static ll a[N], b[N];
    int len1 = mid - l + 2, len2 = r - mid + 1;
    for (int i = 0; i < len1; i++) a[i] = g[(l - 1) * 2 + i];
    for (int i = 0; i < len2; i++) b[i] = g[mid * 2 + i];
    int n = r - l + 2, len = getLen(n);
    NTT(a, len, 1), NTT(b, len, 1);
    for (int i = 0; i < len; i++) a[i] = a[i] * b[i] % mod;
    NTT(a, len, -1);
    for (int i = 0; i < n; i++) g[(l - 1) * 2 + i] = a[i];
    for (int i = n; i < len; i++) g[(l - 1) * 2 + i] = 0;
    for (int i = 0; i < len; i++) a[i] = b[i] = 0;

}
```

多项式 2

```
namespace FFT {
#define mod 998244353
#define G 3
#define maxn 501000
    typedef long long ll;
    int bh[maxn], tmpA[maxn], tmpB[maxn], tmpC[maxn], tmp[maxn], lim = maxn, w[2][maxn];

    int qpow(int a, int b) {
        int ans = 1, tmp = a;
        for (; b; b >>= 1, tmp = 1ll * tmp * tmp % mod)
```

```
        if (b & 1)ans = 1ll * ans * tmp % mod;
    return ans;
}

int qpow(int a, int b, int p) {
    int ans = 1, tmp = a;
    for (; b; b >>= 1, tmp = 1ll * tmp * tmp % p)
        if (b & 1)ans = 1ll * ans * tmp % p;
    return ans;
}

void init(int b) {
    for (int i = 1; i < (1 << b); i <<= 1) {
        int wn = qpow(G, (mod - 1) / (i << 1));
        for (int j = 0; j < i; ++j)w[1][i + j] = (j ? 1ll * wn * w[1][i + j - 1] % mod : 1);
        wn = qpow(G, mod - 1 - (mod - 1) / (i << 1));
        for (int j = 0; j < i; ++j)w[0][i + j] = (j ? 1ll * wn * w[0][i + j - 1] % mod : 1);
    }
}

ll gmod(ll x) {
    return x >= mod ? x - mod : x;
}

void fft(int h[], int len, int flag) {
    if (flag == -1)flag = 0;
    for (int i = 0, j = 0; i < len; ++i) {
        if (i < j)swap(h[i], h[j]);
        for (int k = len >> 1; (j ^= k) < k; k >>= 1);
    }
    for (int i = 1; i < len; i <<= 1)
        for (int j = 0; j < len; j += (i << 1))
            for (int k = 0; k < i; ++k) {
                int x = h[j + k], y = (ll) h[j + k + i] * w[flag][i + k] % mod;
                h[j + k] = gmod((ll) x + y);
                h[j + k + i] = gmod((ll) x - y + mod);
            }
    if (flag == 0) {
        int x = qpow(len, mod - 2);
        for (int i = 0; i < len; ++i)
            h[i] = 1ll * h[i] * x % mod;
    }
}

void poly_mul(const int A[], int lenA, const int B[], int lenB, int C[], int lenc) {
    int l = 1, k = 0, L = min(lenA, lenc) + min(lenB, lenc) + 1;
    if (1ll * lenA * lenB <= 400) {
        for (int i = 0; i < L; ++i)tmpA[i] = 0;
        for (int i = 0; i < lenA; ++i)
            if (A[i])
                for (int j = 0; j < lenB; ++j)
                    if (B[j])
```

```
                        tmpA[i + j] = (tmpA[i + j] + 1ll * A[i] * B[j]) % mod;
        for (int i = 0; i < lenc && i < L; ++i)C[i] = tmpA[i];
        for (int i = L; i < lenc; ++i)C[i] = 0;
        return;
    }
    while (l < L)l <<= 1, k++;
    memset(tmpA, 0, l << 2);
    memset(tmpB, 0, l << 2);
    memcpy(tmpA, A, min(lenA, lenc) << 2);
    memcpy(tmpB, B, min(lenB, lenc) << 2);
    fft(tmpA, l, 1), fft(tmpB, l, 1);
    for (int i = 0; i < l; ++i)tmpA[i] = 1ll * tmpA[i] * tmpB[i] % mod;
    fft(tmpA, l, -1);
    for (int i = 0; i < lenc && i < L; ++i)C[i] = tmpA[i];
    for (int i = L; i < lenc; ++i)C[i] = 0;
}

void poly_inv(int n, const int a[], int C[]) {
    if (n == 1) {
        C[0] = qpow(a[0], mod - 2);
        return;
    }
    static int A[maxn];
    A[0] = qpow(a[0], mod - 2);
    int m = (n + 1) / 2, len = 1, k = 0;
    poly_inv(m, a, C);
    while (len < n + n)len <<= 1, k++;
    memcpy(A, a, n << 2);
    memset(A + n, 0, (len - n) << 2);
    memset(C + m, 0, (len - m) << 2);
    fft(A, len, 1), fft(C, len, 1);
    for (int i = 0; i < len; ++i)C[i] = 1ll * (2ll - 1ll * A[i] * C[i] % mod + mod) * C[i] % mod;
    fft(C, len, -1);
}

void poly_div(const int a[], int lena, const int b[], int lenb, int C[], int &plen) {
    static int A[maxn], B[maxn];
    while (!a[lena - 1] && lena >= 1)lena--;
    while (!b[lenb - 1] && lenb >= 1)lenb--;
    if (lena < lenb) {
        plen = 1, C[0] = 0;
        return;
    }
    plen = lena - lenb + 1;
    memcpy(A, a, lena << 2);
    memcpy(B, b, lenb << 2);
    reverse(A, A + lena);
    reverse(B, B + lenb);
    poly_inv(plen, B, tmp);
    poly_mul(tmp, plen, A, lena, C, plen);
    reverse(C, C + plen);
}

```

```
typedef vector<int> Poly;

int SZ(const Poly &v) { return v.size(); }

void upd(Poly &v) { while (v.size() > 1 && !v.back())v.pop_back(); }

Poly operator*(const Poly &a, const Poly &b) {
    Poly ret(SZ(a) + SZ(b) - 1);
    poly_mul(a.data(), SZ(a), b.data(), SZ(b), ret.data(), SZ(ret));
    upd(ret);
    return ret;
}

Poly operator/(const Poly &a, const Poly &b) {
    int len;
    poly_div(a.data(), SZ(a), b.data(), SZ(b), tmp, len);
    Poly ret = Poly(tmp, tmp + len);
    upd(ret);
    return ret;
}

Poly operator-(const Poly &a, const Poly &b) {
    Poly
    ret(max(SZ(a), SZ(b)));
    for (int i = 0; i < SZ(ret); ++i)
        ret[i] = (1ll * (i < SZ(a) ? a[i] : 0) - (i < SZ(b) ? b[i] : 0) + mod) % mod;
    upd(ret);
    return ret;
}

Poly operator+(const Poly &a, const Poly &b) {
    Poly
    ret(max(SZ(a), SZ(b)));
    for (int i = 0; i < SZ(ret); ++i)
        ret[i] = (1ll * (i < SZ(a) ? a[i] : 0) + (i < SZ(b) ? b[i] : 0)) % mod;
    upd(ret);
    return ret;
}

Poly operator%(const Poly &a, const Poly &b) {
    int len;
    poly_div(a.data(), SZ(a), b.data(), SZ(b), tmp, len);
    poly_mul(b.data(), SZ(b), tmp, len, tmp, SZ(a));
    for (int i = 0; i < SZ(a); ++i)
        tmp[i] = (1ll * a[i] - tmp[i] + mod) % mod;
    for (len = SZ(a); len > 1 && !tmp[len - 1]; len--);
    return Poly(tmp, tmp + len);
}

void print(const Poly &x) {
    printf("\n[len=%d]", SZ(x));
    for (int i = 0; i < SZ(x); ++i)
        printf("%d ", x[i]);
    puts("");
```

```
    }

    Poly getinv(Poly g, int n) {
        for (int i = 0; i < n; ++i)tmpC[i] = 0;
        for (int i = 0; i < g.size() && i < n; ++i)tmpC[i] = g[i];
        poly_inv(n, tmpC, tmp);
        return Poly(tmp, tmp + n);
    }

    int getrt(int p) {
        if (p == 2)return 1;
        for (int i = 2; i < p; i++) {
            int flg = 1, s = p - 1;
            for (int j = 1; 1ll * j * j <= s && flg; j++)
                if (s % j == 0) {
                    flg &= qpow(i, j, p) != 1;
                    if (j != 1)flg &= qpow(i, s / j, p) != 1;
                }
            if (flg)return i;
        }
        exit(1);
    }

#undef maxn
#undef G
#undef mod
};
```

4.2.2 拉格朗日插值

```
//拉格朗日插值，求点值转化为的多项式f(x)
//得到f(k) = \sum_{i = 1} ^ {n} y[i] \pi_{i != j} \frac{k - x[j]}{x[i] - x[j]}

int lagrange(int n, int *x, int *y, int k) {
    ll res = 0;
    for (int i = 0; i < n; i++) {
        ll s1 = 1, s2 = 1;
        for (int j = 0; j < n; j++)
            if (i != j) {
                s1 = s1 * (k - x[j] + mod) % mod;
                s2 = s2 * (x[i] - x[j] + mod) % mod;
            }
        res = (res + 1ll * y[i] * s1 % mod * inv(s2) % mod) % mod;
    }
    return (res + mod) % mod;
}
```

4.2.3 矩阵

循环矩阵快速幂优化

```
const int N = 5e2 + 10;
const ll mod = 998244353;

```

```cpp
ll qpow(ll a, ll b) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

void mul(ll a[], ll b[], int n) {//a = a * b
    static ll res[N];
    for (int i = 0; i < n; i++) res[i] = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            res[(i + j) % n] = (res[(i + j) % n] + 1ll * a[i] * b[j] % mod) % mod;
    memcpy(a, res, sizeof(res));
}

void qpow(ll a[], int n, ll k) {//a = a^k
    static ll res[N];
    for (int i = 0; i < n; i++) res[i] = (i == 0);
    while (k) {
        if (k & 1) mul(res, a, n);
        mul(a, a, n);
        k >>= 1;
    }
    memcpy(a, res, sizeof(res));
}

ll x[N], A[N], ans[N];

int main() {

    //x[i] = a * x[i - 1] + b * x[i] + c * x[i + 1]
    A[n - 1] = a, A[0] = b, A[1] = c;
    qpow(A, n, k);
    //X = A^{k} * X
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            ans[j] = (ans[j] + 1ll * A[(j - i + n) % n] * x[i] % mod) % mod;
    for (int i = 0; i < n; i++)
        printf("%lld%s", ans[i], i == n - 1 ? "\n" : " ");

    return 0;
}
```

矩阵光速幂

```cpp
struct Matrix {
    static const int N = 510;
    int a[N][N], n;
    Matrix(int siz) {
        n = siz;
```

