压缩与熵编码

目录

一. 对下列符号进行 Huffman 编码，并计算压缩比.....1
二. 写出串“good night”之算术编码的编解码过程.....2
三. 对字符串“ababcbababaaaaaaa”进行手工 LZW 编解码.....3
四. 心得体会.....5

一. 对下列符号进行 Huffman 编码，并计算压缩比

符号	A	B	C	D	E	F
出现的次数	40	10	5	15	20	10

Huffman 编码的实现过程，就是不断地找出出现次数最小的符号，自底向上建树，然后深度遍历建好的树，左边编码为 0，右边编码为 1 即可。本人使用 Python 的优先队列来找出出现次数最小的符号，最后实现源码在 src/HuffmanEncoder.py 中。

运行程序，可得到结果

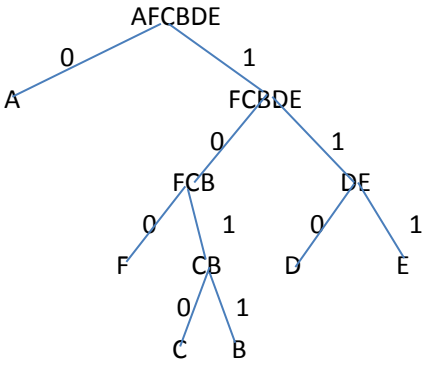
```
A  40 0
E  20 111
D  15 110
F  10 100
B  10 1011
C   5 1010
(3.0, 300.0, 235)
1.27659574468
```

整理为下表

符号	次数	编码	需位数
A	40	0	40
B	10	1011	40
C	5	1010	20
D	15	110	45
E	20	111	60
F	10	100	30
合计	100		235

平均码长为：235/100 = 2.35, 压缩比为 235 / (3 * 100) = 1.27660 : 1

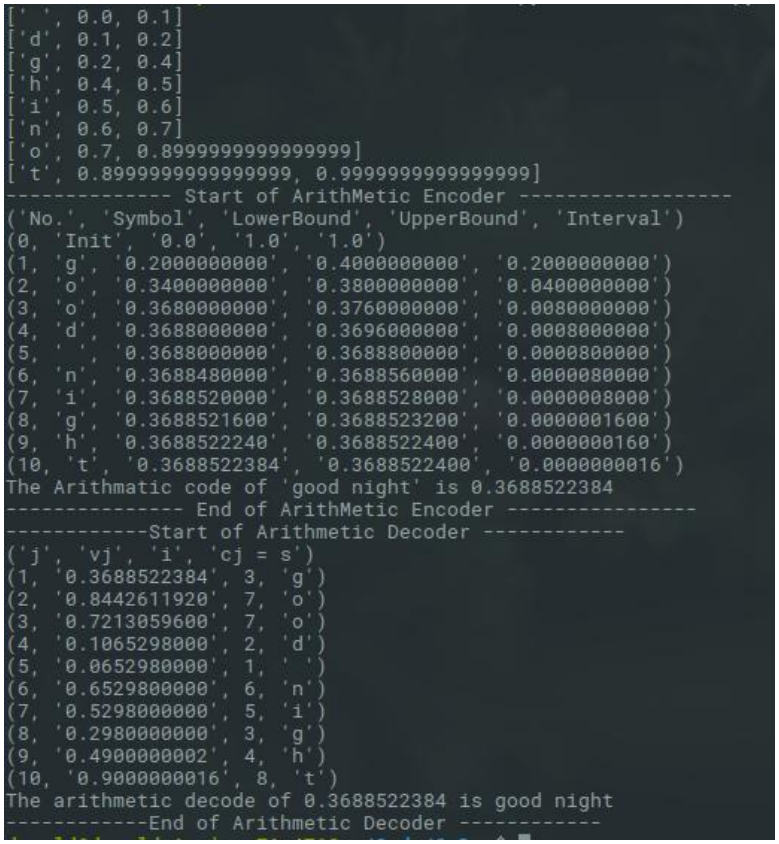
Huffman 编码树如下所示



二. 写出串“good night”之算术编码的编解码过程

本人依据课件中展示的过程，使用 Python 实现了算术编码的编解码，在输出中也做了过程的打印，源码在 src/ArithCoDec.py 中。

运行程序，可得到如下结果



（*注： 由于 Python 对于浮点数处理的机制，导致了上图中一些小数的结果，另外，算术编解码过程中，由于逐层递进的关系，对于计算精度的要求非常高，否则对于较长的字符串将无法顺利的编解码。在 Python 中，可以通过使用 bigfloat 等第三方随机精度的库来解决*）

编解码过程整理如下：

符号表

序号	符号	P	[x, y）
1		0.1	[0.0, 0.1）
2	d	0.1	[0.1, 0.2）
3	g	0.2	[0.2, 0.4）
4	h	0.1	[0.4, 0.5）
5	i	0.1	[0.5, 0.6）
6	n	0.1	[0.6, 0.7）
7	o	0.2	[0.7, 0.9）
8	t	0.1	[0.9, 1.0）

算术编码的编码过程

序号	符号	Li	Ri	Di
0	初始值	0.0	1.0	1.0
1	g	0.2000000000	0.4000000000	0.2000000000
2	o	0.3400000000	0.3800000000	0.0400000000

