

第六章 树

学习要点:

- 理解树的定义和与树相关的结点、度、路径等术语。
- 理解树是一个非线性层次数据结构。
- 掌握树的前序遍历、中序遍历和后序遍历方法。
- 了解树的父结点数组表示法。
- 了解树的儿子链表表示法。
- 了解树的左儿子右兄弟表示法。
- 理解二叉树和**ADT**二叉树的概念。
- 了解二叉树的顺序存储结构。
- 了解二叉树的结点度表示法。
- 掌握用指针实现二叉树的方法。
- 理解线索二叉树结构及其适用范围。

第六章 树

树是一类重要的非线性数据结构，是以分支关系定义的层次结构

6.1 树的定义

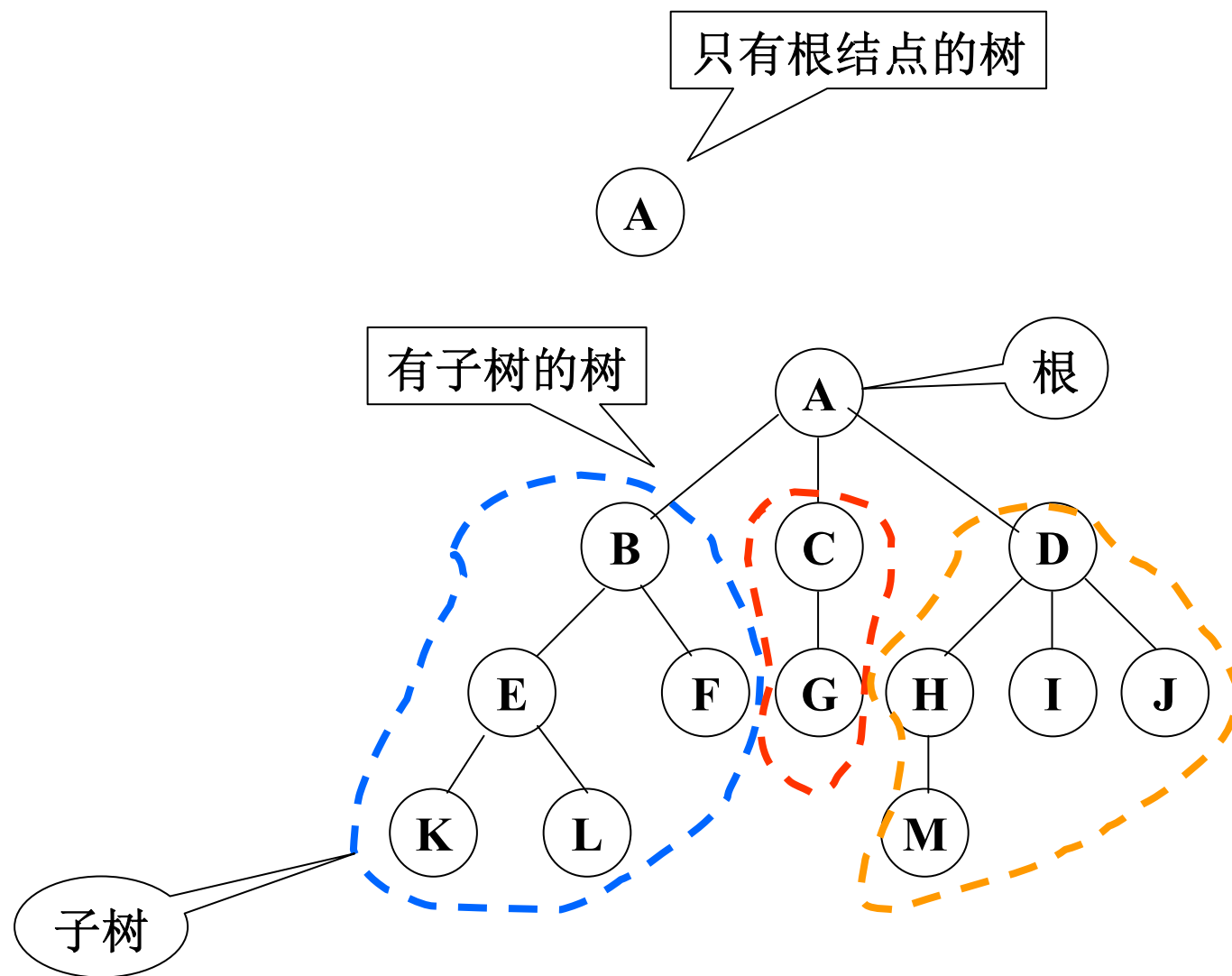
❁ 定义

❁ 递归定义

(1) 单个结点是一棵树，该结点就是树根。

(2) 设 T_1, T_2, \dots, T_k 都是树，它们的根结点分别为 n_1, n_2, \dots, n_k ，而 n 是另一个结点且以 n_1, n_2, \dots, n_k 为儿子，则 T_1, T_2, \dots, T_k 和 n 构成一棵新树。结点 n 就是新树的根。称 n_1, n_2, \dots, n_k 为一组兄弟结点。还称 T_1, T_2, \dots, T_k 为结点 n 的子树。

为了方便起见，空集合也看作是树，称为空树，并用 \wedge 来表示。空树中没有结点。



基本术语

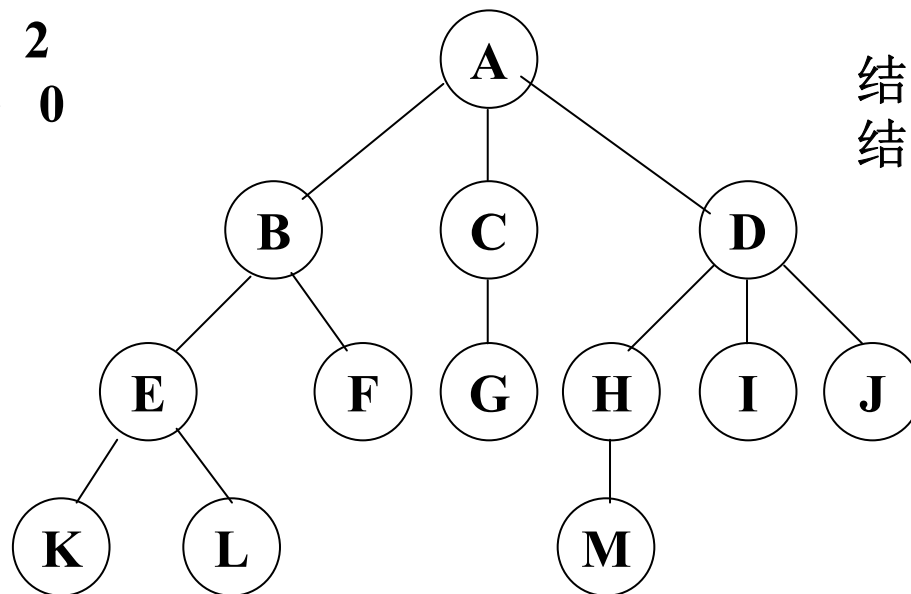
- 结点——表示树中的元素，包括数据项及若干指向其子树的分支
- 结点的度——结点的儿子结点个数
- 树的度——一棵树中最大的结点度数
- 叶结点——度为0的结点
- 分枝结点——度不为0的结点
- 路径——若存在树中的一个节点序列 k_1, k_2, \dots, k_j ，使得结点 k_i 是 k_{i+1} 的父结点($1 \leq i < j$)，则称该结点序列是树中从结点 k_1 到结点 k_j 的一条路径。
- 路径长度——路径所经过的边的数目。
- 祖先、子孙
- 结点的高度——从该结点到各叶结点的最长路径长度
- 树的高度——根结点的高度
- 结点的深度(或层数)——从树根到任一结点 n 有唯一的路径，称该路径的长度为结点 n 的深度(或层数)。从根结点算起，根为第0层，它的孩子为第1层.....
- 有序树——为树的每一组兄弟结点定义一个从左到右的次序
- 左儿子、右兄弟
- 森林—— $m(m \geq 0)$ 棵互不相交的树的集合

结点A的度: 3

结点B的度: 2

结点M的度: 0

树的度: 3



结点A的孩子: B, C, D

结点B的孩子: E, F

结点A的层次: 0

结点M的层次: 3

叶结点: K, L, F, G, M, I, J

分枝节点: A, B, C, D, E, H

结点I的双亲: D

结点L的双亲: E

结点B, C, D为兄弟

结点K, L为兄弟

结点A的高度: 3

结点D的高度: 2

树的高度: 3

结点A是结点F, G的祖先

结点B, C, ...是结点A的子孙

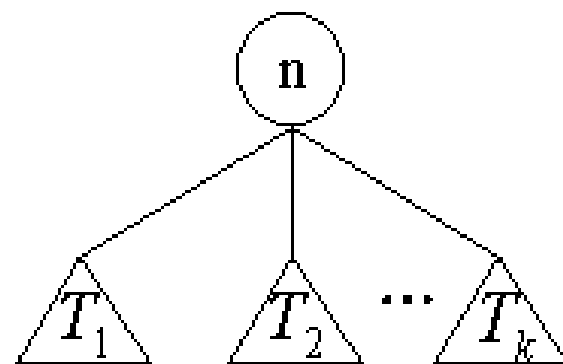
6.2 树的遍历

树的遍历

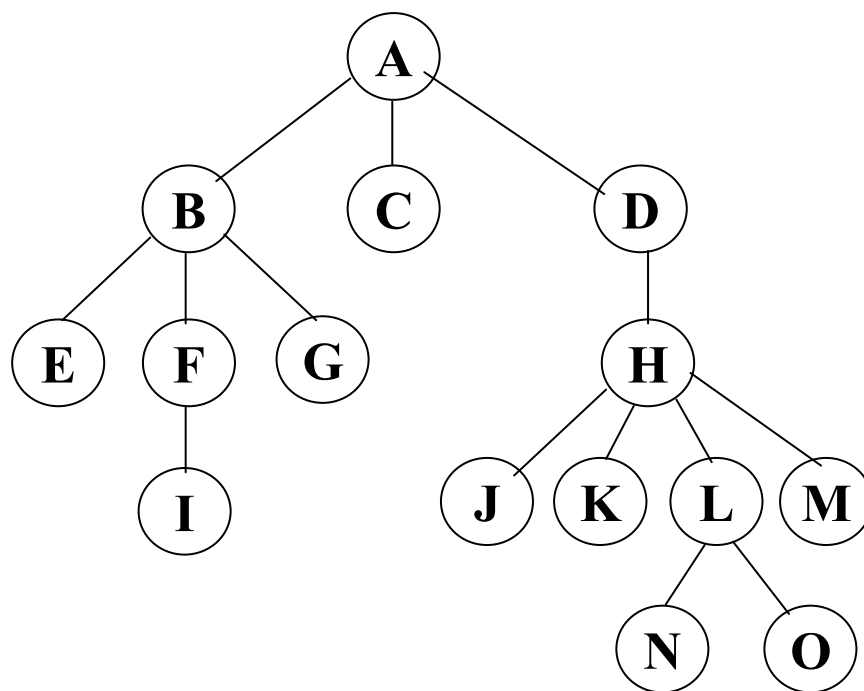
遍历——按一定规律走遍树的各个顶点，且使每一顶点仅被访问一次，即找一个完整而有规律的走法，以得到树中所有结点的一个线性排列。

树 T 的3种遍历方式的递归定义：(如图示)

- (1) 前序遍历——先访问树根 n ，然后依次前序遍历 T_1, T_2, \dots, T_k 。
- (2) 中序遍历——先中序遍历 T_1 ，然后访问树根 n ，接着依次对 T_2, T_3, \dots, T_k 进行中序遍历。
- (3) 后序遍历——先依次对 T_1, T_2, \dots, T_k 进行后序遍历，最后访问树根 n 。



