**深入学习二叉树(一) 二叉树基础**

**前言**

**树**是数据结构中的重中之重，尤其以各类二叉树为学习的难点。本系列文章将着重介绍**一般二叉树、完全二叉树、满二叉树、线索二叉树、霍夫曼树、二叉排序树、平衡二叉树、红黑树、B树**。希望各位读者能够关注专题，并给出相应意见，通过系列的学习做到心中有“树”。

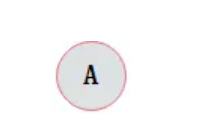
**1 重点概念**

**1.1 结点概念**

**结点**是数据结构中的基础，是构成复杂数据结构的基本组成单位。

**1.2 树结点声明**

本系列文章中提及的结点专指树的结点。例如：结点A在图中表示为：

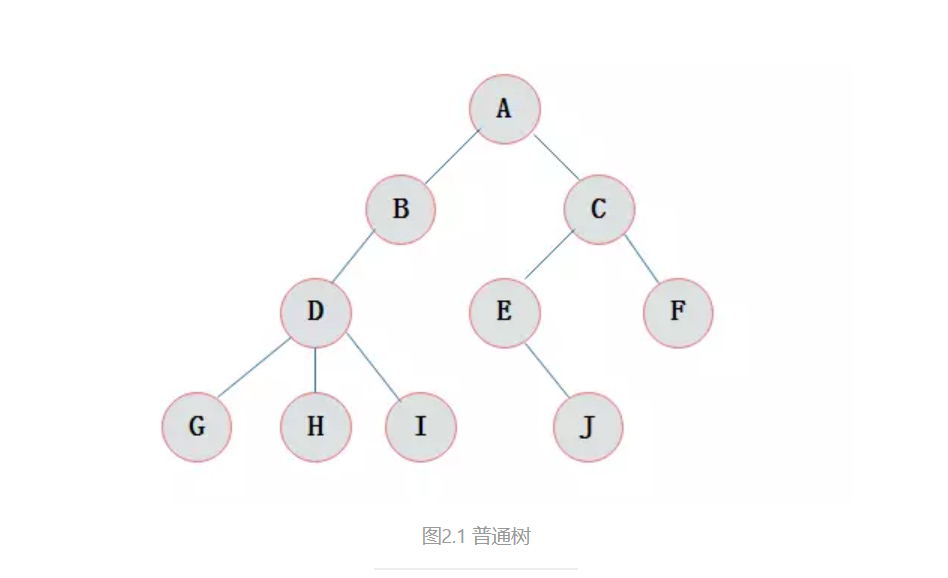


**2 树**

**2.1 定义**

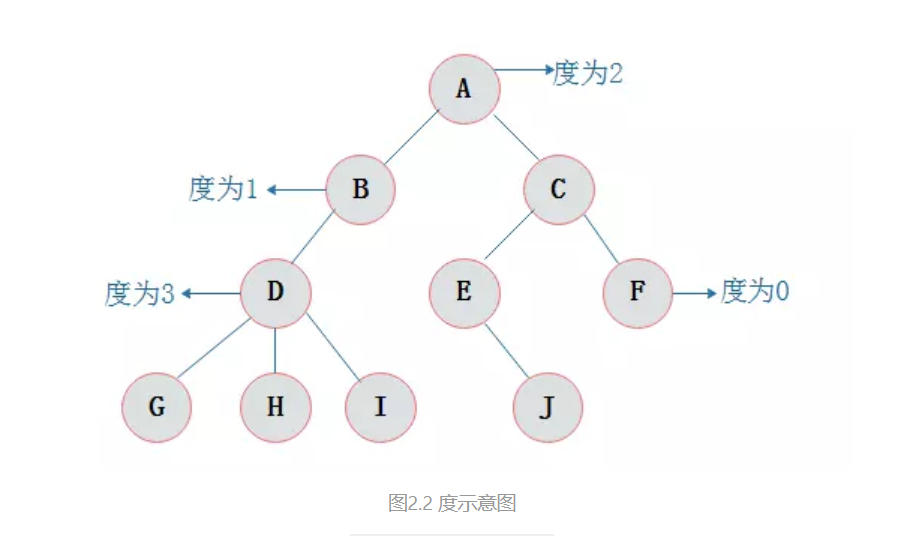
**树（Tree）**是n（n>=0)个结点的有限集。n=0时称为空树。在任意一颗非空树中：  
1）有且仅有一个特定的称为根（Root）的结点；  
2）当n>1时，其余结点可分为m(m>0)个互不相交的有限集T1、T2、......、Tn，其中每一个集合本身又是一棵树，并且称为根的子树。

