# LZW COMPRESSION

a project

*by: Roman Gehtberg*
*and Yacov Mateo*

## Introduction:

**T**his project is a representation of LZW compression algorithm. Based on original algorithm with addition of Graphical User Interface and 8-1 bit file write algorithm created by us.

**L**empel–**Z**iv–**W**elch (**LZW**) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. Two commonly-used file formats in which LZW compression is used are the GIF image format served from Web sites and the TIFF image format. LZW compression is also suitable for compressing text files. A particular LZW compression algorithm takes each input sequence of bits of a given length (for example, 12 bits) and creates an entry in a table (sometimes called a "dictionary" or "codebook") for that particular bit pattern, consisting of the pattern itself and a shorter code. As input is read, any pattern that has been read before results in the substitution of the shorter code, effectively compressing the total amount of input to something smaller.

Our type of LZW algorithm compresses well a text file formats, somewhat well an executable files and bitmap images and doing pretty bad at media files such as .wav .mp3 and jpeg, in some cases - making them bigger.

Any file format could be taken as an input while the output will be compressed .lzw file.

## *Implementation:*

Project is implemented in Java language (jre 1.8) and consists of a 5 different class files. It's also exported as java executable archive for standalone use. Interface is build on a Java's Swing API toolkit with minor use of an older AWT toolkit.

The LZW Compression algorithm we used is based on a HashMap Dictionary, as a hashmap being the best data structure for storing a pairs of values, having O(N) lookup time . Every word we encode is stored as a string with it's entry being an integer. That way we can accumulate an array of integers representing an encoded strings which are then sent to an output.
Dictionary itself starts with a size of 256 as it is filled by encoded 1 byte sequences, (each sequence represent a possible byte state of an uncompressed data stream) after which added all new encoded entries, with a dictionary size being unlimited*[1].

Encoded sequence is written to a file by using our '8-1 file write' algorithm.
Algorithm takes an encoded integer representation and tries to fit it to a minimum bits by adding every 8 bits a control bit. Control bits are representing if integer consists of 1,2 or 3 bytes - that way we can store 0-255 integers as 8+1 bits sequence, 256-65535 as 16+2 bits and 65536-16777216 as 24+3 bits. (Here of course we could add even more, but we decided that 16kk entries is quite enough for a files up to 500MB). Doing so we making sure we can decrypt the bits into proper integers on file decompression. All the bits are concatenated as string representations and then split by 8 to make a byte array that could be written to file altogether.

On decompression we concatenate all the bytes as string of bit representations together and do the reverse step of extracting bits 8+1 by 8+1. Now we have an arraylist of integers that would be compared with a new dictionary we rebuild and decoded chars are extracted to a byte array that written to a file.

GUI is implemented using Java Swing toolkit and mvc model. 'View' contains the information about GUI elements, their positions and properties, while Controller is set to contain methods connected with actions on GUI that implemented in worker, who's main purpose is to do the compression\decompression process itself.

GUI interface consist of six buttons, seven labels and a progress bar.
On each operation there's few buttons that corresponds with it. For example in order to compress – you should click buttons: "Choose the file to compress", "Choose to what file it should be saved" and then the "Compress" button itself.

## *Tests and conclusions:*

After testing a dozen of files we concluded that this algorithm best fits for compressing a text files, being able to squeeze regular text up to 1\3 of it's original size. (Similar to .zip compression rates).
While there are some different files like executables (.exe .bin), bitmap images and general data files (.dat) being compressed on some rate, there are quiet a lot files that doesn't undergo this compression at all. Such as audio\video files in different formats, pictures in formats like .jpeg etc…

And here is a table of some compression examples:

| *File Name* | *Before* | *After* | *Compression Rate* |
|---|---|---|---|
| london_in_polish_source.txt | 329 KB | 126 KB | 2.61 |
| OnTheOrigin.txt | 1027 KB | 451 KB | 2.27 |
| bible.txt | 3953 KB | 1585 KB | 2.49 |
| setupapi.offline.20180915_00333.log | 5748 KB | 897 KB | 6.40 |
| Red_Flowers.bmp | 1372 KB | 917KB | 1.49 |
| HDTunePro.exe | 1216 KB | 754 KB | 1.61 |
| WizTree64.exe | 7887 KB | 4823 KB | 1.63 |
| ProjectTips.pdf | 339 KB | 406 KB | -1.19 |
| 1.wav | 6128 KB | 8173 KB | -1.33 |