计算机实用技术系列丛书　　学苑出版社

数据压缩实用技术

骆新 陈睿　编著

计算机实用技术系列丛书

数据压缩实用技术

骆 新 陈 睿 编著
熊可宜 审校

学苑出版社
1993

(京)新登字 151 号

内容提要

本书主要介绍了有关数据压缩的简史，理论的依据，以及当前许多知名的压缩方法，并有详尽的程序范例．程序范例除了基本的压缩算法外，还包括当前最普遍的 LZH 压缩法与 ZIP 压缩法，并且运用这些方法设计一些数据文件的压缩器．

本书所有的应用实例都附上了完整的源程序，是计算机开发人员的有力参考资料．

欲购本书的用户，请直接与北京 8721 信箱联系，电话：2562329，邮政编码：100080.

计算机实用技术系列丛书

数据压缩实用技术

编　　著：骆　新　　陈　睿
审　　校：熊可宜
责任编辑：徐建军
出版发行：学苑出版社　　邮政编码：100032
社　　址：北京市西城区成方街 33 号
排　　版：北京天奥科技公司照排中心
印　　刷：　施园印刷厂
开　　本：787×1092　1／16
印　　张：13.9　　字数：330 千字
印　　数：15000
版　　次：1993 年 12 月北京第 1 版第 1 次
ISBN7—5077—0806—3／TP・17
本册定价：13.00 元

前　言

当前，随着存储媒体的价格日益昂贵，数据传输效率的要求，数据保密等等诸多因素，使得数据压缩的技术被广泛地应用在各类硬、软件的界面上。

本书主要将告诉您有关数据压缩的简史、理论的依据，以及当前许多知名的压缩方法，并且循序渐进地解说详尽的程序范例，引导您进入数据压缩的世界。这些程序范例除了基本的压缩算法外，还包括了当前最普遍的 LZH 压缩法与 ZIP 压缩法，并且运用这些方法设计一些数据文件的压缩器。

此外还利用 ZIP 压缩法设计一个 COM 可执文件的压缩器，此压缩器的效率可媲美目前市面上的可执行文件压缩器，其速度甚至比市面上的可执行文件压缩器还快，它可将 COM 执行文件压缩起来，并且该压缩文件在压缩状态下仍可执行，使用者无需加上额外的解压缩动作，是一个相当有用的压缩器。所有的应用实例都附上完整的源程序。

几乎所有的源程序都以 C 语言写成，适用于大多数的 C 编译器，但在设计可执行文件压缩器时，由于考虑到自我解压缩的运行速度的问题，因此用汇编语言来写自我解压缩程序，所以读者要了解程序写作方式，并应用到自己写作的程序上，需要具有 C 语言的基础知识。

本书附有一高效率的可执行文件压缩器，此压缩器可压缩一般的执行文件，使它们所占的内存空间变小，而无需解压缩仍可执行，并不影响它们的效率，详情请参见本书的第八章。

编　者

1993 年

目　录

第一章　数据压缩简介 …… 1
1.0　排字与打字机 …… 1
1.1　摩斯(Morse)电码 …… 1
1.2　一般文本的 TAB 压缩 …… 2
1.3　数据压缩的基本概念 …… 3
1.4　目前数据压缩技术的应用 …… 4
第二章　数据压缩理论 …… 5
2.0　数据量的定义 …… 5
2.1　乱度(Entropy) …… 7
2.2　多余信息量(Redundancy) …… 7
2.3　概率理论 …… 9
2.4　事件间的依赖性与 Markov 系统 …… 9
2.5　英文的 Redundancy …… 10
第三章　实际的数据压缩方法 …… 12
3.0　统计式压缩法(Statistical Compression) …… 12
3.1　代换式压缩法(Substitutional Compression) …… 28
3.2　各种压缩法的效率比较 …… 33
第四章　Huffman 压缩法 …… 35
4.0　建立 Huffman 树 …… 35
4.1　Huffman 压缩部分 …… 39
4.2　Huffman 解压缩部分 …… 44
4.3　主程序部分 …… 46
4.4　结语 …… 47
第五章　LZSS 压缩法 …… 48
5.0　二分搜寻法 …… 48
5.1　二分搜寻树建立与管理 …… 52
5.2　压缩部分 …… 58
5.3　解压缩部分 …… 67
5.4　主程序部分 …… 69
5.5　结语 …… 69
第六章　LZHUF 压缩法 …… 71
6.0　LZSS 与 Huffman 算法的优缺点 …… 71
6.1　HZHUF 的二分搜寻树 …… 71

