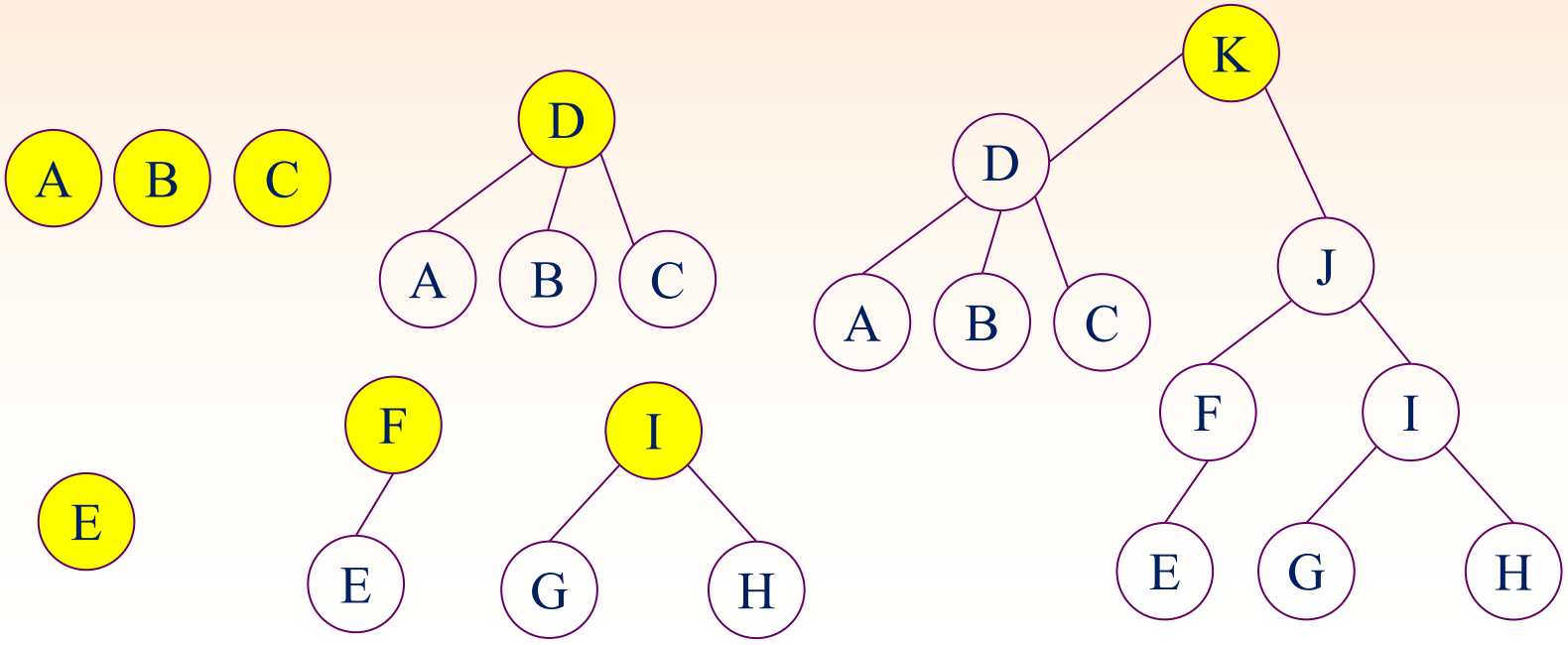
第六章 树和二叉树（1）

树的定义
二叉树的定义及存储结构
二叉树的遍历

6.1 树的定义

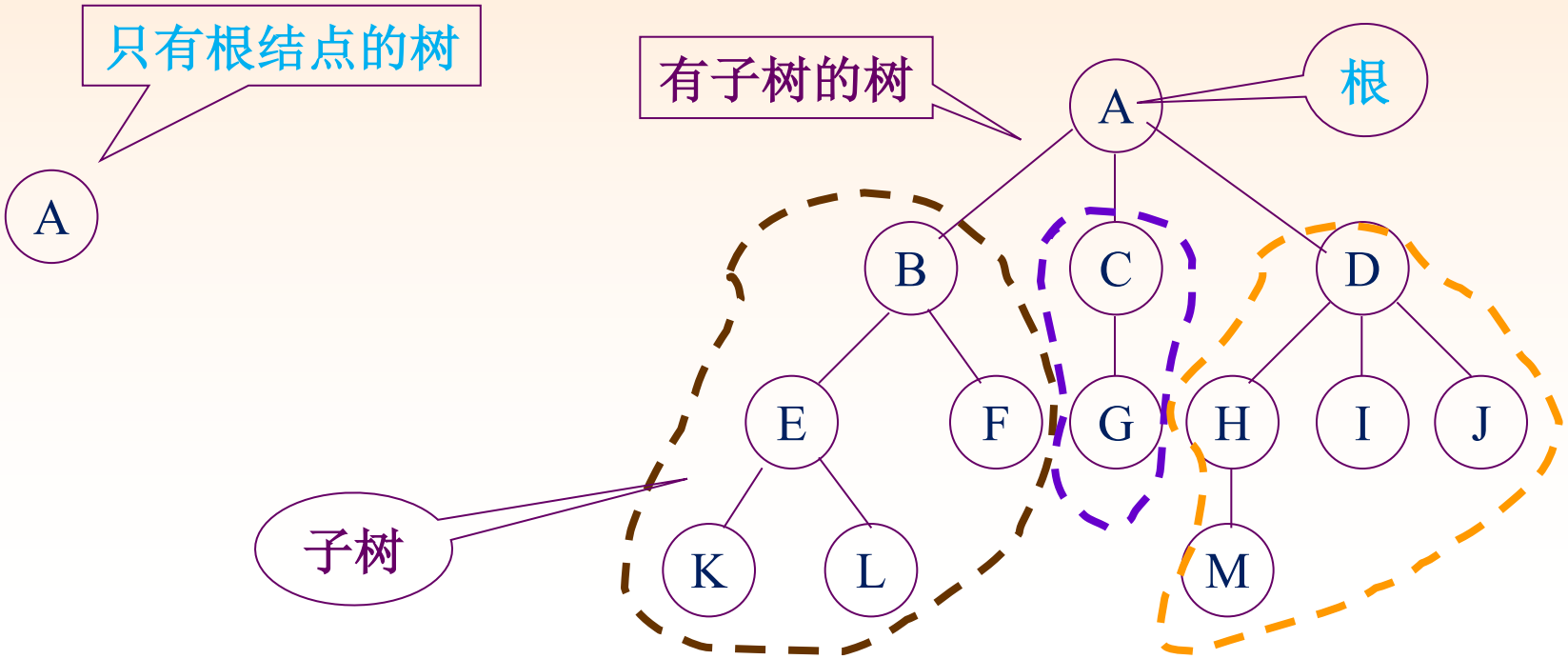
★定义：

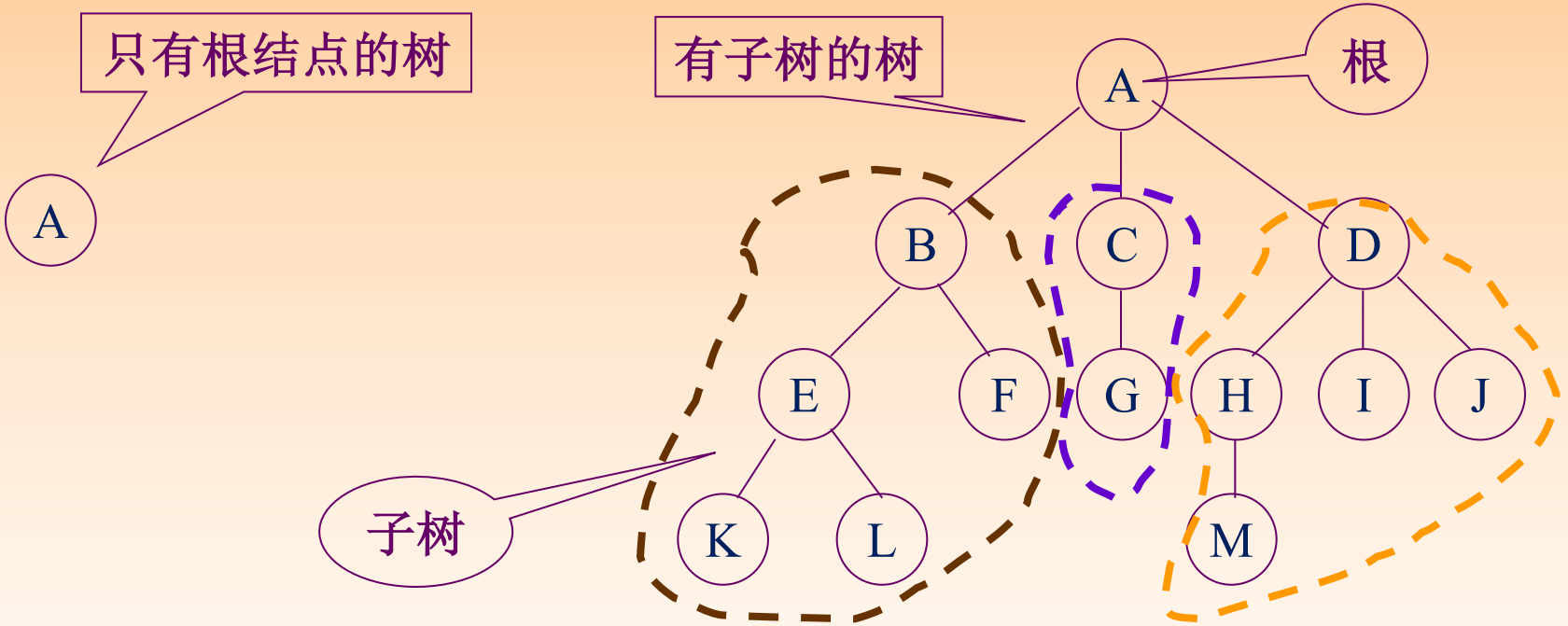
- 若n=1：一个单独的**结点**，称作一棵树。这个结点被称作这个树的根。
- 若$T_1,\ldots,T_m$是树($m\geq 1$)。将它们的根连接到另一个节点v，将组成一颗以v为根的树（含$|T_1|+\ldots+|T_m|+1$结点）。
- 若n=0，则称为空树。



§ 树(tree)的递归定义（有限集的层次划分）

- ★若n=0，则称为空树，
- ★若n>0：
  - ❖有一个特定的结点称为树的根(root)
  - ❖当n>1时，其结点可分为m(m>0)个互不相交的有限集$T_1,\ldots,T_m$，每一个本身又是一棵树，称为根的子树(subtree余)





