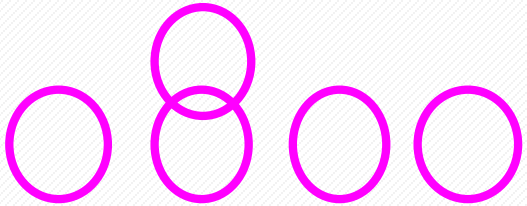
第07讲 树和二叉树的性质与存储结构

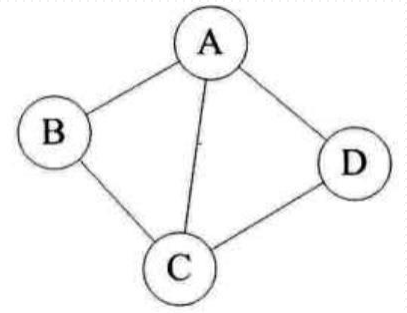
六星教育首席架构师：Vico老师
官方助理冰芯老师QQ：1930070991

集合——数据元素间除“同属于一个集合”外，无其它关系

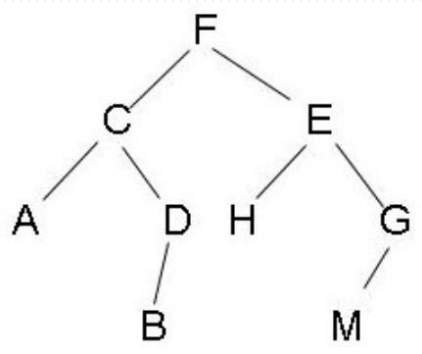


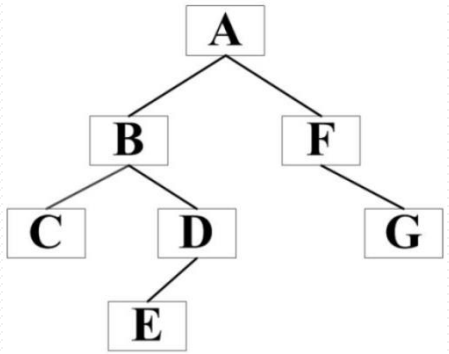
线性结构——一个对一个，如线性表、栈、队列

图形结构——多个对多个，如图



树形结构——一个对多个，如树





7.1 树和二叉树的定义

7.2 案例引入

7.3 树和二叉树的抽象数据类型定义

7.4 二叉树的性质和存储结构

7.5 遍历二叉树

本节教学目标

1. 掌握二叉树的基本概念、性质和存储结构

2. 熟练掌握二叉树的前、中、后序遍历方法

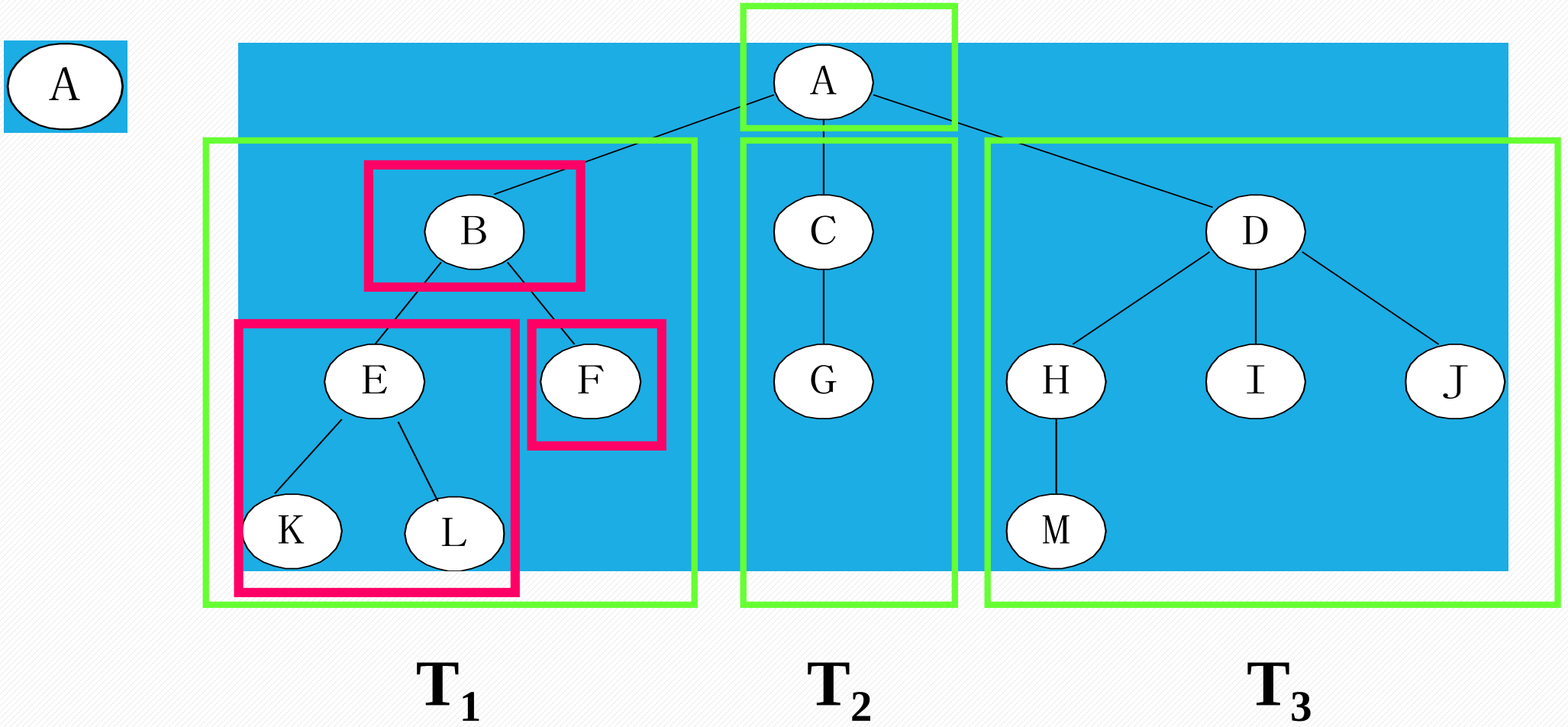
7.1 树和二叉树的定义

树的定义

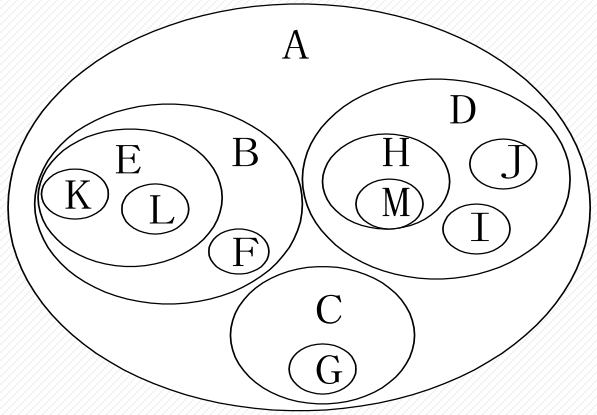
树（Tree）是n（$n \geq 0$）个结点的有限集，它或为空树（$n = 0$）；或为非空树，对于非空树T:

（1）有且仅有一个称之为根的结点；
（2）除根结点以外的其余结点可分为m（$m > 0$）个互不相交的有限集T_1, T_2, …, T_m, 其中每一个集合本身又是一棵树，并且称为根的子树（SubTree）。

