Lab 9

Compression

[Compulsory]

Author: Ragnar Nohre 2017, modified and moved to English Johannes Schmidt 2018-2022

This lab deals with data compression of text, the entropy notion and source memory. It also treats the difference between 8 bit ASCII encoding and UTF-8 encoding.

9.1 Different text encodings

The following exercise deals with ASCII, LATIN-1, and UTF-8 encodings.

1. We already know that a text can be converted to a *byte*-array. Feed the following to your python engine in interactive mode, and interpret the result.

```
>>> txt = 'ABC abc'
>>> b = bytearray(txt, 'ASCII')
>>> len(txt)
>>> len(b)
>>> b[0]
>>> b[1]
>>> b[2]
>>> b[3]
>>> b[4]
>>> b[5]
>>> b[6]
```

Which *byte*-values do the 7 different symbols correspond to (ABCabc and space)?

2. Does the ASCII-encoding include Swedish symbols? *Try* to convert the string 'ÅÄÖ' to an ASCII-*byte*-array, and see what the python interpreter responds.

3. Since ASCII is a 7-bit code it has only place for 128 different symbols. There are different extensions of ASCII that use all 8 bits of a *byte* which thus have place for 256 different symbols. *LATIN-1* is such an example. Try to convert the string 'ÅÄÖ' to a *byte*-array with encoding 'LATIN-1'. How are *Å, Ä, Ö* encoded?

4. UTF-8 is another extension of 7-bit ASCII. UTF-8 is a variable length encoding. Certain symbols are *one* byte long, others are *two* or more bytes long. Convert the string 'ÅÄÖ' to a *byte*-array with encoding 'UTF-8'. How are *Å, Ä, Ö* encoded?

9.2 Compression

In Canvas you find the file *exempeltext.txt (for MAC use exempeltextMac.txt)*. Copy it to your working folder. Have a look at the file's contents so you see what language it is written in etc. Write a python program that does the following:

1. Read and investigate the file...

   (a) Open the file and read its contents into a variable `txt` (of type *string*).

   (b) Convert the string to a *byte*-array with
   `byteArr = bytearray(txt, "utf-8")`

   (c) How many symbols does the string contain? How many bytes does the byte-array contain? Explain differences. Answer!

2. Calculate statistics and entropy...

   (a) Write a function `makeHisto(byteArr)` that retruns a histogram (a list of length 256) which indicates how many times each number/bit-pattern (0-255) occurrs in `byteArr`.

   (b) Write a function `makeProb(histo)` that, given a histogram, returns a *probability distribution*. A probability distribution is a list (of same length as the given histogram) that contains a *normalized histogram*, that is, all numbers sum up to 1.0.

(c) Write a function `entropi(prob)` that returns the probability distribution's entropy, $H = \sum_i P(i) \log \frac{1}{P(i)}$, where log is the base-2 logarithm. You need to avoid *division by zero*. It is therefore important to know that $p \log \frac{1}{p} \to 0$ when $p \to 0$.

(d) Down to how many bytes should it be possible to compress the byte-array `byteArr` if we treat it as a memory-free source (i.e., we do not exploit statistical redundancy) but use an optimal encoding? Answer!

3. Copy the text and shuffle the copy...

(a) Import the python module `random`.

(b) Make a copy (`theCopy`) of `byteArr` and shuffle it with `random.shuffle(theCopy)`. The copy contains now the same numbers as the original, but in random order (as if it was generated by a memory-free source).

(c) Verify that you have not erroneously shuffled also `byteArr`!

4. Below we shall zip-compress `byteArr` and `theCopy`. The purpose is to investigate how well the zip-algorithm can compress our byte-arrays. Theory says it should only be possible to compress data below its entropy if there is a source memory that we can exploit...

(a) Import the python module `zlib`.

(b) The statement `code = zlib.compress(theCopy)` compresses the copy and returns the zip-code as a new *byte*-array. Do it.

(c) How long is the zip-code measured in bytes? How long is it measured in bits? How many source symbols does `theCopy` contain? Hence, down to how many bits/symbol has the zip algorithm managed to compress `theCopy`? Answer!

(d) Repeat the above with the unshuffled byte-array `byteArr`. Down to how many bit/symbol can the zip-algorithm compress this array? Answer!

(e) Now you have three different numbers of bits/symbol: (a) the data source's entropy, (b) the zlib-encoding of `theCopy`, and (c) the zlib-encoding of `byteArr`. Which one is the smallest number? Which one is the highest number? Explain why! Answer!

5. Zip repetitive text...

(a) Put an short text in a variable, and put the same text, but 10 times repeated in a second variable, e.g.

```
t1   = """I hope this lab never ends because
          it is so incredibly thrilling!"""
t10 = 10*t1
```

(b) Down to how many bytes can zlib compress the first and second string, respectively? Answer!

(c) The string `t10` contains 10 times more symbols than `t1`, but is also its zip-code ten times longer than `t1`'s? Explain why/why not! Answer!

9.3 Examination

Submit your python code on Canvas and prepare answers to the questions marked with *Answer!*. Present then your code and answers to your lab assistant.