原 java初、中、高级面试题必备——数据结构与算法—二叉树

2019年06月26日 17:48:26 在IT中穿梭旅行 阅读数 39 编辑

181. 什么是树结构

树(Tree)结构是一种**描述非线性层次关系的数据结构**，其中重要的是树的概念。树是" n个数据结点的集合，在该集合中包含一个根结点，根结点之下分布着一些互不交叉的子集合， 这些子集合是根结点的子树。树结构的基本特征如下：

1. 在一个树结构中，有且仅有一个结点没有直接前驱，这个结点就是树的根结点；
2. 除根结点外，其余每个结点有且仅有一个直接前驱；
3. 每个结点可以有任意多个直接后继。
4. 树里的每一个节点有一个根植和一个包含所有子节点的列表。从图的观点来看，树也可视为一个拥有**N 个节点**和**N-1** 条边的一个有向无环图。

典型的树结构，如图 2-13 所示，可以直观地看到其类似于现实中树的根系，越往下层根系分支越多。图中 A 便是树的根结点，根结点 A 有三个直接后继结点 B、C 和 D，而结点 C 只有一个直接前驱结点 A。

另外，一个树结构也可以是空，此时空树中没有数据结点，也就是一个空集合。如果树结构中仅包含一个结点，那么这也是一个树，树根便是该结点自身。

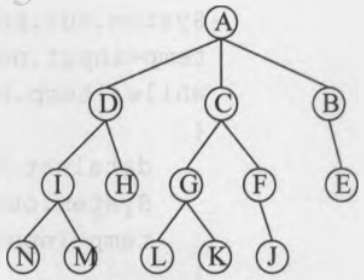


图 2-13 典型的树结构

从树的定义可以看出，树具有层次结构的性质。而从数学的角度来看，树具有递归的特性。在树中的每个结点及其之后的所有结点构成一个子树，这个子树也包括根结点。

182.树的基本概念

对于读者来说，树是一种全新的数据结构，其中包含了许多新的概念。 ·

1. **父结点和子结点**：每个结点子树的根称为该结点的子结点，相应的，该结点称为其子结点的父结点。
2. **兄弟结点**：具有同一父结点的结点称为兄弟结点。
3. **结点的度**：一个结点所包含子树的数量。
4. **树的度**：是指该树所有结点中最大的度。
5. **叶结点**：树中度为零的结点称为叶结点或终端结点。
6. **分支结点**：树中度不为零的结点称为分支结点或非终端结点。 ·
7. **结点的层数**：结点的层数从树根开始计算，根结点为第1层、依次向下为第2、3、……n 层（树是一种层次结构，每个结点都处在一定的层次上）。
8. **树的深度**：树中结点的最大层数称为树的深度。 ·
9. **有序树**：若树中各结点的子树（兄弟结点）是按一定次序从左向右排列的，称为有序树。 ·
10. **无序树**：若树中各结点的子树（兄弟结点）未按一定次序排列，称为无序树。
11. **森林（forest）**： n （«>0）棵互不相交的树的集合

下面从一个例子来分析上述树结构的基本概念。图 2-14 所示为一个基本的树结构。其中，结点 A 为根结点。结点 A 有 3 个子树，因此，结点 A 的度为 3。同理，结点 E 有两个子树，结点 E 的度为 2。所有结点中，结点 A 的度为 3 最大，因此整个树的度为 3。结点 E 是结点 K 和结点 L 的父结点，结点 K 和结点 L 是结点 E 的子结点，结点 K 和结点 L 为兄弟结点。

在这个树结构中，结点 G、结点 H、结点 K、结点 J、结点 N、结点 O、结点 P 和结点 Q 都是叶结点。其余的都是分支结点，整个树的深度为 4。除去根结点 A，留下的子树就构成了一个森林。

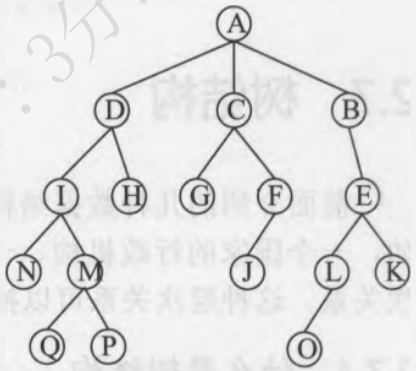


图 2-14 基本的树结构

183.二叉树

二叉树是一种更为典型的树状结构，它是n个结点的集合，每个结点最多只能有两个子结点。二叉树的子树仍然是二叉树。二叉树的一个结点上对应的两个子树分别称为左子树和右子树。由于子树有左右之分，因此二叉树是有序树。

