Information Theory

To compress the given file, we have used three methods: Prediction by partial matching (PPM), Huffman coding algorithm and Zip compression. We wanted to compare between them and check which algorithm has the best compression ratio.

First, we will introduce shortly each used algorithm and the main idea of its implementation. Second, we will compare between all three compression algorithms and determine which one is best for our file (dickens).

What distinguishes our project from other projects, is that we have used a compression method that hasn’t been taught in class (PPM algorithm that was mentioned above). While we were reading articles and doing some previous research on compression methods, we encountered this algorithm. As can be seen later in our work, it has given us the best results, compared to other algorithms. In addition, we have implemented not one compression algorithm, but three (PPM, Huffman and Lempel-Ziv), in order to compare between them and see which one gives us the best compression for the given file.

1. Compression Methods

Prediction by Partial Matching (PPM)

PPM is an adaptive statistical data compression technique based on context modeling and prediction. PPM models use a set of previous symbols in the uncompressed symbol stream to predict the next symbol in the stream.

Predictions are usually reduced to symbol rankings. Each symbol (a letter, a bit or any other amount of data) is ranked before it is compressed and, the ranking system determines the corresponding code word (and therefore the compression rate).

Arithmetic encoding takes a sequence of symbols as input and gives a sequence of bits as output. The intent is to produce a short output for the given input. Each input yields a different output, so the process can be reversed, and the output can be decoded to give back the original input.
In our implementation, a symbol is a non-negative integer. The symbol limit is one plus the highest allowed symbol. For example, a symbol limit of 4 means that the set of allowed symbols is {0, 1, 2, 3}.

Huffman Coding:

Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols.

Our implementation of Huffman coding makes a frequency dictionary sorted from low to high value in order to decide which characters need to be merged. It also uses a heap to represent the tree that is built in Huffman coding. The main functions are “compress” that compresses the given text file and outputs its compressed version, and “decompress” that decompresses the compressed text file and outputs a text file that is equal to the original.

<u>Zip:</u>

Because we have never done a zip compression with python, and wanted to compare the other methods to the most familiar and used one to compress files, we also did a zip compression to the file, with a known package of python.

2. Results

	Original size	Compressed size	Compression ratio
PPM	30 MB	8.95 MB	3.35
Huffman coding	30 MB	16.96 MB	1.77
Zip	30 MB	12 MB	2.5

As can be seen from the results, PPM model is the best compression algorithm for the given text file (‘dickens.txt’).