Introduktion til programmering, ugeseddel 6

Version 1.0

3. oktober 2014

Uge 6 drejer sig om *ind- og udlæsning* (på engelsk *input/output*, ofte forkortet I/O) af tekst samt *halerekursion*. Data til og fra vores programmer er hidtil optrådt direkte i afviklingsvinduet for Mosml; vi skal nu se, hvordan man fra et program kan komme i forbindelse med computerens filsystem. Begrebet "halerekursion" dækker over en bestemt måde at definere funktioner på, således at det rekursive kald til funktionen selv er det *sidste*, som sker i funktionskroppen. Halerekursivt definerede funktioner er ofte (men ikke altid) mere effektive end funktioner, som ikke er.

Ugens opgavetema drejer sig om funktioner til udfyldelse af sudoku-puslespil med anvendelse af filer. Opgaverne træner dels ind- og udlæsning, dels er de oplagte til repetition af højereordensfunktioner og rekursion i almindelighed.

Plan for ugen

Mandag

Input/Output, interaktive programmer og bivirkninger (også kaldet sideeffekter).

Pensum: HR: kap. 14 og kap. 15

Tirsdag

Mere om Input/Output, halerekursion, *records* og *record*-typer.

Pensum: HR: 3.4-3.6 og kap. 17 (samt evt. IP-2: 7.7)

Fredag

Repetition af ugens pensum (v. Torben Mogensen)

1 Opgavetema

Temaet for ugens opgaver er at programmere et lille sudoku-spil.

Sudoku er et puslespil, som er blevet opfundet uafhængigt flere gange; den tidligste "ægte" version af sudoku synes at kunne spores tilbage til det franske dagblad *Le Siècle* i 1892. Den moderne udgave har fundet vej til adskillige aviser på verdensplan i løbet af de sidste 7 år.

Vi betragter her kun den basale variant, som spilles på en matrix af 81 små felter, arrangeret i 9 rækker og 9 søjler. Matricen er desuden inddelt i 9 "bokse" eller "regioner", hver med 3 gange 3 felter.

Nogle af felterne er udfyldt på forhånd, og puslespillet går ud på at udfylde de resterende felter på en sådan måde, at hver af de 9 rækker, hver af de 9 søjler og hver af de 9 regioner kommer til at indeholde en permutation af symbolerne fra et forelagt alfabet af størrelse 9; vi vælger her (som man plejer at se det) alfabetet bestående af cifrene fra 1 til 9.

Her er en lovlig starttilstand for et spil sudoku:

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Og følgende er en vindende tilstand (en "løsning") af ovenstående:

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Nummerering Lad os nummerere rækkerne 0, 1, ..., 8 ovenfra og ned og søjlerne 0, 1, ..., 8 fra venstre mod højre. Også regionerne vil vi nummerere 0, 1, ..., 8, i "normal læseretning" (for vestlige sprog). Sammenhængen mellem rækkenummer r, søjlenummer s og regionsnummer q kunne så udtrykkes i følgende formel: Feltet i række nummer r og søjle nummer s vil ligge i region nummer

$$q = r \texttt{ div 3 * 3 + } s \texttt{ div 3}$$

Omvendt vil region nummer q bestå af de felter,
som både ligger i en af rækkerne q `div 3 * 3`, q `div 3 * 3 + 1` eller q `div 3 * 3 + 2`
og en af søjlerne q `mod 3 * 3`, q `mod 3 * 3 + 1` eller q `mod 3 * 3 + 2`.

Vi vil lave et simpelt SML-program, som kan indlæse en starttilstand for et sudokuspil, som kan præsentere den nuværende tilstand visuelt for brugeren, og som tillader en bruger at udfylde felter, der ikke allerede er fyldt ud.

Filformat En starttilstand er givet ved en fil, hvis indhold *altid* antages at se ud som følger:

- Der er mindst 90 tegn i filen (der kan være flere, men vi er kun interesseret i de 90 første)
- De første 90 tegn i filen er delt op i 9 grupper, hver bestående af 10 tegn: Først 9 tegn, som er et blandt `#"1"`, ..., `#"9"`, `#"*"`, og til sidst tegnet `#"\n"`.

