声明

本论文题目：算术编码算法的分析与实现，作者：叶叶，于 2010 年 10 月 16 日在编程论坛上发表。页面地址：http://programbbs.com/bbs/view12-29356-1.htm 。本论文全文及相关配套程序可以在上述页面中下载。请尊重他人劳动成果，转载或引用时请注明出处。

目录

算术编码算法的分析与实现

作者：叶叶（网名：yeye55）

摘要：分析了算术编码的理论基础，着重介绍 WNC 算法的实现方式。详细讨论了算术编码原理、正规化操作、WNC 算法代码实现等技术。给出了一个切实可行的应用程序。

关键词：算术编码；正规化；Delphi

中图分类号：TP301.6

1 前言

早在 1948 年 C. E. Shannon 提出信息论[1]的时候，就提出了算术编码的思想。但是经过多年的研究，许多学者认为算术编码是无法实现的。算术编码要求进行无限精度的实数运算，这在仅能进行有限精度运算的计算机系统上是无法进行的。随着研究的深入，终于在 1987 年 Ian H. Witten、Radford M. Neal 和 John G. Cleary 发表了一篇论文[2]，提出了一种基于整数运算的算术编码实现算法。该算法后来被命名为 CACM87，并应用于 ITU-T 的 H.236 视频编码标准。也有学者根据作者姓名将该算法称之为 WNC 算法。WNC 算法是一个实用性算法，它可以应用在许多方面。在 Witten 等人的论文[2]中给出了一个使用 C 语言编写的 WNC 算法实现程序的源代码（以下简称“WNC 源代码”）。在许多时候，WNC 源代码已经作为算术编码的范本程序来使用。本文将分析算术编码的理论基础，并着重介绍 WNC 算法的实现方式。同时给出一个在 Delphi 7.0 下开发，使用算术编码算法压缩数据的应用程序。

2 理论

2.1 编码

算术编码将整个要编码的数据映射到一个位于[0,1)的实数区间中。并且输出一个小于 1 同时大于 0 的小数来表示全部数据。利用这种方法算术编码可以让压缩率无限的接近数据的熵值，从而获得理论上的最高压缩率。

算术编码进行编码时，从实数区间[0,1)开始。按照符号的频度将当前的区间分割成多个子区间。根据当前输入的符号选择对应的子区间，然后从选择的子区间中继续进行下一轮的分割。不断的进行这个过程，直到所有符号编码完毕。对于最后选择的一个子区间，输出属于该区间的一个小数。这个小数就是所有数据的编码。现在来举个例子。假设一份数据由“A”、“B”、“C”三个符号组成。现在要编码数据“BCCB”，编码过程如图 2.1 所示。

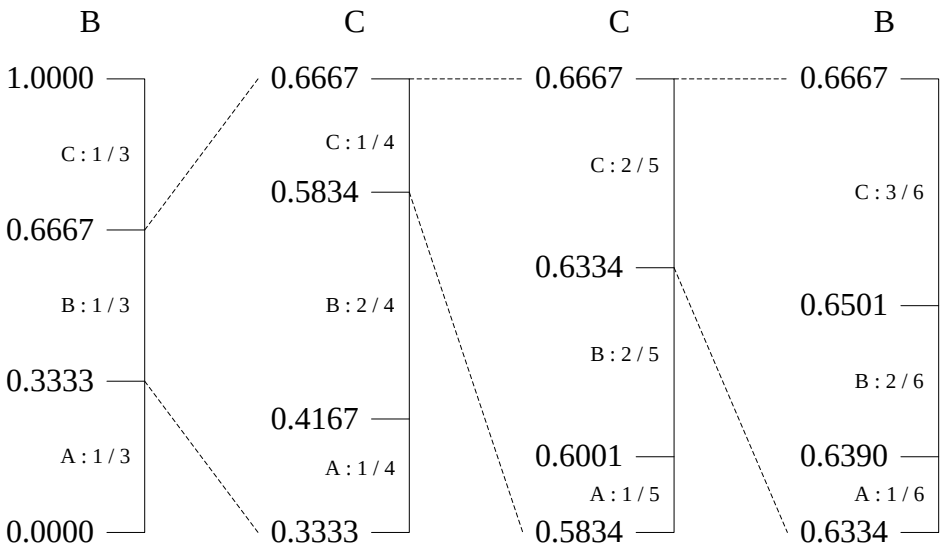

图 2.1 “BCCB”的编码过程

首先说明一点，这里使用的是自适应模型。也就是说一开始时，三个符号的频度都是 1。随着编码的进行再更新频度。另外，在计算时理论上要使用无限小数。这里为了说明方便，四舍五入到小数点后 4 位。