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

根 T_1 T_2 T_3

树用分支来刻画一种层次结构

树是一类重要的非线性数据结构
后续许多数据结构是基于树的！

另外，
线性结构属于特殊的树性结构。

基本术语

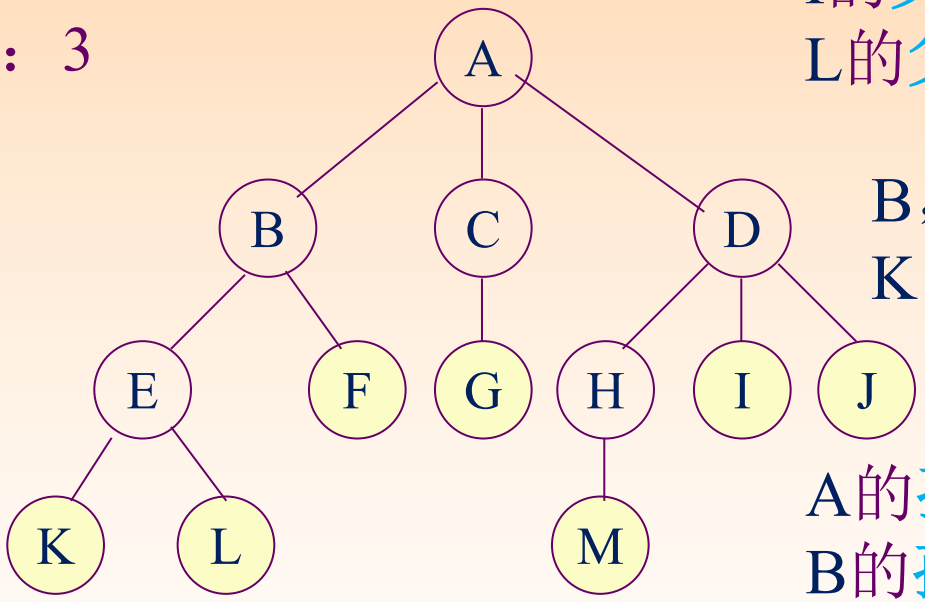
- 结点(node)——表示树中的元素，包括数据项及若干指向其子树的分支
- 结点的度(degree)——结点拥有的子树数
- 叶子(leaf)——度为0的结点
- 树的度(degree)——一棵树中最大的结点度数
- 结点的层次(level)——从根结点算起，根为第一层，它的孩子为第二层……
- 深度(depth)——树中结点的最大层次数
- 孩子(child)——结点子树的根称为该结点的孩子
- 父亲(parent)——孩子结点的上层结点
- 兄弟(brother)——同一父亲的孩子

A的度：3
B的度：2
M的度：0

叶子：K，L，F，G，M，I，J

I的父亲：D
L的父亲：E

树的度：3



B，C，D为兄弟
K，L为兄弟

A的层次：1
M的层次：4

A的孩子：B，C，D
B的孩子：E，F

树的深度：4

F，G为堂兄弟
F，G，M是A的子孙。
A是结点F，G，M的祖先

对比线性结构&树型结构的结构特点

线性结构

第一个数据元素
(无前驱)

最后一个数据元素
(无后继)

其它数据元素
(一个前驱、
一个后继)

树型结构

根结点
(无前驱)

多个叶子结点
(无后继)

其它数据元素
(一个前驱、
多个后继)

