树

目 录

Contents

树

1 树的定义

树(Tree)是n(n≧0)个结点的有限集合T，若n=0时称为空树，否则：

（1） 有且只有一个特殊的称为树的根(Root)结点；

（2） 若n>1时，其余的结点被分为m(m>0)个互不相交的子集T_1，T_2，T_3…T_m，其中每个子集本身又是一棵树，称其为根的子树(Subtree)。

这是树的递归定义，即用树来定义树，而只有一个结点的树必定仅由根组成，如图6-1(a)所示。树作为一种逻辑结构，同时也是一种分层结构，具有两个特点：

（1）树的根结点没有前驱结点，除根结点外的所有结点有且只有一个前驱结点。

（2）树中所以结点可以有零个或多个后继结点。

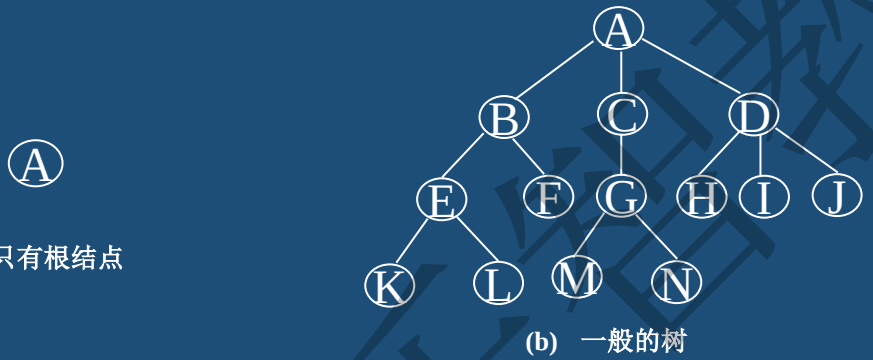
树适合于表达具有层次结构的数据。

树

2 树的基本术语

(1) 结点(node)：一个数据元素及其若干指向其子树的分支。

(2) 结点的度(degree)、树的度：结点所拥有的子树的棵数称为结点的度。树中结点度的最大值称为树的度。



(a) 只有根结点

(b) 一般的树

图6-1 树的示例形式

如图6-1(b)中结点A的度是3，结点B的度是2，结点M的度是0，树的度是3。

树

(3) 叶子(left)结点、非叶子结点：树中度为0的结点称为叶子结点(或终端结点)。相对应地，度不为0的结点称为非叶子结点(或非终端结点或分支结点)。除根结点外，分支结点又称为内部结点。

如图6-1(b)中结点H、I、J、K、L、M、N是叶子结点，而所有其它结点都是分支结点。

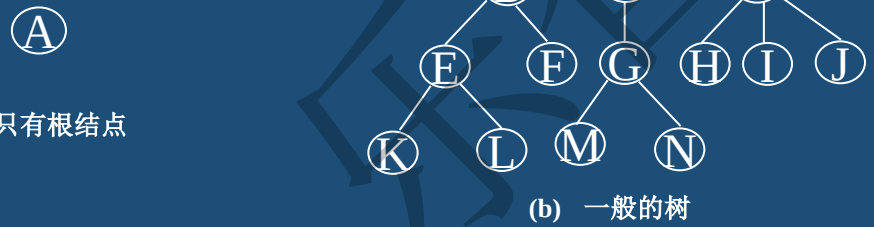


图6-1 树的示例形式

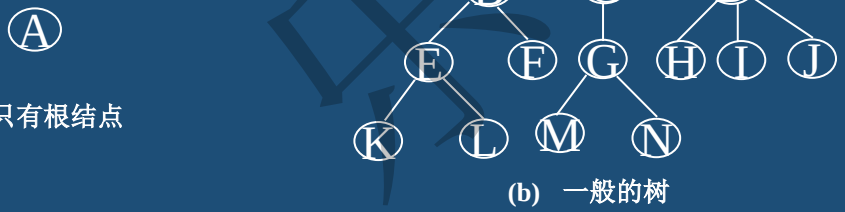
(4) 孩子结点、双亲结点、兄弟结点

一个结点的子树的根称为该结点的孩子结点(child)或子结点；相应地，该结点是其孩子结点的双亲结点(parent)或父结点。

如图6-1(b)中结点B 、C、D是结点A的子结点，而结点A是结点B 、C、D的父结点；类似地结点E 、F是结点B的子结点，结点B是结点E 、F的父结点。

同一双亲结点的所有子结点互称为兄弟结点。

如图6-1(b)中结点B 、C、D是兄弟结点；结点E 、F是兄弟结点。



(a) 只有根结点

(b) 一般的树

图6-1 树的示例形式

树

(5) 层次、堂兄弟结点

规定树中根结点的层次为1，其余结点的层次等于其双亲结点的层次加1。

若某结点在第l(l≧1)层，则其子结点在第l+1层。

双亲结点在同一层上的所有结点互称为堂兄弟结点。如图6-1(b)中结点E、F、G、H、I、J。

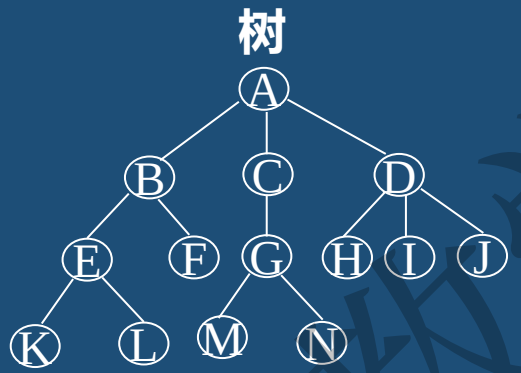
(6) 结点的层次路径、祖先、子孙

从根结点开始，到达某结点p所经过的所有结点成为结点p的层次路径(有且只有一条)。

结点p的层次路径上的所有结点（p除外）称为p的祖先(ancester) 。

以某一结点为根的子树中的任意结点称为该结点的子孙结点(descent)。

(a) 只有根结点



(b) 一般的树

图6-1 树的示例形式

结点的层次从树根开始定义，根结点为第1层，它的子结点为第2层，以此类推。

结点的深度是从根结点开始自顶向下逐层累加。

结点的高度是从叶结点开始自底向上逐层累加的。

树

⑺ 树的深度(depth)：树中结点的最大层次值，又称为树的高度，如图6-1(b)中树的高度为4。

⑻ 有序树和无序树：对于一棵树，若其中每一个结点的子树（若有）具有一定的次序，则该树称为有序树，否则称为无序树。

⑼ 森林(forest)：是m(m≧0)棵互不相交的树的集合。显然，若将一棵树的根结点删除，剩余的子树就构成了森林。

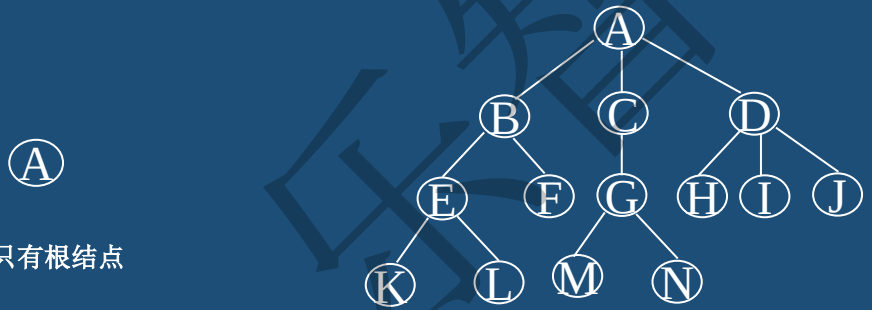


图6-1 树的示例形式

树

4 树的性质(重点)

(1) 树中的结点数等于所有结点的度数加1。

(2) 度为m的树中第i层上至多有 m^{i-1} 个结点（i≥1）。

(3) 高度为h的m叉树至多有 $\frac{m^h-1}{m-1}$ 个结点。

(4)具有n个结点的m叉树的最小高度为 $\lceil \log_m(n(m-1)+1) \rceil$ 。

二叉树

1 二叉树的定义

二叉树(Binary tree)是另一种树形结构，其特点是每个结点至多只有两颗子树（即二叉树中不存在度大于2的结点），并且，二叉树的子树有左右之分，其次序不能随意颠倒。