6.2 Huffman 树的建立与管理 …… 72
6.3 压缩部分 …… 78
6.4 解压缩部分 …… 86
6.5 主程序部分 …… 94
6.6 结语 …… 94
第七章 ZIP 压缩器 …… 95
7.0 ZIP 压缩器简介 …… 95
7.1 ZIP 的字串搜寻方式 …… 95
7.2 缓冲区的配置 …… 104
7.3 压缩部分 …… 106
7.4 解压缩部分 …… 127
7.5 ZIP.H 头文件说明 …… 136
7.6 文件读写函数 …… 138
7.7 主程序部分 …… 140
7.8 结语 …… 142
第八章 可执行文件压缩器 …… 144
8.0 可执行文件压缩器的概貌 …… 144
8.1 DOS 运行 COM 可执行文件的方式 …… 144
8.2 MINICOM 主程序 …… 145
8.3 剖析自我解压缩码 …… 150
8.4 文件压缩后的格式 …… 162
8.5 结语 …… 163
第九章 数据压缩的艺术 …… 164
9.0 最大限度的压缩 …… 164
9.1 再谈 Arithmetic 压缩法 …… 165
9.2 压缩技术在 PC 上的未开发地 …… 165
9.3 提高压缩率的技巧 …… 166
附录 A 本书所有范例的源程序 …… 167
A.0 Huffman 压缩法源程序 …… 167
A.1 LZSS 压缩法源程序 …… 172
A.2 LZHUF 压缩法源程序 …… 179
A.3 ZIP 压缩法源程序 …… 193
A.4 MINICOM 可执行文件压缩器源程序 …… 214

第一章　数据压缩简介

基本上，数据压缩可概略地可分为模拟信息的压缩和数字信息的压缩，前者如声音、影像的压缩技术，后者如一般的文本数据的压缩方法。对于模似信息的压缩技术而言，通常被压缩后的数据就不能再被还原为和原来完全相同的数据。这种压缩技术被称为非精确压缩(Inexact Compression)，而数字信息的压缩则可做到能将被压缩的信息还原为和原来信息完全一样的信息，这种压缩技术则属于后者。

首先必须强调的是统计学和概率论在数据压缩的理论上扮演着相当重要的角色，因此为了使读者大致上有个了解，在详细讨论数据压缩理论之前，先介绍一些数据压缩技术的简史与类似的方法。

1.0　排字与打字机

还在几个世纪前，排字的工人便已知道某些符号字母或字体的使用频率较高，而某些符号则较少，利用这个特点，这些工人多铸造一点使用频率较高的字模，并且放在较容易取得的位置，而使用频率较少的符号则铸造的字模较少，并放在比较不方便的位置，这样便可以加快他们的排字工作。

现在的打字机也运用这种符号使用频率的特性，将使用频率较高的符号排在较灵活的指头(如食指、中指等)能按到的位置，而使用频率较低的符号排在较不灵活的指头按得到的位置，以便加快打字的速度。

这种依符号出现概率不同而作不同处理的应用，已有些"符号编码"的味道，因为若把所有的符号都一视同仁，不考虑符号的出现概率，而且每个符号都铸造出同样多的字模，或是在打字机上按照字母的顺序依次由上而下、由左向右地排列，则排字或打字的效率会很显然降低。

1.1　摩斯(Morse) 电码

最早将符号按使用频率不同的特性应用到通讯上的是著名的摩斯(Morse)电码。摩斯当初的构想是用简单的信号(一点与一线)来表示当时通用字母与符号，用以建立一套电报码系统。他打算将一点与一线最简单的组合分配给使用频率最高的符号，一点与一线最复杂的组合分配给使用频率最低的符号，这样可以减少通讯的时间。

