托斯卡纳

文艺复兴的最初摇篮，意大利的静谧空间…

« 基于颜色特征的图像检索系统设计
入侵检测通信机制的设计 »

基于算术编码的数据压缩算法研究与实现

- 六月 25th, 2008

在现今的电子信息技术领域，由于需要处理的数字化的信息（尤其是多媒体信息）通常会特别庞大，如果不对其进行有效压缩就难以得到实际应用，数据压缩的目的即是通过有效减少数据文件的冗余信息而使数据文件可以以更快的速度传输或在更少的空间储存。因此数据压缩技术已成为当今数字通信、存储和多媒体娱乐的一项关键的共性技术。

本文由香农熵理论和统计编码的原理开始，逐步展开对基于算术编码的数据压缩的研究与应用的讨论：从算术编码的原理、产生条件、以及研究算术编码的目的意义等，到具体算术编码方案的分析比较以及其 C++语言的实现方案，有重点的对算术编码的特点进行了分析和阐述。而针对算术编码在处理二元符号时高压缩比、低复杂度的特点，本文着重探讨了算术编码方法处理二元数据流的过程的特点和效率优势，并将算术编码的不同实现方法进行了分析和比较，特别是对 N 阶自适应编码的特点和处理文字信息的优势进行了分析，然后将其和与之较为类似的 Huffman 编码进行了比较，通过比较得出了算术编码具有但 Huffman 编码不具有的在处理数据流方面的优势，即 Huffman 编码必须在得到全部数据文件之后才可以对文件进行编码处理，而算术编码方法可以在只得到数据流片段的情况下就开始对数据进行压缩，使得当处理数据流信息时在保证高压缩比的同时具有了很大的灵活性。

本文通过对算术算法特点和应用方向的研究，阐明其在数据压缩领域不可取代的地位及在处理流片段数据所具有的在压缩比和灵活性方面的优势，展示出算术编码的强大生命力和独特优势。

最后，应用文中研究得到的算术编码方法和实现模型，在 Windows 系统下，使用 Visual C++ 作为编程工具，实现了算术编码及其应用程序界面，，对于接近二

进制流的文件，本设计具体令人满意的压缩效果，对其他格式的文件也有较好的压缩效果，达到了论文的设计目标。

关键词：算术编码、无损压缩、自适应模式

目 录

摘 要

ABSTRACT

Nowadays, as the digital information (especially the multimedia information) becomes more voluminous in the telegraphy field, the information should be compressed availably. The purpose of data compression is reducing the redundancy of data files effectively for faster transfer and/or smaller space for storage. So the data compression technology becomes a common pivotal technology for digital communication,

storage and multimedia entertainment.
From Shannon entropy theory and the statistics coding theory, this paper sets forth the research and application of the data compression which based on Arithmetic Coding, including the arithmetic coding theory, the having conditions and the purpose of arithmetic coding and then the research of the specific implementation plan with C++ language of arithmetic coding. Against the point of arithmetic coding, this paper analysis and expounds its superiority about it . For the characteristic of the superiority of compression ratio and complexity, this paper probe into the process of deal with the binary data streams and the superiority of efficiency. And taking compared between the different implementation plan, especially the characteristic of Order-N adaptive coding and the superiority in dealing with the textual information. Then compares arithmetic coding with the Huffman coding method which is very resembles with it, getting the superiority in dealing with the data stream fragment: arithmetic coding doesn’t need the complete file before need to encode it, but Huffman coding method cannot do it like this. That means arithmetic do not only have the superiority in compression ratio but also in flexibility.
Through the research of the characteristic and the application direction of arithmetic coding, this paper illuminates the unplaced status of Arithmetic Coding, the outstanding compress ratio and agility in deal with data stream fragments and brings forth the life force and its unique superiority.
At last, by the arithmetic coding method and implement model researched in this paper, completed the coding method and its correspondingly application procedure interface using Visual C++ programming tools in the Windows operating system. The test result shows that to the files close to the binary files, this design procures a satisfactory outcome. And to the files in other format, this design also get a preferably results. So, the design achieves the goal required in this paper.
KEY WORDS arithmetic coding, lossless compression, adaptive model

第一章 绪论

1.1 数据压缩

数据压缩，用一句话说，就是用最少的数码来表示信号，即将字符串的一种表示方式转换为另一种表示方式，新的表示方式包含相同的信息量，但是长度比原来的方式尽可能的短。其作用是：能较快地传输各种信号，如传真、Modem 通信等；在现有的通信干线并行开通更多的多媒体业务，如各种增值业务；紧缩数据存储容量，如 CD－ROM、VCD 和 DVD 等；降低发信机功率，这对于多媒体移动通信系统尤为重要。也就是说，通信时间、传输带宽、存储空间甚至发射能量，都可能成为数据压缩的对象。

数据之所以能够被压缩是基于以下几点的考量：

首先，数据中间常存在一些多余成分，既冗余度。如在一份计算机文件中，某些

符号会重复出现、某些符号比其他符号出现得更频繁、某些字符总是在各数据块中可预见的位置上出现等，这些冗余部分便可在数据编码中除去或减少。冗余度压缩是一个可逆过程，因此叫做无失真压缩，或称保持型编码。
其次，数据中间尤其是相邻的数据之间，常存在着相关性。如图片中常常有色彩均匀的背影，电视信号的相邻两帧之间可能只有少量的变化影物是不同的，声音信号有时具有一定的规律性和周期性等等。因此，有可能利用某些变换来尽可能地去掉这些相关性。但这种变换有时会带来不可恢复的损失和误差，因此叫做不可逆压缩，或称有失真编码、摘压缩等。
此外，人们在欣赏音像节目时，由于耳、目对信号的时间变化和幅度变化的感受能力都有一定的极限，如人眼对影视节目有视觉暂留效应，人眼或人耳对低于某一极限的幅度变化已无法感知等，故可将信号中这部分感觉不出的分量压缩掉或”掩蔽掉”。这种压缩方法同样是一种不可逆压缩。
数据压缩跟编码技术联系紧密，压缩的实质就是根据数据的内在联系将数据从一种编码映射为另一种编码。压缩前的数据要被划分为一个一个的基本单元。基本单元既可以是单个字符，也可以是多个字符组成的字符串。称这些基本单元为源消息，所有的源消息构成源消息集。源消息集映射的结果为码字集。可见，压缩前的数据是源消息序列，压缩后的数据是码字序列。
若定义块为固定长度的字符或字符串，可变长为长度可变的字符或字符串，则编码可分为块到块编码、块到可变长编码、可变长到块编码、可变长到可变长编码等。应用最广泛的 ASCII 编码就是块到块编码。对于数据压缩技术而言，最基本的要求就是要尽量降低数字化的在码事，同时仍保持一定的信号质量。不难想象，数据压缩的方法应该是很多的，但本质上不外乎上述完全可逆的冗余度压缩和实际上不可逆的嫡压缩两类。冗余度压缩常用于磁盘文件、数据通信和气象卫星云图等不允许在压缩过程中有丝毫损失的场合中，但它的压缩比通常只有几倍，远远不能满足数字视听应用的要求。在实际的数字视听设备中，差不多都采用压缩比更高但实际有损的媳压缩技术。只要作为最终用户的人觉察不出或能够容忍这些失真，就允许对数字音像信号进一步压缩以换取更高的编码效率。摘压缩主要有特征抽取和量化两种方法，指纹的模式识别是前者的典型例子，后者则是一种更通用的摘压缩技术。
1.2 数据压缩的现状与发展趋势
设计具体的压缩算法时，设计者首先要做的是寻找一种能尽量精确地统计或估计信息中符号出现概率的方法，然后还要设计一套用最短的代码描述每个符号的编码规则。统计学知识对于前一项工作相当有效，迄今为止，人们已经陆续实现了静态模型、半静态模型、自适应模型、Markov 模型、部分匹配预测模型等概率统计模型。
第一个实用的编码方法是由 D. A. Huffman 在 1952 年的论文”最小冗余度代码的构造方法”[1]中提出的。直到今天，许多”数据结构”教材在讨论二叉树时仍要提及这种被后人称为 Huffman 编码的方法。 Huffman 编码看似简单，但却影响深远，其编码效率高，运算速度快，实现方式灵活，从 20 世纪 60 年代至今，在数据压缩领域得到了广泛的应用。
1968 年前后， P. Elias[2] 发展了 Shannon 和 Fano 的编码方法，构造出从数学角度看来更为完美的 Shannon-Fano-Elias 编码。沿着这一编码方法的思路， 1976 年， J. Rissanen[3] 提出了一种可以成功地逼近信息熵极限的编码方法－算术编码。 1982 年， Rissanen 和 G. G. Langdon[4] 一起改进了算术

编码。之后，人们又将算术编码与 J. G. Cleary 和 I. H. Witten[5] 于 1984 年提出的部分匹配预测模型（ PPM ）相结合，开发出了压缩效果近乎完美的算法。今天，那些名为 PPMC 、 PPMD 或 PPMZ 并号称压缩效果天下第一的通用压缩算法，实际上全都是这一思路的具体实现。