二叉树是n(n≥0)个结点的有限集合。若n=0时称为空树，否则：

⑴ 有且只有一个特殊的称为树的根(Root)结点；

⑵ 若n>1时，其余的结点被分成为二个互不相交的子集T_1,T_2，分别称之为左、右子树，并且左、右子树又都是二叉树。

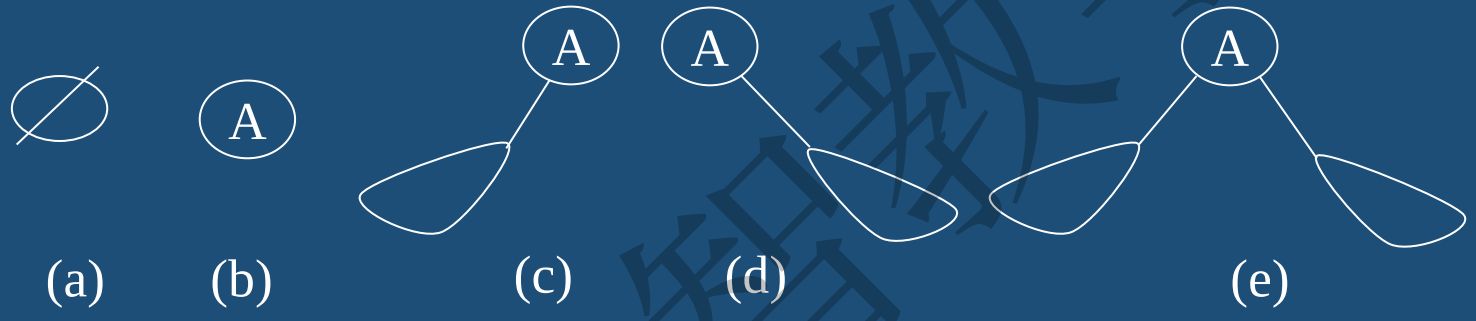
由此可知，二叉树的定义是递归的。

二叉树在树结构中起着非常重要的作用。因为二叉树结构简单，存储效率高，树的操作算法相对简单，且任何树都很容易转化成二叉树结构。上节中引入的有关树的术语也都适用于二叉树。

二叉树

2 二叉树的基本形态

二叉树有5种基本形态，如图6-3所示。



(a) 空二叉树 **(b)** 单结点二叉树 **(c)** 右子树为空

(d) 左子树为空 **(e)** 左、右子树都不空

图6-3 二叉树的基本形态

二叉树

3 满二叉树、完全二叉树和平衡二叉树

一棵深度为k且有2^k-1个结点的二叉树称为满二叉树(Full Binary Tree)。如图6-4(a) 就是一棵深度为4的满二叉树。

满二叉树的特点：

- 基本特点是每一层上的结点数总是最大结点数。
- 满二叉树的所有的支结点都有左、右子树。
- 可对满二叉树的结点进行连续编号，若规定从根结点开始，按“自上而下、自左至右”的原则进行。

平衡二叉树：树上任一结点的左子树和右子树的深度之差不超过1

二叉树

完全二叉树（Complete Binary Tree）：如果深度为k，由n个结点的二叉树，当且仅当其每一个结点都与深度为k的满二叉树中编号从1到n的结点一一对应，该二叉树称为完全二叉树。

或深度为k的满二叉树中编号从1到n的前n个结点构成了一棵深度为k的完全二叉树。其中 $2^{k-1} \leqq n \leqq 2^{k}-1$ 。

完全二叉树是满二叉树的一部分，而满二叉树是完全二叉树的特例。

二叉树

完全二叉树的特点（重点）

（1）若完全二叉树的深度为k，则所有的叶子结点都出现在第k层或k-1层。对于任一结点，如果其右子树的最大层次为*l*，则其左子树的最大层次为*l*或*l*+1。

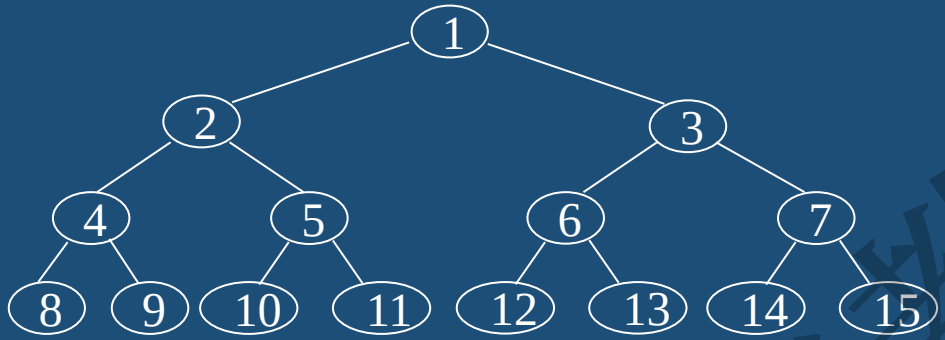
（2）若i≤⌊n/2⌋，则结点i为分支结点，否则为叶子结点

（3）叶子结点只可能在层次最大的两层上出现。对于最大层次中的叶子结点，都依次排列在该层的最左边的位置

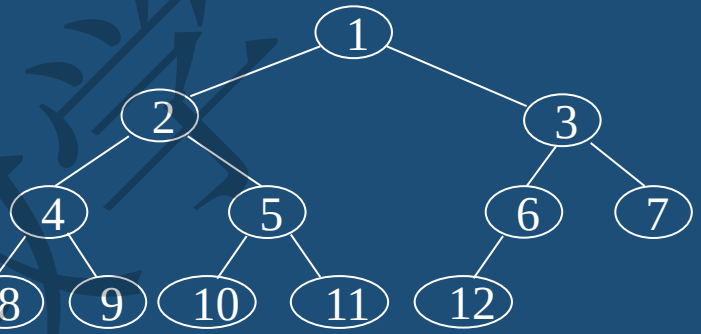
（4）如果有度为1 的结点，只可能有一个，且该结点只有左孩子而无左右孩子。

（5）按层序编号后，一旦出现某结点（其编号为i）为叶子结点或只有左孩子，则编号大于i的结点均为叶子结点。

二叉树



(a) 满二叉树



(b) 完全二叉树

(c) 非完全二叉树

图6-4 特殊形态的二叉树

二叉树

4 二叉树的性质（重点）

性质1：在非空二叉树中，第i层上至多有2^{i-1}个结点($i \geqq 1$)。

性质2：深度为k的二叉树至多有2^k-1个结点（$k \geqq 1$）。

性质3：对任何一棵二叉树，若其叶子结点数为n_0，度为2的结点数为n_2，则$n_0=n_2+1$。

性质4：n个结点的完全二叉树深度为：$\lfloor \log_2 n \rfloor+1$或$\lceil \log_2(n+1) \rceil$。

其中符号：$\lfloor x \rfloor$表示不大于x的最大整数。

$\lceil x \rceil$表示不小于x的最小整数。

二叉树

4 二叉树的性质

性质5：若对一棵有n个结点的完全二叉树(深度为$\lfloor \log_2 n \rfloor +1$)的结点按层（从第1层到第$\lfloor \log_2 n \rfloor +1$层)序自左至右进行编号,则对于编号为i（$1 \leq i \leq n$)的结点：

(1) 若i=1：则结点i是二叉树的根，无双亲结点；否则，若i>1，则其双亲结点编号是$\lfloor i/2 \rfloor$。

(2) 如果2i>n：则结点i为叶子结点，无左孩子；否则，其左孩子结点编号是2i。

(3) 如果2i+1>n：则结点i无右孩子；否则，其右孩子结点编号是2i+1。

二叉树

二叉树的性质

1）非空二叉树上叶子结点数等于度为2的结点数加1，即$N_0 = N_2 + 1$

2）非空二叉树上第K层上至多有2^{K-1}个结点（K≥1）

3）高度为H的二叉树至多有2^H-1个结点（H≥1）

4）对完全二叉树按从上到下、从左至右的顺序依次编号1，2，3，…，N，则有以下关系：

①当i>1时，结点i的双亲结点编号为⌊i/2⌋，即当i为偶数时，其双亲结点的编号为i/2，它是双亲结点的左孩子；当i是奇数时，其双亲结点的编号为（i-1）/2，它是双亲结点的右孩子。