树 T



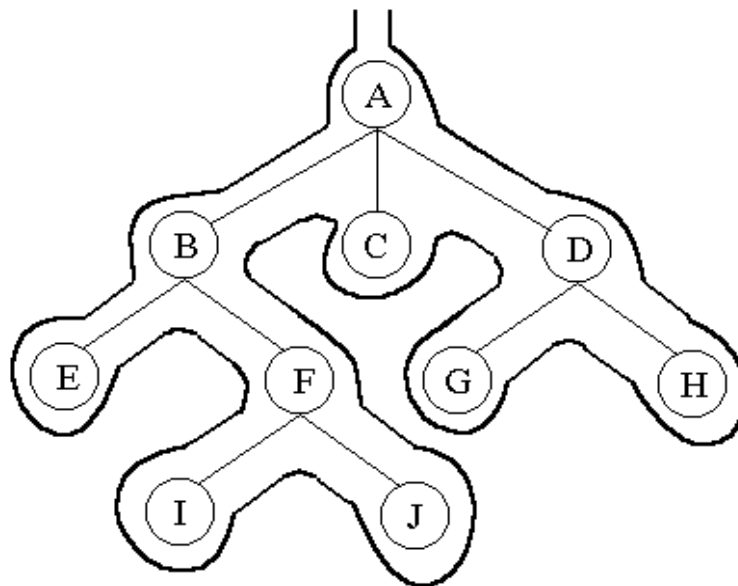
前序遍历: **A B E F I G C D H J K L N O M**

中序遍历: **E B I F G A C J H K N L O M D**

后序遍历: **E I F G B C J K N O L M H D A**

层次遍历: **A B C D E F G H I J K L M N O**

有序树T的3种遍历的非递归方式



按第一次经过的时间次序将各个结点列表:

A B E F I J C D G H

前序列表

按最后一次经过的时间次序将各个结点列表:

E I J F B C G H D A

后序列表

叶结点在第一次经过时列出, 而内部结点在第2次经过时列出:

E B I F J A C G D H

中序列表

有序树T的3种遍历能得到什么信息？

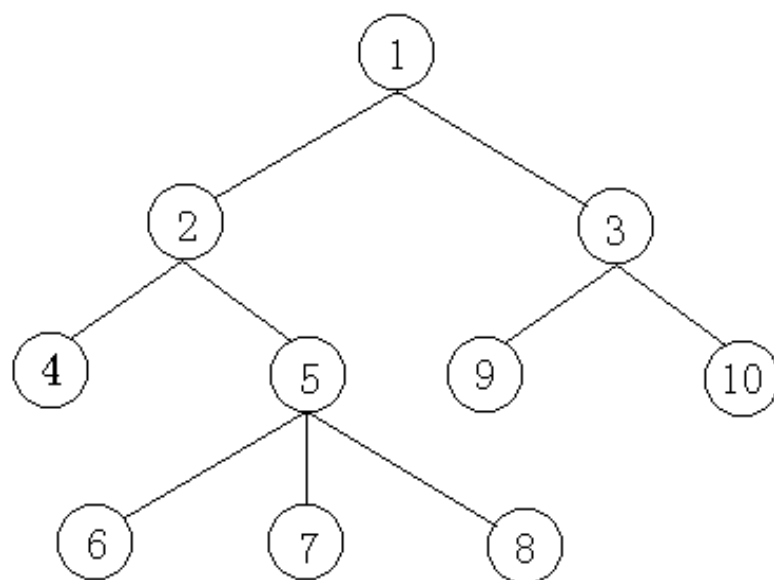
- (1) 在3种不同方式中，各树叶之间的相对次序是相同的，它们都按树叶之间从左到右的次序排列，其差别仅在于内部结点之间以及内部结点与树叶之间的次序有所不同。
- (2) 后序遍历有助于查询结点间的祖先和子孙关系，因为 $\text{postorder}(y) - \text{desc}(y) \leq \text{postorder}(x) \leq \text{postorder}(y)$ 。
其中y是T中的任一结点；x是y的子孙； $\text{desc}(y)$ 是T中y的真子孙数； $\text{postorder}(y)$ 是T中y的后序序号。
- (3) 前序遍历也有助于查询结点间的祖先和子孙关系，因为 $\text{preorder}(y) \leq \text{preorder}(x) \leq \text{preorder}(y) + \text{desc}(y)$ 。
其中y是T中的任一结点；x是y的子孙； $\text{desc}(y)$ 是T中y的真子孙数； $\text{preorder}(y)$ 是T中y的前序序号。

6.3 树的表示法

✦ 树的存储结构

✦ 父结点数组表示法

- (1) 树中的结点数字化为它们的编号 $1, 2, \dots, n$ 。
- (2) 用一个一维数组存储每个结点的父结点。即：
father[k]中是存放结点k的父结点的编号。
- (3) 由于树中每个结点的父结点是唯一的，所以父结点数组表示法可以唯一表示任何一棵树。
- (4) 实例：见下页



(a)

0	1	1	2	2	5	5	5	3	3
1	2	3	4	5	6	7	8	9	10

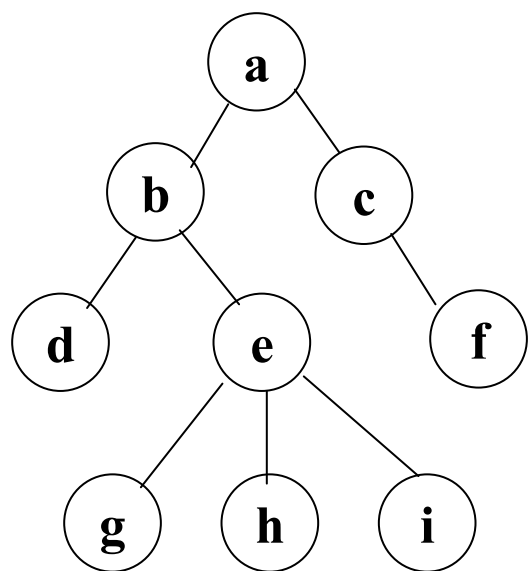
(b)