二叉树还可以进一步细分为两种特殊的类型，

1. **满二叉树**
2. **完全二叉树。**

满二叉树即在二叉树中除最下一层的叶结点外，每层的结点都有两个子结点。典型的满二叉树，如图2-16所示

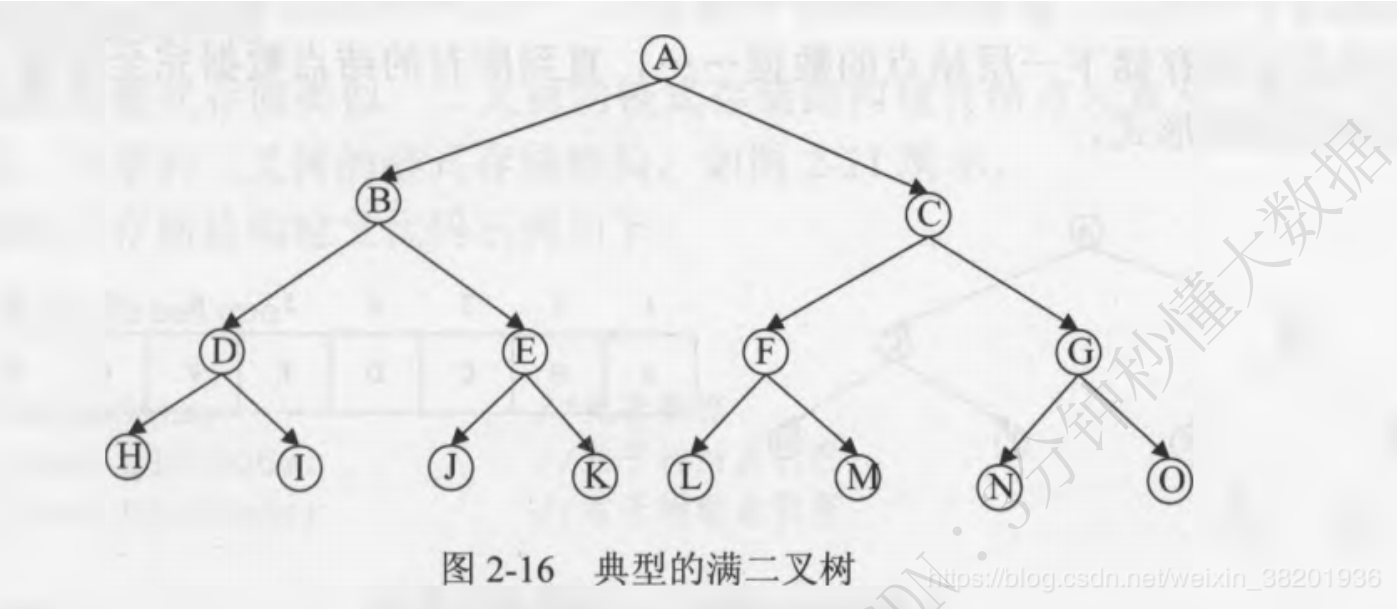


图 2-16　典型的满二叉树

完全二叉树即在二叉树中除二叉树最后一层外，其他各层的结点数都达到最大个数，且最后一层叶结点按照从左向右的顺序连续存在，只缺最后一层右侧若干结点。典型的完全二叉树，如图2-17所

