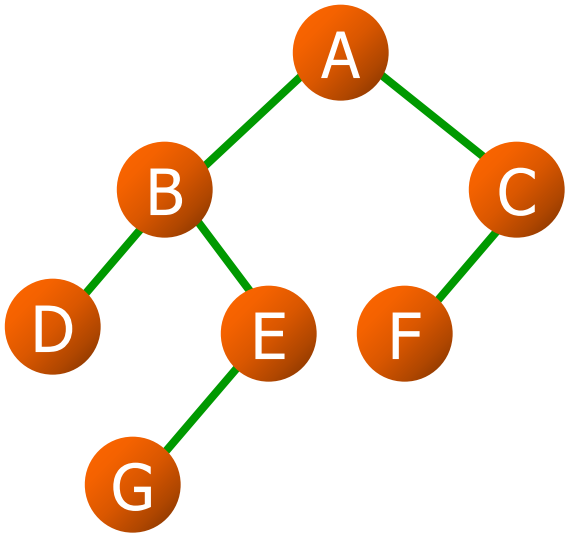
6.6 赫夫曼树

一. 最优二叉树（赫夫曼树）

- 树的路径和路径长度
  - **路径**：从一个结点到另一个结点之间的分支构成这两个结点之间的路径。
  - **路径长度**：路径上的分支数目。
  - **树的路径长度**：从树的根结点到每个结点的路径长度之和。



