CSE 13S: Assignment 6 Write-Up

Johnny Li

March 12, 2023

1 Introduction

Data compression and data security have become increasingly important fields in software engineering, shaping the current computing landscape. From mobile data to IoT to cloud computing, these two areas are essential components of our digital lives.

Data compression algorithms work to reduce the number of bits needed to represent data, leading to smaller file sizes, faster transfer speeds, and decreased storage costs. There are two main types of compression algorithms: lossy and lossless. Lossy algorithms are more efficient in compressing data than lossless algorithms, but they sacrifice data quality by discarding some information. Lossy compression is commonly used for audio and video files, where some loss of data can go unnoticed by the user. On the other hand, lossless algorithms preserve all information and are ideal for files that must maintain their integrity, such as text documents, binaries, and source code.

An interesting example of lossy versus lossless compression can be seen with YouTube videos. The video streaming site uses lossy compression, and during consistent and relatively still portions of the video, the quality and frame rate are normal. However, when something like snowfall or confetti appears on the screen, the compression algorithm struggles to keep up, resulting in a decrease in video quality. This is because the lossy compression algorithm discards some data, leading to a loss of quality when there is an increase in complexity.

Being our last assignment, I expected something more from this. Assignment 6 somehow felt hollow, unlike previous assignments which pushed me to grow as both a student and programmer and learn new things, Assignment 6 didn't feel like the crowning achievement for CSE 13S nor did it push me to new heights. Instead, it felt like a continuation of Assignment 5. I'm not complaining too much since this felt easier than the previous assignment and allowed me to relax a tad bit prior to finals.

With this assignment, I felt like I solidified the ideas and concepts learned from Assignment 5. For most of this class, it felt like you had to learn new things while just two days ago, you were struggling with another concept which you

still hadn't fully grasped yet. And although it seemed like hell then, I feel more confident in my programming skills now than prior to taking this course. I'm using more exotic libraries now and pointers have become easier to grasp. So overall, I haven't learned that many new topics from this Assignment but rather reinforced existing ones from previous assignments.

2 Lempel-Ziv Compression

Lempel-Ziv (LZ) compression is a family of lossless data compression algorithms that includes LZ77 and LZ78. The LZ77 algorithm, published in 1977 by Abraham Lempel and Jacob Ziv, is based on the idea of replacing repeated patterns in data with pairs that consist of a code and a symbol. The code is an unsigned 16-bit integer, and the symbol is an 8-bit ASCII character.

The LZ78 algorithm, published in 1978, is a variant of LZ77 that uses a different approach to constructing the dictionary of patterns. Both algorithms work by building a dictionary of previously seen patterns, and then replacing repeated patterns with a pair consisting of the index of the pattern in the dictionary and the next symbol in the input stream. The compression ratio of LZ algorithms depends on the frequency of repeated patterns in the input data.

The LZ77 algorithm uses a sliding window to keep track of previously seen patterns. The window is a fixed-size buffer that contains the most recent part of the input stream. The algorithm scans the input stream, looking for patterns that match the contents of the window. When a match is found, the algorithm outputs a pair consisting of the index of the matching pattern in the window and the next symbol in the input stream. The algorithm then moves the window forward by the length of the matching pattern, so that it includes the next part of the input stream.

The LZ78 algorithm uses a different approach to building the dictionary of patterns. It starts with an empty dictionary, and then scans the input stream, looking for the longest prefix of the input that is already in the dictionary. When a new prefix is found, it is added to the dictionary, and a pair consisting of the index of the prefix in the dictionary and the next symbol in the input stream is output. The algorithm then continues scanning the input stream, adding new prefixes to the dictionary as they are found.

3 Compression Relationship with Entropy

The efficiency of the Lempel-Ziv Compression algorithm, and most compression algorithms, heavily depend on the entropy of the data; entropy being a measure of the randomness of a dataset, similar to entropy from chemistry. When enthropy is high, it's more difficult for compression algorithms to create order since there are little to no patterns for the algorithm to exploit. In contrast, when entropy is low, it's easier to reduce the size of the data since the data

tends to be more organized and there are more patterns for the algorithm to rely upon.

For example, when compressing a text file, which tend to have relatively low entropy, compression algorithms like LZ77 and LZ78 can achieve high compression ratios due to the patterns and structures inherent in language. However, when compressing something like an encrypted file, which inherently have high entropy due to nature of encryption, algorithms will not be able to exploit patterns so compression ratios will be low. Images and audio files, like encrypted file, have much higher entropy than text files due to the complex and random nature of the data, making high compression ratios more difficult to achieve.

The size of the data is another factor that comes into play since larger data sets have a greater chance of patterns and redundancies, resulting in higher compression ratios. However, this tends to be a double-edged sword as larger files also require more computational power and memory, which can decrease overall efficiency.

Ironically, compression algorithms can increase file sizes if entropy is too high or the algorithm isn't suitable for the specific data type. Thus, choosing the appropriate compression algorithm for a task is crucial as one optimized for video/audio files may not be suitable for text files and vice versa.