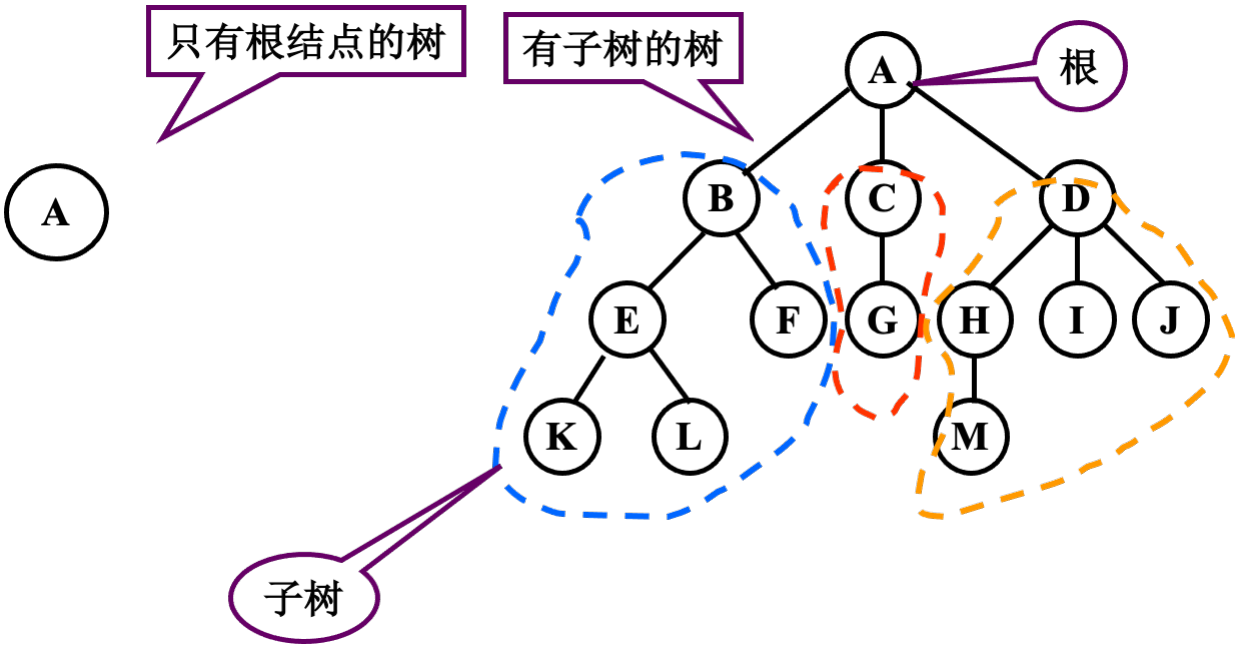
# 06.树、二叉树、线索二叉树

## 1. 树的表达方式

集合中的元素关系呈现出一对多的情况

### 1.1 树的定义

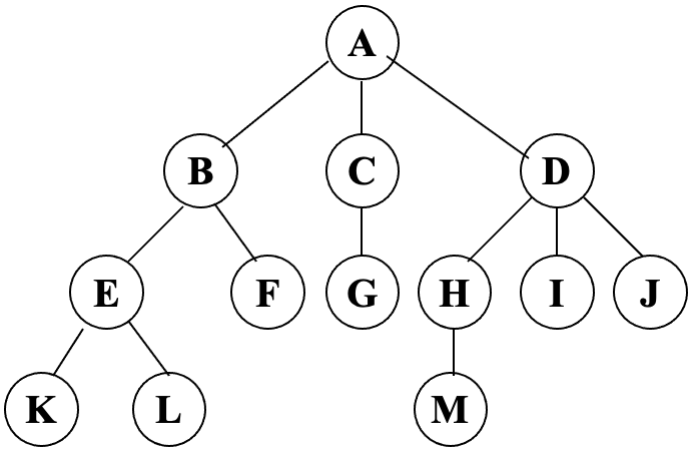
- 树（Tree）是n（n≥0）个节点的有限集合T，它满足两个条件：
  - 有且仅有一个特定的称为根（Root）的节点
  - 其余的节点可以分为m（m≥0）个互不相交的有限集合T1、T2、……、Tm，其中每一个集合又是一棵树，并称为其根的子树（Subtree）。




