

数据结构

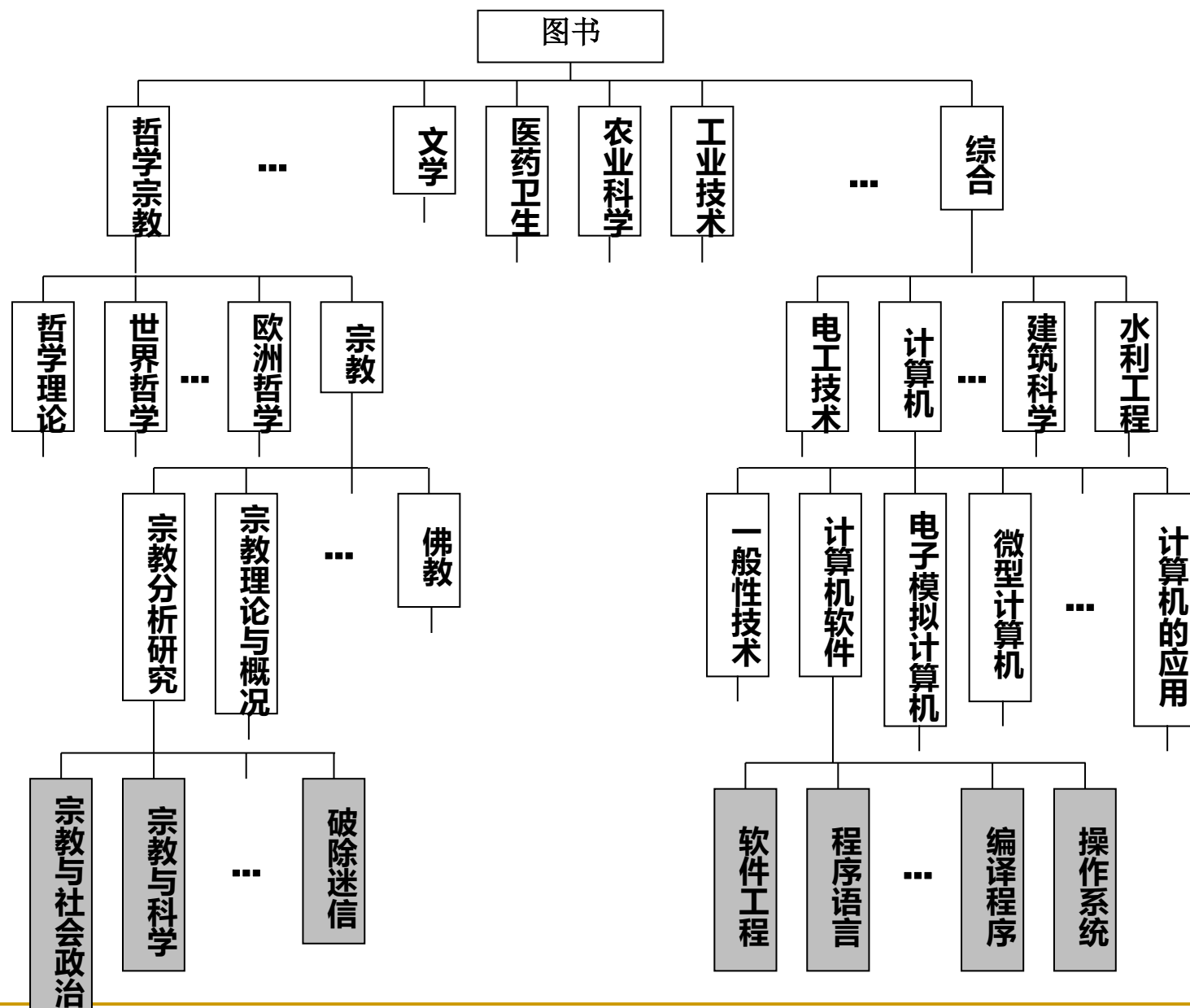
深圳技术大学
大数据与互联网学院

第六章 树和二叉树

- 6.1 树的定义和基本术语
- 6.2 二叉树
- 6.3 遍历二叉树和线索二叉树
- 6.4 树和森林
- 6.5 树与等价问题
- 6.6 赫夫曼树及其应用
- 6.7 回溯法与树的遍历
- 6.8 树的计数

[例1.1] 该如何摆放书，才能让读者很方便地找到你手里这本《数据结构》？





6.1 树的定义和二叉树

一. 树的定义

- 树是有 n ($n \geq 0$) 个结点的有限集合。
- 每个结点都有唯一的直接前驱，但可能有多个后继

6.1 树的定义和二叉树

一. 树的定义

- 如果 $n=0$, 称为空树
- 如果 $n>0$, 称为非空树, 对于非空树, 有且仅有一个特定的称为根(Root)的节点(无直接前驱)
- 如果 $n>1$, 则除根以外的其它结点划分为 m ($m>0$)个互不相交的有限集 T_1, T_2, \dots, T_m , 其中每个集合本身又是一棵树, 并且称为根的子树(SubTree)。
- 树是可以递归定义的

