

# 二叉树

@M了个J

<https://github.com/CoderMJLee>

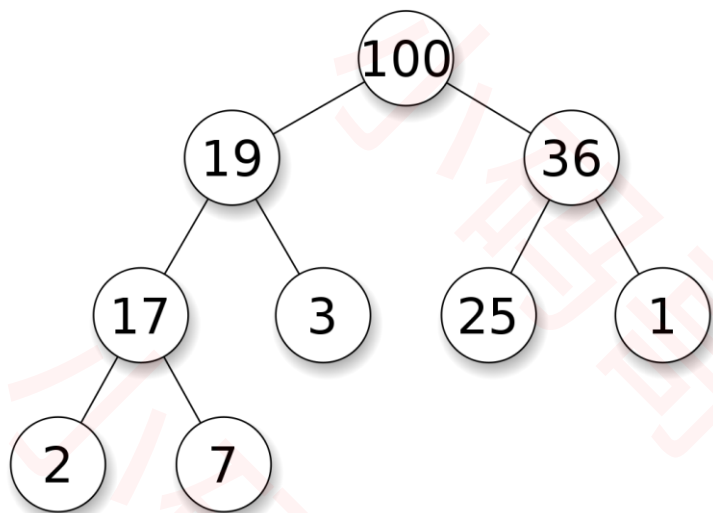
<http://cnblogs.com/mjios>

码拉松

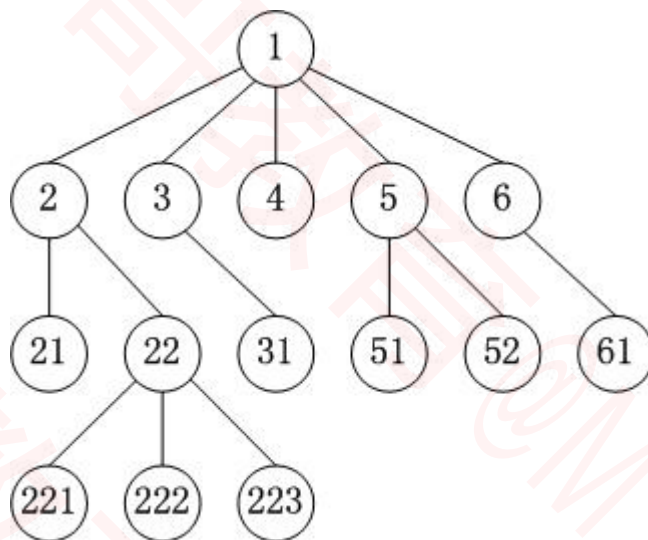


实力IT教育 www.520it.com

# 树形结构



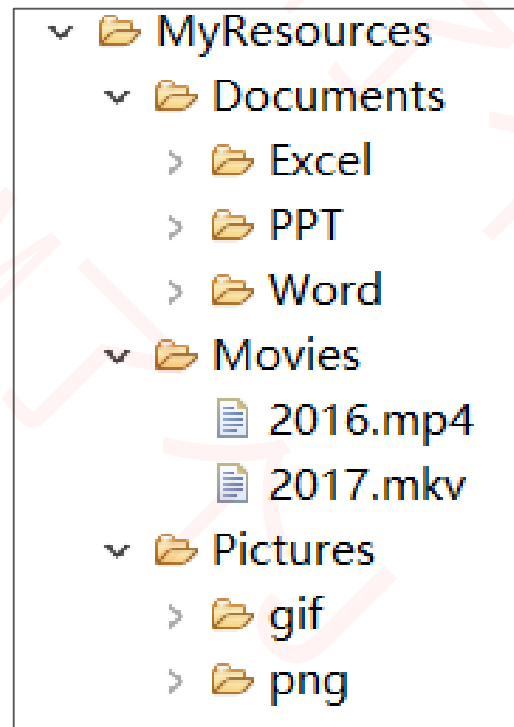
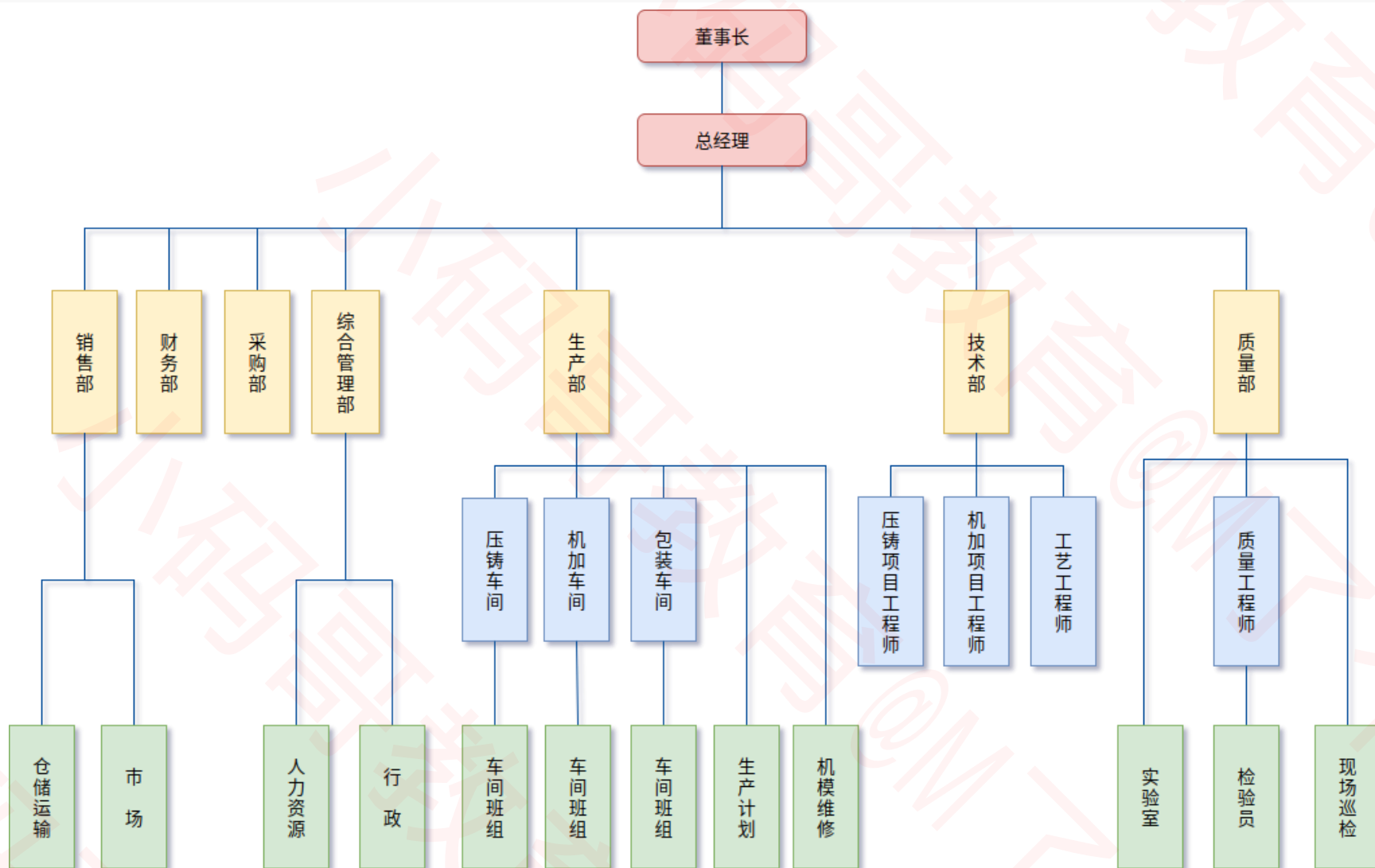
二叉树



