

Python入门与基础

第五讲 树结构与递归

Tree & Recursion

林平之老师



扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: [ninechapter](#)

知乎专栏: <http://zhuanlan.zhihu.com/jiuzhang>

微博: <http://www.weibo.com/ninechapter>

官网: www.jiuzhang.com

**九章课程不提供视频，也严禁录制视频的侵权行为
否则将追求法律责任和经济赔偿
请不要缺课**

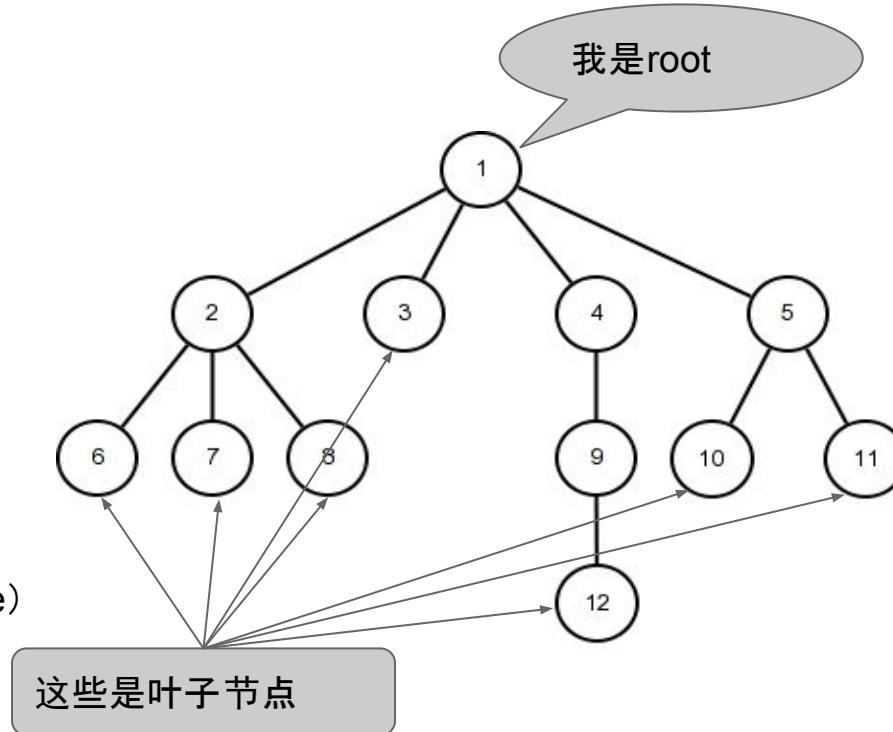
- 二叉树及其遍历
- 递归算法
- 递归算法的时间, 空间复杂度分析

现实中的树是这样的，那么
数据结构中的树呢？



树 Tree (好比一个公司)

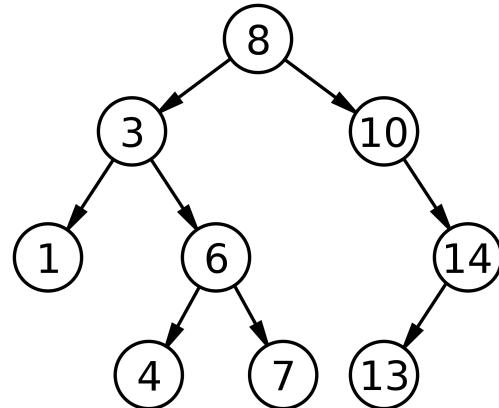
- 由节点(node)组成
- 每个节点有零个或多个子节点(child node)
 - 这是一个manager, 他管理很多人
- 没有父节点的是根节点(root node)
 - 公司的大Boss
- 每个非根节点只有一个父节点(parent node)
 - 除了大Boss, 每个人都有一个manager
- 没有任何子节点的节点叫叶子节点(leaf node)
 - 底层的员工
- 一棵树中, 只有一个root node
 - 大Boss之允许有一个



二叉树 (binary tree)