②当2i≤N时，结点i的左孩子编号为2i，否则无左孩子。

③当2i+1≤N时，结点i的右孩子编号为2i+1，否则无右孩子。

④结点i所在层次（深度）为$\lfloor \log_2 i \rfloor +1$

5）具有N个（N>0）结点的完全二叉树的高度为$\lfloor \log_2 N \rfloor +1$或 $\lceil \log_2 (N+1) \rceil$ 。

二叉树

5 二叉树的存储结构

1. 顺序存储结构

依据二叉树的性质，完全二叉树和满二叉树采用顺序存储比较合适

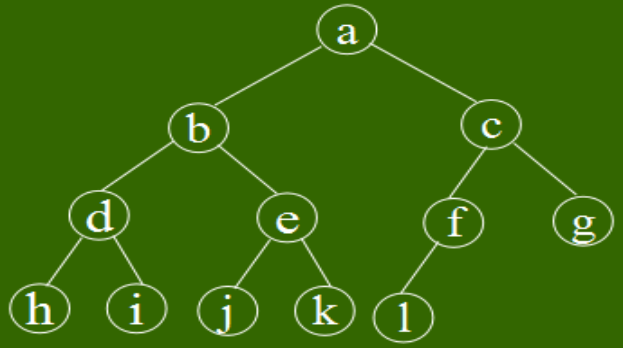
用一组地址连续的存储单元依次“自上而下、自左至右”存储完全二叉树的数据元素。

对于完全二叉树上编号为i的结点元素存储在一维数组的下标值为i-1的分量中，如图6-6(c)所示。

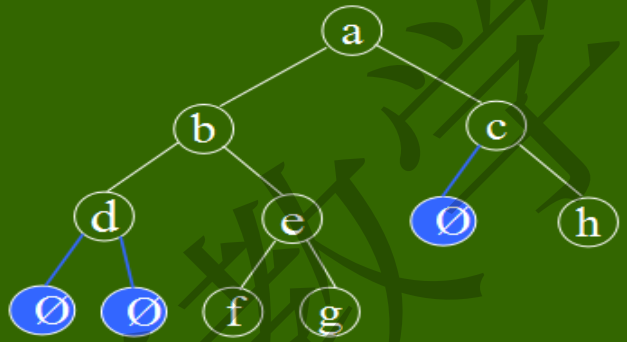
对于一般的二叉树，将其每个结点与完全二叉树上的结点相对照，存储在一维数组中，如图6-6(d)所示。

最坏的情况下，一个深度为k且只有k个结点的单支树需要长度为2^k-1的一维数组。

二叉树



(a) 完全二叉树



(b) 非完全二叉树

1	2	3	4	5	6	7	8	9	10	11	12
a	b	c	d	e	f	g	h	i	j	k	l

(c) 完全二叉树的顺序存储形式

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

(d) 非完全二叉树的顺序存储形式

图6-6 二叉树及其顺序存储形式

二叉树

5 二叉树的存储结构

2. 链式存储结构

设计不同的结点结构可构成不同的链式存储结构。

(1) 结点的类型及其定义

① **二叉链表结点**。有三个域：一个数据域，两个分别指向左右子结点的指针域，如图6-7(a)所示。

② **三叉链表结点**。除二叉链表的三个域外，再增加一个指针域，用来指向结点的父结点，如图6-7(b)所示。

Lchild	data	Rchild

(a) 二叉链表结点

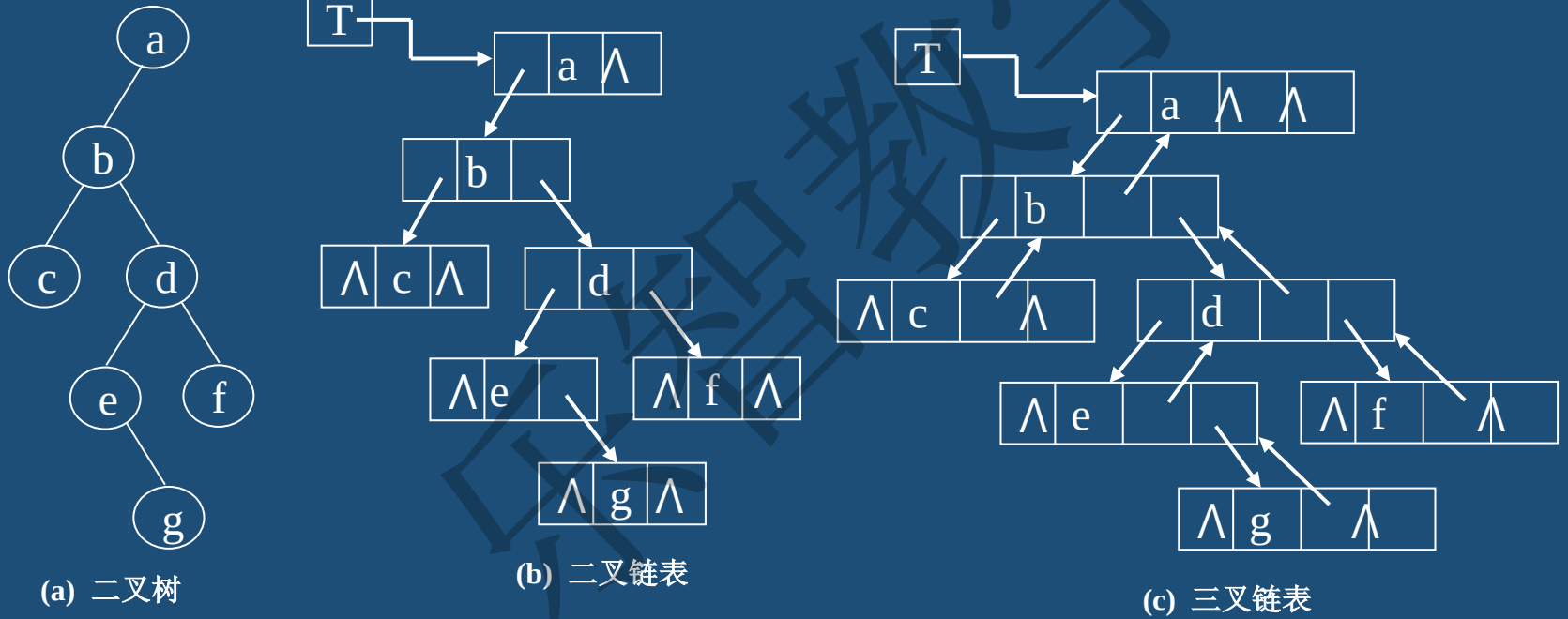
Lchild	data	parent	Rchild

(b) 三叉链表结点

图6-7 链表结点结构形式

二叉树

5 二叉树的存储结构



(a) 二叉树　(b) 二叉链表　(c) 三叉链表

图6-8　二叉树及其链式存储结构

二叉树

6 遍历二叉树（重点）

遍历二叉树(Traversing Binary Tree)：是指**按指定的规律**对二叉树中的**每个结点访问一次且仅访问一次。**

二叉树的基本组成：根结点、左子树、右子树。若能依次遍历这三部分，就是遍历了二叉树。

若以L、D、R分别表示遍历左子树、遍历根结点和遍历右子树，则有六种遍历方案：DLR、LDR、LRD、DRL、RDL、RLD。若规定**先左后右**，则只有前三种情况三种情况，分别是：

DLR——先(根)序遍历。

LDR——中(根)序遍历。

LRD——后（根）序遍历。

此外，还有层序遍历。

6 遍历二叉树

1.先(根)序遍历 （根左右）

算法的递归定义是：

若二叉树为空，则遍历结束；否则

(1) 访问根结点；

(2) 先序遍历左子树(递归调用本算法)；

(3) 先序遍历右子树(递归调用本算法)。

2.中(根)序遍历（左根右）

算法的递归定义是：

若二叉树为空，则遍历结束；否则

(1) 中序遍历左子树(递归调用本算法)；

(2) 访问根结点；

(3) 中序遍历右子树(递归调用本算法)。

6 遍历二叉树

3.后(根)序遍历（左右根）

算法的递归定义是：

若二叉树为空，则遍历结束；否则

(1) 后序遍历左子树(递归调用本算法)；

(2) 后序遍历右子树(递归调用本算法)；

(3) 访问根结点 。

三种遍历的时间复杂度都为O(n)