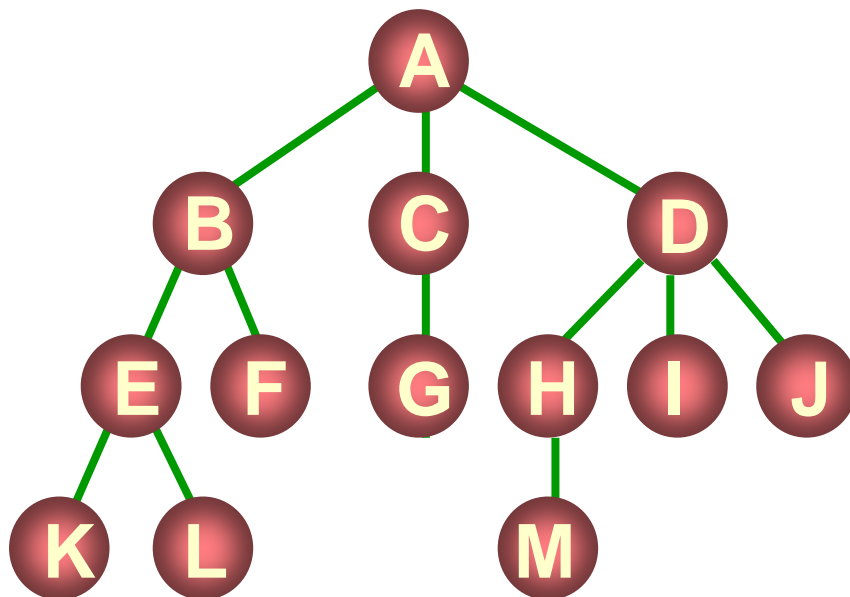
6.1 树的定义和二叉树

一. 树的定义

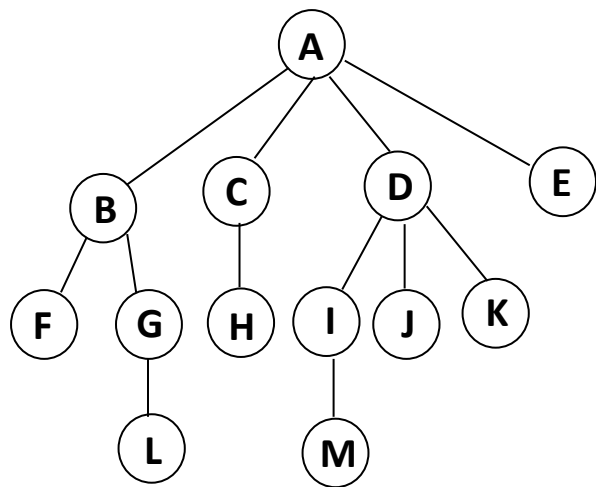
- 其中：A是根；其余结点分成三个互不相交的子集
- $T1 = \{B, E, F, K, L\}$ ； $T2 = \{C, G\}$ ； $T3 = \{D, H, I, J, M\}$ ，
- $T1, T2, T3$ 都是根A的子树，且本身也是一棵树



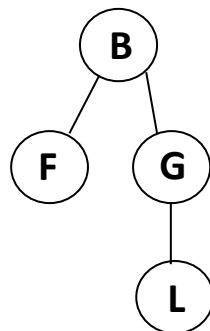
只有根结点的树



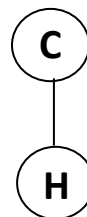
❖ 树与非树



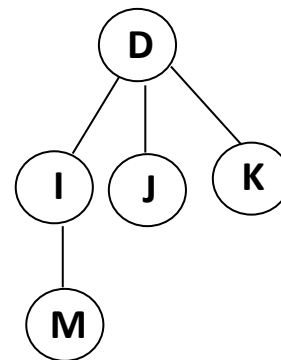
(a) 树 T



(b) 子树 T_{A1}



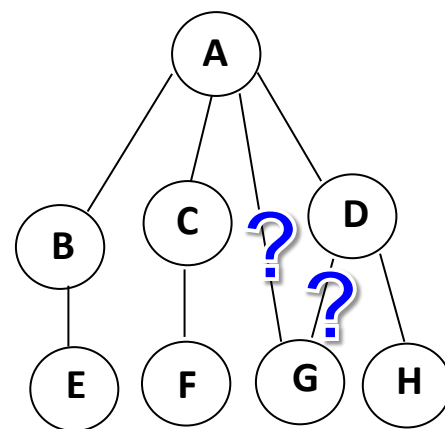
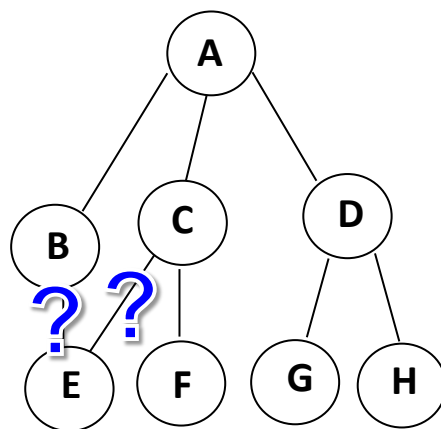
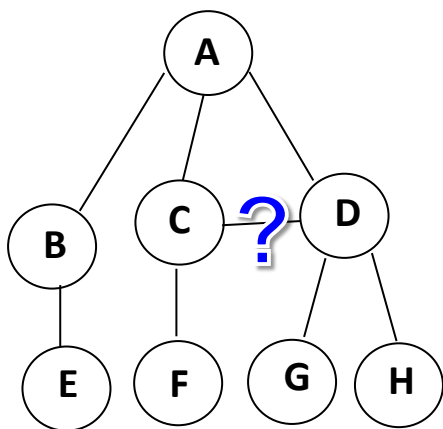
(c) 子树 T_{A2}