- 每个节点**最多**有两个子节点
- 两个子节点分别被称为左孩子 (left child) 和右孩子 (right child)
- 叶子节点: 没有孩子节点的节点

不特别说明的话, 我们提到的树都是指二叉树



子树(sub-tree)

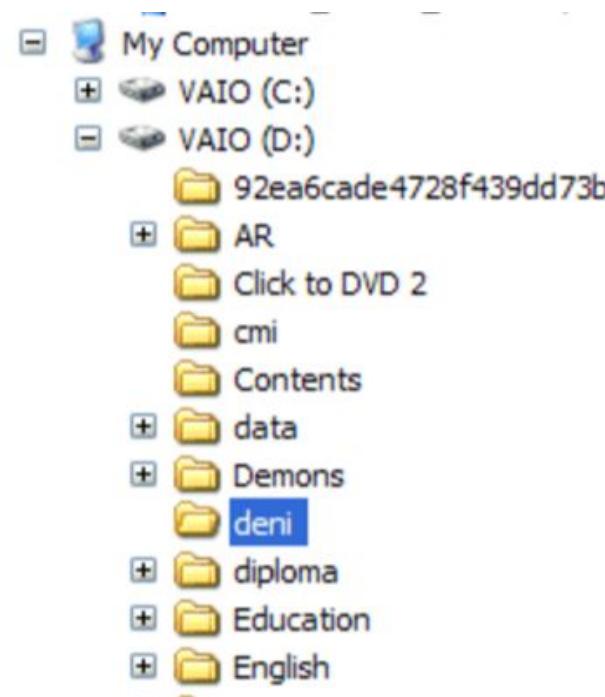
树中的每个节点代表以它为根的一棵树

左孩子所代表的树成为左子树(left sub-tree)

右孩子所代表的树成为右子树(right sub-tree)

- 如何遍历一棵树
- 如何使用树的遍历算法解题
- 如何分析递归程序的时间，并学会分析时间、空间复杂度

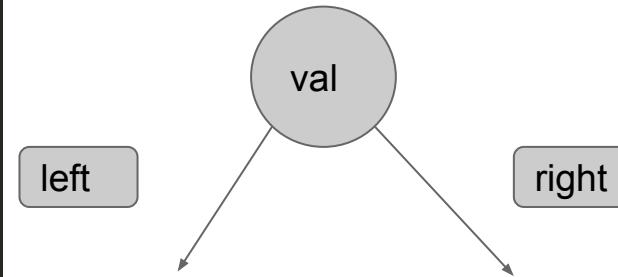
- 文件系统 B+树
- 数据库的索引—第七节课
- 字典树，平衡树等 - 高级数据结构



- 构造一棵二叉树 Construct a binary tree
- 打印出这课二叉 print a binary tree
 - 格式如下 node value: x, parent node, left node, right node

剖析LintCode TreeNode

```
1 class TreeNode:  
2  
3     def __init__(self, val):  
4         self.val = val  
5         self.left, self.right = None, None  
6  
7  
8  
9
```



TreeNode类

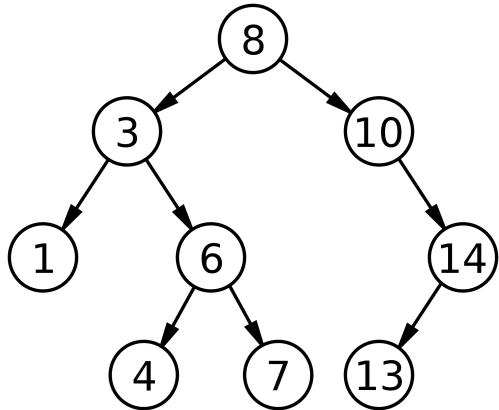
1. left和right分别对应**左右子节点**
2. val表示node的值

Coding & Print 构造和打印二叉树

树(二叉树)的遍历 (Binary Tree Traversal)