- 树的定义具有**递归性**，即“树中还有树”。

### 1.2 树的概念

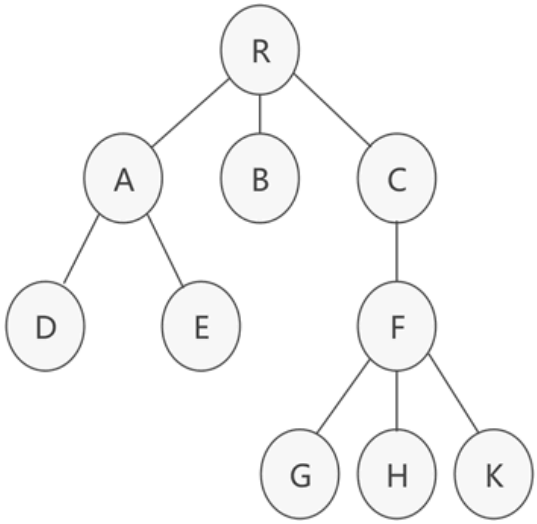
- 结点：使用树结构存储的每一个数据元素都被称为“结点”。例如图中的A就是一个结点。
- 根结点：有一个特殊的结点，这个结点没有前驱，我们将这种结点称之为根结点。
- 父结点（双亲结点）、子结点和兄弟结点：对于ABCD四个结点来说，A就是BCD的父结点，也称之为双亲结点。而BCD都是A的子结点，也称之为孩子结点。对于BCD来说，因为他们都有同一个爹，所以它们互相称之为兄弟结点。
- 叶子结点：如果一个结点没有任何子结点，那么此结点就称之为叶子结点。
- 结点的度：结点拥有的子树的个数，就称之为结点的度。
- 树的度：在各个结点当中，度的最大值。为树的度。
- 树的深度或者高度：结点的层次从根结点开始定义起，根为第一层，根的孩子为第二层。依次类推。




- 结点A的度：3 结点B的度：2 结点M的度：0
- 结点A的孩子：B C D 结点B的孩子：E F
- 树的度：3 树的深度：4
- 叶子结点：K L F G M I J
- 结点A是结点F的祖先
- 结点F是结点K的叔叔结点

## 1.3 树的存储结构

### 1.3.1 双亲表示法




双亲表示法采用顺序表（也就是数组）存储普通树，其实现的核心思想是：顺序存储各个节点的同时，给各节点附加一个记录其父节点位置的变量。

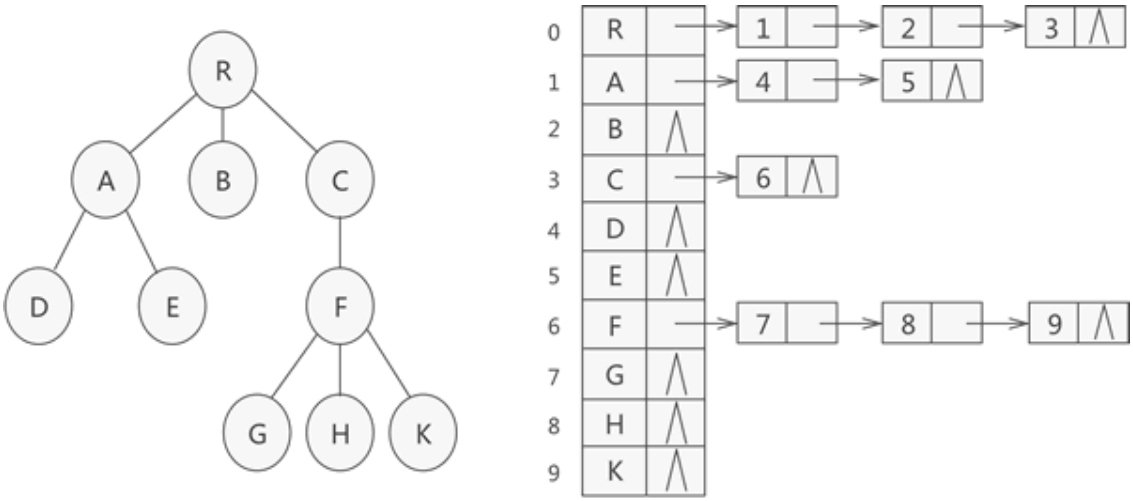
根节点没有父节点（父节点又称为双亲节点），因此根节点记录父节点位置的变量通常置为 −1。