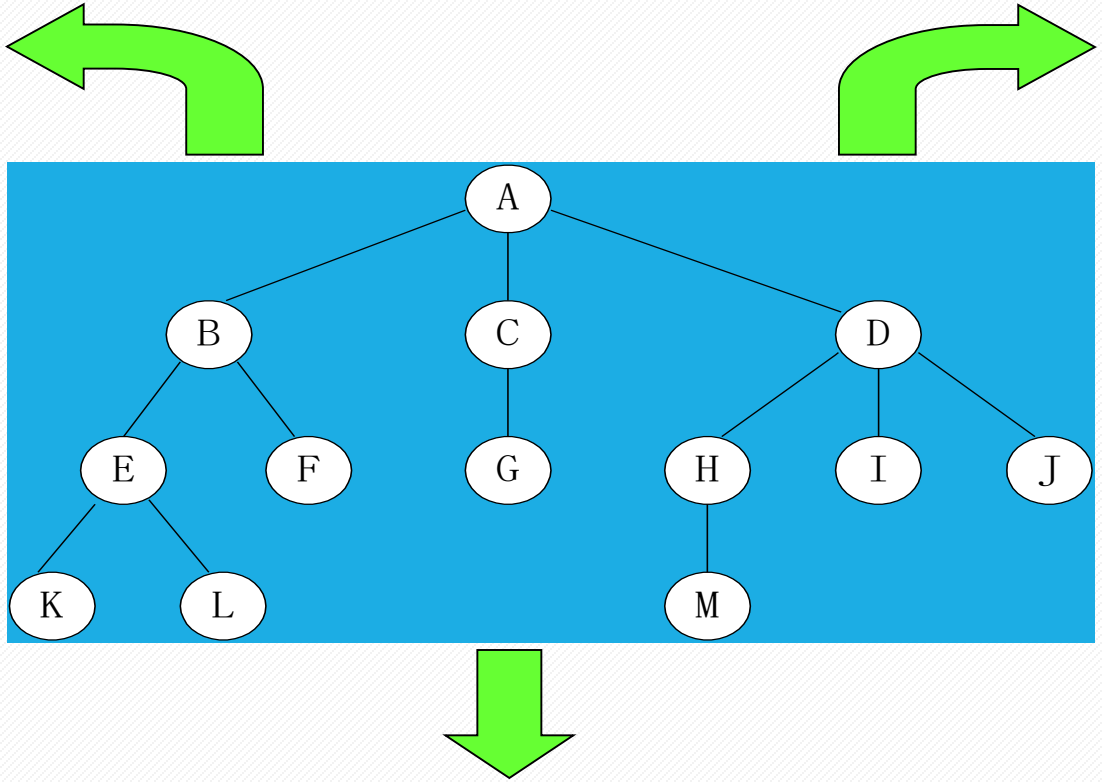
树是n个结点的有限集

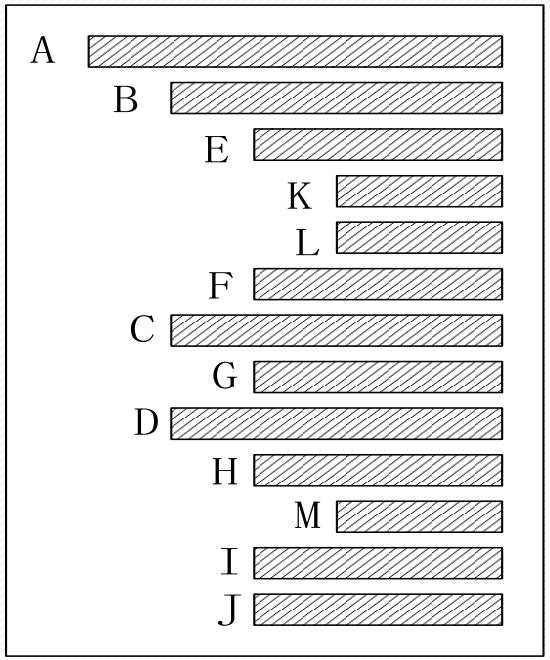


树的其它表示方式



嵌套集合





凹入表示

(A (B (E (K, L) , F) , C (G) , D (H (M) , I, J)))

广义表

基本术语

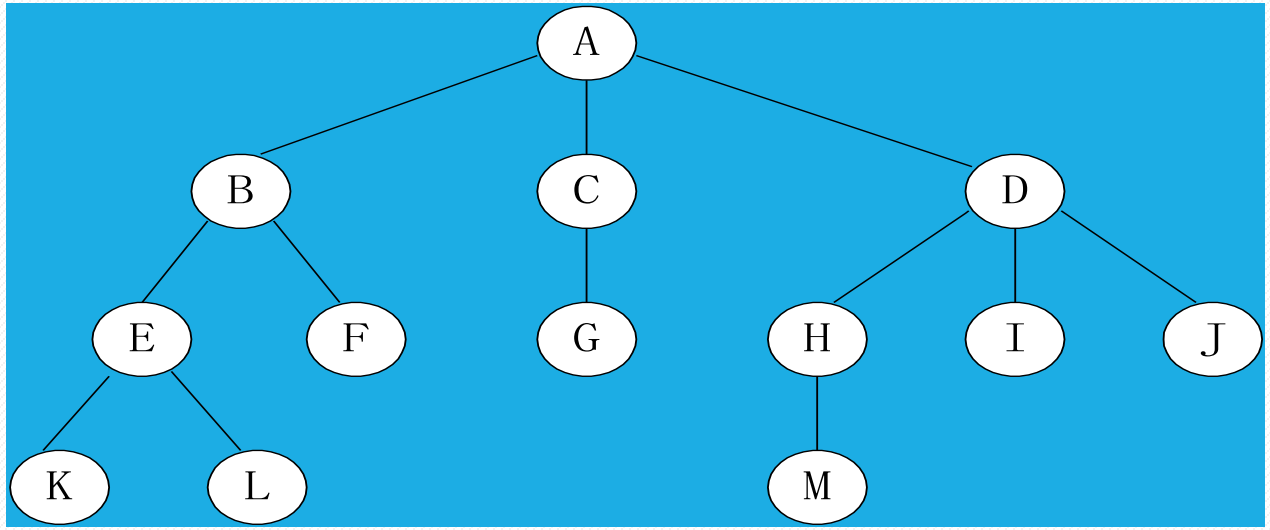
根 —— 即根结点(没有前驱)

叶子 —— 即终端结点(没有后继)

森林 —— 指m棵不相交的树的集合(例如删除A后的子树个数)

有序树 —— 结点各子树从左至右有序，不能互换（左为第一）

无序树 —— 结点各子树可互换位置。



基本术语

双亲 —— 即上层的那个结点(直接前驱)

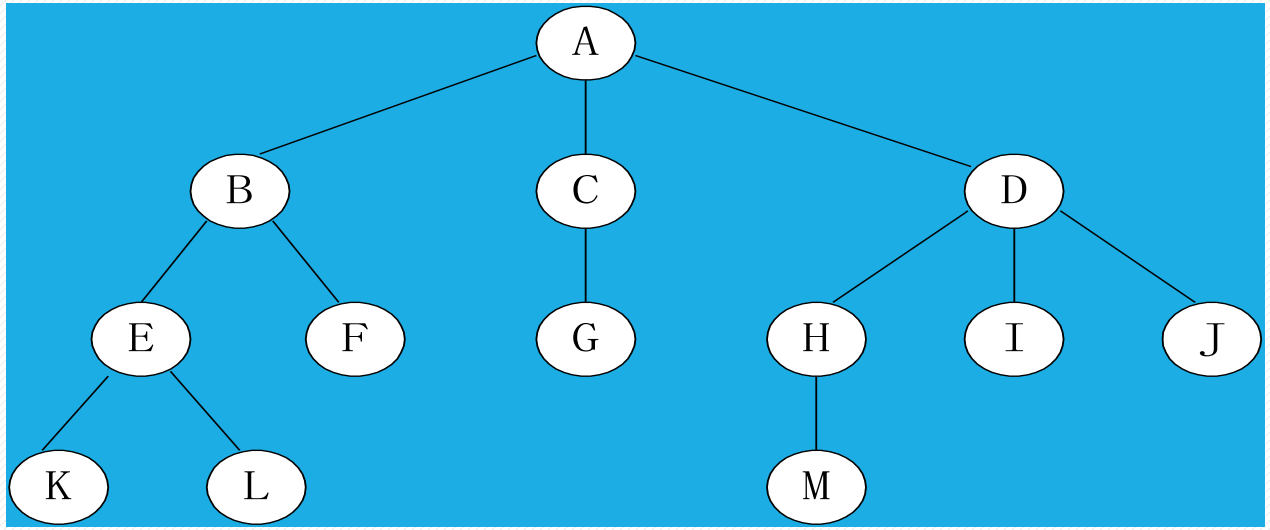
孩子 —— 即下层结点的子树的根(直接后继)

兄弟 —— 同一双亲下的同层结点（孩子之间互称兄弟）

堂兄弟 —— 即双亲位于同一层的结点（但并非同一双亲）

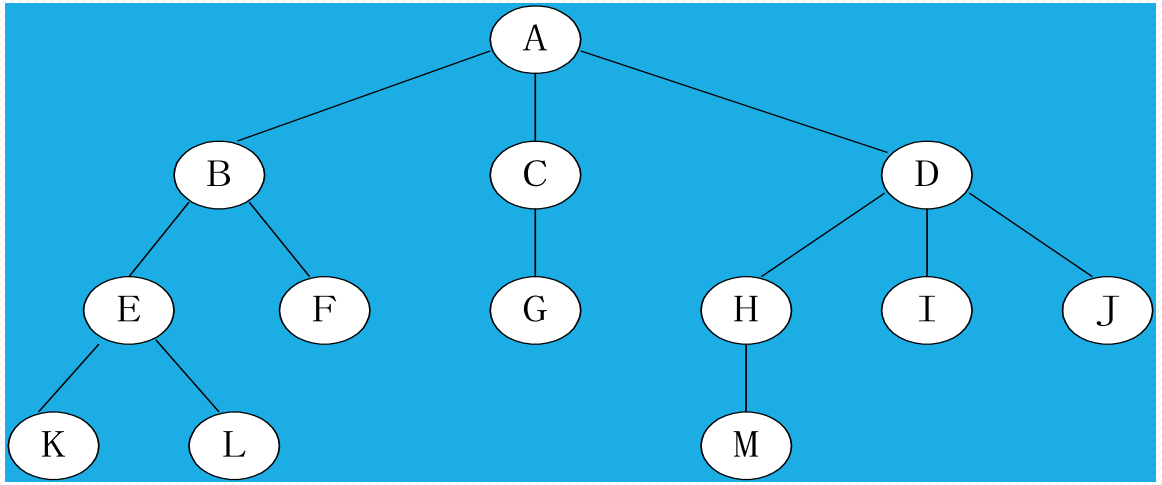
祖先 —— 即从根到该结点所经分支的所有结点

子孙 —— 即该结点下层子树中的任一结点



基本术语

结点 —— 即树的数据元素
结点的度 —— 结点挂接的子树数
结点的层次 —— 从根到该结点的层数（根结点算第一层）
终端结点 —— 即度为0的结点，即叶子节点
分支结点 —— 即度不为0的结点（也称为内部结点）
树的度 —— 所有结点度中的最大值
树的深度(或高度) —— 指所有结点中最大的层数



