《信息论》课程小组大作业

胡家熙　赵文滔　周韧

目录

1 摘要

这篇论文是对LZ77编码、deflate算法、哈夫曼编码、香农编码等编码方式的学习，总结了这些算法的原理，分析了时间复杂度和压缩率，提出了关于多次压缩的效率的问题并进行了相关实验，最后结合所学知识对实验结果进行了分析和解释。基于对哈夫曼算法和LZ77算法的改进型lzss算法的理解，我们还制作了相关的压缩软件。

【关键词】LZ77，deflate，哈夫曼，香农码，复杂度，二次压缩效率

2 引言

数据压缩无处不在，但是在这之前我们从来没有去了解过它，zip等压缩工具就像熟悉而陌生的朋友。这次信息论的学习使我们接触到了数据压缩的基本原理并对数据压缩算法充满好奇心，于是我们在小组项目阶段学习了一些数据压缩算法，分析了它们的效率。我们提出了二次压缩可以取得多高的效率的问题，进行了相关实验，并给出了分析和结论。

3 编码方法

3.1 哈夫曼编码

3.1.1 简介

哈夫曼编码(Huffman Coding)是一种编码方式，是一种用于无损数据压

缩的熵编码（权编码）算法，是可变字长编码(VLC)的一种。美国计算机科学家大卫·霍夫曼于1952年提出一种编码方法，该方法完全依据字符出现概率来构造异字头的平均长度最短的码字，有时称之为最佳编码，一般就叫做哈夫曼编码。

3.1.2 基本原理

首先，哈夫曼编码的编码思想是：根据字符出现的概率大小进行编码，出现概率高的字符使用较短的编码，出现概率低的字符使用较长的编码，这样可以构造出平均长度最短的码字。例如，考虑一个随机变量X，其取值空间为1，2，3，4，5，对应的概率分别是0.25，0.25，0.2，0.15和0.15.为获得X的一个最优二元码，需将最长的码字分配给字符4和5。这两个码字长度必定相等，否则若将这两个码字中较长的码字的最后一位剔除，仍可得到一个前缀码，但此时期望长度变短了。一般地，我们可以将该编码构造成为其中的两个最长码字仅差最后一位有所不同，对于这样的编码，可将字符4和5组合成单个信源字符，其相应的概率差值为0.30。按此思路继续下去，将两个最小概率的字符组合成一个字符，直至仅剩下一个字符为止，然后对字符进行码字分配，最终我们得到如下的表格：

码字长度	码　字	X	概　率
2	01	1	0.25 0.3 0.45 0.55 1
2	10	2	0.25 0.25 0.3 0.45
2	11	3	0.2 0.25 0.25
3	000	4	0.15 0.2
3	001	5	0.15

上述编码的期望长度为2.3比特。

然后考虑上例中随机变量的三元码。现在将三个最小概率的字符组合成一个超字符，得到如下的表格：

码　字	X	概　率
1	1	0.25 0.5 1
2	2	0.25 0.25
00	3	0.2 0.25
01	4	0.15
02	5	0.15

此时编码的期望长度为1.5铁特。

另外，如果D大于等于3，信源字符数目可能不充足，以至于不能每次总可以将D个字符组合起来。在此情形下，可添加虚拟字符并将其放置在原字符集的最后面。虚拟字符的概率为0且插入后可填满一棵树，由于在每

一次简化过程中，字符数均减少D-1个，而要求字符的总数是1+k（D-1），其中k为树的深度。因而，需要添加足够多的虚拟字符，使字符总数恰好为1+k（D-1）。例如：

码　字	X	概　率
1	1	0.25 0.25 0.5 1.0
2	2	0.25 0.25 0.25
01	3	0.2 0.2
02	4	0.1 0.2
000	5	0.1 0.1
001	6	0.1
002	虚拟符	0.0

此时编码的期望长度为1.7铁特。[1]

3.1.3　复杂度分析

哈夫曼编码是通过二叉树的形式构造表示的，其中构造出的二叉树一定是一颗满二叉树。同时，哈夫曼编码使用优先级队列管理结点序列，优先级队列使用最小化堆来维护。

在初始化堆时，时间复杂度是O(n)；而排序重建堆的时间复杂度为O(nlogn)，整个哈夫曼算法的时间复杂度也是O(nlogn)，哈夫曼算法是一种比较高效的压缩编码方式。

