

Programmering og Problemløsning, 2017

Rekursion – Towers of Hanoi og Liniegrafik

Martin Elsman

Datalogisk Institut
Københavns Universitet
DIKU

13. Oktober, 2017

- 1 Rekursion – Towers of Hanoi og Liniegrafik
 - Introduktion
 - Towers of Hanoi
 - Liniegrafik

Rekursion

En metode for hvilken en løsning til et problem findes ved at løse mindre instanser af det samme problem.

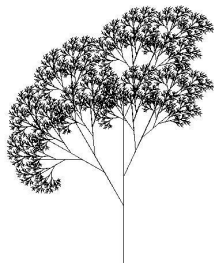
I dag vil vi se på brug af rekursion til to formål:

1 Implementation af spillet Towers of Hanoi.

Spilleren (evt. computeren) skal flytte N skiver der er placeret i orden på den første af tre pinde til den sidste pind. Spilleren må kun flytte en skive af gangen og en stor skive må ikke placeres ovenpå en mindre.

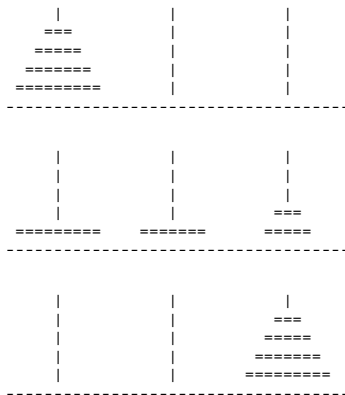
2 Tegning af figurer ved hjælp af linier.

Ved brug af et simpelt F# GUI interface kan vi tegne rekursive figurer med linier.



Spillet Towers of Hanoi

- Spillet spilles med N skiver der kan placeres på tre pinde.
- Udgangspunktet er at alle skiverne ligger i orden på den første pind.
- Spilleren (evt. computeren) skal flytte skiverne (en af gangen) således at alle ender på den fjerneste pind.
- På intet tidspunkt må en stor skive ligge ovenpå en mindre skive.



Vi deler spillet Towers of Hanoi i to dele:

- 1 Et modul der implementerer spilkonfigurering og tegning af pinde og skiver.
Dette modul vil håndhæve reglerne og give både dig og computeren mulighed for at spille.
- 2 En applikation der kan spille spillet ved brug af en **rekursiv** algoritme.

Modulet Pegs:

```
module Pegs

type t
val init      : int -> t
val move      : int -> int -> t -> t
val toString  : t -> string

module App =
  val reset : int -> unit
  val mv    : int -> int -> unit
```

Bemærk:

- Typen `t` refererer til den interne repræsentation af “spil-konfigurationen” (skiverne på pindene).
- Ved at “gemme” repræsentationen kan en bruger kun ændre på konfigurationen ved brug af `init` og `move`.
- Modulet `App` bruges når et menneske skal spille spillet i `fsharp`.

DEMO af modulet Pegs

```
bash-3.2$ fsharpc --nologo -a pegs.fsi pegs.fs
bash-3.2$ fsharpi --nologo -r pegs.dll --readline-
```

```
> open Pegs.App;;
> reset 3;;
```

```
      |           |           |
      ===         |           |
      =====    |           |
      =====    |           |
-----
```

```
val it : unit = ()
```

```
>
```

Den interne spil-repræsentation

- Typen `t` i `Pegs` modulet er internt repræsenteret som en liste (af længde 3) af heltalslister.
- Som eksempel kan en start-konfiguration være repræsenteret som:

```
[[1;2;3]; []; []]
```

- Operationen `Pegs.move` flytter det øverste tal i en liste til en anden hvis reglerne er opfyldt. Ellers *fejler* operationen.

```
val move : int -> int -> t -> t
```

- Operationen `Pegs.init` konstruerer en ny start-konfiguration.

```
val init : int -> t
```

- Operationen `Pegs.toString` konstruerer en streng-repræsentation af en konfiguration (til udskrivning).

```
val toString : t -> string
```

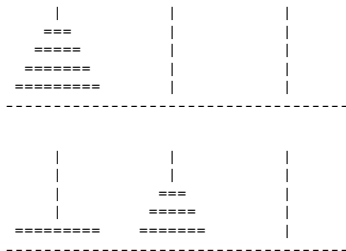

Hanoi algoritmen

Kan vi finde en generel strategi for at gennemføre spillet uden at bryde reglerne?

Ja – Her er en strategi:

For at flytte N skiver fra en startpind til en målpind, ved brug af en reservepind:

- Hvis N er 0: du er færdig!
- Ellers:
 - 1 Flyt (rekursivt) $N - 1$ skiver fra startpinden til reservepinden (med brug af målpinden som reservepind).
 - 2 Flyt 1 pind fra startpinden til målpinden.
 - 3 Flyt (rekursivt) $N - 1$ skiver fra reservepinden til målpinden (med brug af startpinden som reservepind).



Hanoi algoritmen i F# (hanoi.fs)

```
let rec hanoi n src aux tgt pegs =  
    if n = 0 then pegs  
    else let pegs = hanoi (n-1) src tgt aux pegs  
         let pegs = Pegs.move src tgt pegs  
         do printf "%s" (Pegs.toString pegs)  
         let pegs = hanoi (n-1) aux src tgt pegs  
         in pegs
```

```
let play i =  
    let pegs = Pegs.init i  
    do printf "%s" (Pegs.toString pegs)  
    hanoi i 0 1 2 pegs
```

```
let res = play 4
```

Spørgsmål:

- Vi udskriver konfigurationen efter hver flytning.
- Kan vi skrive en simpel funktion til beregning af antal flytninger?

