LAB 1 - TASK 1

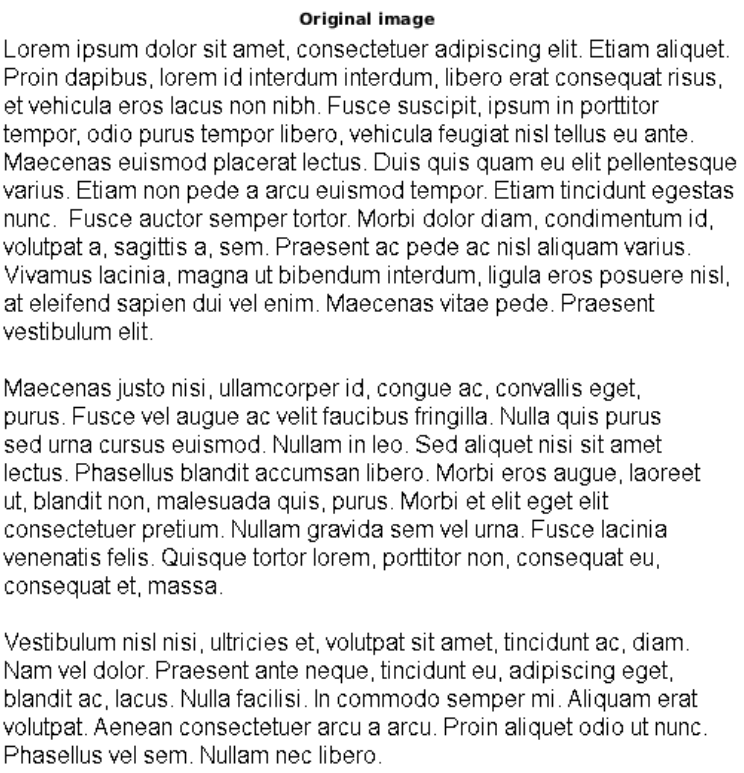
In this task, you will compare the lengths of the bitstreams required to encode a black and white image using four different coding schemes.

```
% Load the input image
lorem_img = imread('lorem_img.png');

% display the raw image
figure(1);
imshow(lorem_img);
title('Original image');

% run-length encode
run_length_code = runlength_encode(lorem_img);

% convert the binary array into an decimal array of run lengths
runs = bin2decArray(run_length_code);

% Huffman encode
% huffman_encode(runs_dict)
```

Unanswered

Figure 1

tutorials



size_raw_data =

250000

size_run_length =

301688

size_huffman =

100981

Run Code

INSTRUCTIONS

This task does not require any MATLAB coding. To start, simply hit the **Run Code** button.

The code in the above window first loads a black and white image and displays it in Figure 1 using the MATLAB function **imshow**. It then shows the lengths of the bit streams required to encode the image using "raw" image coding, run-length encoding where run-lengths are encoded as 8 bit binary numbers, and run-length encoding where the run-lengths are encoded using Huffman coding.

The black and white image is a 500-by-500 array of pixels contained in a file called "lorem_img.png". This is loaded into a MATLAB variable called **lorem_img**, which is a 500-by-500 dimensional array containing 0's and 1's corresponding to black and white pixels. We only need one bit to encode each pixel. One way to encode the image is simply to list out the values of each pixel one after the other, row by row. We will refer to this as the "raw" image coding.

After loading and displaying the image, the code then generates the run-length encoding of the image, where the run lengths are encoded as 8 bit binary numbers. This is done using the function **runlength_encode**, which was written for this lab. This function scans through the image row by row and counts the lengths of runs of consecutive white or black pixels. These lengths are encoded as 8 bit binary numbers. For example, suppose that we have part of a row containing pixel values given by:

1110000011110000111

This contains alternating runs of white and black pixels with lengths 3, 5, 4, 4, and 3. When encoded as 8 bit numbers, this would be encoded using the bit sequence

[00000011][00000101][00000100][00000100][00000011]

where the square brackets are included to help distinguish the 8 bit numbers, but would not be included in the bit stream. Since the longest run that can be encoded by an 8 bit binary number is 255, if the program encounters a run of white (black) pixels longer than 255, it splits it into separate runs of at most 255 pixels separated by runs of black (white) pixels with length 0. For example, a run of 520 pixels would be encoded by run lengths 255, 0, 255, 0, 10.

We can see from the above example that encoding run lengths by 8 bit binary numbers may not be a good idea, especially if there are many short runs, and only a few long runs. The next part of the code finds a more efficient encoding of the run lengths using Huffman coding. The code first converts the binary representation of the run lengths to a decimal representation using the function **bin2decArray**. The function **huffman_encode** then estimates the probabilities that different run lengths occur, uses this information to generate a Huffman dictionary, and uses this dictionary to encode the sequence of run lengths. Note that the second argument of **huffman_encode** enables us to select between using the same dictionary to encode black and white runs, or using separate dictionaries.

Your task is to compare the lengths of bit streams required to encode the image using the "raw" image coding and the run-length encoding where the run lengths are encoded using 8 bit binary numbers and by Huffman coding. Experiment with the code by modifying the image to be encoded and by changing whether one or two dictionaries are used. Use the insight gained to answer the questions below.

LAB 1 - TASK 1 QUESTION 1 (1/1 point)

Is the run-length code always shorter than the raw data?

- Yes
- No ✔

EXPLANATION

The run-length code of the original image in our example is longer than the raw data. This is because when scanning row-by-row, the image contains many runs with short lengths.

You have used 1 of 1 submissions

LAB 1 - TASK 1 QUESTION 2 (1/1 point)

What happens when we transpose the input image before executing run length encoding? (Use the MATLAB transpose operator ')

- (•) The run length code of the transposed image is shorter than that of the original image. ✔
- ( ) The run length code of the transposed image is longer than that of the original image.
- ( ) The run length codes of the transposed and original images have the same length.

EXPLANATION

If we transpose the image, i.e. lorem_img = lorem_img', columns become rows and rows become columns. Thus, we are essentially scanning for runs of consecutive white/black pixels along the columns. The runs along the columns are generally longer, which is better for run length encoding.

You have used 1 of 1 submissions

LAB 1 - TASK 1 QUESTION 3 (1/1 point)

What happens if we use separate dictionaries for the Huffman encoding of the black and white run lengths?

- (•) The Huffman code will be shorter. ✔
- ( ) The Huffman code will be longer.
- ( ) The Huffman code will have the same length.

EXPLANATION

Using separate dictionaries we will be obtain a shorter Huffman code (100981 bits instead of 117374 bits). This is because the distribution of runs of white and black pixels are different, and using separate Huffman dictionaries enables us to exploit this to the fullest extent.

You have used 1 of 1 submissions