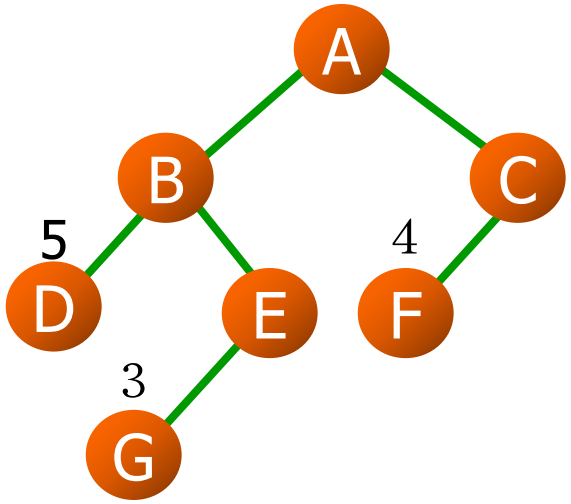
例如：右图所示树的路径长度为：

2*1 + 3*2 + 1*3 = 11

6.6 赫夫曼树

一. 最优二叉树

- **结点的带权路径长度**：从结点到树根之间的路径长度与结点上权的乘积。
- **树的带权路径长度(WPL)**：树中所有叶子结点的带权路径长度之和。

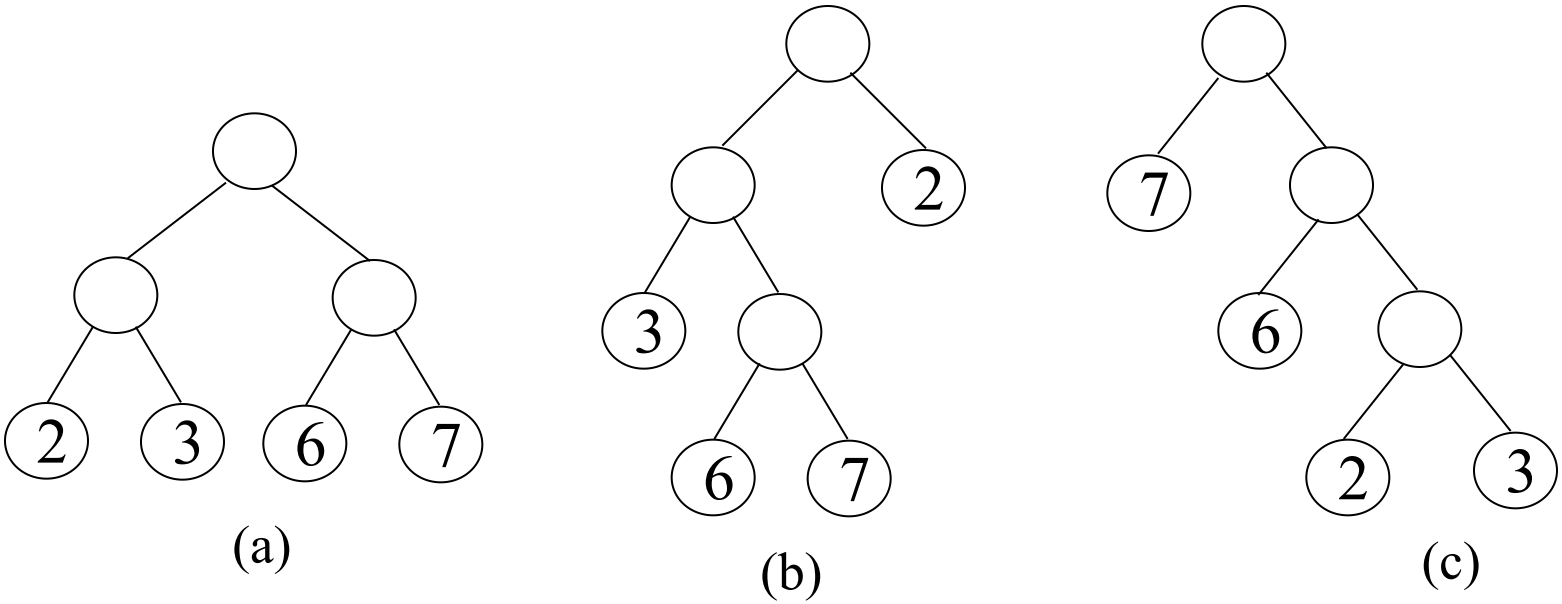


例如：右图所示，树的带权路径长度
WPL： 2*5+3*3+2*4=27

6.6 赫夫曼树

一. 最优二叉树

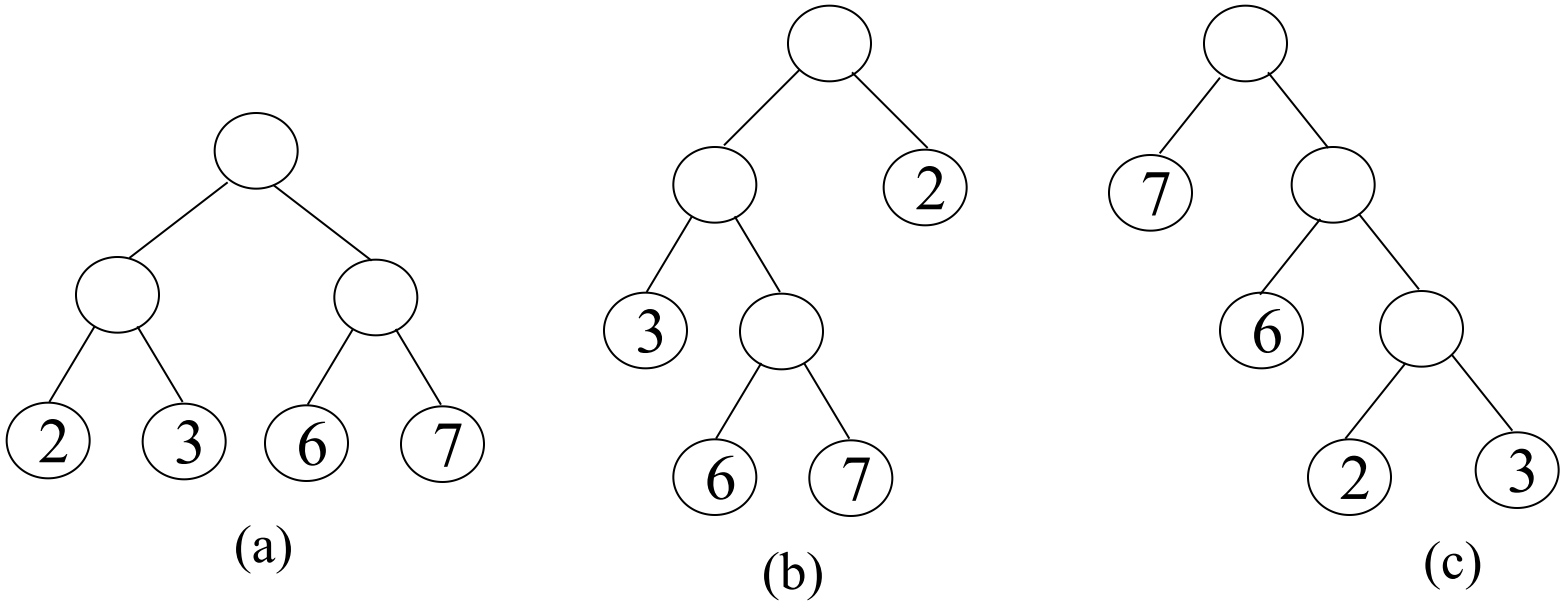
- 具有n个叶子结点(每个结点的权值为w_i) 的二叉树，但在其中**WPL值最小**的二叉树，称为**赫夫曼树**(或最优二叉树)
- 如图是权值分别为2、3、6、7，具有4个叶子结点的二叉树



