**ST表**

int f[N][M];

void init() {

for (int j = 0; j < M; j++) {

for (int i = 1; i + (1 << j) - 1 <= n; i++) {

if (j == 0) f[i][j] = a[i];

else f[i][j] = max(f[i][j - 1], f[i + (1 << j - 1)][j - 1]);

}

}

}

int query(int l, int r) {

int k = log(r - l + 1) / log(2);

return max(f[l][k], f[r - (1 << k) + 1][k]);

}

**动态ST表**

void insert(int u) {

f[u][0] = a[u];

for (int i = 1; u - (1 << i) >= 0; i++) {

f[u][i] = max(f[u][i - 1], f[u - (1 << i - 1)][i - 1]);

}

}

int query(int l, int r) {

int k = log(r - l + 1) / log(2);

return max(f[r][k], f[l + (1 << k) - 1][k]);

}

**FHQ平衡树**

struct Tree {

int l, r;

int key, val;

int size;

}tr[N];

int T, n, idx; int root; int x, y, z;

void pushup(int u) {

tr[u].size = tr[tr[u].l].size + tr[tr[u].r].size + 1;

}

int get\_Node(int val) {

tr[++idx] = {0, 0, rand() \* N, val, 1};

return idx;

}

void split(int u, int val, int &x, int &y) {

if (u == 0) x = y = 0;

else {

if (tr[u].val <= val) {

x = u;

split(tr[u].r, val, tr[u].r, y);

} else {

y = u;

split(tr[u].l, val, x, tr[u].l);

}

pushup(u);

}

}

int merge(int x, int y) {

if (x == 0 || y == 0) return x + y;

if (tr[x].key <= tr[y].key) {

tr[x].r = merge(tr[x].r, y);

pushup(x);

return x;

} else {

tr[y].l = merge(x, tr[y].l);

pushup(y);

return y;

}

}

void insert(int val) {

split(root, val, x, y);

root = merge(x, merge(get\_Node(val), y));

}

void del(int val) {

split(root, val, x, z);

split(x, val - 1, x, y);

y = merge(tr[y].l, tr[y].r);

root = merge(x, merge(y, z));

}

void get\_rank(int val) {

split(root, val - 1, x, y);

cout << tr[x].size << endl;

root = merge(x, y);

}

void get\_val(int rank) {

int u = root;

while (u != 0) {

if (tr[tr[u].l].size + 1 == rank) {

break;

} else if (tr[tr[u].l].size >= rank) {

u = tr[u].l;

} else {

rank -= tr[tr[u].l].size + 1;

u = tr[u].r;

}

}

cout << tr[u].val << endl;

}

void get\_prev(int val) {

split(root, val - 1, x, y);

int u = x;

while (tr[u].r) {

u = tr[u].r;

}

cout << tr[u].val << endl;

root = merge(x, y);

}

void get\_next(int val) {

split(root, val, x, y);

int u = y;

while (tr[u].l) {

u = tr[u].l;

}

cout << tr[u].val << endl;

root = merge(x, y);

}

void solve() {

cin >> n;

insert(-INF); insert(INF);

while (n--) {

int t, x; cin >> t >> x;

if (t == 1) { insert(x); }

else if (t == 2) { del(x); }

else if (t == 3) { get\_rank(x); }

else if (t == 4) { get\_val(x + 1); }

else if (t == 5) { get\_prev(x);}

else { get\_next(x);}

}

}

**文艺平衡树**

struct Tree {

int l, r;

int key, val;

int lazy, size;

}tr[N];

int T, n, m, idx; int root; int x, y, z;

void pushup(int u) { tr[u].size = tr[tr[u].l].size + tr[tr[u].r].size + 1; }

void pushdown(int u) {

swap(tr[u].l, tr[u].r);

tr[tr[u].l].lazy ^= 1;

tr[tr[u].r].lazy ^= 1;

tr[u].lazy = 0;

}

int get\_Node(int val) {

tr[++idx] = {0, 0, rand() \* N, val, 0, 1};

return idx;

