.

Assignment 7 – Huffman Coding

Airi Kokuryo

CSE 13S – Winter 24

Purpose

The purpose of this report is to understand compression using the C programming, as well as discussing questions important in thinking about the Huffman coding.

Questions

- Describe the goal of compression. (As a hint, why is it easy to compress the string "aaaaaaaaaaaaaaa")

  The goal of compression is to reduce the size of a file or data in order to save storage space and decrease transmission time. Compression is effective when there are repetitive patterns or redundancy in the data. In the case of the string "aaaaaaaaaaaaaaa," compression is easy because it consists of repeating the same character, and compression algorithms can represent such patterns more efficiently.

- What is the difference between lossy and lossless compression? What type of compression is Huffman coding? What about JPEG? Can you lossily compress a text file?

  Lossless compression retains all the original data after compression and decompression, ensuring no loss of information. Huffman coding is an example of lossless compression. Lossy compression sacrifices some data to achieve higher compression ratios. JPEG is an example of lossy compression commonly used for images. While text files can be lossily compressed, it's generally not recommended, as it could result in the loss of essential information.

- Can your program accept any file as an input? Will compressing a file using Huffman coding make it smaller in every case?

  The program should be able to accept any file as input. Compressing a file using Huffman coding doesn't guarantee a reduction in size for every type of file. The effectiveness of Huffman coding depends on the presence of repetitive patterns or redundancy in the data.

- How big are the patterns found by Huffman Coding? What kind of files does this lend itself to?

  Huffman coding identifies and exploits patterns at the bit level. It is effective for files with repetitive structures or symbols, such as text files, where certain characters occur more frequently than others.

- Take a picture on your phone. What resolution is the picture? How much space does it take up in your storage (in bytes)?

  The picture on my phone has a resolution of 3024×4032 and is 4,300,800 bytes(4.1MB).

- If each pixel takes 3 bytes, how many bytes would you expect the picture you took to take up? Why do you think that the image you took is smaller? // // If I multiply 3024×4032×3 it would be 36,595,968 bytes in total, but this is much larger than the actual size. The actual size may be smaller due to the

compression applied, because unlike texts, pictures and videos may be compressed and not make much change in human vision.

- What is the compression ratio of the picture you just took? To get this, divide the actual size of the image by the expected size from the question above. You should not get a number above 1.

  The compression ratio is 36,595,968/4,300,800 = 0.1174.

- Do you expect to be able to compress the image you just took a second time with your Huffman program? Why or why not?

  It depends on whether the image has already been compressed and the type of compression applied. If it's a JPEG image, further compression with Huffman coding may not be very effective, as JPEG already employs lossy compression.

- Are there multiple ways to achieve the same smallest file size? Explain why this might be.

  Yes, there can be multiple ways to achieve the same smallest file size, especially in lossless compression. Different compression algorithms or implementations may yield similar results in terms of reducing redundancy and patterns in the data.

- When traversing the code tree, is it possible for a internal node to have a symbol?

  No, an internal node in a Huffman tree does not have a symbol. Symbols are only associated with leaf nodes.

- Why do we bother creating a histogram instead of randomly assigning a tree?

  Creating a histogram helps identify the frequency of each symbol in the input data. This information is crucial for constructing an optimal Huffman tree, as symbols with higher frequencies should have shorter codes. Randomly assigning a tree would likely result in less efficient compression.

- Relate this Huffman coding to Morse code. Why does Morse code not work as a way of writing text files as binary? What if we created more symbols for things like spaces and newlines?

  Huffman coding and Morse code are similar in that they both use variable-length codes, but Huffman codes are optimal in terms of minimizing the average code length for each symbol. Morse code does not directly map to binary and lacks the efficiency of Huffman coding. Creating more symbols for spaces and newlines in Huffman coding could improve efficiency, as frequent symbols are assigned shorter codes.