LCT(Ordered)

```
struct Node
{
	Node *left;
	Node *right;
	Node *parent;
	Node();
	inline bool isRoot();
};
const int maxn = 1e5;
Node pool[maxn], *null = pool;
int node_cnt = 1;
Node::Node(): left(null), right(null), parent(null) { }
inline bool Node::isRoot()
{
	return parent == null
		|| /* have light edge to parent */(parent->left != this && parent->right != this);
}
Node *newNode()
{
	pool[node_cnt].left = null;
	pool[node_cnt].right = null;
	pool[node_cnt].parent = null;
	return pool + node_cnt++;
};
// connect ch -> p (isLeftChild?)
inline void connect(Node *ch, Node *p, int isLeftChild)
{
	if (ch != null)
		ch->parent = p;
	if (isLeftChild >= 0)
	{
		if (isLeftChild)
			p->left = ch;
		else
			p->right = ch;
	}
}
inline void rotate(Node *x)
{
	Node *p = x->parent;
	Node *g = p->parent;
	bool isRootP = p->isRoot();
	bool leftChildX = (x == p->left);

	connect(leftChildX? x->right: x->left, p, leftChildX);
	connect(p, x, !leftChildX);
	connect(x, g, !isRootP? (p == g->left): -1);
}
```

```
// bring x to the root
void splay(Node *x)
{
	while (!x->isRoot())
	{
		Node *p = x->parent;
		Node *g = p->parent;
		if (!p->isRoot())
			rotate((x == p->left) == (p == g->left)? /* zig-zig */p: /* zig-zag
*/x);
		rotate(x);
	}
}

// make node x the root of the virtual tree
void expose(Node *x)
{
	Node *last = null;
	for (Node *y = x; y != null; /* go with light edge */y = y->parent)
	{
		splay(y);
		/* modify heavy edge from y */
		y->right = last;
		last = y;
	}
	// bring x to the root for update
	splay(x);
}

// the root of the tree(where x is)
Node *findRoot(Node *x)
{
	expose(x);

	while (x->left != null)
		x = x->left;
	// bring to splay's root
	splay(x);

	return x;
}
// the parent of x
Node *parent(Node *x)
{
	expose(x);
	x = x->left;
	while (x->right != null)
		x = x->right;
	return x;
}
```

```
// link x to y (y be the parent of x)
bool link(Node *x, Node *y)
{
	if (findRoot(x) == findRoot(y))
		return false;
	expose(x);
	// x is not root now
	if (x->left != null)
		return false;
	x->parent = y;
	return true;
}

// cut the edge between x and x's parent
bool cut(Node *x)
{
	expose(x);
	// x is root now
	if (x->left == null)
		return false;
	x->left->parent = null;
	x->left = null;
}
int n, m;

void init()
{
	node_cnt = 1;
	for (int i = 1; i <= n; i++)
		newNode();
	for (int i = 1; i <= n; i++)
	{
		int parent;
		scanf("%d", &parent);
		if (parent)
			link(pool + i, pool + parent);
	}
}

char cmd[20];
void solve()
{
	scanf("%d", &m);
	for (int i = 0; i < m; i++)
	{
		scanf("%s", cmd);
		if (cmd[0] == 'Q')
		{
			int x;
			scanf("%d", &x);
			printf("%d\n", findRoot(pool + x) - pool);
		}
```

```
            else
            {
                int x, y;
                scanf("%d%d", &x, &y);
                Node *px = parent(pool + x);
                if (px != null)
                    cut(pool + x);
                if (y)
                {
                    if (!link(pool + x, pool + y) && px != null)
                        link(pool + x, px);
                }
            }
        }
}

int main()
{
    bool first = true;
    while (~scanf("%d", &n))
    {
        if (first) first = false; else puts("");
        init();
        solve();
    }
}
```

LCT(Unordered)

```
struct Node
{
    Node *left;
    Node *right;
    Node *parent;
    int value;
    int sum;
    int size;
    int delta;
    bool reverse;
    Node();
    Node(int value);