二叉树的定义

二叉树（Binary Tree）是n（$n \geq 0$）个结点所构成的集合，它或为空树（$n = 0$）；或为非空树，对于非空树T：

（1）有且仅有一个称之为根的结点；
（2）除根结点以外的其余结点分为两个互不相交的子集T_1和T_2，分别称为T的左子树和右子树，且T_1和T_2本身又都是二叉树。

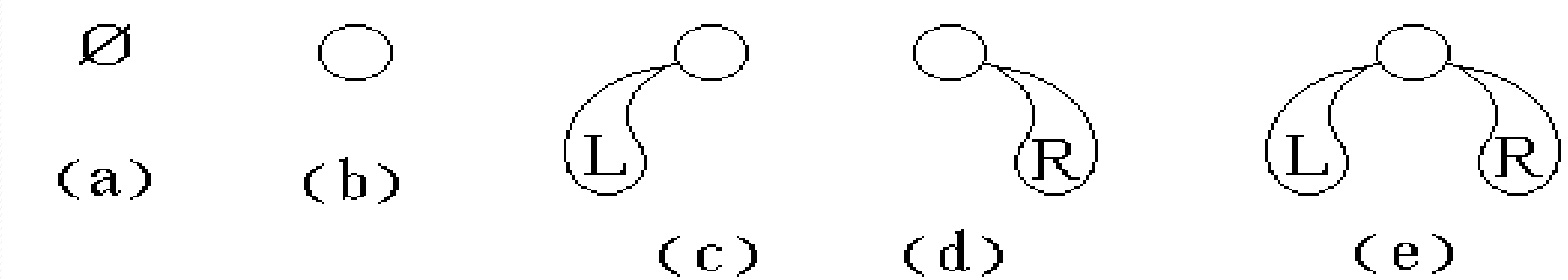
普通树（多叉树）若不转化为二叉树，则运算很难实现

为何要重点研究每结点最多只有两个“叉”的树?

- ✓ 二叉树的结构最简单，规律性最强；
- ✓ 可以证明，所有树都能转为唯一对应的二叉树，不失一般性。

二叉树基本特点：

- 结点的度小于等于2
- 有序树（子树有序，不能颠倒）

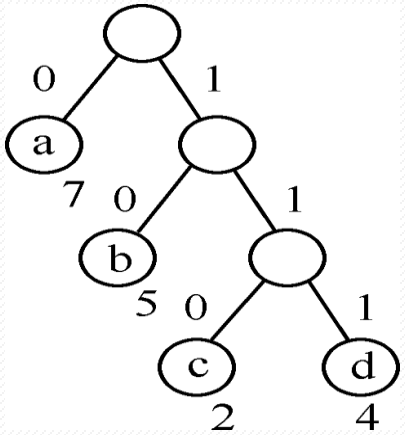


二叉树的五种不同形态

7.2 案例引入

案例7.1： 数据压缩问题

将数据文件转换成由0、1组成的二进制串，称之为编码。



(a) 等长编码方案

字符	编码
a	00
b	01
c	10
d	11

(b) 不等长编码方案1

字符	编码
A	0
B	10
C	110
D	111

(c) 不等长编码方案2

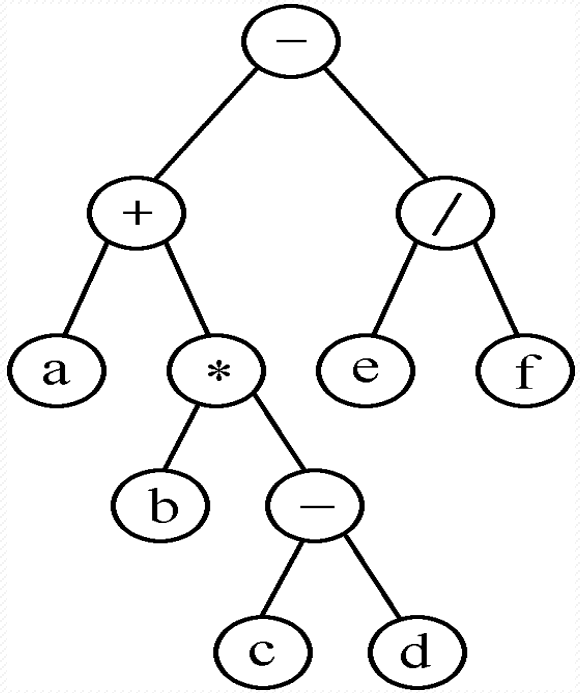
字符	编码
A	0
B	01
C	011
D	111

案例7.2：利用二叉树求解表达式的值

以二叉树表示表达式的递归定义如下：

（1）若表达式为数或简单变量，则相应二叉树中仅有一个根结点，其数据域存放该表达式信息；

（2）若表达式为“第一操作数 运算符 第二操作数”的形式，则相应的二叉树中以左子树表示第一操作数，右子树表示第二操作数，根结点的数据域存放运算符（若为一元运算符，则左子树为空），其中，操作数本身又为表达式。



(a + b *(c-d)-e/f)的二叉树

7.3 树和二叉树的抽象数据类型定义

二叉树的抽象数据类型定义

ADT BinaryTree {

数据对象D: D是具有相同特性的数据元素的集合。

数据关系R:

若D=Φ，则R= Φ；

若D≠Φ，则R= {H}；存在二元关系：

① root 唯一 // 关于根的说明

② $D_j \cap D_k = \Phi$ // 关于子树不相交的说明

③ …… // 关于数据元素的说明

④ …… // 关于左子树和右子树的说明

基本操作 P:

} ADT BinaryTree // 至少有20个

CreateBiTree(&T,definition)

初始条件：definition给出二叉树T的定义。

操作结果：按definition构造二叉树T。

PreOrderTraverse(T)

初始条件：二叉树T存在。

操作结果：先序遍历T，对每个结点访问一次。

InOrderTraverse(T)

初始条件：二叉树T存在。

操作结果：中序遍历T，对每个结点访问一次。

PostOrderTraverse(T)

初始条件：二叉树T存在。

操作结果：后序遍历T，对每个结点访问一次。

7.4 二叉树的性质和存储结构

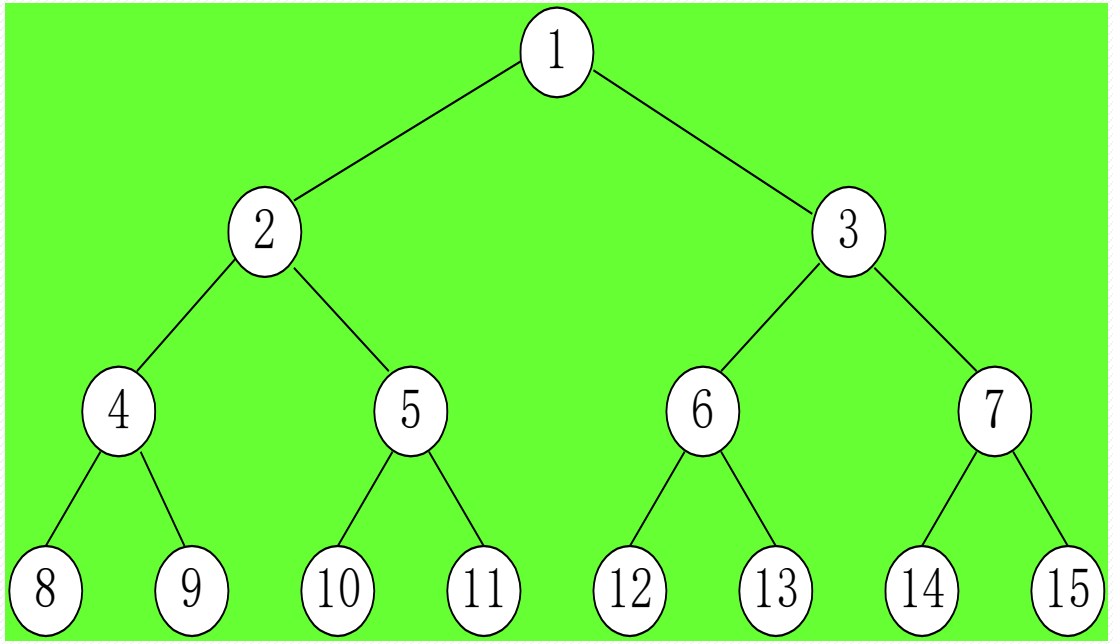
性质1: 在二叉树的第i层上最多有2^{i-1}个结点

提问：第i层上至少有 I 个结点?

