数据压缩技术原理与范例

Featuring fast, efficient data compression techniques in C

Mark Nelson

DISK INCLUDED

M&T BOOKS

希望

科学出版社

龍門書局

数据压缩技术原理与范例

[美]Mark Nelson 著

贾起东 译

李立明 审校

科学出版社

龍門書局

1995

(京)新登字 092 号

内 容 简 介

本书详细介绍了数据压缩的技术和原理，对诸如 PKZIP 和 LHarc 这样的数据压缩程序的工作方式和原理给出了详细的解释，并随附了工作代码。学习完本书以后，即使是初学 C 语言程序设计的人，也能够编写出完整的数据压缩应用程序，并可将之植入任何操作系统或硬件平台。

如果读者打算在编写的其它程序中包含数据压缩程序，则本书可以成为一个卓有价值的工具。书中包含了十余种用 C 语言编写的工作程序，可以很容易地加入到用户的应用程序中去。书中有关数据压缩技术的更深层讨论可帮助读者在创建使用数据压缩的程序时做出明智的决定。

本书适合广大的计算机专业工作人员阅读，对软件开发人员尤其有参考价值，是一本难得的有关数据压缩技术的参考书。

欲购本书的用户，请直接与北京海淀 8721 信箱书刊部联系，电话：2562329，**邮编：**100080

版 权 声 明

数据压缩技术原理与范例

[美]Mark Nelson 著

贾起东 译

李立明 审校

责任编辑 汪亚文

科学出版社
龙门书局 出版

北京东黄城根北街 16 号

邮政编码：100717

兰空印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1995 年 9 月第一版 开本：787×1092 1/16

1995 年 9 月第一次印刷 印张：23.875

印数：**1—5000 册** 字数：551 千字

ISBN 7-03-004846-6/TP・469

定价：35.80 元

致　谢

我想借此机会向所有对本书的出版给予帮助的人表示感谢：感谢 Denise，Joseph 和 Kaitlin 的关心和支持；感谢 Tova 和 Robert 使我始终保持诚实的态度，并准时完成了任务；感谢 Robert X. Cringely 所提供的写书秘诀。最重要的是感谢 Jon Erickson，是他使这一切成为现实。

Mark Nelson

目 录

前言 本书的作用 …… (1)
第一章 数据压缩简介 …… (2)
1.1 读者 …… (2)
1.2 为什么用C …… (2)
1.3 用哪种C …… (3)
1.4 现有的记录 …… (5)
1.5 全书的结构 …… (6)
第二章 数据压缩词汇及其历史 …… (8)
2.1 两个领域 …… (8)
2.2 数据压缩＝建模＋编码 …… (8)
2.3 萌芽时期 …… (9)
2.4 编码 …… (10)
2.4.1 改进 …… (11)
2.5 建模 …… (12)
2.5.1 统计模型 …… (12)
2.5.2 字典方案 …… (13)
2.6 Ziv 和 Lempel …… (14)
2.6.1 LZ77 …… (14)
2.6.2 LZ78 …… (14)
2.7 有损压缩 …… (14)
2.8 需要了解的程序 …… (15)
第三章 萌芽时期：最小冗余度编码 …… (17)
3.1 Shannon-Fano 算法 …… (18)
3.2 Huffman 算法 …… (20)
3.3 用C语言实现 Huffman 算法 …… (22)
3.3.1 BITIO.C …… (23)
3.4 关于函数原型的说明 …… (29)
3.5 MAIN-C.C 和 MAIN-E.C …… (30)
3.5.1 MAIN-C.C …… (35)
3.5.2 ERRHAND.C …… (35)
3.6 Huffman 代码剖析 …… (37)
3.6.1 符号计数 …… (37)
3.6.2 保存计数 …… (38)
3.6.3 建立 Huffman 树 …… (39)