犹太人 J. Ziv 和 A. Lempel 脱离 Huffman 及算术编码的设计思路，创造出了一系列比 Huffman 编码更有效，比算术编码更快捷的压缩算法。这些算法统称为 LZ 系列算法，如： LZ77 算法， LZ78 的压缩算法，以及 LZW 算法。该系列算法的思路并不新鲜，其中既没有高深的理论背景，也没有复杂的数学公式，它们只是用一种极为巧妙的方式将字典技术应用于通用数据压缩领域。这种基于字典模型的思路在表面上虽然和 Shannon 、Huffman 等人开创的统计学方法大相径庭，但在效果上一样可以逼近信息熵的极限。而且 LZ 系列算法在本质上符合信息熵的基本规律。LZ 系列算法的优越性使得使用该算法的压缩软件数量呈爆炸式增长。今天，我们熟悉的 PKZIP 、 WinZIP 、 WinRAR 、Gzip 等压缩工具以及 ZIP、GIF、PNG 等文件格式都是 LZ 系列算法的受益者。

在图像压缩领域，著名的 JPEG 标准是有损压缩算法中的经典。 JPEG 标准由静态图像联合专家组（ Joint Photographic Experts Group ， JPEG ）于 1986 年开始制定，1994 年后成为国际标准。 JPEG 以离散余弦变换（ DCT ）为核心算法，通过调整质量系数控制图像的精度和大小。

CCITT 于 1988 年制定了电视电话和会议电视的 H.261 建议草案。H.261 的基本思路是使用类似 JPEG 标准的算法压缩视频流中的每一帧图像，同时采用运动补偿的帧间预测来消除视频流在时间维度上的冗余信息。在此基础上， 1993 年， ISO 通过了动态图像专家组（ Moving Picture Experts Group ， MPEG ）提出的 MPEG-1 标准。 MPEG-1 可以对普通质量的视频数据进行有效编码。为了支持更清晰的视频图像，特别是支持数字电视等高端应用， ISO 于 1994 年提出了新的 MPEG-2 标准（相当于 CCITT 的 H.262 标准）。 MPEG-2 对图像质量作了分级处理，可以适应普通电视节目、会议电视、高清晰数字电视等不同质量的视频应用。在我们的生活中，可以提供高清晰画面的 DVD 影碟所采用的正是 MPEG-2 标准。

Internet 的发展对视频压缩提出了更高的要求。ISO 于 1999 年通过了 MPEG-4 标准（相当于 CCITT 的 H.263 和 H.263+ 标准）。 MPEG-4 标准拥有更高的压缩比率，支持并发数据流的编码、基于内容的交互操作、增强的时间域随机存取、容错、基于内容的尺度可变性等先进特性。音频数据的压缩技术最早是由无线电广播、语音通信等领域里的技术人员发展起来的。这其中又以语音编码和压缩技术的研究最为活跃。自从 1939 年 H. Dudley 发明声码器以来，人们陆续发明了脉冲编码调制（ PCM ）、线性预测（ LPC ）、矢量量化（ VQ ）、自适应变换编码（ ATC ）、子带编码（ SBC ）等语音分析与处理技术。这些语音技术在采集语音特征，获取数字信号的同时，通常也可以起到降低信息冗余度的作用。为获得更高的编码效率，大多数语音编码技术都允许一定程度的精度损失。而且，为了更好地用二进制数据存储或传送语音信号，这些语音编码技术在将语音信号转换为数字信息之后又总会用 Huffman 编码、算术编码等通用压缩算法进一步减少数据流中的冗余信息。

很显然，在多媒体信息日益成为主流信息形态的数字化时代里，数据压缩技术特别是专用于图像、音频、视频的数据压缩技术还有相当大的发展空间 - 毕

竞，人们对信息数量和信息质量的追求是永无止境的。
1994 年， M. Burrows 和 D. J. Wheeler 共同提出了一种全新的通用数据压缩算法。这种算法的核心思想是对字符串轮转后得到的字符矩阵进行排序和变换，类似的变换算法被称为 Burrows-Wheeler 变换，简称 BWT 。与 Ziv 和 Lempel 另辟蹊径的做法如出一辙， Burrows 和 Wheeler 设计的 BWT 算法与以往所有通用压缩算法的设计思路都迥然不同。如今， BWT 算法在开放源码的压缩工具 bzip 中获得了巨大的成功， bzip 对于文本文件的压缩效果要远好于使用 LZ 系列算法的工具软件。这至少可以表明，即便在日趋成熟的通用数据压缩领域，只要能在思路和技术上不断创新，我们仍然可以找到新的突破口。
分形压缩技术是图像压缩领域近几年来的一个热点。这一技术起源于 B. Mandelbrot 于 1977 年创建的分形几何学。 M. Barnsley 在 20 世纪 80 年代后期为分形压缩奠定了理论基础。从 20 世纪 90 年代开始， A. Jacquin 等人陆续提出了许多实验性的分形压缩算法。今天，很多人相信，分形压缩是图像压缩领域里最有潜力的一种技术体系，但也有很多人对此不屑一顾。无论其前景如何，分形压缩技术的研究与发展都提示我们，在经过了几十年的高速发展之后，也许，我们需要一种新的理论，或是几种更有效的数学模型，以支撑和推动数据压缩技术继续向前跃进。
人工智能是另一个可能对数据压缩的未来产生重大影响的关键词。既然 Shannon 认为，信息能否被压缩以及能在多大程度上被压缩与信息的不确定性有直接关系，假设人工智能技术在某一天成熟起来，假设计算机可以像人一样根据已知的少量上下文猜测后续的信息，那么，将信息压缩到原大小的万分之一乃至十万分之一，恐怕就不再是天方夜谭了。
人们总喜欢畅想未来，但未来终究是未来，对未来的预测其主要基准还是我们现有的应用技术和思维模式，未来技术的发展存在着诸多的不可预测性，但终究离不开现有的技术基础，所以这就是本文所作对压缩算法进行研究和应用探讨的意义所在。
1.3 课题研究的意义
电脑里的数据压缩的主要功用有两个：第一，可以节省存储空间；第二，可以在通信过程中减少对带宽的占用。虽然随着存储技术的快速发展，电脑的主存储器、辅助存储器的容量以及数据通信的带宽都有了很大提高，但如何在相同的空间存储更多的信息以及在已有的带宽情况下，更快捷的传输更多的信息，仍主要取决于数据压缩技术的发展。 如果没有数据压缩技术，那我们每次对数据信息进行提取和操作，与现在相比，将是个漫长复杂的过程，更无法在短时间内对海量信息进行检索，而视频和音频信息也将是人类信息交换的巨大负担。
为方便信息的存储、检索和使用，在进行信息处理时赋予信息元素以代码的过程。编码的目的在于提高信息处理的效率。信息编码必须标准、系统化。设计合理的编码系统是关系信息管理系统生命力的重要因素。信息编码的基本原则是在逻辑上要满足使用者的要求，又要适合于处理的需要；结构易于理解和掌握；要有广泛的适用性，易于扩充。一般应有的代码有两类，一类是有意义的代码，即赋予代码一定的实际意义，便于分类处理；一类是无意义的代码，仅仅是赋予信息元素唯一的代号，便于对信息的操作。在通信理论中，编码是对原始信息符号按一定的数学规则所进行的变换。编码的目的是要使信息能够在保证一定质量的条件下尽可能迅速地传输至信宿。
第二章 算术编码原理及特点

2.1 统计编码
数据压缩可以分为可逆的无失真编码和不可逆的有失真编码两大类基本方法[6]。因为大多数计算机文件都不允许在压缩过程中丢失信息，所以对各种信源都通用的可逆压缩方法就显得更为重要，这类方法主要利用消息或消息序列出现的概率的分布特性，注重寻找概率与码字长度间的最优匹配，叫做统计编码或概率匹配编码，统称为熵编码。
计算机文件表现为字符集合（如文本）或二进制符号（ 一般为 0 和 1）集合。两者的最终形式都是”0″、”1″代码，只是前者一般用 ASCII 码编码表示。文本文件包括电报报文、程序指令等，一般由 10 进制数字 0~9、英文字母及$、*、&等特殊符号组成，对其压缩必须”透明”，即恢复后的文件不许有任何失真。一个符号错误就可能产生灾难性的后果，例如数据库中包含有金融交易，或者控制系统的可执行程序。
对于文本文件，有物理压缩和逻辑压缩两种方法，逻辑压缩实际上只是一种由数据自身特点及设计者技巧来决定的”压缩表示法”，并不具有普遍性，而物理压缩方法则是通过减少计算机文件内部冗余度的方法来实现对源文件的压缩，而信源中的冗余度的表现形式包括：字符分布(character distribution)、字符重复(character repetition)、高使用率模式(high-usage patterns)以及位置冗余(positional redundancy)
香农定理[7]描述了有限带宽;有随机热噪声信道的最大传输速率与信道带宽;信号噪声功率比之间的关系.
在有随机热噪声的信道上传输数据信号时，数据传输率 Rmax 与信道带宽 B，信噪比 S/N 关系为： Rmax=B×log2（1+S/N） 在信号处理和信息理论的相关领域中，通过研究信号在经过一段距离后如何衰减以及一个给定信号能加载多少数据后得到了一个著名的公式，叫做香农（Shannon）定理。它以比特每秒（bps）的形式给出一个链路速度的上限，表示为链路信噪比的一个函数，链路信噪比用分贝（dB）衡量。因此我们可以用香农定理来检测电话线的数据速率。
香农定理由如下的公式给出: C=Blog2(1＋S/N) 其中 C 是可得到的链路速度，B 是链路的带宽，S 是平均信号功率,N 是平均噪声功率，信噪比（S/N）通常用分贝（dB）表示，分贝数=10×log10（S/N）。

