Lea Setruk 345226179
Aviva Shneor Simchon 317766731

Information Theory
File Compression

Comments are also in the code itself

Our data
A text file (for example : 'dickens.txt')

Our purpose

We need to compress the file and to be able to decompress it using algorithms that we learned in class.
The size of the file is initially 30.6 Mo. The compression should get to 12Mo.

Our algorithms

We tested several algorithms and several combination of algorithm until we achieved our purpose.

We first tried to compress the file ('dickens.txt') using LZ'78. After several tries, the code we tried to implement did not succeed to decompress the file. Indeed, it couldn't decode the number characters that was mixed with the code of the algorithm. We tried to fix it but didn't succeed.

We then implemented LZW algorithm. It gave us very good result. We could get with this algorithm a compression to 12.6Mo.

We wanted to improve it. To do so, we added Huffman coding.
We implemented a combination of both algorithm.
Huffman improves the compression, but it's not that significant. The combination compresses to 12.5Mo. Which is a great compression.

We then, implemented the decompression. First, we decompress using Huffman to decode the final file to get the file that has initially been compressed by LZW. Then, we decompress this one with LZW decompression.
Then, we get our initial text file back (decompressed).

What do those algorithms ?

<u>LZW :</u>

Lea Setruk 345226179
Aviva Shneor Simchon 317766731

LZW compression works by reading a sequence of symbols, grouping the symbols into strings, and converting the strings into codes. Because the codes take less space than the strings they replace, we get compression.

LZW compression uses a code table (dictionary). We are working with **ascii** table. Codes 0-255 in the code table are always assigned to represent single bytes from the input file.
To begin, the dictionary contains only the first 256 entries of ascii table, with the rest of the table being blanks.
Then, LZW identifies repeated sequences in the text, and adds them to the code table.
The dictionary keeps a correspondence between the longest encountered words and a list of code values. The words are replaced by their corresponding codes and so the input file is compressed.

(The efficiency of the algorithm increases as the number of long, repetitive words in the input data increases.)

The decoding program that uncompresses the file is able to build the table itself by using the algorithm as it processes the encoded input.
It's taking each code from the compressed file and translating it through the code table to find what character or characters it represents.

<u>Huffman coding</u> :
Huffman coding provides an efficient code by analyzing the frequencies that certain symbols appear in a text. Symbols that appear more often will be encoded as a shorter bit string while symbols that aren't used as much will be encoded as longer strings.
Huffman coding is working such that the code assigned to one character is not the prefix of code assigned to any other character.
Huffman coding works by using a frequency-sorted binary tree to encode symbols.

To do so, we need to build a Huffman Tree from input characters, traverse it and assign codes to characters.

We create a binary tree.

We pick two trees with the smallest frequency and combine them into a new tree with a new parent node with the frequency of both ot its children (the addition). We iterate until there is no more trees to merge.

Every symbol must have its code. We start at the root node. For each left arc, we add 0 to the end of the code and for each right arc, we add 1 to the end of the code.