3.6.4 使用 Huffman 树 …… (39)
3.7 压缩代码 …… (40)
3.8 合并所有的模块 …… (52)
3.8.1 性能 …… (53)
第四章 重大改进:自适应 Huffman 编码 …… (55)
4.1 自适应编码 …… (55)
4.2 更新 Huffman 树 …… (56)
4.2.1 交换的作用 …… (59)
4.2.2 算法 …… (59)
4.2.3 一种增强方式 …… (60)
4.2.4 转义码 …… (60)
4.2.5 溢出问题 …… (61)
4.2.6 按比例缩小的优点 …… (64)
4.3 代码说明 …… (64)
4.3.1 数组的初始化 …… (65)
4.3.2 压缩主程序 …… (66)
4.3.3 还原主程序 …… (66)
4.3.4 对符号编码 …… (67)
4.3.5 更新树 …… (69)
4.3.6 解码符号 …… (73)
4.4 代码 …… (73)
第五章 优于 Huffman:算术编码 …… (85)
5.1 难点 …… (85)
5.2 算术编码:前进一步 …… (85)
5.2.1 实际情况 …… (88)
5.2.2 复杂性 …… (90)
5.2.3 解码 …… (91)
5.2.4 算术编码的优势 …… (91)
5.3 代码说明 …… (92)
5.3.1 压缩程序 …… (92)
5.3.2 还原程序 …… (93)
5.3.3 初始化模型 …… (94)
5.3.4 读入模型 …… (96)
5.3.5 初始化编码程序 …… (97)
5.3.6 编码过程 …… (97)
5.3.7 编码程序复位 …… (99)
5.3.8 解码过程 …… (99)
5.4 概述 …… (101)
5.5 代码 …… (101)

第六章　统计模型……(116)
6.1　更高次序的模型 ……(116)
6.2　有限上下文模型 ……(116)
6.3　自适应模型 ……(117)
6.3.1　一个简单例子 ……(117)
6.3.2　用转义码表示后退(fallback) ……(120)
6.3.3　改进 ……(122)
6.4　最高次序的模型 ……(122)
6.4.1　更新模型 ……(123)
6.4.2　转义概率 ……(123)
6.4.3　计算板 ……(124)
6.4.4　数据结构 ……(125)
6.4.5　完成修补：表 1 和表 2 ……(127)
6.4.6　模型刷新 ……(128)
6.4.7　实现 ……(128)
6.5　结论 ……(128)
6.5.1　增强 ……(128)
6.6　ARITH-N.C 清单 ……(129)
第七章　基于字典的压缩……(155)
7.1　一个例子 ……(155)
7.2　静态方法与自适应方法 ……(156)
7.2.1　自适应方法 ……(156)
7.2.2　一个典型例子 ……(157)
7.3　起源于以色列 ……(159)
7.3.1　历史 ……(159)
7.4　ARC：MS-DOS 字典压缩之父 ……(160)
7.4.1　字典压缩的应用领域 ……(160)
7.5　潜在危险——专利 ……(161)
7.6　结论 ……(162)
第八章　滑动窗口压缩……(163)
8.1　算法 ……(163)
8.1.1　LZ77 的问题……(166)
8.1.2　编码问题 ……(167)
8.2　LZSS 压缩 ……(167)
8.2.1　数据结构 ……(168)
8.2.2　平衡操作 ……(170)
8.2.3　贪婪与最有可能 ……(171)
8.3　代码说明 ……(172)