3.2　香农编码、香农-法诺编码与香农-法诺-伊利亚斯编码

3.2.1　简介

1.香农编码　香农(Shannon)编码是一种常见的可变字长编码，与哈夫曼编码相似。香农编码属于不等长编码，通常将经常出现的消息变成短码，不经常出现的消息编成长码，从而提高通信效率。

2.香农-法诺编码　在数据压缩的领域里，香农-范诺编码是一种基于一组符号集及其出现的或然率（估量或测量所得），从而构建前缀码的技术。这项技术是香农于1948年，在他介绍信息理论的文章“通信数学理论”中被提出的。这个方法归功于范诺，他在不久以后以技术报告发布了它。香农-范诺编码不应该与香农编码混淆，后者的编码方法用于证明Shannon’s noiseless coding theorem，或与Shannon–Fano–Elias coding一起，被看做算术编码的先驱。

3.香农-法诺-伊莱亚斯编码 在消息理论中，香农–法诺–伊莱亚斯码是算术编码的先导，其几率被用于决定码字。

3.2.2 基本原理

1.香农编码 首先，香农编码的理论基础是符号的码字长度完全由该符号出现的概率来决定，其步骤如下：

- (1)将信源符号按概率从大到小顺序排列，为方便起见，令
$$P(a_1)\geq P(a_2)\geq ...\geq P(a_n)$$
- (2)按$-\log P(a_i)\leq l_i\leq -\log P(a_i)+1$计算第i个符号对应的码字的码长(取整)。
- (3)计算第i个符号的累加概率$P_i=P(a_1)+P(a_2)+...+P(a_{i-1})$。
- (4)将累加概率变换成二进制小数，取小数点后l_i位数作为第i个符号的码字。

例如对如下信源编码：

$$\begin{bmatrix} S \\ P(S_i) \end{bmatrix} = \begin{bmatrix} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 \\ 0.20 & 0.19 & 0.18 & 0.17 & 0.15 & 0.10 & 0.01 \end{bmatrix}$$

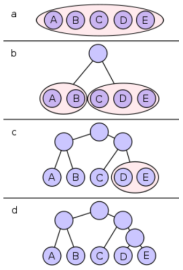
信源符号 s_i	符号概率 $p(s_i)$	累加概率 P_i	$-\log p(s_i)$	码长 l_i	码字
s_1	0.20	0	2.34	3	000
s_2	0.19	0.2	2.41	3	001
s_3	0.18	0.39	2.48	3	011
s_4	0.17	0.57	2.56	3	100
s_5	0.15	0.74	2.74	3	101
s_6	0.10	0.89	3.34	4	1110
s_7	0.01	0.99	6.66	7	1111110

其香农编码如表1所示，以i=4为例，$-\log 0.17\leq l_4<-\log 0.17+1$,即$2.56\leq l_4<3.56$，因此$l_4=3$。累加概率$P_i=0.57$，变成二进制数，为0.1001...。转换的方法为：用P_i乘以2，如果整数部分有进位，则小数点后第一位为1，否则为0，将其小数部分再做同样的处理，得到小数点后的第二位，依此类推，直到得到了满足要求的位数，或者没有小数部分了为止。 [2]

2.香农-法诺编码 香农-法诺的树是根据旨在定义一个有效的代码表的规范而建立的。实际的算法如下：

- (1)对于一个给定的符号列表，制定了概率相应的列表或频率计数，使每个符号的相对发生频率是已知。
- (2) 排序根据频率的符号列表，最常出现的符号在左边，最少出现的符号在右边。
- (3)清单分为两部分，使左边部分的总频率和尽可能接近右边部分的总频率和。
- (4)该列表的左半边分配二进制数字0，右半边是分配的数字1。这意味着，在第一半符号代都是将所有从0开始，第二半的代码都从1开始。
- (5)对左、右半部分递归应用步骤3和4，细分群体，并添加位的代码，直到每个符号已成为一个相应的代码树的叶。

