

第6章 树和二叉树



第6章 树和二叉树

6.1 树的定义和基本术语

6.2 二叉树

6.3 遍历二叉树和线索二叉树

6.4 树和森林

6.5 哈夫曼树及其应用

6.6 赫夫曼树及赫夫曼编码

字符	电码符号	字符	电码符号	字符	电码符号
A	• —	N	— •	1	• ————
B	— • • •	O	———	2	• • ———
C	— • — •	P	• —— •	3	• • • ——
D	— • •	Q	—— • —	4	• • • • —
E	•	R	• — •	5	• • • • •
F	• • — •	S	• • •	6	— • • • •
G	—— •	T	—	7	—— • • •
H	• • • •	U	• • —	8	——— • •
I	• •	V	• • • —	9	——— •
J	• ———	W	• ———	0	———
K	— • —	X	— • • —	?	• • —— • •
L	• — • •	Y	— • • ——	/	— • • — •
M	——	Z	—— • •	()	— • —— • —
				—	— • • • • —

6.6 赫夫曼树及赫夫曼编码

if($a < 60$)

$b = \text{“不及格”};$

else if($a < 70$)

$b = \text{“及格”};$

else if($a < 80$)

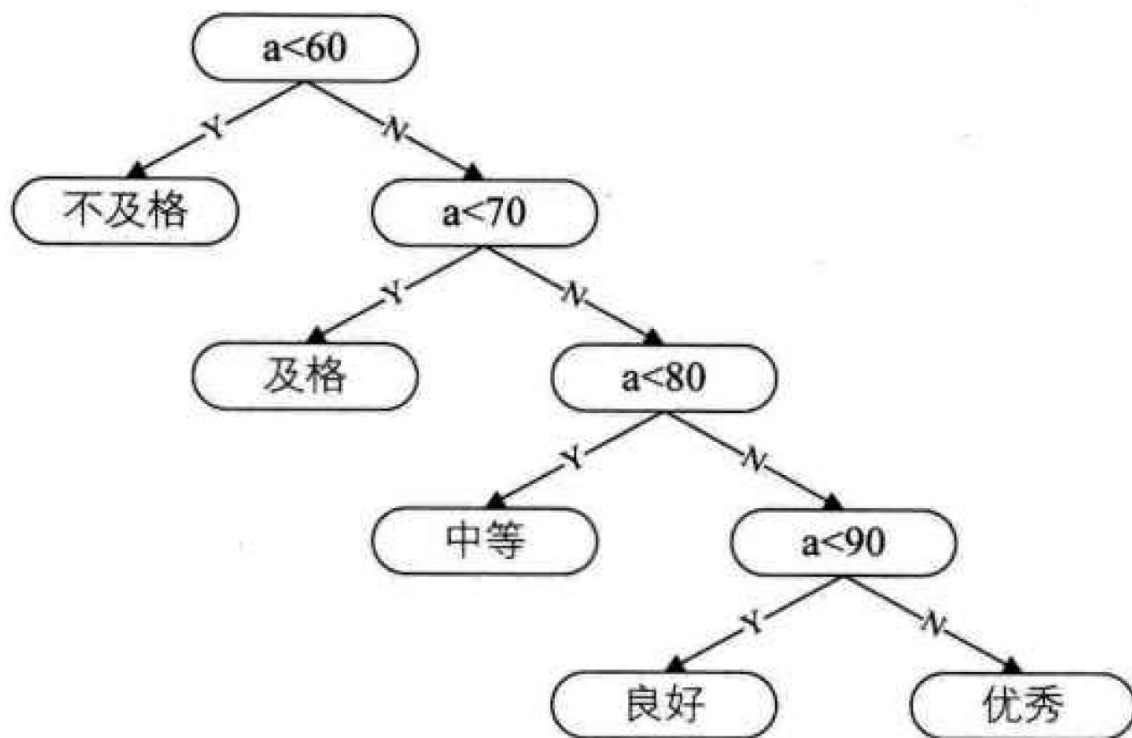
$b = \text{“中等”};$

else if($a < 90$)