观察图 2.1 可以发现算术编码的过程。首先，算术编码是从区间[0,1)开始的。这时三个符号的概率都是 1 / 3，按照这个概率分割区间。第一个输入的符号是“B”，所以我们选择子区间[0.3333,0.6667)作为下一个区间。输入“B”后更新频度，根据新的概率对区间[0.3333,0.6667)进行分割。这时输入的符号是“C”，我们可以选择子区间[0.5834,0.6667)。继续更新频度、分割区间、选择子区间，直到符号全部编码完成。我们最后得到的区间是[0.6390,0.6501)。输出属于这个区间的一个小数，例如 0.64。那么经过算术编码的压缩，数据“BCCB”最后输出的编码就是 0.64。

2.2 解码

算术编码进行解码时仅输入一个小数。解码前首先需要对区间[0,1)按照初始时的符号频度进行分割。然后观察输入的小数位于那个子区间。输出对应的符号，选择对应的子区间，然后从选择的子区间中继续进行下一轮的分割。不断的进行这个过程，直到所有的符号都解码出来。整个过程相当于编码时的逆运算。

在我们的例子中，输入的小数是 0.64。首先，初始时三个符号的概率都是 1 / 3，按照这个概率分割区间。观察图 2.1 可以发现 0.64 落在子区间[0.3333,0.6667)中，于是可以解码出“B”。并且选择子区间[0.3333,0.6667)作为下一个区间。输出“B”后更新频度，根据新的概率对区间[0.3333,0.6667)进行分割。这时 0.64 落在子区间[0.5834,0.6667)中，于是可以解码出“C”。按照上述过程进行，直到所有的符号都解码出来。可见，只需要一个小数就可以完整还原出原来的所有数据。

3 改进

3.1 整数运算

上一节中描述的算法，在当前的计算机系统上是很难实现的。尤其是无限精度的实数运算。所以在实现的时候，需要对算法做一些改进。使得它可以在当前的计算机系统上较快的运行。当然，这种改进是以降低运算精度为代价的。也就是说，这种改进实际上会降低算法的压缩率。但是，它会使算法的实现成为可能。

观察前面描述的算法过程可以发现，运算时区间的上下沿都是小于 1 的小数。那么我们可以省略 0 和小数点，仅仅使用小数的尾数来表示小数。省略 0 和小数点后的尾数，实际上就是一个无限大的整数。使用无限整数的部分高位来表示整数，并在这些整数上进行整数运算就可以模拟出实数运算。在我们的例子里，可以使用区间[3333,6667)来表示区间[0.3333,0.6667)。最后可以输出 64 来表示 0.64。

另外，分割区间、选择子区间的过程，相当于将一个区间映射到另一个更小的区间中（以下简称“映射区间”）。如果我们知道一个符号的频度。以及符号值小于该符号的其它符号的频度总计（以下简称“累积频度（Cumulative Frequency）”）。还有到目前为止所有符号频度的总计（以下简称“总计频度（Total Frequency）”）。那么就可以根据这些频度信息，从当前区间中计算出映射区间。计算的公式如下。

```
Range = High - Low + 1
High = Low + Range * (CumFreq + Freq) div Total - 1
Low = Low + Range * CumFreq div Total
```

其中 Low 表示区间的下沿；High 表示区间的上沿；Range 表示区间的范围；Freq 表示符号频度；CumFreq 表示累积频度；Total 表示总计频度。这些变量中保存的都是整数，并进行整数运算。其中 div 表示整除。另外需要注意一点，这里使用闭区间[Low,High]，而不是使用右开区间[Low,High)。

在我们的例子里，实数运算时四舍五入到小数点后 4 位。那么在整数运算时可以采用 4 位整数来进行。初始区间可以设定在[0,9999]的闭区间中。按照上述公式进行编码计算所得的结果如表 3.1 所示。

输入数据	输入符号	映射区间	区间范围
“”		[0000,9999]	10000
“B”	B	[3333,6665]	3333
“BC”	C	[5832,6665]	834
“BCC”	C	[6332,6665]	334
“BCCB”	B	[6387,6498]	112