(d) 子树 T_{A3}



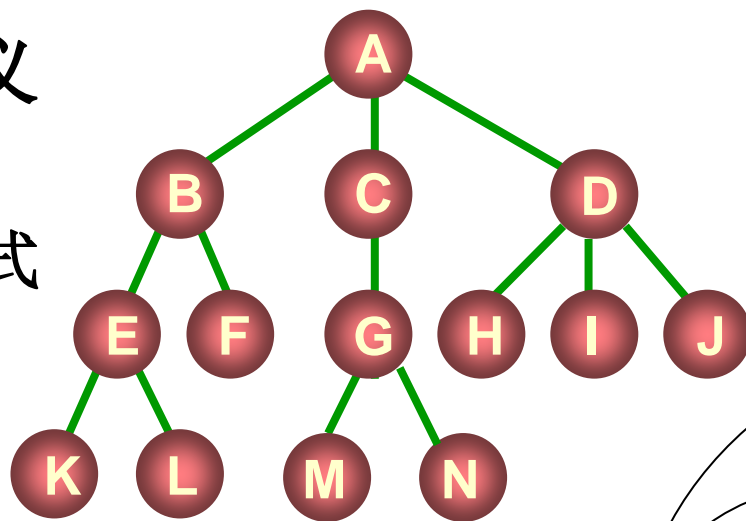
(e) 子树 T_{A4}



6.1 树的定义和二叉树

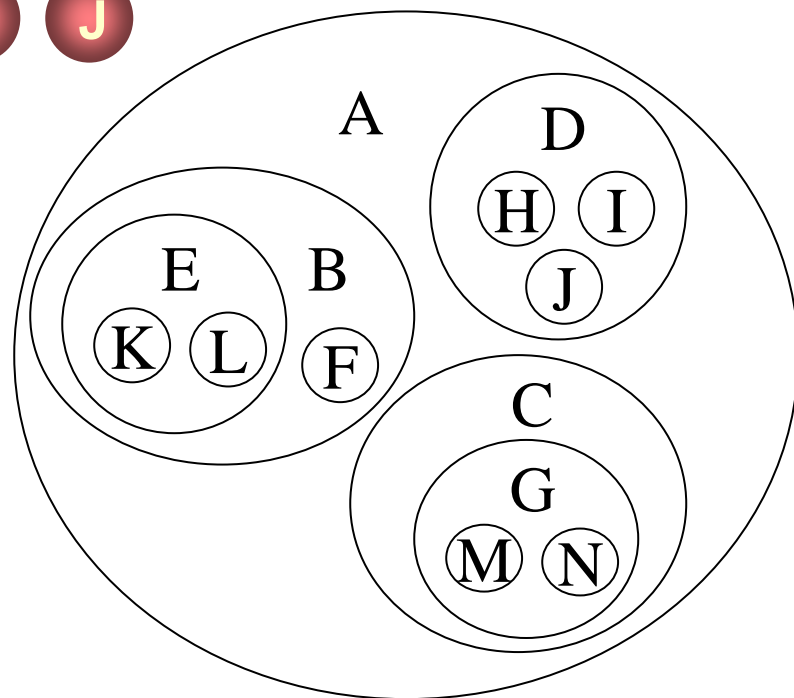
一. 树的定义

- 树状图
- 广义表形式
- 嵌套集合



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

(b) 广义表形式

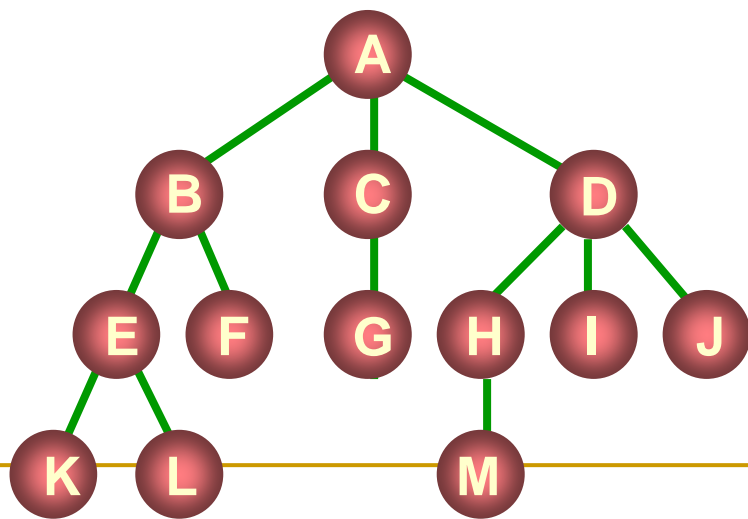


(a) 嵌套集合形式

6.1 树的定义和二叉树

二. 树的术语

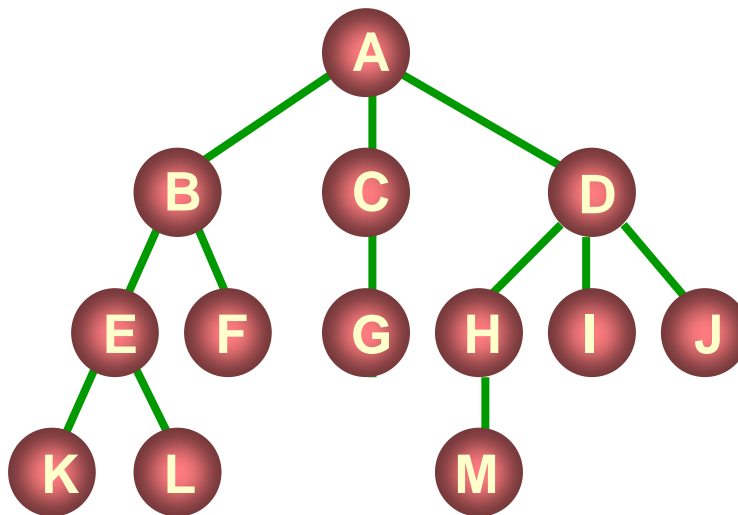
- 结点：包含一个数据元素及若干指向其子树的分支
- 分支：结点之间的连接
- 结点的度：结点拥有的子树数
- 树的度：树中结点度的最大值称为树的度
- 叶结点：度为0的结点[没有子树的结点]
- 分支结点：度不为0的结点[包括根结点]，也称为非终端结点。除根外称为内部结点



