算法-二叉树Part2

102-层序遍历

借助队列

队列先进先出，符合一层一层遍历的逻辑，而是 用栈先进后出适合模拟深度优先遍历也就是递归的逻辑

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root)
    {
        //层序遍历 借助队列 先进先出
        queue<TreeNode*> que;
        if(root!=nullptr)
        {
            que.push(root);
        }
        vector<vector<int>> res;
        while(!que.empty())
        {
            //队列不为空时执行循环体
```

```
            int size = que.size();
            vector<int> vec;
            // 使用固定大小的size
            for(int i=0;i<size;i++)
            {
                TreeNode* node=que.front();     //首元素
                que.pop();
                vec.push_back(node->val);
                if(node->left) que.push(node->left);
                if(node->right) que.push(node->right);
            }
            res.push_back(vec);
        }
        return res;
    }
};
```

199-二叉树的右视图

示例：

输入： [1,2,3,null,5,null,4]
输出： [1, 3, 4]
解释：


```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root)
    {
        queue<TreeNode*> que;
        if(root!=nullptr) que.push(root);
        vector<int> res;
        while(!que.empty())
        {
            int size=que.size();
            for(int i=0;i<size;i++)
            {
                TreeNode* node =que.front();
                que.pop();
                if(i==(size-1)) res.push_back(node->val); //将每一层的最后元素放入result数组中
                if(node->left) que.push(node->left);
                if(node->right) que.push(node->right);
```

```
            }
        }
        return res;
    }
};
```

637-二叉树的层平均值

```
class Solution {
public:
    vector<double> averageOfLevels(TreeNode* root)
    {
        queue<TreeNode*> que;
        vector<double> res;
        if(root!=nullptr) que.push(root);
        while(!que.empty())
        {
            int size = que.size();
            double sum = 0; //计算每一层的结点和
            for(int i=0;i<size;i++)
            {
                TreeNode* cur=que.front();
                que.pop();
                sum+=cur->val;
                if(cur->left) que.push(cur->left);
                if(cur->right) que.push(cur->right);
            }
            res.push_back(sum/size);    //存储均值
        }
        return res;
    }
};
```

429-N叉树的层序遍历

https://leetcode.cn/problems/n-ary-tree-level-order-traversal/

429. N 叉树的层序遍历

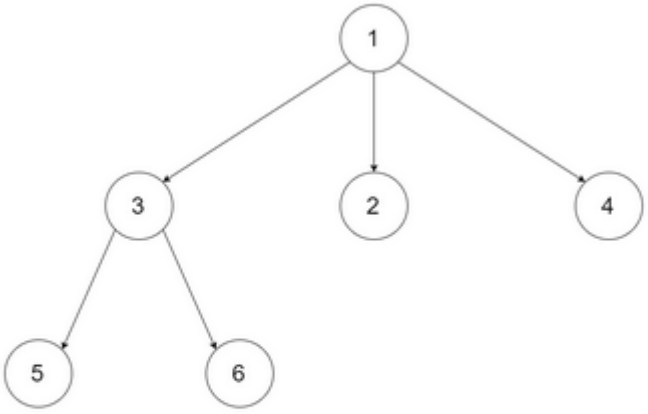
中等 334

相关企业

给定一个 N 叉树，返回其节点值的*层序遍历*。（即从左到右，逐层遍历）。

树的序列化输入是用层序遍历，每组子节点都由 null 值分隔（参见示例）。

示例 1：



输入： root = [1,null,3,2,4,null,5,6]
输出： [[1],[3,2,4],[5,6]]

```cpp
class Solution {
public:
    vector<vector<int>> levelOrder(Node* root)
    {
        queue<Node*> que;
        vector<vector<int>> res;
        if(root!=nullptr) que.push(root);
        while(!que.empty())
        {
            int size=que.size();
            vector<int> vec;
            for(int i=0;i<size;i++)
            {
                Node* cur=que.front();
```

```
                que.pop();
                vec.push_back(cur->val);
                for(int i=0;i<cur->children.size();i++)
                {
                    if(cur->children[i]) que.push(cur->children[i]);
                }
            }
            res.push_back(vec);
        }
        return res;
    }
};
```

515-在每个树行中找最大值

https://leetcode-cn.com/problems/find-largest-value-in-each-tree-row/

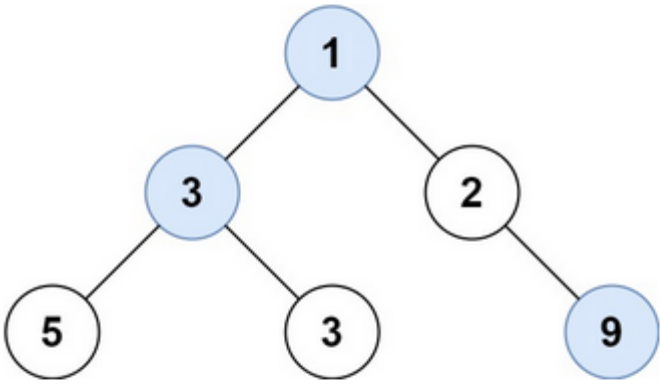
515. 在每个树行中找最大值

中等 291

相关企业

给定一棵二叉树的根节点 `root` ，请找出该二叉树中每一层的最大值。

示例1：