基本术语

★有序树——子树之间存在确定的次序

★无序树——子树之间不存在确定的次序

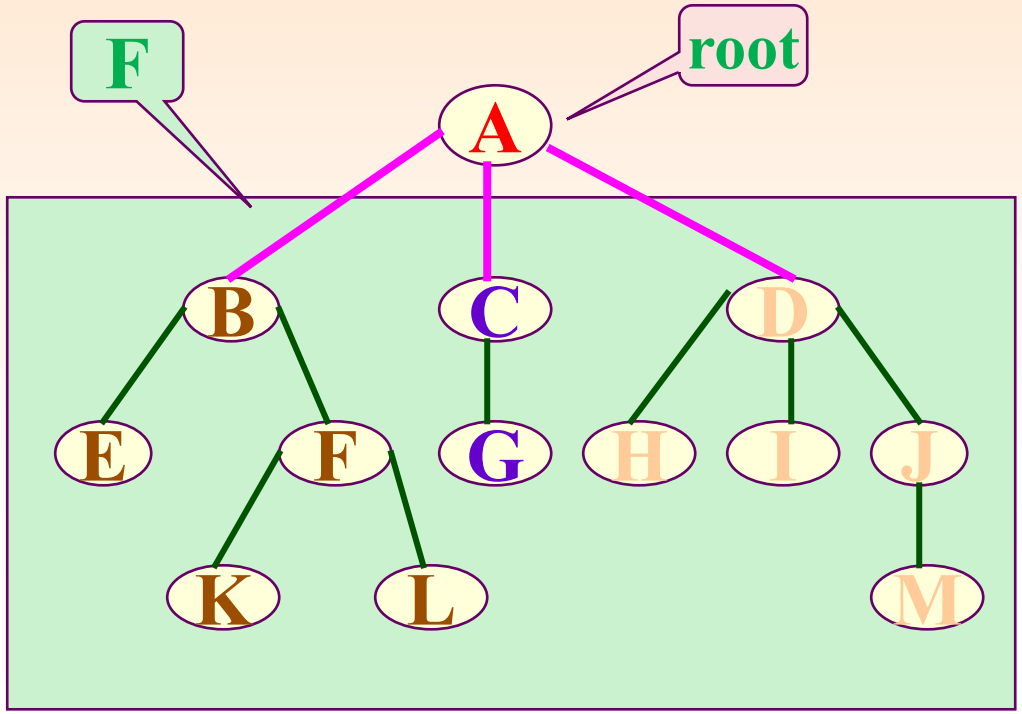
★森林(forest)——m(m≥0)棵互不相交的树的集合

非空树是一个二元组

Tree = （ root，F ）

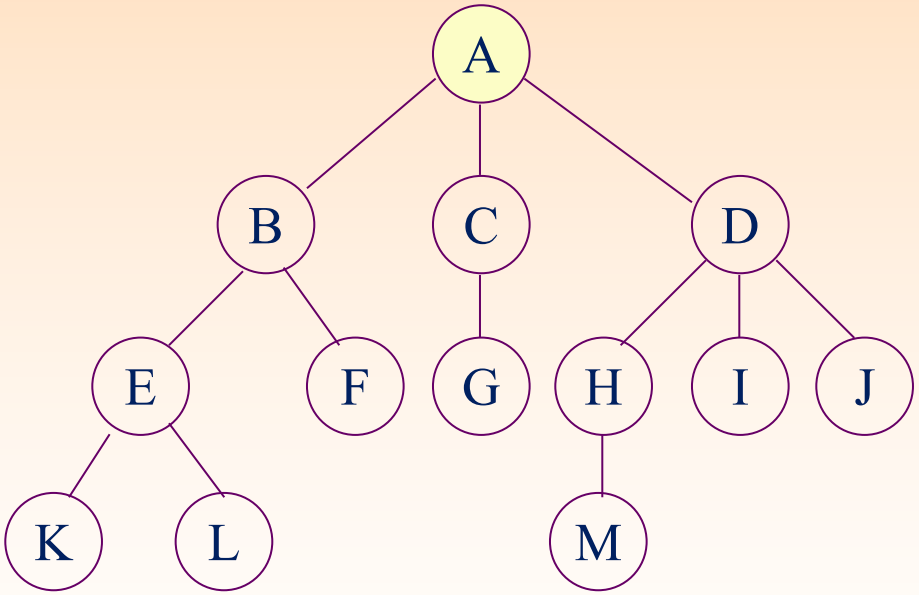
root 被称为根结点。

F 被称为子树森林



基本性质：

树的分支个数 = 树的结点个数-1



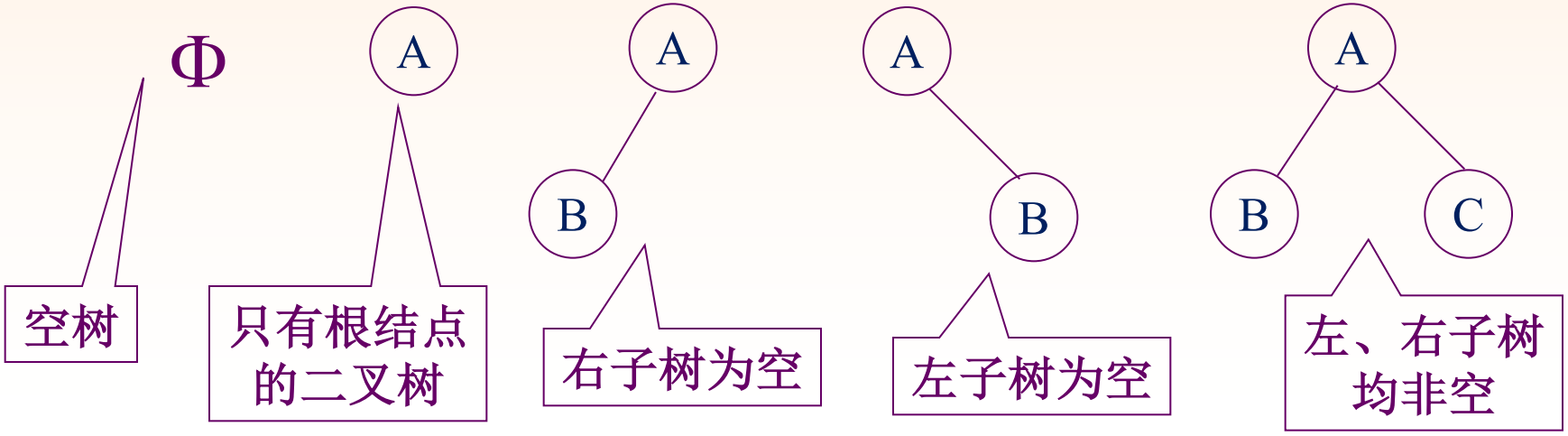
6.2 二叉树

★定义：二叉树是n(n≥0)个结点的有限集，它或为空树(n=0)，或由一个根结点和两棵分别称为左子树和右子树的互不相交的二叉树构成

★特点

- ❖每个结点至多有两棵子树(即，没有度大于2的结点)
- ❖二叉树的子树有左右之分，其次序不能颠倒

★五种基本形态



二叉树性质

性质1：第i层至多有2^{i-1}个节点（i≥1)。

证明：归纳法.