首先，他参考了当时一份叫作 Philadelphia 的报纸，将所有出现在报纸上的符号做个统计，结果他发现，字母 e 约出现 12，000 次，字母 t 约出现 9，000 次，字母 a 约出现 8，000 次、o、m、i、s 约出现 6，400 次，其它符号各有各的出现频率，再根据其它的文件所作的统计发现也有相似的出现频率，少有例外，因此他将"一点"分配给字母 e，"一点一线"分配给字母 a，这样将一点一线越长的组合分配给以出现频率越小的符

号，详细的摩斯电码请参考表 1.0。

如果"一点"所需要的传输时间为一单位，而"一线"所需要的传输时间为三单位，则利用这个方法，一段长为一百个符号的句子平均约需要 940 单位时间。

在此顺便强调一点，上面提到的是一段"有意义的句子"，因为我们对各种符号的出现频率的统计是建立在有意义的句子上的， 如果以同样长度但排列毫无规则的符号组合来传输，也就是说，我们假设所有的符号的出现机会是一样的，则约需要 1160 单位的时间。因此，运用符号使用频率多少的特性，摩斯电码将传输时间节省了约 23%。

摩斯的方法即是一种利用统计方法压缩数据的实例，此方法为我们书后将提到的一种"统计压缩法"的基础。

表 1.0 摩斯电码

a	.	n	.
b	. . .	o	
c	. .	p	. .
d	. .	q	.
e	.	r	. .
f	. . .	s	. . .
g	.	t	
h	. . . .	u	. .
i	. .	v	. . .
j	.	w	.
k	.	x	. .
l	. . .	y	.
m	.	z	. .
1	.	6	. . . .
2	. .	7	. . .
3	. . .	8	. .
4	. . . .	9	.
5	. . . . .	0	

1.2 一般文本的 TAB 压缩

细心的读者会注意到，许多知名且功能强大的文字编辑器都可让使用者选择地设定一个名为"Blank compression"或是"TAB compression"的选项是 ON 或 OFF，比如说 PE2 就有一项"set blankcompress"的命令。

首先要说明的是"Blank compression"或"TAB compression"是相同的，都是以 TAB 控制码来减少使用的空格字符串数目，只是名称略有出入，另外，TAB 控制码也不仅仅用在文件编辑上，它也用在通讯上，这里只是为举例方便。

TAB 是 ASCII 码中的一个控制码名称，它的值是 9，它的作用在于根据目前光标的位置，将光标定在下一个位置座标是光标的起始位置加上 8 的整数倍的位置。假设屏幕的最左边位置的座标定为 0，当程序将字符串一个个显示到屏幕上时，若遇到 TAB 控制码，则将光标移到下一个为 8 的整数倍的位置，再继续显示下一个字符串。一个简单的 C 语言算法表达式为:

```
p += 8-(p - ((p >> 3) << 3));
```

其中 p 为目前光标的位置。

举例来说，当程序在显示一段讯息为"ABCD＼tEFGH"时，其中 ＼t 为 TAB 码，首先假设在显示信息前，屏幕的光标停留在 4 的位置，此时程序遇到了 TAB 控制码，光标将会被移到 8 的位置(下一个为 8 的整数倍的位置)之后再显示"EFGH"等字符串，整个信息的显示结果是"ABCD EFGH"。因此，TAB 也可算是一个光标定位控制码。

当我们输入一段信息为:

```
"        Chapter 1 How to ..."
```

时，注意到这段信息的第一个字母开始于水平位置 10 的地方，我们便可使用 TAB 码的特性，将这段信息以:

```
"＼t Chapter 1 How to ..."
```

的形式储存起来，这样，在这段信息里面节省了 8 个空格字符串（原来节省 9 个空格字符串，但还要扣掉一个 TAB 码)。

因此，利用 TAB 码，我们可在文件中节省许多空格字符串。这种方法即所谓的"空格压缩(Blank compression)"或"TAB 压缩"。

TAB 的标准间距是 8，但目前许多文本编辑器还允许使用者设定不同的 TAB 间距，以达到更好的空格字符串的压缩率。

1.3 数据压缩的基本概念

数据的可压缩性主要建立于数据的重复性，只有重复性的数据才有可能被压缩。

在人类的文字世界，一篇文章、一段乐曲、一场电影、历史事件等等，简单地说就是任何一段有意义的数据都会有相当的重复片段，有些甚至一而再、再而三地重复(比如说目前所谓的"流行"歌曲，整段歌词虽然很长，但算一算全部使用到的字数，通常都在歌词全部长度的十分之一到四分之一之间)。

