# 题目

给定一个二叉树，找出其最小深度。

最小深度是从根节点到最近叶子节点的最短路径上的节点数量。

说明：叶子节点是指没有子节点的节点。

**示例：**

给定二叉树 [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

返回它的最小深度  2.

# 分析

## 方法一：递归

**思路：**

104题最大深度：

https://leetcode-cn.com/problems/maximum-depth-of-binary-tree

//错误示范, 在碰到[0],[1,2]的时候直接错

class Solution {

public:

int minDepth(TreeNode\* root) {

if ( root == nullptr){

return 0;

}

int left = minDepth(root->left);

int right = minDepth(root->right);

return min(left, right) + 1;

}

};

后来看了一下题解，才明白了最大深度和最小深度有一个很大的区别，最大深度能够保证最后一个节点绝对是叶子节点，而最小深度不行。

因为定义说最小深度是从根节点到最近叶子节点的最短路径上的节点数量。

官方还温馨的说明了下: 叶子节点是指没有子节点的节点。

因此遇到下面的情况

3

/ \

9 20

/ \

15 7

原来的代码，肯定是返回9和20对应的深度2，而不是15和7对应的深度3.

为了满足题目需求，需要额外加上一个条件， (root->left == nullptr || root->right == nullptr)

也就是判断当前节点是不是叶子节点，如果不是返回的值就是不为0的节点+1，如果是叶子结点就是两者较小的结果加1

**代码：**

class Solution {

public:

int minDepth(TreeNode\* root) {

if (root == nullptr) return 0;

int left = minDepth(root->left);

int right = minDepth(root->right);

if (root->left == nullptr || root->right == nullptr){

return left == 0 ? right+1 : left +1;

} else{

return min(left, right) + 1;

}

}

};

或：

class Solution {

public:

int minDepth(TreeNode\* root) {

if(!root)return 0;

int left\_depth = minDepth(root->left);

int right\_depth = minDepth(root->right);

if(root->left == nullptr || root->right == nullptr)

{

if(left\_depth == 0)

return right\_depth+1;

return left\_depth+1;

}

else

{

return min(left\_depth,right\_depth)+1;

}

}

};

**复杂度：**

## 方法二：迭代

**思路：**

**代码：**

**复杂度：**