当i=1时，显然正确。

假设第i-1层至多有2^{i-2} 个结点

第i层上最大结点数是第i-1层的2倍，即 $2^{i-2}\ 2=2^{i-1}$.

性质2：深度为k的二叉树至多有2^k-1个结点(k≥1)

证明：

$$\sum_{i=1}^{k}(\text{第}i\text{层的最大结点数}) = \sum_{i=1}^{k} 2^{i-1} = 2^k - 1$$

二叉树性质（continue）

性质3：对任何一棵二叉树T，如果叶子数目为n_0，度为2的结点数为n_2，则$n_0=n_2+1$

证明：

二叉树中除根结点外，其余结点只有一个分支进入。因此$n_0+ n_1+ n_2=B+1$，其中B为分支总数。

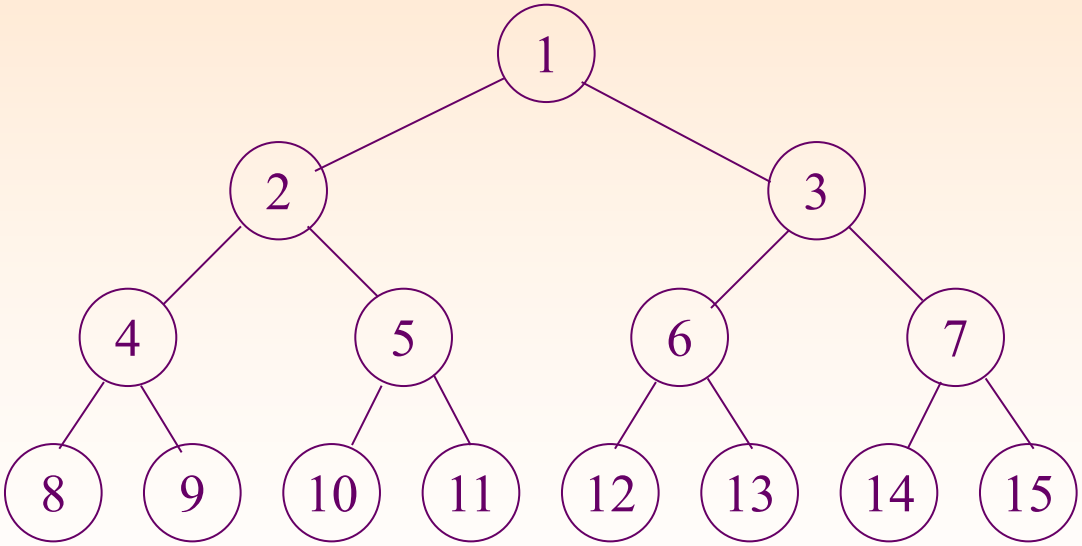
由于分支由度为2和1的结点射出，故$B=n_1 + 2n_2$。

综合以上，$n_0 + n_1 +n_2=2n_2 + n_1 +1$。故$n_0=n_2+1$。

几种特殊形式的二叉树

满二叉树

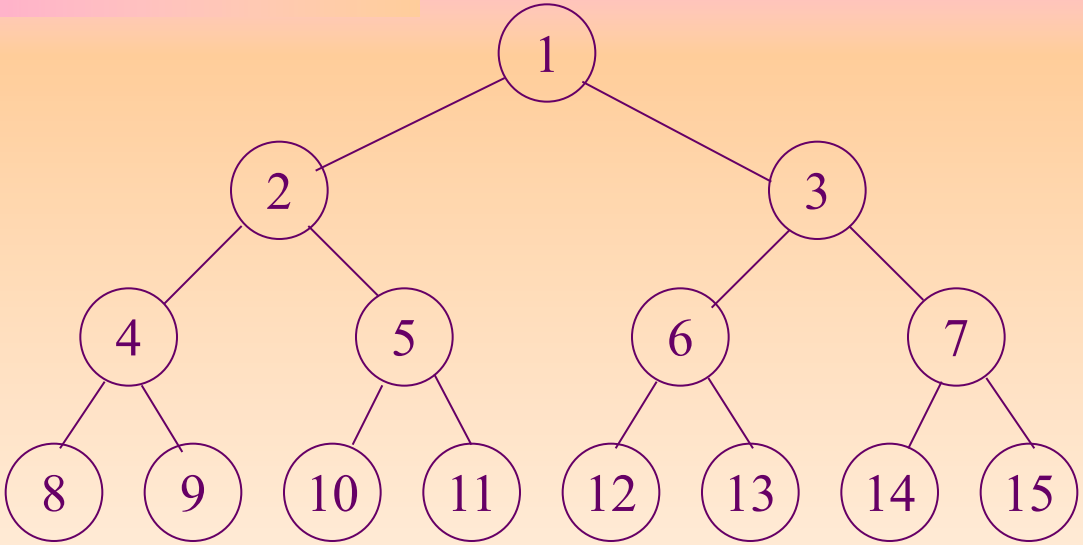
若二叉树深度为k而节点数为2^k-1，它是满的。

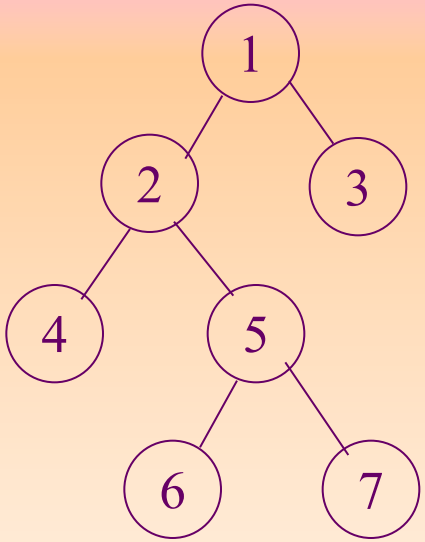


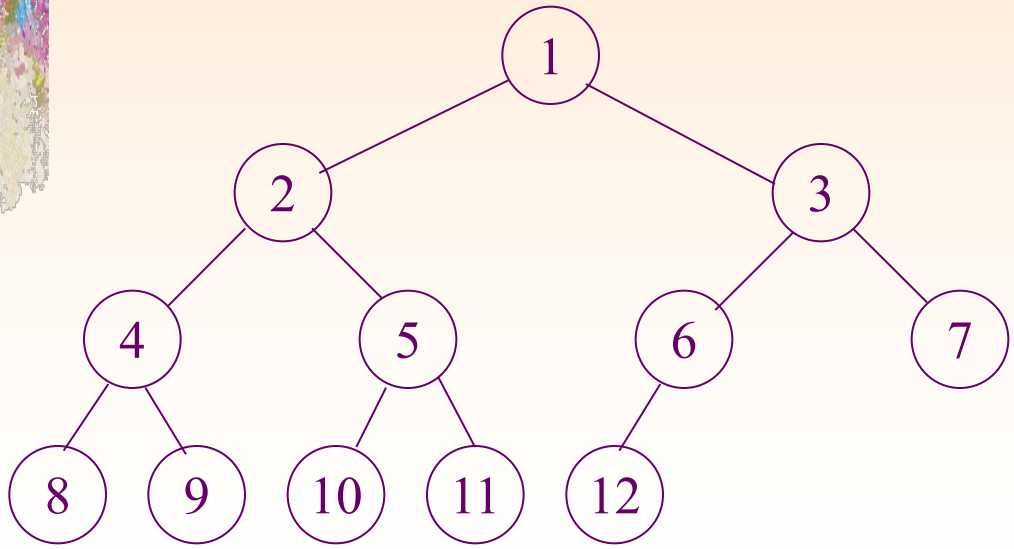
几种特殊形式的二叉树

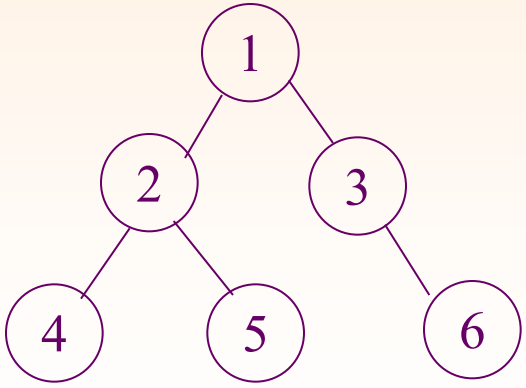
完全二叉树

