Team Reference Document

天津大学 Tianjin University

神偷 TJU_Thief_Master

Coach	**教　练**
Ruiguo Yu	于瑞国
Mei Yu	喻　梅

Team member	**队　员**
Sheng Cao	曹　圣
Tingle Zhou	周挺乐
Yibo Shi	师一博

ACM-ICPC World Finals 2016

目录

- **vim 配置**

```
set number
set showcmd
set hls
filetype on
filetype indent on
filetype plugin on
colorscheme ron
set ts=4
set sw=4
nmap ,s :w<cr>:sh<cr>
nmap ,/ I//<esc>
nmap ,\ I<del><del><esc>
func! Com()
    exec "w"
    let cmd="!g++"
    let flag="-o %< "
    exec cmd." % ".flag
endfunc
func! Run()
    exec "!./%<"
endfunc
nmap ,g :call Com()<cr>
nmap ,r :call Run()<cr>
nmap ,y mkgg"+yG`k
nmap ,p "+p
```

- **扩充栈空间**

```
extern int main2(void) __asm__ ("main2");
int main2() {
    exit(0);
}
int main() {
    int size = 256 << 20;  // 256 MB
    char *p = (char *)malloc(size) + size;
    __asm__ __volatile__(
            "movq  %0, %%rsp\n"
            "pushq $exit\n"
            "jmp main2\n"
            :: "r"(p));
}
```

- **glibc 内建函数**

```
int __builtin_ffs (unsigned int x); //返回右起第一个 1 的位置，最低位为第 1 位
int __builtin_clz (unsigned int x); //返回左起第一个 1 之前的 0 的个数
int __builtin_ctz (unsigned int x); //返回右起第一个 1 之后的 0 的个数
int __builtin_popcount (unsigned int x); //返回 1 的个数
int __builtin_parity (unsigned int x); //返回 1 的个数的奇偶性，奇数个则返回 1
```

- **快速傅里叶变换(FFT)**

```
void FFT(Complex a[], int n, int oper) {
    for (int i = 1, j = 0; i < n; i++) {
        for (int s = n; j ^= s >>= 1, ~j & s; );
        if (i < j) swap(a[i], a[j]);
    }
    for (int m = 1; m < n; m *= 2) {
        double p = PI / m * oper;
        Complex w = Complex(cos(p), sin(p));
        for (int i = 0; i < n; i += m * 2) {
            Complex unit = 1;
            for (int j = 0; j < m; j++) {
                Complex &x = a[i + j + m], &y = a[i + j], t = unit * x;
                x = y - t;
                y = y + t;
                unit = unit * w;
            }
        }
    }
    if (oper == -1) for (int i = 0; i < n; i++) a[i] = a[i] / n;
}
```

- **数论变换(NTT)**

长度 n 必须为 mod - 1 的约数，否则找不到 n 等分点 w，即 pow(w, n) = 1

```
int w = pow(g, (mod - 1) / (2 * m));
if (oper == -1) w = pow(w, mod - 2);
```

无法直接 NTT，则做三次乘法，然后用 CRT 求出，需要用到**__int128**。

```
ans = (ans1 * mod2 * mod3 * inv1 + ans2 * mod3 * mod1 * inv2 + ans3 * mod1 * mod2 * inv3) % (mod1 * mod2 * mod3);
```

常用大素数

```
mod1 = (31 * 31 << 20) + 1        g1 = 3        inv1 = 346,612,643
mod2 = (17 * 59 << 20) + 1        g2 = 6        inv2 = 408,151,354
mod3 = (3 << 18) + 1              g3 = 10       inv3 = 210,725
```

也可以将多项式拆成$P(x) = P1(x) + \sqrt{mod} * P2(x)$，然后共做 7 次 FFT，需要用到 **long double**

- **多项式的逆元**

$$g_{2n}(x) \equiv 2g_n(x) - g_n(x)^2 f(x) \ \ (mod \ \ x^{2n})$$

- **多项式的除法**

设 n 阶多项式$f_n(x) = q_{n-m}(x)g_m(x) + r_{m-1}(x)$，则$q(x)$为$f(x)/g(x)$的商，$r(x)$为余数

记$h_k'(x) = x^k h_k(1/x)$，即 h'=reverse(h),则$q_{n-m}'(x) = f_n'(x)\left(g_m'(x)\right)^{-1} \ \ (mod \ \ x^{n-m+1})$

- **多项式开根号**

$$g_{2n}(x) \equiv (g_n(x) + f(x)g_n(x)^{-1})/2 \ \ (mod \ \ x^{2n})$$

- **矩阵快速幂**

设递推式为$h_n = \sum_{i=1}^{k} a_i h_{n-i}$，令$X = [h_k, h_{k-1}, \ldots, h_1]^{-1}$，$M^{n-1}X = [h_{n+k-1}, \ldots, h_n]^{-1}$

则$M^{p+k} = \sum_{i=1}^{k} a_i \, M^{p+k-i}$，矩阵相乘变为两个多项式相乘，再将 2k-2…k 的部分合并下去

多项式乘可以使用 FFT，合并操作可以视为求除以多项式$(a_k, a_{k-1}, \ldots, a_1, -1)$的余数

- **在线 FFT**

给向量 a 和向量 b，b[0] = 0，求向量 c 为 a 与 b 的卷积。每次给 a[t]，求 c[t + 1]

```
c[t + 1] += a[t] * b[1];
if (t != 0) for (int m = 1; t % m == 0; m *= 2)
    c[t + 1 .. t + m * 2] += a[t - m .. t - 1] * b[m + 1 .. m * 2];
```

- **线段树**

若下标在[0, nn]范围内，其中 nn = ~0u >> __builtin_clz(n)，即大于等于 n 的最小的 $2^k - 1$，则可以直接用 a[l + r]表示[l, r]这个节点。

- **Splay**

```
struct Node {
    Node *ls, *rs, *f;
```

```
    int a, b, minb; //按照 a 排序，保留 b 的最小值
    void update() {
        minb = b;
        if (ls) minb = min(minb, ls->minb);
        if (rs) minb = min(minb, rs->minb);
    }
    Node *clear(int aa, int bb, Node *ff = NULL) {
        a = aa; b = bb; minb = bb;
        f = ff; ls = rs = NULL;
        return this;
    }
    void rot() { //旋转到他的父亲位置
        Node *x = this, *y = f;
        if (x == y->ls) {
            y->ls = x->rs;
            x->rs = y;
            if (y->ls) y->ls->f = y;
        } else {
            y->rs = x->ls;
            x->ls = y;
            if (y->rs) y->rs->f = y;
        }
        x->f = y->f;
        y->f = x;
        if (x -> f) {
            if (x->f->ls == y) x->f->ls = x;
            else x->f->rs = x;
        }
        y->update();
        x->update();
    }
    int dir() { //判断是父亲的左孩子还是右孩子
        if (this->f) {
            if (this == this->f->ls) return -1;
            else return 1;
        }
        return 0;
    }
};
Node c[N], *root, *cp;
Node *splay(Node *x, Node *f = NULL) { //将 x 提为根
    while (x->f != f) {
        if (x->f->f == f) x->rot();
        else if (x->dir() == x->f->dir()) {
            x->f->rot();
            x->rot();
        } else {
            x->rot();
            x->rot();
        }
    }
    return x;
}
void demo() {
    cp = c;
    root = (cp++)->clear(-INF, INF);
    root->rs = (cp++)->clear(INF, INF, root);
    root->update();
    // 想求第 k 大元素的话，需要维护 size 信息
}
```

- **KD-Tree**

```
int id;
struct Point {
    int x[2];
    friend bool operator < (const Point &a, const Point &b) {
        return a.x[id] < b.x[id];
    }
    friend bool operator <= (const Point &a, const Point &b) {
        return a.x[id] <= b.x[id];
    }
};
struct Node {
    Point l, r, x;
    int v, maxv;
};
Point b[N];
Node c[N * 2];
void updateArea(int x, int y) {
    c[x].l.x[0] = min(c[x].l.x[0], c[y].l.x[0]);
    c[x].l.x[1] = min(c[x].l.x[1], c[y].l.x[1]);
    c[x].r.x[0] = max(c[x].r.x[0], c[y].r.x[0]);
    c[x].r.x[1] = max(c[x].r.x[1], c[y].r.x[1]);
}
void update(int d, int l, int r) {
    int t = (l + r) >> 1, ls = d << 1, rs = ls | 1;
    c[d].maxv = c[d].v;
    if (l < t) c[d].maxv = max(c[d].maxv, c[ls].maxv);
    if (t + 1 < r) c[d].maxv = max(c[d].maxv, c[rs].maxv);
}
bool build(int d, int l, int r, int o) { // c[d] => [l..r)
    if (l >= r) return false;
    int t = (l + r) >> 1, ls = d << 1, rs = ls | 1;
    id = o;
    nth_element(b + l, b + t, b + r);
    c[d].l = c[d].r = c[d].x = b[t];
    c[d].v = 0;
    if (build(ls, l, t, o ^ 1)) updateArea(d, ls);
    if (build(rs, t + 1, r, o ^ 1)) updateArea(d, rs);
    update(d, l, r);
    return true;
}
void set(int d, int l, int r, int o, Point i, int x) {
    if (l >= r) return;
```

```
    if (c[d].x.x[0] == i.x[0] && c[d].x.x[1] == i.x[1]) {
        c[d].v = max(c[d].v, x);
        update(d, l, r);
    } else {
        int t = (l + r) >> 1, ls = d << 1, rs = ls | 1;
        id = o;
        if (i <= c[d].x) set(ls, l, t, o ^ 1, i, x);
        id = o;
        if (c[d].x <= i) set(rs, t + 1, r, o ^ 1, i, x);
        update(d, l, r);
    }
}
int get(int d, int l, int r, int o, Point ll, Point rr) {
    if (l >= r) return 0;
    if (c[d].l.x[0] > rr.x[0] || c[d].l.x[1] > rr.x[1] ||
            c[d].r.x[0] < ll.x[0] || c[d].r.x[1] < ll.x[1])
        return 0;
    if (c[d].l.x[0] >= ll.x[0] && c[d].l.x[1] >= ll.x[1] &&
            c[d].r.x[0] <= rr.x[0] && c[d].r.x[1] <= rr.x[1])
        return c[d].maxv;
    int t = (l + r) >> 1, ls = d << 1, rs = ls | 1;
    int ans = 0;
    if (c[d].x.x[0] >= ll.x[0] && c[d].x.x[1] >= ll.x[1] &&
            c[d].x.x[0] <= rr.x[0] && c[d].x.x[1] <= rr.x[1])
        ans = max(ans, c[d].v);
    ans = max(ans, get(ls, l, t, o ^ 1, ll, rr));
    ans = max(ans, get(rs, t + 1, r, o ^ 1, ll, rr));
    return ans;
}
void demo() {
    // b 中的元素顺序会被打乱，b 的元素范围在[0, n)内
    build(1, 0, n, 0);
    get(1, 0, n, 0, x, y);
    set(1, 0, n, 0, z, dp[i]);
}
```

- **树链剖分**

第一次 dfs 求出 f, h, size, zson，第二次 dfs 求出 top, dfn

- **网络流**

```
struct NetWorkFlow {
    struct Edge {
        int t, f;
        Edge *ne, *p;
        Edge *clear(int tt, int ff, Edge *nee) {
            t = tt; f = ff; ne = nee;
            return this;
        }
    };
    Edge b[M * 2], *p, *fe[N], *cur[N];
    int n, s, t, h[N], vh[N];
    void clear(int nn, int ss, int tt) {
        n = nn; s = ss; t = tt;
        for (int i = 0; i < n; i++) fe[i] = NULL;
        p = b;
    }
    void putedge(int x, int y, int f) {
        fe[x] = (p++)->clear(y, f, fe[x]);
        fe[y] = (p++)->clear(x, 0, fe[y]);
        fe[x]->p = fe[y];
        fe[y]->p = fe[x];
    }
    int aug(int i, int f) {
        if (i == t) return f;
        int minh = n;
        Edge *seg = cur[i], *&j = cur[i];
        do {
            if (j->f) {
                if (h[j->t] + 1 == h[i]) {
                    int tmp = aug(j->t, min(j->f, f));
                    if (tmp) {
                        j->f -= tmp;
                        j->p->f += tmp;
                        return tmp;
                    }
                }
                minh = min(minh, h[j->t] + 1);
                if (h[s] == n) return 0;
            }
            j = j->ne;
            if (j == NULL) j = fe[i];
        } while (j != seg);
        if (!--vh[h[i]]) h[s] = n;
        else ++vh[h[i] = minh];
        return 0;
    }
    int flow() {
        if (fe[s] == NULL) return 0;
        int ans = 0;
        for (int i = 0; i <= n; i++) {
            cur[i] = fe[i];
            h[i] = vh[i] = 0;
        }
        vh[0] = n;
        while (h[s] < n) ans += aug(s, INF);
        return ans;
    }
};
```

