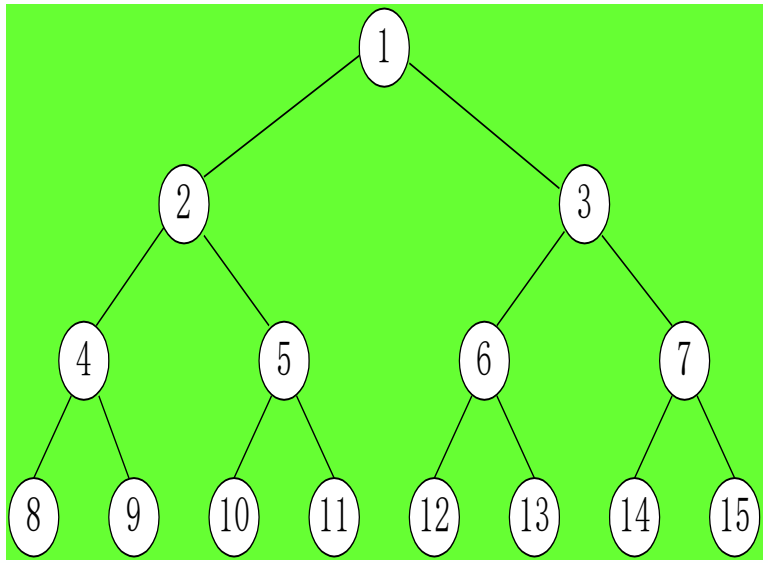
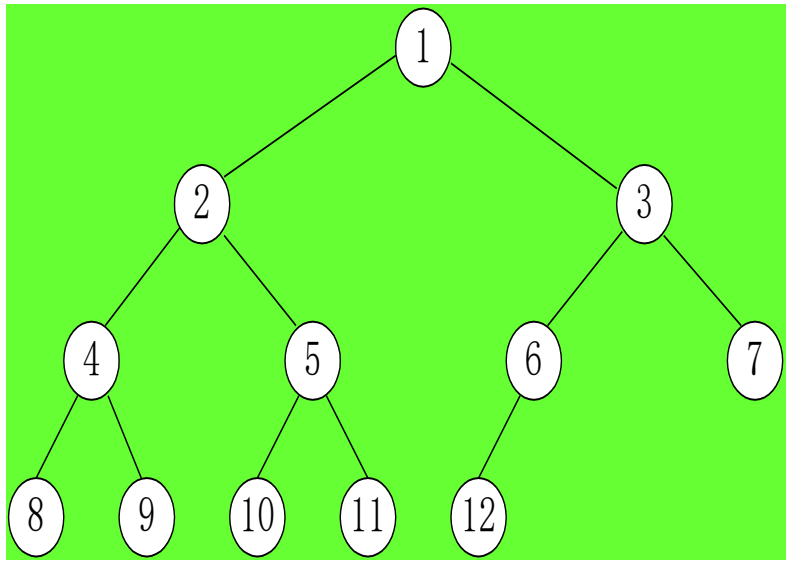
# 满二叉树和完全二叉树的区别

