HW SET #2 (DUE BEFORE CLASS ON FEB 3, WED)

Problem 1 (15 points) Show that $\mathbf{C}$ as defined in Problem 1, HW SET #1, is an orthogonal matrix, i.e., $\mathbf{C}^T\mathbf{C} = \mathbf{C}\mathbf{C}^T = \mathbf{I}_N$ where $\mathbf{I}$ is an $N \times N$ identity matrix. (Therefore, the 2-D inverse DCT (IDCT) can be written as $\mathbf{X} = \mathbf{C}\mathbf{Y}\mathbf{C}^T$.) Hint: $\sum_{k=0}^{N-1} \cos\frac{kmp}{N} = \mathrm{Re}\left[\sum_{k=0}^{N-1} e^{j\frac{kmp}{N}}\right]$

Problem 2 (10 points) One property of vector quantization is that, at high bit rate, the codebook size becomes very large and a large number of images are required for training. Now derive the general relationship between the bit rate and the codebook size and the number of training images required. Let R denote the bit rate in bpp (bits per pixel) and the block size be $M \times M$. How many codewords are there in the codebook? Assume that each pixel in the original image is 8 bits. How many bits are needed to transmit the codebook? Suppose one would need training data 16 times the size of the codebook (e.g., 16000 vectors to train a codebook of 100 codewords), how many images are needed for training?

Problem 3 (40 points) Write a subroutine (method) to construct Huffman codes. The input to the subroutine contains the probability distribution of the symbols, e.g., (0.5,0.3,0.07,0.05,0.05,0.03)

The output is an ASCII text file containing the Huffman codes for the symbols, e.g.,

```
S1 1
S2 01
S3 0011
S4 0010
S5 0001
S6 0000
```

A few notes:

1. The input probabilities may not be monotonically decreasing. Please sort them from the largest to the smallest before you start constructing the codes.

2. After you merge the two symbols that have the lowest probabilities, the sequence of the resulting probabilities may not be monotonically decreasing any more, please re-sort them as necessary after each merge. If there is tie, put the merged probability after the probability it ties with.

3. If there is a tie when you choose the two symbols that have the lowest probabilities, choose the symbols that appear later in the set of symbols.

4. When you assign the 0's and 1's to the branches in the tree, always assign 0 to the lower branch (the symbol that has lower probability) and 1 to the upper branch (the symbol that has higher probability).

(OVER→)

Problem 4 (15 points) Write a program to encode a file using Huffman coding. Treat a file as a sequence of bytes, and each byte as an `unsigned char`. Therefore, the set of symbols (the alphabet) is {S0, S1, S2... S255}. This program should be a two-pass process. In the first pass, the probability (frequency) of each symbol is collected, and a Huffman table is generated based on these probabilities (same as in Problem 3). In the second pass, each input symbol is converted into a codeword (composed of 0's and 1's). These codewords are cascaded together and saved into an output file. If the total number of bits is not a multiple of 8, pad zero bits at the end. Save also the Huffman table into a separate file. Apply your program to each of test files that are available on the web. What is the compression ratio you get for each file (excluding the Huffman table)?

Problem 5 (20 points) Write a Huffman decoder. The input to the decoder contains one bitstream and its corresponding Huffman table. To build an efficient decoder, rather than looking up each incoming codeword in the codebook sequentially, you should utilize the tree structure of the Huffman code.

We will test your encoder and decoder with some test data. Please deposit your code into your directory in `/afs/ece/class/ece796/handin/[your userID]/HW2`. Put all your source files (`*.h`, `*.c`, etc.) and the executables (please name the encoder huffmanencode, and the decoder `huffmandecode)` there. Please also include a `readme.txt` file detailing all the steps, libraries, and the machine the TA needs to know to re-compile your code to re-generate the executables. Note that it is your responsibility to let the TA know how to compile and execute your code. Using C or C++ on PCs or Unix machines in the clusters is required. If you are considering other languages or platforms, please discuss with the instructor. After successful compilation, the TA will test your code by typing:

```
huffmanencode filename huffmantable filename_bits
huffmandecode filename_bits huffmantable filename_dec
```

where `filename` is the name of the original file, `filename_bits` is the compressed bitstream, `filename_dec` is the decompressed file, and `huffmantable` is the Huffman table (an ASCII file with the format as in Problem 3). Note that `huffmantable` is an output file of `huffmanencode`, and an input file to `huffmandecode`. The TA will check the content in `huffmantable` and the content in `filename_bits`, and compare `filename` and `filename_dec` to verify your result.

Project Proposal

Please email to the instructor a one-page (or more) description of the project you would like to do for this course. Include a list of names of the group members. Each group only needs to submit one proposal. Each group should have either two or three members. In your proposal, specify the goals/deliverables both for the mid-term project and the final project. Try to align the mid-term project with the final project in that the mid-term project serves as a checkpoint for your progress toward the final project. For example, if you would like to implement an H.320 video conferencing system, you should finish H.261 by mid-term and finish the rest (audio coding, signaling, and multiplexing) for the final project. Also, keep in mind that you should choose a topic with workload that is proportional the number of group members. For example, building a complete a complete H.320 video conferencing system is perhaps too much for a two-person group. On the other hand, implementation of JPEG is too little. One purpose for the proposal is for the instructor to provide you feedback and suggestions in this aspect.