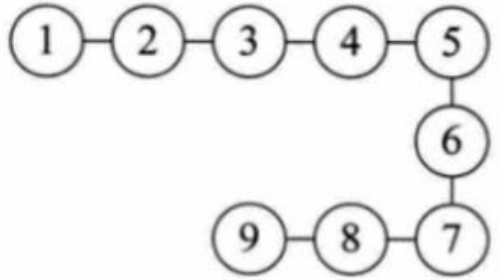
二叉树

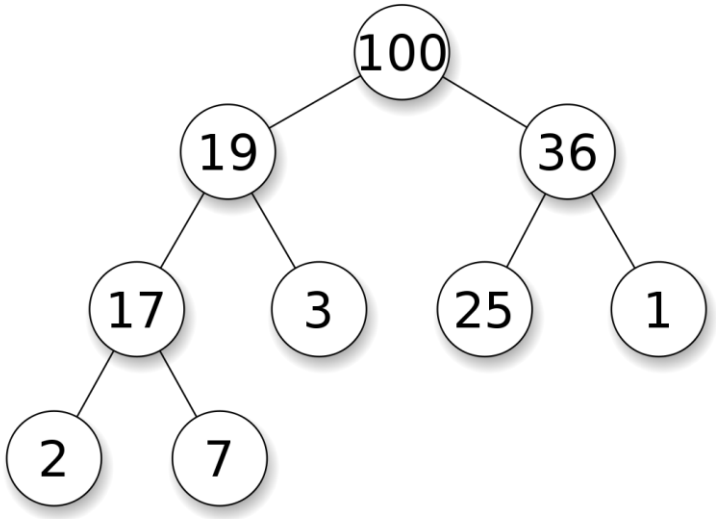
@M了个J

https://github.com/CoderMJLee
http://cnblogs.com/mjios

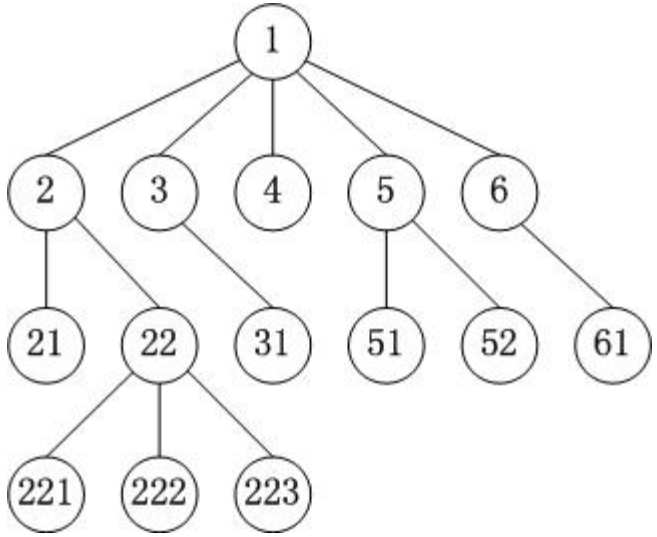
树形结构



线性结构

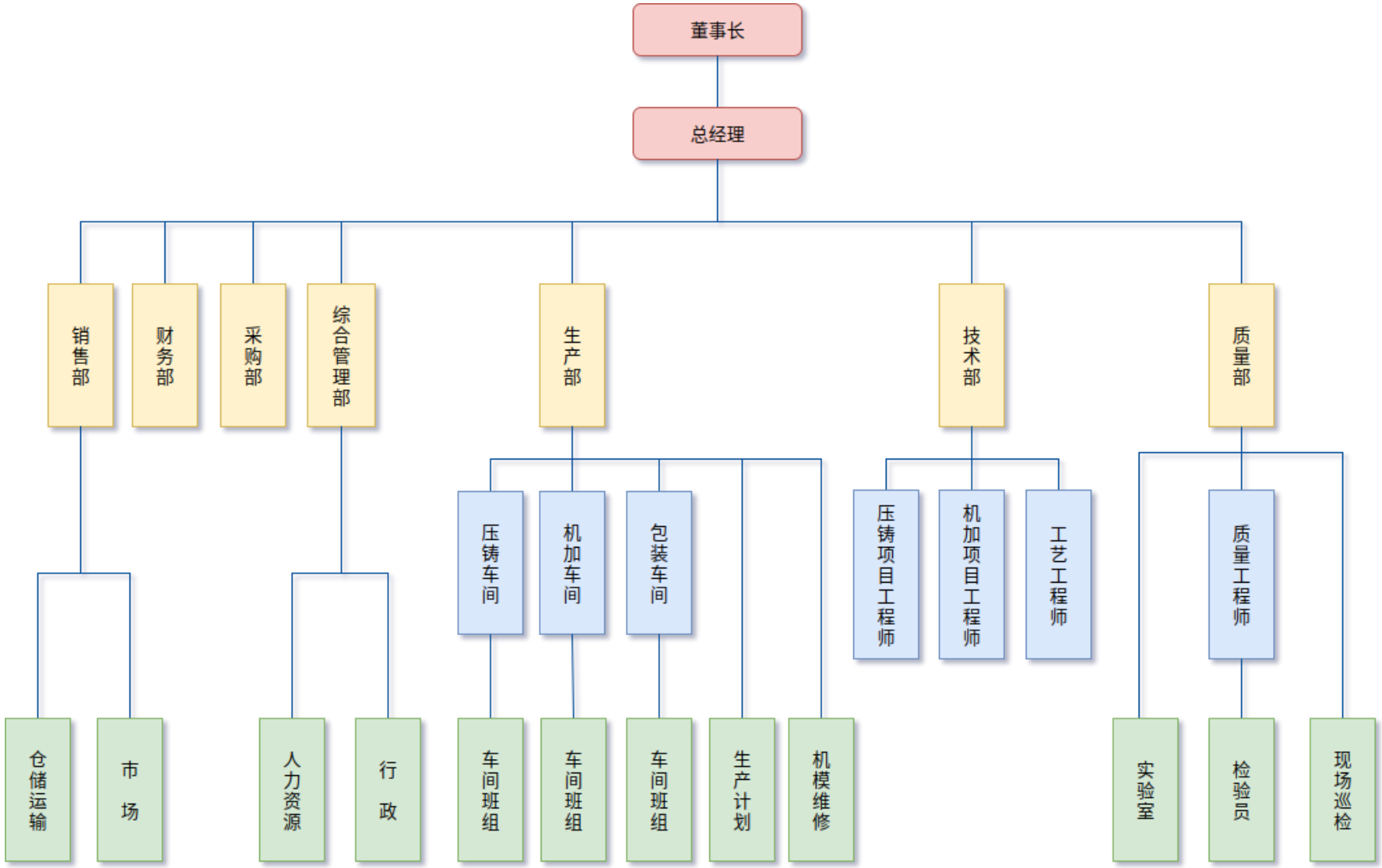


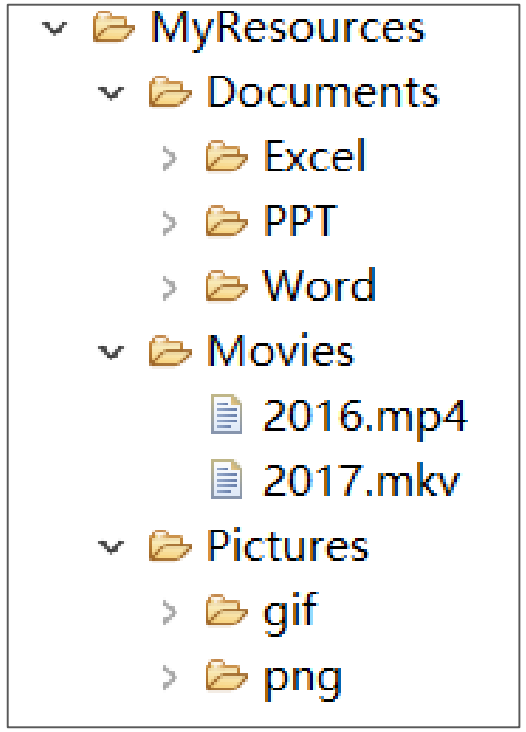
二叉树



多叉树

生活中的树形结构



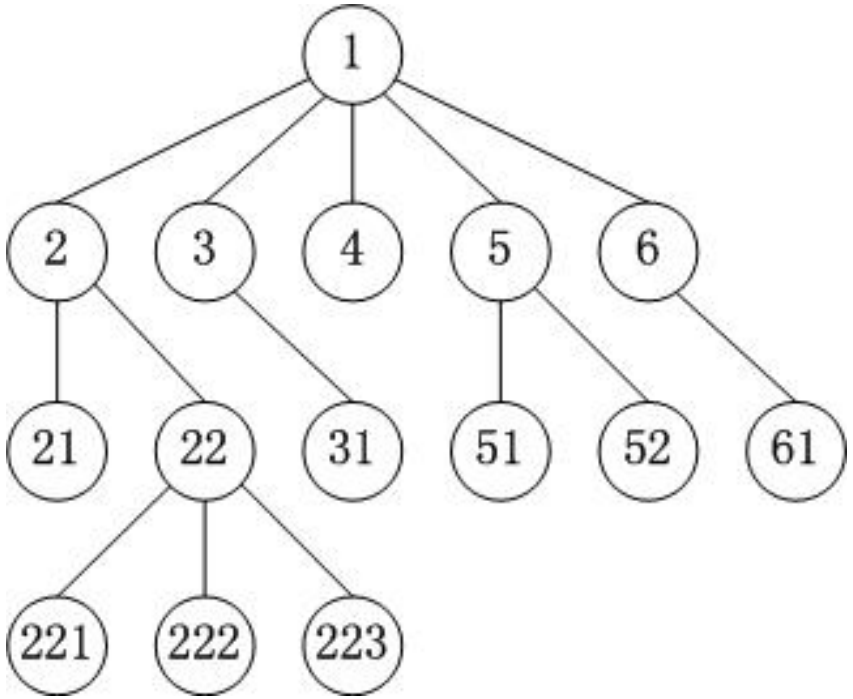


- 使用树形结构可以大大提高效率
- 树形结构是算法面试的重点

树（Tree）的基本概念

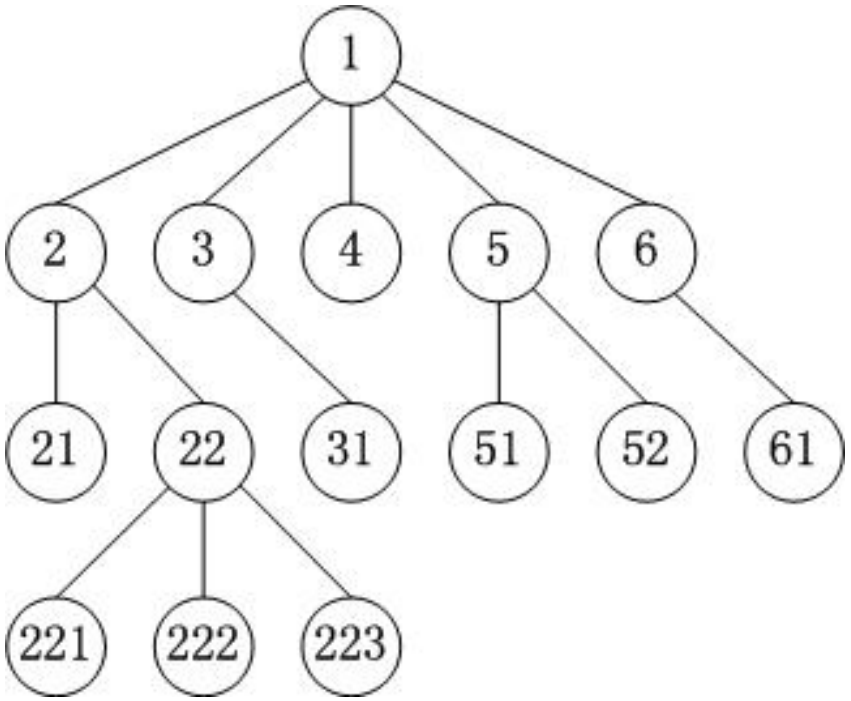
- 节点、根节点、父节点、子节点、兄弟节点
- 一棵树可以没有任何节点，称为空树
- 一棵树可以只有 1 个节点，也就是只有根节点
- 子树、左子树、右子树