下图为香农-法诺算法的一个实例（假设从左到右已经按照符号出现的频率排序完毕）[3]：



3.香农-法诺-伊莱亚斯编码　该算法的描述如下：

- (1)给定一离散随机变数 X，令 p(x) 为 X=x 发生之几率。
- (2) 定义

$$\bar{F}(x) = \sum_{x_i<x} p(x_i) + 0.5p(x)$$

- (3)对每个 X 中的 x，令Z为$\bar{F}$(x)之二次展开。
- (4)令 x 之编码长度L(x)=

$$\left\lceil \log_2 \frac{1}{p(x)} \right\rceil + 1$$

- (5)选定 x 之编码，code(x)为L(x) 在 Z 之小数点后之第一个最高有效位。

具体的例子如下：

令 X = {A, B, C, D}，其发生几率分别为 p = {1/3, 1/4, 1/6, 1/4}。

对于 A

$\bar{F}(A) = \frac{1}{2}p(A) = \frac{1}{2} \cdot \frac{1}{3} = 0.1666...$

在二进制中，Z(A) = 0.**001**0101010...

L(A) = $\left\lceil \log_2 \frac{1}{\frac{1}{3}} \right\rceil + 1 = \mathbf{3}$

code(A) 为 001

对于 B

$\bar{F}(B) = p(A) + \frac{1}{2}p(B) = \frac{1}{3} + \frac{1}{2} \cdot \frac{1}{4} = 0.4583333...$

在二进制中，Z(B) = 0.**011**10101010101...

L(B) = $\left\lceil \log_2 \frac{1}{\frac{1}{4}} \right\rceil + 1 = \mathbf{3}$

code(B) 为 011

3.2.3 复杂度分析

1.香农编码 根据香农编码的基本原理，我们分析了一下它的时间复杂度。首先，第(1)步将信源符号按照概率从大到小顺序排列，如果用堆排序等较快的排序算法，时间复杂度可以达到O(nlogn)。第二步计算每个符号对应的码字的码长，我们知道只用计算n次，所以时间复杂度是O(n)。到了第三步，如果我们用一般的方法，分别求第一个信源符号到第n个信源符号的累加概率，可能会导致较高的时间复杂度，而如果我们牺牲空间复杂度，用一个列表来把每次求出来的累加概率保存下来，我们就可以避免重复计算(我觉得这和实现斐波那契数列的思想是相似的)，从而提高时间性能，这样一来时间复杂度就不会超过O(nlogn)。第(4)步需要将十进制小数变换成二进制小数，这个步骤的复杂度与所有信源符号的最大码长l_{max}有关，因为我们需要执行l_{max}个循环来完成变换。

综上所述，香农编码的时间复杂度为O(max(nlogn,l_{max})),如果其中有信源编码的概率无穷小，或者足够小，那么l_{max}的值有可能非常非常大，这就会造成整个算法的时间复杂度很高，性能十分低下。所以，我们认为香农编码最好用于所有信源编码的概率值都分布在一个区间(概率不能太低)的情况，而且香农编码并不是一个十分高效的算法。

2.香农-法诺编码 同样地，根据香农-法诺编码的基本原理，我们也分析了它的时间复杂度。首先，第(1)步已经给出，不用我们计算。然后，第(2)步同样需要排序，如果使用时间复杂度较低的堆排序等算法，可以实现O(nlogn)的复杂度。第(3)步要把清单划分为两部分，而且要使两部分的频率和尽可能接近。这一步里我们可以用一个列表在每次分成两部分后将两部分的总概率值保存下来，这样每次分割我们只需要从第一个元素开始累加，直到累加概率值接近总概率值的一半。这样的复杂度可以达到O(n)。所以，香农-法诺编码的整体时间复杂度可以控制在O(nlogn)内。

3.香农-法诺-伊莱亚斯编码 进而，根据香农-法诺-伊莱亚斯编码，我们也可以分析出它的复杂度来。首先，第(1)步已经给出，不需要我们计算。然后，第(2)计算累积分布函数，利用香农编码部分的结论，我们可以知道它的复杂度不会超过O(nlogn)。第(3)步将十进制小数转化成二进制小数，与香农编码部分相似，这一步骤的复杂度为l_{max}。第(4)步，计算编码长度，这一步骤的时间复杂度显然是O(n)。

综上所述，香农编码的时间复杂度为$O(\max(n\log n, l_{max}))$。同样地，如果其中有信源编码的概率无穷小，或者足够小，那么l_{max}的值有可能非常非常大，这就会造成整个算法的时间复杂度很高，性能十分低下。相比香农编码，此编码在求最优码方面有了一定的改进，但仍旧有缺陷。