```
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                a[i][j] = 0;
    }
    Matrix operator * (const Matrix& A) const {//重载矩阵乘法
        Matrix res = Matrix(n);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                for (int k = 0; k < n; k++)
                    res.a[i][j] = (res.a[i][j] + 1ll * A.a[i][k] * a[k][j] % mod) % mod;
        return res;
    }
};

Matrix power(Matrix A, ll k) {//普通的快速幂
    Matrix res = Matrix(A.n);
    for (int i = 0; i < res.n; i++) res.a[i][i] = 1;
    while (k) {
        if (k & 1)res = res * A;
        A = A * A;
        k >>= 1;
    }
    return res;
}

Matrix power(Matrix A, string s) {//优化的快速幂
    Matrix res = Matrix(A.n);
    for (int i = 0; i < res.n; i++) res.a[i][i] = 1;
    for (int i = s.length() - 1; i >= 0; --i) {
        if (s[i] != '0') res = res * power(A, s[i] - '0');
        A = power(A, 10);
    }
    return res;
}
```

4.2.4 矩阵求逆

```
//原始矩阵A[0, n - 1][0, n - 1]
//右边一个单位阵I, 在a[0, n - 1][n, (n << 1) - 1]
//将左边A变成单位阵时，右边的I变为A^-1

bool Gauss(ll a[][MAX << 1], int n) {
    for (int i = 0, r; i < n; i++) {
        r = i;
        for (int j = i + 1; j < n; j++)
            if (a[j][i] > a[r][i]) r = j;
        if (r != i) swap(a[i], a[r]);
        if (!a[i][i]) return false;//无解

        ll inv = qpow(a[i][i], mod - 2);
        for (int k = 0; k < n; k++) {
            if (k == i) continue;
            ll t = a[k][i] * inv % mod;
```

```
            for (int j = i; j < (n << 1); j++)
                a[k][j] = (a[k][j] - t * a[i][j] % mod + mod) % mod;
        }
        for (int j = 0; j < (n << 1); j++) a[i][j] = a[i][j] * inv % mod;
    }
    return true;
}

int main() {


    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        a[i][i + n] = 1;
        for (int j = 0; j < n; j++)
            scanf("%lld", &a[i][j]);
    }

}
```

4.2.5 高斯消元

```
bool gauss(double a[][MAX], int n) {
    //a[][]为增广矩阵[0, n) x [0, n)
    //最后解在a[][n]中
    int i, j, k, r;
    for (i = 0; i < n; i++) {
        r = i;
        for (j = i + 1; j < n; j++)
            if (abs(a[j][i]) > abs(a[r][i])) r = j;
        if (abs(a[r][i]) < eps) return 0;//无穷多解
        if (r != i) {
            for (j = 0; j <= n; j++)
                swap(a[r][j], a[i][j]);
        }
        for (j = n; j >= i; j--)
            for (k = i + 1; k < n; k++)
                a[k][j] -= a[k][i] / a[i][i] * a[i][j];
    }
    for (i = n - 1; i >= 0; i--) {
        for (j = i + 1; j < n; j++)
            a[i][n] -= a[j][n] * a[i][j];
        a[i][n] /= a[i][i];
    }
    return 1;//唯一解
}
```

4.3 组合数学

4.3.1 Lucas

```
ll qpow(ll a, ll b) {
    ll res = 1;
```

```cpp
    while (b) {
        if (b & 1)
            res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

ll fac[N];
void init(int siz) {
    fac[0] = 1;
    for (int i = 1; i <= siz; i++) fac[i] = i * fac[i - 1] % mod;
}

ll C(ll n, ll m) {
    if (n < m) return 0;
    return fac[n] * qpow(fac[m], mod - 2) % mod * qpow(fac[n - m], mod - 2) % mod;
}

ll Lucas(ll n, ll m) {
    if (m == 0) return 1;
    return Lucas(n / mod, m / mod) * C(n % mod, m % mod) % mod;
}
```

4.3.2 二项式相关

```cpp
/*
广义二项式:
(1 + x) ^ {-k} = \sum_{i = 0} ^ {inf} C(k + i - 1, i) * x^i * (-1)^i
(1 - x) ^ {-k} = \sum_{i = 0} ^ {inf} C(k + i - 1, i) * x^i
*/
```

4.3.3 卡特兰数

```cpp
//卡特兰数有以下几种形式:
//
//
//

int catalan[N];
void getCatalan(int siz) {
    catalan[0] = catalan[1] = 1;
    for (int i = 2; i <= siz; i++) {
        //catalan[i] = catalan[i - 1] * (4 * i - 2) % mod * qpow(i + 1, mod - 2) % mod;
        catalan[i] = (C(2 * i, i) - C(2 * i, i - 1) + mod) % mod;
    }
}
```

4.3.4 容斥原理

```
/*
集合容斥
f(S) = \sum_{T \belong S} g(T)
g(S) = \sum_{T \belong S} f(T) * (−1) ^ {|S| − |T|}

Min−Max容斥
max(S) = \sum_{T \belong S} min(T) * (-1) ^ {|T| - 1}
min(S) = \sum_{T \belong S} max(T) * (-1) ^ {|T| - 1}
第k大, 同理有第k小
max_k(S) = \sum_{T \belong S} min(T) * (-1) ^ {|T| - k} * C(|T| − 1, k − 1)
对期望也适用
E[max(S)] = \sum_{T \belong S} E[min(T)] * (-1) ^ {|T| - 1}
*/
```

4.3.5 第二类斯特林数

```
//S(n, m) = ifac[m] * \sum{k = 0}^{m} (−1)^k * C(m, k) * (m − k)^n

ll S(ll n, ll m) {
    ll res = 0;
    for (int k = 0, sign = 1; k <= m; k++, sign *= −1)
        res = (res + sign * C(m, k) * qpow(m − k, n) % mod) % mod;
    res = (res + mod) % mod;
    return res * ifac[m] % mod;
}
```

4.3.6 组合数

```
ll C(ll n, ll m) {
    if (m == 0 || n == m) return 1;
    if (m > n) return 0;
    if (m == 1) return n;
    if (c[n][m]) return c[n][m];
    else return c[n][m] = C(n − 1, m) + C(n − 1, m − 1);
}


//预处理版本
ll qpow(ll a, ll b) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

ll fac[N], ifac[N];
void init(int siz) {
    fac[0] = 1;
    for (int i = 1; i <= siz; i++)
        fac[i] = i * fac[i − 1] % mod;
```

```
    ifac[siz] = qpow(fac[siz], mod - 2);
    for (int i = siz; i >= 1; i--) ifac[i - 1] = ifac[i] * i % mod;
}

ll C(ll n, ll m) {
    if (m == 0 || n == m) return 1;
    if (m > n) return 0;
    if (m == 1) return n;
    return fac[n] * ifac[m] % mod * ifac[n - m] % mod;
}
```

4.4 计算几何

4.4.1 凸包

```
#include <stdio.h>
#include <math.h>
#include <algorithm>

using namespace std;

const int maxn = 1005;

struct point
{
    double x, y;
}plist[maxn];
int pstack[maxn], top;

double cross(const point &p0, const point &p1, const point&p2)
{
    return (p1.x - p0.x)*(p2.y - p0.y) - (p2.x - p0.x)*(p1.y - p0.y);
}

double dis(const point &p1, const point &p2)
{
    return sqrt((p2.x - p1.x)*(p2.x - p1.x) + (p2.y - p1.y)*(p2.y - p1.y));
}

bool cmp(const point &p1, const point &p2)
{
    double tmp = cross(plist[0], p1, p2);
    if (tmp > 0.0)return true;
    else if (tmp == 0.0 && dis(plist[0], p1) < dis(plist[0], p2))return true;
    else return false;
}


void Graham(int n)
{

    top = 1;
    pstack[0] = 0;
```

```
    pstack[1] = 1;
    for (int i = 2; i < n; i++) {
        while (top > 0 && cross(plist[pstack[top - 1]], plist[pstack[top]], plist[i]) <= 0) top--;
        pstack[++top] = i;
    }

}

int main()
{
    int n;
    double res;
    while (scanf("%d", &n) != EOF && n) {
        res = 0;
        int k = 0;
        scanf("%lf %lf", &plist[0].x, &plist[0].y);
        point p0 = plist[0];
        for (int i = 1; i < n; i++) {
            scanf("%lf %lf", &plist[i].x, &plist[i].y);
            if (p0.y > plist[i].y || (p0.y == plist[i].y&&p0.x > plist[i].x)) {
                p0.x = plist[i].x;
                p0.y = plist[i].y;
                k = i;
            }
        }

        plist[k] = plist[0];
        plist[0] = p0;

        if (n == 1)
        {
            printf("0.00\n");
            continue;
        }
        else if (n == 2)
        {
            printf("%.2lf\n", dis(plist[0], plist[1]));
            continue;
        }

        sort(plist + 1, plist + n, cmp);
        Graham(n);
        for (int i = 1; i <= top; i++) {
            res += dis(plist[pstack[i - 1]], plist[pstack[i]]);
        }
        res += dis(plist[pstack[top]], plist[0]);
        printf("%.2lf\n", res);
    }
    return 0;
}
```

5 数据结构

5.1 01Trie

```
struct Trie {
    static const int MAX_N = N * 35, rt = 0;
    static const int MAX_BIT = 31;
    int ch[MAX_N][2], sum[MAX_N], tot;
    void init() { tot = 0; ch[tot][0] = ch[tot][1] = sum[tot] = 0; }
    void insert(ll val) {
        int u = rt;
        for (int bit = MAX_BIT; bit >= 0; bit--) {
            int s = (val >> bit) & 1; sum[u]++;
            if (!ch[u][s]) {
                ch[u][s] = ++tot;
                ch[tot][0] = ch[tot][1] = sum[tot] = 0;
            }
            u = ch[u][s];
        }
        sum[u]++;
    }
    ll queryKth(ll val, int k = 1) {//第k大
        int u = rt; ll res = 0;
        for (int bit = MAX_BIT; bit >= 0; bit--) {
            int s = (val >> bit) & 1;
            if (!ch[u][s] && !ch[u][s ^ 1]) return res;
            if (!ch[u][s ^ 1]) u = ch[u][s];
            else if (sum[ch[u][s ^ 1]] >= k) u = ch[u][s ^ 1], res += (1ll << bit);
            else k -= sum[ch[u][s ^ 1]], u = ch[u][s];
        }
        return res;
    }
} trie;
```

5.2 BIT