- 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

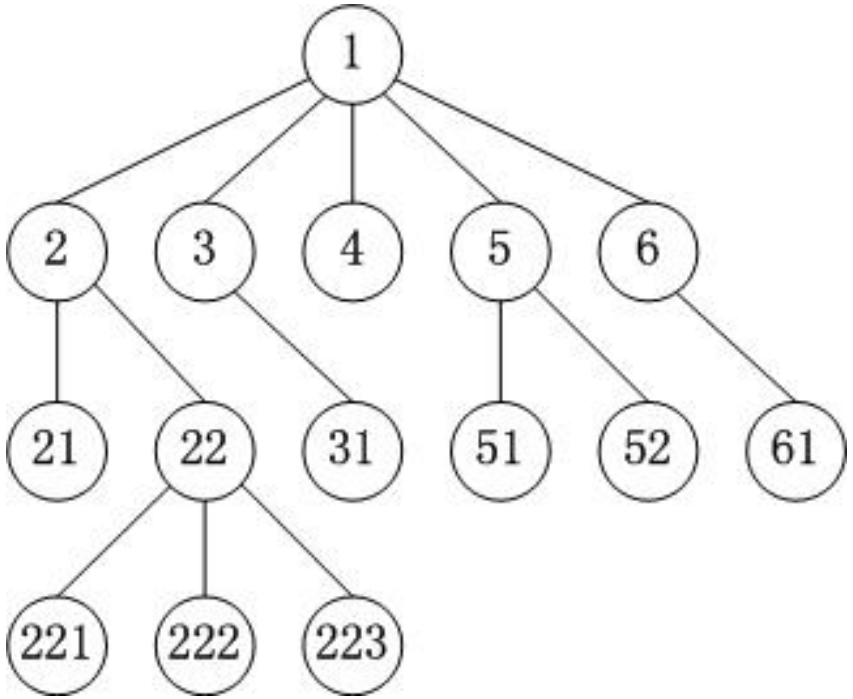


有序树、无序树、森林

- 有序树
  - 树中任意节点的子节点之间有顺序关系

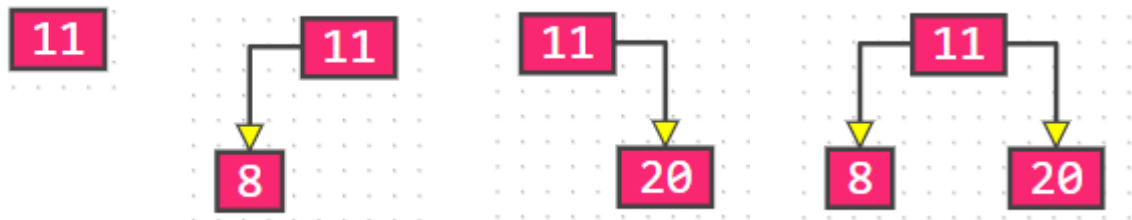
- 无序树
  - 树中任意节点的子节点之间没有顺序关系
  - 也称为“自由树”

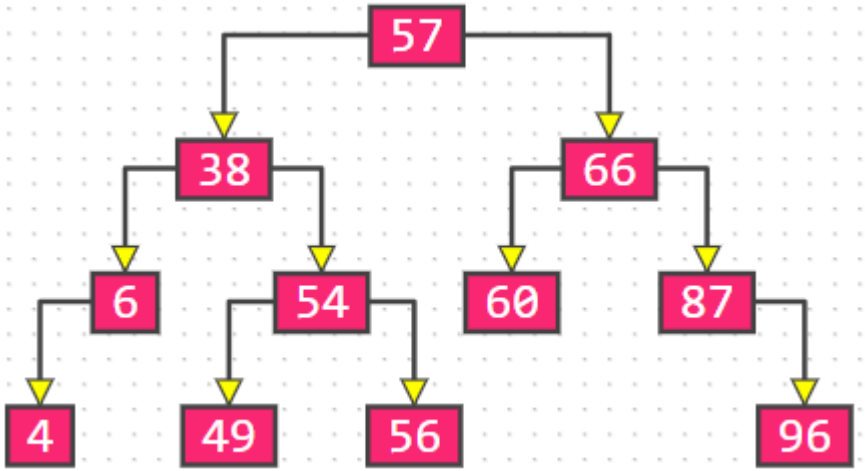
- 森林
  - 由 m（m ≥ 0）棵互不相交的树组成的集合

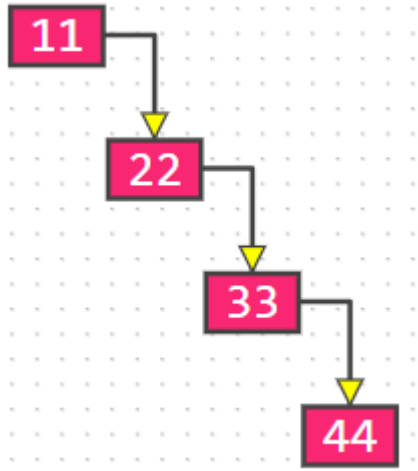


二叉树（Binary Tree）

- 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







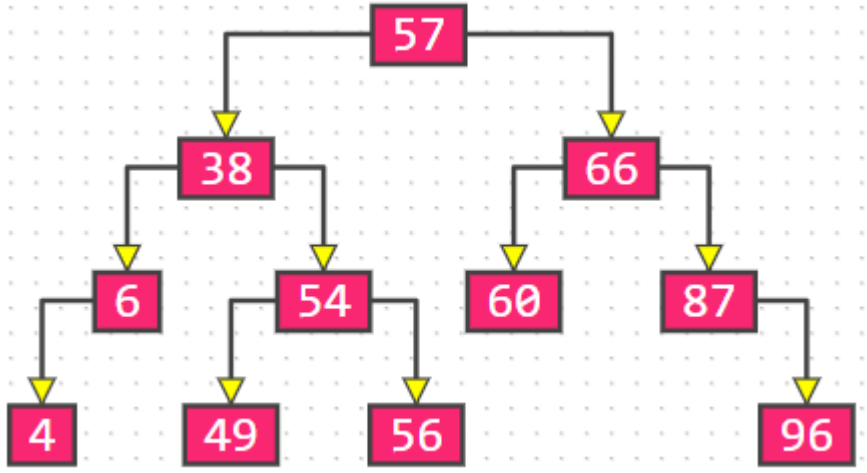
- 二叉树是有序树 还是 无序树?
  - 有序树

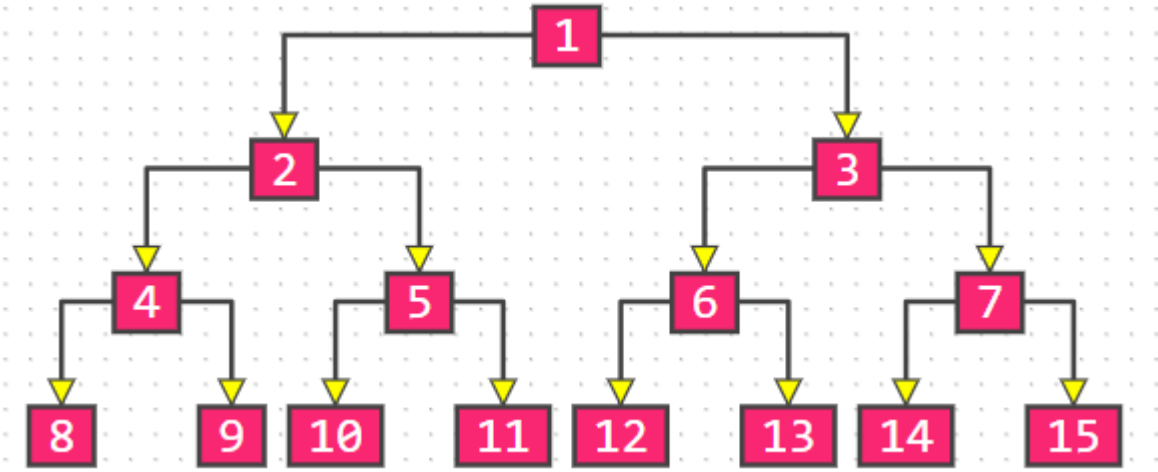
二叉树的性质

- 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（ $h \geq 1$ ）

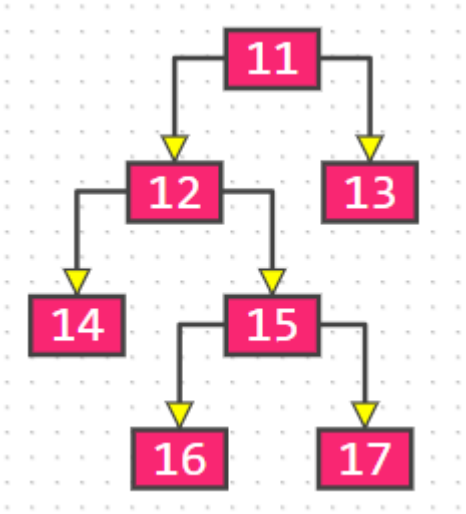
- 对于任何一棵非空二叉树，如果叶子节点个数为 n0，度为 2 的节点个数为 n2，则有: n0 = n2 + 1
  - 假设度为 1 的节点个数为 n1，那么二叉树的节点总数 n = n0 + n1 + n2
  - 二叉树的边数 T = n1 + 2 * n2 = n – 1 = n0 + n1 + n2 – 1
  - 因此 n0 = n2 + 1



