下面我们来看一下哈夫曼算法。哈夫曼算法是指，为各压缩对象文件分别构造最佳的编码体系，并以该编码体系为基础来进行压缩。因此，用什么样式的编码（哈夫曼编码）对数据进行分割，就要由各个文件而定。用哈夫曼算法压缩过的文件中，存储着哈夫曼编码信息和压缩过的数据（图 6-4）。

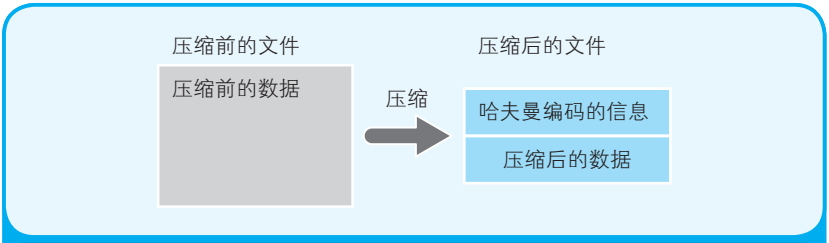


图 6-4　用哈夫曼算法压缩的文件的构造

接下来，我们尝试一下把 AAAAAABBCDDEEEEEF 中的 A～F 这些字符，按照“出现频率高的字符用尽量少的位数编码来表示”这一原则进行整理。按照出现频率从高到低的顺序整理后，结果就如表 6-3 所示。该表中同时也列出了编码的方案。

表 6-3　出现频率和编码（方案）

字符	出现频率	编码（方案）	位数
A	6	0	1
E	5	1	1
B	2	10	2
D	2	11	2
C	1	100	3
F	1	101	3

在表 6-3 的编码（方案）中，随着出现频率的降低，字符编码信息

的数据位数也在逐渐增加，从开始的 1 位、2 位，依次增加到 3 位。不过，这个编码体系是存在问题的。该问题就是，例如 100 这个 3 位的编码，它的意思是用 1、0、0 这 3 个编码来表示 E、A、A 呢？还是用 10、0 这两个编码来表示 B 、A 呢？亦或是用 100 来表示 C 呢？这些都无法进行区分。因此，如果不加入用来区分字符的符号，这个编码（方案）就无法使用。

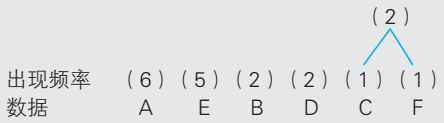
而在哈夫曼算法中，通过借助**哈夫曼树**构造编码体系，即使在不使用字符区分符号的情况下，也可以构建能够明确进行区分的编码体系。也就是说，利用哈夫曼树后，就算表示各字符的数据位数不同，也能够做成可以明确区分的编码。因此，只要掌握了哈夫曼树的制作方法，并用程序将其完成，就可以借助哈夫曼算法实现文件压缩了。不过，与 RLE 算法相比，程序的内容要复杂很多。

接下来我们就来看一下如何制作哈夫曼树。自然界的树是从根开始生枝长叶的。而哈夫曼树则是从叶生枝，然后再生根。图 6-5 展示了对 AAAAAABBCDDEEEEEF 进行编码的哈夫曼树的制作过程。大家也尝试绘制一下吧。尝试过 1 次后，应该就能理解哈夫曼树的制作顺序了。

步骤1：列出数据及其出现频率，（ ）里面表示的是出现频率，这里按照降序排列

出现频率	（6）	（5）	（2）	（2）	（1）	（1）
数据	A	E	B	D	C	F

步骤2：选择两个出现频率最小的数字，拉出两条线，并在交叉地方写上这两位数字的和。当有多个选项时，任意选取即可



步骤3：重复步骤2，可以连接任何位置的数值

步骤4：最后这些数字会被汇集到了1个点上，该点就是根，这样哈夫曼树也就完成了。按照从根部到底部的叶子这一顺序，在左边的树枝（线）处写上0，在右边的树枝（线）处写上1。然后从根部开始沿着树枝到达目标文字后，再按照顺序把通过的树枝上的0或者1写下来，就可以得到哈夫曼编码了