具有相同叶子结点，不同带权路径长度的二叉树

6.6 赫夫曼树

一. 最优二叉树



它们的带权路径长度分别为：

(a) WPL=2×2+3×2+6×2+7×2=36 ；

(b) WPL=2×1+3×2+6×3+7×3=47 ；

(c) WPL=7×1+6×2+2×3+3×3=34 。

其中(c)的 WPL值最小，称这棵树为最优二叉树或Huffman树

6.6 赫夫曼树

一. 最优二叉树

- 赫夫曼树的简单应用——百分制转五级制
  - 将0～100分转变为不及格、及格、中等、良好、优良五个级别
  - 分数的分布比例为：

分数	0-59	60-69	70-79	80-89	90-100
比例数	0.05	0.15	0.40	0.30	0.10

  - 普通方法用四次判断语句
  - 不同成绩的分布规律构造赫夫曼树，通过赫夫曼树进行判定，大大减少判断次数
  - 构造二叉树的思路：把分布比例数作为叶子权值，每个分支结点包含一个判断，把高权值结点放在路径短的地方，低权值放在路径长的地方

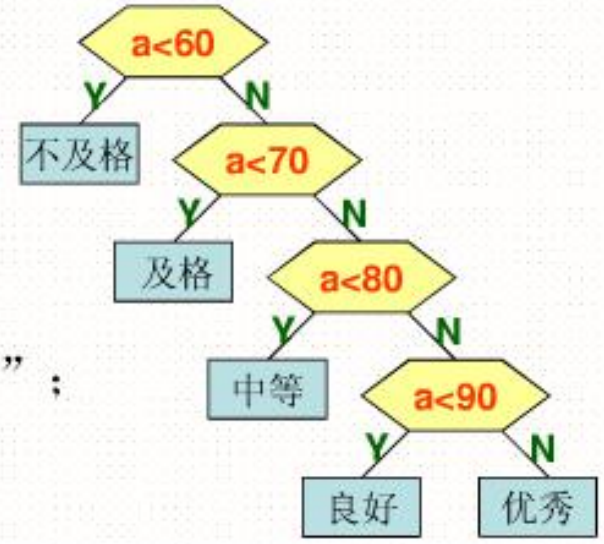
6.6 赫夫曼树

一. 最优二叉树

- 赫夫曼树的简单应用——百分制转五级制
  - 采用普通方法用四次判断语句
  - 若有10000个输入数据，根据分数分布比例需要31500次
  - 80%以上数据需要经过3次以上的比较才能得到结果

a是一个百分制分数

```
If (a<60)  b="不及格"；
else  if (a<70) b="及格"；
      else  if (a<80)  b="中等"；
            else  if (a<90)  b="良好"；
                  else b="优秀"；
```