Simpel funktion til beregning af antal flytninger:

```
let rec hanoi_count n =  
  if n <= 0 then 0  
  else 2*hanoi_count (n-1) + 1  
do printf "%d\n" (hanoi_count 5)
```

Bemærk:

- Funktionen `hanoi_count` n beregner tallet $2^n - 1$.
- Kodefilerne for `pegs.fsi` samt `pegs.fs` er tilgængelige på Absalon (under Filer).

```
bash-3.2$ fsharpc --nologo -a pegs.fsi pegs.fs  
bash-3.2$ fsharpc --nologo -r pegs.dll hanoi.fs  
bash-3.2$ mono hanoi.exe  
...
```

Simpel funktionalitet til Liniegrafik:

Biblioteket `img_util.dll` giver mulighed for at åbne en simpel GUI applikation indeholdende et bitmap der kan tegnes i.

Biblioteksfilerne `img_util.fsi` samt `img_util.fs` er tilgængelige på Absalon (under Filer).

Udvalgte funktioner (`img_util.fsi`)

```
module ImgUtil
```

```
type color = System.Drawing.Color
```

```
val red : color
```

```
type bitmap = System.Drawing.Bitmap
```

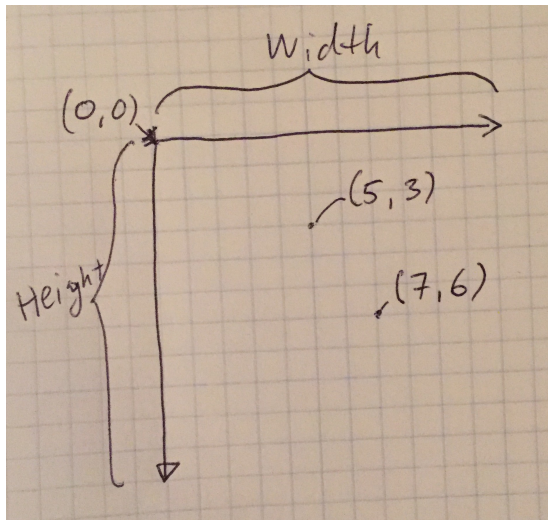
```
val setLine : color -> int*int -> int*int -> bitmap -> unit
```

```
val runSimpleApp : string -> int -> int  
                -> (bitmap -> unit) -> unit
```

```
...
```

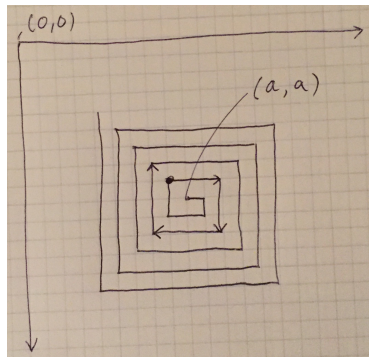
Bitmap Koordinater

Origin (0,0) findes i øverste venstre hjørne.



En simpel applikation

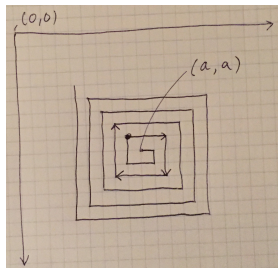
- Konstruer en applikation med et 600×600 bitmap canvas.
- Tegn en rød firkant-spiral.
- Benyt `setLine` funktionaliteten.
- Start spiralen i punkt $(300, 300)$



Koden for spiral.fs

```
open ImgUtil
```

```
let rec spiral bmp s i x y =  
  if i >= 350 then ()  
  else let p1 = (x,y)  
        let p2 = (x+i,y)  
        let p3 = (x+i,y+i)  
        let p4 = (x-s,y+i)  
        let p5 = (x-s,y-s)  
        do setLine red p1 p2 bmp  
        do setLine red p2 p3 bmp  
        do setLine red p3 p4 bmp  
        do setLine red p4 p5 bmp  
        spiral bmp s (i+2*s) (x-s) (y-s)  
  
do runSimpleApp "Spiral" 400 400  
  (fun bmp -> spiral bmp 10 10 200 200 |>  
    ignore)
```

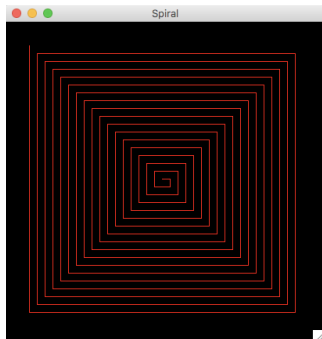


Compilér og kør

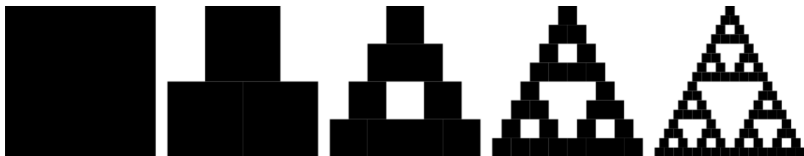
```
$ fsharpc -a img_util.fsi img_util.fs  
$ fsharpc -r img_util.dll spiral.fs  
$ mono32 spiral.exe
```

Bemærk:

- På Mac OS er det nødvendigt at køre med mono32.
- På andre platforme skulle mono virke fint.
- Applikationen kan lukkes med ESC, f.eks.



Sierpinski – tegn trekanter med firkanter!



Kode:

```
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" 450 475 (fun bmp -> triangle bmp  
  400 (25,25) |> ignore)
```