$b = \text{“良好”};$

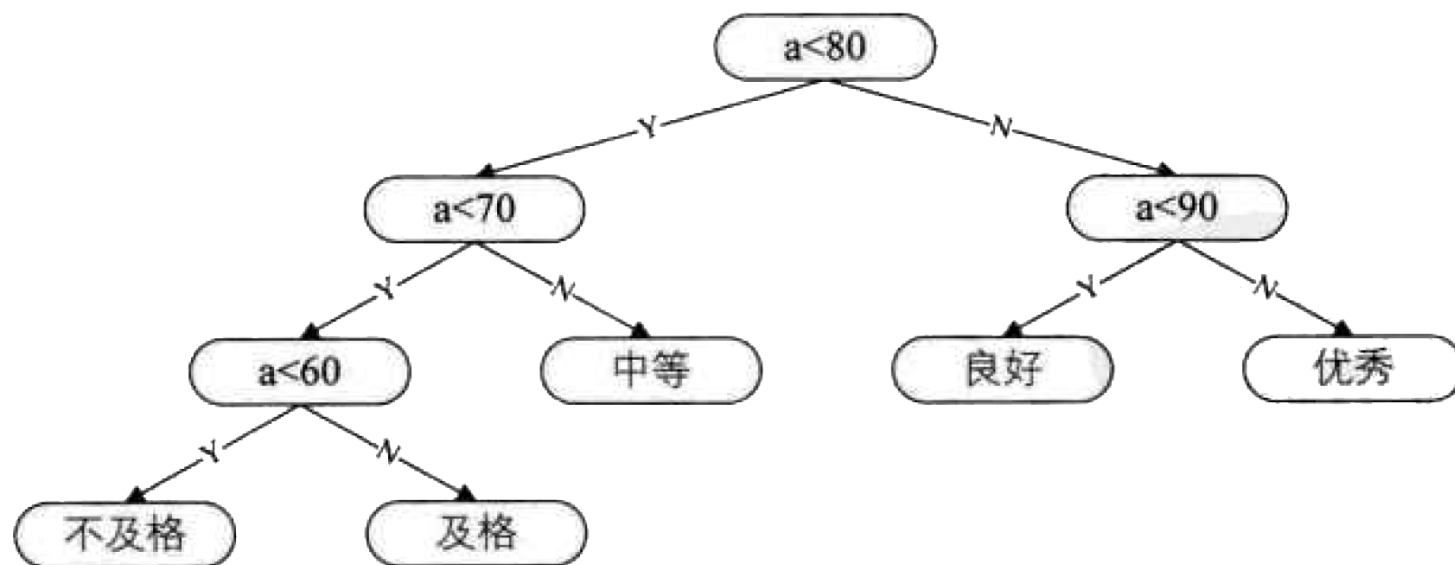
else

$b = \text{“优秀”};$



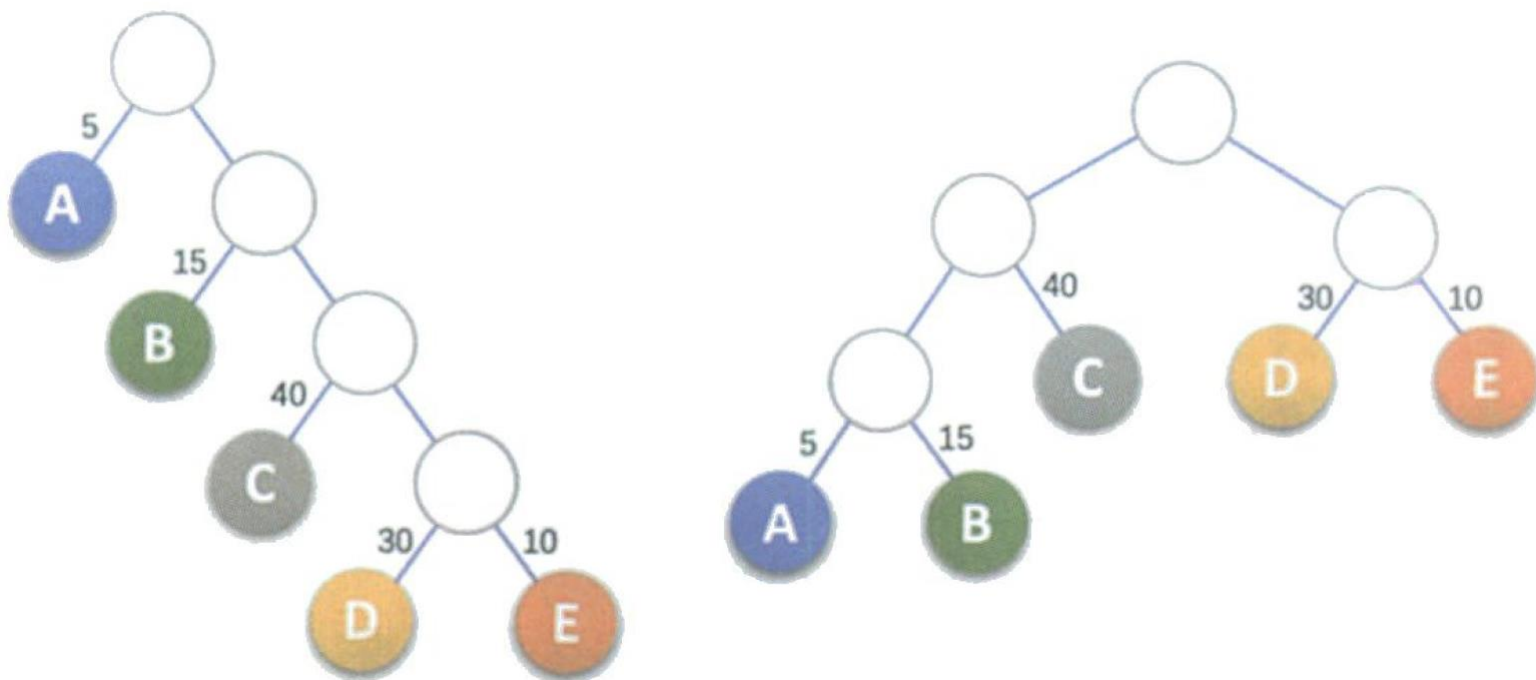
6.6 赫夫曼树及赫夫曼编码

分数	0 ~ 59	60 ~ 69	70 ~ 79	80 ~ 89	90 ~ 100
所占比例	5%	15%	40%	30%	10%



6.6 赫夫曼树及赫夫曼编码

分数	0 ~ 59	60 ~ 69	70 ~ 79	80 ~ 89	90 ~ 100
所占比例	5%	15%	40%	30%	10%



其中A表示不及格、B表示及格、c表示中等、D表示良好、E表示优秀

6.6 赫夫曼树及赫夫曼编码

相关概念

1. 结点间的路径长度----从树中一个结点到另一个结点之间的分支数目

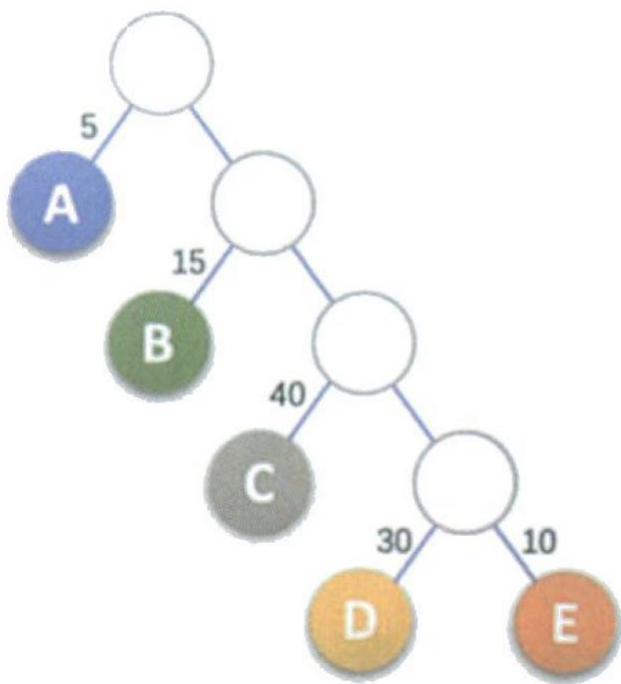
2. 树的路径长度----从根节点到树所有节点长度总和。

n个结点的二叉树的路径长度不小于下述数列前n项的和，即

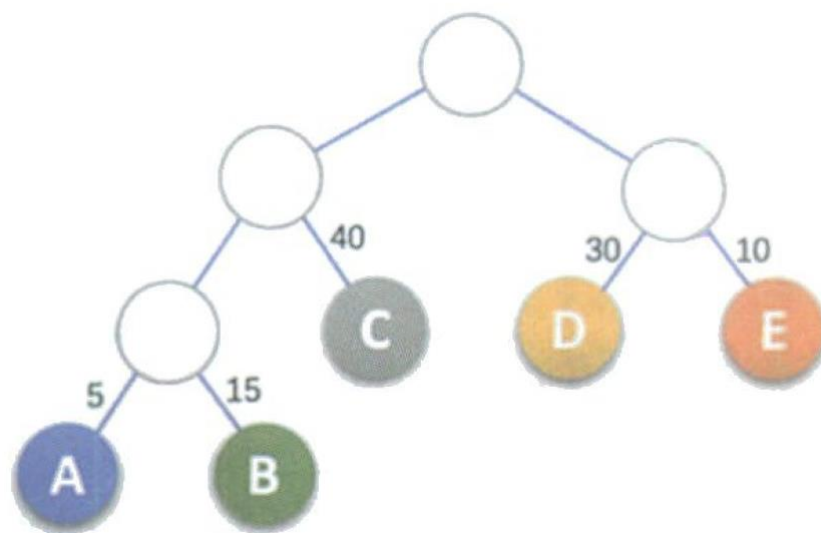
$$PL = 0 + 1 + 1 + 2 + 2 + 2 + 2 + 3 + 3 + \cdots = \sum_{i=1}^n (i-1)2^{i-1}$$

其路径长度最小者为 $PL = \sum_{i=1}^n (i-1)2^{i-1}$

6.6 赫夫曼树及赫夫曼编码



$$1+1+2+2+3+3+4+4=20$$



$$1+2+3+3+2+1+2+2=16$$


6.6 赫夫曼树及赫夫曼编码

相关概念

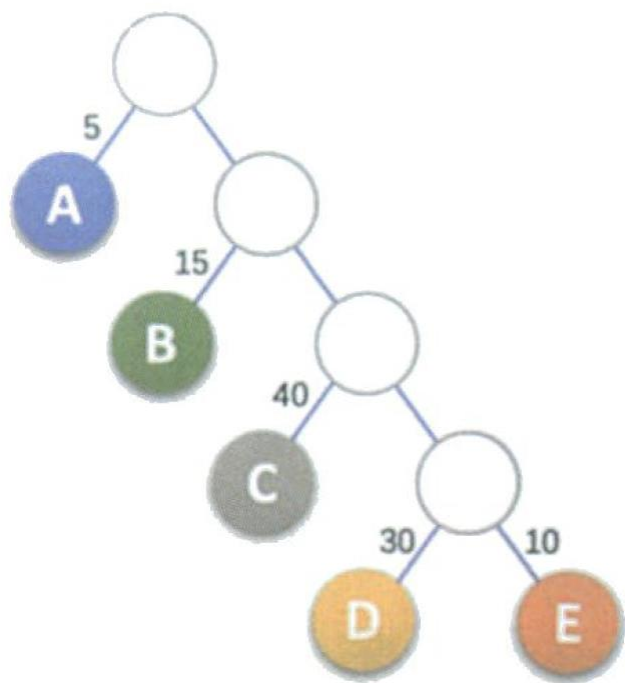
3.叶子结点的权值----对叶子结点赋予的一个有意义的数值量。

4.结点带权的路径长度（WPL）-----从该结点到树根之间的路径长度与结点上权的乘积。

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

 从根结点到第 k 个叶子的路径长度
第 k 个叶子的权值；

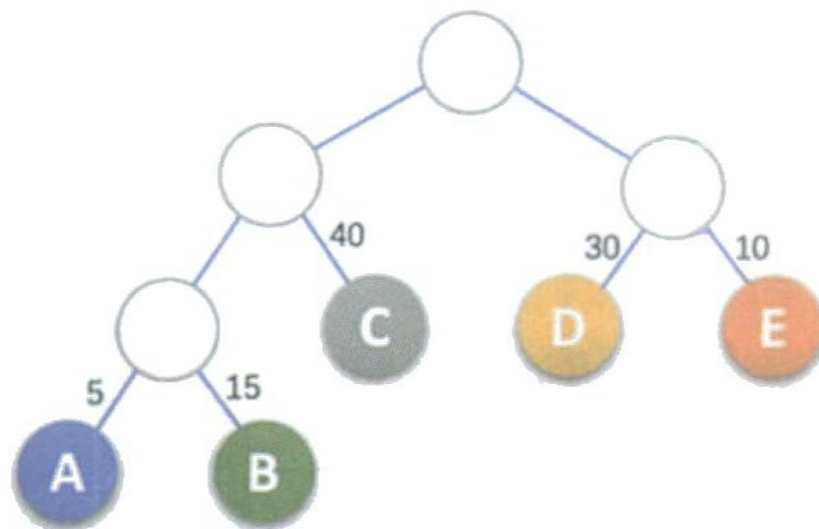
6.6 赫夫曼树及赫夫曼编码



$$1+1+2+2+3+3+4+4=20$$

WPL=

$$5*1+15*2+40*3+30*4+10*4=315$$



$$1+2+3+3+2+1+2+2=16$$

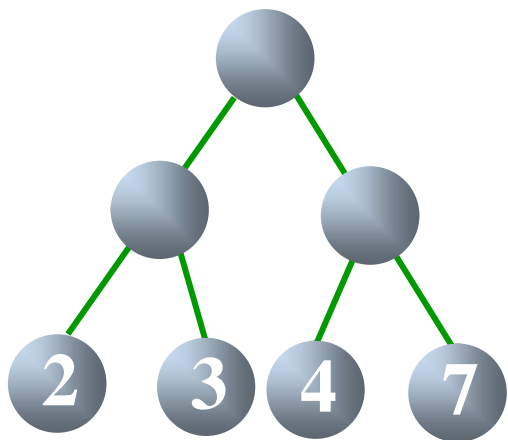
WPL=

$$5*3+15*3+40*2+30*2+10*2=220$$

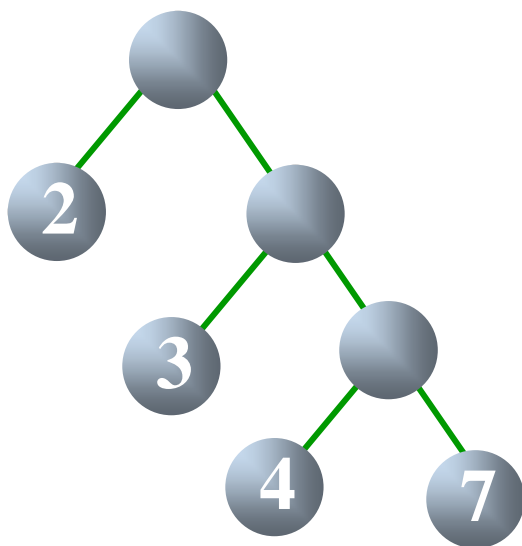
6.6 赫夫曼树及赫夫曼编码

赫夫曼树：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。

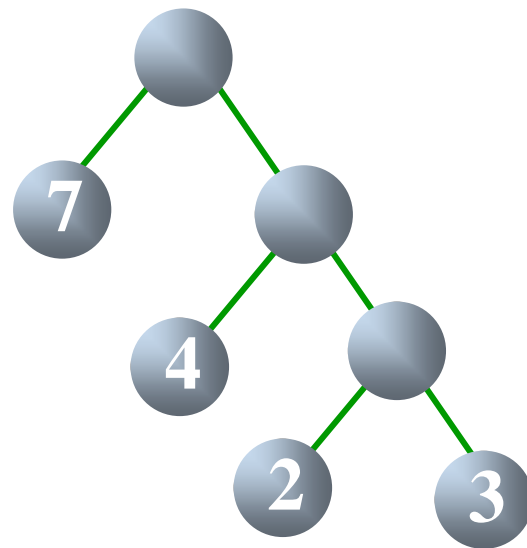
例：给定4个叶子结点，其权值分别为{2, 3, 4, 7}，可以构造出形状不同的多个二叉树。



WPL=32



WPL=41

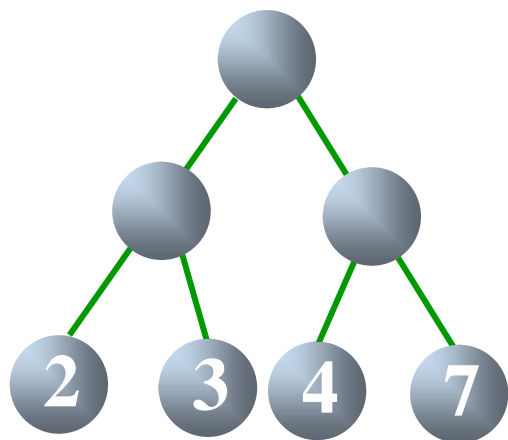


WPL=30

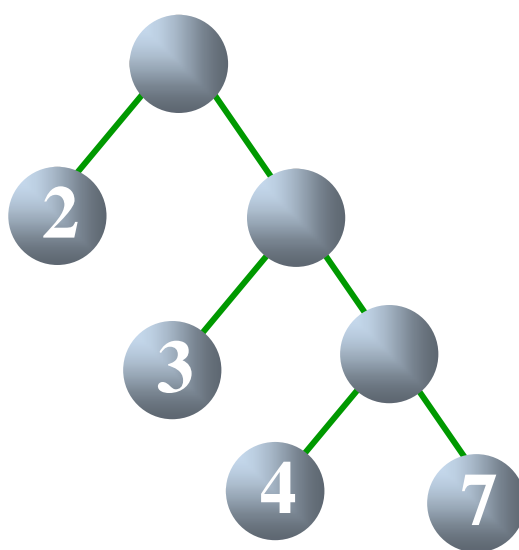
6.6 赫夫曼树及赫夫曼编码

赫夫曼树的特点：

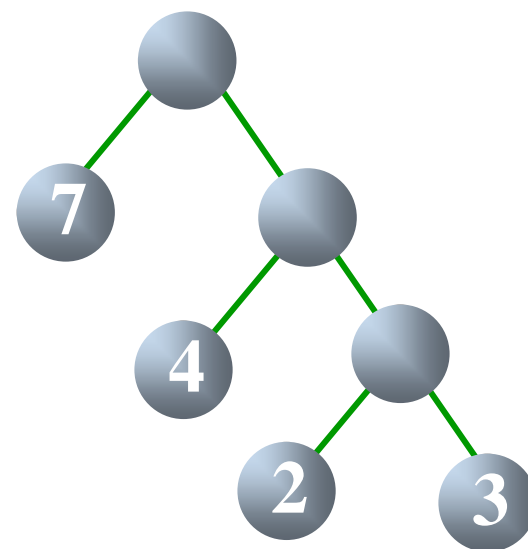
1. 权值越大的叶子结点越靠近根结点，而权值越小的叶子结点越远离根结点。
2. 只有度为0（叶子结点）和度为2（分支结点）的结点，**不存在度为1的结点**。