表 3.1 整数运算的区间变化

将表 3.1 中的数据与图 2.1 中的数据进行对比可以发现，整数运算会降低运算精度。整数运算时最后映射到区间[6387,6498]，实数运算时最后映射到区间[0.6390,0.6501)。由于精

度降低运算出现了误差，但是我们仍旧可以输出 64 代替 0.64 来表示整个数据。所以这种精度的降低是在允许的范围内。

在解码的时候也可以进行整数运算。根据输入的整数数值、当前区间的下沿和总计频度，可以计算出一个估算出来的累积频度（以下简称“估算频度（Estimate Frequency）”）。其计算公式如下。

```
Range = High - Low + 1
EstFreq = ((Value - Low + 1) * Total - 1) div Range
```

其中，Value 表示输入的整数数值；EstFreq 表示估算频度。利用估算频度在当前的累积频度表中查找，当满足 CumFreq ≤ EstFreq ＜ CumFreq + Freq 的条件时，就可以解码出一个符号。利用解码出的符号可以得到对应的累积频度和频度。根据这些频度信息，可以从当前区间中计算出映射区间。这一点同编码时是一样的。计算出映射区间后，更新对应符号的频度，又可以进行新的一轮解码。

在我们的例子中，输入的整数数值是 64。但是 64 本质上是 0.64，所以在参与运算时要将 64 扩展成 6400。初始时区间的范围同编码时是一样的，从[0,9999]开始。利用 6400 进行解码，其过程如表 3.2 所示。

映射区间	估算频度	累积频度			解码符号
		A	B	C	
[0000,9999]	1	0	1	2	B
[3333,6665]	3	0	1	3	C
[5832,6665]	3	0	1	3	C
[6332,6665]	1	0	1	3	B

表 3.2 整数运算的解码过程

可以看出利用一个整数数值 64，就可以解码出全部数据。另外，观察解码过程可以发现。在解码时不仅要计算映射区间，还要计算和查找估算频度。所以算术编码的解码过程通常要比编码过程慢。

在本小节中给出的计算公式都来自 WNC 源代码。观察这些计算公式可以发现，有许多运算是重复的。这意味着，这些公式还有改进的可能。在本文的第 4 节中将给出改进后的计算方法。

3.2 正规化

上述算法实际上是无法实现的。观察表 3.1 可以发现，随着编码的进行区间范围会越来越小，最后区间范围会趋向 0。如果编码较长的数据，区间范围为 0 时就无法继续编码。解决这一问题的方法是使用正规化（Renormalization, 又称“归一化”）。正规化操作就是当区间的上下沿满足一定的条件时，将一定的位数从区间中移出，同时对区间进行一次放大。使用正规化操作，可以在有限区间上模拟无限区间的运算。当然这种模拟同样会降低精度，但是它让无限区间的运算成为可能。下面就来介绍正规化操作的过程。

上一节已经说过，区间的运算可以转换为整数运算。而区间的上下沿都是用整数来保存。

在实现的时候，都是进行二进制整数运算。在本节中为了说明方便全部使用二进制整数来表示区间的上下沿。那么对于一个区间的上下沿有可能出现以下两中情况。

[00101101, 01001011]	或	[10101101, 11001011]
情况 1		情况 2

可以发现这两种情况中，区间上下沿的最高位都是相同的。根据前面对算法的描述，计算时区间总是映射到一个更小的区间中。那么当区间上下沿的最高位相同时，在后续的计算中最高位将不会再发生变化。这意味着我们可以将最高位移出区间并输出。同时将区间的下沿左移 1 位，将区间的上沿左移 1 位加 1。这样在输出一个二进制位的同时，对区间进行了扩展。其过程如下所示。

$$0 \leftarrow [01011010, 10010111] \quad \text{或} \quad 1 \leftarrow [01011010, 10010111]$$

区间的上下沿还会出现其它情况。特别的，如果区间上下沿的最高位不相同，而次高位与最高位只相差一个 1。那么可能会出现一个极端的情况，如下所示。

$$[01111111, 10000000]$$

以上区间的范围只有 1。实际上此时已经无法继续编码了。为了避免出现这种情况，当区间上下沿的最高位不相同时，还需要检查次高位。如果下沿的次高位为 1，而上沿的次高位为 0。那么需要忽略次高位，同时对区间进行一次扩展。另外，需要记录一下我们忽略了一个次高位。因为，我们只是忽略它并没有抛弃它。当出现这种情况时，我们并不知道次高位的趋向。随着区间的变化，次高位可能趋向 1 也有可能趋向 0。所以现在先将它记录下来并忽略掉，等到区间变化趋向稳定后再输出。另外，这种情况可能会连续出现。所以需要记录忽略次数，输出时要按忽略次数输出。忽略次高位，同时对区间进行扩展的操作过程如下所示。