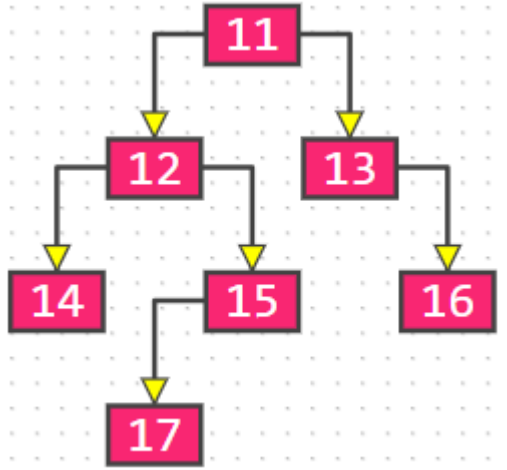


真二叉树（Proper Binary Tree）

- 真二叉树：所有节点的度都要么为 0，要么为 2

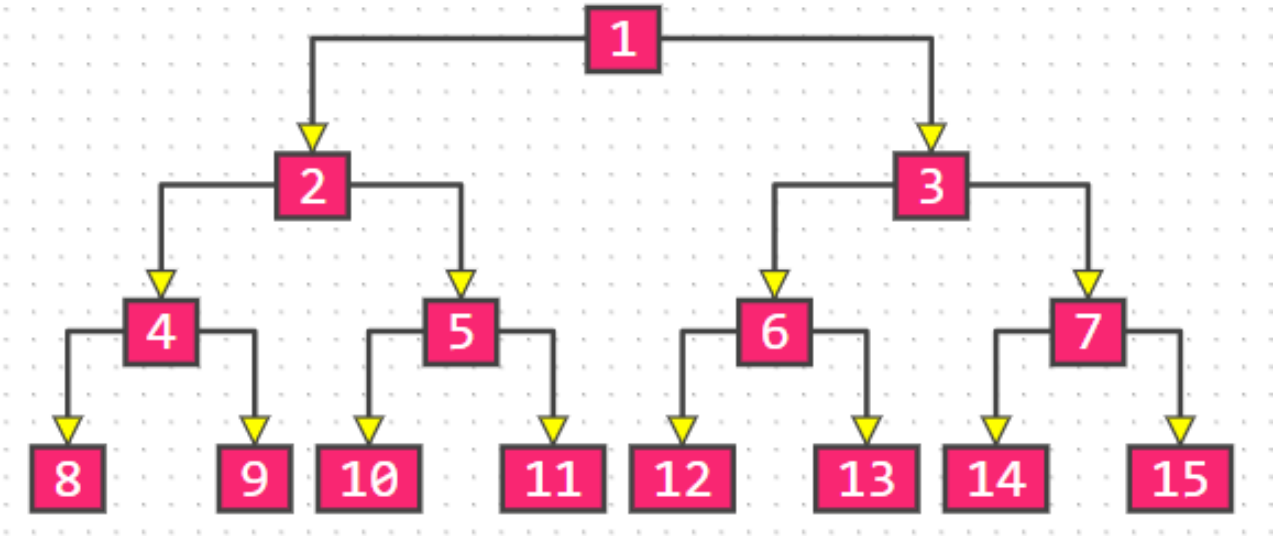


- 下图不是真二叉树



满二叉树（Full Binary Tree）

- 满二叉树：最后一层节点的度都为 0，其他节点的度都为 2



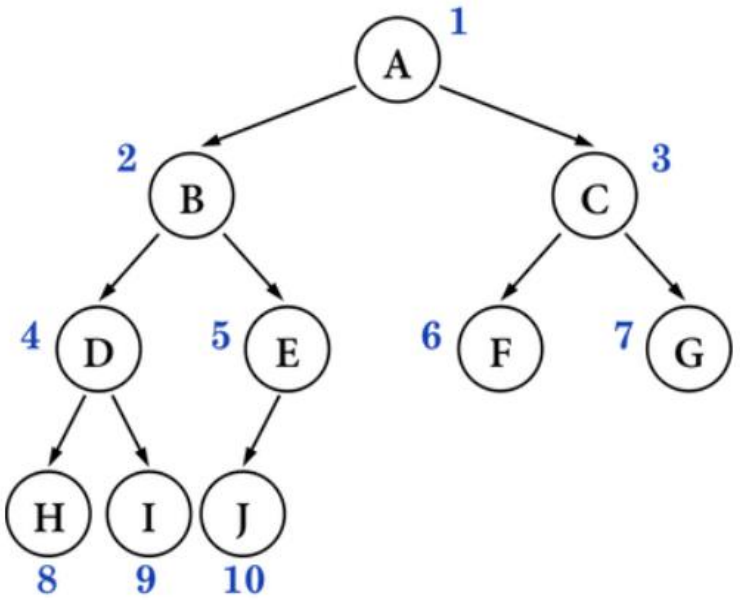
- 假设满二叉树的高度为 h（$h \geq 1$），那么
  - 第 i 层的节点数量：2^{i-1}
  - 叶子节点数量：2^{h-1}
  - 总节点数量 n
    - ✓ $n = 2^h - 1 = 2^0 + 2^1 + 2^2 + \cdots + 2^{h-1}$
    - ✓ $h = \log_2(n+1)$

- 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

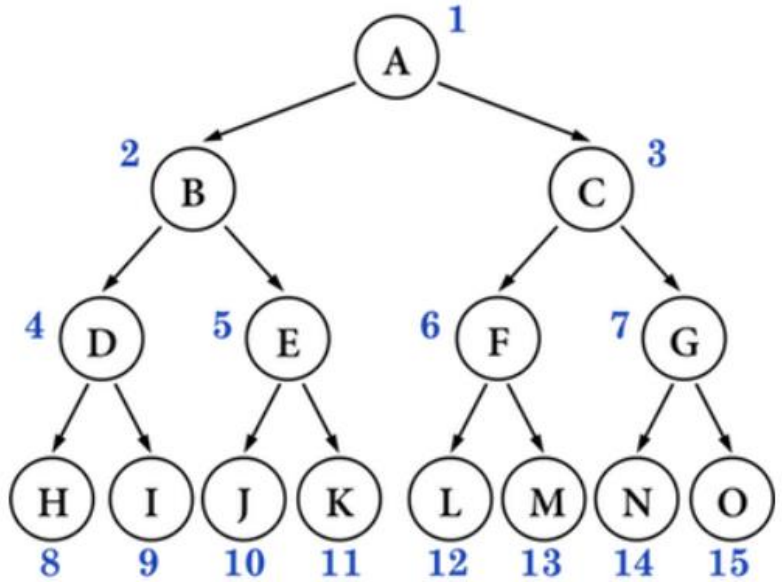
完全二叉树（Complete Binary Tree）

- 完全二叉树：对节点从上至下、左至右开始编号，其所有编号都能与相同高度的满二叉树中的编号对应

完全二叉树



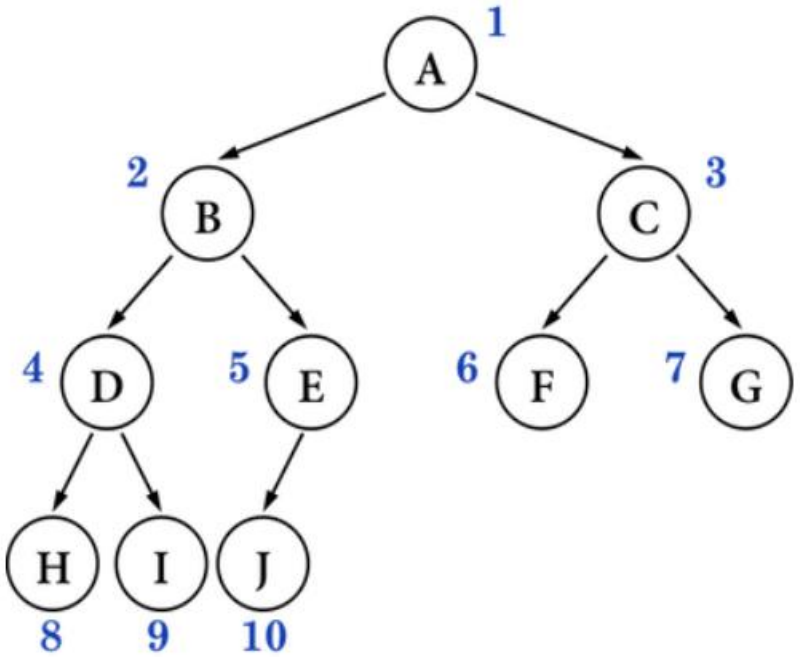
满二叉树



- 叶子节点只会出现最后 2 层，最后 1 层的叶子结点都靠左对齐
- 完全二叉树从根结点至倒数第 2 层是一棵满二叉树
- 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

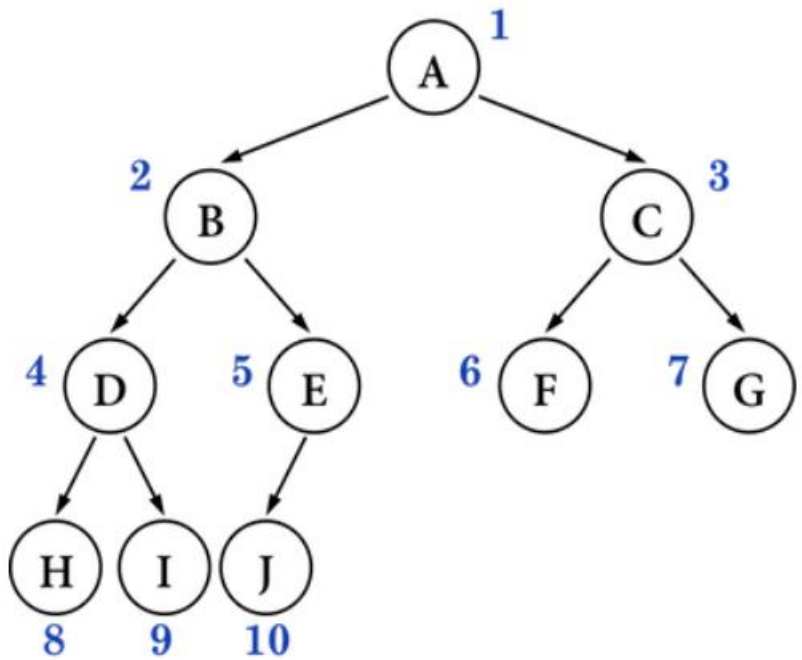
- 度为 1 的节点只有左子树
- 度为 1 的节点要么是 1 个，要么是 0 个
- 同样节点数量的二叉树，完全二叉树的高度最小
- 假设完全二叉树的高度为 h（ $h \geq 1$ ），那么
  - 至少有 2^{h-1} 个节点 （ $2^0 + 2^1 + 2^2 + \cdots + 2^{h-2} + 1$ ）
  - 最多有 $2^h - 1$ 个节点（ $2^0 + 2^1 + 2^2 + \cdots + 2^{h-1}$，满二叉树 ）
  - 总节点数量为 n
    - ✓ $2^{h-1} \leq n < 2^h$
    - ✓ $h - 1 \leq \log_2 n < h$
    - ✓ $h = \text{floor}(\log_2 n) + 1$
    - ➢ floor 是向下取整，另外，ceiling 是向上取整



完全二叉树

完全二叉树的性质

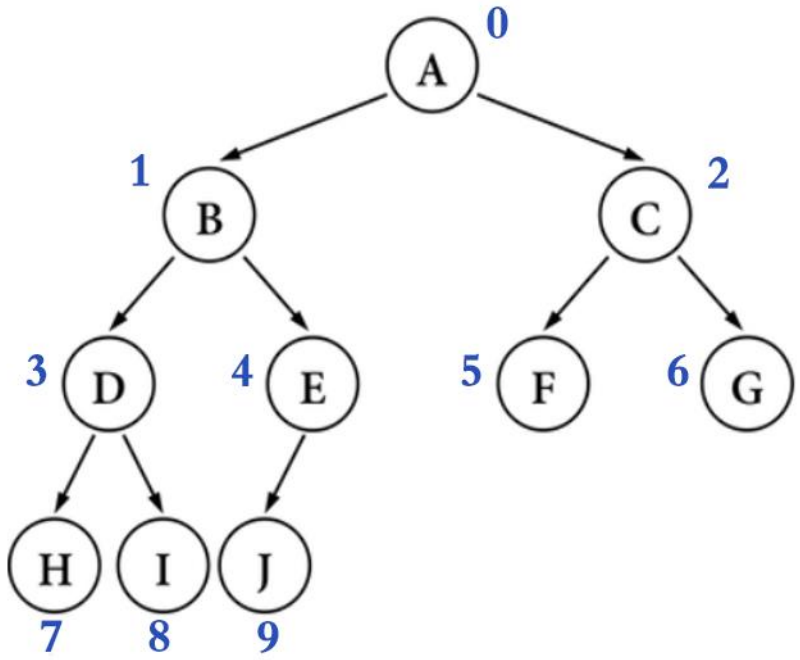
- 一棵有 n 个节点的完全二叉树（n > 0），从上到下、从左到右对节点从 1 开始进行编号，对任意第 i 个节点
  - 如果 i = 1，它是根节点
  - 如果 i > 1，它的父节点编号为 floor(i / 2)
  - 如果 2i ≤ n，它的左子节点编号为 2i
  - 如果 2i > n，它无左子节点
  - 如果 2i + 1 ≤ n，它的右子节点编号为 2i + 1
  - 如果 2i + 1 > n，它无右子节点



