PROJECT #1: LOSSLESS COMPRESSION

Issued: 1/16/2015 Due: 11:59 pm PST, Sunday 2/8/2015

General Instructions:

1. Assignment descriptions and all files mentioned in this homework are available under "Assignments" on DEN Website: http://den.usc.edu.
2. Please follow the submission guidelines for homework, found in the "Course Documents" folder on DEN website.

Problem 1: Written Questions (30%)

1. Huffman Coding (11 points)

Consider the source with skewed statistics:

$$P(A) = \frac{3}{9}, \quad P(B) = \frac{3}{9}, \quad P(C) = \frac{2}{9}, \quad P(D) = \frac{1}{9}$$

a) What is the entropy of bit stream generated from this source? (1 point)
b) Draw the Huffman coding tree and indicate the assigned code. (4 points)
c) What is the average code-word length for fixed-length code in this case? What is the average code-word length if we apply Huffman code? (2 points)
d) Calculate the coding redundancy if Huffman code were to be directly applied? (2 points)
e) Encode the following bit-stream using the Huffman code designed in previous part. (2 points)

$$ABDCABBBCADBCABCAA$$

2. Lempel-Ziv Coding (LZ-78) (10 points)

Suppose we are in the middle of LZ-78 coding. The previous word is "C", and the dictionary is shown below:

Index	Symbol
0	NULL
1	C
2	B
3	D
4	DA

a) Encode the input sequence "…CAEABDDDACCB…" using Lempel-Ziv-78 coding scheme. Show the updated dictionary and coding output for each step. You may display the following table in your report: (8 points)

Dictionary	Input	Left	Output Phrase	Output Character

b) Please show the updated dictionary structure using multi-way tree. (2 points)

3. Arithmetic Coding (9 points)

Given symbol set S={A, B, C, D} with probabilities {0.05, 0.15, 0.35, 0.45}. Assume that the range of each symbol is sequentially set as A, B, C then D.

a) Is arithmetic coding generally more or less efficient than Huffman coding? Please explain. (3 points)
b) Use arithmetic coding to code the string 'CABDD', and show the output and detailed steps in the process. (3 points)
c) Use arithmetic coding to decode '0.450753' (decimal) given that the string is 5 symbols long and show the detailed steps taken. (3 points)

Problem 2: Entropy Coding (40%)

Design and apply the following entropy encoding algorithms. Input data files for this problem are four types of media: audio, text, binary and gray-level images (given as .dat files in the package).

1. Shannon-Fano Coding (10 points) - the Shannon-Fano encoder uses the top-down approach to construct the code tree when encoding. Perform the following steps:

a. Write a program to compute the entropy, relative frequency of symbols, and any necessary statistical information from the source for each input file.
b. Develop a Shannon-Fano encoder based on the global statistics.
c. Apply the encoder to each input file. Report the compressed file sizes and the compression ratios. Compare your results with the theoretical bounds for each case.

2. Huffman Coding with Global Statistics (15 points) - to implement a Huffman encoder using global statistics, please perform the following steps. Remember that the Huffman encoder uses the bottom-up approach to construct the code tree when encoding.

a. Develop the Huffman encoder based on the global statistics.
b. Apply the encoder to each input file. Report the compressed file sizes and the compression ratios. Compare your results with the theoretical bounds for each case.
c. Discuss the different compression ratios achieved in different media and compare these to your results from the Shannon-Fano encoder.

3. Huffman Coding with Locally Adaptive Statistics (15 points) - for real-time compression applications, we have to encode the symbols on the fly. The encoder/decoder must update the statistics of the symbol set and the coding table after encoding/decoding every symbol. To develop an adaptive Huffman encoder and decoder, please do the following.

a. Design an adaptive coding method and implement this procedure in the encoder. Explain your method clearly in your report.
b. Apply your algorithm to the four input files and report the compressed file sizes and the compression ratios. Discuss the results, comparing to theoretical bounds and the global statistics results, as well as the previous results.

NOTE: References [1] and [2] are informative in the design of your algorithms for both variations of Huffman Coding:

Problem 3: Run-Length Coding (30%)

Run-length coding is one of the coding schemes that have been widely used. The basic idea is to encode a sequence of repeated symbols as the number of repetition followed by the symbol itself. For example, "`00 00 00 10 01 01 E8 E8 E8 E8`" is encoded to "`03 00 01 10 02 01 04 E8`".

a. Basic Scheme (5 points)
Write a program to perform encoding and decoding by using the idea described above. Can the program always compress the input file? Why or why not? What is the optimal compression ratio and what is your result for each of the 4 input files (audio.dat, text.dat, binary.dat, image.dat)?

b. Modified Scheme (10 points)
One approach to enhance the compression ratio of the run-length coding is not to encode one singular symbol. For example, "00 10 11 00 00 00 00 21 21 C3 C3 F2" becomes "00 10 11 84 00 82 21 82 C3 81 F2". Note that "84" has MSB="1", which indicates there are four (84-80=4) continuous symbols "00". The singular symbol "F2" is encoded as "81 F2" to avoid confusion because "F2" also has MSB="1". Write a compression program by using this modified approach. Discuss your result for each of the 4 input files (audio.dat, text.dat, binary.dat, image.dat) and compare it with that obtained with the previous approach. You should also be able to improve the compression ratio by applying pre-processing to the input files. Show the best results that you can get for the four input files. You must also design the decoder for Modified scheme.

c. Move-to-Front Transform (15 points)
Since most files naturally contain a "run" of symbols, Run-length coding is effective as a pre-processing stage in the compression process. Besides the Run-length coding, Move-to-Front (MTF) is another excellent pre-processing algorithm that reduces the redundancy in repeat patterns. Please read [3] for more information about the MTF.
The MTF scheme transforms the symbol stream into a sequence of index according to an adaptive symbol tables. It simply "moves to front" the last symbol seen such that its index in the table of codes becomes zero. See an example below:

Input	Output	SymbolTable
dtggggg	3	'abcdefghijklmnopqrstuvwxyz'
d**t**ggggg	3 19	'dabcefghijklmnopqrstuvwxyz'
dt**g**gggg	3 19 7	'tdabcefghijklmnopqrsuvwxyz'
dt**gg**ggg	3 19 7 0	'gtdabcefhijklmnopqrsuvwxyz'
dt**ggg**gg	3 19 7 0	'gtdabcefhijklmnopqrsuvwxyz'
dtggg**g**g	3 19 7 0 0	'gtdabcefhijklmnopqrsuvwxyz'
dtgggg**g**	3 19 7 0 0 0	'gtdabcefhijklmnopqrsuvwxyz'

The MTF, as a pre-process stage, could improve the efficiency of other compression method. First, implement the MTF encoder with your own code and properly initialize the code table. Then, preprocess the uncompressed files with the MTF and perform compression with adaptive Huffman coding. Compare the compression ratio with the result in Problem 2 without preprocessing. Also, implement the decoding process of the MTF. Explain your implementation details and the data structure in your program.

Appendix:

Four input files contained in the homework package:

1. image.dat: A 256 gray-level image of size 512*512
2. text.dat: A sample ASCII text file
3. audio.dat: An audio file in raw data format which has 65536 samples. Each sample is 1 byte.
4. binary.dat: A bi-level image of size 512*512. Each byte in this file is one pixel and has two possible value: '0x00' or '0xFF'.

References:

[1] http://www.binaryessence.com/dct/en000079.htm
[2] http://compgt.googlepages.com/dynamic
[3] https://sites.google.com/site/compgt/mtf