性质2: 深度为k的二叉树最多有2^k-1个结点

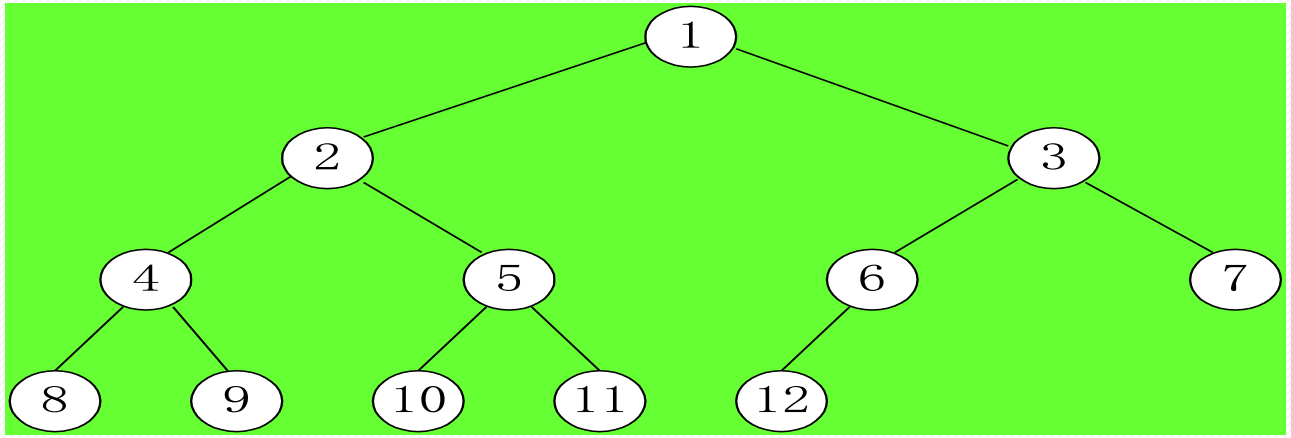
提问：深度为k时至少有 k 个结点?

特殊形态的二叉树



满二叉树：一棵深度为k且有2^k-1个结点的二叉树。
(特点：每层都“充满”了结点)

只有最后一层叶子不满，且全部集中在右边

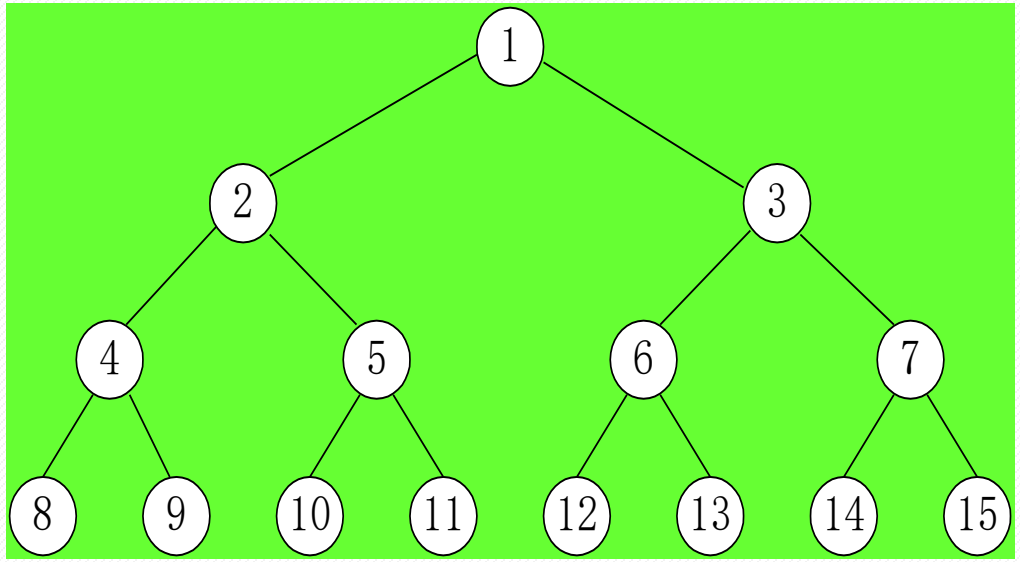


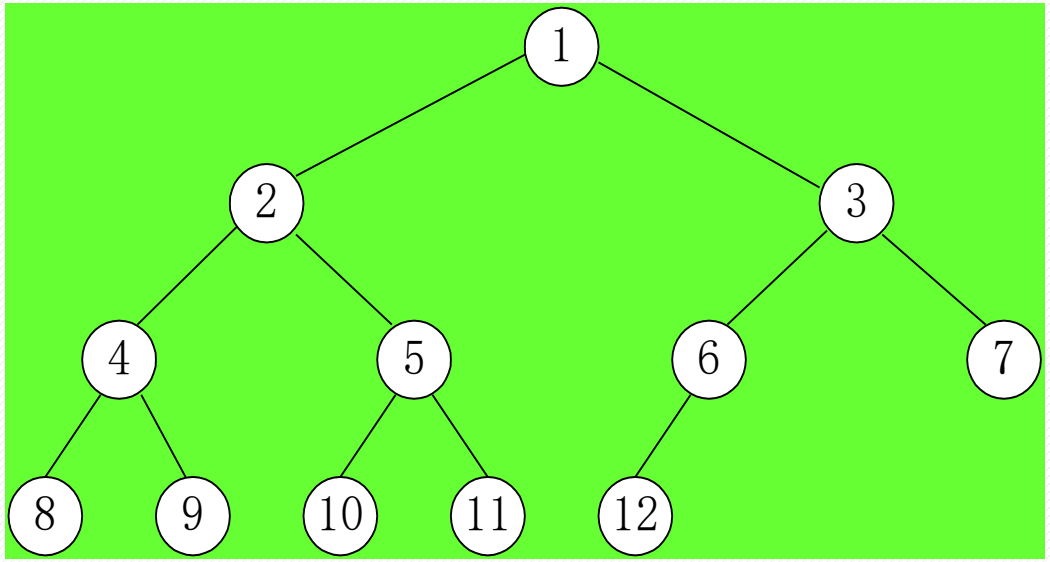
完全二叉树：深度为k，有n个结点，当且仅当其每一个结点都与深度为k的满二叉树中编号从1至n的结点一一对应