讨论熵编码就必需要了解熵（entropy)的概念：(1)熵是信息量的度量方法，它表示某一事件出现的消息越多，事件发生的可能性就越小，数字上就是概率越小。(2)某个事件的信息量用 =-logi 表示，其中 为第 i 个事件的概率，0<1 。香农理论的重要特征即是熵的概念，他证明熵与信息内容的不确定程度有等价关系。熵曾经是波尔兹曼在热力学第二定律引入的概念，我们可以把它理解为分子运动的混乱度。信息熵也有类似意义，例如在中文信息处理时，汉字的静态平均信息熵比较大，中文是 9.65 比特，英文是 4.03 比特。这表明中文的复杂程度高于英文，反映了中文词义丰富、行文简练，但处理难度也大。信息熵大，意味着不确定性也大。
按照香农的理论[7]，信源 S 的熵的定义为 H（S）= = log(1/其中 是符号 在 S 中出现的概率；lo (1/表示包含在 中的信息量，也就是编码 所需要的位数。例如，一幅用 256 灰度级表示的图像，如果每一个象素点灰度的概率均为 =1/256，编码每一个象素点就需要 8 位。熵作为理论上的平均信息量，即编码一个信源符号所需的二进制位数，在实际的压缩编码中的码率很难达到熵值，不过

熵可以作为衡量一种压缩算法的压缩比好坏的标准，码率越接近熵值，压缩比越高。由于在许多场合，开始不知道要编码数据的统计特性，也不一定允许你事先知道它们的编码特性，因此算术编码在不考虑信源统计特性的情况下，只监视一小段时间内码出现的概率，不管统计是平稳的或非平稳的，编码的码率总能趋近于信源的熵值。实现算术编码首先需要知道信源发出每个符号的概率大小，然后再扫描符号序列，依次分割相应的区间，最终得到符号序列所对应的码字。

2.2 算术编码原理

算术编码是一种到目前为止编码效率最高的统计熵编码方法,它比著名的 Huffman 编码效率提高 10 %左右,但由于其编码复杂性和实现技术的限制以及一些专利权的限制,所以并不像 Huffman 编码那样应用广泛。国外对算术编码的研究较多,取得了许多重要的应用,但大多都有专利保护,如 JPEG、JBIG 和 H. 261 中均采用了算术编码;国内的研究相对较少,应用不是很广泛,许多人还不了解。随着算术编码实现技术的改进,必将以其独特的优良性能成为无失真压缩方法的主流。

算术编码的处理过程会产生一个位数逐步增加的实数, 该实数所对应的概率区间对应着编码信息。这个实数的值稍有变化, 其对应的概率区间就会改变, 导致解不出原来的信息, 后面译出的码字全部错误. 这就是算术编码的重要特点: 算术编码是非分组码[6] , 编码器是有记忆的, 编码输出流被看作一个不可分割的整体. 这个特点不仅带来严重的误码扩散问题, 而且给算术编码的灵活应用带来了很大困难. 其他的压缩算法, 像 Huffman 算法、LZW 算法等, 是分组码, 各输出码字之间是独立的, 误码不易扩散. 对于算术编码,从对其基本原理的讨论中看到, 它的编码输出流中是不能随意嵌入冗余编码等附加信息的.算术编码的思想可以追溯到 1948 年 Shannon 的论文[7]中,20 世纪 60 年代初期,Elias 给出了一个递推公式[2],使得算术编码向实用化方向前进了一大步。但此时的算术编码需要无限精度的浮点运算,这对于当时的技术和实用化进程是一个无法克服的障碍。70 年代, Rissanen[3] 和 Pasco 分别提出了用有限精度逼近的算法,从此算术编码开始进入实用化阶段。进入 80 年代,算术编码的研究达到高潮。Rubin、Rissanen 和 Langdon[4] 等人在此期间对算术编码作了大量的理论研究,并给出了一个二进制的算术编码器,Witten 等人给出了一个通用的算术编码器。90 年代以来,算术编码的理论已基本成熟,主要研究工作放在算法的实现技术的改进和应用上。算术编码可以以分数比特逼近信源的熵,统计模型可以与算法很好地分离,易于实现自适应模式,因此编码算法流畅完美,效率很高。

2.2.1 算术编码理论

算术编码(arithmetic coding)是一种无损数据压缩方法，也是一种熵编码的方法[6]。和其它熵编码方法不同的地方在于算术编码跳出了分组编码的范畴，从全序列出发，采用递推形式的连续编码，它不是将单个的信源符号映射成一个码字，而是将整个输入符号序列映射为实数轴上[0,1)区间内的一个小区间，其长度等于该序列的概率；再在该小区间选择一个代表性的二进制小数，作为实际的编码输出，从而达到了高效编码的目的，不论是否二元信源，也不论数据的概率分布如何，其平均码长均能逼近信源的熵。

早在 1948 年，香农[7]就提出将信源符号依其概率降序排序，用符号序列累积概率的二进制表示作为对信源的编码，并从理论上谁了它的优越性；1960 年后，

P.Elias 发现无需排序，只要编、解码端使用相同的符号顺序即可。但当时人们仍然认为算术编码需要无限精度的浮点运算，或随着符号的输入，所需的计算精度和时间也相应增加。1976 年，R.Pasco 和 J.Rissanen[3]分别用定长的寄存器实现了有限精度的算术编码,但仍无法实用,因为后者的方法是"后入先出"(LIFO)的，而前者的方法跃然是"先入先出"(FIFO)的，但却没有解决有限精度计算所固有的进位问题，1979 年 Rissanen 和 G.G.Langdon[4]一起将算术编码系统化，并于 1981 年实现了二进制编码。1987 年 Witten[5]等人了一个实用的算术编码程序，即 CACM87(后用于 ITU-T 的 H.263 视频压缩)；同期 IBM 公司发表了著名的 Q一编码器(后用于 JPEG、JPEG2000 和 JBIG 图像压缩标准),从此算术编码迅速得到了广泛的注意。

算术编码的基本原理是[6]：根据信源可能发现的不同符号序列的概率，把 [0,1)区间划分为互不重叠的子区间，子区间的宽度恰好是各符号序列的概率。这样信源发出的不同符号序列将与各子区间一一对应,因此每个子区间内的任意一个实数都可以用来表示对应的符号序列，这个数就是该符号序列所对应的码字。显然，一串符号序列发生的概率越大，对应的子区间就越宽，要表达它所用的比特数就减少，因而相应的码字就越短。使用算术编码方法进行多元符号编码时，算术编码每次递推都要做乘法，而且必须在一个信源符号周期内完成，有时就难以实时，为此采用了查表等许多近似计算来代替乘法，但若编码对象本身就是二元序列，且其符号概率较小者为 p(L)=2-Q 形式，其中 Q 是正整数，称作不对称数(skew number),则乘以 2-Q 可代之以右移 Q 位，而乘以符号较大者 p(H)=1-2-Q 可代之以移位和相关，这样就完全避免了乘法。因此算术编码很适合二元序列，而 p(L)常用 2-Q 来近似。，随着输入序列，长度的增加，编成 C(s)的长度也随之不断增加，而实际只能用有限长的寄存器 C，这就要求将 C 中已编码的高位码字及时输出。但又不能输出过早，以免后续运算还需调整已输出的。不难想象，当 C 中未输出部分各高位均为"1"时，则低位运算略有增量，就可能进位到已输出部分，特别是当这种连"1"很长时，这就是有限精度算术编码所固有的进位问题。Rissanen 和 Langdon 利用插入 1 个额外的"0"(即所谓"填充位")来隔断进位的扩展,对编码效率会略有影响。类似地,对于区间宽度 A(s)，也只能基于有限位数的寄存器 A 来实现。对于算术的编解码来说，不对称数 Q(s)是一个重要参数，当时假定它是根据信源概率模型已事先确定的一个量，要从一个二进制序列 s 来确定 Q 值，有待于根据该序列的统计特性，选择合适的概率模型，譬如根据符号串 s 后面出现字符 1 条件概率 $p(L|s)\sim=2-Q$ 来确定 Q 值。而当实际上"0"和"1"都有可能成为符号位数值时,应该随着它们在被编码符号串中出现的概率而自适应地改变。算术编码器在每一步都需要知道用于下一个待编码符号的 Q(s)值，以及指出哪个符号是 L 符号，通常串中字符发生的概率与序列的概率模型有关，因此算术编码应用的另一个问题，就是快速自适应地估计条件概率 p(L|s)，从信源的统计特性出发，建立数据的概率模型。而算术编码的最大优点之一，就是具有自适应功能，高二进制信源的字母表主(0,1)，算术编码在初始化预置一个大概率 P 和一个小概率 Q，随着输入版本号概率的变化，自动修改 Q 或 P 的值。一般假定初始值 Q=0.5，当后继输入连续为 H 符号时，Q 值渐渐减小，若连续出现 L 符号时，Q 值增加。增加到超过 0.5 时，Q 和 P 对应的符号相互交换。因此，使用算术编码不必预告定义信源的概率模型，尤其适用于不可能进行概率统计的场合。

