LZ77压缩算法及其派生算法探究

高志坚，蒋春蕾

（西昌学院，四川 西昌 615013）

【摘 要】随着信息量的不断增加，现在用计算机处理的文件越来越大，并且这些信息的表达都存在一定的冗余，因此我们需要使用压缩算法对信息进行压缩。本文介绍了数据压缩算法的发展历程和Z77压缩算法的编程实现。

【关键词】无损压缩；LZ77；冗余；滑动窗口；熵

【中图分类号】TP301.6 【文献标识码】A 【文章编号】1673-1891(2005)01-0088-04

1 通用无损数据压缩

人们在研究中发现，大多数信息的表达都存在着一定的冗余度(空间冗余和时间冗余等等)，通过采用一定的模型和编码方法，可以降低这种冗余度。贝尔实验室的 Claude Shannon 和 MIT 的 R.M. Fano 几乎同时提出了最早的对符号进行有效编码从而实现数据压缩的shannon-Fano 编码方法。D.A. Huffman 于1952年第一次发表了他的论文“最小冗余度代码的构造方法”(A Method for the Construction of Minimum Redundancy Codes)。从此，数据压缩开始在商业程序中实现并被应用在许多技术领域。UNIX 系统上一个不太为现代人熟知的压缩程序 COMPACT 就是 Huffman 0 阶自适应编码的具体实现。80年代初，Huffman 编码又在 CP/M 和 DOS 系统中实现，其代表程序叫 SQ。在数据压缩领域，Huffman 的这一论文事实上开创了数据压缩技术一个值得回忆的时代，60年代、70年代乃至80年代的早期，数据压缩领域几乎一直被 Huffman 编码及其分支所垄断。如果不是后面将要提到的那两个以色列人，也许我们今天还要在 Huffman 编码的0和1的组合中流连忘返。

让我们沿着 Huffman 的轨迹再向后跳跃几年，20世纪80年代，数学家们不满足于 Huffman 编码中的某些致命弱点，他们从新的角度入手，遵循 Huffman 编码的主导思想，设计出另一种更为精确，更能接近信息论中"熵"极限的编码方法——算术编码。凭借算术编码的精妙设计和卓越表现，人们终于可以向着数据压缩的极限前进了。可以证明，算术编码得到的压缩效果可以最大地减小信息的冗余度，用最少量的符号精确表达原始信息内容。当然，算术编码同时也给程序员和计算机带来了新的挑战：要实现和运行算术编码，需要更为艰苦的编程劳动和更加快速的计算机系统。也就是说，在同样的计算机系统上，算术编码虽然可以得到最好的压缩效果，但却要消耗也许几十倍的计算时间。这就是为什么算术编码不能在我们日常使用的压缩工具中实现的主要原因。那么，能不能既在压缩效果上超越Huffman，又不增加程序对系统资源和时间的需求呢？我们必须感谢下面将要介绍的两个以色列人。

直到1977年，数据压缩的研究工作主要集中于熵、字符和单词频率以及统计模型等方面，研究者们一直在绞尽脑汁为使用 Huffman 编码的程序找出更快、更好的改进方法。1977年以后，一切都改变了。

1977年，以色列人Jacob Ziv和Abraham Lempel发表了论文“顺序数据压缩的一个通用算法”(A Universal Alogrithem for Sequential Data Compression)。1978 年，他们发表了该论文的续篇“通过可变比率编码的独立序列的压缩”(Compression of Individual Sequences via Variable-Rate Coding)。

所有的一切都改变了，在这两篇论文中提出的两个压缩技术被称为 LZ77 和 LZ78 （不知为什么，作者名字的首字母被倒置了）。简单地说，这两种压缩方法的思路完全不同于从 Shannon 到 Huffman 到算术压缩的传统思路，人们将基于这一思路的编码方法称作“字典”式编码。字典式编码不

收稿日期：2004-12-03

作者简介：高志坚(1970-)，男，高级程序员。研究方向：网络信息系统。

但在压缩效果上大大超过了Huffman,而且,对于好的实现,其压缩和解压缩的速度也异常惊人。

1984年,Terry Welch 发表了名为"高性能数据压缩技术"(A Technique for High-Performance Data Compression)的论文,描述了他在Sperry Research Center(现在是 Unisys 的一部分)的研究成果。他实现了 LZ78 算法的一个变种——LZW。LZW 继承了 LZ77 和 LZ78 压缩效果好、速度快的优点,而且在算法描述上更容易被人们接受(有的研究者认为是由于Welch 的论文比 Ziv 和 Lempel 的更容易理解),实现也比较简单。不久,UNIX 上出现了使用 LZW 算法的 Compress 程序、该程序性能优良,并有高水平的文档,很快成为了 UNIX 世界的压缩程序标准。紧随其后的是 MS-DOS环境下的ARC 程序 (System Enhancement Associates, 1985),还有象 PKWare、PKARC 等仿制品。LZ78 和 LZW 一时间统治了 UNIX 和 DOS 两大平台。