```
#define lowbit(x) x&-x
int BIT[N];
void upd(int p, int k) { for (; p < N; p += lowbit(p)) BIT[p] += k; }
int ask(int p) { int res = 0; for (; p; p -= lowbit(p)) res += BIT[p]; return res; }
```

5.3 HashTable

```
struct HashTable {
    static const int MOD = 1e7 + 10;
    struct edge {
        int nxt;
        ll num, val;
    } e[MOD];
    int head[MOD], tot;
    void clear() { tot = 0; memset(head, 0, sizeof(head)); }
    void insert(ll u, ll w) { e[++tot] = edge{head[u % MOD], u, w }, head[u % MOD] = tot; }
```

```
    int find(ll u) {
        for (int i = head[u % MOD]; i; i = e[i].nxt)
            if (e[i].num == u) return e[i].val;
        return -1;
    }
} hs;
```

5.4 K-D_Tree

```
//查包含在x1,y1,x2,y2为左下角和右上角的矩形里面权值之和
//K-D Tree 二维划分树
int N, ans, rt, WD, tot, top, rub[MAX];

struct node {
    int x[2], w;
} p[MAX];

struct K_D_tree {
    int ls, rs, siz, mn[2], mx[2], sum;
    //mn[0], mx[0] -> x的取值范围
    //mn[1], mx[1] -> y的取值范围
    node tmp;
} t[MAX];

int operator < (const node &a, const node &b) { return a.x[WD] < b.x[WD]; }

int newnode() {
    if (top) return rub[top--];
    else return ++tot;
}

void push_up(int u) {
    for (int i = 0; i <= 1; i++) {//更新x, y的取值范围
        t[u].mn[i] = t[u].mx[i] = t[u].tmp.x[i];
        if (lc) {//左子树的最大最小值
            t[u].mn[i] = min(t[u].mn[i], t[lc].mn[i]);
            t[u].mx[i] = max(t[u].mx[i], t[lc].mx[i]);
        }
        if (rc) {//右子树的最大最小值
            t[u].mn[i] = min(t[u].mn[i], t[rc].mn[i]);
            t[u].mx[i] = max(t[u].mx[i], t[rc].mx[i]);
        }
    }
    t[u].sum = t[lc].sum + t[rc].sum + t[u].tmp.w;
    t[u].siz = t[lc].siz + t[rc].siz + 1;
}

int build(int l, int r, int wd) {
    if (l > r) return 0;
    int u = newnode();
    WD = wd; nth_element(p + l, p + m, p + r + 1);
    t[u].tmp = p[m];
    t[u].ls = build(l, m - 1, wd ^ 1);
```

```
    t[u].rs = build(m + 1, r, wd ^ 1);
    push_up(u);
    return u;
}

void pia(int u, int num) {//拍扁回炉重做
    if (lc) pia(lc, num);
    p[t[lc].siz + num + 1] = t[u].tmp, rub[++top] = u;
    if (rc) pia(rc, t[lc].siz + num + 1);
}

void check(int &u, int wd) {//检查是否平衡，不平衡则需要重建
    if (t[u].siz * alpha < t[lc].siz || t[u].siz * alpha < t[rc].siz) pia(u, 0), u = build(1, t[u].siz, wd);
}

void insert(int &u, node tp, int wd) {//插入点
    if (!u) {
        u = newnode();
        lc = rc = 0, t[u].tmp = tp;
        push_up(u);
        return;
    }
    if (tp.x[wd] < t[u].tmp.x[wd]) insert(lc, tp, wd ^ 1);
    else insert(rc, tp, wd ^ 1);
    push_up(u);
    check(u, wd);
}

bool in(int x1, int y1, int x2, int y2, int X1, int Y1, int X2, int Y2) {//完全被包含
    return (x1 <= X1 && X2 <= x2 && y1 <= Y1 && Y2 <= y2);
}

bool out(int x1, int y1, int x2, int y2, int X1, int Y1, int X2, int Y2) {//完全无交集
    return (x1 > X2 || x2 < X1 || y1 > Y2 || y2 < Y1);
}

int query(int u, int x1, int y1, int x2, int y2) {
    if (!u) return 0;
    int res = 0;
    if (in(x1, y1, x2, y2, t[u].mn[0], t[u].mn[1], t[u].mx[0], t[u].mx[1])) return t[u].sum;
    if (out(x1, y1, x2, y2, t[u].mn[0], t[u].mn[1], t[u].mx[0], t[u].mx[1])) return 0;
    if (in(x1, y1, x2, y2, t[u].tmp.x[0], t[u].tmp.x[1], t[u].tmp.x[0], t[u].tmp.x[1])) res += t[u].tmp.w;
    res += query(lc, x1, y1, x2, y2) + query(rc, x1, y1, x2, y2);
    return res;
}

void init() {
    ans = rt = top = tot = 0;
}

int main() {
```

```

    init();
    scanf("%d", &N);
    while (1) {
        int op; scanf("%d", &op);
        if (op == 3) break;
        if (op == 2) {
            int x1, y1, x2, y2; scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            printf("%d\n", ans = query(rt, x1 ^ ans, y1 ^ ans, x2 ^ ans, y2 ^ ans));
        }
        else {
            int x, y, w; scanf("%d%d%d", &x, &y, &w);
            insert(rt, node{x ^ ans, y ^ ans, w ^ ans}, 0);
        }
    }


    return 0;
}
```

5.5 Scapegoat

```
namespace Scapegoat_Tree {
#define MAXN (100000 + 10)
    const double alpha = 0.75;
    struct Node {
    Node * ch[2];
    int key, size, cover; // size为有效节点的数量，cover为节点总数量
    bool exist; // 是否存在（即是否被删除）
    void PushUp(void) {
        size = ch[0]->size + ch[1]->size + (int)exist;
        cover = ch[0]->cover + ch[1]->cover + 1;
    }
    bool isBad(void) { // 判断是否需要重构
        return ((ch[0]->cover > cover * alpha + 5) ||
                (ch[1]->cover > cover * alpha + 5));
        }
    };
    struct STree {
    protected:
        Node mem_poor[MAXN]; //内存池，直接分配好避免动态分配内存占用时间
        Node *tail, *root, *null; // 用null表示NULL的指针更方便，tail为内存分配指针，root为根
        Node *bc[MAXN]; int bc_top; // 储存被删除的节点的内存地址，分配时可以再利用这些地址

        Node * NewNode(int key) {
            Node * p = bc_top ? bc[--bc_top] : tail++;
            p->ch[0] = p->ch[1] = null;
            p->size = p->cover = 1; p->exist = true;
            p->key = key;
            return p;
        }
        void Travel(Node * p, vector<Node *>&v) {
            if (p == null) return;
```

```
        Travel(p->ch[0], v);
        if (p->exist) v.push_back(p); // 构建序列
        else bc[bc_top++] = p; // 回收
        Travel(p->ch[1], v);
    }
    Node * Divide(vector<Node *>&v, int l, int r) {
        if (l >= r) return null;
        int mid = (l + r) >> 1;
        Node * p = v[mid];
        p->ch[0] = Divide(v, l, mid);
        p->ch[1] = Divide(v, mid + 1, r);
        p->PushUp(); // 自底向上维护，先维护子树
        return p;
    }
    void Rebuild(Node * &p) {
        static vector<Node *>v; v.clear();
        Travel(p, v); p = Divide(v, 0, v.size());
    }
    Node ** Insert(Node *&p, int val) {
        if (p == null) {
            p = NewNode(val);
            return &null;
        }
        else {
            p->size++; p->cover++;

            // 返回值储存需要重构的位置，若子树也需要重构，本节点开始也需要重构，以本节点为根重构
            Node ** res = Insert(p->ch[val >= p->key], val);
            if (p->isBad()) res = &p;
            return res;
        }
    }
    void Erase(Node *p, int id) {
        p->size--;
        int offset = p->ch[0]->size + p->exist;
        if (p->exist && id == offset) {
            p->exist = false;
            return;
        }
        else {
            if (id <= offset) Erase(p->ch[0], id);
            else Erase(p->ch[1], id - offset);
        }
    }
public:
    void Init(void) {
        tail = mem_poor;
        null = tail++;
        null->ch[0] = null->ch[1] = null;
        null->cover = null->size = null->key = 0;
        root = null; bc_top = 0;
    }
    STree(void) { Init(); }
```

```

        void Insert(int val) {
            Node ** p = Insert(root, val);
            if (*p != null) Rebuild(*p);
        }
        int Rank(int val) {
            Node * now = root;
            int ans = 1;
            while (now != null) { // 非递归求排名
                if (now->key >= val) now = now->ch[0];
                else {
                    ans += now->ch[0]->size + now->exist;
                    now = now->ch[1];
                }
            }
            return ans;
        }
        int Kth(int k) {
            Node * now = root;
            while (now != null) { // 非递归求第K大
                if (now->ch[0]->size + 1 == k && now->exist) return now->key;
                else if (now->ch[0]->size >= k) now = now->ch[0];
                else k -= now->ch[0]->size + now->exist, now = now->ch[1];
            }
        }
        void Erase(int k) {
            Erase(root, Rank(k));
            if (root->size < alpha * root->cover) Rebuild(root);
        }
        void Erase_kth(int k) {
            Erase(root, k);
            if (root->size < alpha * root->cover) Rebuild(root);
        }
    };
#undef MAXN

}
```

5.6 Splay

```
struct Splay {
#define rt  ch[0][1]
#define lc  ch[u][0]
#define rc  ch[u][1]
#define identify(u) (ch[fa[u]][1] == u)
    int tot, totElement;
    int val[N], cnt[N], fa[N], sum[N];
    int ch[N][2];
    Splay() { tot = totElement = 0; }
    void push_up(int u) { sum[u] = sum[lc] + sum[rc] + cnt[u]; }
    void connect(int u, int par, int son) { ch[par][son] = u, fa[u] = par; }
    void rotate(int u) {
        int fc = identify(u), f = fa[u];
```