3	o	0.3680000000	0.3760000000	0.0080000000
4	d	0.3688000000	0.3696000000	0.0008000000
5		0.3688000000	0.3688800000	0.0000800000
6	n	0.3688480000	0.3688560000	0.0000080000
7	i	0.3688520000	0.3688528000	0.0000008000
8	g	0.3688521600	0.3688523200	0.0000001600
9	h	0.3688522240	0.3688522400	0.0000000160
10	t	0.3688522384	0.3688522400	0.0000000016

编码输出为 l_{10} = 0.3688522384

算数编码的解码过程表

J	Vj	I	Ci = s
1	0.3688522384	3	g
2	0.8442611920	7	o
3	0.7213059600	7	o
4	0.1065298000	2	d
5	0.0652980000	1	
6	0.6529800000	6	n
7	0.5298000000	5	i
8	0.2980000000	3	g
9	0.4900000002	4	h
10	0.9000000016	8	t

重构输入字符串为 “good night”

三. 对字符串“ababcbababaaaaaaa”进行手工 LZW 编解码

本人依据课件中展示的过程，使用 Python 实现了 LZW 编码的编解码，在输出中也做了过程的打印，源码在 src/LZWCoDec.py 中。

运行程序，可得到如下结果

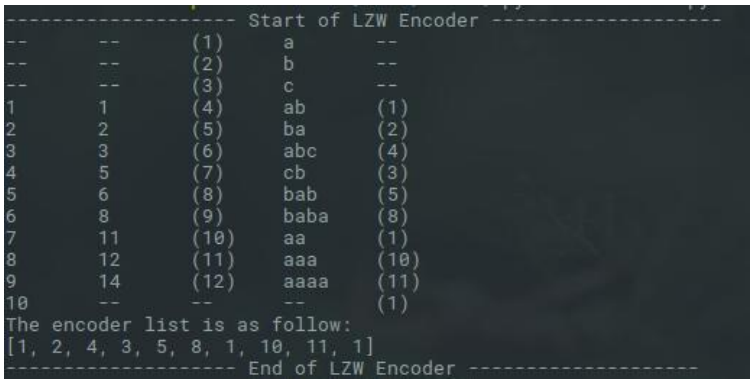


Figure 1 LZW 编码过程

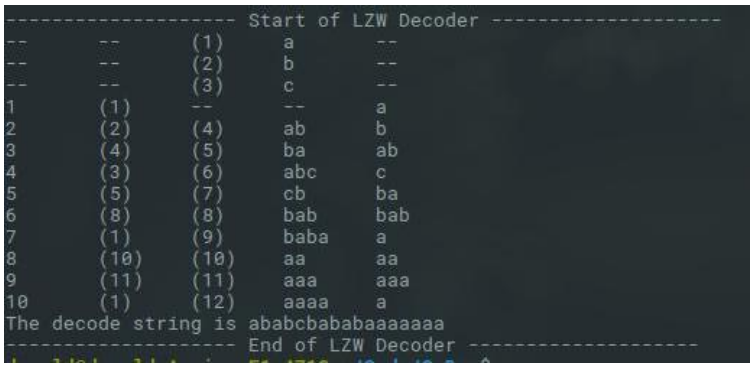


Figure 2 LZW 解码过程

编解码过程整理如下：

编码字符串

位置	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
字符	a	b	a	b	c	b	a	b	a	b	a	a	a	a	a	a	a

LZW 的编码过程

步骤	位置	词典 next	词典 wc	输出 w
		（1）	a	
		(2)	b	
		(3)	c	
1	1	(4）	ab	(1)
2	2	（5)	ba	(2)
3	3	(6)	abc	(4)
4	5	(7)	cb	(3)
5	6	(8)	bab	(5)
6	8	(9)	baba	(8)
7	11	(10)	aa	(1)
8	12	(11)	aaa	(10)
9	14	(12)	aaaa	(11)
10	--	--	--	(1)

LZW 的译码过程

步骤	码字 new	词典 next	词典 Old+newStr[0]	输出 newStr
		(1）	a	
		（2）	b	
		（3）	c	
1	(1）	--	--	a
2	（2）	(4)	ab	b
3	（4）	(5)	ba	ab
4	（3）	(6)	abc	c

5	（5）	(7)	cb	ba
6	（8）	(8)	bab	bab
7	（1）	(9)	baba	a
8	（10）	(10)	aa	aa
9	（11）	(11)	aaa	aaa
10	（1）	(12)	aaaa	a

四． 心得体会

本章作业总体而言较为简单，但却比较有意思。编码本身就是为了更好地服务于计算机的存储和计算而出现的，这也是本人将本章中所有的题目都使用编程来实现的原因。考虑到快速实现，本人使用了 Python，另外，由于代码本身没有经过严格的程序化测试，所以当数据量较大或者出现了本人没有充分考虑到的边界条件输入时，程序的结果可能会出现错误。但总体而言，算法的整体框架都是完整实现的。

Huffman 编码主要是主要建树的过程，当树建好之后，递归遍历生成的树就可以得出编码结果了。由于此处只是考虑叶子节点，所以并没有前后中序之分。另外，Huffman 基于贪心算法来编码，压缩率确实很高。但是在实际传输中应该需要加入恰当的纠错技术，否则只要中间有某一个编码发生错误，则基本上所有数据就都没用了。

算术编码思路也是比较简单，但是，在实际实现中，要非常注重精度的问题。完善的算术编码对精度要求特别高，可以考虑使用字符串代替浮点数。

LZW 编码也是基于贪心算法，充分理解清楚算法之后，实现就不难了。完整的 LZW 编码还会有清空串表的操作，但由于本题目数据量很小，本程序中暂时没有实现这一步。

以上。

2016-10-13