}

void split(int u, int size, int &x, int &y) {

if (u == 0) x = y = 0;

else {

if (tr[u].lazy) pushdown(u);

if (tr[tr[u].l].size < size) {

x = u;

split(tr[u].r, size - tr[tr[u].l].size - 1, tr[u].r, y);

} else {

y = u;

split(tr[u].l, size, x, tr[u].l);

}

pushup(u);

}

}

int merge(int x, int y) {

if (x == 0 || y == 0) return x + y;

if (tr[x].key <= tr[y].key) {

if (tr[x].lazy) pushdown(x);

tr[x].r = merge(tr[x].r, y);

pushup(x);

return x;

} else {

if (tr[y].lazy) pushdown(y);

tr[y].l = merge(x, tr[y].l);

pushup(y);

return y;

}

}

void insert(int val) { root = merge(root, get\_Node(val));}

void prin(int u) {

if (u == 0) return;

if (tr[u].lazy) {

pushdown(u);

}

prin(tr[u].l);

cout << tr[u].val << " ";

prin(tr[u].r);

}

void rev(int l, int r) {

split(root, l - 1, x, y);

split(y, r - l + 1, y, z);

tr[y].lazy ^= 1;

root = merge(x, merge(y, z));

}

**SPLAY有序**

struct Tree {

int s[2], p, v;

int size, cnt;

void init(int \_v, int \_p) {

v = \_v;

p = \_p;

size = 1;

cnt = 1;

}

}tr[N];

int n, m, root, idx;

void pushup(int u) {

tr[u].size = tr[tr[u].s[0]].size + tr[tr[u].s[1]].size + tr[u].cnt;

}

void rotate(int x) {

int y = tr[x].p, z = tr[y].p;

int k = tr[y].s[1] == x;

tr[z].s[tr[z].s[1] == y] = x; tr[x].p = z;

tr[y].s[k] = tr[x].s[k ^ 1]; tr[tr[x].s[k ^ 1]].p = y;

tr[x].s[k ^ 1] = y; tr[y].p = x;

pushup(y);

pushup(x);

}

void splay(int x, int k) {

while (tr[x].p != k) {

int y = tr[x].p, z = tr[y].p;

if (z != k)

if ((tr[y].s[1] == x) ^ (tr[z].s[1] == y)) rotate(x);

else rotate(y);

rotate(x);

}

if (!k) root = x;

}

int find(int v) {

int u = root, p = 0;

while (u) {

if (tr[u].v >= v) {

p = u;

u = tr[u].s[0];

} else {

u = tr[u].s[1];

}

}

splay(p, 0);

return p;

}

int insert(int v) {

int u = root, p = 0;

while (u && tr[u].v != v) {

p = u;

u = tr[u].s[v > tr[u].v];

}

if (u) {

tr[u].cnt++;

} else {

u = ++idx;

if (p) {

tr[p].s[v > tr[p].v] = u;

}

tr[u].init(v, p);

}

splay(u, 0);

return u;

}

int get(int v, int f) { // 前驱 f == 0, 后继 f == 1

// find(v);

int u = find(v);

if (f && tr[u].v > v) return u;

if (!f && tr[u].v < v) return u;

u = tr[u].s[f];

while (tr[u].s[f ^ 1]) u = tr[u].s[f ^ 1];

return u;

}

void del(int x) {

int last = get(x, 0);

int next = get(x, 1);

splay(last, 0);

splay(next, last);

int u = tr[next].s[0];

if (tr[u].cnt > 1) {

tr[u].cnt--;

splay(u, 0);

} else {

tr[next].s[0] = 0;

splay(next, 0);

}

}

int get\_k(int x) {

int u = root;

while (u) {

if (tr[tr[u].s[0]].size >= x) u = tr[u].s[0];

else if (tr[tr[u].s[0]].size + tr[u].cnt >= x) return tr[u].v;

else {

x -= tr[tr[u].s[0]].size + tr[u].cnt;

u = tr[u].s[1];

}

}

return -1;