[01011010, 10010111]	→	[00110100, 10101111]
情况 3		

如果当前有未输出的次高位，又遇到上述三种情况，那么需要区别对待。当遇到情况 1 时，可以确定忽略的次高位趋向 0。此时可以输出 0，以及按忽略次数输出多个 1。然后扩展区间。当遇到情况 2 时，可以确定忽略的次高位趋向 1。此时可以输出 1，以及按忽略次数输出多个 0。然后扩展区间。当遇到情况 3 时，仍然无法确定忽略次高位的趋向。所以此时将忽略次数加 1，再次忽略次高位并扩展区间。

在实现的时候为了计算方便，可以设定一个区间的最大范围。最大范围的取值为 2 的次方数。初始时区间的大小就是最大范围。那么，根据前面的算法描述可以断定，区间的上下沿变化不会超过最大范围。我们将最大范围等分为 4 份，用虚线表示。将区间上下沿的变化情况用实线表示，并在虚线中标示出来。如图 3.1 所示。

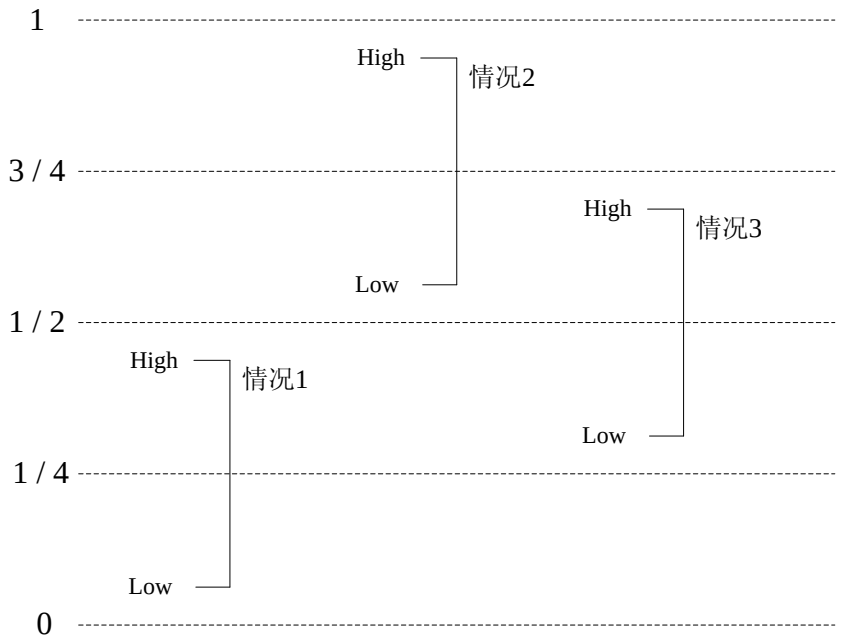

图 3.1 区间变化的三种情况

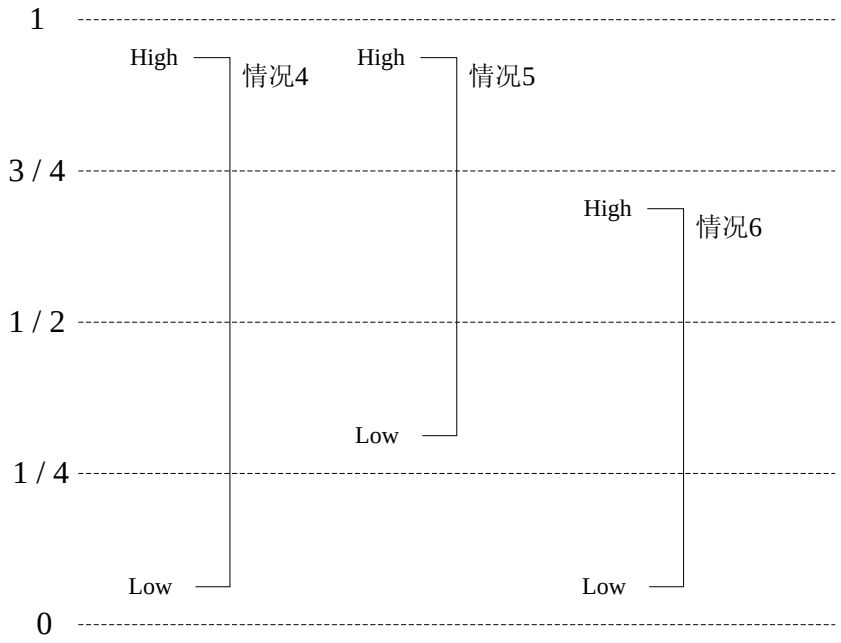

图 3.2 区间变化的另外三种情况

当区间的上下沿是情况 1 和情况 2 时。由于最大范围的取值为 2 的次方数，那么可以断定此时区间上下沿的最高位是相同的。情况 1 的区间位于下半区，上下沿的最高位为 0；情况 2 的区间位于上半区，上下沿的最高位为 1。类似的，情况 3 的区间位于中间区。区间的下沿位于下半区，其最高位为 0，次高位为 1；区间的上沿位于上半区，其最高位为 1，次高位为 0。上述这三种情况都需要进行正规化操作。除此之外，区间上下沿的变化还有另外

三种情况。这三种情况可以不进行正规化操作。直接从当前区间开始，进行下一轮的编码。这三种情况如图 3.2 所示。

综上所述，有三种情况需要进行正规化操作。对于有未输出次高位的情况，可以合并一起处理。对于情况 1 和情况 2，在输出最高位后可以检查忽略次数是否为 0。对于情况 3，可以直接对忽略次数加 1。另外，在解码的时候同样需要进行正规化操作。而且，解码时正规化的操作要和编码时正规化的操作相一致。唯一不同的是，解码时扩展区间需要输入一个位，以填补移出的空位。

4 实现

4.1 编码

本文中编写的实现代码主要基于 WNC 源代码，但同时进行了一些修改。修改的部分将在本节中进行介绍。现在先来看代码的实现。在实现代码的时候我们先要定义一些常量和变量。如下所示。

```
01  const
02      Top = Cardinal(1 shl 31); //最大范围
03      Half = Top shr 1; //最大范围的一半
04      Quar = Half shr 1; //最大范围的四分之一
05  var
06      Low, High, Range, Follow, Value : Cardinal;
```

代码 4.1 常量和变量的定义

前面已经论述过，区间计算时可以使用整数运算。所以这里使用 Cardinal 类型来保存数据。在代码 4.1 中，我们定义了一个区间的最大范围 Top，以及 Half 和 Quar。使用这些常量可以方便的判断出当前区间的位置。另外，Low 表示区间的下沿；High 表示区间的上沿；Range 表示区间的范围；Follow 表示忽略次数；Value 表示解码时输入的数据。编码程序的代码如下所示。

