基于栈的DFS来遍历（基于队列的BFS??

1.二叉树镜像：

法一：第一步交换根节点的左右子树，之后交换左右子树自己的左右子树，这样的操作除了操作对象不同，但是操作步骤是一样，所以非常时候采用递归来做，我们用递归方式来递归交换各个左右子树。

/\*\*

\* 递归实现

\* 关键就在于把大问题转化为子问题即可

\* @param root

\*/

public static void Mirror1(TreeNode root) {

if(root != null) {

TreeNode temp = root.left;

root.left = root.right;

root.right = temp;

Mirror1(root.left);

Mirror1(root.right);

}

}

法二：

如果理解了思路一，我们可以发现只要采取某一种遍历方式去遍历树的所有的非叶子节点，然后交换它的左右子树，当遍历结束，即可得到二叉树的镜像。

仔细观察思路一的代码，这不就是递归实现的先序遍历的变形嘛！。当然我们不必非得采用先序，中序或者后序遍历节点。其实DFS或者BFS都可以的，只要能够保证所有的非叶子节点都访问一次即可。以下我们采用基于栈的DFS来遍历（基于队列的BFS感兴趣的可以自行尝试哦）：

/\*\*

\* 非递归实现, 借助栈或者列表遍历树中每一个非叶子节点

\* 交换其左右子树即可

\* @param root

\*/

public static void Mirror2(TreeNode root) {

if(root == null) {

return;

}

Stack<TreeNode> stack = new Stack<>();

stack.push(root);

while(!stack.isEmpty()) {

TreeNode node = stack.pop();

TreeNode temp = node.left;

node.left = node.right;

node.right = temp;

if(node.left != null) {

stack.push(node.left);

}

if(node.right != null) {

stack.push(node.right);

}

}

}

什么是无向树的直径？就是最长的两点间最短距离。也就是说，对于某两个点i和j，它们间的最短距离最大。而这就是无向图的直径。----其实还是不懂这个解释

<https://blog.csdn.net/WenDavidOI/article/details/50670056>

打印从左往右看到的二叉树（左视图，就是打印每层第一个节点）

**class** Tree {

TreeNode last;

TreeNode nlast;

**public** **void** printTree(TreeNode root) {

Queue<TreeNode> queue = **new** LinkedList<>();

queue.offer(root);

last = root; // 记录当前层最后一个节点

nlast = root; // 记录下一层的最后一个节点

**int** flag = 1;

**while** (!queue.isEmpty()) {

TreeNode t = queue.peek();

**if** (flag == 1) {

System.***out***.print(queue.poll().val + " ");

flag = 0;

} **else** {

queue.poll();

}

**if** (t.left != **null**) {

queue.offer(t.left);

nlast = t.left;

}

**if** (t.right != **null**) {

queue.offer(t.right);

nlast = t.right;

}

// 如果当前输出结点是最右结点，那么换行

**if** (last == t) {

flag = 1;

last = nlast;

}

}

}

}

# 给定一棵二叉树，和一个数值。求二叉树的路径和等于给定值的所有路径

//判断是否有路径

bool hasPathSum(TreeNode root, int sum) {

if(root==NULL)

return false;

sum -= root.val;

if(root.left==NULL&&root.right==NULL&&sum==0)

return true;

return hasPathSum(root.left,sum)||hasPathSum(root.right,sum);

}