- **上下界网络流**

每条边上除了上界还有一个必须满足的下界,其余条件相同。

1. 加入虚拟源点 vs 和虚拟汇点 vt
2. 若边(u,v) 属于 G 那么这条边也属于 D, cap(u,v) = up(u,v) - low(u,v)

3. 对于 G 中的每一个点 v, D 中加入边 (vs,v) ,cap(vs,v) = ed(v)

4. 对于 G 中的每一个点 v, D 中加入边 (v,vt), cap(v,vt) = st(v)

5. 加入边(t,s), cap(t,s) = INF

6. tflow 为所有边的下界的和

7. 求 vs 到 vt 的最大流,若最大流不等于 tflow，则不存在可行流，此问题无解。若相等,恢复原图求最大流。

- **费用流**

```
struct Node {
    int fe, ln, c, le; //ln 上一个点, le 上一条边, c 为 s 到当前点最小花费
    bool d; //是否在队列内
};
struct Edge {
    int f, t, ne, c;
};
Node a[N];
Edge b[M * 2];
int s, t, n, p, cost, flow;
void clear(int nn, int ss, int tt) {
    n = nn; s = ss; t = tt;
    for (int i = 0; i < n; i++) a[i].fe = -1;
    p = cost = flow = 0;
}
void putedge(int x, int y, int f, int c) {
    b[p].ne = a[x].fe; b[p].t = y; b[p].f = f; b[p].c = c; a[x].fe = p++;
    b[p].ne = a[y].fe; b[p].t = x; b[p].f = 0; b[p].c = -c; a[y].fe = p++;
}
inline int add(int &p) {
    int ans = p++;
    if (p == N) p = 0;
    return ans;
}
bool spfa() {
    static int d[N];
    for (int i = 0; i < n; i++) {
        a[i].c = INF;
        a[i].ln = a[i].le = -1;
        a[i].d = false;
    }
    int p = 0, q = 0;
    d[add(q)] = s;
    a[s].d = true;
    a[s].c = 0;
    while (p != q) {
        int u = d[add(p)];
        for (int j = a[u].fe; j != -1; j = b[j].ne) {
            int v = b[j].t;
            if (b[j].f > 0 && b[j].c + a[u].c < a[v].c) {
                a[v].c = a[u].c + b[j].c;
                a[v].ln = u;
                a[v].le = j;
                if (a[v].d == false) {
                    a[v].d = true;
                    d[add(q)] = v;
                }
            }
        }
        a[u].d = false;
    }
    if (a[t].c == INF) return false;
    p = INF;
    q = 0;
    for (int i = t; i != s; i = a[i].ln) {
        d[q++]=i;
        if (p > b[a[i].le].f) p = b[a[i].le].f;
    }
    flow += p;
    for (int i = q - 1; i >= 0; i--) {
        int j = a[d[i]].le;
        cost += b[j].c * p;
        b[j].f -= p;
        b[j ^ 1].f += p;
    }
    return true;
}
void minCostFlow() {
    while (spfa());
}
```

- **平面图最小割**

将其转为对偶图求最短路，对偶图为稀疏图，应使用堆优化的 dijkstra
对于平面图有如下性质：

1. （欧拉公式）如果一个连通的平面图有 n 个点，m 条边和 f 个面，那么 f=m-n+2
2. 每个平面图 G 都有一个与其对偶的平面图 G*
3. G*中的每个点对应 G 中的一个面
4. 对于 G 中的每条边 e，e 属于两个面 f1、f2，加入边(f1*，f2*)。
   如果 e 只属于一个面 f，加入回边(f*，f*)。

平面图 G 与其对偶图 G*之间关系：

1. G 的面数等于 G*的点数，G*的点数等于 G 的面数，G 与 G*边数相同
2. G*中的环对应 G 中的割一一对应

与 S-T 最小割平面图较规则不同，难点在于将一张图的块求出。大体分如下几步进行：

1. 把所有的边都拆成两条有向边，自环删掉。
2. 将每条有向边在另一个图 G‘中用一个点表示。
3. 考察原图中的每个顶点，将所有的与之相连的边极角排序。
4. 遍历每条入边。将其后继设为与之顺时针相邻的出边。也就是在 G’中连一条从这个入边的点到其后继的有向边。注意(S, T)的那条新加边要特殊处理。
5. 在 G'中就是一些不相交的有向环。每个有向环就对应一个区域。找出了所有的区域，我们要的那张图就简单了。
6. 根据对偶图构图，求得 s-t 之间最短路即是对应的最小割

至于“死胡同”问题（构不成平面的边）这样会形成一个特殊的区域，相当于进去死胡同再出来。但是答案不会受到影响，所以直接忽略。

```
Graph b,e;
Point a[N],c[N]; //a 为原始点, c 为原始边
int d[N],next[N],belong[N];
//d 为极角排序数组, next 为下一条边, belong 为左手边的块
int main() {
    int n,m,s,t,i,j,x,y;
    double z;
    scanf("%d%d",&n,&m);
    s=t=0;
    for (i=0;i<n;i++) { //读入原始点
        scanf("%lf%lf",&a[i].x,&a[i].y);
        if (a[i].x<a[s].x) s=i;
        if (a[i].x>a[t].x) t=i;
    }
    b.clear(n);
    c[b.m]=Point(1,0); //添加边框
    b.putedge(t,s,inf);
    c[b.m]=Point(-1,0);
    b.putedge(s,t,inf);
    for (i=0;i<m;i++) { //读入原始边
        scanf("%d%d%lf",&x,&y,&z);
        if (x!=y) {
            c[b.m]=Point(a[y].x-a[x].x,a[y].y-a[x].y);
            b.putedge(x,y,z);
            c[b.m]=Point(a[x].x-a[y].x,a[x].y-a[y].y);
            b.putedge(y,x,z);
        }
    }
    for (i=0;i<n;i++) { //给每个点的原始边排序, 求出下一条边
        int dn=0;
        for (j=b.fe[i];~j;j=b.ne[j]) d[dn++]=j;
        sort(d,d+dn,cmp);
        for (j=1;j<dn;j++) next[d[j]^1]=d[j-1];
        next[d[0]^1]=d[dn-1];
    }
    n=0; //计算每一条边左手边的块号
    for (i=0;i<b.m;i++) belong[i]=-1;
    for (i=0;i<b.m;i++) {
        if (belong[i]==-1) {
            for (j=next[i];j!=i;j=next[j])
                belong[j]=n;
            belong[i]=n++;
        }
    }
    e.clear(n); //构建对偶图
    for (i=0;i<b.m;i+=2) {
        e.putedge(belong[i],belong[i^1],b.v[i]);
        e.putedge(belong[i^1],belong[i],b.v[i]);
    }
    printf("%.4f\n",e.dijkstra(belong[0],belong[1]));
}
```

- **无向图最小割（抄来的）**

```
#define typec int // type of res (or long long)
const typec inf = 0x3f3f3f3f; // max of res
const typec maxw = 1000; // maximum edge weight, g[i][j]=g[j][i]
typec g[V][V], w[V];  int a[V], v[V], na[V];
typec mincut(int n){
    int i, j, pv, zj;    typec best = maxw * n * n;
    for (i = 0; i < n; i++) v[i] = i; // vertex: 0 ~ n-1
    while (n > 1) {
        for (a[v[0]] = 1, i = 1; i < n; i++) {
            a[v[i]] = 0; na[i - 1] = i; w[i] = g[v[0]][v[i]];}
        for (pv = v[0], i = 1; i < n; i++ ) {
            for (zj = -1, j = 1; j < n; j++ )
                if (!a[v[j]] && (zj < 0 || w[j] > w[zj]))  zj = j;
            a[v[zj]] = 1;
            if (i == n - 1) {
                if (best > w[zj]) best = w[zj];
                for (i = 0; i < n; i++)
                    g[v[i]][pv] = g[pv][v[i]]+=g[v[zj]][v[i]];
                v[zj] = v[--n]; break;
            }   pv = v[zj];
            for (j = 1; j < n; j++) if(!a[v[j]]) w[j] += g[v[zj]][v[j]];
    }} return best;}
```

- **有向图最小生成树（抄来的）**

```
const int maxn=1100;  int n,m , g[maxn][maxn] , used[maxn] , pass[maxn] ;
int eg[maxn] , more , queue[maxn];
void combine (int id , int &sum ) {
    int tot = 0 , from , i , j , k ;
    for ( ; id!=0 && !pass[ id ] ; id=eg[id] ) {
        queue[tot++]=id ; pass[id]=1; }
    for ( from=0; from<tot && queue[from]!=id ; from++);
    if  ( from==tot ) return ;
    more = 1 ;
    for ( i=from ; i<tot ; i++) {
        sum+=g[eg[queue[i]]][queue[i]] ;
        if ( i!=from ) {
            used[queue[i]]=1;
            for ( j = 1 ; j <= n ; j++) if ( !used[j] )
                if ( g[queue[i]][j]<g[id][j] ) g[id][j]=g[queue[i]][j] ;}}
    for ( i=1; i<=n ; i++) if ( !used[i] && i!=id ) {
        for ( j=from ; j<tot ; j++){ k=queue[j];
           if (g[i][id]>g[i][k]-g[eg[k]][k]) g[i][id]=g[i][k]-g[eg[k]][k];
    }}}
int mdst( int root ) { // return the total length of MDST
    int i , j , k , sum = 0 ;
    memset ( used , 0 , sizeof ( used ) ) ;
    for ( more =1; more ; ) {
        more = 0 ;   memset (eg,0,sizeof(eg)) ;
        for ( i=1 ; i <= n ; i ++) if ( !used[i] && i!=root ) {
            for ( j=1 , k=0 ; j <= n ; j ++) if ( !used[j] && i!=j )
                if ( k==0 || g[j][i] < g[k][i] ) k=j ;
```

```
        eg[i] = k ;
     } memset(pass,0,sizeof(pass));
for ( i=1;i<=n;i++) if (!used[i] && !pass[i] && i!= root )combine(i,sum);
    }
    for ( i =1; i<=n ; i ++) if ( !used[i] && i!= root ) sum+=g[eg[i]][i];
    return sum ; }
int main(){
   int i,j,k,test,cases; cases=0; scanf("%d%d",&n,&m);
foru(i,1,n) foru(j,1,n) g[i][j]=1000001;
foru(i,1,m) {scanf("%d%d",&j,&k);j++;k++;scanf("%d",&g[j][k]);}
k=mdst(1); if (k>1000000) printf("Possums!\n");  //===no
else printf("%d\n",k);   return 0;}
```

- **强连通分量**

```
void clear(int n) {
    for (int i = 0; i < n; i++) {
        a[i].fe = a[i].scc = a[i].dfn = a[i].low = -1;
        a[i].num = 0;
        a[i].instack = false;
    }
    p = 0;
}
void tarjan(int u) {
    a[u].dfn = a[u].low = idx++;
    a[u].instack = true;
    stk[p++] = u;
    for (int j = a[u].fe; j != -1; j = b[j].ne) {
        int v = b[j].t;
        if (a[v].dfn == 0) {
            tarjan(v);
            a[u].low = min(a[u].low, a[v].low);
        } else if (a[v].instack) {
            a[u].low = min(a[u].low, a[v].dfn);
        }
    }
    if (a[u].low == a[u].dfn) {
        while (stk[--p] != u) {
            a[stk[p]].instack = false;
            a[stk[p]].scc = u;
            a[u].num++;
        }
        a[u].instack = false;
        a[u].scc = u;
        a[u].num++;
    }
}
void demo() {
    idx = p = 0;
    for (int i = 0; i < n; i++) if (a[i].dfn == -1) tarjan(i);
}
```

