2.5 树与二叉树

内容提要

1. 树的定义
2. 树的基本概念

   结点、结点度、根、支、叶结点

   子结点、父结点、兄弟结点

   树的度、路径、长度、深度

   森林、有序、无序

内容提要

3. 二叉树

二叉树的定义

二叉树的性质（每层结点个数、总结点数）

二叉树的存储结构:顺序存储、记录数组结构(结点、左子、右子)、链式存储结构（二叉链表三叉链表）

4. 特殊二叉树

满二叉树（性质）、完全二叉树（性质）、平衡二叉树、二叉排序树）

5. 二叉树的遍历操作（前序、中序、后序）

6. 树的存储结构:

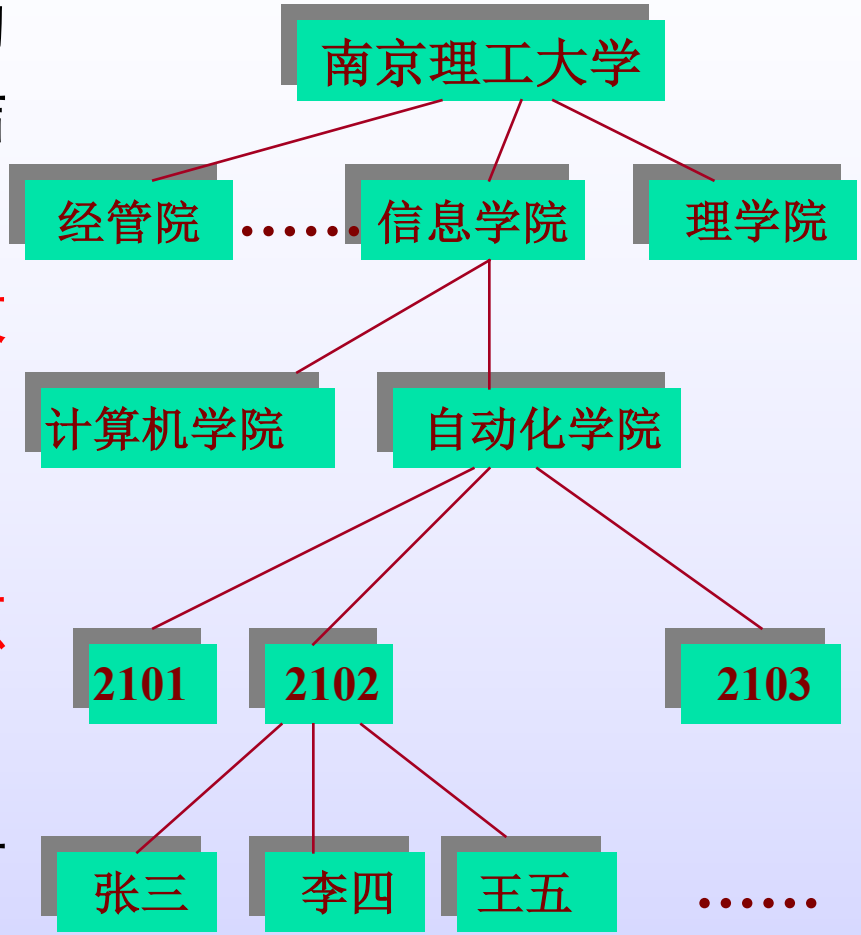
数组实现方法（双亲表示法）、链表实现方式（孩子表示法）、二叉链表实现方式（孩子兄弟表示法）

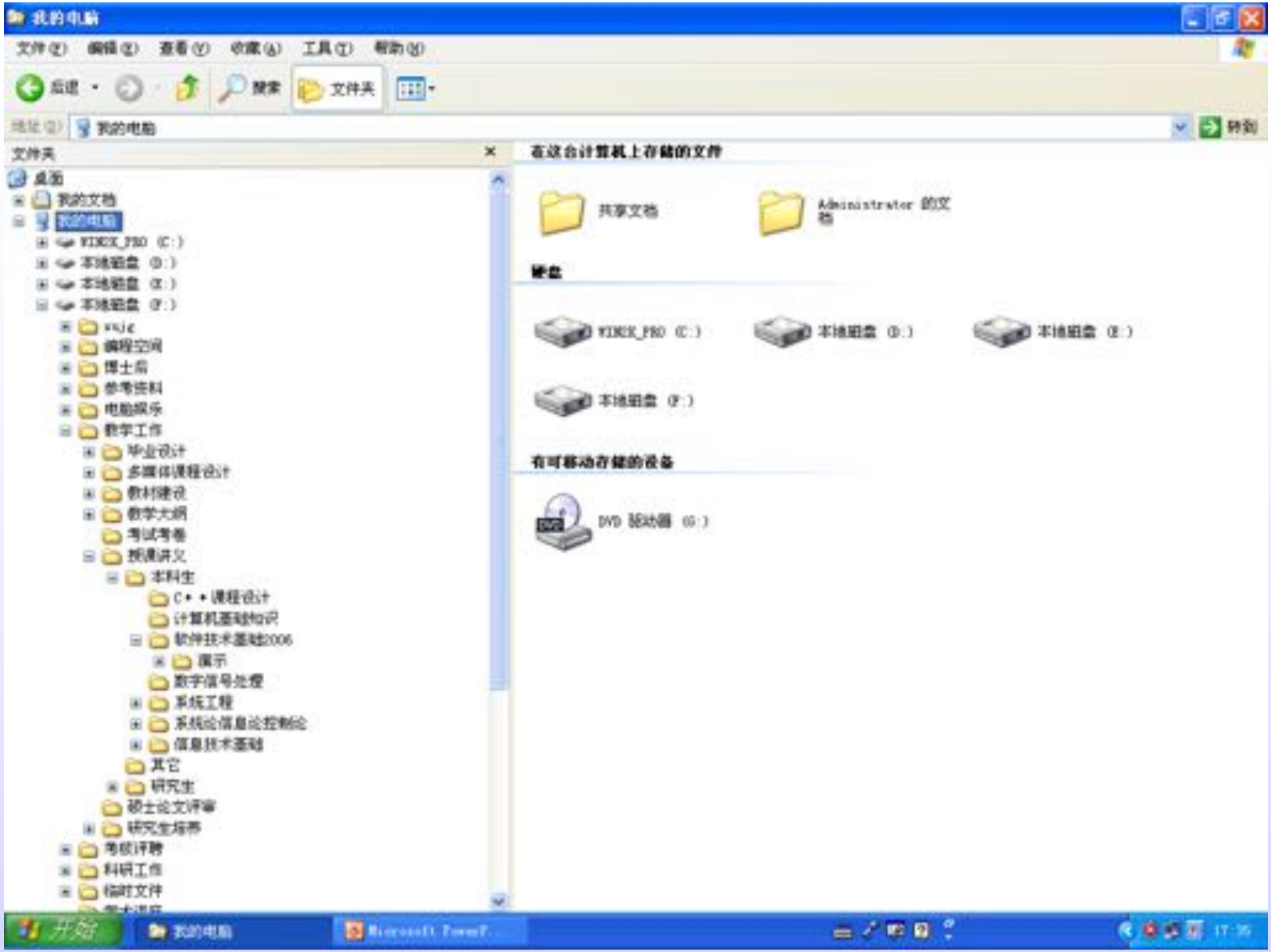
7. 树、森林与二叉树的转换

2.5.1 树的基本概念

树形结构是以分支关系来定义的层次结构。在客观世界中树形结构广泛存在，并应用于：

- 人类社会的族谱、家谱、行政区域划分管理；
- 各种社会组织机构；
- 在计算机领域中，用树表示源程序的语法结构；
- 在OS中，文件系统、目录等组织结构也是用树来表示的。




我的电脑
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退
搜索
文件夹
地址(D) 我的电脑
转到
文件夹
桌面
我的文档
我的电脑
WINXP_PRO (C:)
本地磁盘 (D:)
本地磁盘 (E:)
本地磁盘 (F:)
数学工作
毕业设计
多媒体课程设计
教材建设
教学大纲
考试考卷
授课讲义
本科生
C++课程设计
计算机基础知识
软件技术基础2006
演示
数字信号处理
系统工程
系统论信息论控制论
信息技术基础
其它
研究生
硕士论文评审
研究生指导
考核评聘
科研工作
临时文件
在这台计算机上存储的文件
共享文档
Administrator 的文档
硬盘
WINXP_PRO (C:)
本地磁盘 (D:)
本地磁盘 (E:)
本地磁盘 (F:)
有可移动存储的设备
DVD 驱动器 (G:)
开始
我的电脑

- 加强板 00235-LDTB-14-01-1
- 吊耳 00235-LDTB-14-01-2
- 人孔圈组装 00235-LDTB-14-01-3
  - 双头螺柱 M30×2，L=310／8.8级
  - 螺母 M30／8级
  - 垂制环 00235-LDTB-14-01-3-3
  - 把手 Φ16×290
  - 人孔盖 00235-LDTB-14-01-3-5
    - 盖板 00235-LDTB-14-01-3-5-1
    - 连接板 00235-LDTB-14-01-3-5-2
  - 销轴 20×48
  - 开口销 5×50
- 支耳 00235-LDTB-14-01-3-6
- 连杆1 00235-LDTB-14-01-3-7
- 连杆2 00235-LDTB-14-01-3-7
- 垂制环 00235-LDTB-14-01-3-9
- 上半环 00235-LDTB-14-01-4
- 下半环 00235-LDTB-14-01-5
- 滑式支座 00235-LDTB-14-01-6
- 筋板组 00235-LDTB-14-01-35
- 地脚螺栓 M36×600
- 垫圈 36
- 螺母M36／5级
- ……

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）

树的定义（逻辑结构）

- 树是一种数据结构：

  Tree=（D，R）

  其中：

  D 是具有相同特性的n个数据元素的集合；

  R 是D上逻辑关系的集合，且满足：

  - 在D中存在唯一的称为根的数据元素，没有前件；
  - D中其余数据元素都有且只有一个前件；
  - D中所有元素，或有若干个互不相同的后件（子树），或无后件（叶结点）；

