



第六章 树与二叉树

机电工程与自动化学院 L栋301

任卫红 助理教授

renweihong@hit.edu.cn

<http://faculty.hitsz.edu.cn/renweihong>



6.1 树的概念与基本术语

第一节 树的概念与基本术语

一、树的定义(Tree)

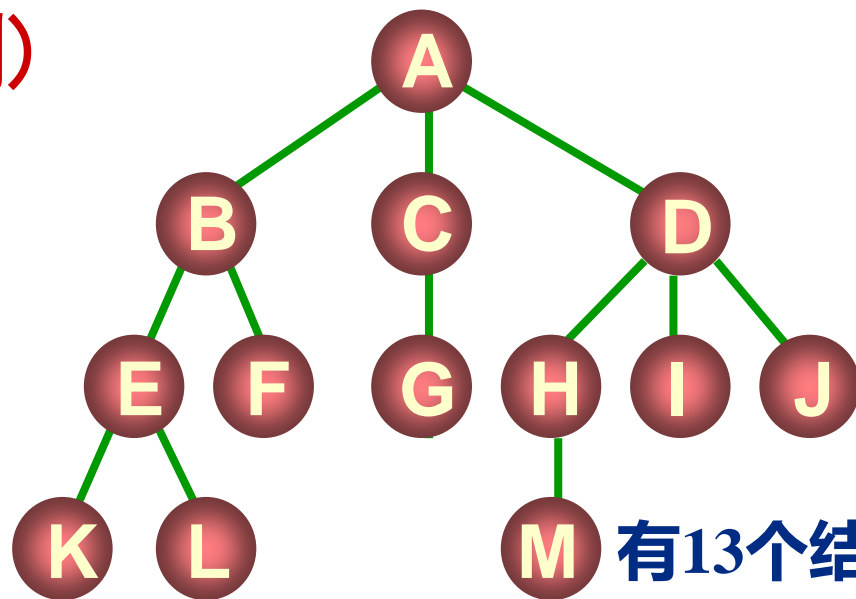
- 树是有 n ($n \geq 0$) 个结点的有限集合。
- 如果 $n=0$, 称为空树;
- 如果 $n>0$, 称为非空树, 对于非空树, 有且仅有一个特定的称为根 (Root) 的节点 (无直接前驱)
- 如果 $n>1$, 则除根以外的其它结点划分为 m ($m>0$) 个互不相交的有限集 T_1, T_2, \dots, T_m , 其中每个集合本身又是一棵树, 并且称为根的子树 (SubTree)。(此为递归定义)
- 每个结点都有**唯一的直接前驱**, 但可能有**多个后继**

第一节 树的概念与基本术语

一、树的定义(举例)



只有根结点的树



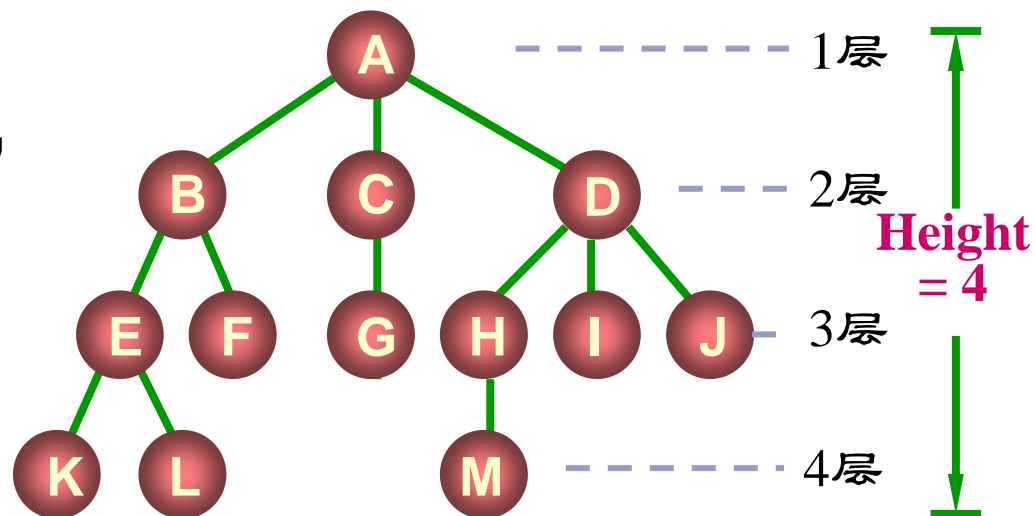
有13个结点的树

- 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

第一节 树的概念与基本术语

二、树的基本术语

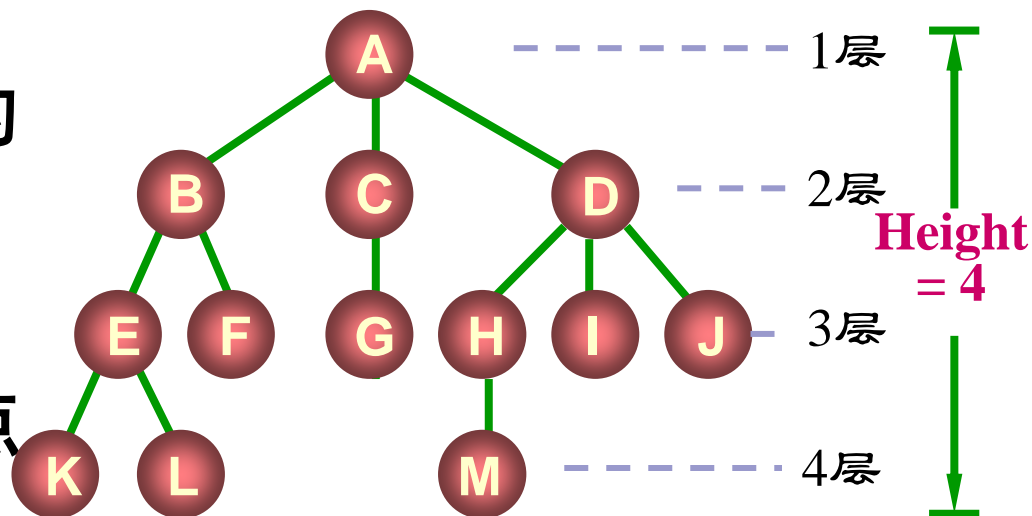
- **结点**：包含一个数据元素及若干指向其子树的分支
- **结点的度**：结点拥有的子树数
- **叶子结点(终端节点)**：度为0的结点[没有子树的结点]
- **分支结点**：度不为0的结点[包括根结点]，也称为非终端结点。除根外称为内部结点



第一节 树的概念与基本术语

二、树的基本术语

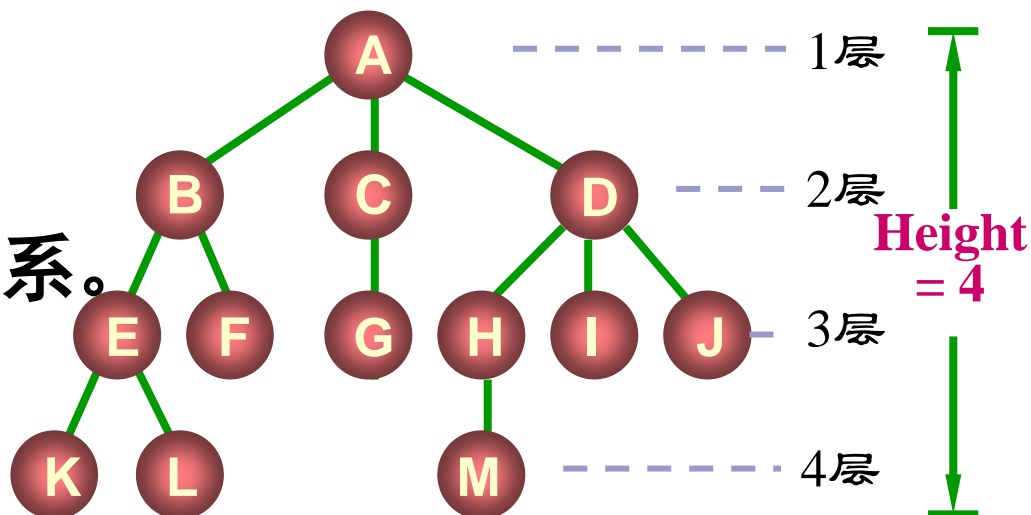
- **孩子**：结点的子树的根[直接后继，可能有多個]
- **双亲**：孩子的直接前驱[最多只能有一个]
- **兄弟**：同一双亲的孩子
- **子孙**：以某结点为根的树中的所有结点
- **祖先**：从根到该结点所经分支上的所有结点



第一节 树的概念与基本术语

二、树的基本术语

- 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
- 有序树：子树之间存在确定的次序关系。
- 无序树：子树之间不存在确定的次序关系。



第一节 树的概念与基本术语

二、树的基本术语

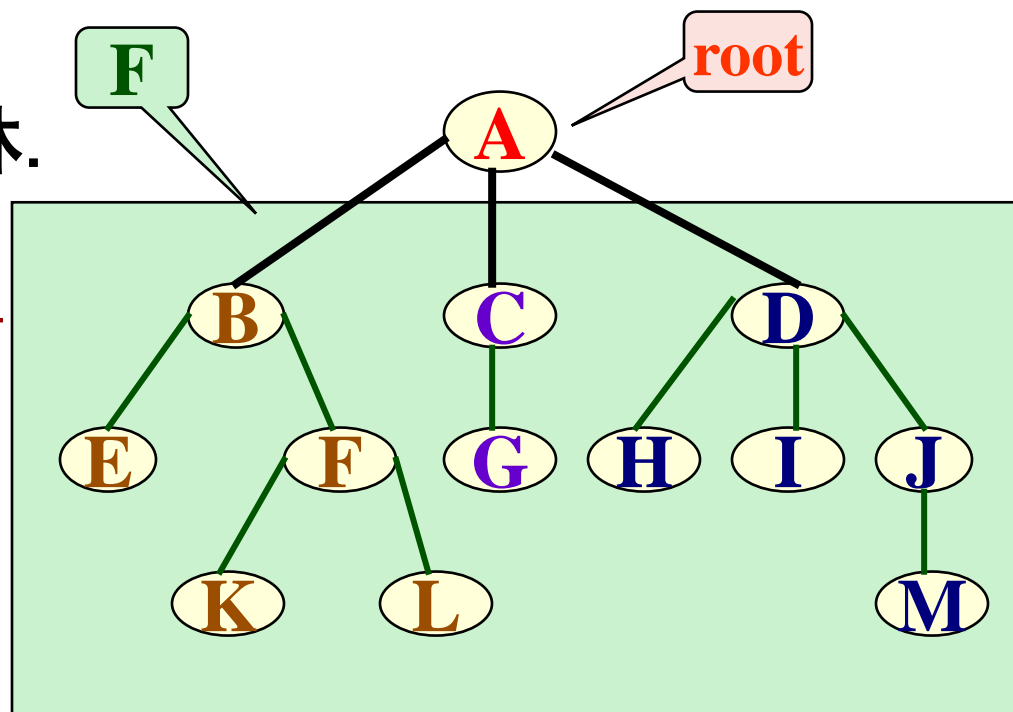
- 森林：互不相交的树的集合。
对树中每个结点而言，
其子树的集合即为森林。

任何一棵非空树是一个二元组

$$\text{Tree} = (\text{root}, F)$$

其中：root 被称为根结点

F 被称为子树森林



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)





6.2 二叉树

第二节 二叉树

一、二叉树(Binary Tree)

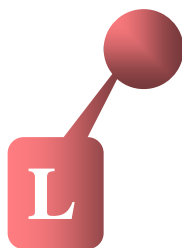
- 二叉树是一种特殊的树
- 每个结点最多有2棵子树
- 二叉树的子树有左右之分



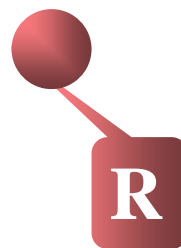
空树



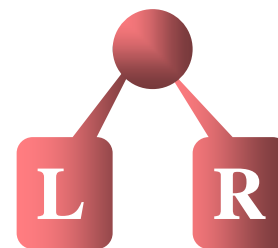
只有根



只有左子树



只有右子树



有左右子树

第二节 二叉树

二、二叉树性质 1

- 在二叉树的第 i 层上至多有 2^{i-1} 个结点

- 证明:

1. $i=1$, 只有一个根节点, 因此 $2^{i-1}=2^0=1$
2. 设第 $i-1$ 层上, 以上性质成立, 即第 $i-1$ 层至多有 $2^{(i-1)-1}$ 结点。由二叉树的定义可知, 任何结点的度小于2, 因此, 第 i 层上的结点数最多为第 $i-1$ 层上的两倍, 即 $2*2^{i-2}=2^{i-1}$

第二节 二叉树

三、二叉树性质 2

- 深度为 k 的二叉树至多有 2^k-1 个结点

- 证明:

1. 由性质 1 , 已知第 i 层上结点数最多为 2^{i-1}

$$2. \quad \sum_{i=1}^k 2^{i-1} = 2^k - 1$$

第二节 二叉树

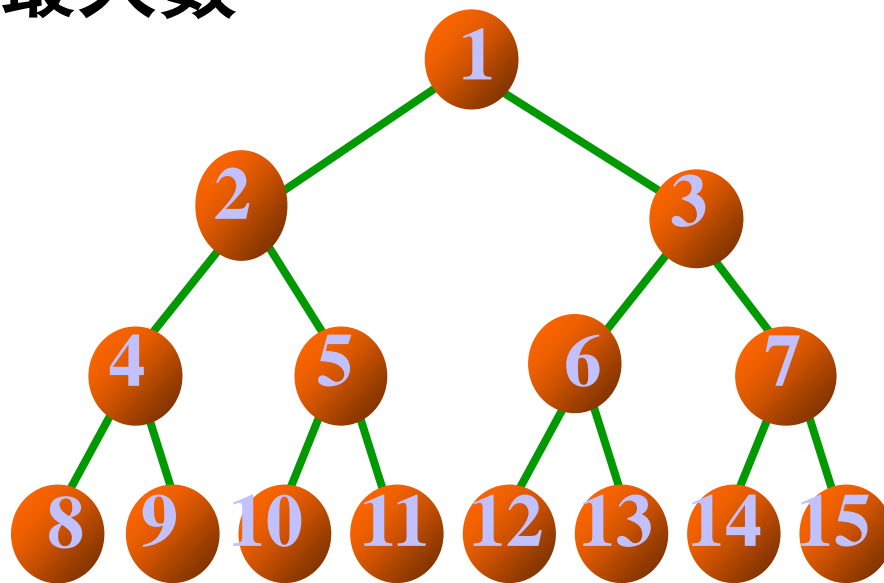
四、二叉树性质 3

- 如果二叉树终端结点数为 n_0 ，度为2的结点数为 n_2 ，则 $n_0=n_2+1$
- 证明：
 1. 设 n_1 为度为1的结点，则总结点数 $n= n_0+n_1+n_2$
 2. 设 B 为二叉树的分支数，除根结点外，每个结点有且只有一个分支，因此 $n=B+1$
 3. 每个分支皆由度为1或2的结点发出， $B=n_1+2n_2$
 4. $n=B+1=(n_1+2n_2)+1 = n_0+n_1+n_2$ ，因此 $n_0=n_2+1$

第二节 二叉树

五、满二叉树

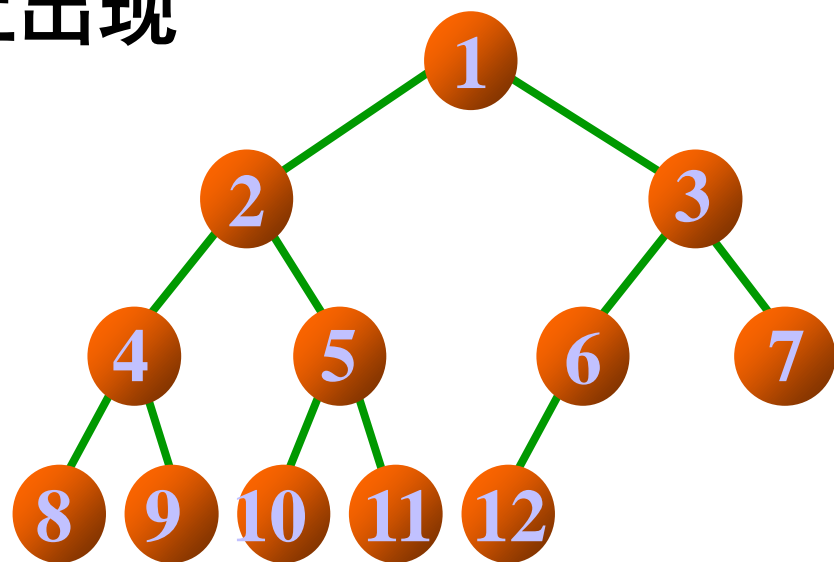
- 一个深度为 k 且有 2^k-1 个结点的二叉树
- 每层上的结点数都是最大数
- 可以自上而下、自左至右连续编号