- **割点、割边**

```
void tarjan(int i, int f) {
    a[i].dfn = a[i].low = idx++;
    for (int j = a[i].fe; j != -1; j = b[j].ne) {
        if (b[j].vis) continue;
        b[j].vis = true;
        b[j ^ 1].vis = true;
        if (a[b[j].t].dfn == -1) {
            tarjan(b[j].t, i);
            a[i].low = min(a[i].low, a[b[j].t].low);
            //if (a[b[j].t].low > a[i].dfn) b[j]是割边
            //if (f == -1) 根节点的几个儿子互不联通
            //else if (a[b[j].t].low >= a[i].dfn) 去掉 i 后，b[j].t 与 f 不连通
        } else a[i].low = min(a[i].low, a[b[j].t].dfn);
    }
}
void demo() {
    for (int i = 0; i < n; i++) a[i].low = a[i].dfn = -1;
    for (int i = 0; i < bp; i++) b[i].vis = false;
    idx = 0;
    for (int i = 0; i < n; i++) if (a[i].dfn == -1) tarjan(i, -1);
}
```

- **二维几何基础**

```
double dmul(Point a, Point b) { //点积
    return a.x * b.x + a.y * b.y;
}
double xmul(Point a, Point b) { //叉积，大于 0 表示 b 在 a 的逆时针方向
    return a.x * b.y - a.y * b.x;
}
double xmul(Point a, Point b, Point c) { //a->b 与 a->c 的叉积
    return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}
int quadrant(Point a) {//象限号，原点为 0，从 x 轴开始顺时针为 1 至 8，第四象限为 8
    const int ans[3][3] = {{4, 3, 2}, {5, 0, 1}, {6, 7, 8}};
    return ans[1 - sig(a.y)][sig(a.x) + 1];
}
bool cmpp(Point a, Point b) { //极角排序
    int p = quadrant(a), q = quadrant(b);
    if (p != q) return p < q;
    double x = xmul(a, b);
    if (sig(x)) return x > 0;
    return square(a) < square(b);
}
Point rot(Point a) { //逆时针旋转 90 度
    return Point(-a.y, a.x);
}
double alpha(Point a, Point b) { //向量 b 在向量 a 的逆时针多少度
    return atan2(xmul(a, b), dmul(a, b));
}
Point rot(Point a, double p, Point b = Point(0,0)) { //点 a 绕点 b 逆时针旋转 p
    Point t1 = a - b, t2 = Point(cos(p), sin(p));
    return b + Point(t1.x * t2.x - t1.y * t2.y, t1.x * t2.y + t1.y * t2.x);
}
```

```
void regular(Line a) { //整理直线，使得直线的极角属于范围(-PI/2,PI/2]
    int x = sig(a.p.x - a.q.x), y = sig(a.p.y - a.q.y);
    if (x == 1 || (x == 0 && y == 1)) swap(a.p, a.q);
}
bool isParallel(Line a, Line b) { //判断是否平行
    return sig(xmul(a.q - a.p, b.q - b.p)) == 0;
}
bool inSameLine(Line a, Line b) { //判断是否共线，要求 ab 平行
    return sig(xmul(a.q - a.p, a.q - b.p)) == 0;
}
bool isVertical(Line a, Line b) { //判断是否垂直
    return sig(dmul(a.q - a.p, b.q - b.p)) == 0;
}
Point cross(Line a, Line b) { //直线 a 与 b 的交点，要求 ab 不平行
    double t1 = xmul(a.p, a.q, b.p), t2 = -xmul(a.p, a.q, b.q);
    return (b.p * t2 + b.q * t1) / (t1 + t2);
}
bool cmpLine(Line a, Line b) { //按直线的方向排序
    return cmpp(a.q - a.p, b.q - b.p);
}
bool inLine(Point a, Line b) { //判断点是否在直线上
    return sig(xmul(a - b.p, a - b.q)) == 0;
}
Point projection(Point a, Line b) { //点在直线上的投影
    Point tmp = b.q - b.p;
    return b.p + tmp * dmul(a - b.p, tmp) / square(tmp);
}
double disLine(Point a, Line b) { //点到直线的距离
    return abs(xmul(a - b.p, a - b.q)) / abs(b.p - b.q);
}
double disSeg(Point a, Line b) { //点到线段距离
    Point x = b.q - b.p, y = a - b.p, z = a - b.q;
    if (sig(dmul(x, y)) <= 0) return abs(y);
    if (sig(dmul(x, z)) >= 0) return abs(z);
    return abs(xmul(y, z)) / abs(x);
}
bool inSeg(Point a, Line b) { //判断点是否在线段上，端点返回 true
    if (sig(xmul(a - b.p, a - b.q)) != 0) return false; //不在直线上
    if (sig(dmul(a - b.p, a - b.q)) > 0) return false;
    //不在线段内，若为端点，则等于 0
    return true;
}
Line perpBis(Line a) { //线段 a 的中垂线
    Point t1 = (a.p + a.q) / 2, t2 = rot(a.q - a.p);
    return Line(t1, t1 + t2);
}
bool hasCross(Line a, Line&b) { //线段之间是否有交点，重合和端点相交返回 false
    if (sig(xmul(a.p, a.q, b.p)) * sig(xmul(a.p, a.q, b.q)) >= 0)
        return false; //b 的两个端点在 a 的同侧
    if (sig(xmul(b.p, b.q, a.p)) * sig(xmul(b.p, b.q, a.q)) >= 0)
        return false; //a 的两个端点在 b 的同侧
    return true;
}
```

- **圆相关**

```
int cross(Line a, Circle b, Point &ans1, Point &ans2) { //求出直线与圆的交点
    double t1 = a.q.x - a.p.x, t2 = a.p.x - b.c.x;
    double t3 = a.q.y - a.p.y, t4 = a.p.y - b.c.y;
    double k1 = t1 * t1 + t3 * t3, k2 = 2 * (t1 * t2 + t3 * t4);
    double k3 = t2 * t2 + t4 * t4 - b.r * b.r;
    double d = k2 * k2 – 4 * k1 * k3;
    if (sig(d) < 0) return 0; //无交点
    if (sig(d) == 0) d = 0; else d = sqrt(d);
    ans1 = a.p + (a.q – a.p) * (-k2 + d) / (2 * k1);
    ans2 = a.p + (a.q - a.p) * (-k2 - d) / (2 * k1);
    return sig(d) + 1;
}
int cross(Circle a, Circle b, Point &ans1, Point &ans2) { //求出两圆的交点
    double d = abs(a.c - b.c);
    if (sig(d) == 0)
        if (sig(a.r - b.r) == 0) return -1; //重合，无数个交点
        else return 0;
    if (sig(a.r + b.r - d) < 0) return 0; //相离
    if (sig(abs(a.r - b.r) - d) > 0) return 0; //内含
    double p1 = alpha(b.c-a.c);
    double p2 = acos(legal((a.r * a.r + d * d - b.r * b.r) / (2 * a.r * d)));
    ans1 = get(a, p1 + p2);
    ans2 = get(a, p1 - p2);
    return sig(p2) + 1;
}
int tangent(Point a, Circle b, Point &ans1, Point &ans2) { //点与圆的两个切点
    double d = abs(a - b.c);
    if (sig(d - b.r) < 0) return 0; //在圆内，无交点
    double al1 = alpha(a - b.c);
    double al2 = acos(legal(b.r / d));
    ans1 = get(b, al1 + al2);
    ans2 = get(b, al1 - al2);
    return sig(al2) + 1;
}
int tangent(Circle a, Circle b, Line &ans1, Line &ans2,
        Line &ans3, Line &ans4) { //求出两圆的共切线，返回公切线个数
    if (a.r < b.r) swap(a, b);
    Point t = b.c - a.c;
    double d = abs(t), al1 = alpha(t), al2;
    if (a.c == b.c && sig(a.r - b.r) == 0) return -1; //重合，无数条公切线
    if (sig(a.r - b.r - d) > 0) return 0; //内含
    if (sig(a.r - b.r - d) == 0) { //内切
        Point p = get(a, al1);
        ans1 = ans2 = Line(p, p + rot(t));
        return 1;
    }
    al2 = acos(legal((a.r - b.r) / d));
    ans1 = Line(get(a, al1 + al2), get(b, al1 + al2));
    ans2 = Line(get(a, al1 - al2), get(b, al1 - al2)); //两条外公切线
```

```
    if (sig(a.r + b.r - d) > 0) return 2; //相交
    if (sig(a.r + b.r - d) == 0) { //外切
        Point p = get(a, al1);
        ans3 = ans4 = Line(p, p + rot(t));
        return 3;
    }
    al2 = acos(legal((a.r + b.r) / d));
    ans3 = Line(get(a, al1 + al2), get(b, al1 + PI + al2));
    ans4 = Line(get(a, al1 - al2), get(b, al1 + PI - al2));
    return 4; //相离
}
```

- **凸包**

```
int convexHull(Point a[], int n, Point ans[]) { //ans[]的大小要为 N+1
    sort(a, a + n, cmpxy);
    n = unique(a, a + n) - a;
    if (n == 1) ans[0] = a[0];
    if (n <= 1) return n;
    int m = 0;
    for (int i = 0; i < n; i++) { //下半圆
        while (m > 1 && sig(xmul(ans[m - 2], ans[m - 1], a[i])) <= 0) m--;
        //去掉等号则允许边上的点
        ans[m++] = a[i];
    }
    int mm = m;
    for (int i = n - 2; i >= 0; i--) { //上半圆
        while (m > mm && sig(xmul(ans[m - 2], ans[m - 1], a[i])) <=0 ) m--;
        //去掉等号则允许边上的点
        ans[m++] = a[i];
    }
    return m - 1;
}
int convexCut(Point a[], int n, Line b, Point ans[]) { //被直线 b 切割，留左手
    int m = 0;
    for (int i = 0; i < n; i++) {
        Point &p = a[i], &q = a[(i + 1) % n];
        int t1 = sig(xmul(b.p, b.q, p)), t2 = sig(xmul(b.p, b.q, q));
        if (t1 >= 0) ans[m++] = p;
        if (t1 * t2 < 0) ans[m++] = cross(Line(p, q), b);
    }
    m = unique(ans, ans + m) - ans; //一条线段被切割时会多出一个点来
    return m;
}
double convexDiameter(Point a[], int n) {
//旋转卡壳求凸包上的最远点对，要求凸包为逆时针，且边上没有点
    if (n == 1) return 0;
    if (n == 2) return abs(a[0] - a[1]);
    double ans = 0;
    for (int i = 0, j = 1; i < n; i++) {
        int ii = (i + 1) % n, jj = j, t;
        do { //求出所有的对踵点，可能有重复
            j = jj;
            jj = (j + 1) % n;
            t = sig(xmul(a[ii] - a[i], a[jj] - a[j]));
            if (t <= 0) ans = max(ans, square(a[i] - a[j])); //对踵点
            if (t == 0) ans = max(ans, square(a[i] - a[jj])); //对踵点
        } while (t > 0);
    }
    return sqrt(ans);
}
bool inPolygon(Point a[], int n, Point b) { //点在多边形内，边界返回 false
    int ans = 0;
    for (int i = 0; i < n; i++) {
        Point &p = a[i], &q = a[(i + 1) % n];
        if (inSeg(b, Line(p, q))) return false; //判断边界返回 false
        int k = sig(xmul(q - p, b - p));
        if (k > 0 && p.y <= b.y + eps && q.y > b.y + eps) ans++;
        if (k < 0 && q.y <= b.y + eps && p.y > b.y + eps) ans--;
    }
    return ans;
}
int halfPlaneIntersection(Line a[], int n, Point ans[]) {
//半平面交，保留每条直线的左手边，求出的凸包在 ans 中，若凸包已退化，则返回 0
//要求必须有边界，若无边界则手动添加边界
    sort(a, a + n, cmpLine);
    static Line b[N];
    static Point c[N]; //c[i]为 b[i]与 b[i+1]的交点
    int l = 0, r = 0;
    b[0] = a[0];
    for (int i = 1; i < n; i++) {
        while (l < r && sig(xmul(a[i].p, a[i].q, c[r - 1])) <= 0) r--;
        while (l < r && sig(xmul(a[i].p, a[i].q, c[l])) <= 0) l++;
        b[++r] = a[i];
        if (sig(xmul(a[i].q - a[i].p, b[r - 1].q - b[r - 1].p)) == 0) {
            r--;
            if (sig(xmul(b[r].p, b[r].q, a[i].p)) > 0) b[r] = a[i];
        }
        if (l < r) c[r - 1] = cross(b[r], b[r - 1]);
    }
    while (l < r && sig(xmul(b[l].p, b[l].q, c[r - 1])) <= 0) r--;
    if (r – l <= 1) return 0; //凸包已退化
    c[r] = cross(b[l], b[r]);
    int m = 0;
    for (int i = l; i <= r; i++) ans[m++] = c[i];
    return m;
}
```