完全二叉树

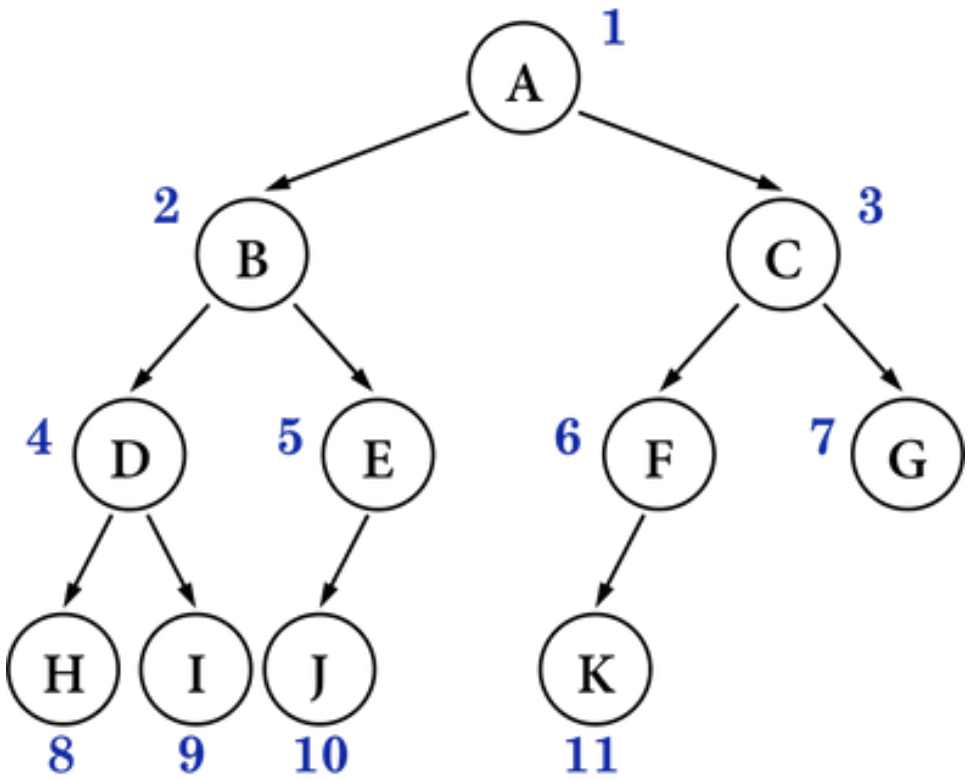
完全二叉树的性质

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



完全二叉树

下图不是完全二叉树



面试题

■ 如果一棵完全二叉树有 768 个节点，求叶子节点的个数

◘ 假设叶子节点个数为 n0，度为 1 的节点个数为 n1，度为 2 的节点个数为 n2

◘ 总结点个数 n = n0 + n1 + n2，而且 n0 = n2 + 1

✓ n = 2n0 + n1 – 1

◘ 完全二叉树的 n1 要么为 0，要么为 1

✓ n1为1时，n = 2n0，n 必然是偶数

➢ 叶子节点个数 n0 = n / 2，非叶子节点个数 n1 + n2 = n / 2

✓ n1为0时，n = 2n0 – 1，n 必然是奇数

➢ 叶子节点个数 n0 = (n + 1) / 2，非叶子节点个数 n1 + n2 = (n – 1) / 2

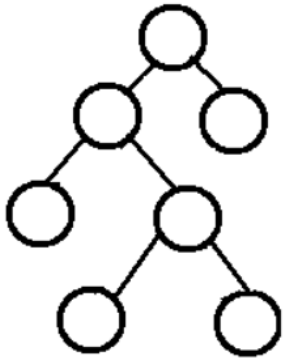
◘ 叶子节点个数 n0 = floor((n + 1) / 2) = ceiling(n / 2)

◘ 非叶子节点个数 n1 + n2 = floor(n / 2) = ceiling((n – 1) / 2)

◘ 因此叶子节点个数为 384

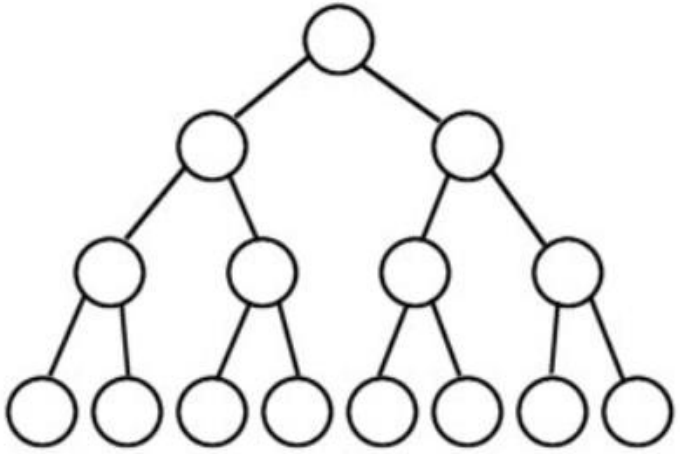
国外教材的说法

- Full Binary Tree：完满二叉树
  - 所有非叶子节点的度都为 2
  - 就的国内说的“真二叉树”