第二节 二叉树

六、完全二叉树（不满的情况）

- 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



第二节 二叉树

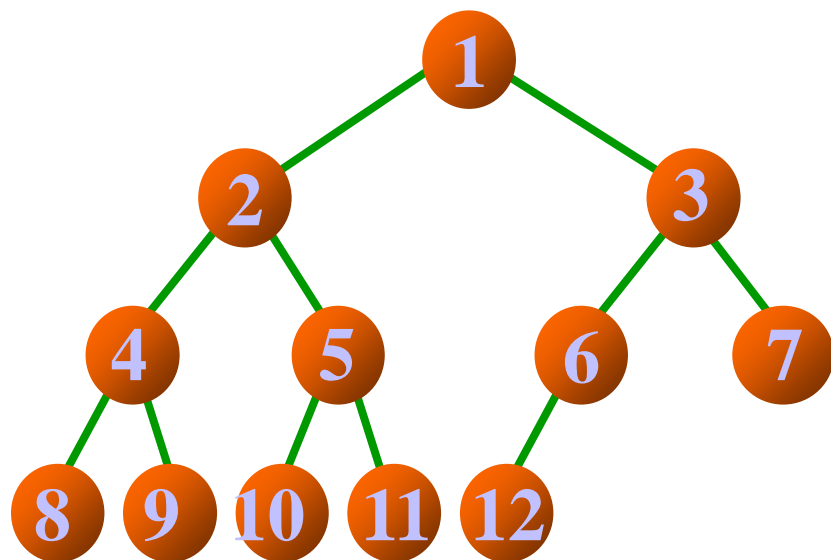
六、完全二叉树(性质4)

- 具有 n 个结点的完全二叉树, 其深度为 $\lfloor \log_2 n \rfloor + 1$
- 设 k 为深度, 由二叉树性质2, 已知

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

即 $2^{k-1} \leq n < 2^k$

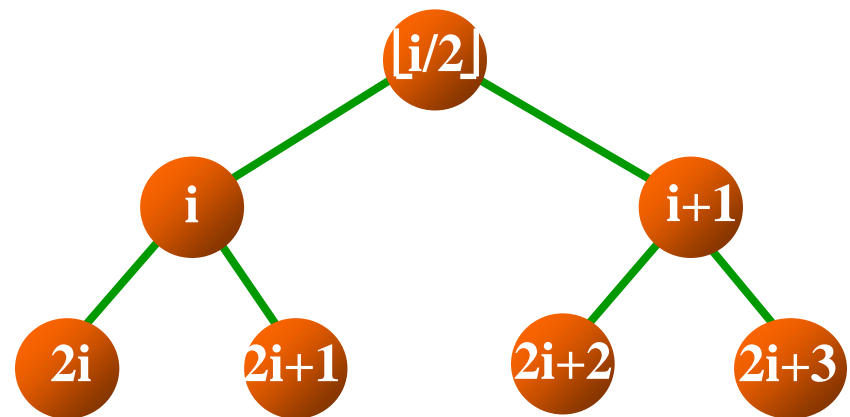
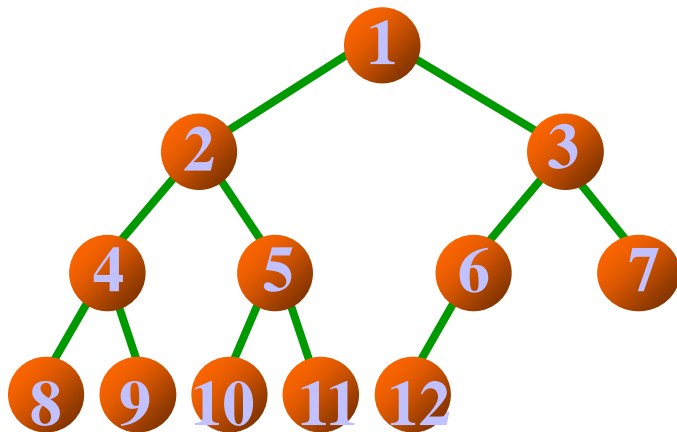
即 $k = \lfloor \log_2 n \rfloor + 1$



第二节 二叉树

六、完全二叉树(性质5)

- 在完全二叉树中，结点 i 的双亲为 $\lfloor i/2 \rfloor$
- 结点 i 的左孩子 $LCHILD(i) = 2i$
- 结点 i 的右孩子 $RCHILD(i) = 2i+1$



作业

1. 设深度为 h 的二叉树上只有叶子结点和同时具有左右子树的结点，则此类二叉树中所包含的结点数目至少为_____。

A. 2^h

C. $2^{(h+1)}$

B. $2^{(h-2)}$

D. $2^{(h-1)}$

作业

2. 二叉树的深度为 k ，则二叉树最多有_____个结点。

A. 2^k

B. $2^{(k-1)}$

C. $2^{(k-1)}$

D. 2^{k-1}

练习

1.【自测题】若一棵度为7的树有8个度为1的结点，有7个度为2的结点，有6个度为3的结点，有5个度为4的结点，有4个度为5的结点，有3个度为6的结点，有2个度为7的结点，该树一共有（ ）叶结点。

☐ A.77

☐ B.35

☐ C.78

☐ D.28

题解:树的度表示节点的子树或直接继承者的数目，若一棵树有 n 个结点，那么其度必然为 $n-1$ ，即节点数目=所有节点度数之和+1。此题中，我们先求出所有结点的度之和，从而推出结点数量，然后再减去度非0的结点数量，剩余的度为0的结点即为叶子结点。

练习

2. 【自测题】若一棵二叉树有2013个结点，且无度为1的结点，则叶结点的个数为（）

☐ A.1007

☐ B.1003

☐ C.1004

☐ D.1001

题解:树的度表示节点的子树或直接继承者的数目，若一棵树有 n 个结点，那么其度必然为 $n-1$ ，即节点数目=所有节点度数之和+1。二叉树有2013个结点，故所有结点度数之和为2012，二叉树中，节点的度只有0，1，2三种可能，由于题中指出无度为1的结点，故 $2012 = \text{度为2的结点数量} \times 2$ ，即度为2的结点数量为1006个，所以，度为0的结点（叶子结点）的数量为 $2013 - 1006 = 1007$ 个。

练习

3. 【自测题】将含有83个结点的完全二叉树从根结点开始编号，根为1号，后面按从上到下、从左到右的顺序对结点编号，那么编号为41的双亲结点编号为（ ）

☐ A.20

☐ B.21

☐ C.40

☐ D.42

题解:在完全二叉树中，若结点*i*有孩子，则该结点的左孩子标号为 $2i$ ，右孩子的标号为 $2i+1$ 。那么编号为41的结点，其双亲节点为20。

练习

4. 【自测题】 设二叉树有 n 个结点，则其深度为（ ）

- ☐ A. $n-1$
- ☐ B. $\log_2 n + 1$
- ☐ C. 无法确定
- ☐ D. n

题解: 题目中并未指出二叉树的类型，是否为完全二叉树等，故无法确定其深度。

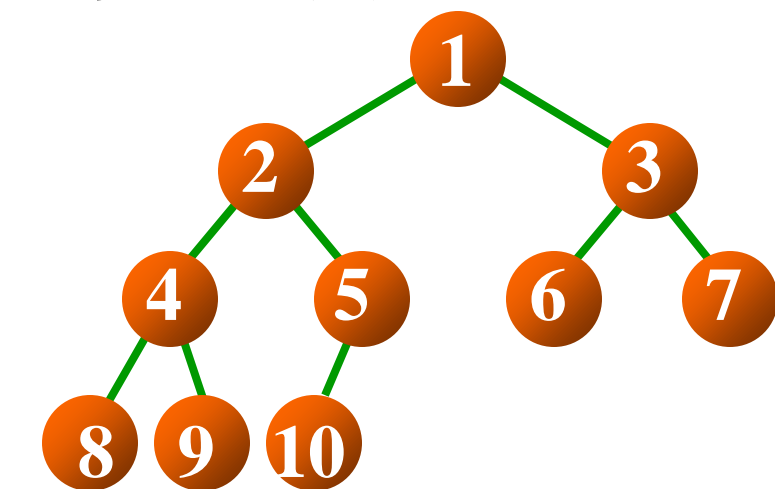
作业

3. 若一个二叉树具有8个度为2的结点，7个度为1的结点，则度为0的结点（叶子）个数是_____。
4. 具有 n 个结点的二叉树中，一共有_____个指针域，其中_____个用来指向结点的左右孩子，有_____个为NULL。

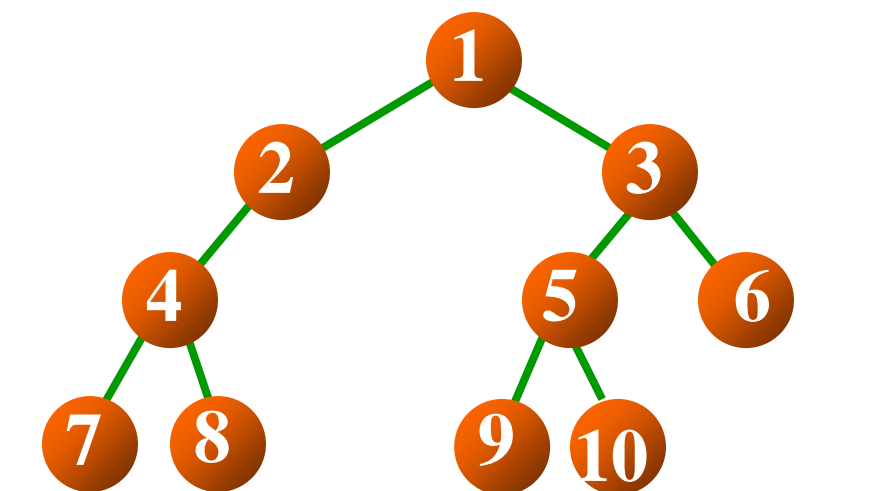
第二节 二叉树

七、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

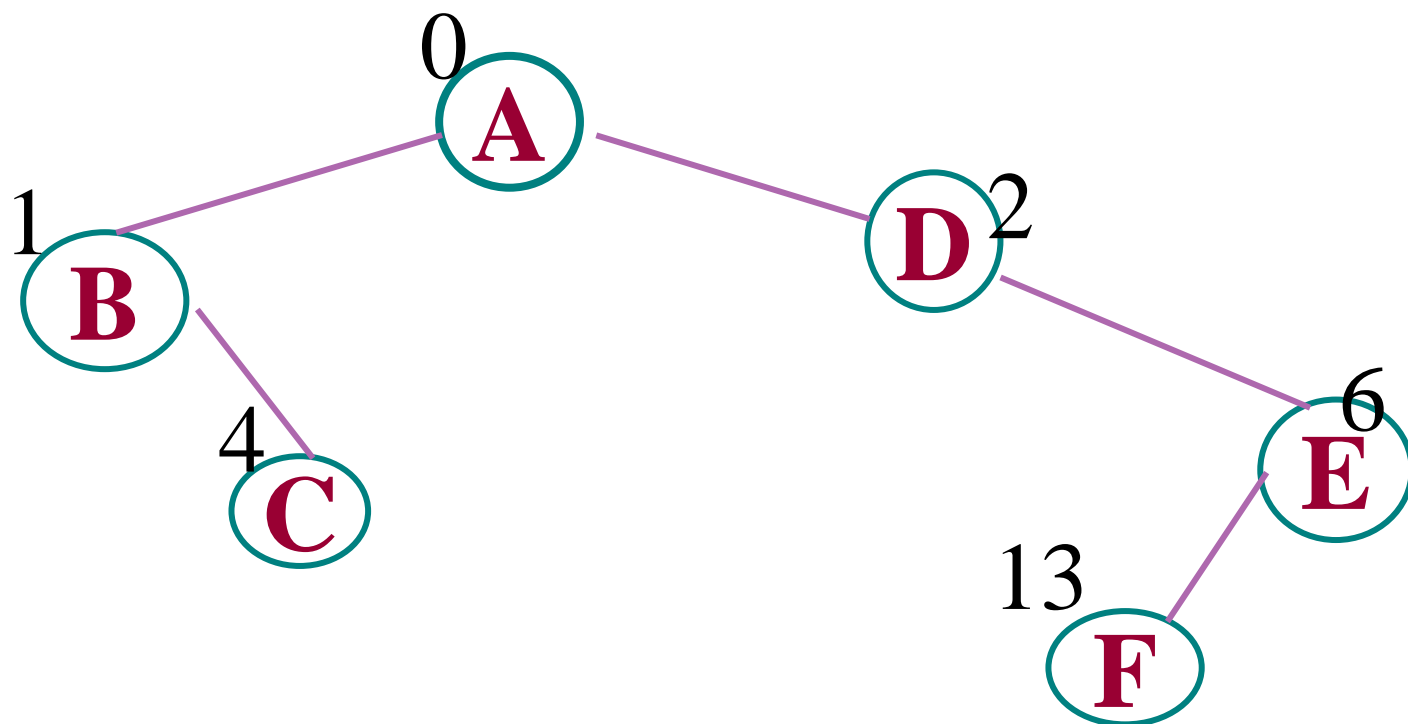


一般二叉树的顺序表示

一、 二叉树的顺序存储表示

```
#define MAX_TREE_SIZE 100;
]class BiTree
{
    char    *Tree;
public:
]    BiTree()
    {
        Tree = new char MAX_TREE_SIZE;
    }
}
```

例:



0 1 2 3 4 5 6 7 8 9 10 11 12 13

| | | | | | | | | | | | | | | |
|---|---|---|--|---|--|---|--|--|--|--|--|--|---|--|
| A | B | D | | C | | E | | | | | | | F | |
|---|---|---|--|---|--|---|--|--|--|--|--|--|---|--|

二叉树的顺序存储缺点: **浪费空间。**

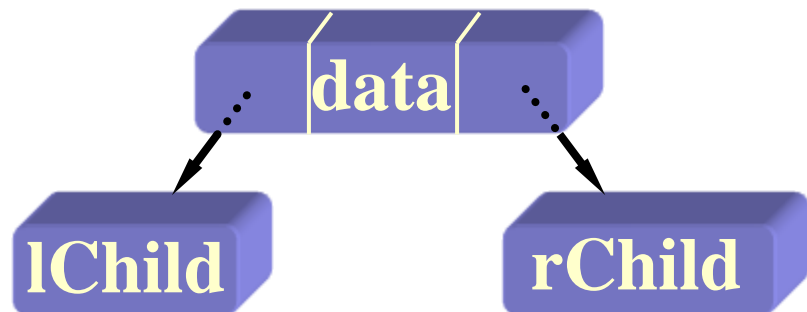


第二节 二叉树

七、二叉树的链式存储结构

1. 二叉链表

- 采用数据域加上左、右孩子指针

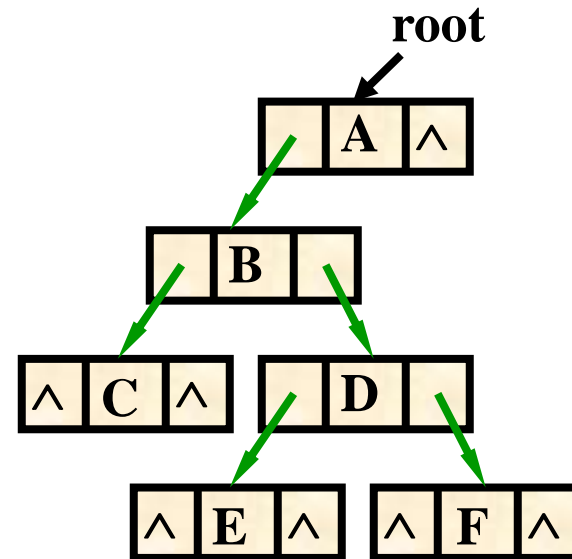
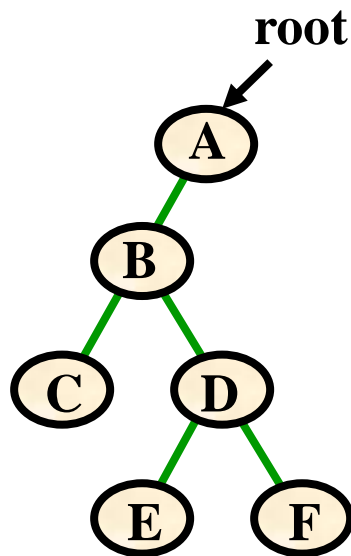


第二节 二叉树

七、二叉树的链式存储结构

1. 二叉链表(举例)

■ 二叉树（左）及其二叉链表（右）



```

class BiTNode          //结点结构
{
    char      data;      //数据
    BiTNode   *lchild,*rchild; //左、右孩子
    friend class BiTree;
public:
    BiTNode(char e,BiTNode *l=NULL,BiTNode *r=NULL):data(e),lchild(l),rchild(r) {} //构造函数
}

class BiTree           //二叉树
{
    BiTNode *Tree;      //树根
    void PreOrderTraverse(BiTNode *T);
public:
    BiTree():Tree(NULL) {} //树根为空
    void PreOrder();
}

```

结点结构:

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

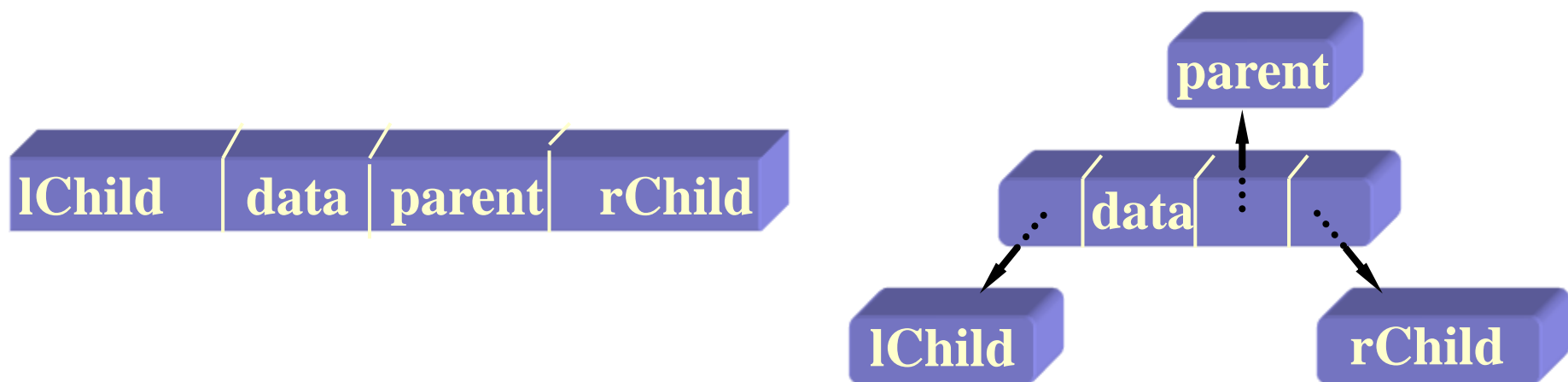


第二节 二叉树

七、二叉树的链式存储结构

2. 三叉链表

- 采用数据域加上左、右孩子指针及双亲指针

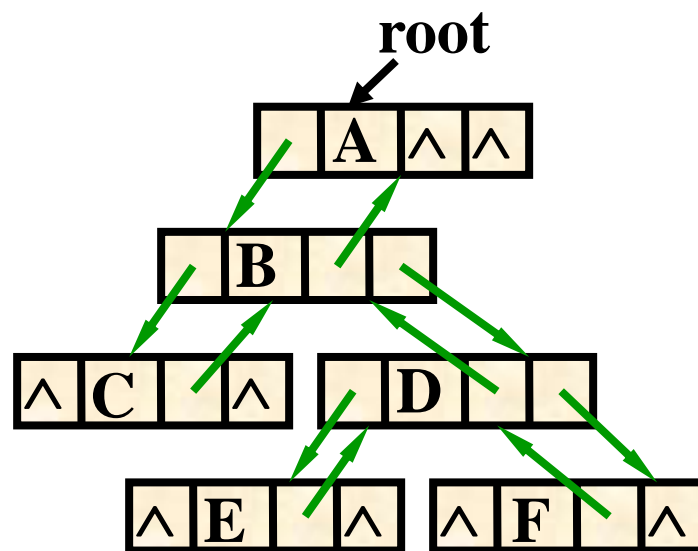
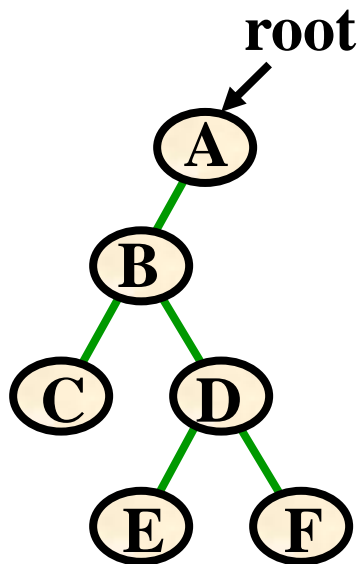


第二节 二叉树

七、二叉树的链式存储结构

2. 三叉链表(举例)

- 二叉树（左）及其三叉链表（右）



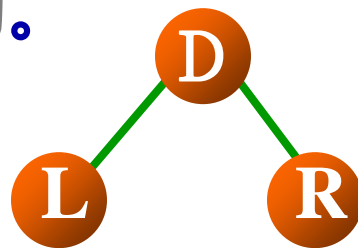


6.3 遍历二叉树

第三节 遍历二叉树

一、遍历二叉树

- 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（非线性结构线性化）
- 对“二叉树”而言，可以有三条搜索路径：
 - 1. 先上后下的按层次遍历；
 - 2. 先左（子树）后右（子树）的遍历；
 - 3. 先右（子树）后左（子树）的遍历。



第三节 遍历二叉树

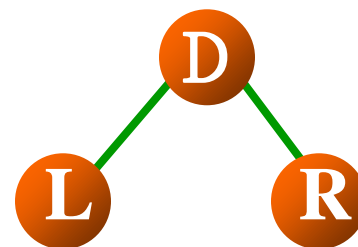
一、遍历二叉树

- 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

1. DLR [先序遍历]

2. LDR [中序遍历]

3. LRD [后序遍历]

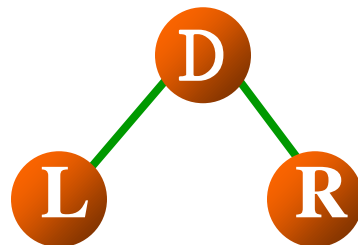


第三节 遍历二叉树

二、先序遍历二叉树

■ 算法：

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

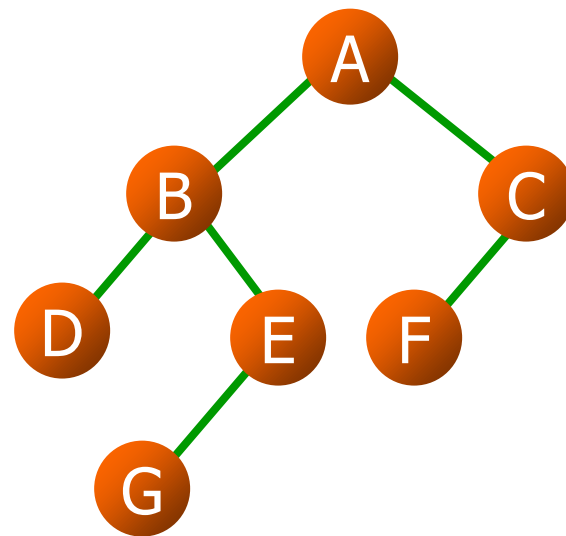


第三节 遍历二叉树

二、先序遍历二叉树

- 算法(举例)：

输出结果： ABDEGCF



第三节 遍历二叉树

二、先序遍历二叉树

```
void BiTree::PreOrderTraverse(BiTNode *T)    //私有方法，先序遍历二叉树T
{
    if(T)
    {
        cout<<T->data<<" ";
        PreOrderTraverse(T->lchild);
        PreOrderTraverse(T->rchild);
    }
}

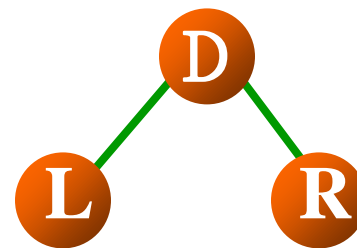
void BiTree::PreOrder()                      //公有接口，先序遍历二叉树
{
    PreOrderTraverse(T);
}
```

第三节 遍历二叉树

三、中序遍历二叉树

■ 算法：

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

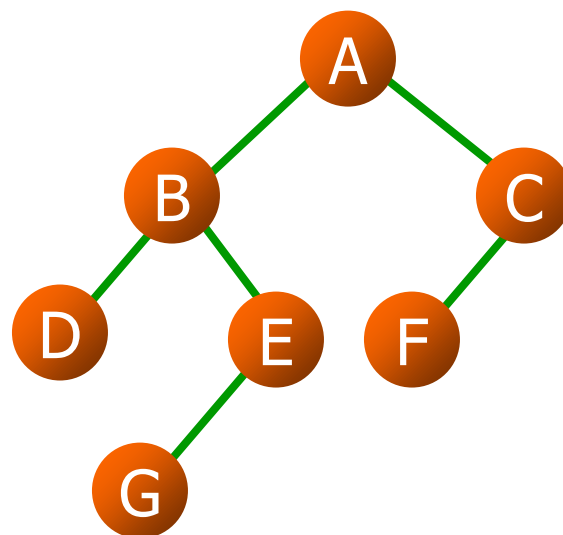


第三节 遍历二叉树

三、中序遍历二叉树

- 算法(举例)：

输出结果： **DBGEAFC**

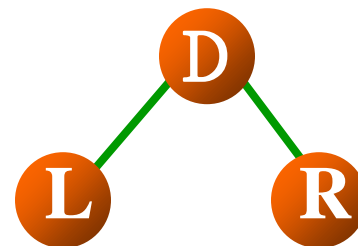


第三节 遍历二叉树

四、后序遍历二叉树

■ 算法：

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

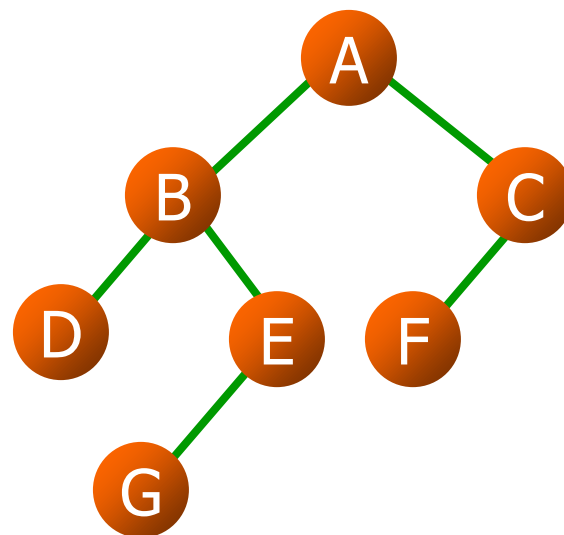


第三节 遍历二叉树

四、后序遍历二叉树

- 算法(举例)：

输出结果：DGEBFCA



第三节 遍历二叉树

四、后序遍历二叉树

```
class BiTree          //二叉树
{
    BiTNode *Tree;    //树根
    void PreOrderTraverse(BiTNode *T);    //先序树T
    void InOrderTraverse(BiTNode *T);    //中序树T
    void PostOrderTraverse(BiTNode *T);    //后序树T

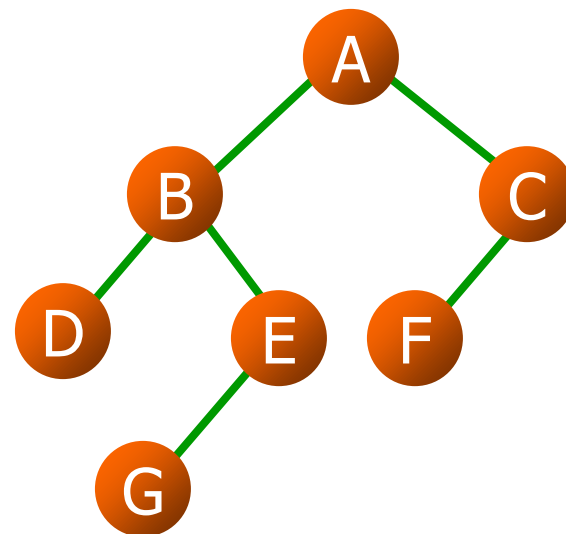
public:
    BiTree():Tree(NULL) {} //树根为空
    void PreOrder();        //先序
    void InOrder();         //后序
    void PostOrder();       //中序
}
```

考虑递归和栈的关系，如何使用栈实现！

第三节 遍历二叉树

五、层次遍历二叉树

输出结果：ABCDEF G

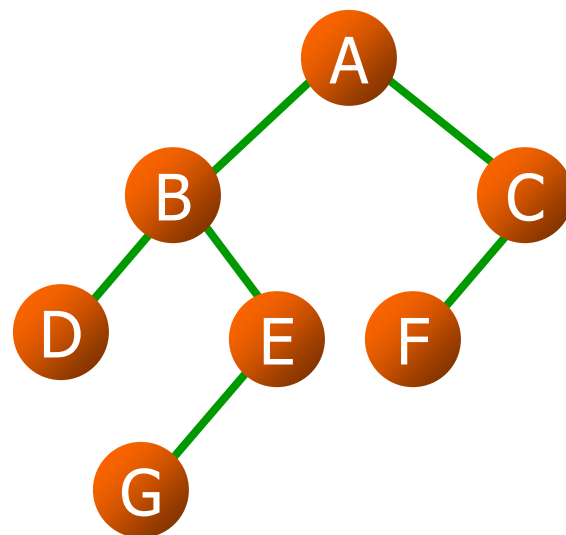


第三节 遍历二叉树

五、层次遍历二叉树

层次遍历可以借助队列实现：

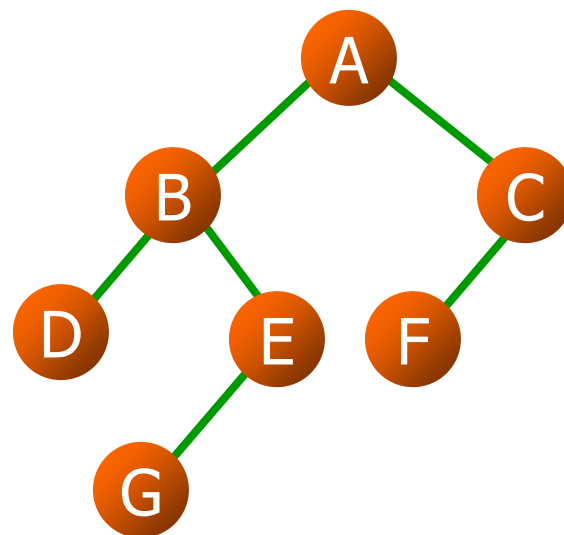
1. 将根结点入队；
2. 判断队列是否为空。若非空，则从队列的头部提取一个结点并访问它，将该结点的左孩子和右孩子依次入队；
3. 重复执行第2步，直至队列为空；



第三节 遍历二叉树

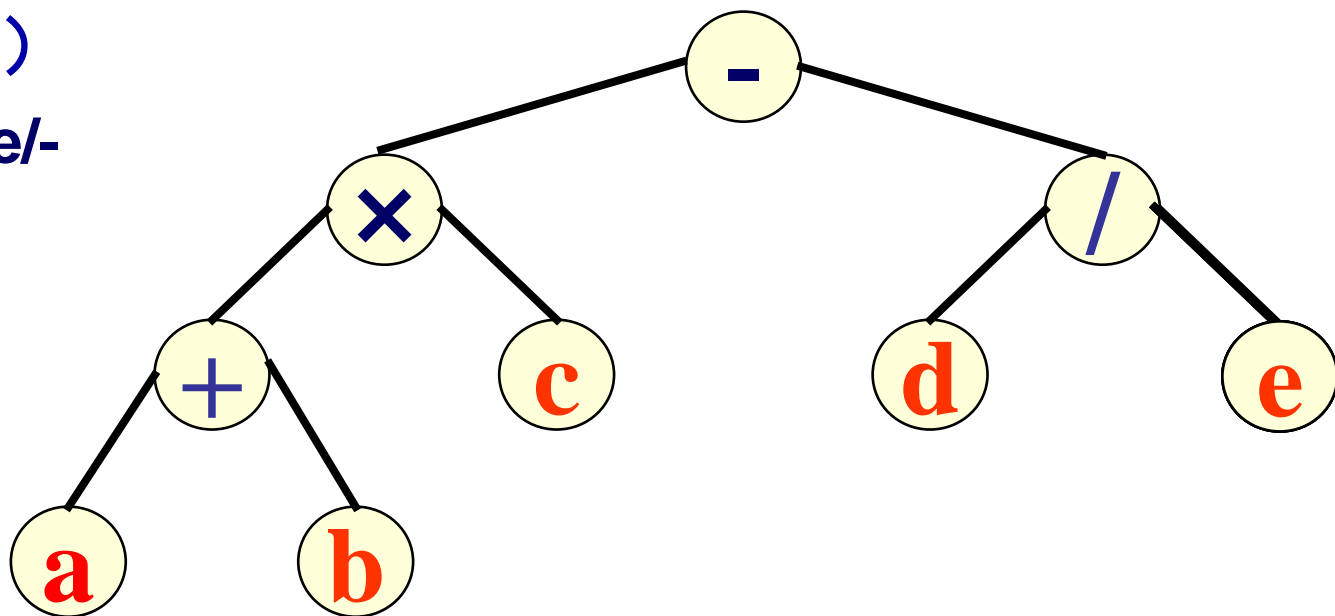
五、层次遍历二叉树

```
void levelTraverse(QueueNode* Q, treeNode* T)
{
    //先进队
    enqueue(Q, T);
    while (!isEmpty(Q))
    {
        treeNode* node = dequeue(Q);
        //将node 进行遍历
        printf("%c ", node->data);
        //如果有左孩子
        if (node->lChild)
        {
            enqueue(Q, node->lChild);
        }
        //如果有右孩子
        if (node->rChild)
        {
            enqueue(Q, node->rChild);
        }
    }
}
```



