第六章 树和二叉树

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

本章节目录

6.1 树

6.2 二叉树

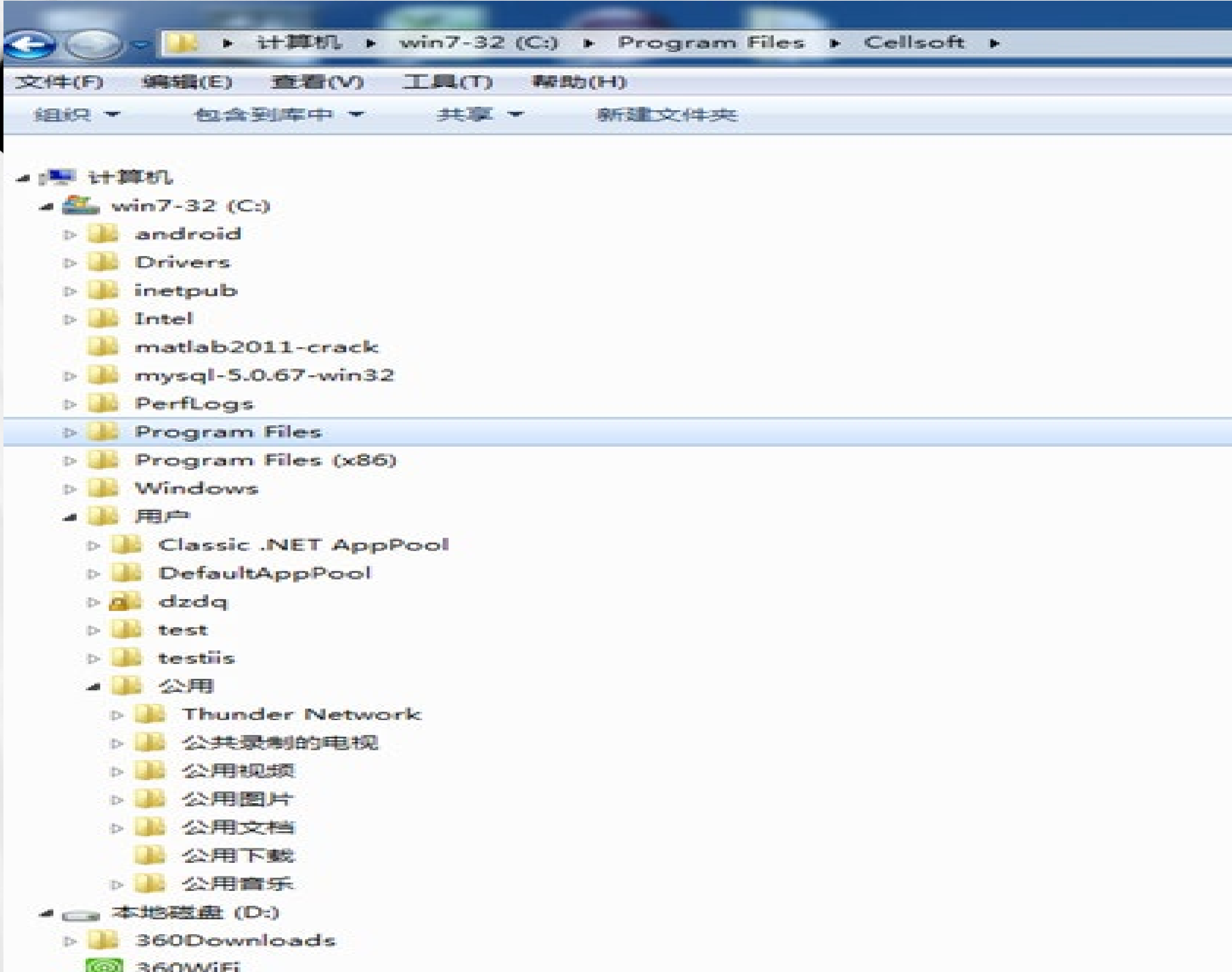
6.3 二叉树的遍历

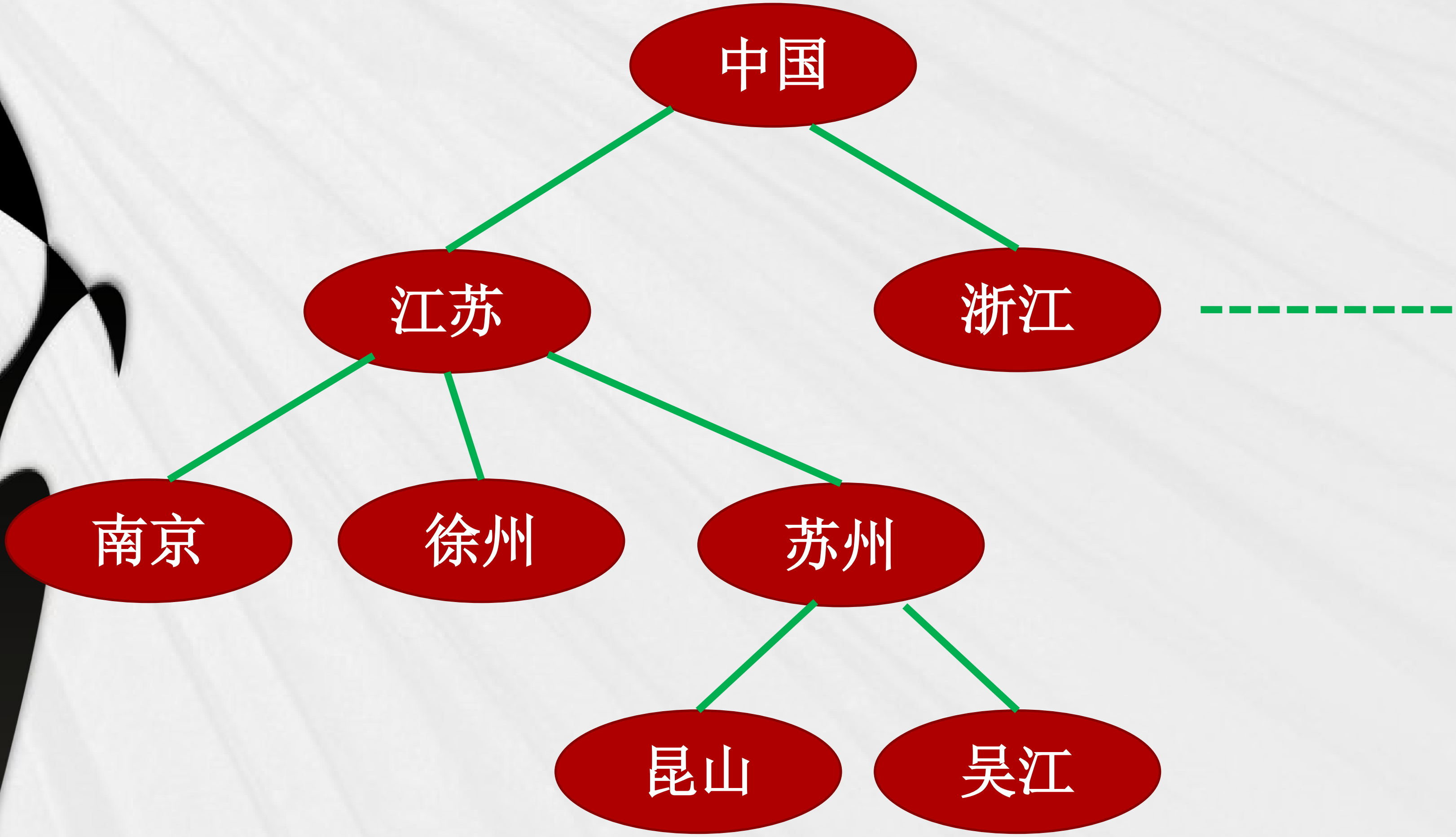
6.4 森林与二叉树的转换

6.5 哈夫曼树及其应用

6.1 树

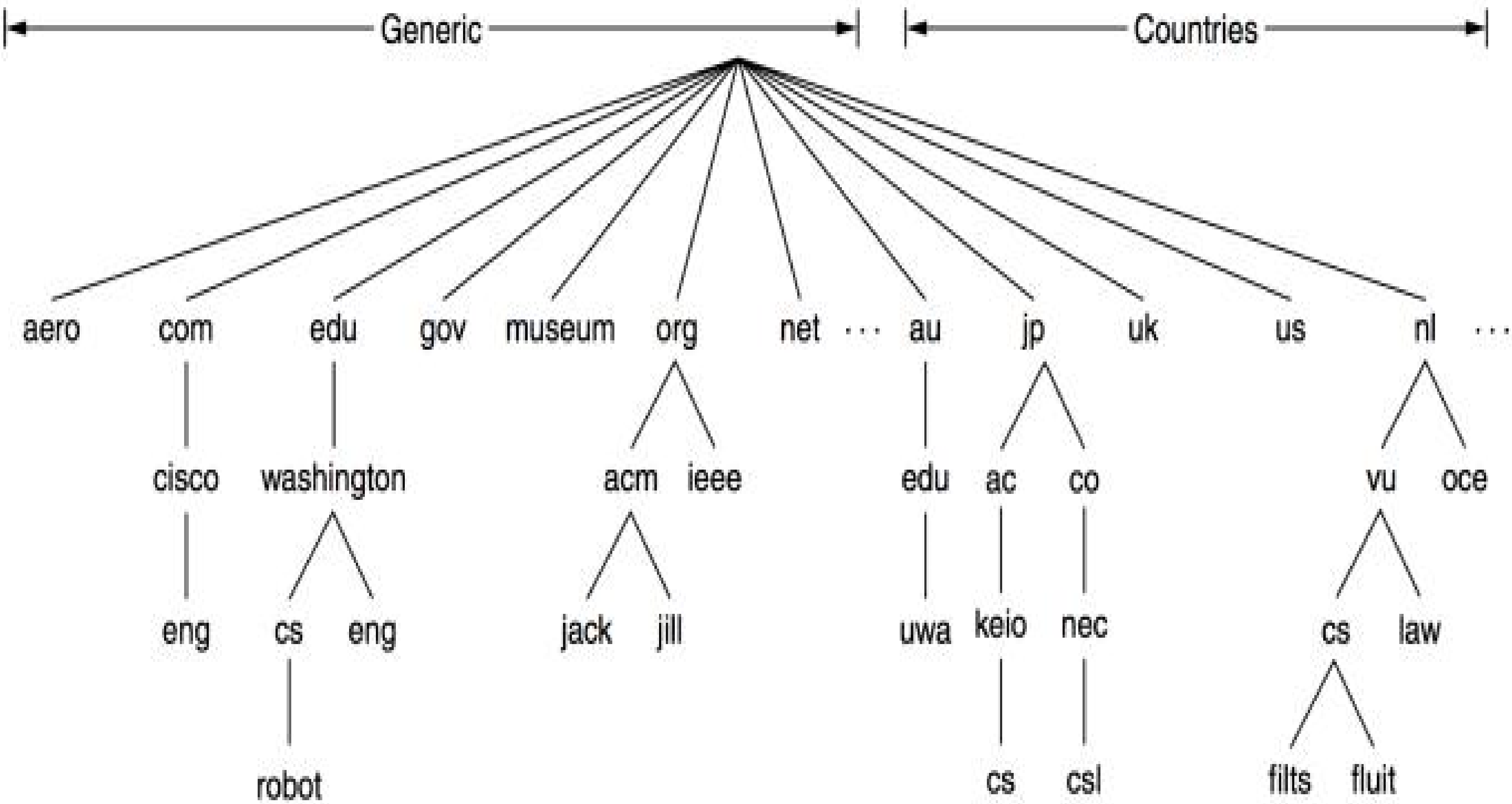
- 树是一种非线性的层次结构。
- 客观世界中的层次结构：人类家族谱系和各种社会管理机构的组织架构。
- 在计算机科学中，树为具有层次关系或分支关系的数据提供了一种自然的表示，可以用来描述操作系统中文件系统的结构，也可以用来组织数据库系统的信息，还可以在编译过程中表示源程序的语法结构（运算符：单目、双目（+，—，*，%）、多目）。
- 图6-1 (下页）