4.层次遍历（需要借助一个队列）

层次遍历二叉树，是从根结点开始遍历，按层次次序“自上而下，从左至右”访问树中的各结点。

6 遍历二叉树

如图6-9所示的二叉树表示表达式：(a+b*(c-d)-e/f)

按不同的次序遍历此二叉树，将访问的结点按先后次序排列起来的次序是：

其先序序列为：　-+a*b-cd/ef

其中序序列为：　a+b*c-d-e/f

其后序序列为：　abcd-*+ef/-

其层序序列为：　-+/a*efb-cd

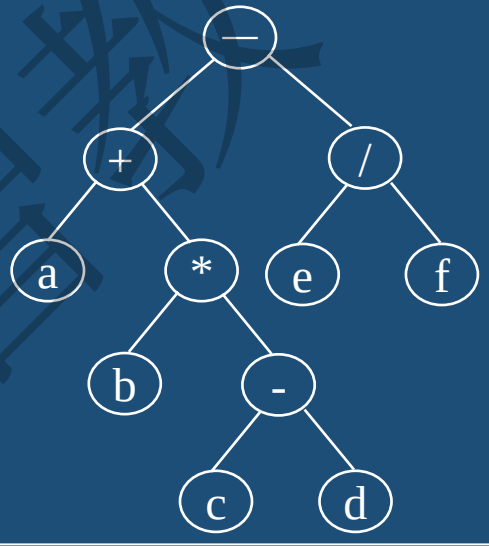


图6-9　表达式 (a+b*(c-d)-e/f)二叉树

二叉树

1.非空二叉树的第K层的结点数最多为（ ）

A. 2^k-1　　B. 2^{k+1}　　C. 2^{k-1}　　D. 2^k

2.设某二叉树中有2000个结点，则该二叉树的最小高度是（ ）

A.9　　B.10　　C.11　　D.12

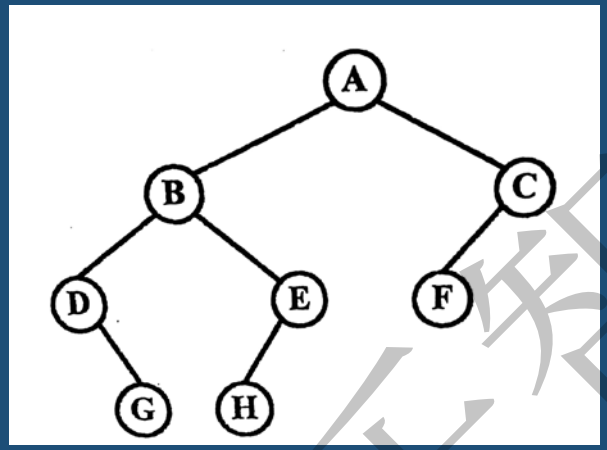
3.若二叉树有11个叶子结点，则该二叉树中度为2 的结点个数是（ ）

A.10　　B.11　　C.12　　D.不确定

答案：C　C　A

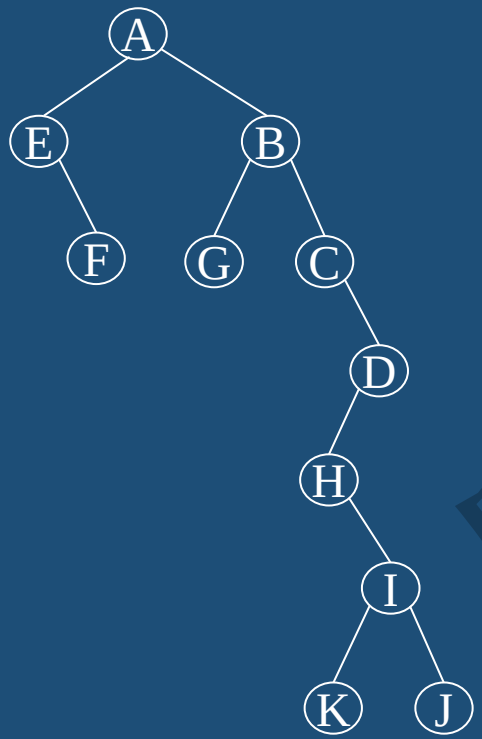
二叉树（真题检测）

4.已知二叉树如图所示。请分别给出该二叉树的先序遍历、中序遍历、后序遍历结果。



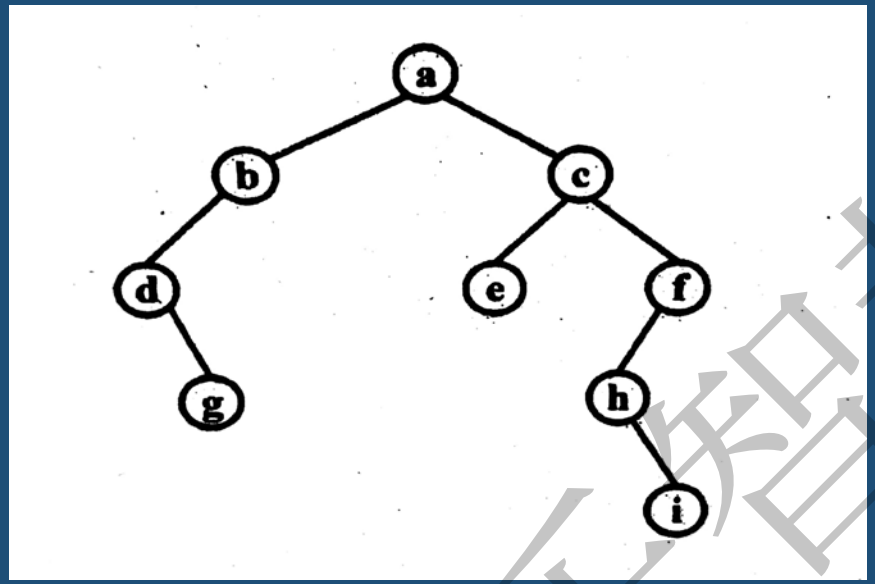
先根序序列：ABDGEHCF
中根序序列：DGBHEAFC
后根序序列：GDHEBFCA

5. 已知二叉树的前序遍历序列为AEFBGCDHIKJ，中序遍历序列是EFAGBCHKIJD，画出此二叉树，并写出后序遍历结果。



后续遍历：**FEGKJIHDCBA**

6. 根据所给的二叉树，写出四种遍历序列（先、中、后、层）。



先序遍历：abdgcefhi
中序遍历：dgbaechif
后续遍历：gdbeihfca
层次遍历：abcdefghi

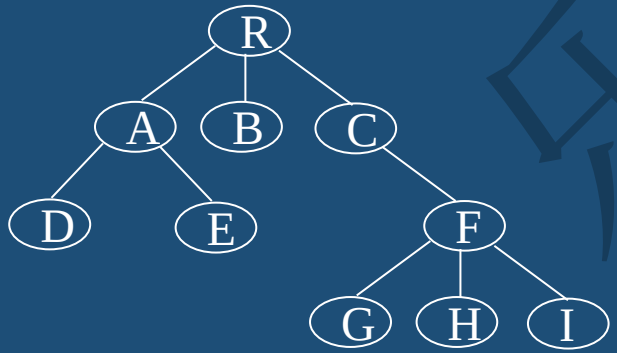
树与森林

1 树的存储结构

1. 双亲表示法(顺序存储结构)

用一组连续的存储空间来存储树的结点，同时在每个结点中附加一个指示器(整数域)，用以指示双亲结点的位置(下标值)。

这种存储结构利用了任一结点的父结点唯一的性质。可以方便地直接找到任一结点的父结点，但求结点的子结点时需要扫描整个数组。



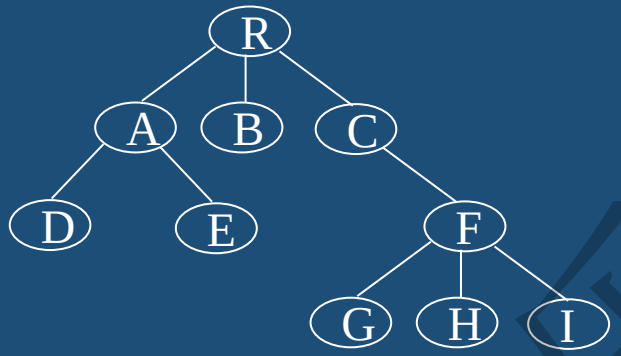
0	R	-1
1	A	0
2	B	0
3	C	0
4	D	1
5	E	1
6	F	3
7	G	6
8	H	6
9	I	6