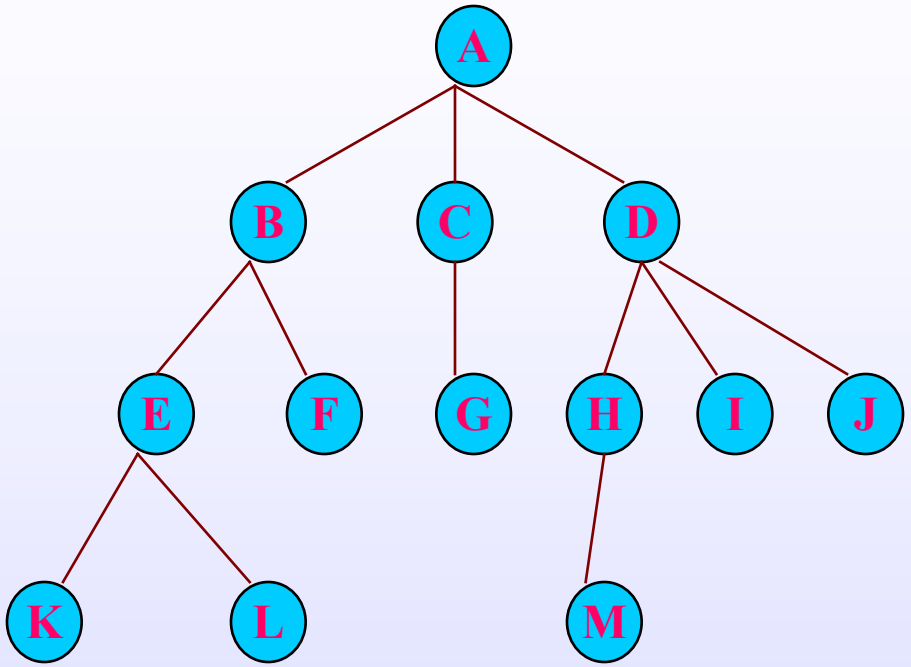
  则称Tree为树。

二、树的表示形式

1. 一般形式
2. 嵌套形式
3. 凹入形式
4. 广义表形式

1. 一般形式

(a)

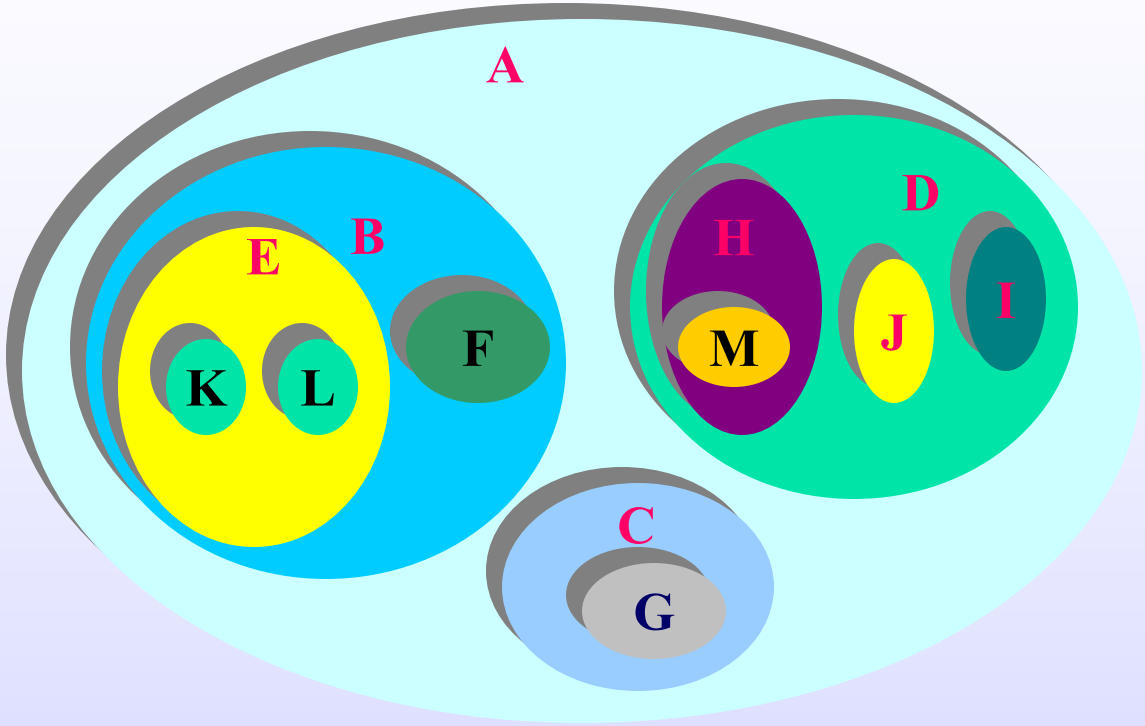


(b)

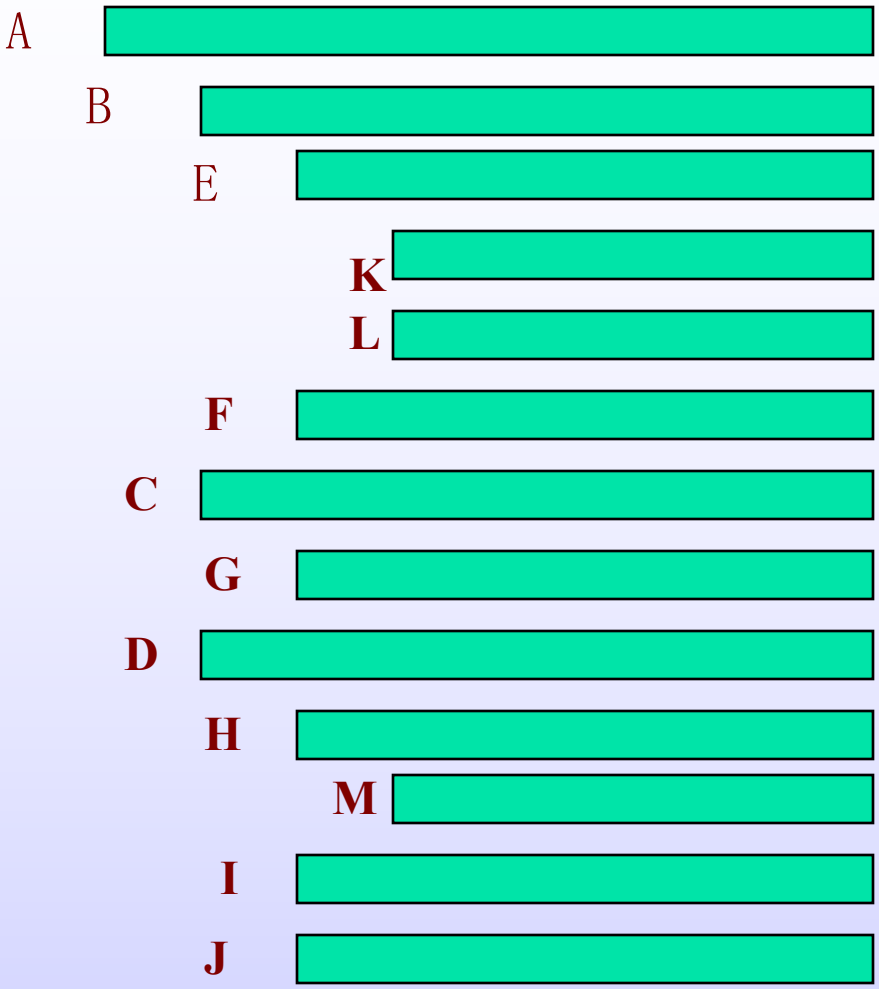
(a)只有根结点的树

(b)一般的树

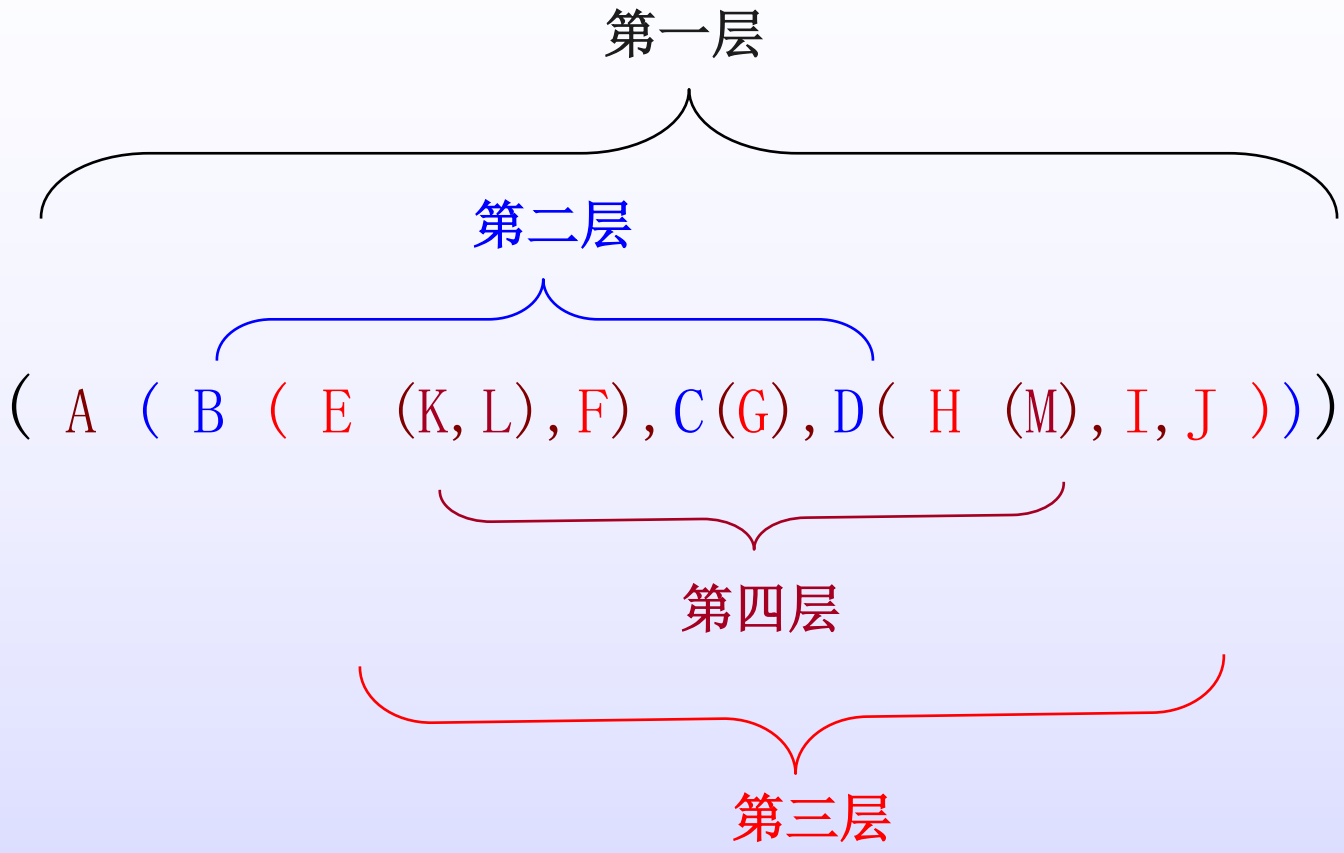
2. 嵌套形式



树的表示(凹入形式)



树的表示(广义表形式)