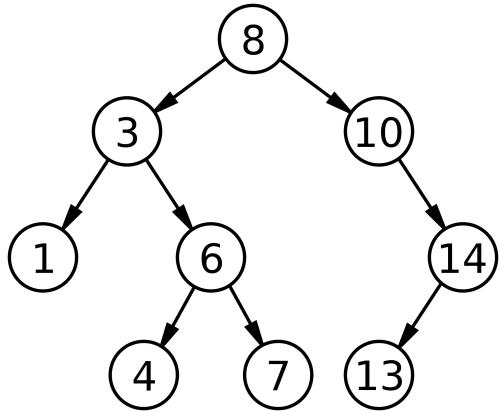
- 先序遍历 (Preorder traversal)
 - 口诀: 根左右
- 中序遍历 (Inorder traversal)
 - 口诀: 左根右
- 后序遍历 (Postorder traversal)
 - 口诀: 左右根

先序遍历 Preorder traversal

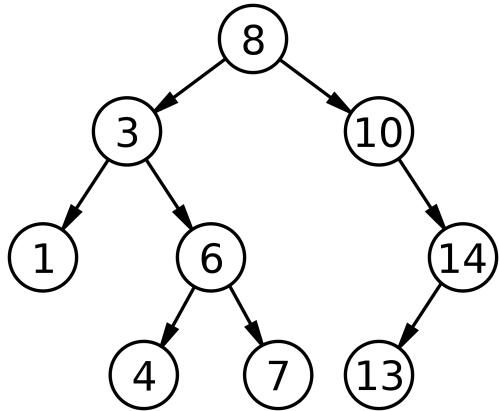


根左右:8, 3, 1, 6, 4, 7, 10, 14, 13 (先序遍历)

中序遍历 Inorder traversal



左根右: 1, 3, 4, 6, 7, 8, 10, 13, 14 (中序遍历)