F.eks. er nedenstående (NB! Bliv ej bange, hvis det er svært at læse) indholdet i en fil, som indeholder starttilstanden for ovenstående sudoku:

```
53**7****\n6**195***\n*98****6*\n8***6***3\n4**8*3**1\n7***2***6\n*6****28*\n***419**5\n****8**79\n
```

Hvis vi fortolker tegnet `#"\n"` som "ny linje", bliver ovenstående lettere at læse:

```
53**7****
6**195***
*98****6*
8***6***3
4**8*3**1
7***2***6
*6****28*
***419**5
****8**79
```

I hele ugesedlen vil vi skrive indholdet af filer som ovenfor, idet tegnet `#"\n"` vil blive fortolket som "ny linje".

Repræsentation i SML Principielt burde et sudoku-spil repræsenteres ved en passende datatype i ML. For at holde jeres opgavebesvarelser relativt begrænsede i deres syntaks, vil vi dog i stedet her bruge "rå" lister til repræsentationen.

En given tilstand af et sudoku-spil vil blive repræsenteret som en liste af ni lister, som alle indeholder ni tegn—altså som en speciel værdi af type `char list list`.

Det gennemgående eksempel ovenfor vil for eksempel blive repræsenteret som følgende liste:

```
[[#"5", #"3", #"*", #"*", #"7", #"*", #"*", #"*", #"*"],
 [#"6", #"*", #"*", #"1", #"9", #"5", #"*", #"*", #"*"],
 [#"*", #"9", #"8", #"*", #"*", #"*", #"*", #"6", #"*"],
 [#"8", #"*", #"*", #"*", #"6", #"*", #"*", #"*", #"3"],
 [#"4", #"*", #"*", #"8", #"*", #"3", #"*", #"*", #"1"],
 [#"7", #"*", #"*", #"*", #"2", #"*", #"*", #"*", #"6"],
 [#"*", #"6", #"*", #"*", #"*", #"*", #"2", #"8", #"*"],
 [#"*", #"*", #"*", #"4", #"1", #"9", #"*", #"*", #"5"],
 [#"*", #"*", #"*", #"*", #"8", #"*", #"*", #"7", #"9"]]
```

Ændring af et sudoku-spils tilstand Vi kan opfatte et felt på et sudokubræt som givet ved to koordinater: *rækken* r og *søjlen* s, som vi kan skrive som et koordinatpar (r, s). (Vi vedtog ovenfor at tælle rækker og søjler fra øverste venstre hjørne og at bruge numrene $0, 1, \ldots, 8$.)

Hvis vi for eksempel ønsker at skrive et 7-tal i det tomme felt på koordinat $(r, s) = (1, 2)$ i vores eksempelsudoku, får vi altså nedenstående (hvor vi har *kursiveret* 7-tallet):

5	3			7				
6		*7*	1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Bemærk, at i ML-repræsentationen af sudoku-tilstanden som en liste af lister svarer ovenstående indsættelse til, at man i liste nummer 1 på plads nummer 2 (med 0-baseret nummerering) erstatter tegnet `#"*"` med tegnet `#"7"`, således at vi får:

```
[[#"5", #"3", #"*", #"*", #"7", #"*", #"*", #"*", #"*"],
 [#"6", #"*", #"7", #"1", #"9", #"5", #"*", #"*", #"*"],
 [#"*", #"9", #"8", #"*", #"*", #"*", #"*", #"6", #"*"],
 [#"8", #"*", #"*", #"*", #"6", #"*", #"*", #"*", #"3"],
 [#"4", #"*", #"*", #"8", #"*", #"3", #"*", #"*", #"1"],
 [#"7", #"*", #"*", #"*", #"2", #"*", #"*", #"*", #"6"],
 [#"*", #"6", #"*", #"*", #"*", #"*", #"2", #"8", #"*"],
 [#"*", #"*", #"*", #"4", #"1", #"9", #"*", #"*", #"5"],
 [#"*", #"*", #"*", #"*", #"8", #"*", #"*", #"7", #"9"]]
```