如何找孩子结点

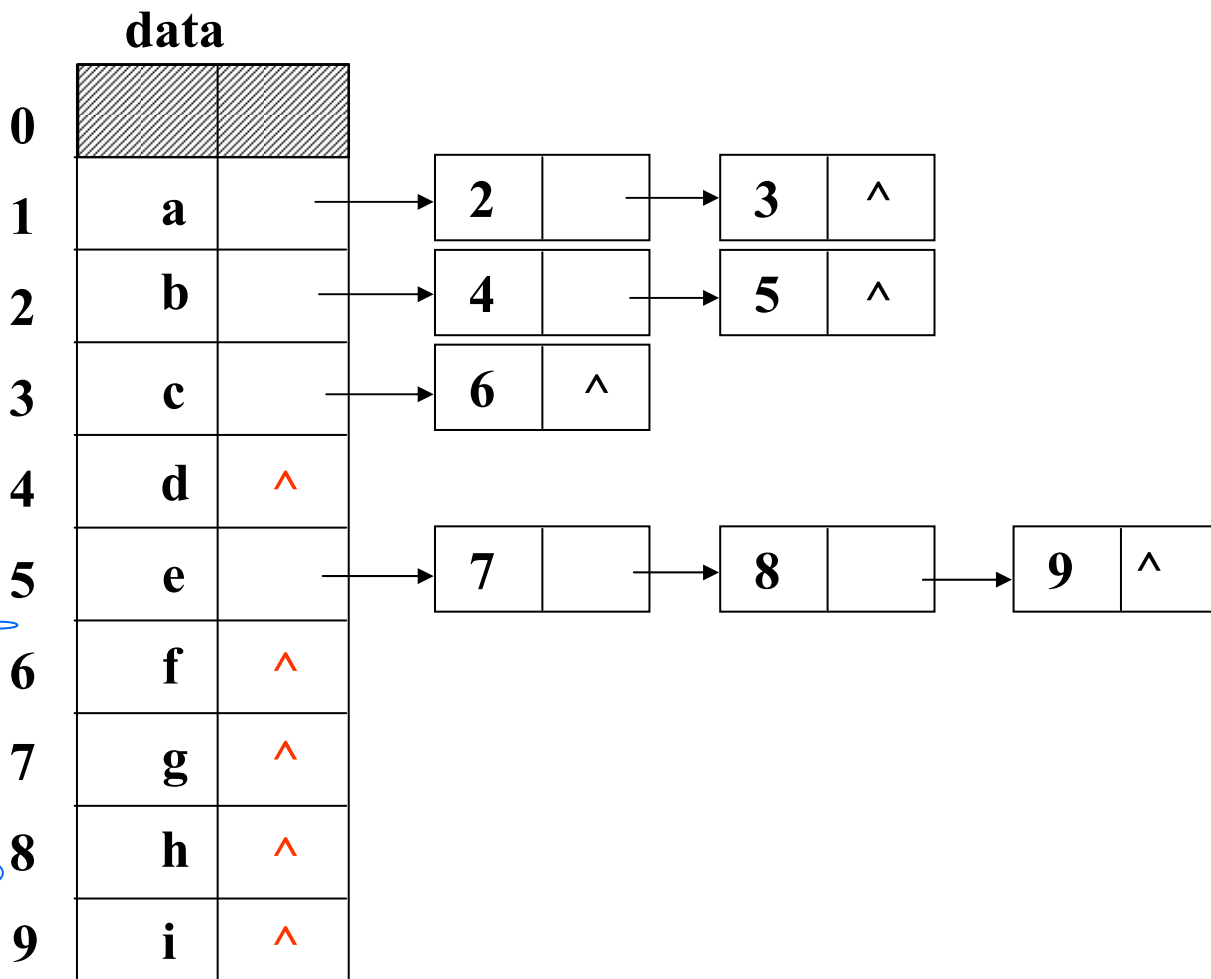
效率分析

- 寻找一个结点的父结点只需要 $O(1)$ 时间。
- 于涉及查询儿子结点和兄弟结点信息的运算，可能要遍历整个数组。

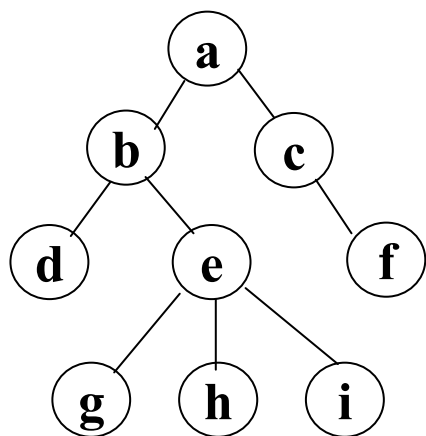
孩子表示法



如何找双亲结点



带双亲的孩子链表

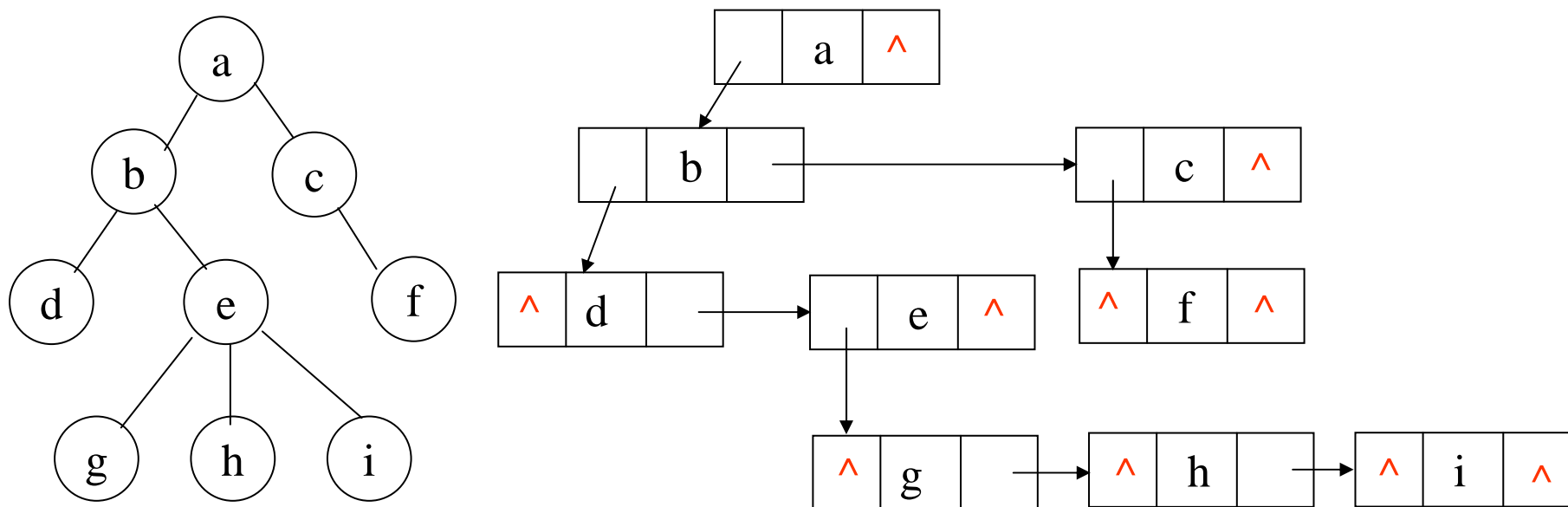


data parent			
1	a	0	→ 2
2	b	1	→ 4
3	c	1	→ 6
4	d	2	^
5	e	2	→ 7
6	f	3	^
7	g	5	^
8	h	5	^
9	i	5	^

2		→ 3	^
4		→ 5	^
6			
7		→ 8	
8		→ 9	^

左儿子右兄弟表示法

- 实现：用二叉链表作树的存储结构，链表中每个结点的两个指针域分别指向其最左儿子和右邻兄弟



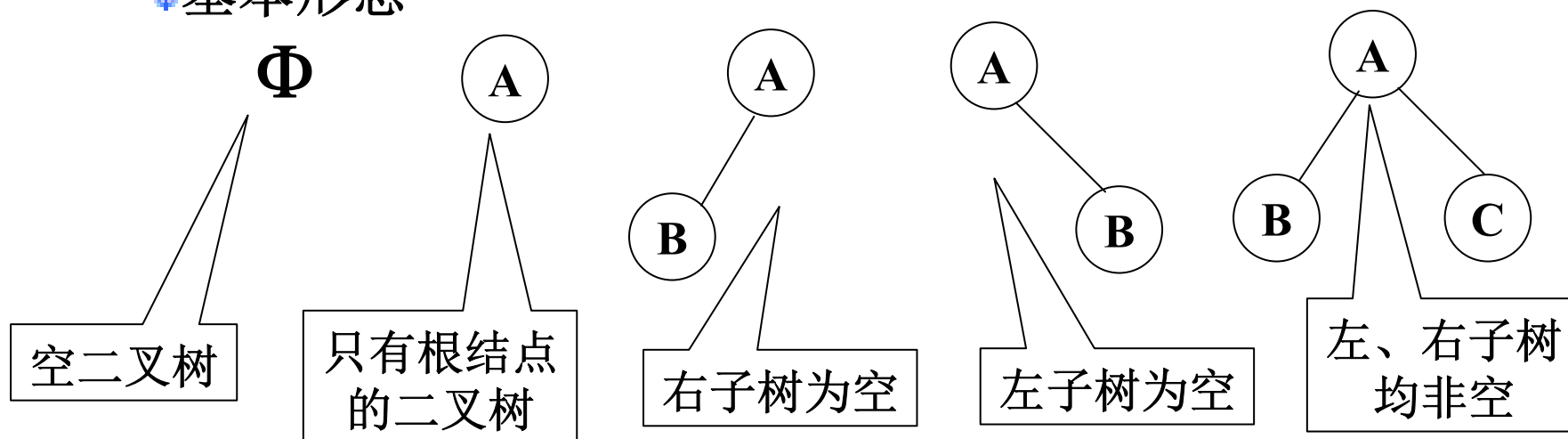
6.4 二叉树

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

✦特点

- ✦每个结点至多有二棵子树(即不存在度大于2的结点)
- ✦二叉树的子树有左、右之分，且其次序不能任意颠倒

✦基本形态



✦ 二叉树性质

- ✦ 高度为 $h \geq 0$ 的二叉树至少有 $h+1$ 个结点。
- ✦ 高度为 $h \geq 0$ 的二叉树至多有 $2^{h+1}-1$ 个结点。
- ✦ 约定空二叉树的高度为 -1 。
- ✦ 含有 $n \geq 1$ 个结点的二叉树的高度至多为 $n-1$ 。
- ✦ 含有 $n \geq 1$ 个结点的二叉树的高度至少为 $\lfloor \log n \rfloor$ ，因此为 $\Omega(\lfloor \log n \rfloor)$ 。

✦ 具有***n***个结点的不同形态的二叉数的数目即所谓的***n***阶Catalan数:

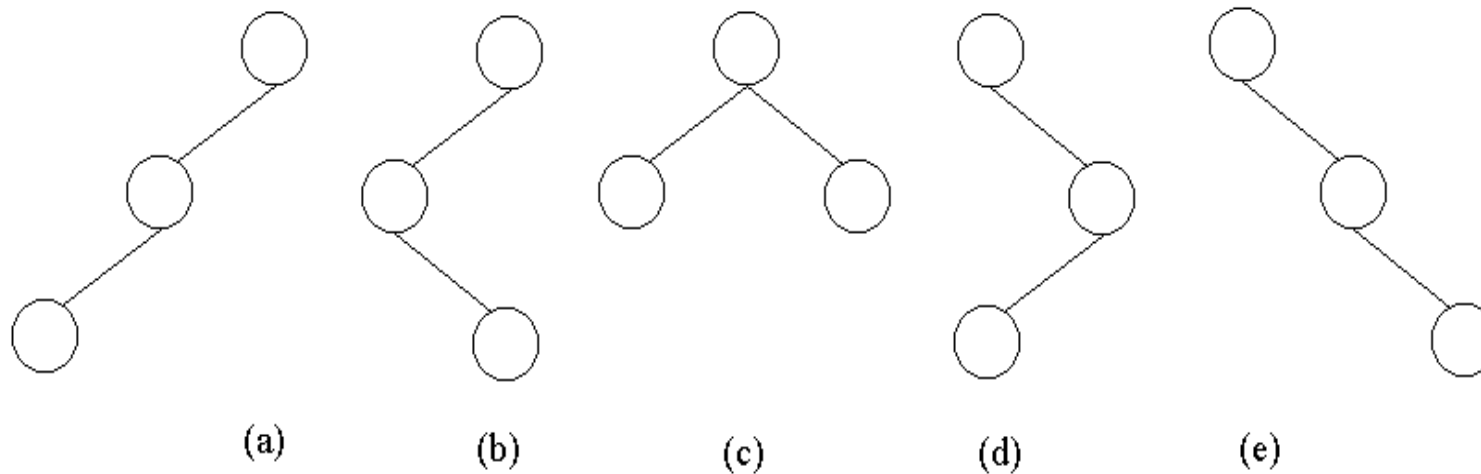
$$B_n = \frac{1}{n+1} \binom{2n}{n}$$

因为一棵具有***n***个结点的二叉树可以看成是由一个根结点、一棵具有***i***个结点的左子树和一棵具有***n-i-1***个结点的右子树所组成, $i=0, 1, 2, 3, \dots, n-1$, 所以 B_n 可以递归地表达为

$$B_n = \begin{cases} 1 & n = 0 \\ \sum_{i=0}^{n-1} B_i B_{n-i-1} & n \geq 1 \end{cases}$$

解此递归式即是上述的显式表达式。

例如：具有3个结点的不同形态的二叉数的数目 $B_3=5$



✦重要性质：对任何一棵二叉树**T**，如果其终端结点数为 **n_0** ，
度为**2**的结点数为 **n_2** ，

则： **$n_0=n_2+1$**

证明： **n_1** 为二叉树**T**中度为**1**的结点数

因为：二叉树中所有结点的度均小于或等于**2**

所以：其结点总数 **$n=n_0+n_1+n_2$**

又二叉树中，除根结点外，其余结点都只有一个分支进入

设**B**为分支总数，则 **$n=B+1$**

又：分支由度为**1**和度为**2**的结点射出

$\therefore B=n_1+2n_2$

于是， **$n=B+1=n_1+2n_2+1=n_0+n_1+n_2$**

$\therefore n_0=n_2+1$

✦ 几种特殊形式的二叉树

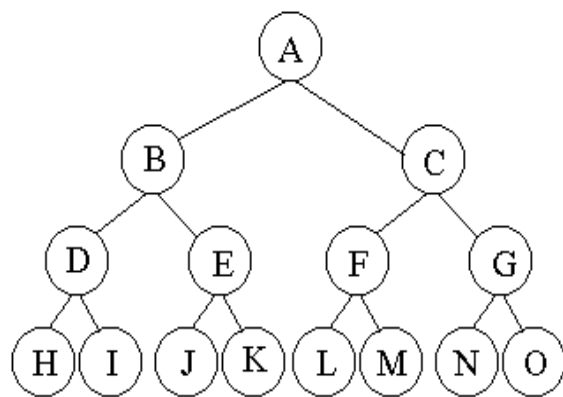
✦ 满二叉树

✦ 定义：一棵高度为 k 且有 $2^{k+1} - 1$ 个结点的二叉树称为 ~

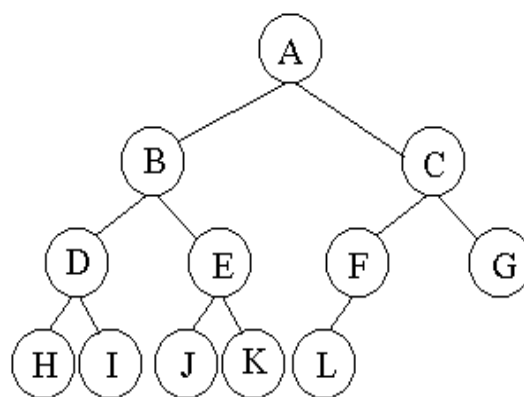
✦ 特点：每一层上的结点数都是最大结点数

✦ 近似满二叉树（完全二叉树）

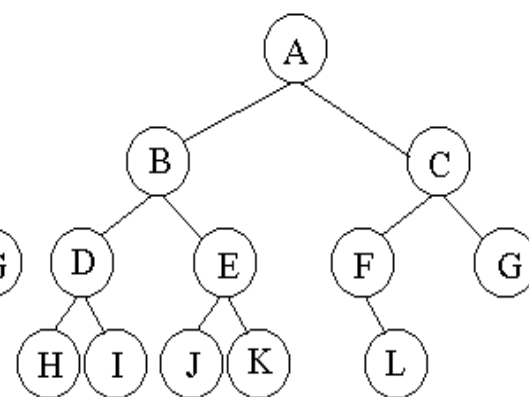
✦ 定义：若一棵二叉树最多只有最下面的**2**层上结点的度数可以小于**2**，并且最下面一层上的节点都集中在该层的最左边，则这种二叉树称为近似满二叉树