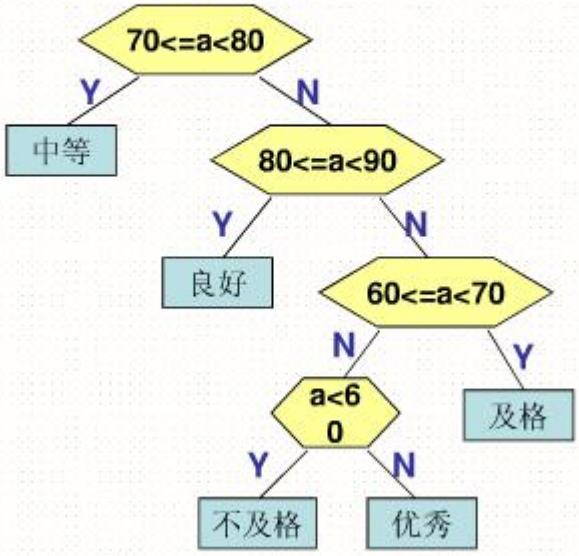


6.6 赫夫曼树

一. 最优二叉树

- 赫夫曼树的简单应用——百分制转五级制
  - 把高比例数据先做比较，程序调整如下，10000个输入数据需要22000次判断

分数	0-59	60-69	70-79	80-89	90-100
比例数	0.05	0.15	0.40	0.30	0.10

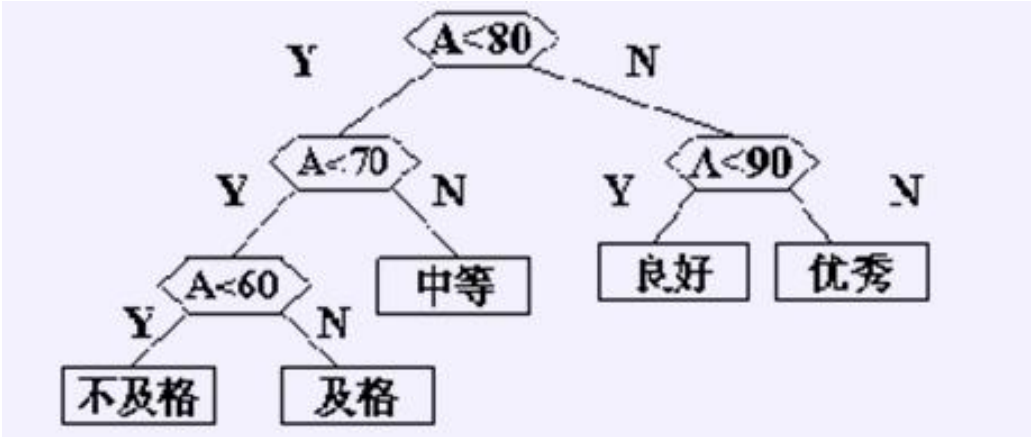


赫夫曼树

6.6 赫夫曼树

一. 最优二叉树

- 赫夫曼树的简单应用——百分制转五级制
  - 改进方法中每个比较有包含两次判断，再做细化，得到如下的判定树
  - 10000个输入数据需要22000次判断

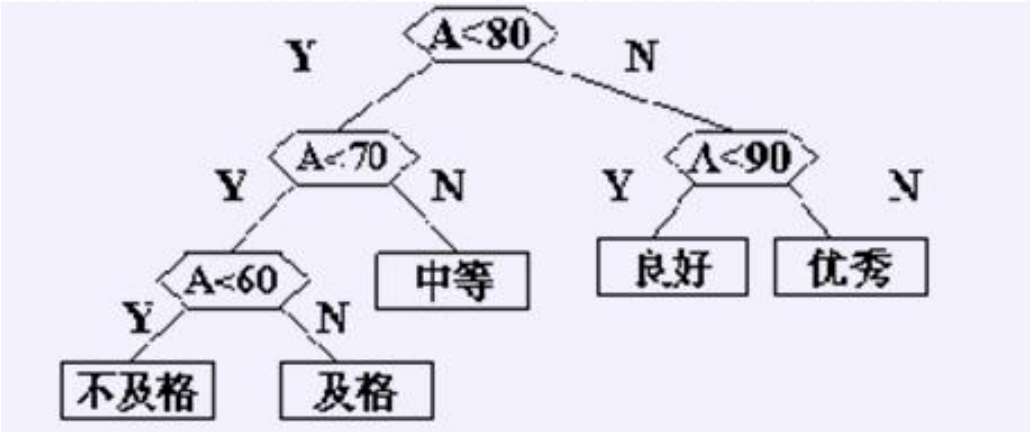


6.6 赫夫曼树

一. 最优二叉树

- 赫夫曼树的简单应用——百分制转五级制
  - 以5、15、40、30和10为权值，构造一颗有5个叶子的赫夫曼树，也就得到同样的这颗二叉树

分数	0-59	60-69	70-79	80-89	90-100
比例数	0.05	0.15	0.40	0.30	0.10



6.6 赫夫曼树

二. 赫夫曼树的构造

- 构造算法：

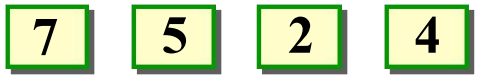
1. 根据给定的n个权值$(w_1, w_2, \ldots, w_n)$构成n棵二叉树的集合$F=\{T_1, T_2, \ldots, T_n\}$，其中每棵二叉树$T_i$中只有一个权值为$w_i$的根结点；
2. 在F中选取两棵根结点的权值最小的树作为左、右子树构造一棵新的二叉树，且置其根结点的权值为其左、右子树权值之和；
3. 在F中删除这两棵树，同时将新得到的二叉树加入F中；
4. 重复2，3，直到F只含一棵树为止。