遍历练习

- 用二叉树来表示表达式
 - 先序遍历得到前缀表达式（波兰式）
 $- \times + a b c / d e$
 - 中序遍历得到中缀表达式
 $(a+b) \times c - d/e$
 - 后续遍历得到后缀表达式（逆波兰式）
 $ab+c \times de/-$



遍历练习

7.【自测题】一棵二叉树满足下列条件:对任一结点,若存在左、右子树,则其值都小于它的左子树上所有结点的值,而大于右子树上所有结点的值。现采用()遍历方式就可以得到这棵二叉树所有结点的递减序列。

- ☐ A.层次
- ☐ B.先序
- ☐ C.后序
- ☐ D.中序

题解:有题可知,此二叉树左子节点大于根节点大于右子节点,故按照左子树、根节点、右子树的顺序遍历可以得到一个递减序列,即中序遍历。



遍历的应用

1. 根据遍历顺序来确定二叉树的结构（逻辑上）
2. 根据先序序列建立二叉树的二叉链表（物理上）

由二叉树的先序和中序序列建树

仅知二叉树的先序序列 “**abcdefg**” 不能唯一确定一棵二叉树，

如果同时已知二叉树的中序序列 “**cbdaegf**”，则会如何？

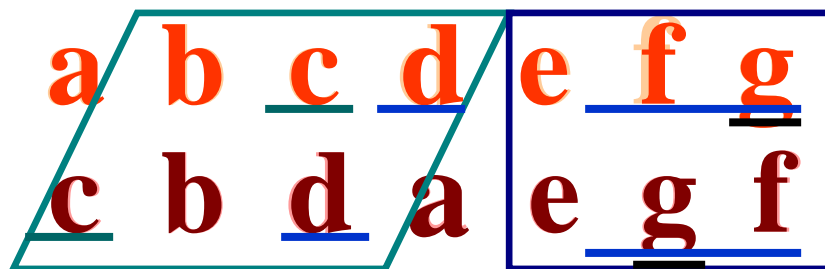
二叉树的先序序列



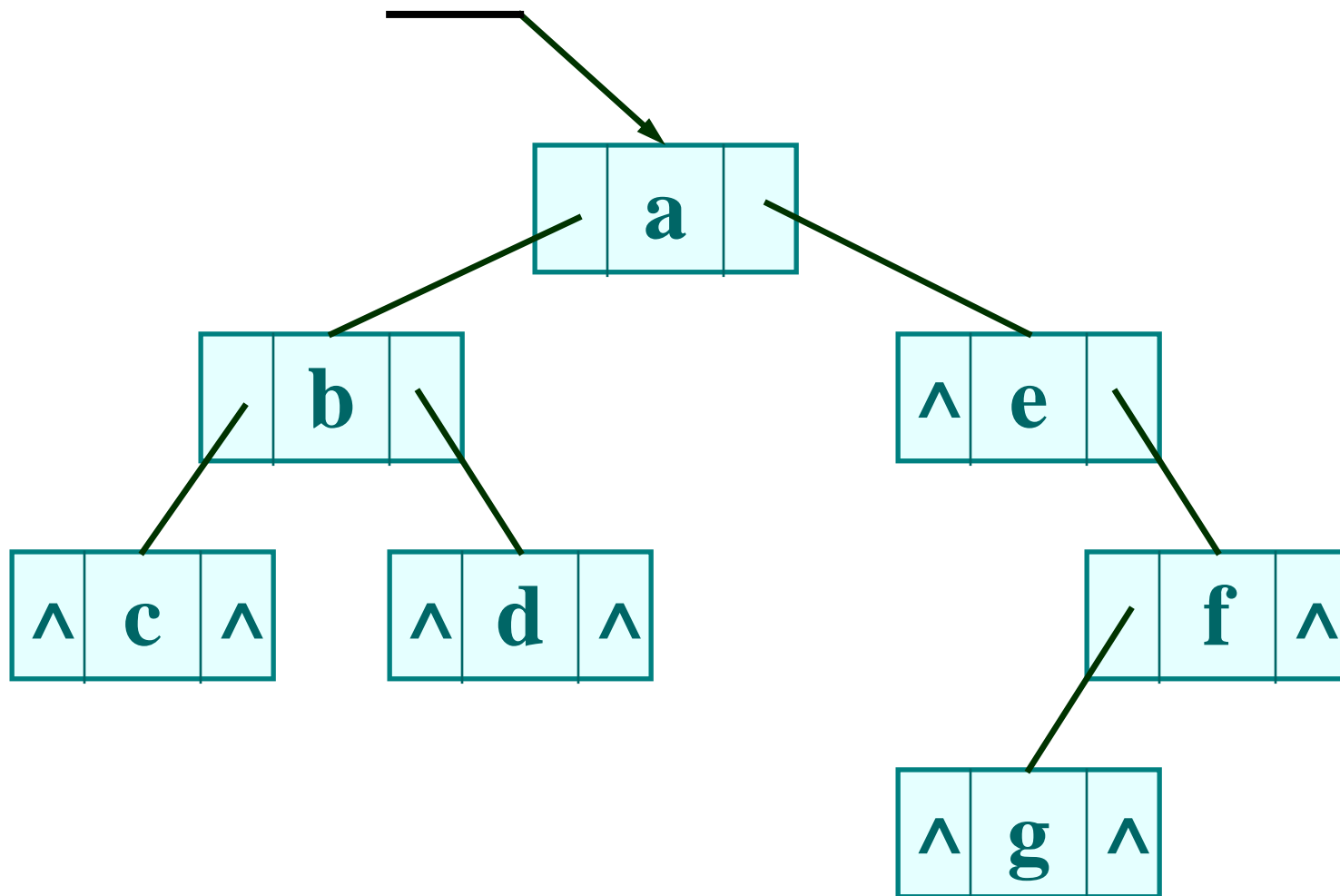
二叉树的中序序列



例:



先序序列
中序序列



作业

- 假设一颗二叉树的先序遍历序列为
ABDEHCFG I
- 中序遍历序列为**DBEHAFCIG**
- 请画出这颗二叉树
- 请写出后序遍历结果

作业

- 思考：如果给出后序遍历序列和中序遍历序列，是否可以确定该二叉树的结构？

假设一颗二叉树的中序：DCBGEAHFIJK
后序：DCEGBFHKJIA,请画出这颗二叉树



建立二叉树的存储结构

不同的定义方法相应有不同的存储结构的建立算法
在此讨论利用先序序列，建立二叉链表

以先序序列创建一棵二叉树

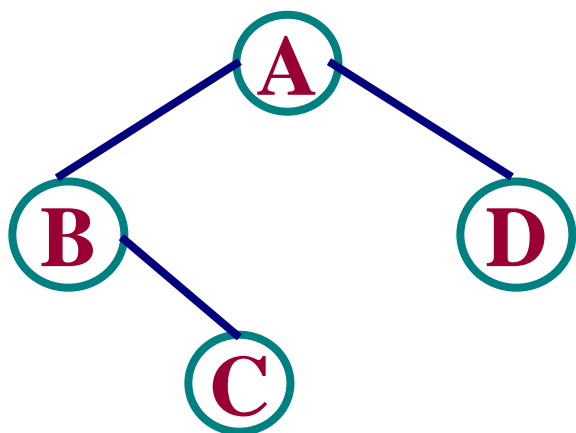
空树

以空白字符 " " 表示

只含一个根结点的
二叉树



以字符串 "A " 表示



以下列字符串表示

$A(\underline{B(\underline{\quad}, \underline{C(\underline{\quad}, \underline{\quad})})}, \underline{D(\underline{\quad}, \underline{\quad})})$


```
void BiTree::CreateTree(BiTNode * T) //私有方法，先序遍历创建二叉树
{
    char    ch;

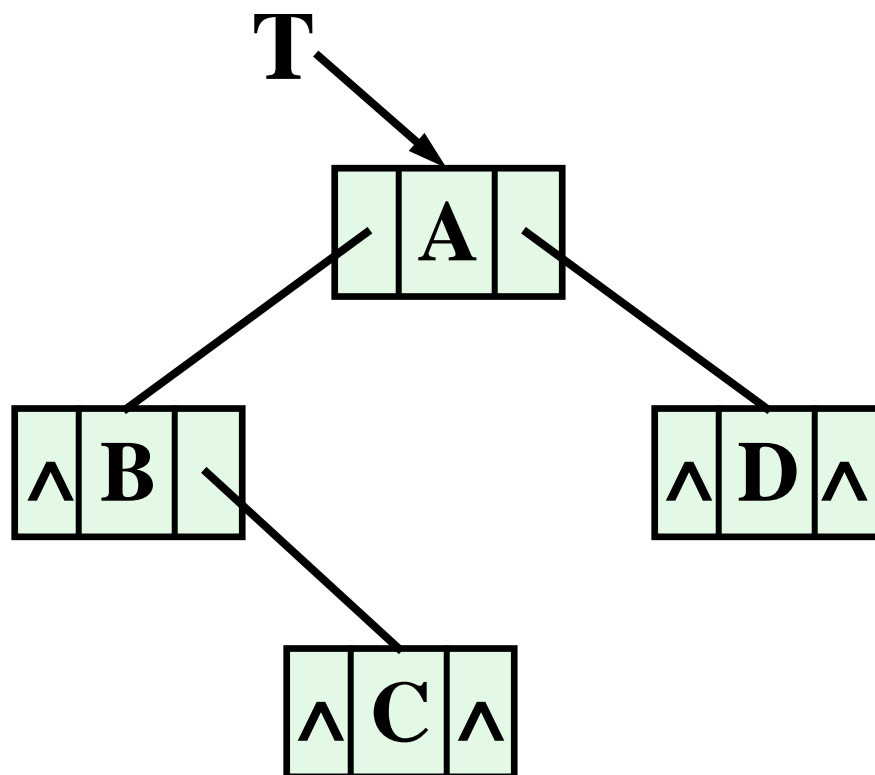
    cin>>ch;

    if(ch!=' ')
    {
        T = new BiTNode(ch); //生成新结点
        CreateTree(T->lchild); //创建左子树
        CreateTree(T->rchild); //创建右子树
    }
    else
        T = NULL;
}

void BiTree::CreateTree() //公有接口
{
    CreateTree(Tree);
}
```

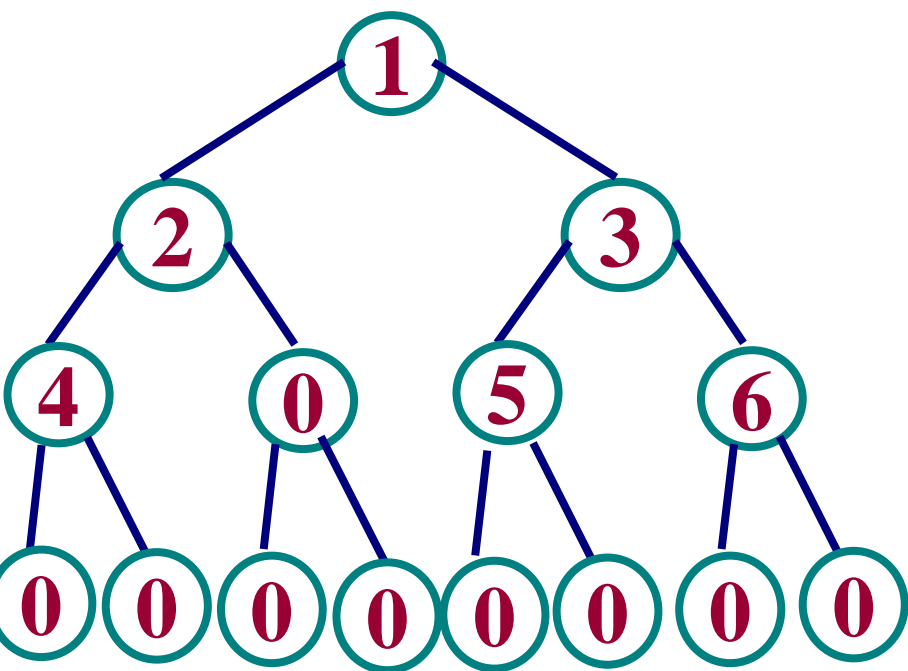
上页算法执行过程举例如下：

A B ■ C ■ ■ D ■ ■



以层次序列创建一棵二叉树（使用队列）

对应的读入是：1 2 3 4 0 5 6 0 0 0 0 0 0 （0是空节点）



```
void Create_Node(Node* t, int x){
    t=new Node;
    t->data=x;
    t->leftchild=NULL;
    t->rightchild=NULL;
}
```

```
void Create_Level(Node* t){
    queue<Node*> q;
    int x;
    cin>>x;
    if (x!=0) {
        Create_Node(t,x);
        q.push(t);
    }
    while (!q.empty()){
        Node* s=q.front();
        cin>>x;
        if (x!=0){
            Create_Node(s->leftchild,x);
            q.push(s->leftchild);
        }
        cin>>x;
        if (x!=0){
            Create_Node(s->rightchild,x);
            q.push(s->rightchild);
        }
        q.pop();
    }
}
```



6.4 线索二叉树

第四节 线索二叉树

问题：遍历是将非线性结构线性化，结点的先后关系的信息（某个结点在序列中的前驱和后继等信息）在遍历的动态过程中得到，如何保存这些动态信息呢？

第四节 线索二叉树

一、增加新指针

- 最简单的方法是在每个结点中，增加前驱(fwd)和后继(bkwd)指针

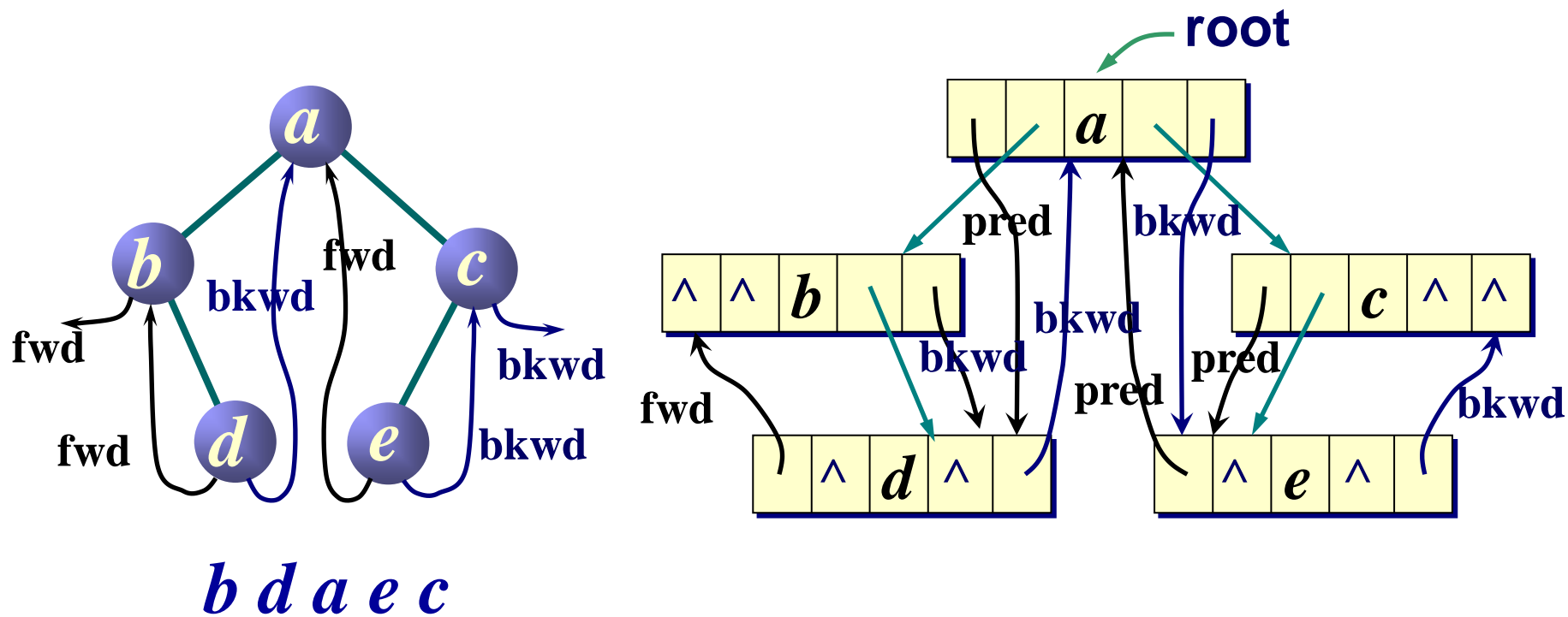
| | | | | |
|-----|--------|------|--------|------|
| fwd | lChild | data | rChild | bkwd |
|-----|--------|------|--------|------|

- 在做二叉树遍历（前、中、后序），将每个结点的前驱和后继信息添入fwd和bkwd域中

第四节 线索二叉树

| fwd | leftChild | data | rightChild | bkwd |
|-----|-----------|------|------------|------|
|-----|-----------|------|------------|------|

中序遍历的二叉链表：



第四节 线索二叉树

二、利用空指针

- 在有 n 个结点的二叉树中，必定存在 $n+1$ 个空链域
- 因为每个结点有两个链域（左、右孩子指针），因此共有 $2n$ 个链域
- 除根结点外，每个结点都有且仅有一个分支相连，即 $n-1$ 个链域被使用