此外，树的定义还需要强调以下两点：  
1）n>0时根结点是唯一的，不可能存在多个根结点，数据结构中的树只能有一个根结点。  
2）m>0时，子树的个数没有限制，但它们一定是互不相交的。  
示例树：图2.1为一棵普通的树：

由树的定义可以看出，树的定义使用了递归的方式。递归在树的学习过程中起着重要作用。

**2.2 结点的度**

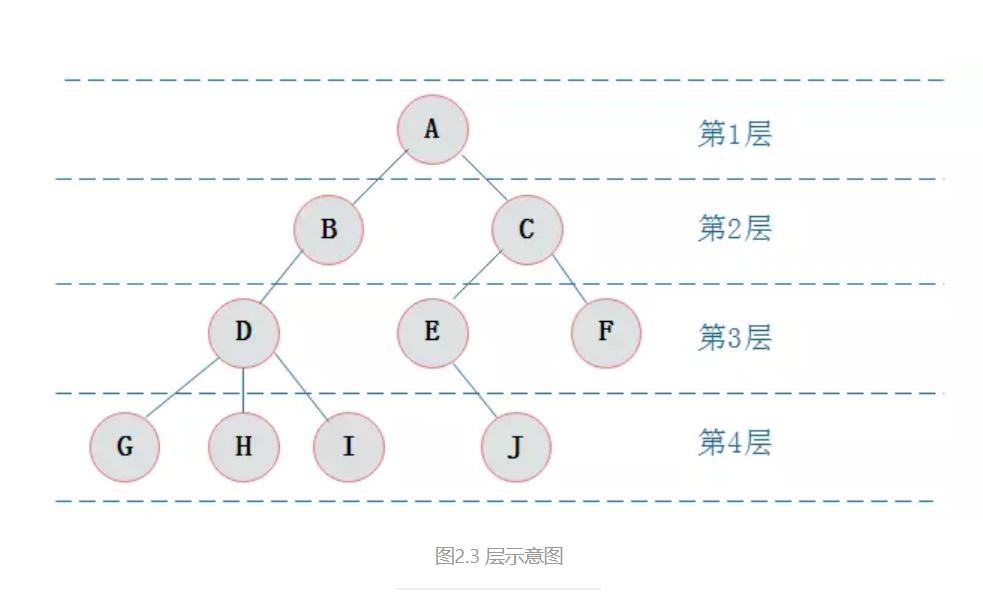
结点拥有的子树数目称为结点的**度**。  
图2.2中标注了图2.1所示树的各个结点的度。

**2.3 结点关系**

结点子树的根结点为该结点的**孩子结点**。相应该结点称为孩子结点的**双亲结点**。  
图2.2中，A为B的双亲结点，B为A的孩子结点。  
同一个双亲结点的孩子结点之间互称**兄弟结点**。  
图2.2中，结点B与结点C互为兄弟结点。

