图 5-10　仅有 1 个文字也会占用 1 簇（512 字节）

以簇为单位进行读写时，1 簇中没有填满的区域会保持不被使用的状态。虽然这看起来是有点浪费，不过该机制就是如此规定的，所以我们也没有什么好办法。另外，如果减少簇的容量，磁盘访问次数就会增加，就会导致读写文件的时间变长。由于在磁盘表面上，表示扇区区分的领域是必要的，因此，如果簇的容量过小，磁盘的整体容量也会减少。扇区和簇的大小，是由处理速度和存储容量的平衡来决定的。

阅读本章后，关于内存和磁盘的亲密关系，大家应该都清楚了吧。虽然现在计算机中的内存和磁盘容量变得越来越大，不过还是要有节约的精神。一个优秀的程序，不仅要运行速度快，还要小。因此，程序员要时刻注意尽量让程序小一些。

下一章，我们将会介绍图像文件的数据形式及文件的压缩机制。

第 6 章 亲自尝试压缩数据

热身问答

阅读正文前，让我们先回答下面的问题来热热身吧。

1. 文件储存的基本单位是什么?
2. DOC、LZH 和 TXT 这些扩展名中，哪一个是压缩文件的扩展名?
3. 文件内容用“数据的值 × 循环次数”来表示的压缩方法是 RLE 算法还是哈夫曼算法?
4. 在 Windows 计算机经常使用的 SHIFT JIS 字符编码中，1 个半角英数是用几个字节的数据来表示的?
5. BMP（BITMAP）格式的图像文件，是压缩过的吗?
6. 可逆压缩和非可逆压缩的不同点是什么?

怎么样？是不是发现有一些问题无法简单地解释清楚呢？下面是笔者的答案和解析，供大家参考。

答案

1. 1 字节（= 8 位 ）
2. LZH
3. RLE 算法
4. 1 字节（= 8 位 ）
5. 没有压缩过
6. 压缩后的数据能复原的是可逆压缩，无法复原的是非可逆压缩

解析

1. 文件是字节数据的集合体。
2. LZH 是用 LHA 等工具压缩过的文件的扩展名。
3. 例如，AAABB 这个数据压缩后就是 A3B2。
4. 半角英文数字是用 1 个字节来表示的，汉字等全角字符是用两个字节来表示的。
5. 因为 BMP 格式的图像文件是没有被压缩的，因此要比 JPEG 格式等压缩过的图像文件大不少。
6. 像照片（JPEG 格式）这样，之所以压缩后也不会让人感到不自然，就是因为使用了非可逆压缩。

本章重点

前几章的内容可能有些难，而本章我们就可以喘口气喝喝茶了，请大家放松心情来阅读。本章的主题是文件的压缩。

各位读者想必都使用过压缩文件吧。压缩文件的扩展名有 LZH[①] 和 ZIP[②] 等。比如，文件太大无法放入软盘保存时，或将大附件添加到电子邮箱时，相信大家都会采用压缩文件的方法。此外，当我们把数码相机拍摄的照片保存到计算机上时，可能也会在不知不觉中使用 JPEG 等压缩格式。那么，为什么文件可以压缩呢？想想真是不可思议。接下来就让我们一起来看看文件的压缩机制吧。

6.1 文件以字节为单位保存

在解说文件的压缩机制之前，我们首先来了解一下保存在文件中的数据形式。文件是将数据存储在磁盘等存储媒介中的一种形式。程序文件中存储数据的单位是字节。文件的大小之所以用 ××KB、××MB 等来表示，就是因为文件是以字节（B = Byte）为单位来存储的[③]。

文件就是字节数据的集合。用 1 字节（= 8 位）表示的字节数据有 256 种，用二进制数来表示的话，其范围就是 00000000～11111111。如果文件中存储的数据是文字，那么该文件就是文本文件。如果是图形，