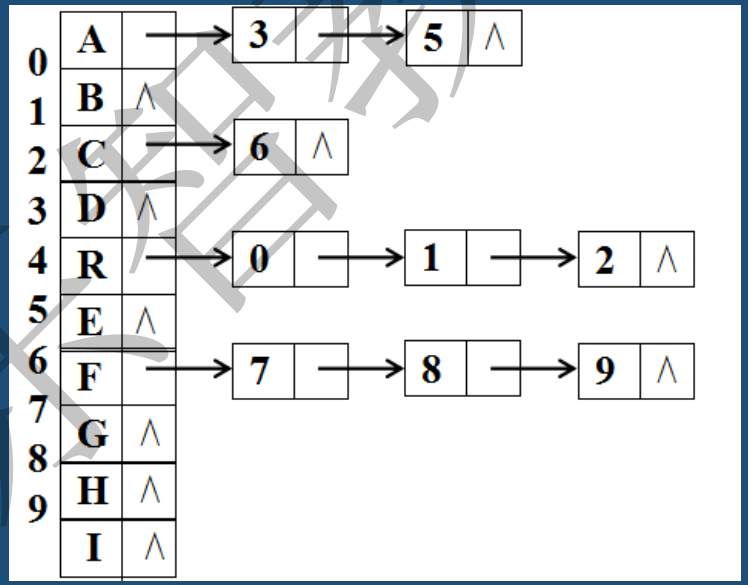
树与森林

1 树的存储结构

2. 孩子链表表示法

树中每个结点有多个指针域，每个指针指向其一棵子树的根结点。





树与森林

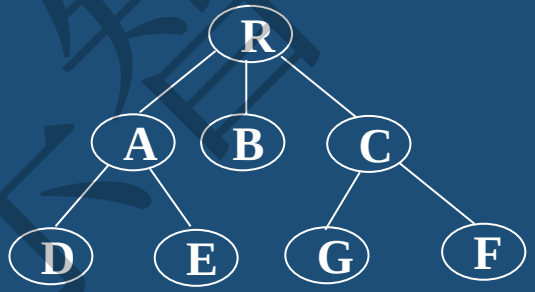
1 树的存储结构

3. 孩子兄弟表示法(二叉树表示法)

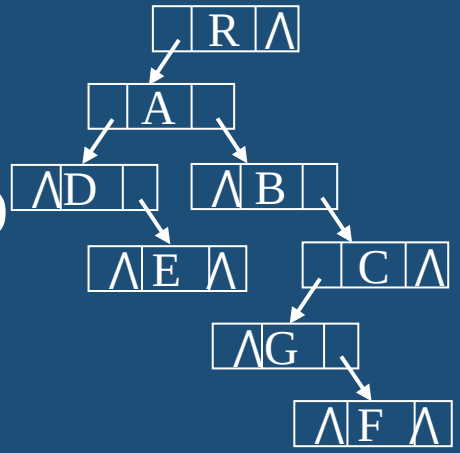
以二叉链表作为树的存储结构，其结点形式如图6-17(a)所示。

两个指针域：分别指向结点的第一个子结点和下一个兄弟结点。

(a) 结点结构



(b) 树



(c) 孩子兄弟存储结构

2 森林与二叉树的转换

1. 树转换成二叉树

对于一般的树，可以方便地转换成一棵唯一的二叉树与之对应。其详细步骤是：

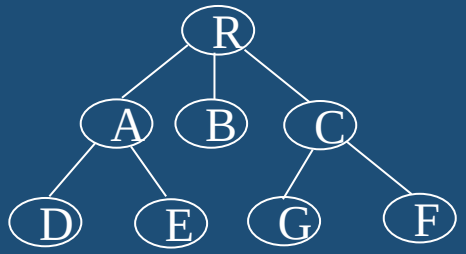
(1) 加虚线。在树的每层按从“左至右”的顺序在兄弟结点之间加虚线相连。

(2) 去连线。除最左的第一个子结点外，父结点与所有其它子结点的连线都去掉。

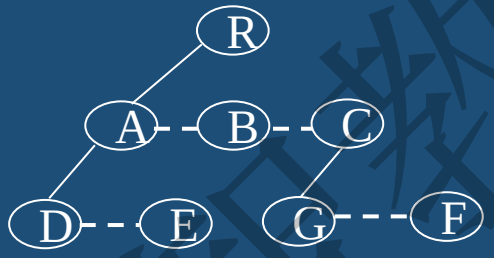
(3) 旋转。将树顺时针旋转45^0，原有的实线左斜。

(4) 整型。将旋转后树中的所有虚线改为实线，并向右斜。该转换过程如图6-19所示。

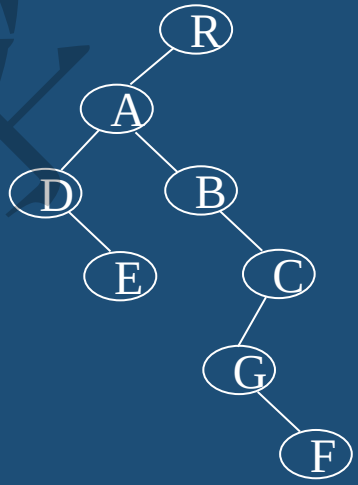
树与森林



(a) 一般的树



(b) 加虚线，去连线后



(C) 转换后的二叉树

这样转换后的二叉树的特点是：

- 二叉树的根结点没有右子树，只有左子树；
- 左子结点仍然是原来树中相应结点的左子结点，而所有沿右链往下的右子结点均是原来树中该结点的兄弟结点。

树与森林

2 森林与二叉树的转换

2. 二叉树转换成树

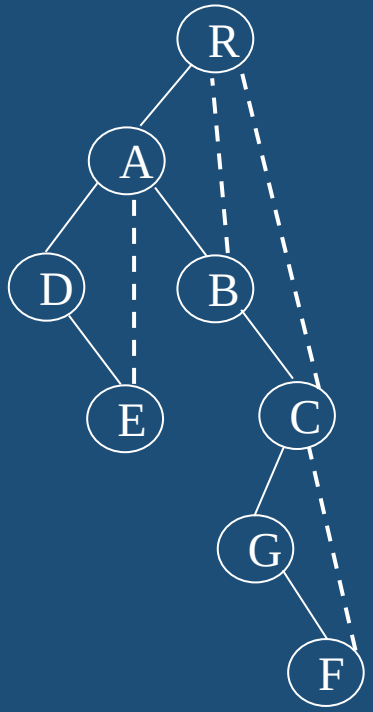
其步骤是：

(1) 加虚线。若某结点i是其父结点的左子树的根结点，则将该结点i的右子结点以及沿右子链不断地搜索所有的右子结点，将所有这些右子结点与i结点的父结点之间加虚线相连，如图6-20(a)所示。

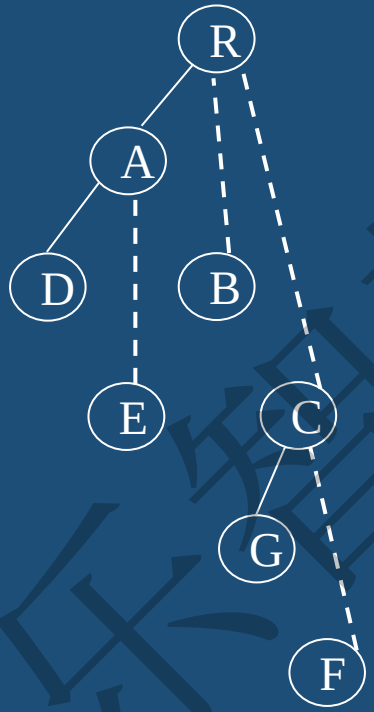
(2) 去连线。去掉二叉树中所有父结点与其右子结点之间的连线，如图6-20(b)所示。