8.3.1 常量和宏 …… (172)
8.3.2 全程变量 …… (173)
8.4 压缩代码 …… (174)
8.4.1 初始化 …… (176)
8.4.2 主循环 …… (176)
8.4.3 退出代码 …… (178)
8.4.4 AddString() …… (178)
8.4.5 DeleteString() …… (181)
8.4.6 二叉树支持函数 …… (182)
8.5 还原函数 …… (183)
8.5.1 改进 …… (185)
8.6 代码 …… (185)
第九章 LZ78 压缩 …… (194)
9.1 LZ77 可以改进吗? …… (194)
9.2 进入 LZ78 …… (195)
9.2.1 LZ78 详述 …… (195)
9.2.2 LZ78 实现 …… (197)
9.3 有效的变体 …… (198)
9.4 还原 …… (200)
9.4.1 缺陷 …… (201)
9.4.2 LZW 实现 …… (202)
9.4.3 树的维护和搜索 …… (202)
9.5 压缩 …… (204)
9.6 还原 …… (205)
9.7 代码 …… (207)
9.8 改进 …… (211)
9.9 专利 …… (219)
第十章 语音压缩 …… (220)
10.1 数字音频的概念 …… (220)
10.1.1 基础 …… (220)
10.1.2 采样变量 …… (224)
10.1.3 基于 PC 的声音 …… (226)
10.2 声音的无损压缩 …… (226)
10.2.1 问题和结果 …… (227)
10.2.2 有损压缩 …… (229)
10.2.3 静止压缩 …… (229)
10.3 压扩(companding) …… (235)
10.4 其他技术 …… (242)
第十一章 有损的图形压缩 …… (243)

11.1 进入压缩 …… (243)
11.1.1 统计和字典的压缩方法 …… (244)
11.1.2 有损压缩 …… (244)
11.1.3 差分调制 …… (245)
11.1.4 自适应编码 …… (245)
11.2 一个可行的标准:JPEG …… (246)
11.2.1 JPEG 压缩 …… (246)
11.2.2 离散余弦变换 …… (247)
11.2.3 DCT 的详细说明 …… (248)
11.3 问题之所在 …… (249)
11.4 DCT 的实现 …… (250)
11.4.1 矩阵相乘 …… (250)
11.5 继续改进 …… (252)
11.5.1 DCT 的输出 …… (252)
11.5.2 量化 …… (253)
11.5.3 选择量化矩阵 …… (254)
11.6 编码 …… (255)
11.6.1 曲徊序列 …… (256)
11.6.2 熵编码 …… (257)
11.6.3 如何处理颜色 …… (258)
11.7 样本程序 …… (258)
11.7.1 输入格式 …… (259)
11.7.2 代码 …… (259)
11.7.3 初始化 …… (260)
11.7.4 正向 DCT 函数 …… (261)
11.7.5 WriteDCTData() …… (262)
11.7.6 OutputCode() …… (263)
11.7.7 文件还原 …… (265)
11.7.8 ReadDCTData() …… (266)
11.7.9 输入 DCT 代码 …… (266)
11.7.10 逆 DCT …… (267)
11.8 完整的代码清单 …… (268)
11.9 支持程序 …… (280)
11.10 一些压缩结果 …… (284)
第十二章 一个归档软件包 …… (287)
12.1 CAR 和 CARMAN …… (287)
12.1.1 CARMAN 命令集 …… (288)
12.1.2 CAR 文件 …… (289)
12.1.3 文件头 …… (289)

12.1.4 存贮文件头……(290)
12.1.5 文件头的 CRC ……(292)
12.1.6 命令行的处理……(293)
12.2 产生文件清单……(295)
12.2.1 打开归档文件……(299)
12.3 主处理循环……(300)
12.3.1 跳过/拷贝输入文件 ……(304)
12.3.2 文件插入……(305)
12.3.3 文件抽取……(306)
12.3.4 消除……(308)
12.4 代码……(308)
附录 A 压缩程序的统计 ……(351)
附录 B 测试程序 ……(355)
词汇表……(364)
参考文献……(370)
其他资源……(371)
后记……(372)

前言　本书的作用

如果你想知道像 PKZIP 和 LHarc 这样的程序是如何工作的，则这本书将十分有用。这些程序中使用的压缩技术在本书中作了详细的描述，并附有工作代码。读完本书后，即使是初学 C 的程序员也能够写出实际上可以移植到任何操作系统或硬件平台上的一个完整的压缩/归档程序。

如果想在自己编写的程序中包含数据压缩，本书将是一个非常有用的工具。它包含可以方便地加入到应用程序中的许多 C 代码的工作程序。各种压缩方法的深入讨论将有助于在创建使用数据压缩的程序时作出明智的决定。

如果想知道为什么图形有损压缩是促成多媒体革命的关键因素，便需要这本书。书中详细描述了类似于 JPEG 算法所使用的基于 DCT 的压缩技术。工作代码则允许人们用这种非常吸引人的新技术进行实验。