- 利用顺序表存储，表元素由数据和父结点构成
- 特点分析：
  - 根结点没有双亲，所以位置域设置为−1
  - 知道一个结点，找他的父结点，非常容易，O(1)级
  - 找孩子节点，必须遍历整个表

| 数组下标 | data | parent |
|---|---|---|
| 0 | R | -1 |
| 1 | A | 0 |
| 2 | B | 0 |
| 3 | C | 0 |
| 4 | D | 1 |
| 5 | E | 1 |
| 6 | F | 3 |
| 7 | G | 6 |
| 8 | H | 6 |
| 9 | K | 6 |

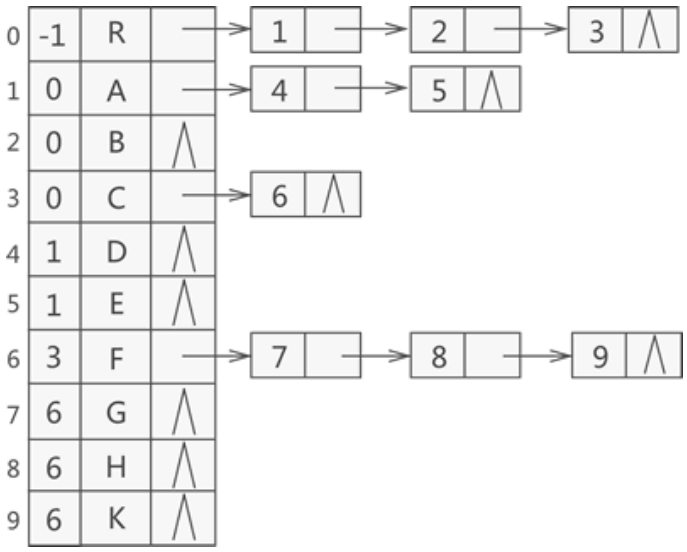
### 1.3.2 孩子表示法

> 孩子表示法存储普通树采用的是 ”顺序表+链表” 的组合结构。
>
> 其存储过程是：从树的根节点开始，使用顺序表依次存储树中各个节点。需要注意，与双亲表示法不同的是，孩子表示法会给各个节点配备一个链表，用于存储各节点的孩子节点位于顺序表中的位置。
>
> 如果节点没有孩子节点（叶子节点），则该节点的链表为空链表。




a) 普通树　　b) 孩子表示法

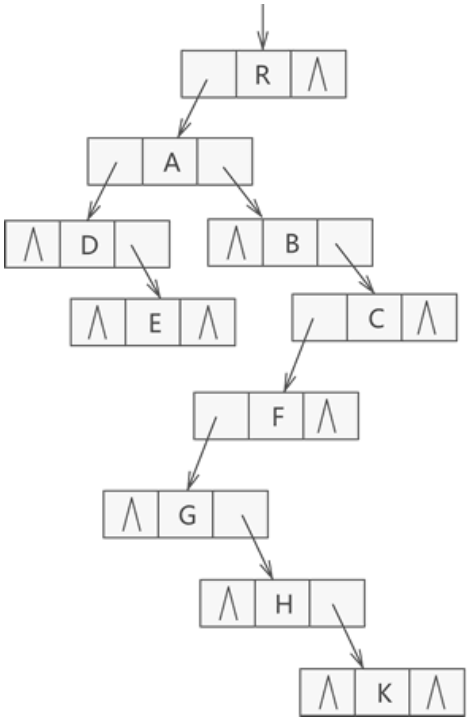
> 使用孩子表示法存储的树结构，正好和双亲表示法相反，查找孩子结点的效率很高，而不擅长做查找父结点的操作。
>
> 我们还可以将双亲表示法和孩子表示法合二为一：