6.1 树的定义和二叉树

二. 树的术语

- 双亲
- 孩子
- 兄弟
- 祖先
- 子孙

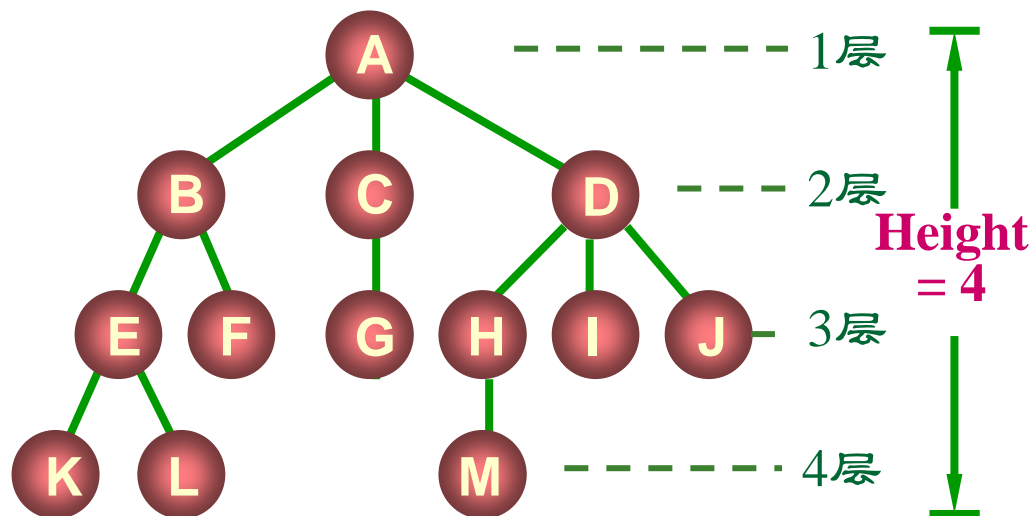
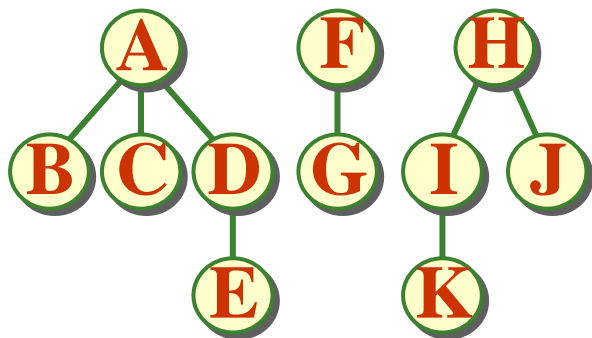


- 有序树/无序树：孩子的位置是否可以交换
 - 关系：孩子是父亲的质因子，例1/2/5-20，用无序树表示
 - 关系：孩子是父亲的递增质因子，例1/2/5-20，用有序树表示

6.1 树的定义和二叉树

二. 树的术语

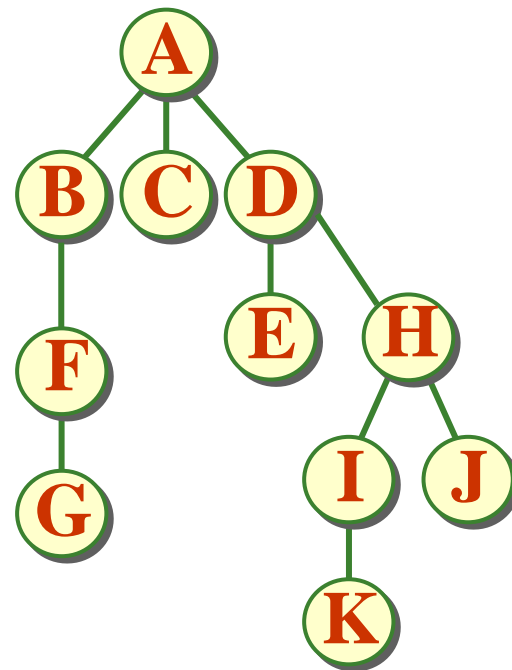
- 层次：根结点为第一层，其孩子为第二层，依此类推
- 深度：树中结点的最大层次
- 森林：互不相交的树的集合。对树中每个结点而言，其子树的集合即为森林