本书提供了这一重要领域的综合参考，目前还没有一本书详细描述压缩算法或者这些算法的 C 语言的具体实现。如果你打算在这一领域工作，那么本书是必不可少的！

第一章　数据压缩简介

本书的主要目的是用C编程语言来解释各种数据压缩技术。数据压缩的目的是减少用于存贮和传输信息的位数。它包括广泛的彼此不同的软件和硬件压缩技术，这些技术除了压缩数据之外没有什么共同之处。例如，用在Compuserve GIF规范中的LZW算法与用于压缩电话线上数字化语音的CCITT G.721规范基本上没有什么共同之处。

本书不会全面浏览每种数据压缩技术，因为该领域在过去的25年中已经有了难以置信的发展。本书将只涉及常用于个人机和中型机的各种数据压缩，包括二进制程序、数据、语音和图形的压缩。

此外，本书将忽略或者是略微涉及实际使用中依赖于硬件的或需要硬件应用的数据压缩技术。当今的许多语音压缩方案是为世界范围的固定带宽数字通讯网络而设计的，但它们需要一种特定类型的硬件以调整到通讯通道的固定带宽。不必满足这种需求的各种不同算法在PC上用于压缩数字化语音，而且这些算法普遍提供了更好的性能。

然而，当今数据压缩中的一些最有趣的领域的确关联着使用新的和更强有力的硬件才能实现的压缩技术。例如，用于多媒体系统的有损图像压缩，现在可以在标准桌面平台上实现。本书将介绍用这些技术中所使用的一些算法进行实验并加以实现的一些实际方法。

1.1　读　者

读者需要掌握一些基本的编程技术以深入研究数据压缩代码。理解块结构代码(如C或Pascal)的能力是必不可少的；另外，很好地理解计算机的体系结构，以便懂得面向位的操作，如移位、逻辑或(OR)和与(AND)操作等，也是必需的。

这并不意味着为了读懂本书就必须是一个C语言大师，甚至也不必是一个程序员。但是理解代码的能力是必需的，因为这里讨论的概念将用可移植的C程序来演示。本书中的C代码编写的着眼点是简单化，希望C的初学者也可以理解这些程序。我们将避免使用C的较复杂的结构，但代码将是C，而不是伪代码或英文。

1.2　为什么用C

使用C来演示数据压缩算法可能增加一些埋怨。一个更加传统的写这本书的方法是使用伪代码来描述算法。但是由于伪代码"程序"的不严格，像"PROCESS FILE UNTIL OUT OF DATA"这样的语句常导致含义不清楚或定义不完整。结果是，伪代码易读，但并不容易翻译成工作程序。

如果不满意伪代码，下一个最好的选择是使用常规的编程语言。虽然存在几百种选择，但是有充分的根据，C似乎是这类书的最好选择。首先，在许多方面C语言已经成为程序员的共同语言。C编译程序支持的计算机范围从低级的8501微控制器到每秒100兆指令(MIPS)的

超级计算机的事实已经说明了这一点。这并不意味着C是所有程序员都选择的语言，这只是表示对于大多数程序员，他们的计算机都应该有一个C编译程序，并且大多数程序员可能经常地使用C代码。正因为如此，许多使用其他语言的程序员仍可以用C来编写代码，而且更多的程序员至少可以读懂C代码。

使用C的第二个原因是该语言没有多大的奇特之处，它用作基本语言元素的少数几个结构可以很容易地翻译成其他语言。所以一个使用C来演示的数据压缩程序，通过一个相当直观的翻译过程就可以转换成一个Pascal工作程序。甚至汇编语言程序员也会发现这个过程并不费劲。

使用C语言最重要的原因可能只是效率问题。C常常被认为是高级汇编语言，因为它允许程序员接近硬件。虽然在当今的C编译程序中可以发现日益增强的优化特性，但是C语言在速度和大小上超过手编汇编语言的可能性是不存在的。然而，由于能方便地将C代码移植到其他机器上去，所以这个缺陷得以弥补。因此对于这类书，C语言可能是最有效的选择。