**2.4 结点层次**

从根开始定义起，根为第一层，根的孩子为第二层，以此类推。  
图2.3表示了图2.1所示树的层次关系

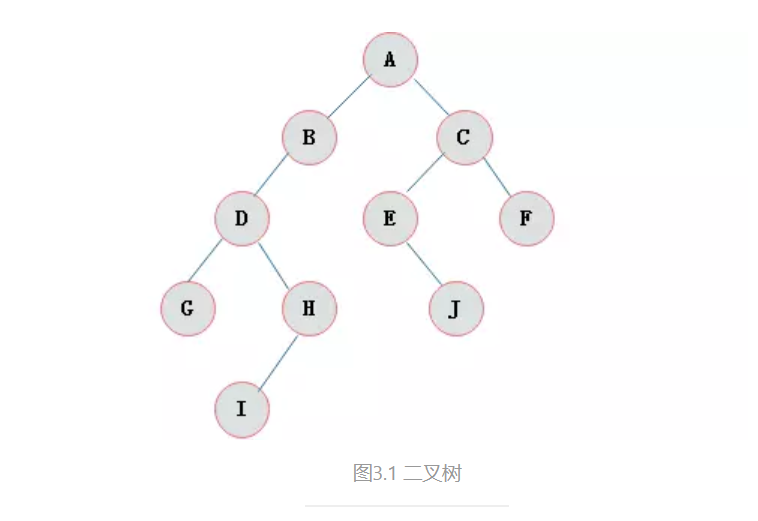
**2.5 树的深度**

树中结点的最大层次数称为树的深度或高度。图2.1所示树的深度为4。

**3 二叉树**

**3.1 定义**

**二叉树**是n(n>=0)个结点的有限集合，该集合或者为空集（称为空二叉树），或者由一个根结点和两棵互不相交的、分别称为根结点的左子树和右子树组成。  
图3.1展示了一棵普通二叉树：

**3.2 二叉树特点**

由二叉树定义以及图示分析得出二叉树有以下特点：  
1）每个结点最多有两颗子树，所以二叉树中不存在度大于2的结点。  
2）左子树和右子树是有顺序的，次序不能任意颠倒。  
3）即使树中某结点只有一棵子树，也要区分它是左子树还是右子树。

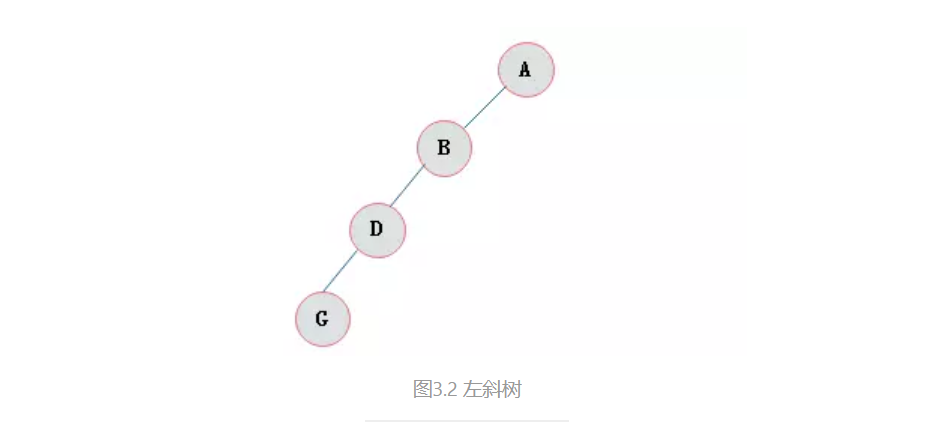
**3.3 二叉树性质**

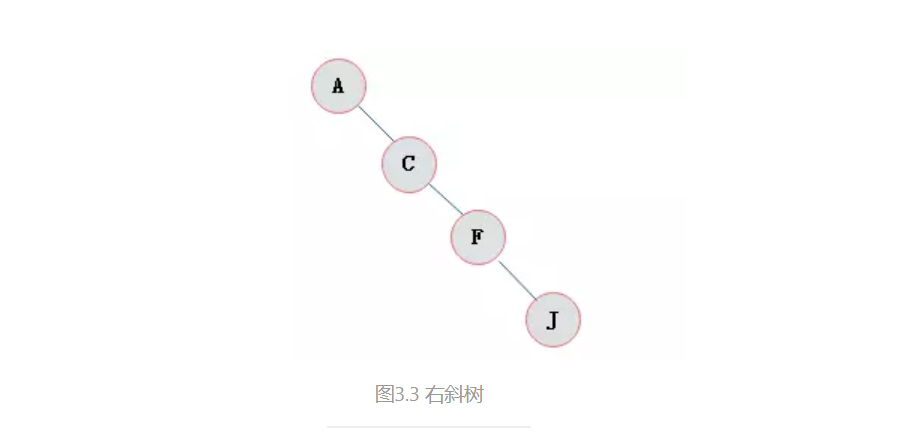
1）在二叉树的第i层上最多有2i-1 个节点 。（i>=1）  
2）二叉树中如果深度为k,那么最多有2k-1个节点。(k>=1）  
3）n0=n2+1 n0表示度数为0的节点数，n2表示度数为2的节点数。  
4）在完全二叉树中，具有n个节点的完全二叉树的深度为[log2n]+1，其中[log2n]是向下取整。  
5）若对含 n 个结点的完全二叉树从上到下且从左至右进行 1 至 n 的编号，则对完全二叉树中任意一个编号为 i 的结点有如下特性：