80年代中期以后,人们对 LZ77 进行了改进,随之诞生了一批我们今天还在大量使用的压缩程序。Haruyasu Yoshizaki (Yoshi) 的 LHarc 和 Robert Jung 的 ARJ 是其中两个著名的例子。LZ77 得以和 LZ78、LZW 一起垄断当今的通用数据压缩领域。

目前,基于字典方式的压缩已经有了一个被广泛认可的标准,从古老的 PKZip 到现在的 WinZip,特别是随着 Internet 上文件传输的流行,ZIP 格式成为了事实上的标准,没有哪一种通用的文件压缩、归档系统敢于不支持 ZIP 格式。

2　多媒体信息的压缩

今天的程序员们和设计师们往往乐此不疲地为计算机更换更大的硬盘,增加更多的内存,其主要目的是为了存放和处理越来越多的声音、图像和视频数据。对声音、图像、视频等多媒体信息的压缩有两条思路,要么采用成熟的通用数据压缩技术进行压缩,要么根据媒体信息的特性设计新的压缩方法。事实上,人们在两条道路上都作了卓有成效的探索。

还记得 GIF 格式吗?GIF可以把原始图形文件以非常小数据量存储,可以在同一个文件中存储多幅图像从而实现动画效果。知道GIF中的图像使用什么方法压缩的吗?LZW!原来如此啊。GIF目前是使用通用压缩技术压缩图像信息的最成功的例子,当然,GIF文件中除了经过LZW压缩的像素信息以外,还保存有图像的各种属性信息以及图像所使用的调色板信息等。GIF精确地保留了原始图像的每一个像素信息,是无损图像压缩的代表。

根据媒体特性量身定制的压缩方法中,行程编码(RLE: Run-Length Encoding)是最为简单、最容易被想到的一种。大多数计算机中产生的图像(和现实世界的图像例如照片不同)都具有着大面积重复的颜色块,为什么非要用无数个完全相同的颜色值来表示某块图像呢?我们完全可以用一个颜色值加一个重复次数来表示这一块图像,冗余度由此减小了,这就是 RLE方法的基本思路。显然,它不适于用来压缩照片、声音等很少连续重复信息的数据。RLE方法最有代表性的实现有 PCX 和 Targa 图形格式。

如果分别考察的话,只有黑白两种颜色的二值图像以及只有256 级灰度变化的图像具有一些独特的地方,可以被压缩算法加以利用。我们知道,传真图像是一种典型的二值图像。国际电报电话咨询委员会(CCITT)为此建立了一系列的压缩标准,专门用于压缩传递二值图像 (可以是无损的或有损的)。对于灰度图像,除了著名的 JPEG 标准以外,后文将要介绍的一种叫 FELICS 的算法可以实现效果非常好的无损压缩。

70年代末80年代初,人们逐渐意识到,对到多数灰度或是彩色图像乃至声音文件,没有必要忠实地保留其所有信息,在允许一定的精度损失的情况下,可以实现更为有效的压缩方法。到80年代末,许多人已经在这一领域取得了不小的收获,设计出了一批在压缩效果上让人惊讶不已的声音和图像压缩算法。在此基础上,国际标准化组织(ISO)和 CCITT 联合组成了两个委员会。委员会的名字我们大概都已经非常熟悉了:静态图像联合专家小组(JPEG)和动态图像联合专家小组(MPEG)。JPEG 的压缩目标是静止图像(灰度的和彩色的),MPEG 的目标则是声音和视频。但他们的基本思路是完全一样的,即保留媒体信息中最有规律、最能体现信息主要特征的数据,而略去其他不重要的数据。他们都取得了令人赞叹的成就。

最后,必须简单地提到与图像压缩领域相关的电子出版印刷领域中的一种叫做 PostScript 的东西。PostScript是作为电子印刷业的标准页面描述语言被设计出来的,它起源于1976年的 Evans &

Sutherland 计算机公司，当时的名字是Design System。1978年，John Warnock 和 Martin Newel将其演变为JAM语言。1982年，John Warnock 和 Chuck Geschke 创建了著名的 Adobe System 公司，并第三次设计和实现了这个语言，并将其称为PostScript。

PostScript 的主要思路是存储和传输预先定义的命令来"画"出图像，而不是存储和传输图像的每一个像素，这特别适用于在激光打印机上的输出。采用类似"从(10, 10)到(100, 100)画一条红色直线"或是"在(50,50)以 40 为半径画一个蓝色的圆"之类的命令存储图像显然比直接存储每个像素节省了不少地方。所以，从压缩技术的角度来看，PostScript 也可以算是压缩方法的一种。根据类似的原理，Windows 中的 WMF 格式、HP 的 HPGL 语言、AutoCAD 中使用的 DXF 格式等等，都可以对某种特定的图像进行有效的压缩。