### 1.3.3 孩子兄弟表示法

> 在树结构中，同一层的节点互为兄弟节点。例如普通树中，节点 A、B 和 C 互为兄弟节点，而节点 D、E 和 F 也互为兄弟节点。
>
> 所谓孩子兄弟表示法，指的是用将整棵树用二叉链表存储起来，具体实现方案是：从树的根节点开始，依次存储各个结点的孩子结点和兄弟结点。
>
> 在二叉链表中，各个结点包含三部分内容：
>
> | 孩子指针域 | 数据域 | 兄弟指针域 |
> | --- | --- | --- |

孩子兄弟表示法示例图

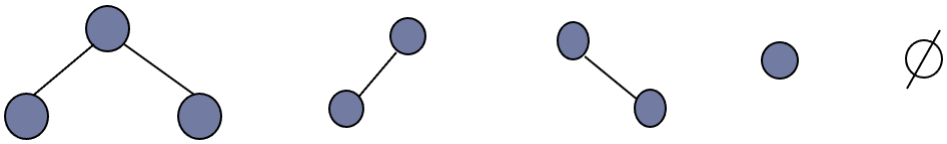



> 在以孩子兄弟表示法构建的二叉链表中，如果要查找结点 x 的所有孩子，则只要根据该结点的 firstchild 指针找到它的第一个孩子，然后沿着孩子结点的 nextsibling 指针不断地找它的兄弟结点，就可以找到结点 x 的所有孩子。

# 2. 二叉树简介

## 2.1 二叉树定义

- 二叉树是每个结点最多有两个子树的树结构。二叉树不允许存在度大于2的树。
- 它有五种最基本的形态：
    - 二叉树可以是空集
    - 根可以有空的左子树或者右子树；
    - 左右子树都是空。只有左子树或者右子树的叫做斜树。

