In this project, you will be required to implement Huffman encoding algorithm.

Step 1: Open/Read an image and save it as a raw file

In this step, you will write Python code that opens an image as a NumPy array (you can use any library of your choice), then save the image as a raw file.

One way to do that would be to flatten the image (reshape it and make it a single row), then convert each number (from 0 to 255) to an 8-bit (one byte) long binary representation of the number:

```
f = open("out.txt", "wb")
raw_data = bytes([4, 100, 200, 44, ...])        # Flattened image
f.write(raw_data)
f.close()
```

Of course, you will need to also store the original dimensions of the image, so you may want to create another file that only contains the dimensions/shape of the image.

Notes:

- The size of the file (in bytes) needs to be identical to the number of pixels in your image, since each pixel will reserve a single byte
- You can read the image as grayscale, or you can read it as RGB then convert it to grayscale

Step 2: Create the Huffman-encoded image

In this step, you will generate (and save) the Huffman encodings for the image, and the Huffman-encoded image. Refer to the set of slides posted on Canvas for the Huffman encoding algorithm.

The output of this step is two files. The first one contains the encodings for each pixel intensity level, something that may look like this:

```
0: 101
1: 110
2: 10010
3: 11101
...
```

You're free to save the encodings however you like, as long as the decoder (Step 3) can use these encodings to regenerate the image.

The second output file of this step would be the encoded image. For example, let's say that the intensities of the first four pixels in the original image were 2, 3, 0, 1; based on the encodings provided above:

2, 3, 0, 1 should be encoded (in binary) as 1001011101101110

Notice here that we have 16 bits, which means that the space stored should only be two bytes (for these four pixels).

Step 2: Decoding the (Huffman) encoded image

In this third and last step, you will need to write Python code that opens the two files generated in the second step, and using the file containing the encodings, you will build the Huffman tree (which you had to create in the second step).

After building the Huffman tree, you would read the sequence of bits stored in the other file, and use the Huffman tree to replace each code with its intensity value (refer to the set of slides posted on Canvas for an example). You may want to store all these intensity values in a list (or a NumPy array).

Finally, read the file containing the dimensions of the image (created in Step 1), reshape the image accordingly, then save it.