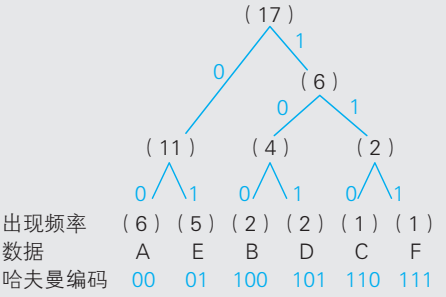


图 6-5　哈夫曼树的编码顺序

6.6 哈夫曼算法能够大幅提升压缩比率

使用哈夫曼树后，出现频率越高的数据所占用的数据位数就越少，而且数据的区分也可以很清晰地实现。但哈夫曼算法为什么达到这么好的效果呢，大家都了解吗？

通过图 6-5 的步骤 2 可以发现，在用枝条连接数据时，我们是从出现频率较低的数据开始的，这就意味着出现频率越低的数据到达根部的枝条数就越多。而枝条数越多，编码的位数也就随之增多了。

而从用哈夫曼算法压缩过的文件中读取数据后，就会以位为单位对该数据进行排查，并与哈夫曼树进行比较看是否到达了目标编码，这就是为什么哈夫曼算法可以对数据进行区分的原因。例如，10001 这个使用图 6-5 所示的哈夫曼编码作成的 5 位数据，到达 100 时，对照哈夫曼树的数据，该数据表示的是 B 这个字符。至此就找到了 1 个字符。然后再顺着哈夫曼树寻找剩下的 01，会发现它表示的是 E 这个字符。

接下来，让我们来看一下哈夫曼算法的压缩比率。用图 6-5 得到的哈夫曼编码表示 AAAAAABBCDDEEEEEF，结果为 00000000000010010011010110101010101111，40 位 = 5 字节（这里为不包含哈夫曼编码信息的情况）。压缩前的数据是 17 字符 = 17 字节，也就是说，我们惊奇地得到了 5 字节 ÷ 17 字节 ≒ 29% 这样高的压缩率。表 6-4 是将表 6-1 中的文件应用哈夫曼算法的 LHA 进行压缩后的结果，大家可以参考一下。可以看出，不管是哪种类型的文件，都得到了很高的压缩比率。

表 6-4 LHA 对各种文件的压缩结果

文件类型	压缩前	压缩后	压缩比率
文本文件	14 862 字节	4119 字节	28%
图像文件	96 062 字节	9456 字节	10%
EXE 文件	24 576 字节	4652 字节	19%

6.7　可逆压缩和非可逆压缩

最后，让我们来看一下图像文件的数据形式。图像文件的使用目的通常是把图像数据输出到显示器、打印机等设备上。Windows 的标准图像数据形式为 BMP[①]，是完全未压缩的。由于显示器及打印机输出的 bit（点）是可以直接映射（mapping）的，因此便有了 BMP = bitmap 这一名称。

除 BMP 格式以外，还有其他各种格式的图像数据形式。比如 JPEG[②] 格式、TIFF[③] 格式、GIF[④] 格式等。与 BMP 格式不同的是，这些图像数据都会用一些技法来对数据进行压缩。

图像文件还可以使用与前文介绍的 RLE 算法、哈夫曼算法不同的其他压缩算法。这是因为，多数情况下，并不要求压缩后的图像文件必须还原到与压缩前同等的质量。与之相比，程序的 EXE 文件以及每个字符、数值都有具体含义的文本文件则必须要还原到和压缩前同样的内容。而对于图像文件来说，即使有时无法还原到压缩前那样鲜明的图像状态，但只要肉眼看不出什么区别，有一些模糊也勉强可以接受。这里，我们把能还原到压缩前状态的压缩称为**可逆压缩**，无法还原到压缩前状态的压缩称为**非可逆压缩**，这一点希望大家记住（图 6-6）。