第四节 线索二叉树

二、利用空指针

- 在结点中增加两个标记位 (LTag, RTag)

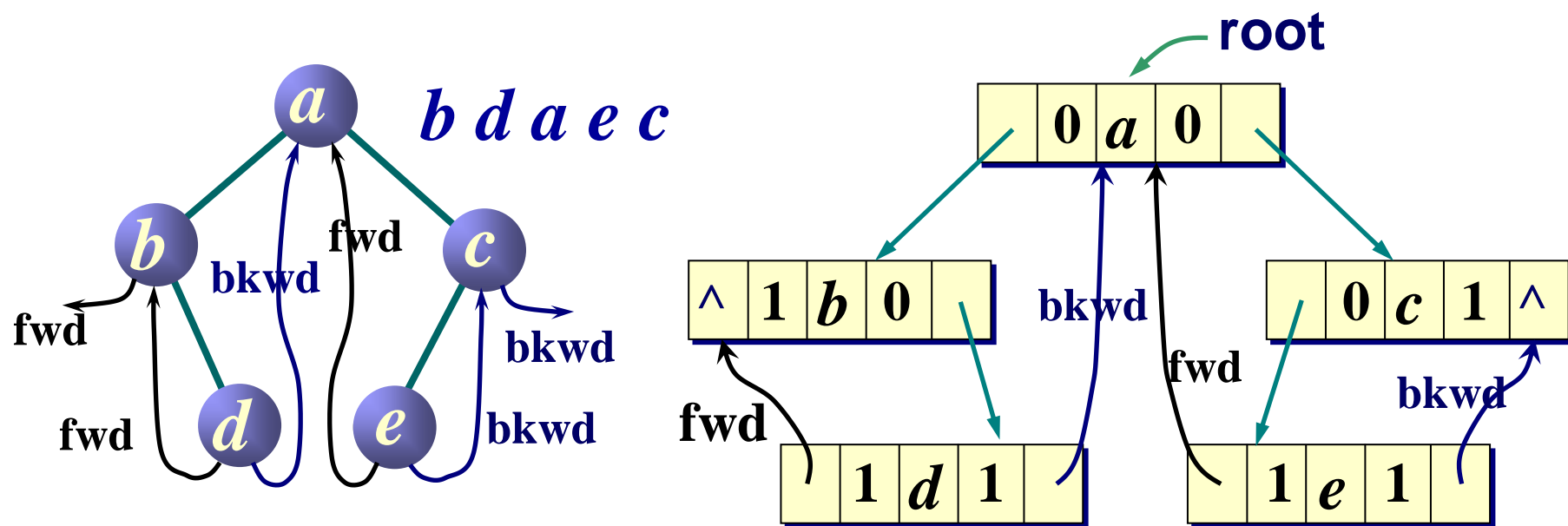
| | | | | |
|--------|------|------|------|--------|
| lChild | LTag | data | RTag | rChild |
|--------|------|------|------|--------|

- LTag=0, lChild域指示结点的左孩子
LTag=1, lChild域指示结点的前驱结点
- RTag=0, rChild域指示结点的右孩子
RTag=1, rChild域指示结点的后继结点

第四节 线索二叉树

二、利用空指针

| lChild | lTag | data | rTag | rChild |
|--------|------|------|------|--------|
|--------|------|------|------|--------|



ltag = 0, *leftChild* 为左子女指针

ltag = 1, *leftChild* 为前驱线索

rtag = 0, *rightChild* 为右子女指针

rtag = 1, *rightChild* 为后继线索



6.5 树与森林

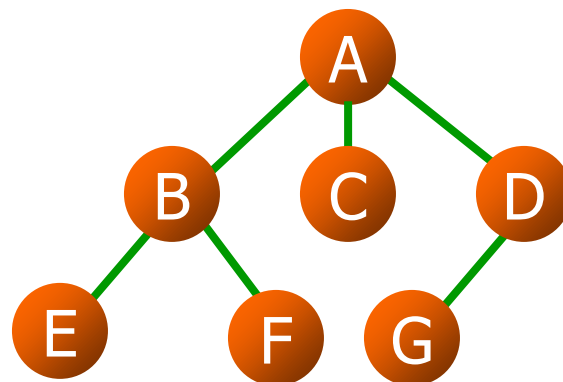
第五节 树与森林

一、树的存储结构

1. 双亲表示法

- 采用一组连续的存储空间
- 由于每个结点只有一个双亲，只需要一个指针

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|----|---|---|---|---|---|---|
| A | B | C | D | E | F | G |
| -1 | 0 | 0 | 0 | 1 | 1 | 3 |

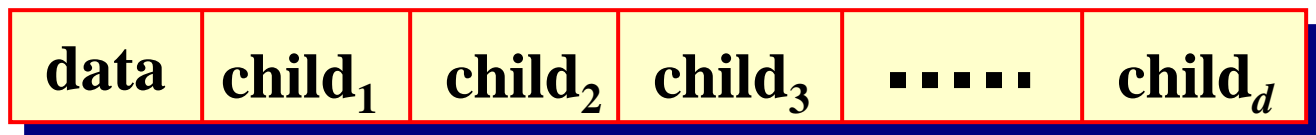


第五节 树与森林

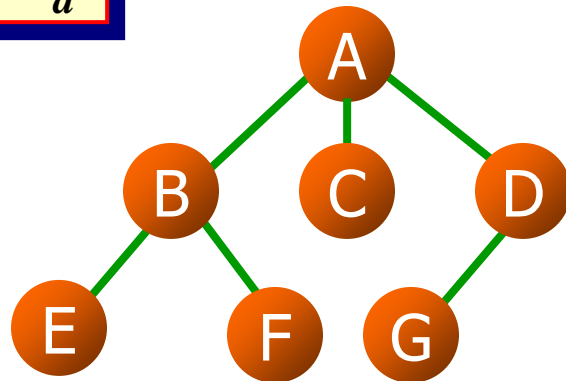
一、树的存储结构

2. 孩子表示法

- 可以采用多重链表，即每个结点有多个指针



- 最大缺点是空链域太多
[$(d-1)n+1$ 个]

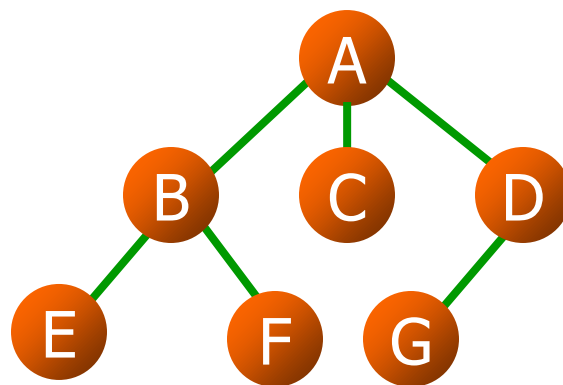
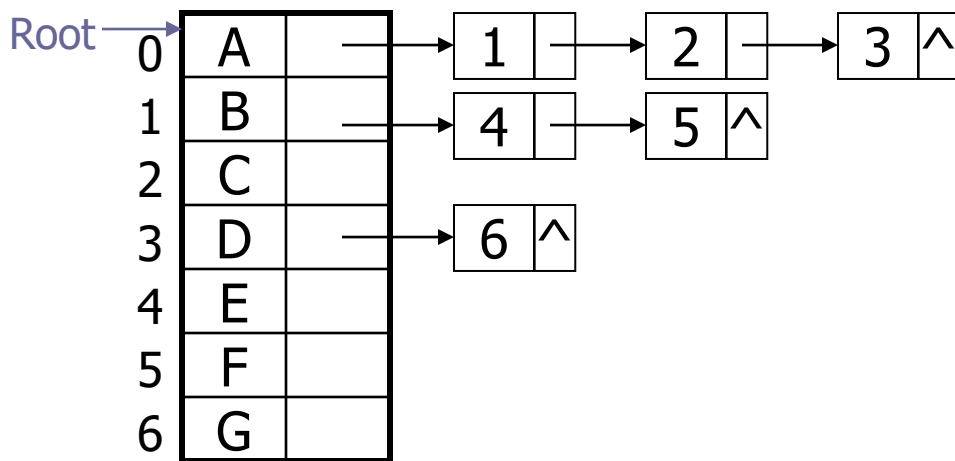


第五节 树与森林

一、树的存储结构

2. 孩子表示法

- 将每个结点排列成一个线性表
- 将每个结点的孩子排列起来，用单链表表示



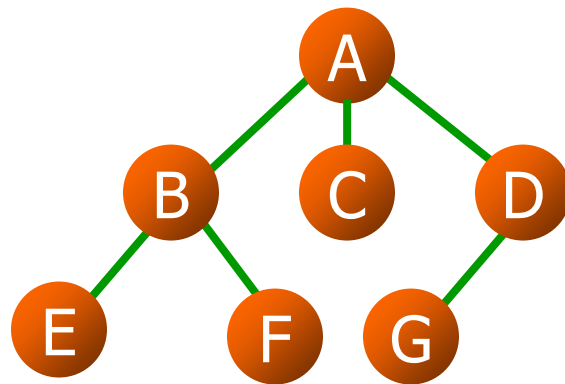
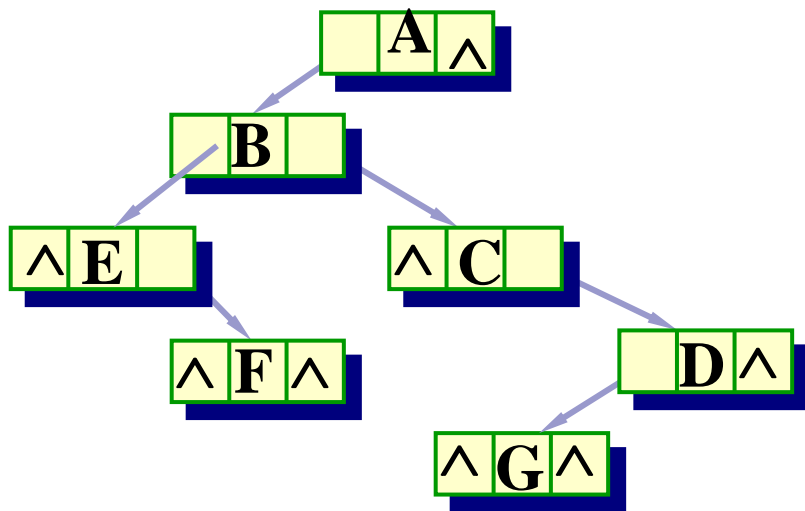
第五节 树与森林

一、树的存储结构

3. 孩子兄弟表示法

- 采用二叉链表
- 左边指针指向第一个孩子，右边指针指向兄弟

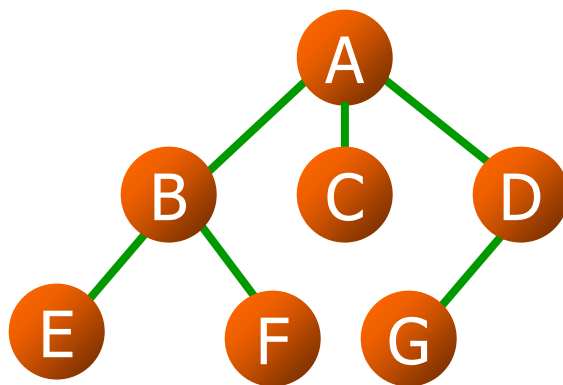
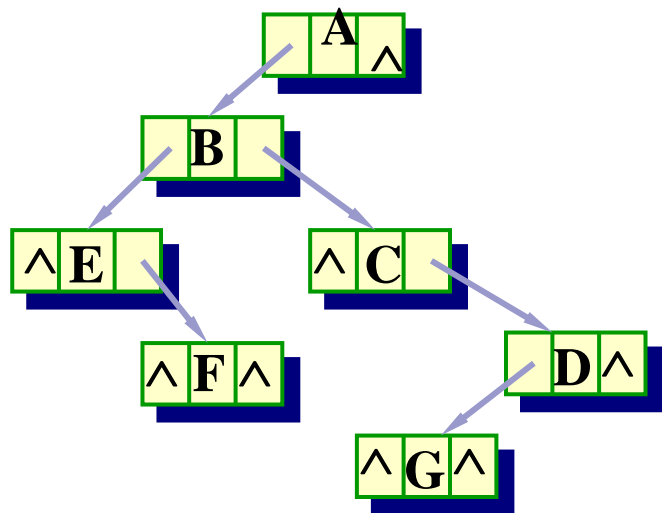
| data | firstChild | nextSibling |
|------|------------|-------------|
|------|------------|-------------|



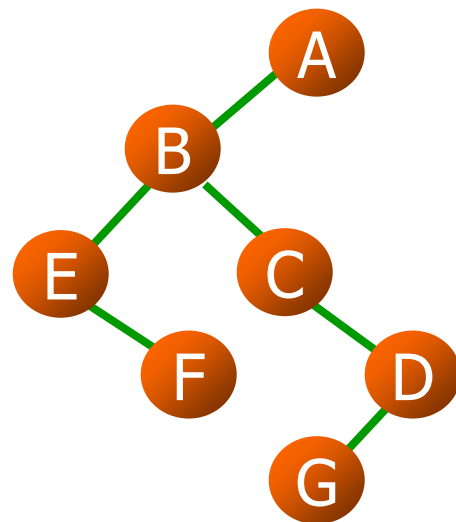
第五节 树与森林

二、树与二叉树的对应关系

- 树与二叉树都可以采用二叉链表作存储结构
- 任意给定一棵树，可以找到一个唯一的二叉树
(没有右子树)



树



对应的 二叉树



树与二叉树的转换:

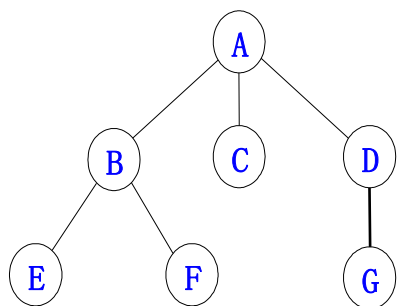
以二叉链表作为存储结构，将其解释为树或二叉树，实现两者之间的转换。

树与二叉树的转换

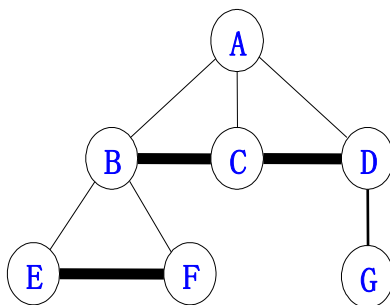
1. 树转换为二叉树

树转换为二叉树的方法是：

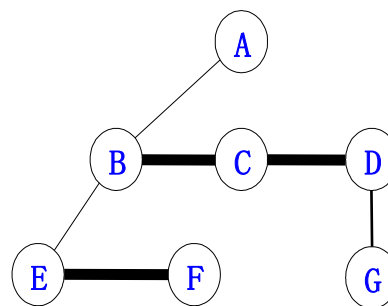
- (1) 树中所有相同双亲结点的兄弟结点之间加一条连线。**
- (2) 对树中不是双亲结点第一个孩子的结点，只保留新添加的该结点与左兄弟结点之间的连线，删去该结点与双亲结点之间的连线。**
- (3) 整理所有保留的和添加的连线，使每个结点的第一个孩子结点连线位于左孩子指针位置，使每个结点的右兄弟结点连线位于右孩子指针位置。**



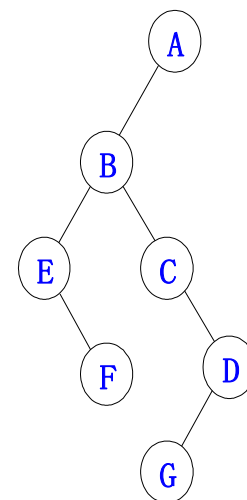
(a)



(b)



(c)



(d)

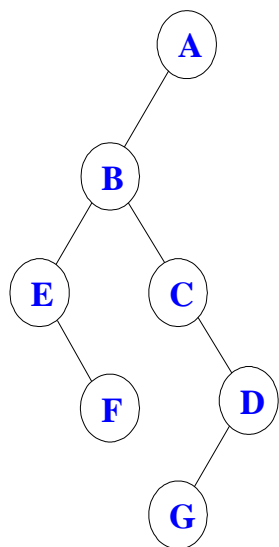
树转换为二叉树的过程

(a) 树; (b) 相邻兄弟加连线; (c) 删除双亲与非第一个孩子连线; (d) 二叉树

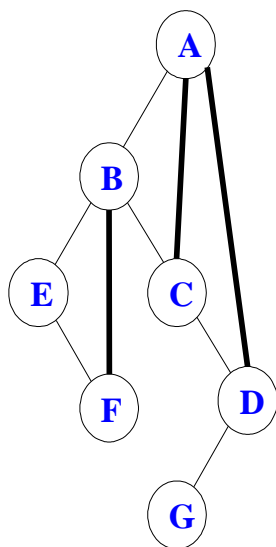
2. 二叉树还原为树

二叉树还原为树的方法是：

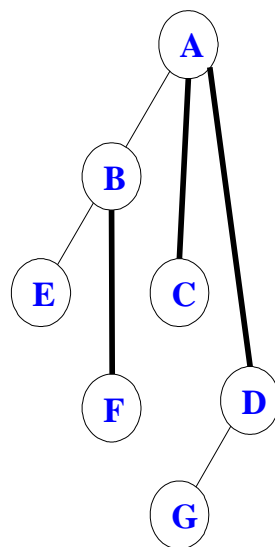
- (1) 若某结点是其双亲结点的左孩子，则把该结点的右孩子、右孩子的右孩子……都与该结点的双亲结点用线连起来。
- (2) 删除原二叉树中所有双亲结点与右孩子结点的连线。
- (3) 整理所有保留的和添加的连线，使每个结点的所有孩子结点位于相同层次高度。