满二叉树和完全二叉树的区别

满二叉树是叶子一个也不少的树，而完全二叉树虽然前 n-1 层是满的，但最底层却允许在右边缺少连续若干个结点。

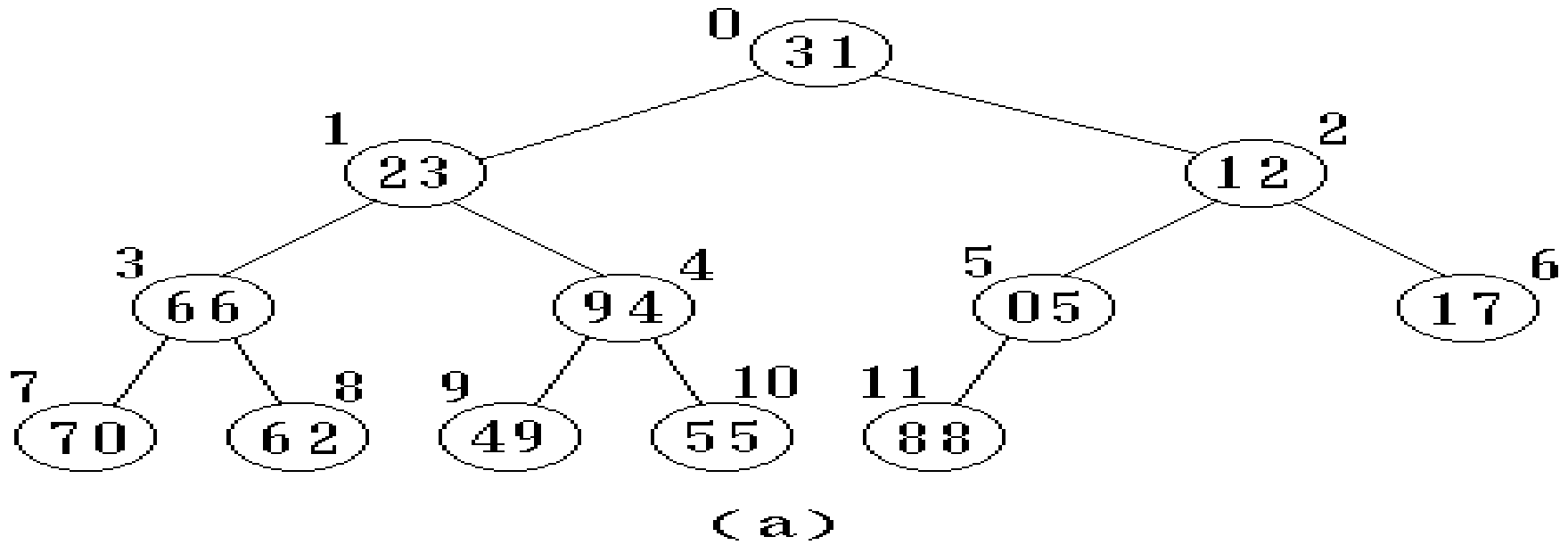
满二叉树是完全二叉树的一个特例。





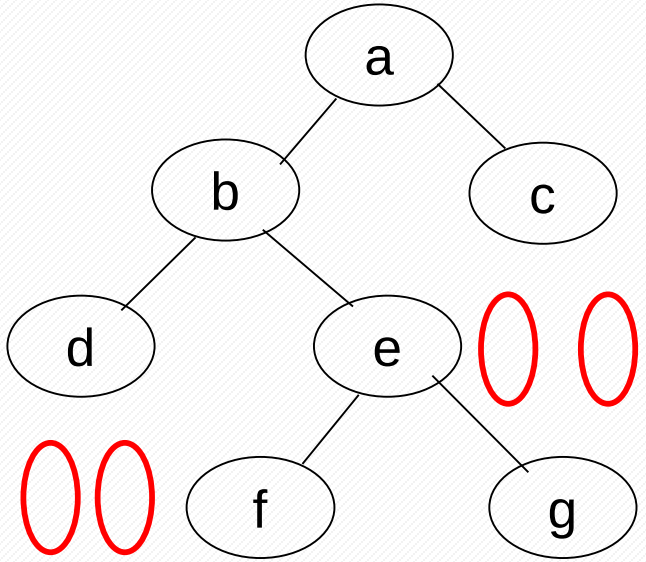
二叉树的顺序存储

实现：按满二叉树的结点层次编号，依次存放二叉树中的数据元素。

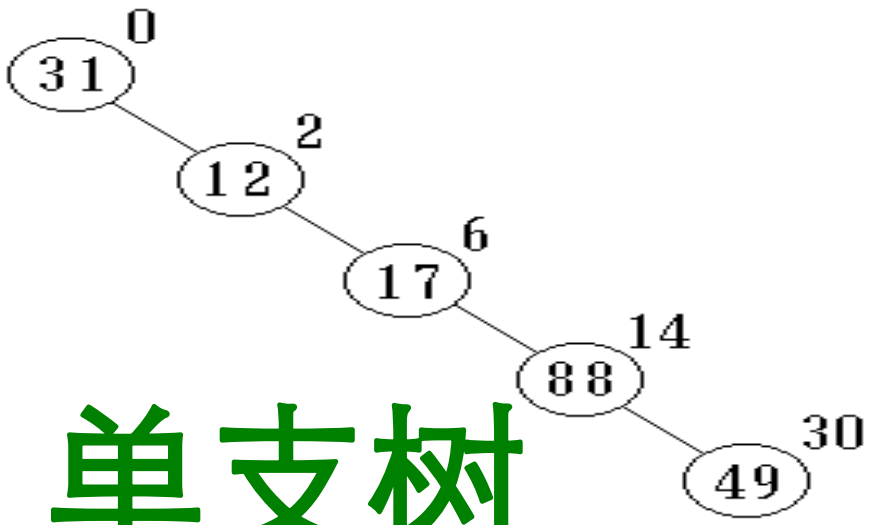


二叉树的顺序存储

0	1	2	3	4	5	6	7	8	9	10
a	b	c	d	e	0	0	0	0	f	g



注：二叉树的顺序存储只适用于完全二叉树



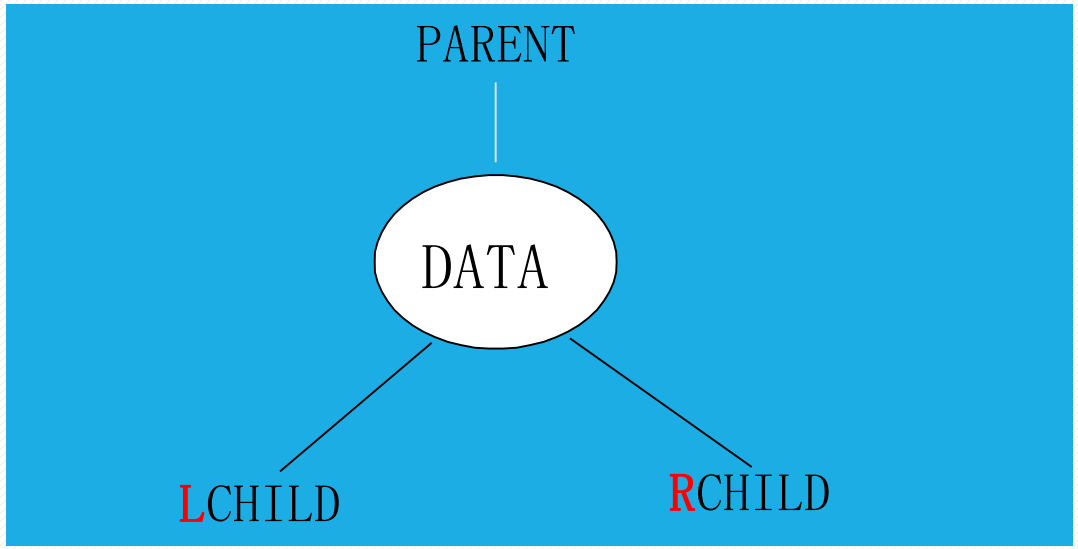
单支树

特点：

结点间关系蕴含在其存储位置中

浪费空间，适于存满二叉树和完全二叉树

二叉树的链式存储

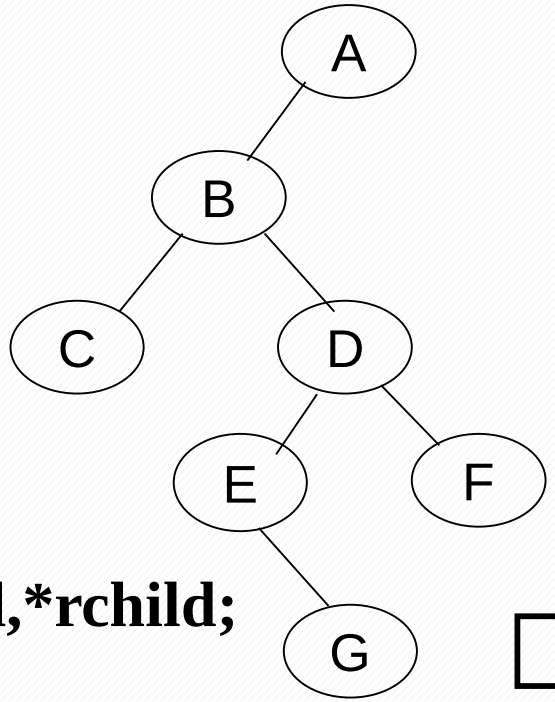


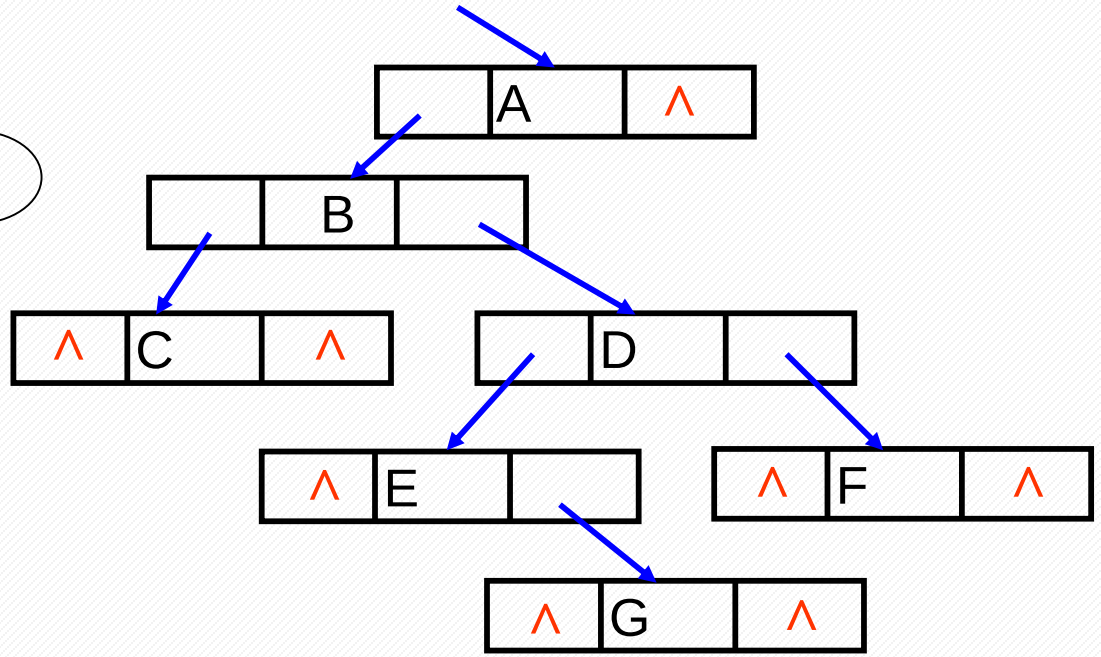
lchild	data	rchild

lchild	data	parent	rchild

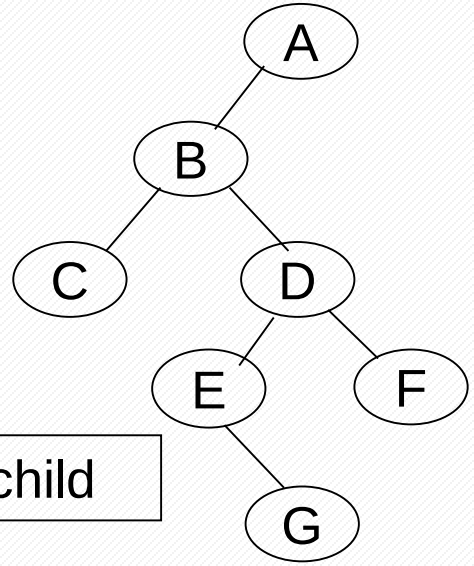
二叉链表


```
typedef struct BiNode
{
        TElemType   data;
        struct  BiNode *lchild,*rchild;
        //左右孩子指针
}BiNode,*BiTree;
```

