Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 7 - individuel opgave

Martin Elsman

23. oktober – 1. november.
Afleveringsfrist: onsdag d. 1. november kl. 22:00

I denne periode skal I arbejde individuelt. Regler for individuelle afleveringsopgaver er beskrevet i "Noter, links, software m.m." → "Generel information om opgaver". Formålet er at arbejde med:

- rekursion
- pattern matching

Opgaverne er delt i øve- og afleveringsopgaver.

Øveopgaver

7ø.0 Omskriv funktionen `insert`, som benyttes i forbindelse med funktionen `isort` (insertion sort) fra forelæsningen, således at den benytter sig af pattern matching på lister.

7ø.1 Omskriv funktionen `bsort` (bubble sort) fra forelæsningen således 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.

7ø.2 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.)

7ø.3 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 ned til dybde 2 (hint: du skal ændre betingelsen `len < 25`).

7ø.4 I stedet for at benytte funktionen `ImgUtil.runSimpleApp` skal du nu benytte funktionen `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 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 samt 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.Ke`

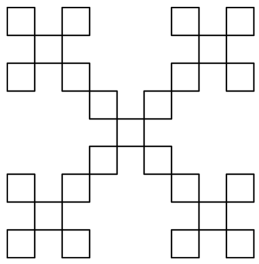
[1] Hvis `e` har typen `System.Windows.Forms.KeyEventArgs` kan betingelsen `e.KeyCode = System.Windows.Forms.Keys.Up` benyttes til at afgøre om det var tasten "Up" der blev trykket på.

Afleveringsopgaver

7i.0 Omskriv funktionen `merge`, som benyttes i forbindelse med funktionen `msort` (merge-sort) fra forelæsningen, således at den benytter sig af pattern matching på lister.

7i.1 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.)

7i.2 Med udgangspunkt i øvelsesopgave 7ø.3 skal du i denne opgave implementere en GUI-applikation der kan tegne en version af X-fractalen som illustreret til højre (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 7ø.4.

7i.3 Du skal i de følgende to opgaver arbejde med en funktion til at bestemme den såkaldte *Levensthein-distance* mellem to strenge a og b. Distancen er defineret som det mindste antal editeringer, på karakter-niveau, det er nødvendigt at foretage på strengen a før den resulterende streng er identisk med strengen b. Som editeringer forstås (1) sletninger af karakterer, (2) indsættelser af karakterer, og (3) substitution af karakterer.

Varianter af Levensthein-distancen mellem to strenge kan således benyttes til at identificere om studerende selv har løst deres indleverede opgaver eller om der potentielt set er tale om plagiatkode ;)

Matematisk set kan Levensthein-distancen $leven(a,b)$, mellem to karakterstrenge a og b, defineres som $lev_{a,b}(|a|,|b|)$, hvor $|a|$ og $|b|$ henviser til længderne af henholdsvis a og b, og hvor funktionen lev er defineret som følger:[2]

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

hvor $1_{(a_i \neq b_j)}$ henviser til *indikatorfunktionen*, som er 1 når $a_i \neq b_j$ og 0 ellers.

Implementér funktionen *leven* direkte efter den matematiske definition (ved brug af rekursion) og test korrektheden af funktionen på nogle små strenge, såsom "house" og "horse" (distance 1) samt "hi" og "hej" (distance 2).

7i.4 Den direkte implementerede rekursive funktion er temmelig ineffektiv når strengene a og b er store. F.eks. tager det en del millisekunder at udregne distancen mellem strengene "dangerous house" and "danger horse". Årsagen til denne ineffektivitet er at en løsning der bygger direkte på den rekursive definition resulterer i en stor mængde genberegninger af resultater der allerede er beregnet.

[2] See `https://en.wikipedia.org/wiki/Levenshtein_distance`.

For at imødekomme dette problem skal du implementere en såkaldt "caching mekanisme" der har til formål at sørge for at en beregning højst foretages en gang. Løsningen kan passende gøre brug af gensidig rekursion og tage udgangspunkt i løsningen for den direkte rekursive definition (således skal løsningen nu implementeres med to gensidigt rekursive funktioner `leven` og `leven_cache` forbundet med `and`). Som cache skal der benyttes et 2-dimensionelt array af størrelse $|a| \times |b|$ indeholdende heltal (initielt sat til -1).

Funktionen `leven_cache`, der skal tage tilsvarende argumenter som `leven`, skal nu undersøge om der allerede findes en beregnet værdi i cachen, i hvilket tilfælde denne værdi returneres. Ellers skal funktionen `leven` kaldes og cachen opdateres med det beregnede resultat. Endelig er det nødvendigt at funktionen `leven` opdateres til nu at kalde funktionen `leven_cache` i hver af de rekursive kald.

Test funktionen på de små strenge og vis at funktionen nu virker korrekt også for store input.

Det skal til slut bemærkes at den implementerede løsning benytter sig af $O(|a| \times |b|)$ plads og at der findes effektive løsninger der benytter sig af mindre plads ($O(\max(|a|,|b|))$). Det er ikke et krav at din løsning implementerer en af disse mere pladsbesparende strategier.

Afleveringsopgaven skal afleveres som et antal `fsx` tekstfiler navngivet efter opgaven, som f.eks. `7i0.fsx`. Tekstfilerne skal kunne oversættes med `fsharpc`,[3] og resultatet skal kunne køres med `mono` eller eventuelt `mono32`. Funktioner skal dokumenteres ifølge dokumentationsstandarden, og udover selve programteksten skal besvarelserne indtastes som kommentarer i de `fsx`-filer, de hører til. Det hele skal samles i en zip fil og uploades på Absalon.

Til øvelserne forventer vi at I arbejder efter følgende skema:

Mandag 23/10: Afslut 6g og start på øvelsesopgaverne fra 7i

Tirsdag 24/10: Arbejd med øvelses- og afleveringsopgaverne

Fredag 27/10 Arbejd med afleveringsopgaverne

[3] Løsningerne kan antage at biblioteket `ImgUtil.dll` fra forelæsningerne er tilgængeligt.