(1) 若 i=1，则该结点是二叉树的根，无双亲, 否则，编号为 [i/2] 的结点为其双亲结点;  
(2) 若 2i>n，则该结点无左孩子， 否则，编号为 2i 的结点为其左孩子结点；  
(3) 若 2i+1>n，则该结点无右孩子结点， 否则，编号为2i+1 的结点为其右孩子结点。

**3.4 斜树**

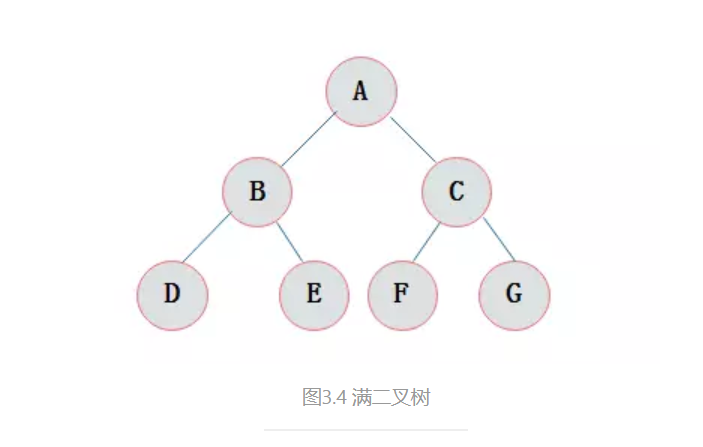
**斜树**：所有的结点都只有左子树的二叉树叫左斜树。所有结点都是只有右子树的二叉树叫右斜树。这两者统称为斜树。



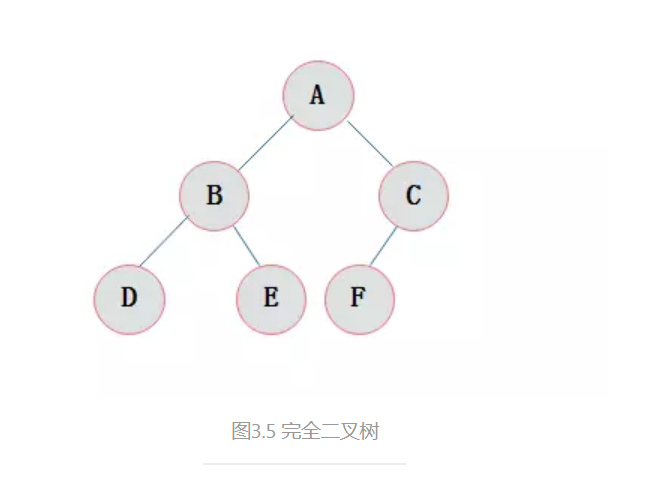


**3.5 满二叉树**

**满二叉树**：在一棵二叉树中。如果所有分支结点都存在左子树和右子树，并且所有叶子都在同一层上，这样的二叉树称为满二叉树。  
满二叉树的特点有：  
1）叶子只能出现在最下一层。出现在其它层就不可能达成平衡。  
2）非叶子结点的度一定是2。  
3）在同样深度的二叉树中，满二叉树的结点个数最多，叶子数最多。