① BMP（Bitmap）是使用 Windows 自带的画笔来做成的一种图像数据形式。

② JPEG（Joint Photographic Experts Group）是数码相机等常用的一种图像数据形式。

③ TIFF（Tag Image File Format）是一种通过在文件头中包含“标签”就能够显示出数据性质的图像数据形式。

④ GIF（Graphics Interchange Format）是由美国 CompuServe 开发的一种数据格式。这种格式要求色数不超过 256 色。

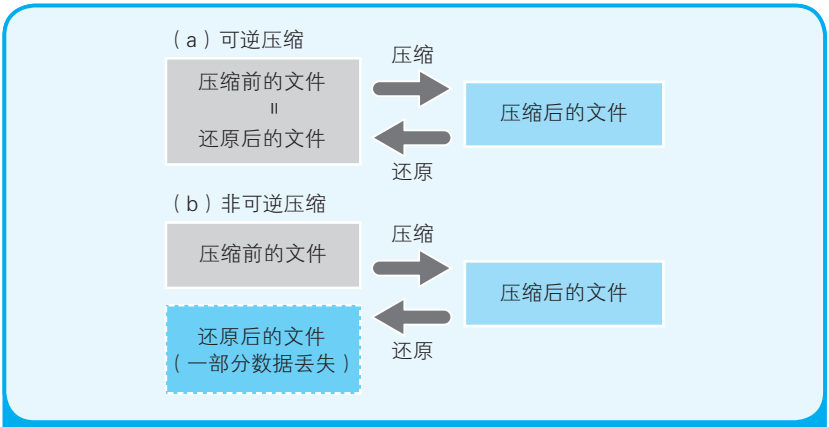


图 6-6 图像文件的压缩有可逆压缩和不可逆压缩两种

图 6-7 中列出了各种格式的图像文件。其中，原始的图像文件是 BMP 格式。通过此图可以看出，JPEG 格式和 GIF 格式的图像文件有一些模糊。这是因为 JPEG 格式[①] 的文件是非可逆压缩，因此还原后的

① 数码相机中经常用到的 JPEG 格式文件，有 3 种压缩方式。

（1）把构成图像的点阵的颜色信息由 RGB（红色、绿色、蓝色）形式转化成 YCbCr（亮度、蓝色色度、红色色度）形式。我们知道，人眼对亮度很敏感，但对颜色的变化却有些迟钝。因此，人眼比较敏感的亮度 Y 就是一个很重要的参数，而表示颜色的 Cb、Cr 则没有那么重要。于是我们就可以通过减少 Cb 和 Cr 的信息间距来缩小图像数据的大小。

（2）将每个点的色素变化看作是波形的信号变化，进行傅里叶变换。傅里叶变换是指将波形按照频率分量进行分解。照片等图像文件的特点是低频率（柔和的颜色变化）的部分较多，高频率（强烈的的颜色变化）的部分较少。因此，这里我们就可以把高频率的部分剪切掉。这样一来，图像数据也就会缩小。虽然剪切掉了高频率部分，但人眼分辨不出什么差别。不过，如果是用 Windows 画笔描绘的简单图形，其中颜色变化强烈的部分就会出现模糊现象。大家不妨使用 Windows 画笔做一个圆形或者四方形的图形，并将其保存成 JPEG 格式。然后再打开这个 JPEG 文件，你就会发现颜色变化强烈的部分变模糊了。

（3）最后，将已经瘦身的图像数据通过哈夫曼算法进行压缩。这样就可以使图像数据进一步缩小。

图像信息有一部分是模糊的。而 GIF 格式的文件虽然是可逆压缩，但因为有色数不能超过 256 色的限制，所以还原后颜色信息会有一些缺失，进而导致了图像模糊。TIFF 格式的图像文件虽然不模糊，但却比原始的 BMP 格式的文件还要大，这是为什么呢？我们知道，TIFF 格式的文件中带有各种标签信息，是可以选择压缩格式的，而这里选择的是与 BMP 同样的无压缩方式。但由于与原始的图像数据相比，TIFF 格式的文件中附加了标签信息，所以结果就比 BMP 格式的文件更大了。

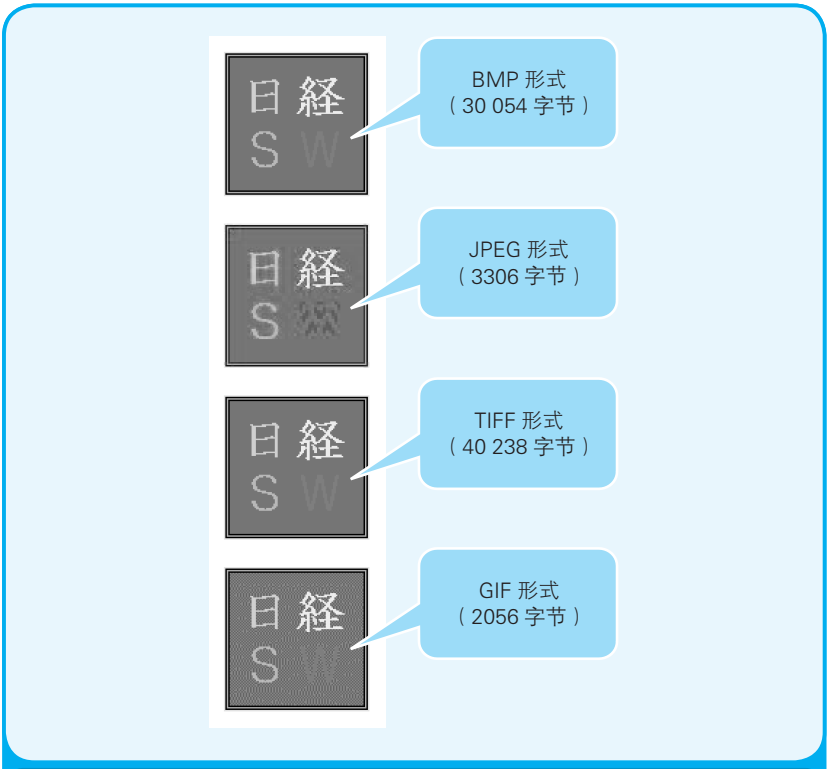
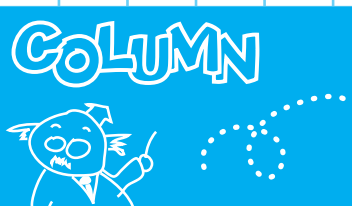