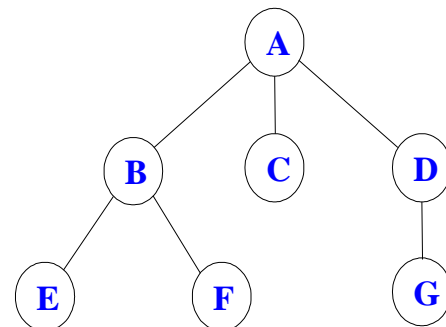
(a)



(b)



(c)



(d)

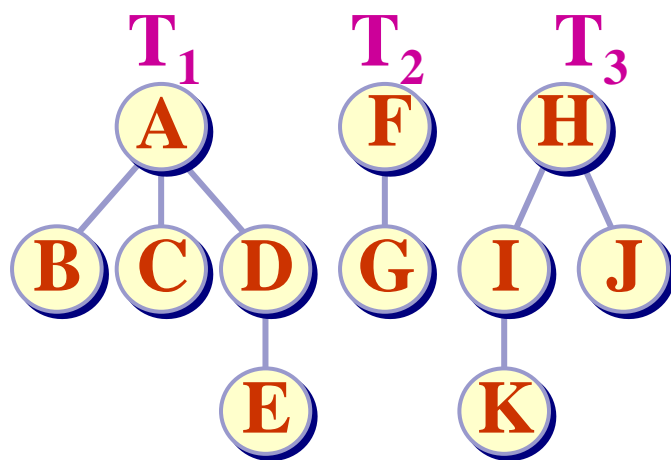
二叉树还原为树的过程

(a) 二叉树; (b) 双亲与非第一个孩子加连线;
(c) 删除结点与右孩子连线; (d) 树

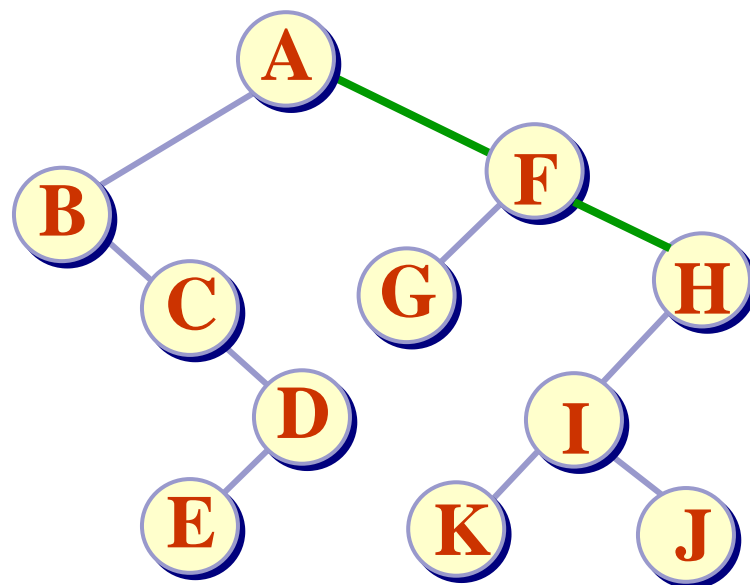
第五节 树与森林

三、森林与二叉树的对应关系

- 如果把森林中的第二棵树的根结点看作是第一棵树的根结点的兄弟，则可找到一个唯一的二叉树与之对应



三棵树的森林



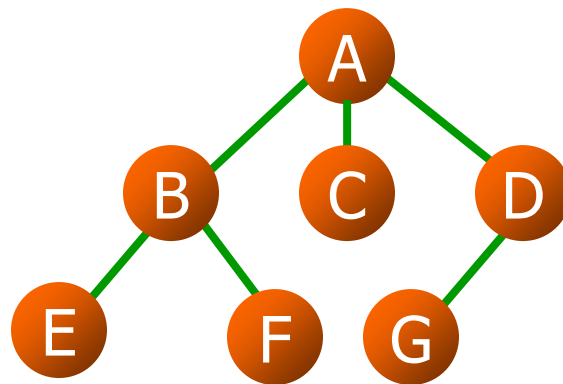
对应的二叉树

第五节 树与森林

四、树的遍历

■ 对树的遍历主要有两种：

1. 先根（次序）遍历
2. 后根（次序）遍历



第五节 树与森林

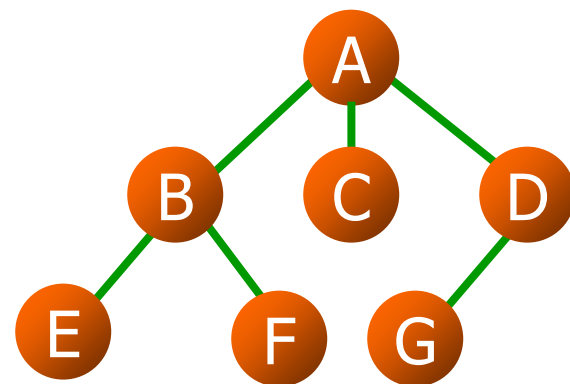
四、树的遍历

1. 先根（次序）遍历

当树非空时

- 访问根结点
- 依次先根遍历根的各棵子树

输出结果：ABEFCDG



第五节 树与森林

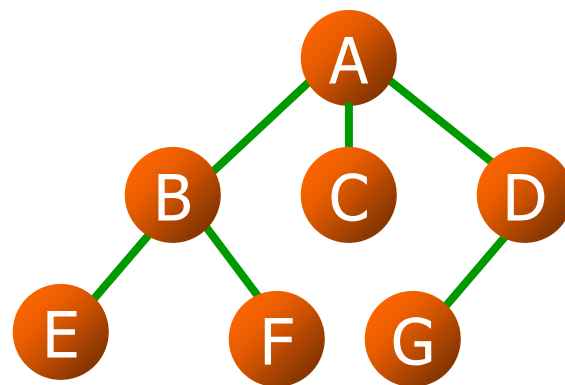
四、树的遍历

2. 后根（次序）遍历

当树非空时

- 依次后根遍历根的各棵子树
- 访问根结点

输出结果：EFBCGDA



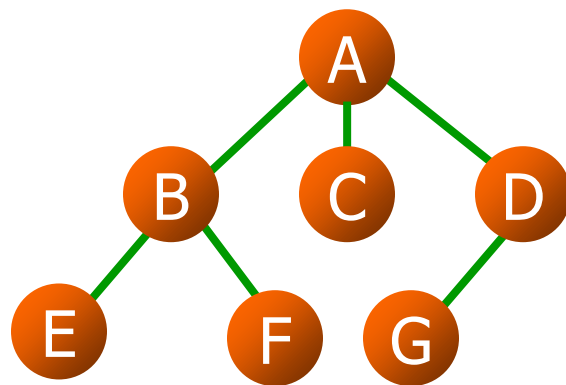
第五节 树与森林

五、森林的遍历

■ 对森林的遍历主要有两种：

1. 先序遍历

2. 中序遍历



第五节 树与森林

五、森林的遍历

1. 先序遍历

若森林不空，则

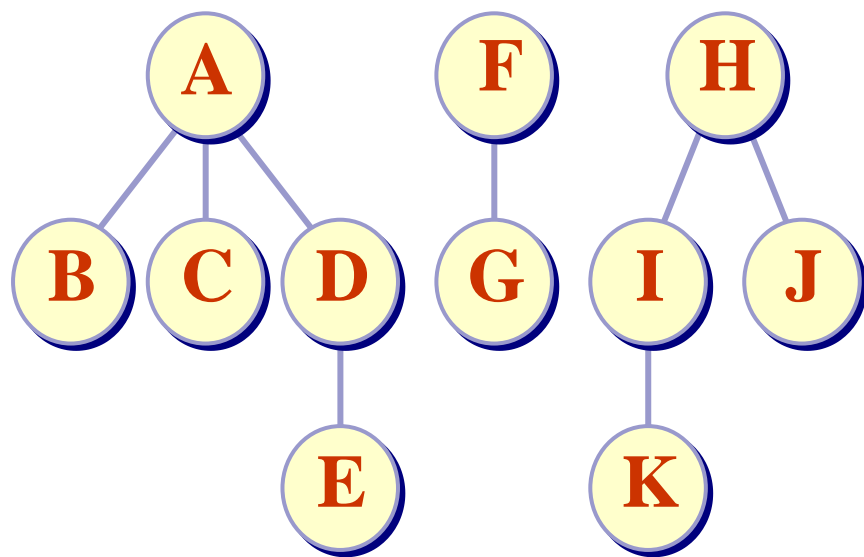
- 访问森林中第一棵树的根结点；
- 先序遍历森林中第一棵树的子树森林；
- 先序遍历森林中(除第一棵树之外)其余树构成的森林。

即：依次从左至右对森林中的每一棵树进行先根遍历。

第五节 树与森林

五、森林的遍历

1. 先序遍历



先序遍历：
ABCDEFGHIKJ

第五节 树与森林

五、森林的遍历

2. 中序遍历

若森林不空，则

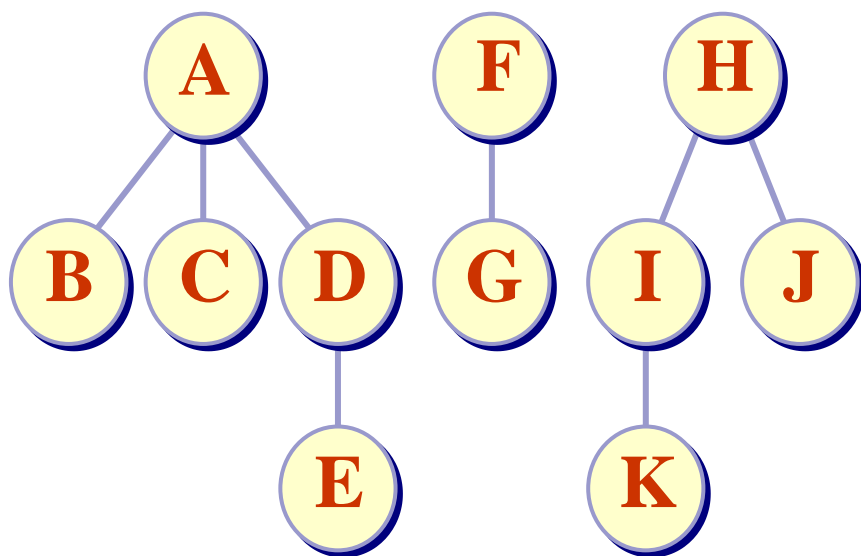
- 中序遍历森林中第一棵树的子树森林；
- 访问森林中第一棵树的根结点；
- 中序遍历森林中(除第一棵树之外)其余树构成的森林。

即：依次从左至右对森林中的每一棵树进行后根遍历。

第五节 树与森林

五、森林的遍历

2. 中序遍历



中序遍历：
BCEDAGFK I JH

思考？

- 当树以二叉链表(孩子兄弟表示法)进行存储，则树的先根遍历和后根遍历与二叉树的遍历算法有什么关系？

树的先根-----二叉树的先序

树的后根-----二叉树的中序

遍历的对应关系

树

森林

二叉树

先根遍历

先序遍历

先序遍历

后根遍历

中序遍历

中序遍历



练习

1. 二叉树的中序和后序遍历结果分别为**EFBCGHIDA**和**FEIHGDCBA**。

- (1) 画出该树，求其先序遍历的结果；
- (2) 将其转换为树，画出转换后的树。
- (3) 求其先根、后根遍历序列。

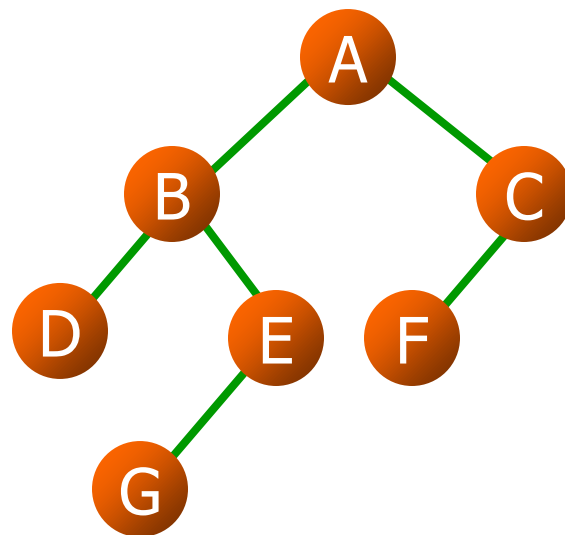


6.6 赫夫曼树及其应用

第六节 赫夫曼树及其应用

一、最优二叉树（赫夫曼树）

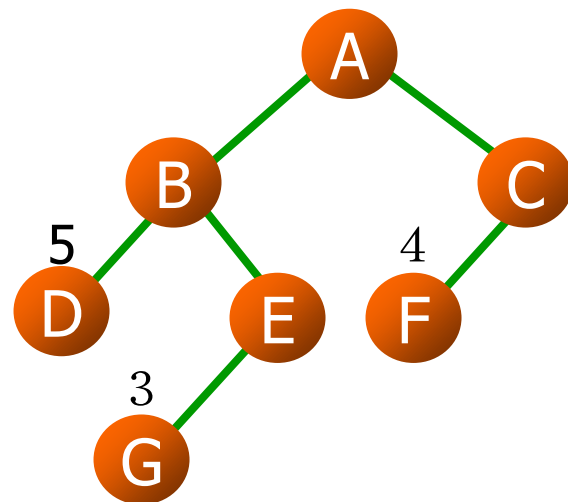
- 路径：从树中一个结点到另一个结点之间的分支构成这两个结点之间的路径
- 路径长度：路径上的分支数目
- 树的路径长度：从树根到每个结点的路径长度之和
- 右树路径长度为：
 $2*1 + 3*2 + 1*3 = 11$



第六节 赫夫曼树及其应用

一、最优二叉树

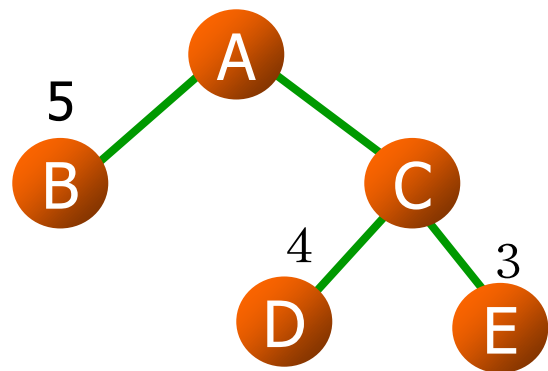
- 结点的带权路径长度：从结点到树根之间的路径长度与结点上权的乘积
- 树的带权路径长度 (WPL)：树中所有叶子结点的带权路径长度之和
- $WPL = 2*5+3*3+2*4=27$



第六节 赫夫曼树及其应用

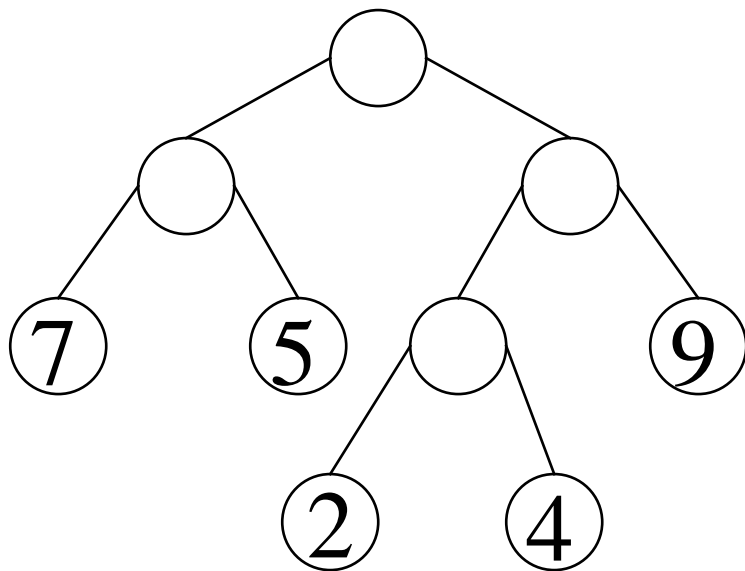
一、最优二叉树

- 最优二叉树：假设二叉树有 n 个叶子，其每个叶子结点带权 w_i ，则带权路径长度WPL最小的二叉树称为最优二叉树或赫夫曼(Huffman)树。
- $WPL = 1*5+2*3+2*4=19$