```
01  procedure ArithCoder_Encode(CumFreq, Freq, Total : Cardinal);
02  var
03      Check, Bit : Cardinal;
04      i : Integer;
05  begin
06      //区间计算
07      Range := Range div Total;
08      Inc(Low, CumFreq * Range);
09      High := Low + (Freq * Range) - 1;
10      //调整区间
11      while True do
12      begin
```

```
13              Check := High xor Low;
14              if (Check and Half) = 0 then
15              begin
16                  //区间位于上半区或下半区
17                  Bit := (High shr 30) and 1;
18                  OutputBit(Bit);
19                  if Follow <> 0 then
20                  begin
21                      if Bit = 0 then Bit := 1
22                      else                Bit := 0;
23                      for i := 1 to Follow do OutputBit(Bit);
24                      Follow := 0;
25                  end;
26              end
27              else if ((Check and Quar) <> 0) and
28                      ((High and Quar) = 0) then
29              begin
30                  //区间位于中间区
31                  Inc(Follow);
32                  Dec(High, Quar);
33                  Dec(Low, Quar);
34              end
35              else break;
36              //扩展区间
37              High := ((High shl 1) + 1) and (Top - 1);
38              Low := (Low shl 1) and (Top - 1);
39          end;
40          Range := High - Low + 1;
41          //完成
42      end;
```