}

int main() {

scanf("%d", &m);

int L = insert(-MOD), R = insert(MOD);

while (m--) { int op, x; scanf("%d%d", &op, &x);

if (op == 1) {insert(x);}

else if (op == 2) {del(x);}

else if (op == 3) { printf("%d\n", tr[tr[find(x)].s[0]].size);}

else if (op == 4) { printf("%d\n", get\_k(x + 1));}

else if (op == 5) { printf("%d\n", tr[get(x, 0)].v);}

else { printf("%d\n", tr[get(x, 1)].v);}

}

return 0;

}

**SPLAY无序**

struct Tree {

int s[2], p, v;

int size, flag;

void init(int \_v, int \_p) {

v = \_v;

p = \_p;

}

}tr[N];

int n, m, root, idx;

void pushup(int u) {

tr[u].size = tr[tr[u].s[0]].size + tr[tr[u].s[1]].size + 1;

}

void pushdown(int u) {

if (tr[u].flag) {

swap(tr[u].s[0], tr[u].s[1]);

tr[tr[u].s[0]].flag ^= 1;

tr[tr[u].s[1]].flag ^= 1;

tr[u].flag = 0;

}

}

void rotate(int x) {

int y = tr[x].p, z = tr[y].p;

int k = tr[y].s[1] == x;

tr[z].s[tr[z].s[1] == y] = x; tr[x].p = z;

tr[y].s[k] = tr[x].s[k ^ 1]; tr[tr[x].s[k ^ 1]].p = y;

tr[x].s[k ^ 1] = y; tr[y].p = x;

pushup(y);

pushup(x);

}

void splay(int x, int k) {

while (tr[x].p != k) {

int y = tr[x].p, z = tr[y].p;

if (z != k)

if ((tr[y].s[1] == x) ^ (tr[z].s[1] == y)) rotate(x);

else rotate(y);

rotate(x);

}

if (!k) root = x;

}

void insert(int v) {

int u = root, p = 0;

while (u) {

p = u;

u = tr[u].s[v > tr[u].v];

}

u = ++idx;

if (p) tr[p].s[v > tr[p].v] = u;

tr[u].init(v, p);

splay(u, 0);

}

int get\_k(int k) {

int u = root;

while (true) {

pushdown(u);

if (tr[tr[u].s[0]].size >= k) u = tr[u].s[0];

else if (tr[tr[u].s[0]].size + 1 == k) return u;

else {

k -= tr[tr[u].s[0]].size + 1;

u = tr[u].s[1];

}

}

return -1;

}

void output(int u) {

pushdown(u);

if (tr[u].s[0]) output(tr[u].s[0]);

if (tr[u].v >= 1 && tr[u].v <= n) printf("%d ", tr[u].v);

if (tr[u].s[1]) output(tr[u].s[1]);

}

int main() {

scanf("%d%d", &n, &m);

for (int i = 0; i <= n + 1; i++) insert(i);

while (m--) {

int l, r; scanf("%d%d", &l, &r);

l = get\_k(l); r = get\_k(r + 2);

splay(l, 0); splay(r, l);

tr[tr[r].s[0]].flag ^= 1;

}

output(root);

return 0;

}

**动态开点**

struct Tree {

int l, r; int sum, lazy;

}tr[N];

int T, n, m, k, tot; int root;

void pushup(int u) { tr[u].sum = tr[tr[u].l].sum + tr[tr[u].r].sum;}

void pushdown(int u, int l, int r) {

if (tr[u].lazy) {

int mid = l + r >> 1;

int d = 1;

if (tr[u].lazy == 1) d = 0;

if (!tr[u].l) tr[u].l = ++tot;

if (!tr[u].r) tr[u].r = ++tot;

tr[tr[u].l].sum = (mid - l + 1) \* d;

tr[tr[u].r].sum = (r - (mid + 1) + 1) \* d;

tr[tr[u].l].lazy = tr[tr[u].r].lazy = tr[u].lazy;

tr[u].lazy = 0;

}

}

void modify(int &u, int L, int R, int l, int r, int lazy) {

if (!u) u = ++tot;

if (L >= l && R <= r) {

int d = 1;

if (lazy == 1) d = 0;

tr[u].sum = (R - L + 1) \* d;

tr[u].lazy = lazy;

return;