1.3 用哪种C

虽然C被宣传成一种"可移植"的语言，但是在给定机器上编译和执行的一个C程序并不能保证在任何其他机器上运行，甚至不能在相同的机器上使用不同的编译程序来编译。需要记住的重要事情是，并不是C语言可移植，而是可以对它进行移植。本书的代码都书写成是可移植的，并且在几种不同的编译程序和环境下都可以顺利地进行编译和运行。这里使用的编译程序/环境包括：

- Microsott C 6.0，MS-DOS 3.3/5.0
- Borland C++2.0，MS-DOS 3.3/5.0
- 带有286或386 DOS扩展驱动程序的Zortech C++，MS-DOS 3.3/5.0
- 带有可移植的C编译程序的交互式UNIX System V3.2

一个重要的可移植性问题是库函数调用。虽然C编程语言在K&R最初的著作(Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language [Englewood Cliffs, N. J.: Prentice-Hall, 1978])中相当好地定义了，但运行库的实现被全部留下以满足实现者的奇想。幸好，美国国家标准学会(American National Stadards Institute, ANSI)在1990年完成了C语言规范，其结果作为ANSI标准XJ11.34出版。这个标准不仅扩展和符合了原始的K&R语言规范，而且进行了一个标准C运行库的定义。这就使得便于书写在不同机器之间以相同方式工作的代码。本书中的代码在只使用ANSI C库函数调用的意图下写成，尽可能避免与编译程序有关的对语言或库的扩展。

给出库的标准后，余下的可移植性问题便围绕着两件事：基本数据类型的大小和处理不一致的编译程序。在16位和32位机器间切换时，大多数数据类型会出现冲突。

幸好，在16位和32位机器间处理这些改变是相当容易的。虽然基本的整型数据类型在16位和32位机器之间不同，但两类机器都有一个16位的"短整型"(short int)数据类型，而且一个"长整型"(long int)在两类机器上通常都是32位。所以当整数的大小确实十分重要时，可以用适当的声明来指明它是16位还是32位。

在当今世界上所用的绝大多数机器上，C编译程序实现的"字符"(char)数据类型都是8

位宽。在本书中，我们将忽略任何其他大小存在的可能性，并坚持 8 位的字符。一般地说，把本书所展示的程序移植到带有不常见字符大小的机器上并不是很困难，但在这上面花费太多的时间会混淆这里的程序重点，即数据压缩。

在书写可移植代码时要处理的最后一个问题是不一致的编译程序问题。在 MS-DOS 领域中，大多数 C 编译程序每两年左右就经历主要的释放和升级，这意味着大多数编译程序供应商已经可以发行他们目前接近于 ANSI C 标准的编译程序新版本。但这不是为许多其他操作系统用户而做的事。特别是 UNIX 用户，将一直频繁地使用与其系统一同提供并且符合较老的 K&R 语言定义的 C 编译程序。虽然 ANSI C 委员会努力使从 K&R 中产生的 ANSI C 向上兼容，但是我们仍要注意几个问题。

第一个问题是函数原型的使用。在 K&R C 中，函数原型通常是在必要时才使用，编译程序假设任何一个没有看到的函数都返回一个整型值，而且毫无怨言地接受这一点。如果一个函数返回一些不寻常的东西，例如一个指针或一个长整数，程序员就要写一个函数原型来通知编译程序：

```
long locate_string();
```

这里，原型告诉编译程序生成使函数返回一个长整数而不是整数的代码，函数原型除此之外没有什么更多的用处。正因为如此，工作在 K&R 规范下的许多程序员很少用或不用函数原型，而且它们在程序中也很少出现。

虽然 ANSI C 委员会试图不去改变 C 的基本性质，但是他们不能拒绝对语言的潜在改进，而这些改进能够通过原型机制的扩展来实现。在 ANSI C 中，函数原型不仅定义该函数返回的类型，还定义所有参数的类型。例如，前面所示的函数使用 ANSI C 编译程序可能有下面的原型：

```
long locate_string (FILE * input_file,char * string);
```