6.1 树的定义和二叉树

二. 概念练习

- 已知一颗树如图所示，求
 - 求树的度和深度
 - 求叶子结点和分支结点的数量
 - 求结点H的度
 - 求结点B的子孙
 - 求结点I的祖先



6.2 二叉树

一. 二叉树的定义

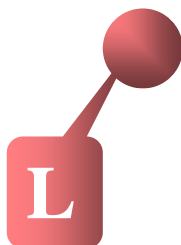
- 每个结点最多有 2 棵子树
- 二叉树的子树有左右之分



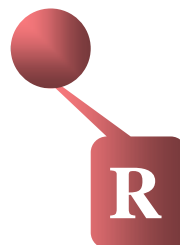
空树



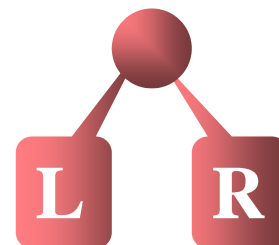
只有根



只有左子树



只有右子树



有左右子树

6.2 二叉树

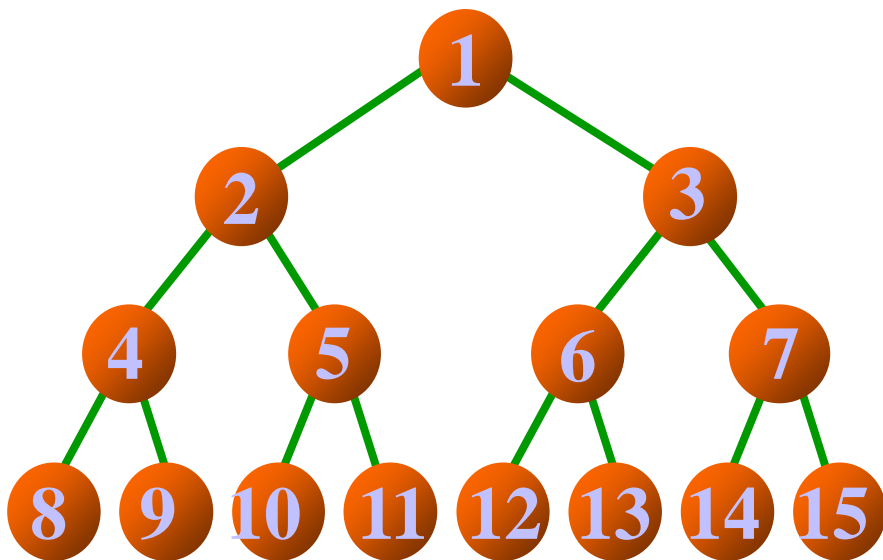
二. 二叉树的性质

- ① 在二叉树的第 i 层上至多有 2^{i-1} 个结点
- ② 深度为 k 的二叉树至多有 2^k-1 个结点
- ③ 如果二叉树终端结点数为 n_0 ，度为2的结点数为 n_2 ，则 $n_0=n_2+1$
 - $n = n_0 + n_1 + n_2$ ， $n = B+1$ ， $B = n_1 + 2n_2$ （ B 是分支数量）
 - $n = n_1 + 2n_2 + 1$
 - $n_0 + n_1 + n_2 = n_1 + 2n_2 + 1$
 - $n_0 = n_2 + 1$

6.2 二叉树

三. 满二叉树

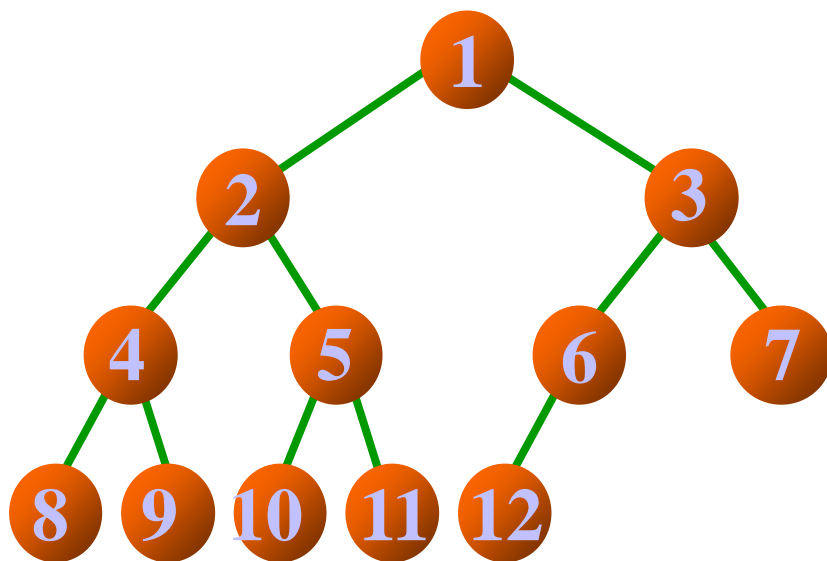
- 一个深度为 k 且有 2^k-1 个结点的二叉树
- 每层上的结点数都是最大数、
- 自上而下、自左至右连续编号



6.2 二叉树

四. 完全二叉树

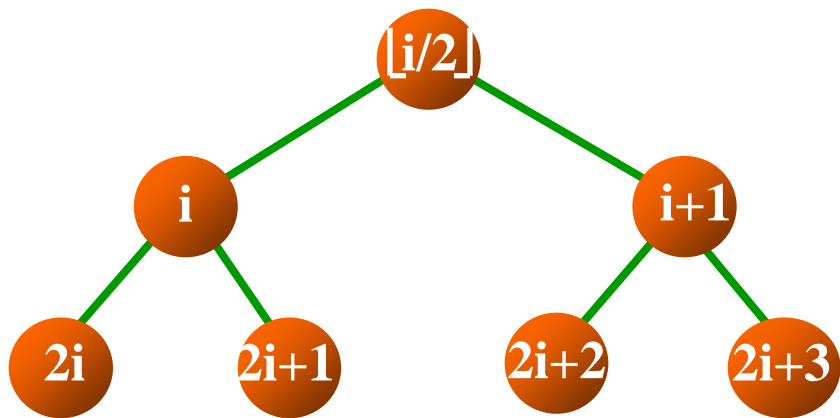
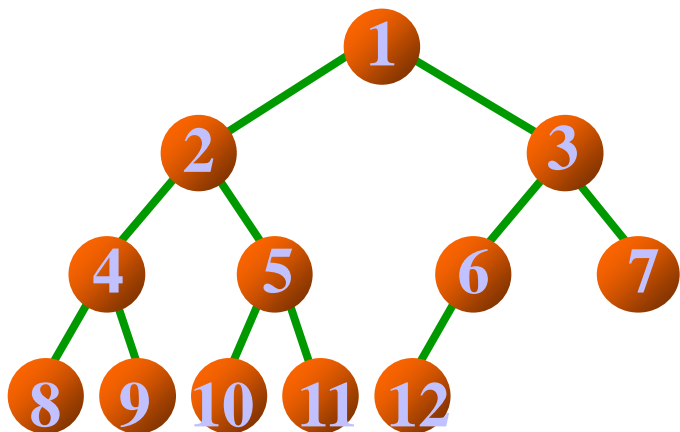
- 当且仅当每一个结点都与深度相同的满二叉树中编号从1到n的结点一一对应的二叉树
- 叶子结点只在最大两层上出现，左子树深度与右子树深度相等或大1



