

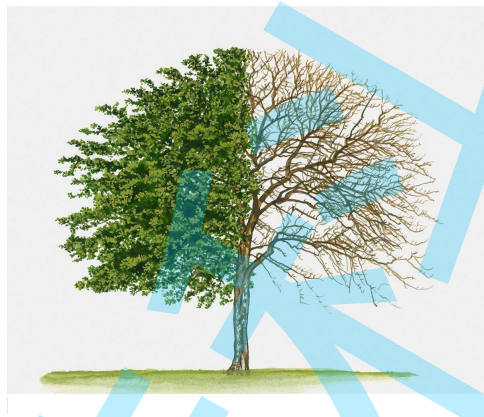
# 二叉树

## 本节目标

- 掌握二叉树数据结构的概念和基本实现
- 掌握二叉树前中后序的递归写法
- 掌握二叉树层序的写法
- 学习二叉树的前中后序的非递归写法
- 完成二叉树相关的面试题练习

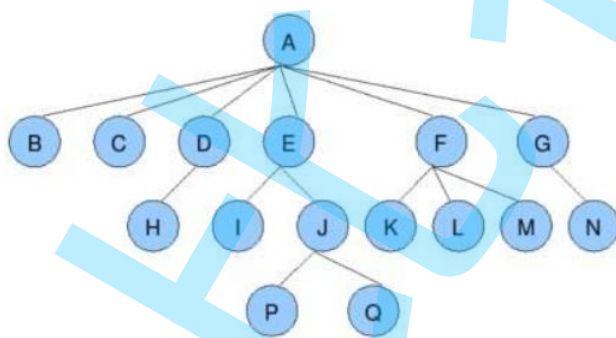
## 1. 树型结构（了解）

### 1.1 概念

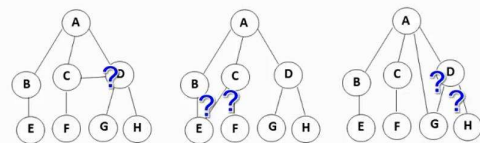


树是一种**非线性**的数据结构，它是由 $n$  ( $n \geq 0$ ) 个有限结点组成一个具有层次关系的集合。把它叫做树是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。它具有以下的特点：

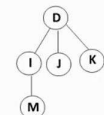
- 有一个特殊的节点，称为根节点，根节点没有前驱节点
- 除根节点外，其余节点被分成 $M$  ( $M > 0$ ) 个互不相交的集合 $T_1$ 、 $T_2$ 、.....、 $T_m$ ，其中每一个集合 $T_i$  ( $1 \leq i \leq m$ ) 又是一棵与树类似的子树。每棵子树的根节点有且只有一个前驱，可以有0个或多个后继
- 树是递归定义的。