Så: *At skrive et tal på koordinat (r, s) i sudokuen svarer til at erstatte et tegn i liste r, plads s med et andet tegn.*

1.1 Filers adresser

Som bekendt er en computers kataloger (også kaldet "mapper", "foldere" eller "directories") og filer ordnet i en træstruktur, og når man arbejder med maskinen, refererer man til mapper og filer ud fra et bestemt punkt ("arbejdskataloget") i denne struktur. Adressen på (referencen til) et katalog eller en fil kaldes også for katalogets eller filens *sti*.

Stier, der ikke indledes med en skråstreg, opfattes *relativt til arbejdskataloget* og kaldes også for *relative stier*. Refererer man for eksempel til `navn1`, betyder det et katalog eller en fil direkte under arbejdskataloget, og `navn1/navn2` betyder filen eller kataloget `navn2` indeholdt i kataloget `navn1` indeholdt i arbejdskataloget. (Under DOS og Windows skal skråstregen vende den anden vej, men når Moscow ML fortolker en tekst som en sti, sørges der for, at skilletegnet bliver det rigtige!)

Stier kan også være *absolutte*, hvilket vil sige, at de indledes med en skråstreg og bevæger sig ned gennem katalogsystemet fra katalogtræets *rod*. Moscow ML's fortolker kunne for eksempel have den absolutte sti `/Users/nina/mosml/bin/mosml`, og ens katalog med kursusmateriale til IP kunne have den absolutte sti `/Users/nina/Documents/studium/ip/`. Under Windows skal stier startes med et drevbogstav og et kolon, eksempelvis `c:`.

I ordrer til operativsystemet kan man bruge to nyttige "forkortelser" under arbejde med stier: Et enkelt punktum (`.`) betyder det aktuelle katalog, og to punktummer (`..`) angiver kataloget et niveau *opad* i katalogtræet. I fortsættelse af det ovennævnte eksempel ville man altså fra sit katalog med IP-kursusmateriale kunne aktivere Moscow ML med ordren `../../../mosml/bin/mosml`

Fra Moscow ML er skrivemåden med punktummerne desværre ikke til rådighed, så når man fra et program skal referere til en fil, *må det normalt ske via filens absolutte sti!* (Indledes stien ikke med en skråstreg, sådan at man altså bruger en *relativ* sti, vil henvisningen blive fortolket *relativt til det katalog, som Moscow ML-fortolkeren selv ligger i!*)

Oplysningerne her skulle være tilstrækkelige til løsning af de opgaver, man møder på kurset, men der kan være grund til at gøre opmærsom på, at biblioteket `OS` indeholder to strukturer `OS.FileSys` og `OS.Path` med en lang række hjælpefunktioner til håndtering af kataloger og stier. (I Mosml er disse strukturer også tilgængelige som de selvstændige biblioteker `FileSys` og `Path`.)

I øvrigt henvises til hele standardbiblioteket for SML, det såkaldte *Standard ML Basis Library* med adressen `http://mosml.org/mosmllib/`

2 Mandagsopgaver

6M1 Benyt `stdOut` til at udskrive teksten "Denne tekst indeholder ikke et linjeskift"på skærmen.

Gentag ovenstående, men sørg denne gang for, at der kommer et linjeskift efter teksten.

6M2 Definer en funktion `lssuffix : string -> string`, som føjer et linjeskift til en tekst. Kaldet `lssuffix "Simonsen"` skal for eksempel returnere teksten `"Simonsen\n"`.

Definer dernæst en funktion `lssuffixout : string -> unit`, som føjer et linjeskift til en tekst og dernæst skriver teksten ud på skærmen.

Hvorfor er returtypen for `lssuffixout` den særlige type `unit`?

