# Random Text

Jon Sporring

October 18, 2019

## 1 Lærervejledningn

**Emne** Functional programming, histograms, random values

**Sværhedsgrad** Middel

## 2 Introduktion

H.C. Andersen (1805-1875) is a Danish author who wrote plays, travelogues, novels, poems, but perhaps is best known for his fairy tales. An example is Little Claus and Big Claus (Danish: Lille Claus og store Claus), which is a tale about a poor farmer, who outsmarts a rich farmer. A translation can be found here: `http://andersen.sdu.dk/vaerk/hersholt/LittleClausAndBigClaus_e.html`. It starts like this:

> Hans Christian Andersen's "Lille Claus og Store Claus" translated by Jean Hersholt.
>
> In a village there lived two men who had the self-same name. Both were named Claus. But one of them owned four horses, and the other owned only one horse; so to distinguish between them people called the man who had four horses Big Claus, and the man who had only one horse Little Claus. Now I'll tell you what happened to these two, for this is a true story."

A translation of the tale is distributed with this exercise as `littleClausAndBigClaus.txt` and will henceforth be called The Story.

Markov Chains are models which can be used to describe texts. It is a probabilistic model, where the probability of the next element is modeled to depend at most on the previous $n$ elements,

$$p(e_i|e_{i-1},e_{i-2},\ldots,e_{i-n}) \tag{1}$$

Elements could be characters or words, and $n$ is called the order of the chain. It turns out that when estimating the probabilities for a natural text, the longer the chain, the more randomly generated texts resemble a human-generated text.

In this assignment, you are to work with simple text processing, analyze the statistics of the text, and use this to generate a new text with similar statistics. You are to write a number of functions, which all are to be placed in a single library file called `textAnalysis.fs`

## 3 Opgave(r)

1. The script `readFile.fsx` reads the content of the text file `readFile.fsx`. Convert this script into a function which can read the content of any text file and has the following type:

   ```
   readText : filename:string -> string
   ```

   Add this function to your library.

2. Add a function to your library which converts a string, such that all letters are converted to lower case, and removes all characters except a...z and space. It should have the following type:

   ```
   convertText : src:string -> string
   ```

3. Write a function,

   ```
   histogram : src:string -> int list
   ```

   which counts occurrences of each lower-case letter of the English alphabet in a string and returns a list. The first element of the list should be the count of 'a's, second the count of 'b's etc. Add this function to your library.

4. Write a white-box test of the library functions above.

5. The script `mockup.fsx` contains the function

   ```
   randomString : hist:int list -> len:int -> string
   ```

   which generates a string of a given length, and contains random characters distributed according to a given histogram. The program is complete in the sense that it compiles and runs without errors, but its `histogram` function is a mockup function and does not produce the correct histograms.

   Your task is to extend your library with the functions from `mockup.fsx` (except the mockup)

6. The program in Exercise 5 generates random strings and cannot easily be tested with either white- or blackbox testing. Nevertheless, the generated random string must have a histogram similar to the input histogram.

   Extend your library with the function,

   ```
   diff : h1:int list -> h2:int list -> double
   ```

   which compares two histograms as the average sum of squared differences,

   $$\text{diff}(h_1,h_2) = \frac{1}{N}\sum_{i=0}^{N-1}(h_1(i)-h_2(i))^2 \qquad (2)$$

   where $h_1$ and $h_2$ are two histograms of $N$ elements.

   Write an application, which uses your function to compare The Story with a random text generated from the histogram of The Story using the developed code from Exercise 5. Repeat this comparison 10 times, each time with a new randomly generated text from the same histogram. Reflect on possible variations in the resulting differences.

7. Extend your library with the function

   ```
   cooccurrence : src:string -> int list list
   ```

   which counts occurrences of each pairs of lower-case letter of the English alphabet including space in a string and returns a list of lists (a table). In the return list, the first element should be a list of the counts of 'a' being the initial character, i.e., how many times “aa”, “ab”, “ac”,...,“az”, “a ” was observed. The second list should containt the counts of combinations starting with 'b', i.e., how many times “ba”, “bb”, “bc”,... was observed and so on. The function should include overlapping pairs, for example, the input string “abcd” has the pairs “ab”, “bc”, and “cd”.

8. Write a white-box test of `cooccurrence`.

9. Extend your library with a function

   ```
   fstOrderMarkovModel : cooc:int list list -> len:int -> string
   ```

   which generates a random string of length `len`, whose character pairs are distributed according to a user specified cooccurrence histogram `cooc`.

10. Extend your library with the function,

    ```
    diff2 : c1:int list list -> c2:int list list -> double
    ```

    which compares two histograms as the average sum of squared differences,

    $$\text{diff2}(c_1, c_2) = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (c_1(i,j) - c_2(i,j))^2 \tag{3}$$

    where $c_1$ and $c_2$ are two cooccurrence histograms of $N$ elements such that $c_1(i,j)$ is the number of times character number $i$ is found following character number $j$.

