LAB 1 - TASK 3 (3 points possible)

In this task you will generate the Huffman dictionary by inspecting the distribution of run lengths.

```
40
41 % recreate 500 by 500 image from the run length code
42 img_new = runlength_decode(new_length_code);
43 figure(3);
44 imshow(img_new);
45 title('Recreated image');
46
47 % Put figure  1 and 2 on top by calling them again
48 figure(2);
49 figure(1);
50
51 % compare the lengths
52 size_raw_data = length(lorem_img(:))
53 size_huffman = length(huffman)
54
55
```

Unanswered

Figure 1

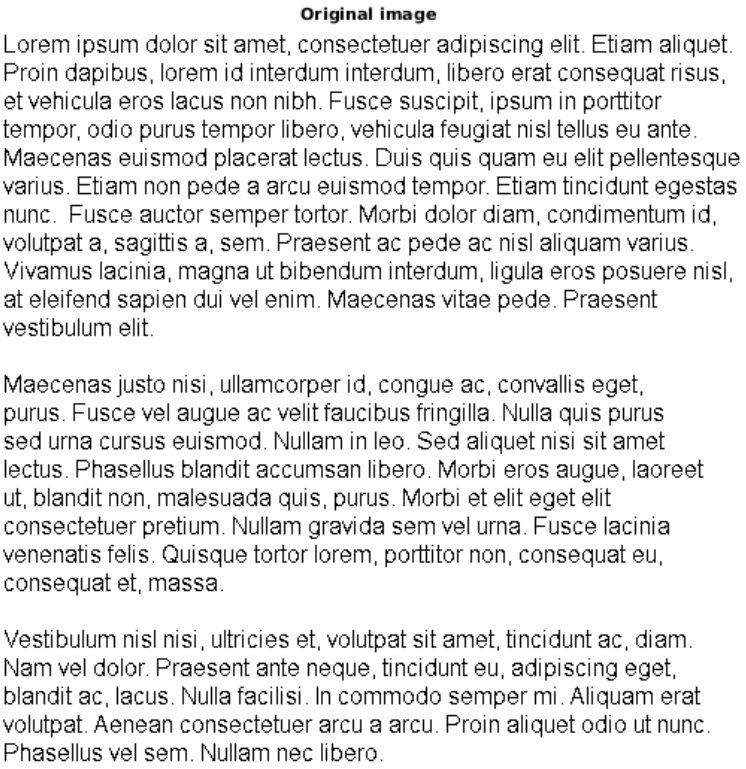


Figure 2

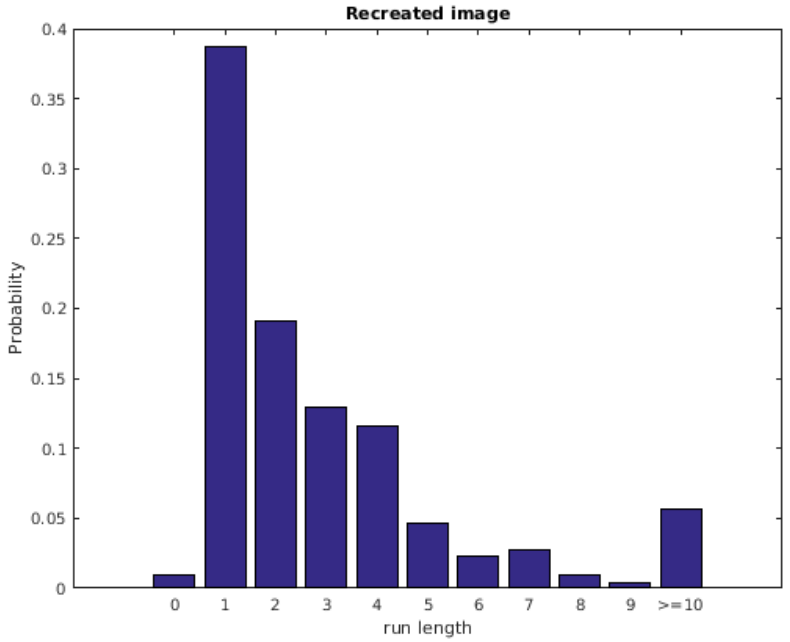
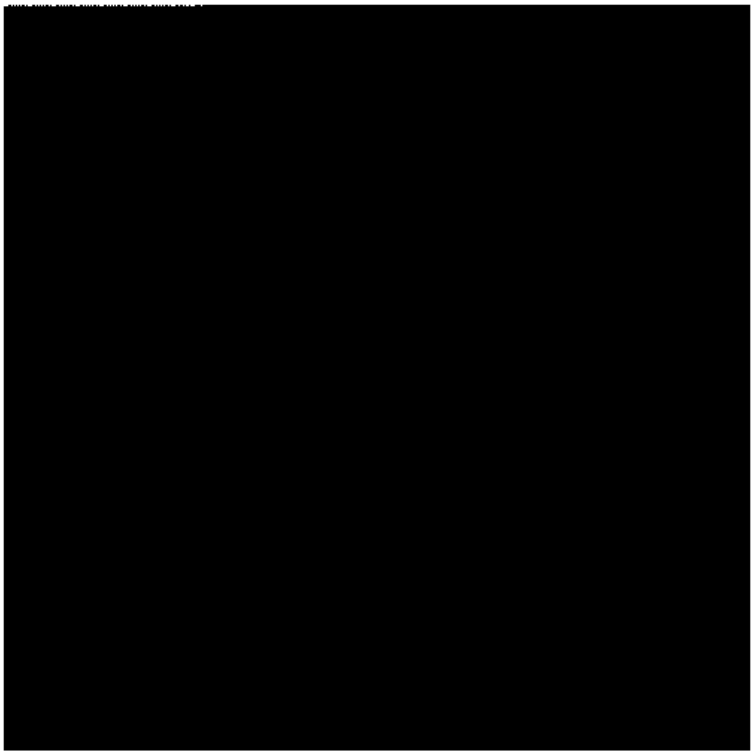