**3.6 完全二叉树**

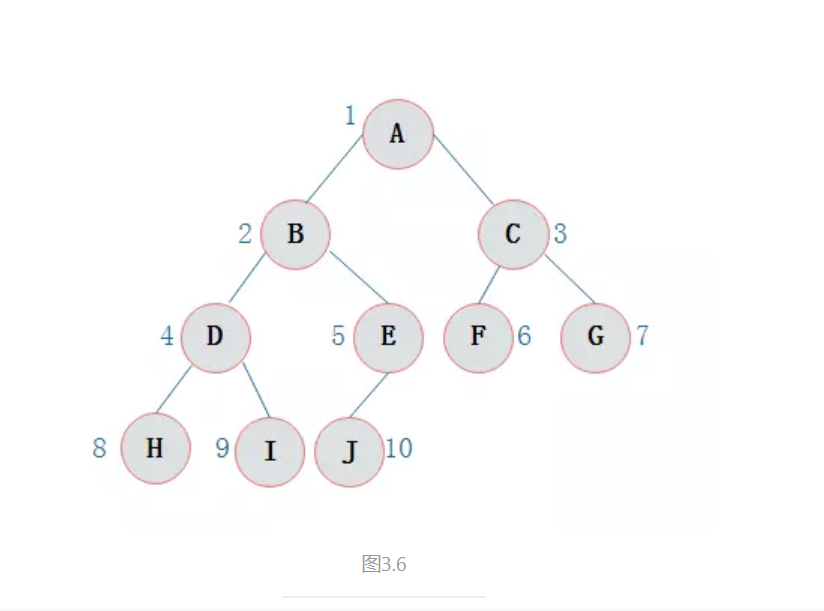
**完全二叉树**：对一颗具有n个结点的二叉树按层编号，如果编号为i(1<=i<=n)的结点与同样深度的满二叉树中编号为i的结点在二叉树中位置完全相同，则这棵二叉树称为完全二叉树。  
图3.5展示一棵完全二叉树

  
**特点**：  
1）叶子结点只能出现在最下层和次下层。  
2）最下层的叶子结点集中在树的左部。  
3）倒数第二层若存在叶子结点，一定在右部连续位置。  
4）如果结点度为1，则该结点只有左孩子，即没有右子树。  
5）同样结点数目的二叉树，完全二叉树深度最小。  
**注**：满二叉树一定是完全二叉树，但反过来不一定成立。

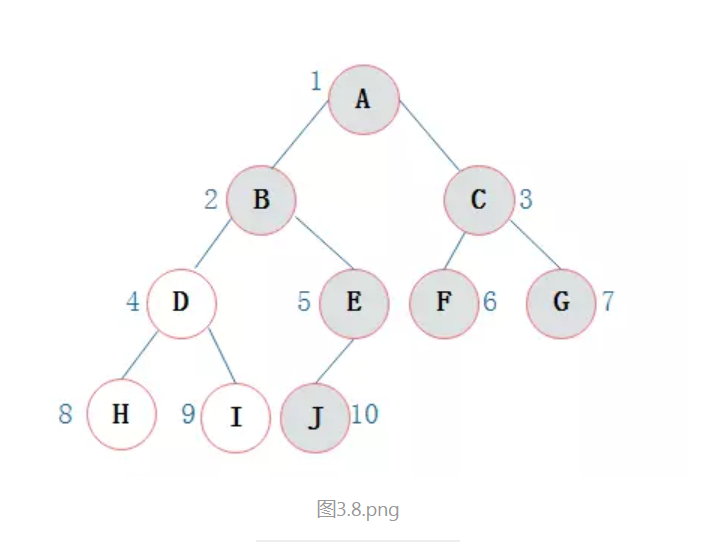
**3.7 二叉树的存储结构**

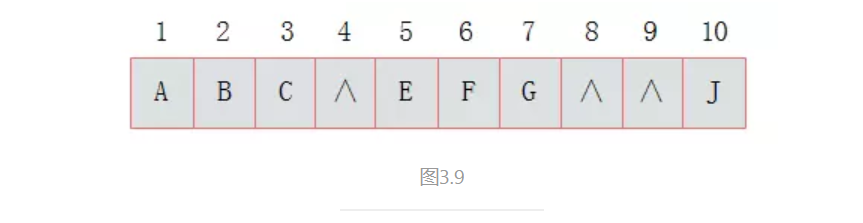
**3.7.1 顺序存储**

二叉树的顺序存储结构就是使用一维数组存储二叉树中的结点，并且结点的存储位置，就是数组的下标索引。

图3.6所示的一棵完全二叉树采用顺序存储方式，如图3.7表示：

由图3.7可以看出，当二叉树为完全二叉树时，结点数刚好填满数组。  
那么当二叉树不为完全二叉树时，采用顺序存储形式如何呢？例如：对于图3.8描述的二叉树：

  
其中浅色结点表示结点不存在。那么图3.8所示的二叉树的顺序存储结构如图3.9所示：

其中，∧表示数组中此位置没有存储结点。此时可以发现，顺序存储结构中已经出现了空间浪费的情况。  
那么对于图3.3所示的右斜树极端情况对应的顺序存储结构如图3.10所示：