}

pushdown(u, L, R);

int mid = L + R >> 1;

if (l <= mid) modify(tr[u].l, L, mid, l, r, lazy);

if (r > mid) modify(tr[u].r, mid + 1, R, l, r, lazy);

pushup(u);

}

int query(int u, int L, int R, int l, int r) {

if (L >= l && R <= r) {

return tr[u].sum;

}

pushdown(u, L, R);

int mid = L + R >> 1;

int ans = 0;

if (l <= mid) ans += query(tr[u].l, L, mid, l, r);

if (r > mid) ans += query(tr[u].r, mid + 1, R, l, r);

return ans;

}

void solve() {

scanf("%d%d", &n, &m);

modify(root, 1, n, 1, n, 2);

while (m--) {

int l, r, k;

scanf("%d%d%d", &l, &r, &k);

modify(root, 1, n, l, r, k);

printf("%d\n", query(root, 1, n, 1, n));

}

}

**动态树**

struct Tree {

int s[2], p, v;

int sum, rev;

}tr[N];

int n, m; int stk[N];

bool isroot(int x) { return tr[tr[x].p].s[0] != x && tr[tr[x].p].s[1] != x;}

void calc(int u) {

swap(tr[u].s[0], tr[u].s[1]);

tr[u].rev ^= 1;

}

void pushup(int u) {

tr[u].sum = tr[tr[u].s[0]].sum ^ tr[tr[u].s[1]].sum ^ tr[u].v;

}

void pushdown(int u) {

if (tr[u].rev) {

calc(tr[u].s[0]);

calc(tr[u].s[1]);

tr[u].rev = 0;

}

}

void rotate(int x) {

int y = tr[x].p, z = tr[y].p;

int k = tr[y].s[1] == x;

if (!isroot(y)) tr[z].s[tr[z].s[1] == y] = x;

tr[x].p = z;

tr[y].s[k] = tr[x].s[k ^ 1]; tr[tr[x].s[k ^ 1]].p = y;

tr[x].s[k ^ 1] = y; tr[y].p = x;

pushup(y);

pushup(x);

}

void splay(int x) {

int top = 0, r = x;

stk[++top] = r;

while (!isroot(r)) {

stk[++top] = r = tr[r].p;

}

while (top) {

pushdown(stk[top--]);

}

while (!isroot(x)) {

int y = tr[x].p, z = tr[y].p;

if (!isroot(y))

if ((tr[y].s[1] == x) ^ (tr[z].s[1] == y)) rotate(x);

else rotate(y);

rotate(x);

}

}

void access(int x) {

int z = x;

for (int y = 0; x; y = x, x = tr[x].p) {

splay(x);

tr[x].s[1] = y;

pushup(x);

}

splay(z);

}

void makeroot(int x) {

access(x);

calc(x);

}

int findroot(int x) {

access(x);

while (tr[x].s[0]) {

pushdown(x);

x = tr[x].s[0];

}

splay(x);

return x;

}

void split(int x, int y) {

makeroot(x);

access(y);

}

void link(int x, int y) {

makeroot(x);

if (findroot(y) != x) tr[x].p = y;

}

void cut(int x, int y) {

makeroot(x);

if (findroot(y) == x && tr[y].p == x && !tr[y].s[0]) {

tr[x].s[1] = tr[y].p = 0;

pushup(x);

}

}

int main() {

scanf("%d%d", &n, &m);

for (int i = 1; i <= n; i++) { scanf("%d", &tr[i].v); }

while (m--) {

int t, x, y;

scanf("%d%d%d", &t, &x, &y);

if (t == 0) {

split(x, y);

printf("%d\n", tr[y].sum);

} else if (t == 1) {

link(x, y);

} else if (t == 2) {

cut(x, y);

} else {

splay(x);

tr[x].v = y;

pushup(x);

}

}

return 0;

}

**可持久化trie**

int T, n, m, k;

int tr[M][2], root[N], max\_id[M], idx; int s[N];

void insert(int i, int k, int p, int q) {

if (k < 0) {

max\_id[q] = i;

return;