Huffman 编码的一个不足是译码复杂度高，由于事先不知道码长，Huffman 码表

实质上是一棵二进制树，解析每一码字的基本方法就是从树根开始，集资根据面临的每一位是 0 还是 1 来决定沿哪一半子树继续译码，直到端节点，这样在运算时就要对码字的每位做出逻辑判决。基要求译码器与一个传输速率为 200Mb/s 的磁盘驱动器。
要使其相适应而不导致系统瓶颈，则判决逻辑的时钟至少不能低于该速率。这并非不能实现，但却不那么简单。通常对于用变长码压缩的大容量数据，解码系统的性能价格比不会最高。而在实现上，算术编码要比霍夫曼编码更复杂，特别是硬件实现时。
算术编码也是变长码，编码过程中的移位和输出都不均匀，也需要缓存，在误差扩散方面，也比分组码更严重；在分组码中，由于误码而破坏分组，过一会儿常能自动恢复们是不在算术码中却往往会一直延续下去，因为它是从全序列出发来编码的。因而算术码流的传输也要求高质量的信道，或采用检错反馈重发的方式。
各种媒体信息（特别是图像和动态视频）数据量非常之大。例如：一幅 640x480 分辨率的 24 位真彩色图像的数据量约力 900kb；一个 100Mb 的硬盘只能存储约 100 幅静止图像画面。显然，这样大的数据量不仅超出了计算机的存储和处理能力，更是当前通信信道的传输速率所不及的。因此，为了存储、处理和传输这些数据，必须进行压缩。相比之下，语音的数据量较小，且基本压缩方法已经成熟，目前的数据压缩研究主要集中于图像和视频信号的压缩方面。图像压缩技术、视频技术与网络技术相结合的应用前景十分可观，如远程图像传输系统、动态视频传输可视电话、电视会议系统等已经开始商品化，MPEG 标准与视频技术相结合的产物一家用数字视盘机和 VCD 系统等都已进入市场。可以预计，这些
技术和产品的发展将对本世纪末到二十一世纪的社会进步产生重大影响。而算术编码作为一种高效的数据编码方法在文本，图像，音频等压缩中有广泛的应用，所以，研究算术编码以更好的利用它是非常必要的。

2.2.2 算术压缩模式
算术编码对整条信息（无论信息有多么长），其输出仅仅是一个数，而且是一个介于 0 和 1 之间的二进制小数。例如算术编码对某条信息的输出为 1010001111，那么它表示小数 0.1010001111，也即十进制数 0.64。
下面借助一个简单的例子来阐释算术编码的基本应用原理。为了表示上的清晰，我们暂时使用十进制表示算法中出现的小数，这丝毫不会影响算法的可行性。
考虑某条信息中可能出现的字符仅有 a b c 三种，我们要压缩保存的信息为 bccb。
在没有开始压缩进程之前，假设我们对 a b c 三者在信息中的出现概率一无所知（我们采用的是自适应模型），没办法，我们暂时认为三者的出现概率相等，也就是都为 1/3，我们将 0 - 1 区间按照概率的比例分配给三个字符，即 a 从 0.0000 到 0.3333，b 从 0.3333 到 0.6667，c 从 0.6667 到 1.0000。用图形表示就是：

```
+-- 1.0000
|
Pc = 1/3 |
|
+-- 0.6667
|
```

Pb = 1/3 |
|
+-- 0.3333
|
Pa = 1/3 |
|
+-- 0.0000

现在我们得到第一个字符 b，来看 b 对应的区间 0.3333 - 0.6667。这时由于多了字符 b，三个字符的概率分布变成：Pa = 1/4，Pb = 2/4，Pc = 1/4。让我们按照新的概率分布比例划分 0.3333 - 0.6667 这一区间，划分的结果可以用图形表示为：

+-- 0.6667
Pc = 1/4 |
+-- 0.5834
|
Pb = 2/4 |

|
+-- 0.4167
Pa = 1/4 |
+-- 0.3333

接着我们拿到字符 c，我们现在要关注上一步中得到的 c 的区间 0.5834 - 0.6667。新添了 c 以后，三个字符的概率分布变成 Pa = 1/5，Pb = 2/5，Pc = 2/5。我们用这个概率分布划分区间 0.5834 - 0.6667：

+-- 0.6667
|
Pc = 2/5 |
|
+-- 0.6334
|
Pb = 2/5 |
|
+-- 0.6001

|
Pa = 1/5 |

|
+-- 0.5834

现在输入下一个字符 c，三个字符的概率分布为：Pa = 1/6，Pb = 2/6，Pc = 3/6。我们来划分 c 的区间 0.6334 - 0.6667

+-- 0.6667
|
Pc = 3/6 |
|

+-- 0.6501
|
Pb = 2/6 |
|
+-- 0.6390
|
Pa = 1/6 |
|
+-- 0.6334

输入最后一个字符 b，因为是最后一个字符，不用再做进一步的划分了，上一步中得到的 b 的区间为 0.6390 - 0.6501，好，让我们在这个区间内随便选择一个容易变成二进制的数，例如 0.64，将它变成二进制 0.1010001111，去掉前面没有太多意义的 0 和小数点，我们可以输出 1010001111，这就是信息被压缩后的结果，我们完成了一次最简单的算术压缩过程。

解压缩是压缩的逆过程解压缩之前我们仍然假定三个字符的概率相等，并得出上面的第一幅分布图。解压缩时我们面对的是二进制流 1010001111，我们先在前面加上 0 和小数点把它变成小数 0.1010001111，也就是十进制 0.64。这时我们发现 0.64 在分布图中落入字符 b 的区间内，我们立即输出字符 b，并得出三个字符新的概率分布。类似压缩时采用的方法，我们按照新的概率分布划分字符 b 的区间。在新的划分中，我们发现 0.64 落入了字符 c 的区间，我们可以输出字符 c。同理，我们可以继续输出所有的字符，完成全部解压缩过程（为了叙述方便，我们暂时回避了如何判断解压缩结束的问题，实际应用中，这个问题并不难解决）。

对以上论述进行归纳，我们可以得到算术编码的特点是：

1）不必预先定义概率模型，自适应模式具有独特的优点；

2）信源符号概率接近时，此时算术编码效率高于其他编码方法。

3）算术编码绕过了用一个特定的代码替代一个输入符号的想法，用一个浮点输出数值代替一个符号流的输入。

算术编码虽然具有其独特的优点，但我们仍需要注意下面几个问题：

1）由于实际的计算机的精度不可能无限长，运算中出现溢出是一个明显的问题，但多数机器都有 16 位、32 位或者 64 位的精度，因此这个问题可使用比例缩放方法解决。

2）算术编码器对整个消息只产生一个码字，这个码字是在间隔[0, 1)中的一个实数，因此译码器在接受到表示这个实数的所有位之前不能进行译码。

3）算术编码也是一种对错误很敏感的编码方法，如果有一位发生错误就会导致整个消息译错。

算术编码是一种高效的熵编码方案，其每个符号所对应的码长被认为是分数[8]。由于对每一个符号的编码都与以前编码的结果有关，所以它考虑的是信源符号序

列整体的概率特性，而不是单个符号的概率特性，因而它能够更大程度地逼近信源的极限熵，降低码率。

第三章 典型算术编码方案分析

3.1 WNC 算法算术编码

WNC 算法建立模型的过程是这样的：

（1） 由于英文字符一共有 256 个，建立两个 257 个元素的数组 cum_freq[] 和 freq[]，其中第 257 个元素是结束标志。cum_freq[] 是将输入符号按照出现概

率的大小进行顺序排列，freq[]用于积累输入符号的出现频率。初始化时由于每个符号的出现概率是一样的，freq[]均为 1，cum_freq[]按照从 256 到 0 的顺序排列。

（2） 更新模型的时候首先要判断是否积累的频率超出了预定的范围，如果超出了的话，就需要将每个符号的出现频率减少一半；如果没有超出就进行下一步。接下来是重新排序，保证当前输入符号按照出现频率有着正确的顺序，最后就可以增加输入符号的出现概率了，此时不但需要增加对应的 freq[] ，而且需要增加对应的 cum_freq[]之前的所有值。因为 cum_freq[] 是最终用来计算输入符号的概率的。
二进制索引树的数据结构，下图所示。

图 3.2 二进制索引树结构
根据二进制索引树可以建立这样的一个前后关系，称节点 a 是节点 b 的前节点，它们满足：a=forw(b),节点 c 是节点 d 的后节点,它们满足 c=back (d) ，其中 forw (),back()如下
forw (i)=i+i&(-i)
back (i)=i&(i-1)
下面的算法采用 BIT 这种数据结构来组织数据，更新模型表。过程是这样的：
（1）按照 WNC 算法中建立模型的方法一样建立一个关于输入符号的模型，在这里一点不同的是只需要 cum_freq[] 来积累输入符号的出现频率。初始化时由于每个符号的出现概率是一样的， cum_freq[] 按照从最大值到 0 的顺序排列。
（2）更新模型中的元素时，由于已经建立的前后关系，只需要更新该元素的积累频率及其之后的元素的积累频率。
（3）判断是否积累的频率超出了预定的范围，如果超出了的话，就需要根据前后关系，将每个符号的积累频率减少一半。需要注意的是由于更新是按照前后关系进行的，所以减少的过程也需要按照这样的顺序进行。
整个编码器的框图如下：