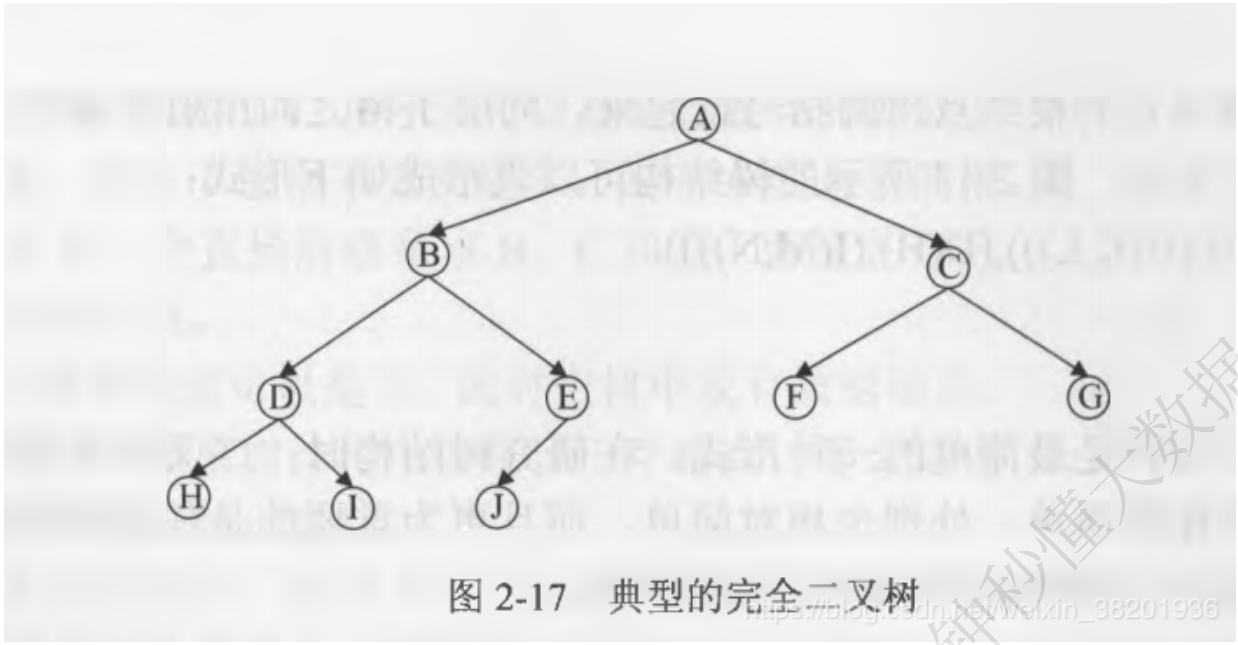


图 2-17　典型的完全二叉树

2．完全二叉树的性质

二叉树中树结构研究的重点是完全二叉树。对于完全二叉树，若树中包含 n 个结点，假设这些结点按照顺序方式存储。那么，对于任意一个结点 m 来说，具有如下性质：

- 如果 m != 1，则结点 m 的父结点的编号为 $m/2$；
- 如果 $2*m \leq n$，则结点 m 的左子树根结点的编号为 $2*m$；若 $2*m>n$，则无左子树，也没有右子树；
- 如果 $2*m+1 \leq n$，则结点 m 的右子树根结点编号为 $2*m+1$；若 $2*m+1>n$，则无右子树。

另外，对于该完全二叉树来说，其深度为$[\log 2n]+1$。

这些基本性质展示了完全二叉树结构的特点，在完全二叉树的存储方式及运算处理上都有重要意义。

完全二叉树的公式：n1是度为1的结点总数，n2为度为2的结点总数，n0为度为0的结点总数。由二叉树的性质可知

n0=n2+1，则n=n1+n2+n0（其中n为完全二叉树的结点总数），由于完全二叉树中度为1的结点数为0或者1,所以得到

n0 =(n)/2 或者n0 =(n+1)/2;