- **三角形**

中线：$M_a = (\sqrt{2(b^2+c^2)} - a^2)/2 = (\sqrt{b^2+c^2+2bc\cos A})/2$

角平分线：$T_a = (\sqrt{bc((b+c)^2-a^2)})/(b+c) = (2bc\cos(A/2))/(b+c)$

内切圆半径：$r = S/P = 4R\sin(A/2)\sin(B/2)\sin(C/2) = a\sin(B/2)\sin(C/2)/\sin((B+C)/2)$

$= \sqrt{(P-a)(P-b)(P-c)/P} = P\tan(A/2)\tan(B/2)\tan(C/2)$

外切圆半径：$R=\frac{abc}{4S}=a/(2\sin(A))=b/(2\sin(B))=c/(2\sin(C))$

内心：$P=(aA+bB+cC)/(a+b+c)$

外心：$d=2|\vec{c}\times\vec{a}|^2, \alpha=-\frac{(\vec{a}\cdot\vec{a})(\vec{b}\cdot\vec{c})}{d}, \beta=\frac{(\vec{b}\cdot\vec{b})(\vec{c}\cdot\vec{a})}{d}, \gamma=\frac{(\vec{c}\cdot\vec{c})(\vec{a}\cdot\vec{b})}{d}, P=\alpha A+\beta B+\gamma C$

垂心：$\alpha=(\vec{a}\cdot\vec{b})(\vec{a}\cdot\vec{c}), \beta=(\vec{b}\cdot\vec{c})(\vec{b}\cdot\vec{a}), \gamma=(\vec{c}\cdot\vec{a})(\vec{c}\cdot\vec{b}), P=(\alpha A+\beta B+\gamma C)/(\alpha+\beta+\gamma)$

- **平面定理**

多边形重心：三角剖分后，以面积为权值求各个重心的加权平均

皮克定理：格点多边形面积 = 内部格点数 + 边上格点数 / 2 - 1

欧拉定理：对于一个平面图/凸多面体，顶点个数 + 面数 - 边数 = 2

- **高维球**

对于半径为 1 的高维球，已知：$V_2=2\pi$，$S_2=\pi$，$V_3=4\pi$，$S_3=\frac{4}{3}\pi$

递推式：$V_n=\frac{S_n}{n}$，$S_n=2\pi V_{n-2}$

- **复数**

```
typedef complex<double> Point;
double dmul(const Point &a, const Point &b) {
    return real(conj(a) * b);
}
double xmul(const Point &a, const Point &b) {
    return imag(conj(a) * b);
}
Point rot(const Point &a, const double &p, const Point &b = Point(0, 0)) {
    //点 a 绕点 b 逆时针旋转 p，exp(Point(0, p))为模长为 1，与 x 轴夹角为 p 的向量
    return (a - b) * exp(Point(0, p)) + b;
}
Point reflect(const Point &p, const Point &a, const Point &b) {
    //点 p 关于直线 ab 的镜像点
    return conj((p - a) / (b - a)) * (b - a) + a;
}
```

- **扩展欧几里得**

```
void exgcd(int a, int b, int &x, int &y) { // 求解 ax + by = gcd(a, b)
    if (b == 0) {
        x = 1; y = 0;
    } else {
        int k = a / b, c = a % b, p, q;
        exgcd(b, c, p, q);
        x = q; y = -k * q + p;
    }
}
// 对于乘法逆元，求 ax + mody = 1 即可，x 即为 a 的逆元，x 在[-mod, mod)范围内
```

- **线性求乘法逆元（mod 为质数）**

```
inv[1] = 1;
inv[i] = mod -(long long)mod / i * inv[mod % i] % mod;
```

- **线性筛素数**

```
for (int i = 2; i < N; i++) {
    if (mpf[i] == 0) mpf[i] = prime[pn++] = i;
    for (int j = 0; j < pn && i * prime[j] < N && prime[j] <= mpf[i]; j++)
        mpf[i * prime[j]] = prime[j];
}
```

- **高斯消元**

每列留下绝对值最大的元素，可以减少精度丢失

- **欧拉函数**

$\varphi(n)$为小于等于n中与n互质的数的个数，若n,m互质，则$\varphi(nm)=\varphi(n)\varphi(m)$

- **大素数判定**

```
const int S = 20; //S 越大，判错概率越小
bool Miller_Rabin(long long p) { //p 是素数返回 true
    if (p < 2) return false;
    if (p == 2) return true;
    if ((p & 1) == 0) return false;
    long long x = p - 1, t = 0;
    while ((x & 1) == 0) x >>= 1, t++;
    for (int i = 0; i < S; i++) {
        long long a = rand() % (p - 1) + 1;
        if (notpri(a, p, x, t)) return false;
    }
    return true;
}
```

- **大数分解**

```
long long mult(long long a, long long b, long long p);//a * b mod p
long long pow(long long a, long long b, long long p); //a ^ b mod p
bool notpri(long long a, long long p, long long x, long long t) {
    long long res = pow(a, x, p);
    long long last = res;
    for (int i = 1; i <= t; i++) {
        res = mult(res, res, p);
        if (res == 1 && last != 1 && last != p - 1) return true;
        last = res;
    }
    if (res != 1) return true;
    return false;
}
vector<long long> div;
long long Pollard_rho(long long x, long long c) {
    long long i = 1, k = 2, x0 = rand() % x, y = x0;
    while (1) {
        i++; x0 = (mul(x0, x0, x) + c) % x;
        long long d = gcd(y - x0, x);
        if (d != 1 && d != x) return d;
        if (y == x0) return x;
        if (i == k) y = x0, k += k;
    }
```

```
}
//质因子存在 div 中，不是有序的
void workfac(long long n) {
    if (Miller_Rabin(n)) {
        div.push_back(n);
        return ;
    }
    long long p = n;
    while (p == n) {
        p = Pollard_rho(p, rand() % (n - 1) + 1);
    }
    workfac(p);
    workfac(n / p);
}
```

- **莫比乌斯反演**

```
int mp[N], pri[M], mu[N], len;
void Mobius() {
    memset(mp, 0, sizeof(mp));
    for (int i = 2; i < N; i++) {
        if (!mp[i]) {
            mp[i] = i; mu[i] = -1; pri[len++] = i;
        }
        for (int j = 0; j < len && pri[j] * i < N; j++) {
            mp[i * pri[j]] = pri[j];
            if (i % pri[j] == 0) {
                mu[i * pri[j]] = 0; break;
            }
            mu[i * pri[j]] = -mu[i];
        }
    }
}
```

- **波利亚**

设$G=\{\pi_1,\pi_2,\cdots,\pi_k\}$是$X=\{a_1,a_2,\cdots,a_n\}$上的一个置换群，用$m$种颜色对$X$中的元素进行染色，

那么不同的个数为$\frac{1}{|G|}\sum_{i=1}^{k} m^{C(\pi_i)}$，其中$C(\pi_i)$为$\pi_i$的循环节的个数

- **Pell 方程**

$x^2-ny^2=1$，其中n不是完全平方数

```
unsigned long long A, B, p[N], q[N], a[N], g[N], h[N];
void pell(int n) {
    p[1] = q[0] = h[1] = 1, p[0] = q[1] = g[1] = 0;
    a[2] = (int)(floor(sqrt(n) + 1e-7));
    for (int i = 2; ; i++) {
        g[i] = -g[i - 1] + a[i] * h[i - 1];
        h[i] = (n - 1llu * g[i] * g[i]) / h[i - 1];
        a[i + 1] = (g[i] + a[2]) / h[i];
        p[i] = a[i] * p[i - 1] + p[i - 2];
        q[i] = a[i] * q[i - 1] + q[i - 2];
        if (1llu * p[i] * p[i] - 1llu * n * q[i] * q[i] == 1) {
            A = p[i]; B = q[i];
            break;
        }
    }
}
```

- **Matrix-Tree 定理**

给定一个无向图G，求它的生成树的个数$T(G)$

令矩阵$D[G]=\begin{cases}D[i][j]=0 & (i\neq j)\\ D[i][j]=v_i\text{的度数} & (i=j)\end{cases}$

令矩阵$A[G]=\begin{cases}A[i][j]=1 & (v_i,v_j\text{有边})\\ A[i][j]=0 & (v_i,v_j\text{无边})\end{cases}$

令矩阵$C[G]=D[G]-A[G]$，那么$T(G)=C[G]$任何一个$n-1$阶主子式的行列式的值

- **Prufer 编码**

一棵标号树的 Pufer 编码规则如下：找到标号最小的叶子节点，输出与它相邻的节点到 prufer 序列，将该叶子节点删去，反复操作，直至剩余 2 个节点。

由 Pufer 编码生成树：任何一个 prufer 序列可以唯一对应到一棵有标号的树，首先标记所有节点为未删除，依次扫描 prufer 序列中的数，比如当前扫描到第 k 个数 u，说明有一个叶子节点连到 u，并在当前操作中被删除，找一个标号最小的未被标记为删除的且在 prufer 序列第 k 个位置后未出现过的节点 v，在 u,v 间连边并将 v 删除，反复操作，最后剩两个节点未被标记为删除，在它们之间连边，这样得到的一个图含有 n-1 条边则是一棵树

- **一些计数问题**

有标号有根树：n^{n-1}

有标号无根树：n^{n-2}

无标号二叉树：$C(2n,n)/(n+1)$

标号为 k 的点度为 vk 的无根树：$(n-2)!/\prod(v_k-1)!$

无标号毛毛虫（除了直径以外的点都是悬挂点的树）：$2^{n-4}+2^{\lfloor(n-4)/2\rfloor}$

有标号 DAG，复杂度$O(N^2)$，$F(n,S)$为S中的顶点度为0的 DAG 个数，

则$-F(n,\emptyset)=\sum_{1\le k\le n}(-1)^{k+1}2^{k(n-k)}C(n,k)F(n-k,\emptyset)$

- **差分序列**

```
F(n) = c0 * C(n, 0) + c1 * C(n, 1) + ... + cp * C(n, p)
S(n) = F(0) + F(1) + ... + F(n)
     = c0 * C(n + 1, 1) + c1 * (n + 1, 2) + ... + cp * C(n + 1, p + 1)
```

- **Java 大数**

```
import java.math.BigInteger;
import java.util.Scanner;
Scanner in = new Scanner(System.in);
```

```
int n = in.nextInt();
BigInteger a = in.nextBigInteger();
System.out.println(a);
```

- **常用大素数**

1,000,000,007 100,000,007 10,000,019 1,000,003 100,003 10,007 1,019 103

- **约数个数**

范围	个数	范围	个数	范围	个数	范围	个数	范围	个数
10^3	32	10^5	128	10^7	448	10^9	1,344	**int32**	1,600
10^4	64	10^6	240	10^8	768	10^{18}	103,680	**int64**	161,280

- **质数个数**

范围	个数	范围	个数	范围	个数
10^3	168	10^5	9,592	10^7	664,579
10^4	1,229	10^6	78,498	10^8	5,761,455

- **浮点数求和(Kahan Summation)**

```
double ans = 0, c = 0;
void add(double x) {
    double y = x - c;
    double t = ans + y;
    c = (t - ans) - y;
    ans = t;
}
```

- **Simpson**

```
double simpson(const T&f,double a,double b,int n){
    const double h=(b-a)/n; double ans=f(a)+f(b);
    for (int i=1;i<n;i+=2) ans+=4*f(a+i*h);
    for (int i=2;i<n;i+=2) ans+=2*f(a+i*h);
    return ans*h/3;
}
printf("%lf\n",simpson(test,0,1,(int)1e6);
```