由图3.10可以看出，对于这种右斜树极端情况，采用顺序存储的方式是十分浪费空间的。因此，顺序存储一般适用于完全二叉树。

**3.7.2 二叉链表**

既然顺序存储不能满足二叉树的存储需求，那么考虑采用链式存储。由二叉树定义可知，二叉树的每个结点最多有两个孩子。因此，可以将结点数据结构定义为一个数据和两个指针域。表示方式如图3.11所示：

定义结点代码：

typedef struct BiTNode{

TElemType data;//数据

struct BiTNode \*lchild, \*rchild;//左右孩子指针

} BiTNode, \*BiTree;

则图3.6所示的二叉树可以采用图3.12表示。

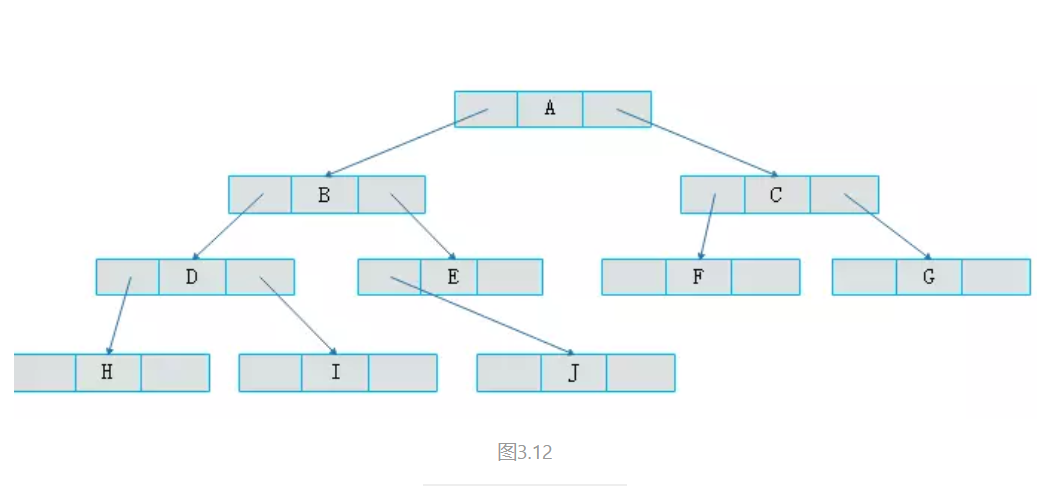


图3.12中采用一种链表结构存储二叉树，这种链表称为二叉链表。

**3.8 二叉树遍历**

二叉树的遍历一个重点考查的知识点。

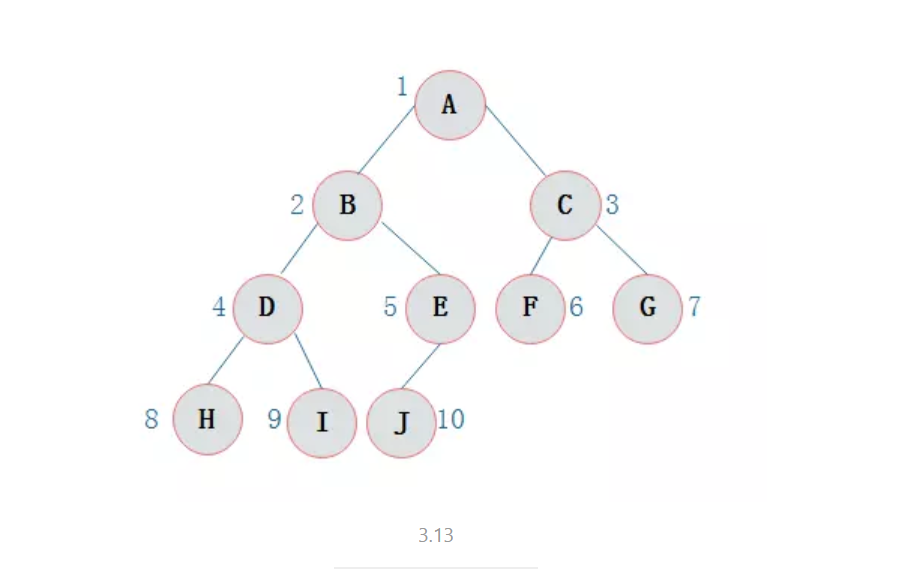
**3.8.1 定义**

**二叉树的遍历**是指从二叉树的根结点出发，按照某种次序依次访问二叉树中的所有结点，使得每个结点被访问一次，且仅被访问一次。  
二叉树的访问次序可以分为四种：

前序遍历  
中序遍历  
后序遍历  
层序遍历

**3.8.2 前序遍历**

**前序遍历**是指从二叉树的根结点出发，按照某种次序依次访问二叉树中的所有结点，使得每个结点被访问一次，且仅被访问一次。（\*先访问根节点——左子树——右子树）

  
图3.13所示二叉树访问如下：

从根结点出发，则第一次到达结点A，故输出A;  
继续向左访问，第一次访问结点B，故输出B；  
按照同样规则，输出D，输出H；  
当到达叶子结点H，返回到D，此时已经是第二次到达D，故不在输出D，进而向D右子树访问，D右子树不为空，则访问至I，第一次到达I，则输出I；  
I为叶子结点，则返回到D，D左右子树已经访问完毕，则返回到B，进而到B右子树，第一次到达E，故输出E；  
向E左子树，故输出J；  
按照同样的访问规则，继续输出C、F、G；

则3.13所示二叉树的前序遍历输出为：  