WPL=32



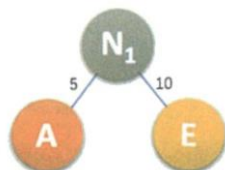
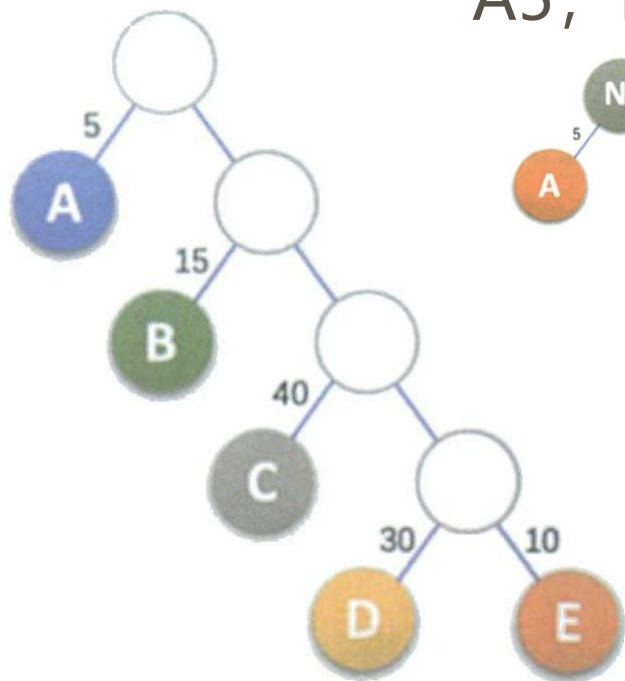
WPL=41



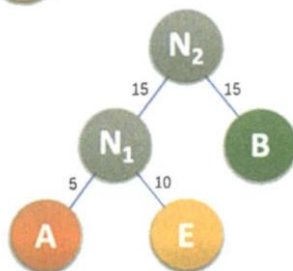
WPL=30

6.6 赫夫曼树及赫夫曼编码

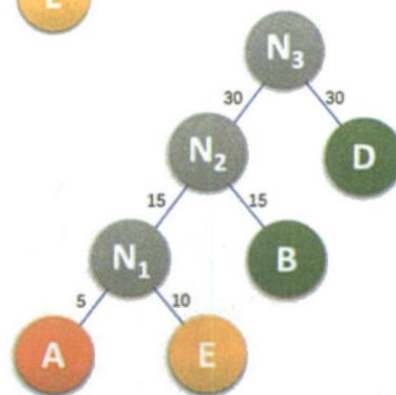
A5, E10, B15, D30, C40



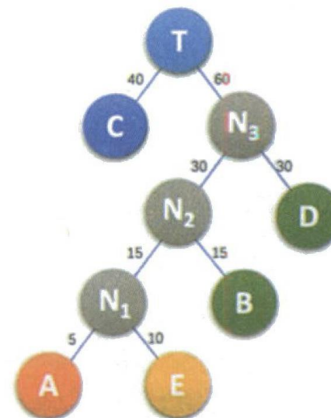
N_1 15, B15, D30, C40



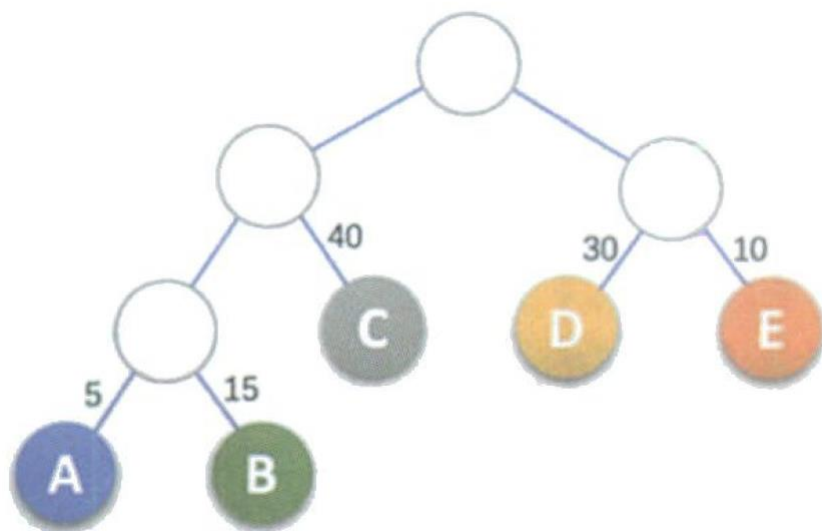
N_2 30, D30, C40



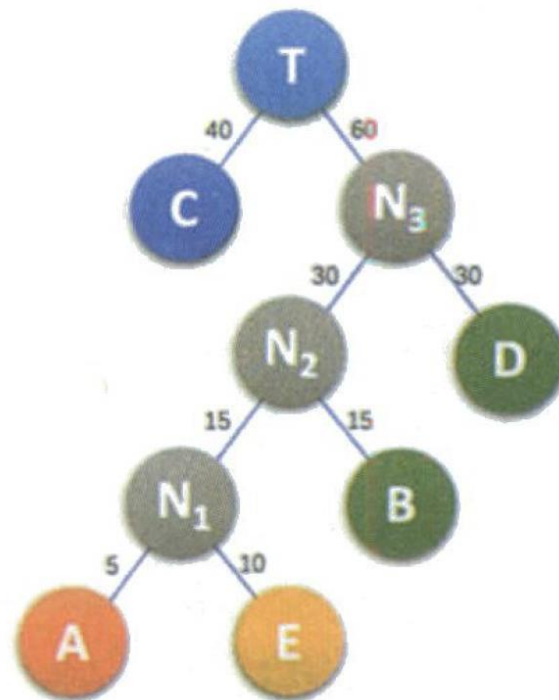
N_3 60, C40



6.6 赫夫曼树及赫夫曼编码



$$\begin{aligned} \text{WPL} &= 5 \times 3 + 15 \times 3 + 40 \times 2 + 30 \times 2 + 10 \times 2 \\ &= 220 \end{aligned}$$



$$\begin{aligned} \text{WPL} &= 40 \times 1 + 30 \times 2 + 15 \times 3 + 10 \times 4 + 5 \times 4 \\ &= 205 \end{aligned}$$

6.6 赫夫曼树及赫夫曼编码

证明 n_0 个叶子结点的哈夫曼树共有 $2n_0-1$ 个结点。

证明：设度为1和2的结点个数分别为 n_1 和 n_2 ，二叉树结点总数为 n ，则有： $n=n_0+n_1+n_2$

根据二叉树的性质知： $n_0=n_2+1$

此外，由哈夫曼树的构造原理可知：哈夫曼树不存在度为1的结点，即 $n_1=0$ ；所以由①②可得：

$$n=n_0+0+n_2=n_0+n_0-1=2n_0-1$$

6.6 赫夫曼树及赫夫曼编码

赫夫曼算法基本思想：

- (1) **初始化**：由给定的 n 个权值 $\{w_1, w_2, \dots, w_n\}$ 构造 n 棵只有一个根结点的二叉树，从而得到一个二叉树集合 $F=\{T_1, T_2, \dots, T_n\}$ ；
- (2) **选取与合并**：在 F 中选取根结点的权值**最小**的两棵二叉树分别作为左、右子树构造一棵新的二叉树，这棵新二叉树的根结点的权值为其左、右子树根结点的权值之和；
- (3) **删除与加入**：在 F 中删除作为左、右子树的两棵二叉树，并将新建立的二叉树加入到 F 中；
- (4) **重复**(2)、(3)两步，当集合 F 中只剩下一棵二叉树时，这棵二叉树便是赫夫曼树。