这就使编译程序为返回类型生成正确的代码，并同时检查变量的正确类型和数目。因为传递错误的变量类型和变量数目给一个函数是 C 程序中程序员最易犯的错误，所以该委员会正确地预见到允许这种形式的类型检查会使 C 语言前进一步。

在许多 ANSI C 编译程序下，使用完整的 ANSI C 函数原型得到很好的鼓励。事实上，如果事先没有遇到函数原型而使用的话，许多编译程序会产生警告信息。这样做确实很好，但同样的函数原型在 UNIX 完全可信赖的可移植 C 编译程序下却不能工作。

对于这个疑难问题的解决并不是很完美，但它是确实可行的。在 ANSI C 下，预定义的宏 ___STDC___ 定义总是用于指明代码是通过一个可能符合 ANSI 的编译程序来编译的。我们可以用预处理程序把头文件中的特定部分打开或关闭，这要根据是否正在用一个不符合 ANSI C 标准的编译程序而定。例如，含有面向位的程序包原型的头文件可能包含如下内容：

```
#ifdef ___STDC___
```

```
FILE * open_bitstream (char * file_name, char * mode);
void close_bitstream(FILE * bitstream);
int read_bit(FILE * bitstream);
int write_bit(FILE * bitstream, int bit);

#else

FILE * open_bitstream();
void close_bitstream();
int read_bit();
int write_bit();

#endif
```

预处理指令对代码的外观并不起什么作用，但它们是书写可移植程序的一个必要部分。由于本书中的程序都假设是在编译程序的警告水平设置为最高的情况下编译的，所以一些“#ifdef”语句将是软件包的一部分。

C 编译程序的 K&R 家族的第二个问题在实际的函数体中。在 K&R C 中，一个特定的函数可能有如下的定义：

```
int foo(c)
char c;
{
/* Function body */
}
```

使用 ANSI C 函数体书写的同一个函数可能是：

```
int foo(char c)
{
/* Function body */
}
```

这两个函数可能看上去一样，但 ANSI C 规则要求对它们区别对待。K&R 函数体将使编译程序在函数体使用字符变量以前，把它“提升”为一个整型变量，而 ANSI C 函数体则将它留作一个字符。把一种整数类型“提升”为另一种类型会使许多隐藏着的问题嵌入到看上去写得很好的代码中，而更为严格的编译程序探测到这种性质的问题时会发出警告。

由于 K&R 编译程序不接受第二种格式的函数体，所以在定义函数的字符变量时要小心。遗憾的是，解决方法又是要么不使用字符参数，要么承受更多丑陋的“#ifdef”预处理程序包。

1.4 现有的记录

“压缩率”和压缩统计的引用贯穿于全书之中。为了使各种压缩格式在同一个水平的竞技

场上，压缩统计总是与“Dr. Dobb's Journal”(1991 年 2 月)压缩比赛中使用的样板压缩文件有关。这些文件由大约 6 兆字节的数据组成，这些数据分成三个大致相等的类。第一类是正文，由手写脚本、程序、备注记录和其他可读文件构成；第二类由二进制数据组成，包括数据库文件、可执行文件和电子表格数据；第三类由用原始屏幕转储格式存储的图形文件组成。

本书中建立和讨论的程序将用三种粗略的方法来判断性能。第一个是程序在压缩过程中消耗的内存量，这个数字将尽可能准确地近似；第二个是程序在压缩整个 Dr. Dobb 数据集时所用的时间量；第三个是整个数据集的压缩率。

不同的人使用不同的公式来计算压缩率，有些人喜欢用每字节位数(bits/byte)，而有些人使用比率，如 2:1 或 3:1(做广告的人好像喜欢这种格式)。在本书中，我们将使用一个简单的压缩百分比公式：

(1－(压缩后的大小/原大小)) * 100

它表示压缩时没有一点变化的文件有一个百分之零的压缩率；压缩到原来大小的三分之一的文件有一个百分之六十七的压缩率；缩小到 0 字节的文件(!)有一个百分之百的压缩率。