① LZH 是用 LHA 等工具压缩过的文件的扩展名。该压缩格式有时也称为 LZH 格式。

② ZIP 是用 PKZIP 等工具压缩过的文件的扩展名。该压缩格式有时也称为 ZIP 格式。

③ 正如本书第 5 章所述，从物理上对磁盘进行读写时是以扇区（512 字节）为单位的。但另一方面，程序则可以在逻辑上以字节为单位对文件的内容进行读写。

那么该文件就是图像文件。在任何情况下，文件中的字节数据都是连续存储的，大家一定要认识到这一点（图 6-1）。

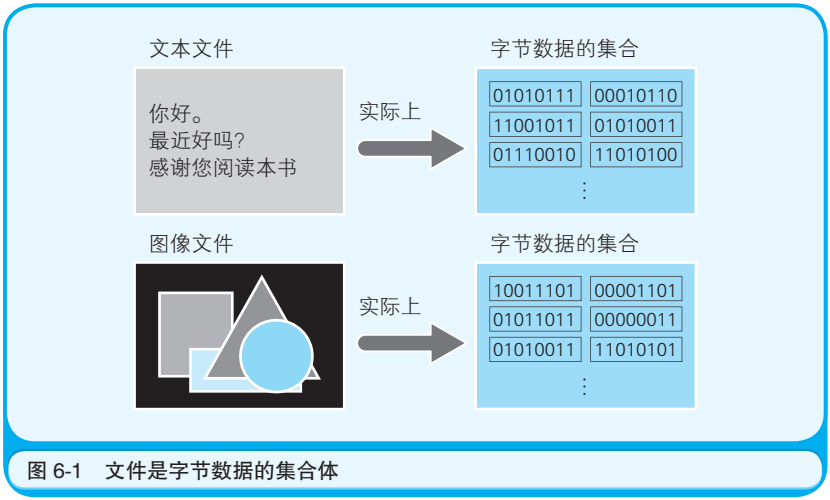

图 6-1　文件是字节数据的集合体

6.2　RLE 算法的机制

接下来就让我们正式看一下文件的压缩机制。首先让我们来尝试一下对存储着 AAAAAABBCDDEEEEEF 这 17 个半角字符的文件（文本文件）进行压缩。虽然这些文字没有什么实际意义，但是很适合用来解说 RLE 算法的压缩机制。

由于半角字母中，1 个字符是作为 1 个字节的数据被保存在文件中的。因此上述文件的大小就是 17 个字节。那么如何才能压缩该文件呢？大家也不妨考虑一下。只要能使文件小于 17 字节，我们可以使用任何压缩方法。

这时，大家是不是会采取将文件的内容用“字符 × 重复次数”这样的表现方式来压缩呢。确实，在观察 AAAAAABBCDDEEEEEF 这个

数据后，不难看出有不少字符是重复出现的。在字符后面加上重复出现次数，AAAAAABBCDDEEEEEF 就可以用 A6B2C1D2E5F1 来表示。A6B2C1D2E5F1 是 12 个字符也就是 12 字节，因此结果就将原文件压缩了 12 字节 ÷ 17 字节≒ 70%。恭喜你，压缩成功了！

像这样，把文件内容用“数据 × 重复次数”的形式来表示的压缩方法称为 RLE（Run Length Encoding，行程长度编码）算法（图 6-2）。RLE 算法是一种很好的压缩方法，经常被用于压缩传真的图像等[①]。因为图像文件本质上也是字节数据的集合体，所以可以用 RLE 算法来压缩。

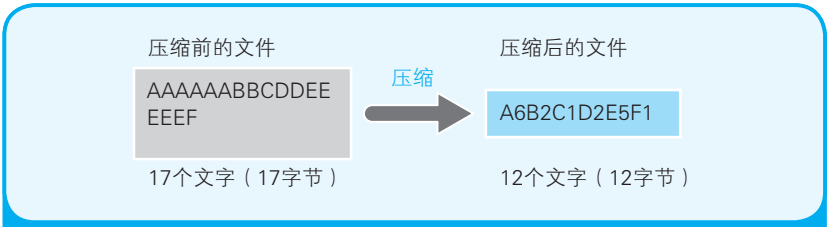

图 6-2　通过数据的重复次数来实现压缩的 RLE 算法

6.3 RLE 算法的缺点

然而，在实际的文本文件中，同样字符多次重复出现的情况并不多见。虽然针对相同数据经常连续出现的图像、文件等，RLE 算法可以发挥不错的效果，但它并不适合文本文件的压缩。不过，因为该压

