Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 7 - gruppeopgave

Jon Sporring

Version 2

21. oktober - 8. november.
Afleveringsfrist: lørdag d. 9. november kl. 23:59.

Emnerne for denne arbejdsseddel er:

- rekursion,
- pattern matching,
- sum-typer,
- endelige træer.

Opgaverne er delt i øve- og afleveringsopgaver. I denne periode skal I arbejde i grupper med jeres afleveringsopgaver. Regler for gruppe- og individuelle afleveringsopgaver er beskrevet i "'Noter, links, software m.m."'→"'Generel information om opgaver"'.

Øveopgaver

7ø0 Betragt insertion sort funktionen `isort`. Omskriv funktionen `insert` således, at den benytter sig af pattern matching på lister.

7ø1 Betragt bubble sort funktionen `bsort`. Omskriv den, at den benytter sig af pattern matching på lister. Funktionen kan passende benytte sig af "nested pattern matching" i den forstand at den kan implementeres med et match case der udtrækker de to første elementer af listen samt halen efter disse to elementer.

7ø2 Opskriv black-box tests for de to sorteringsfunktioner og vær sikker på at grænsetilfældene dækkes (ingen elementer, et element, to elementer, samt flere elementer, sorteret, omvendt sorteret, etc.)

7ø3 Omskriv funktionen `merge`, som benyttes i forbindelse med funktionen `msort` (mergesort) fra forelæsningen, således at den benytter sig af pattern matching på lister.

7ø4 Opskriv black-box tests for sorteringsfunktionen `msort` og vær sikker på at grænsetilfældene dækkes (ingen elementer, et element, to elementer, samt flere elementer, sorteret, omvendt sorteret, etc.)

I det efterfølgende skal der arbejdes med sum-typen:

```
type weekday = Monday | Tuesday | Wednesday | Thursday
             | Friday | Saturday | Sunday
```

som repræsenterer ugens dage.

7ø5 Lav en funktion `dayToNumber : weekday -> int`, der givet en ugedag returnerer et tal, hvor mandag skal give tallet 1, tirsdag tallet 2 osv.

7ø6 Lav en funktion `nextDay : weekday -> weekday`, der givet en ugedag returnerer den næste dag, så mandag skal give tirsdag, tirsdag skal give onsdag, osv, og søndag skal give mandag.

7ø7 Lav en funktion `numberToDay : n : int -> weekday option`, sådan at `numberToDay n` returnerer `None`, hvis `n` ikke ligger i intervallet $1 \ldots 7$, og ellers returnerer ugedagen `Some d`. Det skal gælde, at `numberToDay (dayToNumber d)` $\rightsquigarrow$ `Some d` for alle ugedage `d`.

7ø8 Ved at benytte biblioteket `ImgUtil`, som beskrevet i forelæsningen, er det muligt at tegne simpel liniegrafik samt fraktaler, som f.eks. Sierpinski-fraktalen, der kan tegnes ved at tegne små firkanter bestemt af et rekursivt mønster. Koden for Sierpinski-trekanten er givet som følger:

```
open ImgUtil

let rec triangle bmp len (x,y) =
  if len < 25 then setBox blue (x,y) (x+len,y+len) bmp
  else let half = len / 2
       do triangle bmp half (x+half/2,y)
       do triangle bmp half (x,y+half)
       do triangle bmp half (x+half,y+half)

do runSimpleApp "Sierpinski" 600 600 (fun bmp -> triangle bmp
   512 (30,30) |> ignore)
```

Tilpas funktionen således at trekanten tegnes med røde streger samt således at den kun tegnes 2 rekursionsniveauer ned. Hint: dette kan gøres ved at ændre betingelsen `len < 25`.

7ø9 I stedet for at benytte `ImgUtil.runSimpleApp` funktionen skal du nu benytte `ImgUtil.runApp`, som giver mulighed for at din løsning kan styres ved brug af tastaturet. Funktionen `ImgUtil` har følgende type:

```
val runApp : string -> int -> int
          -> (int -> int -> 's -> System.Drawing.Bitmap)
          -> ('s -> System.Windows.Forms.KeyEventArgs
                 -> 's option)
          -> 's -> unit
```

De tre første argumenter til `runApp` er vinduets titel (en streng) samt vinduets initielle vidde og højde. Funktionen `runApp` er parametrisk over en brugerdefineret type af tilstande (`'s`). Antag at funktionen kaldes som følger:

```
runApp title width height draw react init
```