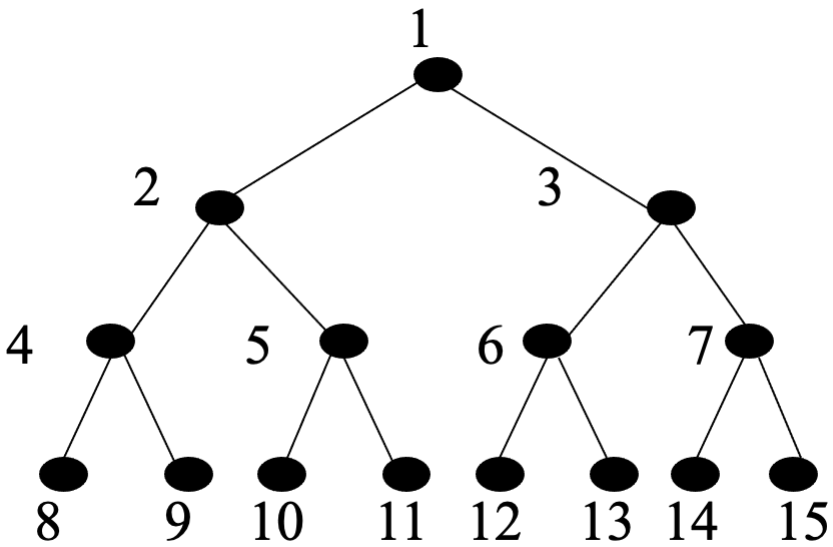


## 2.2 二叉树的概念和性质

### 2.2.1 完全二叉树和满二叉树

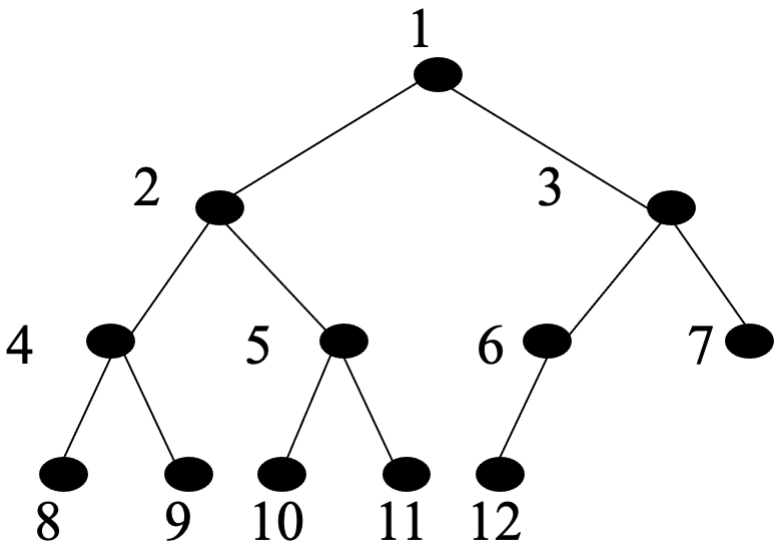
- 满二叉树

> 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，并且所有叶子结点都在同一层上，这样的二叉树称为满二叉树。
>
> 一棵深度为k且有$2^k-1$个结点的二叉树称为满二叉树。( k ≥ 1)




- 完全二叉树

> 如果一棵深度为k，有n个结点的二叉树中各结点能够与深度为k的顺序编号的满二叉树从1到n标号的结点相对应的二叉树称为完全二叉树。




> 特点：
>
> - 所有的叶结点都出现在第k层或k－1层
> - 若任一结点，如果其右子树的最大层次为i，则其左子树的最大层次为i或i＋1

### 2.2.2 二叉树的性质

- 性质1：

> 在二叉树的第i层上的结点最多为$2^{(i-1)}$个。(i ⩾ 1)

- 性质2：

> 深度为k的二叉树至多有$2^{k}$–1个结点。(i ⩾ 1)

- 性质3：

> 在一棵二叉树中，叶结点的数目比度为2的结点数目多一个。
>
> a. 总节点数为各类节点之和：$n = n_0 + n_1 + n_2$
>
> b. 总节点数为所有子节点数加一：$n = n_1 + 2*n_2 + 1$
>
> 故：$n_0 = n_2 + 1$

- 性质4：

> 具有N个结点的完全二叉树的深度为$\log_2 N+1$。（向下取整）
>
> 因为树高为$h$，有$2^{h-1} - 1 < n \leq 2^h - 1$
>
> 所以$h - 1 < log_2(n+1) \leq h$
>
> 所以$h = \lceil \log_2(n+1) \rceil$
>
> 同理$2^{h-1} \leq n < 2^h$
>
> 所以$h - 1 \leq log_2(n) < h$
>
> 所以$h = \lfloor log_2 n \rfloor + 1$