```
        int gc = identify(f), g = fa[f];
        int uc = fc ^ 1, son = ch[u][uc];
        connect(son, f, fc);
        connect(f, u, uc);
        connect(u, g, gc);
        push_up(f);
        push_up(u);
    }
    void splay(int u, int v) {
        v = fa[v];
        while (fa[u] != v) {
            int f = fa[u];
            if (fa[f] != v)
                rotate(identify(u) ^ identify(f) ? u : f);
            rotate(u);
        }
    }
    int creat(int v, int par) {
        val[++tot] = v;
        fa[tot] = par;
        sum[tot] = cnt[tot] = 1;
        return tot;
    }
    void destory(int u) {
        val[u] = cnt[u] = fa[u] = sum[u] = lc = rc = 0;
        tot -= u == tot;
    }
    int find(int v) {
        int u = rt;
        while (1) {
            if (val[u] == v) {
                splay(u, rt);
                return u;
            }
            int nxt = v > val[u];
            if (!ch[u][nxt]) return -1;
            u = ch[u][nxt];
        }
    }
    int insert(int v) {
        totElement++;
        if (totElement == 1) {
            creat(v, 0);
            return rt = tot;
        }
        int u = rt;
        while (1) {
            sum[u]++;
            if (v == val[u]) {
                cnt[u]++;
                return u;
            }
            int nxt = v > val[u];
            if (!ch[u][nxt]) {
```

```
                creat(v, u);
                splay(ch[u][nxt] = tot, rt);
                return tot;
            }
            u = ch[u][nxt];
        }
    }
    void remove(int v) {
        int u = find(v);
        if (!u) return;
        totElement--;
        if (cnt[u] > 1) {
            cnt[u]--, sum[u]--;
            return;
        }
        if (!lc) fa[rt = rc] = 0;
        else {
            int now = lc;
            while (ch[now][1]) now = ch[now][1];
            splay(now, lc);
            connect(rc, now, 1); connect(now, 0, 1);
            push_up(now);
        }
        destory(u);
    }
    int getRank(int v) {
        int k = 0, u = rt;
        while (1) {
            if (v == val[u]) {
                k += sum[lc] + 1;
                splay(u, rt);
                return k;
            }
            else if (v < val[u]) u = lc;
            else {
                k += sum[lc] + cnt[u];
                u = rc;
            }
            if (!u) return 0;
        }
    }
    int atRank(int k) {
        if (k > totElement) return -1;
        int u = rt;
        while (1) {
            if (k > sum[lc] && k <= sum[lc] + cnt[u]) {
                splay(u, rt);
                break;
            }
            if (k <= sum[lc]) u = lc;
            else {
                k -= sum[lc] + cnt[u];
                u = rc;
            }
```

```
        }
        return val[u];
    }
    int upper(int v) {
        int u = rt, minn = 0x3f3f3f3f, p = 0;
        while (u) {
            if (val[u] > v && val[u] < minn) minn = val[u], p = u;
            if (v >= val[u]) u = rc;
            else u = lc;
        }
        if (!p) splay(p, rt);
        return minn;
    }
    int lower(int v) {
        int u = rt, maxx = -0x3f3f3f3f, p = 0;
        while (u) {
            if (val[u] < v && val[u] > maxx) maxx = val[u], p = u;
            if (v <= val[u]) u = lc;
            else u = rc;
        }
        if (!p) splay(p, rt);
        return maxx;
    }
};
```

5.7 Splay 区间翻转

```
struct Splay {
#define rt  ch[0][1]
#define lc  ch[u][0]
#define rc  ch[u][1]
    int tot, totElement;
    int val[MAX], fa[MAX], sum[MAX], tag[MAX];
    int ch[MAX][2];
    Splay() { tot = totElement = 0; }
    void push_up(int u) { sum[u] = sum[lc] + sum[rc] + 1; }
    void push_down(int u) {
        if (tag[u]) {
            tag[lc] ^= 1;
            tag[rc] ^= 1;
            tag[u] = 0;
            swap(lc, rc);
        }
    }
    int identify(int u) { return ch[fa[u]][1] == u; }
    void connect(int u, int par, int son) { ch[par][son] = u, fa[u] = par; }
    void rotate(int u) {
        int fc = identify(u), f = fa[u];
        int gc = identify(f), g = fa[f];
        int uc = fc ^ 1, son = ch[u][uc];
        connect(son, f, fc);
        connect(f, u, uc);
        connect(u, g, gc);
```

```
        push_up(f);
        push_up(u);
    }
    void splay(int u, int v) {
        v = fa[v];
        while (fa[u] != v) {
            int f = fa[u];
            if (fa[f] != v)
                rotate(identify(u) ^ identify(f) ? u : f);
            rotate(u);
        }
    }
    int creat(int v, int par) {
        val[++tot] = v;
        fa[tot] = par;
        sum[tot] = 1;
        return tot;
    }
    int insert(int v) {
        totElement++;
        if (totElement == 1) {
            creat(v, 0);
            return rt = tot;
        }
        int u = rt;
        while (1) {
            sum[u]++;
            if (v == val[u]) return u;
            int nxt = v > val[u];
            if (!ch[u][nxt]) {
                creat(v, u);
                splay(ch[u][nxt] = tot, rt);
                return tot;
            }
            u = ch[u][nxt];
        }
    }
    int queryKth(int k) {
        if (k > totElement) return -1;
        int u = rt;
        while (1) {
            push_down(u);
            if (k > sum[lc] && k <= sum[lc] + 1) {
                splay(u, rt);
                return val[u];
            }
            if (k <= sum[lc]) u = lc;
            else {
                k -= sum[lc] + 1;
                u = rc;
            }
        }
    }
    void reverse(int ql, int qr) {
```

```
        ql = queryKth(ql);
        qr = queryKth(qr + 2);
        splay(ql, rt);
        splay(qr, ch[ql][1]);
        tag[ch[qr][0]] ^= 1;
    }
} splay;
```

5.8 Trie

```
struct Trie {
    static const int MAX_N = 1e6 + 10, rt = 0;
    int ch[MAX_N][26], sum[MAX_N], tot;
    void init() { tot = 0; memset(ch[tot], 0, sizeof(ch[tot])); sum[tot] = 0; }
    void insert(char *s, int val) {
        int len = strlen(s), u = rt;
        for (int i = 0; i < len; i++) {
            int nxt = s[i] - 'a';
            if (!ch[u][nxt]) {
                ch[u][nxt] = ++tot;
                memset(ch[tot], 0, sizeof(ch[tot]));
                sum[tot] = 0;
            }
            u = ch[u][nxt];
        }
        sum[u] += val;
    }
    int query(char *s) {
        int len = strlen(s), u = rt;
        for (int i = 0; i < len; i++) {
            int nxt = s[i] - 'a';
            if (!ch[u][nxt]) return -1;
            u = ch[u][nxt];
        }
        if (!sum[u]) return -1;
        return sum[u];
    }
};
```

5.9 主席树

5.9.1 动态主席树

```
//树状数组套主席树，动态维护区间第k小
//当前点p作用于[p, N]的主席树（普通主席树是在rt[p - 1]的基础上得到）
//查询[1, p]，即可得到rt[p]时的状态
int N, M;
int a[MAX], b[MAX << 1], n;

struct node {
    int x, y, z;
} q[MAX];
---------------动态主席树-----------------------------------------
```

```cpp
int rt[MAX], ru[MAX], rv[MAX], tot;
int lc[MAX_N], rc[MAX_N], sum[MAX_N];

void update(int &now, int l, int r, int p, int v) {
    if (!now) now = ++tot;
    sum[now] += v;
    if (l < r) {
        if (p <= m) update(lc[now], l, m, p, v);
        else update(rc[now], m + 1, r, p, v);
    }
}

void change(int p, int v) {
    int pos = lower_bound(b + 1, b + 1 + n, a[p]) - b;
    for (int i = p; i <= n; i += lowbit(i))
        update(rt[i], 1, n, pos, v);
}

int query(int ql, int qr, int k) {
    ql--;//前缀和相减，[ql, qr]的状态为pre[qr] - pre[ql - 1]
    int cnt1 = 0, cnt2 = 0;
    for (int i = qr; i; i -= lowbit(i)) ru[++cnt1] = rt[i];
    for (int i = ql; i; i -= lowbit(i)) rv[++cnt2] = rt[i];
    int l = 1, r = n;
    while (l < r) {
        int num = 0;
        for (int i = 1; i <= cnt1; i++) num += sum[lc[ru[i]]];
        for (int i = 1; i <= cnt2; i++) num -= sum[lc[rv[i]]];
        if (k <= num) {
            for (int i = 1; i <= cnt1; i++) ru[i] = lc[ru[i]];
            for (int i = 1; i <= cnt2; i++) rv[i] = lc[rv[i]];
            r = m;
        }
        else {
            for (int i = 1; i <= cnt1; i++) ru[i] = rc[ru[i]];
            for (int i = 1; i <= cnt2; i++) rv[i] = rc[rv[i]];
            l = m + 1;
            k -= num;
        }
    }
    return b[l];
}

int main() {
    scanf("%d%d", &N, &M);
    for (int i = 1; i <= N; i++) scanf("%d", &a[i]), b[++n] = a[i];
    for (int i = 1; i <= M; i++) {
        char ch = getchar();
        while (ch != 'Q' && ch != 'C') ch = getchar();
        scanf("%d%d", &q[i].x, &q[i].y);
        if (ch == 'Q') scanf("%d", &q[i].z);
        else b[++n] = q[i].y;
    }
    sort(b + 1, b + 1 + n);
```

```
    n = unique(b + 1, b + 1 + n) - b - 1;

    for (int i = 1; i <= N; i++)
        change(i, 1);

    for (int i = 1; i <= M; i++) {
        if (!q[i].z) {
            change(q[i].x, -1);
            a[q[i].x] = q[i].y;
            change(q[i].x, 1);
        }
        else {
            printf("%d\n", query(q[i].x, q[i].y, q[i].z));
        }
    }

    return 0;
}
```

5.9.2 区间不同数

```
int last[N];//上一种i出现位置
int rt[N], tot;
int lc[MAX_N], rc[MAX_N], sum[MAX_N];

void update(int &now, int pre, int l, int r, int p, int v) {
    now = ++tot;
    sum[now] = sum[pre] + v, lc[now] = lc[pre], rc[now] = rc[pre];
    if (l < r) {
        if (p <= mid) update(lc[now], lc[pre], l, mid, p, v);
        else update(rc[now], rc[pre], mid + 1, r, p, v);
    }
}

int query(int now, int l, int r, int ql, int qr) {
    if (ql <= l && r <= qr) return sum[now];
    int res = 0;
    if (ql <= mid) res += query(lc[now], l, mid, ql, qr);
    if (qr > mid) res += query(rc[now], mid + 1, r, ql, qr);
    return res;
}

int main() {
    for (int i = 1; i <= n; i++) {
        int x; scanf("%d", &x);
        update(rt[i], rt[i - 1], 1, n, i, 1);
        if (last[x]) update(rt[i], rt[i], 1, n, last[x], -1);//上一种出现就删除掉
        last[x] = i;
    }
    while (m--) {
        int ql, qr; scanf("%d%d", &ql, &qr);
        //rt[qr]包含[1, qr]的信息，这里只查询[ql, qr]部分
        printf("%d\n", query(rt[qr], 1, n, ql, n));
```