多叉树



# 生活中的树形结构

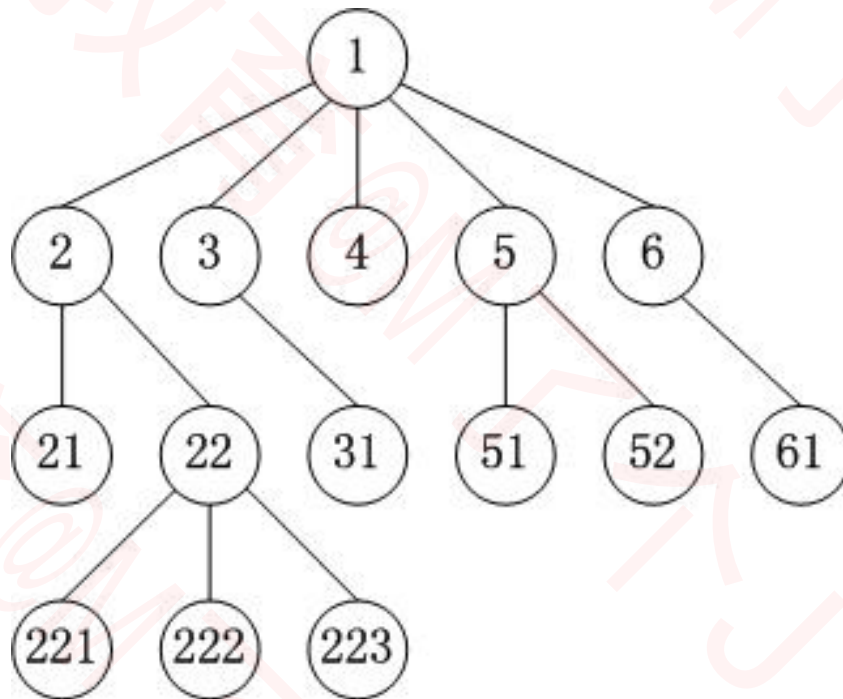


- 使用树形结构可以大大提高效率
- 树形结构是算法面试的重点

# 树 (Tree) 的基本概念

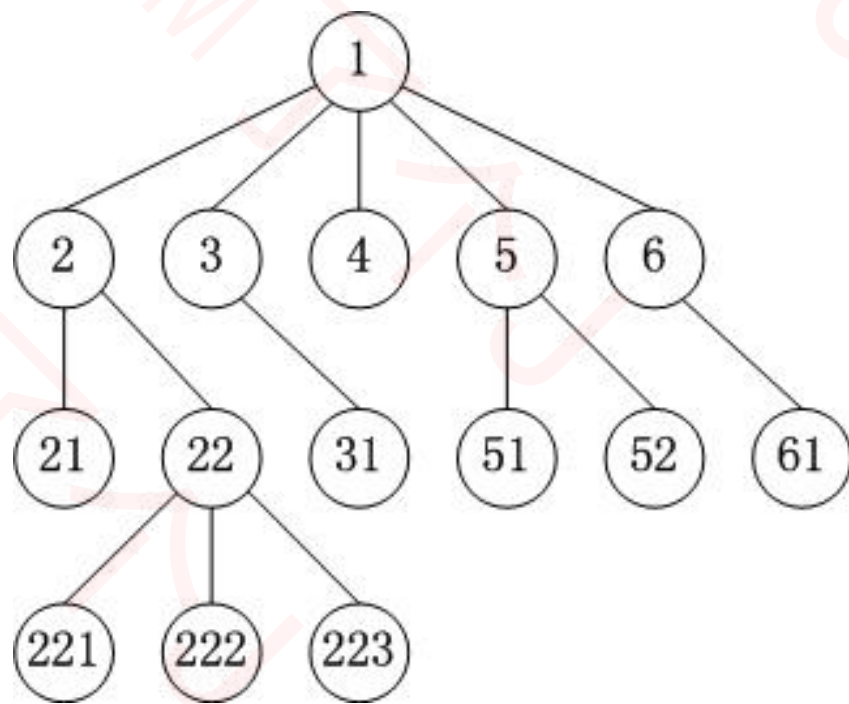
- 节点、根节点、父节点、子节点、兄弟节点
- 一棵树可以没有任何节点，称为**空树**
- 一棵树可以只有1个节点，也就是只有根节点
- 子树、左子树、右子树

- 节点的**度** (degree) : 子树的个数
- 树的**度**: 所有节点度中的最大值
- **叶子节点** (leaf) : 度为0的节点
- **非叶子节点**: 度不为0的节点



# 树 (Tree) 的基本概念

- **层数** (level) : 根节点在第1层, 根节点的子节点在第2层, 以此类推 (有些教程也从第0层开始计算)
- 节点的**深度** (depth) : 从根节点到当前节点的唯一路径上的节点总数
- 节点的**高度** (height) : 从当前节点到最远叶子节点的路径上的节点总数
- 树的**深度**: 所有节点深度中的最大值
- 树的**高度**: 所有节点高度中的最大值
- 树的**深度** 等于 树的**高度**



# 有序树、无序树、森林

## ■ 有序树

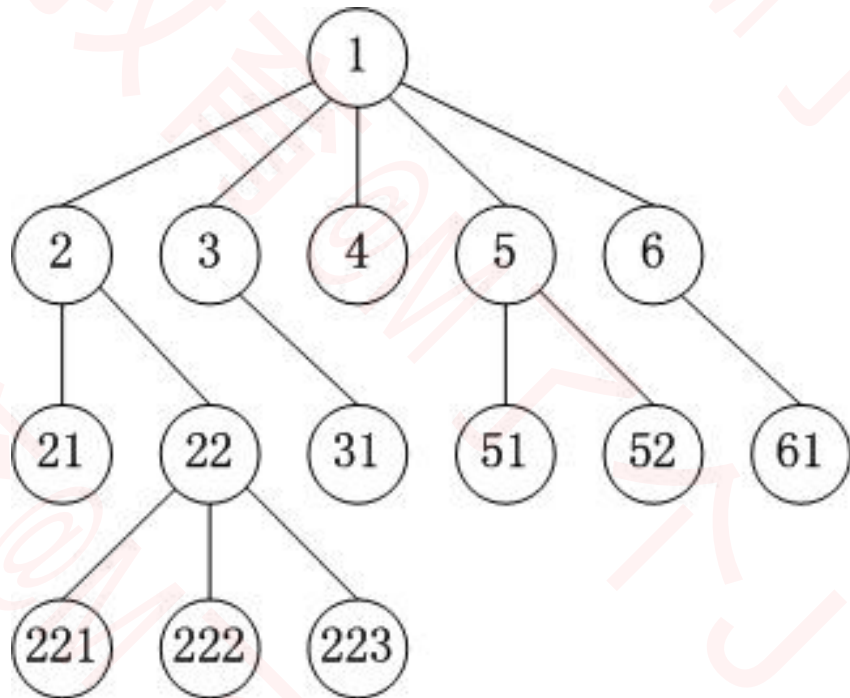
- 树中任意节点的子节点之间有顺序关系

## ■ 无序树

- 树中任意节点的子节点之间没有顺序关系
- 也称为“自由树”

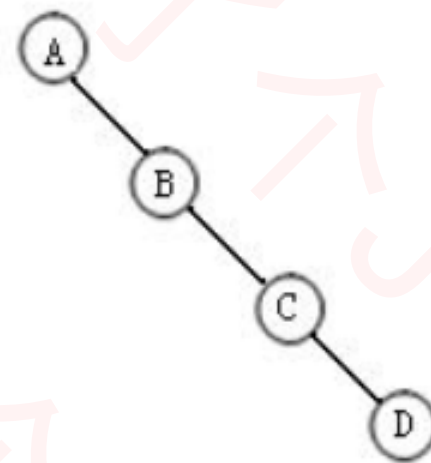
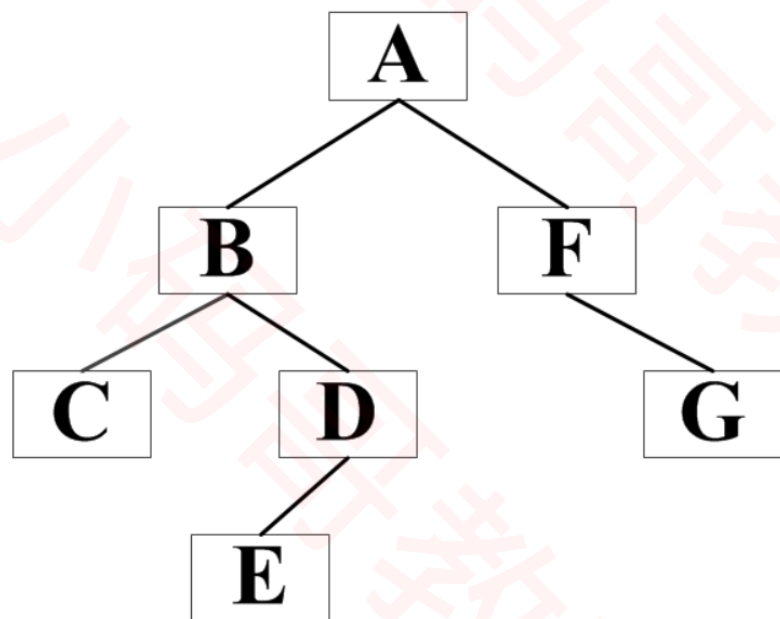
## ■ 森林

- 由 $m$  ( $m \geq 0$ ) 棵互不相交的树组成的集合



## ■ 二叉树的特点

- 每个节点的度最大为2（最多拥有2棵子树）
- 左子树和右子树是有顺序的
- 即使某节点只有一棵子树，也要区分左右子树



- 二叉树是有序树 or 无序树?
- 有序树



# 二叉树的性质

■ 非空二叉树的第 $i$ 层，最多有 $2^{i-1}$ 个节点 ( $i \geq 1$ )