#### ❖ 树与非树？



- 子树是**不相交**的；
- 除了根结点外，每个结点有且仅有一个父结点；
- 一棵 $N$ 个结点的树有 $N-1$ 条边。



### 1.2 概念（重要）

**节点的度**：一个节点含有的子树的个数称为该节点的度；如上图：A的为6

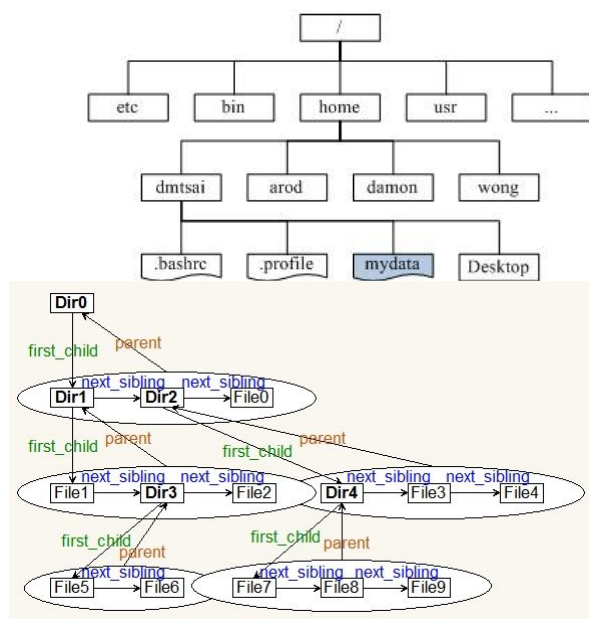
**树的度**：一棵树中，最大的节点的度称为树的度；如上图：树的度为6

**叶子节点或终端节点**：度为0的节点称为叶节点；如上图：B、C、H、I...等节点为叶节点



## 1.4 树的应用

文件系统管理（目录和文件）



## 2. 二叉树（重点）



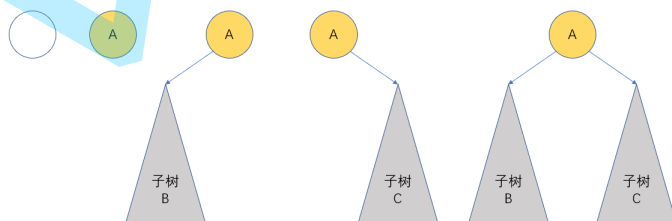
### 2.1 概念

一棵二叉树是结点的一个有限集合，该集合或者为空，或者是由一个根结点加上两棵别称为左子树和右子树的二叉树组成。

二叉树的特点：

1. 每个结点最多有两棵子树，即二叉树不存在度大于 2 的结点。
2. 二叉树的子树有左右之分，其子树的次序不能颠倒，因此二叉树是有序树。

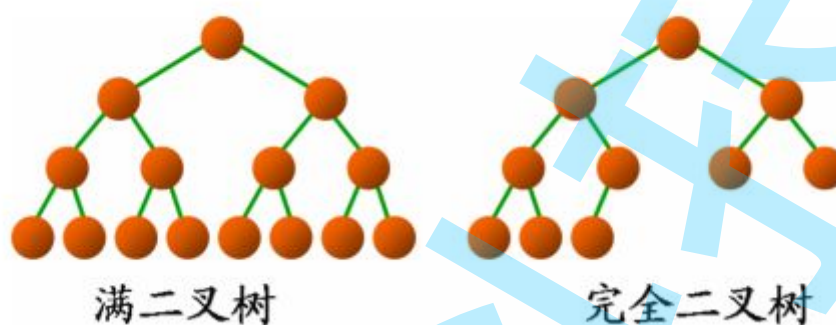
### 2.2 二叉树的基本形态



上图给出了几种特殊的二叉树形态，从左往右依次是：空树、只有根节点的二叉树、节点只有左子树、节点只有右子树、节点的左右子树均存在，一般二叉树都是由上述基本形态结合而形成的。

## 2.3 两种特殊的二叉树

1. **满二叉树**: 一个二叉树，如果每一个层的结点数都达到最大值，则这个二叉树就是满二叉树。也就是说，如果一个二叉树的层数为 $K$ ，且结点总数是 $2^k-1$ ，则它就是满二叉树。
2. **完全二叉树**: 完全二叉树是效率很高的数据结构，完全二叉树是由满二叉树而引出来的。对于深度为 $K$ 的，有 $n$ 个结点的二叉树，当且仅当其每一个结点都与深度为 $K$ 的满二叉树中编号从1至 $n$ 的结点一一对应时称之为完全二叉树。要注意的是满二叉树是一种特殊的完全二叉树。



## 2.4 二叉树的性质

1. 若规定根节点的层数为1，则一棵非空二叉树的第 $i$ 层上最多有 $2^{i-1}$  ( $i \geq 1$ )个结点
2. 若规定只有根节点的二叉树的深度为1，则深度为 $K$ 的二叉树的最大结点数是 $2^k-1$  ( $k \geq 1$ )
3. 对任何一棵二叉树，如果其叶结点个数为 $n_0$ ，度为2的非叶结点个数为 $n_2$ ，则有 $n_0 = n_2 + 1$
4. 具有 $n$ 个结点的完全二叉树的深度 $k$ 为 $\lceil \log_2(n+1) \rceil$ 上取整
5. 对于具有 $n$ 个结点的完全二叉树，如果按照从上至下从左至右的顺序对所有结点从0开始编号，则对于序号为 $i$ 的结点有：
  - 若 $i > 0$ ，双亲序号： $(i-1)/2$ ； $i=0$ ， $i$ 为根节点编号，无双亲节点
  - 若 $2i+1 < n$ ，左孩子序号： $2i+1$ ，否则无左孩子
  - 若 $2i+2 < n$ ，右孩子序号： $2i+2$ ，否则无右孩子

比如：假设一棵完全二叉树中总共有1000个结点，则该二叉树中\_\_\_\_个叶子结点，\_\_\_\_个非叶子结点，\_\_\_\_个结点只有左孩子，\_\_\_\_个只有右孩子。