基本术语

- 结点：包括一个数据元素及若干个指向其它子树的分支；例如，A，B，C，D等。
- 叶结点：无后件结点为叶结点；如K，L，M。
- 根结点：无前件的结点为根；例如，A结点。
- *子结点*：某结点后件为该结点的子结点；例如，结点A的子结点为B，C，D。
- *父结点*：某结点的前件称为该结点的父结点；例如子结点C，B，D的父结点为A。

基本术语续

- 兄弟结点：同一父亲的孩子之间互为兄弟结点（Sibling）；H，I，J互为兄弟。
- 路径：结点的序列n1，n2，…,nk(K≥1)是一条路径。
- 长度：长度等于路径中结点数-1。
- 结点度：结点拥有的子树数数目；例如，A的度为3。
- 树的度：树中结点的最大度数；上述树的度为3。
- 子树：以某结点的一个子结点为根构成的树称为该结点的一棵子树。

基本术语续

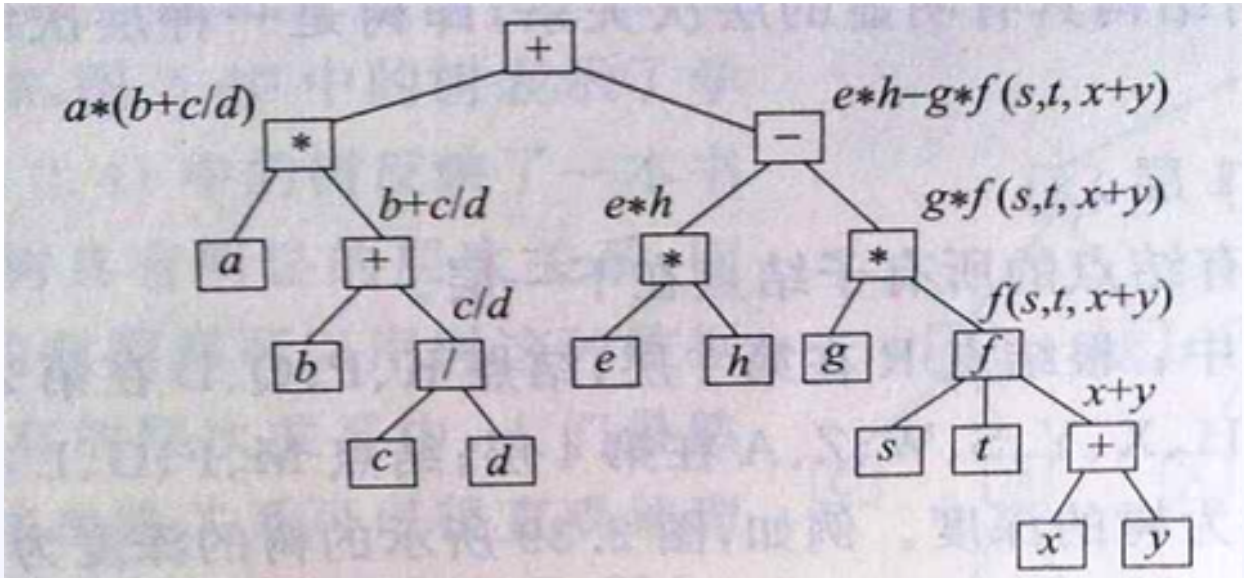
- 高度：从一结点到叶结点的最长路径为该结点的高度；例如，结点A到M的高度为4。
- 深度：从根结点到某结点的路径为该结点的深度；M的深度为4。
- 树的深度：树的最大层次称为树的深度。
- 森林：0棵或多棵互不相交的树的集合。对树中每个结点而言，其子树的集合即为森林。
- 有序：如果将树中结点的各子树看成从左至右是有顺序的（即不能互换），则称该树为有序树。否则，称为无序树。

例：用树结构来表示算术表达式

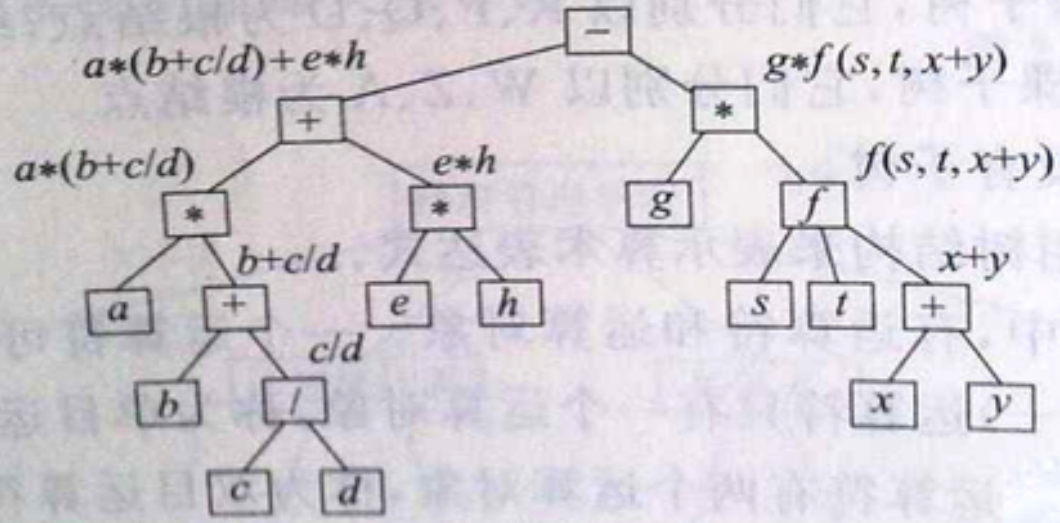
- 用树来表示算术表达式的原则如下：
- 表达式中的每一个运算符在树中对应一个结点，称为运算符结点。
- 运算符的每一个运算对象在树中为该运算符结点的子树（在树中的顺序为从左到右）。
- 运算对象中的单变量均为叶子结点。

例如：

a*（b＋c / d）＋e*h-g*f（s，t，x＋y）



(a) 表达式树之一



(b) 表达式树之二

2.5.2 二叉树及其基本性质

1 什么是二叉树

- 二叉树是另一种树形结构：

Binary_Tree =(D,R)

其中：D 是具有相同性质的数据元素的集合；

R 是在D上某个两元关系的集合，且满足：

- D中存在唯一称为根的数据元素，没有前件；
- D中其余元素都有且仅有一个前件；
- 每个结点至多只有两个子树；
- D中元素，或有两个互不相交后件，或无后件；
- 左、右子树分别又是一棵二叉树。

$$T = \begin{cases} n\text{个结点的集合}（n \geq 0） \\ T\text{ 的度} \leq 2 \\ \text{所有子树都有左、右之分} \\ （\text{次序不能任意颠到}） \end{cases}$$

- **二叉树不一定是树**（注意：尽管二叉树与树有许多相似之处，但二叉树不是树的特殊情形。）
- **二叉树可以为空；而树则不能为空；**
- **二叉树的结点最多只有两个直接后件**
- **二叉树的结点的子树分左子树和右子树，而树则无此区分。**
- **二叉树是有序树　　二叉树的度<=2**

二叉树的五种基本形态

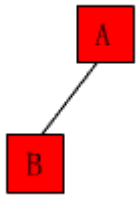
（a）

空结点

（b）

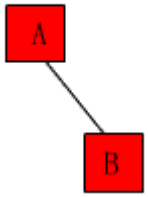
仅根结点

（c）



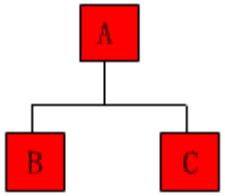
右子树为空的二叉树

（d）



左子树为空的二叉树

（e）

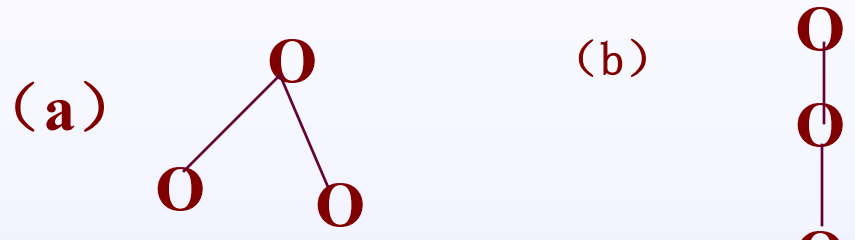


左、右子树非空的二叉树

二叉树与树的区别

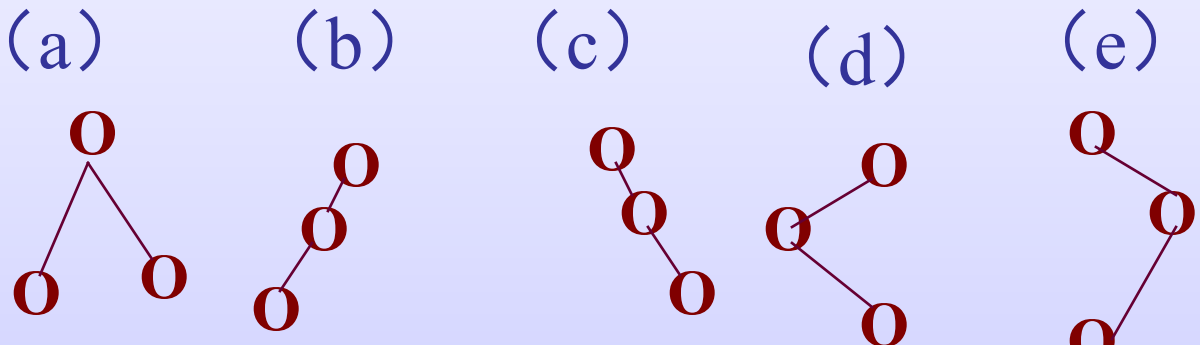
- 表达形式（对3个结点）

普通树



有两种不同形式

二叉树

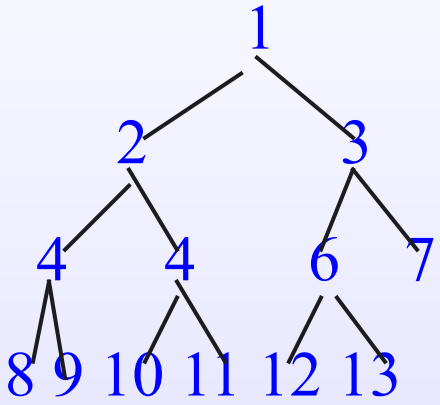


有五种不同形式

2 二叉树的性质

性质一

二叉树的第i层上至多有2^{i-1} 个结点（ $i \geq 1$）



利用归纳法证明：i=1时，只有一个结点，对的；

- 假设对所有的j，$1 \leq j \leq i$，命题成立，即在第j层上，至多有2^{j-1} 个结点。
- 由归纳假设，第i-1层上至多有2^{i-2} 个结点。由于二叉树的每个结点的度至多为2，故第i层上的最大结点数为第i-1层上的最大结点数的2倍，即 $2X2^{i-2} = 2^{i-1}$。