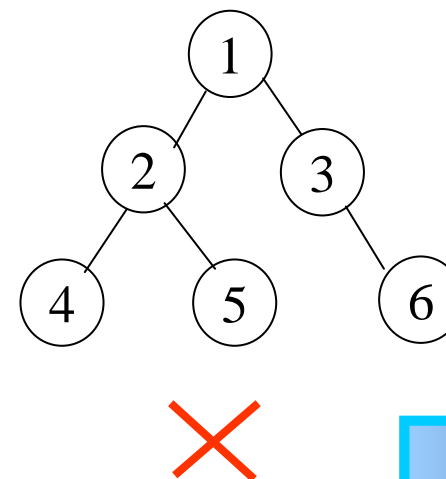
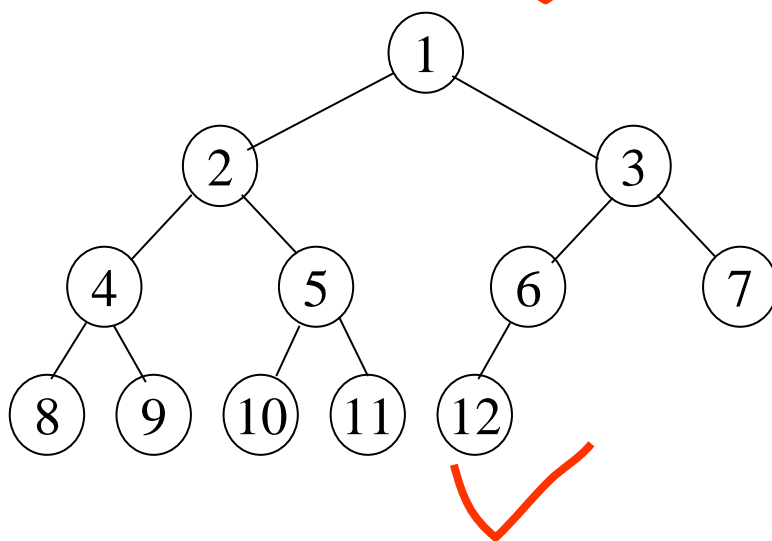
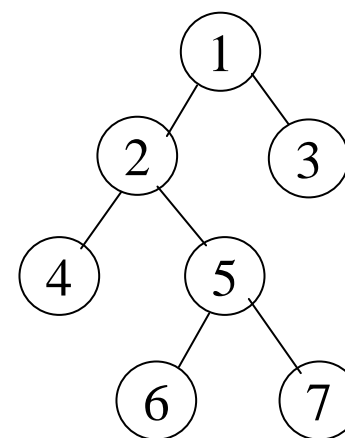
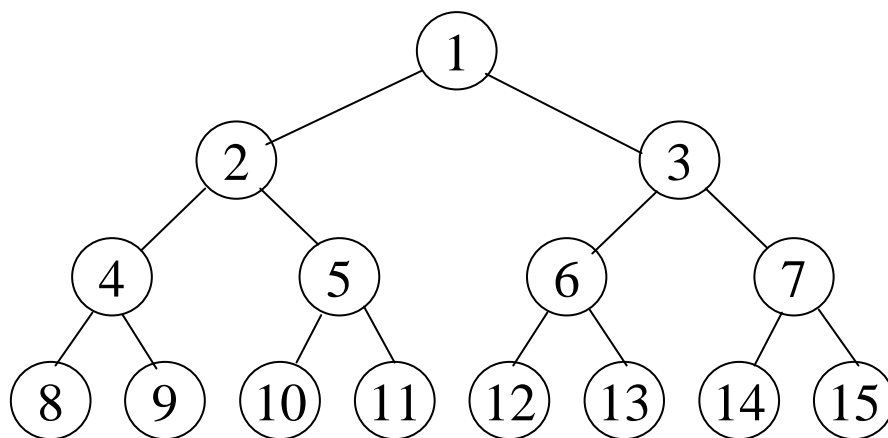
(a) 满二叉树



(b) 近似满二叉树



(c) 非近似满二叉树



✦近似满二叉树性质：

如果对一棵有 n 个结点的完全二叉树的结点按层序编号，则对任一结点 $i(1 \leq i \leq n)$ ，有：

- (1) 如果 $i=1$ ，则结点 i 是二叉树的根，无双亲；如果 $i>1$ ，则其双亲是 $\lfloor i/2 \rfloor$
- (2) 如果 $2i>n$ ，则结点 i 无左孩子；如果 $2i \leq n$ ，则其左孩子是 $2i$
- (3) 如果 $2i+1>n$ ，则结点 i 无右孩子；如果 $2i+1 \leq n$ ，则其右孩子是 $2i+1$

6.5 ADT二叉树

6.5.1 ADT二叉树用来存放有二叉树结构的数据集

ADT二叉树支持的主要基本运算：

(1)BinaryInit() 创建一棵空二叉树。

(2)BinaryEmpty() 判断给定的二叉树是否为空。

(3)Root(T) 返回给定二叉树的根结点标号。

(4)MakeTree(x,T,L,R)以 x 为根结点元素,以 L 和 R 分别为左、右子树,构建一棵新的二叉树 T 。

(5)BreakTree(T,L,R)函数**MakeTree**的逆运算,即将二叉树拆分为根结点元素,左子树 L 和右子树 R 等3部分。

6.5 ADT二叉树

6.5.1 ADT二叉树用来存放有二叉树结构的数据集

ADT二叉树支持的主要基本运算(续)

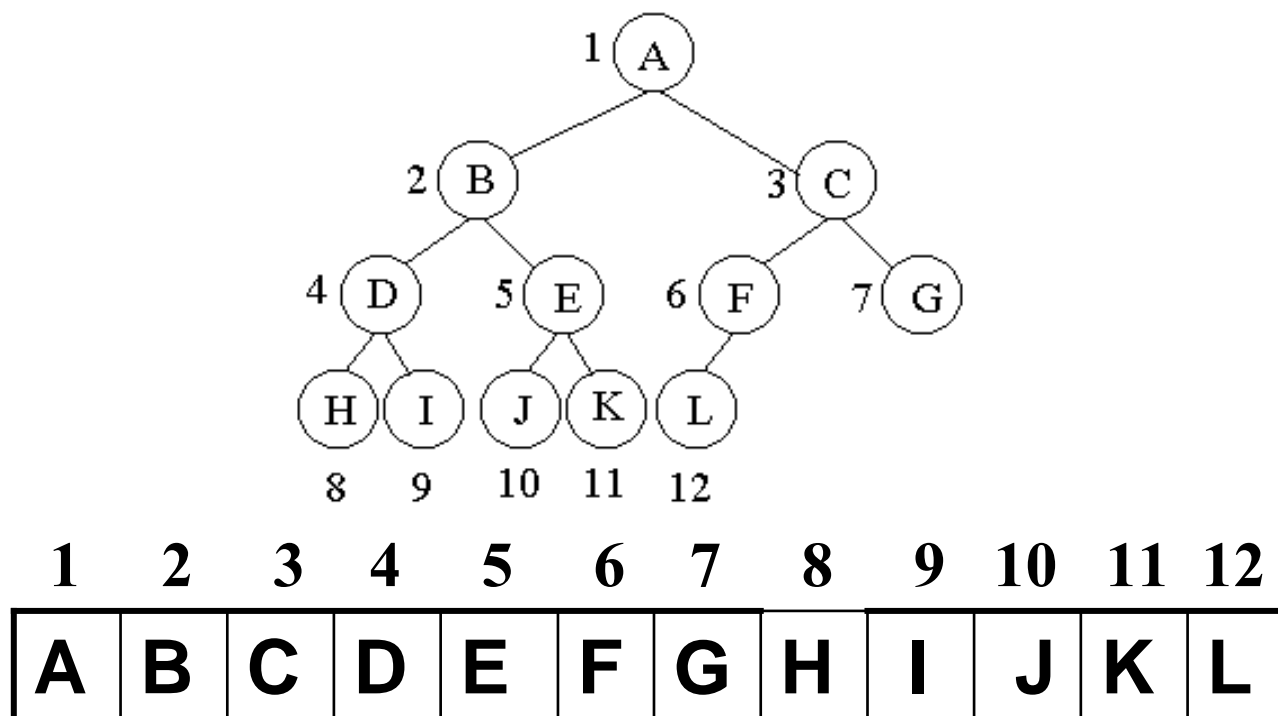
- (6) $\text{PreOrder}(\text{visit}, t)$ 前序遍历给定的二叉树。
- (7) $\text{InOrder}(\text{visit}, t)$ 中序遍历给定的二叉树。
- (8) $\text{PostOrder}(\text{visit}, t)$ 后序遍历给定的二叉树。
- (9) $\text{PreOut}(T)$ 给定的二叉树的前序列表。
- (10) $\text{InOut}(T)$ 给定的二叉树的中序列表。
- (11) $\text{PostOut}(T)$ 给定的二叉树的后序列表。
- (12) $\text{Delete}(t)$ 删除给定的二叉树。
- (13) $\text{Height}(t)$ 求给定的二叉树的高度。
- (14) $\text{Size}(t)$ 求给定的二叉树的结点数。

6.5 ADT二叉树

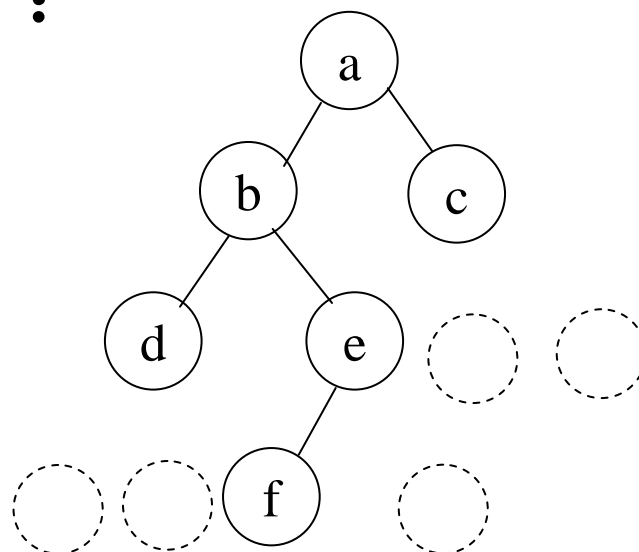
6.5.2 ADT二叉树的实现

6.5.2.1 用顺序存储结构实现(一种无边表示)

- ✦ 适用的对象：近似满二叉树
- ✦ 基本想法：若将所有的结点按层自上而下每层自左至右，从1开始编号。



✦不适用的例子:



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

6.5 ADT二叉树

6.5.2 ADT二叉树的实现

6.5.2.2 二叉树的结点度表示法(另一种无边表示)

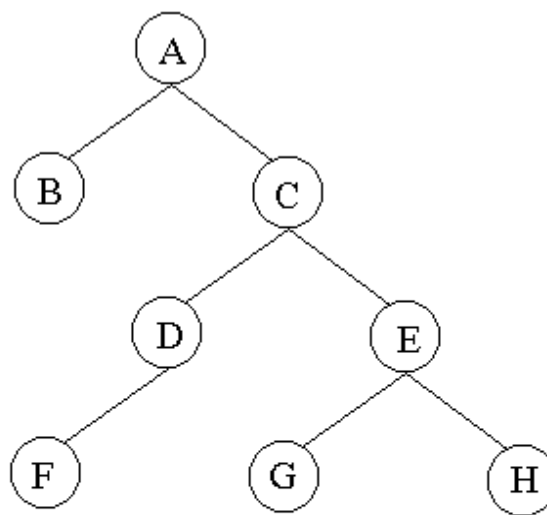
- 基本想法：若将所有的结点按后序列表从1开始编号，并在每个结点中附加一个0到3之间的整数，以表示该结点的分叉特征。该整数为0时，表示相应的结点无儿子；为1时，表示相应结点只有左儿子；为2时，表示相应结点只有右儿子；为3时，表示相应结点既有左儿子又有右儿子。那么，结点之间有如下隐含关系：若结点 i 有右儿子，则 $i-1$ 一定是其右儿子结点。另一方面，结点 i 的左儿子结点必在 i 之前，而其父结点必在 i 之后。照此，二叉树可以用记录数组来表示