```
    }
    return 0;
}
```

5.9.3 区间第 K 小

```
struct Hash {
    int b[N], tot;
    void init() { tot = 0; }
    void insert(int x) { b[++tot] = x; }
    void build() {
        sort(b + 1, b + 1 + tot);
        tot = unique(b + 1, b + 1 + tot) - (b + 1);
    }
    int pos(int x) { return lower_bound(b + 1, b + 1 + tot, x) - b; }
};

const int MAX_N = N * 25;
int rt[N], tot;
int lc[MAX_N], rc[MAX_N], num[MAX_N];

void update(int &now, int pre, int l, int r, int p) {
    now = ++tot;
    num[now] = num[pre] + 1, lc[now] = lc[pre], rc[now] = rc[pre];
    if (l == r) return;
    if (p <= mid) update(lc[now], lc[pre], l, mid, p);
    else update(rc[now], rc[pre], mid + 1, r, p);
}

int query(int now, int pre, int k) {
    int l = 1, r = ha.tot;
    while (l < r) {
        int sum = num[lc[now]] - num[lc[pre]];
        if (k <= sum) {
            now = lc[now], pre = lc[pre];
            r = mid;
        }
        else {
            now = rc[now], pre = rc[pre];
            l = mid + 1;
            k -= sum;
        }
    }
    return ha.b[l];
}
```

5.9.4 树上建树

```
//树上建主席树，支持动态修改边
int N, M, Q;
int a[MAX], b[MAX], n;

int pos(int x) { return lower_bound(b + 1, b + 1 + n, x) - b; }
```

```

struct edge {
    int nxt, to;
}e[MAX << 2];

int cnt, head[MAX];

void add(int u, int v) {
    e[++cnt].nxt = head[u];
    e[cnt].to = v;
    head[u] = cnt;
}

-------------------------主席树--------------------------------------------
int tot, rt[MAX];
int lc[MAX * 400], rc[MAX * 400], num[MAX * 400];

int update(int pre, int l, int r, int p) {
    int now = ++tot;
    num[now] = num[pre] + 1;
    lc[now] = lc[pre], rc[now] = rc[pre];
    if (l < r) {
        if (p <= m)lc[now] = update(lc[pre], l, m, p);
        else rc[now] = update(rc[pre], m + 1, r, p);
    }
    return now;
}

int query(int u, int v, int x, int y, int k) {
    int l = 1, r = n;
    while (l < r) {
        int ls = num[lc[u]] + num[lc[v]] - num[lc[x]] - num[lc[y]];
        if (k <= ls) {
            u = lc[u], v = lc[v], x = lc[x], y = lc[y];
            r = m;
        }
        else {
            u = rc[u], v = rc[v], x = rc[x], y = rc[y];
            k -= ls, l = m + 1;
        }
    }
    return l;
}
--------------------------LCA---------------------------------------------------
int dep[MAX], fa[MAX][20], lg[MAX], siz[MAX], pre[MAX], fff, vis[MAX];

void init(int N) {
    for (int i = 1; i <= N; i++)
        lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);
}

void dfs(int u, int f) {
```

```
    vis[u] = 1;
    //建树，在fa的基础上建树
    rt[u] = update(rt[f], 1, n, pos(a[u]));
    siz[u] = 1;
    dep[u] = dep[f] + 1;
    fa[u][0] = f, pre[u] = fff;
    //此处本来应该是 (1 << i) <= dep[u]
    //但是这里是动态的树，有边的修改，所以祖先节点也需要更新
    for (int i = 1; i <= 18; i++)
        fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for (int i = head[u]; i; i = e[i].nxt)
        if (e[i].to != f) {
            dfs(e[i].to, u);
            siz[u] += siz[e[i].to];
        }
}

int lca(int x, int y) {
    if (dep[x] < dep[y])
        swap(x, y);
    while (dep[x] > dep[y])
        x = fa[x][lg[dep[x] - dep[y]] - 1];
    if (x == y)return x;
    for (int k = lg[dep[x]] - 1; k >= 0; k--)
        if (fa[x][k] != fa[y][k])
            x = fa[x][k], y = fa[y][k];
    return fa[x][0];
}

int main() {
    int T = read();
    N = read(); M = read(); Q = read();
    init(N);

    for (int i = 1; i <= N; i++)a[i] = read(), b[i] = a[i], pre[i] = i;
    sort(b + 1, b + 1 + N);
    n = unique(b + 1, b + 1 + N) - (b + 1);

    for (int i = 1; i <= M; i++) {
        int u, v;
        u = read(); v = read();
        add(u, v); add(v, u);
    }

    for (int i = 1; i <= N; i++)
        if (!vis[i]) {
            fff = i;
            dfs(i, 0);
        }

    char op[10];
    int last_ans = 0;
    while (Q--) {
        scanf("%s", op);
```

```
        int x, y, k;
        x = read(); y = read();
        x ^= last_ans; y ^= last_ans;
        if (op[0] == 'Q') {
            k = read(); k ^= last_ans;
            int xy = lca(x, y);
            last_ans = b[query(rt[x], rt[y], rt[xy], rt[fa[xy][0]], k)];
            printf("%d\n", last_ans);
        }
        else {
            add(x, y); add(y, x);
            int fx = pre[x], fy = pre[y];
            if (siz[fx] < siz[fy]) {
                fff = fy;
                dfs(x, y);
            }
            else {
                fff = fx;
                dfs(y, x);
            }
        }
    }
    return 0;
}
```

5.10 可持久化 01Trie

```
struct Trie {
    static const int MAX_N = N * 35;
    static const int MAX_BIT = 28;
    int rt[N];
    int ch[MAX_N][2], sum[MAX_N], tot;
    void insert(int now, int pre, ll val) {
        rt[now] = ++tot;
        now = rt[now], pre = rt[pre];
        for (int bit = MAX_BIT; bit >= 0; bit--) {
            int s = (val >> bit) & 1;
            sum[now] = sum[pre] + 1;
            if (!ch[now][s]) ch[now][s] = ++tot;
            ch[now][s ^ 1] = ch[pre][s ^ 1];
            now = ch[now][s];
            pre = ch[pre][s];
        }
        sum[now] = sum[pre] + 1;
    }
    ll queryKth(int now, int pre, ll val, int k = 1) {
        now = rt[now], pre = rt[pre];
        ll res = 0;
        for (int bit = MAX_BIT; bit >= 0; bit--) {
            int s = (val >> bit) & 1;
            int lsum = sum[ch[now][s ^ 1]] - sum[ch[pre][s ^ 1]];
            if (lsum >= k) res += (1 << bit), now = ch[now][s ^ 1], pre = ch[pre][s ^ 1];
            else k -= lsum, now = ch[now][s], pre = ch[pre][s];
```

```
        }
        return res;
    }
} trie;
```

5.11 扫描线

```
#include <bits/stdc++.h>
using namespace std;
#define INF 0x3f3f3f3f
#define sz  sizeof
#define eps 1e-9
#define lowbit(x) x&-x
#define lc  u<<1
#define rc  u<<1|1
#define mid (l+r)/2
typedef long long ll;
const int MAX = 1e5 + 10;

int N;
int X[MAX << 1];
int tot, cnt;

int pos(int x) { return lower_bound(X + 1, X + 1 + cnt, x) - X; }

struct Scanline {
    int l, r, h, f;
    bool operator < (const Scanline &rhs) const {
        return h < rhs.h;
    }
} line[MAX << 1];

struct SegmentTree {
    int l, r, sum, len;
} t[MAX << 2];

void push_up(int u) {
    if (t[u].sum) t[u].len = X[t[u].r + 1] - X[t[u].l];
    else t[u].len = t[lc].len + t[rc].len;
}

void build(int u, int l, int r) {
    t[u] = SegmentTree{ l, r, 0, 0 };
    if (l == r) return;
    build(lc, l, mid); build(rc, mid + 1, r);
}

void update(int u, int ql, int qr, int k) {
    if (ql <= t[u].l && t[u].r <= qr) {
        t[u].sum += k;
        return;
    }

```

```
}

int main() {
#ifdef ACM_LOCAL
    freopen("input.in", "r", stdin);
    freopen("output.out", "w", stdout);
    auto start_clock = clock();
#endif
    scanf("%d", &N);
    for (int i = 1; i <= N; i++) {
        int x1, x2, y1, y2; scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        line[++tot] = Scanline{ x1, x2, y1, 1 };
        line[++tot] = Scanline{ x1, x2, y2, -1 };
        X[++cnt] = x1, X[++cnt] = x2;
    }
    sort(line + 1, line + 1 + tot);
    sort(X + 1, X + 1 + cnt);
    cnt = unique(X + 1, X + 1 + cnt) - (X + 1);



#ifdef ACM_LOCAL
    cerr << (double)(clock() - start_clock) / CLOCKS_PER_SEC << "s" << endl;
#endif
    return 0;
}
```

5.12 线性基

```
struct LinerBase {
    static const int MAX_BIT = 60;
    ll num[65], tmp[65];
    bool flag;//能否表示0
    LinerBase() {
        flag = 0;
        memset(num, 0, sizeof(num));
        memset(tmp, 0, sizeof(tmp));
    }
    void insert(ll x) {
        for (int bit = MAX_BIT; ~bit; bit--)
            if (x & (1ll << bit)) {
                if (num[bit]) x ^= num[bit];
                else {
                    num[bit] = x;
                    return;
                }
            }
        flag = 1;
    }
    ll queryMax() {
        ll res = 0;
        for (int bit = MAX_BIT; ~bit; bit--) res = max(res, res ^ num[bit]);
        return res;
```

