Homework: 12
Huffman Coding on Grayscale Images

Md. Al-Amin Babu
(ID: 2110676134)

October 10, 2025

Code Link

Repository: Huffman coding(GitHub Link)
Pipeline used: OpenCV → grayscale → flatten → Huffman over pixels.

1 What is Huffman Coding

Huffman coding is a *lossless* entropy-coding technique that assigns **shorter bit codes** to *more frequent* symbols and **longer codes** to *rarer* symbols while keeping the code *prefix-free* (no code is a prefix of another). For a grayscale image (values $0 \dots 255$), we treat each pixel as a symbol and encode the flattened stream. If a symbol occurs with probability p_i, the ideal code length trends toward $-\log_2 p_i$ bits; thus a skewed histogram yields fewer average bits per pixel than the raw 8 bpp.

2 My Implementation Experience

Pipeline I used

1. Read the image in `cv2.IMREAD_GRAYSCALE` and flatten to a 1D list.
2. Count frequencies with `Counter`.
3. Build the Huffman tree using a min-heap (`heapq`): repeatedly pop the two least frequent nodes, merge, and push back.
4. Generate codes via DFS (left → 0, right → 1; single-symbol fallback: code "0").
5. Replace each pixel by its code to get the bitstream; report Original bits ($N \times 8$), Compressed bits, and Compression ratio.

Interesting parts

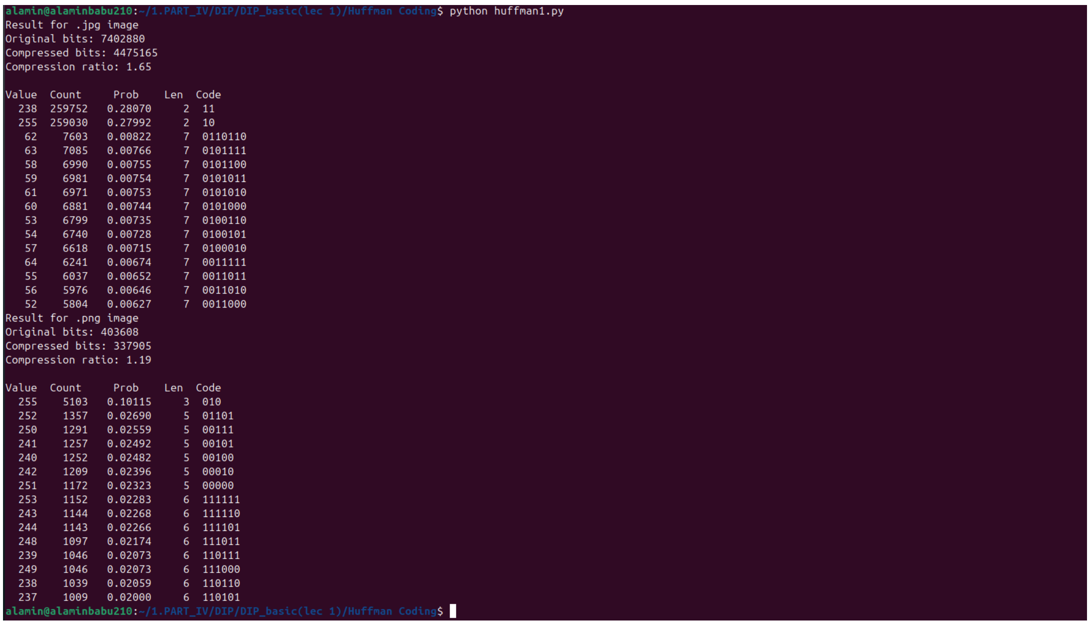
- **Histogram → code length:** dominant gray levels automatically receive very short codes (sometimes 2–3 bits), which makes the theory visible.
- **Compact but complete:** with `heapq` and a small DFS you get a full, prefix-free encoder in only a few dozen lines.

Difficult parts

- **Ties & tree shape:** equal frequencies lead to multiple valid trees; patterns can differ but lengths remain similar. Keeping heap nodes comparable avoids errors.
- **Bitstream details:** a printable bit-string is fine for reporting; a real file needs byte padding and a stored codebook for stand-alone decoding later.
- **Edge case:** if only one intensity exists, assign its code as "0" so the encoder still works.

3 Results (Images)

(a) Input grayscale image.



```
alamin@alaminbabu210:~/1.PART_IV/DIP/DIP_basic(lec 1)/Huffman Coding$ python huffman1.py
Result for .jpg image
Original bits: 7402880
Compressed bits: 4475165
Compression ratio: 1.65

Value  Count     Prob    Len  Code
  238  259752   0.28070     2  11
  255  259030   0.27992     2  10
   62    7603   0.00822     7  0110110
   63    7085   0.00766     7  0101111
   58    6990   0.00755     7  0101100
   59    6981   0.00754     7  0101011
   61    6971   0.00753     7  0101010
   60    6881   0.00744     7  0101000
   53    6799   0.00735     7  0100110
   54    6740   0.00728     7  0100101
   57    6618   0.00715     7  0100010
   64    6241   0.00674     7  0011111
   55    6037   0.00652     7  0011011
   56    5976   0.00646     7  0011010
   52    5804   0.00627     7  0011000
Result for .png image
Original bits: 403608
Compressed bits: 337905
Compression ratio: 1.19

Value  Count     Prob    Len  Code
  255    5103   0.10115     3  010
  252    1357   0.02690     5  01101
  250    1291   0.02559     5  00111
  241    1257   0.02492     5  00101
  240    1252   0.02482     5  00100
  242    1209   0.02396     5  00010
  251    1172   0.02323     5  00000
  253    1152   0.02283     6  111111
  243    1144   0.02268     6  111110
  244    1143   0.02266     6  111101
  248    1097   0.02174     6  111011
  239    1046   0.02073     6  110111
  249    1046   0.02073     6  111000
  238    1039   0.02059     6  110110
  237    1009   0.02000     6  110101
alamin@alaminbabu210:~/1.PART_IV/DIP/DIP_basic(lec 1)/Huffman Coding$
```

(b) Result for jpg and png image after huffman coding.

Figure 1: Experiment images.

4 Effect on My JPG and PNG Images

Important. I compress the *decoded grayscale pixels*, not the raw file bytes; therefore results depend on the image histogram rather than JPG/PNG internals.

Observed console results

JPG image
Original bits: 7 402 880
Compressed bits: 4 475 165
Compression ratio: **1.65**×

PNG image
Original bits: 403 680
Compressed bits: 337 905
Compression ratio: **1.19**×

Why JPG > PNG here

- **JPG (decoded) histogram is highly skewed:** a few gray values (e.g., 238, 255) dominate, so they receive 2–3 bit codes ⇒ strong reduction (about **1.65**×).
- **PNG (decoded) histogram is broader:** many nearby values share the mass; codes are 3–6 bits but distributed across many symbols ⇒ milder reduction (about **1.19**×).
- **Visual fidelity:** Huffman is lossless; reconstructed images match their inputs.

5 Conclusion

Per-pixel Huffman coding clearly shows that *frequent* intensities get short codes, reducing the average bits per pixel. In my tests, the decoded JPG yielded **1.65**× compression (high skew), while the PNG yielded **1.19**× (broader histogram). Modern formats (PNG/JPEG/WebP/AVIF) combine prediction/transform and entropy coding for stronger practical compression, but pixel-wise Huffman is an excellent, explanatory baseline.