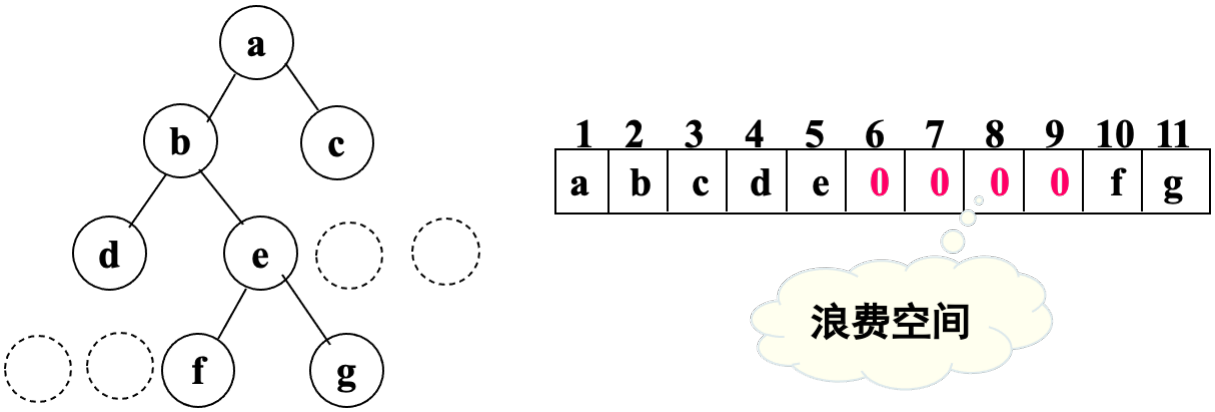
- 性质5：

> 如果有一棵n个结点的完全二叉树，其结点编号按照层次序（从上到下，从左到右），则除根结点外，满足[i/2 , i, 2i, 2i+1]的规则

## 2.3 二叉树的存储

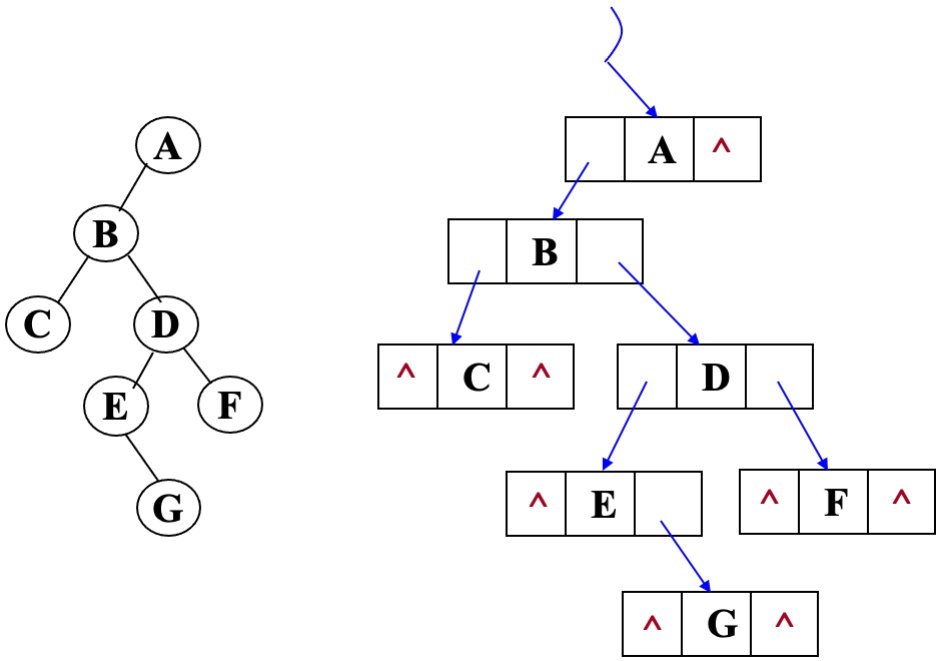
### 2.3.1 顺序存储

- 依靠性质5，可以将任意棵二叉树构造成满二叉树结构或完全二叉树结构，依据下标规则，就可以找到父结点，子结点。




### 2.3.2 链式存储

- 由于二叉树的每个结点最多只能有两个子树，每个结点定义两个指针域和一个数据域即可。



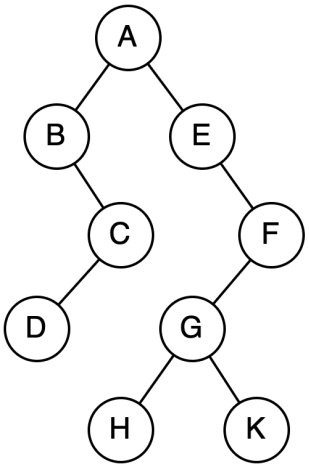

## 2.4 二叉树存储的核心代码

# 3. 二叉树的遍历

## 3.1 遍历思想

- 遍历 ： 沿某条搜索路径周游二叉树，对树中的每一个节点访问一次且仅访问一次。

- 对线性结构而言，只有一条搜索路径（因为每个结点均只有一个后继），故不需要另加讨论。
- 二叉树是非线性结构，每个结点有两个后继，则存在如何遍历即按什么样的搜索路径进行遍历的问题。
    - 按层次，父子关系，知道了父，那么就把其所有的子结点都看一遍
    - 按深度，一条道走到黑，然后再返回走另一条道




