EE669 HW2 Report

Jie Huang

2229747350

jhuang59@usc.edu

Problem 1: Written question

Input	State	Qe	A	C	Mps	output
	12	1EDF	0X10000	0X0000	0	
1	11	2516	F6F8	71088	0	100
0	11	2516	D1E2	71088	0	
0	11	2516	ACCC	71088	0	
1	10	299A	2516	1E60F8	0	101
0	11	2516	A668	7971D6	0	100
0	11	2516	8152	7971D6	0	
0	12	1EDF	B878	F2E3AC	0	0
1	11	2516	F7F8	79BF298	0	1100
1	10	299A	9458	1E73A640	0	101
0	11	2516	D57C	3CE74C80	0	0

Encoding output:1001011000110010101

Problem 2:

(a)BAC:

(1):Please find the below result.

(2):

Table1

Directly convert one byte to 8 bits		
File	Compressed size	Compression ratios
binary.dat	25	0.379
text.dat	67	1.01
audio.dat	67	1.01
image.dat	64	0.97

Table2

Huffman encoding		
Original File	Compressed size	Compression ratios
binary.dat	8	0.121
text.dat	37	0.560
audio.dat	27	0.409
image.dat	39	0.590

Table3

Bit-Plane encoding		
Original File	Compressed size	Compression ratios
binary.dat	9	0.121
text.dat	46	0.697
audio.dat	63	0.954

image.dat	60	0.909

In table 1, we can see the compression ratio of text and audio file is larger than 1. This result is reasonable as all symbols are converted to 8 bits stream. 8 bit binary stream is unnecessary for most symbols. For example,the binary stream of character “1” is 1011001. The “1” was converted to integer base on its ASCII value. Then, integer 49 was converted to binary 1011001. In this case, we need only 7 bits instead of 8 bits.
In table 2, we see that the compression ratio is smaller than that in table 1. This difference is reasonable as huffman coding gave the most frequent symbol the shortest codeword. The average length of codeword is shorter than that in table 1.
I table 3, we can see that bit plane coding perform less well than huffman coding. This result is reasonable because bit plane encoding still need to encode symbols into 8-bit stream. While huffman encoding can save some bits by encode frequent symbols into shorter bits stream. Bit plane encoding can achieve a better result in lossy compression.

(3):

In binary.dat, about 97% of symbols are “1”. The Qe is 0.97*1.5. In this case, we need to renormalize A and C frequently due to small A-Qe value. The more times we renormalize the more output we have. We would have a large compressed file. However, if we initial “0” as LPS. Qe-A would not be small any more and we are expected to have a small compressed file.

(4):

Unary coding. Below is an example of unary:

ASC-Ⅱ	Unary coding	Binary
0	0	0
1	10	01
2	110	10

(b)

(1):

CABAC uses previous bits as context. For each new context, CABAC will generate a new QM coder to encode the current bit. If the context already exists, encode the current bit with corresponding QM coder.
Framework of CABAC:

1. Binarizer
2. context modeling
3. Binary arithmetic coding

Function of each module:

- Binarizer: Map input symbols to binary streams
- context modular: Build QM coder for new context, associate already existing context with QM coder
- Regular coding engine: encoding bit stream by using corresponding QM coder
- Bypass coding engine: Encoding bit stream by using single QM coder

(2):

Table4

Directly convert one byte to 8 bits			
Number of bits in context	File	Compressed size	Compression ratios
	binary.dat	25	0.3787878788
	text.dat	55	0.8333333333
	audio.dat	59	0.8939393939
N=1	image.dat	60	0.9090909091
	binary.dat	25	0.3787878788
	text.dat	58	0.8787878788
	audio.dat	60	0.9090909091
N=2	image.dat	60	0.9090909091
	binary.dat	21	0.3181818182
	text.dat	56	0.8484848485
	audio.dat	59	0.8939393939
N=3	image.dat	59	0.8939393939

Table5

Huffman encoding			
Number of bits in context	File	Compressed size	Compression ratios
	binary.dat	9	0.1363636364
	text.dat	37	0.5606060606
	audio.dat	26	0.3939393939
N=1	image.dat	39	0.5909090909

	binary.dat	9	0.1363636364
	text.dat	37	0.5606060606
	audio.dat	27	0.4090909091
N=2	image.dat	39	0.5909090909
	binary.dat	9	0.1363636364
	text.dat	37	0.5606060606
	audio.dat	26	0.3939393939
N=3	image.dat	38	0.5757575758

Table6

Bit-Plane encoding			
Number of bits in context	File	Compressed size	Compression ratios
	binary.dat	9	0.1363636364
	text.dat	46	0.696969697
	audio.dat	45	0.6818181818
N=1	image.dat	56	0.8484848485
	binary.dat	9	0.1363636364
	text.dat	46	0.696969697
	audio.dat	47	0.7121212121
N=2	image.dat	56	0.8484848485
	binary.dat	9	0.1363636364
	text.dat	46	0.696969697
	audio.dat	49	0.7424242424
N=3	image.dat	56	0.8484848485

(3):
Since pixels are highly correlated. Dividing image into several blocks may have a positive effect on improving the compression ratio. Instead of scanning the image line by line, dividing it into several blocks and encoding blocks one by one.

256

256

1	2	3
4	5	6
7	8	9

Result:

Huffman encoding				
Number of bits in context	Number of blocks(blocks*blocks)	File	Compressed size	Compression ratios
N=1	32	image.dat	39	0.5909090909
N=2	32	image.dat	39	0.5909090909
N=3	32	image.dat	38	0.5757575758

According to the result, this preprocessing technique didn’t contribute to a higher compression ratio. This result is reasonable as huffman coding is already very efficient among those three mapping functions. Preprocessing may be helpful with other mapping functions.