6.2 二叉树

四. 完全二叉树

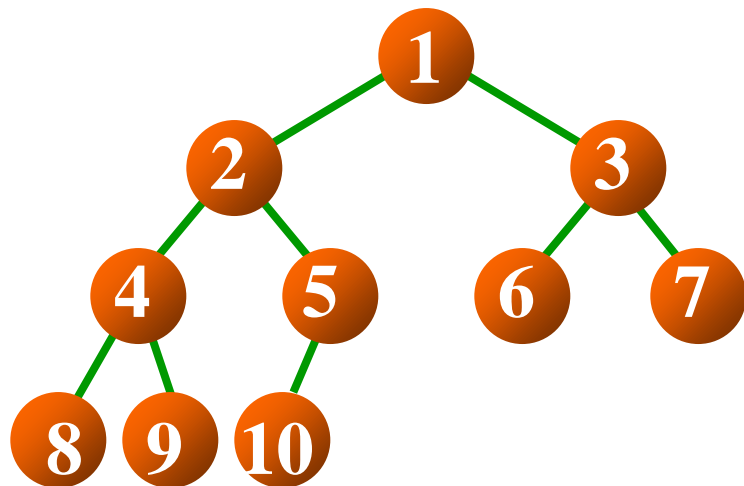
- 性质4: 具有 n 个结点的完全二叉树, 其深度为 $\log_2 n + 1$
- 性质5: 在完全二叉树中, 结点 i 的双亲为 $i/2$, 结点 i 的左孩子 $LCHILD(i) = 2i$, 结点 i 的右孩子 $RCHILD(i) = 2i+1$



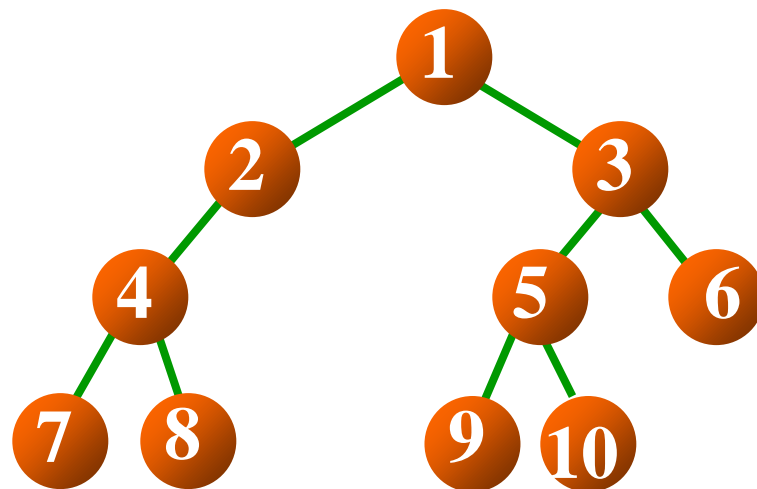
6.2 二叉树

五. 二叉树的顺序存储结构

- 用一组连续的存储单元依次自上而下, 自左至右存储结点



完全二叉树的顺序表示



一般二叉树的顺序表示

6.1 树的定义和二叉树

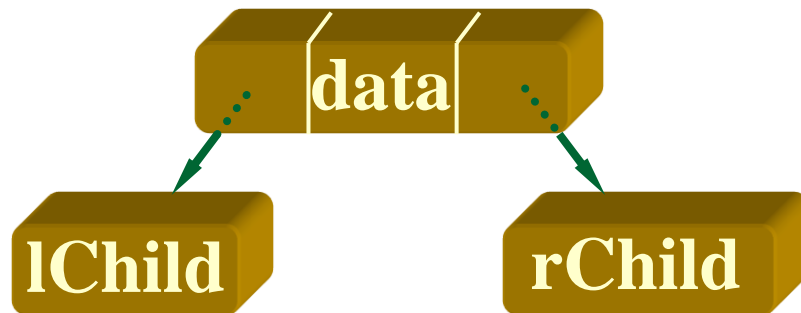
概念练习

- 已知一颗完全二叉树第7层有20个结点，则整棵树的结点数
- 在二叉树中，指针p指向的结点是叶子，则p满足条件？？
- 由4个结点组成的二叉树最多有多少种形态？
- 已知一棵完全二叉树有100个结点，根节点编号为1，按层次遍历编号，则结点45的父亲编号为？ 结点50的孩子编号情况如何？？