```
    }
    ll queryMin() {
        if (flag) return 0;
        for (int bit = 0; bit <= MAX_BIT; bit++)
            if (num[bit]) return num[bit];
    }
    ll queryKth(ll k) {
        ll res = 0, cnt = 0;
        k -= flag; if (!k) return 0;
        for (int i = 0; i < MAX_BIT; i++) {
            for (int j = i - 1; ~j; j--)
                if (num[i] & (1ll << j)) num[i] ^= num[j];
            if (num[i]) tmp[cnt++] = num[i];
        }
        if (k >= (1ll << cnt)) return -1;
        for (int i = 0; i < cnt; i++)
            if (k & (1ll << i)) res ^= tmp[i];
        return res;
    }
} linerBase;

struct LinerBase {
    //区间查询
    //构造的时候, lb[i] = lb[i - 1], lb[i].insert(x, i)
    //ans = lb[qr].queryMax(ql)
    static const int MAX_BIT = 30;
    int num[32], pos[32];
    LinerBase() {
        memset(num, 0, sizeof(num));
        memset(pos, 0, sizeof(pos));
    }
    void insert(int x, int p) {
        for (int bit = MAX_BIT; ~bit; bit--)
            if (x & (1ll << bit)) {
                if (!num[bit]) {
                    num[bit] = x, pos[bit] = p;
                    return;
                }
                else if (p > pos[bit]) {
                    swap(num[bit], x);
                    swap(pos[bit], p);
                }
                x ^= num[bit];
            }
    }
    int queryMax(int ql) {
        int res = 0;
        for (int bit = MAX_BIT; ~bit; bit--)
            if (pos[bit] >= ql)
                res = max(res, res ^ num[bit]);
        return res;
    }
} lb[N];

```

```cpp
LinerBase merge(const LinerBase &a, const LinerBase &b) {
    LinerBase res;
    for (int bit = res.MAX_BIT; ~bit; bit--) res.num[bit] = a.num[bit];
    for (int bit = res.MAX_BIT; ~bit; bit--) if (b.num[bit]) res.insert(b.num[bit]);
    return res;
}
```

5.13 线段树动态开点合并分裂

```cpp

#define mid (l+r)/2
static const int MAX_N = N * 40;
int rt[N], now;
int lc[MAX_N], rc[MAX_N];
ll sum[MAX_N];
int tot, rub[MAX_N];
int newNode() { return rub[0] ? rub[rub[0]--] : ++tot; }
void remove(int &u) {
    lc[u] = rc[u] = sum[u] = 0;
    rub[++rub[0]] = u;
    u = 0;
}
void push_up(int u) { sum[u] = sum[lc[u]] + sum[rc[u]]; }
void build(int &u, int l, int r) {
    u = newNode();
    if (l == r) {
        sum[u] = cnt[l];
        return;
    }
    build(lc[u], l, mid); build(rc[u], mid + 1, r);
    push_up(u);
}
void update(int &u, int l, int r, int p, ll k) {
    if (!u) u = newNode();
    if (l == r) {
        sum[u] += k;
        return;
    }
    if (p <= mid) update(lc[u], l, mid, p, k);
    else update(rc[u], mid + 1, r, p, k);
    push_up(u);
}
ll querySum(int u, int l, int r, int ql, int qr) {
    if (!u) return 0;
    if (ql <= l && r <= qr) return sum[u];
    ll res = 0;
    if (ql <= mid) res += querySum(lc[u], l, mid, ql, qr);
    if (qr > mid) res += querySum(rc[u], mid + 1, r, ql, qr);
    return res;
}
int queryKth(int u, int l, int r, ll k) {
    if (l == r) return l;
    if (k <= sum[lc[u]]) return queryKth(lc[u], l, mid, k);
```

```cpp
    else return queryKth(rc[u], mid + 1, r, k - sum[lc[u]]);
}
void merge(int u, int v, int l, int r) {
    if (l == r) {
        sum[u] += sum[v];
        return;
    }
    if (lc[u] && lc[v]) merge(lc[u], lc[v], l, mid), remove(lc[v]);
    else if (lc[v]) lc[u] = lc[v], lc[v] = 0;
    if (rc[u] && rc[v]) merge(rc[u], rc[v], mid + 1, r), remove(rc[v]);
    else if (rc[v]) rc[u] = rc[v], rc[v] = 0;
    push_up(u);
}
void split(int &newp, int &u, int l, int r, int ql, int qr) {//分裂出[ql, qr]间的点
    if (!u) return;
    if (ql <= l && r <= qr) {
        newp = u;
        u = 0;
        return;
    }
    if (!newp) newp = newNode();
    if (ql <= mid) split(lc[newp], lc[u], l, mid, ql, qr);
    if (qr > mid) split(rc[newp], rc[u], mid + 1, r, ql, qr);
    push_up(u);
    push_up(newp);

}
#undef mid
```

6 树和森林

6.1 LCT

```cpp
int ch[N][2], fa[N], rev[N], siz[N];//基本内容
int sum[N], val[N], tag[N];//另外要维护的
#define lc  ch[u][0]
#define rc  ch[u][1]
#define identify(u) (ch[fa[u]][1] == u)
#define isRoot(u)   (u != ch[fa[u]][0] && u != ch[fa[u]][1])
void flip(int u) { swap(lc, rc); rev[u] ^= 1; }
void push_up(int u) {
    siz[u] = siz[lc] + siz[rc] + 1;
    //...
}
void push_down(int u) {
    if (rev[u]) {
        if (lc) flip(lc);
        if (rc) flip(rc);
        rev[u] = 0;
    }
    //...
}
void update(int u) {//当前点之上的所有点都push_down
```

```cpp
    if (!isRoot(u)) update(fa[u]);
    push_down(u);
}
void rotate(int u) {
    int f = fa[u], fc = identify(u);
    int g = fa[f], gc = identify(f);
    int uc = fc ^ 1, c = ch[u][uc];
    if (!isRoot(f))
        ch[g][gc] = u; fa[u] = g;
    ch[f][fc] = c, fa[c] = f;
    ch[u][uc] = f, fa[f] = u;
    push_up(f); push_up(u);
}
void splay(int u) {//将u变为u所在的Splay的根
    update(u);
    for (int f; f = fa[u], !isRoot(u); rotate(u))
        if (!isRoot(f)) rotate(identify(f) ^ identify(u) ? u : f);
}
int access(int u) {//将(rt, u)之间的路径变为实链
    int pre = 0;
    for (; u; u = fa[pre = u])
        splay(u), rc = pre, push_up(u);
    return pre;
}
void makeRoot(int u) {//将u变为整棵树的根(注意:不一定是当前splay的根)
    u = access(u);
    flip(u);
}
int findRoot(int u) {
    access(u), splay(u);
    while (lc) push_down(u), u = lc;
    splay(u);
    return u;
}
void link(int u, int v) {
    makeRoot(u); splay(u);
    if (findRoot(v) != u) fa[u] = v;
}
void split(int u, int v) {
    makeRoot(u);
    access(v); splay(v);//加了这个就将v变为splay的根
}
void cut(int u, int v) {
    makeRoot(u); splay(u);
    if (findRoot(v) == u && fa[v] == u && !ch[v][0]) {
        fa[v] = ch[u][1] = 0;
        push_up(u);
    }
}
void fix(int u, int k) {
    splay(u); val[u] = k;
}
```

6.2 带权并查集

```
int pre[N], num[N], dis[N];//num[i]表示以i为队头时队内元素数量, dis[i]表示到达i的祖先节点的距离

void merge(int x, int y) {
    dis[x] = num[y];
    num[y] += num[x];
    num[x] = 0;
    pre[x] = y;
}

int find(int x) {//路径压缩并查集
    if (x == pre[x]) return x;
    int t = find(pre[x]);
    dis[x] += dis[pre[x]];
    return pre[x] = t;
}
```

6.3 按秩合并可撤回并查集

```
struct node {
    int x, y, z;
};

struct UnionFind {
private:
    int rk[N], pre[N], siz[N], totNode;//N为最大点数
    stack<node> st;//node记录上次修改的内容
public:
    void init(int tot) {
        totNode = tot;
        for (int i = 1; i <= totNode; i++)
            pre[i] = i, siz[i] = rk[i] = 1;
    }
    int find(int x) { while (x ^ pre[x]) x = pre[x]; return x; }
    void merge(int x, int y) {//按秩合并
        x = find(x), y = find(y);
        if (x == y) return;
        if (rk[x] < rk[y]) swap(x, y);
        st.push(node{ y, rk[x], siz[y] });
        pre[y] = x, rk[x] += rk[x] == rk[y], siz[x] += siz[y];
    }
    int start() { return st.size(); }
    void end(int last) {//撤回merge操作
        while (st.size() > last) {
            node tp = st.top();
            rk[pre[tp.x]] -= tp.y, siz[pre[tp.x]] -= tp.z;
            pre[tp.x] = tp.x;
            st.pop();
        }
    }
    bool judge() { return siz[find(1)] == totNode; }
} uf;
```

6.4 树上 K 祖先

```cpp
//倍增KFA, 空间大点, 但是好写
vector<int> g[N];

int anc[N][20];
void dfs(int u, int fa) {
    anc[u][0] = fa;
    for (int i = 1; i <= 19; i++) anc[u][i] = anc[anc[u][i - 1]][i - 1];
    for (auto &v: g[u])
        if (v != fa) dfs(v, u);
}

int kthFa(int u, int k) {
    int bit = 0;
    while (k) {
        if (k & 1) u = anc[u][bit];
        k >>= 1;
        bit++;
    }
    return u;
}


//树剖KFA
int siz[N], son[N], dep[N], fa[N], top[N];
int id[N], nodeOf[N], cnt;
void dfs(int u, int par) {
    dep[u] = dep[fa[u] = par] + (siz[u] = 1);
    for (auto &v: g[u])
        if (v != par) {
            dfs(v, u);
            siz[u] += siz[v];
            if (!son[u] || siz[v] > siz[son[u]])
                son[u] = v;
        }
}

void dfs2(int u, int topf) {
    nodeOf[id[u] = ++cnt] = u, top[u] = topf;
    if (!son[u]) return;
    dfs2(son[u], topf);
    for (auto &v: g[u])
        if (v != fa[u] && v != son[u]) dfs2(v, v);
}

int kthFa(int u, int k) {
    while (k >= id[u] - id[top[u]] + 1 && u) {
        k -= id[u] - id[top[u]] + 1;
        u = fa[top[u]];
    }
    return nodeOf[id[u] - k];
}
```

6.5 树上启发式合并

```
//树上启发式合并和点分治都能解决树上边、点信息问题，一般来说
//如果已经确定树根，就不能用点分治，因为点分治需要找树的重心
//而树上启发式合并更侧重于处理——子树信息

int siz[N], son[N];
void dfs(int u, int fa) {
    siz[u] = 1;
    for (auto &v: g[u])
        if (v != fa) {
            dfs(v, u);
            siz[u] += siz[v];
            if (!son[u] || siz[v] > siz[son[u]])
                son[u] = v;
        }
}
int vis[N];

void upd(int u, int fa, int k) {

    for (auto &v: g[u])
        if (v != fa && !vis[v]) upd(v, u, k);
}

void dsu(int u, int fa, int keep) {//点信息
    for (auto &v: g[u])
        if (v != fa && v != son[u]) dsu(v, u, 0);
    if (son[u]) dsu(son[u], u, 1), vis[son[u]] = 1;
    //更新自己+子树信息的同时统计答案

    if (son[u]) vis[son[u]] = 0;
    if (!keep) upd(u, fa, -1);
}


int id[N], nodeOf[N], cnt;
void dsu(int u, int fa, int keep) {//边信息
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != fa && v != son[u]) dsu(v, u, 0);
    if (son[u]) dsu(son[u], u, 1);
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != fa && v != son[u]) {
            for (int j = 0; j < siz[v]; j++)
                upd(nodeOf[id[v] + j]);
            for (int j = 0; j < siz[v]; j++)
                update(nodeOf[id[v] + j], 1);
        }
    upd(u);
    update(u, 1);
    if (!keep) {
        for (int j = 0; j < siz[u]; j++)
            update(nodeOf[id[u] + j], -1);
```