(3) 规整化。将图中各结点按层次排列且将所有的虚线变成实线，如图6-20(c)所示。

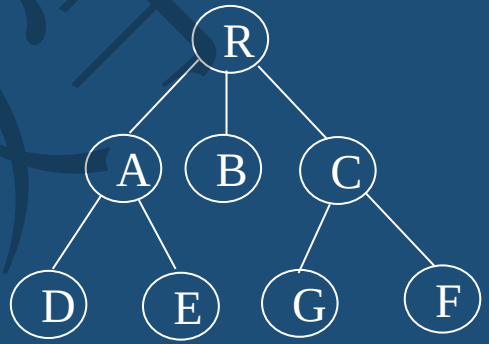
树与森林



(a) 加虚线后



(b) 去连线后



(C) 还原后的树

图6-20 二叉树向树的转换过程

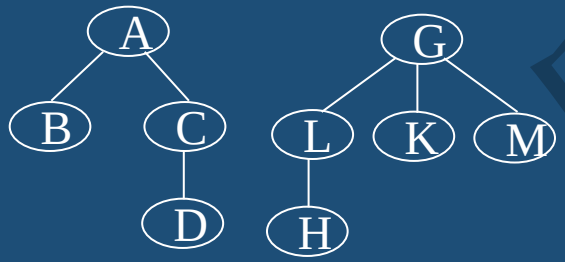
2 森林与二叉树的转换

3. 森林转换成二叉树

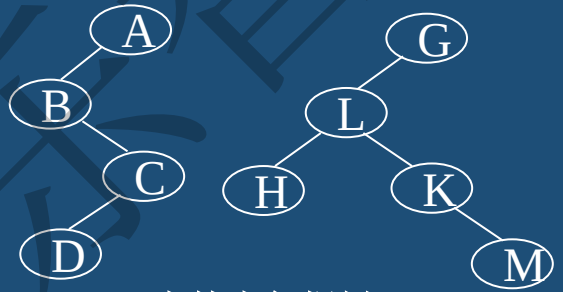
转换步骤：

① 将$F=\{T_1, T_2, \cdots, T_n\}$中的每棵树转换成二叉树。

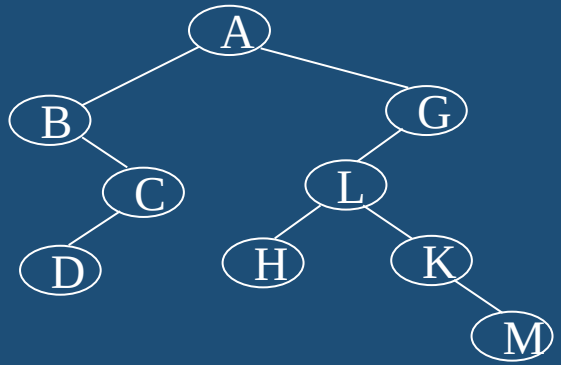
② 按给出的森林中树的次序，从最后一棵二叉树开始，每棵二叉树作为前一棵二叉树的根结点的右子树，依次类推，则第一棵树的根结点就是转换后生成的二叉树的根结点，如图所示。



(a) 森林



(b) 森林中每棵树对应的二叉树



(c) 森林对应的二叉树

2 森林与二叉树的转换

4. 二叉树转换成森林

转换步骤：

① 去连线。将二叉树B的根结点与其右子结点以及沿右子结点链方向的所有右子结点的连线全部去掉，得到若干棵孤立的二叉树，每一棵就是原来森林F中的树依次对应的二叉树，如图6-22(b)所示。

② 二叉树的还原。将各棵孤立的二叉树按二叉树还原为树的方法还原成一般的树，如图6- 22(c)所示。

树与森林

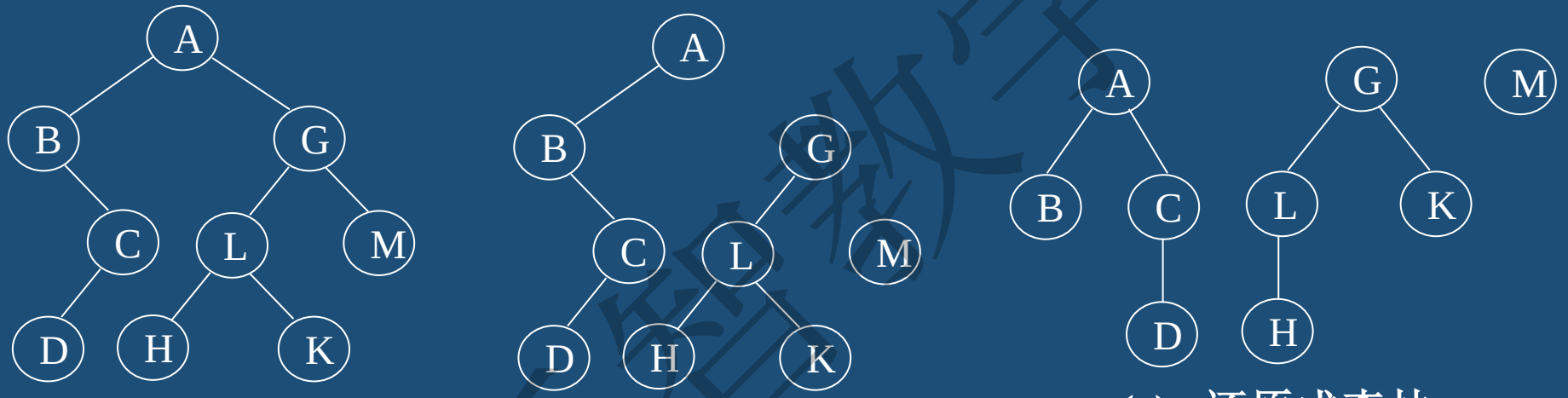


图6-22 二叉树还原成森林的过程

树与森林

3 树和森林的遍历

1. 树的遍历

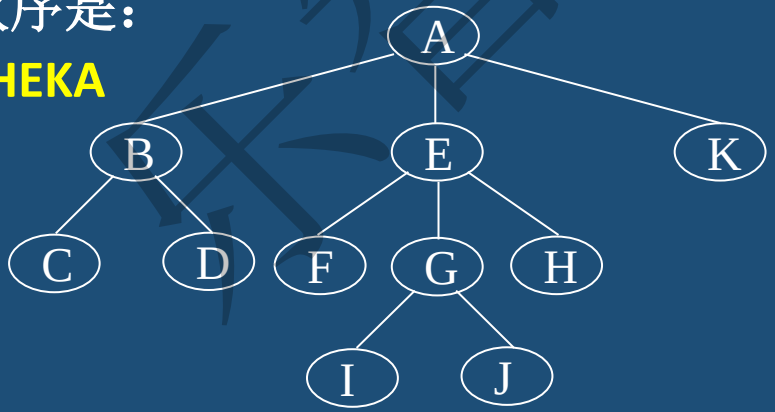
由树结构的定义可知，树的遍历有二种方法。

(1) 先序遍历：先访问根结点，然后依次先序遍历完每棵子树。如图的树，先序遍历的次序是：

ABCDEFGIJHK

(2) 后序遍历：先依次后序遍历完每棵子树，然后访问根结点。如图的树，后序遍历的次序是：

CDBFIJGHEKA



树与森林

说明：

- 树的先序遍历实质上与将树转换成二叉树后对二叉树的先序遍历相同。
- 树的后序遍历实质上与将树转换成二叉树后对二叉树的中序遍历相同。

树与森林

3 树和森林的遍历

2 森林的遍历

设F={T_1, T_2,…,T_n}是森林，对F的遍历有二种方法。

(1) 先序遍历：按先序遍历树的方式依次遍历F中的每棵树。

(2) 中序遍历：按后序遍历树的方式依次遍历F中的每棵树。

树和森林的遍历与二叉树遍历的对应关系

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

树的应用

4 赫夫曼树及其应用

赫夫曼(Huffman)树又称最优树，是一类带权路径长度最短的树。

1. 基本概念

① 结点路径：从树中一个结点到另一个结点的之间的分支构成这两个结点之间的路径。

② 路径长度：结点路径上的分支数目称为路径长度。

③ 树的路径长度：从树根到每一个结点的路径长度之和。

树的应用

④ 结点的带权路径长度：从该结点的到树的根结点之间的路径长度与结点的权(值)的乘积。

权(值)：各种开销、代价、频度等的抽象称呼。

⑤ 树的带权路径长度：树中所有叶子结点的带权路径长度之和，记做：

$WPL=w_1\times l_1+w_2\times l_2+\cdots+w_n\times l_n=\sum w_i\times l_i \quad (i=1,2,\cdots,n)$