- **二次剩余**

```
// a*x^2+b*x+c==0 (mod P) 求 0..P-1 的根
int pDiv2,P,a,b,c,Pb,d;
inline int calc(int x,int Time){
    if (!Time) return 1;  int tmp=calc(x,Time/2);
    tmp=(long long)tmp*tmp%P;
    if (Time&1) tmp=(long long)tmp*x%P;   return tmp;
}
inline int rev(int x){ if (!x) return 0;  return calc(x,P-2);}
inline void Compute(){
    while (1) { b=rand()%(P-2)+2;  if (calc(b,pDiv2)+1==P) return; }
}
int main(){
    srand(time(0)^312314);  int T;
    for (scanf("%d",&T);T;--T) {
        scanf("%d%d%d%d",&a,&b,&c,&P);
        if (P==2)  {
            int cnt=0; for (int i=0;i<2;++i) if ((a*i*i+b*i+c)%P==0) ++cnt;
            printf("%d",cnt);
            for (int i=0;i<2;++i) if ((a*i*i+b*i+c)%P==0) printf(" %d",i);
            puts("");
        }else  {
            int delta=(long long)b*rev(a)*rev(2)%P;
            a=(long long)c*rev(a)%P-sqr( (long long)delta )%P;
            a%=P;a+=P;a%=P;  a=P-a;a%=P;  pDiv2=P/2;
            if (calc(a,pDiv2)+1==P) puts("0");
            else {
                int t=0,h=pDiv2; while (!(h%2)) ++t,h/=2;
                int root=calc(a,h/2);
                if (t>0) {  Compute();  Pb=calc(b,h); }
                for (int i=1;i<=t;++i) {
                    d=(long long)root*root*a%P;
                    for (int j=1;j<=t-i;++j)  d=(long long)d*d%P;
                    if (d+1==P)  root=(long long)root*Pb%P;
                    Pb=(long long)Pb*Pb%P;
                }
                root=(long long)a*root%P;
                int root1=P-root; root-=delta;
                root%=P; if (root<0) root+=P;
                root1-=delta; root1%=P;  if (root1<0) root1+=P;
                if (root>root1) { t=root;root=root1;root1=t;  }
                if (root==root1) printf("1 %d\n",root);
                else printf("2 %d %d\n",root,root1);
    }}}return 0;}
```

- **Hash**

```
int get(int l, int r) {
    int tmp = (long long)h[l - 1] * p[r - l + 1] % mod;
    return (h[r] - tmp + mod) % mod;
}
bool equal(int a, int b, int l) { //a 开始的和 b 开始的长为 l 的字符串是否相同
    if (l == 0) return true;
    return get(a, a + l - 1) == get(b, b + l - 1);
}
void init() {
    p[0] = 1;
    for (int i = 1; i < N; i++) p[i] = (long long)p[i - 1] * 26 % mod;
    h[0] = 0;
    for (int i = 1; i <= n; i++)
        h[i] = ((long long)h[i - 1] * 26 + s[i] - 'a') % mod;
}
```

- **KMP**

```
//next[i] == j 表示满足以下条件的最大的 j
//s2[0..j-1]与 s2[i-j..i-1]相同, 且 s2[j]与 s2[i]不同, 若不存在, 则 next[i] = -1
int kmp(char s1[], char s2[], int next[]) {
    int i, j = 0, k = -1, ans = 0;
    next[0] = -1;
```

```
    while (s2[j]!='\0') {
        while (k != -1 && s2[j] != s2[k]) k = next[k];
        j++; k++;
        if (s2[j] != s2[k]) next[j] = k;
        else next[j] = next[k];
    }
    i = j = 0;
    while (s1[i] != '\0') {
        if (j != -1 && s2[j] == '\0') {
            ans++;
            j = 0;
            // 如果要求可重复的 s2，则不令 j=0，而是和平时一样处理 j
        } else {
            while (j != -1 && s1[i] != s2[j]) j = next[j];
            i++; j++;
        }
    }
    if (s2[j] == '\0') ans++;
    return ans;
}
```

- exKMP

```
void exkmp(char s1[], char s2[], int next[], int ex[]) {
    int i, j, p;
    for (i = 0, j = 0, p = -1; s1[i] != '\0'; i++, j++, p--) {
        if (p == -1) {
            j = 0;
            do p++; while (s1[i + p] != '\0' && s1[i + p] == s2[j + p]);
            ex[i] = p;
        } else if (next[j] < p) ex[i] = next[j];
        else if (next[j] > p) ex[i] = p;
        else {
            j = 0;
            while (s1[i + p] != '\0' && s1[i + p] == s2[j + p]) p++;
            ex[i] = p;
        }
    }
    ex[i] = 0;
}
void demo() {
    nxt[0] = 0;
    exkmp(s2 + 1, s2, nxt, nxt + 1);
    exkmp(s, s2, nxt, ex);
}
```

- Manacher

```
// s[i] + a[i] == s[i] - a[i]
void manacher(char s[], int ls, int a[]) {
    a[0] = 0;
    for (int i = 0, j; i < ls; i = j) {
        while (i - a[i] > 0 && s[i + a[i] + 1] == s[i - a[i] - 1]) a[i]++;
        for (j = i + 1;
                j <= i + a[i] && i - a[i] != i + i - j - a[i + i - j]; j++)
            a[j] = min(a[i + i - j], i + a[i] - j);
        a[j] = max(i + a[i] - j, 0);
    }
}
void demo() {
    ls = strlen(s);
    for (int i = 0; i < ls; i++) {
        ss[i + i + 1] = s[i];
        ss[i + i + 2] = '\0';
    }
    ls = ls * 2 + 1;
    ss[0] = ss[ls] = '\0';
    manacher(ss, ls, a);
}
```

- AC 自动机

```
struct Node {
    Node *ch[K], *fail;
    int match;
    Node *clear() {
        memset(this,0,sizeof(Node));
        return this;
    }
};
Node *que[N];
Node a[N], *root, *superRoot, *cur;
void clear() {
    cur = a;
    superRoot = (cur++)->clear();
    root = (cur++)->clear();
    root->fail = superRoot;
    for (int i = 0; i < K; i++) superRoot->ch[i] = root;
    superRoot->match = -1;
}
void insert(char *s) {
    Node *t = root;
    for (; *s != '\0'; s++) {
        int x = *s - 'a';
        if (t->ch[x] == NULL) t->ch[x] = (cur++)->clear();
        t = t->ch[x];
    }
    t->match++;
}
void build() {
    int p = 0, q = 0;
    que[q++] = root;
    while (p != q) {
        Node *t = que[p++];
        for (int i = 0; i < K; i++) {
            if (t->ch[i]) {
                t->ch[i]->fail = t->fail->ch[i];
                que[q++] = t->ch[i];
```

```
            } else
                t->ch[i] = t->fail->ch[i];
        }
    }
}
int run(char *s) {
    int ans = 0;
    Node *t = root;
    for (; *s; s++) {
        int x = *s - 'a';
        t = t->ch[x];
        for (Node *u = t; u->match != -1; u = u->fail) {
            ans += u->match;
            u->match = -1;
        }
    }
    return ans;
}
```

- 后缀自动机

```
struct Node {
    Node *f, *son[K];
    int maxl, in, v, num;
    Node *clear(int l = 0) {
        memset(this, 0, sizeof(Node));
        maxl = l;
        return this;
    }
};
Node a[2 * N], *ap;
Node *tail, *init;
void clear() {
    ap = a;
    tail = init = (ap++)->clear();
}
void push_back(char c) {
    int x = (c == '#')? 10 : c - '0';
    Node *i = tail;
    tail = (ap++)->clear(i->maxl + 1);
    for (; i != NULL && i->son[x] == NULL; i = i->f) i->son[x] = tail;
    if (i == NULL) tail->f = init;
    else if (i->maxl + 1 == i->son[x]->maxl) tail->f = i->son[x];
    else {
        Node *p = (ap++)->clear(), *q = i->son[x];
        *p = *q;
        q->f = tail->f = p;
        p->maxl = i->maxl + 1;
        for (; i != NULL && i->son[x] == q; i = i->f) i->son[x] = p;
    }
}
int ws[N * 2], wv[N * 2];
void sort(int n, Node a[], int ws[], int wv[]) {
    for (int i = 0; i < n; i++) ws[i] = 0;
    for (int i = 0; i < n; i++) ws[a[i].maxl]++;
    for (int i = 1; i < n; i++) ws[i] += ws[i - 1];
    for (int i = n - 1; i >= 0; i--) wv[--ws[a[i].maxl]] = i;
}
```

- 回文自动机

```
const int maxn=100060;
const int sigma=26;
int n=0;
char s[maxn];
struct palindrome_tree {
    struct state {
        int len,link;
        int to[sigma];
        state():len(-1),link(-1){}
    } st[maxn];
    int last,sz;
    palindrome_tree():last(1),sz(2){st[1].len=st[1].link=0;}
    int add_letter() {
        char c=s[n-1];
        int p=last;
        while(p!=-1 && c!=s[n-st[p].len-2]) p=st[p].link;
        if(p==-1) {
            last=1;
            return 0;
        }
        int ret=0;
        if(!st[p].to[c]) {
            ret=1;
            int q=last=sz++;
            st[p].to[c]=q;
            st[q].len=st[p].len+2;
            do p=st[p].link; while(p!=-1 && c!=s[n-st[p].len-2]);
            if(p==-1) st[q].link=1;
            else st[q].link=st[p].to[c];
        }
        else last=st[p].to[c];
        return ret;
    }
};
int main() {
    palindrome_tree me;
    s[n++]='#';
    int cur=0;
    while((s[n++]=getchar())!='\n') {
        s[n-1]-='a';
        cout<<(cur+=me.add_letter())<<' ';
    }
}
```

- 后缀数组（倍增）

```
inline bool equal(int *r, int p, int q, int l) {
```

```
    return r[p] == r[q] && r[p + l] == r[q + l];
}
void da(int r[], int sa[], int n, int m) {
    static int wa[N], wb[N], wv[N], ws[N];
    int *x = wa, *y = wb;
    for (int i = 0; i < m; i++) ws[i] = 0;
    for (int i = 0; i < n; i++) ws[x[i] = r[i]]++;
    for (int i = 1; i < m; i++) ws[i] += ws[i - 1];
    for (int i = n - 1; i >= 0; i--) sa[--ws[x[i]]] = i;
    for (int j = 1, p = 1; p < n; j *= 2, m = p) {
        p = 0;
        for (int i = n - j; i < n; i++) y[p++] = i;
        for (int i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa[i] - j;
        for (int i = 0; i < n; i++) wv[i] = x[y[i]];
        for (int i = 0; i < m; i++) ws[i] = 0;
        for (int i = 0; i < n; i++) ws[wv[i]]++;
        for (int i = 1; i < m; i++) ws[i] += ws[i - 1];
        for (int i = n - 1; i >= 0; i--) sa[--ws[wv[i]]] = y[i];
        swap(x, y);
        x[sa[0]] = 0;
        p = 1;
        for (int i = 1; i < n; i++) {
            x[sa[i]] = (equal(y, sa[i - 1], sa[i], j))? p - 1 : p++;
        }
    }
}
void calh(int r[], int sa[], int h[], char s[], int n) {
    for (int i = 0, k = 0; i < n; i++) {
        if (k > 0) k--;
        for (int j = sa[r[i] - 1]; s[i + k] == s[j + k]; k++);
        h[r[i]] = k;
    }
}
void demo() {
    for (int i = 0; i < n; i++) r[i] = s[i] - 'a' + 1;
    r[n] = 0;
    da(r, sa, n + 1, 27);
    for (int i = 1; i <= n; i++) r[sa[i]] = i;
    calh(r, sa, h, s, n);
    calst(lg, st, h, n + 1);
}
```

- ST

```
void calst(int lg[], int st[][K], int h[], int n) {
    for (int i = 1; i <= n; i++) st[i][0] = h[i];
    for (int k = 1; k < K; k++) {
        for (int i = 0; i + (1 << k) <= n; i++) {
            st[i][k] = min(st[i][k - 1], st[i + (1 << (k - 1))][k - 1]);
        }
    }
}
inline int get(int l, int r) {
    int k = lg[r - l + 1];
    return min(st[l][k], st[r - (1 << k) + 1][k]);
}
void init(int lg[]) {
    lg[0] = -1;
    for (int i = 1; i < N; i++)
        if (i & i - 1) lg[i] = lg[i - 1];
        else lg[i] = lg[i - 1] + 1;
}
```

