# 题目

给定一棵二叉树的根节点 root 和一个叶节点 leaf ，更改二叉树，使得 leaf 为新的根节点。

你可以按照下列步骤修改从 leaf 到 root 的路径中除 root 外的每个节点 cur ：

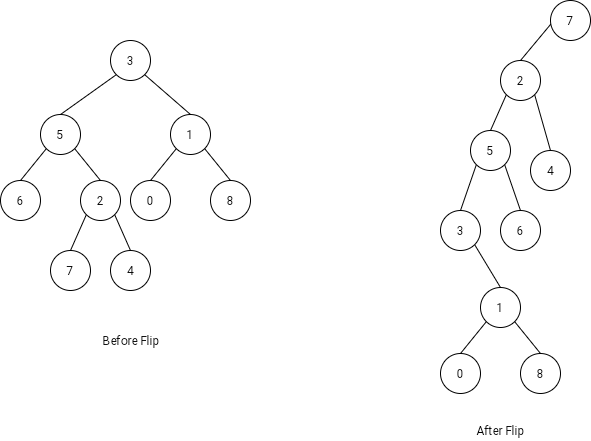
1、如果 cur 有左子节点，则该子节点变为 cur 的右子节点。注意我们保证 cur 至多有一个子节点。

2、cur 的原父节点变为 cur 的左子节点。

返回修改后新树的根节点。

注意：确保你的答案在操作后正确地设定了 Node.parent （父节点）指针，否则会被判为错误答案。

示例 1:



输入: root = [3,5,1,6,2,0,8,null,null,7,4], leaf = 7

输出: [7,2,null,5,4,3,6,null,null,null,1,null,null,0,8]

示例 2:

输入: root = [3,5,1,6,2,0,8,null,null,7,4], leaf = 0

输出: [0,1,null,3,8,5,null,null,null,6,2,null,null,7,4]

提示:

树中节点的个数在范围 [2, 100] 内。

-109 <= Node.val <= 109

所有的 Node.val 都是唯一的。

leaf 存在于树中。

# 分析

思路：首先自底向上地从叶子结点开始打破子节点和父节点的双向指针联系。然后如果按照flip后得到结果树的视角去看再自底向上地重新构建子节点和父节点的双向指针关系。

class Solution {

public:

Node\* flipBinaryTree(Node\* root, Node \* leaf) {

flip(leaf);

return leaf;

}

void flip(Node\* cur){

if (cur == nullptr) return;

Node\* parent = cur -> parent;

//断开当前节点和父节点的联系

if (parent){

if (cur == parent -> left)

parent -> left = nullptr;

else

parent -> right = nullptr;

cur -> parent = nullptr;

}

//递归地去全部“打断”当前节点和父节点的联系

flip(parent);

//flip转换父子关系，重新组装

if (parent){

if (cur -> left)//如果有左孩子需要保留，移动到右孩子的位置上保存

cur -> right = cur -> left;

cur -> left = parent;

parent -> parent = cur;

}

}

};

/\*

// Definition for a Node.

class Node {

public:

int val;

Node\* left;

Node\* right;

Node\* parent;

};

\*/

class Solution

{

public:

Node\* flipBinaryTree(Node\* root, Node \* leaf)

{

if (root == leaf)

return root;

leaf -> left = leaf->parent;

leaf->parent = NULL;

Node\* p = leaf;

while (p->left != root) //5不是根

{

if (p->left->right == p) //7是5的右子

p->left -> right = p->left->left; //6要变成5的右子

p->left -> left = p->left->parent; //5的左子成了3

p->left -> parent = p; //5的父成了7

p = p->left; //7要往上走，到了5的位置。

}

if (p->left->left == p) //是root的左子

p->left -> left = NULL;

else //是root的右子

p->left->right = NULL;

p->left->parent = p; //root的父变成了leaf

return leaf;

}

};

/\*

// Definition for a Node.

class Node {

public:

int val;

Node\* left;

Node\* right;

Node\* parent;

};

\*/

class Solution {

public:

Node\* flipBinaryTree(Node\* root, Node\* leaf) {

auto p = leaf;

vector<Node\*> paths;

while (p) {

paths.emplace\_back(p);

p = p->parent;

}

for (int i = 0; i < paths.size() - 1; ++i) {

auto cur = paths[i], parent = paths[i + 1];

if (parent->left == cur) parent->left = nullptr;

else parent->right = nullptr;

if (cur->left) cur->right = cur->left;

cur->left = parent;

parent->parent = cur;

}

leaf->parent = nullptr;

return leaf;

}

};