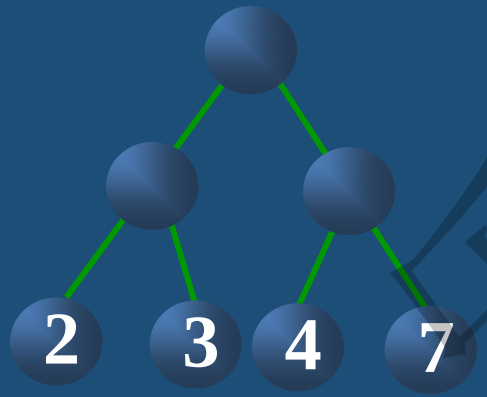
其中：n为叶子结点的个数；w_i为第i个结点的权值；l_i为第i个结点的路径长度。

树的应用

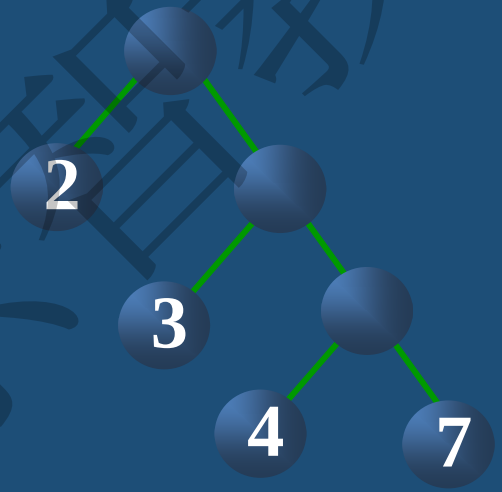
4 赫夫曼树及其应用

哈夫曼树：给定一组具有确定权值的叶子结点，带权路径长度最小的二叉树。

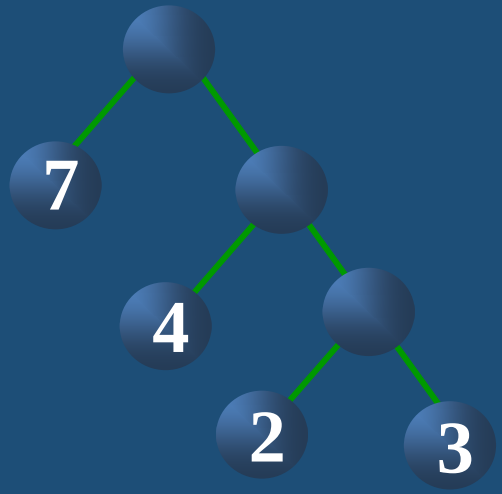
例：给定4个叶子结点，其权值分别为{2，3，4，7}，可以构造出形状不同的多个二叉树



WPL=32



WPL=41



WPL=30

树的应用

4 赫夫曼树及其应用

2.哈夫曼树的构造

树的应用

4 赫夫曼树及其应用

所构造的Huffman树的WPL是：

WPL=6×2+3×3+4×3+8×2+5×3+5×3 =79

树的应用

4 赫夫曼树及其应用

Huffman编码

在电报收发等数据通讯中，常需要将传送的文字转换成由二进制字符0、1组成的字符串来传输。为了使收发的速度提高，就要求电文编码要尽可能地短。此外，要设计长短不等的编码，还必须保证任意字符的编码都不是另一个字符编码的前缀，这种编码称为前缀编码。

Huffman树可以用来构造编码长度不等且译码不产生二义性的编码。

树的应用

4 赫夫曼树及其应用

Huffman编码方法

以字符集C作为叶子结点，次数或频度集W作为结点的权值来构造Huffman树。规定Huffman树中左分支代表“0”，右分支代表“1”。

从根结点到每个叶子结点所经历的路径分支上的“0”或“1”所组成的字符串，为该结点所对应的编码，称之为Huffman编码。

由于每个字符都是叶子结点，不可能出现在根结点到其它字符结点的路径上，所以一个字符的Huffman编码不可能是另一个字符的Huffman编码的前缀。

4 赫夫曼树及其应用

Huffman编码方法

示例:

我们对一个字符串进行统计发现a-f出现的频率分别为a:45，b:13，c:12，d:16，e:9，f:5，我们对该字符串进行采用哈夫曼编码进行存储。

各字符编码为:

a:0

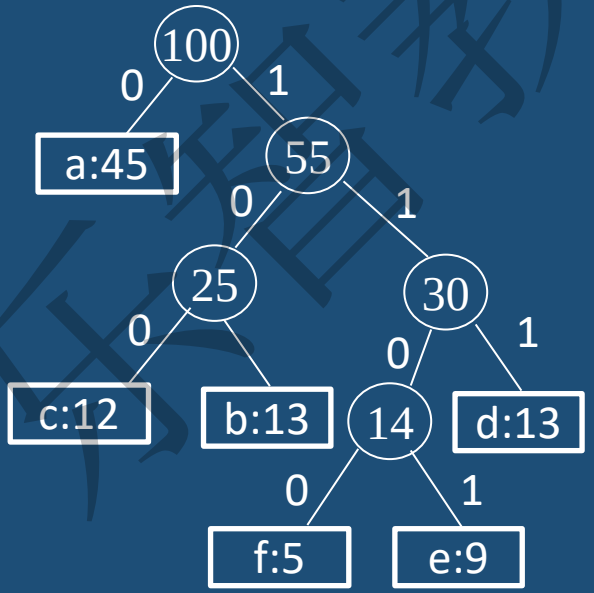
b：101

c：100

d：111

e：1101

f：1100



树（真题检测）

1.一棵度为2的树与一棵二叉树有什么区别？

答案：

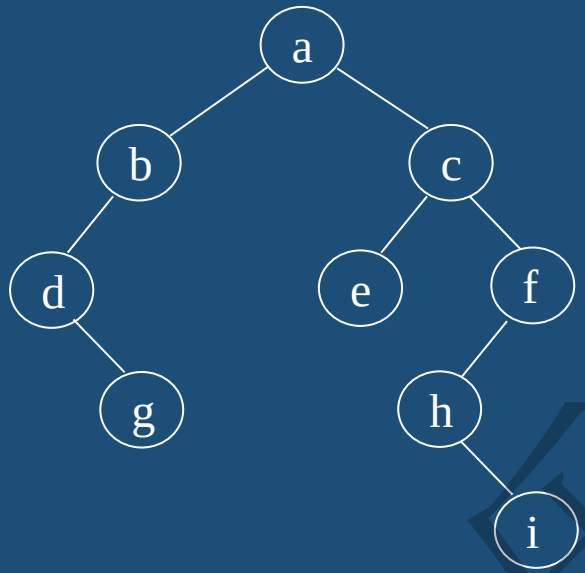
1、度为2的树是不区分左子树和右子树.而二叉树是要分左子树和右子树的.

2、度为2的数不包含空树,而二叉树是可以有空树的.