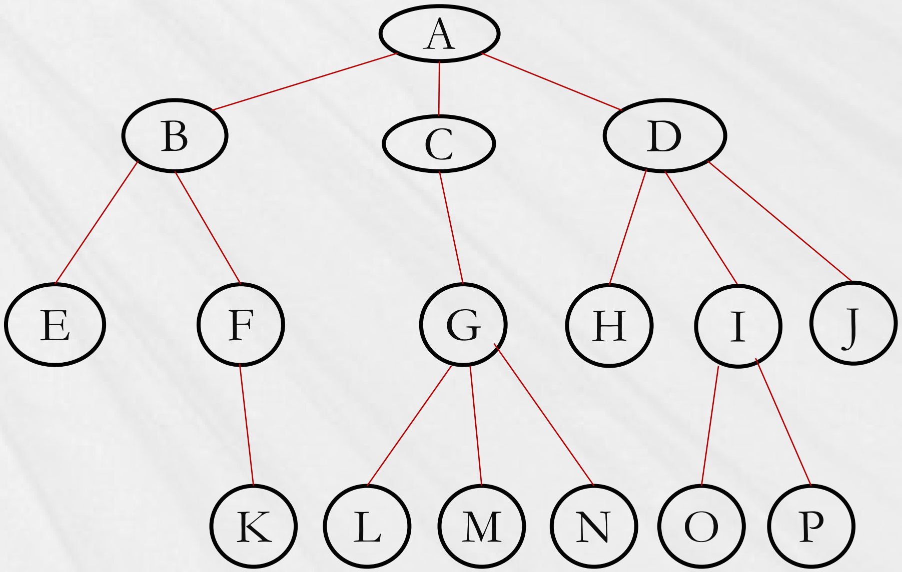

计算机 ▸ win7-32 (C:) ▸ Program Files ▸ Cellsoft ▸
文件(F) 编辑(E) 查看(V) 工具(T) 帮助(H)
组织 包含到库中 共享 新建文件夹
计算机
win7-32 (C:)
android
Drivers
inetpub
Intel
matlab2011-crack
mysql-5.0.67-win32
PerfLogs
Program Files
Program Files (x86)
Windows
用户
Classic .NET AppPool
DefaultAppPool
dzdq
test
testiis
公用
Thunder Network
公共录制的电视
公用视频
公用图片
公用文档
公用下载
公用音乐
本地磁盘 (D:)
360Downloads
360WiFi



人口或者GDP分布

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE


Generic
Countries
aero
com
edu
gov
museum
org
net
...
au
jp
uk
us
nl
...
cisco
washington
acm
ieee
edu
ac
co
vu
oce
eng
cs
eng
jack
jill
uwa
keio
nec
cs
law
robot
cs
csl
filts
fluit


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P

6.1.1 树的定义和表示

树是由n(n≥0)个结点构成的**有限集合**。当n=0时，称为**空树**；对n＞0的树T有：

①有一个特殊的结点称为**根**；

②当n＞1时，除根结点外其他结点被分成m(m ≥0)个互不相交的集合T1，T2，…，Tm。其中，每一个集合Ti（1≤i≤m）本身又是一棵树，称为**根**结点的**子树**。

注意：递归定义

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

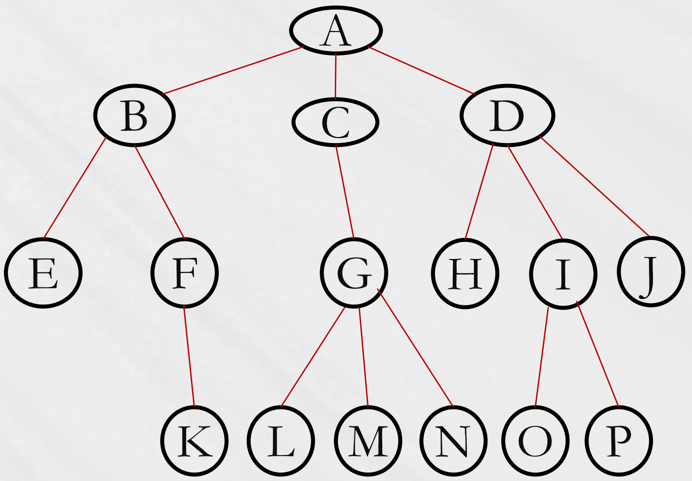
◆树的多种表示方法：

（1）直观表示法（倒立生长的树）

（2）嵌套集合表示法(p70)

（3）凹入表示法(p70)

◆ **树的特征：1对多**



中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

6.1.2 树的基本术语和操作

1.树的基本术语

孩子：一个节点的所有子树的根节点，该节点称为所有子树的根节点的双亲

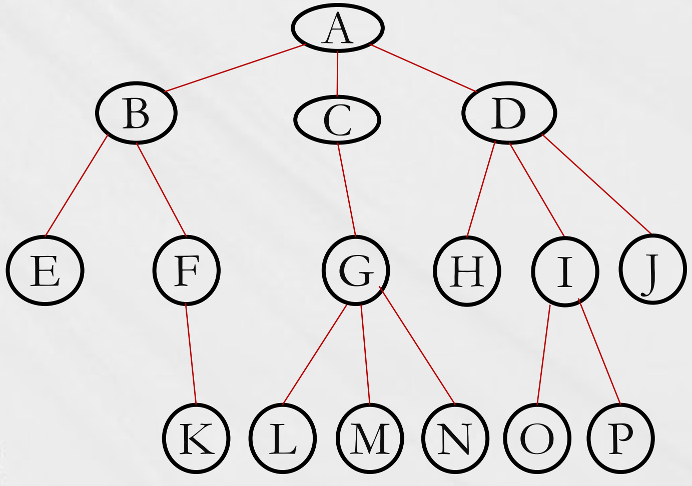
兄弟：具有相同双亲节点的节点

祖先：根节点到该节点所经历的所有节点

子孙：某个节点为根节点的子树中的所有节点

结点的度：子树的个数

树的度： 树中结点度的最大值



1.树的基本术语

叶子结点：度为0的节点

分支结点：度不为0

内部结点：根节点以外的其他分支节点

结点的层数：根为1，其孩子为2，……..

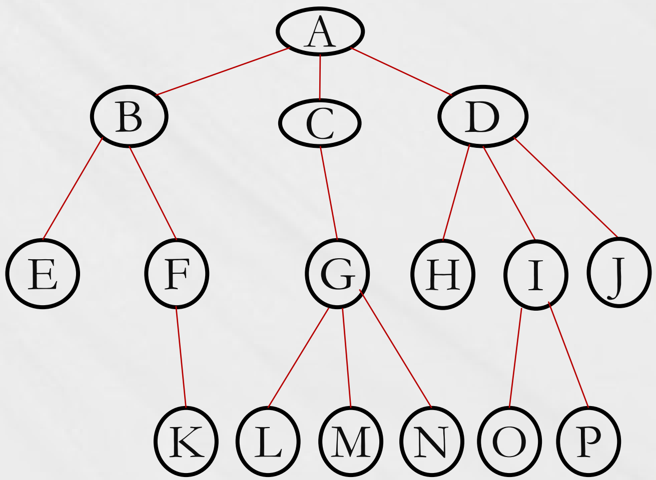
树的高度(深度）：所有节点层次的最大值

堂兄弟：双亲在同一层次上的节点

森林：

有序树：

无序树



中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

2.树的基本操作

◆ 树的基本操作：

创建、销毁、查询、插入、删除、遍历等。

◆树的抽象数据类型

ADT Tree {

数据对象：D 是具有相同特性的数据元素的集合，称为结点集

数据关系：R={H}

若D为空集或者仅具有一个数据元素，则R为空集；否则H如下：

（1）在D中存在唯一的称谓树根的数据元素root，它在关系H下没有直接前驱。

（2）除root以外，每个结点在H下有且仅有一个直接前驱。

（3）除root以外，D中结点可以划分为m个互不相交的子集，每个子集及H又构成了符合本定义的树，分别称为root的子树。

基本操作：初始化，销毁，创建，清空，求树高度，查找某个节点的值，插入子树，删除子树，遍历树

}

6.1.3 树的存储结构

- 双亲表示法
- 孩子链表表示法
- 孩子兄弟表示法

1.双亲表示法---仅用顺序存储结构

❖ 将树结点顺序存储在一个**数组**中。每个数组元素中除存放结点信息外，还需要存放该结点**双亲的下标值**。

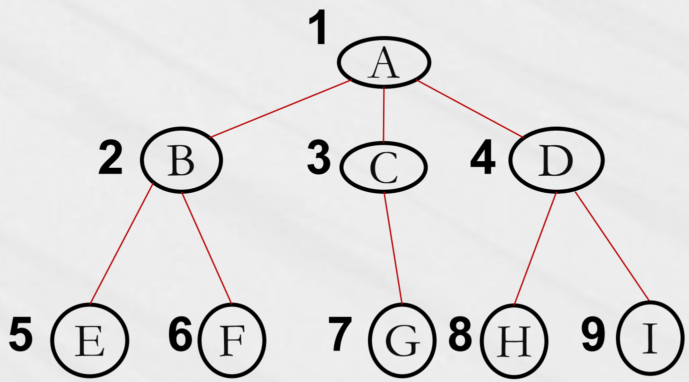
❖ 树的双亲表示法存储结构描述如下：

```
# define MAXSIZE 1000   // 结点个数的最大值
typedef    struct
{
    ElemType  data;          // 结点信息
    int    parent;           // 双亲结点下标
  }        TNode;

TNode    tree[MAXSIZE+1];         // tree［0］未用
```

能存下双亲节点吗？

1.双亲表示法



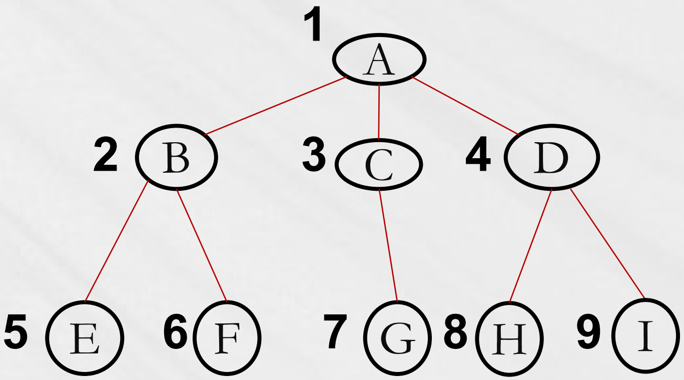
下标:	1	2	3	4	5	6	7	8	9
Data:	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
Parent:	**0**	**1**	**1**	**1**	**2**	**2**	**3**	**4**	**4**