左右根：请同学们写出这棵树的后续遍历

如何遍历一棵树？

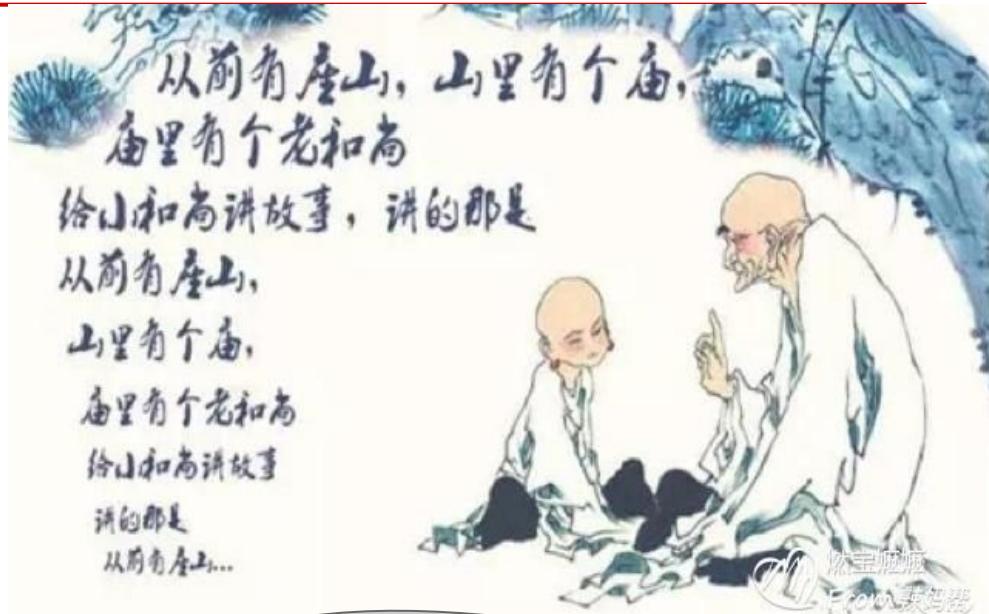
使用递归的**方式**！？

程序实现一般有两种方式：

- 递归的实现方式
- 非递归的实现方式

什么是递归 (Recursion)？

- 数据结构的递归
 - 树就是一种递归的数据结构
- 算法(程序)的递归
 - 函数自己调用自己



没错，这就是递归。。。

- 递归的定义
 - 首先这个问题或者数据结构得是递归定义的
- 递归的出口
 - 什么时候递归终止
- 递归的拆解
 - 递归不终止的时候, 如何分解问题

经典例题Fibonacci

<http://www.lintcode.com/en/problem/fibonacci/>

```
1  class Solution:
2      """
3          @param: n: an integer
4          @return: an ineger f(n)
5      """
6      def fibonacci(self, n):
7          # write your code here
8          if n == 1:
9              return 0
10         if n == 2:
11             return 1
12
13         return self.fibonacci(n - 1) + self.fibonacci(n - 2)
14
15
```

递归的定义：

- 因为斐波那契数列满足 $F(n) = F(n - 1) + F(n - 2)$

递归的出口：

- $n = 0$ 和 $n = 1$ 的时候，问题规模足够小的时候

递归的拆解：

- `return self.fibonacci(n - 1) + self.fibonacci(n - 2)`

Coding 打印出一个树的中序遍历

- 获取所有叶子节点的和 Get leaf sum
- 获取树的高度 Get tree height
- 获取所有root到叶子节点的路径 Get root-to-leaf paths

Binary Tree Leaf Sum

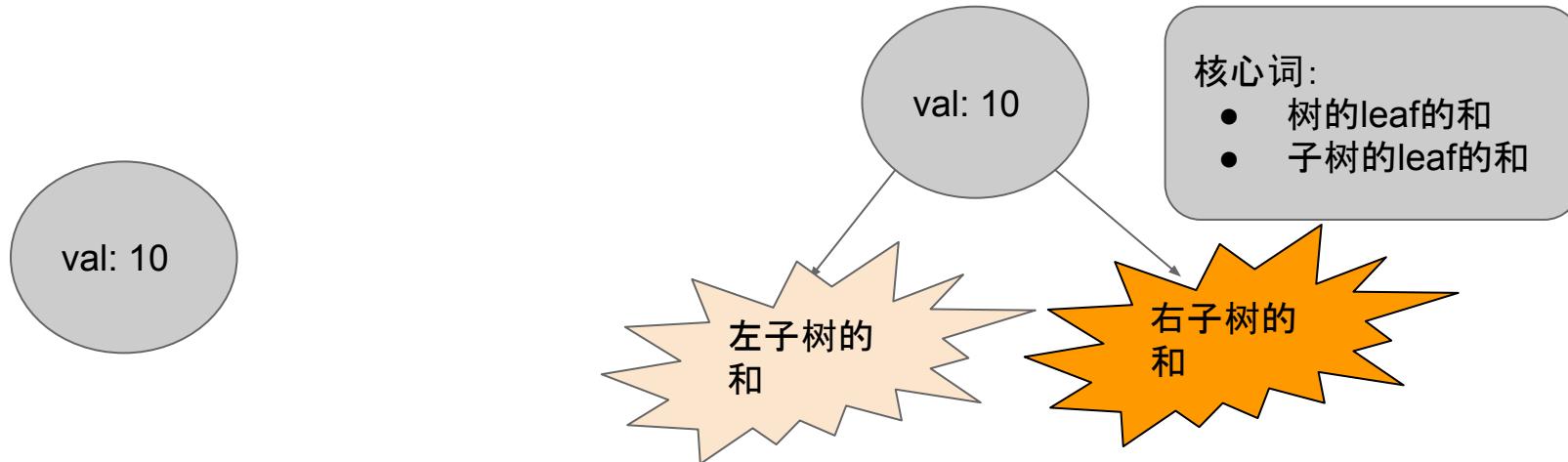
<http://www.lintcode.com/en/problem/binary-tree-leaf-sum/>

<http://www.jiuzhang.com/solution/binary-tree-leaf-sum/>

Recursion 获取叶子节点的和

访问一个Node:

- 如果这个Node是叶子节点, 则sum就是他本身
- 如果这个Node不是叶子节点, 则sum等于左子树的叶子节点和 + 右子树之和



Maximum Depth of Binary Tree

<http://www.lintcode.com/en/problem/maximum-depth-of-binary-tree/>

<http://www.jiuzhang.com/solution/maximum-depth-of-binary-tree/>

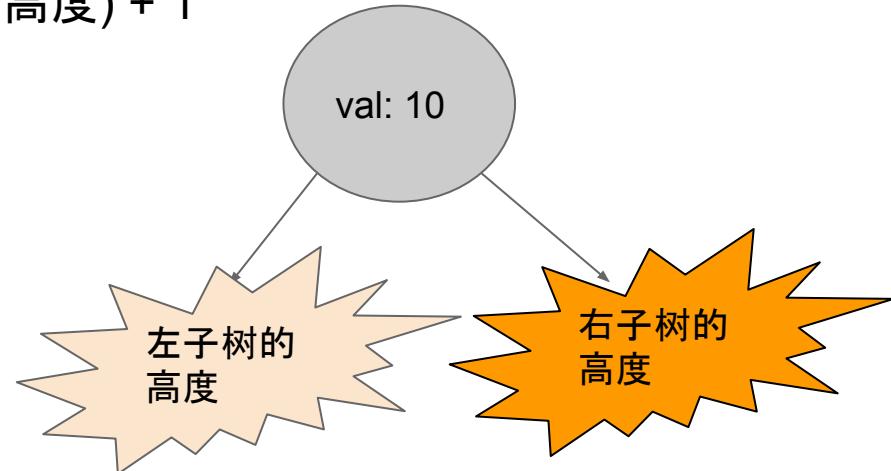
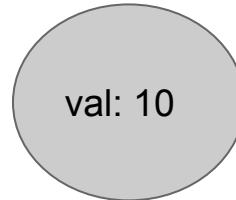
Recursion 获取树的高度

访问一个Node:

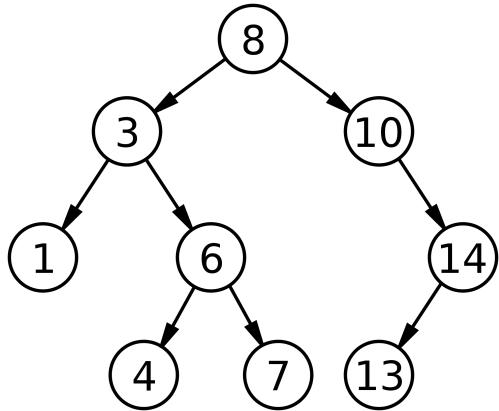
- 如果这个Node是叶子节点, 则高度是1
- 如果这个Node不是叶子节点, 则高度等于
 - $\max(\text{左子树的高度}, \text{右子树的高度}) + 1$

核心词:

- 树的高度
- 子树的高度



Recursion 获取树中root到leaf的所有路径



能否根据之前两个问题，我们来做同样的分析？

Identical Binary Tree

<http://www.lintcode.com/en/problem/identical-binary-tree/>

<http://www.jiuzhang.com/solutions/identical-binary-tree/>

题目大意: 判断两棵Tree是否同构, 同构的定义是可以通过交换左右子树是的他们相同

Recursion 判断子树是否同构

访问一个A树中的Node1, 和B树中的Node2:

- 如果这个Node1和Node2都是NULL, 则同构
- 如果这个Node1和Node2都不是NULL, 则同构的条件是
 - Node1和Node2节点val相同
 - Node1和Node2的left subtree同构且Node1和Node2的right subtree同构
- 他们不同构



Binary Tree Preorder Traversal

<http://www.lintcode.com/en/problem/binary-tree-preorder-traversal/>

Binary Tree Inorder Traversal

<http://www.lintcode.com/en/problem/binary-tree-inorder-traversal/>

Binary Tree Postorder Traversal

<http://www.lintcode.com/en/problem/binary-tree-postorder-traversal/>

更多Traversal的问题 <http://www.lintcode.com/en/tag/binary-tree-traversal/>



谢谢大家