树结构可以分为：

1. **顺序存储结构**
2. **链式存储结构**

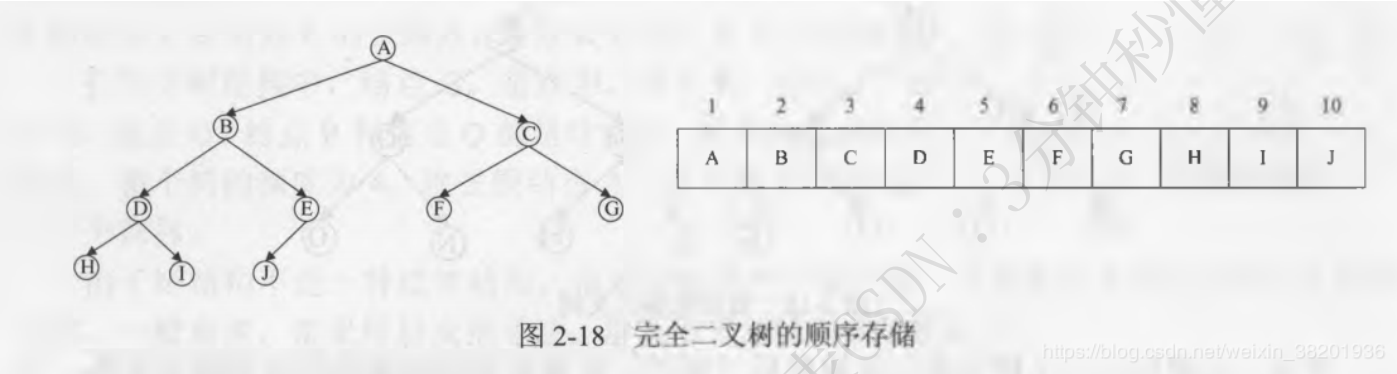


图 2-18 完全二叉树的顺序存储

184.二叉树的顺序存储

顺序存储方式是最基本的数据存储方式。与线性表类似，树结构的顺序存储一般采用一维结构数组来表示。这里的关键是定义合适的次序来存放树中各个层次的数据。

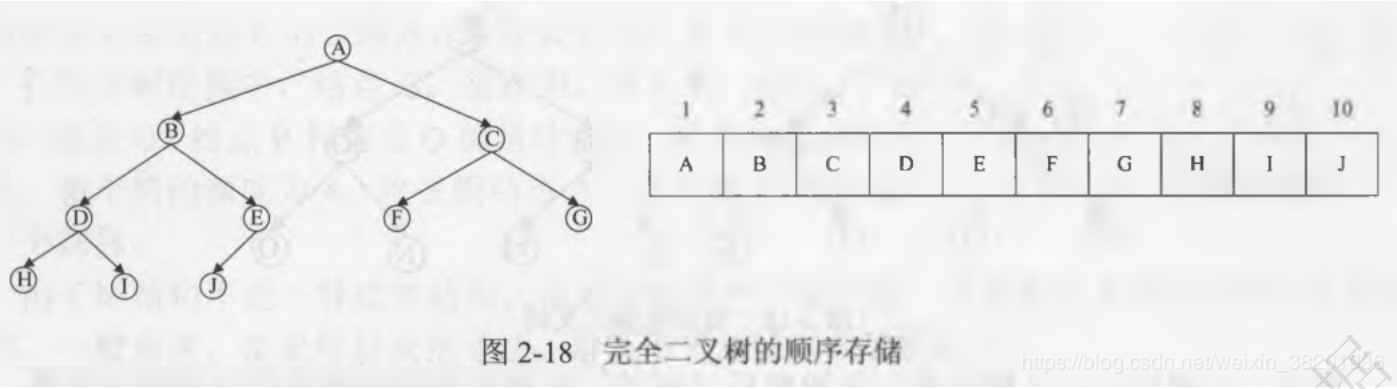


图 2-18 完全二叉树的顺序存储

可以根据前面介绍的完全二叉树的性质来推算各个结点之间的位置关系。

例如： •对于结点D,其位于数组的第4个位置，则其父结点的编号为4/2=2,即结点B

•结点D左子结点的编号为2*4=8,即结点H。

•结点D右子结点的编号为2*4+1=9,即结点I

185.二叉树的链式存储

与线性结构的链式存储类似，二叉树的链式存储结构包含结点元素及分别指向左子树和右子树的引用。典型的二叉树的链式存储结构，如图2-21所示

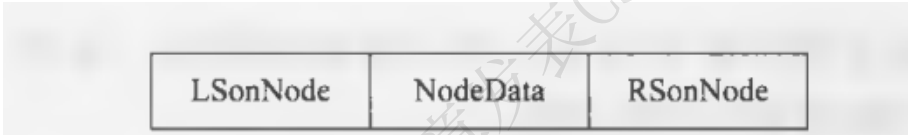


图 2-21 典型的二叉树的链式存储结构

186 遍历二叉树

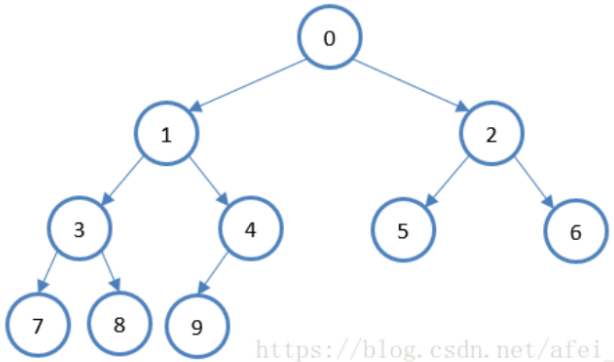
遍历整个二叉树：

1. **·先序遍历：即先访问根结点，再按先序遍历左子树，最后按先序遍历右子树。先序遍历一般也称为先根次序遍历，简称为DLR遍历。**
2. **·中序遍历：即先按中序遍历左子树，再访问根结点，最后按中序遍历右子树。中序遍历一般也称为中根次序遍历，简称为LDR遍历。**