6.6 赫夫曼树

二. 赫夫曼树的构造

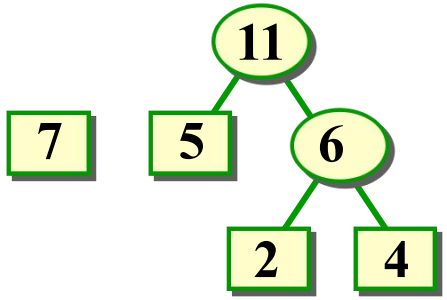
- 举例

F : {7} {5} {2} {4}



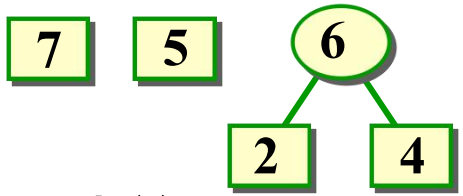
初始

F : {7} {11}



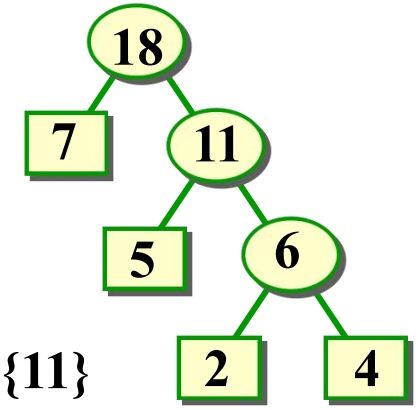
合并{5} {6}

F : {7} {5} {6}



合并{2} {4}

F : {18}



合并{7} {11}

6.6 赫夫曼树

二. 赫夫曼树的构造

- Huffman树的特点
  - 权值最大的结点离根最近，权值最小的结点离根最远
  - 树中没有度为1的结点
  - 如果叶子结点的个数为n，那么树中共有2n-1个结点

6.6 赫夫曼树

三. 赫夫曼编码

- 通过赫夫曼树把电文缩短，减少传送时间

- 编码前
  - 设给出一段报文： GOOD_GOOD_GOOD_GOOOOOOOO_OFF
  - 字符集合是｛O, G, _, D, F｝，各个字符出现的频度(次数)是W＝｛15, 4, 4, 3, 2｝。
  - 若给每个字符以等长编码