6.5 ADT二叉树

6.5.2 ADT二叉树的实现

6.5.2.2 二叉树的结点度表示法(另一种无边表示)

⊕ 例



(a)

(B, 0)	(F, 0)	(D, 1)	(G, 0)	(H, 0)	(E, 3)	(C, 3)	(A, 3)
--------	--------	--------	--------	--------	--------	--------	--------

(b)

6.5 ADT二叉树

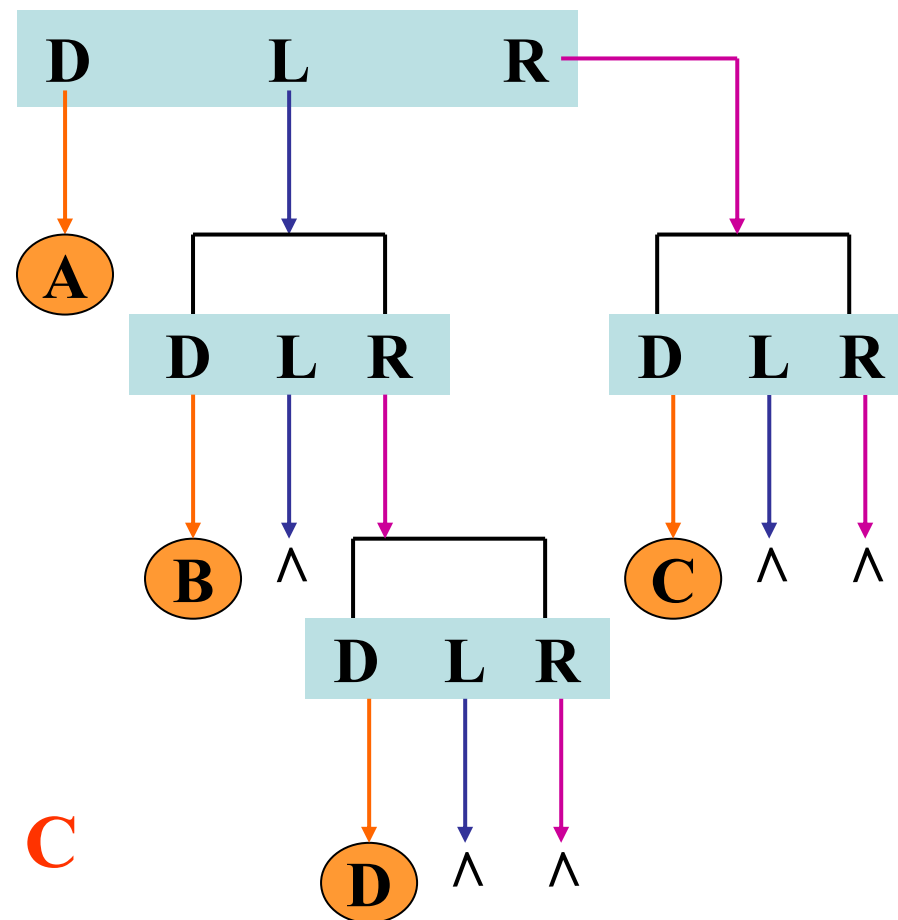
6.5.2 ADT二叉树的实现

6.5.2.3 二叉树的指针实现——带两个指针的结点的实现

P116-120

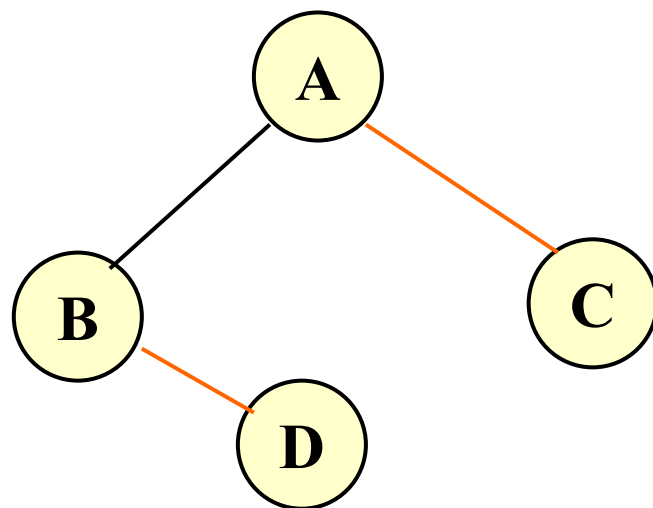
```

graph TD
    A((A)) --- B((B))
    A --- C((C))
    B --- D((D))
    C --- D
  
```

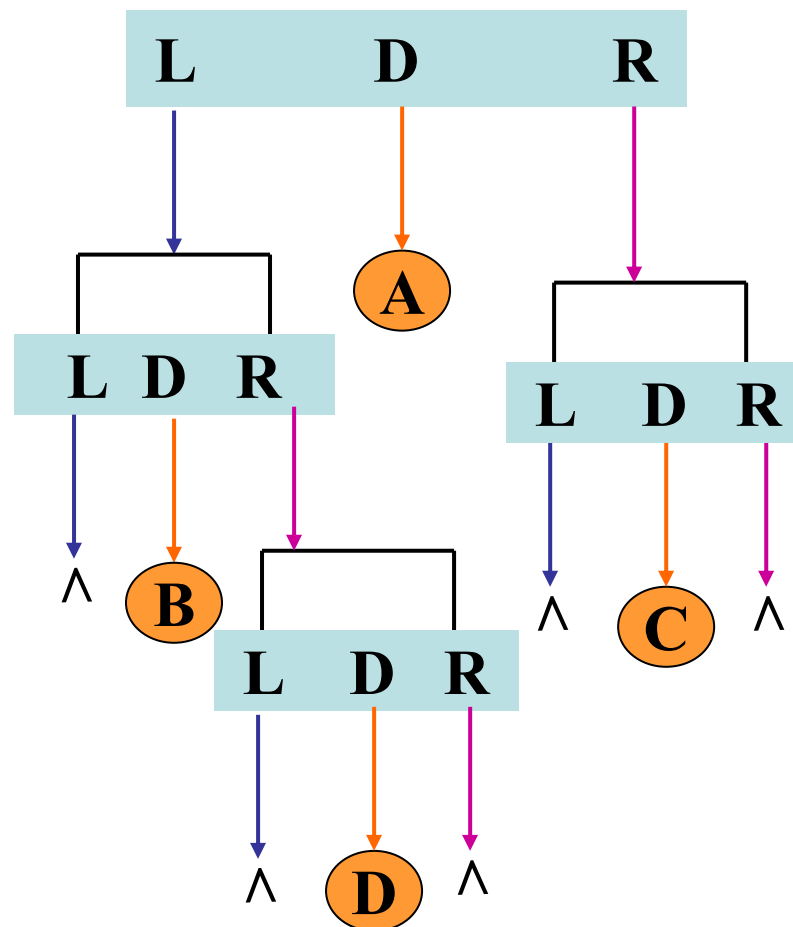


先序遍历序列: A B D C

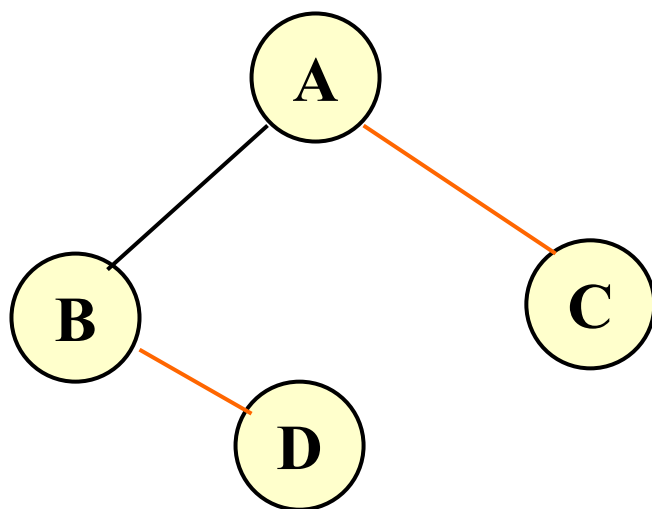
中序遍历:



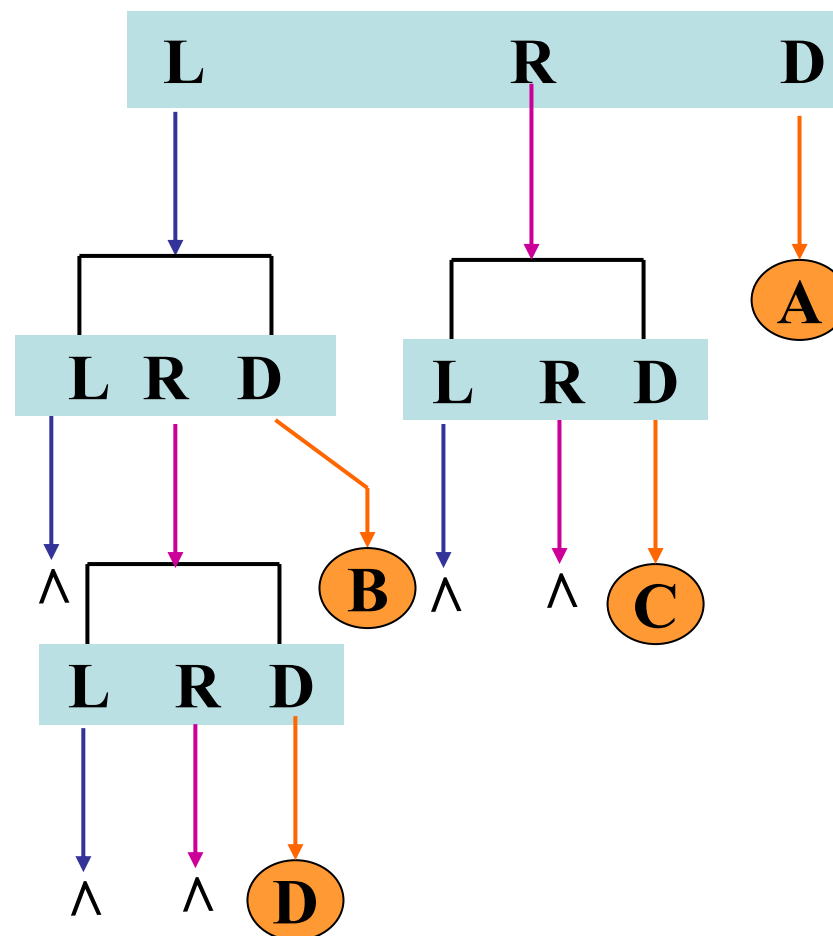
中序遍历序列: B D A C



后序遍历:



后序遍历序列: **D B C A**



6.5.1 二叉树的应用

✦ 哈夫曼树(Huffman)——带权路径长度最短的树

✦ 定义

- ✦ 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间的路径
- ✦ 路径长度：路径上的分支数
- ✦ 树的路径长度：从树根到每一个结点的路径长度之和
- ✦ 树的带权路径长度：树中所有带权结点的路径长度之和