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。**

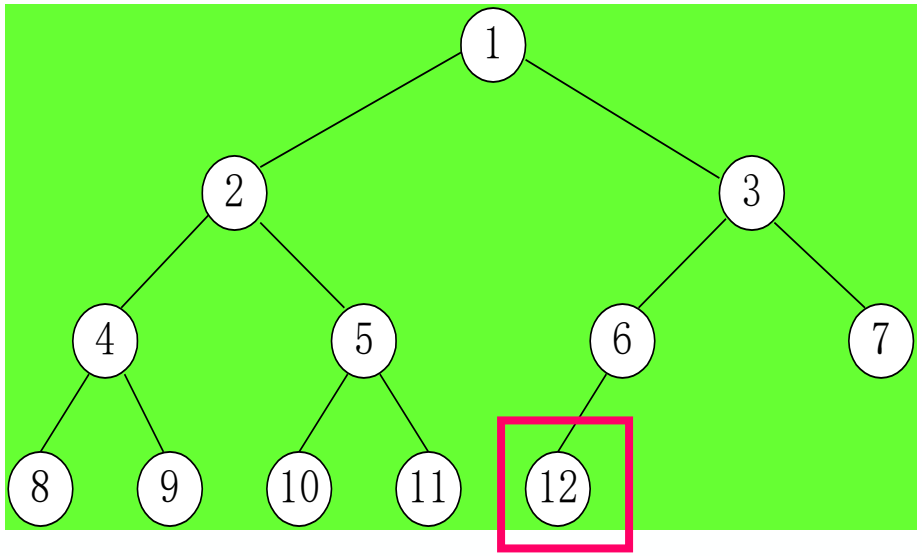


# 练习

一棵完全二叉树有5000个结点，可以计算出其叶结点的个数是（ 2500

**性质4：具有n个结点的完全二叉树的深度必为$[\log_2 n]+1$**




k-1层 $2^{k-1}-1$

k层 $2^{k}-1$

$2^{k-1}-1 < n \leq 2^{k}-1$ 或 $2^{k-1} \leq n < 2^{k}$

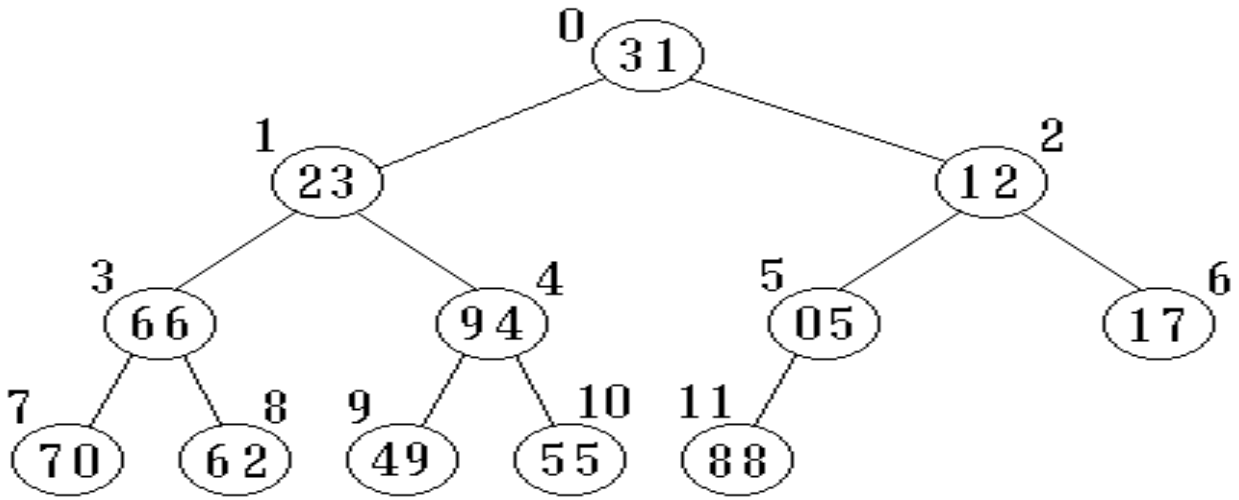
$k-1 \leq \log_2 n < k$，因为$k$是整数