```
输入：root = [1,3,2,5,3,null,9]
输出：[1,3,9]
```

示例2：

```
输入：root = [1,2,3]
输出：[1,3]
```

```cpp
class Solution {
public:
    vector<int> largestValues(TreeNode* root)
    {
        queue<TreeNode*> que;
        if(root != nullptr) que.push(root);
        vector<int> res;
        while(!que.empty())
        {
            int size = que.size();
            int maxValue = INT_MIN;
            for(int i=0;i<size;i++)
```

```
            {
                TreeNode* node = que.front();
                que.pop();
                maxValue = node->val > maxValue ? node->val : maxValue;
                if(node->left) que.push(node->left);
                if(node->right) que.push(node->right);
            }
            res.push_back(maxValue);
        }
        return res;
    }
};
```

116-填充每个节点的下一个右侧节点指针

https://leetcode.cn/problems/populating-next-right-pointers-in-each-node/

116. 填充每个节点的下一个右侧节点指针

中等 918

相关企业

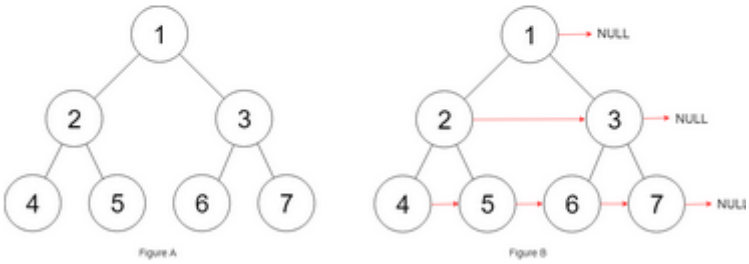
给定一个 **完美二叉树** ，其所有叶子节点都在同一层，每个父节点都有两个子节点。二叉树定义如下：

```
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,#]
```

```
class Solution {
public:
    Node* connect(Node* root)
    {
        queue<Node*> que;
        if(root != nullptr) que.push(root);
        while(!que.empty())
```

```
        {
            int size=que.size();
            vector<int> vec;
            Node* nodePre;  //当前层的头结点
            Node* node;
            for(int i=0;i<size;i++)
            {
                if(i==0)
                {
                    nodePre = que.front();  //当前层的头结点
                    que.pop();
                    node=nodePre;
                }
                else
                {
                    node = que.front();
                    que.pop();
                    nodePre->next = node;   //本层前一个结点的next指针指向本节点
                    nodePre = nodePre->next;
                }
                if(node->left) que.push(node->left);
                if(node->right) que.push(node->right);
            }
            nodePre->next = NULL;   //每层的最后一个结点指向NULL
        }
        return root;
    }
};
```

117-填充每个节点的下一个右侧指针II

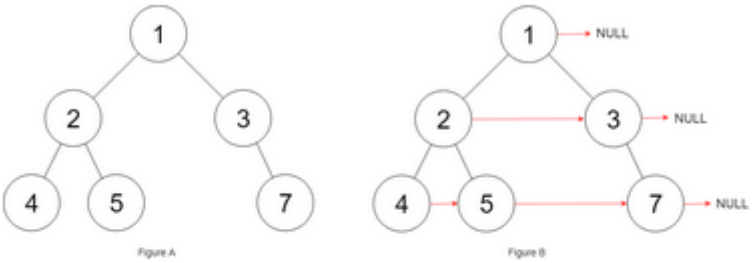
填充它的每个 next 指针，让这个指针指向其下一个右侧节点。如果找不到下一个右侧节点，则将 next 指针设置为 `NULL`。

初始状态下，所有 next 指针都被设置为 `NULL`。

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。

示例：



输入: root = [1,2,3,4,5,null,7]
输出: [1,#,2,3,#,4,5,7,#]
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。序列化输出按层序遍历顺序（由 next 指针连接），'#' 表示每层的末尾。

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* left;
    Node* right;
    Node* next;

    Node() : val(0), left(NULL), right(NULL), next(NULL) {}
```

```
11
12      Node(int _val) : val(_val), left(NULL),
    right(NULL), next(NULL) {}
13
14      Node(int _val, Node* _left, Node* _right, Node*
    _next)
15          : val(_val), left(_left), right(_right),
    next(_next) {}
16  };
17  */
18
19  class Solution {
20  public:
21      Node* connect(Node* root)
22      {
23          queue<Node*> que;
24          if(root != NULL)
25          {
26              que.push(root);
27          }
28          while(!que.empty())
29          {
30              int size=que.size();
31              vector<int> vec;
32              Node* nodePre;  //当前层的头节点
33              Node* node;
34              for(int i=0;i<size;i++)
35              {
36                  if(i==0)
37                  {
38                      nodePre = que.front();  //取出头节点
39                      que.pop();
40                      node = nodePre;
41                  }
42                  else
43                  {
44                      node=que.front();
45                      que.pop();
46                      nodePre->next=node;
47                      nodePre=nodePre->next;
```

```
48                  }
49                  if(node->left) que.push(node->left);
50                  if(node->right) que.push(node->right);
51              }
52              nodePre->next = NULL;
53          }
54          return root;
55      }
56  };
```

117. 填充每个节点的下一个右侧节点指针 II	几秒前
116. 填充每个节点的下一个右侧节点指针	6 分钟前
515. 在每个树行中找最大值	14 分钟前
429. N 叉树的层序遍历	18 分钟前
637. 二叉树的层平均值	25 分钟前
199. 二叉树的右视图	29 分钟前
107. 二叉树的层序遍历 II	32 分钟前
102. 二叉树的层序遍历	39 分钟前