- 性质二

  深度为k的二叉树上最多含有2^k-1个结点（$k \geq 1$）。

  由性质一可见，深度为k的二叉树的最大结点数为：2^{k-1}。

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

性质三

- 性质3：在任意一棵二叉数中，度为0的结点（即叶子结点）总是比度为2的结点多一个。

有n_0个叶子结点，n_1个度为1的结点，n_2个度为2的结点，总的结点数为

$n=n_0+n_1+n_2$

$n=m+1$　　**所有进入分支的总数为m**

$m=n_1+2*n_2$

$n_0=n_2+1$

性质四

- 具有n个结点的二叉树，其深度至少为

$$[\log_2 n]+1$$

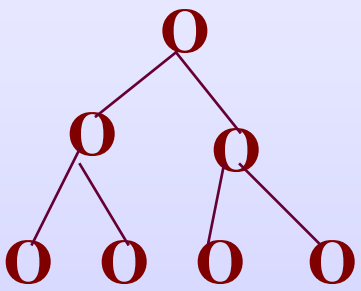
这个性质可以由性质2直接得到。

特殊二叉树

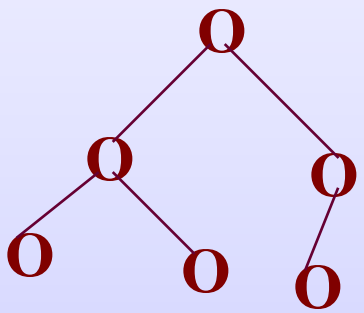
(1)满二叉树

若k为二叉树T的深度，且T中共有2^k-1个结点（$k \geq 1$），则称T为满二叉树。

（a） 满二叉树

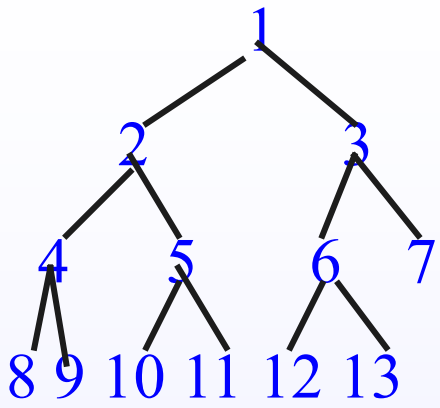


（b）非满二叉树



满二叉树的性质

- 若对满二叉树从第1层开始，自上而下、从左至右给每个结点从1开始编号的话，则称深度为k的满二叉树的结点编号满足：

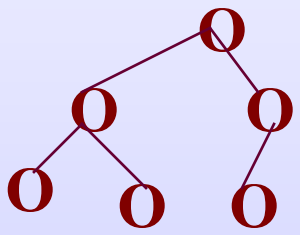


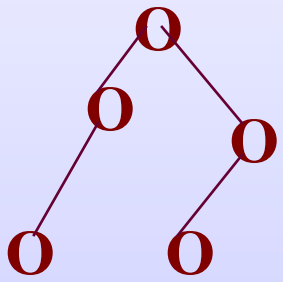
- 对某结点i（1≤ i ≤ 2k-1），其左子树的编号为2*i（偶数），其右子树的编号为 2* i +1（奇数）；（非叶结点）若i>1，则结点i父结点的编号为i/2(取整)。
- 根据这一性质，可用一维数组存储满二叉树的结点数据。知道一个结点的编号，经计算就能求出左、右子树的根及父结点的编号。
- 父结点：int(i/2)，左子结点：2*i，右子结点：2*i+1

（2）完全二叉树

- 深度为k的二叉树T，每层结点数目若满足：
  - 第i层(1 ≤ i ≤ k−1)上的结点个数均为2^{i-1};
  - 第k层从右边连续缺若干个结点(即只能从右至左不间断缺少)； 称这样的树为完全二叉树。

（a） 完全二叉树 （b） 非完全二叉树





特点: 叶结点只可能出现在层次最大的两层上.

完全二叉树的性质

- 设完全二叉树的结点总数为n，深度为k，某结点编号为i（1 ≤i ≤ n ），则有：
  - 若i>1, 则结点i的双亲结点的编号为i/2;
  - 若2* i ≤ n, 则结点i 的左子结点的编号为2* i, 否则, 结点i为叶结点;
  - 若2* i +1≤ n, 则结点i 的右子结点的编号为2*i+1
- 同理, 完全二叉树也可以采用一维树组作为存储结构, 且方法完全同满二叉树, 只不过$n \leq 2^k - 1$ 罢了.
- i—— 父结点：int(I/2),
- 左子结点：2*i ≤ n，右子结点：2*i+1 ≤ n

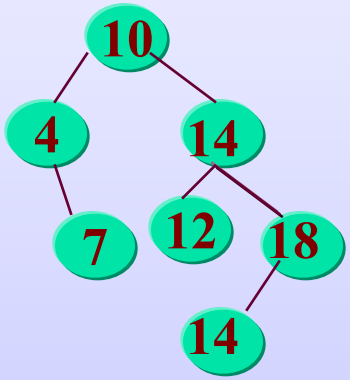
（3）二叉排序树定义一

- 二叉排序树
  - 或者是空二叉树；
  - 或者是：
    - 左子树上所有结点的值均小于根结点的值；
    - 右子树上所有结点的值均大于等于根结点的值；
  - 左、右子树本身又是一棵二叉排序树。
- 区别与有序树

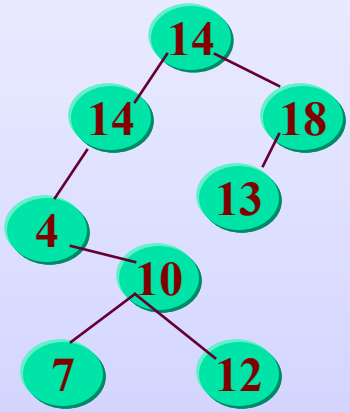
二叉排序树定义（二）

- X是二叉排序树T中的一个结点；
  - 所有的左后裔小于X；
  - 所有的右后裔大于等于X；
  - T可以为空树；
- T称为二叉排序树。

（a）二叉排序树

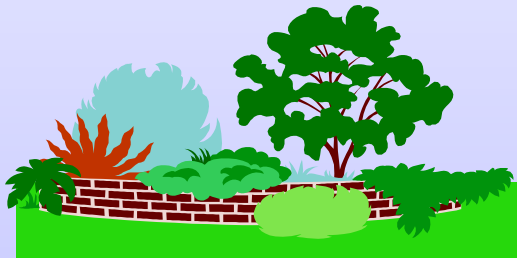


（b）非二叉排序树



2.5.3 二叉树的遍历

- 遍历（Traversing）是树形结构的一种重要运算，即按一定的次序系统地访问结构中的所有结点，使每个结点只被访问一次。
- 遍历的方法很多，常用的有：
  - 前序法（**PreOrder**）
  - 中序法（**InOrder**）
  - 后序法（**PostOrder**）



