树的深度

Saturday, July 28, 2018 9:39 AM

Given binary tree `[3,9,20,null,null,15,7]`,

```
  3
 / \
9  20
  /  \
 15   7
```

用 `return left > right ? left : right;` 来选择较大的一边

最大深度（二叉树）

```
if (root == null){
    return 0;
}

int left = maxDepth(root.left) + 1;
int right = maxDepth(root.right) + 1;

return left > right ? left : right;
```

先找到底部，然后一步步返回：

先看左边：

left = maxDepth(9) + 1
right = maxDepth(9) = 0
所以 left = 1，right = 0

右边：

从最底下开始：

Left = maxDepth(15) + 1 = 1;
Right = maxDepth(7) + 1 = 1 ;
返回 1

maxDepth(20) = 1（刚才返回的）+ 1 = 2

root.left = 1 + 1 = 2 , root.right = 2 + 1 = 3

所以结果 = 3

最大深度（树）

不只是二叉树，这题目其实包含了广搜的做法

题目采用的数据结构是 List<Node>，一个 List 表示一个节点的所有孩子

如果根节点为 null，返回 0

```
int maxDepth = 0;
if (root == null){
    return 0;
}
for (Node child :
        root.children) {
    maxDepth = Math.max(maxDepth, DFS(child));
}
return maxDepth + 1;
```

对一个节点的每一个孩子进行 DFS，和二叉树最大深度一样，一层一层 +1 来计算深度

Given binary tree `[3,9,20,null,null,15,7]`,

```
  3
 / \
9  20
  /  \
 15   7
```

用 `return left < right ? left : right;` 来选择较小的一边

最小深度

```
if (root == null){
    return 0;
}
if (root.left == null && root.right == null){
    return 1;
}
if (root.left == null){
    return minDepth(root.right) + 1;
}
if (root.right == null){
    return minDepth(root.left) + 1;
}

int left = minDepth(root.left) + 1;
int right = minDepth(root.right) + 1;

return left < right ? left : right;
```

如果节点为空，结果为0

如果没有孩子，就返回1

接下来才用递归：

1. Root(3)
   left = minDepth(9) + 1 = 2;
   right = minDepth(20) + 1;
2. minDepth(20)
   left = minDepth(15) + 1 = 2;
   right = minDepth(7) + 1 = 2;
   minDepth = 2

root.left = 2, root.right = 3

所以结果 = 2;