所以$k = \lfloor \log_2 n \rfloor + 1$

*性质5：*对完全二叉树，若从上至下、从左至右编号，则编号为i 的结点，其左孩子编号必为2i，其右孩子编号必为2i＋1；其双亲的编号必为i/2。

# 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。**

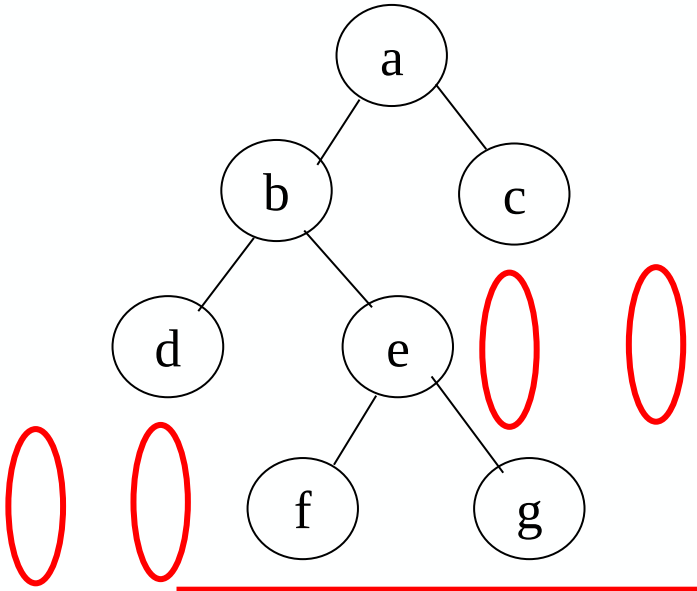


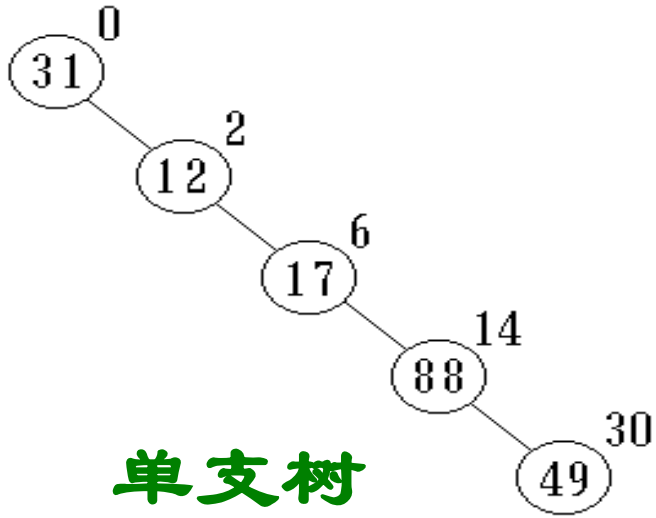

(a)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 23 | 12 | 66 | 94 | 05 | 17 | 70 | 62 | 49 | 55 | 88 |