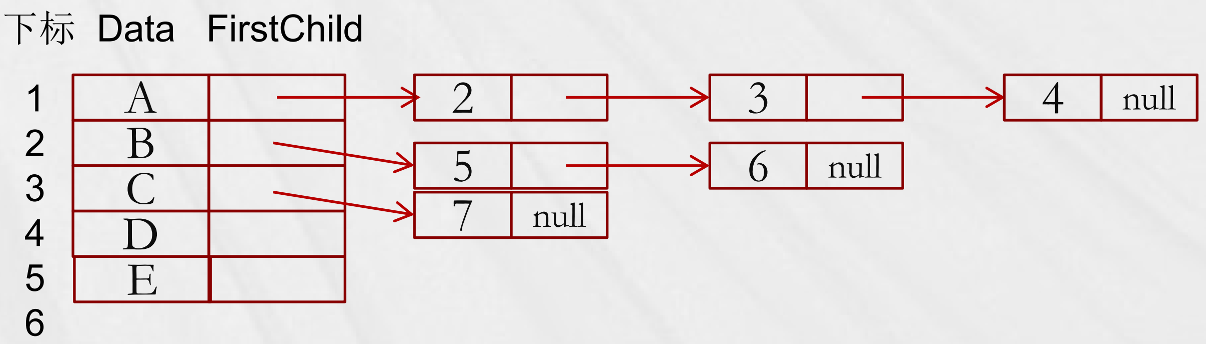
特点： 找双亲结点容易

找某个结点的子孙节点难，可能要遍历整个树

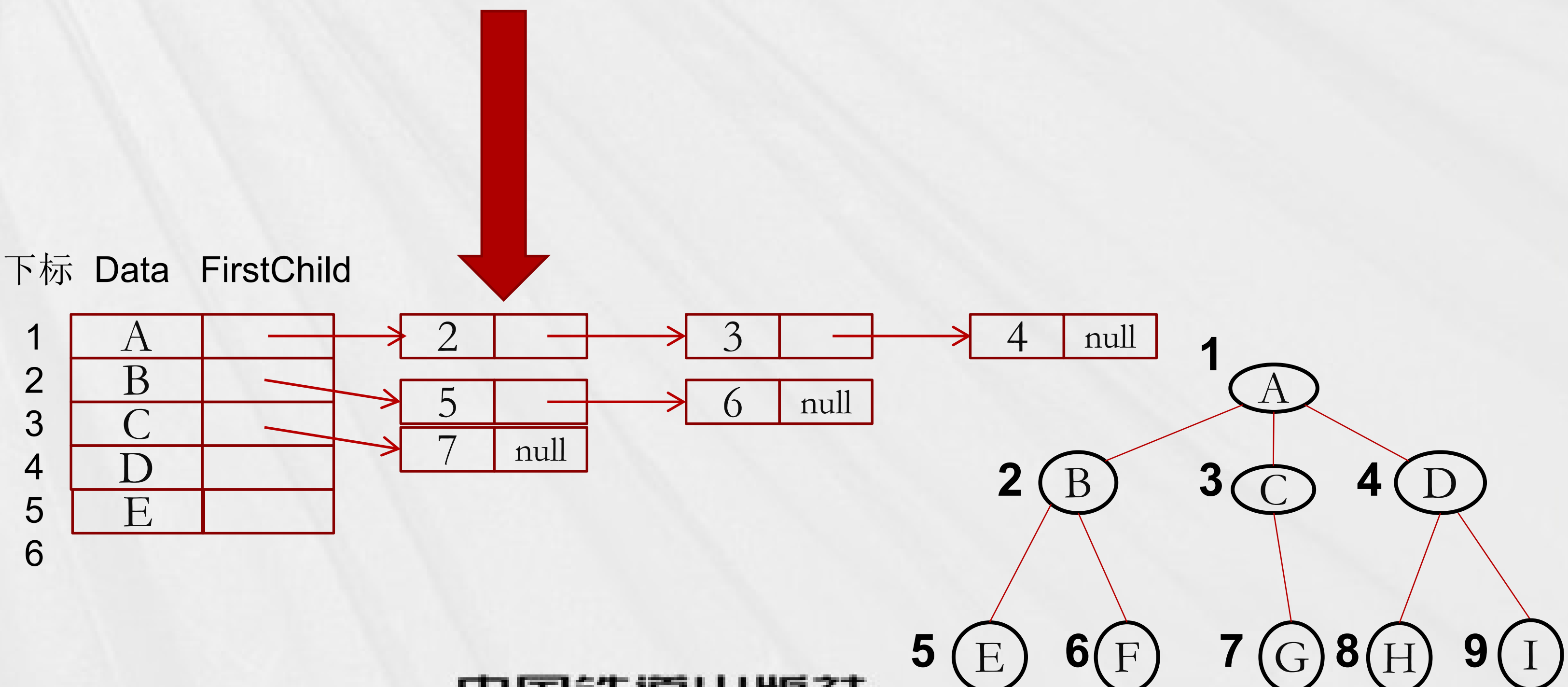
2.孩子链表表示法----顺序存储+链式存储

- 树中所有结点组成一个顺序表-----数组。
- 每个数组元素为一个结点，每个结点设置一个指针域，该指针指向该结点所有孩子结点构成的链表。



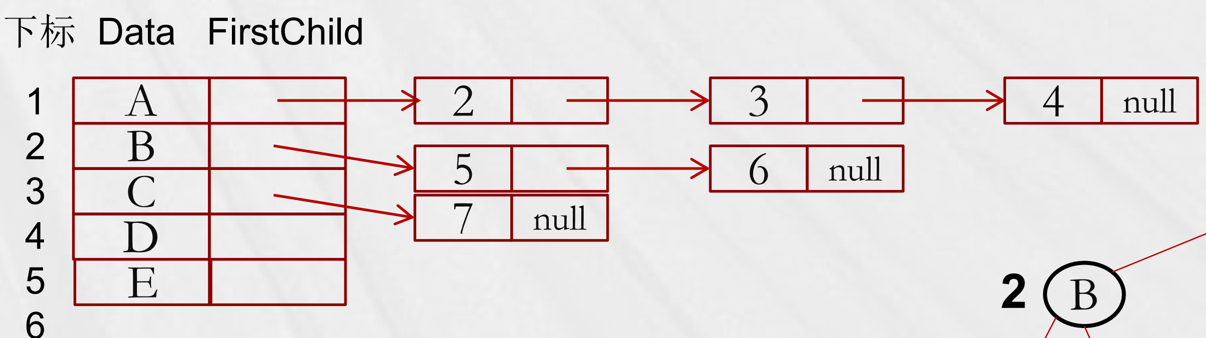


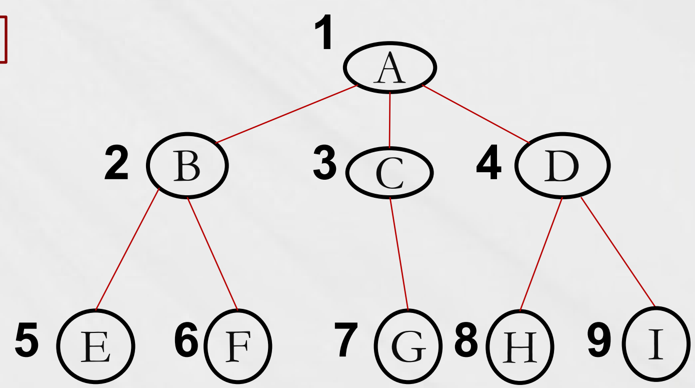
```
#define MAXSIZE   1000        // 结点的个数最多为1000
typedef  struct childnode   // 孩子链表表示法结点的定义
{   int    child;                // 该孩子结点在顺序表中的位置
    struct childnode   *next;   //指向下一个孩子结点的指针
}    ChildNode;
```



孩子链表表示法

```
typedef  struct                    // 顺序表结点的结构定义
{   ElemType    data;                     // 结点信息
    ChildNode   *FirstChild;              // 指向孩子链表的头指针
  }    TNode;
TNode   tree［MAXSIZE+1］;        // 树的孩子链表表示
```





孩子链表表示法

◆孩子链表表示法查找孩子及其子孙容易，但查找双亲难。

◆可以将双亲表示法和孩子链表表示法相结合，在孩子链表表示的顺序表中增加一个parent域，这就是树的**孩子双亲表示法**。

```
typedef   struct                // 顺序表结点的结构定义
{   ElemType  data;             // 结点信息
    int   parent;               //双亲节点
    ChildNode *FirstChild;          // 指向孩子链表的头指针
  }     TNode;

TNode    tree［MAXSIZE+1］;         // 树的孩子链表表示
```

3.孩子兄弟表示法--------链表

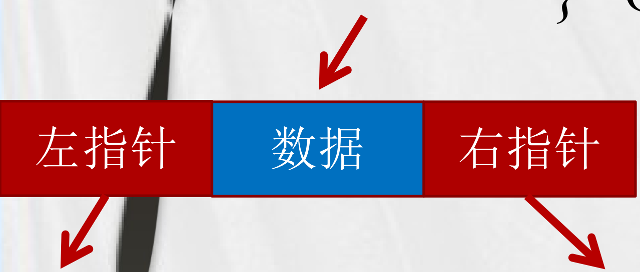
❖ 孩子兄弟**链表的结点结构**是：

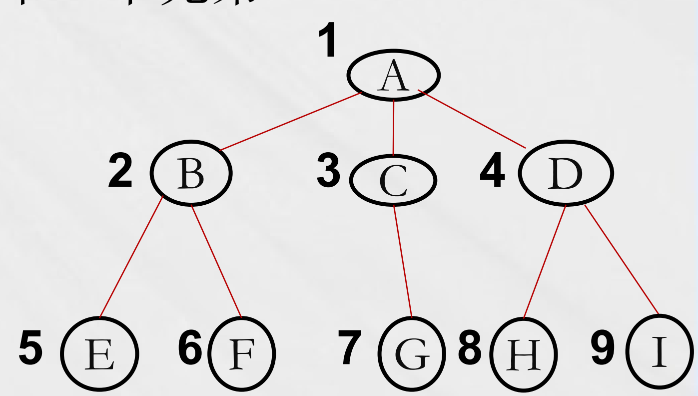
◆ **结点信息域**

◆ **两个指针域**（孩子指针：指向它的第一个孩子结点，兄弟指针：指向它的下一个兄弟结点）。

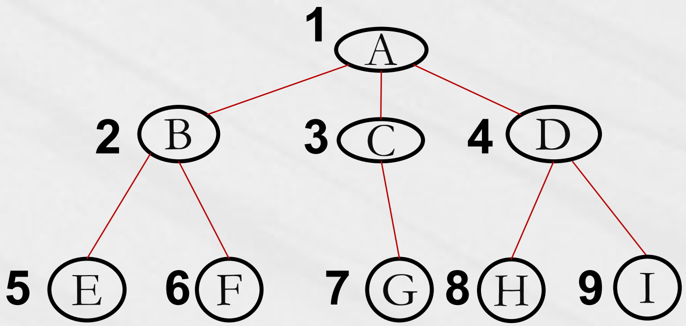
❖ 这种存储结构的描述如下：

```
typedef  struct csnode      {
  ElemType   data;        // 结点信息
  struct  csnode   *FirstChild，  *NextSibling;
                    // 第一个孩子，下一个兄弟
      }   CSNode，*CSTree;
```

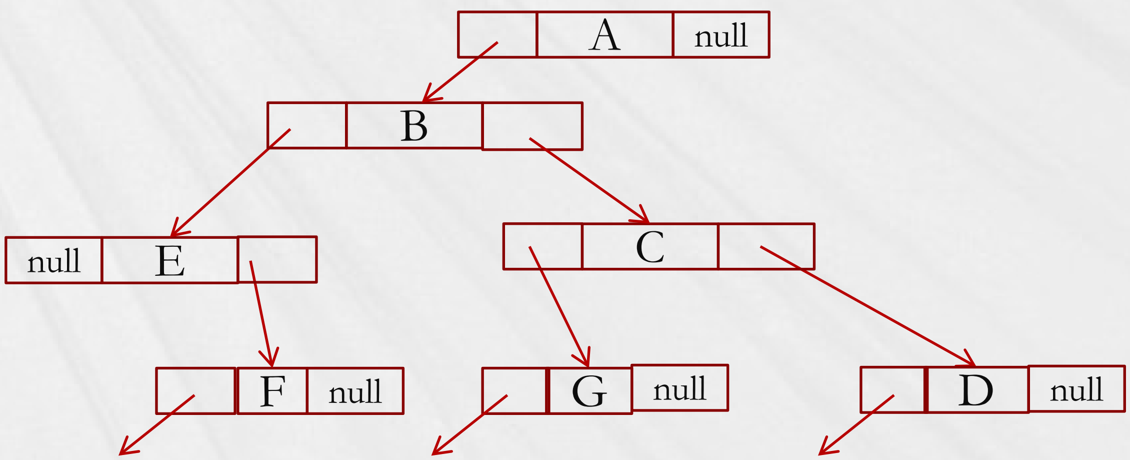




3.孩子兄弟表示法



FirstChild　Data　nextsibling



特点：

便于实现树的 各种操作，如 查找一个结点的第i个孩子。

采用最多的表示方式。

6.2 二叉树

6.2.1 二叉树的定义

二叉树（Binary Tree）是n(n≥0)个结点构成的**有序树**。当n=0时，称为空二叉树；对n＞0的二叉树由一个根结点和两个互不相交的、分别称作左子树和右子树的子二叉树构成。

- 二叉树每个结点的度**不大于2**。
- 二叉树共有五种基本形态 (空二叉树、只有根节点、右子树为空、左子树为空、左右子树均非空）
- 二叉树的抽象数据类型

二叉树的抽象数据类型

ADT BinaryTree {

数据对象：D 是具有相同特性的数据元素的集合，称为结点集

数据关系：R={H}

若D为空集或者仅具有一个数据元素，则R为空集；否则H如下：

（1）在D中存在唯一的称谓树根的数据元素root，它在关系H下没有直接前驱。

（2）除root以外，每个结点在H下有且仅有一个直接前驱。

（3）除root以外，D中结点可以划分为2个互不相交的子集，每个子集及H又构成了符合本定义的二叉树，分别称为root的左子树、右子树。

基本操作：初始化，销毁，创建，清除，求树高度，查找某个节点的值，插入左右子树，删除左右子树，先根、中根、后根遍历树

}

6.2.2 二叉树的性质

性质1 二叉树第i(i≥1)层上至多有2^{i-1}个结点。

性质2 高度为k(k≥1)的二叉树至多有2^k-1个结点。

性质3 在任意二叉树中，$n_0=n_2+1$（n0:度为0的结点数，n2:度为2的节点数）

性质4 具有n个结点的完全二叉树高度为 $\log_2 n+1$。

- **满二叉树**：高度为k含有2^k-1个结点的二叉树。
- **完全二叉树**：对于高度为k，含有n个结点的二叉树，从根结点开始自上向下，自左至右顺序编号，它的每个结点的编号都与相应满二叉树结点顺序编号从1到n相对应。