① RLE 算法经常被用于传真 FAX 等。G3 类传真机是把文字和图形都作为黑白图像来发送的。由于黑白图像的数据中，白或黑通常是部分连续的，因此就没有必要再发送这部分数据的值（白或者黑），而只需附带上重复次数即可，这样压缩效率就得到了大幅提升。例如，像白色部分重复 5 次，黑色部分重复 7 次，白色部分重复 4 次，黑色部分重复 6 次这样的部分图像，就可以用 5746 这样的重复次数数字来进行压缩。

缩机制非常简单，因此使用 RLE 算法的程序也相对更容易编写。笔者曾用自己做成的 RLE 算法压缩程序对各种类型的文件进行过压缩，其结果如表 6-1 所示。

表 6-1　借助 RLE 算法对各文件进行压缩的结果

文件类型	压缩前文件大小	压缩后文件大小	压缩比率
文本文件	14862 字节	29506 字节	199%
图像文件	96062 字节	38328 字节	40%
EXE 文件	24576 字节	15198 字节	62%

通过表 6-1 可以看出，使用 RLE 算法对文本文件进行压缩后，文件却增大了，而且几乎是压缩前的 2 倍。这是因为文本文件中同样字符连续出现的部分并不多。以存储着“This is a pen.”这 14 个字符的文本文件为例，使用 RLE 算法对其进行压缩后，就变成了“T1h1i1s1 1i1s1 1a1 1p1e1n1.1”这样的 28 个字符，是压缩前的 2 倍。由于文章中字符大量连续出现的情况并不多见，因此，使用 RLE 算法后，大部分字符后面都会加上 1，这样一来，压缩后的文件自然变成了之前的 2 倍。

与文本文件不同，图像文件的压缩比率[①]达到了 40%。程序的 EXE 文件的压缩比率也达到了 60%，这是因为 EXE 文件中连续的数据部分，其初始值为 0 的情况很多。

此外，我们也可以在 RLE 算法的基础上再下点功夫，不以 1 个字符为单位，而以字符串为单位来查找重复次数。例如，This is a pen. 中，is 重复了两次。通过利用这个压缩技巧，压缩后的文件也能小一些。由此可见，压缩技巧的拙劣是由所花的功夫决定的。

① 压缩后同压缩前文件大小的比率，称为压缩比率或压缩比。

6.4 通过莫尔斯编码来看哈夫曼算法的基础

压缩技巧实际上有很多种。接下来，我们就来看一下本章要介绍的第二个压缩技巧，即哈夫曼算法。**哈夫曼算法**是哈夫曼（D. A. Huffman）于 1952 年提出来的压缩算法。日本人比较常用的压缩软件 LHA[①]，使用的就是哈夫曼算法。

为了更好地理解哈夫曼算法，首先大家要抛弃掉“半角英文数字的 1 个字符是 1 个字节（8 位）的数据”这一概念。文本文件是由不同类型的字符组合而成的，而且不同的字符出现的次数也是不同的。例如，在某一个文本文件中，A 出现了 100 次左右，Q 仅用到了 3 次，类似这样的情况是很常见的。而哈夫曼算法的关键就在于“多次出现的数据用小于 8 位的字节数来表示，不常用的数据则可以用超过 8 位的字节数来表示”。A 和 Q 都用 8 位来表示时，原文件的大小就是 100 次 × 8 位 + 3 次 × 8 位 = 824 位，而假设 A 用 2 位、Q 用 10 位来表示，压缩后的大小就是 100 次 ×2 位 +3 次 ×10 位 = 230 位。

不过有一点需要注意，不管是不满 8 位的数据，还是超过 8 位的数据，最终都要以 8 位为单位保存到文件中。这是因为磁盘是以字节（8 位）为单位来保存数据的（图 6-3）。为了实现这一处理，压缩程序的内容会复杂很多，不过作为回报，最终得到的压缩率也是相当高的。

下面让我们把当前的话题暂时放下，为了更好地理解哈夫曼算法，先来看一下莫尔斯编码。莫尔斯编码是 1837 年莫尔斯（Samuel F. B. Morse）提出的。莫尔斯编码不是通过语言，而是通过“嗒 嘀 嗒 嘀”这些长点