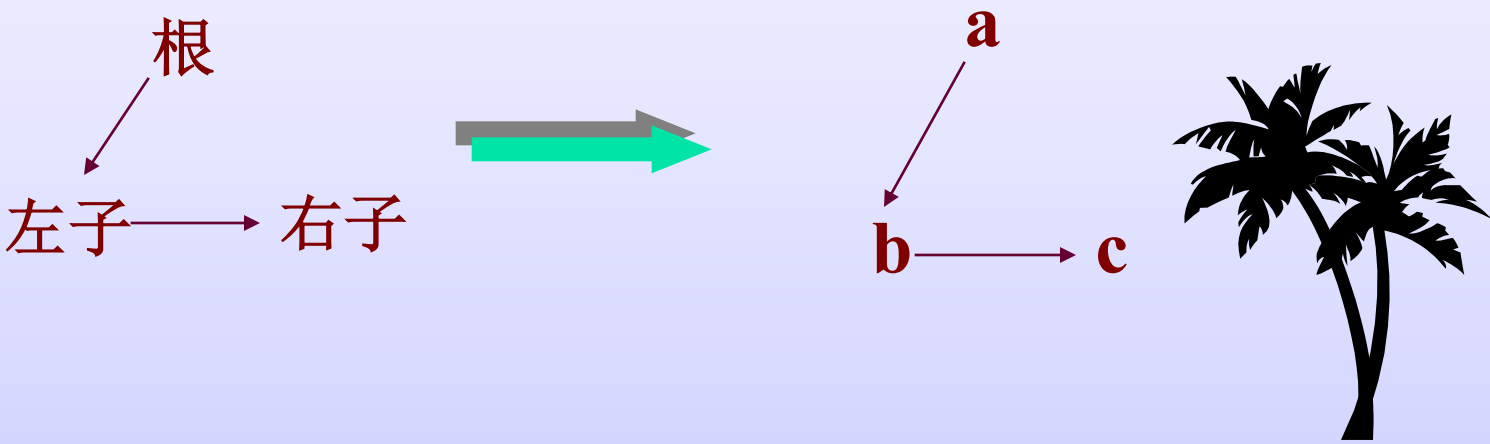
1、前序法（PreOrder）

- 方法描述：
  - 从根结点a开始访问，
  - 接着访问左子结点b，
  - 最后访问右子结点c。
- 即：

A 访问根结点

B 先序遍历左子树

C 先序遍历右子树



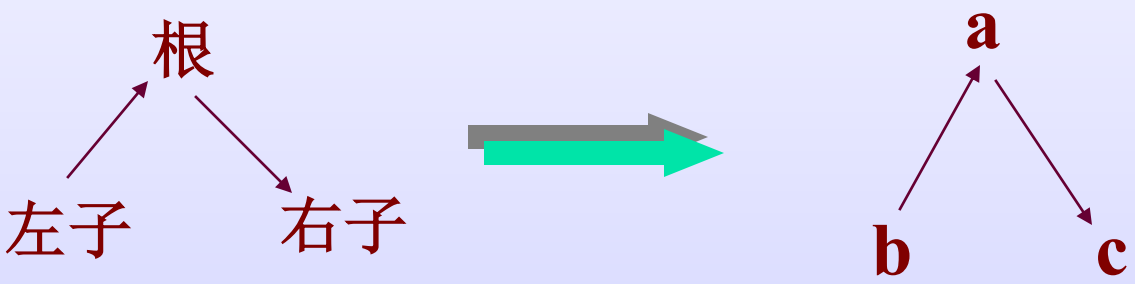
2、中序法（InOrder）

- 方法描述：
  - 从左子结点b开始访问，
  - 接着访问根结点a，
  - 最后访问右子结点c。
- 即：

A 中序遍历左子树

B 访问根结点

C 中序遍历右子树



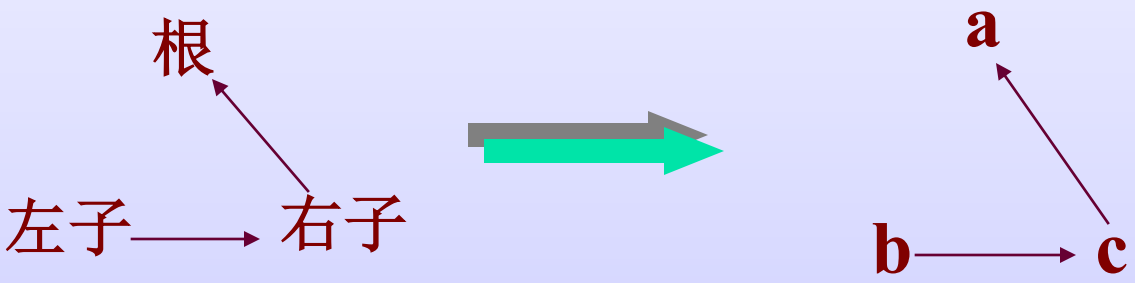
3、后序法（PostOrder）

- 方法描述：
  - 从左子结点b开始访问，
  - 接着访问右子结点c，
  - 最后访问根结点a。
- 即：

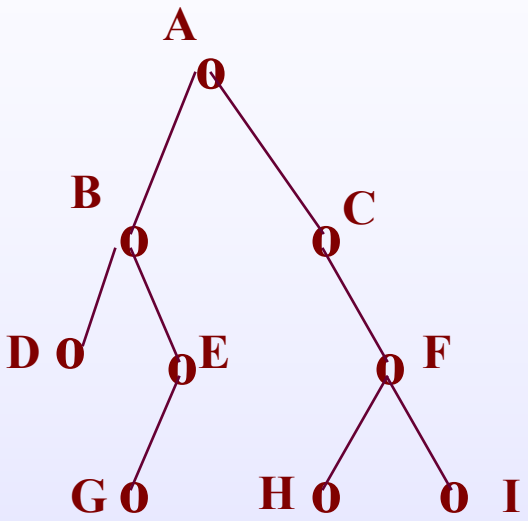
A 后序遍历左子树

B 后序遍历右子树

C 访问根结点



二叉树的遍历举例



- 前序遍历序列： A B D E G C F H I
- 中序遍历序列： D B G E A C H F I
- 后序遍历序列： D G E B H I F C A

课堂练习：

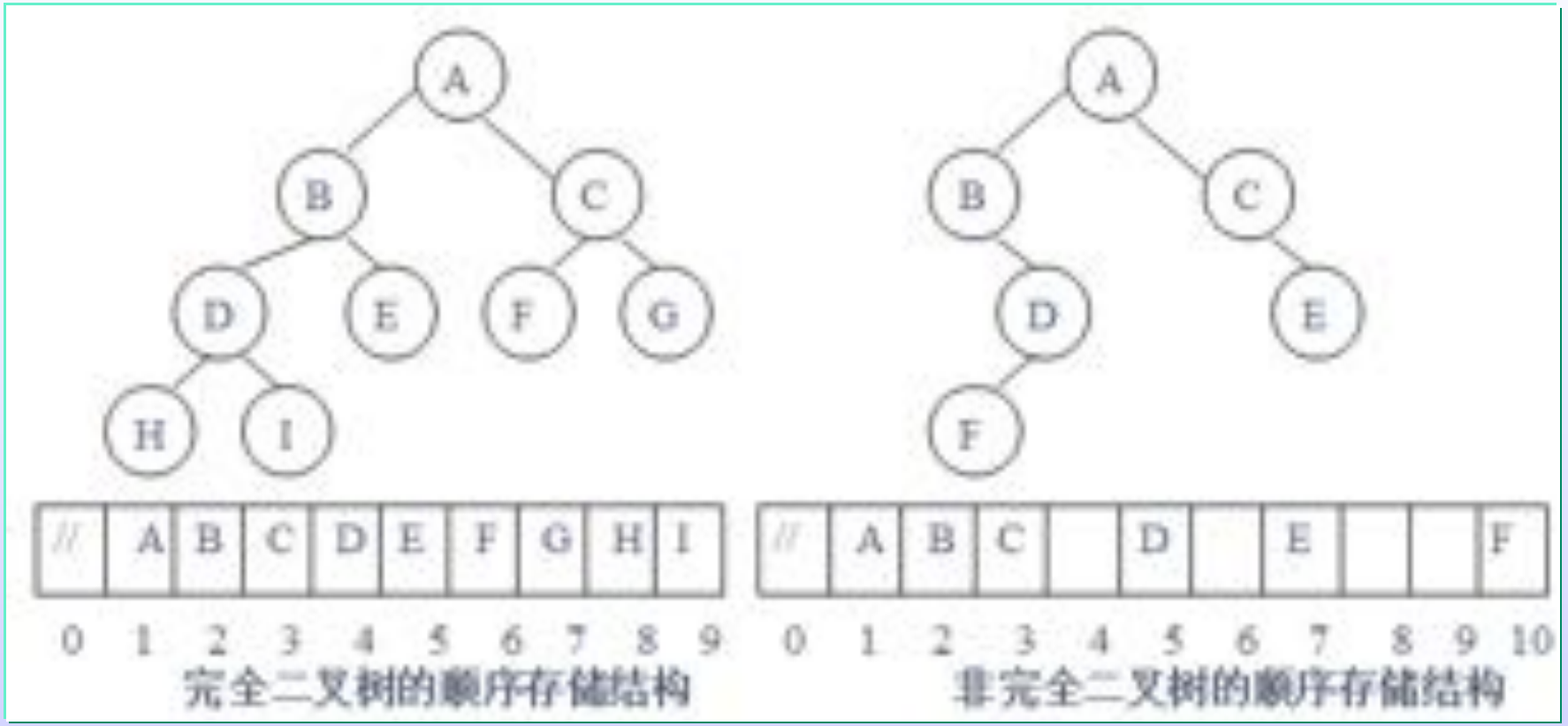
对给定的二叉树

写出三种遍历结果

2.5.4 二叉树的存储结构

1.顺序存储结构

该方法是把二叉树的所有结点，按从上至下、从左至右的顺序，存储在一块地址连续的存储单元中。通常，用一维数组作为存储结构。

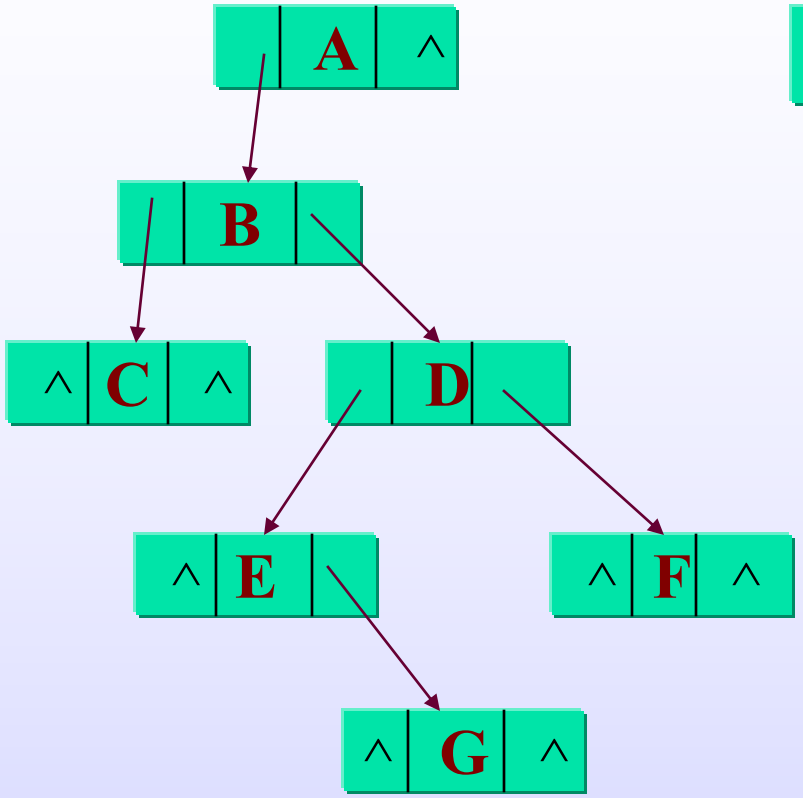


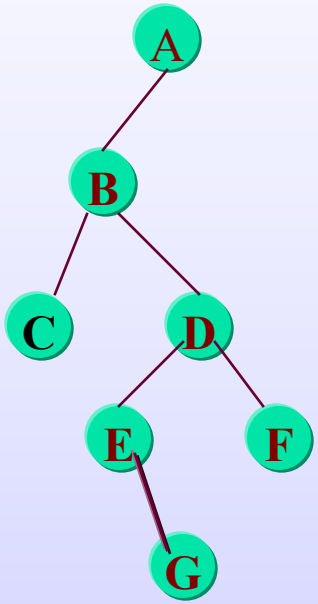
容易看出，一般的二叉树用顺序存储结构容易造成存储空间的浪费。

为克服顺序存储可能浪费存储空间的缺点，二叉树采用链式存储结构。

2.链式存储结构

二叉链表存储结构

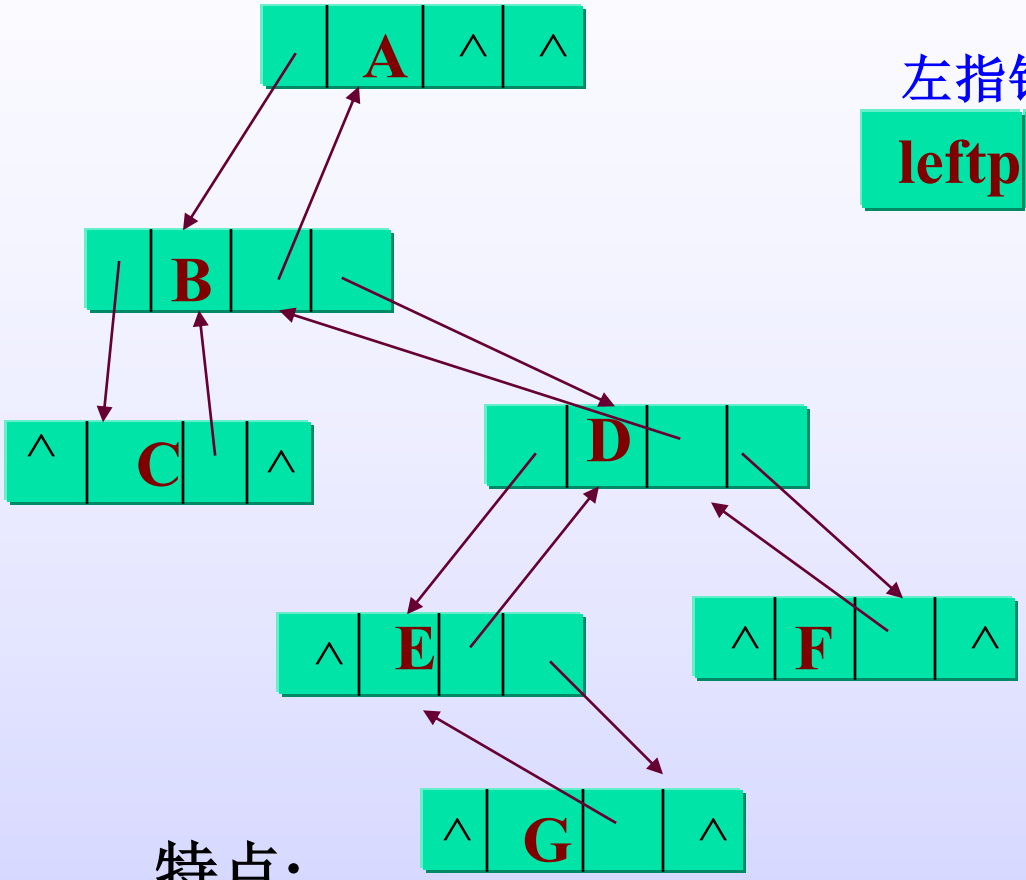




特点:

