# 题目

翻转一棵二叉树。

**示例：**

**输入：**

4

/ \

2 7

/ \ / \

1 3 6 9

**输出：**

4

/ \

7 2

/ \ / \

9 6 3 1

**类似题目：**剑指offer 27

# 分析

## 方法一：递归法

**思路：**

显然，我们从根节点开始，递归地对树进行遍历，并从叶子节点先开始翻转。如果当前遍历到的节点root的左右两棵子树都已经翻转，那么我们只需要交换两棵子树的位置，即可完成以root为根节点的整棵子树的翻转。

**分析：**

class Solution {

public:

TreeNode\* invertTree(TreeNode\* root) {

if(nullptr == root)

return NULL;

TreeNode \*left = invertTree(root->left);

TreeNode \*right = invertTree(root->right);

root->left = right;

root->right = left;

return root;

}

};

**复杂度分析：**

时间复杂度：O(N)，其中N为二叉树节点的数目。我们会遍历二叉树中的每一个节点，对每个节点而言，我们在常数时间内交换其两棵子树。

空间复杂度：O(N)。使用的空间由递归栈的深度决定，它等于当前节点在二叉树中的高度。在平均情况下，二叉树的高度与节点个数为对数关系，即O(logN)。而在最坏情况下，树形成链状，空间复杂度为O(N)。

## 方法二：迭代法

### 思路一：栈模拟

**思路：**

模型：栈模拟二叉树的先序遍历

循环结束条件：栈为空

实现操作：交换栈顶结点的左右子树

**代码**

class Solution {

public:

TreeNode\* invertTree(TreeNode\* root) {

stack<TreeNode\*> s;

s.push(root);

while (!s.empty()) {

TreeNode\* node = s.top();

s.pop();

if (node == NULL) {

continue;

}

swap(node->left, node->right);

s.push(node->left);

s.push(node->right);

}

return root;

}

};

### 思路二：队列模拟

**思路：**

模型：使用队列模拟二叉树的层次遍历（实质上就是层次遍历）

循环结束条件：队列为空

实现操作：交换队首结点的左右子树

**代码：**

class Solution {

public:

TreeNode\* invertTree(TreeNode\* root) {

queue<TreeNode\*> q;

q.push(root);

while (!q.empty()) {

TreeNode\* node = q.front();

q.pop();

if (node == NULL) {

continue;

}

swap(node->left, node->right);

q.push(node->left);

q.push(node->right);

}

return root;

}

};