# THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY ISDN 2602

**Laboratory 4: Source and Channel Coding (5%)**

**Answer Sheet**

Please write down your answer here and submit your answer on GitHub by Wednesday (Oct 29th) 23:59

***Part I: Source Coding***

# Task 1 – Length of the bit streams

In this task, we will compare the lengths of the bit streams for four source coding algorithms applied to a black-and-white image: "raw" image encoding, run-length encoding with lengths encoded as 8-bit binary numbers, and run-length encoding with lengths encoded by Huffman coding with one or two dictionaries.

# Check Point:

1. Write down the lengths of the bit streams using “raw” image encoding and the run-length encoding. Is the run-length code better than the raw encoding? **Explain why**.

Raw image encoding = 250000 bits

Run length encoding (8-bit binary) = 301688 bits

Run length encoding (Huffman, 1 dictionary) = 117374 bits

Run length encoding (Huffman, 2 dictionaries) = 100981 bits

Run length is better than raw encoding but only when paired with Huffman coding. 8-bit run length is worse than raw encoding for images with many short runs. The Huffman-enhanced run-length encoding is significantly more efficient especially when theres a separate dictionary for black and white

1. Type “help transpose” in the command window to learn how to perform matrix transpose operation on a matrix in MATLAB. Revise the MATLAB codes so that the image will be rotated along the diagonal. Then, write down and compare the lengths of the bitstreams for these four source coding algorithms before and after the rotation. **Explain why**.

(original)

Raw image encoding = 250000 bits

Run length encoding (8-bit binary) = 301688 bits

Run length encoding (Huffman, 1 dictionary) = 117374 bits

Run length encoding (Huffman, 2 dictionaries) = 100981 bits

(transposed)

Raw image encoding = 250000 bits

Run length encoding (8-bit binary) = 196680 bits

Run length encoding (Huffman, 1 dictionary) =134892 bits

Run length encoding (Huffman, 2 dictionaries) = 120565 bits

Raw image has no change because each pixel is stored as 1 bit regardless of orientation, the 8-bit binary decreases so it significantly improved after rotation because the rotated image has fewer short runs and more long runs thus reduces the 8-bit-run-length entries. while the Huffman with dictionary increased bits because its optimized for the frequency distribution of run lengths. After rotation the distribution of run lengths likely become less skewed, more varied, and less repetitive which makes Huffman coding less efficient. This shows that compression performance depends not just on the meothd but also on the data structure and orientation

***Fill in the answers to the blanks and Show your result to the TA.***

# Task 2 – Huffman code

In this task, you will generate the Huffman code for a set of run-lengths, and use it to encode the run- lengths of black or white pixels. You will find that Huffman coding enables us to encode the sequence of run lengths using fewer bits than the standard 8-bit encoding.

# Check point:

1. Find an optimal dictionary to represent these 11 symbols using the symbol probabilities and the Huffman coding algorithm. Once you have found it, replace the value of **dict** defined between the line:

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

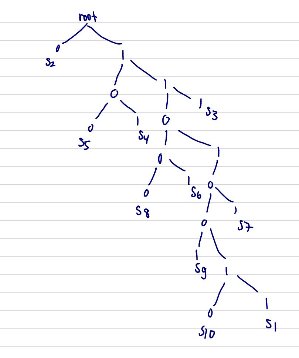
and

*% % % % Do not change the code below % % % %*

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 the **size\_huffman** should be equal to 117374.

# (Commit the revised codes to GitHub. Show your results to TAs.)

1. Attach the corresponding Huffman tree of the revised optimal dictionary.



***Fill in the answers, commit the revised codes to GitHub***

***and Show your result to the TA.***

***Part II: Channel Coding***



# Task 3 – (n,k) block code decoder and Error Correction Capability

In this task, we will implement the (n,k) block code decoder and compare the error correction capability of the repetition code, hamming block code, and no error correction code.

# Check point:

1. Generate a figure with three curves representing the BER performance.

# A graph of a graph with different colored lines AI-generated content may be incorrect.

# （Show your results to the TA）

1. Write down/Insert a screenshot of the modified code in “**blk\_decoder.m**”.

if isequal(S, [1 0 1 0])

msgblk(1) = ~msgblk(1); % D1

elseif isequal(S, [1 0 0 1])

msgblk(2) = ~msgblk(2); % D2

elseif isequal(S, [0 1 1 0])

msgblk(3) = ~msgblk(3); % D3

elseif isequal(S, [0 1 0 1])

msgblk(4) = ~msgblk(4); % D4

end

% If only one syndrome bit is 1 → parity bit error, no correction needed for msgblk

end

A graph of a graph showing a graph of transmission

AI-generated content may be incorrect.

**(Commit the revised codes to GitHub. )**

1. Based on your observations, which coding scheme performs the best? **Explain why**.

Block coding strikes the best balance between error correction capability and transmission efficiency. It corrects errors with minimal overhead, making it ideal for noisy channels like the one stimulated in the task.

***Fill in the answers, commit the revised codes to GitHub***

***and Show your result to the TA.***

**----------------------------------End-----------------------------------**