3.2.4 三种编码方式总结

香农、香农-法诺、香农-法诺-伊莱亚斯编码是按照时间顺序依次出现的，现在我们来对他们进行一个横向比较。首先，香农编码作为最早被发明的编码方式，它提供了一种思路来构建编码，是一个开创性的行为。香农编码的效率不高，实用性不大，但对其他编码方法有很好的理论指导意义。其次，香农-法诺-伊莱亚斯编码更像是对香农编码的一种优化，提出了$\bar{F}(x)$来优化编码的过程，然而在某些特定情况下，它仍然有一定的缺陷。另外，香农-范诺编码则更趋于提出了一种新的解决方案，它其中的一些操作并没有香农编码的显著特征。

总而言之，这三种编码方式的出现都具有特定的意义，都给人们提供过便利。然而，我们认为，在现在这种社会环境下，他们均不如上文提到

的哈夫曼编码，其中一些可能还面临着被彻底淘汰的窘境。

3.3 LZ77编码

3.3.1 简介

LZ77编码是一种基于字典的、“滑动窗”的无损压缩算法，广泛应用于通信、计算机文件存档等方面。

3.3.2 基本原理

LZ77算法使用”滑动窗口”的方法，来进行字符串的匹配。假设有字符串$x_1x_2...x_n$,W是窗口的长度，$x_1...x_{i-1}$已经压缩完成，接下来就是要寻找最大的k，使存在j$\in [i-1-W, i-1]\cap Z$,满足$x_{j+l}=x_{i+l}$，$1\in[0,k)\cap Z$,$x_i,x_{i+1}...x_{i+k-1}$就可以压缩为(P,L,x_{i+k}),P是匹配的起始位置，L是匹配的长度，x_{i+k}是匹配结束后的下一个字符。接下来从x_{i+k+1}开始重复上述操作。如果x_i没有在之前的W个字符中找到匹配，就输出(0,0,x_i)[5]。举个例子，压缩aabcbbabc，得到的结果将是(0,0,a), (1,1,b), (0,0,c), (2,1,b), (5,2,c)。

3.3.3 复杂度分析

设置有滑动窗口即向后缓冲的字符串长度W，可以提高算法的时间效率，仅仅牺牲一点压缩率。设置了W，压缩长为n的字符串，下面计算时间复杂度：设最终将字符串分成了k段，第i段长度为l_i,那么

$$\sum_{i=1}^{k} l_i = n$$

为了匹配出第i段最坏需要扫描W次，每次扫描试图匹配的长度接近l_i,复杂度为

$$\sum_{i=1}^{k} l_i W = Wn$$

，级别为O(n)。

3.3.4 压缩率

最坏情况下经过压缩的文件比原文件还大，比如ab经过压缩之后变为(0,0,a), (0,0,b)。最好的压缩效率会非常理想，设压缩的字符串为$x_1 x_2 ... x_n$，当他们全都相等的时候，压缩为(0,0,x_1),(1,n-2,x_n)，如果n比较大，压缩效果会非常明显，但这种情况几乎不会出现。

3.3.5 拓展

Storer和Szymanski对LZ77提出了改进，将较短的匹配字符串中的每个字符以(F,C)的形式记录下来按会更节省空间[6]。F是标记位，占1比特，F=0时表示直接记录字符，C是字符本身。假设滑动窗口的长度为W，最长匹配长度限制为M，对于匹配长度为m的字符串，当1+logM+logW>m(1+8)时，将字符串中的每个字符采用(F,C)的形式记录下来，但是这并没有显著提升LZ77的压缩率，一个突破性的进展是deflate算法（zip压缩）的发明。

3.4 Deflate算法

3.4.1 简介

DEFLATE是同时使用了LZ77算法与哈夫曼编码（Huffman Coding）的一个无损数据压缩算法。

3.4.2 基本原理

Phil Katz（以下简称PK）提出的deflate算法在LZ77的基础上使用了哈夫曼编码、游程编码[7]。首先按照LZ77的思想用(length，distance)（稍作了调整）来压缩可以压缩的部分，由于重复字符串如果长度为2的话起不到压缩效果，因此设定最小的匹配长度为3。PK将滑动窗口的大小设置为32KB，最大匹配长度设为258，并且更改了匹配规则，当遇到第一次匹配时（离被压缩字符最近的匹配），就不再继续寻找匹配，以提高压缩速度。经过这种改进的LZ77算法压缩之后，所有的字符串都被表示为literal（未压缩）或者length+distance的形式。接下来PK进一步压缩distance, length和literal。