遍历结果
先序遍历：A B C D E F G H K

中序遍历：B D C A E H G K F

后序遍历：D C B H K G F E A

## 3.2 广度遍历

- 算法思想
    - 引入队列，将根结点入队
    - 从队列中取出队头元素，访问该结点，将该结点的所有孩子节点入队
    - 再次从队列中取出队头元素，并访问，以此重复
    - 直到队列为空，说明所有元素都遍历完成
- 算法实现

## 3.3 递归

递归的概念
- 递归其实就是某个函数直接或者间接的调用了自身。这种调用方式叫递归调用。说白了还是一个函数调用。
- 既然是函数调用，那么就有一个雷打不动的原则：所有被调用的函数都将创建一个副本，各自为调用者服务，而不受其他函数的影响。

递归的条件
递归函数分为两个条件，边界条件和递归条件。

- 边界条件：就是递归中止条件，避免出现死循环。也叫做递归出口。
- 递归条件：也就是递归体。将一个大问题分解为一步步小问题。也是递归调用的阶段。

递归函数在具备这两个要素以后，才可以在有限次的计算后得出想要的结果。

## 3.4 深度遍历

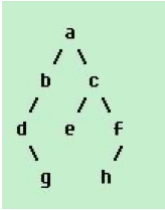
### 3.4.1 先序遍历

- 先访问根结点、然后左子树、最后右子树

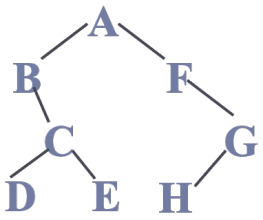
### 3.4.2 中序遍历

### 3.4.3 后序遍历

## 3.5 根据遍历结果重构二叉树

1. 若某二叉树的前遍历访问顺序是序abdgcefh，中序遍历顺序是dgbaechf，则后序遍历的访问顺序是什么。
    - gdbehfca 

2. 已知一棵二叉树的中序序列和后序序列分别是BDCEAFHG 和 DECBHGFA，请画出这棵二叉树。




# 4. 线索二叉树

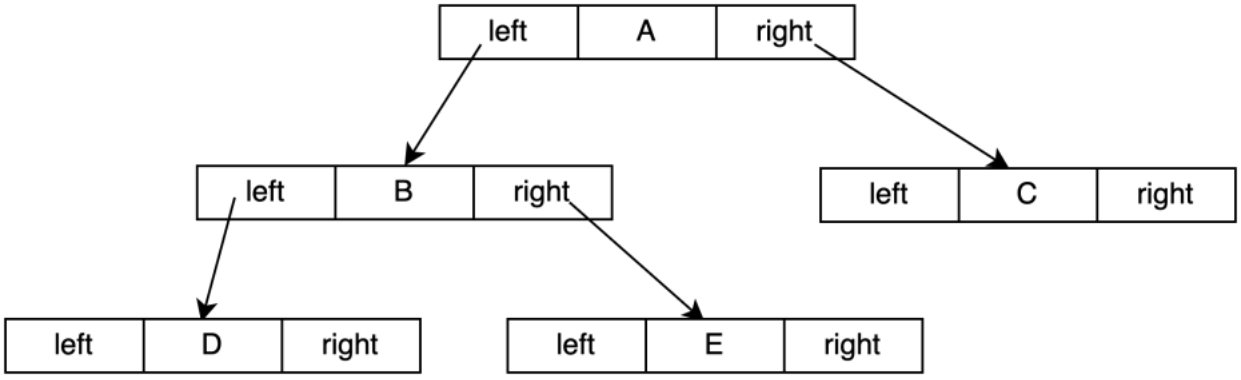
## 4.1 背景介绍

现有一棵结点数目为n的二叉树，采用二叉链表的形式存储，对于每个结点均有指向左右孩子的两个指针域。