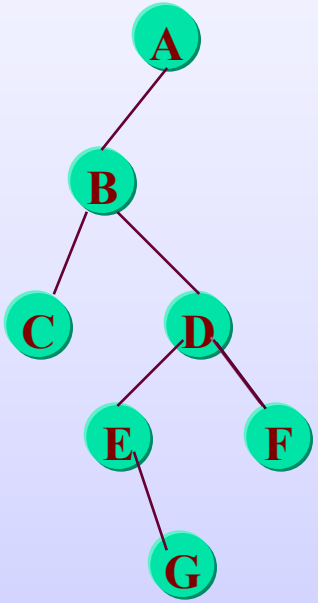
找子易, 找父难.

三叉链表存储结构



左指针 数据 父结点 右指针

leftp	data	parent	rightp



特点：

找子、找父都易。

二叉链表是二叉树最常用的存储结构，下面讨论的二叉树的遍历都是以二叉链表作为二叉树的存储结构。二叉链表中结点结构类型定义如下：

```
typedef char TelemType;
   /*TelemType为字符型，若需要可重新定义*/
 typedef  struct  BiTNode
 {
   TelemType data;
   struct BiTNode *lchild,*rchild;
 }BiTNode,*BiTree;
```

前序遍历算法

```
void PreorderTraverse(BiTree T)
{if(T) /*若二叉树非空，则前序遍历*/

  {
   printf("%c",T->data);
                        /*访问根结点*/
   PreorderTraverse(T->lchild);
              /*递归前序遍历左子树*/
   PreorderTraverse(T->rchild);
              /*递归前序遍历右子树*/
   }
}
```

中序遍历二叉树的递归算法如下：

```
void InorderTraverse(BiTree T)
{
 if(T) /*若二叉树非空，则中序遍历*/
{
    InorderTraverse(T->lchild);
                /*递归中序遍历左子树*/
    printf("%c",T->data);
                        /*访问根结点*/
    InorderTraverse(T->rchild);
                /*递归中序遍历右子树*/
  }
}
```

后序遍历算法

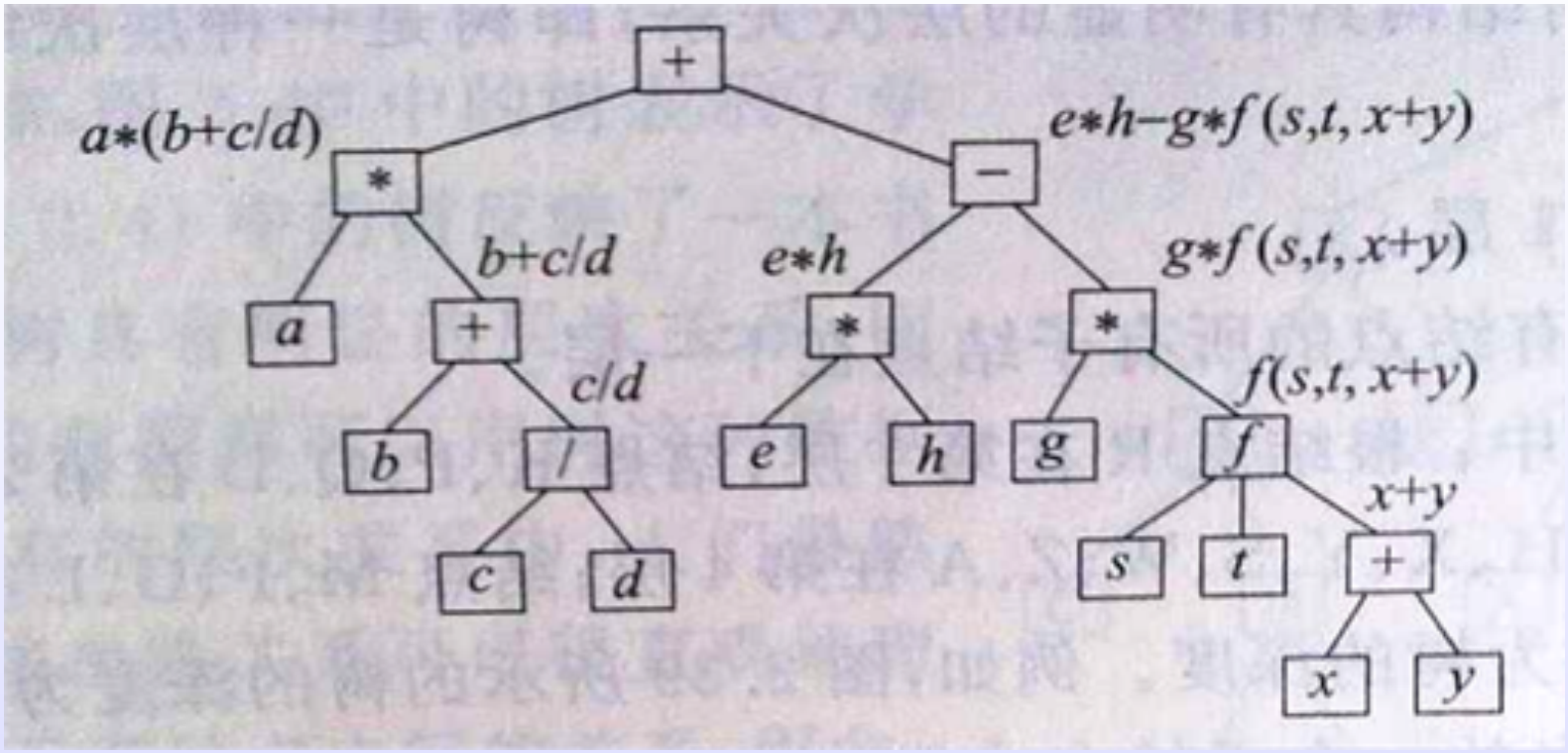
```
void PostorderTraverse(BiTree T)
{
    if(T)/*若二叉树非空，则后序遍历*/
     {
      PostorderTraverse(T->lchild);
                    /*递归后序遍历左子树*/
      PostorderTraverse(T->rchild);
                   /*递归后序遍历右子树*/
       printf("%c",T->data);
                          /*访问根结点*/
      }
   }
```

上述二叉树遍历算法的时间复杂度均为O(n)

2.5.5 树的应用

一、有序树的二叉树表示

- 在计算机中对表达式进行分析和计算是一项重要的任务；
- 表达式可以用有序树表示；
- 树处理不方便，可以转化为二叉树处理；
- 转化原则：
  - 有序树T的结点与二叉树BT的结点一一对应；
  - 有序树T中某个结点N的第一个子结点N1,在二叉树BT中为对应结点N的左子结点；
  - 其他子结点，在二叉树BT中被依次链接成一串起始于N1的右子结点。 参考P139 图2.53



转换为二叉树：

表达式线性化

- 将表达式用有序树表示；

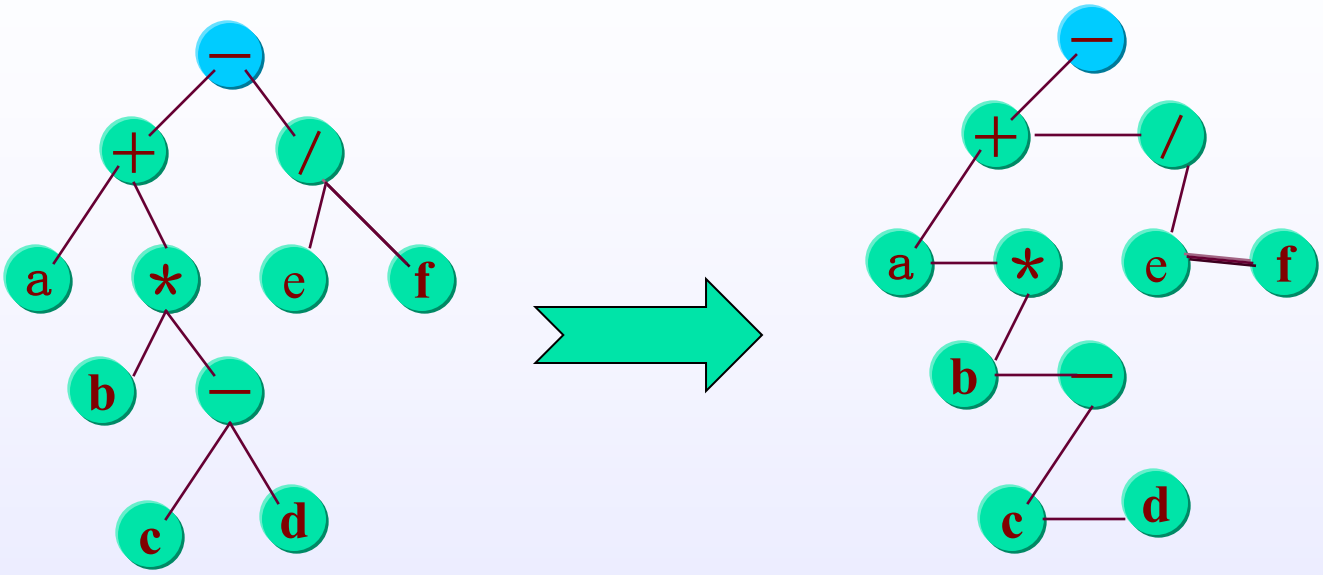
  每个叶结点为运算对象；

  每个非叶结点为运算符；

  每个子树对应一个子表达式。

- 将表达式树转化为二叉树；
- 对对应的二叉树进行中序遍历，其遍历序列即为后缀表达式，逆波兰表达式（维基，百度）；

a + b * （ c - d ） － e / f



中序遍历序列(逆波兰)：ａ ｂ ｃ ｄ －＊＋ｅ ｆ ／－

逆波兰表达式：背景

- 逆波兰表达式又叫做后缀表达式。在通常的表达式中，二元运算符总是置于与之相关的两个运算对象之间，所以，这种表示法也称为中缀表示。波兰逻辑学家J.Lukasiewicz于1929年提出了另一种表示表达式的方法。按此方法，每一运算符都置于其运算对象之后，故称为后缀表示。
- 在数据结构和编译原理这两门课程中都有介绍，例子

a+b ---> a,b,+

a+(b-c) ---> a,b,c,-,+

a+(b-c)*d ---> a,b,c,-,d,*,+

a+d*(b-c)--->a,d,b,c,-,*,+

a=1+3 ---> a=1,3 +

http=(smtp+http+telnet)/1024 写成什么呢?

