Link Cut Tree(LCT)

用于求解动态连通树，维护森林的连通性问题。

用splay来维护虚边和实边

如果没有删边操作的话，树链剖分的效率更优一些。

均摊时间复杂度 $O(\log^2 n)$,Tarjan是这么说的。

支持以下操作：

1. 查询、修改链上的信息（最值，总和等）
2. 随意指定原树的根（即换根）
3. 动态连边、删边
4. 合并两棵树、分离一棵树（其实也就是第三点）
5. 动态维护连通性

原题链接

P3690 【模板】动态树（Link Cut Tree）

具体操作

$access$

LCT核心操作，由于要实链剖分，所以我们并不能保证操作的两点在同一颗splay上，所以需要打通指定节点到根节点，形成一条以根节点为起点，目标节点为终点的实链，该链节点的深度在中序遍历意义下递增，在经过此操作后，目标节点一定是LCT中，根节点所在splay中,中序遍历最后的点。

这也是LCT的主要时间消耗部分，均摊复杂度 $O(\log n)$

$make\ root$

当我们要获取两个节点之间路径的信息或者对路径进行操作时，为确保两个目标节点在同一颗splay下成为祖孙关系,于是需要进行换根操作

$find\ root$

找到目标节点所在连通树的根节点，将目标节点置为根节点后，不停的找左子树，因为原来的树根深度一定是最小的。

$split$

目的是为了访问和操作链，将 x - y 节点拉出一条路径形成一颗splay,此时y为根节点，x在y的左子树上

$link$

连一条 x - y 的边，将x置为该连通树的根节点，然后从 x 向 y 连一条虚边，连之间记得判断两点是否在同一颗连通树内，不然可能会出现环

cut

将 x - y 的边断开

完整代码

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5 + 5;

struct Node {
    int val, res;
    int ch[2], fa;
    bool lazy;
} tree[maxn];

int &ls(int x) { return tree[x].ch[0]; }
int &rs(int x) { return tree[x].ch[1]; }
int &fa(int x) { return tree[x].fa; }

//查询x是f的左儿子还是右儿子，0表示左儿子，1表示右儿子，用于判断旋转方向
bool ident(int x, int f) { return rs(f) == x; }

//建立父子关系，s为0表示为左儿子，反之为右儿子
void connect(int x, int f, bool s) {
    fa(x) = f;
    if (!s)
        ls(f) = x;
    else
        rs(f) = x;
}

//更新节点信息
void update(int x) {
    tree[x].res = tree[ls(x)].res ^ tree[rs(x)].res ^ tree[x].val;
}

bool notRoot(int x) {
    int f = fa(x);
    return ls(f) == x || rs(f) == x;
}

void reverse(int x) {
    swap(ls(x), rs(x));
    tree[x].lazy ^= 1;
}

void pushDown(int x) {
    if (tree[x].lazy) {
        if (ls(x))
            reverse(ls(x));
        if (rs(x))
            reverse(rs(x));
    }
    tree[x].lazy = 0;
```

```
}

//下放x到所在树根节点路径上的所有懒标记
void pushAll(int x) {
    if (notRoot(x))
        pushAll(fa(x));
    pushDown(x);
}

//旋转x
void rotate(int x) {
    int f = fa(x), ff = fa(f);
    bool k = ident(x, f);
    connect(tree[x].ch[!k], f, k);
    fa(x) = ff;
    if (notRoot(f))
        tree[ff].ch[ident(f, ff)] = x;
    connect(f, x, !k);
    update(f), update(x);
}

//将x旋转到原树根节点
void splaying(int x) {
    pushAll(x);
    while (notRoot(x)) {
        int f = fa(x), ff = fa(f);
        if (notRoot(f)) {
            if (ident(x, f) ^ ident(f, ff))
                rotate(x);
            else
                rotate(f);
        }
        rotate(x);
    }
}

void access(int x) {
    //将x到原树根节点构建一条实链
    // x到整个LCT路径上的点作为所在树的根节点
    //并没有将x换成整颗LCT的根节点
    //即让x和LCT的根在一个splay中
    for (int y = 0; x; y = x, x = fa(x)) {
        // y为原树的根节点
        splaying(x);
        rs(x) = y; //将y接到x的右儿子上
        update(x);
    }
}

void makeRoot(int x) {
    //给LCT换根
    access(x);   //将x到LCT根节点的路径构造为一条实链
    splaying(x); //将x换成LCT根节点
    reverse(x);  //翻转深度顺序
}
```

```
int findRoot(int x) {
    //找到x所在树的根节点
    access(x);   //将x到LCT根节点打通，此时x一定在splay的最右
    splaying(x); //将x换成LCT根节点
    //找到原LCT最左侧的节点（中序遍历意义下的左）
    while (ls(x)) {
        pushDown(x);
        x = ls(x);
    }
    splaying(x); //将LCT的根换回去，加速下次查询
    return x;
}

void link(int x, int y) {
    makeRoot(x);           //将x置为LCT的根
    if (findRoot(y) == x) //如果y的根是x，说明x和y已经连通
        return;
    fa(x) = y; //否则在x与y之间连一条虚边
}

//题目不保证断边一定合法时
void cut(int x, int y) {
    makeRoot(x);
    if (findRoot(y) != x || fa(y) != x || ls(y))
        return;         //如果y不是x的子节点，说明x和y不连通
    fa(y) = rs(x) = 0; //否则切断x与y之间的实边
    update(x);
}

//题目保证断边一定合法时
void cut(int x, int y) {
    split(x, y);
    fa(x) = ls(y) = 0;
    update(y);
}

void split(int x, int y) {
    //将x到y的路径分离出来
    makeRoot(x); //将x置为LCT的根
    access(y);   //将y到x的路径构造为一条实链
    splaying(y); //将y换成LCT根节点
    //此时y没有右儿子，y到x的路径就是所求路径
}

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; ++i)
        cin >> tree[i].val;
    while (m--) {
        int opt, x, y;
        cin >> opt >> x >> y;
        if (opt == 0) {
            split(x, y);
```

```
            cout << tree[y].res << endl;
        } else if (opt == 1)
            link(x, y);
        else if (opt == 2)
            cut(x, y);
        else {
            splaying(x);
            tree[x].val = y;
        }
    }
}
```