Dette kald vil starte en GUI applikation med titlen `title`, vidden `width` og højden `height`. Funktionen `draw`, som brugeren giver som 4. argument kaldes initielt når applikationen starter og hver gang vinduets størrelse justeres eller ved at funktionen `react` er blevet kaldt efter en tast er trykket ned på tastaturet. Funktionen `draw` modtager også (udover værdier for den aktuelle vidde og højde) en værdi for den brugerdefinerede tilstand, som initielt er sat til værdien `init`. Funktionen skal returnere et bitmap, som for eksempel kan konstrueres med funktionen `ImgUtil.mk` og ændres med andre funktioner i `ImgUtil` (f.eks. `setPixel`).

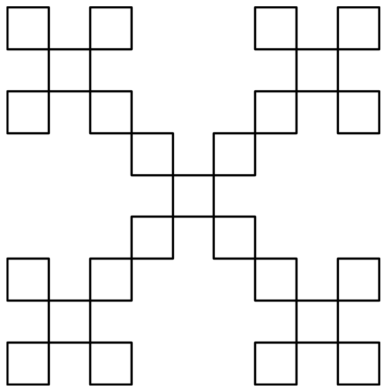
Funktionen `react`, som brugeren giver som 5. argument kaldes hver gang brugeren trykker på en tast. Funktionen tager som argument:

- en værdi svarende til den nuværende tilstand for applikationen, og
- et argument der kan benyttes til at afgøre hvilken tast der blev trykket på.[1]

Funktionen kan nu (eventuelt) ændre på dens tilstand ved at returnere en ændret værdi for denne.

Tilpas applikationen således at dybden af fraktalen kan styres ved brug af piletasterne, repræsenteret ved værdierne `System.Windows.Forms.Keys.Up` og `System.Windows.Forms.Keys.Down`.

7ø10 Med udgangspunkt i øvelsesopgave 7ø8 skal du i denne opgave implementere en GUI-applikation der kan tegne en version af X-fraktalen som illustreret nedenfor (eventuelt i en dybde større end 2).



Bemærk at det ikke er et krav, at dybden på fraktalen skal kunne styres med piletasterne, som det er tilfældet med Sierpinski-fraktalen i øvelsesopgave 7ø9.

[1] Hvis `e` har typen `System.Windows.Forms.KeyEventArgs` kan betingelsen `e.KeyCode = System.Windows.Forms.Keys.Up` benyttes til at afgøre om det var tasten "Up" der blev trykket på.

Afleveringsopgaver

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.

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`, in a module called `textAnalysis`, and you are to write a number of tests of this library, which are to be placed in a single file called `testTextAnalysis.fsx`.

7g0 The script `mockup.fsx` contains a number of functions including

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

The function `randomString` generates an identically and independently distributed string of a given length, where the characters are distributed according to a given histogram. The script 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.

Create the library file `textAnalysis.fs` and add the functions from `mockup.fsx`

7g1 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.

7g2 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
```

7g3 Write a function,

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

which counts occurrences of each of the characters `['a'..'z']@[' ']` 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. Use this function to replace the mockup function in your library.

7g4 Write a white-box test of the library functions `readText`, `convertText`, and `histogram`, and add these to your test file.

7g5 The function `randomString` is not easily tested using white- or blackbox testing since it is a random function. Instead you are to test its output by the histogram of the characters in the string it produces.

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,

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

where h_1 and h_2 are two histograms of N elements.

Extend your test file with a test of `randomString` as follows:

(a) Convert The Story using `convertText` and calculate its histogram.

(b) Use this to generate a random text using `randomString` with length N, where N is the length of the converted The Story.

(c) Calculate the distance between the histograms of The Story and the random texts using `diff`.

(d) Set the test as passed if the difference is below some threshold.

You must choose a threshold in the above, and you must argue for your choice.

7g6 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 returned 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".

7g7 Extend your test file with a white-box test of `cooccurrence`.

7g8 Extend your library with a function

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

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

7g9 The function `markovChain` is a random function, and you are to test its output by the cooccurrences of the characters in the string it produces.

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,

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

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.

Extend your test file with a test of `markovChain` as follows:

(a) Convert The Story using `convertText` and calculate its cooccurrence histogram.

(b) Use this to generate a random text using `markovChain` with length N, where N is the length of the converted The Story.

(c) Calculate the distance between the cooccurrence histograms of The Story and the random texts using `diff2`.

(d) Set the test as passed if the difference is below some threshold.

You must choose a threshold in the above, and you must argue for your choice.

7g10 Extend your library with a function that counts occurrences of each word in a string and returns a list. The list must be organized using the following type:

```
type wordHistogram = (string * int) list
```

For example, a string with the words “a abc ba ba” should result in the following list,

```
[("a", 1); ("abc", 1); ("ba",2)]
```

The function must have the type:

```
wordHistogram : src:string -> wordHistogram
```

7g11 Write a white-box test of `wordHistogram`, and add it to your test file.

7g12 Extend your library with a function

```
randomWords : wHist:wordHistogram -> nWords:int -> string
```

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

7g13 The function `randomWords` is a random function, and you are to test its output by the histogram of the words in the string it produces.

Extend your library with the function,

```
diffw : w1: wordHistogram -> w2:wordHistogram -> double
```

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

$$\text{diffw}(w_1, w_2) = \frac{1}{M} \sum_{i=0}^{M-1} (w_1(i) - w_2(i))^2 \tag{3}$$

where w_1 and w_2 are two word histograms, and M is the total number of different words observed in the two texts.

Extend your test file with a test of `randomWords` as follows:

(a) Convert The Story using `convertText` and calculate its word histogram.

(b) Use this to generate a random text using `randomWords` with M words, where M is the number of words in the converted The Story.

(c) Calculate the distance between the word histograms of The Story and the random texts using `diffw`.

(d) Set the test as passed if the difference is below some threshold.

You must choose a threshold in the above, and you must argue for your choice.

7g14 Extend your library with the function

```
type wordCooccurrences = (string * wordHistogram) list
cooccurrenceOfWords : src:string -> wordCooccurrences
```

which counts occurrences of each pair of words in a string and returns a list of pairs. Each returned pair must be a word and the wordHistogram of counts of cooccurrences. E.g., for the string "a hat and a cat" the function must return,

```
[("a", [("hat", 1); ("cat", 1)]);
 ("and", [("a", 1)]);
 ("cat", []);
 ("hat", [("and", 1)])]
```

7g15 Extend your test file with a white-box test of `cooccurrenceOfWords`.

7g16 Extend your library with a function

```
wordMarkovChain : wCooc: wordCooccurrences -> nWords:int -> string
```

which generates a random string with `nWords` words, whose word-pairs are distributed according to the cooccurrence histogram `wCooc`.

7g17 The function `wordMarkovChain` is a random function, and you are to test its output by the cooccurrences of the words in the string it produces.

Extend your library with the function,

```
diffw2 : c1:wordCooccurrences -> c2:wordCooccurrences -> double
```

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

$$\text{diffw2}(c_1, c_2) = \frac{1}{M^2} \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} (c_1(i,j) - c_2(i,j))^2 \tag{4}$$

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

Extend your test file with a test of `wordMarkovChain` as follows:

(a) Convert The Story using `convertText` and calculate its cooccurrence histogram.

(b) Use this to generate a random text using `wordMarkovChain` with length M, where M is the length of the converted The Story.

(c) Calculate the distance between the cooccurrence histograms of The Story and the random texts using `diffw2`.

(d) Set the test as passed if the difference is below some threshold.

You must choose a threshold in the above, and you must argue for your choice.

7g18 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.

Afleveringen skal bestå af

- en zip-fil, der hedder `7g_<navn>.zip` (f.eks. `7g_jon.zip`)
- en pdf-fil , der hedder `7g_<navn>.pdf` (f.eks. `7g_jon.pdf`)

Zip-filen `7g_<navn>.zip` skal indeholde en og kun en mappe `7g_<navn>`. I den mappe skal der ligge en `src` mappe og filen `README.txt`. I `src` skal der ligge følgende og kun følgende filer: `textAnalysis.fs` og `testTextAnalysis.fs` svarende til hver af delopgaverne. De skal kunne oversættes med `fsharpc`, og de oversattte filer skal kunne køres med mono. Funktioner skal dokumenteres ifølge dokumentationsstandarden som minimum ved brug af `<summary>`, `<param>` og `<returns>` XML-tagsne. Filen `README.txt` skal ganske kort beskrive, hvordan koden oversættes og køres. Pdf-filen skal indeholde jeres rapport oversat fra LaTeX. Husk at pdf-filen skal uploades ved siden af zip-filen på Absalon.

God fornøjelse.