40 专题2：二叉树遍历

更新时间：2019-10-14 09:25:20

> 虚心使人进步，骄傲使人落后。
>
> ——毛泽东

二叉树

什么是二叉树？

什么是树？在计算机科学中，树就是许许多多个 Node 组成的一个结构，要求这个结构联通且无环。

那么 Node 是什么？Node 就是一个节点，这个节点有两大属性，一个是 value，另一个就是它的子 Node。对于我们的二叉树来说，它的子 Node 最多有两个，因此二叉树有三个属性：value、left和right。之前我们已经学过递归的概念，那么这个 Node 的 left 和 right 也都是一个 Node。我们可以一直递归下去，一直到某一个 Node 的 left 和 right 都为空，下面是典型的二叉树 Node 的定义：

```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
```

```
public class TreeNode {
    public int val;
    public TreeNode left;
    public TreeNode right;

    public TreeNode(int value) {
        this.val = value;
    }
}
```

对于一棵树来说，它永远会有一个顶端，在计算机语言中，这个顶端称为 Root Node。

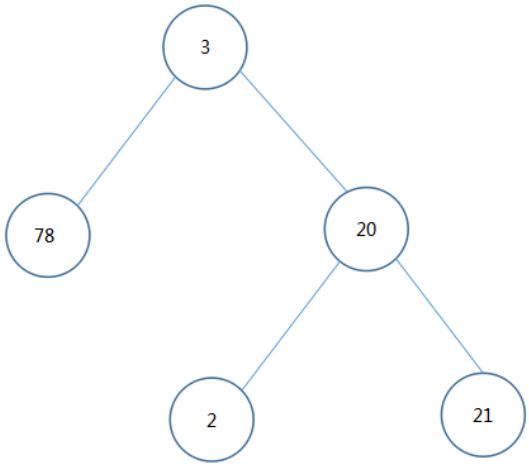
一般来说，我们拿到手中的永远是一棵树的 Root Node，那么如果通过它去得知我们整棵树的情况呢？

举个例子，我们都知道沙漠里有种植物，它的根非常深，跨度也非常广。目前我们只看到这个植物在地面上的那一小部分，我们很想知道地底下这个植物的所有情况，这就引申出了一个概念——二叉树的遍历。

二叉树的遍历

遍历，即了解全貌，得知这棵树所有的 Node 分别是什么，结构是怎么样的。不过在计算机中，我们通常只需要取得这棵树中储存的所有 value，并按照一定的结构顺序来展示出来。那么我们有多少种不同的结构顺序呢？

我们的树结构如下：



前序遍历 (Pre-Order Traversal)

这个顺序其实就是说，我们拿到一个Root Node之后，首先把 Root Node 中的 value 拿到手，然后我们命令计算机按照相同的顺序（即先序遍历）取得 root.left 这棵子树中所有的 value，然后再拿到 root.right 这棵子树中所有的 value。

下面我给出递归的解法：

```python
class Solution(object):
    def preorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        res = []
        if not root:
            return res
        res.append(root.val)
        if root.left: # 左Node存在就按照按照相同的顺序（即先序遍历）取得root.left这棵子树中所有的value
            res.extend(self.preorderTraversal(root.left))
        if root.right: # 右Node存在就按照按照相同的顺序（即先序遍历）取得root.right这棵子树中所有的value
            res.extend(self.preorderTraversal(root.right))
        return res
```

```java
class Solution {
    List<Integer> list = new ArrayList<>();
    public List<Integer> preorderTraversal(TreeNode root) {
        // 递归终止条件，root 为 null 则遍历结束
        if (root == null) {
            return list;
        }
        // 前序遍历添加顺序：root--root.left--root.right
        list.add(root.val);
        preorderTraversal(root.left);
        preorderTraversal(root.right);
        return list;
    }
}
```

此题就是 leetcode 144。

中序遍历 (In-Order Traversal)

这个顺序其实就是说，我们拿到一个Root Node之后，首先命令计算机按照相同的顺序（即中序遍历）取得root.left这棵子树中所有的 value，然后把 Root Node 中的 value 拿到手，最后拿到 root.right 这棵子树中所有的 value。

