EE669 Homework #2

Yifan Wang
wang608@usc.edu

September 23, 2019

1 Problem 1: Written Questions

Encoding: **0100110010**
Initial: $MPS = 0, CurrentState = 12, A = 0x10000, C = 0x0000, carry = false$

Bit Received	A (Hex)	C (Hex)	Qe (Hex)	State	MPS	Output
0	E121	0000	1EDF	12	0	
1	F6F8	1210	2516	11	0	110
0	D1E2	1210	2516	11	0	
0	ACCC	1210	2516	11	0	
1	9458	6718	299A	10	0	10
1	A668	4758	32B4	8	0	11
0	E768	8EB0	2E17	9	0	0
0	B951	8EB0	2E17	9	0	
1	B85C	67A8	32B4	8	0	100
0	85A8	67A8	32B4	8	0	

The encoding result is:

11010110100

2 Problem 2: Encoding using the QM coder

2.1 Binary Arithmetic Coding (BAC)

(1)

File	Raw (byte)	Huffman		Bit Plane	
		Size (byte)	Ratio	Size (byte)	Ratio
audio.dat	65536	64246	0.980	72912	1.112
binary.dat	65536	1138	0.017	9095	0.139
image.dat	65536	76080	1.160	73953	1.128
text.dat	8698	5938	0.682	7615	0.875

Table 1: Compression result for BAC

(2)

From $Table1$, it can found BAC did not achieve a nice compression result expect on $binary.dat$. One reason is that in $binary.dat$ more than 97% are 1 which is ideal for using a BAC. On the other hand, other files would

have almost similar amount of 0 and 1 which means Qe is close to $0x8000$. When doing $A - Qe$, A would be much less than $0x8000$ which need several times to left shift A to make it larger than $0x8000$, during each shift would output 1 bit. In this case, the previous 1 bit input would have a several bit compressed result that lead to a larger compressed file.

For different mapping function, Huffman coding would give better performance on all media expect $image.dat$. Which is a result the Huffman coding has a shorter code length. While bit plane mapping using the 8 bits ASCII code for each symbol. However, due to the characteristic of image, it would perform better than Huffman mapping. Besides, the reason for large file size when using bit plane for $binary.dat$ it that in Huffman, each symbol is reduced to 1 bit. While in bit plane, it would encode the previous 1 bit sequence 8 times (8 bits ASCII). Resulting a compressed file 8 times larger than Huffman mapping's result. It is possible to get better result by using some special mapping function which is related to the file characteristic.

2.2 Context Adaptive Binary Arithmetic Coding (CABAC)

(1)

The main idea of CABAC is to used previous bits a provided information to predict the next bit. The flowchart of using the provided code is shown in $Figure1$. It used a class named QM to store all method. When starting an object, we need to provide a $FILE*$ which is the ptr to saved file and a string to specify function of this class (encode or decode). Then initial function of this class would given each variables a initial value based on the function. Then using a mapping function to map the symbols in the file to a binary sequence. Next step is to compute context which based on previous input (initial as 0 when starting encoding). Context would be saved to a lookup table to predict next bit. Once get a bit, it would be the same process as previous BAC to have the output bits and they are the compressed result.

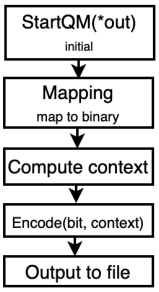

Figure 1: Process of CABAC

(2)

By using previous context, it would be helpful to predict Qe in a more accurate manner. So that compressed file would be much smaller. Compression ratio under required parameters using different mapping method is shown in $Table2$. Both ASCII and Bit Plane would use 8 bits to represent a symbol. While Huffman coding dictionary is derived from $HW1$. It can be found that when choosing a small context length, compression result is not brilliant. Compression ratio may even larger than 1 when 1 and 0 is (almost) equally distributed in the file (pretty similar to BAC).

It is the same case for using mapping from Huffman, file size would slight increase after using CABAC compare to the compressed file size after using Huffman coding alone. Bit plane mapping would be most suitable for compressing images where local value for each pixel would match its neighbour that would be help for reducing size since the consequent symbols would the same. $Table3$ shows the best compression ratio we can achieve. It seems a moderate length of previous context would be helpful during the compression process. Either a too short or too long context would decrease the performance of compression.