- 深度为k，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，称为完全二叉树。
- 另一种直观定义：将满二叉树的最右边的若干个叶子节点删去后得到的树叫做完全二叉树。
- 特点
  - 叶子结点只可能在层次最大的两层上出现
  - 对任一结点，若其右分支下子孙的最大层次为L，则其左分支下子孙的最大层次必为L或L+1
  - 满二叉树也是完全二叉树。


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


1
2
3
4
5
6
7


1
2
3
4
5
6
7
8
9
10
11
12


1
2
3
4
5
6

二叉树性质（continue）

性质4：具有n个节点的完全二叉树的深度是

$$\lfloor \log_2 n \rfloor + 1$$

证明：

设完全二叉树的深度为 k

根据性质2， $2^{k-1} \leq n < 2^k$

即 $k-1 \leq \log_2 n < k$

因为 k 只能是整数，因此， $k = \lfloor \log_2 n \rfloor + 1$ 。

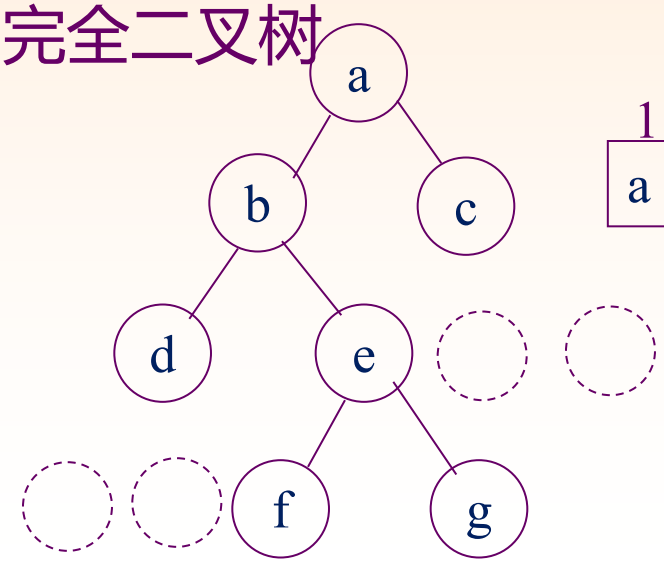
二叉树性质（continue）

- 性质5：如果对一棵有n个结点的完全二叉树的结点按层序编号，则对任一结点i (1≤i≤n)，有：
- (1) 如果i=1，则结点i是二叉树的根，无父亲；否则其父亲是⌊i/2⌋
- (2) 如果2i>n，则结点i无左孩子；否则则其左孩子是2i
- (3) 如果2i+1>n，则结点i无右孩子；否则其右孩子是2i+1