**ABDHIEJCFG**

**3.8.3 中序遍历**

**中序遍历**就是从二叉树的根结点出发，当第二次到达结点时就输出结点数据，按照先向左在向右的方向访问。（\*先访问左子树——根节点——右子树）

图3.13所示二叉树中序访问如下：

从根结点出发，则第一次到达结点A，不输出A，继续向左访问，第一次访问结点B，不输出B；继续到达D，H；  
到达H，H左子树为空，则返回到H，此时第二次访问H，故输出H；  
H右子树为空，则返回至D，此时第二次到达D，故输出D；  
由D返回至B，第二次到达B，故输出B；  
按照同样规则继续访问，输出J、E、A、F、C、G；

则3.13所示二叉树的中序遍历输出为：  
**HDIBJEAFCG**

**3.8.4 后序遍历**

**后序遍历**就是从二叉树的根结点出发，当第三次到达结点时就输出结点数据，按照先向左在向右的方向访问。（\*左子树——右子树——根节点）

图3.13所示二叉树后序访问如下：

从根结点出发，则第一次到达结点A，不输出A，继续向左访问，第一次访问结点B，不输出B；继续到达D，H；  
到达H，H左子树为空，则返回到H，此时第二次访问H，不输出H；  
H右子树为空，则返回至H，此时第三次到达H，故输出H；  
由H返回至D，第二次到达D，不输出D；  
继续访问至I，I左右子树均为空，故第三次访问I时，输出I；  
返回至D，此时第三次到达D，故输出D；  
按照同样规则继续访问，输出J、E、B、F、G、C，A；

则图3.13所示二叉树的后序遍历输出为：  
**HIDJEBFGCA**  
虽然二叉树的遍历过程看似繁琐，但是由于二叉树是一种递归定义的结构，故采用递归方式遍历二叉树的代码十分简单。  
递归实现代码如下：

/\*二叉树的前序遍历递归算法\*/

void PreOrderTraverse(BiTree T)

{

if(T==NULL)

return;

printf("%c", T->data); /\*显示结点数据，可以更改为其他对结点操作\*/

PreOrderTraverse(T->lchild); /\*再先序遍历左子树\*/

PreOrderTraverse(T->rchild); /\*最后先序遍历右子树\*/

}

/\*二叉树的中序遍历递归算法\*/

void InOrderTraverse(BiTree T)

