Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 6 - gruppeopgave

Jon Sporring

7. oktober - 11. oktober.
Afleveringsfrist: lørdag d. 12. oktober kl. 23:59.

Emnerne for denne arbejdsseddel er:

- rekursion.

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

6ø0 Skriv en funktion, `fac : n:int -> int`, som udregner fakultetsfunktionen $n! = \prod_{i=1}^{n} i$ vha. rekursion.

6ø1 Skriv en funktion, `sum : n:int -> int`, som udregner summen $\sum_{i=1}^{n} i$ vha. rekursion. Lav en tabel som i Opgave 3i0 og sammenlign denne implementation af `sum` med `while`-implementation og `simpleSum`.

6ø2 Skriv en funktion, `sum : int list -> int`, som tager en liste af heltal og returnerer summen af alle tallene. Funktionen skal traversere listen vha. rekursion.

6ø3 Den største fællesnævner mellem 2 heltal, t og n, er det største heltal c, som går op i både t og n med 0 til rest. Euclids algoritme[1] finder den største fællesnævner vha. rekursion:

$$\gcd(t,0) = t, \tag{1}$$
$$\gcd(t,n) = \gcd(n, t \ \% \ n), \tag{2}$$

hvor % er rest operatoreren (som i F#).

(a) Implementer Euclids algoritm, som en rekursive funktion

[1] `https://en.wikipedia.org/wiki/Greatest_common_divisor`

```
gcd : t:int -> n:int -> int
```

(b) lav en white- og black-box test af den implementerede algoritme,

(c) Lav en håndkøring af algoritmen, gerne på papir, for `gcd 8 2` og `gcd 2 8`.

6ø4 Lav dine egne implementationer af `List.fold` og `List.foldback` ved brug af rekursion.

6ø5 Benyt `List.fold` og `List.foldback` og dine egne implementeringer fra Opgave 6ø4 til at udregne summen af listen `[0 .. n]`, hvor `n` er et meget stort tal, og sammenlign tiden, som de fire programmer tager. Diskutér forskellene.

Afleveringsopgaver

I denne opgave skal I regne med kædebrøker (continued fractions)[2]. Kædebrøker er lister af heltal, som repræsenterer reelle tal. Listen er endelig for rationelle tal og uendelig for irrationelle tal.

Decimaltal til kædebrøk En kædebrøk skrives som: $x = [q_0; q_1, q_2, \ldots]$, hvilket svarer til tallet,

$$x = q_0 + \frac{1}{q_1 + \frac{1}{q_2 + \ldots}}. \tag{3}$$

F.eks. svarer kædebrøken $[3;4,12,4]$ til følgende decimaltal:

$$\begin{aligned} x &= 3 + \frac{1}{4 + \frac{1}{12 + \frac{1}{4}}} && (4)\\ &= 3 + \frac{1}{4 + \frac{1}{12.25}} && (5)\\ &= 3 + \frac{1}{4.081632653} && (6)\\ &= 3.245. && (7) \end{aligned}$$

Altså er $[3;4,12,4] = 3.245$.

Kædebrøk til decimaltal For et givet tal x på decimalform kan dets kædebrøk $[q_0; q_1, q_2, \ldots]$ udregnes ved følgende algoritme: Lad $x_0 = x$ og $i \geq 0$, udregn

$$\begin{aligned} q_i &= \lfloor x_i \rfloor && (8)\\ r_i &= x_i - q_i && (9)\\ x_{i+1} &= 1/r_i && (10)\\ & && (11) \end{aligned}$$

indtil $r_i = 0$. F.eks. for decimaltallet $x = 3.245$ beregnes:

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

i	x_i	$q_i = \lfloor x_i \rfloor$	$r_i = x_i - q_i$	$x_{i+1} = 1/r_i$
0	3.245	3	0.245	4.081632653...
1	4.081632653...	4	0.081632653	12.25
2	12.25	12	0.25	4
3	4	4	0	-

Resultatet aflæses i anden søjle: $3.245 = [3;4,12,4]$.

Heltalsbrøker til kædebrøker Kædebrøken for en heltals brøk t/n udregnes ved følgende algoritme: Lad $r_{-2} = t$ og $r_{-1} = n$ og $i \geq -2$, udregn

$$r_i = r_{i-2} \,\%\, r_{i-1} \quad \text{(rest ved heltalsdivision)}, \tag{12}$$
$$q_i = r_{i-2} \text{ div } r_{i-1} \quad \text{(heltalsdivision)}, \tag{13}$$

indtil $r_{i-1} = 0$. Så vil $[q_0;q_1,\ldots,q_j]$ vil være t/n som kædebrøk. F.eks. for brøken $t/n = 649/200$ beregnes:

i	r_{i-2}	r_{i-1}	$r_i = r_{i-2} \,\%\, r_{i-1}$	$q_i = r_{i-2}$ div r_{i-1}
0	649	200	49	3
1	200	49	4	4
2	49	4	1	12
3	4	1	0	4
4	1	0	-	-

Kædebrøken aflæses som højre søjle: $649/200 = [3;4,12,4]$.