6M3 Indtast i et SML-afviklingsvindue følgende linjer:

```
val outstrm = TextIO.openOut "udgangsfil.txt";
TextIO.output (outstrm, "Dette er en prøve!\n");
TextIO.flushOut outstrm;
```

og åbn derefter den dannede fil i et passende tekstbehandlingsprogram (for eksempel GEdit, Notesblok, TextEdit eller Emacs) for at kontrollere, at teksten faktisk er blevet skrevet i den. (Se afsnit 1.1 ovenfor om filers adresser.)

Afhængigt af, hvordan tegntabellerne er sat op på maskinen, kan det være nødvendigt med særbehandling af de danske bogstaver, for eksempel med

```
TextIO.output (outstrm, "Dette er en pr\248ve!\n");
```

Prøv at skrive noget mere ud til filen, og bemærk, hvordan tekst først kommer til syne i tekstbehandlingssystemets vindue efter `TextIO.flushOut`. Måske skal tekstvinduet genindlæses (eller lukkes og genåbnes). Husk til sidst at skrive

```
TextIO.closeOut outstrm;
```

6M4 Opret på din maskine med et passende tekstbehandlingsprogram en fil med navnet `indgangsfil.txt`. Tast 3–5 linjers tekst ind, og gem filen (som rå eller "flad" tekst, eng. *plaintext*).

Opret nu en indgangsstrøm til filen med

```
val instrm = TextIO.openIn "indgangsfil.txt";
```

(Se afsnit 1.1 ovenfor om filers adresser.)

Forsøg nu at læse første linje i filen med indgangsstrømmen.

Fortsæt med kald til `TextIO.inputLine`, til alle linjer i filen er læst. Husk til sidst at skrive

```
TextIO.closeIn instrm;
```

3 Tirsdagsopgaver

Du forventes at have forberedt løsninger til nedenstående til første øvelsestime om tirsdagen.

Brug gerne højereordensfunktioner, hvis opportunt og koncist.

6T1 Definer en funktion `listLines : string -> string list`, som danner listen bestående af hver linje i en fil (uden linjernes afsluttende linjeskifttegn). Hvis `filnavn` er en værdi af type `string`, som repræsenterer et filnavn, da skal kaldet `listLines filnavn` altså returnere en liste af de linjer, som findes i `filnavn`.

Hvis `filnavn` for eksempel har følgende indhold:

```
Der er en Trolddom på din Læbe
Der er en Afgrund i dit Blik
Der er i Lyden af din Stemme
En Drøms Æteriske Musik
```

Da skal kaldet `listLines filnavn` returnere værdien

```
["Der er en Trolddom på din Læbe", "Der er en Afgrund i dit Blik",
 "Der er i Lyden af din Stemme", "En Drøms Æteriske Musik"]
```

Pas på! Det sidste tegn i filen behøver ikke at være `#"\n"` (filen kan stoppe brat "midt" på en linje).

6T2 Definer en funktion `listChars : string -> char list list`, som for hver linje i filen danner listen af tegn, som findes i hver linje. Kaldet `listChars filnavn` (hvor `filnavn` er givet som ovenfor) skal for eksempel returnere følgende:

```
[[#"D", #"e", #"r", #, #"e", #"r", #, #"e", #"n", ..., #"L", #"æ", #"b", #"e"],
 [#"D", #"e", #"r", #, ..., #"B", #"l", #"i", #"k"],
 ...
 [#"E", #"n", #, #"D", ..., #"s", #"i", #"k"]]
```

6T3 Skriv et program, som tillader brugeren at indtaste et tegn ad gangen (vha. `stdIn`). Hvis brugeren trykker på tasten Q (i ML-notation: indtaster tegnet `#"q"`) (efterfulgt af linjeskift), standser programmet. Hvis brugeren trykker på en hvilken som helst anden grafisk tast (efterfulgt af linjeskift), beder programmet om et nyt tegn (ved at skrive en passende besked til `stdOut`).

Vink I: Hvis der ikke skrives noget ud på skærmen, har I måske glemt at tømme en buffer.

Vink II: Husk, at semikolonnet `;` tillader at udføre en handling (med bivirkning) efter en anden.