结点的序号

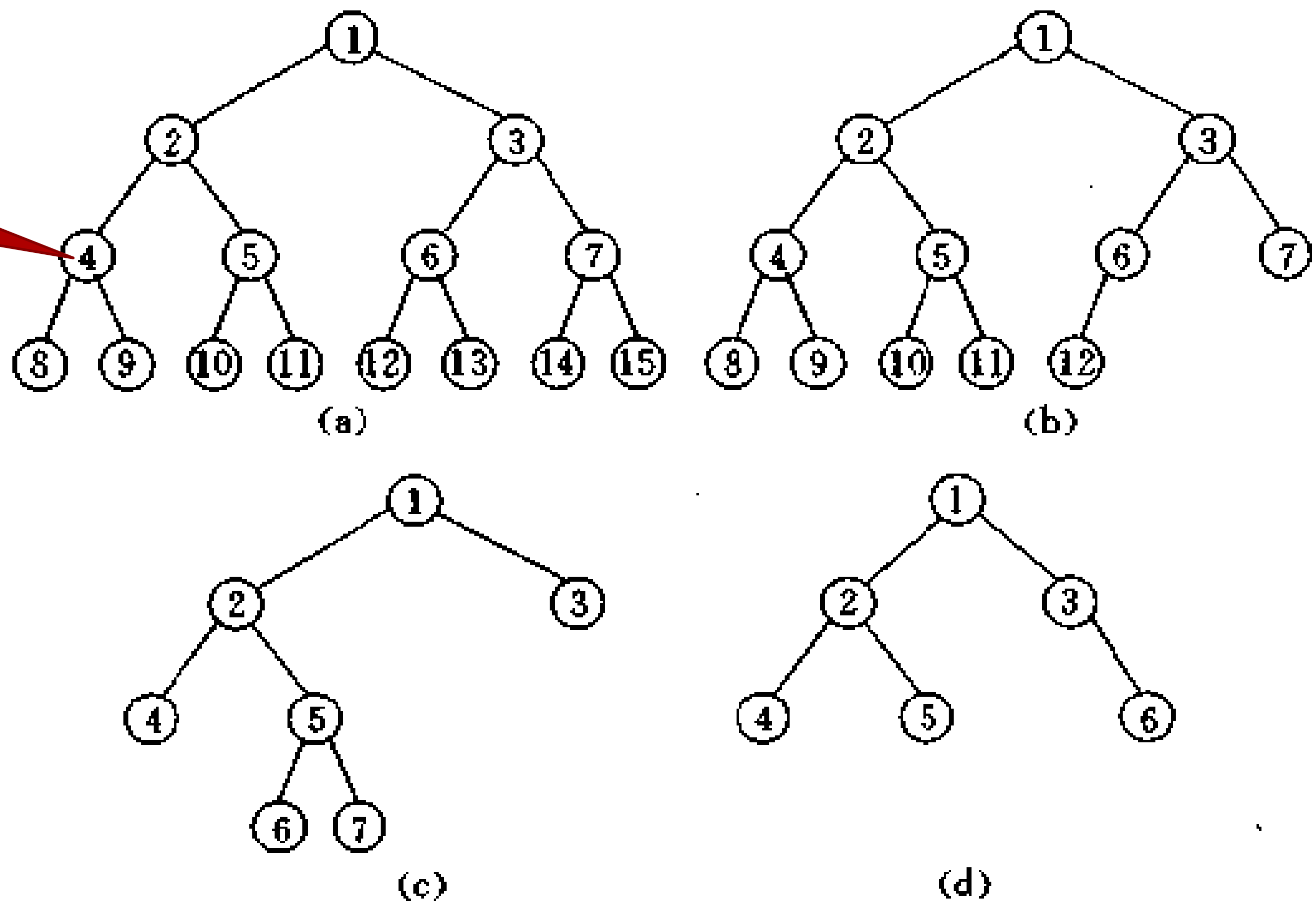
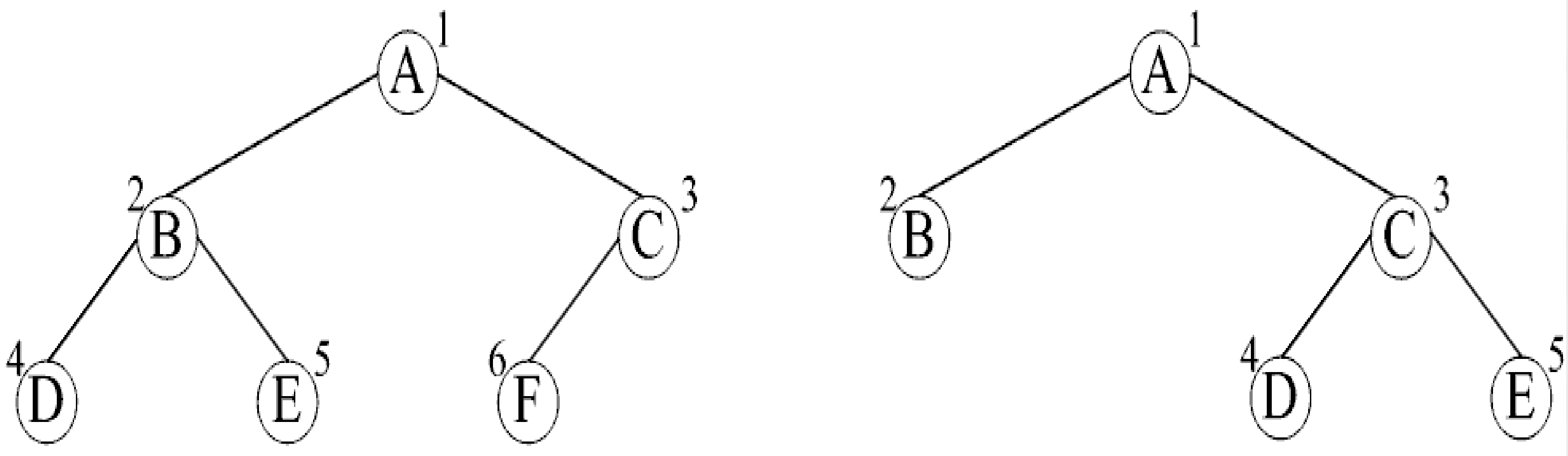


图 6.4 特殊形态的二叉树

(a) 满二叉树；(b) 完全二叉树；(c)和(d)非完全二叉树。

❖ **满二叉树:**高度为k含有2^k-1个结点的二叉树。

❖ **完全二叉树**：对于高度为k，含有n个结点的二叉树，从根结点开始自上向下，自左至右顺序编号，它的每个结点的编号都与相应**满二叉树**结点顺序编号从1到n相对应。



(a) (b)

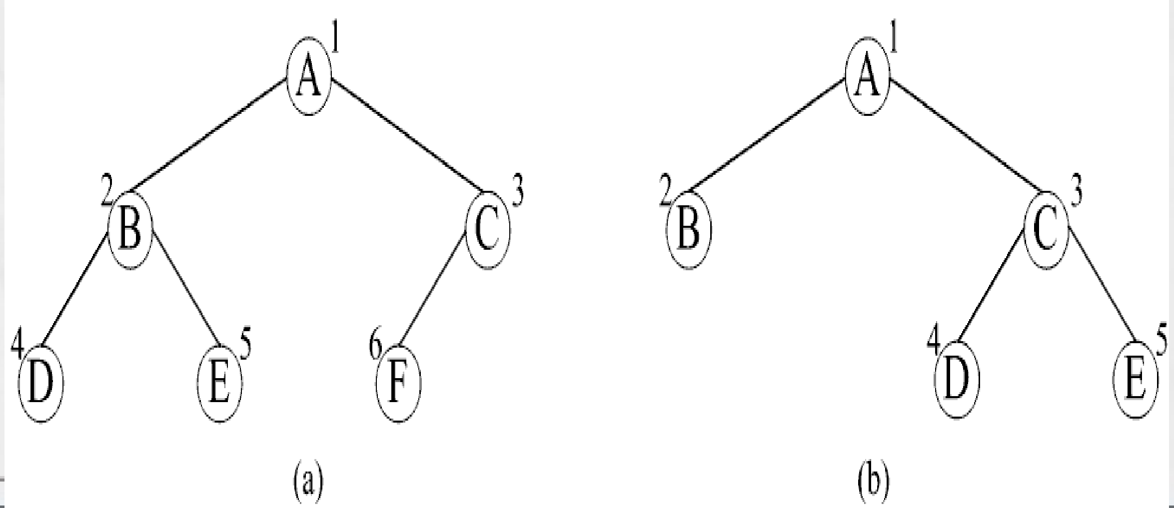
性质5 若对有n个结点的**完全二叉树**按照从上至下、从左至右进行顺序编号，那么，对于编号为i(1≤i≤n)的结点：

1）当i=1时，该结点为根，它无双亲结点；当i＞1时，编号i结点的孩子结点编号为：

2）若**2i≤n**，它有编号为**2i**的左孩子，否则没有左孩子；

3）若**2i+1≤n**，则它有编号为**2i+1**的右孩子，否则没有右孩子。

注意：节点编号间的这种关系用在二叉树的顺序存储结构中



6.2.3 二叉树的存储结构

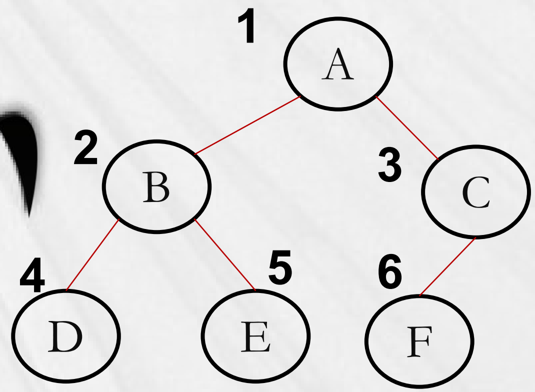
1.顺序存储结构------数组

- 用一组**连续的存储单元**存储二叉树的数据元素，每个数组元素存储树的一个结点的数据信息。
- 必须反映出结点之间的逻辑关系。左孩子、右孩子
- 按照**完全二叉树**进行排列，不存在的结点值为0
- 二叉树的顺序存储结构描述如下：

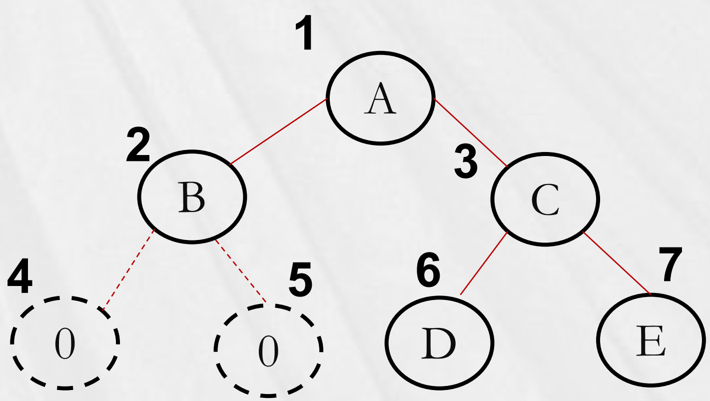
```
# define   MAXSIZE   1000
ElemType    BT［MAXSIZE+1］；  //0元素不用
```

6.2.3 二叉树的存储结构

1.顺序存储结构：第i个结点：父节点⌊ i/2⌋，左右子结点2i和2i+1



下标	1	2	3	4	5	6
值	A	B	C	D	E	F



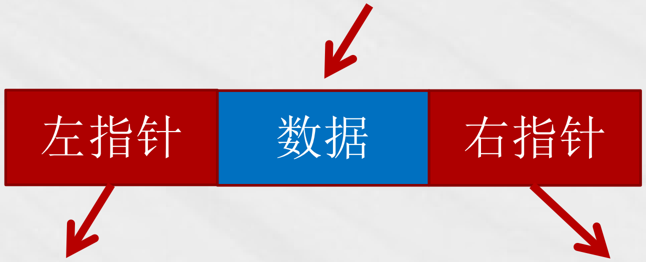
下标	1	2	3	4	5	6	7
值	A	B	C	0	0	D	E

顺序存储结构的特点：对于非完全二叉树，存储空间浪费严重。

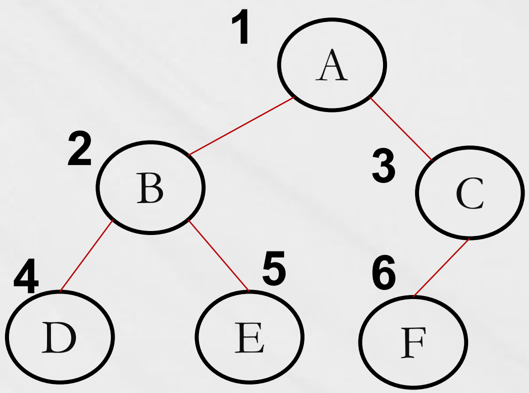
如：仅具有右子树的k个结点的二叉树，需要2^k个存储空间

2.链式存储结构

- 二叉树的结点：**信息域、两个指针域**（分别指向其左、右子树根结点）
- 二叉链表，具体描述如下：




```
typedef   struct  node
{
  ElemType     data;       // 结点信息
  struct node   *lchild，*rchild;   // 左、右指针域
    }   Bnode,    *BTree;
```



- 为了方便实现寻找双亲结点，还可以再增设一个指向其双亲结点的指针域，这样得到的结点结构称为**三叉链表。**

- 每个结点具体描述如下：

```
typedef struct    node3
{
  ElemType        data;
  struct node3    *lchild，  *parent，  *rchild;
   }   Bnode3,  *BTree3;
```

❖ 创建二叉链表：已知左、右子树

```
int CreateBTree(BTree BT, BTree BTl, BTree BTr)
{
    if ((BT=(BTree)malloc(sizeof(Bnode))) !=NULL)
    {
        BT->lchild = BTl;
        BT->rchild = BTr;
        return TRUE;                    }
    return FALSE;
}// CreateBTree
```