6.2 二叉树

六. 二叉树的链式存储结构

- 采用二叉链表，数据域加上左、右孩子指针



6.2 二叉树

六. 二叉树的链式存储结构

- 二叉链表由一个个结点组成，二叉链表结点由一个数据域和两个指针域组成

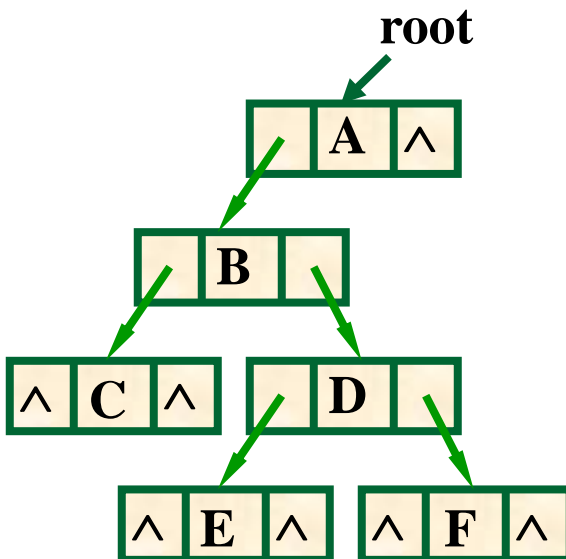
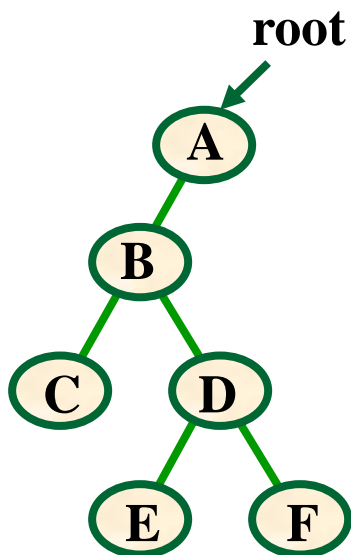
```
class BiTNode {  
    TElemType data;  
    BiTNode *lChild, *rChild;  
}  
class BinTree {  
    BiTNode* root;  
    ... //其他成员  
}
```



6.2 二叉树

六. 二叉树的链式存储结构

■ 二叉链表以及存储表示

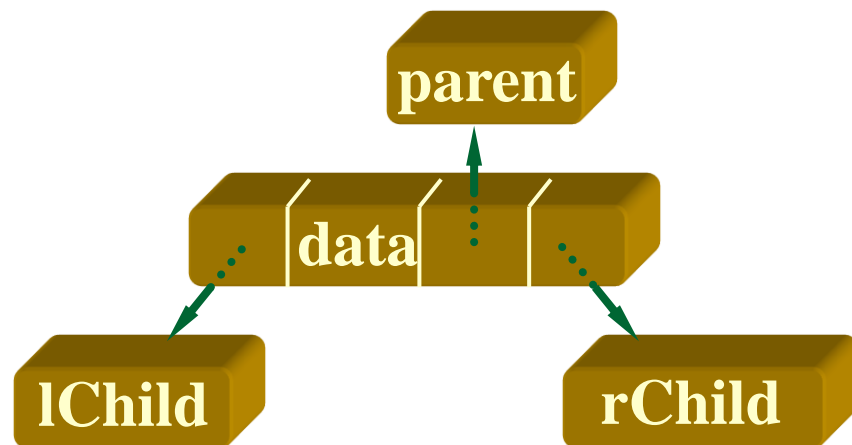


6.2 二叉树

六. 二叉树的链式存储结构

- 三叉链表, 采用数据域加上左、右孩子指针及双亲指针

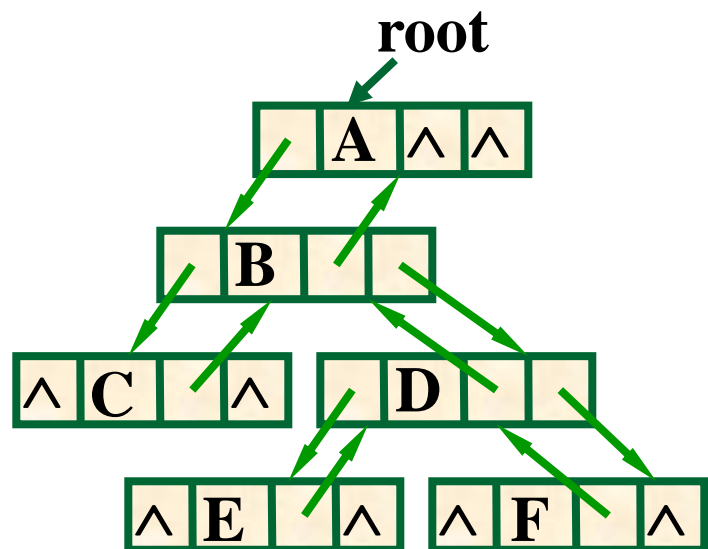
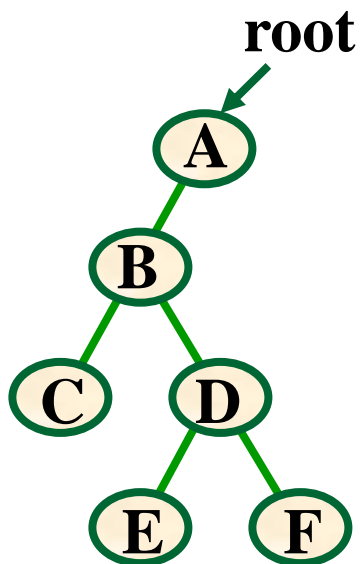
```
class BiTNode {  
    TElemType data;  
    BiTNode *parent, *lChild, *rChild;  
}  
  
class BinTree {  
    BiTNode* root;  
    ... //其他成员  
}
```



6.2 二叉树

六. 二叉树的链式存储结构

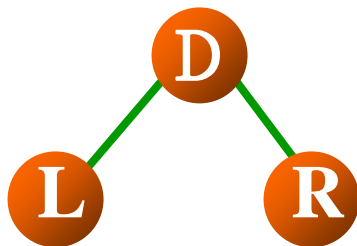
■ 三叉链表以及存储表示



6.3 遍历二叉树

一. 遍历的含义

- 树的遍历就是按某种次序访问树中的结点，要求每个结点访问一次且仅访问一次
- 一个二叉树由根节点与左子树和右子树组成
- 设访问根结点用D表示，遍历左、右子树用L、R表示
- 如果规定先左子树后右子树，则共有三种组合
 - DLR [先序遍历]
 - LDR [中序遍历]
 - LRD [后序遍历]