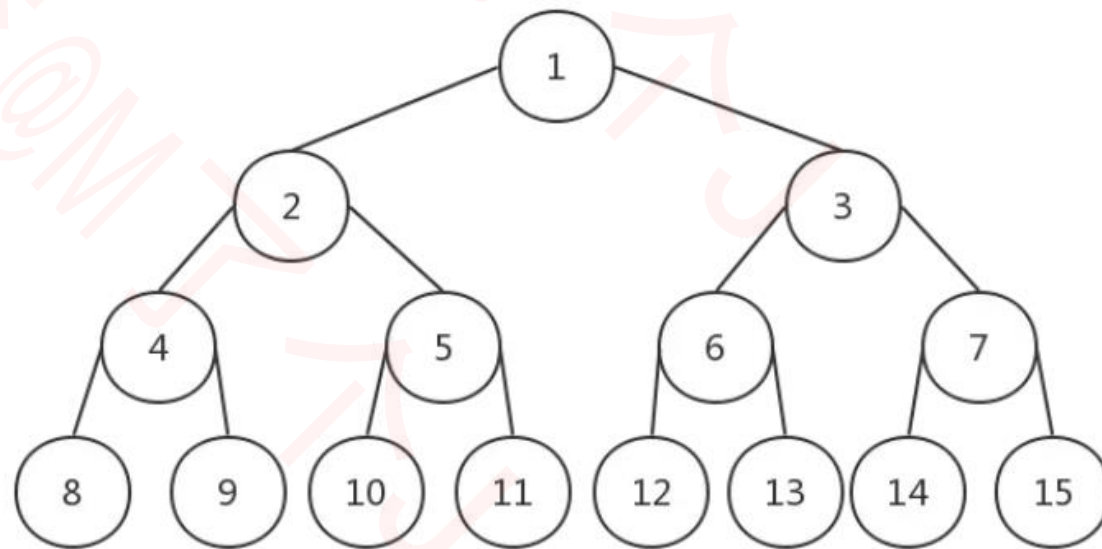
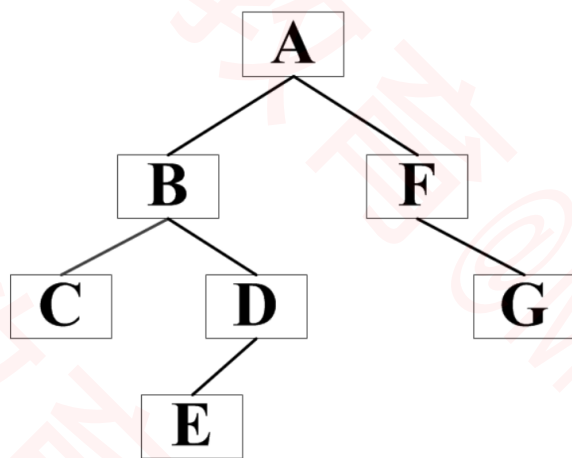
■ 在高度为 $h$ 的二叉树上最多有 $2^h - 1$ 个结点 ( $h \geq 1$ )