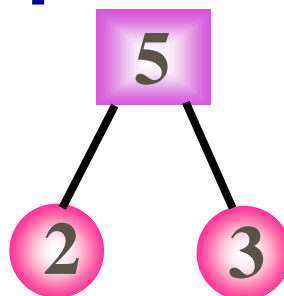
6.6 赫夫曼树及赫夫曼编码

$W=\{2, 3, 4, 5\}$ 赫夫曼树的构造过程

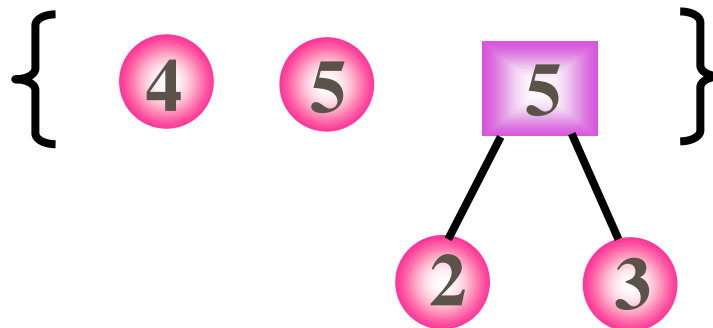
第1步：初始化



第2步：选取与合并



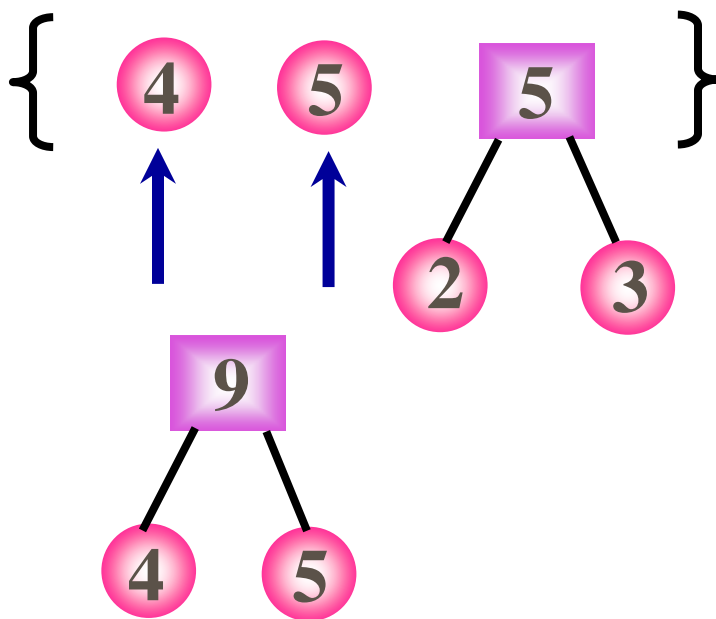
第3步：删除与加入



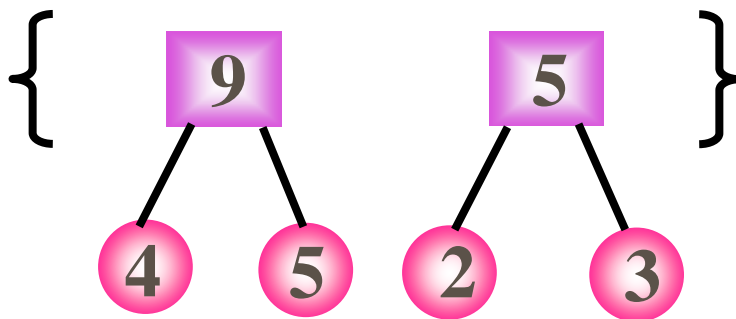
6.6 赫夫曼树及赫夫曼编码

$W=\{2, 3, 4, 5\}$ 赫夫曼树的构造过程

重复第2步



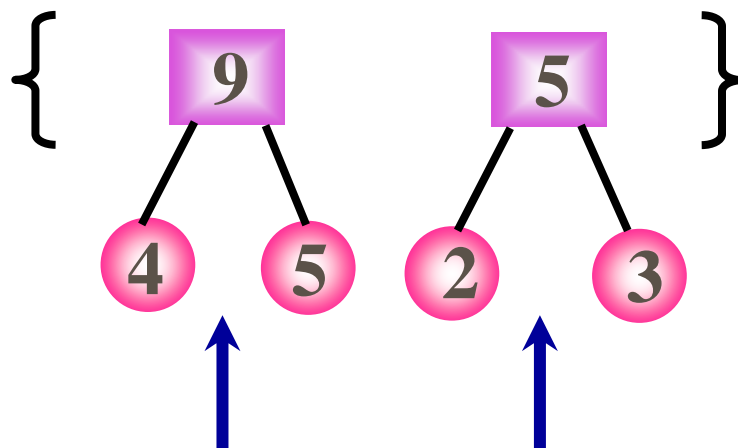
重复第3步



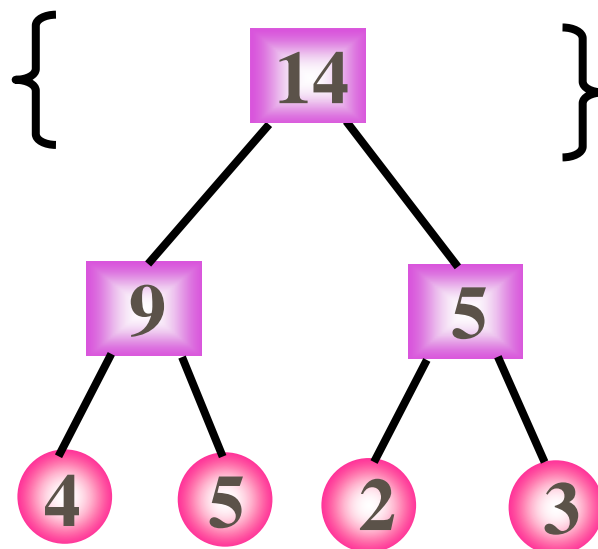
6.6 赫夫曼树及赫夫曼编码

$W=\{2, 3, 4, 5\}$ 赫夫曼树的构造过程

重复第2步



重复第3步



6.6 赫夫曼树及赫夫曼编码

赫夫曼算法的存储结构

1. 设置一个数组huffTree[2n-1]保存赫夫曼树中各点的信息，数组元素的结点结构。

weight	lchild	rchild	parent
--------	--------	--------	--------

```
struct element
{
    int weight;
    int lchild, rchild, parent;
};
```

其中：weight：权值域，保存该结点的权值；

lchild：指针域，结点的左孩子结点在数组中的下标；

rchild：指针域，结点的右孩子结点在数组中的下标；

parent：指针域，该结点的双亲结点在数组中的下标。

6.6 赫夫曼树及赫夫曼编码

伪代码

1. 数组huffTree初始化，所有元素结点的双亲、左右孩子都置为-1；
2. 数组huffTree的前n个元素的权值置给定值w[n]；
3. 进行n-1次合并
 - 3.1 在二叉树集合中选取两个权值最小的根结点，其下标分别为 i_1, i_2 ；
 - 3.2 将二叉树 i_1 、 i_2 合并为一棵新的二叉树k；

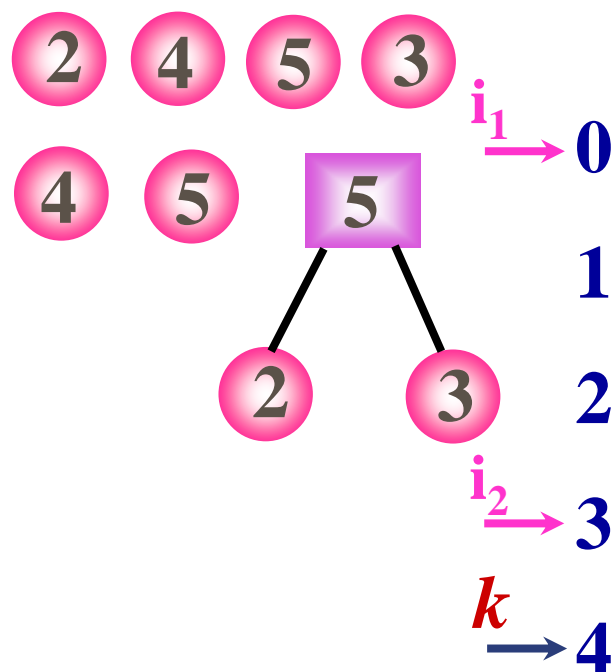
6.6 赫夫曼树及赫夫曼编码

2 4 5 3

	weight	parent	lchild	rchild
0	2	-1	-1	-1
1	4	-1	-1	-1
2	5	-1	-1	-1
3	3	-1	-1	-1
4		-1	-1	-1
5		-1	-1	-1
6		-1	-1	-1

初 态

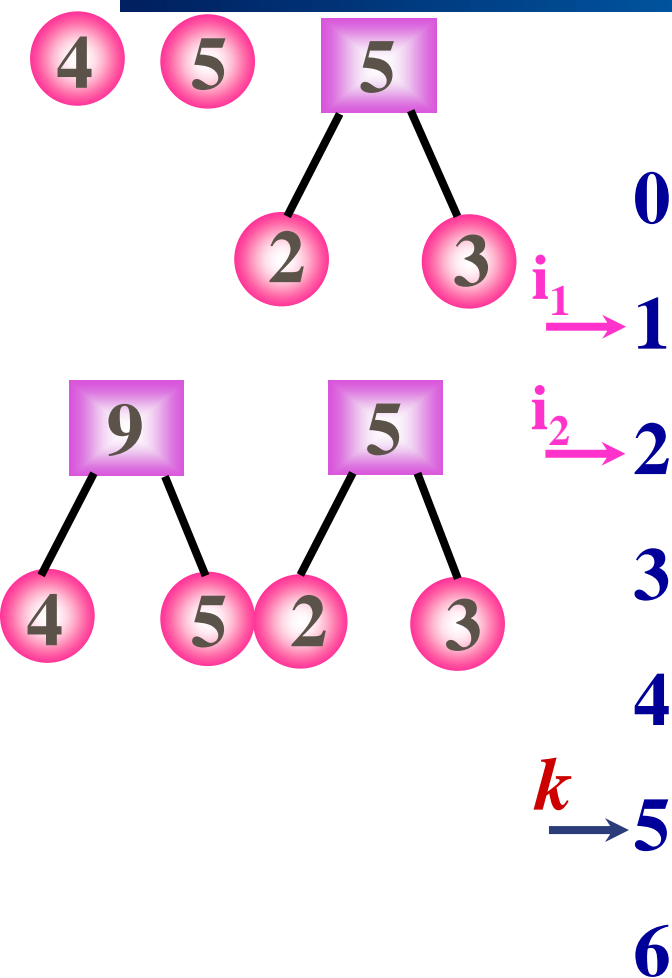
6.6 赫夫曼树及赫夫曼编码



	weight	parent	lchild	rchild
0	2	4-1	-1	-1
1	4	-1	-1	-1
2	5	-1	-1	-1
3	3	4-1	-1	-1
4	5	-1	0-1	-3
5		-1	-1	-1
6		-1	-1	-1

过程

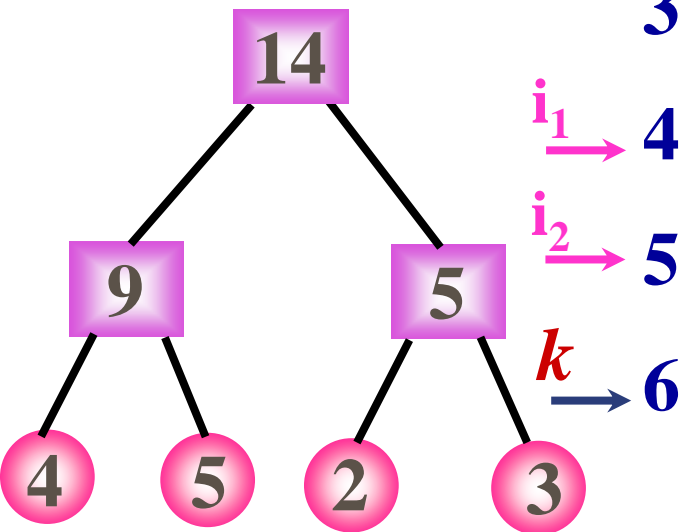
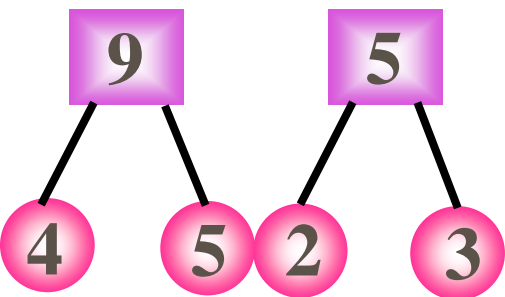
6.6 赫夫曼树及赫夫曼编码



	weight	parent	lchild	rchild
0	2	4-1	-1	-1
1	4	5-1	-1	-1
2	5	5-1	-1	-1
3	3	4-1	-1	-1
4	5	-1	0-1	-3
5	9	-1	1-1	-2
6		-1	-1	-1

过程

6.6 赫夫曼树及赫夫曼编码



weight parent lchild rchild

0	2	4-1	-1	-1
1	4	5-1	-1	-1
2	5	5-1	-1	-1
3	3	4-1	-1	-1
4	5	6-1	0-1	-3
5	9	6-1	1-1	-2
6	14	-1	41	-5

过程

6.6 赫夫曼树及赫夫曼编码

```
void HuffmanTree (element huffTree[ ], int w[ ], int n ) {  
    for (i=0; i<2*n-1; i++) {  
        huffTree [i].parent= -1;  
        huffTree [i].lchild= -1;  
        huffTree [i].rchild= -1;  
    }  
    for (i=0; i<n; i++)  
        huffTree [i].weight=w[i];  
    for (k=n; k<2*n-1; k++) {  
        Select (huffTree, i1, i2);  
        huffTree[k].weight=huffTree[i1].weight+huffTree[i2].weight;  
        huffTree[i1].parent=k;  
        huffTree[i2].parent=k;  
        huffTree[k].lchild=i1;  
        huffTree[k].rchild=i2;  
    }  
}
```


6.6 赫夫曼树及赫夫曼编码

赫夫曼树应用——赫夫曼编码

主要用途是实现数据压缩。

设给出一段报文：

CAST CAST SAT AT A TASA

字符集合是 $\{C, A, S, T\}$ ，各个字符出现的频度(次数)是 $W = \{2, 7, 4, 5\}$ 。

若给每个字符以等长编码

A : 00 T : 10 C : 01 S : 11

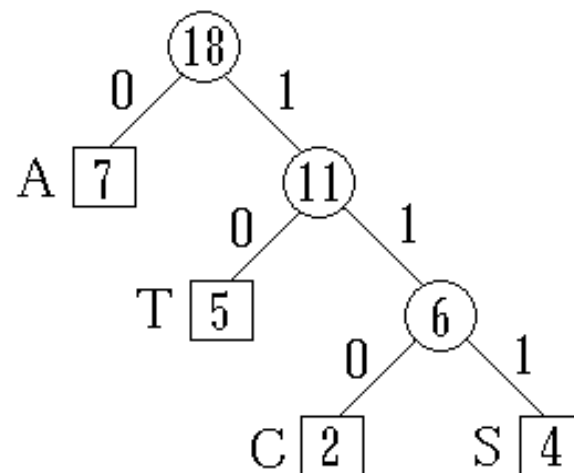
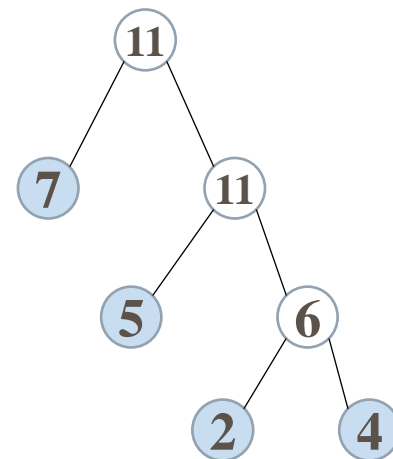
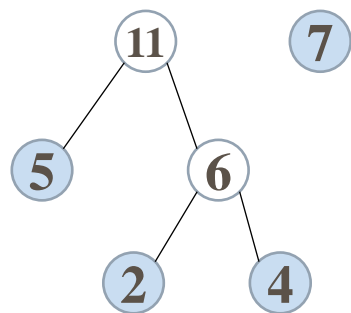
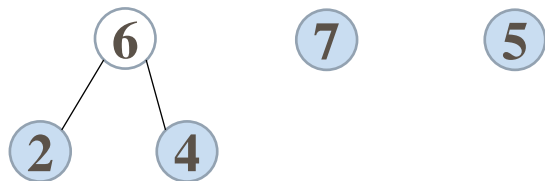
则总编码长度为 $(2+7+4+5) * 2 = 36$ 。