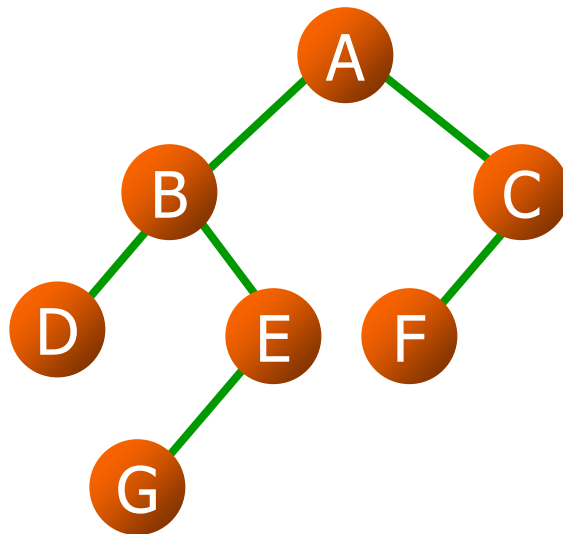


6.3 遍历二叉树

二. 先序遍历

■ 递归算法

- ❑ 若二叉树为空，则返回；否则：
- ❑ 访问根节点 (D)
- ❑ 先序遍历左子树 (L)
- ❑ 先序遍历右子树 (R)



如图输出结果：ABDEGCF
(第一个输出节点必为根节点)

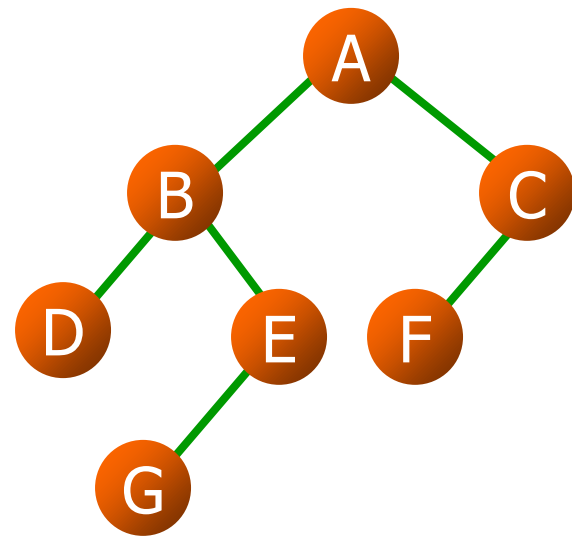
```
void PreOrderTraverse ( BiTNode* T ) {  
    if (T) {  
        cout << T->data;  
        PreOrderTraverse ( T->lChild );  
        PreOrderTraverse ( T->rChild );  
    }  
}
```

6.3 遍历二叉树

三. 中序遍历

■ 递归算法

- ❑ 若二叉树为空，则返回；否则：
- ❑ 中序遍历左子树 (L)
- ❑ 访问根节点 (D)
- ❑ 中序遍历右子树 (R)



输出结果：DBGEAFC

*(先于根节点A输出的节点为左子树的节点
后于根节点A输出的节点为右子树的节点)*

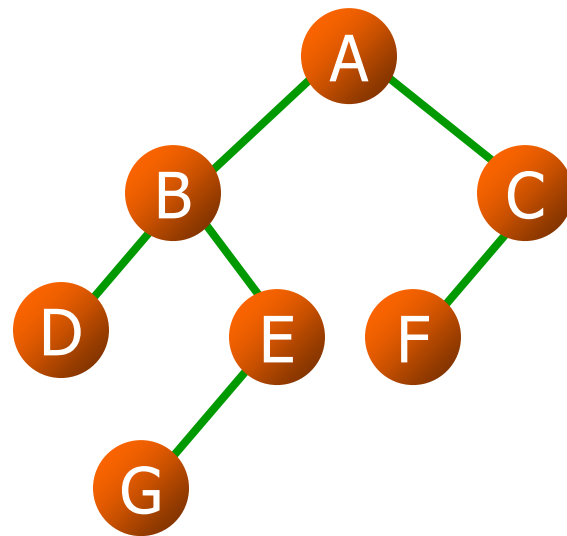
```
void InOrderTraverse ( BiTNode* T ) {  
    if (T) {  
        InOrderTraverse ( T->lChild );  
        cout << T->data;  
        InOrderTraverse ( T->rChild );  
    }  
}
```

6.3 遍历二叉树

四. 后序遍历

■ 算法

- ❑ 若二叉树为空，则返回；否则：
- ❑ 后序遍历左子树(L)
- ❑ 后序遍历右子树(R)
- ❑ 访问根节点(D)



输出结果：DGEBFCA

(最后一个输出节点必为根节点)

```
void PostOrderTraverse ( BiTNode* T ) {  
    if (T) {  
        PostOrderTraverse ( T->lChild );  
        PostOrderTraverse ( T->rChild );  
        cout << T->data;  
    }  
}
```

6.3 遍历二叉树

五. 层次遍历

- 层次遍历二叉树，是从根结点开始遍历，按层次次序“**自上而下，从左至右**”访问树中的各结点。
- 为保证是按层次遍历，必须设置一个队列，非递归算法：
 - 设 T 是指向根结点的指针变量，若二叉树为空，则返回；否则，令 $p=T$ ， p 入队，执行以下循环：
 - (1) 队首元素出队到 p ；
 - (2) 访问 p 所指向的结点；
 - (3) 将 p 所指向的结点的左、右子结点依次入队。直到队空为止。