    O: 000

    G: 001

    _: 010

    D: 011

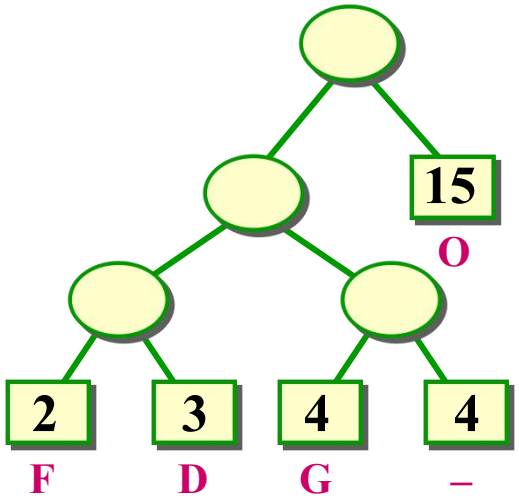
    F: 100

    则总编码长度为（15+4+4+3+2）＊ 3 ＝ 84

6.6 赫夫曼树

三. 赫夫曼编码

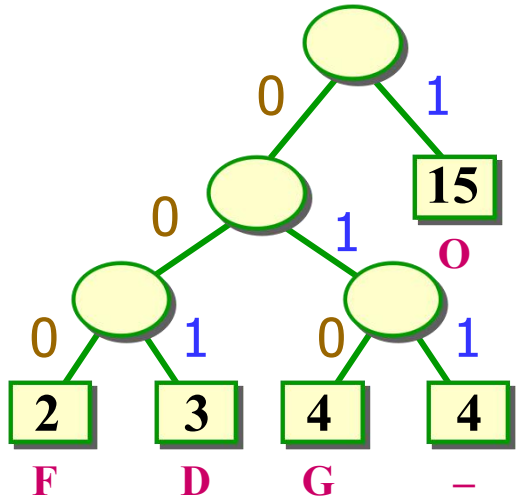
- 编码后
  - 若按各个字符出现的概率不同而给予不等长编码，可望减少总编码长度。
  - 各字符{ O, G, _, D, F }出现概率为
    { 15/28, 4/28, 4/28, 3/28, 2/28 }
  - 各字符出现概率[取整数]为{15, 4, 4, 3, 2}
  - 如果规定，Huffman树的左子树小于右子树，则可构成右图所示Huffman树



6.6 赫夫曼树

三. 赫夫曼编码

- 编码后
  - 令左孩子分支编码为‘0’，右孩子分支编码为‘1’
  - 将根结点到叶子结点路径上的分支编码组合起来，作为该字符的**Huffmanb编码**，则可得到：



O: 1

F: 000

D: 001

G: 010

_ : 011

6.6 赫夫曼树

三. 赫夫曼编码

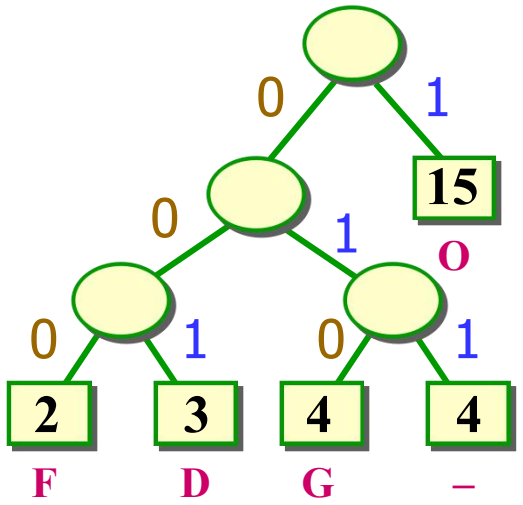
- 则总编码长度为

  15*1+(2+3+4+4)*3 = 54 < 84

- 前缀编码：任何一个字符都不是另一个字符的编码的前缀。

- Huffman是一种前缀编码，解码时不会混淆。

例如：GOOD编码为：01011001



O: 1
_: 011
G: 010
D: 001
F: 000

6.6 赫夫曼树

三. 赫夫曼编码

- 赫夫曼树的实现

Huffman树的构造，用静态链表实现,结构如下。

data	weight	parent	lchild	rchild

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

```
typedef char * * HuffmanCode;
```


17

6.6 赫夫曼树

三. 赫夫曼编码

- 赫夫曼树的实现

设静态链表为HT（含有2n个结点，n=叶子结点数，0号单元未用），过程如下：

① 初始令所有单元的parent、lchild、rchild 均为0，设置1 ~ n结点的data、weight，n++；

② 从1 ~ n-1单元中选取parent为0、且权值最小的两个结点，设为s1、s2，令n为此两结点的父结点，修改：

```
HT[s1].parent = n;      HT[s2].parent = n;
HT[n].lchild = s1;      HT[n].rchild = s2;
HT[n].weight = HT[s1].weight + HT[s2].weight;
n++;
```

③ 重复②直至 n = 2*叶子结点数

6.6 赫夫曼树

三. 赫夫曼编码

- 赫夫曼树的实现

以电文‘ABACCDA’的Huffman树构造为例说明静态链表的变化。

- 初始

⇨

	weight	parent	lchild	rchild
1	A	3	0	0	0
2	B	1	0	0	0
3	C	2	0	0	0
4	D	1	0	0	0
5		0	0	0	0
6		0	0	0	0
7		0	0	0	0

0表示无双亲或无孩子

☞ 第一次合并后：

		weight	parent	lchild	rchild
1	A	3	0	0	0
2	B	1	5	0	0
3	C	2	0	0	0
4	D	1	5	0	0
⇨ 5		2	0	2	4
6		0	0	0	0
7		0	0	0	0