■ 对于任何一棵非空二叉树,如果叶节点个数为 $n_0$ , 度为2的节点个数为 $n_2$ , 则有:  $n_0 = n_2 + 1$

□ 假设度为1的节点个数为 $n_1$ , 那么二叉树的节点总数 $n = n_0 + n_1 + n_2$

□ 二叉树的边数 $T = n_1 + 2 * n_2 = n - 1 = n_0 + n_1 + n_2 - 1$

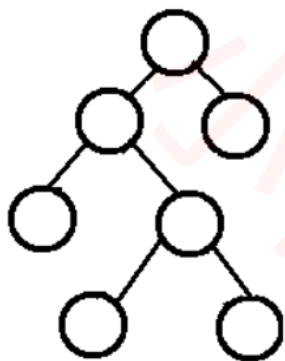
□ 因此 $n_0 = n_2 + 1$



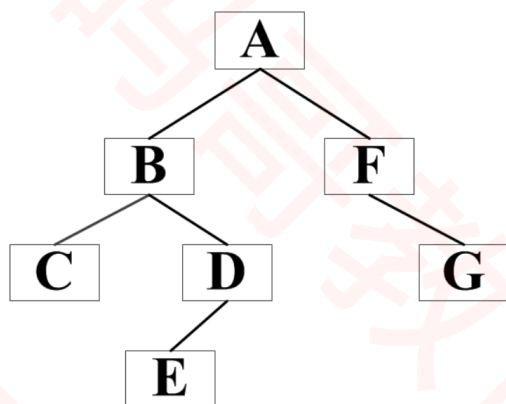


# 真二叉树 (Proper Binary Tree)

■ 真二叉树：所有非叶子节点的度都为2

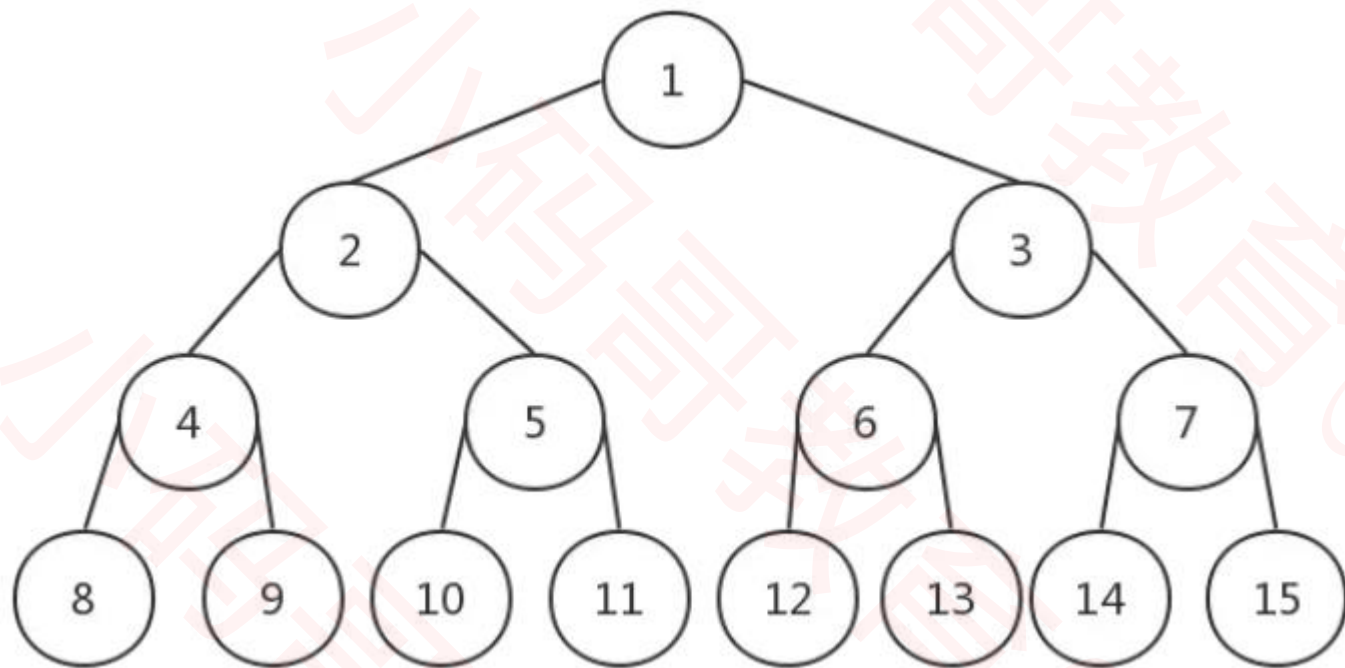


■ 下图不是真二叉树



# 满二叉树 (Full Binary Tree)

■ 满二叉树：所有非叶子节点的度都为2，且所有的叶子节点都在最后一层



■ 假设满二叉的高度为 $h$  ( $h \geq 1$ )，那么

□ 第 $i$ 层的节点数量： $2^{i-1}$

□ 叶子节点数量： $2^{h-1}$

□ 总节点数量 $n$

✓  $n = 2^h - 1$

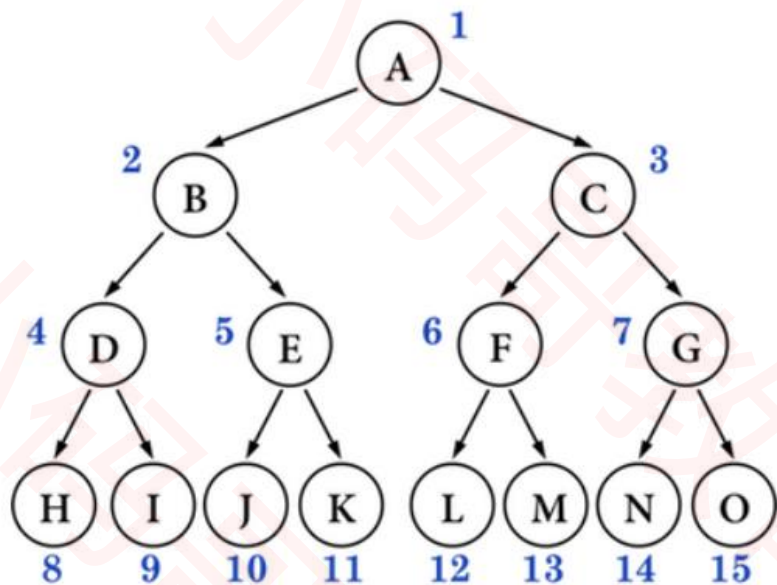
✓  $h = \log_2(n + 1)$

- 在同样高度的二叉树中，满二叉树的叶子节点数量最多、总节点数量最多
- 满二叉树一定是真二叉树，真二叉树不一定是满二叉树

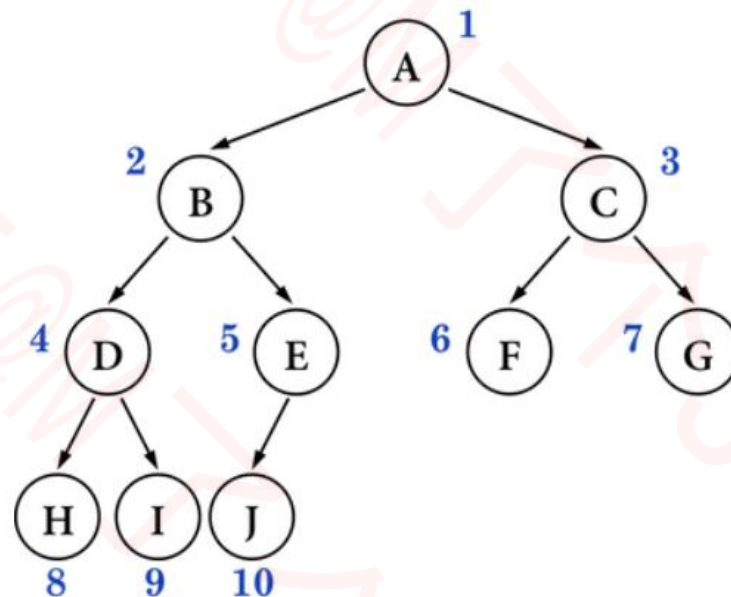
# 完全二叉树 (Complete Binary Tree)

- **完全二叉树**：高度为 $h$ 、有 $n$ 个结点的二叉树，从左至右、从上到下对节点进行编号，当其每一个结点都与高度为 $h$ 的满二叉树中相同编号的结点一一对应时，称之为完全二叉树

满二叉树



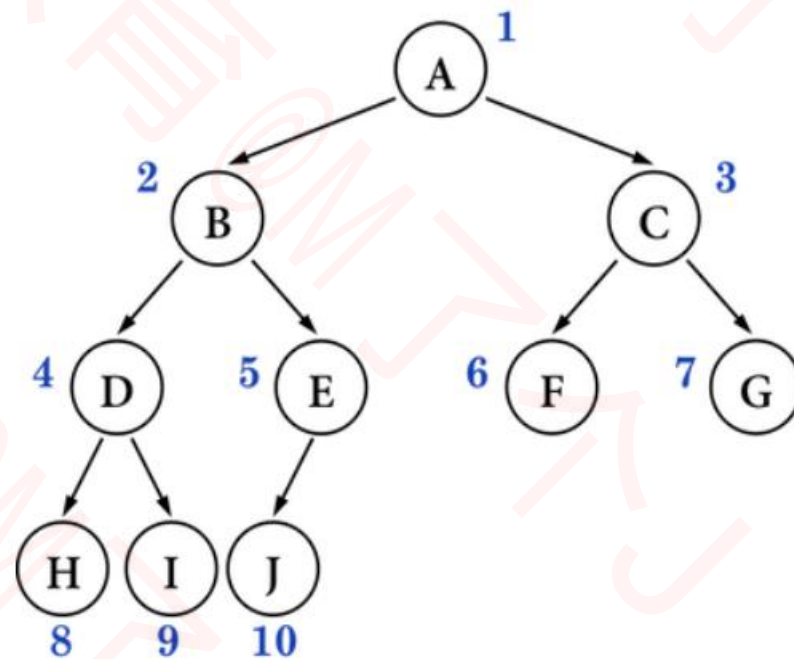
完全二叉树



- 完全二叉树从根结点至倒数第二层是一棵满二叉树，最后一层的叶子结点都靠左对齐
- 满二叉树一定是完全二叉树，完全二叉树不一定是满二叉树

# 完全二叉树的性质

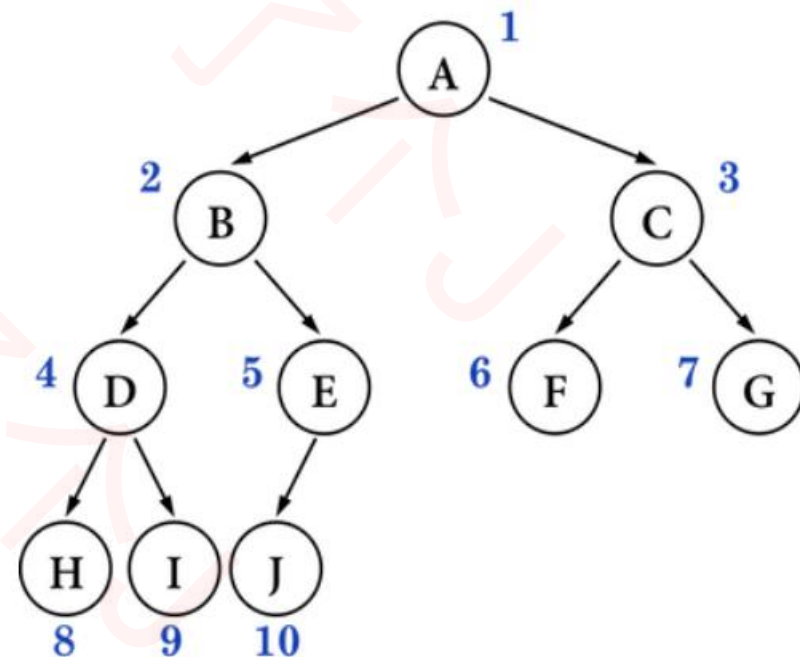
- 叶子节点只会出现在最下面2层
- 度为1的节点只有左子树
- 同样节点数量的二叉树，完全二叉树的高度最小
- 假设完全二叉树的高度为 $h$  ( $h \geq 1$ )，那么
  - 至少有  $2^{h-1}$  个节点
  - 最多有  $2^h - 1$  个节点
  - 总节点数量为 $n$ 
    - ✓  $h = \text{floor}(\log_2 n) + 1$
    - ✓ floor是向下取整的意思
    - $2^{h-1} \leq n < 2^h$
    - $h - 1 \leq \log_2 n < h$