Vink III: Hvis programmet skal spørge om noget mere end en gang, kunne man måske bruge den mest fundamentale teknik på dette kursus ...

4 Fredagsopgaver

Du forventes at have forberedt løsninger til nedenstående til første øvelsestime om fredagen.

6F1 HR 17.3

6F2 HR 17.2 (løs evt. HR 1.4 på normal vis før du går i gang)

5 Gruppeaflevering

Bemærk, at de fire første opgaver er obligatoriske. Opgaven afleveres i Absalon. Der afleveres en fil pr. gruppe, men den skal angive alle deltageres fulde navne i kommentarlinjer øverst i filen. Filens navn skal være af formen `6G-`*`initialer`*`.sml`, hvor initialer er erstattet af gruppemedlemmernes initialer. Hvis f.eks. Bill Gates, Linus Torvalds, Steve Jobs og Gabe Logan Newell afleverer en opgave sammen, skal filen hedde `6G-BG-LT-SJ-GLN.sml`. Brug gruppeafleveringsfunktionen i Absalon.

Gruppeopgaven giver op til 2 point, som tæller til de 20 point, der kræves for eksamensdeltagelse. Genaflevering kan hæve pointtallet fra første aflevering med højest 1 point, så sørg for at gøre jeres bedste allerede i første aflevering.

Gruppeopgaverne går ud på at programmere et lille system, som kan indlæse en sudoku-starttilstand og derefter gentagent bede brugeren om at indtaste koordinater til at opdatere spillets tilstand.

Læs beskrivelsen af ugens opgavetema, før i går i gang.

Brug gerne halerekursive funktioner og højereordensfunktioner, hvor I finder det anvendeligt.

6G1 Definer en funktion `readSudoku : string -> char list list`, således at hvis `filnavn` er navnet på en fil, som indeholder en sudoku-tilstand, da returnerer et kald `readSudoku filnavn` den liste af lister af tegn, som repræsenterer sudoku-tilstanden (læs beskrivelsen af ugens opgavetema igen, hvis dette virker forvirrende).

Vink: Tirsdagsøvelserne indeholder essentielt en løsning af denne opgave.

6G2 Definer en funktion `showSudoku : char list list -> unit`, som skriver en sudoku-tilstand ud på skærmen.

Vink: En tilstand er en liste af lister. Skriv hver liste ud, en ad gangen. Husk linjeskift.

6G3 Definer en funktion `modifySudoku : char list list -> int * int * char -> char list list`, således at hvis `r` og `s` begge er værdier (i intervallet mellem `0` og `8`) af typen `int`, og `xs` repræsenterer en sudoku-tilstand, da returnerer kaldet `modifySudoku xs (r,s,c)` en ny sudokutilstand `ys`, som er resultatet af at erstatte tegnet i række `r`, plads `s` i `xs` med tegnet `c`.

I eksemplet fra afsnittet "Ændring af sudokuspillets tilstand" ovenfor skal man altså for eksempel udføre kaldet `modifySudoku xs (1,2,#"7")`.

Husk: Funktionen skal have den angivne type!

6G4 (Denne opgave kan tjene som inspiration for, eller være direkte brugbar i, ugens individuelle opgaver).

Definer en funktion `regionList : char list list -> int -> char list`, så kaldet `regionList xs n` returnerer listen af tegn, som forekommer i den `n`'te region af `xs`, hvor `xs` repræsenterer en sudokutilstand. I må selv om, hvilken rækkefølge tegnene angives i.

Hvis `xs` er listen af lister af tegn, som repræsenterer starttilstanden i det gennemgående eksempel, da kan kaldet `regionList xs 3` returnere listen `[#"8", #"*", #"*", #"4", #"*", #"*", #"7", #"*", #"*"]`.

6G5 (Valgfri — **men stærkt anbefalet** — ekstraopgave) Skriv et program, som tillader en bruger i dialog med skærmen at angive et filnavn, som indeholder en sudoku-starttilstand, og som dernæst bliver ved at spørge brugeren, hvilke koordinater, som der skal skrives på.

