# 题目

给定一个二叉树，返回其节点值的锯齿形层次遍历。（即先从左往右，再从右往左进行下一层遍历，以此类推，层与层之间交替进行）。

**例如：**

给定二叉树 [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

返回锯齿形层次遍历如下：

[

[3],

[20,9],

[15,7]

]

# 分析

## 方法一：递归法

**思路：广度优先遍历/栈**

这道题其实就是102.二叉树的层次遍历的翻版，多了一个条件if(level %2==0)，层数level从0开始表示二叉树的第一行，如果为偶数行则顺序加入，如果为奇数行则将数字每次插入到最前面。

**代码：**

class Solution {

public:

vector<vector<int>> res;

vector<vector<int>> zigzagLevelOrder(TreeNode\* root)

{

addVector(root,0); //调用递归函数

return res;

}

void addVector(TreeNode\* root,int level)

{

if(root == NULL) return;

if(res.size() == level)

res.resize(level+1);

//level表示层数，也对应二维数组的第一层索引，

if(level %2 == 0)

res[level].push\_back(root->val);

//如果为偶数行则顺序加入，如果为奇数行则将数字每次插入到最前面

else

res[level].insert(res[level].begin(),root->val);

addVector(root->left,level+1);

addVector(root->right,level+1);

}

};

## 方法二：迭代法（队列）

**思路：**深度优先遍历

**代码：**

class Solution {

public:

vector<vector<int>> zigzagLevelOrder(TreeNode\* root) {

vector<vector<int>> res;

if (root == NULL) return {};

queue<TreeNode\*> q;

q.push(root);

int level=0;

while (!q.empty())

{

vector<int>temp; //存放每一层的元素值

int count=q.size(); //队列大小表示当前层数的元素个数

while(count--) //count--逐个对该层元素进行处理

{

TreeNode \*t=q.front();

q.pop();

if(level%2 ==0)

temp.push\_back(t->val);

else

temp.insert(temp.begin(),t->val);

if(t->left) q.push(t->left);

if(t->right) q.push(t->right);

}

level++;

res.push\_back(temp); //将当层元素的vector加入res中

}

return res;

}

};

## 方法三：双栈

class Solution {

public:

vector<vector<int>> zigzagLevelOrder(TreeNode\* root) {

vector<vector<int>> res;

if(root==NULL) return res;

stack<TreeNode\*> sk1, sk2; //交替使用这两个栈遍历所有层

sk1.push(root);

while(!sk1.empty() || !sk2.empty()){

if(!sk1.empty()){

vector<int> tmp;

while(!sk1.empty()){

root = sk1.top();

sk1.pop();

tmp.emplace\_back(root->val);

if(root->left!=NULL) sk2.push(root->left);

if(root->right!=NULL) sk2.push(root->right);

}

res.emplace\_back(tmp);

}

if(!sk2.empty()){

vector<int> tmp;

while(!sk2.empty()){

root = sk2.top();

sk2.pop();

tmp.emplace\_back(root->val);

if(root->right!=NULL) sk1.push(root->right);

if(root->left!=NULL) sk1.push(root->left);

}

res.emplace\_back(tmp);

}

}

return res;

}

};