# 题目

给定一个完美二叉树 ，其所有叶子节点都在同一层，每个父节点都有两个子节点。二叉树定义如下：

struct Node {

int val;

Node \*left;

Node \*right;

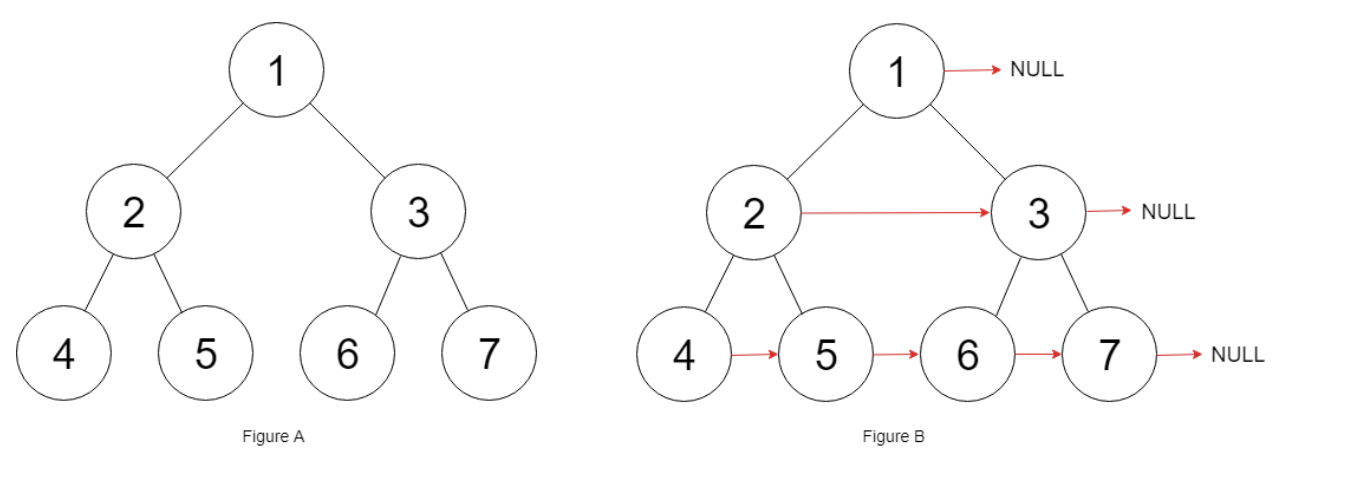
Node \*next;

}

填充它的每个 next 指针，让这个指针指向其下一个右侧节点。如果找不到下一个右侧节点，则将 next 指针设置为 NULL。

初始状态下，所有 next 指针都被设置为 NULL。

示例 1：



输入：root = [1,2,3,4,5,6,7]

输出：[1,#,2,3,#,4,5,6,7,#]

解释：给定二叉树如图 A 所示，你的函数应该填充它的每个 next 指针，以指向其下一个右侧节点，如图 B 所示。序列化的输出按层序遍历排列，同一层节点由 next 指针连接，'#' 标志着每一层的结束。

示例 2:

输入：root = []

输出：[]

提示：

树中节点的数量在 [0, 212 - 1] 范围内

-1000 <= node.val <= 1000

进阶：

你只能使用常量级额外空间。

使用递归解题也符合要求，本题中递归程序占用的栈空间不算做额外的空间复杂度。

# 分析

## 方法一：递归（推荐）

递归实现可以按层次递归处理。首先处理根节点，然后递归处理左右子树，确保它们的 next 指针正确连接。具体代码如下：

class Solution {

public:

Node\* connect(Node\* root) {

if (!root) {

return nullptr;

}

connectTwoNodes(root->left, root->right);

return root;

}

void connectTwoNodes(Node\* node1, Node\* node2) {

if (!node1 || !node2) {

return;

}

// 连接两个节点

node1->next = node2;

// 递归连接左右子树的对应节点

connectTwoNodes(node1->left, node1->right);

connectTwoNodes(node2->left, node2->right);

connectTwoNodes(node1->right, node2->left);

}

};

这段代码的思路是，首先连接当前节点的左右子节点，然后递归地连接左子节点的右子节点和右子节点的左子节点，最后递归处理左右子树。

## 方法二：迭代

使用迭代方法的示例代码：

class Solution {

public:

Node\* connect(Node\* root) {

if (!root) {

return nullptr;

}

Node\* leftmost = root;

while (leftmost->left) {

Node\* head = leftmost;

while (head) {

head->left->next = head->right;

if (head->next) {

head->right->next = head->next->left;

}

head = head->next;

}

leftmost = leftmost->left;

}

return root;

}

};

这段代码的思路是，从根节点开始，每次迭代处理一层，将当前节点的左子节点指向右子节点，并且将当前节点的右子节点指向下一个节点的左子节点（如果下一个节点存在的话）。最后返回根节点即可。

## 方法三：层次遍历

思路：

广度优先搜索（BFS）的思路，通过队列来遍历每一层的节点。具体步骤如下：

1、初始化一个队列 q，并将根节点 root 入队。

2、进入循环，直到队列为空。

3、在循环中，首先获取当前队列的大小 size，表示当前层的节点数。

4、初始化一个指针 prev，用来记录上一个处理过的节点，默认为 nullptr。

5、遍历当前层的节点，依次出队并处理：

- 如果 prev 不为空，则将 prev->next 指向当前节点 current。

- 如果当前节点有左子节点，则将左子节点入队。

- 如果当前节点有右子节点，则将右子节点入队。

- 更新 prev 为当前节点 current。

6、返回根节点 root。

这样，当循环结束时，每个节点的 next 指针都指向了其右侧节点，完成了填充每个节点的 next 指针的操作。

代码：

class Solution {

public:

Node\* connect(Node\* root) {

if (!root) {

return nullptr;

}

queue<Node\*> q;

q.push(root);

while (!q.empty()) {

int size = q.size();

Node\* prev = nullptr;

for (int i = 0; i < size; i++) {

Node\* current = q.front();

q.pop();

if (prev) {

prev->next = current;

}

if (current->left) {

q.push(current->left);

}

if (current->right) {

q.push(current->right);

}

prev = current;

}

}

return root;

}

};