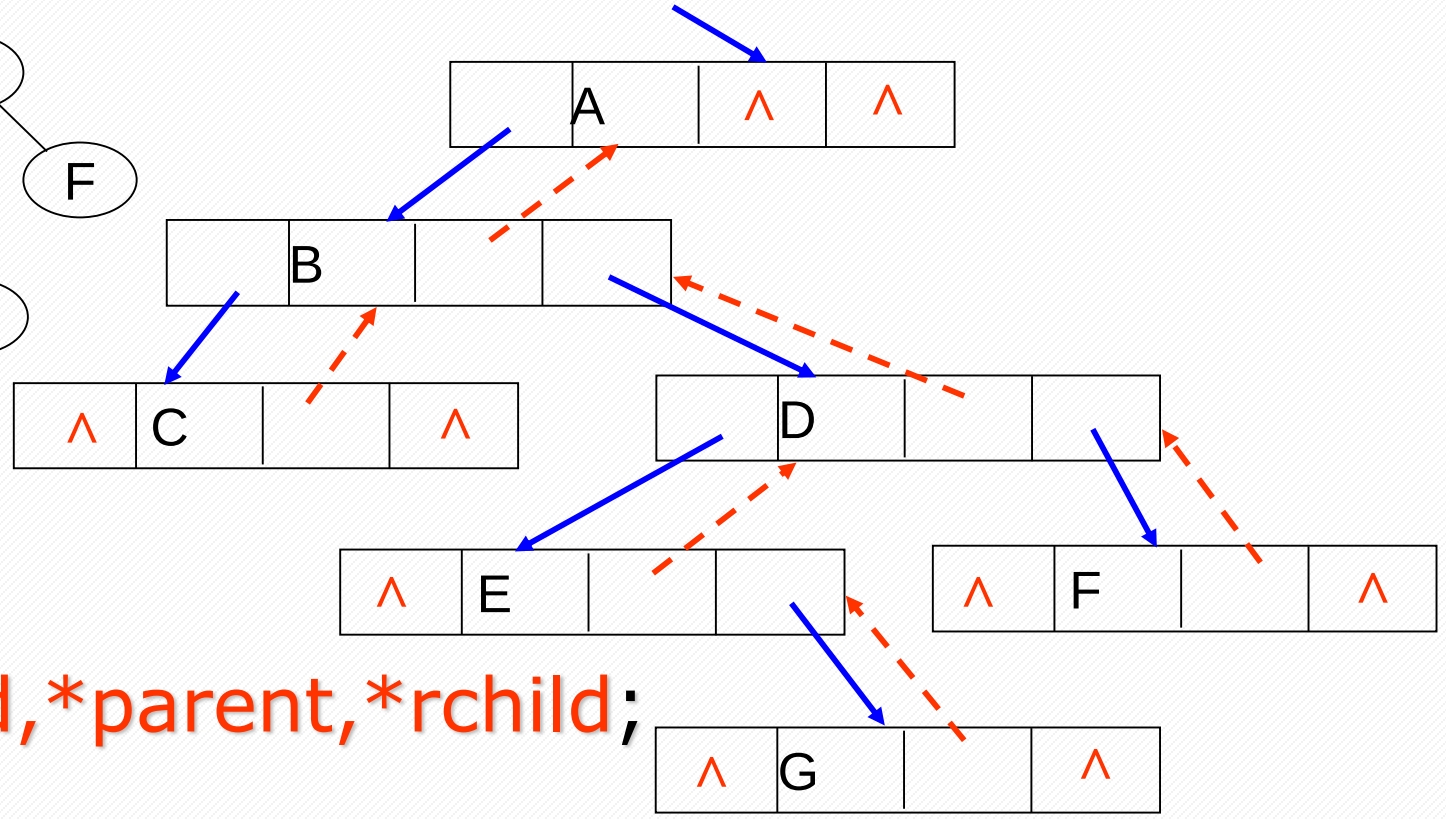




三叉链表



lchild	data	parent	rchild




```
typedef struct TriTNode
{
    TelemType data;
    struct TriTNode *lchild,*parent,*rchild;
} TriTNode,*TriTree;
```

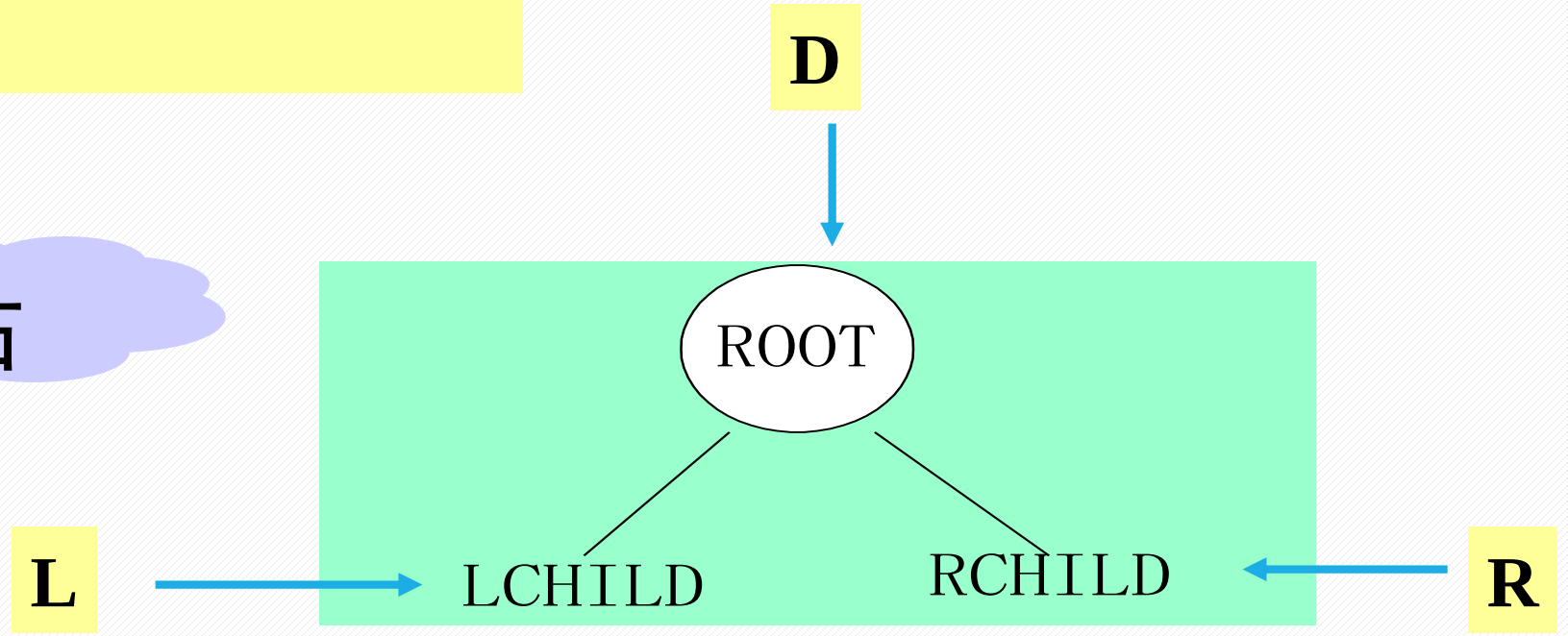
7.5 遍历二叉树

遍历定义——指按某条搜索路线遍访每个结点且不重复（又称周游）。

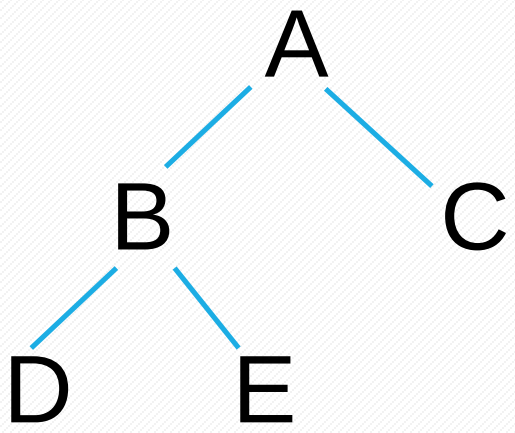
遍历用途——它是树结构插入、删除、修改、查找和排序运算的前提，是二叉树一切运算的基础和核心。