6.3 二叉树的遍历

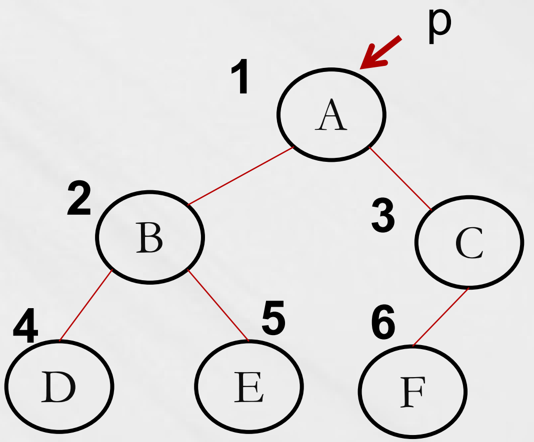
❖ 遍历二叉树

依次访问树中的每一个结点，使得每一个结点均被访问一次，而且仅被访问一次。

❖ 遍历二叉树**实质**是把二叉树的结点进行**线性排列**的过程，从而可以按这种线性排列访问树中的每一个结点，使得每一个结点均被访问一次，而且仅被访问一次（处理数据）。

❖ 三种遍历方法

❖ 给出根节点的地址：指针



6.3 二叉树的遍历

6.3.1 常用的二叉树遍历算法

1 先根遍历：根节点—左子树-右子树

如果根不空，则

① 访问根结点；

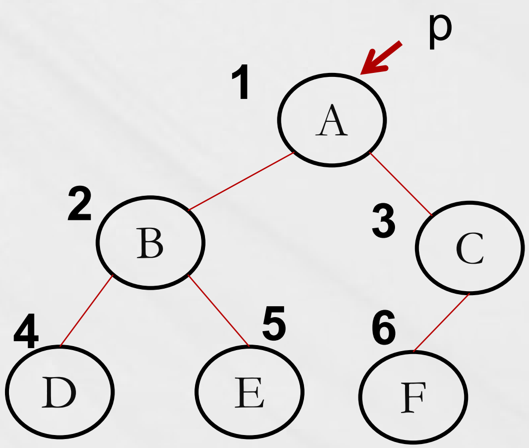
② 按先根顺序遍历左子树；

③ 按先根顺序遍历右子树；

否则返回。

遍历结果：ABDECF

```
void preorder(BTree     p)
{
if (p!＝NULL)
{
   visit(p);
   preorder(p->lchild);
   preorder(p->rchild);
            }
}// preorder
```

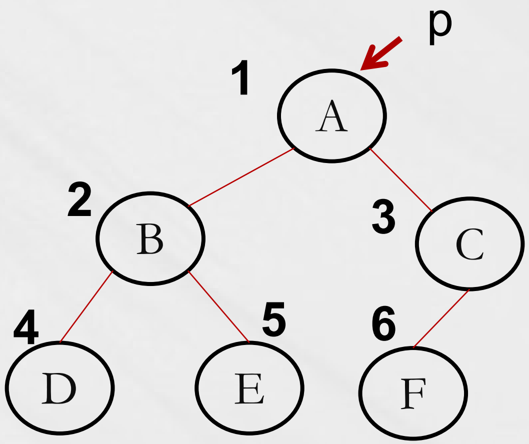


先根遍历—非递归算法-----栈

注意：左右子树的入栈顺序，出栈时再访问结点

```
void preorder(BTree   p)
{
Initstack(s);
push(s,p);
While(!stackempty(s))
{  p=pop(s);
   if (p!＝NULL)
   {
      visit(p);
       push(s,p->rchild);
       push(s,p->lchild);
     }
}
}// preorder
```

非递归算法：
结点先入栈
出栈时在访问



2.中根遍历

如果根不空，则

① 按中根次序遍历左子树；

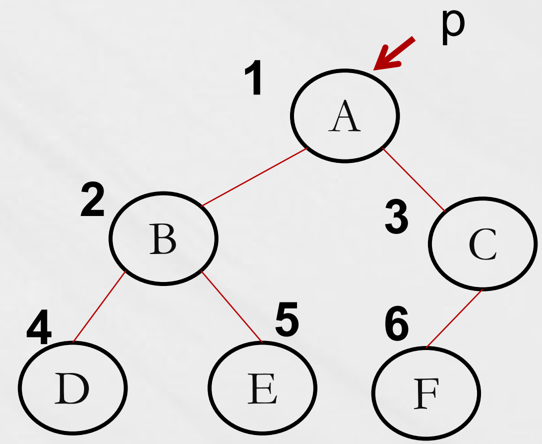
② 访问根结点；

③ 按中根次序遍历右子树；

否则返回。

```
void inorder(BTree   p)
{
if (p!＝NULL)
{
   inorder(p->lchild);
   visit(p);
   inorder(p->rchild);
             }
}// inorder
```

遍历结果：DBEAFC

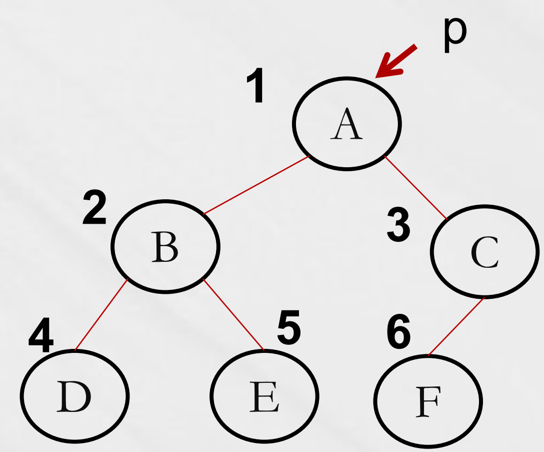


中根遍历—非递归算法

非递归算法：
结点先入栈
出栈时在访问

```
void inorder(BTree p)
{
initstack(s);
do {
    while(p!=NULL)
        {   push(s,p);   p=p->lchild;  }  //左孩子反复入栈

    if(!stackempty(s))
          {  p=pop(s);
             visit(p);
             p=p->rchild;
             }
      }  while(p || !stackempty(s));
}// inorder
```



中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

3.后根遍历

如果根不空，则

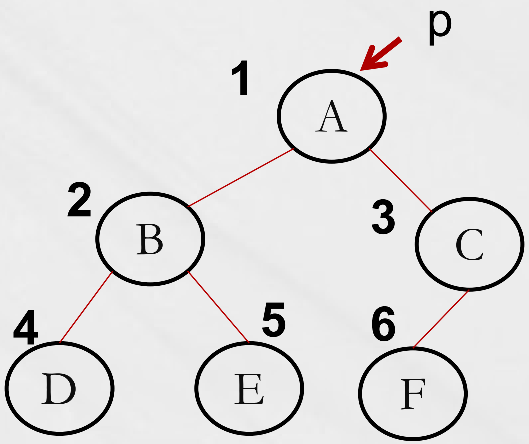
① 按后根顺序遍历左子树；

② 按后根顺序遍历右子树；

③ 访问根结点；

否则返回。

```
void postorder(BTree   p)
{
    if (p!＝NULL)
   {
      postorder(p->lchild);
      postorder(p->rchild);
      visit(p);
    }
}// postorder
```



遍历结果：DEBFCA

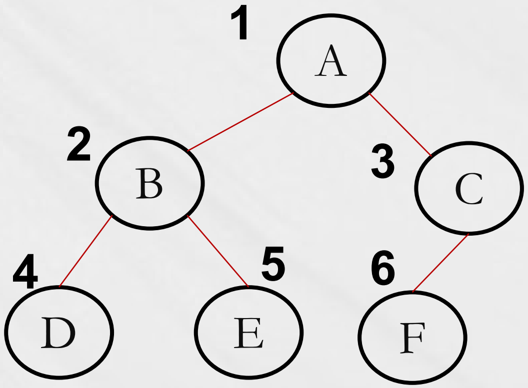
中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

4.层序遍历

- 二叉树的层序遍历是指按层次依次访问同一层的结点，具体做法是：首先访问处于第一层的根结点，然后从左到右依次访问处于第二层的全部结点，再访问从左到右依次访问第三层的全部结点，……，最后从左到右依次访问最下层的全部结点。

- 可以利用队列来实现二叉树的层序遍历：最初队列中只有根结点；取出队首结点进行访问，并把该结点的左孩子和右孩子依次入队；重复上述过程直至队列为空。

- 算法如下：

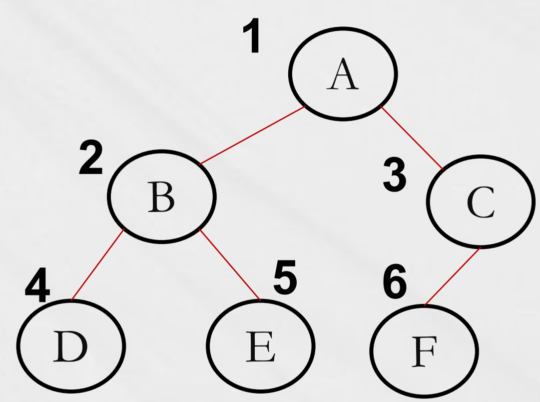


中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

算法：
结点先进队列
出队列时在访问

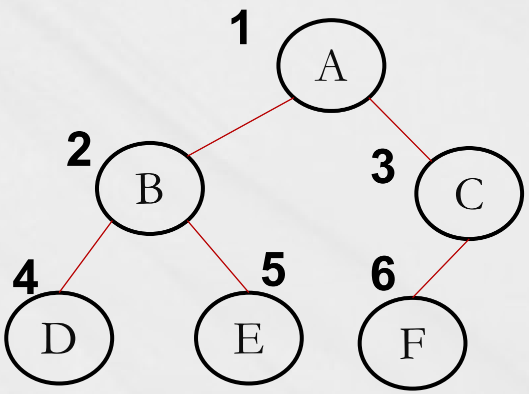
```
void levelorder(BTree   p)
{   BTree   p1;
     InitQueue(q);
     EnQueue(q，p);              // 将根结点p入队
     while( !QueueEmpty(q))  //队列空吗？
   {        p1= DeQueue(q);  //出队列一个节点
            visit(p1);
            if(p1->lchild!＝NULL) EnQueue(p1->lchild);
            if(p1->rchild!＝NULL) EnQueue(p1->rchild);
     }
}// levelorder
```

举例：



遍历算法的复杂度

- n： 节点数
- 时间复杂度：O(n)
- 空间复杂度：O(n)



根据遍历结果确定二叉树

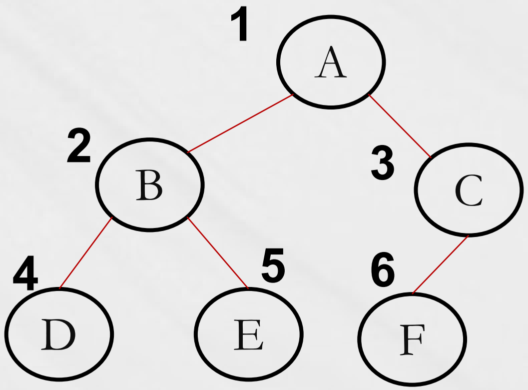
❖ 根据单种遍历结果确定二叉树？

❖ 根据两种遍历结果的组合确定二叉树？

先根+中跟 ？ ABDECF+DBEAFC

中跟+后根 ？ DBEAFC+DEBFCA

先根+后根 ？ ABDECF+DEBFCA



6.3.2 遍历算法的应用

1.创建二叉树

- 如果已知一棵二叉树的遍历序列，就可以创建该二叉树的二叉链表表示。在通常的遍历序列中，均忽略空子树，但是按照遍历序列创建二叉链表表示时，必须用特定的元素表示空子树，如输入“∧”。
- 按照**先根遍历的方式**创建二叉链表表示：**递归算法**

一定按照**先根遍历**完全二叉树顺序给出节点：

A、B、C 、 ∧ 、D、E、F、 ∧ 、 ∧ 、 ∧

1 2 3 4 5 6 7 8

一定按照**先根遍历**完全二叉树顺序给出节点：A、B、C 、∧ 、D、E、 ∧

```
void preCreateBTree(BTree  BT)
{
  data = getData(); // 读取数据
  if (data ==∧)          BT = NULL;
  else
{
BT = (BTree)malloc(sizeof(Bnode));
BT->data = data;
preCreateBTree (BT->lchild);
preCreateBTree (BT->rchild);
    }
} // preCreateBTree
```

如何按照中跟、后根遍历的方式创建二叉树？

2.求二叉树的高度（左子树或者右子树的高度+1）

```
int BTreeHeight(BTree BT)
{
   if (BT! = NULL)
      {  i = BTreeHeight(BT->lchild);
         j = BTreeHeight(BT->rchild);
         if (i<j)  return j + 1;   //加上根节点
            else    return i + 1;         }
      else   return 0;
}// BTreeHeight
```

3.结点数统计------按照中根遍历

统计二叉树中结点总数m和叶子结点个数n_0的算法(m和n0是全局变量，初始值为0）：

```
void  inCount(BTree BT)
{  if (BT != NULL)
   {    inCount(BT->lchild);
        m++;
if ((BT->lchild ==NULL) && (BT->rchild==NULL))
   n0++;
        inCount(BT->rchild);
   }
}// inCount
```

