Lab 9B

Compression, Huffman encoding

[Optional]

Author: Ragnar Nohre, moved to English Johannes Schmidt

This is the third of four optional labs. The optional labs can give together up to 10 *bonus points*. If all regular labs are passed by the general deadline, the accumulated bonus points will count as a bonus on the exam. The scale is that 10 bonus points correspond to 10% of the exam points. This *bonus rule* is only applicable to the first exam (December), and only if the exam is passed without the bonus points.

In order for this lab to qualify for bonus points, you must present and submit your solution before the first exam.

The 10 bonus points are distributed among the optional labs as follows:
Lab8A - 3 points
Lab9A - 1 point
Lab9B - 3 points
Lab11A - 3 points

In this lab you train your skills in python programming and your understanding of Huffman encoding.

5.1 Task 1 - compute average codeword length

In a previous lab you computed the probability distribution and entropy of an example text from Canvas (exempeltext.txt).

Task

Write a program that constructs a huffman code for the same text, base on its probability distribution. Compute also the *average codeword length*. If you do it right, your avarage codeword length should be close to the previously computed entropy.

Hints

1. You can represent a huffman tree in many different ways in python. Here we describe one possibility.

   You can represent a node with the `Node`-class from the previous lab. All nodes store a probability in `n.prio`. A leaf-node stores a *byte*-value in `n.data`. An internal node stores instead two subtrees as a tuple in `n.data`.

   (a) A leaf can be represented by a `Node(p, byte)`-object.

   (b) Internal nodes can be represented by a `Node(p, (t1, t2))`-object, where `t1` and `t2` are two subtrees (i.e. nodes).

   (c) In order to determine whether a node `n` is a leaf or an internal node you can check the data type of `n.data`. If `type(n.data)==int`, it is a leaf, otherwise it is an internal node.

2. Create the huffman tree with the help of a priority queue that is initialised with all the nodes who eventually shall become leaves.

3. Once the tree is constructed, you should be able to compute the desired *average codeword length* with a simple recursion. Do it!

5.2 Task 2, list all codwords

Print a table with all (up to 256) different *byte*s (*octet*s) that occur in the source and their binary encodings. For every *byte* the table should contain:

1. The *byte*-value (number between 0 and 255)

2. ASCII-character in case it is *printable* (when the *byte*-value is between 32 and 127)

3. The binary codeword

4. Number of bits in that codeword (length)

5. The ideal codeword length, that is, $\log_2 \frac{1}{P(x)}$

Below is a part of the desired table:

```
byte= 87 (W) 1110110111100       len=13 log(1/p)=12.3
byte= 89 (Y) 111011010110101     len=15 log(1/p)=14.9
byte= 91 ([) 111011001010        len=12 log(1/p)=11.6
byte= 93 (]) 111011001011        len=12 log(1/p)=11.6
byte= 97 (a) 1011                len= 4 log(1/p)=3.72
byte= 98 (b) 1110101             len= 7 log(1/p)=6.58
byte= 99 (c) 1010011             len= 7 log(1/p)=6.8
byte=100 (d) 10101               len= 5 log(1/p)=4.69
byte=101 (e) 1111                len= 4 log(1/p)=3.63
byte=102 (f) 010011              len= 6 log(1/p)=6.25
byte=103 (g) 00101               len= 5 log(1/p)=5.39
byte=104 (h) 010100              len= 6 log(1/p)=6.2
byte=105 (i) 0000                len= 4 log(1/p)=4.57
byte=106 (j) 10100100            len= 8 log(1/p)=8.0
byte=107 (k) 01000               len= 5 log(1/p)=5.35
byte=108 (l) 0001                len= 4 log(1/p)=4.52
byte=109 (m) 00100               len= 5 log(1/p)=5.42
byte=110 (n) 1000                len= 4 log(1/p)=3.97
byte=111 (o) 01110               len= 5 log(1/p)=5.04
byte=112 (p) 011111              len= 6 log(1/p)=5.92
byte=113 (q) 10100101001001      len=14 log(1/p)=13.9
```

Note that the factual codeword lengths are pretty close to the ideal ones, and that the codewords for common characters (e.g. *a*) are much shorter than for uncommon ones (e.g. *q*).

If you compute the average ($\sum P(x)L(x)$) of the factual codeword lengths, you obtain the average codeword length as desired in task 1. If instead you compute the average of the ideal codeword lengths, you obtain the entropy.

5.2.1 Hints

1. Create a global dict()-object in order to store for every in the source occurring *byte* the corresponding binary codeword.

2. Write a recursive function that traverses the huffman tree and places all leaves' *byte*-values and their codeword in the dict()-object.

3. Iterate over all *byte*-values 0. . . 255. If the corresponding *byte* exists in the dict()-object, print the *byte*-value, etc.

5.3 Examination

Submit your python program on Canvas and present it to your lab assistant.