PK使用了两棵哈夫曼树，一棵对distance编码，另一棵对length、literal进行编码，为了压缩码表，PK使用Deflate树（右倾的哈夫曼树），只需要把码字长度按对应的distance的大小排序，就可以唯一确定码表。

Distance对应的哈夫曼树：为了减小树的深度，PK把1到32768划分成了30个区间，考虑到越小的值出现的概率越大，采用非等距划分，具体见表格1(下面分别为表格1、表格2)

	Extra			Extra			Extra	
Code	bits	Distance	Code	bits	Distance	Code	bits	Distance
0	0	1	10	4	33–48	20	9	1025–1536
1	0	2	11	4	49–64	21	9	1537–2048
2	0	3	12	5	65–96	22	10	2049–3072
3	0	4	13	5	97–128	23	10	3073–4096
4	1	5,6	14	6	129–192	24	11	4097–6144
5	1	7,8	15	6	193–256	25	11	6145–8192
6	2	9–12	16	7	257–384	26	12	8193–12288
7	2	13–16	17	7	385–512	27	12	12289–16384
8	3	17–24	18	8	513–768	28	13	16385–24576
9	3	25–32	19	8	769–1024	29	13	24577–32768

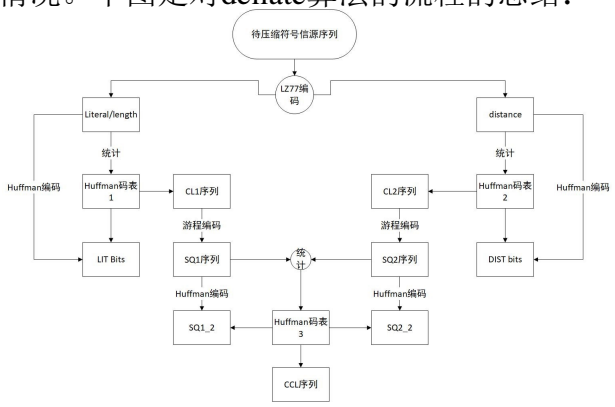
	Extra			Extra			Extra	
Code	bits	Lengths	Code	bits	Lengths	Code	bits	Lengths
257	0	3	267	1	15,16	277	4	67–82
258	0	4	268	1	17,18	278	4	83–98
259	0	5	269	2	19–22	279	4	99–114
260	0	6	270	2	23–26	280	4	115–130
261	0	7	271	2	27–30	281	5	131–162
262	0	8	272	2	31–34	282	5	163–194
263	0	9	273	3	35–42	283	5	195–226
264	0	10	274	3	43–50	284	5	227–257
265	1	11,12	275	3	51–58	285	0	258
266	1	13,14	276	3	59–66			

为了区分一个区间中的不同距离，需要在这个区间的统一码字后面加上一些比特，加的比特数为表中的bits，加的方式也是固定的，比如区间6,9到12依次加00,01,10,11。根据每个区间出现的频率，给区间用哈夫曼编码。literal、length对应的哈夫曼树：把literal、length用同一棵哈夫曼树编码，就可以很好地区分literal和length + distance的形式。literal用0-255表示，256是结束标志，解码之后256表示解码结束。从257开始表示length，257对应的length是3，PK把length划分为29个区间，见表格2。

把distance编码之后，把每个区间对应的码字的长度记录下来，没有出现过的区间的码字长度为0，然后把码字长度按区间的顺序排列，得到CL1，CL1之后用来表示给distance编码的哈夫曼树，distance编码之后成为DIST码流，同理，按表格2中的code从0到285的顺序排列第二棵树的码字长度得到CL2，literal、distance编码之后得到DIST码流。PK分析认为最长的编码不会超过15位，因此CL1, CL2中的数字的值在0-15区间内。进一步压缩CL1, CL2。首先去掉最后的连续出现的0，然后使用游程编码压缩，游程压缩的标记是16,17,18，16是对非零数字的压缩的标记，16后会跟着00,01,01或者11（这四个数是2位2进制），表示16前面的数字又紧接着连续出现3,4,5,6次，比如44444可以表示为4, 16, 01。连续出现大于7次的话，就要分成多段压缩，例如连续8个4压缩为4, 16, 11, 4，连续11个4压缩为4,16,11,16,01。由于0出现频率很高，17，18专门用来标记0的连续出现，这样就省得在17,18前再写0, 17后面跟3比特000-111分别记录3-10个连续的0, 18后面跟7比特0,000,000到1,111,111分别表示11-138个连续的0。