4.在二叉树中寻找指定元素

- 在二叉树中寻找指定元素，可以用任何一种遍历方式访问每一个结点，检查该结点的信息值是否为指定元素的值。**返回指向目的节点的指针。**

- 用**先根遍历**的思想寻找指定元素的算法：

```
BTree   preFind(BTree   p，ElemType item)
{
if (p!＝NULL)
{
      if (p->data == item) return p;
      if ((q = preFind(p->lchild, ElemType item)) != NULL)
           return q;
      if ((q = preFind(p->rchild, ElemType item)) != NULL)
           return q;
     return NULL ;
 }
else   return NULL ;
}// preFind
```

6.4 树与森林

什么是树？
什么是森林？
什么是树的孩子兄弟表示法？
树如何转换成为二叉树？
二叉树如何转换成为树？

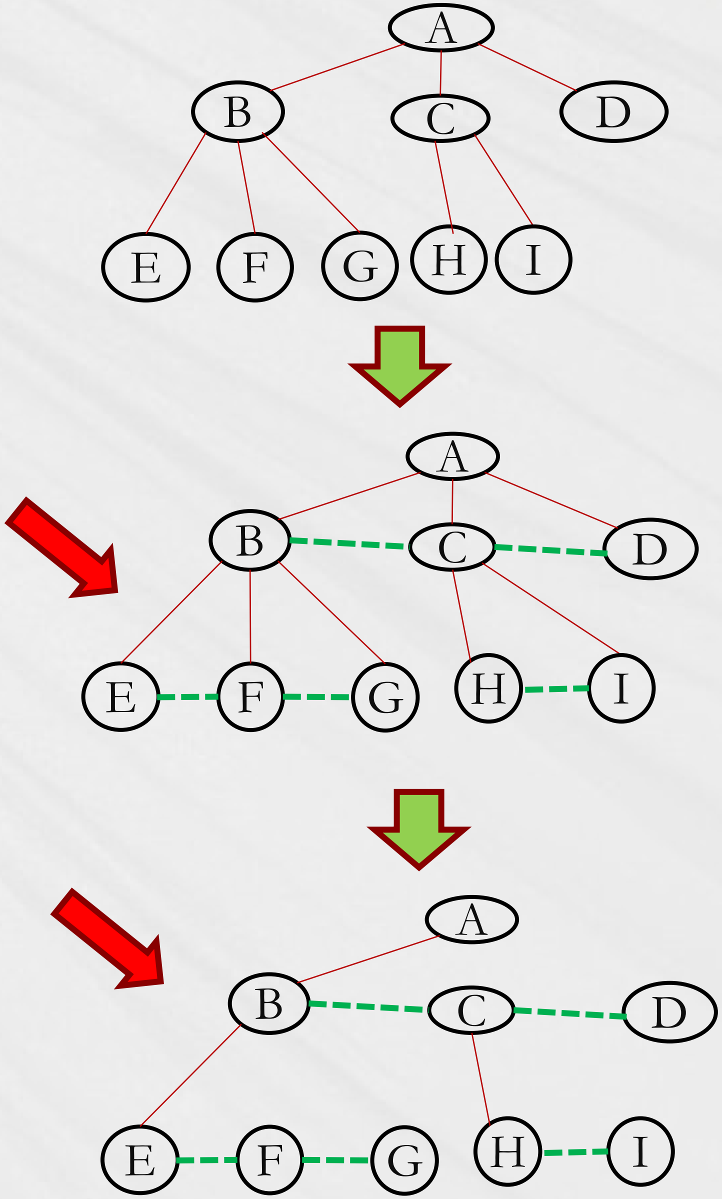
6.4.1 森林转换为二叉树

❖ 树转换为二叉树的基本步骤

①加线：在各兄弟结点之间用虚线相连。可理解为每个结点的兄弟指针指向它的下一个兄弟。

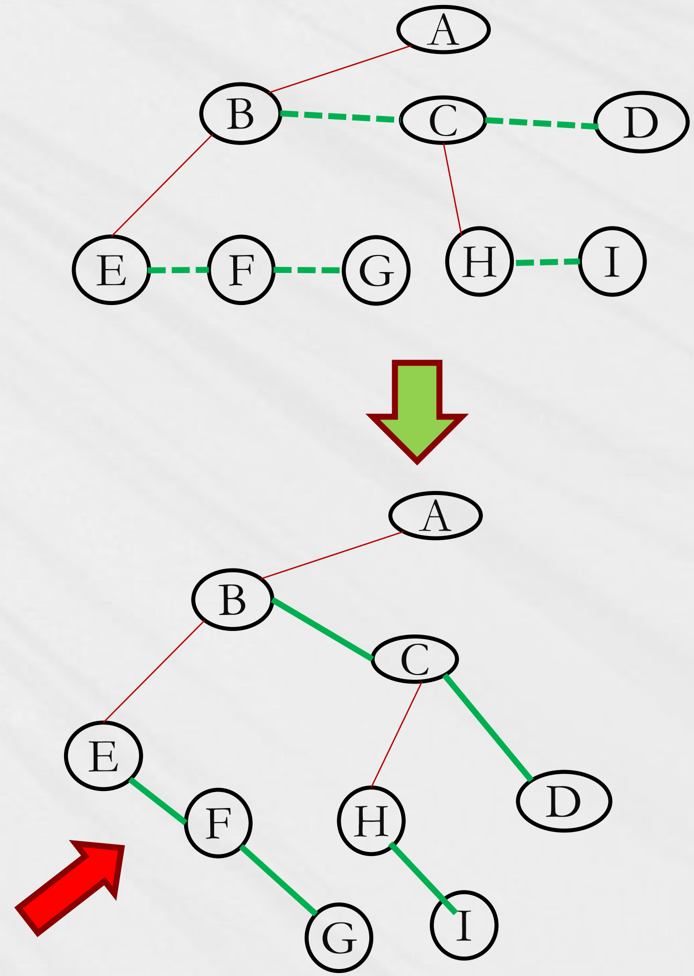
②抹线：对每个结点仅保留它与其第一个孩子的连线，抹去该结点与其他孩子之间的连线。可理解为每个结点仅有一个孩子指针，让它指向自己的第一个孩子。

③旋转：下页



6.4.1 森林转换为二叉树

❖ 树转换为二叉树的基本步骤



③旋转：

把虚线连接的节点变为右子树

实线连接的节点成为左子树

虚线改为实线

注意：树转变为二叉树仅有左子树（树根没有兄弟）。

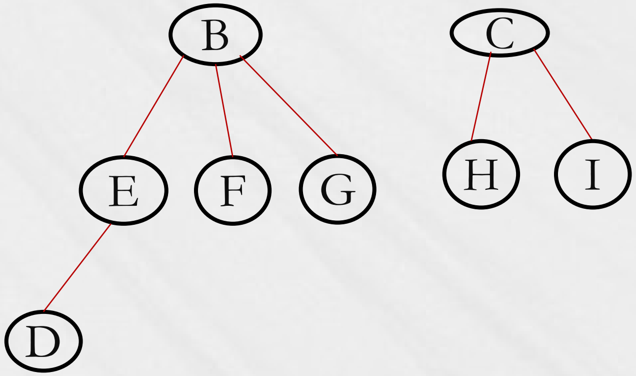
❖ 森林转换为二叉树：

将森林中的树排成一个序列，将森林中的第i+1棵树的根结点视为第i棵树的根结点的下一个兄弟。

①加线：除了在每棵树内部的各兄弟结点间加线外，还要在各棵树的根结点间加线。

②抹线

③旋转



注意：森林转变为二叉树有左、右子树（多个树根看成兄弟）。

6.4.2 二叉树转换为树或者森林

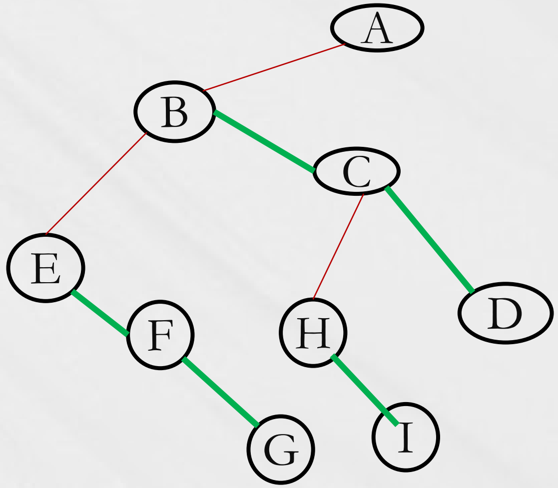
1）**加线**：若某结点是其双亲的左孩子，则将该结点的右孩子、右孩子的右孩子、……等等连续地沿着右孩子不断向右搜索到所有右孩子，都分别与该结点的双亲结点用虚线连接。这里的有右孩子其实该结点的兄弟，都是他们双亲的孩子。