记作：
$$wpl = \sum_{k=1}^n w_k l_k$$

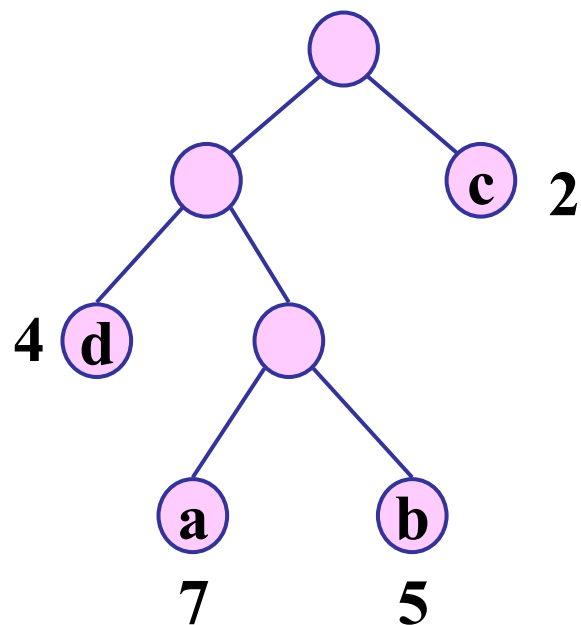
其中： w_k — 权值

l_k — 结点到根的路径长度

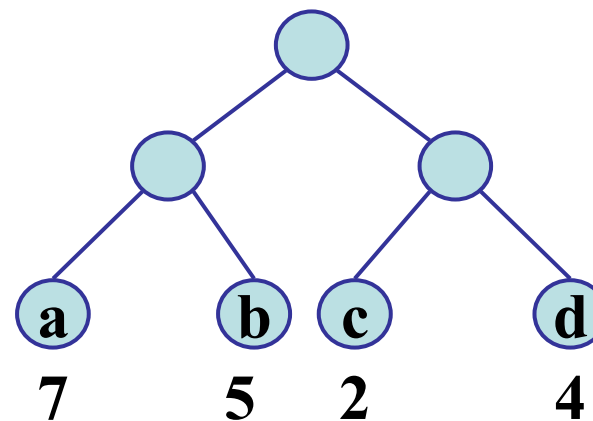
- ✦ **Huffman树**——设有n个权值{w1,w2,.....wn}，构造一棵有n个叶子结点的二叉树，每个叶子的权值为wi,则wpl最小的二叉树叫**Huffman树**

例 有4个结点，权值分别为7，5，2，4，构造有4个叶子结点的二叉树

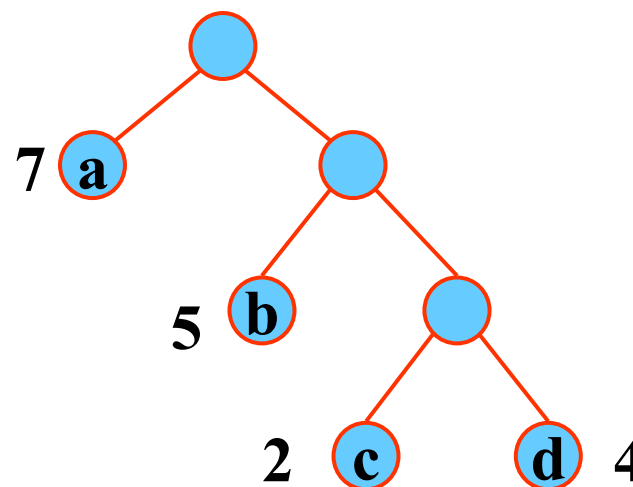
$$WPL = \sum_{k=1}^n W_K L_K$$



$$WPL = 7*3 + 5*3 + 2*1 + 4*2 = 46$$



$$WPL = 7*2 + 5*2 + 2*2 + 4*2 = 36$$



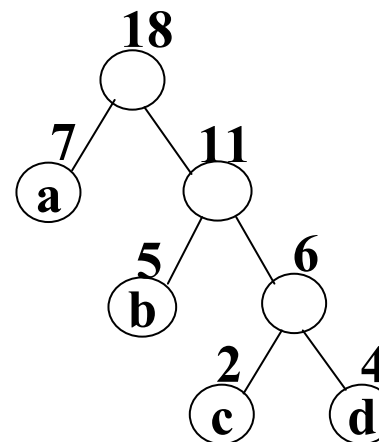
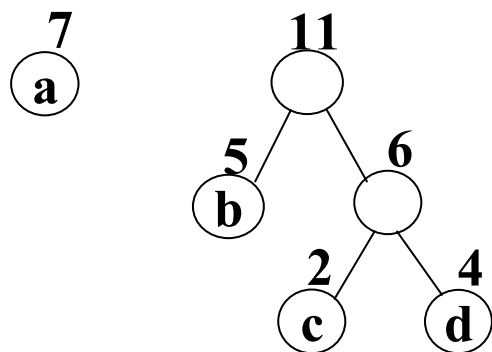
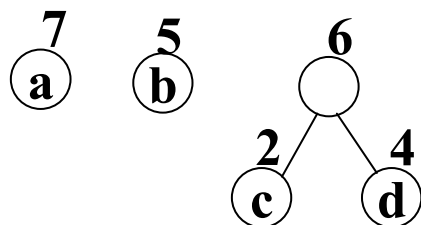
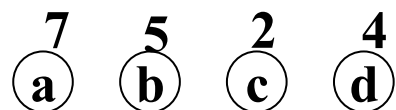
$$WPL = 7*1 + 5*2 + 2*3 + 4*3 = 35$$

✦构造Huffman树的方法——Huffman算法

✦构造Huffman树步骤

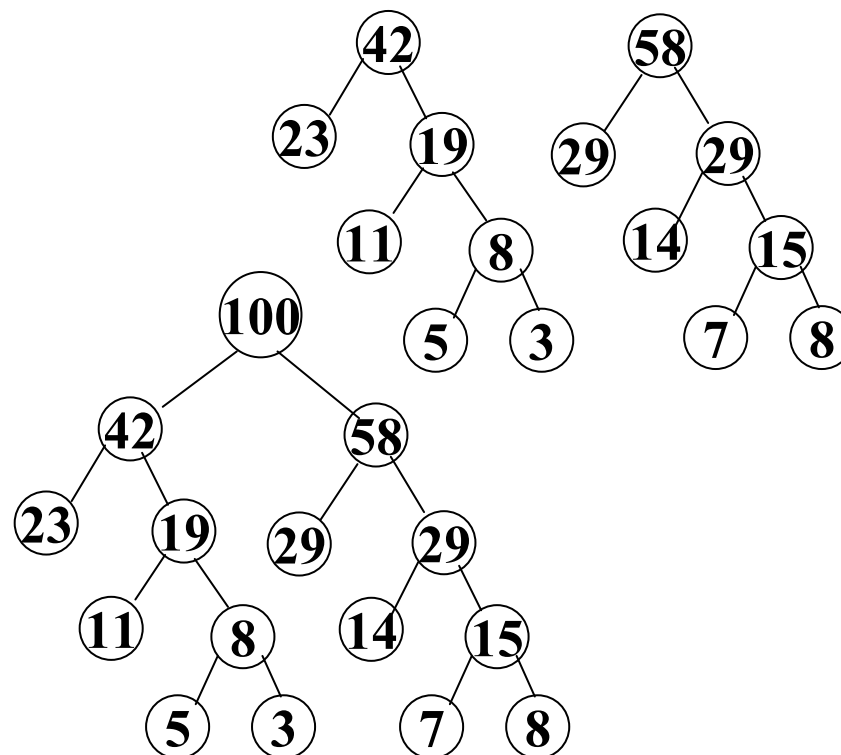
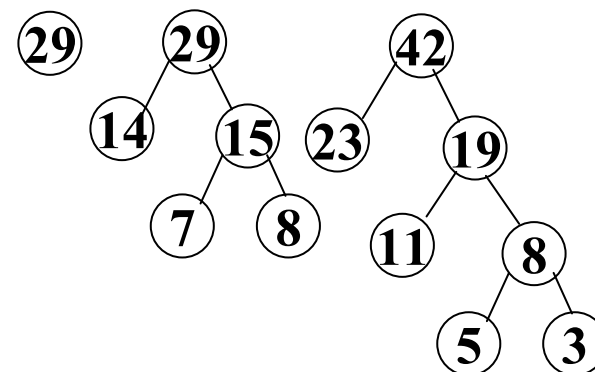
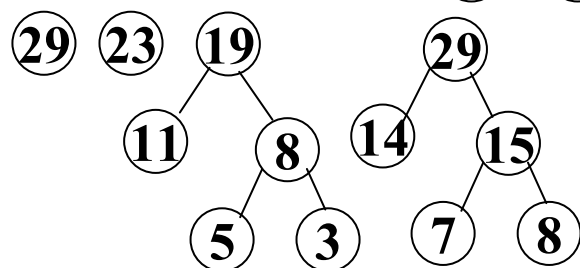
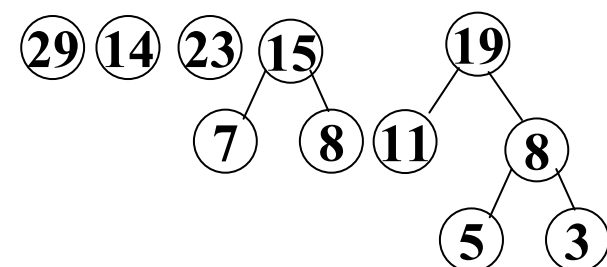
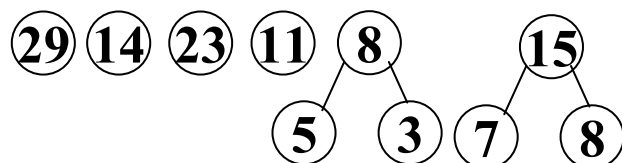
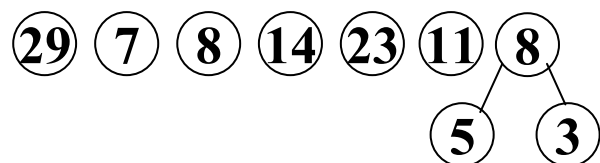
- ✦根据给定的 n 个权值 $\{w_1, w_2, \dots, w_n\}$ ，构造 n 棵只有根结点的二叉树，令起权值为 w_j
- ✦在森林中选取两棵根结点权值最小的树作左右子树，构造一棵新的二叉树，置新二叉树根结点权值为其左右子树根结点权值之和
- ✦在森林中删除这两棵树，同时将新得到的二叉树加入森林中
- ✦重复上述两步，直到只含一棵树为止，这棵树即哈夫曼树

例



例 $w=\{5, 29, 7, 8, 14, 23, 3, 11\}$

⑤ ②⑨ ⑦ ⑧ ⑭ ⑳ ③ ⑪



字符: a e i s t sp nl
权值: 10 15 12 3 4 13 1

压缩与哈夫曼树

[重新演示](#)

✦ Huffman算法实现



Ch5_8.txt

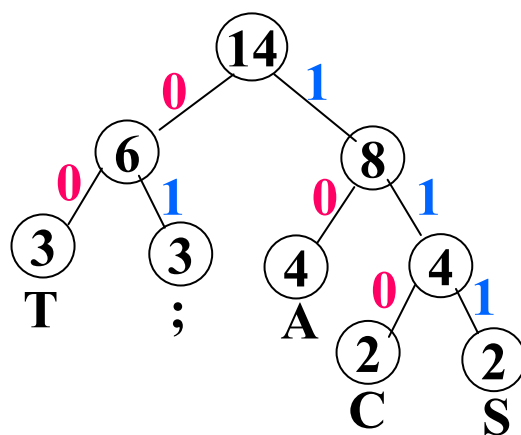
- ✦ 一棵有 n 个叶子结点的Huffman树有 $2n-1$ 个结点
- ✦ 采用顺序存储结构——一维结构数组
- ✦ 结点类型定义

```
typedef struct  
{ int data;  
  int pa,lc,rc;  
}JD;
```