完全二叉树

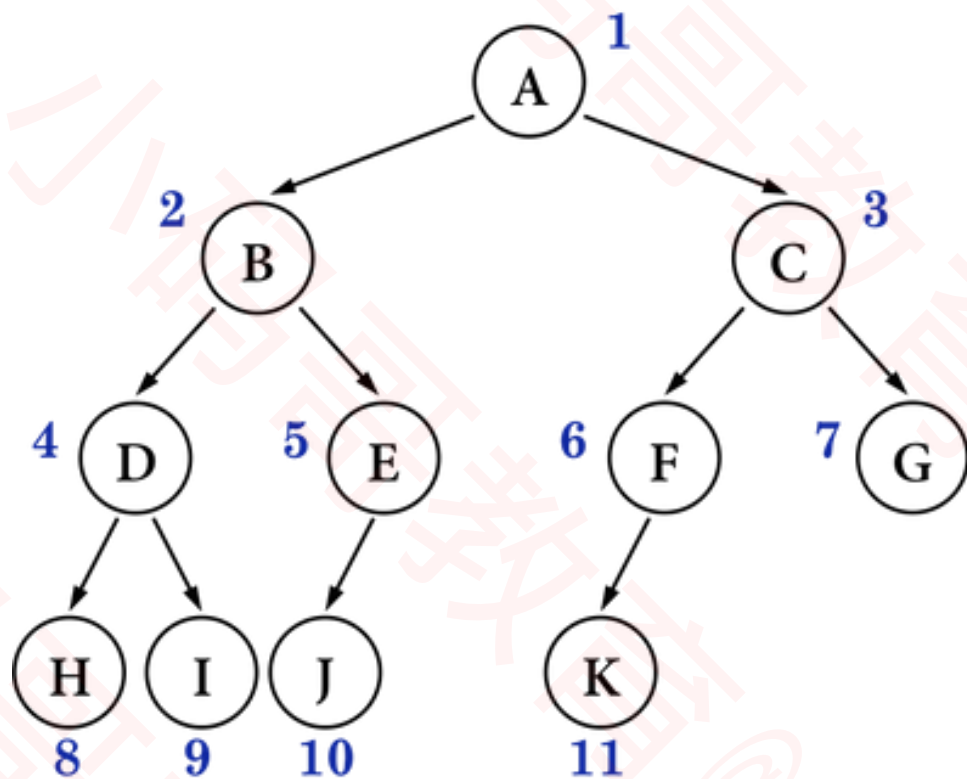
# 完全二叉树的性质

- 一棵有 $n$ 个节点的完全二叉树 ( $n > 0$ )，从上到下、从左到右对节点进行编号，对任意第 $i$ 个节点 ( $0 < i < n$ )
  - 如果 $i = 1$ ，它是根节点
  - 如果 $i > 1$ ，它的父节点编号为 $\text{floor}(i / 2)$
  - 如果 $2i \leq n$ ，它的左子节点编号为 $2i$
  - 如果 $2i > n$ ，它无左子节点
  - 如果 $2i + 1 \leq n$ ，它的右子节点编号为 $2i + 1$
  - 如果 $2i + 1 > n$ ，它无右子节点



完全二叉树

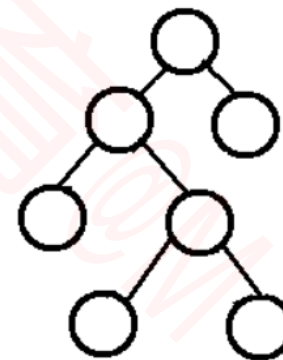
# 下图不是完全二叉树



# 国外教材的说法

## ■ Full Binary Tree: 完满二叉树

□ 所有非叶子节点的度都为2

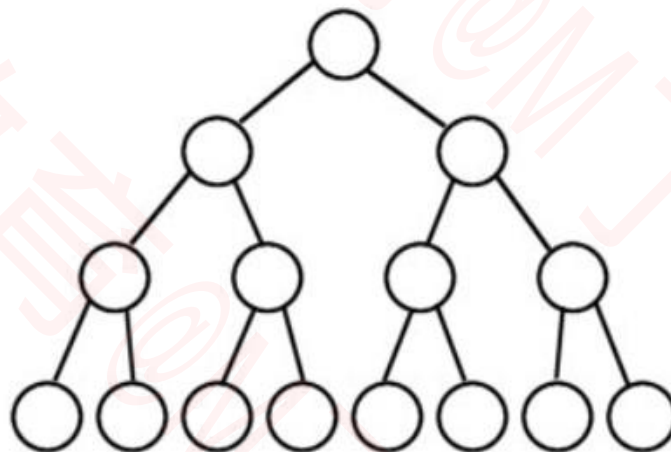


Full Binary Tree

## ■ Perfect Binary Tree: 完美二叉树

□ 所有非叶子节点的度都为2，且所有的叶子节点都在最后一层

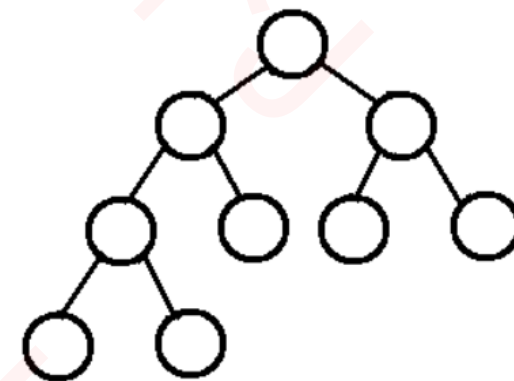
□ 就是国内说的“满二叉树”



Perfect Binary Tree

## ■ Complete Binary Tree: 完全二叉树

□ 跟国内的定义一样



Complete Binary Tree



# 练习 - 翻转二叉树

■ <https://leetcode-cn.com/problems/invert-binary-tree/>

输入:



输出:



■ 请分别用递归、迭代（非递归）两种方式实现

# 二叉树的遍历

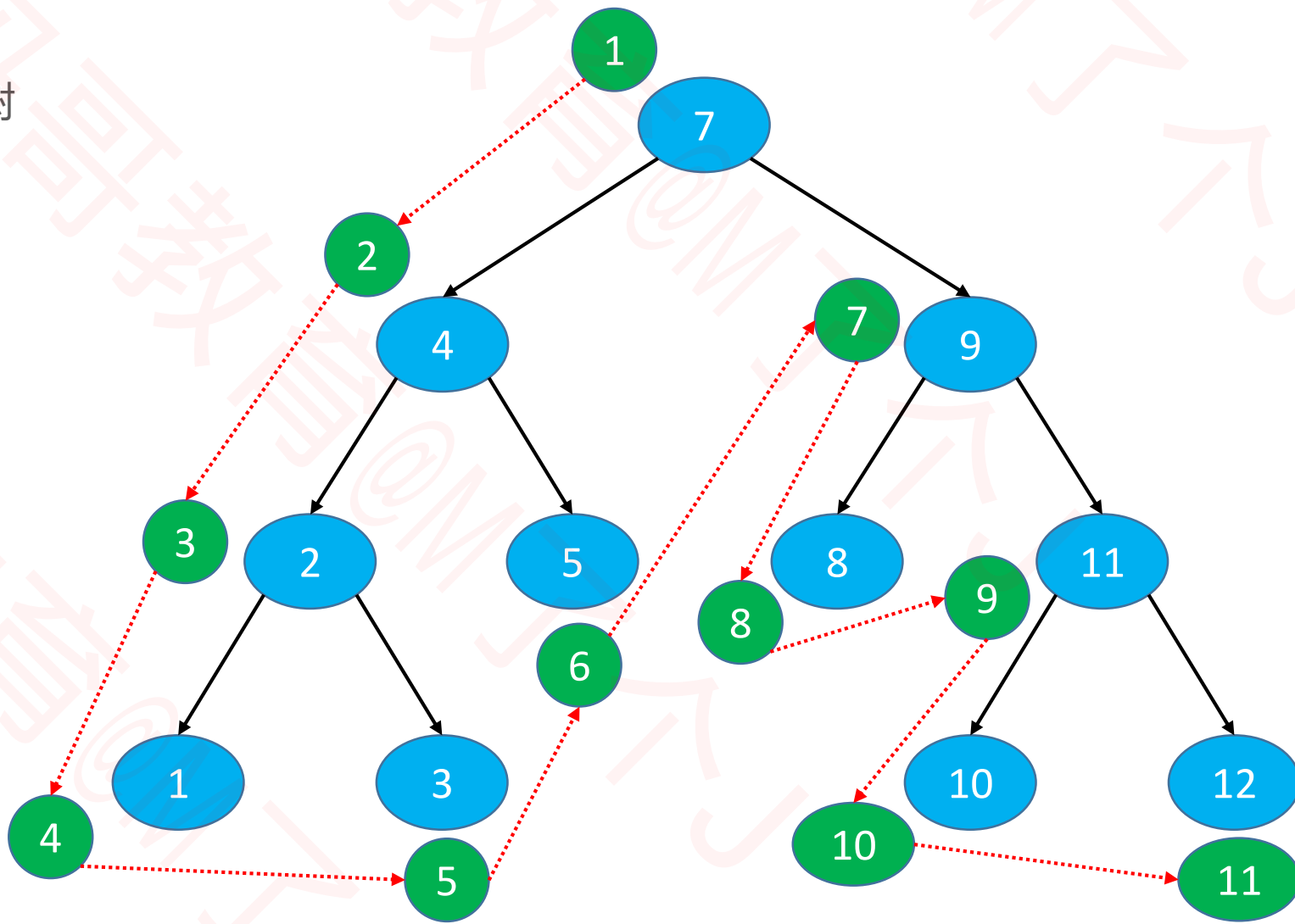
- 遍历是数据结构中的常见操作
  - 把所有元素都访问一遍
- 线性数据结构的遍历比较简单
  - 正序遍历
  - 逆序遍历
- 根据节点访问顺序的不同，二叉树的常见遍历方式有4种
  - 前序遍历 (Preorder Traversal)
  - 中序遍历 (Inorder Traversal)
  - 后序遍历 (Postorder Traversal)
  - 层序遍历 (Level Order Traversal)