- n个叶子结点的Huffman树共2n-1个结点。两两合并，直至一棵树，共生成n-1个结点，完成。

		weight	parent	lchild	rchild
1	A	3	7	0	0
2	B	1	5	0	0
3	C	2	6	0	0
4	D	1	5	0	0
5		2	6	2	4
6		4	7	3	5
7		7	0	1	6

6.6 赫夫曼树

三. 赫夫曼编码

- 赫夫曼编码的实现

用字符数组 char code[n]存放单结点编码。

6.6 赫夫曼树

三. 赫夫曼编码

- 赫夫曼树的实现

编码从叶子结点开始。对每个叶子结点k ，

① 初始令start = n-2

② parent = HT[k].parent;

若HT[parent].lchild = k; 则code[start] ='0';

若HT[parent].rchild = k; 则code[start] ='1' ;

k = parent; start--;

③ 重复②直至HT[p].parent = 0;

④ 每个字符的编码值为code[start+1…n-2];

6.6 赫夫曼树

三. 赫夫曼编码

- 赫夫曼树的实现

```
void HuffmanCoding(HuffmanTree &HT, HuffmanCode &HC, int *w, int n)
      {
    if (n<=1) return;
    m = 2 * n - 1;
    HT = (HuffmanTree)malloc((m+1) * sizeof(HTNode));  // 0号单元未用
    for (i=1; i<=n; i++) { //初始化
          HT[i].weight=w[i-1];    HT[i].parent=0;
          HT[i].lchild=0;     HT[i].rchild=0;
    }
    for (i=n+1; i<=m; i++) { //初始化
          HT[i].weight=0;    HT[i].parent=0;
          HT[i].lchild=0;      HT[i].rchild=0;
}
```

6.6 赫夫曼树

三. 赫夫曼编码

- 赫夫曼树的实现

```
for (i=n+1; i<=m; i++) {  // 建哈夫曼树
    // 在HT[1..i-1]中选择parent为0且weight最小的两个结点,
    // 其序号分别为s1和s2。
    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;
}
```

（转下页）

6.6 赫夫曼树

三. 赫夫曼编码

- 赫夫曼树的实现

（接上页）

```
//--- 从叶子到根逆向求每个字符的哈夫曼编码 ---
  cd = (char *)malloc(n*sizeof(char));     // 分配求编码的工作空间
  cd[n-1] = '\0';                          // 编码结束符。
  for (i=1; i<=n; ++i) {                   // 逐个字符求哈夫曼编码
    start = n-1;                           // 编码结束符位置
    for (c=i, f=HT[i].parent; f!=0; c=f, f=HT[f].parent)
      // 从叶子到根逆向求编码
      if (HT[f].lchild==c) cd[--start] = '0';
      else cd[--start] = '1';
    HC[i] = (char *)malloc((n-start)*sizeof(char));
          // 为第i个字符编码分配空间
    strcpy(HC[i], &cd[start]);     // 从cd复制编码(串)到HC
  }
  free(cd);    // 释放工作空间
}
```

6.6 赫夫曼树

三. 赫夫曼编码

- 课本P148，例2

译码的过程是分解电文中字符串，从根出发，按字符[illegible]

直至叶子结点，便求得该子串相应的字符。具体算法留给读者去完成。

例 6-2 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。

设权 $w=(5,29,7,8,14,23,3,11)$，$n=8$，则 $m=15$，按上述算法可构造一棵赫夫曼树如图 6.26 所示。其存储结构 HT 的初始状态如图 6.27(a)所示，其终结状态如图 6.27(b)所示，所得赫夫曼编码如图 6.27(c)所示。