图 3.3 WNC 算法编解码流程图

其中 install,update_model 的过程如上所述，encode_symbol 部分的具体算法流程如下：
（1）计算此时区间的范围和上界、下界。
范围 range=high-low+1
新的上界：high=low+range×cum_freq{back(symbol)}/cum_feq[0]-1
新的下界：low=low+range×cum_freq[symbol]/cum_freq[0]
（2）根据（1）计算的上界、下界输出比特，以下的过程在一个循环中反复进行，直到跳出。
i) 如果上界小于初始区间的一半，输出 0；
ii) 如果下界大于或等于初始区间的一半，输出 1，同时将区间向下移一半，也就是 low=low-HALF ; high=high-HALF
iii) 如果下界大于或等于初始区间的 1/4 并且上界小于初始区间的 3/4，输出一个与前一个输出位相反的跟随位，并且将区间向下移 1/4，low=low-QUARTER,

high=high-QUARTER;
iv) 如果不在上面的情况中，停止输出，跳出循环，否则的话，扩大当前区间的上下界（因为已经输出了表示位），low=2×low, high=2×high+1。
虽然 WNC 算法可以很好地解决算术编码的概率预定义问题，但是它又不可避免的带来了新的问题，就是关于模型更新以及查找该符号时所带来的庞大计算量的问题。

3.2 基于上下文的二进制算术编码

基于上下文的自适应二进制算术编码（CABAC）是一种体现了新思想、具备新特点的新型二进制算术编码方法。它的主要面对视频流的相关统计特性，其特点在于采用了高效的算术编码思想，充分考虑视频流的数据特点，大大提高了编码效率，针对视频数据流处理的 H.264/AVC 即采用了这种新的二进制算术编码。
CABAC 的基本编码步骤可分为三步：1. 二进制化；2. 上下文建模；3. 二进制算术编码。第一步主要是将非二进制的各语法元素值转换成二进制的比特序列，如果语法元素本身是二进制的，则该步骤可省略；第二步主要是为已二进制化的语法元素的比特序列的每一位提供概率模型，进行概率预测；第三步则进行二进制算术编码，在 CABAC 中，有两种编码模式，一种叫做 regular coding mode，另一种叫做 by pass coding mode，其中 regular coding mode 采用了上下文建模， 而 by pass coding mode 为了加快编码速度不采用上下文建模，整个编码过程如下图所示。

图 3.4 CABAC 编码流程图

CABAC 基本步骤简述：
首先是二进制化：为了降低算术编码的复杂度，提高编码速度，CABAC 采用了二进制的算术编码，而非其它多进制的算术编码，为此需要事先将每一语法元素转换成独一无二的二进制序列，在 H.264 中称 BIT STRING，同时为了便于后面的算术编码，尽可能降低编码复杂度，应该尽可能减小二进制序列的大小，CABAC 采用了 4 种基本二进制转换方式：Unary Binarization, Truncated unary Binarization, kth order Exp-Golomb Binarization 和 Fixed-length Binarization,相应的有 4 种二进制码 Unary code, Truncated unary code, kth order Exp-Golomb code 和 Fixed-length code。
接下来是上下文建模部分：所谓上下文建模，就是建立概率模型，对每一位待编码的比特值和概率进行预测。这些模型可分为 4 种类型. 第 1 种类型的模型必须根据它相邻的已编码的语法元素构成。 第 2 种模型仅局限于对宏块类型和子宏块类型的应用。第 3 种和第 4 种模型仅用于残余数据的编码。下表给出了 H264/AVG 中的所有语法元素和它们所用到的上下文模型索引的对应关系。

语法元素(SE)	片类型(Slice type)			
	SI	I	P,SP	B
mb_skip_flag	-	-	11~13	24~26
mb_field_decoding_flag	70~72	70~72	70~72	70~72
mb_type	0~10	3~10	14~20	27~35
moded_block_pattern(luma)	73~76	73~76	73~76	73~76
coded_block_pattern(chroma)	77~84	77~84	77~84	77~84

mb_qp_delta 60~63 60~63 60~63 60~63
prev_intra4×4_pred_mode_flag 68 68 68 68
rem_intre4×4_pred_mode 69 69 69 69
intra_chroma_pred_mode 64~67 64~67 64~67 64~67
ref_idx - - 54~59 54~59
mvd(horizontal) - - 40~46 40~46
mvd(vertical) - - 47~53 47~53
sub_mb_type - - 21~23 36~39
coded_block_flag 85~104 85~104 85~104 85~104
significant_coeff_flag[] 105~165
277~337 105~165
277~337 105~165
277~337 105~165
277~337
last_significant_coeff_flag[] 166~266
338~398 166~266
338~398 166~266
338~398 166~266
338~398
coeff_abs_level_minus1[] 227~275 227~275 227~275 227~275
表 3.3 语法元素索引表
其中 r 从 0~72 是关于宏块类型、子宏块类型、预测模式以及基于片层和宏块层的控制信息等语法元素的上下文模型索引， 对于这种类型的语法元素，其索引值’ 由公式 r=Ts+xs 计算。式中：Ts 代表上下文索引的初值，也就是表中所给出的值，xs 代表其增量值。它依赖于待编码比特位的索引值。 r 从 73~398 是残余数据的编码,其中语法元素 coded_block_pattern 的上下文模型索引值 r 计算公式同上，而其它残余数据语法元素的上下文模型索引值则由公式 r=Ts+Qs(ctx_cat)+xs 给出。式中：Qs(ctx_cat)值由语法元素及其上下文范畴(con-text category, 简称 ctx-cat 决定，具体值可见下表。值得一提的是在纯帧或纯场编码中，仅实际乃至个模型中的 277 个。

语法元素 上下文范畴(ctx-cat)
0 1 2 3 4
coded_block_flag 0 4 8 12 16
significant_coeff_flag 0 15 29 44 47
last_significant_coeff_flag 0 15 29 44 47
coeff_abs_level_minus1 0 10 20 30 39
表 3.4 语法元素与上下文范畴对应表
二进制算术编码：在 CABAC 中$对每一待编码的比特位的值(0 或)，用 MPS 和 LPS 表示，其中 MPS（ most probable symbol）表示最可能出现的状态，对应 0 和 1 中概率大的那一个；LPS（least probable symbol）表示最不可能出现的状态，对应 0 和 1 中概率小的那一个。 这样只需要一个变量值 pσ 保存 LPS 出现的概

率大小，对应 MPS 出现的概率大小可由(1-pσ)表示。同时，它把概率 pσ 量化成 64 个等级，每一概率 pσ 由其对应的索引σ 唯一给出。这样在 CABAC 中，每一个上下文模型可由两个变量唯一决定，一个是 LPS 的概率值 pσ，一个是 MPS 的值用ω表示。 由于 CABAC 中，pσ 值被量化成 64 个等级，同时，ω值在二进制编码中只有 0 和 1 两种取值。因此在 CABAC 中一共只有 128 种概率状态, 可用 7 比特来表示。

在 H.264 中，片层是自适应编码的基本单元，在一个片层数据编码完，新的片层数据到来时，CABAC 要对各概率状态进行重新初始化，初始化的重要依据 QP 参数，各片层的 QP 参数不同，则在初始化时各概率状态值是不同的。

虽然基于上下文的自适应二进制算术编码算法可以很好地解决算术编码的自适应问题，但是它也不可避免的带来了新的问题，就是关于符号概率表的储存以及查找该符号概率时所带来的复杂度增加的问题。

3.3 自适应算术编码算术及其实现

所谓自适应算术编码是在算术编码基本原理的基础上，根据信源数据来自不同符号集的特性，充分利用信源数据的统计特性，将每种类型的数据分开处理，各自拥有一个独立的统计模型单元，可以得到更高的编码效率。同时利用各种进制数据进行转换的时机,将压缩数据分成不同的数据段,将误码限制在本数据段内,当误码产生时,其影响最多从误码位到本段结束位处[10]。

现在来看一下符号的概率问题:在所有情况下,每种可能的概率都落在区间[0,1)中，而所有概率的和则为 1.这个区间包含了所有概率的可能，所以我们可以以此来对序列进行编码。每个概率会落到此区间的一个子区间，对一个序列重复这样编码，就会得到关于这个消息的一个特定区域。这个区间的任何一个数字都可以成为一个有效的编码。设为 PM(ai)某序列中 ai 的概率。因为概率之和总是为 1,那现在就让我们用这个数来划分区间[0,1)每个数所划分的区间大小取决于它的概率。

设某序列有四个字符 a,b,c,d;其各自概率如下

PM(a)=0.5,PM(b)=0.25,PM(c)=0.125,PM(d)=0.125

我们设变量 high 和 low 来表示区间的上下限，之后依照概率的变化，对上下限进行修正，从上图我们得到如下概率区间：

high 1.0 K(0) 0.0 K(2) 0.75

low 0.0 K(1) 0.5 K(3) 0.875

表 3.1 预设字符序列的概率表

我们假设字符是按照一个固定的概率表出现的，这种模式现实中确实存在，我们称之为静态模式。

编码的第一步是初始化区间 I:=[low,high)by low=0 and high =1，当读取 s1 后，区间将被重新划分为 I',而它的上下限还是叫做 high 和 low。

将新的区间 I'映射到中[low,high)，这完全是按照上表的统计特性来计算的。产生更短代码字的片断数越少，区间 I'就越大。然后在新区间重复前面的过程。

当接收新符号时，区间 I'的划分是以 low 为新下限的，

以上规则对每一步都适用，都是由第一步 low =0 且 high － low ＝1 得到的。因为我们不再需要旧的 high 和 low 的值，所以我们要重写它们：

low := low'

:=high'