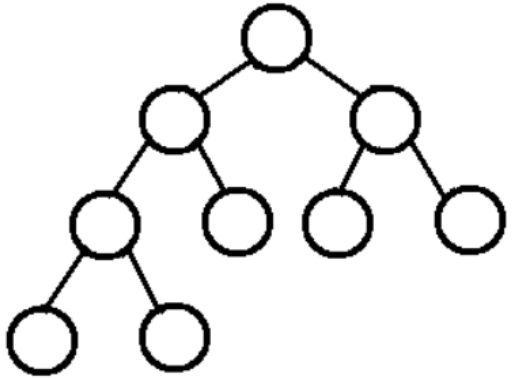
Full Binary Tree

- 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

二叉树的遍历

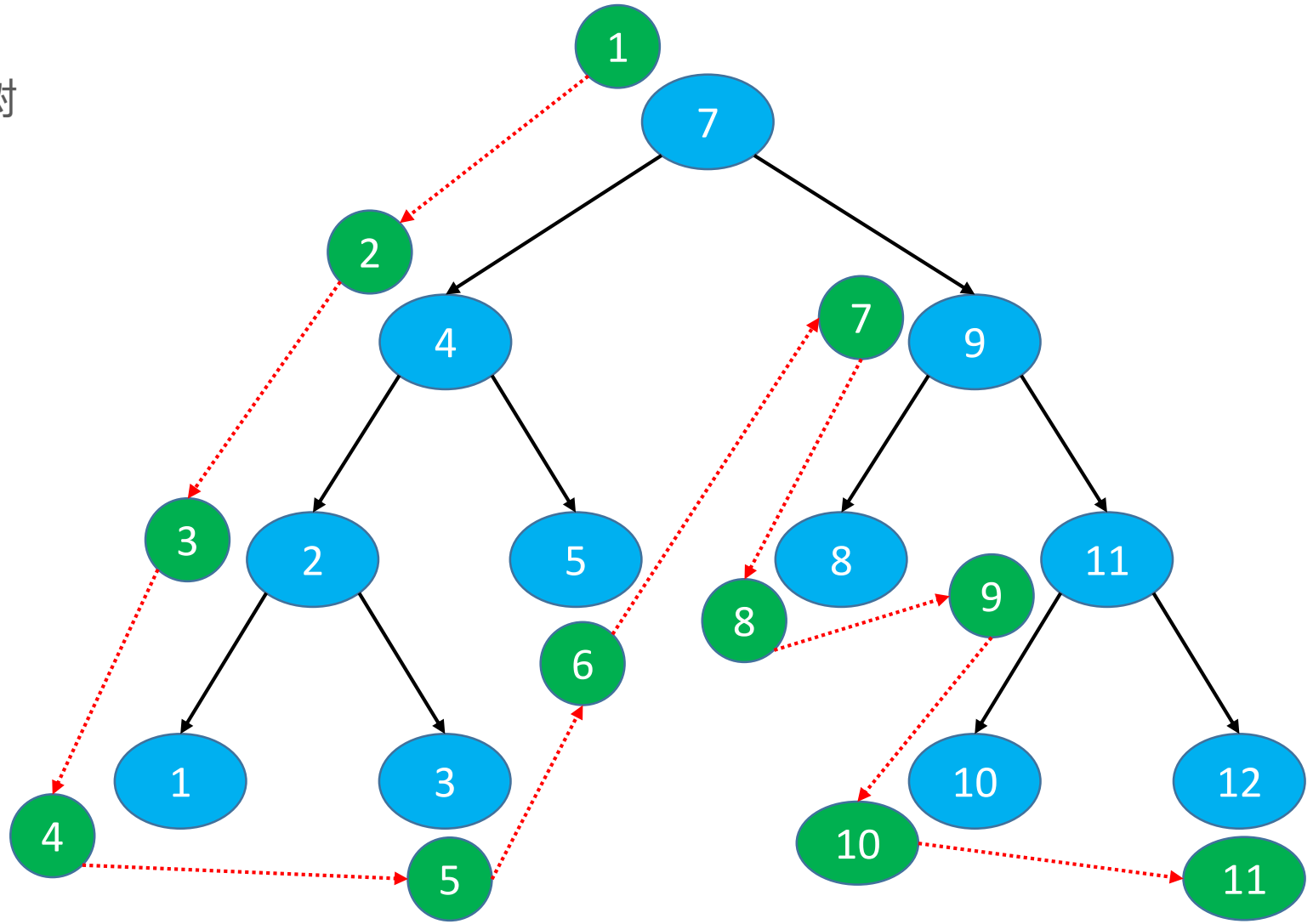
- 遍历是数据结构中的常见操作
  - 把所有元素都访问一遍

- 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



前序遍历 – 非递归

■ 利用栈实现

1. 设置 node = root
2. 循环执行以下操作

□ 如果 node != null

✓ 对 node 进行访问

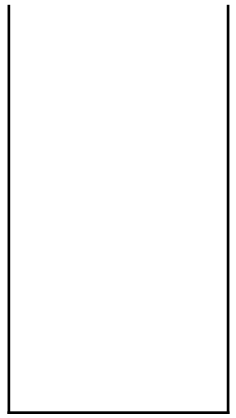
✓ 将 node.right 入栈

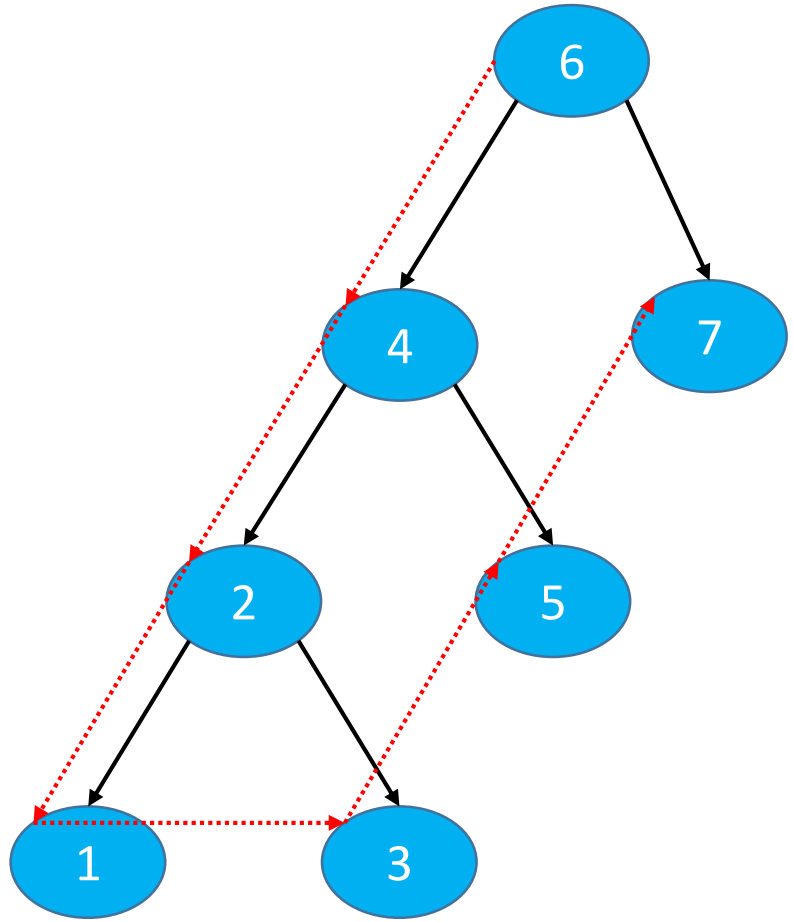
✓ 设置 node = node.left

□ 如果 node == null

✓ 如果栈为空，结束遍历

✓ 如果栈不为空，弹出栈顶元素并赋值给 node

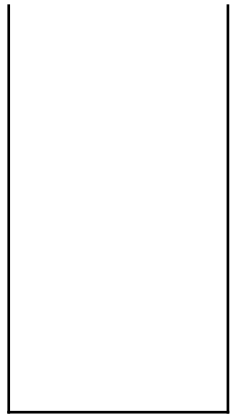


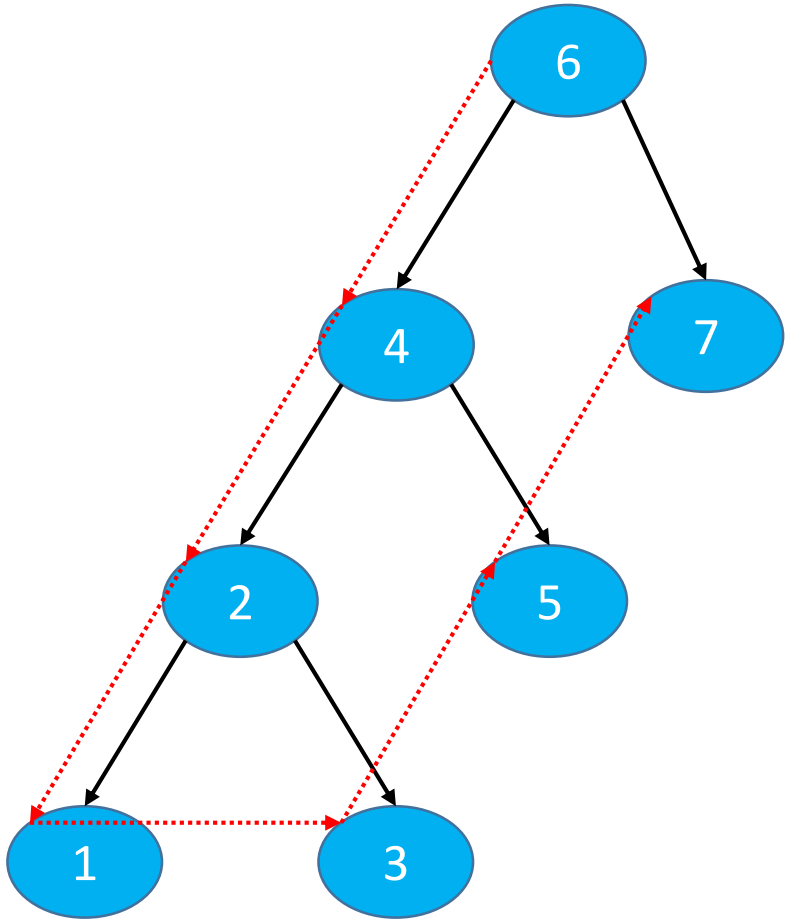


前序遍历 – 非递归

- 利用栈实现

1. 将 `root` 入栈
2. 循环执行以下操作，直到栈为空
   - 弹出栈顶节点 `top`，进行访问
   - 将 `top.right` 入栈
   - 将 `top.left` 入栈