Brugeren skal kunne indtaste koordinater og ændringer i formatet

```
r s c
```

(afsluttet med linjeskift).

Når en ændring er foretaget, skal funktionen skrive den ny tilstand ud på skærmen, så brugeren kan se, hvad den nuværende tilstand er.

Funktionen skal dernæst tillade nye ændringer.

Der er ingen krav om, at programmet skal kunne afgøre, om spillet er færdigt, eller skal forhindre brugeren i at overskrive allerede skrevne værdier.

Vink: Læs eventuelt HR, kap. 16.

6G6 (Valgfri ekstraopgave) Udvid programmet fra opgave 6G5, så det ikke er muligt at overskrive talværdier fra spillets *start*tilstand (men til gengæld at overskrive tidligere værdier, som brugeren selv har angivet).

6 Individuel aflevering

Besvarelsen afleveres i Absalon som en fil med navnet `6I-navn.sml`, hvor *navn* er erstatttet med dit navn. Hvis du hedder Anders A. And, skal filnavnet f.eks. være `6I-Anders-A-And.sml`. Skriv også dit fulde navn som en kommentar i starten af filen.

Den individuelle opgave drejer sig om at implementere checks for, om et sudoku-spil er vundet.

Læs ugens opgavetema igen, hvis du er i tvivl om, hvad det vil sige, at et spil er vundet.

Brug gerne halerekursive funktioner og højereordensfunktioner, hvor du finder det anvendeligt.

6I1 Definer en funktion `sublist : ''a list -> ''a list -> bool`, således at et kald `sublist xs ys` returnerer `true` netop hvis alle elementer i listen `xs` også er elementer i listen `ys` (du kan i denne opgave antage, at `xs` ikke indeholder dubletter).

F.eks. skal `sublist [#"1", #"2"] [#"1", #"3"]` returnere `false`, mens kaldet `sublist [#"1", #"2"] [#"2", #"1"]` skal returnere `true`.

Vink: HR, afsnit 5.7. Hvis du har `member` til rådighed, skal du blot for hvert element i `xs` checke, at det også er med i `ys`.

6I2 Definer en funktion `column : int -> 'a list list -> 'a list`, således at et kald `column n xs` returnerer listen af de elementer, som (ved 0-baseret nummerering) står på plads `n` i lister fra `xs`.

F.eks. skal kaldet

```
column 1 [[#"3", #"6"], [#"6", #"2"], [#"5", #"9"]]
```

returnere listen

```
[#"6", #"2", #"9"].
```

Hvis en af listerne i `xs` har færre end `n+1` elementer, skal der rejses en passende undtagelse.

Vink I: Definer først (eller find i HR ...) en funktion, som returnerer det `n`'te element i en liste.

Vink II: Hvad sker der, hvis man tager funktionen fra foregående vink og kombinerer den passende med `map`?

6I3 Forklar *kort og præcist*, hvordan `sublist` kan bruges til at checke, om tallene 1–9 alle er blevet brugt netop en gang i hver række i (ML-repræsentationen af) et sudoku-spil

Forklar dernæst, hvordan `sublist` og `column` kan bruges til at checke, om tallene 1–9 alle er blevet brugt netop en gang i hver søjle i (ML-repræsentationen af) et sudoku-spil.

6I4 Definer en funktion `rowcolCheck : char list list -> bool`, som checker, om tallene 1–9 alle er blevet brugt netop en gang i hver række og hver søjle i (ML-repræsentationen af) et sudoku-spil (funktionen skal returnere `true`, hvis dette er tilfældet).

6I5 Definer til slut en funktion `checkSudoku : char list list -> bool`, som checker, om (ML-repræsentationen af) en sudukotilstand repræsenterer en vindende tilstand.

Vink I: Tre kriterier skal være opfyldt: Et for rækker, et for søjler, og et for regioner. Du har i opgave 6i4 løst problemet for rækker og søjler og mangler således kun at løse problemet for regioner. Måske kan en funktion fra ugens gruppeopgaver benyttes?