```cpp
    }
}

int main() {

    int rt = 1; dfs(rt, 0);
    dsu(rt, 0, 0);

    return 0;
}
```

6.6 树剖 LCA

```cpp

int son[N], siz[N], top[N], fa[N], dep[N];
void dfs(int u, int par) {
    dep[u] = dep[fa[u] = par] + (siz[u] = 1);
    int max_son = -1;
    for (auto &v: g[u])
        if (v != par) {
            dfs(v, u);
            siz[u] += siz[v];
            if (max_son < siz[v])
                son[u] = v, max_son = siz[v];
        }
}
void dfs2(int u, int topf) {
    top[u] = topf;
    if (!son[u]) return;
    dfs2(son[u], topf);
    for (auto &v: g[u])
        if (v != fa[u] && v != son[u]) dfs2(v, v);
}
int LCA(int x, int y) {
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        x = fa[top[x]];
    }
    return dep[x] < dep[y] ? x : y;
}
```

6.7 树的直径

```cpp
//树的直径性质:
//1、直径两端点一定是两个叶子节点
//2、距离任意点最远的点一定是直径的一个端点，这个基于贪心求直径方法的正确性可以得出
//3、对于两棵树，如果第一棵树直径两端点为(u,v)(u,v)，第二棵树直径两端点为(x,y)(x,y)，用一条边将两棵树连接，那么新树的直径一定是u,v,x,y,u,v,x,y,中的两个点
//4、对于一棵树，如果在一个点的上接一个叶子节点，那么最多会改变直径的一个端点
//5、若一棵树存在多条直径，那么这些直径交于一点且交点是这些直径的中点

//P3761 [TJOI2017]城市
```

```cpp
//求去掉一条高速公路，并且重新修建一条一样的高速公路(即交通费用一样)，使得这个地区的两个城市之间的最大交通费用最小
//只需每次去掉一条路，查询两边连通块树的直径，树的半径
//最后答案为 两个连通块的树的直径 和 树的半径之和 的最小值

const int MAX = 5e3;
int N;
int dis[MAX], tag[MAX], ans, half;
//tag标记两个点是否要分开


struct edge {
    int nxt, to, w;
} e[MAX << 1];

int head[MAX], tot;

void add(int u, int v, int w) {
    e[++tot].to = v;
    e[tot].w = w;
    e[tot].nxt = head[u];
    head[u] = tot;
}

void init() {
    tot = 0;
    memset(head, 0, sz(head));
    memset(tag, 0, sz(tag));
}

pii dfs(int u, int fa) {//查找树的直径
    pii mx = make_pair(dis[u], u);
    for (int i = head[u], v = e[i].to; i; i = e[i].nxt, v = e[i].to)
        if (v != fa && !tag[v]) {
            dis[v] = dis[u] + e[i].w;
            mx = max(mx, dfs(v, u));
        }
    return mx;
}

pii getDiameter(int start) {//从任意出发点开始，查找树的直径的两个端点，长度为dis[second]
    dis[start] = 0;
    start = dfs(start, 0).second;
    dis[start] = 0;
    return make_pair(start, dfs(start, 0).second);
}

bool findHalf(int u, int fa, int d, int totLen, int end) {//查找树的半径
    if (u == end) {
        half = min(half, max(totLen - d, d));
        return true;
    }
    for (int i = head[u], v = e[i].to; i; i = e[i].nxt, v = e[i].to)
        if (v != fa && !tag[v]) {
```

```
            if (findHalf(v, u, d + e[i].w, totLen, end)) {
                half = min(half, max(totLen - d, d));
                return true;
            }
        }
    return false;
}

pii getHalf(int s) {//得到直径和半径
    pii t = getStartEnd(s); int totLen = dis[t.second];
    half = INF; findHalf(t.first, 0, 0, totLen, t.second);
    return make_pair(totLen, half);
}

bool solve(int u, int fa, int end) {
    if (u == end) return true;
    for (int i = head[u], v = e[i].to; i; i = e[i].nxt, v = e[i].to)
        if (v != fa) {
            if (solve(v, u, end)) {
                //查找v块中树的直径的一半，查找u块中树的直径的一半
                //答案是所有..中最小的
                tag[v] = 1; pii h1 = getHalf(u); tag[v] = 0;
                tag[u] = 1; pii h2 = getHalf(v); tag[u] = 0;
                int n1 = h1.first, n2 = h2.first, n3 = h1.second + h2.second + e[i].w;
                ans = min(ans, max(max(n1, n2), n3));
                tag[v] = tag[u] = 0;
                return true;
            }
        }
    return false;
}

int main() {
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    cin >> N;
    init();
    for (int i = 1; i < N; i++) {
        int u, v, w; cin >> u >> v >> w;
        add(u, v, w); add(v, u, w);
    }
    ans = INF;
    pii t = getStartEnd(1);
    int start = t.first, end = t.second;
    solve(start, 0, end);
    cout << ans << endl;

    return 0;
}
```

6.8 树的重心

```
//性质
//以重心为根，所有的子树的大小都不超过整个树大小的一半
```

```
//树的重心最多有两个
//树的重心到其他节点的距离是最小的
//把两个树通过一条边相连得到一个新的树，那么新的树的重心在连接原来两个树的重心的路径上
//把一个树添加或删除一个叶子，那么它的重心最多只移动一条边的距离

int maxp[N], siz[N], rt;

void getRt(int u, int fa, int all) {//树的重心
    siz[u] = 1, maxp[u] = 0;
    for (int i = head[u], v = e[i].to; i; i = e[i].nxt, v = e[i].to)
        if (v != fa) {
            getRt(v, u, all);
            siz[u] += siz[v];
            maxp[u] = max(maxp[u], siz[v]);
        }
    maxp[u] = max(maxp[u], all - siz[u]);//fa子树为all - siz[u]
    if (maxp[u] < maxp[rt]) rt = u;
}

int main() {

    maxp[rt = 0] = n; getRt(1, 0, n);

    return 0;
}
```

6.9 树链剖分 (点权)

```
int siz[N], son[N], dep[N], fa[N], top[N];
int nodeOf[N], id[N], cnt;
void dfs(int u, int par) {
    dep[u] = dep[fa[u] = par] + (siz[u] = 1);
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != par) {
            dfs(v, u);
            siz[u] += siz[v];
            if (!son[u] || siz[v] > siz[son[u]])
                son[u] = v;
        }
}
void dfs2(int u, int topf) {
    nodeOf[id[u] = ++cnt] = u, top[u] = topf;
    if (!son[u]) return;
    dfs2(son[u], topf);
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != fa[u] && v != son[u]) dfs2(v, v);
}

int ask(int x, int y) {
    int res = 0;
    while (top[x] != top[y]) {
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        res += query(1, id[top[x]], id[x]);
```

```
        x = fa[top[x]];
    }
    if (dep[x] > dep[y]) swap(x, y);
    return res += query(1, id[x], id[y]);
}

int main() {

    int rt = 1; dfs(rt, 0); dfs2(rt, rt);
    build(1, 1, cnt);

    //x对应线段树上的点id[x]
    //线段树上的点x对应树上点nodeOf[x]

    return;
}
```

6.10 树链剖分 (边权)

```
//维护边权的树剖
//化边权为点权用线段树维护
//此处3种操作:
//1. 单边修改(修改第k条加入的边)
//2. 将(u, v)间的边反转符号
//3. 查询(u, v)间边的sum, max, min

#include <bits/stdc++.h>
#define INF 0x3f3f3f3f
#define lc  u<<1
#define rc  u<<1|1
#define mid (t[u].l+t[u].r)/2
using namespace std;
typedef long long ll;
const int MAX = 2e5 + 10;

int N, M;

struct edge {
    int nxt, to, w, from;//需要额外记录from
} e[MAX << 1];
int head[MAX], tot;//tot为边数*2
void add(int u, int v, int w) { e[++tot] = edge{ head[u], v, w, u }; head[u] = tot; }

//重链剖分
int siz[MAX], son[MAX], dep[MAX], fa[MAX], top[MAX];
int id[MAX], cnt;//cnt为点对应在线段树上的位置
void dfs(int u, int par) {
    dep[u] = dep[fa[u] = par] + (siz[u] = 1);
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != par) {
            dfs(v, u);
            siz[u] += siz[v];
            if (!son[u] || siz[v] > siz[son[u]])
```

```
                son[u] = v;
        }
}
void dfs2(int u, int topf) {
    id[u] = ++cnt, top[u] = topf;
    if (!son[u]) return;
    dfs2(son[u], topf);
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != fa[u] && v != son[u]) dfs2(v, v);
}
int lca(int x, int y) {
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        x = fa[top[x]];
    }
    return dep[x] < dep[y] ? x : y;
}

struct node {//维护最大/小值, 和
    int mx, mn, sum;
    node(int mx = -INF, int mn = INF, int sum = 0): mx(mx), mn(mn), sum(sum) {}
    node merge(const node &rhs) { return node(max(mx, rhs.mx), min(mn, rhs.mn), sum + rhs.sum); }
    void upd() {//符号反转
        sum = -sum;
        mx = -mx, mn = -mn;
        swap(mx, mn);
    }
};
struct SegmentTree {
    int l, r, tag;
    node v;
} t[MAX << 2];
void build(int u, int l, int r) {
    t[u] = SegmentTree{ l, r, 0, node() };
    if (l == r) return;
    build(lc, l, mid); build(rc, mid + 1, r);
}
void push_up(int u) { t[u].v = t[rc].v.merge(t[lc].v); }
void push_down(int u) {
    if (t[u].tag) {
        t[lc].v.upd(), t[lc].tag ^= 1;
        t[rc].v.upd(), t[rc].tag ^= 1;
        t[u].tag = 0;
    }
}
void modify(int u, int p, int k) {//单点修改值
    if (t[u].l == t[u].r) {
        t[u].v = node{ k, k, k };
        return;
    }
    push_down(u);
    if (p <= mid) modify1(lc, p, k);
    else modify1(rc, p, k);
```