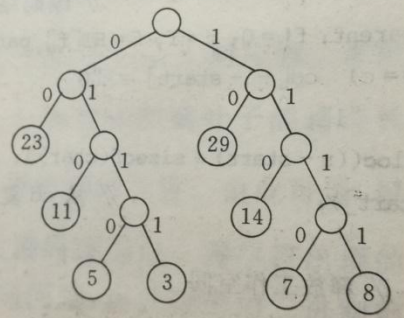


图 6.26 例 6-2 的赫夫曼树

6.6 赫夫曼树

三. 赫夫曼编码

HT

	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

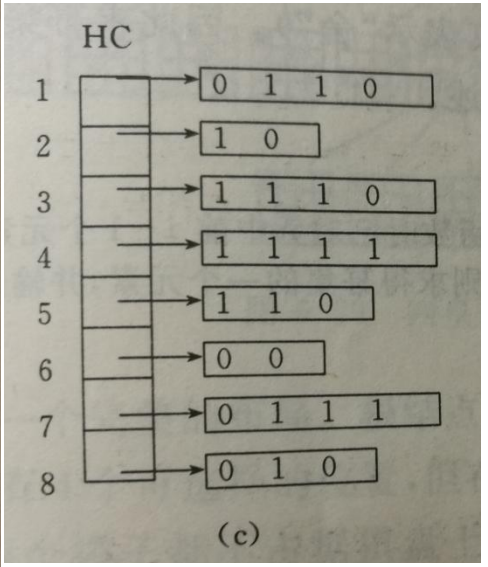
(a)

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

(b)

HC



(c)

6.6 赫夫曼树

三. 赫夫曼解码

- 算法流程
  - 定义指针p指向赫夫曼树结点，实际是记录结点数组的下标
  - 定义指针i指向编码串，定义ch逐个取编码串的字符
  - 初始化：读入编码串，设置p指向根结点，i为0
  - 执行以下循环：

1. 取编码串的第i个字符放入ch
2. 如果ch是字符0，表示往左孩子移动，则p跳转到左孩子
3. 如果ch是字符1，表示往右孩子移动，则p跳转到右孩子
4. 如果ch非0非1，表示编码串有错误，输出error表示解码失败
5. 检查p指向的结点是否为叶子
   1. 如果是叶子，输出解码，p跳回根节点
   2. 如果不是叶子，设置ch为3
6. 继续循环，一直到编码串末尾

- 循环执行完后，如果ch值为3，输出解码失败，否则成功结束

练习

- 已知二叉树如图所示，完成以下要求

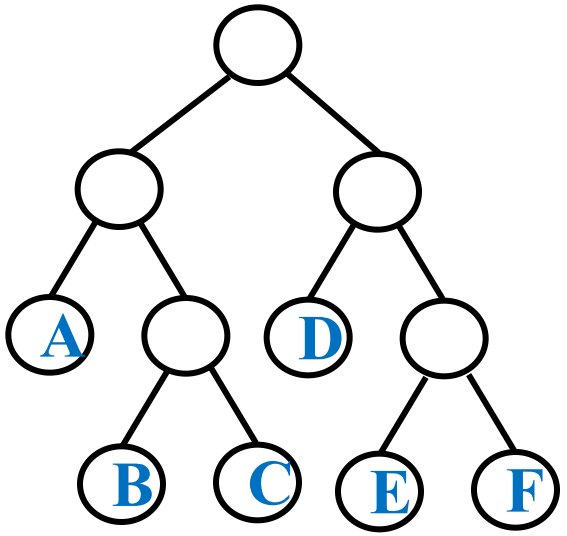
① 求树的路径长度。

② 若从左到右的叶子结点的权值分别为3、14、34、26、16、7，求树的带权路径长度。

③ 该树是否是赫夫曼树？如果不是，请构建赫夫曼树，计算WPL值，并设计赫夫曼编码。

④ 对以下编码串进行解码，若解码异常则输出error。

- 11011000101
- 11101
- 000100



练习

- 假设用于通信的电文是由字符集{a, b, c, d, e, f, g, h}中的字符构成， 这8个字符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} 。
  - 请画出对应的huffman树（按左子树根结点的权小于等于右子树根结点的权的次序构造）
  - 求出每个字符的huffman编码。

第六章总结

- 树的概念和术语，拥有唯一根结点，父子关系
  - 结点、度、叶结点、孩子、兄弟、祖先、双亲、层次、深度、森林
- 二叉树的概念和五个性质（性质5）
- 满二叉树和完全二叉树
- 二叉树的数组表示和链式表示
- 二叉树遍历：先序、中序、后序、层次
- 二叉树的创建方法
  - 先序遍历+空子树0、层次遍历+空子树0
  - 先序+中序推导、后序+中序推导
- 线索二叉树：增加新指针，增加标志位
- 树的存储结构
  - 数组表示、多重链表、单链表
  - 孩子兄弟表示法：左边是孩子，右边是兄弟
  - 树转换为二叉树，森林转换为二叉树
- 树的先根和后根遍历对应二叉树的先序和中序遍历
- 树的路径长度、树的带权路径长度
- 最优二叉树，即赫夫曼树：构建、编码