}

int t = s[i] >> k & 1;

if (tr[p][t ^ 1]) tr[q][t ^ 1] = tr[p][t ^ 1];

tr[q][t] = ++idx;

insert(i, k - 1, tr[p][t], tr[q][t]);

max\_id[q] = max(max\_id[tr[q][0]], max\_id[tr[q][1]]);

}

int query(int x, int root, int sum) {

int p = root;

for (int i = 23; i >= 0; i--) {

int t = sum >> i & 1;

if (max\_id[tr[p][t ^ 1]] >= x) {

p = tr[p][t ^ 1];

} else {

p = tr[p][t];

}

}

return sum ^ s[max\_id[p]];

}

void solve() {

cin >> n >> m;

max\_id[0] = -1;

root[0] = ++idx;

insert(0, 23, 0, root[0]);

for (int i = 1; i <= n; i++) {

cin >> s[i];

s[i] = s[i] ^ s[i - 1];

root[i] = ++idx;

insert(i, 23, root[i - 1], root[i]);

}

while (m--) {

char ops[3];

cin >> ops;

if (\*ops == 'A') {

int x; cin >> x;

++n;

s[n] = x ^ s[n - 1]; root[n] = ++idx;

insert(n, 23, root[n - 1], root[n]);

} else {

int l, r, x; cin >> l >> r >> x;

cout << query(l - 1, root[r - 1], s[n] ^ x) << endl;

}

}

}

**树链部分**

struct Tree {

int l, r;

int sum, maxs;

}tr[N \* 4];

int n, m, tot; int h[N], e[M], w[N], ne[M], idx;

int sz[N], fa[N], dep[N], id[N]; int l[N], r[N], hs[N], top[N];

void pushup(int u) {

tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;

tr[u].maxs = max(tr[u << 1].maxs, tr[u << 1 | 1].maxs);

}

void build(int u, int l, int r) {

if (l == r) { tr[u] = {l, r, w[id[l]], w[id[l]]};}

else {

tr[u] = {l, r};

int mid = l + r >> 1;

build(u << 1, l, mid); build(u << 1 | 1, mid + 1, r);

pushup(u);

}

}

void modify(int u, int x, int v) {

if (tr[u].l >= x && tr[u].r <= x) {

tr[u].sum = tr[u].maxs = v;

return;

}

int mid = tr[u].l + tr[u].r >> 1;

if (x <= mid) modify(u << 1, x, v);

if (x > mid) modify(u << 1 | 1, x, v);

pushup(u);

}

int query\_maxs(int u, int l, int r) {

if (tr[u].l >= l && tr[u].r <= r) { return tr[u].maxs; }

int mid = tr[u].l + tr[u].r >> 1;

int maxs = -1e9;

if (l <= mid) maxs = max(maxs, query\_maxs(u << 1, l, r));

if (r > mid) maxs = max(maxs, query\_maxs(u << 1 | 1, l, r));

return maxs;

}

int query\_sum(int u, int l, int r) {

if (tr[u].l >= l && tr[u].r <= r) { return tr[u].sum; }

int mid = tr[u].l + tr[u].r >> 1;

int sum = 0;

if (l <= mid) sum += query\_sum(u << 1, l, r);

if (r > mid) sum += query\_sum(u << 1 | 1, l, r);

return sum;

}

void add(int a, int b) {e[idx] = b; ne[idx] = h[a]; h[a] = idx++;}

void dfs1(int u, int f) {

sz[u] = 1;

hs[u] = -1;

fa[u] = f;

dep[u] = dep[f] + 1;

for (int i = h[u]; i != -1; i = ne[i]) {

int j = e[i];

if (j == f) continue;

dfs1(j, u);

sz[u] += sz[j];

if (hs[u] == -1 || sz[hs[u]] < sz[j]) {

hs[u] = j;

}

}

}

void dfs2(int u, int t) {

l[u] = ++tot;

id[tot] = u;

top[u] = t;

if (hs[u] != -1) {

dfs2(hs[u], t);