在电脑的世界里，符号的表现方法通常是存储体的最小单位来表示，在 PC 上，常见的方式是以一个字节表示 256 种符号，对于中文码，一般是两个字节来表示上万个中文符号。在一个文件中，重复的字句随处可遇。而在电脑程序中，总是以一组有限的程序指令集来组合出一个有意的程序，因此重复的程序码也占了整个程序相当的比例。

总之，以有限的符号来描述一段有意义的数据，必然会产生许多的重复，而将重复性愈大的数据重新以愈短的码来编排便是数据压缩的基本精神所在。

1.4 目前数据压缩技术的应用

当今，由于存储媒介的昂贵、传输效率的要求、数据保密等诸多因素，使得数据压缩技术广泛地应用在各种硬软件界面上。

在通讯方面，人们将要传输的数据以软件或硬件的方式压缩处理后再传输，以节省昂贵的通讯费用。如目前有许多调制解调器(Modem)具备有数据压缩的功能，这些调制解调器在传输一段数据前先将数据压缩起来再传输出去，而接受该数据的调制解调器则将接收到的数据解压缩后再传给应用软件。另外，对于没有压缩功能的调制解调器，人们则将要压缩的文件先以压缩软件压缩之后再传出，而使用者接收到这些被处理过的文件时，利用同样的压缩软件将它还原(解压缩)，这样也可达到节省传输时间的目的。

在数据的储存方面，目前许多硬式或软式驱动器提供一种硬件界面，通过这种界面，当数据被写进磁盘时，会先被压缩起来，之后才真正写到磁盘上；而当应用程序要从磁片上读取数据时，该界面首先将数据解压缩后再传给应用程序。对于没有这种特殊硬件界面的用户，一些程序设计员也提供了一种功能相同的软件界面，该界面类似一种驱动程序，它常驻在内存，并且将所有对磁盘写入的动作搁置起来，要写入磁盘的数据先被界面程序压缩后，才真正写入磁盘上，同样地，界面程序也搁置所有的读取动作，由磁盘读回的数据会先被界面程序解压缩后再传给应用程序。

用来备份文件的驱动器也通常具有压缩的能力，通过压缩后的磁盘往往可存储比原来数据多一倍左右的数据。

此外，一般没有昂贵备份装置的使用者也大多将要备份的文件先使用压缩软件压缩之后，再存到磁盘上，这样可省下大量的磁盘费用。

许多电脑上的影像数据，由于过于庞大，且影像的重复性又高，因此通常是以压缩的方式存储在文件中。

在数据保密方面，由于被压缩的数据的类型已完全不同于原数据类型，因此，除非已确切知道该数据的压缩方式，并且也拥有该压缩方式的解压缩器，否则根本无法由已压缩的数据解得原来的数据，因此数据压技术还额外提供了数据保密的附加作用。

数据压缩的概念已广泛地被应用在软件设计上，诸如数据库、上述的软件压缩界面、一般文件压缩器、压缩型数据机的软件模拟器、可执行文件压缩等等。由于 CPU 的运作速度已大大提高，因此在时间与空间两者的考虑中，大多数人愿意以较廉价的时间换取较昂贵的空间，意即以压缩与解压缩所花的少量的时间来换取较大的存储空间。

第二章　数据压缩理论

本章将论述一些数据压缩的基本理论，并使用到一些简易的数学表达式，若读者不谙数学或对基本理论没兴趣或者急于得到实际的应用知识，可跳过本章，但作者建议你还是阅读一下为好。

数据压缩理论的重要性之一在于它提示供了我们一种评估方法，它让我们知道一段数据"理论上"可以被压缩到怎么样的程度。因此，当一个压缩算法被发展出来时，我们可利用理论上的压缩率加以比较，评估这个算法到底有多"完美"。