① LHA 是吉崎荣泰开发的一款免费压缩软件。

和短点的组合来传递文本信息的。想必大家在电影中也都看到过发送莫尔斯电码的设备。

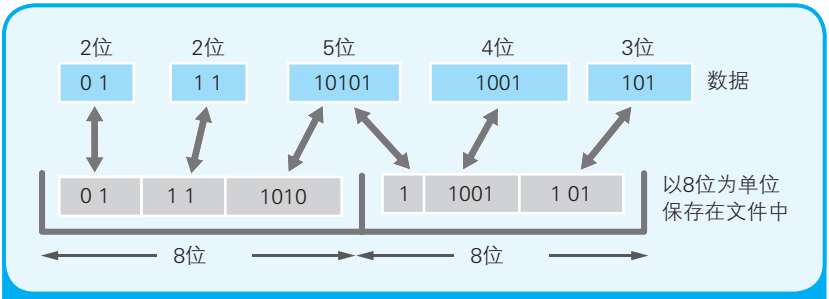

图 6-3　非 8 位数据的读写

接下来我们就来仔细讲解一下莫尔斯编码。对数字领域比较熟悉的读者可能会认为“莫尔斯编码的短点是 0，长点是 1，其中 1 个字符用 8 位来表示”，但实际上，根据字符种类的不同，莫尔斯电码符号的长度也是不同的。表 6-2 是莫尔斯编码的示例。大家把 1 看作是短点（嘀），把 11 看作是长点（嗒）即可。

表 6-2　莫尔斯编码和位长

字符	对应的位数据	位长
A	1 0 1 1	4 位
B	1 1 0 1 0 1 0 1	8 位
C	1 1 0 1 0 1 1 0 1	9 位
D	1 1 0 1 0 1	6 位
E	1	1 位
F	1 0 1 0 1 1 0 1	8 位
字符间隔	0 0	2 位

1：短点、11：长点、0：短点和长点的分隔符

莫尔斯编码把一般文本中出现频率高的字符用短编码来表示。这

里所说的出现频率，不是通过对出版物等文章进行统计调查得来的，而是根据印刷行业的印刷活字数目而确定的。如表 6-2 所示，假设表示短点的位是 1，表示长点的位是 11 的话，那么 E（嘀）这一字符的数据就可以用 1 位的 1 来表示，C（嗒嘀嗒嘀）这一字符的数据就可以用 9 位的 110101101 来表示。在实际的莫尔斯编码中，如果短点的长度是 1，长点的长度就是 3，短点和长点的间隔就是 1。这里的长度指的是声音的长度。接下来，就让我们尝试一下用莫尔斯编码来表示前面提到的 AAAAAABBCDDEEEEEF 这个 17 个字符的文本。在莫尔斯编码中，各个字符之间需要加入表示间隔的符号。这里我们用 00 来进行区分。因此，AAAAAABBCDDEEEEEF 这个文本，就变成了 A × 6 次 + B × 2 次 + C × 1 次 + D × 2 次 + E × 5 次 + F × 1 次 + 字符间隔 × 16 = 4 位 × 6 次 + 8 位 × 2 次 + 9 位 × 1 次 + 6 位 × 2 次 + 1 位 × 5 次 + 8 位 × 1 次 + 2 位 × 16 次 = 106 位 ≒ 14 字节。因为文件只能以字节为单位来存储数据，因此不满 1 字节的部分就要圆整成 1 个字节。如果所有字符占用的空间都是 1 个字节（8 位），这样文本中列出来的 17 个字符 = 17 字节，那么摩尔斯电码的压缩比率就是 14 ÷ 17 ≒ 82%，并不太突出。

6.5 用二叉树实现哈夫曼编码

刚才已经提到，莫尔斯编码是根据日常文本中各字符的出现频率来决定表示各字符的编码的数据长度的。不过，该编码体系，对 AAAAAABBCDDEEEEEF 这样的特殊文本并不是最适合的。在莫尔斯编码中，E 的数据长度最短，而在 AAAAAABBCDDEEEEEF 这个文本中，出现最频繁的是字符 A。因此，应该给 A 分配数据长度最短的编码。这样做才会使压缩率更高。