Mapping	File	Raw (byte)	Context=1		Context=2		Context=3	
			Size (byte)	Ratio	Size (byte)	Ratio	Size (byte)	Ratio
ASCII	audio.dat	65536	66252	1.011	66138	1.009	65929	1.006
	binary.dat	65536	1913	0.029	1886	0.029	1863	0.028
	image.dat	65536	66961	1.022	66552	1.016	66523	1.015
	text.dat	8698	9010	1.036	8947	1.029	8850	1.017
Huffman	audio.dat(compressed)	53166	54697	0.835	54694	0.835	54560	0.833
	binary.dat(compressed)	8192	950	0.014	945	0.014	938	0.014
	image.dat(compressed)	62433	64665	0.987	64601	0.986	64562	0.985
	text.dat(compressed)	4864	5054	0.581	5056	0.581	5051	0.581
Bit Plane	audio.dat	65536	45141	0.689	45097	0.688	44955	0.686
	binary.dat	65536	7597	0.116	7532	0.115	7480	0.114
	image.dat	65536	57217	0.873	56933	0.869	56779	0.866
	text.dat	8698	6447	0.741	6327	0.727	6294	0.724

Table 2: Compression result for CABAC using differen mapping.

File	Raw (byte)	ASCII			Huffman			Bit Plane		
		Context	Size (byte)	Ratio	Context	Size (byte)	Ratio	Context	Size (byte)	Ratio
audio.dat	65536	17	34332	***0.524***	17	40785	0.622	8	45203	0.689
binary.dat	65536	22	1411	0.022	9	866	***0.013***	30	4234	0.065
image.dat	65536	10	58670	0.895	12	63399	0.967	8	56600	***0.864***
text.dat	8698	17	4127	0.474	16	3597	***0.414***	2	6294	0.724

Table 3: Best compression result achieved for CABAC using different mapping and context length.

(3)

One idea to pre-processing the image is to separate large image into small blocks (windows), since there would be higher probability that one block would have much similar pixel value which is ideal for using bit plane mapping function to get a nice compression ratio. Only if we can separate blocks into right amount number of blocks. Some test result is shown in $Figure4$. For simplicity I only tried the $2^n \times 2^n$ window size. Result is slightly better than using bit plane on the whole image. And compression ratio is expected to reduce when using other window size where height and width may not the same or equal only to 2^n.

Block window	Context	Size (byte)	Ratio
4×4	9	58986	0.900
8×8	10	56330	0.859
16×16	1	56485	0.861
32×32	2	56295	***0.858***
64×64	3	56385	0.860

Table 4: Pre-processed image.dat's compression result using Bit Plane mapping

More:

One more advanced method is to adaptively separate image into block. The idea is to check the similarity or smoothness inside a block, if it is less than a threshold, stop separate the block (which is similar to build a binary tree). Otherwise, continue separate it, until not separable, meet the smoothness requirement or meet the pre-set minimum block size. Using the example in $Figure2$, where we kept separate image into 2 sub-image. The $blue$ window shows the left sub-image after first separate. Continuing do the second dividing get one in $yellow$ window, since there remains some dissimilarity in this sub-image, keeping separate. Resulting

the one in red window where all the pixel are similar to each other, that would be nice for compression so that stop separating for this sub-image. While for the another part of sub-image, it might need further dividing to get a smooth sub-image for compression.

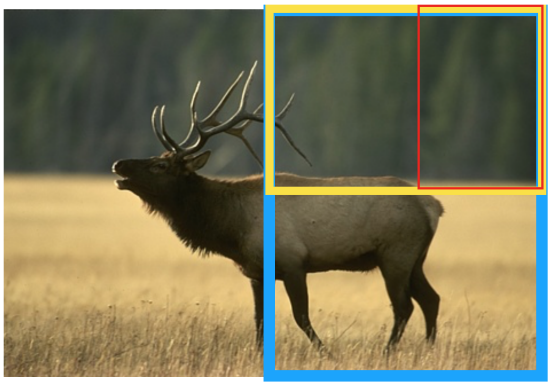

Figure 2