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。

因各字符出现的概率为 $\{2/18, 7/18, 4/18, 5/18\}$ 。

6.6 赫夫曼树及赫夫曼编码

C A S T
2 7 4 5



6.6 赫夫曼树及赫夫曼编码

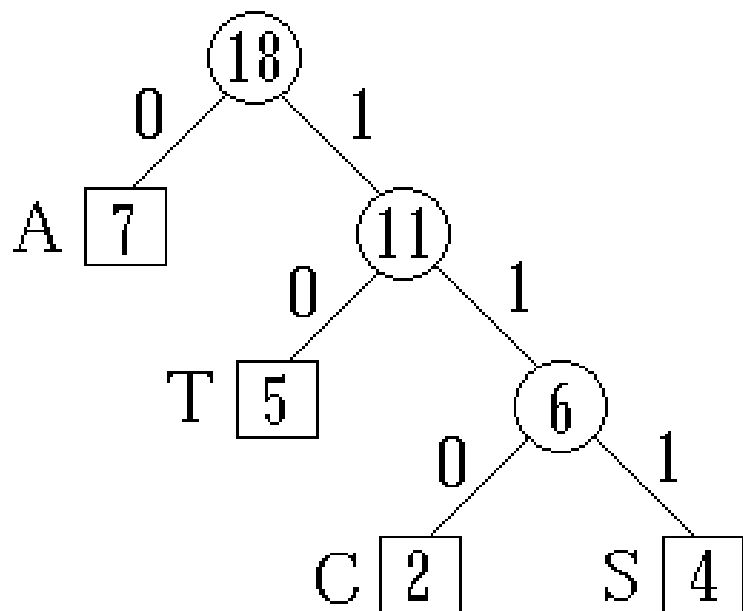
化整为 $\{ 2, 7, 4, 5 \}$ ，以它们为各叶结点上的权值，建立赫夫曼树。左分支赋 0，右分支赋 1，得赫夫曼编码(变长编码)。

A : 0 T : 10 C : 110 S : 111

它的总编码长度： $7*1+5*2+(2+4)*3 = 35$ 。比等长编码的情形要短。

总编码长度正好等于
赫夫曼树的带权路径长
度WPL。

赫夫曼编码是一种无
前缀的编码。解码时不会
混淆。



6.6 赫夫曼树及赫夫曼编码

```
typedef struct{
    unsigned int weight;
    unsigned int parent, lchild, rchild;
} HTNode, *HuffmanTree;
typedef char **HuffmanCode;

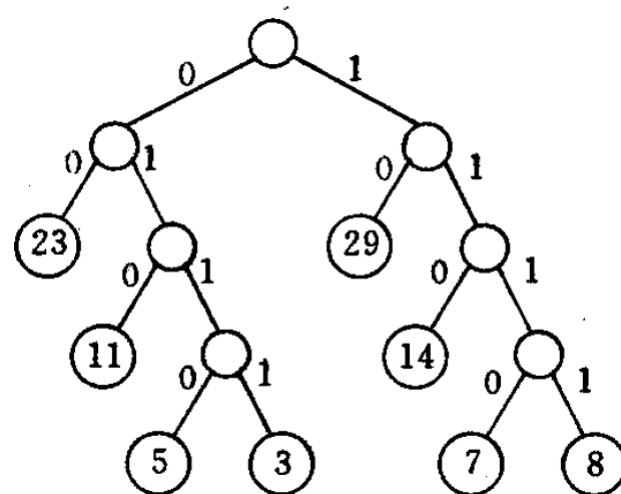
void HuffmanCoding(HuffmanTree &HT, HuffmanCode &HC, int *w, int n){
    HuffmanTree p; char *cd;
    int i, s1, s2, start; unsigned int c, f;
    if (n <= 1) return; // n为字符数目, m为结点数
    int m = 2 * n - 1;
    HT = (HuffmanTree)malloc((m + 1) * sizeof(HTNode)); // 0号单元未用
    for (p = HT, i = 1; i <= n; ++i, ++p, ++w) {
        p->weight = *w; p->parent = 0; p->lchild = 0; p->rchild = 0;
    }
    // *W={5,29,7,8,14,23,3,11}
    // *p = { *w,0,0,0 };
    for (; i <= m; ++i, ++p) {
        p->weight = 0; p->parent = 0; p->lchild = 0; p->rchild = 0;
    }
    // *p={ 0,0,0,0 };
}
```

6.6 赫夫曼树及赫夫曼编码

```
for (i = n + 1; i <= m; ++i){ // 建赫夫曼树
    Select(HT, i - 1, s1, s2);
    HT[s1].parent = i; HT[s2].parent = i; HT[i].lchild = s1;
    HT[i].rchild = s2;
    HT[i].weight = HT[s1].weight + HT[s2].weight;
}
// 从叶子到根逆向求赫夫曼编码
HC = (HuffmanCode)malloc((n + 1) * sizeof(char *));
cd = (char *)malloc(n * sizeof(char)); //分配求编码的工作空间
cd[n - 1] = '\0';
for (i = 1; i <= n; ++i){
    start = n - 1; //编码结束符位置
    for (c = i, f = HT[c].parent; f != 0; c = f, f = HT[f].parent)
        if (HT[f].lchild == c) cd[--start] = '0';
        else cd[--start] = '1';
    HC[i] = (char *)malloc((n - start) * sizeof(char));
    strcpy(HC[i], &cd[start]);
    printf("%s\n", HC[i]);
}
free(cd);
}
```

6.6 赫夫曼树及赫夫曼编码

已知某系统在通信联络中只可能出现8种字符,其概率分别为
0.05,0.29,0.07,0.08,0.14,0.23,0.03,
0.11,试设计赫夫曼编码。



	weight	parent	lchild	rchild
1	5	0	0	0
2	29	0	0	0
3	7	0	0	0
4	8	0	0	0
5	14	0	0	0
6	23	0	0	0
7	3	0	0	0
8	11	0	0	0
9	-	0	0	0
10	-	0	0	0
11	-	0	0	0
12	-	0	0	0
13	-	0	0	0
14	-	0	0	0
15	-	0	0	0

	HT			
	weight	parent	lchild	rchild
1	5	9	0	0
2	29	14	0	0
3	7	10	0	0
4	8	10	0	0
5	14	12	0	0
6	23	13	0	0
7	3	9	0	0
8	11	11	0	0
9	8	11	1	7
10	15	12	3	4
11	19	13	8	9
12	29	14	5	10
13	42	15	6	11
14	58	15	2	12
15	100	0	13	14

HC

Line	Value
1	0 1 1 0
2	1 0
3	1 1 1 0
4	1 1 1 1
5	1 1 0
6	0 0
7	0 1 1 1
8	0 1 0

6.6 赫夫曼树及赫夫曼编码

赫夫曼树应用——赫夫曼编码

前缀编码：一组编码中任一编码都不是其它任何一个编码的前缀。

前缀编码保证了在解码时不会有多种可能。

6.6 赫夫曼树及赫夫曼编码

例：一组字符{A, B, C, D, E, F, G}
出现的频率分别是{9, 11, 5, 7, 8, 2, 3}，
设计最经济的编码方案。

编码方案：

A: 00

B: 10

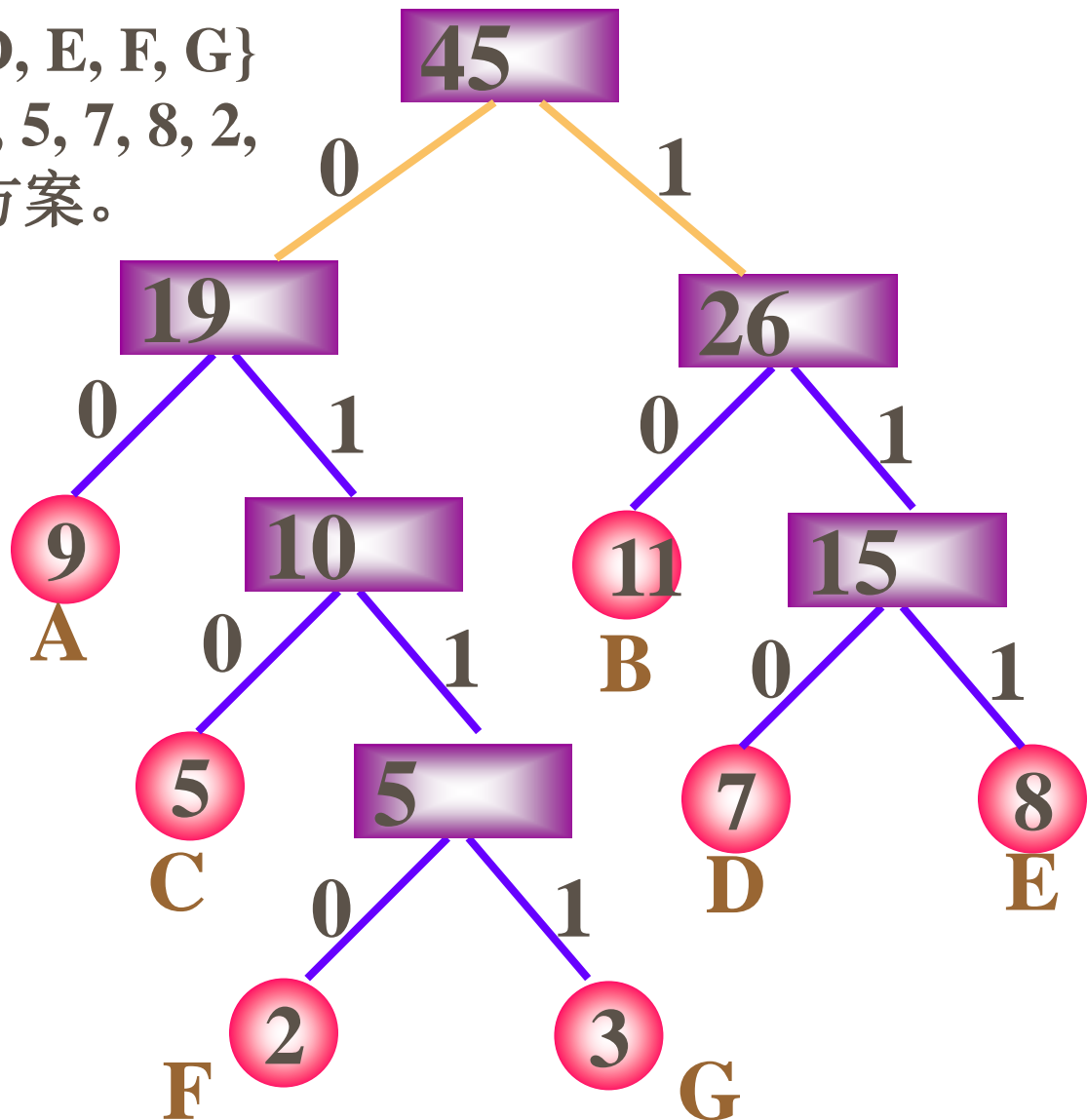
C: 010

D: 110

E: 111

F: 0110

G: 0111



6.6 赫夫曼树及赫夫曼编码

例 假设用于通信的电文仅由8个字母 {a, b, c, d, e, f, g, h} 构成，它们在电文中出现的概率分别为{ 0.07, 0.19, 0.02, 0.06, 0.32, 0.03, 0.21, 0.10 }，试为这8个字母设计哈夫曼编码。如果用0~7的二进制编码方案又如何？