{

if(T==NULL)

return;

InOrderTraverse(T->lchild); /\*中序遍历左子树\*/

printf("%c", T->data); /\*显示结点数据，可以更改为其他对结点操作\*/

InOrderTraverse(T->rchild); /\*最后中序遍历右子树\*/

}

/\*二叉树的后序遍历递归算法\*/

void PostOrderTraverse(BiTree T)

{

if(T==NULL)

return;

PostOrderTraverse(T->lchild); /\*先后序遍历左子树\*/

PostOrderTraverse(T->rchild); /\*再后续遍历右子树\*/

printf("%c", T->data); /\*显示结点数据，可以更改为其他对结点操作\*/

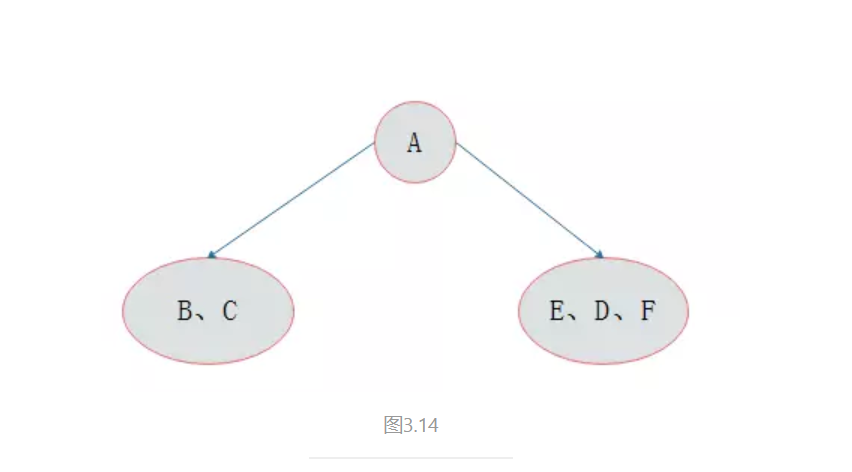
}

**3.8.5 层次遍历**

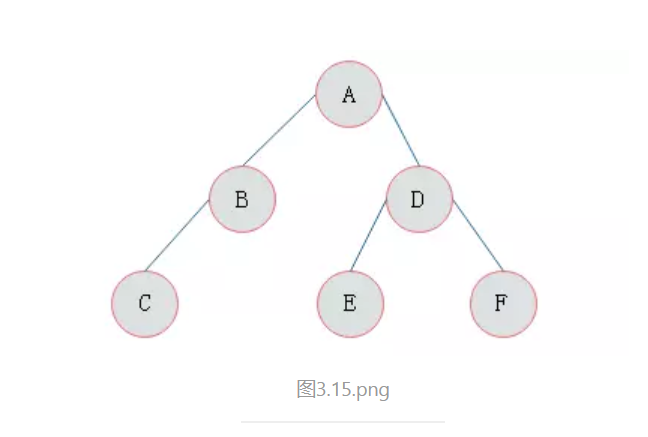
层次遍历就是按照树的层次自上而下的遍历二叉树。针对图3.13所示二叉树的层次遍历结果为：  
**ABCDEFGHIJ**

**3.8.6 遍历常考考点**

对于二叉树的遍历有一类典型题型。  
1）已知前序遍历序列和中序遍历序列，确定一棵二叉树。  
例题：若一棵二叉树的前序遍历为ABCDEF，中序遍历为CBAEDF，请画出这棵二叉树。  
分析：前序遍历第一个输出结点为根结点，故A为根结点。在中序遍历中根结点处于左右子树结点中间，故结点A的左子树中结点有CB，右子树中结点有EDF。  
如图3.14所示：



按照同样的分析方法，对A的左右子树进行划分，最后得出二叉树的形态如图3.15所示：

2）已知后序遍历序列和中序遍历序列，确定一棵二叉树。  
后序遍历中最后访问的为根结点，因此可以按照上述同样的方法，找到根结点后分成两棵子树，进而继续找到子树的根结点，一步步确定二叉树的形态。  
**注**：已知前序遍历序列和后序遍历序列，不可以唯一确定一棵二叉树。