6I6 (Valgfri ekstraopgave) Indbyg checket fra foregående opgave i det interaktive sudoku-spil fra de valgfri gruppeopgaver. Stop spillet, når det er vundet (med en passende meddelelse til brugeren).

7 Valgfri ekstraopgaver

Hvis man har tid, kan nedenstående opgaver løses; instruktorernes hjælp til disse opgaver vil være begrænset, da studerende, som endnu ikke har løst de obligatoriske opgaver, har fortrinsret.

- HR: 15.1 eller (mere krævende) 15.3
- HR: 14.4 (bemærk at der mangler et `'a` argument i typerne for `fileFoldr` og `fileFoldl`)

Til slut: Skriv en sudoku*løser*, som givet en starttilstand automatisk finder frem til en løsning uden menneskelig indblanding. Teknikker fra kombinatorisk søgning (se f.eks. 8-dronninge-problemet) kan ret let overføres.

8 Ugens nød

Ugens nød er en opgave for særligt interesserede studerende, som enten har mere tid til rådighed end de 20 timer, som der forventes brugt på kurset, eller som har særlige forudsætninger. Det forventes *ikke*, at man som studerende kan lave ugens nød, medmindre man lægger en god portion ekstraarbejde.

Afleveringsfristen er den samme som for den individuelle opgave. Ved fredagsforelæsningen i uge 7 (dvs. ugen efter efterårsferien) kåres den bedste løsning, og vinderen får ud over hæderen og forelæserens respekt en lille præmie.

Dovne tvedelende træer

Når fænomener skal repræsenteres i en computer, kan man pakke data mere eller mindre tæt. Som regel fås en hurtig tilgangstid, hvis man er villig til at afsætte meget lager, mens lagerforbruget tilsvarende kan mindskes, hvis der må bruges længere beregningstid på at finde data frem.

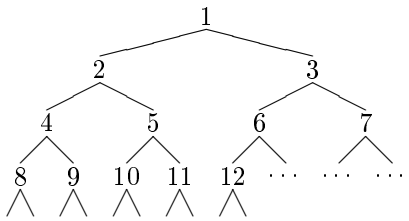
I yderste konsekvens kan man benytte en såkaldt "doven" repræsentation, hvor ingen data er lagret direkte, men først beregnes, når der er behov for dem. En sådan repræsentation gør det til gengæld muligt at håndtere datastrukturer, der i princippet er uendeligt store: Forud for hver anvendelse regner man sig frem til de dele, som skal bruges (og kun endelige dele skal bruges ad gangen).

Afsnit 9.11 i lærebogen af Hansen & Rischel omtaler ganske kort, hvordan SML, der ellers benytter "ivrig" evaluering, kan simulere doven evaluering ved at pakke et udtryk `e` ind som funktion `(fn _ => e)`. Emnet for ugens nød er dovne tvedelende træer (*lazy binary trees*) med følgende definition:

```
datatype 'elt infbtree = Node of (unit -> 'elt infbtree) * 'elt * (unit -> 'elt infbtree)
```

Som eksempel på en værdi af type `int infbtree` opstilles de positive hele tal i et uendeligt træ, idet hver knude n har venstre og højre undertræ med rod henholdsvis $2n$ og $2n+1$:

```
fun posinttree n = Node (fn _ => posinttree (2 * n),n,fn _ => posinttree (2 * n + 1))
val posints = posinttree 1
```



Figur 1: Det uendelige træ `posints`.

Figur 1 illustrerer `posints`.