2) **抹线**：把原二叉树中所有双亲结点与其右孩子的连线抹去。这里的抹去的连线是兄弟关系，包括森林中各树根结点之间的兄弟关系。

3) **整理**：把虚线改为实线，把结点按层次排列。

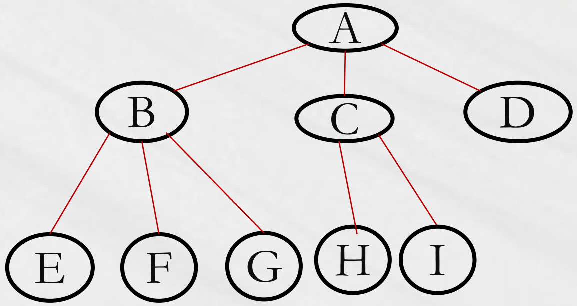
注意：每个节点的右子树的特点

举例：



中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

6.4.3树和森林的遍历



树的遍历：

❖ **先根遍历：**

递归定义：若根不空，访问根节点，按先根顺序遍历第1棵子树，而先根顺序遍历第2棵子树，……

❖ **中根遍历：**

递归定义：若根不空，按中根顺序遍历第1棵子树，访问根节点，按中根顺序遍历第2棵子树，……

❖ **后根遍历：**

递归定义：若根不空，按后根顺序遍历所有子树，访问根节点。

❖ **层序遍历：**类似于二叉树

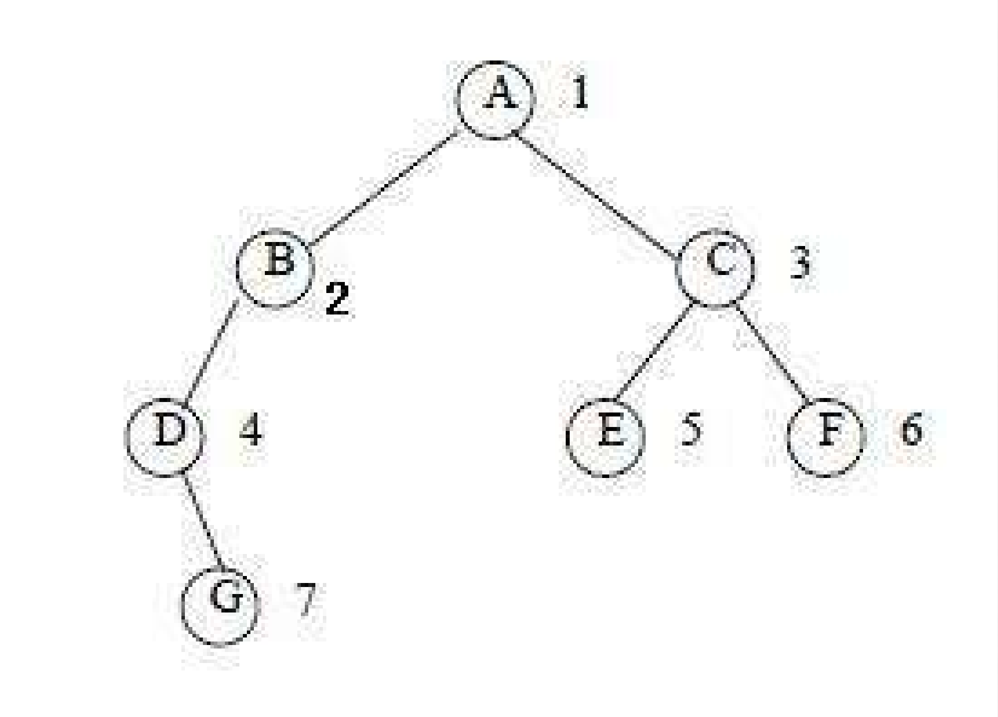
6.4.3树和森林的遍历

森林**的遍历：**

从第一颗树开始按照树的遍历顺序进行。

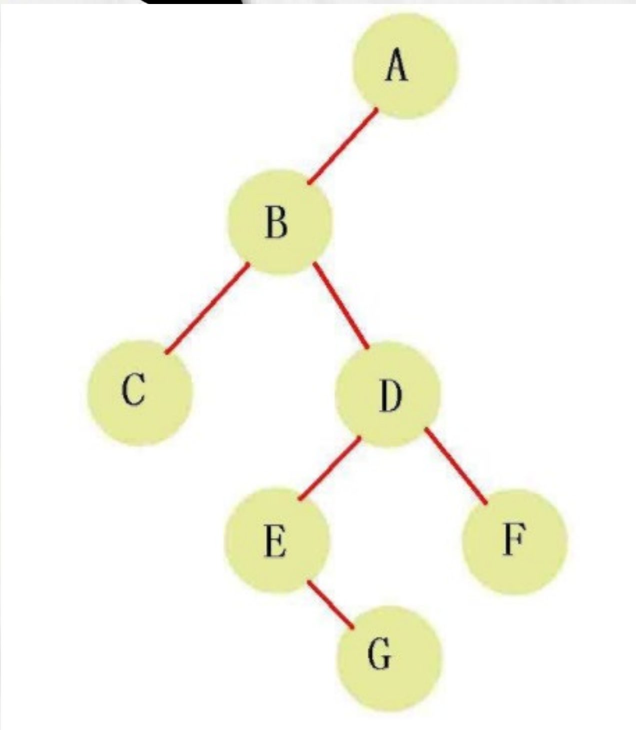
二叉树遍历练习

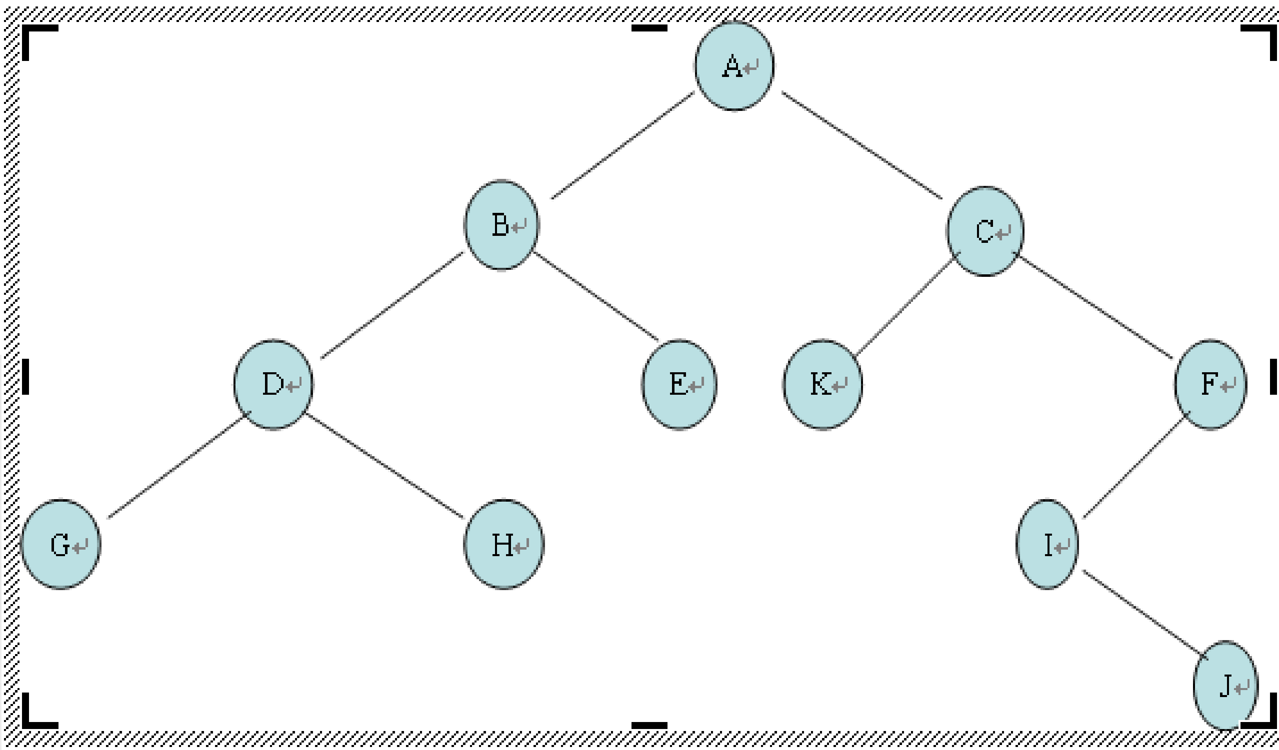
给出先跟、中跟、后跟，层序遍历结果



二叉树遍历练习

给出先跟、中跟、后跟，层序遍历结果



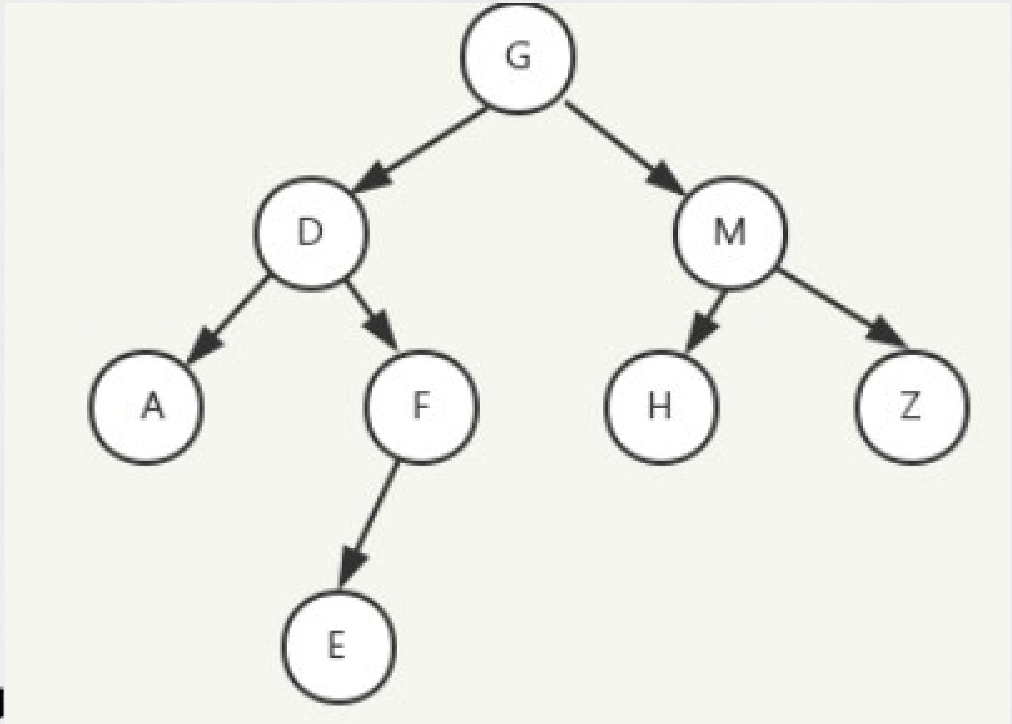


二叉树遍历练习

根据遍历结果还原二叉树

先跟（先序）：GDAFEMHZ

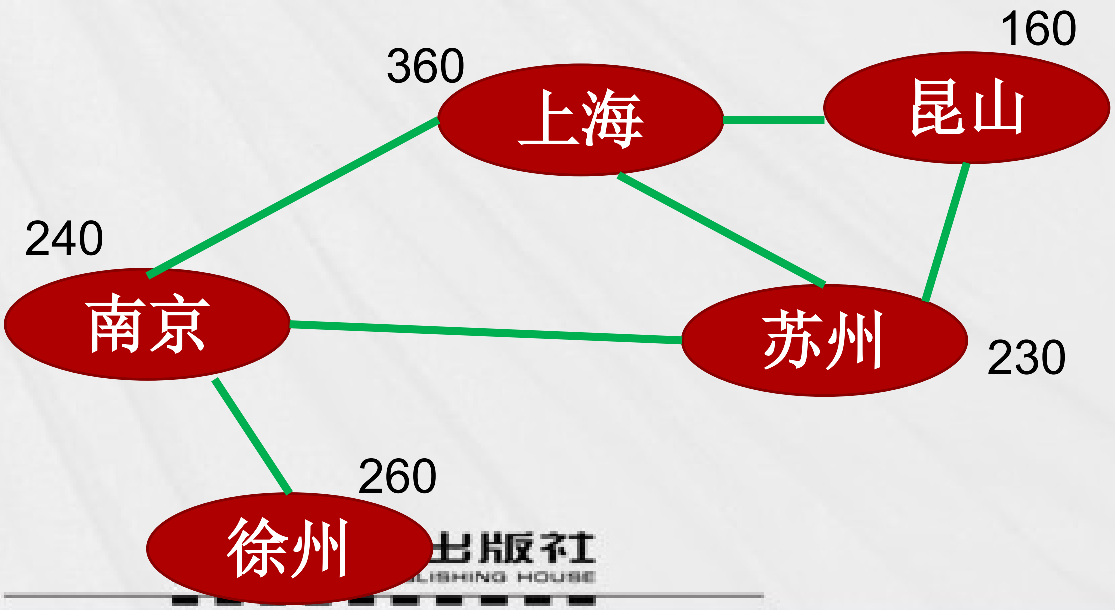
中跟（中序）：ADEFGHMZ



6.5 哈夫曼树及其应用

6.5.1 哈夫曼树-----最优二叉树，用于数据压缩

- **路径**：树中节点序列n1,n2,……nk，前一个节点是后一节点的双亲，称为n1到nk的一条路径。
- **路径长度**：路径上节点的数量减1。
- **树的路径长度**：从根结点到每一个结点的路径长度之和。
- **权值**：给节点赋予一个实际意义的值。



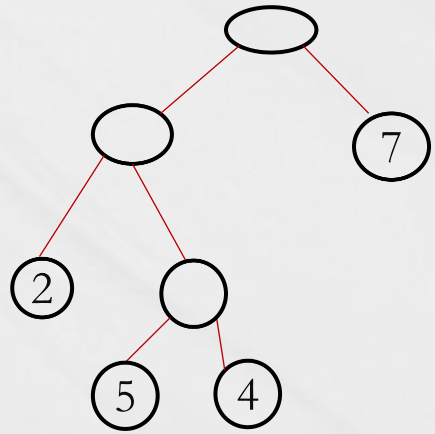
6.5 哈夫曼树及其应用

6.5.1 哈夫曼树-----最优二叉树，用于数据压缩

❖ **带权路径长度**：

设一棵二叉树有n个叶子结点，每个叶子结点拥有的权值分别为w_1、w_2、…、w_n，从根结点到每个叶子结点的路径长度分别为l_1、l_2、…、l_n。

❖ **树的带权路径长度**（WPL）：每个叶子的路径长度与该叶子权值乘积之和。



❖ 2*2+3*5+3*4+1*7

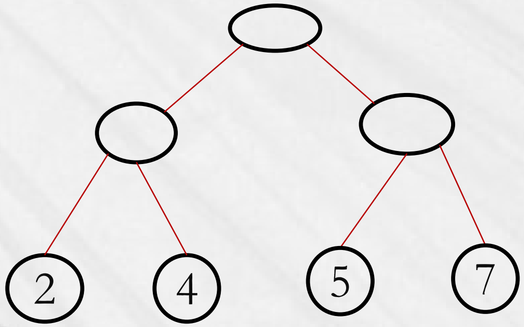
WPL=38

6.5 哈夫曼树及其应用

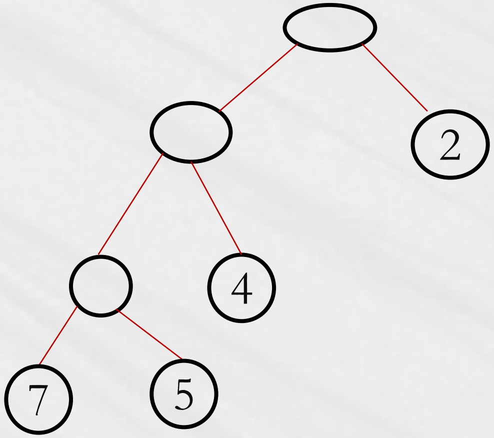
6.5.1 哈夫曼树-----最优二叉树，用于数据压缩

- 带权路径长度：
- 举例：对于一组具有确定权值的叶子结点，可以构造多种具有不同WPL的二叉树。权值为2、4、5、7的四个叶子节点。

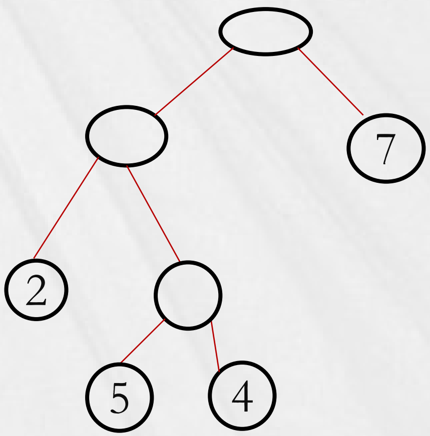
中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE



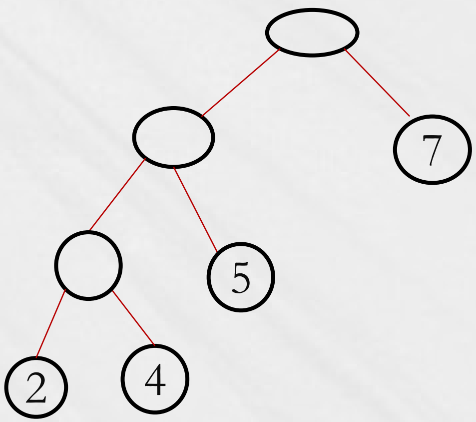
WPL=36



WPL=46



WPL=38



WPL=35

特征：权值大的节点靠近树根，权值小的远离树根。

哈夫曼树：具有最小带权路径长度的二叉树。

6.5.2哈夫曼算法----如何构造哈夫曼树

若已知n个权值分别为w_1、w_2、…、w_n的结点，构造以这n个结点为叶子结点的哈夫曼树的步骤：

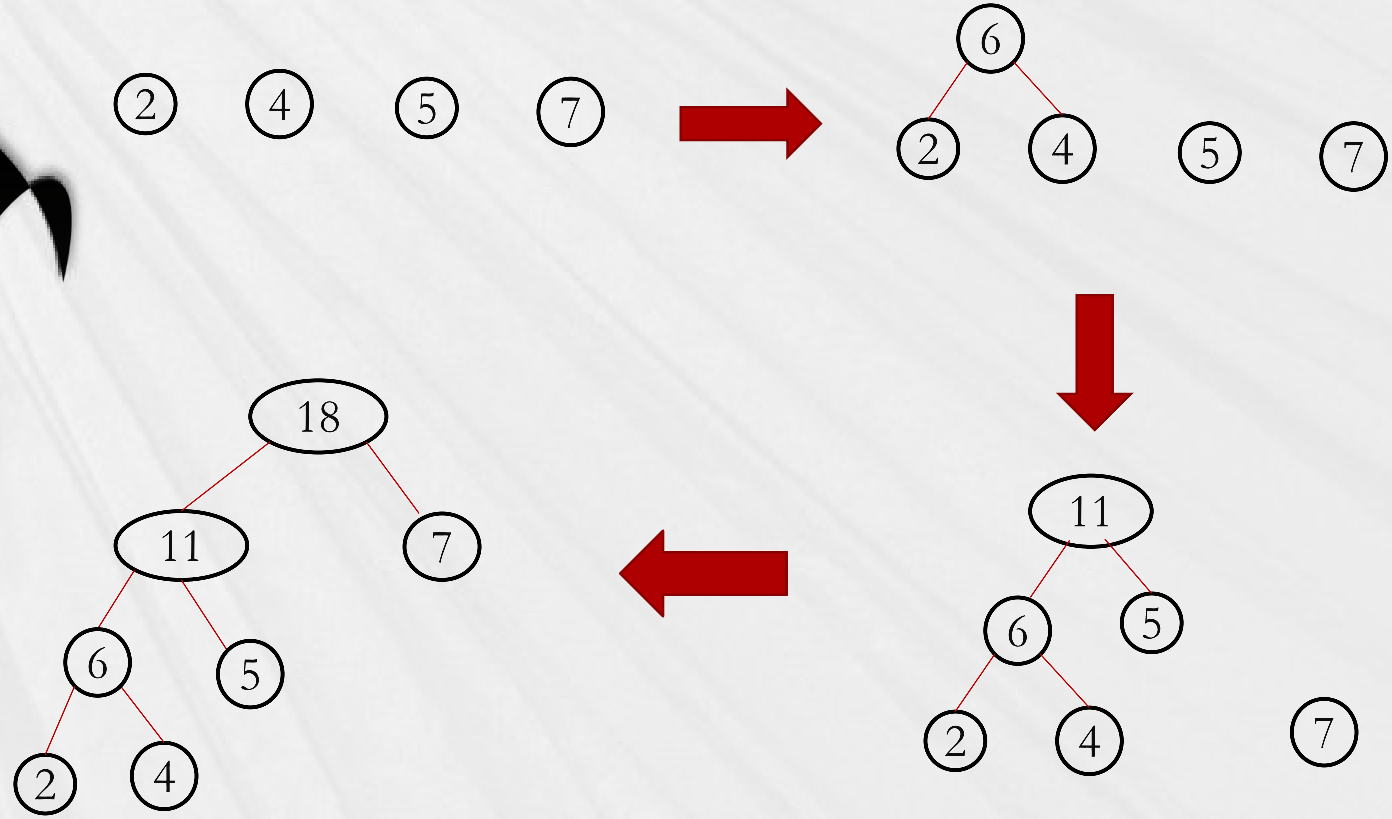
1) 把这n个叶子结点看做n棵**仅有根结点**的二叉树组成的**二叉树森林**。

2) 在二叉树森林中选择出**根节点权值最小**的两棵二叉树，以这两棵二叉树作为左右子树构造一棵新的二叉树，新二叉树的根结点权值为左右子树根结点权值之和，从森林中删除选择出的这两棵子树，同时把新二叉树加入森林，这时森林中还有n–1棵二叉树。

3) 重复第2)步直到森林中只有一棵二叉树为止。此树就是哈夫曼树。

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

约定：单结点子树作为右子树



WPL=35

推论：具有n个叶子结点哈夫曼树结点总数为2n–1

❖ 哈夫曼树的顺序存储结构---数组

```
# define MAXSIZE    20      // 结点的个数最多为20
typedef struct
{
int data;                  // 权值域
int lchild， rchild;       // 左、右孩子结点在数组中的下标
int  tag;    // tag=0 结点独立；tag=1 结点已并入树中
   }   Nodeh;
Nodeh   H[MAXSIZE];   //全局数组
```

◆哈夫曼算法

```
void huffman (r[ ]， n)      //r[ ]存n个叶子节点的权值
 {   for(i=1;i<=n; i++)       //初始化
     { H[ i ].tag = 0;H[ i ].lchild = 0;H[ i ].rchild = 0;H[ i ].data = r [ i ]; }
     j = 0;
     while (j < n – 1) {          //合并n-1次
       FindMin2(H， &x1， &x2， 2*n – 1); //分别为权值最小的元素下标
       j++;
       H[x1].tag = 1;    H[x2].tag = 1;
       H[n+j].data = H[x1].data + H[x2].data; // 合并子树根结点权值
       H[n+j].tag = 0;   H[n+j].lchild = x1;    H[n+j].rchild = x2;
        }
}// huffman
```

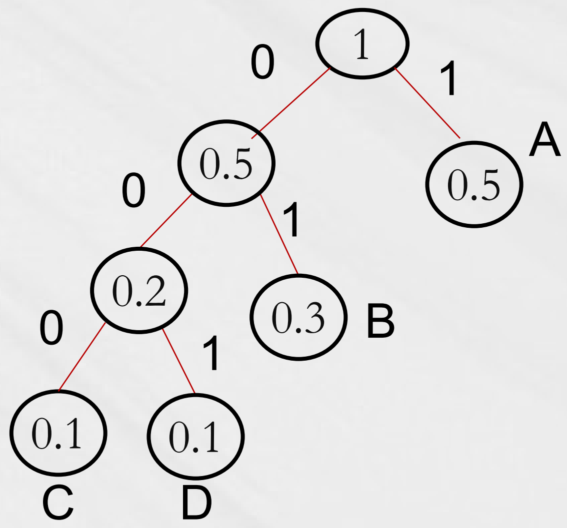
中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

6.5.3 哈夫曼编码

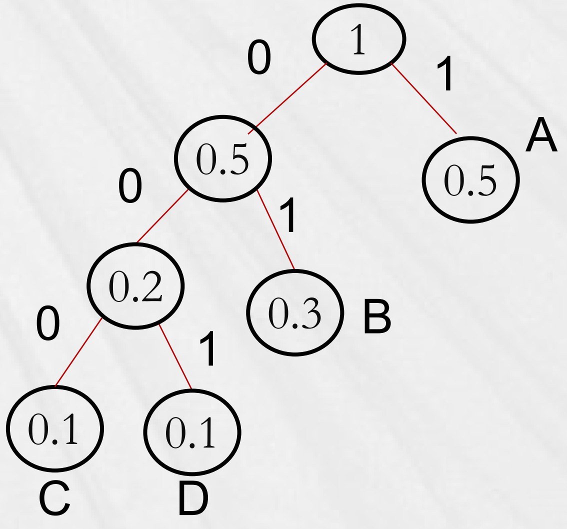
- 哈夫曼算法用于通信中字符的编码方法，称为哈夫曼编码
- 编码：将传送的文字转换为二进制字符串的过程。
- 译码：将二进制字符串恢复为原始文字的过程。
- 等长编码：字符出现的频度相同的情况下，编码效率高，编码和译码简单
- ASCII码：7位二进制表示一个字符
- 不等长编码：字符出现的频度相差较大的情况
- 前缀编码：任一字符的编码都不是另一字符编码的前缀

设需要编码的字符集为{d_1，d_2，…，d_n}，
各个字符的使用频率集为{w_1，w_2，…，w_n}，
以d_1，d_2，…，d_n为叶子结点，
以w_1，w_2，…，w_n为相应叶子结点的权值构造**哈夫曼树**。

规定从哈夫曼树的每个结点到其左孩子的树枝上标上0，到其右孩子的树枝上标上1，则从根结点到每个叶子结点所经过的树枝对应的0和1组成的序列便为该结点对应字符的编码。这样的编码我们称为**哈夫曼编码。**



举例：设需要编码的字符集为{A，B，C，D}，
各个字符的使用频率集为{0.5，0.3，0.1，0.1}，
构造哈夫曼树



A 1，B 01，C 000，D 001

A B A C A B D A

Haffnman: 1 01 1 000 1 01 001 1

哈夫曼编码的译码：从左到右扫描二进制串

本章小结

❖树作为一种层次结构，树的各种基本操作多是通过递归算法实现的。

❖二叉树中每个结点至多有两棵子树，有五个特有的性质。

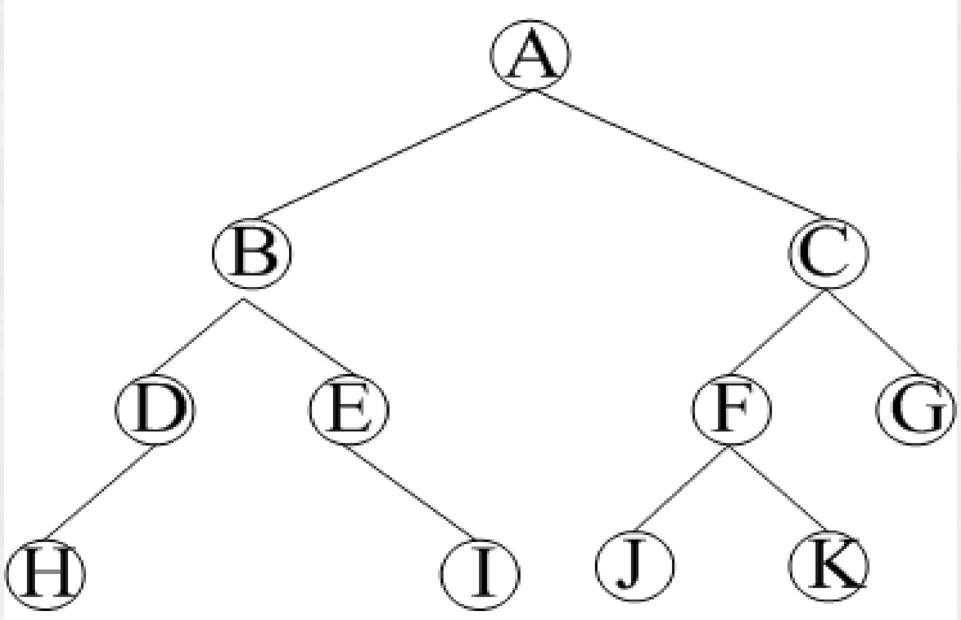
❖二叉树的遍历方法有：先根、中根、后根和按层次遍历。

❖二叉树和森林可以互相转换。

❖哈夫曼算法，通过从二叉树森林中选择加权路径长度最小的二叉树，有效的实现了最优二叉树的构造过程，而哈夫曼编码则是该算法在编码领域的重要应用，反映了二叉树重要的应用价值。

本章习题

1.对下图所示的二叉树，请写出按先根、中根、后根和层序遍历的结点序列。



本章习题

2. 假设二叉树采用二叉链表存储结构，编写一个算法，求出二叉树中的最大结点值。

3. 假设二叉树采用二叉链表存储结构，编写一个不同于书中的算法，求出二叉树中的叶子结点数。

4. 现有按先根遍历某二叉树的结果为ABCDEFGHI，中根遍历的结果为BCAEDGHFI，试画出这棵二叉树。

5. 试将第1题中的二叉树转换为森林。

6. 给定一组叶子结点的权值分别为3，5，7，9，11，请画出哈夫曼树的构造过程及最后结果。

7. 假定字符集{a，b，c，d，e，f}中各个字符的使用频率依次为0.07，0.09，0.12，0.22，0.23，0.27，试用哈夫曼树设计该字符集的哈夫曼编码。