    inline bool isRoot();
    inline void push();
    inline void update();
};
const int maxn = 1e5;
Node pool[maxn], *null = pool;
int node_cnt = 1;
Node::Node(): value(0)
{
    left = null;
    right = null;
    parent = null;
```

```
	sum = 0;
	size = 0;
	delta = 0;
	reverse = false;
}
inline bool Node::isRoot()
{
	return parent == null
		|| /* have light edge to parent */(parent->left != this &&
parent->right != this);
}
inline void Node::push()
{
	if (reverse)
	{
		reverse = false;
		swap(left, right);
		if (left != null)
			left->reverse = !left->reverse;
		if (right != null)
			right->reverse = !right->reverse;
	}
	if (left != null)
		left->delta += delta;
	if (right != null)
		right->delta += delta;
	value += delta;
	sum += delta * size;
	delta = 0;
}
inline void Node::update()
{
	sum = value;
	size = 1;
	sum += left->sum;
	size += left->size;
	sum += right->sum;
	size += right->size;
}
Node *newNode()
{
	pool[node_cnt].left = null;
	pool[node_cnt].right = null;
	pool[node_cnt].parent = null;
	return pool + node_cnt++;
};
// connect ch -> p (isLeftChild?)
inline void connect(Node *ch, Node *p, int isLeftChild)
{
	if (ch != null)
		ch->parent = p;
	if (isLeftChild >= 0)
```

```
	{
		if (isLeftChild)
			p->left = ch;
		else
			p->right = ch;
	}
}
inline void rotate(Node *x)
{
	Node *p = x->parent;
	Node *g = p->parent;
	bool isRootP = p->isRoot();
	bool leftChildX = (x == p->left);

	connect(leftChildX? x->right: x->left, p, leftChildX);
	connect(p, x, !leftChildX);
	connect(x, g, !isRootP? (p == g->left): -1);

	p->update();
}

// bring x to the root
void splay(Node *x)
{
	while (!x->isRoot())
	{
		Node *p = x->parent;
		Node *g = p->parent;
		if (!p->isRoot())
			g->push();
		p->push();
		x->push();
		if (!p->isRoot())
			rotate((x == p->left) == (p == g->left)? /* zig-zig */p: /* zig-zag
*/x);
		rotate(x);
	}
	x->push();
	x->update();
}

// make node x the root of its aux tree
// == access(x)
void expose(Node *x)
{
	Node *last = null;
	for (Node *y = x; y != null; /* go with light edge */y = y->parent)
	{
		splay(y);
		/* modify heavy edge from y */
		y->right = last;
		last = y;
```

```
        }
        // bring x to the root for update
        splay(x);
}

// make the node as the root of its represention tree
// == evert
void makeRoot(Node *x)
{
        expose(x);
        x->reverse = !x->reverse;
}

// x and y connected?
bool connected(Node *x, Node *y)
{
        if (x == y)
                return true;
        expose(x);
        expose(y);
        // x->parent must be not null if connected
        return x->parent != null;
}

// link x to y (y be the parent of x)
bool link(Node *x, Node *y)
{
        if (connected(x, y))
                return false;
        makeRoot(x);
        // add a light edge between x and y
        x->parent = y;
        return true;
}

// cut the edge between x and x's parent
bool cut(Node *x, Node *y)
{
        makeRoot(x);
        expose(y);
        // not exist edge(x, y)
        if (y->left != x || x->left != null || x->right != null)
                return false;
        x->parent = null;
        y->left = null;
        return true;
}

int sum(Node *x, Node *y)
{
        if (!connected(x, y))
                return 0;
```

```
	makeRoot(x);
	expose(y);
	return y->sum;
}

bool add(Node *x, Node *y, int delta)
{
	if (!connected(x, y))
		return false;
	makeRoot(x);
	expose(y);
	y->delta += delta;
	return true;
}
```