我们以后所要介绍的种种压缩方法都会以理论上的数学方式加以评估它的压缩率。

2.0　数据量的定义

通常一句话所包含的数据"量"并不易以客观的方式来表示，因为我们很难用一些单位(如公斤、公尺、磅、小时...)来描述一句话的数据量的多少。

但主观上，我们通常能够很直观地了解到一句话对我们的重要性(也可说是内含数据的量)，比如说，如果有人对你说"地球是圆的"，你可能不觉得它有什么重要性，换句话说，它包含的数据量很少，因为你早就知道这个事实。但如果有人突然来对你说"你被炒鱿鱼了"，你可能一听到这话就跳起来，大吼大叫就跑出去，也就是说，这句话对你很重要，也可说它包含的数据量多，而且你原来不知道。

注意到上面所举的例子中的"你早就知道"与"你原来不知道"，还有"地球是圆的"与"你被炒鱿鱼了"的发生概率。对于"地球是圆的"来说，它是一个已知的事实，且它也是必然的现象，也就是说，对于一个已知的事实和必然的现象，它发生概率是一。但对"你被炒鱿鱼了"这种事，对大多数人来说，发生的概率似乎很低，也不是"必然"的现象。由这两个例子，我们可以观察到，一句话所包含的信息的多少与这句话可能发生的概率与你是否已经知道有关。也就是说，愈不可能发生的事情被讲出来，所包含的信息量愈多(当然条件是这句不是谎话)，这对我们如何定义信息量相当重要。

再举一个更直接的例子，假设有一间办公室包含了九个办公间，它的格局如图 2.0 所示。

Y座标				
	3	A	B	C
	2	D	E	F
	1	G	H	I
		1	2	3 (X 座标)

图 2.0 包含了九个办公间的大办公室

当我们要指定一个办公室间时，我们可以直接使用办公室的代号 A、B、C、D、E、

F、G、H 或 I 来表示，或者我们也可以用座标(X，Y)的方式来表示．假如有个话务员负责将外来的电话转至各个办公间，那打电话进来的人可能会告诉话务员(1，1)、(1，2)、(1，3)、(2，1)、(2，2)、(2，3)、(3，1)、(3，2)或(3，3)办公间，我们假设每间办公室接至电话的机会是均等的，因此如果是第一种情形，不管是 A 至 I，每个外来电话可能转进的办公室概率都是 1／9 ．如果是第二种情形，当话务员听到第一数字时，这个数字可能是 1 到 3 中的一个，也就是说，第一个数字的概率是 1／3，同样的，第两个数字的概率也是 1／3 ．

现在，读者要注意的是，不管打电话进来的人用什么方式告诉话务员，他要表示的信息量是相同的．第一个方式的信息量多少必然等于第二个方式的信息量．但第一个方式告诉话务员一个 A 到 I 值，而第二个方式告诉话务员两个值(X，Y)，因此第二个方案的每一个值 X 和 Y 所包含的信息量相加起来，必然等于第一个方式所包含的信息量．

其次，读者可能会注意到，以第一种方式告诉接线生的数值的概率是 1／9，而以第二种方式告诉话务员的两个数值的概率分别都是 1／3 ，且 1／3＊1／3＝1／9 ，由此，我们得到一个相当重要的结论:

信息量是相加的，但相对应的信息概率是相乘的．

由这个结论，我们有了信息量的定义:

信息量 I ＝ −log p

其中，p 为某一事件的发生概率，I 为该事件所包含的信息量多少．我们选用 log 函数当作信息量的定义的原因是因为 log 函数满足了上面的结论:信息量是相加的，而相对应的概率是相乘的．加上一个负号的原因是因为概率值 p＜＝1，因此 log(p) 会得到负值，加上一个负号使得信息量为正值，比负的信息量有意义些．

现在我们已经有了信息量的定义，但这个定义代表的意思相当抽象化，一时还不容易明白它的意义，下面举个例子:

假设有个二进制系统，它的信息都以 0 和 1 来表示，且我们也假设 0 和 1 出现的概率是相同的，都是 1／2 ，因此我们很自然地取 2 为 log 函数的底，因为每个数字(0 或 1)出现的概率都是 1／2 ，因此:

−log2 1／2 ＝ 1

就代表了每数字所代表的信息量为 1 bit／digit(每个数字一个位)．

再举一个例子，英文字母共有二十六个，我们假设一篇文章里头仅有这二十六个字母组成，且每个字母出现的概率相等，也就是说，每个字母出现的概率是 1／26，则每个字母所含有的信息量为:

−log2 1／26 ＝ 4.7 bits／letter (每个字母 4.7 个位)