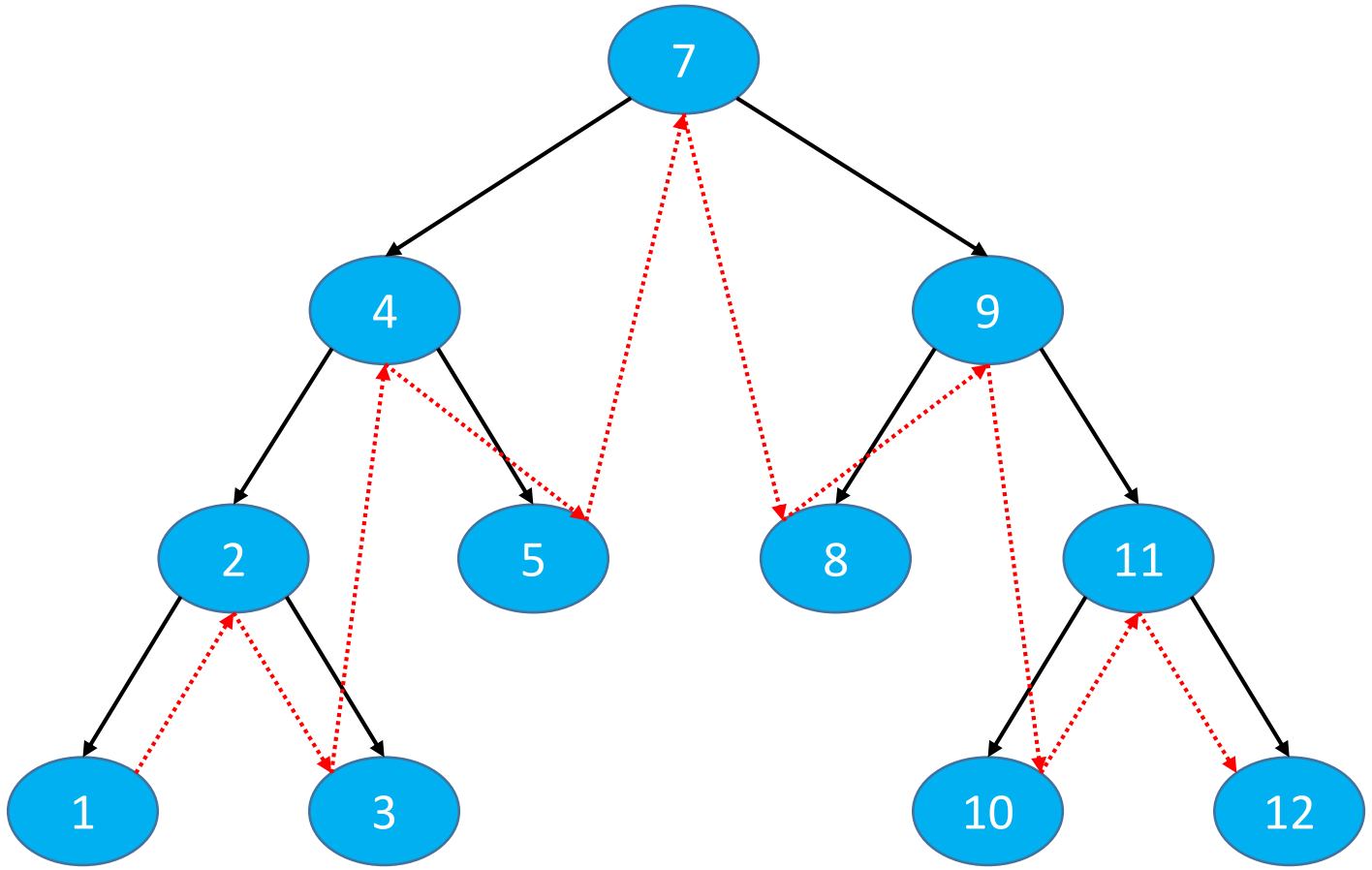


中序遍历（Inorder Traversal）

- 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



中序遍历 – 非递归

■ 利用栈实现

1. 设置 node = root
2. 循环执行以下操作

□ 如果 node != null

✓ 将 node 入栈

✓ 设置 node = node.left

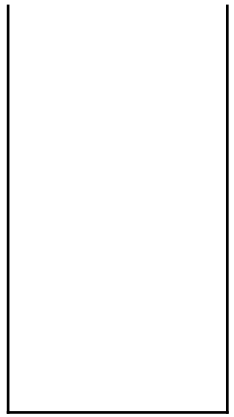
□ 如果 node == null

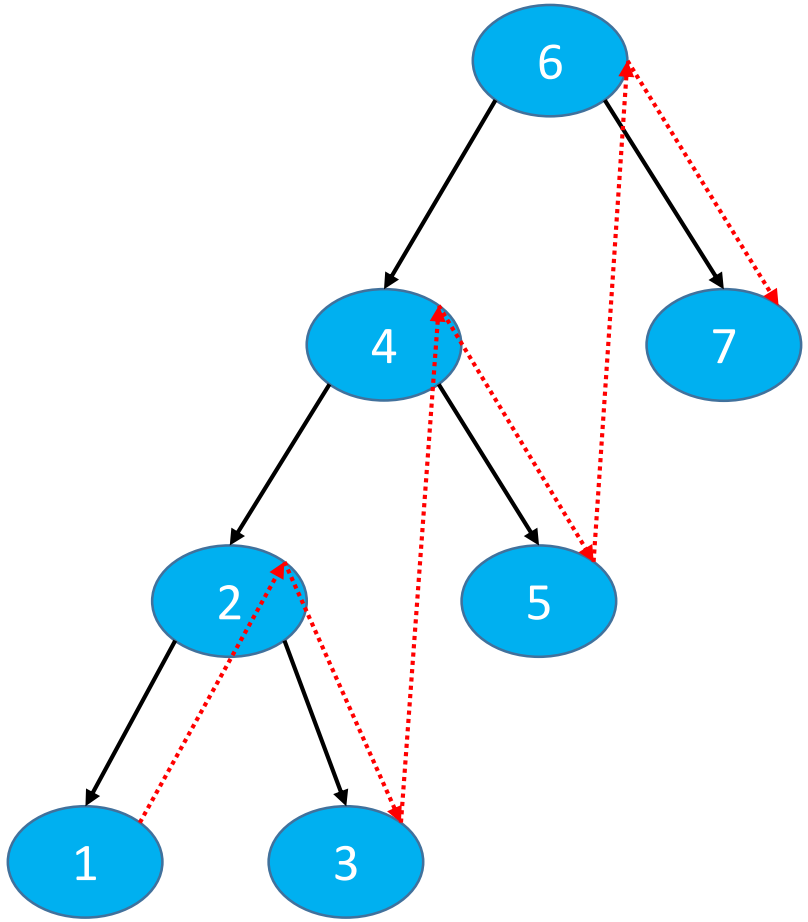
✓ 如果栈为空，结束遍历

✓ 如果栈不为空，弹出栈顶元素并赋值给 node

➢ 对 node 进行访问

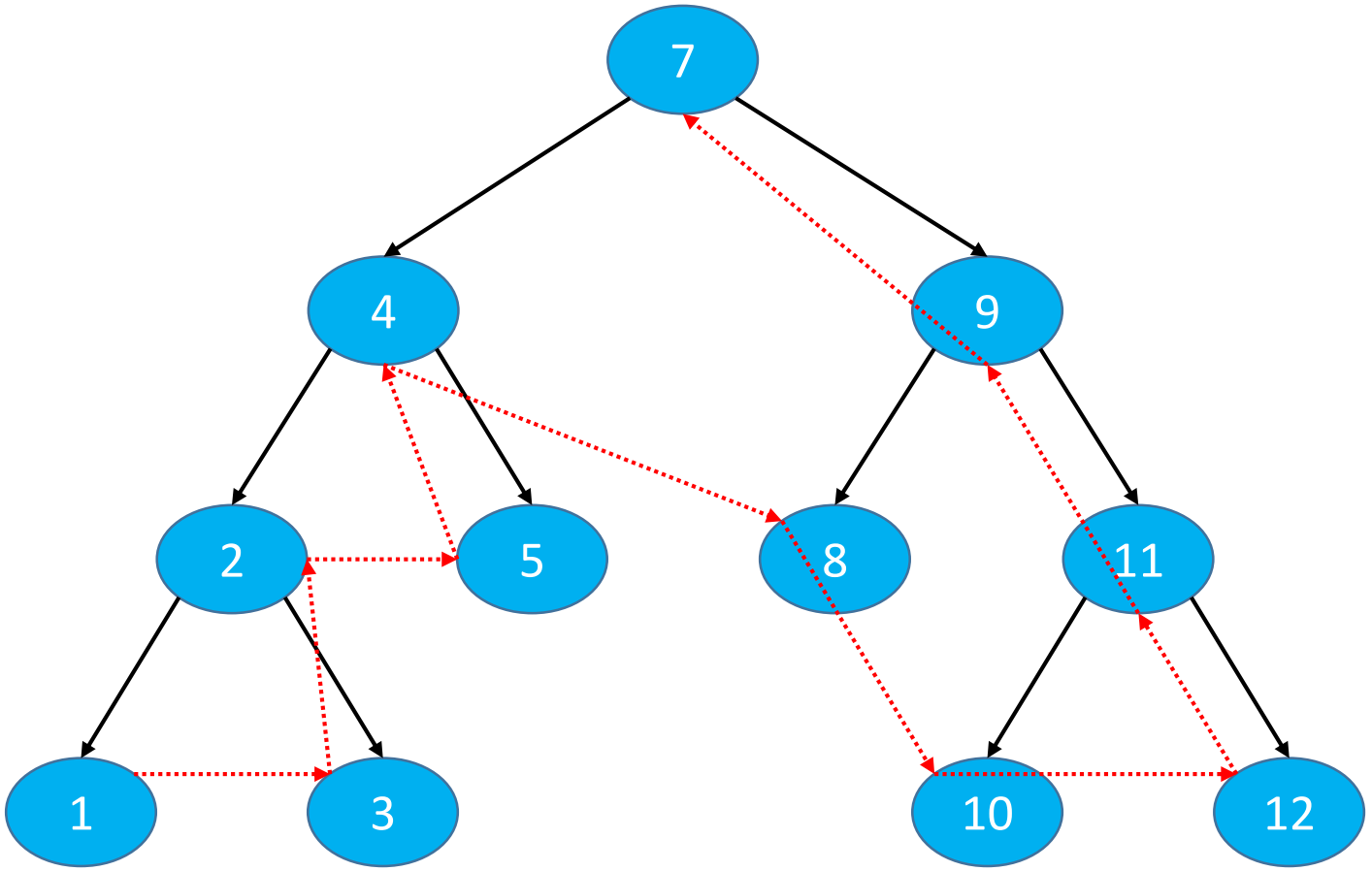
➢ 设置 node = node.right





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

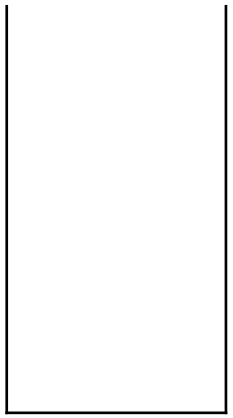


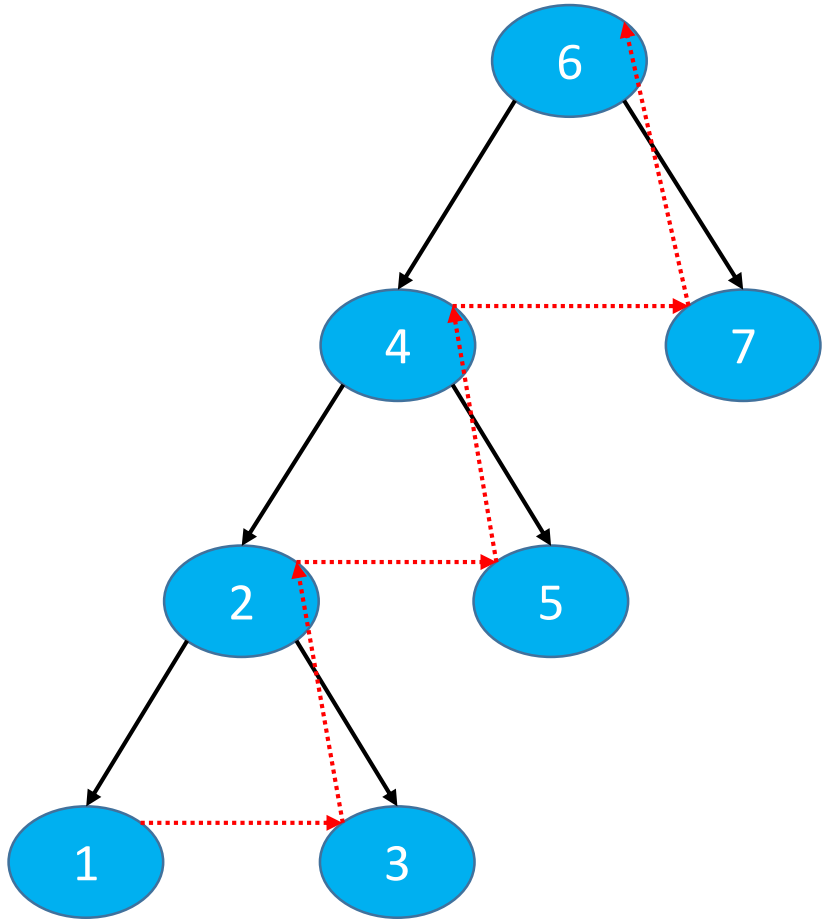
后序遍历 – 非递归

- 利用栈实现

1. 将 `root` 入栈
2. 循环执行以下操作，直到栈为空

- 如果栈顶节点是叶子节点 或者 上一次访问的节点是栈顶节点的子节点
  - ✓ 弹出栈顶节点，进行访问
- 否则
  - ✓ 将栈顶节点的`right`、`left`按顺序入栈