✦ Huffman编码：数据通信用的二进制编码

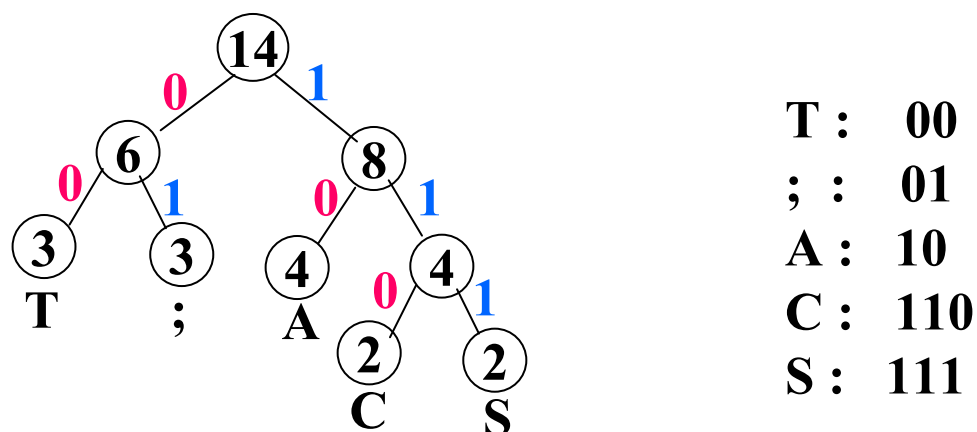
- ✦ 思想：根据字符出现频率编码，使电文总长最短
- ✦ 编码：根据字符出现频率构造Huffman树，然后将树中结点引向其左孩子的分支标“0”，引向其右孩子的分支标“1”；每个字符的编码即为从根到每个叶子的路径上得到的0、1序列

例 要传输的字符集 $D=\{C,A,S,T,;\}$
字符出现频率 $w=\{2,4,2,3,3\}$



T : 00
; : 01
A : 10
C : 110
S : 111

- 译码：从**Huffman**树根开始，从待译码电文中逐位取码。若编码是“0”，则向左走；若编码是“1”，则向右走，一旦到达叶子结点，则译出一个字符；再重新从根出发，直到电文结束



例 电文是{CAS;CAT;SAT;AT}
其编码 “11010111011101000011111000011000”
电文为“1101000”
译文只能是“CAT”

6.6 线索二叉树

6.6.1 引入线索二叉树的动因

用指针实现二叉树时，每个结点只有指向其左、右儿子结点的指针，所以从任一结点出发直接只能找到该结点的左、右儿子。在一般情况下无法直接找到该结点在某种遍历序下的前驱和后继结点。若在每个结点中增加指向其前驱和后继结点的指针，虽可提高遍历的效率却降低了存储效率。注意到用指针实现二叉树时， n 个结点二叉树中有 $n+1$ 个空指针。若利用这些空指针存放指向结点在某种遍历次序下的前驱或后继的指针，那么，可以期望提高遍历的效率。

6.6 线索二叉树

6.6.2 有关概念和术语

- ✦ “线索”：所引入的非空指针称为“线索”。
- ✦ 线索二叉树：加上了线索的二叉树称为线索二叉树。
- ✦ 线索标志位：为了区分一个结点的指针是指向其儿子结点的指针，还是指向其前驱或后继结点的线索，在每个结点中增加的2个位—LeftThread、RightThread分别称为左、右线索标志位。
- ✦ 线索化：对一棵非线索二叉树以某种次序遍历使其变为一棵线索二叉树的过程称为二叉树的线索化。

6.6 线索二叉树

6.6.3 线索二叉树结点类型定义

P120

6.6 线索二叉树

6.6.4 二叉树线索化：

由于线索化的实质是将二叉树中的空指针改为指向其前驱结点或后继结点的线索(并做上线索标志)，而一个结点的前驱或后继结点只有遍历才能知道，因此线索化的过程是在对二叉树遍历的过程中修改空指针的过程。

为了记下遍历过程中访问结点的先后次序，可引入指针 p 指引遍历，而引入指针 pre 跟踪 p 的前驱。首先将 pre 和 p 初始化为遍历的第一结点。然后让 p 往遍历的方向走找 pre 的后继。一旦找到，则它们互为前驱和后继，建立相应线索。接着将 p 赋给 pre ，重复下去直到遍历结束。

6.6 线索二叉树

6.6.4 二叉树的中序线索化

增加一个头结点，其LeftChild指针指向二叉树的根结点，其RightChild指针指向中序遍历的最后一个结点。而最后一个结点的RightChild指针指向头结点。

这样一来，就好象为二叉树建立了一个双向线索链表，既可从 中序遍历的第一个结点起进行中序的遍历；也可从 中序遍历的最后一个结点起进行逆中序的遍历。

6.6 线索二叉树

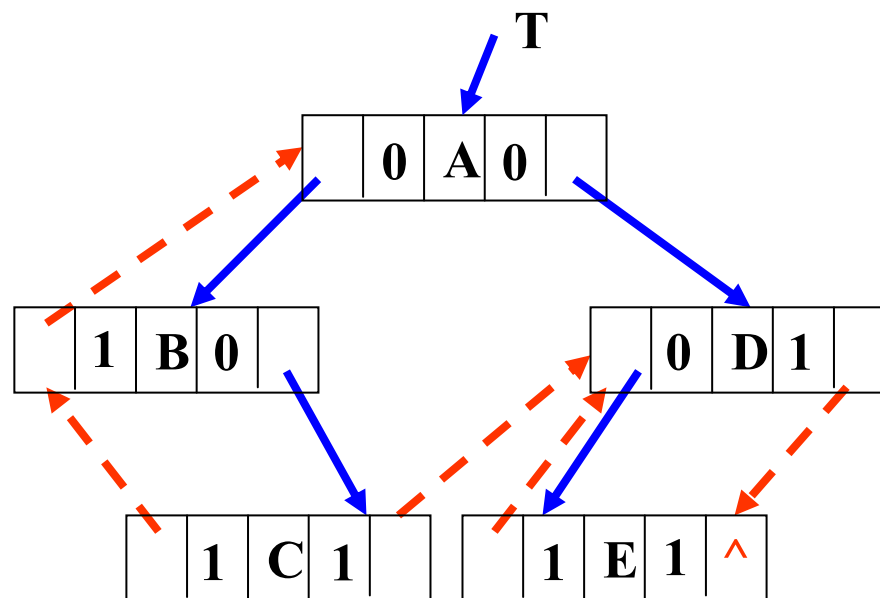
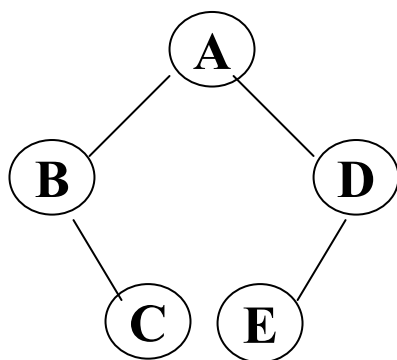
6.6.5 线索二叉树与非线索二叉树比较

✚ 优点:

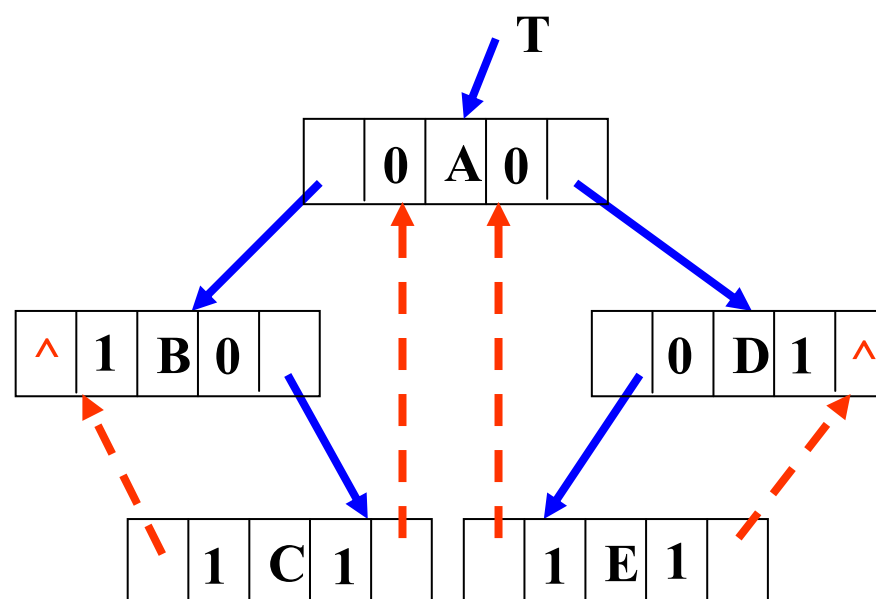
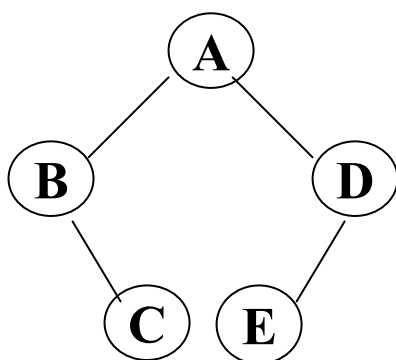
对于找前驱和后继结点2种运算而言，线索二叉树优于非线索二叉树。

✚ 缺点:

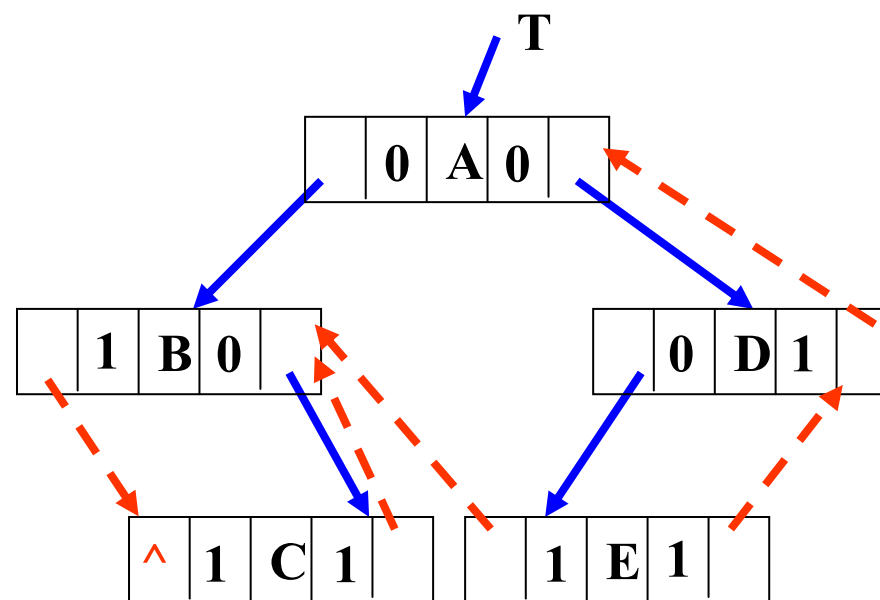
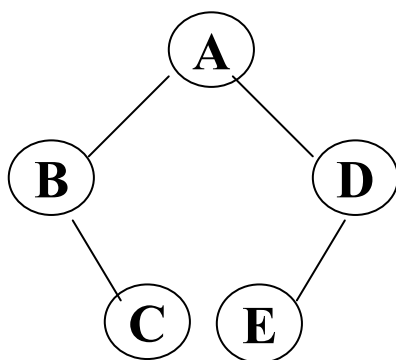
在进行结点插入和删除运算时，线索二叉树比非线索二叉树的时间开销大。原因在于在线索二叉树中进行结点插入和删除时，除了修改相应指针外，还要修改相应的线索。