Figure 3



prob =

0.0092 0.3870 0.1911 0.1294 0.1162 0.0461 0.0232 0.0274 0.0099 0.0040 0

[Warning: runlength_decode: Invalid Run Length]

size_raw_data =

```
    250000

size_huffman =

    104799
```

Run Code

You have used 0 of 10 submissions

INSTRUCTIONS

The MATLAB code contained in the initial code window extracts run lengths from a black and white image and then uses the dictionary contained in the variable **dict** to encode them. However, the dictionary given in the initial code window is not optimal. Your task is to use the Huffman coding algorithm to find an optimal dictionary with which to replace the initial dictionary. This is part of the work performed by the function **huffman_encode** you used in Task 1.

If you click the **Run Code** button, the initial code will show 3 figures. Figure 1 displays the original image. Figure 2 shows the probability distribution of the run lengths. Figure 3 shows the image reconstructed from the run length encoding using the dictionary contained in **dict**. The code also returns the values of three variables: **prob**: the probabilities of different run lengths, **size_raw_data**: the length of the encoding or the raw data, and **size_huffman**: the length of the encoding of the run lengths with the dictionary. Because the original dictionary is a valid dictionary, the original and reconstructed images are the same. However, because the dictionary is sub-optimal, the length of the bit stream is longer than that obtained by using the dictionary in Task 1 (117,374).

Let's first see how the initial code works. The first part of the code up to the comment "% huffman code" is the same as in Task 1. The vector **runs** contains the lengths of alternating runs of white and black pixels, where the maximum run length is kept at 255 using the technique used in Task 2.

The next part of the code computes the distribution of the run-lengths by using the function **histogram**. Note that there are in total 11 bins. The first 10 bins contain runs with lengths exactly equal to N, where N runs from 0 to 9 pixels. The last bin contains the runs with length equal to or greater than 10 pixels. The probability of each bin is shown graphically using the function **bar**, and the probabilities are returned in the variable **prob**. Recall from the lab video that if the run length is equal to or greater than 10 pixels, we use an escape code followed by an eight bit binary number encoding the exact run length. Thus, we need to send encode 11 possible symbol values. Notice that the symbol probabilities are not all equal.

The dictionary containing the encoding of the 11 symbol values are contained in the cell array **dict**. Each array in **dict** is a binary vector corresponding to the binary codeword used to represent the corresponding symbol. Thus, the first 10 arrays are used to represent run lengths from 0 to 9. The final array is the escape code.

Initially, the dictionary uses the standard 4 bit binary representation of the numbers from 0 to 10. Thus, each symbol is represented by codewords of equal length. However, from lecture, we know that since the symbols occur with different probabilities, we can obtain a better encoding using variable length codewords, where shorter codewords are used to represent more frequent symbols.

Your task is to find an optimal dictionary for representing these 11 symbols using the symbol probabilities and the Huffman algorithm. Once you have found it, replace the value of **dict** defined below the line

% % % % Revise the following code to generate a valid and efficient dictionary % % % %

Do not change the other parts of the code.

The remaining part of the code uses this dictionary to encode the run lengths, and to measure the length of the resulting bit stream. It also checks whether the dictionary is valid by reconstructing the image from the run lengths encoded by the dictionary using the function **huffman_encode_dict**. If your dictionary is correct, the original and reconstructed images should be the same and **size_huffman** should be equal to 117374.