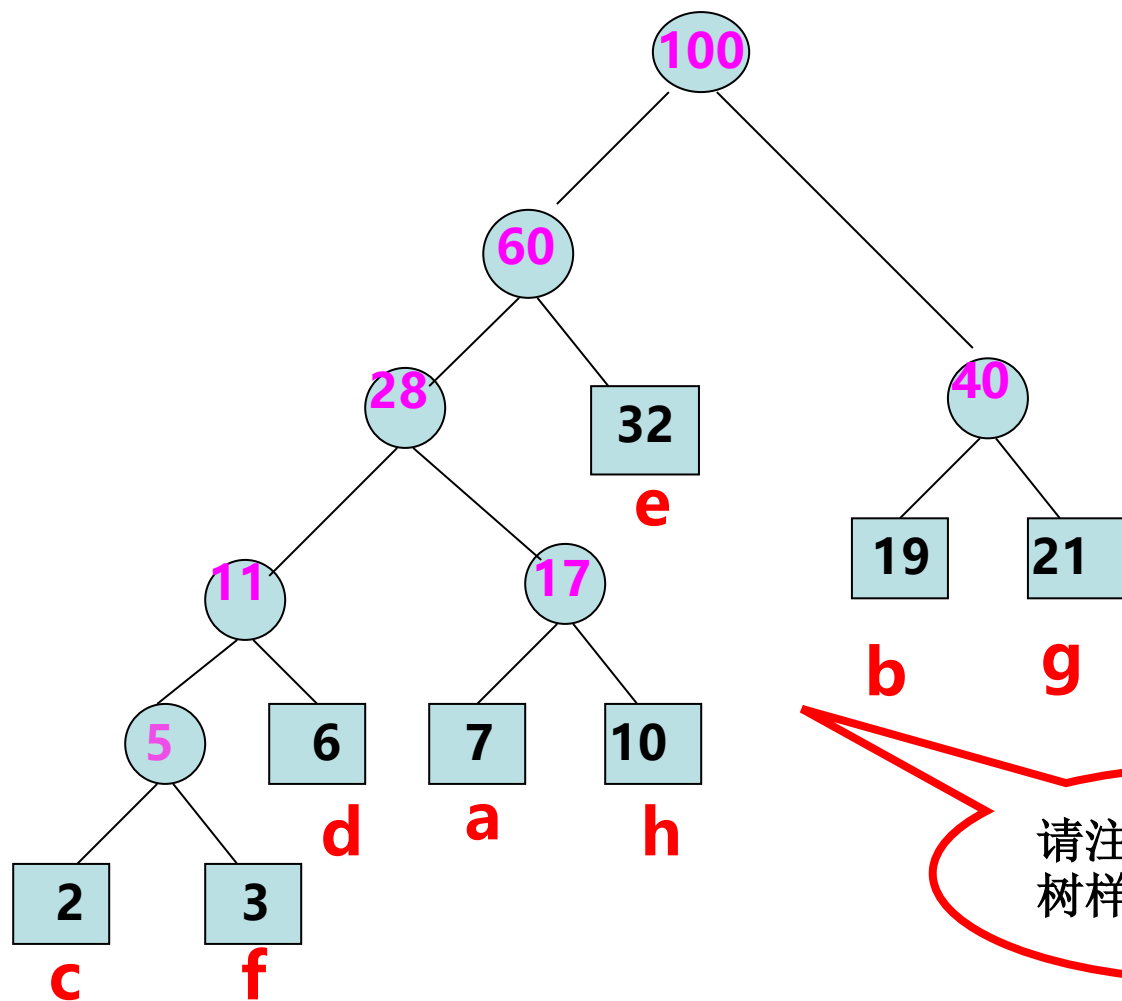
解：先将概率**放大100倍**，以方便构造哈夫曼树。

放大后的权值集合 $w = \{ 7, 19, 2, 6, 32, 3, 21, 10 \}$ ，

按哈夫曼树构造规则（**合并、删除、替换**），可得到哈夫曼树。

6.6 赫夫曼树及赫夫曼编码

$w=\{ 7, 19, 2, 6, 32, 3, 21, 10 \}$ 在机内存储形式为:

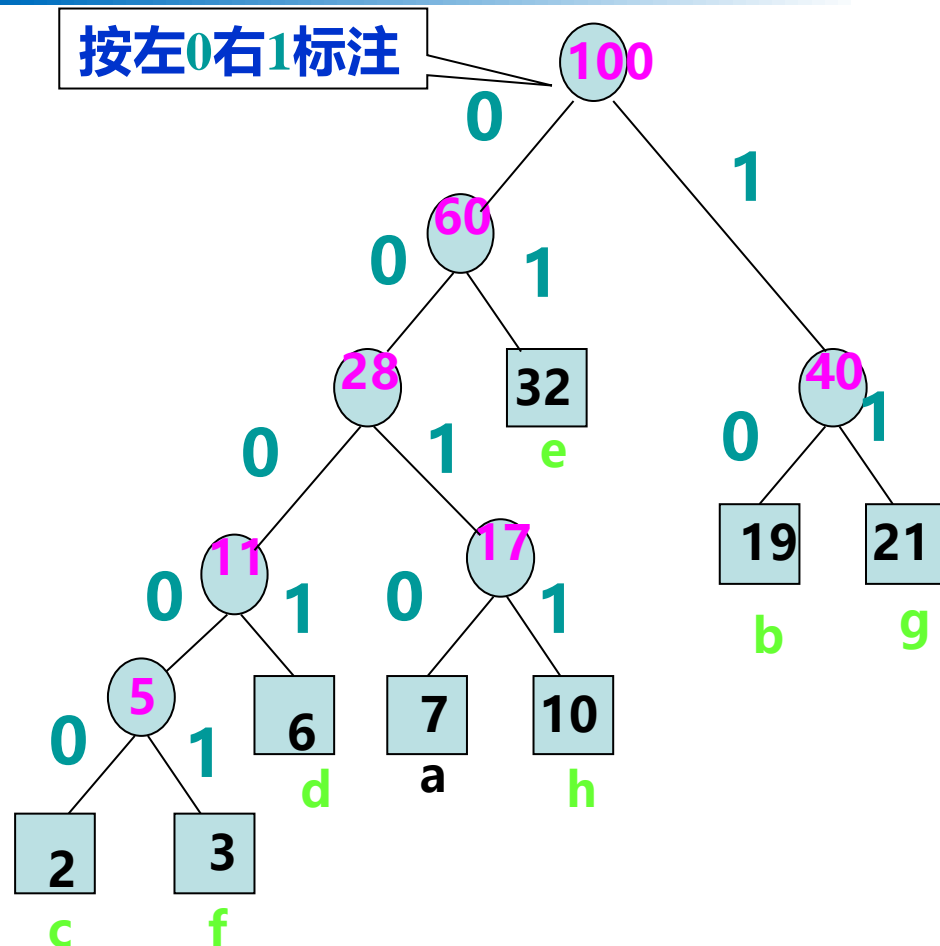


请注意：赫夫曼树样式不惟一！

6.6 赫夫曼树及赫夫曼编码

对应赫夫曼编码:

符	编码	频率	符	编码	频率
a	0010	0.07	a	000	0.07
b	10	0.19	b	001	0.19
c	00000	0.02	c	010	0.02
d	0001	0.06	d	011	0.06
e	01	0.32	e	100	0.32
f	00001	0.03	f	101	0.03
g	11	0.21	g	110	0.21
h	0011	0.10	h	111	0.10

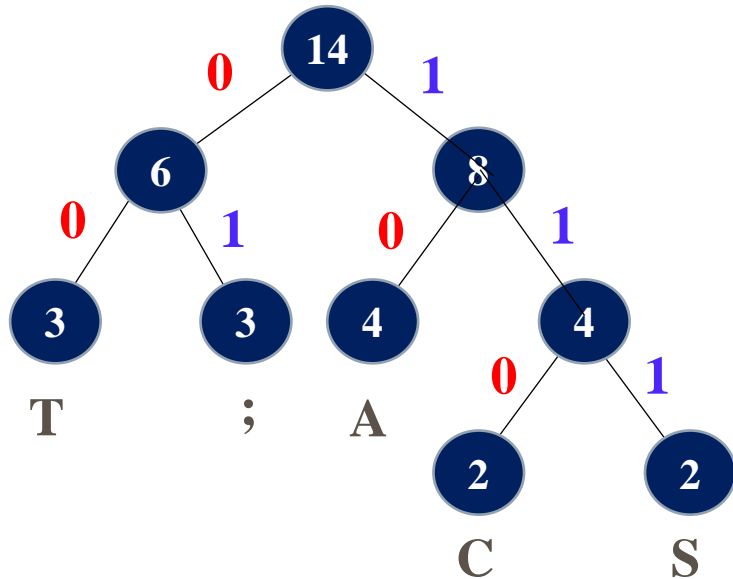


Huffman码的WPL = $2(0.19+0.32+0.21) + 4(0.07+0.06+0.10) + 5(0.02+0.03)$
= $1.44+0.92+0.25=2.61$