# 前序遍历 (Preorder Traversal)

■ 访问顺序

□ 根节点、前序遍历左子树、前序遍历右子树

□ 7、4、2、1、3、5、9、8、11、10、12



# 中序遍历 (Inorder Traversal)

## ■ 访问顺序

□ 中序遍历左子树、根节点、中序遍历右子树

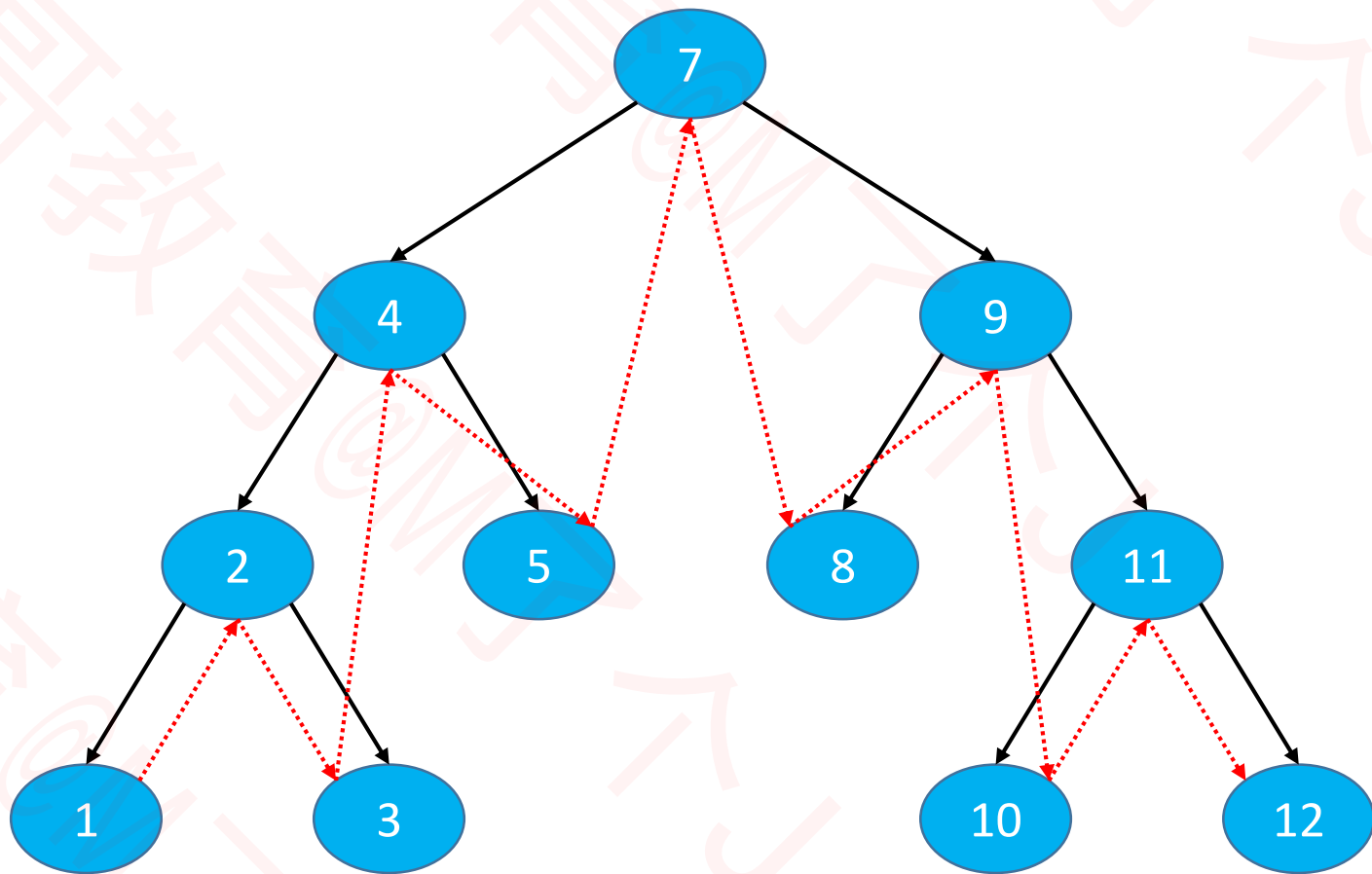
□ 1、2、3、4、5、7、8、9、10、11、12

## ■ 如果访问顺序是下面这样呢？

□ 中序遍历右子树、根节点、中序遍历左子树

□ 12、11、10、9、8、7、5、4、3、2、1

## ■ 二叉搜索树的中序遍历结果是升序或者降序的

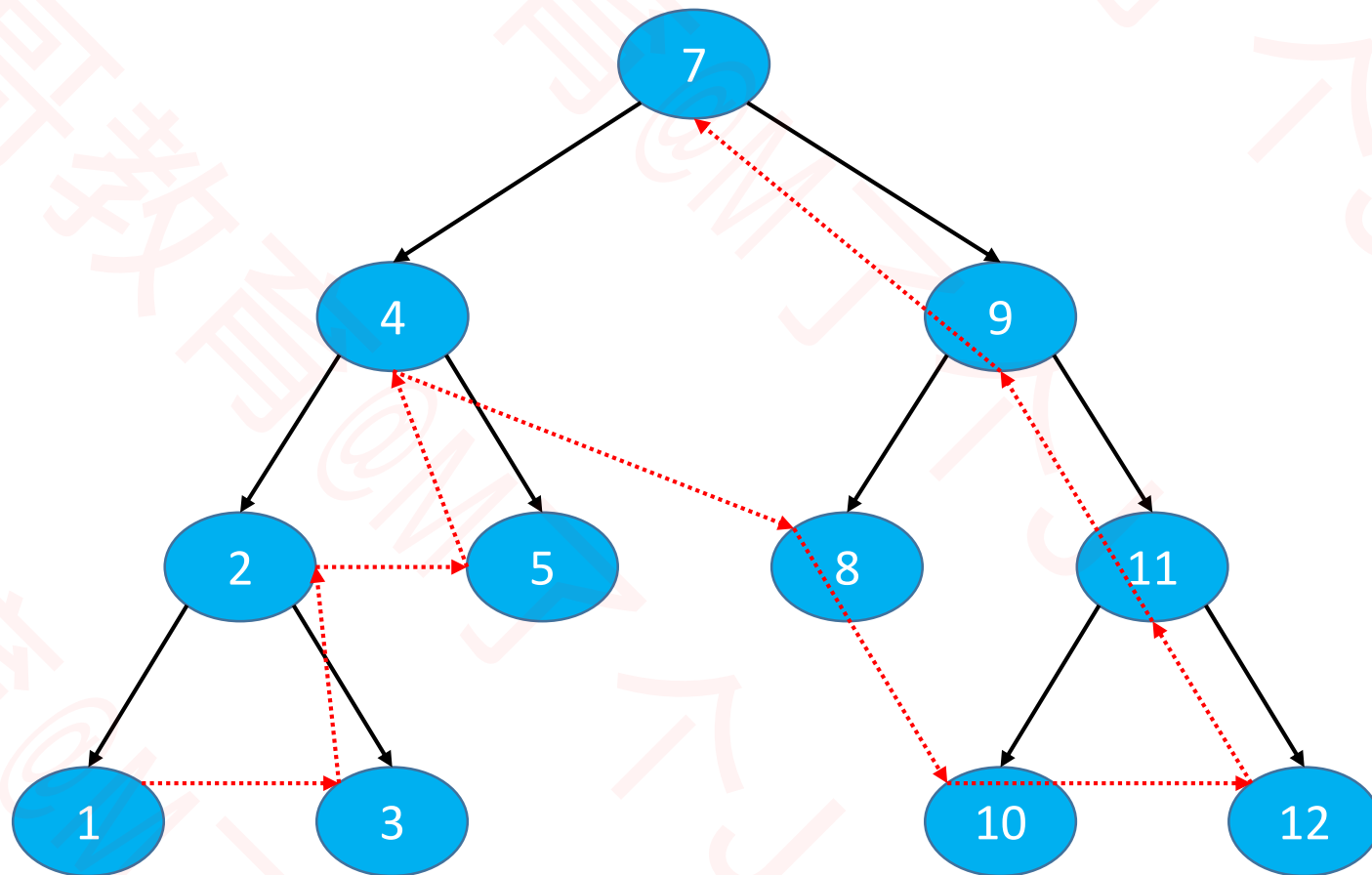


# 后序遍历 (Postorder Traversal)

## ■ 访问顺序

□ 后序遍历左子树、后序遍历右子树、根节点

□ 1、3、2、5、4、8、10、12、11、9、7



# 层序遍历 (Level Order Traversal)

## ■ 访问顺序

□ 从上到下、从左到右依次访问每一个节点

□ 7、4、9、2、5、8、11、1、3、10、12

## ■ 实现思路：使用队列

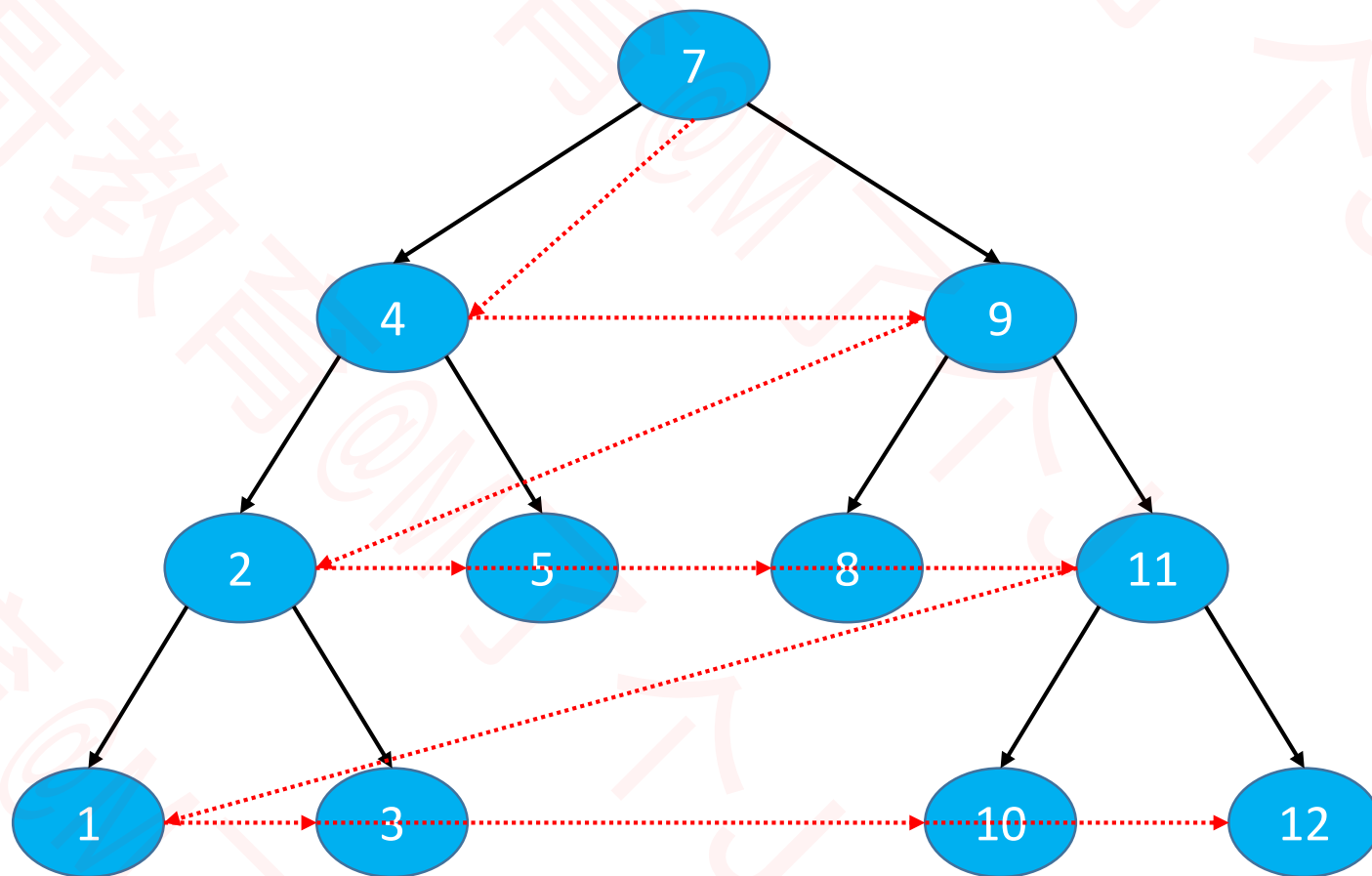
1. 将根节点入队

2. 循环执行以下操作，直到队列为空

□ 将队头节点A出队，进行访问

□ 将A的左子节点入队