Kædebrøker af heltalsbrøker t/n er særligt effektive at udregne vha. Euclids algoritme for største fællesnævner. Algoritmen i Opgave 6ø3 regner rekursivt på relationen mellem heltalsdivision og rest: Hvis $a = t$ div n er heltalsdivision mellem t og n, og $b = t \,\%\, n$ er resten efter heltalsdivision, så er $t = an + b$. For (12)–(13) skal der altså gælde, at $r_{i-2} = q_i r_{i-1} + r_i$. Algoritmen i Opgave 6ø3 regner udelukkende på r_i som transformationen $(r_{i-2}, r_{i-1}) \to (r_{i-1}, r_i) = (r_{i-1}, r_{i-2} \,\%\, r_{-1})$ indtil $(r_{i-2}, r_{i-1}) = (r_{i-2}, 0)$. Hvis man tilføjer beregning af q_i i rekursionen, har man samtidigt beregnet brøken som kædebrøk.

Kædebrøker til Heltalsbrøker En kædebrøk kan approximeres som en heltalsbrøk $\frac{t_i}{n_i}, i \geq 0$ ved følgende algorime. Lad $t_{-2} = n_{-1} = 0$ og $t_{-1} = n_{-2} = 1$, udregn

$$t_i = q_i t_{i-1} + t_{i-2}, \tag{14}$$
$$n_i = q_i n_{i-1} + n_{i-2}, \tag{15}$$

indtil i er passende stor, eller der ikke er flere cifre q_i. F.eks. for kædebrøken $[3;4,12,4]$ beregnes,

$$\frac{t_0}{n_0} = \frac{q_0 t_{-1} + t_{-2}}{q_0 n_{-1} + n_{-2}} = \frac{3 \cdot 1 + 0}{3 \cdot 0 + 1} = \frac{3}{1} = 3, \tag{16}$$

$$\frac{t_1}{n_1} = \frac{q_1 t_0 + t_{-1}}{q_1 n_0 + n_{-1}} = \frac{4 \cdot 3 + 1}{4 \cdot 1 + 0} = \frac{13}{4} = 3.25, \tag{17}$$

$$\frac{t_2}{n_2} = \frac{q_2 t_1 + t_0}{q_2 n_1 + n_0} = \frac{12 \cdot 13 + 3}{12 \cdot 4 + 1} = \frac{159}{49} = 3.244897959\ldots, \tag{18}$$

$$\frac{t_3}{n_3} = \frac{q_3 t_2 + t_1}{q_3 n_2 + n_1} = \frac{4 \cdot 159 + 13}{4 \cdot 49 + 4} = \frac{649}{200} = 3.245. \tag{19}$$

Altså kan kædebrøkken $[3;4,12,4]$ approximeres som heltalsbrøkkerne $3/1$, $13/4$, $159/49$ og $649/200$ med gradvist stigende nøjagtighed.

6g0 Skriv en rekursiv funktion

```
cfrac2float : lst:int list -> float
```

som tager en liste af heltal som kædebrøk og udregner det tilsvarende reelle tal.

6g1 Skriv en rekursiv funktion

```
float2cfrac : x:float -> int list
```

som tager et reelt tal og udregner dens repræsentation som kædebrøk.

6g2 Skriv en rekursiv funktion

```
frac2cfrac : t:int -> n:int -> int list
```

som tager tæller og nævner i brøken t/n og udregner dens repræsentation som kædebrøk udelukkende ved brug af heltalstyper.

6g3 Skriv en rekursiv funktion

```
cfrac2frac : lst:int list -> i:int -> int * int
```

som tager en kædebrøk og et index og returnerer t_i/n_i approximationen som tuplen `(ti, ni)`.

6g4 Saml alle ovenstående funktioner i et bibliotek bestående af dets interface og implementationsfil (`continuedFraction.fsi continuedFraction.fs`), og skriv applikation som udfører en white- og black-box test af funktionerne.

Afleveringen skal bestå af

- en zip-fil, der hedder `6g_<navn>.zip` (f.eks. `6g_jon.zip`)

Zip-filen `6g_<navn>.zip` skal indeholde en og kun en mappe `6g_<navn>`. I den mappe skal der ligge en `src` mappe og filen `README.txt`. I `src` skal der ligge følgende og kun følgende filer: `continuedFraction.fsi, continuedFraction.fs` og `6g4.fsx` svarende til de relevante delopgaver. 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. Udover selve koden skal besvarelser indtastes som kommentarer i de fsx-filer, de hører til. Filen `README.txt` skal ganske kort beskrive, hvordan koden oversættes og køres.

God fornøjelse.