    Write an application, which uses your function to compare The Story with a random text generated from the cooccurrences of The Story using the developed code from Exercise 5. Repeat this comparison 10 times, each time with a new randomly generated text from the same cooccurrence histogram. Reflect on possible variations in the resulting differences.

11. The function `randomString` may be considered a zero-order Markov Chain model, since each generated character is independent on any previously generated characters. The function `fstOrderMarkovModel` generates new characters dependent on the previous character and is therefore a first-order Markov Chain model. Consider a similar extension to an n'th-order Markov Chain where the occurrences of n-tupples of characters are stored in `n : int list list ... list`. What possible pit-falls are there with this representation?

12. Extend your library with a function that counts occurrences of each word in a string and returns a list. The counts must be organized as a list of trees using the following `Tree` type:

    ```
    type Tree = Node of char * int * Tree list
    ```

    An illustration of a value of this type is shown in Figure 1 Words are to be represented as the sequence of characters from the root til a node. The associated integer to each node counts the occurrence of a word ending in that node. Thus, if the count is 0, then no word with that endpoint has occurred. For example, a string with the words “a abc ba” should result in the following tree,

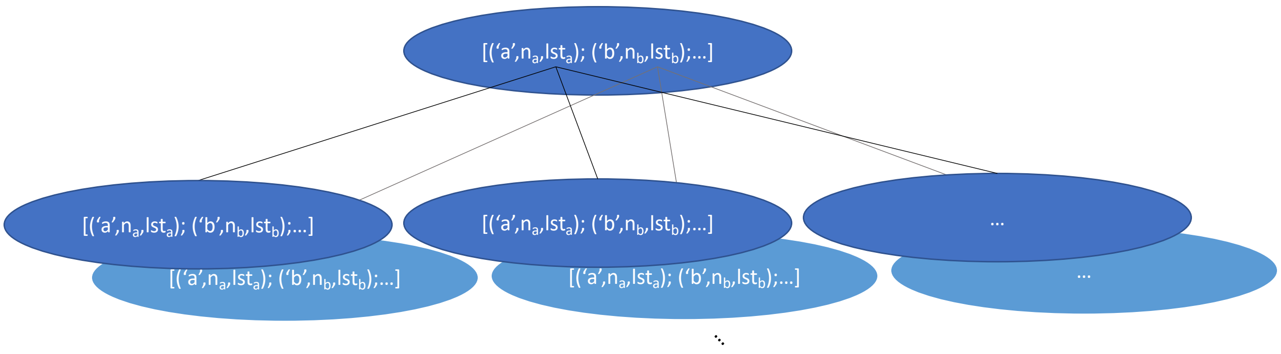
[('a',n_a,lst_a); ('b',n_b,lst_b);...]
[('a',n_a,lst_a); ('b',n_b,lst_b);...]
[('a',n_a,lst_a); ('b',n_b,lst_b);...]
...
[('a',n_a,lst_a); ('b',n_b,lst_b);...]
[('a',n_a,lst_a); ('b',n_b,lst_b);...]
...

Figure 1: An illustration of a list of values of the type `Tree`.

```
[Node ('a', 1, [Node ('b', 0, [Node ('c', 1, [])])]);
 Node ('b', 0, [Node ('a', 1, [])])]
```

Notice, the counts are zero for the combinations "ab" and "b", which are words not observed in the string. The function must have the type:

```
wordHistogram : src:string -> Tree list
```

13. Write a white-box test of `wordHistogram`.

14. Extend your library with a function

```
randomWords : wHist:Tree list -> nWords:int -> string
```

which generates a string with `nWords` number of words randomly selected to match the word-histogram in `wHist`.

15. Extend your library with the function,

```
diffw : t1:Tree list -> t2:Tree list -> double
```

which compares two word-histograms as the average sum of squared differences,

$$\text{diffw}(t_1,t_2) = \frac{1}{M}\sum_{i=0}^{M-1}(t_1(i)-t_2(i))^2 \qquad (4)$$

where $t_1$ and $t_2$ are two word histograms, and $M$ is the total number of different words observed in the two texts.

Write an application, which uses your function to compare The Story with a random text generated from the word-histogram of The Story using the developed code from Exercise 12. Repeat this comparison 10 times, each time with a new randomly generated text from the same histogram. Reflect on possible variations in the resulting differences.

16. In terms of a Markov Chain, what order is `randomWords`? Suggest an extension of `type Tree` to include first order Markov Chains. Speculate on whether this principle can be used to extend to n'th order mordels, and, in case, speculate on how the storage requirements will grow as n grows.

17. Write a short report, which

    - is no larger than 5 pages;

- includes answers to questions posed;
- contains a brief discussion on how your implementation works, and if there are any possible alternative implementations, and in that case, why you chose the one, you did;
- includes output that demonstrates that your solutions work as intented.