对CL1, CL2分别压缩之后得到SQ1, SQ2两个序列，序列中数字的大小在0-18间，PK又统计0-18出现的次数（两个序列联合起来统计），然后对0-18采用哈夫曼编码，得到哈夫曼表3，使用的依然是Deflate树，因此又可以用码字的长度来记录哈夫曼表3，PK分析认为码字的长度范围是0-7，于是只需要3比特记录每个码字的长度，把0-18对应的码字长度依次列出，再交换顺序，把0经常出现的位置放到序列的最后。交换前是按照0到18的码字长度排列的，交换后的顺序是：16,17,18,0,8,7,9,6,10,5,11,4,12,3,13,2,14,1,15，去掉序列末尾的连续0，得到CCL。使用哈夫曼表3对SQ1, SQ2进行压缩，得到SQ1_2, SQ2_2。

经过deflate算法压缩后的文件可以表示为：Header, HLIT, HDIST, HCLEN, CCL, SQ1_2, SQ2_2, LIT编码流与DIST编码流。Header占3个比特，HLIT占5比特，记录CL1元素个数，CL1个数等于HLIT+257（因为至少有0-255总共256个literal，还有一个256表示解码结束，但length的个数不定）。HDIST占5比特，记录CL2元素个数，CL2个数等于HDIST+1。HCLEN占4比特，记录CCL个数，CCL个数等于HCLEN+4，因为PK认为CCL个数不会低于4个，即使对于整个文件只有1个字符的情况。下图是对deflate算法的流程的总结：



3.4.3 复杂度分析

使用LZ77对全文压缩的复杂度为O(n)，使用哈夫曼编码对literal, distance, length压缩的复杂度为O(nlogn)，使用游程编码对CL1, CL2进行压缩的复杂度为O(1), 使用哈夫曼编码对SQ1, SQ2压缩的复杂度为O(1)，因此ZIP编码的复杂度为O(nlogn)。

3.4.4 压缩率

由于ZIP设定的length的最大值是258KB，考虑到length、distance在哈夫曼之后的最短理论长度为1比特，于是长度为258KB的重复字符串理论上可以被压缩成2比特（未计算zip文件开头header, CL1_2, CL2_2等占用的空间），最大压缩率为1032:1。

3.4.5 思考与探索

关于压缩率我们进行了思考与探索，对已经被压缩的文件再进行压缩，文件会继续变小吗？首先我们做了一些实验来推测结论，以下是我们自己计算出的实验结果：

原始(KB)	类型	1 次压缩大小(KB)	2 次压缩大小(KB)
18088	MP4	12062	12044
6039	PDF	4746	4744
3532	GIF	3500	3502
3103	PPTX	3076	3077
9745	MP4	4577	4567
1167	DOCX	1157	1157
3855	PDF	3566	3568
5590	JPG	5590	5593

对大多数文件而言，第一次的压缩效果都比较明显，但是对几乎所有文件而言，第二次压缩都是无效的，部分文件甚至“反膨胀”，我们对此进行了思考。Deflate算法本身的压缩效率是不错的，因此如果一个文件中有不少重复出现的字符串，使用deflate算法就可以把这些重复出现的字符串压缩成比较随机的模式，因为如果压缩过后的文件中还包含大量重复的字符串，那么使用deflate算法必定可以继续对文件压缩，使文件进一步缩小。当文件中的字符出现非常随机时，不会再有大量重复出现的（长度大于2的）字符串，文件就很难再使用deflate算法进行压缩了。如果使用LZ77算法，由于匹配不到重复出现的字符串，原本大小为8比特的字符就会增加一个标记位变成9比特，当大量的字符从8比特变成9比特时，必然导致“反膨胀”。如果对一个字符出现很随机的文件使用哈夫曼编码，也是很有可能“反膨胀”的，根本原因是因为码表的存在。考虑极端情况下文件中每个字符都是不一样的，那么哈夫曼编码之后的文件中的码表的大小就会超过原文件的大小，压缩后的文件必然大于原文件。游程编码对字符随机出现的文件的处理会更加糟糕，因为游程编码可以看做是LZ77编

