题目地址（513. 找树左下角的值）

https://leetcode-cn.com/problems/find-bottom-left-tree-value/

入选理由

1. 和昨天一样，这是一道典型的树的搜索题，大家用两种搜索方式感受一下

标签

- 树
- BFS
- DFS

难度

- 中等

题目描述

给定一个二叉树，在树的最后一行找到最左边的值。

示例 1:

输入:

```
    2
   / \
  1   3
```

输出:
1

示例 2:

输入:

```
       1
      / \
     2   3
```

```
     /   / \
    4   5   6
       /
      7

输出:
7
```

BFS

思路

其实问题本身就告诉你怎么做了

> 在树的最后一行找到最左边的值。

问题再分解一下

- 找到树的最后一行
- 找到那一行的第一个节点

不用层序遍历简直对不起这个问题，这里贴一下层序遍历的流程

```
令curLevel为第一层节点也就是root节点
定义nextLevel为下层节点
遍历node in curLevel,
  nextLevel.push(node.left)
  nextLevel.push(node.right)
令curLevel = nextLevel, 重复以上流程直到curLevel为空
```

代码

代码支持：JS，Python，Java，CPP

JS Code：

```
var findBottomLeftValue = function (root) {
  let curLevel = [root];
  let res = root.val;
  while (curLevel.length) {
    let nextLevel = [];
    for (let i = 0; i < curLevel.length; i++) {
```

```
      curLevel[i].left && nextLevel.push(curLevel[i].left);
      curLevel[i].right && nextLevel.push(curLevel[i].right);
    }
    res = curLevel[0].val;
    curLevel = nextLevel;
  }
  return res;
};
```

Python Code:

```
class Solution(object):
    def findBottomLeftValue(self, root):
        queue = collections.deque()
        queue.append(root)
        while queue:
            length = len(queue)
            res = queue[0].val
            for _ in range(length):
                cur = queue.popleft()
                if cur.left:
                    queue.append(cur.left)
                if cur.right:
                    queue.append(cur.right)
        return res
```

Java：

```
class Solution {
    Map<Integer,Integer> map = new HashMap<>();
    int maxLevel = 0;
    public int findBottomLeftValue(TreeNode root) {
        if (root == null) return 0;
        LinkedList<TreeNode> deque = new LinkedList<>();
        deque.add(root);
        int res = 0;
        while(!deque.isEmpty()) {
            int size = deque.size();
            for (int i = 0; i < size; i++) {
                TreeNode node = deque.pollFirst();
                if (i == 0) {
                    res = node.val;
                }
                if (node.left != null)deque.addLast(node.left);
                if (node.right != null)deque.addLast(node.right);
            }
        }
        return res;
```

```
    }
}
```

CPP：

```cpp
class Solution {
public:
    int findBottomLeftValue_bfs(TreeNode* root) {
        queue<TreeNode*> q;
        TreeNode* ans = NULL;
        q.push(root);
        while (!q.empty()) {
            ans = q.front();
            int size = q.size();
            while (size--) {
                TreeNode* cur = q.front();
                q.pop();
                if (cur->left )
                    q.push(cur->left);
                if (cur->right)
                    q.push(cur->right);
            }
        }
        return ans->val;
    }
}
```

复杂度分析

- 时间复杂度：$O(N)$，其中 N 为树的节点数。
- 空间复杂度：$O(Q)$，其中 Q 为队列长度，最坏的情况是满二叉树，此时和 N 同阶，其中 N 为树的节点总数

DFS

思路

树的最后一行找到最左边的值，转化一下就是找第一个出现的深度最大的节点，这里用先序遍历去做，其实中序遍历也可以，只需要保证左节点在右节点前被处理即可。 具体算法为，先序遍历 root，维护一个最大深度的变量，记录每个节点的深度，如果当前节点深度比最大深度要大，则更新最大深度和结果项。

代码

代码支持：JS，Python，Java，CPP

JS Code：

```
function findBottomLeftValue(root) {
  let maxDepth = 0;
  let res = root.val;

  dfs(root.left, 0);
  dfs(root.right, 0);

  return res;

  function dfs(cur, depth) {
    if (!cur) {
      return;
    }
    const curDepth = depth + 1;
    if (curDepth > maxDepth) {
      maxDepth = curDepth;
      res = cur.val;
    }
    dfs(cur.left, curDepth);
    dfs(cur.right, curDepth);
  }
}
```

Python Code:

```
class Solution(object):

    def __init__(self):
        self.res = 0
        self.max_level = 0

    def findBottomLeftValue(self, root):
        self.res = root.val
        def dfs(root, level):
            if not root:
                return
            if level > self.max_level:
                self.res = root.val
                self.max_level = level
            dfs(root.left, level + 1)
            dfs(root.right, level + 1)
        dfs(root, 0)

        return self.res
```

Java Code:

```
class Solution {
    int max = 0;
    Map<Integer,Integer> map = new HashMap<>();
    public int findBottomLeftValue(TreeNode root) {
        if (root == null) return 0;
        dfs(root,0);
        return map.get(max);
    }

    void dfs (TreeNode node,int level){
        if (node == null){
            return;
        }
        int curLevel = level+1;
        dfs(node.left,curLevel);
        if (curLevel > max && !map.containsKey(curLevel)){
            map.put(curLevel,node.val);
            max = curLevel;
        }
        dfs(node.right,curLevel);
    }

}
```

CPP:

```
class Solution {
public:
    int res;
    int max_depth = 0;
    void findBottomLeftValue_core(TreeNode* root, int depth) {
        if (root->left || root->right) {
            if (root->left)
                findBottomLeftValue_core(root->left, depth + 1);
            if (root->right)
                findBottomLeftValue_core(root->right, depth + 1);
        } else {
            if (depth > max_depth) {
                res = root->val;
                max_depth = depth;
            }
        }
    }
    int findBottomLeftValue(TreeNode* root) {
        findBottomLeftValue_core(root, 1);
        return res;
    }
};
```

复杂度分析

- 时间复杂度：$O(N)$，其中 N 为树的节点总数。
- 空间复杂度：$O(h)$，其中 h 为树的高度。

上一页 下一页