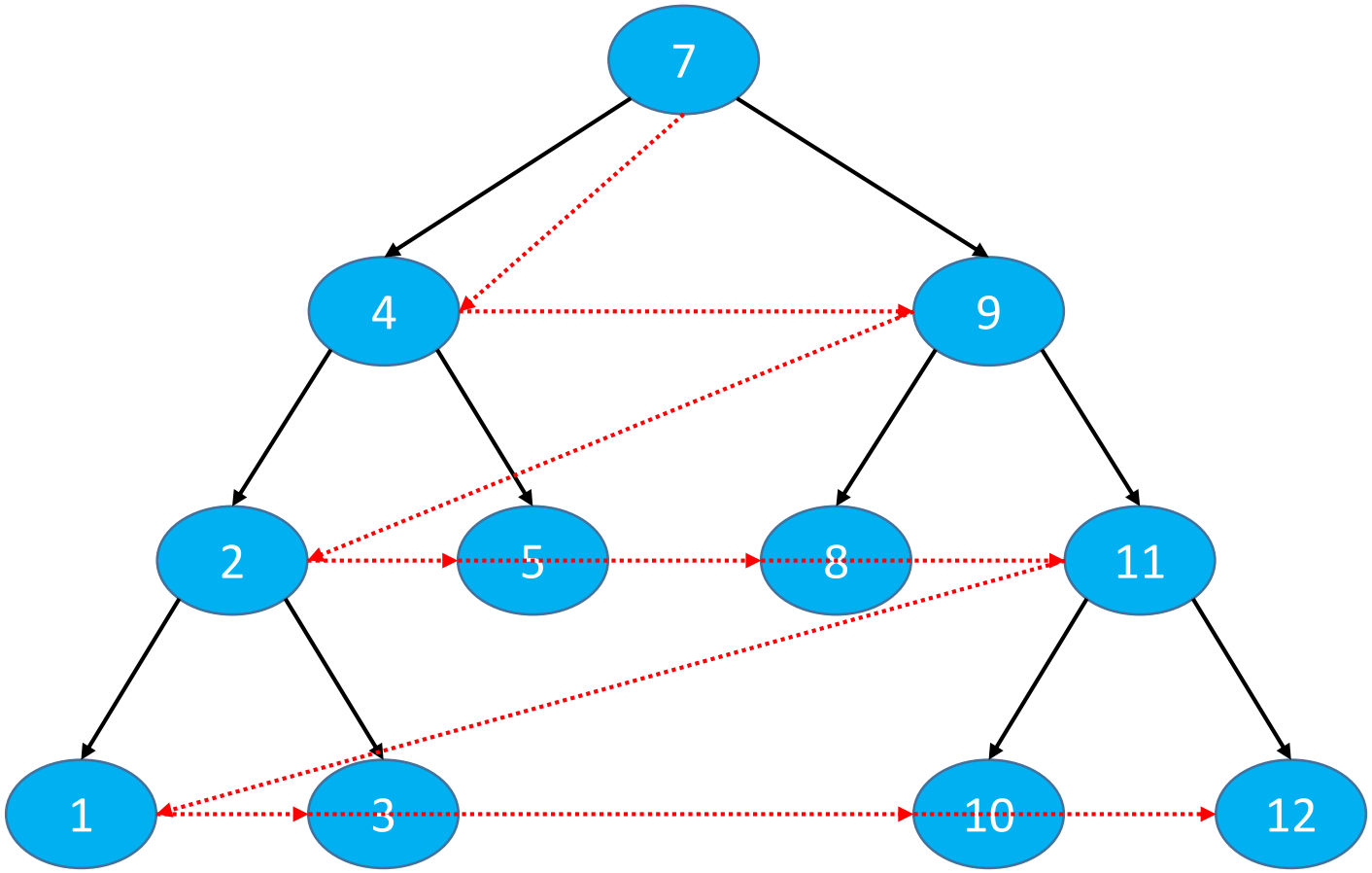


层序遍历（Level Order Traversal）

- 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



四则运算

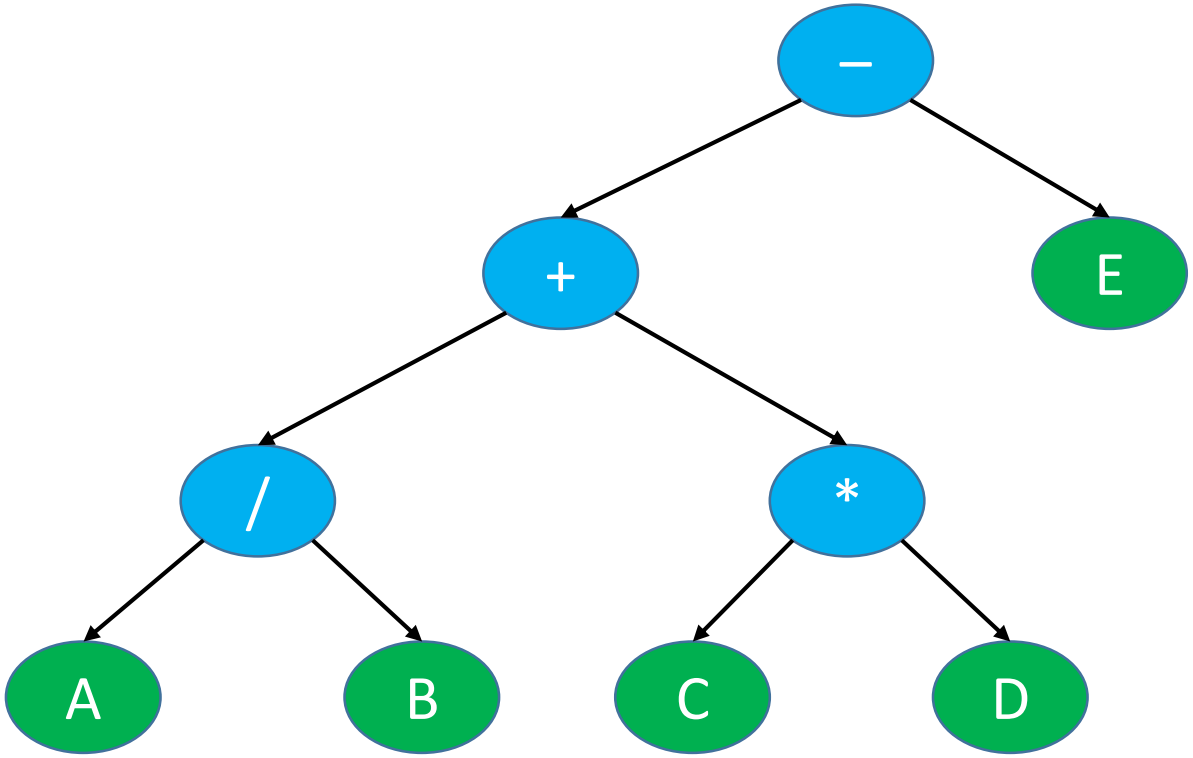
- 四则运算的表达式可以分为3种
  - 前缀表达式（prefix expression），又称为波兰表达式
  - 中缀表达式（infix expression）
  - 后缀表达式（postfix expression），又称为逆波兰表达式

前缀表达式	中缀表达式	后缀表达式
+ 1 2	1 + 2	1 2 +
+ 2 * 3 4	2 + 3 * 4	2 3 4 * +
+ 9 * – 4 1 2	9 + (4 – 1) * 2	9 4 1 – 2 * +

表达式树

- 如果将表达式的操作数作为叶子节点，运算符作为父节点（假设只是四则运算）
  - 这些节点刚好可以组成一棵二叉树
  - 比如表达式：A / B + C * D – E

- 如果对这棵二叉树进行遍历
  - 前序遍历
    - ✓ – + / A B * C D E
    - ✓ 刚好就是前缀表达式（波兰表达式）
  - 中序遍历
    - ✓ A / B + C * D – E
    - ✓ 刚好就是中缀表达式（波兰表达式）
  - 后序遍历
    - ✓ A B / C D * + E –
    - ✓ 刚好就是后缀表达式（逆波兰表达式）



思考

- 如果允许外界遍历二叉树的元素？你会如何设计接口？

```
public void inorder(Visitor<E> visitor) {
    if (visitor == null) return;
    inorder(visitor, root);
}

private void inorder(Visitor<E> visitor, Node<E> node) {
    if (node == null) return;
    inorder(visitor, node.left);
    visitor.visit(node.element);
    inorder(visitor, node.right);
}

public static interface Visitor<E> {
    void visit(E element);
}
```

```
bst.inorder(new Visitor<Integer>() {
    public void visit(Integer element) {
        System.out.println(element);
    }
});
```

增强遍历接口

```
public void inorder(Visitor<E> visitor) {
    if (visitor == null) return;
    inorder(root, visitor);
}

private void inorder(Node<E> node, Visitor<E> visitor) {
    if (node == null || visitor.stop) return;

    inorder(node.left, visitor);
    if (visitor.stop) return;
    visitor.stop = visitor.visit(node.element);
    inorder(node.right, visitor);
}
```

```
public static abstract class Visitor<E> {
    boolean stop;
    /**
     * @return 如果返回true，就代表停止遍历
     */
    public abstract boolean visit(E element);
}
```

```
bst.inorder(new Visitor<Integer>() {
    public boolean visit(Integer element) {
        System.out.print(element + " ");
        return element == 4 ? true : false;
    }
});
```

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

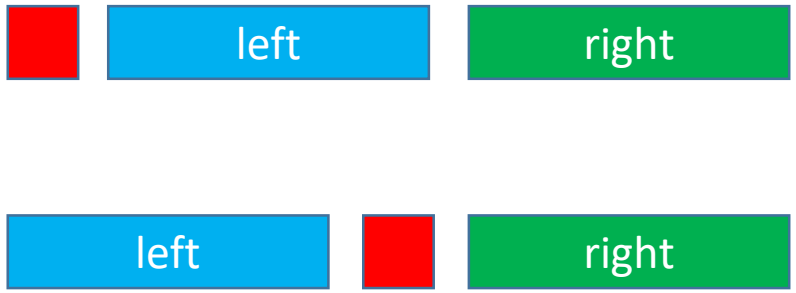
根据遍历结果重构二叉树

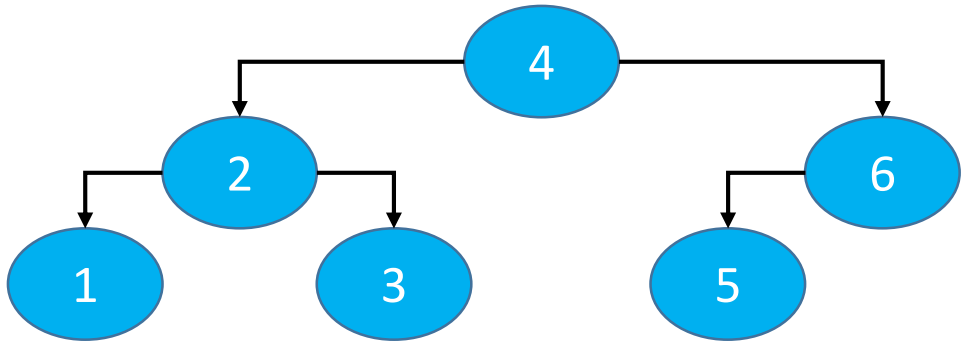
- 以下结果可以保证重构出唯一的一棵二叉树
  - 前序遍历 + 中序遍历
  - 后序遍历 + 中序遍历

- 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

前序遍历+中序遍历重构二叉树

- 前序遍历：4 2 1 3 6 5
- 中序遍历：1 2 3 4 5 6





练习 – 利用前序遍历树状打印二叉树

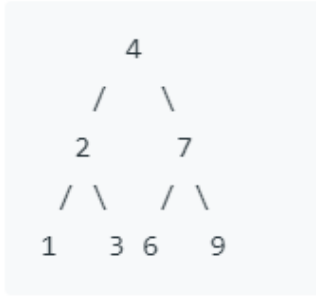
```
【4】
〖L〗【2】
〖L〗〖L〗【1】
〖L〗〖R〗【3】
〖R〗【6】
〖R〗〖L〗【5】
〖R〗〖R〗【7】
```

```java
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    toString(sb, root, "");
    return sb.toString();
}

private void toString(StringBuilder sb, Node<E> node, String prefix) {
    if (node == null) return;

    sb.append(prefix).append("【")
    .append(node.element).append("】").append("\n");

    toString(sb, node.left, prefix + "〖L〗");
    toString(sb, node.right, prefix + "〖R〗");
}
```

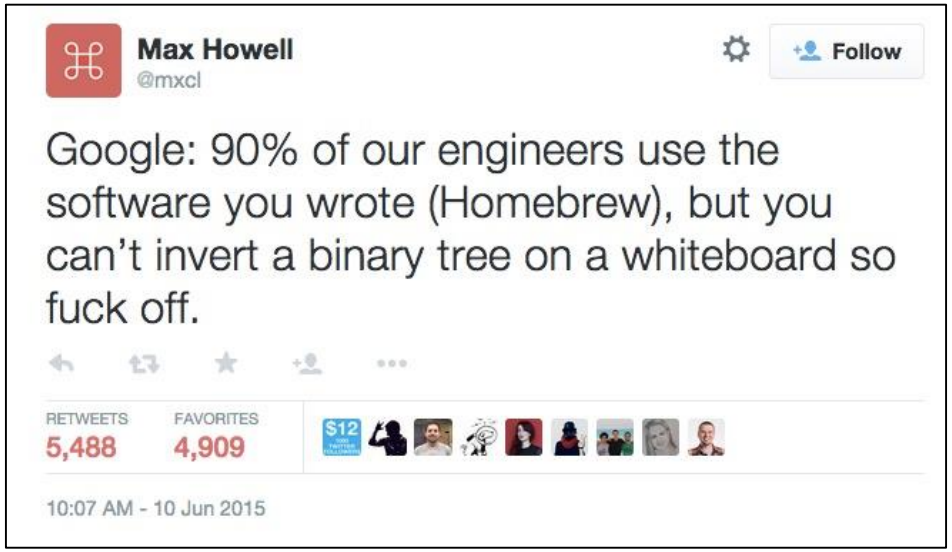
练习 - 翻转二叉树

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

输入：



输出：



- 请分别用递归、迭代（非递归）方式实现

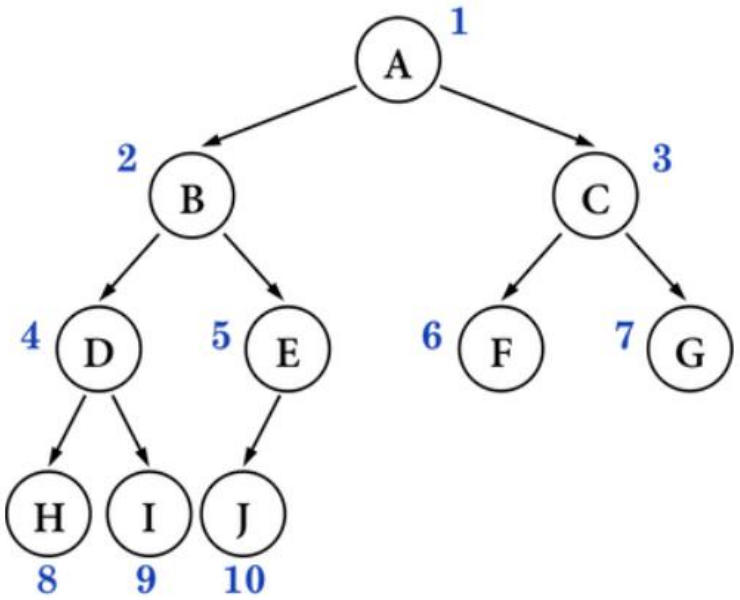
练习 – 计算二叉树的高度

- 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`，将 `node.left` 入队
  - 如果 `node.left==null && node.right!=null`，返回 `false`
  - 如果 `node.right!=null`，将 `node.right` 入队
  - 如果 `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`