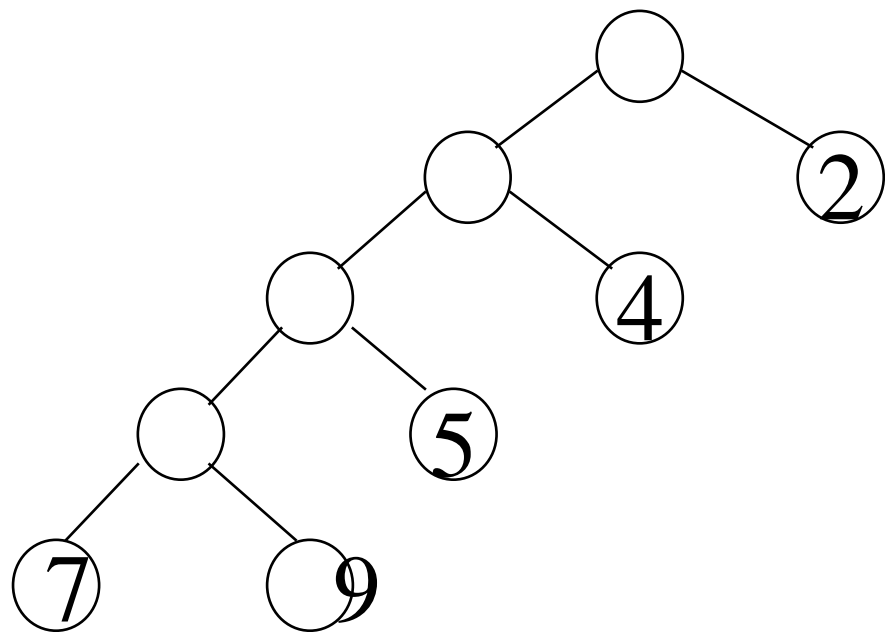


第六节 赫夫曼树及其应用

例： 设5个叶子结点，分别带权7、9、5、4、2，
至少画出2种二叉树，并计算带权路径长度。



$$\begin{aligned}
 \text{WPL}(T) &= \\
 7 \times 2 &+ 5 \times 2 + 2 \times 3 + 4 \times 3 \\
 &+ 9 \times 2 \quad = 60
 \end{aligned}$$



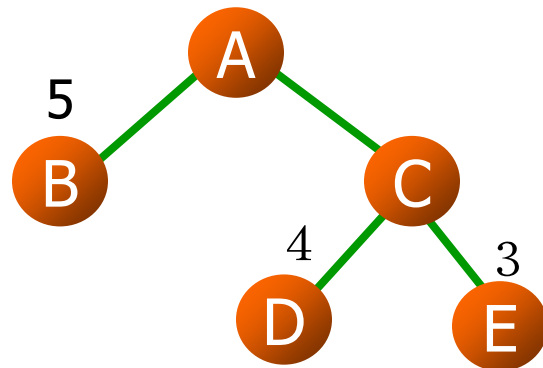
$$\begin{aligned}
 \text{WPL}(T) &= \\
 7 \times 4 &+ 9 \times 4 + 5 \times 3 + 4 \times 2 + 2 \times 1 \\
 &= 89
 \end{aligned}$$



第六节 赫夫曼树及其应用

二、Huffman树(构造)

- 在Huffman树中，权值最大的结点离根最近
- 权值最小的结点离根最远



第六节 赫夫曼树及其应用

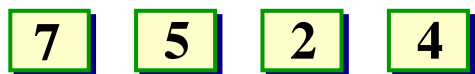
二、Huffman树(算法)

1. 根据给定的 n 个权值(w_1, w_2, \dots, w_n)构成 n 棵二叉树的集合 $F = \{T_1, T_2, \dots, T_n\}$, 其中每棵二叉树 T_i 中只有一个带权为 w_i 的根结点, 左右子树为空
2. 在 F 中选取两棵根结点的权值最小的树作为左右子树构造一棵新的二叉树, 且置其根结点的权值为其左右子树根结点的权值之和
3. 在 F 中删除这两棵树, 同时将新得到的二叉树加入 F 中
4. 重复2, 3, 直到 F 只含一棵树为止

第六节 赫夫曼树及其应用

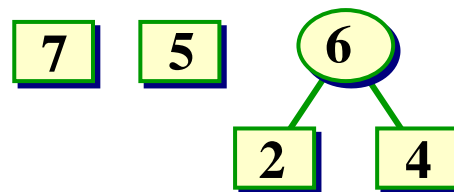
二、Huffman树 (举例)

F : {7} {5} {2} {4}



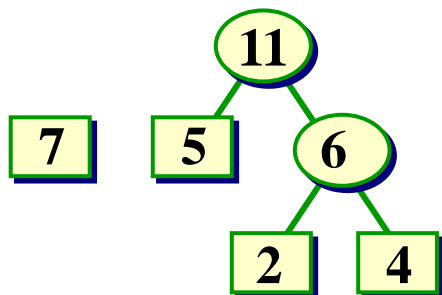
初始

F : {7} {5} {6}



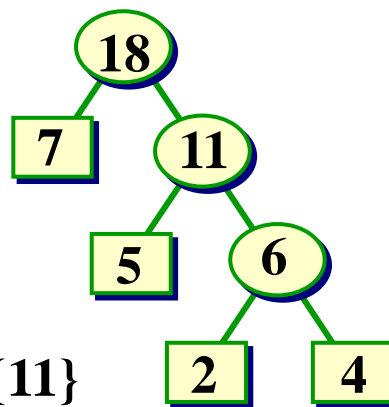
合并{2} {4}

F : {7} {11}



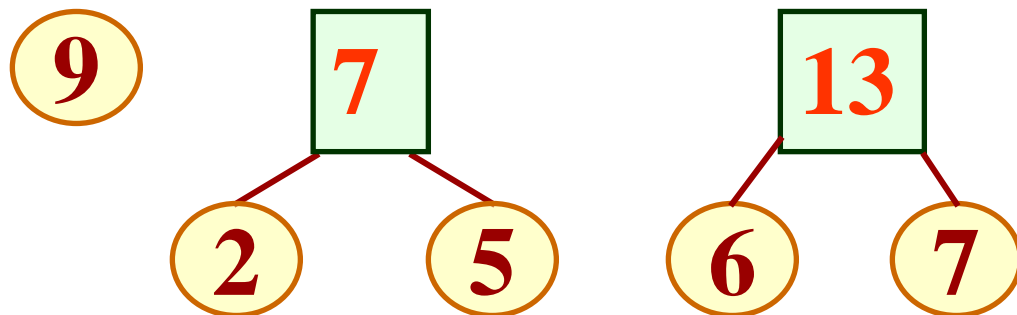
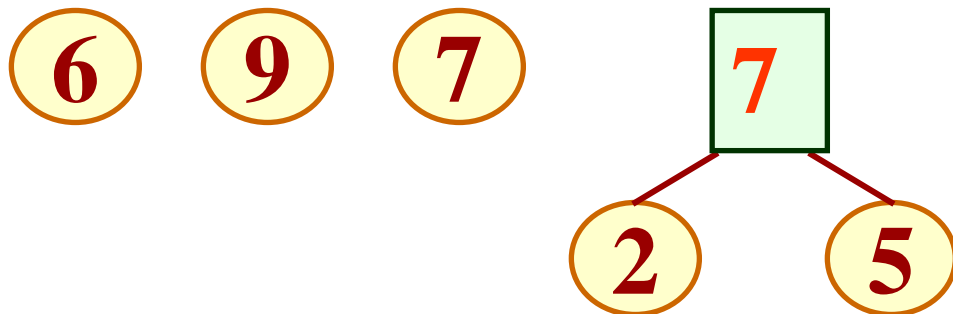
合并{5} {6}

F : {18}

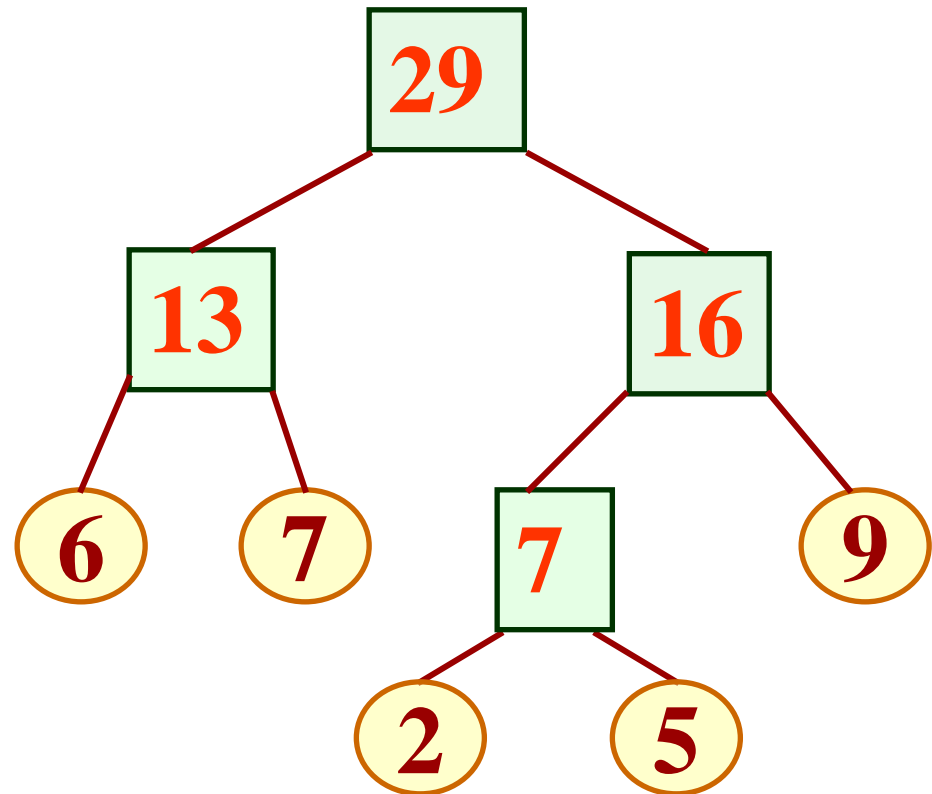
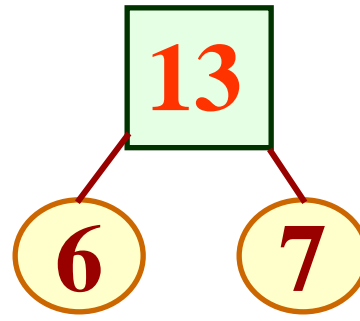
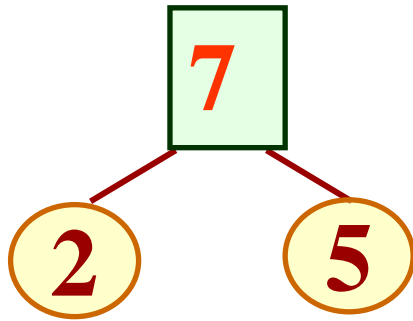


合并{7} {11}

例如: 已知权值 $W=\{ 5, 6, 2, 9, 7 \}$



9



第六节 赫夫曼树及其应用

作业： 设叶子结点a、b、c、d、e的权值分别为5、4、10、12、9，构造Huffman树，并计算带权路径长度。

第六节 赫夫曼树及其应用

三、Huffman编码

- 设给出一段报文: GOOD_GOOD_GOOD_G00000000_OFF
- 字符集合是 { 0, G, _, D, F }, 各个字符出现的频度(次数)是 $W = \{ 15, 4, 4, 3, 2 \}$ 。
- 若给每个字符以等长编码
0: 000 G: 001 _: 010 D: 011 F: 100
- 则总编码长度为 $(15+4+4+3+2) * 3 = 84$.

第六节 赫夫曼树及其应用

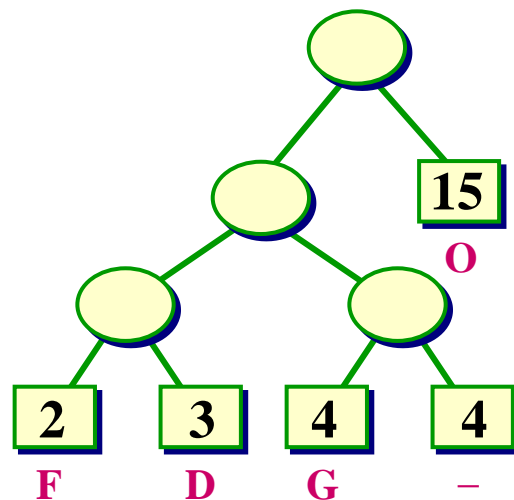
三、Huffman编码

- 若按各个字符出现的概率不同而给予不等长编码，可望减少总编码长度。
- 各字符 { 0, G, _, D, F } 出现概率为
{ 15/28, 4/28, 4/28, 3/28, 2/28 }, 化整为
{ 15, 4, 4, 3, 2 }

第六节 赫夫曼树及其应用

三、Huffman编码

- 各字符出现概率[取整数]为{15, 4, 4, 3, 2}
- 如果规定, Huffman树的左子树小于右子树, 则可构成右图所示Huffman树

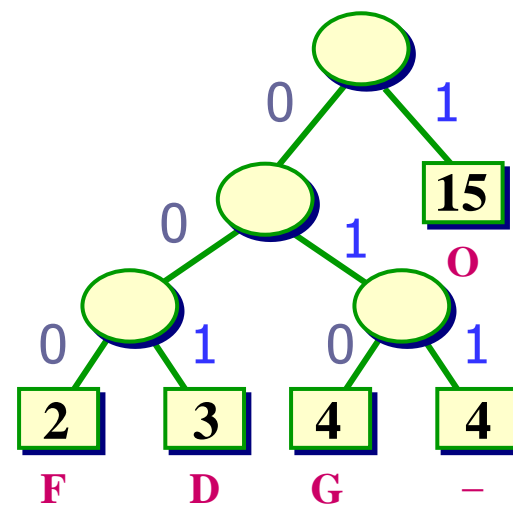


第六节 赫夫曼树及其应用

三、Huffman编码

- 令左孩子分支为编码 ‘0’，右孩子分支为编码 ‘1’
- 将根结点到叶子结点路径上的分支编码，组合起来，作为该字符的Huffman码，则可得得到：

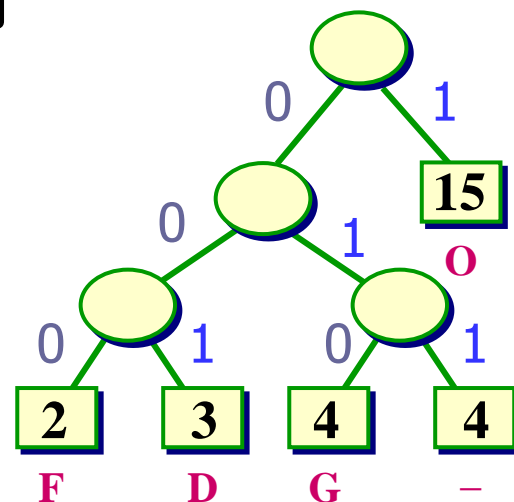
0:1 _:011 G:010 D:001
F:000



第六节 赫夫曼树及其应用

三、Huffman编码

- 则总编码长度为 $15*1 + (2+3+4+4)*3 = 54 < 84$
- Huffman是一种前缀编码，解码时不会混淆
- 如GOOD编码为：01011001



前缀编码

若设计长短不等的编码，则必须是任一个字符的编码都不是另一个字符编码的前缀，称为前缀编码。

利用**赫夫曼树**可以构造一种不等长的二进制编码，并且构造所得的赫夫曼编码是一种**最优前缀编码**，即使**所传电文的总长度最短**。



第六节 赫夫曼树及其应用

练习：对电文 ‘abbaccdeeccdabcc ‘ 进行 Huffman 编码。

第六节 赫夫曼树及其应用

四、Huffman编码实现

Huffman树的构造。用静态链表实现, 结构如下。

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

n 个叶子结点的Huffman树共 $2n-1$ 个结点。(两两合并, 直至一个, 共生成 $n-1$ 个结点)

设静态链表为HT，计算过程如下：

① 初始令 n =叶子结点数，设置 $0 \sim n-1$ 结点的data、weight，并令所有单元的parent, lchild, rchild为0

② 从 $0 \sim n-1$ 中选取parent为0，权值最小的两个结点，设为 $s1, s2$ ，令 n 为此两结点的父结点，修改：

$HT[s1].parent = n; \quad HT[s2].parent = n;$

$HT[n].lchild = s1; \quad HT[n].rchild = s2;$

$HT[n].weight = HT[s1].weight + HT[s2].weight;$

$HT[n].parent = 0;$

$n++;$

③ 重复②直至 $n=2*叶子结点数-1$

第六节 赫夫曼树及其应用

二、Huffman树(定义结点[双亲孩子])

```
#define MAXCODELEN    15    // 每个字符编码的最大长度
#define MAXHUFFMANCODENO 10 // 编码的字符的最大数目
#define MAXCODESTRINGLEN 100// 编码字符串最大长度
typedef struct {
    char c;                // 字符
    int weight;            // 权值
    int parent, lchild, rchild; // 双亲、左右孩子指针
    char code[MAXCODELEN]; // 字符对应的Huffman编码
} HTNode, *HuffmanTree;   // 定义Huffman树的结点
HTNode HT[2*MAXHUFFMANCODENO];
```