□ 将A的右子节点入队



# 遍历的应用

## ■ 前序遍历

- 树状结构展示（注意左右子树的顺序）

## ■ 中序遍历

- 二叉搜索树的中序遍历按升序或者降序处理节点

## ■ 后序遍历

- 适用于一些先子后父的操作

## ■ 层序遍历

- 计算二叉树的高度
- 判断一棵树是否为完全二叉树



# 根据遍历结果重构二叉树

■ 以下结果可以保证重构出唯一的一棵二叉树

□ 前序遍历 + 中序遍历

□ 中序遍历 + 后序遍历

□ 前序遍历 + 后序遍历

✓ 如果它是一棵真二叉树 (Proper Binary Tree) , 结果是唯一的

✓ 不然结果不唯一

# 练习 – 计算二叉树的高度

■ 递归

■ 迭代

# 练习 – 判断一棵树是否为完全二叉树

■ 如果树为空，返回false

■ 如果树不为空，开始层序遍历二叉树（用队列）

□ 如果`node.left != null` && `node.right != null`，将`node.left`、`node.right`按顺序入队

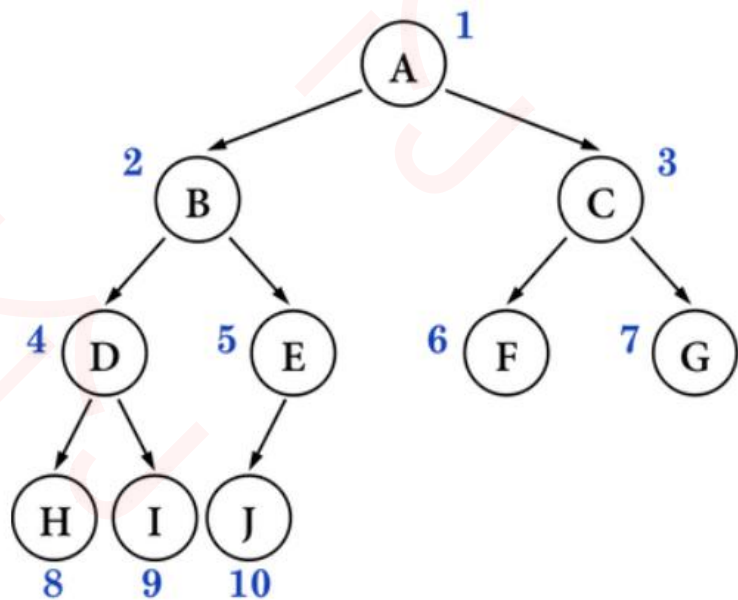
□ 如果`node.left == null` && `node.right != null`，返回false

□ 如果`node.left != null` && `node.right == null` 或者 `node.left == null` && `node.right == null`