## 2.5 二叉树的存储

二叉树的存储结构分为：顺序存储和类似于链表的链式存储。

顺序存储在下节介绍。

二叉树的链式存储是通过一个一个的结点引用起来的，常见的表示方式有二叉和三叉表示方式，具体如下：

```
1 // 孩子表示法
2 class Node {
3     int val; // 数据域
4     Node left; // 左孩子的引用，常常代表左孩子为根的整棵左子树
5     Node right; // 右孩子的引用，常常代表右孩子为根的整棵右子树
6 }
7
8 // 孩子双亲表示法
9 class Node {
```

```

10     int val;           // 数据域
11     Node left;         // 左孩子的引用，常常代表左孩子为根的整棵左子树
12     Node right;        // 右孩子的引用，常常代表右孩子为根的整棵右子树
13     Node parent;       // 当前节点的根节点
14 }

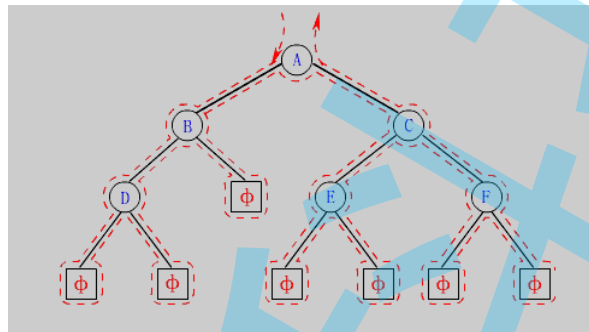
```

孩子双亲表示法后序在平衡树位置介绍，本文采用孩子表示法来构建二叉树。

## 2.6 二叉树的基本操作

### 2.6.1 二叉树的遍历

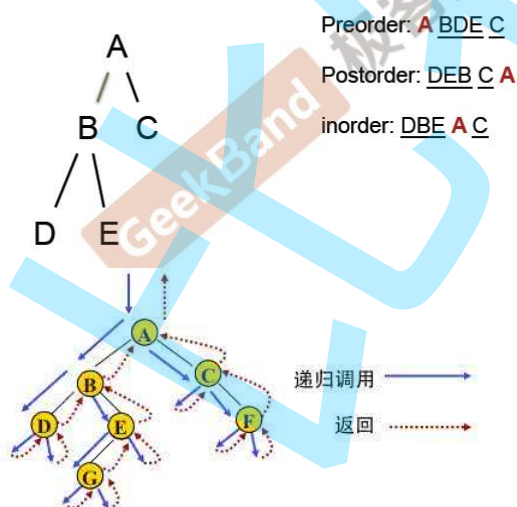
所谓**遍历(Traversal)**是指沿着某条搜索路线，依次对树中每个结点均做一次且仅做一次访问。访问结点所做的操作依赖于具体的应用问题(比如：打印节点内容、节点内容加1)。遍历是二叉树上最重要的操作之一，是二叉树上进行其它运算之基础。



在遍历二叉树时，如果没有进行某种约定，每个人都按照自己的方式遍历，得出的结果就比较混乱，**如果按照某种规则进行约定，则每个人对于同一棵树的遍历结果肯定是相同的**。如果N代表根节点，L代表根节点的左子树，R代表根节点的右子树，则根据遍历根节点的先后次序有以下遍历方式：

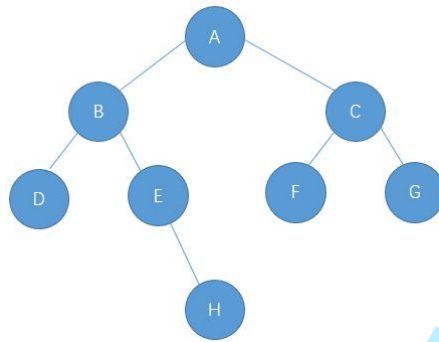
1. NLR：前序遍历(Preorder Traversal 亦称先序遍历)——访问根结点——>根的左子树——>根的右子树。
2. LNR：中序遍历(Inorder Traversal)——根的左子树——>根结点——>根的右子树。
3. LRN：后序遍历(Postorder Traversal)——根的左子树——>根的右子树——>根结点。

由于被访问的结点必是某子树的根，**所以N(Node) 、L(Left subtree) 和R(Right subtree) 又可解释为根、根的左子树和根的右子树**。NLR、LNR和LRN分别又称为先根遍历、中根遍历和后根遍历。