3. **·后序遍历：即先按后序遍历左子树，再按后序遍历右子树，最后访问根结点。后序遍历一般也称为后根次序遍历，简称为LRD遍历。**

- 先序遍历：DLR（D 在最前面）
- 中序遍历：LDR（D 在中间）
- 后序遍历：LRD（D 在最后面）

二、二叉树举例



针对以上这个二叉树，3 种遍历结果为：

先序遍历：0 1 3 7 8 4 9 2 5 6

中序遍历：7 3 8 1 9 4 0 5 2 6

后序遍历：7 8 3 9 4 1 5 6 2 0

1. 能够运用**递归**方法解决树的为前序遍历、中序遍历和后序遍历问题
2. 能用运用**迭代**方法解决树的为前序遍历、中序遍历和后序遍历问题
3. 能用运用**广度优先搜索**解决树的**层序遍历**问题

187.二叉树的前序遍历:

给定一个二叉树，返回它的 *前序* 遍历。

示例:

```
输入: [1,null,2,3]
   1
    \
     2
    /
   3

输出: [1,2,3]
```

进阶: 递归算法很简单，你可以通过迭代算法完成吗?


```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
         if(root==null){
            return new ArrayList<>();
        }else{
            List<Integer> list = new ArrayList<>();
            Stack<TreeNode> stack = new Stack<>();
            stack.push(root);
            while (!stack.isEmpty()){
                TreeNode tmp = stack.pop();
                list.add(tmp.val);
```

```
                if(tmp.right!=null)
                    stack.push(tmp.right);
                if(tmp.left!=null)
                    stack.push(tmp.left);
            }
            return list;
        }
    }
}
```

188.二叉树的中序遍历:

给定一个二叉树，返回它的*中序*遍历。

示例:

```
输入: [1,null,2,3]
   1
    \
     2
    /
   3

输出: [1,3,2]
```

进阶: 递归算法很简单，你可以通过迭代算法完成吗?

解题思路:

- 遇到一个结点，就把它压入栈，并去遍历它的左子树
- 当左子树遍历结束后，从栈顶弹出这个结点并访问它
- 然后按其右指针再去中序遍历该节点的右子树

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
         if(root==null){
            return new ArrayList<>();
        }else{
            List<Integer> list = new ArrayList<>();
            Stack<TreeNode> stack = new Stack<>();
            stack.push(root);
            while (!stack.isEmpty()){
                TreeNode tmp = stack.pop();
                list.add(tmp.val);
                if(tmp.right!=null)
                    stack.push(tmp.right);
                if(tmp.left!=null)
                    stack.push(tmp.left);
            }
            return list;
        }
    }
}
```


189.二叉树的后序遍历

给定一个二叉树，返回它的 *后序* 遍历。

示例:

```
输入: [1,null,2,3]
   1
    \
     2
    /
   3

输出: [3,2,1]
```

进阶: 递归算法很简单，你可以通过迭代算法完成吗?


```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
         if(root == null){
            return new ArrayList<>();
        }
        else{
            List<Integer> list = new ArrayList<>();
            Stack<TreeNode> stack = new Stack<>();
            stack.push(root);
            while (!stack.isEmpty()){
```

```
                TreeNode tmp = stack.pop();
                list.add(tmp.val);
                if(tmp.left!=null){
                    stack.push(tmp.left);
                }
                if(tmp.right!=null){
                    stack.push(tmp.right);
                }
            }
            Collections.reverse(list);
            return list;
        }

    }
}
```

190.层序遍历 - 介绍

层序遍历就是逐层遍历树结构。

广度优先搜索是一种广泛运用在树或图这类数据结构中，遍历或搜索的算法。 该算法从一个根节点开始，首先访问节点本身。 然后遍历它的相邻节点，其次遍历它的二级邻节点、三级邻节点，以此类推。

当我们在树中进行广度优先搜索时，我们访问的节点的顺序是按照层序遍历顺序的。

通常，我们使用一个叫做**队列的数据结构**来帮助我们做广度优先搜索。

191.二叉树的深度

给定一个二叉树，找出其最大深度。

二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

说明: 叶子节点是指没有子节点的节点。

示例:
给定二叉树 `[3,9,20,null,null,15,7]`,

返回它的最大深度 3。


```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public int maxDepth(TreeNode root) {
        if(root==null){
            return 0;
        }else{
            int left_dep =maxDepth(root.left);
            int right_dep =maxDepth(root.right);
            return Math.max(left_dep,right_dep)+1;
        }
    }
}
```


下一章 队列和栈 链接:

人脸识别主要算法原理

人脸识别