代码 4.2 编码程序

代码 4.2 中 CumFreq 表示符号的累积频度；Freq 表示符号的频度；Total 表示所有符号的总计频度。OutputBit 函数用以输出一个位的数据。需要说明的是，在 WNC 源代码中编码部分与模型部分是合并在一起的。在本文的实现代码中将这两部分分离。编码部分专门负责区间的计算和正规化，而模型部分专门负责符号频度信息的统计。这样分离的好处是，可以使用不同的模型进行压缩。从而使编码部分可以和各种模型相配合，例如 PPM。另外一点，本文实现代码中区间的计算与 WNC 源代码中区间的计算并不相同。采用本文中的计算方法可以减小运算量。同时允许出现较大的总计频度。这样可以减少削减频度的次数。还有一点，本文实现正规化的代码。从表面上看与 WNC 源代码不一样，但实质上是一致的。

所有数据都编码完成后，还需要多输出一个位，以便最后一个符号的解码。输出的这个位是 1 还是 0 可以根据当前的区间下沿判断。另外，如果此时忽略次数不为 0，那么需要按忽略次数输出 1 或 0。收尾操作的代码如下所示。

```
01  procedure ArithCoder_FinishEncode;
02  var
03      i : Integer;
04  begin
05      //输出剩余编码
06      Inc(Follow);
07      if Low < Quar then
08      begin
09          OutputBit(0);
10          for i := 1 to Follow do OutputBit(1);
11      end
12      else
13      begin
14          OutputBit(1);
15          for i := 1 to Follow do OutputBit(0);
16      end;
17      //完成
18  end;
```

代码 4.3 编码的收尾操作

4.2 解码

由于将编码部分与模型部分相分离。在解码的时候需要两个函数来完成。第一个函数根据总计频度，从当前的区间中计算出估算频度。这个估算频度交给模型部分解码出符号。根据解码出的符号可以统计出符号的累积频度和频度。第二个函数根据符号的累积频度和频度重新计算区间。这样才能保证区间的变化与编码时相一致。第一个函数的代码如下。

```
01  function ArithCoder_DecodeTarget(Total : Cardinal): Cardinal;
02  begin
03      Range := Range div Total;
04      Result := (Value - Low) div Range;
05  end;
```

代码 4.4 解码程序 1

代码 4.4 中的函数返回估算频度，这个估算频度交给模型部分解码。另外，代码 4.4 第 3 行计算得到的 Range 并不是区间范围，而是一个临时的值。这个值将在第二个函数中用到，这里保存这个值可以防止重复运算。第二个函数的代码如下。

```
01  procedure ArithCoder_Decode(CumFreq, Freq : Cardinal);
02  var
03      Check : Cardinal;
04  begin
05      //区间计算
06      Inc(Low, CumFreq * Range);
07      High := Low + (Freq * Range) - 1;
08      //调整区间
09      while True do
10      begin
11          Check := High xor Low;
12          if (Check and Half) = 0 then
13          begin
14              //区间位于上半区或下半区
15          end
16          else if ((Check and Quar) <> 0) and
17                  ((High and Quar) = 0) then
18          begin
19              //区间位于中间区
20              Dec(High, Quar);
21              Dec(Low, Quar);
22              Dec(Value, Quar);
23          end
24          else break;
25          //扩展区间
26          High := ((High shl 1) + 1) and (Top - 1);
27          Low := (Low shl 1) and (Top - 1);
28          Value := ((Value shl 1) or InputBit) and (Top - 1);
29      end;
30      Range := High - Low + 1;
31      //完成
32  end;
```

代码 4.5 解码程序 2

代码 4.5 中的函数主要对区间进行重新计算和调整，同时还会输入数据。InputBit 是一个输入函数，每次调用时会返回一个位的数据。观察代码 4.5 中的代码可以发现。解码时进行的正规化操作，每次左移时最高位就被抛弃了。对于 Value 来说左移后需要输入一个位的数据填补到最低位。

4.3 统计模型

除了编码部分还需要模型部分上述代码才可以运行。模型部分需要使用一个数据结构来

记录符号的累积频度或频度。编码前需要查找一个符号的累积频度和频度。编码后需要更新一个符号的频度。另外，由于精度的限制。每当总计频度达到一个限定值的时候，需要对所有符号的频度减半。可以看出，查询和维护模型使用的数据结构需要占用大量的耗时。有研究指出[3]，当使用自适应模型时，模型使用的数据结构和算法对于压缩算法的整体性能有着决定性的作用。