Et kald af formen `bredde` k t vil "høste" de første k elementer af det uendelige tvedelende træ t "bredde først":

```
local fun bredde' 0 _ = []
        | bredde' k (Node (left,cargo,right) :: ts)
          = cargo :: bredde' (k - 1) (ts @ [left (), right ()])
in fun bredde k t = bredde' k [t] end;
bredde 10 posints;
val it = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] : int list
```

Opgave

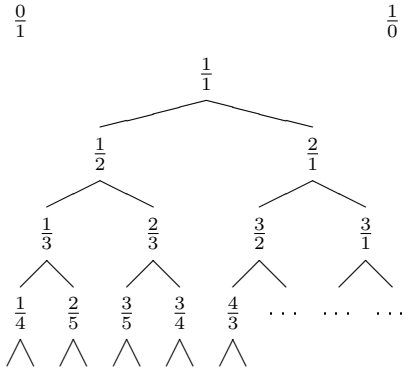
Selve nødden går ud på at konstruere en funktion `filtinord : ('elt -> bool) -> 'elt infbtree -> 'elt list` og en værdi `sternBrocot : qnum infbtree` som nærmere beskrevet i det følgende.

Et funktionskald af formen `filtinord` p t skal returnere en liste med knuderne fra t i "inorder", men dog sådan, at hvis en knude ikke opfylder p, skal hverken den eller nogen af dens underliggende knuder medtages i listen. (For at kaldet kan afsluttes, må man forudsætte, at der kun bliver endeligt mange knuder tilbage i listen.)

For eksempel skal der gælde:

```
filtinord (fn n => n mod 4 > 0 andalso n <= 20) posints;
val it = [2, 10, 5, 11, 1, 6, 13, 3, 14, 7, 15] : int list
```

Figur 2 viser det såkaldte Stern-Brocot-træ (efter Moritz Stern og Achille Brocot), som er et uendeligt tvedelende træ med alle de positive rationale tal.



Figur 2: Stern-Brocot-træet med alle positive rationale tal.

Træet konstrueres på følgende måde: Indholdet af en given plads findes ved, at man opsøger den nærmest forudgående brøk $\frac{a}{b}$ til venstre for pladsen og den nærmest forudgående brøk $\frac{c}{d}$ til højre for pladsen. På pladsen skrives i så fald $\frac{a+c}{b+d}$.

Betragt for eksempel de to pladser lige under $\frac{2}{3}$: De forudgående brøker for den venstre efterfølger er $\frac{1}{2}$ og $\frac{2}{3}$; den venstre efterfølger skal derfor være $\frac{1+2}{2+3} = \frac{3}{5}$. De forudgående brøkder for den højre efterfølger er $\frac{2}{3}$ og $\frac{1}{1}$, så på den plads skal der stå $\frac{2+1}{3+1} = \frac{3}{4}$.

Man kan vise (men at gøre dette indgår ikke i den stillede opgave), at

- Brøker konstrueret på den beskrevne måde bliver altid uforkortelige.
- Alle positive rationale tal kommer netop én gang med i træet.
- Tallene vil blive placeret efter størrelse (i forhold til "inorder"). Træet bliver med andre ord et "søgetræ" (jævnfør Example 8.1 i lærebogen af Hansen & Rischel).

Vælg en repræsentation `qnum` af de rationale tal (jævnfør lærebogens afsnit 4.2, 4.3 og 8.7.1), og konstruer Stern-Brocot-træet som en værdi `sternBrocot : qnum infbtree`.

Farey-rækken

Farey-rækken $\mathcal{F}_n$ af orden n er de uforkortelige brøker mellem 0 og 1, hvis nævner højst er n, ordnet efter størrelse. For eksempel gælder

$$\mathcal{F}_7 = \frac{0}{1}, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{2}{7}, \frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \frac{1}{2}, \frac{4}{7}, \frac{3}{5}, \frac{2}{3}, \frac{5}{7}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7}, \frac{1}{1}$$

Antag, at `enum : qnum -> int` og `denom : qnum -> int` i den repræsentation af rationale tal, man har valgt, finder henholdsvis tallets tæller og nævner, og at `nul : qnum` repræsenterer $\frac{0}{1}$. Da kan man som afprøvning af sine konstruktioner tjekke, at

```
nul :: filtinord (fn q => enum q <= denom q andalso denom q <= n) sternBrocot
```

beregner Farey-rækken af orden n. (Der er også andre (og nemmere) måder at beregne $\mathcal{F}_n$ på.)