设我们要编码的序列为 abaabcda 第一步如下：

low = 0
high = 0＋0.5•1 =0.5
新区间变成了 然后对 b 编码：
low = 0＋0.5•（0.5-0）=0.25
high = 0+0.5•（0.5-0）＋0.25•（0.5-0）=0.375
a：
low = 0.25
high = 0.25+0.5•（0.375-0.25）=0.3125
a ：
low = 0.25
high = 0.25＋0.5•（0.3125-0.25）=0.28125
b：
low = 0.25+0.5•（0.28125-0.25）=0.265625
high = 0.25+0.5•（0.28125-0.25）＋0.25•（0.28125-0.25）
c:
low = 0.265625+0.5•（0.2734375-0.265625）+0.25•（0.2734375-0.265625）= 0.271484375
high = 0.265625+0.5•（0.2734375-0.265625）+0.25•（0.2734375-0.265625）＋0.125•（0.2734375-0.265625）

d:
low = 0.271484375+（0.5+0.25+0.125）•（0.2724609375-0.271484375)
= 0.2723388672
high = 0.2724609375
最后一个是 a:
low = 0.2723388672
high = 0.2723388672＋0.5•（0.2724609375-0.2723388672）
= 0.2723999024
这样最后得到的区间是 [0.2723388672, 0.2723999024)接下来是实现编码。我们可以不加处理地存储区间[0.2723388672, 0.2723999024)但这很低效。因为我们知道这个区间是由前面那个序列得到的，所以我们只储存这个区间的任意一个数就足够了。下面的辅助定理可以说明这个原理
辅助定理 1 所有等长消息的编码形成的区间构成一个完整的区间 I:=[0,1)。
这可以从图 2 中看出来。这个定理的直接结论就是无限小的区间可以代表一个无限长的序列。实际中，不存在无限长的序列，但一个很长的序列所导致的小区间可能会使一般计算机无法处理。一种解决办法是将长序列分段处理。最后一个例子中我们以处理 0.27234 为例。虽然实际中常常不会发生已知消息什么时候会结束这样的情况(因为传输距离一般都很长)，但这里我们仍旧假设我们已知。
至于解码，要解码就先得知道编码背景是怎样的。给出 V:=Code(S)我们就要恢复原序列 S。我们假设已知消息长度 l 已知。第一步中我们先将 V 与每个区间 I’:=[K(ak-1), K(ak)]
进行比较，找出包含 V 的区间，它取决于序列的第一个符号：s1 我们在解码时，要修正概率区间来算出下一个符号：
low’ := low＋K(ai-1)•(high-low)

high’ := low＋K(ai)•(high-low)
i 必须满足：
low <= V <= high
ai 是编码序列的下一个符号。这次又是一个一般方程的特例。这种重复与编码时很像，所以解码应该不成问题。
解码范例：

我们现在来对前面编码序列进行解码，前面得到的值为 V=0.27234 ，假设已知长度 l=8 可以看出 [0,0.5)之内，我们可以得到一个’a’，然后再设
low =0
high =0.5
再次重复可以看到 V 在以下区间内：
low = 0＋0.5•（0.5－0）=0.25
high = 0＋0.75•（0.5－0）=0.3125
解码得到’b’，相应的边界是划线部分。下一轮：
low = 0.25＋0•（0.3125－0.25）=0.25
high = 0.25＋0.5•（0.3125－0.25）=0.28125
得到的结果是’a’。略去重复部分，可以最后结果是：
low = 0.272338672＋0•（0.2724609375－0.2723388672）=0.2723388672
high = 0.272338672＋0.5•（0.2724609375－0.2723388672）=0.2723999024
虽然在简单意义上，我们认为所有的符号独立的随机出现的. 然而,它其实通常很大程度上与语言环境有有关,以德文为例,字母’ u ’的平均概率约为 4.35% . 但
如果它的前面是一个’ q ’,这时字母’ u ’的概率的增幅几乎达到 100% 而以前面 N 个字母的情况为参考的模式称为 N 阶模型.
最有用的编码方法是为各种不同的数据源共用而开发的方法。这意味着通常数据来源的概率分布是未知的。它也不能简单的去数当前的信号流。只考虑传真文件：当我们开始读第一页时，也许后续几页还正在传输中，也就是说，不能同时处理信号的全部而仅仅是其中的一个片断。所以唯一能作的有用的事就是"估计"。很明显，我们所作的估计要与当前生成的信号相匹配。这就是本设计要采用的自适应模式。先来看下面的例子：
为了便于说明，下面将采取，0 阶的意思是认为概率只针对当前符号，而并不考虑符号的前续符号。为实现此目标，先定义一个足够长的 K 序列。K 中每个元素的都是字母表 S 中的字母。然后以 0 来初始化该序列中的所有元素。在每次编码前，将元素从输入流中取出，同时使符号计数器中的值和序列的熵增加。然后将此序列中符号的概率重新作分配。
s K(a) K(b) K(c) K(d)
a 1 0 0 0
b 1 1 0 0
c 2 1 0 0
d 2 1 0 1

表 3.2 0 阶自适应模式符号概率分配表
以下面的字母表为例:

A=a,b,c,d
对序列 abad 编码。上表列出了对其概率的计算结果。该结果证明了在新符号读入后，概率分布正确的反应了符号传输的实际情况。这说明当序列很长时，计算结果与实际情况将会非常的接近。
在对输入符号进行逐字编码后，解码器即可对生成的编码结果进行正确的解码，解码步骤是编码的逆过程。这样编码的优点是只要对数据流所传送的符号进行编解码就可以了，而不用去管它的前续和后续符号是怎样的。
基于模型的自适应算术编码主要存在两个缺点：
（1） 在整个算术编码的过程中会产生一个位数逐步增加的实数，该实数对应的概率区间对应着编码信息。这个实数的值稍有变化，会导致解不出原来的信息，后面译出的码字全部错误。但是由于计算机的精度有限，所以在实际的计算时是不可能无限地划分下去的，也就是说到了一定的时候，计算机已经无法识别两个区间的范围了。这样就给算术编码的最终结果带来误差。
（2） 只有当信源完整地把一段符号序列发送完之后，编码器才能确定一段子区间与之对应，编出相应的码字。这不但要占用相当大的存储空间，还增加了编码延时，这对实时系统是十分不利的，而且也非常不利于解码方的接受。因为编码时是根据所有的这些符号计算出每个符号对应的概率，然后利用这些概率信息进行编码的。所以必须有一种可以增加输出的算法来更好的完成这个要求。为了解决这两个问题，Written，Neal 和 Cleary 提出了一种建立模型的方法（WNC 算法）来改进算术编码的性能。由于在英文中最多只有 256 个字符，也就是符号所对应的数组最多只需要 257 个元素。在初始化这个数组后，WNC 算法利用输入符号动态更新这个数组，假设数组的积累频率就是它的出现频率，完成编码的操作。虽然这些积累频率是一个局部的频率，但是如果解码方以同样的操作进行，则这个过程是完全可逆的。通过这个模型的建立就可以解决上面的第一个问题。另外由于 WNC 算法是采用模型的方法，不需要考虑符号的全局概率，而只需要根据该符号现在的积累情况，就可以采用增加传输和接收的方法，完成当前符号的编码，并且将其传输出去，而解码方就可以根据完全类似的过程进行解码，得出这个符号来。这样就可以解决上面的第二个问题。由于解码算法是编码算法的逆过程，有着近似相同的动作，所以在文章后面只介绍编码算法，而不深入讨论解码算法。

第四章 算术编码系统的实现

4.1 软件模块设计

本软件主要实现基于算术编码的数据压缩和解压，也就是将第三章对自适应算术编码方法的研究用具体的开发工具-- VC++ 进行实现。
该软件完整的工作过程是：
1）数据文件压缩过程：首先在界面输入文件的压缩路径，然后通过设定的压缩模式（这里设定为 0 阶模式）对数据文件进行压缩。压缩过程是依照文件在硬件中的存储方式将其完全当作二进制文件来读取并进行压缩的。压缩后生成后缀名为 .ac 的文件，将其存储在预定的输出路径中。
2）数据文件解压过程：根据在界面输入的文件路径，按照其中存储的模式信息对其读取后再通过解压缩模块将其解压。解压的过程完全是压缩的逆过程，因为算术编码是一种无损压缩，并且其压缩处理的模式也决定了它解压生成的将是与

压缩前完全一样的数据文件，因此，通过解压最后生成的文件即是与原文件完全一致的文件。
模块设计：本设计是以算术编码的方式实现对数据的压缩，所以系统模块来划分，包括输入模块、压缩模块、解压模块以及输出模块。这样的划分功能明确，下面即从模块划分的内聚性和耦合性方面进行分析比较。
内聚：是信息隐蔽功能的自然扩展。内聚的模块在软件过程中，完成单一的任务，同程序其他部分执行的过程交互很少，简而言之，内聚模块（理想情况下）应该只完成一件事。即争取模块划分的高内聚。
本软件模块划分是从顺序和功能的聚合方式划分，因此本设计模块中各个部分的都是完成一个具体功能的必不可少的组成部分，该模块中的所有部分都是为了完成一项具体功能而协同工作、紧密联系、不可分割。因此本设计的内聚方式为功能内聚，所以内部联系度、清晰性、可重用性、可修改性和可理解性这 5 个与聚合性有关的属性都达到了良好的标准。
耦合：是程序结构中模块相互关联的度量。耦合取决于各模块间接口的复杂程度、调用模块的方式，以及哪些信息通过接口。
本软件模块划分使得模块之间没有直接关系，其所采用的耦合方式为非直接耦合，即它们之间的联系完全是通过主模块的控制和调用来实现的，并且对数据做何种处理完全由主模块（界面）决定。因此在对修改的敏感性、可重用性、可修改性及可理解性这 4 个与耦合性相关的模块属性方面也都达到了良好的标准。