以电文 ‘**ABACCD**A’的Huffman树构造为例说明静态链表的变化。

0表示无双亲无孩子

➡ 初始

| | Data | weight | Parent | Left Child | Right Child |
|---|------|--------|--------|------------|-------------|
| 0 | A | 3 | 0 | 0 | 0 |
| 1 | B | 1 | 0 | 0 | 0 |
| 2 | C | 2 | 0 | 0 | 0 |
| 3 | D | 1 | 0 | 0 | 0 |
| 4 | | | | | |
| 5 | | | | | |
| 6 | | | | | |

- ✎ 选取表中0~3中双亲为0的权值最小的两个结点，第4单元做此两结点的双亲，修改双亲、左右孩子、权值。

| | Data | weight | Parent | Left Child | Right Child |
|---|------|--------|--------|------------|-------------|
| 0 | A | 3 | 0 | 0 | 0 |
| 1 | B | 1 | 4 | 0 | 0 |
| 2 | C | 2 | 0 | 0 | 0 |
| 3 | D | 1 | 4 | 0 | 0 |
| 4 | | 2 | 0 | 1 | 3 |
| 5 | | | | | |
| 6 | | | | | |

- ✎ 选取表中0~4中双亲为0的权值最小的两个结点，第5单元做此两结点的双亲，修改双亲、左右孩子、权值。

| | Data | weight | Parent | Left Child | Right Child |
|---|------|--------|--------|------------|-------------|
| 0 | A | 3 | 0 | 0 | 0 |
| 1 | B | 1 | 4 | 0 | 0 |
| 2 | C | 2 | 5 | 0 | 0 |
| 3 | D | 1 | 4 | 0 | 0 |
| 4 | | 2 | 5 | 1 | 3 |
| 5 | | 4 | 0 | 2 | 4 |
| 6 | | | | | |

- ☞ 选取表中0~5中双亲为0的权值最小的两个结点，第6单元做此两结点的双亲，修改双亲、左右孩子、权值。完成。

| | Data | weight | Parent | Left Child | Right Child |
|---|------|--------|--------|------------|-------------|
| 0 | A | 3 | 7 | 0 | 0 |
| 1 | B | 1 | 4 | 0 | 0 |
| 2 | C | 2 | 5 | 0 | 0 |
| 3 | D | 1 | 4 | 0 | 0 |
| 4 | | 2 | 5 | 1 | 3 |
| 5 | | 4 | 7 | 2 | 4 |
| 6 | | 7 | 0 | 0 | 5 |

第六节 赫夫曼树及其应用

二、Huffman树(程序)

```
void CreateHuffmanTree(int n, char *c, int *w)
{
    int i, m, s1, s2;
    m = 2*n - 1;
    for (i=1; i<=n; ++i) { // 为Huffman树的叶子结点赋初值
        HT[i].c = c[i-1];
        HT[i].weight = w[i-1];
        HT[i].parent = 0; // i结点为根结点
        HT[i].lchild = HT[i].rchild = 0;
    }
    for (i=n+1; i<=m; ++i) { // 为Huffman树的内部结点赋初值
        HT[i].weight = 0;
        HT[i].parent = HT[i].lchild = HT[i].rchild = 0;
    }
    for (i=n+1; i<=m; ++i) {
        Select(i-1, &s1, &s2); //从树集合中, 找出根最小和次小的两棵树
        HT[s1].parent = i; HT[s2].parent = i; // 将最小的两棵树合并成一棵树
        HT[i].lchild = s1; HT[i].rchild = s2;
        HT[i].weight = HT[s1].weight + HT[s2].weight;
    }
}
```

第六节 赫夫曼树及其应用

四、Huffman编码实现

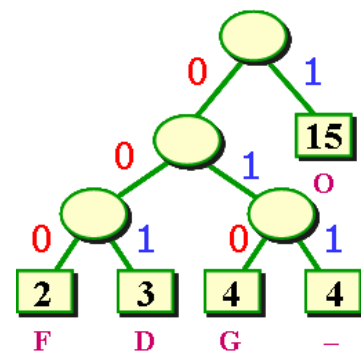
编码实现。

| | | | | |
|-------|--------|--------|-----|----|
| start | Bit[0] | Bit[1] | ... | \0 |
|-------|--------|--------|-----|----|

用字符数组 `char code[n]` 存放单结点编码。

第六节 赫夫曼树及其应用

四、Huffman编码实现



编码从叶子结点开始。对每个叶子结点k,

- ① 初始令 $start = n-2$ //n-1为' /0'
- ② $parent = HT[k].parent$;
若 $HT[parent].lchild = k$; 则 $code[start] = '0'$;
若 $HT[parent].rchild = k$; 则 $code[start] = '1'$;
 $k = parent$; $start--$;
- ③ 重复②直至 $HT[k].parent = 0$;
- ④ 每个字符的编码值为 $code[start+1 \cdots n-2]$;

以电文 ‘ABACCD A’ 的A、B字符编码为例说明编码过程。

| | | weight | Parent | Left Child | Right Child |
|---|---|--------|--------|------------|-------------|
| 0 | A | 3 | 6 | 0 | 0 |
| 1 | B | 1 | 4 | 0 | 0 |
| 2 | C | 2 | 5 | 0 | 0 |
| 3 | D | 1 | 4 | 0 | 0 |
| 4 | | 2 | 5 | 1 | 3 |
| 5 | | 4 | 6 | 2 | 4 |
| 6 | | 7 | 0 | 0 | 5 |

A字符的code变化:

start: 2

| | | | |
|--|--|---|----|
| | | 0 | \0 |
|--|--|---|----|

A: 0

B字符的code变化:

start:2

| | | | |
|--|--|---|----|
| | | 0 | \0 |
|--|--|---|----|

start:1

| | | | |
|--|---|---|----|
| | 1 | 0 | \0 |
|--|---|---|----|

start:0

| | | | |
|---|---|---|----|
| 1 | 1 | 0 | \0 |
|---|---|---|----|

B: 110

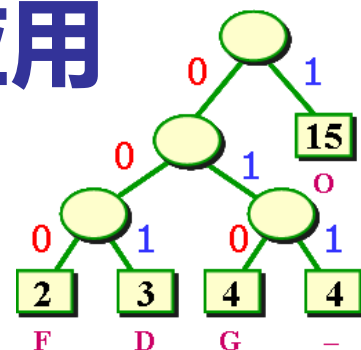
第六节 赫夫曼树及其应用

作业：对电文 ‘abbaccdeeccdabcc ‘进行 Huffman 编码。

画出生成的静态链表——Huffman 树和各字符编码, 最后形成的 code 数组。

第六节 赫夫曼树及其应用

四、Huffman译码 (算法)

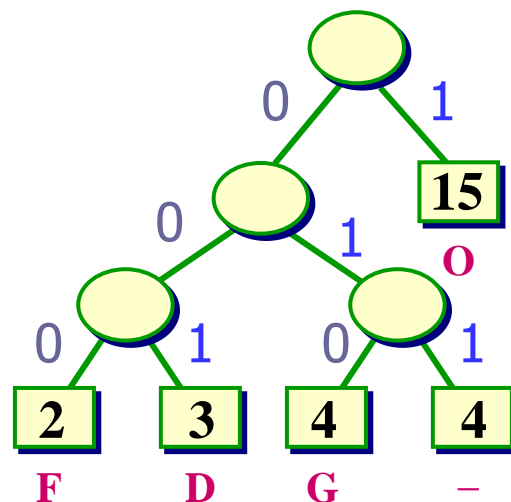


- 1、指针指向Huffman树的根结点，取第一个Huffman码
- 2、如果Huffman码为 ‘0’ ，将指针指向当前结点的左子树的根结点；如果Huffman码为 ‘1’ ，将指针指向当前结点的右子树的根结点
- 3、如果指针指向的当前结点为叶子结点，则输出叶子结点对应的字符；否则，取下一个Huffman码，并返回2
- 4、如果Huffman码序列未结束，则返回1继续译码

第六节 赫夫曼树及其应用

四、Huffman译码

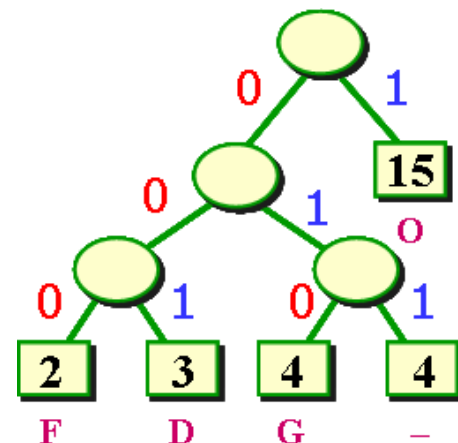
- 如Huffman编码序列01011001，译码后的字符串为GOOD



第六节 赫夫曼树及其应用

四、Huffman译码(程序)

```
int ShowHuffmanDecode(int n, char *dString)
{
    int i, c, Root;
    Root = 2*n - 1;          // Huffman树的根
    c = Root;                // 指针指向Huffman树的根
    for (i=0; i<strlen(dString); i++) {
        if (dString[i] == '0')
            c = HT[c].lchild; // Huffman码为0, 指针指向左子树的根结点
        else
            c = HT[c].rchild;
        if ((HT[c].lchild == 0) && (HT[c].rchild == 0)) {
            cout << HT[c].c;    // 结点为叶子, 输出字符
            c = Root;
        }
        // Huffman码为1, 指针指向右子树的根
    }
    if (HT[c].parent!=0 ) return (ERROR);
    //如果循环结束时, c不是根结点, 则表明编码不完整。
    else
        return(CORRECT);
}
```



第六节 赫夫曼树及其应用

作业：假设某班成绩分布如下表：

| | | | | | |
|-----|------|-------|-------|-------|------|
| 分数 | <60 | 60~69 | 70~79 | 80~89 | >90 |
| 比例数 | 0.03 | 0.04 | 0.5 | 0.3 | 0.13 |

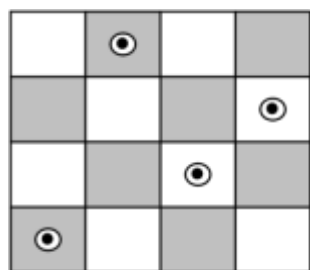
为查找某一分数段同学成绩，需进行比较。试构造二叉树，使得总比较次数最少。

N皇后问题

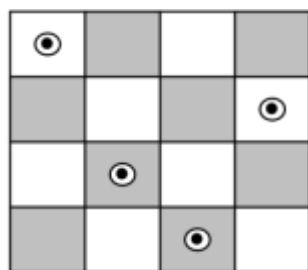
在 $n \times n$ 格的棋盘上放置彼此不受攻击的 n 个皇后。按照国际象棋的规则，皇后可以攻击与之处在同一行或同一列或同一斜线上的棋子。

n 后问题等价于在 $n \times n$ 格的棋盘上放置 n 个皇后，任何2个皇后不放在同一行或同一列或同一斜线上。

$n=1$ 显而易见。 $n=2、3$ ，问题无解。 $n \geq 4$ 时，以4后为例

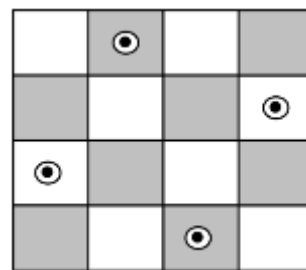


(a)



(b)

4后问题的两种无效布局



4后问题的一个有效布局

求解过程图示

继续使用回溯法查找，
就可以得到问题的其它
解。

问题：

- 1、解怎么表示？
- 2、解如何组织？
- 3、怎么找到最优解？

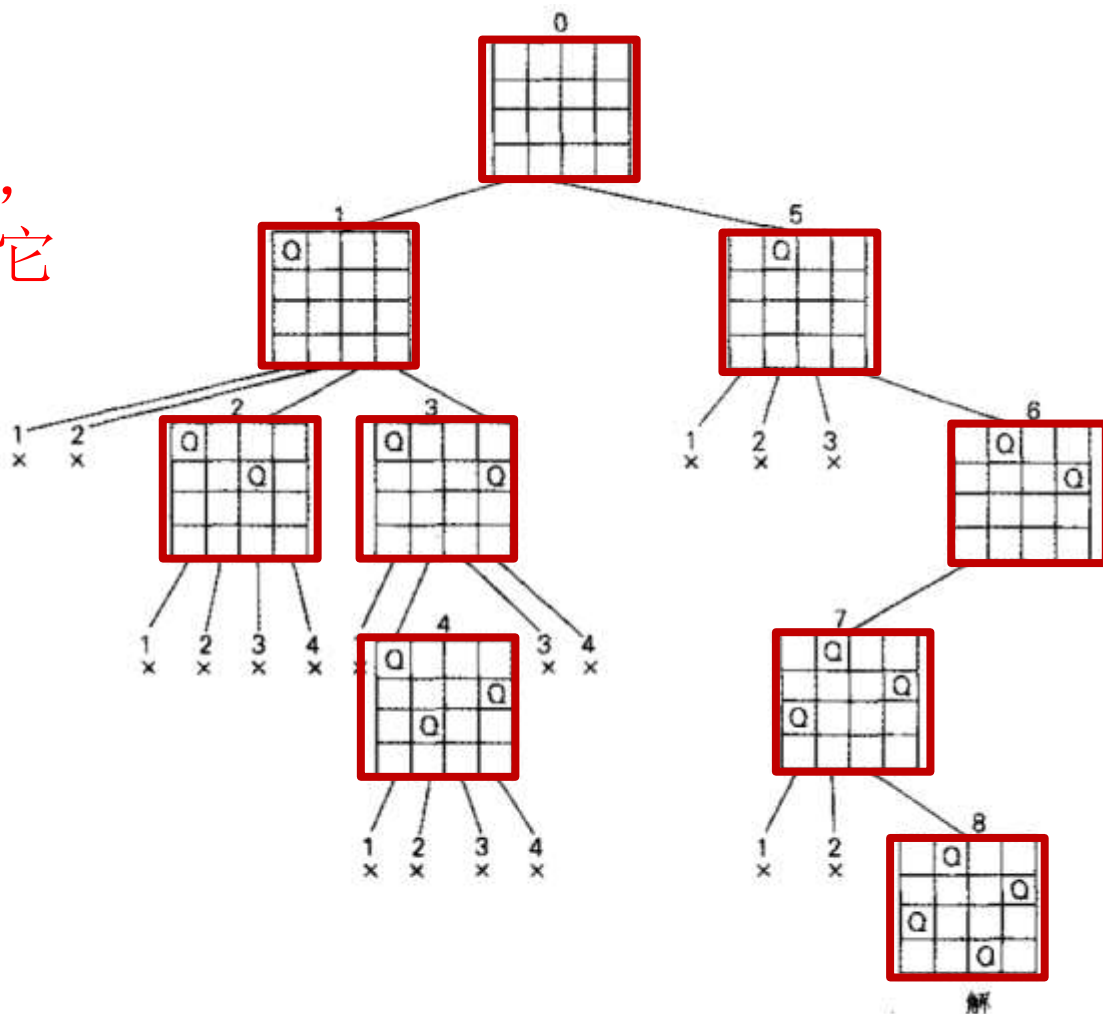


图 11.2 用回溯法解 4 皇后问题的状态空间树。x 表示一个试图把皇后放在指定列的不成功的尝试。节点上方的数字指出了节点被生成的次序

回溯法

- 回溯法在问题的解空间树中，按深度优先策略，从根结点出发搜索解空间树。
- 算法搜索至解空间树的任意一点时，先判断该结点是否包含问题的解。
 - 如果肯定不包含，则跳过对该结点为根的子树的搜索，逐层向其祖先结点回溯；
 - 否则，进入该子树，继续按深度优先策略搜索。

回溯法


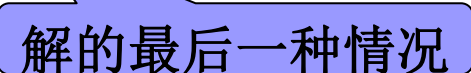
- 给定问题有一个约束集合以及目标函数.
- 可以用状态空间树 *state space tree* 来表示解空间
 - 状态空间树的根代表了0个选择的状态
 - 深度为1的节点代表第1次选择后的状态
 - 深度为2的节点代表第2次选择后的状态
 - ...
 - 状态空间树上一条从根到叶子的路径代表了备选解
- *可行问题Feasibility problem*: 一些选择可以到达可行的解, 一些选择不能达到可行解

回溯法的关键问题

- 节点的含义是什么？（根据问题确定）
- 节点在树中的关系是什么？（根据问题确定）
- 如何产生新的节点？（树的遍历算法）
- 如何判断节点是否是所求解？（easy！）

递归回溯

回溯法对解空间作深度优先搜索，因此，在一般情况下用递归方法实现回溯法。

```
void backtrack (int t)
{
    if (t>n) output(x), 
    else
        for (int i=f(n,t);i<=g(n,t);i++) {
            x[t]=h(i);//当前解 
            if (constraint(t)&&bound(t)) backtrack(t+1);
        }
}
```

学习总结

- 树的定义和基本术语
- 二叉树
 - 二叉树的定义、性质、存储结构
- 遍历二叉树和线索二叉树
- 树和森林
 - 树的存储结构，森林与二叉树的转换
 - 树和森林的遍历
- 赫夫曼树及其应用
- 回溯法与树的遍历