Information Theory

Lab 7: Lossless compression – LZW algorithm

Description of the tasks

All tasks realized during classes are `.pdf` files formatted similarly as this document. The tasks will be of different kinds. Every task will be appropriately marked:

- Tasks to be realized during classes are marked with $\square$ – you won't get points for them, but you still need to do them.
- Pointed tasks to be realized during classes are marked with $\diamond$ – you need to do them during class and show to your teacher, and in the case you don't manage to do so (or are absent) they become your homework ($\star$).
- Homeworks are marked with $\star$ – they also have assigned a number of points, and you need to deliver them to your teacher before a deadline (usually before the next class).
- You may use any programming language you like for the programming tasks, but you are limited to only the standard library of that language and libraries widely accepted as standard.

Objective

Learn about the popular LZW (Lempel–Ziv–Welch) compression algorithm, used for example in the GIF format.

Preparation

- For this task will be needed text corpuses and an image, which can be downloaded from: http://www.cs.put.poznan.pl/ibladek/students/timkod/lab_lzw.zip.

1 LZW *10pt*◇

Task

The lossless compression method LZW (Lempel-Ziv-Welch) was published by Terry Welch in 1984. It was an improvement over the LZ78 algorithm proposed by Abraham Lempel and Jacob Ziv in 1978. LZW is a dictionary-based compression method, which means that it stores often repeating sequences in a dictionary, and references to the dictionary are placed in the text instead of the original sequences. In LZW, the dictionary during both compression and decompression is created on the fly.

LZW method is easy do program and usually gives relatively good compression. It doesn't give guarantees for compression (especially, when there is no redundancy in the data), and in some cases it can even make the files bigger. It is used for example in the GIF format, and optionally can be used in PDF and TIFF; it is also used by the UNIX tool *compress*.

Create your own implementation of LZW.

- Test your implementation on "norm_wiki_sample.txt", "wiki_sample.txt", and some short text examples. You can show the results as a pure text, with codes going beyond ASCII represented for example as '*256', etc. ***(5pt)***
- Test your implementation on the image "lena.bmp". You should produce the binary result, as LZW usually does. ***(5pt)***

If you want, you can skip one of the points above and have the task marked as done with 5/10.

Sources

- https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch
- https://en.wikipedia.org/wiki/Huffman_coding
- http://www.inference.org.uk/itprnn/book.pdf