图 4.1 软件模块设计

4.2 软件模块的具体实现
按照软件模块的划分，本软件共分为四个模块，分别为输入模块、压缩模块、解压模块、输出模块。输入模块和输出模块主要是用 C++类库中的方法执行对文件的确认/建立、读取/写入、以及对文件的相关位置进行标记；而压缩模块和解压模块的主要功能则是用算术编码的方法对指定的文件进行编码和解码的操作，具体实现下面分输入输出、压缩、解压三个部分进行说明。

4.2.1 输入输出模块的实现
这一部分是对源文件的读写，功能实现比较便捷，在 VC 中，采用标准库中的 fstream 类实现对源文件的打开和读取，因为 fstream 类已经包含有对文件按二进制读取等功能，所以可以很容易实现对文件的打开、关闭、读写和（输入、输出）。

图 4.2 输入输出模块流程图
在以对话框为基础的软件界面上添加文本输入框，一个作为源文件路径的输入，另一个作为目标文件路径的输入,在此处文本框接收的路径将直接赋给处理文件打开关闭以及读写的模块，

图 4.3 输入输出路径接收的界面

新建文本框后，使用 VC++中的类向导功能，对两个文本输入框进行编辑。设置 ID 和消息影射，以用来将获得的路径信息传递给编码解码的模块进行数据的输入与输出。
调用的方法包括 VC++类库中 fstream 类中 .open, .read 和 .write 等方法：
下面代码是实现模块流程图的具体方法：

```
//定义 SetFile 类将指定的文件对象转换为 ArithmeticCoderC 类的成员变量
void ArithmeticCoderC::SetFile( fstream *file )
{
mFile = file;
}
//将比特数字记录到比特缓冲区 mBitBuffer ，当读取计数变量 mBitCount 满
8 //位后，将缓冲区记录写入文件
void ArithmeticCoderC::SetBit( const unsigned char bit )
{
// 向缓冲区增加比特
mBitBuffer = (mBitBuffer << 1) | bit;
//同时计数
mBitCount++;
//如果计数满 8 位，就将其写入文件
if(mBitCount == 😎// buffer full
{
// 写入操作
mFile->write(reinterpret_cast(&mBitBuffer),sizeof(mBitBuffer));
mBitCount = 0;
}
}
//根据文件是否读完决定是否继续对文件进行读取
unsigned char ArithmeticCoderC::GetBit()
{
//当比特计数为 0 时
if(mBitCount == 0) // buffer empty
{
if( !( mFile->eof() ) ) //若未读完
mFile->read(reinterpret_cast(&mBitBuffer),sizeof(mBitBuffer));//继//
续读取

else//否则清空比特缓冲区，并置比特计数为满
mBitBuffer = 0;

mBitCount = 8;
}
// 从缓冲区取出比特，继续处理
unsigned char bit = mBitBuffer >> 7;
mBitBuffer <<= 1;
```

```
mBitCount--;
return bit;//返回比特值
}
```

4.2.2 压缩模块的实现

对于压缩算法来说 ，最重要的部分是算法的设计与实现，接下来介绍的就是压缩部分的实现。

图 4.4 压缩模块功能实现的界面

对于压缩模块来说，最主要的部分就是对数据进行压缩的算法，压缩模块将接收"源文件"和"输出文件"所获得的路径信息，然后将读取的数据进行算术编码，主要步骤为：

1） 读取缓冲数据信息；
2） 对缓冲信息进行编码；
3） 暂存缓冲信息
4） 判断是否读取完全，若是，进行下一步

否则转到第一步；

5） 输出编码内容；
6） 添加编码信息以便解码

压缩模块流程见下图：

图 4.5 压缩模块流程图

在软件界面中添加"压缩"按钮，使用 VC++中的类向导功能，设置 Object ID 和功能方法，并将其加入类 CACodingDlg。增添消息影射方法：BN_CLICKED。双击后，在增添对数据进行压缩的代码，从而实现压缩功能。

```
//编码实现
void ArithmeticCoderC::Encode( const unsigned int low_count,
const unsigned int high_count,

const unsigned int total )
{
//将下步要进行编码的区间分为 total 个区间
mStep = ( mHigh - mLow + 1 ) / total; //
// 更新上界
mHigh = mLow + mStep * high_count - 1; // interval open at the top => -1
// 更新下界
mLow = mLow + mStep * low_count;
//用 e2/e3 方法
while( ( mHigh < g_Half ) || ( mLow >= g_Half ) )//当上界低于 1/2 或下界高于 1/2 时
{
//当上界低于 1/2,
if( mHigh < g_Half )
{
```

```
SetBit( 0 ); 置比特位为 0
mLow = mLow * 2;
mHigh = mHigh * 2 + 1;
// 应用 e3 方法
for(; mScale > 0; mScale-- )
SetBit( 1 );
}
//当下界等于 1/2 时
else if( mLow >= g_Half )
{
SetBit( 1 );置比特位为 1
mLow = 2 * ( mLow - g_Half );
mHigh = 2 * ( mHigh - g_Half ) + 1;
//应用 e3 方法
for(; mScale > 0; mScale-- )
SetBit( 0 );
}
}
while( ( g_FirstQuarter <= mLow ) && ( mHigh < g_ThirdQuarter ) )
{

mScale++;
mLow = 2 * ( mLow - g_FirstQuarter );
mHigh = 2 * ( mHigh - g_FirstQuarter ) + 1;
}
}
//当编码结束时的收尾处理
void ArithmeticCoderC::EncodeFinish()
{

// mLow 和 mHigh 的分布情况有两种，所以用两个比特来区分它们,
// which means that two bits are enough to distinguish them.
if( mLow < g_FirstQuarter ) // mLow < FirstQuarter < Half <= mHigh
{
SetBit( 0 );
for( int i=0; i
SetBit(1);
}
else // mLow < Half < ThirdQuarter <= mHigh
{
SetBit( 1 ); //解码器自动会在其后加 0
}
// 清空输出缓冲区
```

```
SetBitFlush();
}
```

4.2.3 解压模块的实现

压缩只是一种中间状态，目的是实现更快地传输或减小存储空间，但处于压缩状态的文件是不可以使用的，所以必需要有解压步骤，实现对压缩文件的解压，恢复文件的原本状态，因为算术编码是无损压缩，所以实现压缩文件无失真的解压是必然的步骤。解压是压缩的逆过程，具体流程如下：

图 4.6 解压功能实现的界面

接下来是解压模块，解压模块的解压过程是压缩方式的逆过程，解压模块同样将接收"源文件"和"输出文件"的路径信息，然后将已进行压缩编码的数据进行码处理，主要步骤为：

1） 读取压缩编码信息，以确定解压模式；
2） 读取缓冲信息；
3） 对缓冲信息进行解码；
4） 暂存缓冲信息判断是否读取完全，若是，进行下一步
否则转到第二步；
5） 输出编码内容；

图 4.7 解压模块流程图

```
void ArithmeticCoderC::DecodeStart()//建立解码缓冲区
{
for( int i=0; i<31; i++ ) // 还是只使用 31 位比特
mBuffer = ( mBuffer << 1 ) | GetBit();
}

unsigned int ArithmeticCoderC::DecodeTarget( const unsigned int
total )/*确定解码的开始区间*/

{

mStep = ( mHigh - mLow + 1 ) / total;

return ( mBuffer - mLow ) / mStep;
}

void ArithmeticCoderC::Decode( const unsigned int low_count,
const unsigned int high_count )//解码实现函数
{
//重新确定解码上下界
mHigh = mLow + mStep * high_count - 1;
mLow = mLow + mStep * low_count;
//分情况确定解码模式
```

```
while( ( mHigh < g_Half ) || ( mLow >= g_Half ) ) / /mHigh < Half <= mLow
{
if( mHigh < g_Half )
{
mLow = mLow * 2;
mHigh = mHigh * 2 + 1;
mBuffer = 2 * mBuffer + GetBit();
}
else if( mLow >= g_Half )
{
mLow = 2 * ( mLow - g_Half );
mHigh = 2 * ( mHigh - g_Half ) + 1;
mBuffer = 2 * ( mBuffer - g_Half ) + GetBit();
}
mScale = 0;
}

while( ( g_FirstQuarter <= mLow ) && ( mHigh < g_ThirdQuarter ) )
{
mScale++;
mLow = 2 * ( mLow - g_FirstQuarter );
mHigh = 2 * ( mHigh - g_FirstQuarter ) + 1;
mBuffer = 2 * ( mBuffer - g_FirstQuarter ) + GetBit();
}
}
```

最后，为了方便本设计软件的演示，在操作界面加入了关键代码的显示按键，可以实现对与运行程序在同一文件夹中的名为"ArithmeticCoderC"的 .cpp 文件以文本文件的方式打开。实现界面如下：

图 4.8 包含关键代码查看的软件界面

图 4.9 关键代码查看

4.3 压缩效率分析

按字节对 250 万字的汉语语料库进行统计，结果表明，153 个符号的覆盖率超过 99.99% ，199 个符号的覆盖率达到 100%。对数据压缩领域普遍认可的 Calgary 语料库中的英语文本文件及计算机源程序文件做统计，100 个符号的覆盖率超过 99. 99%，104 个符号则实现完全覆盖。因此,探讨数据压缩对不同类型字符文件的压缩做用，不仅仅是对语言文字文本压缩有意义。