码的一种退化。Deflate算法是对上述三种算法的综合应用，这可以适当抑制反膨胀，但是并不能从本质上改变第二次压缩的效率，因此第二次使用deflate算法压缩之后，几乎所有的文件都没有明显减小，不少还反而增大了。

从压缩是有极限的角度来分析问题，也可以解释二次压缩效果不好的现象。假设每个字符经过压缩之后的编码长度为l_i，从Kraft不等式

$$\sum_{i=1}^{m} D^{-l_i} \leq 1$$

出发，结合拉格朗日算子我们可以推导出最佳编码的平均字符长度$L^* \geqslant H_D($X)这个式子告诉我们，越随机的文本越难被压缩，当我们把一个文本压缩成为近乎随机的字符序列之后，压缩空间就很小了，这就是压缩的极限。通过这次大作业的实践、思考，我还可以解释为什么要把压缩率跟熵建立联系：压缩率跟文本的有序性相关，越有序的文本压缩空间越大；熵正是度量文本有序性的标尺，因此压缩率和文本的熵之间可以建立起数量关系，压缩正是把文本有序向无序的转换。

为了进一步直观感受压缩的极限，考虑分别对2^n个n比特的文本进行压缩，由于压缩之后能够分别还原出原来的2^n个文本，这2^n个压缩文件必定两两不同，$2^1+...+2^{n-1}=2-2^n/(1-2)=2^n-2$,因此最优情况下，至少有$2^{n-1}$个压缩文件的长度大于等于n-1，当n比较大时，这些文件的压缩效果忽略不计，这些文件就达到了压缩的极限。运用前面的方法来分析这个现象，可以得知这些到达压缩极限的文件必定是文本出现比较随机的文件。

关于压缩的另一点思考是：既然哈夫曼是最佳的编码方式，那么为什么不连续使用两次哈夫曼编码，而要先用LZ77压缩，再使用哈夫曼编码呢？哈夫曼编码能够把文件本身压缩至最小，但这个“最小”没有考虑码表的大小。为了使接受者能够解压缩，必须在文件中写出码表，而算上码表之后，连续使用哈夫曼编码的压缩率就比不上先使用LZ77，再使用哈夫曼编码的压缩率了。

4 结论

这次项目我们对LZ77编码、deflate算法、哈夫曼编码、香农编码等编

码方式进行了学习，总结了原理、时间复杂度和压缩率。我们发现相比香农系列的三种编码，哈夫曼编码是更优的编码方式。我们还对二次压缩的效率展开了探究，发现如果对同一个文件连续两次使用LZ77、哈夫曼编码或者deflate算法，那么第二次的压缩效果微乎其微，甚至会让部分文件变大。我们分析认为第一次使用deflate算法压缩之后能让大多数文件从有序变得比较随机，文本的熵增大了，同时字符对应的编码长度在减小。根据压缩极限定理可知，文本的可压缩空间已经大大减小。从更实际的角度进行分析，我们发现LZ77、哈夫曼编码、游程编码在压缩由随机字符组成的文件的时候效率是不理想的，因而deflate算法也不能取得好的效果。

5 参考文献

1 Thomas M.Cover, Joe A.Thomas. 《Element of Information Theory》

2 https://baike.baidu.com/item/香农编码/22353186?fr=aladdin

3 https://blog.csdn.net/abcjennifer/article/details/8022445?ops_request_misc=&request_id=&biz_id=102&utm_term=香农-范诺编码&utm_medium=distribute.pc_search_result.none-task-blog-2 all sobaiduweb default-1-8022445

4 https://en.wikipedia.org/wiki/Shannon%E2%80%93Fano%E2%80%93Elias_coding

5 J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory. 23 (3): 337–343., May 1977.

6 J. A. Storer and T. G. Szymanski, "Data Compression via Textual Substitution," Journal of the ACM. 29 (4): 928–951, October 1982.

7 D. Salomon, Data Compression The Complete Reference, Springer-Verlag London Limited 2007.