结点为n的二叉树一共有n−1条有效分支路径。那么，则二叉链表中存在

2n −（ n−1 ）= n + 1个 空指针域

那么，这些空指针造成了空间浪费。

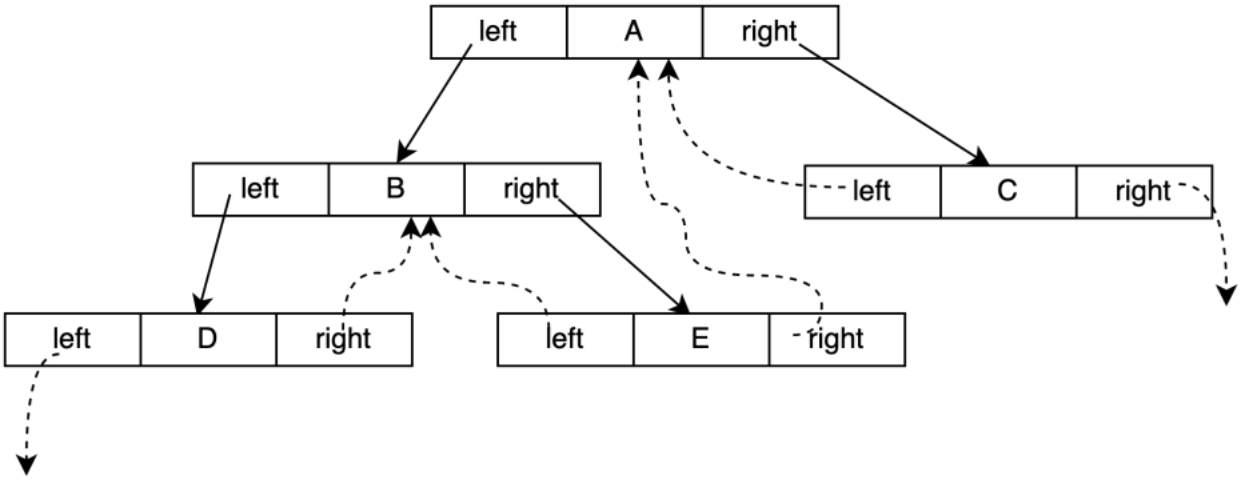



当对二叉树进行中序遍历时可以得到二叉树的中序序列。

> 如图所示二叉树的中序遍历结果为DBEAC，可以得知A的前驱结点为E，后继结点为C。
>
> 这种关系的获得是建立在完成遍历后得到 的，那么可不可以在建立二叉树时就记录下前驱后继的关系呢，那么在后续寻找前驱结点和后继结点时将大大提升效率。

## 4.2 线索化

- 将某结点的空指针域指向该结点的前驱后继，定义规则如下：
  - 若结点的左子树为空，则该结点的左孩子指针指向其前驱结点。
  - 若结点的右子树为空，则该结点的右孩子指针指向其后继结点。
- 这种指向前驱和后继的指针称为线索。将一棵普通二叉树以某种次序遍历，并添加线索的过程称为线索化。




## 4.3 线索化的改进

- 可以将一棵二叉树线索化为一棵线索二叉树，那么新的问题产生了。我们如何区分一个结点的 lchild指针是指向左孩子还是前驱结点呢?
- 为了解决这一问题，现需要添加标志位ltag，rtag。并定义规则如下：
  - ltag为0时，指向左孩子，为1时指向前驱
  - rtag为0时，指向右孩子，为1时指向后继


- 在遍历过程中，如果当前结点没有左孩子，需要将该结点的 lchild 指针指向遍历过程中的前一个结点，所以在遍历过程中，设置一个指针（名为 pre ），时刻指向当前访问结点的前一个结点。

## 4.4 线索化的优势

- 递归遍历需要使用系统栈，非递归遍历需要使用内存中的空间来帮助遍历，而线索化之后就不需 要这些辅助了，直接可以像遍历数组一样遍历。
- 线索二叉树核心目的在于加快查找结点的前驱和后继的速度。如果不使用线索的话，当查找一个 结点的前驱与后继需要从根节点开始遍历，当然，如果二叉树数据量较小时，可能线索化之后作用不 大，但是当数据量很大时，线索化所带来的性能提升就会比较明显。