- Dancing Links（精确覆盖）

```
struct Node {
    Node *up, *down, *left, *right;
    int size; //head 表示自己的 size
    //left 节点用-i-1 表示是哪一行，一般节点用一个非负数表示是哪一列
    Node *clear(int s, Node *l = NULL, Node *d = NULL) {
        size = s;
        if (l == NULL) left = right = this;
        else {
            right = l->right;
            left = l;
            l->right = right->left = this;
        }
        if (d == NULL) up = down = this;
        else {
            d->size++;
            up = d->up;
            down = d;
            d->up = up->down = this;
        }
        return this;
    }
    void disrow() {
        left->right = right;
        right->left = left;
    }
    void discol() {
        up->down = down;
        down->up = up;
    }
    void conrow() {
        left->right = this;
        right->left = this;
    }
    void concol() {
        up->down = this;
        down->up = this;
    }
};
Node a[N * M + N + M + 1], *ap;
Node *head[M + 1];
Node *left[N];
```

```cpp
void clear(int n, int m) {
    ap = a;
    head[M] = (ap++)->clear(0); //head[M]是额外的点
    for (int i = 0; i < m; i++) head[i] = (ap++)->clear(0, head[M]);
    for (int i = 0; i < n; i++) left[i] = (ap++)->clear(-i - 1);
}
void addrule(int i, int j) {
    (ap++)->clear(j, left[i], head[j]);
    //表示在第 i 行第 j 列插入一个节点，即第 i 行可以覆盖第 j 列
}
void delrow(Node *x) {
    for (Node *i = x->right; i != x; i = i->right)
        if (i->size >= 0) {
            head[i->size]->size--;
            i->discol();
        }
}
void delcol(int x) {
    head[x]->disrow();
    for (Node *i = head[x]->down; i != head[x]; i = i->down) delrow(i);
}
void choose(Node *x) {
    Node *i = x;
    do {
        if (i->size >= 0) delcol(i->size);
        else {
            int p = -i->size - 1; //行首，标记选了第 p 行
        }
        i = i->right;
    } while (i != x);
}
void conrow(Node *x) {
    for (Node *i = x->left; i != x; i = i->left)
        if (i->size >= 0) {
            head[i->size]->size++;
            i->concol();
        }
}
void concol(int x) {
    for (Node *i = head[x]->up; i != head[x]; i = i->up) conrow(i);
    head[x]->conrow();
}
void unchoose(Node *x) {
    Node *i = x->left;
    while (i != x) {
        if (i->size >= 0) concol(i->size);
        i = i->left;
    }
    if (i->size >= 0) concol(i->size);
}
bool findans() {
    if (head[M]->right == head[M]) return true;
    int minv = INF;
    Node *p;
    for (Node *i = head[M]->right; i != head[M]; i = i->right)
        if (i->size < minv) {
            minv = i->size;
            p = i;
        }
    if (minv == 0) return false; //某个 head 无法被覆盖
    for (Node *i = p->down; i != p; i = i->down) {
        choose(i); //尝试用 i 所在的那一行去覆盖 p
        if (findans()) return true;
        unchoose(i);
    }
    return false;
}
void demo() {
    clear(n, m); //用 n 行覆盖 m 列
    addrule(i, j);
    choose(left[i]); //强制选择第 i 行
    findans();
}
```

- **Dancing Links（模糊覆盖）**

```cpp
void delcol(int x) {
    head[x]->disrow();
    for (Node *i = head[x]->down; i != head[x]; i = i->down) i->disrow();
}
void choose(Node *x) {
    int ans;
    Node *i = x;
    do {
        if (i->size >= 0) {
            delcol(i->size);
            i->conrow();
        } else {
            int p = -i->size - 1; //行首，标记选了第 p 行
        }
        i = i->right;
    } while (i != x);
}
void concol(int x) {
    head[x]->conrow();
    for (Node *i = head[x]->up; i != head[x]; i = i->up) i->conrow();
}
int h() {
    unordered_map<Node *, bool> has;
    int ans=0;
    for (Node *i = head[M]->right; i != head[M]; i = i->right)
        if (!has[i]) {
            ans++;
            has[i] = true;
            for (Node *j = i->down; j != i; j = j->down)
```

```
                for (Node *k = j->right; k != j; k = k->right)
                    if (k->size >= 0) has[head[k->size]] = true;
        }
    return ans;
}
void findans(int cur) {
    if (cur + h() >= ans) return;
    if (head[M]->right == head[M]) {
        ans = min(ans, cur);
        return;
    }
    int minv = INF;
    Node *p;
    for (Node *i = head[M]->right; i != head[M]; i = i->right)
        if (i->size < minv) {
            minv = i->size;
            p = i;
        }
    if (minv == 0) return; //某个 head 无法被覆盖
    for (Node *i = p->down; i != p; i = i->down) {
        choose(i); //尝试用 i 所在的那一行去覆盖 p
        findans(cur + 1);
        unchoose(i);
    }
}
```

- **后缀数组（DC3）（乐神）**

```
const int maxn=1000010;
int wa[maxn],wb[maxn],wv[maxn],wt[maxn],r[3*maxn],sa[3*maxn];
char str[maxn];
int rank[maxn],height[maxn];
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
int c0(int *r,int a,int b) {
    return r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];
}
int c12(int k,int *r,int a,int b) {
    if(k==2) return r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);
    else return r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];
}
void sort(int *r,int *a,int *b,int n,int m) {
    int i;
    for(i=0;i<n;i++) wv[i]=r[a[i]];
    for(i=0;i<m;i++) wt[i]=0;
    for(i=0;i<n;i++) wt[wv[i]]++;
    for(i=1;i<m;i++) wt[i]+=wt[i-1];
    for(i=n-1;i>=0;i--) b[--wt[wv[i]]]=a[i];
    return;
}
void dc3(int *r,int *sa,int n,int m) {
    int i,j,*rn=r+n,*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;
    r[n]=r[n+1]=0;
    for(i=0;i<n;i++) if(i%3!=0) wa[tbc++]=i;
    sort(r+2,wa,wb,tbc,m);
    sort(r+1,wb,wa,tbc,m);
    sort(r,wa,wb,tbc,m);
    for(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)
        rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
    if(p<tbc) dc3(rn,san,tbc,p);
    else for(i=0;i<tbc;i++) san[rn[i]]=i;
    for(i=0;i<tbc;i++) if(san[i]<tb) wb[ta++]=san[i]*3;
    if(n%3==1) wb[ta++]=n-1;
    sort(r,wb,wa,ta,m);
    for(i=0;i<tbc;i++) wv[wb[i]=G(san[i])]=i;
    for(i=0,j=0,p=0;i<ta && j<tbc;p++)
        sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
    for(;i<ta;p++) sa[p]=wa[i++];
    for(;j<tbc;p++) sa[p]=wb[j++];
    return;
}
void calheight(int *r,int *sa,int n) {
    int i,j,k=0;
    for(i=1;i<n;i++) rank[sa[i]]=i;
    for(i=0;i<n-1; height[rank[i++]] = k )
        for(k?k--:0,j=sa[rank[i]-1]; r[j+k]==r[i+k];k++);
}
void init_lg() {
    int i;
    lg[1]=0;
    for(i=2;i<102020;i++) lg[i]=lg[i>>1]+1;
}
void init_RMQ(int n) {
    int i,j,k;
    for(i=1;i<=n;i++) minv[i][0]=height[i];
    for(j=1;j<=lg[n];j++) {
        for(k=0;k+(1<<j)-1<=n;k++) {
            minv[k][j]=min(minv[k][j-1],minv[k+(1<<(j-1))][j-1]);
        }
    }
}
int lcp(int l,int r) {
    l=Rank[l];
    r=Rank[r];
    if(l>r) swap(l,r);
    l++;
    int k=lg[r-l+1];
    return min(minv[l][k],minv[r-(1<<k)+1][k]);
}
int dp[maxn];
int main(){
    while(scanf("%s",str)){
        if(str[0]=='.')return 0;
        int n,m=0;
        for(n=0;str[n];n++)r[n]=str[n],m=max(m,r[n]);
```

```
        r[n++]=0;
        dc3(r,sa,n,m+1);
        calheight(r,sa,n);
        dp[rank[0]]=n;
        for(int i=rank[0]+1;i<n;i++)dp[i]=min(dp[i-1],height[i]);
        for(int i=rank[0]-1;i+1;i--)dp[i]=min(dp[i+1],height[i+1]);
        n--;
        for(int i=n;i;i--){
            if(n%i==0&&dp[rank[n/i]]==n-n/i){printf("%d\n",i);break;}
}}}
```

- GAUSS（乐神）

```
const int N=100;
struct mat{
    double a[N+1][N+1];
    mat(){for(int i=0;i<=N;i++)for(int j=0;j<=N;j++)a[i][j]=0;}
};
double b[N+1];
double x[N+1];
double *solve(mat m,double *b,bool &yes){
    for(int i=0;i<N;i++) m.a[i][N]=b[i];
    for(int i=0;i<N;i++) {
        int tmp=i;
        for(int j=i+1;j<N;j++)
            if(fabs(m.a[j][i])>fabs(m.a[tmp][i])) tmp=j;
        swap(m.a[tmp],m.a[i]);
        if(fabs(m.a[i][i])<=1e-7)yes=0;
        for(int j=i+1;j<=N;j++)
            m.a[i][j]/=m.a[i][i];
        for(int j=0;j<N;j++)
            if(i!=j) {
                for(int k=i+1;k<=N;k++)
                    m.a[j][k]-=m.a[i][k]*m.a[j][i];
            }
    }
    for(int i=0;i<N;i++) x[i]=m.a[i][N];
    yes=1;
    return x;
}
```

- SPLAY（乐神）

```
const int maxn = 220000, inf = 1000111000;
int son[maxn][2],pre[maxn],val[maxn],cnt[maxn],tot,rt;
int laz[maxn],rev[maxn];
int n,m,k1,k2;
int a[maxn];
struct Splay{
    void init(int n){
        tot=0;
        rt=1;
        build(0,son[0][1],1,n);
    }
    void newnode(int fa,int &x,int v){
        x=++tot;
        val[x]=v;
        son[x][0]=son[x][1]=0;
        pre[x]=fa;cnt[x]=1;
        laz[x]=rev[x]=0;
    }
    void build(int fa,int &x,int l,int r){
        if(r<l){ x=0; return ;}
        int mid=l+r>>1;
        x=++tot;
        laz[x]=rev[x]=0;
        val[x]=a[mid];
        cnt[x]=r-l+1;
        pre[x]=fa;
        build(x,son[x][0],l,mid-1);
        build(x,son[x][1],mid+1,r);
    }
    void push_up(int x){
        cnt[x]=cnt[son[x][0]]+cnt[son[x][1]]+1;
    }
    void push_down(int x){
        if(rev[x]){
            swap(son[x][0],son[x][1]);
            rev[son[x][0]]^=1;
            rev[son[x][1]]^=1;
            rev[x]=0;
        }
        if(laz[x]){
            for(int i=0;i<2;i++){
                laz[son[x][i]]+=laz[x];
                val[son[x][i]]+=laz[x];
            }
            laz[x]=0;
        }
    }
    void rotate(int x,int c){  //c=1 zig c=0 zag  asume pre[x]>0
        int y=pre[x];
        push_down(y);
        push_down(x);
        son[y][!c]=son[x][c];
        pre[son[x][c]]=y;
        son[x][c]=y;
        pre[x]=pre[y];
        pre[y]=x;
        if(pre[x])son[pre[x]][son[pre[x]][0]!=y]=x;
        push_up(y);
    }
    void splay(int x,int goal){
        push_down(x);
        while(pre[x]!=goal){
            int y=pre[x],z=pre[y];
            if(z==goal) rotate(x,son[y][0]==x);
```

```
            else {  // 同向 yx ,否则 xx
                if(son[z][0]==y){
                    if(son[y][0]==x)rotate(y,1),rotate(x,1);
                    else rotate(x,0),rotate(x,1);
                } else {
                    if(son[y][0]==x)rotate(x,1),rotate(x,0);
                    else rotate(y,0),rotate(x,0);
        }}}
        push_up(x);
        if(goal==0)rt=x;
    }
    void find(int k){
        int cur=rt;
        while(1){
            push_down(cur);
            if(cnt[son[cur][0]]==k-1)break;
            if(k>cnt[son[cur][0]]){
                k-=cnt[son[cur][0]]+1;
                cur=son[cur][1];
            }
            else cur=son[cur][0];
        }
        splay(cur,0);
    }
}S;
int main() {
    int ca=1;
    while(cin>>n>>m>>k1>>k2,n){
        for(int i=1;i<=n;i++)scanf("%d",&a[i]);
        S.init(n);
        printf("Case #%d:\n",ca++);
        char ch[100];
        while(m--){
            scanf("%s",ch);
            int x,k;
            if(ch[0]=='q'){
                S.find(1);printf("%d\n",val[rt]);
            }
            else if(ch[0]=='a'){
                scanf("%d",&x);
                S.find(k2+1);
                S.push_down(rt);
                int cur=son[rt][0];
                val[cur]+=x;laz[cur]+=x;
            }
            else if(ch[0]=='r'){
                S.find(k1+1);
                S.push_down(rt);
                int cur=son[rt][0];
                rev[cur]^=1;
            }
            else if(ch[0]=='i'){
                n++;
                scanf("%d",&x);
                S.find(2);
                int cur=son[rt][0];
                S.push_down(cur);
                S.newnode(cur,son[cur][1],x);
                S.push_up(cur);
                S.push_up(rt);
            }
            else if(ch[0]=='d'){
                n--;
                S.find(2);
                son[rt][0]=0;
                S.push_up(rt);
            }
            else {
                scanf("%d",&x);
                if(x==2){
                    S.find(2);
                    int cur=son[rt][0],v=val[cur];
                    son[rt][0]=0;
                    S.push_up(rt);
                    S.find(n-1);
                    S.newnode(rt,son[rt][1],v);
                    S.push_up(rt);
                }
                else {
                    S.find(n-1);
                    int cur=son[rt][1],v=val[cur];
                    son[rt][1]=0;
                    S.push_up(rt);
                    S.find(1);
                    S.newnode(rt,son[rt][0],v);
                    S.push_up(rt);
}}}}}
```