图 6-7　各种形式的图像文件的画质对比

压缩算法的种类大概有一二十种。之所以会存在如此多的压缩算法，是因为压缩比率、压缩需要的处理时间（程序的复杂程度）以及各种文件的需求等是不一样的。因此，至今学界都不能提出一个万能的压缩算法。而这也为各位读者提供了一个展露才能的机会。大家不妨尝试一下，自己原创一个压缩算法。不过有一点需要注意，文本文件不能进行非可逆压缩。至于原因，想必大家也都清楚了吧。

接下来的一章，我们将会返回到本书的主题，对程序的运行环境进行说明。



如果是你，你会怎样介绍？

向沉迷游戏的中学生讲解内存和磁盘

笔者：你现在最想要的东西是什么？

中学生：现在？游戏机呗。

笔者：那你都有什么游戏机啊？

中学生：任天堂 DS 和 PlayStation。

笔者：（太好了！这样就可以向他讲解内存和磁盘了）嗯嗯。那么，任天堂 DS 使用的是盒式磁带，PlayStation 使用的是 CD，对吧。磁盘和 CD 有什么不同呢？

中学生：CD 可以存放大量的数据，另外图像和声音也很有冲击力啊。

笔者：说得对！说得对！ 你知道吗，任天堂 DS 和 PlayStation 都是计算机的一种。计算机不仅可以玩游戏，也可以进行文档处理和上网，所以说任天堂 DS 和 PlayStation 就是游戏专用的计算机。

中学生：这个我也知道啊。

笔者：计算机是运行软件的机械设备，这些软件可以放在磁盘及 CD 中，这些你知道吧？

中学生：当然知道了。

笔者：CD 就像唱片一样，是通过表面的凹凸来存储软件的，这一点想必你也知道吧。那么盒式磁带中是什么样子你知道吗？

中学生：简单啊。里面有内存啊。

笔者：了不起！答对了！那你知道内存是如何存储软件的吗？

中学生：……

笔者：是通过电流的有无来存储的。你可以这样理解，有电流时是凸，无电流时是凹。

中学生：那么，为什么 CD 能存储更多的数据呢？

笔者：（唉，这还真是个难题……怎么回答好呢……有了！）盒式磁带使用大量内存的话也可以放入大量数据啊。不过，到时候 1 个盒式磁带就要几千元了。

中学生：几千元，买不起啊。

笔者：对啊。正是因为如此，数据量大的软件才放在成本较低的 CD 中进行存储。不过，CD 中存储的

软件，也要复制到游戏机的内存中才能运行。

中学生：那不就是说，最后还是用到了内存吗？

笔者：确实是这样，游戏机的内存，只能放入少量的数据。因此，游戏机是一边把 CD 中存储的软件部分复制入内存，一边运行游戏的。

中学生：怪不得游戏中会出现 Loading…呢。原来如此，明白了。

笔者：对，就是这样！正如刚才所说的，计算机中用来存储数据的手段，有类似于 CD 这样的磁盘和内存这两种。而且从现状来看，磁盘比内存要便宜。

中学生：那么，所有的游戏都放在磁盘中的话不也挺好嘛。

笔者：这个提议虽说不错，但正如刚才所说的那样，游戏要在游戏机中运行必须要复制到内存中才行。

中学生：盒式磁盘的数据也要复制到内存中吗？

笔者：不用。盒式磁带的情况下，可以将游戏机主机的内存完整置换，所以不需要往内存中复制数据。只有磁盘才必须把数据复制到内存中。

中学生：原来如此啊。

笔者：明白了吗？

中学生：知道啦。

笔者：确定？

中学生：确定。我要继续玩游戏去了。

笔者：那么，刚才玩游戏时的数据，存储在了什么地方你知道吗？

中学生：这个话题，咱们就不说了吧。

笔者：喂，等一下！

中学生：Byebye！