这种测量压缩的方法可能并不完善，但它在 100%时说明完善，而在 0%时则说明完全失败。事实上，经过压缩程序后变大的文件将显示一个负的压缩率。

1.5 全书的结构

全书共十二章，并有一盒配套软盘(单卖)。其内容编排大致相应于数据压缩的历史发展过程，即开始于 1950 年左右的“萌芽时期”，并持续到现在。

第二章是一个参考章节，试图建立基本的数据压缩词汇。它讨论了信息论的产生，还介绍了在本书其他部分多次用到的一系列概述、术语、专门用语和理论。即使你是一个数据压缩的初学者，掌握第二章将会使你达到参加信息的“鸡尾酒会”的水平，这意味着即便没有完全理解其中错综复杂的事物，但仍能进行关于数据压缩的有共同语言的谈话。

第三章以可变长度的位编码开始，讨论数据压缩的诞生。Shannon-Fano 编码和 Huffman 编码的发展代表着数据压缩和信息论的诞生，这些编码方法今天仍在广泛使用。另外，本章还讨论了建模(modeling)和编码(coding)——数据压缩硬币的两个面的不同。

标准的 Huffman 编码在用于高性能数据压缩时有一个较大的问题。压缩程序必须将 Huffman 编码统计的一个完整拷贝传递给还原程序。随着压缩程序累积更多的统计并试图增加它的压缩率，统计值将占据更多的空间和工作，从而抵消了进一步的压缩。第四章讨论了解决这个疑难问题的方法：自适应 Huffman 编码。这是一种较新的变革，归因于对 CPU 和内存的需求。自适应编码大大扩展了 Huffman 编码的范围，从而极大地改进了压缩率。

Huffman 编码必须为每个代码使用整数个位，这通常并非最优。一种更新的改进——算术编码，为每个代码使用小数值的位数，使得它进一步改进了压缩性能。第五章解释这种新改进是如何工作的，并展示如何把算术编码程序和统计模型集成在一起。

第六章讨论统计模型。无论是使用 Huffman 编码、自适应 Huffman 编码还是算术编码，都需要一个统计模型来驱动编码程序。这一章展示一些使用有限的内存资源来实现强有力模型

的有趣技术。

字典压缩方法采用与前四章中所讨论的技术完全不同的方法。第七章是关于用单个代码代表一串字符的压缩方法的综述。由于具备与合理的内存需求相关联的高性能压缩，字典方法实际上已经成为小型机上通用数据压缩的一个事实标准。

基于字典的压缩之父 Ziv 和 Lempel 于 1977 年发表了一篇论文，提出了一种目前已经变得非常流行的数据压缩的滑动字典方法。第八章着眼于用于流行归档程序(如 PKZIP)中的 LZ77 压缩的最新变化。

第九章详细考查了最早广泛流行的基于字典的压缩方法之一：LZW 压缩。LZW 是用在 UNIX 的 COMPRESS 程序以及 MS-DOS 的 ARC 程序早期版本中的压缩方法。这一章还将介绍由 Ziv 和 Lempel 于 1978 年发表的 LZW 压缩的基础知识。

直到第九章所讨论的所有压缩技术都是“无损的”(lossless)。有损的方法可以用在语音和图形中，并且能够达到极高的压缩率。第十章展示如何使用像线性预测编码和自适应 PCM 这样的技术，将有损压缩应用在数字化语音数据上。

第十一章讨论用于计算机图形的有损压缩技术。工业界正在对仍未完成的 JPEG 标准快速地进行标准化。用在 JPEG 标准中的技术在这一章中介绍。

第十二章讲述如何将所有的一切放到一个归档程序中。一个通用归档程序应该能够压缩和还原文件，并且同时保留文件名、日期、属性、压缩率和压缩方法。一种归档格式应该可以理想地移植到不同类型的机器上。第十二章开发了一个样板归档程序，它把前面几章中使用的技术放在一起，形成一个完整的程序。