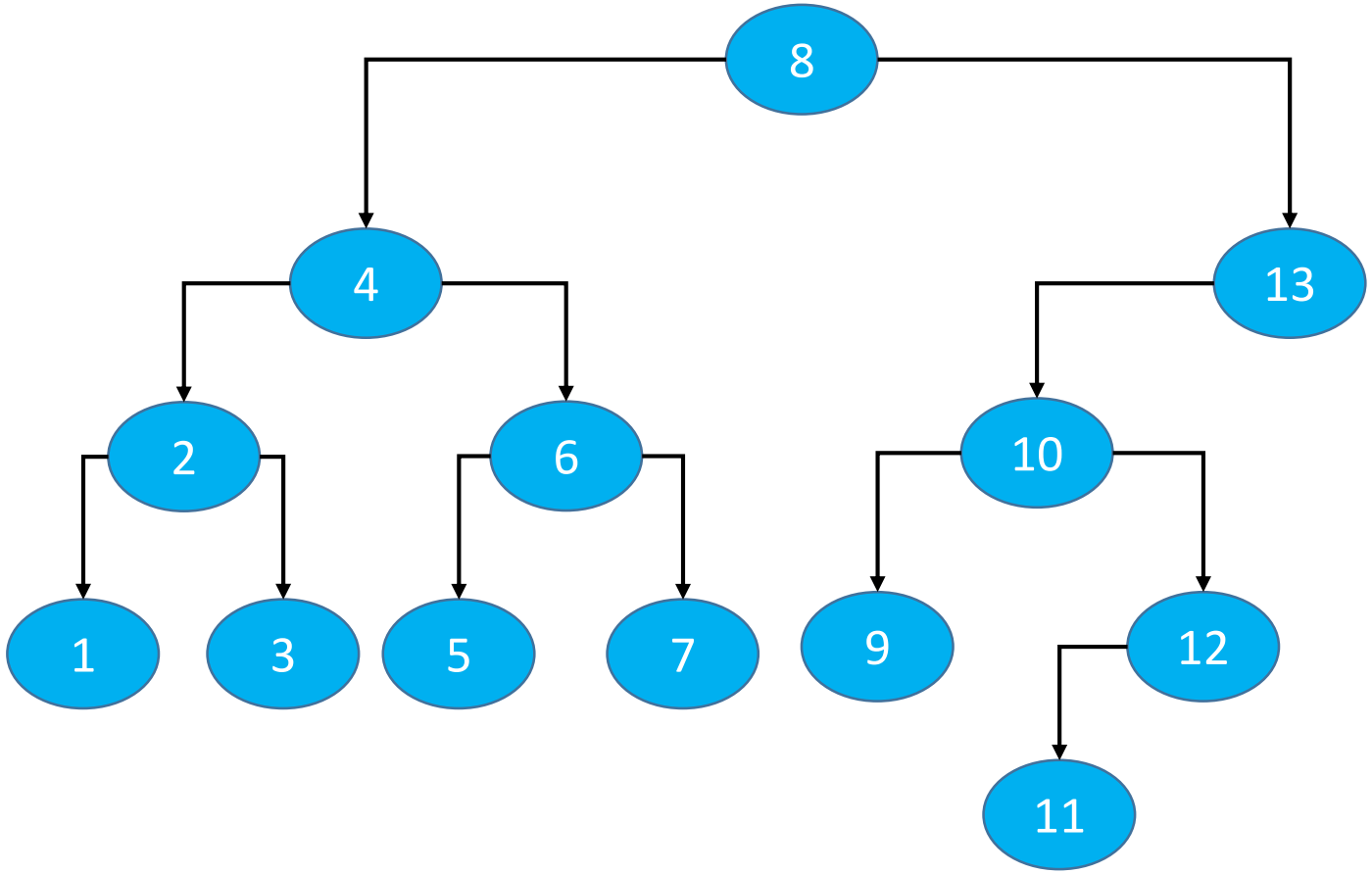
前驱节点（predecessor）

- 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

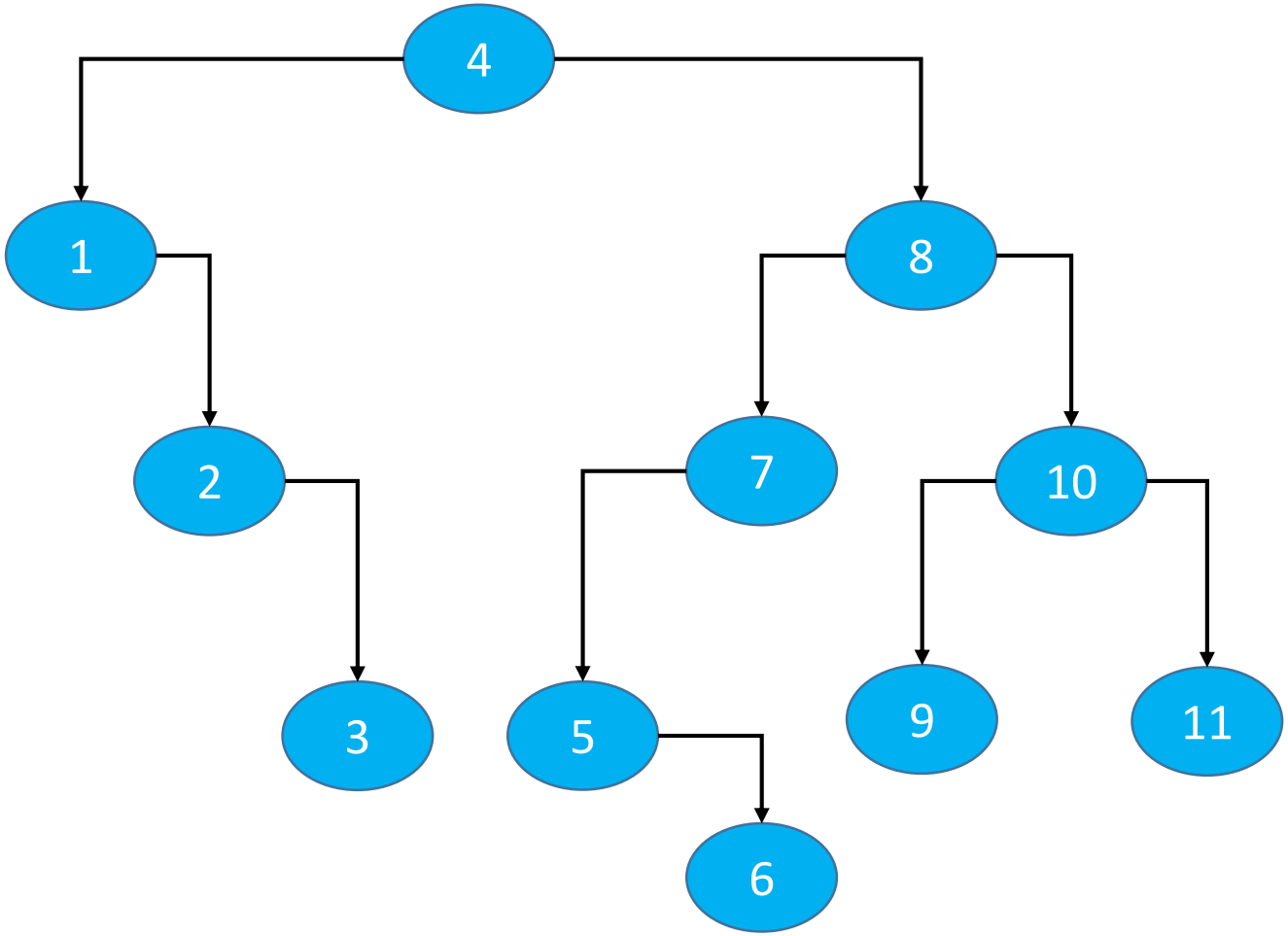
后继节点（successor）

- 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

作业

- 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）

作业

- 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/

作业

- 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/

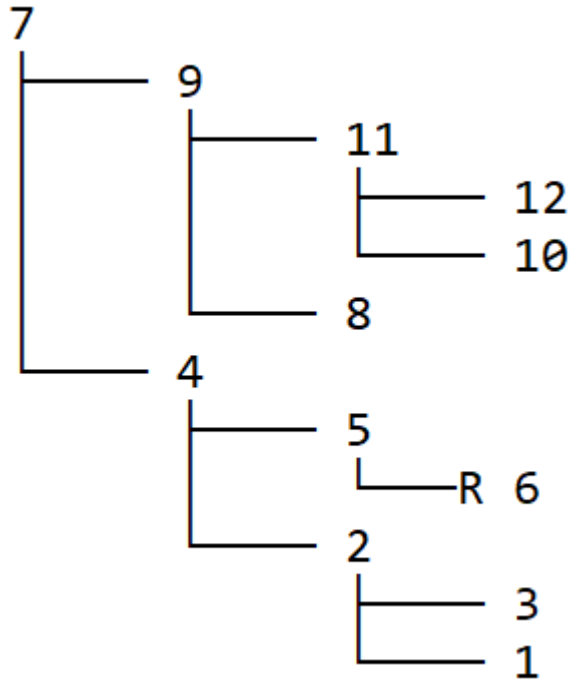
- 从前序与中序遍历序列构造二叉树
  - https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/

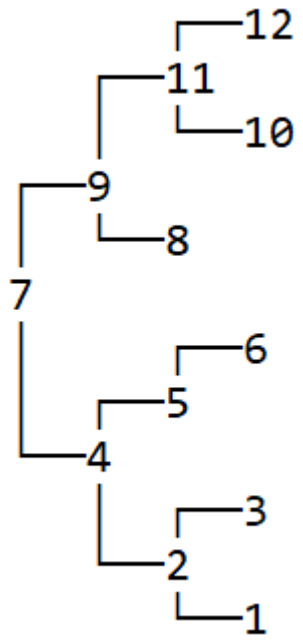
- 根据前序和后序遍历构造二叉树
  - https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-postorder-traversal/

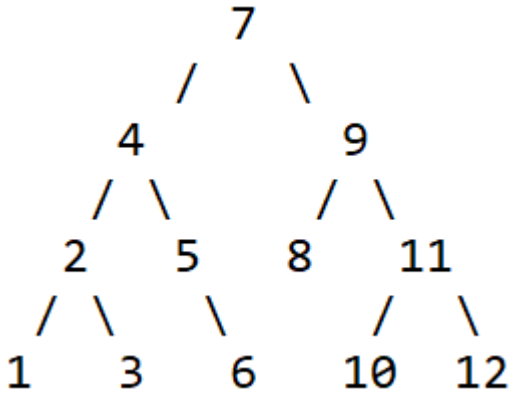
- 对称二叉树
  - https://leetcode-cn.com/problems/symmetric-tree/

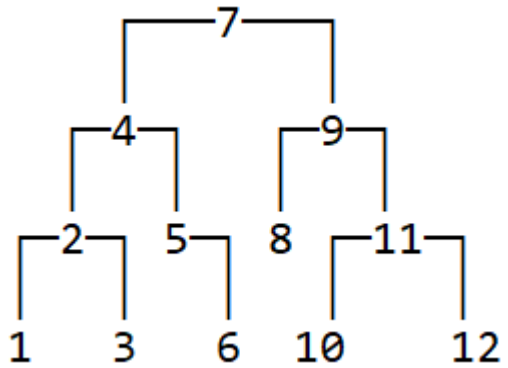
作业

- 树状形式打印二叉树
- 比如给定一个二叉搜索树：[7, 4, 9, 2, 5, 8, 11, 1, 3, 6, 10, 12]
- 尝试输出以下格式









- 开源项目：https://github.com/CoderMJLee/BinaryTrees

作业

- 已知前序、中序遍历结果，求出后序遍历结果
- 已经中序、后序遍历结果，求出前序遍历结果