逆波兰表达式：优点

- 它的优势在于只用两种简单操作，入栈和出栈就可以搞定任何普通表达式的运算。其运算方式：
- 　　如果当前字符为变量或者为数字，则压栈，如果是运算符，则将栈顶两个元素弹出作相应运算，结果再入栈，最后当表达式扫描完后，栈里的就是结果。

a b c d － * ＋ e f / －

（1） a b c d 入栈

（2）遇 ‘-’，c d 出栈，

运算后再压栈；

（3）遇 ‘*’ ，（c - d）和 b 出栈，

运算后再压栈；

（4）遇 ‘+’，b*（c-d）和a 出栈，　运算后再压栈；

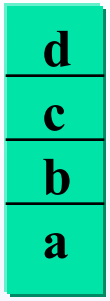
（5）遇 ‘/’，f e 出栈，

运算后再压栈；

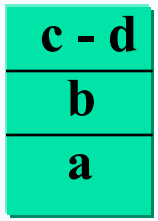
（6）遇 ‘-’，a+b*（c-d）

和e/f出栈，运算后再压栈。

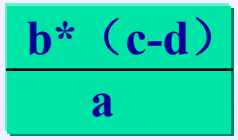
（1）



（2）

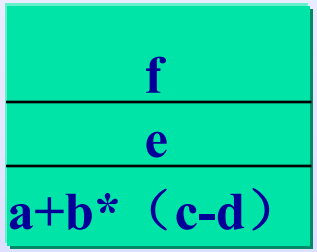


（3）

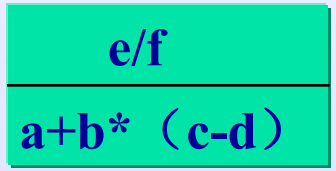


（4）

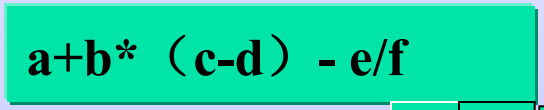
（4）



（5）



（6）



课堂练习

- 5+((1+2)*4)-3

1）写出有序树
2） 转换为二叉树
3） 写出中序遍历结果（逆波兰表达式）
4） 列出运算过程中栈的状态和动作

二、Huffman(哈夫曼）树

（1）Huffman树的定义

（2）构造Huffman树

（3）Huffman编码

（4）Huffman编码的译码

1. Huffman树的定义

- Huffman树也称为最优树，是一类带权路径最短的二叉树。
- 树的带权路径长度定义为：

$$WPL = \sum_{k=1}^{n} w_k L_k$$

其中：

n 是树中叶结点的个数

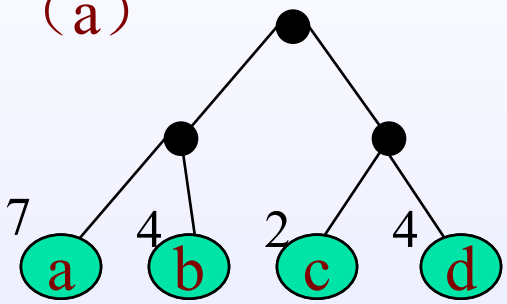
w_k 是第**k**个结点的权值

L_k 是第**k**个结点的路径长度

Huffman树举例

- 以下有三棵树：

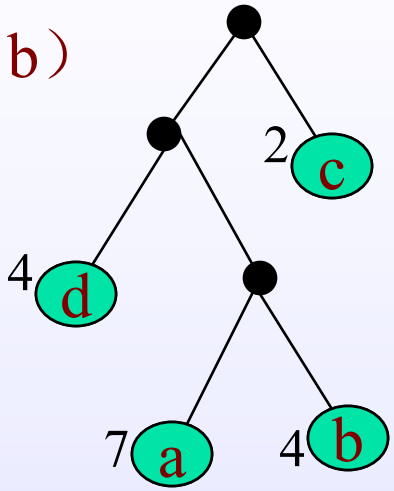
（a）



$WPL_a = 7x2+4x2+2x2+4x2$

$= 34$

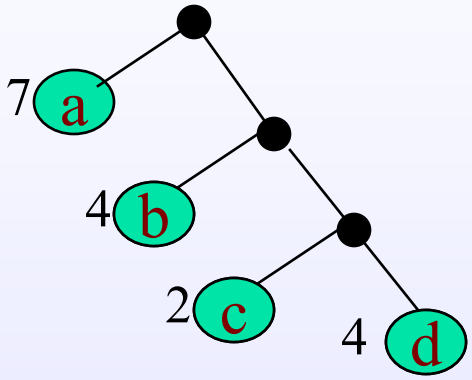
（b）



$WPL_b = 7x3+4x3+2x1+4x2$

$= 43$

（c）



$WPL_c =$

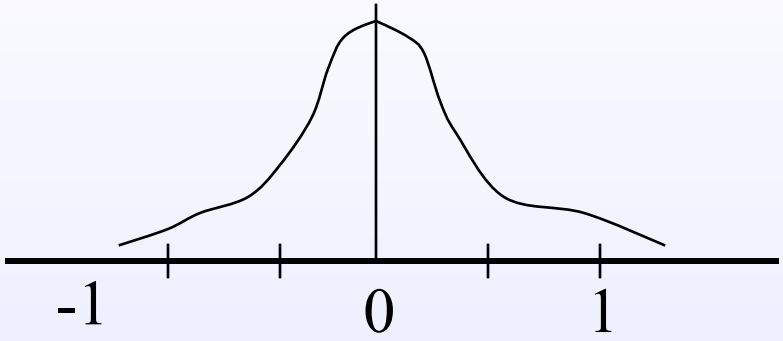
$7x1+4x2+2x3+4x3$

$= 33$ √

- **事实证明按哈夫曼树构造二叉树，可得到很好的特性，应用于实际问题，可提高处理效率。**

应用举例

- 由统计规律可知，考试成绩的分布符合正态分布：



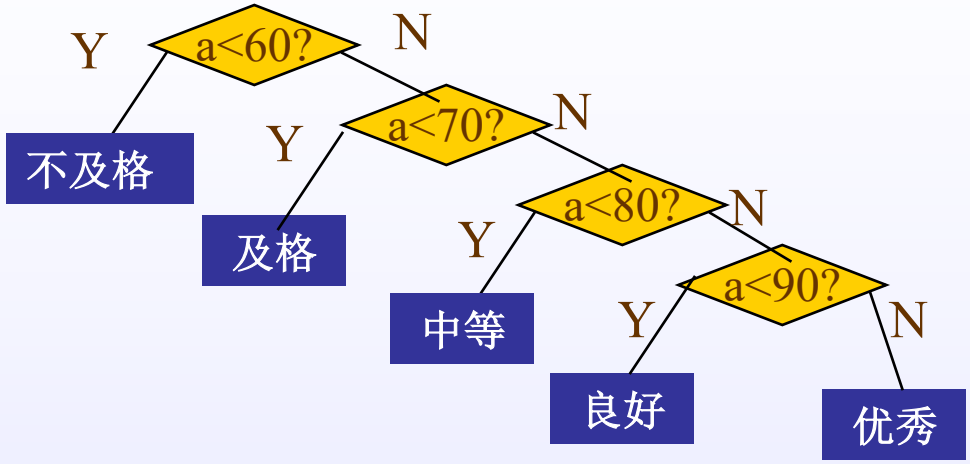
分数	0～59	60 ～69	70 ～79	80 ～89	90 ～100	
比例数	0.06	0.14	0.40	0.30	0.10	

- 根据正态分布规律，在60 ～89分之间的同学占84%，而不及格和优秀成绩的同学是少数。

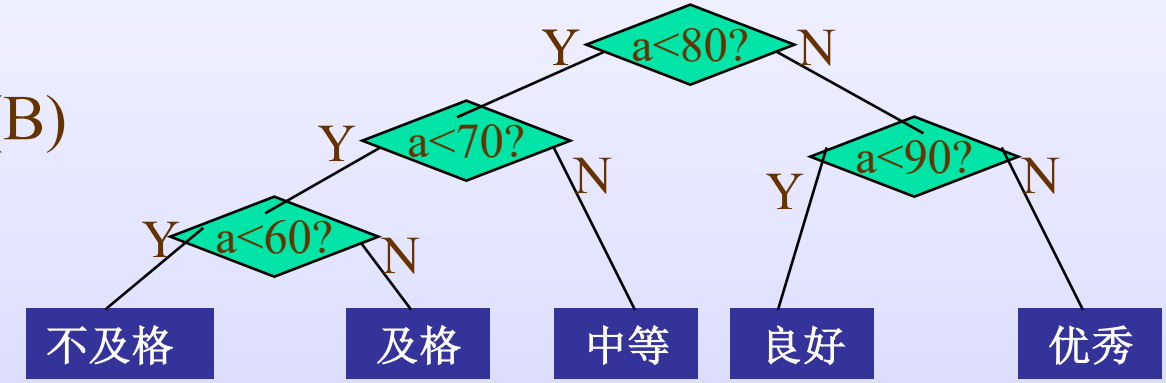
将百分制转换成五分制

- 判定树比较：

(A)



(B)