# 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 |



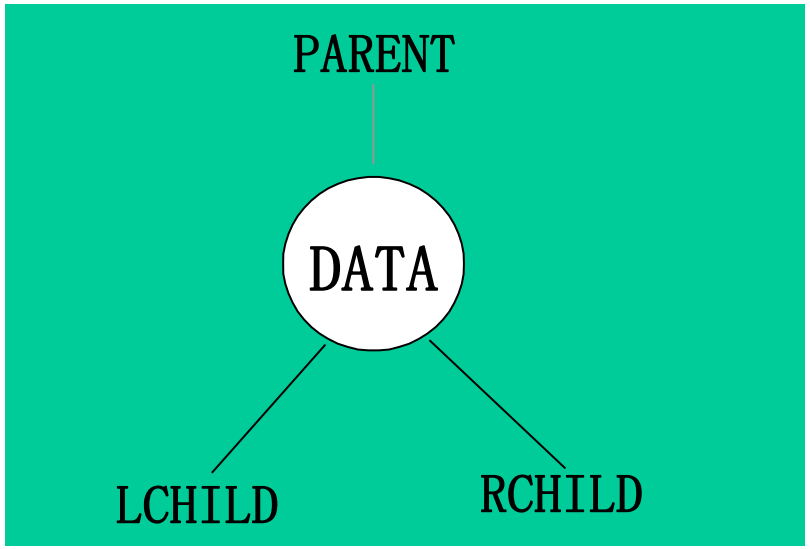

单支树

**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**

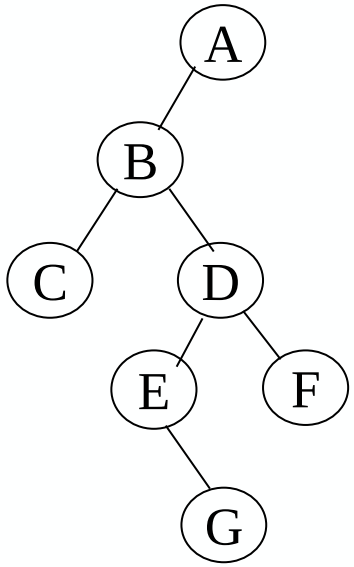
# 二叉树的链式存储

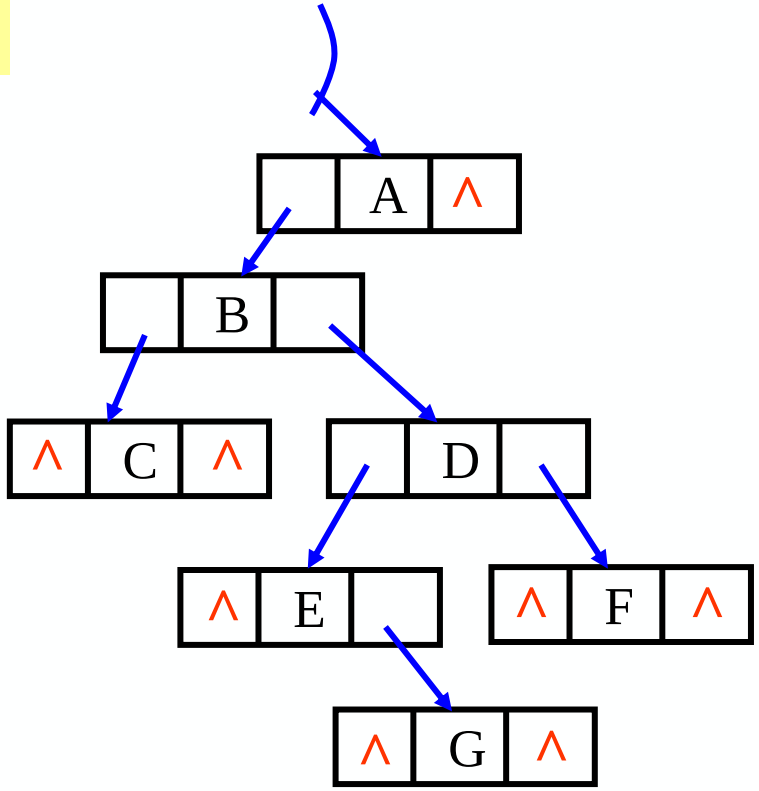



| lchild | data | rchild |
|---|---|---|

| lchild | data | parent | rchild |
|---|---|---|---|

# 二叉链表



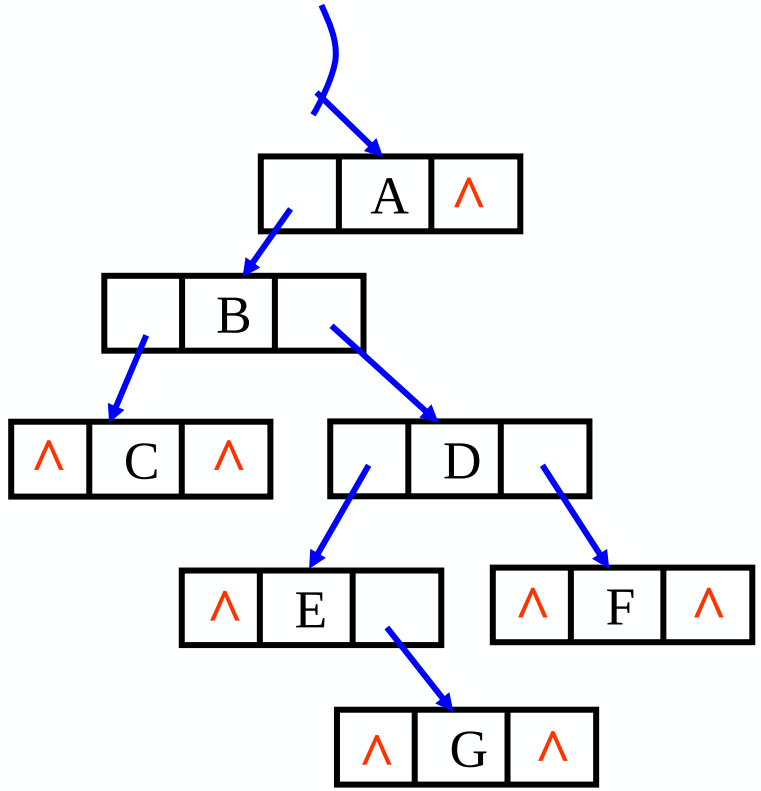

```
typedef struct BiNode{
   TElemType   data;
   struct  BiNode   *lchild,*rchild; //左右孩子指针
}BiNode,*BiTree;
```

