Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
Department of Mechanical Engineering

6.050J/2.110J **Information and Entropy** **Spring 2003**

Issued: February 7, 2003 **Due: February 21, 2003**

Problem Set 2

Calendar note: This assignment is due two Fridays from lecture since the compression unit is two weeks long.

Laboratory Experiment

This exercise shows you the basis behind compression of images. Most video compression codecs today use a form of the Discrete Cosine Transform (DCT) in their core. The DCT separates an image into its spatial frequency components in the x and y directions. For example, a solid color image will have a DCT with low frequencies while a line drawing image will have a DCT with very high frequencies. The transform of an image has the same number of coefficients as the image, i.e. an 8×8 image will have an 8×8 DCT. The transform is reversible, in the sense that the original image can be recovered, exactly, from its DCT.

In MATLAB, type `dctdemo` to view a demonstration of the DCT. The following is an explanation of what is happening. Clicking on "Info" will give you a little more information.

1. The original image is divided into 8×8 blocks of pixels.
2. DCT is applied to each block, resulting in an 8×8 block of DCT coefficients. The values in the upper left of each block hold the amount of low x and y frequencies and the values in the lower right the amount of high x and y frequencies.
3. Compression occurs when small values of the DCT coefficients are set to zero. This can be seen by moving the horizontal slider to block out certain values and clicking "Apply".
4. The dropped (small) values are set to zero, and the inverse DCT used to produce an approximation of the original image.

The error image is the subtraction of the original pixel values from the reconstructed pixel values. When we try to compare the quality of a lossy compression algorithm, such as this one, qualitative analysis is too subjective. Therefore, various numerical formulas have been developed to assign a quantitative value to the quality of the compressed image. In this demo, the mean squared error is used for this purpose.

Problem 1: Is it Over-Compressed or is it Modern Art?

Write your own video image compressor in MATLAB! This will be as simple as possible, so don't sweat. Write all your MATLAB commands in `ps2p1.m`.

a. Start off by loading the flower image by typing the following in MATLAB.

```
load imdemos flower;    % Load flower matrix from image demos library
flower = double(flower);% Convert the matrix to double precision
colormap('gray');       % Grayscale for any pictures you want to display
                        % (This creates a blank window that we'll use soon)
imshow(flower,[0 255]); % Display flower in the window
```

b. Perform a 2D DCT operation on 8×8 blocks of the image matrix. You might find the commands `blkproc` and `dct2` useful. Use the first form of `blkproc` explained in its help page (`help blkproc`), where FUN is `'dct2'` (put single quotes around dct2). To keep help information from scrolling off the screen, type `more on` at the MATLAB prompt.

c. Set to zero all coefficients with absolute value less than 10. An example to do this can be found in the help of dct2. We will later want to vary this cutoff value and count the number of non-zero values. After this step, many codecs apply run length and variable length encoding to transmit or store the image efficiently for later use.

d. Perform a 2D Inverse DCT operation on each of the 8×8 blocks of the image matrix. The values of the resulting matrix will be in floating point, but you should round the elements to the nearest integer since image pixel values are integers. Use the `blkproc` and `idct2` commands similarly to the way you performed the forwards DCT in Part b. Then use the `round` command to complete the reconstruction of the image.

e. Determine the mean squared error between the original and reconstructed image. The definition of mean squared error to use is `mean2((x - y).^2)`. Values will appear larger than in the demo because our error definition does no normalization. The answer you should receive is 10.6891. Note that you can see your new image with the `imshow` command the same way you displayed the original image. If you want a new window for the image, so it doesn't just paint over the previous one, type figure at the MATLAB prompt.

Now compose a graph, having along the x-axis the number of non-zero values of the matrix after the cutoff procedure (before you did the inverse DCT), and the mean squared error (after you did the inverse DCT and rounding) on the y-axis. The number of non-zero values is the number of bytes required to store the image (ignoring overhead), so smaller means more compression. Range the cutoff value from 0 to 100 in increments of 4. The use of `nnz` and `plot` will do the trick. It will be easiest to use them if you create a vector with the numbers of non-zero values and another vector with the mean squared errors.

Write in `ps2diary` the largest byte size for which you can detect the difference between the original image and the reconstructed image by eye. Include any comments in the diary file. Be sure the script in `ps2p1.m` is executable in MATLAB.

Problem 2: Compression is Fun and Easy

Now you can get your hands dirty with the LZW (Lempel-Ziv-Welch) compression algorithm.[1] Although this problem will not require the use of MATLAB, you may use it if you wish. A good reference to the LZW algorithm is http://www.rasip.fer.hr/research/compress/algorithms/fund/lz/lzw.html. There are many derivatives of LZW that are more powerful, but we'll stay with what was explained in lecture and in the web link.

a. Use the 7 bit ASCII table that can be found in Chapter 2 of the notes, handed out at the first lecture, for your basic alphabet. Notice that the ASCII table ends at HEX `7F` (Hexadecimal is Base 16 using A - F for values 11 - 15, so HEX `7F` is 127). Your dictionary will therefore range from HEX `80` through HEX `FF`. Encode the following message in HEX using the LZW technique:

`peter piper picked a peck of pickled peppers`

[1] The 2.110/6.050 staff would like to thank Joe Huang, the Spring 2000 class TA, for developing the image compression exercise used in this assignment. It is, in our opinion, really cool because it truly links what we discuss in class to the real world; once you've completed it, you will understand and have experience with the way JPEG images are created. For this reason, the problem has not changed much since last two years, and we are trusting you to complete it without consulting last year's solutions. We hope you enjoy it!

Be sure that you can decode the message properly. Use the examples in Section 3.2 of the notes as a guide for formatting your solution. Put your results in ps2diary and include your dictionary entries. Note: HEX `20` is the space. For this exercise use HEX `7F` (DEL) to signify the end of the message and HEX `00` (NUL) for the start of the message.

b. It is proposed to improve this coding by using only the ASCII control characters HEX `10 - 1F` to hold dictionary entries. In one paragraph, summarize the advantages and disadvantages of this proposal. Things to consider might include ability to handle various kinds of messages, efficiency, complexity of the coder and decoder, and adherence to widely accepted standards. Under what circumstances might this proposal be a good idea or a bad idea?

c. **Optional (just for fun):** Implement the LZW coding and decoding operations of part above, using any computer language you know and have access to. Assume that any message you handle will be short so that the dictionary will not extend beyond HEX FF (naturally a robust implementation would need to handle dictionary overflow).

Turning in Your Solutions

You should have two files (one M-file and one diary). Name the M-file `ps2p1.m` and name the diary `ps2diary`. You can rename your files at your `athena%` prompt using the following command.

```
mv oldfilename newfilename
```

Turn in this problem set by e-mailing your M-files and diary along with your answers to any problem(s) not done using MATLAB, to 6.050-submit@mit.edu. You may do this either by attaching them to the e-mail as text files, or by pasting their content directly into the body of the e-mail (if you do this, please somehow indicate where each file begins and ends). Alternatively, you may turn in your solutions on paper in room 38-344. The deadline for submission is the same no matter which option you choose.

Your solutions are due 5:00 PM on Friday, February 21, 2003. Later that day solutions will be posted on the web.