下面我给出递归的解法：

```python
class Solution(object):
    def inorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        res = []
        if not root:
            return res
        if root.left: # 左Node存在就按照按照相同的顺序（即中序遍历）取得root.left这棵子树中所有的value
            res.extend(self.inorderTraversal(root.left))
        res.append(root.val)
        if root.right: # 右Node存在就按照按照相同的顺序（即中序遍历）取得root.right这棵子树中所有的value
            res.extend(self.inorderTraversal(root.right))
        return res
```

```java
class Solution {
    List<Integer> list = new ArrayList<>();
    public List<Integer> inorderTraversal(TreeNode root) {
        // 递归终止条件，root 为 null 则遍历结束
        if (root == null) {
            return list;
        }
        // 中序遍历添加顺序：root.left--root--root.right
        inorderTraversal(root.left);
        list.add(root.val);
        inorderTraversal(root.right);
        return list;
    }
}
```

此题就是 leetcode 94。

后序遍历 **(Post-Order Traversal)**

这个顺序其实就是说，我们拿到一个Root Node之后，首先命令计算机按照相同的顺序（即后序遍历）取得root.left这棵子树中所有的 value，然后再拿到 root.right 这棵子树中所有的 value，最后把 Root Node 中的value 拿到手。

下面我给出递归的解法：

```python
class Solution:
    def postorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        if not root:
            return res
        if root.left: # 左Node存在就按照按照相同的顺序（即后序遍历）取得root.left这棵子树中所有的value
            res.extend(self.postorderTraversal(root.left))
        if root.right: # 右Node存在就按照按照相同的顺序（即右序遍历）取得root.right这棵子树中所有的value
            res.extend(self.postorderTraversal(root.right))
        res.append(root.val)
        return res
```

```java
class Solution {
    List<Integer> list = new ArrayList<>();
    public List<Integer> postorderTraversal(TreeNode root) {
        // 递归终止条件，root 为 null 则遍历结束
        if (root == null) {
            return list;
        }
        // 后序遍历添加顺序：root.left--root.right--root
        postorderTraversal(root.left);
        postorderTraversal(root.right);
        list.add(root.val);
        return list;
    }
}
```

此题就是 leetcode 145。

层序遍历 **(Level-Order Traversal)**

这里额外补充一种遍历方式，即一层一层地取得我们的 value。

```python
class Solution:
    def levelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        def dfs(node, level, res):
            if not node:
                return
            if len(res) < level: # 如果该层还未初始化
                res.append([])
            res[level-1].append(node.val) # 当前node属于第level层
            dfs(node.left, level+1, res) # node.left属于第level+1层
            dfs(node.right, level+1, res) # node.right属于第level+1层

        res = []
        dfs(root, 1, res)
        return res
```

```java
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        dfs(res, root, 0);
        return res;
    }

    private void dfs(List<List<Integer>> res, TreeNode root, int level) {
        // 递归终止条件，root 为 null 则遍历结束
        if (root == null) {
            return;
        }
        // 根据 level 以及集合的大小判断是否需要加入新的层级，也就是新的的 List
        if (level >= res.size()) {
            res.add(new ArrayList<>());
        }
        res.get(level).add(root.val);
        // 向下一个 level 中的左树进行遍历
        dfs(res,root.left,level+1);
        // 向下一个 level 中的左树进行遍历
        dfs(res,root.right,level+1);
    }
}
```

此题就是 leetcode 102。

总结

一下子看了这么多东西，让我们来总结一下吧：

1. 树就是许许多多个Node组成的一个结构，要求这个结构联通且无环；
2. Node就是一个节点，这个节点有两大属性，一个是 value，另一个就是它的子 Node；
3. 对于二叉树来说，它的子 Node 最多有两个，因此二叉树有三个属性：value、left和right；
4. 二叉树的遍历方式：
    - 前序遍历
    - 中序遍历
    - 后序遍历
    - 层序遍历

```
}
```

39 专题1：LRU Cache 最近最少使用算法

41 专题3：二分算法