二进制等长码的WPL = $3(0.19+0.32+0.21+0.07+0.06+0.10+0.02+0.03)=3$

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

(1) 每个字符的赫夫曼编码



T	00
;	01
A	10
C	110
S	111

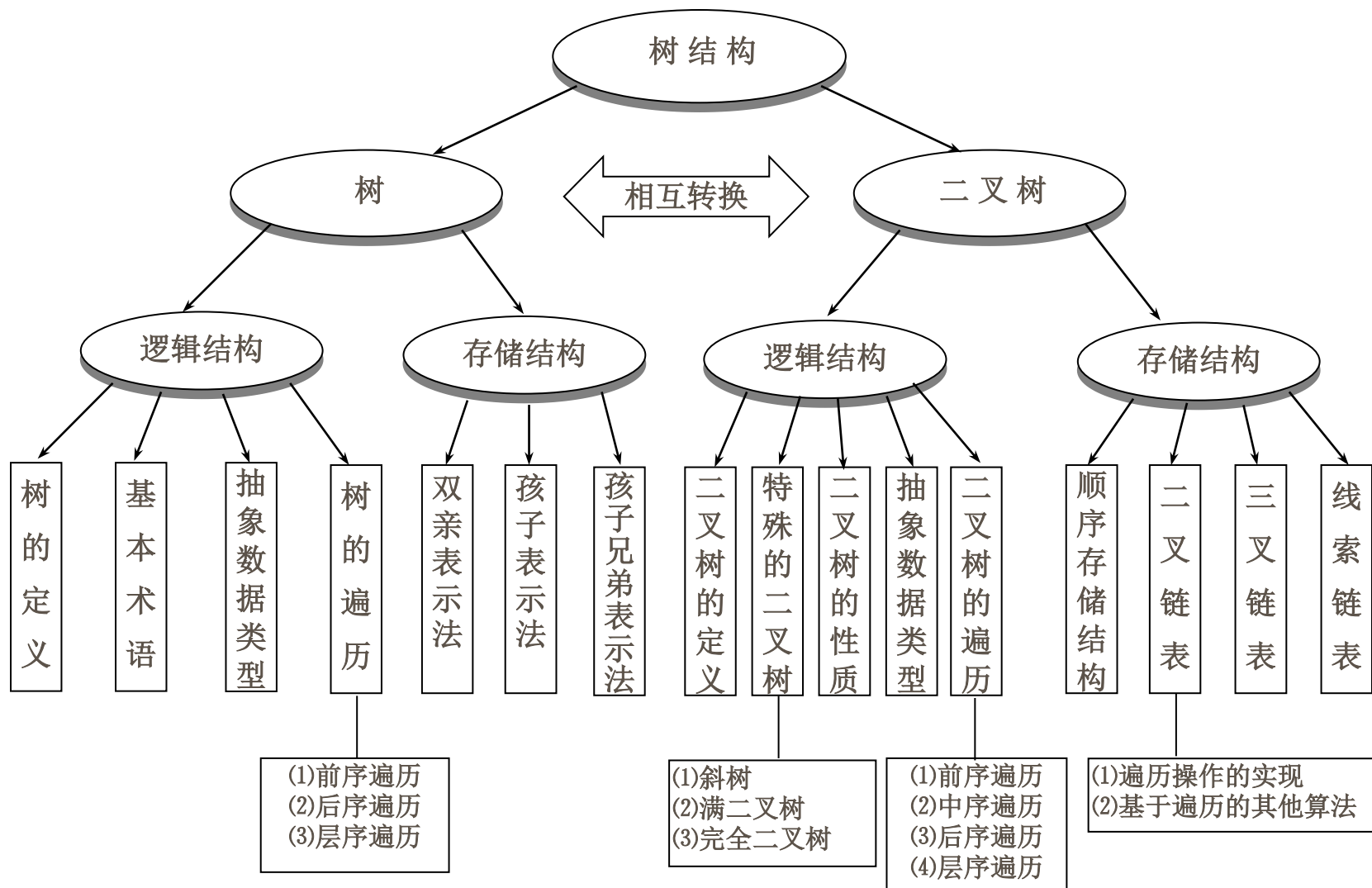
(2) 电文 {CAS;CAT;SAT;AT} 的编码是

11010111011101000011111000011000

(3) 还原下述编码{1111000011101011101001111100111011010}

SAT;CAS;TSC;CCA

小结



小结

1、树的概念

结点的度：结点的子节点个数

树的度：结点的最大度数

路径长度：路经过边的个数 用 l 表示

树的路径长度： $\sum_{i=1}^n l_i$ (l_i 是第 i 个结点的路径长度)

有序树：若树中各结点的子 $\sum_{i=1}^n w_i \times l_i$ 的次序从左向右安排的，且相对次序不能随意变换，则称为有序树

无序树：节点没有左右之分

小结

2、树的基本性质

PS: $N_{\text{总}}$ —— 树的总结点个数
 N_i —— 度为 i 的结点个数
 B —— 树的分支数，即边数

(1) ★最重要的公式，联立起来求解方程组，可以解决大部分选择题

$$\textcircled{1} N_{\text{总}} = N_0 + N_1 + N_2 + \dots + N_m$$

$$\textcircled{2} B = (0 \times N_0) + (1 \times N_1) + (2 \times N_2) + \dots + (m \times N_m)$$

$$\textcircled{3} N_{\text{总}} = B + 1$$

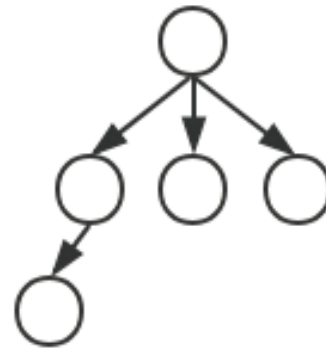
例如二叉树的一个性质： $N_0 = N_2 + 1$

就是联立的3个方程组，最后把 N_1 消掉了得出的结论

小结

2、树的基本性质

- (2) 度为 m 的树中第 i 层至多有 m^{i-1} 个结点
- (3) 深度为 h 的 m 叉树至多有 $m^h - 1$ 个结点（满 m 叉树时，等比数列求和）
- (4) 具有 n 个结点的 m 叉树的最小高度为 $\lceil \log_m(n(m-1)+1) \rceil$
- (5) 树的结点总个数确定下，“完全” m 叉树时，高度最小
- (6) 树的结点总个数确定下，单边树，可以使高度最大



“完全” m 叉树



小结

3、二叉树的概念（至多只有两颗子树，有左右之分）

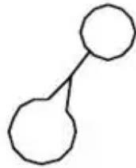
（1）5种形态：



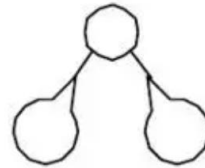
(a) 空二叉树



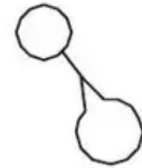
(b) 只有根结点的
二叉树



(c) 只有左子树的
二叉树

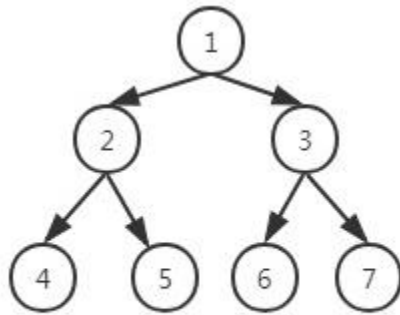


(d) 左右子树均非
空的二叉树

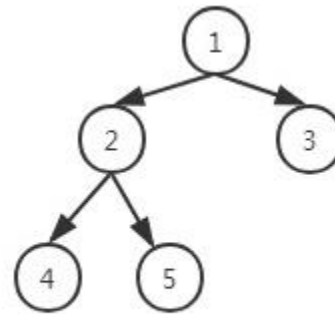


(e) 只有右子树的
二叉树

（2）满二叉树（特殊的完全二叉树） 和 完全二叉树



满二叉树



完全二叉树

小结

4、二叉树的几个重要性质（对比前面第二大点树度为2的情况）

- (1) 性质1：在二叉树的第 i 层上至多有 2^{i-1} 个结点 ($i \geq 1$)
- (2) 性质2：深度为 k 的二叉树上至多含 $2^k - 1$ 个结点 ($k \geq 1$)
- (3) 性质3：对任何一棵二叉树，若它含有 n_0 个叶子结点（0度节点）、 n_2 个度为 2 的结点，则必存在关系式：

$$n_0 = n_2 + 1$$

- (4) 性质4：具有 n 个 ($n > 0$) 结点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor + 1$

小结

4、二叉树的几个重要性质（对比前面第二大点树度为2的情况）

(5) 性质5：若对含 n 个结点的完全二叉树从上到下且从左至右进行1至 n 的编号，则对完全二叉树中任意一个编号为 i 的结点：

- 1) 若 $i=1$ ，则该结点是二叉树的根，无双亲，
否则，编号为 $\lfloor i/2 \rfloor$ 的结点为其双亲结点；
- 2) 若 $2i > n$ ，则该结点无左孩子，
否则，编号为 $2i$ 的结点为其左孩子结点；
- 3) 若 $2i+1 > n$ ，则该结点无右孩子结点，
否则，编号为 $2i+1$ 的结点为其右孩子结点；
- 4) 若 i 为偶数，且 $i \neq n$ ，则其右兄弟为 $i+1$ 若 i 为奇数，
且 $i \neq 1$ ，则其左兄弟为 $i-1$
- 5) i 所在层次为 $\lfloor \log_2 i \rfloor + 1$ ， $i > \lfloor n/2 \rfloor$ 是叶子节点。

小结

5、二叉树的存储结构

(1) 顺序存储结构：（数组）下标从1开始，左孩子 $2i$ ，右孩子 $2i+1$
适合存放完全二叉树和满二叉树（这样不会浪费空间）

(2) 链式存储结构：

（至少包含3个域 左孩子lchild，数据域data，右孩子rchild）

```
typedef struct BiTNode {  
    ElemType data;  
    struct BiTNode *lchild, *rchild;  
}BiTNode, *BiTree;
```

重要结论：含有 n 个结点的二叉链表含有 $n+1$ 个空域（记）

（可以利用这些空链域来组成线索链表）

小结

6、二叉树的遍历（二叉树的大部分考题和操作都围绕遍历）

先序遍历DLR

- (1) 访问根结点;
- (2) 先序遍历左子树;
- (3) 先序遍历右子树。

中序遍历LDR

- (1) 中序遍历左子树;
- (2) 访问根结点;
- (3) 中序遍历右子树。

后序遍历LRD

- (1) 后序遍历左子树;
- (2) 后序遍历右子树;
- (3) 访问根结点。

小结

6、二叉树的遍历（二叉树的大部分考题和操作都围绕遍历）

遍历的实质：怎么把一个二维的结构变成一个一维的序列

(1) 递归写法（打印操作在I II III的位置上为不同的遍历）

```
void Order(BiTree T) {  
    if(T != NULL) {  
        // I 先序遍历  
        Order(T->lchild);  
        // II 中序遍历  
        Order(T->rchild);  
        // III 后序遍历  
    }  
}
```

递归工作栈的深度恰好为树的深度

小结

6、二叉树的遍历（二叉树的大部分考题和操作都围绕遍历）

（2）非递归算法

- ①先序遍历（栈）：右孩子先进栈，左孩子后进栈
- ②中序遍历（栈）：如果栈顶结点左孩子存在，则左孩子入栈。若左孩子不存在，则出栈并输出栈顶，再看右孩子是否存在。若存在就把右孩子进栈
- ③后序遍历：每遇到一个结点，先把它入栈，根据访问情况设置访问标签，遍历完右子树后，结点退栈访问
- ④层次遍历（队列）：用一个队列，若队列不空，出队一个元素并输出，把其左右孩子分别入队（若不空）

小结

6、二叉树的遍历（二叉树的大部分考题和操作都围绕遍历）

（3）其他知识点

中序序列 与 （任何一个先，后，层）序列可以**唯一确定**一颗二叉树。先序和后序不行

先序的第一个结点是**根**

后序的最后一个结点是**根**

中序可以分成**两个子序列**

先：A BCDEFGHI （中A 左BC 右 DEFGHI）根据下面中序序列得出的结论

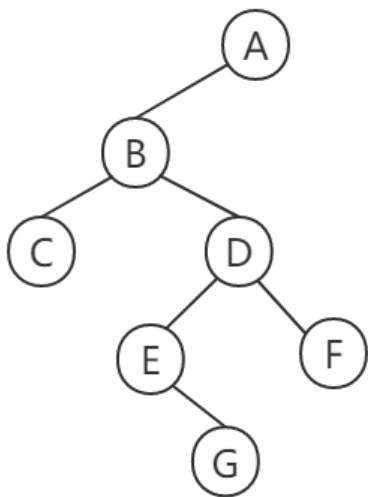
中：BC A EDGHFI （左 中 右）

小结

7、二叉树的建立（用#表示空节点）

按先序遍历序列建立二叉树的二叉链表

- ①从键盘输入二叉树的结点信息，建立二叉树的存储结构；
- ②在建立二叉树的过程中按照二叉树先序方式建立。



ABC##DE#G##F###

```
Status CreateBiTree(BiTree &T){
    scanf(&c);/-->cin>>c;
    if(c == '#') T = NULL; //递归结束，建空树
    else{                    //递归建立二叉树
        if(!(T = BiTNode*)malloc(sizeof(BiTNode)))
            exit(OVERFLOW); //T = new BiTNode;
        T->data = c; //生成根结点
        CreateBiTree(T->lchild); //构造左子树
        CreateBiTree(T->rchild); //构造右子树
    }
    return OK;
}
```