- 若输入1万个数据，按A的判定过程进行操作，约需比较3.2万次，而按B比较,则仅需2.2万次。

2. 构造Huffman树

构造Huffman树算法步骤：

1）将n个带权值w_i（i≤n）的结点构成n棵二叉树的集合T={T_1，T_2，……，T_n}，每棵二叉树只有一个根结点。

2）在T中选取两个权值最小的结点作为左右子树，构成一个新的二叉树，其根结点的权值取左右子树权值之和；

3）在T中删除这两棵树，将新构成的树加入到T中；

4）重复2）、3）步的操作，直到T中只含一棵树为止，该树就是Huffman树。

构造Huffman树举例

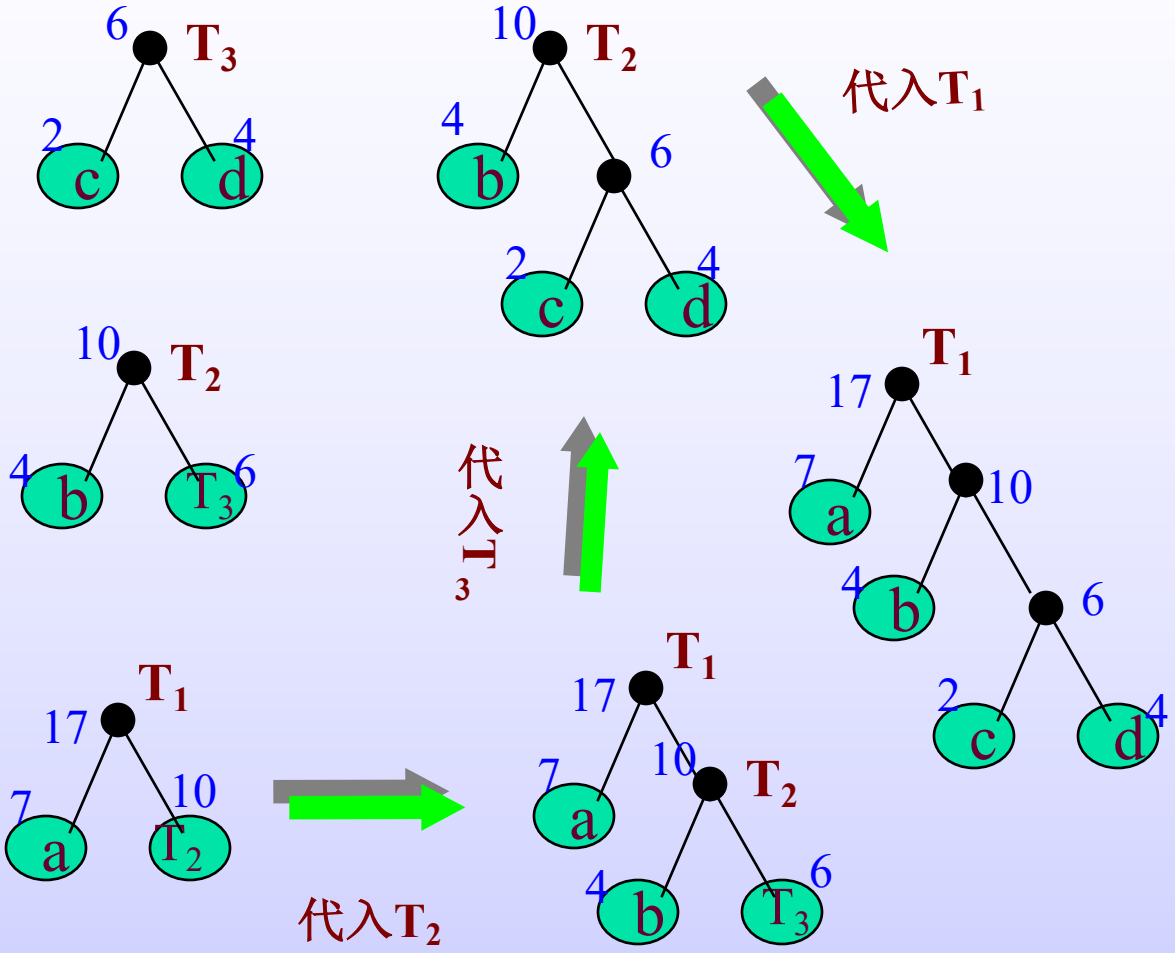
- 以权值分别为7，4，2，4的结点a、b、c、d构造Huffman树。T= { a b c d }

（a）T={ a b c d }

（b）T={ a b T_3 }

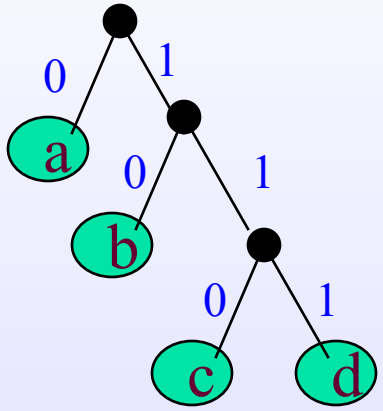
（c）T={ a T_2 }

（d）T={ T_1 }



3. Huffman编码

- 编码：用二进制数的不同组合来表示字符的方法。
- 前缀编码：一种非等长度的编码(任一个字符的编码都不是另一个字符编码的前缀)。



编码：A（0）

B（10）

C（110）

D（111）

方法约定：

1）左分支为‘0’(较小权重)

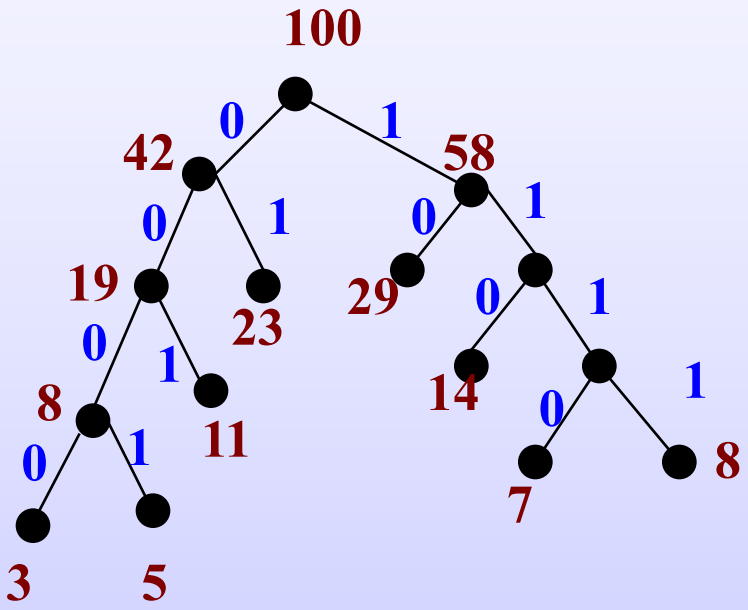
2）右分支为‘1’

3）由叶到根路径上字符组成的二进制串就是该叶结点的编码。

- Huffman编码：一种非等长度的编码。以给定权值的结点构造Huffman树，按二进制前缀编码的方式构成的编码为Huffman编码。

Huffman编码举例

- 在某系统的通信联络中可能出现8种字符，其频率分别为0.05、0.29、0.07、0.08、0.14、0.23、0.03、0.11，设权值分别为{5，29，7，8，14，23，3，11}，n=8，其Huffman树为：

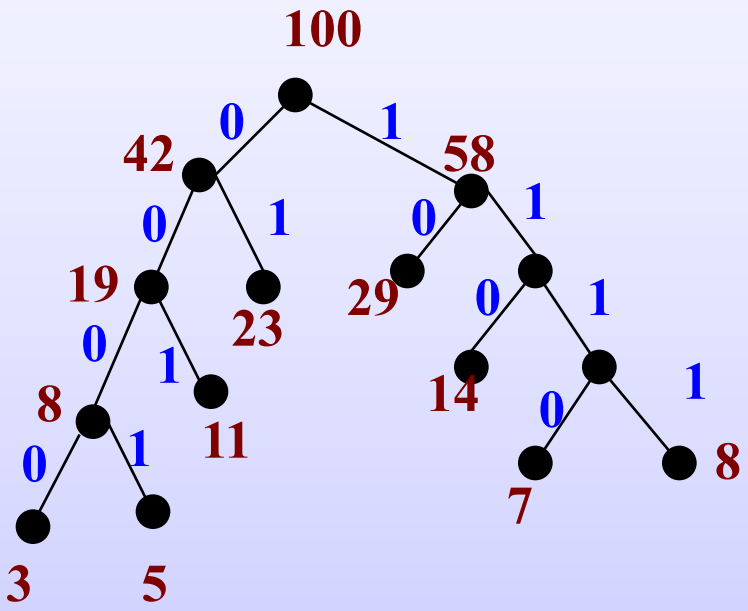


Huffman编码为：

A	5	0001
B	29	10
C	7	1110
D	8	1111
E	14	110
F	23	01
G	3	0000
H	11	001

4.Huffman编码存储结构

- 由于Huffman树中没有度为1的结点，则n个叶结点的Huffman树共有2n-1个结点。例如，4个叶结点的Huffman树，共有7个结点。因此可以用长度为2n-1的一维数组存放。
- 求Huffman编码：从根到叶的编码。因此要知道每个结点的父结点。

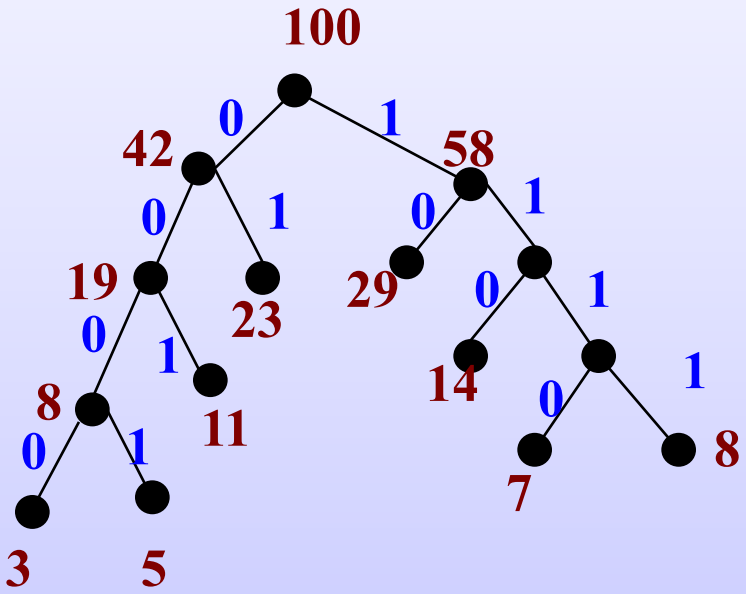


Huffman编码为：

A	5	0001
B	29	10
C	7	1110
D	8	1111
E	14	110
F	23	01
G	3	0000
H	11	001

5.Huffman编码的译码

- 从Huffman编码树上不难看出，代码全部在叶结点上，根据Huffman编码，就能求出相应的字符。该过程称为“译码”。
- 译码是根据从根到叶的Huffman编码求相应的字符。因此要知道每个结点的左右子结点。
- 例如，根据“1111”，就能求出对应的字符是“8”。



作业

- 在某系统的通信联络中可能出现10种字符（A-J），其频率分别为0.04、0.15、0.07、0.08、0.14、0.21、0.02、0.12、0.16、0.01。试建立其Huffman树并给出其Huffman编码。

总结

1. 树的定义
2. 树的基本概念
3. 二叉树
4. 特殊二叉树
5. 二叉树的遍历操作