# 练习

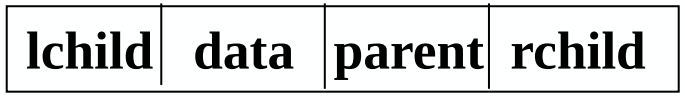
## 在n个结点的二叉链表中，有___n+1___个空指针域

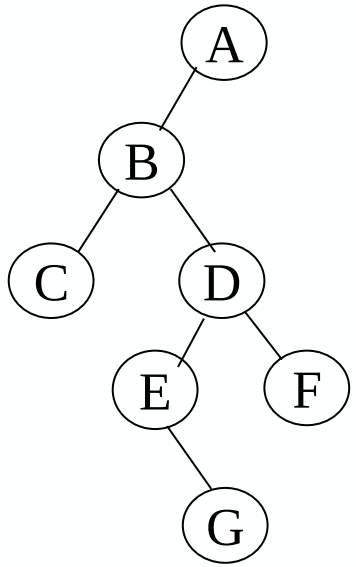
分析：必有2n个链域。除根结点外，每个结点有且仅有一个双亲，所以只会有n - 1个结点的链域存放指针，指向非空子女结点。

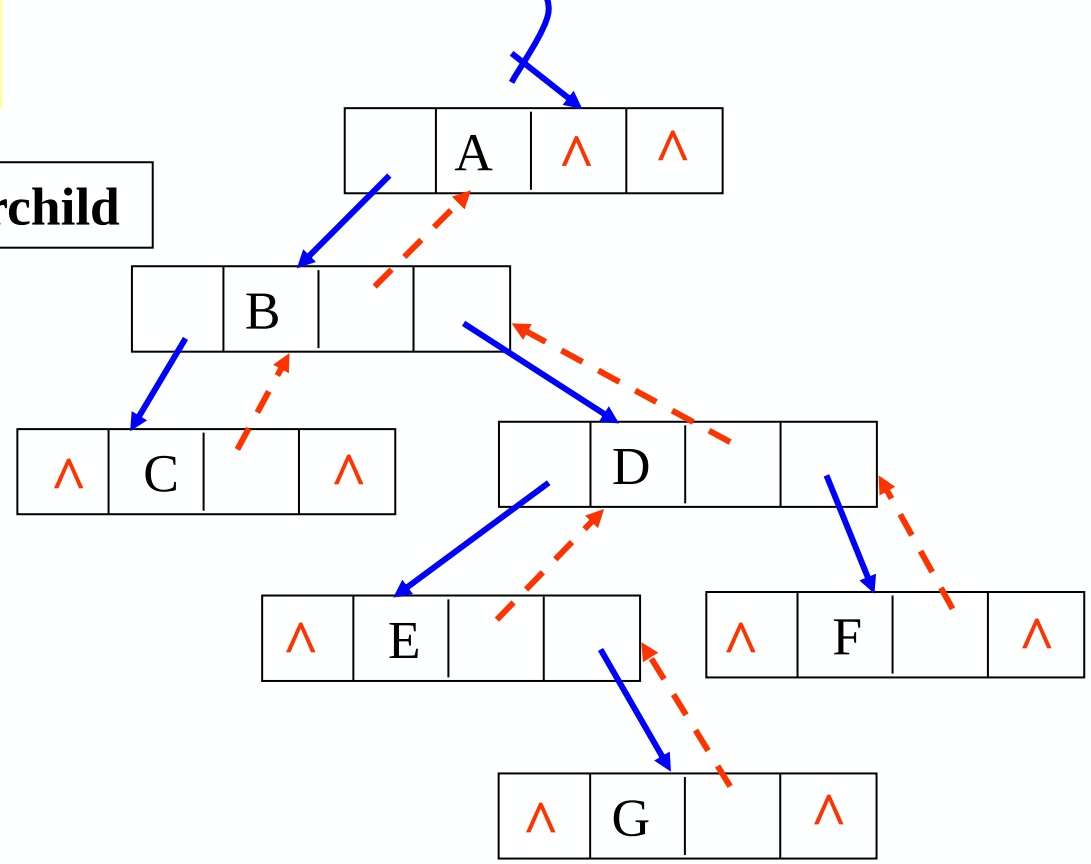
空指针数目 = 2n - (n-1)=n+1



# 三叉链表



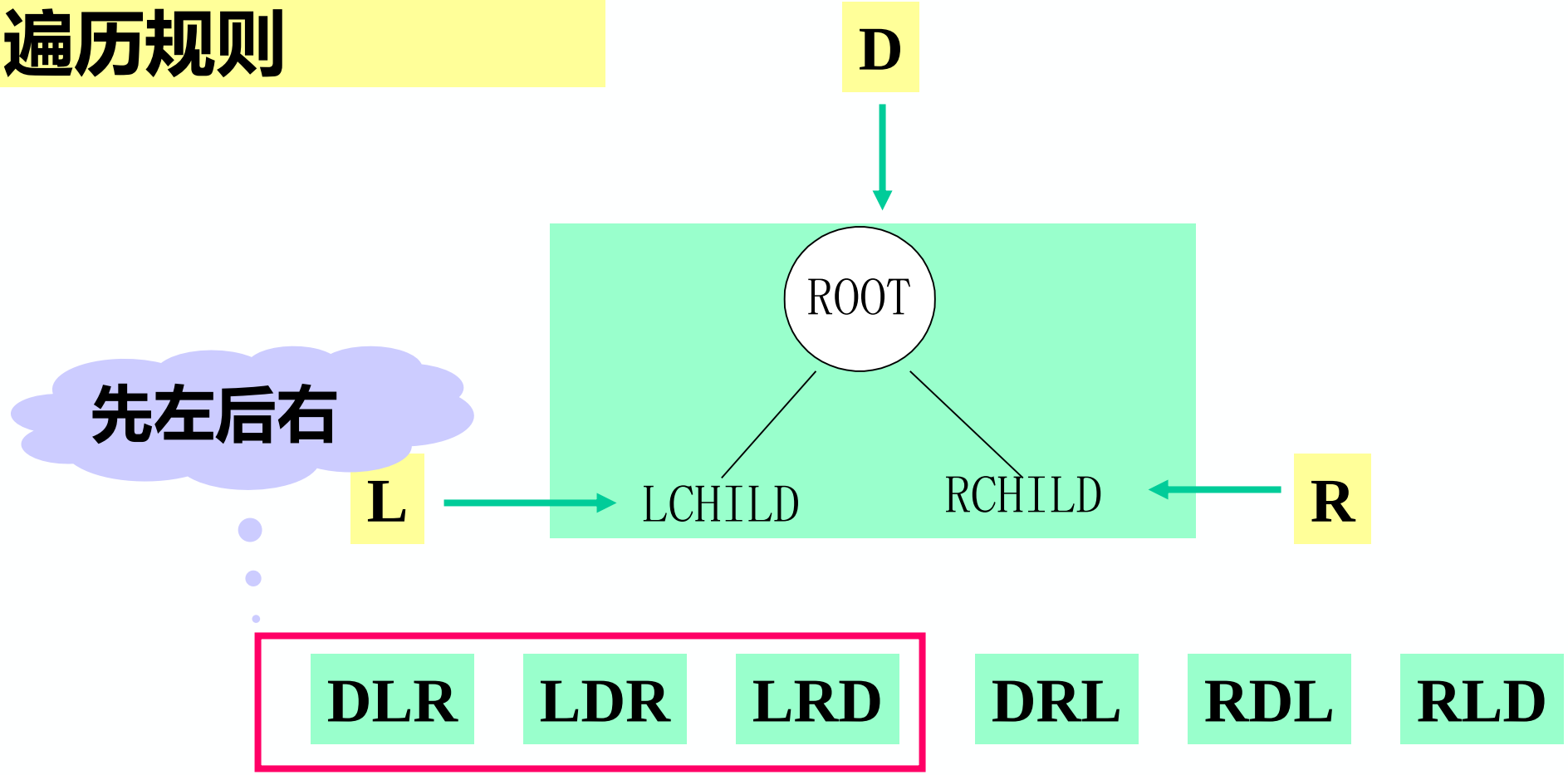

```
typedef struct TriTNode
{  TelemType data;
   struct TriTNode *lchild,*parent,*rchild;
 }TriTNode,*TriTree;
```

# 5.5 遍历二叉树和线索二叉树

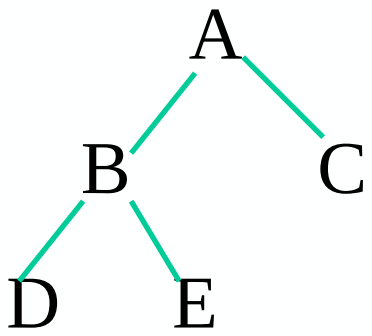
**遍历定义**——指按某条搜索路线遍访每个结点且不重复（又称周游）。

**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




**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**

**LDR—中序遍历，即先左再根再右**

**LRD—后序遍历，即先左再右再根**