小结

8、线索二叉树：（利用好 n 个结点有 $n+1$ 个空指针）

(1) 目的：加快查找结点的前驱和后继的速度（把他们链起来）

(2) 结点结构：

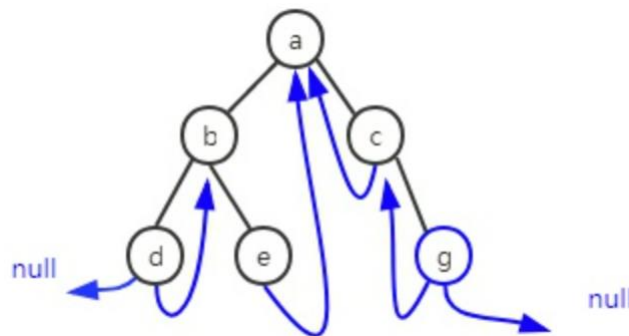
ltag	lchild	data	rchild	rtag
------	--------	------	--------	------

① ltag = 0 (左孩子) ltag = 1 (前驱结点)

rtag = 0 (右孩子) rtag = 1 (后继结点)

② 这种结点构成的二叉链表叫线索链表

③ 对二叉树以某种次序遍历使其变成为线索二叉树的过程叫线索化。有时候为了仿照线性表，线索链表上也加一个头结点。



小结

8、线索二叉树：（利用好 n 个结点有 $n+1$ 个空指针）

（3）线索二叉树的遍历：（代码了解）

①求中序线索二叉树中序序列的第一个结点

```
ThreadNode* FirstNode(ThreadNode *p) {  
    while(p->ltag == 0)  
        p = p->lchild; //最左下的结点不一定是叶子结点  
    return p;  
}
```

②求中序线索二叉树结点 p 在中序序列下的后继结点

```
ThreadNode* NextNode(ThreadNode *p) {  
    if(p->rtag == 0)  
        return FirstNode(p->rchild);  
    else  
        return p->rchild; //后继  
}
```

小结

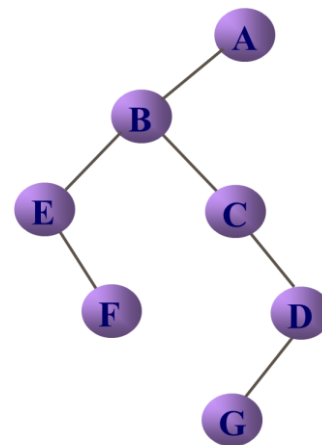
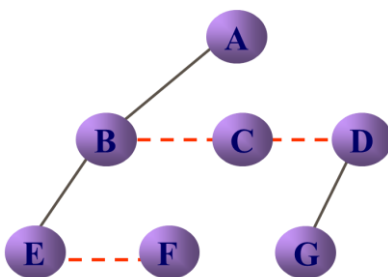
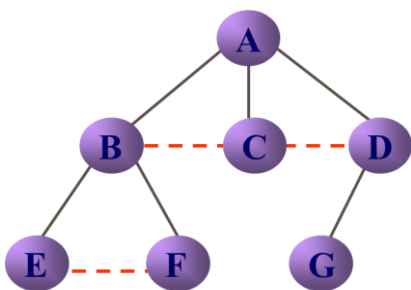
9、树、森林与二叉树转换

(1) 树转换为二叉树

①加线：在所有的兄弟结点之间加线

②去线：对树中的每一个结点，只保留它与第一个结点的连线，删除与其他孩子结点之间的连线

③旋转：调整成一颗二叉树（第一个孩子是二叉树的左孩子，兄弟转换过来是它的右孩子）



小结

9、树、森林与二叉树转换

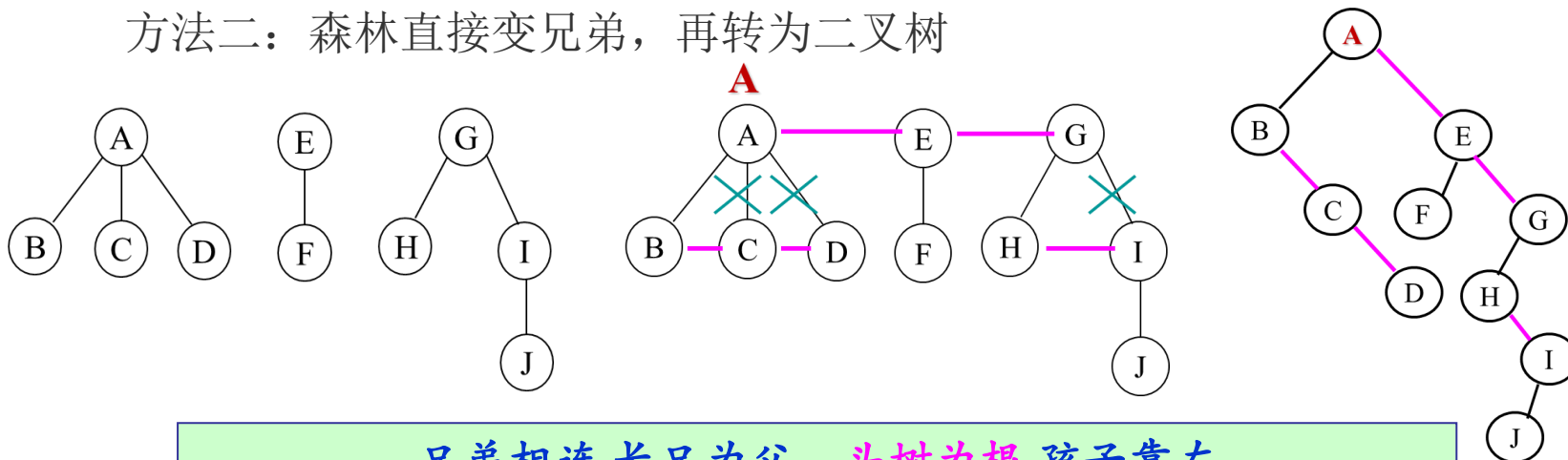
(2) 森林转换成二叉树

方法一：

①转换：把每一棵树转换成二叉树

②连接：第一棵二叉树不变，从第二棵二叉树开始，依次将后一棵二叉树的根节点作为前一棵二叉树根节点的右孩子，用线连起来

方法二：森林直接变兄弟，再转为二叉树



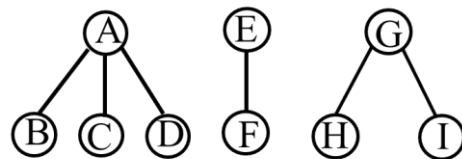
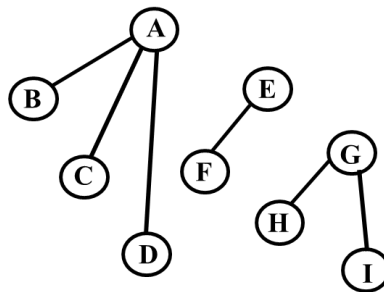
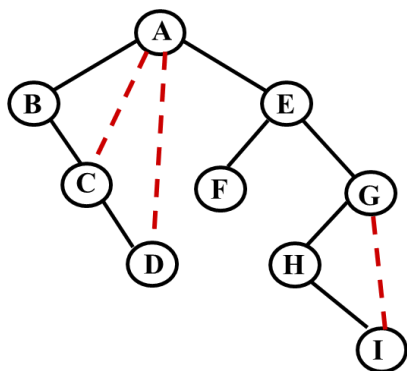
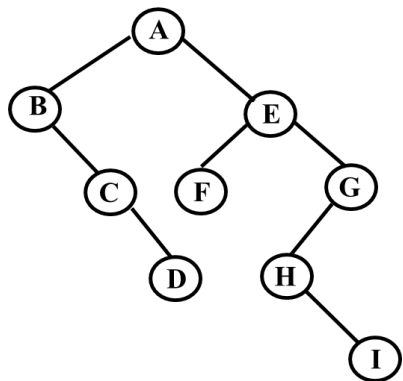
兄弟相连 长兄为父，头树为根 孩子靠左

小结

9、树、森林与二叉树转换

(3) 二叉树转换成树或森林

- ①加线——若某结点x是其双亲y的左孩子，则把结点x的右孩子、右孩子的右孩子、……，都与结点y用线连起来；
- ②去线——删去原二叉树中所有的双亲结点与右孩子结点的连线；
- ③层次调整——整理由(1)、(2)两步所得到的树或森林，使之层次分明。



小结

10、树、森林的遍历

(1) 树的遍历

树的遍历 { 深度优先遍历 (先根、后根)
 广度优先遍历 (层次)

先根遍历

- ① 访问根结点;
- ② 依次先根遍历根结点的每棵子树。

后根遍历

- ① 依次后根遍历根结点的每棵子树;
- ② 访问根结点。

- ① 树的先根遍历与二叉树的先序遍历相同;
- ② 树的后根遍历相当于二叉树的中序遍历;
- ③ 树没有中序遍历, 因为子树无左右之分。

小结

10、树、森林的遍历

(2) 森林的遍历

森林的遍历 { 深度优先遍历 (先序、中序)
 广度优先遍历 (层次)

先序遍历

- ① 若森林为空，返回；
- ② 访问森林中第一棵树的根结点；
- ③ 先序遍历第一棵树的根结点的子树森林；
- ④ 先序遍历除去第一棵树之后剩余的树构成的森林。

中序遍历

- ① 若森林为空，返回；
- ② 中序遍历森林中第一棵树的根结点的子树森林；
- ③ 访问第一棵树的根结点；
- ④ 中序遍历除去第一棵树之后剩余的树构成的森林。

小结

11、赫夫曼树

(1) 赫夫曼树相关概念

- ① 叶子结点的权值——对叶子结点赋予的一个有意义的数值量。
- ② 结点带权的路径长度（WPL）——从该结点到树根之间的路径长度与结点上权的乘积。

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

 从根结点到第 k 个叶子的路径长度
第 k 个叶子的权值；

小结

11、赫夫曼树

(1) 赫夫曼树相关概念

③ 赫夫曼树——带权路径长度 (WPL) 最小的二叉树

- ✓ 权值越大的叶子结点越靠近根结点，而权值越小的叶子结点越远离根结点。
- ✓ 只有度为0（叶子结点）和度为2（分支结点）的结点，不存在度为1的结点。

小结

11、赫夫曼树

(2) 赫夫曼树存储结构

weight	lchild	rchild	parent
--------	--------	--------	--------

(3) 构造赫夫曼树

- ①构造森林全是根，②选用两小造新树，
- ③删除两小添新人，④重复2、3 剩单根。

1. 数组huffTree初始化，所有元素结点的双亲、左右孩子都置为-1；
2. 数组huffTree的前n个元素的权值置给定值w[n]；
3. 进行n-1次合并
 - 3.1 在二叉树集合中选取两个权值最小的根结点，其下标分别为 i_1 ， i_2 ；
 - 3.2 将二叉树 i_1 、 i_2 合并为一棵新的二叉树k；

小结

11、赫夫曼树

(4) 赫夫曼编码

构造好Huffman树后，将树中每个分支结点的左分支上标上“0”，右分支上标上“1”，把从根到叶子结点的路径上分支符号（0或1）连接起来得到的二进制编码称为叶子结点的Huffman编码。

①为什么哈夫曼编码能够保证是前缀编码？

因为没有一片树叶是另一片树叶的祖先，所以每个叶结点的编码就不可能是其它叶结点编码的前缀。

②为什么哈夫曼编码能够保证字符编码总长最短？

因为哈夫曼树的带权路径长度最短，故字符编码的总长最短。

正在答疑
