Assignment number 2
Source codes

Deadline: April, 5

Sources of information. We represent in `Python` a source of information S with a q-letter alphabet Σ as a list containing q two-tuples. Each tuple is a pair `("a",`$w(\mathtt{a})$`)` with $\mathtt{a} \in \Sigma$ an alphabet letter and $w(\mathtt{a})$ a number representing the relative frequency of the letter: they can be

- floating point numbers giving the probabilities assigned to each letter: $w(\mathtt{a}) = P(S = \mathtt{a}) = p(\mathtt{a}) \in [0, 1]$, or otherwise
- positive integers $w(\mathtt{a}) \in \mathbb{N}$ counting the number of times that the letter `a` appears in some text to be compressed.

This gives flexibility to the way of representing a source of information S. The alphabet Σ is the set of first elements of the tuples, and the probability of each letter is always given by:

$$p(\mathtt{a}) = \frac{w(\mathtt{a})}{\sum_{\mathtt{a}\in\Sigma} w(\mathtt{a})}.$$

Notice that if the values $w(\mathtt{a})$ are already probabilities then de denominator is equal to 1, and if the values $w(\mathtt{a})$ count the number of occurrences of `a` in some string, then the denominator is the total number of letters in the string.

For example the source

```
[("0",0.9),("1",0.1)]
```

is a binary source with binary alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ that produces the symbols `0` and `1` with probabilities $p(\mathtt{0}) = 0.9$ and $p(\mathtt{1}) = 0.1$, respectively. As a source it exactly *the same source* than the following one

```
[("0",18),("1",2)]
```

attached to the string `"00000100000000000100"`.

Sources and entropy. Let `src` denote a source, with the format previously described, and let `str` denote a `Python` string. Implement the following functions:

- `source(str)` that outputs the source corresponding to a given string `str`.
- `source_extension(src,k)` that outputs the `k`–th. source extension of a given source `src`.
- `entropy_source(src)` that outputs the entropy of a source `src` (a floating-point number measuring bits).

Source coding. Create three functions `shannon_code`, `shannon_fano_code` and `huffman_code` that take as input a source of information and give as output two results: a list containing a binary code for that source, according to the name of the function, and the mean length of this code.

Here by a "Shannon code" for a source S with probabilities $p_1, p_2, \ldots, p_n$ we understand a prefix code with words $\mathbf{c}_i$ of lengths $\ell(\mathbf{c}_i) = \lceil \log_2(1/p_i) \rceil$, the integer parts by excess of the logarithms of the inverses of the probabilities. This is the code used in Shannon's source coding theorem for proving the second inequality (cf. theory sessions, Lesson 3). In general, this is the worst of the three codes.

The other two codes, Shannon-Fano and Huffman are described and studied in the theory sessions (Lesson 4).

For example, if the input source is the 2-extension of the binary source described previously, then correct outputs of the three functions, respectively, could be:

- `["0","1000","1001","1111111"]`, 1.6;
- `["0","10","110","111"]`, 1.29; and
- `["0","10","110","111"]`, 1.29.

Namely, after executing the command `cod,ml=huffman_code([("0",0.9),("1",0.1)])` the variables `cod` and `ml` should contain the list of binary words `["0","10","110","111"]` and the floating point number 1.29, respectively, and analogously for the other two functions.

In this example, Shannon-Fano code is already optimal, but this is not always true: for the source corresponding to `"setzejutgesdunjutjatmengenfetgedunpenjat"` the mean lengths are 3.875, 3.550 and 3.425, respectively.

Delivering. Deliver a single .py file including your implementations of the six functions.