✓ 那么后面遍历的节点应该都为叶子节点，才是完全二叉树

✓ 否则返回false

□ 遍历结束，返回true



# 前驱节点 (predecessor)

- 前驱节点：中序遍历时的前一个节点
- 如果是二叉搜索树，前驱节点就是前一个比它小的节点

■ `node.left != null`

□ 举例：6、13、8

□ `predecessor = node.left.right.right.right...`

✓ 终止条件：`right`为`null`

■ `node.left == null && node.parent != null`

□ 举例：7、11、9、1

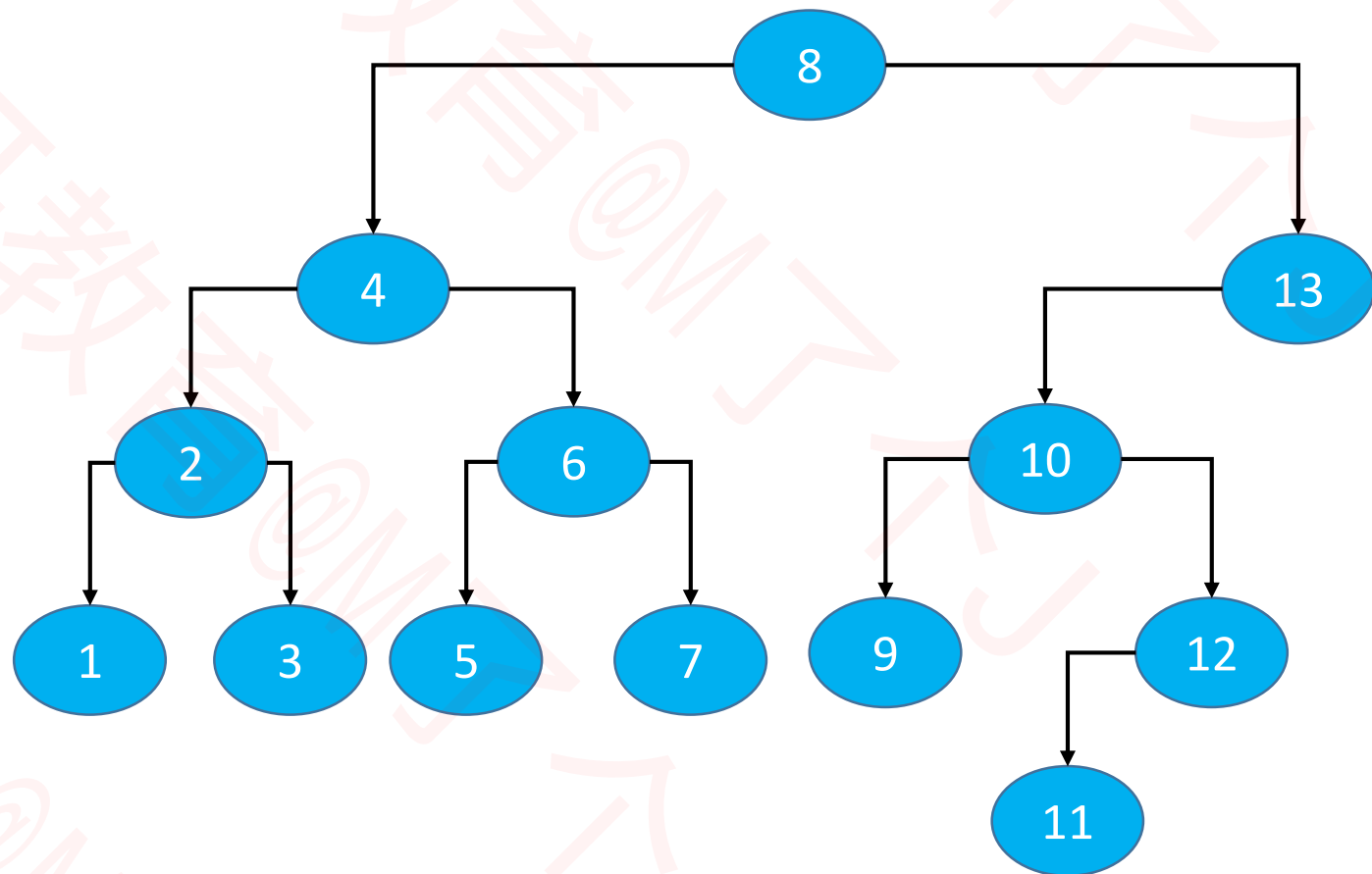
□ `predecessor = node.parent.parent.parent...`

✓ 终止条件：`node`在`parent`的右子树中

■ `node.left == null && node.parent == null`

□ 那就没有前驱节点

□ 举例：没有左子树的根节点



1、2、3、4、5、6、7、8、9、10、11、12、13

# 后继节点 (successor)

- 后继节点：中序遍历时的后一个节点
- 如果是二叉搜索树，后继节点就是后一个比它大的节点

■ `node.right != null`

□ 举例：1、8、4

□ `successor = node.right.left.left.left...`

✓ 终止条件：left为null

■ `node.right == null && node.parent != null`

□ 举例：7、6、3、11

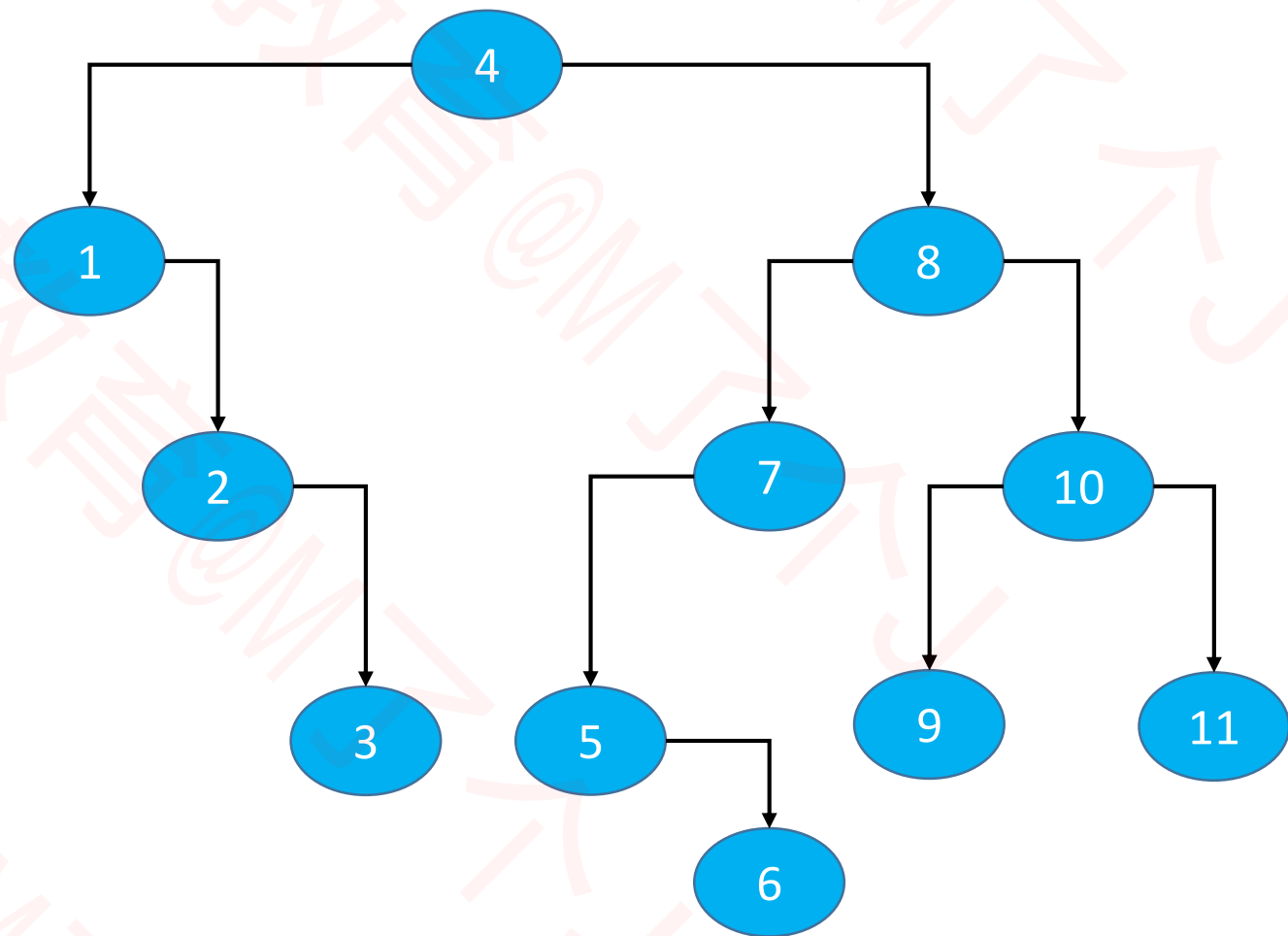
□ `successor = node.parent.parent.parent...`

✓ 终止条件：node在parent的左子树中

■ `node.right == null && node.parent == null`

□ 那就没有前驱节点

□ 举例：没有右子树的根节点



1、2、3、4、5、6、7、8、9、10、11

- 如果允许外界遍历二叉树的元素？二叉树该如何设计接口？

- 二叉树的前序遍历: <https://leetcode-cn.com/problems/binary-tree-preorder-traversal/> (递归+迭代)
- 二叉树的中序遍历: <https://leetcode-cn.com/problems/binary-tree-inorder-traversal/> (递归+迭代)
- 二叉树的后序遍历: <https://leetcode-cn.com/problems/binary-tree-postorder-traversal/> (递归+迭代)
- 二叉树的层次遍历: <https://leetcode-cn.com/problems/binary-tree-level-order-traversal/> (迭代)
- 二叉树的最大深度: <https://leetcode-cn.com/problems/maximum-depth-of-binary-tree/> (递归+迭代)



- 二叉树的层次遍历II: <https://leetcode-cn.com/problems/binary-tree-level-order-traversal-ii/>
- 二叉树最大宽度: <https://leetcode-cn.com/problems/maximum-width-of-binary-tree/>
- N叉树的前序遍历: <https://leetcode-cn.com/problems/n-ary-tree-preorder-traversal/>
- N叉树的后序遍历: <https://leetcode-cn.com/problems/n-ary-tree-postorder-traversal/>
- N叉树的最大深度: <https://leetcode-cn.com/problems/maximum-depth-of-n-ary-tree/>

- 二叉树展开为链表

- <https://leetcode-cn.com/problems/flatten-binary-tree-to-linked-list/>

- 从中序与后序遍历序列构造二叉树

- <https://leetcode-cn.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/>

- 从前序与中序遍历序列构造二叉树

- <https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/>

- 根据前序和后序遍历构造二叉树

- <https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-postorder-traversal/>

# 作业

- 已知前序、中序遍历结果，求出后序遍历结果
- 已知中序、后序遍历结果，求出前序遍历结果
- 再思考一种二叉树的非递归前序遍历方法（要跟课堂上的不同实现）