这个答案也告诉我们，如果仅要表示二十六字母，每个字母仅用 4.7 个位的空间来储存即可。

同样的，阿拉伯数字共有十个，我们也假设每个数字出现的概率相等，也就是 1／10 ，则每个数字所含有的信息量为:

−log2 1／10 ＝ 3.32 bits／digit(每个数字 3.32 个位)

这个答案也告诉我们，如果仅要表示十个阿拉伯数字，每个数字仅需用 3.32 个位的空间来存储即可．

最后，关于已经知道的事件，它的发生概率是 1 ，因此它所包含的信息量为:

$-\log 1 = 0$

如我们先前所讨论过的，没有包含任何信息.

2.1 乱度(Entropy)

在 2.0 节中所谓的信息量，换成另一个学术味较浓的名词，便是乱度(Entropy)，这里的 Entropy 和物理中热力学的 Entropy 虽然不尽相同，但有着异曲同工之妙，尽管这样，还是不可混为一谈.

通常 Entropy 所表示的意义就是混乱的程度，对于人类的各种信息而言， 若混乱的程度越高，则该信息所能表达的意义就越丰富，反之，若某信息的重复性很高，或者该信息的样式遵循某种规则，则该信息所包含的意义就不多，当然， 这里是以某种特定的观点来衡量的.

所以基本上，一组信息其内容"越乱"，则其所包含的信息量越高，反之，一组信息其内容越"整齐"，则它所包含的信息量就越低，这也是我们使用这个名词的原因之一，而一个单独的事件的 Entropy 也就是其信息量，就被定义为:

Entropy $H(s) = -\log p(Si)$

其中，p(Si) 代表某一符号的出现概率.

在实际应用上，我们一般是对一整段句子中所包含的信息量较感兴趣，而不是单一个字母或数字，而在整段句子中(或者说是一连串符号的总和会比较接近实际)，每个字母数字或符号所包含的信息量的平均值就是所谓的平均 Entropy.

对于一组符号 S1、S2、S3、...、Sn，假设他们在一段句子中出现的概率分别为 p(S1)、p(S2)、p(S3)、...、p(Sn)，则平均 Entropy 被定义为:

$$\text{平均 Entropy } H(S1, S2, \ldots, Sn) = -\sum ni = \sum [p(Si) * \log p(Si)]$$
$$= -[p(S1) * \log p(S1) + p(S2) * \log p(S2) +$$
$$p(S3) * \log p(S3) + \ldots + p(Sn) * \log p(Sn)]$$

根据统计，实际的英文字母在一般文章中出现的概率并不完全相同，如同 Morse 在统计他的电报码时对各个字母、符号所做的统计. 我们将实际的英文字母在一般文章中出现的概率代入上面 Entropy 的公式中会得到:

$$\text{平均 } H = -[p('A') * \log p('A') + p('B') * \log p('B') +$$
$$p('C') * \log p('C') + \ldots + p('Z') * \log p('Z')$$
$$= 4.05 \text{ bits / letter (每个字母4.05 个位)}$$

这个平均值比把每个字母出现的概率都看成相同时所得到每个字母含有 4.7 个位的信息量更接近实际些.

2.2 多余信息量(Redundancy)

另一个与乱度 (Entropy) 有关的信息量决定性因素是多余量或多余信息量(Redundancy)，Redundancy 在英文的字义为过多、冗长、多余的意思，简单地说，多余量在此

所代表的意义可简述为:

在一个句子中，包含的字母个数比实际需要的字母个数多多少。

例如，一段句子"我昨 X 二十一点 Y 家"，在这段句子中，虽然 X，Y 的部分没有明显地指出，但一般人都能明了它的意思大概是说"我昨晚二十一点回家"。或"我昨天二十一点回家"。这就是说所谓的多余量，象 X、Y 这类的多余信息愈多，它内含的多余信息量越大。

多余量虽名为多余，但它也相对地提高了信息的保护作用，比方说，在一个仅用两个符号 X 与 Y 的系统中，相同系统若要以二进制方式传输这两种符号，它可以这样编码:将 X 编码为 [000] 或是 [111]，这样在传递过程中，若有杂讯干扰，我们也可以容忍一个位数的错误，例如，[000] 在传输过程中，一个位被干扰成 [010]或 [100]，我们也可根据 [010] 或 [100] 中有两个'0'而判断接收到的是一个 X。