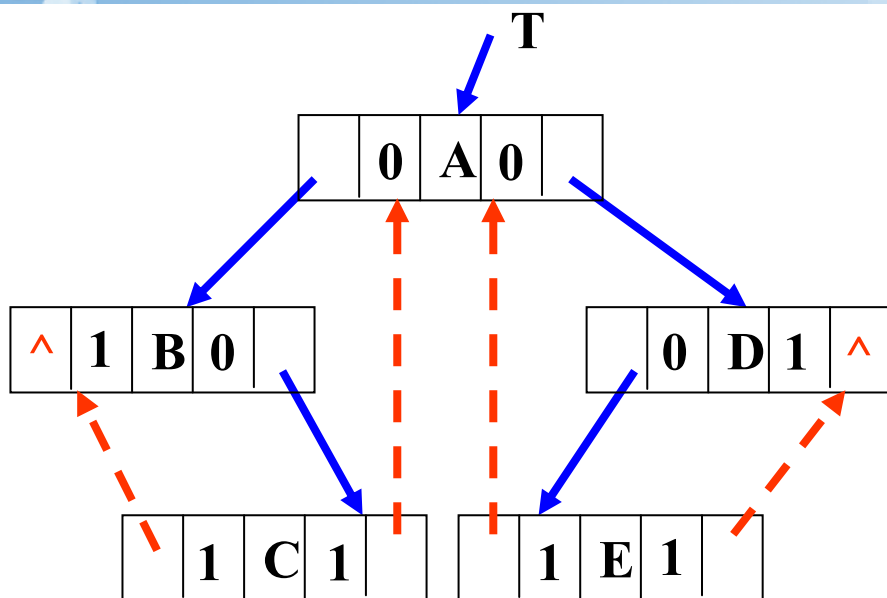
先序序列: ABCDE
先序线索二叉树



中序序列: **BCAED**
中序线索二叉树



后序序列: **CBEDA**
后序线索二叉树



中序序列: **BCAED**

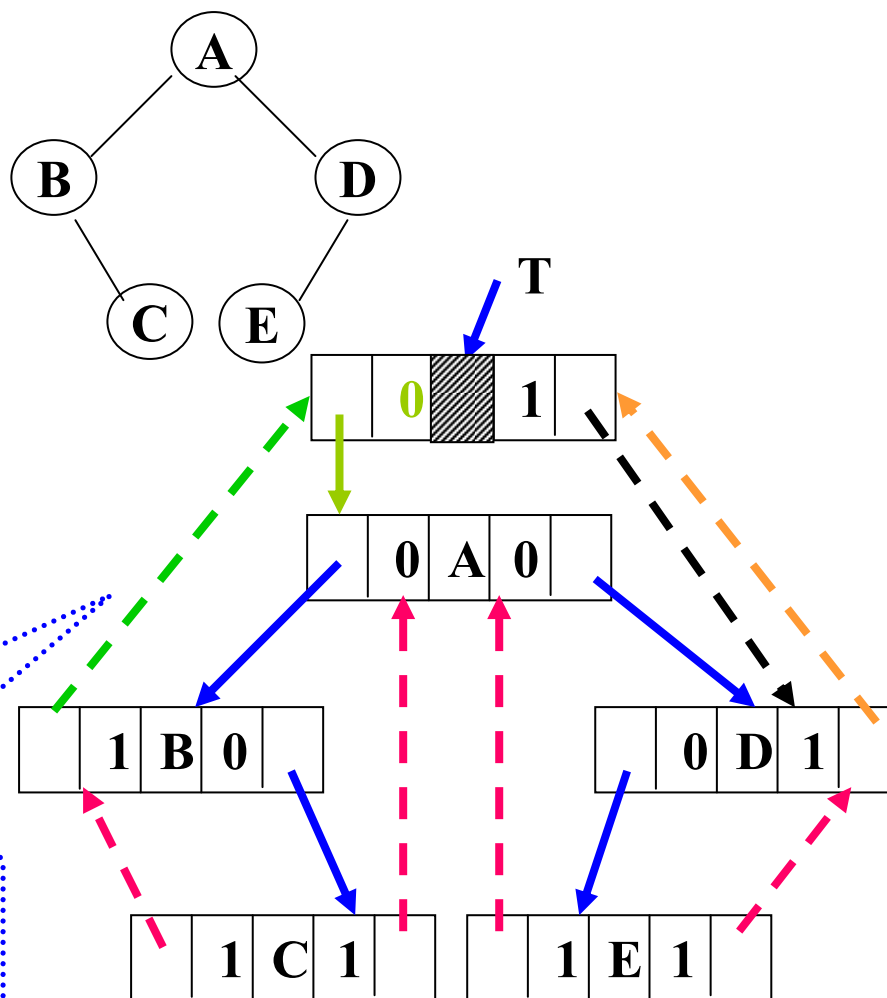
中序线索二叉树

头结点:

lt=0, lc指向根结点

rt=1, rc指向遍历序列中最后一个结点

遍历序列中第一个结点的lc域和最后一个结点的rc域都指向头结点



中序序列: **BCAED**

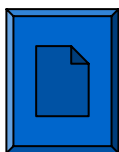
带头结点的中序线索二叉树

✚ 算法

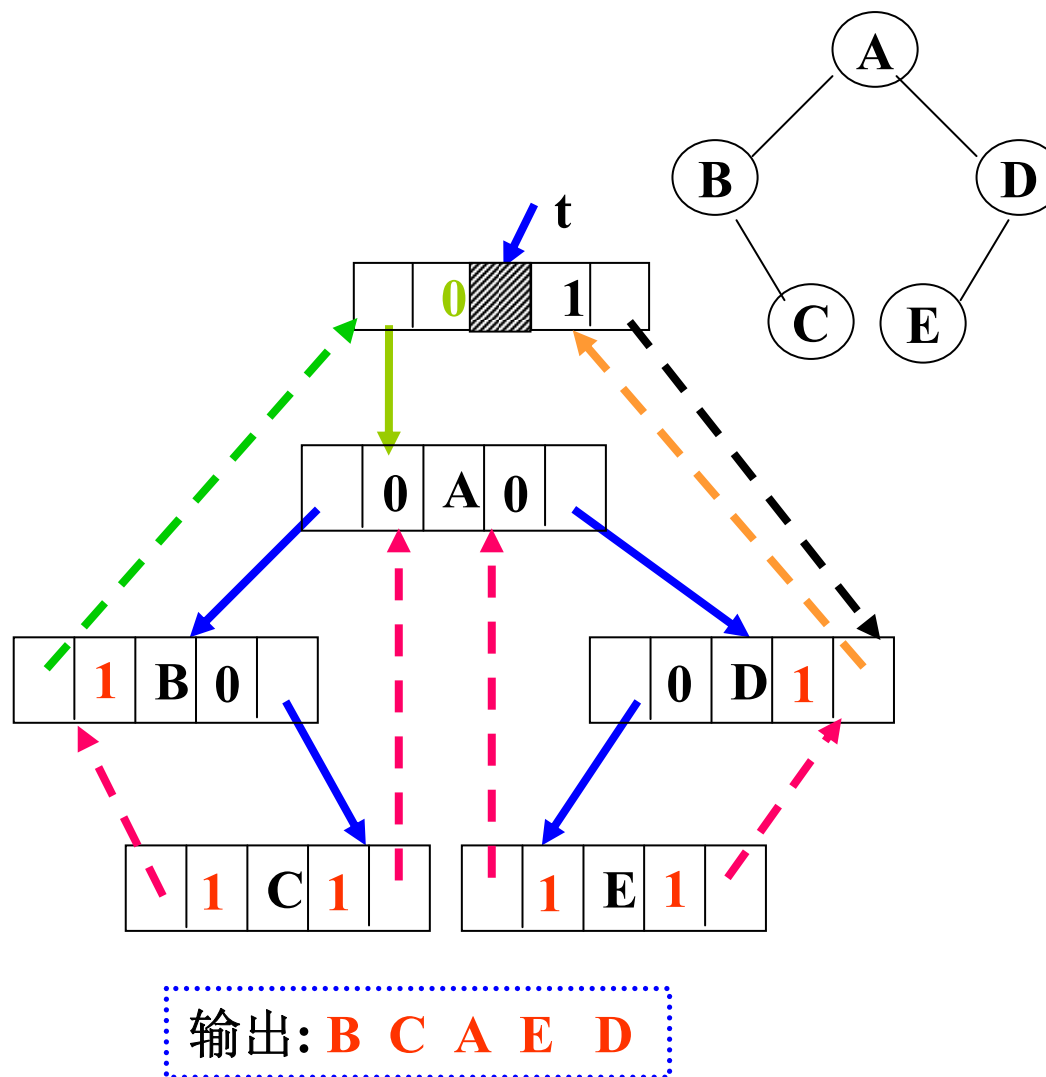
✚ 按中序线索化二叉树



Ch5_20.txt

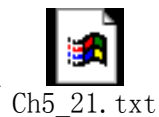


Ch5_20.c



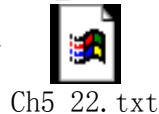
算法

按中序线索化二叉树

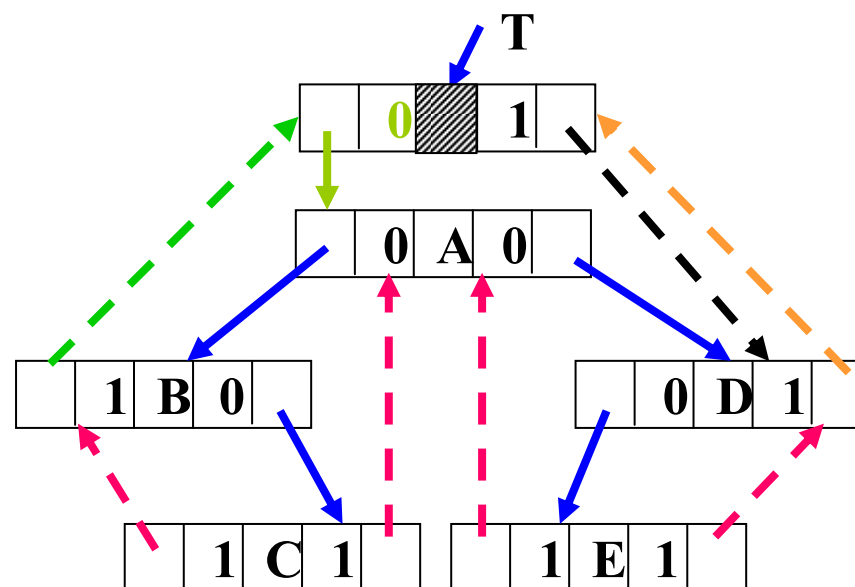
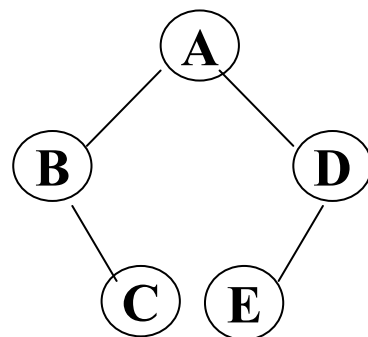


Ch5_21.txt

遍历中序线索二叉树



Ch5_22.txt



中序序列: **BCAED**

在中序线索二叉树中找结点后继的方法: 带头结点的中序线索二叉树

- (1) 若 $rt=1$, 则 rc 域直接指向其后继
- (2) 若 $rt=0$, 则结点的后继应是其右子树的左链尾 ($lt=1$) 的结点

在中序线索二叉树中找结点前驱的方法:

- (1) 若 $lt=1$, 则 lc 域直接指向其前驱
- (2) 若 $lt=0$, 则结点的前驱应是其左子树的右链尾 ($rt=1$) 的结点

THE END