3 LZ77算法的实现过程

为了更好地说明LZ77算法的原理，首先介绍算法中用到的几个术语：

(1) 输入数据流(input stream:要被压缩的字符序列。

(2) 字符(character):输入数据流中的基本单元。

(3) 编码位置(coding position):输入数据流中当前要编码的字符位置，指前向缓冲存储器中的开始字符。

(4)前向缓冲存储器(Lookahead buffer):存放从编码位置到输入数据流结束的字符序列的存储器。

(5)窗口(window):指包含W个字符的窗口，字符是从编码位置开始向后数也就是最后处理的字符数。

(6)指针(pointer):指向窗口中的匹配串且含长度的指针。

LZ77算法是Lempel-Ziv在1977年发明的。LZ77算法有很多派生的算法(其中包括LZSS)。它们大都基本上是类似的。LZ77算法总是包括一个(sliding window滑动窗口，是一个容量可变的存储器)和一个预读缓存器 (read ahead buffer)。这个sliding windows基本上说是一个历史缓存器，用来存放前n个字节的输入流(input stream)。一个sliding window大小可以从0K到64K。LZSS使用的是一个4K的sliding window,这也是通常使用的大小。预读缓存器与sliding window是对应的。它保存这个流的前n个字节。预读缓存器的大小通常在0到258K之间。这个算法就是基于这个建立的。用下n个字节填充预读缓存器 (n是预读缓存器的大小)。查找这个sliding window,找到和预读缓存器中的数据最匹配的。如果这个匹配的长度大于最小匹配长度 (最小匹配长度取决于编码器，通常取决于sliding window的长度。比如一个4K的sliding window，最小匹配长度为2)，然后输出一个<length,distance>，长度(length)是这个匹配的长度，距离(distance)是在向后多少字节的地方这个匹配被找到的。为了弄清以上的问题，我们来看一个例子：

(我们要用一个10字节长的sliding window,和一个5字节的预读缓存器)

输入流(Input Stream):

A A A A A A A A A A B A B A A A A A

---------- =========

Sliding Window Read Ahead Buffer

假设这个sliding window中包含10个A (在上面这个例子中)，这就是最后被读入的10个字节。预读缓存器中包含B A B A A .编码第一步，在sliding window中找与预读缓存器中的数据大于2个的匹配。在sliding window中没有找到B A B A A,所以输出B到数据流中。然后，sliding window移动一个字节。

当前输出流:B

输入流:

A A A A A A A A A A B A B A A A A A

---------- =========

Sliding Window Read Ahead Buffer

现在，预读存储器中包含A B A A A。然后再与sliding window比较。可知，在sliding window中可以找到A B这个长度为2的匹配，它是在从现在向前2字节的位置。于是输出一个<length,distance>。length为2，backwards distance也是2,输出<2,2>。

当前输出流:B <2,2>

输入流:

A A A A A A A A A A A B A B A A A A A
---------- ========
Sliding Window　　Read Ahead
Buffer

预读缓存器中包含A A A A A。在sliding window中向前8个字节的地方有一个长度为5的匹配。输出就是<5,8>。

最后输出流:B <2,2> <5,8>

很简练是不是?现在让我们来将这个流解码。假设最后解码的10个字节是10个A。

输入流:

A A A A A A A A A A A B <2,2> <5,8>

预读缓存器和sliding window在解码时是用不上的。前十个A是字面上的，所以它们像这样被解码:

当前输出流:A A A A A A A A A A

下一个字节的B也是一个literal，所以像这样解码:

当前输出流:A A A A A A A A A A B

然后遇到了一个<length,distance>:<2,2>,意思是向后退两个字节复制2字节到输出流:

当前输出流:

A A A A A A A A A A B
<--
2 bytes back
A A A A A A A A A A B A B

|->
copied to output

接下来是<5,8>,意思是退后8字节,复制5字节到输出:

当前输出流:

A A A A A A A A A A B A B
<----------
8 bytes back
A A A A A A A A A A B A B A A A A A

|---------->
copied to output

现在有个问题。怎么确定哪个字节是literal,哪个字节表示<length,distance>?解决办法是用将数据编成一个组用一个8位的数据作为前缀来标识。当这个前缀字节转换成二进制时，用标记on和off来表示后8个字节是literal还是<length,distance>。例如,编码流(在上一个例子中)是:

B <2,2> <5,8>，这里有一个literal和两个distance。前缀字节应该是011(0表示literal,1表示<length,distance>),然后,在剩下的5个位上补0,这样凑成了一个字节:01100000 -> 96(十进制)。最后的流应该是:

96 B 2 2 5 8

这里所描述的LZ77是这个算法通常的形式。有一些编码/解码器实际上输出<distance,length>而不是<length,distance>，或者用不同的方法来表示前缀。

LZ77 Compresses Algorithms and Derives Algorithms to Probe into

GAO Zhi-jian, JIANG Chun-lei

(*Xichang college, Xichang 615013, Sichuan*)

Abstract:With the constant increase of the amount of information, the file dealt with the computer now is bigger and bigger, and expressions of the information all have certain redundancy, so we need to use the algorithm of compressing to compress information. This text introduce data compress development course of algorithm , introduce LZ77 compress to become realization Cheng algorithm in detail.

Key word:Compress harmlessly LZ77; Redundant; Slip_window; Entropy