1 Problem Statement

To develop an algorithm for image compression, following the JPEG's algorithm to effectively reduce the file size of digital images while maintaining a visually acceptable level of quality. The algorithm should utilize a "lossy" compression approach to achieve significant data reduction and ensure that the compressed data can be efficiently stored and later decompressed to reconstruct a recognizable image.

2 Implemented Algorithm

2.1 Compression Algorithm for Gray Scale Images

1. The quantization matrix corresponding to the given quality factor `q` was computed.
2. The image was padded along its width and height to ensure both dimensions were multiples of 8, using edge values for padding. This allowed the image to be divided into 8×8 patches.
3. For each 8×8 patch:
   (a) The Discrete Cosine Transform (DCT) matrix was computed using MATLAB's predefined `dct` function.
   (b) The DCT coefficients were quantized by element-wise division with the quantization matrix.
   (c) The quantized DCT coefficients were traversed in a zigzag order:
      i. The first element (DC coefficient) was stored in an array of DC values from all patches.
      ii. The remaining elements (AC coefficients) were stored in a separate array of AC values from all patches.
4. Encoding of coefficients:
   (a) The DC values array was Huffman encoded using MATLAB's `huffmanenco` function, producing an encoded array and a Huffman dictionary for decoding.
   (b) The AC values array underwent Run-Length Encoding (RLE), yielding two arrays: one containing values and the other their corresponding run lengths.
   (c) The RLE-encoded AC values were then Huffman encoded separately using MATLAB's `huffmanenco` function, producing an encoded array and a Huffman dictionary for decoding.
5. The following data was stored in a `.mat` file:
   (a) Original image dimensions (`org_height`, `org_width`).
   (b) Padded image dimensions (`height`, `width`).
   (c) Huffman-encoded DC values (`dc_encoded_data`) and their Huffman dictionary (`dc_huffman_table`).
   (d) Huffman-encoded AC values (`encoded_ac_value`) and their Huffman dictionary (`ac_value_huffman_table`).
   (e) AC Run-Length values (`ac_run_length`).

2.2 Decompression Algorithm for Gray Scale Images

1. The `.mat` file was read to load the following:

   (a) Original image dimensions (`org_height`, `org_width`).

   (b) Padded image dimensions (`height`, `width`).

   (c) Huffman-encoded DC values (`dc_encoded_data`) and their Huffman dictionary (`dc_huffman_table`).

   (d) Huffman-encoded AC values (`encoded_ac_value`) and their Huffman dictionary (`ac_value_huffman_table`).

   (e) AC Run-Length values (`ac_run_length`).

2. The DC coefficients were decoded using the `HuffmanDeco` function and the Huffman dictionary (`dc_huffman_table`), retrieving the original DC values.

3. The AC coefficients were decoded using the `HuffmanDeco` function and the Huffman dictionary (`ac_value_huffman_table`), retrieving the RLE-encoded AC values.

4. Run-length decoding was applied to the AC coefficients, reconstructing the original sequence of AC coefficients for each patch.

5. The DC coefficients and decoded AC coefficients for each patch were combined.

6. The coefficients were rearranged back into 8×8 matrices by reversing the zigzag traversal.

7. The quantized DCT coefficients were de-quantized by multiplying each 8×8 matrix element-wise with the quantization matrix.

8. The Inverse Discrete Cosine Transform (IDCT) was applied using MATLAB's predefined `idct` function to convert the frequency-domain data back into the spatial domain for each patch.

9. The 8×8 patches were combined to reconstruct the padded image.

10. The padded image was cropped to the original dimensions (`org_height`, `org_width`), producing the final decompressed image.