```
    push_up(u);
}
void modify2(int u, int ql, int qr) {//反转区间符号
    if (ql <= t[u].l && t[u].r <= qr) {
        t[u].v.upd(), t[u].tag ^= 1;
        return;
    }
    push_down(u);
    if (ql <= mid) modify2(lc, ql, qr);
    if (qr > mid) modify2(rc, ql, qr);
    push_up(u);

}
node query(int u, int ql, int qr) {//区间查询
    if (ql <= t[u].l && t[u].r <= qr) return t[u].v;
    push_down(u);
    if (ql <= mid && qr > mid) return query(lc, ql, qr).merge(query(rc, ql, qr));
    else if (ql <= mid) return query(lc, ql, qr);
    else return query(rc, ql, qr);
}


void update(int x, int y) {//反转(x, y)间所有边的符号
    while (top[x] != top[y]) {
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        modify2(1, id[top[x]], id[x]);
        x = fa[top[x]];
    }
    if (x != y) {
        if (dep[x] > dep[y]) swap(x, y);
        modify2(1, id[son[x]], id[y]);
    }
}

node ask(int x, int y) {//查询(x, y)间的信息
    node res;
    while (top[x] != top[y]) {
        if(dep[top[x]] < dep[top[y]]) swap(x, y);
        res = res.merge(query(1, id[top[x]], id[x]));
        x = fa[top[x]];
    }
    if (x != y) {
        if (dep[x] > dep[y]) swap(x, y);
        res = res.merge(query(1, id[son[x]], id[y]));
    }
    return res;
}

void init() {//多组样例初始化
    tot = cnt = 0;//tot为边数, cnt为树上的点对应线段树的位置
    memset(head, 0, sizeof(head));
    memset(son, 0, sizeof(son));
}
```

```
int main() {

    scanf("%d", &N);
    for (int i = 1; i < N; i++) {
        int u, v, w; scanf("%d%d%d", &u, &v, &w);
        add(u, v, w); add(v, u, w);
    }

    //预处理
    int rt = 1; dfs(rt, 0); dfs2(rt, rt);//重链剖分
    build(1, 1, cnt);
    for (int i = 1; i <= tot; i += 2) {
        int u = e[i].from, v = e[i].to;
        if (dep[u] < dep[v]) swap(u, v);//边对应的点为以rt为根时(u, v)中的儿子节点
        modify1(1, id[u], e[i].w);//化边权为点权
    }

    scanf("%d", &M);
    while (M--) {
        char op[10]; int x, y; scanf("%s%d%d", op, &x, &y);
        if (op[0] == 'C') {//单点修改值
            int u = e[(x << 1) - 1].from, v = e[(x << 1) - 1].to;
            if (dep[u] < dep[v]) swap(u, v);//边对应的点为以rt为根时(u, v)中的儿子节点
            modify1(1, id[u], y);//再找到线段树上对应的点id[u]
        }
        else if (op[0] == 'N') update(x, y);//反转(x, y)间所有边的符号
        else {//查询(x, y)间的信息
            node tmp = ask(x, y);
            int ans = 0;
            if (op[0] == 'S') ans = tmp.sum;
            else if (op[1] == 'A') ans = tmp.mx;
            else ans = tmp.mn;
            printf("%d\n", ans);
        }
    }

    return 0;
}
```

6.11 点分树

```
//建立点分树

struct Grap {
    struct edge {
        int nxt, to;
        ll w;
    } e[MAX << 1];
    int head[MAX], tot;
    void add(int u, int v, int w) { e[++tot] = edge{ head[u], v, w }; head[u] = tot; }
    //G1为原树, G2点分树
    //G2中v为当前点到下一个重心w靠近u的点
} G1, G2;
```

```cpp


int root, rt;
int maxp[MAX], siz[MAX], vis[MAX], fa[MAX];
void getRt(int u, int par, int all) {//求树的重心，将分治复杂度降为LogN
    siz[u] = 1, maxp[u] = 0;
    for (int i = G1.head[u], v = G1.e[i].to; i; i = G1.e[i].nxt, v = G1.e[i].to)
        if (v != par && !vis[v]) {
            getRt(v, u, all);
            siz[u] += siz[v];
            maxp[u] = max(maxp[u], siz[v]);
        }
    maxp[u] = max(maxp[u], all - siz[u]);
    if (maxp[u] < maxp[rt]) rt = u;
}
void rebuild(int u) {//建点分树
    vis[u] = 1;
    for (int i = G1.head[u], v = G1.e[i].to; i; i = G1.e[i].nxt, v = G1.e[i].to)
        if (!vis[v]) {
            maxp[rt = 0] = N; getRt(v, 0, siz[v]);
            G2.add(u, v, rt); fa[rt] = u;
            rebuild(rt);
        }
}


int main() {

    maxp[rt = 0] = N; getRt(1, 0, N); root = rt;
    rebuild(root);


    return 0;
}
```

6.12 点分治

```cpp
//用于解决树上静态路径统计问题
//只需要考虑经过当前树根的两边组成的路径
//经过的树根若是不同一定能在当前树的子树中解决
//如距离为K/小于K的路径有多少....

int maxp[N], siz[N], vis[N], rt;

void getRt(int u, int fa, int all) {//求树的重心
    siz[u] = 1, maxp[u] = 0;
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != fa && !vis[v]) {
            getRt(v, u, all);
            siz[u] += siz[v];
            maxp[u] = max(maxp[u], siz[v]);
        }
    maxp[u] = max(maxp[u], all - siz[u]);
```

```
    if (maxp[u] < maxp[rt]) rt = u;
}

void calc(int u) {//具体题目具体分析
//可以用store记录一下当前树所加的路径

//若路径是有向的，有时候要扫两边(从前往后，从后往前)
//如某括号题[USACO12NOV]Balanced Trees G(https://www.luogu.com.cn/problem/P3060)


//最后结束的时候清空即可
}

void dfs(int u) {
    vis[u] = 1;
    calc(u);
    for (int i = head[u], v; i; i = e[i].nxt)
        if (!vis[v = e[i].to]) {
            maxp[rt = 0] = N; getRt(v, 0, siz[v]);
            dfs(rt);
        }
    vis[u] = 0;//多组清空
}

int main() {

    maxp[rt = 0] = N; getRt(1, 0, N);
    dfs(rt);

    return 0;
}
```

6.13 虚树

```
//虚树可以处理多次询问，并且每次询问只需要树上的K个关键点
//建立的虚树能保证点数 < 2 * K
//如果对虚树做dp，总体复杂度和∑K有关
//考虑dp的时候，需要同时考虑非关键点对答案的影响

int n;

struct edge {
    int nxt, to;
} e[N << 1];
int head[N], tot;
void add(int u, int v) { e[++tot] = edge{ head[u], v }, head[u] = tot;}

int dep[N], fa[N], topfa[N], siz[N], son[N], dfn[N], cnt;
void dfs(int u, int par) {
    dep[u] = dep[fa[u] = par] + (siz[u] = 1);
    int max_son = -1;
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != par) {
```

```
            dfs(v, u);
            siz[u] += siz[v];
            if (max_son < siz[v]) son[u] = v, max_son = siz[v];
        }
}
void dfs2(int u, int topf) {
    topfa[u] = topf, dfn[u] = ++cnt;
    if (!son[u]) return;
    dfs2(son[u], topf);
    for (int i = head[u], v; i; i = e[i].nxt)
        if ((v = e[i].to) != fa[u] && v != son[u]) dfs2(v, v);
}
int LCA(int x, int y) {
    while (topfa[x] != topfa[y]) {
        if (dep[topfa[x]] < dep[topfa[y]]) swap(x, y);
        x = fa[topfa[x]];
    }
    return dep[x] < dep[y] ? x : y;
}
int getDis(int x, int y) { return dep[x] + dep[y] - 2 * dep[LCA(x, y)]; }

//建立虚树
int tag[N];//tag[u] = 1 <=> 关键点
vector<int> g[N];//虚树边
void add_edge(int u, int v) { g[u].push_back(v); }
int st[N], top, rt;//rt为虚树根
void insert(int u) {
    if (top == 1) {
        st[++top] = u;
        return;
    }
    int lca = LCA(u, st[top]);
    if (lca != st[top]) {
        while (top > 1 && dfn[st[top - 1]] >= dfn[lca])
            add_edge(st[top - 1], st[top]), top--;
        if (lca != st[top]) add_edge(lca, st[top]), st[top] = lca;
    }
    st[++top] = u;
}
bool cmp(const int &x, const int &y) { return dfn[x] < dfn[y]; }
void build(vector<int> &v) {
    st[top = 1] = rt;
    sort(v.begin(), v.end(), cmp);
    for (auto &i: v) {
        tag[i] = 1;
        if (i != rt) insert(i);
    }
    while (top > 1) add_edge(st[top - 1], st[top]), top--;
}


void dp(int u) {
    //...
}
```

```
void clear(int u) {//清空虚树边和标记, 也可以和dp合并
    for (auto &v: g[u]) clear(v);
    g[u].clear(); tag[u] = 0;
}
void solve() {
    //...
    dp(rt); clear(rt);
    //...
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i < n; i++) {
        int u, v; scanf("%d%d", &u, &v);
        add(u, v); add(v, u);
    }
    //此处距离为1, 所以用dep替代dis, dis[fa[rt] = 0] = -1
    dep[0] = -1, rt = 1;
    dfs(rt, 0); dfs2(rt, rt);


    int Q; scanf("%d", &Q);
    while (Q--) {
        int K; scanf("%d", &K);//读取关键点
        for (int i = 1; i <= K; i++) scanf("%d", &a[i]);
        //构建虚树
        build(a);
        solve();
    }

    return 0;
}
```

6.14 轻重链划分

```
int siz[N], son[N], dep[N], fa[N], top[N];
int id[N], nodeOf[N], cnt;//划分点
void dfs(int u, int par) {
    dep[u] = dep[fa[u] = par] + (siz[u] = 1);
    for (auto &v: g[u])
        if (v != par) {
            dfs(v, u);
            siz[u] += siz[v];
            if (!son[u] || siz[v] > siz[son[u]])
                son[u] = v;
        }
}

void dfs2(int u, int topf) {
    nodeOf[id[u] = ++cnt] = u, top[u] = topf;
    if (!son[u]) return;
    dfs2(son[u], topf);
    for (auto &v: g[u])
```

```
        if (v != fa[u] && v != son[u]) dfs2(v, v);
}
```