由上一个范例，我们可将多余量定义为:

$$\text{Redundancy} = \frac{\text{Maximum Entropy} - \text{Actual Entropy}}{\text{Maximum Entropy}}$$

在上例中，Maximum Entropy 表示 [000] 或 [111] 所包含的信息量，因为不管是 [000] 或 [111]，0 与 1 的出现概率假设都为 1/2，所以连续三个 0 或 1 的概率为 1/2*1/2*1/2 =1/8，而它的 Entropy：

Maximum Entropy = −log2 1/8 = 3

但实际 X 和 Y 两个符号的出现频率都为 1/2，因此 Actual Entropy 所表示的就是实际符号所包含的平均信息量，因此:

Actual Entropy = −log2 1/2 = 1

而上述范例中，X、Y 符号所构成的系统其多余信息量就是:

Redundancy = (3−1)/3 =2/3

这个 2/3 所表示的意义是:

因为三个二进制数字所代表的信息量应该是三个位的量，但实际上，它仅代表一个二进制数字的信息量，有 2/3 部分是多余的。

如果上述的系统在传输 X、Y 符号时，采用 X 编码为 [0]，Y 编码为 [1] 的方式，则它的:

Maximum Entropy = −log2 1/2 =1

Actual Entropy = −log2 1/2 = 1

Redundancy = (1−1)/1 = 0

完全无多余的信息量，因为一个二进制数字(0 或 1) 所代表的信息量应该是一个位的量，而实际上，它也正代表一个二进制数字的信息量(X 或 Y)。但这样，该系统就完全没有抗干扰的能力了。

一般人类的语言发音中就含有相当巨额的多余信息量，例如在十分嘈杂的环境中，虽然彼此交谈时，对方讲的内容并没有字字句句都听得清楚，但还是能够由片段字句中了解对方的意思。

2.3 概率理论

前面提到过不少任意事件的出现概率的问题(像"个别的英文字母在整篇文章中出现的概率"或是十个阿拉伯数字个别的出现概率等等)，事实上，当我们提到一个事件的出现概率，我们指的是一个事件在一大堆数不清的事件中所出现的概率(比如说当我们提到"A"的出现概率，我们所指的是在一篇又臭又长的文章中"A"出现的概率，这篇文章越长，"A"的出现概率越普遍化、一般化，也可以说越准确吧)。因此，像字母"A"的出现概率，我们一般是定义为:

$$P('A') = \frac{'A'在这篇文章中出现的次数}{一篇文章中全部字母的数量}$$

但严格地说，这个定义还不够一般化，因为所谓"一篇文章"的长度、普遍性等等都是问题，因此，更严格的定义是将"一篇文章"的长度延长到无限大，因此，下的定义会更一般化些:

令 N = 所有事件出现的总和(如文章中的字母总数)

$$P(Si) = \lim_{N\to\infty} \frac{事件\ Si\ 出现的次数(如文章中的某个字符串)}{N}$$

上式定义了一个事件出现的概率值，但实际上，我们不可能计算"无限多"个事件(也就是说，不可能有一篇无限长的文章让你算个够)，因此，我们仅能尽可能合理地增加上式中 N 的量值，使得到的概率值更遍化一般化些。

2.4 事件间的依赖性与 Markov 系统

在真实的世界中，每个单一的事件并不都是各自独立的，它们多少都受到前一个或后一个或某个事件的影响。比如说，在所有的英文单字中，'TH' 的组合远多于'HT'的组合，也就是说，当我们考虑'T'这个字母时，不单只考虑它在二十六个字母中的平均出现概率，我们还要考虑'T'的前后各是什么字母。事实上，这种依存关系不仅仅发生在两个字母之间，象单字的某个区段(如 tion、ary、tive、ful...)、整个单字、甚至整段句子都会因为前后的内容而决定该段单字、句子在整篇文章中的出现概率，因此我们不能只考虑单独的字母的出现概率。

现在，我们考虑一个系统，它所使用的符号有着最简单的两个符号的依存关系，也就是说，单一符号的前面的单一符号会影响到该单一符号的出现概率。这种系统我们称它为 first-order Markov source(第一级 Markov 系统)，因为我们必须考虑到符号前的"一个"符号。