二叉树的顺序存储表示

- 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
- 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

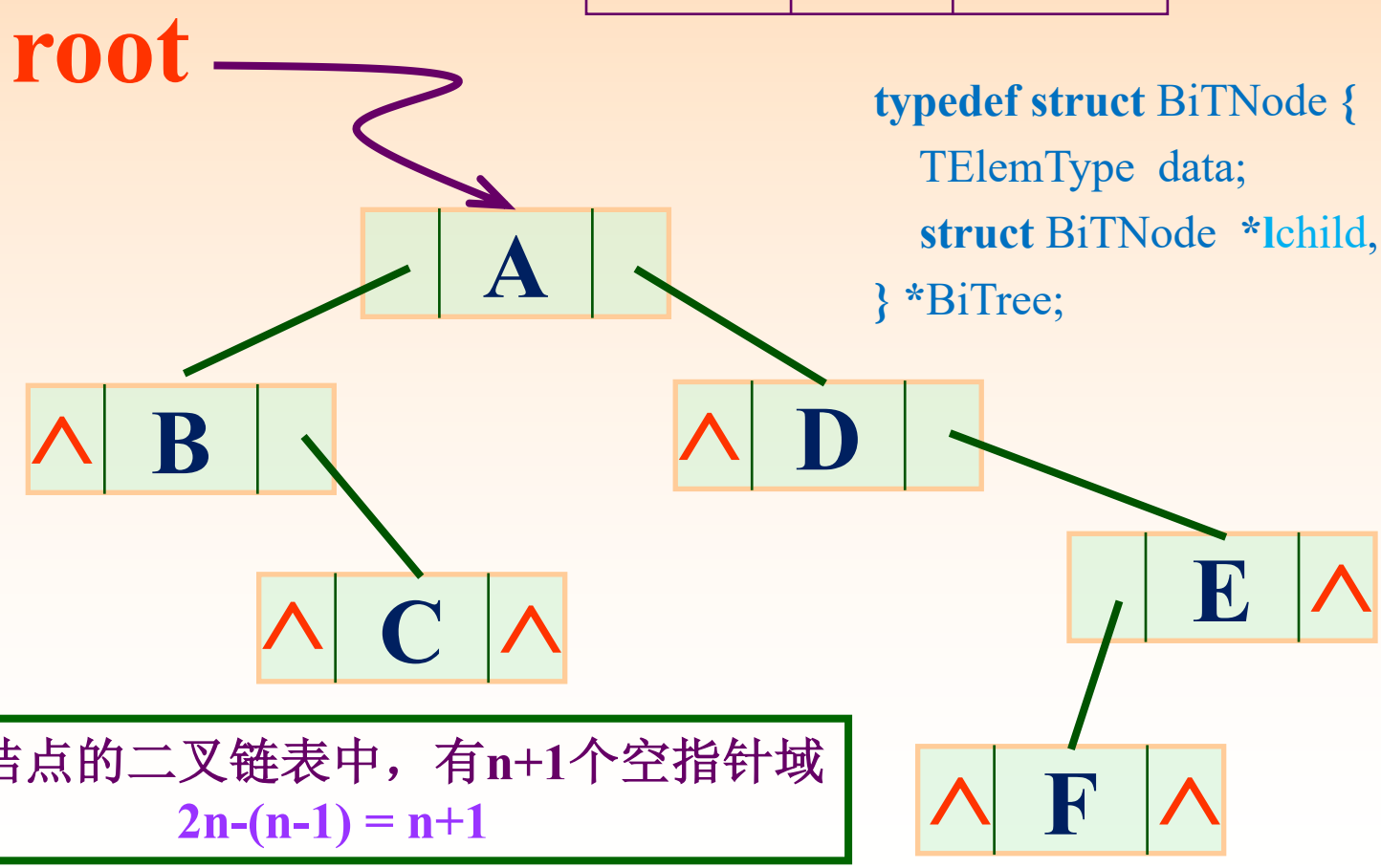
```
#define  MAX_TREE_SIZE  100
// 二叉树的最大结点数
typedef TElemType
      SqBiTree[MAX_TREE_SIZE];
// 1号单元存储根结点
SqBiTree  bt;
```



1	2	3	4	5	6	7	8	9	10	11
a	b	c	d	e	0	0	0	0	f	g

二叉树的链式存储表示-二叉链表

结点结构:

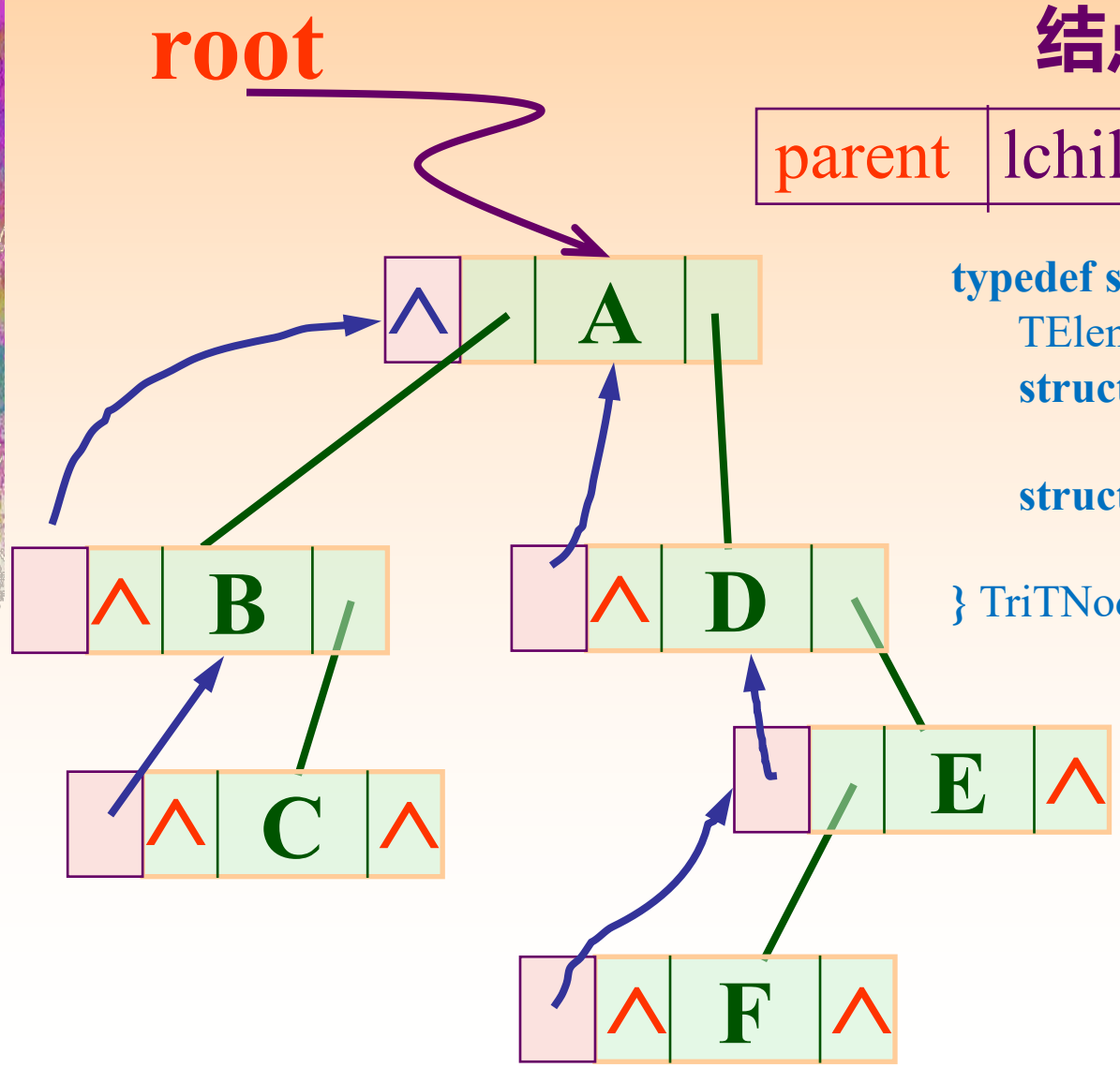



```
typedef struct BiTNode {
    TElemType  data;
    struct BiTNode  *lchild, *rchild;
} *BiTree;
```

在n个结点的二叉链表中，有n+1个空指针域

2n-(n-1) = n+1

二叉树的链式存储表示-三叉链表



结点结构

parent	lchild	data	rchild

```
typedef struct TriTNode {
    TElemType      data;
    struct TriTNode *lchild, *rchild;
                        // 左右孩子指针
    struct TriTNode *parent;
                        //父指针
} TriTNode, *TriTree;
```

6.3 遍历二叉树

问题描述：顺着某一条搜索路径**巡访**二叉树中的结点，使得每个结点**均被访问恰好一次**。

“**遍历**”是任何类型均有的操作。

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。

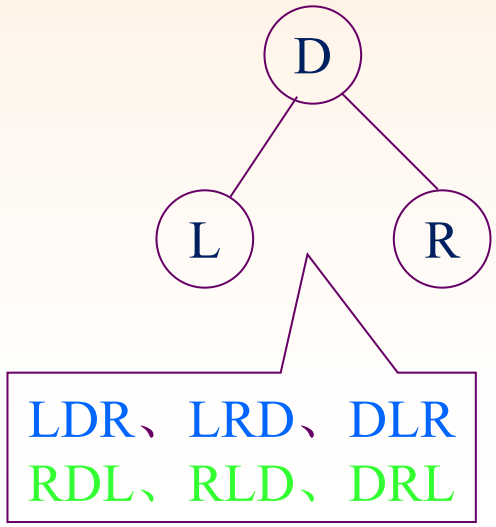
而二叉树是非线性结构，**每个结点有两个后继**，则**存在如何遍历**即按什么样的搜索路径遍历的问题。

有三种搜索策略：

1．**先上后下**的按层次遍历；(顺序存储结构）

2．**先左**（子树）**后右**（子树）的遍历；

3．**先右**（子树）**后左**（子树）的遍历。



3种先左后右的遍历算法

①**先**序遍历DLR

②**中**序遍历LDR

③**后**序遍历LRD

①先序遍历

若二叉树为空树，则空操作；否则，

（1）访问根结点；

（2）先序遍历左子树；

（3）先序遍历右子树。