- **点分治（乐神）**

```
int t,n,m,a[size],K;
vector<int>V[size];
int vis[size],sum[size],id,tmp;
void find(int u,int fa,int num){
    sum[u]=1;int K=0;
    for(int i=0;i<V[u].size();i++){
        int to=V[u][i];
        if(vis[to]||to==fa)continue;
        find(to,u,num);
        K=max(K,sum[to]);
        sum[u]+=sum[to];
    }
    K=max(K,num-sum[u]);
    if(K<tmp)tmp=K,id=u;
}
```

```
P b[size];
int tot;
void dfs(int u,int fa,int mi,int ma){
    sum[u]=1;
    mi=min(mi,a[u]);
    ma=max(ma,a[u]);
    if(ma<=mi+K)b[tot++]=P(mi,ma);
    for(int i=0;i<V[u].size();i++){
        int to=V[u][i];
        if(vis[to]||to==fa)continue;
        dfs(to,u,mi,ma);
        sum[u]+=sum[to];
}}
ll gao(int u,int mi,int ma){
    tot=0;
    dfs(u,0,mi,ma);
    sort(b,b+tot);
    ll ans=0;
    for(int i=0;i<tot;i++){
        int p=lower_bound(b,b+i,P(b[i].second-K,0))-b;
        ans+=i-p;
    }
    return ans;
}
ll work(int u,int num){
    tmp=n;
    find(u,0,num);
    u=id;
    ll ans=gao(u,a[u],a[u]);
    vis[u]=1;
    for(int i=0;i<V[u].size();i++){
        int to=V[u][i];
        if(!vis[to])ans-=gao(to,a[u],a[u]);
    }
    for(int i=0;i<V[u].size();i++){
        int to=V[u][i];
        if(!vis[to])ans+=work(to,sum[to]);
}return ans;}
int main(){
    cin>>t;
    while(t--){
        cin>>n>>K;
        for(int i=1;i<=n;i++)V[i].clear(),scanf("%d",&a[i]),vis[i]=0;
        for(int i=1;i<n;i++){
            int x,y;
            scanf("%d%d",&x,&y);
            V[x].push_back(y);
            V[y].push_back(x);
        }
        cout<<work(1,n)*2<<endl;
}}
```

- **分治并查集（乐神）**

```
const int size = 60000;
typedef double dd;
int n,m,ans[size],TIM;
struct node{
    int u,v,len,id,st,ed;
}e[size],cpy[size];
int cmp(node a,node s){return a.len<s.len;}
int fa[size],d[size],sta[size*3],top;
void init(){
    for(int i=1;i<=n;i++)fa[i]=i,d[i]=1;
    top=0;
}
int get(int x){
    while(x!=fa[x])x=fa[x];
    return x;
}
int uni(int x,int y){
    x=get(x);y=get(y);
    if(x==y)return 0;
    if(d[x]>d[y])swap(x,y);
    fa[x]=y;sta[top++]=x;
    if(d[x]==d[y]){sta[top++]=-y;d[y]++;}
    return 1;
}
void resume(int tmp){
    while(top>tmp){
        if(sta[top-1]<0){
            d[-sta[top-1]]--;
            fa[sta[top-2]]=sta[top-2];
            top-=2;
        }
        else {
            fa[sta[top-1]]=sta[top-1];
            top--;
}}}
int gao(int l,int r,int m,int ttt,int n){
    for(int i=0;i<=m;i++){
        if(l>=e[i].st&&r<=e[i].ed){
            n-=(uni(e[i].u,e[i].v));
            swap(e[i--],e[m--]);
        }
        else if(l>e[i].ed||r<e[i].st)swap(e[i--],e[m--]);
    }
    if(n==1){
        TIM=l;
        return 1;
    }
    if(l<r){
        int mid=l+r>>1;
        int i,j;
```

```
        for(i=0,j=m;i<=j;i++)
            if(e[i].st>mid)swap(e[i--],e[j--]);
        if(gao(l,mid,j,top,n))return 1;
        for(i=0,j=m;i<=j;i++)
            if(e[i].ed<=mid)swap(e[i--],e[j--]);
        if(gao(mid+1,r,j,top,n))return 1;
    }
    resume(ttt);
    return 0;
}
int work(int p){
    for(int i=0;i<m;i++)e[i]=cpy[i];
    int cur=0,pre=0,tim=1;
    while(pre<m){
        if(cur==m)e[pre++].ed=tim;
        else {
            e[cur].st=tim;
            while(e[cur].len-e[pre].len>p){
                e[pre].ed=tim-1;pre++;
            }
            tim++;cur++;
    }}
    init();
    return gao(1,tim,m-1,top,n);
}
```

- **上下界最大流（乐神）**

```
const int size= 205 ;
const int inf=100000000;
int S,T,S1,T1;
struct node{
    int to,rev,f,next,id;
}E[1000000];
int head[size],tot,q[size],f,r,lev[size];
int ans[size*size],n,m;
void add(int x,int y,int c){
    E[tot].to=y;E[tot].next=head[x];E[tot].f=c;E[tot].rev=tot+1;
    head[x]=tot++;
    E[tot].to=x;E[tot].next=head[y];E[tot].f=0;E[tot].rev=tot-1;
    head[y]=tot++;
}
int bfs(int S,int T){
    f=r=0;q[f++]=S;
    memset(lev,-1,sizeof(lev));lev[S]=0;
    while(f!=r){
        int u=q[r++];
        for(int i=head[u];i!=-1;i=E[i].next){
            int to=E[i].to;
            if(E[i].f&&lev[to]==-1)
                {lev[to]=lev[u]+1;q[f++]=to;if(to==T)return 1;}
}}return 0;}
int dfs(int u,int T,int f){
    if(u==T)return f;
    int ans=0,c;
    for(int i=head[u];i!=-1;i=E[i].next){
        int to=E[i].to;
        if(lev[to]==lev[u]+1&&E[i].f){
            c=dfs(to,T,min(f-ans,E[i].f));
            E[i].f-=c;E[ E[i].rev ].f+=c;
            ans+=c;if(ans==f)return f;
}}return ans;}
int max_flow(int S,int T){
    int ans=0;
    while(bfs(S,T)){ans+=dfs(S,T,inf);}
    return ans;
}
int init(){
    S=0;T=n+1;int sum=0;tot=0;
    memset(head,-1,sizeof(head));
    for(int i=1;i<=m;i++){
        int a,b,c,d;scanf("%d%d%d%d",&a,&b,&c,&d);
    }
    if(max_flow(S,T)==sum)return 1;
    return 0;
}
```

- **树链剖分（乐神）**

```
const int maxn=50010;
vector<int>V[maxn];
typedef long long ll;
int siz[maxn],dep[maxn],top[maxn],son[maxn],fa[maxn],w[maxn],val[maxn];
int n,m,q,x,y,tmp,tot;
int in(){
    char ch;
    while(ch=getchar(),ch<'0'||ch>'9');
    int ans=ch-'0';
    while(ch=getchar(),ch>='0'&&ch<='9')ans=10*ans+ch-'0';
    return ans;
}
void bfs(int u,int f){
    siz[u]=1;dep[u]=dep[f]+1;fa[u]=f;
    int to,ma=0;
    for(int i=0;i<V[u].size();i++) if(f!=V[u][i]){
        to=V[u][i];
        bfs(to,u);
        siz[u]+=siz[to];
        if(siz[to]>siz[ma])ma=to;
    }
    son[u]=ma;
}
void bfs(int u){
    w[u]=++tot;
    if(!fa[u]||son[fa[u]]!=u)top[u]=u;
    else top[u]=top[fa[u]];
```

```
    if(son[u])bfs(son[u]);
    for(int i=0;i<V[u].size();i++)
        if(fa[u]!=V[u][i]&&son[u]!=V[u][i])
            bfs(V[u][i]);
}
int lz[maxn*3],date[maxn];
ll seg[maxn*3];
ll build(int p,int l,int r){
    lz[p]=0;
    if(l==r) return seg[p]=date[l];
    int mid=(l+r)/2;
    seg[p]=build(p*2,l,mid)+build(p*2+1,mid+1,r);
}
void down(int p,int l,int r){
    lz[p*2]+=lz[p];lz[p*2+1]+=lz[p];
    int mid=(l+r)/2;
    seg[p*2]+=lz[p]*(mid-l+1);
    seg[p*2+1]+=lz[p]*(r-mid);
    lz[p]=0;
}
int get(int p,int l,int r,int x){
    if(l==r)return seg[p];
    if(lz[p])down(p,l,r);
    int mid=(l+r)/2;
    if(mid>=x)return get(p*2,l,mid,x);
    return get(p*2+1,mid+1,r,x);
}
void add(int p,int l,int r,int L,int R,int x){
    if(L>r||R<l)return ;
    if(l>=L&&r<=R)lz[p]+=x,seg[p]+=(r-l+1)*x;
    else {
        int mid=(l+r)/2;
        if(lz[p])down(p,l,r);
        add(p*2,l,mid,L,R,x);
        add(p*2+1,mid+1,r,L,R,x);
        seg[p]=seg[p*2]+seg[p*2+1];
}}
void init(){
    for(int i=0;i<=n;i++){
        V[i].clear();son[i]=fa[i]=w[i]=val[i]=siz[i]=dep[i]=top[i]=tot=0;
    }
    for(int i=1;i<=n;i++)val[i]=in();
    while(m--){
        x=in();y=in();
        V[x].push_back(y);
        V[y].push_back(x);
    }
    bfs(1,0);
    bfs(1);
    for(int i=1;i<=n;i++)date[w[i]]=val[i];
    build(1,1,n);
}
void work(int x,int y,int z){//cout<<x<<' '<<y<<' '<<z<<endl;
    if(dep[x]>dep[y])swap(x,y);
    if(top[x]==top[y])add(1,1,n,w[x],w[y],z);
    else {
        int fy=top[y],fx=top[x];
        if(dep[fy]>dep[fx]){
            add(1,1,n,w[fy],w[y],z);
            work(x,fa[fy],z);
        }
        else {
            add(1,1,n,w[fx],w[x],z);
            work(fa[fx],y,z);
}}}
int main(){
    while(~scanf("%d%d%d",&n,&m,&q)){
        init();
        while(q--){
            char ch;
            scanf(" %c%d",&ch,&x);
            if(ch=='Q')printf("%d\n",get(1,1,n,w[x]));
            else {
                scanf("%d%d",&y,&tmp);
                if(ch=='D')tmp*=-1;
                work(x,y,tmp);
}}}}
```

- **模线性方程组（乐神）**

```
ll extend_Euclid(ll a, ll b, ll &x, ll &y) {
    if(b == 0) {
        x = 1;
        y = 0;
        return a ;
    }
    ll ans = extend_Euclid(b, a % b, x, y);
    ll tmp = x;
    x = y;
    y = tmp - (a / b) * y;
    return ans;
}
ll g,l,ans,d;
ll a[size],n,b[size];
ll get(){
    ll x, y;
    g = extend_Euclid(a[0], b[0], x, y);
    l = a[0];
    ll a1 = a[0], b1 = b[0], a2, b2, c;
    for(ll i = 1; i < n; i ++){
        a2 = a[i], b2 = b[i];
        d = extend_Euclid(a1, a2, x, y);
        if( (b2 - b1) % d) return -1;
        x %= a2;
        c = (b2 - b1) % a1;
```

```
        x *= c / d;
        x += (b2 - b1 - c) / a1;
        l = l / d * a2;
        b1 = (x) % a2 * a1 + b1;
        a1 = l;
        b1 %= a1;
    }
    return  (b1 % a1 + a1) % a1;
}
int main(){
    while(cin>>n){
        for(ll i=0;i<n;i++)cin>>a[i]>>b[i];
        ans=get();
        cout<<ans<<endl;
}}
```