请同学们根据以上二叉树的三种遍历方式，给出以下二叉树的：





前序遍历: \_\_\_\_\_

中序遍历: \_\_\_\_\_

后序遍历: \_\_\_\_\_

## 2.6.2 二叉树的基本操作

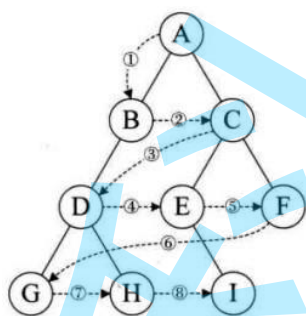
```
1 // 前序遍历
2 void preOrderTraversal(Node root);
3
4 // 中序遍历
5 void inOrderTraversal(Node root);
6
7 // 后序遍历
8 void postOrderTraversal(Node root);
9
10 // 遍历思路-求结点个数
11 static int size = 0;
12 void getSize1(Node root);
13
14 // 子问题思路-求结点个数
15 int getSize2(Node root);
16
17 // 遍历思路-求叶子结点个数
18 static int leafSize = 0;
19 void getLeafSize1(Node root);
20
21 // 子问题思路-求叶子结点个数
22 int getLeafSize2(Node root);
23
24 // 子问题思路-求第 k 层结点个数
25 int getKLevelSize(Node root);
26
27 // 获取二叉树的高度
28 int getHeight(Node root);
29
30 // 查找 val 所在结点, 没有找到返回 null
31 // 按照 根 -> 左子树 -> 右子树的顺序进行查找
32 // 一旦找到, 立即返回, 不需要继续在其他位置查找
33 Node find(Node root, int val);
```

## 2.7 基础面试题

1. 二叉树的前序遍历。 [OJ链接](#)
2. 二叉树中序遍历。 [OJ链接](#)
3. 二叉树的后序遍历。 [课堂不讲解，课后完成作业] [OJ链接](#)
4. 检查两颗树是否相同。 [OJ链接](#)
5. 另一颗树的子树。 [OJ链接](#)
6. 二叉树最大深度。 [OJ链接](#)
7. 判断一颗二叉树是否是平衡二叉树。 [OJ链接](#)
8. 对称二叉树。 [OJ链接](#)

## 2.8 二叉树的层序遍历

设二叉树的根节点所在层数为1，层序遍历就是从所在二叉树的根节点出发，首先访问第一层的树根节点，然后从左到右访问第2层上的节点，接着是第三层的节点，以此类推，自上而下，自左至右逐层访问树的结点的过程就是层序遍历。



```
1 // 层序遍历
2 void levelOrderTraversal(Node root);
3
4 // 判断一棵树是不是完全二叉树
5 boolean isCompleteTree(Node root);
```

## 2.9 进阶面试题

1. 二叉树的构建及遍历。 [OJ链接](#)
2. 二叉树的分层遍历。 [OJ链接](#)
3. 给定一个二叉树，找到该树中两个指定节点的最近公共祖先。 [OJ链接](#)
4. 二叉树搜索树转换成排序双向链表。 [OJ链接](#)
5. 根据一棵树的前序遍历与中序遍历构造二叉树。 [OJ链接](#)
6. 根据一棵树的中序遍历与后序遍历构造二叉树 ([课堂不讲解，课后完成作业])。 [OJ链接](#)
7. 二叉树创建字符串。 [OJ链接](#)

## 2.10 前中后序的非递归实现

```
1 // 前序遍历
2 void preOrderTraversal(Node root);
3
4 // 中序遍历
5 void inOrderTraversal(Node root);
6
7 // 后序遍历
8 void postOrderTraversal(Node root);
```

1. 二叉树前序非递归遍历实现。 ([课堂不讲解, 课后完成作业]) [OJ链接](#)
2. 二叉树中序非递归遍历实现。 ([课堂不讲解, 课后完成作业]) [OJ链接](#)
3. 二叉树后序非递归遍历实现。 ([课堂不讲解, 课后完成作业]) [OJ链接](#)

## 内容重点总结

---

- 掌握二叉树的概念及两大类遍历方式：前中后序遍历 和 层序遍历
- 掌握基本操作及常见面试题的代码实现
- 了解二叉树前中后序的非递归写法

## 课后作业

---

- 博客总结二叉树概念及其遍历方式
- 完成课堂代码