在 WNC 源代码中使用了一个 move-to-front 的算法来实现模型。如果符号容量为 N，那么这个算法需要 4 个 N 个元素的数组来完成算法。当然这并不是一个最优秀的算法。Peter M. Fenwick 在 1994 年提出了一个新的数据结构：二进制索引树（Binary Indexed Tree，以下简称“BIT 结构”）[4]。BIT 结构只需要 1 个 N 个元素的数组就可以完成算法。目前，在统计累计频度的数据结构中，BIT 结构是速度最快的。同时它的内存占用也是最低的。本文中算术编码程序的模型部分，就是使用 BIT 结构来实现的。

由于本文主要讨论算术编码的实现，所以对于 BIT 结构的实现这里不做介绍。请感兴趣的读者自行查阅相关的资料。

5 分析

算术编码算法是所有基于统计模型的压缩算法中压缩率最高的一种算法。与哈夫曼编码相比，算术编码突破了以变长编码替换符号的瓶颈。算术编码使用一个小数来表示整个数据。并且根据符号的概率来调整小数的位数。这使得算术编码可以无限的接近数据的熵极限。另外由于算术编码的特点，它可以很好的与高阶模型相配合。这使得算术编码的用途越来越广泛。

算术编码在实现时也有许多缺点。首先一点，就是它的速度较慢。算术编码虽然可以获得比哈夫曼编码更高的压缩率。但是，算术编码的速度却比哈夫曼编码慢很多。另外一点，经过算术编码后的压缩数据无法进行随机访问。哈夫曼编码是使用变长编码替换符号的方法压缩数据。只要建立相关的索引表，就可以从压缩数据的任意位置开始解码符号。算术编码解码一个符号需要依赖以前解码符号的统计信息。所以使用算术编码解码一份压缩数据时，只能从压缩数据的最开头开始。这限制了算术编码的灵活性。

另外，许多算术编码的实现方法受美国专利的保护。如果在软件中使用这些算术编码的实现方法，需要得到相关的授权。一些实现方法的授权是免费的，而另一些则需要收取一定的授权费用。这些授权协议对于商业软件来说是可行的，但对于自由软件和开源软件项目来说它是不可行的。一个著名的例子就是压缩软件 bzip2。由于考虑到算术编码的版权限制，bzip2 放弃了算术编码的使用，转而使用哈夫曼编码。正是由于算术编码的版权限制，许多研究人员已经开始放弃对算术编码的研究。进而将目光转向了与算术编码本质上相同的区间编码。

6 结束语

算术编码由于其更高的压缩率，以及更适用于高阶模型，已经被大量的使用。本文分析了算术编码的理论基础，着重介绍 WNC 算法的实现方式。详细讨论了算术编码原理、正规化操作、WNC 算法代码实现等技术。并给出了一个切实可行的应用程序。希望本文能够对压缩算法的研究人员有所帮助。

参考文献

[1] C. E. Shannon, A Mathematical Theory of Communication, Bell System Technical Journal, 1948, Vol 27, 379-423, 623-656.

[2] Ian H. Witten, Radford M. Neal, John G. Cleary, Arithmetic Coding for Data Compression, Communications of the ACM, 1987, 30(6), 520-541.

[3] Alistair Moffat, Linear Time Adaptive Arithmetic Coding, IEEE Transactions on Information Theory, 1990, 36(2), 401-406.

[4] Peter M. Fenwick, A New Data Structure for Cumulative Frequency Tables, Software-Practice and Experience, 1994, 24(3), 327-336.

附录

本论文的配套程序没有单独提供。我在编程论坛上发表过另一篇论文《区间编码算法的分析与实现》（以下简称“《区》”），页面地址：http://programbbs.com/bbs/view12-29351-1.htm 。在论文《区》的配套程序中有一个 ArithmeticCoding.pas 文件。该文件实现的就是本论文论述的算术编码算法。所以论文《区》的配套程序也可以作为本论文的配套程序使用。相关的项目说明请参考论文《区》中的项目说明。

本论文的第 4.3 节并没有对 BIT 结构进行详细的论述。但在论文《区》有进行详细的介绍。在论文《区》中论述了 BIT 结构的理论基础、介绍了频度信息的统计和更新操作。并且给出了相关操作的源代码。这部分的内容请参考论文《区》中的第 5 节。

另外，在论文《区》中有哈夫曼编码、区间编码和算术编码的比较测试。其测试结果对于本论文也同样适用。所以本论文中没有给出算术编码的性能测试。相关的测试结果请参考论文《区》中的第 7 节。

叶叶

2010 年 9 月 10 日