至于前面我们提到的那种仅考虑每个字母的平均出现概率，而不考虑符号间的依存关系的系统称为 zero-order Markov source (第零级 Markov 系统)，因为我们不需考虑符号前任何符号，而将每个字母视为各自独立、毫无关联的个体。

现在为简便明了起见，假设我们有一个 first-order Markov Source(第一级 Markov 系统)，这个系统使用 n 个符号，要计算这个系统的 Entropy，我们需要将所有可能出现的 n 个符号的组合概率统计出来(比如说，我们使用 0 和 1 两个符号，则我们得先统计出 00、01、10 和 11 各种配对出现的概率)。

假设 p(x，y)为 xy 型配对的出现概率，则:

Average H(x，y) = $-\sum_x\sum_y$ [p(x，y)。logp(x，y)]

这里的 H(x，y)表示在这种系统中每个符号所代表的平均信息量，这个 H(x，y)又称为条件乱度(Conditional Entropy)。

同样地，我们可以扩展这个方法到一个 second-order Markov source(第二级 Markov 系统)，在这个系统中，一个符号的出现概率是决定于前两个符号的，因此我们的 Conditional Entropy 则为:

H(x，y，z) = $-\sum_x\sum_y\sum_z$ [p(x，y，z)。log[p(x，y，z)]

其中，p(x，y，z)为 xyz 型配对的出现概率。

对于更高级的 Markov 系统(n-order Markov source)我们也可以依此类推，得到更普遍化的平均 Entropy。

下列，我们把由二十六个英文字母加上一个空格字符串共二十七个符号所组成的系统视为不同级的 Markov 系统，并计算以不同的观点来看待此系统时，所得到平均每个符号所含有的信息量。

一、zero-order Markov source

每个符号分别独立，且每个符号出现概率视为相同

4.75 bits / letter(每个符号 4.75 个位)

二、zero-order Markov source

每个符号分别独立，每个符号依实际生活中出现概率代入 Entropy 公式

4.05 bits / letter(每个符号 4.05 个位)

三、first-order Markov source

考虑两个符号间的依存关系

3.32 bits / letter(每个符号 3.32 个位)

四、second-order Markov source

考虑三个符号间的依存关系

3.10 bits / letter(每个符号 3.10 个位)

由此我们可以预测，当不断地提高 Markov 系统级数时，所得到的 Entropy 会逐渐逼近一个极值，那个极值也表示一个符号系统能达到的最高压缩率。

2.5 英文的 Redundancy

当我们收到一个这样的信息"INFORMATIO"时，毫无疑问，下一个字母必然是'N'，任何会使用英语的人都会作这样的猜测，这个例子只是简单地告诉我们一个事实，人类使用的语言的多余信息量是很大的。

在 1949 年，Shannon 和 Weaver 两位研究英文的多余信息量，他们参考大量的文

件，并把英文视为 second-order Markov source(第二级 Markov 系统，也就是考虑三个字母间的依存关系)，他们花很多时间统计出所有三个英文字母可能出现的组合与相对应的出现频率，并求得 H(x，y，z) = 3.10bits／letter(每个字母 3.10 个位)。

应用更普遍化、更趋近的平均值（包括计算单字、文句与片语在文章中的出现概率）有人得到了英文的平均 Entropy H＝2.10 bits／letter(每个字母 2.10 个位)。

最后，为了要求得"全部"英文的平均 Entropy，有些"志愿者"接受一些测验，这些测验包含了非常多的文章样本，但每篇都不完整，而这些"志愿者"的工作就是去"猜"那些在文章中漏失的部分。这样的测验主要是根据志愿者能由残缺不全的文章中，"猜"出多少原文的内容，进而统计出一篇文章中多余信息量的多少，然后计算出 Entropy 的量值。

最后统计结果发现，在所有的文章样本中平均出来的 Entropy 通常落在 0.6 和 1.3 bits／letter 之间，这显示了英文的多余信息量有多大了，这也就表示，电脑中所储存的英文文件数据，实际上只要花费目前存储空间的 1／8(每个字母只要一个位的存储空间)就够了，由此也揭示了信息压缩的重要性。