- **主席树（乐神）**

```
/* 静态主席树求区间第 K 大
 * poj 2104
 * 主要思想：
 * 1. 将数据离散化
 * 2. 用线段树来维护信息，维护处于该区间的数的个数
 * 3. 将原有数组中的每一个数依次插入，每次插入一个值的时候新建一棵线段树
 *    这样就有了 n+1 棵线段树，对于询问(l,r,k)，只需查询 T[l-1]和 T[r]两棵线
 *    段树即可，其中 T[i]表示插入第 i 个数后的线段树询问过程，只要利用二分
 *    思想：若这两棵线段树的左儿子区间的数的个数差不小于 k,则答案往左儿树
 *    中找，否则往右子树找。
 * 4. 上述建树空间复杂度肯定太大，所以要空间重用。
 *    注意当插入一个数的时候，插入后的线段树（新）和插入前的线段树（旧）
 *    之间有很多信息都是一样的，容易发现只有从修改位置到根的 log(n)的路
 *    径是不一样的，所以新树中的其他不变的子树只要重用旧子树的相应位置就
 *    好，这样每次建树空间复杂度为 log(n)了。
 * 5. 详细见代码
 */
#define m (l+r)/2
const int N = 100100;
const int M = N * 30;
int n, q, num, a[N], b[N], l, r, k;
int T[M], cnt[M], lson[M], rson[M], tot;
int hash(int a){ return lower_bound(b + 1, b + num + 1, a) - b; }
int init(int l, int r){
    int root = tot ++;
    cnt[root] = 0;
    if(l < r) lson[root] = init(l, m), rson[root] = init(m + 1, r);
    return root;
}
int upd(int pre_root, int pos, int val){
    int cur_root = tot ++, ret = cur_root;
    cnt[cur_root] = cnt[pre_root] + val;
    int l = 1, r = num;
    while(r > l){
        if(pos <= m){
            rson[cur_root] = rson[pre_root];
            cur_root = lson[cur_root] = tot ++;
            pre_root = lson[pre_root];
            cnt[cur_root] = cnt[pre_root] + val;
            r = m;
        } else {
            lson[cur_root] = lson[pre_root];
            cur_root = rson[cur_root] = tot ++;
            pre_root = rson[pre_root];
            cnt[cur_root] = cnt[pre_root] + val;
            l = m + 1;
    }}
    return ret;
}
int query(int pre, int cur, int k){
    int l = 1, r = num;
    while(r > l){
        if(k <= cnt[lson[cur]] - cnt[lson[pre]]){
            pre = lson[pre];
            cur = lson[cur];
            r = m;
        } else {
            k -= cnt[lson[cur]] - cnt[lson[pre]];
            pre = rson[pre];
            cur = rson[cur];
            l = m + 1;
    }}
    return l&r;
}
int main(){
    while(~scanf("%d %d", &n, &q)){
        for(int i = 1; i <= n; i ++) scanf("%d",&a[i]), b[i] = a[i];
        sort(b + 1, b + n + 1);
        num = unique(b + 1, b + n + 1) - b - 1;
        tot = 0;
        T[0] = init(1, num);
        for(int i = 1; i <= n; i ++) T[i] = upd(T[i-1], hash(a[i]), 1);
        while(q --){
            scanf("%d %d %d", &l, &r, &k);
            printf("%d\n", b[query(T[l - 1], T[r], k)]);
}}}
```

- **主席树+并查集（乐神）**

```
//BZOJ 3674: 可持久化并查集加强版
int get(int root,int pos,int tag){
    int l=1,r=n;
    while(r>l){
        if(pos<=m)r=m,root=lson[root];
        else l=m+1,root=rson[root];
    }
    if(tag)return dp[root];
    return fa[root];
```

```
}
int get_fa(int root,int pos){
    int tmp=pos;
    while((tmp=get(root,tmp,0))!=pos)pos=tmp;
    return tmp;
}
int main(){
    while(~scanf("%d%d",&n,&q)) {
        tot=0;
        T[0]=init(1,n);
        int ans=0;
        for(int i=1;i<=q;i++){
            int p,a,b;  //b=1;
            p=read();a=read();
            if(p==2){
                a^=ans;
                T[i]=T[a];
            }
            if(p==1){
                b=read();
                a^=ans;b^=ans;
                T[i]=T[i-1];
                int fa=get_fa(T[i],a),fb=get_fa(T[i],b);
                if(fa!=fb){
                    int dpa=get(T[i],fa,1),dpb=get(T[i],fb,1);
                    if(dpa<dpb)swap(dpa,dpb),swap(fa,fb);
                    T[i]=upd(T[i],fb,fa,0);
                    if(dpa==dpb)T[i]=upd(T[i],fa,dpa+1,1);
            }}
            if(p==3){
                b=read();
                a^=ans;b^=ans;
                T[i]=T[i-1];
                ans=(get_fa(T[i],a)==get_fa(T[i],b));
                printf("%d\n",ans);
}}}}
```

- **主席树套树状数组（乐神）**

```
/* ZOJ 2112 动态第 k 大
 * 考虑静态第 k 大的主席树做法，第 i 棵树 T[i]保存的是数组前 i 个元素的信息
 * 对询问(i,j,k)，只用取出 T[j]和 T[i-1]即可
 * 若数组元素有修改，做法也差不多
 * 令 T[i]只记录数组前 i 个元素的信息
 * 修改 a[i]=j 的时候，需对所有 k>=i 的 T[k]修改
 * 对询问，也只用取出 T[j]和 T[i-1]即可
 * 上述操作类似树状数组，所以没必要真的修改那么对 T[k]
 */
int upd(int pre_root, int pos, int val){
}
void UPDATE(int pos, int val, int d){
    for( ; pos <= n ; pos += pos & -pos) T[pos] = upd(T[pos], val, d);
}
int use[N];
int query(int pre, int cur, int k){
    int l = 1, r = num, P = pre, C = cur, pp = S[pre], cc = S[cur];
    for(pre = P; pre > 0 ; pre -= pre & -pre) use[pre] = T[pre];
    for(cur = C; cur > 0 ; cur -= cur & -cur) use[cur] = T[cur];
    while(r > l){
        int left_sum = cnt[lson[pp]], right_sum = cnt[lson[cc]];
        for(pre = P; pre > 0 ; pre -= pre & -pre)
            left_sum += cnt[lson[use[pre]]];
        for(cur = C; cur > 0 ; cur -= cur & -cur)
            right_sum += cnt[lson[use[cur]]];
        if(k <= right_sum - left_sum){
          for(pre = P; pre > 0 ; pre -= pre & -pre) use[pre] = lson[use[pre]];
          for(cur = C; cur > 0 ; cur -= cur & -cur) use[cur] = lson[use[cur]];
          pp = lson[pp]; cc = lson[cc]; r = m;
        } else {
          k -= right_sum - left_sum;
          for(pre = P; pre > 0 ; pre -= pre & -pre) use[pre] = rson[use[pre]];
          for(cur = C; cur > 0 ; cur -= cur & -cur) use[cur] = rson[use[cur]];
          pp = rson[pp]; cc = rson[cc]; l = m + 1;
    }}
    return l&r;
}
int main(){
    int t;
    scanf("%d", &t);
    while(t --){
        scanf("%d %d", &n, &q);
        num = 0;
        for(int i = 1; i <= n; i ++) scanf("%d",&a[i]), b[num++] = a[i];
        for(int i = 0; i < q; i ++) {
            scanf(" %c%d%d", &Q[i].ch, &Q[i].i, &Q[i].j);
            if(Q[i].ch == 'Q') scanf("%d", &Q[i].k);
            else b[num++] = Q[i].j;
        }
        sort(b, b + num);
        num = unique(b , b + num) - b;
        tot = 0;
        S[0] = T[0] = init(1, num);
        for(int i = 1; i <= n ; i ++)
            S[i] = upd(S[i-1], hash(a[i]), 1), T[i] = T[0];
        for(int i = 0; i < q; i ++) {
            if(Q[i].ch == 'Q'){
                printf("%d\n", b[query(Q[i].i - 1, Q[i].j, Q[i].k)- 1] );
            } else {
                UPDATE(Q[i].i, hash(a[Q[i].i]), -1);
                UPDATE(Q[i].i, hash(a[Q[i].i] = Q[i].j), 1);
}}}}
```

- **Link-Cut Tree**

```
struct Node{
    Node *fa,*ch[2];
    bool rev, root;
    int val, minv;
};
Node pool[N];
Node *nil,*tree[N];
int cnt = 0;
void init(){
    cnt = 1;
    nil = tree[0] = pool;
    nil->ch[0] = nil->ch[1] = nil;
    nil->val = 0;
    nil->minv = 0;
}
Node *newnode(int val, Node *f){
    pool[cnt].fa = f;
    pool[cnt].ch[0]=pool[cnt].ch[1]=nil;
    pool[cnt].rev = false;
    pool[cnt].root = true;
    pool[cnt].val = val;
    pool[cnt].minv = val;
    return &pool[cnt++];
}
//左右子树反转******真正把结点变为根
void update_rev(Node *x) {
    if(x == nil) return ;
    x->rev = !x->rev;
    swap(x->ch[0],x->ch[1]);
}
//splay 向上更新信息******
void update(Node *x) {
    if(x == nil) return ;
    x->minv = x->val;
    Node*y = x->ch[0];
    if(y->minv > x->minv)
        x->minv = y->minv;
    y = x->ch[1];
    if(y->minv > x->minv)
        x->minv = y->minv;
}
//splay 下推信息******
void pushdown(Node *x) {
    if(x->rev != false){
        update_rev(x->ch[0]);
        update_rev(x->ch[1]);
        x->rev = false;
    }
}
//splay 在 root-->x 的路径下推信息******
void push(Node *x) {
    if(!x->root) push(x->fa);
    pushdown(x);
}
//将结点 x 旋转至 splay 中父亲的位置******
void rotate(Node *x) {
    Node *f = x->fa, *ff = f->fa;
    int t = (f->ch[1] == x);
    if(f->root)
        x->root = true, f->root = false;
    else ff->ch[ff->ch[1] == f] = x;
    x->fa = ff;
    f->ch[t] = x->ch[t^1];
    x->ch[t^1]->fa = f;
    x->ch[t^1] = f;
    f->fa = x;
    update(f);
}
//将结点 x 旋转至 x 所在 splay 的根位置******
void splay(Node *x) {
    push(x);
    Node *f, *ff;
    while(!x->root){
        f = x->fa,ff = f->fa;
        if(!f->root) {
            if((ff->ch[1]==f)&&(f->ch[1] == x))
                rotate(f);
            else rotate(x);
        }
        rotate(x);
    }
    update(x);
}
//将 x 到树根的路径并成一条 path******
Node *access(Node *x) {
    Node *y = nil;
    while(x != nil){
        splay(x);
        x->ch[1]->root = true;
        (x->ch[1] = y)->root = false;
        update(x);
        y = x;
        x = x->fa;
    }
    return y;
}
//将结点 x 变成树根******
void be_root(Node *x){
    access(x);
    splay(x);
    update_rev(x);
}
//将 x 连接到结点 f 上******
void link(Node *x, Node *f){
    be_root(x);
    x->fa = f;
}
//将 x,y 分离******
void cut(Node *x,Node *y){
    be_root(x);
    access(x);
    splay(y);
    y->fa = nil;
}
Node *find(Node *root){
    if(root->ch[0] == nil) return root;
    return find(root->ch[0]);
}
Node*road[N];
int main(){
    int n,q,t;
    Node*x,*y,*z;
    scanf("%d",&t);
    char word[20];
    while(t--){
        scanf("%d",&n);
        init();
        int u,v,t;
        for(int i = 1;i <= n; i++)
            tree[i] = newnode(0,nil);
        for(int i = 1;i < n ;i++){
            scanf("%d%d%d",&u,&v,&t);
            road[i] = x = newnode(t,nil);
            link(tree[u],x);
            link(tree[v],x);
        }
        while(1){
            scanf("%s",word);
            if(word[0] == 'D') break;
            scanf("%d%d",&u,&v);
            if(word[0] == 'Q'){
                be_root(tree[u]);
                y = access(tree[v]);
                printf("%d\n",y->minv);
            } else {
                splay(road[u]);
                road[u]->val = v;
                update(road[u]);
}}}}
```