Source Encoding and Compression
Spring 2016

Exercise tasks

The tasks are solved independently and submitted to the teacher before taking the examination. Submissions can be done either in paper form or email attachment. The tasks should be submitted as one bundle (not one task at a time). The minimum requirement is that five out of the ten tasks, given below, are solved acceptably. In case of difficulties, contact the teacher for assistance.

The tasks are mostly applications of coding algorithms to test data. All students solve the same tasks, so, in order to make the solutions individual, the following rules for generating the test data will be used.

(A) **Numeric test value**: the sum of 'day' and 'month' of your birthday. For example, 27^{th} of May gives the value 27+5 = 32.

(B) **Character string**: your full name (first, middle, and last name) and home city concatenated to a single string, e.g. "JOHANN SEBASTIAN BACH LEIPZIG", including spaces. Use only upper-case letters.

(C) **Binary test string**: Throw a dice (Finnish: noppa) *n* times. Values 1-5 produce symbol 0, and only value 6 produces symbol 1 to the string.
E.g. {4, 2, 1, 6, 3, 6, 4, 1} is mapped to the binary string "00010100". (If you do not have a dice, use some other random number generator for values 1-6).

Task definitions (10):

1. Apply (a) **α-coding**, (b) **β-coding**, c) **γ-coding**, d) **Zeckendorf**-coding to your birthday number (see test value (A)). Compare the code lengths obtained by these four methods.

2. Compute the **entropy** (as bits per character) of the alphabet occurring in your test character string (B), based on the character frequencies in the string.

3. Derive **Shannon-Fano** codes for the characters occurring in string (B), using their frequencies in the string to obtain empirical probabilities. Present the encoded string as a sequence of bits.

4. The same as task 3, but using the **Huffman code**.

5. Derive an **extended Huffman code** for a binary test string (C) of 16 bits, using
   (a) 2-bit blocks,
   (b) 4-bit blocks.
   Compute the code lengths.

6. Apply the basic idea of **arithmetic coding** to a 6-bit binary test string (C) by assuming symbol probabilities P(0) = 5/6 and P(1) = 1/6. Derive the successive (floating-point) intervals for the symbols in your test string. Determine the code length in bits, based on the final interval.

7. Apply the **Burrows-Wheeler transform** to your *family name*. The actual coding is not required.

8. Apply the **implicit dictionary** technique (cf. LZ77) to a 16-bit binary test string (C) as follows. Assume that two artificial symbols (0 and 1) are first added to the front, to prevent non-matches. At a given cursor position, find the longest previous match, encode it, and move the cursor correspondingly. The matches are represented as <length, position> pairs. Encode these numbers with Rice-code (see lecture notes p.11) and compare the total length to the original.

9. Apply the **LZW** method to derive a coding dictionary, based on your test string (B).

10. Take the initial character of your family name, and derive an 8 x 8 bitmap image, illustrating the shape of the character. Value 0 represents white, and value 1 is black. As an example, the character 'X' could be mapped to the following image.

```
1 0 0 0 0 0 0 1
0 1 0 0 0 0 1 0
0 0 1 0 0 1 0 0
0 0 0 1 1 0 0 0
0 0 0 1 1 0 0 0
0 0 1 0 0 1 0 0
0 1 0 0 0 0 1 0
1 0 0 0 0 0 0 1
```

Apply **hierarchical block coding** to the image. Did you get any compression gain?