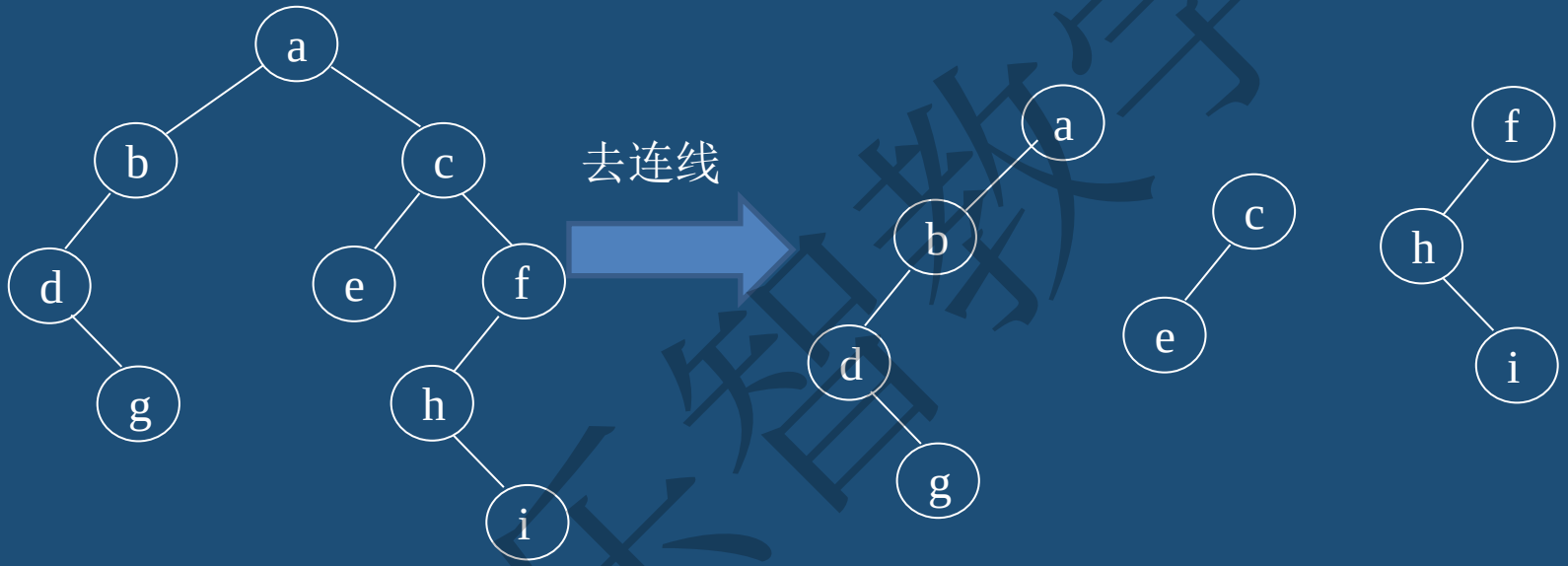
总之,二叉树的定义要比度为2的树定义更为严格,更为详细.

树（真题检测）

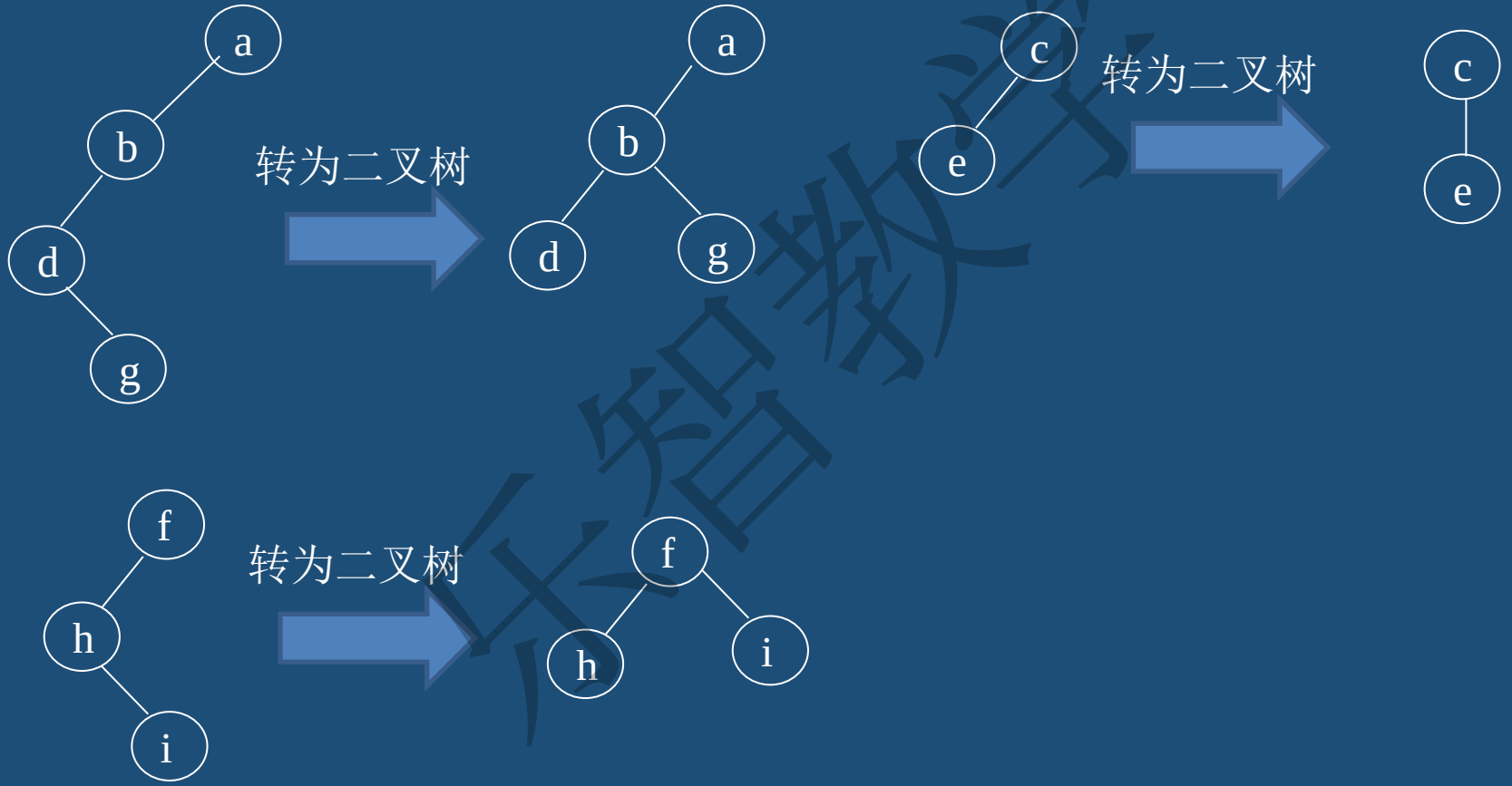
2.画出该二叉树对应的森林。



树（真题检测）



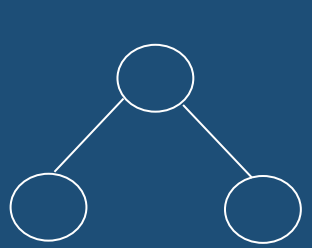
树（真题检测）



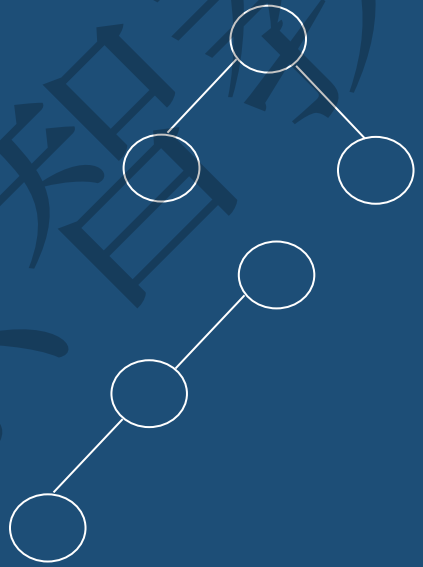
树（真题检测）

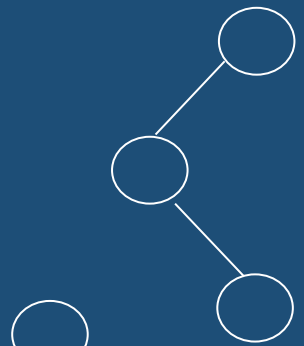
4.画出具有三个结点的树和三个结点的二叉树的所有形态

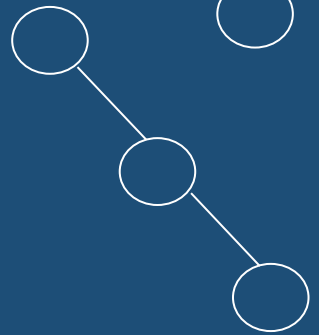
三个结点的树有两种形态

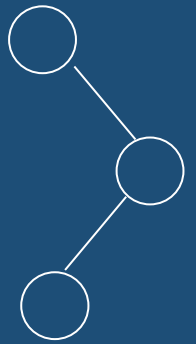


三个结点的二叉树有五种形态









谢 谢 观 看