}

for (int i = h[u]; i != -1; i = ne[i]) {

int j = e[i];

if (j == fa[u] || j == hs[u]) continue;

dfs2(j, j);

}

r[u] = tot;

}

int q\_maxs(int a, int b) {

int maxs = -1e9;

while (top[a] != top[b]) {

if (dep[top[a]] > dep[top[b]]) {

maxs = max(maxs, query\_maxs(1, l[top[a]], l[a]));

a = fa[top[a]];

} else {

maxs = max(maxs, query\_maxs(1, l[top[b]], l[b]));

b = fa[top[b]];

}

}

if (dep[a] >= dep[b]) {

maxs = max(maxs, query\_maxs(1, l[b], l[a]));

} else {

maxs = max(maxs, query\_maxs(1, l[a], l[b]));

}

return maxs;

}

int q\_sum(int a, int b) {

int sum = 0;

while (top[a] != top[b]) {

if (dep[top[a]] > dep[top[b]]) {

sum += query\_sum(1, l[top[a]], l[a]);

a = fa[top[a]];

} else {

sum += query\_sum(1, l[top[b]], l[b]);

b = fa[top[b]];

}

}

if (dep[a] >= dep[b]) {

sum += query\_sum(1, l[b], l[a]);

} else {

sum += query\_sum(1, l[a], l[b]);

}

return sum;

}

int main() {

cin >> n; memset(h, -1, sizeof h);

for (int i=0;i< n - 1; i++){int a, b;cin >> a >> b;add(a, b);add(b, a); }

for (int i = 1; i <= n; i++) { cin >> w[i]; }

dfs1(1, 0); dfs2(1, 1); build(1, 1, n);

cin >> m;

while (m--) {

char ops[10]; cin >> ops;

if (ops[1] == 'M') {

int a, b; cin >> a >> b;

cout << q\_maxs(a, b) << endl;

} else if (ops[1] == 'S') {

int a, b; cin >> a >> b;

cout << q\_sum(a, b) << endl;

} else if (ops[1] == 'H') {

int x, v; cin >> x >> v;

modify(1, l[x], v);

}

}

}

**主席树**

struct Tree {

int l, r;

int cnt;

}tr[N \* 20];

int T, n, m, k; int a[N], root[N], idx; vector<int> ve[N];

void pushup(int u) { tr[u].cnt = tr[tr[u].l].cnt + tr[tr[u].r].cnt;}

int build(int l, int r) {

int p = ++idx;

if (l == r) return p;

int mid = l + r >> 1;

tr[p].l = build(l, mid); tr[p].r = build(mid + 1, r);

return p;

}

int insert(int p, int l, int r, int x) {

int q = ++idx; tr[q] = tr[p];

if (l == r) { tr[q].cnt++; return q;}

int mid = l + r >> 1;

if (x <= mid) tr[q].l = insert(tr[p].l, l, mid, x);

else tr[q].r = insert(tr[p].r, mid + 1, r, x);

pushup(q);

return q;

}

int query(int p, int q, int l, int r, int L, int R) {

if (l >= L && r <= R) { return tr[q].cnt - tr[p].cnt;}

int mid = l + r >> 1; int ans = 0;

if (L <= mid) ans += query(tr[p].l, tr[q].l, l, mid, L, R);

if (R > mid) ans += query(tr[p].r, tr[q].r, mid + 1, r, L, R);

return ans;

}

void solve() {

cin >> n >> k; root[0] = build(0, n);

for (int i = 1; i <= n; i++) {

cin >> a[i]; ve[a[i]].push\_back(i);

int t = 0;

if (ve[a[i]].size() > k) {

t = ve[a[i]][ve[a[i]].size() - k - 1];

}

root[i] = insert(root[i - 1], 0, n, t);

}

cin >> m; int last = 0;

while (m--) {

int l, r; cin >> l >> r;

l = (l + last) % n + 1; r = (r + last) % n + 1;

if (l > r) swap(l, r);

last = query(root[l - 1], root[r], 0, n, 0, l - 1);

cout << last << endl;

}

}