为了更直接的展示出本设计所实现的压缩性能，下面将会使用本设计对一些文件进行压缩，文件类型包括 .bin, .txt, .doc, .mp3 。

参数

类型 压缩前 压缩后 压缩比 压缩时长

.txt 文件 462KB 354KB 0.766 2.6s

.doc 文件 581KB 497KB 0.855 3.1s
. bin 文件 541KB 483KB 0.893 2.9s
.mp3 文件 1075KB 1072KB 0.997 4.8s

表 5.1

测试环境：
系 统：Windows XP
处理器: AMD Sempron64
内 存：单条 512MB
通过以上比较可以看出，使用本设计软件对文件进行压缩时，效率最优的是.txt 文件，其次是文件.doc，下来是.bin 和 .mp3 文件。.txt 文件的压缩比达到了 0.766，分析其原因，应该是由于本设计软件是针对二进制数据文件设计的，即将目标文件作为二进制数据文件进行处理，而并没有对不同类型的文件进行特别处理，所以对于不同文件，其数据特性越接近二进制数据流，即其中二进制字符"0"与"1"的分布越偏离 1:1，本设计软件对其的处理能力就越强。比如前面对.txt 文件进行处理时，因为.txt 文件是采用 ASCII 码为主要记录格式的文字信息记录文件，所以其中对各种字符和标记符号的编码特性是依照 ASCII 码编码特性的，因此其编码中二进制字符"0"与"1"的分布并不十分接近 1:1 的概率比值，而.doc 文件因为与.txt 文件相比，采用了较多的控制符号，所以其文件中二进制字符"0"与"1"的分布与.txt 相比更靠近 1:1，所以可以获得较.txt 文件为高的压缩比；对于.bin 文件，由于其本身就是二进制文件，所以它自身的二进制字符"0"与"1"的分布在通常情况下已经较为接近 1:1 的概率比值，所以对其进行多次压缩测试时，压缩比在 0.6 到接近 1.0 的范围浮动，而其文件的概率大多分布在 0.8 左右，所以上表采取了与这一数值相近的一个测试结果来作为不同类型文件压缩效率的比较。但是对一些有特殊用途的应用软件使用的文件如 mp3 等文件，加之其本身就是有较大压缩效果的文件，本设计软件的处理能力就不能使人非常满意了。

4.4 软件设计的优点与不足

本软件设计的优点在于：
1）对二进制数据流的高压缩比：在二进制数据"0"、"1"的出现概率接近 1:1 的时候，它有着非常有优势的压缩比；

2）为了实现日后对其进行改进，本设计在模式设置的时候将压缩解压模块的自适应模式模块设计为独立的，可以选择不同的压缩解压模式进行。当日后要对其扩充或改变其自适应阶数，及相关参数，使其适合于特别的文件类型。
3）使用简单方便的文件流类 fstream 对数据文件进行读写操作，因为 fstream 是由标准 C/C++ 类库提供的，所以具有良好的通用性和可移植性。
本软件设计的不足：
1）缺少对进程的控制，在对文件进行压缩解压的时候，由于不能实现进程控制，所以不能及时了解到压缩的进度，而且在压缩解压过程中途也不能对操作进行取消；
2）由于对数据文件操作完全是利用标准 C/C++ 类库提供的函数实现的，所以在运行过程中，本设计存在对 CPU 的占用问题，当处理较大型文件的时候，会出现 CPU 占用率过高的情况，可能影响其他软件的同时运行。

4.5 软件设计值得改进的地方
本软件实现了用自适应模式对二进制数据流/文件进行编解码处理，但还可以进行一些改进，用于提升性能：
1）可以增加对压缩解压过程的进程控制，使得在压缩和解压过程中，用户可以根据情况随时对编码过程进行操作控制，这可以使软件使用的灵活性大大增加，并且有可能在实际应用过程中，避免一些意外状况的发生；
2）增加对占用 CPU 的问题的考虑，使得在软件运行过程中，可以多些对多任务的支持，而不会因为本软件的运行占用 CPU 而影响其他软件并行运行。

第五章 算术编码总结

算术编码是一种无失真的编码方法，能有效地压缩信源冗余度，属于熵编码的一种[11]。算术编码的一个重要特点就是可以按分数比特逼信信源熵，突破了Huffman 编码每个符号只不过能按整数个数比特逼近信源熵的限制。对信源进行算术编码，往往需要两个过程，第一个过程是建立信源概率表，第二个过程是对信源发出的符号序列进行扫描编码。而自适应算术编码在对符号序列进行扫描的过程中，可一次完成上述两个过程，即根据恰当的概率估计模型和当前符号序列中各符号出现的频率，自适应地调整各符号的概率估计值，同时完成编码。尽管从编码效率上看不如已知概率表的情况，但正是由于自适应算术编码具有实时性好、灵活性高、适应性强等特点，在图像压缩、视频图像编码等领域都得到了广泛的应用。经过算术编码，我们描述了一种高效率的数据压缩的编码方法，并已证明它符合 bijective 编码的要求。现在我们不仅通过整数实现，而且也通过浮点数实现了该压缩算法。我们看到算术编码可以按输入顺序进行，对每个编码不需得到其全部消息就可将信息的已编码部分输出。这种特性可用三种按比例变化的方式压缩，可以加大工作间隔而不会溢出甚至可以按需要限定其每次处理的码长。我们也看到一般编码的有效限度并注意所增加输入序列长度，即可使其平均码长与熵接近而有此熵值与选择的压缩模式有关。我们注意到在一种模式质量被实际限定后，任何编码器都可以满足这个压缩比。这里，我们同样认识到算术编码的另一优势 - 因为它的统计模式可以方便切换，故可为输入的数据挑选最适合的模式。

虽然算术编码在过去的几十年开始使用并被看好，但现在和未来却产生了一些变化和新动向。然而有时人们声称已发明更好的算法而且多次测试得出其比以前任何方法都好 - 但却忽略了算术编码是一种无损编码。因为 Shannon 定理确定地告诉我们压缩后的熵是不可能比其压缩前的熵更低的。一个人可以把数据的冗余任意删除，但要达到熵值是几乎不可能的。

然而我们可以至少从两方面使我们的算法变得更好存储器使用率和编码速度。如果算术编码使用存储器的话，那这将会是最好的压缩方法。对于简单模式，它一般只用固定容量的存储器。而且它生成了无法被压缩得更小的编码。要注意这种编码的熵极限取决于如下模式：$H(S) \leqslant HM(S) \leqslant |Code(S)|$。我们应区分源序列本来的熵值 H(S)和我们采用的编码模式所能得到的熵值下限 HM(S)。但如果选择了不适当的模式，即使算术编码能达到 HM(S)，它仍旧离最佳压缩效果很远。因为输入数据一般不可预测，我们不得不为特别的编码内容找到适当的模式，算术编码允许使用专门设计的标准组件，因此编码器能在不同模式间切换，甚至在编码过程中也可进行切换。现在已经开发出很多模式，最流行的模式系列之一是

PPM (Prediction with Partial Match) －部分匹配预测编码。它对输入长度可能变化的内容进行压缩非常有效，而很多其他高效编码方式则需要更多可靠的正确的条件才行。
随着算术编码器速度的提高，集成编码器变得常见，但现代 CPU 浮点运算能力的提高可能会改变这种趋势，我们已经知道基于浮点的算术编码的实现是可能的一种非常有效的集成实现是界限编码器它表现出对比特数据按比例压缩的良好特性。因此替换现有的 CPU 中对比特不敏感的部分就是现在的主要问题。有些报道说有人将压缩速度提升 50%而同时码长只增加 0.01%，对于这些数字则须要小心对待，因为它们只反映出了编码器的表现，而不是因为采用了新的有效的模式。可以看出，算术编码的最有趣的研究领域是编码模式。对一个好的编码器来说编码大小、存储器使用率和编码速度作为次要因素都取决于所选择的模式，而算术编码发展的过程本身就是这样的例证。

参考文献

[1] D.A.Huffman. A Method for the Construction of Minimum Redundancy Codes. [J] Proceedings of the Institute of Radio Engineers. 40(9):1098-1101, September 1952.
[2] Elias P. Information theory and coding.[J] NY: New York , Mc2 Graw2Hill , 1963
[3] Rissanen [J] . IBM J . Res. Develop. , 1976 ,20(3) :198
[4] Rissanen J , G.G.Langdon. [J]. IBM J . Res. Develop. , 1979 ,23 (2) :149
[5] Witten I H , Neal R M, Cleary J G. Communications of the ACM . [J]. 1987 ,30(6) :520
[6] 吴乐南. 数据压缩（第二版）[M]. 四川：电子工业出版社
[7] C.E.Shannon. A Mathematical Theory of Communication.[J]. Bell Syst.Tech, 1948,7:398-403
[8] David Salomon. 数据压缩原理与应用[M]. 四川：电子工业出版社
[9] Paul G. Howard. Arithmetic Coding for Data Compression[EB/pdf]. http://www.cs.duke.edu/~jsv/Papers/HoV94.arithmetic_coding.pdf
[10] Amir Said. Introduction to Arithmetic Coding Theory and Practice[R]. Palo Alto, CA: Hewlett-Packard Laboratories,2004
[11] Amir Said. Comparative Analysis of Arithmetic Coding Computational Complexity. [EB/pdf]. http://www.hpl.hp.com/techreports

附 录

算法源代码

```
#include “stdafx.h”

#include “ArithmeticCoderC.h”
```