遍历规则

先左后右



DLR **LDR** **LRD** **DRL** **RDL** **RLD**



先序遍历：A B D E C

中序遍历：D B E A C

后序遍历：D E B C A

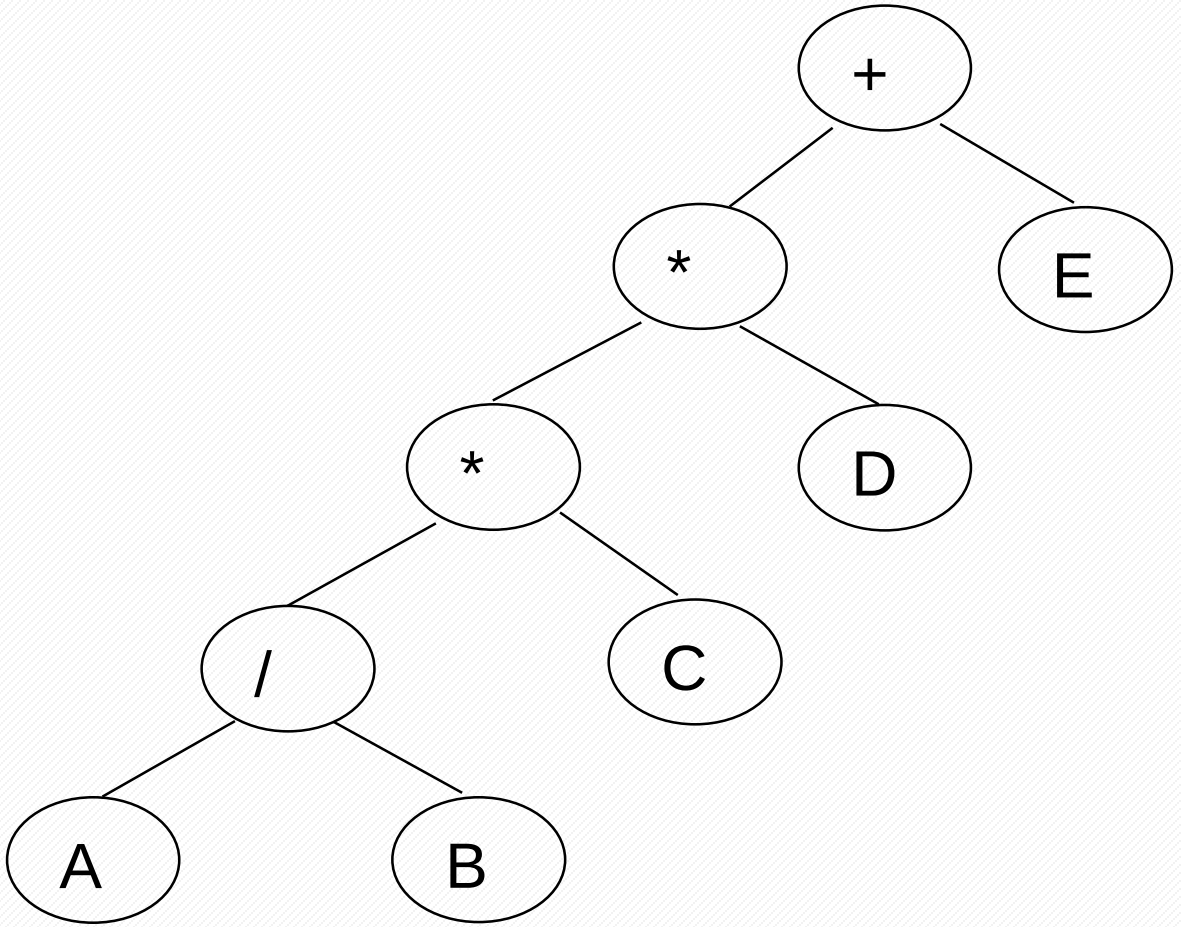
口诀：

DLR—先序遍历，即先根再左再右　　(先序：根、左、右)

LDR—中序遍历，即先左再根再右　　(中序：左、根、右)

LRD—后序遍历，即先左再右再根　　(后序：左、右、根)

用二叉树表示算术表达式



先序遍历DLR
+ * * / A B C D E
前缀表示

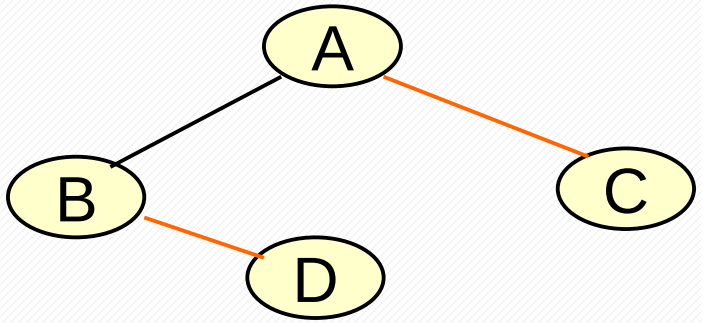
中序遍历LDR
A / B * C * D + E
中缀表示

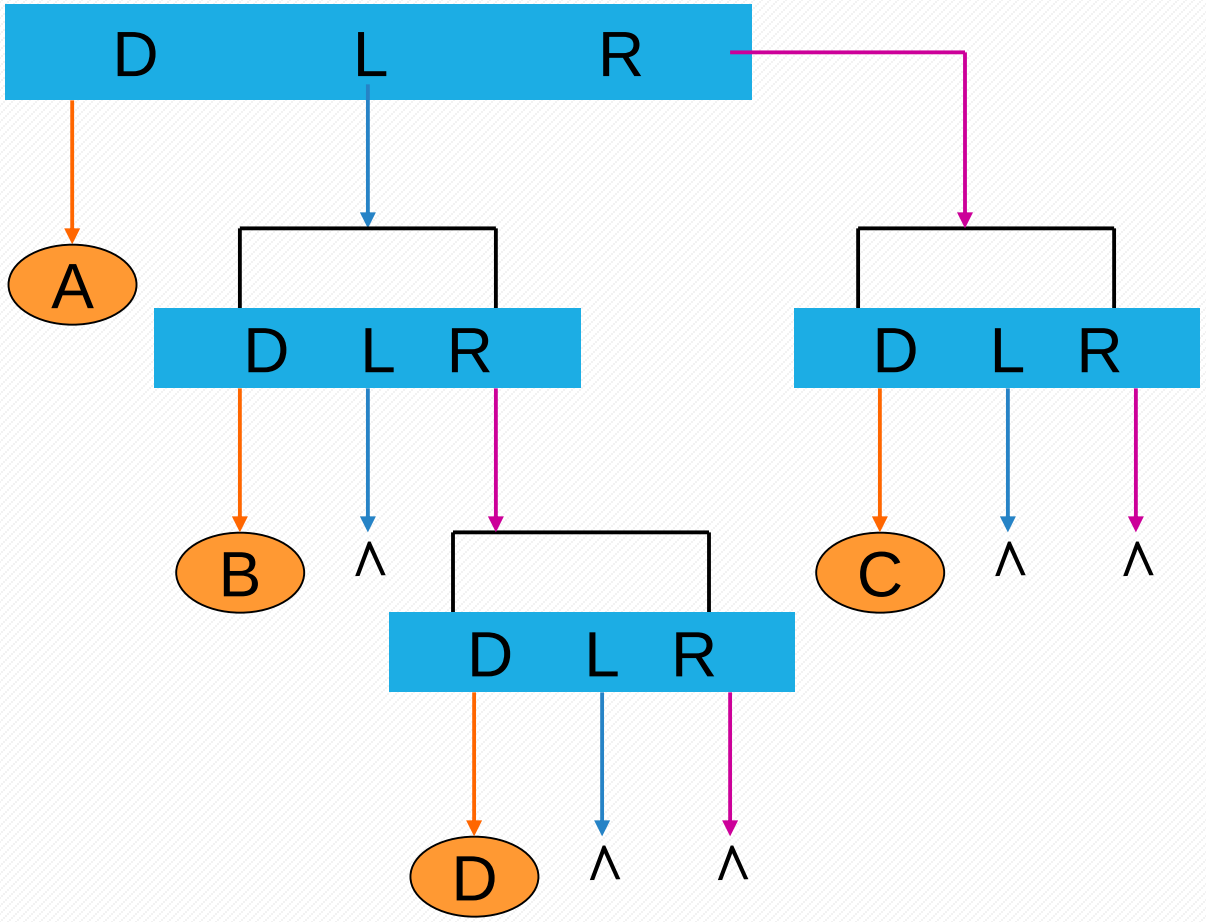
后序遍历LRD
A B / C * D * E +
后缀表示

层序遍历
+ * E * D / C A B

遍历的算法实现 - 先序遍历

若二叉树为空，则空操作
否则
　访问根结点 (D)
　前序遍历左子树 (L)
　前序遍历右子树 (R)



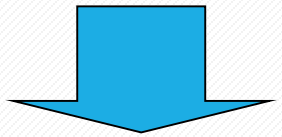


先序遍历序列：A B D C

遍历的算法实现 - 用递归形式格外简单!

```
long Factorial ( long n )
{
        if ( n == 0 )
                return 1;                       // 基本项
        else
                return n * Factorial (n-1);     // 归纳项
}
```

则三种遍历算法可写出:

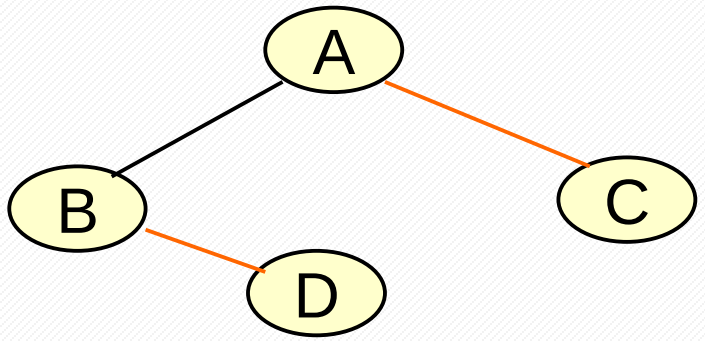


先序遍历算法

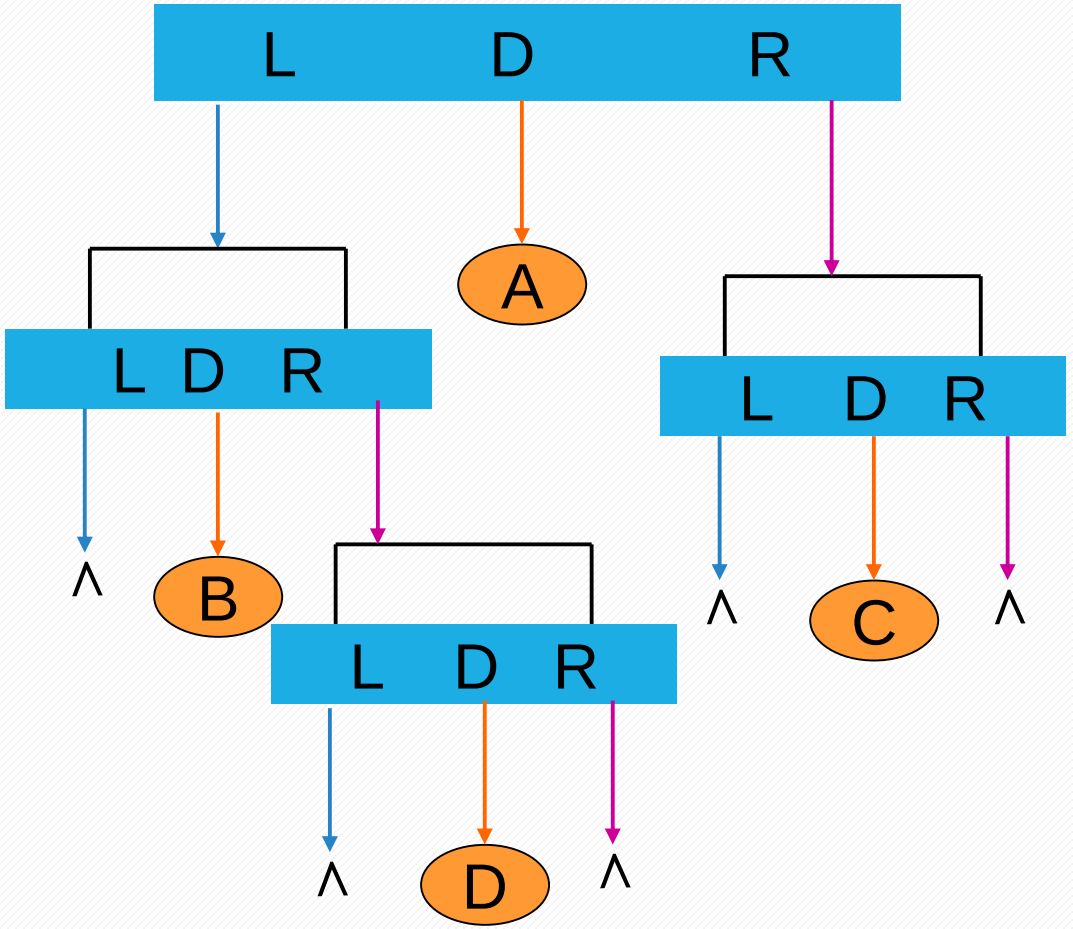
```
Status PreOrderTraverse(BiTree T)
{
        if(T==NULL)
                return OK;                          // 空二叉树
        else
        {
                cout<< T->data;                     // 访问根结点
                PreOrderTraverse(T->lchild);        // 递归遍历左子树
                PreOrderTraverse(T->rchild);        // 递归遍历右子树
        }
}
```

遍历的算法实现 - 中序遍历

若二叉树为空，则空操作
否则:
　　中序遍历左子树 (L)
　　访问根结点 (D)
　　中序遍历右子树 (R)



中序遍历序列：B　D　A　C

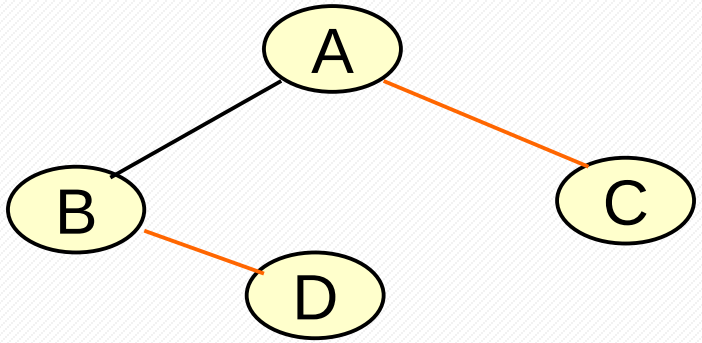


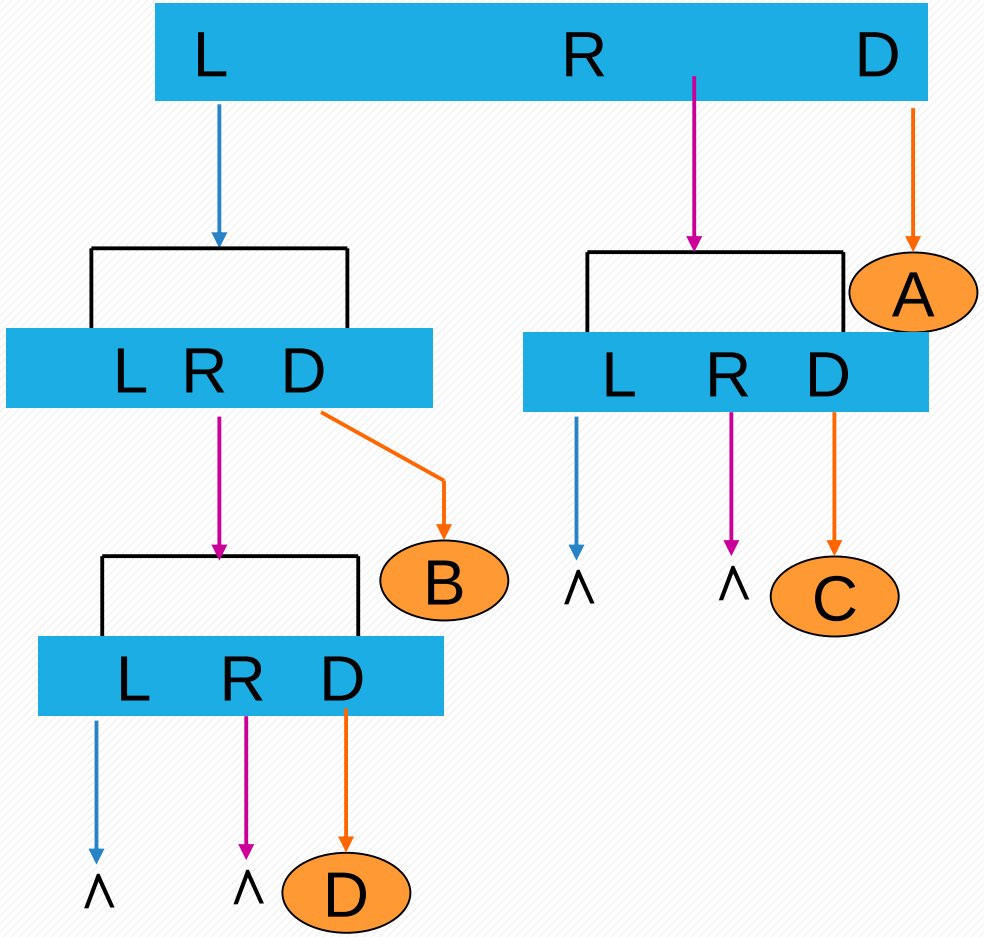
中序遍历算法

```
Status InOrderTraverse(BiTree T)
{
  if(T==NULL) return OK;                    //空二叉树
  else{
    InOrderTraverse(T->lchild);             //递归遍历左子树
  cout<<T->data;                          //访问根结点
    InOrderTraverse(T->rchild);             //递归遍历右子树
   }
}
```

遍历的算法实现 - 后序遍历

若二叉树为空，则空操作
否则
　后序遍历左子树 (L)
　后序遍历右子树 (R)
　访问根结点 (D)





后序遍历序列：　D　B　C　A

后序遍历算法

```
Status PostOrderTraverse(BiTree T)
{
        if(T==NULL)
                return OK;                          // 空二叉树
        else
        {
                PostOrderTraverse(T->lchild);       // 递归遍历左子树
                PostOrderTraverse(T->rchild);       // 递归遍历右子树
                cout<<T->data;                      // 访问根结点
        }
}
```

CimFAX
传真服务器常用
问答

将来的你

冲刺 加油!

让我们全力以赴

一定会感激现在拼命的自己